Derin Öğrenme
Ders Notu

Dr. Cahit Karakuş

“Balbiti”

İçindekiler

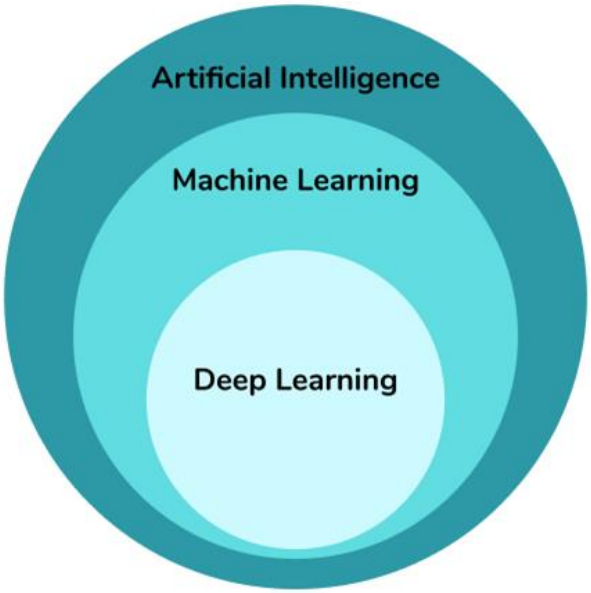
1. Giriş

Derin öğrenme, tamamen yapay sinir ağlarına dayanan bir makine öğrenimi alt kümesidir. Sinir ağları insan beynini taklit edecek şekilde tasarlandığından, derin öğrenme de aynı şekilde bir insan beyni taklitçisidir.

Derin öğrenmede her şeyi açıkça programlamak zorunda değiliz. Bir modeli eğitim veri kümesi üzerinde eğitiriz ve model, test ve doğrulama veri kümesinde de neredeyse doğru tahminde bulunana kadar onu doğaçlama yaparız.

Derin öğrenme modelleri, programcıdan yalnızca küçük bir girdi ile kendi başlarına doğru özelliklere odaklanabilir ve boyutsallık sorununu çözmede oldukça faydalıdır. Derin öğrenme yeni bir kavram değildir. Bir süredir ortalıkta dolaşıyor. Şu anda sahip olduğumuz kadar fazla işlem gücüne veya veriye sahip olmadığımız için bu günlerde tüm işlem gücü son 20 yılda muazzam bir şekilde arttığından, sahneye derin öğrenme ve makine öğrenimi girdi.

1960'ların ortalarında bir derin öğrenme ağı üzerinde çalışırken, Alexey Grigorevich Ivakhnenko ilk yayını yayınladı. Esasen, çok sayıda doğrusal olmayan işlem birimi kullanarak özellik çıkarma ve dönüştürme yapan bir makine öğrenimi alt kümesidir. Sonraki katmanların her biri, bir önceki katmanın çıktısını girdi olarak kullanır. Derin öğrenme, bilgisayarla görme, konuşma tanıma, doğal dil işleme vb. dahil olmak üzere çeşitli uygulamalar için uygundur.



Yukarıdaki görüntü, Makine Öğreniminin Yapay Zekanın bir alt kümesi olduğunu ve Derin Öğrenmenin, Makine Öğreniminin bir alt kümesi olduğunu göstermektedir.

Derin öğrenme (aynı zamanda derin yapılandırılmış öğrenme, hiyerarşik öğrenme ya da derin makine öğrenmesi) bir veya daha fazla gizli katman içeren yapay sinir ağları ve benzeri makine öğrenme algoritmalarını kapsayan çalışma alanıdır. Yani en az bir adet yapay sinir ağının (YSA) kullanıldığı ve birçok algoritma ile, bilgisayarın eldeki verilerden yeni veriler elde etmesidir. Derin öğrenme gözetimli, yarı gözetimli veya gözetimsiz olarak gerçekleştirilebilir. Derin yapay sinir ağları pekiştirmeli öğrenme yaklaşımıyla da başarılı sonuçlar vermiştir.

Derin öğrenme, Yüksek bilgi işleme gücü ve büyük veri kümeleriyle birlikte katmanlı yapay sinir ağlarının güçlü matematiksel modelleri oluşturabildiği bir makine öğrenimi alt kümesidir. Verinin yapısına göre hangi parametrelere ne ağırlık verileceğini kendisi keşfetmektedir. Derin öğrenme, ham girdiden daha yüksek seviyedeki özellikleri aşamalı olarak çıkarmak için birden çok katman kullanan bir makine öğrenme algoritmaları sınıfıdır.

Modern derin öğrenme modellerinin çoğu, yapay sinir ağlarına, özellikle de Konvolüsyonel Sinir Ağlarına (CNN - Convolutional Neural Networks) dayanmaktadır, ancak aynı zamanda derin düşünce ağları ve derin düğümler gibi derin üretken modellerde katmansal olarak düzenlenmiş öneri formülleri veya gizli değişkenleri de içerebilirler. Yapay sinir ağları (YSA) biyolojik sistemlerde bilgi işleme ve dağıtılmış iletişim düğümlerinden esinlenmiştir. YSA'ların biyolojik beyinlerden çeşitli farklılıkları vardır. Özellikle, sinir ağları statik ve sembolik olma eğilimindeyken, çoğu canlı organizmanın biyolojik beyni dinamik (plastik) ve analogdur.

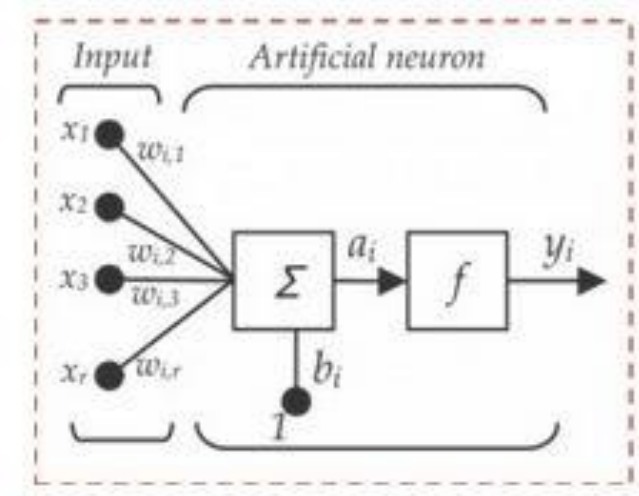
Derin öğrenme, her seviye girdi verilerini biraz daha soyut ve bileşik bir temsile dönüştürmeyi öğrenir. Bir görüntü tanıma uygulamasında, ham girdi bir piksel matrisi olabilir; birinci temsili katman pikselleri soyutlayabilir ve kenarları kodlayabilir; ikinci katman kenar düzenlemelerini oluşturabilir ve kodlayabilir; üçüncü katman bir burnu ve gözleri kodlayabilir; ve dördüncü katman görüntünün bir yüz içerdiğini tanıyabilir. Daha da önemlisi, derin bir öğrenme süreci hangi özelliklerin hangi seviyeye en uygun şekilde yerleştirileceğini öğrenebilir. Elbette bu, elle ayarlama ihtiyacını tamamen ortadan kaldırmaz; örneğin, değişen sayıda katman ve katman boyutu farklı derecelerde soyutlama sağlayabilir.

Derin öğrenme, Yapay Zekada (AI) Makine Öğreniminin bir alt kümesidir. Bir yapay zeka, verileri işlemek, kalıplar oluşturmak ve kararlar almak için insan beynini taklit ettiğinde, bu Derin Öğrenmedir. Ham girdiden aşamalı olarak üst düzey özellikleri çıkarmak için birden çok katman kullanır. "Derin öğrenmedeki" "derin" kelimesi, verilerin dönüştürüldüğü katman sayısını ifade eder. Bu sayede yapılandırılmamış ve etiketsiz verilerden örüntüler öğrenmek mümkündür. Derin öğrenme algoritması, deneyimlerden nasıl öğrendiğimize benzer şekilde, her seferinde sonucunu iyileştiren bir görevi tekrar tekrar gerçekleştirir.

Arama motorlarında, sosyal medyada, e-ticaret platformlarında her saniye yaratılan muazzam miktarda veri, derin öğrenmeyi büyük bir potansiyele dönüştürdü. Buna ek olarak, bugün mevcut olan güçlü bilgi işlem gücü, derin öğrenmede kullanılan algoritmalar üzerinde

büyük bir etki yarattı. Dahası, kendi kendine giden arabalar, AlphaGo, sesli asistanlar gibi yapay zekadaki atılımlar, derin öğrenme sayesinde mümkün.

Derin öğrenmede, her seviye girdi verilerini biraz daha soyut ve bileşik bir temsile dönüştürmeyi öğrenir. Bir görüntü tanıma uygulamasında, ham girdi bir piksel matrisi olabilir; birinci temsil katmanı pikselleri soyutlayabilir ve kenarları kodlayabilir; ikinci katman, kenar düzenlemelerini oluşturabilir ve kodlayabilir; üçüncü katman bir burnu ve gözleri kodlayabilir; ve dördüncü katman, görüntünün bir yüz içerdiğini fark edebilir. Daha da önemlisi, derin öğrenme süreci hangi özelliklerin hangi seviyeye en uygun şekilde yerleştirileceğini kendi başına öğrenebilir. (Elbette bu, elle ayarlama ihtiyacını tamamen ortadan kaldırmaz; örneğin, değişen sayıda katman ve katman boyutu, farklı soyutlama dereceleri sağlayabilir.)



Nöronun farklı bileşenleri şu şekilde belirtilir:

• x1, x2,…, xN: Bunlar nöronun girdileridir. Bunlar, giriş katmanındaki gerçek gözlemler veya gizli katmanlardan birinin ara değeri olabilir.

• w1, w2,…, wN: Her girişin ağırlığı.

• bi: Eğilim ya da Sapma birimleri olarak adlandırılır. Bunlar, her ağırlığa karşılık gelen aktivasyon fonksiyonunun girişine eklenen sabit değerlerdir. Kesişme terimine benzer şekilde çalışır.

• a: Şu şekilde temsil edilebilen nöronun aktivasyonu olarak adlandırılır

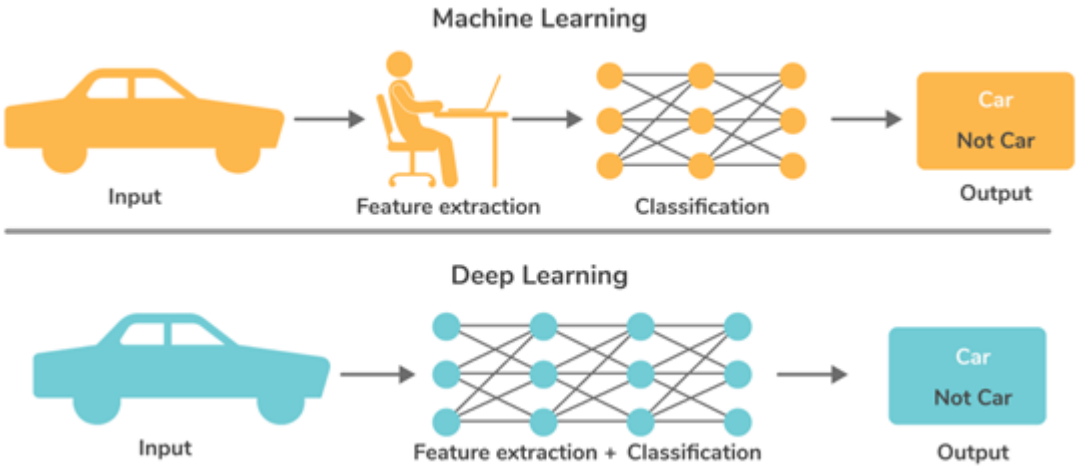
• ve y: nöronun çıktısıdır

$$a = f(\sum_{i=0}^{N} w_i x_i)$$

Derin Öğrenme Algoritmaları:
Gözetimli öğrenme
Kümeleme
Boyut indirgeme
Yapılandırılmış tahmin
Anomali tespiti
Sinir ağları
Pekiştirmeli öğrenme

Derin Öğrenme ile Makine Öğrenimi arasında farklar:
Derin Öğrenme: Derin Öğrenme, tekrarlayan bir sinir ağının ve bir yapay sinir ağının bağlantılı olduğu bir Makine Öğrenimi alt sınıfıdır. Algoritmalar, makine öğrenimi algoritmalarıyla aynı şekilde oluşturulur, ancak daha birçok algoritma seviyesi vardır. Yapay sinir ağı, algoritmanın tüm ağlarının bir araya getirilmesini ifade eder. Çok daha basit bir ifadeyle derin öğrenme kavramı olan beyindeki tüm sinir ağlarını birbirine bağlayarak insan beynini taklit eder. Her türlü karmaşık problemin üstesinden gelmek için algoritmalar ve bir teknik kullanır.

Makine Öğrenimi: Makine öğrenimi, bir sistemin o seviyeye programlanmak zorunda kalmadan deneyimlerinden öğrenmesini ve büyümesini sağlayan bir Yapay Zeka (AI) alt kümesidir. Veriler, öğrenmek ve doğru sonuçlar elde etmek için Machine Learning tarafından kullanılır. Makine öğrenimi algoritmaları, daha fazla veri elde ederek öğrenme ve performanslarını iyileştirme yeteneğine sahiptir. Makine öğrenimi şu anda diğer uygulamaların yanı sıra sürücüsüz arabalarda, siber dolandırıcılık tespitinde, yüz tanımada ve Facebook arkadaş önerisinde kullanılmaktadır.



The following table illustrates the difference between them:

Derin Öğrenme:

- Derin Öğrenme, Makine Öğreniminin bir alt sınıfıdır.
- Derin Öğrenme, verileri temsil etmek için çok farklı bir veri temsili (YSA) olan sinir ağlarını kullanır.
- Bunda çıktı, sayısal değerlerden metin veya ses gibi serbest biçimli öğelere kadar değişir.
- Verileri işleme katmanlarından geçirerek veri özelliklerini ve ilişkilerini değerlendirmek için bir sinir ağı kullanır.
- Genellikle milyonlarca veri noktasıyla ilgilenir.
- Makine Öğrenimi, Derin Öğrenmeye dönüşmüştür. Esasen, makine öğreniminin derinliğini ifade eder.

Makine öğrenme:

- Makine Öğrenimi, Derin Öğrenmenin bir süper sınıfıdır.
- Makine Öğrenimi, yapılandırılmış verileri kullandığından, verileri Derin Öğrenme'den farklı bir şekilde temsil eder.
- Bunda, çıktı sayısal değerlerden oluşur
- Girdileri model işlevlerine dönüştürmek ve gelecekteki eylemleri tahmin etmek için çeşitli otomatik teknikler kullanır.
- Genellikle binlerce veri noktasıyla ilgilenir.
- Yapay Zeka, Makine Öğrenimi'ne dönüştü.

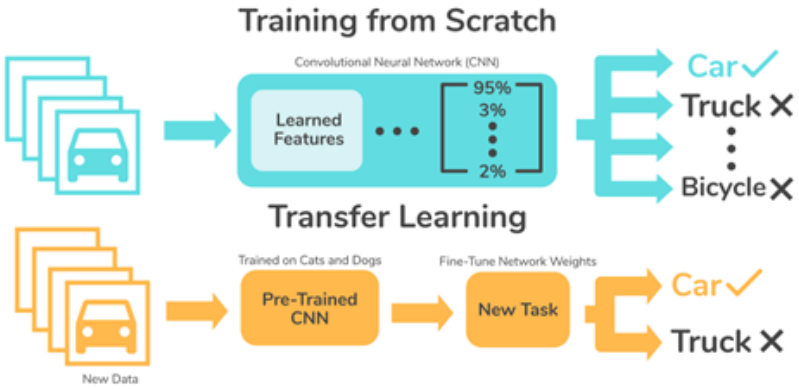
What are the applications of deep learning?

Derin öğrenme uygulamalarından bazıları şunlardır: -

• Örüntü tanıma ve doğal dil işleme.
• Görüntülerin tanınması ve işlenmesi.
• Otomatik çeviri.
• Duygu analizi.
• Soruları yanıtlamak için sistem.
• Nesnelerin Sınıflandırılması ve Tespiti.
• Makineye Göre El Yazısı Oluşturma.
• Otomatik metin oluşturma.
• Siyah Beyaz görüntülerin renklendirilmesi.

Derin öğrenme bağlamında transfer öğrenme

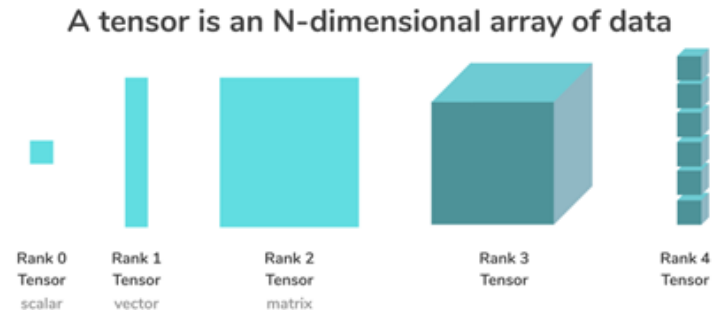
Aktarım öğrenimi, veri bilimcilerinin benzer bir görev için kullanılan önceki bir makine öğrenimi modelinden öğrendiklerini kullanmalarına olanak tanıyan bir öğrenme tekniğidir. Bu öğrenmede, insanların bilgilerini aktarma yeteneği örnek olarak kullanılmaktadır. Bisiklete binmeyi öğrenirseniz, diğer iki tekerlekli araçları kullanmayı daha kolay öğrenebilirsiniz. Otonom otomobil sürüşü için eğitilmiş bir model otonom kamyon sürüşü için de kullanılabilir. Özellikler ve ağırlıklar, yeni modeli eğitmek için kullanılabilir ve yeniden kullanılmasına izin verir. Sınırlı veri olduğunda, bir modeli hızlı bir şekilde eğitmek için aktarım öğrenimi etkili bir şekilde çalışır.



Yukarıdaki görüntüde, ilk diyagram bir modeli sıfırdan eğitmeyi temsil ederken, ikinci diyagram, farklı araç sınıflarını sınıflandırmak için kediler ve köpekler üzerinde önceden eğitilmiş bir modelin kullanılmasını temsil eder, böylece transfer öğrenmeyi temsil eder.

Derin öğrenmede tensör nedir?

Tensör, vektörlerin ve matrislerin genelleştirilmesini temsil eden çok boyutlu bir dizidir. Derin öğrenmede kullanılan temel veri yapılarından biridir. Tensörler, temel veri türlerinin n boyutlu dizileri olarak temsil edilir. Tensördeki her elemanın veri tipi aynıdır ve veri tipi her zaman bilinir. Şeklin yalnızca bir bölümünün (yani boyutların sayısı ve her boyutun boyutu) bilinmesi mümkündür. Çoğu işlem, girdileri aynı şekilde tam olarak biliniyorsa, tam olarak bilinen tensörler verir, ancak diğer durumlarda, bir tensörün şekli yalnızca grafik yürütme zamanında belirlenebilir.



Derin öğrenme bağlamında bir devirden kastınız nedir?

Epoch, derin öğrenmede kullanılan ve derin öğrenme algoritmasının tam eğitim veri kümesinde yaptığı geçiş sayısını ifade eden bir terminolojidir. Gruplar genellikle veri kümelerini gruplamak için kullanılır (özellikle veri miktarı çok büyük olduğunda). "Yineleme" terimi, modelde bir toplu iş yürütme sürecini ifade eder.

Toplu iş boyutu eğitim veri kümesinin tamamıysa, dönem sayısı yineleme sayısına eşittir. Pratik nedenlerden dolayı bu genellikle böyle değildir. Birçok modelin oluşturulmasında birkaç dönem kullanılır.

Genel bir ilişki:
d * e = ben * b
burada,
d veri kümesi boyutudur
e dönem sayısıdır
ben yineleme sayısı
b yığın, grup boyutudur

2. Anomali Tespiti

Veri analizinde, anomali tespiti (aynı zamanda aykırı değer tespiti), verilerin çoğunluğundan önemli ölçüde farklılaşarak şüphe uyandıran nadir öğelerin, olayların veya gözlemlerin tanımlanmasıdır. Tipik olarak anormal öğeler, banka dolandırıcılığı, yapısal bir kusur, tıbbi sorunlar veya bir metindeki hatalar gibi bir tür soruna dönüşecektir. Anormallikler ayrıca aykırı değerler, yenilikler, gürültü, sapmalar ve istisnalar olarak da adlandırılmaktadır.

Özellikle, kötüye kullanım ve ağa izinsiz giriş tespiti bağlamında, ilginç nesneler genellikle nadir nesneler değil, beklenmedik etkinlik patlamalarıdır. Bu model, bir aykırı değerin nadir bir nesne olarak genel istatistiksel tanımına uymaz ve uygun şekilde bir araya getirilmediği sürece birçok aykırı değer algılama yöntemi (özellikle denetimsiz yöntemler) bu tür verilerde başarısız olmaktadır. Bunun yerine, bir küme analizi algoritması, bu modellerin oluşturduğu mikro kümeleri tespit edebilmektedir.

Üç geniş anomali tespit tekniği kategorisi mevcuttur[4]. Denetimsiz anomali tespit teknikleri, veri setindeki örneklerin çoğunluğunun normal olduğu varsayımı altında, veri setinin geri kalanına en az uyan örnekleri arayarak etiketlenmemiş bir test veri setindeki anormallikleri tespit etmektedir. Denetimli anomali tespit teknikleri, "normal" ve "anormal" olarak etiketlenmiş bir veri seti gerektirir ve bir sınıflandırıcının eğitimini içermektedir (diğer birçok istatistiksel sınıflandırma probleminden temel fark, aykırı değer tespitinin doğal dengesiz doğasıdır). Yarı denetimli anomali tespit teknikleri, belirli bir normal eğitim veri setinden normal davranışı temsil eden bir model oluşturur ve ardından kullanılan model tarafından bir test örneğinin oluşturulma olasılığını test etmektedir.

3. Yapay Sinir Ağları

İnsan beyni:

İnsan beyninin nasıl çalıştığını daha yeni anlamaya başladık ; daha yolun başındayız. Akıllı makineler yapmanın bir yolu, insan beynini özellikle organizasyonel prensiplerini taklit etmeye çalışmaktır.

Beyin son derece karmaşık, doğrusal olmayan ve paralel bir bilgisayardır ve yoğun şekilde bağlı 10^{11} nörondan oluşur (nöron başına ~ 10^4 bağlantı). Bir nöron, silikon mantık geçidine (10^{-9}sn) kıyasla çok daha yavaştır (10^{-3}sn), doğrudur, nöronlar arasındaki muazzam bağlantı, nispeten yavaş hızı oluşturur. Ancak, *karmaşık algısal kararlara çok hızlı bir şekilde ulaşılır* (birkaç yüz milisaniye içinde)

100 Adım kuralı: Bireysel nöronlar birkaç milisaniye içinde çalıştığından, hesaplamalar yaklaşık 100'den fazla seri adım içermez ve bir nörondan diğerine gönderilen bilgi çok küçüktür (birkaç bit)

Plastisite: Beynin nöral yapısının bir kısmı doğumda bulunurken, diğer kısımları çevreye uyum sağlamak için özellikle yaşamın erken dönemlerinde öğrenme yoluyla geliştirilir (yeni girdiler). Biyolojik Nöronlarda, farklı dallanma yapılarına sahip çeşitli farklı nöronlar (motor nöron, merkez üstü çevresel olmayan görsel hücreler…) mevcuttur. Ağın bağlantıları ve tek tek sinapsların güçleri ağın işlevini oluşturur.

Beyindeki Biyolojik Nöronlar:

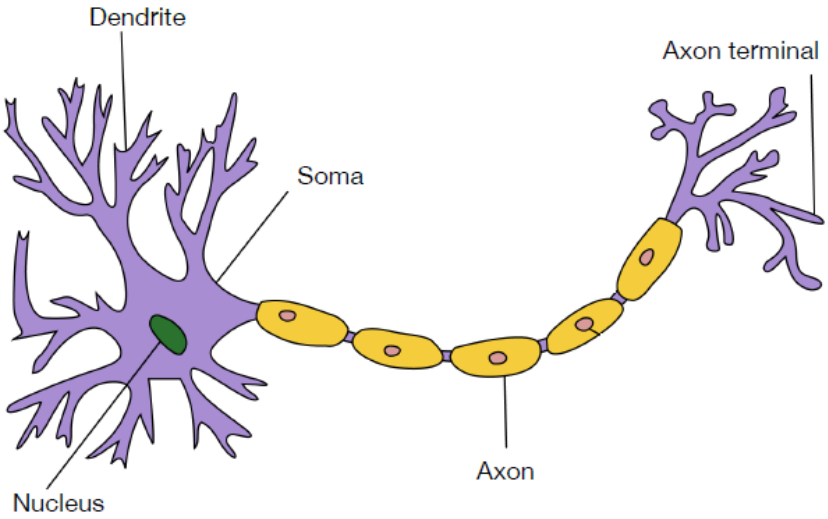
İnsan beyni yaklaşık 100 milyar nörondan oluşur.

dendritler: hücrelere elektrik sinyalleri taşıyan sinir lifleri

hücre gövdesi: girdilerinin doğrusal olmayan bir işlevini hesaplar

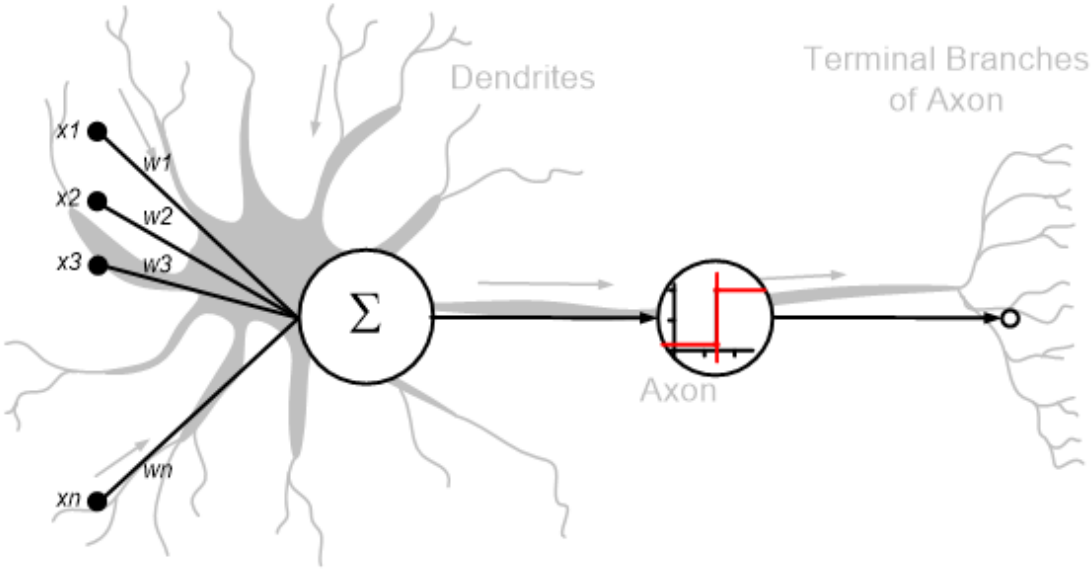
akson: elektrik sinyalini hücre gövdesinden diğer nöronlara taşıyan tek uzun lif. Nöronun aksonu, diğer birçok nöronun dendritlerine bağlıdır. Nöronlar, dendritlerden elektrik sinyalleri alır ve bunları aksona gönderir.

sinaps: gücü hücreye girişi etkileyen kimyasal bir bağlantıyı düzenleyen, bir hücrenin aksonu ile diğerinin dendriti arasındaki temas noktası.



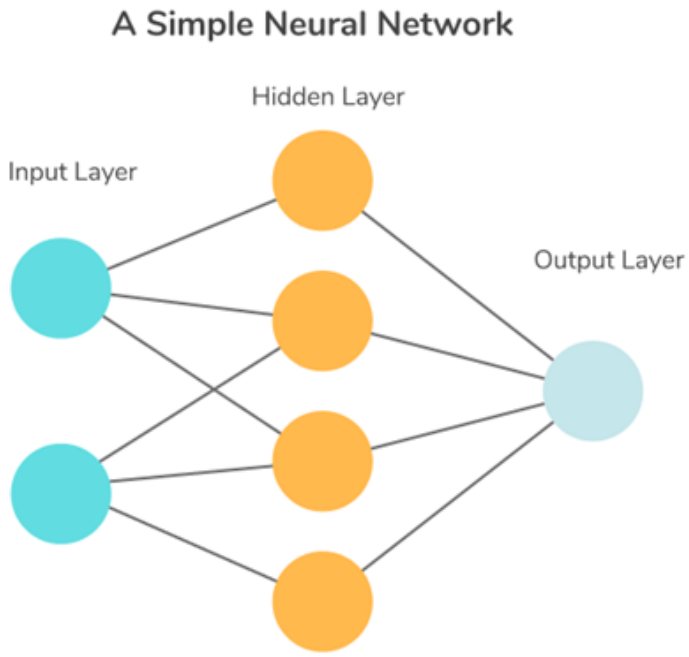
İnsan beyninden ilham alan hesaplama model, Yapay Sinir Ağları:

- Basit işlem birimlerinden (nöronlar) oluşan büyük ölçüde paralel, dağıtılmış sistem
- Edinilen bilgiyi depolamak için nöronlar arasındaki sinaptik bağlantı güçleri kullanılır.
- Bilgi, ağ tarafından bir öğrenme süreci aracılığıyla çevresinden edinilir.



- Basit işlem birimlerinden (nöronlar) oluşan büyük ölçüde paralel, dağıtılmış sistem
- Edinilen bilgiyi depolamak için nöronlar arasındaki sinaptik bağlantı güçleri kullanılır.
- Bilgi, ağ tarafından bir öğrenme süreci aracılığıyla çevresinden edinilir.
- Örneklerden öğrenmek: etiketli veya etiketsiz
- Adaptivite: bir şeyler öğrenmek için bağlantı güçlerini değiştirmek
- Doğrusal olmama: doğrusal olmayan aktivasyon fonksiyonları gereklidir

- Hata toleransı: Nöronlardan veya bağlantılardan biri hasar görürse, tüm ağ hala oldukça iyi çalışıyor.
- Bu nedenle, aşağıdakilerle karakterize edilen sorunlar için klasik çözümlerden daha iyi alternatifler olabilirler: Yüksek boyutluluk, gürültülü, kesin olmayan veya eksik veriler; ve açıkça ifade edilmiş matematiksel bir çözüm veya algoritmanın olmaması



3.1. Beyin ve Yapay Sinir Ağları

Benzerlikler:
- Nöronlar, nöronlar arasındaki bağlantılar
- Öğrenme = bağlantıların değişmesi, nöronların değişmesi değil
- Büyük paralel işleme

Ancak yapay sinir ağları çok daha basit:
- nöron içindeki hesaplama büyük ölçüde basitleştirildi
- ayrık zaman adımları
- tipik olarak çok sayıda uyaran içeren bir tür denetimli öğrenim

Bir sinir ağı, basit işlem birimlerinden (yapay nöronlar) oluşan büyük ölçüde paralel, dağıtılmış bir işlemcidir. Beyne iki açıdan benziyor:
- Bilgi, ağ tarafından bir öğrenme süreci aracılığıyla çevresinden edinilir
- Edinilen bilgiyi depolamak için nöronlar arasındaki sinaptik bağlantı güçleri kullanılır.

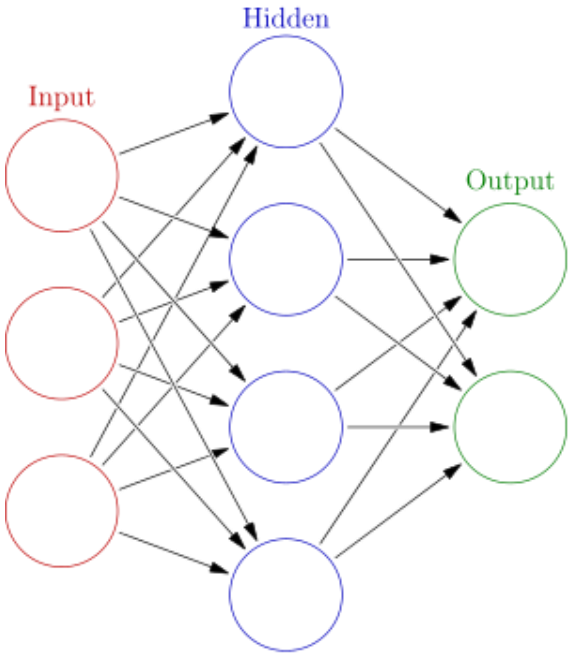
Yapay sinir ağları (YSA) veya bağlantı sistemi, hayvan beyinlerini oluşturan biyolojik sinir ağlarından belirsiz bir şekilde esinlenen bilgisayar sistemleridir. Bu tür sistemler, genellikle

herhangi bir göreve özgü kural ile programlanmadan, örnekleri dikkate alarak görevleri yerine getirmeyi "öğrenir".

YSA, biyolojik bir beyindeki nöronları gevşek bir şekilde modelleyen "yapay nöronlar" adı verilen bağlı birimler veya düğümler koleksiyonuna dayanan bir modeldir. Her bağlantı, biyolojik bir beyindeki sinapslar gibi, bir yapay nörondan diğerine bilgi, bir "sinyal" iletebilir. Bir sinyal alan yapay bir nöron, onu işleyebilir ve daha sonra ona bağlı ek yapay nöronları işaret edebilir. Ortak YSA uygulamalarında, yapay nöronlar arasındaki bağlantıdaki sinyal gerçek bir sayıdır ve her bir yapay nöronun çıkışı, girdilerinin toplamının doğrusal olmayan bir fonksiyonu tarafından hesaplanır. Yapay nöronlar arasındaki bağlantılara "kenarlar" denir. Yapay nöronlar ve kenarlar tipik olarak öğrenme ilerledikçe ayarlanan bir ağırlığa sahiptir. Ağırlık, bir bağlantıdaki sinyalin gücünü arttırır veya azaltır. Yapay nöronlar, sadece toplam sinyal bu eşiği geçtiğinde sinyalin gönderileceği bir eşik değerine sahip olabilir. Tipik olarak, yapay nöronlar katmanlar halinde toplanır. Farklı katmanlar, girdilerinde farklı türde dönüşümler gerçekleştirebilir. Sinyaller, muhtemelen katmanları birden çok kez geçtikten sonra, ilk katmandan (giriş katmanı) son katmana (çıkış katmanı) gider.

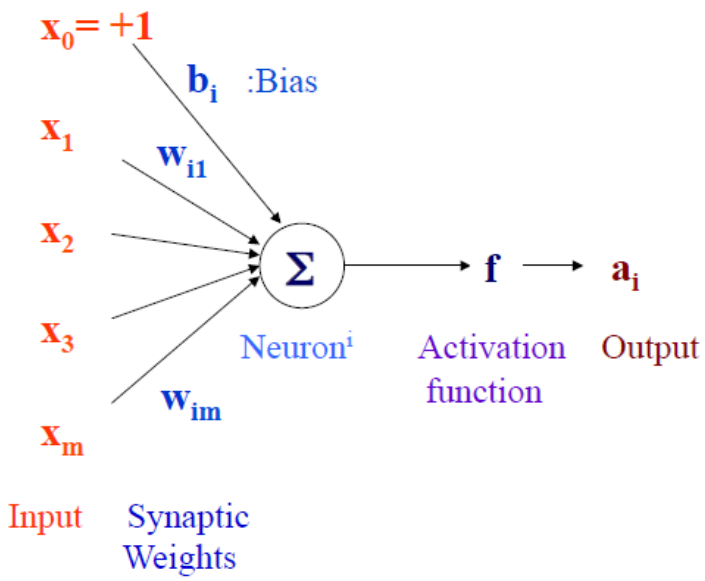
YSA yaklaşımının asıl amacı, problemleri bir insan beyninde olduğu gibi çözmekti. Bununla birlikte, zamanla dikkat, belirli görevlerin yerine getirilmesine ve biyolojiden sapmalara yol açtı. Yapay sinir ağları, bilgisayar görme, konuşma tanıma, makine çevirisi, sosyal ağ filtreleme, oyun tahtası ve video oyunları ve tıbbi teşhis gibi çeşitli görevlerde kullanılmıştır.

Derin öğrenme yapay bir sinir ağında birden fazla gizli katmandan oluşur. Bu yaklaşım, insan beyninin ışığı ve sesi görme ve işitme biçimini modellemeye çalışır. Derin öğrenmenin bazı başarılı uygulamaları bilgisayarla görme ve konuşma tanımadır.



Yapay bir sinir ağı, bir beyindeki geniş nöron ağına benzer şekilde birbirine bağlı bir düğüm grubudur. Burada, her dairesel düğüm bir yapay nöronu temsil eder ve bir ok, bir yapay nöronun çıkışından diğerinin girişine bir bağlantıyı temsil eder.

Yapay Nöron Modeli



Eğilim – Meyillenme (Bias):

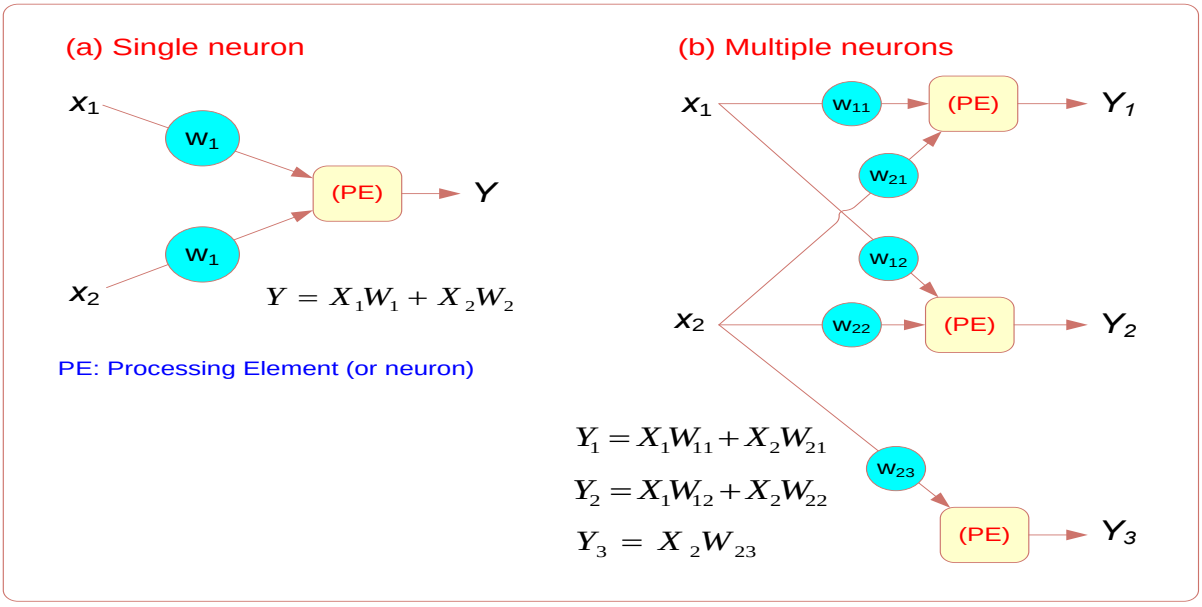
Bias: eşik değer, eğilim ya da sapma değeri.

Yapay bir nöron: Girdisinin ağırlıklı toplamını hesaplar (net girdisi denir). Sapmasını ekler. Bu değeri bir aktivasyon işlevinden geçirir. Nöronun çıkışı sıfırın üzerindeyse tetiklendiğini, "ateşlendiğini" (yani aktif hale geldiğini) söylüyoruz. Sapma, sabit +1.0 girdisine kenetlenen başka bir ağırlık olarak dahil edilebilir. Bu ekstra serbest değişken (eğilim ya da sapma), nöronu daha güçlü hale getirir.

$$a_i = f(n_i) = f\left(\sum_{j=1}^{n} w_{ij}x_j + b_i\right)$$

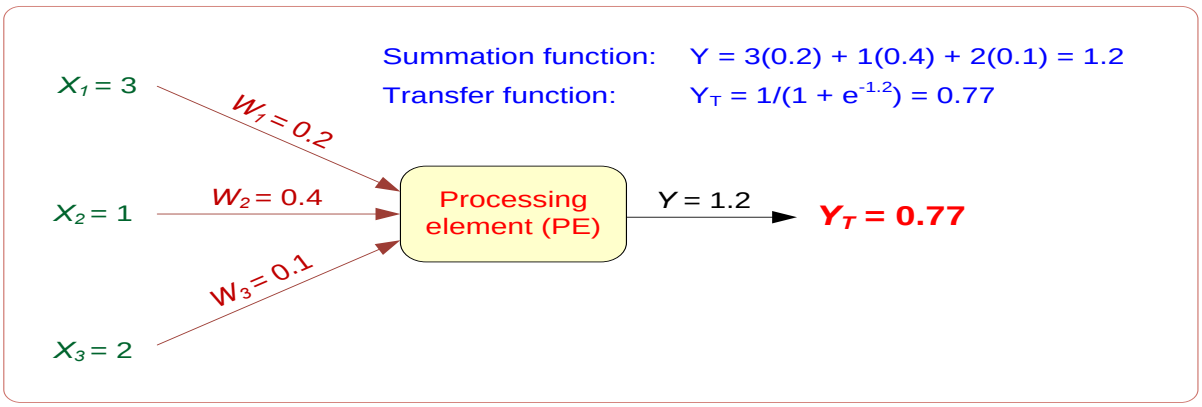
3.2. Yapay Sinir Ağları (YSA) Unsurları

- İşleme öğesi (Processing element - PE)
- Ağ Mimarisi
  - Gömülü Katmanlar
  - Paralel İşlem
- Ağ Bilgisi İşleme
  - Girişler
  - Çıkışlar
  - Bağlantı Ağırlıkları (Connection weights)
  - Toplama İşlevi (Summation function)



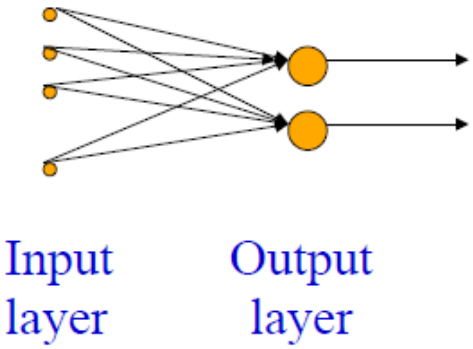
Transformation (Transfer) Function:

- Linear function
- Sigmoid (logical activation) function [0 1]
- Tangent Hyperbolic function [-1 1]

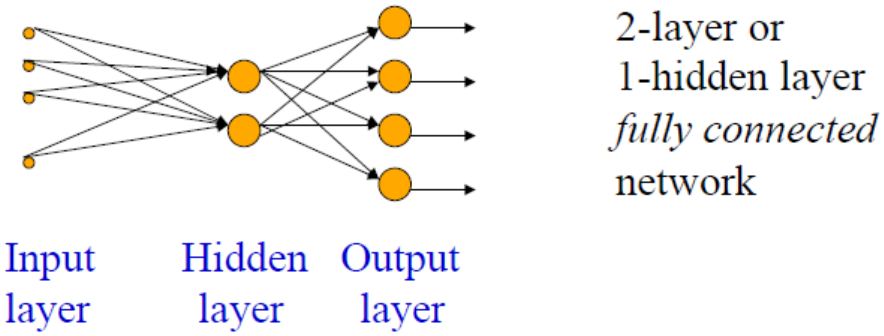


Farklı Ağ Topolojileri:

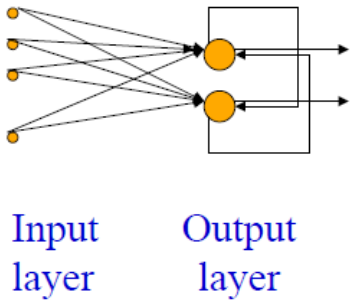
- **Tek katmanlı ileri beslemeli ağlar:** Çıktı katmanına yansıyan girdi katmanı.



- **Çok katmanlı ileri beslemeli ağlar:** Bir veya daha fazla gömülü katman. Girdi projeleri yalnızca önceki katmanlardan bir katmana yansıtılır. Tipik olarak, yalnızca bir katmandan diğerine bağlanırlar.

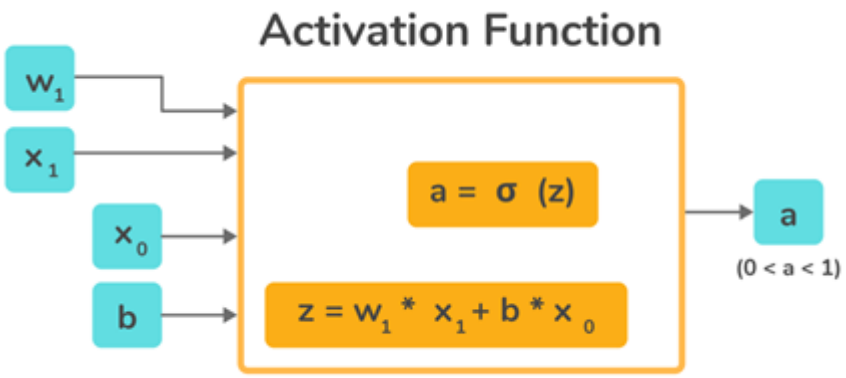


- **Tekrarlayan ağlar:** Bazı girişlerinin bazı çıkışlarına (ayrık zaman) bağlı olduğu geri beslemeli bir ağ.

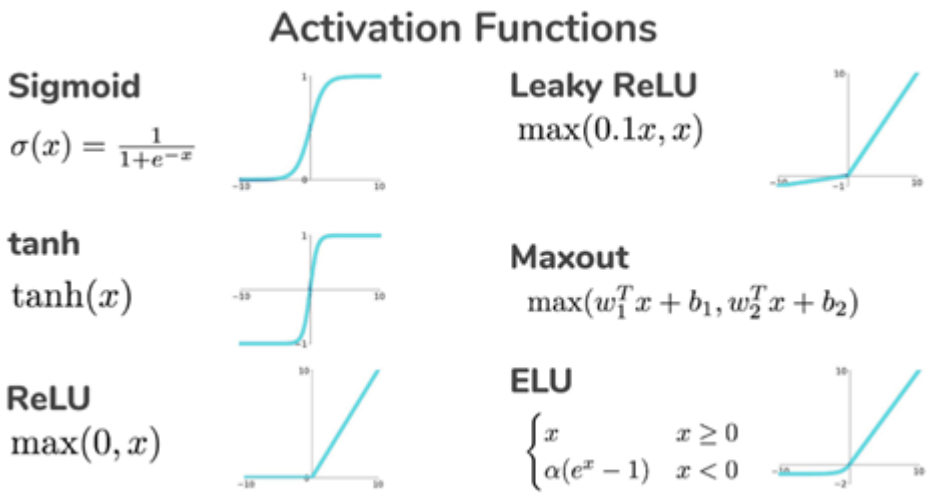


3.3. Aktivasyon fonksiyonları

Bir yapay sinir ağının etkinleştirme işlevi, ağın verilerdeki karmaşık kalıpları öğrenmesine yardımcı olmak için tanıtılan bir işlevdir. Beynimizde görülen nöron tabanlı bir modelle karşılaştırıldığında, sürecin sonunda bir sonraki nörona neyin ateşleneceğini belirlemekten aktivasyon işlevi sorumludur. Bir YSA'da bir aktivasyon işlevi aynı işi gerçekleştirir. Bir önceki hücrenin çıkış sinyalini alır ve onu bir sonraki hücreye giriş olarak kullanılabilecek bir formata dönüştürür.



Aşağıda, farklı etkinleştirme işlevi (Aktivizasyon fonksiyon) türleri verilmiştir:



Sigmoid işlevi: Sigmoid işlevi, çoğunlukla ileri beslemeli sinir ağlarında kullanılan bir YSA'da doğrusal olmayan bir etkinleştirme işlevidir. Her yerde pozitif türevleri olan ve gerçek girdi değerleri için tanımlanmış belirli bir düzgünlük derecesine sahip türevlenebilir bir gerçek fonksiyondur. Sigmoid işlevi, derin öğrenme modellerinin çıktı katmanında bulunur ve olasılığa dayalı çıktıları tahmin etmek için kullanılır.

Hiperbolik Tanjant İşlevi (Tanh): Tanh işlevi, -1 ila 1 aralığına sahip daha yumuşak ve sıfır merkezli bir işlevdir. Çok katmanlı sinir ağları için daha yüksek eğitim performansı sağladığından, tanh işlevi sigmoid işlevinden çok daha yaygın olarak kullanılır. Tanh fonksiyonunun birincil avantajı, geri yayılıma yardımcı olan sıfır merkezli bir çıktı vermesidir.

Softmax işlevi: softmax işlevi, bir gerçek sayı vektöründen olasılık dağılımı oluşturmak için sinir ağlarında kullanılan başka bir etkinleştirme işlevi türüdür. Bu işlev, olasılıkların toplamı 1'e eşit olacak şekilde 0 ile 1 arasında bir sayı döndürür. Bu işlev en yaygın olarak çok sınıflı modellerde kullanılır ve hedef sınıf en yüksek olasılığa sahip olacak şekilde her sınıf için olasılıkları döndürür. DL mimarisinin hemen hemen tüm çıktı katmanlarında bulunabilir.

Softsign işlevi: Bu, en yaygın olarak regresyon hesaplama sorunları ve derin öğrenmeye dayalı metin okuma uygulamalarında kullanılır.

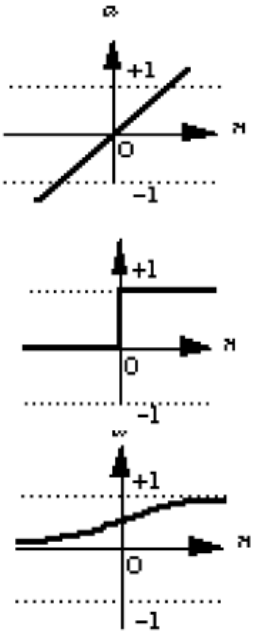
Düzeltilmiş Doğrusal Birim İşlevi: Doğrultulmuş doğrusal birim (ReLU) işlevi, üstün performans ve olağanüstü sonuçlar vermeyi vaat eden hızlı öğrenen bir yapay zekadır (AI). Derin öğrenmede, ReLU işlevi, performans ve genelleme açısından sigmoid ve tanh işlevleri gibi diğer AF'lerden daha iyi performans gösterir. İşlev, doğrusal modellerin özelliklerini koruyan ve gradyan iniş yaklaşımlarını optimize etmeyi kolaylaştıran kabaca doğrusal bir işlevdir. Her giriş öğesinde, ReLU işlevi, sıfırdan küçük tüm değerleri sıfıra ayarlayarak bir eşik işlemi gerçekleştirir.

Üstel Doğrusal Birim İşlevi: Üstel doğrusal birimler (ELU'lar) işlevi, sinir ağı eğitimini hızlandırmak için kullanılabilen bir AF türüdür (tıpkı ReLU işlevi gibi). ELU fonksiyonunun en büyük avantajı, pozitif değerler için özdeşlik kullanarak ve modelin öğrenme özelliklerini artırarak kaybolan gradyan problemini çözebilmesidir.

Nöronun çıktısının genliğini sınırladığı için limit işlevi olarak da adlandırılır.

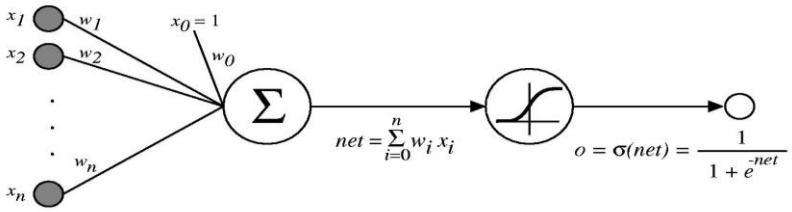
Many types of activations functions are used:

- linear: $a = f(n) = n$
- threshold: $a = \{1 \text{ if } n >= 0$ (hardlimiting)
  $0 \text{ if } n < 0$
- sigmoid: $a = 1/(1+e^{-n})$
- ...



Name	Input/Output Relation	Icon	MATLAB Function
Hard Limit	$a = 0 \quad n < 0$ $a = 1 \quad n \geq 0$		hardlim
Symmetrical Hard Limit	$a = -1 \quad n < 0$ $a = +1 \quad n \geq 0$		hardlims
Linear	$a = n$		purelin
Saturating Linear	$a = 0 \quad n < 0$ $a = n \quad 0 \leq n \leq 1$ $a = 1 \quad n > 1$		satlin
Symmetric Saturating Linear	$a = -1 \quad n < -1$ $a = n \quad -1 \leq n \leq 1$ $a = 1 \quad n > 1$		satlins
Log-Sigmoid	$a = \frac{1}{1+e^{-n}}$		logsig
Hyperbolic Tangent Sigmoid	$a = \frac{e^{n}-e^{-n}}{e^{n}+e^{-n}}$		tansig
Positive Linear	$a = 0 \quad n < 0$ $a = n \quad 0 \leq n$		poslin
Competitive	$a = 1$ neuron with max n $a = 0$ all other neurons		compet

Sigmoid unit:

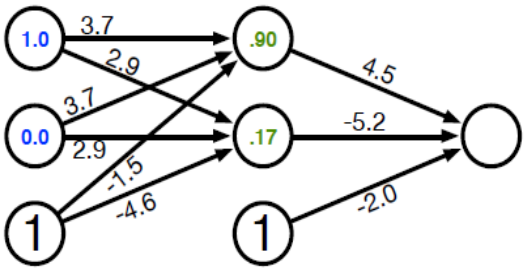


$\sigma(x)$ is the sigmoid function

$$\frac{1}{1+e^{-x}}$$

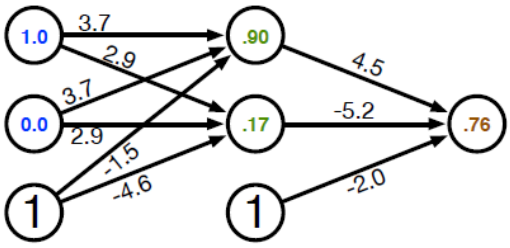
Nice property: $\frac{d\sigma(x)}{dx} = \sigma(x)(1-\sigma(x))$

Örnek: Basit Sinir Ağı



$$\text{sigmoid}(1.0 \times 3.7 + 0.0 \times 3.7 + 1 \times -1.5) = \text{sigmoid}(2.2) = \frac{1}{1+e^{-2.2}} = 0.90$$

$$\text{sigmoid}(1.0 \times 2.9 + 0.0 \times 2.9 + 1 \times -4.5) = \text{sigmoid}(-1.6) = \frac{1}{1+e^{1.6}} = 0.17$$

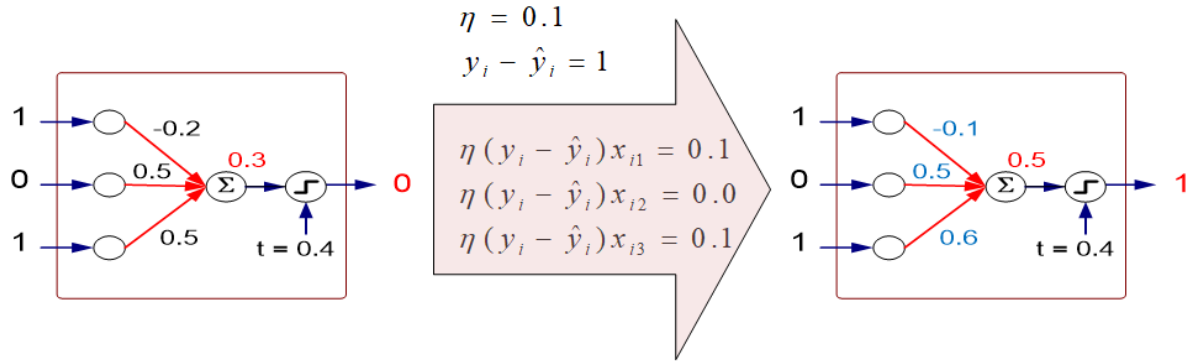


- Output

$$\text{sigmoid}(.90 \times 4.5 + .17 \times -5.2 + 1 \times -2.0) = \text{sigmoid}(1.17) = \frac{1}{1+e^{-1.17}} = 0.76$$

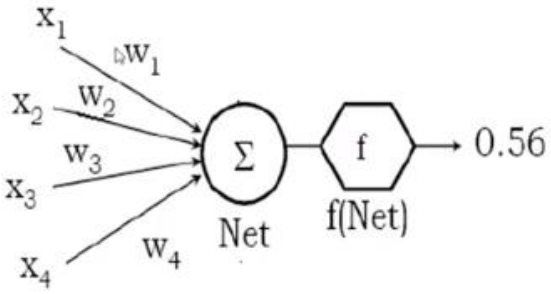
Bir numuneyi işleme örneği:

X_1	X_2	X_3	Y
1	0	1	1



Örnek:

Girişler	Ağırlıklar
$x_1 = 0.5$	$w_1 = -0.2$
$x_2 = 0.6$	$w_2 = 0.6$
$x_3 = 0.2$	$w_3 = 0.2$
$x_4 = 0.7$	$w_4 = -0.1$



Hücrenin net girdisi;

$Net = \Sigma x_i w_i \qquad i=1...4$

$Net = 0.5*(-0.2)+0.6*0.6+0.2*0.2+0.7*(-0.1)$

$Net = 0.23$

- **Sigmoid aktivasyon fonksiyonuna göre hücrenin çıkışı;**

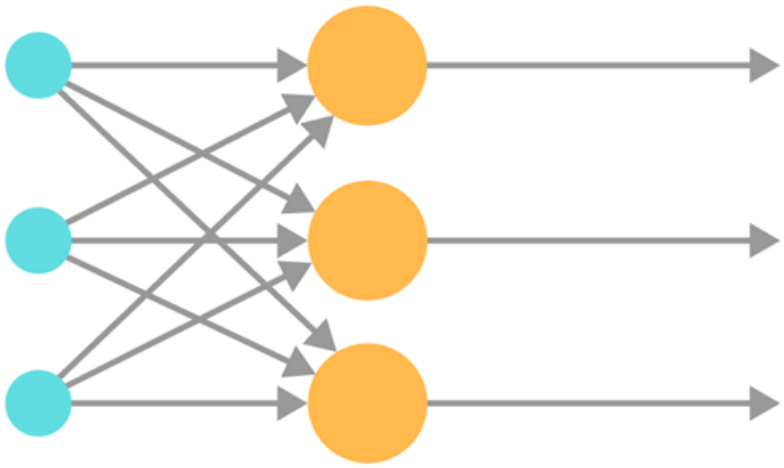
$f(Net) = 1 / (1+e^{-0.23})$

$f(Net) = 0.56$

3.4. Derin sinir ağlarının farklı türleri

Feed Forward Sinir Ağı:

Akış kontrolünün giriş katmanında başladığı ve çıkış katmanına geçtiği en temel sinir ağı türüdür. Bu ağlar yalnızca tek bir katmana veya tek bir gizli katmana sahiptir. Bu ağda geri yayılım mekanizması yoktur çünkü veriler yalnızca tek bir şekilde akar. Bu ağın giriş katmanı, girişte bulunan ağırlıkların toplamını alır. Bu ağlar, bilgisayarla görme tabanlı yüz tanıma yönteminde kullanılmaktadır.



Radial Basis Function Neural Network:

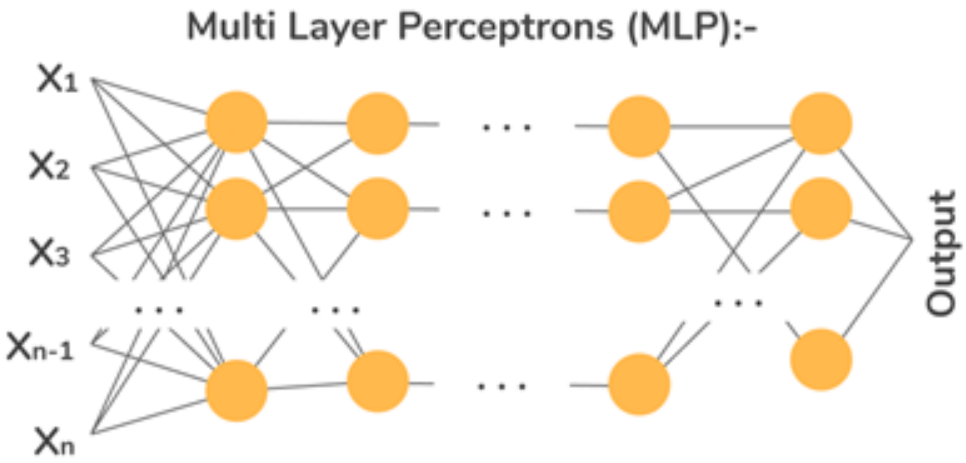
Radyal Temel Fonksiyonlu Sinir Ağı:

Bu tür sinir ağı genellikle birden fazla katmana, tercihen iki katmana sahiptir. Bu ağ türünde herhangi bir konumdan merkeze olan bağıl uzaklık belirlenir ve bir sonraki katmana aktarılır. Kesintileri önlemek için, gücü mümkün olan en kısa sürede geri yüklemek için güç restorasyon sistemlerinde yaygın olarak radyal tabanlı ağlar kullanılır.

Multi-Layer Perceptrons (MLP):

Çok Katmanlı Algılayıcılar (MLP):

Çok katmanlı algılayıcı (MLP), bir tür ileri beslemeli yapay sinir ağıdır (YSA). MLP'ler, birbirini takip eden tamamen bağlantılı katmanlardan oluşan en basit derin sinir ağlarıdır. Her ardışık katman, önceki katmanın tüm çıktılarının ağırlıklı toplamı olan (tamamen bağlantılı) doğrusal olmayan işlevler koleksiyonundan oluşur. Konuşma tanıma ve diğer makine öğrenimi sistemleri büyük ölçüde bu ağlara dayanır.



3.5. Sinir ağlarının avantajları ve dezavantajları

Sinir ağlarının avantajları şunlardır:

• Sinir ağları son derece uyarlanabilirdir ve hem sınıflandırma hem de regresyon problemlerinin yanı sıra çok daha karmaşık problemler için kullanılabilirler. Sinir ağları da oldukça ölçeklenebilir. Her biri kendi nöron setine sahip, istediğimiz kadar katman oluşturabiliriz. Çok fazla veri noktası olduğunda, sinir ağlarının en iyi sonuçları ürettiği gösterilmiştir. Görüntüler, metinler vb. gibi doğrusal olmayan verilerle en iyi şekilde kullanılırlar. Sayısal bir değere dönüştürülebilen herhangi bir veriye uygulanabilirler.

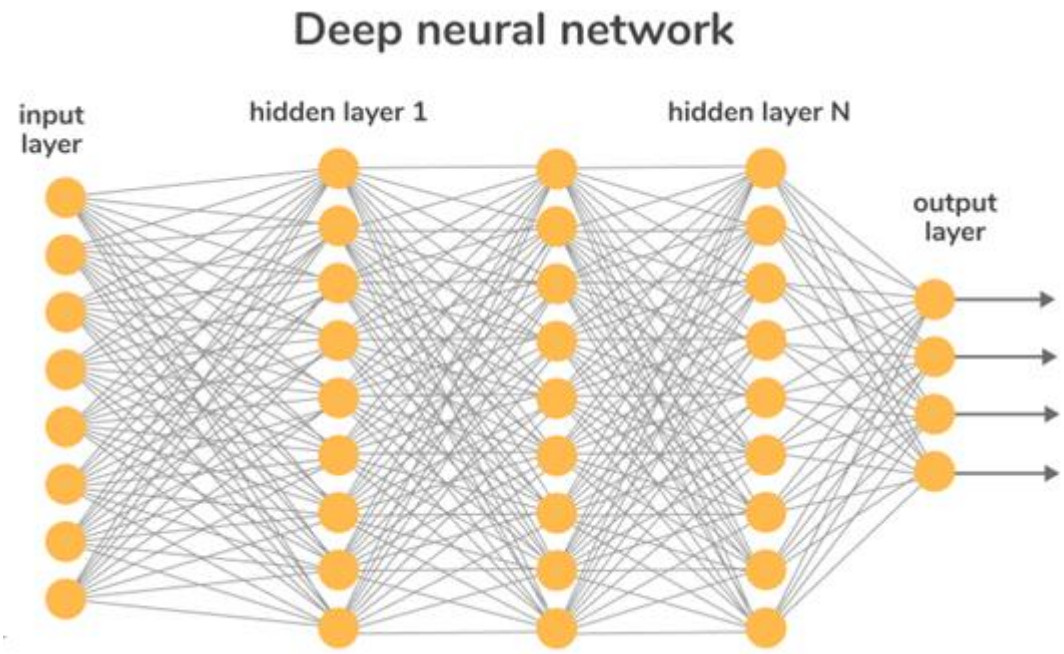
• Sinir ağı modu bir kez eğitildikten sonra, çıktıları çok hızlı bir şekilde verirler. Böylece zamandan tasarruf sağlarlar.

Sinir ağlarının dezavantajları şunlardır: -

- Sinir ağlarının "kara kutu" yönü, iyi bilinen bir dezavantajdır. Yani, sinir ağının nasıl veya neden belirli bir sonuç ürettiği hakkında hiçbir fikrimiz yok. Bir sinir ağına bir köpek görüntüsü girdiğimizde ve bunun bir ördek olduğunu tahmin ettiğinde, onu bu tahmini yapmaya neyin sevk ettiğini anlamakta zorlanabiliriz.
- Bir sinir ağı modeli oluşturmak uzun zaman alır.
- Sinir ağları modelleri, her katmanda çok sayıda hesaplama yapılması gerektiğinden, hesaplama açısından pahalıdır.
- Bir sinir ağı modeli, eğitmek için geleneksel bir makine öğrenimi modelinden önemli ölçüde daha fazla veri gerektirir.

3.6. Derin sinir ağı (Deep neural network - DNN)

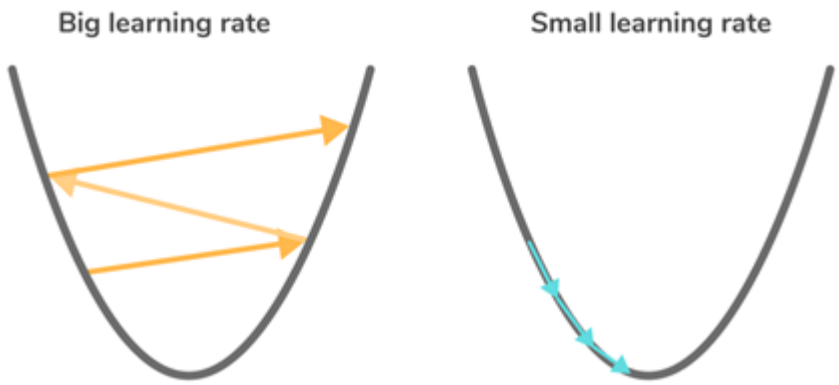
Giriş ve çıkış katmanları arasında çok sayıda katmana sahip bir yapay sinir ağı (YSA), derin sinir ağı (DNN) olarak bilinir. Derin sinir ağları, derin mimarileri kullanan sinir ağlarıdır. "Derin" terimi, tek bir katmanda daha fazla sayıda katmana ve birime sahip olan işlevleri ifade eder. Daha yüksek düzeyde desenler yakalamak için daha fazla ve daha büyük katmanlar ekleyerek daha doğru modeller oluşturmak mümkündür. Aşağıdaki görüntü derin bir sinir ağını göstermektedir.



3.7. Sorular

Explain learning rate in the context of neural network models. What happens if the learning rate is too high or too low?
Öğrenme hızını sinir ağı modelleri bağlamında açıklayın. Öğrenme oranı çok yüksek veya çok düşükse ne olur?

Öğrenme hızı 0 ile 1 arasında değişen bir sayıdır. Sinir ağı eğitim modellerinde en önemli ayarlanabilir hiperparametrelerden biridir. Öğrenme oranı, bir sinir ağı modelinin belirli bir duruma ne kadar hızlı veya yavaş adapte olduğunu ve öğrendiğini belirler. Daha yüksek bir öğrenme oranı değeri, modelin yalnızca birkaç eğitim dönemine ihtiyaç duyduğunu ve hızlı değişiklikler ürettiğini gösterirken, daha düşük bir öğrenme oranı, modelin yakınsamasının uzun zaman alabileceğini veya hiçbir zaman yakınsayıp kötü bir çözümde takılıp kalmayacağını gösterir. Sonuç olarak, çok düşük veya çok yüksek bir öğrenme oranı kullanmak yerine, deneme yanılma yoluyla iyi bir öğrenme oranı değeri oluşturulması önerilir.



Yukarıdaki görüntüde, büyük bir öğrenme oranının bizi istenen çıktıdan uzaklaşmaya yönlendirdiğini açıkça görebiliriz. Ancak, küçük bir öğrenme oranına sahip olmak, sonunda bizi istenen çıktıya götürür.

What do you mean by end-to-end learning?

Uçtan uca öğrenmeden kastınız nedir?

Bir modelin ham verilerle beslendiği ve tüm verilerin aynı anda herhangi bir ara adım olmadan istenen sonucu oluşturmak için eğitildiği bir derin öğrenme prosedürüdür. Tüm farklı adımların sıralı olarak değil aynı anda eğitildiği bir derin öğrenme yöntemidir. Uçtan uca öğrenme, genellikle daha düşük sapma ile sonuçlanan örtük özellik mühendisliği gereksinimini ortadan kaldırma avantajına sahiptir. Sürücüsüz otomobiller, uçtan uca öğrenme içeriğinizde kullanabileceğiniz mükemmel bir örnektir. İnsan girdisi tarafından yönlendirilirler ve görevleri yerine getirmek için bir CNN (**Convolutional Neural Network)** kullanarak bilgileri otomatik olarak öğrenmek ve yorumlamak üzere programlanırlar. Bir başka iyi örnek, kaydedilmiş bir ses klibinden (giriş) yazılı bir dökümün (çıktı) üretilmesidir. Buradaki model, tüm adımları ve görevleri yönetebileceği gerçeğine odaklanarak ortadaki tüm adımları atlar.

What do you understand about gradient clipping in the context of deep learning?

Derin öğrenme bağlamında gradyan kırpma hakkında ne anlıyorsunuz?

Gradyan Kırpma, geri yayılım sırasında meydana gelen gradyanların patlaması (zaman içinde büyük hata gradyanlarının biriktiği ve eğitim sırasında sinir ağı model ağırlıklarında büyük değişikliklere neden olduğu bir durum) sorunuyla başa çıkmak için bir tekniktir. Eğimlerin patlaması sorunu, eğitim sırasında eğimler aşırı büyüdüğünde ve modelin kararsız hale gelmesine neden olduğunda ortaya çıkar. Gradyan beklenen aralığı geçtiyse, gradyan değerleri, belirli bir minimum veya maksimum değere tek tek yönlendirilir. Gradyan kırpma, bir sinir ağını eğitirken sayısal kararlılığı artırır, ancak modelin performansı üzerinde çok az etkisi vardır.

Explain Forward and Back Propagation in the context of deep learning.

İleri ve Geri Yayılımı derin öğrenme bağlamında açıklayın.

İleri Yayılım: Ağın girdi katmanı ile çıktı katmanı arasındaki gizli katman, girdileri ağırlıklarla alır. Her gizli katmandaki her düğümdeki aktivasyonun çıktısını hesaplıyoruz ve bu, son çıktı katmanına ulaşana kadar bir sonraki katmana yayılıyor. Girdilerden ileriye doğru yayılma olarak bilinen son çıktı katmanına geçiyoruz.

Geri Yayılım: Ağın son katmanından hata bilgilerini ağ içindeki tüm ağırlıklara gönderir. Önceki çağın (yani yineleme) hata oranına dayalı olarak bir sinir ağının ağırlıklarının ince ayarını yapmak için kullanılan bir tekniktir. Ağırlıklara ince ayar yaparak hata oranlarını düşürebilir ve modelin genellemesini iyileştirebilir, böylece daha güvenilir hale getirebilirsiniz. Geri yayılım süreci aşağıdaki adımlara ayrılabilir: Eğitim verilerini ağ üzerinden yayarak çıktı üretebilir. Ardından, hedef ve çıkış değerlerini kullanarak çıkış aktivasyonları için hata türevini hesaplar. Önceki katmanın çıktı aktivasyonundaki hatanın türevini hesaplamak için geri yayılabilir ve tüm gizli katmanlar için böyle devam edebilir.

Daha önce elde edilen türevleri ve tüm gizli katmanları kullanarak ağırlıklar için hata türevini hesaplar. Ağırlıklar, bir sonraki katmandan elde edilen hata türevlerine göre güncellenir.

What do you mean by hyperparameters in the context of deep learning?

Derin öğrenme bağlamında hiperparametreler ile ne demek istiyorsunuz?

Hiperparametreler, ağ topolojisini (örneğin, gizli birimlerin sayısı) ve ağın nasıl eğitildiğini (Örn: Öğrenme Hızı) belirleyen değişkenlerdir. Modeli eğitmeden önce, yani ağırlıkları ve sapmayı optimize etmeden önce ayarlanırlar.

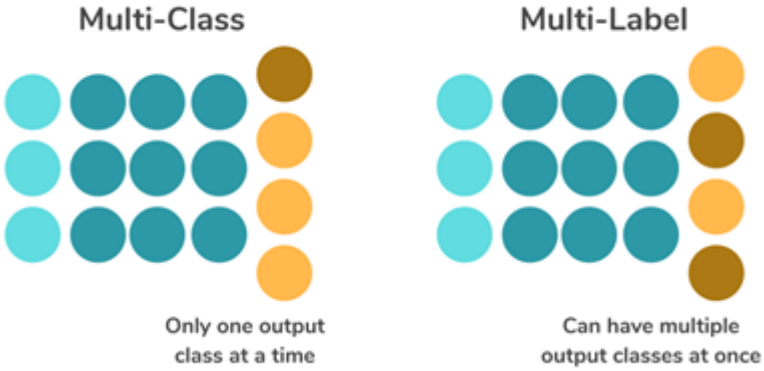
Aşağıda bazı hiperparametre örnekleri verilmiştir:

Gizli katman sayısı: Düzenlileştirme teknikleri ile bir katman içindeki birçok gizli birim doğruluğu artırabilir. Birim sayısı azaltılırsa, eksik yerleştirme meydana gelebilir.

Öğrenme Hızı: Öğrenme hızı, bir ağın parametrelerinin güncellenme hızıdır. Öğrenme süreci, düşük bir öğrenme oranı ile yavaşlar, ancak sonunda birleşir. Daha hızlı bir öğrenme oranı, öğrenme sürecini hızlandırır, ancak yakınsamayabilir. Düşen bir Öğrenme oranı genellikle arzu edilir.

Hiperparametreler mantıksal devre olarak tasarlanabilir. Herbir giriş durumları için çıkış elde edilir. Tüm girişler için çıkış elde edildikten sonra mantıksal devre tasarımı yapılır.

Difference between multi-class and multi-label classification problems.



Çok sınıflı bir sınıflandırma probleminde sınıflandırma görevi, birbirini dışlayan ikiden fazla sınıfa (kesişimi olmayan veya ortak özelliği olmayan sınıflar) sahipken, çok etiketli bir sınıflandırma probleminde, görevler farklı olmasına rağmen, her etiketin farklı bir sınıflandırma görevi vardır. bir şekilde ilişkilidir. Örneğin, kedi, köpek veya ayı olabilecek bir grup hayvan fotoğrafını sınıflandırmak, her örneğin yalnızca bir tür olabileceğini varsayan çok sınıflı bir sınıflandırma problemidir, bu da bir görüntünün kedi veya kedi olarak kategorize edilebileceğini gösterir. köpek, ama ikisi aynı anda değil.
Şimdi aşağıdaki resmi değiştirmek istediğinizi varsayalım.

Yukarıdaki resim, her iki canlıyı da betimlediği için hem kedi hem de köpek olarak kategorize edilmelidir. Çok etiketli bir sınıflandırma sorununda her örneğe bir etiket kümesi tahsis edilir ve sınıflar birbirini dışlamaz. Çok etiketli bir sınıflandırma probleminde, bir örüntü bir veya daha fazla sınıfa ait olabilir.

What are the advantages of transfer learning?
Aktarım yoluyla öğrenmenin avantajları nelerdir?

Transfer öğrenmenin avantajları şunlardır:
Daha iyi başlangıç modeli: Diğer öğrenme yöntemlerinde, sıfırdan bir model oluşturmalısınız. Transfer öğrenme daha iyi bir başlangıç noktasıdır çünkü başlangıç modelinin ayrıntılarını bilmeden görevleri daha üst düzeyde gerçekleştirmemizi sağlar.

Daha yüksek öğrenme oranı: Problem zaten benzer bir görev için öğretildiği için, transfer öğrenme eğitim sırasında daha hızlı bir öğrenme oranına izin verir.

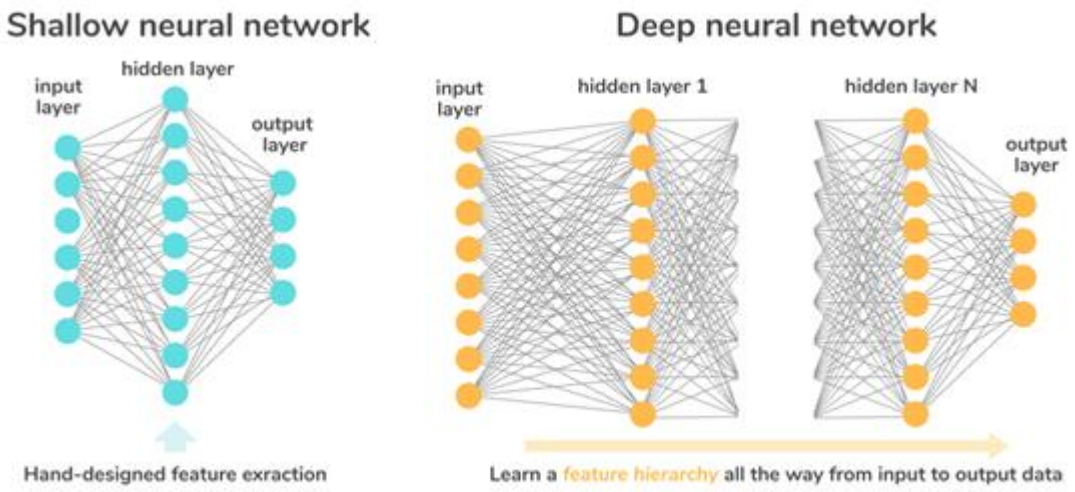
Eğitimden sonra daha yüksek doğruluk: Transfer öğrenimi, daha iyi bir başlangıç noktası ve daha yüksek öğrenme oranı sayesinde daha doğru çıktıyla sonuçlanan bir derin öğrenme modelinin daha yüksek bir performans düzeyinde yakınsamasına olanak tanır.

Is it possible to train a neural network model by setting all biases to 0? Also, is it possible to train a neural network model by setting all of the weights to 0?
Tüm yanlılıkları 0'a ayarlayarak bir sinir ağı modeli eğitmek mümkün müdür? Ayrıca, tüm ağırlıkları 0'a ayarlayarak bir sinir ağı modeli eğitmek mümkün müdür?

Evet, tüm yanlılıklar sıfıra ayarlansa bile, sinir ağı modelinin öğrenme şansı vardır.
Hayır, sinir ağı bir görevi tamamlamayı asla öğrenemeyeceğinden, tüm ağırlıkları 0'a ayarlayarak bir modeli eğitmek imkansızdır. Tüm ağırlıklar sıfıra ayarlandığında, her w için türevler sabit kalır, bu da nöronların her yinelemede aynı özellikleri öğrenmesine neden olur. Ağırlıkların herhangi bir sabit başlatılması, sadece sıfır değil, muhtemelen kötü bir sonuç üretecektir.

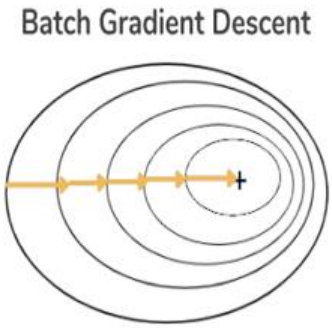
Explain the difference between a shallow network and a deep network.
Sığ ağ ile derin ağ arasındaki farkı açıklayın.

Her sinir ağında bir gizli katman ile giriş ve çıkış katmanları bulunur. Sığ sinir ağları, yalnızca bir gizli katmana sahip olanlardır, oysa derin sinir ağları çok sayıda gizli katman içerir. Hem sığ hem de derin ağlar herhangi bir işleve sığabilir, ancak sığ ağlar çok sayıda girdi parametresi gerektirirken, derin ağlar birkaç katmanları nedeniyle az sayıda girdi parametresi ile işlevlere uyabilir. Model, her katmandaki girdinin yeni ve soyut bir temsilini öğrendiğinden, şu anda sığ ağlar yerine derin ağlar tercih edilmektedir. Sığ ağlara kıyasla, parametre sayısı ve hesaplamalar açısından da çok daha verimlidirler.



Explain Batch Gradient Descent.

Toplu Dereceli İniş: Toplu Dereceli İniş, her adımda tüm eğitim seti üzerinde hesaplama gerektirir (gradyan inişinin her adımında yer alır) ve bu nedenle çok büyük eğitim setlerinde oldukça yavaştır. Sonuç olarak, Batch Gradient Descent, hesaplama açısından son derece pahalı hale gelir. Bu, dışbükey veya biraz pürüzsüz olan hata manifoldları için idealdir. Batch Gradient Descent, özelliklerin sayısı arttıkça güzel bir şekilde ölçeklenir.



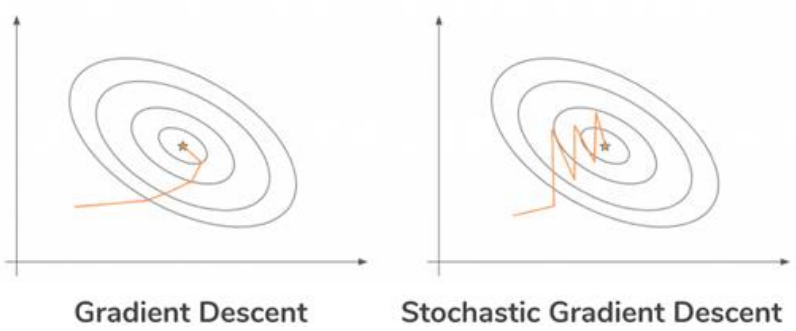
Explain Stochastic Gradient Descent. How is it different from Batch Gradient Descent?

Stochastic Gradient Descent:

Stokastik Gradyan İnişini açıklayın. Batch Gradient Descent'den farkı nedir?

Stokastik Dereceli İniş: Stokastik Dereceli İniş, her adımda degradeleri hesaplamak için tüm eğitim setinin kullanılması olan Toplu Dereceli İniş ile büyük zorluğun üstesinden gelmeyi amaçlar. Bu, doğası gereği stokastiktir, yani her adımda eğitim verilerinin "rastgele" bir örneğini seçer ve ardından gradyanı hesaplar; bu, aynı anda değiştirilecek çok daha az veri olduğundan Toplu Gradyan İniş'ten önemli ölçüde daha hızlıdır. Stokastik Gradyan İnişi,

kısıtlanmamış optimizasyon problemleri için en uygun olanıdır. SGD'nin stokastik doğasının bir dezavantajı vardır, çünkü bir kez minimum değere yaklaştığında sabitlenmez ve bunun yerine sekerek bize model parametreleri için iyi ama optimal olmayan bir değer verir. Bu, zıplamayı azaltacak ve bir süre sonra SGD'nin küresel minimumda yerleşmesine izin verecek olan her adımda öğrenme oranını düşürerek çözülebilir.
İkisi arasındaki farklar şunlardır:



Toplu Gradyan İniş

- Gradyan, tüm eğitim veri kümesi kullanılarak hesaplanır.
- Stokastik Gradyan İnişten daha yavaştır ve hesaplama açısından daha pahalıdır.
- Büyük eğitim örnekleri için önerilmez.
- Doğada deterministiktir (rastgele değil).
- Yakınsamak için yeterli zaman verildiğinde, en iyi cevabı verir.
- Veri noktalarını rastgele karıştırmaya gerek yoktur.
- Bunda sığ yerel minimumlardan çıkmak zordur.
- Bu durumda yakınsama yavaştır.

Stokastik Gradyan İniş

- Gradyanı hesaplamak için tek bir eğitim örneği kullanılır.
- Batch Gradient Descent'den daha hızlı ve hesaplama açısından daha ucuzdur.
- Büyük eğitim örnekleri için önerilir.
- Doğası gereği stokastiktir (rastgele).
- İyi bir çözüm sağlar, ancak en iyisi değildir.
- Veri örneğinin rastgele bir sırada olmasını istediğimiz için, her dönem için eğitim setini karıştıracağız.
- Sığ yerel minimumlardan kaçma şansı daha yüksektir.
- Yakınsama noktasına çok daha hızlı ulaşır**.**

Which deep learning algorithm is the best for face detection?

Yüz tanıma, çeşitli makine öğrenimi yöntemleri kullanılarak gerçekleştirilebilir, ancak en iyileri Evrişimli Sinir Ağları ve derin öğrenmeyi kullanır. Aşağıdakiler bazı dikkate değer yüz algılama algoritmalarıdır: FaceNet, Probablisit, Face Embedding, ArcFace, Cosface ve Spherface.

While building a neural network architecture, how will you decide how many neurons and the hidden layers should the neural network have?

Bir sinir ağı mimarisi oluştururken, sinir ağının kaç nörona ve gizli katmanlara sahip olması gerektiğine nasıl karar vereceksiniz?

Bir iş problemi göz önüne alındığında, bir sinir ağı mimarisi tasarlamak için gereken nöronların ve gizli katmanların tam sayısını belirlemek için açık ve hızlı bir kural yoktur. Bir sinir ağındaki gizli katmanın boyutu, çıktı katmanlarının boyutu ile girdi katmanlarının boyutu arasında bir yerde olmalıdır. Bununla birlikte, bir sinir ağı mimarisi oluşturmaya başlamanıza yardımcı olabilecek birkaç temel yol vardır:

- Herhangi bir benzersiz gerçek dünya öngörücü modelleme problemine yaklaşmanın en iyi yöntemi, benzer gerçek dünya durumlarında sinir ağlarıyla çalışma önceki deneyimlerine dayalı olarak herhangi bir veri kümesi için neyin en iyi performansı göstereceğini görmek için bazı temel sistematik deneylerle başlamaktır. Ağ yapılandırması, kişinin problem alanı anlayışına ve sinir ağlarıyla ilgili önceki uzmanlığına dayalı olarak seçilebilir. Benzer konularda kullanılan katmanların ve nöronların sayısı, bir sinir ağının konfigürasyonunu değerlendirirken her zaman iyi bir başlangıç noktasıdır.
- Basit sinir ağı mimarisiyle başlamak ve tahmin edilen çıktı ve doğruluğa dayalı olarak sinir ağının karmaşıklığını kademeli olarak artırmak en iyisidir.

Can a deep learning model be solely built on linear regression?

Derin öğrenme modeli yalnızca doğrusal regresyon üzerine inşa edilebilir mi?

Evet, sorun doğrusal bir denklemle temsil ediliyorsa, her katman için aktivasyon işlevi olarak doğrusal bir işlev kullanılarak derin ağlar oluşturulabilir. Öte yandan, doğrusal işlevlerin bileşimi olan bir sorun doğrusal bir işlevdir ve derin bir ağ uygulayarak başarılabilecek olağanüstü bir şey yoktur çünkü ağa daha fazla düğüm eklemek makine öğrenme modelinin tahmin kapasitesini artırmaz. .

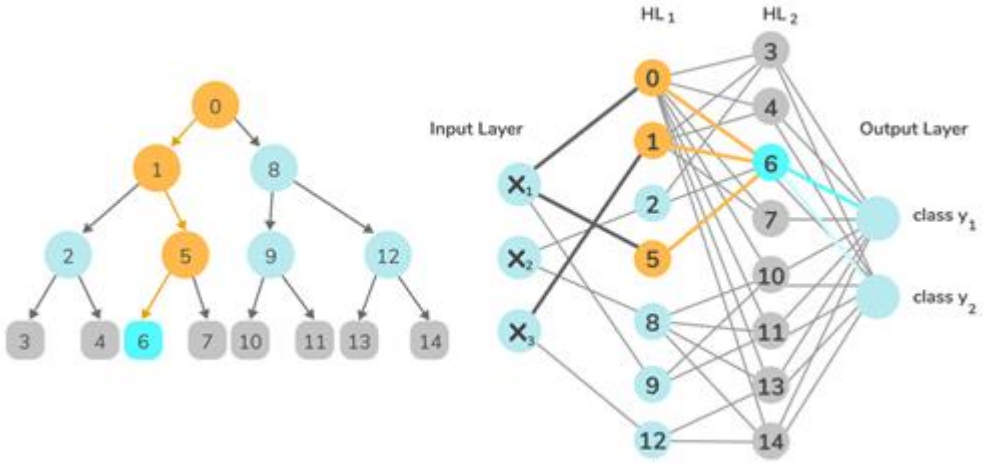
According to you, which one is more powerful - a two layer neural network without any activation function or a two layer decision tree?

Size göre hangisi daha güçlü - aktivasyon fonksiyonu olmayan iki katmanlı bir sinir ağı mı yoksa iki katmanlı bir karar ağacı mı?

İki katmanlı bir sinir ağı, üç katmandan oluşur: bir giriş katmanı, bir gizli katman ve bir çıkış katmanı. Sinir ağlarıyla uğraşırken, girdiler ve yanıt değişkenleri arasındaki karmaşık ve doğrusal olmayan işlevsel eşlemelerle uğraşırken gerekli olduğu için bir etkinleştirme işlevi önemlidir. İki katmanlı bir sinir ağında aktivasyon fonksiyonu olmadığında, bu sadece doğrusal bir ağdır. Aktivasyon işlevi olmayan bir Sinir Ağı, yalnızca sınırlı kapasiteye sahip olan ve sıklıkla iyi performans göstermeyen bir Lineer Regresyon Modelidir.

İki katman derinliğine sahip bir karar ağacı, iki katmanlı bir karar ağacı olarak bilinir. Karar Ağaçları, verilerin bir parametreye göre sürekli olarak bölündüğü bir tür denetimli makine öğrenimidir (yani, makine, eğitim verilerinde girdinin ne olduğu ve ilgili çıktının ne olduğu ile beslenir). Ağacı açıklamak için iki varlık, karar düğümleri ve yapraklar kullanılabilir. Kararlar veya nihai sonuçlar yapraklarla temsil edilir. Ve veriler karar düğümlerinde ayrılır.

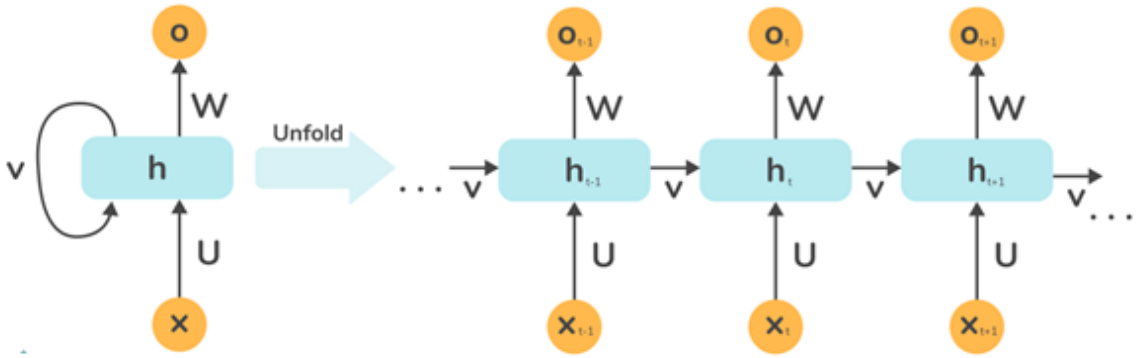
Bu iki modeli karşılaştırırken, iki katmanlı sinir ağı (aktivasyon fonksiyonu olmadan) iki katmanlı karar ağacından daha güçlüdür, çünkü iki katmanlı sinir ağı bir model oluştururken daha fazla özelliği dikkate alacaktır, oysa iki katmanlı karar ağaç yalnızca 2 veya 3 özelliği dikkate alacaktır.



Soldaki şekil 2 katmanlı bir karar ağacını ve sağdaki şekil 2 katmanlı bir sinir ağını göstermektedir.

How does Recurrent Neural Network backpropagation vary from Artificial Neural Network backpropagation?
Tekrarlayan Sinir Ağı geri yayılımı, Yapay Sinir Ağı geri yayılımından nasıl farklıdır?

Tekrarlayan Sinir Ağlarında Geri Yayılım, aşağıdaki resimde gösterildiği gibi Tekrarlayan Sinir Ağlarındaki her düğümün ek bir döngüye sahip olması anlamında Yapay Sinir Ağlarından farklıdır:



Bu döngü, özünde, ağa geçici bir bileşen ekler. Bu, genel bir yapay sinir ağı ile imkansız olan, verilerden sıralı bilgilerin yakalanmasına izin verir.

What exactly do you mean by exploding and vanishing gradients?
Patlayan ve kaybolan gradyanlarla tam olarak ne demek istiyorsunuz?

Minimum değere doğru artan adımlar atarak, gradyan iniş algoritması hatayı en aza indirmeyi amaçlar. Bir sinir ağındaki ağırlıklar ve yanlılıklar bu işlemler kullanılarak güncellenir.

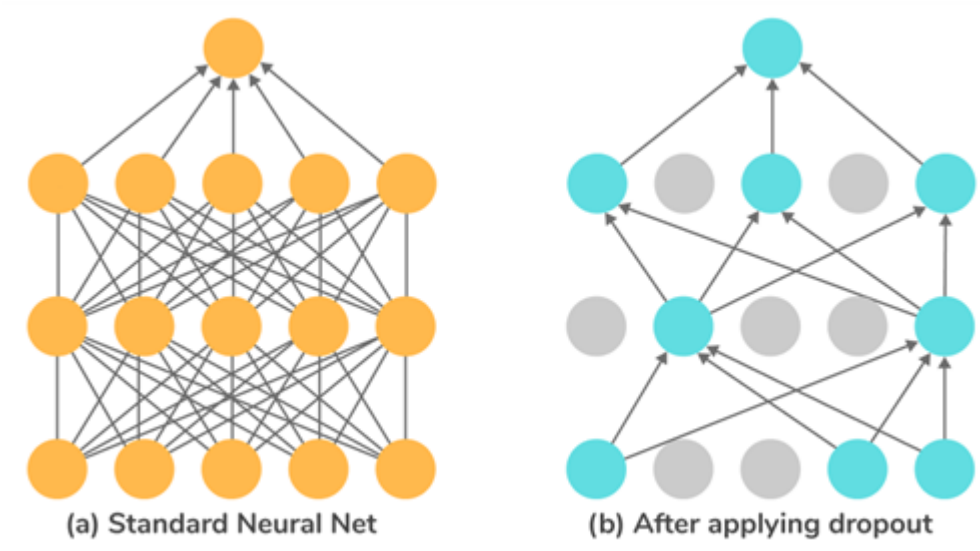
Bununla birlikte, bazen adımlar aşırı derecede büyür ve ağırlıkların ve yanlılık terimlerinin güncellemelerinin artmasıyla sonuçlanır - ağırlıkların taştığı (veya NaN, yani Sayı Değil) noktaya kadar. Patlayan bir gradyan bunun sonucudur ve kararsız bir yöntemdir.

Öte yandan, adımlar aşırı derecede küçükse, ağırlıklarda ve sapma terimlerinde küçük, hatta ihmal edilebilir değişikliklerle sonuçlanır. Sonuç olarak, her seferinde neredeyse aynı ağırlıklara ve yanlıklara sahip, hiçbir zaman en az hata işlevine ulaşmayan bir derin öğrenme modeli eğitebiliriz. Kaybolan gradyan buna denir.

What do you know about Dropout?

Bırakma hakkında ne biliyorsunuz?

Bırakma, fazla uydurmayı önlemeye yardımcı olan ve dolayısıyla genellenebilirliği artıran bir düzenleme yaklaşımıdır (yani model, yalnızca eğitim veri seti ile sınırlı olmak yerine, genel olarak girdilerin çoğu için doğru çıktıyı tahmin eder). Genel olarak, %20 ile iyi bir başlangıç noktası olmak üzere nöronların yüzde 20 ila yüzde 50'si gibi düşük bir bırakma değeri kullanmalıyız. Çok düşük bir olasılığın etkisi yoktur, oysa çok yüksek bir sayı ağın yetersiz öğrenmesine neden olur.



Daha büyük bir ağda bırakma yöntemini kullandığınızda, modelin bağımsız temsilleri öğrenmek için daha fazla fırsatı olduğundan daha iyi sonuçlar elde etme olasılığınız daha yüksektir.

4. Algorithms used in Deep Learning

Here is the list of top 10 most popular deep learning algorithms:

1. Convolutional Neural Networks (CNNs)
2. Long Short Term Memory Networks (LSTMs)
3. Recurrent Neural Networks (RNNs)
4. Generative Adversarial Networks (GANs)
5. Radial Basis Function Networks (RBFNs)
6. Multilayer Perceptrons (MLPs)
7. Self Organizing Maps (SOMs)
8. Deep Belief Networks (DBNs)
9. Restricted Boltzmann Machines(RBMs)
10. Autoencoders

Derin öğrenme algoritmaları neredeyse her tür veriyle çalışır ve karmaşık sorunları çözmek için büyük miktarda bilgi işlem gücü ve bilgi gerektirir.

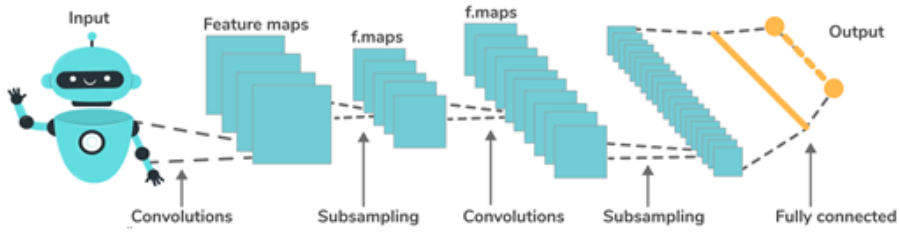
What do you understand about Neural Networks in the context of Deep Learning?
Derin Öğrenme bağlamında Sinir Ağları hakkında ne anlıyorsunuz?

Sinir Ağları, insan vücudundaki biyolojik sinir ağlarına çok benzeyen yapay sistemlerdir. Bir sinir ağı, insan beyninin nasıl çalıştığını taklit eden bir yöntem kullanarak bir veri yığınındaki temel ilişkileri tanımaya çalışan bir dizi algoritmadır. Göreve özel kurallar olmadan, bu sistemler çeşitli veri kümelerine ve örneklere maruz kalarak görevleri yapmayı öğrenirler. Buradaki fikir, bu veri kümelerinin önceden kodlanmış bir anlayışıyla programlanmak yerine, sistemin beslendiği verilerden tanımlayıcı özellikleri türetmesidir. Sinir ağları, eşik mantık hesaplama modelleri üzerine kuruludur. Sinir ağları, değişen girdilere uyum sağlayabildiğinden, çıktı kriterlerinin yeniden tasarlanmasını gerektirmeden mümkün olan en iyi sonucu üretebilirler.

4.1. Convolutional Neural Networks (CNNs)

Konvolüsyonel Sinir Ağları çoğunlukla bilgisayar görsellerde kullanılır. MLP'lerdeki tam bağlantılı katmanların aksine, bir veya daha fazla evrişim katmanı, CNN modellerinde evrişim işlemleri gerçekleştirerek girdiden basit özellikler çıkarır. Her katman, önceki katmanın çıktılarının uzamsal olarak yakın alt kümelerinin çeşitli koordinatlarındaki ağırlıklı toplamların doğrusal olmayan işlevlerinden oluşur ve ağırlıkların yeniden kullanılmasına izin verir.

AI sistemi, gerçek dünyadan bir dizi görüntü veya video verildiğinde resim sınıflandırma, yüz tanımlama ve görüntü anlamsal bölümleme gibi belirli bir görevi yerine getirmek için bu girdilerin özelliklerini otomatik olarak çıkarmayı öğrenir.

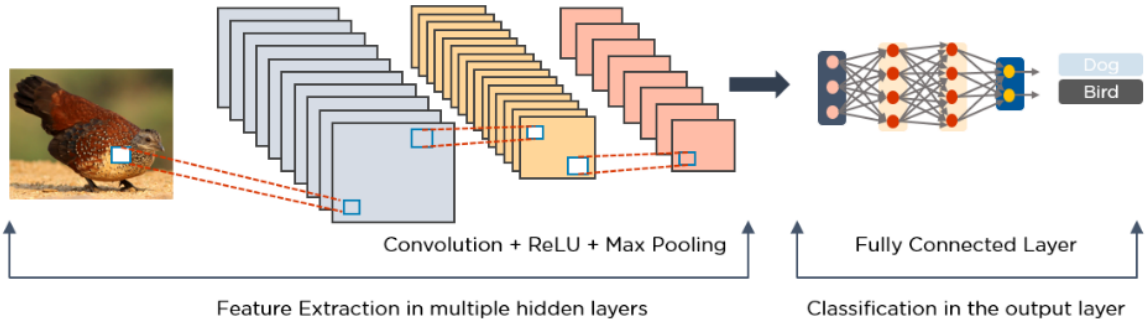


ConvNets olarak da bilinen CNN'ler, çoklu katmanlardan oluşur ve esas olarak görüntü işleme ve nesne algılama için kullanılır. Yann LeCun, ilk CNN'i 1988'de LeNet olarak adlandırıldığında geliştirdi. Posta kodları ve rakamlar gibi karakterleri tanımak için kullanıldı.
CNN'ler, uydu görüntülerini tanımlamak, tıbbi görüntüleri işlemek, zaman serilerini tahmin etmek ve anormallikleri tespit etmek için yaygın olarak kullanılmaktadır.

CNN'ler Nasıl Çalışır?

CNN'ler, verilerden özellikleri işleyen ve çıkaran birden çok katmana sahiptir:
Evrişim Katmanı: CNN, evrişim işlemini gerçekleştirmek için birkaç filtreye sahip bir evrişim katmanına sahiptir.
Rektifiye Lineer Birim (ReLU): CNN'ler, öğeler üzerinde işlemleri gerçekleştirmek için bir ReLU katmanına sahiptir. Çıktı, düzeltilmiş bir özellik haritasıdır.
Havuzlama Katmanı: Düzeltilmiş özellik haritası daha sonra bir havuz katmanına beslenir. Havuzlama, özellik haritasının boyutlarını azaltan bir alt örnekleme işlemidir. Havuzlama katmanı daha sonra havuzlanmış özellik haritasından elde edilen iki boyutlu dizileri düzleştirerek tek, uzun, sürekli, doğrusal bir vektöre dönüştürür.
Tam Bağlantılı Katman: Havuzlama katmanından gelen düzleştirilmiş matris, görüntüleri sınıflandıran ve tanımlayan bir girdi olarak beslendiğinde, tam bağlantılı bir katman oluşur.

Aşağıda CNN aracılığıyla işlenen bir görüntü örneği bulunmaktadır.



In a Convolutional Neural Network (CNN), how can you fix the constant validation accuracy?
Bir Evrişimli Sinir Ağında (CNN), sabit doğrulama doğruluğunu nasıl düzeltebilirsiniz?

Herhangi bir sinir ağını eğitirken, sürekli doğrulama doğruluğu yaygın bir sorundur, çünkü ağ yalnızca örneği hatırlayarak aşırı uyum sorununa neden olur. Bir modele aşırı uydurma, sinir ağı modelinin eğitim örneğinde takdire şayan bir performans gösterdiğini, ancak modelin performansının doğrulama setinde bozulduğunu gösterir. CNN'nin sürekli doğrulama doğruluğunu iyileştirmenin bazı yolları aşağıda verilmiştir:

- Veri setini üç bölüme ayırmak her zaman iyi bir fikirdir: eğitim, doğrulama ve test etme.
- Sınırlı verilerle çalışırken, bu zorluk, sinir ağının parametreleriyle deneyler yapılarak aşılabilir.
- Eğitim veri setinin boyutunu artırarak.
- Toplu normalleştirme kullanarak.
- Düzenlileştirme uygulayarak
- Ağın karmaşıklığını azaltarak

4.2. Long Short Term Memory Networks (LSTMs)

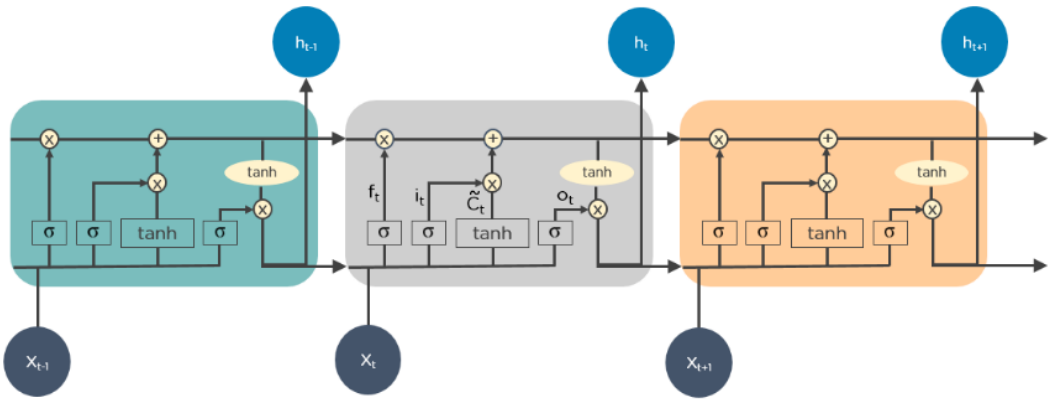
LSTMs are a type of Recurrent Neural Network (RNN) that can learn and memorize long-term dependencies. Recalling past information for long periods is the default behavior.

LSTMs retain information over time. They are useful in time-series prediction because they remember previous inputs. LSTMs have a chain-like structure where four interacting layers communicate in a unique way. Besides time-series predictions, LSTMs are typically used for speech recognition, music composition, and pharmaceutical development.

How Do LSTMs Work?

- First, they forget irrelevant parts of the previous state
- Next, they selectively update the cell-state values
- Finally, the output of certain parts of the cell state

Below is a diagram of how LSTMs operate:

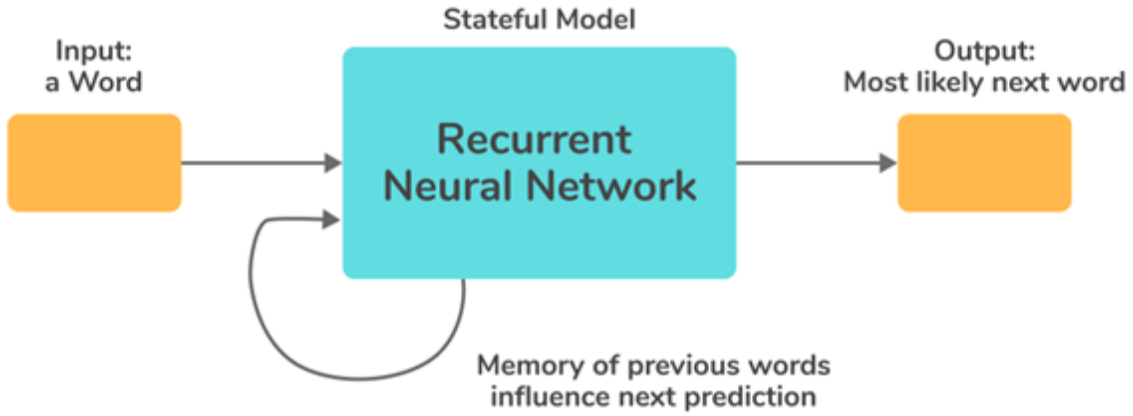


4.3. Recurrent Neural Networks (RNNs)

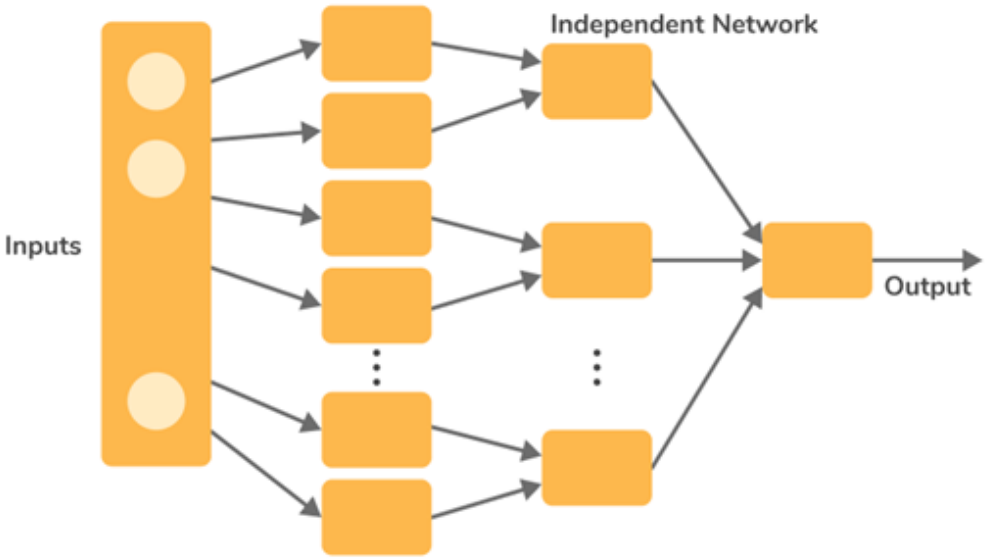
Recurrent Neural Network (RNN):
Tekrarlayan Sinir Ağları:

Sıralı giriş verisi zaman serisi problemini çözmek için Tekrarlayan Sinir Ağları oluşturuldu. RNN'nin girişi, mevcut giriş ve önceki örneklerden oluşur. Sonuç olarak, düğüm bağlantıları yönlendirilmiş bir grafik oluşturur. Ayrıca, bir RNN'deki her nöronun, önceki örneklerin hesaplamalarından elde edilen bilgileri depolayan bir dahili belleği vardır. RNN modelleri, değişken giriş uzunluğuna sahip verileri işlemedeki üstünlüklerinden dolayı, doğal dil işlemede (NLP) yaygın olarak kullanılır. Bu durumda AI'nın amacı, doğal dil modelleme, kelime yerleştirme ve makine çevirisi gibi insan tarafından konuşulan doğal dilleri anlayabilen bir sistem oluşturmaktır.

Bir RNN'deki her ardışık katman, ağırlıklı çıktı toplamlarının ve önceki durumun doğrusal olmayan fonksiyonlarından oluşur. Sonuç olarak, RNN'nin temel birimi "hücre" olarak adlandırılır ve her hücre katmanlardan ve tekrarlayan sinir ağı modellerinin sıralı olarak işlenmesine izin veren bir dizi hücreden oluşur.



Modular Neural Network:
Bu ağ, tek bir ağ olmaktan ziyade çok sayıda küçük sinir ağlarından oluşur. Alt ağlar, ortak bir hedefe ulaşmak için bağımsız olarak çalışan daha büyük bir sinir ağı oluşturmak için birleşir. Bu ağlar, büyük-küçük bir sorunu daha küçük parçalara bölmek ve sonra onu çözmek için son derece kullanışlıdır.



Sequence to Sequence Model:

Sıradan Sıraya Model:

Çoğu durumda, bu ağ iki RNN ağından oluşur. Ağ, kodlama ve kod çözmeye dayalıdır; bu, girdiyi işleyen bir kodlayıcıya ve çıktıyı işleyen bir kod çözücüye sahip olduğu anlamına gelir. Bu ağ türü, giren metnin uzunluğu, çıkan metnin uzunluğundan farklı olduğunda, genellikle metin işleme için kullanılır.

4.4. Deep Belief Networks (DBNs)

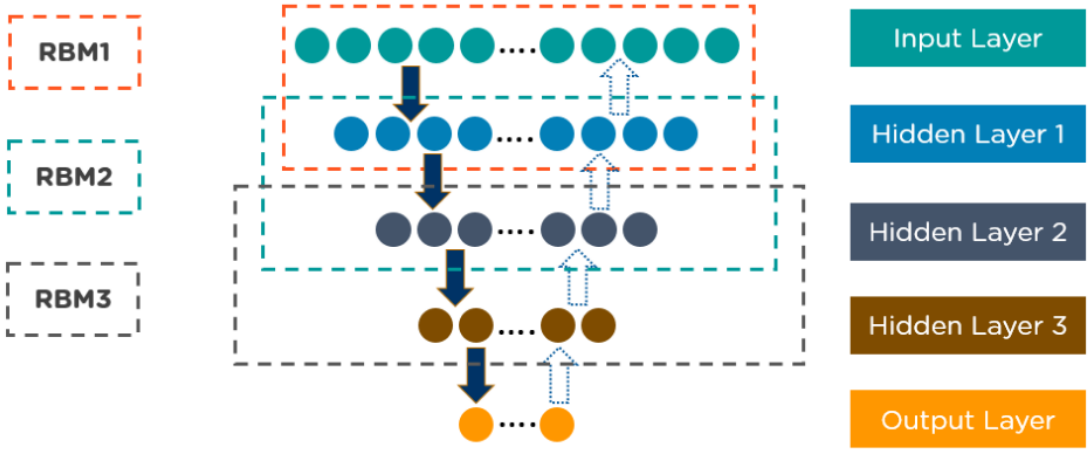
DBN'ler, birden çok stokastik, gizli değişken katmanından oluşan üretken modellerdir. Gizli değişkenler ikili değerlere sahiptir ve genellikle gizli birimler olarak adlandırılır.

DBN'ler, katmanlar arasında bağlantıları olan bir Boltzmann Makinesi yığınıdır ve her bir RBM katmanı, hem önceki hem de sonraki katmanlarla iletişim kurar. Derin İnanç (Kanı) Ağları (DBN'ler) görüntü tanıma, video tanıma ve hareket yakalama verileri için kullanılır.

DBN'ler Nasıl Çalışır?

- Açgözlü öğrenme algoritmaları DBN'leri eğitir. Açgözlü algoritma, her aşamada yerel olarak en uygun seçimi yapma problem çözme buluşsal yöntemini izleyen herhangi bir algoritmadır. Açgözlü öğrenme algoritması, yukarıdan aşağıya, üretken ağırlıkları öğrenmek için katman katman bir yaklaşım kullanır.
- DBN'ler, Gibbs örnekleme adımlarını en üstteki iki gizli katmanda çalıştırır. Bu aşama, en üstteki iki gizli katman tarafından tanımlanan RBM'den bir örnek alır.
- DBN'ler, modelin geri kalanı boyunca tek bir atasal örnekleme geçişini kullanarak görünür birimlerden bir örnek alır.
- DBN'ler, her katmandaki gizli değişkenlerin değerlerinin aşağıdan yukarıya tek bir geçişle çıkarılabileceğini öğrenir.

Below is an example of DBN architecture:



4.5. Autoencoders

What are autoencoders? Explain the different layers of autoencoders.
Otomatik kodlayıcılar nelerdir? Otomatik kodlayıcıların farklı katmanlarını açıklayın.

Otomatik kodlayıcı, çıktı katmanının giriş katmanıyla aynı boyuta sahip olması koşuluyla bir tür sinir ağıdır. Başka bir deyişle, çıktı katmanındaki çıktı birimlerinin sayısı, girdi katmanındaki girdi birimlerinin sayısına eşittir. Otomatik kodlayıcı, girdiden çıktıya verileri denetimsiz bir şekilde çoğalttığı için çoğaltıcı sinir ağı olarak da bilinir.

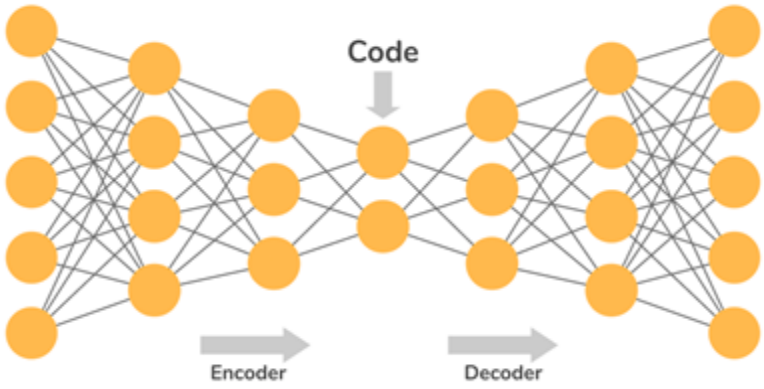
Girişi ağ üzerinden göndererek, otomatik kodlayıcılar girişin her bir boyutunu yeniden oluşturur. Bir girdiyi çoğaltmak için bir sinir ağı kullanmak basit görünebilir, ancak çoğaltma işlemi sırasında girdinin boyutu küçülür, bu da daha küçük bir temsille sonuçlanır. Giriş ve çıkış katmanlarına kıyasla, sinir ağının orta katmanları daha az birime sahiptir. Sonuç olarak, girdinin azaltılmış temsili orta katmanlarda depolanır. Girdinin bu azaltılmış temsili, çıktıyı yeniden oluşturmak için kullanılır.

Otomatik kodlayıcıların mimarisindeki farklı katmanlar şunlardır:

Kodlayıcı: Kodlayıcı, giriş görüntüsünü bir gizli uzay temsiline sıkıştıran ve onu daha düşük bir boyutta sıkıştırılmış bir temsil olarak kodlayan, tamamen bağlı, ileri beslemeli bir sinir ağıdır. Orijinal görüntünün deforme olmuş temsili sıkıştırılmış görüntüdür.

Kod: Kod çözücüye sağlanan girişin azaltılmış gösterimi ağın bu bölümünde saklanır.

Kod çözücü: Kodlayıcı gibi, kod çözücü de kodlayıcıyla aynı yapıya sahip ileri beslemeli bir ağdır. Bu ağ, koddan gelen girdiyi orijinal boyutlarına yeniden monte etmekten sorumludur.



Yukarıdaki resimde gördüğümüz gibi, girdi kodlayıcıda sıkıştırılır, ardından Kodda saklanır ve ardından orijinal girdi kod çözücü tarafından koddan açılır. Otomatik kodlayıcının temel amacı, girdiyle aynı olan bir çıktı sağlamaktır.

Mention the applications of autoencoders.
Otomatik kodlayıcıların uygulamalarından bahsedin.
Otomatik kodlayıcıların uygulamaları şunlardır: -

Görüntü Gürültüsünü Giderme: Görüntüleri gürültüden arındırma, otomatik kodlayıcıların çok başarılı olduğu bir beceridir. Gürültülü bir görüntü, bozulmuş veya içinde az miktarda parazit (yani, görüntülerde rastgele parlaklık veya renk bilgisi değişimi) bulunan görüntüdür. Görüntü denoising, görüntünün içeriği hakkında doğru bilgi elde etmek için kullanılır.

Boyutsallık Azaltma: Giriş, kod adı verilen orta katmanda depolanan otomatik kodlayıcılar tarafından azaltılmış bir temsile dönüştürülür. Bu, girdiden gelen bilgilerin sıkıştırıldığı yerdir ve artık her düğüm, bu katman modelden çıkarılarak bir değişken olarak ele alınabilir. Sonuç olarak, kod çözücüyü kaldırarak, çıktı olarak kodlama katmanı ile boyut azaltma için bir otomatik kodlayıcının kullanılabileceği sonucuna varabiliriz.

Özellik Çıkarma: Otomatik Kodlayıcıların kodlama bölümü, yeniden yapılandırma hatasını azaltarak giriş verilerinde bulunan önemli gizli özelliklerin öğrenilmesine yardımcı olur. Kodlama sırasında, orijinal özellik kombinasyonlarından oluşan yeni bir koleksiyon oluşturulur.

Görüntü Renklendirme: Siyah beyaz bir görüntüyü renkli bir görüntüye dönüştürmek, otomatik kodlayıcıların uygulamalarından biridir. Renkli bir görüntüyü gri tonlamaya da dönüştürebiliriz.

Veri Sıkıştırma: Veri sıkıştırma için otomatik kodlayıcılar kullanılabilir. Yine de, aşağıdaki nedenlerden dolayı nadiren veri sıkıştırma için kullanılırlar: Kayıplı sıkıştırma: Otomatik kodlayıcının çıkışı girişle aynı değildir, ancak yakın fakat bozulmuş bir temsildir. Kayıpsız sıkıştırma için en iyi seçenek değildirler. Veriye özel: Otomatik kodlayıcılar, yalnızca eğitildikleri verilerle aynı olan verileri sıkıştırabilir. Sağlanan eğitim verileriyle ilgili özellikleri öğrenmeleri bakımından jpeg veya gzip gibi geleneksel veri sıkıştırma algoritmalarından farklıdırlar. Sonuç olarak, elle yazılmış rakamlarla eğitilmiş bir otomatik kodlayıcı tarafından sıkıştırılacak bir manzara fotoğrafını tahmin edemeyiz.

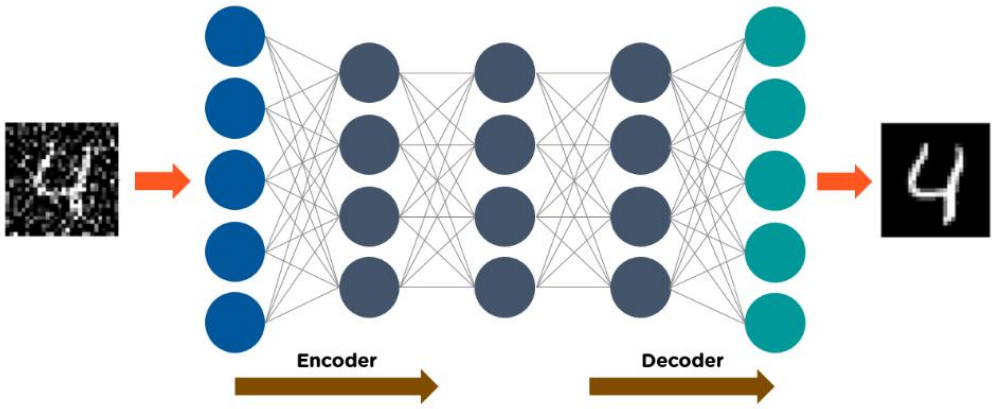
Otomatik kodlayıcılar, giriş ve çıkışın aynı olduğu belirli bir ileri beslemeli sinir ağı türüdür. Geoffrey Hinton, 1980'lerde denetimsiz öğrenme sorunlarını çözmek için otomatik kodlayıcılar tasarladı. Verileri girdi katmanından çıktı katmanına kopyalayan eğitimli sinir ağlarıdır. Otomatik kodlayıcılar, farmasötik keşif, popülerlik tahmini ve görüntü işleme gibi amaçlar için kullanılır.

Otomatik Kodlayıcılar Nasıl Çalışır?

Bir otomatik kodlayıcı üç ana bileşenden oluşur: kodlayıcı, kod ve kod çözücü.

- Otomatik kodlayıcılar, bir girdi alacak ve onu farklı bir temsile dönüştürecek şekilde yapılandırılmıştır. Daha sonra orijinal girdiyi mümkün olduğu kadar doğru bir şekilde yeniden oluşturmaya çalışırlar.
- Bir rakamın görüntüsü net olarak görülmediğinde, otomatik kodlayıcı sinir ağına beslenir.
- Otomatik kodlayıcılar önce görüntüyü kodlar, ardından girdinin boyutunu daha küçük bir temsile küçültür.
- Son olarak, otomatik kodlayıcı, yeniden oluşturulmuş görüntüyü oluşturmak için görüntünün kodunu çözer.

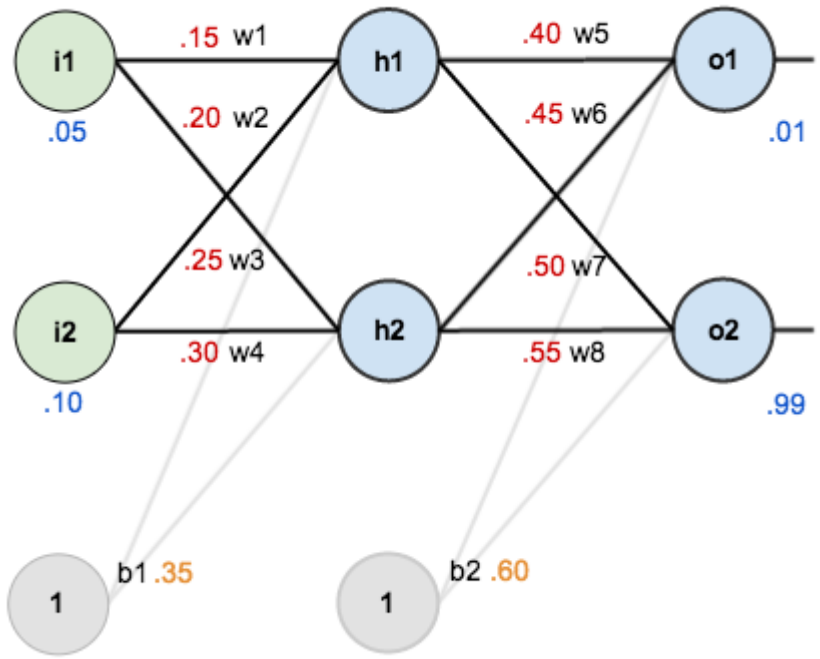
Aşağıdaki resim, otomatik kodlayıcıların nasıl çalıştığını gösterir:



5. Örnekler

NN Example 1

In order to have some numbers to work with, here are the initial weights, the biases, and training inputs/outputs:



The goal of backpropagation is to optimize the weights so that the neural network can learn how to correctly map arbitrary inputs to outputs. For the rest of this tutorial we're going to work with a single training set: given inputs 0.05 and 0.10, we want the neural network to output 0.01 and 0.99.

The Forward Pass

To begin, lets see what the neural network currently predicts given the weights and biases above and inputs of 0.05 and 0.10. To do this we'll feed those inputs forward though the network. We figure out the *total net input* to each hidden layer neuron, *squash* the total net input using an *activation function* (here we use the *logistic function*), then repeat the process with the output layer neurons.

Here's how we calculate the total net input for h_1:

$$net_{h1} = w_1 * i_1 + w_2 * i_2 + b_1 * 1$$

$$net_{h1} = 0.15 * 0.05 + 0.2 * 0.1 + 0.35 * 1 = 0.3775$$

We then squash it using the logistic function to get the output of h_1:

$$out_{h1} = \frac{1}{1+e^{-net_{h1}}} = \frac{1}{1+e^{-0.3775}} = 0.593269992$$

Carrying out the same process for h_2 we get:

$$out_{h2} = 0.596884378$$

We repeat this process for the output layer neurons, using the output from the hidden layer neurons as inputs.

Here's the output for o_1:

$$net_{o1} = w_5 * out_{h1} + w_6 * out_{h2} + b_2 * 1$$
$$net_{o1} = 0.4 * 0.593269992 + 0.45 * 0.596884378 + 0.6 * 1 = 1.105905967$$
$$out_{o1} = \frac{1}{1+e^{-net_{o1}}} = \frac{1}{1+e^{-1.105905967}} = 0.75136507$$

And carrying out the same process for o_2 we get:

$$out_{o2} = 0.772928465$$

Calculating the Total Error

We can now calculate the error for each output neuron using the **squared error function** and sum them to get the total error:

$$E_{total} = \sum \frac{1}{2}(target - output)^2$$

Some sources refer to the target as the *ideal* and the output as the *actual*.

The $\frac{1}{2}$ is included so that exponent is cancelled when we differentiate later on. The result is eventually multiplied by a learning rate anyway so it doesn't matter that we introduce a constant here. For example, the target output for o_1 is 0.01 but the neural network output 0.75136507, therefore its error is:

$$E_{o1} = \frac{1}{2}(target_{o1} - out_{o1})^2 = \frac{1}{2}(0.01 - 0.75136507)^2 = 0.274811083$$

Repeating this process for o_2 (remembering that the target is 0.99) we get:

$$E_{o2} = 0.023560026$$

The total error for the neural network is the sum of these errors:

$$E_{total} = E_{o1} + E_{o2} = 0.274811083 + 0.023560026 = 0.298371109$$

The Backwards Pass

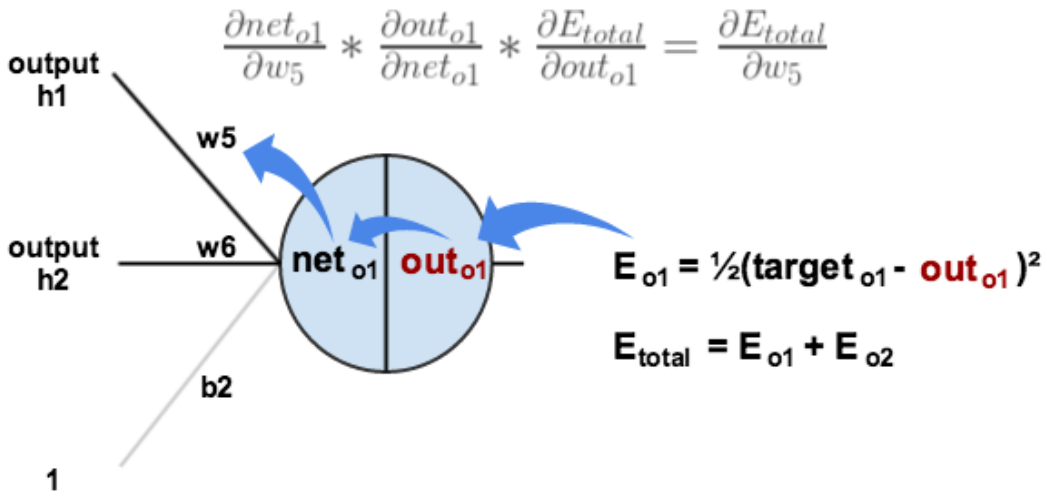
Our goal with backpropagation is to update each of the weights in the network so that they cause the actual output to be closer the target output, thereby minimizing the error for each output neuron and the network as a whole.

Output Layer

Consider w_5. We want to know how much a change in w_5 affects the total error, aka $\frac{\partial E_{total}}{\partial w_5}$. $\frac{\partial E_{total}}{\partial w_5}$ is read as "the partial derivative of E_{total} with respect to w_5". You can also say "the gradient with respect to w_5".

By applying the **chain rule** we know that:

Visually, here's what we're doing:



We need to figure out each piece in this equation.

First, how much does the total error change with respect to the output?

$E_{total} = \frac{1}{2}(target_{o1} - out_{o1})^2 + \frac{1}{2}(target_{o2} - out_{o2})^2$

$\frac{\partial E_{total}}{\partial out_{o1}} = 2 * \frac{1}{2}(target_{o1} - out_{o1})^{2-1} * -1 + 0$

$\frac{\partial E_{total}}{\partial out_{o1}} = -(target_{o1} - out_{o1}) = -(0.01 - 0.75136507) = 0.74136507$

$-(target - out)$ is sometimes expressed as $out - target$

When we take the partial derivative of the total error with respect to out_{o1}, the quantity $\frac{1}{2}(target_{o2} - out_{o2})^2$ becomes zero because out_{o1} does not affect it which means we're taking the derivative of a constant which is zero.

Next, how much does the output of o_1 change with respect to its total net input?

The partial **derivative of the logistic function** is the output multiplied by 1 minus the output:

$out_{o1} = \frac{1}{1+e^{-net_{o1}}}$

$\frac{\partial out_{o1}}{\partial net_{o1}} = out_{o1}(1 - out_{o1}) = 0.75136507(1 - 0.75136507) = 0.186815602$

Finally, how much does the total net input of $o1$ change with respect to w_5?

$$net_{o1} = w_5 * out_{h1} + w_6 * out_{h2} + b_2 * 1$$
$$\frac{\partial net_{o1}}{\partial w_5} = 1 * out_{h1} * w_5^{(1-1)} + 0 + 0 = out_{h1} = 0.593269992$$

Putting it all together:

$$\frac{\partial E_{total}}{\partial w_5} = 0.74136507 * 0.186815602 * 0.593269992 = 0.082167041$$

You'll often see this calculation combined in the form of the **delta rule**:

$$\frac{\partial E_{total}}{\partial w_5} = -(target_{o1} - out_{o1}) * out_{o1}(1 - out_{o1}) * out_{h1}$$

Alternatively, we have $\frac{\partial E_{total}}{\partial out_{o1}}$ and $\frac{\partial out_{o1}}{\partial net_{o1}}$ which can be written as $\frac{\partial E_{total}}{\partial net_{o1}}$, aka δ_{o1} (the Greek letter delta) aka the *node delta*. We can use this to rewrite the calculation above:

$$\delta_{o1} = -(target_{o1} - out_{o1}) * out_{o1}(1 - out_{o1})$$

Therefore:

$$\frac{\partial E_{total}}{\partial w_5} = \delta_{o1} out_{h1}$$

Some sources extract the negative sign from δ so it would be written as:

$$\frac{\partial E_{total}}{\partial w_5} = -\delta_{o1} out_{h1}$$

To decrease the error, we then subtract this value from the current weight (optionally multiplied by some learning rate, eta, which we'll set to 0.5):

$$w_5^+ = w_5 - \eta * \frac{\partial E_{total}}{\partial w_5} = 0.4 - 0.5 * 0.082167041 = 0.35891648$$

Some sources use α (alpha) to represent the learning rate, **others use** η (eta), and **others** even use ϵ (epsilon).

We can repeat this process to get the new weights w_6, w_7, and w_8:

$$w_6^+ = 0.408666186$$
$$w_7^+ = 0.511301270$$
$$w_8^+ = 0.561370121$$

We perform the actual updates in the neural network *after* we have the new weights leading into the hidden layer neurons (ie, we use the original weights, not the updated weights, when we continue the backpropagation algorithm below).

Hidden Layer

Next, we'll continue the backwards pass by calculating new values for w_1, w_2, w_3, and w_4.

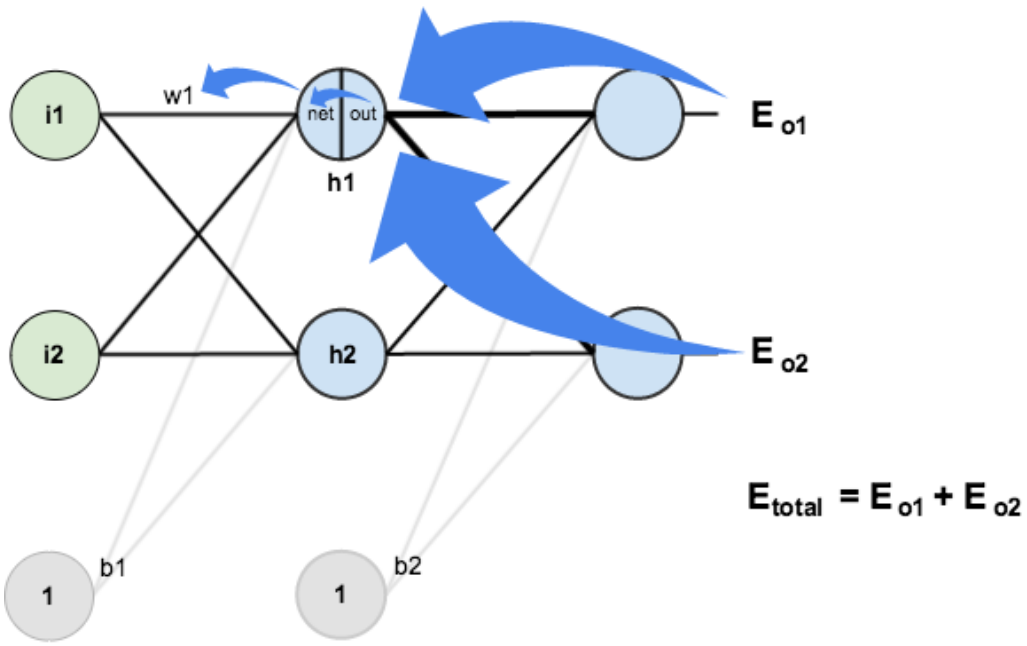
Big picture, here's what we need to figure out:

Visually:

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

$$\downarrow$$

$$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}}$$



We're going to use a similar process as we did for the output layer, but slightly different to account for the fact that the output of each hidden layer neuron contributes to the output (and therefore error) of multiple output neurons. We know that out_{h1} affects both out_{o1} and out_{o2} therefore the $\frac{\partial E_{total}}{\partial out_{h1}}$ needs to take into consideration its effect on the both output neurons:

Starting with $\frac{\partial E_{o1}}{\partial out_{h1}}$:

We can calculate $\frac{\partial E_{o1}}{\partial net_{o1}}$ using values we calculated earlier:

And $\frac{\partial net_{o1}}{\partial out_{h1}}$ is equal to w_5:

$$net_{o1} = w_5 * out_{h1} + w_6 * out_{h2} + b_2 * 1$$
$$\frac{\partial net_{o1}}{\partial out_{h1}} = w_5 = 0.40$$

Plugging them in:

Following the same process for $\frac{\partial E_{o2}}{\partial out_{h1}}$, we get:
$\frac{\partial E_{o2}}{\partial out_{h1}} = -0.019049119$

Therefore:

Now that we have $\frac{\partial E_{total}}{\partial out_{h1}}$, we need to figure out $\frac{\partial out_{h1}}{\partial net_{h1}}$ and then $\frac{\partial net_{h1}}{\partial w}$ for each weight:
$out_{h1} = \frac{1}{1+e^{-net_{h1}}}$
$\frac{\partial out_{h1}}{\partial net_{h1}} = out_{h1}(1 - out_{h1}) = 0.59326999(1 - 0.59326999) = 0.241300709$

We calculate the partial derivative of the total net input to h_1 with respect to w_1 the same as we did for the output neuron:
$net_{h1} = w_1 * i_1 + w_3 * i_2 + b_1 * 1$
$\frac{\partial net_{h1}}{\partial w_1} = i_1 = 0.05$

Putting it all together:

$\frac{\partial E_{total}}{\partial w_1} = 0.036350306 * 0.241300709 * 0.05 = 0.000438568$

You might also see this written as:

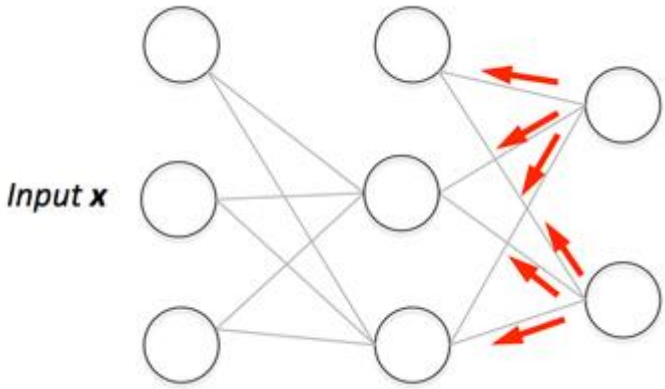
$\frac{\partial E_{total}}{\partial w_1} = \delta_{h1} i_1$

We can now update w_1:
$w_1^+ = w_1 - \eta * \frac{\partial E_{total}}{\partial w_1} = 0.15 - 0.5 * 0.000438568 = 0.149780716$

Repeating this for w_2, w_3, and w_4
$w_2^+ = 0.19956143$
$w_3^+ = 0.24975114$
$w_4^+ = 0.29950229$

Finally, we've updated all of our weights! When we fed forward the 0.05 and 0.1 inputs originally, the error on the network was 0.298371109. After this first round of backpropagation, the total error is now down to 0.291027924. It might not seem like much, but after repeating this process 10,000 times, for example, the error plummets to 0.0000351085. At this point, when we feed forward 0.05 and 0.1, the two outputs neurons generate 0.015912196 (vs 0.01 target) and 0.984065734 (vs 0.99 target).

NN Example 2: Backpropagation Step by Step

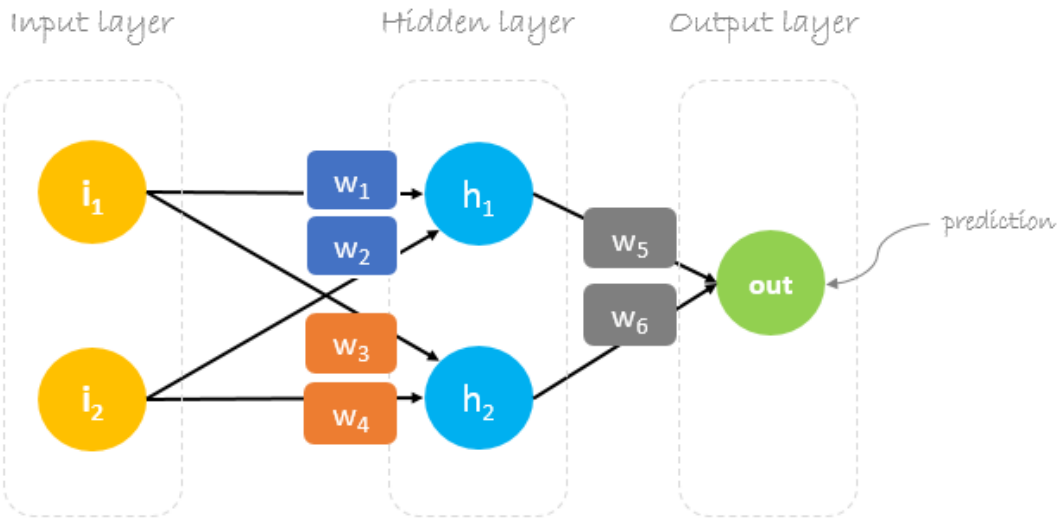


If you are building your own neural network, you will definitely need to understand how to train it. Backpropagation is a commonly used technique for training neural network. There are many resources explaining the technique, but this post will explain backpropagation with concrete example in a very detailed colorful steps.

Overview

In this post, we will build a neural network with three layers:

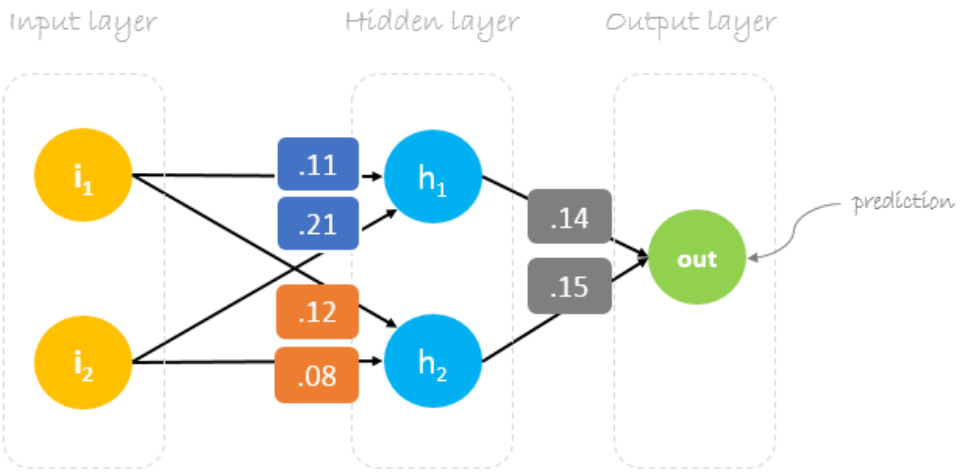
- **Input** layer with two inputs neurons
- One **hidden** layer with two neurons
- **Output** layer with a single neuron



Weights, weights, weights

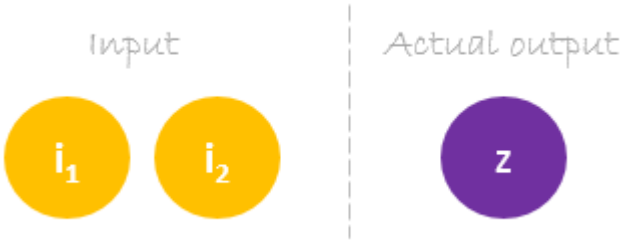
Neural network training is about finding weights that minimize prediction error. We usually start our training with a set of randomly generated weights.Then, backpropagation is used to update the weights in an attempt to correctly map arbitrary inputs to outputs.

Our initial weights will be as following: w1 = 0.11, w2 = 0.21, w3 = 0.12, w4 = 0.08, w5 = 0.14 and w6 = 0.15

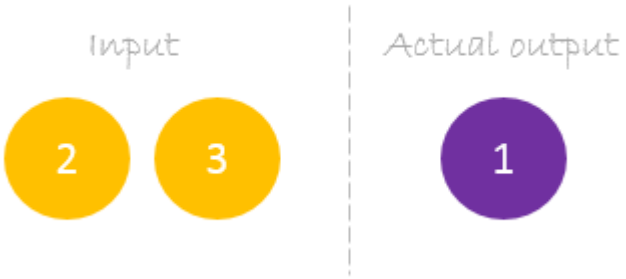


Dataset

Our dataset has one sample with two inputs and one output.

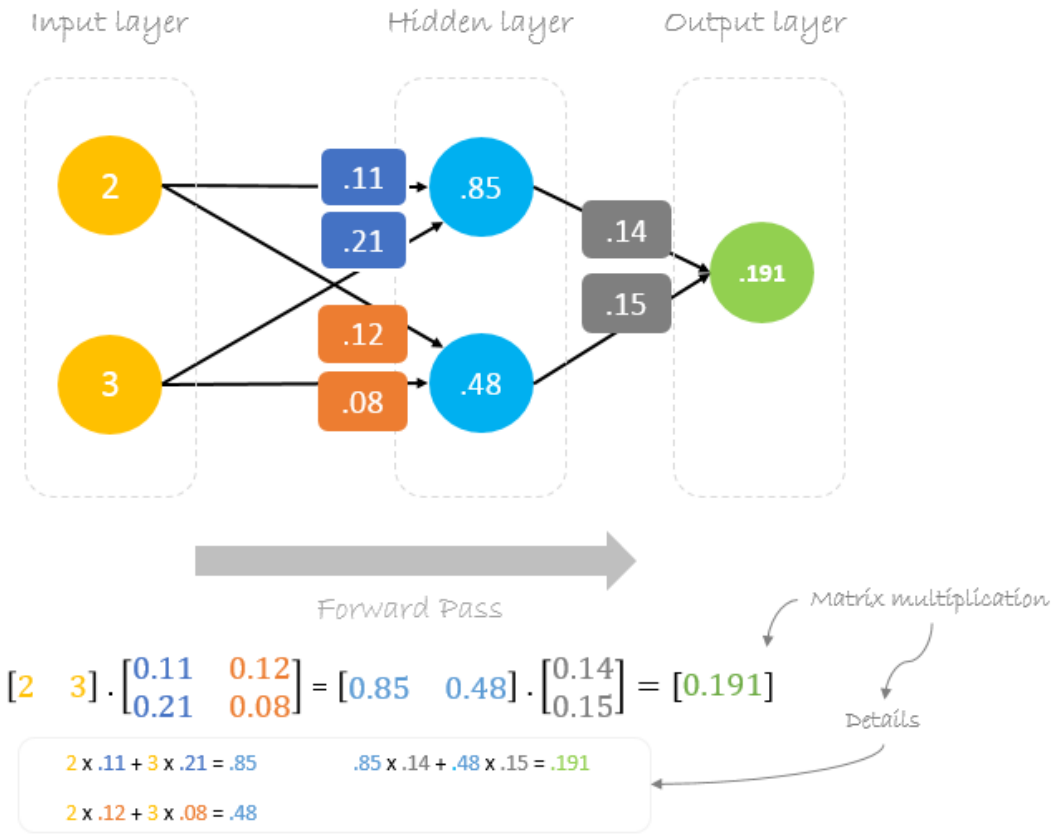


Our single sample is as following inputs=[2, 3] and output=[1].



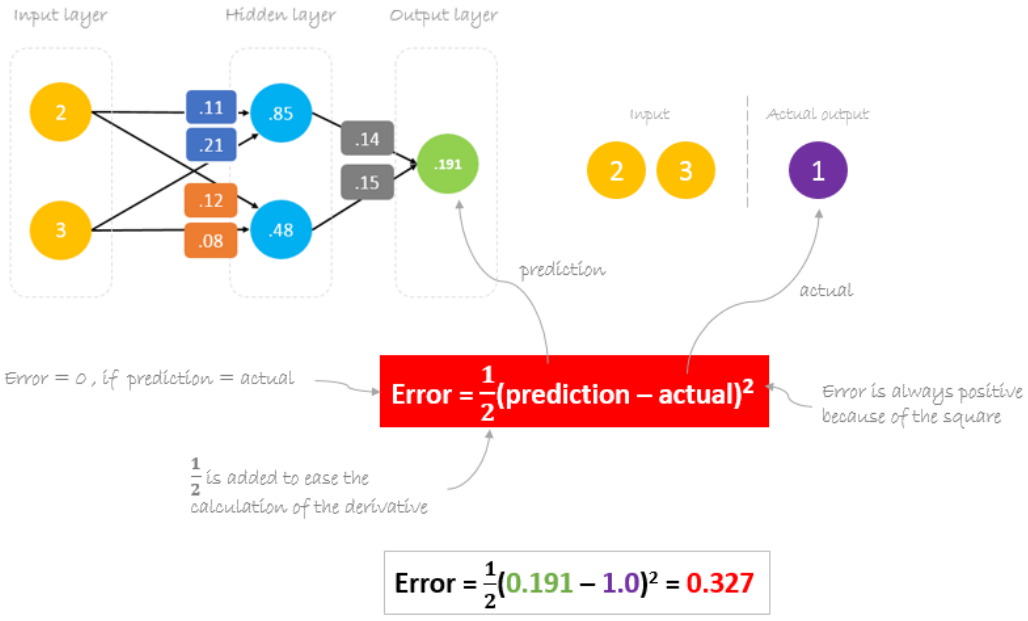
Forward Pass

We will use given weights and inputs to predict the output. Inputs are multiplied by weights; the results are then passed forward to next layer.



Calculating Error

Now, it's time to find out how our network performed by calculating the difference between the actual output and predicted one. It's clear that our network output, or **prediction**, is not even close to **actual output**. We can calculate the difference or the error as following.



Reducing Error

Our main goal of the training is to reduce the **error** or the difference between **prediction** and **actual output**. Since **actual output** is constant, "not changing", the only way to reduce the error is to change **prediction** value. The question now is, how to change **prediction** value?

By decomposing **prediction** into its basic elements we can find that **weights** are the variable elements affecting **prediction** value. In other words, in order to change **prediction** value, we need to change **weights** values.

$$\text{prediction} = \text{out}$$

$$\text{prediction} = (h_1)\, w_5 + (h_2)\, w_6$$

$$h_1 = i_1 w_1 + i_2 w_2$$

$$h_2 = i_1 w_3 + i_2 w_4$$

$$\text{prediction} = (i_1 w_1 + i_2 w_2)\, w_5 + (i_1 w_3 + i_2 w_4)\, w_6$$

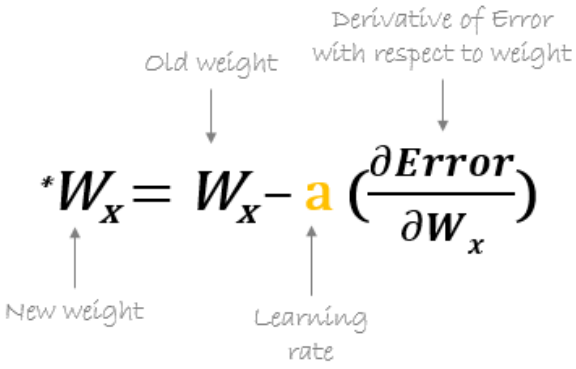
to change ***prediction*** value,
we need to change ***weights***

The question now is **how to change\update the weights value so that the error is reduced?** The answer is **Backpropagation!**

Backpropagation

Backpropagation, short for “backward propagation of errors”, is a mechanism used to update the **weights** using gradient descent. It calculates the gradient of the error function with respect to the neural network’s weights. The calculation proceeds backwards through the network.

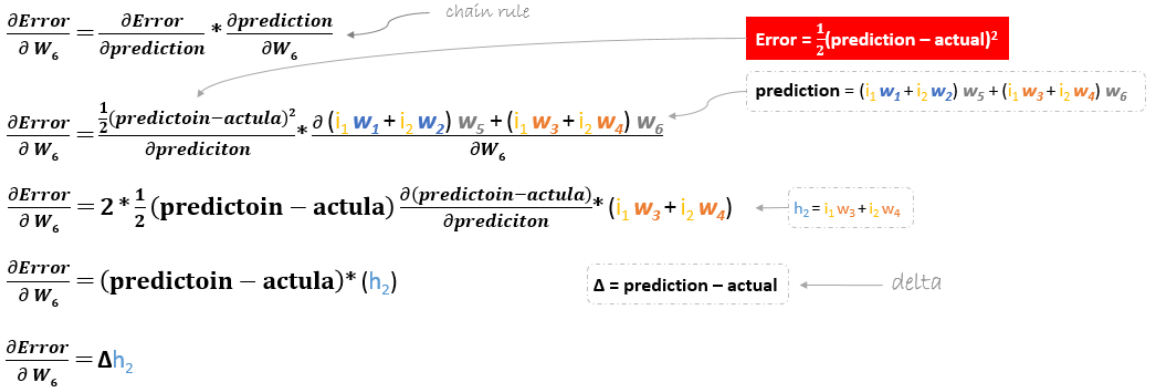
Gradient descent is an iterative optimization algorithm for finding the minimum of a function; in our case we want to minimize th error function. To find a local minimum of a function using gradient descent, one takes steps proportional to the negative of the gradient of the function at the current point.



For example, to update w6, we take the current w6 and subtract the partial derivative of **error** function with respect to w6. Optionally, we multiply the derivative of the **error** function by a selected number to make sure that the new updated **weight** is minimizing the error function; this number is called ***learning rate***.

$$^*W_6 = W_6 - a\left(\frac{\partial Error}{\partial W_6}\right)$$

The derivation of the error function is evaluated by applying the chain rule as following



So to update w6 we can apply the following formula

$$^{*}W_6 = W_6 - a\,\Delta h_2$$

Similarly, we can derive the update formula for w5 and any other weights existing between the output and the hidden layer.

$$^{*}W_5 = W_5 - a\,\Delta h_1$$

However, when moving backward to update w1, w2, w3 and w4 existing between input and hidden layer, the partial derivative for the error function with respect to w1, for example, will be as following.

$$\frac{\partial Error}{\partial W_1} = \frac{\partial Error}{\partial prediction} * \frac{\partial prediction}{\partial h_1} * \frac{\partial h_1}{\partial W_1}$$ ← chain rule

Error = $\frac{1}{2}$(prediction – actual)^2

prediction = $(h_1)\,w_5 + (h_2)\,w_6$

$h_1 = i_1 w_1 + i_2 w_2$

$$\frac{\partial Error}{\partial W_1} = \frac{\partial \frac{1}{2}(predictoin - actula)^2}{\partial prediciton} * \frac{\partial\,(h_1)\,w_5 + (h_2)\,w_6}{\partial h_1} * \frac{\partial\, i_1 w_1 + i_2 w_2}{\partial w_1}$$

$$\frac{\partial Error}{\partial W_1} = 2 * \frac{1}{2}(\mathbf{predictoin - actula}) \frac{\partial (predictoin - actula)}{\partial prediciton} * (w_5) * (i_1)$$

$$\frac{\partial Error}{\partial W_1} = (\mathbf{predictoin - actula}) * (w_5 i_1)$$

Δ = prediction – actual ← delta

$$\frac{\partial Error}{\partial W_1} = \Delta\, w_5 i_1$$

We can find the update formula for the remaining weights w2, w3 and w4 in the same way. In summary, the update formulas for all weights will be as following:

updated weights

$$^{*}w_6 = w_6 - a\,(h_2 \,.\, \Delta)$$
$$^{*}w_5 = w_5 - a\,(h_1 \,.\, \Delta)$$
$$^{*}w_4 = w_4 - a\,(i_2 \,.\, \Delta w_6)$$
$$^{*}w_3 = w_3 - a\,(i_1 \,.\, \Delta w_6)$$
$$^{*}w_2 = w_2 - a\,(i_2 \,.\, \Delta w_5)$$
$$^{*}w_1 = w_1 - a\,(i_1 \,.\, \Delta w_5)$$

We can rewrite the update formulas in matrices as following

$$\begin{bmatrix} w_5 \\ w_6 \end{bmatrix} = \begin{bmatrix} w_5 \\ w_6 \end{bmatrix} - a\,\Delta \begin{bmatrix} h_1 \\ h_2 \end{bmatrix} = \begin{bmatrix} w_5 \\ w_6 \end{bmatrix} - \begin{bmatrix} a h_1 \Delta \\ a h_2 \Delta \end{bmatrix}$$

$$\begin{bmatrix} w_1 & w_3 \\ w_2 & w_4 \end{bmatrix} = \begin{bmatrix} w_1 & w_3 \\ w_2 & w_4 \end{bmatrix} - a\,\Delta \begin{bmatrix} i_1 \\ i_2 \end{bmatrix} . \begin{bmatrix} w_5 & w_6 \end{bmatrix} = \begin{bmatrix} w_1 & w_3 \\ w_2 & w_4 \end{bmatrix} - \begin{bmatrix} a\, i_1 \Delta w_5 & a\, i_1 \Delta w_6 \\ a\, i_2 \Delta w_5 & a\, i_2 \Delta w_6 \end{bmatrix}$$

Backward Pass

Using derived formulas we can find the new **weights**.

Learning rate: is a hyperparameter which means that we need to manually guess its value.

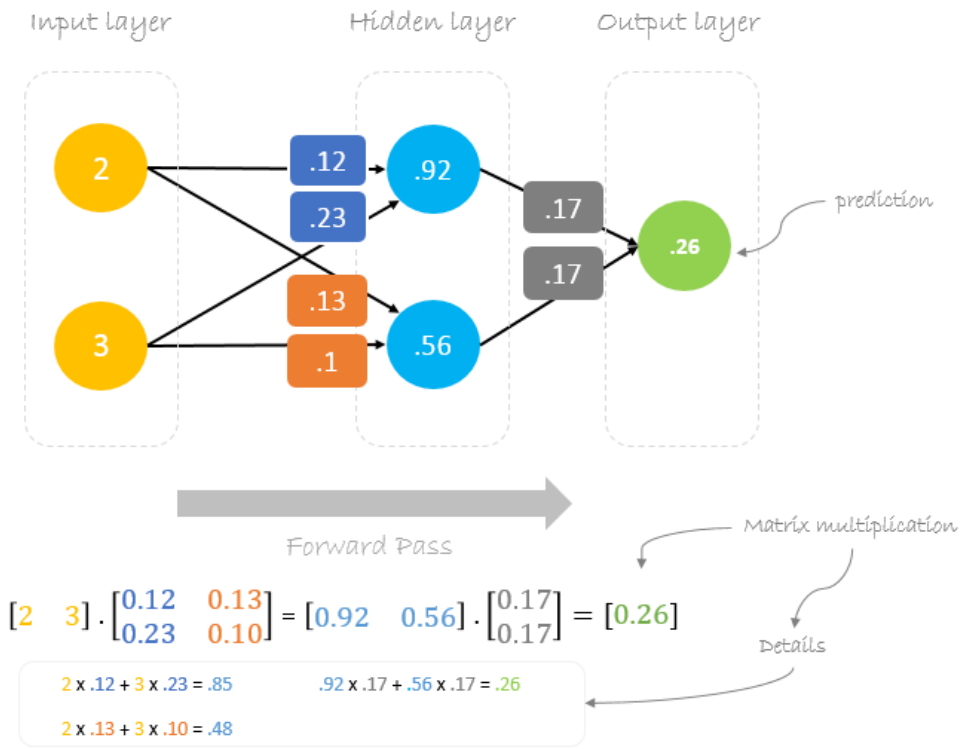
$\Delta = 0.191 - 1 = -0.809$ ← **Delta** = prediction - actual

$a = 0.05$ ← **Learning rate,** we smartly guess this number

$$\begin{bmatrix} w_5 \\ w_6 \end{bmatrix} = \begin{bmatrix} 0.14 \\ 0.15 \end{bmatrix} - 0.05(-0.809)\begin{bmatrix} 0.85 \\ 0.48 \end{bmatrix} = \begin{bmatrix} 0.14 \\ 0.15 \end{bmatrix} - \begin{bmatrix} -0.034 \\ -0.019 \end{bmatrix} = \begin{bmatrix} 0.17 \\ 0.17 \end{bmatrix}$$

$$\begin{bmatrix} w_1 & w_3 \\ w_2 & w_4 \end{bmatrix} = \begin{bmatrix} .11 & .12 \\ .21 & .08 \end{bmatrix} - 0.05(-0.809)\begin{bmatrix} 2 \\ 3 \end{bmatrix}.[0.14 \quad 0.15] = \begin{bmatrix} .11 & .12 \\ .21 & .08 \end{bmatrix} - \begin{bmatrix} -0.011 & -0.012 \\ -0.017 & -0.018 \end{bmatrix} = \begin{bmatrix} .12 & .13 \\ .23 & .10 \end{bmatrix}$$

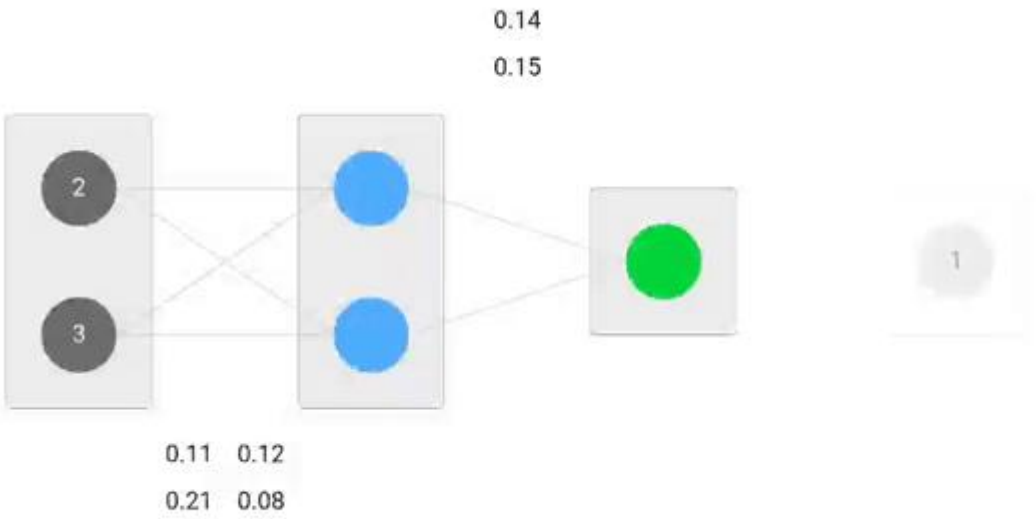
Now, using the new **weights** we will repeat the forward passed



$$[2 \quad 3].\begin{bmatrix} 0.12 & 0.13 \\ 0.23 & 0.10 \end{bmatrix} = [0.92 \quad 0.56].\begin{bmatrix} 0.17 \\ 0.17 \end{bmatrix} = [0.26]$$

We can notice that the **prediction** 0.26 is a little bit closer to **actual output** than the previously predicted one 0.191. We can repeat the same process of backward and forward pass until **error** is close or equal to zero.

Backpropagation Visualization



6. Yararlanılan Kaynaklar

1. Machine Learning (ML) Algorithms For Beginners with Code Examples in Python. https://medium.com/towards-artificial-intelligence/machine-learning-algorithms-for-beginners-with-python-code-examples-ml-19c6afd60daa
2. https://www.w3schools.com/python/python_ml_getting_started.asp
3. https://www.tutorialspoint.com/machine_learning_with_python/knn_algorithm_finding_nearest_neighbors.htm
4. https://brilliant.org/wiki/feature-vector/
5. https://stanford.edu/~shervine/l/tr/teaching/cs-229/
6. https://stanford.edu/~shervine/l/tr/teaching/cs-221/
7. https://stanford.edu/~shervine/l/tr/teaching/cs-230/