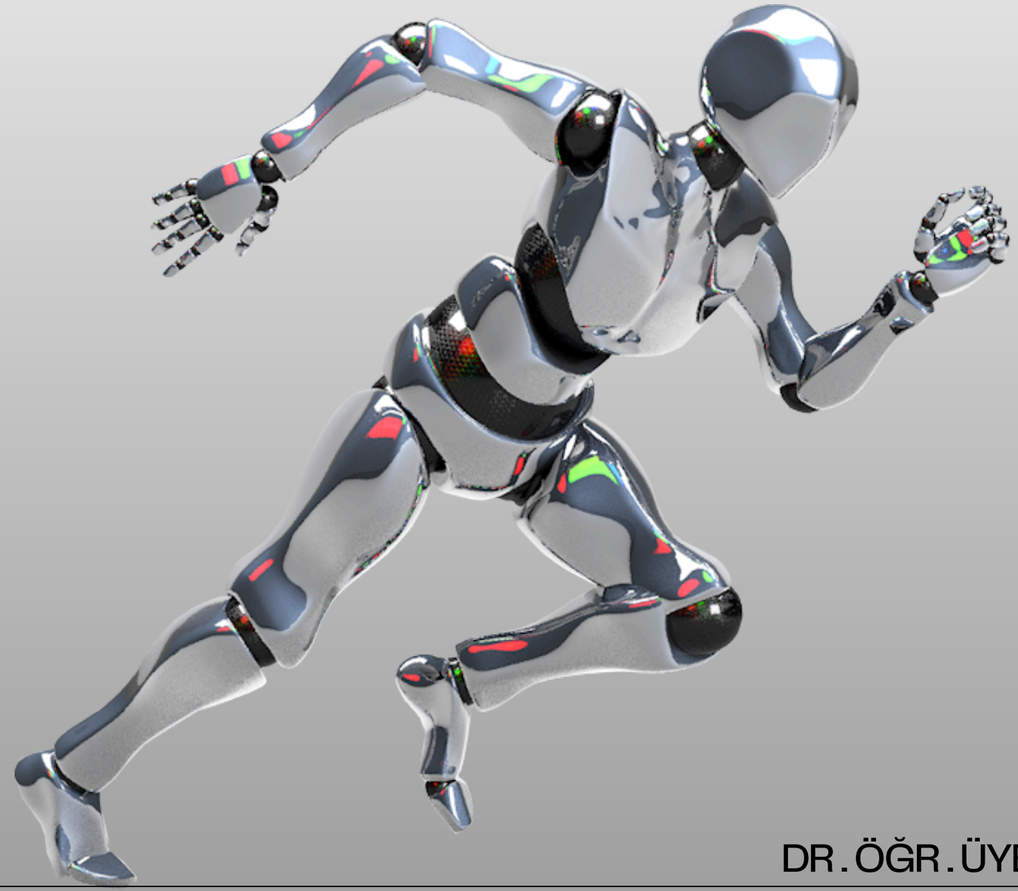
1.HAFTA

YAPAY SİNİR AĞLARINA GİRİŞ



DR. ÖĞR. ÜYESİ ALPER TALHA KARADENİZ

SAMSUN ÜNİVERSİTESİ

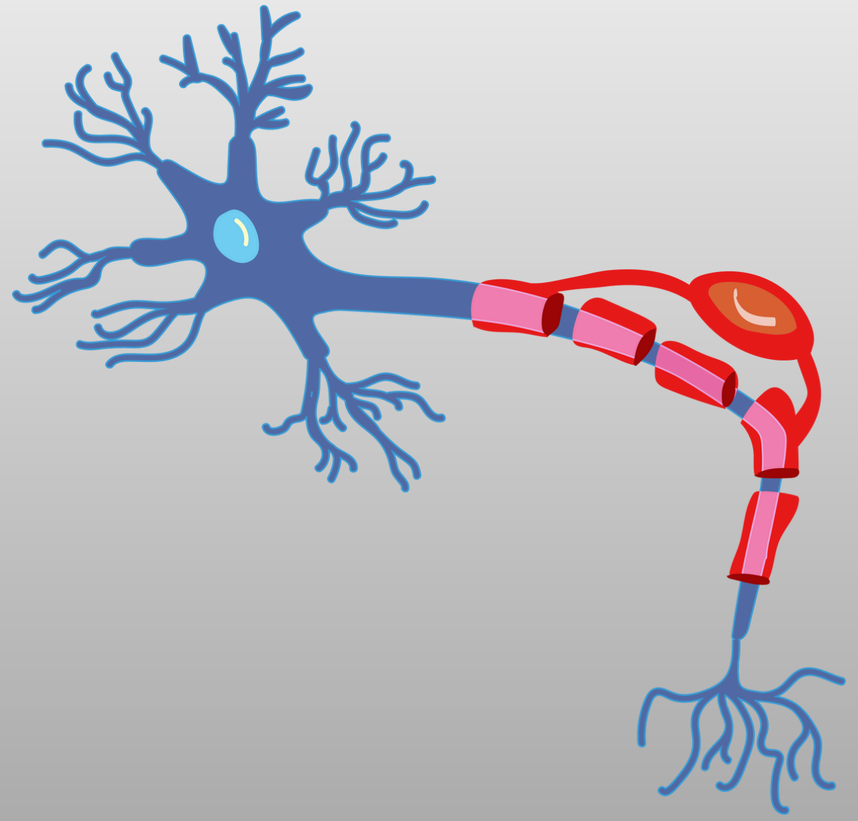
BİYOLOJİK NÖRON YAPISI

TEMEL BİLEŞENLER:

- **Dendritler: Diğer nöronlardan gelen sinyalleri alan dallı yapılar**
- **Hücre gövdesi (Soma): Gelen sinyalleri işleyen merkezi bölüm**
- **Akson: Çıkış sinyalini diğer nöronlara ileten uzun fiber**
- **Sinaps: Nöronlar arası bağlantı noktaları**

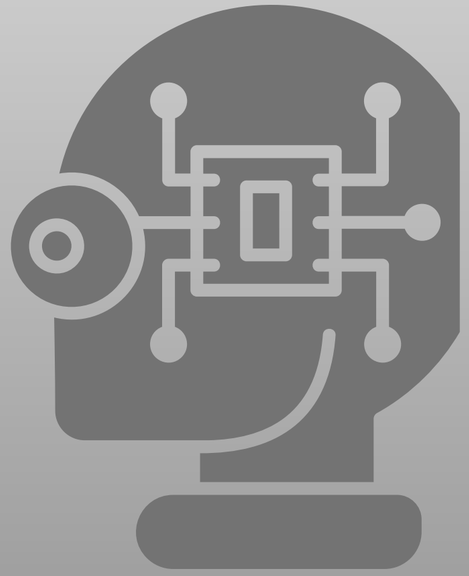
ÇALIŞMA PRENSİBİ:

- **Dendritler elektrik sinyalleri alır**
- **Hücre gövdesi sinyalleri toplar ve değerlendirir**
- **Eşik değer aşılırsa akson boyunca sinyal iletilir**
- **Sinapslar aracılığıyla diğer nöronlara sinyal aktarılır**



YAPAY NÖRON İLE KARŞILAŞTIRILMASI:

Biyolojik Nöron	Yapay Nöron
Dendritler → Girdiler ($x_1, x_2, \dots x_n$)	Giriş değerleri
Sinaptik ağırlıklar → Ağırlıklar ($w_1, w_2, \dots w_n$)	*Bağlantı kuvvetleri*
Hücre gövdesi → Toplama fonksiyonu	$\Sigma(w_i \times x_i)$
Eşik değer → Aktivasyon fonksiyonu	*f(net)*
Akson çıkışı → Çıkış (y)	Sonuç



Temel Benzerlikler:

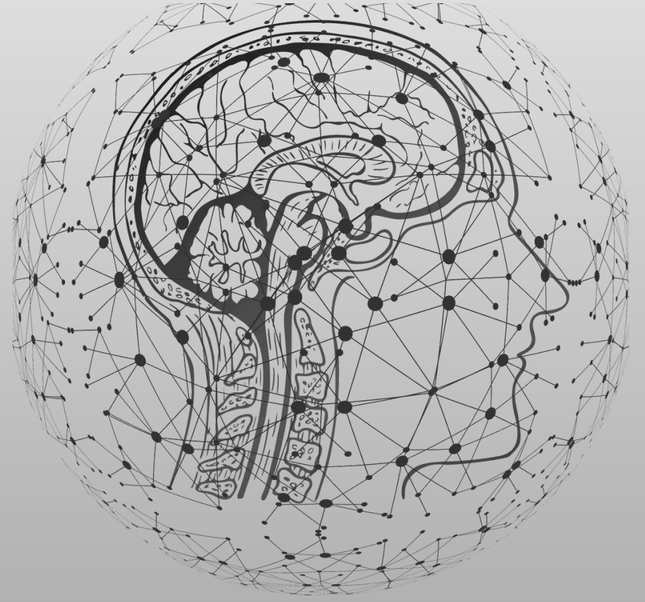
- Her ikisi de paralel işlem yapar
- Ağırlıklı girdi toplamı kullanır
- Eşik tabanlı karar mekanizması
- Öğrenme yoluyla ağırlıkları günceller

YAPAY SİNİR AĞLARI:

- Yapay Sinir Ağı (Artificial Neural Network – ANN), insan beyninin çalışma prensiplerinden esinlenerek geliştirilmiş, birbirine bağlı yapay nöronlardan oluşan hesaplama modelidir.

TEMEL ÖZELLİKLER:

- Paralel işlem: Binlerce nöron aynı anda çalışır
- Dağıtık bellek: Bilgi ağırlıklarda saklanır
- Öğrenme yetisi: Verilerden örüntüleri öğrenir
- Genelleme: Yeni verilere adapte olabilir
- Hataya tolerans: Bazı nöronlar arızalansa da çalışır



NEDEN KULLANILIR ? :

Geleneksel programlamanın zorlandığı alanlar:

- Görüntü tanıma: "Kedi" nasıl tanımlanır? Kurallarla zor
- Konuşma tanıma: Aksanlar, gürültü, farklı sesler
- Doğal dil işleme: Dil kuralları karmaşık ve esnek
- Örüntü tanıma: Karmaşık veri setlerindeki gizli ilişkiler

ANN' NIN AVANTAJLARI:

- Doğrusal olmayan ilişkileri modelleyebilir
- Eksik veya gürültülü verilerle çalışabilir
- Otomatik özellik çıkarımı yapar
- Büyük veri setlerinden öğrenir

ÖRNEK–El Yazısı Rakamı Tanıma:

Geleneksel yaklaşım: Her rakam için kurallar yazılması
– "0" için: Kapalı döngü, ortada boşluk…
– "1" için: Dikey çizgi, genellikle sağda…
Problem: Herkesin yazısı farklı!

ANN yaklaşımı:
– Binlerce el yazısı örneği göster
– Ağ kendisi özellikları öğrenir
– Yeni yazıları %98+ doğrulukla tanır

ANN'in Tarihçesi :

1943 – İlk Yapay Nöron
- McCulloch & Pitts: İlk matematiksel nöron modeli
- Binary girdi/çıktı ile mantık kapıları

1958 – Perceptron
- Frank Rosenblatt: İlk öğrenebilen yapay nöron
- Doğrusal sınıflandırma problemi çözümü
- "Mark I Perceptron" donanımı

1969 – İlk Kış Dönemi
- Minsky & Papert: "Perceptrons" kitabı
- XOR problemini çözemediği gösterildi
- Araştırmalarda durgunluk

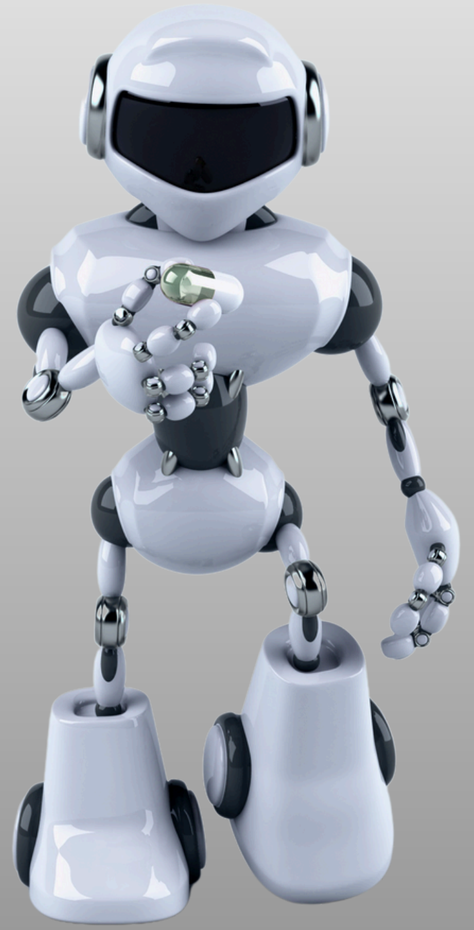
1986 – Rönesans
- Backpropagation algoritması yaygınlaştı
- Çok katmanlı ağlar geliştirildi
- Doğrusal olmayan problemler çözüldü

2006 – Derin Öğrenme
- Geoffrey Hinton: "Deep Learning" terimi
- GPU'lar ile büyük ağlar eğitildi
- ImageNet başarısı (2012)

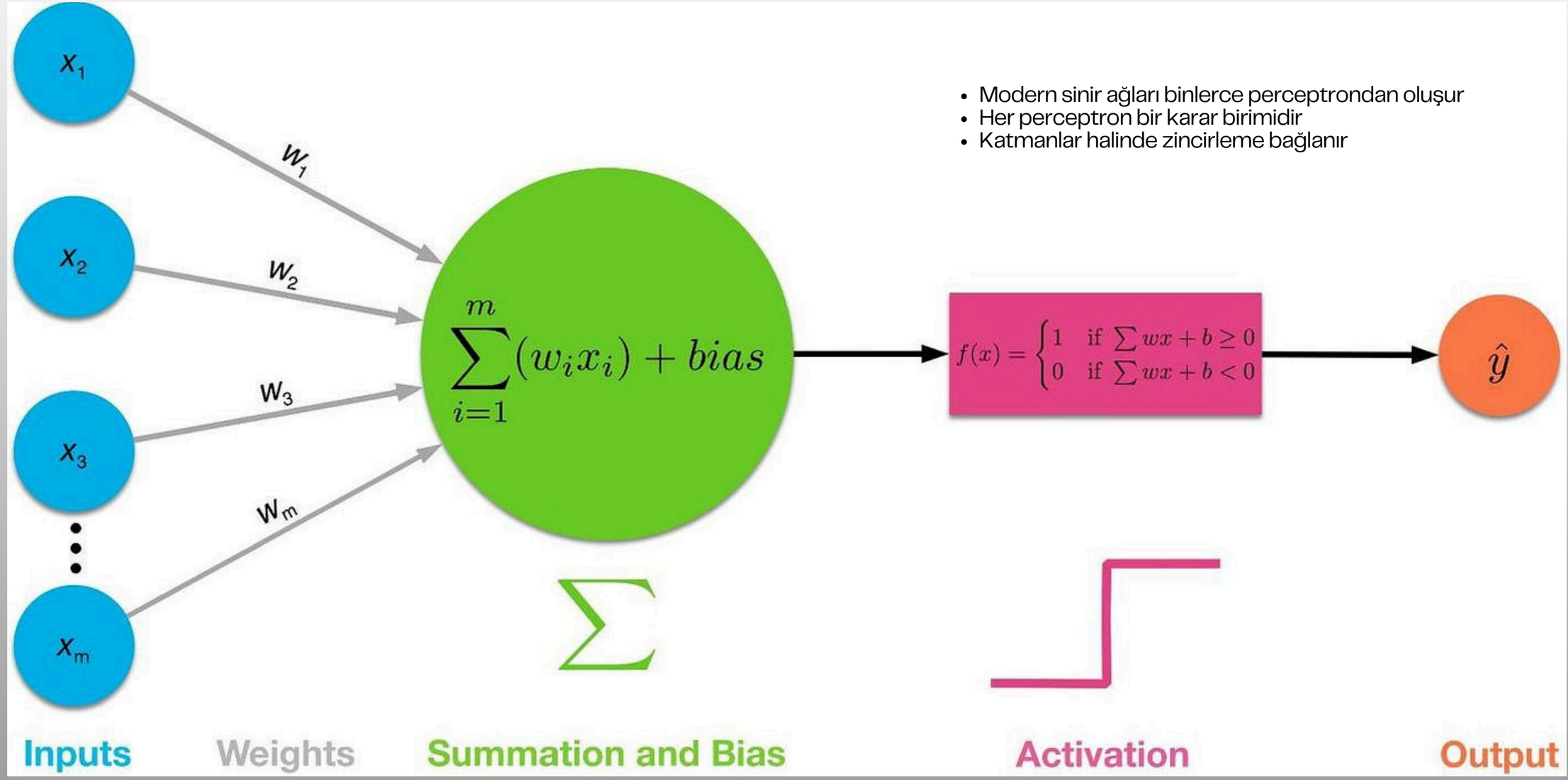
2010'lar – AI Devrimi
- Büyük veri + güçlü donanım
- AlexNet, ResNet, Transformer mimarileri
- ChatGPT, GPT-4 gibi büyük dil modelleri

Modern Kullanım Alanları:

1. Bilgisayarlı Görü
   - Nesne tanıma: Tesla'nın otonom araçları
   - Medikal görüntüleme: Kanser teşhisi
   - Yüz tanıma: Güvenlik sistemleri
   - Görüntü üretimi: DALL-E, Midjourney
2. Doğal Dil İşleme
   - Çeviri: Google Translate
   - Chatbotlar: ChatGPT, Claude
   - Ses tanıma: Siri, Alexa
   - Metin özetleme: Haber sitelerinde
3. Oyunlar ve Strateji
   - AlphaGo: Go oyununda dünya şampiyonu
   - OpenAI Five: Dota 2'de profesyonel takımları yendi
   - AlphaStar: StarCraft II'de grandmaster seviyesi
4. Bilim ve Araştırma
   - Protein katlanması: AlphaFold
   - İlaç keşfi: Yeni moleküller
   - Hava durumu tahmini: Meteoroloji
   - Finansal tahminler: Algoritmik trading
5. Günlük Hayat
   - Netflix: Film önerileri
   - Spotify: Müzik önerileri
   - Instagram: Filtreler ve etiketleme
   - Google Arama: Sonuç sıralama



Perceptron : Yapay sinir ağlarının en temel ve en basit yapı taşıdır. 1958 yılında Frank Rosenblatt tarafından geliştirilmiş, öğrenme yetisi olan ilk yapay nöron modelidir.

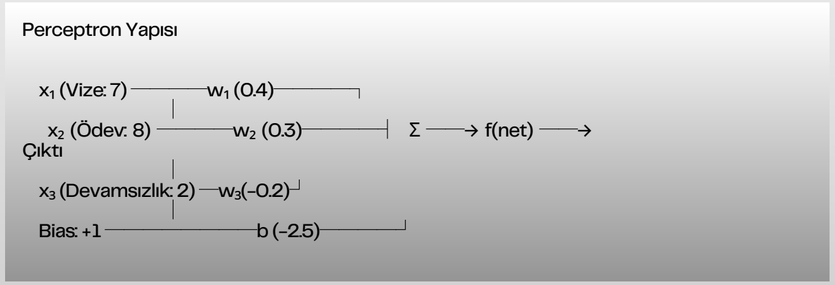


- Modern sinir ağları binlerce perceptrondan oluşur
- Her perceptron bir karar birimidir
- Katmanlar halinde zincirleme bağlanır

Örnek :

📊 Girdilerimiz:

- x_1: Vize notu (0–10 arası)
- x_2: Ödev notu (0–10 arası)
- x_3: Devamsızlık sayısı (0–5 arası)



Adım 1: Ağırlıklı Toplam (Σ)

net = $(x_1 \times w_1) + (x_2 \times w_2) + (x_3 \times w_3) + (\text{bias} \times b)$
net = $(7 \times 0.4) + (8 \times 0.3) + (2 \times -0.2) + (1 \times -2.5)$
net = 2.8 + 2.4 + (-0.4) + (-2.5)
net = 2.3

Adım 2: Aktivasyon Fonksiyonu

Step Function (Basamak Fonksiyonu):
f(net) = 1, eğer net ≥ 0
f(net) = 0, eğer net < 0
Bizim durumumuzda: net = 2.3 ≥ 0
Sonuç: f(2.3) = 1

Sonuç: Öğrenci GEÇTİ! ✅

🔹 Bias (Sapma)
→ Karar eşiğini ayarlayan sabit değerdir.
Negatifse geçmeyi zorlaştırır, pozitifse kolaylaştırır.

🔹 Ağırlık (Weight)
→ Girdilerin önemini belirleyen katsayılardır.
Her girdi (vize, ödev, devamsızlık) farklı etkiye sahiptir.

🔹 Aktivasyon Fonksiyonu
→ Sonuç pozitif mi negatif mi diye karar verir.
En basiti “step function”:

- $x \geq 0 \rightarrow 1$ (GEÇTİ)
- $x < 0 \rightarrow 0$ (KALDI)

Uygulama : Basit Perceptron Görselleştirmesi :

Perceptron, tıpkı beynimizde bir nöron gibi:

1. Birden fazla girdi alır (örn: yaş, not, devamsızlık)
2. Her girdiye farklı önem verir (ağırlıklar)
3. Hepsini toplar ve karar verir (geçer/kalır, spam/değil)

Matematiksel Model :

Girdi: $x_1, x_2, \_, x_n$
Ağırlıklar: $w_1, w_2, \_, w_n$
Bias: b

Net girdi: net = $\Sigma(w_i \times x_i) + b$
Çıkış: y = f(net)

Aktivasyon fonksiyonu (Step function):
f(net) = 1 eğer net ≥ 0
0 eğer net < 0

Pratik Örnek - AND Kapısı

python

```
# AND kapısı için perceptron
# Girdi: (x₁, x₂), Çıktı: x₁ AND x₂

Eğitim verisi:
(0,0) → 0
(0,1) → 0
(1,0) → 0
(1,1) → 1

Öğrenilen ağırlıklar:
w₁ = 0.5, w₂ = 0.5, b = -0.7

Test:
(1,1): net = 0.5×1 + 0.5×1 - 0.7 = 0.3 ≥ 0 → Çıktı: 1 ✓
(0,1): net = 0.5×0 + 0.5×1 - 0.7 = -0.2 < 0 → Çıktı: 0 ✓
```

Öğrenme Algoritması (Perceptron Learning Rule)

```
1. Ağırlıkları rastgele başlat
2. Her eğitim örneği için:
   a. Çıktıyı hesapla: y = f(Σ(wᵢ × xᵢ) + b)
   b. Hatayı bul: error = hedef - y
   c. Ağırlıkları güncelle:
      wᵢ = wᵢ + α × error × xᵢ
      b = b + α × error
3. Tüm örnekler doğru sınıflandırılana kadar tekrarla
```

Perceptron'un Sınırları

Çözebileceği problemler:

- AND, OR, NOT kapıları
- Doğrusal olarak ayrılabilir sınıflandırma

Çözemediği problemler:

- XOR kapısı (doğrusal olarak ayrılamaz)
- Karmaşık örüntüler

XOR Problemi Örneği

```
XOR gerçek tablosu:
(0,0) → 0
(0,1) → 1
(1,0) → 1
(1,1) → 0

Bu problem tek doğru ile ayrılamaz!
Çözüm: Çok katmanlı ağ gerekir
```

NumPy ile Basit Implementasyon:

Görselleştirme Kavramları :

1. Karar sınırı: Sınıfları ayıran çizgi
2. Ağırlık vektörü: Karar sınırına dik vektör
3. Bias: Karar sınırının orijinden uzaklığı
4. Öğrenme süreci: Karar sınırının iteratif olarak iyileştirilmesi

Özet ve Çıkarımlar :

- Perceptron, yapay sinir ağlarının temel taşıdır
- Basit ama güçlü bir öğrenme algoritması sunar
- Modern derin öğrenme modellerinin temeli
- Doğrusal problemler için hâlâ kullanışlı
- Çok katmanlı ağlarla sınırları aşılabilir

```python
import numpy as np

class Perceptron:
    def __init__(self, learning_rate=0.1):
        self.learning_rate = learning_rate
        self.weights = None
        self.bias = None

    def fit(self, X, y, epochs=100):
        # Ağırlıkları başlat
        self.weights = np.random.rand(X.shape[1])
        self.bias = np.random.rand()

        # Eğitim döngüsü
        for epoch in range(epochs):
            for i in range(len(X)):
                # Tahmin yap
                net_input = np.dot(X[i], self.weights) + self.bias
                prediction = 1 if net_input >= 0 else 0

                # Ağırlıkları güncelle
                error = y[i] - prediction
                self.weights += self.learning_rate * error * X[i]
                self.bias += self.learning_rate * error

    def predict(self, X):
        net_input = np.dot(X, self.weights) + self.bias
        return np.where(net_input >= 0, 1, 0)
```