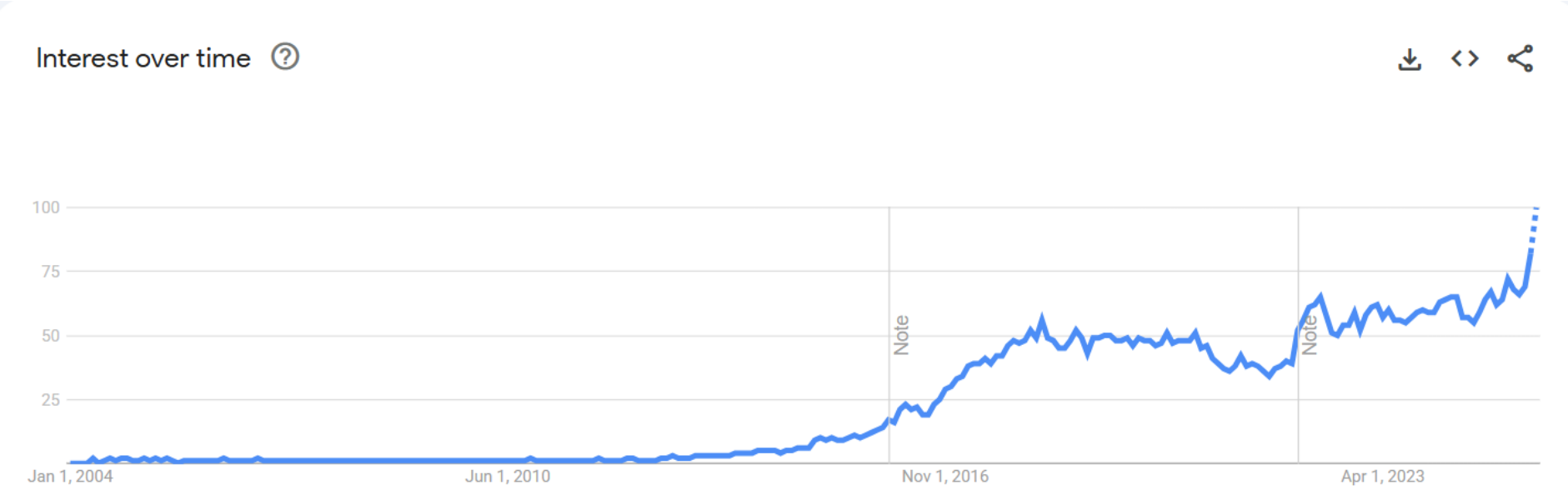
Dr.Öğr.Üyesi Furkan Göz

2025

Trends


Interest over time
100
75
50
25
Note
Note
Jan 1, 2004
Jun 1, 2010
Nov 1, 2016
Apr 1, 2023

Yapay Sinir Ağı (Artificial Neural Network)

- Biyolojik sinir sisteminden esinlenilerek geliştirilmiş matematiksel modeldir.
- Veri ile öğrenen bir yapı kurar:
  Girdi → Ağırlıklar → Aktivasyon → Çıktı
- Çok sayıda nöron ve katmandan oluşur (giriş, gizli, çıkış katmanı).

Nerelerde kullanılır?

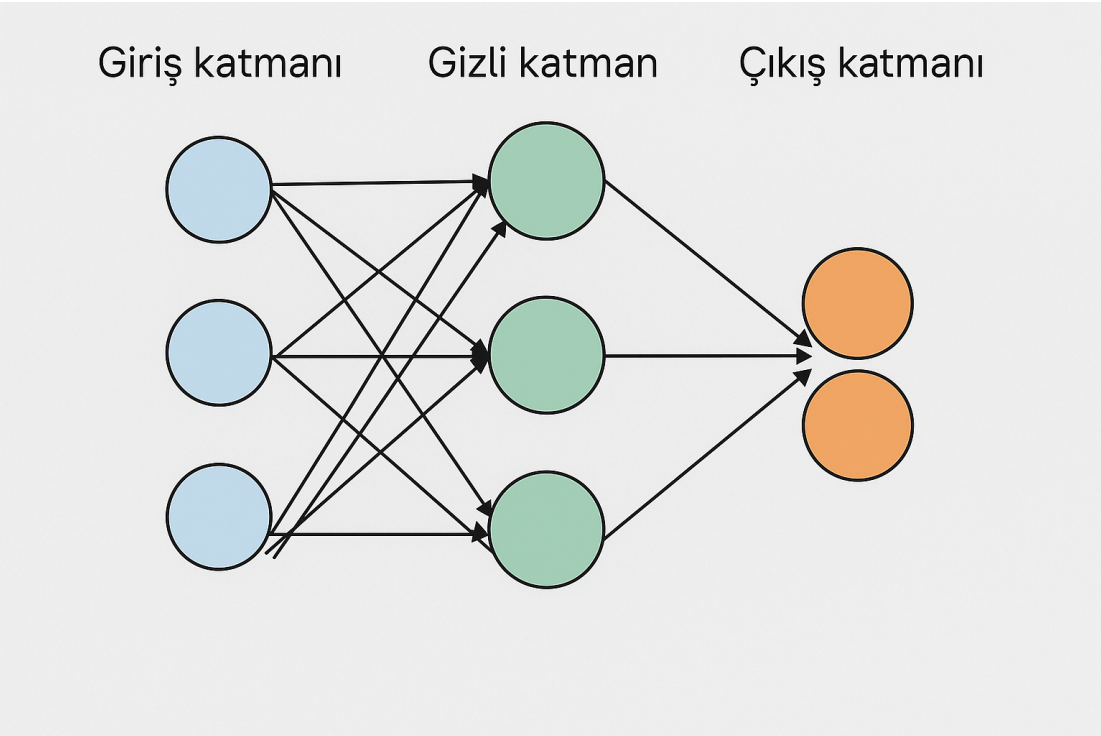
- Görüntü tanıma, ses işleme, metin sınıflandırma, öneri sistemleri

Neden güçlüdür?

- Karmaşık ilişkileri öğrenebilir, genelleme yeteneği yüksektir

YSA Katmanları: Giriş - Gizli - Çıkış

- Giriş katmanı: Ham verinin (sayısal özellikler) alındığı katmandır.
- Gizli katman(lar): Ağırlıklı toplama + aktivasyon fonksiyonu uygulanır.
- Çıkış katmanı: Tahmin veya sınıf sonucu üretilir.



Yapay Nöron Nasıl Çalışır?

- Her nöron, kendisine gelen verileri (girdiler) ağırlıklarla çarpar.
- Bu değerlerin toplamına bir bias eklenir.
- Sonuç bir aktivasyon fonksiyonuna girer ve çıktı üretilir.

Formül:

$$z = w_1x_1 + w_2x_2 + \cdots + w_nx_n + b \quad \Rightarrow \quad a = f(z)$$

Burada:

- x_i: girişler
- w_i: ağırlıklar
- b: bias
- f(z): aktivasyon fonksiyonu (ör: ReLU, sigmoid)

Veri ile örüntüleri yakalayacak şekilde ağırlık ve bias değerlerini öğrenmektir.

Bias Neden Eklenir?

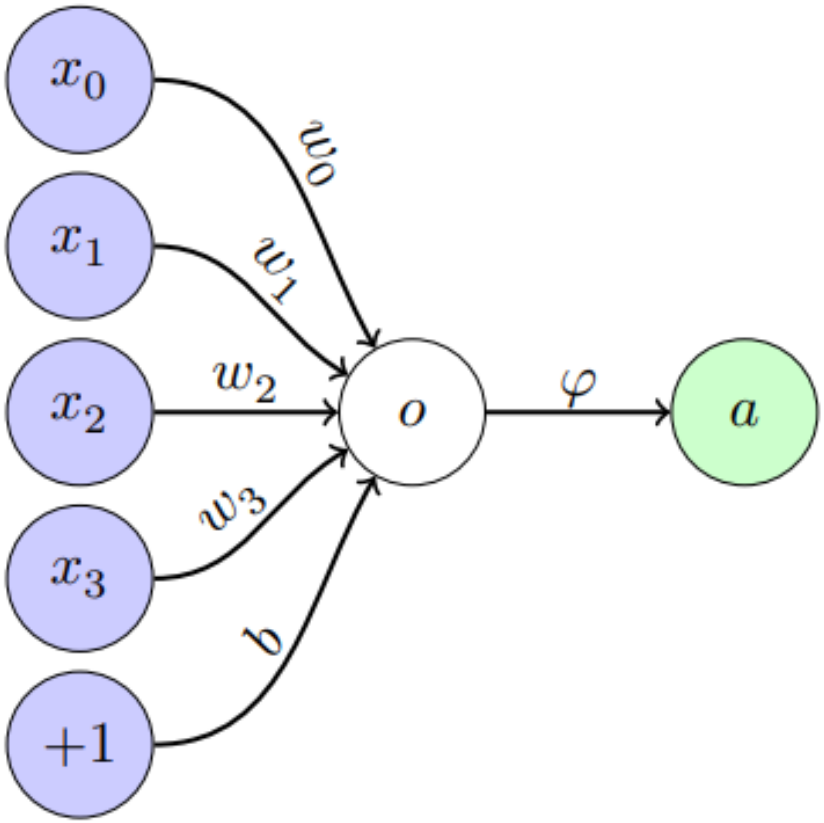
- Modelin yalnızca girişlere bağlı kalmadan, daha esnek bir karar sınırı öğrenmesini sağlar.
- Eğer bias olmasaydı, her nöron girişler sıfır olduğunda da sıfır üretirdi.

Örnek: $z = w_1x_1 + w_2x_2 + b$

- Eğer $w_1 = 1$, $x_1 = 2$, $b = 0$ ise $\rightarrow z = 2$
- Aynı durumda $b = +3$ olursa $\rightarrow z = 5$

Bias modelin çıkışını yukarı veya aşağı kaydırarak daha esnek tahminler yapmasını sağlar.

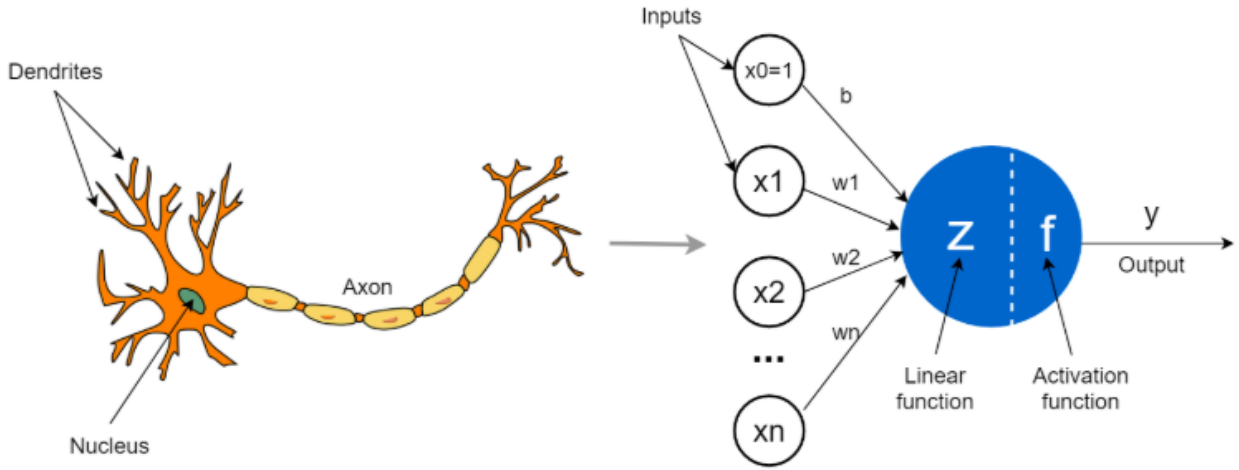
Percepton



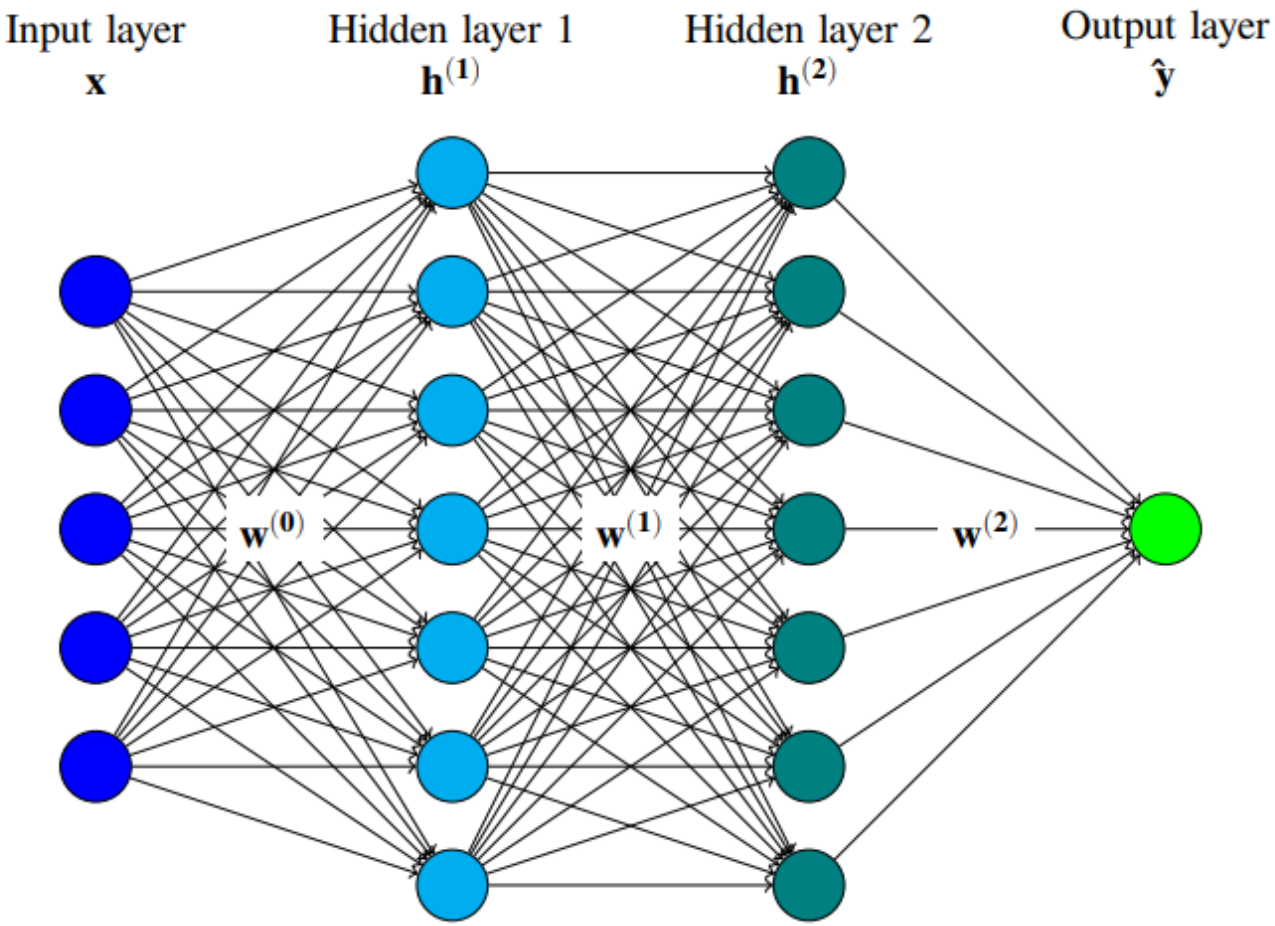
Nöronun Bileşenleri

- $x_1, x_2, ..., x_n$: Girdiler
- $w_1, w_2, ..., w_n$: Ağırlıklar
- z: Lineer toplam
- f(z): Aktivasyon sonucu

Nöron, veriyi işler → sonucu aktivasyon fonksiyonuna gönderir → diğer nöronlara aktarır.



Multilayer Percepton



Girdi ve Çıktı Katmanlarındaki Nöron Sayısı

Girdi Katmanı:

- Girdi katmanındaki nöron sayısı, veri setindeki bağımsız (açıklayıcı) değişkenlerin sayısına eşittir.
- Örnek: Veri setinde 4 özellik varsa → 4 giriş nöronu gerekir.

Çıktı Katmanı:

- Çıktı katmanındaki nöron sayısı, yapılmak istenen göreve bağlıdır:
  - Regresyon: Tahmin edilecek sürekli değişken sayısı kadar nöron
  - İkili sınıflandırma: 1 nöron → pozitif sınıfa ait olasılığı verir (genellikle sigmoid ile)
  - Çok sınıflı sınıflandırma: Sınıf sayısı kadar nöron (genellikle softmax ile)

Örnek Problem – 3 Sınıflı Sınıflandırma

- Veri setinde 4 adet açıklayıcı özellik var.
- Hedef değişken 3 sınıftan birine ait → çok sınıflı sınıflandırma problemi

Ağ Yapısı:

- Giriş katmanı: 4 nöron
- Çıktı katmanı: 3 nöron (her biri bir sınıf için)
- Aktivasyon: `Softmax`

Amaç: Her giriş için 3 sınıfa ait olasılık üretmek.

Gizli Katman Sayısı ve Nöronlar

- Girdi ve çıktı katmanı sayısı veriye göre belirlenir.
- Ancak gizli katman sayısı ve her katmandaki nöron sayısı, modelin başarımını etkileyen hiperparametrelerdir.
- Bu değerler genellikle:
  - Deneme-yanılma (grid search, random search)
  - Uzman bilgisi
  - Otomatik ayarlama yöntemleri (AutoML)

Fazla nöron veya katman → overfitting riski
Az nöron → öğrenme yetersiz olabilir

Iris Veri Seti

```python
from sklearn.datasets import load_iris
import numpy as np
import pandas as pd
data = load_iris()
X = data.data
y = data.target
df = pd.DataFrame(X, columns=data.feature_names)
df["target"] = y
input_dim = X.shape[1]
output_dim = len(np.unique(y))
print(df.head())
print(f"Giris: {input_dim}")
print(f"Cikis: {output_dim}")
```

Neden 4 Giriş, 3 Çıkış Nöronu?

Giriş Katmanı:

- Sepal uzunluğu, sepal genişliği, petal uzunluğu, petal genişliği
- Toplam 4 özellik $\rightarrow$ 4 nöron

Çıkış Katmanı:

- 3 sınıf: Setosa, Versicolor, Virginica
- Her sınıfa ait olasılık üretileceği için $\rightarrow$ 3 çıkış nöronu
- Aktivasyon: Softmax

Sonuç: Ağ mimarisi doğrudan veri yapısına bağlı olarak tanımlanır.

Aktivasyon Fonksiyonu

- Derin öğrenme modelleri, günümüzün üretici yapay zeka (generative AI) sistemlerinin temelidir.
- Aktivasyon fonksiyonu, yapay sinir ağında her bir nöronun çıktısını belirler.
- Girdi bilgisine göre, nöronun çıktı üretip üretmeyeceğine yani bu bilginin bir sonraki katmana aktarılıp aktarılmayacağına karar verir.

Amaç: Ağın öğrenme kapasitesini artırmak için doğrusal olmayanlık (non-linearity) kazandırmak.

Aksiyon Potansiyeli

- Aktivasyon fonksiyonlarının temel fikri, biyolojik sinir sistemindeki aksiyon potansiyeli kavramına dayanmaktadır.
- Aksiyon potansiyeli: Bir sinir hücresinin (nöronun), belirli bir elektriksel eşik değeri aşıldığında diğer nöronlara sinyal göndermesi sürecidir.
- Yapay sinir ağlarında bu mekanizma, nörona gelen toplam sinyalin bir aktivasyon fonksiyonu aracılığıyla işlenmesiyle modellenir.
- İlk örneklerden biri 1962 yılında Frank Rosenblatt tarafından önerilmiştir.

Aktivasyon fonksiyonu, nörona gelen toplam girdinin belli bir değeri aşıp aşmadığını değerlendirerek çıktı üretip üretmeyeceğini belirler.

Doğrusal Olmayanlık Neden Gerekli?

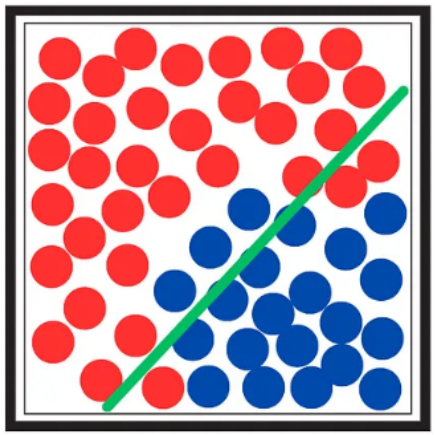
- Sinir ağı katmanlarında sadece doğrusal işlemler varsa:
  - Tüm model tek bir matris çarpımı gibi davranır.
  - Karmaşık örüntüleri ayırt edemez.
- Aktivasyon fonksiyonu:
  - Modele doğrusal olmayanlık katar.
  - Öğrenme kapasitesini ciddi şekilde artırır.

Doğrusal Olmayanlık (Non-Linearity)

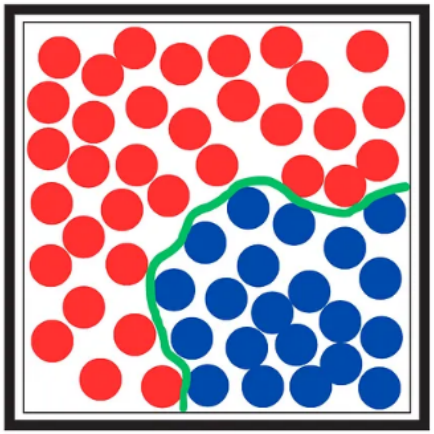
- Gerçek dünya verisi çoğunlukla doğrusal olmayan (non-linear) yapıya sahiptir.
- Sadece doğrusal (linear) işlemlerle karmaşık problemler çözülemez.
- Aktivasyon fonksiyonu, sinir ağına doğrusal olmayanlık kazandırarak bu örüntüleri öğrenmesini sağlar.

Örnek: Köpek/kedi tanıma problemi – Aynı türdeki hayvanlar farklı poz, açı ve arka planda olabilir. Bu çeşitlilik ancak doğrusal olmayan modellerle yakalanabilir.

Doğrusal?



Neural Network without an Activation Function



Neural Network with an Activation Function

Aktivasyon Fonksiyonları Neden Gerekli?

- Katmanlar arası yalnızca doğrusal (lineer) işlem yapılırsa model karmaşık ilişkileri öğrenemez.
- Aktivasyon fonksiyonu: ağırlıklı toplam üzerine uygulanan doğrusal olmayan dönüşüm.
- Derin öğrenme modelinin doğrusal olmayan örüntüleri öğrenmesini sağlar.

Amaç: Modelin daha esnek, güçlü ve genelleyebilir olmasını sağlamak.

Gradyan Yayılımı ve Öğrenme Süreci

- Sinir ağı eğitilirken, ağırlıklar geri yayılım (backpropagation) algoritması ile güncellenir.
- Aktivasyon fonksiyonunun türevi (gradyanı), her ağırlığın ne kadar güncelleneceğini belirler.
- Yanlış tahmin yapıldığında bu gradyanlar sayesinde hatalar geri yayılır.

Öğrenme sürecinin temel taşı, aktivasyon fonksiyonunun türevidir.

Karar Verme Mekanizması

- Her nöron, önceki katmandan gelen ağırlıklı toplamı alır.
- Aktivasyon fonksiyonu bu değere göre nöronun çıkış üretip üretmeyeceğine karar verir.
- Böylece, bazı özellikler daha önemli kabul edilir; bazıları bastırılır.

Örnek: Görseldeki kenarlar, şekiller, desenler gibi özelliklerin ağırlığı farklı olabilir. Aktivasyon fonksiyonu bu ayrımı yapar.

Girdiler ile Çıktılar Arasındaki İlişki

- Sinir ağı katman katman öğrenir: alt katmanlar basit, üst katmanlar daha karmaşık yapılar yakalar.
- Aktivasyon fonksiyonu, bu katmanlar arasında bilgi dönüşümünü sağlar.
- Böylece ağ daha üst seviyeli özellikleri birleştirerek sonuç üretir.

Örnek: Alt katman: kenar – Üst katman: kulak – En üst katman: “kedi” nesnesi

Örnek: Basit bir ikili sınıflandırma problemi düşünelim. Girdiler ve öğrenilmiş ağırlıklar:

- $a = 2, \quad b = 3$
- $w_1 = 0.5, \quad w_2 = 0.3$
- $\text{bias} = 2$

İleri Yayılım (Forward Propagation):

$$z = w_1 \cdot a + w_2 \cdot b + \text{bias} = 0.5 \cdot 2 + 0.3 \cdot 3 + 2 = 3.9$$

Aktivasyon (Sigmoid):

$$f(z) = \frac{1}{1 + e^{-z}} = \frac{1}{1 + e^{-3.9}} \approx 0.98$$

Aktivasyon Değerinin Yorumu

- Hesapladığımız değer: $f(z) \approx 0.98$
- Bu, nöronun "aktif olma" olasılığını temsil eder.
- Sigmoid fonksiyonu çıktıyı $[0, 1]$ aralığına indirger $\rightarrow$ olası sınıf yorumlamasına uygundur.

Yorum:

- f(z) değeri 1'e ne kadar yakınsa, model bu girdilerin belirli bir sınıfa ait olduğundan o kadar emindir.
- Bu nedenle ikili sınıflandırmada sigmoid sıkça tercih edilir.

Aktivasyon Fonksiyonu Olmadan Ne Olur?

- Katmanlar arasında hiçbir aktivasyon fonksiyonu kullanılmazsa:
  - Tüm ağ sadece doğrusal işlemler dizisi haline gelir.
  - Karmaşık örüntüler öğrenilemez.
- Modelin çıktısı, girişlerin lineer bir kombinasyonu olur.
- Derinliğin anlamı kalmaz.
- Bu durumda, model sadece veriyi düz bir çizgi (veya düzlem) ile ayırabilir.

Sonuç: Aktivasyon fonksiyonu olmadan derinlik anlamsız hale gelir; model karmaşık görevlerde başarısız olur.

Softmax Aktivasyon Fonksiyonu

Amaç: Çok sınıflı sınıflandırma problemlerinde, her sınıfa ait olasılığı hesaplamak.

Tanım: Softmax, birden fazla çıktı nöronunun her birine olasılık atar. Çıktıların toplamı daima 1 olur.

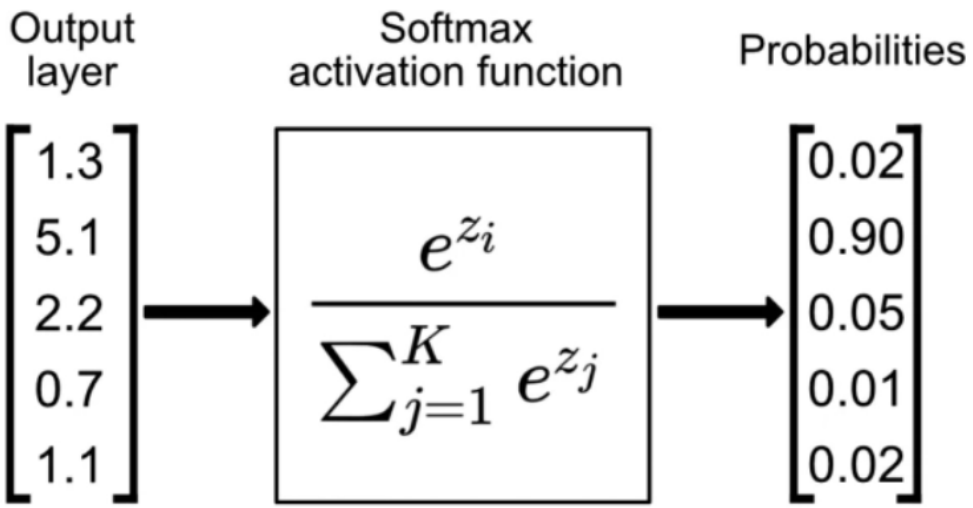
Formül:

$$f(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \quad \text{(her sınıf i için)}$$

Özellikler:

- K: sınıf sayısı
- z_i: her bir sınıf için skor (model çıkışı)
- Çıktılar $[0, 1]$ aralığında olur ve toplamları 1'dir

Softmax


Output layer
Softmax activation function
Probabilities
1.3
5.1
2.2
0.7
1.1
$\frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$
0.02
0.90
0.05
0.01
0.02

Yaygın Aktivasyon Fonksiyonları

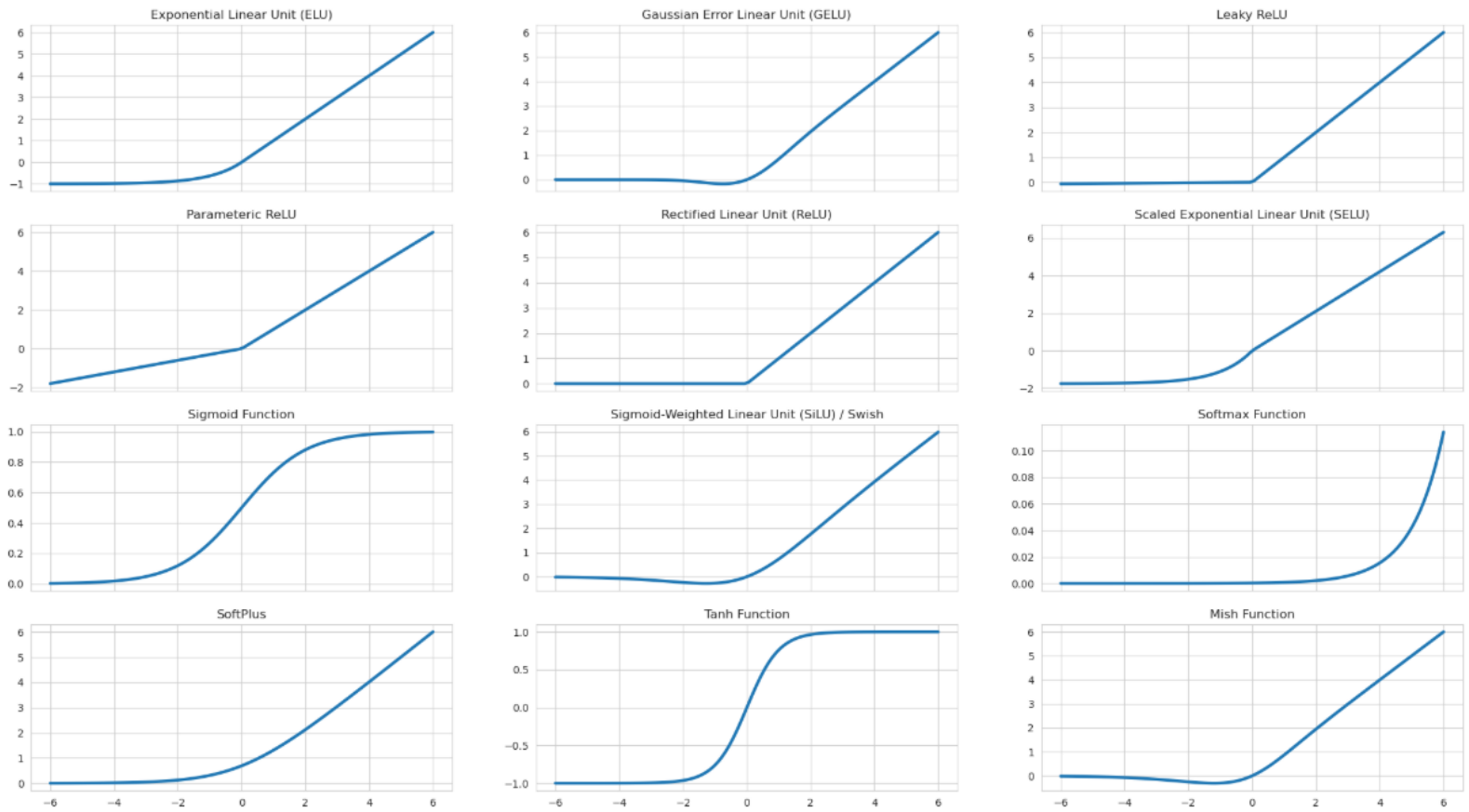
- Sigmoid: $f(x) = \frac{1}{1+e^{-x}}$ $(0, 1)$ aralığında çıktı üretir.
- Tanh: $f(x) = \tanh(x)$ $(-1, 1)$ aralığında çıktı verir, sıfır merkezlidir.
- ReLU: $f(x) = \max(0, x)$ negatif girişleri sıfırlar.

Hangi Durumda Hangi Fonksiyon?

- Sigmoid: İkili sınıflandırma çıktı katmanında
- ReLU: Genel amaçlı, hızlı öğrenme sağlar
- Tanh: Sıfır merkezli çıktı gereken durumlar
- Leaky ReLU, ELU: Ölü nöron riskine karşı
- GELU: Transformer mimarilerinde

Aktivasyon fonksiyonu doğru seçilmezse öğrenme durabilir!

Aktivasyon


Exponential Linear Unit (ELU)
Gaussian Error Linear Unit (GELU)
Leaky ReLU
Parameteric ReLU
Rectified Linear Unit (ReLU)
Scaled Exponential Linear Unit (SELU)
Sigmoid Function
Sigmoid-Weighted Linear Unit (SiLU) / Swish
Softmax Function
SoftPlus
Tanh Function
Mish Function

TensorFlow ile Sinir Ağı Modeli Tanımı

```
1  from tensorflow.keras.models import Sequential
2  from tensorflow.keras.layers import Dense, InputLayer
3
4  model = Sequential([
5      InputLayer(shape=(10, )),
6      Dense(units=20, activation="relu"),
7      Dense(units=3, activation="softmax")
8  ])
9
10 model.summary()
```

- TensorFlow'un Keras API'si ile çok sınıflı bir model tanımlanır
- Modelin giriş, gizli ve çıkış katmanları net biçimde belirtilmiştir

Model Mimarisi Açıklaması

- Giriş Katmanı:
  - `InputLayer(shape=(10, ))` → Model 10 özellikten oluşan bir veri bekliyor
- Gizli Katman:
  - `Dense(20, activation="relu")` → 20 nöron, ReLU aktivasyonu ile çalışır
  - Bu katman, öğrenilecek temsil (feature) çıkarımı yapar
- Çıkış Katmanı:
  - `Dense(3, activation="softmax")` → 3 sınıf için olasılık çıktısı üretir
- model.summary():
  - Katman yapısı ve öğrenilebilir parametre sayıları gösterilir

Not: Bu yapı, çok sınıflı sınıflandırma görevleri için uygundur.

MLP Modeli (Iris Veriseti)

```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, InputLayer

model = Sequential([
    InputLayer(input_shape=(4, )),
    Dense(units=100, activation="relu"),
    Dense(units=3, activation="softmax")
])
```

- 4 giriş özelliği: `(sepal/petal uzunluk/genişlik)`
- 100 nöronlu ReLU aktivasyonlu gizli katman
- 3 sınıflı çıktı → `softmax` ile olasılık dağılımı

Kayıp Fonksiyonu (Loss Function) Nedir?

- Modelin tahmini ile gerçek değer arasındaki farkı ölçer.
- Bu fark, sayısal bir “kayıp” değeri olarak ifade edilir.
- Amaç: Bu kaybı en aza indirerek daha doğru tahminler yapmak.
- Kayıp ne kadar küçükse, model o kadar başarılı demektir.

Not: Kayıp fonksiyonu olmadan model öğrenemez.

Neden Kayıp Fonksiyonuna İhtiyaç Duyarız?

- Sinir ağı bir tahmin yapar, örneğin: `[0.2, 0.3, 0.5]`
- Gerçek sınıf: `[0, 0, 1]`
- Kayıp fonksiyonu, bu iki vektör arasındaki farkı sayısal olarak ölçer.
- Bu fark (kayıp), ağın ağırlıklarının nasıl güncelleneceğini belirler.
- Öğrenme süreci, bu kaybı azaltacak şekilde ağırlıkları günceller.

Kayıp = Modelin yaptığı hata

Farklı Kayıp Fonksiyonları

1. Mean Squared Error (MSE):
   - Regresyon problemlerinde kullanılır.
   - Tahmin ve gerçek değer farklarının karesinin ortalaması.
2. Binary Crossentropy:
   - İkili (binary) sınıflandırma için uygundur.
   - Tahmin edilen olasılıkla gerçek değerin uyumunu ölçer.
3. Categorical Crossentropy:
   - Çok sınıflı sınıflandırmalarda kullanılır (örneğin Iris).
   - Softmax çıkışları ile birlikte çalışır.

Iris Veriseti için Doğru Kayıp Fonksiyonu

- Iris veri setinde 3 sınıf vardır.
- Modelin çıkışı: `Dense(3, activation="softmax")`
- Bu durumda kullanılacak doğru kayıp fonksiyonu:
  - `categorical_crossentropy` (etiketler one-hot ise)
  - `sparse_categorical_crossentropy` (etiketler integer kodlu ise)

Not: Etiket formatı loss seçiminde önemlidir.

Optimizasyon Nedir?

- Modelin yaptığı tahmini, gerçek değere yaklaştırmak için ağırlıkları güncelleme işlemidir.
- Amaç: Kayıp fonksiyonunun (loss) değerini azaltmak.
- Bu işlem her eğitim örneği için (veya mini-batch için) tekrar edilir.

Optimizasyon Nasıl Çalışır?

- Her ileri yayılımda (forward pass), bir tahmin yapılır.
- Tahmin ile gerçek değer arasındaki fark → kayıp (loss) hesaplanır.
- Optimizasyon algoritması bu kaybı azaltacak şekilde ağırlıkları günceller.
- Bu süreçte gradyanlar (türevler) kullanılır → geriye yayılım (backpropagation)

Model her iterasyonda daha iyi hale gelir.

Yaygın Optimizasyon Algoritmaları

1. SGD (Stochastic Gradient Descent)
   - Basit ve yaygın kullanılan yöntem
   - Ağırlıkları gradyan yönünde küçük adımlarla günceller
2. Adam
   - En popüler yöntemlerden biridir
   - Öğrenme hızını adaptif şekilde ayarlar
   - Daha hızlı ve stabil öğrenme sağlar
3. RMSprop, Adagrad, vb.
   - Belirli durumlarda avantaj sağlar

Keras ile Optimizasyon Seçimi

```
model.compile(
    optimizer="adam",
    loss="categorical_crossentropy",
    metrics=["accuracy"]
)
```

- `optimizer="adam"` → Öğrenme sürecini yöneten algoritma
- `loss="categorical_crossentropy"` → Hedefi sınıflandırma olan problemler için uygun
- `metrics=["accuracy"]` → Eğitim sırasında başarı metriği

Veri Ön İşleme (Data Preprocessing) Nedir?

- Makine öğrenmesi modelleri ham veriyi doğrudan işleyemez.
- Bu yüzden veriler modele uygun şekilde temizlenip dönüştürülmelidir.
- Bu adıma veri ön işleme (preprocessing) denir.

Veri Ön İşlemede Yapılan Temel Adımlar

- Eksik veri temizleme (NaN doldurma/silme)
- Etiketleme: Kategorik verileri sayısallaştırma (Label Encoding / One-Hot)
- Özellik ölçekleme: Tüm sayısal verileri ortak ölçeğe getirme (örnek: [0, 1])
- Normalizasyon / Standartlaştırma
- Veriyi eğitim/teste ayırma

Ön işleme yapılmadan model eğitmek genelde başarısız olur.

Keras için Veri Ön İşleme Örneği

```
1  from sklearn.datasets import load_iris
2  from sklearn.model_selection import train_test_split
3  from sklearn.preprocessing import StandardScaler
4  from tensorflow.keras.utils import to_categorical
5
6  X, y = load_iris(return_X_y=True)
7  y = to_categorical(y)
8
9  X_train, X_test, y_train, y_test = train_test_split(X, y,
       test_size=0.2)
10
11 scaler = StandardScaler()
12 X_train = scaler.fit_transform(X_train)
13 X_test = scaler.transform(X_test)
```

Regularization (Düzenleme) Nedir?

- Sinir ağları çok güçlü modellerdir → Kolayca ezberleyebilirler (overfitting).
- Regularization, modelin sadece eğitimi değil genelleme kabiliyetini artırır.
- Amaç: Modelin karmaşıklığını kontrol altında tutmak.

Yaygın Regularization Yöntemleri

1. L1 ve L2 Regularization:
   - Ağırlıklara ceza (penalty) uygulanır.
   - L1: Ağırlıkların toplam mutlak değeri
   - L2: Ağırlıkların karelerinin toplamı (ridge)
2. Dropout:
   - Eğitim sırasında rastgele bazı nöronlar devre dışı bırakılır.
   - Ağa çeşitlilik kazandırır, ezberlemeyi engeller.

Keras ile Dropout Kullanımı

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout,
    InputLayer

model = Sequential([
    InputLayer(input_shape=(4, )),
    Dense(64, activation="relu"),
    Dropout(0.5),
    Dense(3, activation="softmax")
])
```

- `Dropout(0.5)` → Eğitim sırasında %50 nöron geçici olarak kapatılır
- Overfitting riskini azaltır

L2 Regularization (Keras)

```python
from tensorflow.keras.regularizers import l2
from tensorflow.keras.layers import Dense

Dense(64, activation="relu", kernel_regularizer=l2(0.01))
```

- `kernel_regularizer` ile ağırlıklara ceza uygulanır
- `l2(0.01)` → Her ağırlık için küçük bir ceza eklenecek
- Model çok büyük ağırlıklar öğrenemez → Aşırı karmaşıklık önlenir

İleri Yayılım (Forward Pass)

- Girdi verisi, katmanlar boyunca sırayla işlenir.
- Her katmanda:
  - Ağırlıklı toplam + bias hesaplanır.
  - Aktivasyon fonksiyonu uygulanır.
- En son çıkış katmanı $\rightarrow$ tahmin sonucunu üretir.

Amaç: Veriye karşılık gelen tahmini ($\hat{y}$) hesaplamak.

Örnek: $y = \text{softmax}(W_2 \cdot \text{ReLU}(W_1 \cdot x + b_1) + b_2)$

Kayıp Fonksiyonu Nedir?

- Modelin tahmininin doğru cevaptan ne kadar sapma gösterdiğini ölçer.
- Geriye yayılım (backpropagation) için temel bilgi sağlar.
- Amaç: kayıp değerini minimuma indirmek.

Yaygın Fonksiyonlar:

- `Mean Squared Error (MSE)` → Regresyon problemleri
- `Categorical Crossentropy` → Çok sınıflı sınıflandırma
- `Binary Crossentropy` → İkili sınıflandırma

Not: Kayıp küçükse → model iyi öğrenmiş demektir.

Mean Squared Error (MSE)

- Regresyon problemlerinde kullanılır.
- Gerçek ve tahmin değerleri arasındaki farkın karesi alınır.
- Küçük farklar az, büyük farklar daha çok cezalandırılır.

Formül: $\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$

Örnek:
Gerçek değerler: `[3, 5, 2]`
Tahminler: `[2.5, 5.5, 1.5]`
$\text{MSE} = \frac{(3-2.5)^2+(5-5.5)^2+(2-1.5)^2}{3} = 0.25$

Not: MSE değeri 0'a yaklaştıkça model daha başarılıdır.

Binary Crossentropy

- İkili sınıflandırma problemlerinde (0/1) kullanılır.
- Modelin 1'e olan güveni ve 0'a olan güveni ayrı ayrı cezalandırılır.

Formül: $\text{Loss} = -[y \cdot \log(\hat{y}) + (1 - y) \cdot \log(1 - \hat{y})]$

Örnek:
Gerçek: `1`, Tahmin: `0.9` $\rightarrow$ $\text{Loss} = -\log(0.9) = 0.105$ Gerçek: `1`, Tahmin: `0.1` $\rightarrow$ $\text{Loss} = -\log(0.1) = 2.30$

Categorical Crossentropy

- Çok sınıflı sınıflandırma problemlerinde kullanılır.
- Her sınıf için olasılık tahmini yapılır (softmax).
- Gerçek sınıfın olasılığı ne kadar düşükse, kayıp o kadar büyür.

Örnek:
Gerçek sınıf: `[0, 0, 1]` Tahmin: `[0.1, 0.2, 0.7]`
Kayıp $= -\log(0.7) \approx 0.357$

Model yanlış sınıfa yüksek olasılık verirse yüksek ceza alır.

Model Eğitimi (fit)

- Model, eğitilen veri ile öğrenir.
- `fit()` fonksiyonu:
  - Girdi verisini ve etiketleri alır.
  - Her örnek üzerinden ağırlıkları günceller.
  - Birden fazla kez tekrar eder (epoch sayısı kadar).
- Eğitim sırasında kayıp değeri azalırsa model doğru öğreniyor demektir.

Örnek kullanım:

```
1 model.fit(X_train, y_train, epochs=10, batch_size=32)
```

Hiperparametreler Nedir?

- Modelin dışında ayarlanan, öğrenme sürecini doğrudan etkileyen değerlerdir.
- Öğrenilen değil, önceden belirlenen parametrelerdir.
- Doğru ayarlamak: daha iyi genelleme ve daha az hata demektir.

Temel Hiperparametreler:

- `epoch` – Eğitim tekrar sayısı
- `batch size` – Aynı anda işlenecek örnek sayısı
- `learning rate` – Ağırlıkların ne kadar değişeceği

Hiperparametrelerin Etkisi

- Epoch sayısı
  - Az olursa: Öğrenme eksik kalır (underfitting)
  - Fazla olursa: Aşırı öğrenme (overfitting) riski
- Batch size
  - Küçük: Gürültülü ama sık güncelleme → iyi genelleme
  - Büyük: Daha istikrarlı ama öğrenme yavaş
- Learning rate
  - Çok küçük: Yavaş öğrenme
  - Çok büyük: Öğrenme kararsızlaşabilir

Epoch – Eğitim Tekrar Sayısı

- Epoch: Tüm eğitim verisinin modele bir kez gösterilmesidir.
- 10 epoch: eğitim verisi modele 10 defa tekrar gösterilir.

Örnek:
Bir model 1000 veriyle eğitiliyorsa:

- 1 epoch $\rightarrow$ model 1000 veri görür
- 10 epoch $\rightarrow$ model toplamda 10.000 veri görür

Epoch sayısı artarsa: Öğrenme artar ama aşırı ezber riski doğar (overfitting)

Batch Size – Mini Grup Eğitimi

- Veriler modele parça parça sunulur.
- Batch size: her iterasyonda işlenecek veri sayısıdır.

Örnek:
1000 verilik bir veri setinde:

- Batch size = 100 → her epoch 10 adım (1000 / 100)
- Batch size = 250 → her epoch 4 adım

Küçük batch: sık güncelleme, genelleme iyi
Büyük batch: daha stabil ama öğrenme yavaş

Learning Rate – Öğrenme Hızı

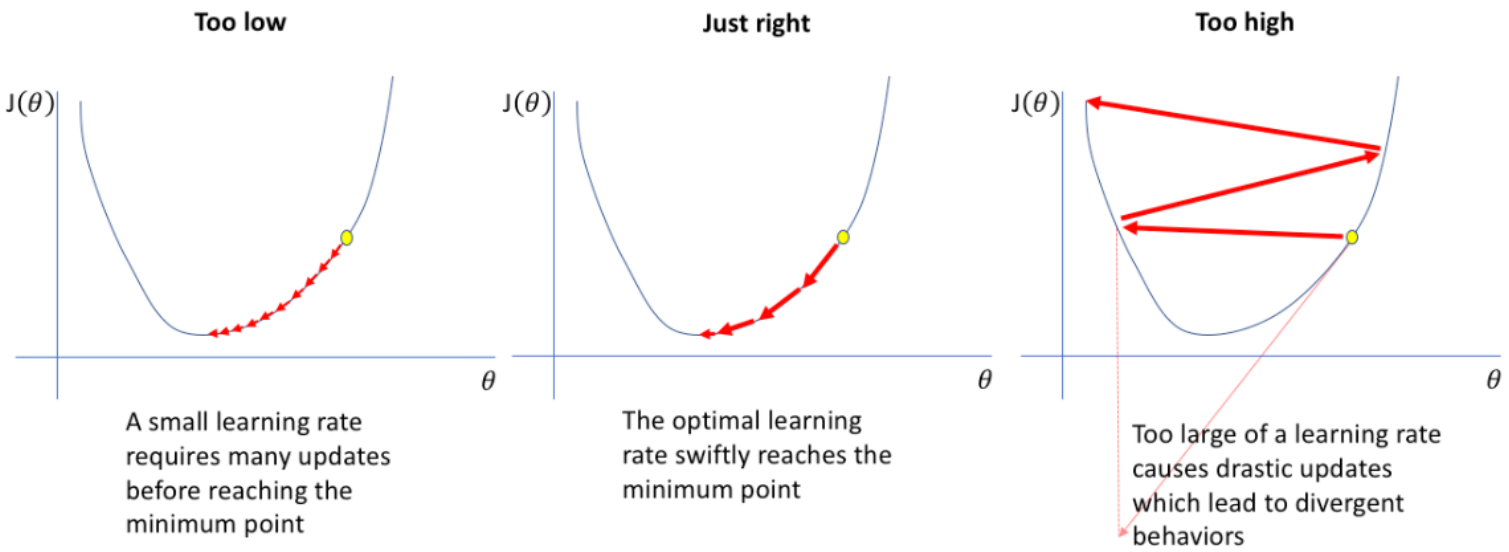
- Her adımda ağırlıklar ne kadar değişecek?
- Learning rate (lr) çok küçükse: öğrenme çok yavaş
- Learning rate çok büyükse: öğrenme dengesizleşebilir

Örnek:

- lr $= 0.001 \rightarrow$ küçük adımlarla ilerler
- lr $= 0.1 \rightarrow$ büyük adımlarla sıçrayarak öğrenir (riskli olabilir)

Çözüm: deneme yanılma ya da optimizasyon teknikleri

Learning Rate

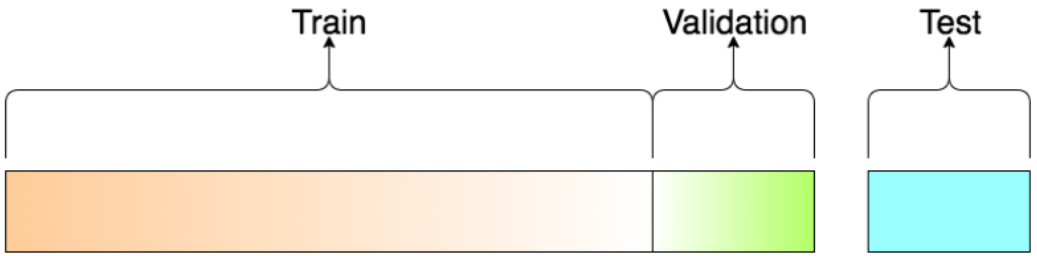


Eğitim ve Doğrulama Ayrımı

- Model, eğitim verisi ile öğrenir.
- Ancak yalnızca eğitim verisiyle test edilirse:
  - Gerçek başarı ölçülemez.
  - Aşırı öğrenme (overfitting) fark edilmez.
- Bu nedenle:
  - Eğitim verisi: Öğrenme için
  - Doğrulama verisi (validation): Öğrenme sırasında değerlendirme için
  - Test verisi: Son değerlendirme için

Amaç: Modelin genelleme başarısını ölçmek

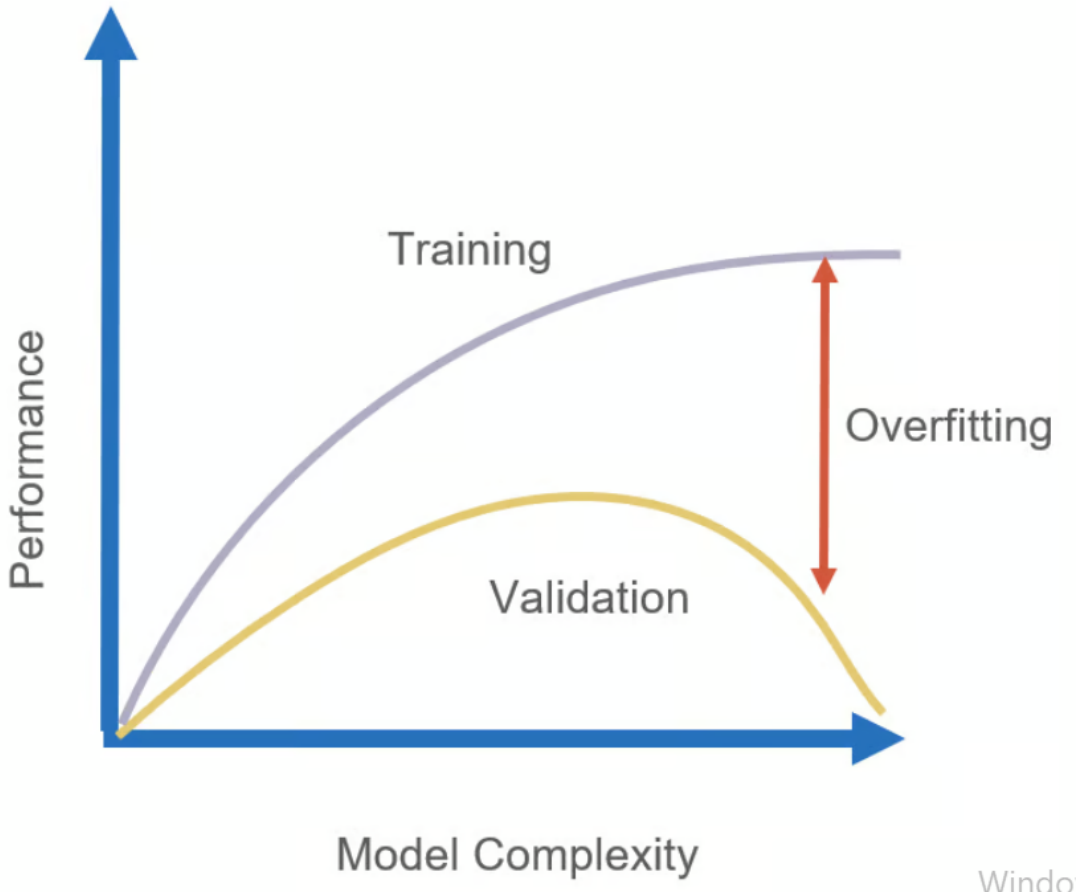
Train-Validation



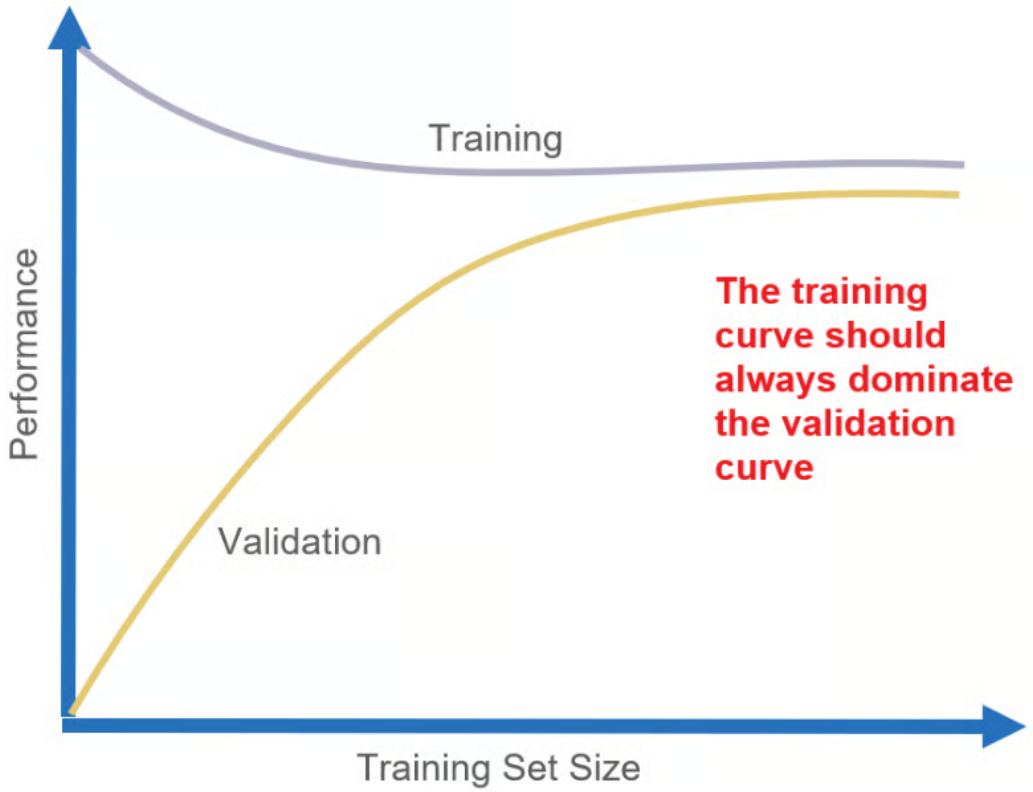
Model Eğitimi Sırasında İzleme

- `fit()` sırasında model kayıp (loss) ve doğruluk (accuracy) değerlerini takip eder.
- Bu değerler her epoch sonunda güncellenir.
- Görselleştirme ile:
  - Öğrenme eğrisi takip edilir
  - Overfitting tespit edilir

Ezber


Training
Overfitting
Validation
Performance
Model Complexity

Ezber


Training
Performance
The training curve should always dominate the validation curve
Validation
Training Set Size

Backpropagation (Geri Yayılım)

- Sinir ağı çıktısıyla gerçek değer karşılaştırılır → hata hesaplanır.
- Bu hata, ağın gerisine doğru yayılır.
- Katmanlardaki ağırlıklar, hataya göre güncellenir.
- Amaç: ağırlıkları öyle ayarla ki kayıp fonksiyonu minimum olsun.

"Yanlışı fark et → nerede yaptığını öğren → kendini düzelt."

Geri Yayılım Nasıl İşler?

- 1. Forward Pass: Girdi → çıkış üretilir
- 2. Loss Hesabı: Çıkış ile gerçek değer farkı
- 3. Backward Pass: Hata, ağırlıklara doğru yayılır
- 4. Gradient Descent: Ağırlıklar güncellenir

- Sinir ağı öğrenirken amaç: kayıp (loss) fonksiyonunu en aza indirmek
- Geri yayılım ile hesaplanan gradyanlar kullanılarak ağırlıklar güncellenir
- Bu güncellemeyi yapan algoritmaya optimizer denir

Popüler Optimizer'lar:

- SGD (Stochastic Gradient Descent)
- Adam
- RMSProp

Doğru optimizer, hızlı ve dengeli öğrenme sağlar.

SGD, Adam ve RMSProp Karşılaştırması

Özellik	SGD	Adam	RMSProp
Hız	Yavaş	Hızlı	Orta
Bellek Kullanımı	Az	Orta	Orta
Uyarlamalı Adım	Hayır	Evet	Evet
Kararlılık	Düşük	Yüksek	Orta

Not: Adam genellikle en dengeli sonuçları verir.

Basit Sinir Ağı Eğitimi (Keras)

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

model = Sequential([
  Dense(16, activation="relu", input_shape=(X_train.shape
    [1],)),
  Dense(8, activation="relu"),
  Dense(1, activation="sigmoid")
])

model.compile(optimizer="adam",
              loss="binary_crossentropy",
              metrics=["accuracy"])
history = model.fit(X_train, y_train,
                    epochs=20, batch_size=32,
              validation_split=0.2)
```

Modelin Test Verisi ile Değerlendirilmesi

- Eğitimden sonra modelin gerçek başarısı test verisiyle ölçülür.
- `evaluate()` fonksiyonu doğruluk ve kayıp değerlerini verir.

Kod Örneği:

```
loss, acc = model.evaluate(X_test, y_test)
print(f"Test Dogruluk: {acc:.2f}")
```

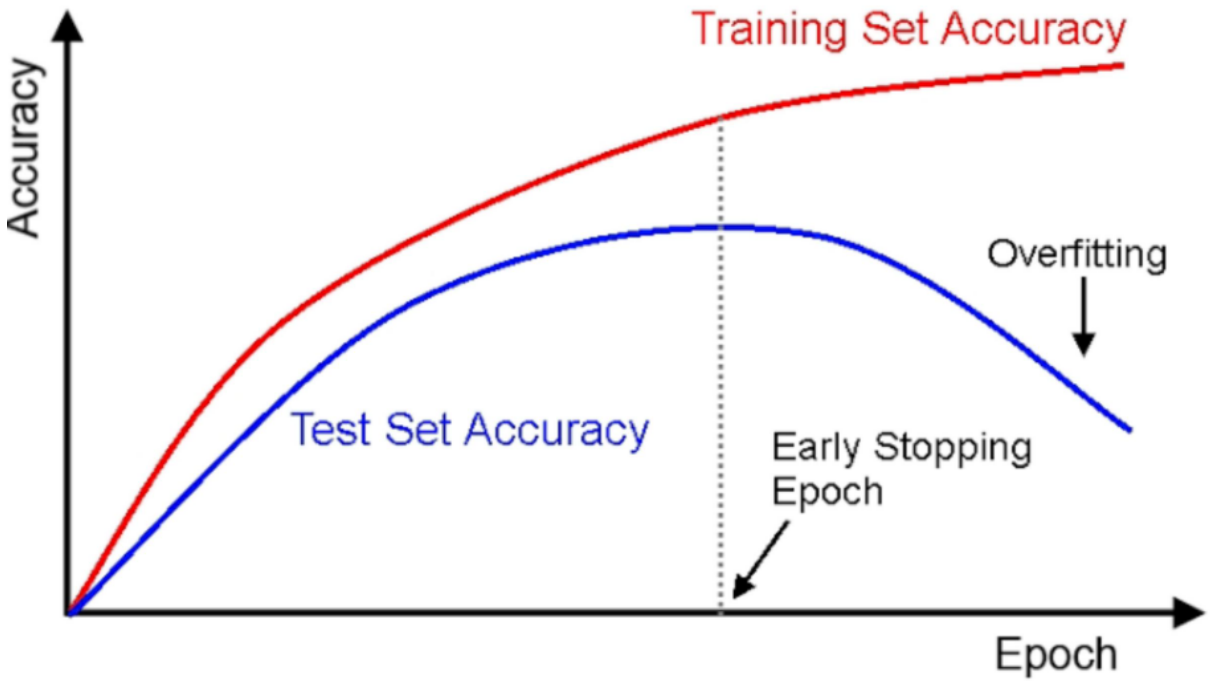
Test verisinde yüksek başarı, modelin genelleme gücünü gösterir.

Erken Durdurma (Early Stopping)

- Eğitim sırasında doğrulama kaybı iyileşmeyi durdurursa:
  - Model aşırı öğrenmeye başlar
  - Artık öğrenme gerçekleşmez
- Erken durdurma, bu durumda eğitimi otomatik sonlandırır
- Eğitim süresi kısalır, model genellemesi artar

Kullanım: `EarlyStopping(monitor="val_loss", patience = 3)`

Erken Durdurma



EarlyStopping Kullanımı

```
from tensorflow.keras.callbacks import EarlyStopping

early_stop = EarlyStopping(
    monitor="val_loss", patience=3, restore_best_weights=
    True)

model.fit(X_train, y_train,
          epochs=50, batch_size=32,
          validation_split=0.2,
          callbacks=[early_stop])
```

Açıklama: 3 epoch boyunca iyileşme yoksa eğitim durur.

Epoch ve Batch Size Deneyleri

```
model.fit(X_train, y_train,
          epochs=10, batch_size=32,
          validation_split=0.2)

model.fit(X_train, y_train,
          epochs=50, batch_size=8,
          validation_split=0.2)
```

Gözlem:

- Çok düşük batch size → daha yavaş ama hassas öğrenme
- Fazla epoch → aşırı öğrenmeye neden olabilir

Loss / Accuracy Grafikleri

```
history = model.fit(...)

plt.plot(history.history["loss"], label="Egitim Kaybi")
plt.plot(history.history["val_loss"], label="Dogrulama
    Kaybi")
plt.legend()
plt.title("Kayip Grafigi")
plt.show()
```

Eğitim ve doğrulama eğrileri birlikte analiz edilmelidir.

Veri Yükleme ve Ön İşleme

```python
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.utils import to_categorical

X, y = load_iris(return_X_y=True)
y = to_categorical(y)

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Model Tanımı

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout,
    InputLayer
from tensorflow.keras.regularizers import l2

model = Sequential([
    InputLayer(input_shape=(4, )),
    Dense(64, activation="relu", kernel_regularizer=l2
    (0.01)),
    Dropout(0.5),
    Dense(3, activation="softmax")
])
```

- 4 giriş → 64 ReLU + Dropout → 3 sınıf için softmax çıkışı
- L2 ve Dropout ile overfitting riski azaltılır

Model Derleme (Compile)

```
model.compile(
    optimizer="adam",
    loss="categorical_crossentropy",
    metrics=["accuracy"]
)
```

- `adam`: Adaptif öğrenme oranı ile hızlı optimizasyon
- `categorical_crossentropy`: Çok sınıflı sınıflandırma için ideal
- `metrics=["accuracy"]`: Başarı ölçütü olarak doğruluk kullanılır

Model Eğitimi (Fit)

```
history = model.fit(
    X_train, y_train,
    validation_split=0.2,
    epochs=50,
    batch_size=8,
    verbose=1
)
```

- Eğitim verisinin %20'si doğrulama (validation) için ayrılır
- 50 epoch boyunca küçük gruplar (batch_size=8) ile öğrenir

Modeli Değerlendirme ve Tahmin

```python
test_loss, test_acc = model.evaluate(X_test, y_test)
print(f"Test dogrulugu: {test_acc:.2f}")

sample = X_test[0].reshape(1, -1)
prediction = model.predict(sample)
print("Tahmin:", prediction)
```

- `evaluate()` → modelin test setindeki başarısını ölçer
- `predict()` → sınıflara ait olasılıkları verir

Fashion MNIST Veriseti Tanıtımı

- Fashion MNIST, klasik MNIST rakam veri setinin modern bir versiyonudur.
- Toplam 70.000 adet gri tonlamalı 28x28 piksel giysi görseli içerir.
- 10 farklı sınıfa ayrılmıştır (her sınıf 7.000 örnek):
  - 0 = T-shirt/top
  - 1 = Trouser
  - 2 = Pullover
  - 3 = Dress
  - 4 = Coat
  - 5 = Sandal
  - 6 = Shirt
  - 7 = Sneaker
  - 8 = Bag
  - 9 = Ankle boot
- Eğitim: 60.000 örnek, Test: 10.000 örnek

1. Fashion MNIST Veri Setini Yükleme

```python
from tensorflow.keras.datasets import fashion_mnist
from tensorflow.keras.utils import to_categorical
import matplotlib.pyplot as plt

(X_train, y_train), (X_test, y_test) = fashion_mnist.
    load_data()

print(X_train.shape, y_train.shape)

plt.imshow(X_train[0], cmap="gray")
plt.title(f"Label: {y_train[0]}")
plt.show()
```

- 60.000 eğitim ve 10.000 test örneği içerir
- Görseller 28x28 piksel, etiketler 0–9 arası sayılar

```
X_train = X_train / 255.0
X_test = X_test / 255.0

X_train = X_train.reshape(-1, 784)
X_test = X_test.reshape(-1, 784)

y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)
```

- Görsel veriyi normalize etmek öğrenmeyi kolaylaştırır
- One-hot encoding → 10 sınıf için uygun çıkış vektörü

3. MLP Modeli Oluşturma

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout

model = Sequential([
    Dense(256, activation="relu", input_shape=(784,)),
    Dropout(0.4),
    Dense(128, activation="relu"),
    Dropout(0.3),
    Dense(10, activation="softmax")
])
```

- Giriş: 784 nöron (düzleştirilmiş görsel)
- 2 gizli katman + Dropout → ezberlemeyi önleme
- Çıkış: 10 sınıf için softmax

4. Modeli Derleme ve Eğitme

```
model.compile(
    optimizer="adam",
    loss="categorical_crossentropy",
    metrics=["accuracy"]
)

history = model.fit(
    X_train, y_train,
    epochs=15,
    batch_size=128,
    validation_split=0.1,
    verbose=1
)
```

- `validation_split=0.1` → eğitimden %10'u doğrulama için

```
1 test_loss, test_acc = model.evaluate(X_test, y_test)
2 print(f"Test dogrulugu: {test_acc:.4f}")
3
4 import numpy as np
5 idx = np.random.randint(0, len(X_test))
6 prediction = model.predict(X_test[idx].reshape(1, 784))
7 print("Tahmin:", prediction.argmax())
```

- Test doğruluğu genellikle
- `argmax()` → en yüksek olasılığı veren sınıf seçilir

6. Eğitim ve Doğrulama Doğruluk Grafiği

```
1  import matplotlib.pyplot as plt
2
3  plt.plot(history.history["accuracy"], label="train")
4  plt.plot(history.history["val_accuracy"], label="val")
5  plt.xlabel("Epoch")
6  plt.ylabel("Dogruluk")
7  plt.title("Egitim vs Dogrulama şııBaarm")
8  plt.legend()
9  plt.grid(True)
10 plt.show()
```

- Modelin aşırı öğrenip öğrenmediği bu grafikle anlaşılır
- Eğitimin sonunda doğrulama başarımı duruyorsa → erken durdurma düşünülebilir

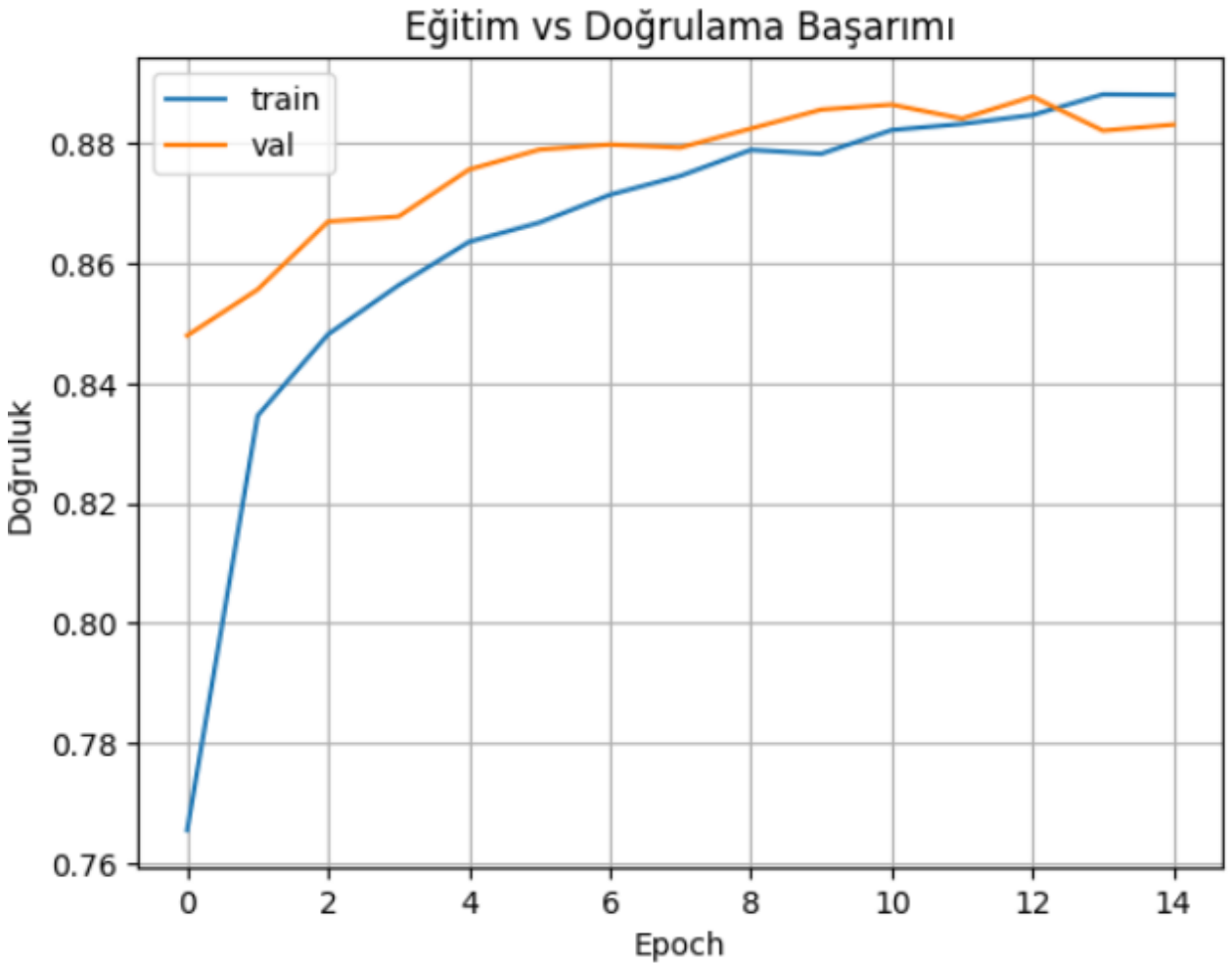


Figure: Caption