T.C.
VAN YÜZÜNCÜ YIL ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ
İSTATİSTİK ANABİLİM DALI

# AĞ TEHDİT ALGILAMADA DERİN ÖĞRENME UYGULAMALARI

YÜKSEK LİSANS TEZİ

HAZIRLAYAN: Mesut KAPAR
DANIŞMAN: Dr. Öğr. Ü. Murat CANAYAZ

VAN-2019

T.C.
VAN YÜZÜNCÜ YIL ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ
İSTATİSTİK ANABİLİM DALI

# AĞ TEHDİT ALGILAMADA DERİN ÖĞRENME UYGULAMALARI

YÜKSEK LİSANS TEZİ

HAZIRLAYAN: Mesut KAPAR

VAN-2019

## KABUL VE ONAY SAYFASI

İstatistik Anabilim Dalı' nda Dr. Öğr. Ü. Murat CANAYAZ danışmanlığında, Mesut KAPAR tarafından sunulan "**Ağ Tehdit Algılamada Derin Öğrenme Uygulamaları**" isimli bu çalışma Lisansüstü Eğitim ve Öğretim Yönetmeliği' nin ilgili hükümleri gereğince 06/08/2019 tarihinde aşağıdaki jüri tarafından oy birliği ile başarılı bulunmuş ve Yüksek Lisans tezi olarak kabul edilmiştir.

Başkan: İmza:

Prof. Dr. H. Eray ÇELİK

Üye: İmza:

Dr. Öğr. Üyesi Murat CANAYAZ

Üye: İmza:

Dr. Öğr. Üyesi Yılmaz KAYA

Fen Bilimleri Enstitüsü Yönetim Kurulu'nun …./..../….. tarih ve ..... sayılı kararı ile onaylanmıştır.

Enstitü Müdürü

## TEZ BİLDİRİMİ

Tez içindeki bütün bilgilerin etik davranış ve akademik kurallar çerçevesinde elde edilerek sunulduğunu, ayrıca tez yazım kurallarına uygun olarak hazırlanan bu çalışmada bana ait olmayan her türlü ifade ve bilginin kaynağına eksiksiz atıf yapıldığını bildiririm.

Mesut KAPAR

# ÖZET

## AĞ TEHDİT ALGILAMADA DERİN ÖĞRENME UYGULAMALARI

KAPAR, Mesut
Yüksek Lisans Tezi, İstatistik Anabilim Dalı
Tez Danışmanı: Dr. Öğr. Ü. Murat CANAYAZ
Eylül 2019, 73 sayfa

Ağ saldırı tespit sistemleri günümüz bilişim sistemlerinde kritik bir yer teşkil ederken, önemli bir araştırma alanı olarak yükselmeye ve yapay sinir ağlarının kullanımı bu alanda giderek daha popüler hale gelmeye başlamıştır. Buna rağmen, bu alanda yapay sinir ağı mimarileri ve bu mimarilerin parametreleri hakkında karşılaştırma çalışmalarında eksiklikler vardır. Bu çalışmada, ağ saldırı tespit sistemleri alanında kullanılan yapay sinir ağları mimarileri ile ileride yapılacak olan mühendislik ve akademik çalışmalar için bir temel oluşturulması amaçlanmıştır. Bu doğrultuda kıyaslama veri seti olarak kabul edilen KDD-99 veri seti kullanılmıştır. İleri beslemeli yapay sinir ağı mimarisi ve evrişimsel sinir ağı mimarisi bu veri seti üzerinde kullanılmış ve % 90 üzeri başarı elde edilerek sonuçlar karşılaştırılmıştır.

**Anahtar kelimeler:** Derin öğrenme, Evrişimsel sinir ağları, KDD-99 veri seti, Nids, Yapay sinir ağları.

## ABSTRACT

### DEEP LEARNING APLICATIONS FOR NETWORK INTRUSION DETECTION

KAPAR, Mesut
M.Sc. Thesis, Department of Statistics
Supervisor: Asst. Prof. Dr. Murat CANAYAZ
September 2019, 73 pages

Network intrusion detection systems are a critical part of today's information systems, it has started to rise as an essential research area, and the use of artificial neural networks has become increasingly popular in this area. However, there are deficiencies in the comparison studies about artificial neural network architectures and their parameters. In this study, it is aimed to create a basis for future research and academic studies with artificial neural network architectures used in the field of network intrusion detection systems. In this respect, KDD-99, which is accepted as the benchmark data set, was used. Feedforward artificial neural network architecture and convolutional neural network architecture have been used on this dataset, and over 90% success has been achieved, and the results have been compared.



**Keywords:** Artificial neural networks, Convolutional neural networks, Deep learning, KDD-99 dataset, Nids.

# ÖN SÖZ

Bu tez çalışmasında verdiği desteklerden dolayı Van Yüzüncü Yıl Üniversitesi Bilgisayar Bilimleri Araştırma ve Uygulama Merkezi' nde çalışan bütün arkadaşlarıma, her türlü ilgi ve yardımlarını esirgemeyen danışmanım Sayın Dr. Öğr. Ü. Murat CANAYAZ' a, değerli katkılarından dolayı hocam Prof. Dr. H. Eray ÇELİK' e, arkadaşım Arş. Gör. Fatih ULUDAĞ ve Öğr. Gör. Fırat KAPAR' a ayrıca çalışmalarımda bana her zaman destek olan aileme ve değerli dostlarıma en içten teşekkürlerimi sunarım.

2019

Mesut KAPAR

# İÇİNDEKİLER

**Sayfa**

**Sayfa**

# ÇİZELGELER LİSTESİ

**Çizelge** **Sayfa**

# ŞEKİLLER LİSTESİ

**Şekil** **Sayfa**

**Şekil** **Sayfa**

## SİMGELER VE KISALTMALAR

Bu çalışmada kullanılmış bazı simgeler ve kısaltmalar, açıklamaları ile birlikte aşağıda sunulmuştur.

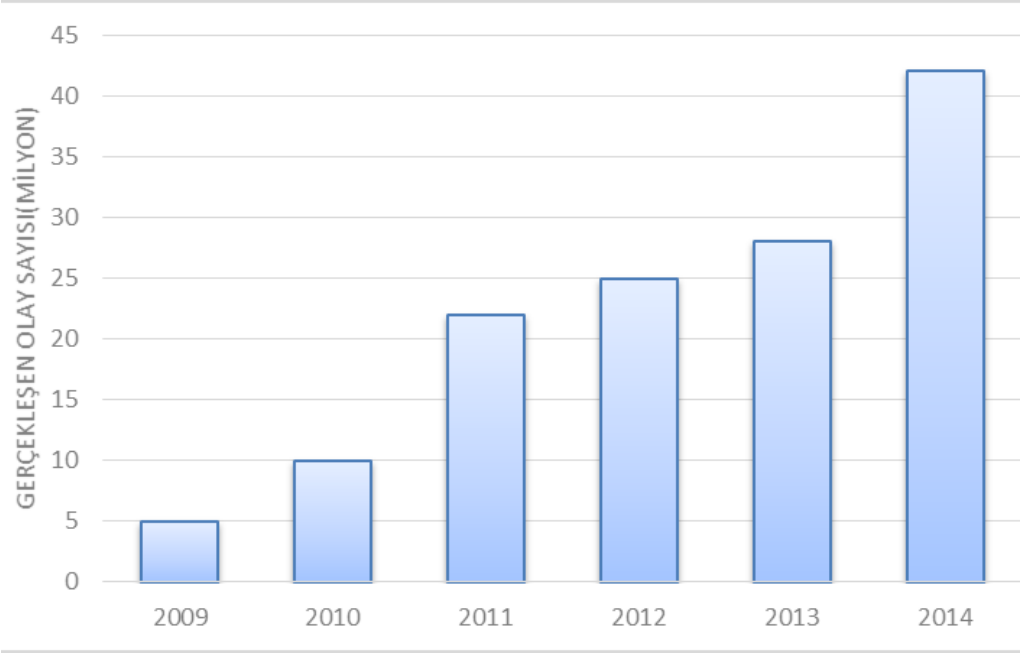
| Kısaltmalar | Açıklama |
|---|---|
| **CNN** | Convolutional Neural Network |
| **DDOS** | Distributed Denial of Service |
| **DIDS** | Distributed Intrusion Detection System |
| **FFNN** | Feed Forward Neural Network |
| **HIDS** | Host Based Intrusion Detection System |
| **IDS** | Intrusion Detection System |
| **NIDS** | Network Intrusion Detection System |
| **RNN** | Recurrent Neural Network |
| **R2L** | Remote to Local Attack |
| **SOM** | Self Organized Maps |
| **U2R** | User to Root Attack |

# 1. GİRİŞ

Bilişim teknolojilerinin günlük yaşamda yaygınlaşmasıyla birlikte, bilgisayar güvenliği bir zorunluluk haline gelmiştir. Bilgisayarlı sistemlerin toplu kullanımı, "sıfır gün" güvenlik açıkları, mobil tehditler vb. gibi kritik tehditlere yol açmıştır. Güvenlik alanındaki araştırmalar önemli ölçüde artmış olmasına rağmen, henüz bu tehtidlerde yeterince azalma görülmemiştir.

Bilgisayar ağlarının evrimi, özellikle günümüz ağ ortamındaki internet güvenliği ve gelişmiş bilgi işlem olanaklarındaki bilgisayar güvenliği endişelerini büyük ölçüde arttırmıştır.

*Symantec* İnternet Güvenliği Tehdit Raporu'na göre, 2010' da üç milyardan fazla kötü amaçlı yazılım saldırısı rapor edilmiş ve hizmet reddi saldırılarının, 2013' e kadar çarpıcı bir şekilde arttığı belirtilmiştir (Ahmed ve ark., 2016).




Şekil 1.1. Bilgi güvenliği olaylarının büyümesi (Bilgi Güvenliği Küresel Anketi Araştırması, 2015).

Dünya, internetin ve yeni ağ teknolojilerinin ortaya çıkışıyla birlikte daha fazla bağlantılı hale gelmiştir. Çok sayıda kişisel, ticari, askeri ve devlet bilgisi bu ağ yapısı üzerinde bulunmaktadır. İnternet üzerinden kolaylıkla elde edilebilecek fikri mülkiyet nedeniyle ağ güvenliği büyük önem kazanmıştır.

Halen iki temelde farklı ağ, veri ağı ve anahtarlardan oluşan senkron ağ bulunmaktadır. İnternet bir veri ağı olarak kabul edilir. Mevcut veri ağı bilgisayar tabanlı yönlendiricilerden oluştuğu için, yönlendiricilere yerleştirilen "Truva atları" gibi özel programlar aracılığıyla bilgi elde edilebilir.

Anahtarlardan oluşan senkron ağ, verileri tamponlamaz ve bu nedenle saldırganlar tarafından tehdit edilmez. Bu yüzden, internet gibi veri ağlarında ve internete bağlanan diğer ağlarda güvenlik vurgulanmaktadır (Daya, 2013).

### 1.1. Ağ Tabanlı Saldırı Tespit Sistemleri (NIDS)

Niyaz ve ark. (2016), ağ saldırısı tespit sistemleri (NIDS)' ni, ağ sistemi yöneticilerinin, bir kuruluşun ağındaki çeşitli güvenlik ihlallerini algılaması için temel bir araç olarak tanımlamışlardır. Bir NIDS, bir kuruluşun ağ cihazlarına giren veya çıkan ağ trafiğini izler, analiz eder ve saldırı yapılırsa uyarı verir.

Alom ve ark. (2015), saldırı tespit sistemi (IDS)' i, üç temel işleve göre gruplandırmışlardır.

*Bilgi Kaynaklarını İzleme:* Yetkisiz erişim aktiviteleri için bilgisayarlar veya ağlardaki kaynaklarla ilgili aktiviteleri izler.

*Analiz:* İzleme sürecinde toplanan olay günlüklerini ve verileri kullanarak yetkisiz aktiviteleri tespit eder. Suistimal yaklaşımı ve anormallik tespiti analiz yaklaşımı en yaygın olanlarıdır.

*Yanıt:* Bir saldırı algılandığında sistemin gerçekleştirdiği eylemler kümesidir.

IDS işlevlerine göre aşağıdaki üç ana kategoride sınıflandırılabilir:

a) Ağ tabanlı saldırı tespit sistemi (NIDS)

NIDS, gelen paketleri veritabanında kayıtlı bilinen saldırı içerikleriyle karşılaştırır. Eğer eşleşme tespit edilirse, tanımlanmış kurallara göre kullanıcıya

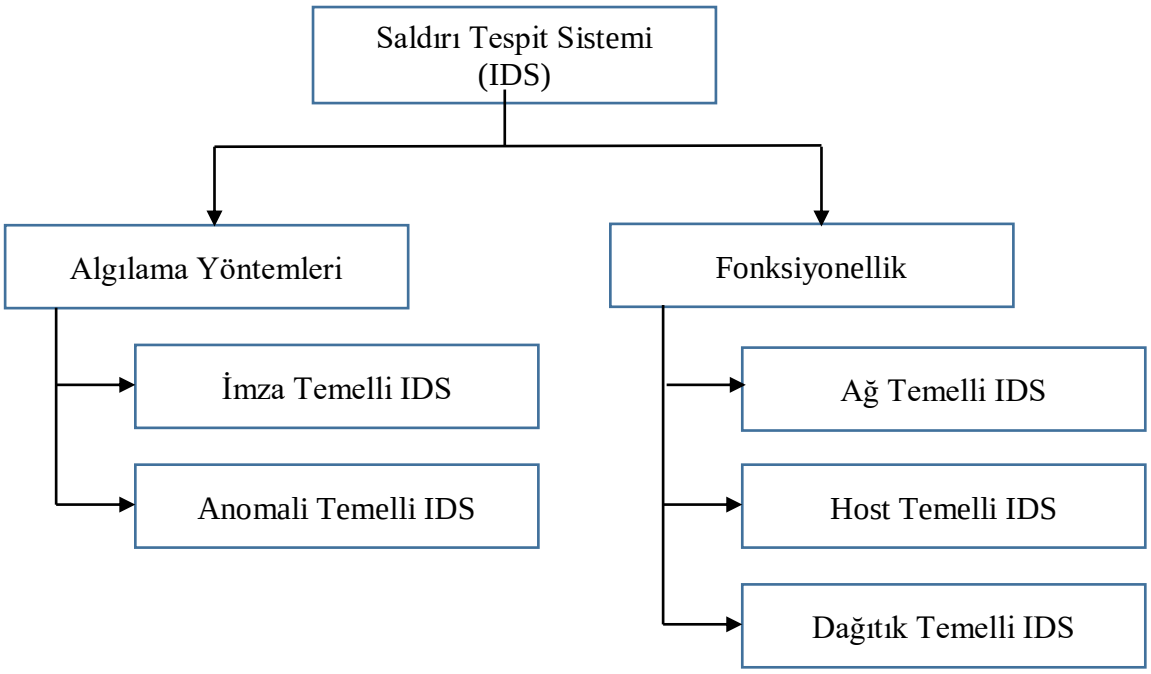
uyarıverir. NIDS örnekleri arasında SNORT, BRO, CISCO, NIDS, NETPROWLER vb. yer almaktadır.

b) Host tabanlı saldırı tespit sistemi (HIDS)

HIDS, ağ paketlerinden ziyade bir bilgisayar sisteminin iç kısımlarını izler ve analiz eder. Saldırının olup olmadığını belirleyen çeşitli sistem günlüklerini veya denetim günlüklerini izler, bunları dikkate alır ve saldırının gerçekleşmesi durumunda sistem yöneticisini uyarır. HIDS örnekleri arasında OSSEC, CISCO HIDS, TRIPWIRE vb. yer almaktadır.

c) Dağıtık saldırı tespit sistemi (DIDS)

DIDS'de saldırıları denetleyen algılayıcılar, NIDS, HIDS veya her ikisinin bir kombinasyonu olabilir. Ağdaki dağıtılmış tüm algılayıcılar merkezi bir yönetim sistemine rapor verir. Algılayıcılardan herhangi biri bir saldırı tespit ederse, DIDS yönetim konsolu ağın güvenliğini sağlayan diğer algılayıcılarda da imzaları günceller.




Şekil 1.2. Saldırı tespit sistemi metodolojisi.

Ayrıca, IDS, algılama tekniklerine dayalı olarak iki kategoriye ayrılabilir:

a) İmza tabanlı saldırı tespit sistemi

İmza tabanlı bir IDS, ağdaki ağ trafiği paketlerini izler ve bunları imza veritabanları veya bilinen tehditlerin kuralları ile eşleştirir.

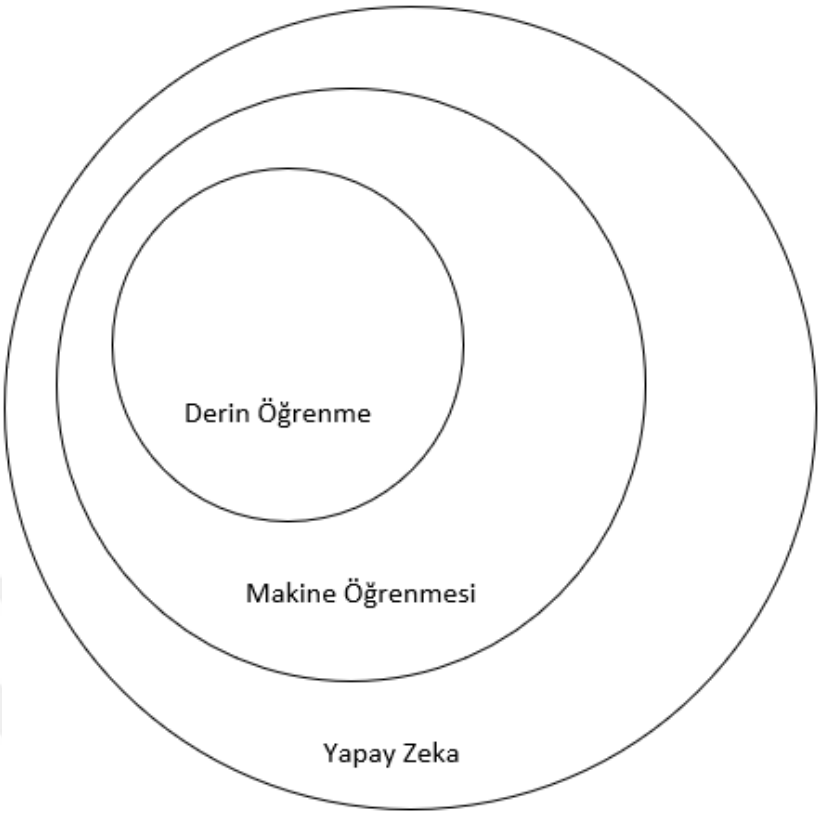
b) Anormallik Tabanlı Saldırı Tespit Sistemi

Anormallik tabanlı bir IDS, ağ trafiğini izler ve normal trafikle karşılaştırır. Normal trafikten herhangi bir sapma, yöneticiyi veya kullanıcıyı anormal bir davranış gösterdiğine ve yüksek hatalı pozitif orana neden olduğu konusunda uyarır.

## 1.2. Derin Öğrenme

Makine öğrenmesi teknolojisi, modern topluma bir çok yönden etkide bulunmaktadır. Web aramalarından, sosyal ağlarda içerik filtrelemeye, e-ticaret sitelerindeki öneriler ile kameralar, akıllı telefonlar gibi tüketici ürünlerinde giderek daha fazla yer almaktadır. Makine öğrenim sistemleri, görüntülerdeki nesneleri tanımlamak, konuşmayı metne dönüştürmek, haber öğelerini, yayınları veya ürünleri kullanıcıların ilgi alanlarıyla eşleştirmek ve alakalı arama sonuçlarını göstermek için kullanılır. Bu uygulamalar derin öğrenme denilen bir teknikler sınıfından faydalanmaktadır (LeCun ve ark., 2015).

Derin öğrenme, makine öğrenmesinin bir alt alanıdır; bu da yapay zekanın alt alanıdır. Bu ilişkinin grafiksel gösterimi Şekil 1.3’ te gösterilmiştir.




Şekil 1.3. Derin öğrenmeyi, makine öğrenmesinin bir alt alanı; makine öğrenmesini de yapay zekanın bir alt alanı olarak tanımlayan Venn şeması (Goodfellow ve ark., 2016).

Yapay zekanın temel amacı, insanların sezgisel ve otomatik olarak yaklaştığı sorunları çözmek için kullanılabilecek bir algoritma ve teknik sağlamaktır. Ancak bu işlem bilgisayarlar için çok zorlayıcıdır. Örneğin, bir görüntünün içeriğini yorumlamak ve anlamak bir insanın çok az çabayla gerçekleştirebileceği bir eylemken, makinelerin başarması için son derece zor bir eylemdir (Rosebrock, 2017).

Yapay zeka geniş ve çeşitli bir çalışma kümesine sahipken, makine öğrenmesi alt alanı özellikle örüntü tanıma ve verilerden öğrenme ile ilgilenme eğilimindedir.

Yapay sinir ağları (YSA), beynin yapısından ve işlevinden ilham alan, verilerden öğrenen ve örüntü tanıma konusunda uzmanlaşmış bir makine öğrenme algoritmaları sınıfıdır. Derin öğrenme de yapay sinir ağları algoritmaları ailesine aittir ve çoğu zaman iki temirm birbirinin yerine kullanılabilir (Rosebrock, 2017).

# 2. KAYNAK BİLDİRİŞLERİ

Bu bölümde, derin öğrenme ile ağ saldırı tespitinde kullanılan bazı teknikler ve yapılan çalışmalar hakkındaki bilgiler derlenip sunulmuştur.

Ağ saldırı tespitinde literatürde birçok metot ve tanımlama bulunmaktadır. Ağ saldırı tespiti üzerine yapılan araştırmalar, Dennig (1987)' in literatürüyle başladı. Dennig, dışarıdan içeriye yapılan saldırılardan, içerideki kişilerin gerçekleştirdiği ihlallere kadar geniş bir yelpazedeki güvenlik ihlallerini tespit etmeyi amaçlayan gerçek zamanlı bir saldırı tespit sistemi modeli ortaya koydu.

Daha sonra, çeşitli teknikler kullanılarak ağ saldırı tespitine yönelik birçok araştırma çalışması yapılmıştır. Ding ve ark. (2009), algılama modelinin yerel minimum seviyeye düşmesini önlemek ve eğitim aşamasını hızlandırmak için modellerinde *Adam* algoritmasını kullanmışlardır. KDD-99 veri seti ile test ve eğitim yapmışlardır. Önerilen bu model ile % 95,57 eğitim doğruluğu elde etmişlerdir. Bununla birlikte, nadir görülen saldırı tiplerinin tespitinde önemli bir gelişme olmamıştır.

Ghanbari ve ark. (2017), dağıtılmış hizmet reddi saldırısı (DDOS) tespitine yönelik bir model önerdiler. Önerilen sinir ağı modeli, evrişimli sinir ağı mimarisi (CNN)' i kullanır. Eğitim ve test işlemlerini, Uygulamalı İnternet Veri Analizi Merkezi (CAIDA) veri seti ile gerçekleştirmişlerdir. Böylelikle % 80.77' lik gerçek pozitif orana ulaşmışlardır. Bununla birlikte, modellerin, gerçek dünyadaki uygulamalar için uygun olmayan denetimli öğrenmeye dayandığı sonucuna varmışlardır.

Liu ve ark. (2017), saldırı tespit sistemi için evrişimli sinir ağı mimarisini uyguladılar. Araştırmalarında KDD-99 veri setini eğitim için kullandılar ve modellerini test ettiler. Bununla birlikte; Sigmoid, tanjant hiperbolik ve düzeltilmiş lineer birim gibi çeşitli aktivasyon yöntemlerini karşılaştırmak için statik bir evrişimsel sinir ağı modeli kullandılar. Çalışma, Sigmoidi, bir aktivasyon fonksiyonu olarak kullanan bir sinir ağı modelinin, diğer aktivasyon fonksiyonlarından daha iyi performans gösterdiğini göstermektedir.

Tran ve ark. (2017), büyük ölçekli ham girdi verilerine sahip evrişimli bir sinir ağının, basit ama minimalist bir mimariden iyi deneysel sonuçlar verdiğini

göstermektedir. Bununla birlikte, çalışmalarında bulunan bu sonucu, çok katmanlı ileri besleme sinir ağı gibi diğer bir sinir ağı mimarisiyle karşılaştırmamışlardır.

Farahnakian ve ark. (2018), KDD-99 veri setiyle otomatik kodlayıcının ağ saldırı tespitindeki performansını değerlendirdiler. Ayrıca, gizli katman sayısının önerilen sistem performansı üzerindeki etkilerini araştırdılar. Deneyleri, % 94.01 doğruluk oranı ile sonuçlanmıştır. Bununla birlikte, araştırmaları, oluşturulan sinir ağının ayrıntılı yapısı hakkında bilgi vermez ve bu araştırmalarında KDD-99 veri setinin yalnızca % 10' unu kullanmışlardır.

Powers ve ark. (2008), IDS için, kendi kendini düzenleyen haritalar (SOM) ağı kullanan bir sistem sundu. Araştırmalarında KDD-99 veri setini kullandılar. Araştırmalarında, bütün saldırı ailelerini kullanmak ve sınıflandırmak yerine, yalnızca genel bir saldırıyı tek tek sınıflandırdılar. Ayrıca, 28 tane özellik ile ağlarını eğittiler.

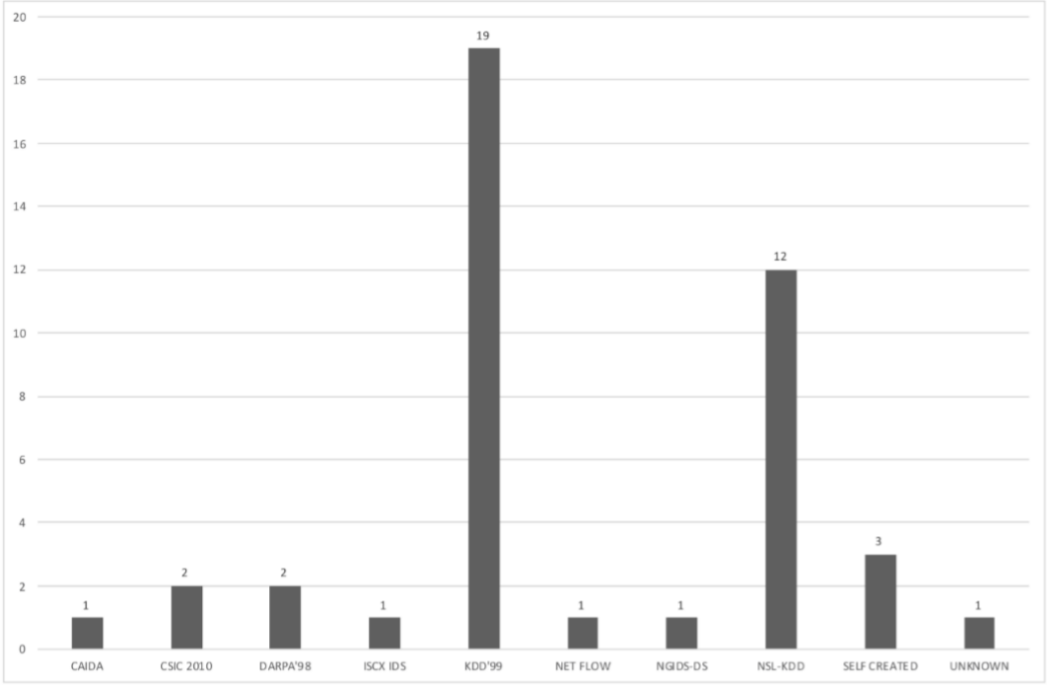
Wang ve ark. (2010), ağ tabanlı saldırı tespit sistemleri için bir başka hibrit yapay sinir ağı yarattılar. Bu araştırmada ileri beslemeli sinir ağlarını tekrarlayan sinir ağlarıyla kendi sistemlerini karşılaştırdılar. Bu karşılaştırma, sinir ağı mimarisi farklılıkları ve parametreler hakkında bilgi vermemektedir.

Kapsamlı bir karşılaştırma çalışması da Kwon ve ark. (2007) tarafından yapılmıştır. Bu çalışmada oto kodlayıcı ağ mimarisiyle, tekrarlayan sinir ağları (RNN)' nı karşılaştırdılar. Bu araştırma, otomatik kodlayıcı algılama doğruluğunun, statik bir test veri setinde RNN ağından daha yüksek olduğunu belirtti.

Chiba ve ark. (2018), sinir ağları mimarisinin karşılaştırılması ile ilgili önemli bir çalışma yapmışlardır. Bu çalışmada, bileşen parametrelerini değiştirerek 48 sinir ağı oluşturmuşlardır ve bunları KDD-99 veri seti ortamında karşılaştırmışlardır. Bununla birlikte, araştırmaları, ileri beslemeli yapay sinir ağları ( FFNN ) ile sınırlıdır.

Bontemps ve ark. (2016), NIDS için özyinelemeli sinir ağı (RNN) oluşturmuşlardır. Ayrıca KDD-99 veri setini kullanmışlardır. Ancak veri setlerini bir zaman serisi olarak hazırlamışlardır. Araştırmaları, zaman serisi tahmin oranına sahip RNN' nin statik veri setine sahip RNN' den daha yüksek olduğunu göstermiştir.

Araştırmış olduğum makalelerde kullanılan veri setlerinin dağılımı Şekil 2.1' de gösterilmektedir. NIDS alanında en yaygın kullanılan veri setinin KDD-99 veri seti olduğu açıkça görülmektedir.




Şekil 2.1. Veri setine dayalı çalışmaların dağılımı.

# 3. MATERYAL VE YÖNTEM

Bu tez kapsamında yapılan çalışmada, birçok yöntem ve araçtan yararlanılmıştır. Çalışmada yapılan uygulamada KDD-99 veri seti kullanılmıştır. Derin öğrenme algoritmalarında, python dili ile birlikte gelen keras kütüphaneleri kullanılmıştır. Python dili ile yazılan derin öğrenme algoritmalarının çalıştırılması UBUNTU işletim sisteminde gerçekleştirilmiştir.

## 3.1. KDD-99 Veri Seti

KDD-99 veri seti, algoritmaları eğitmek için en çok kullanılan veri setidir (Ravipati ve Abualkibash, 2019). Bu veri seti Stolfo ve ark. (2000) tarafından oluşturulmuştur. Bu veri seti DARPA'98 IDS değerlendirme programında toplanan verilere dayanılarak oluşturulmuştur. DARPA'98 veri seti, her biri 100 bayt olmak üzere yaklaşık 5 milyon bağlantı kaydının, 7 haftalık ağ trafiğinin yaklaşık 4 gigabayt sıkıştırılmış ham (ikili) tcpdump verisidir. İki haftalık test verisi yaklaşık 2 milyon bağlantı kaydından oluşmaktadır. KDD eğitim veri seti, her biri 41 nitelik içeren ve belirli bir saldırı türüne sahip, normal bağlantı veya anormal bağlantı olarak etiketlenen yaklaşık 4.900.000 bağlantıdan oluşmaktadır (Tavallaee ve ark., 2009).

Veri setinde bulunan saldırı türleri aşağıdaki dört kategoriden birine dâhil olmaktadır:

a) Denial of Service Attack (DoS):

Bu saldırı türü, saldırganın bellek kaynaklarını aşırı tüketerek bu kaynakları meşgul etmesi veya dolu hale getirmesi ile normal kullanıcıların makineye erişmesini tamamen engelleyen saldırı türüdür.

b) User to Root Attack (U2R):

Bu saldırı türünde saldırgan, normal bir kullanıcı hesabıyla (bir sözlük saldırısıyla, paket koklamayla veya sosyal mühendislik yöntemleriyle ele geçirilmiş olabilir.) erişim sağlayıp, sistemde bulunan güvenlik açığından yönetici yetkisi elde ettiği bir istismar türüdür.

c) Remote to Local Attack (R2L):

Bu saldırı türü, ağ üzerinden bir makineye paket gönderebilen ancak bu makinede hesabı olmayan saldırganın makinenin kullanıcısı olarak erişim sağlamak için bazı güvenlik açıklarından yararlandığı bir saldırı türüdür.

d) Probing Attack:

Bu saldırı türü, ağdaki güvenlik kontrollerini aşmak için bir bilgisayar ağı hakkında bilgi toplama saldırısıdır.

Veri setinde bulunan test verileri ile eğitim verilerinin aynı olasılık dağılımına sahip olmadığının belirtilmesi önemlidir. Ayrıca bu test verileri, eğitim verilerinde bulunmayan belirli saldırı türlerini içerir. Bazı güvenlik uzmanları, yeni saldırıların çoğunun bilinen saldırıların başka bir sürümü olduğuna ve bilinen saldırı imzalarının yeni saldırı türlerini yakalamak için yeterli olabileceğine inanmaktadır (Tavallaee ve ark., 2009).

KDD'99 özelikleri üç grupta sınıflandırılabilir:

**1.** *Temel Özellikler:* Bu özellikler, bir TCP/IP bağlantısından çıkartılabilen tüm özellikleri kapsar.

**2.** *Trafik Özellikleri:* Bu özellikler kendi içinde iki gruba ayrılır:

a) Aynı Host Özellikleri: Bu özellik, geçerli bağlantı zamanına kadar olan son 2 saniyede, bu bağlantıyla aynı hedef makineye yapılan bağlantıların protokolü, servisi vb. İstatistikleri inceler.

b) Aynı Servis Özellikleri: Bu özellik, geçerli bağlantı zamanına kadar olan son 2 saniyedeki aynı hizmete sahip olan bağlantıları inceler.

Yukarıda bahsedilen iki tür trafik özelliğinin zamana dayalı olduğu belirtiliyor. Ancak, ana bilgisayarları (veya bağlantı noktalarını) 2 saniyeden daha uzun bir zaman aralığı kullanarak (örneğin her dakika bir kez) tarayan birkaç yavaş *Probing* saldırısı vardır (Tavallaee ve ark., 2009).

Sonuç olarak, bu saldırılar 2 saniyelik zaman aralığına sahip saldırılar oluşturmaz. Bu sorunu çözmek için, "aynı host" ve "aynı servis" özellikleri yeniden hesaplanır. Böylelikle 2 saniyelik bir zaman aralığı yerine 100 bağlantının bağlantı niteliklerine dayanarak hesaplama yapılmış olur. Bu özelliklere bağlantı tabanlı trafik özellikleri denir.

**3.** *İçerik Özellikleri:* R2L ve U2R saldırıları, DoS ve Probing saldırılarının çoğundan farklı olarak sık sık sıralı düzenlere sahip değildir. Bunun nedeni, DoS ve Probing saldırılarının çok kısa bir süre içinde bazı hostlar ile çok fazla bağlantı kurmasıdır; ancak R2L ve U2R saldırıları paketlerin veri bölümlerine gömülüdür ve normalde sadece tek bir bağlantı içerir. Bu tür saldırıları tespit etmek ve veri bölümünde şüpheli davranışı arayabilmemiz için başarısız oturum açma sayısı gibi özelliklere ihtiyacımız vardır. Bu özelliklere içerik özellikleri denir (Tavallaee ve ark., 2009).

Dhanabal ve ark. (2015), KDD veri setinde bulunan 41 niteliği üç grup halinde Çizelge 3.1, Çizelge 3.2 ve Çizelge 3.3 'teki gibi gruplandırmışlardır.

Çizelge 3.1. Ağ bağlantısındaki vektörlerin temel nitelikleri.

| **Numara** | **Nitelik Adı** | **Açıklama** | **Örnek Veri** |
|---|---|---|---|
| 1 | Duration | Bağlantı süresi. | 0 |
| 2 | Protocol_type | Bağlantıda kullanılan protokol. | Tcp |
| 3 | Service | Servisin kullandığı hedef ağ. | ftp_data |
| 4 | Flag | Bağlantının durumu – Normal bağlantı veya hatalı bağlantı. | SF |
| 5 | Src_bytes | Tek bir bağlantıda, kaynaktan hedefe gönderilen byte sayısı. | 491 |
| 6 | Dst_bytes | Tek bir bağlantıda, hedeften kaynağa gönderilen byte sayısı. | 0 |
| 7 | Land | Kaynak ve hedef IP adresleri ve port numaraları aynı ise bu değer 1, değilse 0 olur. | 0 |
| 8 | Wrong_fragment | Bağlantıdaki toplam hatalı parça sayısı. | 0 |
| 9 | Urgent | Bağlantıdaki toplam *urgent* işaretli paketlerin sayısı. *Urgent* paketler, *Urgent* biti aktif olan paketlerdir. | 0 |

Çizelge 3.2. Ağ bağlantısındaki vektörlerin içeriği ile ilgili nitelikleri.

| **Numara** | **Nitelik Adı** | **Açıklama** | **Örnek Veri** |
|---|---|---|---|
| 10 | Hot | İçerikteki "hot" göstergelerinin sayısı. Örneğin; Bir sisteme giriş. | 0 |
| 11 | Num_failed_logins | Başarısız giriş sayısı. | 0 |
| 12 | Logged_in | Giriş yapma durumu: Eğer giriş başarılıysa 1; değilse değer 0 olur. | 0 |
| 13 | Num_compromised | Tehlikeli koşulların sayısı. | 0 |
| 14 | Root_shell | Eğer bağlantıda root sheel bulunuyorsa 1; bulunmuyorsa değer 0 olur. | 0 |
| 15 | Su_attempted | Eğer "su root" komutu denenmiş ya da kullanılmışsa değer 1, aksi takdirde değer 0 olur. | 0 |
| 16 | Num_root | Bağlantıda "root" olarak gerçekleştirilen erişim sayısı ve işlem sayısıdır. | 0 |
| 17 | Num_file_creations | Bağlantıdaki dosya oluşturma işlemlerinin sayısıdır. | 0 |
| 18 | Num_shells | *Shell* istem sayısıdır. | 0 |
| 19 | Num_access_files | Erişim kontrolü dosyalarındaki işlem sayısıdır. | 0 |
| 20 | Num_outbound_cmds | Bir ftp oturumunda giden komut sayısıdır. | 0 |
| 21 | Is_hot_login | Giriş, "hot" listesine aitse, yani root veya admin ise 1 değerini alır; aksi takdirde 0 değerini alır. | 0 |
| 22 | Is_guest_login | Giriş, "guest" girişi ise 1 değerini; aksi takdirde 0 değerini alır. | 0 |

Çizelge 3.3. Her ağ bağlantısı vektörünün zamanla ilişkili trafik nitelikleri.

| Numara | Nitelik Adı | Açıklama | Örnek Veri |
|---|---|---|---|
| 23 | Count | Son iki saniyede, geçerli bağlantının hedef hostu ile aynı hedef hosta olan bağlantı sayısı. | 2 |
| 24 | Srv_count | Son iki saniyede, geçerli bağlantının servisiyle (portuyla) aynı servise (porta) olan bağlantı sayısı. | 2 |
| 25 | Serror_rate | 23. Nitelikte toplanan bağlantılar arasında, s0, s1, s2 veya s3 bayrağının etkin olduğu bağlantıların yüzdesi. | 0 |
| 26 | Srv_serror_rate | 24. Nitelikte toplanan bağlantılar arasında, s0, s1, s2 veya s3 bayrağının etkin olduğu bağlantıların yüzdesi. | 0 |
| 27 | Rerror_rate | 23. Nitelikte toplanan bağlantılar arasında, REJ bayrağının etkin olduğu bağlantıların yüzdesi. | 0 |
| 28 | Srv_rerror_rate | 24. Nitelikte toplanan bağlantılar arasında, REJ bayrağının etkin olduğu bağlantıların yüzdesi. | 0 |
| 29 | Same_srv_rate | 23. Nitelikte toplanan bağlantılar arasında, aynı servise olan bağlantıların yüzdesi. | 1 |
| 30 | Diff_srv_rate | 23. Nitelikte toplanan bağlantılar arasında, farklı servislere olan bağlantıların yüzdesi. | 0 |
| 31 | Srv_diff_host_rate | 24. Nitelikte toplanan bağlantılar arasında, farklı hedef hostlara yapılan bağlantıların yüzdesi. | 0 |

Çizelge 3.4. Ağ bağlantısındaki vektörlerin host tabanlı trafik nitelikleri.

| Numara | Nitelik Adı | Açıklama | Örnek Veri |
|---|---|---|---|
| 32 | Dst_host_count | Aynı ip adreslerine sahip hedef hostlara yapılan bağlantıların sayısı. | 150 |
| 33 | Dst_host_srv_count | Aynı port numarasına sahip bağlantıların sayısı. | 25 |
| 34 | Dst_host_same_srv_rate | 32. Nitelikte toplanan bağlantılar arasında aynı servise olan bağlantıların yüzdesi. | 0.17 |
| 35 | Dst_host_diff_srv_rate | 32. Nitelikte toplanan bağlantılar arasında farklı servislere olan bağlantıların yüzdesi. | 0.03 |
| 36 | Dst_host_same_src_port _rate | 33.Nitelikte toplanan bağlantılar arasında aynı kaynak portuna olan bağlantıların yüzdesi. | 0.17 |
| 37 | Dst_host_srv_ diff_host_rate | 33. Nitelikte toplanan bağlantılar arasında, farklı hedef makinelere yapılan bağlantıların yüzdesi. | 0 |
| 38 | Dst_host_serror_rate | 32. Nitelikte toplanan bağlantılar arasında s0, s1, s2, s3 bayrağı etkin olan bağlantıların yüzdesi. | 0 |
| 39 | Dst_host_srv_serror_rate | 33. Nitelikte toplanan bağlantılar arasında, s0, s1, s2 veya s3 bayrağı etkin olan bağlantıların yüzdesi. | 0 |
| 40 | Dst_host_rerror_rate | 32. Nitelikte toplanan bağlantılar arasında, REJ bayrağını etkinleştiren bağlantıların yüzdesi. | 0.05 |
| 41 | Dst_host_srv_rerror_rate | 33. Nitelikte toplanan bağlantılar arasında, REJ bayrağı etkin olan bağlantıların yüzdesi. | 0 |

## 3.2. Makine Öğrenmesi

Makine öğrenmesi, bilgisayarların öğrenmesini sağlayan algoritmaların tasarımı ve analizi ile ilgilenen ve hızla gelişen bir alandır. Makine öğrenmesi resimlerden nesnelerin otomatik tespiti, ses tanısı, veri tabanlarından bilgi keşfi ve tahminleyici analitik gibi birçok alanda kullanılmaktadır (Alpaydın, 2016).

Aileler çocuklarına kedi ve köpek arasındaki farkı öğretmek için onlara asla kedi ve köpeğin bilimsel tanımlarını vermezler. Bunun yerine çocuklar, bu canlıları gördüklerinde ailelerinden aldıkları, bu köpektir veya bu kedidir bilgileriyle eğitilirler. Yeni bir kedi veya köpekle karşılaştıklarında bunları doğru tanımlamalarını doğru bir şekilde yapabiliyorlarsa çocuk öğrenmiş demektir. Tıpkı insanlar gibi bazı görevleri gerçekleştirmek için bilgisayarlar da öğrenebilmektedir (Mitchell, 1997).

Makine öğrenmesi, veriden bilgi çıkarmayla ilgilenen istatistik, yapay zekâ ve bilgisayar bilimlerinin kesişiminde bulunan bir alandır ve *tahminleyici analitik* veya *istatistiksel öğrenme* olarak da isimlendirilmektedir. Makine öğrenmesi metotlarının uygulamaları artık günlük hayatın hemen her köşesinde yer almaya başlamıştır. Facebook, Amazon veya Netflix gibi karmaşık web siteleri başarılarını kullandıkları makine öğrenmesi algoritmalarına borçludur. Ticari uygulamaların dışında makine öğrenmesi veriye dayanan araştırmalarda da giderek artan bir etkiye sahiptir.

Mitchell, makine öğrenmesini, bir bilgisayar programını da kapsayacak biçimde bir işteki performansın deneyimle artması olarak tanımlamıştır. Daha açık bir tanımla; bir bilgisayar programı eğer *P* ile ölçülen *T*' de bulunan işlerdeki performansı *E* deneyimine göre artıyorsa *P* performans ölçüsü ve *T* işlerinin bazı sınıfları konusunda *E* deneyiminden öğrenir (Mitchell, 1997).

Tanımda sözü geçen;

*T* görevi: Herhangi bir dilden başka bir dile çeviri,

*P* performans ölçüsü: Yapılan çevirilerin doğruluk yüzdesi,

*E* deneyimi: Sistem üzerinde yapılan tüm çeviriler olarak düşünülebilir.

Makine öğrenmesi, performans optimizasyonu için örnek veri ya da geçmiş deneyimleri kullanarak bilgisayar programlamaktır (Alpaydın, 2016).

Makine öğrenmesi algoritmaları

- Metin veya belge sınıflandırma
- Doğal dil işleme
- Ses tanısı
- Optik karakter tanımlama
- Hesaplamalı biyoloji
- Sahtekârlık tespiti
- Oyun
- Sürücüsüz araç kontrolü
- Tıbbi tanı
- Öneri sistemleri

gibi alanlarda bir çok başarılı uygulamaya sahiptir (Mohri ve ark., 2012).

Makine öğrenmesi problemleri genel olarak 5 ana başlıkta toplanabilir;

1. Sınıflandırma
2. Regresyon
3. Ranking
4. Kümeleme
5. Boyut azaltma veya manifold öğrenmesi. (Mitchell, 1997)

### 3.2.1. Bazı tanım ve terminolojiler

Bu kısımda tez boyunca kullanılacak bazı terimlerin tanımları verilecektir.

Örnek: Makine öğrenmesi için kullanılacak veri setindeki gözlemler ve nesneler,

Nitelik: Örneği en iyi açıkladığı düşünülen özellikler,

Etiket: Örneklere atanan değer veya sınıflar,

Eğitim verisi: Öğrenme algoritmasının eğitiminde kullanılan örneklerin oluşturduğu veri seti,

Test verisi: Öğrenme algoritmasının performansını değerlendirmek için kullanılan veri seti,

Hata fonksiyonu: Tahmin edilen etiket ile gerçek etiket arasındaki farkı veya kaybı ölçen fonksiyon,

Hipotez kümesi: Nitelikleri etiketlere dönüştüren fonksiyon ailesi (Alpaydın, 2016).

### 3.2.2. Makine öğrenmesi yaşam döngüsü

Makine öğrenmesi algoritmaları eğitim verisinin türü, eğitim verisinin aldığı metot ve bunun sırası ile öğrenme algoritmasının değerlendirilmesinde kullanılan test verisine göre farklı senaryolara ayrılabilir (Mohri ve ark., 2012).

Bu senaryoları şu şekilde belirlemiştir;

1. Denetimli Öğrenme: Öğrenme algoritması etikete sahip bir eğitim verisi alır ve etiketi bilinmeyen veriler için tahmin yapar. Bu tip öğrenme makine öğrenmesinde en çok kullanılan öğrenme senaryosudur. Sınıflandırma, regresyon ve ranking problemleri denetimli öğrenme türündeki problemlere dahildir.
2. Denetimsiz Öğrenme: Öğrenme algoritması etiketsiz bir eğitim verisine sahiptir ve bilinmeyen noktalar için tahmin yapar. Bu tip öğrenmede etiketsiz verilere sahip olunduğundan öğrenme algoritmasının performans değerlendirmesini niceliksel olarak yapmak zordur. Kümeleme ve boyut azaltma algoritmaları denetimsiz öğrenme algoritmalarıdır.
3. Yarı-denetimli Öğrenme: Öğrenme algoritması hem etiketli eğitim verisini hem de etiketsiz eğitim verisi ile çalışır ve bilinmeyen noktalar için tahmin yapmaya çalışır. Yarı-denetimli öğrenme genel olarak verilerin kolay etiketlerin zor elde edildiği problemlerde kullanılır.
4. Transdüktif Çıkarsama: Yarı- denetimli öğrenmede olduğu gibi öğrenme algoritması etiketsiz eğitim verisi ile birlikte etiketli eğitim verisini alır. Fakat transdüktif çıkarsamanın amacı belli test örneklerinin etiketlerini tahmin etmektir.
5. Çevrimiçi Öğrenme: Daha önceki senaryoların tersine çevrimiçi öğrenme eğitim ve test aşamalarının karıştırıldığı bir öğrenme türüdür. Her bir aşamada öğrenme algoritması bir etiketsiz nokta alır tahmin yapar bu tahmini gerçek etiket ile karşılaştırır ve bir hata hesaplar. Çevrimiçi öğrenmede amaç kümülatif hatayı minimize etmektir.
6. Takviyeli Öğrenme: Takviyeli öğrenmede de test ve eğitim aşamaları karıştırılır. Bilgi toplamak için öğrenme algoritması aktif bir şekilde çevreyle etkileşim içindedir ve bazı durumlarda çevreyi de etkiler ve her bir eylem için bir karşılık alır.

7. Aktif Öğrenme: Öğrenme algoritması uyarlamalı veya interaktif şekilde eğitim örneklerini toplar (Mohri ve ark., 2012).

Bu çalışma kapsamında denetimli öğrenme algoritmaları ele alınacağından, aşağıdaki örnek üzerinde bir denetimli öğrenme algoritması açıklanmıştır.

Bir bölgedeki dairelere ait büyüklük ($m^2$) ve fiyatların (₺) bilgisinin kaydedildiği bir veri setine sahip olunduğu düşünülsün.

Çizelge 3.5. Makine öğrenmesi örnek veri (Ng, 2000)

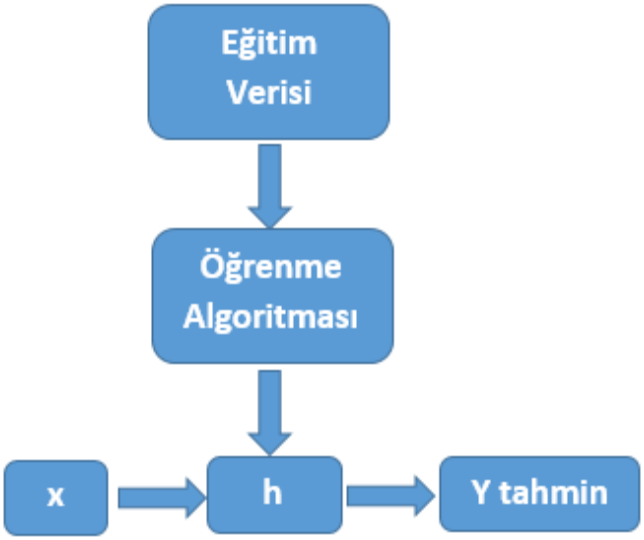
| Büyüklük ($m^2$) | Fiyat (TL) |
|---|---|
| 120 | 170.000 |
| 130 | 185.000 |
| 145 | 225.000 |
| 180 | 255.000 |
| 150 | 200.000 |
| . | . |
| . | . |
| . | . |

Bu veri setini baz alarak aynı bölgedeki dairelerin fiyatları büyüklüklerinin bir fonksiyonu olarak ifade edilebilir mi? problemi klasik bir denetimli öğrenme problemidir. Her bir örnek $x^{(i)}$ ile gösterilir. Bu örnek için $x^{(i)}$ $i$. dairenin büyüklüğüne karşılık gelmektedir. Etiketler $y^{(i)}$ ile gösterilecektir. $\left(x^{(i)}, y^{(i)}\right)$ çiftine eğitim örneği, $m$ tane eğitim örneğinin oluşturduğu $\left\{\left(x^{(i)}, y^{(i)}\right) \middle| i = 1,2, \dots, m\right\}$ kümesine de eğitim verisi denilmektedir. Bu tez kapsamında örneklerin uzayı $X$ ile, etiket değerlerinin oluşturduğu uzay $Y$ ile gösterilecektir.

Denetimli öğrenme problemi matematiksel olarak;

$$h: X \longrightarrow Y \tag{3.1}$$

şeklinde gösterilebilir. Burada $h$ fonksiyonu hipotez olarak isimlendirilir. $h(x)$ değeri de $x$ örneğine karşılık gelen tahmin değeridir. Denetimli öğrenmede amaç $h(x)$ değerleri ile $x$'in gerçek etiketleri arasındaki farkların toplamını minimum yapacak $h$ hipotezinin belirlenmesidir (Şekil 3.1).




Şekil 3.1. Makine öğrenmesi yaşam döngüsü (Mitchell, 1997).

Eğer etiket değişkeni sürekli ise bu tip bir probleme regresyon, kategorik ise problem sınıflandırma problemi olarak isimlendirilir.

Birçok problemde örnekler birden çok nitelik ile gösterilir. Daire örneğinde veri seti;

Çizelge 3.6. Makine öğrenmesi çok değişkenli örnek veri çizelgesi (Ng, 2000)

| Büyüklük ($m^2$) | Oda Sayısı | Fiyat (TL) |
|---|---|---|
| 120 | 3 | 170.000 |
| 130 | 3 | 185.000 |
| 145 | 4 | 225.000 |
| 180 | 4 | 255.000 |
| . | . | . |
| . | . | . |

Olarak ele alınırsa bu durumda bir örnek $\left(x_1^{(i)}, x_2^{(i)}\right)$ çifti ile gösterilecektir. Bu tip bir problem için denetimli öğrenme algoritması uygulandığı zaman ilk yapılması gereken bir $h$ hipotezinin nasıl tanımlanacağıdır. Bu problem için bir dairenin fiyatı büyüklük ve oda sayısının lineer bir fonksiyonu olarak ifade edilsin.

Bu durumda $h$;

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 \tag{3.2}$$

olarak ifade edilebilir. Burada $\theta_i$ ler eğitim verisinden elde edilecek parametrelerdir. Eğitim verisindeki örnekler kullanılarak tahmin edilen $h(x)$ ve $y$ değerleri arasındaki hata minimize edilmeye çalışılır.

Formel bir şekilde hata fonksiyonu

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 \quad (3.3)$$

şeklinde tanımlanır. (Ng, 2000)

Öğrenme algoritmasında amaç $J(\theta)$ yı minimum yapacak $\theta$ nın bulunmasıdır. En dik iniş algoritması bu amaç için kullanılabilir. En dik iniş algoritması bir $\theta$ başlangıç değeriyle başlar ve $j = 0,1,2,\ldots,n$ için;

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta) \quad (3.4)$$

döngüsünü tekrarlar.

Burada $\alpha$ öğrenme hızı olarak isimlendirilmektedir. Bu algoritmanın uygulanması için yukarıdaki denklemin sağ tarafındaki kısmi türevin belirlenmesine ihtiyaç vardır.

Kolaylık olması adına elde bir adet örnek olduğu düşünülsün dolayısıyla $J$ deki toplam sembolü göz ardı edilebilir (Ng, 2000).

$$\frac{\partial}{\partial \theta_j} J(\theta) = \frac{\partial}{\partial \theta_j}\frac{1}{2}(h_\theta(x) - y)^2$$

$$= 2.\frac{1}{2}(h_\theta(x) - y)\frac{\partial}{\partial \theta_j}(h_\theta(x) - y)$$

$$= (h_\theta(x) - y)\frac{\partial}{\partial \theta_j}\left(\sum_{i=0}^{n}\theta_i x_i - y\right)$$

$$= (h_\theta(x) - y)x_j$$

Bir eğitim örneği için yukarıdaki sonuç şu güncelleme kuralını vermektedir.

$$\theta_j := \theta_j + \alpha\left(y^{(i)} - h_\theta(x^{(i)})\right)x_j{}^{(i)} \quad (3.5)$$

Bu kurala ortalama kare güncelleme kuralı denilmektedir (Ng, 2000). Bu sonuç tek bir eğitim örneği için geçerlidir. Birden fazla örnek için bu kuralı kullanmak için iki tane yol bulunmaktadır. Bunlardan birincisi

$$\theta_j \coloneqq \theta_j + \alpha \sum_{i=1}^{m} \left(y^{(i)} - h_\theta\left(x^{(i)}\right)\right) x_j{}^{(i)} \qquad her\ j\ için$$

dir. Toplamın içindeki büyüklük $\frac{\partial}{\partial \theta_j} J(\theta)$ türevine karşılık gelmektedir. Bu güncelleme kuralı $J$ hata fonksiyonunda en dik iniş algoritması olarak bilinmektedir. Bu metot tüm eğitim verisindeki her bir örneği tek tek incelediğinden yığın en dik iniş algoritması (batch gradient descent) olarak ta isimlendirilir.

*tekrarla {*
1 *den m'e kadar {*
$\theta_j \coloneqq \theta_j + \alpha \left(y^{(i)} - h_\theta\left(x^{(i)}\right)\right) x_j{}^{(i)}$ (3.6)
*her j için*
}
}

Bu algoritmada veri seti üzerinden tekrarlı bir şekilde geçilir ve her eğitim örneği ile karşılaştığımızda bu eğitim verisinin hatasının gradiyentine göre parametreler güncellenir. Buna stokastik en dik iniş algoritması (stochastic gradient descent) denir. Eğitim verisinin büyük olduğu durumlarda stokastik en dik iniş algoritması yığın en dik iniş algoritmasına tercih edilir (Ng, 2000).
Olasılıksal yorum etiketlerin ve niteliklerin;

$$y^{(i)} = \theta^T x^{(i)} + \varepsilon^{(i)} \qquad (3.7)$$

modeli ile ilişkili olduğu varsayılsın. Burada $\varepsilon^{(i)}$ gözlenmeyen niteliklere karşılık gelen hata terimini ifade etmektedir. $\varepsilon^{(i)}$ lerin 0 ortalamalı ve $\sigma^2$ varyanslı Normal dağılıma

sahip oldukları ve bağımsız ve özdeşçe dağılmış oldukları varsayılır. Bu varsayım $\varepsilon^{(i)} \sim N(0, \sigma^2)$ ile gösterilir (Ng, 2000). Yani $\varepsilon^{(i)}$ lerin yoğunluğu

$p\left(\varepsilon^{(i)}\right) = \frac{1}{\sqrt{2\pi\sigma}} \exp\left(-\frac{(\varepsilon^{(i)})^2}{2\sigma^2}\right)$ ile verilir.

Buradan

$$p\left(y^{(i)}\middle|x^{(i)};\theta\right) = \frac{1}{\sqrt{2\pi\sigma}} exp\left(-\frac{\left(y^{(i)}-\theta^T x^{(i)}\right)^2}{2\sigma^2}\right) \quad (3.8)$$

eşitliği elde edilebilir. $p\left(y^{(i)}\middle|x^{(i)};\theta\right)$ gösterimi $x^{(i)}$ verildiğinde $y^{(i)}$ nin dağılımı $\theta$ parametresine bağlıdır anlamına gelmektedir. $\theta$ bir rasgele değişken olmadığından bir koşul belirtmez. $p\left(y^{(i)}\middle|x^{(i)};\theta\right)$ büyüklüğü $x$ ve $y$ ye bağlı bir fonksiyondur. Bu fonksiyonu $\theta$ nın bir fonksiyonu olarak ifade etmek için $\varepsilon^{(i)}$ lerin bağımsız ve özdeş oldukları varsayımı da kullanılarak;

$$L(\theta) = \prod_{i=1}^{m} p\left(y^{(i)}\middle|x^{(i)};\theta\right)$$
$$= \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi\sigma}} exp\left(-\frac{\left(y^{(i)}-\theta^T x^{(i)}\right)^2}{2\sigma^2}\right) \quad (3.9)$$

şeklinde $L$ fonksiyonu tanımlanabilir. Bu fonksiyon olabilirlik fonksiyonu olarak isimlendirilir. $L(\theta)$ yı maksimum yapan $\theta$ değerinin bulunması maksimum olabilirlik metodu olarak bilinmektedir. $L(\theta)$ yı maksimum yapan $\theta$ değerinin yerine $L(\theta)$ nın sürekli artan bir fonksiyonun maksimum değeri de bulunabilir. Bunun için log olabilirlik fonksiyonu;

$$l(\theta) = logL(\theta)$$
$$= log \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi\sigma}} exp\left(-\frac{\left(y^{(i)} - \theta^T x^{(i)}\right)^2}{2\sigma^2}\right)$$

$$= \sum_{i=1}^{m} log \frac{1}{\sqrt{2\pi\sigma}} exp\left(-\frac{\left(y^{(i)} - \theta^T x^{(i)}\right)^2}{2\sigma^2}\right)$$

$$= mlog \frac{1}{\sqrt{2\pi\sigma}} - \frac{1}{\sigma^2} \cdot \frac{1}{2} \sum_{i=1}^{m} \left(y^{(i)} - \theta^T x^{(i)}\right)^2 \quad (3.10)$$

şeklinde tanımlanır. Yani $l(\theta)$ yı maksimum yapan değer $\sum_{i=1}^{m} \left(y^{(i)} - \theta^T x^{(i)}\right)^2$ değerini minimum yapacaktır (Ng, 2000).

Bir problemin çözümüne makine öğrenmesi ile ulaşılmaya çalışılırken belli adımların uygulanması gerekmektedir. Literatürde izlenmesi gereken adımlar için bir çok standart belirlenmesine rağmen bu aşamalar genel olarak birbirine oldukça benzemektedir.

Veri madenciliği için birçok uzmanın bir araya gelip bu adımları standartlaştırmaya çalıştığı Veri Madenciliği için Çarpraz Endüstri Standard Süreç Modeli (Cross-Industry Standard Process for Data Mining - CRISP) makine öğrenmesi algoritmaları için de kullanılabilir. (Shearer, 2000)

Bu modelde problemin çözümüne ulaşmak için gerekli adımlar

1. Problemin tanımlanması
2. Veriyi anlama
3. Veri ön işleme
4. Modelleme
5. Model Değerlendirmesi ve Seçimi
6. Uygulamaya Geçilmesi

olarak belirlenmiştir (Kartal, 2015; Brownlee, 2014).

#### 3.2.2.1. Problemin tanımlanması

Problem tanımlanması aşamasında makine öğrenmesi algoritmaları yardımıyla çözülmesi amaçlanan problemin açıkça ifade edilebilmesi gerekir. Problem Mitchell tarafından verilen tanıma tam olarak uydurulmalıdır.

#### 3.2.2.2. Veriyi anlama

Makinenin öğrenmek için kullanacağı veri, ele alınan probleme uygun bir biçimde oluşturulmalıdır. Toplanan veri düzenli (veritabanı formatı gibi), ya da düzensiz (Twitter'dan elde edilen twitler gibi) olabilir. Veri toplama aşamasında farklı kaynaklardan yararlanılabilmektedir. Bunlardan ilki internette yer alan hazır veri setleridir. Bu veri setleri, erişim ve kullanım kolaylığı bakımından makine öğrenmesi çalışmalarına zaman kazanma açısından önemli avantaj sağlamaktadır. Nitekim verinin analizler için hazırlanması belki de makine öğrenmesi sürecinin en zor ve zaman alıcı aşaması olarak kabul edilebilir. Bazı çalışmalarda da gerekli olan veri seti araştırmacılar tarafından internetten özel bazı araçlarla elde edilirken, kimi çalışmalarda da internetteki hazır veri setlerinden birini kullanmak yerine, ele alınan probleme uygun özgün veri setleri problem için özellikle oluşturulmaktadır. Veriye uygun modelin kurulmasından önce mevcut verinin iyi anlaşılması ve iyi analiz edilmesi gerekmektedir. Veri seti elde edildikten sonra veri hakkında ön fikir edinilmesi için verinin normalleştirilmesi gibi bazı basit istatistiksel metotlardan yararlanılabilir (Kartal, 2015; Maurizio, 2011).

#### 3.2.2.3. Veriyi ön işleme

Veri analizine geçmeden önce verinin analizler için hazır hale getirilmesi gerekmektedir. Nitekim pratik hayatta veriler çoğu zaman eksik, tutarsız veya hatalı olabilmektedir. Literatürde veri ön-işleme (data preprocessing) olarak adlandırılan veri hazırlama süreci Kartal (2015) ve Han ve Kamber (2006) tarafından veri temizleme, veri bütünleştirme, veri seçme, veri dönüştürme, veri madenciliği, desen değerlendirmesi ve bilgi sunumu başlıkları altında ele almıştır. Veri hazırlama sürecinde gerçekleştirilecek işlemlerin herhangi bir standardı olmayıp, kullanılan veri setine göre değişmektedir (Kartal, 2015; Maurizio, 2011).

#### 3.2.2.4. Modelleme

Bir makine öğrenmesi algoritmasında model, girdiyi çıktıya dönüştüren matematiksel bir bağıntı veya veri içerisinde saklı bulunan örüntüdür (Flach, 2012).

Makine öğrenmesi algoritması, oluşturulabilecek modeller arasından en iyisini seçmeye çalışacaktır.

Bu tez kapsamında ele alınacak modelleri belirlemek için yapay sinir ağları algoritmaları kullanılacaktır.

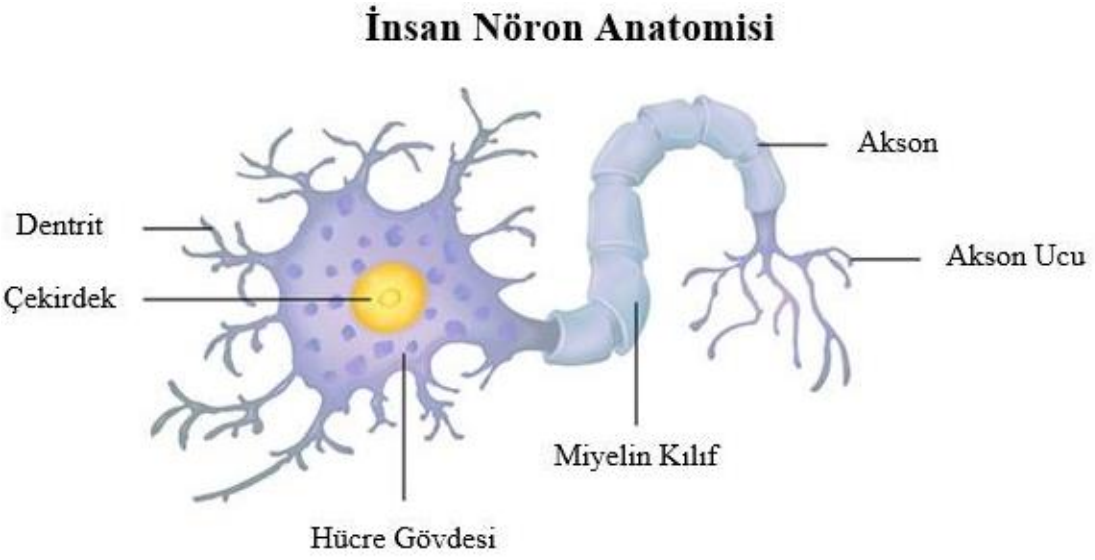
#### 3.2.2.4.1. Yapay sinir ağları

Yapay sinir ağları derin öğrenme sistemlerinin yapı taşlarıdır. Tezin bu bölümünde mimari, düğüm tipleri ve ağları eğitmek için kullanılan algoritmalar dâhil olmak üzere sinir ağlarının temelleri açıklanacaktır.

Zekâ, örüntü tanıma ve nesne algılamasını içeren birçok işin otomatikleştirilmesi oldukça zordur ancak bunlar hayvanlar ve küçük çocuklar tarafından kolayca ve doğal olarak yapılabilmektedir. Örneğin, köpekler sahiplerini ve yabancı kişileri nasıl kolayca birbirinden ayırt edebilmektedir? Küçük bir çocuk, bir okul otobüsü ile bir transit otobüs arasındaki farkı tanımayı nasıl öğrenir? Beyinlerimiz, bilinçaltında biz bile fark etmeden her gün karmaşık örüntü tanıma görevlerini nasıl gerçekleştiriyor? Cevap kendi bedenlerimizde yatar. İnsan vücudu sinir sistemine bağlı biyolojik sinir ağları içerir ve bu ağlar çok sayıda birbirine bağlı nöronlardan (sinir hücreleri) oluşur. "Yapay Sinir Ağı" (YSA) sinir sistemimizdeki sinir bağlantılarını taklit etmeye çalışan (veya en azından ilham alan) bir hesaplama sistemidir. Yapay sinir ağları aynı zamanda "sinir ağları" veya "yapay sinir sistemleri" olarak da adlandırılır.

Bir sistemin YSA olarak kabul edilmesi için, grafikteki her bir düğümün basit bir hesaplama yaptığı, etiketli, yönlendirilmiş bir grafik yapısı içermesi gerekir. Graf teorisinden, yönlendirilmiş bir grafın, düğüm çiftlerini birbirine bağlayan bir dizi düğümden (yani, köşeler) ve bir dizi bağlantıdan (yani, kenarlar) oluştuğu bilinmektedir. Şekil 3.3' te böyle bir YSA grafiğinin bir örneği görülebilir. Her düğüm basit bir hesaplama yapar. Her bağlantı daha sonra bir sinyalin (yani hesaplamanın çıktısı) bir düğümden diğerine, sinyalin büyütüldüğünü veya azaldığını gösteren bir ağırlıkla etiketlenmiş bir sinyal taşır. Bazı bağlantıların, sinyalin sınıflandırmasında çok önemli olduğunu belirten, sinyali güçlendiren büyük, pozitif ağırlıkları vardır. Diğerleri sinyalin gücünü azaltan negatif ağırlıklara sahiptir, böylece düğüm çıktısının son sınıflandırmada daha az önemli olduğunu belirtir. Böyle bir sistem, bir öğrenme algoritması kullanılarak

değiştirilebilen bağlantı ağırlıklarına sahip bir grafik yapısından oluşuyorsa bu sisteme yapay sinir ağı denmektedir.

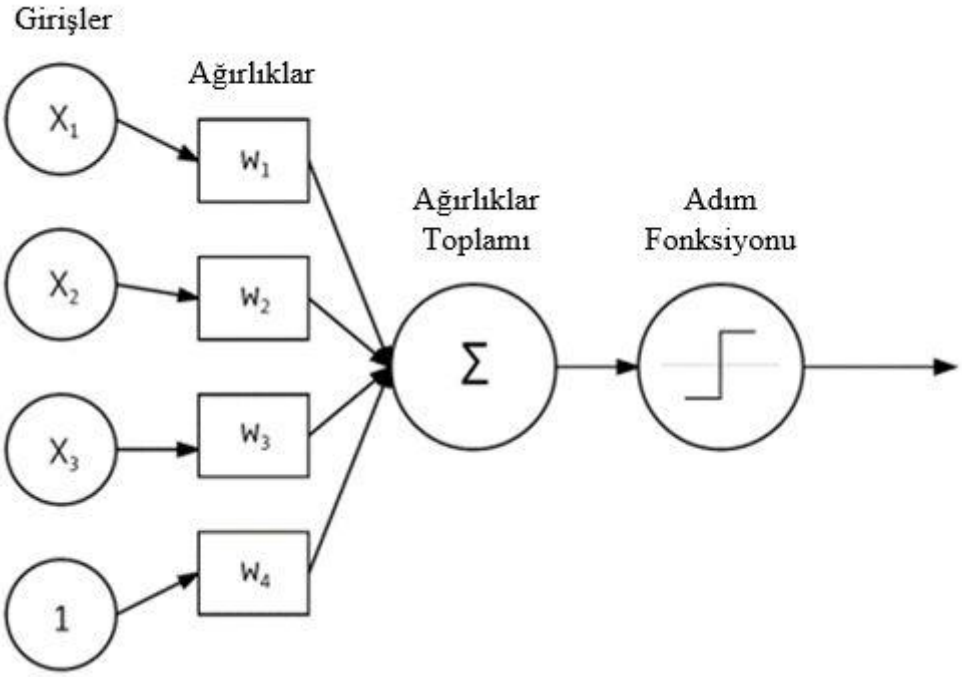



Şekil 3.2. Biyolojik bir nöronun yapısı.

Beynimiz, her biri yaklaşık 10.000 diğer nöronla bağlantılı olan yaklaşık 10 milyar nörondan oluşur. Nöronun hücre gövdesi *soma* olarak adlandırılır. Girişler (dendritler) ve çıkışlar (aksonlar), sinir hücresini diğer sinir hücrelerine bağlar (Şekil 3.2).

Her bir sinir hücresi, dendritlerinde diğer nöronlardan elektrokimyasal girdiler almaktadır. Bu elektriksel girişler sinir hücresini aktive etmek için yeterince güçlü ise, aktive edilmiş nöron sinyali aksonu boyunca ileterek diğer nöronların dendritlerine iletir. Bu bağlı nöronlar da aktive olabilir, böylece mesajı iletme işlemini sürdürürler. Buradaki en önemli nokta, bir nöron aktiflenmesinin ikili bir işlem olduğudur (nöron aktif ya da pasiftir). Farklı "aktiflik dereceleri" yoktur. Basitçe söylemek gerekirse, bir nöron ancak somada alınan toplam sinyal belirli bir eşiği aştığında aktifleşir.

Bununla birlikte, YSA' lar beyin hakkında bilinenlerden ve nasıl çalıştığından basitçe ilham almıştır. Derin öğrenmenin amacı beyinlerimizin nasıl işlediğini taklit etmek değil, kendi çalışmalarımızda benzer paralellikler çizmemizi ve anlamamızı sağlayan parçaları almaktır.




Şekil 3.3. YSA yapısı.

Girdilerin basit ağırlıklı toplamını gerçekleştiren basit bir YSA yapısı Şekil 3.3' te gösterilmiştir. $x1, x2$ ve $x3$ YSA' nın girdilerini göstermektedir ve veri setindeki tek bir satıra karşılık gelmektedir. Sabit 1 değeri bias değeridir ve genellikle veri setinin içine dâhil edilir. Bu girdiler nitelik vektörü olarak ele alınır.

Her bir $x$ YSA' ya, $w1, w2, \ldots, wn$ ağırlıkarından oluşan bir $w$ ağırlık vektörü ile bağlanır. Bunun anlamı her bir $x$ girdisi için buna karşılık gelen bir $w$ ağırlık değeri vardır. Son olarak, Şekil 3.3' ün sağında bulunan çıktı düğümü ağırlıklı toplamı alır, $f$ ("nöronun" aktifliğini belirlemek için kullanılır) bir aktivasyon işlevi uygular ve bir değer verir. Çıktı matematiksel olarak:

$\mathrm{w_1x_1} + \mathrm{w_2x_{2+}}\ldots\mathrm{w_nx_n}$

$f\ (\sum_{i=1}^{n} w_i x_i)$

$f(a\check{g})\quad a\check{g} = \sum_{i=1}^{n} w_i x_i$ (3.11)

ile gösterilebilir. Çıktı değerinin nasıl ifade edildiğine bakılmaksızın, sadece girdilerin ağırlıklı toplamı alınır ve ardından bir $f$ aktivasyon işlemi uygulanır.

Aktivasyon Fonksiyonları

En basit aktivasyon fonksiyonu adım fonksiyonudur.

$$f(net) = \{1 if net > 0\} değilse\ 0 \quad (3.12)$$

Denklemden görülebileceği gibi bu çok basit bir aktivasyon fonksiyonudur. Eğer ağırlıklı toplam 0 dan büyükse 1 değilse 0 değerini alır. Bununla birlikte, sezgisel ve kullanımı kolay olmakla birlikte, adım fonksiyonu diferansiyellenemez, bu da en dik iniş uygularken ve ağı eğitirken sorunlara yol açabilir. Bunun yerine, YSA literatürünün tarihinde kullanılan daha yaygın bir aktivasyon fonksiyonu sigmoid fonksiyondur. Sigmoid fonksiyonu:

$$t = \sum_i^n w_i x_i \qquad s(t) = 1/(1 + e^{-t}) \quad (3.13)$$

şeklindedir.

Sigmoid fonksiyonu, öğrenme için aşağıda belirtilen sebeplerden ötürü adım fonksiyonundan daha iyi bir seçimdir.

- Her yerde sürekli ve diferansiyellenebilir.
- $y$ - eksenine göre simetriktir.
- Asimptotik olarak limit değerine yaklaşır.

Buradaki birincil avantaj, sigmoid fonksiyonunun düzgünlüğünün, öğrenme algoritmaları geliştirmeyi kolaylaştırmasıdır. Bununla birlikte, sigmoid fonksiyonunun kullanımında iki büyük sorun vardır:

- Sigmoid fonksiyonunun çıktıları sıfır merkezli değildir.

- Doymuş nöronlar gradyanı öldürür, çünkü gradyanın deltası son derece küçük olacaktır.

Hiperbolik tanjant veya tanh (sigmoidin benzer bir şekli ile) 1990' ların sonlarına kadar aktivasyon fonksiyonu olarak da yoğun bir şekilde kullanıldı. tanh fonksiyonu:

$$f(z) = \tanh(z) = (e^{z} - e^{-z})/(e^{z} + e^{-z}) \tag{3.14}$$

ile verilir.

Tanh fonksiyonu sıfır merkezlidir fakat ağ doygunluğa ulaştığında gradyanların ölme problemi devam etmektedir.

Hahnloser ve ark. (2000), çalışmalarında;

$$f(x) = max(0, x) \tag{3.15}$$

ile gösterilen Rectified Linear Unit (ReLU) fonksiyonunu tanıtmışlardır. ReLU fonksiyonu doygun değildir ve ayrıca oldukça hesaplamalı olarak etkindir. Ampirik olarak, ReLU aktivasyon fonksiyonu hemen hemen bütün uygulamalarda hem sigmoid hem de tanh fonksiyonlarından daha iyi performans gösterme eğilimindedir. Hahnloser ve Seung (2003) çalışmalarında ReLU aktivasyon fonksiyonunun öncekilerden daha güçlü biyolojik motivasyonlara sahip olduğu tespit edilmiştir. 2015 itibariyle, ReLU derin öğrenmede kullanılan en popüler aktivasyon fonksiyonudur (LeCun ve ark., 2015). Bununla beraber sıfır değerinde gradyanın bulunamama problemi bu aktivasyon fonksiyonu kullanıldığında da devam etmektedir.

Maas ve ark. (2013) çalışmasında, Leaky ReLU aktivasyonu:

$$f(net) = \begin{cases} net, & if net >= 0 \\ \alpha \; x \; net, & otherwise \end{cases} \tag{3.16}$$

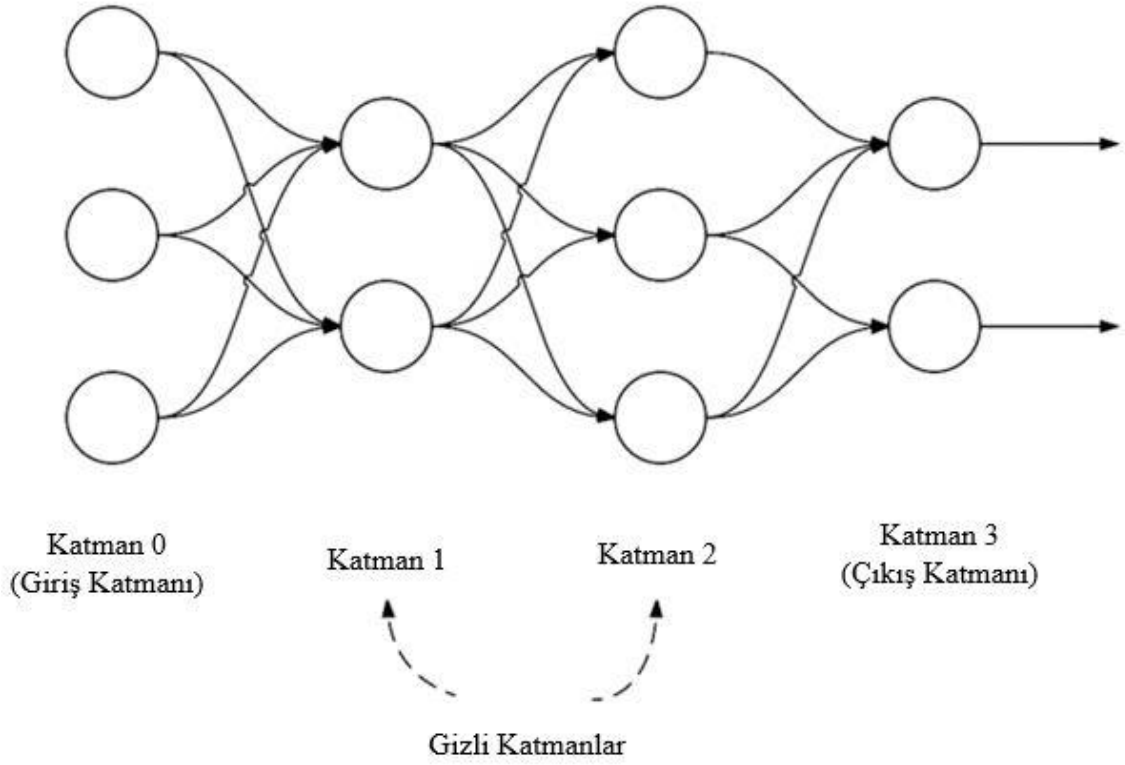
ile verilmiştir. Bu fonksiyon ReLU fonksiyonun aksine negatif değerler alabilmektedir. Parametrik ReLU, Leaky ReLU dayanarak elde edilir ve alfa parametresinin öğrenilmesini içerir. Clever ve ark. (2015) çalışmalarında Exponential Linear Unit (ELUs) aktivasyon fonksiyonunu tanıtmışlardır. ELU fonksiyonu:

$$f(net) = \begin{cases} net, & if net >= 0 \\ \alpha\, x\, (\exp(net) - 1), & otherwise \end{cases} \quad (3.17)$$

ile gösterilir. Burada alfa sabittir ve ağ mimarisi oluşturulduğunda ayarlanır. Bu alfa parametresi, öğrenilen parametrik ReLU' dan farklıdır. ELU, ReLU' ya göre daha yüksek sınıflandırmaya sahiptir (Clevert ve ark., 2015).

İleri Beslemeli Ağ Mimarileri

Birçok farklı YSA mimarisi olsa da, en yaygın mimari, Şekil 3.4' te gösterildiği gibi ileri beslemeli yapay sinir ağıdır.




Şekil 3.4. 3 giriş düğümlü, 2 düğümlü gizli katman, 3 düğümlü ikinci bir gizli katman ve 2 düğümlü bir son çıkış katmanına sahip ileri beslemeli bir yapay sinir ağı örneği.

Bu tür bir mimaride, düğümler arasındaki bağlantıya sadece $i$ katmanındaki düğümlerden $i + 1$ katmanındaki düğümlere izin verilir. İleri beslemeli terimi buradan kaynaklanmaktadır. Bu mimaride katmanlar arası veya geriye dönük bağlantılar yoktur. İleri beslemeli sinir ağları geriye dönük bağlantılar içerdiğinden, bu tip mimariye tekrarlayan sinir ağları denir. İleri beslemeli bir yapay sinir ağı tanımlamak için normal olarak her katmandaki düğüm sayısını belirtmek adına bir tam sayı dizisi kullanılır. Örneğin, Şekil 3.4' teki ağ, 3 – 2 – 3 - 2 ileri beslemeli bir sinir ağıdır.

İlk katman giriş katmanı olarak isimlendirilir ve $x_i$ girdilerini içerir. Bu girdiler bir görüntünün piksel değerleri veya bu görüntüden elde edilen nitelik vektörü olabilir. İkinci ve üçüncü katman gizli katman olarak isimlendirilir. Problemin karmaşıklığına göre gizli katman sayısı kullanıcı tarafından belirlenir.

Son katman çıktı katmanı olarak isimlendirilir. Bu katmanda YSA' nın sınıflandırma çıktısı elde edilir. Bu katmandaki düğüm sayısı problemdeki sınıf sayısına eşittir. Örneğin, el yazısı rakamlarını sınıflandırmak için bir YSA inşa edersek, çıkış katmanımız her biri 0 - 9 rakamları için bir tane olmak üzere 10 düğümden oluşur.

Sinirsel öğrenme, bir ağdaki düğümler arasındaki ağırlıkları ve bağlantıları değiştirme yöntemini ifade eder. Biyolojik olarak öğrenme Hebb' in ilkesi ile tanımlanır: " Bir A hücresinin aksonu, bir B hücresini uyarmaya yeterince yakın olduğunda ve durmadan ya da devamlı olarak ateşlediğinde (harekete geçirdiğinde), B hücresini ateşleyen hücrelerden biri olan A hücresinin etkinliğinin artması, olan bir hücrede ya da her iki hücrede bazı büyüme prosesleri veya metabolik değişimler meydana getirir. " (Hebb, 2005).

YSA' lar açısından, bu ilke, aynı girdiyle sunulduğunda benzer çıkışlara sahip olan düğümler arasındaki bağlantı gücünde bir artış olması gerektiği anlamına gelir. Buna korelasyon öğrenmesi denir çünkü nöronlar arasındaki bağlantıların gücü nihayetinde çıktılar arasındaki korelasyonu temsil etmektedir. YSA' lar denetimli, denetimsiz ve yarı denetimli öğrenme problemlerinde kullanılabilmektedir.

Perceptron Algoritması

İlk olarak 1958 yılında Rosenblatt (1958) tarafından tanıtılan Perceptron tartışmasız YSA algoritmalarının en eski ve en basitidir. Bu çalışmadan sonra Perceptron

tabanlı teknikler sinir ağı topluluğunun en çok kullandığı teknikler olmuştur. 1969 yılında Minsky ve Papert tarafından yayınlanan çalışmada tek katmanlı perceptronların lineer olmayan veri noktalarını ayırt edemediği belirlenmiştir (Minsky ve Papert, 1969). Birçok gerçek dünya verisi lineer olmayan ilişkiye sahip olduğundan Perceptron özelinde yapay sinir ağı çalışmaları durma noktasına gelmiştir. Ancak geriye yayılım algoritmasının kullanılmaya başlamasından itibaren yapay zekâ kışı olarak nitelendirilebilecek dönem sona ermiş ve günümüze kadar hız kesmeden devam etmiştir. Perceptron, daha gelişmiş çok katmanlı ağlar için zemin hazırlamasının anlaşılması için çok önemli bir algoritmadır. Tezin bu bölümünde Perceptron mimarisi incelenecek ve Perceptron' u eğitmek için kullanılan eğitim prosedürü (delta kuralı) açıklanacaktır. Ayrıca ağın sonlandırma kriterlerine de değinilecektir.

VE, VEYA ve XOR Veri Kümeleri

Bitsel operatörler ve ilişkili bitsel veri kümeleri iki giriş bitini kabul eder ve işlemi uyguladıktan sonra son bir çıkış biti üretir. İki giriş biti verildiğinde, her biri 0 veya 1 değerini alır. Bu iki bitin dört olası kombinasyonu vardır. AND, OR ve XOR için olası giriş ve çıkış değerlerini sağlar (Şekil 3.5):

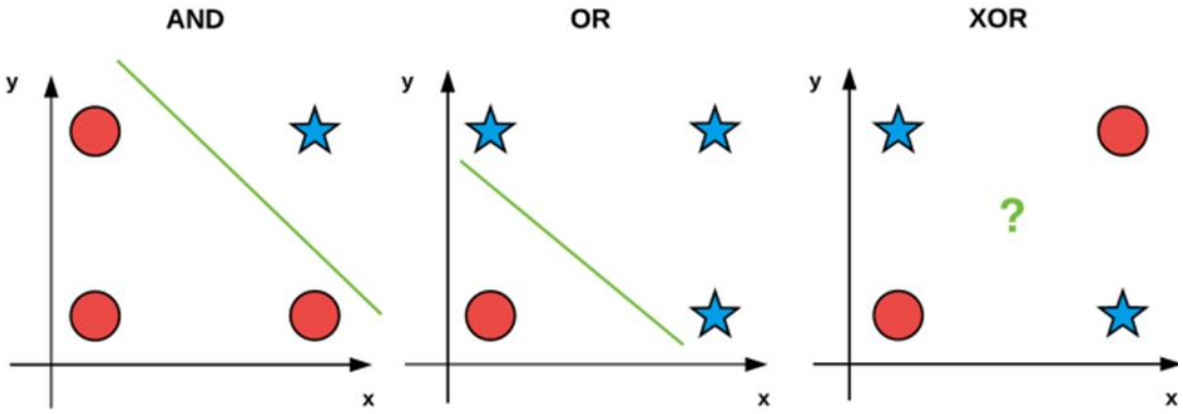
| $x_0$ | $x_1$ | $x_0 \& x_1$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

| $x_0$ | $x_1$ | $x_0 \mid x_1$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

| $x_0$ | $x_1$ | $x_0 \wedge x_1$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

Şekil 3.5. VE, VEYA ve XOR Veri Kümeleri Tablosu.

Makine öğrenmesi algoritmalarını test etmek ve hata ayıklamak için sık sık bu basit “bitsel veri kümeleri” kullanılır.

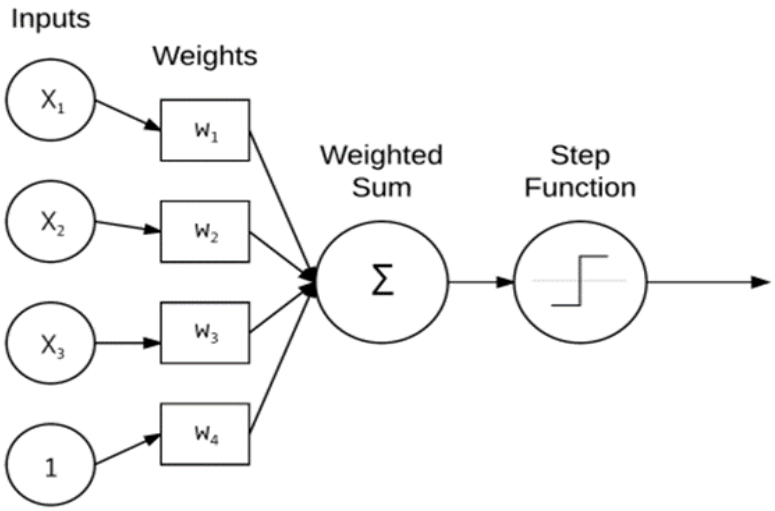



Şekil 3.6. Bitsel veri kümeleri.

Hem AND hem OR doğrusal olarak ayrılabilirdir yani 0 ve 1 sınıflarını ayıran bir çizgi çizilebilir. Fakat aynı şey XOR için doğru değildir. Bu nedenle, XOR, doğrusal olmayan bir şekilde ayrılabilir bir veri kümesi örneğidir. İdeal olarak, makine öğrenimi algoritmaları, gerçek dünyada karşılaşılan çoğu veri kümesi doğrusal olmadığından lineer olmayan sınıfları ayırabilmesi istenir.

Perceptron Mimarisi

Rosenblatt (1958), bir Perceptron' u, özellik vektörlerinin (veya ham piksel yoğunluğunun) etiketli örneklerini (yani denetimli öğrenmeyi) kullanarak öğrenen ve bu girişlere karşılık gelen çıkış sınıfı etiketleriyle eşleştiren bir sistem olarak tanımladı.




Şekil 3.7. Perceptron ağının mimarisi.

$x_i$ girdileri ve bunlara karşılık gelen $w1, w2, \ldots, wn$ ağırlıklarından ağın çıktı düğümüne doğru bağlantılar vardır. Bu düğüm girdilerin ağırlıklı toplamını alır ve çıktının sınıf etiketini belirlemek için adım fonksiyonunu uygular. Perceptron çıktısı birinci sınıf için ya 0 ya da 1' dir.

Perceptron Eğitim Prosedürü ve Delta Kuralı

Perceptron eğitimi, oldukça basit bir işlemdir. Burada amaç, eğitim setindeki her örneği doğru bir şekilde sınıflandıran bir ağırlık seti elde etmektir. Perceptronu eğitmek için ağ, tekrar tekrar eğitim verileri ile beslenir. Ağ tam bir eğitim verisi kümesini her gördüğünde, bir epoch geçtiği söylenir. İki veri sınıfımızı doğrusal olarak ayırmak için $w$ ağırlık vektörü öğrenilinceye kadar normalde birçok epoch geçer.

Perceptron eğitim algoritmasının sözde kodu aşağıda verilmiştir:

1. $w$ ağırlık vektörüne küçük rastgele değerler atanır.
2. Perceptron yakınsayıncaya kadar;
   a) D veri setindeki herbir nitelik vektörü ve sınıf etiketi üzerinde döngü devam eder.
   b) $x$ değeri alınır ve bu değer ağdan geçirilerek $y_j = f(w(t).x_j)$ çıktı değeri hesaplanır.
   c) Her $0 \leqslant i \leqslant n$ için $w: w_i(t+1) = w_i(t) + alpha(d_j - y_j)x_{j,i}$ ağırlıkları güncellenir.

Alpha değeri bizim öğrenme hızımızdır ve attığımız adımın ne kadar büyük (veya küçük) olduğunu kontrol eder. Bu değerin doğru ayarlanması çok önemlidir. Daha büyük bir α değeri, doğru yönde bir adım atmamıza neden olacaktır; ancak, bu adım çok büyük olabilir ve yerel / global minimum değerinin atlanmasına neden olabilir. Tersine, küçük bir $\alpha$ değeri küçük adımlarını doğru yönde atmamızı sağlar, bu da yerel / global minimumu aşmamamızı sağlar; ancak, bu küçük adımlar, öğrenme için uzun bir zaman alabilir.

Perceptron Eğitimini Sonlandırma

Perceptron eğitim süreci, tüm örnekler doğru sınıflandırılıncaya kadar veya belirlenen epoch sayısına ulaşılıncaya kadar devam eder. Sonlandırma alpha yeterince küçük olduğunda veya eğitim verisi lineear ayrılabilir olduğunda gerçekleşir.

Geriye Yayılım ve Çok Katmanlı Ağlar

Geriye yayılım algoritması sinir ağları tarihinde en önemli algoritmadır. Geriye yayılım algoritması olmadan bugün kullanılan derin öğrenme algoritmalarını eğitmek asla mümkün olmazdı. Geriye yayılım algoritması modern sinir ağlarının ve derin öğrenmenin köşe taşı olarak düşünülebilir.
Orjinal geriye yayılım algoritmasının ortaya çıkışı 1970' lere kadar gitmektedir. Ancak Rumelhart 1986 çalışmasında derin ağları eğitmek için algoritmanın hızlı halini önermiştir.

Geriye yayılım algoritması iki kısımdan oluşmaktadır:

1. İleri geçiş ilk kısımdır. Burada girdiler ağ boyunca işlenir ve çıktı tahminler elde edilir. Bu kısım yayılım kısmı olarakta isimlendirilir.
2. Geri dönüş kısmında ağın son katmanındaki hata fonksiyonun gradyanı hesaplanır ve ağın ağırlıklarını güncellemek için yinelemeli bir şekilde zincir kuralı bu gradyana uygulanır.

İleri Geçiş

İleriye geçiş işleminin amacı, bir dizi nokta çarpımı ve aktivasyon uygulayarak girdilerimizi ağ üzerinden yaymak ve ağın çıkış katmanına ulaşana kadar ilerlemektir. Bu süreci görselleştirmek için önce XOR veri kümesini göz önüne alalım (Şekil 3.8) (sol kısım).

| $x_0$ | $x_1$ | $y$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

| $x_0$ | $x_1$ | $x_2$ |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

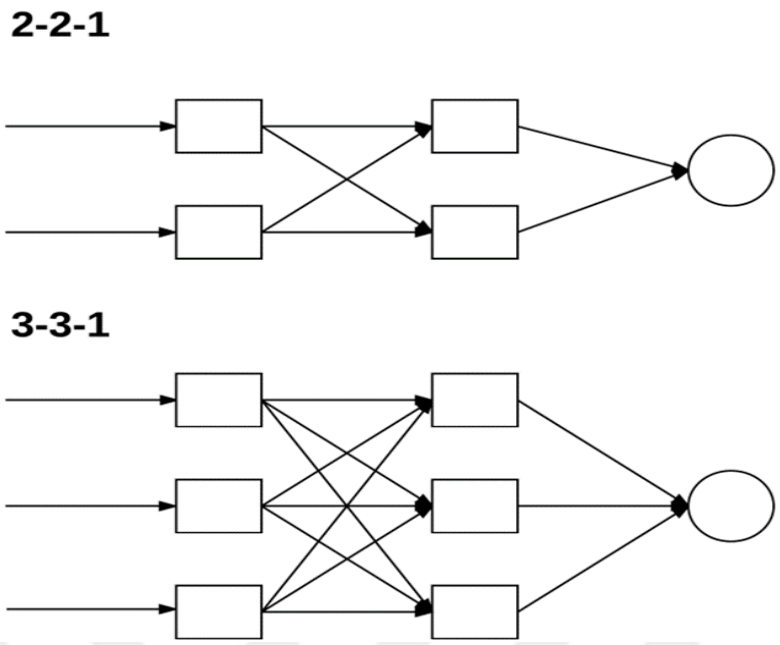
Şekil 3.8. Bitsel XOR veri kümesi (sınıf etiketleri dahil)(sol kısım), XOR veri kümesi yanlı sütun eklenmiş tasarım matrisi (sağ kısım).

Burada, tasarım matrisindeki (sol kısım' da) her $x$ girişinin 2 boyutlu olduğunu görebiliriz ( Her veri noktası iki sayı ile temsil edilir ). Örneğin, ilk veri noktası nitelik vektörü $(0, 0)$, ikinci veri noktası $(0, 1)$ vb. ile temsil edilir. Çıktı değerleri en sağdaki sütunda yer almaktadır. Hedef çıktı değerlerimiz sınıf etiketleridir. Tasarım matrisinden bir girdi verildiğinde amaç hedef çıktı değerini doğru tahmin etmektir.

Bu problem ile ilgili mükemmel bir sınıflandırma doğruluğu elde etmek için en azından tek bir gizli katmana sahip ileriye dönük bir sinir ağına ihtiyaç vardır. Bu sinir ağında 2 - 2 - 1 mimarisi ele alınsın (Şekil 3.9) (üst kısım). Önyargı $b$ terimini ağımıza dâhil etmenin iki yolu vardır:

1. Ayrı bir değişken olarak kullanma
2. Önyargıyı, özellik vektörlerine 1 'lerden oluşan bir sütun ekleyerek ağırlık matrisinde eğitilebilir bir parametre olarak kullanma.

Giriş nitelik vektörünün boyutunu değiştirildiğinden (normalde sinir ağı uygulamasının kendi içinde gerçekleştirilir, böylece tasarım matrisini açıkça değiştirmeye gerek kalmaz), bu (algılanan), ağ mimarimizi 2 - 2 - 1' den bir (iç) 3 – 3 - 1 mimarisine değiştirir (Şekil 3.9) (alt kısım). Bu ağ mimarisine hala 2 - 2 - 1 olarak bakılır, ancak uygulama söz konusu olduğunda, ağırlık matrisine gömülü önyargı teriminin eklenmesi nedeniyle aslında mimari 3 - 3 - 1 şeklindedir.

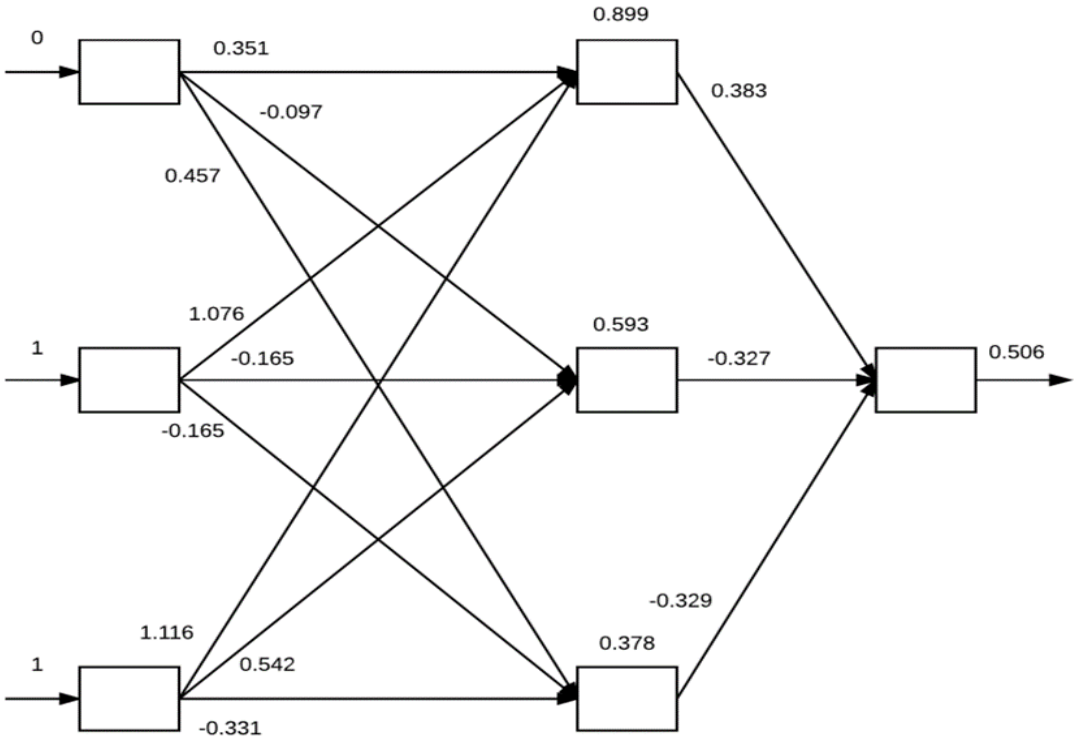



Şekil 3.9. Üst kısım: 2 - 2 - 1 mimarisi. Alt kısım: Gerçek iç ağ mimarisi temsili, önyargı nedeniyle 3 - 3 - 1'dir.

Son olarak, hem giriş katmanı hem de tüm gizli katmanları bir önyargı terimi gerektirir ancak çıktı katmanı bir önyargı gerektirmez. Önyargı hilesini uygulamanın faydası, önyargı parametresini artık açıkça izlemeye gerek kalmamasıdır. Artık ağırlık matrisi içinde eğitilebilir bir parametredir, böylece eğitimi daha verimli ve uygulaması oldukça kolaydır.

İleriye doğru geçişi görmek için öncelikle ağdaki ağırlıklar Şekil 3.10' daki gibi başlatılır. Ağırlık matrisindeki her ok kendisiyle ilişkilendirilmiş bir değere sahiptir. Bu, belirli bir düğüm için geçerli ağırlık değeridir ve verilen bir girişin arttığı veya azaldığı miktarı gösterir. Bu ağırlık değeri daha sonra geri yayılma aşamasında güncellenecektir. Örnek olarak Şekil 3.10' un en solunda, özellik vektörü (0, 1, 1) (ve hedef çıktı değeri 1) ağa sunulsun. Burada, ağdaki üç giriş düğümüne 0, 1 ve 1 atanmıştır. Değerleri ağ üzerinden yaymak ve son sınıflandırmayı elde etmek için, nokta çarpımı girdiler ve ağırlık değerleri arasında alınmalı ve ardından bir aktivasyon fonksiyonu (bu durumda sigmoid işlevi $\sigma$) uygulanmalıdır.

$$1.\ \sigma\left((0 \times 0.351) + (1 \times 1.076) + (1 \times 1.116)\right) = 0.899 \qquad (3.18)$$




Şekil 3.10. Ağırlıklandırılmış sinir ağı.

$$2.\ \sigma\left((0 \times -0.097) + (1 \times -0.165) + (1 \times 0.542)\right) = 0.593 \quad (3.19)$$

$$3.\ \sigma\left((0 \times 0.457) + (1 \times -0.165) + (1 \times -0.331)\right) = 0.378 \quad (3.20)$$

Çıktı tahminini hesaplamak için, nokta çarpımı bir sigmoid aktivasyonunun ardından bir kez daha hesaplanır.

$$\sigma\left((0.899 \times 0.383) + (0.593 \times -0.327) + (0.378 \times -0.329)\right) = 0.506 \quad (3.21)$$

Yani ağın çıktısı 0.506' dır. Bu çıktının doğru sınıflandırma olup olmadığını belirlemek için bir adım fonksiyonu uygulanabilir.

$$f(net) = \begin{cases} 1, & if net > 0 \\ 0, & otherwise \end{cases} \quad (3.22)$$

Adım fonksiyonunu $net = 0.506$ ile uygulayarak, ağın aslında doğru sınıf etiketi olan 1 olduğu tahmin edilmiştir. Fakat bu değer için ağın bu sınıf etiketinin çok kararlı

olmadığı söylenebilir. İdeal olarak bu tahminin $0.98 - 0.99$' a yakın olması beklenirdi. Ağın gerçekten öğrenmesini sağlamak için geriye geçiş aşamasının uygulanması gerekmektedir.

Geriye Geçiş

Geriye yayılım algoritmasını uygulamak için aktivasyon fonksiyonu hatanın $wi,j$ $hata(E), netj$ ve $oj$ çıktısına göre kısmi türevinin hesaplanabilmesi diferansiyellenebilir olmalıdır.

$$\frac{\partial \mathrm{E}}{\partial w_{i,j}} = \frac{\partial \mathrm{E}}{\partial o_j} \frac{\partial o_j}{\partial net_j} \frac{\partial net_j}{\partial w_{i,j}} \tag{3.23}$$

Optimizasyon Metotları ve Düzenlileştirme

Optimizasyon algoritmaları, sinir ağlarını güçlendiren ve verilerden kalıpları öğrenmelerini sağlayan algoritmalardır. "Neredeyse tüm derin öğrenme çok önemli bir algoritma tarafından desteklenmektedir: Stokastik Gradyan İnişi (SGD)" (Goodfellow ve ark., 2016). Saf rastgeleliğe güvenmek yerine, tam anlamıyla ağırlıkları ve önyargıyı geliştirmemizi sağlayan bir optimizasyon algoritması tanımlanması gerekir. Tezin bu bölümünde, sinir ağlarını ve derin öğrenme modellerini eğitmek için kullanılan en yaygın algoritma olan Gradyan iniş algoritması tanıtılacaktır. Gradyan inişin birçok çeşidi vardır ancak her durumda fikir aynıdır; parametreleri yinelemeli olarak değerlendirilir, hata hesaplanır, daha sonra kayıp en aza olacak şekilde küçük bir adım atılır.

Gradyan İniş

Gradyan iniş algoritmasının iki ana uygulaması bulunmaktadır.

1. Standart uygulaması.
2. Daha çok kullanılan stokastik versiyonu.

Hata Bölgesi ve Optimizasyon Yüzeyi

Gradyan iniş yöntemi, bir hata bölgesi (aynı zamanda bir optimizasyon yüzeyi olarak da bilinir) üzerinde çalışan yinelemeli bir optimizasyon algoritmasıdır. Gradyan iniş örneğinde ağırlık değerleri $x$ ekseninde ve bu değerlerde hata fonksiyonunun aldığı değerler de $y$ ekseninde gösterilir.


Şekil 3.11. Hata fonksiyonunun 2 boyutlu ve daha gerçekçi olarak 3 boyutlu örneği.

Görüldüğü gibi hata bölgesi birçok tepe ve düzlüğe sahiptir. Her bir tepe hata fonksiyonunun yerel maksimumu olarak isimlendirilir. Hata bölgesindeki en yüksek değere sahip lokal maksimum global maksimum olarak tanımlanır. Benzer şekilde küçük bölgelerdeki küçük değerlere de yerel minimum ismi verilir. Yerel minimumların en küçüğü de global minimum olarak isimlendirilir. İdeal olarak ağırlık değerlerinin en optimal değerleri aldığı bu global minimum bulunmak istenir. Tezin bu bölümünde gradyan iniş algoritması tanıtalacaktır.

Gradyan İniş

$f: \mathbb{R}^d \rightarrow \mathbb{R}$ diferansiyellenebilir bir fonksiyonun gradyanı $f$' nin kısmi türevlerinin vektörüdür ve $\nabla f(w)$ ile gösterilir. Yani;

$$\nabla f(w) = \left(\frac{\partial f(w)}{\partial w[1]}, \ldots, \frac{\partial f(w)}{\partial w[d]}\right) \tag{3.24}$$

gradyan iniş yinelemeli bir algoritmadır. $w$' ya bir başlangıç değeri verilir sonra her bir adımda mevcut noktadaki gradyanın negatifi yönünde bir adım atılır. Yani güncelleme adımı,

$$w^{(t+1)} = w^{(t)} - \eta \nabla f(w^{(t)}) \tag{3.25}$$

denklemindeki gibidir.

Burada $\eta$ adım büyüklüğüdür. Sezgisel olarak gradyan $w(t)$ civarında $f$' nin en yüksek artış hızını işaret ettiğinden, algoritma zıt yönde küçük bir adım atmakta, böylece fonksiyonun değeri düşmektedir. Sonunda, $T$ adım sonrasında algoritma

$$\overline{w} = \frac{1}{T}\sum_{i=1}^{T} w^{(t)} \tag{3.26}$$

ortalama vektörünü verir. Çıktı ayrıca $w^{(T)}$ son vektörü veya en iyi performans vektörü

$$\underset{t \in [T]}{\operatorname{argmin}} f(w^{(t)}) \tag{3.27}$$

‘ nü verebilir. Gradyan iniş diferansiyellenebilir olmayan ve stokastik duruma genelleştirildiğinde ortalamanın alınması oldukça faydalı olabilir.

Stokastik Gradyan İniş

Gradyan iniş algoritması büyük veri setlerinde oldukça yavaş çalışmaktadır. Bu gibi durumlarda standart gradyan iniş algoritmasında, gradyan hesaplaması ve ağırlık güncellemesi, veri setinin bir bölümünü kullanarak yapılabilir. Bu modifikasyon daha gürültülü güncellemelere ayrıca gradyan boyunca daha fazla adım atılmasına neden olsa da daha hızlı bir yakınsamaya neden olur. Ayrıca hata üzerinde ve sınıflandırma doğruluğunda negatif bir etkisi de yoktur. Skotastik gradyan iniş, derin sinir ağlarının

eğitimi için en önemli algoritmadır. Bu algoritmanın ortaya çıkışından 50 yıl üzeri zaman geçmesine rağmen büyük ağların eğitiminde en çok kullanılan algoritmadır.

#### 3.2.2.4.2. Evrişimsel sinir ağları (CNN)

CNN, sinir ağlarının daha kompleks formu olarak karşımıza çıkmaktadır. Çok katmanlı bu ağ yapısında özellik çıkarımı için konvulasyon katmanı, havuz katmanı, tam bağlantılı katman gibi 3 ana katman mevcuttur. Bu bölümde CNN' de yer alan evrişimsel katman, havuzlama katmanı, aktivasyon fonksiyonu, hata fonksiyonu, düzenlileştirme, eniyileme gibi 6 önemli yapı taşı açıklanacaktır. Normalde yapay sinir ağlarında giriş katmanındaki her nöron bir sonraki katmana çıkış nöronu ile bağlıdır. Bu da fully connected olarak isimlendirilir. CNN yapısında ise son katmana kadar FC yapısı kullanılmaz (Heaton, 2018).

CNN görüntü, ses işleme, biyomedikal gibi alanlarda yaygın olarak kullanılmaktadır. Literatüre bakıldığında CNN' in kullanıldığı en popüler alan imge sınıflandırmadır.

##### 3.2.2.4.2.1. Evrişimsel katman

Bu katman filtre bankası olarak tanımlanan ilk giriş katmanıdır. Bir konvolüsyon katmanı $K$ adet genişlik ve yüksekliği olan filtreler içerir. Bu filtreler genellikle $3\,x\,3$' lük veya $5\,x\,5'$lik kare olmaktadır. Bu filtrelerin amacı imge üzerinde kenarları elde etmek ve bu kenarları kullanarak imge üzerindeki şekilleri belirlemektir. Bunun ileri aşaması olarak bu şekiller kullanılarak imge üzerindeki objenin tanımlanmasını sağlamaktır. Bu filtrelerin çalışma şeklini şöyle açıklamak mümkündür. İmgenin üzerinde belirlenen adım sayısına göre gezinerek üzerinden geçtiği piksel değerleri ile filtrenin elemanları bire bir çarpılır. Bu değerler toplanarak, filtrenin merkezindeki piksel değeri bu toplam değer ile değiştirilir. Bu katmandaki amaç filtre değerleri uygulayarak aktivasyon haritaları oluşturmaktır. Daha derin katmanlarda öğrenilmiş filtre değerleri kullanılarak objelerin tanımlanması sağlanmaktadır. Diğer bir önemli husus da filtrenin imge üzerinde kaç adım gezineceği ve giriş imgesine padding eklenip eklenmeyeceğidir (Rosebrock, 2017). İmgenin üzerinde gezdirilen filtreler için hangi ağırlıkların

seçileceğine başlangıçta rastgele olarak karar verilir. Konvolüsyon katmanından sonra kullanılan adım sayısına göre imgenin boyutunda küçülme olabilmektedir. CNN içerisinde her bir katmanda farklı sayıda filtreler mevcuttur. Bu filtrelerin sonuçları birleştirilerek bir sonraki katmanın beslenmesi için kullanılır. Ağın eğitimi boyunca CNN tarafından bu filtre değerleri öğrenilmektedir. Bilindiği üzere girişteki imgenin diğer katmanlarda orijinal boyutunun korunması isteniyorsa başlangıçta kenarlarına padding uygulanır (Rosebrock, 2017).

Çıkış imgesinin boyutunun ne olacağı aşağıdaki denklem yardımıyla hesaplanır.

$$((W - F + 2P)S) + 1 \tag{3.28}$$

Burada $W$ imgenin boyutu varsayılan olarak karedir. $F$ filtre boyutu, $P$ padding miktarı, $S$ ise adım sayısıdır. Buradan elde edilen değerin tamsayı olması gerekmektedir. Örneğin $227 * 227'$ lik bir imge üzerinde $7 * 7'$ lik filtre, $P$ oranı 0 ve stride (adım) 5 olduğunu düşünürsek, bir sonraki katmandaki imgenin boyutu:
$((227 - 7 + 2 * 0)/5) + 1$ denkleminden $45 * 45$ olacaktır (Rosebrock, 2017).

3.2.2.4.2.2. Aktivasyon katmanı

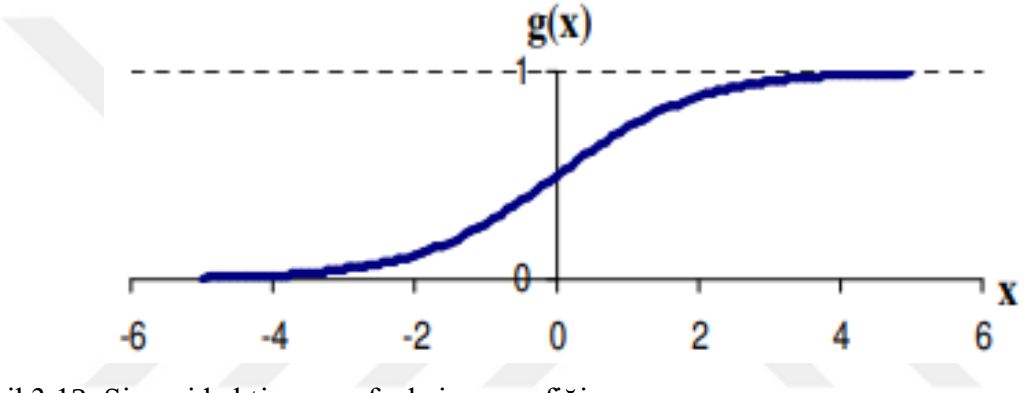
Her konvulasyon katmanını takip eden bu katman öğrenilen bir parametre değeri barındırmadığından katman olarak hesaba katılmaz. Konvulasyon katmanından sonra nöronların aktivasyonu için değerler burada aktivasyon fonksiyonundan geçirilirler. Sigmoid, ReLU, ELU, Leaky ReLU gibi birçok aktivasyon fonksiyonu vardır. En yaygın olarak kullanılan aktivasyon fonksiyonu ReLU' dur (Rosebrock, 2017). Burada aktivasyon fonksiyonları kısaca tanıtılacaktır. Bu aktivasyon fonksiyonlarına ait grafikler Şekil 3.13' te gösterilmektedir. Şekilde gösterilen Step ve tanh aktivasyon fonksiyonları yapay sinir ağlarında 2000' lere kadar kullanılan aktivasyon fonksyonlarıdır (Rosebrock, 2017).

3.2.2.4.2.2.1. Sigmoid

Sigmoid fonksiyonunun aktivasyon fonksiyonu aşağıdaki gibidir:

$$G(x) = \frac{1}{1+e^{-x}} \tag{3.29}$$

Bu fonksiyon, geriye yayılım algoritmaları ile eğitilmiş sinir ağlarında kullanım için özellikle avantajlıdır. Çünkü ayırt edilmesi kolaydır ve bu eğitim için gerçekleşen hesaplama kapasitesini en aza indirebilir. Bu fonksiyon, $(-\infty, \infty)$ aralığını, $(0, 1)$ aralığına Şekil 3.12' de görüldüğü gibi eşler (Karlik ve Olgac, 1998).



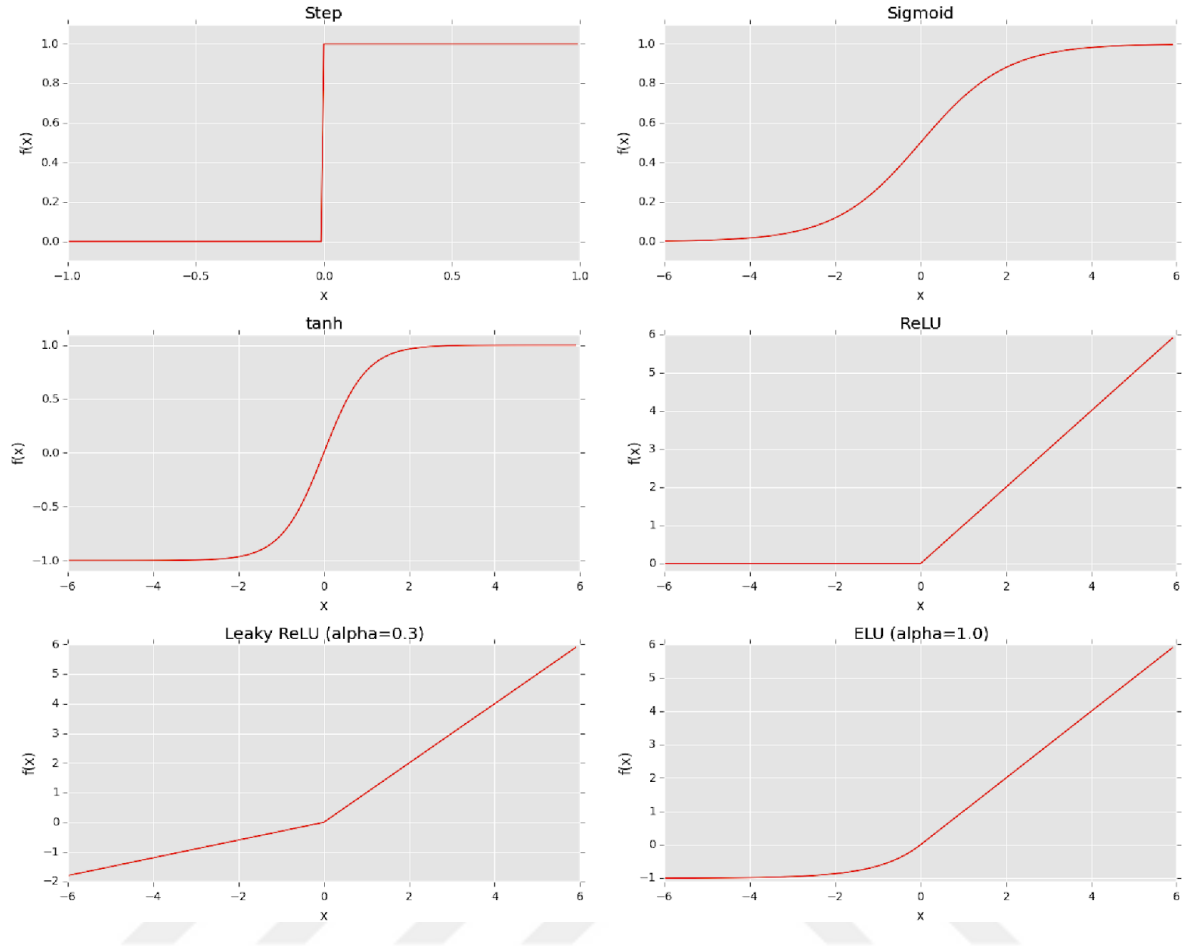

Şekil 3.12. Sigmoid aktivasyon fonksiyon grafiği.

3.2.2.4.2.2.2. ReLU

Son zamanlarda sigmoidal fonksiyonun, gradyan tabanlı öğrenmeye neden olduğu ve varyantlarının eğitim sinir ağında kötü performans gösterdiği kanıtlanmıştır. Bu sorunun çözümünde ReLU popüler olmuştur. Çünkü bu fonksiyon, sigmoid fonksiyonuna kıyasla, stokastik gradyan inişinin yakınsamasını hızlandırmıştır.
ReLU aşağıdaki şekilde tanımlanmaktadır (Wang ve ark., 2018).

$$f_{relu} = \max(0, x) \tag{3.30}$$

ReLU fonksiyonuna ait grafik Şekil 3.13' de verilmektedir.




Şekil 3.13. Aktivasyon fonksiyonları (Rosebrock, 2017).

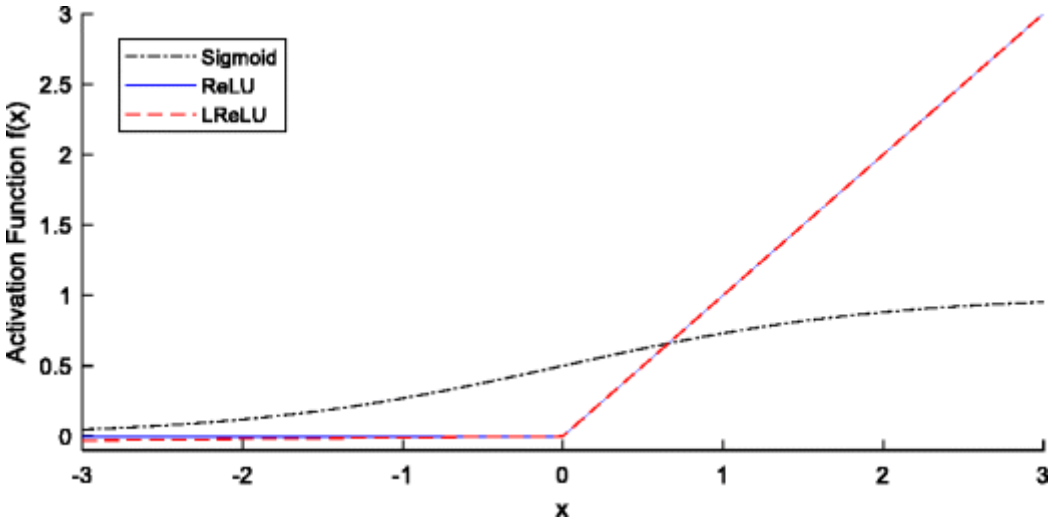
3.2.2.4.2.2.3. Leaky ReLU

ReLU fonksiyonunun 0 olduğu yerlerde ağırlık değerlerinin güncellenmemesi gibi bazı dejavantajlarını ortadan kaldırmak için Mass ve ark, (2013) tarafından aşağıdaki gibi tanımlanan bu fonksiyon önerilmiştir

$$U_{i,j,k} = max(Z_{i,j,k}, 0) + min(Z_{i,j,k}, 0) \quad (3.31)$$

$\lambda$ parametresi [0,1] aralığında tanımlanır. Bu fonksiyonda küçük bir değer ile birim aktif olmasa bile gradient sağlanır. Leaky ReLU fonksiyonuna ait grafik Şekil 3.13'te verilmektedir.

Sigmoid, ReLU ve Leaky ReLU Yu aktivasyon fonksiyonlarının farklılıkları Şekil 3.14' te gösterilmiştir (Wang ve ark., 2018).




Şekil 3.14. Sigmoid, ReLU ve Leaky ReLU fonksiyonları farklılıkları.

3.2.2.4.2.2.4. ELU

ReLU aktivasyon fonksiyonunun aksine negatif değerlerin de hesaba katıldığı bu aktivasyon fonksiyonunun kullanılması ile yüksek doğruluk oranı elde edilmiş ve hızlı öğrenme sağlandığı gözlenmiştir. Bu fonksiyon aşağıdaki şekilde tanımlanır (Rosebrock, 2017).

$$U_{i,j,k} = max(Z_{i,j,k}, 0) + min(\lambda(e^{Z_{i,j,k}} - 1), 0) \quad (3.30)$$

$\lambda$ parametresi, ELU fonksiyonundaki negatif değerlerin miktarını kontrol eder. ELU fonksiyonuna ait grafik Şekil 3.13' te verilmektedir.

3.2.2.4.2.3. Havuzlama katmanı

Havuzlama işlemi genellikle aktivasyon katmanından sonra yerleştirilir. Bu katmanda yapılan işlem aşağı örnekleme katmanı olarak da adlandırılır. Temel amacı, sonraki konvolüsyon katmanı için giriş boyutunu (Genişlik x Yükseklik) azaltmaktır (İnik ve Ülker, 2017).

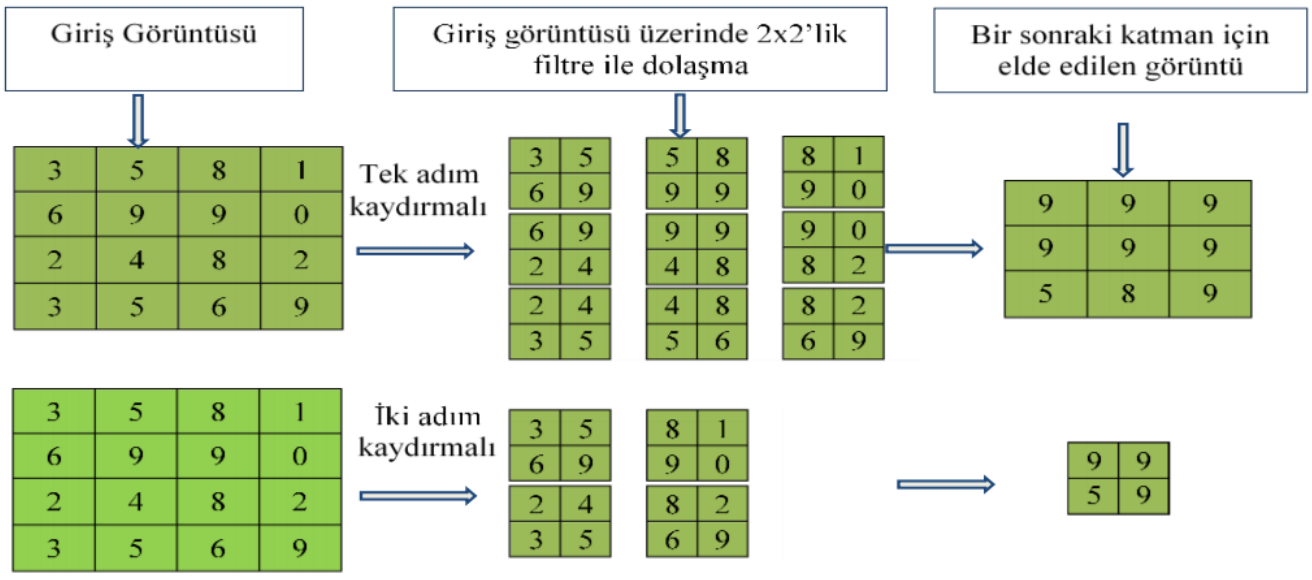
CNN' de giriş değerlerinin büyüklüğünü (size) küçültmek için kullanılacak ilk işlem konvolüsyon katmanındaki stride (adım) sayısının 1' den büyük olmasını sağlamak,

ikinci işlem ise havuzlama katmanı kullanmaktır. Havuzlama katmanının ilk görevi genişlik ve yükseklik gibi uzaysal büyüklüğün azaltılmasını sağlamaktır. Parametre sayısı azaldığından dolayı aşırı öğrenmenin üstesinden gelmeye yardımcı olur (Rosebrock, 2017). Havuzlama katmanında MAX ve ortalama gibi lineer veya non-lineer işlemler uygulanarak özellik haritalarının boyutu azaltılmaktadır. Bu katman $W_{input}\ x\ H_{input}\ x\ D_{input}$ giriş değerlerini kabul eder. Çıkış değerleri için 2 parametreye ihtiyaç duyar (Rosebrock, 2017).

Bunlar $F$ havuz büyüklüğü ve $S$ adım büyüklüğü değerleridir. Çıkış değerleri aşağıdaki şekilde tanımlanır.

$$W_{\text{output}} = ((W_{\text{input}} - F)/S) + 1$$
$$H_{\text{output}} = ((H_{\text{input}} - F)/S) + 1$$
$$D_{\text{output}} = D_{\text{input}} \qquad (3.31)$$

Havuzlama işleminin yapılışı ile ilgili örnek Şekil 3.15' te verilmiştir (Karlik ve Olgac, 1998).




Şekil 3.15. 5x5'lik giriş görüntüsüne 2x2 filtre ile bir ve iki adım kaymalı maksimum havuzlama işleminin uygulanması.

3.2.2.4.2.4. Hata fonksiyonu

Bu fonksiyon CNN ağlarında eğitim aşamasında rehberlik eden önemli bir yapıtaşıdır. Hedef değer ile elde edilen değer arasında ne kadarlık bir fark olduğunu ortaya koyduğundan dolayı ağın başarısında önemli bir faktör olarak karşımıza çıkmaktadır. Hata fonksiyonu bir imge sınıflandırma probleminde ağın doğru tercih yapıp yapmadığının kalitesini ortaya çıkarır. Kayıp fonksiyonu olarak CNN ağlarında kullanılan birçok fonksiyon bulunmaktadır. Burada en yaygın olarak kullanılan Hinge Loss, Cross Entropy, Softmax gibi kayıp fonksiyonları ele alınacaktır (Rosebrock, 2017).

3.2.2.4.2.4.1. Hing hata

Bu fonksiyon genellikle SVM gibi geniş marjinli sınıflandırıcıların eğitiminde kullanılır. Genel olarak aşağıdaki şekilde tanımlanır (Krause ve ark., 2013).

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1) \quad (3.32)$$

Burda $s_j$ elde edilen skor, $s_{yi}$ ise hedef skor olarak ifade edilir. Burada da max kullanılmasından dolayı negatif değerler elemine edilir.

3.2.2.4.2.4.2. Çapraz entropi ve softmax

Çapraz entropi diğer bir adıyla log loss fonksiyonu sınıflandırma modelinin performansının ölçülmesinde çıkış değerlerinin 0 ile 1 arasında dağılım olasılığını hesaplayan bir fonksiyondur. Dolayısıyla yukarıda Hinge fonksiyonunda tanımlanan $s_j$ ve $s_y$ değerleri kullanılarak çapraz entropi genel ifadesi aşağıdaki şekilde ifade edilmektedir.

$$L_i = -\log(e^{s_{y_i}} / \sum_j e^{s_j}) \quad (3.33)$$

Her çıktı için bu olasılıkların toplanması ile softmax fonksiyonu elde edilir (Rosebrock, 2017).

$$L_i = \frac{1}{N}\sum_{i=1}^{N} L_i \tag{3.34}$$

Softmax fonskiyonu, negatif değerleri elemine eden bir özelliğe sahiptir. Kısaca açıklayacak olursak her girdi değeri için elde edilen çıkış değerlerinin olasılığının exponensiyali hesaplanır. Bu değerler daha sonra normalleştirilir. Normalleştirilen olasılıkların eksi logaritması alınır. Tüm veri seti üzerinde bu işlem gerçekleştirilerek en düşük kayıp fonksiyonuna sahip çıktı elde edilir (Rosebrock, 2017).

3.2.2.4.2.5. Düzenlileştirme

Aşırı öğrenme CNN ağlarında göz önünde bulundurulması gereken bir problemdir. Bu problemin üstesinden gelmek için çeşitli düzenlileştirme yöntemleri mevcuttur. Düzenlileştirme modelin henüz görmediği test verileri üzerinde doğru sonuçlar verebilmesi için bir genelleştirme sağlar. Buradaki amaç ağın en iyi şekilde eğitilmesini sağlamak ve sonuçta ağın tanımadığı test verileri için en fazla doğruluk oranını elde etmektir. Bu bölümde lp-norm, dropout gibi düzenlileştirme yöntemleri tanıtılacaktır.

3.2.2.4.2.5.1. Lp - norm düzenlileştirme

Bu yöntemde Li gibi hata fonksiyonuna "karmaşıklık cezası" eklenerek modelin esnek bir yapıya sahip olması için orta bir yol aranır (LeCun ve ark., 2012)

$$L_i = \frac{1}{N}\sum_{i=1}^{N} L_i + \lambda R(W) \tag{3.35}$$

Buradaki ilk terim yukarıda bahsedilen softmax hata fonksiyonudur. İkinci terim ise düzenlileştirme cezası terimidir. Burdaki $\lambda$ uygulanan düzenlileştirme miktarını belirler. $w$ ağdaki ağırlık değerleridir.

Şunu da belirtelim ki aşırı düzenlileştirme istenmeyen bir durumdur. L1 ve L2 normda 2 tane düzenlileştirme cezasının yer aldığı düzenlileştirme mevcuttur.

L1 için metot:

$$R(W) = \sum_i \sum_j |W_{i,j}| \tag{3.36}$$

L2 için:

$$R(W) = \sum_i \sum_j W_{i,j}^2 \tag{3.37}$$

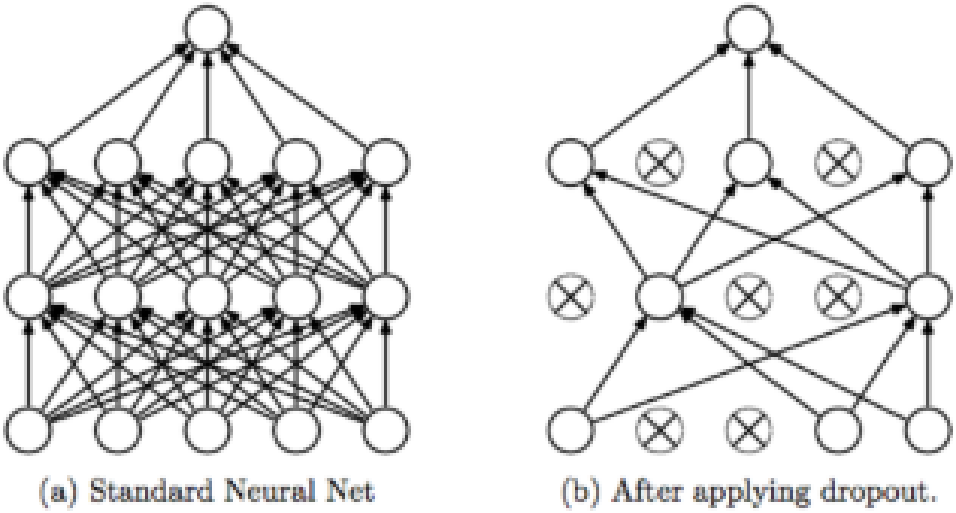
Bir de bunların birlikte kullanıldığı ElasticNet formu mevcuttur (Rosebrock, 2017).

$$R(W) = \sum_i \sum_j \beta W_{i,j}^2 + |W_{i,j}| \tag{3.38}$$

3.2.2.4.2.5.2. Dropout

Dropout katmanları sinir ağlarında çok özel bir işleve sahiptir. Bu katman basit tabir ile tam bağlantı katmanında rastgele seçilen bazı nöron bağlantılarının koparılması demektir. Bu sayede aşırı öğrenme sorunu azaltılarak bir düzenlileştirme sağlanmaktadır. Ayrıca bu katman sadece eğitim sırasında kullanılmaktadır. Test sırasında kullanılmamaktadır (Srivastava ve ark., 2014).

Dropout işleminden sonra tam bağlantı katmanındaki bağlantılar Şekil 3.16' da gösterilmektedir (Srivastava ve ark., 2014).




Şekil 3.16. Standart Yapay Sinir Ağı (soldaki), dropout işlemi uygulandıktan sonraki yapı (sağdaki).

# 4. BULGULAR

Bu tez kapsamında farklı sinir ağı mimarileri KDD-99 veri setinden elde edilen 41 nitelik üzerinde uygulanmıştır. Öncelikle veri seti modeller için uygun hale getirilmiştir. Bunun için öncelikle kategorik değişkenler nümerik değişkenlere dönüştürülmüştür. Evrişimsel sinir ağlarını uygulayabilmek için veri seti resimlere dönüştürülmüştür. Bu resimlerin her birinin boyutu $32x32$' lik matris olarak ayarlanmıştır. Daha sonra bu sinir ağlarının başarı oranları karşılaştırılmıştır.

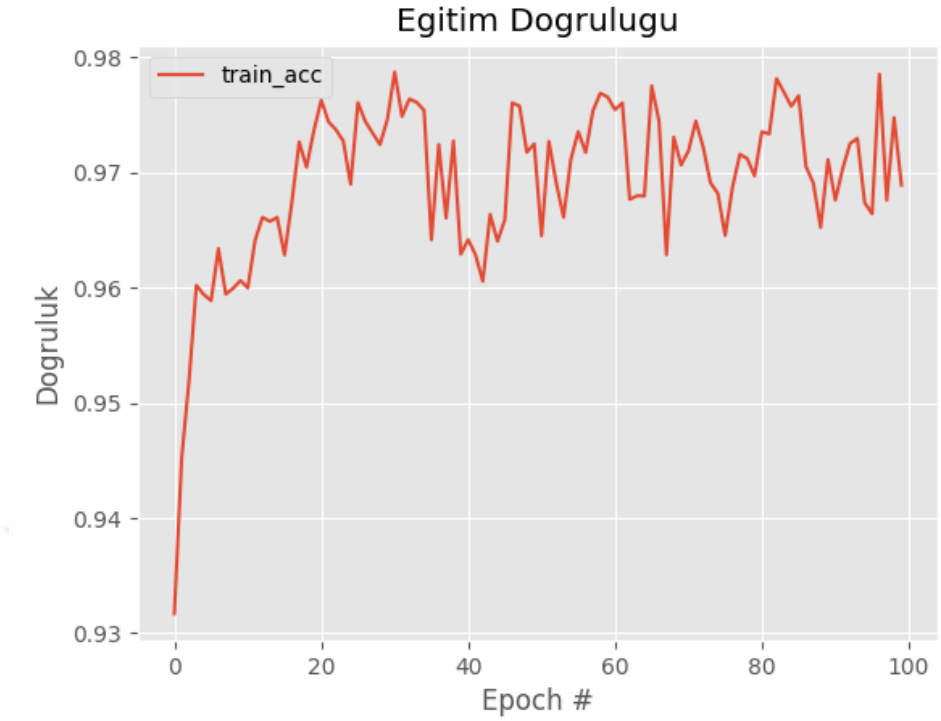
KDD-99 veri seti toplam 137823 kayıt içermektedir. Bu kayıtlardan 68327 adedi normal trafik içeren yani sınıf değişkeni 0 olan kayıtlarken, geri kalan 69496 adedi anormal trafik içeren yani sınıf değişkeni 1 olan kayıtlara aittir. Bu very seti sınıf değişkeni ile birlikte toplam 42 niteliğe sahiptir. Bu nitelikler tezin giriş bölümünde detaylı bir şekilde açıklanmıştır. Veri setinde herhangi bir 'null' değer bulunmamaktadır. 429 tekrar eden kayıt bulunmaktadır. Veri seti tekrar eden kayıtlardan temizlenmiştir.

Şekil 4.2' de 3 katmanlı ileri beslemeli yapay sinir ağının başarı sonuçları gösterilmiştir. Bu sinir ağında aktivasyon fonksiyonu olarak girdi ve gizli katmanda ReLU, çıktı katmanında Sigmoid fonksiyonu kullanılmıştır. Veri setinin yüzde 66.6' sı eğitim verisi, geriye kalanı ise test verisi olarak ayrılmıştır. Hata fonksiyonu olarak da binary cross entropy kullanılmıştır. Ağırlıklar düzgün dağılımdan atanmıştır daha sonra ağırlıkların güncellenmesi için Adam optimizasyonu kullanılmıştır. Şekil 4.1' de 100 epoch sonucunda elde edilen sonuçlar verilmiştir.

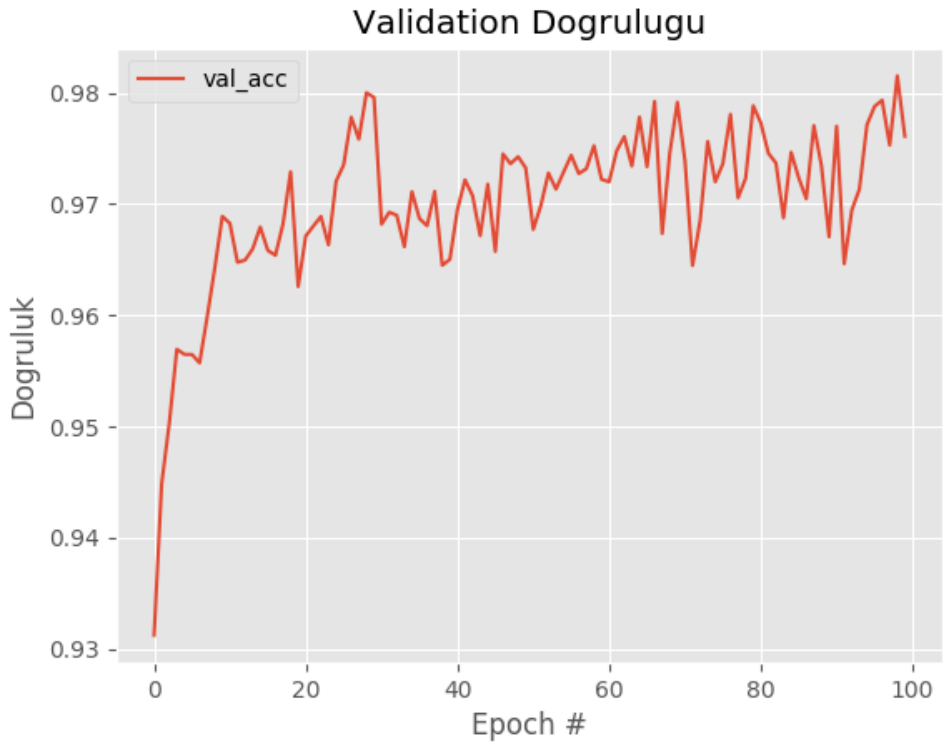
| | Precision | Recall | F1-Score | Support |
|---|---|---|---|---|
| *0* | 0.94 | 0.98 | 0.96 | 17282 |
| *1* | 0.98 | 0.94 | 0.96 | 17174 |

Şekil 4.1. Epoch işlemiyle elde edilen sonuçlar.

Her bir iterasyonda doğruluk değerleri Şekil 4.2' de verilmiştir.



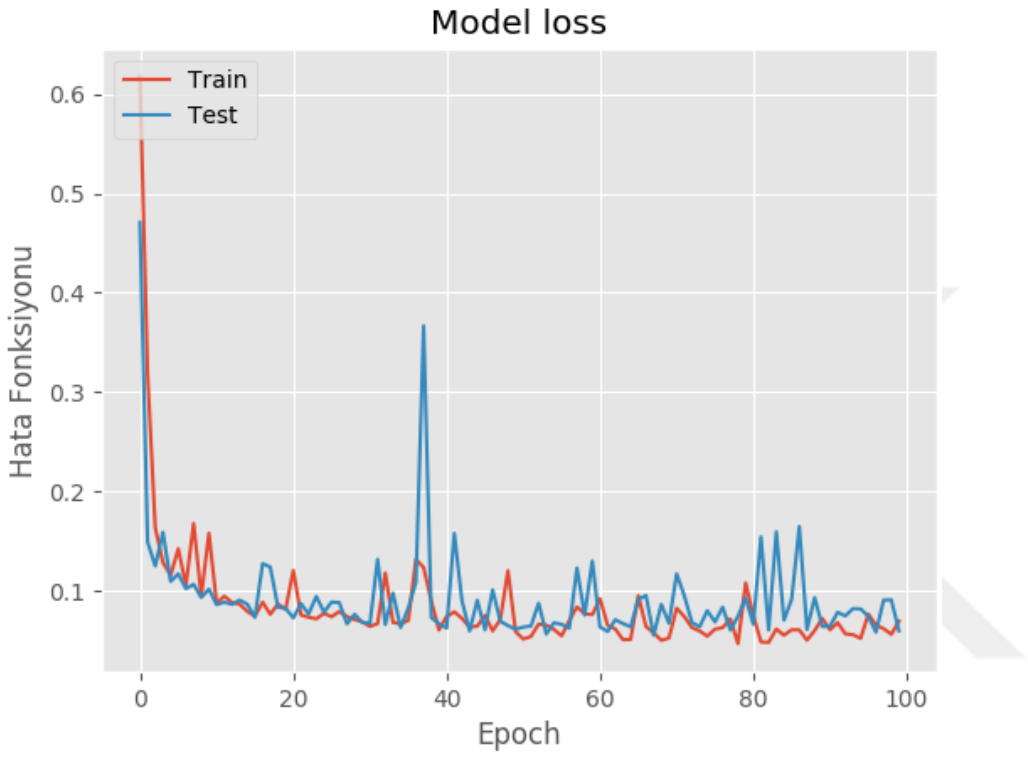

Şekil 4.2. 3 katmanlı yapay sinir ağı eğitim doğruluğu.




Şekil 4.3. 3 katmanlı yapay sinir ağı test doğruluğu.

Eğitim ve test verilerindeki doğruluk oranlarına bakıldığında yetersiz öğrenmenin ve aşırı uyumun olmadığı gözlemlenebilir (Şekil 4.3).

Şekil 4.4' te bulunan grafikte optimizasyon adımlarında hata fonksiyonunun değerleri verilmiştir.




Şekil 4.4. Hata fonksiyonu grafiği.

Evrişimsel sinir ağları 2 boyutlu veriler üzerinde diğer sinir ağlarına nazaran başarılı sonuçlar vermemektedir. KDD-99 veri seti üzerinde direkt olarak uygulanan evrişimsel sinir ağları yüzde 10' un altında bir başarı oranına sahiptir. Bu durum konvolüsyon işleminin veri setinin nitelik sayısını azaltmasından kaynaklanmaktadır. Evrişimsel sinir ağları 3 boyutlu veriler üzerinde daha başarılı sonuçlara sahiptir. Bu nedenle bu tez kapsamında veri seti resimlere dönüştürülüp daha sonra evrişimsel sinir ağı uygulanmıştır. Bu düzenleme, evrişimsel sinir ağlarının bu veri setinde başarı oranını yüzde 90' ın üzerine çıkarmıştır. Veri seti resimlere dönüştürülürken değerler $0 - 255$ aralığına ölçeklenmiştir. Çizelge 4.1' te verilerin dönüştürülmesinden sonra elde edilen resimlere birer örnek verilmiştir.

Çizelge 4.1. Örnek verilerin resimlere dönüştürüldükten sonraki görünümleri.

| Kriter |  |  |  |  |  |
|---|---|---|---|---|---|
| duration | 0 | 0 | 0 | 0 | 315 |
| protocol_type | 1 | 1 | 1 | 2 | 2 |
| service | 24 | 24 | 49 | 44 | 44 |
| flag | 9 | 9 | 5 | 9 | 9 |
| src_bytes | 199 | 232 | 0 | 146 | 146 |
| dst_bytes | 420 | 8153 | 0 | 0 | 105 |
| land | 0 | 0 | 0 | 0 | 0 |
| wrong_fragment | 0 | 0 | 0 | 0 | 0 |
| urgent | 0 | 0 | 0 | 0 | 0 |
| hot | 0 | 0 | 0 | 0 | 0 |
| num_failed_logins | 0 | 0 | 0 | 0 | 0 |
| logged_in | 1 | 1 | 0 | 0 | 0 |
| num_compromised | 0 | 0 | 0 | 0 | 0 |
| root_shell | 0 | 0 | 0 | 0 | 0 |
| su_attempted | 0 | 0 | 0 | 0 | 0 |
| num_root | 0 | 0 | 0 | 0 | 0 |
| num_file_creations | 0 | 0 | 0 | 0 | 0 |
| num_shells | 0 | 0 | 0 | 0 | 0 |
| num_access_files | 0 | 0 | 0 | 0 | 0 |
| num_outbound_cmds | 0 | 0 | 0 | 0 | 0 |
| is_host_login | 0 | 0 | 0 | 0 | 0 |
| is_guest_login | 0 | 0 | 0 | 0 | 0 |
| count | 30 | 5 | 123 | 13 | 3 |
| srv_count | 32 | 5 | 6 | 1 | 2 |
| serror_rate | 0 | 0.2 | 1 | 0 | 0 |
| srv_serror_rate | 0 | 0.2 | 1 | 0 | 0 |
| rerror_rate | 0 | 0 | 0 | 0 | 0 |
| srv_rerror_rate | 0 | 0 | 0 | 0 | 0 |
| same_srv_rate | 1 | 1 | 0.05 | 0.08 | 0.67 |
| diff_srv_rate | 0 | 0 | 0.07 | 0.15 | 0.67 |
| srv_diff_host_rate | 0.09 | 0 | 0 | 0 | 0 |
| dst_host_count | 255 | 30 | 255 | 255 | 117 |
| dst_host_srv_count | 255 | 255 | 26 | 1 | 2 |
| dst_host_same_srv_rate | 1 | 1 | 0.1 | 0 | 0.02 |
| dst_host_diff_srv_rate | 0 | 0 | 0.05 | 0.6 | 0.54 |
| dst_host_same_src_port_rate | 0 | 0.03 | 0 | 0.88 | 0.78 |
| dst_host_srv_diff_host_rate | 0 | 0.04 | 0 | 0 | 0 |
| dst_host_serror_rate | 0 | 0.03 | 1 | 0 | 0 |
| dst_host_srv_serror_rate | 0 | 0.01 | 1 | 0 | 0 |
| dst_host_rerror_rate | 0 | 0 | 0 | 0 | 0 |
| dst_host_srv_rerror_rate | 0 | 0.01 | 0 | 0 | 0 |

Evrişimsel sinir ağlarının elde edilen veri seti üzerindeki başarı oranlarını karşılaştırmak için ilk olarak diğerlerine nispeten daha küçük bir mimariye sahip bir evrişimsel sinir ağı kullanılmıştır. Veri setinin % 25' lik kısmı test verisi olarak ayrılmıştır. Bu ağın mimarisi GİRDİ ==> CONV ==> RELU ==> FC olarak gösterilebilir. Şekil 4.5' te bu mimariden elde edilen başarı oranı gösterilmiştir.


Şekil 4.5. Evrişimsel sinir ağı eğitim sonuçları.

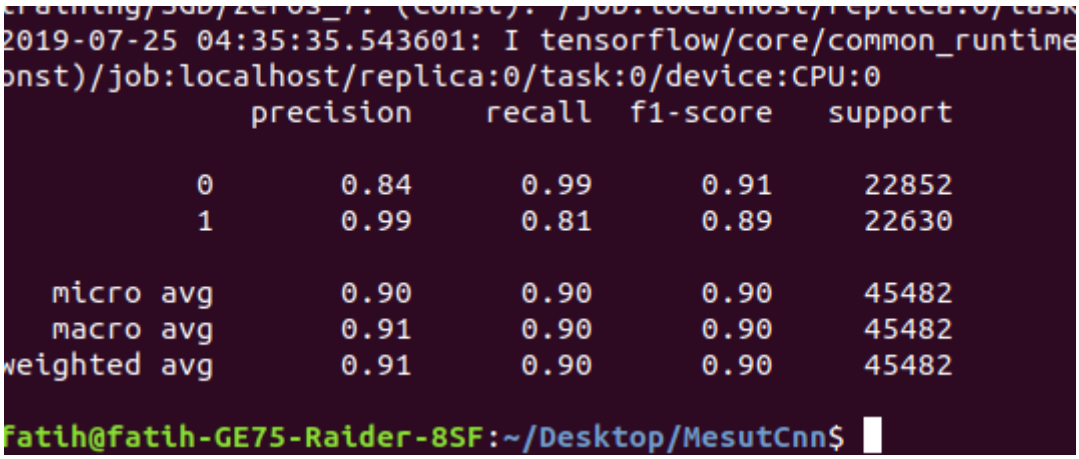
LeNet ilk olarak 1998 yılında LeCun ve ark, (2015) tarafından belgelerin sınıflandırılması amacıyla kullanılmıştır. Şekil 4.6' da LeNet mimarisi verilmiştir

| Layer | | Feature Map | Size | Kernel Size | Stride | Activation |
|---|---|---|---|---|---|---|
| Input | Image | 1 | 32x32 | - | - | - |
| 1 | Convolution | 6 | 28x28 | 5x5 | 1 | tanh |
| 2 | Average Pooling | 6 | 14x14 | 2x2 | 2 | tanh |
| 3 | Convolution | 16 | 10x10 | 5x5 | 1 | tanh |
| 4 | Average Pooling | 16 | 5x5 | 2x2 | 2 | tanh |
| 5 | Convolution | 120 | 1x1 | 5x5 | 1 | tanh |
| 6 | FC | - | 84 | - | - | tanh |
| Output | FC | - | 10 | - | - | softmax |

Şekil 4.6. LeNet Mimarisi.

Şekil 4.7' de LeNet mimarisinden elde edilen başarı oranı gösterilmiştir.

Şekil 4.8' de her bir epoch' ta hata fonksiyonun eğitim ve test verisindeki değerleri ve eğitim ile test verisindeki başarı oranları gösterilmiştir




Şekil 4.7. LeNet eğitim sonuçları.


Şekil 4.8. LeNet hata fonksiyonu değerleri.

VGGNet, ilk olarak Simonyan ve Zisserman tarafından 2014 çalışmalarında tanıtılmıştır. Bu çalışma çok küçük (3 × 3) filtreleri olan bir mimarinin giderek daha fazla eğitilebileceğini göstermiştir. VGGNet, tüm mimaride 3 × 3 çekirdek kullanması bakımından benzersizdir. Şekil 4.9' da VGGNet mimarisi verilmiştir

| Layer Type | Output Size | Filter Size / Stride |
|---|---|---|
| Input Image | 32 x 32 x 3 | |
| CONV | 32 x 32 x 32 | 3 x 3, K = 96 |
| CONV | 32 x 32 x 32 | 3 x 3, K = 32 |
| POOL | 16 x 16 x 32 | 2 x 2 |
| DROPOUT | 16 x 16 x 32 | |
| CONV | 16 x 16 x 64 | 3 x 3, K = 64 |
| CONV | 16 x 16 x 64 | 3 x 3, K = 64 |
| POOL | 8 x 8 x 64 | 2 x 2 |
| DROPOUT | 8 x 8 x 64 | |
| FC | 512 | |
| DROPOUT | 512 | |
| FC | 10 | |
| softmax | 10 | |

Şekil 4.9. VGGNet mimarisi.

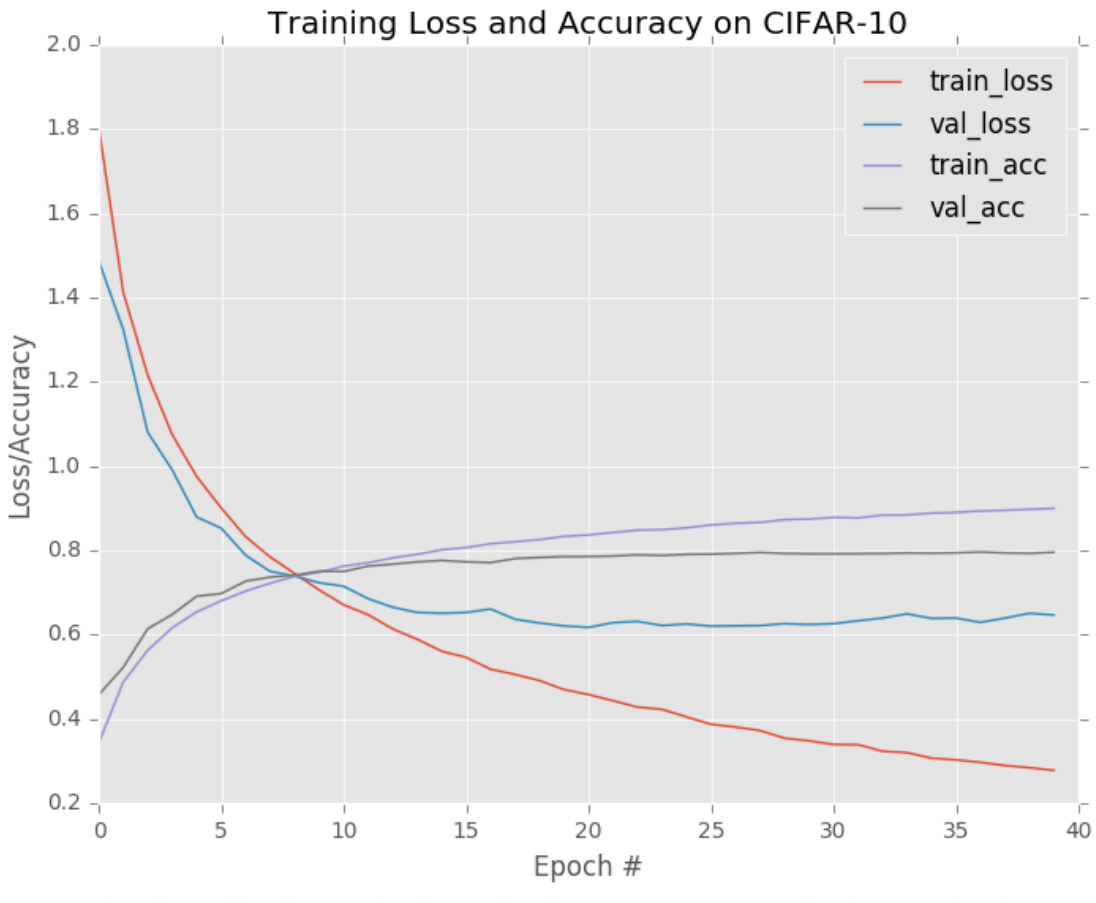
Şekil 4.10’ da VGGNet mimarisinden elde edilen başarı oranı gösterilmiştir.

Şekil 4.11’ de her bir epoch’ ta hata fonksiyonun yığın normalizasyonu ile eğitim ve test verisindeki değerleri ile eğitim ve test verisindeki başarı oranları gösterilmiştir. Şekil 4.12’ de her bir epoch’ ta hata fonksiyonun yığın normalizasyonu olmadan eğitim ve test verisindeki değerleri ile eğitim ve test verisindeki başarı oranları gösterilmiştir

```
019-07-25 00:37:00.197843: I tensorflow/core/common_runtime/
GD/zeros_3: (Const)/job:localhost/replica:0/task:0/device:CP
              precision    recall  f1-score   support

           0       0.96      0.97      0.96     17282
           1       0.97      0.95      0.96     17174

   micro avg       0.96      0.96      0.96     34456
   macro avg       0.96      0.96      0.96     34456
eighted avg       0.96      0.96      0.96     34456

bound method Sequential.summary of <keras.engine.sequential.
0d31ed690>>
```

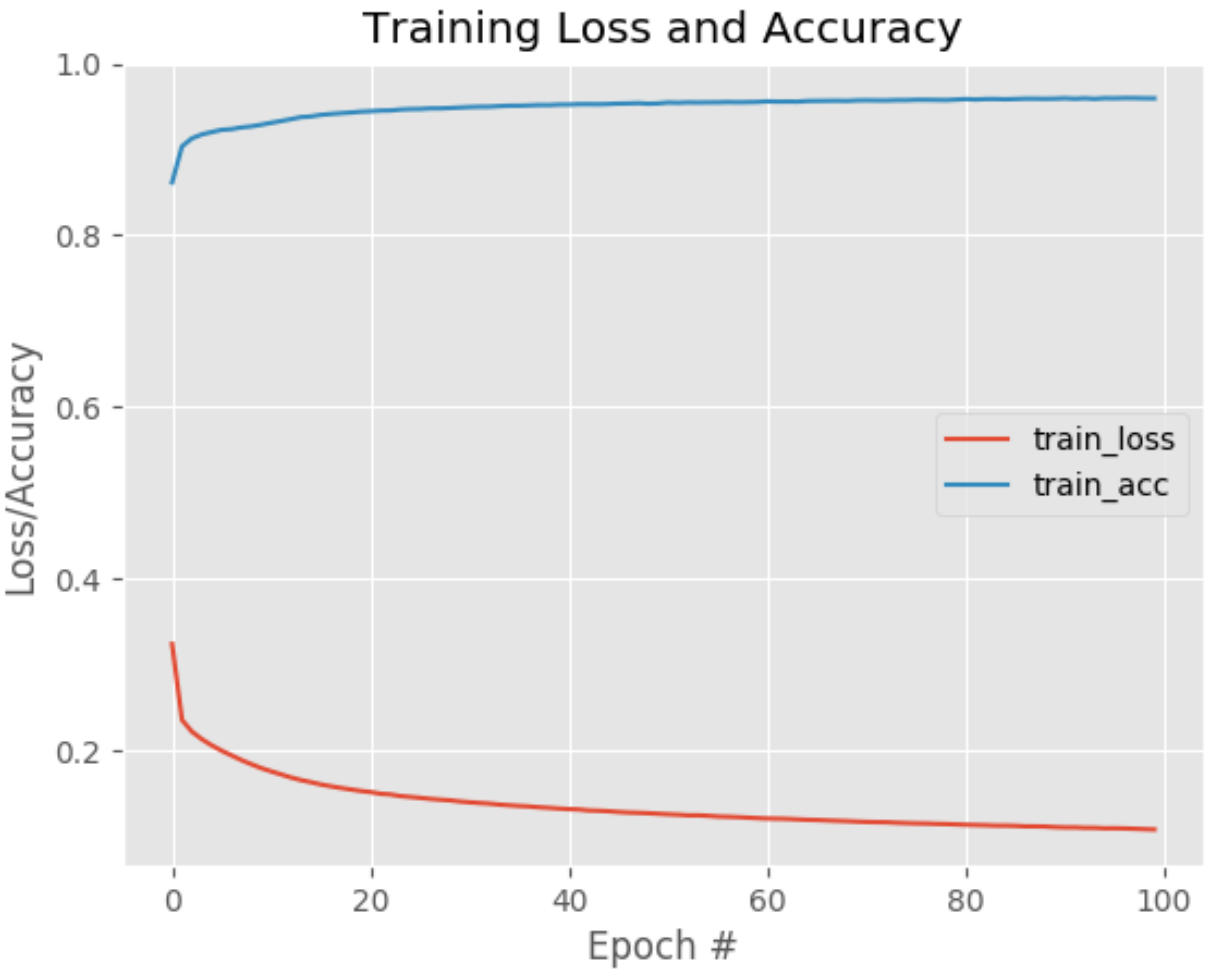
Şekil 4.10 VGGNet başarı oranları.


Şekil 4.11. VGGNet hata fonksiyonu değerleri (yığın normalizasyonu ile).




Şekil 4.12. VGGNet hata fonksiyonu değerleri (yığın normalizasyonu olmadan).

Bununla birlikte, veri setinin % 33' lük kısmı test verisi olarak ayrılarak da eğitim gerçekleştirilmiştir. Eğitim sonucunda elde edilen sonuçlar Şekil 4.13 ve Şekil 4.14' te gösterilmiştir.




Şekil 4.13. Evrişimsel sinir ağı ile eğitim verisi doğruluk grafiği (% 33’ lük test verisi ile).


Şekil 4.14. Evrişimsel sinir ağı ile test verisi doğruluk grafiği (% 33’ lük test verisi ile).

# 5. SONUÇ

Ağ saldırı tespit sistemleri için yapılan bu çalışmada KDD-99 veri seti üzerinde derin öğrenme algoritmalarının performansları karşılaştırılmıştır. İlk olarak, KDD-99 veri seti bir dizi veri düzenleme prosedüründen geçirilip, derin öğrenme algoritmalarının daha başarılı olacağı ve ona uygun girdi olabilecek şekline dönüştürülmüştür. Daha sonra ileri beslemeli yapay sinir ağı mimarileri KDD-99 veri seti üzerinde uygulanmıştır. Yapılan birçok denemeden sonra en başarılı olan ağ mimarisi ve parametreleriyle birlikte başarı oranı % 97.8 olarak bulunmuştur. Uygulamalar 32 gb ram bellek, RTX-2070 ekran kartı, i7 işlemci ve ubuntu işletim sistemine sahip bilgisayarda gerçekleştirilmiştir. Evrişimsel sinir ağları, iki boyutlu veriler üzerinde başarılı olmayan sonuçlara sahiptir. Bu çalışmada literatürde bulunandan farklı olarak öncelikle veri seti resimlere dönüştürülmüş ve LeNet, VGGNet gibi literatürde sıklıkla kullanılan evrişimsel sinir ağları mimarileri uygulanmıştır. Öncelikle baz model olarak dar bir evrişimsel sinir ağı modeli uygulanmış ve % 90,1 başarı oranı elde edilmiştir. Daha sonra, daha büyük bir yapıya sahip LeNet mimarisi veri setine uygulanmış ve % 91,4 gibi daha başarılı bir oran elde edilmiştir. Son olarak 3 x 3' lük konvolüsyonlar içeren VGGNet mimarisi uygulanarak, % 96 gibi daha başarılı bir oran elde edilmiştir. Bu uygulamalar, % 25' lik test verisi kullanılarak gerçekleştirilmiştir. Test verisi % 33 yapılınca da bu sonuçlara yakın sonuçlar çıkmaktadır.

Çalışmanın sonucunda, ileri beslemeli yapay sinir ağlarının uygun düğüm sayısı ve parametrelerle diğer modellere göre daha başarılı sonuçlar verdiği gözlemlenmiştir. Bunun sebebi ileri beslemeli yapay sinir ağlarının, evrişimsel sinir ağlarına göre daha fazla bağlantıya sahip olmasıdır. Yine de bu tip bir veride, veriyi resimlere dönüştürüp bilinen evrişimsel sinir ağları mimarilerine uygulamak, beklenenden daha başarılı sonuçlar üretmiştir.

**KAYNAKLAR**


Ahmed, M., Mahmood, A. N., & Hu, J., 2016. A survey of network anomaly detection techniques. ***Journal of Network and Computer Applications,*** **60:** 19-31.

Alom, M. Z., Bontupalli, V., Taha, T. M. 2015. Intrusion detection using deep belief networks. ***In 2015 National Aerospace and Electronics Conference (NAECON).*** 2015, USA. 339-344.

Alpaydin, E., 2016. Yapay Öğrenme, Boğaziçi Üniversitesi Yayınevi, İstanbul. 37-259.

Bontemps, L., McDermott, J., Le-Khac, N. A. 2016. Collective anomaly detection based on long short-term memory recurrent neural networks. ***In International Conference on Future Data and Security Engineering.*** November 2016, Vietnam. 141-152.

Brownlee, J., 2014. Machine learning mastery. ***https://machinelearningmastery.com/discover-feature-engineering-how-to-engineer-features-and-how-to-get-good-at-it.*** Erişim Tarihi : 21.07.2019.

Chiba, Z., Abghour, N., Moussaid, K., El Omri, A., Rida, M., 2018. A novel architecture combined with optimal parameters for back propagation neural networks applied to anomaly network intrusion detection. ***Computers & Security,*** **75:** 36-58.

Clevert, D. A., Unterthiner, T., Hochreiter, S., 2015. ***Fast and Accurate Deep Network Learning by Exponential Linear Units (elus)***. arXiv preprint arXiv:1511.07289.

Daya, B., 2013. ***Network Security: History, Importance, and Future.*** University of Florida Department of Electrical and Computer Engineering, 4.

Denning, D. E., 1987. An intrusion-detection model. ***IEEE Transactions on Software Engineering,*** **2:** 222-232.

Dhanabal, L., Shantharajah, S. P., 2015. A study on NSL-KDD dataset for intrusion detection system based on classification algorithms. ***International Journal of Advanced Research in Computer and Communication Engineering,*** **4**(6): 446-452.

Farahnakian, F., Heikkonen, J., 2018. A deep auto-encoder based approach for intrusion detection system. ***In 2018 20th International Conference on Advanced Communication Technology (ICACT).*** February 2018, Finland. 178-183.

Flach, P. 2012. ***Machine Learning: The Art and Science of Algorithms That Make Sense of Data***. Cambridge University Press.

Ghanbari, M., Kinsner, W., Ferens, K. 2017. Detecting a distributed denial of service attack using a pre-processed convolutional neural network. ***In 2017 IEEE Electrical Power and Energy Conference (EPEC)***. October 2017, Canada. 1-6.

Goodfellow, I., Bengio, Y., Courville, A., 2016. ***Deep learning.*** MIT press.

Hahnloser, R. H., Sarpeshkar, R., Mahowald, M. A., Douglas, R. J., Seung, H. S. 2000. Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. ***Nature, 405***(6789): 947.

Hahnloser, R. H., Seung, H. S. 2001. Permitted and forbidden sets in symmetric threshold-linear networks. ***In Advances in Neural Information Processing Systems,*** 2001, USA. 217-223.

Heaton, J. (2018). Ian goodfellow, yoshua bengio, and aaron courville: Deep learning.

Hebb, D. O. 2005. **The organization of behavior: *A neuropsychological theory.*** Psychology Press.

Javaid, A., Niyaz, Q., Sun, W., Alam, M. 2016. A deep learning approach for network intrusion detection system. ***In Proceedings of the 9th EAI International Conference on Bio-inspired Information and Communications Technologies (formerly BIONETICS).*** May 2016, USA. 21-26.

Karlik, B., & Olgac, A. V. (2011). Performance analysis of various activation functions in generalized MLP architectures of neural networks. *International Journal of Artificial Intelligence and Expert Systems*, 1(4), 111-122.

Kartal, E., 2015 ***Sınıflandırmaya dayalı makine öğrenmesi teknikleri ve kardiyolojik risk değerlendirmesine ilişkin bir uygulama.*** İstanbul Üniversitesi, Fen Bilimleri Enstitüsü, İstanbul.

Krause, J., Stark, M., Deng, J., & Fei-Fei, L. (2013). 3d object representations for fine-grained categorization. In *Proceedings of the IEEE International Conference on Computer Vision Workshops* (pp. 554-561).

Kwon, D., Kim, H., Kim, J., Suh, S. C., Kim, I., Kim, K. J.,2017. A survey of deep learning-based network anomaly detection. ***Cluster Computing.*** 1-13.

LeCun, Y., Bengio, Y., Hinton, G., 2015. Deep learning. ***Nature,* 521** (7553): 436.

LeCun, Y., Bottou, L., Orr, G. B., & Müller, K. R., 1998. Neural networks: Tricks of the trade. *Springer Lecture Notes in Computer Sciences*, **1524**(5-50), 6.

Liu, Y., Liu, S., Zhao, X., 2017. Intrusion detection algorithm based on convolutional neural network. ***DEStech Transactions on Engineering and Technology Research, (iceta).***

Maas, A. L., Hannun, A. Y., Ng, A. Y., 2013. Rectifier nonlinearities improve neural network acoustic models. ***In Proc. Icml,* 30** (1): 3.

Maurizio, M., 2011. Data Mining Concepts And Techniques. ***http://www.dsi.unive.it/~marek/files/06%20-%20datamining.pdf***. Erişim tarihi: 21.07.2019.

Özkan, İ. N. İ. K., & Ülker, E. Derin Öğrenme ve Görüntü Analizinde Kullanılan Derin Öğrenme Modelleri. *Gaziosmanpaşa Bilimsel Araştırma Dergisi*, *6*(3), 85-104.

Ravipati, R. D., & Abualkibash, M. (2019). Intrusion Detection System Classification Using Different Machine Learning Algorithms on KDD-99 and NSL-KDD Datasets-A Review Paper. *International Journal of Computer Science & Information Technology (IJCSIT) Vol*, *11*.Mitchell, T. M., 1997. Does machine learning really work?. ***AI magazine***, ***18*** (3): 11-11.

Mohri, M., Rostamizadeh, A., Talwalkar, A. 2012. ***Foundations Of Machine Learning.*** MIT press.

Powers, S. T., He, J. 2008. A hybrid artificial immune system and Self Organising Map for network intrusion detection. ***Information Sciences,* 178** (15): 3024-3042.

Rosebrock, A., September 2017. ***Deep Learning for Computer Vision with Python: Starter Bundle.*** PyImageSearch, USA, 1, USA. 330.

Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, *15*(1), 1929-1958.

Tavallaee, M., Bagheri, E., Lu, W., Ghorbani, A. A., 2009. A detailed analysis of the KDD CUP 99 data set. ***In 2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications,*** July 2009, Canada. 1-6.

Tran, N. N., Sarker, R., & Hu, J., 2017. An Approach for Host-Based Intrusion Detection System Design Using Convolutional Neural Network. ***In International Conference on Mobile Networks and Management***. December 2017, Australia. 116-126.

Wang, B., Li, F., Zhang, S., 2009. Research on intrusion detection based on campus network. ***In 2009 Third International Symposium on Intelligent Information Technology Application***. November 2009. 468-471.

Wang, G., Hao, J., Ma, J., & Huang, L. (2010). A new approach to intrusion detection using Artificial Neural Networks and fuzzy clustering. ***Expert systems with applications, 37***(9): 6225-6232.

Wang, S. H., Phillips, P., Sui, Y., Liu, B., Yang, M., & Cheng, H. (2018). Classification of Alzheimer's disease based on eight-layer convolutional neural network with leaky rectified linear unit and max pooling. *Journal of medical systems*, *42*(5), 85.

Ng, A., 2000. CS229 Lecture notes. http://docs.apwg.org/reports/apwg_trends_report_q4_2016.pdf. Erişim tarihi: 21.07.2019.

Rosenblatt, F., 1958. The perceptron: a probabilistic model for information storage and organization in the brain. ***Psychological review,*** **65**(6): 386.

# ÖZ GEÇMİŞ

1989 yılında Van'da doğdu. İlk, orta ve lise öğrenimini Van'da tamamladı. 2013 yılında Fırat Üniversitesi Bilgisayar Mühendisliği Bölümü'nden mezun oldu. 2013 yılında Gıda Tarım ve Hayvancılık Bakanlığında Bilgisayar Mühendisi olarak çalışmaya başladı. 2014 yılında Van Yüzüncü Yıl Üniversitesine naklen geçti. 2015 yılında Van Yüzüncü Yıl Üniversitesi Fen Bilimleri Enstitüsü, İstatistik Anabilim Dalı'nda Yüksek Lisans eğitimine başladı.

**T.C**
**VAN YÜZÜNCÜ YIL ÜNİVERSİTESİ**
**FEN BİLİMLERİ ENSTİTÜSÜ**
**LİSANSÜSTÜ TEZ ORİJİNALLİK RAPORU**

**Tarih:** 11/09/2019

Tez Başlığı / Konusu: Ağ Tehdit Algılamada Derin Öğrenme Uygulamaları

Yukarıda başlığı/konusu belirlenen tez çalışmamın Kapak sayfası, Giriş, Ana bölümler ve Sonuç bölümlerinden oluşan toplam 73 sayfalık kısmına ilişkin, 10/09/2019 tarihinde şahsım/tez danışmanım tarafından Ithenticate intihal tespit programından aşağıda belirtilen filtreleme uygulanarak alınmış olan orijinallik raporuna göre, tezimin benzerlik oranı % 5 (Beş) tir.

Uygulanan filtreler aşağıda verilmiştir:

- Kabul ve onay sayfası hariç,
- Teşekkür hariç,
- İçindekiler hariç,
- Simge ve kısaltmalar hariç,
- Gereç ve yöntemler hariç,
- Kaynakça hariç,
- Alıntılar hariç,
- Tezden çıkan yayınlar hariç,
- 7 kelimeden daha az örtüşme içeren metin kısımları hariç (Limit inatch size to 7 words)

Van Yüzüncü Yıl Üniversitesi Lisansüstü Tez Orijinallik Raporu Alınması ve Kullanılmasına İlişkin Yönergeyi inceledim ve bu yönergede belirtilen azami benzerlik oranlarına göre tez çalışmamın herhangi bir intihal içermediğini; aksinin tespit edileceği muhtemel durumda doğabilecek her türlü hukuki sorumluluğu kabul ettiğimi ve yukarıda vermiş olduğum bilgilerin doğru olduğunu beyan ederim.

Gereğini bilgilerinize arz ederim.

11/09/2019

Adı Soyadı: Mesut KAPAR

Öğrenci No: 159102182

Anabilim Dalı: İstatistik

Programı: Tezli Yüksek Lisans

Statüsü: Y. Lisans ☒ Doktora ☐

**DANIŞMAN ONAYI**
UYGUNDUR

Dr. Öğr. Üyesi Murat CANAYAZ

**ENSTİTÜ ONAYI**
UYGUNDUR

Prof. Dr. Suat ŞENSOY
Enstitü Müdürü