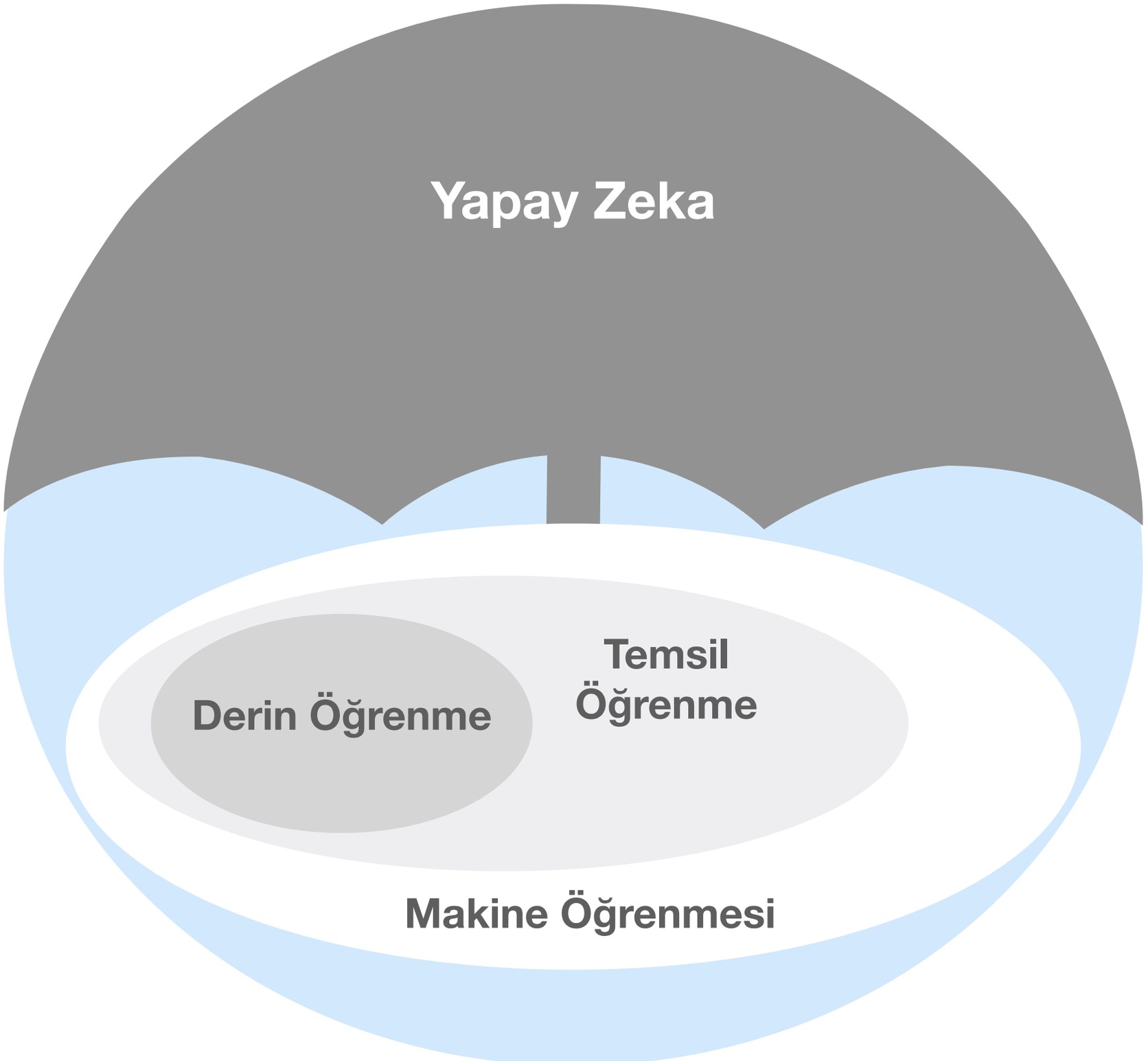
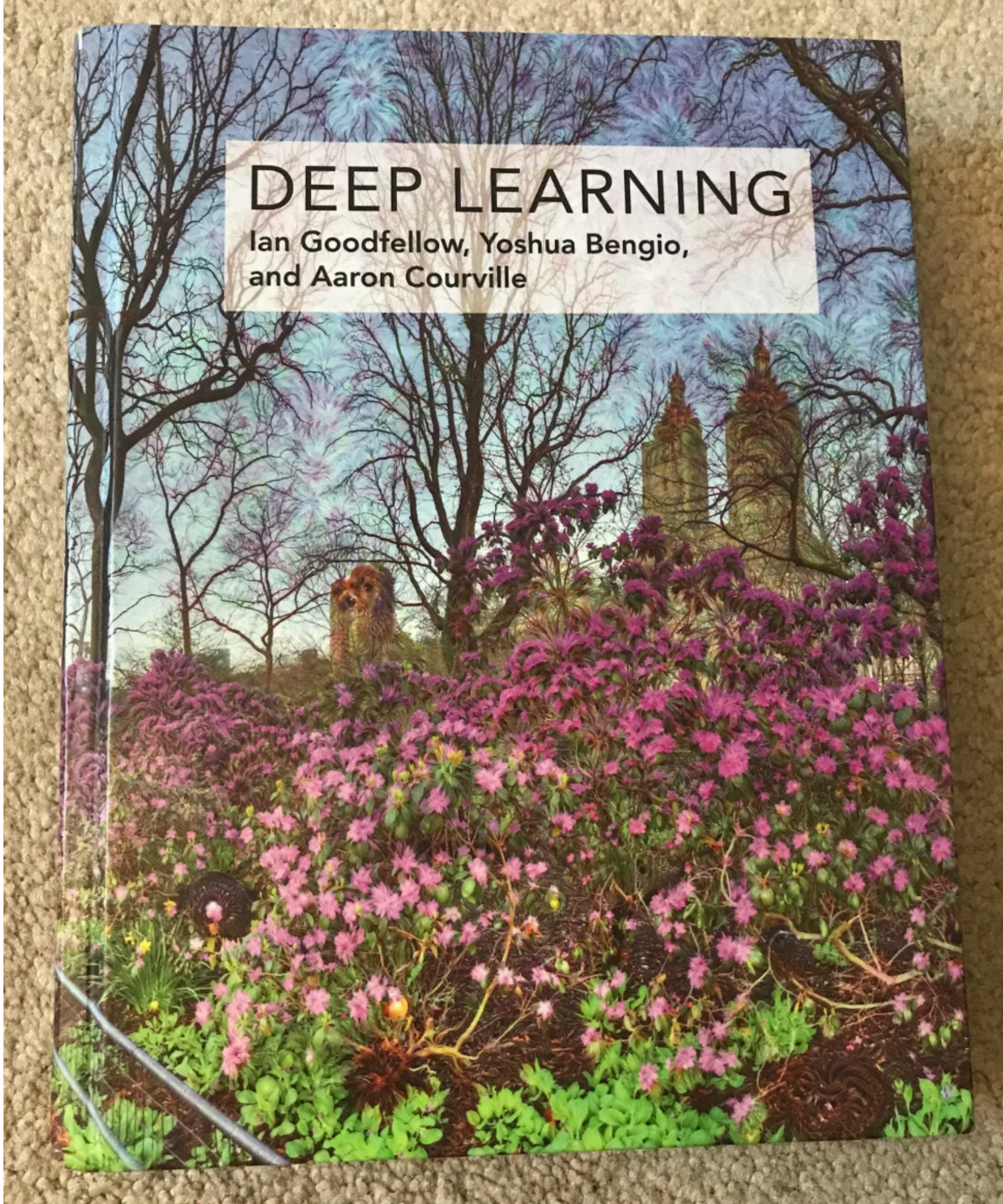
Makine Öğrenimi Giriş

Ece Asilar

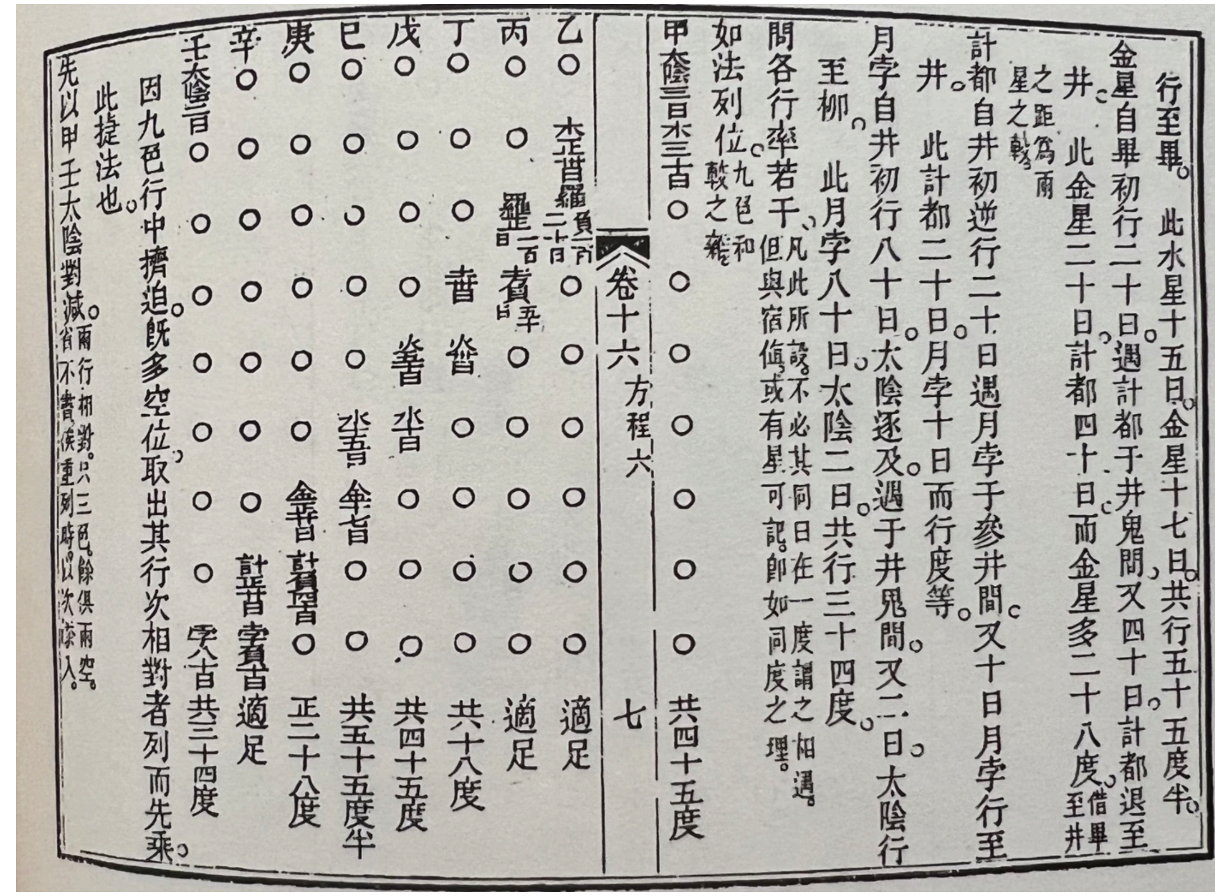
Hanyang University
CERN
ece.asilar@cern.ch


Yapay Zeka
Derin Öğrenme
Temsil
Öğrenme
Makine Öğrenmesi

- Tüm seviyeler icin uygun
- Matematiğini öğrenmek icin şahane



- René Descartes (1596–1650) koordinat düzlemi, cebirsel denklemlerin geometrik temsili.
- Carl Friedrich Gauss (1777–1855) özellikle doğrusal denklemler sistemlerinin çözümünde ve Gauss eliminasyon yönteminde katkıda bulunmuştur.
- Augustin-Louis Cauchy (1789–1857) matris teorisi ve determinantlar.
- Arthur Cayley (1821–1895) ve James Joseph Sylvester (1814–1897) matris teorisini tanıtmış ve modern lineer cebirin temel unsurları olan doğrusal dönüşümleri incelemiştir.



Nine Chapters on the Mathematical Arts (*Jiuzhang suanshu* 九章算術, c. 1st century CE

Bulmaca olsun:
Önce kim ?

birazcık tarih...

1940
Dark Era
Until 1940

???

1943
Neural Nets
McCulloch & Pitt

1950
Computing Machinery and Intelligence
Alan Turing

1958
Perceptron
Rosenblatt

1960
ADALINE
Widrow & Hoff

1969
XOR problem
Minsky & Papert

1974
Backpropagation
Werbos (and more)

1980
Self Organizing Map
Kohonen

1980
Neocogitron
Fukushima

1982
Hopfield Network
John Hopfield

1985
Boltzmann Machine
Hinton & Sejnowski

1986
Multilayer Perceptron
Rumelhart, Hinton & Williams

1986
Restricted Boltzmann Machine
Smolensky

1986
RNNs
Jordan

1990
LeNet
Lecun

1997
Bidirectional RNN
Schuster & Paliwal

1997
LSTMs
Hochreiter & Schmidhuber

2006
Deep Belief Networks - pretraining
Hinton

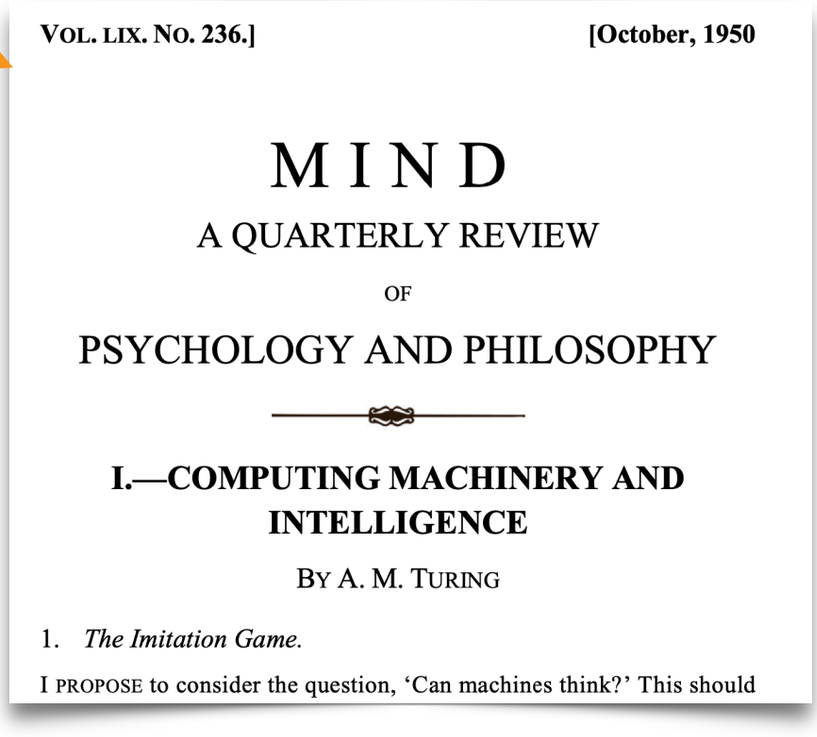
2006
Deep Boltzmann Machines
Salakhutdinov & Hinton

2012
Dropout
Hinton

2014
GANs
Goodfellow

2017
Capsule Networks
Sabour, Frosst, Hinton

Made by Favio Vázquez



VOL. LIX. NO. 236.] [October, 1950

MIND

A QUARTERLY REVIEW

OF

PSYCHOLOGY AND PHILOSOPHY

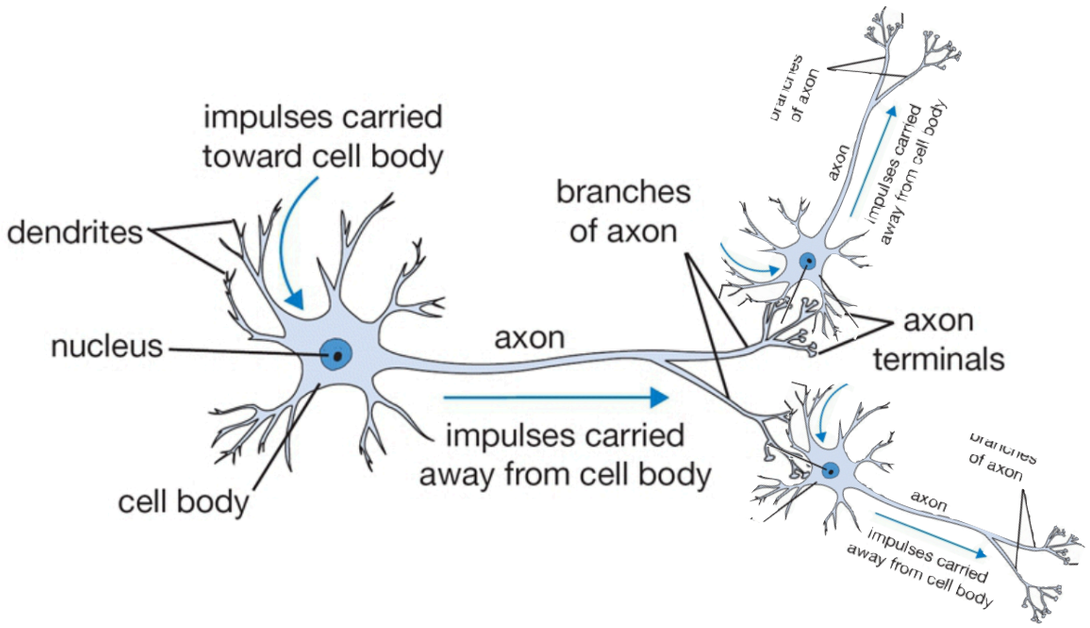
I.—COMPUTING MACHINERY AND INTELLIGENCE

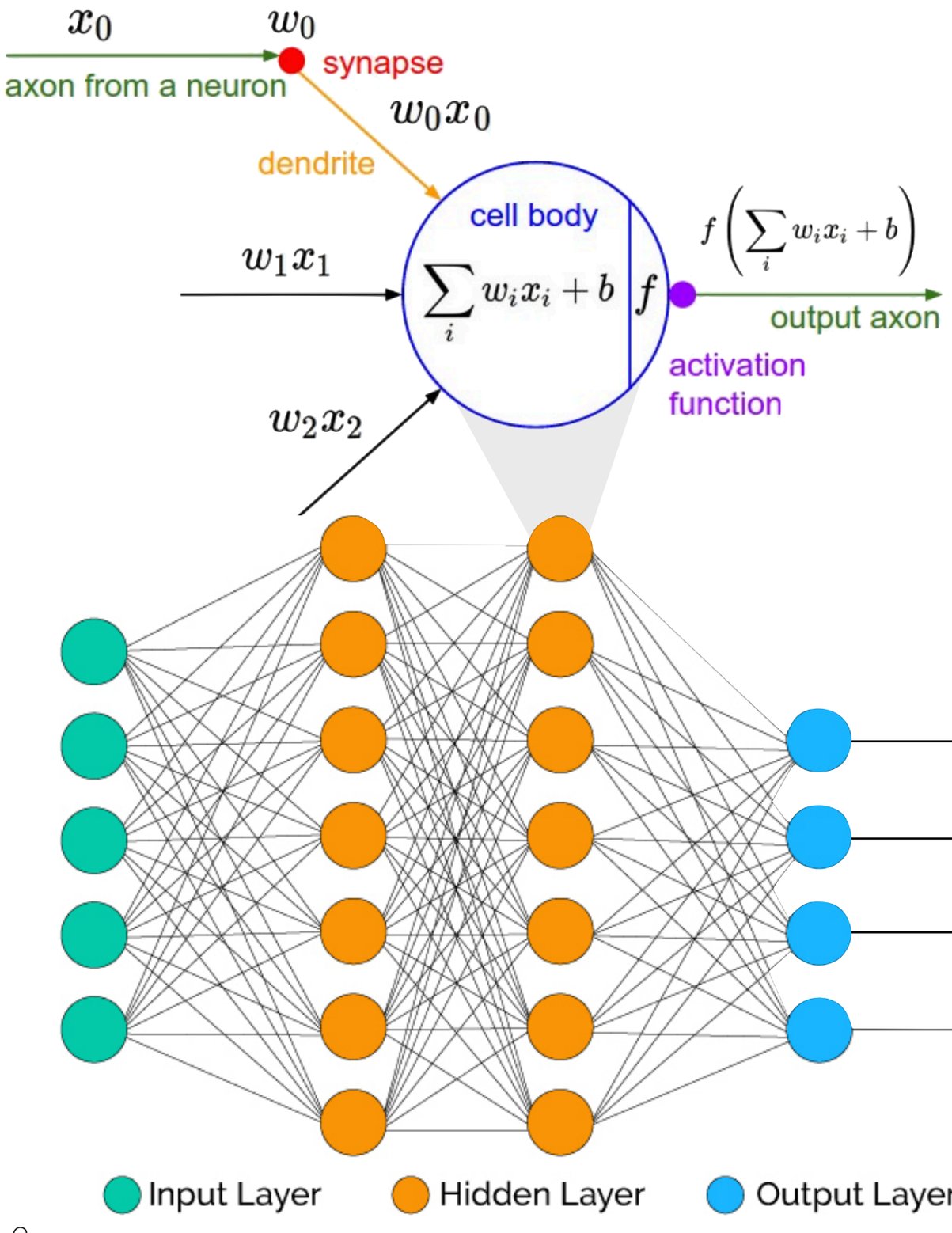
BY A. M. TURING

1. *The Imitation Game.*

I PROPOSE to consider the question, 'Can machines think?' This should

Derin Öğrenme Nedir ?



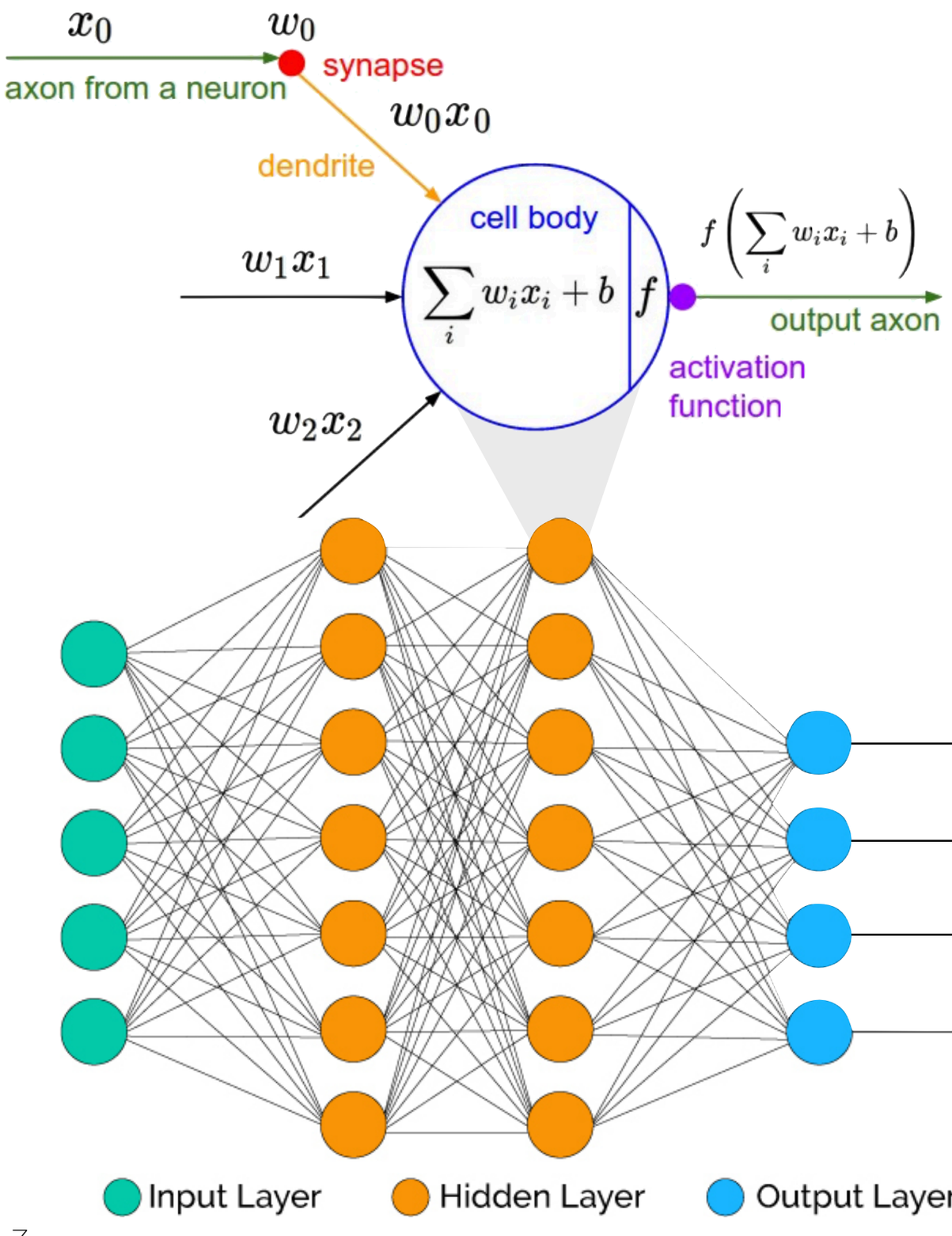


- Verileriniz için iyi temsiller (özellikler) bulma
- Birden fazla katmanlı özellik öğrenme
- Model, yalnızca özelliklerin izin verdiği kadar iyi olabilir

Derin Öğrenme Nedir ?

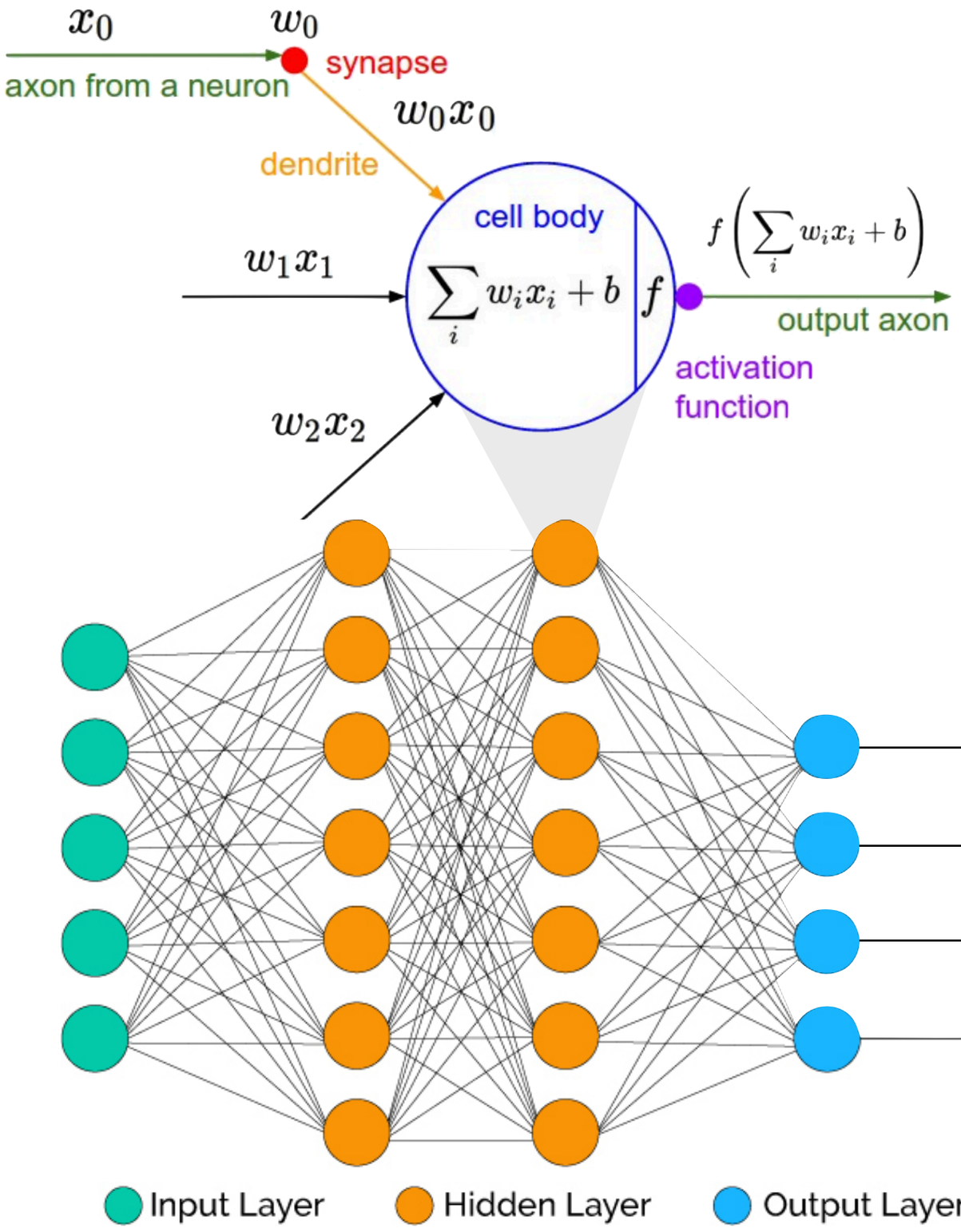
Burada:

- x : Girdi vektörü
- W : Ağırlık matrisi
- b : Bias terimi
- $f(\cdot)$: Aktivasyon fonksiyonu



Model inşa etmek için ne gerekli ?

- Etiketlenmiş veri (zorunlu değil)
- Girdileri başlatan ağırlıklara ihtiyacım var.
- Katmanlar arasındaki bağların dağılımını tanımlamam gerekiyor
- Her bir hücrenin icindeki fonksiyonu secmem gerekiyor.
- Son katmandaki fonksiyonları ayrica tanimlamaliyim
- Kayıp fonksiyonu belirlemeliyim
- Optimizasyon algoritması secmem gerekiyor



Derin öğrenmede temel görevler

- Sınıflandırma (Classification)
- Regresyon (Regression)
- Kümeleme (Clustering)
- Anomali Tespiti (Anomaly Detection)
- Görüntü ve Ses İşleme (Image and Speech Processing)
- Doğal Dil İşleme (Natural Language Processing - NLP)
- Zaman Serisi Tahmini (Time Series Forecasting)

Temel bilgiler

Aktivasyon fonksiyonları

Neden Aktivasyon Fonksiyonları?

- Doğrusal olmayan dönüşümler sağlar.
- Katmanların daha karmaşık özellikleri öğrenmesine olanak tanır.

Önemli Aktivasyon Fonksiyonları:

- **Sigmoid**

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

  - 0 ile 1 arasında değer alır.
  - Küçük değişimlere duyarlı ancak **gradyan kaybolma** problemi yaşanabilir.
- **ReLU (Rectified Linear Unit)**

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

  - Negatif değerleri sıfırlar, pozitifleri olduğu gibi bırakır.
  - Daha hızlı öğrenme sağlar ve gradyan kaybolma problemini azaltır.
- **Softmax** (Çok sınıflı sınıflandırmalarda kullanılır)

$$\text{softmax}(x_i) = \frac{e^{xi}}{\sum_j e^{xj}}$$

  - Çıktıları olasılıklara dönüştürür (toplamı 1 olacak şekilde normalizasyon yapar).

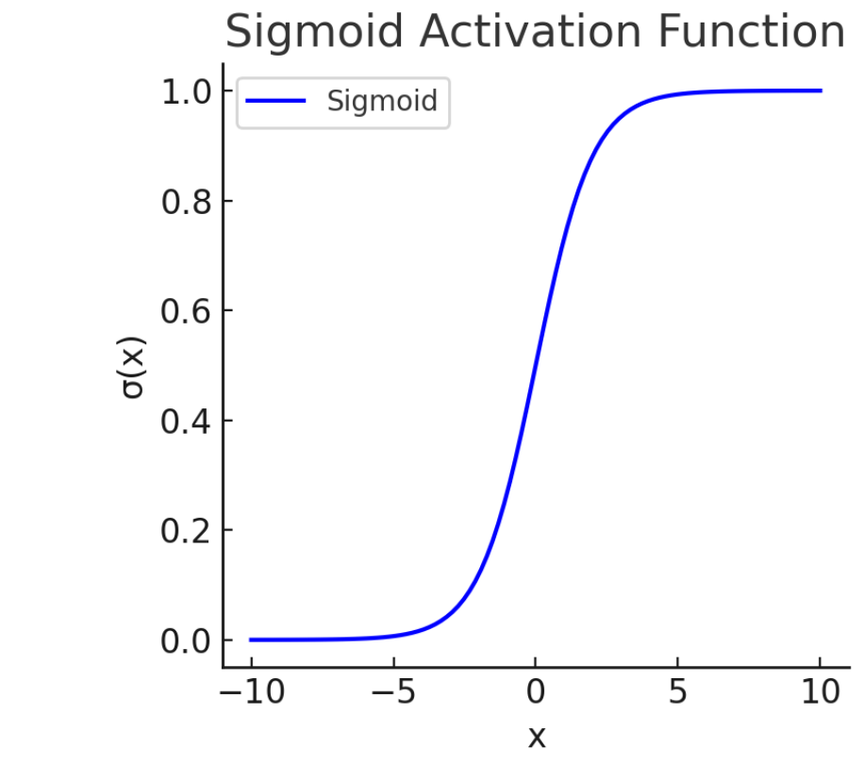
Aktivasyon fonksiyonları - python

```python
import numpy as np

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

# ReLU Function
def relu(x):
    return np.maximum(0, x)

# Softmax Function
def softmax(x):
    e_x = np.exp(x - np.max(x))
    return e_x / e_x.sum()

# Generate input range
x = np.linspace(-10, 10, 100)

# Compute function values
y_sigmoid = sigmoid(2)
y_relu = relu(2)
y_softmax = softmax(2)

print(y_sigmoid , y_relu, y_softmax)
```

Aktivasyon fonksiyonları - python

```python
import numpy as np

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

# ReLU Function
def relu(x):
    return np.maximum(0, x)

# Softmax Function
def softmax(x):
    e_x = np.exp(x - np.max(x))
    return e_x / e_x.sum()

# Generate input range
x = np.linspace(-10, 10, 100)

# Compute function values
y_sigmoid = sigmoid(2)
y_relu = relu(2)
y_softmax = softmax(2)

print(y_sigmoid , y_relu, y_softmax)
```

Aktivasyon fonksiyonları - python



Burada, **Sigmoid, ReLU ve Softmax aktivasyon fonksiyonlarının grafiklerini** oluşturdum:

- **Sigmoid**: Küçük girişlerde neredeyse sıfıra, büyük girişlerde neredeyse bire yakınsar.
- **ReLU**: Negatif değerleri sıfıra kırpar, pozitif değerleri değiştirmez.
- **Softmax**: Çıktıları olasılıklara dönüştürerek toplamı 1 yapar.

Geri yayılım Algoritması ve Optimizasyon (Back propagation)

Nasıl Çalışır?

- **İleri yayılım:** Girdi, ağdan geçirilerek çıktı hesaplanır.
- **Hata hesaplama:** Tahmin edilen ve gerçek değer arasındaki fark hesaplanır.
- **Geri yayılım:** Zincir kuralı kullanılarak hatanın her ağırlık üzerindeki türevleri hesaplanır.
- **Ağırlık güncelleme:** Gradyan iniş algoritması (Gradient Descent) kullanılarak ağın ağırlıkları güncellenir.

Minimizasyon (en küçükleme) deyince aklınıza ne geliyor ?

Geri yayılım Algoritması ve Optimizasyon (Back propagation)

🔷 Gradyan İniş Güncelleme Kuralı:

$$W = W - \eta \frac{\partial L}{\partial W}$$

Burada:

- W : Güncellenen ağırlıklar
- η (eta) : Öğrenme oranı
- L : Kayıp fonksiyonu

Optimizasyon Algoritmaları

- **SGD (Stochastic Gradient Descent)**
- **Adam (Adaptive Moment Estimation)** (En çok kullanılan, hızlı yakınsama sağlar)
- **RMSProp (Root Mean Square Propagation)**

Geri yayılım Algoritması ve Optimizasyon $y = \sigma(wx + b)$

Eğer **kayıp fonksiyonu** $L = (y - y_{true})^2$ yani **ortalama kare hata (MSE)** olsun.

Ağırlık w için türev almak istediğimizde:

1️⃣ **Kayıp fonksiyonunun çıkışa göre türevi:**

$$\frac{\partial L}{\partial y} = 2(y - y_{true})$$

2️⃣ **Çıkışın gizli katmana göre türevi (aktivasyon fonksiyonu türevi):**

$$\frac{\partial y}{\partial h} = \sigma'(h)$$

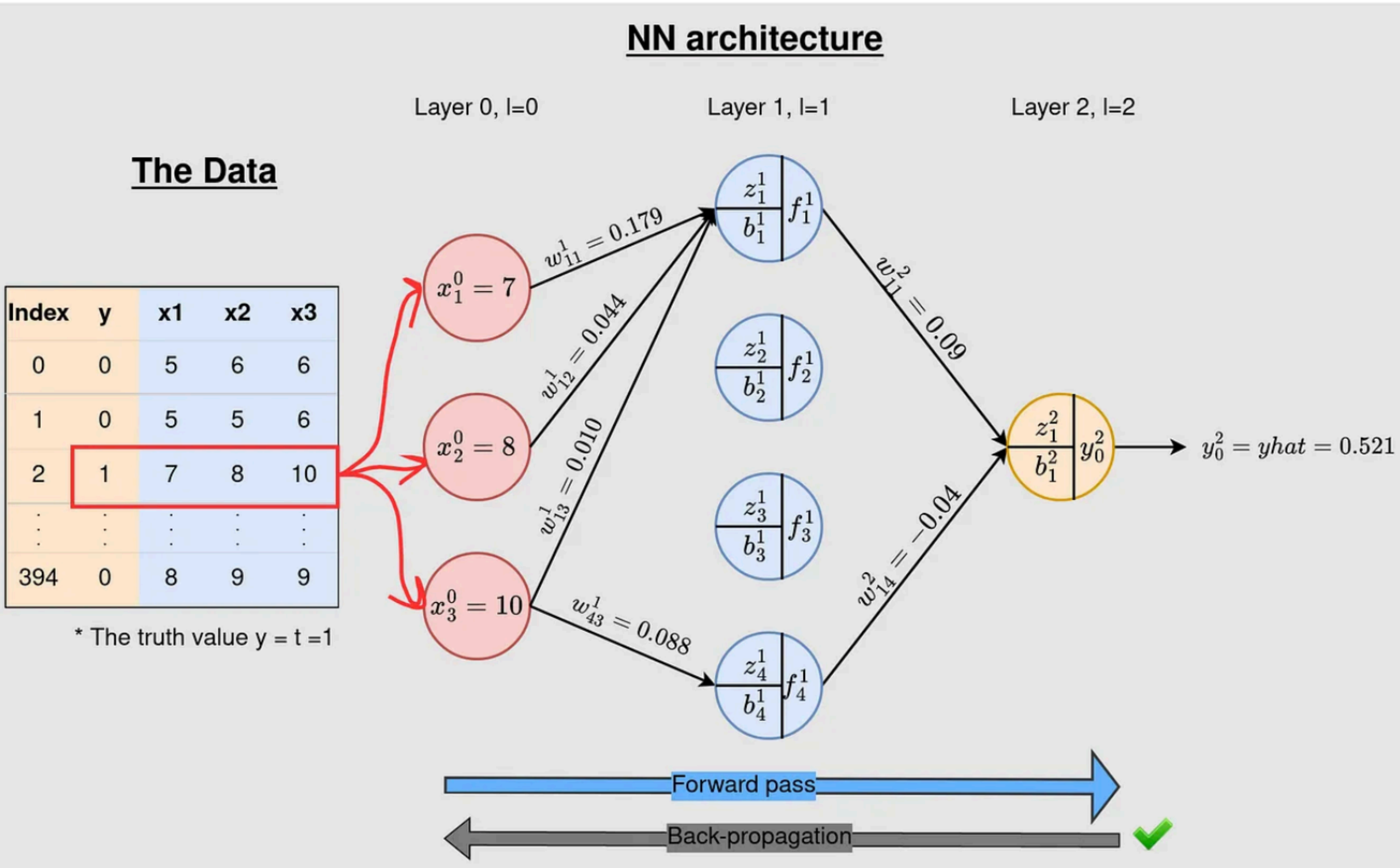
3️⃣ **Gizli katmanın ağırlığa göre türevi:**

$$\frac{\partial h}{\partial w} = x$$

Sonuç olarak:

$$\frac{\partial L}{\partial w} = 2(y - y_{true}) \cdot \sigma'(h) \cdot x$$

- Geri yayılım Algoritması ve Optimizasyon



🔷 Sol tarafta **veri seti (The Data)** gösteriliyor.

🔷 Her satırda:

- **y**: Gerçek çıktı (etiket)
- **x1, x2, x3**: Modelin giriş özellikleri (features)

İşlenen veri noktası:

- **Seçilen örnek (Index 2)** → $y = 1, x_1 = 7, x_2 = 8, x_3 = 10$
- **Bu örnek sinir ağına giriş olarak veriliyor.**

🔷 **Her giriş nöronu, ilk gizli katmandaki (hidden layer) nöronlara bağlanıyor**.

🔷 **Her bağlantının ağırlığı var:**

- $w_{11}^1 = 0.179$
- $w_{12}^1 = 0.044$
- $w_{13}^1 = 0.010$
- $w_{14}^1 = 0.088$

📌 **Nöronların hesaplama adımı:**

Her nöron için şu formül uygulanıyor:

$$z = \sum (w_i \cdot x_i) + b$$

Burada:

- w_i → Ağırlıklar
- x_i → Giriş özellikleri
- b → Bias terimi

🔷 Gizli katmandaki **çıktılar ikinci katmana iletiliyor**.

🔷 **Bağlantı ağırlıkları (w)** ile tekrar çarpılıyor:

- $w_{11}^2 = 0.09$
- $w_{14}^2 = -0.04$

🔷 **Çıktı nöronu (y hat = 0.521)** hesaplanıyor.

- **y hat ($\hat{y}$)** → Modelin tahmini sonucu.
- Burada $\hat{y} = 0.521$ olarak hesaplanmış.
- Gerçek değer $y = 1$ idi.

🔷 **Hata hesaplanıyor:**

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

- Gerçek değer $y = 1$ ve model tahmini $\hat{y} = 0.521$
- **Hata (L) büyükse, model öğrenmesi için geri yayılım başlar.**

🔷 **Geri yayılım başlar:**

- Hatanın türevi hesaplanır.
- Hata, **zincir kuralı ile geriye doğru yayılır**.
- **Ağırlıklar güncellenir**:

$$w = w - \eta \frac{\partial L}{\partial w}$$

  - η → Öğrenme oranı
  - $\frac{\partial L}{\partial w}$ → Hatanın ağırlıklara etkisi

Teşekkürler