

# GİRİŞ

Eğitmen  
Özgür YILDIRIM



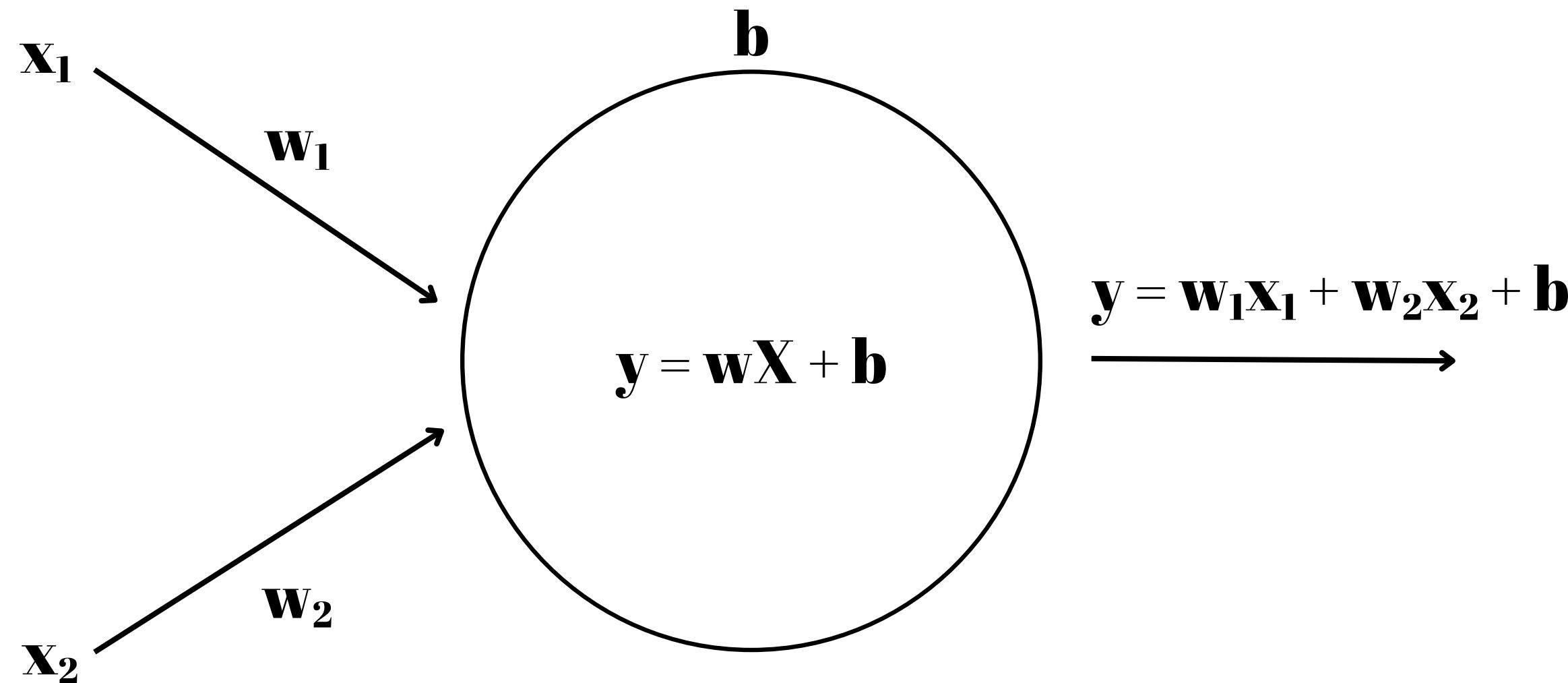
[linkedin.com/in/Ozgur-yldrm/](https://www.linkedin.com/in/Ozgur-yldrm/)

[github.com/OzgurYldrm](https://github.com/OzgurYldrm)

**Sunum + Notebook + Homework → Github**

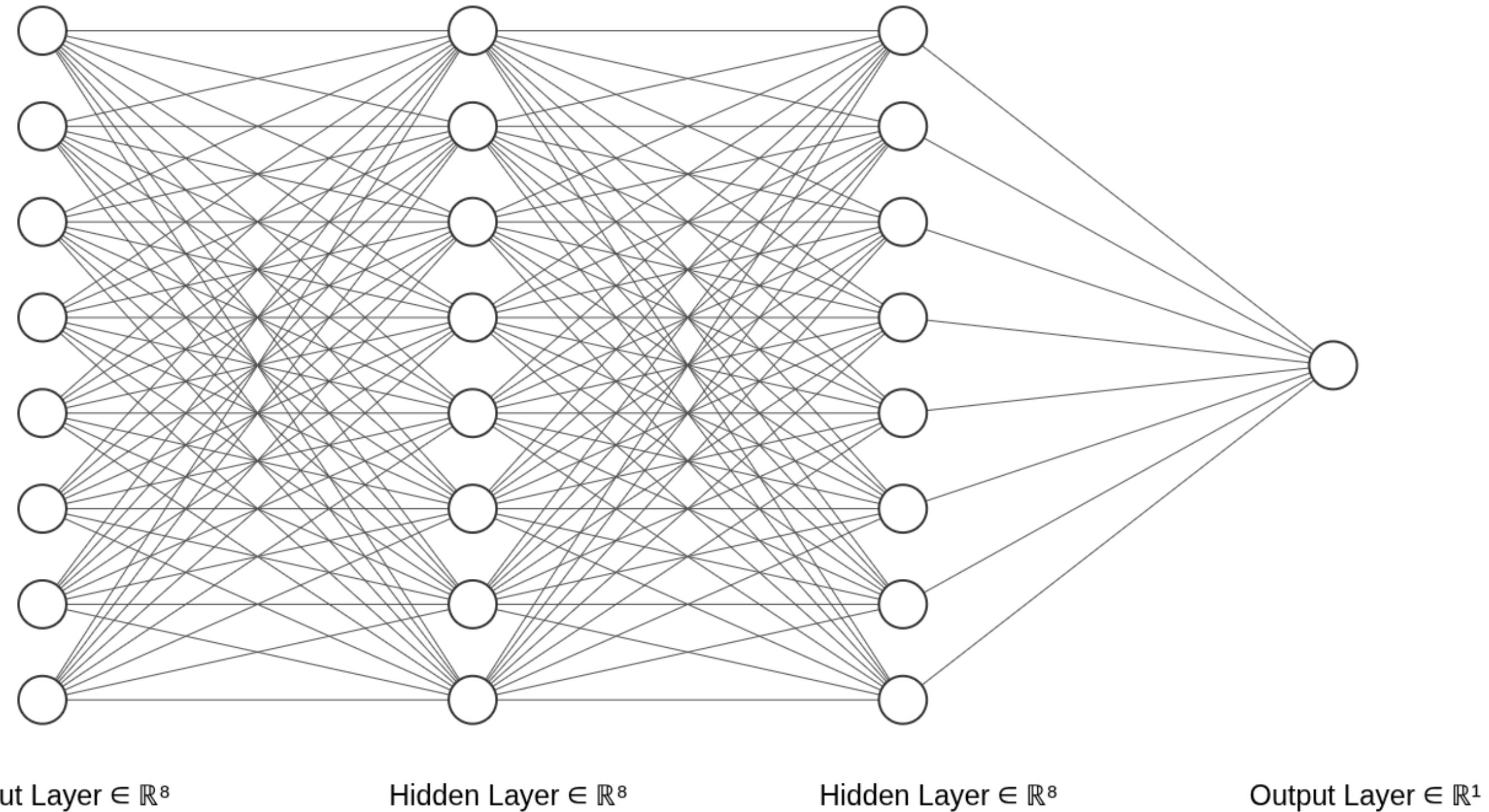
- **Neural Network**
- **Activation Functions**
- **Backpropagation**
- **Regularization**
- **Optimizers**
- **Normalization**
- **Dropout**
- **LR Scheduler**
- **Training Model**

# Neuron



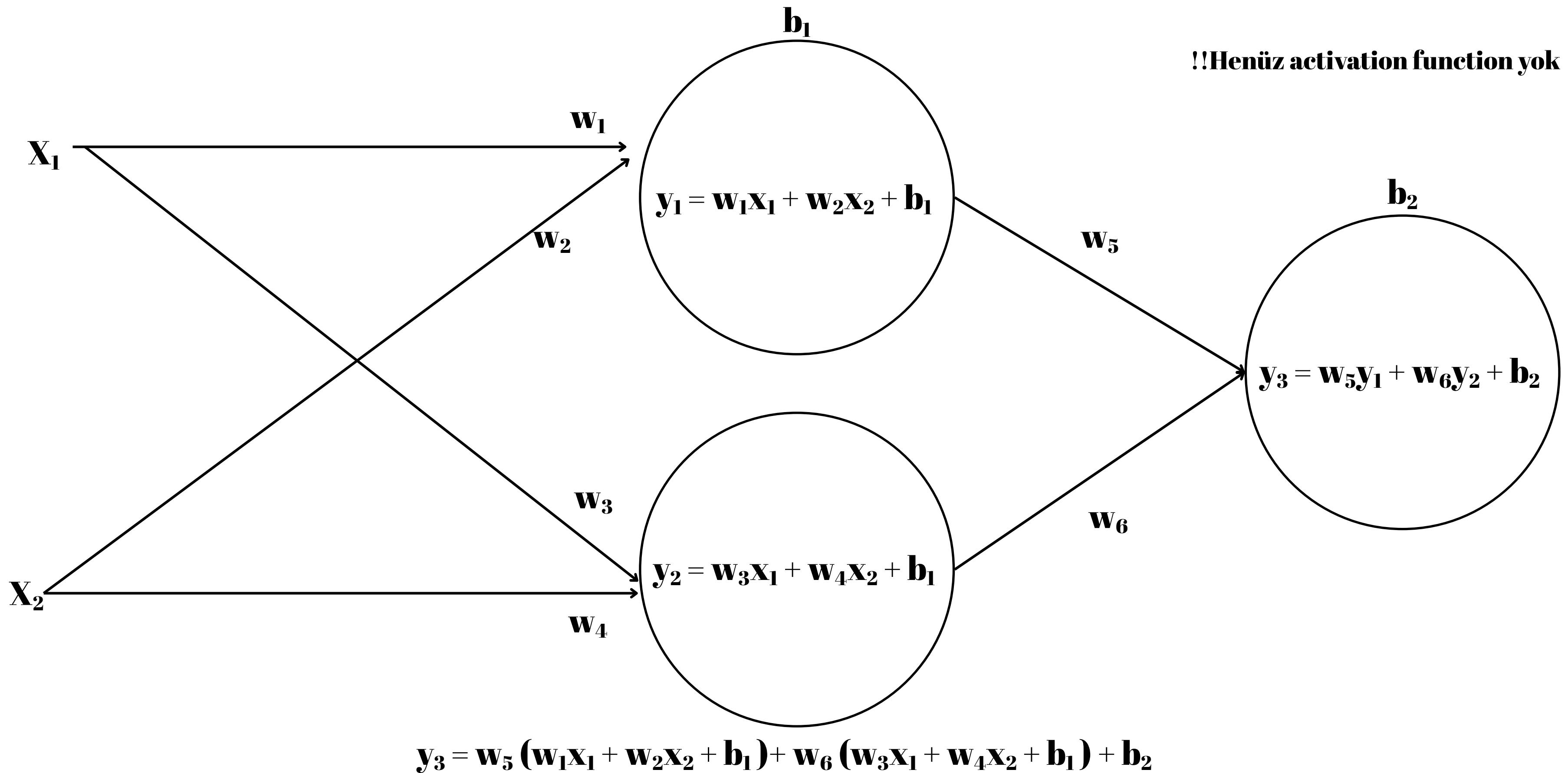
**Yukarıdaki neuron gösterimi temsilidir. Bu yapıları fiziksel bir varlık olarak düşünmeyiniz. Sadece matematiksel modelleri uzun uzun yazmak yerine bu şekilde göstermek ağ yapısını anlamaya yardımcı oluyor.**

# Neural Network

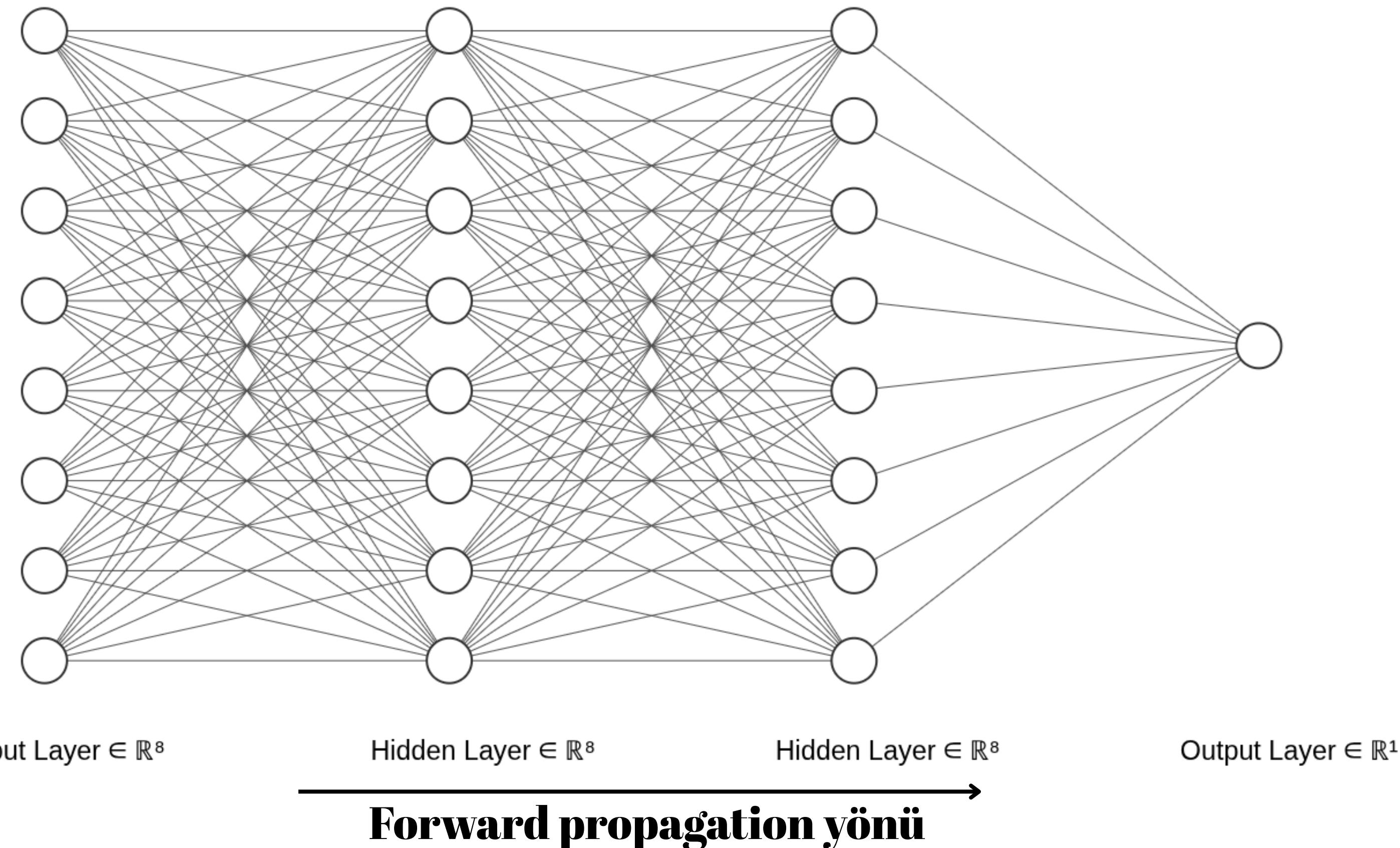


- **Her bir çizgi  $wx$  olarak düşünülebilir.**
- **Her nöron kendine gelen “ $wx$ ” değerlerini toplayıp bias ekleyip bazı dönüşümler uygulayıp sonraki katmana doğru iletir.**

# Forward Propagation



# Forward Propagation



**Output layer çıktısı modelin tahmini (prediction) olarak adlandırılır.**

## Activation Function

- NN'lerin en büyük problemlerinden biri her katman doğrusal bir işlem yaptığı için kaç tane nöron olursa olsun son durumda tek bir linear fonksiyon şeklinde yazılabilirler.**

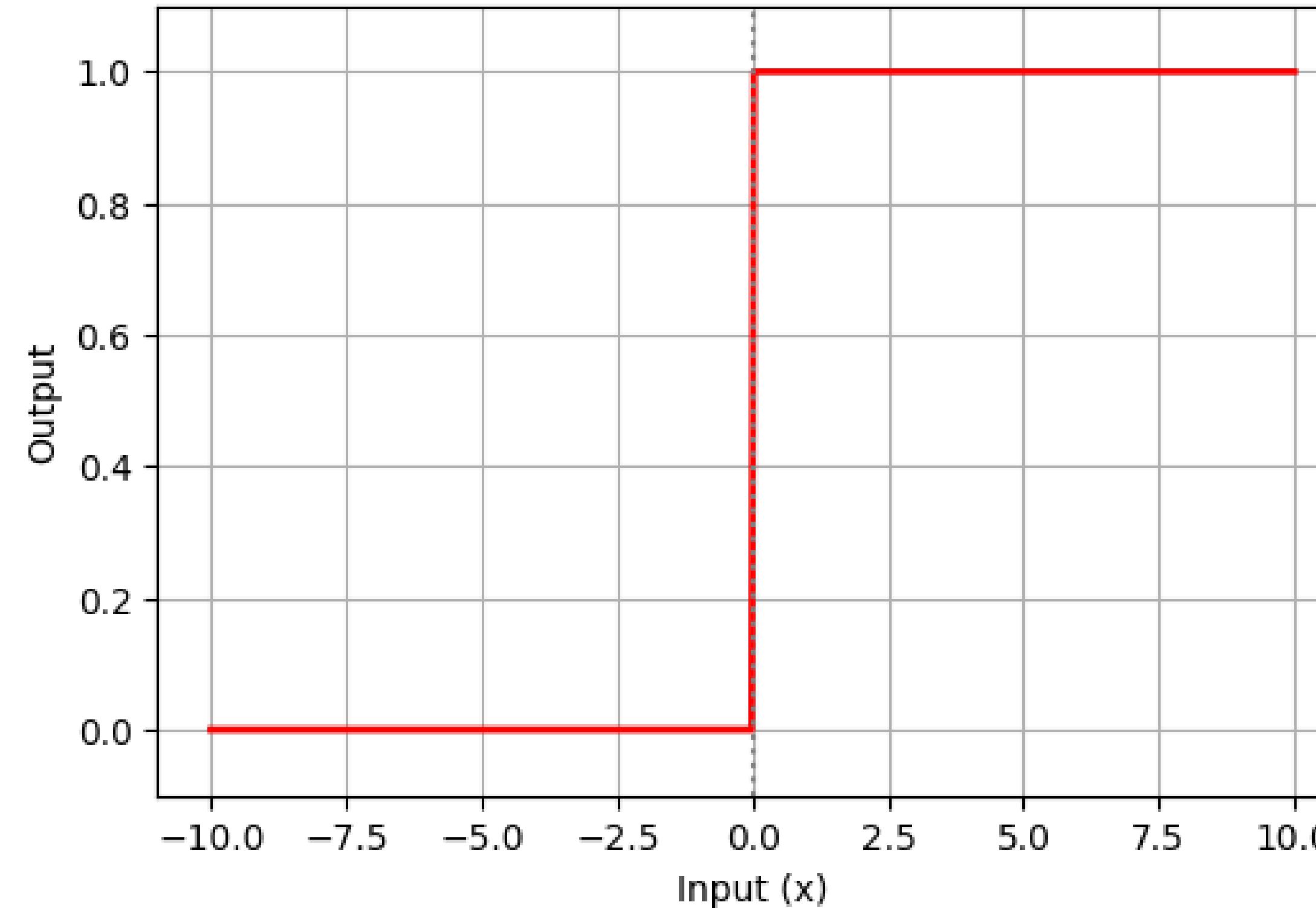
$$y_3 = w_5 (w_1x_1 + w_2x_2 + b_1) + w_6 (w_3x_1 + w_4x_2 + b_1) + b_2$$

- Gerçek hayat problemleri doğrusal değildir. Bu yüzden yapay sinir ağlarının doğrusal olmayan ilişkileri öğrenmesi için bazı özel fonksiyonlar kullanılır: Activation Function**
- Bu fonksiyonların karmaşık ilişkileri yakalayabilecek potansiyelde, türevlenebilir olması önemlidir.**

Threshold like	Threshold	Sign	Softsign	TanhShrink	Softshrink	HardShrink				
Sigmoid like	Sigmoid	logsigmoid	tanh	hardsigmoid	hardtanh					
ReLU like	ReLU	pReLU	LeakyReLU	RELU6	RRELU	Softplus	GELU	Swish	Hardswish	Mish
ELU like	ELU	SELU	CELU							
Softmax like	Softmax	LogSoftmax	Softmin							

# Threshold

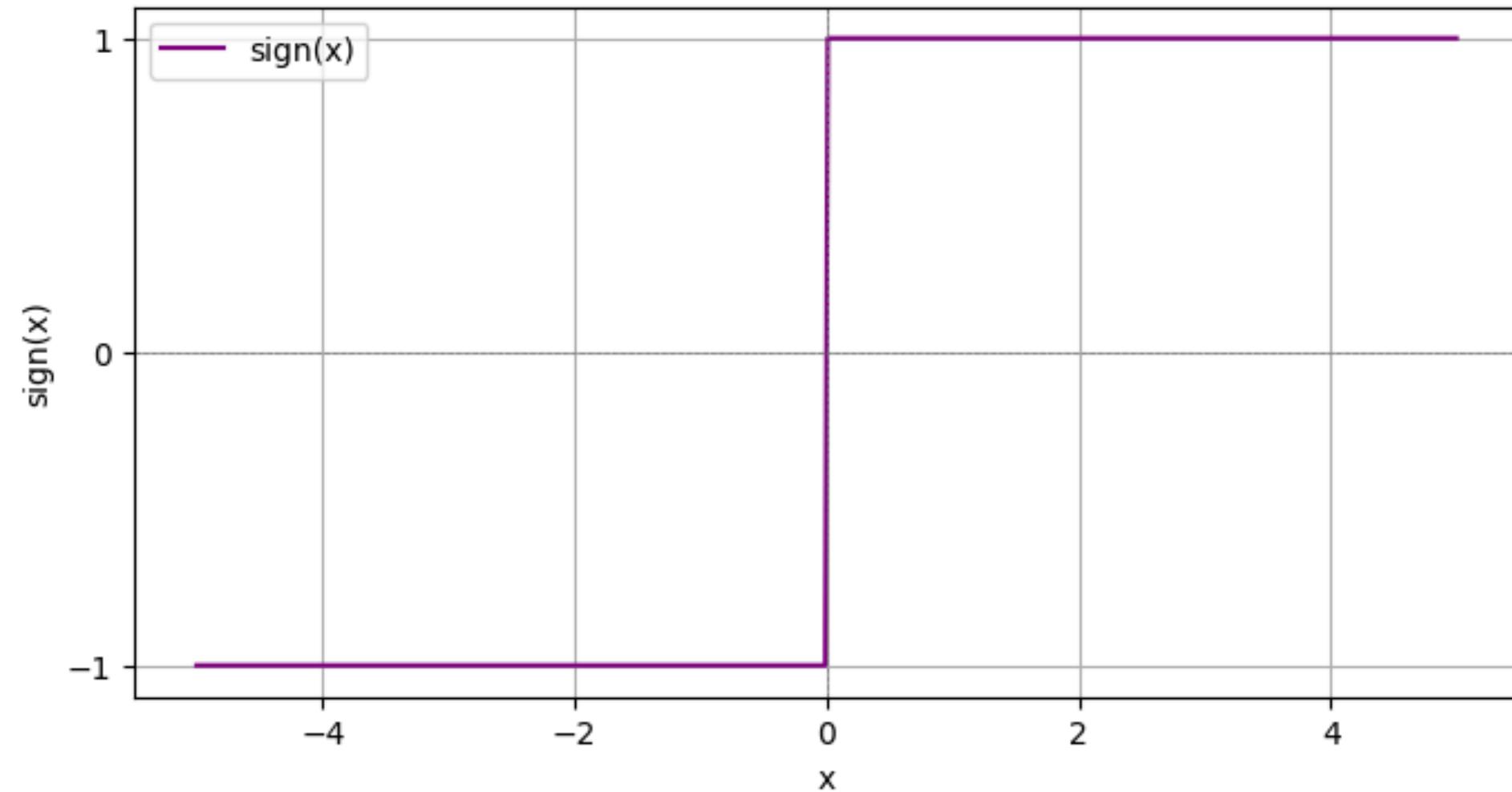
Threshold Activation Function



- ML'in ilk zamanları kullanılmış.
- En basic activation function
- Türevinin olmaması öğrenme durumunu olumsuz etkilemektedir.
- Günümüzde çok sık kullanılmamaktadır.

# Sign - Softsign

Sign Activation Function



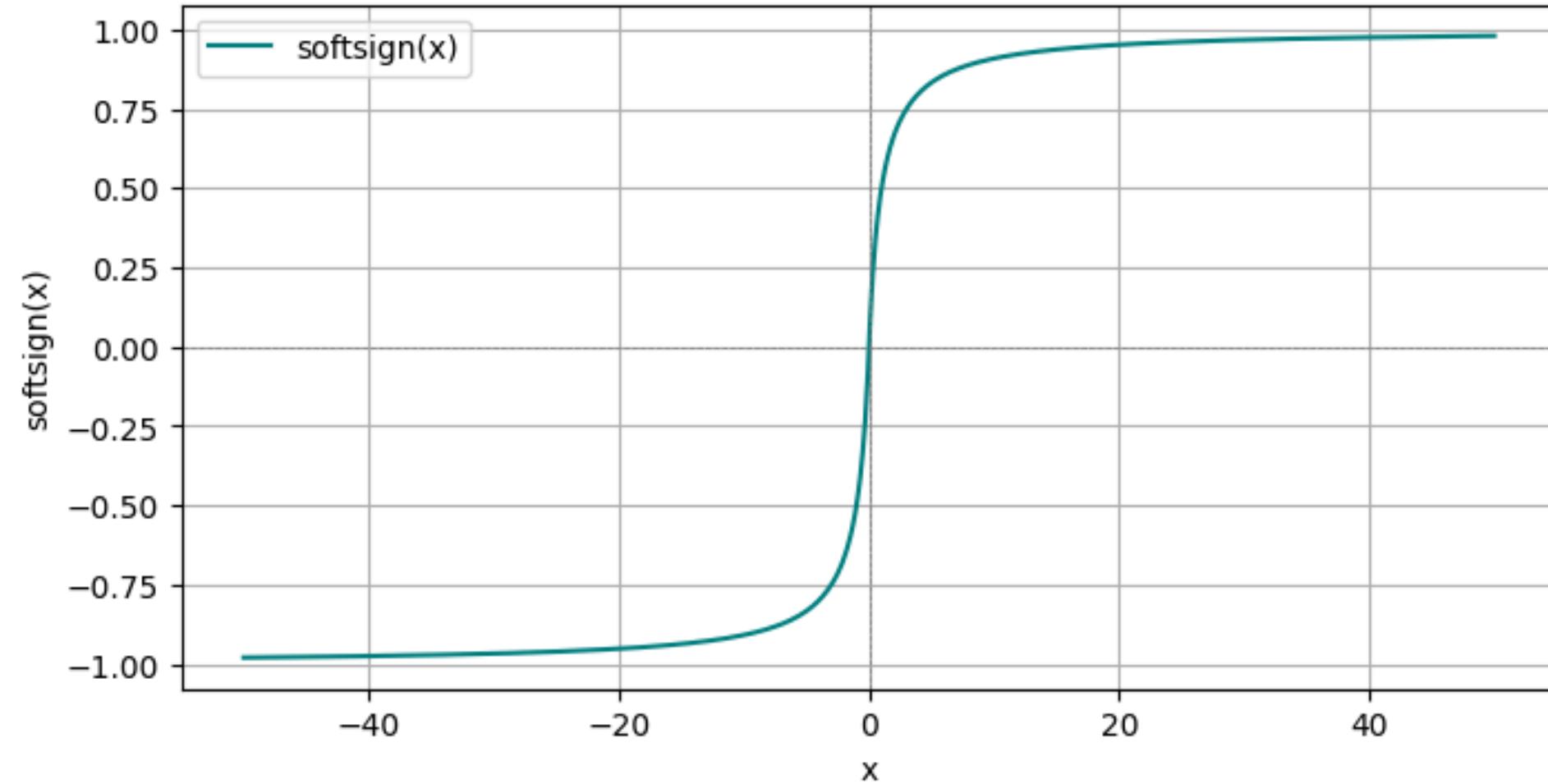
Sign Fonksiyonu Tanımı

$$\text{sign}(x) = -1 \quad \text{if } x < 0$$

$$\text{sign}(x) = 0 \quad \text{if } x = 0$$

$$\text{sign}(x) = +1 \quad \text{if } x > 0$$

Softsign Activation Function

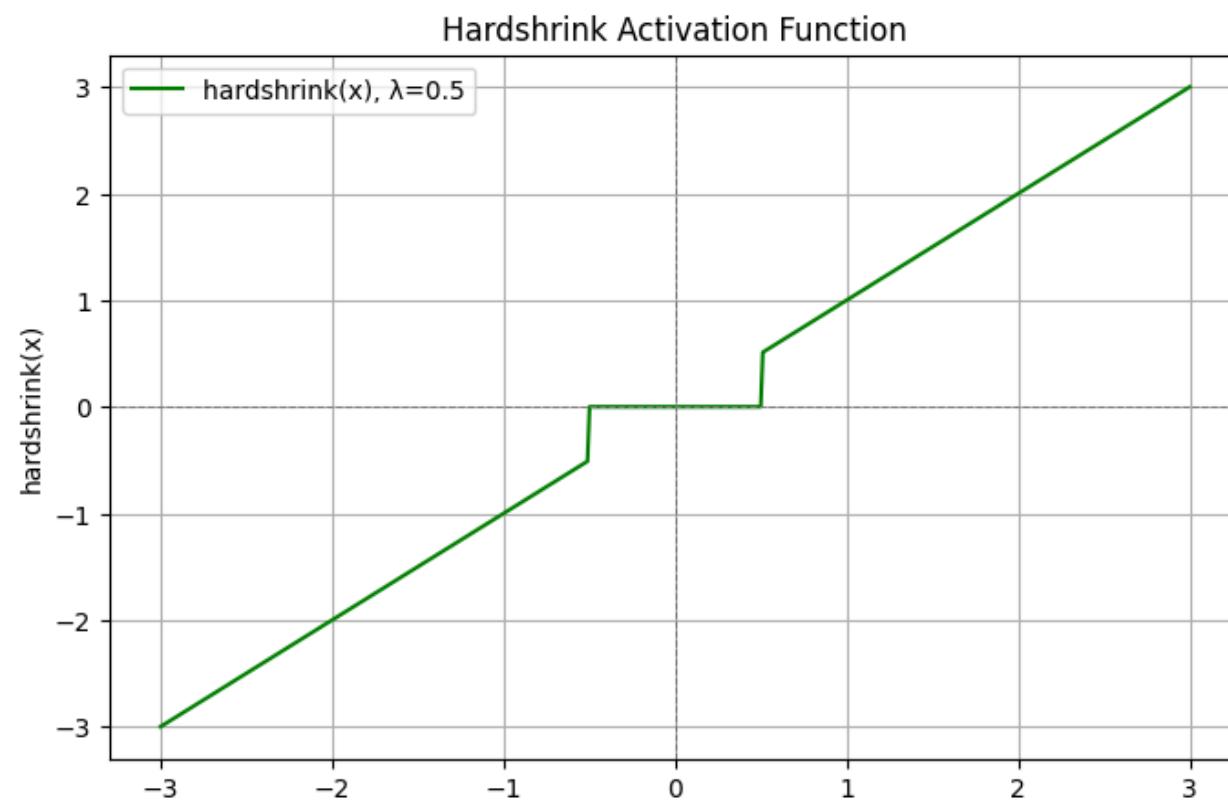


Softsign Fonksiyonu

$$\text{softsign}(x) = \frac{x}{1 + |x|}$$

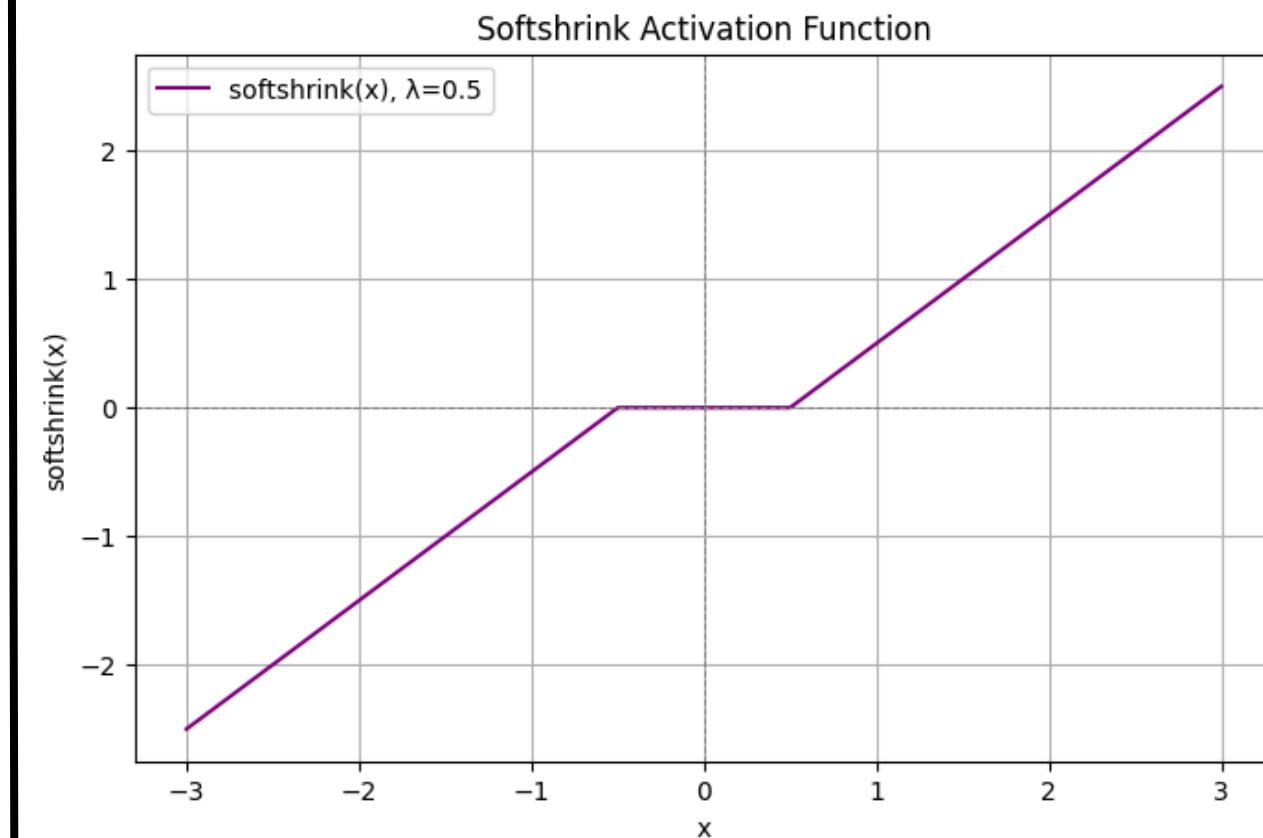
# Shrink

## Hardshrink



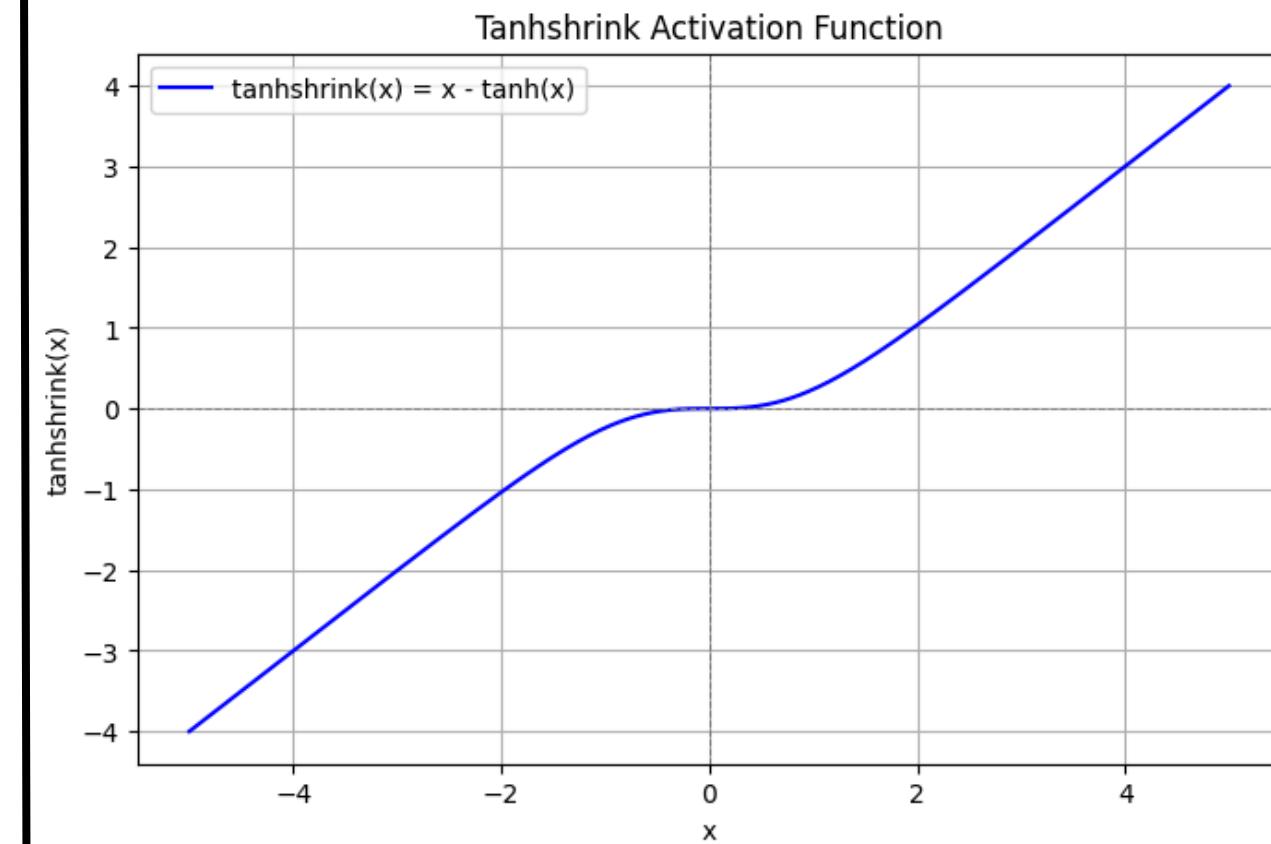
$$\begin{aligned} \text{hardshrink}(x) &= x && \text{if } |x| > \lambda \\ \text{hardshrink}(x) &= 0 && \text{if } |x| \leq \lambda \end{aligned}$$

## softshrink



$$\begin{aligned} \text{softshrink}(x) &= x - \lambda && \text{if } x > \lambda \\ \text{softshrink}(x) &= x + \lambda && \text{if } x < -\lambda \\ \text{softshrink}(x) &= 0 && \text{otherwise} \end{aligned}$$

## tanh shrink



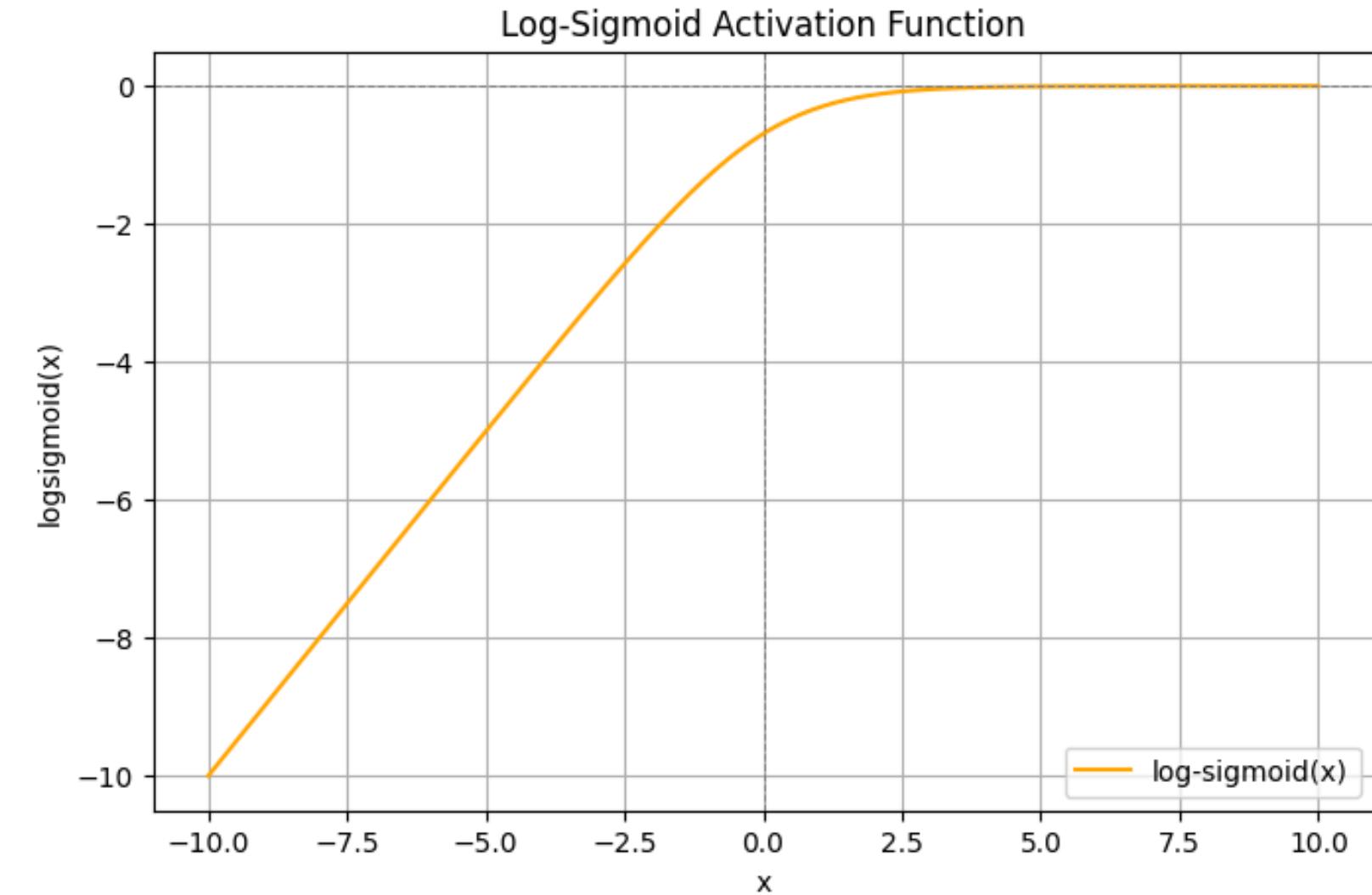
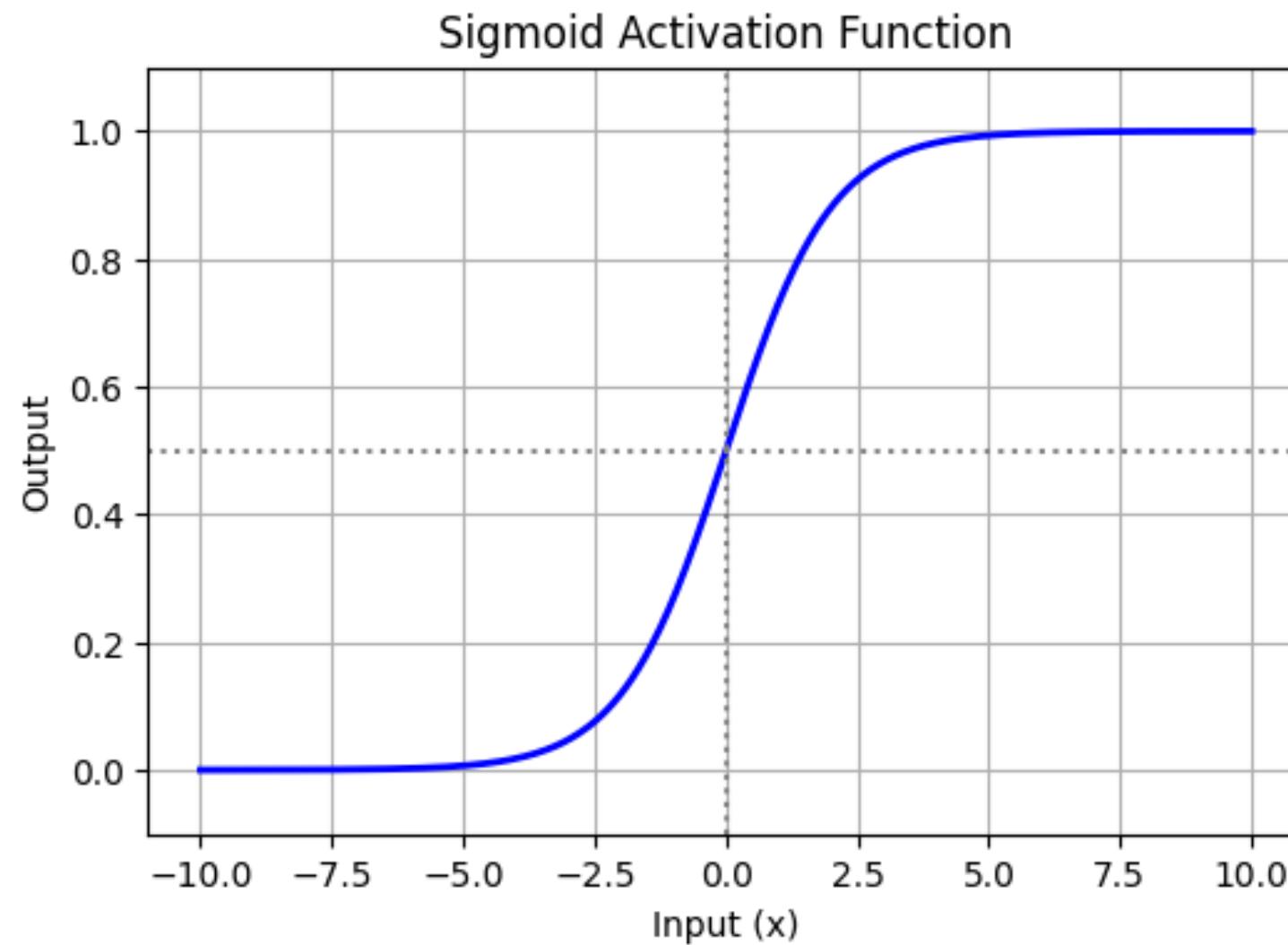
$$\text{tanhshrink}(x) = x - \tanh(x)$$

Temel mantığı 0'a çok yakın değerleri 0' a yuvarlamaktır.

Shrink fonksiyonu sparse gösterim elde etmek için kullanılır.

Bazı durumlarda gürültüleri (çok küçük anlamsız değerleri) elimine etmek için kullanılır

# Sigmoid - LogSigmoid

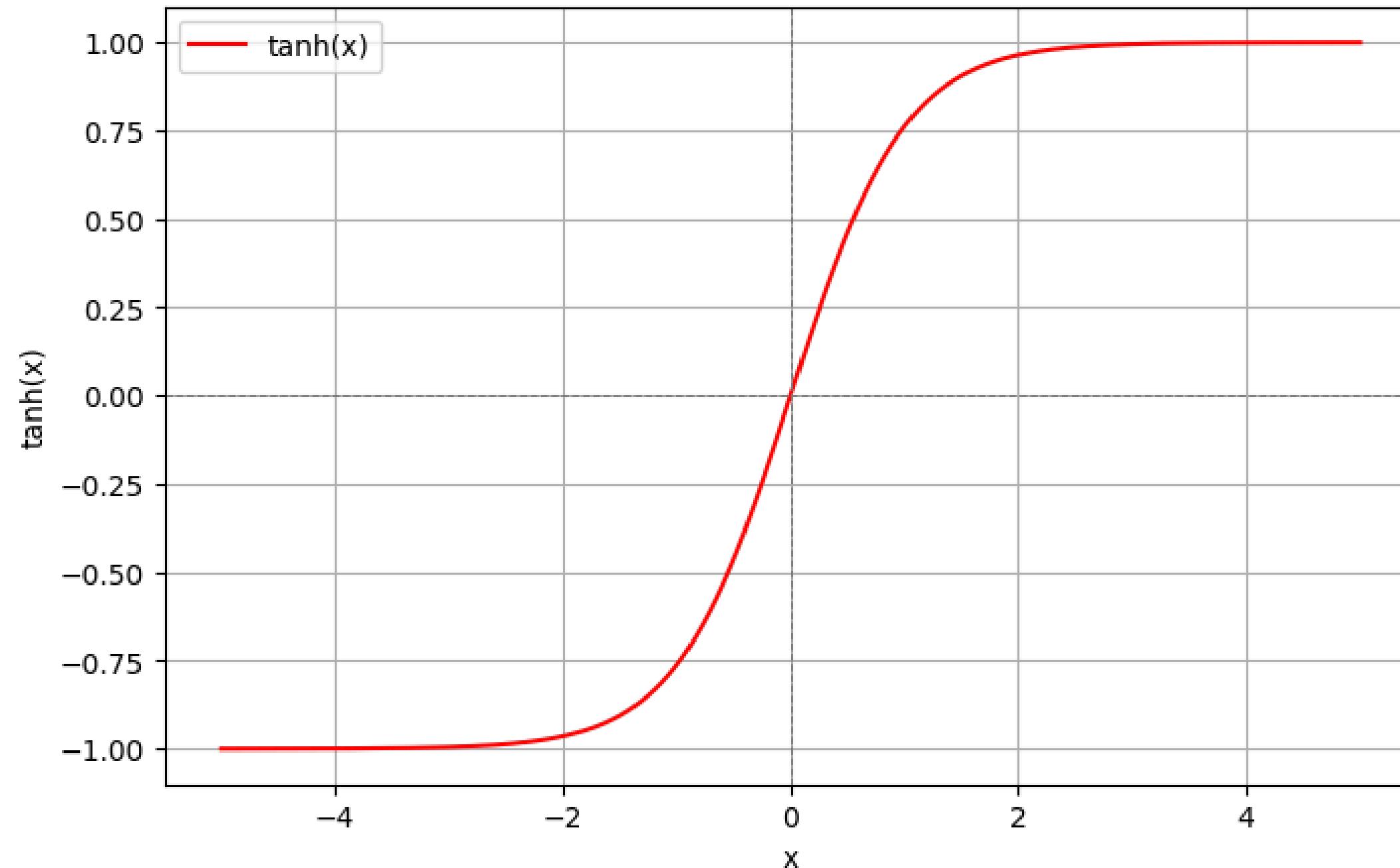


- **Sürekli ve türevlenebilir.**
- **Doğrusal değildir. Kompleks ilişkileri öğrenebilir.**
- **Genelde çıkış katmanında olasılıksal tahmin için kullanılır.**
- **Üç noktalarda türev sıfıra yaklaşır. Vanishing gradient problemi yaşayabilir.**

- **Sigmoid fonksiyonunun logaritmasıdır.**
- **Sigmoid çıktılarını logaritmik ölçekte ifade eder.**
- **Sayısal olarak daha stabildir**

# tanh

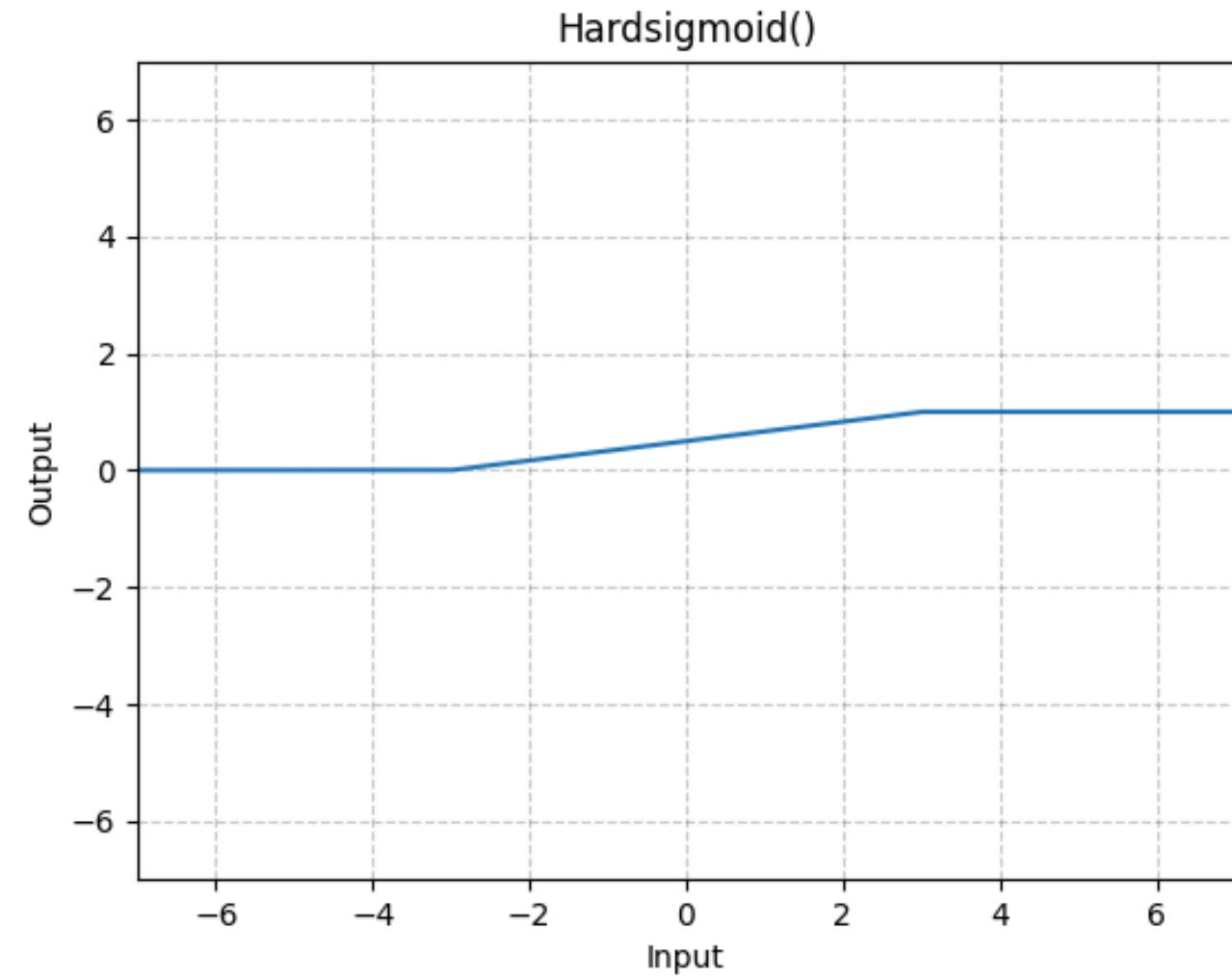
Tanh Activation Function



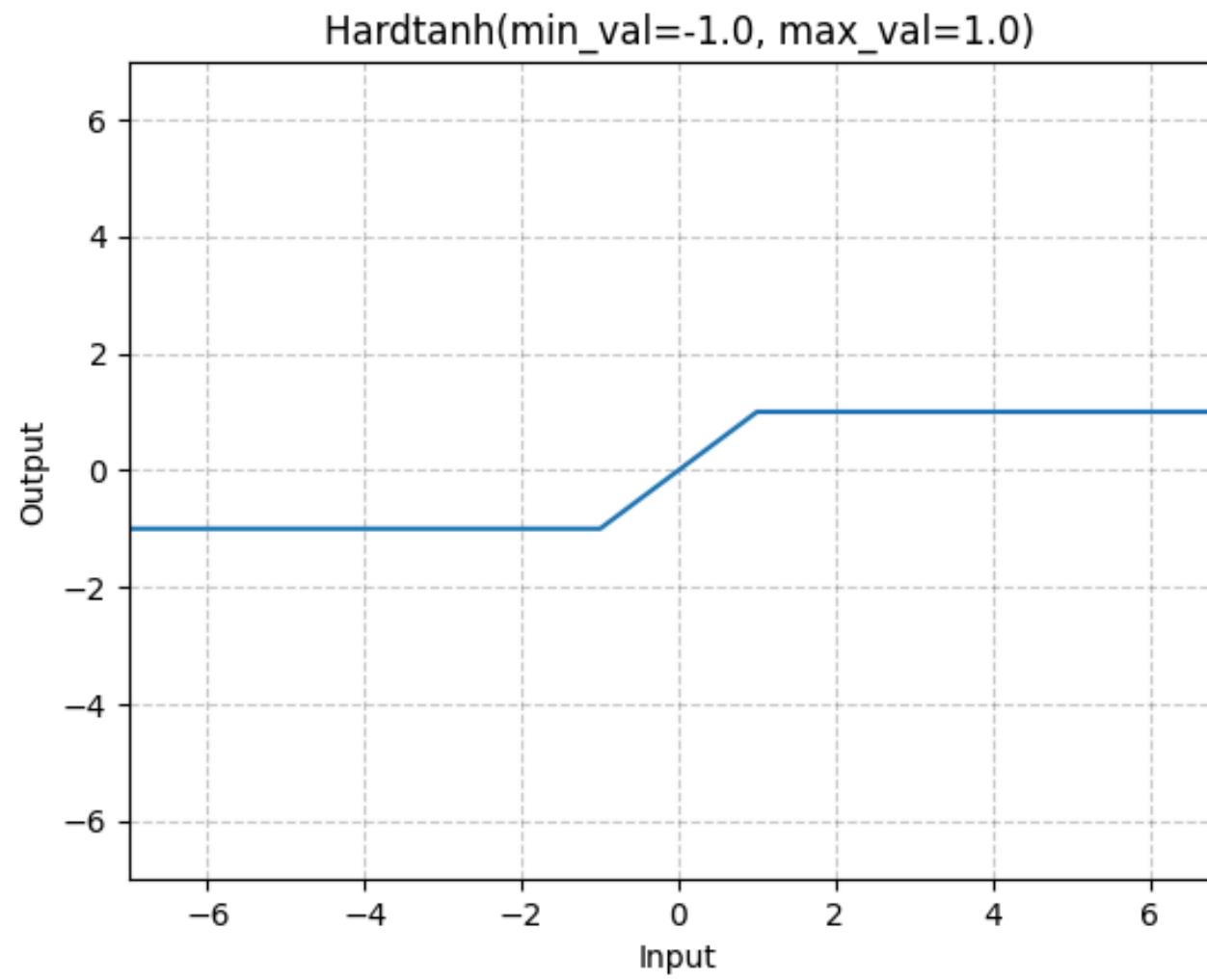
- Çıktısı **-1 ile +1 arasında değişir.**
- **0 merkezli çıktı verir. Sigmoid'e göre sinir ağının daha kararlı öğrenmesini sağlayabilir.**
- **Pozitif-negatif simetrik verilerde etkili olabilir.**

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

# Hardsigmoid - Hardtanh



$$\text{Hardsigmoid}(x) = \begin{cases} 0 & \text{if } x \leq -3, \\ 1 & \text{if } x \geq +3, \\ x/6 + 1/2 & \text{otherwise} \end{cases}$$

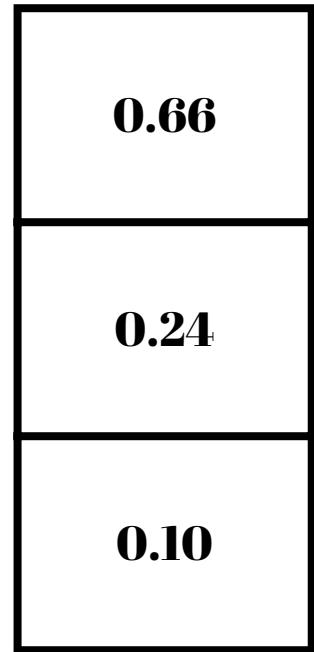
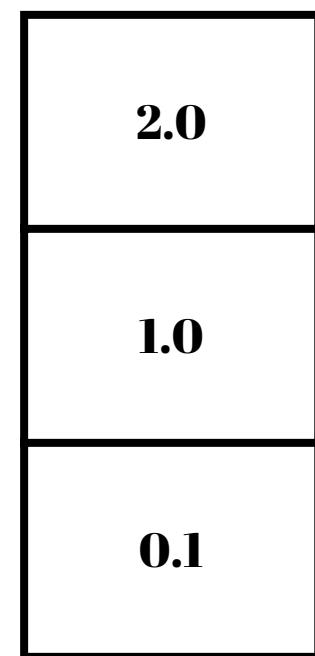


$$\text{HardTanh}(x) = \begin{cases} \text{max\_val} & \text{if } x > \text{max\_val} \\ \text{min\_val} & \text{if } x < \text{min\_val} \\ x & \text{otherwise} \end{cases}$$

- Sigmoid değerlerini yaklaşık olarak tahmin eden bir fonksiyondur.
- Hesaplaması kolay olduğu için kullanılabilir. Eğitim sürecini hızlandırır.

- tanh değerlerini yaklaşık olarak tahmin eden bir fonksiyondur.
- Hesaplaması kolay olduğu için kullanılabilir. Eğitim sürecini hızlandırır.

# Softmax



$$\begin{aligned} e^{2.0} &= 7.38 \\ e^{1.0} &= 2.71 \\ e^{0.1} &= 1.10 \end{aligned}$$

{

11.21

$$\begin{aligned} y_1 &= (7.38 / 11.21) \\ y_2 &= (2.71 / 11.21) \\ y_3 &= (1.10 / 11.21) \end{aligned}$$

$$\text{softmax}(z)_i = \frac{e^{z_i}}{\sum_{j=1}^n e^{z_j}}$$

- **Softmax girdileri toplamları 1 olacak şekilde sınıfsal olasılığa dönüştürür.**
- **Sigmoid 2'li sınıflandırma problemlerinde işe yarar ancak daha fazla sınıf olduğunda doğrudan kullanılamaz. Bu noktada softmax çoklu sınıflandırma problemlerinde kullanılabilir.**
- **Olasılıksal ve göreli çıktılar verir. Sınıfların birbirine göre yorumu yapılabilir. (Bu sınıf'a ait olma olasılığı daha fazla gibi)**

# LogSoftmax

$$\text{logsoftmax}(x_i) = \log\left(\frac{e^{x_i}}{\sum_{j=1}^n e^{x_j}}\right) = x_i - \log\left(\sum_{j=1}^n e^{x_j}\right)$$

2.0
1.0
0.1

$$\begin{aligned} e^{2.0} &= 7.38 \\ e^{1.0} &= 2.71 \\ e^{0.1} &= 1.10 \end{aligned}$$

} 11.21

$$\ln(11.21) = 2.41$$

$$y_1 = 2.0 - 2.41$$

$$y_2 = 1.0 - 2.41$$

$$y_3 = 0.1 - 2.41$$

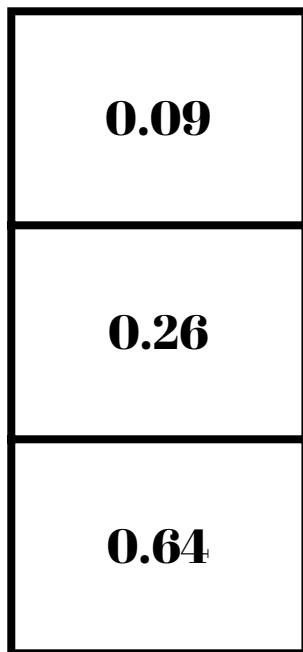
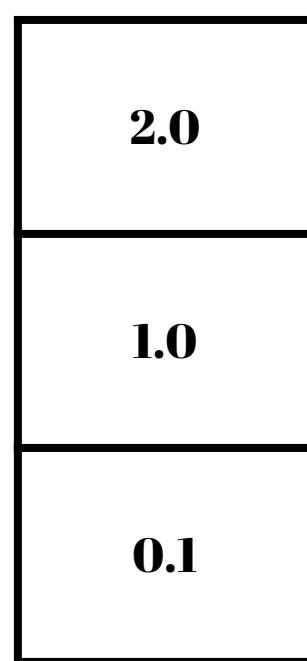
-0.41
-1.41
-2.31

- LogSoftmax softmax'in logaritma alınmış halidir.
- Numerical Stability daha yüksektir.
- İşlemsel olarak kullanılması mantıklıdır. Ancak tahmin üretmek için kullanılınlca yorum yapması zordur.

## Softmin

$$\text{Softmin}(x_i) = \frac{e^{-x_i}}{\sum_j e^{-x_j}}$$

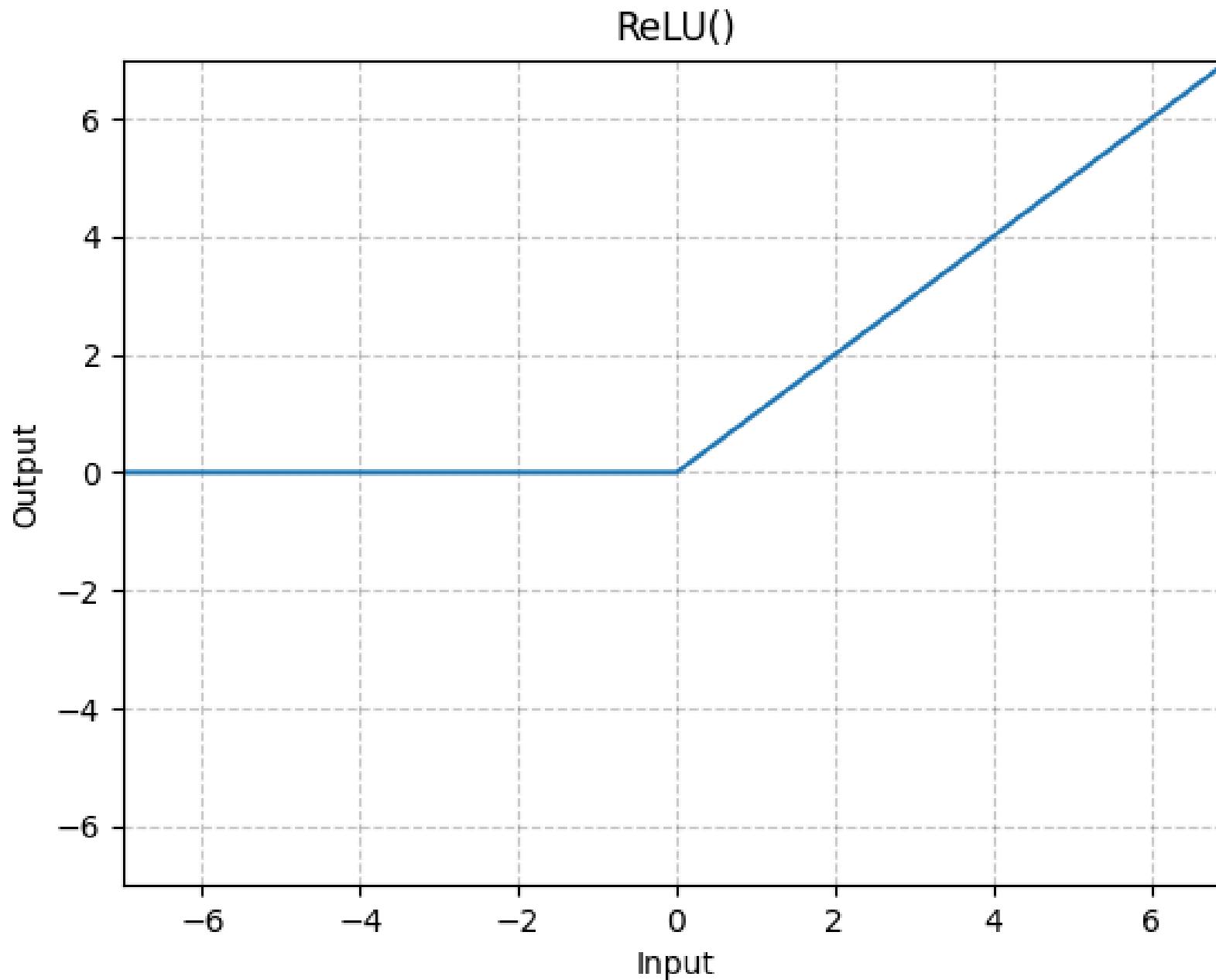
- Softmax'in ters işleyen halidir.
- Küçük girdiler olasılıksal çıktısı daha büyük olur



$$\left. \begin{array}{l} e^{-2.0} = 0.13 \\ e^{-1.0} = 0.36 \\ e^{-0.1} = 0.9 \end{array} \right\} 1.39 \quad \begin{aligned} y_1 &= (0.13/1.39) \\ y_2 &= (0.36/1.39) \\ y_3 &= (0.9/1.39) \end{aligned}$$

# Rectified Linear Unit (ReLU)

$$y = \max(0, x)$$

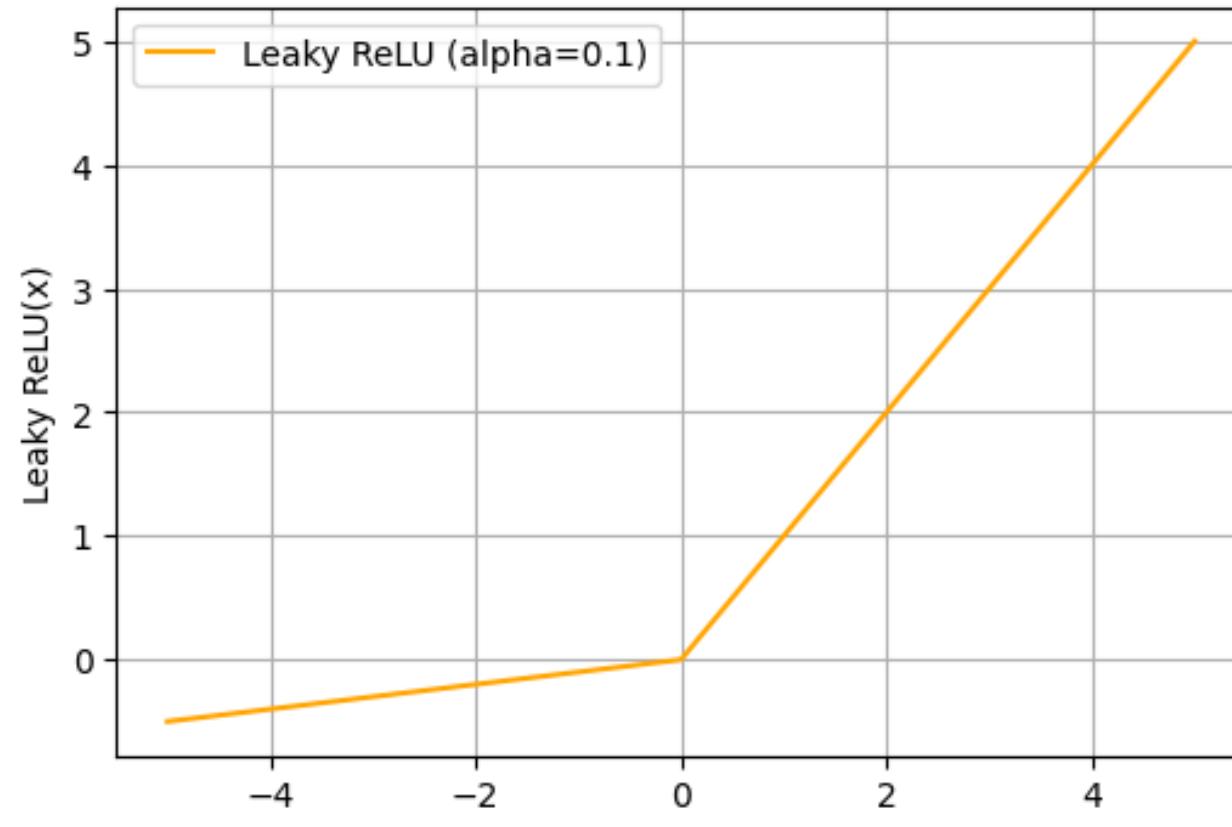


- **Doğrusal bir fonksiyon değil (tanım gereği)**
- **Hesaplaması kolaydır.**
- **0'dan büyük girdiler için türev 0'a yakın olmadığı için vanishing gradient problemi yaşamaz.**
- **0'dan küçük girdiler output 0 olduğu için ölü nöron problemi yaşanabilir. O nöron üzerinde öğrenme mümkün olmayabilir.**
- **Sparse output verir. Bazı durumlarda faydalı olabilir.**
- **Günümüzde default kullanılan aktivasyon fonksiyonudur.**

# pReLU - LeakyReLU - ReLU6

## LeakyReLU

Leaky ReLU Aktivasyon Fonksiyonu



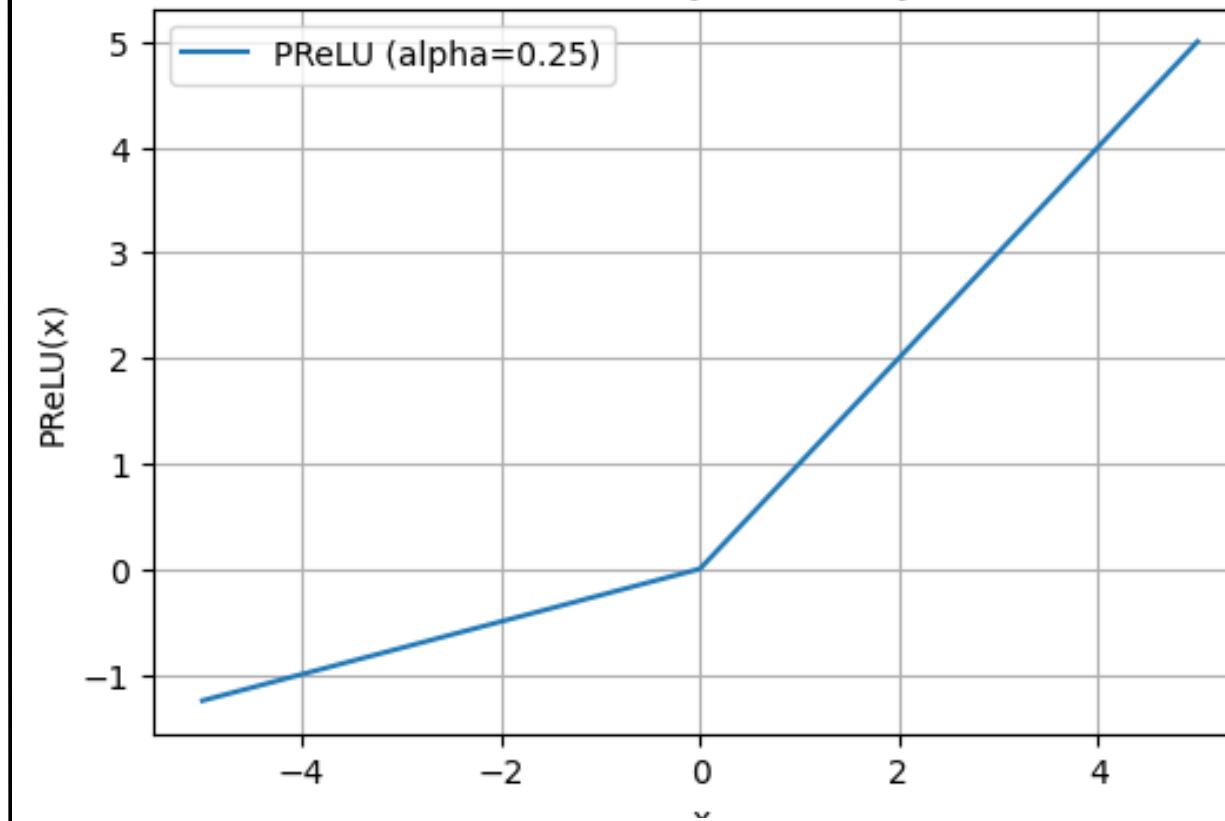
$$f(x) = \max(x, \alpha x)$$

**alpha** sabit

**Ölü nöron problemini çözmek**

## pReLU

pReLU Aktivasyon Fonksiyonu

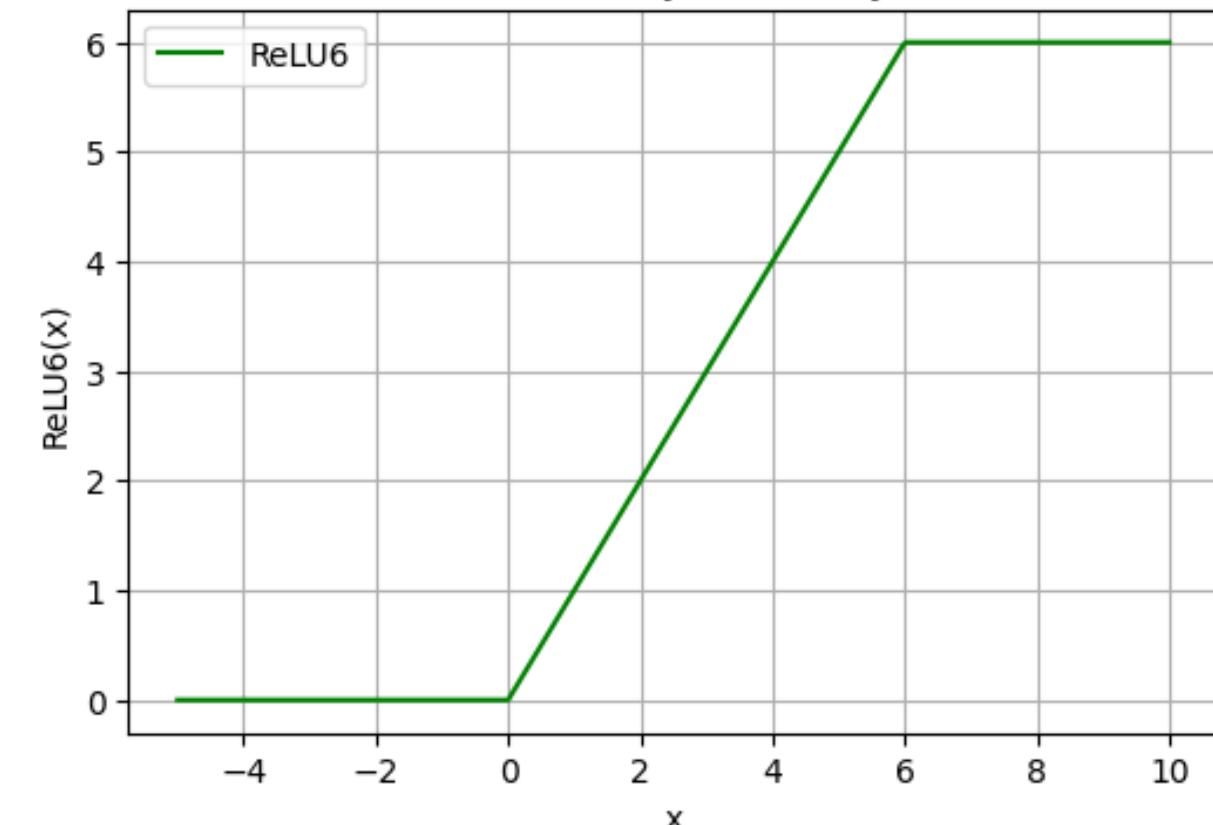


$$f(x) = \max(x, \alpha x)$$

**alpha** öğrenilebilir

## ReLU6

ReLU6 Aktivasyon Fonksiyonu



$$f(x) = \min(\max(0, x), 6)$$

**üst sınır 6'dır**

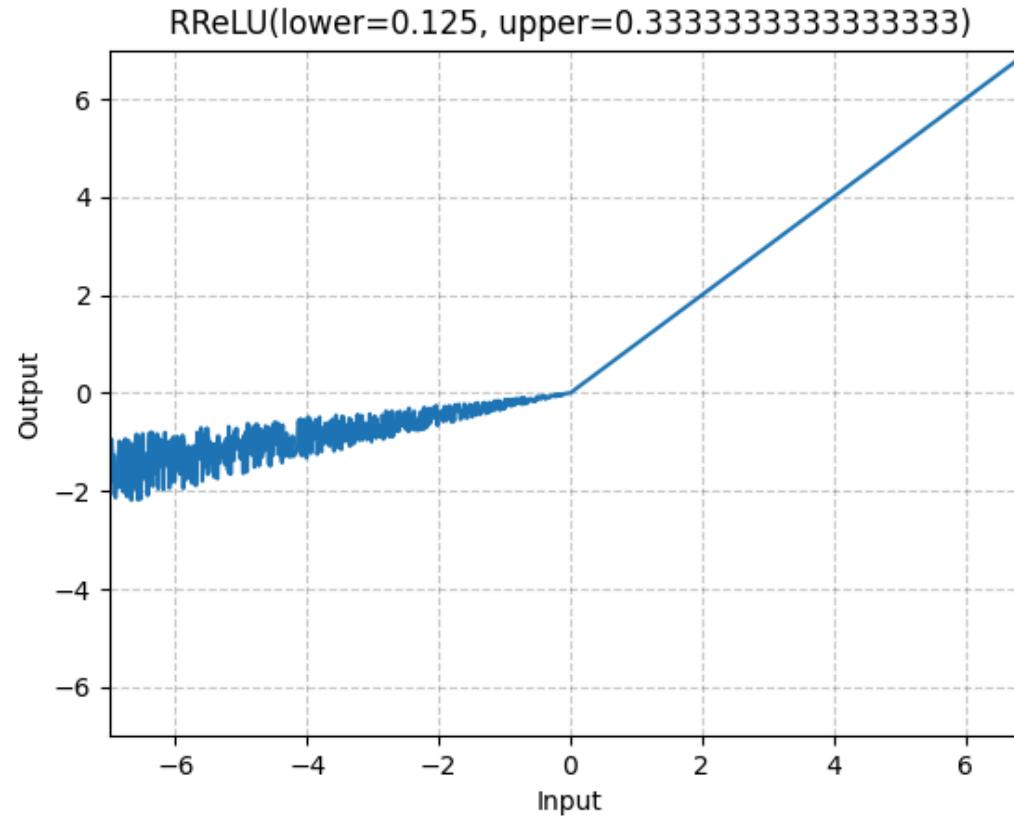
\*Rectifier Nonlinearities Improve Neural Network Acoustic Models (LReLU)

\*Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification (PReLU)

\*Convolutional Deep Belief Networks on CIFAR-10

# RReLU - Softplus - GELU

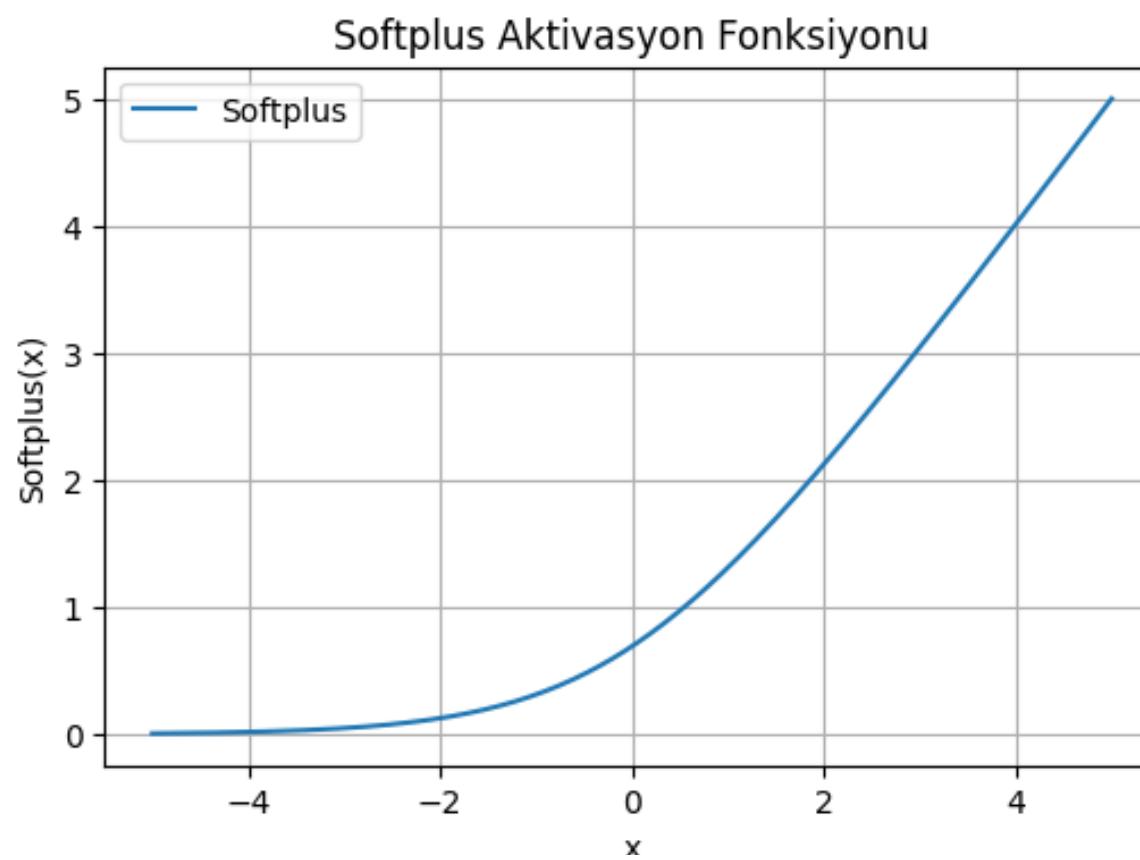
## RReLU



$$f(x) = \max(x, \alpha x)$$

**Negatif kısım random olduğu için overfittingi önleme şansı var**

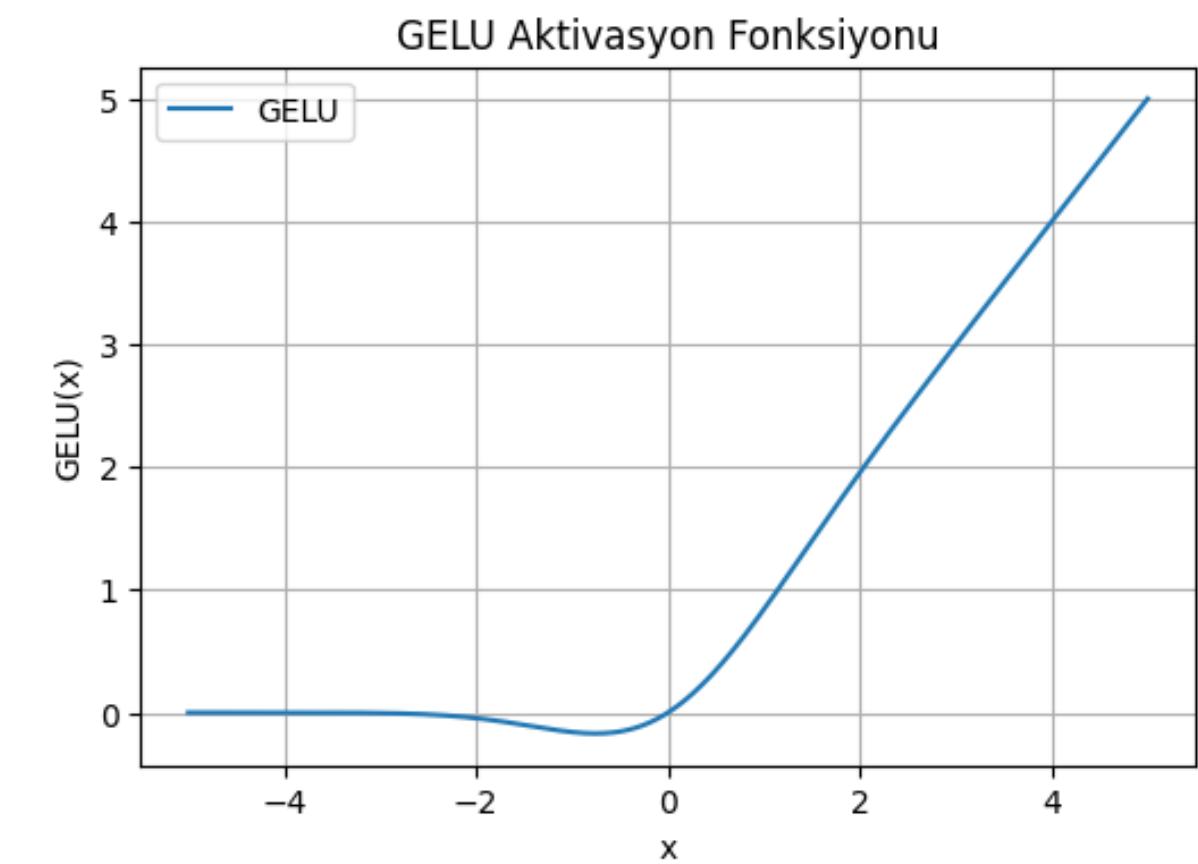
## Softplus



$$f(x) = \frac{1}{\beta} \log(1 + e^{\beta x})$$

**smooth approximation to ReLU**

## GELU



$$f(x) = 0.5x \left(1 + \tanh\left(\sqrt{\frac{2}{\pi}}(x + 0.044715x^3)\right)\right)$$

**Transformer mimarilerinde kullanılabilir.**

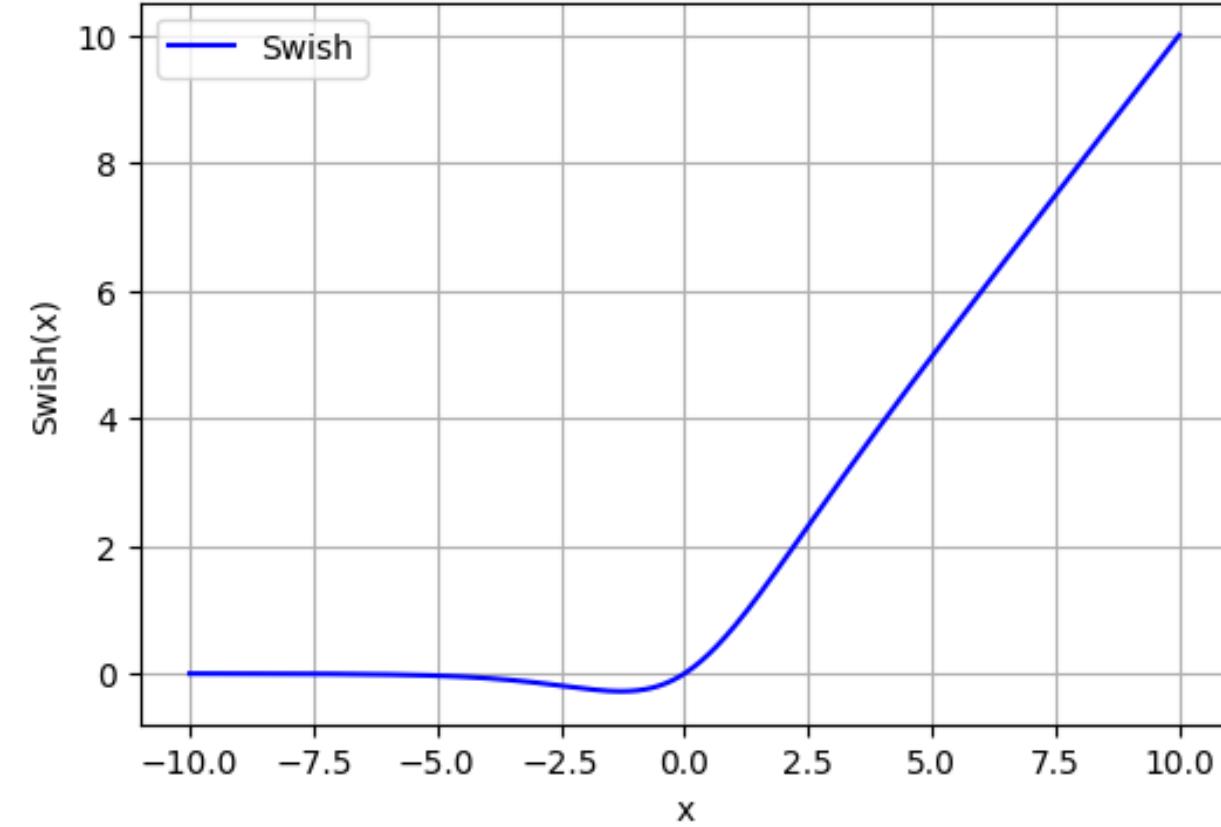
**\*Gaussian Error Linear Units (GELUs)**

**\*Empirical Evaluation of Rectified Activations in Convolutional Network**

# Swish - HardSwish - Mish

## SiLU - Swish

Swish Aktivasyon Fonksiyonu

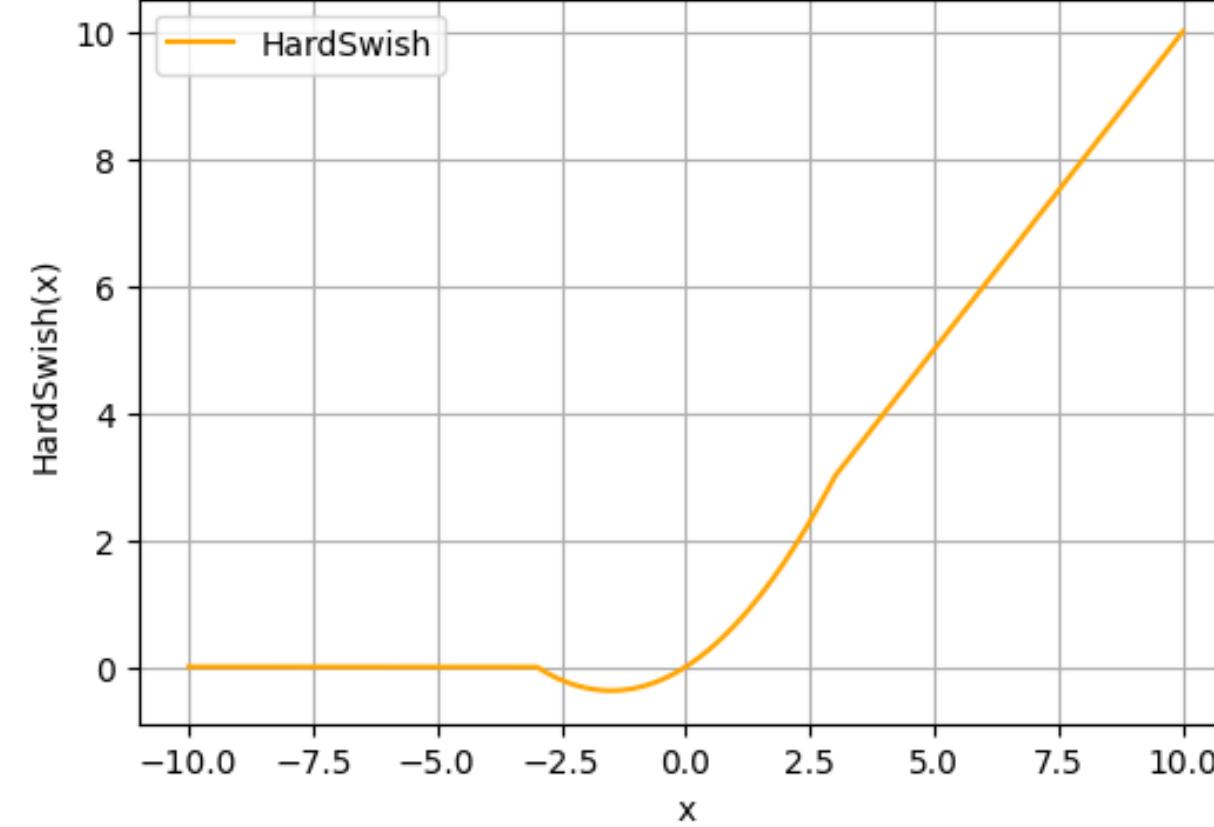


$$\text{Swish}(x) = x \cdot \sigma(x) = \frac{x}{1 + e^{-x}}$$

Pozitif kısımda neredeyse lineer ve negatif kısımda bir miktar çıktı verir

## Hardswish

HardSwish Aktivasyon Fonksiyonu

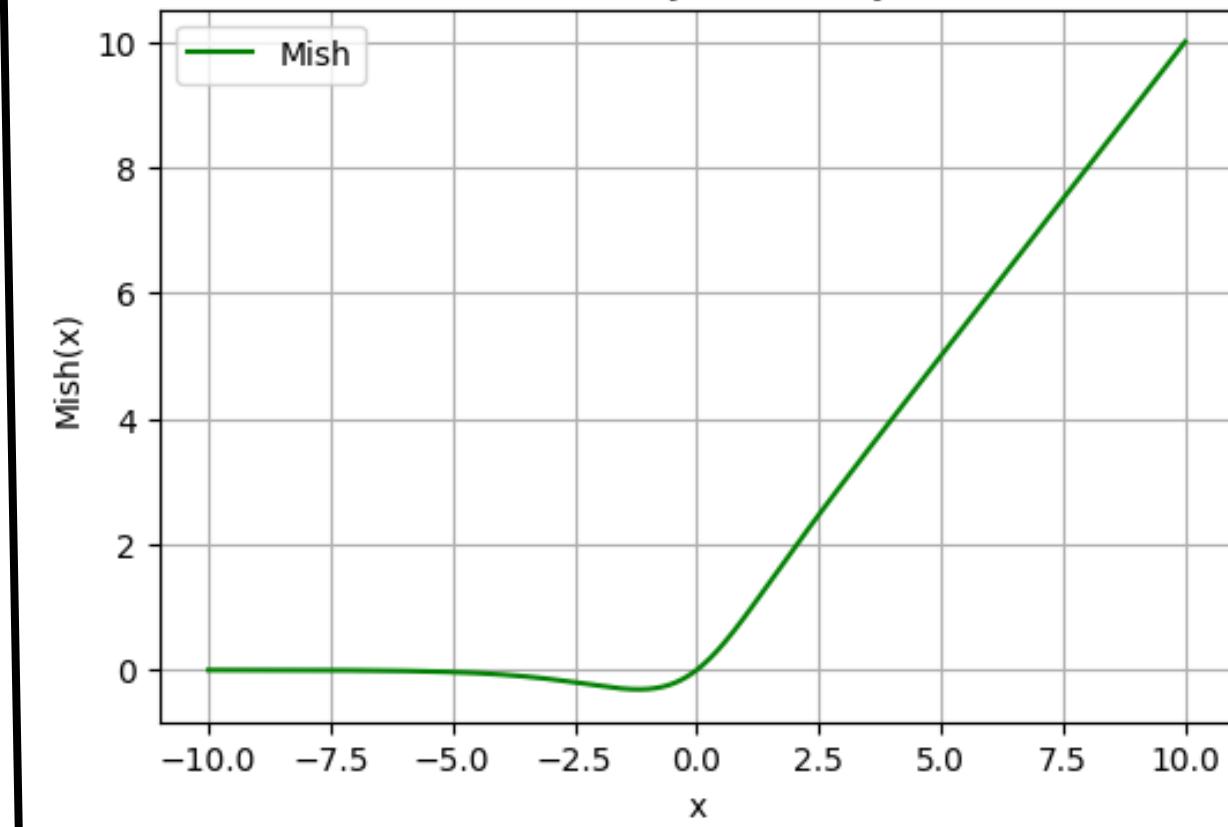


$$\text{HardSwish}(x) = x \cdot \frac{\min(\max(0, x + 3), 6)}{6}$$

Swish'in kolay hesaplanan hali

## Mish

Mish Aktivasyon Fonksiyonu



$$\text{Mish}(x) = x \cdot \tanh(\ln(1 + e^x))$$

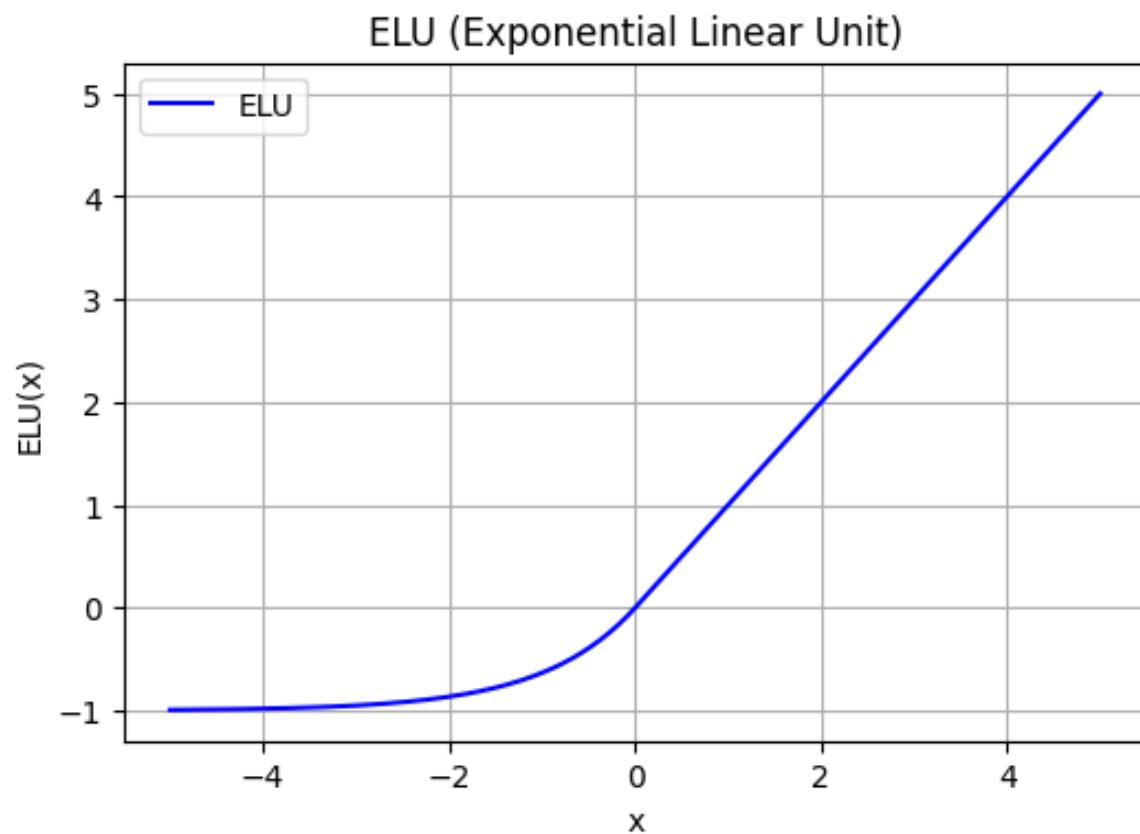
Swish benzeri özellikleri taşıyan başka bir aktivasyon fonksiyonu

\*SWISH: A SELF-GATED ACTIVATION FUNCTION

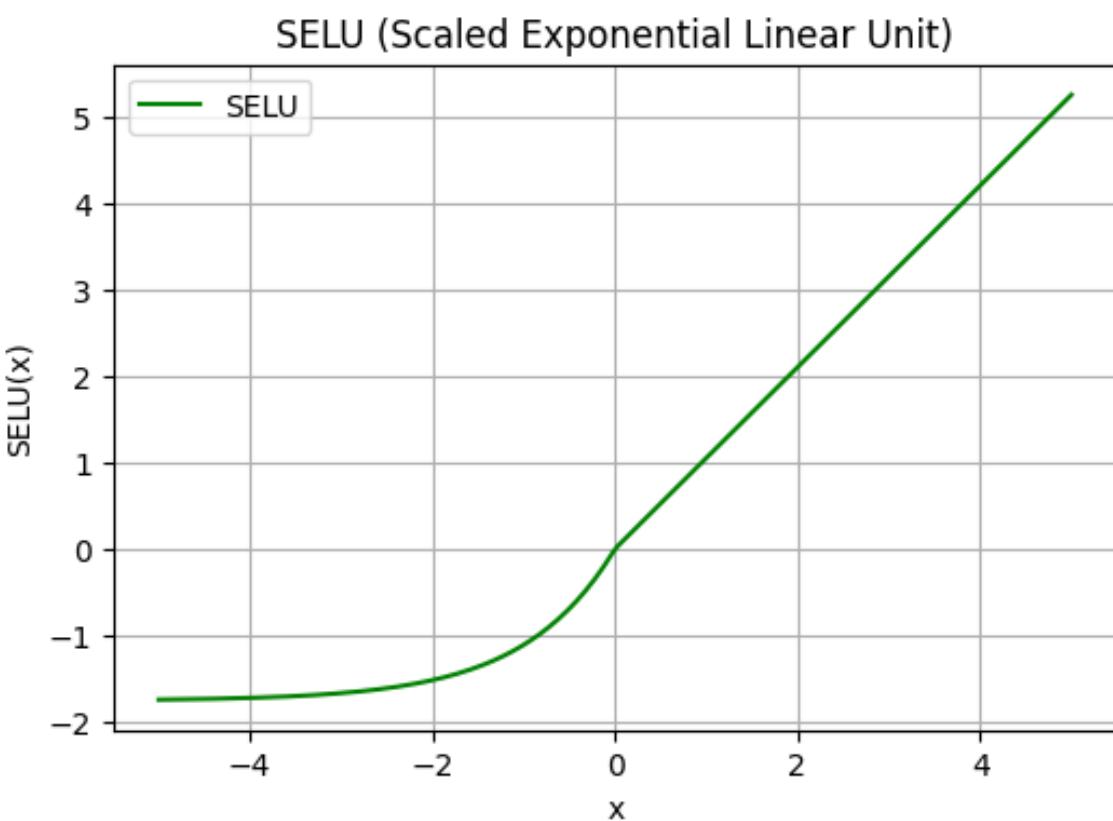
\*Mish: A Self Regularized Non-Monotonic Activation Function

# ELU - SELU- CELU

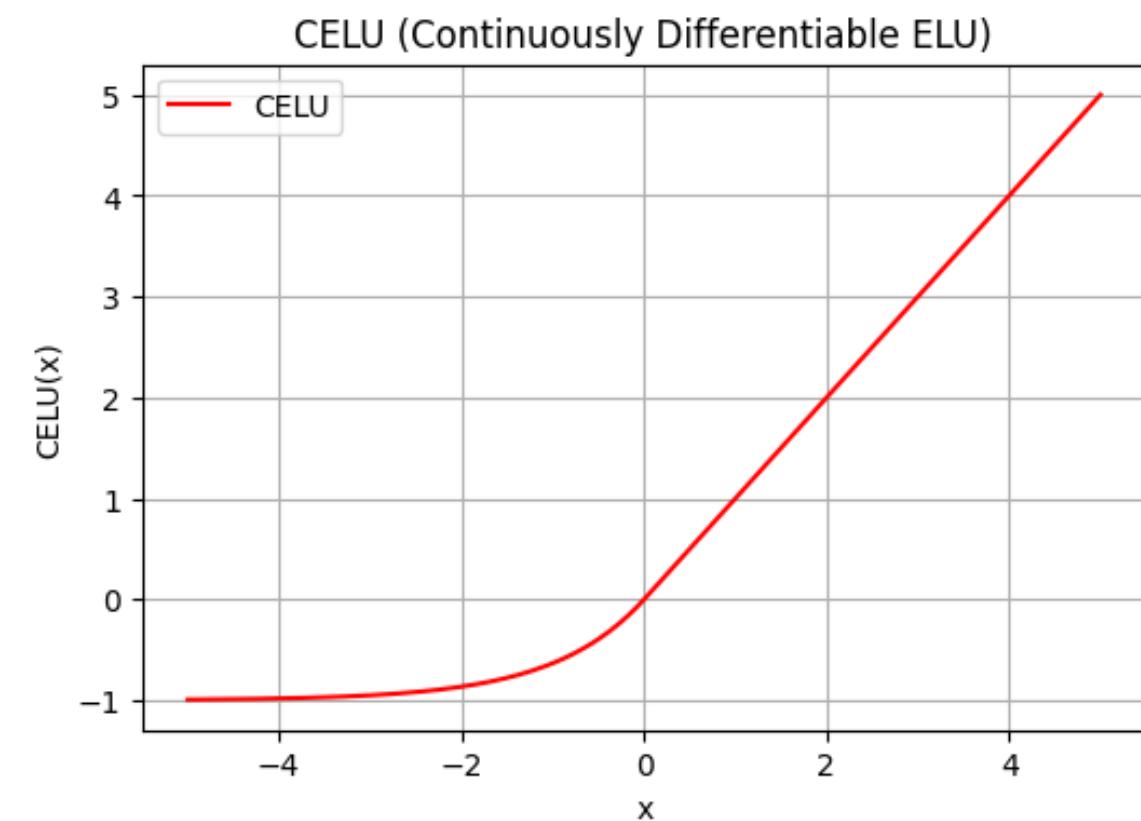
## ELU



## SELU



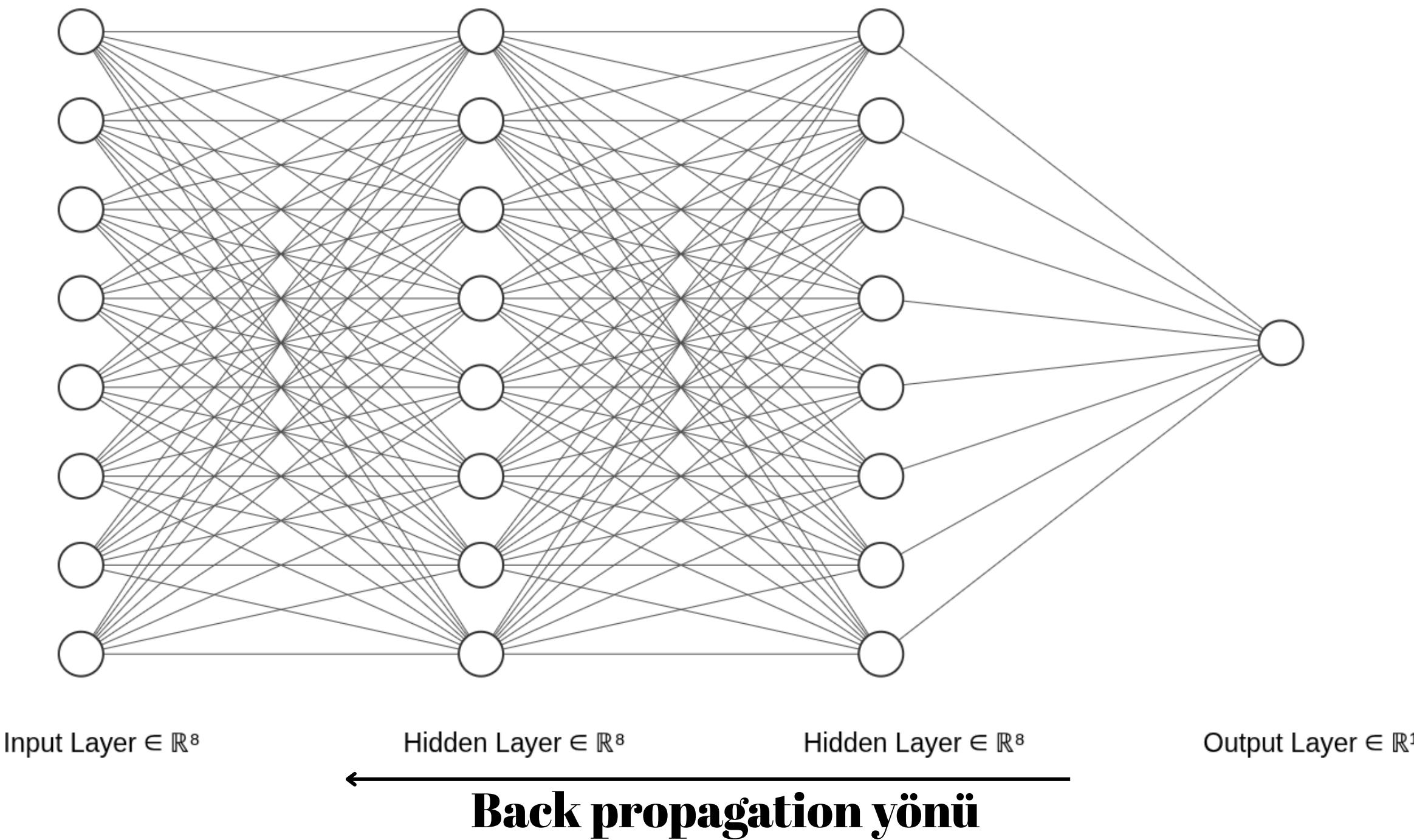
## CELU



\*Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)

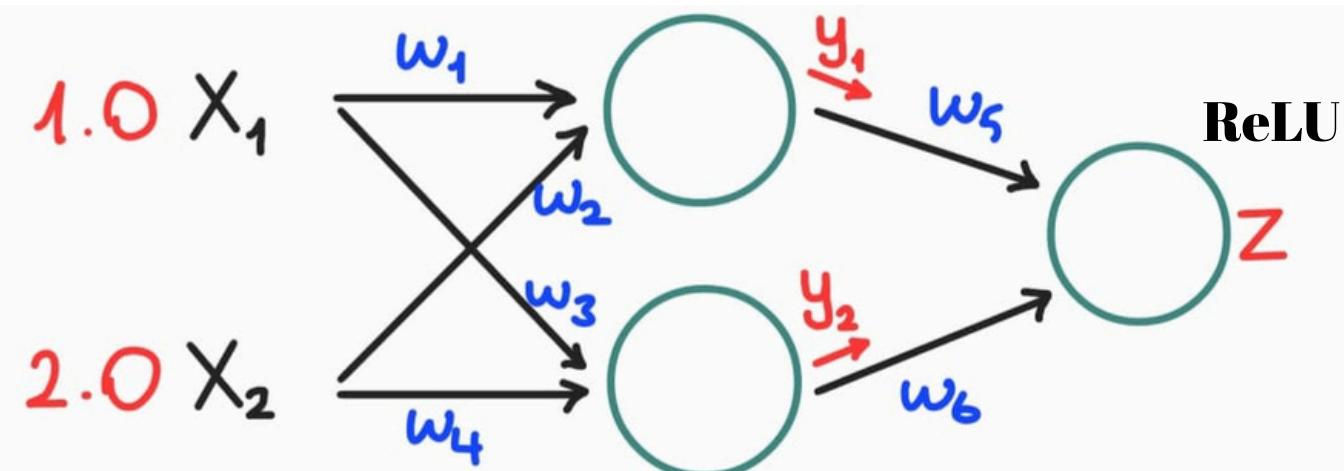
\*Self-Normalizing Neural Networks

# Backpropagation



- **Backpropagation** = Zincir kuralı (chain rule) kullanılarak son katmandan ilk katmana doğru parametrelerin güncellenmesi işlemidir.
- Buradaki temel mantık bir katmanın çıkışının bir sonraki katmanın girdisi olmasına dayanmaktadır.
- Bu yöntem ile kullanılan optimizer (Gradient Descent vb.) sadece son katmandaki ağırlıkları değil önceki katmanlardaki ağırlıkları da güncelleyebilir.

# Backpropagation



$$\begin{array}{ll} w_1 \rightarrow 0.1 & w_4 \rightarrow 0.4 \\ w_2 \rightarrow 0.2 & w_5 \rightarrow 0.5 \\ w_3 \rightarrow 0.3 & w_6 \rightarrow 0.6 \end{array}$$

$$y_{\text{true}} = 1.0$$

$$\text{Loss} = \text{MSE} = \frac{1}{2} \sum (y - \hat{y})^2$$

$$\begin{aligned} y_1 &= w_1 \cdot x_1 + w_2 \cdot x_2 \\ &= 0.1 \times 1.0 + 0.2 \times 2.0 = 0.5 \end{aligned}$$

$$\frac{\partial L}{\partial \hat{y}} = (\hat{y} - y) = -0.09$$

$$\begin{aligned} y_2 &= w_3 \cdot x_1 + w_4 \cdot x_2 \\ &= 0.3 \times 1.0 + 0.4 \times 2.0 = 1.1 \end{aligned}$$

$$\frac{\partial \hat{y}}{\partial z} = \frac{\partial \text{ReLU}}{\partial z} \quad \frac{\partial \text{ReLU}}{\partial z} = 1$$

$$\begin{aligned} Z &= w_5 \cdot y_1 + w_6 \cdot y_2 \\ &= 0.5 \times 0.5 + 1.1 \times 0.6 = 0.91 \end{aligned}$$

$$\frac{\partial L}{\partial w_5} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z} \cdot \frac{\partial z}{\partial w_5} \stackrel{y_1}{=} = -0.045$$

$$\hat{y} = \text{ReLU}(0.91) = 0.91$$

$$\frac{\partial L}{\partial w_6} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z} \cdot \frac{\partial z}{\partial w_6} \stackrel{y_2}{=} = -0.099$$

## **iteration - batch - epoch**

**batch:** Verinin bölündüğü her bir küçük gruba denir. **batch\_size** bu grupların içerisindeki veri sayısını belirtir.

**320 adet verimiz varsa ve batch\_size = 32 ise**

**$320/32 = 10$  adet batch'imiz olur**

**iteration:** model eğitim aşamasında modelin parametrelerinin güncellendiği her bir adımdır. Genellikle modelden 1 batch geçtikten sonra güncelleme yapılır ve 1 iterasyon sayılır.

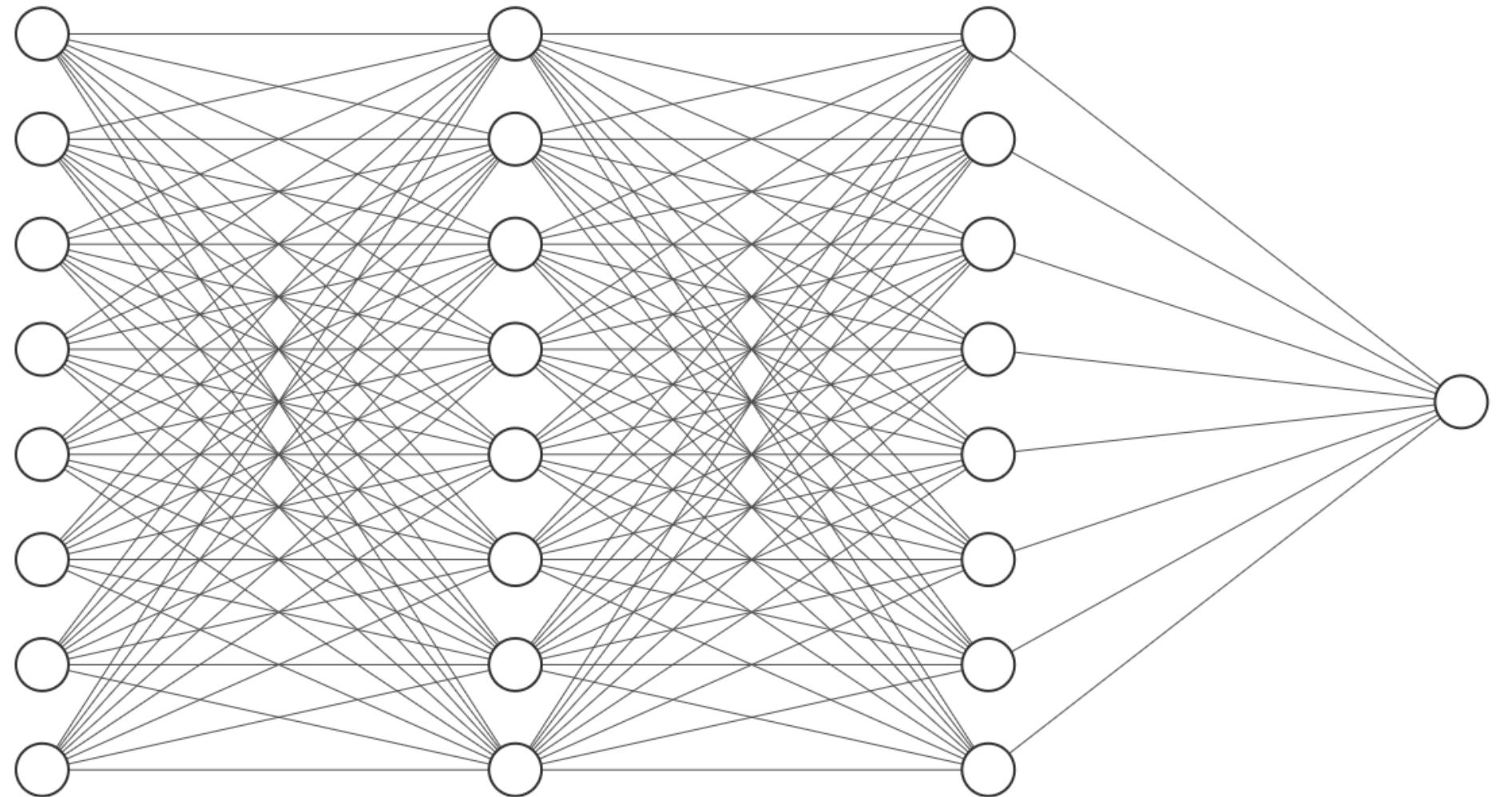
**epoch:** Bütün verinin modelden 1 tur geçmesini ifade eder.

**1 adet batch geçti → 1 iteration**

**10 adet batch geçti → 1 epoch**

**Toplam 20 epoch varsa → 200 adet iteration olur.**

# Vanishing-Exploding Gradient

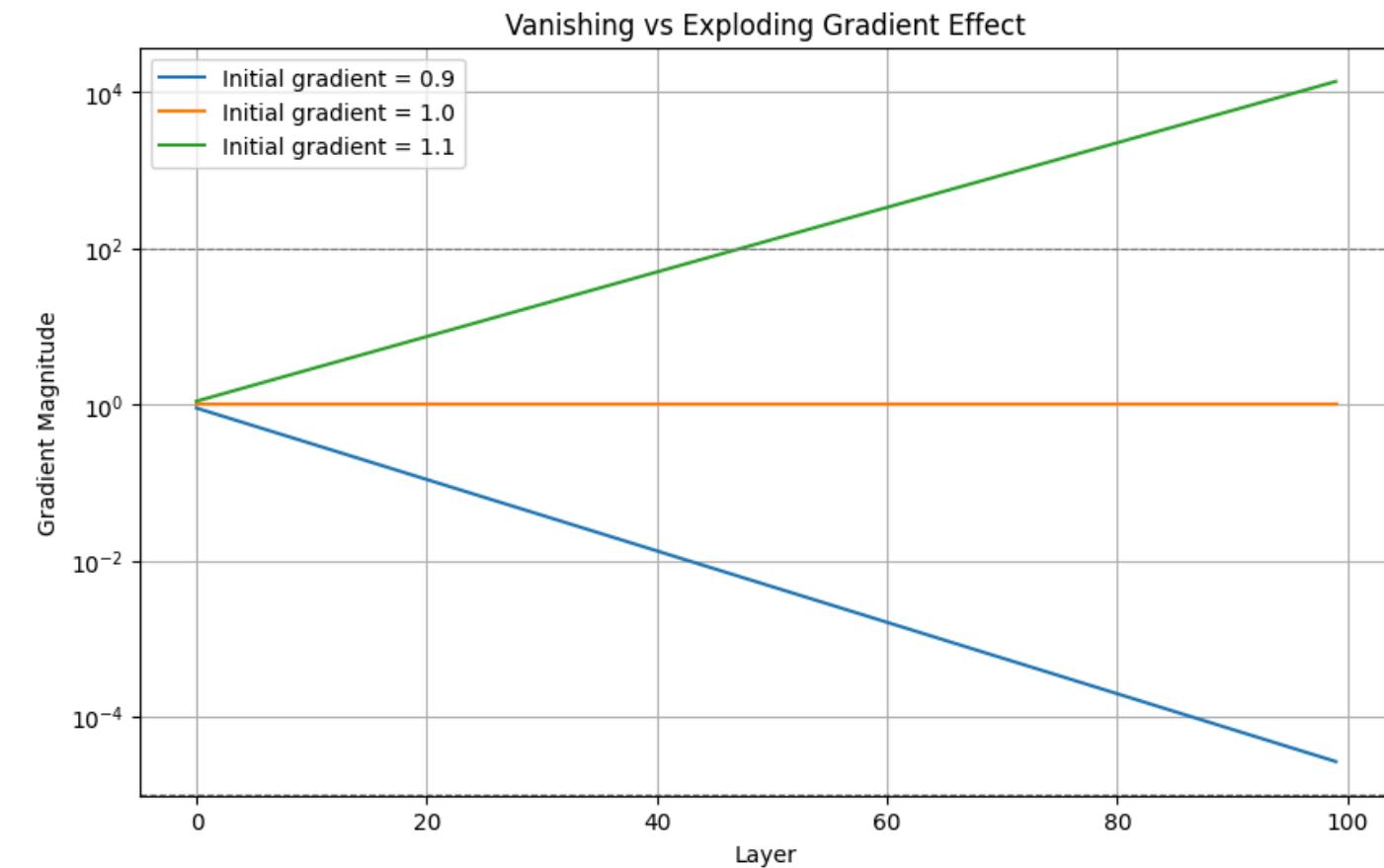


Input Layer  $\in \mathbb{R}^8$

Hidden Layer  $\in \mathbb{R}^8$

Hidden Layer  $\in \mathbb{R}^8$

Output Layer  $\in \mathbb{R}^1$



**Vanishing Gradient:** Gradyanların 0'a yakınsaması ile öğrenmenin duracak konuma gelmesi

**Exploding gradient:** Gradyanların çok büyük değerlere ulaşması ile modelin bozulması

Her layer içerisinde gradyanların 0.9 ile çarpıldığını düşünün. 150 layer sonra 0.000000137

Her layer içerisinde gradyanların 1.1 ile çarpıldığını düşünün. 150 layer sonra 1617717

# Loss Functions

MSE	<u>nn.MSELoss</u>
MAE	<u>nn.L1Loss</u>
Huber	<u>nn.HuberLoss</u>
SmoothL1	<u>nn.SmoothL1Loss</u>

Module-1'e dönmek  
isteyebilirsin



## Loss Functions

### Binary Cross Entropy

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

- $\hat{y}$  değeri sigmoid uygulanmış çıktıdır.
- Eğer kullanılan NN sigmoid uygulayarak çıktı veriyorsa sadece BCE kullanılır. eğer logit olarak çıktı veriyorsa BCEwithLogitsLoss kullanılır.
- BCEwithLogitsLoss numerical stability yüksektir. Overflow yaşama ihtimali daha düşüktür.

### Binary Cross Entropy with Logits Loss

$$l_n = -\omega_n [y_n \cdot \log_\sigma(x_n) + (1 - y_n) \cdot \log(1 - \sigma(x_n))]$$

- $x$  değeri sigmoid uygulanmadan önce elde edilen logit değerleridir.

## Loss Functions

BCE sadece ikili sınıflandırma problemlerinde kullanılabilir. NLL ve Cross Entropy çoklu sınıflandırma problemlerinde kullanılır.

### Negative Log Likelihood (NLL)

girdisi LogSoftmax uygulanmış değerler  
eğer gerçek class  $i$  ise

$$NLL = -p[i]$$

Örnek olarak gerçek class  $0$  ise

$$-p[0] = -(-0.41) = 0.41$$

**p**

### Cross Entropy

Girdileri Logit değerlerdir. CE hem logsoftmax hem de NLL uygulamaktır

$$\text{CrossEntropyLoss}(x,y) = \text{NLLoss}(\log(\text{softmax}(x)),y)$$

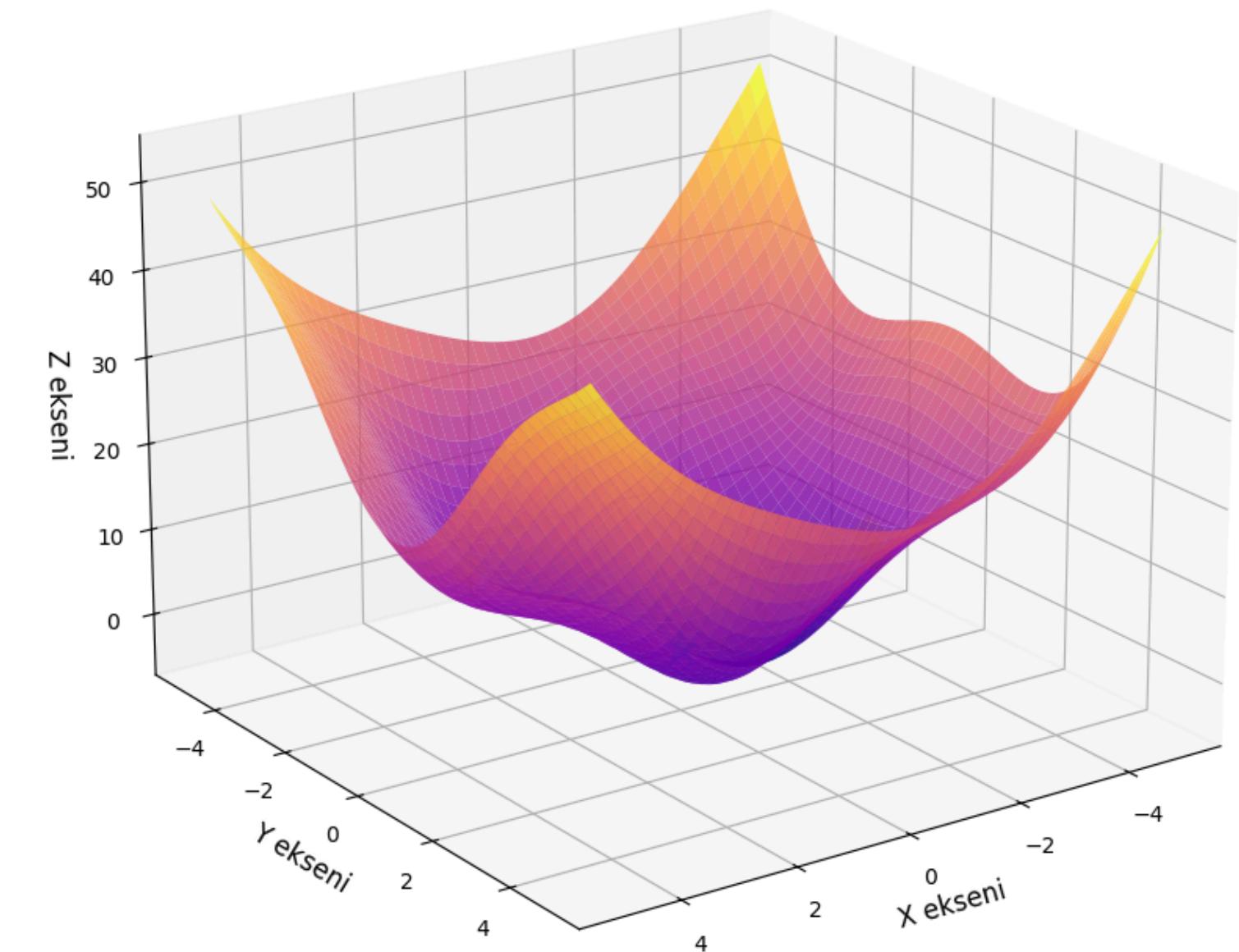
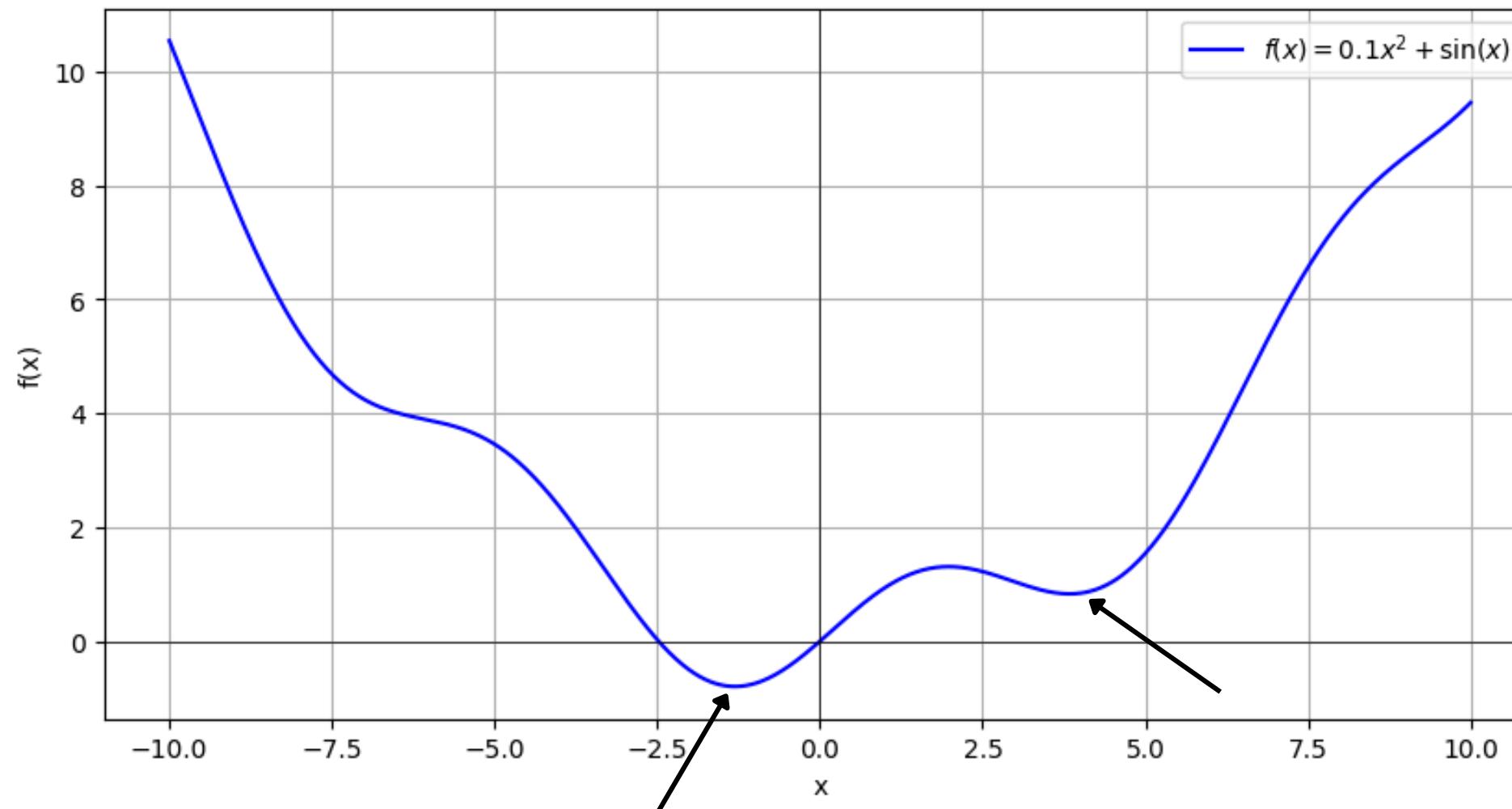
Cross Entropy kullanımı numerical olarak daha stabildir.

# Local-Global Minimum

Daha Pürüzsüz Bir Fonksiyonun Minimumları

Genel (Global) Minimum  
Yerel (Local) Minimumlar

Local ve Global Minimum Örnekleri



Model eğitiminde hedef loss function için global minimuma ulaşmaktadır.

Ancak local minimumlarda da türev 0 olduğu için eğitimin o noktada durma riski vardır.

## L2 Regularization

### Ridge

$$J(\theta) = \frac{1}{2m} \sum (y - X\theta)^2 + \frac{\lambda}{2m} \|\theta\|^2 \quad \text{lambd: regularization katsayısı}$$

- Overfitting'i önlemek için loss function'a ekstra terim ekler.
- Eklenen terim parametrelerin fazla büyümeyi engelleyerek aşırı öğrenmeyi engeller.  
(Çok büyük parametre büyük loss demek)
- Büyük parametre → büyük ceza
- Derin öğrenme modellerinde kullanılan weight decay terimi birçok durumda L2 regularization'ı ifade etmektedir

$$\theta := \theta - \alpha \left( \frac{1}{m} X^T (X\theta - y) + \frac{\lambda}{m} \theta \right)$$

Regularization terimi loss function'a eklenmeden doğrudan türevi alınarak parametre güncelleme adımına da konulabilir

## L1 Regularization

$$J(\theta) = \frac{1}{2m} \sum (y - X\theta)^2 + \frac{\lambda}{m} \|\theta\|_1 \quad \text{Lasso}$$

- L2 kadar sık kullanılmaz.
- Bazı özellikleri sıfıra çeker. Sadece önemli özelliklerin katsayısı 0 haricinde kalır.
- Boyutu büyük veri setlerinde çok faydalıdır

Lambda Optimizer içerisinde parametre olarak verilebilir

# Kümülatif Kare toplamı

$$G_t = \sum_{i=1}^t g_i^2$$

x	G <sub>t</sub>
5	25
6	<b>25+36 = 61</b>
7	<b>61 + 49 = 110</b>

- **Girdilerin karelerini toplar.**
- **Devamlı büyümeye eğilimindedir. Bu yüzden belirli bir iterasyon sonra G<sub>t</sub> çok büyük değerlere ulaşabilir.**

# Exponential Moving Average

$$EMA_t = \alpha \cdot x_t + (1 - \alpha) \cdot EMA_{t-1} \quad \alpha = \frac{2}{N+1} \quad N: \text{Window genişliği}$$

index	x	EMA
0	10	10
1	12	11
2	13	12
3	15	13.5
4	14	13.75

**n = 3 , alpha = 1/2**

$$EMA_0 = 10$$

$$EMA_1 = \frac{1}{2} * 12 + \frac{1}{2} * 10 = 11$$

$$EMA_2 = \frac{1}{2} * 13 + \frac{1}{2} * 11 = 12$$

.

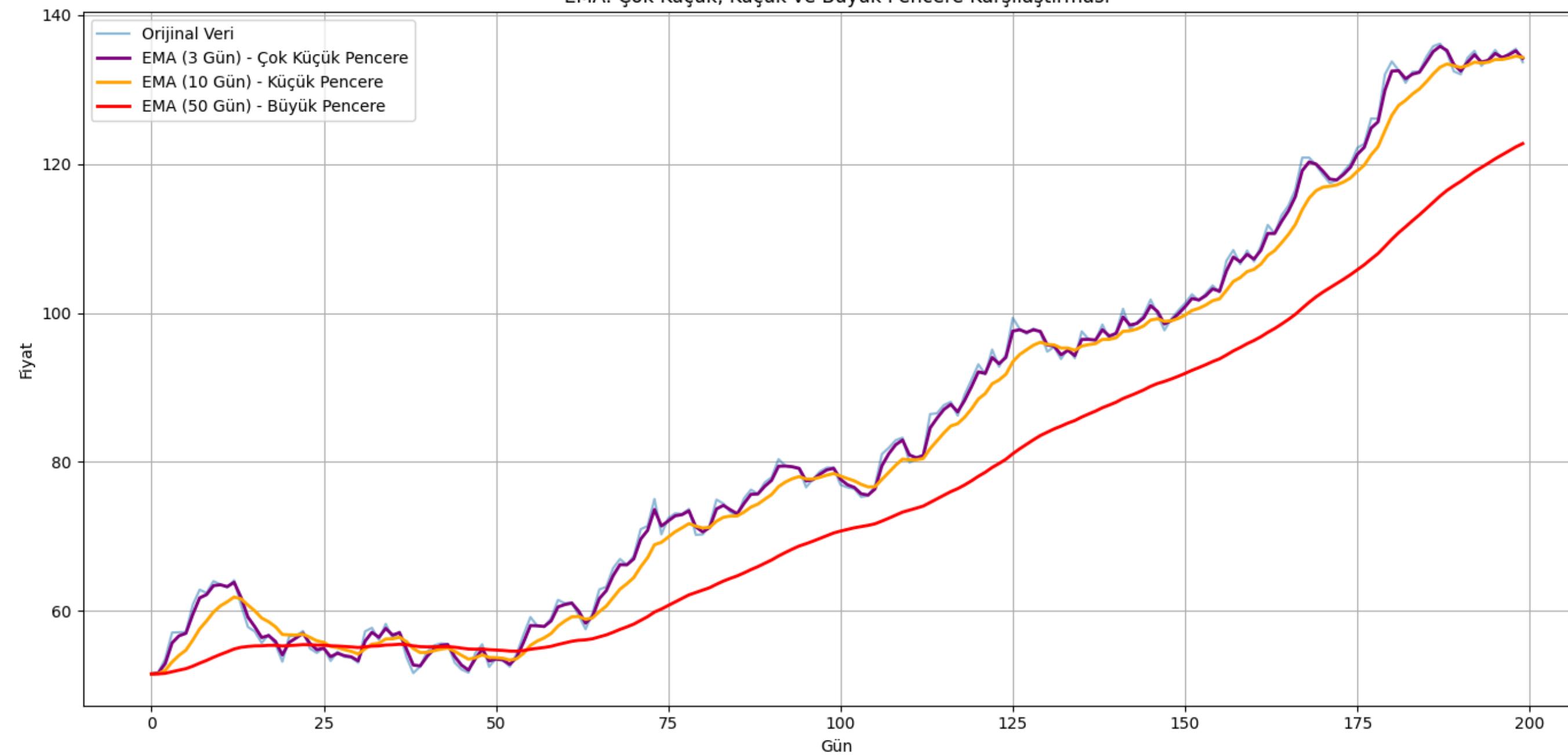
.

.

- EMA geçmiş ortalamaları tutarak dalgalanmaları yumusatır.
- Son değerlere daha fazla önem vererek (N) trendleri yakalamayı sağlar.
- Kareler toplamı gibi sürekli büyümeye eğilimi yoktur.
- alpha = 0.9 → Son 10 örnek
- alpha = 0.99 → son 100 örnek

# Exponenetial Moving Average

EMA: Çok Küçük, Küçük ve Büyük Pencere Karşılaştırması



**EMA trendleri yakalama konusunda iyidir.**

**Çok küçük N değeri dalgalanmaları yumuşatamaz.**

**Çok büyük N değeri geçmişte takılı kalır :(**

**optimal bir değer hem dalgalanmaları yumuşatır hem de trendleri yakalar.**

# Gradient

$$J(\theta) = \frac{1}{2}(\theta_1 x_1 + \theta_2 x_2 - y)^2$$

$$\nabla J = \left[ \frac{\partial J}{\partial \theta_1}, \frac{\partial J}{\partial \theta_2} \right] = (\hat{y} - y) \cdot [x_1, x_2]$$

## Optimizers

**Torch içerisinde öğrenmenin gerçekleşmesini sağlayan algoritmalarıdır.**

**loss.backward() → gradyanları hesaplar.**

**optimizer.step() → Hesaplanmış gradyanları kullanarak parametreleri günceller.**

**SGD - Momentum - ASGD**

**AdaGrad - RMSProp - Adadelta**

**Adam - AdamW - AdaFactor**

## Stochastic Gradient Descent

**Gradient Descent** → Tüm veri geçsin - güncelleme yap

**SGD** → Bir adet veri geçsin Güncelleme yap

**Mini-Batch GD** → bir Batch geçsin güncelleme yap

Tanımlamalara rağmen SGD terimi çoğu zaman mini-batch yerine kullanılır.

$$\theta := \theta - \eta \cdot \nabla_{\theta} J(\theta; x^{(i)}, y^{(i)})$$

↓                      ⌈ ⌋  
learning rate      gradyan

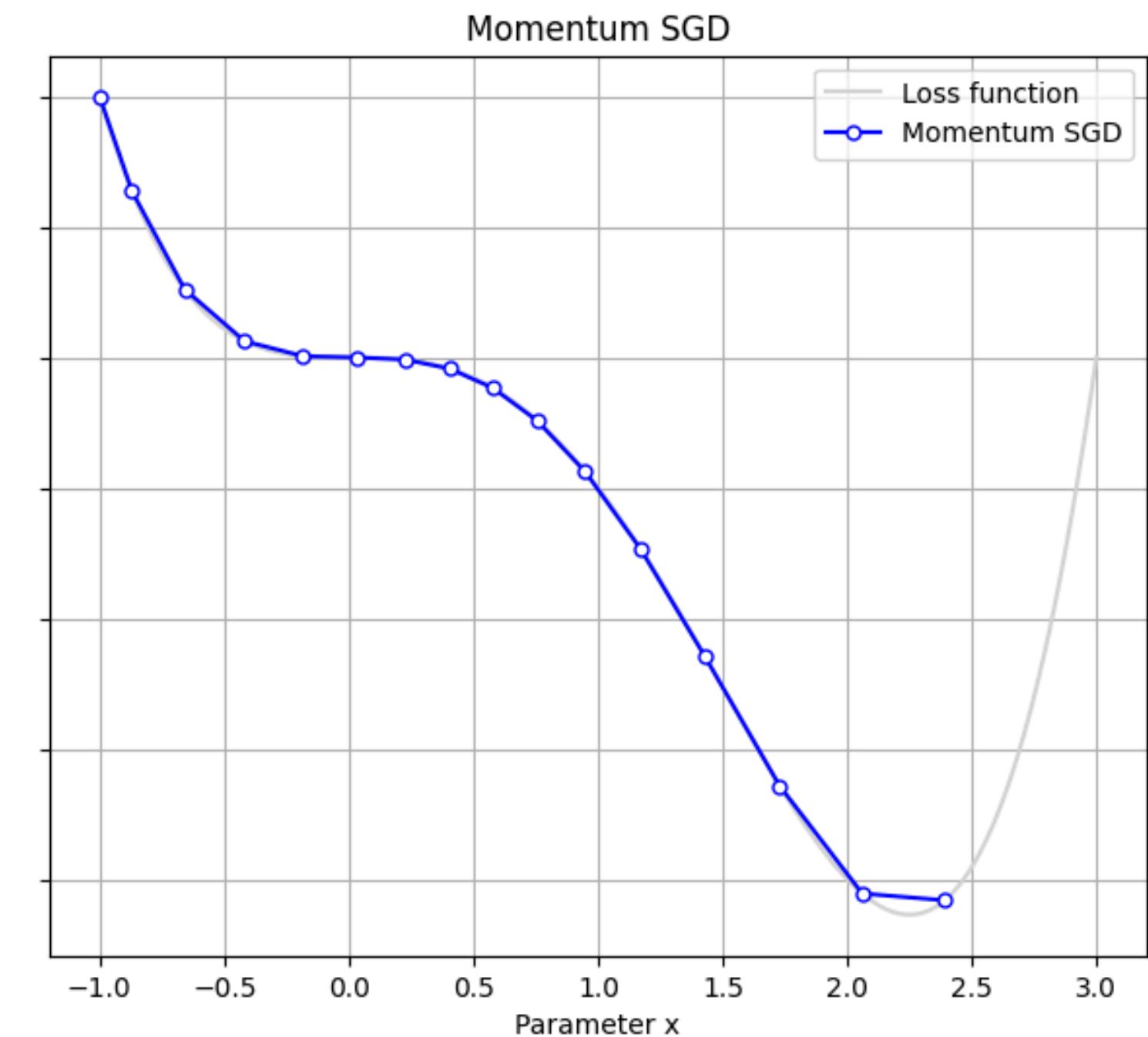
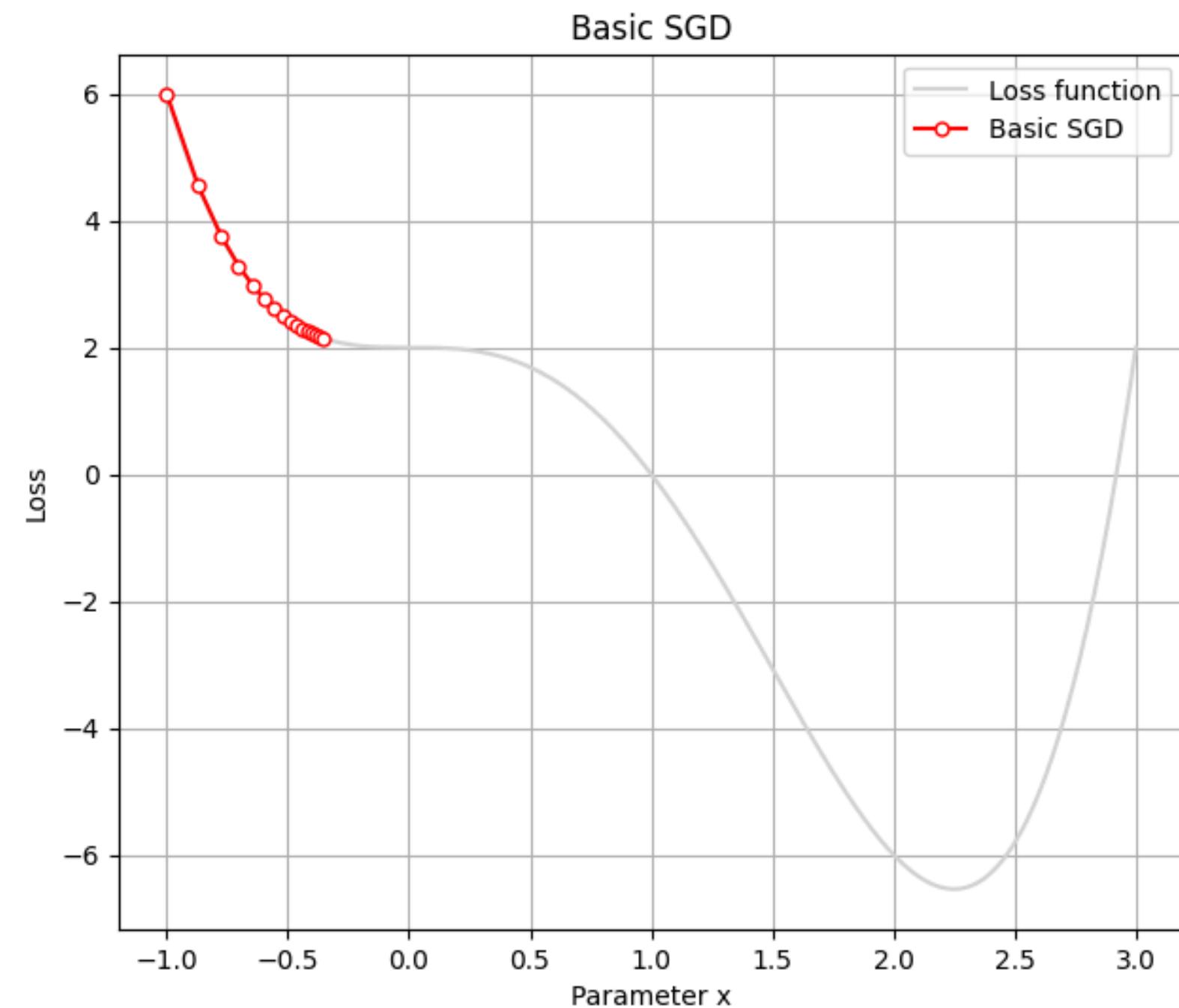
## **SGD-Momentum**

$$v_t = \beta v_{t-1} + (1 - \beta) \eta \nabla_{\theta} J(\theta)$$
$$\theta := \theta - v_t$$

- Gradyanların geçmiş değerlerini de EMA ile tutarak dalgalanmaları önler ve trendleri takip eder. Optimizasyon süreci hızlanır
- Birçok durumda normal SGD'ye göre daha iyi iş yapar.
- Beta ( $0 < \beta < 1$ ) geçmiş örneklerin ne kadar hatırlanacağını belirler. Genelde 0.9 ya da 0.99 olur.
- $V_t$  değeri her bir parametre için ayrı ayrı tutulur. Yani güncellenen her bir parametrenin geçmiş güncelleme değerleri de göz önünde bulundurulur.
- Her bir parametre için  $V_t$  değeri tutmak fazladan hafıza demektir. (10 milyar parametre varsa 10 milyar  $V_t$  değeri demek.)

# SGD-Momentum Grafiği

Basic SGD vs Momentum SGD on Loss with Local & Global Minima



# Averaged Stochastic Gradient Descent (ASGD)

$$\bar{\theta}_t = \frac{1}{t} \sum_{i=1}^t \theta_i$$

**Belirli bir  $t_0$  adımından sonra parametrelerin ortalamasını alır.**  
 $t_0 = 100$

t	theta (SGD)	theta (final)
101	2	2
102	1.5	1.75
103	1.2	1.57
104	1.3	1.5
105	1.1	1.42

- $t_0$  ‘dan bitişe kadar SGD normal işini yapar.  
**Average kısmı bir yandan ortalama değerleri hesaplar.**
- Eğitim bittiğinden sonra Average kısmının elde ettiği nihai theta (ağırlıklar-parametreler) kullanılır.
- Özellikle eğitimin sonuna doğru kullanılması modelin kararlı bir bitiş yapmasını sağlar

## AdaGrad

$$w_{t+1} = w_t - \frac{\eta}{\sqrt{G_t} + \varepsilon} \cdot \nabla f(w_t)$$

$$G_t = \sum_{i=1}^t (\nabla f(w_i))^2$$

**epsilon:** sıfıra bölmeyi engellemek için koyulan çok küçük değer

- Learning rate'i her parametre için ayrı ayrı ayarlar.
- Nadir durumların güncelleme oranı daha fazla olur ( $G_t$  değerleri küçük)
- Başlangıçta iyi çalışabilir, ancak bir süre sonra  $G_t$  değeri çok büyüdüğü için öğrenme durma noktasına gelir. Bu yüzden büyük veri setleri ve uzun eğitimler için uygun değildir.
- NLP ve Recommendation systems gibi sparse data olan yerlerde tercih edilebilir.

## RMSProp

$$w_{t+1} = w_t - \frac{\eta}{\sqrt{G_t} + \varepsilon} \cdot \nabla f(w_t)$$

$$G_t = \gamma G_{t-1} + (1 - \gamma)(\nabla f(w_t))^2$$

- **Root Mean Square Propagation (RMSProp) düzgün olmayan yüzeylerde öğrenme sağlayabilir.**
- **AdaGrad'ın problemi olan öğrenme durması problemini çözer. (EMA)**
- **Her parametre için özel bir öğrenme oranı ayarlanır.**
- **Sequential modellerde kullanılabilir.**

## AdaDelta

$$E[g^2]_t = \rho E[g^2]_{t-1} + (1 - \rho)(\nabla f(w_t))^2$$

$$E[\Delta w^2]_t = \rho E[\Delta w^2]_{t-1} + (1 - \rho)(\Delta w_t)^2$$

$$\Delta w_t = -\frac{\sqrt{E[\Delta w^2]_{t-1} + \varepsilon}}{\sqrt{E[g^2]_t + \varepsilon}} \cdot \nabla f(w_t)$$

$$w_{t+1} = w_t + \Delta w_t$$

- **Hem gradyanların hem de geçmişteki güncelleme oranlarının EMA'sını tutar.**
- **Dinamik olarak her parametre için öğrenme oranını kendisi ayarlar.**
  - **Torch içerisindeki lr parametresi sadece ilk başta kullanmak için**
- **Derin sinir ağlarında ve büyük veri setlerinde başarılıdır.**

# Adam

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) \nabla f(w_t)$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) (\nabla f(w_t))^2$$

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

$$w_{t+1} = w_t - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

**m<sub>t</sub>** → Birinci moment

**v<sub>t</sub>** → ikinci moment

- Momentum ve RMSProp'un avantajlarını birleştirir.
- **m<sub>t</sub>** → yön bilgisi
- **v<sub>t</sub>** → büyüklük bilgisi
- Öğrenme oranını her parametre için ayrı ayrı ve adaptif olarak ayarlar. (Learning Rate var!!)
- Özellikle ilk güncellemler için bias düzeltmesi vardır.
- Her türlü yerde kullanılır (alayına gider)
- Hyperparametre ayarları çok hassas değildir

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) \nabla f(w_t)$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) (\nabla f(w_t))^2$$

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

$$w_{t+1} = w_t - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t + \epsilon}} - \eta \lambda w_t$$

### Hikaye vakti

**Adam paper’ı ilk defa çıktığında weight decay olayı loss function kısmına implemente edildi ve bu durum regularization’ın tam olarak sağlanamamasına sebep oldu. Bu problemi fark eden arkadaşlar weight decay teriminin gradyan hesaplama kısmına değil de doğrudan parametre güncelleme kısmına eklediler böylece regularization daha etkili oldu.**

- **Büyük oranda Adam'a benzer.**
- **Büyük modellerde  $v_t$  hesaplanırken matrix'i tek bir satır ve tek bir sütun olarak yakınsar ve güncelleme için burda elde edilen 2 vektörü kullanır.**
- **Çok büyük modellerin eğitimi için hafıza ve hız avantajı sağlar.**

## linear - lazy linear - identity layers

### linear

$$\mathbf{y} = \mathbf{x}\mathbf{A}^T + \mathbf{b}$$

- **$\mathbf{w}\mathbf{x} + \mathbf{b}$  işlemini uygulayan en temel katmandır.**
- **Torch içerisinde matrix işlemlerinin gösterimi farklı olabilir. Bu framework'ün parametreleri nasıl depolandığı ile alakalıdır.**

**LazyLinear:** Linear katmanının `in_features` parametresi olmadan tanımlanabilen halidir. İlk `forward propagation` sırasında bu katman otomatik olarak `in_features` algılar ve normal Linear layer halini alır.

### identity

$$f(\mathbf{x}) = \mathbf{x}$$

- **Verilen girdinin ayını çıkarır.**
- **Algoritmadan ziyade kodlama kolaylığı sağlar.**
- **Örnek olarak kod içerisinde 100 tane normalization layer silmek yerine normalization layer tanımını identity yaparsak aynı sonucu elde ederiz.**

# Batch Normalization

Batch Normalization Formülü

1. Normalize:  $\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$

2. Scale & Shift:  $y_i = \gamma \hat{x}_i + \beta$

Notlar:

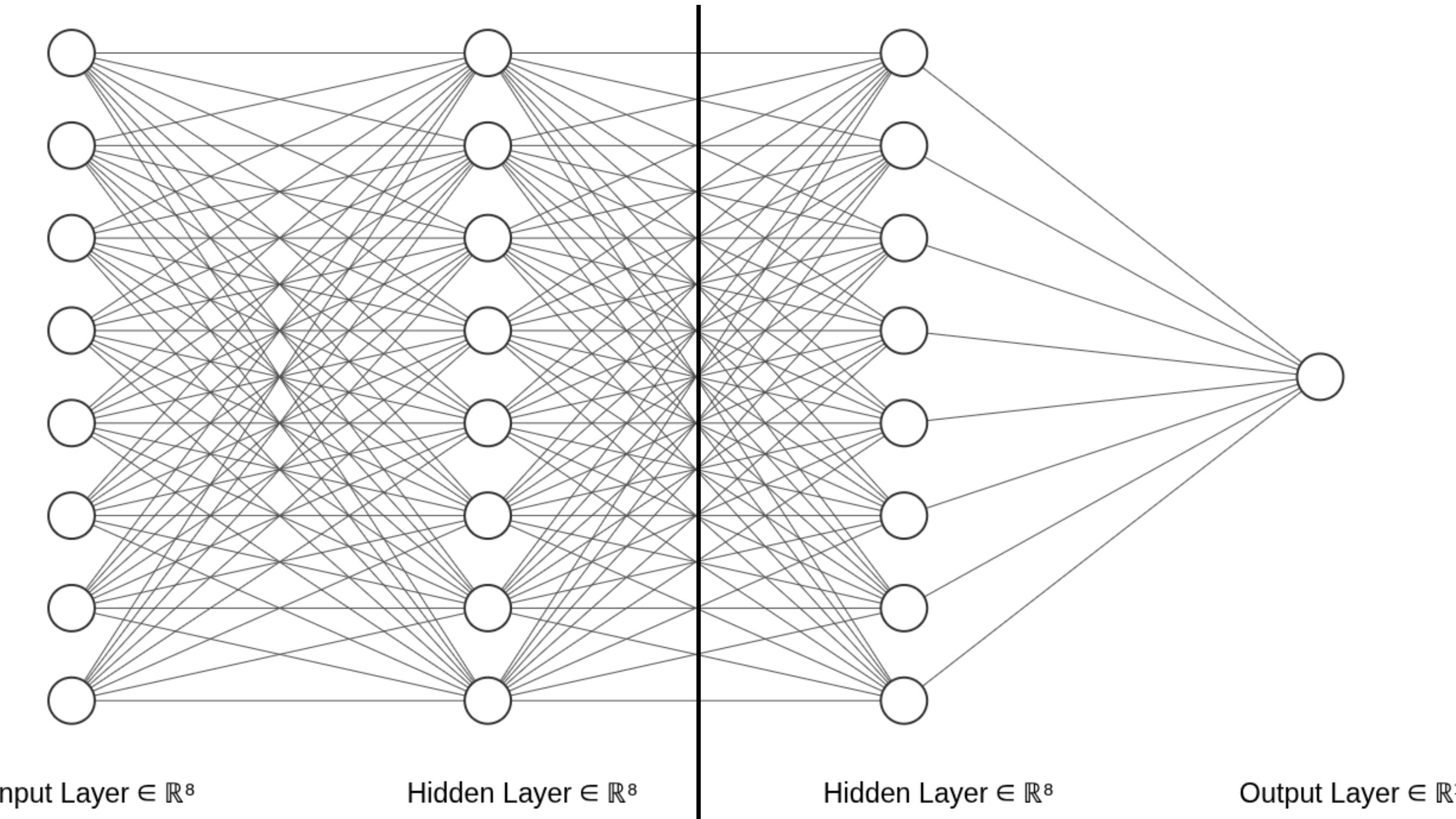
- $\mu_B$ : mini-batch ortalaması
- $\sigma_B^2$ : mini-batch varyansı
- $\epsilon$ : küçük sabit
- $\gamma, \beta$ : öğrenilen parametreler

- Internal covariate shift sorununu azaltır.
- NN içerisinde gizli katmanlardan önce kullanılabilir.
- Uyguladığı işlemleri her bir batch üzerinde uygular.  
Başka bir batch geldiğinde hesaplamalar tekrar yapılır.
- Önce normalize eder sonra “scale-shift” uygular.
- Modelin daha hızlı öğrenmesini sağlar
- Regularization etkisi gösterebilir.
- Test sırasında mean ve variance hesaplamak mümkün olmayabilir (1 örnek vb.). Bu durumda eğitim aşamasındaki avg\_mean ve avg\_variance kullanılır.

$x \rightarrow wx + b \rightarrow \text{batch normalization} \rightarrow \text{activation function}$  (Daha yaygın)

$x \rightarrow wx + b \rightarrow \text{activation function} \rightarrow \text{batch normalization}$

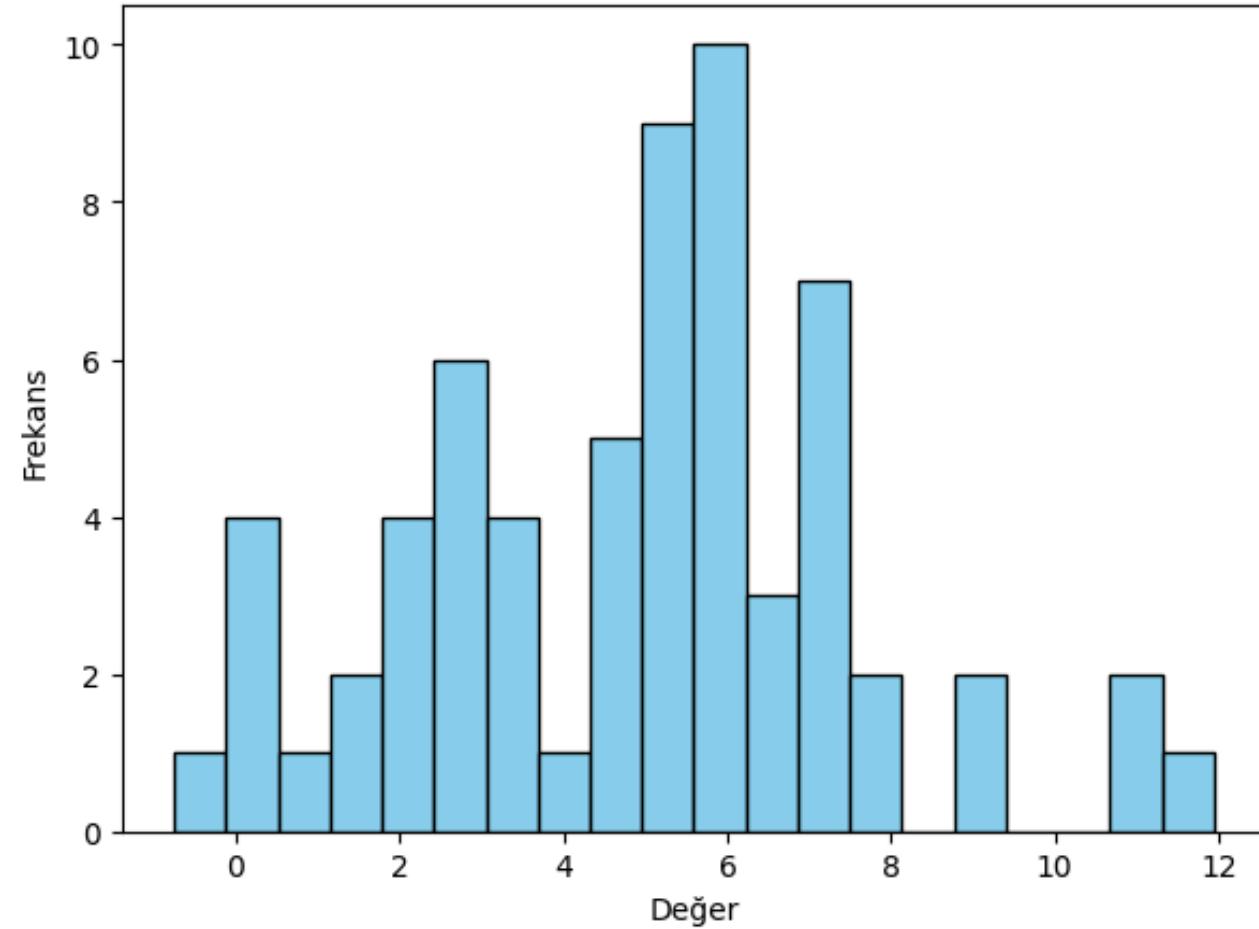
# Batch Normalization



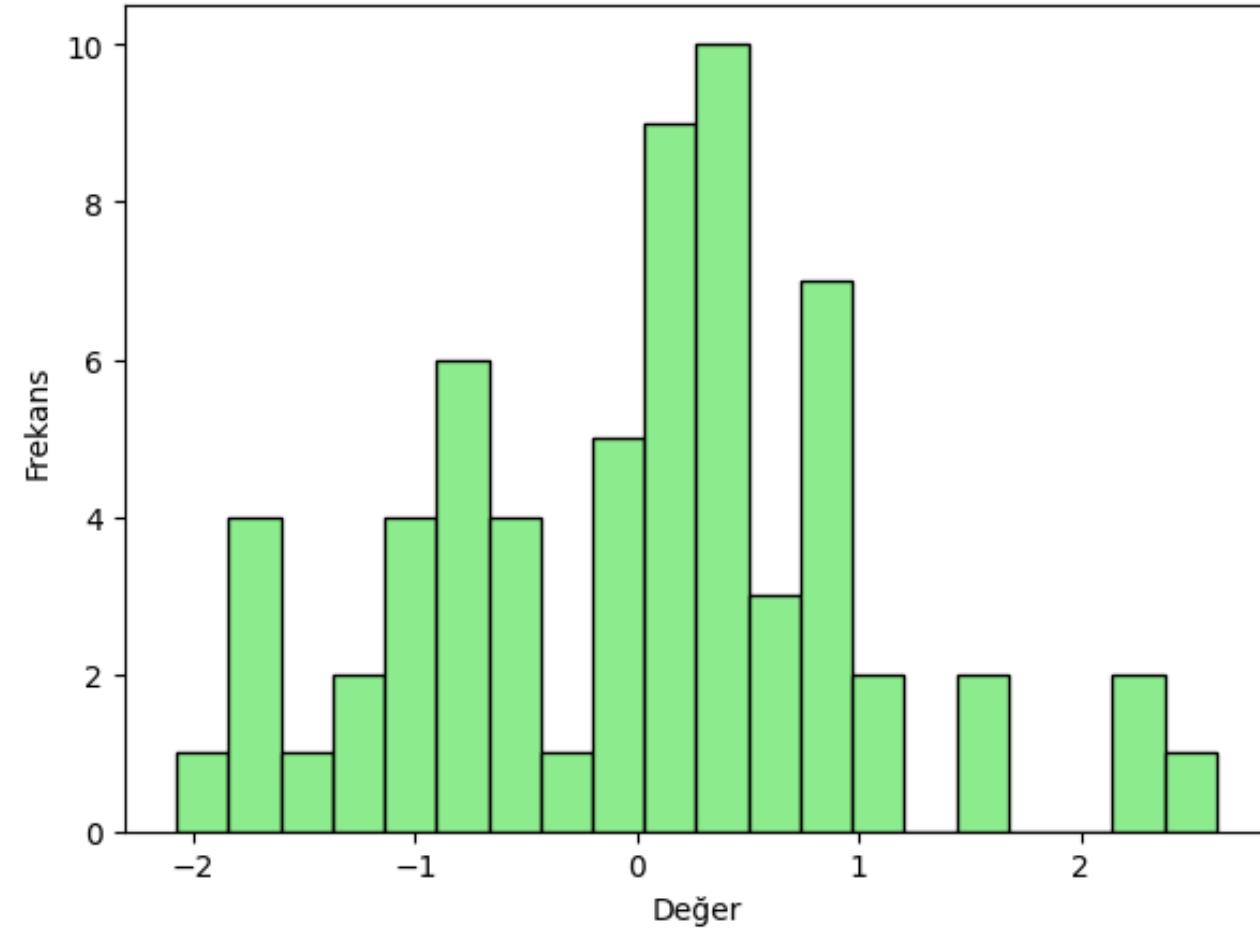
**Batch Normalization**

# Batch Normalization

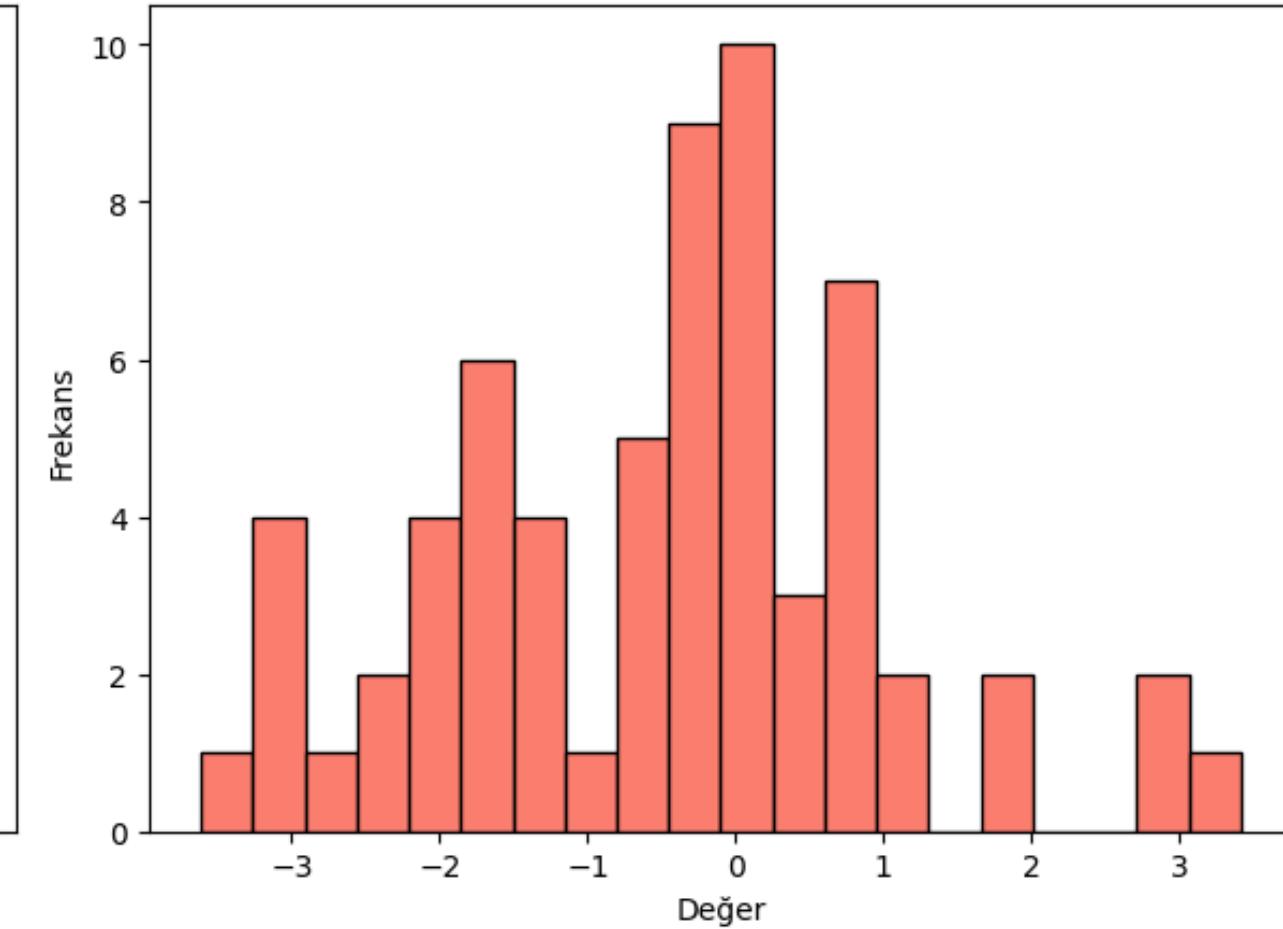
1. Orijinal Feature[0]  
Mean  $\approx 4.87$ , Std  $\approx 2.71$



2. Normalize Edilmiş Feature[0]  
Mean  $\approx 0.00$ , Std  $\approx 1.00$



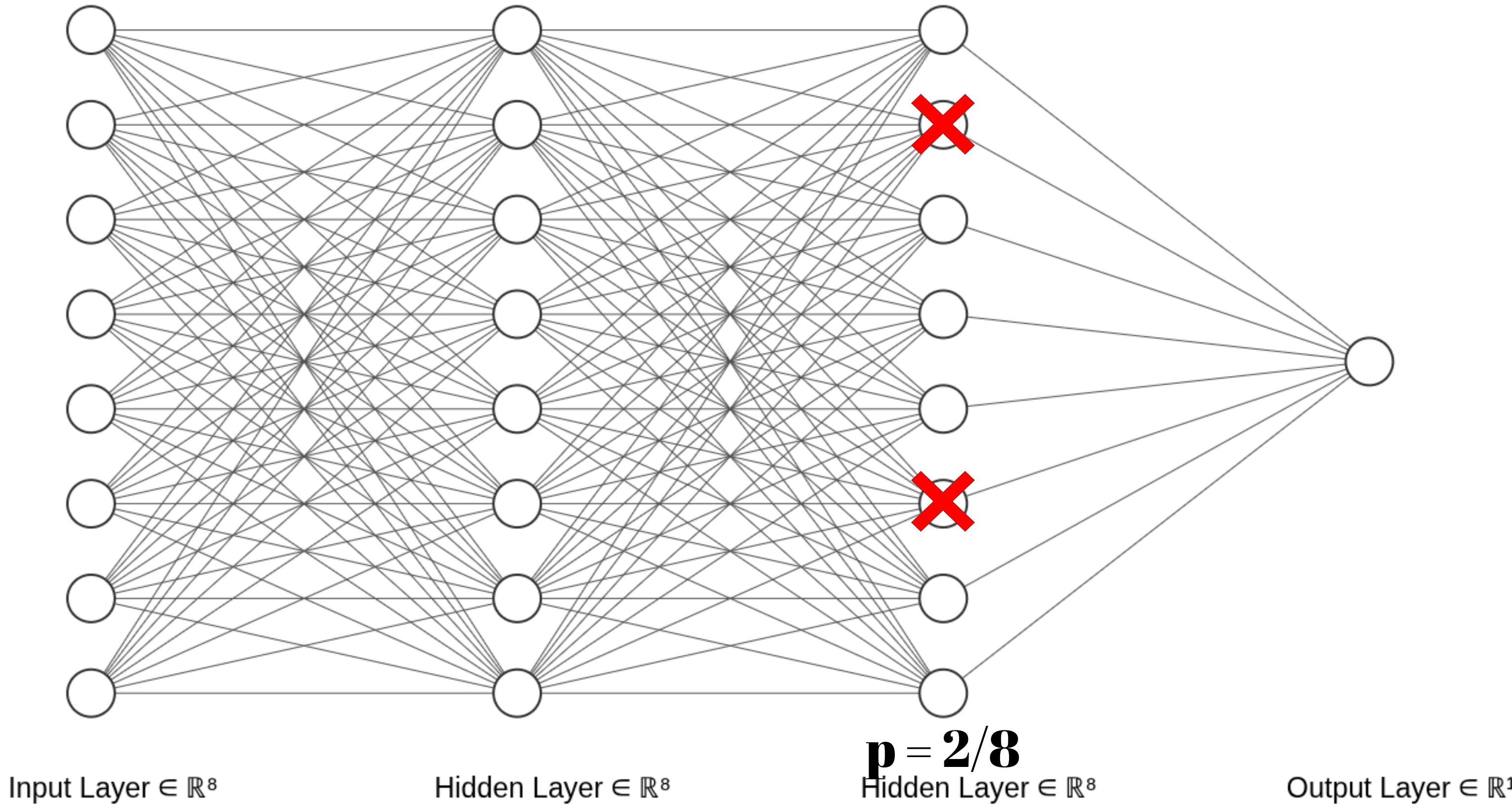
3. Scale & Shift Uygulanan Feature[0]  
Mean  $\approx -0.50$ , Std  $\approx 1.50$



**Lazy Batch Norm:** torch.nn içerisindeki batch norm'un weight initialization yapılmamış halidir. İlk forward pass sırasında weight initialize edilir.

BatchNorm2D ve BatchNorm3D farklı veri tipleri (image-video) üzerinde kullanılabilen normalizasyon katmanlarıdır. Yeri geldiğinde bahsedilecek (ins →

# Dropout



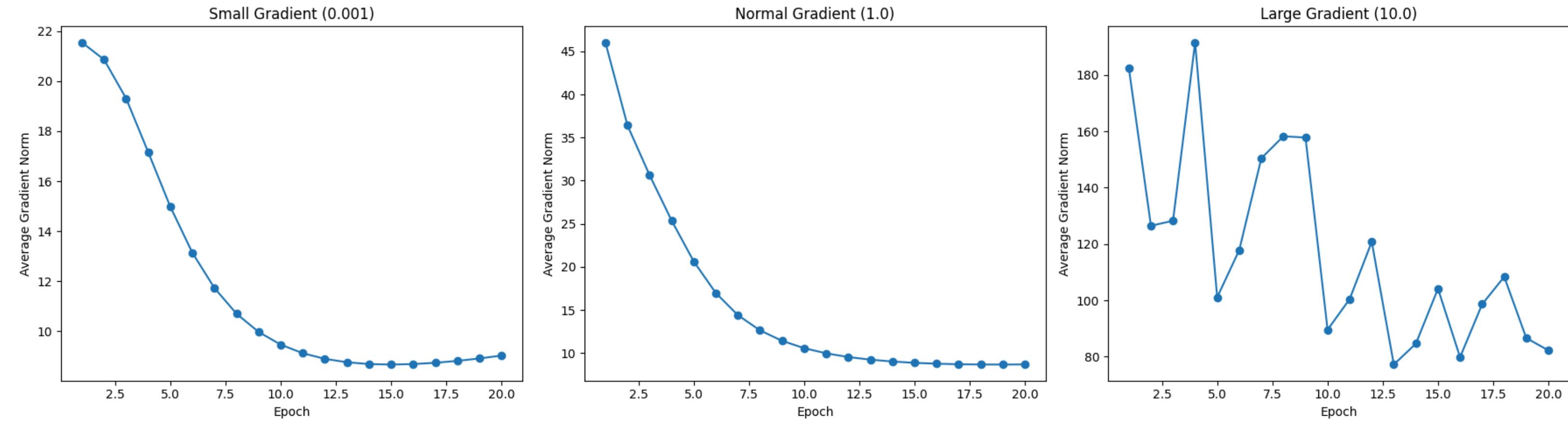
**Random olarak  $p$  olasılığında katmandaki nöronları etkisizleştir.**

**Nöronların her biri Bernoulli dağılımına göre bağımsız seçilir.**

## **Dropout**

- Etkisizleştirilen nöron  $f(x) = 0$  olur. Böylece o nöron üzerinden öğrenme sağlanamaz.
- Overfitting'i önleme konusunda başarılıdır. Model tek bir nöron üzerine kurulu olmaz. Tek bir nöronun sonucu belirlemeye çok etkili olduğunu varsayılm (katsayısı fazla). O nöron dropout ile sıfırlanırsa artık o nörondan öğrenme sağlanamaz. Bu da modeli diğer özellikleri de öğrenmeye iter
- Tahmin alma sırasında dropout deaktif edilmelidir. Çünkü bütün nöronların tahmine katkı sağlaması gereklidir. (`model.eval()` yapmak torch içinde dropout katmanının deaktif edilmesini sağlar.)

# Weight Initialization



**Parametreleri random bir şekilde başlatmak vanishing gradient ve exploding gradient'a sebep olabilir.**

**Random initialization:** Machine Learning ilk zamanlarında kullanılan genelde [-0.01,0.01] arasında eşitleme

**Zero initialization:** Tüm weightleri 0'a eşitleme. Yanlış bir yöntem Farklı özellikler öğrenilemez.

# LeCun Initialization

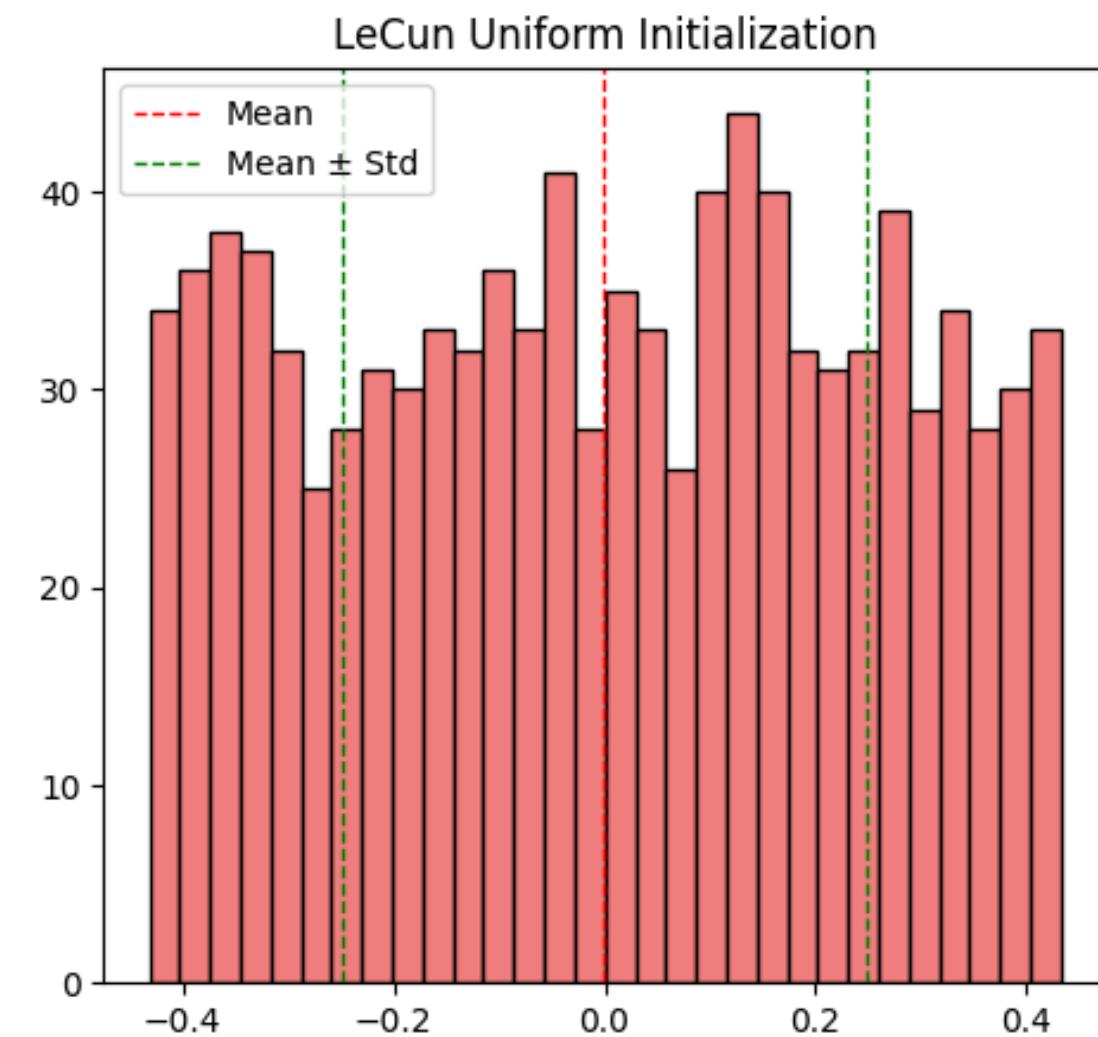
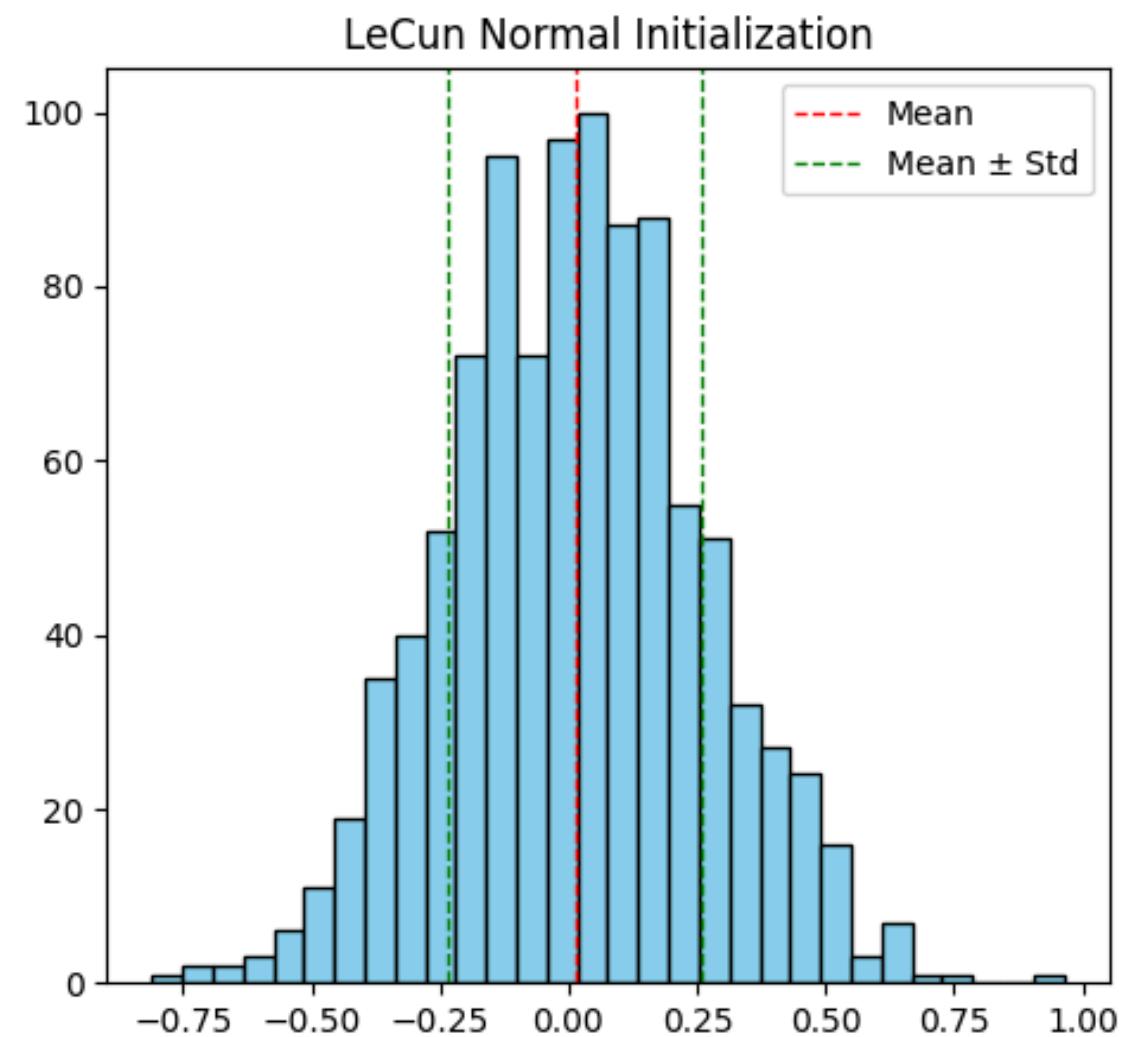
Variance (Normal):  $\text{Var}(w) = \frac{1}{n_{in}}$

Normal Distribution:  $w \sim \mathcal{N}(0, \frac{1}{n_{in}})$

Uniform Distribution:  $w \sim U[-\sqrt{3/n_{in}}, \sqrt{3/n_{in}}]$

**n<sub>in</sub>** = Bir önceki katmandaki nöron sayısı

**n<sub>out</sub>** = ilgili katmandaki nöron sayısı



**Vanishing ve exploding gradient'a çözüm olması için önerildi.**

**Katmana giren varyans ne ise çıkan da o olmalıdır düşüncesi**

**Sigmoid, tanh ve SELU ile iyi çalışır.**

**Normal dağılım:** Gauss eğrisi benzeri bir dağılım sağlar. Üç değerlerin seçilme ihtimali daha azdır. Birçok durumda default olarak kullanılır. Derin modellerde stabilité sağlar.

**Uniform Dağılım:** Bir aralık içerisinde eşit dağılım sağlar. Üç değer seçilme ihtimali de eşittir. Aralığın dışına çıkmama garantisini olduğu için bazı özel durumlarda tercih edilebilir.

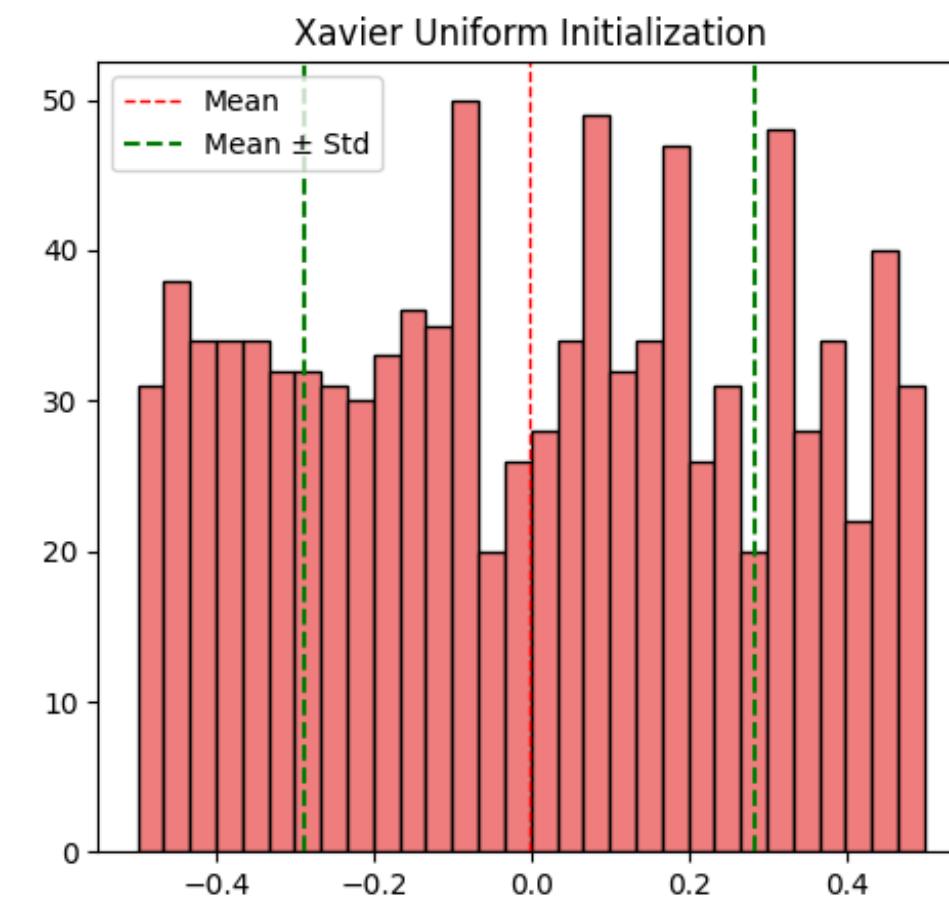
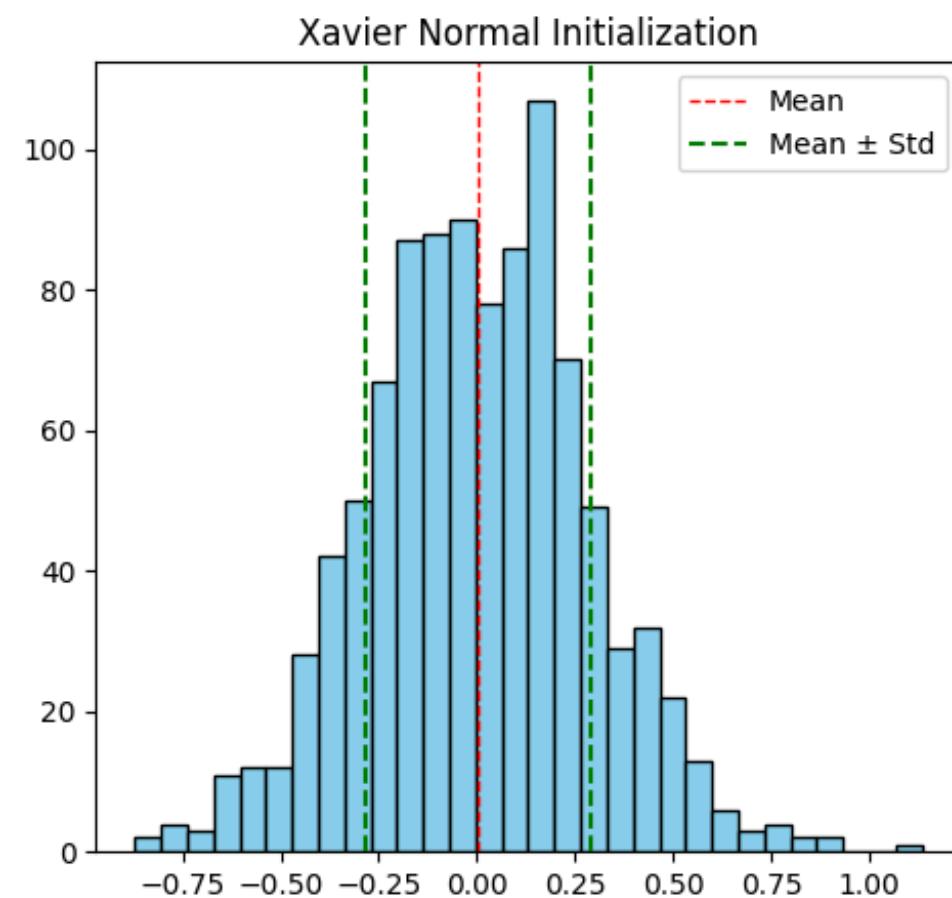
# Xavier Initialization

Xavier (Glorot) Initialization

$$\text{Variance (Normal): } \text{Var}(w) = \frac{2}{n_{in} + n_{out}}$$

$$\text{Normal Distribution: } w \sim \mathcal{N}\left(0, \frac{2}{n_{in} + n_{out}}\right)$$

$$\text{Uniform Distribution: } w \sim U\left[-\sqrt{\frac{6}{(n_{in} + n_{out})}}, \sqrt{\frac{6}{(n_{in} + n_{out})}}\right]$$



Hem giriş hem de çıkış varyansını düzenlemeyi baz alır.

**LeCun ve Xavier ReLU aktivasyon fonksiyonları için uygun değildir çünkü ReLU yaklaşık olarak çıktılarının yarısını 0 olarak vereceği için ortalama varyans yarıya düşmüştür.**

# He Initialization

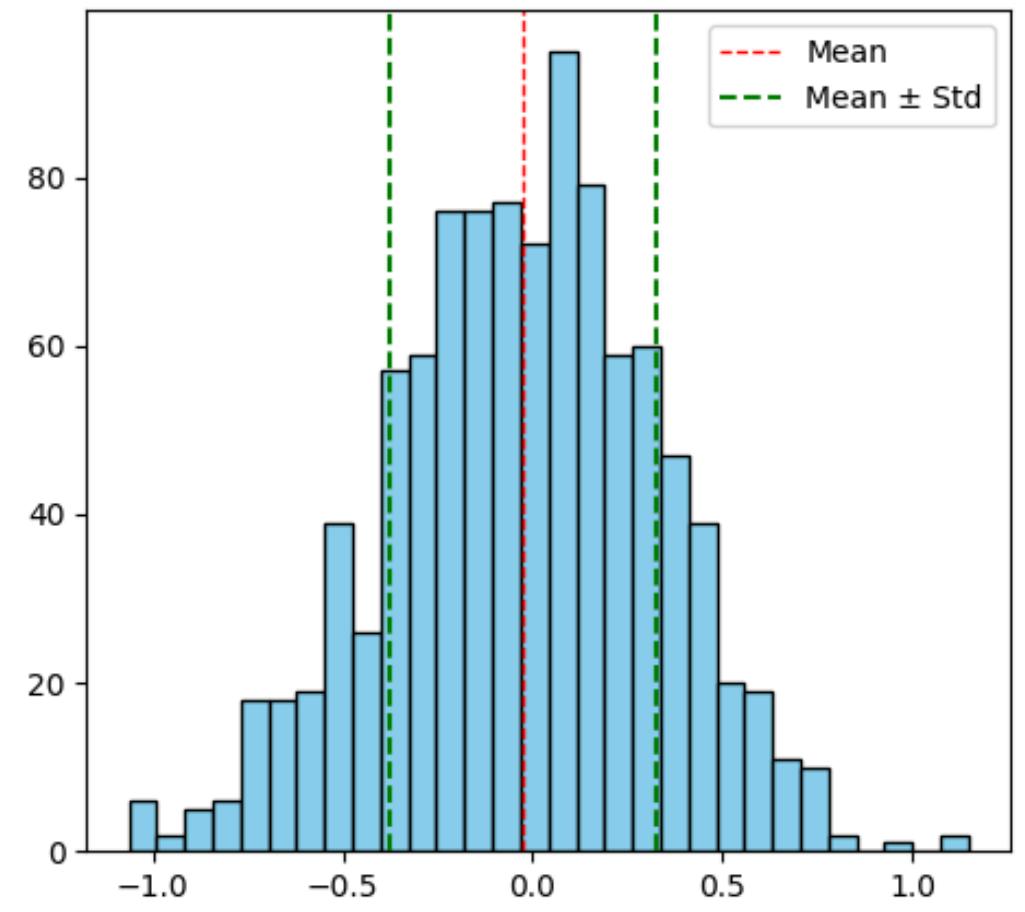
He Initialization

Variance (Normal):  $\text{Var}(w) = \frac{2}{n_{in}}$

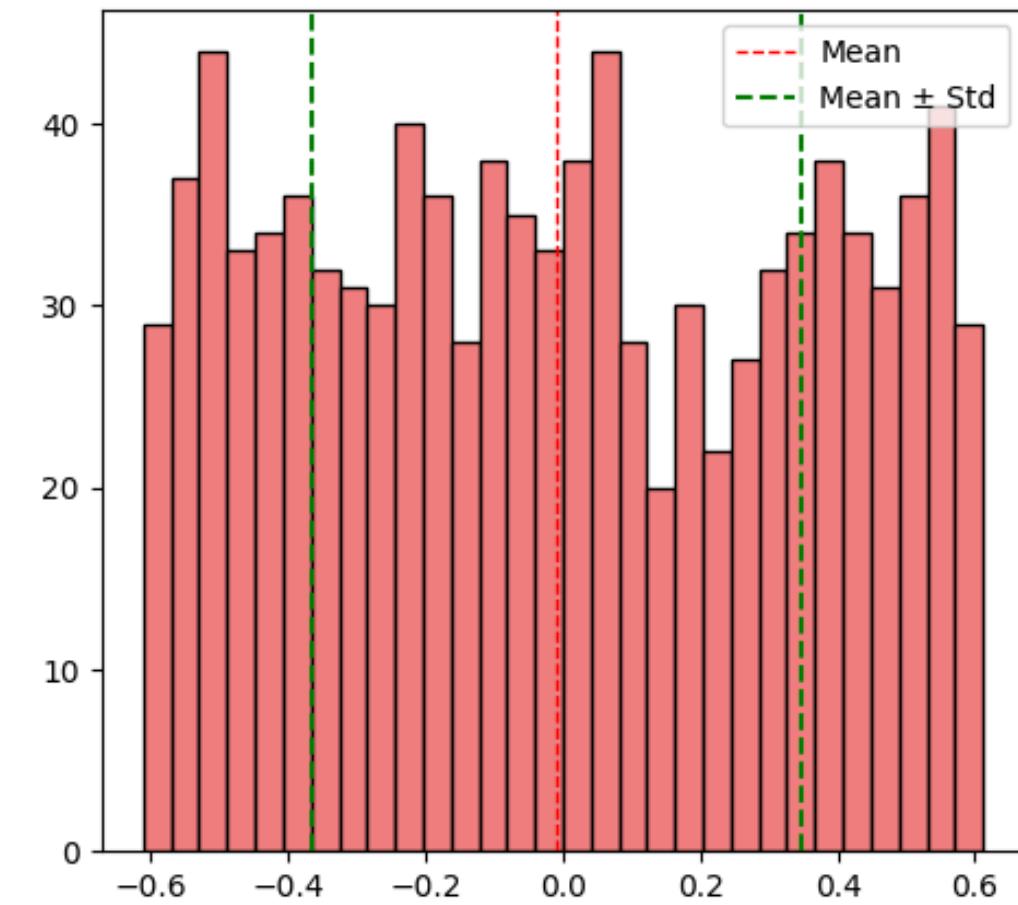
Normal Distribution:  $w \sim \mathcal{N}(0, \frac{2}{n_{in}})$

Uniform Distribution:  $w \sim U[-\sqrt{6/n_{in}}, \sqrt{6/n_{in}}]$

He Normal Initialization



He Uniform Initialization



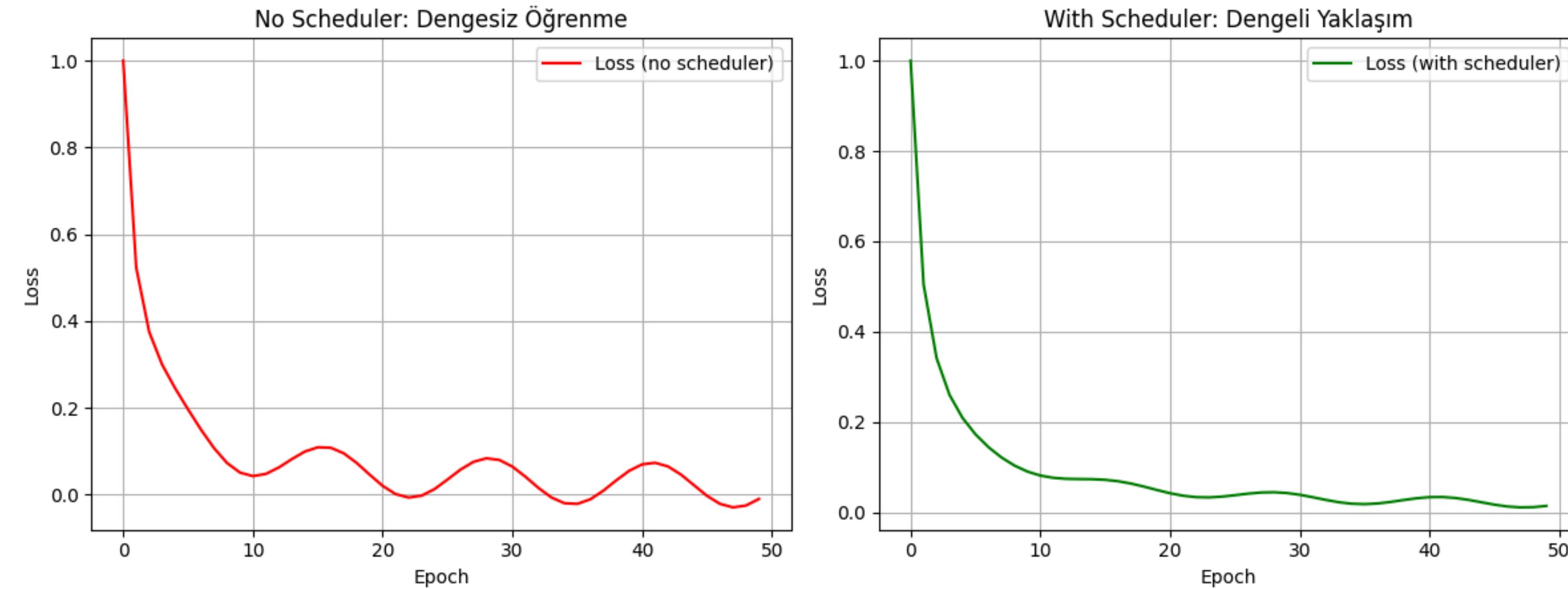
**ReLU ile ilgili problemi çözmek için tasarlanmıştır. Varyans ikiyle çarpılarak yarıya düşmenin önüne geçilmeye sağlanmıştır.**  
 **$n_{out}$  yok çünkü ReLU için kritik olan kısım giriş**

# Learning Rate Scheduler

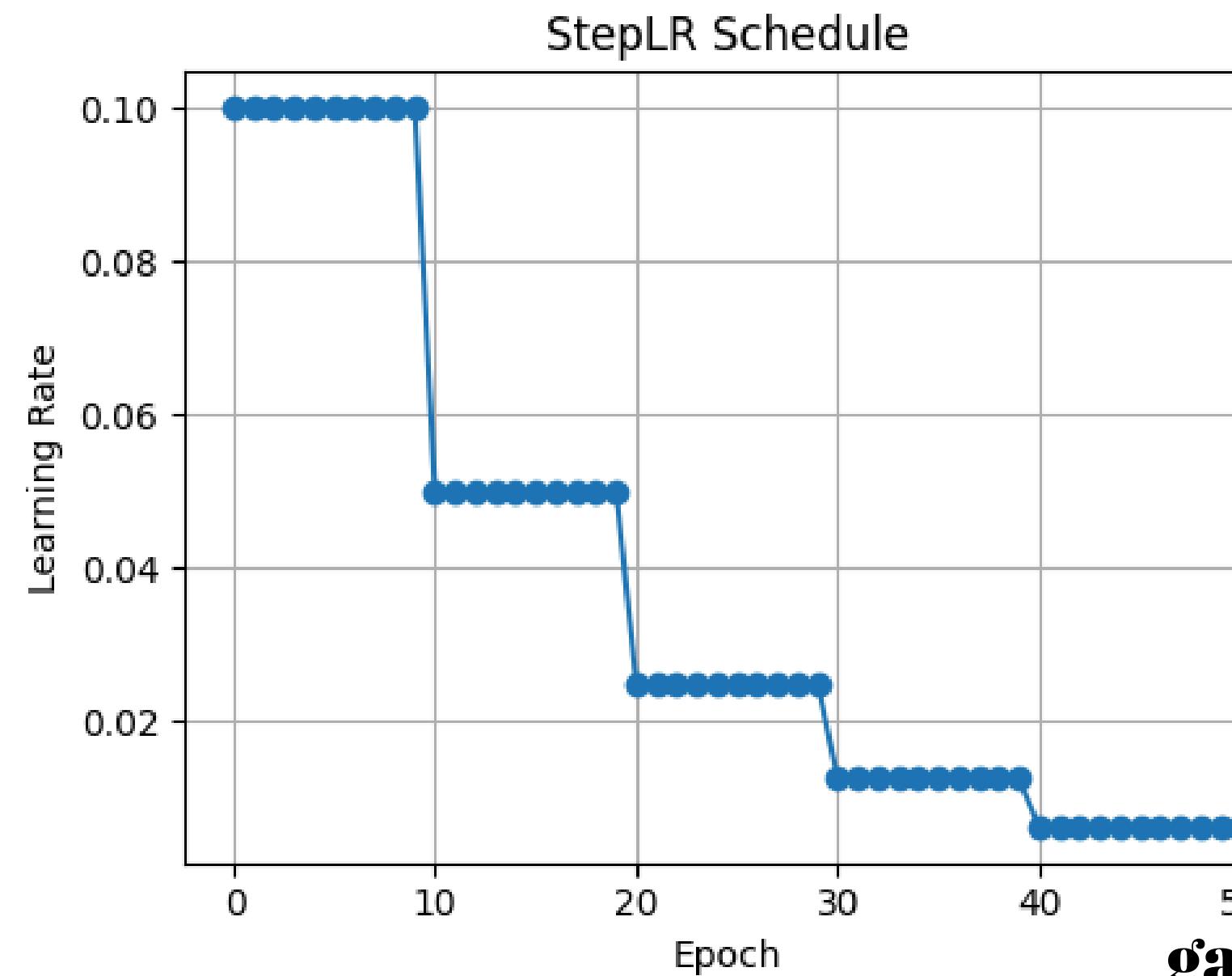
**Learning Rate değerinin eğitim sırasında değiştirilmesini sağlar.**

- **Başta büyük öğrenme oranlarıyla hızlı öğrenme sağlanması, sonra doğru küçük öğrenme oranlarıyla daha güvenli bir bitiş sağlanması**
- **Eğitimin sonunda küçük learning rate ile overfitting'in kısmen önlenmesi**
- **Oscillation (Sallanma) azaltılması**

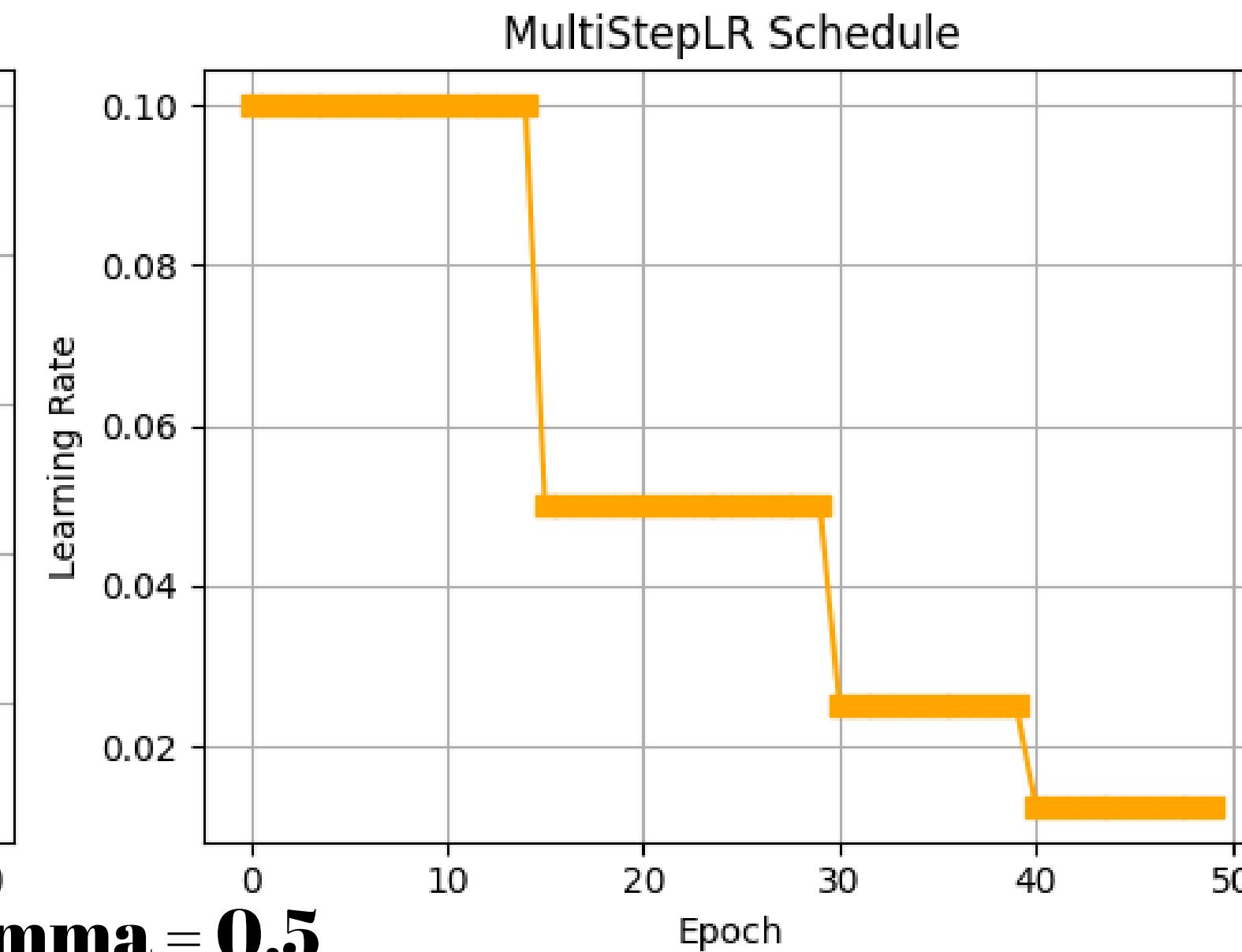
Learning Rate Scheduler Etkisi



## Step - Multistep LR



$$\text{gamma} = 0.5$$



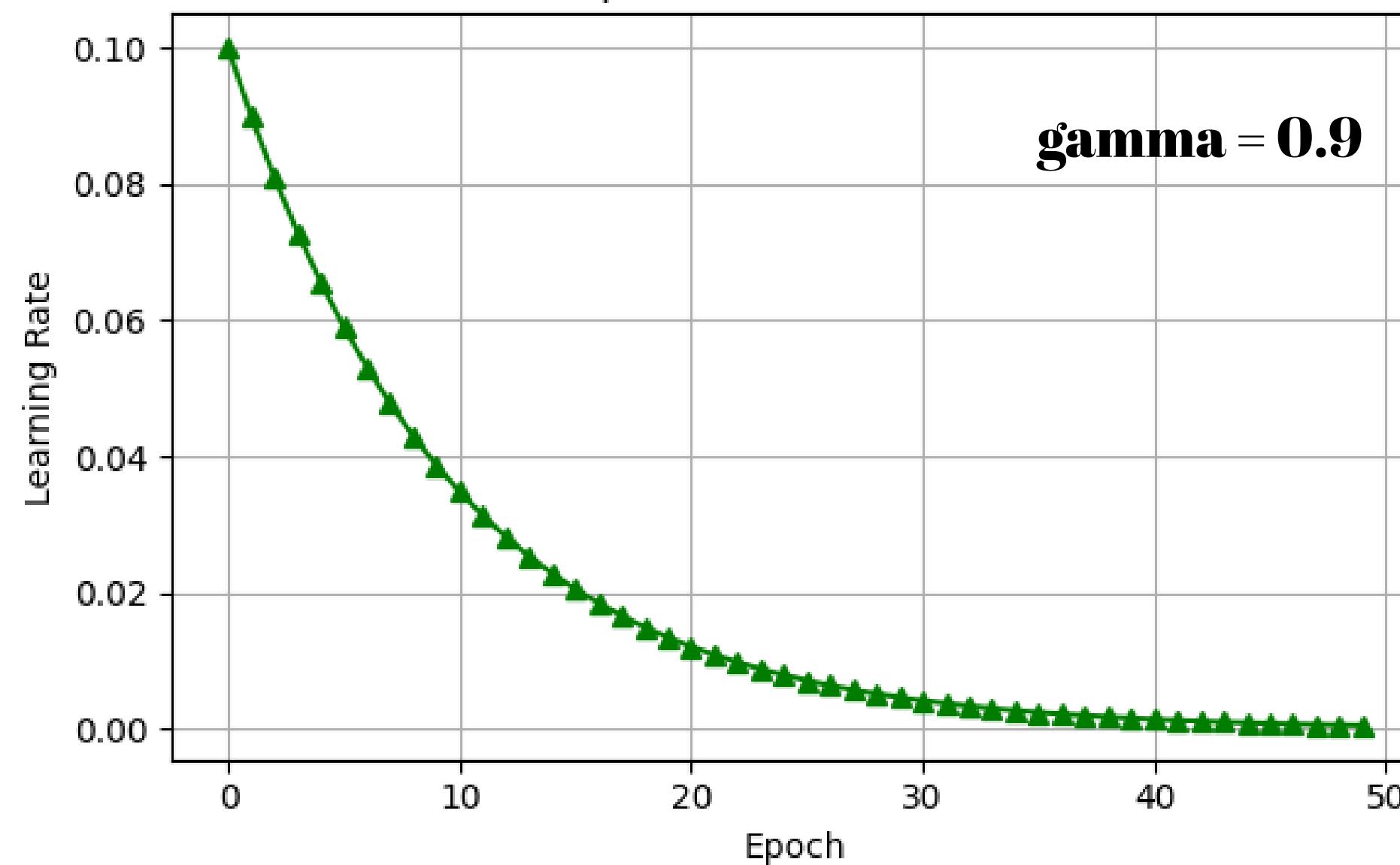
$$lr_t = lr_{t-1} * \text{gamma}$$

**Step Learning Rate:** Her n epoch'tan sonra learning rate'i gamma ile çarpar.

**Multistep Learning Rate:** Önceden belirlenmiş her epoch'tan sonra learning rate'i gamma ile çarpar.

# Exponential-Polynomial LR

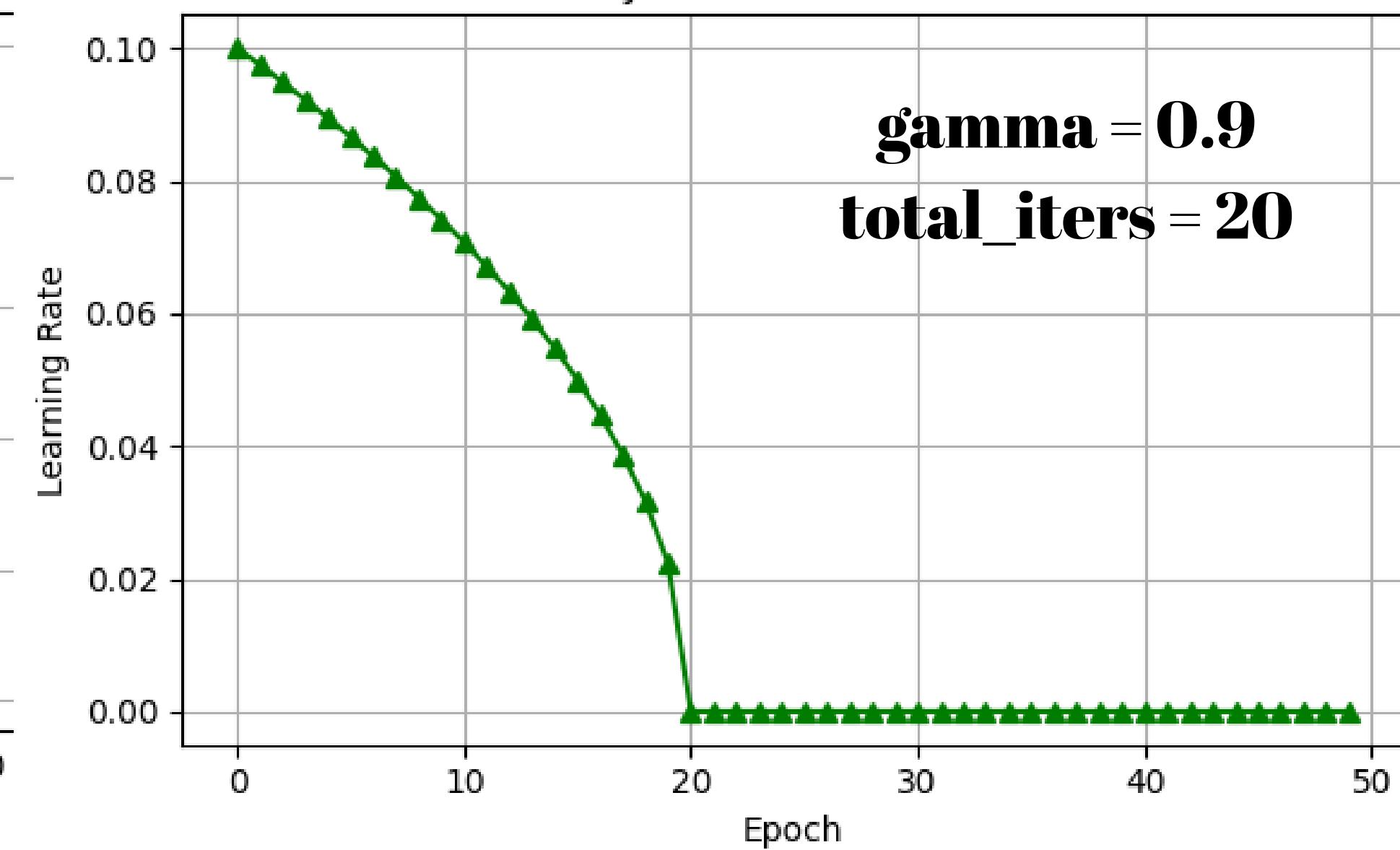
ExponentialLR Schedule



$$lr_t = lr_{t-1} * \text{gamma}$$

Her epoch için learning rate belirlenen  
gamma ile çarpılır

PolynomialLR Schedule

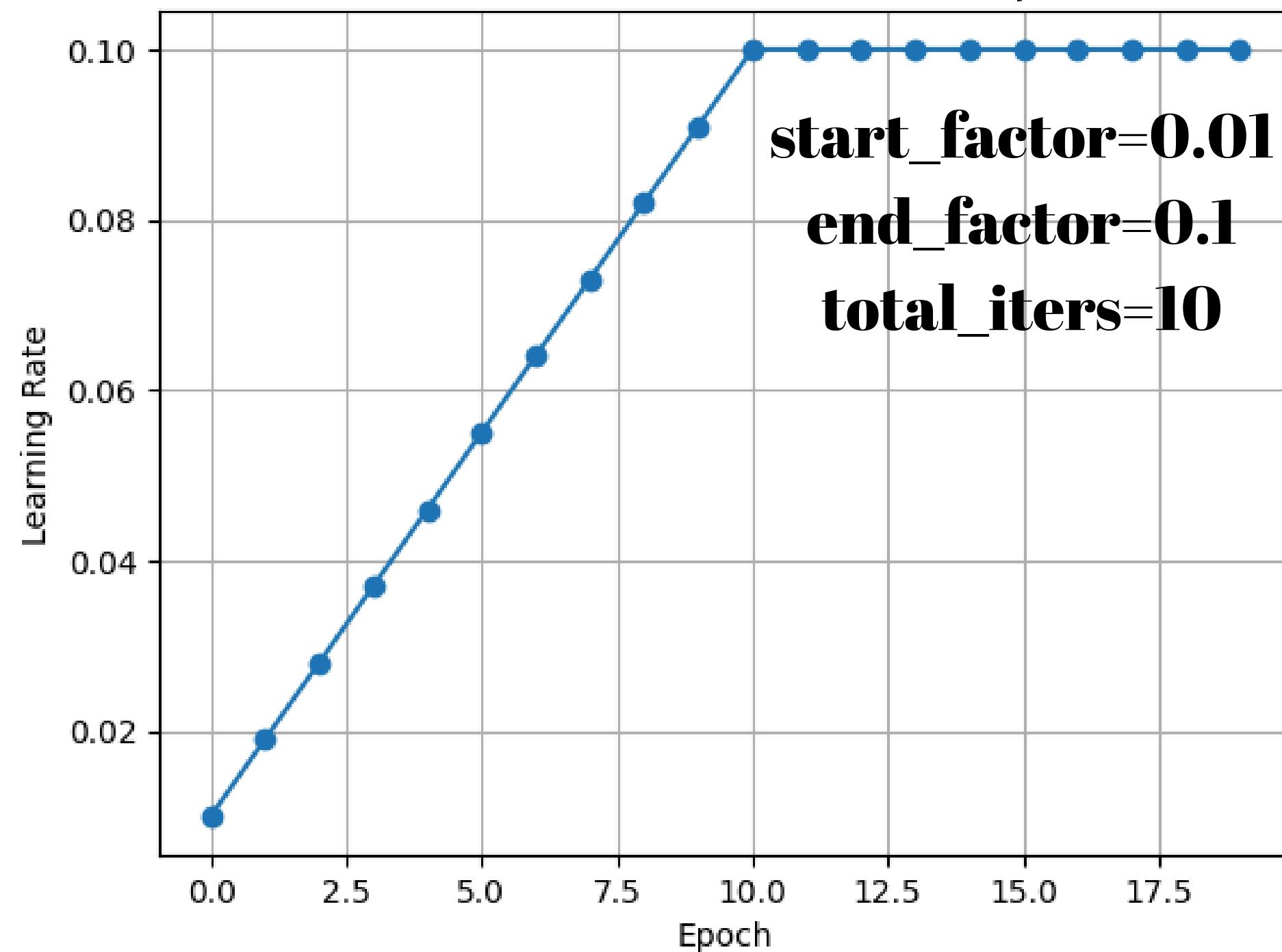


$$lr(t) = lr_0 \cdot \left(1 - \frac{t}{T}\right)^{\gamma}$$

t: current\_epoch  
T: total\_iters  
total\_iters'e kadar verilen polinom şeklinde  
learning\_rate'i günceller

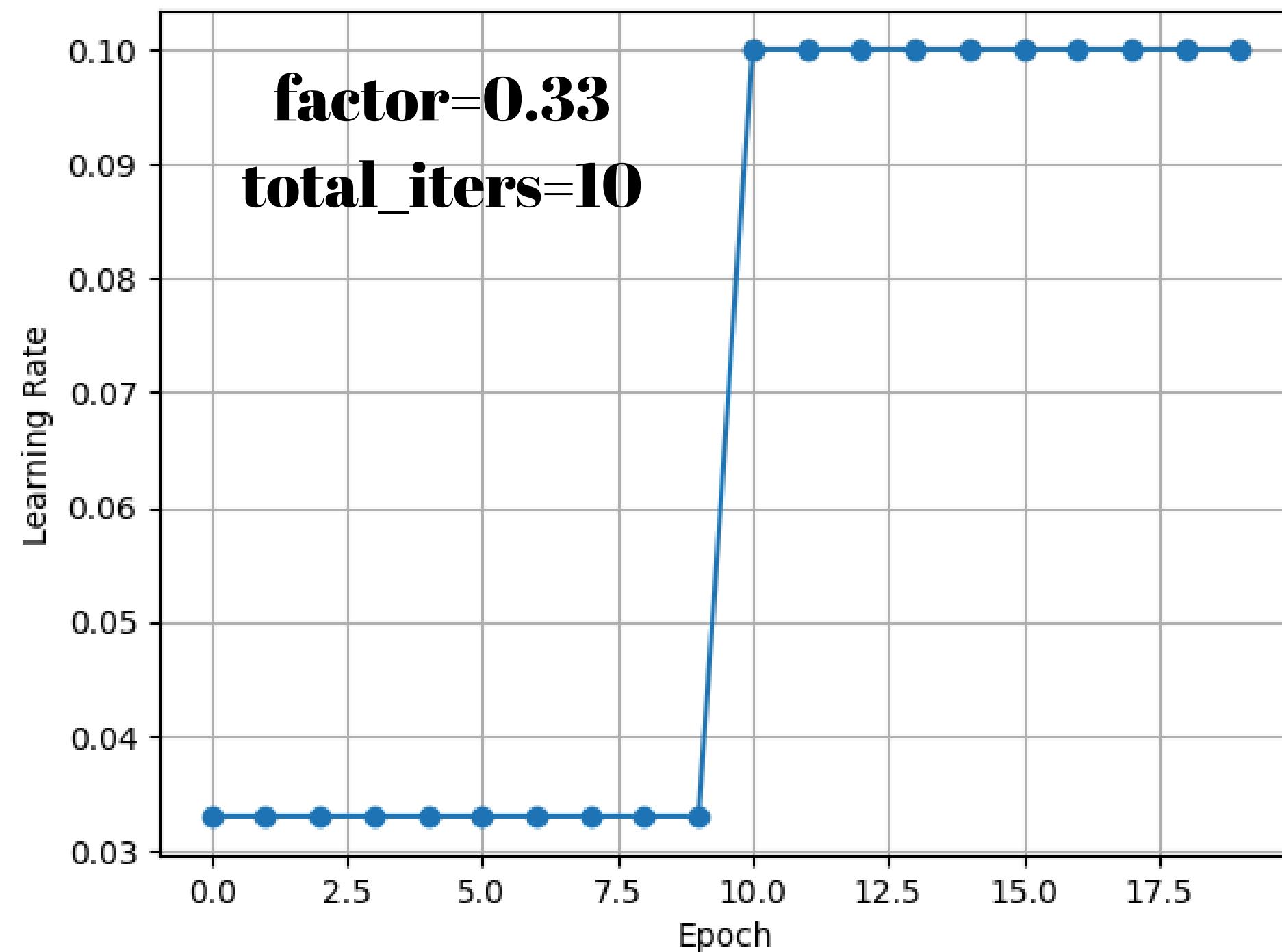
# Linear-Constant LR

LinearLR: from 0.01 to 0.1 over 10 epochs



**total\_iters boyunca start ve end arasında linear değişim sağlar**

ConstantLR



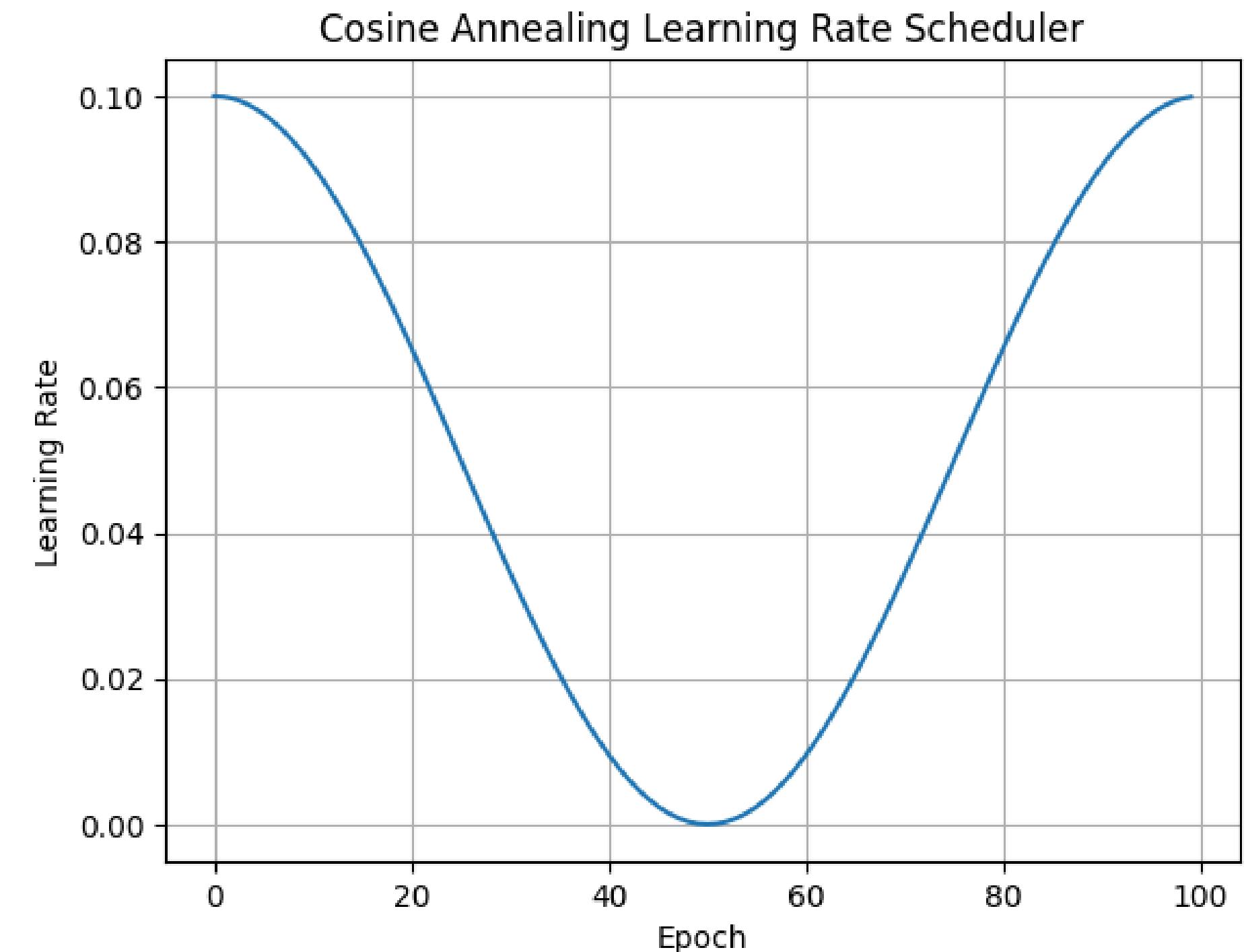
**total\_iters'e kadar lr\*factor.**  
**Sonra normal learning rate'e döner.**

# CosineAnnealingLR

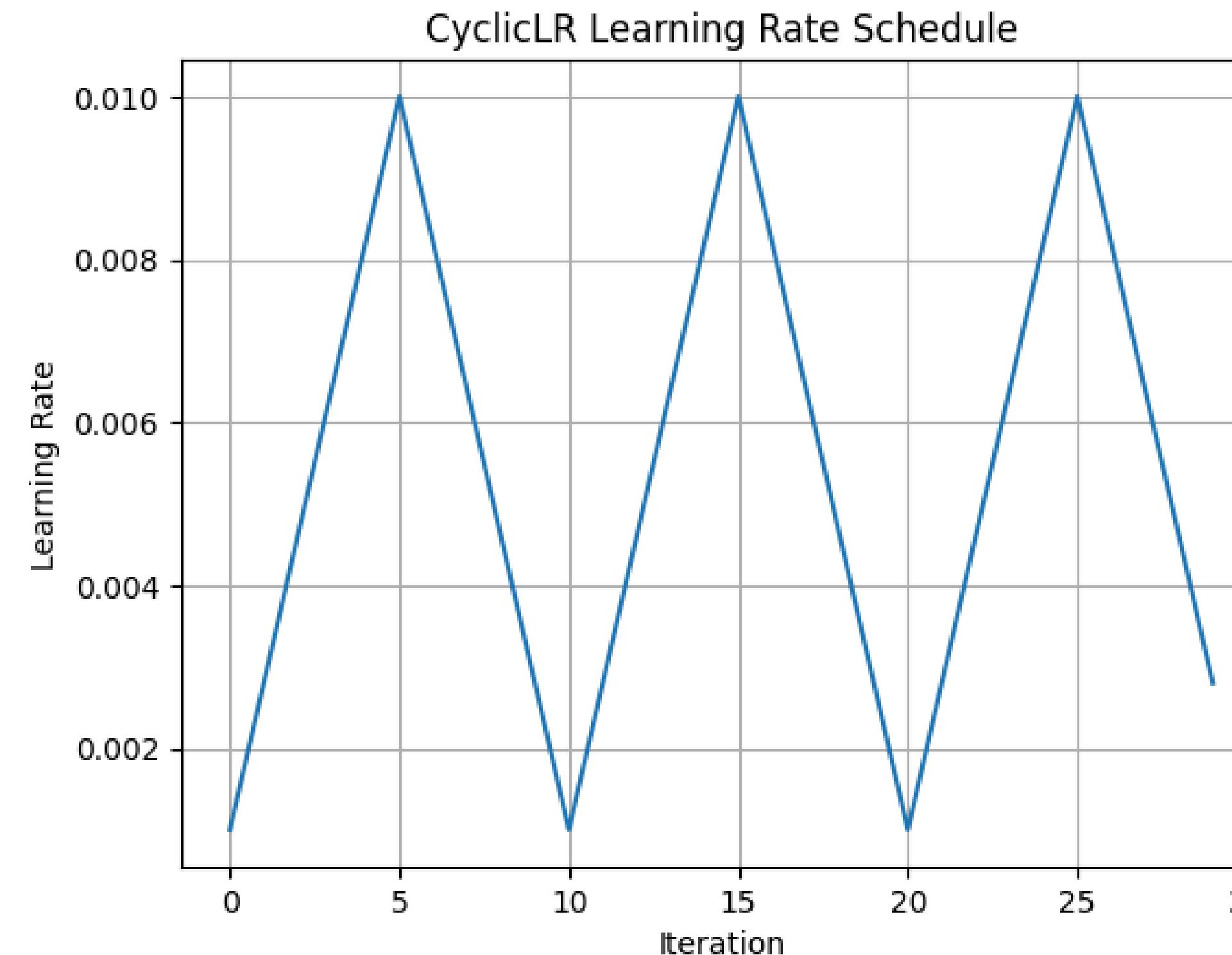
$$\eta_t = \eta_{min} + \frac{1}{2}(\eta_{max} - \eta_{min})\left(1 + \cos\left(\frac{T_{cur}}{T_{max}}\pi\right)\right)$$

$\eta_t$	: t. adımda kullanılan öğrenme oranı
$\eta_{max}$	: başlangıçtaki en yüksek öğrenme oranı
$\eta_{min}$	: minimum (düşebileceği en düşük) öğrenme oranı
$T_{cur}$	: şu anki epoch veya iteration sayısı
$T_{max}$	: öğrenme oranının sıfıra kadar düşeceği toplam epoch/iteration sayısı

- **Büyük verili ve uzun eğitimlerde kullanılır.**
- **Modelin lokal minimumlardan kurtulmasını sağlayabilir**

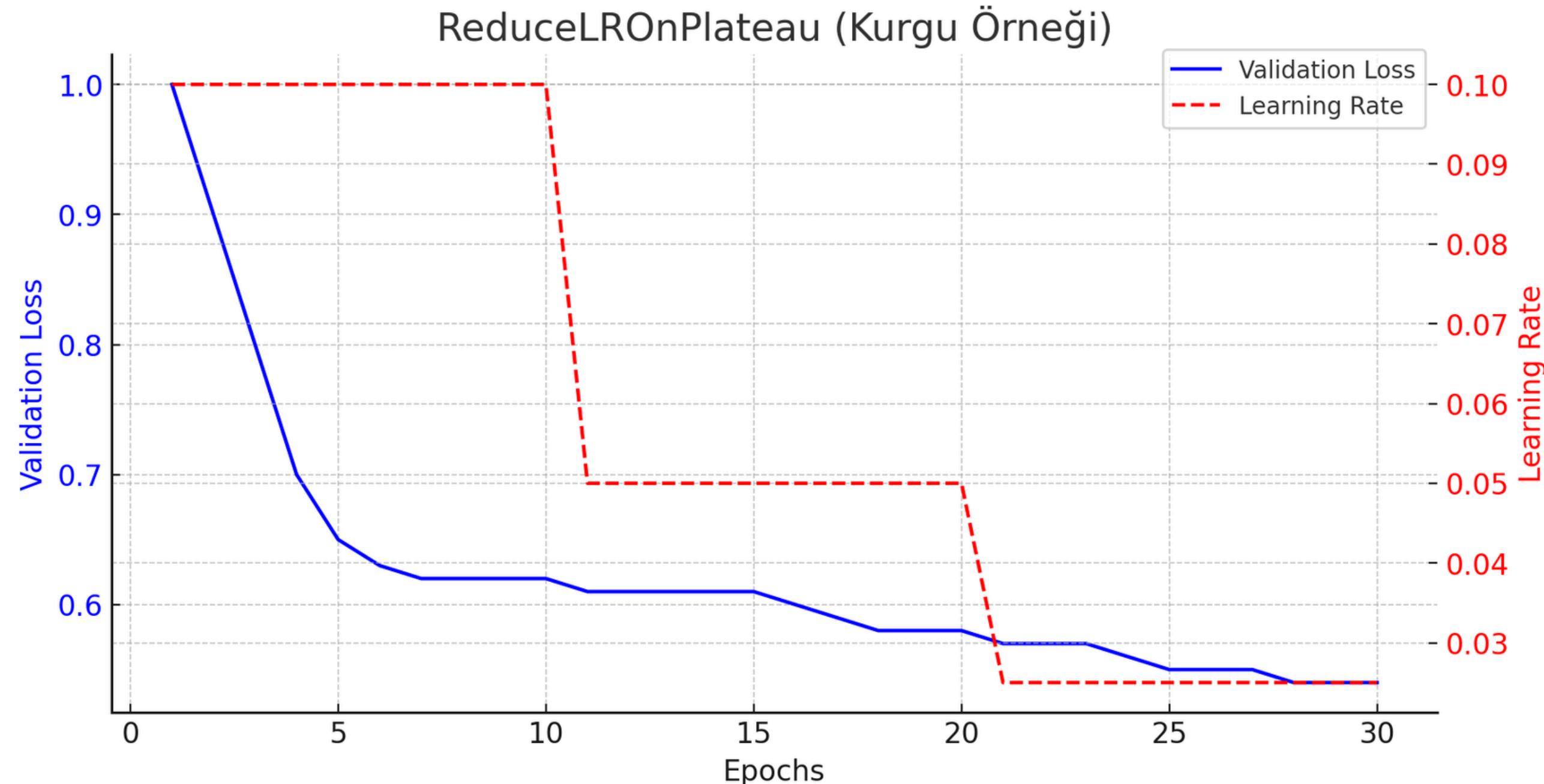


# CyclicLR



\*Cyclical Learning Rates for Training Neural Networks

# ReduceLROnPlateau



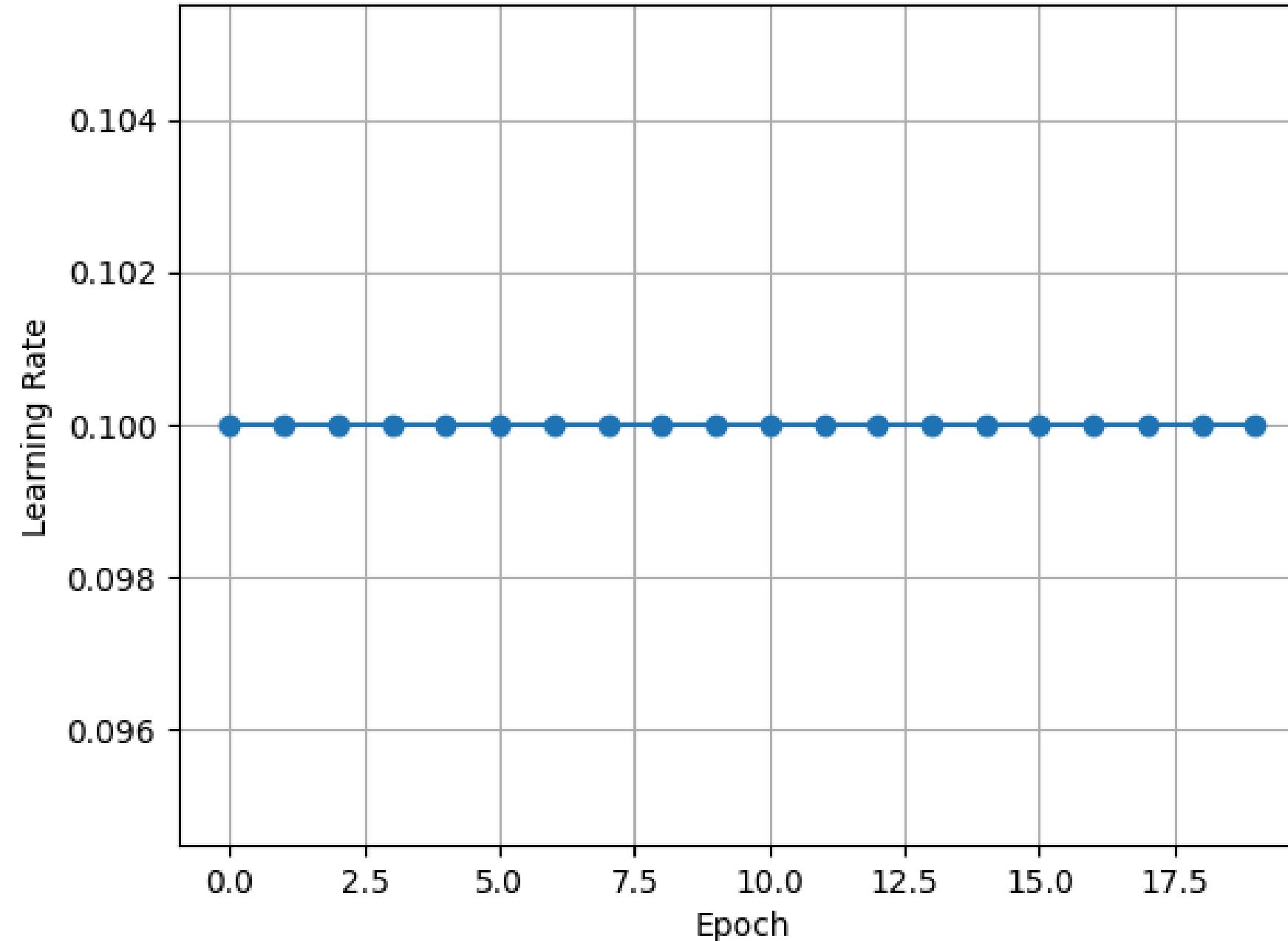
**Validation loss değişmediği durumda lr azaltılarak modelin öğrenmeye devam etmesi amaçlanır**

## **ChainedScheduler - SequentialScheduler**

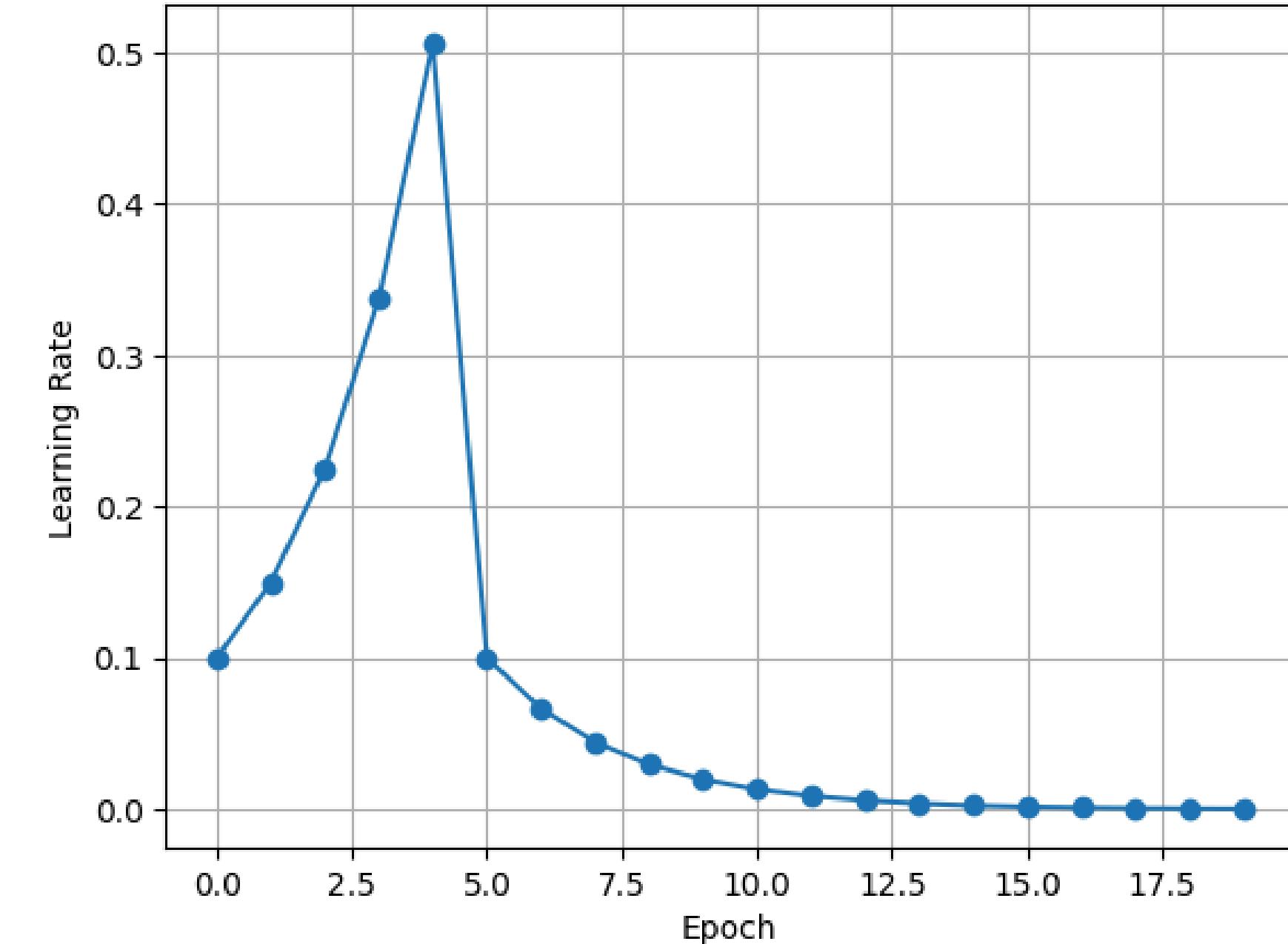
**expl = ExponentialLR(optimizer, gamma=3/2)**

**exp2 = ExponentialLR(optimizer, gamma=2/3)**

ChainedScheduler



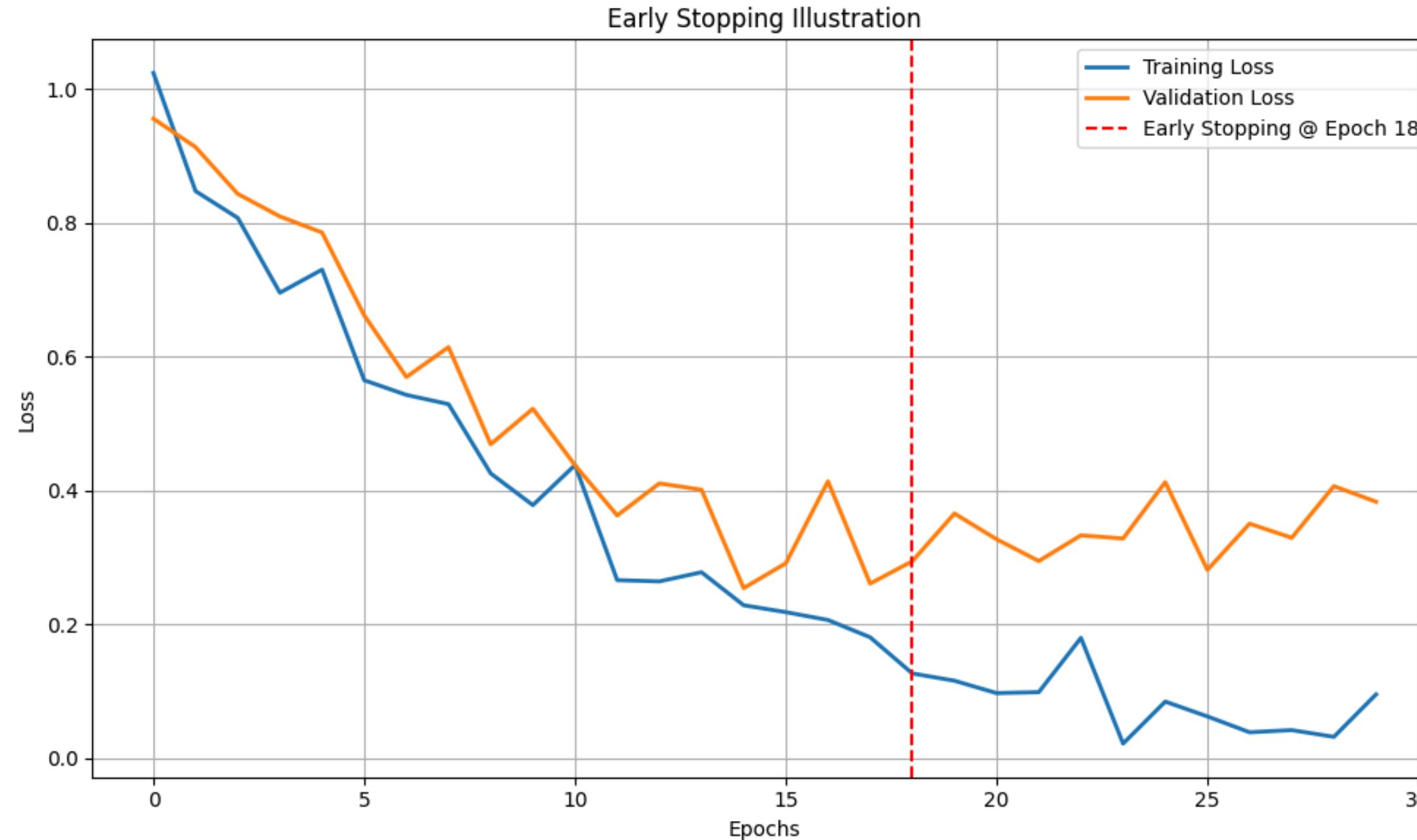
SequentialScheduler



**Birden fazla scheduler’ı aynı anda kullanmayı  
sağlar**

**Birden fazla scheduler’ı sırayla kullanmayı  
sağlar**

# Early Stopping



**Overfitting başladığı anda model eğitiminin sonlandırılması işlemidir.**  
**Val\_loss belirli bir süre azalmazsa model eğitimi durdurulabilir.**  
**Patience:** Kaç epoch boyunca gelişme olmazsa durdurulacak

## Motivation for CNN-Sequential Models

**Vanilla Neural Network yapısı çok güçlü bir yapı olsa da bazı veri tipleri üzerinde doğrudan başarı gösteremez. Bu yüzden bu veri tiplerine özgü olarak modifiye edilmiş NN'ler kullanılır.**

- **Resim-Video Verisi → Convolutional Neural Network (Module 5)**
- **Text Verisi → Sequential Models (Module 6)**
- **Zaman serisi verisi → Sequential Models (ins anlatcاز bi ara)**
- **Generative Task → GAN, VAE, Diffusion Models**
- **Deep Reinforcement → Q-Network**