T.C.
YILDIZ TEKNİK ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

ADİ DİFERANSİYEL DENKLEMLERİN FONKSİYONEL BAĞLANTILI YAPAY SİNİR AĞLARIYLA ÇÖZÜMÜ

Sena Nur KURMANÇ

YÜKSEK LİSANS TEZİ
Matematik Anabilim Dalı
Matematik Programı

Danışman
Doç. Dr. Muttalip ÖZAVŞAR

Temmuz, 2022

T.C.
YILDIZ TEKNİK ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

ADİ DİFERANSİYEL DENKLEMLERİN FONKSİYONEL BAĞLANTILI YAPAY SİNİR AĞLARIYLA ÇÖZÜMÜ

Sena Nur KURMANÇ tarafından hazırlanan tez çalışması 18.07.2022 tarihinde aşağıdaki jüri tarafından Yıldız Teknik Üniversitesi Fen Bilimleri Enstitüsü Matematik Anabilim Dalı Matematik Programı **YÜKSEK LİSANS TEZİ** olarak kabul edilmiştir.

Doç. Dr. Muttalip ÖZAVŞAR
Yıldız Teknik Üniversitesi
Danışman

Jüri Üyeleri

Doç. Dr. Muttalip ÖZAVŞAR, Danışman
Yıldız Teknik Üniversitesi _____

Doç. Dr. Erdoğan Mehmet ÖZKAN, Üye
Yıldız Teknik Üniversitesi _____

Dr. Öğr. Üye. Abdullah SÖNMEZ, Üye
İstanbul Sabahattin Zaim Üniversitesi _____

Danışmanım Doç. Dr. Muttalip ÖZAVŞAR sorumluluğunda tarafımca hazırlanan ADİ DİFERANSİYEL DENKLEMLERİN FONKSİYONEL BAĞLANTILI YAPAY SİNİR AĞLARIYLA ÇÖZÜMÜ başlıklı çalışmada veri toplama ve veri kullanımında gerekli yasal izinleri aldığımı, diğer kaynaklardan aldığım bilgileri ana metin ve referanslarda eksiksiz gösterdiğimi, araştırma verilerine ve sonuçlarına ilişkin çarpıtma ve/veya sahtecilik yapmadığımı, çalışmam süresince bilimsel araştırma ve etik ilkelerine uygun davrandığımı beyan ederim. Beyanımın aksinin ispatı halinde her türlü yasal sonucu kabul ederim.

Sena Nur KURMANÇ

İmza

Aileme...

TEŞEKKÜR

Tez çalışmam sırasında gösterdiği yol ve yöntemlerle desteğini esirgemeyen sayın hocam Doç. Dr. Muttalip ÖZAVŞAR'a ve maddi manevi desteklerini esirgemeyen aileme teşekkür ederim.

Sena Nur KURMANÇ

İÇİNDEKİLER

SİMGE LİSTESİ vii

KISALTMA LİSTESİ viii

ŞEKİL LİSTESİ ix

TABLO LİSTESİ x

ÖZET xi

ABSTRACT xii

1 GİRİŞ 1

1.1 Literatür Özeti 1

1.2 Tezin Amacı 2

1.3 Hipotez 2

2 YAPAY SİNİR AĞLARI 3

2.1 Tanım 3

2.2 Tarihsel Gelişim 4

2.3 Yapay Sinir Ağlarının Kullanım Alanları 6

2.4 Yapay Sinir Ağlarının Genel Özellikleri 7

2.4.1 Paralellik Özelliği 7

2.4.2 Doğrusal Olmama 7

2.4.3 Hata Toleransı 7

2.4.4 Öğrenme, Genelleme ve Uyum Sağlama Özellikleri 7

2.5 Biyolojik Sinir Hücresi 8

2.6 Yapay Nöron 9

2.7 Yapay Sinir Ağının Yapısı 11

2.7.1 İleri Beslemeli Yapay Sinir Ağları 12

2.7.2 Geri Beslemeli (Yineleyen) Yapay Sinir Ağları 12

2.8 Yapay Sinir Ağlarında Öğrenme 13

2.8.1 Denetimli Öğrenme: 13

2.8.2 Denetimsiz Öğrenme: 14

2.8.3 Destekli Öğrenme: 14

3 ÇOK KATMANLI ALGILAYICI - ÇKA (MULTILAYER PERCEPTRON - MLP) 15

3.1 Çok Katmanlı Algılayıcının Öğrenme Kuralı 17

3.1.1 Gradyan İnişi 23

3.1.2 Gradyan İnişi Optimizasyon Algoritmaları 25

4 TEK KATMANLI FONKSİYONEL BAĞLANTILI YAPAY SİNİR AĞI (SINGLE-LAYER FUNCTIONAL LINK ARTIFICIAL NEURAL NETWORK) 32

4.1 Tek Katmanlı Fonksiyonel Bağlantılı Yapay Sinir Ağı Modelleri 33

4.1.1 Laguerre Yapay Sinir Ağı (Laguerre Neural Network - LgNN) . . 33

4.1.2 Chebyshev Yapay Sinir Ağı (Chebyshev Neural Network - ChNN) 34

4.1.3 Legendre Yapay Sinir Ağı (Legendre Neural Network - LeNN) . 36

5 İLERİ BESLEMELİ YAPAY SİNİR AĞLARI İLE ADİ DİFERANSİYEL DENKLEMLERİN ÇÖZÜMÜ 38

5.1 İleri Beslemeli Yapay Sinir Ağları ile Adi Diferansiyel Denklemlerin Genel Çözümü 38

5.1.1 Gradyan Hesaplama 39

5.2 Birinci Mertebeden Diferansiyel Denklemlerin Çözümü 41

5.3 İkinci Mertebeden Diferansiyel Denklemlerin Çözümü 43

5.4 Birinci Mertebeden Diferansiyel Denklem Sistemlerinin Çözümü 45

6 DENEYSEL SONUÇLAR 46

7 SONUÇ VE ÖNERİLER 57

KAYNAKÇA 58

TEZDEN ÜRETİLMİŞ YAYINLAR 61

SİMGE LİSTESİ

∇	Gradyan operatörü
Δw	w ağırlığındaki değişim miktarı
E	Maliyet fonksiyonu
w^*	İdeal ağırlık vektörü
η	Öğrenme katsayısı parametresi
$\nabla_w E$	E maliyet fonksiyonunun, w ağırlık vektörüne göre gradyanı
$\phi(.)$	Aktivasyon fonksiyonu
γ	Momentum sabiti
δ_j	j. nöronun hata değeri

KISALTMA LİSTESİ

Adagrad	Adaptive Gradient (Adaptif Gradyan)
Adam	Adaptif Moment Tahmin
ChNN	Chebyshev Yapay Sinir Ağı
ÇKA	Çok Katmanlı Algılayıcı
FLANN	Fonksiyonel Bağlantılı Yapay Sinir Ağı
LeNN	Legendre Yapay Sinir Ağı
LgNN	Laguerre Yapay Sinir Ağı
NAG	Nesterov Hızlandırılmış Gradyan
RBFNN	Radyal Tabanlı Fonksiyon Sinir Ağı
RMSprop	Kare Ortalamalarının Karekökü Yayılımı (Root Mean Square Propagation)
SGD	Stokastik Gradyan İnişi (Stochastic Gradient Descent)
vd.	ve diğerleri
YSA	Yapay Sinir Ağı

ŞEKİL LİSTESİ

Şekil 2.1 Biyolojik Sinir Hücresi 8
Şekil 2.2 Nöronlar arası sinaptik bağlantının gösterimi 9
Şekil 2.3 Yapay Nöron Modeli 9
Şekil 2.4 Tek Gizli Katmana Sahip İleri Beslemeli Yapay Sinir Ağı 12
Şekil 2.5 Çıktı Katmanından Gizli Katmana Bağlantıların Olduğu Çok Katmanlı Algılayıcı Modeli 13
Şekil 3.1 Çok Katmanlı Algılayıcının Genel Görünümü 15
Şekil 3.2 Tek Gizli Katmana Sahip Çok Katmanlı Algılayıcı Modeli 16
Şekil 3.3 Çıktı Katmanındaki j. Nöronun Sinaptik Ağırlığını Ayarlamak için Nöral Bağlantıların Gösterimi 18
Şekil 3.4 Gizli Katmanda Bulunan j. Nöronun Sinaptik Ağırlığını Ayarlamak için Nöral Bağlantıların Gösterimi 20
Şekil 3.5 Gradyan inişinin geometrik gösterimi 24
Şekil 3.6 Momentum kullanılmadan gradyan inişi 26
Şekil 3.7 Momentum ile gradyan inişi 26
Şekil 3.8 Yerel Minimum, Global Minimum ve Düz Bölgeye Sahip Maliyet Fonksiyonunun Gösterimi 27
Şekil 4.1 Tek Katmanlı Fonksiyonel Bağlantılı Yapay Sinir Ağı Modeli 32
Şekil 4.2 Tek Katmanlı Laguerre Yapay Sinir Ağı Modeli 33
Şekil 4.3 Tek Katmanlı Chebyshev Yapay Sinir Ağı Modeli 35
Şekil 4.4 Tek Katmanlı Legendre Yapay Sinir Ağı Modeli 36
Şekil 6.1 Analitik sonuçlar ile, sırasıyla, LgNN, ChNN ve LeNN sonuçları arasındaki hataların yarı logoritmik grafikleri (Örnek 6.1) 53
Şekil 6.2 Analitik sonuçlar ile, sırasıyla, LgNN, ChNN ve LeNN sonuçları arasındaki hataların yarı logoritmik grafikleri (Örnek 6.2) 54
Şekil 6.3 Maple 11 [42] sonuçları ile, sırasıyla, LgNN, ChNN ve LeNN sonuçları arasındaki hataların yarı logoritmik grafikleri (Örnek 6.3) 55
Şekil 6.4 Maple 11 [42] sonuçları ile, sırasıyla, LgNN, ChNN ve LeNN sonuçları arasındaki hataların yarı logoritmik grafikleri (Örnek 6.4) 56

TABLO LİSTESİ

Tablo 2.1 Aktivasyon Fonksiyonları ve Türevleri 10
Tablo 6.1 Analitik, LgNN, ChNN ve LeNN Sonuçları (Örnek 6.1) 48
Tablo 6.2 Analitik sonuçlar ile, LgNN, ChNN ve LeNN sonuçları arasındaki hata (Örnek 6.1) 49
Tablo 6.3 Analitik, LgNN, ChNN ve LeNN sonuçları (Örnek 6.2) 50
Tablo 6.4 Analitik sonuçlar ile, LgNN, ChNN ve LeNN sonuçları arasındaki hata (Örnek 6.2) 50
Tablo 6.5 Maple 11[42] , LgNN, ChNN ve LeNN sonuçları (Örnek 6.3) 51
Tablo 6.6 Maple 11 ile, LgNN, ChNN ve LeNN sonuçları arasındaki hata (Örnek 6.3) 51
Tablo 6.7 Maple 11 [42] , LgNN, ChNN ve LeNN sonuçları (Örnek 6.4) 51
Tablo 6.8 Maple 11 [42] ile, LgNN, ChNN ve LeNN sonuçları arasındaki hata (Örnek 6.4) 52

ÖZET

ADİ DİFERANSİYEL DENKLEMLERİN FONKSİYONEL BAĞLANTILI YAPAY SİNİR AĞLARIYLA ÇÖZÜMÜ

Sena Nur KURMANÇ

Matematik Anabilim Dalı
Yüksek Lisans Tezi

Danışman: Doç. Dr. Muttalip ÖZAVŞAR

Bu çalışmada, adi diferansiyel denklemlerin nümerik çözümleri tek katmanlı fonksiyonel bağlantılı yapay sinir ağı (FLANN) kullanılarak elde edilmiştir. Ağı eğitmek için, geri yayılım algoritması ile Adaptif moment tahmin (Adam) optimizasyon tekniği kullanılmıştır.

Fonksiyonel bağlantılı yapay sinir ağı (FLANN) modeli olarak, Laguerre yapay sinir ağı (LgNN), Chebyshev yapay sinir ağı (ChNN) ve Legendre yapay sinir ağı (LeNN) ele alınmıştır. Bu modellerin karşılaştırılabilmesi için, Emden-Fowler ve Lane-Emden tipi denklemler üzerinde deneysel çalışmalar yapılmıştır.

Anahtar Kelimeler: Yapay sinir ağları, Adi diferansiyel denklemler, Geri yayılım algoritması, Adaptif moment tahmin (Adam), Fonksiyonel bağlantılı yapay sinir ağı (FLANN), Laguerre yapay sinir ağı (LgNN), Chebyshev yapay sinir ağı (ChNN), Legendre yapay sinir ağı (LeNN)

YILDIZ TEKNİK ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

ABSTRACT

SOLUTIONS OF ORDINARY DIFFERENTIAL EQUATIONS BY FUNCTIONAL LINK ARTIFICIAL NEURAL NETWORKS

Sena Nur KURMANÇ

Department of Mathematics
Master of Science Thesis

Supervisor: Assoc. Prof. Dr. Muttalip ÖZAVŞAR

In this thesis, numerical solutions of ordinary differential equations were obtained by using single layer functional link artificial neural network (FLANN). The back propagation algorithm with Adaptive moment estimation (Adam) optimization technique is used to train the neural network.

As the functional link artificial neural network (FLANN) model, Laguerre artificial neural network (LgNN), Chebyshev artificial neural network (ChNN), and Legendre artificial neural network (LeNN) are considered. In order to compare these artificial neural networks, some experimental studies are conducted on Emden-Fowler and Lane-Emden type equations.

Keywords: Artificial neural networks, Ordinary differential Equations, Back propagation algorithm, Adaptive moment estimation (Adam), Functional link artificial neural network (FLANN), Laguerre artificial neural network (LgNN), Chebyshev artificial neural network (ChNN), Legendre artificial neural network (LeNN)

YILDIZ TECHNICAL UNIVERSITY
GRADUATE SCHOOL OF SCIENCE AND ENGINEERING

1
GİRİŞ

1.1 Literatür Özeti

Diferansiyel denklemler, astrofizik, biyokimya, ekonomi gibi birçok farklı alanda karşılaşılan problemleri modellemek için kullanılmaktadır. Ancak, bu diferansiyel denklemlerin her zaman analitik çözümleri elde edilememektedir. Bu nedenle, Runge-Kutta, sonlu fark metodu gibi sayısal yöntemler diferansiyel denklemleri çözmek için tercih edilmektedir. Ayrıca, bilgisayar biliminin gelişmesi ile birlikte, yapay sinir ağlarını (YSA) içeren yöntemlerde kullanılmaya başlanmıştır. İlk olarak Lee and Kang [1] birinci mertebeden adi diferansiyel denklemleri çözmek için Hopfield yapay sinir ağı modelini önermiştir. Daha sonra, Meade ve Fernandez [2, 3] lineer ve lineer olmayan adi diferansiyel denklemleri çözmek, için ileri beslemeli yapay sinir ağı ve B_1-spline'lar kullanmıştır. Ardından Lagaris vd. [4] yapay sinir ağı içeren bir deneme çözümü oluşturarak adi ve kısmi diferansiyel denklemleri çözmüştür. Bu deneme çözümü iki bölümden oluşmaktadır. İlk bölüm diferansiyel denklemin başlangıç ve sınır koşullarını sağlamaktadır. İkinci bölümü ise ileri beslemeli bir yapay sinir ağı içermektedir.

Malek and Beidokhti [5] düşük ve yüksek mertebeden diferansiyel denklemleri çözmek için yapay sinir ağı ve Nelder-Mead optimizasyon tekniğine dayanan hibrit bir metod geliştirmiştir. Adi diferansiyel denklemleri çözmek için Yazdi vd. [6] yapay sinir ağlarına dayanan koçan en küçük ortalama kareler (Kernel Least Mean Squares) algoritmasını önermiştir. Kumar ve Yadav [7] diferansiyel denklemlerin çözümünde kullanılan Çok katmanlı yapay sinir ağı (Multilayer Perceptron) ve Radyal tabanlı fonksiyon sinir ağları (RBFNN) modellerini içeren kapsamlı bir çalışma sunmuştur.

Yapay sinir ağlarının diferansiyel denklemlerin çözümündeki etkisi görüldükçe, farklı tip yapay sinir ağları da bu amaçla kullanılmaya başlanmıştır. Örneğin, Fonksiyonel Bağlantılı Yapay Sinir Ağları (FLANN) diferansiyel denklemleri çözmek amacıyla kullanılan ağ türlerinden biridir. Bu ağ modelinin hesaplama maliyeti az olduğu için, uygulamada kolaylık sağlamaktadır. Qu vd. [8] kesirli diferansiyel denklemleri

çözmek için, kosinüs taban fonksiyonlu sinir ağı kullanmıştır. Mall and Chakraverty [9] başlangıç ve sınır değer problemlerini çözmek için Legendre yapay sinir ağı modelini (LeNN) geliştirmiştir. Hadian-Rasanan vd. [10] Lane-Emden diferansiyel denklemlerini çözmek için kesirli Legendre yapay sinir ağından faydalanmıştır. Mall and Chakraverty [11–13] adi ve kısmi diferansiyel denklemleri çözmek için, fonksiyonel bağlantılı yapay sinir ağında Chebyshev polinomlarını kullanmıştır. Legendre yapay sinir ağı (LeNN) ile aşırı öğrenme makinesi (extreme learning machine) algoritmasını kullanarak, Hou vd. [14] adi diferansiyel denklem sistemlerini çözmüştür. Chakraverty vd. [15], Lane-Emden denklemini ve Duffing osilatör denkleminin bir uygulama problemini çözmek için Laguerre yapay sinir ağı modelini (LgNN) önermiştir. Chen vd. [16] farklı bir Laguerre yapay sinir ağı modeli önermiş ve bu model ile Black-Scholes diferansiyel denklemini çözmüştür. Ayrıca, Mall and Chakraverty [17] Van der Pol Duffing osilatör denklemini Hermite yapay sinir ağını kullanarak çözmüştür.

1.2 Tezin Amacı

Bu Yüksek Lisans tezinin amacı, lineer ve lineer olmayan adi diferansiyel denklemlerin nümerik çözümlerini fonksiyonel bağlantılı yapay sinir ağı kullanarak elde etmek ve kullanılan fonksiyonel bağlantılı yapay sinir ağı modelleri ile elde edilen sonuçları kıyaslamaktır.

1.3 Hipotez

Fonksiyonel bağlantılı yapay sinir ağı kullanılarak adi diferansiyel denklemlerin nümerik çözümleri elde edilebilir.

2
YAPAY SİNİR AĞLARI

2.1 Tanım

Canlılar, beyinleri aracılığıyla öğrenme, öğrendiklerini farklı durumlarda kullanma ve hatırlama gibi becerileri gerçekleştirirler. Canlılardaki bu özellikler, eski zamanlardan günümüze kadar insanların dikkatini çekmeyi başarmıştır. İnsanların çevrelerinde gördüklerinden ilham alma ve bu gördüklerini taklit etme isteği, bilim insanlarını beyne benzer yapılar üretmeye sevk etmiştir. Bu konuda yapılan çalışmalar sonucunda, yapay zeka ve yapay zekanın bir alt dalı olan yapay sinir ağları ortaya çıkmıştır.

Literatürde yapay sinir ağlarının farklı tanımları yer almaktadır. Bu tanımlardan biri, Simon Haykin'in [18] yapay sinir ağı ile beynin benzerliğine dikkat çektiği tanımdır: "Sinir ağı, basit işlem birimlerinden oluşan, deneyimsel bilgileri depolamaya yönelik doğal bir eğilimi olan ve bu bilgilerin kullanılmasını sağlayan büyük ölçekte paralel dağıtılmış bir işlemcidir. İki yönden beyin ile benzerlik göstermektedir:

1. Bilgi, ağ tarafından öğrenme süreciyle çevreden elde edilir.
2. Elde edilen bilgileri biriktirmek için sinaptik ağırlıklar olarak da bilinen nöronlar arası bağlantı güçleri kullanılır."

Biyolojik sinir ağları, birbirine paralel olarak bağlı olan nöronlardan oluşmaktadır. Nöronlarda bilgi işlenir ve nöronları birbirine bağlayan sinapslar aracılığıyla, bilgi diğer nöronlara iletilir. Sinir ağlarının bu işleyişi, yapay sinir ağlarının oluşturulmasına esin kaynağı olmuştur. Yapay sinir ağları da benzer olarak yapay nöron adı verilen basit işlem birimlerinden ve bu yapay nöronlar arası bağlantıyı sağlayan sinaptik ağırlık değerlerinden oluşacak şekilde tasarlanmıştır. Birbirine paralel olarak bağlı nöronlardan oluşması nedeniyle, yapay sinir ağlarına paralel dağıtılmış ağlar da denmektedir.

Canlıların sinir sistemi, dışarıdan aldığı sinyalleri işler ve bu sinyallere uygun yanıt

üretir. Benzer şekilde, yapay sinir ağları da dışarıdan verilen girdi değerlerine karşılık uygun çıktı değerlerini üreten modellerdir. Çıktı değerleri, alınan girdi değerlerine ve sinaptik ağırlıklara bağlı olarak hesaplanır. Yapay sinir ağının sinaptik ağırlıkları, bir problemin örnek verilerine göre ayarlanabilir. Optimal ağırlık değerleri bulunduğunda, yapay sinir ağında öğrenme gerçekleşir. Yapay sinir ağı öğrenme özellliği sayesinde, karmaşık problemleri çözebilmektedir.

Yapay sinir ağını oluşturan nöronların yapısına ve nöronların birbirleri ile bağlantı biçimlerine göre farklı yapay sinir ağı çeşitleri oluşturulmuştur. Her bir YSA modeli avantajları ve dezavantajları göz önüne alınarak, birçok farklı alanda ve problem çeşidinde kullanılabilmektedir.

2.2 Tarihsel Gelişim

İnsan beyninin ve sinir sisteminin işleyişi, uzun yıllar bilim insanlarının araştırma konusu olmuştur. Zamanla bu alanda elde edilen bilgilerin artması, araştırmacılara sinir sistemini modelleyebilme olanağı tanımıştır. Bu sayede, yapay sinir ağları alanında yapılan araştırmalar da artmıştır.

Yapay sinir ağları üzerine yapılan önemli çalışmalardan ilki, 1943 yılında Warren S. McCulloch ve Walter Pitts [19] tarafından yayımlanan makaledir. Bu çalışmada, bir sinir hücresinin işleyişine dayanan, matematiksel bir modelleme ortaya konulmuştur. McCulloch-Pitts modeli tek bir nörondan oluşan iki farklı çıktı değeri alabilen basit bir yapıdır. Ancak, daha sonra geliştirilen yapay sinir ağı modellerine öncülük etmiştir.

Yapay sinir ağı modellerinin geliştirilmesiyle, bu modeller için uygun ağırlık değerlerinin belirlenmesi konusu önem kazanmıştır. Donald Hebb [20], 1949 yılında bu sorunu çözmek için bir yöntem geliştirmiştir ve bu yönteme Hebbian öğrenme kuralı denmektedir. Bu öğrenme kuralı, canlıların şu özelliğini esas almaktadır: Sinir ağlarında daha sık kulllanılan bağlantılar pekiştirilir ve bu sayede canlılar, alışkanlık edinme ve tekrar yoluyla öğrenme gibi becerileri gerçekleştirirler. Ağırlık değerlerini güncellemek için, bu öğrenme kuralı ağın girdi ve çıktı değerlerini kullanmaktadır. Dolayısıyla, Hebbian öğrenme kuralı bir denetimsiz öğrenmedir. Hebbian öğrenme kuralı, kendisinden sonra oluşturulan birçok öğrenme algoritmasına zemin oluşturmuş ve günümüzde de kullanılmaktadır.

1958 yılında Frank Rosenblatt [21] tek katmanlı yapay sinir ağı olan ”algılayıcı” adını verdiği modeli geliştirmiştir. Aynı zamanda, bu modelin ağırlık değerlerini ”algılayıcı öğrenme kuralı” adını verdiği bir yöntemle belirlemiştir. Ayrıca algılayıcının bazı sınıflandırma problemlerinin çözümünde kullanılabileceğini ortaya koymuştur.

Rosenblatt'ın bu katkısı, yapay sinir ağlarına olan ilginin artmasına neden olmuştur.

1959 yılında Bernard Widrow ve Marcian Hoff [22], kısaltılmış hali ADALINE olan adaptif doğrusal nöron (AdAptive Linear Neuron) adlı yapay sinir ağını geliştirmiştir. Ayrıca, ADALINE ağının eğitilmesinde kullanmak için, en küçük ortalama kareler (Least Mean Square-LMS) adlı öğrenme algoritmasını da öne sürmüştür. Bu algoritmaya delta kuralı da denmektedir. İlerleyen zamanlarda bu yapay sinir ağı modeli uyarlanabilir sinyal işlemede kullanılmıştır.

Marvin Minsky ve Samuel Papert [23], 1969 yılında algılayıcı adlı bir kitap yayınladı. Bu kitapta algılayıcıların çözmesi mümkün olmayan birçok probleme yer verdi. Bu kitabın yayımlanmasından sonra yapay sinir ağlarına olan ilgi azaldı ve bu süreçte birkaç araştırmacı dışında, araştırmacıların çoğu başka alanlara yöneldi.

1980'li yıllarda, birkaç bilim insanının yapay sinir ağları alanında olan çalışmalarını yayımlaması ile, bu alana olan ilgi tekrar arttı. Bu çalışmalardan biri, 1982 yılında Teuvo Kohonen [24] tarafından yayınlanan kendi kendine öğrenme nitelik haritaları (Self-Organising Maps- SOM) adlı makaledir. Diğer önemli çalışma ise, 1982'de John Hopfield'ın [25] geliştirdiği Hopfield sinir ağıdır. Hopfield, bu ağı kullanarak gezgin satıcı problemini çözdü ve yapay sinir ağlarının zor problemleri çözebileceğini göstermiş oldu. Bu şekilde, yapay sinir ağlarına olan ilginin yeniden artmasını sağladı.

Algılayıcılar, bazı problemlerin çözümünde yetersiz kaldıklarında, birden fazla algılayıcının birleştirilmesi ile oluşan çok katmanlı algılayıcılar ileri sürülmüştür. Çok katmanlı algılayıcıları eğitmek için kullanılan geri yayılım algoritması ise, 1974'te Werbos [26], 1985'te Parker [27] ve 1986 yılında Rumelhart ve çalışma arkadaşları [28] tarafından birbirlerinden bağımsız olarak ortaya konulmuştur. Ancak geri yayılım algoritmasına olan ilgi, Rumelhart ve arkadaşları tarafından yazılan kitap ile artmıştır. Bu algoritmanın geliştirilmesi ile, Minsky ve Papert'ın kitaplarında yer verdiği problemler çözülebilmiştir. Dolayısıyla bu öğrenme algoritması yapay sinir ağlarına olan inancın artmasına sebep olmuştur. 1988 yılında D. S. Broomhead ve David Lowe [29], radyal tabanlı fonksiyon yapay sinir ağı (Radial Basis Function Neural Networks - RBFNN) türünü geliştirdiler. Bu ağ, çok katmanlı algılayıcılara alternatif olarak öne sürülmüştür.

Yapay sinir ağları alanının gelişmesini sağlayan bu çalışmalar sayesinde, günümüze kadar farklı YSA türleri geliştirilmiş ve yapay sinir ağları farklı alanlarda kullanılmıştır. Özellikle bilgisayar biliminin gelişmesi, yapay sinir ağları alanında çalışmayı kolaylaştırmıştır.

2.3 Yapay Sinir Ağlarının Kullanım Alanları

Hata toleransı ve öğrenme gibi, yapay sinir ağının sahip olduğu özellikler sayesinde, yapay sinir ağları gerçek hayat problemlerinde kolaylıkla kullanılabilmektedir. Günümüzde, yapay sinir ağları sağlık, güvenlik ve finans gibi farklı alanlarda kullanılmaktadır ve bu alanlardaki kullanım amaçlarını genel olarak şu şekilde sınıflandırabiliriz:

- **Örüntü tanıma:** Ağa verilen bir modeli, önceden belli olan sınıflardan birine atama işlemidir. Bu süreçte ağ, her bir sınıfa ait girdi örnekleri ile eğitilir. Bu sayede, ağa daha önce sunulmayan bir girdi verildiğinde, ağ bu girdinin hangi sınıfa ait olduğunu belirleyebilir. İnsanlar örüntü tanıma işlemini günlük hayatta kolaylıkla gerçekleştirebilirler. Örneğin, daha önce çocukluk halini görmediğimiz bir kişinin çocukluk fotoğraflarına baktığımızda, o kişiyi fotoğraflarda bulabiliriz. Ses tanıma veya el yazısı tanıma gibi uygulamalar da kullanılan yapay sinir ağları, bu alana örnek olarak verilebilir.

- **Kontrol:** Yapay sinir ağları, robot ve uçak teknolojilerinde bu amaçla kullanılmaktadır. Beyin bu işlemi, canlıların aynı anda birçok hareketini kontrol ederek yapmaktadır. Yapay sinir ağları kontrol amacı ile kullanıldığında, sisteme uygun yönlendirme sinyallerini verecek şekilde eğitilirler.

- **Veri ilişkilendirme:** Ağa sunulan verilerin hatalı veya eksik kısmını belirleme ve bu kısımları tamamlama işlemidir. Yapay sinir ağlarının eksik veya hatalı kısımları belirleyebilmesi için, ağa veriler hatasız şekilde ard arda sunulur. Bu şekilde ağı eğittikten sonra, ağ hatalı verileri belirleyerek tamamlayabilir. Görüntü işlemede yapay sinir ağları bu amaçla kullanılmaktadır.

- **Tahmin:** Yapay sinir ağlarının tahmin etmesi istendiğinde, öncelikle ağa belli bir sistemin verileri sunulur. Bu verilere dayanarak ağ, sistemin gelecekteki verilerini tahmin edebilir. Örneğin, yapay sinir ağları zaman serisi tahmini, hava durumu tahmini alanlarında bu amaçla kullanılmaktadır.

- **Kümeleme:** YSA'nın girdi değerleri arasındaki benzerlikleri bulup, girdileri gruplara ayırmasına kümeleme denmektedir. YSA, veri madenciliğinde sıklıkla bu amaçla kullanılmaktadır.

- **Fonksiyon yaklaşımı:** Yapay sinir ağlarının yapısı, fonksiyon yaklaşımı amacıyla kullanılmaya oldukça uygundur. Bu problem türünde girdi ve çıktı değerleri bellidir, fakat girdileri çıktılara dönüştüren fonksiyon bilinmemektedir. Yapay sinir ağları, girdi değerlerini çıktı değerlerine dönüştüren modeller olduğu için, fonksiyon yaklaşımı amacıyla birçok yapay sinir ağı türü kullanılabilmektedir.

2.4 Yapay Sinir Ağlarının Genel Özellikleri

Yapay sinir ağı, öğrenme, genelleme ve paralellik gibi beyne benzer özelliklere sahiptir. Bu özellikleri sayesinde, problem çözümlerinde geleneksel algoritmalara karşı avantaj sağlamaktadır. Her bir yapay sinir ağı modelinin kendine özgü özellikleri olmasına rağmen, tüm modellerin sahip olduğu bazı genel özelliklerden bahsedebiliriz:

2.4.1 Paralellik Özelliği

Yapay sinir ağlarına paralel dağıtılmış işlemciler de denmektedir. Ağı oluşturan nöronların herbiri diğer nöronlardan bağımsız olarak eşzamanlı hesaplama yapar. Ardından bulunan çıktı, diğer nöronlara aktarılır. Ağda yer alan nöronlar, bir problemin bütününü ele almak yerine; her bir nöron problemin farklı bir kısmını ele alarak paralellik özelliği gösterir. YSA'nın paralel yapısı, geleneksel seri işlemciye sahip sistemlere göre daha hızlı işlem yapabilmesini sağlar.

2.4.2 Doğrusal Olmama

Yapay nöronlar doğrusal veya doğrusal olmayacak şekide ayarlanabilirler. Eğer yapay sinir ağları doğrusal olmayan yapay nöronlar tarafından oluşturulursa, elde edilen sistem de doğrusal olmaz. Yapay sinir ağlarının doğrusal olmaması, birçok karmaşık ve doğrusal olmayan problemde kullanılabilmesine olanak sağlamaktadır.

2.4.3 Hata Toleransı

Seri işlemciye sahip sistemlerde, yapılan bir hata bütün sistemin bozulmasına neden olur iken, yapay sinir ağları paralel yapısı sayesinde hatalara karşı daha dayanıklıdır. Nöronlarda veya bağlantı değerlerinde oluşan hasarlar, yapay sinir ağlarında ufak sapmalara neden olur. Ayrıca verilerin eksik veya hatalı olması durumunda, yapay sinir ağları veriler arasındaki ilişkiyi bularak mantıklı sonuçlar üretebilir.

2.4.4 Öğrenme, Genelleme ve Uyum Sağlama Özellikleri

Yapay sinir ağlarında öğrenme, ağa sunulan örnek durumlar aracılığıyla gerçekleşir. Yapay sinir ağı bir problemin örnek verileri arasındaki ilişkiyi keşfederek, sinaptik bağlantı değerlerini ayarlar. Uygun sinaptik ağırlık değerlerinin belirlenmesi ile, yapay sinir ağlarında öğrenme gerçekleşir ve bu sürece yapay sinir ağının eğitilmesi adı verillir.

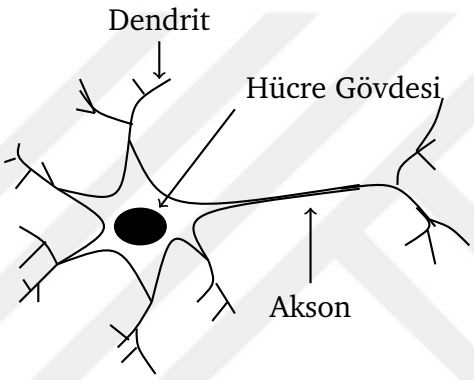
Yapay sinir ağları bir problem için eğitildikten sonra genelleme yapabilir. Bu sayede, bu problemin farklı verileri için de uygun sonuçlar üretebilir. Ayrıca verilerde eksik

veya hatalı kısımlara rağmen uygun çıktıyı üretebilir.

Yapay sinir ağları, eğitildiği bir problemde yapılan değişikliklere uyum sağlayabilirler. Yapılan değişikliğe göre, YSA ağırlıklarını tekrar ayarlayabilir.

2.5 Biyolojik Sinir Hücresi

Beyin fonksiyonlarının ve sinir sisteminin temel ögesi sinir hücreleridir. Bu hücrelere nöron da denmektedir. İnsan merkezi sinir sisteminin yaklaşık 100 milyar sinir hücresinden oluştuğu tahmin edilmektedir [30]. Sinir hücrelerinin aralarında yapısal olarak farklılıkları olmasına rağmen, temel bir işleyiş tarzları vardır. Bir sinir hücresi, hücre gövdesi, akson ve dendritlerden oluşur (Şekil 2.1).

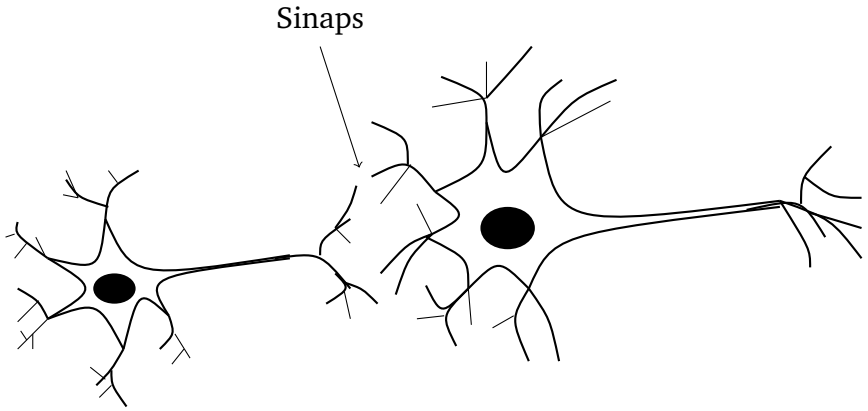


Şekil 2.1 Biyolojik Sinir Hücresi

Dendritler hücre gövdesini çevreleyen uzantılardır. Diğer sinir hücrelerinden veya dış ortamdan aldıkları elektrokimyasal sinyalleri hücre gövdesine iletirler.

Çekirdek ve hücre organellerini içeren hücre gövdesi, hücrenin yönetimini sağlamaktadır. Hücre gövdesi, dendritler aracılığıyla sürekli olarak sinyaller alır. Gelen uyarılar o nöronun eşik değerine ulaşırsa bir elektriksel çıktı oluşturulur ve aksona iletilir. Eşik değeri, çıktı oluşması için gereken minimum uyaran şiddetidir. Her bir nöronun spesifik bir eşik değeri vardır. Eşik değerine ulaşamayan uyarılara sinir hücresi yanıt vermez.

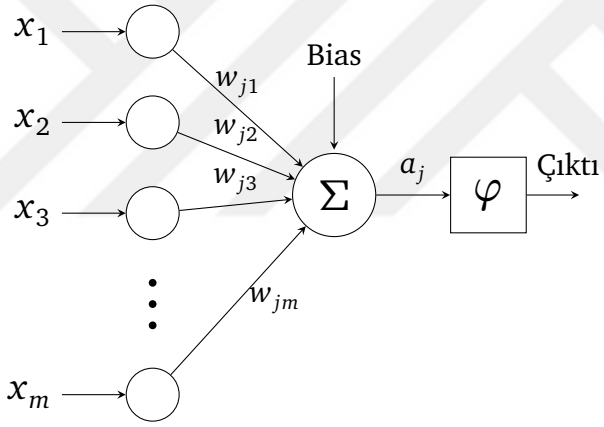
Akson, hücre gövdesinden çıkan uzantıdır. Nöronda oluşan çıktının diğer sinir hücrelerine iletilmesini sağlamaktadır. Aksonların son kısımlarında uzantıları vardır. Bu uzantılar aracılığıyla diğer hücrelerle bağlantı kurarlar. Aksonun uzantıları ile diğer bir sinir hücresinin dendritleri arasında sinaps adı verilen boşluklar vardır (Şekil 2.2). Bu sinaptik boşluklarda elektrokimyasal bir iletim gerçekleşir.



Şekil 2.2 Nöronlar arası sinaptik bağlantının gösterimi

2.6 Yapay Nöron

Yapay sinir ağlarının temel elemanı olan yapay sinir hücrelerine, işlem birimi de denilmektedir. Sinir hücresinin yapısının basitleştirilmesiyle oluşturulmuş bir modellemedir (Şekil 2.3):



Şekil 2.3 Yapay Nöron Modeli

Yapay nöron aşağıdaki temel bölümlerden oluşmaktadır:

- **Ağırlık:** Her bir girdinin, yapay nöron üzerindeki etkisi, o girdiye ait ağırlık değeri tarafından belirlenir. Pozitif ağırlık girdinin etkisini artırır iken, negatif ağırlık girdinin etkisini azaltır. w_{ji} ağırlık değeri, j. yapay nöronun x_i girdisinin bağlantı değerini sembolize etmektedir.
- **Toplam Fonksiyonu:** Yapay nöronun kullanım amacına göre farklı toplam fonksiyonları kullanabilir. Genellikle ağırlıklı toplam fonksiyonu kullanılır. Bu fonksiyon, her bir girdi değeri ile o girdi değerine ait ağırlık değerini çarparak lineer toplamlarını hesaplar. Toplam fonksiyonu ile elde edilen sonuca nöronun net girdisi a_j denir.

- **Aktivasyon Fonksiyonu:** Net girdinin aktivasyon fonksiyonu (transfer fonksiyonu) değeri hesaplanarak nöronun çıktısı elde edilir. Aktivasyon fonksiyonu geniş değer aralığına sahip girdilerin daha sınırlı bir değer aralığına aktarılmasını sağlar. Literatürde, farklı aktivasyon fonksiyonları bulunmaktadır. Aktivasyon fonksiyonu, ağın kullanıldığı uygulama alanı ile aktivasyon fonksiyonunun avantaj ve dezavantajları göz önüne alınarak seçilir.

- **Bias:** Net girdi hesaplanırken, bias değeri toplam fonksiyonu ile elde edilen sonuca eklenebilir. Net girdiye eklenmesi, aktivasyon fonksiyonun sonucunu arttırır veya azaltır. Ağırlık değerleri gibi, bias değerinin de ağ için en uygun değeri bulunmaya çalışılır. Her yapay nörona bias değeri eklenmek zorunda değildir.

Literatürde sık kullanılan birkaç aktivasyon fonksiyonu ve bu fonksiyonların türevleri Tablo 2.1'te verilmiştir.

Tablo 2.1 Aktivasyon Fonksiyonları ve Türevleri

Aktivasyon Fonksiyonu	**Türev**	**Açıklama**
Lineer Fonksiyon $\varphi(x) = x$	$\varphi'(x) = 1$	Yapay nöronun çıktısı net girdiye eşit olur.
Eşik Fonksiyonu $\varphi(x) = \begin{cases} 1, \text{ if } x \geq 0 \\ 0, \text{ if } x < 0 \end{cases}$	x=0 noktasında türevi yoktur.	Net girdinin büyüklüğüne göre yapay nöron 0 veya 1 değerini alır. Sınıflandırma problemlerinde kullanılır.
Sigmoid Fonksiyonu $\varphi(x) = \frac{1}{1+\exp(-x)}$	$\varphi'(x) = \varphi(x)[1-\varphi(x)]$	Çok sık kullanılan bir aktivasyon fonksiyonudur. $[0,1]$ aralığında değerler alır. Sürekli ve lineer olmayan bir fonksiyondur.
Hiperbolik tanjant Fonksiyonu $\varphi(x) = tanh(x) = \frac{\exp(x)-\exp(-x)}{\exp(x)+\exp(-x)}$	$\varphi'(x) = 1-\varphi^2(x)$	$[-1,1]$ aralığında değer alır.
RELU $\varphi(x) = max(0,x)$	$\varphi'(x) = \begin{cases} 1, \text{ if } x \geq 0 \\ 0, \text{ if } x < 0 \end{cases}$	$[0, \infty)$ aralığında değerler alır. Hesaplama kolaylığı nedeniyle sık kullanılan bir aktivasyon fonksiyonudur.
Gauss Radyal Temel Fonksiyonu $\varphi(x) = \exp(\frac{\|\mathbf{x}-\mathbf{c}\|^2}{2\delta^2})$	$\varphi'(x) = \frac{-2(\mathbf{x}-\mathbf{c})\varphi(x)}{\delta^2}$	Radyal tabanlı fonksiyon yapay sinir ağları (RBFNN) için sık kullanılan temel fonksiyonlardan biridir. c ve δ parametreleri probleme göre belirlenmektedir.

(Şekil 2.3) esas alınarak, j. nöronun a_j net girdisi ve y_j çıktısı aşağıdaki gibi yazılabilir:

$$\begin{aligned} a_j &= \sum_{i=1}^{m} x_i w_{ji} + b_j \\ y_j &= \varphi(a_j) \end{aligned} \tag{2.1}$$

Denklem 2.1 'de; $x_1, x_2, x_3, \ldots, x_m$ girdi değerlerini, $w_{j1}, w_{j2}, w_{j3}, \ldots, w_{jm}$ ise ağırlık değerlerini belirtmektedir. Net girdiyi hesaplamak için ağırlıklı toplam fonksiyonu kullanılmıştır. Burada, j. nörona ait bias değeri b_j ile gösterilmektedir. j. nöronun y_j çıktısını hesaplamak için, net girdinin φ aktivasyon fonksiyon değeri hesaplanmaktadır.

Denklem 2.1'de görüldüğü gibi, bias değeri toplam fonksiyonunun sonucuna eklenmektedir. Bunun yerine, bias değerini denklemde ağın bir ağırlık değeri gibi yazabiliriz:

$$\begin{aligned} a_j &= \sum_{i=0}^{m} x_i w_{ji} \\ y_j &= \varphi(a_j) \end{aligned} \tag{2.2}$$

Burada, $b_j = w_{j0}$ olarak alınmıştır. Bias değerine ait girdi değeri ise $x_0 = +1$ olarak alınabilir.

2.7 Yapay Sinir Ağının Yapısı

Yapay sinir ağları, birbirlerine ağırlık değerleri ile bağlanan yapay sinir hücrelerinden oluşmaktadır. Yapay sinir hücreleri katmanlar oluşturur ve bu katmanlar arasında bağlantılar vardır. Genel olarak yapay sinir ağları, girdi, gizli ve çıktı katmanı olmak üzere üç katmandan oluşur:

- **Girdi katmanı:** Bu katmanda yer alan nöronlar, dış dünyadan aldıkları verileri diğer katmanlara iletirler. Bu katmandaki nöronlar, gelen veriler üzerinde herhengi bir işlem yapmaz. Bu nedenle, bazı araştırmacılar ağın katman sayısını hesaplarken, girdi katmanını bu hesaplamaya dahil etmezler.
- **Gizli katmanlar:** Girdi katmanı ile çıktı katmanı arasındaki her bir katmana gizli

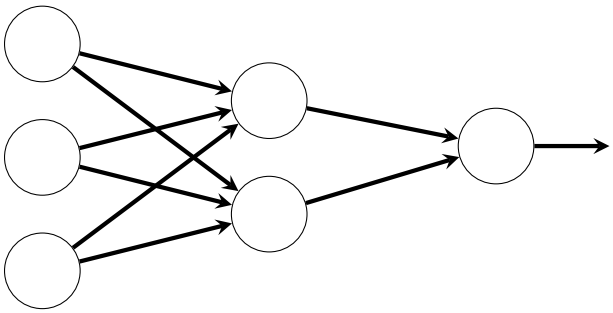
katman denir. Girdi katmanından gelen bilgiler burada işlenir. Bu katmandaki nöronlar, gelen bilgiler arasındaki ilişkileri ortaya çıkarırlar. Ağın kullanıldığı problem türüne göre, gizli katman sayısı ve gizli katmandaki nöron sayısı kullanıcı tarafından belirlenir. Ayrıca bazı yapay sinir ağı modellerinde gizli katman yer almayabilir.

- **Çıktı katmanı:** Bu katmanda, gizli katmandan gelen veriler işlenir ve çıktı değerleri elde edilir. Çıktı katmanındaki nöron sayısı, ağın kullanıldığı probleme göre belirlenir.

Yapay sinir ağlarının yapısı, bilginin akış yönü ve yapay nöronların birbirleri ile olan bağlantı biçimlerine göre farklılıklar göstermektedir. Yapılarına göre yapay sinir ağları, ileri beslemeli ve geri beslemeli olmak üzere ikiye ayrılır:

2.7.1 İleri Beslemeli Yapay Sinir Ağları

Bu tip yapay sinir ağlarında, katmanlar arasındaki bağlantı tek bir yönde ilerler. Bağlantıların yönü girdi katmanından çıktı katmanına doğrudur. Her bir nöron, kendinden bir önceki katmandaki nöronlardan gelen bilgileri alır ve bir sonraki katmana iletir. Geriye doğru bilgi akışı veya aynı katmandaki nöronlar arasında bilgi akışı yoktur. Çok katmanlı algılayıcı (Multilayer Perceptrons-MLP) ve Radyal tabanlı fonksiyon yapay sinir ağı (Radial Basis Function Neural Networks - RBFNN) [29] ileri beslemeli yapay sinir ağı türlerindendir. İleri beslemeli yapay sinir ağına örnek Şekil 2.4 ile verilmiştir.

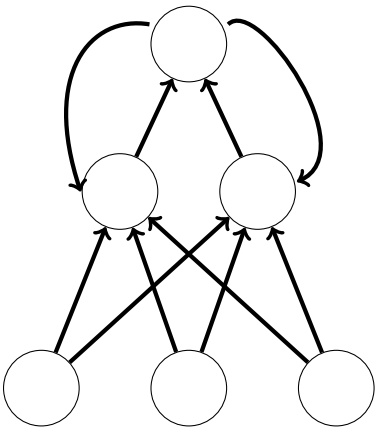


Şekil 2.4 Tek Gizli Katmana Sahip İleri Beslemeli Yapay Sinir Ağı

2.7.2 Geri Beslemeli (Yineleyen) Yapay Sinir Ağları

”Geri beslemeli sinir ağlarının, ileri beslemeli ağlarda yer alan bağlantılara ek olarak, nöronlarının kendi kendilerine bağlantıları veya önceki katmanlardaki nöronlarla bağlantıları vardır. Bu yineleme, kısa süreli bir bellek görevi görür ve ağın geçmişte olanları hatırlamasını sağlar” [31]. Zaman serisi tahmini, sistem tanımlama gibi

alanlarda, geri beslemeli yapay sinir ağları kullanılmaktadır. Şekil 2.5 ile geri beslemeli yapay sinir ağlarına bir örnek verilmiştir.



Şekil 2.5 Çıktı Katmanından Gizli Katmana Bağlantıların Olduğu Çok Katmanlı Algılayıcı Modeli

2.8 Yapay Sinir Ağlarında Öğrenme

Yapay sinir ağlarının en önemli avantajlarından biri, ağın öğrenme özelliğidir. Yapay sinir ağlarında öğrenme, örnek veriler üzerinden gerçekleşir. Ağa, belli bir sistemin örnek verileri sunulur. Ardından, ağ bu örnekler arasındaki ilişkiyi saptamaya çalışır. Ağ istenen çıktıyı elde edene kadar, ağırlık değerlerini tekrarlı bir şekilde değiştirir. Bu işlem, optimal ağırlık değerleri bulunana kadar devam eder. Bu sürece ağın eğitilmesi adı verilir. Ayrıca, ağın eğitilmesi sırasında yapılan işlemlerin tamamına öğrenme algoritması denmektedir. Ağın optimal ağırlık değerlerini elde etmesine de, ağın öğrenmesi denmektedir.

Sistemin örnekleri, genellikle eğitim seti ve test seti olmak üzere iki kümeye ayırılır. İlk olarak, ağa eğitim seti sunulur ve bu set kullanılarak, ağın sistemi öğrenmesi istenir. Bu set, test setine göre daha çok örnek veri içerir. Ardından ağın öğrenmesini sınamak için, test seti ağa sunulur.

Yapay sinir ağlarında 3 farklı öğrenme türü vardır; denetimli öğrenme, denetimsiz öğrenme ve destekli öğrenme.

2.8.1 Denetimli Öğrenme:

Bu öğrenme türüne öğretmen ile öğrenme adı da verilmektedir. Denetimli öğrenmede, dışarıdan ağa sunulan veriler girdi ve çıktı ikililerinden oluşmaktadır. Girdi ve istenen çıktı değerleri bilindiği için; bu veriler, YSA üzerinde bir nevi öğretmen görevi görürler. Ağ istenen çıktı değerlerine yaklaşmak için, elde ettiği çıktı değerleri ile istenen çıktı

değerleri arasındaki hatayı minimize etmeye çalışır. Hata değerini minimize etmek için, ağırlık ve eşik değerlerini sürekli olarak günceller. Bu güncellemeler, optimal ağırlık değerleri elde edilene kadar tekrarlanır.

2.8.2 Denetimsiz Öğrenme:

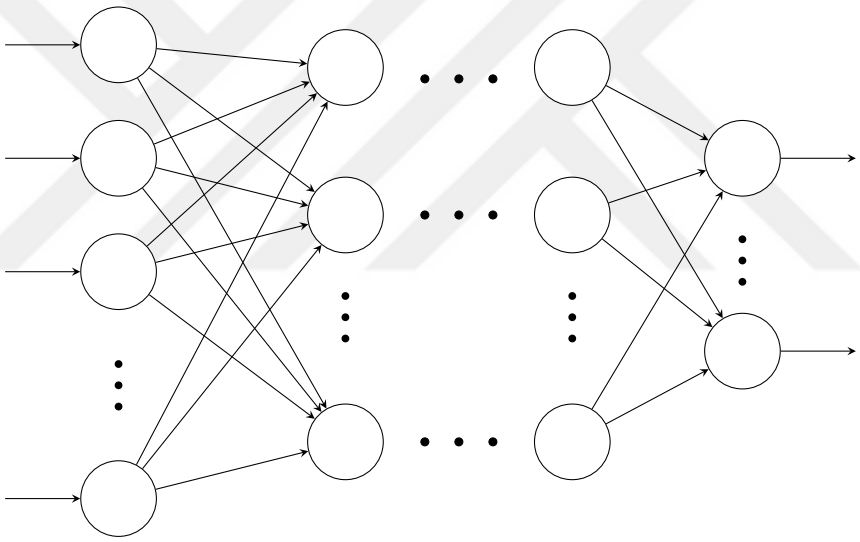
Denetimli öğrenmenin aksine bu öğrenme türünde girdilere karşılık gelen çıktı değerleri bilinmemektedir. Ağ girdi değerleri arasındaki ilişkiyi veya modelleri bularak çıktıları oluşturur. Kümeleme (clustering) problemlerinde kullanılan YSA, bu problem türünde denetimsiz öğrenme gerçekleştirmektedir.

2.8.3 Destekli Öğrenme:

Denetimsiz öğrenmeye benzer şekilde, bu öğrenme türünde de ağ istenen çıktı değerlerini bilmemektedir. Ancak yapay sinir ağına yardımcı olacak şekilde, dışarıdan gelen bir bilgi, kural vb. vardır. Destekli öğrenme bu yönüyle de denetimli öğrenmeye benzemektedir. Çünkü dışarıdan gelen bilgi, ağ üzerinde öğretmen görevi görür. Destekli öğrenmede, genellikle ağın eğitilmesine yardımcı olan bilgi, ağın hata değeridir. Çıkan hata değerine göre, ağın ağırlıklarında değişiklikler yapılır.

3
ÇOK KATMANLI ALGILAYICI - ÇKA (MULTILAYER PERCEPTRON - MLP)

ÇKA, ileri beslemeli bir yapay sinir ağı türüdür. Çok katmanlı algılayıcıların girdi katmanı ile çıktı katmanı arasında bir veya daha fazla gizli katman bulunmaktadır (Şekil 3.1).

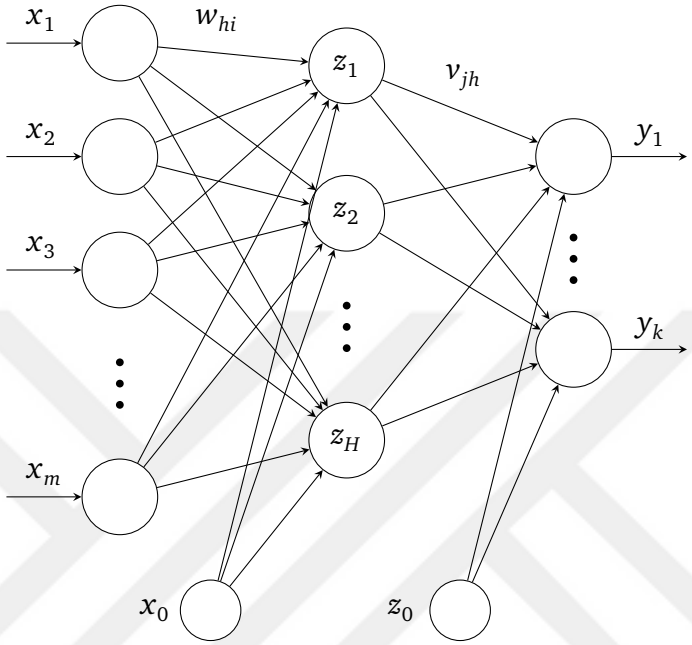


Şekil 3.1 Çok Katmanlı Algılayıcının Genel Görünümü

Girdi katmanında yer alan girdiler, ilk olarak birinci gizli katmana iletilir. Bu katmandaki her bir gizli nöron, girdi değeri ile her bir girdiye ait ağırlık ve bias değerini hesaba katarak gizli nöronun çıktısını hesaplar. Ardından bu değerler ikinci gizli katmandaki nöronlara iletilir. Böylece bilgi, girdi katmanından çıktı katmanına doğru ilerler. Çok katmanlı algılayıcılara bu sebeple ileri beslemeli sinir ağı da denilmektedir. ”Gizli nöronlar öznitelik detektörü işlevi görürler; bu nedenle, çok katmanlı bir algılayıcının işleyişinde kritik rol oynarlar. Öğrenme süreci çok katmanlı algılayıcıda ilerledikçe, gizli nöronlar eğitim verilerini karakterize eden belirgin özellikleri aşamalı olarak "keşfetmeye" başlarlar. Bunu, girdi verilerinden nitelik uzayı

adı verilen yeni bir alana doğrusal olmayan bir dönüşüm gerçekleştirerek yaparlar" [18]. Bu dönüşümün doğrusal olmamasının sebebi, gizli katmanlarda kullanılan doğrusal olmayan aktivasyon fonksiyonlarıdır.

ÇKA'nın çalışma prensibini daha kolay göstermek için, tek gizli katmana sahip bir çok katmanlı algılayıcı ele alalım (Şekil 3.2).



Şekil 3.2 Tek Gizli Katmana Sahip Çok Katmanlı Algılayıcı Modeli

x_i, $i = 1, 2, .., m$ ağın girdi değerlerini, z_h, $h = 1, 2, .., H$ ise gizli katmanın nöronlarını belirtmektedir. Ağın çıktı nöronları da y_j, $j = 1, 2, ..., k$ ile gösterilmektedir. Burada x_0 ve z_0 sırasıyla, girdi katmanının ve gizli katmanın bias nöronlarıdır. w_{hi} girdi katmanını gizli katmana bağlayan ağırlık değerlerini, v_{jh} ise gizli katmanı çıktı katmanına bağlayan ağırlık değerlerini belirtmektedir. O halde, ağın çıktısını şu şekilde formülize edebiliriz:

$$y_j = \sigma\left(\sum_{h=1}^{H} v_{jh}\varphi\Big(\sum_{i=1}^{m} w_{hi}x_i + w_{h0}\Big) + v_{j0}\right) \quad j = 1, 2, .., k. \tag{3.1}$$

Formüldeki φ sembolü, gizli katmanda kullanılan aktivasyon fonksiyonunu, σ sembolü ise çıktı katmanında kullanılan aktivasyon fonksiyonunu belirtmektedir. Daha önce denklem 2.2'de gösterildiği gibi; $x_0 = 1$ ve $z_0 = 1$ alınarak, bias değerlerini ağın ağırlık değerleri kümesinin içine ekleyebiliriz. O halde, denklem 3.1'i şu şekilde tekrar yazabiliriz:

$$y_j = \sigma\Big(\sum_{h=0}^{H} v_{jh}\varphi\big(\sum_{i=0}^{m} w_{hi}x_i\big)\Big) \quad j = 1, 2, .., k. \tag{3.2}$$

3.1 Çok Katmanlı Algılayıcının Öğrenme Kuralı

Geri yayılım algoritması, çok katmanlı algılayıcıları eğitmek için en çok kullanılan yöntemlerden biridir. Ayrıca, bu algoritmaya genelleştirilmiş delta kuralı da denmektedir. Geri yayılım algoritması bir denetimli öğrenme türüdür. Dolayısıyla istenen çıktı değerleri ağa sunulur ve ağın çıktı değerlerinin istenen çıktı değerlerine yakınsaması amaçlanır. Geri yayılım algoritması iki aşamadan oluşur; ileri doğru hesaplama ve geriye doğru hesaplama.

1. ***İleri Doğru Hesaplama***

   İlk aşamada, çok katmanlı algılayıcıya girdi değerleri sunulur ve ağın çıktı değerleri hesaplanır. Bu hesaplama sırasında, ağa atanan ağırlık ve bias değerleri değiştirilmez. Girdi katmanına iletilen bilgiler, herhangi bir değişiklik yapılmadan ilk ara katmana gönderilir. Girdi değerleri, ilk ara katmandaki nöronlarda işlenir ve bir sonraki katmana iletilir. Bilgi, ağın girdi katmanından çıktı katmanına doğru ilerler.

   Ara katmanlarda ve çıktı katmanında yer alan herbir nöron, bir önceki katmandan gelen değerler ile ağırlıkları hesaba katarak net girdiyi denklem 3.3 ile hesaplar:

$$a_j = \sum_i w_{ji}y_i \tag{3.3}$$

   Burada, y_i'ler j. nöronun girdilerini belirtmektedir. Eğer j. nöron ilk gizli katmanda yer alıyorsa, y_i'ler ağın girdi değerleridir. w_{ji} ise, i. nöronu j. nörona bağlayan ağırlık değeridir. Daha öncede ifade edildiği gibi, bias değeri ekstra bir nöron kullanılarak net girdiye eklenebilmektedir. Bu nedenle, bias değerini ayrı bir şekilde incelemeye gerek yoktur. j. nöronun çıktısı ise, bir aktivasyon fonksiyonunda net girdinin değeri hesaplanarak bulunur:

$$y_j = \varphi(a_j). \tag{3.4}$$

   Ağın çıktı katmanında yer alan nöronların da çıktı değerleri hesaplandıktan sonra, geri yayılım algoritmasının ilk aşaması tamamlanır.

2. ***Geriye Doğru Hesaplama***

Bu aşamada, ilk olarak ağın çıktıları ile istenen çıktı değerleri arasındaki fark bulunarak toplam hata hesaplanır. Çıktı katmanındaki herhangi bir yapay nöronda oluşan hata,

$$e_k = t_k - y_k \tag{3.5}$$

şeklinde hesaplanabilir. e_k, k. yapay nöronun hatasını belirtmektedir. Burada, k. nörondan istenen çıktı değeri t_k ile, bu nöronun çıktı değeri ise y_k ile gösterilmektedir. Çıktı katmanındaki bütün işlem birimlerinde oluşan toplam hata ise, şu şekilde ifade edilebilir:

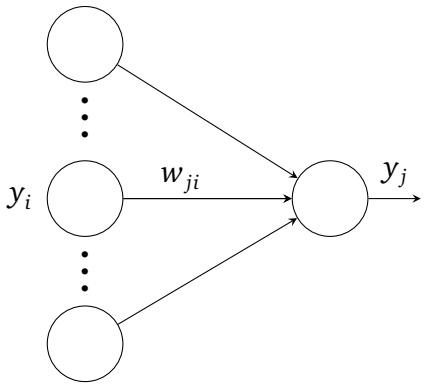
$$E = \frac{1}{2}\sum_k e_k^2 \tag{3.6}$$

Denklem 3.6'ya maliyet fonksiyonu da denmektedir. Maliyet fonksiyonunu hesaplamak için, denklem 3.6'da belirtilen fonksiyondan farklı bir fonksiyon da kullanılabilir. Toplam hatayı en aza indirmek için, hata ağın ağırlık değerlerine dağıtılır ve ağırlık değerleri değiştirilir. Ağırlıklar değiştirilirken, ağırlıkların yapay sinir ağında bulunduğu yere göre, iki farklı durum söz konusu olur:

- Ara katman ile çıktı katmanı arasındaki ağırlıkların değiştirilmesi
- Ara katmanlar arasındaki ağırlıkların veya girdi katmanı ile ara katman arasındaki ağırlıkların değiştirilmesi

i. *Ara Katman ile Çıktı Katmanı Arasındaki Ağırlıkların Değiştirilmesi*

Çıktı katmanından bir önceki katmanda yer alan i. nöron ile çıktı katmanındaki j. nöronu birbirine bağlayan w_{ji} ağırlığını ele alalım (Şekil 3.3).



Şekil 3.3 Çıktı Katmanındaki j. Nöronun Sinaptik Ağırlığını Ayarlamak için Nöral Bağlantıların Gösterimi

Öncelikle, ağırlıkda yapılacak değişimden hata fonksiyonunun ne kadar etkilendiğini bulmak için, $\frac{\partial E}{\partial w_{ji}}$ kısmi türevi hesaplanmalıdır:

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial E}{\partial e_j}\frac{\partial e_j}{\partial y_j}\frac{\partial y_j}{\partial a_j}\frac{\partial a_j}{\partial w_{ji}} \tag{3.7}$$

Çıktı katmanındaki nöronu j notasyonu ile ifade ettiğimiz için, 3.5 ve 3.6 denklemlerini şu şekilde tekrar yazabiliriz:

$$\begin{aligned} E &= \frac{1}{2}\sum_j e_j^2 \\ e_j &= t_j - y_j \end{aligned} \tag{3.8}$$

O halde, $\frac{\partial E}{\partial e_j}$ ve $\frac{\partial e_j}{\partial y_j}$ kısmi türevleri,

$$\begin{aligned} \frac{\partial E}{\partial e_j} &= e_j, \\ \frac{\partial e_j}{\partial y_j} &= -1 \end{aligned} \tag{3.9}$$

şeklindedir.

3.3 ve 3.4 denklemleri kullanılarak, $\frac{\partial y_j}{\partial a_j}$ ve $\frac{\partial a_j}{\partial w_{ji}}$ kısmi türevleride hesaplanabilir:

$$\begin{aligned} \frac{\partial y_j}{\partial a_j} &= \varphi'(a_j), \\ \frac{\partial a_j}{\partial w_{ji}} &= y_i \end{aligned} \tag{3.10}$$

3.9 ve 3.10 denklemlerini, 3.7 denkleminde yerine yazdığımızda;

$$\frac{\partial E}{\partial w_{ji}} = -e_j\varphi'(a_j)y_i \tag{3.11}$$

elde ederiz. Aynı zamanda, 3.11 denklemini şu şekilde yazabiliriz:

$$\frac{\partial E}{\partial w_{ji}} = -\delta_j y_i \; ; \quad \delta_j = \frac{\partial E}{\partial a_j} \tag{3.12}$$

Burada, δ_j sembolü, j. çıktı nöronunun hata değerini belirtmektedir. Ayrıca 3.12 denkleminden görüleceği üzere; toplam hatanın, ara katman ile çıktı

katmanı arasındaki ağırlığa göre türevi, çıktı katmanındaki nöronun δ_j hatası ile o nörona gelen girdinin çarpımına eşittir.

Hata fonksiyonunun ağırlık değişiminden etkilenme miktarı $\frac{\partial E}{\partial w_{ji}}$ hesaplandıktan sonra; Δw_{ji} ağırlık değerinde yapılacak değişim, gradyan inişi algoritması kullanılarak hesaplanabilir:

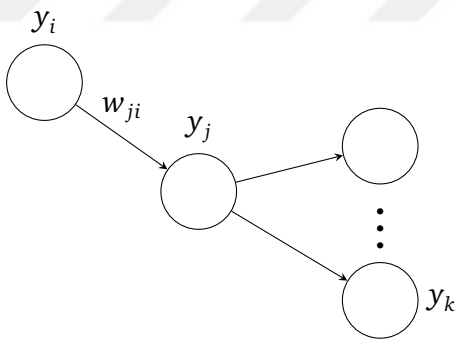
$$\Delta w_{ji} = -\eta \frac{\partial E}{\partial w_{ji}} \tag{3.13}$$

Burada η öğrenme katsayısıdır. Ağırlıkda yapılacak olan değişimin miktarını belirler. Genellikle, 0 ile 1 arasında değerler alır.

ii. *Ara Katmanlar Arasındaki Ağırlıkların veya Girdi Katmanı ile Ara Katman Arasındaki Ağırlıkların Değiştirilmesi*

Gizli katmanda yer alan bir nöron ele alındığında, bu nörondan istenen çıktı değeri ve dolayısıyla bu nöronun hata değeri bilinmemektedir. Geri yayılım algoritmasında; gizli katmanlardaki nöronların hataları, kendinden sonraki katmanlardaki nöronların hatalarına bağlı olarak hesaplanır.

Gizli katmanda yer alan bir j. nöronu ele aldığımızda, bu nörona gelen ağırlıklardan birini w_{ji} sembolü ile gösterebiliriz. Bu bağlantı, i. nöronu j. nörona bağlayan ağırlık değerini belirtmektedir (Şekil 3.4).



Şekil 3.4 Gizli Katmanda Bulunan j. Nöronun Sinaptik Ağırlığını Ayarlamak için Nöral Bağlantıların Gösterimi

Toplam hatanın w_{ji} ağırlığına göre türevini, zincir kuralını kullanarak şu şekilde ifade edebiliriz:

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial E}{\partial y_j} \frac{\partial y_j}{\partial a_j} \frac{\partial a_j}{\partial w_{ji}} \tag{3.14}$$

Burada, j. nöronun çıktı ve net girdi değerleri sırasıyla, y_j ve a_j ile gösterilmektedir.

Toplam hata, daha önce denklem 3.6 ile şu şekilde belirtilmişti:

$$E = \frac{1}{2}\sum_{k} e_k^2 \tag{3.15}$$

O halde $\frac{\partial E}{\partial y_j}$ kısmi türevini şu şekilde yazabiliriz:

$$\frac{\partial E}{\partial y_j} = \sum_{k} e_k \frac{\partial e_k}{\partial y_j} \tag{3.16}$$

Burada, k notasyonu çıktı katmanındaki nöronları belirtmek için kullanılmaktadır. Denklem 3.16'de yer alan $\frac{\partial e_k}{\partial y_j}$ kısmi türevine zincir kuralı uygulanırsa, 3.16 denklemini tekrar şu şekilde yazabiliriz:

$$\frac{\partial E}{\partial y_j} = \sum_{k} e_k \frac{\partial e_k}{\partial a_k}\frac{\partial a_k}{\partial y_j} \tag{3.17}$$

Burada a_k, k. çıktı nöronunun net girdisini belirtmektedir. Denklem 3.5'de belirtildiği gibi, çıktı katmanındaki k. nöronunun hata değeri e_k ise,

$$e_k = t_k - y_k \tag{3.18}$$

şeklindedir.

Ayrıca, 3.3 ve 3.4 denklemleri kullanılarak, k. nöronun çıktısı şu şekilde yazılabilir:

$$\begin{aligned} y_k &= \varphi(a_k) \\ a_k &= \sum_{j=0}^{m} w_{kj} y_j \end{aligned} \tag{3.19}$$

Burada, m sembolü k. nörona gelen toplam girdi sayısını belirtmektedir. O halde 3.18 ve 3.19 denklemleri kullanılarak, $\frac{\partial e_k}{\partial a_k}$ ve $\frac{\partial a_k}{\partial y_j}$ türevleri;

$$\begin{aligned} \frac{\partial e_k}{\partial a_k} &= -\varphi_k'(a_k) \\ \frac{\partial a_k}{\partial y_j} &= w_{kj} \end{aligned} \tag{3.20}$$

şeklinde yazılabilir.

Denklem 3.20'yi, 3.16 denkleminde yerine yazarsak;

$$\frac{\partial E}{\partial y_j} = -\sum_{k} e_k \varphi_k'(a_k) w_{kj} \tag{3.21}$$

elde ederiz.

Daha önce 3.11 ve 3.12 eşitliklerinde belirtildiği gibi, j. çıktı nöronunun hatası δ_j,

$$\delta_j = e_j \varphi_j'(a_j) \tag{3.22}$$

şeklinde gösterilmektedir. Burada, çıktı katmanındaki nöron için k notasyonunu kullandığımız için, 3.21 denklemi şu şekilde yeniden yazılabilir:

$$\frac{\partial E}{\partial y_j} = -\sum_k \delta_k w_{kj}. \tag{3.23}$$

3.14 denklemindeki, $\frac{\partial y_j}{\partial a_j}$ ve $\frac{\partial a_j}{\partial w_{ji}}$ türevleri ise

$$\begin{aligned} \frac{\partial y_j}{\partial a_j} &= \varphi_j'(a_j)\,; \qquad y_j = \varphi(a_j) \\ \frac{\partial a_j}{\partial w_{ji}} &= y_i\,; \qquad a_j = \varphi(\sum_{i=0}^{m} w_{ji} y_i). \end{aligned} \tag{3.24}$$

şeklinde bulunur. Burada y_i, j. gizli nörona gelen girdiyi belirtmektedir.

O halde 3.23 ve 3.24 denklemlerini, 3.14 denkleminde yerine koyarsak;

$$\frac{\partial E}{\partial w_{ji}} = -\Big(\sum_k \delta_k w_{kj}\Big)\varphi_j'(a_j) y_i \tag{3.25}$$

elde ederiz. j. gizli nöronun hatasını belirten δ_j ise şu şekildedir:

$$\delta_j = -\Big(\sum_k \delta_k w_{kj}\Big)\varphi_j'(a_j). \tag{3.26}$$

Denklem 3.26'yı, 3.25 denkleminde yerine koyarsak;

$$\frac{\partial E}{\partial w_{ji}} = -\delta_j y_i \tag{3.27}$$

denklemini elde ederiz.

$\frac{\partial E}{\partial w_{ji}}$ hesaplandıktan sonra, denklem 3.13'deki gibi gradyan inişi algoritması kullanılarak ağırlıkda yapılacak değişimler hesaplanabiir.

3.12 ve 3.27 denklemlerinde görüldüğü üzere; $\frac{\partial E}{\partial w_{ji}}$ toplam hatanın gizli veya çıktı katmanda yer alan bir nöronun ağırlığına göre türevi, o nöronun hatası δ_j ile o nörona gelen girdinin çarpımının eksi işaretlisine eşittir. Ayrıca, bir nöronun

hatası δ_j, o nöronun gizli veya çıktı katmanında yer almasına göre şu şekilde bulunmaktadır:

- Eğer j. nöron çıktı katmanında yer alıyorsa; δ_j, nöronun çıktısının türevi $\varphi'_j(a_j)$ ile o nöronun hata değerinin e_j çarpımına eşittir.
- j. nöron gizli katmanda yer alan bir nöron ise; δ_j, j. nöronun bulunduğu katmandan sonraki katmanda yer alan nöronların hatalarının ağırlıklı toplamı ile $\varphi'_j(a_j)$'nin çarpımıdır.

Denklem 3.26'da görüldüğü gibi, gizli katmandaki bir nöronun hatasını hesaplayabilmek için, o nöronun bulunduğu katmandan bir sonraki katmanda yer alan nöronların hatalarının bilinmesi gerekmektedir. Dolayısıyla çıktı katmanındaki nöronların hataları hesaplandıktan sonra, hatalar ağ üzerinde geriye doğru yayılır ve girdi katmanına kadar olan nöronların hataları sırasıyla hesaplanır. Bu nedenle, algoritmaya geri yayılım algoritması denmektedir.

3.6 ile gösterilen hatayı minimize etmek için, gradyan inişi dışında farklı iteratif metodlar da kullanılabilir. Bu metodlarda, iterasyona başlamak için, ilk olarak başlangıç ağırlık değerleri atanır. Ardından hata minimize olana kadar ağırlık değerleri değiştirilir. Bu çalışmada, bazı gradyan inişi optimizasyon algoritmalarından bahsedilecektir.

3.1.1 Gradyan İnişi

Gradyan inişi algoritması, uygulama kolaylığı nedeniyle, en sık kullanılan tekniklerden biridir. Diferansiyellenebilir $E(\mathbf{w})$ fonksiyonunu minimize etmek için, bu yöntem fonksiyonun $\mathbf{w}$ vektörüne göre gradyanını kullanmaktadır:

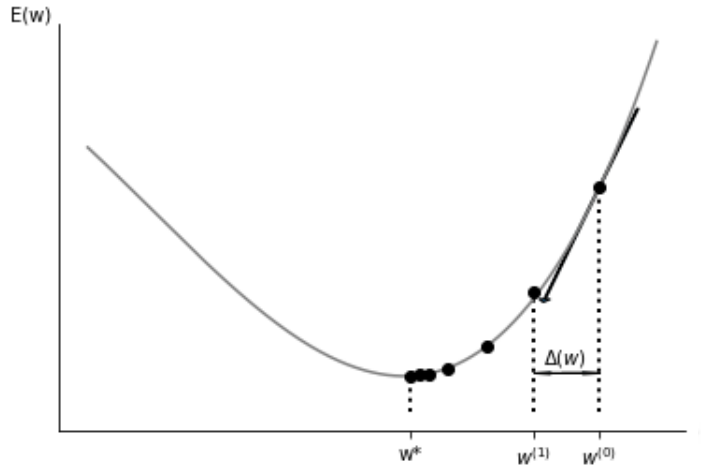
$$\nabla E(\mathbf{w}) = \left[\frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, ..., \frac{\partial E}{\partial w_l} \right]^T$$

Algoritma $w^{(0)}$ başlangıç ağırlık değerinden başlayarak, gradyanın tersi yönde ağırlık değerlerini günceller (Şekil 3.5):

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E(\mathbf{w}^{(\tau)}) \tag{3.28}$$

Burada, τ iterasyonun adım sayısını göstermektedir. Yukarıdaki denklemde kullanılan η öğrenme katsayısının seçimi önemlidir. Öğrenme katsayısı büyük bir değer seçilirse,

algoritma hata fonksiyonunun minimum değerini aşabilir ve bu minimum değere yakınsamayabilir. Eğer öğrenme katsayısı küçük bir değer seçilirse, minimum değere yakınsamak çok uzun vakit alabilir.



Şekil 3.5 Gradyan inişinin geometrik gösterimi

Bir fonksiyonu minimize etmek için, gradyan inişi algoritmasını hesaplarken, ne kadar veri kullandığımıza bağlı olarak üç farklı gradyan inişi çeşidinden bahsedebiliriz: Toplu gradyan inişi (batch gradient descent), stokastik gradyan inişi (stochastic gradient descent) ve mini-toplu gradyan inişi (mini-batch gradient descent).

3.1.1.1 Toplu Gradyan İnişi (Batch Gradient Descent)

Toplu gradyan inişi hesaplanırken, eğitim kümesindeki tüm veriler kullanılır. Bütün veri kümesi için gradyan vektörü hesaplandıktan sonra, ağırlıklar güncellenir:

$$w_{\tau+1} = w_{\tau} - \eta \nabla_{w} E(w_{\tau}) \tag{3.29}$$

Burada, τ iterasyonun adım sayısını göstermektedir. $\nabla_{w} E(w_{\tau})$ ise E fonksiyonunun w parametresine bağlı gradyanını belirtmektedir.

Her seferinde tüm veri için gradyan inişinin hesaplanması, bu algoritmanın çok yavaş ilerlemesine sebep olur. Ayrıca, belleğe sığamayacak büyük veri kümeleri için uygulanması zordur. Fakat her bir güncellemede tüm veri seti incelendiği için, uygun şartlar altında, algoritma hata fonksiyonunun minimum değerine yakınsamayı garanti eder.

3.1.1.2 Stokastik Gradyan İnişi (Stochastic Gradient Descent - SGD)

Stokastik gradyan inişinde, ağa rastgele bir eğitim örneği x_i ve t_i sunulur. Ardından sunulan örnek için gradyan inişi hesaplanarak parametreler (ağırlık ve bias değerleri) güncellenir. Bu işlem tüm eğitim örnekleri için gerçekleştirilir ve örnekler rastgele seçilir.

$$w_{\tau+1} = w_\tau - \eta \nabla_w E(w_\tau; x_i; t_i) \tag{3.30}$$

Parametreler güncellenirken, her yinelemede bir örnek ele alındığı için, stokastik gradyan inişi algoritmasının hesaplanması hızlıdır ve uygulanması kolaydır. Dolayısıyla büyük veri kümeleri söz konusu olduğunda; maliyet fonksiyonunun minimum değerine, stokastik gradyan inişi toplu gradyan inişine göre daha hızlı yakınsar. Fakat, tek bir örnek üzerinden parametrelerde değişiklik yapıldığı için, algoritma parametrelerde yüksek varyanslı güncellemelere neden olur.

3.1.1.3 Mini-Toplu Gradyan İnişi (Mini-Batch Gradient Descent)

Mini-toplu gradyan inişi, diğer iki gradyan inişi yönteminin de iyi özelliklerini almaktadır. Her bir parametre güncellemesini, n tane eğitim örneğinden oluşan eğitim setinin küçük bölümleri için gerçekleştirir:

$$w_{\tau+1} = w_\tau - \eta \nabla_w E(w_\tau; x_{(i:i+n)}; t_{(i:i+n)}) \tag{3.31}$$

Mini-toplu gradyan inişi, toplu gradyan inişine göre daha hızlı, stokastik gradyan inişine göre ise daha kararlıdır.

3.1.2 Gradyan İnişi Optimizasyon Algoritmaları

Gradyan inişi algoritmasının dezavantajlı yönlerini gidermek için, zamanla bu algoritmadan yola çıkarak farklı algoritmalar geliştirilmiştir. Bu bölümde, bir fonksiyonu minimize etmek için, fonksiyonun birinci dereceden türevini içeren bazı algoritmalara yer verilmiştir. İkinci dereceden türev içeren metodlara göre, bu metodlar hesaplama ve uygulama açısından daha kolaydır.

3.1.2.1 Momentum

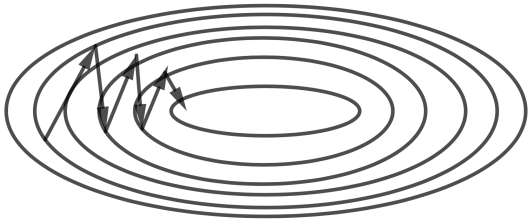
Maliyet fonksiyonunu en aza indirmek için, gradyan inişi algoritması kullanıldığında bazı problemlerle karşılaşılabilmektedir. Momentum algoritması, bu problemlerin

çözümü için gradyan inişine geçmiş gradyan değerlerini de eklemektedir:

$$\begin{aligned} v_{t+1} &= \gamma v_t + \eta \nabla_w E(w) \\ w_{t+1} &= w_t - v_{t+1} \end{aligned} \tag{3.32}$$

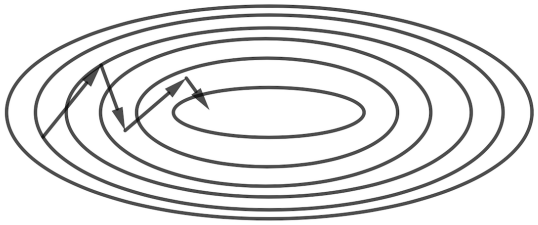
Burada, γ sembolüne momentum sabiti adı verilmektedir. 0 ile 1 arasında değerler almaktadır. Ancak genelikle 0.9 değerini alır. Daha önceki gradyan değerlerinin ne kadarının, işleme katılacağını belirler. İterasyon basamağı sayısı burada t ile gösterilmektedir. v_t momentum değerini belirtmektedir ve başlangıçta 0 olarak alınır.

"Yapay sinir ağlarının farklı katmanlarında yer alan ağırlıkların kısmi türevleri genellikle farklı büyüklüktedir" [32]. Dolayısıyla, gradyan inişinde kısmi türevi büyük olan ağırlıklarda daha fazla değişim olur iken, kısmi türevi küçük olan ağırlıklarda değişim daha azdır. Bu nedenden dolayı, gradyan inişi algoritması, hata fonksiyonunun minimum noktasını ararken çok fazla salınım yapmaktadır (Şekil 3.6).



Şekil 3.6 Momentum kullanılmadan gradyan inişi

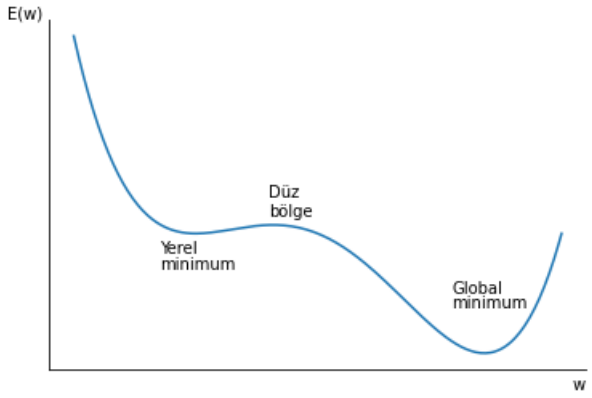
Eğer yapay sinir ağındaki parametrelerin ardışık iterasyonlardaki kısmi türevleri aynı işarete sahip ise, momentum değerleri artar ve böylece bu parametrelerde yapılacak değişiklikler daha büyük olur. Benzer şekilde, parametrelerin ardışık iterasyonlardaki kısmi türevleri farklı işaretlere sahip ise, momentum değeri azalır ve bu paremetrelerde yapılan değişiklikler de azalır. Böylece, momentum algoritması gradyan inişine göre daha hızlı yakınsar ve daha az salınım yapar (Şekil 3.7).



Şekil 3.7 Momentum ile gradyan inişi

Maliyet fonksiyonunun grafiği yerel minimum noktalarına veya düz bölgelere sahip

olabilir (Şekil 3.8). Bu durumda, çoğunlukla gradyan inişi algoritması global minimum noktasına ulaşamamaktadır. Gradyan inişi algoritması, fonksiyonun belli bir noktadaki gradyanını kullandığı için, eyer noktalarından oluşan düz bölgeleri geçmesi uzun sürmektedir. Ayrıca, eğer algoritma yerel minimum noktasına ulaşırsa, global minimum noktasına erişemeyebilir.



Şekil 3.8 Yerel Minimum, Global Minimum ve Düz Bölgeye Sahip Maliyet Fonksiyonunun Gösterimi

Momentum, gradyan inişi algoritmasının düz bölgelerden ve yerel minimum noktalarından geçişini kolaylaştırır. Momentum algoritmasında, geçmiş iterasyonlardaki gradyan değerlerinin de eklenmesi ile, algoritma düz bölgelerde daha hızlı gezinebilmekte ve yerel minimum noktalarından kaçınabilmektedir.

3.1.2.2 Nesterov Hızlandırılmış Gradyan (Nesterov Accelerated Gradient - NAG)

Momentum algoritması geçmiş gradyan vektörlerini ekleyerek, gradyan inişine göre daha büyük adımlar atmaktadır. Ancak momentum algoritmasının bu özelliği, global minimum noktasını geçmesine sebep olabilir. Ayrıca gradyan inişine göre daha az salınım yapmasına rağmen, momentum algoritması da salınım yapmaktadır. Momentum algoritmasının bu eksikliklerini gidermek için; Nesterov hızlandırılmış gradyan algoritması [33], parametrelerin gelecekteki konumunun gradyanını hesablamaktadır:

$$\begin{aligned} v_{t+1} &= \gamma v_t + \eta \nabla_w E(w - \gamma v_t) \\ w_{t+1} &= w_t - v_{t+1} \end{aligned} \tag{3.33}$$

Denklem 3.33 'de v_{t+1} terimini hesaplarken, mevcut konumun gradyanı değil bir sonraki konumun gradyanı dikkate alınmaktadır. Burada da, γ momentum sabiti 0.9 olarak alınmaktadır.

Momentum algoritmasında, önce mevcut konumun gradyanı hesaplanır. Ardından önceden hesaplanmış tüm gradyanların toplamı yönünde, parametrelerde bir güncelleme yapılır. Nesterov hızlandırılmış gradyan algoritmasında ise, önce geçmiş gradyanların toplamı yönünde bir ilerleme kaydedilir, ardından gelinen konumdaki gradyan hesaplanarak parametrelerde güncelleme yapılır. Algoritmanın gelecekteki konumunun gradyanını dikkate alması, algoritmanın çok hızlı ilerlemesini engeller.

3.1.2.3 Adagrad

Gradyan inişi ve momentum algoritmalarında, öğrenme katsayısı sabittir. Adagrad [34] algoritmasında ise, öğrenme katsayısı her bir parametre ve her bir iterasyon adımı için değiştirilmektedir.

Ağırlık vektöründeki her bir ağırlık değerini w_i ile gösterirsek, t. iterasyon adımında ki hata fonksiyonunun gradyanı $g_{t,i}$ şu şekilde ifade edilebilir:

$$g_{t,i} = \nabla_{w_t} E(w_{t,i}) \tag{3.34}$$

Adagrad algoritması her bir parametre için öğrenme katsayısını belirlerken, geçmiş gradyanları da hesaba katmaktadır. Adagrad algoritmasının parametreleri güncelleme kuralı şu şekildedir:

$$w_{t+1,i} = w_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} g_{t,i} \tag{3.35}$$

Burada, G_t köşegen matrisidir. $G_{t,ii}$ bu matrisin (i,i) köşegen elemanıdır ve w_i parametresine göre, t. adıma kadar olan geçmiş gradyanların karelerinin toplamına eşittir. ϵ ise kesirli ifadenin 0 ile bölünmesini engellemek için atanan bir sabittir ve genellikle 10^{-8} alınır.

Adagrad algoritması öğrenme katsayısını kendi belirlediği için, önceki algoritmalara göre daha avantajlıdır. Fakat geçmiş gradyanların karelerini topladığı için, paydanın değeri sürekli artar ve bu nedenle, öğrenme katsayısının değeri giderek azalır. Öğrenme katsayısı çok küçüldüğünde, parametrelerde yapılan güncellemeler de azalır.

3.1.2.4 Adadelta

Adagrad'ın sahip olduğu öğrenme katsayısı problemini çözmek için, Adadelta algoritması [35] geliştirilmiştir. "Önceki gradyanların karelerini verimsiz bir şekilde

depolamak yerine; gradyanların toplamı, tüm geçmiş gradyanların karelerinin azalan bir ortalaması olarak, özyinelemeli bir şekilde tanımlanır" [36].

Adadelta algoritmasının parametreleri güncelleme kuralı şu şekildedir:

$$\begin{aligned} \Delta w_t &= -\frac{RMS[\Delta w]_{t-1}}{RMS[g]_t} g_t \\ w_{t+1} &= w_t + \Delta w_t \end{aligned} \tag{3.36}$$

Denklem 3.36 'da g_t, t. iterasyon basamağındaki gradyan değerini belirtmektedir. $RMS[g]_t$ ise t. adıma kadar olan gradyanların karelerinin ortalama karekök hatasını hesaplamaktadır:

$$RMS[g]_t = \sqrt{E[g^2]_t + \epsilon} \tag{3.37}$$

ϵ değeri, Adagrad'da olduğu gibi paydanın sıfıra eşit olmasını engelleyen sabittir. $E[g^2]_t$ ise mevcut konumdaki gradyanın karesinin ve geçmiş gradyanların karelerinin üssel olarak azalan bir ortalaması olmak üzere, şu şekilde tanımlanır:

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1-\gamma) g_t^2 \tag{3.38}$$

Burada γ sabiti, momentum algoritmasındaki momentum sabitine benzerdir ve genellikle 0.9 alınır.

Benzer şekilde, $RMS[\Delta w]_t$ ve $E[\Delta w^2]_t$ ifadeleri sırasıyla, parametre değişimlerinin karelerinin ortalama karekök hatasını ve parametre değişimlerinin karelerinin üssel olarak azalan ortalamasını belirtir.

$$RMS[\Delta w]_t = \sqrt{E[\Delta w^2]_t + \epsilon} \tag{3.39}$$

$$E[\Delta w^2]_t = \gamma E[\Delta w^2]_{t-1} + (1-\gamma) \Delta w_t^2 \tag{3.40}$$

3.1.2.5 RMSprop

RMSprop algoritması, Adadelta gibi, Adagrad algoritmasının sürekli azalan öğrenme katsayısı sorununu çözmek için geliştirilmiştir. Bu algoritma, Adadelta algoritması ile

aynı zaman aralığında, Geoff Hinton [37] tarafından ortaya konulmuştur:

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1-\gamma) g_t^2$$
$$w_{t+1} = w_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t \tag{3.41}$$

γ sabitinin 0.9, η öğrenme katsayısının da 0.001 olarak alınması Hinton [37] tarafından önerilmiştir.

3.1.2.6 Adaptif Moment Tahmin (Adaptive Moment Estimation - Adam)

Adaptif moment tahmin algoritması [38], RMSprop ve momentum algoritmalarının özelliklerini taşır. Adam algoritması, geçmiş gradyanların üssel olarak azalan m_t ortalamasını ve geçmiş gradyanların karelerinin üssel olarak azalan v_t ortalamasını kullanır.

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$$
$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \tag{3.42}$$

Burada, m_t gradyanların birinci momentini, v_t ise gradyanların ikinci momentini belirtmektedir. Başlangıçta m_t ve v_t momentleri sıfır vektörleri olarak alınırlar. Momentlere başlangıçta 0 değeri atandığı için, iterasyonun başlarında momentler sıfıra yakın değerler alır. Aynı zamanda β_1 ile β_2, 1'e yakın değerler aldığında da, momentler sıfıra meyilli olurlar. Bu nedenlerden dolayı, momentler normalize edilir:

$$\hat{m_t} = \frac{m_t}{1-\beta_1^t}$$
$$\hat{v_t} = \frac{v_t}{1-\beta_2^t} \tag{3.43}$$

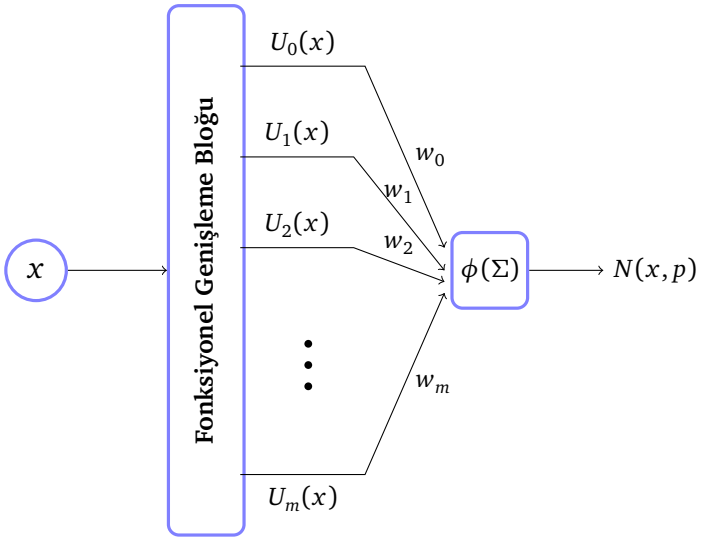
$\hat{m_t}$ ve $\hat{v_t}$ değişkenlerini hesapladıktan sonra, parametreler denklem 3.44 kullanılarak güncellenir.

$$w_{t+1} = w_t - \frac{\eta}{\sqrt{\hat{v_t}} + \epsilon} \hat{m_t} \tag{3.44}$$

Yazar [38], β_1 ve β_2 değerlerinin sırasıyla, 0.9 ve 0.999 olarak alınmasını önermiştir. Ayrıca ϵ, diğer algoritmalarda olduğu gibi 10^{-8} olarak alınır.

4
TEK KATMANLI FONKSİYONEL BAĞLANTILI YAPAY SİNİR AĞI (SINGLE-LAYER FUNCTIONAL LINK ARTIFICIAL NEURAL NETWORK)

Tek katmanlı fonksiyonel bağlantılı yapay sinir ağı (FLANN), gizli katmanı olmayan bir sinir ağı çeşididir. FLANN modeli, Pao ve Philips [39] tarafından tanıtılmıştır. Fonksiyonel bağlantılı yapay sinir ağı bir girdi nöronu, bir çıktı nöronu ve gizli katman yerine bir fonksiyonel genişleme bloğundan oluşmaktadır (Şekil 4.1). Ağa sunulan girdiler, ortogonal, trigonometrik, üstel gibi polinomlar kullanılarak genişletilir. Ardından, fonksiyonel genişleme bloğu tarafından üretilen genişletilmiş girdiler, ağırlık değerleri ile çarpılır. Elde edilen çarpımlar toplanır ve bu toplamın aktivasyon fonksiyonu değeri hesaplanarak ağın çıktı değeri elde edilir.



Şekil 4.1 Tek Katmanlı Fonksiyonel Bağlantılı Yapay Sinir Ağı Modeli

$x = (x1, x2, ..., xh)^T$ girdi vektörü, $j = \{0, 1, 2, \ldots, m\}$ için U_j ve w_j, sırasıyla, genişletilmiş girdi verileri ve ağırlık değerleri olmak üzere, ağın çıktısını şu şekilde yazabiliriz:

$$z = \sum_{j=0}^{m} w_j U_j(x)$$
$$N(x,p) = \phi(z) \tag{4.1}$$

Burada, ϕ aktivasyon fonksiyonunu belirtmektedir. Yapay sinir ağının ağırlık değerleri ise, $p = [w_0, w_1, w_2, \ldots, w_m]$ vektörü ile gösterilmiştir.

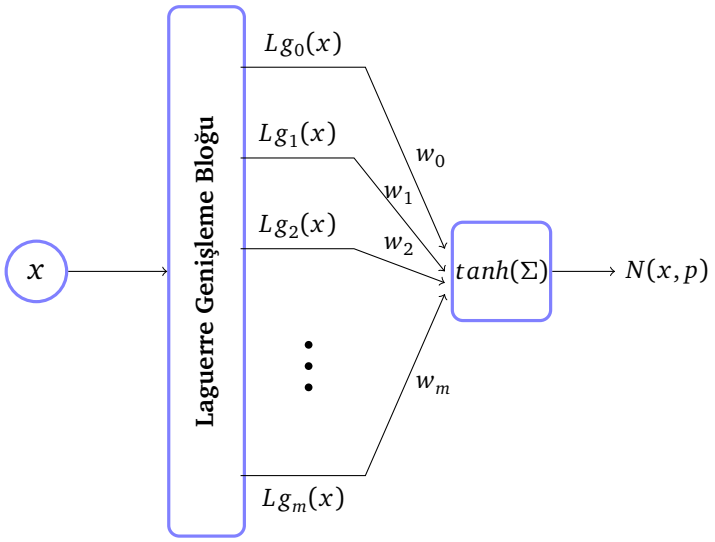
Tek katmanlı fonksiyonel bağlantılı sinir ağının parametre sayısı, çok katmanlı algılayıcının parametre sayısına göre daha azdır. Bu nedenle, FLANN daha hızlı eğitilir ve işlem kolaylığı sağlar. Dolayısıyla, uygulanması çok katmanlı algılayıcıya göre daha kolaydır.

4.1 Tek Katmanlı Fonksiyonel Bağlantılı Yapay Sinir Ağı Modelleri

Bu tez çalışmasında, Laguerre, Chebyshev ve Legendre yapay sinir ağları ele alınacaktır.

4.1.1 Laguerre Yapay Sinir Ağı (Laguerre Neural Network - LgNN)

Laguerre yapay sinir ağı modeli, bir girdi nöronu, bir çıktı nöronu ve Laguerre ortogonal polinomlarını kullanan bir fonksiyonel genişleme bloğu içerir (Şekil 4.2). Burada Laguerre polinomları, girdi verilerini daha yüksek boyutlu bir uzaya dönüştürür.



Şekil 4.2 Tek Katmanlı Laguerre Yapay Sinir Ağı Modeli

Laguerre polinomları, denklem 4.2 ile verilen Laguerre diferansiyel denklemini sağlayan ortogonal polinomlardır.

$$xy''(x) + (1-x)y'(x) + ny(x) = 0 \tag{4.2}$$

Aşagıda verilen verilen rekürans bağıntısı ile yüksek mertebeden Laguerre polinomları elde edilebilir.

$$\begin{aligned} &Lg_0(x) = 1 \\ &Lg_1(x) = -x + 1 \\ &Lg_{n+1}(x) = (2n+1-x)Lg_n(x) - n^2 Lg_{n-1}(x), \quad (n \geq 1). \end{aligned} \tag{4.3}$$

Şekil 4.2 'de, ilk m Laguerre polinomunu kullanan bir Laguerre yapay sinir ağı gösterilmiştir. Burada, girdi olarak h boyutlu $x = (x1, x2, ..., xh)^T$ girdi vektörü ele alınırsa, Laguerre polinomları kullanılarak elde edilen genişletilmiş girdi değerleri şu şekilde yazılabilir:

$$\begin{aligned} \Big[Lg_0(x_1), Lg_1(x_1), Lg_2(x_1), \ldots, Lg_m(x_1); \quad Lg_0(x_2), Lg_1(x_2), Lg_2(x_2), \ldots, Lg_m(x_2); \\ \ldots; Lg_0(x_h), Lg_1(x_h), Lg_2(x_h), \ldots, Lg_m(x_h) \Big] \end{aligned} \tag{4.4}$$

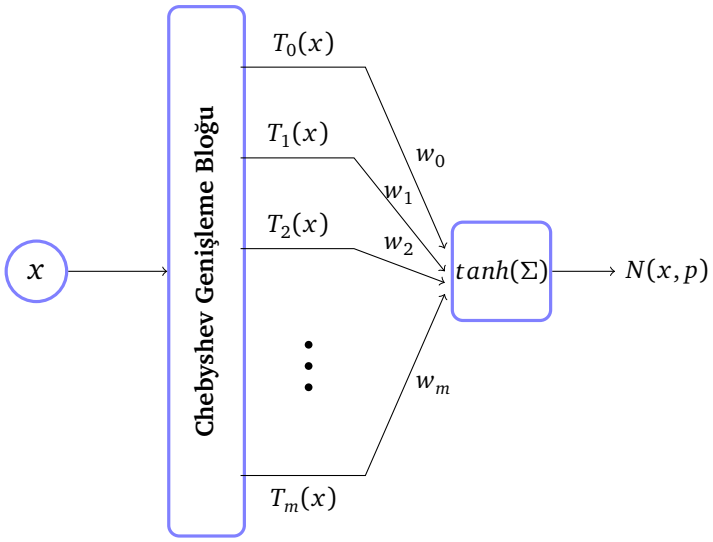
Laguerre yapay sinir ağı modelinde, $tanh$ aktivasyon fonksiyonu kullanılmıştır. x girdi vektörü ve $p = [w_0, w_1, w_2, \ldots, w_m]$ ağırlık vektörüne göre sinir ağının çıktısı şu şekilde hesaplanabilir:

$$N(x, p) = tanh\Big(\sum_{j=0}^{m} w_j Lg_j(x)\Big). \tag{4.5}$$

4.1.2 Chebyshev Yapay Sinir Ağı (Chebyshev Neural Network - ChNN)

Tek katmanlı Chebyshev yapay sinir ağı modeli, Şekil 4.3 ile gösterilmektedir. Bir girdi nöronu, bir çıktı nöronu ve gizli katman yerine fonksiyonel genişleme bloğundan oluşmaktadır. Bu genişleme bloğunda Chebyshev polinomları kullanılmaktadır.

Bu yapay sinir ağı modelinde, Chebyshev polinomları kullanılarak genişletilmiş girdi



Şekil 4.3 Tek Katmanlı Chebyshev Yapay Sinir Ağı Modeli

değerleri elde edilir. Yapay sinir ağına girdi vektörü olarak da h boyutlu $x = (x1, x2, ..., xh)^T$ vektörü sunulmuştur.

İlk iki Chebyshev polinomu şu şekildedir:

$$\begin{aligned} T_0(x) &= 1 \\ T_1(x) &= x. \end{aligned} \tag{4.6}$$

Diğer Chebyshev polinomları ise, aşağıda verilen rekürans formülü yardımıyla elde edilebilir:

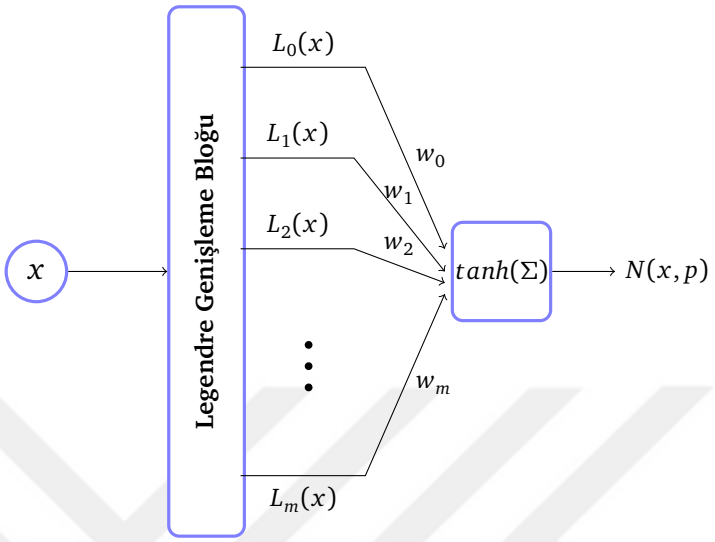
$$T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x). \tag{4.7}$$

İlk m Chebyshev polinomu ile m boyutlu $p = [w_0, w_1, w_2, \ldots, w_m]$ ağırlık vektörü kullanılarak elde edilen, Chebyshev yapay sinir ağı modelinin çıktısını şu şekilde yazabiliriz:

$$N(x,p) = tanh\Big(\sum_{j=0}^{m} w_j T_j(x)\Big) \tag{4.8}$$

4.1.3 Legendre Yapay Sinir Ağı (Legendre Neural Network - LeNN)

Tek katmanlı Legendre yapay sinir ağı modeli, Legendre polinomlarını kullanarak girdi hücrelerini daha yüksek boyutlu bir uzaya dönüştürür (Şekil 4.4). Böylece, gizli katmana ve girdi katmanını gizli katmana bağlayan ağırlık değerlerine ihtiyaç kalmaz.



Şekil 4.4 Tek Katmanlı Legendre Yapay Sinir Ağı Modeli

Legendre diferansiyel denkleminin bir çözümü olan ortogonal Legendre polinomları, aşağıda verilen rekürans bağıntısı ile elde edilir:

$$\begin{aligned} L_0(x) &= 1 \\ L_1(x) &= x \\ L_{n+1}(x) &= \frac{1}{n+1}[(2n+1)xL_n(x) - nL_{n-1}(x)]. \end{aligned} \tag{4.9}$$

h boyutlu $x = (x1, x2, ..., xh)^T$ girdi vektörü yapay sinir ağına sunulursa, Legendre polinomları kullanılarak oluşturulan genişletilmiş girdi verileri şu şekilde yazılabilir:

$$\begin{aligned} \Big[L_0(x_1), L_1(x_1), L_2(x_1), \ldots, L_m(x_1); \quad & L_0(x_2), L_1(x_2), L_2(x_2), \ldots, L_m(x_2); \\ & \ldots; L_0(x_h), L_1(x_h), L_2(x_h), \ldots, L_m(x_h)\Big] \end{aligned} \tag{4.10}$$

x girdi değerine ve parametrelere bağlı Legendre yapay sinir ağı modelinin çıktısı, $tanh$ aktivasyon fonksiyonu kullanılarak şu şekilde hesaplanır:

$$N(x,p) = tanh\Big(\sum_{j=0}^{m} w_j L_j(x)\Big). \tag{4.11}$$

5
İLERİ BESLEMELİ YAPAY SİNİR AĞLARI İLE ADİ DİFERANSİYEL DENKLEMLERİN ÇÖZÜMÜ

Adi ve kısmi diferansiyel denklemleri çözmek için, farklı yapay sinir ağı modelleri kullanılmaktadır. İleri beslemeli yapay sinir ağları, bu alanda en çok kullanılan ağlardan biridir. Bu bölümde, ileri beslemeli yapay sinir ağları ile adi diferansiyel denklemlerin çözümünden bahsedilecektir.

5.1 İleri Beslemeli Yapay Sinir Ağları ile Adi Diferansiyel Denklemlerin Genel Çözümü

Diferansiyel denklemlerin nümerik çözümlerini elde etmek için birçok farklı yöntem geliştirilmiştir. Lagaris vd. [4], adi ve kısmi diferansiyel denklemleri çözmek için, ileri beslemeli yapay sinir ağının kullanıldığı bir yöntem öne sürmüşlerdir. Bu yöntemde, başlangıç ve sınır değer problemlerini çözmek için, yapay sinir ağı içeren bir deneme çözümü kullanılmaktadır. İleri beslemeli yapay sinir ağları fonksiyon yaklaşımı amacıyla da kullanıldığı için, ağı içeren deneme çözümü gerçek çözüme yakınsayabilmektedir. Bu deneme çözümü differansiyellenebilir ve kapalı analitik bir formdadır. Çözüm iki terimin toplamından oluşmaktadır; ilk terim başlangıç/sınır değer koşullarından oluşmaktadır. İkinci terim ise ileri beslemeli yapay sinir ağını içermektedir. Deneme çözümünü gerçek çözüme yaklaştırmak için, öğrenme algoritmaları kullanılarak ağ eğitilmektedir.

Söz edilen yöntemi açıklamak için, belirli sınır koşullarına sahip genel bir diferansiyel denklem ele alalım:

$$F\big(\vec{x}, y(\vec{x}), \nabla y(\vec{x}), \nabla^2 y(\vec{x}), \ldots, \nabla^n y(\vec{x})\big) = 0, \quad \vec{x} \in D \tag{5.1}$$

Burada, $\vec{x} = (x_1, x_2, ..., x_n) \in \mathbb{R}^n$ ve $y(\vec{x})$ denklemin çözümünü belirtmektedir. $D \subset \mathbb{R}^n$ tanım kümesi, ∇ ise diferansiyel operatördür.

D tanım bölgesi ve S bu bölgenin sınırı olmak üzere, tanım bolgesinin ve sınırının ayrıklaştırılmış hali $\hat{D}$ ve $\hat{S}$ ile gösterilmektedir. O halde, denklem 5.1 şu formda yazılabilir:

$$F\big(\vec{x}_i, y(\vec{x}_i), \nabla y(\vec{x}_i), \nabla^2 y(\vec{x}_i), \ldots, \nabla^n y(\vec{x}_i)\big) = 0, \quad \forall \vec{x}_i \in \hat{D} \tag{5.2}$$

Bu denklemin çözümünde kullanılacak deneme çözümü, $y_t(\vec{x}, \vec{p})$ ile gösterilmektedir. Burada, $\vec{p}$ vektörü yapay sinir ağının belirlenmesi gereken parametrelerinden oluşan bir vektördür.

Deneme çözümünü denklem 5.2'de yerine koyarsak, sınır koşullarını da dikkate aldığımızda, denklem 5.2 şu şekilde bir optimizasyon problemine dönüşmektedir:

$$\min_{\vec{p}} \sum_{x_i \in \tilde{D}} F\big((\vec{x}_i, y_t(\vec{x}_i, \vec{p}), \nabla y_t(\vec{x}_i, \vec{p}), \nabla^2 y_t(\vec{x}_i, \vec{p}), \ldots, \nabla^n y_t(\vec{x}_i, \vec{p})\big)^2 \tag{5.3}$$

Deneme çözümü, iki terimin toplamı olarak şu şekilde ifade edilebilir:

$$y_t(\vec{x}, \vec{p}) = A(\vec{x}) + f(\vec{x}, N(\vec{x}, \vec{p})) \tag{5.4}$$

Burada, $A(\vec{x})$ başlangıç/sınır değer koşullarını belirtir. Optimize edilmesi gereken herhangi bir parametre içermez. İkinci terim f ise, sınır koşullarına etki etmeyecek şekilde, ileri beslemeli yapay sinir ağının $N(\vec{x}, \vec{p})$ çıktısını içerir. $\vec{p}$ vektörü ağırlık ve bias değerlerinden oluşmaktadır.

Problem 5.3'ü minimize etmek için, her bir $\vec{x}_i$ vektörüne karşılık gelen hatanın, yani $F(\vec{x}_i)$'nin sıfır olması gerekmektedir. Bunun için, yapay sinir ağı eğitilmelidir. Ağı eğitmek için farklı algoritmalar kullanılabilir. Örneğin, geri yayılım algoritması bu amaçla en sık kullanılan algoritmalardan biridir.

5.1.1 Gradyan Hesaplama

$N(\vec{x}, \vec{p})$'yi içeren deneme çözümünü diferansiyel denklemde yerine koymak için, deneme çözümünün x'e göre türevleri gerekmektedir. Bu nedenle ağın çıktısının x'e göre türevleri bilinmelidir. Ayrıca deneme çözümünü denklemde yerine koyduktan sonra, 5.3 probleminin minimize edilmesi gerekmektedir. Bu amaçla kullanılacak birçok algoritma için, deneme çözümünün ve türevlerinin, ağırlık ve bias değerlerine göre türevleri gerekmektedir.

Gradyan hesaplamalarını göstermek için, n girdi birimine, bir gizli katmana ve bir çıktı birimine sahip bir çok katmanlı algılayıcı ele alalım. Gizli katmandaki yapay nöron sayısını H ve çıktı biriminde kullanılan aktivasyon fonksiyonunu doğrusal fonksiyon alalım. Girdi vektörü $\vec{x} = (x_1, x_2, \ldots, x_n)$ olmak üzere, çok katmalı algılayıcının çıktısı şu şekilde yazılabilir:

$$N = \sum_{i=1}^{H} v_i \sigma(zi) \tag{5.5}$$

Burada, $\sigma(z_i), \quad i = 1, 2, \ldots, H$ gizli katmanda yer alan nöronların çıktılarını, σ ise gizli katmanda kullanılan sigmoid aktivasyon fonksiyonunu belirtmektedir. Ayrıca, gizli katmanı çıktı katmanına bağlayan ağırlık değerleri de v_i ile gösterilmiştir. Gizli katmandaki nöronların çıktıları ise şu şekilde yazılabilir:

$$\sigma(z_i) = \sigma\Big(\sum_{j=1}^{n} w_{ij} x_j + u_i\Big) \quad i = 1, 2, \ldots, H \tag{5.6}$$

Yukarıdaki denklemde, w_{ij}, girdi katmanını çıktı katmanına bağlayan ağırlık değerlerini, u_i ise gizli katmandaki nöronların bias değerlerini göstermektedir.

İlk olarak, ağın çıktısının girdi değerlerine göre türevlerini hesaplayalım. Yapay sinir ağı çıktısının, x_j girdisine göre türevi;

$$\frac{\partial N}{\partial x_j} = \sum_{i=1}^{H} v_i w_{ij} \sigma_i^{(1)} \tag{5.7}$$

şeklindedir. Denklem 5.7'de yer alan, $\sigma_i = \sigma(z_i)$ dir. $\sigma^{(1)}$ sembolü ise, aktivasyon fonksiyonunun birinci mertebeden türevini belirtmektedir.

Benzer şekilde, ağın çıktısının x_j girdi vektörüne göre k. mertebeden türevi ise şu şekilde gösterilir:

$$\frac{\partial^k N}{\partial x_j^k} = \sum_{i=1}^{H} v_i w_{ij}^k \sigma_i^{(k)} \tag{5.8}$$

Yukarıdaki denklemde $\sigma^{(k)}$, aktivasyon fonksiyonunun k. mertebeden türevini belirtmektedir. w_{ij}^k ise, w_{ij} ağırlık değerinin k. kuvvetini göstermektedir.

Genel olarak, ağın çıktısının herhangi bir girdiye göre, herhangi bir mertebeden türevi de aynı şekilde ifade edilebilir:

$$\frac{\partial^{\lambda_1}}{\partial x_1^{\lambda_1}}\frac{\partial^{\lambda_2}}{\partial x_2^{\lambda_2}}\cdots\frac{\partial^{\lambda_n}}{\partial x_n^{\lambda_n}}N=\sum_{i=1}^{n}v_iP_i\sigma_i^{(\Lambda)}=N_g(\vec{x}) \tag{5.9}$$

Burada;

$$P_i=\prod_{k=1}^{n}w_{ik}^{\lambda_k}\quad , \qquad \Lambda=\sum_{i=1}^{n}\lambda_i \qquad dir. \tag{5.10}$$

Ağın herhangi girdilere göre türevi alındığında, sonucun tek gizli katmana sahip ileri beslemeli bir ağın çıktısına $Ng(\vec{x})$ eşit olduğu denklem 5.9 ile görülmektedir. Bu ağ, ilk başta ele aldığımız, ağın parametrelerini içermektedir. Burada, ilk başta ele aldığımız ağın v_i parametresinin yerini v_iP_i almıştır. Ayrıca gizli katmandaki nöronların aktivasyon fonksiyonu olarak, Λ. mertebeden sigmoid aktivasyon fonksiyonu alınmıştır.

Ağın çıktısının girdi değerlerine göre türevleri hesaplandıktan sonra, N_g'nin ağın ağırlık ve bias değerlerine göre gradyanı;

$$\begin{aligned}
&\frac{\partial N_g}{\partial v_i}=P_i\sigma_i^{(\Lambda)}\\
&\frac{\partial N_g}{\partial u_i}=v_iP_i\sigma_i^{(\Lambda+1)}\\
&\frac{\partial N_g}{\partial w_{ij}}=x_jv_iP_i\sigma_i^{(\Lambda+1)}+v_i\lambda_jw_{ij}^{\lambda_j-1}\left(\prod_{k=1,k\neq j}w_{ik}^{\lambda_k}\right)\sigma_i^{(\Lambda)}
\end{aligned} \tag{5.11}$$

şeklinde yazılabilir.

Hatanın ağırlık ve bias değerlerine göre gradyanı belirlendikten sonra, 5.3 ifadesini optimize etmek için geri yayılım algoritması veya farklı minimizasyon yöntemleri kullanılabilir.

5.2 Birinci Mertebeden Diferansiyel Denklemlerin Çözümü

Tanım 5.1. Birinci mertebeden diferansiyel denklem için başlangıç değer problemi

$$\begin{aligned} &\frac{dy}{dx} = f(x,y), \quad x \in [a,b] \\ &y(a) = A. \end{aligned} \tag{5.12}$$

şeklinde tanımlanabilir.

5.12 probleminin iki kısımdan oluşan deneme çözümü şu şekilde yazılabilir:

$$y_t(x,p) = A + (x-a)N(x,p) \tag{5.13}$$

5.13 eşitliğinde yer alan $N(x,p)$, ileri beslemeli yapay sinir ağının çıktısıdır. Burada, $x = (x_1, x_2, \ldots, x_n)$ girdi değerini, p ise ağırlık değerlerinden (ağın bias değerleri de ağırlık vektörüne dahil edilmiştir) oluşan ağırlık vektörünü belirtmektedir.

Deneme çözümünü 5.12 denkleminde yerine koyabilmek için, 5.13 deneme çözümünün x'e göre birinci mertebeden türevi

$$\frac{dy_t(x,p)}{dx} = N(x,p) + (x-a)\frac{dN}{dx} \tag{5.14}$$

şeklinde hesaplanır. Daha sonra, deneme çözümünü ve türevini denklem 5.12'de yerine koyduğumuzda, hata fonksiyonunu şu şekilde yazabiliriz:

$$E(x,p) = \frac{1}{2}\sum_i \left\{ \frac{dy_t(x_i,p)}{dx} - f\Big(x_i, y_t(x_i,p)\Big) \right\}^2 \tag{5.15}$$

Denklem 5.15'de yer alan x_i değerleri, x'in tanımlı olduğu $[a,b]$ aralığındaki noktalardır.

5.15 denklemi ile verilen hata fonksiyonu elde edildikten sonra, bu fonksiyonu minimize etmek için farklı algoritmalar kullanılabilir. Burada, gradyan inişi algoritmasının uygulanışı gösterilecektir. Tek gizli katmana sahip ileri beslemeli bir yapay sinir ağı düşünelim. Girdi katmanını gizli katmana bağlayan ağırlıklar w_j ile, gizli katmanı çıktı katmanına bağlayan ağırlıklar ise v_l ile gösterilsin. O halde, ağırlık değerleri şu şekilde güncellenir:

$$\frac{\partial E(x,p)}{\partial w_j} = \frac{\partial}{\partial w_j}\left(\sum_i \left\{\frac{dy_t(x_i,p)}{dx} - f\Big(x_i, y_t(x_i,p)\Big)\right\}^2\right) \tag{5.16}$$

$$\frac{\partial E(x,p)}{\partial v_l} = \frac{\partial}{\partial v_l}\left(\sum_i \left\{\frac{dy_t(x_i,p)}{dx} - f\Big(x_i, y_t(x_i,p)\Big)\right\}^2\right) \tag{5.17}$$

$$w_j^{k+1} = w_j^k - \eta \frac{\partial E(x,p)}{\partial w_j^k} \tag{5.18}$$

$$v_l^{k+1} = v_l^k - \eta \frac{\partial E(x,p)}{\partial v_l^k} \tag{5.19}$$

Burada, k iterasyon sayısını belirtmektedir. Hata fonksiyonu minimize edilene kadar, ağırlık değerleri her iterasyon adımında güncellenir.

5.3 İkinci Mertebeden Diferansiyel Denklemlerin Çözümü

Tanım 5.2. İkinci mertebeden diferansiyel denklem için başlangıç değer problemi

$$\begin{aligned} &\frac{d^2y}{dx^2} = f(x, y, \frac{dy}{dx}), \quad x \in [a,b] \\ &y(a) = A, \quad y^{'}(a) = B. \end{aligned} \tag{5.20}$$

şeklindedir.

Tanım 5.20 ile verilen denklemin deneme çözümü;

$$y_t(x,p) = A + B(x-a) + (x-a)^2 N(x,p) \tag{5.21}$$

şeklindedir. x girdi değeri ve p ağırlık vektörü olmak üzere, $N(x,p)$ yapay sinir ağının çıktısını belirtmektedir.

Deneme çözümünün x'e göre birinci ve ikinci mertebeden türevleri;

$$\frac{dy_t(x,p)}{dx} = B + 2(x-a)N(x,p) + (x-a)^2 \frac{dN}{dx} \tag{5.22}$$

$$\frac{d^2y_t(x,p)}{dx^2} = 2N(x,p) + 4(x-a)\frac{dN}{dx} + (x-a)^2 \frac{d^2N}{dx^2} \tag{5.23}$$

şeklindedir. 5.22, 5.23 denklemleri ve deneme çözümü, 5.20 ikinci mertebeden başlangıç değer probleminde yerine yazıldıktan sonra, hata fonksiyonu şu şekilde yazılabilir:

$$E(x,p) = \frac{1}{2}\sum_i \left\{ \frac{d^2 y_t(x_i,p)}{dx^2} - f\left(x_i, y_t(x_i,p), \frac{dy_t(x_i,p)}{dx}\right)\right\}^2 \tag{5.24}$$

Hata fonksiyonu belirlendikten sonra, yapay sinir ağının parametreleri güncellenir. Bu güncelleme hata fonksiyonu minimize edilene kadar tekrarlanır.

Tanım 5.3. İkinci mertebeden diferansiyel denklem için Dirichlet sınır değer problemi

$$\begin{aligned} &\frac{d^2 y}{dx^2} = f(x, y, \frac{dy}{dx}), \quad x \in [a,b] \\ &y(a) = A, \quad y(b) = B. \end{aligned} \tag{5.25}$$

olarak tanımlanır.

İkinci mertebeden Dirichlet sınır değer probleminin deneme çözümü şu şekilde yazılabilir:

$$y_t(x,p) = \frac{bA - aB}{b-a} + \frac{B-A}{b-a}x + (x-a)(x-b)N(x,p) \tag{5.26}$$

Deneme çözümünü, 5.25 denkleminde yerine koyabilmek için, deneme çözümünün x'e göre türevleri alınır:

$$\frac{dy_t(x,p)}{dx} = \frac{B-A}{b-a} + (x-b)N(x,p) + (x-a)N(x,p) + (x-a)(x-b)\frac{dN}{dx} \tag{5.27}$$

$$\frac{d^2 y_t(x,p)}{dx^2} = 2N(x,p) + 2(x-a)\frac{dN}{dx} + 2(x-b)\frac{dN}{dx} + (x-a)(x-b)\frac{d^2 N}{dx^2} \tag{5.28}$$

Deneme çözümünün x'e göre türevleri 5.25 sınır değer probleminde yerine koyulduktan sonra, hata fonksiyonu

$$E(x,p) = \frac{1}{2}\sum_i \left\{ \frac{d^2 y_t(x_i,p)}{dx^2} - f\left(x_i, y_t(x_i,p), \frac{dy_t(x_i,p)}{dx}\right)\right\}^2 \tag{5.29}$$

şeklinde hesaplanır.

5.4 Birinci Mertebeden Diferansiyel Denklem Sistemlerinin Çözümü

Tanım 5.4. K tane birinci mertebeden diferansiyel denklemden oluşan denklem sistemi,

$$\begin{aligned} &\frac{dy_r}{dx} = f_r(x, y_1, y_2, ..., y_K), \quad r = 1, 2, ..., K \quad ve \quad x \in [a, b] \\ &y_r(a) = A_r, \quad r = 1, 2, ..., K \end{aligned} \tag{5.30}$$

olarak tanımlanabilir.

Her bir $r = 1, 2, ..., K$ için y_{t_r}, bir ileri beslemeli yapay sinir ağı içeren deneme çözümü belirtmektedir. İleri beslemeli yapay sinir ağlarının çıktıları ise, $N_r(x, p_r)$, $r = 1, 2, ..., K$ ile gösterilmektedir. Burada, p_r her bir yapay sinir ağının ağırlık vektörünü, x ise girdiyi belirtmektedir. O halde, deneme çözümlerini şu şekilde ifade edebiliriz:

$$y_{t_r}(x, p_r) = A_r + (x - a)N_r(x, p_r) \quad \forall r = 1, 2, ..., K \tag{5.31}$$

Deneme çözümlerinin x'e göre türevleri;

$$\frac{dy_{t_r}(x, p_r)}{dx} = (x - a)N_r(x, p_r) + \frac{dN_r}{dx} \quad \forall r = 1, 2, ..., K \tag{5.32}$$

şeklindedir. Deneme çözümleri ve türevleri elde edildikten sonra hata fonksiyonu hesaplanabilir:

$$E(x, p) = \frac{1}{2} \sum_i \sum_{r=1}^{K} \left[\frac{dy_{t_r}(x_i, p_r)}{dx} - f_r\left(x_i, y_{t_1}(x_i, p_1), ..., y_{t_K}(x_i, p_K)\right) \right]^2. \tag{5.33}$$

6
DENEYSEL SONUÇLAR

Bu bölümde, farklı tipte lineer ve lineer olmayan başlangıç değer problemlerinin nümerik çözümleri araştırılmıştır. Bu amaçla Emden-Fowler ve Lane-Emden diferansiyel denklemleri ele alınmıştır. Denklemleri çözmek için LgNN, ChNN ve LeNN modelleri kullanılmışır. Ağı eğitmek için ise, geri yayılım algoritması ile Adaptif moment tahmin (Adam) optimizasyon tekniği kullanılmıştır. Tüm sonuçlar Python 3.8 kullanılarak elde edilmiştir.

İkinci mertebeden Emden-Fowler denkleminin genel hali şu şekilde ifade edilebilir:

$$\begin{aligned} &\frac{d^2y}{dx^2} + \frac{r}{x}\frac{dy}{dx} + kf(x)g(y) = 0 \quad x \geq 0 \\ &y(0) = \alpha, \quad y'(0) = 0. \end{aligned} \tag{6.1}$$

Denklem 6.1'de, $f(x)$ ve $g(y)$ sırasıyla, x ve y 'ye bağlı fonksiyonlardır. r, k ve α ise sabit sayılardır. Bu denklem, $x = 0$ singüler noktasına sahiptir.

Emden-Fowler denkleminde $f(x) = 1$, $g(y) = y^m$, ve $r = 2$ olarak alınırsa, standart Lane-Emden denklemini elde ederiz:

$$\begin{aligned} &\frac{d^2y}{dx^2} + \frac{2}{x}\frac{dy}{dx} + y^m = 0 \quad x \geq 0 \\ &y(0) = \alpha, \quad y'(0) = 0. \end{aligned} \tag{6.2}$$

Örnek 6.1. Standart Lane-Emden denkleminde $m = 1$ olarak alındığında,

$$\begin{aligned} &\frac{d^2y}{dx^2} + \frac{2}{x}\frac{dy}{dx} + y = 0 \\ &y(0) = 1, \quad y'(0) = 0. \end{aligned} \tag{6.3}$$

denklemi elde edilir.

Bu denklemin analitik çözümü şu şekildedir [40], [41]:

$$y(x) = \frac{sin(x)}{x} \tag{6.4}$$

Fonksiyonel bağlantılı yapay sinir ağı modelini içeren, denklemi çözmek için kullanılacak deneme çözümü,

$$y_t(x,p) = 1 + x^2 N(x,p) \tag{6.5}$$

şeklinde yazılır. LgNN, ChNN ve LeNN modellerinde, sırasıyla, ilk beş Laguerre, Chebyshev ve Legendre polinomu kullanılmıştır. $[0,1]$ aralığındaki 20 eş aralıklı nokta ağa sunulmuştur. Denklemin analitik, LgNN, ChNN ve LeNN sonuçları tablo 6.1 ile verilmiştir. $[0,1]$ aralığındaki noktalar için elde edilen analitik sonuçlar ile, LgNN, ChNN ve LeNN sonuçları arasındaki mutlak hata tablo 6.2 ile gösterilmiştir. Analitik çözümün sonuçları ile, sırasıyla, LgNN, ChNN ve LeNN sonuçları arasındaki hataların yarı logoritmik grafikleri de Şekil 6.1 ile verilmiştir.

Örnek 6.2. Denklem 6.2'de, $m = 5$ olarak alındığında;

$$\begin{aligned} &\frac{d^2y}{dx^2} + \frac{2}{x}\frac{dy}{dx} + y^5 = 0 \\ &y(0) = 1, \quad y'(0) = 0. \end{aligned} \tag{6.6}$$

lineer olmayan Lane-Emden denklemi elde edilir.

Denklemin $x \geq 0$ için analitik çözümü [40]:

$$y(x) = \left(1 + \frac{x^2}{3}\right)^{-1/2} \tag{6.7}$$

şeklindedir. Ağı eğitmek için, $[0,1]$ aralığında yer alan 10 eş aralıklı nokta

Tablo 6.1 Analitik, LgNN, ChNN ve LeNN Sonuçları (Örnek 6.1)

x	**Analitik**	**LgNN**	**ChNN**	**LeNN**
0	1.00000000	1.00000000	1.00000000	1.00000000
0.1000	0.99833417	0.99833661	0.99836312	0.99834582
0.1500	0.99625422	0.99625563	0.99621011	0.99620495
0.2000	0.99334665	0.99334360	0.99314230	0.99317438
0.2500	0.98961584	0.98960467	0.98918561	0.98927631
0.3000	0.98506736	0.98504498	0.98438473	0.98454639
0.3500	0.97970802	0.97967238	0.97879167	0.97902529
0.4000	0.97354586	0.97349627	0.97245653	0.97275184
0.4500	0.96659008	0.96652738	0.96542049	0.96575794
0.5000	0.95885108	0.95877762	0.95771097	0.95806510
0.5500	0.95034042	0.95025985	0.94933906	0.94968259
0.6000	0.94107079	0.94098783	0.94029910	0.94060738
0.6500	0.93105601	0.93097599	0.93057062	0.93082574
0.7000	0.92031098	0.92023938	0.92012246	0.92031650
0.7500	0.90885168	0.90879355	0.90891924	0.90905605
0.8000	0.89669511	0.89665441	0.89693003	0.89702512
0.8500	0.88385930	0.88383823	0.88413934	0.88421712
0.9000	0.87036323	0.87036154	0.87056038	0.87064832
0.9500	0.85622685	0.85624110	0.85625048	0.85636960
1.0000	0.84147098	0.84149384	0.84132889	0.84148006

kullanılmıştır. Fonksiyonel bağlantılı yapay sinir ağı modellerinde, ilk 5 ortogonal polinom kullanılmıştır. Analitik sonuçlar ile LgNN, ChNN ve LeNN modellerinin sonuçlarının kıyaslanması tablo 6.3 ile verilmiştir. Analitik çözüm ile, LgNN, ChNN ve LeNN çözümleri arasındaki mutlak hata ise tablo 6.4 ile gösterilmiştir. Analitik sonuçlar ile sırasıyla, LgNN, ChNN ve LeNN sonuçları arasındaki hataların grafikleri de Şekil 6.2 ile verilmiştir.

Örnek 6.3.

$$\begin{aligned} y'' + \frac{3}{x}y' + 2x^2y^2 = 0 \\ y(0) = 1, \quad y'(0) = 0, \end{aligned} \tag{6.8}$$

lineer olmayan bir Emden-Fowler denklemidir.

Bu denklemin deneme çözümü şu şekilde yazılabilir:

$$y_t(x,p) = 1 + x^2 N(x,p) \tag{6.9}$$

Fonksiyonel bağlantılı yapay sinir ağı modelleri, $[0,1]$ aralığındaki 10 eş aralıklı

Tablo 6.2 Analitik sonuçlar ile, LgNN, ChNN ve LeNN sonuçları arasındaki hata (Örnek 6.1)

x	LgNN Hata	**ChNN** Hata	**LeNN** Hata
0	0	0	0
0.1000	2.440e-06	2.89500e-05	1.1650e-05
0.1500	1.410e-06	4.41100e-05	4.9270e-05
0.2000	3.050e-06	2.04350e-04	1.7227e-04
0.2500	1.117e-05	4.30230e-04	3.3953e-04
0.3000	2.238e-05	6.82630e-04	5.2097e-04
0.3500	3.564e-05	9.16350e-04	6.8273e-04
0.4000	4.959e-05	1.08933e-03	7.9402e-04
0.4500	6.270e-05	1.16959e-03	8.3214e-04
0.5000	7.346e-05	1.14011e-03	7.8598e-04
0.5500	8.057e-05	1.00136e-03	6.5783e-04
0.6000	8.296e-05	7.71690e-04	4.6341e-04
0.6500	8.002e-05	4.85390e-04	2.3027e-04
0.7000	7.160e-05	1.88520e-04	5.5200e-06
0.7500	5.813e-05	6.75600e-05	2.0437e-04
0.8000	4.070e-05	2.34920e-04	3.3001e-04
0.8500	2.107e-05	2.80040e-04	3.5782e-04
0.9000	1.690e-06	1.97150e-04	2.8509e-04
0.9500	1.425e-05	2.36300e-05	1.4275e-04
1.0000	2.286e-05	1.42090e-04	9.0800e-06

nokta için eğitilmiştir. Laguerre yapay sinir ağı modelinde, ilk 6 Laguerre polinomu kullanılmıştır. ChNN ve LeNN modellerinde ise, ilk 5 Chebyshev ve Legendre polinomları kullanılmıştır. Denklemin Maple 11 [42] sonuçları ile, LgNN, ChNN ve LeNN sonuçları tablo 6.5'de verilmiştir. Maple 11 [42] ile, LgNN, ChNN ve LeNN sonuçları arasındaki mutlak hata ise tablo 6.6 ile gösterilmiştir. Maple 11 [42] ile, her bir FLANN modeli arasındaki hatanın grafiği ise, Şekil 6.3 ile verilmiştir.

Örnek 6.4. Denklem 6.10 ile, lineer olmayan bir Emden-Fowler denklemi verilmiştir.

$$\begin{aligned} &y'' + \frac{8}{x}y' + xy^2 = x^4 + x^5 \\ &y(0) = 1, \quad y'(0) = 0. \end{aligned} \tag{6.10}$$

Denklemin deneme çözümü şu şekilde ifade edilebilir:

$$y_t(x,p) = 1 + x^2 N(x,p) \tag{6.11}$$

Tablo 6.3 Analitik, LgNN, ChNN ve LeNN sonuçları (Örnek 6.2)

x	**Analitik**	**LgNN**	**ChNN**	**LeNN**
0	1.00000000	1.00000000	1.00000000	1.00000000
0.1	0.99833748	0.99833840	0.99833303	0.99831151
0.2	0.99339926	0.99338343	0.99331759	0.99305859
0.3	0.98532927	0.98527436	0.98511802	0.98447819
0.4	0.97435470	0.97425115	0.97404599	0.97313921
0.5	0.96076892	0.96062480	0.96046326	0.95961153
0.6	0.94491118	0.94474986	0.94471851	0.94427102
0.7	0.92714554	0.92699907	0.92711820	0.92724142
0.8	0.90784129	0.90774012	0.90793194	0.90847624
0.9	0.88735650	0.88731439	0.88743266	0.88798289
1.0	0.86602540	0.86601788	0.86597203	0.86618944

Tablo 6.4 Analitik sonuçlar ile, LgNN, ChNN ve LeNN sonuçları arasındaki hata (Örnek 6.2)

x	LgNN Hata	**ChNN** Hata	**LeNN** Hata
0	0	0	0
0.1000	9.2000e-07	4.4500e-06	2.59700e-05
0.2000	1.5830e-05	8.1670e-05	3.40670e-04
0.3000	5.4910e-05	2.1125e-04	8.51080e-04
0.4000	1.0355e-04	3.0871e-04	1.21549e-03
0.5000	1.4412e-04	3.0566e-04	1.15739e-03
0.6000	1.6132e-04	1.9267e-04	6.40160e-04
0.7000	1.4647e-04	2.7340e-05	9.58800e-05
0.8000	1.0117e-04	9.0650e-05	6.34950e-04
0.9000	4.2110e-05	7.6160e-05	6.26390e-04
1.0000	7.5200e-06	5.3370e-05	1.64040e-04

Deneme çözümünde yer alan yapay sinir ağı modelleri, $[0, 1]$ aralığındaki 10 eş aralıklı nokta için eğitilmiştir. LgNN ve LeNN yapay sinir ağı modellerinde, sırasıyla, ilk 5 Laguerre ve Legendre polinomları kullanılır iken, ChNN modelinde ise, ilk 6 Chebyshev polinomu kullanılmıştır. Denklemin Maple 11 [42] sonuçları ile, LgNN, ChNN ve LeNN sonuçları Tablo 6.7 ile verilmiştir. Maple 11 [42] sonuçları ile, sırasıyla, LgNN, ChNN ve LeNN sonuçları arasındaki hata ve hatann grafikleri ise tablo 6.8 ve şekil 6.4 ile gösterilmiştir.

Tablo 6.5 Maple 11[42] , LgNN, ChNN ve LeNN sonuçları (Örnek 6.3)

x	**Maple 11**	**LgNN**	**ChNN**	**LeNN**
0	1.00000000	1.00000000	1.00000000	1.00000000
0.1	0.99999166	0.99998814	0.99998599	0.99995072
0.2	0.99986667	0.99984396	0.99979883	0.99972765
0.3	0.99932527	0.99928284	0.99916486	0.99907859
0.4	0.99786939	0.99782557	0.99765512	0.99749869
0.5	0.99480789	0.99478190	0.99462205	0.99415400
0.6	0.98926958	0.98926099	0.98918133	0.98781351
0.7	0.98022937	0.98020664	0.98023911	0.97679108
0.8	0.96655340	0.96645546	0.96656485	0.95889900
0.9	0.94706857	0.94681535	0.94690948	0.93141662
1.0	0.92065853	0.92016168	0.92016793	0.89107881

Tablo 6.6 Maple 11 ile, LgNN, ChNN ve LeNN sonuçları arasındaki hata (Örnek 6.3)

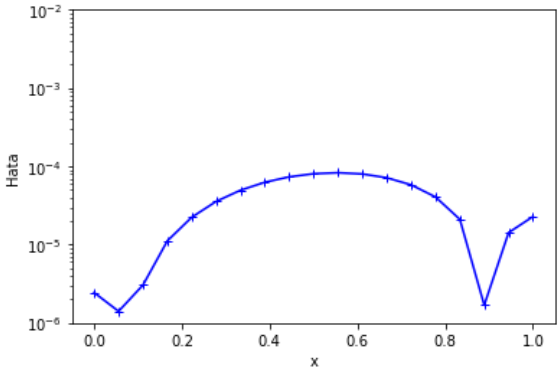
x	LgNN Hata	**ChNN** Hata	**LeNN** Hata
0	0	0	0
0.1000	3.5200e-06	5.6700e-06	4.094000e-05
0.2000	2.2710e-05	6.7840e-05	1.390200e-04
0.3000	4.2430e-05	1.6041e-04	2.466800e-04
0.4000	4.3820e-05	2.1427e-04	3.707000e-04
0.5000	2.5990e-05	1.8584e-04	6.538900e-04
0.6000	8.5900e-06	8.8250e-05	1.456070e-03
0.7000	2.2730e-05	9.7400e-06	3.438290e-03
0.8000	9.7940e-05	1.1450e-05	7.654400e-03
0.9000	2.5322e-04	1.5909e-04	1.565195e-02
1.0000	4.9685e-04	4.9060e-04	2.957972e-02

Tablo 6.7 Maple 11 [42] , LgNN, ChNN ve LeNN sonuçları (Örnek 6.4)

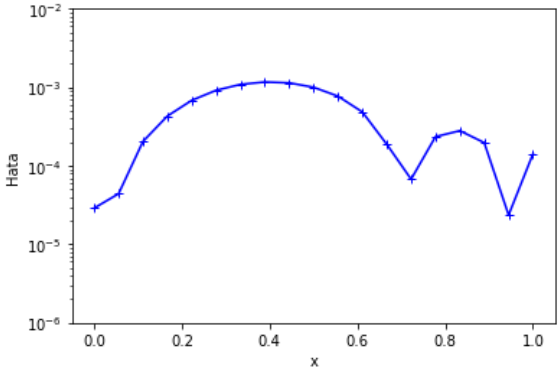
x	**Maple 11**	**LgNN**	**ChNN**	**LeNN**
0	1.00000000	1.00000000	1.00000000	1.00000000
0.1	0.99996668	0.99997457	0.99994913	0.99997701
0.2	0.99973433	0.99978885	0.99968339	0.99977965
0.3	0.99911219	0.9992149	0.99909661	0.99920955
0.4	0.99793933	0.9980277	0.99803933	0.99809623
0.5	0.99612622	0.99610901	0.99632951	0.99636346
0.6	0.99372097	0.99354405	0.99390096	0.99412187
0.7	0.99100463	0.99071096	0.99101179	0.99178771
0.8	0.98861928	0.98836308	0.98843507	0.99022791
0.9	0.98773192	0.98770443	0.98755418	0.99093169
1.0	0.99023588	0.99045805	0.99028518	0.99620848

Tablo 6.8 Maple 11 [42] ile, LgNN, ChNN ve LeNN sonuçları arasındaki hata (Örnek 6.4)

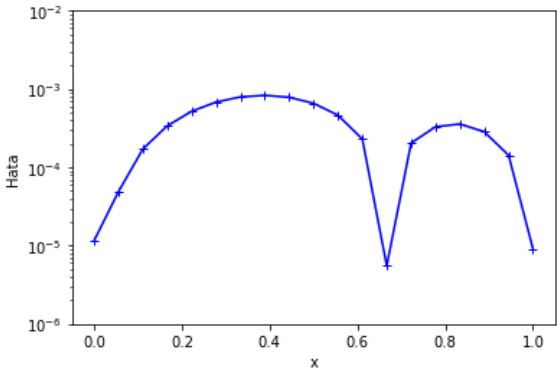
x	**LgNN** Hata	**ChNN** Hata	**LeNN** Hata
0	0	0	0
0.1000	7.8900e-06	1.7550e-05	1.03300e-05
0.2000	5.4520e-05	5.0940e-05	4.53200e-05
0.3000	1.0271e-04	1.5580e-05	9.73600e-05
0.4000	8.8370e-05	1.0000e-04	1.56900e-04
0.5000	1.7210e-05	2.0329e-04	2.37240e-04
0.6000	1.7692e-04	1.7999e-04	4.00900e-04
0.7000	2.9367e-04	7.1600e-06	7.83080e-04
0.8000	2.5620e-04	1.8421e-04	1.60863e-03
0.9000	2.7490e-05	1.7774e-04	3.19977e-03
1.0000	2.2217e-04	4.9300e-05	5.97260e-03



(a) LgNN

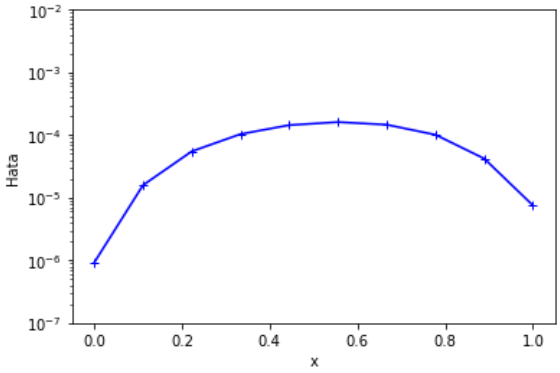


(b) ChNN

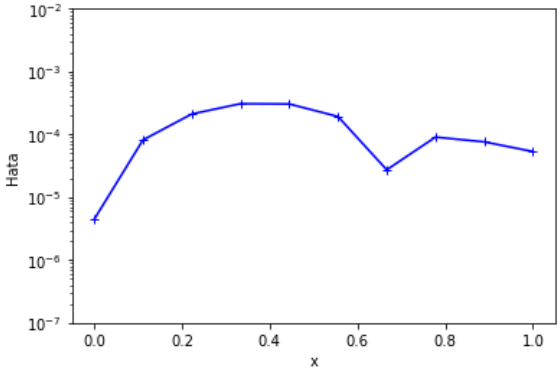


(c) LeNN

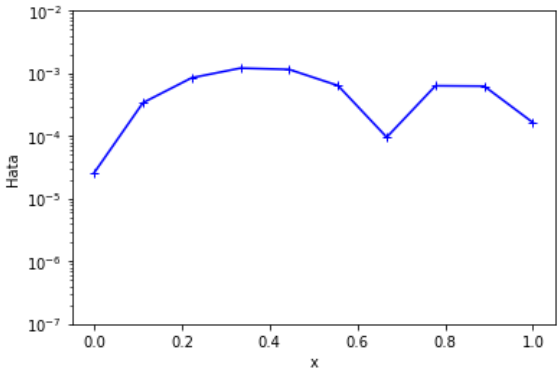
Şekil 6.1 Analitik sonuçlar ile, sırasıyla, LgNN, ChNN ve LeNN sonuçları arasındaki hataların yarı logoritmik grafikleri (Örnek 6.1)



(a) LgNN

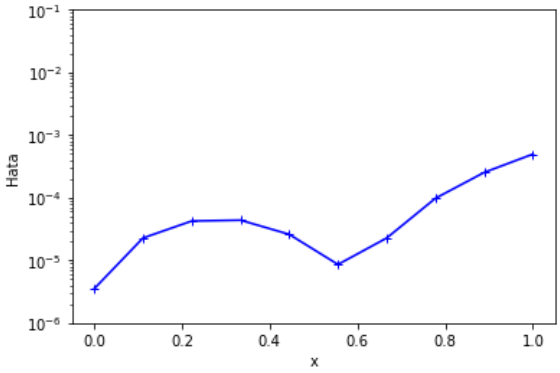


(b) ChNN

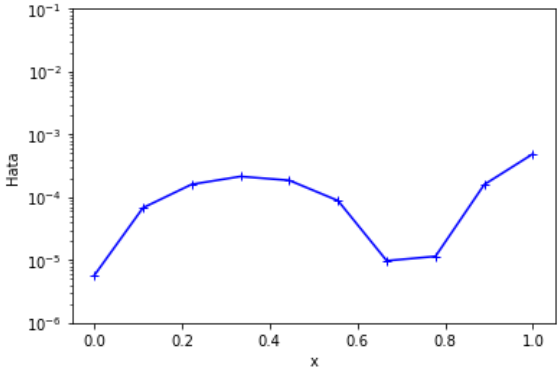


(c) LeNN

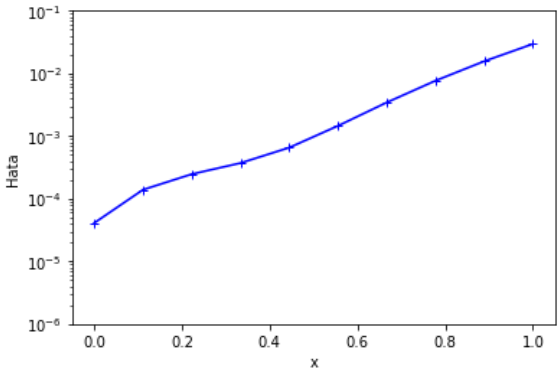
Şekil 6.2 Analitik sonuçlar ile, sırasıyla, LgNN, ChNN ve LeNN sonuçları arasındaki hataların yarı logoritmik grafikleri (Örnek 6.2)



(a) LgNN

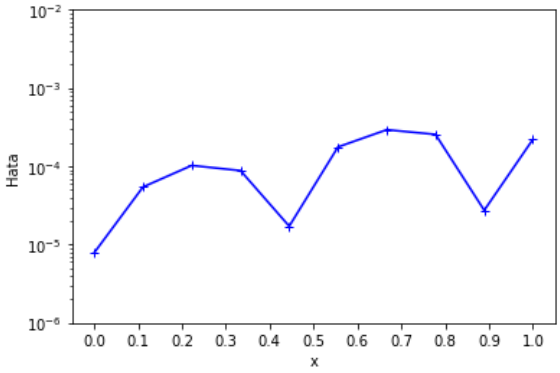


(b) ChNN

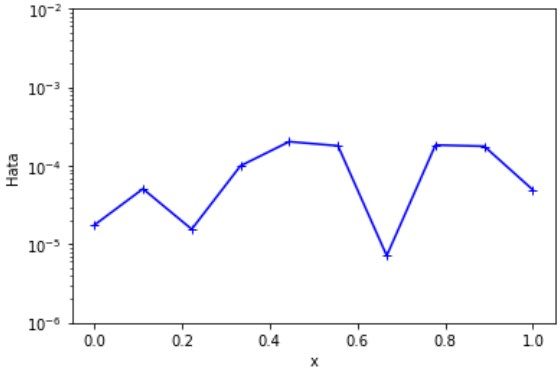


(c) LeNN

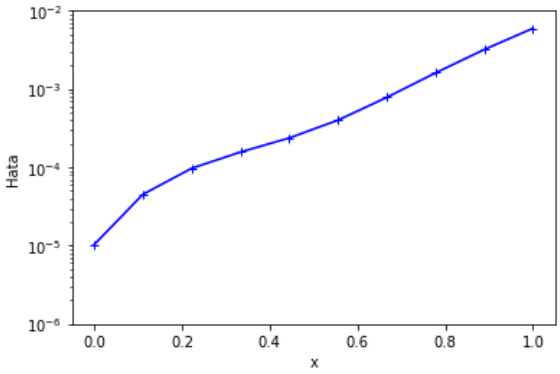
Şekil 6.3 Maple 11 [42] sonuçları ile, sırasıyla, LgNN, ChNN ve LeNN sonuçları arasındaki hataların yarı logoritmik grafikleri (Örnek 6.3)



(a) LgNN



(b) ChNN



(c) LeNN

Şekil 6.4 Maple 11 [42] sonuçları ile, sırasıyla, LgNN, ChNN ve LeNN sonuçları arasındaki hataların yarı logoritmik grafikleri (Örnek 6.4)

7
SONUÇ VE ÖNERİLER

Bu çalışmada, tek katmanlı fonksiyonel bağlantılı yapay sinir ağı (FLANN) modelleri, lineer ve lineer olmayan singüler ikinci dereceden başlangıç değer problemlerinin nümerik çözümlerini elde etmek için kullanılmıştır. Denklemleri çözmek için 3 farklı tek katmanlı fonksiyonel bağlantılı yapay sinir ağı modeli kullanılmıştır. Bu sinir ağları, Laguerre yapay sinir ağı (LgNN), Chebyshev yapay sinir ağı (ChNN) ve Legendre yapay sinir ağıdır (LeNN). Ağları eğitmek için, geri yayılım öğrenme algoritması ve adaptif moment tahmin (Adam) optimizasyon tekniği kullanılmıştır.

Tek katmanlı fonksiyonel bağlantılı yapay sinir ağı modellerini kıyaslamak için, Emden-Fowler ve Lane-Emden tipi denklemlerin çözümleri araştırılmıştır. Bu denklemlerin $x = 0$ singüler noktasına sahip olması, denklemleri çözmeyi zorlaştırmaktadır. Fakat, yapay sinir ağları kullanılarak, bu denklemlerin nümerik çözümleri elde edilebilmektedir.

Deneysel çalışmalarda elde edilen nümerik çözümler incelendiğinde; Laguerre yapay sinir ağı ile Chebyshev yapay sinir ağının, Legendre yapay sinir ağına göre daha iyi sonuçlar verdiği görülmektedir. Ancak, Laguerre yapay sinir ağı kullanarak elde edilen nümerik çözümlerin hata değerleri, Chebyshev yapay sinir ağı kullanarak elde edilenlere göre genellikle daha az olmaktadır.

Bu çalışmada, fonksiyonel bağlantılı yapay sinir ağı (FLANN) modelleri adi diferansiyel denklemleri çözmek için kullanılmıştır. İleriki çalışmalarda, kısmi diferansiyel denklemlerin nümerik çözümlerini elde etmek için de kullanılabilirler.

KAYNAKÇA

[1] H. Lee, I. S. Kang, "Neural algorithm for solving differential equations," *Journal of Computational Physics*, vol. 91, no. 1, pp. 110–131, 1990.

[2] A. J. Meade Jr, A. A. Fernandez, "The numerical solution of linear ordinary differential equations by feedforward neural networks," *Mathematical and Computer Modelling*, vol. 19, no. 12, pp. 1–25, 1994.

[3] A. J. Meade Jr, A. A. Fernandez, "Solution of nonlinear ordinary differential equations by feedforward neural networks," *Mathematical and Computer Modelling*, vol. 20, no. 9, pp. 19–44, 1994.

[4] I. E. Lagaris, A. Likas, D. I. Fotiadis, "Artificial neural networks for solving ordinary and partial differential equations," *IEEE transactions on neural networks*, vol. 9, no. 5, pp. 987–1000, 1998.

[5] A. Malek, R. S. Beidokhti, "Numerical solution for high order differential equations using a hybrid neural network—optimization method," *Applied Mathematics and Computation*, vol. 183, no. 1, pp. 260–271, 2006.

[6] H. S. Yazdi, M. Pakdaman, H. Modaghegh, "Unsupervised kernel least mean square algorithm for solving ordinary differential equations," *Neurocomputing*, vol. 74, no. 12-13, pp. 2062–2071, 2011.

[7] M. Kumar, N. Yadav, "Multilayer perceptrons and radial basis function neural network methods for the solution of differential equations: A survey," *Computers & Mathematics with Applications*, vol. 62, no. 10, pp. 3796–3811, 2011.

[8] H. Qu, X. Liu, "A numerical method for solving fractional differential equations by using neural network," *Advances in Mathematical Physics*, vol. 2015, 2015.

[9] S. Mall, S. Chakraverty, "Application of legendre neural network for solving ordinary differential equations," *Applied Soft Computing*, vol. 43, pp. 347–356, 2016.

[10] A. H. Hadian-Rasanan, D. Rahmati, S. Gorgin, K. Parand, "A single layer fractional orthogonal neural network for solving various types of lane–emden equation," *New Astronomy*, vol. 75, p. 101 307, 2020.

[11] S. Mall, S. Chakraverty, "Chebyshev neural network based model for solving lane–emden type equations," *Applied Mathematics and Computation*, vol. 247, pp. 100–114, 2014.

[12] S. Mall, S. Chakraverty, "Numerical solution of nonlinear singular initial value problems of emden–fowler type using chebyshev neural network method," *Neurocomputing*, vol. 149, pp. 975–982, 2015.

[13] S. Chakraverty, S. Mall, "Single layer chebyshev neural network model with regression-based weights for solving nonlinear ordinary differential equations," *Evolutionary Intelligence*, pp. 1–8, 2020.

[14] Y. Yang, M. Hou, J. Luo, "A novel improved extreme learning machine algorithm in solving ordinary differential equations by legendre neural network methods," *Advances in Difference Equations*, vol. 2018, no. 1, pp. 1–24, 2018.

[15] S. Chakraverty, *Mathematical methods in interdisciplinary sciences*. John Wiley & Sons, 2020.

[16] Y. Chen, H. Yu, X. Meng, X. Xie, M. Hou, J. Chevallier, "Numerical solving of the generalized black-scholes differential equation using laguerre neural network," *Digital Signal Processing*, vol. 112, p. 103 003, 2021.

[17] S. Chakraverty, S. Mall, *Artificial neural networks for engineers and scientists: solving ordinary differential equations*. CRC Press, 2017.

[18] S. Haykin, *Neural networks and learning machines, 3/E*. Pearson Education India, 2010.

[19] W. S. McCulloch, W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[20] D. O. Hebb *et al.*, *The organization of behavior: A neuropsychological theory*. New York: Wiley, 1949.

[21] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain.," *Psychological review*, vol. 65, no. 6, p. 386, 1958.

[22] B. Widrow, M. E. Hoff, "Adaptive switching circuits," Stanford Univ Ca Stanford Electronics Labs, Tech. Rep., 1960.

[23] M. Minsky, S. Papert, "An introduction to computational geometry," *Cambridge tiass., HIT*, 1969.

[24] T. Kohonen, "Self-organized formation of topologically correct feature maps," *Biological cybernetics*, vol. 43, no. 1, pp. 59–69, 1982.

[25] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proceedings of the national academy of sciences*, vol. 79, no. 8, pp. 2554–2558, 1982.

[26] P. Werbos, "Beyond regression:" new tools for prediction and analysis in the behavioral sciences," *Ph. D. dissertation, Harvard University*, 1974.

[27] D. B. Parker, "Learnins logic.," *Technical Report*, 1985.

[28] D. E. Rumelhart, G. E. Hinton, R. J. Williams, "Learning internal representations by error propagation," California Univ San Diego La Jolla Inst for Cognitive Science, Tech. Rep., 1985.

[29] D. S. Broomhead, D. Lowe, "Radial basis functions, multi-variable functional interpolation and adaptive networks," Royal Signals and Radar Establishment Malvern (United Kingdom), Tech. Rep., 1988.

[30] I. N. Da Silva, D. H. Spatti, R. A. Flauzino, L. H. B. Liboni, S. F. dos Reis Alves, "Artificial neural networks," *Cham: Springer International Publishing*, vol. 39, 2017.

[31] E. Alpaydin2nd, "Introduction to machine learning," 2010.

[32] C. C. Aggarwal, *Linear Algebra and Optimization for Machine Learning: A Textbook*. Springer, 2020.

[33] Y. Nesterov, "A method for unconstrained convex minimization problem with the rate of convergence o (1/kˆ 2)," in *Doklady an ussr*, vol. 269, 1983, pp. 543–547.

[34] J. Duchi, E. Hazan, Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization.," *Journal of machine learning research*, vol. 12, no. 7, 2011.

[35] M. D. Zeiler, "Adadelta: An adaptive learning rate method," *arXiv preprint arXiv:1212.5701*, 2012.

[36] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.

[37] G. Hinton, N. Srivastava, K. Swersky, "Neural networks for machine learning lecture 6a overview of mini-batch gradient descent," *Cited on*, vol. 14, no. 8, p. 2, 2012.

[38] D. P. Kingma, J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[39] Y.-H. Pao, S. M. Phillips, "The functional link net and learning optimal control," *Neurocomputing*, vol. 9, no. 2, pp. 149–164, 1995.

[40] H. T. Davis, *Introduction to nonlinear differential and integral equations*. US Atomic Energy Commission, 1960.

[41] C. M. Bender, K. A. Milton, S. S. Pinsky, L. Simmons Jr, "A new perturbative approach to nonlinear problems," *Journal of mathematical Physics*, vol. 30, no. 7, pp. 1447–1455, 1989.

[42] H. Demir, İ. Ç. Süngü, "Numerical solution of a class of nonlinear emden-fowler equations by using differential transform method," *Cankaya University Journal of Science and Engineering*, vol. 12, no. 2, pp. 75–82,

TEZDEN ÜRETİLMİŞ YAYINLAR

Konferans Bildirisi

1. S.N. KURMANÇ, ”Comparıson of Functional Link Artificial Neural Networks for Solving Lane-Emden type Equations”, 8th International Congress on Fundamental and Applied Sciences 2021 (ICFAS2021), 19-21 October, 2021, ss.167