İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

YAPAY SİNİR AĞLARI
VE
GEZGİN SATICI PROBLEMİNE UYGULANMALARI

YÜKSEK LİSANS TEZİ

Müh. Murat YILDIRIMHAN

Anabilim Dalı : Endüstri Mühendisliği
Programı : Endüstri Mühendisliği

OCAK 2003

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

YAPAY SİNİR AĞLARI
VE
GEZGİN SATICI PROBLEMİNE UYGULANMALARI

YÜKSEK LİSANS TEZİ

Müh. Murat YILDIRIMHAN

(507991068)

Tezin Enstitüye Verildiği Tarih : 24 Aralık 2002
Tezin Savunulduğu Tarih : 20 Ocak 2003

Tez Danışmanı : Prof. Dr. Füsun ÜLENGİN
Diğer Jüri Üyeleri : Doç. Dr. Alpaslan FIĞLALI
Doç. Dr. Kerem CIĞIZOĞLU

OCAK 2003

ÖNSÖZ

Bu tezin yazım sürecinde, bana en çok sorulan sorulardan biri de "Böyle bir konuda araştırma yapmakla ne elde edeceksin?" sorusuydu. Ben, bu tip farklı bilimsel yaklaşımlarla olaylara yaklaşmanın, insanın düşünce yapısını geliştireceğine ve onu farklılaştıracağına inanmaktayım. Bu nedenle, lisans öğrenimim sırasında ilgi duyduğum Kombinatoryal Optimizasyon problemlerine çözüm yaklaşımlarından biri olan Yapay Sinir Ağlarını akademik düzeyde ele almaya karar verdim. Bu kararın alınmasında, lise yıllarında Fen bilimlerine ve özellikle de Biyolojiye duyduğum ilginin etkisi de bulunmaktadır.

Bu çalışmada ele alınan konular, diğer çalışma alanlarından farklı olarak, konunun özünü teşkil eden, sonuçların türetildiği temel teoremlere sahip değildir. Tüm bu konular sadece birbiriyle ilişkisiz, özelleşmiş birtakım teknik ve yaklaşımlar topluluğu olarak görülebilir. Ancak, ilgili birçok çalışmanın incelenmesi ile birlikte Yapay Sinir Ağlarını ve Gezgin Satıcı Problemine uygulanmalarını, sistematik bir biçimde sunulabilecek tutarlı bir konu haline getiren temel prensipleri ortaya çıkarmak mümkündür. Oldukça yoğun bir çalışma gerektiren böyle bir amacı gerçekleştirmek adına ve bir adım daha ileri giderek, konunun gerçek hayatta karşılaşılan optimizasyon problemlerine uygulanmasına yönelik yapılabilecek başka çalışmalarda elinizdeki bu çalışmanın önemli bir rol üstleneceğini düşünüyorum. Bu çalışmanın yazımı sürecinde yardımlarını ve desteklerini esirgemeyen başta tez danışmanım Prof Dr. Füsun ÜLENGİN olmak üzere herkese teşekkür ederim.

Ocak, 2003 **Murat YILDIRIMHAN**

İÇİNDEKİLER

TABLO LİSTESİ

Sayfa No

ŞEKİL LİSTESİ

Sayfa No

Sayfa No

YAPAY SİNİR AĞLARININ
GEZGİN SATICI PROBLEMİNE UYGULANMALARI

ÖZET

Bu çalışma, adından da anlaşılabileceği şekilde, iki temel kavram üzerinde durmaktadır: Yapay Sinir Ağları ve Gezgin Satıcı Problemi.

Gezgin Satıcı Problemi, kombinatoryal optimizasyonun en popüler problemlerinden bir tanesidir ve optimizasyon alanında geliştirilen yeni yaklaşımların kendilerini ispatlamaya çalıştıkları ilk çalışma alanlarından biri olarak haklı bir üne sahiptir. Gezgin Satıcı Problemi, kombinatoryal optimizasyon problemlerinin doğası gereği, "Np-complete" bir optimizasyon problemidir. Bu tip problemlerin çözümünde, optimum sonuca ulaşmada karşılaşılan zorluklardan dolayı, hızlı bir biçimde optimum çözüme yakın çözümler üreten sezgisel yöntemler sık sık kullanılmaktadır. Yapay Sinir Ağları işte bu noktada, optimizasyon problemlerinin çözümüne yönelik geliştirilen yeni sezgisel yaklaşımlardan biri olarak karşımıza çıkmaktadır.

Yapay Sinir Ağları, insan vücudundaki Merkezi Sinir Sisteminin yapısına ve işleyişine ait temel ilkelerin modellenmesi niyetiyle ortaya çıkmış bir kavramdır. Paralel bir düzende sıralanmış işlemciler sistemi şeklinde de adlandırabileceğimiz Yapay Sinir Ağları ile, Merkezi Sinir Sisteminden esinlenilerek, bilişsel ve duyumsal görevlerin yerine getirilmesinde seri işlemcilere yani günümüz bilgisayarlarına nazaran daha tatmin edici sonuçların daha kolay bir biçimde elde edilmesi hedeflenmektedir.

Yapay Sinir Ağları, 1980'li yılların başından itibaren, kombinatoryal optimizasyon teorisinde zor olarak adlandırılan bazı problemlerin çözümüne yönelik alternatif bir yaklaşım olarak ele alınmaya başlanmıştır. 15 yılı aşan bu süreçte yapılan araştırmalar neticesinde geliştirilen sinir ağı yaklaşımlarının büyük bir çoğunluğu iki temel başlık altında toplanabilir: (1) Hopfield ağ yapıları ve (2) Öz-düzenlemeli haritalar.

Sinir ağı yaklaşımlarının, optimizasyon problemlerinde çözüm öncesi gerek modelleme ve programlama süreçlerinde gerekse veri toplama ve set oluşturma süreçlerinde getirdikleri katkılar göz ardı edilemez. Tüm bunların yanında, Yapay Sinir Ağlarının uygulama potansiyelinin hala çok uzağında bulunulduğu da düşünüldüğünde, Yapay Sinir Ağları ilgi gösterilmeyi hak eden bir çalışma alanı olarak karşımıza çıkmaktadır.

ARTIFICIAL NEURAL NETWORKS
FOR SOLVING THE TRAVELLING SALESMAN PROBLEM

SUMMARY

This study, as the name implies, brings together two principal concepts: Artificial Neural Networks and the Travelling Salesman Problem.

The Travelling Salesman Problem is one of the most popular problems in combinatorial optimization and endowed with a right reputation as a first study range that new ideas about optimization try to proved themselves. The Travelling Salesman Problem, as natural for combinatorial optimization problems, is an Np-complete optimization problem. Because of the difficulties in finding optimal solutions for those kind of problems, heuristics are generally used for finding rapid and nearby optimal solutions. At this point, Artificial Neural Networks play a part of the new heuristics developed for finding good solutions to those kind of optimization problems.

Artificial Neural Networks, also called Parallel Distributed Processing Systems, are intended for modeling the organizational principles of the Central Nervous System, with the hope that the biologically inspired computing capabilities of the Artificial Neural Network will allow the cognitive and sensory tasks to be performed more easily and more satisfactorily than with conventional serial processors.

Artificial Neural Networks have been accepted an alternative approach for finding good solutions to some Np-complete problems in combinatorial optimization theory since the beginning of 1980s. Vast majority of neural network approaches, developed during this over-15-year period, can be collected under two principal headings: (1) Hopfield networks and (2) Kohonen's Self-organizing Maps.

The contributions of neural network approaches in either modelling and programming stages or data collecting and information gathering stages of solution period of combinatorial optimization problems can not be overpassed. Also, the practical capabilities of Artificial Neural Networks are still not to be realized. Once all the facts written above are thought about, then it is easy to understand how a deserve-attention study range Artificial Neural Networks is.

GİRİŞ

Konu başlığındaki Yapay Sinir Ağları ve Gezgin Satıcı Problemi kavramları, aslında, elinizdeki çalışmanın kapsamı hakkında yeterli ipucunu içermektedir. Söz konusu iki kavramın popülerlik kazanmasında ve gelişmesinde, aynı ortak nedenlerin yattığı rahatlıkla söylenebilir: Basit yapıları, kolay uygulanabilirlikleri ve (belki de) ilgi çeken, merak uyandıran isimleri. Bu kavramları bir araya getiren nedenler ise; bir taraftan Gezgin Satıcı Probleminin kombinatoryal optimizasyon alanında geliştirilen yeni yaklaşımların kendilerini ispatlamaya çalıştıkları ilk çalışma alanlarından biri olması; diğer taraftan ise Yapay Sinir Ağlarının optimizasyon problemlerinin çözümüne yönelik geliştirilen yeni yaklaşımlardan biri olmasıdır.

Elinizdeki çalışma Yapay Sinir Ağlarını, optimizasyon problemlerinin çözümüne yönelik olarak kullanılabilen yeni sezgisel yaklaşımlardan biri olarak ele almakta ve söz konusu yaklaşımın, kendini ispat etmesi adına, Gezgin Satıcı Probleminin çözümüne yönelik olarak öne sürdüğü teknikleri bir araya toplamaktadır. Böylelikle, optimal çözüm üretmede sıkıntılar yaşanan kombinatoryal optimizasyon problemlerine iyi çözümler üretmek adına kullanılabilecek Yapay Sinir Ağlarının sahip olduğu özellikler ortaya konmaya çalışılmaktadır.

Bu içerikte tasarlanan çalışma, tahmin edilebileceği gibi, temel olarak iki ayrı bölümden oluşmaktadır. İlk bölümde, Yapay Sinir Ağları kavramı tek başına ele alınarak konuyu kavramaya yardımcı olacak temel noktalar açıklanmaya çalışılılmıştır. Bu anlamda bu bölüm, Yapay Sinir Ağlarının yalnızca optimizasyona yönelik uygulamalarında değil diğer tüm uygulamalarında da yararlanabilecek bir bilgi bütünlüğüne sahiptir.

Dört ayrı başlık altında toplanan bu ilk bölümün ardından, Gezgin Satıcı Probleminin temel özellikleri ve söz konusu probleme üretilen çözüm yaklaşımları kısaca ele alınarak Yapay Sinir Ağlarının Gezgin Satıcı Probleminin çözümüne yönelik olarak öne sürdüğü teknikler iki ayrı bölüm halinde sıralanmaktadır. Burada sıralanan her bir teknik için, söz konusu tekniğin Gezgin Satıcı Problemine yaklaşımı ve bu yaklaşımda yaşanan geliştirmeler tarihsel bir sırada verilmekte ve ardından takip ettikleri algoritmalar bilgisayar uygulamalarıyla elde edilen çıktılar üzerinden açıklanmaktadır.

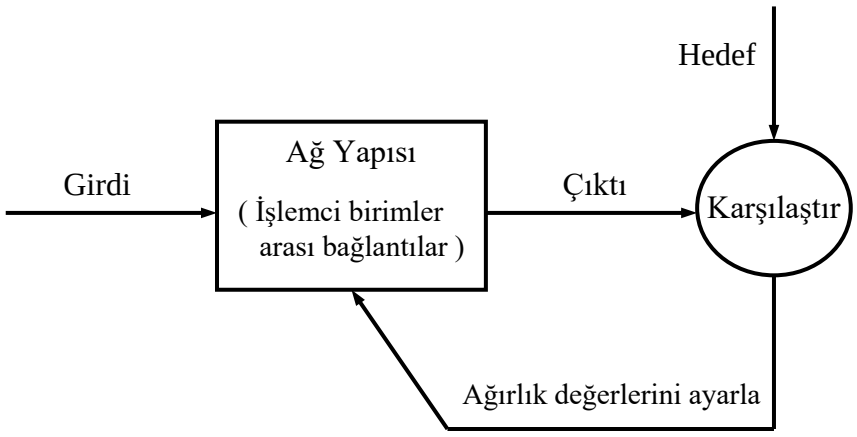
Çalışmanın sonunda ise "Sonuç ve Öneriler" başlığı altında, yapılan genel bir değerlendirmenin ardından ilave zaman ve destek ile çalışmanın nasıl geliştirilebileceği vurgulanarak Yapay Sinir Ağlarının, kombinatoryal optimizasyon alanındaki diğer yaklaşımlar arasındaki konumu ve Gezgin Satıcı Probleminin çözümüne yönelik ortaya koyduğu katkılar, karşılaştırmalı olarak ve toplu bir biçimde ortaya konmaktadır.

1. YAPAY SİNİR AĞLARINA GİRİŞ

1.1 Yapay Sinir Ağları Kavramı

Yapay Sinir Ağları, paralel bir düzende sıralanmış basit işlemci elemanlardan oluşurlar. Biyolojik sinir sistemlerinden esinlenilen bu basit işlemci elemanlardan her biri, kendi özel girdi setini işler ve çıktısını bağlantı kurmuş olduğu kendi özel işlemci setine iletir. Bir sinir ağı yapısı içinde iletilen ve işlem gören bilgiden yani ağ yapısının fonksiyonundan, büyük oranda, işlemci elemanların aralarında kurdukları bağlantılar sorumludur. Sinir ağları eğitilerek söz konusu bağlantıların ağırlık değerleri uygun şekilde ayarlanabilir ve herhangi bir ağ yapısının belirli bir fonksiyonu yerine getirmesi sağlanabilir.

Yapay Sinir Ağları, genellikle belirli bir girdinin hedeflenen bir çıktıya ulaşmasını sağlayacak şekilde eğitilirler. Bu durum aşağıdaki şekilde ele alınmaktadır:



Şekil 1.1 Yapay Sinir Ağlarında Gözetimli Öğrenme (*Demuth ve Beale, 1997; s.23*).

Ağ yapısında, belirli bir girdiye ait elde edilen çıktı ile hedeflenen çıktı değerleri karşılaştırılarak işlemci elemanlar arasındaki bağlantı ağırlık değerleri yeniden düzenlenir. Bu işlem, bir çok girdi-çıktı çifti ile göz yumulabilir bir fark elde edilene kadar tekrarlanır.

1.2 Yapay Sinir Ağlarının Yetenekleri

Yapay Sinir Ağları ile ele alınan bu yeni hesaplama şekli, öğrenme sayesinde girdi-çıktı çifti arasındaki ilişkinin gelişimine bağlıdır. Bu yeni hesaplama şekli, günümüzde, seri dijital işlemcilerin erişim gücünün ötesindeki karmaşık fonksiyonları yerine getirmek ve pratik alanlarda zor olarak adlandırılan problemlere çözümler üretmek amacıyla kullanılmaktadır. Bu durum, mevcut teknoloji ile gerekli işlemsel hız ve veri depolama özelliklerinin gerçekleştirilememesi yanında, özellikle bu problemlerin doğasında var olan tanımlama ve modelleme güçlüklerinden kaynaklanmaktadır.

Sinir ağları araştırmaları elli yılı aşan bir tarihe sahip olmasına rağmen, pratik alanlardaki başarılı uygulamaları son onbeş yıl içinde gerçekleştirilebilmiştir. Yapay Sinir Ağlarının yaygın olarak kullanılan bazı yetenekleri aşağıdaki şekilde sayılabilir:

- Sınıflandırma ve Veri Tanımlama. Yapay Sinir Ağları çok çeşitli örnekler arasından, tanımlanan modellere ait örnekleri ayırt edecek ve sınıflandıracak şekilde eğitilebilirler. Bu sayede örneğin, kanser hastalığı teşhis ve tedavisinde iyi ve köyü niyetli hücrelerin saptanması amacıyla ya da bir radar sisteminde gelen sinyallerin hangi cisimlere ait olduğunun belirlenmesi amacıyla sinir ağları kullanılabilmektedir. Aynı şekilde, Yapay Sinir Ağları ses ve görüntüye dayalı kontrol sistemlerinde, girdilerin ağ yapısına öğretilen orijinal modele uyup uymadığının belirlenmesi amacıyla da kullanılabilmektedir.

- Değer Tahmininde Bulunma. Yapay Sinir Ağları bir değişkenin, kendisine ya da ilgili başka değişkenlere ait tarihsel değerlerinden hareketle, geleceğe ilişkin değerinin tahmin edilmesi problemlerinde de kullanım şansı bulmaktadır. Bu bağlamda, ekonomi ve meteoroloji bilimlerine ilişkin parametrelerin değer tahmini ilk olarak akla gelen örneklerdir.

- Gürültülü Veri Setlerini İndirgeme. Yapay Sinir Ağlarının çok sayıda modeli tanımlayabilecek şekilde eğitilebilme yetenekleri, farklı bir şekilde, ağ yapısına tanıtılacak veri setlerinin eksik ya da gereksiz verilerle dolu olduğu durumlarda, orijinal modellerin saptanması amacıyla da kullanılabilmektedir. Uygun şekilde eğitilmiş bir ağ yapısına gürültülü bir veri setinin tanıtılması durumunda ağ yapısı, söz konusu veri setinin ait olduğu orijinal modeli tanımlayabilmektedir. Yapay Sinir Ağlarının bu yeteneği pratikte, özellikle imge yenileme problemlerinde başarı ile kullanılmaktadır.

İnsan beyninin bilgi işleme stratejilerinden esinlenilerek geliştirilen Yapay Sinir Ağları, yukarıda tanımlanan yetenekleri sayesinde, çok çeşitli alanlarda uygulama şansı bulabilmektedir. Hatta bu konuda, Yapay Sinir Ağlarının uygulama alanlarına ilişkin tüm potansiyelinin hala çok uzağında bulunulduğu rahatlıkla söylenebilir.

Yapay Sinir Ağları hakkında yakın bir zamanda yapılan bir çalışmada, bilgisayar bütünleşik imalat sistemlerinde Yapay Sinir Ağlarının ilişki kurma ve genelleme yapma yeteneklerinden faydalanma olanağına dikkat çekilmekte ve imalat sistemlerinde, mevcut teknolojiler ile üstesinden gelinemeyen kompleks süreçlerin planlanmasında ve hayata geçirilmesinde, sinir ağları teknolojilerinin kullanımı örnek olarak verilmektedir (*May, 1994*). Yine aynı çalışmada, süreç modelleme, optimize etme, izleme ve kontrol, teşhis ve tanımlama ile ticari ürün imalatı gibi karmaşık ve kompleks imalat görevlerinin yerine getirilmesinde, üniversitelerin ve sanayinin sinir ağlarına başvurma ilgisinin arttığı vurgulanmaktadır. Yapay Sinir Ağlarının, girdi setleri uzayından çıktı setleri uzayına doğrusal olmayan, karmaşık ilişkileri kurabilecek şekilde nasıl eğitilebildikleri ve girdi setlerinin yetersiz ya da gereksiz verilerle dolu olduğu durumlarda, doğru eşleştirmeleri yapabilecek çağrışımlı bir hafıza olarak nasıl tasarlanabildikleri düşünüldüğünde; modelleme, analiz etme, tahminde bulunma ve sistem performansını optimize etme amacıyla Yapay Sinir Ağları kullanımının ne kadar uygun olduğu anlaşılabilir.

Günümüzde Yapay Sinir Ağları, insan seslerinin tanımlanması ve analiz edilmesi; model tanımlama ve sınıflandırma; enerji rezerv tahminlerinin yapılması; pazar analizleri için finansal eğilimlerin tanımlanması ve yorumlanması; bir çok parçadan oluşan tasarımların imalatı; haberleşme ağları güzergahlarının güvenilir ve esnek bir şekilde tayin edilmesi; süreç modelleme, izleme ve kontrolü gibi gittikçe genişleyen bir teknolojik tabanda ortaya çıkan problemlerin çözümünde geniş ölçüde kullanılmaktadır. Yapay Sinir Ağlarının uygulama olanağı bulduğu bu alanlarda, beraberinde getirdiği temel üstünlükler şu şekilde sayılabilir:

1. Sinir ağı modelleri, her bir yeni veri tanıtımının ardından sürekli bir biçimde güncellenebilmekte ve performansını her an optimum tutabilmektedir.
2. Sinir ağları, çok büyük miktarlarda ve çeşitte girdi değişkenleri ile başarıyla ve süratle başa çıkabilmektedirler.
3. Sinir ağları, gürültülü verileri süzebilme ve eksik veriler için interpolasyon ile tahmin yapabilme yeteneklerine sahiptirler.

Yapay Sinir Ağlarının, yukarıda sayılan tüm üstünlüklerine ve genişleyen kullanım alanlarına rağmen, başarı düzeyleri uygulamadan uygulamaya farklılıklar göstermektedir. Ancak uygulamalarda yaşanan bu gibi olumsuz durumlar, kesinlikle, sinir ağlarının gelişimini aksatmamalı; model kurmada ve öğrenme kurallarının tayininde yapılan sınırlamaları inceleme ve bu sınırlamalarda geliştirmeler yapma fırsatı olarak görülmelidir.

1.3 Yapay Sinir Ağları ve Modern Bilgisayarlar

Yapay Sinir Ağları araştırmalarının altında yatan motivasyon, insan vücudunun özellikle eşzamanlı görsel idrak ve konuşulanı anlama gibi işlevleri ile öğrenme sayesinde farklı ortamlara uyum sağlama gibi yeteneklerinin, son derece karmaşık bir sinir ağından ibaret olan beyinde gerçekleştirilen yapısal ve işlevsel ilkelerden kaynaklandığı inancıdır.

İnsan beyni, sinir adı verilen, elektrik sinyallerini nakledebilen özel hücrelerin birer işlemci olarak görev aldığı muazzam bir sinir ağından meydana gelmektedir. Beyin, işlem gücü tek başına zayıf olan bu çok sayıdaki sinir hücrelerini, son derece karmaşık bir ağ yapısı içinde organize ederek güçlü bir işlemci olarak ortaya çıkarır. Bu muazzam sinir ağı, kural bazlı ve seri biçimde işlem yapan merkezi bir işlemciden oluşan modern bilgisayarlardan, gerek yapısal gerekse işlevsel anlamda pek çok farklılıklar göstermektedir. Örneğin, insan beyni, bilgisayarların aksine, metabolizmanın durumunu o anki mevcut verilerin, girdilerin yanı sıra geçmiş tecrübelere de bağlı olarak tayin eder. Sinir ağlarındaki değerlendirmelerde önemli bir yer tutan bu geçmiş tecrübeler, sinir ağı yapısı içinde, sinir hücrelerinin faaliyetlerinde, aralarında kurmuş oldukları bağlantılarda ve bu bağlantıların gücünde temsil edilirler ve bu temsil, sürekli bir biçimde sisteme giren yeni verilerle birlikte güncellenir.

Sinir ağları ile modern bilgisayarlar arasındaki bazı temel, yapısal ve işlevsel farklılıklar aşağıdaki şekilde sıralanabilir: Yapılan karşılaştırmalar, kolaylık olması açısından iki sütun halinde, yan yana yazılmıştır:

Tablo 1.1 Sinir ağları ile modern bilgisayarlar arasındaki temel yapısal ve işlevsel farklılıklar (*Bose ve Liang, 1996; s.29*).

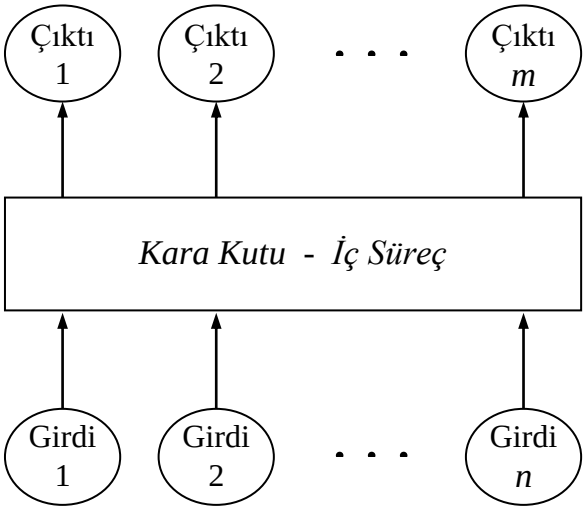
Yapay Sinir Ağları	Modern Bilgisayarlar
▪ Yapay Sinir Ağlarında gerçekleştirilen işlem paralel, kesikli ve düzensizdir.	▪ Modern bilgisayarlarda ise işlem seri, sıralı ve eşzamanlıdır.
▪ Yapay Sinir Ağları örnek veriler ile eğitilirler. Eğitim sonucu öğrenme, sinir ağı yapısında, işlemci elemanlarda ve aralarında kurdukları bağlantıların gücünde değişimler şeklinde gözlemlenir.	▪ Modern bilgisayarlar, mantıksal süreçler şeklinde talimatlar ile programlanırlar. Programlama sırasında saptanan parametre değerleri işlem sırasında değişikliğe uğramazlar.

Yapay Sinir Ağları	Modern Bilgisayarlar
▪ Yapay Sinir Ağlarında bellek elemanları ile işlemci elemanlar birdirler. Bilgiler ağ yapısında, işlemci elemanlar arasındaki bağlantılarda depolanırlar.	▪ Modern bilgisayarlarda bellek elemanları ile işlemci elemanlar ayrıdırlar. Bilgiler ayrıca tanımlanan ve yenilenebilen ayrı bir yerde depolanırlar.
▪ Yapay Sinir Ağları hata toleranslıdırlar. Birbirine benzer girdiler aynı sonuca, çıktıya yakınsayabilirler.	▪ Modern bilgisayarlarda hata toleransı yoktur. Girdiler ile çıktılar arasında birebir bir ilişki söz konusudur.

Sinir ağları üzerine yapılan çalışmalarda, ağ yapılarının karakteristiklerinin ve işleyişinin nitel olarak ve uygulamalı problemlerin çözümüne yönelik olarak anlaşılmasına çalışılmaktadır. Bu çalışmalarda amaç, insan beyninin yapısal ve işlevsel özelliklerinin kopyalanmasından çok; modern bilgisayarlarla olan benzerlik ve farklılıklarından hareketle daha etkin sonuçlar üretecek modeller geliştirmek olmalıdır. Bu bakış açısı daha kullanışlıdır. Zira insan beyni ve sinir sistemi hakkında elimizdeki mevcut bilgi miktarı son derece azdır. Ayrıca ele alınacak bir özelliğin hayata geçirilebilmesi, mevcut teknolojiye ve uygulama alanına bağlı olarak da değişiklik gösterebilmektedir.

1.4 Yapay Sinir Ağlarında Genel Yapı

Yapay Sinir Ağları hakkında, literatürde, çok sayıda ve bir o kadar da çok çeşitte ağ yapısı söz konusudur. Sinir ağlarının genelinde ortaklık gösteren özelliklerden hareketle aşağıdaki temel sinir ağı yapısı çizilebilir. Ele alınan bu temel sinir ağı yapısını girdi katmanı, orta katman ve çıktı katmanı olmak üzere üçe ayırarak ele almak ve incelemek mümkündür:



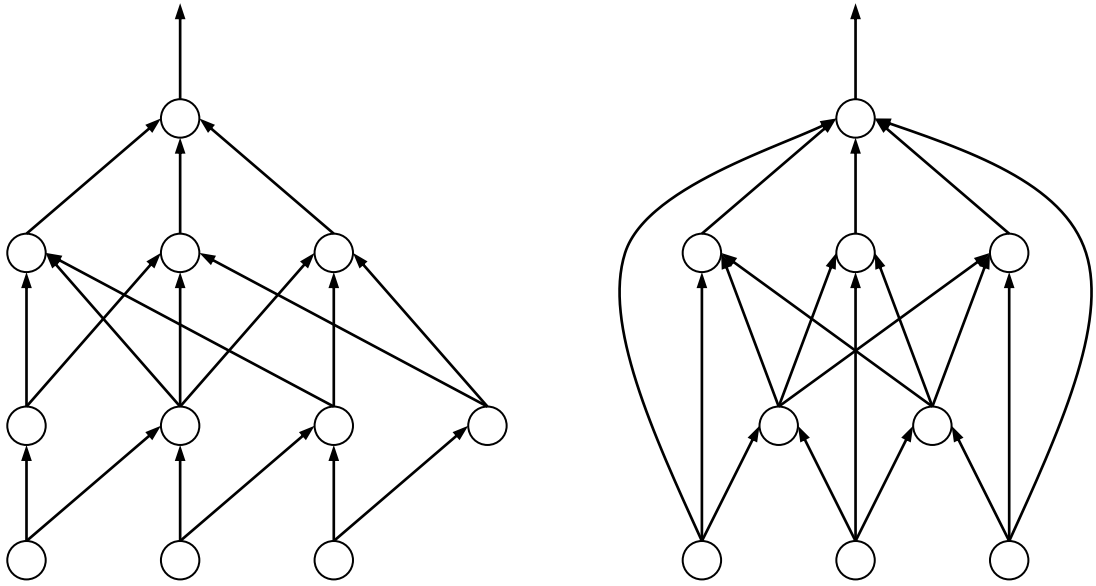
Şekil 1.2 Yapay Sinir Ağlarının temel yapısı (*Masters, 1993; s.9*).

- Girdi Katmanı. Bir sinir ağı yapısında, genellikle çok sayıda olmak üzere, en az bir adet girdi bulunur. Yukarıdaki şekilde girdi örneklerinin simgelenmesi amacıyla her bir girdi birer işlemci birim olarak ele alınmıştır. Aslında girdi katmanında işlemci birimler bulunmaz; diğer bir ifade ile bu birimlerde herhangi bir işlem gerçekleştirilmez.

- Çıktı Katmanı. Bir sinir ağı yapısında, ağ yapısının fonksiyonuna bağlı olmak üzere, en az bir adet çıktı bulunur. Çıktı katmanındaki birimler sinir ağının sonuçlarını ortaya çıkarırlar. Bu birimler girdi katmanındaki birimlerin aksine gerçektirler; kendi girdi setlerini işlerler ve çıktılarını üretirler.

- Orta Katman. Ağ yapısında orta katman, girdi birimlerinin "kara kutu" adını verebileceğimiz bir süreçten geçerek çıktı birimleri ile ilişki kurmasını sağlar. Burada söz edilen kara kutunun özelliği, seçilen sinir ağı modeli tarafından belirlenir. Bu seçimdeki en önemli kararlardan biri ideal katman sayısının tayinidir. İlerleyen bölümlerde de değinileceği gibi, sinir ağı yapılarında iki adetten fazla saklı katman kullanmayı gerektiren hiç bir teorik neden yoktur; hatta, uygulamada karşılaşılan problemlerin büyük bir çoğunluğu için tek bir saklı katman kullanmak yeterli olmaktadır.

Katmanlar arasındaki bu düzen çok yaygın olmasına rağmen, girdi birimlerinin çıktı birimleri ile doğrudan bağlantı kurduğu ağ yapılarına da rastlanmaktadır. Aşağıda söz konusu bağlantı düzenlerine ait, iki ayrı örnek verilmektedir: Yapay Sinir Ağları ile ilgili teorik çalışmalarda, genellikle, birinci örnekteki bağlantı düzenini esas alan ağ modelleri ele alınmaktadır.



Çok katmanlı ve ileri sürümlü bir Sinir Ağı Modeli

Katmansız ve kestirme yollu bir Sinir Ağı Modeli

Şekil 1.3 Yapay Sinir Ağlarında farklı bağlantı düzenleri (*Bose ve Liang, 1996; s.121*).

1.5 Yapay Sinir Ağları Araştırmalarında Yaşanan Problemler

Yapay Sinir Ağları ile ilgili gerek teorik gerekse pratik alanlarda yapılan çalışmalarda, küçük ancak can sıkıcı bir takım eksikliklerden kaynaklanan sorunlar yaşanabilmektedir. Bu eksiklikler, temel olarak, verilen bir probleme ait etkin ağ yapısı tasarımının ilk defasında tamamlanmasının neredeyse imkansız olmasından gelmektedir. Ağ yapılarında etkin tasarımlara, ancak denemeler neticesinde, bir çok adımdan sonra ulaşılabilmektedir ve elde edilen tasarımlar için olabilecek en iyi tasarım nitelendirmesi asla yapılamamaktadır. Her zaman daha iyi sonuçlar üreten bir tasarım söz konusudur.

Yapay Sinir Ağlarında yaşanan önemli bir problem, eğitilen bir sinir ağına tanıtılabilecek mümkün veri setleri için, ağ yapısının doğru çalışıp çalışmayacağı sorunudur. Bu konuda, ağ yapılarının performanslarını değerlendirme teknikleri son derece yetersiz kalmaktadır. Aslında iyi eğitilen ve ardından makul bir veri seti ile test edilen bir ağ yapısının kullanımı sırasında sorun yaşanması oldukça az rastlanan bir durumdur.

Ağ yapılarında performans değerlendirme son derece güç olduğundan, sinir ağları ile ilgili yeni ve güçlü teoremlerin ispatı her zaman heyecan verici karşılanmaktadır. Ancak bu tip araştırmaların çoğu, belirli problemleri çözmeye yönelik, özel ağ yapılarının yetenekleri ile ilgilidirler. Yapay Sinir Ağları araştırmalarında diğer bir problem de burada yaşanmaktadır. Araştırmalar sonucu elde edilen sonuçlar pratik önemden çok teorik öneme sahiptirler. Bu sebeple elde edilen sonuçlar, teorik ve pratik alanlardaki uygulamalarda farklılıklar gösterebilmektedirler. Örneğin; bir problemin çözümü için tek saklı katmanın yeterli olacağını söyleyen bir teorem bulunmasına rağmen, uygulamada iki saklı katman daha iyi sonuçlar verebilmektedir. Önerilen ağ yapısı için teknoloji ve eğitim gerekliliklerinin karşılanamamasından kaynaklanan bu tip durumlar, daha önce de belirtildiği gibi, az rastlanan ve deneme yanılma sonucu çözüm üretilebilecek durumlardır.

Yapay Sinir Ağları üzerine yapılan çalışmalarda sıkça sorulan sorulardan biri de, ağ yapılarının ne tip fonksiyonları öğrenebileceği problemidir. Aslında ele alınan her bir problem, ağ yapısından öğrenmesi istenen fonksiyonun tanımlanması şeklinde görülebilir. Aynı zamanda Yapay Sinir Ağı yapılarının kendileri de birer fonksiyonu yerine getirirler. Kendilerine tanıtılan girdileri işlerler ve istenen çıktıları üretirler. Bu durumda, ele alınan problemi temsil eden bir fonksiyona yakınlaşacak, ağ yapısını temsil eden diğer bir fonksiyonu aradığımızı söyleyebiliriz. Yine, bir ağ yapısının, gerçek fonksiyon değerlerine yeteri kadar yakın çıktılar üretmeyi öğrenebilmesi durumunda, ele alınan probleme çözüm üretebileceğini söyleyebiliriz.

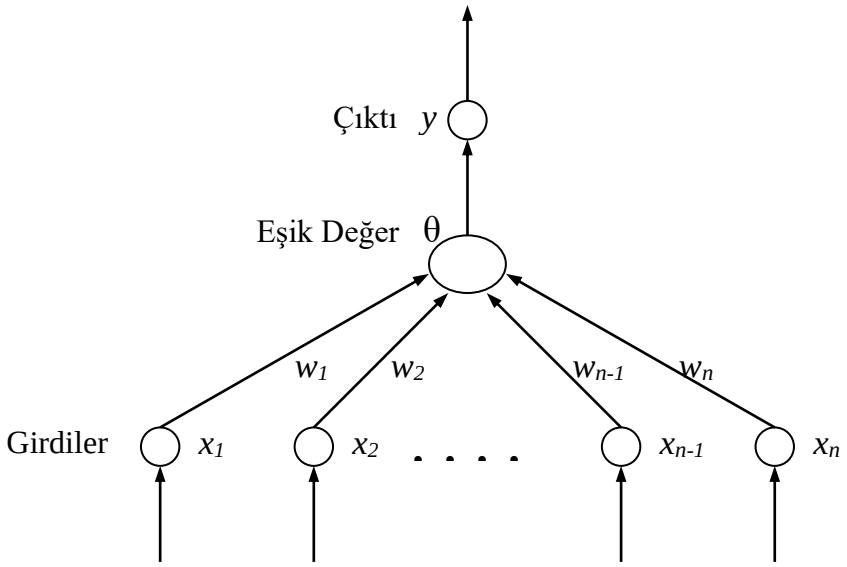
2. YAPAY SİNİR AĞLARINDA TEMEL KAVRAMLAR

2.1 Yapay Sinir Ağlarında Birim Elemanlar

Sinir ağları çalışmaları için en önemli nedenlerden biri de, paralel bir düzende yerel çözümler üreten basit işlemci birimlerin, biyolojik sinir hücreleri gibi, oluşturduğu ağ yapıları ile pratik alanlardaki zor problemlere yönelik, hızlı ve optimum sonuca yakın, sezgisel (*heuristic*) çözümler ortaya koyma ihtiyacıdır. Yapay Sinir Ağları, fonksiyonlarını daha küçük ve daha basit işlemci birimlere dağıtan bir yapıya sahiptirler. Böylelikle, pratik alanlardaki karmaşık süreçlerin parçalara ayrılması sonucu, ekonomik ve teknolojik olarak mümkün olmayan problem çözümlerine ulaşılabilmektedir.

Sinir ağı yapılarının yukarıda değinilen parçalayıcı özelliği, beraberinde ağ yapısındaki işlemci birimlerin kurdukları bağlantı sayılarının sınırlandırılması gerekliliğini getirmektedir. Burada, her türlü sinir ağı modeli için geçerlilik gösteren, "İşlemci birimlerin tüm olası bağlantılarının kurulduğu ağ yapılarından kaçınılması gerekir." ifadesi, sinir ağlarının mevcut teknolojik imkanlar ile pratik alanlarda kullanılabilmesi ve hayata geçirilmesi açısından son derece önem taşımaktadır. Ancak, teorik alanda çalışan bir çok araştırmacı bu problemi görmezlikten gelmektedir. Literatürdeki çoğu araştırma sonucu, tam bağlantılı sinir ağı modellerine dayanmaktadır.

Yapay Sinir Ağları araştırmalarında bilim adamları, insan beyninde ya da modern bilgisayarlarda ortaya çıkabilecek kesikli süreçleri ifade etmek ve matematiksel açıdan analiz edebilmek için hücre (*neuron*) adını verdikleri basit mantıksal birimler modellemişlerdir. Aşağıda, n adet girdisi ve bir adet çıktısı olan bir işlemci birim ele alınmaktadır. Bu birimde $(n+1)$ adet parametre bulunmaktadır. Bunlar n adet girdiye ait ağırlık değerleri ve işlemci birimin kendi eşik değeridir.



Şekil 2.1 Bir İşlemci Birim modeli (*Bose ve Liang, 1996; s.23*).

Ele alınan bu işlemci birim, *k* indisli kesikli zaman aralıklarında, seçilen transfer denklemine uygun olarak girdilerini işler ve bir çıktı hesaplar. Sözü edilen transfer denklemleri, girdilerin ağırlıklı toplamlarından karmaşık diferansiyel denklem topluluklarına kadar uzanan geniş bir yelpazede seçilebilirler. Aşağıda en basit transfer denklemlerinden biri ele alınmaktadır. Bu transfer denklemine göre girdilerin ağırlıklı toplamları eşik değerini geçerse işlemci birimin çıktısı 1; aksi halde 0 olarak hesaplanır.

$$y_i(k+1) = \begin{cases} 0 & \text{eğer} \quad \sum_{i=1}^{n} w_i x_i(k) \geq \theta_i \\ 1 & \text{aksi halde} \end{cases} \qquad (2.1)$$

Yukarıda ele alınan, son derece basit hücre modellerinin oluşturduğu bir ağ yapısı ile herhangi bir mantıksal işlem hesaplanabilir. Hatta, uygun seçilecek ağırlık ve eşik değerleri ile bu ağ modellerinin modern bilgisayarları simüle etmesi mümkündür; yani böyle bir ağ modeli bir bilgisayarın yaptığı her şeyi yapabilir.

Sinir ağları modelleri, biyolojik olanlar da dahil olmak üzere, gerçek sinir ağı yapıları üzerine bir çok basitleştirici varsayımlarda bulunurlar. Bu varsayımlar, sinir ağlarının istenilen özelliklerinin açığa çıkarılması, anlaşılabilmesi ve matematiksel açıdan analiz edilebilmesi için son derece gereklidirler. Tanımlanan bu işlemci elemanlardan oluşturulacak bir sinir ağı modelinin, aşağıdaki şekilde bir çok basitleştirici varsayımlara dayandırılması mümkündür:

- Bir hücrenin ($k+1$). zamandaki durumu, o hücrenin k. zamandaki durumuna ve k. zamanda o hücreye gelen girdilere bağlıdır.

- Bir hücrenin ($k+1$). zamanda uyarılabilmesi sadece ve sadece, k. zamanda o hücreye gelen uyarıcı girdilerin miktarının hücrenin eşik değerini aşması ve aynı zamanda herhangi bir engelleyici girdi olmaması durumunda mümkündür.

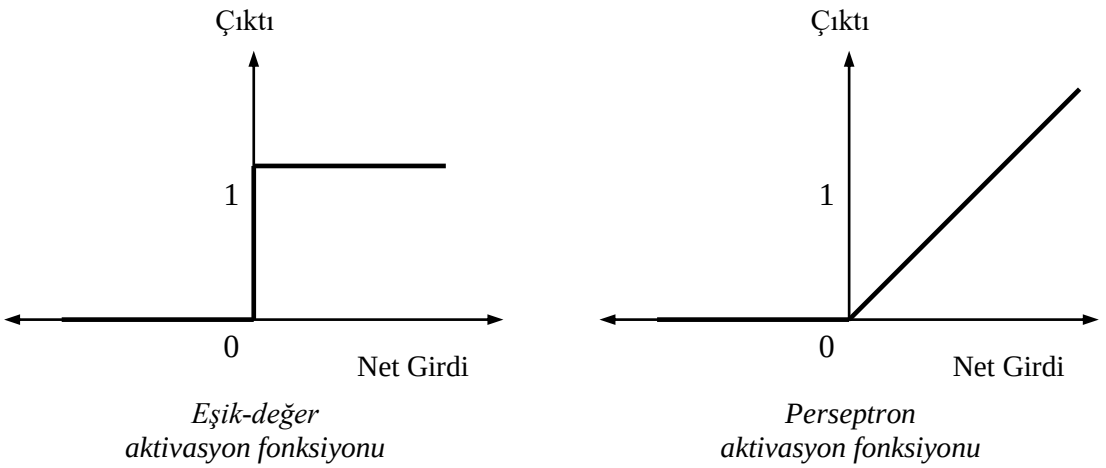
- Her bir hücrenin girdisi ve çıktısı arasında standart bir gecikme olduğu varsayılır. Bunun sonucu olarak ağ yapısındaki tüm hücreler ve dolayısıyla tüm sinir ağı aynı zaman çizelgesinde, eşzamanlı olarak çalışacaktır.

2.2 Aktivasyon Fonksiyonları

Aktivasyon fonksiyonları doğrusal olmayan fonksiyonlardır ve ağ yapılarında işlemci birimlerin net girdi değerlerine uygulanarak söz konusu işlemci birimlerin çıktı değerlerinin belirlenmesinde kullanırlar. Aktivasyon fonksiyonları genellikle, belirledikleri çıktıların alabilecekleri değer aralıklarını sınırlandıracak şekilde seçilirler. Pratik alanlardaki uygulamalarda kullanılan en yaygın değer aralığı [0, 1] aralığıdır. Yaygın olarak kullanılan değer aralıklarından bir diğeri ise [-1, 1] aralığıdır.

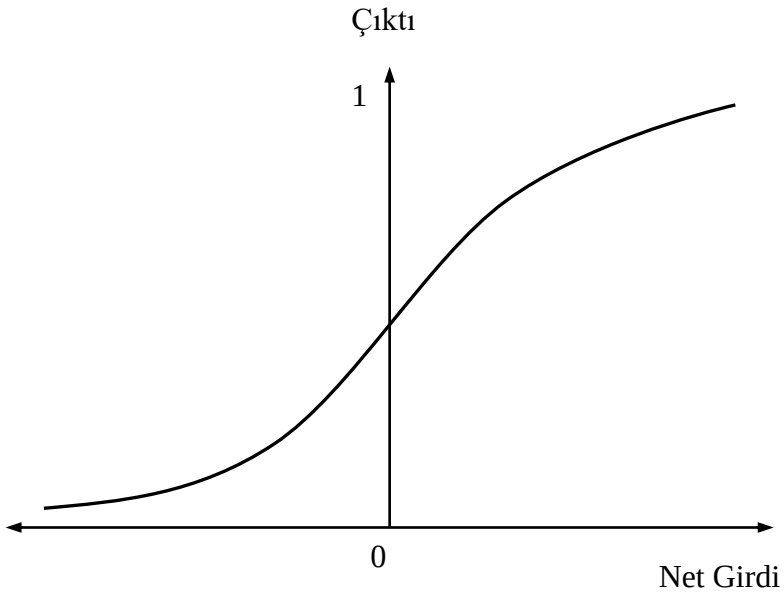
Orijinal perseptronu da içine alan ilk yapay sinir ağı modellerinde, en basit aktivasyon fonksiyonlarından biri olan Eşik-değer aktivasyon fonksiyonu kullanılmıştır. Bu aktivasyon fonksiyonuna göre, girdilerin ağırlıklı toplamları ile elde

edilen net girdi değerinin işlemci birimin eşik değerini aşması durumunda çıktı değeri 1, aksi halde 0 olarak hesaplanır. Daha sonra geliştirilen perseptron modellerinde, işlemci birimlerin çıktı değerlerini, girdilerin ağırlıklı toplamlarının eşik değeri aştığı durumlarda, direkt net girdi değeri olarak hesaplayan aktivasyon fonksiyonları kullanılmıştır. Aktivasyon fonksiyonlarının seçimindeki bu gelişim, kolay diferansiyellenebilen aktivasyon fonksiyonlarının getirdiği bir takım avantajlardan kaynaklanmaktadır. Aşağıda sırasıyla, tipik bir eşik-değer aktivasyon fonksiyonu ile daha sonra geliştirilen perseptron aktivasyon fonksiyonunu gösteren şekiller çizilmiştir:



Şekil 2.2 Eşik-değer ve perseptron aktivasyon fonksiyonları (*Masters, 1993; s.80*).

Günümüzde en yaygın şekilde kullanılan aktivasyon fonksiyonları S şeklindeki sigmoid fonksiyonlarıdır. Sigmoid bir aktivasyon fonksiyonu, türevi her zaman pozitif olan ve sınırlandırılmış bir değer aralığında monoton artış gösteren bir fonksiyondur. Uygulamalarda en çok kullanılan sigmoid fonksiyonlardan biri logistik fonksiyonudur:



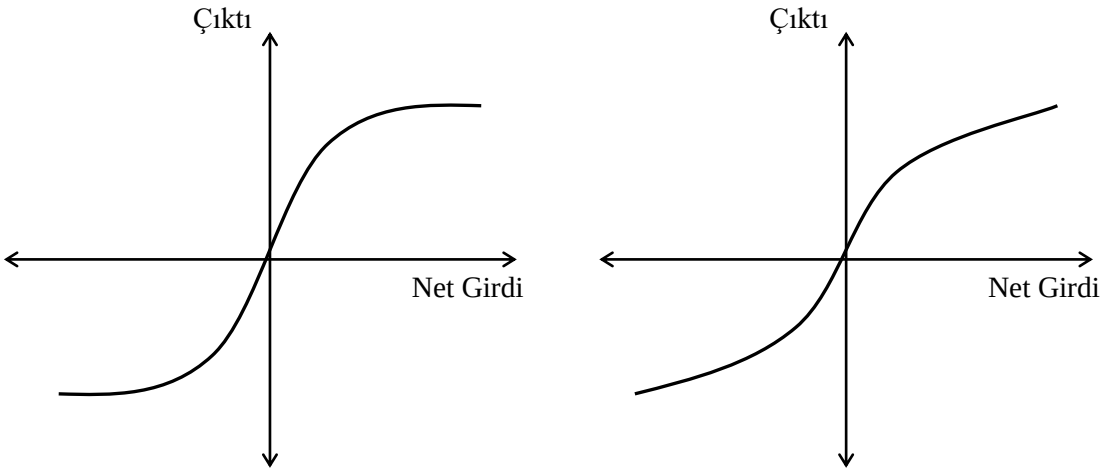
Şekil 2.3 Logistik aktivasyon fonksiyonu (*Blum, 1992; s.39*).

$$f(x) = 1 / (1 + e^{-x})$$

Bu fonksiyonun önemli bir üstünlüğü de türevinin kolaylıkla hesaplanabilmesidir. Fonksiyonun herhangi bir noktadaki türevinin değeri, fonksiyonun o noktadaki değerinden hareketle aşağıdaki formül vasıtasıyla elde edilebilmektedir:

$$f'(x) = f(x) * [1 - f(x)]$$

Pratik alanlardaki bir çok uygulamada görülmüştür ki aktivasyon fonksiyonunun seçiminin, ağ yapısının performansına etkisi son derece azdır. Ancak, ağ yapısında eğitim hızını belirleyen en önemli faktör aktivasyon fonksiyonudur. Bu konuda yapılan araştırmalardan birinde, daha küçük türev değerine sahip sigmoid fonksiyonlarının, temel geri-yayılım (*backpropagation*) eğitim algoritması için daha yavaş öğrenmeye neden olduğu ortaya çıkarılmıştır. Söz konusu çalışmada örnek olarak ele alınan iki aktivasyon fonksiyonunun şekilleri aşağıda verilmektedir. Bunlar arasında ilk sırada ele alınan aktivasyon fonksiyonu için eğitim hızı diğerinden daha yüksek çıkmaktadır.



Şekil 2.4 Aktivasyon fonksiyonu seçiminin eğitim hızına etkisi (*Masters, 1993; s.82*).

2.3 Çıktı Katmanında Doğrusal İşlemci Birimler

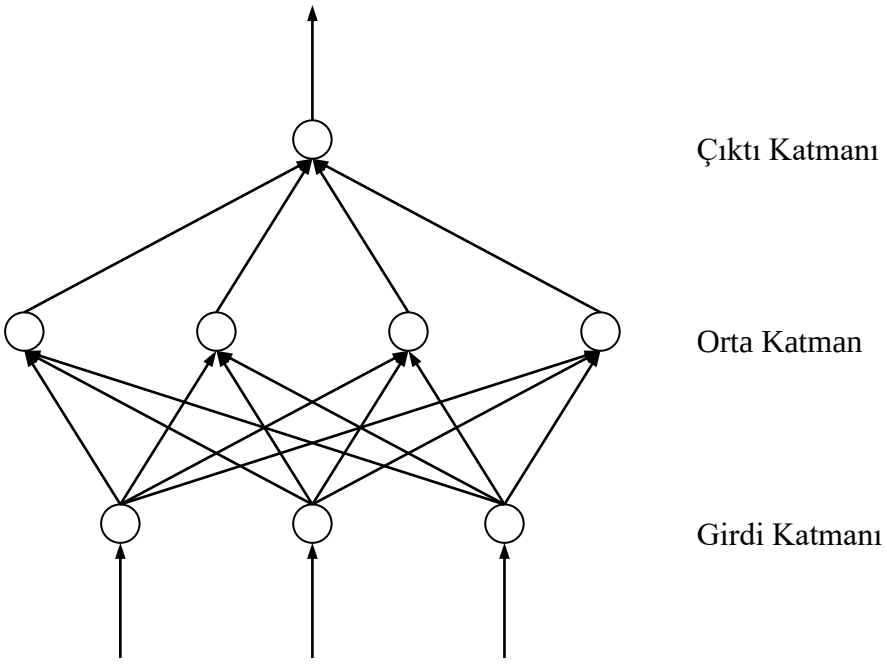
Şimdiye kadar saklı katmanlardaki işlemci birimler ile çıktı katmanındaki işlemci birimler için aynı aktivasyon fonksiyonunun kullanıldığı varsayıldı. Bu durum en çok tercih edilen ve çoğu zaman tavsiye edilen durumdur. Ancak bu ifadeden, aynı fonksiyon kullanımının her zaman doğru ve gerekli olduğu anlamı çıkarılmamalıdır. Aslında, bazı uygulamalarda aynı aktivasyon fonksiyonu seçiminin istenmeyen sonuçlara neden olduğu da görülmüştür. Böyle bir durumda, çıktı katmanındaki işlemci birimler için $f(x) = x$ aktivasyon fonksiyonu olarak seçilebilir. Yani, söz konusu işlemci birimlerin net girdi değerleri, direkt olarak işlemci birimin çıktı değeri olarak kabul edilebilir.

Çıktı katmanında yer alan işlemci elemanlar için, doğrusal aktivasyon fonksiyonu kullanımının, beraberinde getirdiği üstünlükleri yanında eksiklikleri de söz konusudur. Doğrusal aktivasyon fonksiyonları ile çıktı değerlerinin belirli değer aralıklarında sıkıştırılması önlenebileceği ve hesaplamalarda önem taşıyan uç değerler

korunabileceği gibi; aksi durumlarda uç değerlerin zarar veren etkileri de ortaya çıkarılmış olabilir. Bu bakımdan konuya en iyi yaklaşım şu şekilde belirlenebilir: Öncelikle değer aralığını sınırlandıran aktivasyon fonksiyonları kullanılmalıdır ve doğrusal aktivasyon fonksiyonu kullanmak için açık, net bir neden görülmedikçe ağ yapısı değiştirilmemelidir.

2.4 Perseptron ve Doğrusal Ayrıştırılabilirlik Şartı

İlk defa 1950'li yılların sonunda Frank Rosenblatt tarafından ortaya çıkarılan perseptron (*perceptron*), ilk ve en basit sinir ağı yapılarından biridir. Biyolojik ağ modellerine dayanılarak tasarlanan bu sinir ağı yapısı, aynı zamanda eğitilebilen ilk Yapay Sinir Ağı yapısıdır. Ancak, söz konusu ağ yapısının, kullanım alanını kısıtlandıran önemli bir eksikliği vardır. O da, perseptronun sadece, örnek veri seti doğrusal bir şekilde ayrılabilen sınıflandırma problemlerini çözme yeteneğine sahip olmasıdır. Günümüzde, pratik alanlarda kullanılan benzer pek çok ağ yapısının anlaşılmasına katkı sağlaması açısından, bu basit ağ yapısının ele alınması yarar sağlayacaktır.



Şekil 2.5 Perseptron - temel yapı.

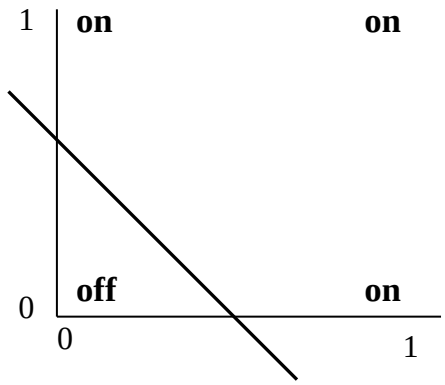
Perseptron üç katmanlı ve ileri beslemeli bir ağ yapısına sahiptir. Yukarıda verilen şeklinden de anlaşılabileceği gibi, bu sinir ağı yapısında ilk katman, girdileri içeren ve herhangi bir işlemin gerçekleştirilmediği girdi katmanıdır. İkinci, orta katmanda yer alan işlemciler, girdiler ile çıktı katmanındaki işlemci birim arasında bağlantı oluşturur. Bu katmandaki işlemci birimler, ağ yapısının doğrusal olmayan bir fonksiyonu yerine getirmesi açısından önem taşırlar. Arzu edilen dönüşümün doğrusal olması halinde orta katman gereksizdir. Son katmandaki işlemci birim ise, orta katmandaki işlemci birimlerden gelen girdiler (ağırlıklı toplamları) ile eşik değerinden hareketle sinir ağı yapısının çıktısını üretir.

Perseptronda, ilk katmandaki girdilerin orta katman ile ilişki kurmasını sağlayan bağlantıların ağırlık değerleri ve orta katmandaki işlemci birimlerin eşik değerleri sabittir. Yani, ağ yapısında hesaplamanın gerçekleştirildiği bu ilk kısım eğitilememektedir. Ancak çıktı katmanındaki işlemci birim için söz konusu parametreler sabit değildirler ve eğitilebilirler.

Bir perseptronun eğitilmesi, girdi katmanında her bir örnek veri için, orta katmandaki işlemcilerin çıktılarının hesaplanması ve ardından bu değerlerden hareketle hesaplanan son katmandaki işlemci birimin çıktı değerinin hedef değer ile karşılaştırılması şeklinde gerçekleştirilir. Uygun bir eğitim algoritması ile çıktı katmanındaki işlemci birimin parametreleri güncellenir ve bir sonraki örnek veri için işlem, çıktı ve hedef arasında göz yumulabilir bir fark elde edilene kadar tekrarlanır.

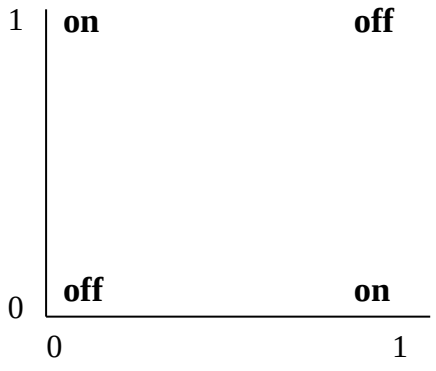
Öğrenme kavramının ilk olarak ele alındığı, perseptron adlı bu sinir ağı yapısının son derece ciddi bir sorunu vardır. Eğitim setindeki örnek girdilerin çıktı-hedef değerlerine bağlı olarak, doğrusal bir şekilde ayrıştırılabilmesi durumunda, söz konusu eğitim seti bir perseptron tarafından öğrenilebilmektedir. Aksi halde, doğrusal bir ayrıştırmanın gerçekleştirilemediği durumlarda, perseptronun ele alınan eğitim setini öğrenmesi mümkün değildir. Aşağıda doğrusal ayrıştırılabilen ve ayrıştırılamayan iki ayrı örnek eğitim seti için bu durumu açıklayan grafikler ele alınmaktadır:

Girdi		Çıktı
i	j	
0	0	0
0	1	1
1	0	1
1	1	1



Doğrusal ayrıştırılabilen veri seti

Girdi		Çıktı
i	j	
0	0	0
0	1	1
1	0	1
1	1	0



Doğrusal ayrıştırılamayan veri seti

Şekil 2.6 Doğrusal ayrıştırılabilen ve ayrıştırılamayan örnek veri setleri (*Masters, 1993; s.5*).

Perseptronun pratik alanlarda kullanım sahasını son derece sınırlandıran bu durum nedeniyle, Yapay Sinir Ağları araştırmaları uzun bir süre kesintiye uğramıştır. Yapay Sinir Ağlarının popülerliğini kaybettiği bu dönem, 1980'li yılların sonunda çok katmanlı ve ileri beslemeli ağ yapılarının etkin eğitim algoritmaları ile birlikte geliştirilmesi sonucu sona ermiştir. Doğrusal ayrıştırılabilirlik şartını gerektirmeyen bu yeni ağ yapısı, Yapay Sinir Ağlarının pratik alanlardaki problemlere gerçek çözümler üretebileceğini tüm dünyaya göstermiştir.

Ancak, Yapay Sinir Ağlarının popüler olduğu son on beş yıllık zaman zarfı içinde geliştirilen sinir ağı modellerinin büyük bir çoğunluğu teorik öneme sahiptir. Birçok ağ modeli pratik alanlardaki uygulamalarda, ya yavaş kaldığından ya beraberinde etkin bir eğitim algoritması geliştirilemediğinden ya da yüksek miktarda teknoloji gerektirdiğinden istenen performansı verememektedir. Ancak etkinliği kanıtlanmış az sayıdaki özel sinir ağı yapıları ile pratik alanlardaki pek çok problem

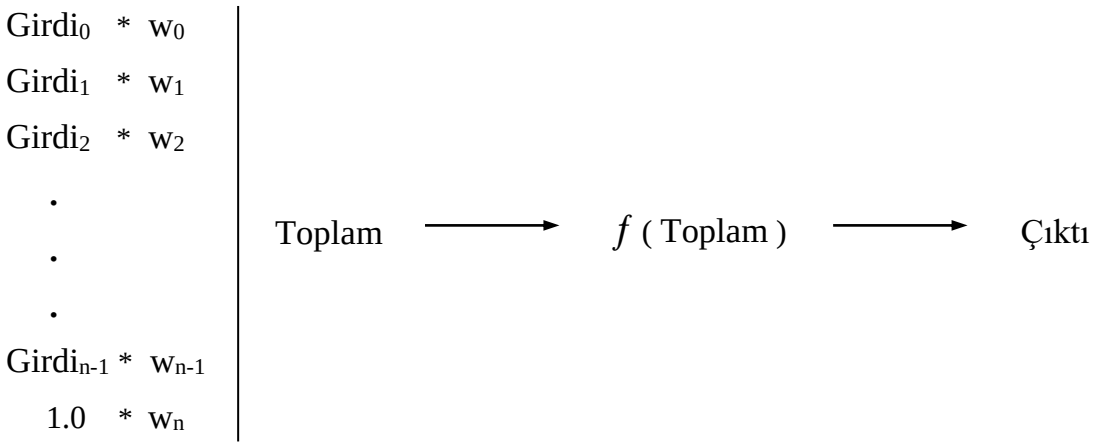
daha hızlı ve daha etkin bir biçimde çözülebilmektedir. Hatta bu modeller sayesinde daha önce çözüm üretilemeyen bir çok probleme de çözümler üretilebilmektedir.

2.5 İleri Sürümlü Yapay Sinir Ağları

İleri sürümlü Yapay Sinir Ağı yapıları, iki ya da daha fazla katmanda, mantıksal bir biçimde sıralanan işlemci birimlerden oluşurlar. Bu ağ yapılarında, bilgi akışının sadece belirli bir yönde ve katman atlamadan gerçekleşmesine izin verilmektedir.

Çok katmanlı ve İleri sürümlü sinir ağı yapılarında, her biri en az bir adet işlemci birim içeren bir girdi katmanı ve bir çıktı katmanı söz konusudur. Bu iki katman arasında, bir ya da daha fazla katman halinde saklı işlemci birimler yer alırlar. Her bir katmandaki işlemci birimlerin girdileri, sadece ve sadece bir önceki katmanda yer alan işlemci birimlerden gelir ve çıktıları, bir sonraki katmanda yer alan işlemci birimlere gider.

Sinir ağı yapısındaki her bir işlemci birimin çıktısı, söz konusu işlemci birimin girdilerinin bir fonksiyonudur. Ağ yapısının temel taşları olan bu işlemci birimlerden biri aşağıda şematik olarak ele alınmaktadır:



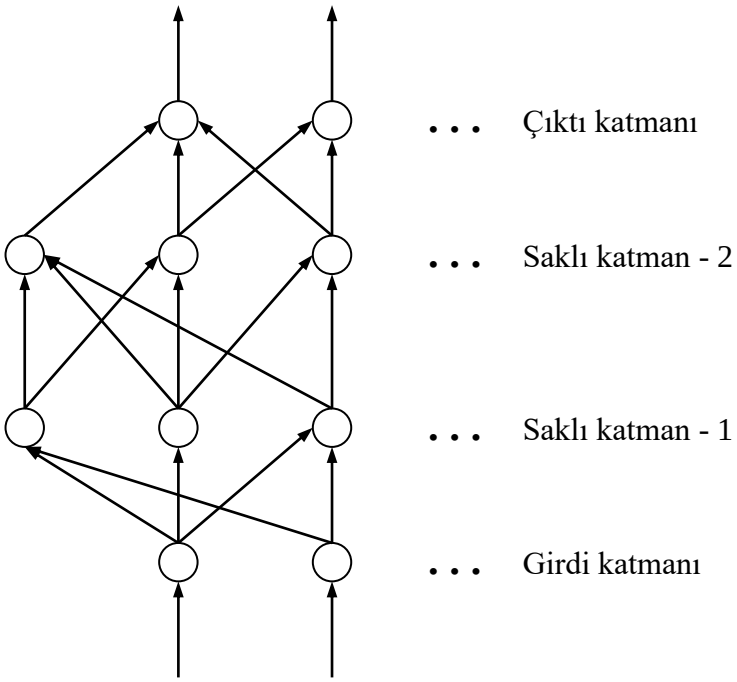
Şekil 2.7 Ağ yapılarında işlemci birimlerin çıktı değerlerinin, girdilerinden hareketle hesabı (*Masters, 1993; s.78*).

Yukarıda ele alınan işlemci birim, *0*'dan *n-1*'e kadar indislenen *n* adet girdiye sahiptir. Aynı zamanda, "Hata değeri" adı verilen ve değeri her zaman 1'e eşit olan ilave bir girdiye daha sahiptir. Buradaki (*n+1*) adet girdinin (*n+1*) adet ağırlık değeri ile çarpılıp toplanması sonucu elde edilen değere, uygun bir aktivasyon fonksiyonu uygulanarak işlemci birimin çıktısı hesaplanır. Hata değerini de içeren, girdi değerlerinin ağırlıklı toplamları "Net girdi" olarak adlandırılır. Net girdi değerinden hareketle işlemci birimin çıktı hesabı aşağıdaki şekilde formüle edilebilir: Söz konusu formülde w_n terimi işlemci birimin hata değerini göstermektedir.

$$\text{Çıktı} = f(\text{Net Girdi}) = f\left(\sum_{i=1}^{n-1} x_i w_i + w_n\right) \qquad (2.2)$$

Böyle bir işlemci birimin hesaplamaya ilişkin karakteristikleri, temel olarak ağırlık değerleri tarafından belirlenir. Bir takım temel gereksinimlerin karşılanması şartıyla, aktivasyon fonksiyonunun ağ yapısının performansına katkısı son derece azdır. Ancak ağ yapılarının eğitim hızı, büyük ölçüde aktivasyon fonksiyonunun seçimine bağlıdır. Bu konu daha sonra ayrıntısıyla ele alınacaktır.

Çok katmanlı ve İleri sürümlü sinir ağı yapılarında, girdi ve çıktı katmanları arasında saklı işlemci birimleri içeren, genellikle bir adet orta katman yer alır. Böyle bir sinir ağı yapısı, "Üç katmanlı sinir ağı" şeklinde adlandırılır. Nadir de olsa, girdi ve çıktı katmanları arasında iki adet saklı katmana da gerek duyulabilmektedir. Bu tip dört katmanlı bir sinir ağı yapısı aşağıdaki şekilde çizilebilir: Söz konusu şekilde dairelerle gösterilen her bir işlemci birim seçilen aktivasyon fonksiyonuna göre bir çıktı hesaplar. İşlemci birimlerin girdileri bir önceki katmandan gelir ve çıktıları bir sonraki katmana gider. Ağ yapısındaki her bir işlemci birim için, genellikle aynı aktivasyon fonksiyonu kullanılır. Daha önce de belirtildiği gibi aşağıdaki şekilde, girdi katmanındaki işlemci birimler tanımsaldırlar ve bu katmanda herhangi bir işlem gerçekleştirilmemektedir.



Şekil 2.8 Dört katmanlı ve İleri sürümlü bir sinir ağı modeli (*Masters, 1993; s.79*).

2.6 Çok Katmanlı ve İleri Sürümlü Sinir Ağlarının Kullanım Alanları

Çok katmanlı ve İleri sürümlü Yapay Sinir Ağları, genel anlamda birer fonksiyon tahmincisidirler. Bu tip ağ yapıları, en azından teorik anlamda, kendilerine öğretilebilen herhangi bir fonksiyonun işlevini yerine getirebilirler. Çok katmanlı ve İleri sürümlü sinir ağlarının öğrenme ve genelleştirme yetenekleri şaşırtıcı ölçüde büyüktür. Pratik alanlarda ele alınan problemlerin büyük bir bölümünü oluşturan, (1) Sonlu sayıda noktalar topluluğu içeren fonksiyonlar ile (2) Sınırlı bir alanda tanımlanmış sürekli fonksiyonlar, üç katmanlı yani tek saklı katmana sahip ağ yapıları tarafından öğrenilebilmektedir (*Masters, 1993; s.86*).

Yukarıda sıralanmayan bir çok fonksiyon da üç katmanlı ağ yapıları tarafından öğrenilebilmektedir. Fonksiyonlardaki kesikli yapılar ya da sınırlı bir alanda tanımlanmamış sürekli fonksiyon yapıları, gerçek hayatta da karşılık bulabilen bir takım varsayımlar altında, Çok katmanlı ve İleri sürümlü sinir ağlarına öğretilebilmektedirler. Pratikte ağ yapısında ikinci bir saklı katmana ihtiyaç duyma,

yalnız bir durumdan kaynaklanabilir. Genellikle süreklilik arz eden, ancak birkaç noktada sıçramalar gösteren fonksiyonların öğrenilmesinde, iki saklı katmana sahip ağ yapıları daha yüksek bir performans göstermektedirler.

Bir fonksiyonun herhangi bir ağ yapısı tarafından öğrenilememesinin en temel nedeni, söz konusu fonksiyonun sınırlı bir alanda tanımlanmış olma koşulunu ciddi biçimde ihlal etmesidir. Ancak böyle durumlarda bile, trigonometrik aktivasyon fonksiyonları kullanımı gibi, bir takım çözümlere ulaşabilmek her zaman mümkündür. Sonuç olarak denilebilir ki, Çok katmanlı ve İleri sürümlü bir Yapay Sinir Ağı ele alınacak herhangi bir fonksiyonu öğrenebilir. Eğer öğrenmede bir sorun yaşanıyorsa bu sadece modelin kendisinden kaynaklanmıyordur. Öğrenmede yaşanan sorunlar, verilen eğitimin yetersizliğinden kaynaklanabileceği gibi ağ yapısında, saklı katmandaki işlemci birimlerin az sayıda tanımlanmış olmasından dolayı da kaynaklanabilir ya da deterministik olmayan bir fonksiyonu öğrenme gayretinden dolayı ağ yapısının öğrenmesinde sorunlar yaşanabilir.

Çok katmanlı ve İleri sürümlü sinir ağlarının önemli bir üstünlüğü de işlem hızı açısından, kullanımda olan en hızlı modeller arasında yer almalarıdır. Bir çok model, problem çözümlerinde belirli çıktı değerlerine yakınsamak adına tekrarlar ve döngüler kullanmaktadırlar ki bu da onları yavaşlatmaktadır. Eşzamanlı işlem yapmanın söz konusu olduğu durumlarda, bu tip sinir ağları, belki de tek uygulanabilir tercih olarak karşımıza çıkmaktadırlar.

Çok katmanlı ve İleri sürümlü sinir ağlarının en temel eksikliği, bu tip ağ yapıları için etkin ve hızlı eğitim algoritmalarının geliştirilememiş olmasıdır. Günümüzde kullanılan ve nispeten yavaş kabul edilen eğitim algoritmaları, genellikle en yakın minimuma yakınsamakta; üstelik elde ettikleri sonuçların global minimum oldukları garantisini de verememektedirler.

Sonuç olarak diyebiliriz ki; Çok katmanlı ve İleri sürümlü sinir ağları özellikle, uzun eğitim süreçlerinin sorun oluşturmayacağı ve hızlı ve seri işlem yapmanın bir gereklilik arz ettiği problemlerin çözümü için idealdirler.

3. YAPAY SİNİR AĞLARINDA ÖĞRENME

3.1 Yapay Sinir Ağlarında Öğrenme Kavramı

Neden Yapay Sinir Ağları sorusunun cevaplarından biri de, sinir ağlarının modern bilgisayarlarda olduğu şekilde tam ve eksiksiz bir programlama gerektirmemesidir. Bir sinir ağı, kendisine tanıtılacak girdi-çıktı ikililerinden hareketle arzu edilen fonksiyonları öğrenebilir; eldeki verilerden kurallar çıkarabilir.

Burada, işlemci elemanların oluşturduğu bir sinir ağı modelinde, sürekli bir bellek dolayısıyla öğrenme yetisi, hücreler arasındaki kapalı ya da geri bildirimli devrelerin varlığına bağlıdır. Aksi taktirde, dış uyarıcıların yokluğunda, ağ yapısının etkinliği bir süre sonra durur ve tüm hücreler sakin duruma geçerler. Bu bellek ve öğrenme formuna insan beyninde de rastlanmaktadır. Ancak, beyninde tüm bilgiler bu şekilde depolanmazlar. İnsan beyninde bellek kavramını kısa ve uzun dönemli bellek şeklinde ikiye ayırmamız mümkündür. Kısa dönemli bellek, kapalı devre oluşturan sinir hücreleri arasında yankılanan sinyaller şeklinde gözlenir ve dinamik bir yapıya sahiptir. Uzun dönemli bellek ise statiktir ve sinir hücrelerinin eşik değerlerinde, aralarında kurdukları bağlantıların gücünde ve oluşturdukları ağ yapısının deseninde değişimler şeklinde gözlenir.

Sinir ağlarında öğrenme, daha iyi bir çıktı elde etmek amacıyla, hücreler arası bağlantıların ağırlık değerlerinin ayarlanması süreci şeklinde tanımlanabilir. Literatürde yer alan bir çok öğrenme kuralı, Hebb adlı bir bilim adamının sinir bilimine olan katkılarından hareketle ortaya çıkmıştır (*Hebb, 1949*). Bu bilim adamı şu hipotezi ileri sürerek sinir ağlarında öğrenme kavramını açıklamıştır: "Bir *i* sinir hücresi bir *j* sinir hücresi tarafından sürekli bir biçimde uyarılır ise, bu iki sinir hücresi arasında, *i* hücresinin *j* hücresi tarafından uyarılma etkinliğini arttıracak şekilde, bazı metabolik değişmeler ve büyüme süreçleri gözlenir". Bu hipotez, matematiksel olarak aşağıdaki şekilde, bir eşitlik ile ifade edilebilir:

x_i : *j* hücresinin *i* hücresinden gelen girdisi,

y_j : *j* hücresinin çıktısı,

w_{ij} : *i* ile *j* hücreleri arasındaki bağlantının ağırlık değeri,

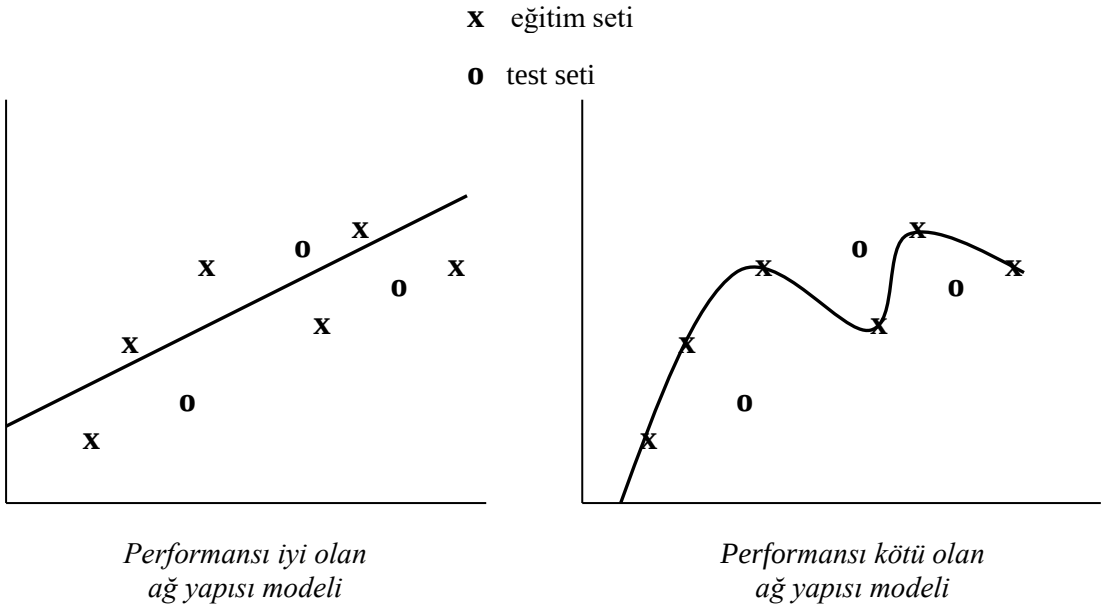
$\alpha > 0$: öğrenme hızı parametresi, olmak üzere (*the Hebbian Learning Rule*):

$$w_{ij}^{yeni} = w_{ij}^{eski} + \alpha\, x_i\, y_j \qquad (3.1)$$

Yapay Sinir Ağları genellikle, aşağıdaki iki farklı yöntemden biri ile eğitilirler. Bunlar arasında daha yaygın olan yöntem Gözetimli öğrenme yöntemidir (*supervised training*). Bu yöntemde öncelikle, eğitim setini oluşturacak girdileri ve bu girdiler ağ yapısına tanıtıldığında elde edilmesi istenen çıktıları içeren örnekler toplanır. Ardından, eldeki girdiler ağ yapısına tanıtılarak hesaplanan çıktı ile hedeflenen çıktı değerleri karşılaştırılır ve ağ yapısındaki bağlantıların ağırlık değerleri güncellenir. Bu güncelleme, karşılaştırma sonucu elde edilen hata miktarını azaltacak şekilde gerçekleştirilir. Ağ yapısında ağırlık değerlerinin güncellenmesi, her bir girdi-çıktı çiftinin tanıtılmasının ardından yapılabileceği gibi tüm eğitim seti tanıtıldıktan sonra da yapılabilir. Burada dikkat edilmesi gereken husus, eğitim setinden örnek seçimi sırasında, bu seçimin her defasında rasgele yapılması gerekliliğidir. Aksi takdirde istenmeyen sonuçlara ulaşılması olasıdır.

Ele alacağımız diğer eğitim yöntemi ise Gözetimsiz öğrenme yöntemidir (*unsupervised training*). Bu yöntemde de diğer yöntemde olduğu gibi, öncelikle bir eğitim setine ihtiyaç vardır. Ancak bu eğitim setinde, toplanan girdilerin hedeflenen çıktı değerlerinin bilinmesine gerek yoktur. Bu yöntemde, eldeki girdilerin birbirinden farklı sınıflardan geldiği ve ağ yapısının hesapladığı çıktı değerinin söz konusu girdinin ait olduğu sınıfı tanımladığı varsayılır. Eğitim sürecinde ağ yapısından, eldeki verilerin belirgin, göze çarpan özelliklerini bulması ve bu özelliklerden faydalanarak girdileri farklı sınıflarda gruplandırılması istenir.

Bir sinir ağının eğitim sürecinden hemen sonra, öncelikle performansının değerlendirilmesi gerekir. Ağ yapısının performansının değerlendirildiği bu süreç bir çok bakımdan eğitim sürecinden daha fazla önem taşır. Çünkü başarılı bir eğitimin en belirgin göstergesi, ağ yapısının, eğitim setine ait olmayan bir girdi için hesapladığı çıktı değerinin doğruluk derecesidir. Buradaki genel prosedür, toplanan veri setinin iki ayrı gruba ayrılması şeklindedir. Bu gruplardan biri ağ yapısının eğitiminde diğeri ise eğitilen ağ yapısının performansının testinde kullanılır.



Şekil 3.1 Eğitim setindeki verilerden hareketle hesaplanan küçük hata değerleri, ağ yapısının performansının iyi olduğu anlamına gelmez (*Masters, 1993; s.11*).

Doğal olarak, performans testi için kullanılan verilerden hareketle hesaplanan hata değerinin, eğitim süreci sonucu elde edilen hata değerinden büyük olması beklenir. Ancak, eğer bu fark çok fazla ise yeni veri setindeki önemli bir bilginin ağ yapısı tarafından henüz öğrenilmediği, dolayısıyla eğitim setimizin tam ve eksiksiz bir şekilde dizayn edilmediği sonucuna varılır. Bu durumda eldeki, iki ayrı veri setinin birleştirilerek yeni eğitim seti olarak kullanılması gerekir. Ardından, yeniden eğitilen ağ yapısının performansının üşenilmeden, tekrardan test edilmesi zorunludur.

3.2 Geri-yayılım Eğitim Algoritması

Geri-yayılım (*backpropagation*) adı verilen eğitim algoritması, Çok katmanlı ve İleri sürümlü ağ yapılarının eğitimi amacıyla geliştirilen ilk etkin yöntemdir. Söz konusu algoritma, doğrusal ayrıştırılabilirlik şartı nedeniyle Yapay Sinir Ağlarının kaybettiği ilginin yeniden artmasına, adeta tek başına vesile olmuştur. Bununla beraber, günümüzde orijinal geri-yayılım eğitim algoritmalarının kullanıldığı çok sayıda başarılı çalışma da söz konusudur.

Geri-yayılım eğitim algoritması, temel olarak bir eğim iniş (*gradient descent*) algoritmasıdır. Çok değişkenli bir fonksiyonda eğim, fonksiyonun en dik olduğu doğrultuyu gösterir. Dolayısıyla, bu doğrultuda atılacak küçük bir adım, fonksiyonun değerinde maksimum artışı sağlayacaktır. Tabii ki, aynı doğrultuda ancak zıt yönde atılacak bir adım da fonksiyonun değerinde maksimum azalışla sonuçlanacaktır. Burada bizim fonksiyonumuz, eğitim setinin ağ yapısına tanıtılması sonucu elde edilecek hata fonksiyonudur. İşte, geri-yayılım eğitim algoritması, ağ yapısının hata fonksiyonu için eğim hesabına, ardından elde edilen doğrultuda ancak zıt yönde bir adım atmaya ve bu işlemi gerektiği kadar tekrarlamaya dayanmaktadır. Ele alınan algoritmada, her defasında hata miktarını azaltacak en iyi doğrultuda (en azından yerel olarak) adım atıldığından dolayı, kısa zamanda minimum hata düzeyine inileceği varsayılmaktadır.

Geri-yayılım eğitim algoritmasının bu şekilde adlandırılmasının temel nedeni, çıktı katmanındaki hata değerinin eğim hesabı ile, işlem yönüne ters şekilde geriye doğru yayılmasıdır. Aslında diğer bir çok eğitim algoritmasında da kullanılan bu operasyon, söz konusu algoritmanın tarihsel önemine binaen kendisine isim olarak verilmiştir.

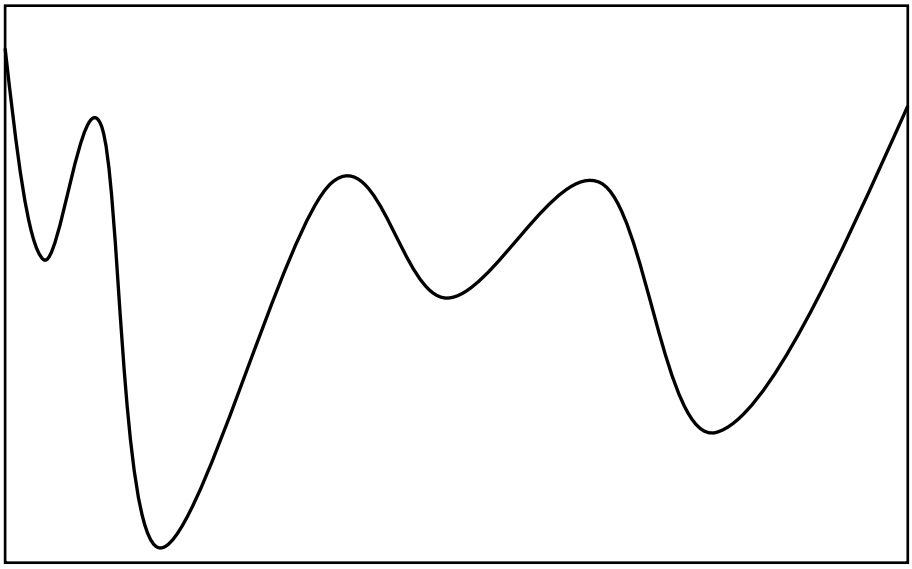
Geri-yayılım eğitim algoritmasında, her bir adımda atılan mesafe "Öğrenme katsayısı" (*learning parameter*) şeklinde adlandırılmaktadır. Bu parametre, sinir ağları çalışmalarında alınması gereken önemli kritik kararlar arasında yer

almaktadır. Çünkü bu mesafenin çok küçük seçilmesi durumunda, yakınsamanın dolayısıyla öğrenmenin son derece yavaş gerçekleşmesi söz konusudur. Aksi durumda ise, söz konusu mesafenin çok büyük seçilmesine bağlı olarak hata fonksiyonunda çılgınca yapılacak sıçramalar sonucunda, yakınsama dolayısıyla öğrenme asla gerçekleşmeyecektir. Geri-yayılım eğitim algoritmasında iki temel kusurdan bahsetmek mümkündür:

- Geri-yayılım eğitim algoritmasının kusurlarından ilki, hata fonksiyonunda en iyi değişim için kullanılan eğim teriminin global değil aksine yerel bir gösterge olmasıdır. Söz konusu gösterge, çok küçük uzaklık farklarında bile çok farklı doğrultuları gösterebilmektedir. Bu nedenle meydana gelen bu dalgalanmalar minimum hata seviyesine ulaşma rotasını dolambaçlaştırmakta ve eğitim süresini uzatmaktadır.
- Algoritmanın diğer kusuru ise, doğrultu tayininin ardından ne mesafede adım atılması gerektiğinin belirlenmesinin son derece güç olmasıdır. Gerektiğinden küçük adımlarla ilerlemeye karar verildiği durumlarda, her defasında doğrultu tayini ve ardından ağ yapısında yapılan hesaplamalar için gereken süre uzamaktadır. Diğer taraftan, gerektiğinden büyük adımların atılması durumunda, sınırı aşmak, hata değerini yükseltmek dolayısıyla yakınsamayı bozmak gibi istenmeyen sonuçlar olasıdır.

3.3 Öğrenme Sürecinde Yaşanan Problemler

- Yerel Minimum Değerlere Saplanmak. Bir fonksiyonun minimum değerini bulmaya çalışan herkes, ele aldığı fonksiyonun ortasında ağır bir topun bulunduğu bir tramboline benzemesini ister. Böyle bir fonksiyonun kesit görünüşü daha önce yukarıda ele alınmıştı. Ancak gerçek hayatta karşılaşılan problemler, aşağıda verilen şekildeki gibi bir kesit görüntüye sahiptirler. Dolayısıyla minimum değere ulaşma süreci için kullanılan herhangi bir algoritmanın global minimum değere ulaşma başarısı, şanslı bir başlangıç noktası seçimine bağlı olmaktadır.



Şekil 3.2 Yerel minimum değerlere saplanma olasılığı yüksek bir fonksiyonun kesit görünümü (*Masters, 1993; s.112*).

Bu noktada, yerel minimum değerlerinden kaçınma amacıyla uyulabilecek, birbirinden farklı iki ayrı prosedür söz konusudur. Bunlardan ilki, yerel minimum değerlerin etrafındaki bir noktadan başlamaktan sakınılması gerektiğini söyler. Yukarıdaki şekil ele alındığında, öğrenme sürecine şeklin sol tarafında yer alan bir noktadan başlanması gerekir. Aksi halde, sağ taraftaki bir noktadan başlanması durumunda, daha sonra yerel minimum değerlerden kaçmak son derece güçleşmektedir.

Yerel minimum değerlerden kaçınmak amacıyla uyulabilecek ikinci prosedür ise, bulunan noktanın yerel bir minimum olup olmadığını saptayan ve yerel minimum ise bu noktadan kaçmayı sağlayan bir süreci işletmektir. Bu amaçla sinir ağları çalışmalarında kullanılan iki ayrı yöntemden söz etmek mümkündür. Bunlardan ilki, uygulanması daha kolay olan ve daha az teknolojik gereksinime gereksinim duyan Benzetimli Tavlama yöntemidir. İkinci yöntem ise daha karmaşık olan ancak daha üstün bir yöntem olan Genetik algoritmalardır. Söz konusu bu yöntemler, diğer eğim esaslı algoritmalarla birlikte kullanılarak Yapay Sinir Ağlarındaki hibrid yaklaşımları oluştururlar. Aslında, bu şekilde oluşturulan hibrit yaklaşımlar, Yapay

Sinir Ağlarında çözüm kalitesini arttırmak amacıyla sık sık başvurulan yaklaşımlardır (*Smith ve diğerleri, 1996*).

Yapay sinir ağlarında ağ yapısının eğitiminde kullanılan eğim (*gradient*) esaslı algoritmalar, şaşırtıcı derecede kolay bir biçimde yerel minimum değerlere saptanmaktadırlar. Bu sebeple ağırlık değerlerinin saptanmasında, sadece bir başlangıç noktasından hareketle global minimuma ulaşmaya çalışma ölümcül bir ihtiyatsızlık olacaktır. Dolayısıyla ağ yapılarında öğrenme prosedürü her zaman, farklı başlangıç pozisyonları için tekrarlanmalıdır. Ayrıca şunu da belirtmekte yarar vardır ki, yapay sinir ağları genellikle çok sayıda global minimuma sahiptirler. Bu durum ağ yapılarının doğasında var olan simetrik yapılardan kaynaklanmaktadır. Ancak burada bahsedilen global minimumlar yerel minimum değerleri gibi ciddi problemlere yol açmazlar.

- Hatalı Minimum Değer Saptamak. Yapay sinir ağlarında, öğrenmede, yerel minimum problemi ile ilişkili bir diğer problem de hatalı saptanan minimum değerlerdir. Burada problem, ister yerel minimum olsun isterse global minimum, minimum bir noktaya ulaşıldığına karar verilirken bakılan eğim değerinin, sıfıra ne kadar yakın olması gerektiğinin saptanmasıdır. Ağ yapılarında elde edilen hata fonksiyonları neredeyse düz olan geniş yüzeylere sahip olabilirler ve bu yüzeyler aşağıya doğru eğimlerle sonuçlanabilirler. Dolayısıyla temel geri-yayılım eğitim algoritması, eğim değeri çok küçük olduğundan, yanlışlıkla minimum değere ulaşıldığını söyleyebilir. Bu nedenle, küçük eğim değerlerine bakarak yakınsamanın gerçekleştiği asla kabul edilmemelidir.

Ağ yapılarından elde edilen hata fonksiyonları eğim teriminin çok küçük değerler gösterdiği geniş yüzeylere sahip olabildiğinden bu yapılardaki sayısal değişkenlik yanlış yorumlara yol açabilmektedir. Kullanılan eğitim algoritmalarının etkinliğini arttırmaya ve uygulama sırasında yaşanan problemleri çözmeye yönelik en basit ve de en etkin çözüm, mümkün olduğu kadar çok sayıda ve farklı yapıda ele alınan eğitim algoritmasını tekrarlanmaktır.

4. YAPAY SİNİR AĞLARINDA GÖZETİMSİZ EĞİTİM VE KOHONEN AĞ YAPILARI

4.1 Yapay Sinir Ağlarında Gözetimsiz Eğitim

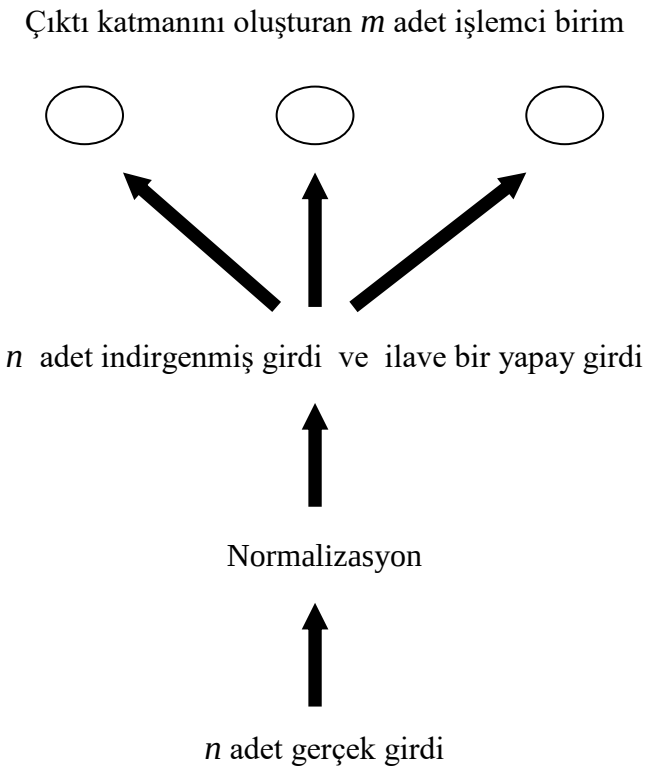
Günümüzde, Yapay Sinir Ağları uygulamalarının büyük bir çoğunluğu Gözetimli eğitim (*supervised training*) algoritmalarını kullanmaktadır. Söz konusu eğitim algoritmalarında, eğitim setindeki her bir girdi seti için istenen çıktı değerleri de aynı zamanda bilinmektedir. Eğitim süreçlerinde, eğitim setindeki girdi-çıktı ikilileri ağ yapısına tanıtılır ve ağ yapısından bunlar arasındaki ilişkiyi öğrenmesi istenir. Gözetimli eğitim algoritmalarında daha çok, ağ yapısının kendisine sunulan veri setinden çıkardığı, öğrendiği örüntülerle (*pattern*) ilgilenilir. Bu algoritmalar kullanılarak eğitilen ağ yapılarının, görevlerini yerine getirirken kullandığı yöntemlerin anlaşılması oldukça zordur. Gözetimsiz eğitim (*unsupervised training*) algoritmaları, yukarıda sayılan iki konuda Gözetimli eğitim algoritmalarından farklılık gösterirler:

- Bu algoritmaların kullanılacağı eğitim süreçlerinde, eğitim setindeki her bir girdi seti için çıktı değerlerinin de bilinmesine gerek yoktur.
- Bu algoritmalar kullanılarak eğitilen ağ yapılarının görevlerini yerine getirirken kullandıkları yöntemler daha ulaşılabilirdir.

Gözetimsiz eğitim algoritmaları ile eğitilen sinir ağlarının en ünlüsü, adını kendisini bulan kişiden alan (Teuvo Kohonen), Kohonen ağ yapısıdır. Kohonen ağ yapısında, Rekabete dayalı eğitim (*competitive learning*) adı verilen bir çeşit eğitim süreci söz konusudur. Diğer bir çok eğitim sürecinde girdi setlerinin ağ yapısına tanıtımının ardından, ağ yapısındaki tüm işlemci birimler için ağırlık değerleri güncellenir. Ancak Rekabete dayalı eğitim sürecinde, ağ yapısındaki işlemci birimler ağırlık değerlerini güncelleyebilmek için yarışırlar ve bu yarışın sonucunda sadece bir ya da birbirine komşu birkaç işlemci birimin ağırlık değerlerini değiştirmesine izin verilir.

4.2 Kohonen Ağ Yapıları

Kohonen ağ yapıları, aslında iki katmanlı ve ileri sürümlü bir sinir ağı yapısıdır. Ancak, ağ yapısına tanıtılacak girdi setlerinin normalize edilmesi gerektiğinden, girdi katmanını takip eden bir normalizasyon katmanı dolayısıyla, üç katmanlı bir ağ yapısı şeklinde de görülebilir. Burada, normalizasyon işlemi girdi katmanında gerçekleştirilen bir ön işlem olarak ele alınacak ve Kohonen ağ yapısında katman sayısı iki olarak kabul edilecektir. Kohonen sinir ağı modelinin temel yapısı aşağıdaki şekilde çizilebilir:



Şekil 4.1 Kohonen sinir ağı modelinin temel yapısı (*Masters, 1993; s.329*).

Bazı girdi setlerinin normalizasyon işlemi sonucunda, "Yapay girdi" adı verilen ilave bir bileşen üretilebilmektedir. Bu nedenle, ağ yapısına tanıtılacak bir girdi setinin n bileşenden oluşması durumunda, ağ yapısı girdi katmanındaki işlemci birim sayısı $n+1$ olmaktadır. Burada, söz konusu yapay girdiye bağlı ağ yapısının

üreteceği aktivasyon, diğer gerçek girdilerin bir fonksiyonudur ve normalizasyon işlemi sonucunda ilave bir bileşen oluşmaması durumunda sıfır olarak ele alınır.

Kohonen ağ yapısında, girdi katmanındaki (n+1) adet işlemci birim, çıktı katmanında, Kohonen hücresi adı verilen m adet işlemci birimle bağlantı kurar. Çıktı katmanındaki işlemci birimler son derece basit bir mantıkla işlem görürler. Bunların çıktı değerleri, net girdi değerlerine eşittir. Diğer bir ifade ile Kohonen hücrelerinin çıktı değeri girdilerinin ağırlıklı toplamına eşittir. Bu noktada elde edilen net girdi değerinin uygulandığı bir aktivasyon fonksiyonu söz konusu değildir. Dolayısıyla, bir Kohonen hücresinin çıktı değeri aşağıdaki şekilde formüle edilebilir: Aşağıdaki formülde, s yapay girdi bileşenini temsil etmektedir.

$$\text{Çıktı} = \sum_{i=0}^{n-1} x_i w_i + s\, w_n \qquad (4.1)$$

Kohonen ağ yapıları, temel olarak kendisine tanıtılan veri setlerini sınıflara ayırmak amacıyla kullanılmaktadır. Eğitim sürecinin ardından elde edilen ağırlık değerleri kullanılarak yukarıdaki formül yardımıyla, ağ yapısına tanıtılan bir girdi seti için Kohonen hücrelerinin çıktı değerleri hesaplanır. Bunlar arasında en büyük çıktı değerine sahip olan kazanan olarak ilan edilir. Bu şekilde, aynı zamanda tanıtılan girdi setinin ait olduğu sınıf da belirlenmiş olmaktadır.

Dikkat edilecek olursa, Kohonen sinir ağı, çok katmanlı ve ileri sürümlü sinir ağlarına nazaran oldukça zayıf bir sınıflandırıcı olarak gözükmektedir. Ağ yapısında saklı bir katman yoktur ve dolayısıyla girdi-çıktı ikilileri arasında lineer bir ilişki kurulmaktadır. Bu nedenle, pratik uygulanabilirliğe sahip Kohonen ağ yapıları, çıktı katmanında çok sayıda işlemci birime sahiptirler ki bu işlemci birimlerden her biri tek bir özellik üzerine yoğunlaşmıştır. Ancak, Kohonen ağ yapılarının kullanışlılığı, temel olarak oldukça hızlı bir biçimde eğitilebilmesine ve de ürettiği sonuçların kolay yorumlanabilmesine dayanmaktadır. Ayrıca, biyolojik ağ modellerine görünüş olarak benzerliği de bunda büyük bir rol oynamaktadır.

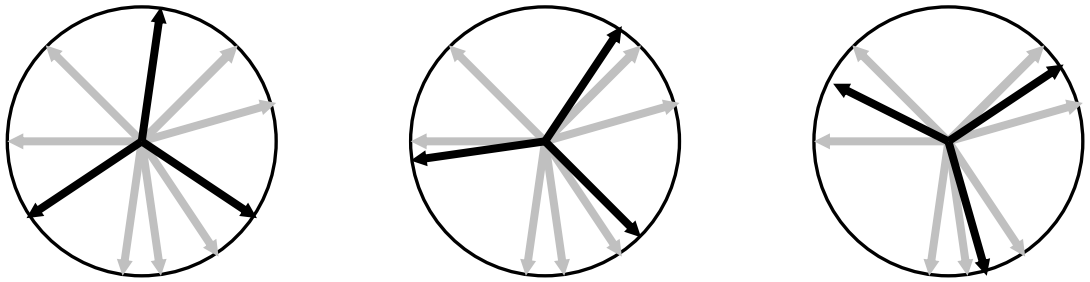
4.3 Kohonen Ağ Yapılarının Eğitimi

Kohonen ağ yapıları, diğer bir çok ağ yapısından farklı bir mantıkla eğitilirler. Rekabete dayalı öğrenme (*competitive learning*) adı verilen bu farklı eğitim sürecinde, her bir girdi tanıtımında çıktı katmanındaki işlemci birimler adeta birbirleri ile yarışırlar. Yarış sonucunda sadece kazanan işlemci birim öğrenmeye hak kazanır.

Buradaki yarışın mantığı son derece basittir. Çıktı katmanındaki her bir işlemci birim indirgenmiş girdi vektörleri ile bağlantı kurmuş durumdadır. Bu bağlantıların ağırlık değerleri de tıpkı girdi değerleri gibi birim uzunluğa indirgenirler. Her bir Kohonen hücresinin çıktı değeri, girdi ve ağırlık vektörlerinin çarpılıp toplanması sonucu elde edilen net girdi değerine eşittir. Yarışta, ağ yapısına tanıtılan her bir girdi setine karşılık en büyük çıktı değerine sahip olan işlemci birimler kazanan olarak ilan edilirler.

Yarışmanın sonucunda kazanan işlemci birime ait ağırlık vektörleri öyle şekilde güncellenir ki, ağ yapısı benzer bir girdi tanıtımına daha güçlü ve kararlı bir şekilde karşılık verir. Bu şekilde kazanan işlemci birimlerin pozisyonları da güçlendirilmiş olur. Kohonen ağ yapısında, eğitim sürecinin ardından her şeyin yolunda gitmesi durumunda, çıktı katmanındaki işlemci birimlere ait ağırlık vektörleri sabit pozisyonlara yakınsarlar ve yeni girdi tanıtımlarının bu pozisyonlar üzerine etkisi göz ardı edilebilecek derecede azalır. Ağ yapısının test sürecinin ardından, ağ yapısına tanıtılan herhangi bir verinin sınıflandırılması, çıktı katmanında elde edilen en büyük çıktı değerine sahip işlemci birim bulunarak gerçekleştirilebilir.

Yukarıda ele alınan Rekabete dayalı eğitim süreci, girdi vektörlerinin iki boyutlu olduğu durumlar için, aşağıdaki şekilde görsel hale getirilebilir. Burada girdi ve ağırlık vektörlerinin birim uzunluğa indirgenmesi ile kastedilen, bu vektörlerin birim bir daire içinde bulunmalarını sağlamaktır:



Şekil 4.2 Rekabete dayalı eğitim süreci aşamaları (*Masters, 1993; s.333*).

Yukarıdaki şekiller, Kohonen ağ yapısında tipik bir eğitim sürecinin ardışık üç aşamasını göstermektedir. Şekillerden de anlaşılabileceği gibi, ele alınan ağ yapısının çıktı katmanında üç adet işlemci birim bulunmaktadır ve bunlara ilişkin ağırlık değerleri koyu renkli vektörler ile gösterilmektedir. Açık renkli vektörler ise eğitim setindeki veri vektörlerini simgelemektedir. En soldaki şekilde eğitim sürecinin başlangıcı ele alınmaktadır ve bu nedenle ağırlık vektörleri rasgele konumlanmış durumdadır. Ortadaki şekilde, eğitim süreci ilerledikçe ağırlık vektörlerinin verilerin doğal kümelenmelerine bağlı olarak pozisyon almaya başladığı görülmektedir ki nihai pozisyonları en sağdaki şekilde gösterilmektedir. Dikkat edilirse her bir ağırlık vektörü, temsil ettiği kümenin merkezine yakın bir noktada konumlanmıştır.

4.4 Ağırlık Değerlerinin Güncellenmesi

Kohonen ağ yapılarında, bir eğitim setinin ağ yapısına tanıtılmasının ardından işlemci birimlerin ağırlık vektörlerinin güncellenmesi amacıyla kullanılan iki popüler yöntem söz konusudur. Teuvo Kohonen tarafından ileri sürülen "Toplamalı yöntem"de (*additive method*), girdi vektörünün bir bölümü ağırlık vektörüne eklenmekte, ardından elde edilen toplam yeniden normalize edilerek birim uzunluğa indirgenmektedir. Bu sayede ağırlık vektörü yavaş yavaş veri vektörüne doğru itilmektedir. x, ağ yapısına tanıtılan girdi vektörünü ve w^t, t anında kazanan işlemci birime ait ağırlık vektörünü göstermek üzere;

$$w^{t+1} = \frac{w^t + \alpha x}{\| w^t + \alpha x \|} \qquad (4.2)$$

Yukarıdaki formülde α öğrenme katsayısını, paydadaki işlem ise hesaplanan vektör uzunluğunu göstermektedir. Buradaki uzunluk hesabında farklı yöntemler kullanılabilir. En yaygın kullanılanı öklit uzunluğudur.

Yukarıda tanımlanan ve genellikle iyi sonuçlar üreten ilk yöntemin bazı uygulamalarda aşırı değişkenlik gösterebilmesinden dolayı, alternatif ikinci bir yöntem olarak "Çıkarmalı yöntem"i (*subtractive method*) geliştirilmiştir. Aslında, bu ikinci yöntem temel "Widrow-Hoff" öğrenme algoritması olarak görülebilir. Bu yöntem, bir ağırlık vektörünü bir veri vektörüne yaklaştırmak maksadıyla bu vektörler arasındaki farkın hesaplanmasına dayanmaktadır. Ardından bu fark vektörünün bir bölümü ağırlık vektörüne eklenerek ağırlık vektörünün veri vektörüne yakınsaması amaçlanır. x, ağ yapısına tanıtılan girdi vektörünü ve w^t, t anında kazanan işlemci birime ait ağırlık vektörünü göstermek üzere;

$$\begin{aligned} e &= x - w^t \\ w^{t+1} &= w^t + \alpha e \end{aligned} \qquad (4.3)$$

Kohonen ağ yapılarının eğitim süreçlerinde bir çok araştırmacı, eğitim setindeki her bir girdi tanıtımının hemen ardından, ağ yapısındaki ağırlık değerlerini güncelleme eğilimindedir. Ancak bu durum, her bir girdi tanıtımı sonucunda, ağırlık vektörlerinin pozisyonlarında sürekli salınımlar meydana getirdiği için tavsiye edilmemektedir. Onun yerine eğitim setinin tamamının ağ yapısına tanıtılmasının ardından toplu bir değerlendirme yapmak daha mantıklı görülmekte ve tavsiye edilmektedir.

Kohonen ağ yapılarının eğitim süreçlerinde karşılaşılan diğer bir problem ise, tıpkı girdi vektörlerinde olduğu gibi normalize edilerek birim uzunluk

vektörlerine indirgenmesi gereken ağırlık vektörlerinin her bir düzeltmenin hemen ardından yeniden normalize edilmesi isteğidir. Bu da tavsiye edilmeyen bir davranıştır. Öncelikle gereksizdir. Çünkü, her ne kadar normalizasyon, yapılan ilk güncelleme ile birlikte kaybolsa da, girdi vektörleri indirgenmiş olduğundan dolayı ağırlık vektörleri de yaklaşık olarak indirgenmiş kalır. Ayrıca, eğitim sürecinde yapılan her yeni normalizasyon işlemi sonucunda yakınsama kaybedilebilmektedir. Dolayısıyla ağırlık vektörlerinin birim uzunluğa indirgenmesi aynı zamanda yakınsamaya da engel olabilmektedir. Üstelik, eğitim sürecinin ardından elde edilen ağırlık vektörlerinin birim uzunluğa indirgenip indirgenmeyeceği hakkında da ortak bir karar söz konusu değildir. Elde edilen ağırlık vektörleri aynen bırakılabilecekleri gibi, normalize edilerek ağ yapısının genelleme yapma yeteneğinin geliştirilebileceği düşünülebilir. Fakat, bu iki durum arasında son derece ufak bir fark söz konusu olduğundan kesin bir sonuç ortaya konulamamıştır.

4.5 Öğrenme Katsayısı

Kohonen ağ yapılarının eğitim süreçlerinde kullanılan, yukarıdaki formüllerdeki α sabiti öğrenme katsayısı olarak adlandırılır. Bu sabit, her zaman için 0 ile 1 arasında bir değer alır. Ancak en iyi sonuçlar, eğitim süreci ilerledikçe değerinin yavaş yavaş azaltılması ile elde edilir.

Öğrenme katsayısının büyük seçilmesi bir dereceye kadar eğitim sürecini hızlandırır. Ancak değerinin gereğinden büyük seçilmesi durumunda, eğitim süresince, ağırlık vektörlerinin bir sarkaç gibi salınması ve buna bağlı olarak yakınsamanın gerçekleşmemesi ihtimali söz konusudur. Öğrenme katsayısının değerinin uygun seçilip seçilmediğini belirlemenin en iyi yollarından biri, maksimum ortalama düzeltme vektörünün uzunluğunu gözlemektir. Bu vektörün uzunluğunun hızlı ve düzenli bir biçimde azalması gerekir. Bu değerin yavaş bir biçimde azalması öğrenme katsayısının küçük seçildiğini gösterirken sürekli bir biçimde ya da arada sırada yükselmesi durumunda seçilen değerin azaltılması gerekir.

Yukarıda yapılan saptamalar yalnızca, eğitim setinin tamamının ya da bir bölümünün ağ yapısına tanıtımının ardından, ağırlık vektörlerinde ortalama düzeltmenin yapıldığı durumlar için geçerlidir. Bu saptamalar, her bir girdi tanıtımının ardından ağırlık değerlerinin güncellendiği durumlar için önem taşımamaktadır.

4.6 Ağ Yapısı Hata Değerinin Ölçülmesi

Kohonen ağ yapıları için, ağ yapısı hata değerinin belirlenmesi amacıyla kullanılabilecek herhangi bir resmi yöntem bulunmamaktadır. Bu durum bir ölçüde eğitim sürecinin denetlenemez yapısından ve de doğru ya da yanlış cevap tanımlamalarının bulunmamasından kaynaklanmaktadır. Ayrıca, Kohonen ağ yapıları için kullanılan eğitim algoritmalarını tamamlayacak herhangi bir yöntem de söz konusu değildir. Hatırlanırsa, hata değerlerinin kareleri ortalaması eğim kökenli eğitim algoritmalarında tamamlayıcı bir rol üstlenmekteydi. Bununla birlikte, ağ yapısı hata değerini belirlemek amacıyla kullanılabilecek bazı numaralar da bulunmaktaydı.

Kohonen ağ yapılarının eğitim süreçlerinde nihai hedef, her bir Kohonen hücresine ait ağırlık vektörünün bir şekilde, veri kümelenmelerinin doğal eğilimini göstermesidir. Dolayısıyla, eğitim setindeki vektörler ile onlara en yakın ağırlık vektörleri arasındaki düzeltme vektörlerini incelemek, ağ yapısı hata değerinin belirlenmesi için yeterli olacaktır. Elde edilen ortalama düzeltme vektörü uzunluğu, elbette ki ağ yapısının performansı hakkında ipuçları verecektir. Ancak alternatif bir hata değeri göstergesini daha dikkate almayı gerektiren sebepler vardır.

Mükemmel bir hata değeri ölçütünün, ağ yapısının performansının geliştirilip geliştirilemeyeceği hakkında bilgi vermesi gerekir. Ortalama düzeltme vektörü uzunluğu böyle bir bilgi verememektedir. Eğitim setindeki verilerin büyük bir çoğunluğunun, en yakın ağırlık vektörüne makul bir uzaklıkta olması durumunda, makul bir ağ yapısı hata değeri elde edilecektir. Yine, eğitim setindeki verilerin bir

bölümünün ilişkili oldukları ağırlık vektörüne çok yakın, diğer bölümünün ise çok uzakta olması durumunda da makul bir ağ yapısı hata değeri elde edilecektir. Ancak ilk durumda ağ yapısına işlemci birim eklemek işe yaramazken ikinci durumda, uzakta kalan veriler daha iyi temsil edilebileceğinden, ağ yapısının performansı yükselecektir.

Böyle bir durumda, eğitim setindeki veriler için maksimum düzeltme vektörünün uzunluğunu araştırmak gerekir. Büyük bir düzeltme vektörü ağ yapısına işlemci birim eklenmesi gerektiğini gösterecektir. Bu nedenle ağ yapıları için ortalama hata değerinin yanında maksimum hata değerinin de göz önünde tutulması gerekir.

4.7 Eğitim Sürecinde Yakınsamanın Belirlenmesi

Kohonen ağ yapılarında, eğitim sürecinin ne zaman sona erdirileceği kararı nasıl verilir? Aslında, söz konusu sorunun cevabı oldukça basittir. Eğitim sürecine, ağırlık vektörlerinin pozisyonlarında gözlemlenen değişimlerin önemsiz hale geldiği noktada son verilebilir. Ağ yapısında, çıktı katmanındaki her bir işlemci birim için elde edilen düzeltme vektörlerinin en büyüğünün uzunluğuna bakılarak eğitim sürecinde yakınsamanın tamamlandığı sonucuna varılabilir.

Ancak, girdi setindeki her bir verinin ağ yapısına tanıtımının ardından düzeltme yapılması durumunda, söz konusu karar yukarıda anlatıldığı kadar basit olmamaktadır. Böyle bir durumda, eğitim sürecinde yakınsama tamamlanmış olsa bile, ağ yapısına tanıtılacak ardışık iki veri seti, ağırlık vektörlerini ters taraflara çekebilmektedir. Öğrenme katsayısının küçük seçilmesi bu salınımı hafifletse bile aynı zamanda yakınsamanın kaybolduğunu gizleyebilmektedir. Burada düzeltme vektörlerine bakılarak yakınsama kararının verilmesi oldukça zor bir iştir. Bu durum, tüm eğitim setinin ağ yapısına tanıtılmasının ardından düzeltme yapılması tavsiyesinin sebeplerinden biridir.

Dikkat edilirse ağ yapısı hata değerinin, eğitim sürecinde yakınsamaya ilişkin bir gösterge olarak neredeyse hiçbir değeri yoktur. Ağ yapısı hata değerinin nihai değerine doğru nasıl gelişeceği hakkında bir bilgi olmadığından dolayı sabitleşmesine bakılarak karar verilmesi mümkün değildir. Ağ yapısı hata değerinin bir çok iterasyon boyunca sabit kalmasına karşın ağırlık vektörlerinin önemli miktarlarda yer değiştirmesi mümkündür. Hatta, eğitim sürecinin gelişmesi karşısında ağ yapısı hata değerinin yükselmesi bile mümkündür. Bu nedenle, eğitim süreçlerinde yakınsama kararının alınmasında ağ yapısı hata değerinin göz ardı edilmesi gerekir.

Eğitim süreçlerinde küçük öğrenme katsayılarının kullanıldığı durumlarda yakınsama kararının alınmasına ayrıca dikkat edilmesi gerekir. Öğrenme katsayısının küçük seçilmesinden dolayı, yakınsamanın çok uzağında olunmasına rağmen ağırlık vektörlerinin düzeltilme miktarları küçülebilir. Buradaki probleme etkin bir çözüm, maksimum düzeltme vektörü uzunluğunun öğrenme katsayısına bölünerek elde edilecek oranın yakınsama göstergesi olarak kullanılmasıdır. Bu yaklaşım gerek Toplamalı (*additive*) gerekse Çıkarmalı (*subtractive*) öğrenme formülleri için de oldukça etkin sonuçlar üretmektedir.

4.8 Öğrenmeyi Reddeden İşlemci Birimler

Kohonen ağ yapılarının eğitim süreçlerinde, sadece kazanan işlemci birimler ağırlık vektörlerini güncellemekte ve yapılan düzeltmeler, eğitimin bir sonraki aşamasında söz konusu birimlerin kazanma olasılığını arttıracak şekilde gerçekleşmektedir. Dolayısıyla, ağ yapısında bir ya da daha fazla işlemci birimin eğitim süreci boyunca hiç kazanamaması söz konusudur. İşlem süresini uzatan ve ilave bellek gerektiren bu ölü işlemci birimler, eğitim setini temsil yükünü üstlenmemekte hatta diğer, öğrenebilen işlemci birimlerin de aşırı yüklenmesine neden olmaktadır. Bu nedenle, aşırı yüklenen işlemci birimlerin bir kısım temsil özelliklerinin bu ölü işlemci birimlere transfer edilmesi gerekir.

Bu işlemi yerine getirmek için çok çeşitli yöntemler söz konusudur. Ancak, bu yöntemlerin hiç birisi üzerinde görüş birliği yoktur. Bu alandaki en popüler yöntemlerden bir bölümü "Conscience" adı verilen bir mekanizma söz konusudur. Bu mekanizmaya göre, her bir işlemci birimin eğitim sürecinde kaç kez kazandığı hakkında bir çetele tutulur. Bu işlemde amaç, her bir işlemci birimin eşit sayıda kazanmasını sağlamaktır. Gereğinden fazla kazanan işlemci birimler için bir ceza puanı geliştirilerek söz konusu işlemci birimlerin aktivasyon seviyeleri ayarlanır.

Ancak pratikte rastlanan problemler için daha kullanışlı bir yöntem daha söz konusudur. Bu yeni yöntem, yukarıda anlatılan yöntemler gibi kazanma sayısında eşitliği zorunlu tutmamakta, yerine işlemci birimler arasında uzmanlaşmanın optimizasyonuna çalışmaktadır. Bu yöntemde, aşırı yüklenmiş işlemci birimlerden ölü işlemci birimlere yapılan transfer işlemi, tüm işlemci birimlerin öğrenmesi sağlandıktan hemen sonra sonlandırılır. Bu nedenle, ağ yapısındaki bir kısım işlemci birimlerin, eğitim setinin küçük bir bölümünü temsil etmesi söz konusu olabilir. Bazı durumlarda problem olan bu durum, pratik alanlardaki uygulamaların büyük bir çoğunluğu için artan uzmanlaşma düzeyinden sebep avantaj sağlayabilmektedir.

Ağ yapısında öğrenmeyi reddeden ölü bir işlemci birime, eğitim setindeki verilerin bir kısmını temsil etme yeteneğinin verilmesi süreci üç aşamadan oluşmaktadır:

- İlk aşamada, eğitim setindeki her bir veri için ağ yapısında kazanan işlemci birimler saptanır. Bunlar arasında, en küçük çıktı değeri ile kazanan işlemci birime ait veri seti ağ yapısı tarafından en az temsil edilen veri seti olacaktır. Dolayısıyla, bu veri vektörü için ayrı bir ağırlık vektörü tahsis etmek daha mantıklı olacaktır.
- İkinci adımda, ilk adımda saptanan veri vektörünü temsil edecek ağ yapısındaki ölü işlemci birimler araştırılır. Bu ölü işlemci birimler arasında maksimum çıktı değerine sahip işlemci birim söz konusu veri setini temsil etmek üzere seçilir.

- Son adımda, seçilen işlemci birimin ağırlık vektörünün belirlenmesi amacıyla ilk adımda saptanan veri vektörü kullanılır. Bu amaçla tüm yapılması gereken yapay girdi bileşenini de içeren indirgenmiş girdi vektörünü aynen kopyalamaktır.

4.9 Öz-düzenlemeli Haritalar

Yapay Sinir Ağlarında, ağırlık vektörlerinin kendi kendini organize ettiği, Öz-düzenlemeli haritalar (*Self Organizing Maps*) adı verilen sinir ağı yapıları, Teuvo Kohonen'in orijinal çalışmasının temel bölümlerinden birini oluşturmaktadır.

Ağ yapılarında kendi kendine, otomatik organizasyonun dayandığı temel prensip, ağ yapısına tanıtılan bir bilginin sadece tek bir Kohonen hücresi üzerinde toplanması yerine, çıktı katmanındaki belirli bir bölgeye yayılmasıdır. Bu prensibe göre, bir Kohonen hücresinin verilen bir özelliği temsil etmesi durumunda, bu hücreye komşu diğer hücrelerin de aynı özelliği temsil etmesi gerekir. Bu ise, ağ yapısına yapılan her bir veri tanıtımda, birden fazla Kohonen hücresinin öğrenmesine izin verilmesi ile sağlanabilir. Öz-düzenlemeli haritaların öğrenme süreçlerindeki bu durum, aşağıdaki şekilde ifade edilebilir:

$$w_i(t+1) = w_i(t) + \alpha(t)\,[\,x(t) - w_i(t)\,], \qquad i \in N_c(t)$$

$$w_i(t+1) = w_i(t), \qquad \textit{aksi halde}$$

Yukarıdaki ifadelerde, t kesikli zaman indeksini, $\alpha(t) \in [0,1]$ öğrenme katsayısını ve $N_c(t)$ çıktı katmanındaki kazanan Kohonen işlemci birimine komşu işlemci birimler kümesini göstermektedir. Ayrıca burada, t zaman indeksine bağlı olan $N_c(t)$ komşu işlemci birim kümesi ve $\alpha(t)$ öğrenme katsayısı parametreleri, öğrenme sürecinin başlangıcında oldukça büyük iken süreç ilerledikçe gitgide küçülmektedirler.

Öz-düzenlemeli haritalar yaklaşımında izlenen algoritma şu şekilde tanımlanabilir. Öncelikle, ağ yapısındaki ağırlık vektörlerine gelişigüzel küçük değerler atanarak oluşabilecek herhangi bir simetrik yapının önüne geçilir. Ardından, her zamanki gibi, bir girdi seti ağ yapısına tanıtılarak kazanan Kohonen hücresi saptanır ve söz konusu Kohonen hücresinin ağırlık vektörü belirli bir öğrenme katsayısı ile güncellenir. İlave olarak, kazanan Kohonen hücresinin komşusu olan hücrelerin de öğrenmesine izin verilir. Ancak, bu hücrelerin öğrenmelerinde kullanılacak öğrenme katsayısı, kazanan hücre için kullanılandan daha küçük seçilir. Dolayısıyla komşu hücrelerin ağırlık vektörlerindeki düzeltme miktarı daha az

gerçekleştirilmiş olur. Yine de, söz konusu hücrelerin ağırlık vektörleri de, ağ yapısına tanıtılan girdi vektörüne doğru yakınsar. Böylelikle ağ yapısına tanıtılan bilgi, çıktı katmanında kazanan Kohonen hücresinin çevresindeki bir bölgede toplanmış olur.

Öz-düzenlemeli haritalar yaklaşımının en basit uygulamalarında, çıktı katmanındaki işlemci birimlerin doğrusal bir biçimde sıralandığı tek boyutlu bir yerleşim kullanılmaktadır. Ancak iki boyutlu yerleşimler en yaygın kullanılanlarıdır. Bu tip yerleşimlerde, çıktı katmanındaki işlemci birimler adeta dikdörtgen bir ızgara üzerine dizilirler ki bu durumda kazanan hücrelerin dört bir yanında komşuları bulunur. Yine, daha büyük boyutlarda yerleşimleri kullanan uygulamalar da söz konusudur.

Öz-düzenlemeli haritalar yaklaşımı ile ilgili çalışmalarda, genellikle kazanan hücreye en yakın hücreler öğrenmeye dahil edilirler. Çok daha karmaşık olan ve en yakın birkaç komşu hücreyi daha ele alan algoritmalar da söz konusudur. Ancak bu durumda kullanılan öğrenme katsayısının kazanan hücreye uzaklıkla orantılı olacak şekilde süratle düşürülmesi gerekir. Hatta, bazı karmaşık algoritmalarda, birden fazla komşu hücrenin öğrenmesine izin verilmekte ilaveten kazanan hücreye uzak kalan hücrelerin ağırlık değerleri de, tanıtılan girdi setine ters yönde olacak

şekilde, çok küçük öğrenme katsayıları kullanılarak güncellenmektedir. Böylelikle bilginin belirli bir bölgede toplanma etkinliği arttırılmaktadır.

Öz-düzenlemeli haritalar yaklaşımı uygulamalarında, ağ yapılarının eğitim setindeki özellikleri öğrenme şekilleri son derece büyüleyicidir. Ancak, eğitim sürecine ilave edilen her bir ağırlık güncelleme adımı öğrenmeyi gitgide yavaşlatmaktadır. Daha da kötüsü, komşularının da öğrenmesine izin verilen kazanan işlemci birimlerin uzmanlaşma dereceleri düşmektedir. Bu nedenle, belirli bir performans değeri için ağ yapısında genellikle, daha çok sayıda işlemci birim kullanımına gerek duyulmaktadır. Özellikle bu iki durum, Öz-düzenlemeli haritaların kullanımını birçok durumda imkansızlaştırabilmektedir.

Öz-düzenlemeli haritalar yaklaşımı çok boyutlu, karmaşık veri setleri içindeki doğal ilişkileri rahatlıkla ortaya çıkarabilmekte ve bu özellikleri sayesinde çok boyutlu veri setlerine ilişkin sıralama, kümeleme, görüntüleme vb. problemlerin çözümlerine yönelik kullanılabilmektedirler. Benzer şekilde, sonlu sayıda olurlu çözüme sahip optimizasyon problemlerine ait teori ve tekniklerden ibaret olan, "Kombinatoryal optimizasyon" problemlerinin çözümüne yönelik geliştirilen Öz-düzenlemeli harita uygulamalarına da rastlamak mümkündür. Burada ifade edilen kombinatoryal optimizasyon, kesikli matematiksel yapılar üzerinde hesaplama yapma problemini ele almakta ve bunu yaparken bir probleme ait herhangi bir çözümün değerinin sayısal olarak ölçülebileceği ve bu değerin, problemin diğer çözümlerinin değerleri ile karşılaştırılabileceği varsayımlarını dikkate almaktadır.

5. GEZGİN SATICI PROBLEMİ

5.1 Gezgin Satıcı Probleminin Temel Özellikleri

Gezgin Satıcı Problemi, kombinatoryal optimizasyonun en temel zor-problemlerinden bir tanesidir ve optimizasyon problemlerinde çok değişkenli fonksiyonların en küçüklenmesine ya da en büyüklenmesine yönelik etkin yöntemlerin araştırılması ile ilişkilidir. Söz konusu problemi çözmek amacıyla geliştirilen algoritmalar ve sezgisel yaklaşımlar, aşağıdaki soruya cevap üretmeye çalışmaktadırlar:

> *n* adet şehir seti ve her bir şehir çifti arasındaki mesafelerin verildiği bir durumda, her bir şehrin sadece bir kez ziyaret edildiği ve tekrar başlangıç noktasına dönüldüğü en kısa tur nasıl oluşturulabilir?

Verilen bir şehir seti için bir satıcının, toplam mesafeyi minimize edecek şekilde, tüm şehirleri ziyaret ederek tekrar başladığı noktaya döneceği en uygun rotanın bulunmasını isteyen Gezgin Satıcı Problemi, bir tamsayılı programlama modeli olarak aşağıdaki şekilde formüle edilebilir (*Johnson ve McGeoch, 1997; s.86*):

$$\textbf{Min} \quad \sum_{i=1}^{N} \sum_{j=1}^{N} d(c_i, c_j) * \left[x_{i,N}\, x_{j,1} + \sum_{k=1}^{N-1} x_{i,k}\, x_{j,k+1} \right]$$

$$\textbf{Subject to} \quad \sum_{i=1}^{N} x_{i,k} = 1, \qquad 1 \le k \le N$$

$$\textbf{and} \quad \sum_{k=1}^{N} x_{i,k} = 1, \qquad 1 \le i \le N \qquad (5.1)$$

$$\textbf{and} \quad x_{i,k} \in \{0,1\}, \quad 1 \le i,k \le N$$

Yukarıda ele alınan modelde $x_{ik} = 1$ eşitliği, c_i şehrinin turda k. sırada ziyaret edilen şehir olduğunu göstermektedir. Yine modeldeki ilk kısıt turdaki her bir pozisyonda sadece bir şehir bulunmasını söylerken ikinci kısıt ise her bir şehrin sadece tek bir pozisyonda yer almasını söylemektedir.

5.2 Gezgin Satıcı Problemine Çözüm Yaklaşımları

Gezgin Satıcı Problemi, şehir sayısının (n) artması ile birlikte, olası tur sayısının ($n!/2n$) şehir sayısına bağlı olarak artış göstermesi nedeniyle kolaylıkla çözülemez hale gelebilmektedir. Gezgin Satıcı Problemi bir "NP-complete" optimizasyon problemidir. Bu ifade, makul büyüklükte bir problem için, ele alınması gereken çok sayıda varsayımın varlığından dolayı, optimum bir çözüm aramanın, (hesaplama maliyetlerinin çok yüksek olması nedeniyle) mümkün olmadığı problemler için kullanılmaktadır.

"NP-complete" optimizasyon problemlerinde işlem zamanının hızlı bir biçimde artış göstermesi ve bir çok uygulamada, hesaplama sürelerinin uzunluğundan dolayı, problemin çözümüne ulaşmanın mümkün olmadığı durumlar ile karşılaşılması, araştırmacıları şu iki alternatif yoldan birini izlemek durumunda bırakmaktadır:

1. Hızlı bir biçimde, optimum çözüme yakın çözümler üretecek sezgisel yöntemler aramak;
2. Genele yönelik olmayan ancak pratik alanlardaki özel durumlar için iyi sonuçlar üretecek algoritmalar geliştirmek.

Gezgin Satıcı Problemi, basit yapısından ve kolay uygulanabilirliğinden dolayı (ve belki de ilgi uyandıran isminden dolayı), yukarıdaki alternatif yollarla ilgili geliştirilen yeni fikirler için, kendilerini ispatlamaya çalıştıkları ilk çalışma alanlarından biri olarak onlarca yıldır hizmet vermektedir ve rahatlıkla söylenebilir ki Gezgin Satıcı Problemi bu bakımdan, optimizasyona dair

büyük başarı hikayelerinden bir tanesidir. Optimizasyon teknikleri üzerinde yapılan araştırmaların, bilgisayarların işlem hızında ve bellek kapasitelerindeki sürekli artışla birleşmesi sonucunda, optimum çözüm üretilen Gezgin Satıcı Problemlerinde şehir sayısı 318 şehirden (*Crowder ve Padberg, 1980*) 2.392 şehire (*Padberg ve Rinaldi, 1987*) ve ardından 7.397 şehire (*Applegate ve diğerleri, 1994*) yükselmiştir. Her ne kadar buradaki rekorlara ulaşılırken yıllarla ölçülebilecek işlem süreleri gerekmekte ise de günümüzde 1.000 şehir civarındaki Gezgin Satıcı Problemlerine saatlerle ifade edilebilecek işlem süreleri içinde optimum çözümler üretilebilmektedir.

5.2.1 Klasik Sezgisel Çözüm Yaklaşımları

Gezgin Satıcı Problemini diğer kombinatoryal optimizasyon problemlerinden farklı kılan optimum çözüm üretmedeki bu başarılar aynı şekilde, geleneksel sezgisel yaklaşımların söz konusu probleme ürettikleri sonuçların yüksek kalitesinde de kendisini göstermektedir. "Tur oluşturma" (*tour construction*) sezgiselleri adını verebileceğimiz bir çok geleneksel sezgisel yaklaşım[1], diğer kombinatoryal optimizasyon problemleri için yetersiz kalırken Gezgin Satıcı Problemi için yüksek performans göstermektedirler. Söz konusu sezgisel yaklaşımlarla, optimum sunuca %10-15 yakın çözümler üretebilmek mümkündür. Ayrıca, bu yaklaşımların ürettikleri çözümleri başlangıç noktası kabul edip geliştirmeye çalışan ve klasik yerel optimizasyon teknikleri adı verilen algoritmalar ile çok daha yüksek performans düzeylerine ulaşılabilmektedir. Örneğin, verilen geçerli bir turda, turun parçalara ayrılarak söz konusu parçaların, tur uzunluğunu azaltacak şekilde, yeniden birleştirildiği bir operasyonun tekrarına dayanan "2-Opt", "3-Opt" klasik yerel optimizasyon teknikleri ile bu performans düzeyi %3-4'lere çekilmiştir. Yine, buradaki arama sürecinde tüm olası durumların teker teker gözden geçirilmesini

---

1. Söz konusu "tur oluşturma" adı verilen geleneksel sezgisel yaklaşımlardan birkaçı:

- En yakın komşuluk,
- En uygun dahil etme,
- Clarke-Wright tasarruf yaklaşımı (*Clarke ve Wright, 1964*)

önleyen basit ve etkin bir algoritma ile (*Lin ve Kernighan, 1973*) %1-2 performans düzeylerine ulaşılabilmiştir. Hatta, Lin ve Kernighan tarafından geliştirilen bu yerel tarama algoritması, 1989 yılına kadar Gezgin Satıcı Probleminin çözümüne yönelik geliştirilen en iyi sezgisel yaklaşım olarak kabul görmüştür. Günümüzde söz konusu algoritmanın geliştirilen varyasyonları, 1.000.000 şehire kadar Gezgin Satıcı Problemleri ile bir saat içinde başa çıkabilmektedir. Elde edilen bu başarı düzeyleri pratik açıdan geliştirilemeyecek ölçüde yüksek başarı düzeyleridir.

5.2.2 Yeni Sezgisel Çözüm Yaklaşımları

Klasik sezgisel çözüm yaklaşımları ile elde edilen başarılı sonuçlar, Gezgin Satıcı Problemine katkıda bulunmak adına; Benzetimli tavlama, Genetik algoritmalar, Yapay Sinir Ağları gibi optimizasyon amaçlı geliştirilen yeni çözüm yaklaşımlarına, fazla bir hareket sahası bırakmamaktadırlar. Bu yeni yaklaşımlarla elde edilen sonuçlar pratik açıdan fazla bir önem taşımamakta, daha çok "En iyi hangisi?" sorusuna cevap bulma amacıyla kullanılmaktadır. Ancak yine de, bu yeni yaklaşımlar beraberlerinde, ele alınan problemlere yönelik iyi ya da olurlu çözümler oluşturmayan yerel minimum değerlerden kaçınmak; klasik yaklaşımlarda karşılaşılan detaylı modelleme ve programlama ihtiyaçlarından kurtulmak gibi birtakım katkı ve avantajlar da getirmektedirler. Diğer taraftan, Gezgin Satıcı Probleminin çözümüne yönelik yapılan bu çalışmalar, diğer optimizasyon problemlerinin çözüm kalitesinde anlamlı iyileştirmelere yol açacak şekilde de kullanılabilirler.

Johnson ve McGeoch (*1997*) söz konusu bu yaklaşımlar arasında, klasik yerel optimizasyon teknikleri ile birlikte Benzetimli tavlama (*simulated annealing*), Genetik algoritmalar ve Yapay Sinir Ağları gibi yerel optimizasyonun yeni varyasyonlarını da saymışlar ve bu yeni yerel optimizasyon tekniklerinin Gezgin Satıcı Problemine nasıl uyarlandıklarını ve gerek teorik gerekse pratik bakımdan göreli başarılarını değerlendirmeye çalışmışlardır. Bu amaçla daha önce, yukarıda verilen istatistikler de yine aynı çalışmadan alınmışlardır.

Elinizde çalışmada ele alınan Yapay Sinir Ağları haricinde, burada sözü geçen yeni yerel optimizasyon tekniklerinden biri olan Benzetimli Tavlama yöntemi, aslında; çözüm sürecinde eldeki mevcut çözüme komşu olan çözümleri tarayan bir yerel tarama algoritmasıdır. Ancak, diğer tarama algoritmalarından temel farklılığı, komşu çözümlerin seçiminde gelişigüzel bir sıra takip etmesidir. Bu sayede Benzetimli Tavlama yöntemi, amaç fonksiyonunun değerinde artışlara yol açan yukarı yönlü (*uphill*) hareketlere de izin vermekte ve böylece yerel minimum değerlere saplanma sorununa çözüm üretmektedir.

İlk defa Kirkpatrick ve diğerleri (*1983*) ve Cerny (*1985*) tarafından ileri sürülen yöntemin günümüzdeki geliştirilen varyasyonları, aşağıda verilen temel algoritmaya dayanmaktadır: Bu algoritmaya göre yöntemin tarama sürecinde, metallerin tavlanması süreci simüle edilerek sıcaklık (*temperature*) kontrol parametresinin düşüşü gözlemlenmektedir.

Tablo 5.1 Benzetimli Tavlama yöntemi temel algoritma planı (*Johnson ve McGeoch, 1997; s.48*).

1. Bir S başlangıç çözümü üret ve bu başlangıç çözümünü $\boldsymbol{S}^*$ şampiyon çözüm değişkenine ata.
2. T kontrol parametresine bir başlangıç sıcaklık değeri ata.
3. Döngü (*While not yet frozen*)
    - 3.1. Döngü (*While not yet at equilibrium*)
        - 3.1.1. Bir $\boldsymbol{S}'$ rassal komşu çözüm belirle.
        - 3.1.2. Δ = *Tur Uzunluğu*($\boldsymbol{S}'$) - *Tur Uzunluğu*(S)
        - 3.1.3. Eğer $\Delta \leq 0$ (aşağı yönlü hareket):
            $S = \boldsymbol{S}'$
            Eğer *Tur Uzunluğu*(S) < *Tur Uzunluğu*($\boldsymbol{S}^*$);
            $\boldsymbol{S}^* = S$
        - 3.1.4. Aksi takdirde (yukarı yönlü hareket):
            $r \in [0, 1]$ olmak üzere eğer $r < e^{-\Delta/T}$;
            $S = \boldsymbol{S}'$
        - 3.1.5. Döngü sonu (*equilibrium*)
    - 3.2. T sıcaklık değerini düşür.
    - 3.3. Döngü sonu (*frozen*)
4. Yeni bir $\boldsymbol{S}^*$ şampiyon çözüme ulaşıldığında birinci adıma geri dön.

Yapay Sinir Ağları haricinde, yukarıda sözü geçen diğer bir yeni yerel optimizasyon tekniği de Genetik Algoritmalar yöntemidir. Söz konusu yöntemin, optimizasyon amaçlı bir yaklaşım olarak kullanım geçmişi 1970'li yıllara kadar uzanmaktadır (*Goldberg, 1989*).

Genetik Algoritmalar, Gezgin Satıcı Problemine klasik sezgisel çözüm yaklaşımları ile üretilen başarılı sonuçlara en çok katkıda bulunan (hesaplama maliyetlerinde bir miktar artışla birlikte) yeni sezgisel çözüm yaklaşımlarından biridir. Genetik Algoritmalar yaklaşımının, Gezgin Satıcı Problemine günümüzdeki en iyi uyarlamaları, aşağıda verilen temel algoritma planını takip etmektedirler: Bu algoritma planında da bir önceki algoritma planında olduğu gibi, birtakım işlemler ve kavramlar açıkça belirtilmemiştir. Her iki durum için de söz konusu tanımlamalar yapılarak özel algoritma yapıları ortaya çıkarılabilir.

Tablo 5.2 Genetik Algoritmalar yöntemi temel algoritma planı (*Johnson ve McGeoch, 1997; s.71*).

1. **S** = {S_1, . . , S_k} k adet başlangıç çözümü içeren bir popülasyon üret.
2. Bir **A** yerel optimizasyon algoritmasını kullanarak **S** popülasyonu için elde edilen *S* yerel optimum çözümünü belirle.
3. Döngü (*While not yet converged*)
   - 3.1. **S** popülasyonundan k' adet ayrık, 1 ve 2 elemanlı alt popülasyonlar seç.
   - 3.2. Seçilen her bir 1-elemanlı alt popülasyon için rassal bir mutasyon işlemi gerçekleştirerek yeni bir çözüm elde et.
   - 3.3. Seçilen her bir 2-elemanlı alt popülasyon için rassal bir çaprazlama işlemi gerçekleştirerek yeni bir çözüm elde et.
   - 3.4. Elde edilen k' adet çözüme **A** yerel optimizasyon algoritmasını uygula ve sonuçları bir **S'** popülasyonunda topla.
   - 3.5. Bir seçme stratejisi kullanarak **S** Y **S'** birleşim kümesinden k adet sağ kalan çözüm seç ve bunları **S** popülasyonuna ata.
4. İkinci adıma yani S popülasyonu için elde edilen S yerel optimum çözümüne geri dön.

5.3 Gezgin Satıcı Problemine Yapay Sinir Ağları Yaklaşımları

Yöneylem Araştırması ve Matematik Programlamadaki temel sezgisel yaklaşımların yanı sıra, 15 yılı aşan bir süreçte yapılan araştırmalar neticesinde, kombinatoryal optimizasyon problemlerinin çözümüne yönelik çok sayıda Yapay Sinir Ağı yaklaşımı geliştirilmiştir. Bu yaklaşımların büyük bir çoğunluğu iki temel başlık altında toplanabilir:

1. Hopfield Ağ Yapıları,
2. Öz-düzenlemeli Haritalar.

Yukarıda verilen iki temel yaklaşımın her ikisi için de eksikliklerden, kusurlardan bahsetmek mümkündür. Bunlar arasında, Hopfield ağ yapılarının olurlu olmayan çözümler üretme eğilimleri ve Öz-düzenlemeli haritaların ise daha geniş bir sınıftaki kombinatoryal optimizasyon problemlerinin çözümüne yönelik olarak genelleştirilememe eksikliği sayılabilir. Takip eden bölümlerde söz konusu bu iki temel yaklaşım daha detaylı bir biçimde ele alınacaktır.

5.4 Yapay Sinir Ağları & Genetik Algoritmalar İşbirliği

Gezgin Satıcı Problemi aslında, benzer "NP-complete" optimizasyon problemlerine yönelik geliştirilen farklı optimizasyon yaklaşımlarının kendilerini ispatlamaya çalıştıkları ilk çalışma alanlarından biri durumundadır. Bu bakımdan, Gezgin Satıcı Problemine yönelik geliştirilen ve birkaçının yukarıda ele alındığı, birbirinin benzeri olan ancak diğerleri arasında çekinik kalan yaklaşımlara diğer kombinatoryal optimizasyon problemlerinin çözümünde anlamlı farklar oluşturabilir gözüyle bakmak gerekir. Söz konusu yöntemlerde var olan bir özellik başka bir optimizasyon probleminin çözümünde bir anlam ifade edebilir.

Gezgin Satıcı Problemine günümüzde üretilen en iyi çözümlerin arkasında, yukarıda birkaçının ele alındığı yeni sezgisel çözüm yaklaşımların birkaçının birlikte kullanıldığı hibrit yaklaşımlar yer almaktadır. Bu anlamda olaya

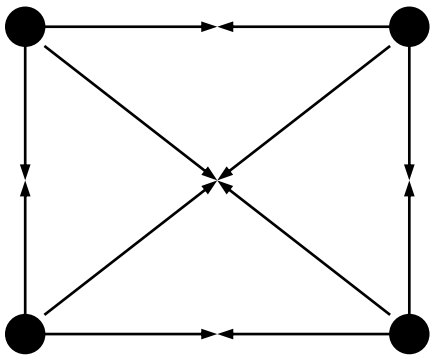
bakıldığında bu duruma olanak sağlayan Genetik Algoritmalar yaklaşımını, diğer yaklaşımlar arasında özel bir yere koymak mümkündür. Ancak, hibrit yaklaşımlarda temel algoritmanın yanında ikinci bir algoritmanın kullanılması, işlem süresinin ve dolayısıyla hesaplama maliyetlerinin kat kat artması anlamına gelmektedir ki bu durum, modelleme ve programlama süreçlerinde tam, eksiksiz ve detaylı bir çalışma gerektirmeyen Yapay Sinir Ağları yaklaşımlarının Genetik Algoritmalar ile birlikte kullanılmasını destekleyen bir neden olarak görülebilir.

Yapay Sinir Ağları ve Genetik algoritmalar işbirliği, yukarıda anlatıldığı şekilde Yapay Sinir Ağlarının Genetik algoritma uygulamalarında kullanılması (*Liong ve diğerleri, 2001*) yanında, tersi şekilde de, yani Yapay Sinir Ağları uygulamalarında Genetik algoritmalardan yararlanma şeklinde de gerçekleştirilebilmektedir. Söz konusu işbirliğinin ele alındığı bir çalışmada (*De Falco ve diğerleri, 1997*) Yapay Sinir Ağlarında (1) ağ yapısı deseninin belirlenmesi, (2) en iyi öğrenme yönteminin seçimi amaçlarıyla Genetik algoritmalar kullanılmaktadır. Hatta kullanılan Genetik algoritma yaklaşımının performansını arttırmak amacıyla bulanık (*fuzzy*) bir yeniden birleştirme işlemi de yaklaşıma ilave edilmiştir. Bu durum daha önce de dile getirildiği şekilde, birkaç yaklaşımın daha iyi çözümler elde etmek amacıyla aynı yaklaşım içinde birlikte kullanıldığı hibrit yaklaşımlara da iyi bir örnek oluşturmaktadır.

6. HOPFİELD&TANK SİNİR AĞI MODELİ

6.1 Hopfield Ağ Yapılarında Temel Yapı

Hopfield ağ yapıları, temel olarak kendiliğinden çağrışım (*auto-association*) amacıyla kullanılan tek katmanlı ve tam bağlantılı sinir ağı yapılarıdır (*Trenaman, 1998; s.10*). Ağ yapısındaki her bir birim, basit birer eşik değer işlemci birimidir ve her bir işlemci birim çifti arasında ağırlıklandırılmış, çift yönlü bir bağlantı söz konusudur. Aşağıdaki şekilde basit bir Hopfield ağ yapısı ele alınmaktadır:



Şekil 6.1 Basit bir Hopfield ağ yapısı modeli (*Trenaman, 1998; s.10*).

Kendiliğinden çağrışım amacıyla kullanımı yanında Hopfield ağ yapıları, Yapay Sinir Ağlarında kombinatoryal optimizasyon problemlerinin çözümüne yönelik geliştirilen ilk sezgisel yaklaşımlar arasında yer almaktadır.

Hopfield ağ yapılarında, problemdeki her bir değişkenin ağ yapısındaki bir düğümle ve değişkenler arasındaki kısıtların da bağlantı ağırlık değerleriyle temsil edilmesi öngörülmektedir. Bu noktada, pozitif ağırlık değerleri değişkenler arasındaki karşılıklı destekleyici ilişkileri gösterirken negatif ağırlık değerleri engelleyici ilişkileri göstermektedir. Sonuç olarak, kararlı bir yapıya ulaşmanın son hedef olduğu Hopfield ağ yapılarının söz konusu kararlı durumları, kısıtlar altında ele alınan

değişkenlerin doğruluk ve yanlışlık atamalarını yansıtacaktır. Cohen and Grossberg (*1983*) çalışmalarında, kararlı bir yapıya ulaşmanın son hedef olduğu Hopfield ağ yapılarında kararlı bir yapının salt

$w_{ij} = w_{ji}$ $\quad i \neq j$ için
$w_{ii} = 0$ $\quad$ her i için

olması durumunda gerçekleşeceğini göstermişlerdir. Bu durum aşağıda verilen ağ yapısı enerji fonksiyonu (*a Liapunov function*) kullanılarak ispatlanabilmektedir. Hopfield ağ yapılarıyla optimizasyon problemlerinin çözümünde bu enerji fonksiyonu, problem kısıtlarından elde edilen diğer bir enerji fonksiyonu ile karşılaştırılarak ağ yapısı ağırlık değerleri belirlenmektedir.

$$E = -1/2 \sum_i \sum_j w_{ij} o_i o_j - \sum_j I_j o_j + \sum_j q_j o_j \qquad (6.1)$$

Bu fonksiyonda:

E : ağ yapısı enerji değeri,
w_{ij} : i. birimden j. birime bağlantı ağırlık değeri,
o_j : j. birime ait çıktı değeri,
I_j : j. birime ait girdi değeri,
q_j : j. birime ait eşik değer.

olarak tanımlanmıştır.

6.2 Hopfield Ağ Yapılarında Öğrenme Süreci

Hopfield ağ yapıları, başlangıçta ağ yapısı durumunun hedeflenen şekilde bir duruma getirildiği ve bağlantı ağırlık değerlerinin tayin edildiği, öğrenilmiş ağ yapılarıdır. Girdi setinin ağ yapısına tanıtıldığı ve ağ yapısının kurgulandığı bu ilk süreç, Yapay Sinir Ağlarındaki temel öğrenme süreçlerinden farklılık göstermektedir. Çünkü, bu süreç boyunca bağlantı ağırlık değerleri değişikliğe uğramamaktadırlar. Hopfield ağ yapılarındaki bu durağan ve kararlı yapı aslında, ağ yapısındaki kalıcı bir hafızanın, belleğin göstergesidir.

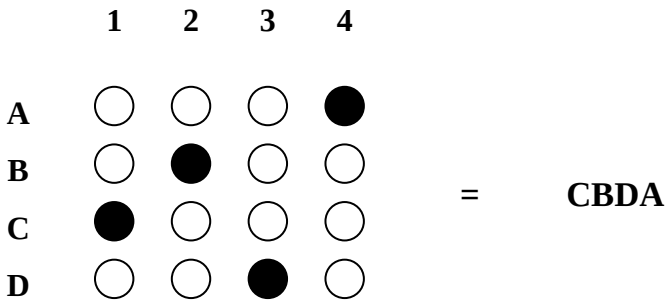
Bu ilk sürecin ardından ağ yapısının durumu bir E enerji fonksiyonu ile tanımlanır. Bu enerji fonksiyonu, Gezgin Satıcı Problemi için, şehir çiftleri arasındaki mesafelerin verildiği bir şehir setine ait, her bir şehrin sadece bir kez ziyaret edildiği ve başlangıç noktasına dönüldüğü en kısa turu temsil edecektir. Hopfield ağ yapılarında hedeflenen durumlar aslında, burada değinilen enerji fonksiyonunun minimum değerlerinde yerleşmiş durumdadır. Bu nedenle, Hopfield-Tank sinir ağı modeli, söz konusu E enerji fonksiyonunun yerel minimumlarını arayan bir eğim iniş algoritmasını kullanır. Bu modelde, elde edilen yerel minimum değerlerin, ele alınan Gezgin Satıcı Problemi için iyi bir çözüm oluşturduğu varsayımı söz konusudur. Ancak bu varsayım, Hopfield ağ yapılarının Gezgin Satıcı Problemine olurlu olmayan çözümler üretme eğilimini de beraberinde getirmektedir.

Hopfield çalışmalarında göstermiştir ki, ağ yapısının başlangıçta tanımlanan durumdan farklı bir durum ile güncellenmesi halinde, tanımlanan enerji fonksiyonunun değeri hiçbir zaman artmamakta, ya sabit kalmakta ya da düşmektedir (*Trenaman, 1998; s.18*). Bu nedenle, söz konusu ağ yapısı için enerji fonksiyonunun yerel minimumlarında saklı sonlu sayıda durum söz konusu olduğundan, izlenen bu süreç Gezgin Satıcı Problemi için optimum sonuca yakın bir çözüm içeren bir ağ yapısı ile sonuçlanacaktır. Ardından ağ yapısındaki işlemci birimlerin çıktı değerleri okunarak probleme çözüm olarak bulunan en kısa tur tanımlanabilir.

6.3 Hopfield Ağ Yapıları ve Gezgin Satıcı Problemi

Hopfield-Tank sinir ağı modelinde, Gezgin Satıcı Problemleri, n ağ yapısındaki işlemci birim (şehir) sayısı olmak üzere, nxn boyutlu ve "0-1" elemanlarından oluşan bir matris ile ifade edilebilmektedir. "0" ve "1" elemanlarından oluşan böyle bir matrisin temsil ettiği bir turda, matristeki "1" elemanı, ait olduğu satırdaki şehrin ait olduğu sütundaki sırada ziyaret edileceğini göstermektedir. Bu noktada, şehirler arasında istenilen şekilde bir tur tanımı için (dolayısıyla Gezgin Satısı Problemine üretilen çözümün olurlu bir çözüm olması için), ilgili matristeki her bir satırda ve her bir sütunda sadece ve sadece bir adet "1" elemanı bulunması şarttır.

Diğer taraftan, ele alınan problemdeki şehir setine ait en kısa tur hesabında, kat edilen toplam mesafe gösterge olarak kabul edilmektedir. Minimize edilmeye çalışılan bu gösterge, ilgili turda yer alan şehir çiftleri arasındaki mesafelerin toplanmasıyla elde edilmektedir.



Şekil 6.2 Dört şehirli bir Gezgin Satıcı Problemi için, Hopfield ağ yapısının durumunu gösteren matris ve bu matrisin temsil ettiği tur (*Trenaman, 1998; s.24*).

Hatırlanacak olursa, Hopfield ağ yapılarında ağ yapısının kurgulandığı ilk sürecin ardından, ağ yapısının durumu bir E enerji fonksiyonu ile tanımlanmaktaydı. Gezgin Satıcı Problemlerine yönelik geliştirilen Hopfield ağ yapılarının durumu, aşağıdaki şekilde bir enerji fonksiyonu ile tanımlanabilir (*Khanna, 1990; s.143*). Daha önce de belirtildiği gibi, böyle bir enerji fonksiyonu temel olarak, şehirler arasında tanımlanacak geçerli bir turda kat edilecek toplam mesafenin hesabına dayanmaktadır. Ancak söz konusu enerji fonksiyonu, problemin amaç fonksiyonundan farklı olarak, istenen şekilde bir tur tanımı için gerekli olan ve ceza terimleri adı verilen ilave terimler de içermektedir.

Aşağıda ele alınan enerji fonksiyonundaki ilk iki terim, tur içerisinde bir şehrin iki kez ziyaret edilmesini engellemekte; üçüncü terim ise her şehrin ziyaret edilmesini garanti altına almaktadır. Son terim ise, minimize edilmeye çalışılan gösterge olan şehirler arasında tanımlanan turun toplam uzunluğunu hesaplamaktadır. Ayrıca, fonksiyondaki A, B, C, D harfleri sabitleri, d şehirler arasındaki mesafeleri, V_{xi} ise şehirler matrisinde x nolu satırda temsil edilen şehrin tanımlanan turdaki ziyaret sırasını (i) göstermektedir.

$$
\begin{aligned}
E \;=\; & A \;/\; 2 \sum_{x} \sum_{i} \sum_{j \neq i} V_{xi} * V_{xj} \\
& + B \;/\; 2 \sum_{x} \sum_{i} \sum_{j \neq i} V_{xi} * V_{yi} \\
& + C \;/\; 2 \left[\left(\sum_{x} \sum_{i} V_{xi} \right) - n \right]^2 \\
& + D \;/\; 2 \sum_{x} \sum_{i} \sum_{j \neq i} d_{xy} * V_{xi} * \left(V_{y,i+1} * V_{y,i-1} \right)
\end{aligned}
\qquad (6.2)
$$

Yukarıda tanımlanan enerji fonksiyonunda *A, B, C* yeteri kadar büyük seçildiğinde, ilk üç terim sıfır değeri alacak, istenen bir tur yapısı elde edilecek ve enerji fonksiyonunun değeri dördüncü terime yani tanımlanan turda kat edilen toplam mesafeye eşitlenecektir.

Gezgin Satıcı Problemi için Hopfield sinir ağı modelinde, enerji fonksiyonunun yukarıdaki şekilde tanımlanmasının ardından, işlemci birimler arasındaki ağırlık değerlerinin tayini, aşağıda verilen formül ile yapılabilir. Söz konusu formül, daha önce tanımlanan iki ayrı enerji fonksiyonunun karşılaştırılması sonucu elde edilmektedir. Formülde yer alan p_{ij} teriminin değeri, $i = j$ için 1, diğer durumlar için 0 olarak ele alınmaktadır.

$$
\begin{aligned}
w_{xi,yj} \;=\; & - A * p_{xy} \left(1 - p_{ij}\right) \\
& - B * p_{ij} \left(1 - p_{xy}\right) \\
& - C \\
& - D * d_{xy} \left(p_{j,i+1} + p_{j,i-1} \right)
\end{aligned}
\qquad (6.3)
$$

Ağ yapısındaki ağırlık değerleri tayin edildikten ve diğer sistem parametreleri seçildikten sonra, Hopfield sinir ağı modelinde aşağıdaki algoritma kullanılarak probleme çözüm üretilmektedir:

Tablo 6.1 Hopfield ağ yapısında işlemci birimlerin çıktı değişkenlerinin hesabı (*Fu, 1994; s.42*).

5. $o_j(t)$, t anındaki j. işlemci birimin çıktı değerini göstermek üzere; ağ yapısındaki işlemci birimlerin çıktı değişkenlerine $o_j(0)$, rassal küçük başlangıç değerleri ata.

6.

$$o_j(t+1) = f\{\sum_i w_{ji}\, o_j(t) + I_j\}$$

$$f(a) = \begin{cases} 1, & a > q_j \\ 0, & a < q_j \\ o_j(t), & a = q_j \end{cases}$$

7. İşlemci birimlerin çıktı değişkenlerini t zaman indeksine göre güncelleyen algoritmanın ikinci adımını, işlemci birimlerin çıktı değerlerinde yaşanan değişim ortadan kalkıncaya kadar tekrar et. Böylece ağ yapısında ulaşılan durağan ve kararlı yapı, ele alınan problemin optimal çözümünü verecektir.

Hopfield ve Tank (*1985*) çalışmalarında, 10-şehir bir Gezgin Satıcı Problemi için A = B = D = 500 ve C = 200 seçmişler ve sürekli bir aktivasyon fonksiyonu kullanmışlardır (Yukarıda verilen Hopfield sinir ağı modelinde kesikli yapıda formülasyonlar ele alınmıştır). Söz konusu çalışmada yapılan denemelerin büyük bir çoğunluğu istenilen şekilde, geçerli turlarla sonuçlanmış ve bunların yarıya yakını da optimal sonucu vermiştir. Ancak, burada elde edilen sonuçlar başlangıçta seçilen sistem parametrelerine son derece bağlı kalmaktadır. Üstelik bu parametre değerlerinin tespiti için herhangi bir sistematik yöntem de bulunmamaktadır. Ayrıca, Hopfield-Tank sinir ağı modelinde, daha önce de belirtildiği gibi, tanımlanan enerji fonksiyonunun yerel minimum değerlerinin, ele alınan Gezgin Satıcı Problemi için iyi birer çözüm oluşturduğu varsayımı yapılmaktadır. İşte bu noktada, elde edilen yerel minimum noktalarda üzerinde çalışılan problemlere yönelik olurlu çözümler garanti edilememektedir. Diğer bir ifade ile, kullanılan enerji fonksiyonunun tüm minimum değerleri, ele alınan problem için olurlu bir çözüm sunamamaktadır.

6.4 Hopfield Ağ Yapıları Yaklaşımına Bir Örnek

Hopfield ağ yapısı yaklaşımının, Gezgin Satıcı Probleminin çözümüne yönelik nasıl uygulandığı ile ilgili olarak Thompson ve Thompson (*1987*) tarafından geliştirilen bir bilgisayar uygulamasına ulaşmak mümkündür. Söz konusu bilgisayar programı, probleme ürettiği çıktı verilerinden hareketle daha önce tanımlanan Hopfield ağ yapısı algoritma adımları izlenebilecek şekilde geliştirilmiştir. Programın kodları ve burada ele alınan problem için ürettiği çıktı verileri *EK.1*'de ayrıca verilmektedir.

Aşağıda, geliştirilen bilgisayar programından elde edilen, bir Gezgin Satıcı Problemine Hopfield ağ yapısı yaklaşımı ile çözüm üretme süreci adım adım ele alınmaktadır. Ancak bu süreçte, sistem parametrelerinin ve şehir koordinatlarının seçimi program tarafından rassal bir şekilde belirlenmektedir. Aşağıda, programın ürettiği çıktı verilerinin bir bölümüne yer verilmiştir. Bu verilerin ışığında, (Hopfield ağ yapısı algoritma adımları takip edilerek) ağ yapısındaki işlemci birimlerin çıktı değerlerinin yakınsamalarından hareketle problemin çözümüne nasıl ulaşıldığı rahatlıkla izlenebilmektedir. Ele alınan süreçte işlemci birimlerin eşik değerleri "0" olarak saptanmıştır.

```
Step: 1

      Input Voltages
                 1          2          3          4          5
      A      -8.43176   -6.73021   -5.14208   -3.66349   -2.23470
      B      -2.96753   -1.11288    0.15747   -5.44852   -4.03152
      C      -1.26019   -1.05439   -3.48170    1.65450   -2.31212
      D       0.17568   -6.03228  -10.69222   -8.25715   -2.70486
      E      -3.08320    0.10076   -7.62895   -7.24145   -1.44345

      Output Voltages
                 1          2          3          4          5
      A       0.00000    0.00000    0.00000    0.00000    0.00000
      B       0.00000    0.00000    1.00000    0.00000    0.00000
      C       0.00000    0.00000    0.00000    1.00000    0.00000
      D       1.00000    0.00000    0.00000    0.00000    0.00000
      E       0.00000    0.99996    0.00000    0.00000    0.00000
```

```
Step: 2

       Input Voltages
                    1           2           3           4           5
       A        -9.15455   -11.24608    -7.37414    -5.17148    -0.72740
       B        -8.50394    -5.85047     3.19446    -9.39395    -5.13549
       C        -5.69938    -8.09817    -7.89868     6.24268    -2.94818
       D         4.55374   -10.72067   -17.86475   -12.92350    -3.33700
       E        -7.05208     2.91327   -12.00456   -12.73527    -2.50124

       Output Voltages
                    1           2           3           4           5
       A         0.00000     0.00000     0.00000     0.00000     0.00000
       B         0.00000     0.00000     1.00000     0.00000     0.00000
       C         0.00000     0.00000     0.00000     1.00000     0.00000
       D         1.00000     0.00000     0.00000     0.00000     0.00000
       E         0.00000     1.00000     0.00000     0.00000     0.00000

Step: 3

       Input Voltages
                    1           2           3           4           5
       A        -9.87027   -15.71707    -9.58404    -6.66447     0.76475
       B       -18.52984   -12.54098     4.20092   -17.84472   -13.22850
       C       -13.57066   -17.07181   -14.27158     7.30853   -10.57796
       D         3.84911   -17.36248   -26.96570   -22.58203   -10.96289
       E       -14.78882     3.69757   -18.33641   -21.98142   -10.54846

       Output Voltages
                    1           2           3           4           5
       A         0.00000     0.00000     0.00000     0.00000     1.00000
       B         0.00000     0.00000     1.00000     0.00000     0.00000
       C         0.00000     0.00000     0.00000     1.00000     0.00000
       D         1.00000     0.00000     0.00000     0.00000     0.00000
       E         0.00000     1.00000     0.00000     0.00000     0.00000

       Tour: DEBCA   dist =  3.03235095514277E+0000
```

Yukarıda görülen program çıktılarında, üçüüncü adımda elde edilen ilk geçerli tur yapısı daha sonraki adımlarda da bir değişikliğe uğramamaktadır. Dolayısıyla üçüncü adım itibarıyla problem çözümüne ulaşıldığı görülmektedir. Bu durumun, düşük şehir sayısından dolayı tanımlanan enerji fonksiyonundaki yerel minimum nokta sayısının azlığından kaynaklandığını söylemek mümkündür. Şehir sayısının artması ile birlikte Hopfield ağ yapısı yaklaşımında yaşanan eksiklikleri izleme adına, Miguel de Campos (*2001*) tarafından geliştirilen başka bir bilgisayar programına başvurulabilir. Söz konusu bilgisayar programında, ele alınan bir 10-şehir Gezgin Satıcı Problemi için çözüm üretme sürecinde, enerji fonksiyonunun yerel minimum noktalarından birine saplanıp kalma gibi modelin daha önce bahsedilen eksiklikleri rahatlıkla izlenebilmektedir.

6.5 Hopfield Ağ Yapıları Yaklaşımında Yaşanan Gelişmeler

Hopfield ve Tank (*1985*), kombinatoryal optimizasyonun temel problemlerinden biri olan Gezgin Satıcı Probleminin bir Hopfield ağ yapısı kullanılarak çözülebileceğini göstermişlerdir. Ancak, ele alınan probleme ilişkin çok sayıda terim ve parametre içeren bir enerji fonksiyonunun minimizasyonuna dayanan bu teknik, genellikle olurlu olmayan çözümler üretmektedir ve bu durum Wilson ve Pawley (*1988*) tarafından ortaya konmuştur. Daha sonra yapılan Hopfield ağ yapıları ile Gezgin Satıcı Problemine çözüm üretme çalışmaları, modelin olurlu çözümler üretmesini garanti altına alacak şekilde, gerek ele alınan enerji fonksiyonunun geliştirilmesi (*Van den Bout ve Miller, 1988*); gerekse kullanılan başlangıç parametre değerlerinin daha optimum bir biçimde seçilmesi (*Kamgar-Parsi ve Kamgar-Parsi, 1992*; *Lai ve Coghill, 1992*) üzerinde yoğunlaştırılmıştır. Son olarak, Smith ve diğerleri (*1998*), elde edilen sonuçların olurluluğunu garanti altına alırken aynı zamanda yerel minimum çözümlerden kaçmaya imkan veren geliştirilmiş bir Hopfield ağ yapısı modeli ortaya koymuşlardır. Söz konusu çalışmada, Hopfield-Tank sinir ağı modelinde yapılan geliştirmeler şu iki temel nokta üzerinde toplanabilir:

1. Çok sayıda terim ve parametre içeren enerji fonksiyonundaki ceza terimlerini bir değişken içinde toplayarak bunların neden oldukları yerel minimum nokta sayısındaki artışı azaltmak.

2. Enerji fonksiyonunun minumum değerine ulaşma sürecinde daha etkin eğim-iniş algoritmaları kullanmak.

Günümüze kadar yapılan tüm bu çalışmalar neticesinde söylenebilir ki, temel Hopfield-Tank sinir ağı modeli ve varyasyonları ile, günümüzde 200 şehirden az şehir setine sahip Gezgin Satıcı Problemleri için çözümler üretebilmek mümkündür (*Altınel ve diğerleri, 2000*). Ancak elde edilen çözümlerin kalitesi bakımından olaya aynı şekilde iyimser bakmak mümkün değildir.

Yukarıda sayılan tüm gelişmelere rağmen, Gezgin Satıcı Probleminin çözümünde Hopfield ağ yapılarını kullanma fikrinin pek de tutulduğu söylenemez. Bu durumu bir kaç temel nedene bağlamak mümkündür. 1995 yılında yayımlanan bir bilimsel çalışmada, özellikle öklit uzaklığının dikkate alınmadığı Gezgin Satıcı Problemlerinde, Hopfield ağ yapıları kullanılarak elde edilen sonuçların, Yöneylem Araştırması ve Matematik Programlamadaki geleneksel sezgisel yöntemler ile elde edilen sonuçlardan çok da iyi olmadığına dikkat çekilmiştir (*Gee ve Prager, 1995*). Yani, onca yıldır Hopfield ağ yapıları üzerine yapılan çalışmalarda, olurlu olmayan çözümler üretme riski ortadan kaldırılmaya çalışılırken elde edilen sonuçların kalitesi göz ardı edilmiştir.

Ayrıca Hopfield ağ yapılarının, gerçek hayatta karşılaşılan optimizasyon problemlerinin çözümünde gösterdikleri performanslarının yeteri kadar test edilmediğini söylemek mümkündür. Konu ile ilgili literatürde yer alan çalışmaların büyük bir bölümü, Gezgin Satıcı Probleminin çözümüne yönelik geliştirilen Hopfield sinir ağı modellerinin eksiklikleri üzerinde ve bu eksiklikleri gidermeye yönelik öneriler üzerinde yoğunlaşmaktadır.

Gerçek hayatta karşılaşılan optimizasyon problemlerinin çözümünde, sinir ağı yaklaşımlarının performanslarının diğer sezgisel yöntemlerin performansları ile karşılaştırıldığı literatürün azlığı aslında, Yapay Sinir Ağlarının "*NP-complete*" problemlerin çözümüne yönelik kullanımındaki esas nedeninin göz önünde tutulmadığı sonucunu doğurmaktadır. "*NP-complete*" problemler için sezgisel yaklaşımlar, hızlı çözümlere ulaşma ihtiyacından dolayı kullanılmaktadır. Global optimal bir çözüme ulaşma hedefi, o kadar da önem taşımamaktadır. Ancak bu ifade ile, iyi sonuçlar üretmediği bilinen bir yaklaşımın optimizasyon problemlerinin çözümüne yönelik uygulanması gerekir, sonucu da çıkarılmamalıdır.

Hopfield ağ yapılarının Gezgin Satıcı Probleminin çözümünde, Yöneylem Araştırması ve Matematik Programlamadaki diğer sezgisel yöntemlerle

karşılaştırıldığında düşük kalan performansı çok daha basit bir kavramla Smith (*1996*)'da şu şekilde açıklanmaktadır:

Geleneksel sezgisel yöntemlerin aksine Hopfield ağ yapıları, Gezgin Satıcı Problemi için tam sayılı ve ikinci dereceden (*quadratic*) bir formülasyon kullanmaktadır. Bu durum, en küçüklenmeye çalışılan amaç fonksiyonunda çok sayıda yerel minimum noktalar oluşmasına neden olmaktadır. Öklit uzaklığının dikkate alınmadığı, asimetrik Gezgin Satıcı Problemlerinde ise bu sayı daha da artmaktadır. Dolayısıyla, basit bir eğim-iniş algoritması kullanan Hopfield ağ yapıları söz konusu yerel minimum noktalardan herhangi birine kolaylıkla saplanmakta ve doğrusal formülasyon kullanan diğer sezgisel yöntemlerin çözüm kalitesi Hopfield ağ yapılarından daha yüksek çıkmaktadır. Hopfield ağ yapılarının Gezgin Satıcı Problemine ürettikleri çözümlerin kalitesindeki bu düşüklük aynı zamanda, gerçek hayatta karşılaşılan optimizasyon problemlerinin çözümünde Hopfield ağ yapılarının kullanımını da olumsuz bir şekilde etkilemektedir.

Hopfield ağ yapılarında çözüm kalitesini arttırmaya yönelik yapılan geliştirmeler neticesinde, gerçek hayatta karşılaşılan bir çok optimizasyon probleminin çözümünde Hopfield ağ yapılarının kullanımı gerçekleştirilebilmiştir. Söz konusu optimizasyon problemleri arasında bir montaj hattı için farklı araba modellerinin sıralanması ve bir posta dağıtım şebekesinde dağıtım merkezlerinin konumlandırılması sayılabilir. Smith ve diğerleri (*1998*) Hopfield ağ yapılarının ve Öz-düzenlemeli haritaların, yukarıda sayılan problemlerin çözümü için kullanımında, diğer sezgisel yaklaşımlarla (*simulated annealing*) etkin bir biçimde rekabet edebileceklerini, elde edilen sonuçları karşılaştırarak göstermişlerdir. Bu noktada, ele alınan yöntemlerin performanslarının benzerliğini gösteren sonuçların yanında, Sinir ağı yaklaşımlarının uygulamalardaki hız ve hesaplama gücü özellikleri de göz önüne alındığında; Yapay Sinir Ağı yaklaşımlarının kombinatoryal optimizasyon problemlerinin çözümüne yönelik kullanışlılığı da ortaya çıkmaktadır.

7. ÖZ-DÜZENLEMELİ HARİTALAR

7.1 Öz-düzenlemeli Haritalar Yaklaşımında Yaşanan Gelişmeler

Hopfield ağ yapıları için daha önce yapılan değerlendirmelere benzer bir değerlendirme, Öz-düzenlemeli haritaların Gezgin Satıcı Problemine çözüm üretme yaklaşımı ile ilgili literatür için de yapılabilir (*Angeniol, De La Croix ve Le Texier, 1988*; *Fort, 1988*; *Favata ve Walker, 1991*). Ancak bu seferki değerlendirme nedeni, sadece performansların karşılaştırıldığı ilgili literatür azlığından kaynaklanmamakta, daha çok Öz-düzenlemeli haritaların büyük bir çoğunluğunun Elastik ağ yapısı adı verilen bir yaklaşımdan türemiş olmasına dayanmaktadır (*Durbin ve Willshaw, 1987*).

Sinir ağı yapılarında Öz-düzenlemeli haritaların (*self-organizing maps*) temel prensipleri (*Kohonen, 1982*) bir elastik yörünge kavramı ile açıklanabilir. Söz konusu elastik yörünge, aynı zamanda şehir setinin de yer aldığı öklit uzayında hareket edebilen çember şeklinde bir yolda sıralanmış işlemci birimleri temsil etmektedir. Bu elastik yörünge, tüm şehirlerin üzerinden geçecek şekilde işlemci birimler ile şehirleri eşleştirdiğinde, elde edilen yol (*path*) Gezgin Satıcı Problemine çözüm oluşturmaktadır. Dolayısıyla, bu yaklaşımı kullanan herhangi bir Öz-düzenlemeli harita da, sadece öklit uzaklığının kullanıldığı Gezgin Satıcı Problemlerinin çözümüne yönelik olarak kullanılabilmektedir.

Yukarıdaki durum göz önünde bulundurularak, daha geniş bir sınıftaki Kombinatoryal optimizasyon problemlerinin çözümüne yönelik genelleştirilebilen yeni Öz-düzenlemeli harita yaklaşımları daha sonraki bilimsel çalışmalarda ortaya konmuştur. Smith ve diğerleri (*1998*)'nin çalışmalarında bu amaçla ileri sürdükleri yöntem, olasılık değişkenlerinden oluşan ve permütasyon matrisi (*permutation matrix*) adı verilen bir fikre dayanmaktadır. Bu yöntemde, Elastik ağ yapısı orijinli diğer

yaklaşımların aksine, çıktı katmanındaki işlemci birimler öklit uzayında yer değiştirmemekte; yerine permütasyon matrisindeki i. satırdaki şehrin j. sütundaki sırada ziyaret edilme olasılığını gösteren w_{ij} bağlantı ağırlık değerleri değişmektedir.

Günümüze kadar yapılan tüm bu çalışmalar neticesinde söylenebilir ki, Gezgin Satıcı Problemine yönelik geliştirilen tüm Yapay Sinir Ağı yaklaşımları arasında Kohonen tarafından ileri sürülen ve sinir ağı yapılarının kendi kendine organizasyonuna dayanan Öz-düzenlemeli haritalar, hem optimum sonuca en yakın hem de en hızlı sonuçlar üretmesi bakımından, günümüzdeki en iyi sinir ağı yaklaşımı olarak karşımıza çıkmaktadır (*Altınel ve diğerleri, 2000*).

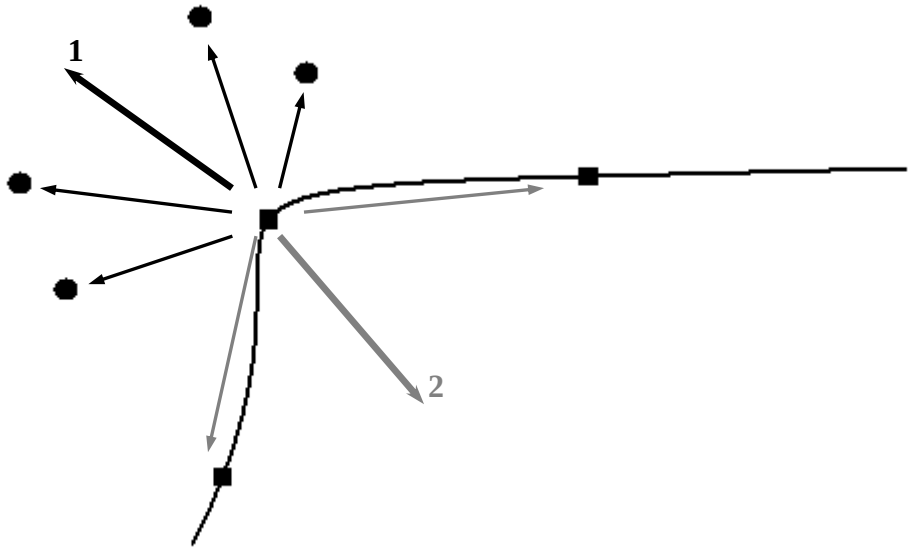
7.2 Elastik Ağ Yapısı Yaklaşımı

Elastik ağ yapısı yaklaşımı, Gezgin Satıcı Probleminin çözümüne yönelik geliştirilen Yapay Sinir Ağı yaklaşımlarından biridir. Söz konusu yöntem, çember şeklinde kapalı bir yörüngenin tüm şehirlerden yeterince yakın geçene dek şehirlerin pozisyonlarına doğru uzatıldığı, tekrar eden bir sürecin uygulanmasına dayanmaktadır.

Elastik ağ yapısı yaklaşımında, ele alınan düzlem üzerindeki her bir şehir, başlangıçta küçük bir çember şeklinde tanımlanan bir yörünge üzerindeki bir nokta ile eşleştirilir ve süreç boyunca şehirlerin pozisyonlarına doğru büyüyen bu çember, en sonunda tüm şehirlerden geçen kapalı bir yörünge halini alır. Böylelikle Gezgin Satıcı Problemine çözüm oluşturacak bir tur tanımı yapılmış olur. Bu eşleştirmede, yörünge üzerindeki birbirine yakın, komşu noktaların ele alınan düzlemde de mümkün olduğunca yakın kalmasına dikkat edilir. Söz konusu yöntemde, başlangıçta tanımlanan çamber şeklindeki yörüngenin, şehirlerin pozisyonlarına doğru uzaması sırasında çember üzerindeki her bir nokta aşağıdaki iki temel kuvvetin etkisi altındadır:

1. Birinci gruptaki kuvvetler, çember üzerindeki her bir noktayı kendisine yakın olan şehirlere doğru çekerek çemberin uzamasına ve şehirlerin pozisyonlarına yakınlaşmasına neden olurlar.

2. İkinci gruptaki kuvvetler ise, çember üzerindeki her bir noktayı yine çember üzerindeki komşu noktalara doğru çekerek uzayan yörüngenin toplam uzunluğunun minimizasyonuna çalışırlar.



Şekil 7.1 Yörünge üzerindeki noktaların etkisi altında kaldığı kuvvetler (*Budnik ve Filipova, 1996*).

Yukarıda tanımlanan kuvvetlerin etkisi altında, ele alınan düzlemdeki her bir şehir, söz konusu yöntem ile, yörüngedeki belirli bir bölge ile ilişkili hale getirilmiş olmaktadır. Buradaki ilişkinin gücü, her bir şehrin pozisyonundan kaynaklanan bileşke kuvvetlere bağlı olarak belirlenmekte ve bu bağımlılık, süreç ilerledikçe sürekli değişmektedir. Başlangıçta, düzlemdeki şehirler ile yörüngedeki noktalar arasındaki ilişkilerin gücü birbirine çok yakın iken süreç ilerledikçe, uzaklık farklarının küçülmesinden dolayı düzlemdeki her bir şehir, yörüngede kendisine daha yakın olan noktalarla ilişki kurar hale gelmektedir.

Elastik ağ yapısı yaklaşımının, Gezgin Satıcı Probleminin çözümüne yönelik nasıl uygulandığı ile ilgili olarak Budnik ve Filipova (*1996*) tarafından geliştirilen bir bilgisayar uygulamasına ulaşmak mümkündür. Elastik ağ yapısı yaklaşımının algoritmik yapısı daha sonra ele alınacak Öz-düzenlemeli harita yaklaşımı ile benzerlik gösterdiğinden, bu noktada daha fazla ayrıntıya girilmeyecektir.

7.3 Kohonen Öz-düzenlemeli Haritalar Yaklaşımı

Gezgin Satıcı Problemi gibi optimizasyon problemlerinin çözümüne yönelik ileri sürülen sinir ağı yaklaşımlarından biri de, Teuvo Kohonen'in çalışmalarında önemli bir yer tutan, Öz-düzenlemeli haritalardır. Söz konusu yaklaşımda temel düşünce, ağ yapısının kendi kendisini organize etmesi için örnek verilerin, (ağ yapısında bir kararlılığa ulaşılana dek) sürekli ve rasgele bir biçimde ağ yapısına tanıtılması gerekliliğidir.

Öz-düzenlemeli harita yaklaşımlarının, Gezgin Satıcı Probleminin çözümüne yönelik nasıl uygulandığı ile ilgili olarak Lorenzo (*1998*) tarafından geliştirilen bir bilgisayar uygulamasına ulaşmak mümkündür. Söz konusu bilgisayar programı, probleme konu olan şehirlerin pozisyonlarının kullanıcı tarafından ekrandan girilmesini sağlayacak şekilde geliştirilmiştir. Programın kodları ve burada ele alınan problem için ürettiği çıktı verileri *EK.2*'de ayrıca verilmektedir. Bundan sonraki bölümde, bu bilgisayar uygulamasında esas alınan Öz-düzenlemeli harita yaklaşımı için, ağ yapısı tasarımı ve takip edilen algoritmanın adımları ayrıntılı bir biçimde ele alınacaktır.

7.3.1 Ağ Yapısı Tasarımı

Gezgin Satıcı Problemini çözmeye yönelik tasarlanan böyle bir sinir ağı yapısı, iki ayrı katmanda dizilen iki ayrı işlemci birim grubunun oluşturduğu bir yapı olarak karşımıza çıkmaktadır.

İlk katmanda dizilen gruptaki her bir işlemci birim, hem gruptaki diğer işlemci birimler ile hem de kendileri ile bağlantı kurmuş durumdadır. Bu bağlantıların ağırlık değerleri Gezgin Satıcı Problemi için, işlemci birimler (şehirler) arasındaki mesafelere bağlı olarak tayin edilirler. Bağlantı ağırlık değerlerinin hesabı, aşağıdaki şekilde formüle edilebilir:

$$r[i,j] = e^{-d(i,j)^2 / 2\alpha} \qquad (7.1)$$

$r[i,j]$: i. ile j. işlemci birimler arasındaki bağlantı ağırlık değeri.

$d(i,j)$: i. ile j. işlemci birimler arasındaki öklit uzaklığı.

$\alpha > 0$: öğrenme katsayısı.

Elastik ağ yapısı yaklaşımında yörünge üzerindeki noktalarla eşleştirebileceğimiz bu birinci gruptaki işlemci birimler arasındaki bağlantı ağırlık değerleri, ağ yapısına her bir şehir tanıtımında konumlarını değiştiren işlemci birimler arasındaki uzaklığa bağlı olarak yeniden hesaplanmaktadır.

Elde edilen bu tek katmanlık ağ yapısına veri girişini sağlamak amacıyla ikinci bir katmanda başka bir grup işlemci birime daha gereksinim vardır. Yalnız, bu ikinci katmandaki işlemci birimler, birinci katmandaki yapıdan farklı bir biçimde kurgulanırlar. Bu kurguda, ikinci katmandaki her bir işlemci birim sadece ve sadece birinci katmandaki işlemci birimler ile bağlantı kurar; kendi aralarında bağlantı kurmazlar.

7.3.2 Takip Edilecek Algoritmanın Adımları

Ele alınan öz-düzenlemeli haritalar yaklaşımında, yukarıdaki şekilde ağ yapısı tasarımı tamamlandıktan sonra takip edilecek algoritmanın temel adımları şu şekilde sıralanabilmektedir:

1. İlk adımda, ağ yapısında birinci grupta yer alan işlemci birimler, istenen son düzene göre sıralanırlar. Gezgin Satıcı Problemi için istenen son düzen, tüm şehirleri birbirine bağlayan bir döngü şeklindedir ve tanımlanan bu döngünün uzunluğunun en kısa olması istenir.

Ağ yapısı tasarımının da gerçekleştirildiği bu ilk adımda, birinci grupta yer alan işlemci birim sayısının (k), ele alınan problemdeki şehir sayısından küçük olmaması şartıyla, k adet işlemci birim bir yörünge üzerinde yer alacak şekilde sıralanır. Rasgele sıralanan bu işlemci birimler algoritmada, x ve y koordinatlarıyla tanımlanırlar.

Hatırlanacak olursa, algoritmadaki daha sonraki adımlar için, tanımlanan bu ağ yapısına girdi sağlamak amacıyla ikinci bir grup işlemci birime daha ihtiyaç duyulmaktaydı. Gezgin Satıcı Probleminde, girdiler şehirlerin koordinatları olduğundan, x ve y koordinatlarını temsil edecek iki adet işlemci birim bu katmanda yeterli olacaktır ($n = 2$).

Son olarak, algoritmada kullanılacak öğrenme katsayısı, işlemci birimler arasındaki ağırlık değerleri gibi parametrelere rasgele bir biçimde başlangıç değerleri atanarak ikinci adıma geçilir.

2. Problemde tanımlanan şehir setinden rasgele bir seçim yap ve seçilen şehrin koordinatlarını girdi katmanındaki işlemci birimlere ata. (X ve Y değerleri).

3. Ağ yapısında birinci gruptaki işlemci birimler arasından, aşağıda tanımlanan değeri minimum yapan işlemci birimi bul.

$$(X - w_{Xj})^2 + (Y - w_{Yj})^2$$

$w_{X\,j}$ ve $w_{Y\,j}$, ağ yapısında birinci grupta tanımlanan j. işlemci birime ait ağırlık değerleri. Bu değerler aslında, söz konusu işlemci birimin ele alınan düzlemdeki koordinatlarını göstermektedir.

4. Ağ yapısında birinci grupta yer alan tüm işlemci birimler için tüm $w_{X\,i}$ ve $w_{Y\,i}$ ağırlık değerlerini aşağıdaki şekilde güncelle.

$$w_{Xi} = w_{Xi} + \alpha * r_{ij} * (X - w_{Xi})$$

$$w_{Yi} = w_{Yi} + \alpha * r_{ij} * (Y - w_{Yi})$$

5. ά ve α öğrenme katsayılarını küçülterek birinci gruptaki işlemci birimler arasındaki bağlantı ağırlık değerlerini (r) yeniden hesapla.

6. İkinci adıma dön.

7.3.3 Kohonen Öz-düzenlemeli Haritalar Yaklaşımına Bir Örnek

Bu bölümde, Ele alınacak Gezgin Satıcı Problemi için seçilen şehir koordinatları aşağıda bir tablo halinde verilmektedir. Geliştirilen bilgisayar programında şehirlerin ve işlemci birimlerin yer alacağı düzlem, koordinat ekseninde $0<x<1$ ve $0<y<1$ sınırlandırılmış alanı şeklinde tanımlanmıştır:

Tablo 7.1 5-Şehir Gezgin Satıcı Problemi seçilen şehir koordinatları.

Şehir Koordinatları		
	X	***Y***
0	0,268	0,531
1	0,508	0,949
2	0,525	0,340
3	0,093	0,139
4	0,938	0,883

Aynı zamanda ağ yapısı tasarımının da gerçekleştirildiği izlenen algoritmanın ilk adımında, işlemci birim sayısı ele alınan problemdeki şehir sayısının iki katı olacak şekilde (10 tane) seçilmiştir. Bilgisayar programının çıktı verilerinde, söz konusu işlemci birimlere atanan başlangıç koordinat değerleri aşağıdaki şekilde

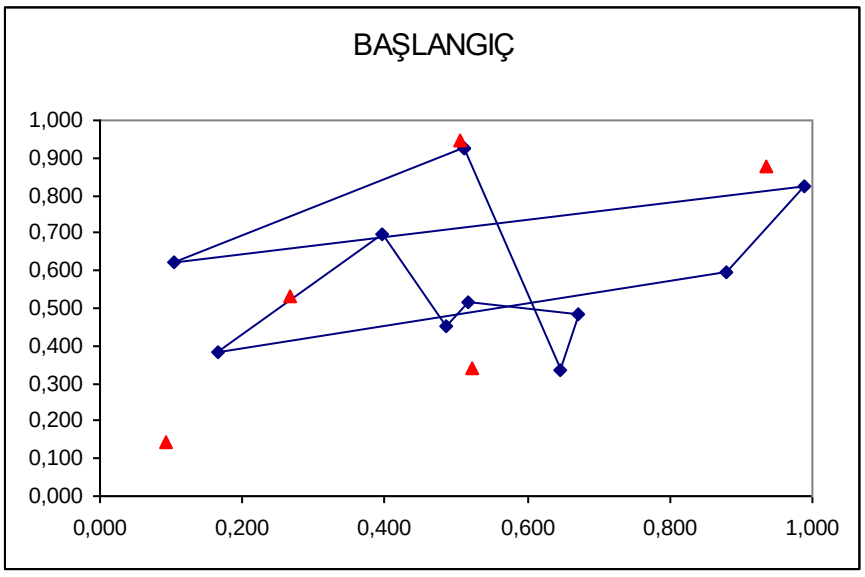
gerçekleşmiştir: Buradaki işlemci birimlere ait w_{Xi} ve w_{Yi} ağırlık değerleri, ele alınan şehir sistemine ait koordinatlar ile çakışacak şekilde yakınsadıklarında, bilgisayar programının Gezgin Satıcı Problemine ürettiği en kısa tur tanımı da yapılmış olacaktır.

Tablo 7.2 İşlemci Birimlere atanan Başlanıç koordinat değerleri.

İşlemci Birim Başlangıç Koordinatları

	X	*Y*		*X*	*Y*
0	0,510	0,924	**5**	0,395	0,699
1	0,647	0,336	**6**	0,166	0,385
2	0,671	0,485	**7**	0,878	0,597
3	0,518	0,516	**8**	0,989	0,822
4	0,486	0,450	**9**	0,103	0,624

Şekil 7.2'de, 5-şehir bir Gezgin Satıcı Problemi için, yukarıda seçilen şehirlerin konumları ve ağ yapısındaki işlemci birimlere atanan rassal başlangıç koordinat değerleri bir grafik üzerinde gösterilmektedir. Dikkat edilecek olursa, işlemci birimlerin başlangıçta istenildiği şekilde kapalı bir yörünge üzerinde sıralanmadıkları fark edilecektir. Ancak bu durum, şehir koordinatlarının ağ yapısına tanıtılması ile birlikte zamanla düzelecektir.

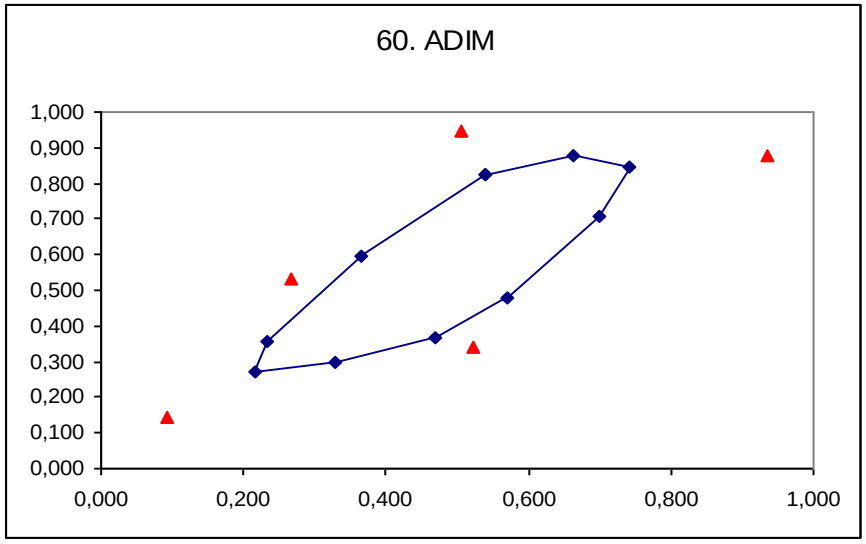


Şekil 7.2 5-Şehir Gezgin Satıcı Problemi için adım - 0.

Bu noktadan sonra, öğrenme katsayısı gibi gerekli sistem parametrelerinin de atamaları yapılarak algoritmanın diğer adımlarına geçilir. Öncelikle, ele alınan problemin şehir setinden rasgele bir seçim yapılır ve söz konusu seçim ağ yapısına tanıtılır. Ağ yapısına tanıtılan şehire en yakın işlemci birim tespit edilir ve elde edilen veriler ışığında işlemci birimlerin koordinatları güncellenir. Takip edilen algoritmanın bu bir adımı için, geliştirilen bilgisayar programının ürettiği çıktılara örnek olarak aşağıdaki veriler gösterilebilir:

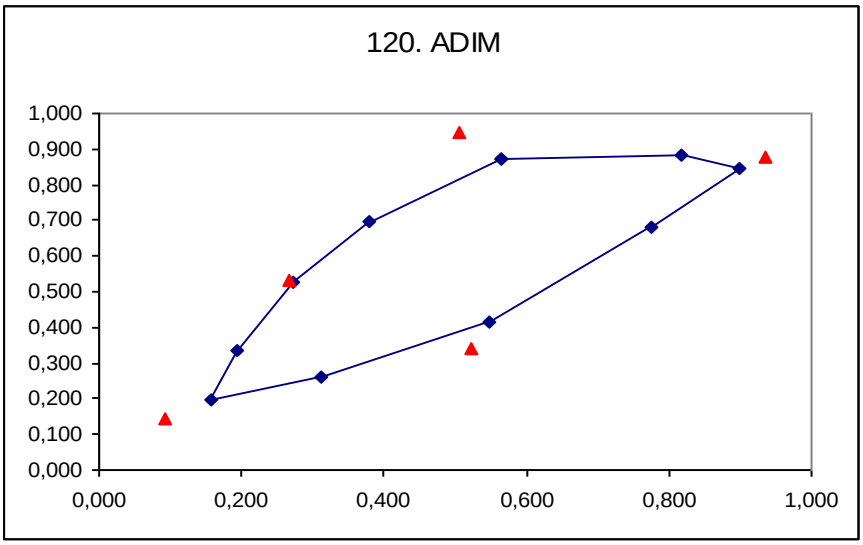
```
CHOSE A RANDOM PATTERN (Algoritma 2. adım)

       Choose a city       city is: 1
       x1: 0.5060048209318149    y1: 0.9488270103326957

SEARCH FOR MINIMAL (Algoritma 3. adım)

       d0=0.002041143125096919   d1=0.3559415785649712
       d2=0.21627388540422732    d3=0.17410666553750662
       d4=0.2219718943372627     d5=0.11960289022297033
       d6=0.3105806655907678     d7=0.1505112314384999
       d8=0.12812049265264963    d9=0.2485711999171257

       mindist= 0.002041143125096919 for : 0

UPDATE WEIGHTS (Algoritma 4. adım)

       gn[0].weight=(0.4985765048525768 ,0.9121230860807812 )
       gn[1].weight=(0.5747138695410613 ,0.588480427328807  )
       gn[2].weight=(0.605829407118573  ,0.598192261684896  )
       gn[3].weight=(0.4884968868148432 ,0.5720703638013079 )
       gn[4].weight=(0.439994705996446  ,0.5051962974197807 )
       gn[5].weight=(0.3576565658623543 ,0.6466399906499821 )
       gn[6].weight=(0.2452705932955198 ,0.4921256708189498 )
       gn[7].weight=(0.6139325493000105 ,0.6148411050297651 )
       gn[8].weight=(0.7357574721266006 ,0.7930365559711824 )
       gn[9].weight=(0.2823008103471305 ,0.7427349034406745 )

DECREASE LEARNING PARAMETERS (Algoritma 5. adım)
RE-COMPUTE r MATRIX

       start to calculate r
        r[0][1]= 0.8229573574808847     r[0][2]= 0.49411986269887576
        r[0][3]= 0.26301929356848097    r[0][4]= 0.15792198227990428
        r[0][5]= 0.1299630572252131     r[0][6]= 0.15792198227990428
        r[0][7]= 0.26301929356848097    r[0][8]= 0.4941198626988755
        r[0][9]= 0.8229573574808847     . . . . .
       r has been calculated

       PLOT RESULT EVERY 10 SESSIONS
       theta = 0.4950125  phi = 0.4950125
```

Yukarıda anlatılan bu süreç, parametre değerlerinde gerekli ayarlamalar da yapılarak sürekli tekrarlanır ve işlemci birimlerin şehir setindeki şehirlerin koordinatlarına yakınsamaları ile son bulur. Bu noktada, işlemci birimler kapalı bir yörünge üzerinde yer alacak şekilde sıralandığından, ele alınan Gezgin Satıcı Problemine üretilen en kısa tur da tanımlanmış olmaktadır. Aşağıda, bu yakınsamanın rahatlıkla izlenebileceği şekilde, takip edilen sürecin faklı adımlarında sistemin durumunu gösteren şekiller çizilmiştir:



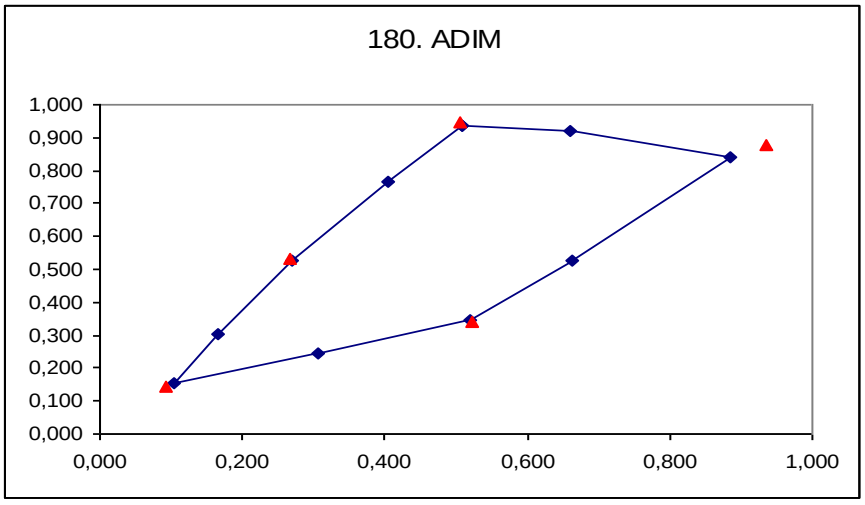
Şekil 7.3 5-Şehir Gezgin Satıcı Problemi için adım - 60.

Şekil 7.3'te ele alınan 5-şehir Gezgin Satıcı Problemi için takip edilen algoritmanın 60. adımının ardından ağ yapısındaki işlemci birimlerin yeni konumları görülebilmektedir. Dikkat edilecek olursa, ağ yapısındaki işlemci birimler artık, istenen şekilde bir kapalı yörünge üzerinde sıralanmış durumdadırlar.



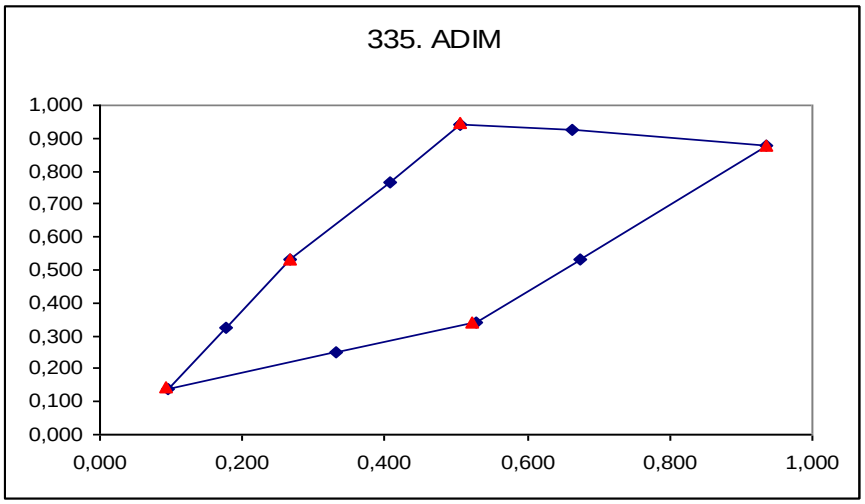
Şekil 7.4 5-Şehir Gezgin Satıcı Problemi için adım - 120.

Şekil 7.4'te kapalı bir yörünge üzerinde sıralanan ağ yapısındaki işlemci birimlerin koordinatlarının, Elastik ağ yapısı yaklaşımına benzer şekilde, ele alınan problemdeki şehirlerin koordinatlarına doğru yakınsadığı görülmektedir.



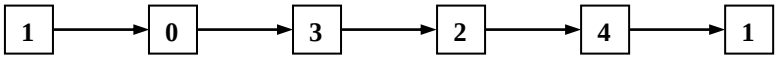
Şekil 7.5 5-Şehir Gezgin Satıcı Problemi için adım - 180.

Şekil 7.5'te ağ yapısındaki işlemci birimlerin şehirlerin koordinatlarına yakınsaması sürecinin sona yaklaştığını görmek mümkündür. Bu noktadan sonra işlemci birimler ile şehirler arasında bir eşleştirme yapılabileceğinden en kısa tur tanımı da rahatlıkla yapılabilmektedir.



Şekil 7.6 5-Şehir Gezgin Satıcı Problemi için adım - 335.

İşlemci birimlerin koordinatları ile şehirlerin koordinatlarının tam olarak çakışması Şekil 6.13'de de görülebileceği şekilde 335. adımda gerçekleşmektedir. Sonuç olarak, ele alınan 5-şehir Gezgin Satıcı Problemine üretilen son çözüm (en kısa şehir turu) şu şekilde gerçekleşmektedir:



Şekil 7.7 5-Şehir Gezgin Satıcı Problemine üretilen çözüm.

SONUÇ VE ÖNERİLER

"Yapay Sinir Ağları ve Gezgin Satıcı Problemine Uygulanmaları" isimli bu çalışma, adından da anlaşılabileceği şekilde, iki temel kavram üzerinde durmaktadır: (1) Yapay Sinir Ağları, (2) Gezgin Satıcı Problemi. Buradaki her iki kavram da basitliklerinden ve kolay uygulanabilirliklerinden dolayı ve belki de merak uyandıran isimlerinden dolayı popülerlik kazanan ve gelişen kavramlardır.

Bu kavramlardan Gezgin Satıcı Problemi, gerçek hayatta karşılaşılan optimizasyon problemlerinin büyük bir çoğunluğunu kapsayan kombinatoryal optimizasyonun en popüler problemlerinden bir tanesidir. Gezgin Satıcı Problemi aynı zamanda, tarihi konumu açısından, optimizasyon alanında geliştirilen yeni fikirlerin, yeni yaklaşımların kendilerini ispatlamaya çalıştıkları en önemli çalışma alanlarından bir tanesidir. Yapay Sinir Ağları kavramı ise, optimizasyon problemlerinin çözümüne yönelik geliştirilen bu yeni yaklaşımlardan biri olarak karşımıza çıkmaktadır.

Çalışma konusuna bakıldığında akla ilk gelen sorular, "Diğer yaklaşımlarla belirli bir dereceye getirilen, Gezgin Satıcı Problemine çözüm üretme sürecini Yapay Sinir Ağları ne şekilde geliştirilmiştir?; "Yapay Sinir Ağları bu sürece ne şekilde nokta koymuştur ya da ne gibi katkılar getirmiştir?" şeklinde kesin yargılar içeren sorulardır. Ancak ben, bu şekilde bir yaklaşım içinde olmamak gerektiğini düşünüyorum. Bunun nedeni, gerek Yapay Sinir Ağları ve gerekse onların esin kaynağı olan biyolojik sinir ağları hakkında sahip olduğumuz bilgi miktarının son derece az olmasıdır. Ayrıca, Gezgin Satıcı Probleminin bir uygulama problemi değil; geliştirilen yaklaşımların kendilerini ispat etmeye çalıştıkları bir teori problemi olduğunu da unutmamak gerekir. Yine Gezgin Satıcı Probleminin optimal çözüm üretilemeyen bir problem olduğunu ve söz konusu probleme üretilen çözümlerin

kalitesinde iyileştirmeler yapma sürecinin asla sonlanmayacak bir süreç olduğunu da göz ardı etmemek gerekir.

Elbette ki, Yapay Sinir Ağlarının diğer yaklaşımlara göre üstünlükleri ve Gezgin Satıcı Problemine katkıları incelenmesi ve ortaya konulması gereken bir konudur ancak unutulmamalıdır ki Yapay Sinir Ağlarının uygulama alanlarına ilişkin tüm potansiyelinin hala çok uzağında bulunulmaktadır. Bu şekilde bir bakış açısının yerine, Yapay Sinir Ağları kombinatoryal optimizasyon problemlerinin çözümüne yönelik kullanılan yaklaşımlardan biri olarak görülmeli ve "Yapay Sinir Ağlarının, kendini ispat etmesi adına, Gezgin Satıcı Problemine getirdiği çözüm yaklaşımları nelerdir?" sorusu göz önünde tutularak bu çalışma ele alınmalıdır.

Gezgin Satıcı Problemi aynı zamanda, "Np-complete" bir optimizasyon problemidir. Bu tip problemlerin çözümünde, optimum sonuca ulaşmada karşılaşılan zorluklardan dolayı, araştırmacılar şu iki alternatif yoldan birini izlemek durumunda kalmaktadırlar. (1) Hızlı bir biçimde, optimum çözüme yakın çözümler üretecek sezgisel yöntemler aramak; (2) Genele yönelik olmayan ancak pratik alanlardaki özel durumlar için iyi sonuçlar üretecek algoritmalar geliştirmek. Elinizdeki çalışmada, birinci yola ilişkin ortaya konan ve optimum sonuca yakın, hızlı çözümler üreten sezgisel yaklaşımlardan biri olarak Yapay Sinir Ağları ele alınmaktadır.

Yapay Sinir Ağlarının, girdi setleri uzayından çıktı setleri uzayına doğrusal olmayan, karmaşık ilişkileri kurabilecek şekilde nasıl eğitilebildikleri ve girdi setlerinin yetersiz ya da gereksiz verilerle dolu olduğu durumlarda, doğru eşleştirmeleri yapabilecek çağrışımlı bir hafıza olarak nasıl tasarlanabildikleri düşünüldüğünde; modelleme, analiz etme, tahminde bulunma, sistem performansını optimize etme vb. amaçlarla Yapay Sinir Ağları kullanımının ne kadar uygun olduğu görülmektedir.

Ancak, Yapay Sinir Ağlarının değer tahmininde bulunma, veri tanımlama ve sınıflandırma gibi problemlerin çözümündeki gücü, kombinatoryal

optimizasyon problemlerine uygulandığında aynı başarıyı gösterememektedir. Günümüzde, diğer yaklaşımlarla rekabet edebilecek sonuç kalitelerinin elde edildiği başarılı uygulamalarına rağmen, Yapay Sinir Ağlarının kombinatoryal optimizasyon problemlerinin çözümündeki ünü düşük kalmıştır. Bu durumu iki temel nedene bağlamak mümkündür (*Smith, 1999; s.15*):

1. Yapılan araştırmaların ilk yıllarından itibaren başlayan tartışmalar birçok araştırmacının cesaretini kırmıştır.
2. Araştırmacılar, biyolojik sinir ağlarının davranışlarını, uygun teknolojik (hardware) gelişimleri beklerken dijital işlemcilerde simüle etmek zorunda kaldılar. Dolayısıyla, yapılan simülasyonlar, Yapay Sinir Ağlarının uygulama potansiyelini değerlendirecek şekilde tasarlandığından, diğer yaklaşımlarla rekabet edemeyecek büyük işlem süreleri ile sonuçlanmıştır.

Maalesef, Yapay Sinir Ağlarının optimizasyon problemlerinin çözümüne yönelik ünü kurtarılmadıkça, bu alanda yapılan ve yapılacak başarılı uygulamalar hakettikleri ilgiyi göremeyeceklerdir. Bu nedenle, elinizdeki çalışmada da yapıldığı şekilde, optimizasyon problemlerinin çözümünde kullanılan Yapay Sinir Ağları yaklaşımlarının mevcut durumlarının ve potansiyelinin ortaya konması son derece önem taşımaktadır. 15 yılı aşan bir süreçte yapılan çalışmalarda, başlangıçtaki olurlu olmayan çözümlerden ve düşük kalitede elde edilen sonuçlardan günümüzdeki diğer sezgisel yaklaşımlarla rekabet edebilecek kalitede üretilen sonuçlara ulaşılmıştır. Ancak, önümüzde hala, ele alınması ve geliştirilmesi gereken çok sayıda çalışma alanı bulunmaktadır. Bu çalışma alanlarını şu şekilde sıralamak mümkündür (*Smith, 1999; s.28*):

Bu bağlamda öncelikle, asla sona ermeyecek bir süreç olan üretilen sonuçların kalitesindeki geliştirme sürecinde, Kaotik Sinir Ağları gibi alternatif sinir ağı modellerinin araştırılması çalışmalarına devam edilmelidir. Yine bu süreçte, Yapay Sinir Ağları bir hibrit yaklaşım içinde, mesela Genetik algortimalar ile birlikte kullanılarak elde edilen sonuçların kalitesinin yükseltilmesine çalışılabilir.

Daha çok sayıda gerçek hayattaki pratik uygulamaların çözümünde sinir ağı modelleri kullanılmalı ve Yapay Sinir Ağlarının uygulama potansiyeli açığa çıkarılmalıdır. Günümüzdeki uygulamalar çoğu Gezgin Satıcı Probleminin çözümüne yöneliktir. Bu nedenle, farklı problemler için uygulamalar yapılarak problem çeşitliliğini sağlamak gerekmektedir.

Ayrıca, gerçek hayatta karşılaşılan problemlerin çözümünde sinir ağı yaklaşımlarının performanslarının diğer yaklaşımlarla karşılaştırıldığı ve değerlendirildiği literatür sayısı son derece azdır. Yapılacak çalışmalarda üretilen sonuçların kalitesinde yapılacak değerlendirmeler sayesinde, sinir ağlarının hem optimizasyon problemlerine çözüm üretmedeki yetenekleri ve diğer yaklaşımlar üzerine avantajları açığa çıkarılmış olacak hem de Yapay Sinir Ağlarına bu alanda ihtiyacı olan popülerlik kazandırılmış olacaktır.

Gezgin Satıcı Problemine Çözüm Yaklaşımlarının Karşılaştırılması

"Gezgin Satıcı Problemine Çözüm Yaklaşımları" başlığı altında ele alındığı şekilde, geliştirilen algoritmaların ve sezgisel yaklaşımların ürettikleri sonuçların kalitesindeki yükseklikten dolayı, Gezgin Satıcı Problemini diğer kombinatoryal optimizasyon ("Np-hard") problemlerinden ayrı tutmak mümkündür.

Diğer taraftan, Gezgin Satıcı Problemine klasik yaklaşımlarla üretilen çözümler optimum sonuca oldukça yakındır ve geliştirilen yeni yaklaşımlarla ortaya konan katkılar pratik bir anlamdan çok istatistiksel bir anlam ifade etmektedir. Dolayısıyla, Yapay Sinir Ağları gibi yeni yaklaşımların elde edilen çözümlerin kalitesi açısından getirdikleri katkılardan bahsetmek söz konusu değildir. Ancak yine de, çözüm öncesi gerek modelleme ve programlama süreçlerinde gerekse veri toplama ve set oluşturma süreçlerinde Yapay Sinir Ağlarının ve diğer yeni yaklaşımların getirdiği katkılar göz ardı edilemez. Tüm bunların yanında, Yapay Sinir Ağlarının uygulama potansiyelinin hala çok uzağında bulunulduğu da düşünüldüğünde, Yapay

Sinir Ağlarının kombinatoryal optimizasyon problemlerinin çözümüne yönelik kullanımı ilgi gösterilmesi gereken bir çalışma alanı olarak karşımıza çıkmaktadır.

Hopfield ve Tank (*1985*), Gezgin Satıcı Probleminin bir Hopfield ağ yapısı kullanılarak çözülebileceğini gösterdikleri çalışmalarında, bir 10-şehir Gezgin Satıcı Problemi için seçtikleri 20 rassal başlanıç noktası ile 16 adet geçerli tur tanımına yakınsamışlar ve bu geçerli turların yarıya yakını da optimal sonucu vermiştir. Küçük boyutta bir problem için elde edilen bu sonuç oldukça cesaret vericidir. Ancak, ardından yaptıkları 30-şehir Gezgin Satıcı Problemi çalışmasında geçerli turları üretecek uygun parametre değerlerini saptayamadıklarından tatmin edici bir sonuca ulaşamamışlardır. Üstelik, sonraki yıllarda farklı araştırmacılarca yapılan çalışmalarda, aynı 10-şehir Gezgin Satıcı Problemi için bile, Hopfield ve Tank'in elde ettiği kalitede sonuçlara ulaşılamamıştır. Geçen 15 yılı aşan zaman dilimi içinde, Hopfield-Tank sinir ağı modeli üzerinde, elde edilen sonuçların kalitesini yükseltmek adına yapılan geliştirme çalışmalarında, 200 şehire kadar Gezgin Satıcı Problemleri için olurlu çözümlere ulaşılabilmiş ancak, elde edilen tur uzunlukları ve harcanan işlem süreleri açısından klasik sezgisel yaklaşımların ürettikleri sonuçların kalitesine ulaşılamamıştır.

Durbin ve Willshaw (*1987*), Hopfield ve Tank'in çalışmalarında ele aldıkları 30-şehir Gezgin Satıcı Problemini geliştirdikleri Elastik ağ yapısı yaklaşımı ile çözmüşler ve 100 şehire kadar Gezgin Satıcı Problemleri için Benzetimli Tavlama gibi yeni yaklaşımlarla rekabet edebilecek kalitede sonuçlar elde etmişlerdir. Daha sonra yapılan ve 1.000 şehire kadar çıkarılan uygulamaları neticesinde, yine de, Elastik ağ yapısı yöntemi ile klasik sezgisel yaklaşımların ürettikleri sonuçların kalitesine ulaşılamamıştır. Durbin ve Willshaw'ın çalışmalarıyla çakışacak şekilde Fort (*1988*), Gezgin Satıcı Probleminin çözümüne yönelik bir Öz-düzenlemeli Harita yaklaşımı geliştirmiştir. Bu çalışmada, aynı 30-şehir Gezgin Satıcı Problemi için daha kısa sürede sonuca ulaşılırken elde edilen sonuçların kalitesinde benzer seviye tutturulamamıştır. Sonraki yıllarda yapılan çalışmalarda, bu iki tekniğin üstün tarafları bir araya getirilmeye çalışılmış ve yapılan geliştirmeler neticesinde önce 10.000 ve

daha sonra 30.000-şehir Gezgin Satıcı Problemine çözüm üretilebilmiştir. Ancak yine de, elde edilen sonuçların kalitesinde klasik sezgisel yaklaşımların ürettikleri sonuçların kalitesine ulaşılamamıştır.

Buraya kadar anlatılan, Yapay Sinir Ağlarının optimizasyon problemlerinin çözümüne yönelik öne sürdüğü yaklaşımların performanslarının değerlendirilmesi süreci, farklı bir şekilde, karşılaştırılmalı olarak aşağıdaki tabloda yapılmaktadır:

Tablo 8.1 Optimizasyon problemlerine Yapay Sinir Ağları yaklaşımlarının karşılaştırılması.

***n**-şehir bir Gezgin Satıcı Problemi için*	**Hopfield-Tank sinir ağı modeli**	**Elastik Ağ Yapısı Yaklaşımı**	**Öz-düzenlemeli Haritalar**
Ağ yapısındaki İşlemci Birim sayısı	n^2	2*n*	2*n*
Çözüm üretmede harcanan işlem süreleri	*Kötü*	*Orta*	*İyi*
Elde edilen sonuçların kalitesi (tur uzunlukları)	*Kötü*	*En iyi*	*İyi*
Çözüm üretilebilen maksimum şehir sayısı	*200*	*1.000*	*30.000*
Asimetrik problemler için çözüm imkanı	*Mümkün*	*Mümkün değil*	*Çalışılıyor*

Ayrıca yukarıdaki karşılaştırmalara ek olarak, Sinir ağı yaklaşımlarının Benzetimli Tavlama gibi optimizasyon alanındaki diğer yeni yaklaşımlar ile de karşılaştırılmasında günümüzdeki durum şu şekilde ifade edilebilir:

Sinir ağı yaklaşımları bir çok durumda, Benzetimli Tavlama ile elde edilen çözüm kalitesine ulaşabilmekte, hatta bazı durumlar için daha iyi sonuçlar üretebilmektedirler.

KAYNAKLAR

Altınel, İ.K. Aras, Necati ve Oommen, B.J.; 2000. Fast, efficient and accurate solutions to the Hamiltonian path problem using Neural Approaches. *Computers and Operations Research*; **27**: 461-494.

Angeniol, B. De La Croix, G. ve Le Texier, J.Y.; 1988. Self-organizing Feature Maps and the Travelling Salesman Problem. *Neural Networks*; **1**: 289-293.

Applegate, D. Bixby, R. Chvàtal, V. ve Cook, W.; 1994. *The Traveling Salesman Problem: A case study in Local Optimization: s.3, Johnson D.S.- McGeoch L.A. (private communication)*.

Blum, Adam; 1992. Neural Networks in C++ "an object-oriented framework for building connectionist systems". John Wiley&Sons, Inc.

Bose, N.K. ve Liang, P.; 1996. Neural Network Fundamentals with Graphs, Algorithms and Applications. McGraw-Hill, Inc.

Budnik, A. ve Filipova, T.; 1996. Elastic Net Method for the Travelling Salesman Problem. http://nuweb.jinr.dubna.su/~filipova/tsp.html

Cerny, V.; 1985. A Thermodynamical Approach to the TSP: An Efficient Simulation Algorithm. *J.Optimization Theory and Appl.*; **45**: 41-51.

Clarke, G. ve Wright, J.W.; 1964. Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research*; **12**: 568-581.

Cohen, M.A. ve Grossberg, S.G.; 1983. Absolute Stability of Global Pattern Formation and Parallel Memory Storage by Competitive Neural Networks. *IEEE Transactions on Systems, Man and Cybernetics*; **13**: 815-826.

Crowder, H. ve Padberg, M.; 1980. Solving large-scale symmetric travelling salesman problems to optimality. *Management. Science*; **26**: 495-509.

De Falco, I. Cioppa, A.D. Natale, P. ve Tarantino, E.; 1997. Artificial Neural Networks Optimization by means of Evolutionary Algorithms. *Soft Computing in Engineering Design and Manufacturing*; Springer-Verlag, Inc.

Demuth, Howard ve Beale, Mark; 1997. Neural Network Toolbox for use with MATLAB User's Guide. Version 4; Mathworks, Inc.

Durbin, R. and Willshaw, D.; 1987. An Analogue Approach to the Traveling Salesman Problem Using an Elastic Net Method. *Nature*; **326** (16): 689-691.

Favata, F. ve Walker, R.; 1991. A Study of the Application of Kohonen-type Neural Networks to the Travelling Salesman Problem. *Biological Cybernetics*; **64**: 463-468.

Fort, J.C.; 1988. Solving a Combinatorial Problem via Self-organizing Process: An application of the Kohonen algorithm to the Travelling Salesman Problem. *Biological Cybernetics*; **59**: 33-40.

Foulds, L.R.; 1984. Combinatorial Optimization for Undergraduates. Springer-Verlag, Inc.

Fu, Limin; 1994. Neural Networks in Computer Intelligence. McGraw-Hill, Inc.

Gee, A.H. ve Prager, R.W.; 1995. Limitations of Neural Networks for Solving Traveling Salesman Problems. *IEEE Transactions on Neural Networks*; **6**: 280-282.

Goldberg, D.E.; 1989. Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley Publishing Company, Inc.

Hebb, D.O.; 1949. Organization of Behavior. John Wiley&Sons, Inc.

Hopfield, J.J. ve Tank, D.W.; 1985. Neural Computation of Decisions in Optimization Problems. *Biological Cybernetics*; **52**: 141-152.

Johnson, D.S. ve McGeoch L.A.; 1997. The Traveling Salesman Problem: A Case Study in Local Optimization. *Local Search in Combinatorial Optimization: pages: 215-310, Aarts E.H.L. - Lenstra J.K. (editors)*; John Wiley and Sons, Inc.

Kamgar-Parsi, B. ve Kamgar-Parsi, B.; 1992. Dynamical Stability and Parameter Selection in Neural Optimization. *Proceedings International Joint Conference on Neural Networks*; **IV**: 566-571.

Khanna, Tarun; 1990. Foundations of Neural Networks. Addison-Wesley Publishing Company, Inc.

Kirkpatrick, S. Gelatt, C.D. ve Vecchi, M.P.; 1983. Optimization by Simulated Annealing. *Science*; **220**: 671-680.

Kohonen, T.; 1982. Self-organized Formation of Topologically Correct Feature Maps. *Biological Cybernetics*; **43**: 59-69.

Lai, W.K. ve Coghill, G.G.; 1992. Genetic Breeding of Control Parameters for the Hopfield-Tank Neural Net. *Proceedings International Joint Conference on Neural Networks*; **IV**: 618-623.

Lin, S. ve Kernighan, B.W.; 1973. An effective heuristic algorithm for the travelling salesman problem. *Operations Research*; **21**: 498-516.

Liong, S.Y. Khu, S.T. ve Chan, W.T.; 2001. Derivation of Pareto Front with Genetic Algorithm and Neural Network. *Journal of Hydrologic Engineering*; **January/February**: 52-61.

Lorenzo, P.; 1998. Travelling Salesman Problem. http://www.patol.com/java/TSP/index.html

Masters, Timothy; 1993. Practical Neural Network Recipes in C++. Academic Press.

May, G.S.; 1994. Manufacturing ICs the Neural way. *IEEE Spectrum*. **31** (9): 47-51.

Miguel de Campos; 2001. Finding solutions of the Travelling Salesman Problem using a neural net (Hopfield net). http://www.ineti.pt/proj/fractal/neural/hope.html

Padberg, M. ve Rinaldi, G.; 1987. Optimization of a 532-city symmetric travelling salesman problem by branch and cut. *Operations Research Letters*; **6**: 1-7.

Ross, Timothy J.; 1995. Fuzzy Logic with Engineering Applications. McGraw-Hill, Inc.

Smith, S. Palaniswami, M. ve Krishnamoorthy, M.; 1996. A Hybrid Neural Approach to Combinatorial Optimization. *Computers and Operations Research*; **23** (6): 597-610.

Smith, K; 1996. An argument for abandoning the Traveling Salesman Problem as a neural-network benchmark. *IEEE Transactions on Neural Networks*; **7**: 1542-1544.

Smith, K. Palaniswami, M. ve Krishnamoorthy, M.; 1998. Neural Techniques for Combinatorial Optimization with Applications. *IEEE Transactions on Neural Networks*; **9** (6): 1301-1318.

Smith, K.; 1999. Neural Networks for Combinatoryal Optimization: A Review of More Than a Decade of Research. *Journal on Computing*; **11** (1): 15-34.

Thompson, B. ve Thompson, B.; 1987. Neurons, Analog Circuits, and the Traveling Salesperson-Hopfield.com. *AI EXPERT*; **2** (7); Miller Freeman Public., Inc.

Trenaman, A. ve diğerleri; 1998. Hopfield Net, http://www.cs.may.ie/~trenaman/nnets/hopfield/index.html

Van den Bout, D.E. ve Miller, T.K.; 1988. A Travelling Salesman Objective Function that works. *Proceedings International Joint Conference on Neural Networks*; **II**: 299-303.

Wilson, G.V. ve Pawley, G.S.; 1988. On the Stability of the TSP algorithm of Hopfield and Tank. *Biological Cybernetics*; **58**: 63-70.

ÖZGEÇMİŞ

Murat YILDIRIMHAN, 14.Ağustos.1977 tarihinde Trabzon'un Çaykara ilçesinde doğdu. Tahsil hayatının büyük bir bölümünü Bursa'nın farklı ilçelerinde geçirdi. İlkokulu Karacabey ve Mustafakemalpaşa, ortaokulu Keles ve liseyi İnegöl'de okudu. 1994 yılında yapılan üniversite sınavları neticesinde İ.T.Ü. İşletme Fakültesi Endüstri Mühendisliği bölümünü kazandı. Bir yıl İngilizce Hazırlık ile birlikte 1999 yılında lisansını tamamlayarak aynı yıl İ.T.Ü. Fen Bilimleri Enstitüsü Endüstri Mühendisliği programında yüksek lisansa başladı. Elinizdeki, "Yapay Sinir Ağlarının Gezgin Satıcı Problemine Uygulanmaları" isimli çalışma ile neticelenen yüksek lisans öğrenimi sırasında özel bir bankanın genel müdürlüğünde çalıştı.

EK - 1

Hopfield ağ yapısı yaklaşımının, Gezgin Satıcı Probleminin çözümüne yönelik nasıl uygulandığı ile ilgili olarak Thompson ve Thompson (*1987*) tarafından geliştirilen bilgisayar uygulamasının program kodları ve çıktıları:

```
PROGRAM traveling_salesperson ;

CONST
  max_city= 'E' ;
  max_position = 5 ;
  a = 500.0 ;
  b = 500.0 ;
  c = 200.0 ;
  d = 300.0 ;
  u0= 0.02 ;
  n = 7 ;
  h = 0.01 ;

TYPE
  cities = 'A' .. max_city ;
  positions = 1 .. max_position ;

VAR
  u : ARRAY [cities,positions] OF real ;
  dist : ARRAY [cities,cities] OF real ;

FUNCTION v(city : cities ; position : positions) : real ;

  FUNCTION tanh(r : real) : real ;
    VAR
      r1,r2 : real ;
    BEGIN
      IF r > 20.0
       THEN tanh := 1.0
      ELSE IF r < -20.0
       THEN tanh := -1.0
      ELSE
       BEGIN
        r1 := exp(r) ;
        r2 := exp(-r) ;
        tanh := (r1 - r2) / (r1 + r2) ;
       END ;
     END ; (* tanh *)

  BEGIN
   v := (1.0 + tanh(u[city,position] / u0)) / 2.0 ;
  END ; (* v *)
```

```
FUNCTION f(city : cities ; position : positions) : real ;

FUNCTION col_sum(cty : cities) : real ;
    VAR
     col : positions ;
     sum : real ;
    BEGIN
     sum := 0.0 ;
     FOR col := 1 TO max_position DO
      IF col <> position
       THEN sum := sum + v(cty,col) ;
     col_sum := sum ;
    END ; (* col_sum *)

FUNCTION row_sum(p : positions) : real ;
    VAR
     row : cities ;
     sum : real ;
    BEGIN
     sum := 0.0 ;
     FOR row := 'A' TO max_city DO
      IF row <> city
       THEN sum := sum + v(row,p) ;
     row_sum := sum ;
    END ; (* row_sum *)

FUNCTION matrix_sum : real ;
   VAR
    row : cities ;
    col : positions ;
    sum : real ;
   BEGIN
    sum := 0.0 ;
    FOR row := 'A' TO max_city DO
     FOR col := 1 TO max_position DO
      sum := sum + v(row,col) ;
    matrix_sum := sum ;
   END ; (* matrix_sum *)

FUNCTION dist_sum : real ;
   VAR
    c : cities ;
    sum : real ;
   BEGIN
    sum := 0.0 ;
    IF position = max_position
     THEN
      FOR c := 'A' TO max_city DO
       sum := sum + dist[city,c] * (v(c,1) + v(c,position - 1))
    ELSE IF position = 1
```

```
      THEN
       FOR c := 'A' TO max_city DO
        sum := sum + dist[city,c] * (v(c,position + 1)
             + v(c,max_position))
     ELSE
      FOR c := 'A' TO max_city DO
       sum := sum + dist[city,c] * (v(c,position + 1)
            + v(c,position - 1)) ;
     dist_sum := sum ;
    END ; (* dist_sum *)

   BEGIN
   f := -u[city,position] - a * col_sum(city) - b
        * row_sum(position)- c * (matrix_sum - n) - d * dist_sum ;
   END ; (* f *)

PROCEDURE iterate ;
  CONST
   tol = 1.0E-05 ;
  VAR
   step : integer ;
   c1 : cities ;
   i : positions ;
   nr : real ;
   u_old : ARRAY [cities,positions] OF real ;
   ch : char ;

FUNCTION norm : real ;
   VAR
    cx : cities ;
    ix : positions ;
    max,max_comp : real ;
   BEGIN
    max := 0.0 ;
    FOR cx := 'A' TO max_city DO
     FOR ix := 1 TO max_position DO
      BEGIN
       IF abs(u_old[cx,ix] - u[cx,ix]) > max
        THEN max := abs(u_old[cx,ix] - u[cx,ix]) ;
       IF abs(u[cx,ix]) > max_comp
        THEN max_comp := abs(u[cx,ix]) ;
      END ;
    norm := max / max_comp ;
   END ; (* norm *)

PROCEDURE print_matrix ;
   VAR
    c1 : cities ;
    i : positions ;
    vv : real ;
    t : ARRAY [1 .. max_position] OF char ;
    t_count : integer ;
```

```
PROCEDURE write_tour ;
    VAR
     i : positions ;
     t_dist : real ;
    BEGIN
     t_dist := 0.0 ;
     FOR i := 1 TO max_position - 1 DO
      t_dist := t_dist + dist[t[i],t[i+1]] ;
     t_dist := t_dist + dist[t[max_position],t[1]] ;
     write(output,'Tour: ') ;
     FOR i := 1 TO max_position DO
      write(output,t[i]) ;
     writeln(output,'   dist = ',t_dist) ;
    END ; (* write_tour *)

PROCEDURE matrix_heading ;
    VAR
     i : positions ;
    BEGIN
     write(output,'  ') ;
     FOR i := 1 TO max_position DO
      write(output,i : 12) ;
     writeln ;
    END ; (* matrix_heading *)

    BEGIN
    t_count := 0 ;
    FOR i := 1 TO max_position DO
     t[i] := chr(0) ;
    writeln(output) ;
    writeln(output,'Step: ',step,' norm = ',nr) ;
    writeln(output) ;
    writeln(output,'Input Voltages') ;
    matrix_heading ;
    FOR c1 := 'A' TO max_city DO
     BEGIN
      write(output,c1,'    ') ;
      FOR i := 1 TO max_position DO
       write(output,u[c1,i] : 12 : 5) ;
      writeln(output) ;
     END ;
    writeln(output) ;
    writeln(output,'Output Voltages') ;
    matrix_heading ;
    FOR c1 := 'A' TO max_city DO
     BEGIN
      write(output,c1,'    ') ;
      FOR i := 1 TO max_position DO
       BEGIN
        vv := v(c1,i) ;
        write(output,vv : 12 : 5) ;
        IF (vv>0.8) AND (t_count < max_position) AND (t[i] = chr(0))
         THEN
```

```
          BEGIN
           t_count := t_count + 1 ;
           t[i] := c1 ;
          END ;
        END ;
       writeln(output) ;
      END ;
     IF t_count = max_position
      THEN write_tour ;
   END ; (* print_matrix *)

   BEGIN
   step := 0 ;
   REPEAT
     step := step + 1 ;
     move(u,u_old,sizeof(u)) ;
     FOR c1 := 'A' TO max_city DO
      FOR i := 1 TO max_position DO
       u[c1,i] := u[c1,i] + h * f(c1,i) ;
     nr := norm ;
     IF ((step MOD 10) = 0) OR (step < 10)
      THEN print_matrix ;
    UNTIL keypressed OR (nr < tol) ;
    IF keypressed
     THEN read(kbd,ch) ;
    print_matrix ;
   END ; (* iterate *)

PROCEDURE initialize ;
   TYPE
    location = RECORD
                x : real ;
                y : real ;
               END ;
    city_array = ARRAY [cities] OF location ;
   CONST
    u00 = -0.01386 ;
   VAR
    c1,c2 : cities ;
    i : positions ;
    city_loc : city_array ;
    ch : char ;
   BEGIN
    randomize ;
    FOR c1 := 'A' TO max_city DO
     BEGIN
      city_loc[c1].x := random ;
      city_loc[c1].y := random ;
     END ;
     FOR c1 := 'A' TO pred(max_city) DO
     BEGIN
      dist[c1,c1] := 0.0 ;
      FOR c2 := succ(c1) TO max_city DO
```

```
       BEGIN
       dist[c1,c2] := sqrt(sqr(city_loc[c1].x - city_loc[c2].x) +
                           sqr(city_loc[c1].y - city_loc[c2].y)) ;
       dist[c2,c1] := dist[c1,c2] ;
       END ;
     END ;
    dist[max_city,max_city] := 0.0 ;
    FOR c1 := 'A' TO max_city DO
     FOR i := 1 TO max_position DO
      u[c1,i] := u00 + (((2 * random - 1.0) / 10.0) * u0) ;
    clrscr ;
    writeln('TSP         [c] 1987 Knowledge Garden Inc.') ;
    writeln('                     473A Malden Bridge Rd') ;
    writeln('                     Nassau, NY 12123') ;
    writeln ;
    writeln('Press <Space Bar> to begin - Press again to stop
iterating.') ;
    read(kbd,ch) ;
   END ; (* initialize *)

   BEGIN
    initialize ;
    iterate ;
   END.
```

Step: 1

Input Voltages

	1	2	3	4	5
A	-8.43176	-6.73021	-5.14208	-3.66349	-2.23470
B	-2.96753	-1.11288	0.15747	-5.44852	-4.03152
C	-1.26019	-1.05439	-3.48170	1.65450	-2.31212
D	0.17568	-6.03228	-10.69222	-8.25715	-2.70486
E	-3.08320	0.10076	-7.62895	-7.24145	-1.44345

Output Voltages

	1	2	3	4	5
A	0.00000	0.00000	0.00000	0.00000	0.00000
B	0.00000	0.00000	1.00000	0.00000	0.00000
C	0.00000	0.00000	0.00000	1.00000	0.00000
D	1.00000	0.00000	0.00000	0.00000	0.00000
E	0.00000	0.99996	0.00000	0.00000	0.00000

Step: 2

Input Voltages

	1	2	3	4	5
A	-9.15455	-11.24608	-7.37414	-5.17148	-0.72740
B	-8.50394	-5.85047	3.19446	-9.39395	-5.13549
C	-5.69938	-8.09817	-7.89868	6.24268	-2.94818
D	4.55374	-10.72067	-17.86475	-12.92350	-3.33700
E	-7.05208	2.91327	-12.00456	-12.73527	-2.50124

Output Voltages

	1	2	3	4	5
A	0.00000	0.00000	0.00000	0.00000	0.00000
B	0.00000	0.00000	1.00000	0.00000	0.00000
C	0.00000	0.00000	0.00000	1.00000	0.00000
D	1.00000	0.00000	0.00000	0.00000	0.00000
E	0.00000	1.00000	0.00000	0.00000	0.00000

Step: 3

Input Voltages

	1	2	3	4	5
A	-9.87027	-15.71707	-9.58404	-6.66447	0.76475
B	-18.52984	-12.54098	4.20092	-17.84472	-13.22850
C	-13.57066	-17.07181	-14.27158	7.30853	-10.57796
D	3.84911	-17.36248	-26.96570	-22.58203	-10.96289
E	-14.78882	3.69757	-18.33641	-21.98142	-10.54846

Output Voltages

	1	2	3	4	5
A	0.00000	0.00000	0.00000	0.00000	1.00000
B	0.00000	0.00000	1.00000	0.00000	0.00000
C	0.00000	0.00000	0.00000	1.00000	0.00000
D	1.00000	0.00000	0.00000	0.00000	0.00000
E	0.00000	1.00000	0.00000	0.00000	0.00000

Tour: DEBCA dist = 3.03235095514277E+0000

Step: 4

Input Voltages

	1	2	3	4	5
A	-17.57883	-27.14336	-18.77183	-15.14253	0.24197
B	-28.45548	-19.16458	5.19732	-26.21097	-21.24059
C	-21.36322	-25.95571	-20.58077	8.36371	-18.13144
D	3.15153	-23.93787	-35.97564	-32.14398	-18.51252
E	-22.44820	4.47402	-24.60494	-31.13510	-18.51521

Output Voltages

	1	2	3	4	5
A	0.00000	0.00000	0.00000	0.00000	1.00000
B	0.00000	0.00000	1.00000	0.00000	0.00000
C	0.00000	0.00000	0.00000	1.00000	0.00000
D	1.00000	0.00000	0.00000	0.00000	0.00000
E	0.00000	1.00000	0.00000	0.00000	0.00000

Tour: DEBCA dist = 3.03235095514277E+0000

Step: 5

Input Voltages

	1	2	3	4	5
A	-25.21031	-38.45539	-27.86776	-23.53581	-0.27557
B	-33.73716	-23.72194	8.18376	-29.94886	-22.17255
C	-25.60148	-32.75077	-24.82685	12.88472	-18.60939
D	7.49968	-28.44750	-42.89548	-36.57155	-18.98666
E	-26.22372	7.24271	-28.81079	-36.38999	-19.40229

Output Voltages

	1	2	3	4	5
A	0.00000	0.00000	0.00000	0.00000	0.00000
B	0.00000	0.00000	1.00000	0.00000	0.00000
C	0.00000	0.00000	0.00000	1.00000	0.00000
D	1.00000	0.00000	0.00000	0.00000	0.00000
E	0.00000	1.00000	0.00000	0.00000	0.00000

Step: 6

Input Voltages

	1	2	3	4	5
A	-25.76547	-42.65429	-29.87272	-24.84515	1.21206
B	-43.51073	-30.23374	9.14033	-38.19408	-30.09519
C	-33.27374	-41.47789	-31.03048	13.88414	-26.08256
D	6.76559	-34.91204	-51.74613	-45.99360	-26.45606
E	-33.76875	7.98372	-34.97458	-45.39959	-27.28050

Output Voltages

	1	2	3	4	5
A	0.00000	0.00000	0.00000	0.00000	1.00000
B	0.00000	0.00000	1.00000	0.00000	0.00000
C	0.00000	0.00000	0.00000	1.00000	0.00000
D	1.00000	0.00000	0.00000	0.00000	0.00000
E	0.00000	1.00000	0.00000	0.00000	0.00000

Tour: DEBCA dist = 3.03235095514277E+0000

Step: 7

Input Voltages

	1	2	3	4	5
A	-33.31509	-53.81121	-38.85763	-33.14141	0.68481
B	-53.18656	-36.68041	10.08734	-46.35684	-37.93861
C	-40.86927	-50.11773	-37.17207	14.87357	-33.48100
D	6.03885	-41.31193	-60.50826	-55.32143	-33.85076
E	-41.23833	8.71731	-41.07673	-54.31909	-35.07992

Output Voltages

	1	2	3	4	5
A	0.00000	0.00000	0.00000	0.00000	1.00000
B	0.00000	0.00000	1.00000	0.00000	0.00000
C	0.00000	0.00000	0.00000	1.00000	0.00000
D	1.00000	0.00000	0.00000	0.00000	0.00000
E	0.00000	1.00000	0.00000	0.00000	0.00000

Tour: DEBCA dist = 3.03235095514277E+0000

Step: 8

Input Voltages

	1	2	3	4	5
A	-40.78920	-64.85656	-47.75269	-41.35470	0.16283
B	-62.76563	-43.06262	11.02487	-54.43798	-45.70359
C	-48.38884	-58.67117	-43.25225	15.85311	-40.80545
D	5.31937	-47.64782	-69.18278	-64.55599	-41.17152
E	-48.63321	9.44357	-47.11786	-63.14940	-42.80136

Output Voltages

	1	2	3	4	5
A	0.00000	0.00000	0.00000	0.00000	1.00000
B	0.00000	0.00000	1.00000	0.00000	0.00000
C	0.00000	0.00000	0.00000	1.00000	0.00000
D	1.00000	0.00000	0.00000	0.00000	0.00000
E	0.00000	1.00000	0.00000	0.00000	0.00000

Tour: DEBCA dist = 3.03235095514277E+0000

Step: 9

Input Voltages

	1	2	3	4	5
A	-48.18858	-75.79145	-56.55880	-49.48586	-0.35392
B	-67.70421	-47.38101	13.95303	-57.89360	-46.39093
C	-52.35685	-65.13908	-47.27162	20.29922	-41.05666
D	9.64584	-51.92036	-75.77055	-68.65944	-41.41906
E	-52.14688	12.16256	-51.09858	-68.08414	-43.44557

Output Voltages

	1	2	3	4	5
A	0.00000	0.00000	0.00000	0.00000	0.00000
B	0.00000	0.00000	1.00000	0.00000	0.00000
C	0.00000	0.00000	0.00000	1.00000	0.00000
D	1.00000	0.00000	0.00000	0.00000	0.00000
E	0.00000	1.00000	0.00000	0.00000	0.00000

Step: 10

Input Voltages

	1	2	3	4	5
A	-48.51396	-79.61699	-58.27686	-50.53570	1.13449
B	-77.13811	-53.65621	14.85191	-65.85937	-54.07138
C	-59.76155	-73.54231	-53.25080	21.22450	-48.30536
D	8.89030	-58.15017	-84.29244	-77.76061	-48.66414
E	-59.43268	12.85437	-57.03949	-76.77680	-51.08335

Output Voltages

	1	2	3	4	5
A	0.00000	0.00000	0.00000	0.00000	1.00000
B	0.00000	0.00000	1.00000	0.00000	0.00000
C	0.00000	0.00000	0.00000	1.00000	0.00000
D	1.00000	0.00000	0.00000	0.00000	0.00000
E	0.00000	1.00000	0.00000	0.00000	0.00000

Tour: DEBCA dist = 3.03235095514277E+0000

Step: 20

Input Voltages

	1	2	3	4	5
A	-103.74350	-167.97970	-126.68966	-112.62316	0.32423
B	-156.93916	-108.87686	27.54237	-131.76287	-112.14068
C	-122.58750	-148.90723	-105.66926	37.25159	-102.28751
D	12.27294	-112.94112	-160.78067	-153.37910	-102.61198
E	-120.44125	23.58462	-109.09567	-151.10297	-108.74859

Output Voltages

	1	2	3	4	5
A	0.00000	0.00000	0.00000	0.00000	1.00000
B	0.00000	0.00000	1.00000	0.00000	0.00000
C	0.00000	0.00000	0.00000	1.00000	0.00000
D	1.00000	0.00000	0.00000	0.00000	0.00000
E	0.00000	1.00000	0.00000	0.00000	0.00000

Tour: DEBCA dist = 3.03235095514277E+0000

Step: 30

Input Voltages

	1	2	3	4	5
A	-148.29184	-242.49307	-183.16071	-163.37368	1.13437
B	-225.63877	-157.28992	40.54689	-187.89378	-159.31123
C	-176.75107	-215.53842	-151.54806	54.40129	-145.76173
D	19.18052	-160.96556	-228.42774	-217.91874	-146.05517
E	-172.70850	34.81638	-154.64685	-215.41442	-155.55373

Output Voltages

	1	2	3	4	5
A	0.00000	0.00000	0.00000	0.00000	1.00000
B	0.00000	0.00000	1.00000	0.00000	0.00000
C	0.00000	0.00000	0.00000	1.00000	0.00000
D	1.00000	0.00000	0.00000	0.00000	0.00000
E	0.00000	1.00000	0.00000	0.00000	0.00000

Tour: DEBCA dist = 3.03235095514277E+0000

Step: 40

Input Voltages

	1	2	3	4	5
A	-193.96906	-315.27013	-239.62060	-214.66003	0.32748
B	-291.23293	-202.59800	50.78378	-242.12103	-207.30605
C	-228.38493	-277.32264	-194.56420	67.26187	-190.41364
D	21.58763	-205.92218	-291.13071	-280.12720	-190.67903
E	-222.87953	43.45000	-197.36669	-276.47801	-203.21807

Output Voltages

	1	2	3	4	5
A	0.00000	0.00000	0.00000	0.00000	1.00000
B	0.00000	0.00000	1.00000	0.00000	0.00000
C	0.00000	0.00000	0.00000	1.00000	0.00000
D	1.00000	0.00000	0.00000	0.00000	0.00000
E	0.00000	1.00000	0.00000	0.00000	0.00000

Tour: DEBCA dist = 3.03235095514277E+0000

Step: 50

Input Voltages

	1	2	3	4	5
A	-229.89022	-375.69990	-285.29343	-255.65400	1.13731
B	-347.09165	-242.04964	61.56600	-287.69972	-245.37709
C	-272.43237	-331.67501	-231.94306	81.54206	-225.46142
D	27.60455	-245.05597	-346.31398	-332.54745	-225.70143
E	-265.35185	52.78227	-234.47758	-328.80135	-240.99023

Output Voltages

	1	2	3	4	5
A	0.00000	0.00000	0.00000	0.00000	1.00000
B	0.00000	0.00000	1.00000	0.00000	0.00000
C	0.00000	0.00000	0.00000	1.00000	0.00000
D	1.00000	0.00000	0.00000	0.00000	0.00000
E	0.00000	1.00000	0.00000	0.00000	0.00000

Tour: DEBCA dist = 3.03235095514277E+0000

Step: 52

Input Voltages

	1	2	3	4	5
A	-240.85186	-391.27455	-298.09053	-267.57044	0.08958
B	-360.30471	-250.66313	62.40753	-298.97787	-256.70048
C	-282.78777	-343.09311	-240.16641	82.16507	-236.21577
D	25.74428	-253.60964	-357.88848	-345.40676	-236.45101
E	-275.60732	53.35088	-242.65049	-340.91098	-252.25737

Output Voltages

	1	2	3	4	5
A	0.00000	0.00000	0.00000	0.00000	0.99987
B	0.00000	0.00000	1.00000	0.00000	0.00000
C	0.00000	0.00000	0.00000	1.00000	0.00000
D	1.00000	0.00000	0.00000	0.00000	0.00000
E	0.00000	1.00000	0.00000	0.00000	0.00000

Tour: DEBCA dist = 3.03235095514277E+0000

EK - 2

Öz-düzenlemeli harita yaklaşımının, Gezgin Satıcı Probleminin çözümüne yönelik nasıl uygulandığı ile ilgili olarak Lorenzo (*1998*) tarafından geliştirilen bilgisayar uygulamasının program kodları ve çıktıları:

```
-----------------------------------------------------------------------

public class City{

        public double x,y;
        public int update,choose;

        public City(double x,double y){
                this.x = x;
                this.y = y;

                update = 0;
                choose = 0;
        }

        public double dist(City c){
                double dx = this.x - c.x;
                double dy = this.y - c.y;

                return Math.sqrt(dx*dx + dy*dy);
        }
}

-----------------------------------------------------------------------

public class geoNeuron{

        public double x,y;
        public double wx,wy;
        public int update,choose;

        public geoNeuron(double x,double y){
                this.x = x;
                this.y = y;

                this.wx = Math.random();
                this.wy = Math.random();

                update = 0;
                choose = 0;
        }
```

```
        public double dist(geoNeuron c){
                double dx = this.x - c.x;
                double dy = this.y - c.y;

                return Math.sqrt(dx*dx + dy*dy);
        }

        public double wdist(geoNeuron c){
                double dx = this.wx - c.wx;
                double dy = this.wy - c.wy;

                return Math.sqrt(dx*dx + dy*dy);
        }
}
```

```
import java.applet.*;
import java.util.*;
import java.awt.*;
import java.net.*;
import java.io.*;

public class TSP extends Applet implements Runnable {

        public int     NCITY = 5;
        public int     NGEONEURON;
        public static final double     COUNTRY = 1.00;
        public static final double     NEAR = 0.05;

  public static final Color bkC = new Color(0x000090);
  public static final Color bk2C = new Color(0x000050);
  public static final Color lnC = new Color(0xff0000);
  public static final Color ln2C = new Color(0xcccc00);
  public static final Color fgC = new Color(0xffffff);

        public Image   homeI,offscreen;
  public int imagewidth ,imageheight;
        public Thread  animator    = null;
        public boolean please_stop = false;
        Font mF = new Font("Courier", Font.BOLD, 12);
        Font sF = new Font("Courier", Font.BOLD, 8);
  public int counter;

        public City city[];
        public geoNeuron gn[];

        public double r[][];

        public double theta, phi, momentum;

        public Scrollbar cscroll;
```

```
// ------------------------ Init. section --------------------------

                public void kohonenInit(){
                        theta = 0.5;
        phi    = 0.5;
        momentum = 0.995;

                        NCITY = cscroll.getValue()/10;
                        NGEONEURON = NCITY*2;

   city = new City[NCITY];
      for(int i = 0; i<NCITY; i++)
      city[i] = new City(Math.random()*COUNTRY,
                Math.random()*COUNTRY);

   double alpha = 0.0;
   gn = new geoNeuron[NGEONEURON];
   for(int i = 0; i<NGEONEURON; i++){
      gn[i] = new geoNeuron(0.5+0.5*Math.cos(alpha),
               0.5+0.5*Math.sin(alpha));
      alpha += Math.PI *2.0 / (double)(NGEONEURON);
   }

   r = new double[NGEONEURON][NGEONEURON];

   makeR(theta);

                        counter = 0;
                }


// ------------------------ Problem section --------------------------

                public void makeR(double th){

   for(int i=0; i<NGEONEURON; i++){
      r[i][i]= 1.0;
      for(int j=i+1; j<NGEONEURON; j++){
             r[i][j] = Math.exp( -1.0 *
               ( gn[i].dist(gn[j])*gn[i].dist(gn[j]) )/(2.0*th*th));
             r[j][i] = r[i][j];

      }
   }
                }

public void run() {
      int idx,j;
      double x1,x2,mindist;
      int count = 0;
      while(!please_stop) {

            counter++;
```

```
// CHOSE A RANDOM PATTERN
            idx = (int)(Math.random()*NCITY);
            x1 = city[idx].x+(Math.random()*NEAR)-NEAR/2;
            x2 = city[idx].y+(Math.random()*NEAR)-NEAR/2;
            city[idx].choose++;

// SEARCH FOR MINIMAL
            mindist = 100000.0;
            j = -1;
            for(int i=0; i<NGEONEURON;i++){
               double d = (x1 - gn[i].wx)*(x1 - gn[i].wx)
                         + (x2 - gn[i].wy)*(x2 - gn[i].wy);
               if(d<mindist){
                       mindist = d;
                       j = i;
               }
            }
            gn[j].update++;

// UPDATE WEIGHTS
            for(int i=0; i<NGEONEURON;i++){
               gn[i].wx += (phi * r[i][j] * (x1 - gn[i].wx));
               gn[i].wy += (phi * r[i][j] * (x2 - gn[i].wy));
            }

// DECREASE LEARNING PARAMETERS
            phi *= momentum;
            theta *= momentum;

// RE-COMPUTE r MATRIX
            makeR(theta);

// PLOT RESULT EVERY 10 SESSIONS
            count = (count++)%10;
               if(count==0){
                       paint(this.getGraphics());

        try {Thread.sleep(10);} catch (InterruptedException e){};
            }
    }
    animator = null;
}

// --------------------- Functional  section -----------------------

public void init() {

    cscroll=new Scrollbar(Scrollbar.HORIZONTAL,NCITY*10,10,30,200);
    cscroll.setLineIncrement(10);
    cscroll.setPageIncrement(10);
    add(cscroll);

    kohonenInit();
}
```

```
private int toXReal(double val){int w = this.size().width;
          return (int)(val *((double)w/2.0-50.0) / COUNTRY +25.0);}
private int toYReal(double val){int h = this.size().height;
          return (int)(val *((double)h-50.0) / COUNTRY +25.0);}

private int to2XReal(double val){int w = this.size().width;
          return (int)((double)w/2.0 + val *((double)w/2.0-50.0) /
      COUNTRY +25.0);}
private int to2YReal(double val){int h = this.size().height;
          return (int)(val *((double)h-50.0) / COUNTRY +25.0);}

public void paintLeft(Graphics g) {
              Dimension size = this.size();
                            int w = size.width, h = size.height;

                            g.setFont(mF);

// CLEAR ALL
      g.setColor(bkC);
      g.fillRect(0, 0, w, h);

// DRAW GRID
      g.setColor(bk2C);
      for(double i=0; i<=COUNTRY; i+=(COUNTRY/20.0)){
g.drawLine(toXReal(0.0),toYReal(i),toXReal(COUNTRY),toYReal(i));
g.drawLine(toXReal(i),toYReal(0.0),toXReal(i),toYReal(COUNTRY));
      }

//DRAW PATH
      g.setColor(lnC);
      for(int i=0; i<NGEONEURON; i++){
g.drawLine( toXReal(gn[i].wx),toYReal(gn[i].wy),
toXReal(gn[(i+1)%NGEONEURON].wx),toYReal(gn[(i+1)%NGEONEURON].wy) );
g.drawString(""+i+"-"+(gn[i].update*100/counter)+"%",
toXReal(gn[i].wx),toYReal(gn[i].wy));
      }

      g.setColor(fgC);

// DRAW CITYS
      for(int i=0; i<NCITY; i++){
g.drawOval( toXReal(city[i].x)-4, toYReal(city[i].y)-4,8,8);
g.drawString(""+i+"-"+(city[i].choose*100/counter)+"%",
toXReal(city[i].x),toYReal(city[i].y)+8);
      }
}
            public void paintRight(Graphics g) {
                Dimension size = this.size();
                       int w = size.width, h = size.height;

// CLEAR ALL
       g.setColor(bkC);
       g.fillRect(0, 0, w, h);
                      g.setFont(sF);
```

```
// DRAW CITYS
       g.setColor(fgC);
       for(int i=0; i<NCITY; i++){
g.drawOval( to2XReal(city[i].x)-4, to2YReal(city[i].y)-4,8,8);

       }

       g.setColor(ln2C);

       for(int i=0; i<NGEONEURON; i++)
                for(int j=i+1; j<NGEONEURON; j++){
g.drawLine( to2XReal(gn[i].x),to2YReal(gn[i].y),
to2XReal(gn[j].x),to2YReal(gn[j].y));

g.drawString(""+r[i][j],to2XReal((gn[i].x+gn[j].x)/2),
to2YReal((gn[i].y+gn[j].y)/2));

                }
                        g.setFont(mF);
        g.setColor(fgC);

g.drawString("phi="+phi+" theta="+theta,to2XReal(0.0),
to2YReal(0.0)+20);
}

public void paint(Graphics g) {
        Dimension size = this.size();
                        int w = size.width, h = size.height;
                        this.setBackground(bkC);


if ((offscreen == null)||((imagewidth != w)||(imageheight != h))) {
            offscreen = this.createImage(w, h);
            imagewidth = w;
            imageheight = h;
        }

                Rectangle clip = new Rectangle(toXReal(0),
toYReal(0),toXReal(COUNTRY),toYReal(COUNTRY));

                Graphics goff = offscreen.getGraphics();
                goff.clipRect(clip.x, clip.y, clip.width,
clip.height);
            Graphics g1 = this.getGraphics();
                g1.clipRect(clip.x, clip.y, clip.width,
clip.height);

            paintLeft(goff);
            g1.drawImage(offscreen, 0, 0, this);

            clip = null;
            goff = null;
            g1 = null;
            System.gc();
```

```
// CLEAR ALL
            g.setColor(bkC);
            g.fillRect(w/2+30,0,w/2+130, 20);
            g.setColor(fgC);

            g.drawString("# of
                        city:"+cscroll.getValue()/10,w/2+30,20);
}

// Start the animation
    public void start() {
        animator = new Thread(this);

        animator.start();

}

// Stop it.
    public void stop() {
        if (animator != null) animator.stop();
        animator = null;
}

// Stop and start animating on mouse clicks.
    public boolean mouseDown(Event e, int x, int y) {

// if running, stop it.  Otherwise, start it.
      if (animator != null){
         please_stop = true;
      }else{
         please_stop = false;
                       animator = new Thread(this);

                       kohonenInit();
        animator.start();
      }
      return true;
    }

}
```

```
**************************************************************************
city coordinats

city[0]=(0.2678816374506062,0.5313091971473904)
city[1]=(0.5076256468799432,0.9492825988758083)
city[2]=(0.5254459340204259,0.33994624087863)
city[3]=(0.0932585512648979,0.13876087574136065)
city[4]=(0.9383494478483632,0.8825162598810746)

NGEONEURON coordinats

alpha= 0.0
gn[0]=(1.0,0.5)
gn[0].weight=(0.510133155423283,0.9237329150865935)

alpha= 0.6283185307179586
gn[1]=(0.9045084971874737,0.7938926261462366)
gn[1].weight=(0.6474168104116554,0.3358069675529367)

alpha= 1.2566370614359172
gn[2]=(0.6545084971874737,0.9755282581475768)
gn[2].weight=(0.6707383834431202,0.484740542046352)

alpha= 1.8849555921538759
gn[3]=(0.3454915028125263,0.9755282581475768)
gn[3].weight=(0.518071322699788,0.5158857316949699)

alpha= 2.5132741228718345
gn[4]=(0.09549150281252633,0.7938926261462367)
gn[4].weight=(0.48606950334411814,0.4496304120535798)

alpha= 3.141592653589793
gn[5]=(0.0,0.5000000000000001)
gn[5].weight=(0.39498161703981227,0.6993425787332356)

alpha= 3.7699111843077517
gn[6]=(0.09549150281252627,0.2061073738537635)
gn[6].weight=(0.16592983835288788,0.3847688290677569)

alpha= 4.39822971502571
gn[7]=(0.3454915028125262,0.024471741852423234)
gn[7].weight=(0.8775912248530651,0.5966796671536858)

alpha= 5.026548245743669
gn[8]=(0.6545084971874736,0.02447174185242318)
gn[8].weight=(0.9892267289855341,0.8217193774832369)

alpha= 5.654866776461628
gn[9]=(0.9045084971874737,0.20610737385376332)
gn[9].weight=(0.10288916854905983,0.6236294410076832)

start to calculate r
r[0][1]= 0.8261466278774511      r[0][2]= 0.5010832592258778
r[0][3]= 0.2700854214241597      r[0][4]= 0.16381508883516016
r[0][5]= 0.1353352832366127      r[0][6]= 0.16381508883516024
r[0][7]= 0.2700854214241597      r[0][8]= 0.5010832592258777
r[0][9]= 0.8261466278774509

r[1][2]= 0.826146627877451       r[1][3]= 0.5010832592258777
r[1][4]= 0.2700854214241597      r[1][5]= 0.16381508883516024
```

```
r[1][6]= 0.1353352832366127      r[1][7]= 0.16381508883516024
r[1][8]= 0.2700854214241597      r[1][9]= 0.5010832592258777
. . .
. .
.
r[7][8]= 0.8261466278774511      r[7][9]= 0.5010832592258778

r[8][9]= 0.826146627877451

r has been calculated
********************************************************************************

CHOSE A RANDOM PATTERN **1**
Choose a city
city is: 0
x1: 0.27925023503314406 x1: 0.27925023503314406

SEARCH FOR MINIMAL
d=0.21414921990454644      d=0.17046869040943588
d=0.1547024810132678       d=0.057081693374042966
d=0.0481106603690622       d=0.04460291570060861
d=0.03186139666951147      d=0.36348767980693086
d=0.5934903184504479       d=0.041293727525051734
mindist= 0.03186139666951147 for : 6

UPDATE WEIGHTS
gn[0].weight=(0.49122210236616703,0.8908837724225221)
gn[1].weight=(0.6225038465331032,0.3484523214281595)
gn[2].weight=(0.6386725505382739,0.48784818667605306)
gn[3].weight=(0.4858202756460771,0.516803440661944)
gn[4].weight=(0.43425266682613106,0.4679327321082006)
gn[5].weight=(0.34717607154760927,0.6263685708892656)
gn[6].weight=(0.22259003669301597,0.45372512748519817)
gn[7].weight=(0.630432529322773,0.5661129684144788)
gn[8].weight=(0.8113480612038214,0.7467979217781189)
gn[9].weight=(0.1267054450311439,0.6099971474068941)

DECREASE LEARNING PARAMETERS
RE-COMPUTE r MATRIX

start to calculate r
r[0][1]= 0.8245584438546885      r[0][2]= 0.4976068323370306
r[0][3]= 0.266546651219106       r[0][4]= 0.16085631742867326
r[0][5]= 0.13263543478318265     r[0][6]= 0.16085631742867332
r[0][7]= 0.266546651219106       r[0][8]= 0.49760683233703046
r[0][9]= 0.8245584438546883

r[1][2]= 0.8245584438546885      r[1][3]= 0.49760683233703046
r[1][4]= 0.266546651219106       r[1][5]= 0.16085631742867332
r[1][6]= 0.13263543478318265     r[1][7]= 0.16085631742867332
r[1][8]= 0.266546651219106       r[1][9]= 0.49760683233703046
. . .
. .
.
r[7][8]= 0.8245584438546885      r[7][9]= 0.4976068323370306

r[8][9]= 0.8245584438546885

r has been calculated

PLOT RESULT EVERY 10 SESSIONS
theta = 0.4975  phi = 0.4975
```

```
**************************************************************************

CHOSE A RANDOM PATTERN **2**
Choose a city
city is: 1
x1: 0.5060048209318149 x1: 0.5060048209318149

SEARCH FOR MINIMAL
d=0.002041143125096919     d=0.3559415785649712
d=0.21627388540422732      d=0.17410666553750662
d=0.2219718943372627       d=0.11960289022297033
d=0.3105806655907678       d=0.1505112314384999
d=0.12812049265264963      d=0.2485711999171257
mindist= 0.002041143125096919 for : 0

UPDATE WEIGHTS
gn[0].weight=(0.4985765048525768,0.9121230860807812)
gn[1].weight=(0.5747138695410613,0.588480427328807)
gn[2].weight=(0.605829407118573,0.598192261684896)
gn[3].weight=(0.48849688681484327,0.5720703638013079)
gn[4].weight=(0.439994705996446,0.5051962974197807)
gn[5].weight=(0.3576565658623543,0.6466399906499821)
gn[6].weight=(0.2452705932955198,0.49212567081894987)
gn[7].weight=(0.6139325493000105,0.6148411050297651)
gn[8].weight=(0.7357574721266006,0.7930365559711824)
gn[9].weight=(0.2823008103471305,0.7427349034406745)
DECREASE LEARNING PARAMETERS
RE-COMPUTE r MATRIX

start to calculate r
r[0][1]= 0.8229573574808847      r[0][2]= 0.49411986269887564
r[0][3]= 0.26301929356848097     r[0][4]= 0.1579219822799042
r[0][5]= 0.1299630572252131      r[0][6]= 0.15792198227990428
r[0][7]= 0.26301929356848097     r[0][8]= 0.4941198626988755
r[0][9]= 0.8229573574808846

r[1][2]= 0.8229573574808847      r[1][3]= 0.4941198626988755
r[1][4]= 0.26301929356848097     r[1][5]= 0.15792198227990428
r[1][6]= 0.1299630572252131      r[1][7]= 0.15792198227990428
r[1][8]= 0.26301929356848097     r[1][9]= 0.4941198626988755
. . .
. .
.
r[7][8]= 0.8229573574808847      r[7][9]= 0.49411986269887564

r[8][9]= 0.8229573574808847

r has been calculated

PLOT RESULT EVERY 10 SESSIONS
theta = 0.4950125  phi = 0.4950125
**************************************************************************

CHOSE A RANDOM PATTERN **3**
Choose a city
city is: 4
x1: 0.9267948696624156 x1: 0.9267948696624156

SEARCH FOR MINIMAL
d=0.18457022954084956      d=0.20748914159023796
d=0.1810275796005904       d=0.28538794338253504
```

```
d=0.3755790328619888       d=0.3772113959967891
d=0.6129831093772357       d=0.16686870746435284
d=0.043628132544565136     d=0.433532264241038
mindist= 0.043628132544565136 for : 8

UPDATE WEIGHTS
gn[0].weight=(0.6033167935569346,0.9036526564603641)
gn[1].weight=(0.6205540545780078,0.6261092042745063)
gn[2].weight=(0.6309203542460625,0.6200261163964225)
gn[3].weight=(0.5166940590307908,0.5917192040605392)
gn[4].weight=(0.4780495081280856,0.53429995868293)
gn[5].weight=(0.43175714257287345,0.6766965181888922)
gn[6].weight=(0.4119683702519019,0.5863847225675615)
gn[7].weight=(0.7413845801726533,0.721838594987233)
gn[8].weight=(0.8303233718742983,0.8348434176878624)
gn[9].weight=(0.5448510485795168,0.7976317622815374)

DECREASE LEARNING PARAMETERS
RE-COMPUTE r MATRIX

start to calculate r
r[0][1]= 0.8213432953887355      r[0][2]= 0.4906225649010004
r[0][3]= 0.25950378120296136     r[0][4]= 0.15501241861972714
r[0][5]= 0.1273184107353049      r[0][6]= 0.1550124186197272
r[0][7]= 0.25950378120296136     r[0][8]= 0.49062256490100026
r[0][9]= 0.8213432953887354

r[1][2]= 0.8213432953887355      r[1][3]= 0.49062256490100026
r[1][4]= 0.25950378120296136     r[1][5]= 0.1550124186197272
r[1][6]= 0.1273184107353049      r[1][7]= 0.1550124186197272
r[1][8]= 0.25950378120296136     r[1][9]= 0.49062256490100026
. . .
. .
.
r[7][8]= 0.8213432953887355      r[7][9]= 0.4906225649010004

r[8][9]= 0.8213432953887355

r has been calculated

PLOT RESULT EVERY 10 SESSIONS
theta = 0.4925374375  phi = 0.4925374375
**************************************************************************

CHOSE A RANDOM PATTERN **4**
Choose a city
city is: 1
x1: 0.5268865633262991 x1: 0.5268865633262991

SEARCH FOR MINIMAL
d=0.006450650476734647     d=0.10011222190144231
d=0.10587555234080241      d=0.113412066799906
d=0.157646313709098        d=0.0723700263357316
d=0.1301341369923042       d=0.08864892877513175
d=0.1008140132670473       d=0.017405276914727118
mindist= 0.006450650476734647 for : 0

UPDATE WEIGHTS
gn[0].weight=(0.5656720438116023,0.9158081609391427)
gn[1].weight=(0.5826615901693872,0.7483711189885367)
gn[2].weight=(0.6057805906577616,0.6945282291695211)
```

```
gn[3].weight=(0.5179968173045028,0.6347434797854701)
gn[4].weight=(0.48177818894000146,0.5643840768784543)
gn[5].weight=(0.43772262138790485,0.6924763491878293)
gn[6].weight=(0.4207423075771037,0.6124921992784269)
gn[7].weight=(0.713968445919511,0.7482316181577934)
gn[8].weight=(0.7569978728855262,0.8574349337077508)
gn[9].weight=(0.5375836540013117,0.850505543100838)

DECREASE LEARNING PARAMETERS
RE-COMPUTE r MATRIX

start to calculate r
r[0][1]= 0.8197161844435044      r[0][2]= 0.4871151581182307
r[0][3]= 0.25600054695552876     r[0][4]= 0.15212795510833357
r[0][5]= 0.12470174690859595     r[0][6]= 0.15212795510833366
r[0][7]= 0.25600054695552876     r[0][8]= 0.48711515811823053
r[0][9]= 0.8197161844435041

r[1][2]= 0.8197161844435042      r[1][3]= 0.48711515811823053
r[1][4]= 0.25600054695552876     r[1][5]= 0.15212795510833366
r[1][6]= 0.12470174690859595     r[1][7]= 0.15212795510833366
r[1][8]= 0.25600054695552876     r[1][9]= 0.48711515811823053
. . .
. .
.
r[7][8]= 0.8197161844435044 r[7][9]= 0.4871151581182307

r[8][9]= 0.8197161844435042

r has been calculated

PLOT RESULT EVERY 10 SESSIONS
theta = 0.49007475031250003  phi = 0.49007475031250003
**************************************************************************

CHOSE A RANDOM PATTERN **5**
Choose a city
city is: 2
x1: 0.5484783498455853 x1: 0.5484783498455853

SEARCH FOR MINIMAL
d=0.3351095832828906       d=0.17024905460194065
d=0.13098341058858964      d=0.08947490273945796
d=0.05607194359482073      d=0.13850442193062537
d=0.09211492678611974      d=0.19635282821407635
d=0.3141486549125909       d=0.26362483411888876
mindist= 0.05607194359482073 for : 4

UPDATE WEIGHTS
gn[0].weight=(0.5643901839547163,0.872668856965693)
gn[1].weight=(0.5783729812071614,0.6967829563693977)
gn[2].weight=(0.5921012370220408,0.6092203414738286)
gn[3].weight=(0.5302419257509535,0.5152044384759531)
gn[4].weight=(0.514466253641609,0.4530356509179219)
gn[5].weight=(0.4822156567758751,0.5497447323146047)
gn[6].weight=(0.45123581829242365,0.5467681855786775)
gn[7].weight=(0.6932061570768098,0.696660957229755)
gn[8].weight=(0.7414518938486641,0.8186475913809781)
gn[9].weight=(0.5382494634820308,0.8191344079220187)
```

```
DECREASE LEARNING PARAMETERS
RE-COMPUTE r MATRIX

start to calculate r
r[0][1]= 0.8180759517601792      r[0][2]= 0.48359786612413325
r[0][3]= 0.2525100235849852      r[0][4]= 0.14926891362337855
r[0][5]= 0.12211330858065345     r[0][6]= 0.1492689136233786
r[0][7]= 0.2525100235849852      r[0][8]= 0.4835978661241331
r[0][9]= 0.8180759517601791

r[1][2]= 0.8180759517601792      r[1][3]= 0.4835978661241331
r[1][4]= 0.2525100235849852      r[1][5]= 0.1492689136233786
r[1][6]= 0.12211330858065345     r[1][7]= 0.1492689136233786
r[1][8]= 0.2525100235849852      r[1][9]= 0.4835978661241331
. . .
. .
.
r[7][8]= 0.8180759517601792      r[7][9]= 0.48359786612413325

r[8][9]= 0.8180759517601792

r has been calculated

PLOT RESULT EVERY 10 SESSIONS
theta = 0.48762437656093754  phi = 0.48762437656093754
**************************************************************************

CHOSE A RANDOM PATTERN **6**
Choose a city
city is: 4
x1: 0.956488373967409 x1: 0.956488373967409

SEARCH FOR MINIMAL
d=0.15392441915086374      d=0.17885477808732678
d=0.20950260613572008      d=0.3193329690240743
d=0.383025682201672        d=0.3381451260589217
d=0.37050254432500085      d=0.10524728892684411
d=0.0508056960430553       d=0.17942324812739363
mindist= 0.0508056960430553 for : 8

UPDATE WEIGHTS
gn[0].weight=(0.6568524688711637,0.8758626240317017)
gn[1].weight=(0.6249303457109066,0.720107416338714)
gn[2].weight=(0.6186239422053635,0.6293818096564134)
gn[3].weight=(0.5556229520766821,0.5372962677399312)
gn[4].weight=(0.5466397888552468,0.4845653593092271)
gn[5].weight=(0.5406128767548797,0.5911740151088491)
gn[6].weight=(0.5703814990109994,0.6268139287562624)
gn[7].weight=(0.7982330603588509,0.7722756534609798)
gn[8].weight=(0.8463089234044255,0.8515938567341382)
gn[9].weight=(0.7050907265157078,0.845892757483164)

DECREASE LEARNING PARAMETERS
RE-COMPUTE r MATRIX

start to calculate r
r[0][1]= 0.8164225247209735      r[0][2]= 0.4800709173028292
r[0][3]= 0.24903264359641283     r[0][4]= 0.14643560905002936
r[0][5]= 0.11955332964967841     r[0][6]= 0.14643560905002942
r[0][7]= 0.24903264359641283     r[0][8]= 0.480070917302829
r[0][9]= 0.8164225247209734
```

```
r[1][2]= 0.8164225247209735      r[1][3]= 0.480070917302829
r[1][4]= 0.24903264359641283     r[1][5]= 0.14643560905002942
r[1][6]= 0.11955332964967841     r[1][7]= 0.14643560905002942
r[1][8]= 0.24903264359641283     r[1][9]= 0.480070917302829
. . .
. .
.
r[7][8]= 0.8164225247209735      r[7][9]= 0.4800709173028292

r[8][9]= 0.8164225247209735

r has been calculated

PLOT RESULT EVERY 10 SESSIONS
theta = 0.48518625467813287  phi = 0.48518625467813287
********************************************************************************

CHOSE A RANDOM PATTERN **7**
Choose a city
city is: 3
x1: 0.105425467517264 x1: 0.105425467517264

SEARCH FOR MINIMAL
d=0.8145443040555453       d=0.5820515925378159
d=0.4823899487640566       d=0.3439840526191289
d=0.2991130396443041       d=0.37410324445937243
d=0.43280447995750926      d=0.8531648698867412
d=1.0252910460322389       d=0.8281438233695522
mindist= 0.2991130396443041 for : 4

UPDATE WEIGHTS
gn[0].weight=(0.6176743829131026,0.8251003124751507)
gn[1].weight=(0.5621600177678192,0.6525990251915168)
gn[2].weight=(0.49908779801722775,0.5203751401088468)
gn[3].weight=(0.37729208093612004,0.38839301156577305)
gn[4].weight=(0.33256866477491676,0.3277646296278124)
gn[5].weight=(0.3682277514254966,0.4209288679803399)
gn[6].weight=(0.46208216851209805,0.5184053798124737)
gn[7].weight=(0.7145230479421485,0.698463919477497)
gn[8].weight=(0.793670232066524,0.8025558057758997)
gn[9].weight=(0.6703067640309255,0.806187683978999)

DECREASE LEARNING PARAMETERS
RE-COMPUTE r MATRIX

start to calculate r
r[0][1]= 0.8147558309931716      r[0][2]= 0.4765345446590645
r[0][3]= 0.24556883905865032     r[0][4]= 0.14362834907314673
r[0][5]= 0.117022034903269       r[0][6]= 0.14362834907314678
r[0][7]= 0.24556883905865032     r[0][8]= 0.4765345446590643
r[0][9]= 0.8147558309931714

r[1][2]= 0.8147558309931715      r[1][3]= 0.4765345446590643
r[1][4]= 0.24556883905865032     r[1][5]= 0.14362834907314678
r[1][6]= 0.117022034903269       r[1][7]= 0.14362834907314678
r[1][8]= 0.24556883905865032     r[1][9]= 0.4765345446590643
. . .
. .
.
r[7][8]= 0.8147558309931716      r[7][9]= 0.4765345446590645
```

```
r[8][9]= 0.8147558309931715

r has been calculated

PLOT RESULT EVERY 10 SESSIONS
theta = 0.4827603234047422  phi = 0.4827603234047422
*************************************************************************

CHOSE A RANDOM PATTERN **8**
Choose a city
city is: 1
x1: 0.5069434582278964 x1: 0.5069434582278964

SEARCH FOR MINIMAL
d=0.02342046847866977      d=0.08040965689090006
d=0.16845864920629494      d=0.3109465894745709
d=0.39398236815307264      d=0.2791464803551792
d=0.1720299824694898       d=0.09704011627668449
d=0.09864268692592952      d=0.04220012661792649
mindist= 0.02342046847866977 for : 0

UPDATE WEIGHTS
gn[0].weight=(0.5642178859011663,0.8760975619977632)
gn[1].weight=(0.5404415896510331,0.7619995714446662)
gn[2].weight=(0.5008950081313589,0.6147797176725901)
gn[3].weight=(0.3926623673841027,0.45268838706272646)
gn[4].weight=(0.3446594761042022,0.36957357590377005)
gn[5].weight=(0.37606430044335026,0.4497297679576866)
gn[6].weight=(0.4651927636801342,0.5469956647411551)
gn[7].weight=(0.6899143023868851,0.726000112208009)
gn[8].weight=(0.7277081726371157,0.8320441650549989)
gn[9].weight=(0.6060507827896754,0.8551769283831487)

DECREASE LEARNING PARAMETERS
RE-COMPUTE r MATRIX

start to calculate r
r[0][1]= 0.813075798547325        r[0][2]= 0.47298898582648613
r[0][3]= 0.24211904141886398      r[0][4]= 0.1408474339718326
r[0][5]= 0.11451963984998949      r[0][6]= 0.1408474339718327
r[0][7]= 0.24211904141886398      r[0][8]= 0.47298898582648596
r[0][9]= 0.8130757985473248

r[1][2]= 0.8130757985473249       r[1][3]= 0.47298898582648596
r[1][4]= 0.24211904141886398      r[1][5]= 0.1408474339718327
r[1][6]= 0.11451963984998949      r[1][7]= 0.1408474339718327
r[1][8]= 0.24211904141886398      r[1][9]= 0.47298898582648596
. . .
. .
.
r[7][8]= 0.813075798547325        r[7][9]= 0.47298898582648613

r[8][9]= 0.8130757985473249

r has been calculated

PLOT RESULT EVERY 10 SESSIONS
theta = 0.4803465217877185  phi = 0.4803465217877185
*************************************************************************
```

```
CHOSE A RANDOM PATTERN **9**
Choose a city
city is: 3
x1: 0.1103708083980233 x1: 0.1103708083980233

SEARCH FOR MINIMAL
d=0.7674114775562291       d=0.5884287516175949
d=0.3906251962046208       d=0.1858863161251531
d=0.1138261297282585       d=0.17487126724628074
d=0.3024559907733976       d=0.6949014169931726
d=0.8784629616103027       d=0.7762194080011386
mindist= 0.1138261297282585 for : 4

UPDATE WEIGHTS
gn[0].weight=(0.533512600905586,0.8254039702065195)
gn[1].weight=(0.49042391077396275,0.6881261733668388)
gn[2].weight=(0.4121684511459838,0.5039133015126358)
gn[3].weight=(0.28241110348867715,0.3254132926207284)
gn[4].weight=(0.23211972947726062,0.2529621489719359)
gn[5].weight=(0.2722955465634972,0.32361018626600735)
gn[6].weight=(0.3845777071573948,0.4515296914711098)
gn[7].weight=(0.6225127916560806,0.6563134886573914)
gn[8].weight=(0.685941858283686,0.7843310311651098)
gn[9].weight=(0.5787838682173359,0.8151100212062181)

DECREASE LEARNING PARAMETERS
RE-COMPUTE r MATRIX

start to calculate r
r[0][1]= 0.8113823556758024      r[0][2]= 0.4694344830740688
r[0][3]= 0.2386836813142946      r[0][4]= 0.1380931564165799
r[0][5]= 0.1120463505559917      r[0][6]= 0.13809315641657996
r[0][7]= 0.2386836813142946      r[0][8]= 0.46943448307406865
r[0][9]= 0.8113823556758022

r[1][2]= 0.8113823556758023      r[1][3]= 0.46943448307406865
r[1][4]= 0.2386836813142946      r[1][5]= 0.13809315641657996
r[1][6]= 0.1120463505559917      r[1][7]= 0.13809315641657996
r[1][8]= 0.2386836813142946      r[1][9]= 0.46943448307406865
. . .
. .
.
r[7][8]= 0.8113823556758024      r[7][9]= 0.4694344830740688

r[8][9]= 0.8113823556758023

r has been calculated

PLOT RESULT EVERY 10 SESSIONS
theta = 0.47794478917877986  phi = 0.47794478917877986
********************************************************************************

CHOSE A RANDOM PATTERN  **10**
Choose a city
city is: 1
x1: 0.49074730241401243 x1: 0.49074730241401243

SEARCH FOR MINIMAL
d=0.018250963898698506     d=0.07045133774358844
d=0.20835021119126776      d=0.4379630682393049
d=0.5577153208761358       d=0.4445487215440154
```

```
d=0.2632990691469462       d=0.10571328409131867
d=0.06673688282112882      d=0.02691680812041092
mindist= 0.018250963898698506 for : 0

UPDATE WEIGHTS
gn[0].weight=(0.5130731493338633,0.8866519807573673)
gn[1].weight=(0.49054932074835866,0.7910575046856297)
gn[2].weight=(0.42979869806947507,0.6047960865402315)
gn[3].weight=(0.30617760158571755,0.3970699406464274)
gn[4].weight=(0.2491893832276282,0.299201758065739)
gn[5].weight=(0.2839940683070301,0.35734484768709535)
gn[6].weight=(0.39158499647797446,0.48466366314955805)
gn[7].weight=(0.6074812980151242,0.6902218292838846)
gn[8].weight=(0.6421472728122676,0.8222982385822012)
gn[9].weight=(0.5446436428806434,0.8687975282140769)

*************************************************************************

CHOSE A RANDOM PATTERN **20**
Choose a city
city is: 0
x1: 0.2793823241792803 x1: 0.2793823241792803

SEARCH FOR MINIMAL
d=0.22901548161834837      d=0.12655778404792695
d=0.02886067179954723      d=0.007283166947219182
d=0.020117464942129338     d=0.0383407321388283
d=0.0982292608034643       d=0.2850190518319296
d=0.3768692661463984       d=0.319757550151054
mindist= 0.007283166947219182 for : 3

UPDATE WEIGHTS
gn[0].weight=(0.5844385555123249,0.8432316579967859)
gn[1].weight=(0.4681721380615517,0.7490270921701532)
gn[2].weight=(0.364513034488589,0.6020676484051285)
gn[3].weight=(0.3240074825027743,0.5214159191064518)
gn[4].weight=(0.3481157779708628,0.4755319578084637)
gn[5].weight=(0.4236044189836067,0.47204300876400435)
gn[6].weight=(0.5600327476598468,0.57928858323036)
gn[7].weight=(0.7380854244598255,0.7498483530858489)
gn[8].weight=(0.7891786863212588,0.8298058932879652)
gn[9].weight=(0.7115340292215435,0.8528535673910456)

*************************************************************************

CHOSE A RANDOM PATTERN **30**
Choose a city
city is: 2
x1: 0.5182300467816038 x1: 0.5182300467816038

SEARCH FOR MINIMAL
d=0.23212499213268697      d=0.10025724672534032
d=0.05702797174893441      d=0.09031919532958459
d=0.06378241102920212      d=0.014144400504353637
d=0.03901888996810459      d=0.22030125804016598
d=0.35207994767145856      d=0.33465951949056205
mindist= 0.014144400504353637 for : 5

UPDATE WEIGHTS
gn[0].weight=(0.6473047126037905,0.7690616215298217)
gn[1].weight=(0.46389490794714694,0.6193645397681738)
```

```
gn[2].weight=(0.3077078858907935,0.3866303381563009)
gn[3].weight=(0.2742556262096751,0.27021242960215996)
gn[4].weight=(0.35527801691255595,0.27903693167036003)
gn[5].weight=(0.45194604030850655,0.33260777393600643)
gn[6].weight=(0.556337520454758,0.44552980908066214)
gn[7].weight=(0.7239296294004658,0.6497553617607661)
gn[8].weight=(0.8203022111728927,0.7779998377837338)
gn[9].weight=(0.7914232036821931,0.8043315076260503)

**************************************************************************

CHOSE A RANDOM PATTERN **40**
Choose a city
city is: 0
x1: 0.2456528662123411 x1: 0.2456528662123411

SEARCH FOR MINIMAL
d=0.20440766790488946      d=0.13569575394227668
d=0.03689897520849689      d=0.03706786186303328
d=0.0807767274206259       d=0.09789138963723254
d=0.09969406876742162      d=0.15063316130981674
d=0.2410272897871192       d=0.23149342767320064
mindist= 0.03689897520849689 for : 2

UPDATE WEIGHTS
gn[0].weight=(0.4821741288070994,0.8288963805663417)
gn[1].weight=(0.40825341131302095,0.7202210379901915)
gn[2].weight=(0.34560579180513123,0.5778138351058761)
gn[3].weight=(0.36544590603092353,0.46749041432113714)
gn[4].weight=(0.45285664419945165,0.3996058876536426)
gn[5].weight=(0.5015241534597266,0.3794852918201004)
gn[6].weight=(0.531730573325497,0.41406867333263314)
gn[7].weight=(0.6210720871828591,0.5825304345390006)
gn[8].weight=(0.6562238737423404,0.7677598443269807)
gn[9].weight=(0.5660133226434424,0.8445604319904584)

**************************************************************************

CHOSE A RANDOM PATTERN **50**
Choose a city
city is: 2
x1: 0.5261836743595895 x1: 0.5261836743595895

SEARCH FOR MINIMAL
d=0.27452900055493523      d=0.1615233147349127
d=0.09435307638377727      d=0.05635256633194649
d=0.01591184854968386      d=0.0037225895287891246
d=0.014891562693110415     d=0.1240253843022256
d=0.2701879293321428       d=0.3123394253296888
mindist= 0.0037225895287891246 for : 5

UPDATE WEIGHTS
gn[0].weight=(0.495538787185762,0.8502698390548032)
gn[1].weight=(0.40474119987251345,0.7095530373826118)
gn[2].weight=(0.3338566369593139,0.5560995754999327)
gn[3].weight=(0.35049067603749834,0.4451868785984179)
gn[4].weight=(0.4451065429124503 3,0.3741728604798691)
gn[5].weight=(0.5016583926571465,0.36292835967475884)
gn[6].weight=(0.5339071391249507,0.42178072689809454)
gn[7].weight=(0.6085996409197676,0.6314259990598003)
gn[8].weight=(0.6072224272606006,0.824218942577302)
```

```
gn[9].weight=(0.5481807171219614,0.8820893323435139)

**************************************************************************

CHOSE A RANDOM PATTERN **60**
Choose a city
city is: 4
x1: 0.9143157417607362 x1: 0.9143157417607362

SEARCH FOR MINIMAL
d=0.1778819826244351       d=0.3967495926603701
d=0.7338181657622327       d=0.8502339957401707
d=0.6744799526481995       d=0.4751527974611488
d=0.32688769298852144      d=0.12616420289978447
d=0.0767284280662772       d=0.1165055758925937
mindist= 0.0767284280662772 for : 8

UPDATE WEIGHTS
gn[0].weight=(0.539035386810225,0.8247592625425594)
gn[1].weight=(0.3663883586373343,0.5946548319044619)
gn[2].weight=(0.23327041155381725,0.35850308232448025)
gn[3].weight=(0.21717858144720442,0.26856880939359384)
gn[4].weight=(0.3304571151320176,0.29695020645546977)
gn[5].weight=(0.4678636990973236,0.36470635949029123)
gn[6].weight=(0.5694464316742557,0.48118119489321287)
gn[7].weight=(0.6991656456258516,0.7090075575180809)
gn[8].weight=(0.7406446747896496,0.8478727086609124)
gn[9].weight=(0.6638293938711257,0.8793825659409485)

**************************************************************************

CHOSE A RANDOM PATTERN **70**
Choose a city
city is: 4
x1: 0.960590987592621 x1: 0.960590987592621

SEARCH FOR MINIMAL
d=0.14743597907913622      d=0.34992117419569635
d=0.5556098108648643       d=0.6514997918544045
d=0.5596068965213461       d=0.382491149214525
d=0.16821226476694662      d=0.020247494178182567
d=0.009721864661705848     d=0.03730120399334878
mindist= 0.009721864661705848 for : 8

UPDATE WEIGHTS
gn[0].weight=(0.611475358765767,0.8438399982960929)
gn[1].weight=(0.414632145265186,0.6813904650132961)
gn[2].weight=(0.30287424886547126,0.5303044690175439)
gn[3].weight=(0.28592611457631845,0.4322298535300306)
gn[4].weight=(0.392032447067018,0.390563695864995)
gn[5].weight=(0.5350793608552145,0.4391981075033579)
gn[6].weight=(0.6879759112831304,0.6101597260762339)
gn[7].weight=(0.859491608867339,0.8279201661596961)
gn[8].weight=(0.8979564839086792,0.8773438926551426)
gn[9].weight=(0.8151024520984551,0.8826080557155617)

**************************************************************************

CHOSE A RANDOM PATTERN **80**
Choose a city
city is: 1
```

```
x1: 0.5201080007685498 x1: 0.5201080007685498

SEARCH FOR MINIMAL
d=0.054194904146122685     d=0.1899348370280533
d=0.2616720624804285       d=0.3389973108602465
d=0.3672417246569066       d=0.2652851750860546
d=0.12965528195868797      d=0.10920752749557869
d=0.08132167211553365      d=0.028538242091125307
mindist= 0.028538242091125307 for : 9

UPDATE WEIGHTS
gn[0].weight=(0.46868963275081854,0.7753733666288692)
gn[1].weight=(0.3189274278667076,0.5990945521520827)
gn[2].weight=(0.26450621812329683,0.5173217752768968)
gn[3].weight=(0.23917855546161013,0.44349436832405353)
gn[4].weight=(0.3006349515933041,0.3871244741706253)
gn[5].weight=(0.4820853302649967,0.438894495295889)
gn[6].weight=(0.672369674235114,0.6304914405199687)
gn[7].weight=(0.8080878411595711,0.8450667607689449)
gn[8].weight=(0.7351479403866569,0.8933046960761358)
gn[9].weight=(0.6130874882964014,0.8867506781994682)

**************************************************************************

CHOSE A RANDOM PATTERN **90**
Choose a city
city is: 4
x1: 0.9578265060038073 x1: 0.9578265060038073

SEARCH FOR MINIMAL
d=0.24523529561591997      d=0.49048682196800286
d=0.6530343729983035       d=0.8373580587279703
d=0.897431656126658        d=0.6297190089229032
d=0.39188820405439717      d=0.1548493272259289
d=0.1081234554254508       d=0.15949942037483744
mindist= 0.1081234554254508 for : 8

UPDATE WEIGHTS
gn[0].weight=(0.501355272803477,0.8116780194568474)
gn[1].weight=(0.334846459990919,0.6054056428543342)
gn[2].weight=(0.26722665710066507,0.491146578728831)
gn[3].weight=(0.22242853257673806,0.36392193362626524)
gn[4].weight=(0.25278389182205396,0.27751515819950595)
gn[5].weight=(0.4224784716839787,0.33315347306762244)
gn[6].weight=(0.5740782530184466,0.458033279463758)
gn[7].weight=(0.7205498512995231,0.6983397924718103)
gn[8].weight=(0.7383176170585405,0.8622665837416632)
gn[9].weight=(0.6387942262571594,0.895246368121542)

**************************************************************************

CHOSE A RANDOM PATTERN **100**
Choose a city
city is: 2
x1: 0.502069121734622 x1: 0.502069121734622

SEARCH FOR MINIMAL
d=0.28481793081635937      d=0.145436369112526
d=0.08231896478176573      d=0.08596425445483571
d=0.09263002908932283      d=0.012298286754975367
d=0.014452874793534659     d=0.17796658635385357
```

```
d=0.37138626989115137      d=0.3539723086435588
mindist= 0.012298286754975367 for : 5

UPDATE WEIGHTS
gn[0].weight=(0.5359558880585267,0.8681431137655385)
gn[1].weight=(0.3871771715560967,0.6989934772546732)
gn[2].weight=(0.28558128524255855,0.5206447872036635)
gn[3].weight=(0.22285479297222002,0.34533157624935934)
gn[4].weight=(0.26557629366028307,0.2583206045163704)
gn[5].weight=(0.42717766425366793,0.31778749586565497)
gn[6].weight=(0.5562060067966648,0.4183957761910097)
gn[7].weight=(0.7362029076854513,0.6629986542830864)
gn[8].weight=(0.8256485755558136,0.8462718931988288)
gn[9].weight=(0.7087486162978069,0.8927114649217234)

*************************************************************************

CHOSE A RANDOM PATTERN **110**
Choose a city
city is: 4
x1: 0.9318778391842508 x1: 0.9318778391842508

SEARCH FOR MINIMAL
d=0.1525680164455968       d=0.24823203948240075
d=0.5399804577938707       d=0.9667815803744664
d=1.1106069449230278       d=0.8191702153845899
d=0.34788610259635655      d=0.03441933734093026
d=0.005236969269782932     d=0.05118878833958761
mindist= 0.005236969269782932 for : 8

UPDATE WEIGHTS
gn[0].weight=(0.5566953496343806,0.9129222367898666)
gn[1].weight=(0.44442802105097295,0.797354735119638)
gn[2].weight=(0.2918760300163151,0.526195353638773)
gn[3].weight=(0.17088242927464292,0.2638308342959938)
gn[4].weight=(0.15207297785817322,0.17848532798439165)
gn[5].weight=(0.293783329765983,0.2508997964850531)
gn[6].weight=(0.5595601763622239,0.45625496269894594)
gn[7].weight=(0.8246473897260594,0.773195569223023)
gn[8].weight=(0.8810198886682414,0.877758348292209)
gn[9].weight=(0.7435240695653768,0.9030695996612411)

*************************************************************************

CHOSE A RANDOM PATTERN **120**
Choose a city
city is: 3
x1: 0.10790785388530233 x1: 0.10790785388530233

SEARCH FOR MINIMAL
d=0.7505170658919094       d=0.38498110127087526
d=0.18450010877679035      d=0.060622596727825134
d=0.010786882153678942     d=0.07614629464796573
d=0.283014506646947        d=0.7433339280824898
d=1.1245294858573698       d=1.0576610760143677
mindist= 0.010786882153678942 for : 4

UPDATE WEIGHTS
gn[0].weight=(0.5663746986919692,0.8753506217231827)
gn[1].weight=(0.380284012639606,0.6959512986957589)
gn[2].weight=(0.2728380936351286,0.5244525422659765)
```

```
gn[3].weight=(0.19328784957935316,0.3329677415060069)
gn[4].weight=(0.1578584382220325,0.1972988447069952)
gn[5].weight=(0.3124145219620404,0.2577068529693065)
gn[6].weight=(0.547506398037303,0.413071103995513)
gn[7].weight=(0.775704318532568,0.6813045294081232)
gn[8].weight=(0.8991521352889924,0.8458959333329339)
gn[9].weight=(0.8174153976763838,0.8849558007254825)

**************************************************************************

CHOSE A RANDOM PATTERN **130**
Choose a city
city is: 2
x1: 0.5325908675200279 x1: 0.5325908675200279

SEARCH FOR MINIMAL
d=0.3301419293539476       d=0.17758715246580675
d=0.1152956326208877       d=0.11685309741199162
d=0.17323149948537234      d=0.08034973964245452
d=0.01338389923225563      d=0.2772322870370224
d=0.45105982208446443      d=0.3999130751200193
mindist= 0.01338389923225563 for : 6

UPDATE WEIGHTS
gn[0].weight=(0.542002856292308,0.8952113456069847)
gn[1].weight=(0.3776389566696901,0.7127228552037669)
gn[2].weight=(0.2703740117972858,0.5364495965507833)
gn[3].weight=(0.19255758499921383,0.3476051207457218)
gn[4].weight=(0.14577648874244734,0.1929690688469749)
gn[5].weight=(0.29758271927559,0.24670129803038693)
gn[6].weight=(0.5481086494133032,0.40488203132338746)
gn[7].weight=(0.7909280176215275,0.6988330358962651)
gn[8].weight=(0.9025200291910157,0.8643953657407832)
gn[9].weight=(0.7892170365328429,0.897354746886862)

**************************************************************************

CHOSE A RANDOM PATTERN **140**
Choose a city
city is: 2
x1: 0.5400296436448678 x1: 0.5400296436448678

SEARCH FOR MINIMAL
d=0.3003297836059415       d=0.16749545716987418
d=0.10432441414496123      d=0.15017885515706877
d=0.19706455508751014      d=0.05184913780515884
d=5.27257151460334E-4      d=0.07982042566968459
d=0.35288360089086246      d=0.347861000488067
mindist= 5.27257151460334E-4 for : 6

UPDATE WEIGHTS
gn[0].weight=(0.5411362120876079,0.9013016106392783)
gn[1].weight=(0.39333425076360584,0.7354058811659229)
gn[2].weight=(0.26442555331065093,0.5216914868735107)
gn[3].weight=(0.1590107428581135,0.2853994522826809)
gn[4].weight=(0.1371016487648979,0.18398232076687693)
gn[5].weight=(0.35465517973022753,0.2745753147073836)
gn[6].weight=(0.5259551553938874,0.3633324845922024)
gn[7].weight=(0.6831029852833299,0.5582886879932762)
gn[8].weight=(0.8782477990747687,0.830563328653131)
gn[9].weight=(0.7752454955090077,0.8934161072534481)
```

```
**************************************************************************

CHOSE A RANDOM PATTERN **150**
Choose a city
city is: 1
x1: 0.519446151046657 x1: 0.519446151046657

SEARCH FOR MINIMAL
d=9.267065397201875E-4     d=0.044134415430442074
d=0.23969639020613442      d=0.6043179064396976
d=0.7447058712045462       d=0.46091077974389644
d=0.33505671695434136      d=0.1784810204190354
d=0.14557117858485652      d=0.0694447965060924
mindist= 9.267065397201875E-4 for : 0

UPDATE WEIGHTS
gn[0].weight=(0.5352498850776714,0.9173081114268357)
gn[1].weight=(0.4169996887348031,0.7757207499286345)
gn[2].weight=(0.26627694789594253,0.521541498889729)
gn[3].weight=(0.14781146524344302,0.25215676704118156)
gn[4].weight=(0.13460261422034114,0.16200575158758077)
gn[5].weight=(0.36675891818570777,0.27284397850106196)
gn[6].weight=(0.5191409551328119,0.35553577800009295)
gn[7].weight=(0.6698039232484825,0.5398395224677507)
gn[8].weight=(0.8830023807213441,0.833154731020118)
gn[9].weight=(0.7534955336327771,0.8979938075707715)

**************************************************************************

CHOSE A RANDOM PATTERN **160**
Choose a city
city is: 2
x1: 0.5199186446096159 x1: 0.5199186446096159

SEARCH FOR MINIMAL
d=0.3645437774504208       d=0.22358592765909935
d=0.10658414684879487      d=0.1455002959278072
d=0.1894651251029848       d=0.03481090451585031
d=6.330469580644382E-4     d=0.08438286936744166
d=0.40687211698841763      d=0.37948167397491006
mindist= 6.330469580644382E-4 for : 6

UPDATE WEIGHTS
gn[0].weight=(0.5275223373457251,0.9323018594367853)
gn[1].weight=(0.4202412239537591,0.7908110598768664)
gn[2].weight=(0.26304091513767913,0.5300668187314354)
gn[3].weight=(0.145297555910459,0.25749387676090335)
gn[4].weight=(0.12239938481753586,0.15944839293818353)
gn[5].weight=(0.36099831768418084,0.26778172510202597)
gn[6].weight=(0.5219938395702338,0.3479735367712537)
gn[7].weight=(0.6768364867085592,0.5420454671386371)
gn[8].weight=(0.8933367138960409,0.8398139201978585)
gn[9].weight=(0.7386410003433439,0.9042414840980554)

**************************************************************************

CHOSE A RANDOM PATTERN  **170**
Choose a city
city is: 0
x1: 0.2710435773978865 x1: 0.2710435773978865
```

```
SEARCH FOR MINIMAL
d=0.22238924124350257      d=0.0960970340207225
d=1.8181830249720384E-4    d=0.0981144994967035
d=0.1835301315224055       d=0.08884711124765483
d=0.10374747728904404      d=0.16288922155828062
d=0.47107677673976656      d=0.32087784004578546
mindist= 1.8181830249720384E-4 for : 2

UPDATE WEIGHTS
gn[0].weight=(0.5174462427321792,0.9440310841545511)
gn[1].weight=(0.4172231875407716,0.7911033989124232)
gn[2].weight=(0.265583990844675,0.5356246696578683)
gn[3].weight=(0.155057011329455,0.27946469734109614)
gn[4].weight=(0.1140694328141218,0.14839144132775256)
gn[5].weight=(0.3432841575723506,0.2555708544675761)
gn[6].weight=(0.5258765668927012,0.34771307034672333)
gn[7].weight=(0.674554585207982,0.5365317042373807)
gn[8].weight=(0.890257707833017,0.8407440348829158)
gn[9].weight=(0.7008078459060161,0.9135799516014783)

*************************************************************************

CHOSE A RANDOM PATTERN **180**
Choose a city
city is: 0
x1: 0.2855902268274663 x1: 0.2855902268274663

SEARCH FOR MINIMAL
d=0.23573452689246005      d=0.09335169472828546
d=9.44078931294905E-4      d=0.06410746201717389
d=0.15862062040391864      d=0.06998817092867743
d=0.07940257103580109      d=0.14323303327133133
d=0.4694467785320982       d=0.3101074280202205
mindist= 9.44078931294905E-4 for : 2

UPDATE WEIGHTS
gn[0].weight=(0.5080747722758338,0.93652539915041)
gn[1].weight=(0.40535843589060255,0.7662080365005013)
gn[2].weight=(0.2707881639805539,0.5261951557301632)
gn[3].weight=(0.16587052945784925,0.3023699671424753)
gn[4].weight=(0.10335379489711977,0.15401530496409235)
gn[5].weight=(0.30696051039208233,0.2430514852426882)
gn[6].weight=(0.518693108467462,0.34839982874179043)
gn[7].weight=(0.6636101716030076,0.52498408198616)
gn[8].weight=(0.8840436296954423,0.8403307784175341)
gn[9].weight=(0.6608620743391324,0.9180954332606475)

*************************************************************************


CHOSE A RANDOM PATTERN **335**
Choose a city
city is: 3
x1: 0.1028241007283194 x1: 0.1028241007283194

SEARCH FOR MINIMAL
d=0.8007339193881481       d=0.48243201067284247
d=0.17638768641586766      d=0.038838021701468564
d=1.012488859740458E-4     d=0.06389073169104234
d=0.2189194239127525       d=0.48018445970032875
d=1.2337310534990964       d=0.9219376816131303
```

```
mindist= 1.012488859740458E-4 for : 4

UPDATE WEIGHTS
gn[0].weight=(0.5072553444850841,0.9420081366941265)
gn[1].weight=(0.4069553295668272,0.768228013953598)
gn[2].weight=(0.2672692771335046,0.5302317086012399)
gn[3].weight=(0.17763439587797888,0.3260145426254217)
gn[4].weight=(0.09521366446565864,0.13875539724821195)
gn[5].weight=(0.3320414031598314,0.25007097275580475)
gn[6].weight=(0.5274916856499048,0.340189236048232)
gn[7].weight=(0.6752587189789383,0.5342957033686792)
gn[8].weight=(0.9352092841359052,0.8792151899405805)
gn[9].weight=(0.6625731635705433,0.9239194513942505)

**************************************************************************
```