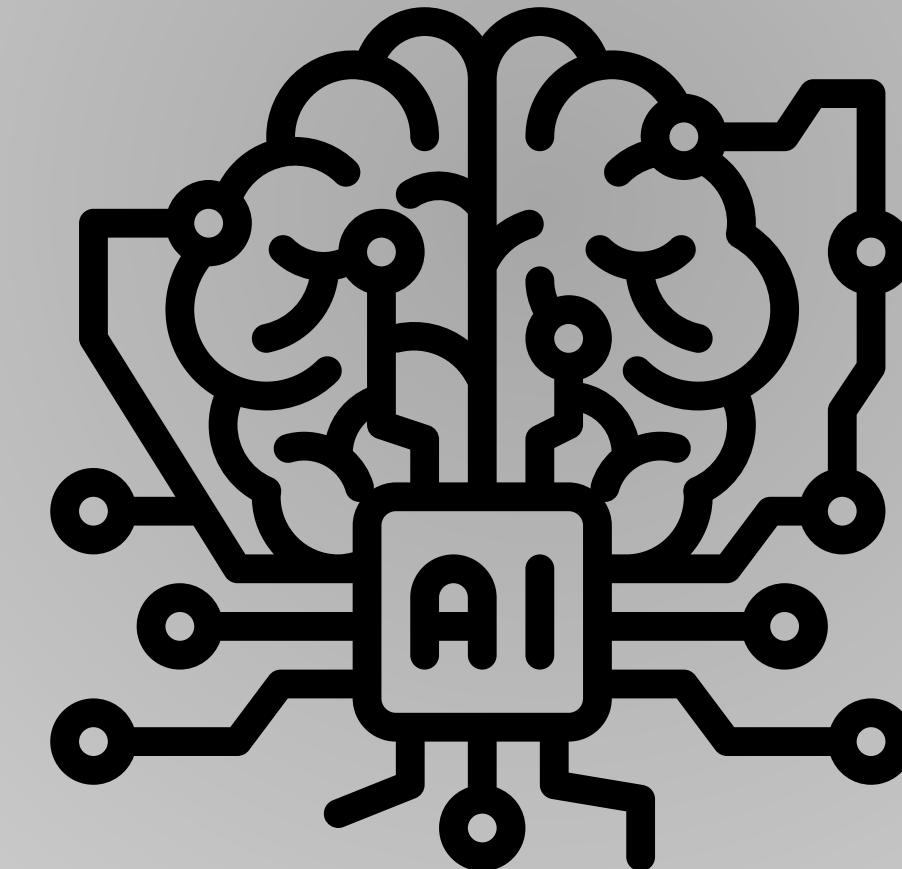




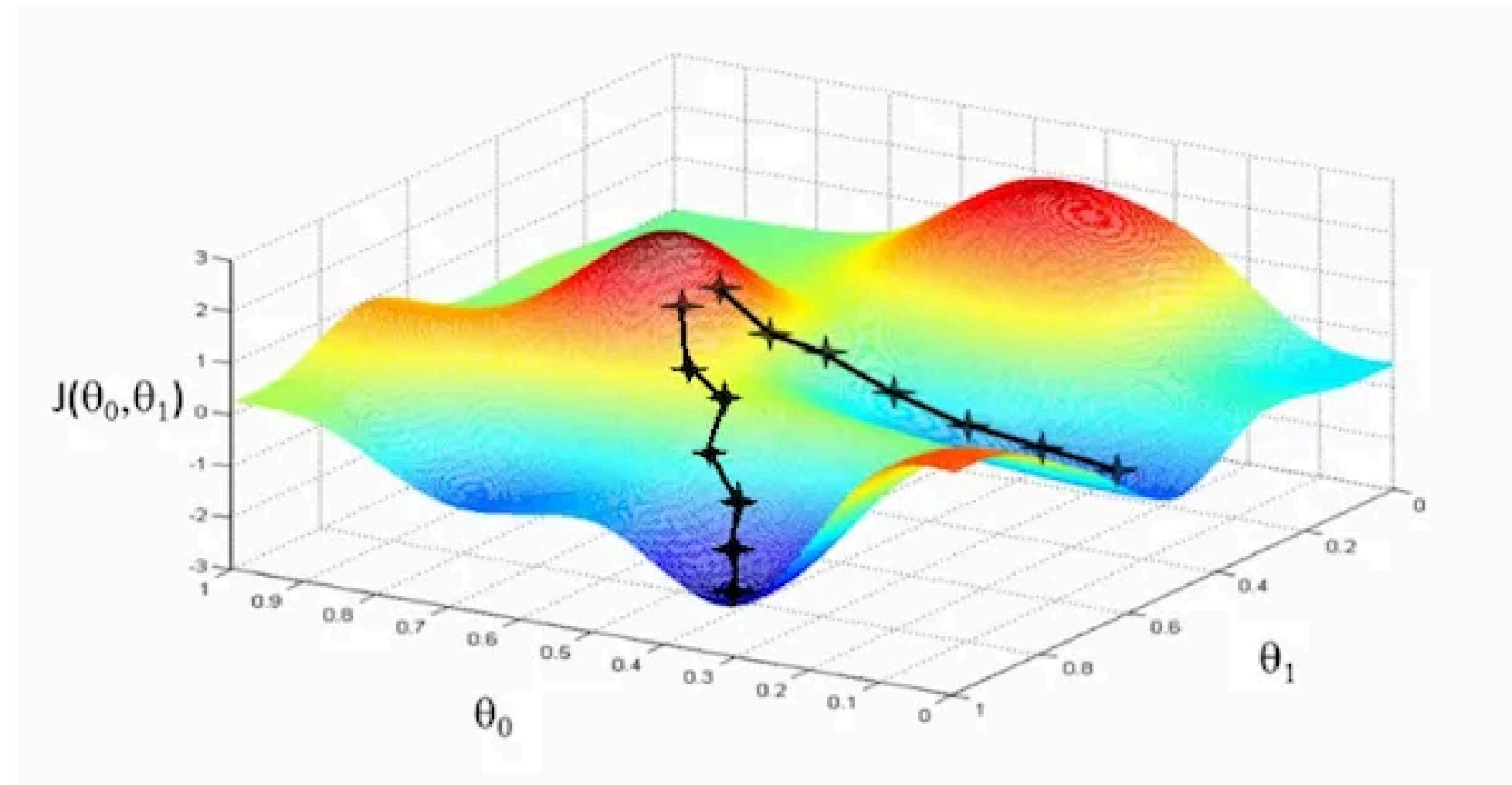
5.HAFTA

# YAPAY SINIR AĞLARI



SAMSUN ÜNİVERSİTESİ

DR. ÖĞR. ÜYESİ ALPER TALHA KARADENİZ



## Kayıp Fonksiyonları (Loss Functions)

Kayıp Fonksiyonları (Loss Functions), derin öğrenme modellerinin eğitimi sırasında modelin tahminlerinin gerçek değerlerden ne kadar uzak olduğunu hesaplayan matematiksel fonksiyonlardır. Bu fonksiyonlar, modelin performansını değerlendirmek için kullanılır ve eğitim sürecindeki ağırlık güncellemelerini yönlendirmeye yardımcı olurlar.

## Ortalama Hata Karesi (Mean Squared Error)

Ortağın Hata (Mean Squared Error) Kaybı, en yaygın kullanılan kayıp fonksiyonlarından biridir ve özellikle regresyon problemlerinde tercih edilir. MSE, modelin tahminlerini gerçek değerlerle karşılaştırır ve her örneğin hata karelerinin ortalamasını hesaplar.

$$MSE = \frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_i)^2$$

Burada;

- N, toplam veri noktalarının sayısını temsil eder.
- $y_i$ , gerçek değeri ifade eder.
- $\hat{y}_i$ , modelin tahminini ifade eder.

## Örnek kod:

```
1 import tensorflow as tf
2
3 # Gerçek değerler
4 y_true = [3, -0.5, 2, 7]
5
6 # Modelin tahminleri
7 y_pred = [2.5, 0.0, 2, 8]
8
9 # Ortalama Hata Karesi (Mean Squared Error) hesaplama
10 mse = tf.keras.losses.MeanSquaredError()
11 loss = mse(y_true, y_pred)
12 print("Mean Squared Error:", loss.numpy())
```

## Kategorik Çapraz Entropi (Categorical Crossentropy)

Kategorik Çapraz Entropi (Categorical Crossentropy) Kaybı, sınıflandırma problemlerinde yaygın olarak kullanılır ve sınıflar arasındaki farkı ölçmek için kullanılır. Bu kayıp fonksiyonu, gerçek etiketler ve modelin çıktıları arasındaki farkları hesaplar.

$$\text{CategoricalCrossentropy} = - \sum_{i=1}^C y_i \log(\hat{y}_i)$$

C: Sınıf sayısı

$y_i$ : Gerçek etiketin i. sınıf'a ait olma olasılığı

$\hat{y}_i$ : Modelin tahmini, i. sınıf'a ait olma olasılığı

## Örnek kod:

```
1 import tensorflow as tf
2
3 # Gerçek etiketler (one-hot kodlaması)
4 y_true = [[0, 1, 0], [1, 0, 0], [0, 0, 1]]
5
6 # Modelin tahminleri (olasılıklar)
7 y_pred = [[0.2, 0.7, 0.1], [0.9, 0.1, 0.0], [0.1, 0.3, 0.6]]
8
9 # Kategorik Çapraz Entropi (Categorical Crossentropy) hesaplama
10 cce = tf.keras.losses.CategoricalCrossentropy()
11 loss = cce(y_true, y_pred)
12 print("Categorical Crossentropy:", loss.numpy())
```

## İkili Çapraz Entropi (Binary Crossentropy)

İkili Çapraz Entropi (Binary Crossentropy) Kaybı, iki sınıfı sınıflandırma problemlerinde kullanılır. Bu kayıp fonksiyonu, gerçek etiketler ve modelin çıktıları arasındaki farkı ölçer ve her bir sınıfın bağımsız olarak ele alındığı durumlarda kullanılır.

Binary Crossentropy formülü:

$$\text{BinaryCrossentropy} = - \sum_{i=1}^C (y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i))$$

$y_i$ : Gerçek etiket (0 veya 1)

$\hat{y}_i$ : Modelin tahmini (olasılık)

## Örnek kod:

```
1 import tensorflow as tf
2
3 # Gerçek etiketler (0 veya 1)
4 y_true = [1, 0, 1, 1]
5
6 # Modelin tahminleri (olasılık)
7 y_pred = [0.9, 0.2, 0.8, 0.95]
8
9 # İkili Çapraz Entropi (Binary Crossentropy) hesaplama
10 bce = tf.keras.losses.BinaryCrossentropy()
11 loss = bce(y_true, y_pred)
12 print("Binary Crossentropy:", loss.numpy())
```

---

## Seyrek Kategorik Çapraz Entropi (Sparse Categorical Crossentropy)

Seyrek Kategorik Çapraz Entropi (Sparse Categorical Crossentropy) Kaybı, sınıflandırma problemlerinde yaygın olarak kullanılır ve sınıflar arasındaki farkı ölçmek için kullanılır. Ancak, farklı olarak, gerçek etiketler one-hot kodlaması yerine tamsayılar olarak temsil edilir.

Sparse Categorical Crossentropy formülü:

$$\text{SparseCategoricalCrossentropy} = - \sum_{i=1}^C \log(\hat{y}_i[y_i])$$

C: Sınıf sayısı

$y_i$ : Gerçek etiketin tamsayı değeri (0'dan C'ye kadar)

$\hat{y}_i$ : Modelin tahmini, i. sınıfa ait olma olasılığı

```
1 import tensorflow as tf
2
3 # Gerçek etiketler (tamsayı değerleri)
4 y_true = [1, 0, 2]
5
6 # Modelin tahminleri (olasılıklar)
7 y_pred = [[0.2, 0.7, 0.1], [0.9, 0.1, 0.0], [0.1, 0.3, 0.6]]
8
9 # Seyrek Kategorik Çapraz Entropi (Sparse Categorical Crossentropy) hesaplama
10 sparse_cce = tf.keras.losses.SparseCategoricalCrossentropy()
11 loss = sparse_cce(y_true, y_pred)
12 print("Sparse Categorical Crossentropy:", loss.numpy())
```

# Optimizasyon Algoritmaları

Optimizasyon algoritmaları, modelin kayıp fonksiyonunu minimize ederek modelin daha iyi performans göstermesini sağlar. Bu algoritmalar, ağırlıkları güncellemek için gradyan iniş (gradient descent) ve türev tabanlı teknikler kullanır.

## Gradyan Iniş (Gradient Descent)

Gradyan Iniş (Gradient Descent), belki de en temel optimizasyon algoritmasıdır ve derin öğrenme modellerinin eğitimiinde sıkça kullanılır. Amacı, kayıp fonksiyonunun ağırlıklar üzerinden türevini alarak, negatif gradyana doğru ağırlıkları güncellemektir.

```
1 import tensorflow as tf
2
3 # Eğitim verileri ve etiketleri (örnek)
4 train_data = [...] # Eğitim verileri
5 train_labels = [...] # Eğitim etiketleri
6
7 # Model oluşturma (örnek)
8 model = tf.keras.Sequential([...])
9
10 # Optimizasyon algoritması seçimi (Gradyan İniş)
11 optimizer = tf.keras.optimizers.SGD(learning_rate=0.01)
12
13 # Modeli derleme
14 model.compile(optimizer=optimizer, loss='mse', metrics=['accuracy'])
15
16 # Modeli eğitme
17 model.fit(train_data, train_labels, epochs=10, batch_size=32)
```

---

## Adam Optimizasyonu (Adam Optimization)

Adam Optimizasyonu, gradyan iniş algoritmasının bir türevidir ve daha hızlı ve verimli bir şekilde çalışır. Bu algoritma, adaptif momentum ve adaptif öğrenme hızı kullanarak ağırlıkları günceller.

```
1 import tensorflow as tf
2
3 # Eğitim verileri ve etiketleri (örnek)
4 train_data = [...] # Eğitim verileri
5 train_labels = [...] # Eğitim etiketleri
6
7 # Model oluşturma (örnek)
8 model = tf.keras.Sequential([...])
9
10 # Optimizasyon algoritması seçimi (Adam Optimizasyonu)
11 optimizer = tf.keras.optimizers.Adam(learning_rate=0.001)
12
13 # Modeli derleme
14 model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['accuracy'])
15
16 # Modeli eğitme
17 model.fit(train_data, train_labels, epochs=10, batch_size=32)
```

# Eğitim, Doğrulama ve Test Veri Setleri

Yapay sinir ağları ile çalışırken veriler üç ayrı gruba ayrılır: eğitim, doğrulama ve test veri setleri. Her birinin amacı farklıdır.

---

## 1. Eğitim (Training) Veri Seti

- **Tanım:** Modelin öğrenme işlemini gerçekleştirdiği veri setidir.
- **Görev:**
  - Girdi (input) ve çıktı (output) arasındaki ilişkileri öğrenmek.
  - Ağırlıklar ve bias değerleri, eğitim verisi üzerinden hesaplanan hata yardımıyla güncellenir.
- **Özellikler:**
  - Model, bu veriler üzerinden defalarca (epoch sayısı kadar) geçer.
  - Öğrenme algoritmaları (ör. geri yayılım ve optimizasyon yöntemleri) bu set üzerinde çalışır.

## 2. Doğrulama (Validation) Veri Seti

- **Tanım:** Eğitim sırasında modelin öğrenme sürecini kontrol etmek için ayrılan veri setidir.
- **Görev:**
  - Modelin genel başarısını eğitim esnasında izlemek.
  - Hiperparametre seçiminde yol gösterici olmak (ör. öğrenme oranı, katman sayısı).
  - Aşırı öğrenme (overfitting) olup olmadığını belirlemek.
- **Özellikler:**
  - Doğrulama verisi ile yapılan değerlendirmelerde ağırlıklar güncellenmez.
  - Eğitim hataları ile doğrulama hataları karşılaştırılarak eğitim sürecine yön verilir.

### 3. Test (Testing) Veri Seti

- **Tanım:** Eğitim tamamlandıktan sonra modelin gerçek performansını ölçmek için kullanılan veri setidir.
- **Görev:**
  - Modelin hiç görmediği veriler üzerinde ne kadar başarılı olduğunu değerlendirmek.
  - Modelin genelleme yeteneğini ölçmek.
- **Özellikler:**
  - Test seti, eğitim veya doğrulama aşamalarında kullanılmaz.
  - Sadece nihai performansı ölçmek için ayrıılır.

## 4. Genel Kullanım

- Eğitim, doğrulama ve test setleri birbirinden ayrı olmalıdır.
- Tipik bölmeye oranları:
  - %70 → Eğitim
  - %15 → Doğrulama
  - %15 → Test



**Validation**



# **Overfitting (Aşırı Öğrenme), Underfitting (Eksik Öğrenme) ve Bias-Variance Çelişkisi**

**M**akine öğrenmesi uygulamalarında temel amaç eldeki veriler üzerinden örüntüler elde etmek ve yeni veriler için bu örüntüler üzerinden doğru tahminler yapabilmektir. Bu tahminleri yapabilmek için makine öğrenmesi uygulamaları sonucunda bir model elde ederiz. Peki model nedir ?

*Model girdilerin çıktılara eşlenmesi için kullanılan bir sistemdir. Örneğin amacımız ev fiyatlarını tahmin etmek olsun . Bunun için evin metrekare bilgisini girdi olarak kullanan bir model oluştururuz ve çıktı olarak da evin fiyatını elde ederiz.*

*Bir model bir problem hakkında teori üretir. Buradaki teori evin metrekare bilgisi ile fiyatı arasında bir ilişki olmasıdır. Eğitim veri çalışmaları sonucunda bu ilişkiyi öğrenmiş olan bir model oluşturmuş oluruz.*

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \cdots + \beta_n x^n + \varepsilon.$$

Burada  $y$  tahminlenen değer ve  $x$ 'ler ise modelin değişkenleridir. Beta terimleri ise model parametreleridir ve bu parametre değerleri eğitim seti üzerinde yapılan çalışmalar sonucunda öğrenilmektedir. epsilon ise modelin hata değeridir. Model Beta değerlerini öğrendikten sonra istediğimiz  $x$  değişkenlerini formüle koyarak  $y$  değerini bulabiliriz.

Modelimizdeki değişken sayısına bağlı olarak (degree) farklı grafikler elde ederiz.

## Types of Polynomial (Degree)

**Constant  
Polynomial  
(Degree 0)**

$$-\frac{2}{3}$$

**Linear  
Polynomial  
(Degree 1)**

$$\frac{3}{4}x - 6$$

**Quadratic  
Polynomial  
(Degree 2)**

$$3x^2 - 2x + 7$$

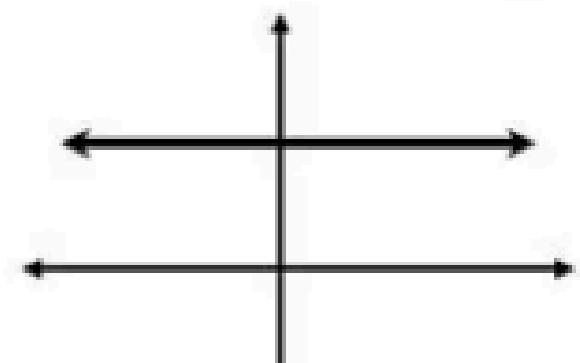
$$5y^2 - \frac{1}{4}$$

**Cubic  
Polynomial  
(Degree 3)**

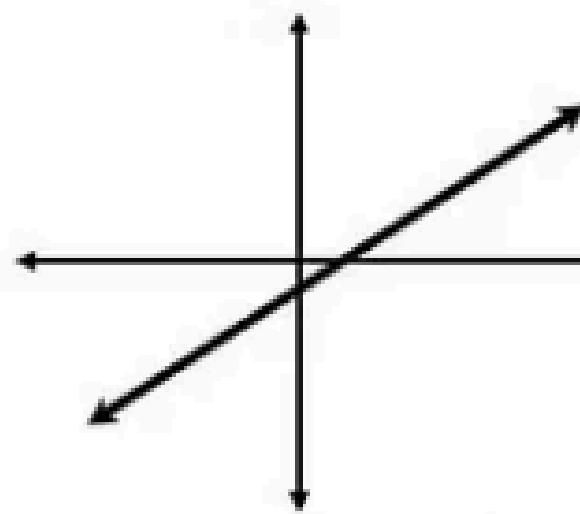
$$5x^3$$

$$2y^3 - y + 4$$

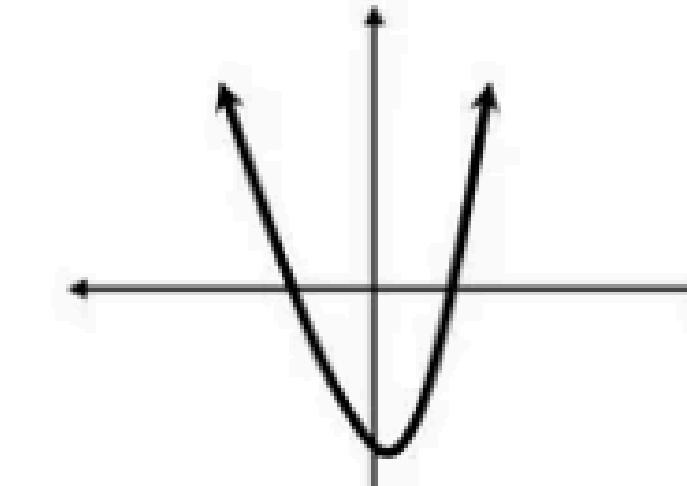
## Graphs of Polynomial Functions:



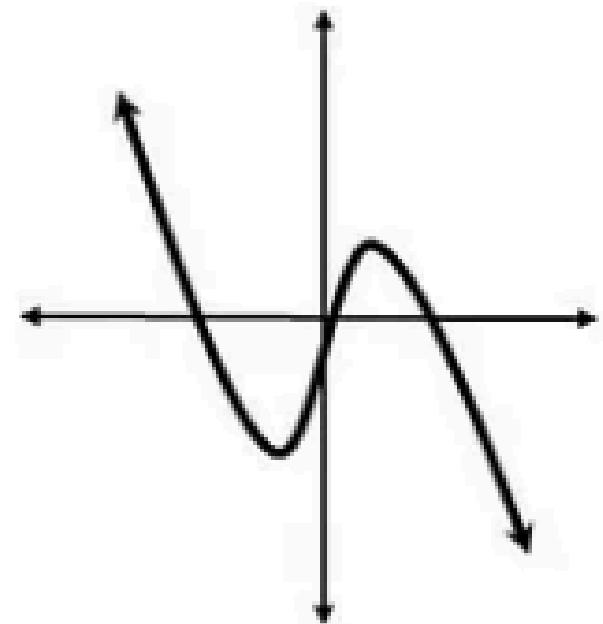
**Constant Function**  
(degree = 0)



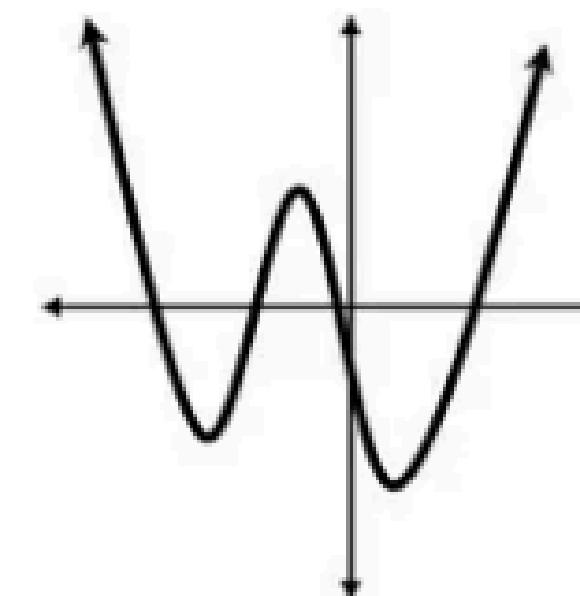
**Linear Function**  
(degree = 1)



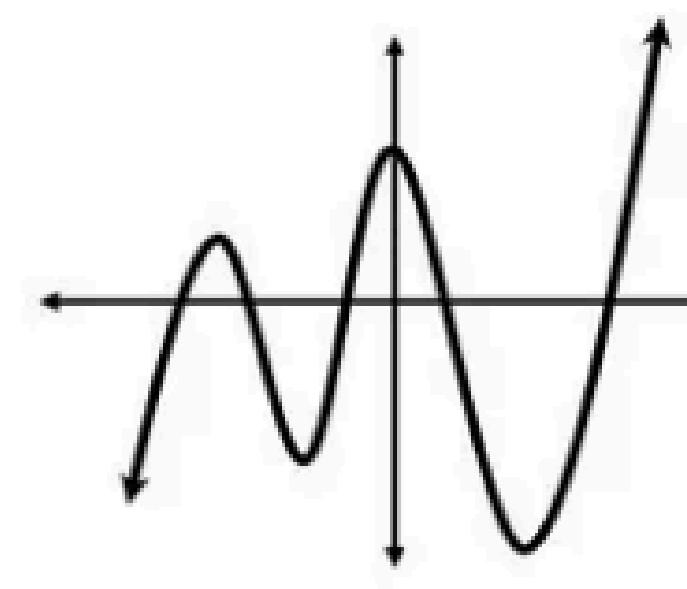
**Quadratic Function**  
(degree = 2)



**Cubic Function**  
(deg. = 3)

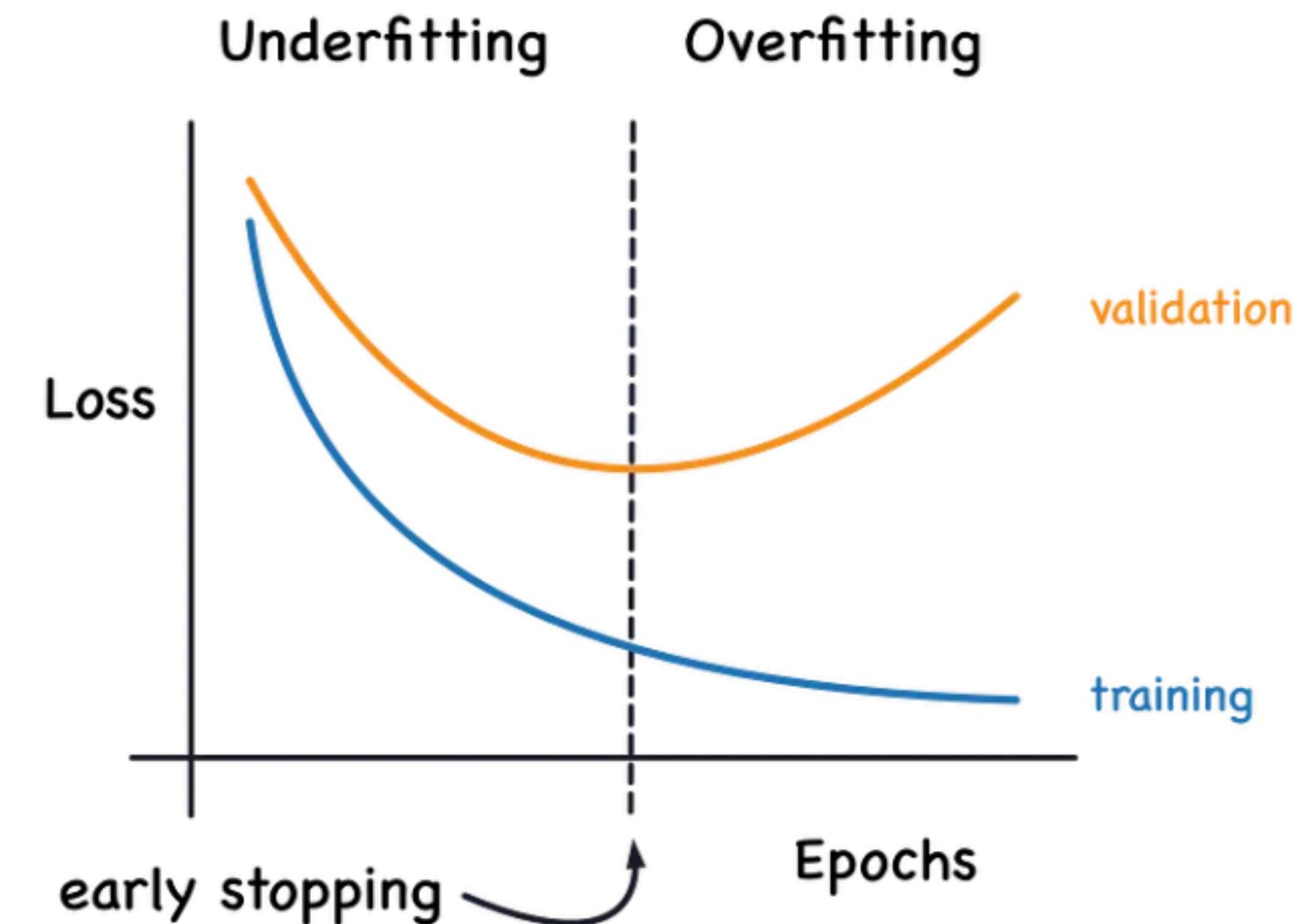


**Quartic Function**  
(deg. = 4)



**Quintic Function**  
(deg. = 5)

Makine öğreniminde gelecek veriler hakkında tahmin yapabilmek için verilerimizi eğitim verileri (train) ve test verileri olmak üzere iki alt kümeye ayırıyoruz. Modelimizi eğitim verilerinden elde edilen örüntülere göre oluşturuyoruz. Bu işlem sonucunda iki şeyden biri olabilir; modelimiz aşırı öğrenebilir veya eksik öğrenebilir. Bu durumda modelimiz yeterli öngörüde bulunamayacak ve tahminlerimizde hata oranı yüksek olacaktır.



## Overfitting

Eğer modelimiz, eğitim için kullandığımız veri setimiz üzerinde gereğinden fazla çalışıp ezber yapmaya başlamışsa ya da eğitim setimiz tek düzeye ise overfitting olma riski büyük demektir. Eğitim setinde yüksek bir skor aldığımız bu modele, test verimizi gösterdiğimizde muhtemelen çok düşük bir skor elde edeceğiz. Çünkü model eğitim setindeki durumları ezberlemiştir ve test veri setinde bu durumları aramaktadır. En ufak bir değişiklikte ezberlenen durumlar bulunamayacağı için test veri setinde çok kötü tahmin skorları elde edebilirsiniz. Overfitting problemi olan modellerde yüksek varyans, düşük bias durumu görülmektedir.

Bu genellikle model çok karmaşık olduğunda (yani gözlem sayısına kıyasla çok fazla özellik / değişken varsa) gerçekleşir. Bu model eğitim verilerinde çok yüksek tahminin doğruluğuna sahip olacaktır, ancak eğitimsiz veya yeni verilerde muhtemelen çok doğru tahminleme yapamayacaktır. Bu sorun modelin genelleştirme yapamamasından kaynaklanmaktadır. Bu tip modeller verilerdeki değişkenler arasındaki gerçek ilişkiler yerine eğitim verilerindeki “gürültüyü” öğrenir veya açıklar.

Overfitting problemi aşağıdaki yöntemler uygulanarak çözülebilmektedir;

- Öz nitelik sayısını azaltmak: Birbirleriyle yüksek korelasyonlu olan kolonlar silinebilir ya da faktör analizi gibi yöntemlerle bu değişkenlerden tek bir değişken oluşturulabilir.
- Daha fazla veri eklemek : Eğer eğitim seti tek düzeye ise daha fazla veri ekleyerek veri çeşitliliği arttırılır.

- **Regularization (Düzenleme)** : Düzenleme, modelin karmaşıklığını azaltmak için bir kullanılan tekniktir. Bunu kayıp fonksiyonunu cezalandırarak yapar. Yani modelde ağırlığı yüksek olan değişkenlerin ağırlığını azaltarak bu değişkenlerin etki oranını azaltır. Bu yöntem, aşırı öğrenme probleminin çözülmesine yardımcı olur. Kayıp fonksiyonu, gerçek değer ile öngörülen değer arasındaki farkın karelerinin toplamıdır. Değişkenlerin ağırlığını azaltmak için regularization değerini artttırmak gerekmektedir. En popüler Regularization metotları **Lasso** ve **Ridge** teknikleridir.

## **Underfitting**

Aşırı öğrenmenin aksine, bir model yetersiz öğrenmeye sahipse, modelin eğitim verilerine uymadığı ve bu nedenle verilerdeki trendleri kaçırduğu anlamına gelir. Ayrıca modelin yeni veriler için genelleştirilemediği anlamına da gelir. Tahmin ettiğiniz gibi bu problem genellikle çok basit bir modelin sonucudur (yetersiz tahminleyici bağımsız değişken eksikliği).

Underfitting sorunu olan modellerde hem eğitim hem de test veri setinde hata oranı yüksektir. Düşük varyans ve yüksek bias'a sahiptir. Bu modeller eğitim verilerini çok yakından takip etmek yerine, eğitim verilerinden alınan dersleri yok sayar ve girdiler ile çıktılar arasındaki temel ilişkiyi öğrenemez.

Underfit'in overfit kadar yaygın olmadığını belirtmek gerekmektedir. Yine de, veri analizinde her iki problemden de kaçınmak gereklidir.

## **Varyans-Bias Çelişkisi**

*Varyans*, model eğitim veri setinde iyi performans gösterdiğinde, ancak bir test veri kümesi veya doğrulama veri kümesi gibi, eğitilmemiş bir veri kümesinde iyi performans göstermediğinde ortaya çıkar. Varyans, gerçek değerden tahmin edilen değerin ne kadar dağıtık olduğunu söyler.

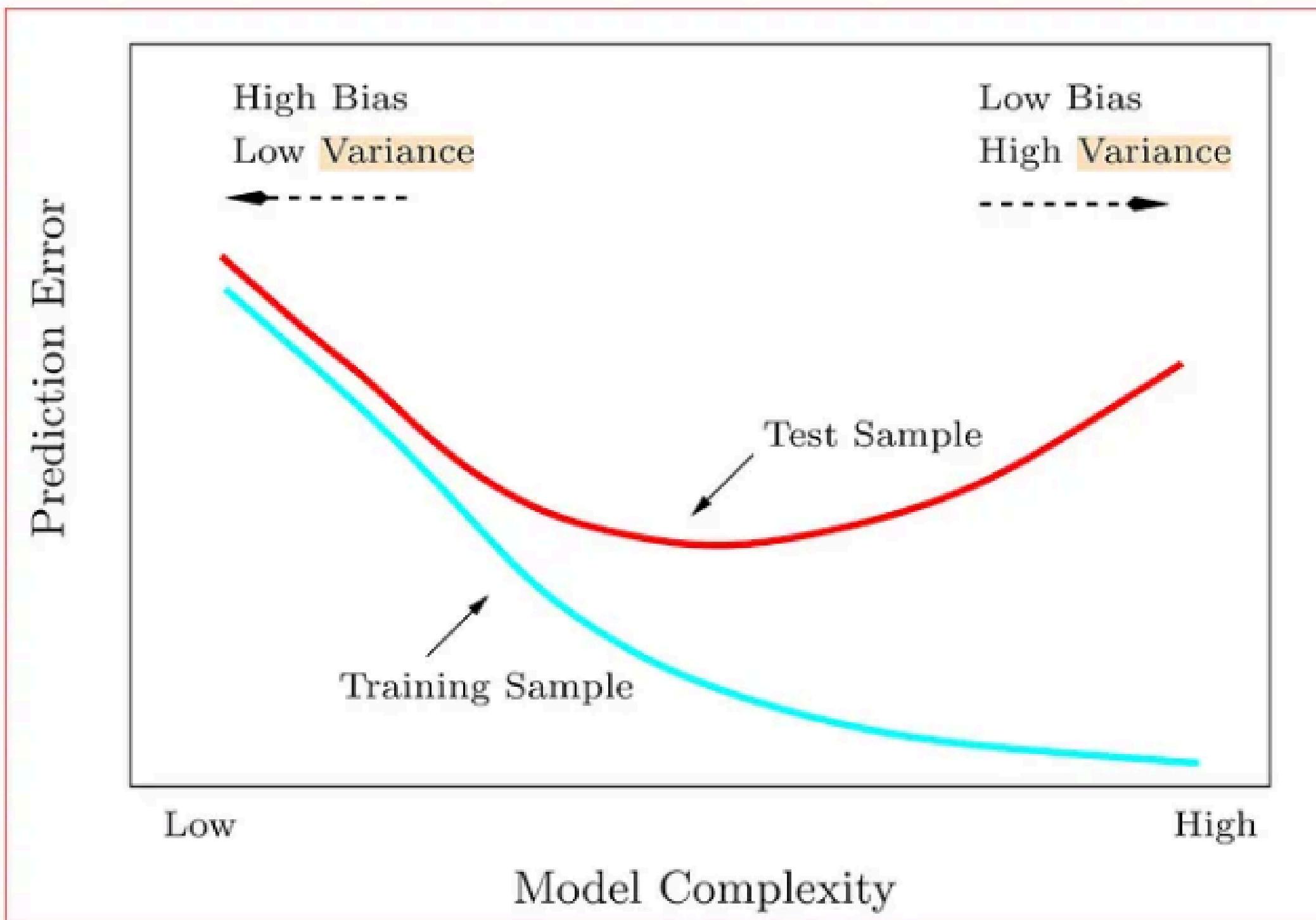
*Bias*, gerçek değerlerden tahmin edilen değerlerin ne kadar uzak olduğunu söyler. Tahmin edilen değerler gerçek değerlerden uzaksa, bias yüksektir.



High bias , high variance and just fit

Yukarıdaki grafiklere bakarsak, yüksek bias'a sahip bir modelin çok basit olduğu görülmektedir. Yüksek varyansa sahip bir model, veri noktalarının çoğu uymaya çalışır ve bu da modeli karmaşık yapar ve modellenmesini zorlaştırrır.

Aşağıdaki grafikten görüldüğü gibi model karmaşıklığı arttıkça eğitim seti üzerinde hatalı tahmin oranı azaltmakta ancak test veri seti üzerinde tahmin hatası artmaktadır.



- Yüksek Bias Düşük Varyans: Modeller tutarlıdır, ancak ortalama hata oranı yüksektir.
- Yüksek Bias Yüksek Varyans : Modeller hem hatalı hem de tutarsızdır .
- Düşük Bias Düşük Varyans: Modeller ortalama olarak doğru ve tutarlıdır. Modellerimizde bu sonucu elde etmek için çabalamaktayız.
- Düşük Bias Yüksek Varyans: Modeller bir dereceye kadar doğrudur ancak ortalamada tutarsızdır. Veri setinde ufak bir değişiklik yapıldığında büyük hata oranına neden olmaktadır.

Yüksek bias veya yüksek varyansa sahip olduğumuzu bulmanın yolu nedir?

Eğer model yüksek bias'a sahipse aşağıdaki sonuçlarla karşılaşmamız kaçınılmazdır;

- Modelin eğitim setinin hata oranı yüksektir.
- Test / doğrulama veri seti hata oranı eğitim seti ile benzer oranda yüksektir.

Eğer model yüksek varyans'a sahipse aşağıdaki sonuçlarla karşılaşmamız kaçınılmazdır;

- Modelin eğitim setinin hata oranı düşüktür.
- Modelin test/doğrulama veri setinin hata oranı yüksektir.

Yüksek bias problemini çözmek için aşağıdaki yöntemleri uygulayabiliriz.

- **Daha fazla veri eklemek** : Daha fazla veri ekleyerek veri çeşitliliğini artırmak gereklidir.
- **Daha fazla değişken eklemek** : Model karmaşıklığının artmasını sağlamaktadır.
- **Regularization (düzenleme)** : Değişkenlerin ağırlığını artırmak için regularization değerini azaltın.

# Düzenlileştirme (Regularization)

## Teknikleri ile Aşırı Öğrenmeyi Önleme: L1 ve L2 Düzenlileştirme

### Düzenlileştirme Nedir?

Düzenlileştirme, makine öğrenmesi ve derin öğrenme modellerinde aşırı öğrenmeyi önlemek için kullanılan bir tekniktir. Modelin karmaşıklığını kontrol altında tutarak, modelin eğitim verisine aşırı uyum sağlamasını ve genelleme yeteneğinin düşmesini engeller. Düzenlileştirme, modelin parametrelerine ceza ekleyerek gerçekleşir ve bu cezalar modelin daha basit ve genelleyici olmasını sağlar.

## L1 Düzenlileştirme (Lasso)

L1 düzenlileştirme, modelin bazı parametrelerinin sıfıra yakın olmasını teşvik eder. Bu, parametrelerin mutlak değerlerinin toplamını ceza terimi olarak ekleyerek yapılır. L1 düzenlileştirme, aşağıdaki maliyet fonksiyonuna eklenen bir terimle ifade edilir:

$$\text{Maliyet Fonksiyonu} = \text{Kayıp Fonksiyonu} + \lambda \sum_{i=1}^n |\theta_i|$$

Burada:

- $\lambda$ , düzenlileştirme parametresidir ve modelin ne kadar cezalandırılacağını belirler.
- $\theta_i$ , modelin parametreleridir.

L1 düzenlileştirme, bazı parametrelerin tamamen sıfıra inmesini sağladığı için modelde seçicilik sağlar ve gereksiz parametreleri elimine eder. Bu, özellikle yüksek boyutlu veri setlerinde etkili olabilir.

## L2 Düzenlileştirme (Ridge)

L2 düzenlileştirme, modelin parametrelerinin büyük olmasını cezalandırır ve tüm parametrelerin küçük olmasını teşvik eder. Bu, parametrelerin karelerinin toplamını ceza terimi olarak ekleyerek yapılır. L2 düzenlileştirme, aşağıdaki maliyet fonksiyonuna eklenen bir terimle ifade edilir:

$$\text{Maliyet Fonksiyonu} = \text{Kayıp Fonksiyonu} + \lambda \sum_{i=1}^n \theta_i^2$$

L2 düzenlileştirme, parametrelerin büyüklüğünü sınırlar ve modelin daha düzgün ve genelleyici olmasını sağlar. Bu, özellikle aşırı öğrenmenin önlenmesinde etkili bir yöntemdir.

## L1 ve L2'nin Birleşimi: Elastic Net

Elastic Net, L1 ve L2 düzenlileştirmenin birleşimidir ve her iki yöntemin avantajlarını bir araya getirir. Elastic Net, aşağıdaki maliyet fonksiyonuna eklenen bir terimle ifade edilir:

$$\text{Maliyet Fonksiyonu} = \text{Kayıp Fonksiyonu} + \lambda_1 \sum_{i=1}^n |\theta_i| + \lambda_2 \sum_{i=1}^n \theta_i^2$$

Elastic Net, hem parametrelerin sıfıra yakın olmasını teşvik eder hem de büyük parametreleri cezalandırır. Bu sayede, model hem daha seçici olur hem de genelleyici yeteneği artırılır.

# Örnek :

Bir araştırmacı, bir şirketin reklam bütçesinin satışlara etkisini incelemek istiyor. Elinde 50 gözlemden oluşan veri seti var ve değişkenler şunlar:

- $X_1$ : Televizyon reklam harcaması (bin \$)
- $X_2$ : Radyo reklam harcaması (bin \$)
- $X_3$ : Gazete reklam harcaması (bin \$)
- $Y$ : Satış miktarı (bin birim)

Araştırmacı, çoklu doğrusal regresyon kullanarak satışları tahmin etmek istiyor. Ancak  $X_1$  ve  $X_2$  arasında güçlü bir korelasyon var. Bu nedenle modelde hem L1 (Lasso) hem de L2 (Ridge) düzenlileştirmesini içeren Elastic Net uygulanması öneriliyor.

Elastic Net parametreleri:

- $\alpha = 0.1$  (toplam düzenlileştirme katsayısı)
- $l1\_ratio = 0.7$  (L1 ağırlığı)

1. Elastic Net'in kayıp fonksiyonunu yazınız.
2. Modelin ağırlıklarını (coefficients) ve bias'ını tahmini olarak yorumlayınız.
3. L1 ve L2 bileşenlerinin bu veri setinde hangi etkileri olacağını açıklayınız.

### Çözüm Adımları

#### 1. Elastic Net Kayıp Fonksiyonu

Elastic Net kayıp fonksiyonu:

$$\text{Loss} = \text{RSS} + \alpha \cdot (l1\_ratio \cdot \|\beta\|_1 + (1 - l1\_ratio) \cdot \|\beta\|_2^2)$$

Burada:

- $\text{RSS} = \sum_{i=1}^n (Y_i - \hat{Y}_i)^2$
- $\beta = [\beta_1, \beta_2, \beta_3]$
- $\alpha = 0.1, l1\_ratio = 0.7$

Yani:

$$\text{Loss} = \sum_{i=1}^{50} (Y_i - (\beta_0 + \beta_1 X_{1i} + \beta_2 X_{2i} + \beta_3 X_{3i}))^2 + 0.1 \cdot (0.7 \cdot (|\beta_1| + |\beta_2| + |\beta_3|) + 0.3 \cdot (\beta_1^2 + \beta_2^2 + \beta_3^2))$$

## 2. Tahmini Ağırlıklar (Coefficients) ve Bias

- $X_1$  ve  $X_2$  arasında yüksek korelasyon olduğu için L2 (Ridge) bileşeni ağırlıkları dengeleyerek çok büyük olmalarını engeller.
- L1 (Lasso) bileşeni, gereksiz veya çok küçük katkı yapan  $X_3$  gibi değişkenlerin ağırlığını sıfıra yaklaşırabilir.

Yaklaşık yorum:

- $\beta_1$  ve  $\beta_2 \rightarrow$  pozitif ve orta-yüksek değerler (çünkü yüksek korelasyon ve katkı)
- $\beta_3 \rightarrow$  küçük veya sıfıra yakın (gazete reklamının etkisi düşükse L1 sıfıra itebilir)
- $\beta_0 \rightarrow$  veri setinin ortalamasına göre bias

### 3. L1 ve L2 Bileşenlerinin Etkisi

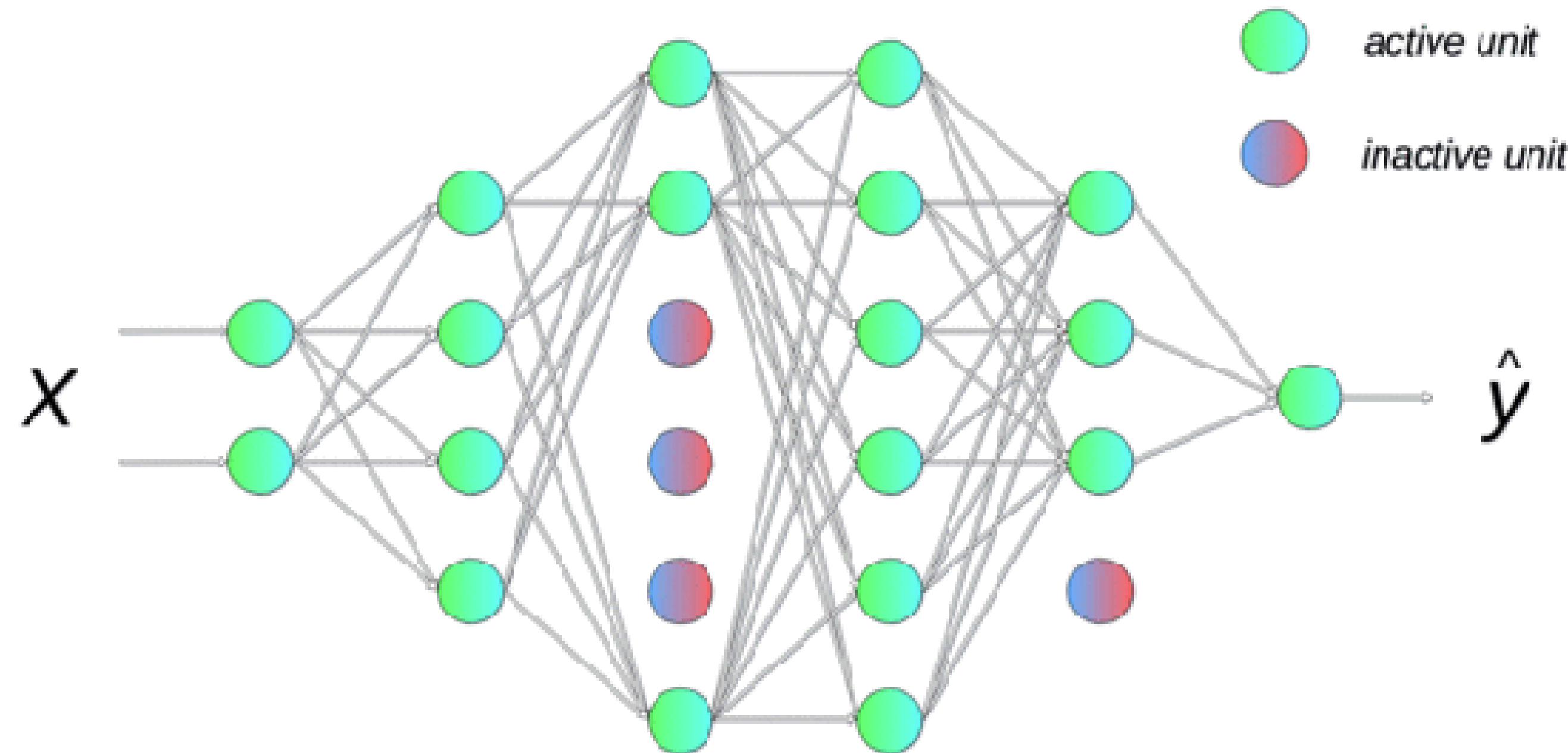
- L1 (Lasso) etkisi:
  - Bazı ağırlıkları tam sıfıra itebilir
  - Modeli daha sade ve yorumlanabilir yapar
- L2 (Ridge) etkisi:
  - Ağırlıkların çok büyük olmasını engeller
  - Özellikle korelasyonlu değişkenlerde dengeli dağılım sağlar
- Elastic Net:
  - L1 ve L2'yi birleştirerek hem değişken seçimi yapar hem de korelasyon sorunlarını dengeler.

# Dropout (Düşürme Tekniği) :

Yapay sinir ağları çok güçlündür; çok sayıda nöron ve bağlantıya sahiptir. Bu güç bazen ezberleme (**overfitting**) sorununa yol açar. Yani ağ, eğitim verisini çok iyi öğrenir ama yeni verilerde başarısı düşer. Dropout, bu sorunu engellemek için kullanılan bir düzenlileştirme (**regularization**) tekniğidir.

## Nasıl Çalışır?

1. Ağdaki nöronlardan rastgele bir kısmı seçilir.
2. Seçilen nöronların çıkışı 0 yapılır, yani o turda ağdan çıkarılır.
3. Kalan nöronlar normal şekilde çalışır ve ağırlıkları günceller.
4. Test veya tahmin aşamasında **dropout uygulanmaz**; tüm nöronlar aktif olur.
5. Eğitim sırasında kapanan nöronların etkisi dengelensin diye testte çıkışlar uygun şekilde ölçeklendirilir.



$$p^{[0]} = 0.0 \quad p^{[1]} = 0.0 \quad p^{[2]} = 0.5 \quad p^{[3]} = 0.0 \quad p^{[4]} = 0.25$$

Dropout'un çalışma şekli

## Dropout Oranı

- Dropout oranı, kapatılacak nöronların yüzdesidir.
- Örnekler:
  - $0.5 \rightarrow$  Nöronların %50'si kapatılır
  - $0.2 \rightarrow$  Nöronların %20'si kapatılır
- Çok yüksek oran  $\rightarrow$  öğrenme zorlaşır.
- Çok düşük oran  $\rightarrow$  etkisi azalır.
- Genellikle 0.2–0.5 arası değerler kullanılır.

## Avantajları

- Overfitting'i azaltır: Model, eğitim verisini ezberlemez.
- Genelleme yeteneğini artırır: Yeni verilere daha iyi uyum sağlar.
- Daha dayanıklı ve sağlam modeller üretir: Farklı alt ağlar öğrenildiği için model kararlı hale gelir.

## Dezavantajları

- Eğitim süresini biraz uzatır.
- Yanlış dropout oranı seçilirse model öğrenmede zorluk yaşar.

## Matematiksel Gösterim

Bir nöronun çıkışı  $h$  olsun. Dropout uygulandığında:

$$h' = h \cdot r$$

- $r$  = rastgele 0 veya 1 (Bernoulli dağılımı)
  - $r = 0 \rightarrow$  nöron kapalı
  - $r = 1 \rightarrow$  nöron açık

## Örnek:

Bir sinir ağı katmanı şu şekilde:

- 4 giriş nöronu:  $x = [1, 2, 3, 4]$
- 2 çıkış nöronu:  $y = Wx$ , ağırlık matrisi:

$$W = \begin{bmatrix} 0.1 & 0.2 & 0.3 & 0.4 \\ 0.5 & 0.6 & 0.7 & 0.8 \end{bmatrix}$$

- Dropout oranı  $p = 0.5$  (yani %50 nöron kapatılacak)

Soru: Eğitim sırasında bir dropout maskesi  $m = [1, 0, 1, 0]$  olarak seçildiğinde katman çıkışı ne olur?

## Çözüm Adımları

### 1. Dropout maskesini uygula:

Maskeyi giriş nöronlarına uygularız:

$$x_{\text{drop}} = x \odot m = [1, 2, 3, 4] \odot [1, 0, 1, 0] = [1, 0, 3, 0]$$

### 2. Lineer dönüşümü hesapla:

$$y = Wx_{\text{drop}} = \begin{bmatrix} 0.1 & 0.2 & 0.3 & 0.4 \\ 0.5 & 0.6 & 0.7 & 0.8 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 3 \\ 0 \end{bmatrix}$$

İlk çıkış nöronu:

$$0.1 * 1 + 0.2 * 0 + 0.3 * 3 + 0.4 * 0 = 0.1 + 0 + 0.9 + 0 = 1.0$$

İkinci çıkış nöronu:

$$0.5 * 1 + 0.6 * 0 + 0.7 * 3 + 0.8 * 0 = 0.5 + 0 + 2.1 + 0 = 2.6$$

Yani, dropout sonrası çıkış:

$$y_{\text{drop}} = [1.0, 2.6]$$

# Uygulama: Dropout uygulaması ile eğitim performansı karşılaştırması:

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.utils import to_categorical
```

## Açıklama:

- Gerekli kütüphaneleri yükliyoruz:
  - **numpy** : Sayısal işlemler için
  - **tensorflow** ve **keras** : Derin öğrenme modeli oluşturmak için
  - **mnist** : El yazısı rakamlar veri seti
  - **Sequential** : Katmanlı model yapısı
  - **Dense** ve **Dropout** : Sinir ağları katmanları
  - **Adam** : Optimizasyon algoritması
  - **to\_categorical** : Etiketleri one-hot encoding'e dönüştürme

```
def load_data():
    # MNIST veri setini yükle
    (x_train, y_train), (x_test, y_test) = mnist.load_data()

    # Veriyi normalize et (0-255 -> 0-1) ve düzleştir (28x28 -> 784)
    x_train = x_train.reshape(-1, 784).astype('float32') / 255.0
    x_test = x_test.reshape(-1, 784).astype('float32') / 255.0

    # Etiketleri one-hot encoding yap (örnek: 5 -> [0,0,0,0,0,1,0,0,0,0])
    y_train = to_categorical(y_train, 10)
    y_test = to_categorical(y_test, 10)

    return (x_train, y_train), (x_test, y_test)
```

## Açıklama:

- MNIST veri setini yükliyoruz (60k eğitim, 10k test örneği)
- Görüntüleri 28x28'den 784 boyutlu vektörlere dönüştürüyoruz
- Piksel değerlerini 0-1 aralığına normalize ediyoruz
- Sınıf etiketlerini one-hot encoding formatına çeviriyoruz

```
def create_model_without_dropout():
    model = Sequential([
        Dense(512, activation='relu', input_shape=(784,)), # 1. Gizli Katman
        Dense(512, activation='relu'),                      # 2. Gizli Katman
        Dense(512, activation='relu'),                      # 3. Gizli Katman
        Dense(10, activation='softmax')                    # Çıkış Katmanı
    ])
    return model
```

### Açıklama:

- 3 gizli katmanlı bir sinir ağı oluşturuyoruz:
  - Her katmanda 512 nöron ve ReLU aktivasyon fonksiyonu
  - Çıkış katmanında 10 nöron (10 sınıf için) ve softmax aktivasyonu
- Dropout katmanı içermiyor

```
def create_model_with_dropout():
    model = Sequential([
        Dense(512, activation='relu', input_shape=(784,)),
        Dropout(0.5), # %50 dropout
        Dense(512, activation='relu'),
        Dropout(0.5), # %50 dropout
        Dense(512, activation='relu'),
        Dropout(0.5), # %50 dropout
        Dense(10, activation='softmax')
    ])
    return model
```

## Açıklama:

- Önceki modelle aynı yapıda, ancak:
  - Her gizli katmandan sonra %50 dropout ekleniyor
  - Dropout, eğitim sırasında rastgele nöronları devre dışı bırakarak overfitting'i önlüyor

```
def train_and_evaluate(model, x_train, y_train, x_test, y_test, epochs=20, batch_size=128):
    # Modeli derle (Adam optimizer ve cross-entropy loss ile)
    model.compile(optimizer=Adam(learning_rate=0.001),
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])

    # Modeli eğit
    history = model.fit(x_train, y_train,
                         batch_size=batch_size,
                         epochs=epochs,
                         validation_data=(x_test, y_test),
                         verbose=1)

    return history.history
```

### Açıklama:

- Modeli Adam optimizer ile derliyoruz (öğrenme oranı 0.001)
- Kayıp fonksiyonu olarak kategorik cross-entropy kullanıyoruz
- 20 epoch boyunca eğitim yapıyoruz (batch size = 128)
- Eğitim ve validation (test) verisi üzerindeki performansı kaydediyoruz

```
def main():
    # Veriyi yükle
    (x_train, y_train), (x_test, y_test) = load_data()

    # Dropout KULLANMAYAN model
    print("Dropout KULLANMAYAN model eğitiliyor...")
    model_no_dropout = create_model_without_dropout()
    history_no_dropout = train_and_evaluate(model_no_dropout, x_train, y_train, x_test,
y_test)

    # Dropout KULLANAN model
    print("\nDropout KULLANAN model eğitiliyor...")
    model_with_dropout = create_model_with_dropout()
    history_with_dropout = train_and_evaluate(model_with_dropout, x_train, y_train, x_t
est, y_test)

    # Sonuçları karşılaştır
    print("\nKarşılaştırma Sonuçları:")
    print(f"Dropout KULLANMAYAN - Son epoch test accuracy: {history_no_dropout['val_ac
curacy'][-1]:.4f}")
    print(f"Dropout KULLANAN - Son epoch test accuracy: {history_with_dropout['val_accu
racy'][-1]:.4f}")

if __name__ == "__main__":
    main()
```

## Açıklama:

1. MNIST veri setini yükliyoruz
2. Dropout kullanmayan modeli oluşturup eğitiyoruz
3. Dropout kullanan modeli oluşturup eğitiyoruz
4. İki modelin test doğruluk (accuracy) değerlerini karşılaştırıyoruz

## Beklenen Çıktı

Dropout KULLANMAYAN model eğitiliyor...

Epoch 1/20

```
469/469 [=====] - 4s 8ms/step - loss: 0.2200 - accuracy: 0.933
8 - val_loss: 0.1023 - val_accuracy: 0.9683
```

...

Epoch 20/20

```
469/469 [=====] - 3s 7ms/step - loss: 0.0012 - accuracy: 0.999
8 - val_loss: 0.1325 - val_accuracy: 0.9802
```

Dropout KULLANAN model eğitiliyor...

Epoch 1/20

```
469/469 [=====] - 4s 8ms/step - loss: 0.5068 - accuracy: 0.846
1 - val_loss: 0.1493 - val_accuracy: 0.9555
```

...

Epoch 20/20

```
469/469 [=====] - 3s 7ms/step - loss: 0.1196 - accuracy: 0.964
8 - val_loss: 0.0757 - val_accuracy: 0.9825
```

Karşılaştırma Sonuçları:

Dropout KULLANMAYAN - Son epoch test accuracy: 0.9802

Dropout KULLANAN - Son epoch test accuracy: 0.9825

## Sonuç Yorumu:

- Dropout kullanan model genellikle daha iyi validation accuracy elde eder (overfitting'i önler)
- Dropout kullanmayan model eğitim verisinde çok yüksek accuracy (%99.98) gösterirken, test verisinde daha düşük performans gösterir (overfitting belirtisi)