Perceptron ve Lineer Fonksiyonlar

- Lineer fonksiyonlar, y = W.x+ b şeklinde tanımlanan fonksiyonlardır.

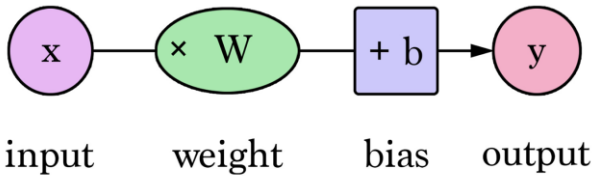
- Bu fonksiyonlarda y değeri x'in değerine bağlı olduğu için bağımlı değişken, x değeri ise herhangi bir girdi olabileceği için bağımsız değişken olarak tanımlanır.

- W ve b değerleri ise fonksiyonun parametreleri olarak tanımlanmaktadır.

$$y = Wx + b$$

yapay sinir ağı modelinin en küçük parçası olan perceptron'da bu şekilde y = W.x+b lineer fonksiyonuyla tanımlanır.
W değeri ağırlık parametresi
x değeri girdi
b değeri bias
y değeride ağın çıktısı

Burada x girdi değerimiz, örneğin kedi resimlerini tanıyorsak kedi resmine ait matrisi, y ise bu resmin kediye ne kadar benzediğine dair skoru verir. Parametrelerimiz olan W ağırlık ve b bias değerlerini bu çıktı skorunu iyileştirmek için kullanılır.



Bias (b)

- W.x+b fonsiyonunda W ağırlık(ları) x'e bağlı olarak hareket etmektedir.
- W.x çarpımının sonucu her durumda x in değerinden etkilenmektedir. Bu nedenle W.x matrisinin çarpım sonucu modeli fit edecek doğruyu sadece x/y doğrultusunda hareket ettirebilmektedir.
- Bias b değişkeni elde edilen doğrunun ötelenmesinde (shift) kullanılır.
- Bağımlı W parametresi ile x/y doğrusunda hareket sağlanırken, bağımsız b parametresi ile x/y doğrultusunda oluşturulan doğrunun aşağı yukarı hareketi sağlanır;
- yani bias b değeri doğruyu shift işlem yapabilme yeteneği sayesinde problem fit edecek en uygun doğru oluşturulabilir.
- Bias değeri x girdi değeri ile herhangi bir etkileşime girmeden direk çıktı skor değerine etki etmektedir.
- Bias değerleri genelde belli aralıktaki rastgele değerlerden oluşturulur. Bununla birlikte sabit olarak belirlendiği durumlarda vardır. Rastgele oluşturulduğunda belli bir dağılıma göre hareket etmesi genelde gözetilir.

Weight

- Ağırlık değerler başlangıçta tamamen rastgele ya da belli bir aralıkta belli bir dağılama göre rastgele de belirlenebilir. Örneğin gaussian dağılımına göre başlatılabilir, veya sabit sayılarla da başlatılabilir.
- Daha önce eğitilmiş bir modelin ağırlık değerleri kullanılarak da model başlatılabilir.
- Sabit değer veya 0 ile başlatılması bazı problemlere sebep olabilir.

1. Modelde kullanılan tüm ağırlık değerleri aynı olduğu için tüm bağlar aynı değeri alır, ağırlıkların güncellenmesi de aynı olabilir. Bu durumda öğrenme gerçekleşmez.
2. Bunun yerine genel eğilim ağırlık değerlerinin bir kısmının 0'a yakın pozitif, bir kısmının 0'a yakın negatif seçilmesi daha iyi olur.
3. Her zaman 0 yakın seçilmesi çok fazla katmana sahip ağlarda geri besleme işleminin etkisinin başlangıç katmanlarına etkisini azaltır.

- Hangi girdinin en çok katkısı var?
- Hangi girdi veya girdiler çıkış faktörünü etkiliyor?

You should know

- Dış ortamdan gelen bilgiler, girdi katmanına gelir. Bu bilgiler, ağın öğrenmesi istenen bilgilerdir.
- Bilgilerin ağa veriliş şekli eldeki veri setine göre değişiklik gösterir.
- Girdi katmanından giren her bilgi (xi) eş zamanlı olarak gizli katmana iletilir. Farklı katmanlardaki nöronlar arasındaki bağlantıların Ağırlık Değerleri (wi) vardır.
- Ağırlık değerleri nöronlar arasındaki bağlantının gücünü belirtir.

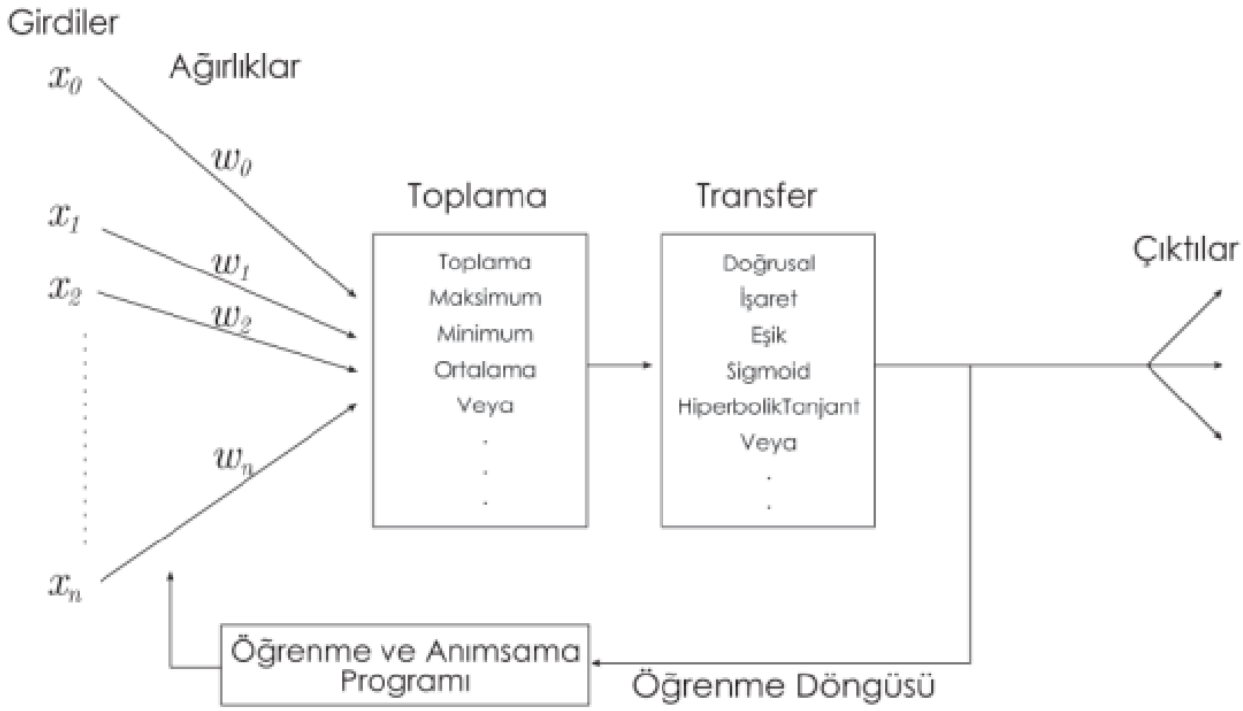
You should know1

- Başlangıç ağırlık değerleri belirlendikten sonra, girdi katmanındaki hücrelerden gelen her bilgi (xi) kendisine ait bağlantının ağırlık değeri (wi) ile çarpılıp Birleştirme(Toplama) Fonksiyonu yardımıyla gizli katmandaki her bir hücrenin toplam Net Girdi (s) değeri bulunur.
- Net girdiyi oluşturan birleştirme fonksiyonu, kullanılan ağın yapısına göre ağırlık değerleri ile çarpılmış girdi değerlerinin toplanması, ortalaması, en büyüğü veya en küçüğünün alınması gibi birleştirme işlemlerinden herhangi biri olabilir.
- Bir problem için en uygun birleştirme fonksiyonu çeşidini bulmak için herhangi bir formül yoktur. Kullanılacak birleştirme fonksiyonu genellikle deneme yanılma yoluyla bulunmaktadır. Toplama işleminin yürütüldüğü basit bir birleştirme fonksiyonu Denklemi

$$s = \sum x_i w_i$$

You should know2

- Sonraki aşamada birleştirme fonksiyonunun çıktısı Transfer Fonksiyonuna gönderilir.
- Bu fonksiyon, aldığı değeri bir algoritma ile gerçek bir çıktıya dönüştürür.
- Kullanılan transfer fonksiyonuna göre çıktı değeri genellikle [-1,1] veya [0,1] arasındadır. Genellikle doğrusal olmayan bir fonksiyondur.
- Doğrusal olmayan transfer fonksiyonlarının kullanılması yapay sinir ağlarının karmaşık ve çok farklı problemlere uygulanmasını sağlamıştır.
- Sıklıkla kullanılan transfer fonksiyonları
- Doğrusal (Linear), Adım/İşaret (Step/Signum), Eşik (Threshold), Sigmoid,
- Hiperbolik Tanjant, Lojistik vb. fonksiyonlardır.



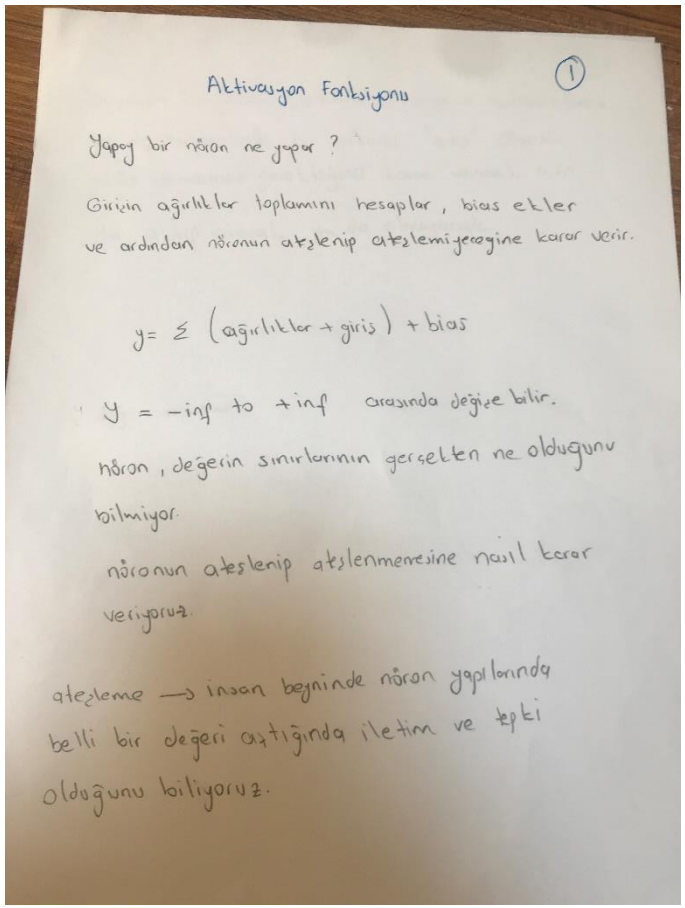
Şekil 2.6 Detaylı Bir Yapay Sinir Hücresi

Aktivasyon Fonksiyonları(Activation Function)

- Aktivasyon fonksiyonları yapay sinir ağlarının çok önemli bir özelliğidir.
- Temel olarak bir nöronun aktif olup olmayacağına karar verirler.
- Nöronun aldığı bilginin verilen bilgilerle ilgili olup olmadığı veya göz ardı edilip, edilmemesi (gerekmediğine) karar verirler.
- Muhtemel sonsuz giriş kümesine sahip işlem elemanlarından önceden belirlenmiş sınırlar içinde çıkışlar üretir.

Aktivasyon Fonksiyonları(Activation Function)

- Bir sinir ağı feature(özellik) yani giriş değeri olarak istediği değeri alabilir.
- Bir resim için bu değer 0-255 arasında bulunan değerlerdir. ,
- Bir sinyal için o zaman aralığına karşılık gelen frekans değerleri
- Herhangi bir işleme karar vermek için alacak değerlerin sonu ve sınırı yoktur.
- Çıkış katmanı değeri için bu durum böyle değildir.
- Karmaşık değerlerin bulunduğu giriş katmanı ile, çıkış değişkeni arasında işlevsel eşlemler hakkında bilgi edinmek ve anlam kazandırmak için çok önemlidir.
- ASIL AMAÇ : GİRİŞ SİNYALİNİN ÇIKIŞ SİNYALİNE DÖNÜŞTÜRÜLMESİDİR.



Aktivasyon Fonksiyonu ①

Yapay bir nöron ne yapar ?

Girişin ağırlıklar toplamını hesaplar, bias ekler ve ardından nöronun ateşlenip ateşlemeyeceğine karar verir.

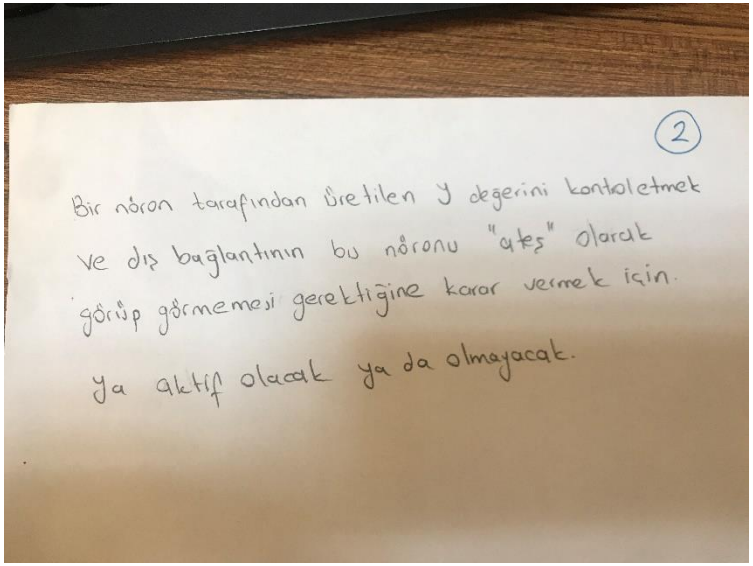
$$y = \Sigma\,(\text{ağırlıklar} * \text{giriş}) + \text{bias}$$

y = -inf to +inf arasında değişebilir.

nöron, değerin sınırlarının gerçekten ne olduğunu bilmiyor.

nöronun ateşlenip ateşlenmemesine nasıl karar veriyoruz.

ateşleme → insan beyninde nöron yapılarında belli bir değeri aştığında iletim ve tepki olduğunu biliyoruz.



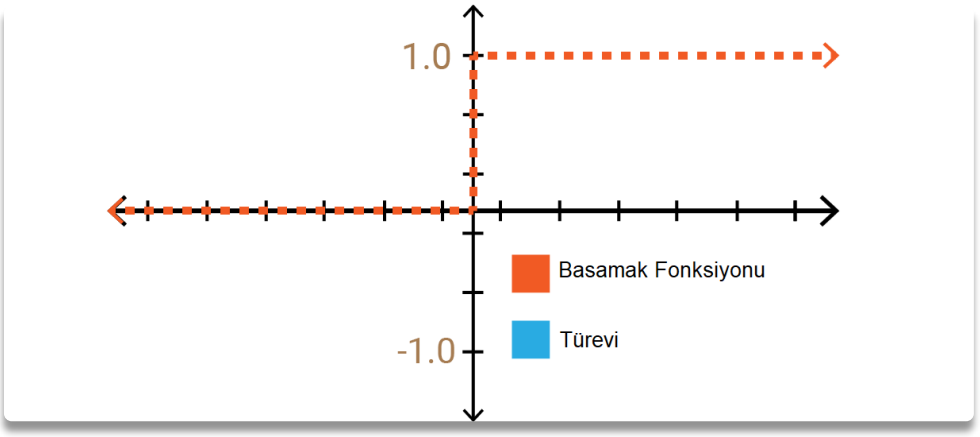
②

Bir nöron tarafından üretilen y değerini kontroletmek ve dış bağlantının bu nöronu "ateş" olarak görüp görmemesi gerektiğine karar vermek için.

ya aktif olacak ya da olmayacak.

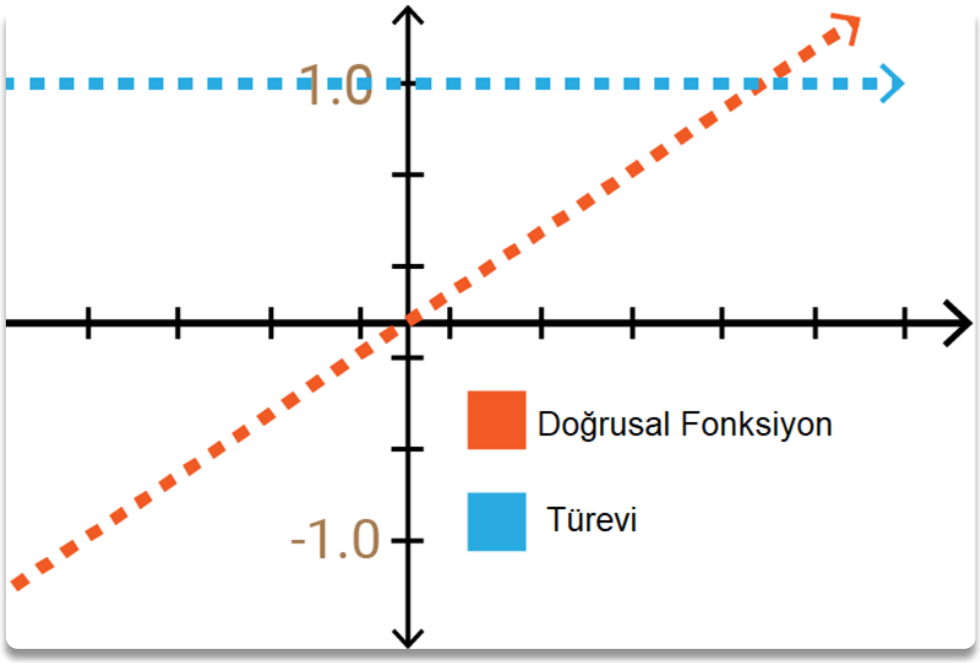
Basamak Fonksiyonu

- İkili değer alan bir fonksiyondur
- ikili sınıflandırıcı olarak kullanılır. Bu yüzden genellikle çıkış katmanlarında tercih edilir.
- Gizli katmanlarda türevi, öğrenme değeri temsil etmediği için kullanılması tavsiye edilmez



Doğrusal (Linear) Fonksiyon

- f(x) = x fonskiyonu
- Aralık : (-sonsuz to sonsuz)

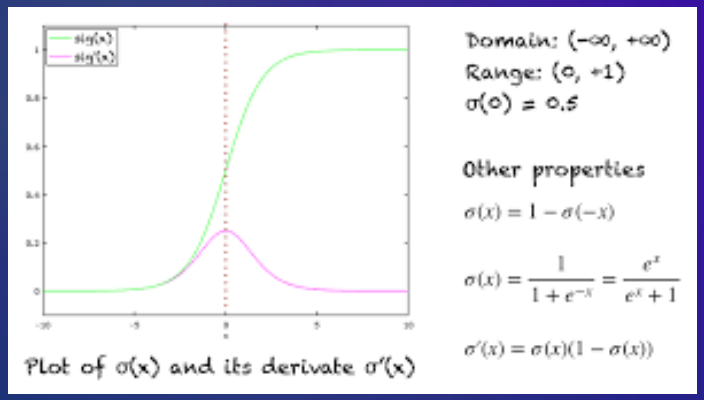


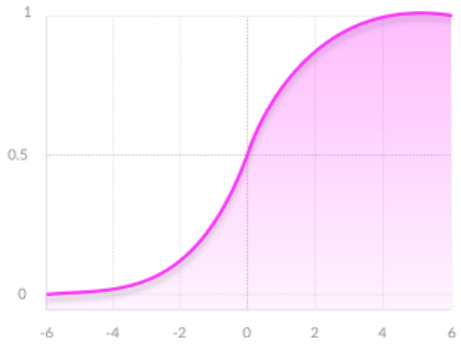
Fakat bu fonksiyonun önemli bir sorunu var! Türevinin sabit olması. **Peki neden türevine ihtiyacımız var ve sabit olmasının olumsuzluğu nedir?** geriye yayılım (backpropagation) algoritması ile öğrenme işlemini nöronlar için gerçekleştirmiş oluyorduk. Bu algoritma türev alan bir sistemden oluşuyor. A=c.x, **x**'e göre türevi alındığında **c** sonucuna erişiriz. Bu **x** ile bir ilişkinin kalmadığı anlamına gelir. Peki türevi hep sabit bir değer çıkıyorsa öğrenme işlemi gerçekleşiyordur diyebilir miyiz? Maalesef, hayır!
Bir başka sorun daha var! Tüm katmanlarda doğrusal fonksiyon kullanıldığında giriş katmanı ile çıkış katmanı arasında hep aynı doğrusal sonuca ulaşılır. **Doğrusal fonksiyonların doğrusal bir şekilde birleşimi yine bir başka doğrusal fonksiyondur.** Bu en başta birbirine bağlayabiliriz dediğimiz nöronların yani ara katmanların işlevsiz kaldığı anlamına gelir. Yani ilk amacımız olan çok katmanda çalışma yeteneğimizi kaybettik!

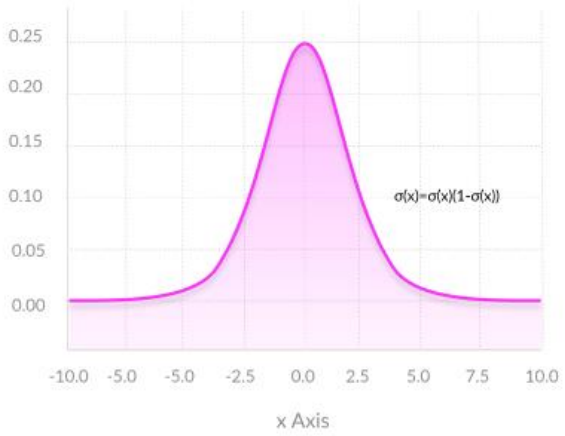
Sigmoid Fonksiyon

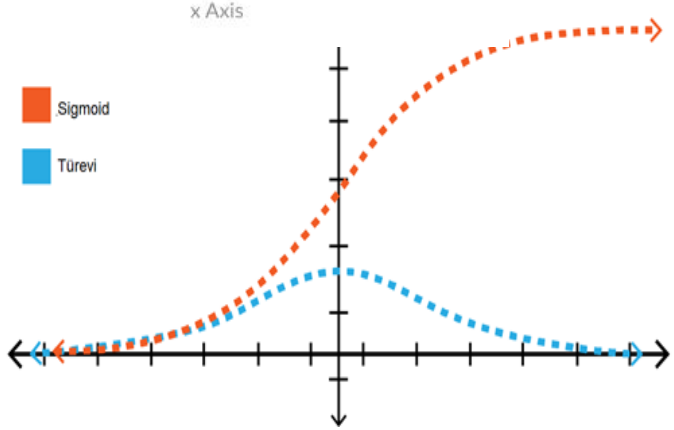
f(x) = 1 / 1 + exp(-x)

0 < çıkış < 1









y değerleri **x**'teki değişikliklere çok az tepki vermektedir. Bu ne gibi bir problem doğurur, Bu bölgelerde türev değerleri çok küçük olur ve 0'a yakınsar.

Buna **gradyanların ölmesi/kaybolması (vanishing gradient)** denir ve öğrenme olayı minimum düzeyde gerçekleşir.

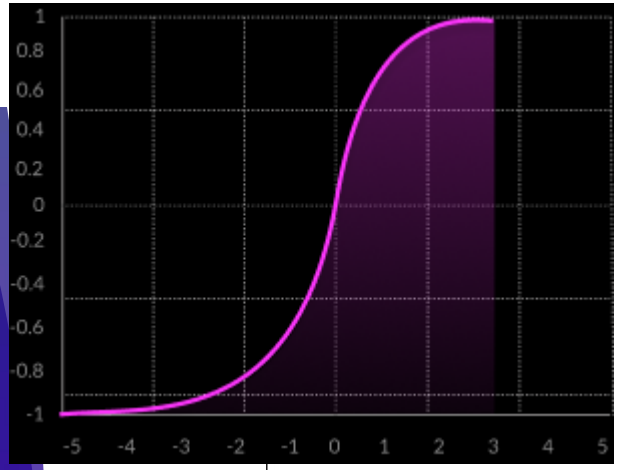
0 olursa gerçekleşmez!

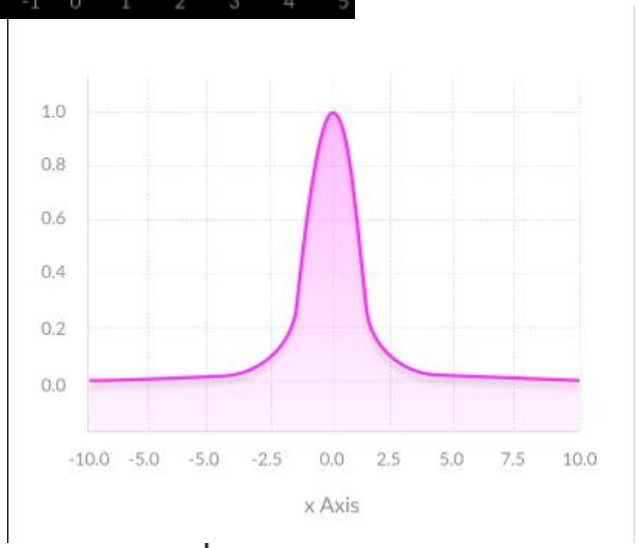
Yavaş bir öğrenme olayı gerçekleştiğinde hatayı minimize eden optimizasyon algoritması yerel (lokal) minimum değerlere takılabilir ve yapay sinir ağı modelinden alınabilecek maksimum performansı alamayız.

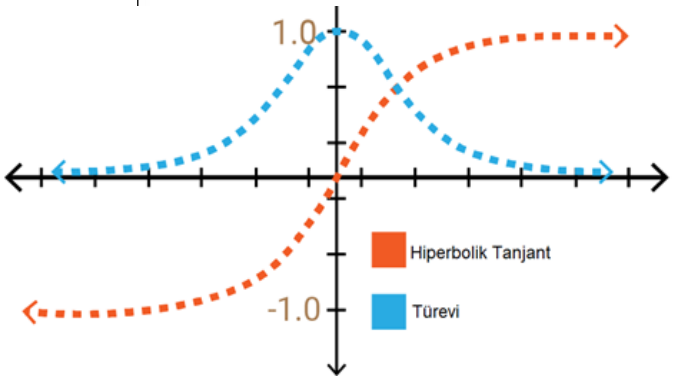
Hiperbolik Tanjant Fonksiyonu

f(x) = 1 — exp(-2x) / 1 + exp(-2x)

1 < çıkış < -1

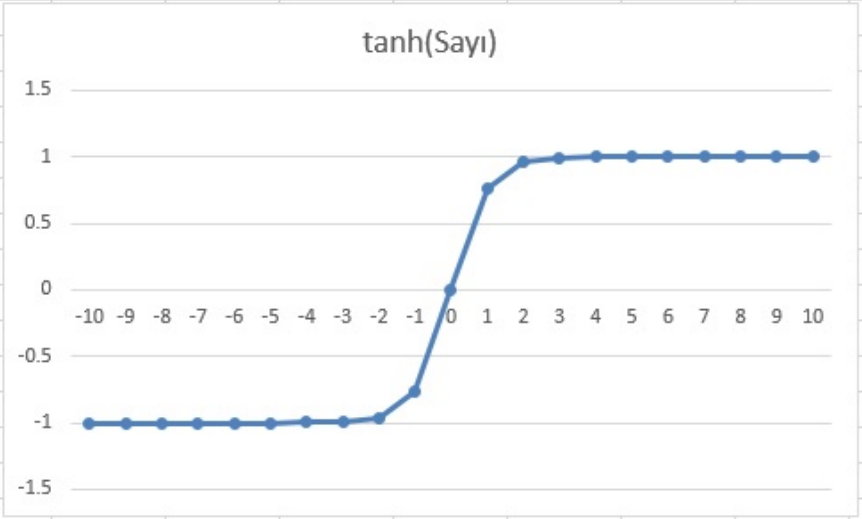


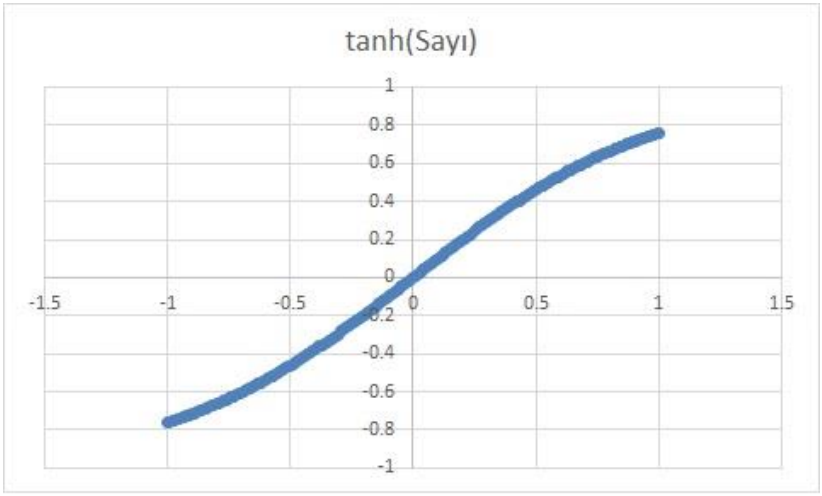




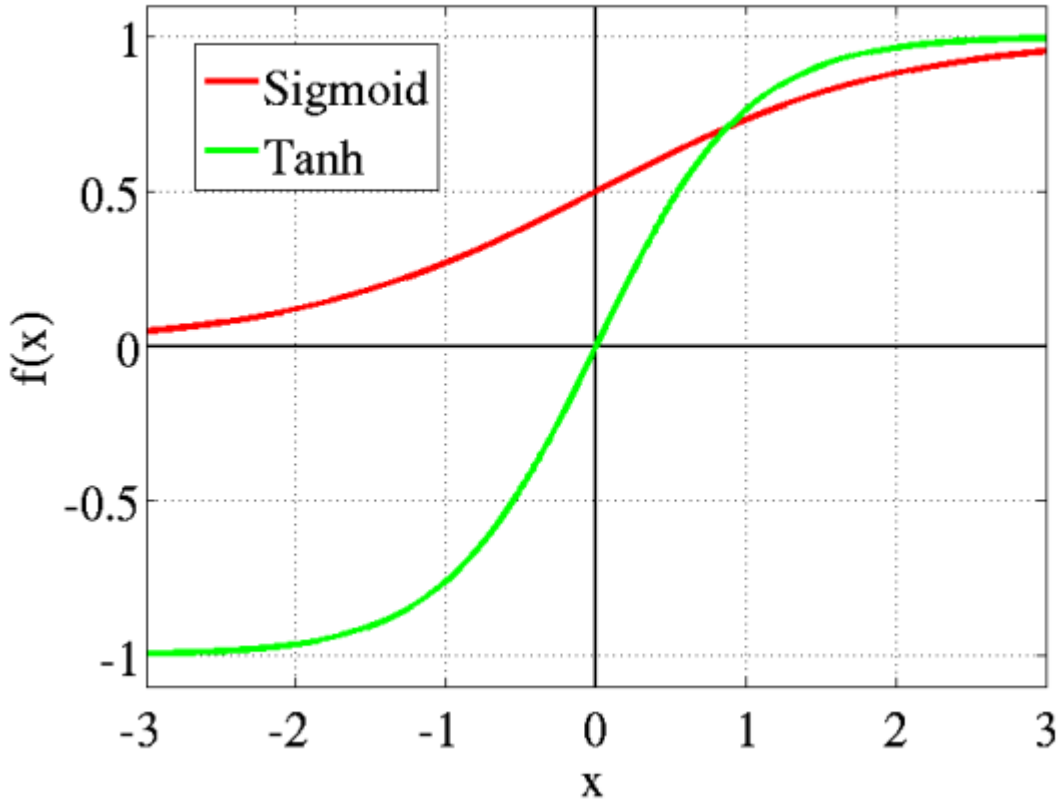
fonksiyonuna göre avantajı ise türevinin daha dik olması yani daha çok değer alabilmesidir. Bu daha hızlı öğrenme ve sınıflama işlemi için daha geniş aralığa sahip olmasından dolayı daha verimli olacağı anlamına gelmektedir. Ama yine fonksiyonun uçlarında gradyanların ölmesi problemi devam etmektedir.

Sayı	tanh(Sayı)
-10	-1
-9	-1
-8	-1
-7	-1
-6	-0.99999
-5	-0.99991
-4	-0.99933
-3	-0.99505
-2	-0.96403
-1	-0.76159
0	0
1	0.761594
2	0.964028
3	0.995055
4	0.999329
5	0.999909
6	0.999988
7	0.999998
8	1
9	1
10	1





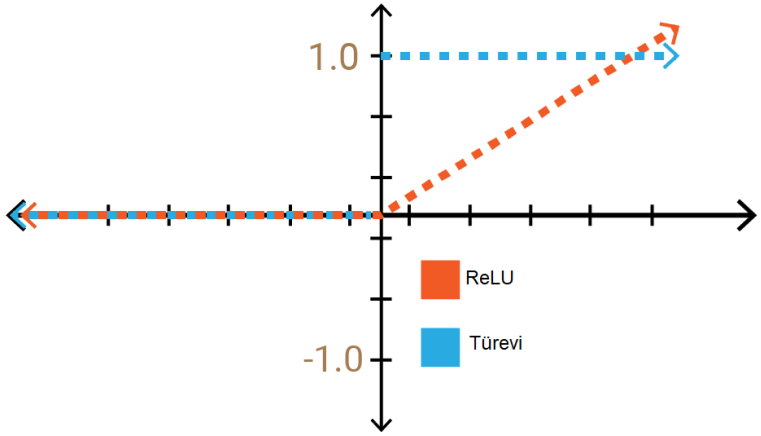
Sigmoid ve Tanh farkı



ReLU (Rectified Linear Unit) Fonksiyonu

ReLU [0, +∞) aralığında değer alır.

R(z) =max(0,z)



Sigmoid ve Hiperbolik Tanjant neredeyse tüm nöronların aynı şekilde ateşlenmesine/aktive olmasına sebep oluyordu.

Bu aktivasyon yoğun yani çok işlem gerektiriyor demektir.

Ağdaki bazı nöronların aktif olup, *aktivasyon seyrek* yani verimli bir hesaplama yükü olsun isteriz.

ReLU ile bunu sağlamış oluyoruz.

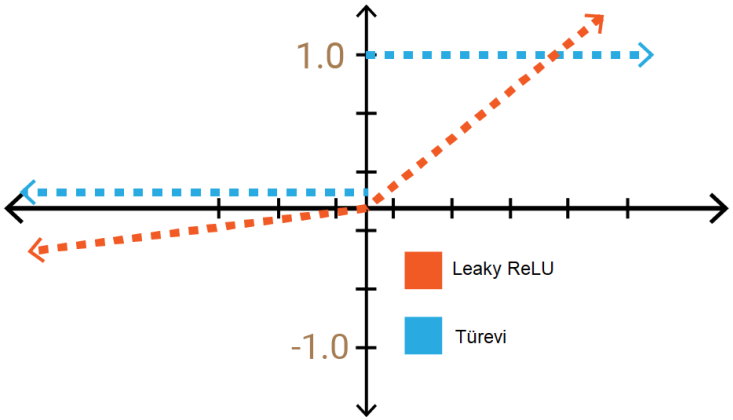
Negatif eksende 0 değerlerini alması ağın daha hızlı çalışacağı anlamına da gelmektedir. **Hesaplama yükünün sigmoid ve hiperbolik tanjant fonksiyonlarına göre az olması çok katmanlı ağlarda daha çok tercih edilmesine sebep olmuştur.**

ReLU hızı kazandıran sıfır değer bölgesinin türevinin de sıfır olması! Yani öğrenmenin o bölgede gerçekleşmiyor olması

Sızıntı (Leaky) ReLU Fonksiyonu

ReLU [0, +∞) aralığında değer alır.

R(z) =max(0,z)



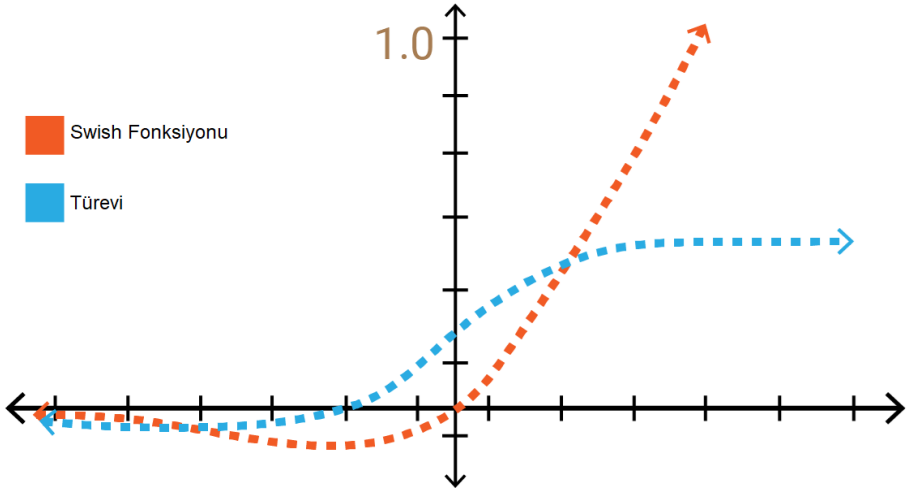
Softmax

Sigmoid fonksiyonuna çok benzer bir yapıya sahiptir. Aynı Sigmoid'te olduğu gibi sınıflayıcı olarak kullanıldığında oldukça iyi bir performans sergiler. **En önemli farkı sigmoid fonksiyonu gibi ikiden fazla sınıflamak gereken durumlarda özellikle derin öğrenme modellerinin çıkış katmanında tercih edilmektedir.**

Girdinin belirli sınıfa ait olma olasılığını 0–1 aralığında değerler üreterek belirlenmesini sağlamaktadır. Yani olasılıksal bir yorumlama gerçekleştirir.

Swish Fonksiyonu

f(x)=2x*sigmoid(beta*x)

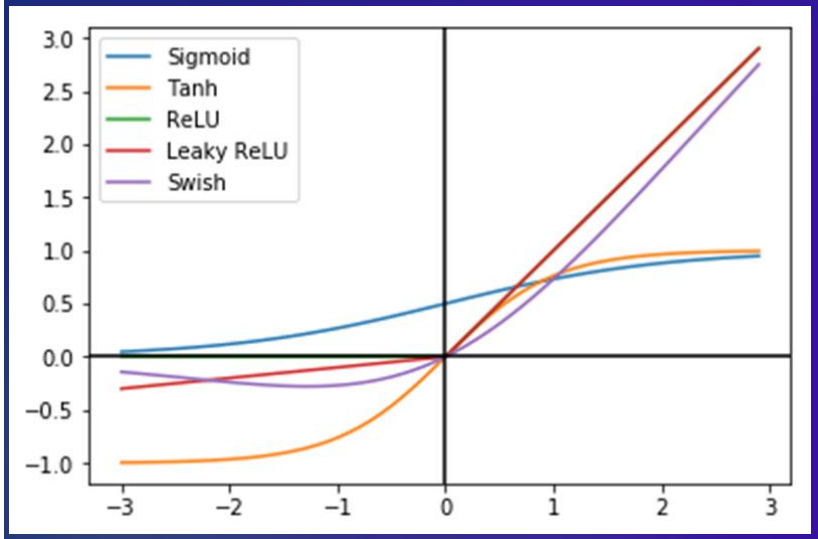


ReLU’dan en önemli farkı negatif bölgede değer alır.

Leaky ReLU’da aynı şekilde değer alıyordu, ondan farkı ne?

Swish’in negatif bölgede aldığı değerler doğrusal değildir.

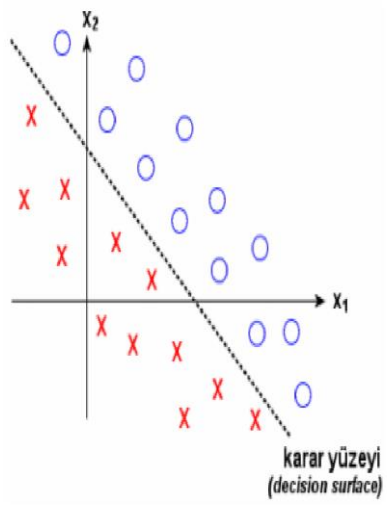
Diğer tüm aktivasyon fonksiyonları monotondur.




```
import math
import matplotlib.pyplot as plt
import numpy as np

# Aktivasyon fonksiyonlarının tanımlamalarının matematiksel olarak yapılması
# Sigmoid Fonksiyonu
def sigmoid(x):
    a = []
    for i in x:
        a.append(1/(1+math.exp(-i)))
    return a
# Hiperbolik Tanjant Fonksiyonu
def tanh(x, derivative=False):
    if (derivative == True):
        return (1 - (x ** 2))
    return np.tanh(x)
# ReLU Fonksiyonu
def re(x):
    b = []
    for i in x:
        if i<0:
            b.append(0)
        else:
            b.append(i)
    return b
# Leaky ReLU Fonksiyonu
def lr(x):
    b = []
    for i in x:
        if i<0:
            b.append(i/10)
        else:
            b.append(i)
    return b

# Grafik için oluşturulacak aralıkların belirlenmesi
x = np.arange(-3., 3., 0.1)
sig = sigmoid(x)
tanh = tanh(x)
relu = re(x)
leaky_relu = lr(x)
swish = sig*x

#Fonksiyonların ekrana çizilmesi ve gösterilmesi
line_1, = plt.plot(x,sig, label='Sigmoid')
line_2, = plt.plot(x,tanh, label='Tanh')
line_3, = plt.plot(x,relu, label='ReLU')
line_4, = plt.plot(x,leaky_relu, label='Leaky ReLU')
line_5, = plt.plot(x,swish, label='Swish')
plt.legend(handles=[line_1, line_2, line_3, line_4, line_5])
plt.axhline(y=0, color='k')
plt.axvline(x=0, color='k')
```

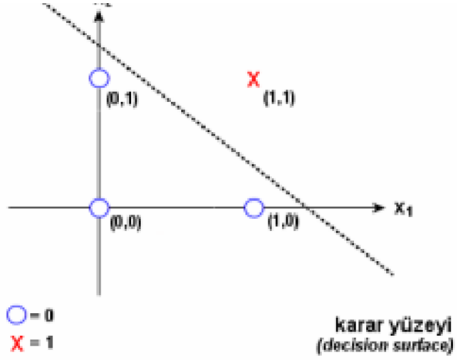
AKTİVASYON FONKSİYON	DENKLEM	ARALIK
Doğrusal Fonksiyon	$f(x) = x$	$(-\infty, \infty)$
Basamak Fonksiyonu	$f(x) = \begin{cases} 0 \; için \; x < 0 \\ 1 \; için \; x \geq 0 \end{cases}$	$\{0, 1\}$
Sigmoid Fonksiyon	$f(x) = \sigma(x) = \frac{1}{1 + e^{-x}}$	$(0, 1)$
Hiperbolik Tanjant Fonksiyonu	$f(x) = \tanh(x) = \frac{(e^{x} - e^{-x})}{(e^{x} + e^{-x})}$	$(-1, 1)$
ReLU	$f(x) = \begin{cases} 0 \; için \; x < 0 \\ x \; için \; x \geq 0 \end{cases}$	$[0, \infty)$
Leaky (Sızıntı) ReLU	$f(x) = \begin{cases} 0.01 \; için \; x < 0 \\ x \; için \; x \geq 0 \end{cases}$	$(-\infty, \infty)$
Swish Fonksiyonu	$f(x) = 2x\sigma(\beta x) = \begin{cases} \beta = 0 \; için \; f(x) = x \\ \beta \to \infty \; için \; f(x) = 2\max(0, x) \end{cases}$	$(-\infty, \infty)$

İki-boyutlu bir öklid uzayında birisine “ χ ” yada “o ” “ χ ” ve “o ” sembolleri, tek bir doğru ile izole edilebilmiştir. Bu sebeple, “ χ ” ve “o ” sembolleri lineer ayrıştırılabilirdir (linearly separable)



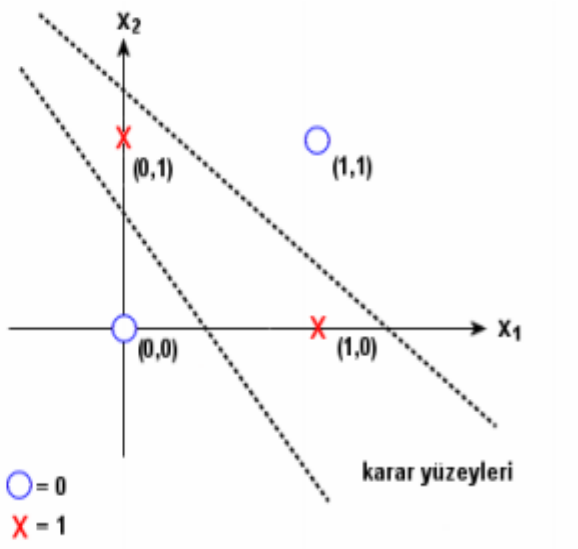
- lojik-VE fonksyonu lineer ayrıştırılabilirdir. Çünkü tek bir doğru ile 0 ve 1 çıkışları izole edilebilmektedir.

X1	X2	VE
0	0	0
0	1	0
1	0	0
1	1	1



lojik-ÖZELVEYA (XOR) fonksyonu lineer ayrıştırılamamaktadır.
0 ve 1
çıkışlarını izole edebilmek için tek bir doğru yeterli değildir.

X1	X2	Özel VEYA
0	0	0
0	1	1
1	0	1
1	1	0



XOR problemini bu kadar özel kılan, en basit non-lineer problem oluşudur. XOR probleminin, lineer yaklaşımlar ile çözülmesi mümkün değildir

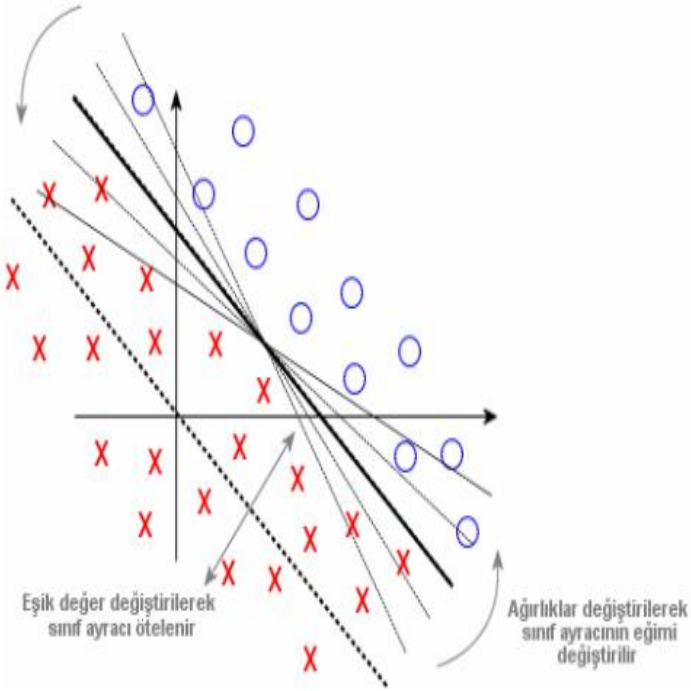
İki sınıfı birbirinden ayırmak için kullanılan yüzey, sınıf ayracı olarak tanımlanır.

tek katmanlı yapay sinir ağı 2-girişli

Bu durumda, sınıf ayracı iki-boyutlu öklid uzayında bir doğru belirtir.

Eşik değerin değiştirilmesi, sınıf ayracını ötelerken, ağırlıkların değiştirilmesi eğimini etkiler.

Eğitim sırasında eşik değer ve ağırlıklar değiştirilerek, doğrunun, sınıfları ayıracak şekilde konumlandırılması sağlanır….



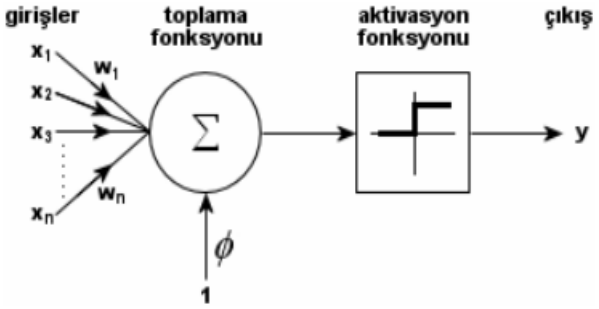
1958 yılında psikolog Frank Rosenblatt tarafından "zeki sistemlerin temel özelliklerinden bazılarını simüle etmek" amacıyla geliştirilen perceptron modeli, bir sinir hücresinin birden fazla girdiyi alarak bir çıktı üretmesi prensibine dayanır.

Ağın çıktısı, girdi değerlerinin ağırlıklı toplamının bir eşik değer ile karşılaştırılması sonucu elde edilir.
Toplam eşikten eşit veya büyük ise çıktı değeri 1, küçük ise 0 seçilir.

En Basit Tek Katmanlı Algılayıcı

Giriş ve hedef vektörleri : Giriş değerlerinin sayısı için bir kısıtlama yoktur.

Giriş vektörü, negatif yada pozitif herhangi bir değeri içerebilir. Giriş değerlerinin 1'den büyük yada küçük olması performansı etkilemez. Perceptronun üretmesini istediğimiz çıkışlara hedef değer denir.



Giriş vektörü için söylenenler, hedef vektör için geçerli değildir.
Kullanılan aktivasyon fonksiyonu sebebi ile hedef vektör, sadece 0 yada 1 değerlerini içerebilir.
Toplama fonksiyonu : Ağırlıklı toplam fonksiyonu kullanılır.
Yüzey ayracının ötelenmesi gereken durumlarda eşik değer ilave edilebilir.

$$NET = \left(\sum_{i=1}^{n} w_i \cdot x_i \right) + \phi$$

Aktivasyon fonksyonu : Perceptron, sınıflandırma amacı ile geliştirildiğinden, farklı sınıfları temsil edecek değerler üretilmelidir. Sınıflandırma işleminde, klasik küme kuramındaki ait olma yada ait olmama durumları incelenir. Bulanık mantık da olduğu gibi bir aitlik derecesi vermek mümkün olmamaktadır. Bu amaçla aktivasyon fonksyonu olarak basamak fonksiyon kullanılır.

$$y = f(NET) = \begin{cases} 1 & Eger \quad NET > \phi \\ 0 & aksi \quad takdirde \end{cases}$$

HEBB öğrenme kuralı

Hebb kuralı, "**Birlikte ateşleyen nöronlar birlikte bağlanır**" (ingilizcesi: *Cells that fire together, wire together*) ifadesiyle özetlenebilir. Eğer bir nöron başka bir nöronu sürekli olarak etkinleştiriyorsa, aralarındaki sinaptik bağlantı (bağlantı ağırlığı) güçlenir.

Hebb kuralı, yapay sinir ağlarında şu şekilde ifade edilebilir:

$\Delta w_{ij} = \eta \cdot x_i \cdot y_j$

Bu formülde:

- Δw_{ij}= i. nöron ile j. nöron arasındaki ağırlık değişimini ifade eder. Ağırlık değişim miktarı.
- η(eta) : öğrenme oranı (learning rate) olarak bilinir ve ağırlıkların ne kadar güncelleneceğini belirler.
- x_i= i. nöronun girdisini ifade eder.
- y_j= jj. nöronun çıktısını ifade eder.

Bu denklem, nöronlar arasındaki bağlantıların, giriş ve çıkışlar aynı anda aktif olduğunda güçleneceğini gösterir.

En Basit Tek Katmanlı Algılayıcı

- Öğrenme kuralı (Hebb kuralı) : Eğer perceptronun ürettiği çıkış ile hedef değer aynı olursa, ağırlıklarda herhangi bir değişme olmaz.
- Bu durum sınıf ayracının doğru pozisyonda olduğunu gösterir. Fakat ağ, hedef değerden farklı bir çıkış üretmiş ise, ki bu durumda ağırlıklar değiştirilerek sınıf ayracının pozisyonu değiştirilmelidir, iki durum söz konusu olabilir.

$$w_i\,(new) = w_i\,(old) + x_i\,y$$

$$b\,(new) = b\,(old) + y$$

Bu ağlarda ara katman bulunmaz. Algılayıcı, eğitilebilen tek bir işlemci elemandan oluşmaktadır.

Burada eğitilebilirlikten kasıt wi ağırlık değerlerinin değiştirilebilir olması demektir.

İşlemci elemana girdi değerleri ve bu değerlere karşılık gelen çıktı değerleri gösterilerek öğrenme kuralına göre ağın çıktı değeri hesaplanmaktadır.

Olması gereken çıktı değerine ulaşılana kadar ağırlıklar değiştirilmektedir. Bu noktada çıktı değeri ile istenilen değer arasındaki fark Hata olarak belirtilir.

Amaç toplam kullanılan performans fonksiyonunun çıktısının sıfır değerine indirilmesidir. Algılayıcıda, anlaşılacağı gibi ileri beslemeli ve danışmanlı öğrenme uygulanmaktadır.

Yapay sinir ağlarında kullanımı ve Uygulama alanları

Yapay sinir ağları, öğrenme sürecinde Hebb kuralını kullanarak bağlantı ağırlıklarını günceller. Örneğin, bir yapay nöron girişlerinden aldığı sinyalleri işler ve doğru çıktıyı üretmeye çalışır. Eğer girdiler ve çıktı arasındaki ilişki tutarlıysa, bu bağlantılar güçlenir ve nöron daha iyi öğrenir.

Denetimsiz öğrenme (unsupervised learning) olarak bilinen bu süreç, hata sinyali veya geri yayılım olmadan, sadece nöronların birlikte aktif olmasıyla gerçekleşir.

Hebb öğrenme kuralı, **model tanıma**, **hafıza modelleme** ve **veri analizi** gibi alanlarda kullanılır. Biyolojik sinir sistemlerinde hafızanın nasıl oluştuğuna dair temel bir model olarak kabul edilirken, yapay sinir ağlarında da benzer prensiplerle kullanılır.

ADALINE((*Adaptive Linear Neuron*))

Bernard Widrow, 1950'lerin sonlarında, Frank Rosenblatt'ın perceptronu geliştirdiği sırada, sinir ağları üzerine çalışmaya başlamıştır. 1960 yılında Widrow ve onun lisansüstü öğrencisi Marcian Hoff, ADALINE ağı ile En Küçük Kareler (Least Mean Square) algoritması olarak adlandırılan bir öğrenme kuralı geliştirdiler. Açık adı 'ADAptive LInear NEuron' veya 'ADAptive LINear Element' olan bu sinir hücresi modeli yapısal olarak algılayıcıdan farklı değildir. Ancak, algılayıcı aktivasyon fonksiyonu olarak eşik fonksiyonu kullanırken ADALINE doğrusal fonksiyon kullanır. Her iki modelde yalnızca doğrusal olarak ayrılabilen problemlere çözüm üretebilmektedir.

Widrow-Hoff kuralı da denilen En Küçük Kareler algoritması, perceptronun öğrenme kuralından daha güçlüdür.

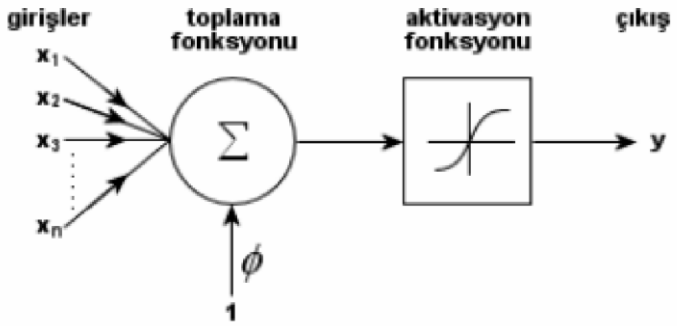
Perceptron öğrenme kuralı bir çözüme yakınsamayı garanti etse dahi, eğitim kalıplarının sınır çizgisine yakınlığından dolayı gürültüye duyarlı olabilir. En küçük kareler algoritması, ortalama karesel hatayı minimize ettiğinden eğitim kalıplarını sınır çizgisinden olabildiğince uzak tutmaya çalışır..

ADALINE ?

Doğrusal Ayrılabilir (Linearly Separable) desenleri sınıflandırma bildiğini, buna karşılık bir çok problemin doğrusal ayrılabilir olmadığını ve bu durumun da algılayıcı için önemli sınırlamalara neden olduğunu bilinmektedir.

Özellikle
mantıksal XOR problemlerinin çözümünün olmaması nedeniyle algılayıcı,
günümüzde doğrusal olmayan problemlerin çözümünde çok kullanılmamaktadır.

ADALINE



ADALINE modelinde, perceptronda olduğu gibi basamak aktivasyon fonksiyonu kullanılabilir.
Fakat genellikle hiperbolik tanjant fonksiyonu yada sigmoidal fonksiyonu kullanılır. öğrenme kuralı, perceptronda kullanılan Hebb kuralından farklıdır.

Giriş ve hedef vektörleri : Perceptronlarda olduğu gibi, giriş değerlerinin sayısı için bir kısıtlama yoktur.
Giriş vektörü, negatif yada pozitif herhangi bir değeri içerebilir.
Giriş değerlerinin 1'den büyük yada küçük olması performansı etkilemez.
Hedef vektörüne getirilecek kısıtlamalar ise, aktivasyon fonksiyonları ile ilişkilidir.
Eğer perceptronda olduğu gibi basamak aktivasyon fonksiyonu kullanılırsa, hedef değerler sadece 0 veya 1 olabilir.
Fakat hiperbolik tanjant yada sigmoidal aktivasyon fonksiyonları kullanılırsa, bu durumda hedef vektör, 0 ve 1 aralığındaki değerleri de içerebilir.

ADALINE

Toplama fonksiyonu : Ağırlıklı toplam fonksiyonu kullanılır. Yüzey ayracının ötelenmesi gereken durumlarda eşik değer ilave edilebilir.

$$NET = \left(\sum_{i=1}^{n} w_i \cdot x_i \right) + \phi$$

Aktivasyon fonksiyonu : ADALINE modelinde, genellikle hiperbolik tanjant fonksiyonu yada sigmoidal fonksiyon kullanılır. Sigmoidal fonksiyonda β değerini değiştirerek, farklı eğimlerde fonksiyonlar elde etmek mümkündür.

$$y = f(NET) = \frac{e^{NET} - e^{-NET}}{e^{NET} + e^{-NET}} \quad \text{yada} \quad y = f(NET) = \frac{1}{1 + e^{-\beta . NET}}$$

$$e = y^* - y$$

Öğrenme kuralı (Delta kuralı) : $*$ y hedef değeri ve y gerçek çıkışı göstermek ı $kareselhata = (y^* - y)^2$ alını elde etmek amacı ile hatayı tanımlayalım.

Hatanın negatif değerlerinden kurtulmak için karesel hatayı tanımlayalım.

ADALINE

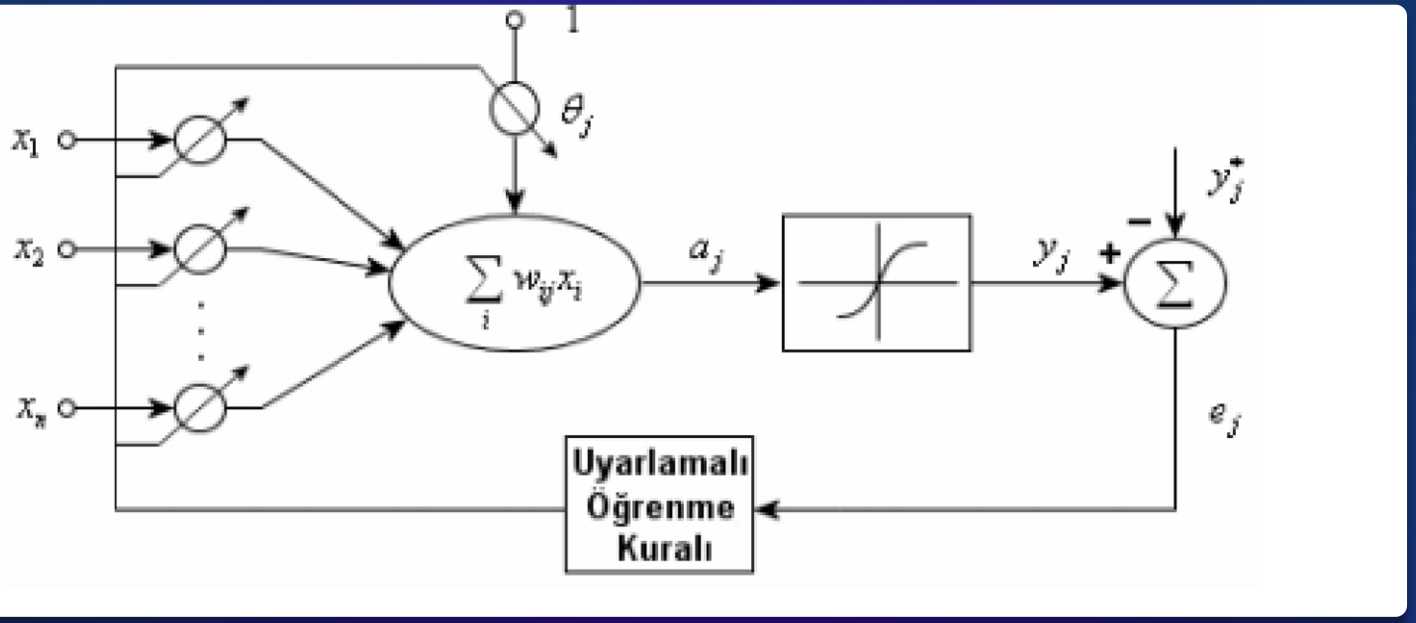
Hata yüzeyinin minimum noktasını bulabilmek için gradyanın tersi yönde İlerlenir.

$$\frac{\partial(kareselhata)}{\partial w_i} = -2\left(y^* - \sum_i w_i x_i\right)\cdot x_i \Rightarrow \eta \cdot e \cdot x_i$$

Yukardaki ifadede katsayılar, öğrenme katsayısına, η , dahil edilmiştir. Böylelikle ağırlıkları güncellemek için kullanılacak delta öğrenme kuralı bulunmuş olur.

$$\Delta w_i = \eta \cdot e \cdot x_i \quad \Rightarrow \quad w_i(k+1) = w_i(k) + \eta \cdot e \cdot x_i$$

Eşik değerin değiştirilmesi $\phi(k+1) = \phi(k) + \eta \cdot e$


1
θ_j
x_1
x_2
x_n
$\sum_i w_{ij} x_i$
a_j
y_j
y_j^*
e_j
Uyarlamalı Öğrenme Kuralı

ADALINE

ADALINE

DALINE Modelinin Özellikleri:

- **Lineer Aktivasyon:** ADALINE, aktivasyon fonksiyonu olarak doğrusal (lineer) bir fonksiyon kullanır, yani çıktı lineer bir şekilde hesaplanır. Bu özellik, doğrusal olarak ayrılabilen problemleri çözmek için uygundur.
- **Hata Temelli Öğrenme:** ADALINE, hataya dayalı bir öğrenme modelidir. Gerçek çıkış ve hedef çıkış arasındaki farkı minimize ederek öğrenir.

ADALINE

- ADALINE, sinyallerin işlenmesi, model tanıma, veri sınıflandırma gibi alanlarda kullanılır. Temel yapısı ve öğrenme prensibi, yapay sinir ağlarının daha kompleks modelleri için de bir temel oluşturmuştur.
- Özetle, **ADALINE** modelinde öğrenme süreci, tahmin edilen çıkış ile doğru çıkış arasındaki farkı minimize ederek, ağırlıkların güncellenmesine dayanır. Bu öğrenme modeli, daha kompleks yapay sinir ağları ve öğrenme algoritmalarının temelini oluşturur.

Perceptron Adaline benzerlikleri

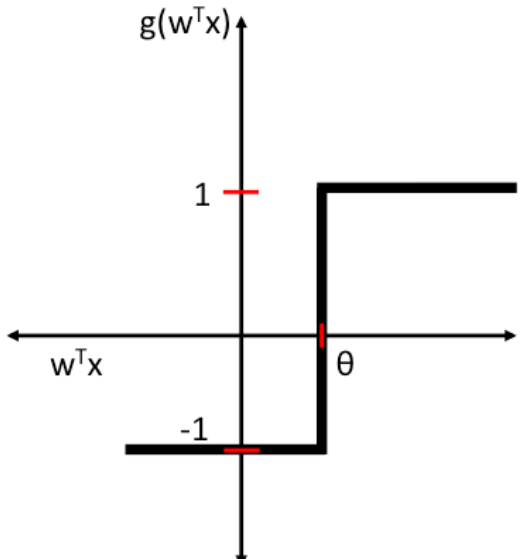
Her ikisi de doğrusal bir karar sınırına sahip.
her ikisi de yinelemeli öğrenebilir, örneklemle öğrenme
her ikisi de bir eşik işlevi kullanır.

İki algoritmadaki ilk adım, net giriş z olarak adlandırılan, özellik değişkenlerimizin x ve w model ağırlıklarının doğrusal birleşimi olarak hesaplamaktır.

$$\mathbf{z} = w_1 x_1 + \cdots + w_m x_m = \sum_{j=1}^{m} x_j w_j$$
$$= \mathbf{w}^T \mathbf{x}$$

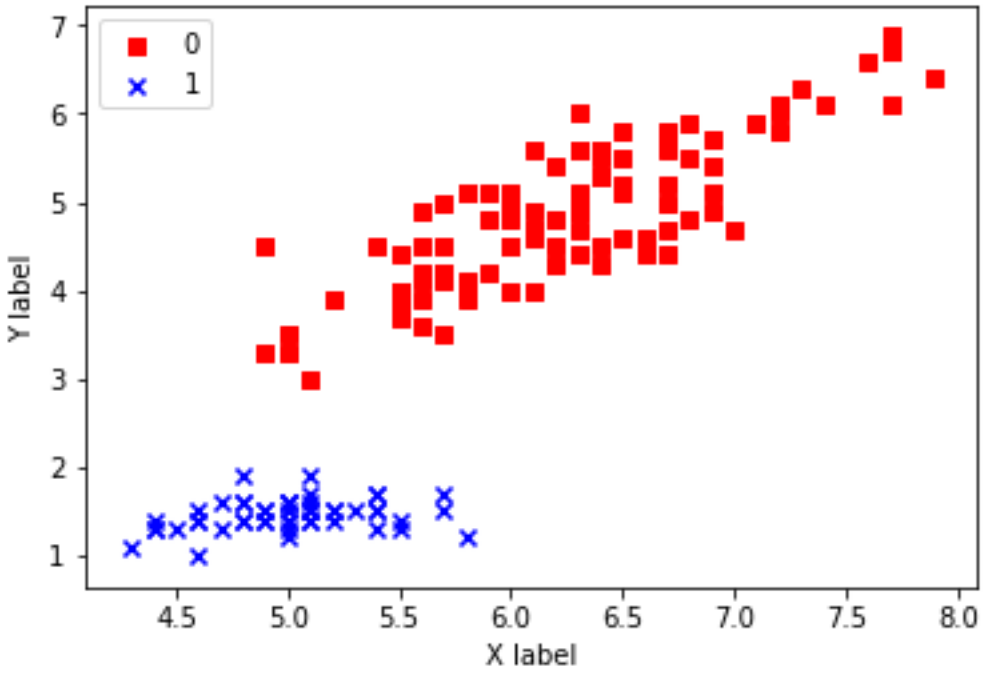
Daha sonra, Perceptron ve Adaline'da, bir tahmin yapmak için bir eşik fonksiyonu tanımlarız.

$$g(\mathbf{z}) = \begin{cases} 1 & \text{if } \mathbf{z} \geq \theta \\ -1 & \text{otherwise.} \end{cases}$$

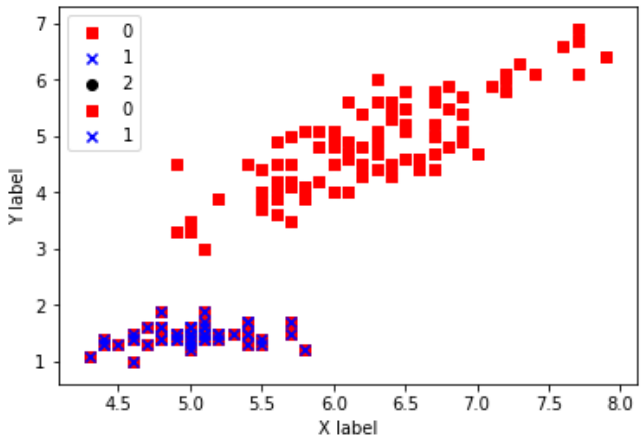


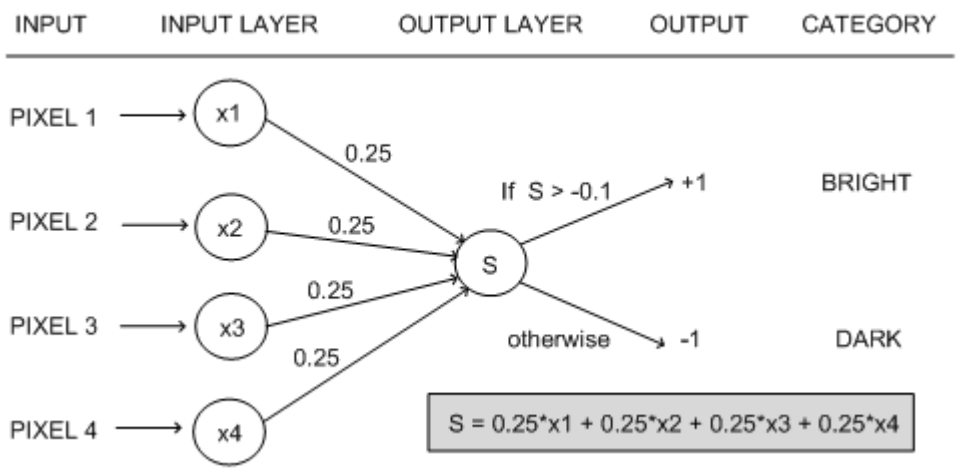
Perceptron Adaline farklılıkları

```
from sklearn.datasets import load_iris

#load the iris dataset
iris = load_iris()
#our inputs will contain 2 features
X = iris.data[:, [0, 2]]
#the labels are the following
y = iris.target
import matplotlib.pyplot as plt
def plot_scatter(X,y):
    colors = ["red","blue","black","yellow","green","purple","orc
    markers = ('s', 'x', 'o', '^', 'v')

    for i, yi in enumerate(np.unique(y)):
        Xi = X[y==yi]
        plt.scatter(Xi[:,0], Xi[:,1],
                    color=colors[i], marker=markers[i], label=yi)

    plt.xlabel('X label')
    plt.ylabel('Y label')
    plt.legend(loc='upper left')

#Generate the Scatterplot
plot_scatter(X,y)

#Classifier for y = 0
y = np.where(y == 0, 1, 0)
plot_scatter(X,y)
```



```
class Perceptron(object):
    #The constructor of our class.
    def __init__(self, learningRate=0.01, n_iter=50, random_state=1):
        self.learningRate = learningRate
        self.n_iter = n_iter
        self.random_state = random_state
        self.errors_ = []

    def fit(self, X, y):
        #for reproducing the same results
        random_generator = np.random.RandomState(self.random_state)

        #Step 0 = Get the shape of the input vector X
        #We are adding 1 to the columns for the Bias Term
        x_rows, x_columns = X.shape
        x_columns = x_columns+1

        #Step 1 - Initialize all weights to 0 or a small random number
        #weight[0] = the weight of the Bias Term
        self.weights = random_generator.normal(loc=0.0, scale=0.001, size=x_co

        #for how many number of training iterrations where defined
        for _ in range(self.n_iter):
            errors = 0
            for xi, y_actual in zip(X, y):
                #create a prediction for the given sample xi
                y_predicted = self.predict(xi)
                #print(y_actual, y_predicted)
                #calculte the delta
                delta = self.learningRate*(y_actual - y_predicted)
                #update all the weights but the bias
                self.weights[1:] += delta * xi
                #for the bias delta*1 = delta
                self.weights[0] += delta

                #if there is an error. Add to the error count for the batch
                errors += int(delta != 0.0)

            #add the error count of the batch to the errors variable
            self.errors_.append(errors)

        #print(self.errors_)

    def Errors(self):
        return self.errors_

    def z(self, X):
        #np.dot(X, self.w_[1:]) + self.w_[0]
        z = np.dot(X, self.weights[1:]) + self.weights[0]
        return z

    def predict(self, X):
        #Heaviside function. Returns 1 or 0
        return np.where(self.z(X) >= 0.0, 1, 0)

ppn = Perceptron(learningRate=0.1, n_iter=15)
ppn.fit(X, y)
print(ppn.errors )
```




INPUT
INPUT LAYER
OUTPUT LAYER
OUTPUT
CATEGORY
PIXEL 1
PIXEL 2
PIXEL 3
PIXEL 4
x1
x2
x3
x4
0.25
0.25
0.25
0.25
S
If S > -0.1
+1
BRIGHT
otherwise
-1
DARK
S = 0.25*x1 + 0.25*x2 + 0.25*x3 + 0.25*x4