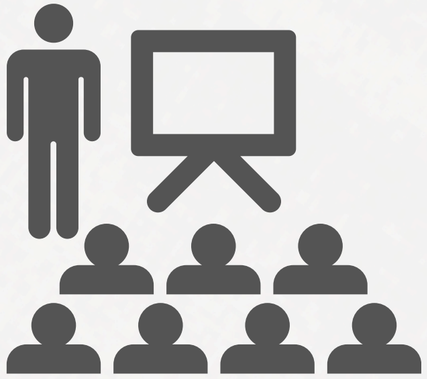
4.HAFTA

YAPAY SİNİR AĞLARI



DR. ÖĞR. ÜYESİ ALPER TALHA KARADENİZ

SAMSUN ÜNİVERSİTESİ

BACKPROPAGATİON ALGORİTMASI :

Backpropagation yapay sinir ağı modelinin eğitilmesi sırasında kullanılan bir optimizasyon algoritmasıdır.

Optimizasyon algoritmaları, modelin loss fonksiyonunu minize etmek veya en aza indirmek için modelin iç parametrelerini (weight, bias) nasıl güncelleyeceğini belirler.

Bizim en çok kullandığımız optimizasyon algoritması Stochastic Gradient Descent'tir (SGD). Bu basit optimizasyon algoritması her bir eğitim verisi noktası veya veri yığını (batch) gradyanları hesaplar ve parametreleri küçük adımlarla günceller. SGD, büyük veri kümeleri üzerinde çalışırken hızlı bir şekilde öğrenme sağlayabilir.

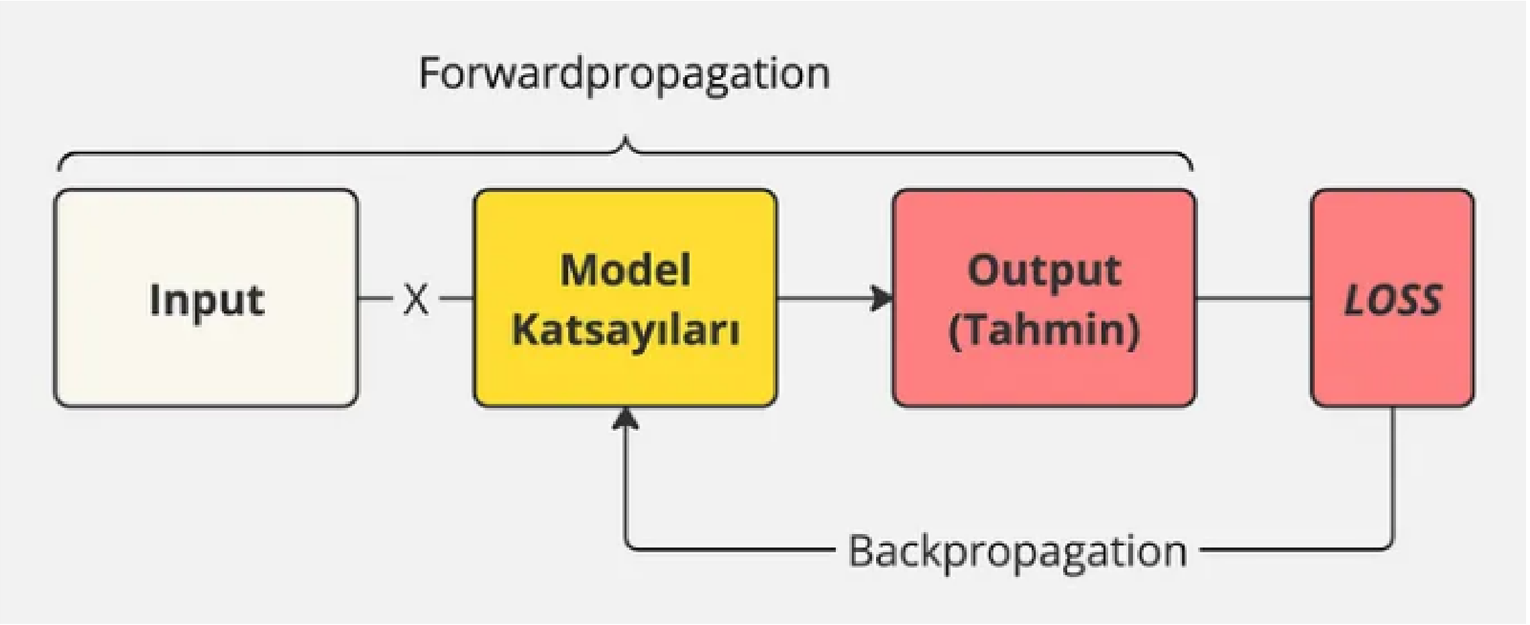
Backpropagation, bu hatanın ağı geriye doğru nasıl yayıldığını hesaplayarak çalışır.

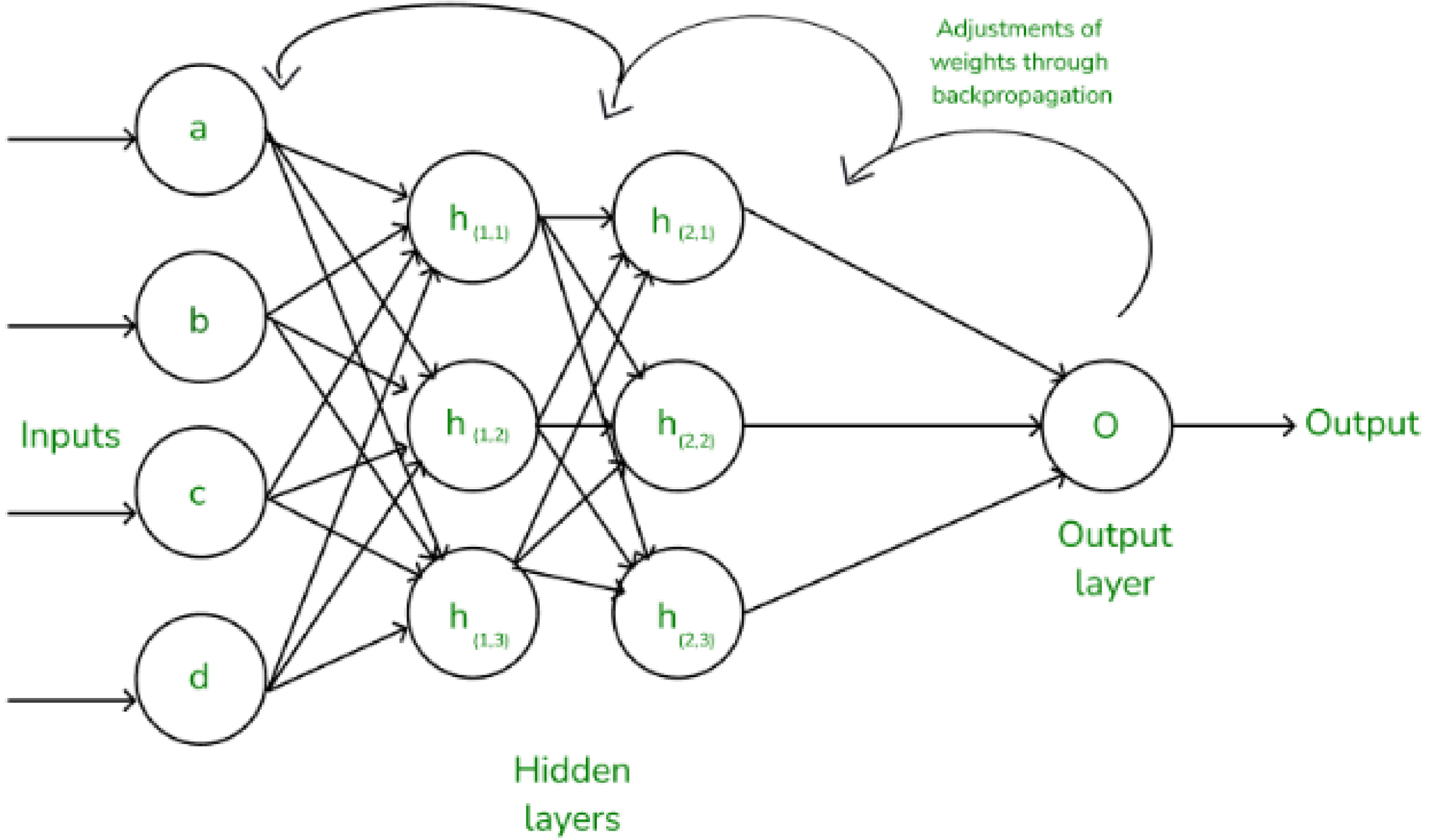
1. ilk olarak veri noktaları ağın girişine verilir ve ardından ağın son katmanındaki çıkış üretilir.
2. Çıkış ile gerçek değerler arasındaki uyumsuzluğu ölçen bir kayıp fonksiyonu hesaplanır. Bu kayıp fonksiyonu ağın tahminlerinin ne kadar yanlış olduğunu ölçer.
3. Ardından bu hata geriye doğru katmanlardan başlayarak ağın içineyayılır. Bu yayılma işlemi her katmandaki ağırlıkların ve biasların katkısını hesaplar.

4. Her katmandaki ağırlıkların ve biasların katkısı hesaplandıktan sonra, bu bilgiler kullanılarak ağırlıklar ve biaslar güncellenir. Genellikle bir optimizasyon algoritması ağırlıkları ve biasları güncellemek için kullanılır(SGD).

5. Bu süreç eğitim verilerinin tamamı veya bir veri yığını(batch) için tekrarlanır. Birden fazla yineleme(epoch) boyunca ağın ağırlıkları ve biasları eğitilir.

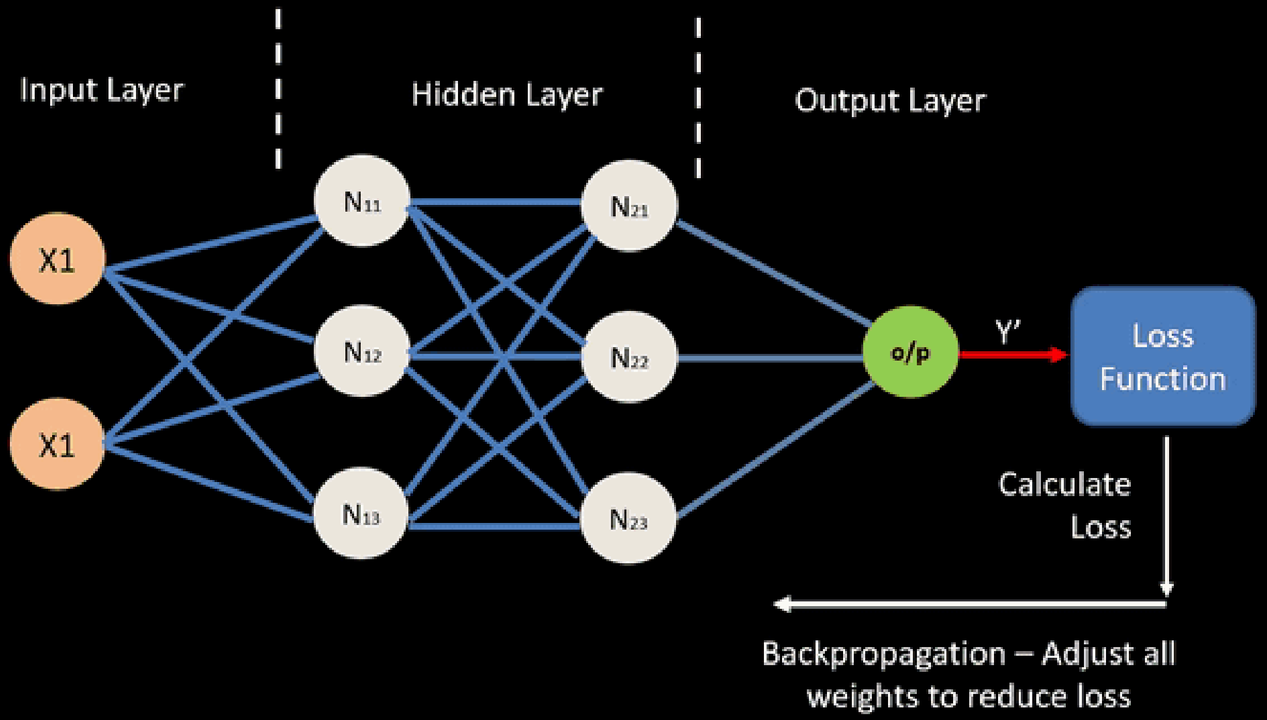
Backpropagation yapay sinir ağlarının temelini oluşturur. Ağın tahminlerini gerçek değerlere yaklaştırmak için her yinelemede ağırlıkların ve biasların güncellenmesi modelin daha iyi bir şekilde verileri öğrenmesini sağlar. Bu nedenle backpropagation derin öğrenme ve yapay sinir ağı modellerinin eğitimi için temel bileşendir.




Adjustments of
weights through
backpropagation
a
b
c
d
Inputs
h (1,1)
h (1,2)
h (1,3)
h (2,1)
h (2,2)
h (2,3)
Hidden
layers
O
Output
layer
Output

Algoritmanın Amacı :

- Sinir ağının çıktısı ile gerçek değer arasındaki farkı **ölçmek**
- Bu hatanın, her bir ağırlık ve bias üzerinde oluşturduğu etkiyi **hesaplamak**
- Ağırlıkları ve bias değerlerini **güncellemek**
- Hatanın tekrar tekrar azalmasını sağlayarak, ağın doğru tahminler yapmasına katkıda bulunmak



Geri Yayılım (Backpropagation) Algoritmasının Çalışma Mantığı

1. Giriş

Yapay sinir ağları, girdi verilerini işleyerek çıktı üretir. Ancak üretilen çıktı çoğu zaman gerçek değerle tam olarak örtüşmez. Bu durumda, hatanın nasıl azaltılacağı sorusu ortaya çıkar. Geri yayılım algoritması, hatanın ağ boyunca geriye doğru izlenmesini ve her parametrenin bu hataya katkısının hesaplanmasını sağlar. Böylece ağ, ağırlıklarını doğru yönde güncelleyerek öğrenme sürecini gerçekleştirir.

2. İleri Yayılım (Forward Propagation)

Girdi verisi, ağdaki katmanlardan geçirilir.

- Her nöron ağırlıklı toplam alır:

  $$z = \sum(w \cdot x) + b$$

- Aktivasyon fonksiyonu uygulanır:

  $$a = f(z)$$

- Son katmanda tahmin ($\hat{y}$) elde edilir.

Bu aşama yalnızca ağın tahmin üretmesini sağlar.

3. Hata Hesaplama

Tahmin ($\hat{y}$) ile gerçek değer (y) arasındaki fark hata fonksiyonu (loss function) ile ölçülür.

Örnek: Ortalama Kare Hata (MSE)

$$L = \frac{1}{2}(y - \hat{y})^2$$

Amaç, bu hata fonksiyonunu minimuma indirmektir.

4. Geri Yayılım

Hata, çıkış katmanından başlayarak girişe doğru aktarılır. Bunun için türev alma ve zincir kuralı kullanılır.

1. **Çıkış katmanındaki hata:**
   $\delta^{(L)} = (\hat{y} - y) \cdot f'(z^{(L)})$
2. **Gizli katmanlardaki hata:**
   $\delta^{(l)} = (W^{(l+1)})^T \delta^{(l+1)} \cdot f'(z^{(l)})$

Her katmanın hatası, kendisinden sonraki katmanın hatası ile ilişkilidir.

5. Gradyan Hesaplama

Her bir parametrenin hataya katkısı bulunur.

- Ağırlıklar için:
  $$\frac{\partial L}{\partial W^{(l)}} = a^{(l-1)} \cdot (\delta^{(l)})^T$$
- Biaslar için:
  $$\frac{\partial L}{\partial b^{(l)}} = \delta^{(l)}$$

6. Parametre Güncelleme

Ağırlıklar ve biaslar gradyanlar kullanılarak güncellenir.

- Ağırlık:
  $W^{(l)} := W^{(l)} - \eta \cdot \frac{\partial L}{\partial W^{(l)}}$
- Bias:
  $b^{(l)} := b^{(l)} - \eta \cdot \frac{\partial L}{\partial b^{(l)}}$

Burada η öğrenme oranıdır.

7. Tekrarlama

İleri yayılım, hata hesaplama, geri yayılım ve ağırlık güncelleme adımları defalarca tekrarlanır. Her tekrarda hata biraz daha azalır. Ağ, zamanla doğru tahminler yapmayı öğrenir.

Hata Fonksiyonları (Loss Functions)

Bir yapay sinir ağı eğitilirken, modelin tahminleri ile gerçek değerler arasındaki farkın nicel olarak ölçülmesine **hata fonksiyonu** denir.

Hata fonksiyonu, modelin öğrenme sürecinin merkezinde yer alır. Modelin ağırlıkları bu hata fonksiyonunu minimize edecek şekilde güncellenir.

- **MSE**, sayısal tahminlerde hatayı ölçer, büyük hataları öne çıkarır.
- **Cross-Entropy**, sınıflandırmada modelin verdiği olasılıkların doğruluğunu ölçer; yanlış ama emin tahminlerde güçlü gradyan sağlar.

1. Ortalama Kare Hatası (Mean Squared Error – MSE)

Tanım

MSE, genellikle **regresyon problemlerinde** kullanılır. Modelin tahmini değer ile gerçek değer arasındaki farkın karesinin ortalamasını hesaplar.

Matematiksel olarak ifade edilecek olursa:

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

- N: Örnek sayısı
- y_i: Gerçek değer
- $\hat{y}_i$: Modelin tahmini

> Kare alma işlemi, hem pozitif hem de negatif farkların eşit şekilde değerlendirilmesini sağlar ve büyük hataları daha fazla cezalandırır.

Olasılıksal Yorumu

MSE, hataların normal dağıldığı (Gauss dağılımı) varsayımına dayanmaktadır. Maksimum olasılık tahmini (MLE) ile minimize edilebilir. Bu nedenle regresyon problemleri için doğal bir seçimdir.

Avantajları

- Hesaplaması kolaydır ve lineer regresyonda konveks bir kayıp fonksiyonu sağlar.
- Büyük hataları öncelikli olarak düzeltir.

Dezavantajları

- Sınıflandırma problemlerinde MSE kullanıldığında, gradyan "doygunluk" bölgelerinde çok küçük olur. Bu durum, ağırlık güncellemelerinin yavaş gerçekleşmesine sebep olur.

Örnek

Gerçek değerler: [100, 150, 200]

Tahminler: [110, 140, 195]

1. Farklar: [-10, 10, 5]
2. Kareleri: [100, 100, 25]
3. Ortalama: $MSE = \frac{100+100+25}{3} = 75$

2. Çapraz Entropi (Cross-Entropy Loss)

Tanım

Çapraz entropi, genellikle **sınıflandırma problemlerinde** kullanılır. Modelin tahmin ettiği olasılık dağılımı ile gerçek dağılım arasındaki farklı ölçer.

İkili sınıflandırma için formül:

$$CE = -\frac{1}{N}\sum_{i=1}^{N}\left[y_i \cdot \log(\hat{y}_i) + (1-y_i) \cdot \log(1-\hat{y}_i)\right]$$

- y_i = gerçek etiket (0 veya 1)
- $\hat{y}_i$ = modelin tahmini olasılık (0 ile 1 arası)

Çok sınıflı sınıflandırma için (Softmax çıktısı):

$$CE = -\sum_{k=1}^{K} y_k \log(p_k)$$

- K = sınıf sayısı
- p_k = modelin tahmini olasılık
- y_k = gerçek sınıf dağılımı (one-hot)

Özellikleri

- Model, doğru sınıfa yüksek olasılık verdikçe kayıp azalır.
- Yanlış ama emin tahminlerde kayıp artar.

Avantajları

- Sınıflandırma problemleri için doğrudan uygundur.
- Yanlış tahminlerde gradyan büyük olduğu için öğrenme hızlıdır.

Örnek (Binary)

Gerçek etiket: Spam = 1

- Model tahmini: 0.9 → $CE = -\log(0.9) \approx 0.105$ (küçük hata)
- Model tahmini: 0.1 → $CE = -\log(0.1) \approx 2.302$ (büyük hata)

Örnek (Multiclass)

Sınıflar: Kedi, Köpek, Kuş

- Gerçek sınıf: Köpek (etiket: [0,1,0])
- Tahmin: [0.2, 0.7, 0.1]

$$CE = -(0 \cdot \log 0.2 + 1 \cdot \log 0.7 + 0 \cdot \log 0.1) = -\log 0.7 \approx 0.357$$

KIYASLAMA:

Özellik	MSE	Cross-Entropy
Kullanım Alanı	Regresyon	Sınıflandırma
Ölçülen	Tahmin hatası (sayı)	Olasılık farkı
Büyük hataya duyarlılık	Yüksek	Orta/Gradyan bazlı
Gradyan problemi	Doygunlukta küçük	Yanlış tahminde büyük
Olasılıksal yorum	Gaussian varsayımı	KL-divergence (bilgi kuramı)

Örnek:

Bir teknoloji şirketi, akıllı bir sıcaklık tahmin cihazı ve "konforlu oda" sınıflandırma sistemi geliştirmiştir.

Veriler:

Gün	Gerçek Sıcaklık (°C)	Tahmin Sıcaklık (°C)	Gerçek Konforlu Oda (1=evet, 0=hayır)	Tahmin Konfor Olasılığı
Pazartesi	22	20	1	0.7
Salı	25	28	0	0.4
Çarşamba	20	18	1	0.9
Perşembe	23	21	0	0.2

1. Bu veriler için **MSE'yi** hesaplayınız (sıcaklık tahminleri için).
2. Bu veriler için **Binary Cross-Entropy kaybını** hesaplayınız (konfor sınıflandırması için).
3. Sonuçları yorumlayınız: Hangi günlerde model daha iyi veya daha kötü tahmin yapmış?

MSE Hesabı (Sıcaklık Tahminleri)

Adım 1: Hataları hesapla

Hata = Tahmin Sıcaklık - Gerçek Sıcaklık

Gün	Tahmin - Gerçek (Hata)
Pazartesi	20 - 22 = -2
Salı	28 - 25 = 3
Çarşamba	18 - 20 = -2
Perşembe	21 - 23 = -2

Adım 2: Hataların karelerini al

Gün	Hata²
Pazartesi	$(-2)^2 = 4$
Salı	$3^2 = 9$
Çarşamba	$(-2)^2 = 4$
Perşembe	$(-2)^2 = 4$

Adım 3: MSE'yi hesapla

Ortalama = (4 + 9 + 4 + 4) / 4 = 21 / 4 = 5.25

MSE = 5.25

Binary Cross-Entropy Hesabı (Konfor Tahmini)

Adım 1: Her gün için CE hesapla

Formül: CE = -(y * log(tahmin) + (1-y) * log(1-tahmin))

Gün	CE Hesabı	Sonuç
Pazartesi	-(1*log(0.7)* + *0*log(0.3))	0.357
Salı	-(*0*log(0.4)* + *1*log(0.6))	0.511
Çarşamba	-(1*log(0.9)* + *0*log(0.1))	0.105
Perşembe	-(*0*log(0.2)* + *1*log(0.8))	0.223

Adım 2: Ortalama CE'yi hesapla

Ortalama = (0.357 + 0.511 + 0.105 + 0.223) / 4 ≈ 0.299

Cross-Entropy ≈ 0.299

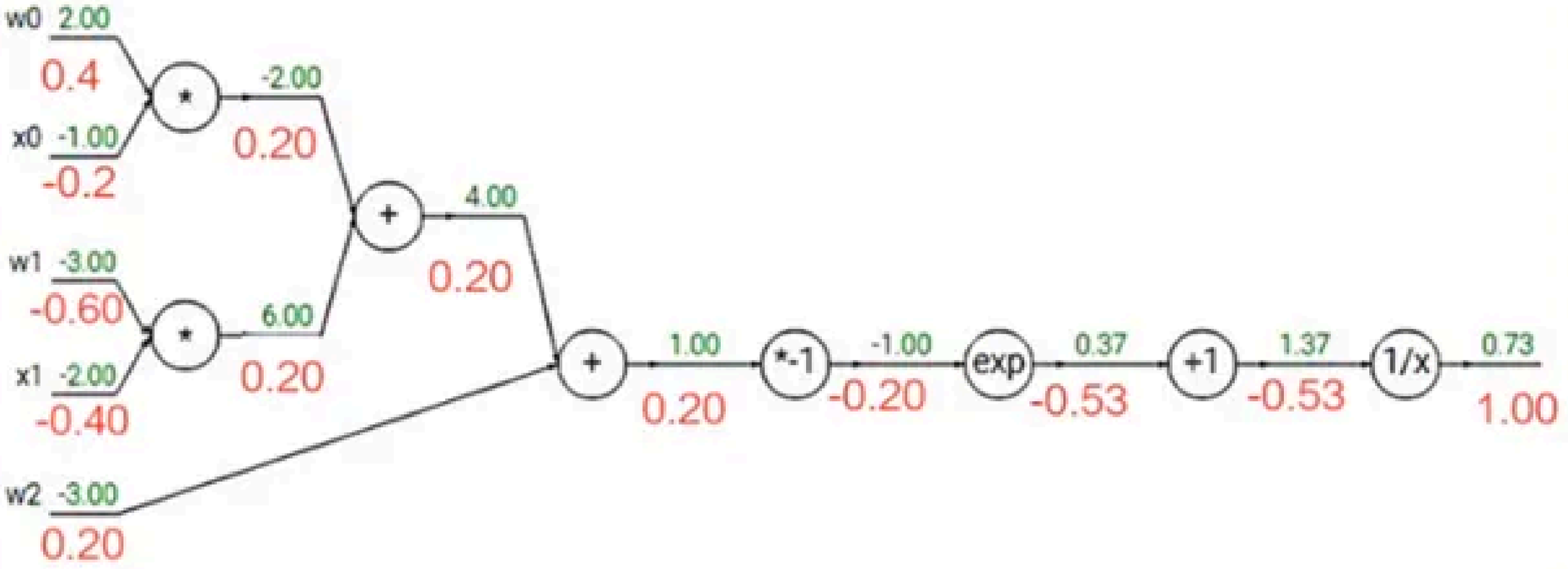
Adım 3: Yorumlama

- MSE 5.25 → Sıcaklık tahminlerinde ortalama hata 5.25 °C². En büyük hata Salı günü.
- Cross-Entropy 0.299 → Konfor tahminlerinde genel başarı yüksek. En büyük hata Salı, en düşük hata Çarşamba günü.
- Salı günü hem sıcaklık hem de konfor tahmini modelin düzeltmeye ihtiyaç duyduğu gün olarak öne çıkıyor.

ZINCIR KURALI VE TÜREV:

Zincir Kuralının Mantığı

- Yapay sinir ağları birden fazla katmandan oluşur.
- Her nöron bir önceki katmandan aldığı veriyi işler, ağırlık ve bias ile toplar ve aktivasyon fonksiyonundan geçirir.
- Ağın hatasını (loss) minimize etmek için ağırlık ve bias değişimlerini bilmemiz gerekir.
- Ancak, ağırlıklar hatayı dolaylı etkiler: ağırlık → nöron girişi → aktivasyon → hata.
- İşte bu nedenle **zincir kuralı** kullanılır: hatanın ağırlık ve bias üzerindeki etkisi, bu katmanlar arasındaki türevlerin çarpımı ile hesaplanır.



$$f(x) = e^x \rightarrow \frac{df}{dx} = e^x$$

$$f(x) = \frac{1}{x} \rightarrow \frac{df}{dx} = -\frac{1}{x^2}$$

$$f_a(x) = ax \rightarrow \frac{df}{dx} = a$$

$$f_c(x) = c + x \rightarrow \frac{df}{dx} = 1$$

Formül:

$$\frac{dL}{dw} = \frac{dL}{da} \cdot \frac{da}{dz} \cdot \frac{dz}{dw}$$

- L = hata fonksiyonu
- a = nöron çıkışı (aktivasyon)
- z = lineer kombinasyon (weighted sum)

Hata Fonksiyonu Türevi

- MSE (Mean Squared Error) örneği:

$$L = \frac{1}{2}(a - y)^2$$

- Türevi:

$$\frac{\partial L}{\partial a} = a - y$$

Açıklama:

- Tahminimiz a ile gerçek değer y arasındaki farkı buluyoruz.
- Negatifse tahmin küçük, pozitifse tahmin büyük.
- Bu değer, ağırlık ve bias güncelleme yönünü belirler.

Aktivasyon Fonksiyonunun Türevi

- Sigmoid örneği:

$$a = \sigma(z) \quad , \quad \frac{\partial a}{\partial z} = \sigma(z)(1 - \sigma(z))$$

Açıklama:

- Aktivasyon fonksiyonu, nöron çıktısını doğrusal olmayan bir forma dönüştürür.
- Türev, nöron çıktısının girişe duyarlılığını gösterir.
- Çıktı 0 veya 1'e yakınsa, türev küçük → öğrenme yavaş. Orta değerlerde türev büyük → öğrenme hızlı.

Lineer Kombinasyon Türevi

- Nöron girişinin ağırlığa göre türevi:

$$z = w \cdot x + b$$

$$\frac{\partial z}{\partial w} = x \quad , \quad \frac{\partial z}{\partial b} = 1$$

Açıklama:

- Ağırlık değişirse lineer kombinasyon ne kadar değişir? → x
- Bias değişirse lineer kombinasyon ne kadar değişir? → 1

Ağırlık ve Bias Türevi (Zincir Kuralı Uygulaması)

$$\frac{\partial L}{\partial w} = (a - y) \cdot \sigma(z)(1 - \sigma(z)) \cdot x$$

$$\frac{\partial L}{\partial b} = (a - y) \cdot \sigma(z)(1 - \sigma(z)) \cdot 1$$

Açıklama:

- Hata → aktivasyon → lineer kombinasyon zinciri ile ağırlığın hataya etkisini bulduk.
- Negatif türev → ağırlığı artır, pozitif türev → ağırlığı azalt.
- Bias için de aynı mantık geçerli.

Ağırlık ve Bias Güncelleme

$$w_{\text{yeni}} = w - \eta \frac{\partial L}{\partial w} \quad , \quad b_{\text{yeni}} = b - \eta \frac{\partial L}{\partial b}$$

- η = öğrenme oranı
- Güncelleme, hatayı azaltacak yönde yapılır.

Açıklama:

- Küçük öğrenme oranı → yavaş ama güvenli öğrenme
- Büyük öğrenme oranı → hızlı ama dalgalı öğrenme

Çok Katmanlı Ağ Mantığı

- Çıkış katmanı hatayı hesaplar.
- Gizli katmanların ağırlıkları doğrudan hatayı görmez, sadece sonraki katmana etkisi üzerinden türev alır.
- Zincir kuralı ile dıştan içe doğru türevler çarpılır:

$$\frac{\partial L}{\partial w_{hidden}} = \frac{\partial L}{\partial a_{out}} \cdot \frac{\partial a_{out}}{\partial z_{out}} \cdot \frac{\partial z_{out}}{\partial a_{hidden}} \cdot \frac{\partial a_{hidden}}{\partial z_{hidden}} \cdot \frac{\partial z_{hidden}}{\partial w_{hidden}}$$

- Bu sayede tüm ağırlıklar ve biaslar, hatayı minimize edecek şekilde güncellenir.

Sayısal Örnek

- Giriş $x = 0.5$, ağırlık $w = 0.8$, bias $b = 0.1$, gerçek değer $y = 1$
- Lineer kombinasyon: $z = 0.8 * 0.5 + 0.1 = 0.5$
- Aktivasyon (sigmoid): $a = \sigma(0.5) \approx 0.6225$
- Hata türevi: $\frac{\partial L}{\partial a} = 0.6225 - 1 = -0.3775$
- Aktivasyon türevi: $\frac{\partial a}{\partial z} = 0.6225 * (1 - 0.6225) \approx 0.2350$
- Lineer kombinasyon türevi: $\frac{\partial z}{\partial w} = x = 0.5$
- Ağırlık türevi: $\frac{\partial L}{\partial w} = -0.3775 * 0.2350 * 0.5 \approx -0.0443$
- Bias türevi: $\frac{\partial L}{\partial b} = -0.3775 * 0.2350 * 1 \approx -0.0887$

Güncelleme ($\eta = 0.1$):

$$w_{\text{yeni}} = 0.8 - 0.1 * (-0.0443) \approx 0.8044$$

$$b_{\text{yeni}} = 0.1 - 0.1 * (-0.0887) \approx 0.1089$$

Örnek :

Ağ Yapısı

- Giriş katmanı: 1 nöron x
- Gizli katman: 1 nöron h
- Çıkış katmanı: 1 nöron $\hat{y}$
- Aktivasyon fonksiyonu: Sigmoid
- Hata fonksiyonu: MSE

Başlangıç değerleri:

Parametre	Değer
x	0.5
w1 (giriş→gizli)	0.4
b1	0.1
w2 (gizli→çıkış)	0.3
b2	0.2
y (gerçek)	1

Adım 1: İleri Yayılım (Forward Pass)

Amaç: Ağın tahminini bulmak ve hata hesaplamak.

1. Gizli katman lineer kombinasyon:

$$z_1 = w1 \cdot x + b1 = 0.4 * 0.5 + 0.1 = 0.3$$

2. Gizli katman aktivasyonu (sigmoid):

$$h = \sigma(z_1) = \frac{1}{1 + e^{-0.3}} \approx 0.574$$

3. Çıkış katmanı lineer kombinasyon:

$$z_2 = w2 \cdot h + b2 = 0.3 * 0.574 + 0.2 \approx 0.372$$

4. Çıkış katmanı aktivasyonu:

$$\hat{y} = \sigma(z_2) = \frac{1}{1 + e^{-0.372}} \approx 0.592$$

5. Hata (MSE):

$$L = \frac{1}{2}(\hat{y} - y)^2 = \frac{1}{2}(0.592 - 1)^2 \approx 0.0832$$

Açıklama: Bu adımda ağın tahmini 0.592 çıktı, gerçek değer 1. Hata 0.0832. Hata, geri yayılım ile ağırlıklara iletilecek.

Adım 2: Çıkış Katmanı Hata Türevi

1. Hata fonksiyonu türevi çıkışa göre:

$$\frac{\partial L}{\partial \hat{y}} = \hat{y} - y = 0.592 - 1 = -0.408$$

Açıklama: Tahmin gerçek değerden küçük olduğu için negatif çıktı. Bu, ağırlığın artması gerektiğini gösteriyor.

2. Sigmoid türevi:

$$\frac{\partial \hat{y}}{\partial z_2} = \hat{y}(1 - \hat{y}) = 0.592 * (1 - 0.592) \approx 0.242$$

3. Zincir kuralı ile çıkış katmanı hata sinyali:

$$\delta_2 = \frac{\partial L}{\partial z_2} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z_2} = -0.408 * 0.242 \approx -0.099$$

Açıklama: Hata sinyali, ağırlık güncellemesi için çıkış katmanına göre hazır.

Adım 3: Çıkış Katmanı Ağırlık ve Bias Türevleri

- Ağırlık w2 türevi:

$$\frac{\partial L}{\partial w2} = \delta_2 \cdot h = -0.099 * 0.574 \approx -0.057$$

- Bias b2 türevi:

$$\frac{\partial L}{\partial b2} = \delta_2 * 1 = -0.099$$

Açıklama: Hata, gizli katmandaki aktivasyona ve çıkış katmanı hatasına bağlı. Zincir kuralı ile türevleri çarpıyoruz.

Adım 4: Gizli Katman Hata Türevleri

1. Gizli katman hata sinyali:

$$\delta_1 = \delta_2 \cdot w2 \cdot h * (1 - h) = -0.099 * 0.3 * 0.574 * (1 - 0.574)$$

- Ara hesap: $0.574 * (1 - 0.574) = 0.574 * 0.426 \approx 0.244$
- $\delta_1 = -0.099 * 0.3 * 0.244 \approx -0.0072$

2. Gizli katman ağırlık ve bias türevleri:

$$\frac{\partial L}{\partial w1} = \delta_1 \cdot x = -0.0072 * 0.5 \approx -0.0036$$

$$\frac{\partial L}{\partial b1} = \delta_1 * 1 = -0.0072$$

Açıklama: Hata, zincir kuralı ile gizli katmana geri yayılır. Çıkış katmanı hatası, ağırlık ve aktivasyon türevleriyle çarpılarak gizli katmandaki türev elde edilir.

Adım 5: Ağırlık ve Bias Güncelleme

Öğrenme oranı $\eta = 0.1$

Parametre	Yeni Değer
w2	0.3 - 0.1*(-0.057) = 0.3057
b2	0.2 - 0.1*(-0.099) = 0.2099
w1	0.4 - 0.1*(-0.0036) = 0.40036
b1	0.1 - 0.1*(-0.0072) = 0.10072

Açıklama:

- Çıkış katmanı hatası daha büyük olduğu için w2 ve b2 daha fazla güncellendi.
- Gizli katman hatası daha küçük, bu nedenle w1 ve b1 çok az değişti.
- Zincir kuralı sayesinde hata, katmanlar arasında doğru şekilde dağıtıldı.

Geri Yayılım ile Ağırlık Güncelleme :

Ağ Yapısı

- **Giriş katmanı:** 2 nöron (x1, x2)
- **Çıkış katmanı:** 1 nöron (ŷ)
- Aktivasyon: Sigmoid
- Hata fonksiyonu: MSE

Başlangıç değerleri:

Parametre	Değer
x1	0.6
x2	0.1
w1 (x1→y)	0.5
w2 (x2→y)	0.4
b	0.2
y (gerçek)	1
Öğrenme oranı η	0.1

Açıklama: Bu örnekte sadece tek katmanlı bir ağ var. Ama iki girdi olduğu için zincir kuralı ile ağırlıkların nasıl güncellendiğini görebiliriz.

Adım 1: İleri Yayılım (Tahmin ve Hata)

1. Lineer kombinasyon:

$$z = w1 * x1 + w2 * x2 + b = 0.5 * 0.6 + 0.4 * 0.1 + 0.2 = 0.5$$

2. Sigmoid aktivasyonu:

$$\hat{y} = \sigma(z) = \frac{1}{1 + e^{-0.5}} \approx 0.622$$

3. Hata (MSE):

$$L = \frac{1}{2}(\hat{y} - y)^2 = \frac{1}{2}(0.622 - 1)^2 \approx 0.071$$

Açıklama: Tahmin 0.622, gerçek değer 1. Ağın hatası 0.071. Bu hatayı ağırlıklara iletip güncelleyeceğiz.

Adım 2: Çıkış Katmanı Hata Sinyali

1. Hata fonksiyonu türevi:

$$\frac{\partial L}{\partial \hat{y}} = \hat{y} - y = 0.622 - 1 = -0.378$$

2. Sigmoid türevi:

$$\frac{\partial \hat{y}}{\partial z} = \hat{y} * (1 - \hat{y}) = 0.622 * (1 - 0.622) \approx 0.235$$

3. Zincir kuralı ile hata sinyali:

$$\delta = \frac{\partial L}{\partial z} = \frac{\partial L}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial z} = -0.378 * 0.235 \approx -0.0888$$

Açıklama: Bu değer, **çıktının lineer kombinasyonuna göre hatanın büyüklüğünü ve yönünü** gösterir. Negatif → ağırlık artırılmalı, pozitif → azaltılmalı.

Adım 3: Ağırlık ve Bias Türevleri

- w1 türevi:

$$\frac{\partial L}{\partial w1} = \delta * x1 = -0.0888 * 0.6 \approx -0.0533$$

- w2 türevi:

$$\frac{\partial L}{\partial w2} = \delta * x2 = -0.0888 * 0.1 \approx -0.0089$$

- Bias türevi:

$$\frac{\partial L}{\partial b} = \delta * 1 = -0.0888$$

Açıklama:

- Zincir kuralı ile ağırlığın her girdi üzerindeki etkisini bulduk.
- x1 daha büyük olduğundan w1 türevi büyük → ağırlık daha çok güncellenecek.
- x2 küçük → w2 az değişecek.
- Bias, tüm nöronlar için sabit etki gösterdiği için kendi türevi δ ile aynı.

Adım 4: Ağırlık ve Bias Güncelleme

Öğrenme oranı: η = 0.1

$$w1_{\text{yeni}} = w1 - \eta * \frac{\partial L}{\partial w1} = 0.5 - 0.1 * (-0.0533) \approx 0.5053$$

$$w2_{\text{yeni}} = 0.4 - 0.1 * (-0.0089) \approx 0.4009$$

$$b_{\text{yeni}} = 0.2 - 0.1 * (-0.0888) \approx 0.2089$$

Açıklama:

- Hata yönünde ağırlıkları güncelledik.
- Büyük girdi → ağırlık daha fazla değişti, küçük girdi → az değişti.
- Bias tüm nöronları etkilediği için hatayı direkt azaltacak şekilde güncellendi.

1. İleri yayılım → ağ tahmin etti → hata bulundu.
2. Çıkıştaki hata türevi ve aktivasyon türevi → ağırlıklara etkisi hesaplandı.
3. Zincir kuralı → hatayı ağırlık ve biaslara doğru şekilde dağıttı.
4. Öğrenme oranı ile hata azaltacak şekilde ağırlıklar güncellendi.
5. Ağ bu işlemi tekrar tekrar yaparak hatayı küçültür → öğrenir.

NumPy ile Backpropagation Kodlama :

Bu örnekte, çok basit bir sinir ağı kullanacağız:

- **Giriş Katmanı:** 2 nöron
- **Gizli Katman:** 3 nöron
- **Çıkış Katmanı:** 1 nöron

Amacımız, bu ağı $X = [0.5, 0.8]$ girdisiyle eğiterek $Y = 0.7$ çıktısını doğru bir şekilde tahmin etmesini sağlamak.

1. Adım: Başlangıç Ayarları ve Fonksiyonlar

```
# Sigmoid aktivasyon fonksiyonu
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

# Sigmoid fonksiyonunun türevi
def sigmoid_derivative(x):
    return x * (1 - x)

# Rastgele ağırlık ve sapmaları belirleme
np.random.seed(1)

# Gerekli matris boyutları
input_size = 2
hidden_size = 3
output_size = 1

# Ağırlık ve sapmaların rastgele başlatılması
weights_hidden = np.random.randn(input_size, hidden_size)  # (2, 3)
biases_hidden = np.random.randn(1, hidden_size)             # (1, 3)
weights_output = np.random.randn(hidden_size, output_size) # (3, 1)
biases_output = np.random.randn(1, output_size)            # (1, 1)

# Öğrenme oranı
learning_rate = 0.1
```

2. Adım: İleri Yayılım (Forward Propagation)

```
# Girdi ve hedef veri
X = np.array([[0.5, 0.8]])  # (1, 2)
y = np.array([[0.7]])       # (1, 1)

# 1. Gizli katman için ileri yayılım
hidden_layer_input = np.dot(X, weights_hidden) + biases_hidden
hidden_layer_output = sigmoid(hidden_layer_input)

# 2. Çıkış katmanı için ileri yayılım
output_layer_input = np.dot(hidden_layer_output, weights_output) + biases_output
predicted_output = sigmoid(output_layer_input)

# Tahmin ve hata
print("Tahmin Edilen Çıktı:", predicted_output)
error = y - predicted_output
print("Hata:", error)
```

3. Adım: Geri Yayılım (Backpropagation)

```
# Çıkış katmanı için delta (hata gradyanı)
d_predicted_output = error * sigmoid_derivative(predicted_output)

# Gizli katman için delta
error_hidden_layer = np.dot(d_predicted_output, weights_output.T)
d_hidden_layer = error_hidden_layer * sigmoid_derivative(hidden_layer_output)

# Ağırlık ve sapma türevlerinin (gradyanların) hesaplanması
d_weights_output = np.dot(hidden_layer_output.T, d_predicted_output)
d_biases_output = np.sum(d_predicted_output, axis=0, keepdims=True)

d_weights_hidden = np.dot(X.T, d_hidden_layer)
d_biases_hidden = np.sum(d_hidden_layer, axis=0, keepdims=True)
```

4. Adım: Ağırlık ve Sapma Güncelleme

```
# Ağırlık ve sapmaların güncellenmesi
weights_output += d_weights_output * learning_rate
biases_output += d_biases_output * learning_rate
weights_hidden += d_weights_hidden * learning_rate
biases_hidden += d_biases_hidden * learning_rate

print("Güncellenmiş ağırlıklar ve sapmalar.")
```

5. Adım: Tüm Süreci Bir Döngüde Çalıştırma

```
# Eğitim döngüsü
epochs = 10000

for epoch in range(epochs):
    # İleri Yayılım (Adım 2)
    hidden_layer_input = np.dot(X, weights_hidden) + biases_hidden
    hidden_layer_output = sigmoid(hidden_layer_input)
    output_layer_input = np.dot(hidden_layer_output, weights_output) + biases_output
    predicted_output = sigmoid(output_layer_input)

    # Hata Hesaplama
    error = y - predicted_output

    # Geri Yayılım (Adım 3)
    d_predicted_output = error * sigmoid_derivative(predicted_output)
    error_hidden_layer = np.dot(d_predicted_output, weights_output.T)
    d_hidden_layer = error_hidden_layer * sigmoid_derivative(hidden_layer_output)
    d_weights_output = np.dot(hidden_layer_output.T, d_predicted_output)
    d_biases_output = np.sum(d_predicted_output, axis=0, keepdims=True)
    d_weights_hidden = np.dot(X.T, d_hidden_layer)
    d_biases_hidden = np.sum(d_hidden_layer, axis=0, keepdims=True)
```

```
    # Ağırlık Güncelleme (Adım 4)
    weights_output += d_weights_output * learning_rate
    biases_output += d_biases_output * learning_rate
    weights_hidden += d_weights_hidden * learning_rate
    biases_hidden += d_biases_hidden * learning_rate

    # Her 1000 epokta bir hata yazdırma
    if (epoch % 1000) == 0:
        mse = np.mean(np.square(error))
        print(f"Epok: {epoch}, Hata (MSE): {mse:.4f}")

print("\n--- Eğitim Sonucu ---")
print("Eğitim sonrası tahmin:", predicted_output)
```

Döngünün sonunda, modelin tahmini hedef değere (0.7) çok daha yakın olacaktır, bu da modelin başarıyla öğrendiğini gösterir.