



İleri Yapay Zeka

Hazırlayan: Doç. Dr. Yunus Egi

Ders 2



Ana Başlıklar

- Derin Öğrenme uygulaması

Örnek Problem

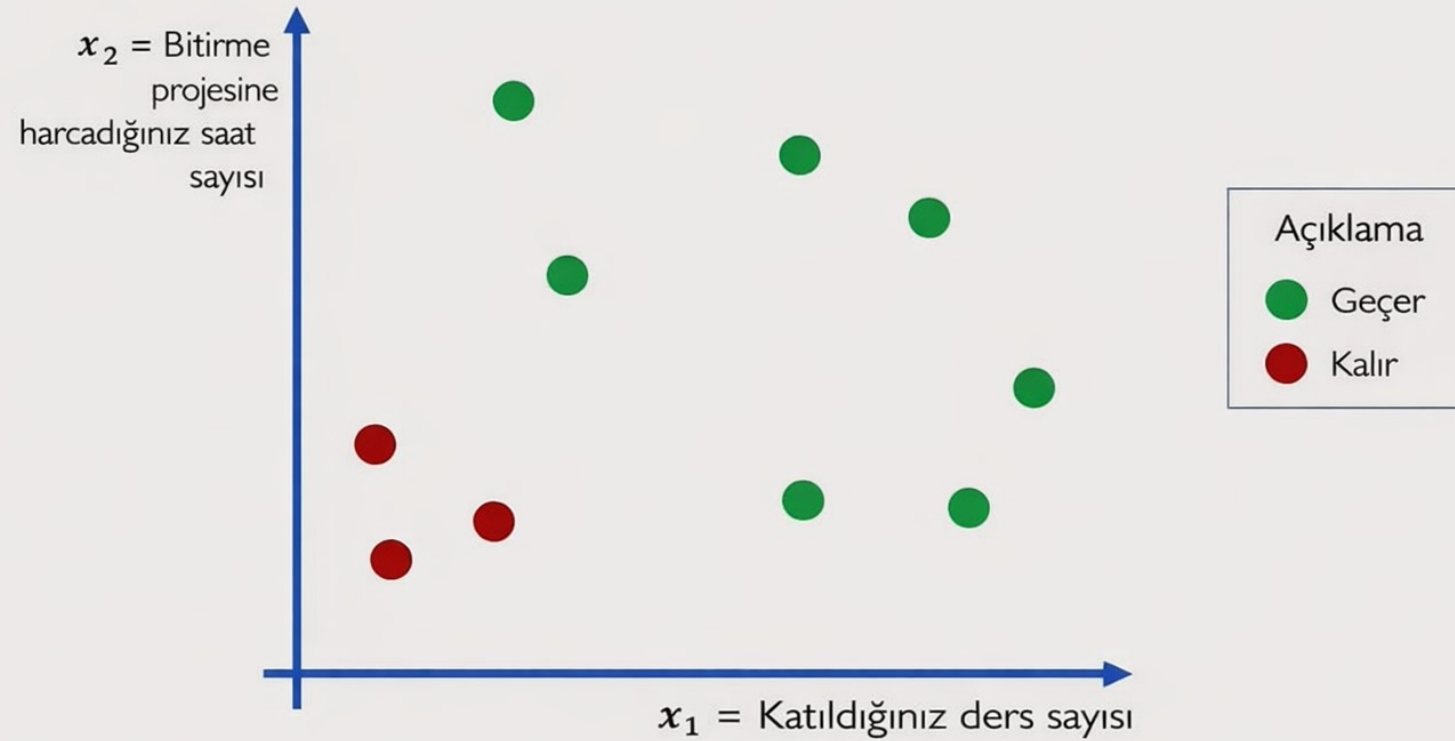
Bu dersi geçecek miyim?

Basit iki özellikli bir modelle başlayalım

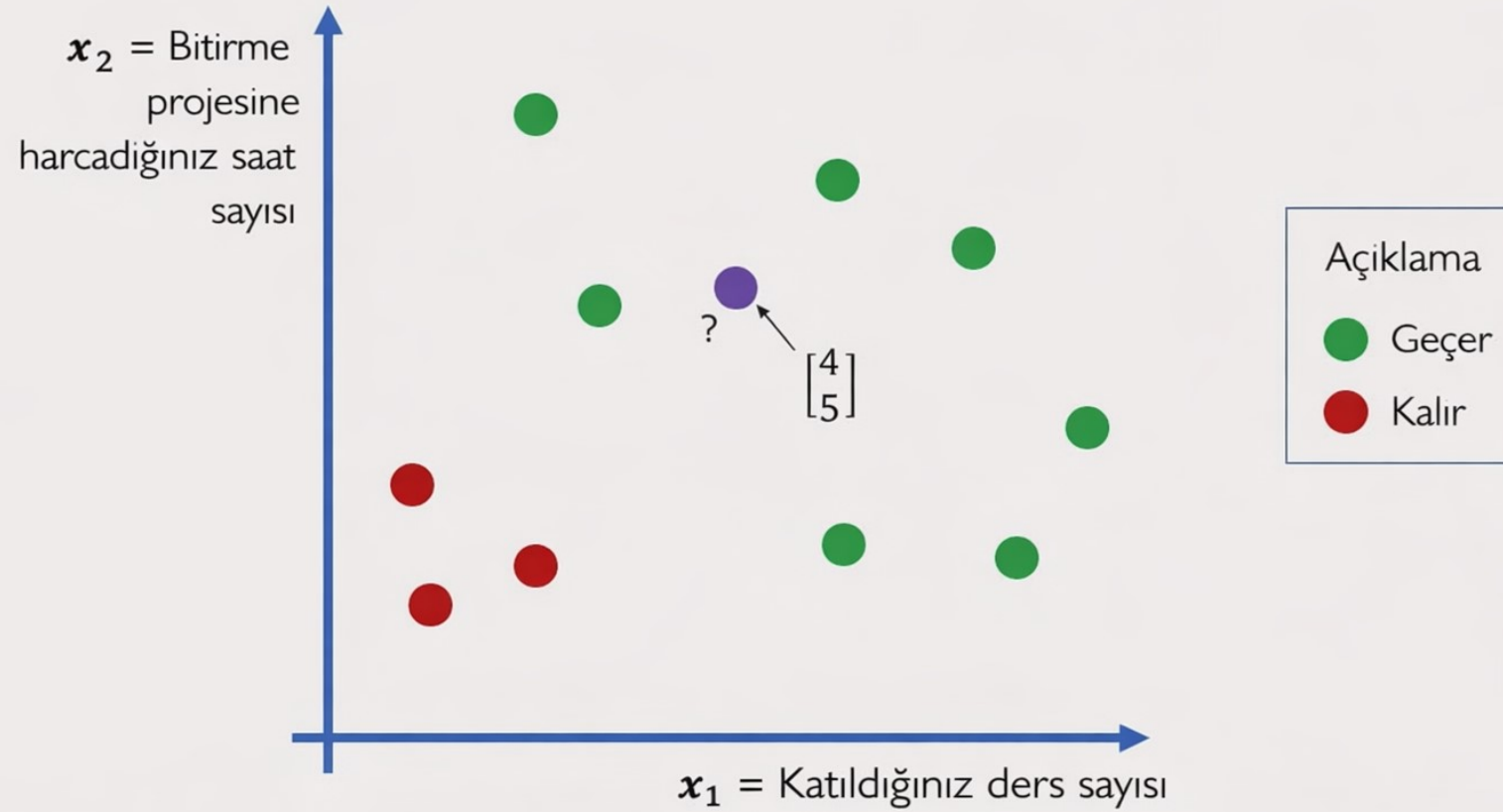
x_1 = Katıldığınız ders sayısı

x_2 = Bitirme projesine harcadığınız saat sayısı

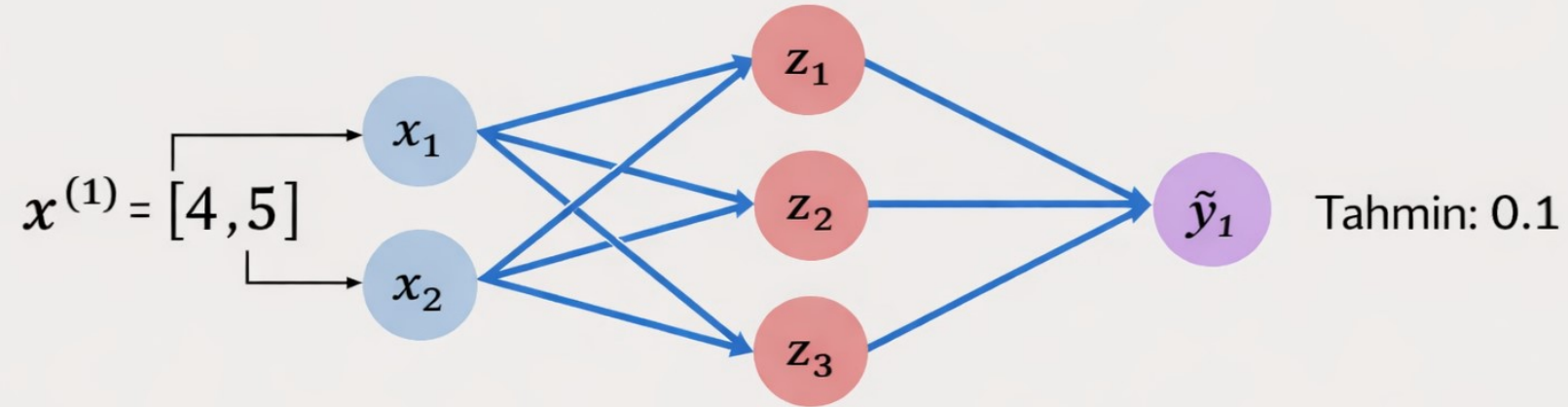
Örnek Problem: Bu dersi geçecek miyim?



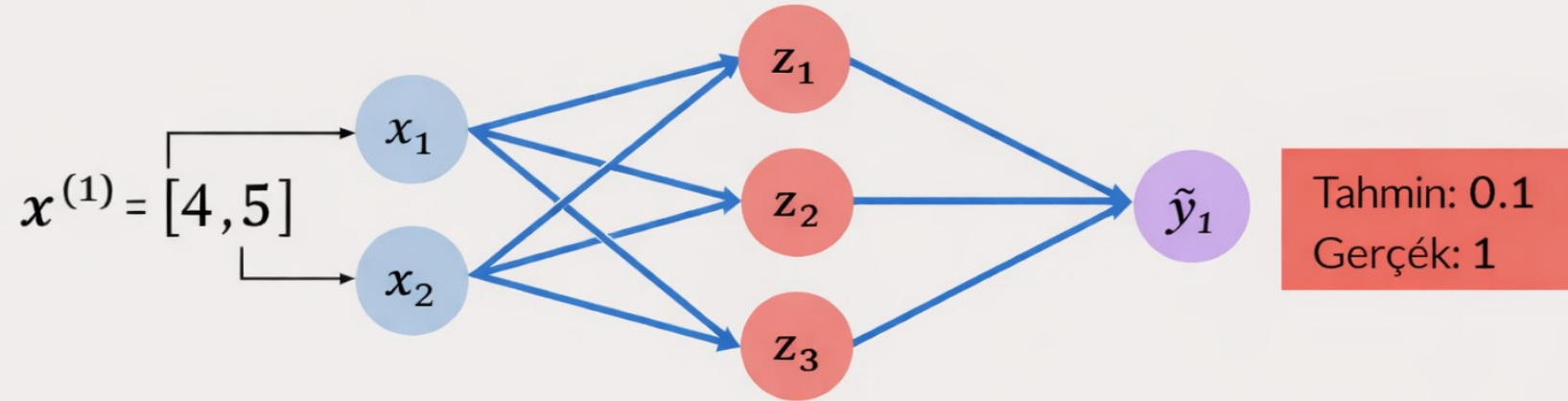
Örnek Problem: Bu dersi geçecek miyim?



Örnek Problem: Bu dersi geçecek miyim?

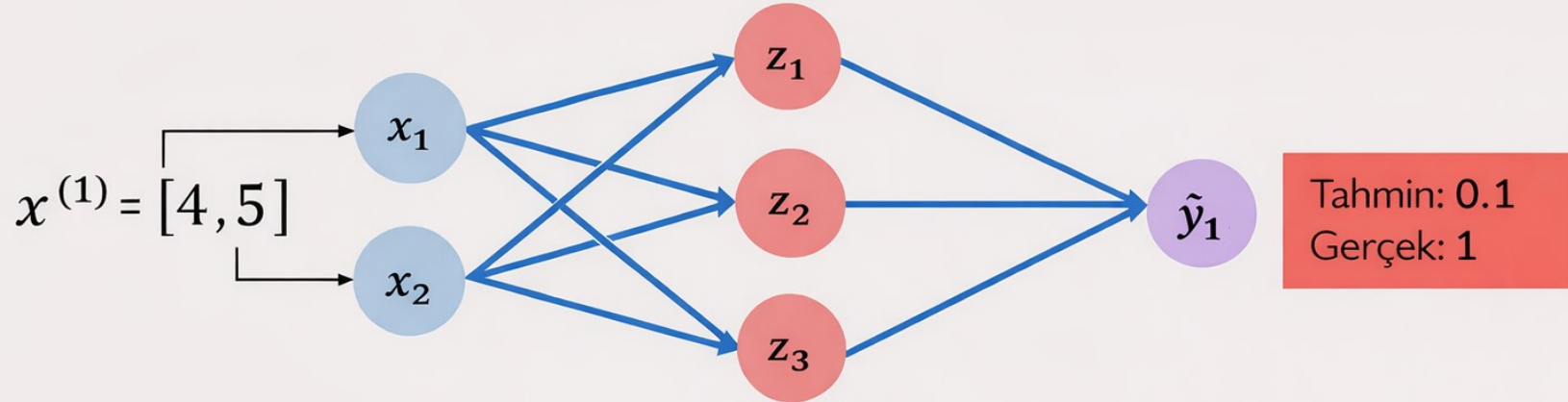


Örnek Problem: Bu dersi geçecek miyim?



Kayıp Ölçümü

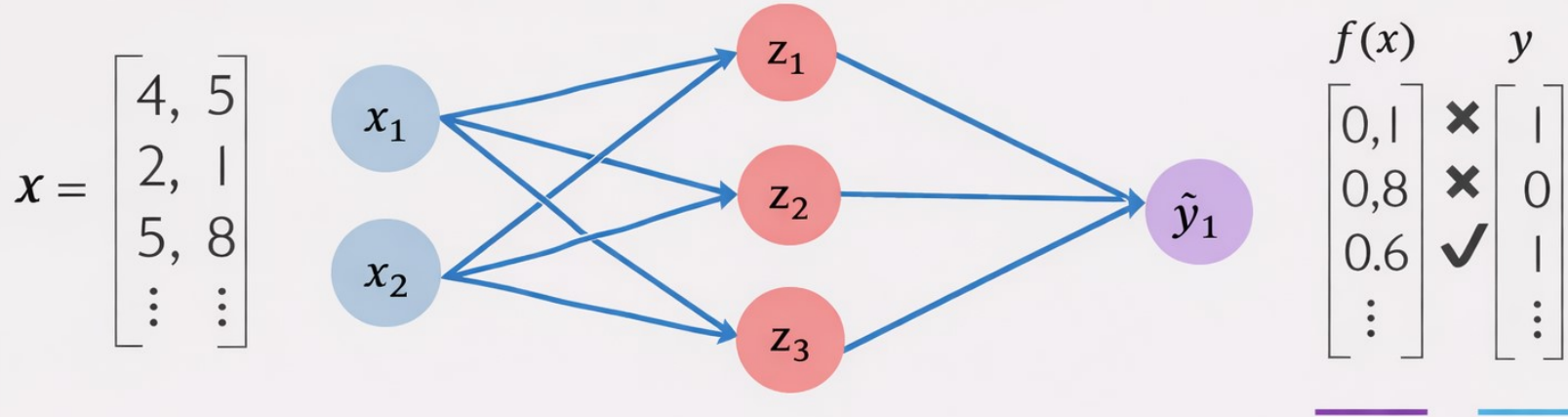
Ağımızın **kayıbı**, hatalı tahminlerden kaynaklanan maliyeti ölçer



$$\mathcal{L}(\underbrace{f(x^{(i)}, W)}_{\text{Tahmin}}, \underbrace{y^{(i)}}_{\text{Gerçek}})$$

Ampirik Kayıp

Ampirik kayıp, tüm veri kümemiz üzerindeki toplam kaybi ölçer



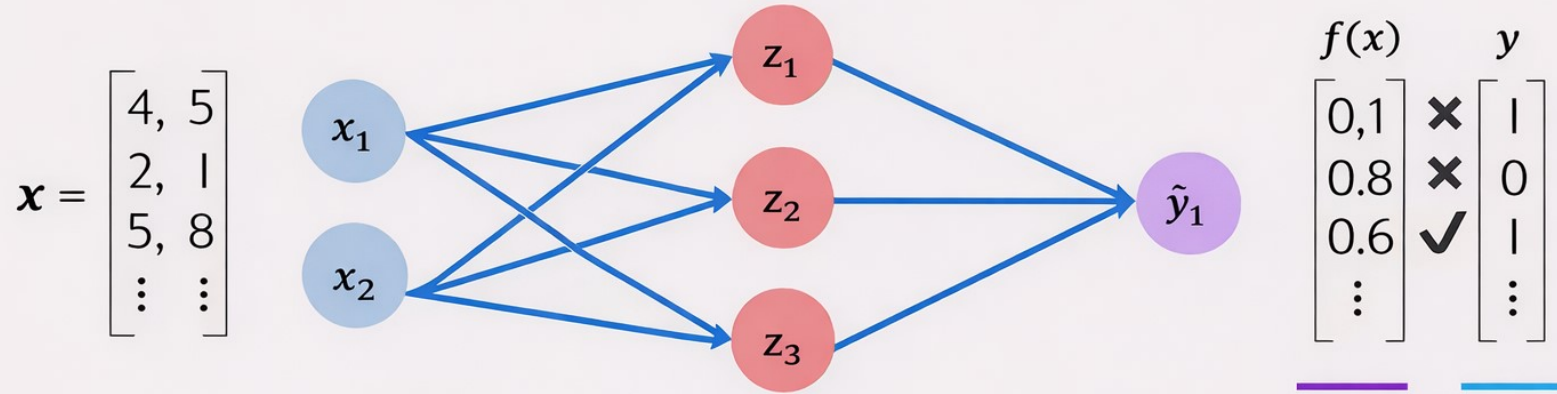
Aynı zamanda:

- Amaç fonksiyonu
- Maliyet Fonksiyonu
- Ampirik Risk

$$J(W) = \frac{1}{n} \sum_{i=1}^n \mathcal{L}(\underbrace{f(x^{(i)}, W)}_{\text{Tahmin}}, \underbrace{y^{(i)}}_{\text{Gerçek}})$$

Binary Cross Entropy Kaybı

Binary cross entropy kaybı, 0 ile 1 arasında olasılık veren modellerle kullanılabilir



$$J(W) = -\frac{1}{n} \sum_{i=1}^n \underbrace{y^{(i)}}_{\text{Gerçek}} \log(\underbrace{f(x^{(i)}; W)}_{\text{Tahmin}}) + (1 - \underbrace{y^{(i)}}_{\text{Gerçek}}) \log(1 - \underbrace{f(x^{(i)}; W)}_{\text{Tahmin}})$$



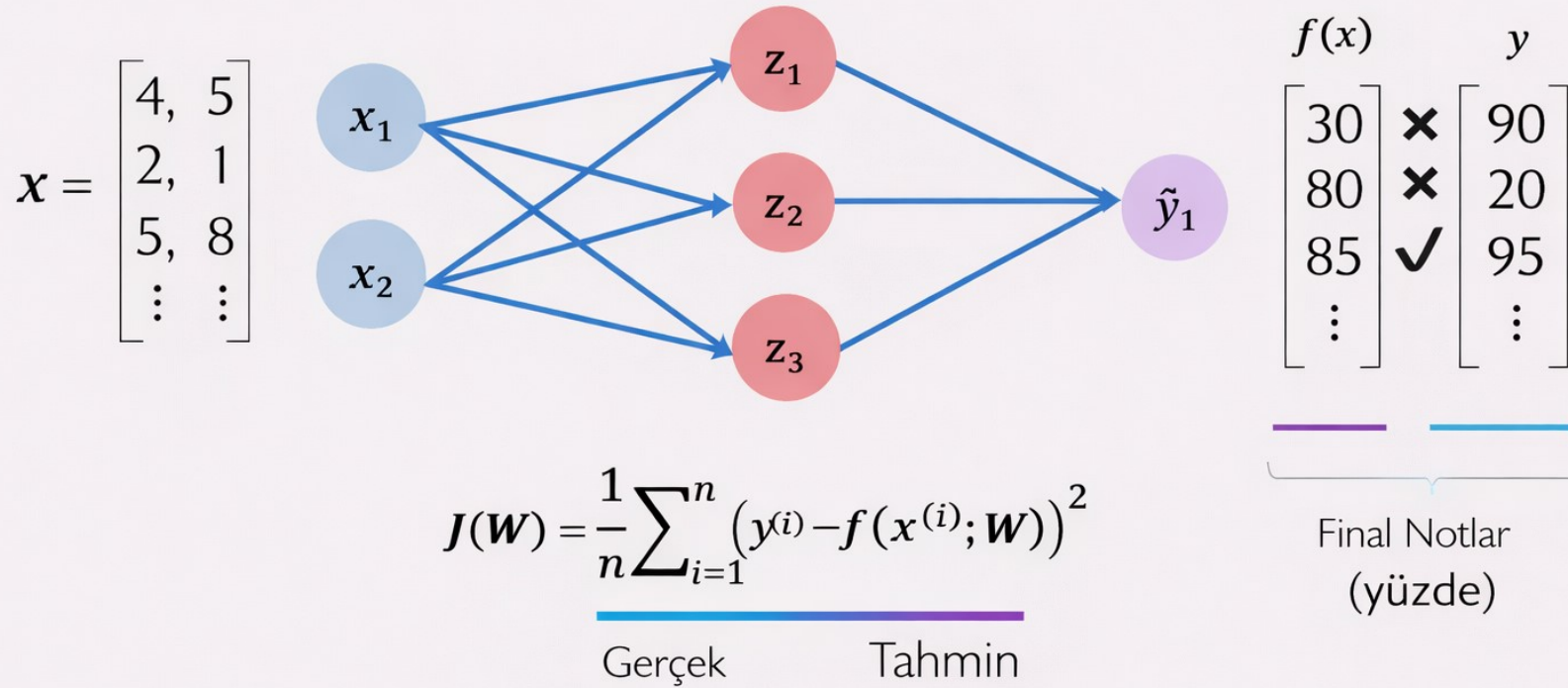
```
kayıp = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(y, predicted))
```



```
kayıp = torch.nn.functional.cross_entropy(predicted, y)
```

Ortalamanın Karesi Hata Kaybı

Ortalamanın karesi hata kaybı sürekli gerçek sayılar üreten regresyon modelleri ile



Sinir Ağının Eğitim

Kayıp Optimizasyonu

En düşük kaybı sağlayan **ağ ağırlıklarını** bulmak istiyoruz

$$W^* = \operatorname{argmin}_W \frac{1}{n} \sum_{i=1}^n \mathcal{L}(f x^{(i)}; W), y^{(i)})$$

$$W^* = \operatorname{argmin}_W J(W)$$

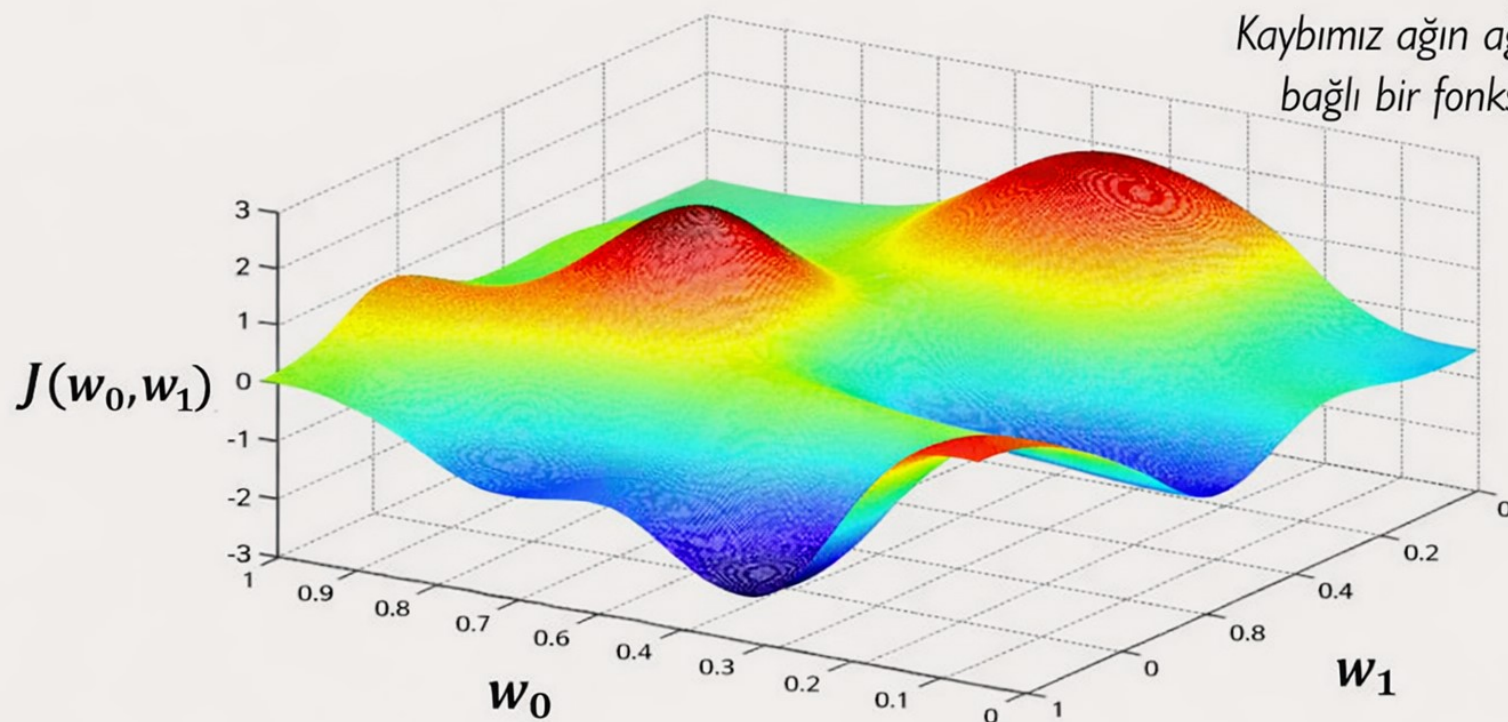
Remember:
 $W = \{W^{(0)}, W^{(1)}, \dots\}$

Kayıp Optimizasyonu

$$W^* = \underset{W}{\operatorname{argmin}} J(W)$$

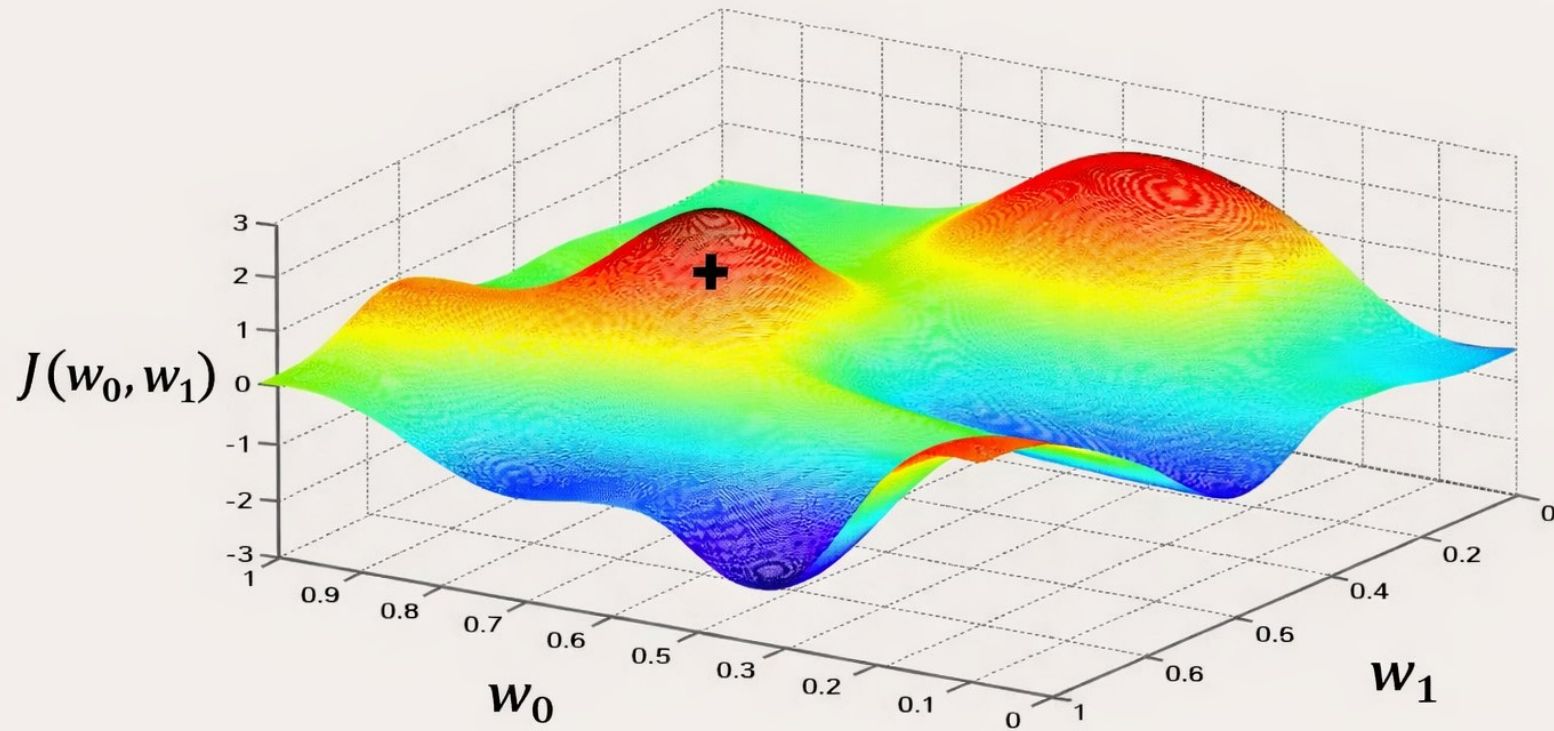
Unutmayın:

*Kaybımız ağırlıklarına
bağlı bir fonksiyondur!*



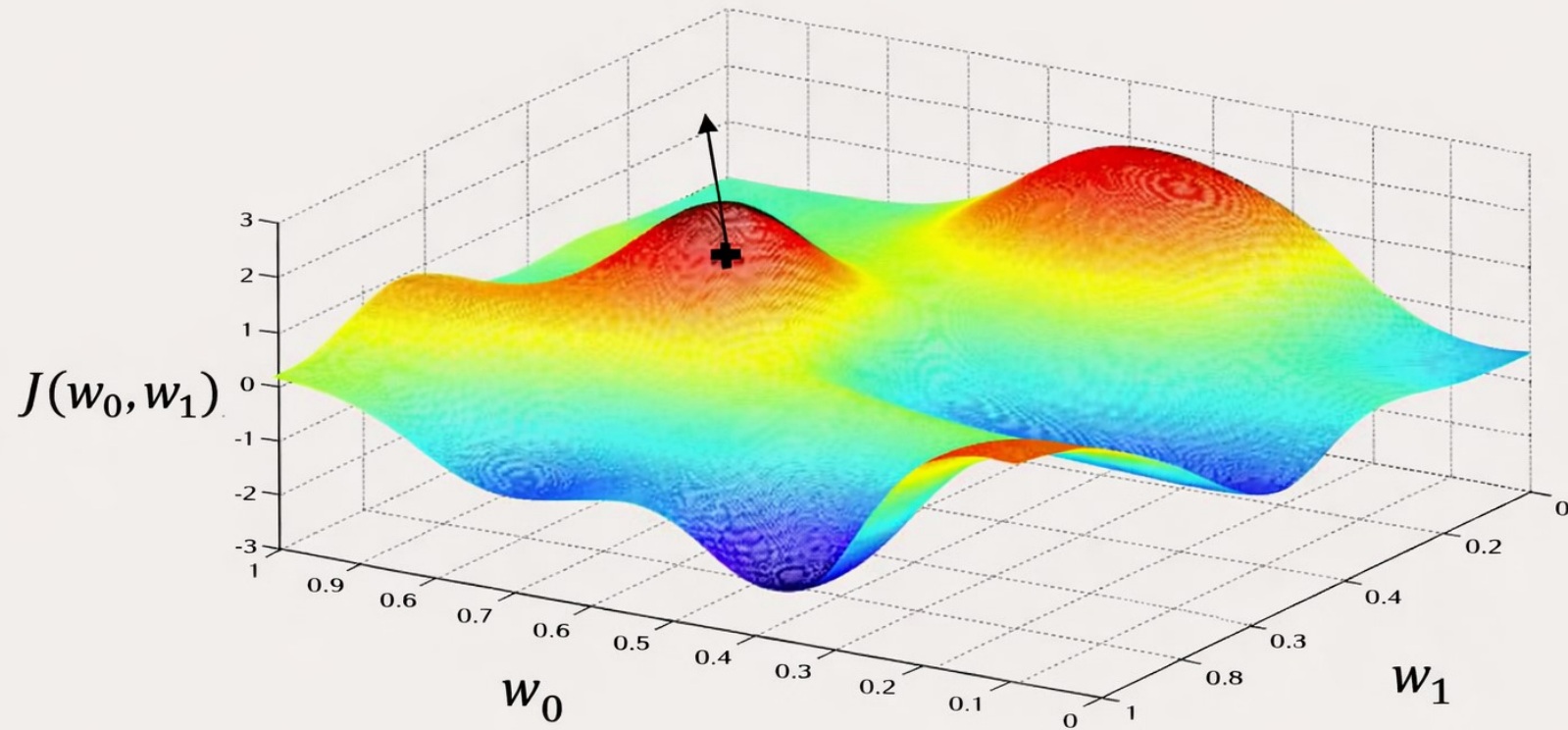
Kayıp Optimizasyonu

Rastgele bir başlangıç noktası (w_0, w_1) seç



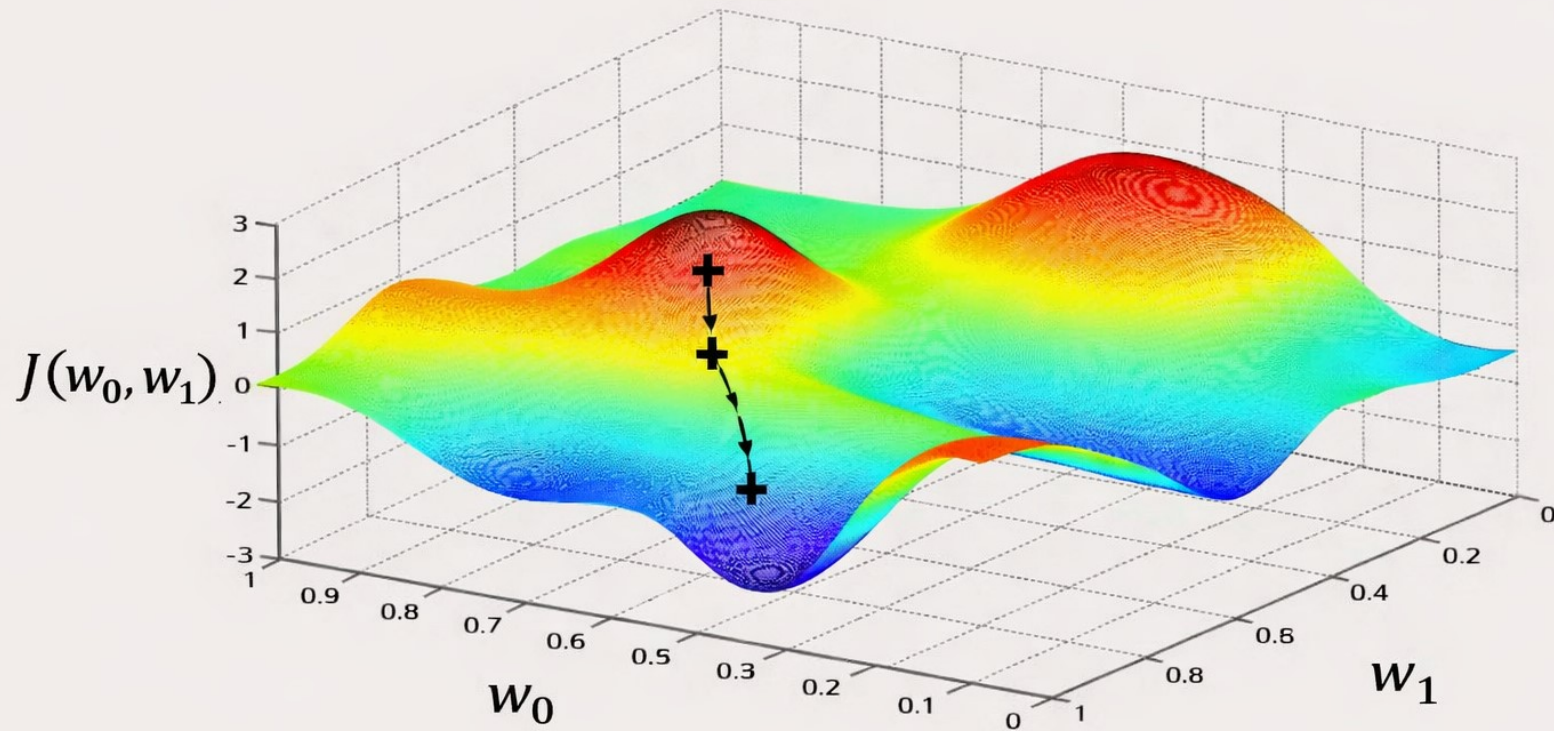
Kayıp Optimizasyonu

Gradyanı hesapla: $\frac{\partial J(W)}{\partial W}$



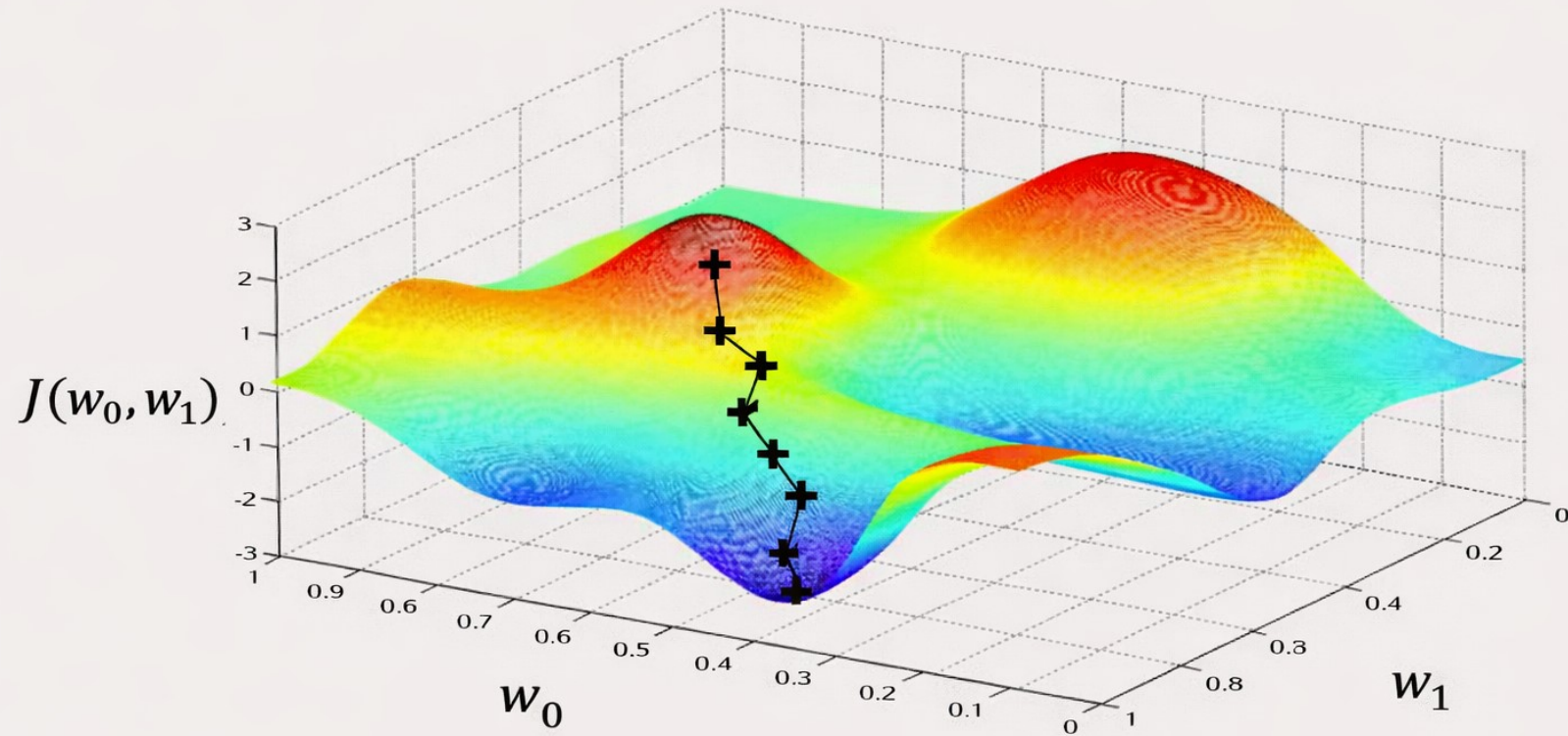
Kayıp Optimizasyonu

Gradientin ters yönünde küçük bir adım at



Gradyan İnişi

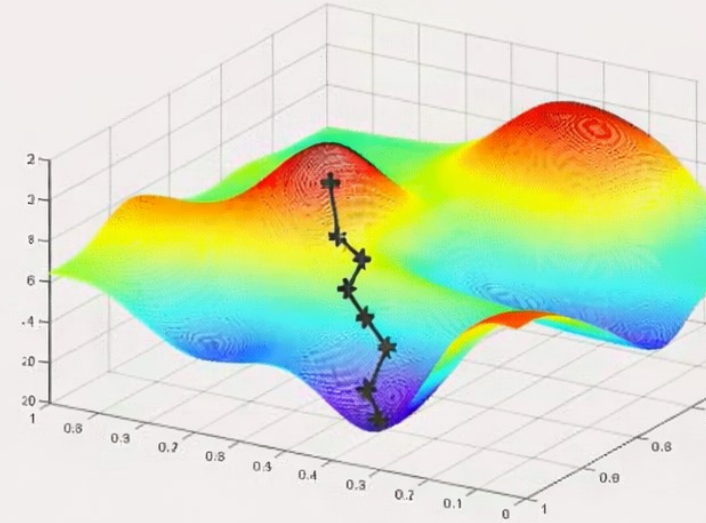
Yakınsamaya kadar tekrarla



Gradyan İnişi

Algoritma

1. Ağırlıkları rastgele baslat $\sim \mathcal{N}(0, \sigma^2)$
2. Ta ki yakınsama olana kadar:
3. Gradyanı hesapla, $\frac{\partial J(W)}{\partial W}$
4. Ağırlıkları güncelle, $\mathbf{W} \leftarrow \mathbf{w} - \frac{\partial J(W)}{\partial W}$
5. Ağırlıkları döndür

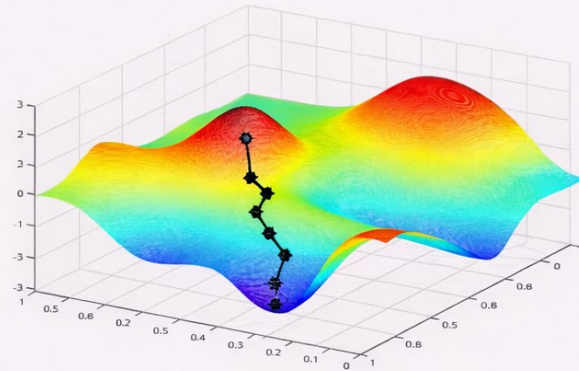


Cok hesaplama
yoğun olabilir!

Gradyan İnişi

Algoritma

1. Ağırlıkları rastgele başlat $\sim \mathcal{N}(0, \sigma^2)$
2. Yakınsama sağlanana kadar döngü:
3. Gradyanı hesapla, $\frac{\partial J(W)}{\partial W}$
4. Ağırlıkları güncelle, $W \leftarrow W - \eta \frac{\partial J(W)}{\partial W}$
5. Ağırlıkları döndür



Hesaplama açısından
çok yoğun olabilir!



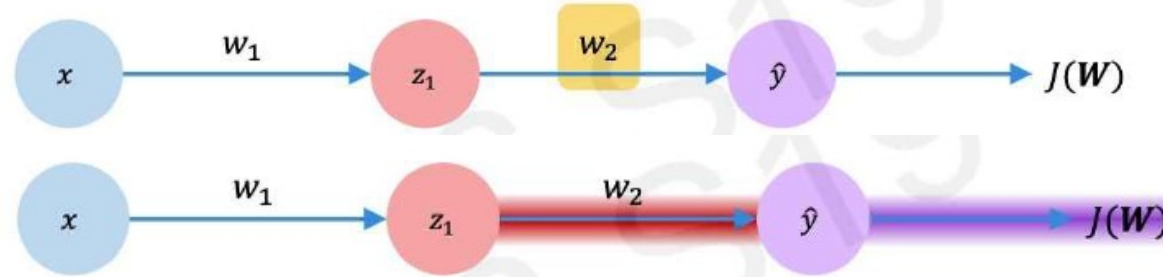
```
import tensorflow as tf

weights = tf.Variable([tf.random.normal()])

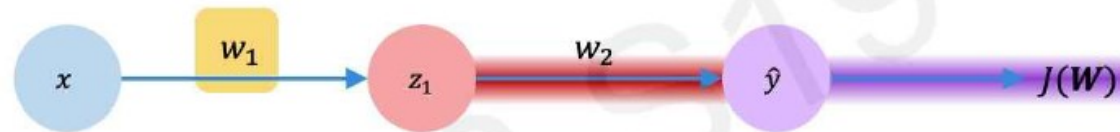
while True:  # loop forever
    with tf.GradientTape() as g:
        loss = compute_loss(weights)
        gradient = g.gradient(loss, weights)

    weights = weights - lr * gradient
```

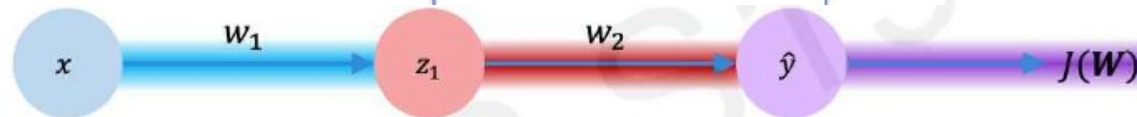
Gradyan İnişi: Geri Yayılım hesabı



Zincir kuralı uygulanırsa $\frac{\partial J(W)}{\partial w_2} = \frac{\partial J(W)}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial w_2}$



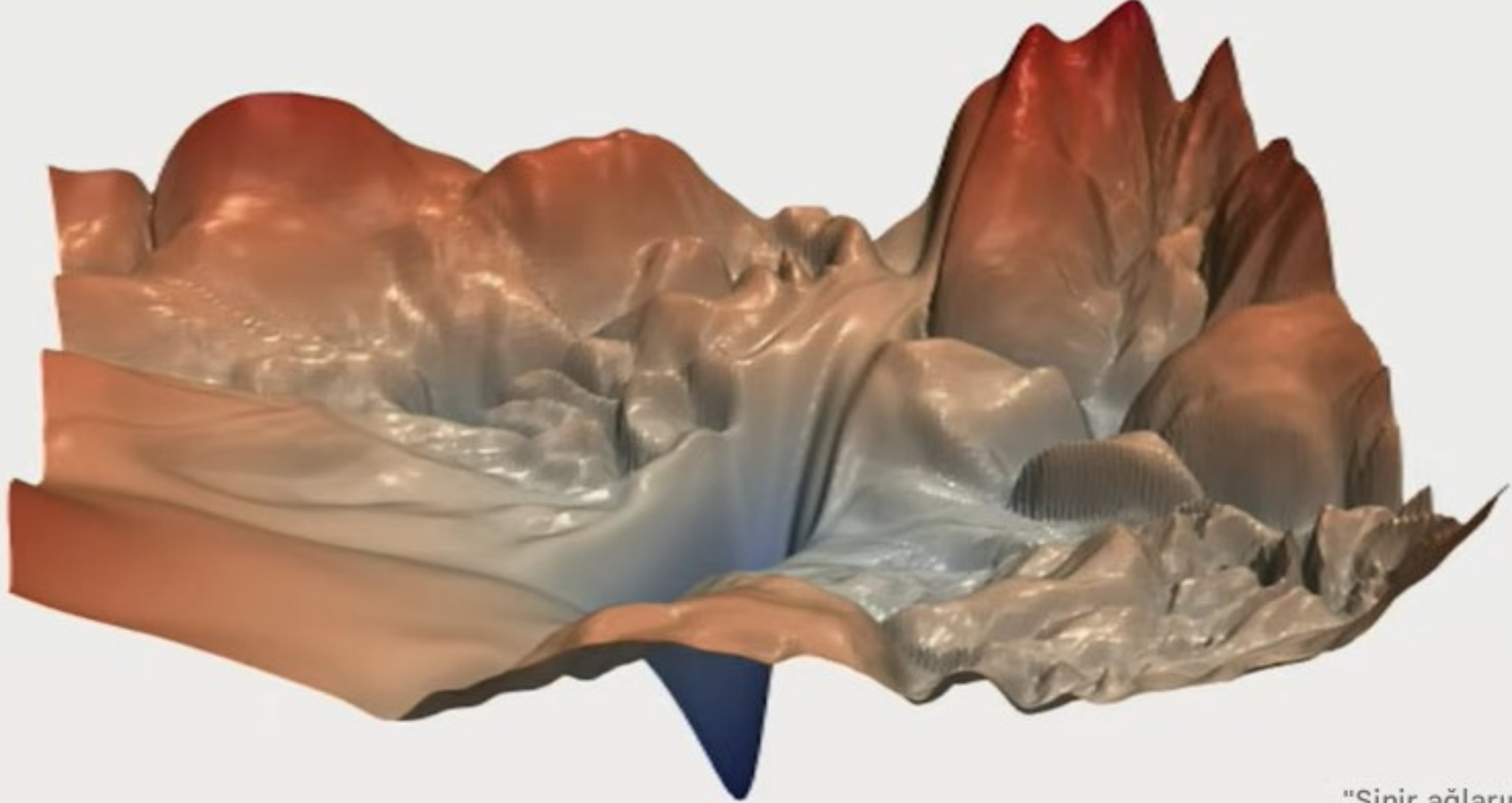
$$\frac{\partial J(W)}{\partial w_1} = \frac{\partial J(W)}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial w_1}$$



$$\frac{\partial J(W)}{\partial w_1} = \frac{\partial J(W)}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial z_1} * \frac{\partial z_1}{\partial w_1}$$

Pratikte Sinir Ağları: Optimizasyon

Sinir ađlarını eđitmek zordur.



"Sinir ađlarının kayıp dađılımını
görselleřtirme". Aralık 2017.

Kayıp Fonksiyonlarını Optimize Etmek Zor Olabilir

Unutmayın:

Gradyan inişi ile optimizasyon

$$\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\partial J(\mathbf{W})}{\partial \mathbf{W}}$$

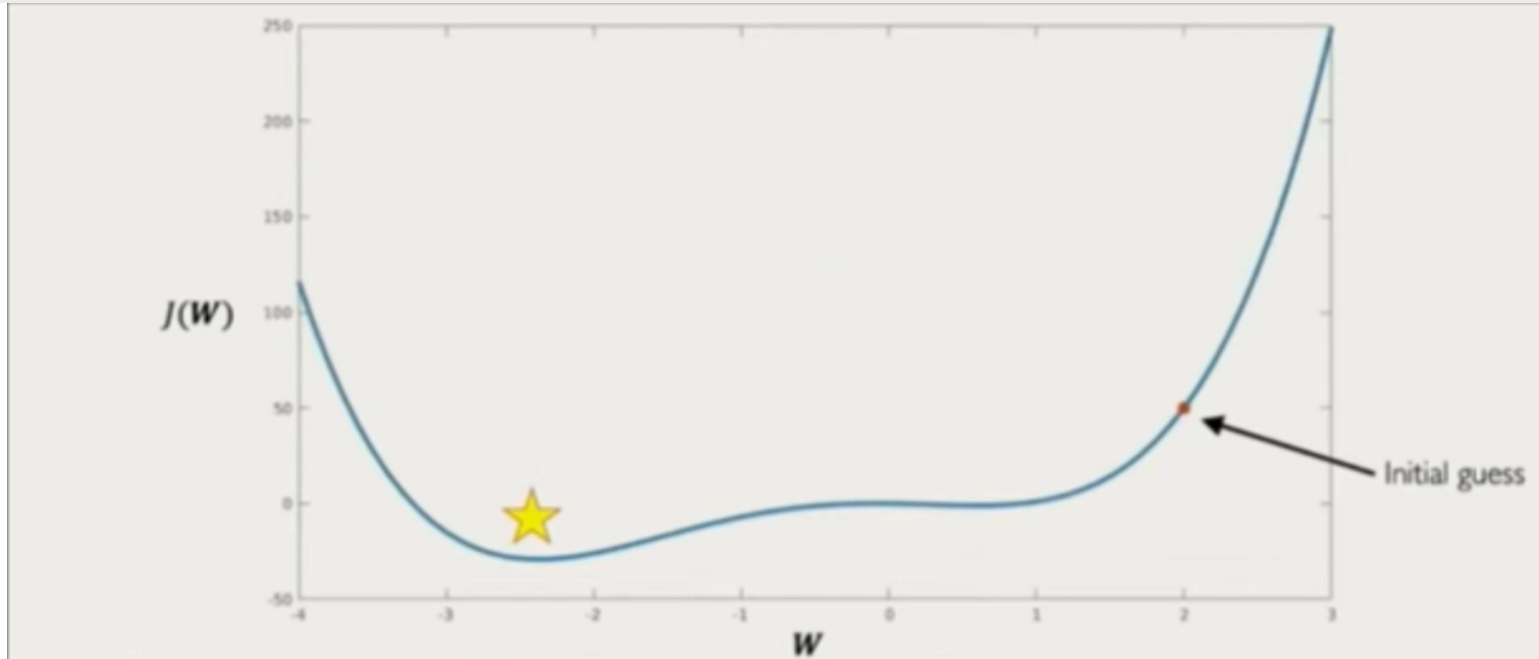
- $\mathbf{W}$ = ağırlıklar (model parametreleri)
- η (eta) = öğrenme oranı (learning rate)
- $J(\mathbf{W})$ = kayıp fonksiyonu
- $\partial J(\mathbf{W}) / \partial \mathbf{W}$ = kayıp fonksiyonunun türevi (gradyan)

Öğrenme Hızını Ayarlama

Düşük öğrenme oranı yavaş yakınsama gösterir ve yanlış yerel minimumlarda takılı kalır.

Yüksek öğrenme oranları aşırıya kaçır, kararsız hale gelir ve ıraksar.

Kararlı öğrenme oranları sorunsuz bir şekilde yakınsar ve yerel minimumlardan kaçınır.



Bununla nasıl başa çıkılır?

Fikir 1:

Farklı öğrenme hızlarını deneyin ve hangisinin "tam doğru" sonucu verdiğini görün.

Fikir 2:

Daha akıllıca bir şey yapın!

Ortama "uyum sağlayan" uyarlanabilir bir öğrenme hızı tasarlayın.

Uyarlanabilir Öğrenme Oranları

- Öğrenme oranları artık sabit değil
- Şu faktörlere bağlı olarak artırılabilir veya azaltılabilir:
 - Gradyanın büyüklüğü
 - Öğrenmenin ne kadar hızlı olduğu
 - Belirli ağırlıkların büyüklüğü
 - vb...

Gradyan inişi Algoritmaları

Algoritma	TF Uygulaması	PyTorch Uygulaması	Referans
• SGD	 <code>tf.keras.optimizers.SGD</code>	 <code>torch.optim.SGD</code>	Kiefer & Wolfowitz, 1952.
• Adam	 <code>tf.keras.optimizers.Adam</code>	 <code>torch.optim.Adam</code>	Kingma et al., 2014.
• Adadelta	 <code>tf.keras.optimizers.Adadelta</code>	 <code>torch.optim.Adadelta</code>	Zeiler et al., 2012.
• Adagrad	 <code>tf.keras.optimizers.Adagrad</code>	 <code>torch.optim.Adagrad</code>	Duchi et al., 2011.
• RMSProp	 <code>tf.keras.optimizers.RMSProp</code>	 <code>torch.optim.RMSProp</code>	Duchi et al., 2011.

Detaylar için: <http://runder.io/optimizing-gradient-descent/>

Hepsini Bir Araya Getirmek



```
import tensorflow as tf

model = tf.keras.Sequential([...])

# en sevdiğiniz optimizasyon algoritmasını seçin
optimizer = tf.keras.optimizers.SGD()

while True: # sonsuz döngü

    # ağ üzerinde ileri yayılım yap
    prediction = model(x)

    with tf.GradientTape() as tape:
        # kaybı hesapla
        loss = compute_loss(y, prediction)

    # ağırlıkları gradyanla güncelle
    grads = tape.gradient(loss, model.trainable_variables)
    optimizer.apply_gradients(zip(grads, model.trainable_variables))
```

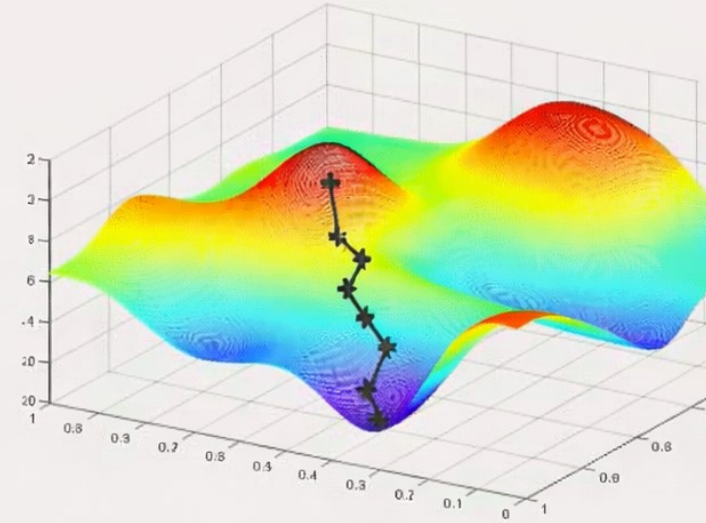
- `tf.keras.Sequential [...]` – Ağ
- `tf.keras.optimizers.SGD()` – Optimizasyon Algoritması
- `compute_loss(y, prediction)` – Kayıp Fonksiyonu
- `apply_gradients(zip(...))` – Gradyan Güncellemesi

Sinir Ağları Uygulamada: Mini-Batchlar

Gradyan İnişi

Algoritma

1. Ağırlıkları rastgele baslat $\sim \mathcal{N}(0, \sigma^2)$
2. Ta ki yakınsama olana kadar:
3. Gradyanı hesapla, $\frac{\partial J(W)}{\partial W}$
4. Ağırlıkları güncelle, $\mathbf{W} \leftarrow \mathbf{w} - \frac{\partial J(W)}{\partial W}$
5. Ağırlıkları döndür

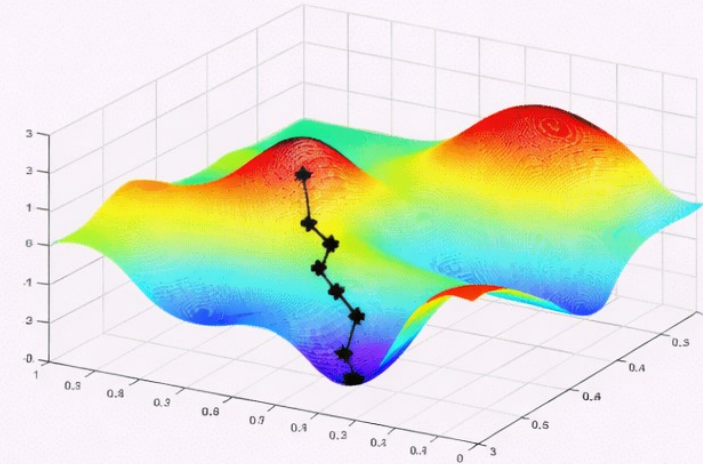


Cok hesaplama
yoğun olabilir!

Mini-Batch Gradyan İnişi

Algoritma

1. Ağırlıkları rastgele başlat $\sim \mathcal{N}(0, \sigma^2)$
2. Yakinsama sağlanana kadar döngü:
3. B veri noktası topluluğu seç
4. Gradyanı hesapla, $\frac{1}{B} \frac{\partial J(w)}{\partial W}$
5. Ağırlıkları güncelle, $W \leftarrow W - \eta \frac{\partial J(w)}{\partial W}$
6. Ağırlıkları döndür

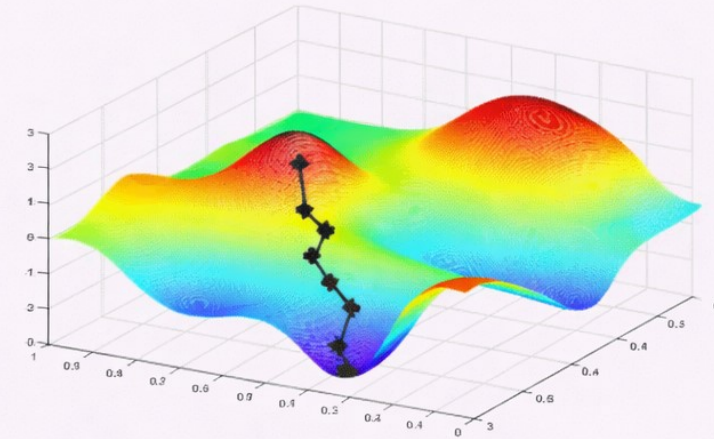


Hesaplama açısından
çok yoğun olabilir!

Stokastik Gradyan İnişi

Algoritma

1. Ağırlıkları rastgele başlat $\sim \mathcal{N}(0, \sigma^2)$
2. Yakınsama sağlanana kadar döngü:
3. Tek bir veri noktası i seç
4. Gradyanı hesapla, $\frac{\partial J(W)}{\partial W}$ ★
5. Ağırlıkları güncelle, $\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\partial J(W)}{\partial W}$
6. Ağırlıkları döndür

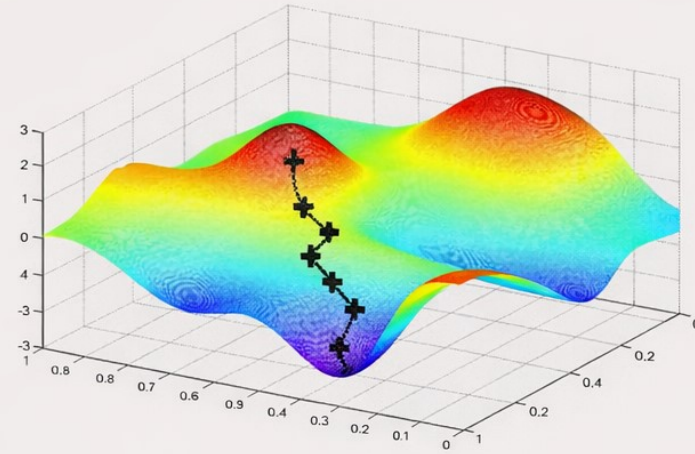


Hesaplama açısından
çok yoğun olabilir!

Stokastik Gradyan İnişi

Algoritma

1. Ağırlıkları rastgele başlat $\sim \mathcal{N}(0, \sigma^2)$
2. Ta ki yakınsama olana kadar:
3. B veri noktası seç
4. Gradyanı hesapla, $\frac{\partial J(\mathbf{W})}{\partial \mathbf{W}} = \frac{1}{B} \sum_{k=1}^B \frac{\partial J_k(\mathbf{W})}{\partial \mathbf{W}}$
5. Ağırlıkları güncelle, $\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\partial J(\mathbf{W})}{\partial \mathbf{W}}$
6. Ağırlıkları döndür



Mini-Batch'ler ile Eğitim

Gradyan'ı daha doğru tahmin etme

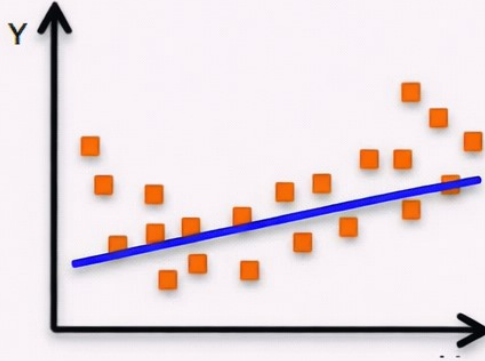
- Daha yumuşak yakınsama
- Daha büyük öğrenme hızlarını mümkün kılar

Mini-Batch'ler hızlı eğitime yol açar!

Hesaplama paralelleştirilebilir +
GPU'larda önemli hız artışlarına erişilebilir

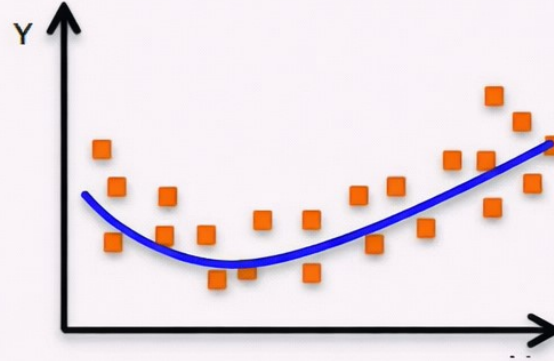
Pratikte sinir ağırları : Aşırı Öğrenme

Aşırı Öğrenme Problemi



**Yetersiz Öğrenme
(Eksik Öğrenme)**

Model, veriyi tam
öğrenebilecek kapasiteye
sahip değildir.



İdeal Uyum

Model çok karmaşıktır,
fazla parametre içerir
ve genelleme başarısı düşüktür



Aşırı Öğrenme

Model çok karmaşıktır,
fazla parametre içerir ve
genelleme başarısı düşüktür

Düzenleştirme

Nedir?

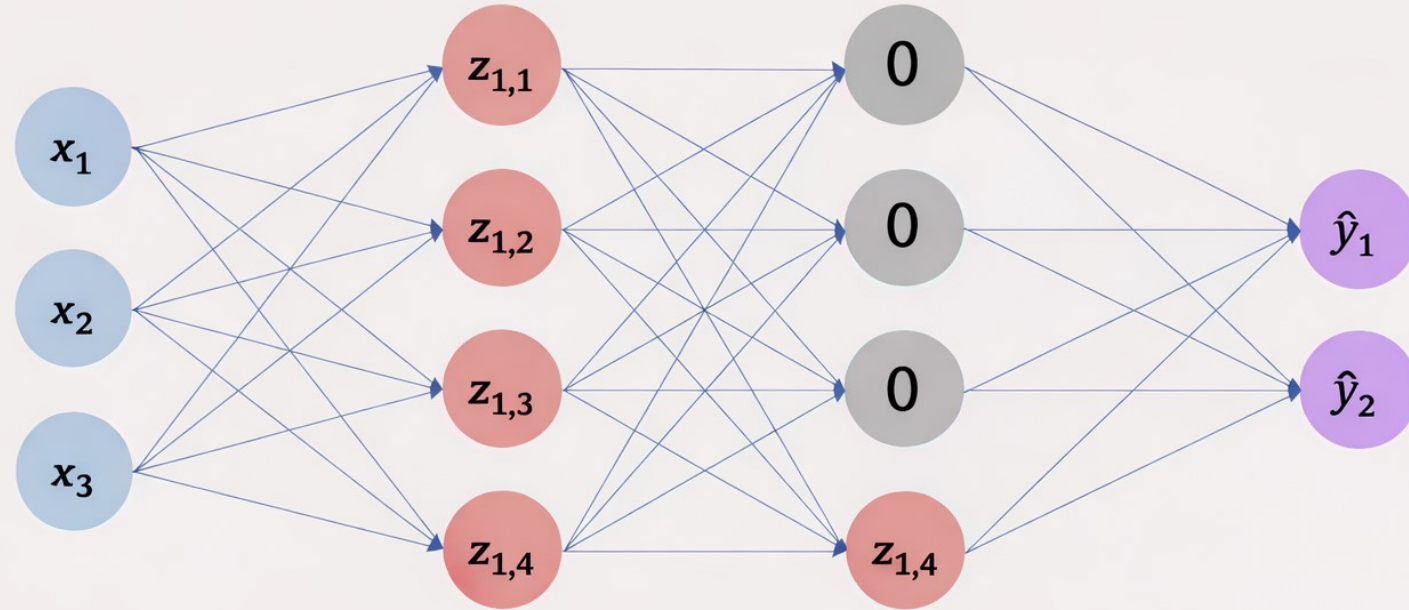
Optimizasyon problemini sınırlayarak
kompleks modelleri engelleyen bir yöntem

Neden ihtiyacımız var ?

Modelimizin genellenmesini daha önce karşılaşılmamış veri için daha iyi sağlıyor

Düzenleme I: Dropout

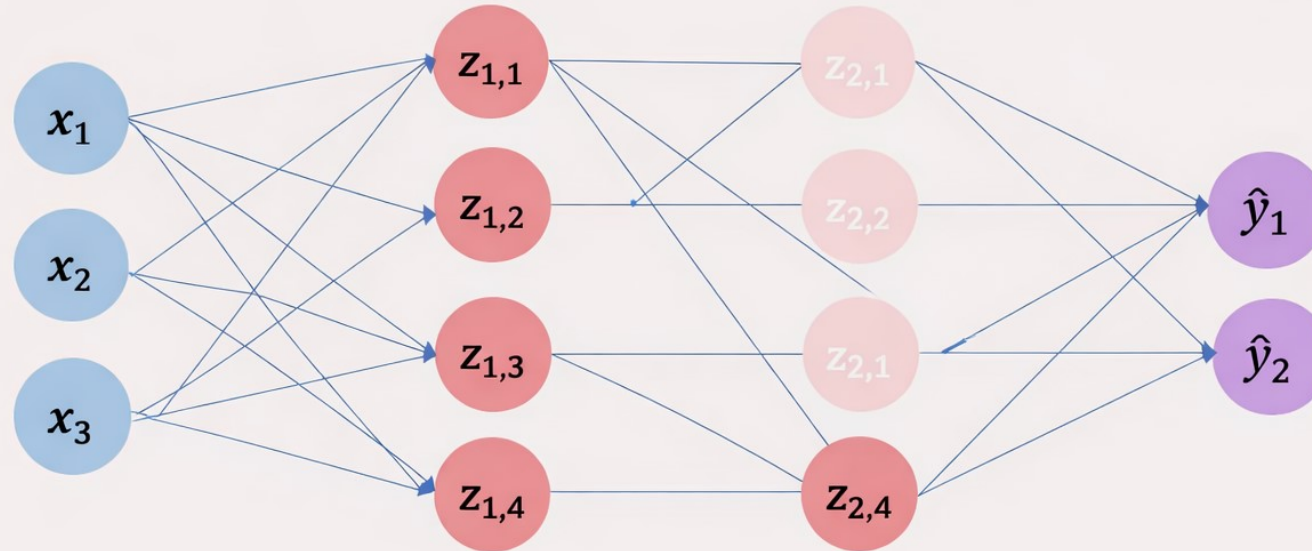
- Eğitim sırasında, rastgele bazı aktivasyonları 0 olarak ayarla



Düzenleme I: Dropout

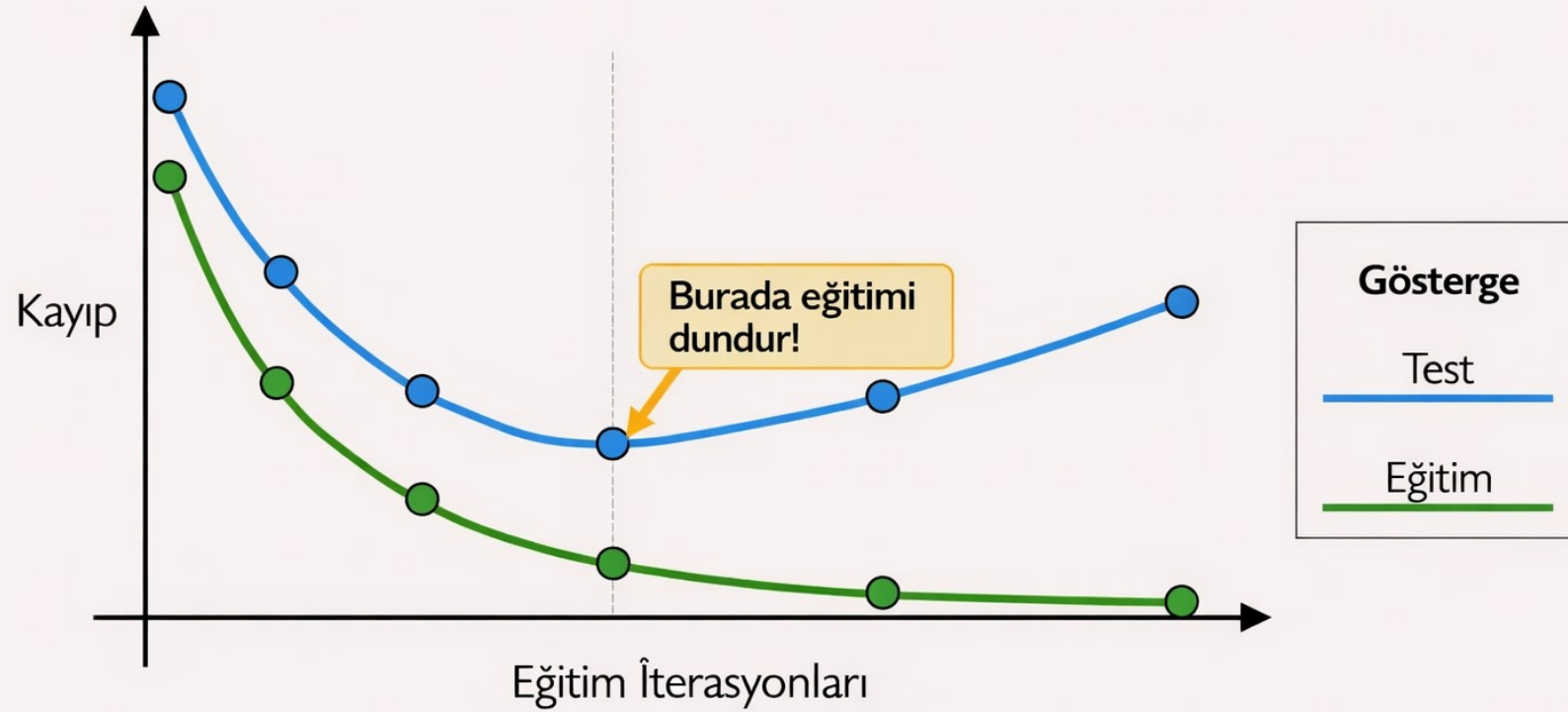
- Eğitim sırasında, rastgele bazı aktivasyonları 0'a ayarla
 - Genellikle katmandaki aktivasyonların %50'sini "düşür"
 - Ağın herhangi bir nörona aşırı bağımlı olmasını engeller

 `tf.keras.layers.Dropout(p=0.5)`
 `torch.nn.Dropout(p=0.5)`



Düzenleme 2: Erken Durdurma

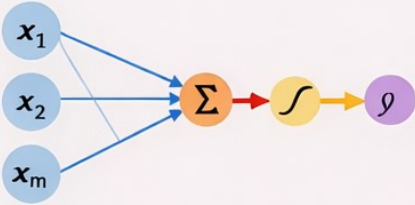
- Aşırı öğrenmeye fırsat vermeden eğitimi durdur



Temelde ne öğrendik

Algılayıcı

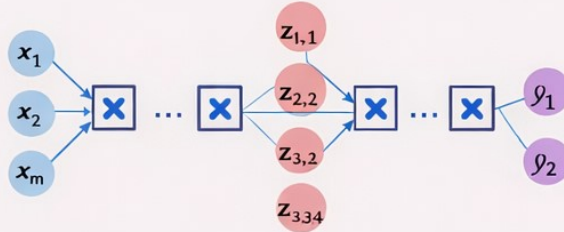
- Yapısal yapı taşları
- Doğrusal olmayan aktivasyon fonksiyonları



- Yapısal yapı taşları
- Doğrusal olmayan aktivasyon fonksiyonları

Yapay Sinir Ağları

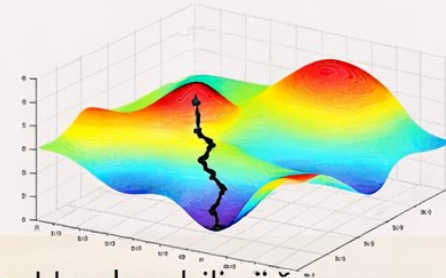
- Algılayıcıları yığılı sinir ağları oluşturma
- Geri yayılım ile optimizasyon



- Algılayıcıları yığılı sinir ağları oluşturma
- Geri yayılım ile optimizasyon

Uygulamada Eğitim

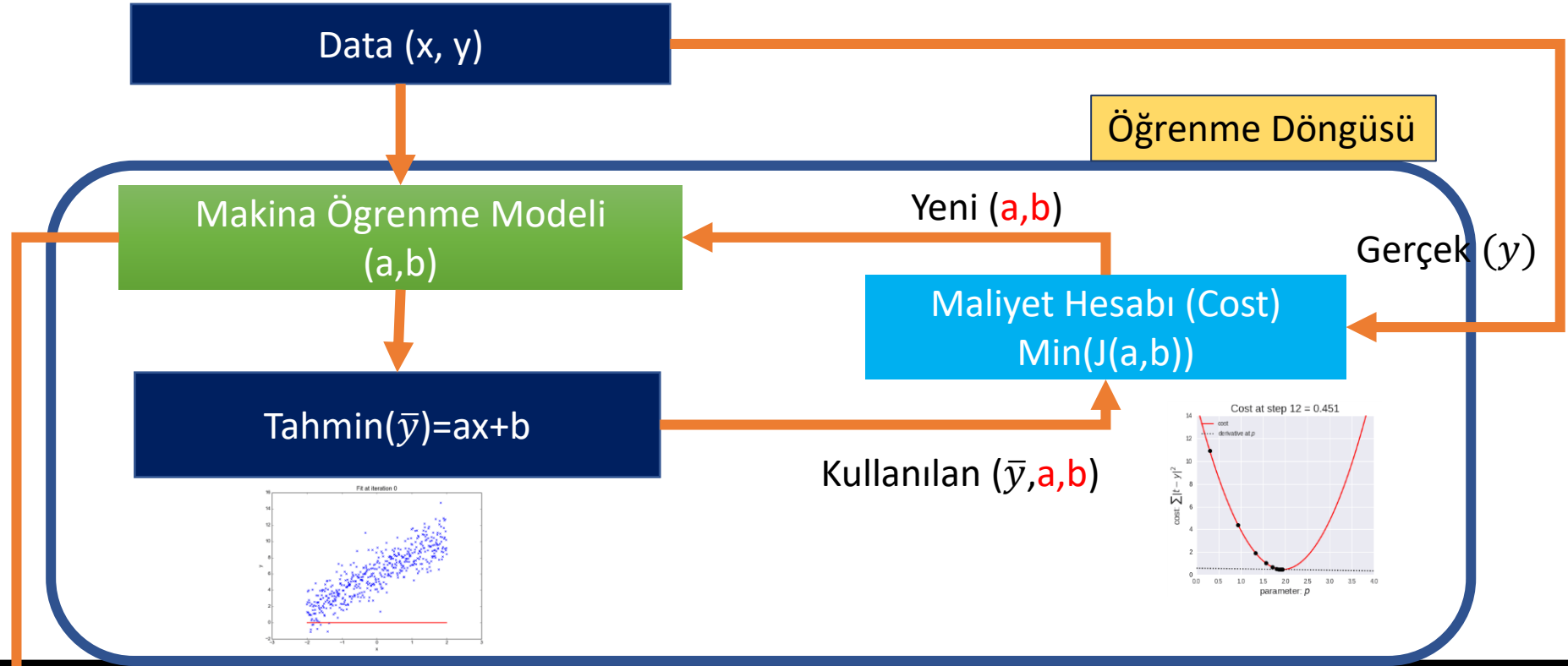
- Uyarlanabilir öğrenme
- Gruplama
- Düzenleme



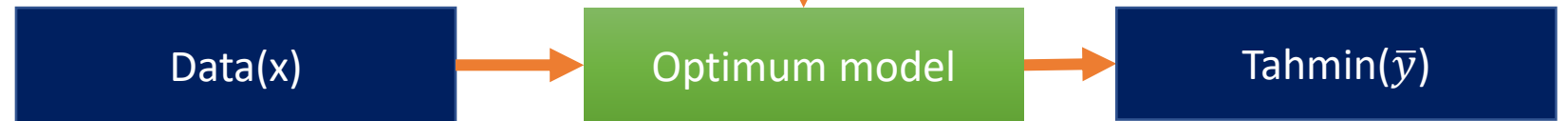
- Uyarlanabilir öğrenme
- Gruplama
- Düzenleme

EĞİTİM(Training)

Derin öğrenme ile eğitim ve tahmin.



TAHMİN(Prediction)



Gerekli Katsayılar(a,b)