

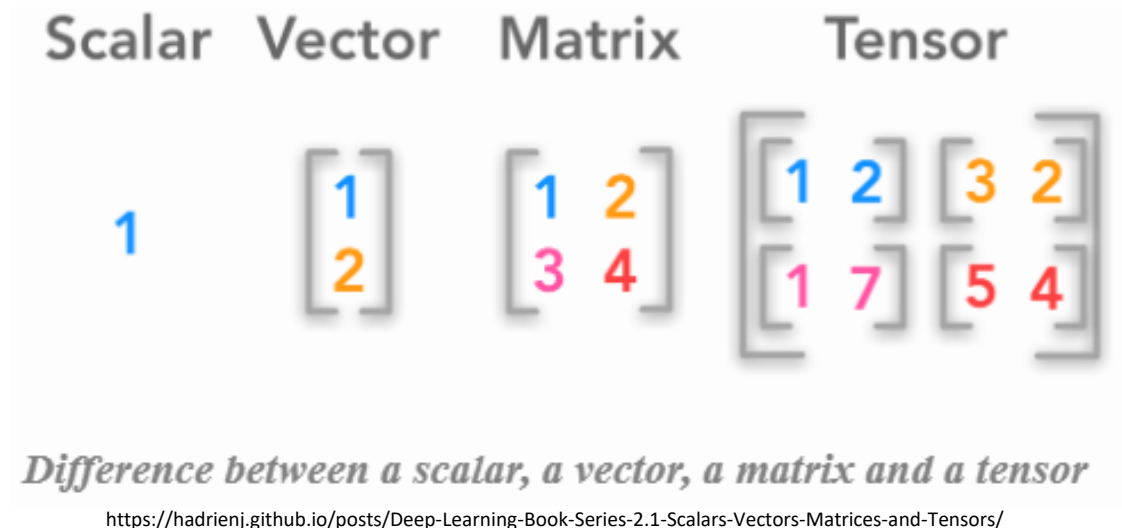
Derin Öğrenmeye Giriş  
**HAFTA 2**  
**Matematiksel Temeller**  
**Tensör işlemleri**  
**Aktivasyon fonksiyonları**

**Dr. Öğretim Üyesi Burcu ÇARKLI YAVUZ**

**[bcarkli@sakarya.edu.tr](mailto:bcarkli@sakarya.edu.tr)**

# Matematiksel Yapıların Derin Öğrenmedeki Rolü

- Derin öğrenme, büyük veri setleri üzerinde çalışırken matematiksel yapıları yoğun şekilde kullanır.
- Sinir ağlarının yapısı ve optimizasyon süreçleri, temel matematiksel yapılara (skaler, vektör, matris ve tensör) dayanır.
- Bu yapıların doğru anlaşılması, sinir ağlarının eğitimi ve performansını optimize etmek için önemlidir.



# Skalerler, Vektörler, Matrisler ve Tensörler

**Skalerler:** Skalerler, sıcaklık, zaman veya ölçülebilir diğer nicelikleri temsil etmek için kullanılabilen tek sayısal değerlerdir. 0. dereceden bir tensördür. Büyüklükleri vardır ancak yönleri yoktur. Örneğin 5, -3.14,  $2/3$

**Derin öğrenmede skaler değerler nerede kullanılır?**

**Öğrenme oranı:** Modelimizin ne kadar hızlı öğreneceğini belirler. Örneğin, 0.01 gibi küçük bir değer.

**Bias değerleri:** Her nöronun sahip olduğu ek bir parametre. Örneğin, 0.5.

**Aktivasyon fonksiyonlarının eşik değerleri:** ReLU fonksiyonundaki 0 gibi.

**Kayıp fonksiyonu çıktıları:** Modelimizin performansını ölçen tek bir sayı.

# Skalerler, Vektörler, Matrisler ve Tensörler

**Vektörler:** Skalerlerin sıralı dizileridir ve 1. dereceden tensöre bir örnektir.

Vektörler, **vektör uzayları** olarak bilinen nesnelerin üyeleridir. Bir vektör uzayı, belirli bir uzunluktaki (veya boyuttaki) *tüm* olası vektörlerin tüm koleksiyonu olarak düşünülebilir .

```
import numpy as np
```

```
x = np.array([1, 2, 3, 4])  
x
```

```
array([1, 2, 3, 4])
```

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

$$\begin{bmatrix} 7 \\ -3 \\ 5 \end{bmatrix}$$

A simple vector of size 3

# Skalerler, Vektörler, Matrisler ve Tensörler

## Derin öğrenmede vektörler nerede kullanılır?

**Girdi verileri:** Bir resmin gri tonlamalı piksel değerleri bir vektör olabilir.

**Nöron ağırlıkları:** Her nöronun girdilere verdiği önem bir vektördür.

**Gradyanlar:** Kayıp fonksiyonunun her parametre için türevi bir vektör oluşturur.

**Aktivasyon değerleri:** Bir katmandaki tüm nöronların çıktıları bir vektördür.

Örnek:  $[0.1, 0.2, 0.3]$  girdisine sahip bir nöronumuz var ve ağırlıklarımız  $[0.5, 0.5, 0.5]$  ise, çıktımız ne olur? (Bias'ı ihmal edelim)

Cevap:  $0.1 * 0.5 + 0.2 * 0.5 + 0.3 * 0.5 = 0.05 + 0.1 + 0.15 = 0.3$

# Skalerler, Vektörler, Matrisler ve Tensörler

## Dot Product (iç çarpım, nokta çarpımı)

İki vektörün iç çarpımını hesaplayabilmek için ikisinin de size ı (eleman sayısı) eşit olmalıdır. Aşağıdaki örnekteki vektörlerin ikisi için de size= 3 tür.

İç çarpım sonucu daima skalerdir.

$$a \cdot b = \begin{bmatrix} 7 \\ -3 \\ 5 \end{bmatrix} \cdot \begin{bmatrix} 2 \\ 8 \\ 1 \end{bmatrix}$$

The dot product between two vectors

The dot product then gives us,

$$(7 * 2) + (-3 * 8) + (5 * 1) = 14 + -42 + 5 = -23$$

# Skalerler, Vektörler, Matrisler ve Tensörler

**Matrisler:** Matris, aynı boyuttaki vektörlerin bir araya gelmesiyle oluşan bir yapıdır. İki boyutlu dizilerdir (2. dereceden bir tensör).

array() fonksyonu ile 2 boyutlu diziler (matrisler) oluşturabiliriz:

```
A = np.array([[1, 2], [3, 4], [5, 6]])  
A
```

```
array([[1, 2],  
       [3, 4],  
       [5, 6]])
```

$$\begin{bmatrix} 15 & 9 & 0 \\ 3 & 2 & -1 \\ -5 & 1 & 4 \\ -19 & 100 & 3 \end{bmatrix}$$

4×3 matris örneği

# Skalerler, Vektörler, Matrisler ve Tensörler

**shape:** Bir dizinin shape i onun her boyutunda kaç değer olduğunu verir. Örneğin 2 boyutlu bir dizide (matriste) shape satır ve sütun sayısını verir.

```
A = np.array([[1, 2], [3, 4], [5, 6]])  
A
```

```
array([[1, 2],  
       [3, 4],  
       [5, 6]])
```

```
A.shape
```

```
(3, 2)
```

```
x = np.array([1, 2, 3, 4])  
x
```

```
array([1, 2, 3, 4])
```

```
x.shape
```

```
(4,)
```



# Skalerler, Vektörler, Matrisler ve Tensörler

**Transpoze işlemi:** Bir vektörün transpozesi alındığında satır vektörü ise sütun vektörüne, sütun vektörü ise satır vektörüne dönüşür. Bir matrisin transpozesi alındığında satırlar sütuna, sütunlar satırlara dönüşür.

$$\begin{bmatrix} 1 \\ 2 \end{bmatrix}^T = \begin{bmatrix} 1 & 2 \end{bmatrix}$$

*Vector transposition*

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}^T = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$

*Square matrix transposition*

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}^T = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix}$$

*Non-square matrix transposition*

# Skalerler, Vektörler, Matrisler ve Tensörler

## Transpoze işlemi:

```
A = np.array([[1, 2], [3, 4], [5, 6]])  
A
```

```
array([[1, 2],  
       [3, 4],  
       [5, 6]])
```

```
A_t = A.T  
A_t
```

```
array([[1, 3, 5],  
       [2, 4, 6]])
```

```
A.shape
```

```
(3, 2)
```

```
A_t.shape
```

```
(2, 3)
```

# Skalerler, Vektörler, Matrisler ve Tensörler

## Matris Toplamı:

İki matrisin toplanabilmesi için satır ve sütun sayılarının (shape) eşit olması gerekir.

Her iki matrisin i. Satır ve j. Sütunundaki elemanları toplanarak toplam matrisi elde edilir.

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} + \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 2 & 3 \end{bmatrix} = \begin{bmatrix} 2 & 6 \\ 5 & 9 \\ 7 & 9 \end{bmatrix}$$
  
$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} + \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 2 & 3 \end{bmatrix} = \begin{bmatrix} 2 & 6 \\ 5 & 9 \\ 7 & 9 \end{bmatrix}$$
  
$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} + \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 2 & 3 \end{bmatrix} = \begin{bmatrix} 2 & 6 \\ 5 & 9 \\ 7 & 9 \end{bmatrix}$$

# Skalerler, Vektörler, Matrisler ve Tensörler

## Matris Toplamı:

```
A = np.array([[1, 2], [3, 4], [5, 6]])  
A
```

```
array([[1, 2],  
       [3, 4],  
       [5, 6]])
```

```
B = np.array([[2, 5], [7, 4], [4, 3]])  
B
```

```
array([[2, 5],  
       [7, 4],  
       [4, 3]])
```

```
# Add matrices A and B  
C = A + B  
C
```

```
array([[ 3,  7],  
       [10,  8],  
       [ 9,  9]])
```

# Skalerler, Vektörler, Matrisler ve Tensörler

## Matrise bir skaler eklenmesi

A

```
array([[1, 2],  
       [3, 4],  
       [5, 6]])
```

```
# Exemple: Add 4 to the matrix A  
C = A+4  
C
```

```
array([[ 5,  6],  
       [ 7,  8],  
       [ 9, 10]])
```

# Skalerler, Vektörler, Matrisler ve Tensörler

**Broadcasting:** Numpy farklı şekillerdeki diziler üzerinde işlemler yapabilir. Daha küçük dizi, daha büyük olanın şekline uyacak şekilde genişletilir.

$$\begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \\ A_{3,1} & A_{3,2} \end{bmatrix} + \begin{bmatrix} B_{1,1} \\ B_{2,1} \\ B_{3,1} \end{bmatrix}$$

is equivalent to

$$\begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \\ A_{3,1} & A_{3,2} \end{bmatrix} + \begin{bmatrix} B_{1,1} & B_{1,1} \\ B_{2,1} & B_{2,1} \\ B_{3,1} & B_{3,1} \end{bmatrix} = \begin{bmatrix} A_{1,1} + B_{1,1} & A_{1,2} + B_{1,1} \\ A_{2,1} + B_{2,1} & A_{2,2} + B_{2,1} \\ A_{3,1} + B_{3,1} & A_{3,2} + B_{3,1} \end{bmatrix}$$

# Skalerler, Vektörler, Matrisler ve Tensörler

## Farklı shape leri olan matrislerin toplanması

```
A = np.array([[1, 2], [3, 4], [5, 6]])  
A
```

```
array([[1, 2],  
       [3, 4],  
       [5, 6]])
```

```
B = np.array([[2], [4], [6]])  
B
```

```
array([[2],  
       [4],  
       [6]])
```

```
# Broadcasting
```

```
C=A+B  
C
```

```
array([[ 3,  4],  
       [ 7,  8],  
       [11, 12]])
```

# Skalerler, Vektörler, Matrisler ve Tensörler

**Matrislerin ve vektörlerin çarpımı:** Çarpma işleminin yapılabilmesi için birinci matrisin sütun sayısı ile ikinci matrisin satır sayısının eşit olması gerekmektedir.

Aşağıdaki örnekte (3x2) boyutunda bir matris ile (2x1) boyutundaki bir matris çarpılmış ve (3x1) boyutunda bir matris elde edilmiştir.

$$\begin{bmatrix} A & B \\ C & D \\ E & F \end{bmatrix} \times \begin{bmatrix} G \\ H \end{bmatrix} = \begin{bmatrix} A \times G + B \times H \\ C \times G + D \times H \\ E \times G + F \times H \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} \times \begin{bmatrix} 2 \\ 4 \end{bmatrix} = \begin{bmatrix} 1 \times 2 + 2 \times 4 \\ 3 \times 2 + 4 \times 4 \\ 5 \times 2 + 6 \times 4 \end{bmatrix} = \begin{bmatrix} 10 \\ 22 \\ 34 \end{bmatrix}$$



# Skalerler, Vektörler, Matrisler ve Tensörler

## Matrislerin ve vektörlerin çarpımı:

```
A = np.array([[1, 2], [3, 4], [5, 6]])  
A
```

```
array([[1, 2],  
       [3, 4],  
       [5, 6]])
```

```
B = np.array([[2], [4]])  
B
```

```
array([[2],  
       [4]])
```

```
C = np.dot(A, B)  
C
```



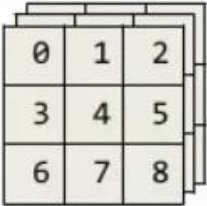
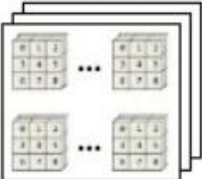
```
array([[10],  
       [22],  
       [34]])
```

```
C = A.dot(B)  
C
```

```
array([[10],  
       [22],  
       [34]])
```

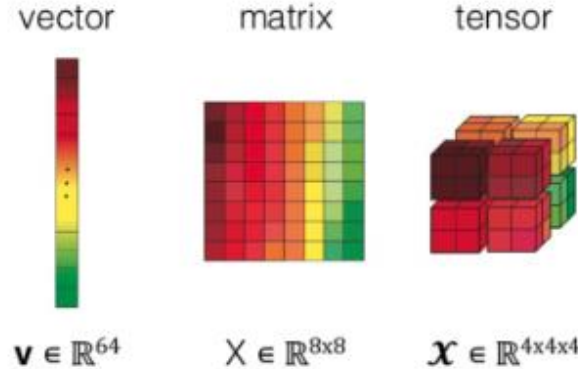
# Skalerler, Vektörler, Matrisler ve Tensörler

**Tensörler:** Üç veya daha fazla boyutlu bir sayı dizisidir. Örneğin, 3x3x3 boyutunda bir küp. Tensörler, özellikle görüntü işleme gibi daha karmaşık veri yapılarını temsil etmek için kullanılır.

Dimensions	Example	Terminology
1		Vector
2		Matrix
3		3D Array (3 <sup>rd</sup> order Tensor)
N		ND Array

# Skalerler, Vektörler, Matrisler ve Tensörler

## Tensörler:



## Derin öğrenmede tensörler nerede kullanılır?

**Renkli resim verileri:** Genelde 3 boyutludur (genişlik, yükseklik, renk kanalları).

**Video verileri:** 4 boyutlu olabilir (zaman, genişlik, yükseklik, renk kanalları).

**3B konvolüsyon işlemleri:** 3B görüntü işleme veya video analizinde kullanılır.

**Batch işlemleri:** Birden çok örneği aynı anda işlerken kullanılır.

➤ Bir veri grubu kullanıldığında, tensörün ilk eksenini grubun büyüklüğü (örnek sayısı) için ayrılmıştır.

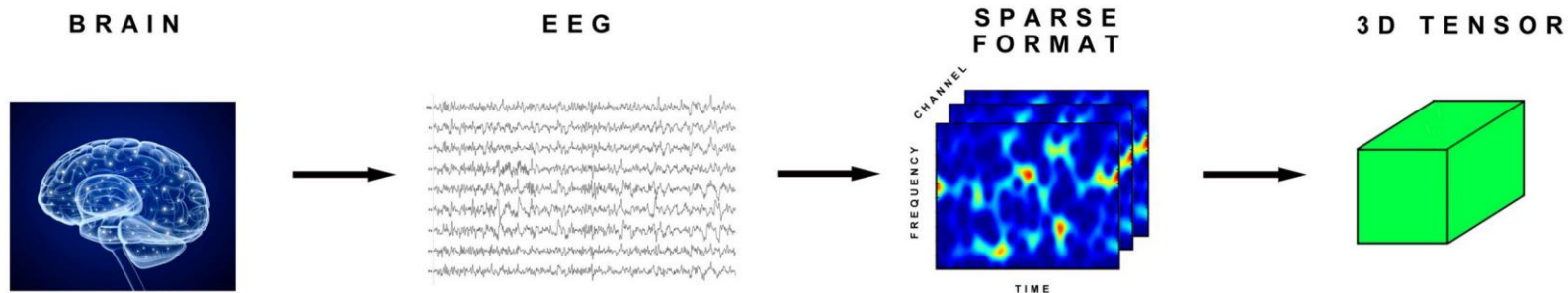
# Skalerler, Vektörler, Matrisler ve Tensörler

**Tensörler:** 3D tensörler, zaman serisi verileri için çok etkilidir.

## Tıbbi Taramalar

Beyinden gelen bir EEG (elektroensefalogram) sinyalini 3D tensör olarak kodlayabiliriz, çünkü 3 parametre olarak kapsüllenebilir:

➤ (time, frequency, channel)



➤ Eğer EEG taraması olan birden fazla hastamız olsaydı, bunun gibi bir 4D tensör olurdu:

➤ (sample\_size, time, frequency, channel)

Kaynak: <https://hackernoon.com/learning-ai-if-you-suck-at-math-p4-tensors-illustrated-with-cats-27f0002c9b32>

# Skalerler, Vektörler, Matrisler ve Tensörler

## Tensörler: Hisse fiyatları

➤ Bir günde 6.5 saat boyunca, her dakika için yüksek, düşük ve nihai hisse fiyatını 2D tensöründe **(390,3)** saklardık ( $6.5 \times 60 = 390$ ).

➤ Tipik bir hafta için (beş gün), şu şekilde bir 3D tensörümüz olurdu:

**(week\_of\_data, minutes, high\_low\_price)**

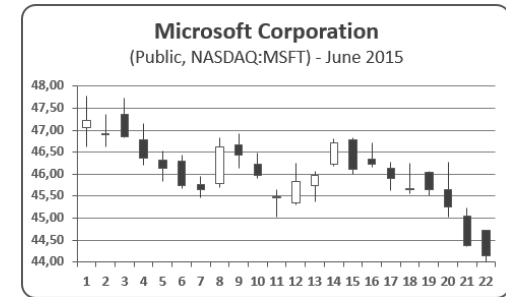
**(5,390,3)**

➤ 10 farklı stokumuz olsaydı, her biri bir haftalık veriye sahip olsaydı, şu şekilde olan 4D tensörümüz olurdu:

**(10,5,390,3)**

➤ Şimdi 4D tensörümüz tarafından temsil edilen bir hisse senedi koleksiyonu olan ortak bir fonumuz olduğunu düşünelim. Belki de portföyümüzü temsil eden 25 adet yatırım fonu koleksiyonumuz var, şimdi 5D tensör şeklimiz var:

**(25,10,5,390,3)**



Kaynak: <https://www.experfy.com/blog/learning-ai-if-you-suck-at-math-part4-tensors-illustrated-with-cats/>

# Skalerler, Vektörler, Matrisler ve Tensörler

## Tensörler: Metin verileri

- Metin verilerini de bir 3D tensörde saklayabiliriz.
- Örneğin Tweet, 140 karakterdir. Twitter, milyonlarca karakter türüne izin veren UTF-8 standardını kullanıyor, ancak temel ASCII ile aynı oldukları için, yalnızca ilk 128 karakterle ilgileniyoruz.
- Tek bir tweet, 2D olarak **(140,128)** şeklinde kapsüllenebilir.
- Eğer 1 milyon tweet kullanırsak bunu 3D şekil tensörü olarak depolayacağız:

**(tweet\_sayisi, tweet, karakter)**

**(1000000,140,128)**

# Skalerler, Vektörler, Matrisler ve Tensörler

## Tensörler: Görüntüler

- 4D tensörler Jpeg formatında bir dizi görüntüyü saklamak için kullanılabilir. Daha önce belirtildiği gibi, bir görüntü üç parametreyle saklanır:
  - Yükseklik
  - Genişlik
  - Renk derinliği
- Görüntü bir 3D tensördür, ancak görüntü seti 4D olur. Dördüncü alan örnek numarası için kullanılır.

# Skalerler, Vektörler, Matrisler ve Tensörler

## Tensörler: Görüntüler

- TensorFlow genelde görüntü verilerini aşağıdaki gibi depolar:

(sample\_size, yükseklik, genişlik, color\_depth)

- MNIST veri setinde 60.000 görüntü var. 28 piksel genişliğinde x 28 piksel yüksekliğindedir. Gri skalayı temsil eden 1 renk derinliğine sahiptirler.
- Bu yüzden MNIST veri setinin 4D tensörünün aşağıdaki gibi bir şekle sahip olduğunu söyleyebiliriz:

(60000,28,28,1)

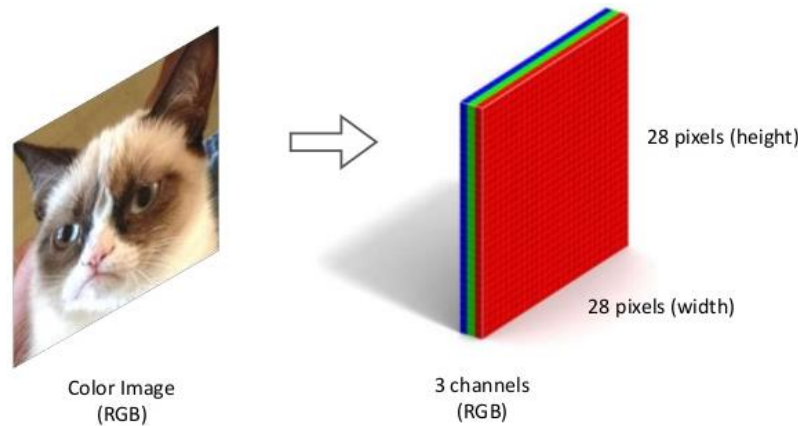
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4
5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6
7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7
8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8
9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9



# Skalerler, Vektörler, Matrisler ve Tensörler

## Tensörler: Görüntüler

- Renkli fotoğraflar, çözünürlüklerine ve kodlarına bağlı olarak farklı renk derinliğine sahip olabilir.
- Tipik bir JPG görüntüsü RGB kullanır ve böylelikle her biri kırmızı, yeşil, mavi olmak üzere 3'er renk derinliğine sahip olur.
- 750 piksel x 750 piksel görüntü için tensör boyutu:  
**(750,750,3)**



<https://stackoverflow.com/questions/47598968/python-reading-rgb-image-for-deep-learning>

# Skalerler, Vektörler, Matrisler ve Tensörler

## Tensörler: Video görüntüleri

- Bir 5D tensör video verilerini saklayabilir. TensorFlow'da video verileri şöyle kodlanır:
  - **(sample\_size, frames, width, height, color\_depth)**
- Beş dakikalık bir video çekersek ( $60 \text{ saniye} \times 5 = 300$ ),  $1920 \text{ piksel} \times 1080 \text{ piksel}$ , saniyede 15 örneklenmiş renkli karede ( $300 \text{ saniye} \times 15 = 4500 \text{ kare}$ ) aşağıdaki gibi görünen bir 4D tensör depolardı:
  - **(4500,1920,1080,3)**
- Tensördeki beşinci alan, video setimizde birden fazla video varken devreye giriyor. Tam olarak 10 tane video olsaydı, 5D'lik bir tensörümüz olurdu:
  - **(10,4500,1920,1080,3)**

# Skalerler, Vektörler, Matrisler ve Tensörler

## Numpy Örnekler:

### ➤ 0B Tensör (Skaler)

```
In [1]: import numpy as np
```

```
In [2]: a=np.array(3)
```

```
In [3]: a  
Out[3]: array(3)
```

```
In [4]: a.ndim  
Out[4]: 0
```

### • 1B Tensör (Dizi)

```
In [11]: b=np.array([1,3,5,4,2])
```

```
In [12]: b.ndim  
Out[12]: 1
```

```
In [13]: b  
Out[13]: array([1, 3, 5, 4, 2])
```

```
In [14]: b.size  
Out[14]: 5
```

# Skalerler, Vektörler, Matrisler ve Tensörler

## Numpy Örnekler: ➤ 2B Tensör Matris

```
In [19]: c=np.array([[1,2,3],[4,5,6],[7,8,9]])
```

```
In [20]: c
```

```
Out[20]:  
array([[1, 2, 3],  
       [4, 5, 6],  
       [7, 8, 9]])
```

```
In [21]: c.ndim
```

```
Out[21]: 2
```

```
In [22]: c.size
```

```
Out[22]: 9
```

```
In [23]: c.shape
```

```
Out[23]: (3, 3)
```

# Skalerler, Vektörler, Matrisler ve Tensörler

## Numpy Örnekler: ➤ 3B Tensör

```
In [29]: d=np.random.rand(3,2,3)
```

```
In [30]: d
```

```
Out[30]:
```

```
array([[[0.81726371, 0.8597048 , 0.89624965],  
        [0.03068633, 0.80458248, 0.4441645 ]],  
       [[0.66730579, 0.172524 , 0.61339767],  
        [0.33361243, 0.67674554, 0.05245607]],  
       [[0.69444104, 0.07656473, 0.94894536],  
        [0.62734666, 0.38506151, 0.5115817 ]]])
```

```
In [31]: d.ndim
```

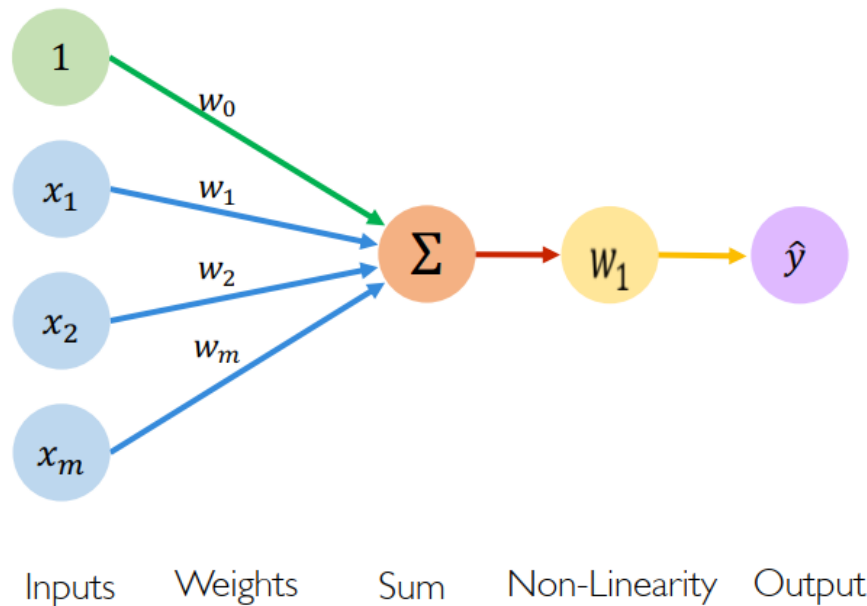
```
Out[31]: 3
```

```
In [32]: d.shape
```

```
Out[32]: (3, 2, 3)
```

# Yapay Sinir Ağları

## The Perceptron: Forward Propagation



Output

Linear combination of inputs

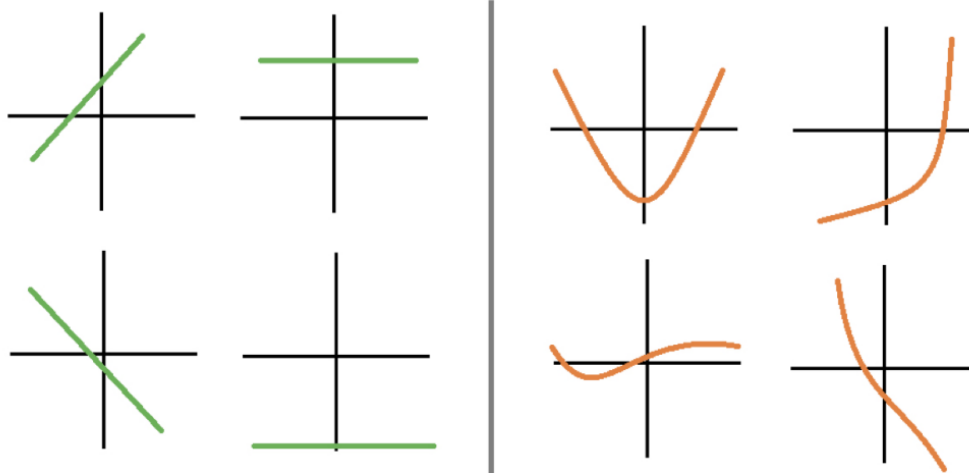
$$\hat{y} = g \left( w_0 + \sum_{i=1}^m x_i w_i \right)$$

Non-linear activation function

Bias

# Aktivasyon Fonksiyonunun Önemi

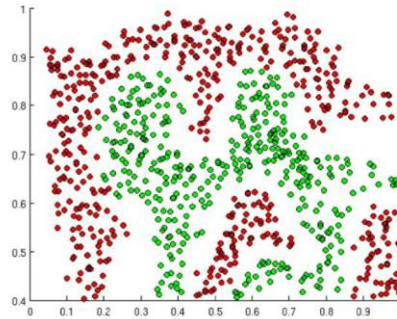
## Doğrusal ve Doğrusal Olmayan Fonksiyonlar:



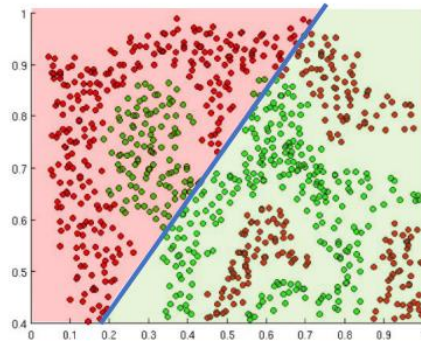
- Düz çizgilerle (doğrusal) sınırlı kalmayan veri ilişkilerini doğru bir şekilde yansıtmak için, sağ taraftaki gibi eğimli ve kıvrımlı (doğrusal olmayan) çizgilerle temsil edilen aktivasyon fonksiyonları gereklidir.
- Bu fonksiyonlar, modelimizin gerçek dünya verilerinin sunduğu zengin ve karmaşık yapıları öğrenmesini sağlar.

# Aktivasyon Fonksiyonunun Önemi

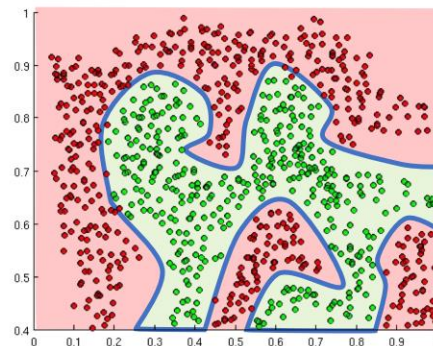
The purpose of activation functions is to **introduce non-linearities** into the network



What if we wanted to build a Neural Network to distinguish green vs red points?



Linear Activation functions produce linear decisions no matter the network size



Non-linearities allow us to approximate arbitrarily complex functions



# Derin Öğrenmede Aktivasyon Fonksiyonları

- Aktivasyon fonksiyonları, yapay sinir ağlarında her bir nöronun çıktısını belirleyen matematiksel fonksiyonlardır.
- Bu fonksiyonlar, ağın doğrusal olmayan yapılar öğrenmesini sağlar ve derin öğrenme modellerinin karmaşık örüntüleri yakalama yeteneğini artırır.

## Aktivasyon Fonksiyonlarının Önemi

- Doğrusal olmayan yapıları modelleme
- Gradyan akışını kontrol etme
- Çıktı aralığını normalize etme
- Ağın öğrenme kapasitesini artırma

# Keras Aktivasyon Fonksiyonları

relu function  
sigmoid function  
softmax function  
softplus function  
softsign function  
tanh function  
selu function  
elu function  
exponential function  
leaky\_relu function  
relu6 function  
silu function  
hard\_silu function  
gelu function  
hard\_sigmoid function  
linear function  
mish function  
log\_softmax function

# Yaygın Aktivasyon Fonksiyonları

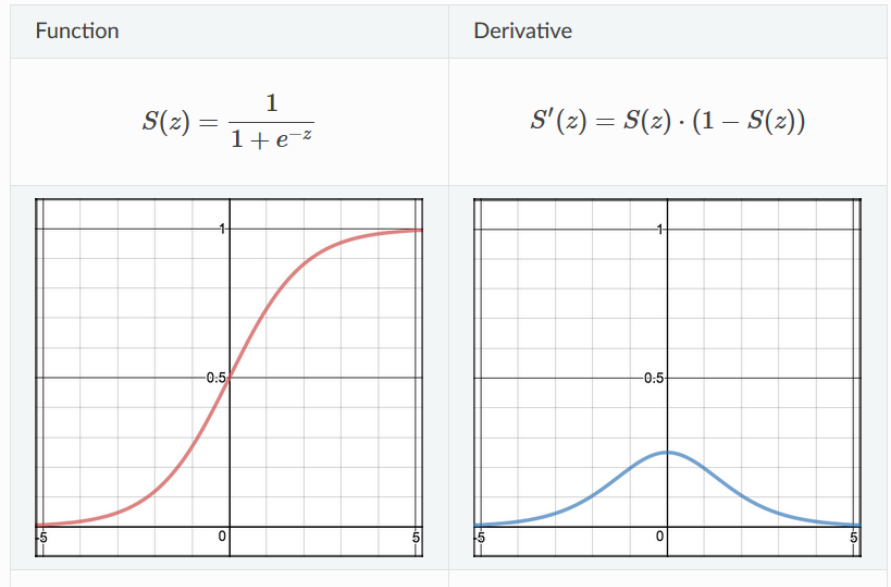
## Sigmoid Fonksiyonu:

### Özellikleri:

- Çıktı aralığı: (0, 1)
- S-şeklinde bir eğri
- Gradyan vanishing (gradyan yok olması) problemi yaşayabilir

### Kullanım alanları:

- İkili sınıflandırma problemlerinde çıkış katmanı
- Eski yapay sinir ağı modellerinde yaygın kullanım



`keras.activations.sigmoid(x)`

# Yaygın Aktivasyon Fonksiyonları

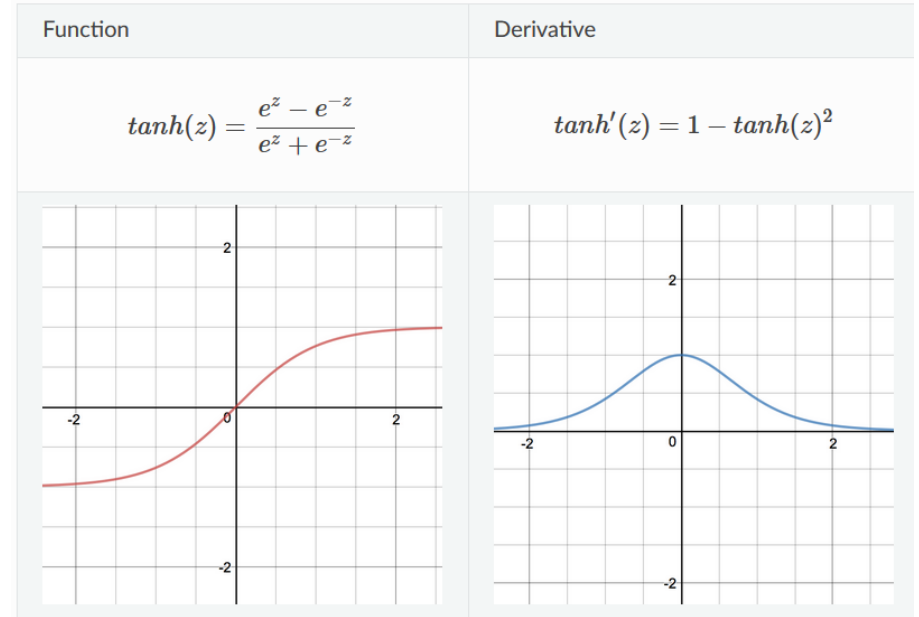
## Hiperbolik Tanjant (tanh) Fonksiyonu:

### Özellikleri:

- Çıktı aralığı:  $(-1, 1)$
- Sigmoid'e benzer, ancak sıfır merkezli
- Sigmoid'e göre daha güçlü gradyanlar

### Kullanım alanları:

- RNN ve LSTM gibi tekrarlayan ağlarda
- Gizli katmanlarda



`keras.activations.tanh(x)`

# Yaygın Aktivasyon Fonksiyonları

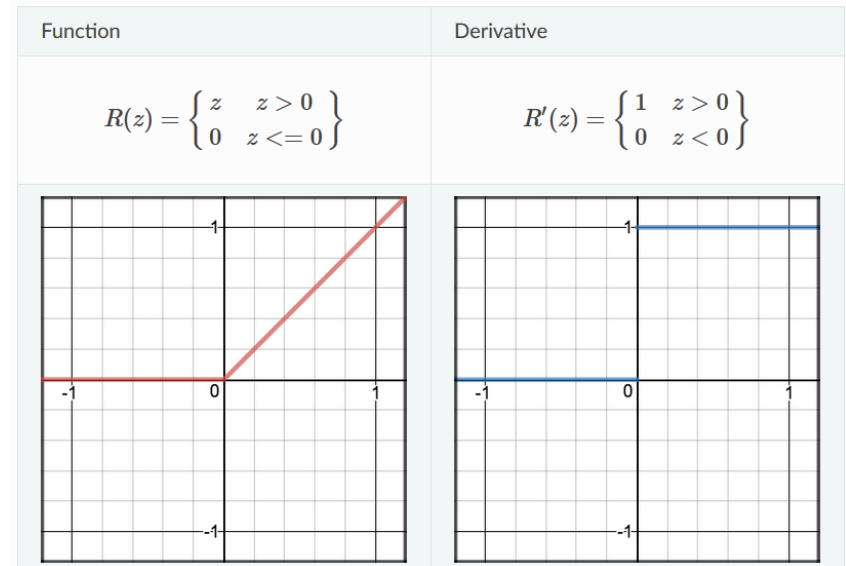
## Rectified Linear Unit (ReLU):

### Özellikleri:

- Çıktı aralığı:  $[0, \infty)$
- Hesaplama açısından verimli
- Seyrek aktivasyon sağlar
- Negatif değerler için gradyan sıfır (dying ReLU problemi)

### Kullanım alanları:

- Çoğu derin öğrenme modelinde varsayılan seçim
- Özellikle CNN'lerde yaygın kullanım



```
keras.activations.relu(x, alpha=0.0, max_value=None, threshold=0.0)
```

# Yaygın Aktivasyon Fonksiyonları

## Leaky ReLU:

Formül:

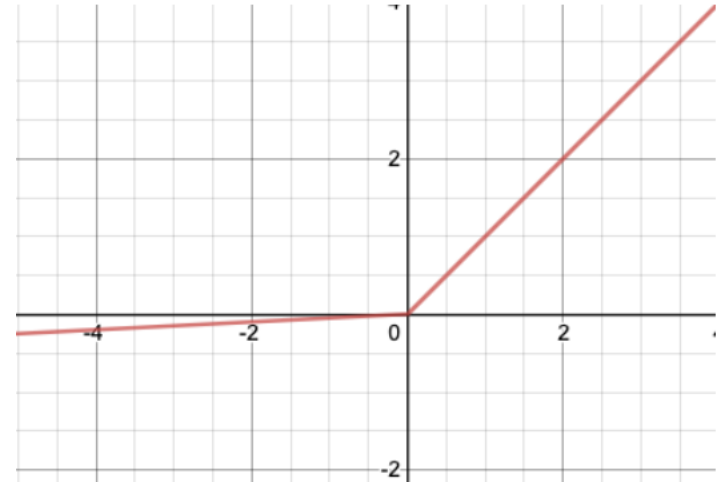
$f(x) = \max(\alpha x, x)$ ,  $\alpha$  küçük bir pozitif sabittir (genellikle 0.01)

Özellikleri:

- ReLU'nun bir varyasyonu
- Negatif değerler için küçük bir eğim sağlar
- Dying ReLU problemini azaltır

Kullanım alanları:

- ReLU'nun alternatifi olarak, özellikle dying ReLU problemi yaşanan durumlarda



`keras.activations.leaky_relu(x, negative_slope=0.2)`

# Yaygın Aktivasyon Fonksiyonları

## Parametric ReLU (PReLU):

Formül:

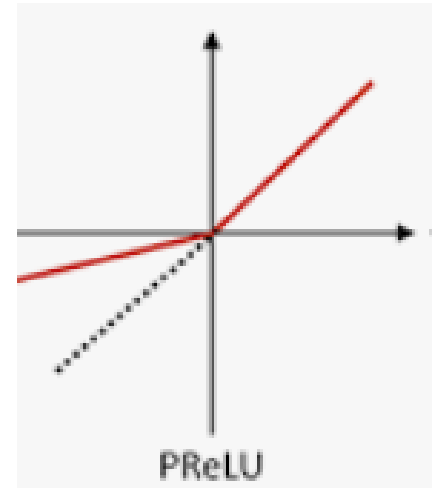
$f(x) = \max(\alpha x, x)$ ,  $\alpha$  öğrenilebilir bir parametre

Özellikleri:

- Leaky ReLU'nun öğrenilebilir versiyonu
- Her nöron için ayrı  $\alpha$  değeri öğrenilebilir

Kullanım alanları:

- Derin ağlarda, özellikle görüntü sınıflandırma görevlerinde



# Yaygın Aktivasyon Fonksiyonları

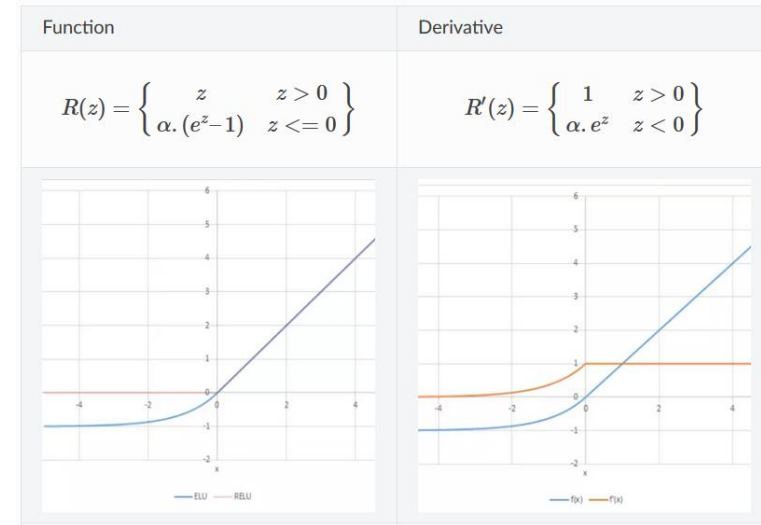
## Exponential Linear Unit (ELU):

### Özellikleri:

- Negatif girdiler için yumuşak doygunluk
- Ortalama aktivasyonu sifıra daha yakın
- ReLU'ya göre gürültüye daha dayanıklı

### Kullanım alanları:

- Derin ağlarda, özellikle hızlı öğrenme ve yüksek doğruluk gerektiren durumlarda



`keras.activations.elu(x, alpha=1.0)`



# Yaygın Aktivasyon Fonksiyonları

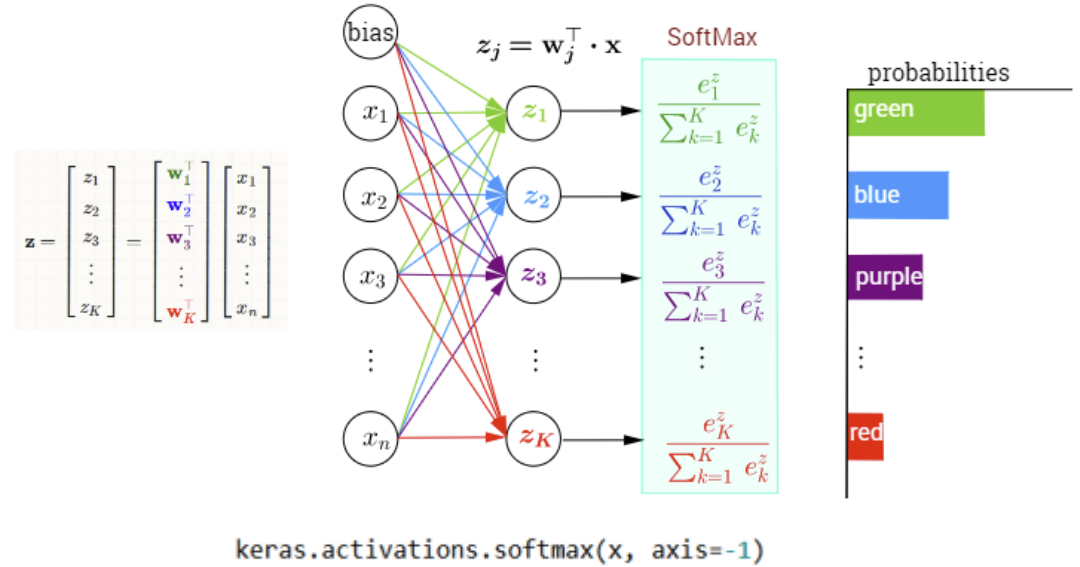
## Softmax:

### Özellikleri:

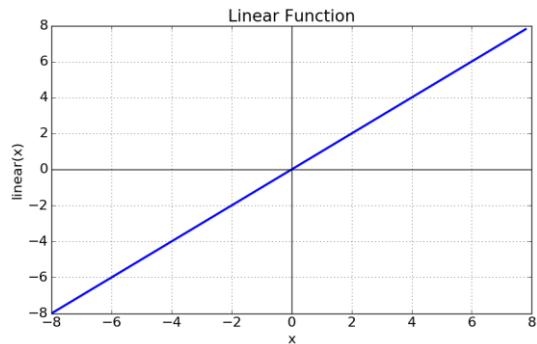
- Çıktıları olasılık dağılımına dönüştürür
- Tüm çıktıların toplamı 1'dir

### Kullanım alanları:

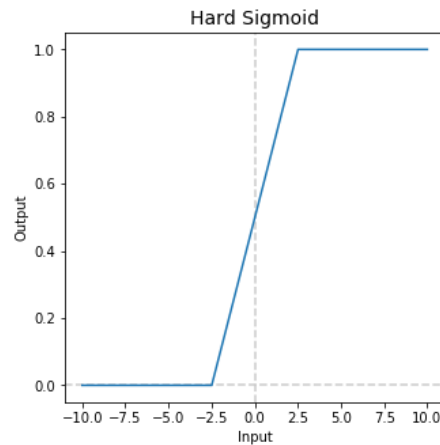
- Çok sınıflı sınıflandırma problemlerinde çıkış katmanı



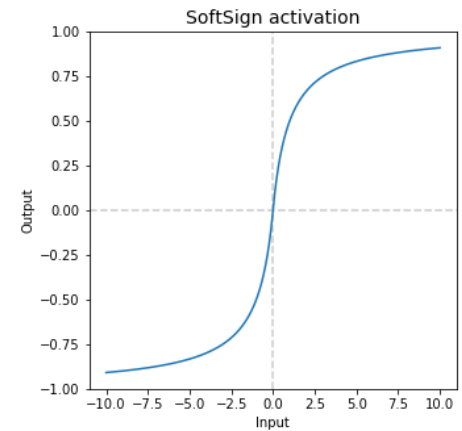
# Yaygın Aktivasyon Fonksiyonları



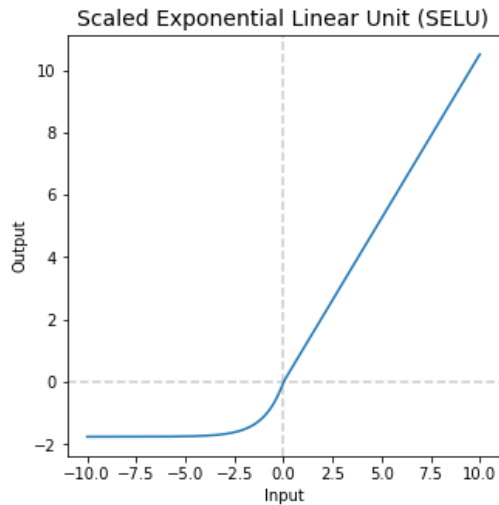
`keras.activations.linear(x)`



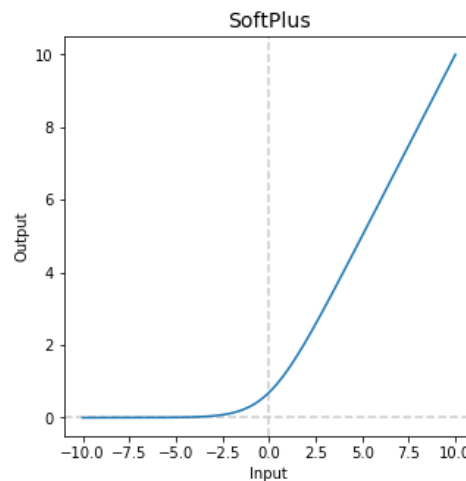
`keras.activations.hard_sigmoid(x)`



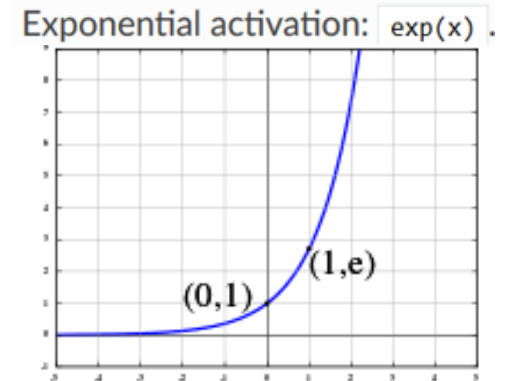
`keras.activations.softsign(x)`



`keras.activations.selu(x)`



`keras.activations.softplus(x)`



`keras.activations.exponential(x)`

# Aktivasyon Fonksiyonu Seçimi

- Aktivasyon fonksiyonları, derin öğrenme modellerinin performansını önemli ölçüde etkiler. Doğru aktivasyon fonksiyonunu seçmek, modelin öğrenme kapasitesini artırabilir ve eğitim sürecini hızlandırabilir.
- Aktivasyon fonksiyonu seçimi, problemin doğasına ve ağıın mimarisine bağlıdır.
- Ancak, en iyi sonucu elde etmek için genellikle deneysel yaklaşım ve farklı seçeneklerin denenmesi gerekir.

## Genel öneriler:

- Gizli katmanlar için ReLU veya varyasyonlarını kullanın
- Gradyan akışı sorunları yaşıyorsanız, Leaky ReLU veya ELU deneyin
- Çıkış katmanı için:
  - İkili sınıflandırma: Sigmoid
  - Çok sınıflı sınıflandırma: Softmax
  - Regresyon: Doğrusal aktivasyon

# Kaynaklar

- <https://hadrienj.github.io/posts/Deep-Learning-Book-Series-2.1-Scalars-Vectors-Matrices-and-Tensors/>
- [https://tr.d2l.ai/chapter\\_preliminaries/linear-algebra.html#sec-linear-algebra](https://tr.d2l.ai/chapter_preliminaries/linear-algebra.html#sec-linear-algebra)
- [https://tr.d2l.ai/chapter\\_appendix-mathematics-for-deep-learning/index.html](https://tr.d2l.ai/chapter_appendix-mathematics-for-deep-learning/index.html)
- <https://www.quantstart.com/articles/scalars-vectors-matrices-and-tensors-linear-algebra-for-deep-learning-part-1/>
- <https://keras.io/activations/>