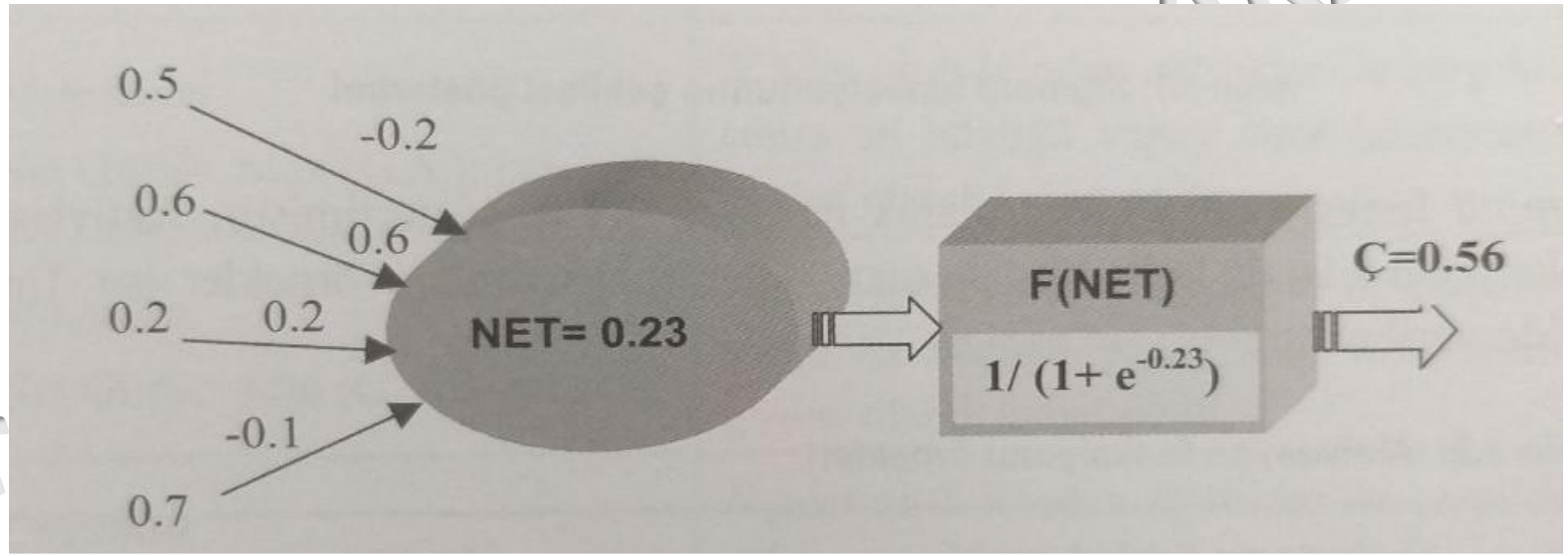
Yapay Sinir Ağları

Dr. Öğr. Üyesi Fatma AKALIN

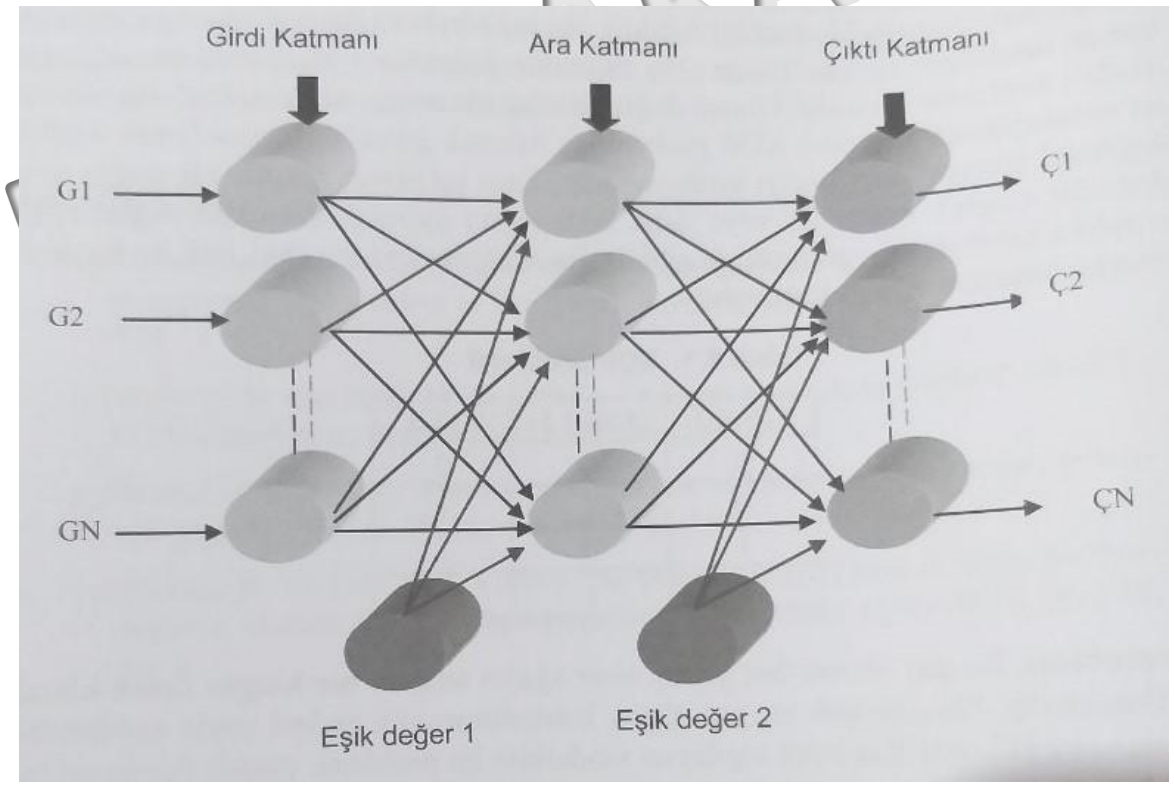
TKA Ağının Öğrenme Kuralını Somutlaştıralım

Bir önceki slaytta tek katmanlı algılayıcıların çalışma prensibini irdelemiştik.



ÇKA(Çok Katmanlı Algılayıcı) Modelinin Yapısı

ÇKA ağlarının yapısı yanda sunulmuştur. Şekilde de görüldüğü gibi ÇKA ileriye doğru bağlantılı ve 3 katmandan oluşan bir ağdır. Bu katmanlar; Girdi Katmanı, Ara Katmanlar ve Çıktı Katmanı'dır.



Girdi Katmanı: Dış dünyadan gelen verileri alarak **ara katmana gönderir**. Bu katmanda **BİLGİ İŞLEMİ OLMAZ**. Gelen her bilgi geldiği gibi bir sonraki katmana gider. Birden fazla girdi gelebilir. **Her proses elemanının sadece bir tane girdisi ve bir tane çıktısı vardır**. **Bu çıktı** bir sonraki katmanda bulunan bütün **proses elemanlarına gönderilir**. Yani girdi katmanındaki her proses elemanı bir sonraki katmanda bulunan proses elemanlarının **hepsine bağlıdır**.

Ara Katmanlar: Ara katmanlar girdi katmanından gelen bilgileri **İŞLEYEREK** bir sonraki katmana gönderir. Bir ÇKA ağında **birden fazla ara katman ve her katmanda birden fazla proses elemanı** olabilir. Ara katmandaki **her proses elemanı bir sonraki katmandaki bütün proses elemanlarına** bağlıdır.

Çıktı Katmanı: Girdi katmanından gelen girdilere karşılık **ARA KATMANDAN GELEN BİLGİLERİ İŞLEYEREK** ağın ürettiği çıktıları belirlemek suretiyle dış dünyaya gönderir. Bir çıktı katmanında birden fazla proses elemanı olabilir. Her proses elemanı bir önceki katmanda bulunan bütün proses elemanlarına bağlıdır. Her proses elemanının sadece bir tane çıktısı vardır

ÇKA ağı **öğretmenli öğrenme stratejisini** kullanır. Ağa hem örnekler hem de örneklerden elde edilmesi gereken çıktılar verilmektedir. Ağ kendisine gösterilen örneklerden **genellemeler** yaparak **problem uzayını temsil eden bir çözüm uzayı** üretmektedir. Daha sonra gösterilen **benzer örnekler** için bu **çözüm uzayı** kullanılarak sonuçlar ve çözümler **üretebilmektedir**

Yapay sinir ağlarına ve proses elemanlarının yapısına genel bir bakış yaptıktan sonra günümüzde geliştirilen modellere temel olması nedeni ile ilk yapay sinir ağı modelleri incelenecektir.

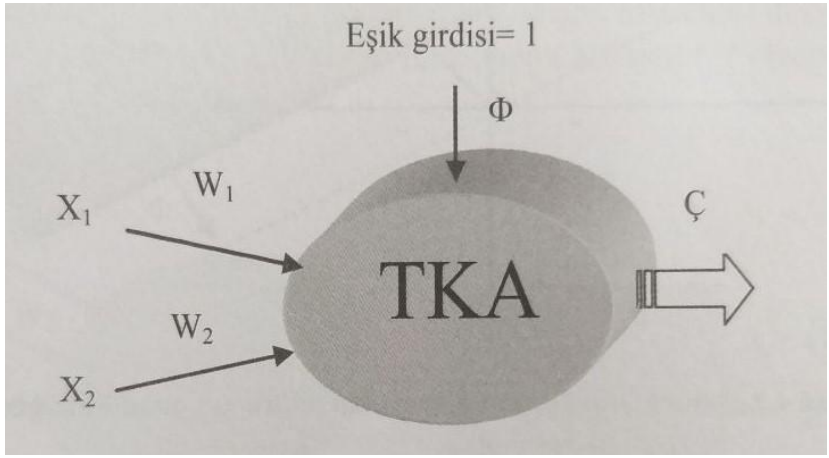
Tek katmanlı Algılayıcılar

Tek katmanlı yapay sinir ağları **sadece girdi ve çıktı katmanlarından** oluşur. Her ağın **bir veya daha fazla girdisi ve çıktısı vardır**. Çıktı üniteleri bütün girdi ünitelerine bağlanmaktadır ve her bağlantının bir ağırlığı vardır. Aşağıdaki şekil en basit şekli ile tek katmanlı bir ağa örnektir.

Bu ağlarda proses elemanlarının değerlerini ve ağın çıktısının 0 olmasını önleyen eşik değeri vardır. Matematiksel denklemi aşağıda görüldüğü gibi formulüze edilebilir.

$$Ç=f(\sum_{i=1}^{m} w_i x_i + \varphi)$$

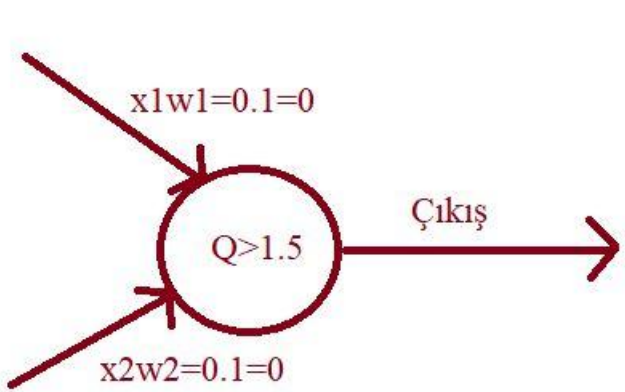
$$f(g) = \begin{cases} 1 \; eğer \; Ç > 0 \; ise \\ -1 \; aksi \; taktirde \end{cases}$$

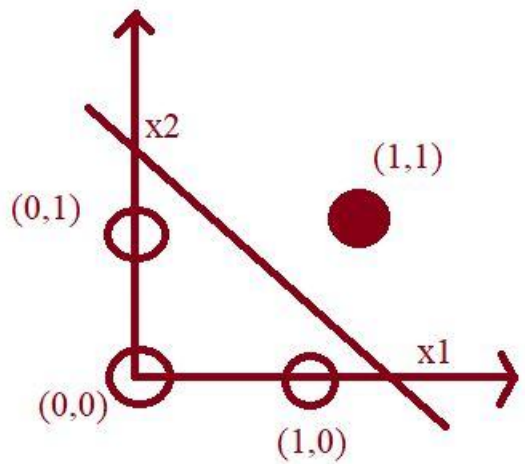


Tek katmanlı Algılayıcılar ile AND Mantık Kapısı Probleminin Çözümü

VE (AND) MANTIK KAPISI PROBLEMİ

Giriş 1	Giriş 2	Çıkış
0	0	0
0	1	0
1	0	0
1	1	1

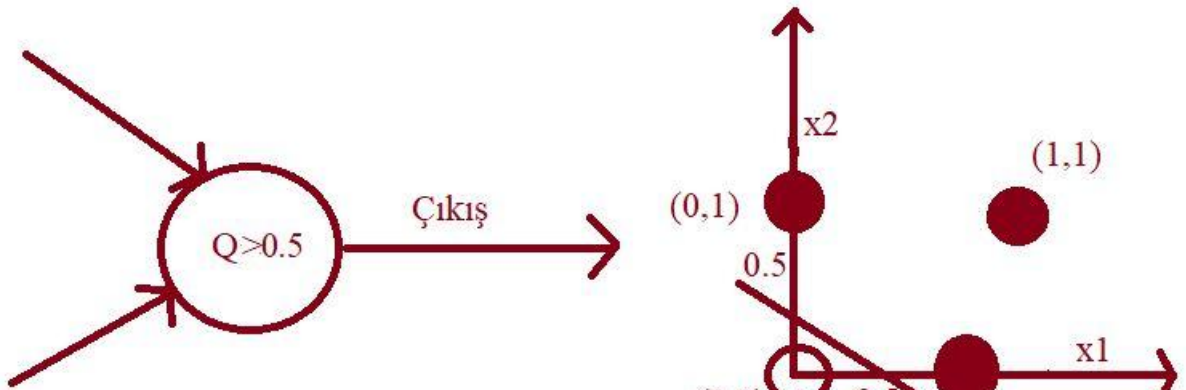




Tek katmanlı Algılayıcılar ile OR Mantık Kapısı Probleminin Çözümü

VEYA (OR) MANTIK KAPISI PROBLEMİ

Giriş 1	Giriş 2	Çıkış
0	0	0
0	1	1
1	0	1
1	1	1



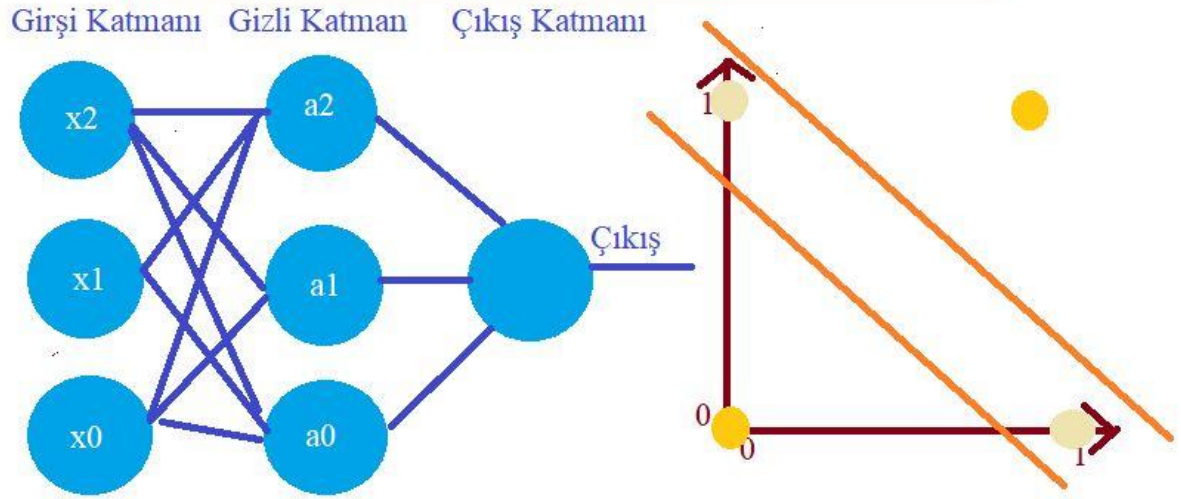
Çok Katmanlı Algılayıcılar

Bir önceki bölümde anlatılan yapay sinir ağlarının ilk modeli olan **tek katmanlı yapay sinir ağı modeli ile doğrusal olmayan ilişkiler öğrenilememektedir**. Bu sorunu çözmek için çok katmanlı algılayıcılar geliştirilmiştir. Şimdi çok katmanları algılayıcılar konusunu ayrıntılı olarak irdeleyeceğiz.

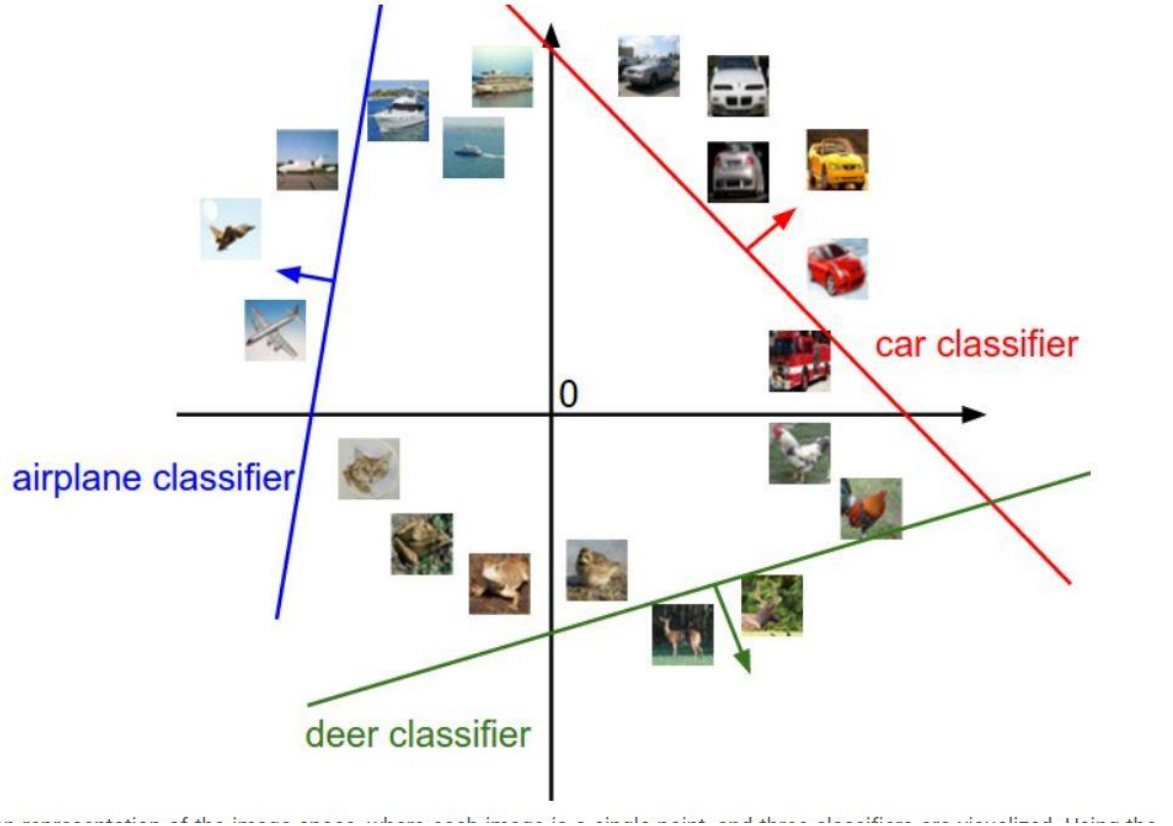
Çok katmanlı Algılayıcılar ile XOR Mantık Kapısı Probleminin Çözümü

XOR MANTIK KAPISI PROBLEMİ

Giriş 1	Giriş 2	Çıkış
0	0	0
0	1	1
1	0	1
1	1	0



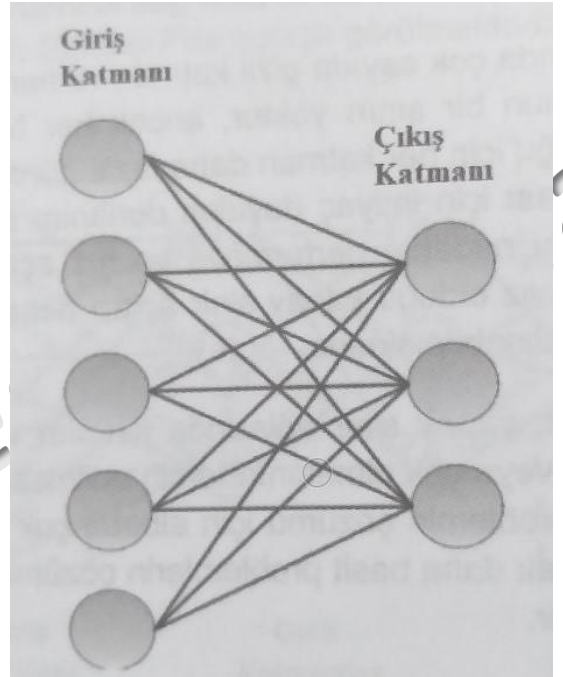
Yandaki şekilde 3 farklı yerden yapılan ayırma işlemi problemi çok katmanlı algılayıcıların kullanılmak suretiyle çözülmesine işaret etmektedir. (XOR problemi gibi)



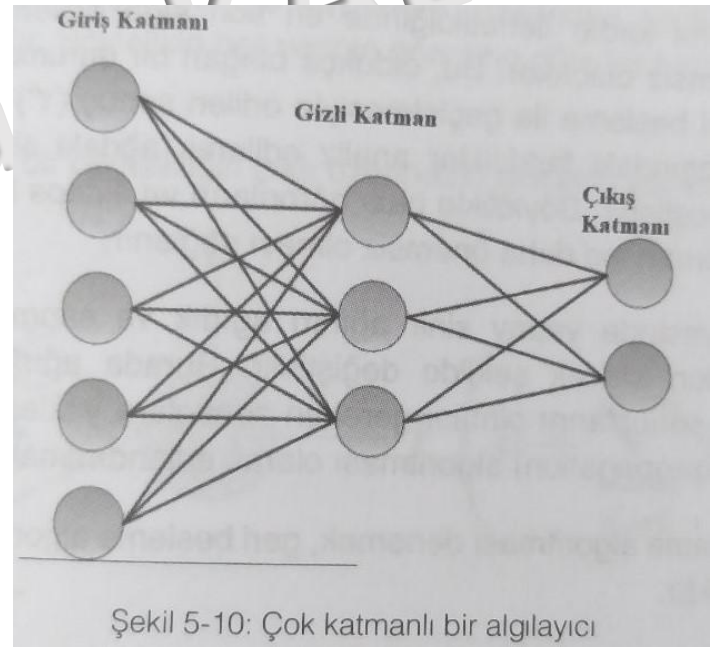
https://cs231n.github.io/linear-classify/

Özetle bir gizli katmanın bulunmadığı yapay sinir ağları **tek katmanla algılayıcı** olarak adlandırılmaktadır. Fakat günümüzde tek katmanlı modelden daha karmaşık yapay sinir ağları kullanılmaktadır.

Bir veya birden fazla gizli katmanı bulunan yapay sinir ağları ise çok katmanlı algılayıcı olarak adlandırılmaktadır. Tek katmanlı algılayıcı ve çok katmanlı algılayıcıya ilişkin tasvirler yanda gösterilmektedir.



Şekil 5-9: Tek katmanlı bir algılayıcı



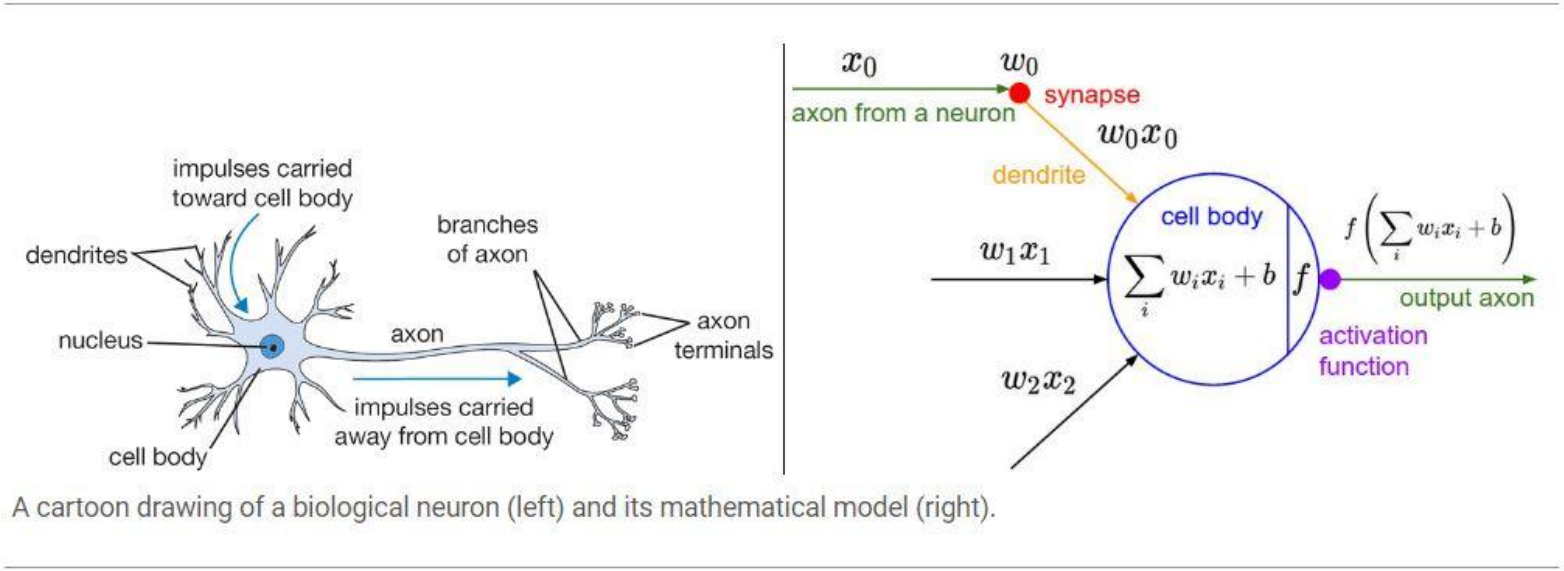
Şekil 5-10: Çok katmanlı bir algılayıcı

Çok katmanlı algılayıcılar **kompleks sistemler için** tek bir gizli katmanın yeterli gelmediği durumlarda kullanılan yapay sinir ağı modelidir. Çok **katmanlı algılayıcılarda bir veya birden fazla gizli katman bulunmaktadır**. Böyle bir mimaride **her katmanda birden fazla nöron yer alır** ve bu nöronlar aldıkları **sinyali işleyip kendisinden sonraki katmanda yer alan diğer nöronlara iletir.**

Çok katmanlı algılayıcıların çalışma prensibi, tek katmanlı algılayıcılara benzer yapıdadır. Ağda **ileri besleme** algoritması aynı şekilde **soldan sağa doğru** işletilir. Tek katmanlı mimariden farklı olarak **gizli katmanlarda üretilen sonuçlar ayrı ayrı aktivasyon fonksiyonlarından geçirilerek bir çıktı** değeri elde edilir ve bu değer bir sonraki katmanın nöronlarına aktarılır. En son gizli katman işlediği sinyali doğrudan çıkış katmanına aktarır. Böylece ağda ileri besleme yoluyla bir çıktı elde edilmiş olur.

Bir katmandaki nöron bir önceki ve sonraki katmanların nöronlarına sinapslar ile bağlantılıdır. Sinyaller bu sinapslar üzerinden iletilir.

Biyolojik Sinir Sistemi	Yapay Sinir Sistemi
Dendrit	Toplama Fonksiyonu
Akson	Çıktı
Sinaps	Ağırlıklar
Hücre Gövdesi	Aktivasyon Fonksiyonu



A cartoon drawing of a biological neuron (left) and its mathematical model (right).

https://cs231n.github.io/neural-networks-1/

Sinyallerin iletildiği yapay sinir ağında **her sinapsın ağırlığı farklıdır** ve sinyal, iletimde kullanılan **sinapsın ağırlığı ile çarpılır**. Sinaps ağırlıklarını bir nevi katsayı olarak düşünebiliriz. Dolayısıyla **bir nörondan gelen sinyalin önemi başka bir nörona göre daha fazla olabilir**. Bu önemi belirten şey ise **ilgili nöronun sinapsinin ağırlığıdır**.

Bir yapay sinir ağı ilk oluşturulduğunda hangi verinin veya hangi nörondan gelen verinin daha önemli olduğu bilinmemektedir. Bu nedenle ***ilk başta sinapslerin ağırlıkları yani weight değerleri ve nöronların sapma yani bias değerleri rastgele olarak atanır.***

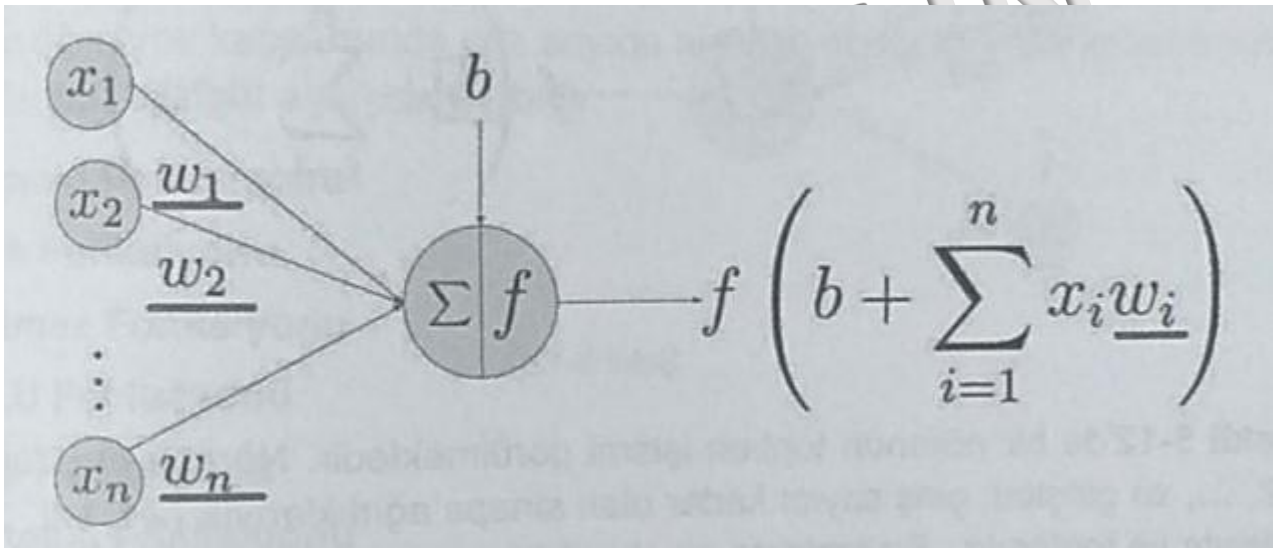
Rastgele oluşturulmuş bu yapay sinir ağında **ileri besleme algoritması** çıkış katmanına kadar ilerletildiğinde **en son elde edilen sonuç çok büyük ihtimalle anlamsız olacaktır**. Bu oldukça olağan bir durumdur. Çünkü giriş verilerinin ağdan ileri besleme ile geçirilmesi ile edilen sonuç ve olması gereken sonuç arasında farklılıklar analiz edilerek ağdaki sinaps ve nöronların ağırlıkları güncellenir. Böylelikle bazı **nöronların ve sinaps bağlantılarının** daha **önemli** bazılarının ise daha **önemsiz** olması sağlanır. Bu işlem sayesinde yapay sinir ağının ağır**lık ve sapma değerleri olması gerekene yakın olacak şekilde değiştirilir.** Burada ağırlıkları güncelleyerek **ileri besleme sonuçlarının olması gereken sonuçları yaklaştıran algoritma geri yayılım algoritması olarak adlandırılmaktadır.**

Yani ileri besleme algoritması; denemek, geri besleme algoritması ise hatalardan ders çıkarmaktadır.

YAPAY SİNİR AĞI İLE İLGİLİ KAVRAMLARI İNCELEYELİM

Ağırlık (Weight) Değerleri

Ağırlık, **girdideki bir değişikliğin çıktı üzerindeki şiddetini** etkilemektedir. Yandaki şekilde bir nöronun toplam işlemi görülmektedir. Nörona uygulanan x1,x2, .. xn **girişleri**; giriş sayısı kadar olan **sinaps ağırlıkları** ile (w1,w2…wn) çarpılır ve toplanır. Bu toplama ek olarak **bias**(sapma) değeri de eklenir.



Tüm giriş verilerinin, iletildiği sinaps üzerindeki **ağırlık ile çarpılması aslında bize verilerin önemlerinin farklı olduğunu** göstermektedir. Daha sonra geri yayıllm algoritması vasıtasıyla bu ağırlıklar **güncellenerek** önemli olan verileri ileten sinaps ağırlıklarının artması önemsiz olan ağırlıkların ise azalması sağlanır

UYARI

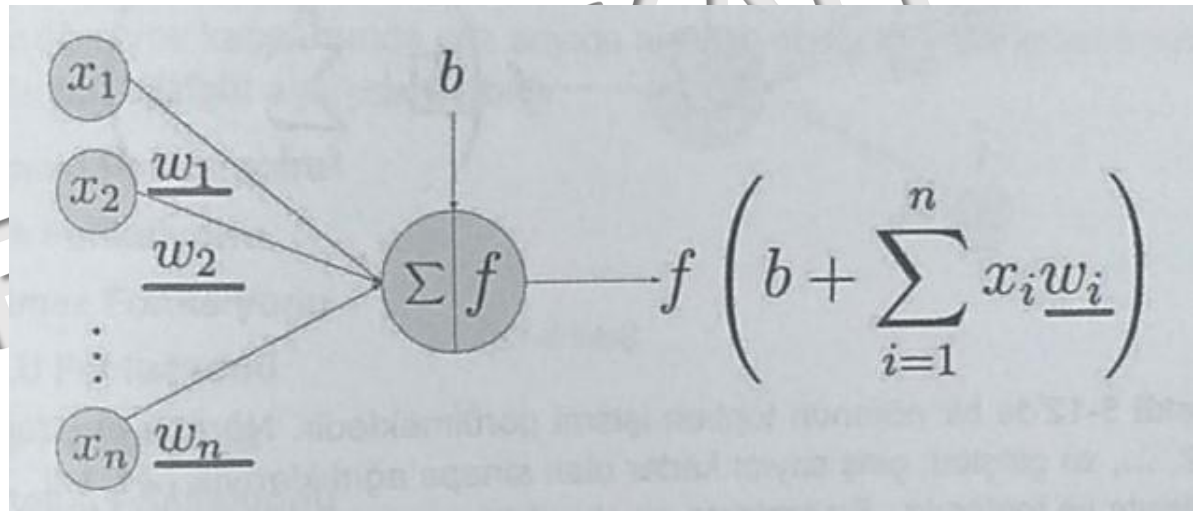
Bir yapay sinir ağı ilk oluşturulduğunda hangi verinin veya hangi nörondan gelen verinin daha önemli olduğu bilinmemektedir. Bu nedenle ilk başta sinapslarin ağırlıkları yani **weight** değerleri ve nöronları sapma yani **bias** değerleri **rastgele** olarak atanır

Sapma(Bias) Değeri

Yapay sinir ağı içerisinde bir **nöron** aldığı iletileri **sinaps ağırlıkları** ile çarpıp topladıktan sonra **kendi ağırlık değerini** de bu toplama ekler. Böylelikle her sinapsın bir değer ağırlığı olduğu gibi, her nöronunda aslında bir değer ağırlığına sahip olması sağlanır.

Sapma değeri, bir nöronun çıkışa etkisini ve yapay sinir ağı içerisindeki önemini etkilemektedir.

Yandaki şekilde görüldüğü gibi mevcut toplama ek olarak **bias** değeri de **eklenmiştir**. İletilerin ağırlıklı toplama işleminden sonra bir s**apma değerinin eklenmesi sonucunda** ileti şiddeti sinaps ağırlığından bağımsız da değişebileceğini göstermektedir.



UYARI

Bir yapay sinir ağı ilk oluşturulduğunda **hangi nöronun daha önemli veya önemsiz olduğu bilinmemektedir.** Bu nedenle ilk başta nöronların ağırlıkları yani bias değerleri **rastgele** olarak atanır.

Aktivasyon Fonksiyonları

Aktivasyon fonksiyonları nöronun içerisinde **wx + b** yani sinyal x ağırlık + sapma sonucu elde edilen **iç toplam değerini ağda iletebilecek şekilde normalize eden** fonksiyonlardır.

İç toplam sonrası elde edilen sonuç, yapay sinir ağında **yayılmaya uygun olmayacak** bir aralıkta olabilir. Bu nedenle **veriyi bir aralığı sıkıştırmak** amacıyla aktivasyon fonksiyonları kullanılmaktadır.

Aktivasyon fonksiyonları **oluşturulacak olan sinir ağı modeline** ve **çözülecek probleme** göre farklılık gösterebilmektedir. Bu yüzden model oluştururken birden fazla aktivasyon fonksiyonunu **deneyimleyerek** çıktılar üzerinde yorum yapmak en optimum kararı vermek açısından önem arz etmektedir.

Ayrıca aktivasyon fonksiyonları ağdaki **tüm nöronların ürettiği sinyallerin belirli standartlarda olmasını sağlar**. Böylelikle sinyaller arasında büyük bir fark oluşmaz.

Derin öğrenme kapsamında **çok sayıda aktivasyon fonksiyonu** kullanılmaktadır. Bunlardan bazıları;

1-Sigmoid Fonksiyonu

2-Tanh Fonksiyonu

3-Softmax Fonksiyonu

4-ReLU Fonksiyonu

5-Step Fonksiyonu

6-Softplus Fonksiyonu

7-ELU Fonksiyonu

8-Swish Fonksiyonu

Günümüzde yapay sinir ağlarında en çok tercih edilen aktivasyon fonksiyonları **Sigmoid, Softmax, Tanh ve ReLU** fonksiyonudur.

Dersin akışında bahsedildiği üzere ağırlıklar ile çarpılıp toplama ve sapma değeri ekleme işlemlerinin ardından elde edilen nöron içi toplam değeri, bir sonraki nörona aktarılmak için bir aktivasyon fonksiyonuna tabi tutulur. Ardından **elde edilen çıktı değeri**, bir **sonraki katmandaki nöronlara** veya **doğrudan çıkış katmanına** aktarılabilir.

Sigmoid, Softmax, Tanh ve ReLU aktivasyon fonksiyonlarını inceleyelim.

Sigmoid Aktivasyon Fonksiyonu

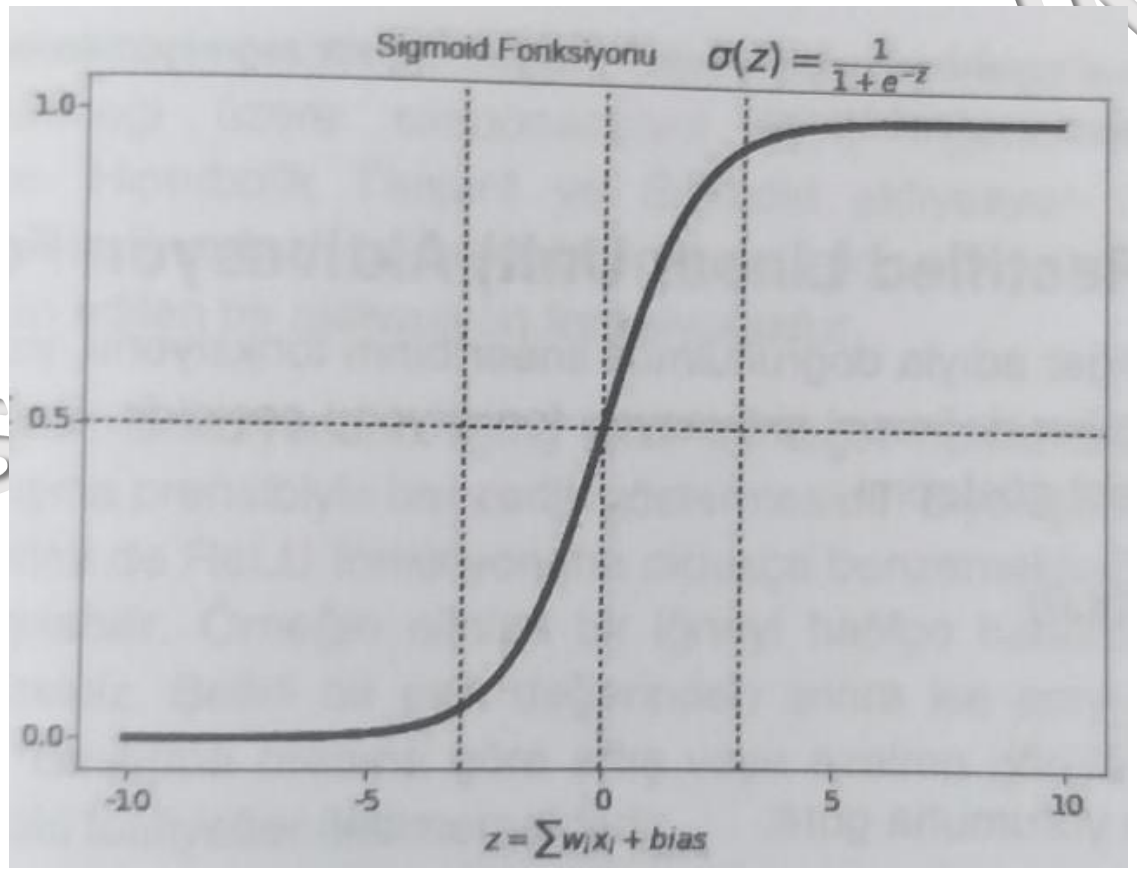
Sigmoid fonksiyonu diğer bir ifade ile sigmoidal eğri, başlıca en yaygın kullanılan aktivasyon fonksiyonlarındandır. Sigmoid; giriş verisini **0 ve 1** aralığına sıkıştıran matematiksel bir işlemdir. Sinir ağlarında **bir nöronun çıkış sinyaline normalizasyon sağlaması** amacıyla uygulanır. Böylelikle tüm iletiler **0 ile 1 önemi arasında sıkıştırılmış** olur. Geometrik olarak **«S»** şeklinde bir eğriye sahip olarak bilinen sigmoid fonksiyonu **gizli katmanlarda** ya da **çıkış katmanında** kullanılabilen doğrusal olmayan bir fonksiyondur.

Sigmoid fonksiyonu **ikili bir çıktının olasılıklarını tahmin etmek** için kullanılır. Örneğin iyi huylu ve kötü huylu kanser hücrelerini tespit etme, sahte ve doğal kullanıcı yorumlarını tespit etme vb. durumlarda kullanılabilir.

Sigmoid fonksiyonunun matematiksel formülü aşağıdaki gibidir.

$$P(x) = \frac{e^x}{1 + e^x} = \frac{1}{1 + e^{-x}}$$

Sigmoid fonksiyonunun grafiksel gösterimi yanda verilmiştir.



Aşağıdaki kod parçacığında sigmoid aktivasyon fonksiyonunun Python programlama dilinde kodlanması verilmiştir.

```
def aktivasyon_fonksiyonu(val):
    return 1.0 / (1.0 + exp(-val))
```

NOT Geri yayılım algoritması uygulanırken sigmoid fonksiyonunun türevi alınır. Sigmoid fonksiyonunun türevinin hesaplanması aşağıdaki **Şekil 5-15**'te görülmektedir. Geri yayılım algoritması ile ilgili ayrıntılara Geri Yayılım Algoritması bölümünde yer vereceğiz.

$$\sigma(x) = \frac{1}{1+e^{-x}} \quad (1)$$

$$\frac{d(\sigma(x))}{dx} = \frac{0*(1+e^{-x}) - (1)*(e^{-x}*(-1))}{(1+e^{-x})^2} \quad (2)$$

$$\frac{d(\sigma(x))}{dx} = \frac{(e^{-x})}{(1+e^{-x})^2} = \frac{1\text{-}1+(e^{-x})}{(1+e^{-x})^2} = \frac{1+e^{-x}}{(1+e^{-x})^2} - \frac{1}{(1+e^{-x})^2} \quad (3)$$

$$\frac{d(\sigma(x))}{dx} = \frac{1}{1+e^{-x}} * \left(1 - \frac{1}{1+e^{-x}}\right) = \sigma(x)(1-\sigma(x)) \quad (4)$$

Şekil 5-15: Ters Sigmoid fonksiyonunun işlem adımları

Aşağıdaki kod parçacığında ters sigmoid fonksiyonunun Python programlama dilinde kodlanması görülmektedir.

```
def aktivasyon_turevi(output):
    return output * (1.0 - output)
```

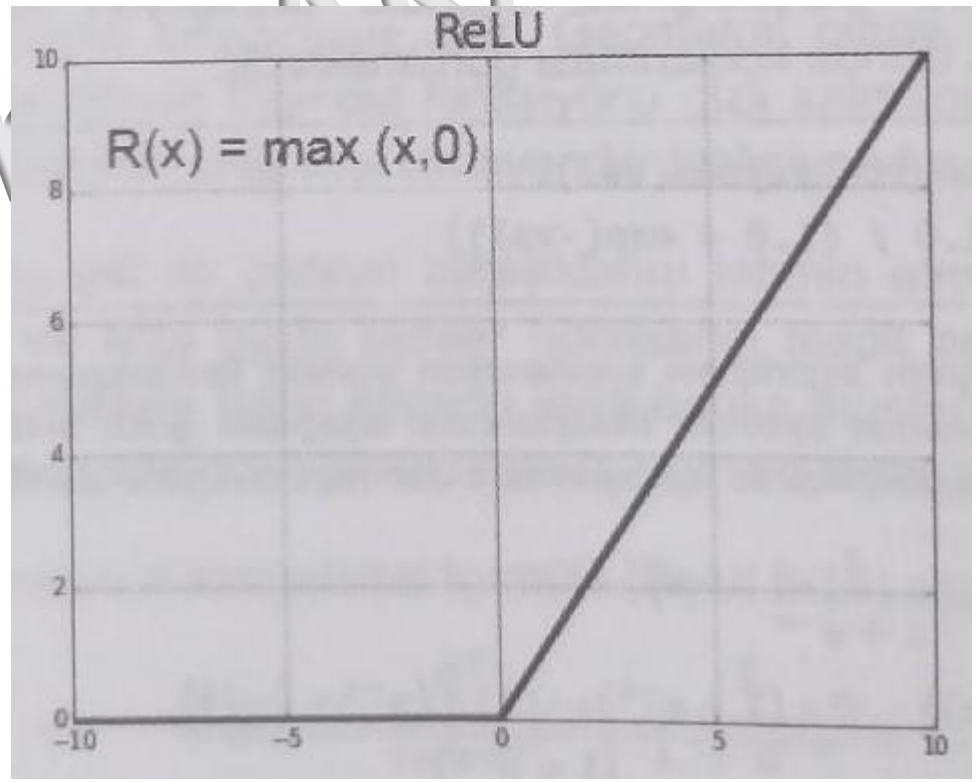
ReLU (Rectified Linear Unit) Aktivasyon Fonksiyonu

ReLU bir diğer adıyla doğrultulmuş lineer birim fonksiyonu, yaygın olarak tercih edilen bir diğer doğrusal aktivasyon fonksiyonu çeşididir. ReLU fonksiyonunun matematiksel gösterimi **f(x)=max(x,0)** şeklindedir.

Bu tanımın yorumuna göre;

Nöron çıkışı x, sıfırdan büyük ise sonuç x,

Nöron çıkışı x, sıfırdan küçük ise sonuç 0 olmaktadır.



ReLU genellikle evrişimli yapay sinir ağları uygulamalarında **gizli katmanlarda sıklıkla** tercih edilmektedir. ReLU fonksiyonunun diğer aktivasyon fonksiyonlarına göre **avantajı ise aynı anda nöronları aktif etmemesidir.** Yani negatif değer üreten bir nöron, ReLU fonksiyonunda aktif hale getirilemeyeceğinden daha hızlı çalışır.

ReLU, matematiksel olarak oldukça basit bir denklemdir. Formülünden de anlaşılabileceği üzere **eksponansiyel veya trigonometrik** bir ifade içermediğinden **Hiperbolik Tanjant ve Sigmoid aktivasyon fonksiyonlarına oranla** daha verimli ve hızlı çalışmaktadır. Bu yüzden çok katmanlı yapay sinir ağlarında tercih edilen bir aktivasyon fonksiyonudur.

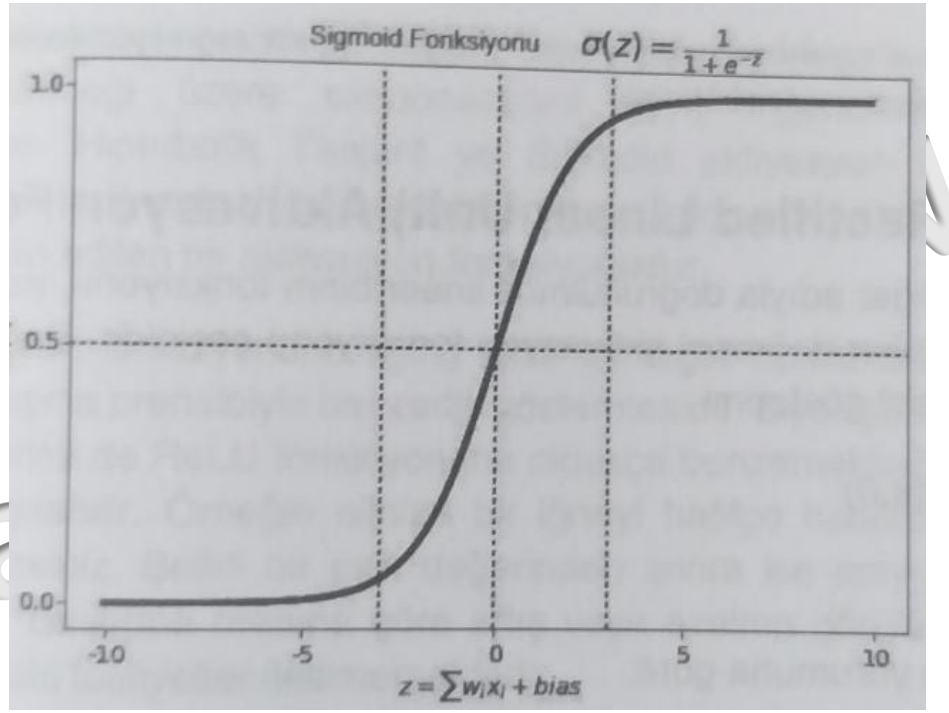
ReLU aktivasyon fonksiyonunu ilginç kılan diye diğer nokta **insan beynindeki nöronların** çalışma prensibi ile **benzerlik** göstermesidir. Biyolojik sinir sisteminde bir sinyalin iletimi de ReLU fonksiyonuna oldukça benzemektedir.

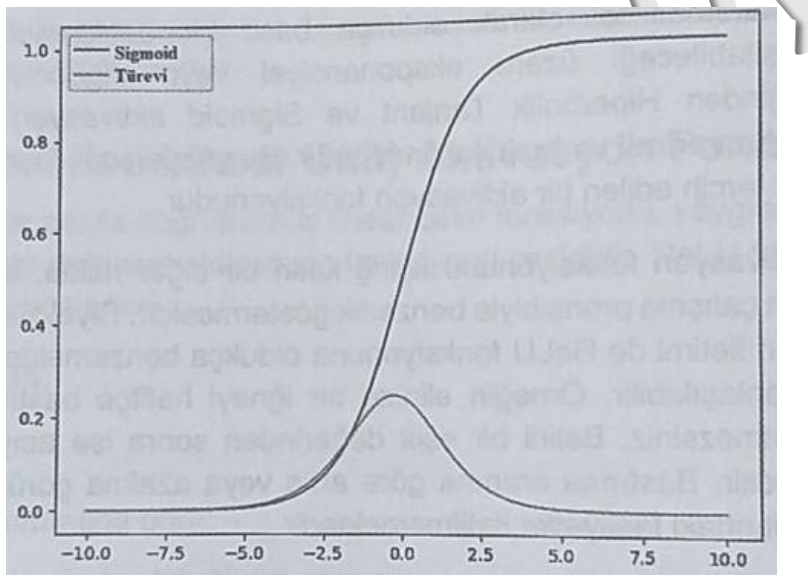
Bu basit bir deneyle anlaşılabilir. Örneğin elinize bir iğneyi hafifçe bastırdığınızda hiçbir şey hissetmezsiniz. **Belirli bir eşik değerinden sonra ise acıyı aynen olduğu gibi hissedersiniz**. Bastırma oranına göre artış veya azalma görülür.

Ancak belirli değerinin altındaki faaliyetler iletilmemektedir.

ReLU, insan sinir sisteminin bu özelliğinden **esinlenilerek** geliştirildiğinden yapay sinir ağları ekosisteminde **oldukça yaygın** bir kullanım alanına sahiptir.

Geometrik yorum ile ReLu ve Sigmoid aktivasyon fonksiyonlarını kıyaslayalım. Sağdaki şekilde sunulan sigmoid aktivasyon fonksiyonunun en sağ ve en sol uçlarına bakacak olursak, **X değerinin değişiminin Y değişkenine çok az etki ettiği** görülmektedir. Bu, belirli bir X değerinden sonra giriş değişkeninin değişiminin **Y yani çıktıya etkisinin neredeyse sıfır olması anlamına** gelmektedir.





Soldaki şekilde **Sigmoid aktivasyon fonksiyonun türevi** görülmektedir. Sigmoid fonksiyonunun tam değere yaklaştığı yani **X değerinin değişiminin Y değerine çok az etki ettiği noktalarda** sigmoid fonksiyonunun türevi 0'a yakınsamaktadır.

Bu durum **gradyanların ölmesi** (vanishing gradient) problemi olarak adlandırılır ve yapay sinir ağının geri yayılım ile öğrenmesine engel teşkil edebilecek bir durumdur.

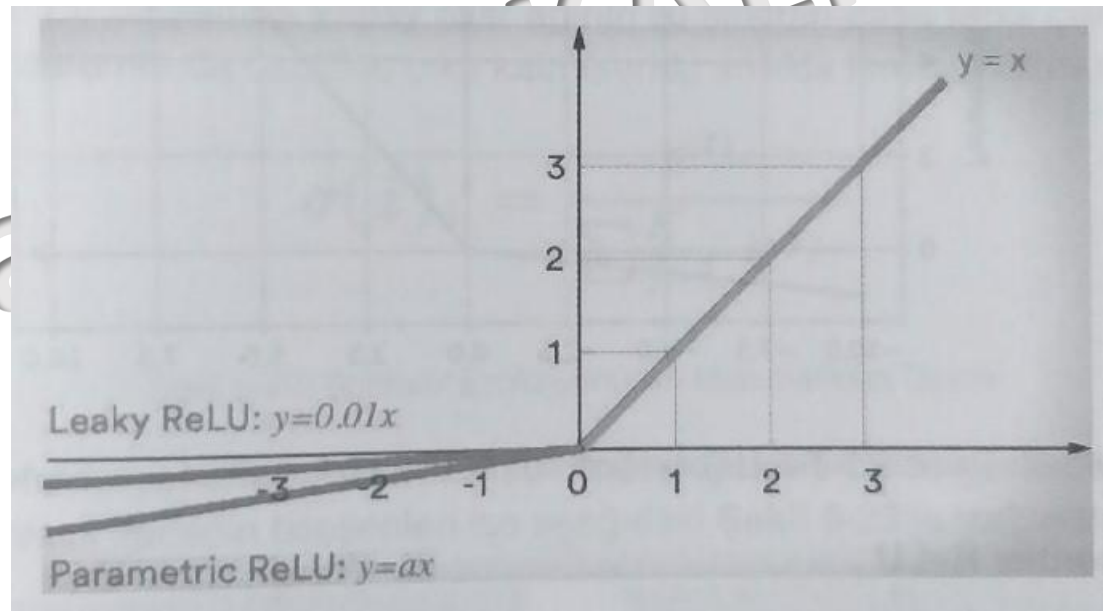
Sigmoid fonksiyonunda **çıktının 0'a yakınsaması gradyanların ölmesi** problemini oluşturssa da bu problem, **ReLU fonksiyonunda oluşmaz**. Dolayısıyla ReLU aktivasyon fonksiyonu aynı zamanda gradyanların ölmesi problemine de bir çözüm olarak geliştirilmiştir. **Buna karşın ReLU fonksiyonunda bazı nöronlar**, eşik değerinin altında sinyal ürettiklerinde doğrudan **saf dışı** kalmaktadır. Bu durum yapay sinir ağında çok sayıda **etkisiz(ölü) nöronun oluşmasına** sebep olabilir. Bu problem ise **dying ReLU** olarak adlandırılmaktadır.

UYARI

Dying ReLU problemine çözüm bulmak amacıyla **ReLU aktivasyon fonksiyonlarının alternatifleri** araştırılmıştır. Bu araştırmalar sonucunda **Leakly ReLU**, **Parameterized ReLU** ve **Exponential ReLU** fonksiyonları geliştirilmiştir.

Leakly ReLU (Sızıntı ReLU)

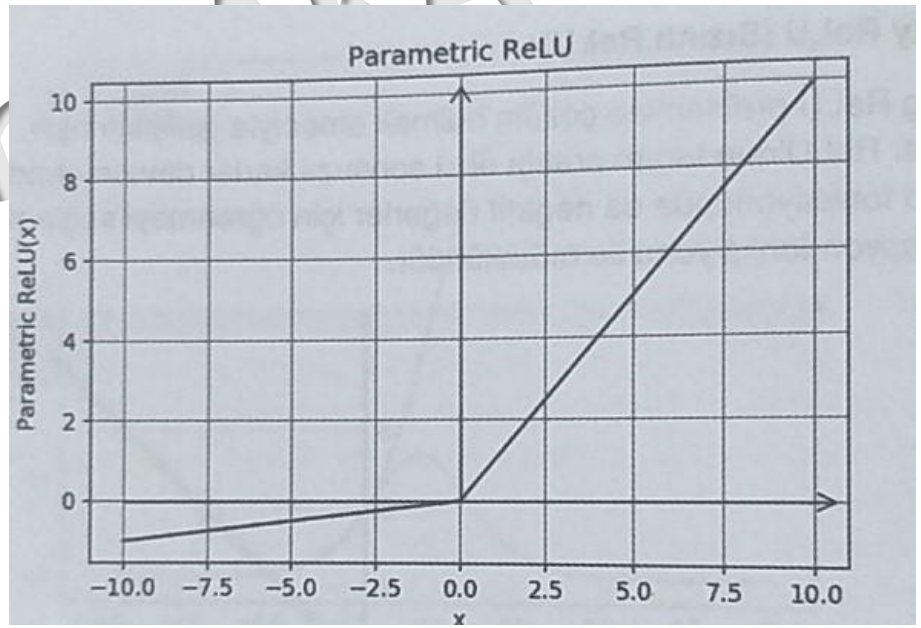
Dying ReLU problemine çözüm bulmak amacıyla geliştirilmiştir. Sızdırılan ya da Sızıntı ReLU'nun tanım aralığında eksi sonsuza kadar devam etmektedir. Kısacası ReLU fonksiyonunda da **negatif değerler için öğrenmeyi sağlamak için Leakly ReLU** aktivasyon fonksiyonu ile **mümkündür**.



Parameterized ReLU

Parameterized ya da parametric ReLU, negatif giriş değerleriyle ilgili küçük bir değişiklikle Leakly ReLU'dan ayrılmaktadır. **PReLU** olarak da bilinmektedir. Yandaki şekilde görülen grafikte **pozitif kısım lineer** iken, fonksiyonun **negatif kısmı** eğitim aşamasında **sisteme adapte** olarak öğrenmektedir. Parametrik ReLU genellikle büyük ölçekli veriler üzerinde çalışan evrişimli yapay sinir ağlarında tercih edilmektedir.

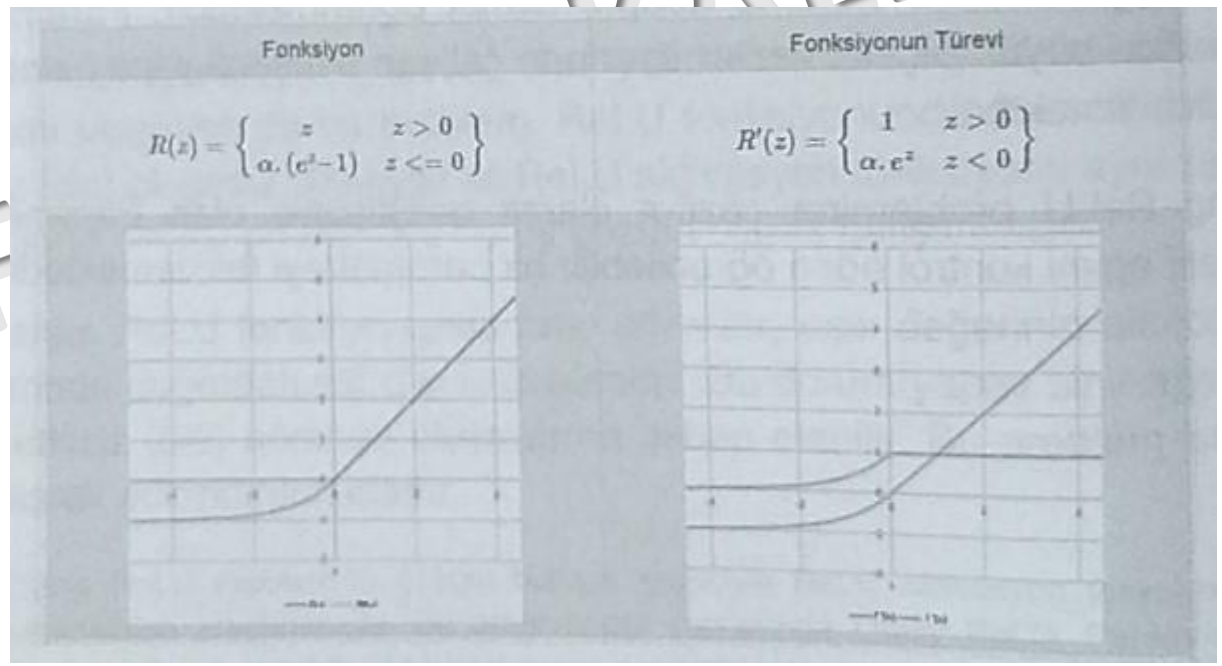
Dying ReLU problemine çözüm olarak geliştirilmiş olan Parametrik ReLU negatif eğimi kontrol eden öğrenilebilir bir parametreyi **tetiklemektedir**.



Exponential ReLU

Üstel doğrusal birim veya diğer adı ile **Exponential ReLU (ELU)**, daha hızlı sıfıra yakınsama ve daha doğru sonuçlar üretme eğiliminde olan bir fonksiyondur. Diğer aktivasyon fonksiyonlarından farklı olarak, ELU'nun ekstra bir **alfa sabiti** bulunmaktadır. Bu sayede ELU, negatif çıktılar üretebilir. Hatırlayacağınız üzere ReLU aktivasyon fonksiyonunda negatif çıktılar üretilemiyordu.

Negatif çıktılar ELU aktivasyon fonksiyonunda **doğrudan 0'a yuvarlanmadığı** için nöronların etkisizleşmesinin de önüne geçilmiş olunur.



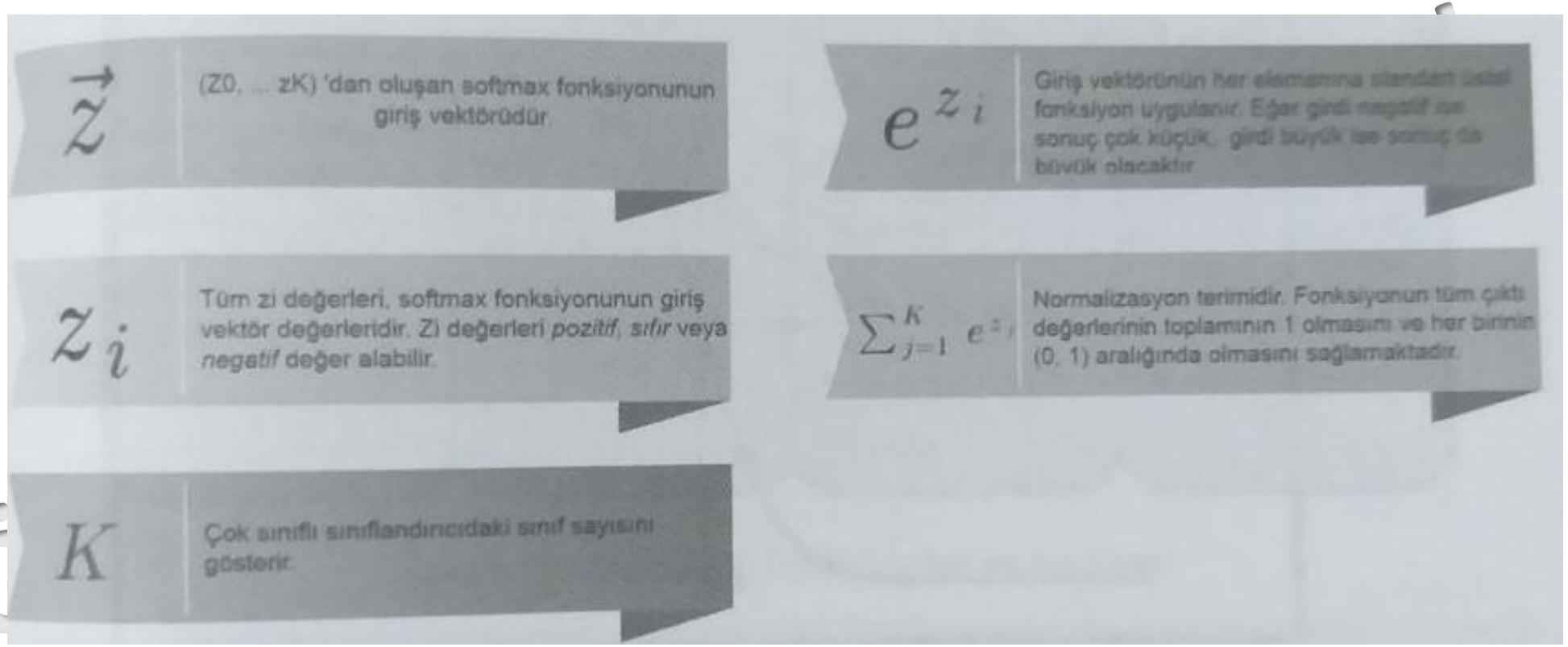
Softmax Aktivasyon Fonksiyonu

Softmax fonksiyonu, yapısı itibariyle **sigmoid aktivasyon fonksiyonuna oldukça benzerlik** göstermektedir. **Softmax** aktivasyon fonksiyonunun sigmoid aktivasyon fonksiyonundan en belirgin farkı; **sigmoid çıkışı 1 ve 0 aralığına sıkıştırırken** Softmax aktivasyon fonksiyonu 2'den fazla sınıflandırmayı destekler. Bu bağlamda yapay sinir ağının 2 sınıftan daha farklı çıktı vermesi gerektiği durumlarda **özellikle çıktı katmanında** sıklıkla tercih edilmektedir.

Softmax fonksiyonunun matematiksel ifadesi yanda görülmektedir.

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

Softmax fonksiyonun matematiksel ifadesinin bileşenleri yanda izah edilmektedir.



Softmax fonksiyonu, K giriş vektörünü, toplamı 1 olan bir K gerçek değer vektörüne dönüştüren fonksiyondur. Bu fonksiyona giriş olarak verilen değerler pozitif, negatif veya sıfır olabilir.

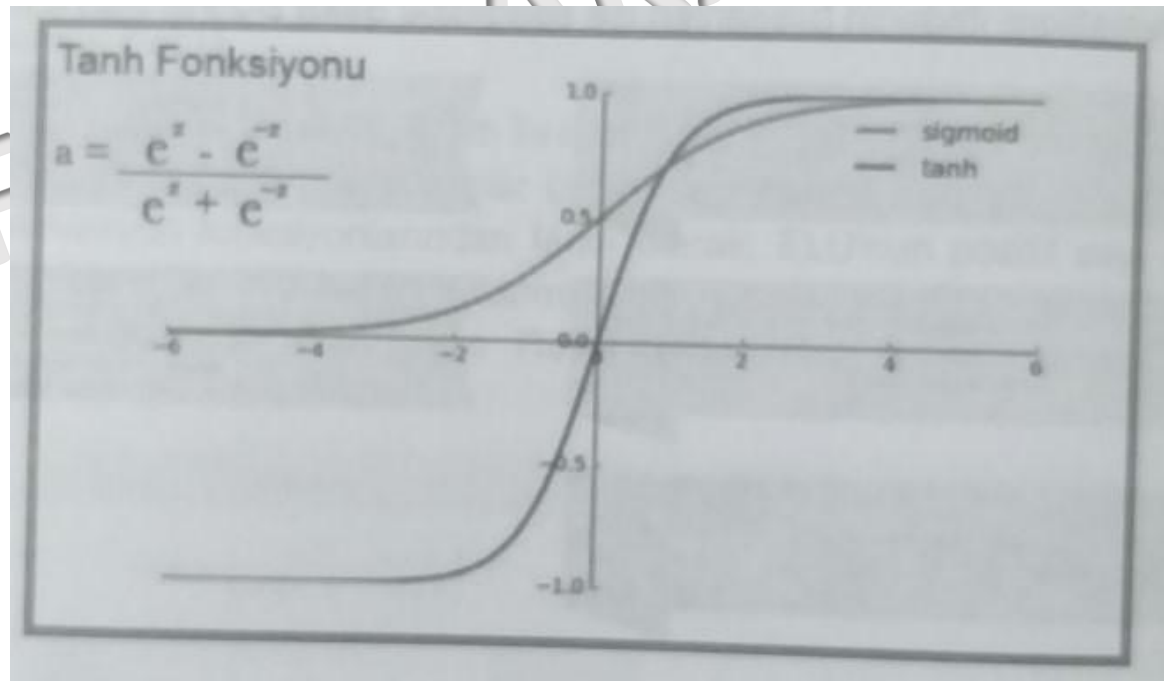
Girişlerden biri negatif ise Softmax fonksiyonu çıktı olarak küçük bir değer verirken girdi büyük ise 0 ile 1 arasında bulunan büyük bir değeri çıktı olarak vermektedir.

Sigmoid fonksiyonuna göre en önemli farkı ikiden fazla sınıflandırma yapılması gereken problemlerde kullanılabilir olmasıdır. Bu nedenle Softmax aktivasyon fonksiyonu sınıflayıcı olarak çıkış katmanında tercih edilebilir.

Hiperbolik Tanjant (Tanh) Aktivasyon Fonksiyonu

Hiperbolik tanjant fonksiyonu tıpkı softmax fonksiyonu gibi yapı itibariyle sigmoid aktivasyon fonksiyonuna oldukça benzemektedir.

Hiperbolik tanjant fonksiyonu aynı zamanda **tanh fonksiyonu** olarak da bilinmektedir. Tanh ve sigmoid aktivasyon fonksiyonlarının 2 boyutlu koordinat düzlemi üzerinde gösterimi yanda görülmektedir.

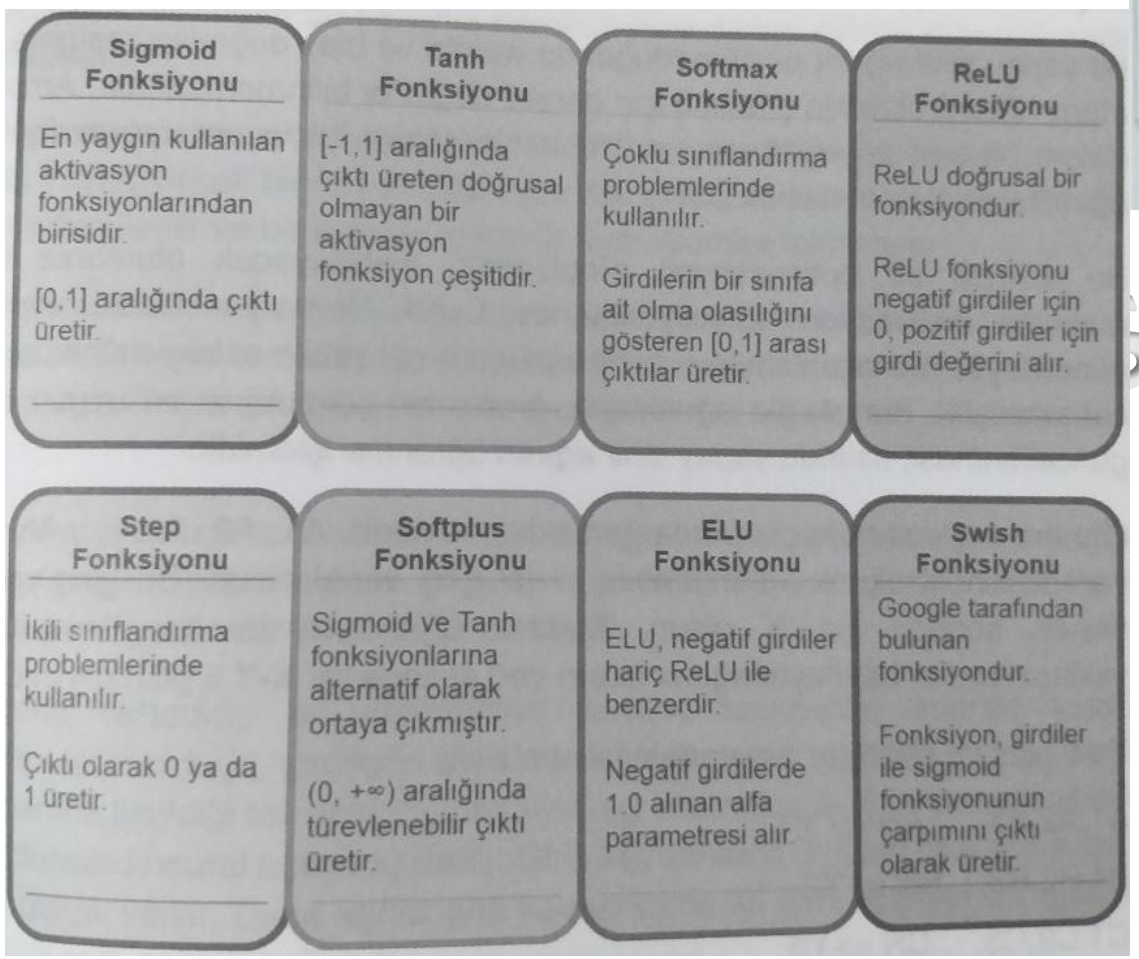


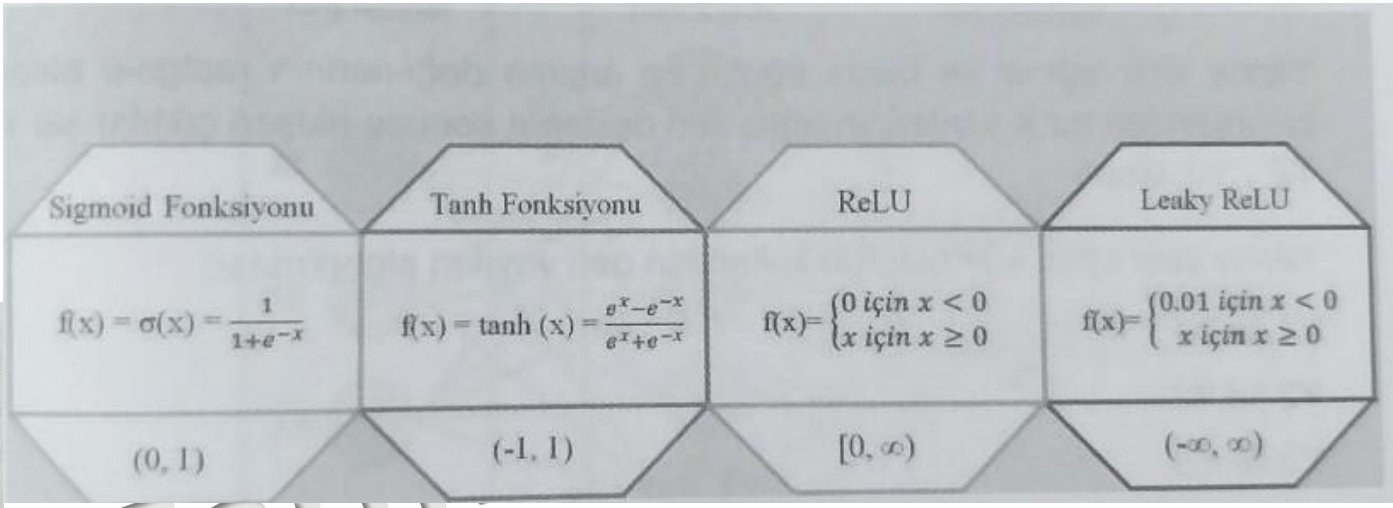
Hiperbolik tanjant fonksiyonunu **(-1,+1) tanım aralığında** geçerlidir. Sigmoid fonksiyonu ise hatırlayacağınız üzere **(0,1) tanım aralığında** geçerli idi. Basit bir ifadeyle **hiperbolik tanjant** fonksiyonu sigmoid fonksiyonuna göre **daha geniş tanım aralığına** sahiptir. Bu durum sınıflandırma problemlerinde daha geniş aralığa sahip olmasını ifade eder.

Avantajlarının yanı sıra *hiperbolik tanjant fonksiyonunda da tıpkı sigmoid fonksiyonu gibi* gradyanların ölmesi (vanishing gradient) problemi bulunmaktadır.

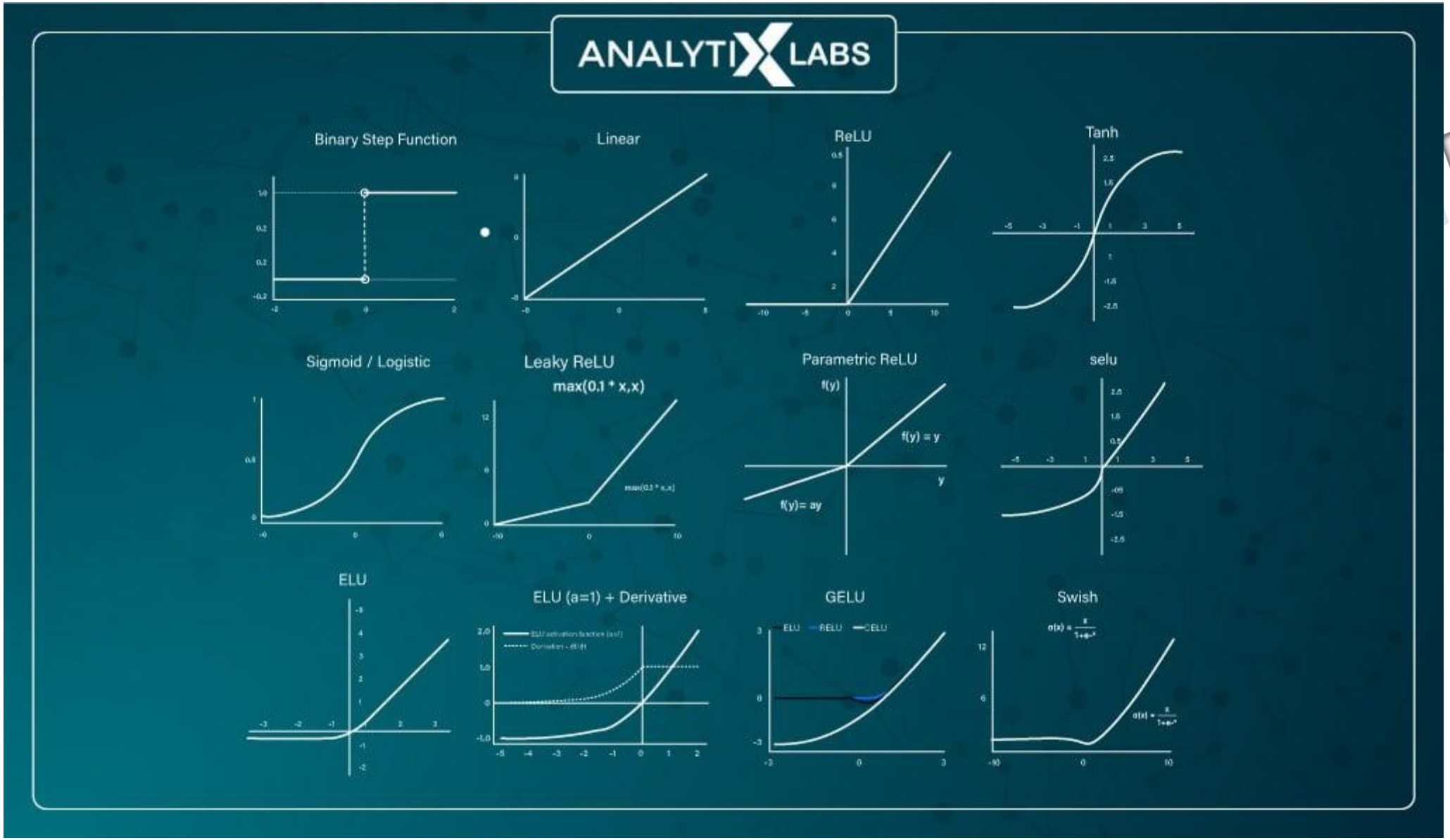
Hiperbolik tanjant fonksiyonu hiperbolik sinüs ve hiperbolik kosinüsün birbirlerine oranlarından oluşmaktadır. Diğer bir ifadeyle iki üstel ifadenin yarı farkının iki üstel ifadenin yarı toplamına oranıdır.

Aktivasyon Fonksiyonları ve Özellikleri



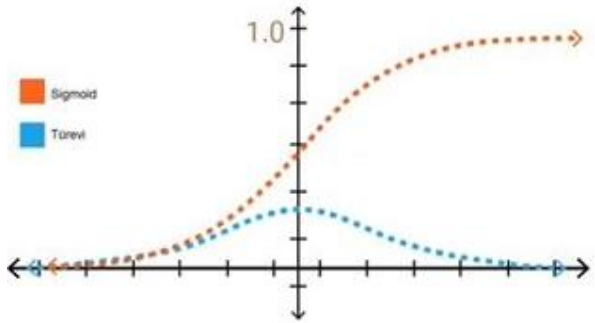


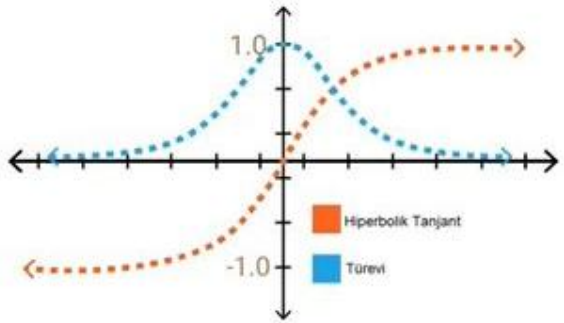
Aktivasyon Fonksiyonlarının Matematiksel Formülleri ve Tanım Aralıkları

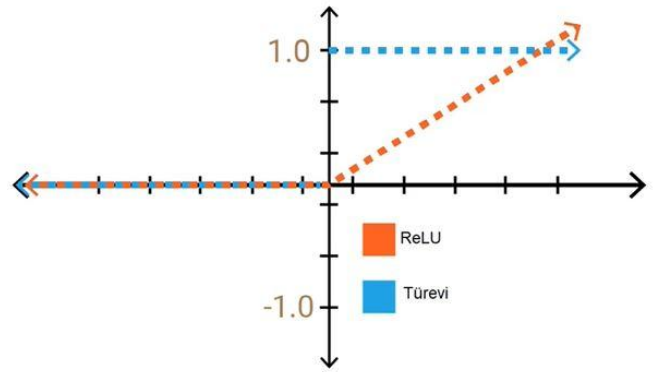


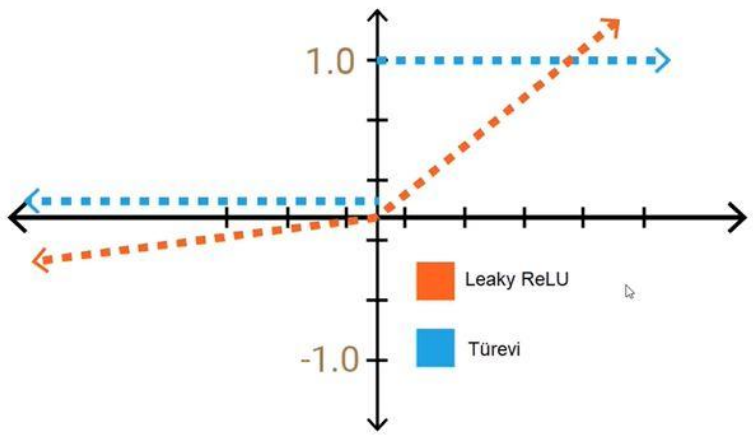
https://www.google.com/imgres?imgurl=https%3A%2F%2Fwww.analytixlabs.co.in%2Fblog%2Fwp-content%2Fuploads%2F2023%2F08%2FTypes-of-activation-function.jpg&tbnid=kwmOCfeyHfD2QM&vet=10CAIQxiAoAGoXChMI4IWy4YnMgwMVAAAAAB0AAAAAEA4..i&imgrefurl=https%3A%2F%2Fwww.analytixlabs.co.in%2Fblog%2Factivation-function-in-neural-network%2F&docid=Cqshe0935WmAzM&w=1024&h=594&itg=1&q=softmax%20activation%20function&hl=en-GB&ved=0CAIQxiAoAGoXChMI4IWy4YnMgwMVAAAAAB0AAAAAEA4

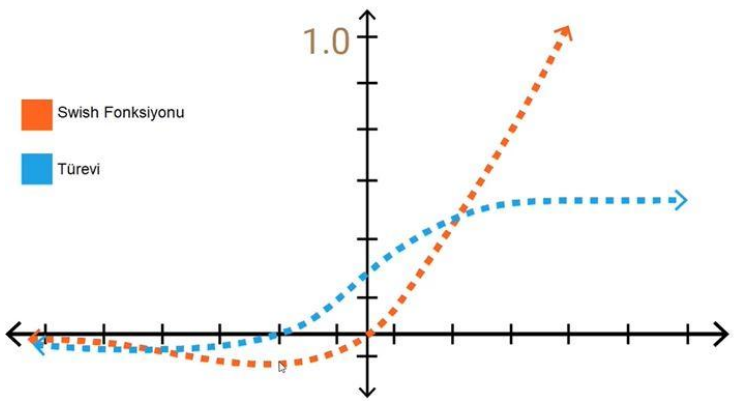
AKTİVASYON FONKSİYONU SEÇİMİ











https://ayyucekizrak.medium.com/derin-%C3%B6%C4%9Frenme-i%C3%A7in-aktivasyon-fonksiyonlar%C4%B1n%C4%B1n-kar%C5%9F%C4%B1la%C5%9Ft%C4%B1r%C4%B1lmas%C4%B1-cee17fd1d9cd

Yapay Sinir Ağlarının Öğrenmesi

Bir yapay sinir ağı, yapısındaki **ağırlık ve sapma** değerlerine göre **giriş verilerini işleyerek** bir **çıktı** üretmektedir. Yani bir yapay sinir ağı, nöron ve sinaps ağırlıkları sayesinde karar vermektedir.

Bir yapay sinir ağı ilk oluşturulduğunda **weight ve bias** değerleri **rastgele** olarak atanır. Zira problemin çözümü için **gerekli değerler bilinmemektedir**. **Amaç**, sinir ağının bir veri ile eğitilmesi ve problemin çözümü için **en uygun ağırlıklara** sahip olmasıdır. Burada da ağırlıkların problemin çözümüne en uygun biçimde güncellenmesi, aslında yapay sinir ağının **öğrenme işlemidir**.

Öğrenme işlemi, birkaç adımda gerçekleşmektedir. **A1, A2, .. AN** dizisi giriş verisi olsun. Bu giriş verilerinin bilinen sonucu ise Y olsun. Makine öğrenmesinin denetimli öğrenme yaklaşımından hatırlayıcığımız üzere veri setinde bir **X-Y ilişkisi** olmalıdır.

Yani giriş ve sonuçlar arasındaki bilinen ilişki:

A1,A2,A3…AN=>**Y1**

B1,B2, B3…BN=>**Y2**

C1,C2,C3…CN=>**Y3** olarak ifade edilebilir.

Yapay sinir ağının ilk başta ağırlık ve sapma değerlerinin rastgele atandığı bilindiğinden bu **X verilerinin ağda ileri besleme sonucunda** oluşan çıktılar ise **Y1',Y2',Y3'** olsun.

Yapay Sinir ağını eğitmek için kullanılan **geri yayılım algoritması**;

Y1 ile Y1’

Y2 ile Y2’

Y3 ile Y3’

arasındaki hata oranını belirli teknik ve metriklerle hesaplayarak yapay sinir ağının ağırlıklarının güncellenmesinde kullanılır. Bu işlem sırasıyla **ileri besleme, hatayı hesaplama, ağırlıkları güncelleme(geri yayılım) ve tekrar etme** aşamalarından oluşmaktadır

İleri Besleme Algoritması

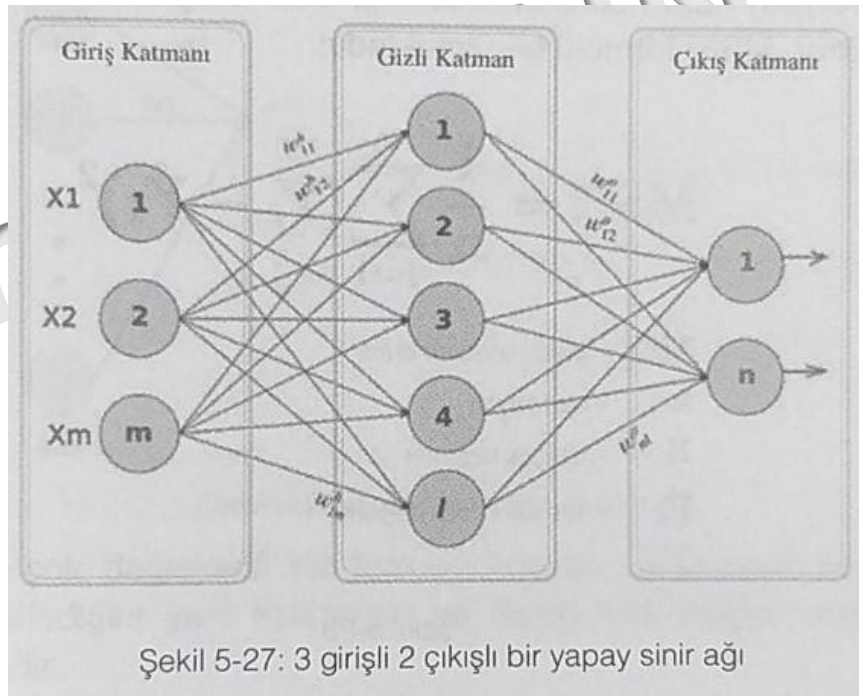
İleri besleme algoritması; **giriş nöronlarına giriş verilerinin uygulanması** ile başlayan bir süreçtir. Kısaca giriş verileri bir sinyale dönüştürülür ve **alıcı nöron** tarafından **her sinyal geldiği** sinapsin ağırlık değeri ile çarpılır ve toplanır. Ardından bu toplama, nöronun **bias değeri eklenir** (Feed Summary Hesaplaması).

Bu işlemin ardından toplam sonucu **aktivasyon fonksiyonundan geçirilerek ağda yayılabilecek bir sinyal** elde edilir. Bu sinyal bir sonraki nörona bir sinaps üzerinden **iletilir** ve bir sonraki rolünde aynı işlemler **tekrarlanır**. Bu işlem aktif katmandaki **her nöron** için tekrarlanmaktadır ve **soldan sağa doğru bir ilerleyişle** daima bir sonraki katmanın nöronlarına **sinyallerin iletilmesi** amaçlanır.

Sinyal, çıkış katmanına ulaştığında ise ağdan bir çıktı elde edilmiş olur. Ağırlıkları ve sapma değerleri olan bir yapay sinir ağından **her nöron** için **hesaplamanın** yapılması ile **çıkış katmanından bir sonuç elde edilmesi** ileri besleme algoritması olarak bilinmektedir. Tekrarın olmadığı bir katmandan sadece kendinden sonraki katmana bağlantıların olduğu girdilerin giriş katmanından gizli katmana oradan da çıkış katmanına iletildiği **tek yönlü yapay sinir ağı modeline ileri beslemeli sinir ağı(feedforward neural network)** denir.

Giriş katmanında üretilen çıktı **gizli katmana girdi** olarak verilir. Daha sonra **gizli katmanlarda da aynı işlemler tekrarlanır** ve üretilen çıktı en sonunda çıktı katmanına ulaşır. Bir yapay sinir ağının giriş katmanında giriş parametresi sayısı kadar nöron bulunmalıdır. Tüm verilerin ağa sağlanabilmesi için bu gereklidir.

Gizli katmandaki **nöron sayısı** problemin karmaşıklığına göre değişmektedir. Gizli katmanlardaki nöron sayısı, sinir ağı mimarisini tasarlayan kişi tarafından belirlenir. Nöron sayısı **sezgisel olarak veya deneme yanılma** yoluyla belirlenebilir. Çıkış katmanında nöron sayısı ise belirlenen çıktı sınıf sayısı adetince olmalıdır. İleri beslemeli sinir ağında kullanılan modelin çıktısı, hata oranı kullanılarak geri besleme işlemi gerçekleştirmeden tekrarsız bir şekilde oluşturulur. Bu yüzden katmanlar arasında **geri besleme bağlantıları bulunmaz**. Eğer ileri beslemeli sinir ağı modelinde geri beslenme olacak ise geri besleme işlemine geri beslemeli sinir ağı denir.



Şekil 5-27: 3 girişli 2 çıkışlı bir yapay sinir ağı

Hatanın Hesaplanması

Loss, geri besleme algoritmasında kullanılmak üzere hesaplanan **hata değeridir**. Ağırlıkların güncellenmesinde kısmi türevde kullanıldığı için **yapay sinir ağının** öğrenmesinde hayati önem taşır. Bir her X-Y (Giriş-Sonuç) ikilisi için **beklenen sonuç** ile **tahmin edilen çıktı** arasındaki **fark** kullanılarak hata değeri hesaplanır. Hesaplanan **hata** başlangıçta rastgele olarak atanan ağırlıklar ve sapma değerlerini düzenlemek için **geri besleme algoritmasında** kullanılır.

Maliyet hesabının **minimize edilmesi için öncelikle maliyetin bulunması gereklidir**. Maliyet her bir proses elemanı için elde edilecek loss değerlerin toplanılmasına karşılık gelir. Loss fonksiyonları, geri yayılım algoritmasındaki hata miktarının hesaplanması için kullanılan bir parametredir ve problemin tipine göre seçilir.

Hatayı hesaplamak için birçok metrik tercih edilebilir.

Bu doğrultuda regresyon tipi bir problem için **MSE(l2 loss)** ve **MAE(l1 loss)** kayıp fonksiyonları kullanılırken sınıflandırma tipi problemler için **binary cross entropy** ve **categorical cross entropy** kayıp fonksiyonları kullanılmaktadır.

Mean Squared Error (MSE)

$$\text{MSE}=\frac{1}{N}\sum_{i=1}^{N}(y_i - y_i')^2$$

Mean Absolute Error (MAE)

$$\text{MAE}=\frac{1}{n}\sum_{j=1}^{n}|y_j - y_j'|$$

https://towardsdatascience.com/understanding-the-3-most-common-loss-functions-for-machine-learning-regression-23e0ef3e14d3

CCE= $$L = -\frac{1}{m}\sum_{i=1}^{m} y_i \cdot \log(p_i)$$

$$BinaryCrossEntropy = -\frac{1}{N}\sum_{i=1}^{N}[y_i * log(y_{pred}) + (1 - y_i) * log(1 - y_{pred})]$$

Binary cross entropy formula

Formül:

$$L = -\frac{1}{m}\sum_{i=1}^{m} y_i \log(p_i)$$

Burada:

- y_i: Gerçek etiket (one-hot encoded formatında 0 veya 1 olur).
- p_i: Modelin tahmin ettiği olasılık değeri.
- m: Örnek sayısı.

p_i Nasıl Hesaplanır?

Tahmin edilen değerler **softmax aktivasyon fonksiyonundan** geçerek **olasılığa dönüştürülür**. Yani:

$$p_i = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

Burada:

- z_i: Modelin i. sınıf için ürettiği ham skor (logit).
- **Softmax fonksiyonu**, tahminleri 0 ile 1 arasına çeker ve tüm sınıfların toplamının 1 olmasını sağlar.

pi değeri **modelin tahmin ettiği olasılık** değeridir ve doğrudan hesaplama sürecine dahil edilir.
z_i , modelin çıktı katmanından önceki aktivasyon fonksiyonuna girmeden önceki ham skorudur.Başka bir deyişle, sinir ağının son katmanında ağırlıklar ve bias'lar ile hesaplanan değerdir.
zi değeri **softmax fonksiyonundan** geçirilerek **pi=ypred** elde edilir

https://medium.com/@omerkaanvural/cross-entropy-loss-nedir-56805a94cb09

https://www.v7labs.com/blog/cross-entropy-loss-guide#:~:text=Categorical%20Cross%20Entropy%20is%20also,N%20classes%20for%20each%20image.

https://gombru.github.io/2018/05/23/cross_entropy_loss/

Kayıp Fonksiyonu Hesaplama 1

Hedef ve tahmin değerler aşağıda verilen çıktılar üzerinden ağın loss fonksiyon değerini Mean Squared Error (MSE) vasıtasıyla hesaplayınız.

MSE formülü:

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(T_i - Y_i)^2$$

Hesaplama:

1. $(T_1 - Y_1)^2 = (2-0)^2 = 4$
2. $(T_2 - Y_2)^2 = (2-2)^2 = 0$
3. $(T_3 - Y_3)^2 = (3-1)^2 = 4$
4. $(T_4 - Y_4)^2 = (5-3)^2 = 4$

Toplam hata: $4 + 0 + 4 + 4 = 12$

Ortalama hata:

$$\text{MSE} = \frac{12}{4} = 3$$

Hedefler (Targets): T={[2],[2],[3],[5]}
Tahminler (Predicted): Y={[0],[2],[1],[3]}

Kayıp Fonksiyonu Hesaplama 2

Mean Absolute Error (MAE) (Ortalama Mutlak Hata), tahminler ile gerçek değerler arasındaki mutlak farkların ortalamasını hesaplar.

$$\text{MAE} = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|$$

Gerçek Değerler (Targets):

$$y = [3, 5, 2]$$

Tahminler (Predictions):

$$\hat{y} = [2.5, 4.8, 2.2]$$

Toplam MAE Hesaplaması:

Şimdi, her bir örneğin mutlak farklarını topluyoruz ve örnek sayısına bölüyoruz.

$$\text{MAE} = \frac{1}{3}(0.5 + 0.2 + 0.2)$$

$$\text{MAE} = \frac{1}{3} \times 0.9 = 0.3$$

Adım adım MAE hesaplaması:

1. **Örnek 1:**
   - Gerçek değer: $y_1 = 3$
   - Tahmin edilen değer: $\hat{y}_1 = 2.5$

   Mutlak fark:

   $$|y_1 - \hat{y}_1| = |3 - 2.5| = 0.5$$

2. **Örnek 2:**
   - Gerçek değer: $y_2 = 5$
   - Tahmin edilen değer: $\hat{y}_2 = 4.8$

   Mutlak fark:

   $$|y_2 - \hat{y}_2| = |5 - 4.8| = 0.2$$

3. **Örnek 3:**
   - Gerçek değer: $y_3 = 2$
   - Tahmin edilen değer: $\hat{y}_3 = 2.2$

   Mutlak fark:

   $$|y_3 - \hat{y}_3| = |2 - 2.2| = 0.2$$

Kayıp Fonksiyonu Hesaplama 3

Hedef ve tahmin değerler aşağıda verilen çıktılar üzerinden ağın loss fonksiyon değerini Categorical Cross Entropy (CCE) vasıtasıyla hesaplayınız.

Hedefler (Targets) Matrisi:

$$T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Tahminler (Predictions) Matrisi:

$$Y = \begin{bmatrix} 0.7 & 0.2 & 0.1 \\ 0.1 & 0.8 & 0.1 \\ 0.2 & 0.2 & 0.6 \end{bmatrix}$$

Categorical Cross-Entropy (CCE) Formülü:

$$\text{CCE} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{C} T_{ij} \cdot \log(Y_{ij})$$

Burada:

- N = 3 (toplam örnek sayısı),
- C = 3 (sınıf sayısı).

Adım adım hesaplama:

1. **Örnek 1:**
   - Hedef: $[1, 0, 0]$
   - Tahmin: $[0.7, 0.2, 0.1]$

   CCE hesaplaması:

$$\text{CCE}_1 = -\left(1 \cdot \log(0.7) + 0 \cdot \log(0.2) + 0 \cdot \log(0.1)\right)$$
$$= -\log(0.7) \approx 0.357$$

2. **Örnek 2:**
   - Hedef: $[0, 1, 0]$
   - Tahmin: $[0.1, 0.8, 0.1]$

   CCE hesaplaması:

$$\text{CCE}_2 = -\left(0 \cdot \log(0.1) + 1 \cdot \log(0.8) + 0 \cdot \log(0.1)\right)$$
$$= -\log(0.8) \approx 0.223$$

3. **Örnek 3:**
   - Hedef: $[0, 0, 1]$
   - Tahmin: $[0.2, 0.2, 0.6]$

   CCE hesaplaması:

$$\text{CCE}_3 = -\left(0 \cdot \log(0.2) + 0 \cdot \log(0.2) + 1 \cdot \log(0.6)\right)$$
$$= -\log(0.6) \approx 0.511$$

Hedefler (Targets) Matrisi:

$$T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Tahminler (Predictions) Matrisi:

$$Y = \begin{bmatrix} 0.7 & 0.2 & 0.1 \\ 0.1 & 0.8 & 0.1 \\ 0.2 & 0.2 & 0.6 \end{bmatrix}$$

Toplam CCE Hesaplaması:

Şimdi bu değerlerin toplamını alalım:

$$\text{CCE}_{\text{total}} = \frac{1}{3}\left(0.357 + 0.223 + 0.511\right)$$

Toplam:

$$\text{CCE}_{\text{total}} = \frac{1}{3} \cdot 1.091 = 0.364$$

Kayıp Fonksiyonu Hesaplama 4

Binary Cross-Entropy (BCE), ikili sınıflandırma problemlerinde kullanılan bir kayıp fonksiyonudur ve tahminler ile gerçek değerler arasındaki farkı hesaplar. İkili sınıflandırmada hedef değerler 0 veya 1 olabilir ve tahmin edilen değerler olasılık değeri olarak 0 ile 1 arasında bir değer alır.

$$\text{BCE} = -\frac{1}{N}\sum_{i=1}^{N}[y_i \cdot \log(\hat{y}_i) + (1-y_i)\cdot\log(1-\hat{y}_i)]$$

Diyelim ki 3 örneğimiz var ve bu örnekler için gerçek etiketler ve tahmin edilen olasılık değerleri şu şekilde:

Gerçek Etiketler (Targets):

$$y = [1, 0, 1]$$

Tahminler (Predictions):

$$\hat{y} = [0.8, 0.3, 0.9]$$

Adım adım BCE hesaplaması:

1. **Örnek 1:**
   - Gerçek etiket: $y_1 = 1$
   - Tahmin edilen değer: $\hat{y}_1 = 0.8$

   BCE hesaplaması:

$$\text{BCE}_1 = -[1 \cdot \log(0.8) + (1-1)\cdot\log(1-0.8)]$$
$$= -\log(0.8) \approx 0.223$$

2. **Örnek 2:**
   - Gerçek etiket: $y_2 = 0$
   - Tahmin edilen değer: $\hat{y}_2 = 0.3$

   BCE hesaplaması:

$$\text{BCE}_2 = -[0 \cdot \log(0.3) + (1-0)\cdot\log(1-0.3)]$$
$$= -\log(0.7) \approx 0.357$$

3. **Örnek 3:**
   - Gerçek etiket: $y_3 = 1$
   - Tahmin edilen değer: $\hat{y}_3 = 0.9$

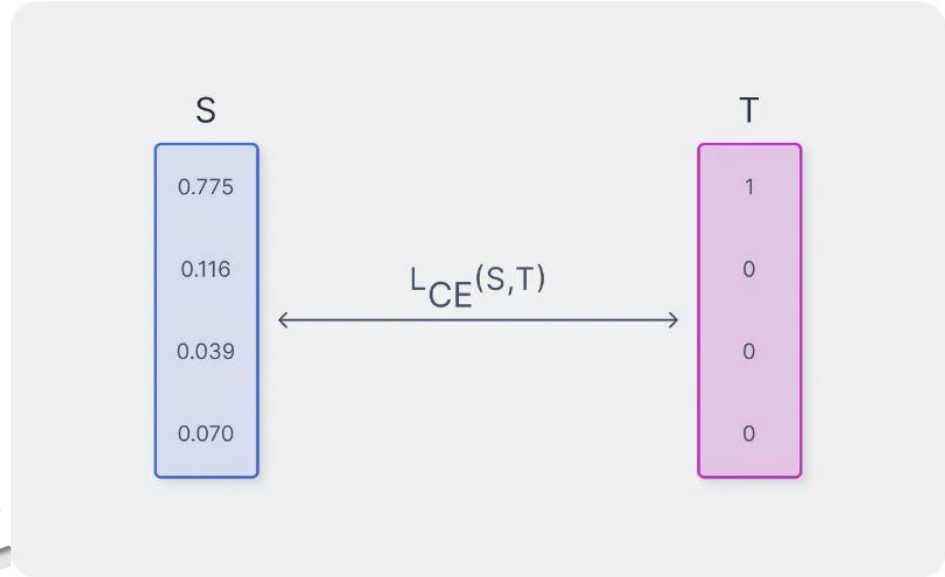
   BCE hesaplaması:

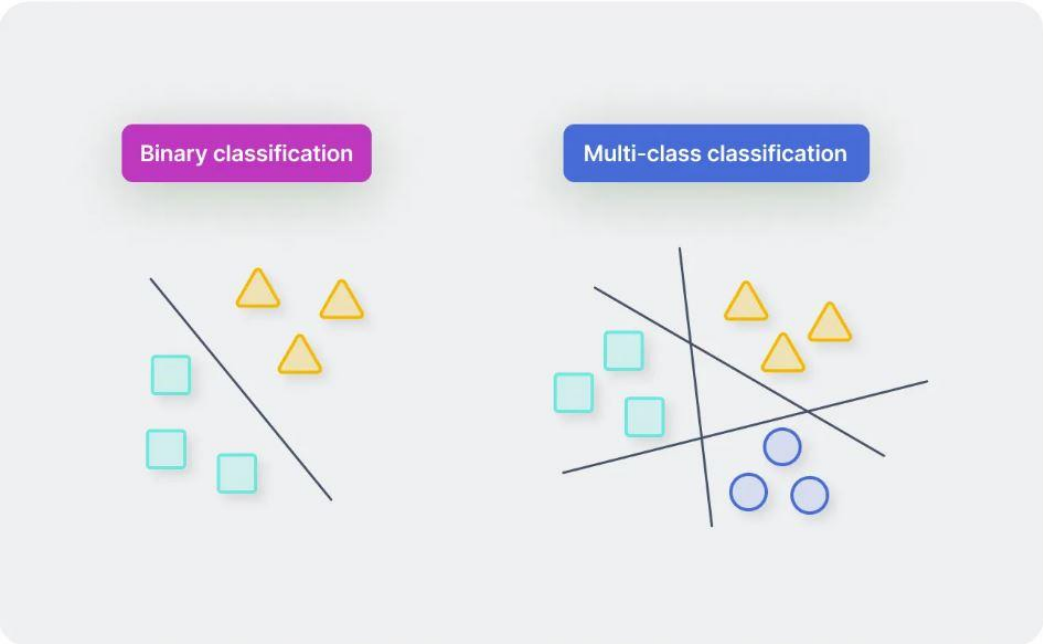
$$\text{BCE}_3 = -[1 \cdot \log(0.9) + (1-1)\cdot\log(1-0.9)]$$
$$= -\log(0.9) \approx 0.105$$

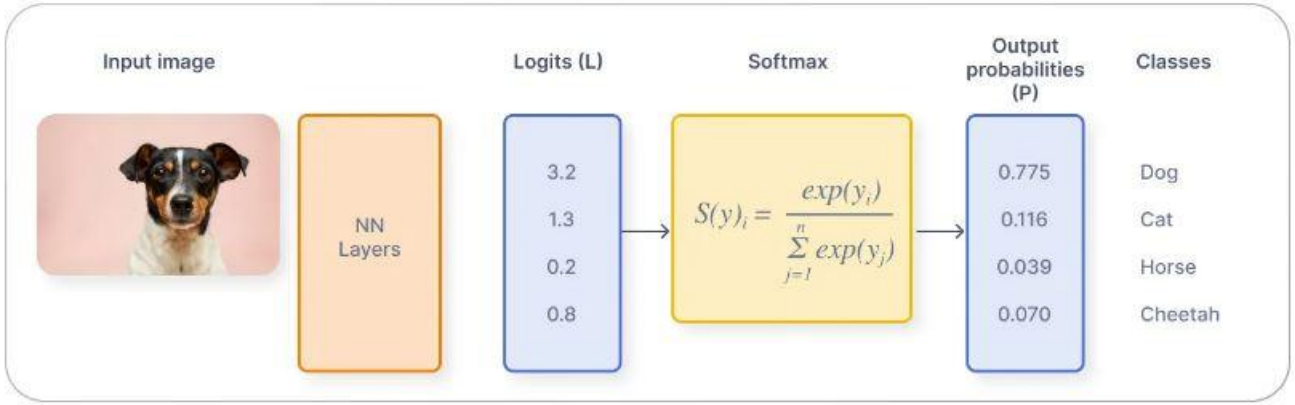
$$\text{BCE}_{\text{total}} = \frac{1}{3}(0.223 + 0.357 + 0.105)$$

$$\text{BCE}_{\text{total}} = \frac{1}{3} \times 0.685 = 0.228$$

Özellik	Regresyon	Sınıflandırma
Hedef	Sürekli sayılar	Etiketler/Kategoriler
Çıktı	Sürekli sayılar	Belirli sınıflar
Örnekler	Ev fiyatı tahminleri, sıcaklık tahminleri	E-posta spam olup olmama, görüntü sınıflandırma
Kayıp Fonksiyonları	MSE, MAE, RMSE	BCE, CCE
Performans Ölçütleri	R^2, MSE, MAE	Accuracy, Precision, Recall, F1-Score







https://www.v7labs.com/blog/cross-entropy-loss-guide

Geri Yayılım Algoritması

İleri besleme algoritması işletilip bir **çıkış üretildiğinde tahmin edilen** değerle **gerçek değer karşılaştırılır** ve bir **hata oranı** belirlenir. Hesaplanan **hata oranı** kullanılarak **ters yönde** yani **sağdan sola doğru**(çıkış katmanından giriş katmanı yönüne) bir **geri yayılım işlemi** başlatılır. **Geri yayılımda** hata oranına göre **ağırlık ve sapma** değeri **güncellenir.** Çıkışa etkisi daha fazla yada daha az olan nöronun ağırlığı arttırılır ya da azaltılır.

Giriş katmanında giriş değerleri ile güncellenen ağırlıklar çarpılır ve toplama fonksiyonu ile toplanır. Elde edilen değere sapma değeri eklenerek aktivasyon fonksiyonundan geçirilir. Aktivasyon fonksiyonundan geçirilerek elde edilmiş sinyal bir sonraki katmana iletilir. **İleri yayılımda bu işlem giriş katmanından başlayıp çıkış katmanına kadar sürdürülürken** geri yayılımda ise bu işlemi çıkış katmanından başlayıp giriş katmanına kadar devam eder. **İleri besleme sonucu oluşan sonuç**, bilinen sonuçtan çıkarılır ve **çeşitli hata tespit metrikleriyle ağırlıklar güncellenir**. Böylelikle ***en verimli*** sinir ağı modeli oluşturulmaya çalışılır.

Bu işlem **geri yayılım (back propagation) algoritması** olarak bilinmektedir

Geri Yayılım

Geri yayılımda, modelin ağırlıkları loss fonksiyonuna göre güncellenir. Amaç, loss fonksiyonunun değerini minimize etmektir. Bu işlem, **gradient descent (gradyan inişi)** algoritmasına dayanır. Adımlar şöyle olur:

a) Error (Hata) Hesaplama:

Modelin çıktısı ile gerçek değer arasındaki fark (hata) hesaplanır:

Error=Predicted Output−True Output

b) Gradient (Gradyan) Hesaplama:

Gradyan, loss fonksiyonunun çıktısının ağırlıklar üzerindeki türevini temsil eder. Yani, her ağırlığın loss fonksiyonuna olan katkısını gösterir. Bu gradyanlar, loss fonksiyonunu minimize etmek için gereklidir. Örneğin, bir ağırlığın gradyanı şu şekilde hesaplanır:

$$\partial Loss/\partial wi$$

Bu, ağırlığın, loss fonksiyonu üzerindeki etkisini gösteren türevdir.

c) Zincir Kuralı (Chain Rule):

Sinir ağlarında birden fazla katman olduğu için, gradyan hesaplamaları **zincir kuralı**na dayanır. Zincir kuralı, bir katmandaki gradyanın önceki katmanlara nasıl iletildiğini açıklar. Bu işlem her katman için tekrar edilir, bu da "geri yayılım" adımını oluşturur.

Hata değerimiz L

Ağdaki ağırlıkları ağırlık değerleri W1,W2,W3,W4,W5,W6

Ağdaki **sapma** değerleri B1,B2,B3 olsun.

Ağırlık ve sapma değerlerinin her birinin yapay sinir ağına etkisi aşağıdaki formül de gösterilmektedir.

$$\frac{\partial L}{\partial w_1}, \frac{\partial L}{\partial w_2}, \frac{\partial L}{\partial w_3}, \frac{\partial L}{\partial w_4}, \frac{\partial L}{\partial w_5}, \frac{\partial L}{\partial w_6}, \frac{\partial L}{\partial b_1}, \frac{\partial L}{\partial b_2}, \frac{\partial L}{\partial b_3}$$

Bu kısmi türev ise türevin zincir kuralı ile elde edilmektedir

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial Y_{\text{Tahmin}}} \cdot \frac{\partial Y_{\text{Tahmin}}}{\partial w_1}$$

d) Ağırlıkların Güncellenmesi (Weight Update):

Gradyanlar hesaplandıktan sonra, ağırlıklar aşağıdaki formüle göre güncellenir:

$$wi = wi - \eta . (\partial Loss / \partial wi)$$

wi: Ağırlık,
η: Öğrenme oranı (learning rate),
∂Loss/∂wiAğırlığın gradyanı (türev)

Örnek Bir Regresyon Problemi için Geri Yayılım

Örnek olarak basit bir lineer regresyon modelini ele alalım. Modelin formülü şu şekilde:

y=wx+b

Burada:

w ağırlık,

b bias (terim),

x giriş verisi,

y modelin tahmini çıktısı.

a) İleri Yayılım:

- Giriş $x = 2$, gerçek hedef $y_{\text{true}} = 5$.
- Başlangıçta $w = 1, b = 0$.
- Modelin tahmini çıktısı:

$$y = wx + b = 1 \cdot 2 + 0 = 2$$

b) Loss Hesaplama (MSE):

MSE kaybını hesaplayalım:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_{\text{pred}} - y_{\text{true}})^2$$

Burada $n = 1$ olduğu için:

$$\text{MSE} = (2 - 5)^2 = 9$$

c) Gradyan Hesaplama:

Gradyanlar (gradient), loss fonksiyonunun türevlerini ifade eder. Bu türevler, ağırlıkların ve bias'ın loss üzerindeki etkisini ölçer. Her bir parametre (w ve b) için türev hesaplanır:

1. Ağırlık (w) için türev:

MSE'nin türevini w'ye göre alalım:

$$\frac{\partial \text{MSE}}{\partial w} = 2 \cdot x \cdot (y_{\text{pred}} - y_{\text{true}})$$

Burada:

- x: Giriş değeri,
- $(y_{\text{pred}} - y_{\text{true}})$: Tahmin hatası.

Değerleri yerine koyarak hesaplayalım:

$$\frac{\partial \text{MSE}}{\partial w} = 2 \cdot 2 \cdot (2 - 5) = 2 \cdot 2 \cdot (-3) = -12$$

2. Bias (b) için türev:

MSE'nin türevini b'ye göre alalım:

$$\frac{\partial \text{MSE}}{\partial b} = 2 \cdot (y_{\text{pred}} - y_{\text{true}})$$

Burada yalnızca tahmin hatasını kullanıyoruz çünkü b'nin x ile çarpımı yoktur.

Değerleri yerine koyarak hesaplayalım:

$$\frac{\partial \text{MSE}}{\partial b} = 2 \cdot (2 - 5) = 2 \cdot (-3) = -6$$

∂MSE/ ∂w türevini anlamak için **Mean Squared Error (MSE)** fonksiyonunun matematiksel türevini adım adım inceleyelim:

1. MSE Fonksiyonu:

Mean Squared Error (MSE), tahmin (y_{pred}) ve gerçek (y_{true}) değerler arasındaki farkın karesinin ortalamasıdır:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_{\text{pred},i} - y_{\text{true},i})^2$$

Bir örnek üzerinde çalışıyoruz ve burada yalnızca bir veri noktası ($n = 1$) var:

$$\text{MSE} = (y_{\text{pred}} - y_{\text{true}})^2$$

2. w için türev:

$y_{\text{pred}} = wx + b$ olduğunu biliyoruz. MSE'yi w'ye göre türev alalım:

Adım 1: MSE'nin y_{pred}'ye göre türevi:

Öncelikle, MSE'nin y_{pred}'ye göre türevini alalım:

$$\frac{\partial \text{MSE}}{\partial y_{\text{pred}}} = 2 \cdot (y_{\text{pred}} - y_{\text{true}})$$

Bu ifade, tahmin hatasının iki katıdır ($2 \cdot \text{hata}$).

Adım 2: y_{pred}'nin w'ye göre türevi:

$y_{\text{pred}} = wx + b$ olduğundan, y_{pred}'nin w'ye göre türevini alalım:

$$\frac{\partial y_{\text{pred}}}{\partial w} = x$$

Adım 3: Zincir Kuralı:

Şimdi, MSE'nin w'ye göre türevini bulmak için zincir kuralını uygulayalım:

$$\frac{\partial \text{MSE}}{\partial w} = \frac{\partial \text{MSE}}{\partial y_{\text{pred}}} \cdot \frac{\partial y_{\text{pred}}}{\partial w}$$

Değerleri yerine koyarak:

$$\frac{\partial \text{MSE}}{\partial w} = 2 \cdot (y_{\text{pred}} - y_{\text{true}}) \cdot x$$

Sonuç:

Bu yüzden, w'ye göre türev şu şekilde ifade edilir:

$$\frac{\partial \text{MSE}}{\partial w} = 2 \cdot x \cdot (y_{\text{pred}} - y_{\text{true}})$$

Aynı mantıkla b'nin türevini hesapladığımızda, x yerine 1 çarpanıyla türev alınır:

$$\frac{\partial \text{MSE}}{\partial b} = 2 \cdot (y_{\text{pred}} - y_{\text{true}})$$

d) Ağırlık ve Bias Güncellemesi:

Güncelleme işlemi, gradient descent (gradyan inişi) algoritması kullanılarak yapılır. Her bir parametre, aşağıdaki formül ile güncellenir:

$$\text{Yeni Değer} = \text{Eski Değer} - \eta \cdot \text{Gradyan}$$

Burada:

- η: Öğrenme oranı,
- Gradyan: Hesaplanan türev değeridir.

1. Ağırlık (w) Güncellemesi:

Ağırlık (w) için formülü uygulayalım:

$$w = w - \eta \cdot \frac{\partial \text{MSE}}{\partial w}$$

Değerleri yerine koyalım:

$$w = 1 - 0.1 \cdot (-12) = 1 + 1.2 = 2.2$$

2. Bias (b) Güncellemesi:

Bias (b) için formülü uygulayalım:

$$b = b - \eta \cdot \frac{\partial \text{MSE}}{\partial b}$$

Değerleri yerine koyalım:

$$b = 0 - 0.1 \cdot (-6) = 0 + 0.6 = 0.6$$

Kısmi türev, çok değişkenli bir fonksiyonda bir değişkenin ilgili fonksiyonu ne kadar etkilediğini yani **sonucuna ne kadar etki ettiğini hesaplamak için kullanılmaktadır**. Bu durumda **kısmi türev**, her bir ağırlık ve bias değerlerinin **hata değerini nasıl etkilediğinin** elde edilmesine olanak sağlamaktadır.

Geri yayılım algoritması ise yapısı itibariyle ağdaki bir **nöronun genel olarak ağın ileri besleme sonucuna elde edilen değerin gerçek değerden farkına** yani hata oranına ne kadar etki ettiğini hesaplar ve bu etkiye göre nöronun ağırlık değerini günceller (arttırır veya azaltır).

Hata değerimiz L

Ağdaki ağırlıkları ağırlık değerleri W1,W2,W3,W4,W5,W6

Ağdaki **sapma** değerleri B1,B2,B3 olsun.

Ağırlık ve sapma değerlerinin her birinin yapay sinir ağına etkisi aşağıdaki formül de gösterilmektedir.

$$\frac{\partial L}{\partial w_1}, \frac{\partial L}{\partial w_2}, \frac{\partial L}{\partial w_3}, \frac{\partial L}{\partial w_4}, \frac{\partial L}{\partial w_5}, \frac{\partial L}{\partial w_6}, \frac{\partial L}{\partial b_1}, \frac{\partial L}{\partial b_2}, \frac{\partial L}{\partial b_3}$$

Bu kısmi türev ise türevin zincir kuralı ile elde edilmektedir

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial Y_{\text{Tahmin}}} \cdot \frac{\partial Y_{\text{Tahmin}}}{\partial w_1}$$

Model içerisinde **farklı ağırlık vektörleri** aynı başarımı verebilmektedir. Örneğin f = x.w + b şeklinde tanımlı **cost fonksiyonu** olan bir modelde x = [1,1,1,1] matrisinin girdi vektörümüz olduğunu varsayalım. w1 = [1,0,0,0], w2 = [0.25,0.25,0.25,0.25] şeklide iki adette de ağırlık vektörümüz olsun . Bu durumda **f1 = w1.x = 1 ve f2 = w2.x = 1** olarak hesaplanır ve her iki matris çarpımlarının sonucu aynı olur. **Yani x girdi değeri için aynı cost değeri üreten birbirinden farklı birden fazla ağırlık vektörü olabilir. Bu durumda hangi ağırlık vektörü tercih edilmelidir?**

https://www.linkedin.com/pulse/derin-%C3%B6%C4%9Frenme-uygulamlar%C4%B1nda-ba%C5%9Far%C4%B1m-iyile%C5%9Ftirme-necmettin-%C3%A7arkac%C4%B1/?originalSubdomain=tr

Küçük ağırlıkların tercih edilmesinin birkaç önemli sebebi vardır:

1. Aktivasyon Fonksiyonlarının Doyma Problemi (Saturation)

•Büyük ağırlıklar, aktivasyon fonksiyonlarının (örneğin Sigmoid veya Tanh) **doyma bölgesine** girmesine neden olabilir.

•**Doyma bölgesine giren nöronlar**, gradyan sıfıra yakın olacağı için **geri yayılım (backpropagation) sırasında güncellenmez**.

•Bu durum özellikle **vanishing gradient (kaybolan gradyan) problemini** artırır.

2. Optimizasyonun Kolaylaşması

•Büyük ağırlıklar, **hata fonksiyonunun eğimini artırarak** optimizasyon sürecini kararsız hale getirebilir.

•Küçük ağırlıklar, **daha düzgün ve kontrollü bir gradyan akışı** sağlar.

•Böylece, optimizasyon algoritmaları daha stabil çalışır ve daha hızlı yakınsama görülür.

3. Aşırı Öğrenmenin (Overfitting) Önlenmesi

•Büyük ağırlıklar, modelin **çok karmaşık karar sınırları** oluşturmasına yol açar.

•Bu da eğitim verisine fazla uyan ama genel performansı kötü olan **aşırı öğrenmiş** bir modelle sonuçlanabilir.

•Küçük ağırlıklar, **modelin daha genel ve esnek olmasını** sağlayarak genelleştirmeyi artırır.

4. Simetriyi Kırmak (Symmetry Breaking)

•Eğer ağırlıklar çok büyük veya aynı olursa, **tüm nöronlar aynı çıktıyı üretebilir** ve öğrenme süreci etkisiz hale gelir.

•Küçük rastgele ağırlıklarla başlamak, modelin farklı özellikleri öğrenmesine yardımcı olur.

5. Sayısal Kararlılık

•Büyük ağırlıklar, **overflow (taşma) veya underflow (küçük sayıların sıfıra yuvarlanması) gibi sayısal hatalara neden olabilir**.

•Küçük ağırlıklar, sayısal kararlılığı artırarak hataları minimize eder.

Model iyileştirmede çalışma mantığı küçük ağırlıklı değerlere (weight) sahip modelin büyük ağırlıklara sahip modele göre daha basit, yorumlanabilir olduğu varsayımına dayanır. Bu nedenle büyük değer üreten ağırlıklar yerine düşük değerli ağırlıkları tercih etmek başarımı artıracak, ezberlemeyi önleyecektir. Bunun içinde cezalandırma (penalth) fonksiyonları kullanılarak modelin ağırlıkları sıfır veya sıfıra yakın değerlerde tutulmaya çalışılır. Cezalandırma (penalthy) işlemi için yaygın olarak **L1 ve L2** fonksiyonları kullanılır.

https://www.linkedin.com/pulse/derin-%C3%B6%C4%9Frenme-uygulamlar%C4%B1nda-ba%C5%9Far%C4%B1m-iyile%C5%9Ftirme-necmettin-%C3%A7arkac%C4%B1/?originalSubdomain=tr

Yapay sinir ağlarında **L1** ve **L2 cezalandırma metrikleri** (penalty terms), ağırlıkların büyüklüğünü sınırlayarak modelin **genelleme yeteneğini** artırmak için kullanılan **düzenlileştirme** (regularization) yöntemleridir. Bu yöntemler, aşırı öğrenmeyi (overfitting) önlemek için uygulanır.

L1 regularizasyonu, kayıp fonksiyonuna (loss function) bir regularizasyon terimi ekleyerek modelin ağırlıklarını cezalandırır. Regularizasyon terimi, modelin parametrelerinin (ağırlıklarının) L1 normu (mutlak değerlerinin toplamı) ile çarpılmasıyla elde edilir:

1. L1 Cezalandırma (L1 Regularization)

L1 düzenlileştirme, ağırlıkların **mutlak değerlerinin toplamını** ceza terimi olarak ekler. Bu, modelin ağırlıklarının bazılarını sıfıra indirme eğiliminde olmasını sağlar.

Formül:

Eğitim sırasında optimizasyon hedefi:

$$\text{Loss}_{L1} = \text{Loss}_{\text{orijinal}} + \lambda \sum_i |w_i|$$

- $\text{Loss}_{\text{orijinal}}$: Modelin hata fonksiyonu (örneğin, MSE veya BCE).
- λ: Düzenlileştirme katsayısı; cezalandırmanın ne kadar güçlü olduğunu belirler.
- $|w_i|$: Ağırlıkların mutlak değerleri.

Etkileri:

- **Ağırlıkların sıfıra yaklaştırılması:** L1 düzenlileştirme, gereksiz ağırlıkları sıfıra indirir ve modelin daha **seyrek (sparse)** hale gelmesini sağlar.
- **Özellik seçimi:** L1, özellikle yüksek boyutlu veri kümelerinde özellik seçiminde faydalıdır çünkü bazı ağırlıkları sıfır yapar ve bunlara karşılık gelen girişler modelden tamamen çıkarılır.

L2 regularizasyonu (L2 normu veya Ridge regularizasyonu olarak da bilinir), aşırı uyumu (overfitting) önlemek ve modelin genelleştirme yeteneğini artırmak için kullanılan bir regularizasyon tekniğidir. L2 regularizasyonu, modelin ağırlıklarının büyüklüğünü kontrol etmek için kayıp fonksiyonuna bir regularizasyon terimi ekler.

2. L2 Cezalandırma (L2 Regularization)

L2 düzenlileştirme, ağırlıkların **karesel toplamını** ceza terimi olarak ekler. Bu, modelin tüm ağırlıklarının küçük (ama sıfırdan farklı) olmasını sağlar.

Formül:

Eğitim sırasında optimizasyon hedefi:

$$\text{Loss}_{L2} = \text{Loss}_{\text{orijinal}} + \lambda \sum_i w_i^2$$

- w_i^2: Ağırlıkların kareleri.

Etkileri:

- **Ağırlıkları küçültür:** L2, tüm ağırlıkları küçük değerler almaya iter, ancak sıfıra indirmez.
- **Aşırı öğrenmeyi önler:** Ağırlıkların çok büyük değerlere ulaşmasını engelleyerek modelin daha iyi genelleme yapmasını sağlar.

3. L1 ve L2'nin Kombinasyonu: Elastic Net

Bazı durumlarda L1 ve L2 düzenlileştirmelerini birleştirerek her iki yöntemin avantajlarını kullanmak mümkündür. Bu yönteme **Elastic Net** denir.

Formül:

$$\text{Loss}_{\text{ElasticNet}} = \text{Loss}_{\text{orijinal}} + \lambda_1 \sum_i |w_i| + \lambda_2 \sum_i w_i^2$$

- Hem seyrek (sparse) bir model hem de düzgün bir genelleme sağlar.

Öğrenme Oranı (Learning Rate)

Öğrenme oranı **geri besleme algoritmasında** kullanılan hatanın sonucu etkileme oranıdır. Bir her X-Y ikilisi için beklenen sonuç ile tahmin edilen çıktı arasındaki fark kullanılarak hata değeri hesaplanır. Hesaplanan **hata** başlangıçta **rastgele olarak atanan ağırlık ve sapma değerlerini güncellemek** için **geri besleme algoritmasında kullanılır.**

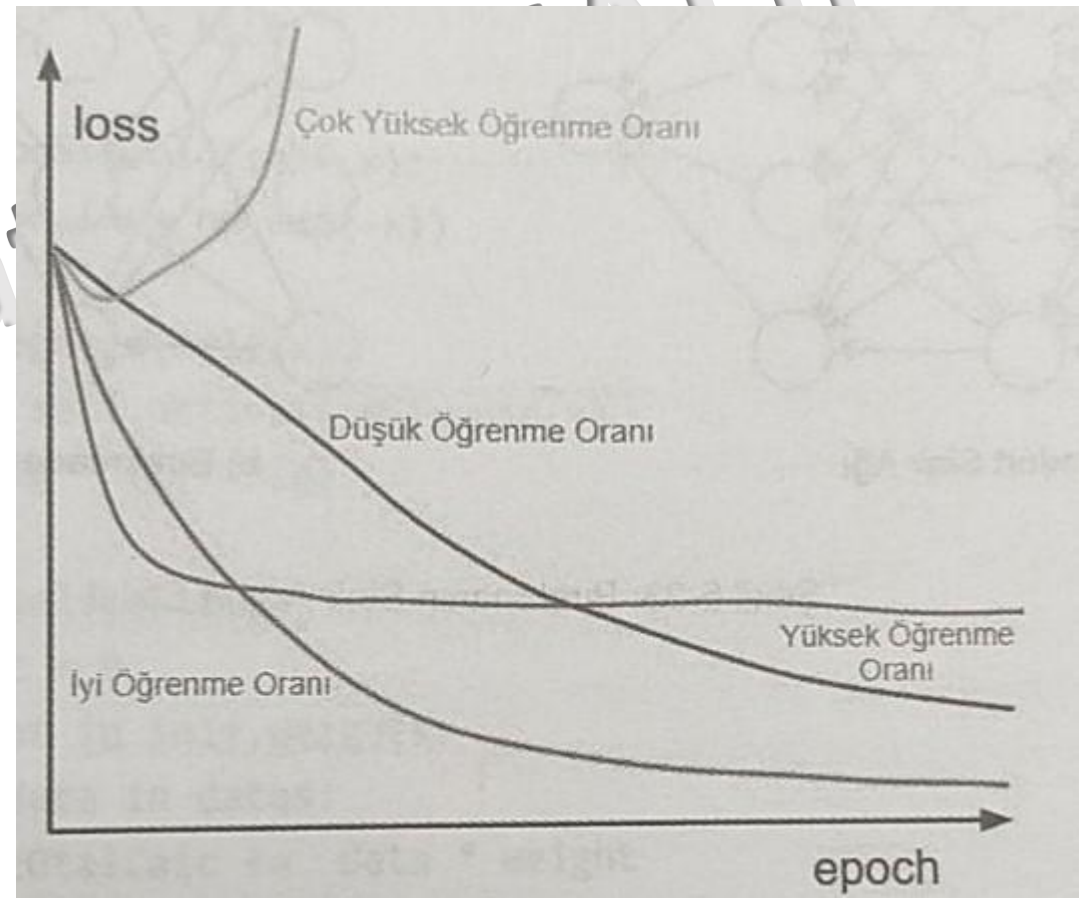
Bu işlem; tüm eğitim setinde yapılan hata, **maksimum tekrar sayısına ulaşılana kadar tekrarlanır.**

Öğrenme oranı, yani learning rate, modele sahip olduğu ağırlıklar güncellendiğinde **tahmini hataya göre modelin ne kadar değiştirileceğini** kontrol eden hiper parametredir.

Hiperparametreler modeli geliştiren kişi tarafından ve diğer tüm etkenlerden **bağımsız bir öğrenme parametresidir.** Öğrenme oranının sezgisel bir değer alması gerektiğinden ötürü doğru değerleri seçmek oldukça zordur. Zira öğrenme oranına çok küçük bir değer verilirse **uzun bir eğitim süreci** ile karşı karşıya kalınabilir. Bunun yanında eğitim hiçbir zaman tam olarak gerçekleşmeyebilir. Bu senaryonun tersi de mümkündür. Çok büyük bir öğrenme oranı, geri yayılım sonucunda nöron ağırlıklarını çok şiddetli değiştirerek tam olarak ulaşılması gereken hassas ağırlıkları asla ulaşamayabilir. Bu durumda istenen başarıyla bir öğrenme süreci gerçekleşmeyecektir.

UYARI

Öğrenme oranı bir **hiper parametredir**. Yani diğer parametrelere göre değeri **kestirilemez**. Tamamen bağımsız bir değişkendir. Öğrenme oranında değişiklikler yapılarak **en optimum** öğrenme oranı elde edilmelidir. En iyi öğrenme oranını bulmak, deneme yanılma ile sürdürülen bir süreçtir



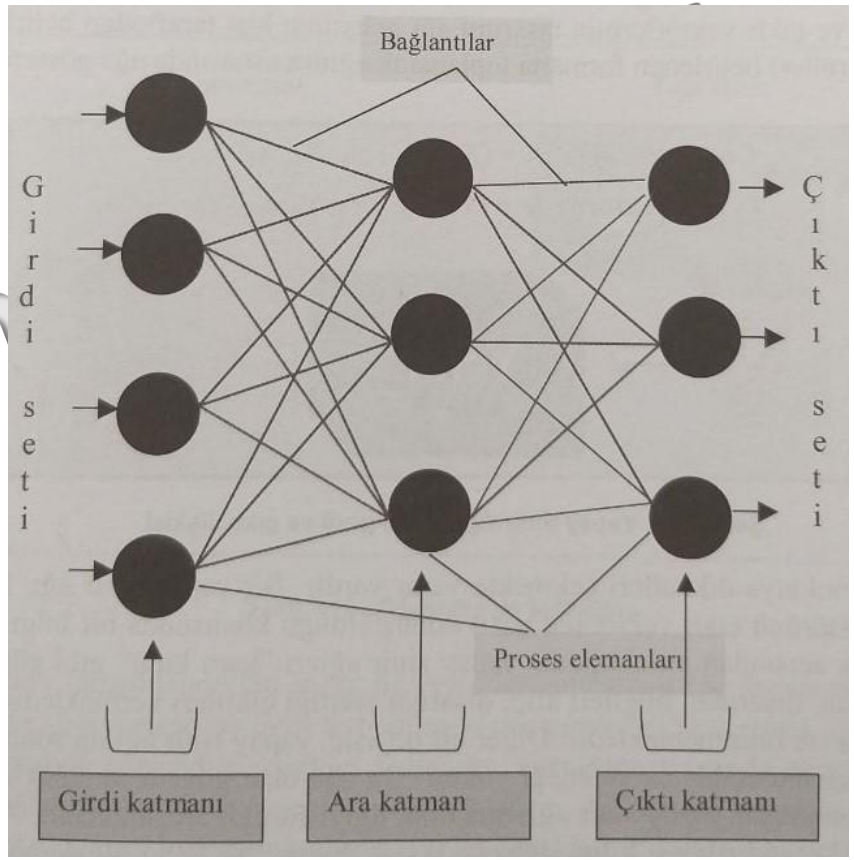
Tekrar Sayısı (Epoch)

Epoch değeri, tüm eğitim verilerinin **yapay sinir ağına kaç defa geçirileceğini** ifade etmektedir. Bu nedenle model eğitim aşamasında tekrar sayısının belirlenmesi oldukça önemli bir aşamadır.

Tekrar sayısının **fazla olması ağın daha başarılı olmasını** sağlarken tekrar sayısının **çok yüksek olması ağın aşırı uyumlu** hale gelmesine, yani ezberci davranmasına neden olmaktadır. Ağın aşırı uyumlu hale gelmesine overfitting yani aşırı öğrenme denilmektedir. Aynı şekilde tekrar sayısının az olması ağın eksik öğrenmesine neden olabilir bu durum ise underfitting olarak bilinmektedir

ÖZETLE, ANN yani sinir ağları **en az bir adet gizli katmanı** olan bir yapı sunar. Bu sinir ağı yapısında initialize etmem ve güncellemem gereken parametrelerin başında weight ve bias değerleri gelmektedir.

UNUTULMAMALIDIR Kİ Weight/bias değerlerini 0 olarak initialize edersem eğitim sürecinde belirli bir iterasyondan sonra gradient descent'in farklılık oluşturmayacak ve aynı hesaplamaları yapmasından dolayı **öğrenememe** problemine neden olacaktır. Bu nedenle hedefe yönelik rastgele ve küçük ağırlıklar ile ağın initilize edilmesi eğitime çeşitlilik katarak ve farklı ayrıntıları öğrenebilmesini sağlayacaktır.



Yapay sinir ağları tıpkı makine öğrenmesi yönteminde olduğu gibi denetimli ve denetimsiz öğrenmeyi desteklemektedir.

Bununla birlikte makine öğrenmesinden farklı olarak feature extraction ve feature selection işlemleri otomatik yapılmaktadır. Otomatik olarak **çıkarılan** özniteliklerin **anlamlı** olup olmaması **eğitim ve sınıflandırma sonucunda** görülmektedir. Bize düşen görev mimarinin başarılı bir şekilde tasarlanmasıdır. Yani bizler **inşa ettiğimiz model ve seçtiğimiz parametreler** vasıtasıyla **özellik mühendisliği** yapıyoruz.

Bu doğrultuda **katman sayısı**, **katmanlardaki nöron sayısı**, **kullanılan optimizasyon algoritmaları** ve **öğrenme oranı** parametrelerini efektif olarak seçmek önem arz etmektedir. Çünkü parametreler, veriler ve kullanılan donanıma bağlı olarak titizlikle ayarlanması gereken yapılardır. Kısaca veriden öğrenen yapay sinir ağı yapılarının/derin **öğrenme modellerinin** tasarlanırken **nasıl olması gerektiği** var olan koşullar dikkate alınarak (problemin amacı, veri kümesi, donanım) modeli tasarlayan kişiye bırakılmıştır. Yani hiçbir zaman **net bir şey söyleyemeyeceğiz** farklı durumlara bağlı olarak **hiperparametre** kavramı seçilmelidir.

Kaynaklar

Prof. Dr. Ercan Öztemel, Yapay Sinir Ağları, Papatya Yayıncılık, 5. Basım Kasın 2020

Dr. Öğr. Üyesi Atınç Yılmaz, Öğr. Gör. Umut Kaya,Derin Öğrenme, Kodlab Yayınları, 3 Bası Ocak 2021

https://chatgpt.com/

https://www.blackbox.ai/