MODÜL 5: Yapay Sinir Ağları (Neural Networks)

Şu ana kadar kurduğumuz modeller (Lineer ve Lojistik Regresyon) aslında **tek katmanlıdır**. Yani veriyi alır, tek bir işlemden geçirir ve sonucu verir.

Bu modeller **Lineer (Doğrusal)** sınırlar çizer.

- Verileri bir çizgiyle (veya düzlemle) ikiye ayırabilirsen çalışırlar.
- Ancak gerçek hayat "kıvrımlıdır".

🧠 Sorun: XOR Problemi

Aşağıdaki durumu düşün:

- (0, 0) 0 (Siyah)
- (0, 1) 1 (Beyaz)
- (1, 0) 1 (Beyaz)
- (1, 1) 0 (Siyah)

Buna **XOR (Exclusive OR)** kapısı denir. Bunu kağıda çizdiğinde (Siyahlar çapraz, Beyazlar çapraz), **tek bir düz çizgiyle** siyahları ve beyazları ayıramazsın. İki tane çizgiye ihtiyacın vardır.

İşte bu yüzden **Gizli Katmanlara (Hidden Layers)** ihtiyacımız var.

🏗️ Mimari: Çok Katmanlı Algılayıcı (MLP)

Artık tek bir nöron değil, birbirine bağlı nöron ordusu kuracağız.

1. **Input Layer (Girdi Katmanı):** Verilerin girdiği yer.
2. **Hidden Layer (Gizli Katman):** Özelliklerin işlendiği, büküldüğü yer. Burası modelin "zeka"sıdır.
3. **Output Layer (Çıktı Katmanı):** Son kararın verildiği yer.

Matematiksel olarak bu, **Arka Arkaya Matris Çarpımı** demektir:

$$Z_1 = X \cdot W_1 + b_1$$

$$A_1 = Aktivasyon(Z_1)$$

$$Z_2 = A_1 \cdot W_2 + b_2$$

$$Tahmin = Sigmoid(Z_2)$$

📉 Yeni Oyuncu: ReLU Aktivasyon Fonksiyonu

Gizli katmanlarda artık Sigmoid kullanmıyoruz. Çünkü Sigmoid türevi çok küçültür (Vanishing Gradient sorunu) ve öğrenme yavaşlar.

Modern AI'ın standardı **ReLU (Rectified Linear Unit)** fonksiyonudur. Mantığı aşırı basittir: **"Negatifse 0 yap, Pozitifse olduğu gibi bırak."**

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

- `ReLU(-5) = 0`
- `ReLU(10) = 10`

Bu basit fonksiyon, modele "Lineer Olmayan" (Non-linear) karmaşık şekilleri öğrenme gücü verir.

🧠 Konu 1: Neden Aktivasyon Fonksiyonuna Muhtacız? (Lineerlik Tuzağı)

Şöyle düşünebilirsin: *"Madem derin öğrenme yapıyoruz, neden araya fonksiyon koymadan dümdüz matrisleri birbiriyle çarpmıyoruz?"*

Matematiksel bir ispat yapalım.

Diyelim ki 2 katmanlı bir sinir ağımız var ama **Aktivasyon Fonksiyonu YOK**.

1. **Katman 1:** $Z_1 = X \cdot W_1 + b_1$
2. **Katman 2:** $Z_2 = Z_1 \cdot W_2 + b_2$

Şimdi Z1'i ikinci denklemde yerine koyalım:

$$Z_2 = (X \cdot W_1 + b_1) \cdot W_2 + b_2$$

Parantezi açalım (Dağılma özelliği):

$$Z_2 = X \cdot (W_1 \cdot W_2) + (b_1 \cdot W_2 + b_2)$$

Dikkat et!

- $W_1 \cdot W_2$ çarpımı, aslında tek bir yeni matristir. Buna W_{yeni} diyelim.
- $b_1 \cdot W_2 + b_2$ işlemi, tek bir sabit sayıdır. Buna b_{yeni} diyelim.

Sonuç:

$$Z_2 = X \cdot W_{yeni} + b_{yeni}$$

Sonuç Şok Edici: Araya aktivasyon fonksiyonu koymazsan, 100 tane gizli katman da koysan, milyarlarca nöron da kullansan, matematiksel olarak modelin **Tek Katmanlı Lineer Regresyon** (y = w.x + b) ile birebir aynıdır.

Özetle: Aktivasyon fonksiyonu (Sigmoid, ReLU vb.), sisteme **"Doğrusal Olmayanlık" (Non-Linearity)** katar.

- Lineer model sadece düz çizgi çizebilir.
- Non-lineer model (Aktivasyonlu) kağıdı bükebilir, kıvırabilir ve XOR gibi karmaşık sınırları çizebilir.

💀 Konu 2: "Vanishing Gradient" (Kaybolan Gradyan) Problemi

- Bu sorun, yapay zeka dünyasını 2010'lara kadar durduran "Kış Dönemi"nin (AI Winter) baş sorumlusudur.
- Modeli eğitirken **Backpropagation (Geri Yayılım)** yapıyoruz. Yani türevleri sondan başa doğru zincirleme çarpıyoruz (Zincir Kuralı).
- Diyelim ki 4 katmanlı bir ağımız var. En baştaki ağırlığın (w1) türevi şöyle hesaplanır:

$$\frac{\partial Loss}{\partial w_1} = \frac{\partial Loss}{\partial y} \cdot \underbrace{\frac{\partial y}{\partial h_4} \cdot \frac{\partial h_4}{\partial h_3} \cdot \frac{\partial h_3}{\partial h_2} \cdot \frac{\partial h_2}{\partial h_1}}_{\text{Zincirleme Çarpım}} \cdots$$

Buradaki terimlerin her biri, **Aktivasyon Fonksiyonunun Türevidir.**

Suçlu: Sigmoid Fonksiyonu

Sigmoid fonksiyonunu hatırla: . Türevi şudur: $S(x) \cdot (1 - S(x))$.

- Sigmoid'in türevinin alabileceği **maksimum değer 0.25**'tir. (Tam merkezde).
- Uçlara gidildikçe türev **0**'a yaklaşır.

Şimdi matematiksel felakete bak: Zincir kuralında sürekli bu küçük sayıları çarpıyoruz.

$$Gradient \approx 0.25 \times 0.25 \times 0.25 \times 0.25 \times \dots$$

Bir sayıyı sürekli 1'den küçük bir sayıyla çarparsan ne olur? Hızla **Sıfır** olur.

Sonuç: Ağın en başındaki katmanlara (girişe yakın olanlara) ulaşan hata bilgisi **0.00000001** gibi bir şey olur.

- **Sonuç:** İlk katmanlar hiçbir şey öğrenemez. Ağırlıkları güncellenmez.
- Model sadece son katmanlarını eğitir, derinliğin bir anlamı kalmaz. Buna **Vanishing Gradient** denir.

🚀 Kurtarıcı: ReLU (Rectified Linear Unit)

Bilim insanları yıllarca bu sorunu çözmeye çalıştı ve çözüm şaşırtıcı derecede basit çıktı: **ReLU.**

Formül: Formül: $f(x) = \max(0, x)$

Neden ReLU kullanıyoruz?

ReLU'nun türevini düşünelim (Eğimini):

1. **Eğer Girdi < 0 ise:** Fonksiyon düz çizgi (0). Eğim = **0**.
2. **Eğer Girdi > 0 ise:** Fonksiyon y = x doğrusu. Eğim = **1**.

İşte büyü burada! **Eğim (Türev) 1'dir.**

Zincir kuralına geri dönelim:

$$Gradient \approx 1 \times 1 \times 1 \times 1 \dots$$

Sonuç: Hata sinyali, hiç küçülmeden (sönümlenmeden) en sondan en başa kadar **%100 şiddetle** akar. Bu sayede 100 katmanlı derin ağları (Deep Learning) eğitebiliriz.

- **Tek Dezavantaj (Dead ReLU):** Eğer girdi negatifse türev 0 olur ve nöron "ölür" (hiç öğrenmez). Ama pozitif bölgede mükemmel çalışır.

🗺️ Toparlarsak

1. **Neden Aktivasyon Fonksiyonu?**
   - Olmazsa ağımız "derin" değil, sadece karmaşık yazılmış bir lineer regresyon olur. Eğri bükemez, zor problemleri çözemez.
2. **Neden Sigmoid değil de ReLU?**
   - Sigmoid türevi küçüktür (max 0.25). Katman sayısı arttıkça türevlerin çarpımı sıfıra gider (Vanishing Gradient). Ağın başı öğrenmeyi durdurur.
   - ReLU'nun türevi pozitifle 1'dir. Hata sinyali kaybolmadan akar.

🔍 Matris Çarpımının Anatomisi

Senaryomuzdaki boyutları hatırlayalım:

- **X (Girdi):** (4 Örnek, 3 Özellik) $\rightarrow$ Şekil: `(4, 3)`
- **W1 (Ağırlıklar):** (3 Özellik, 5 Nöron) $\rightarrow$ Şekil: `(3, 5)`

1. Boyut Kontrolü (Shape Check)

Çarpımın yapılabilmesi için **iç boyutların** (Girdinin sütunu ile Ağırlığın satırı) eşit olması şarttır.

$$(4, \mathbf{3}) \cdot (\mathbf{3}, 5) \rightarrow \text{Sonuç: } (4, 5)$$

- **3 ve 3:** Eşleşiyor ✅ (Bu sayılar "Feature" sayısıdır. Her özellik, kendi ağırlığıyla çarpılmalı).
- **4:** Sonuçta 4 satır olacak (Çünkü 4 farklı ev/veri örneğimiz var).
- **5:** Sonuçta 5 sütun olacak (Çünkü 5 farklı nöronumuz/filtremiz var).

🧮 Adım Adım İşlem (Satır x Sütun)

Hayal edelim:

- **X'in ilk satırı (İlk Ev):** $[x_1, x_2, x_3]$ (Örn: `[10, 2, 5]`)
- **W1'in ilk sütunu (1. Nöronun Ağırlıkları):** $\begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix}$ (Örn: `[0.1, -0.5, 0.2]`)

Bu ikisi çarpıştığında **tek bir sayı** (Z matrisinin sol üst köşesindeki hücre) oluşur:

$$z_{1,1} = (x_1 \cdot w_1) + (x_2 \cdot w_2) + (x_3 \cdot w_3) + b_1$$

Sayısal örnekle:

$$z_{1,1} = (10 \cdot 0.1) + (2 \cdot -0.5) + (5 \cdot 0.2)$$

$$z_{1,1} = (1) + (-1) + (1) = \mathbf{1}$$

Bu işlem:

1. **X'in 1. Satırı** ile **W'nin 1. Sütunu** çarpılır Sonuç Matrisinin `(0,0)` hücresi.
2. **X'in 1. Satırı** ile **W'nin 2. Sütunu** çarpılır Sonuç Matrisinin `(0,1)` hücresi.
3. ...ve böyle devam eder.

Böylece her bir veri örneği (satır), ağdaki her bir nöronla (sütun) tek tek etkileşime girmiş olur.

🔍 Görsel Analiz: Bir Nöronun Anatomisi

Senin örneğindeki sayıları kullanalım:

- **Girdi (X):** 3 Özellik (Metrekare, Oda Sayısı, Bina Yaşı).
- **Gizli Katman:** 5 Nöron var.

Şimdi bu 5 nörundan sadece **1 tanesini** (diyelim ki "Nöron A") ele alalım.

"Nöron A"nın bir karar verebilmesi için **elindeki bütün verilere** bakması gerekir.

1. Metrekareye bakmalı Bunun için 1 ağırlık lazım (w1).
2. Oda sayısına bakmalı Bunun için 1 ağırlık lazım (w2).
3. Yaşına bakmalı Bunun için 1 ağırlık lazım (w3).

Yani; **3 tane girdi olduğu için, Nöron A'nın tam 3 tane ağırlığı vardır.**

Eğer 5 tane gizli nöronumuz varsa:

- Nöron 1'in de 3 ağırlığı var.
- Nöron 2'nin de 3 ağırlığı var.
- ...
- Nöron 5'in de 3 ağırlığı var.

Toplam Ağırlık Sayısı: tane ağırlık.

🎛️ Matris İçindeki Yeri (Sütun Mantığı)

Kodda `W1` matrisini `(3, 5)` boyutunda tanımlamıştık. Bunu bir tablo gibi düşünürsen:

- **Satırlar (3):** Girdiler (Metrekare, Oda, Yaş).
- **Sütunlar (5):** Senin nöronların (Nöron 1, Nöron 2, ..., Nöron 5).

Bu matristeki **HER BİR SÜTUN, tek bir nöronu temsil eder.**

```
        Nöron 1   Nöron 2   Nöron 3   Nöron 4   Nöron 5
       (Sütun 0) (Sütun 1) (Sütun 2) (Sütun 3) (Sütun 4)
          ↓         ↓         ↓         ↓         ↓
m2    [[ 0.1,      0.5,     -0.2,      0.9,      0.1 ],
Oda    [ 0.8,     -0.1,      0.0,      0.4,     -0.5 ],  <- W1 Matrisi (3, 5)
Yaş    [-0.5,      0.2,      1.5,     -0.7,      0.3 ]]
```

Buraya bakarsan:

- **Nöron 1 (İlk Sütun):** Ağırlıkları `[0.1, 0.8, -0.5]`. (3 tane).
- **Nöron 2 (İkinci Sütun):** Ağırlıkları `[0.5, -0.1, 0.2]`. (3 tane).