9.HAFTA

# YAPAY SİNİR AĞLARI

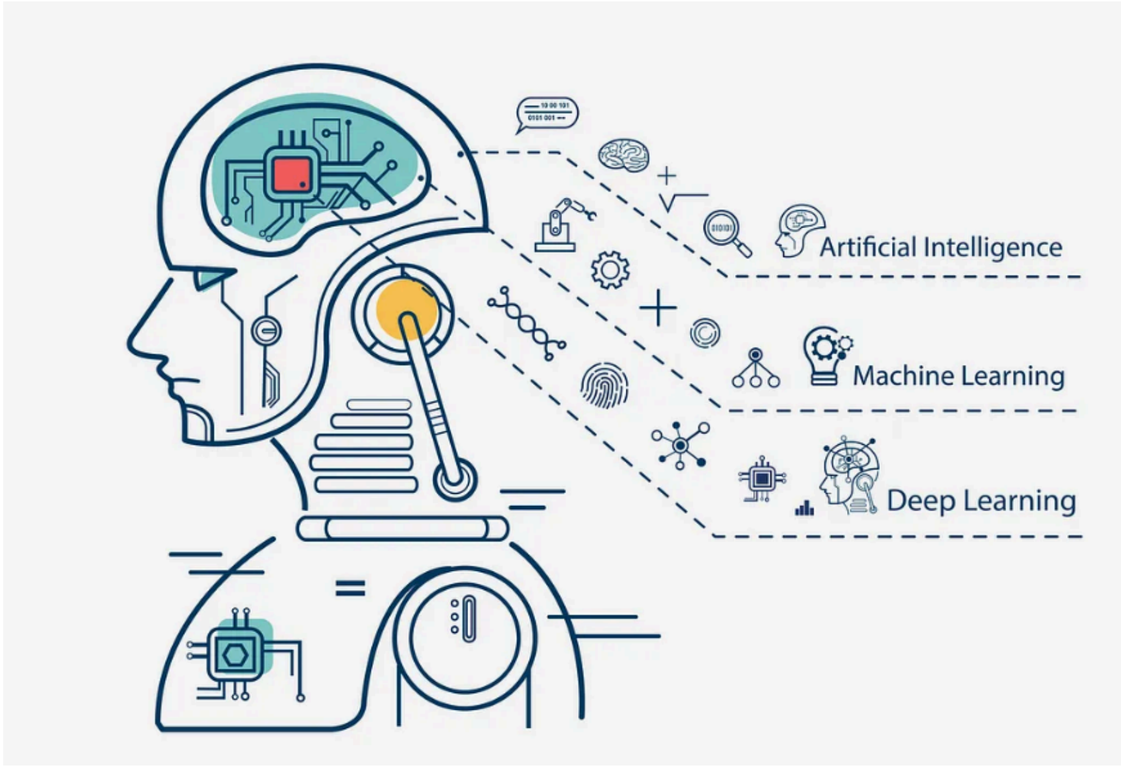
DR. ÖĞR. ÜYESİ ALPER TALHA KARADENİZ

SAMSUN ÜNİVERSİTESİ

# DERIN ÖĞRENME :

Derin Öğrenme (Deep Learning), makine öğrenmesinin bir alt dalı olan ve verilerden karmaşık desenler çıkarmak için yapay sinir ağlarını (Artificial Neural Networks – ANN's) kullanan bir alandır. "Derin" terimi, bu ağların birden fazla ("derin") gizli katmana sahip olmasından gelir.

- Geleneksel algoritmaların çözemediği (nesne tanıma, doğal dil işleme, ses tanıma gibi) son derece karmaşık problemleri çözebilir.



# Temel Kavramlar ve Yapı Taşları :

## 1. Yapay Sinir Ağı (YSA - ANN) Anatomisi :

Temel bir sinir ağı üç katmandan oluşur:

- Girdi Katmanı (Input Layer): Verinin ağa verildiği katman. Her bir düğüm (nöron) bir özelliği (feature) temsil eder.

- Gizli Katman(lar) (Hidden Layer(s)): Veri üzerindeki hesaplamaların ve öğrenmenin asıl gerçekleştiği katmanlar. Derin öğrenmede bu katman sayısı çok fazla olabilir (100+).

- Çıktı Katmanı (Output Layer): Ağın nihai tahmininin veya sınıflandırmasının yapıldığı katman. Problem tipine göre (ikili sınıflandırma, çoklu sınıflandırma, regresyon) aktivasyon fonksiyonu ve nöron sayısı değişir.

## 2 . Bir Nöronun (Aktivasyon) Matematigi

Bir nöron, girdileri alır, ağırlıklı toplamını hesaplar, bir bias (önyargı) ekler ve bir aktivasyon fonksiyonundan geçirir.

- Hesaplama: `z = (w1*x1 + w2*x2 + ... + wn*xn) + b`
- Aktivasyon: `a = f(z)`
  - `w` : Ağırlık (Weight), öğrenilmesi gereken parametre.
  - `x` : Girdi değeri (Input).
  - `b` : Bias terimi.
  - `f` : Aktivasyon Fonksiyonu.

## 3. Aktivasyon Fonksiyonları (Activation Functions)

Ağa doğrusal olmama (non-linearity) ekleyerek karmaşık fonksiyonları modellemeyi mümkün kılar.

- **Sigmoid:** `f(z) = 1 / (1 + e^{-z})`. (0-1 arası çıktı verir, eskiden yaygındı.)
- **Tanh (Hiperbolik Tanjant):** `f(z) = (e^z - e^{-z}) / (e^z + e^{-z})`. (-1 ile 1 arası çıktı verir, ortalaması 0'a yakın olduğu için Sigmoid'den genelde daha iyidir.)
- **ReLU (Rectified Linear Unit):** `f(z) = max(0, z)`. **Günümüzün en popüler fonksiyonu.** Hesaplaması çok hızlıdır ve gradyan kaybı (vanishing gradient) problemine karşı daha dayanıklıdır. Dezavantajı: Negatif girdilerde tamamen 0 olur ("Ölü ReLU" problemi).
- **Leaky ReLU:** Negatif girdiler için çok küçük bir eğim ( `0.01z` gibi) ekleyerek "Ölü ReLU" problemini çözmeye çalışır.
- **Softmax:** Çıktı katmanında, **çoklu sınıflandırma** problemleri için kullanılır. Tüm nöronların çıktılarını olasılık dağılımına çevirir (toplamları 1 olur).

# 4. Kayıp (Loss) ve Maliyet (Cost) Fonksiyonları

Modelin tahminlerinin gerçek değerlerden ne kadar "uzak" olduğunu ölçer. Amacımız bu maliyeti minimize etmektir.

- **Ortalama Karesel Hata (MSE - Mean Squared Error)**: Regresyon problemleri için.
- **Çapraz Entropi (Cross-Entropy)**: Sınıflandırma problemleri için. İkili (Binary) ve Kategorik (Categorical) çeşitleri vardır.

# 5. Gradyan İnişi (Gradient Descent) ve Geri Yayılım (Backpropagation)

- **Gradyan İnişi:** Maliyet fonksiyonunu minimize etmek için kullanılan optimizasyon algoritması. Fonksiyonun eğimine (gradyan) bakarak parametreleri (ağırlık ve bias'ları) günceller.
  - **Öğrenme Oranı (Learning Rate):** Gradyan yönünde her adımda ne kadar ilerleneceğini belirleyen **çok önemli** bir hiper-parametredir. Çok büyükse optimum noktayı ıskalar, çok küçükse çok yavaş öğrenir.
- **Geri Yayılım (Backpropagation):** Gradyanı hesaplamak için kullanılan **algoritmadır**. Hatayı çıktı katmanından girdi katmanına doğru zincir kuralı (chain rule) ile yayar. Böylece her bir parametrenin hataya ne kadar katkıda bulunduğu hesaplanır ve Gradyan İnişi ile güncellenir.

## 6. Overfitting (Aşırı Öğrenme) ve Regularization (Düzenlileştirme)

Modelin eğitim verisine çok iyi uyum sağlayıp, görmediği verilerde (test seti) kötü performans göstermesidir.

1. **Dropout:** Eğitim sırasında rastgele nöronları devre dışı bırakır. Ağ, hiçbir nörona aşırı güvenemez, daha robust (sağlam) öğrenir.
2. **L1/L2 Regularization:** Ağırlıklara bir ceza terimi ekleyerek çok büyük değerlere ulaşmalarını engeller. L1 bazı ağırlıkları tamamen 0 yaparak özellik seçimi yapar, L2 ise ağırlıkları küçültür.
3. **Erken Durdurma (Early Stopping):** Validation (doğrulama) hatası artmaya başladığı anda eğitimi durdurur.
4. **Veri Arttırımı (Data Augmentation):** Özellikle görüntü işlemede, mevcut veriyi döndürme, ölçeklendirme, aynalama gibi işlemlerle çoğaltmak.

# Gelişmiş Ağ Mimarileri

Yapay sinir ağlarının temel yapıları (örneğin ileri beslemeli ağlar – feedforward neural networks, MLP) birçok problemde başarılı olsa da, daha karmaşık veri türleri (görüntü, ses, metin, zaman serisi vb.) için yeterli değildir. Bu nedenle, farklı ihtiyaçlara uygun olarak geliştirilmiş ileri düzey (gelişmiş) ağ mimarileri ortaya çıkmıştır.

Bu mimariler, özel katmanlar, bellek mekanizmaları veya dikkat (attention) yöntemleri kullanarak performansı artırır.

## 1. Evrişimsel Sinir Ağları (Convolutional Neural Networks – CNN)

- Görüntülerdeki yerel özellikleri (kenar, köşe, doku gibi) öğrenmek için geliştirilmiştir.
- Küçük filtreler (kernel) ile görüntü üzerinde kaydırma işlemi yapılır ve önemli özellikler çıkarılır.
- İlk katmanlarda basit şekiller, daha derin katmanlarda ise karmaşık nesneler öğrenilir.
- Kullanım alanları: Görüntü sınıflandırma, nesne tespiti, yüz tanıma, tıbbi görüntü analizi.

## 2. Tekrarlayan Sinir Ağları (Recurrent Neural Networks – RNN)

- Sıralı verilerde geçmiş bilgiyi hatırlayarak geleceği tahmin etmeye uygundur.
- Bir önceki adımdaki çıktı, sonraki adımlara aktarılır.
- Uzun bağımlılık sorununu çözmek için gelişmiş türleri vardır:
  - LSTM (Long Short-Term Memory)
  - GRU (Gated Recurrent Unit)
- Kullanım alanları: Metin işleme, konuşma tanıma, zaman serisi tahmini, duygu analizi.

## 3. Transformer Mimarisi

- Sıralı verilerde özellikle metin işlemede kullanılan modern bir mimaridir.
- "Dikkat (attention) mekanizması" sayesinde uzun cümlelerdeki bağlamı korur.
- RNN'den farklı olarak veriyi paralel işler, bu nedenle daha hızlıdır.
- ChatGPT, BERT gibi modellerin temelinde bu mimari vardır.
- Kullanım alanları: Çeviri, sohbet botları, metin üretimi, bilgi çıkarımı.

## 4. Otoenkoderler (Autoencoders)

- Veriyi sıkıştırma ve yeniden üretme amacıyla geliştirilmiştir.
- Encoder bölümü veriyi düşük boyuta indirger, Decoder bölümü ise tekrar açar.
- Gizli katmanlarda verinin önemli özellikleri öğrenilir.
- Kullanım alanları: Gürültü giderme, boyut indirgeme, anomali tespiti.

## 5. Üretici Çekişmeli Ağlar (Generative Adversarial Networks – GAN)

- İki ağdan oluşur:
  - Üretici (Generator) → Sahte veri üretir.
  - Ayırt edici (Discriminator) → Gerçek ve sahteyi ayırt etmeye çalışır.
- Bu iki ağın rekabeti sayesinde üretici giderek daha gerçekçi veriler üretir.
- Kullanım alanları: Görsel ve video üretimi, deepfake, sanatsal içerik üretimi, veri artırma.

## 6. Kapsül Ağlar (Capsule Networks – CapsNet)

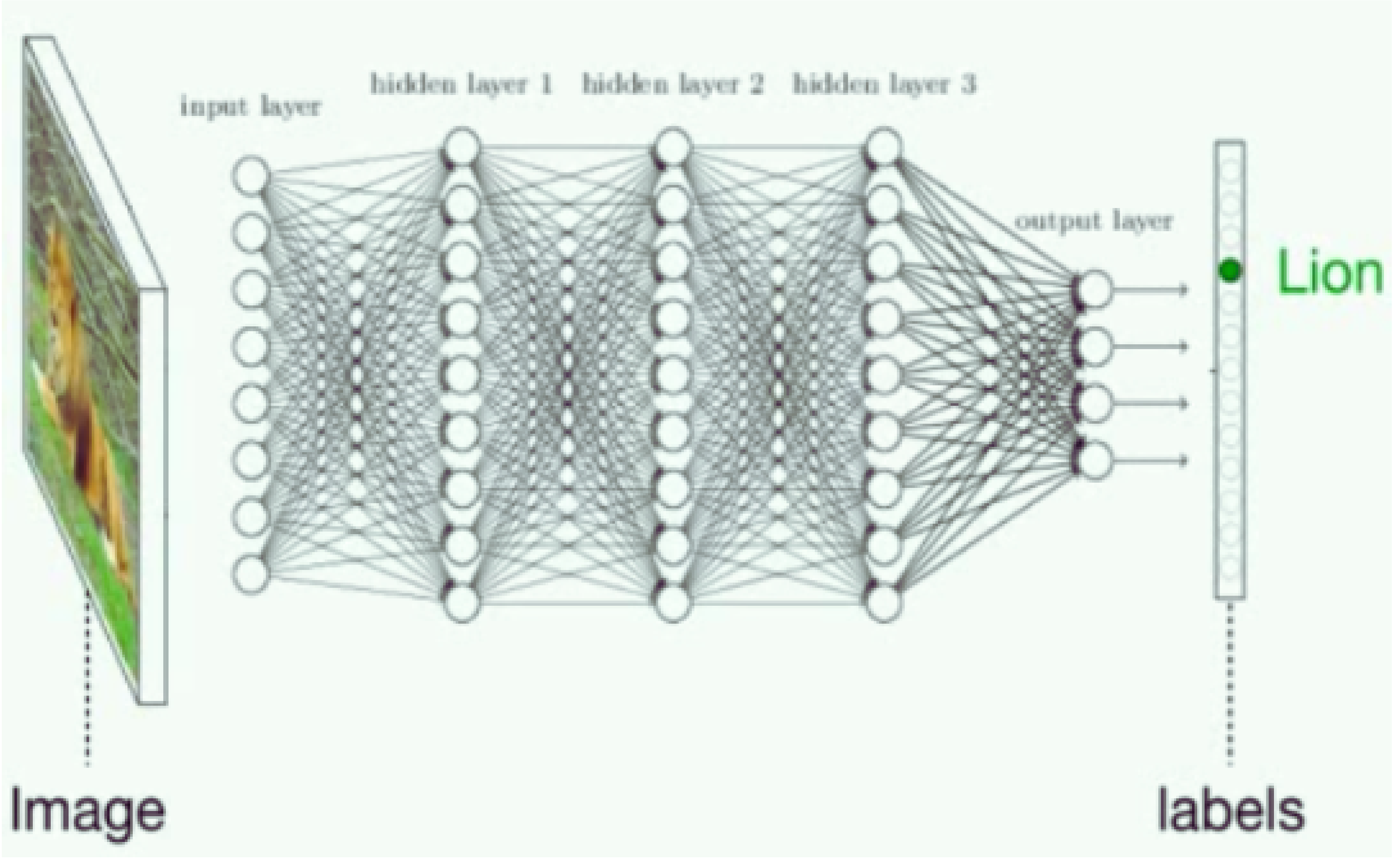
- CNN'lerin eksikliklerini gidermek için geliştirilmiştir.
- Nesnelerin parçaları arasındaki uzamsal ilişkileri daha iyi öğrenir.
- Örneğin: Bir yüzün parçaları yanlış yerde olsa bile, CapsNet doğru yorum yapabilir.
- Henüz araştırma aşamasındadır, pratikte yaygın değildir.

# Derin Ağlar (Deep Neural Networks – DNN)

Derin Ağlar, yapay sinir ağlarının **birden fazla gizli katmana** sahip olan türüdür.

- Basit bir sinir ağı (shallow neural network) yalnızca bir gizli katmana sahiptir.
- Derin sinir ağlarında ise giriş ile çıkış arasında **birden fazla gizli katman** bulunur.

Bu katmanlar sayesinde model, **karmaşık ilişkileri** ve **soyut özellikleri** öğrenebilir.


input layer
hidden layer 1
hidden layer 2
hidden layer 3
output layer
Lion
Image
labels

## Yapısı

Bir DNN şu bölümlerden oluşur:

1. **Giriş Katmanı (Input Layer):** Verinin ağa verildiği katmandır.
2. **Gizli Katmanlar (Hidden Layers):** Veriyi dönüştüren ve özellik çıkarımı yapan katmanlardır. Birden fazla gizli katman DNN'i "derin" yapar.
3. **Çıkış Katmanı (Output Layer):** Sonucun elde edildiği katmandır. Sınıflandırmada genellikle softmax veya sigmoid aktivasyonu kullanılır.

## Derinlik Neden Önemlidir?

- Tek katmanlı ağlar yalnızca **basit doğrusal ilişkileri** öğrenebilir.
- Çok katmanlı (derin) ağlar, verideki **karmaşık ve doğrusal olmayan ilişkileri** öğrenebilir.
- Her katman bir "soyutlama düzeyi" öğrenir:
  - İlk katmanlar basit özellikleri (ör. bir görüntüde kenarlar),
  - Orta katmanlar daha karmaşık yapıları (ör. şekiller),
  - Son katmanlar ise üst düzey kavramları (ör. yüz, araba, kedi) öğrenir.

## Çalışma Prensibi

1. Giriş verisi ilk katmana verilir.
2. Katmanlar boyunca veriye **ağırlıklar ve aktivasyon fonksiyonları** uygulanarak dönüşümler yapılır.
3. Son katmanda elde edilen çıktı, tahmin veya karar olarak alınır.
4. Eğitim sırasında **geri yayılım (backpropagation)** ve **gradyan inişi (gradient descent)** ile ağın ağırlıkları güncellenir.

## Avantajları

- Karmaşık verilerde yüksek doğruluk sağlar.
- Görüntü, ses, metin gibi farklı veri türlerinde esnek şekilde kullanılabilir.
- Özellik çıkarımını otomatik yapar (elle özellik seçmeye gerek kalmaz).

---

## Dezavantajları

- Çok sayıda parametre içerdiği için **hesaplama maliyeti yüksektir**.
- Aşırı öğrenmeye (overfitting) eğilimlidir.
- Büyük miktarda veri ve güçlü donanım (GPU/TPU) gerektirir.

## Kullanım Alanları

- Görüntü tanıma (ör. yüz tanıma, nesne algılama)
- Ses işleme (ör. konuşma tanıma)
- Doğal dil işleme (ör. metin sınıflandırma, çeviri)
- Oyun ve robot kontrolü
- Finans, sağlık, biyoinformatik gibi alanlarda tahmin ve analiz

# Parametre Paylaşımı ve Hesaplama Karmaşıklığı

Derin öğrenmede özellikle **CNN (Convolutional Neural Networks)** ve diğer gelişmiş ağlarda iki önemli kavram vardır: **parametre paylaşımı** ve **hesaplama karmaşıklığı**. Bu kavramlar, ağın verimliliğini ve performansını doğrudan etkiler.

## 1. Parametre Paylaşımı

Parametre paylaşımı, bir ağırlık setinin **birden fazla giriş veya konum için tekrar kullanılması** demektir.

**Örnek**

- CNN'de bir filtre (kernel) tüm görüntü üzerinde kaydırılır.
- Bu filtredeki ağırlıklar **her pozisyonda aynı şekilde uygulanır**.
- Böylece her konum için ayrı ağırlık öğrenmeye gerek kalmaz.

### Avantajları

- **Parametre sayısını azaltır:** Tek bir filtre, tüm görüntüye uygulanır.
- **Daha hızlı eğitim:** Daha az parametre, daha az hesaplama.
- **Genelleme kabiliyetini artırır:** Aynı özellik farklı konumlarda öğrenilmiş olur.

# 2. Hesaplama Karmaşıklığı

Hesaplama karmaşıklığı, bir ağın **işlem yaparken harcadığı zaman ve kaynak miktarı** ile ilgilidir.

### Örnek

- Tam bağlantılı (fully connected) bir katmanda her nöron tüm girişlerle bağlantılıdır.
- Giriş boyutu büyükse, ağırlık sayısı çok fazla olur ve hesaplama maliyeti artar.
- CNN'de parametre paylaşımı ve küçük filtreler sayesinde **hesaplama maliyeti önemli ölçüde azalır.**

## Hesaplama karmaşıklığını etkileyen faktörler

1. **Katman sayısı ve nöron sayısı** → Ne kadar fazla, o kadar maliyetli.
2. **Filtre boyutu ve sayısı (CNN'de)** → Daha büyük veya daha fazla filtre, daha çok hesaplama.
3. **Giriş veri boyutu** → Daha büyük görüntü veya uzun diziler daha fazla işlem gerektirir.
4. **Paralel işleme imkanları** → GPU/TPU gibi donanımlar karmaşıklığı azaltabilir.

# Katman Tipleri: Dense, BatchNormalization, Dropout

Yapay sinir ağlarında farklı katman tipleri, modelin öğrenme kapasitesini, kararlılığını ve genelleme yeteneğini etkiler. En sık kullanılan katman tipleri şunlardır:

## 1. Dense Katmanı (Fully Connected Layer)

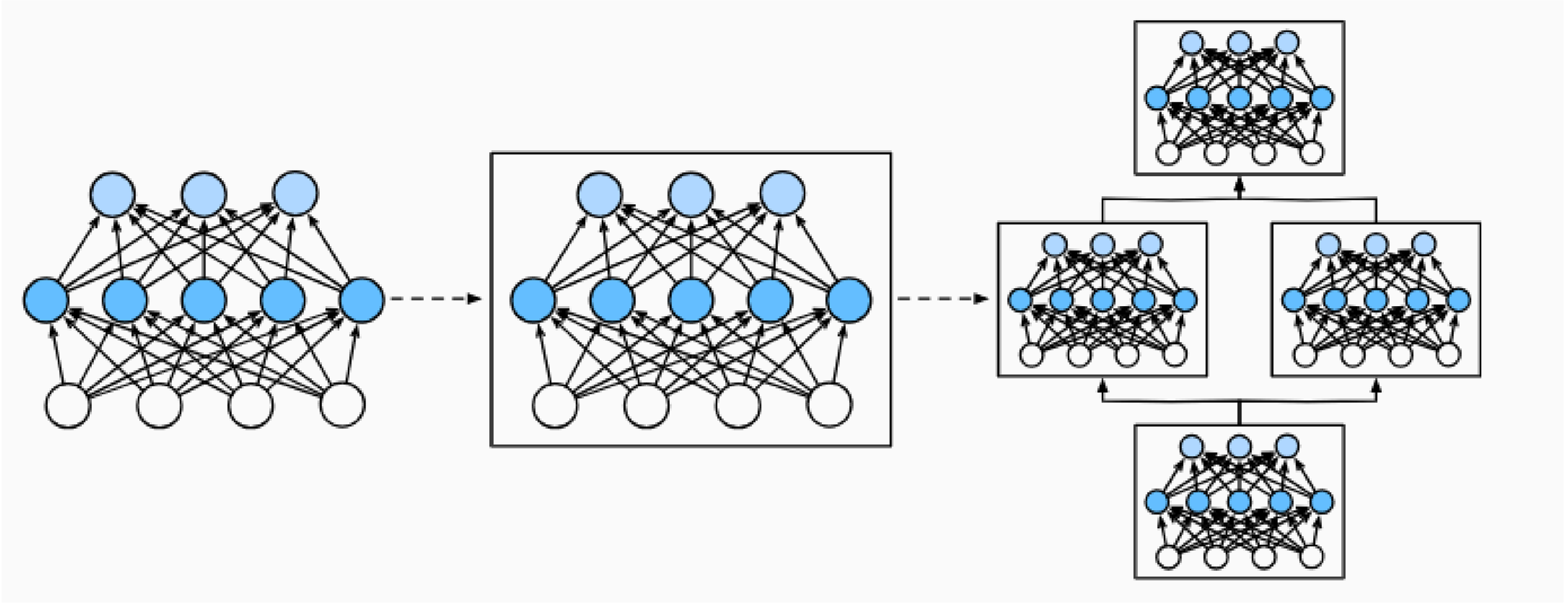
- Dense katman, her nöronun **bir önceki katmandaki tüm nöronlara bağlı olduğu** katmandır.
- Her bağlantı bir **ağırlık (weight)** ile çarpılır ve sonuç aktivasyon fonksiyonu ile işlenir.

### Görevleri

- Veriyi dönüştürmek
- Özellikler arasındaki karmaşık ilişkileri öğrenmek

### Kullanım Alanları

- Çıkış katmanı (output) olarak sınıflandırma veya regresyon problemlerinde
- Gizli katmanlarda soyut özellik çıkarımı



# 2. BatchNormalization Katmanı

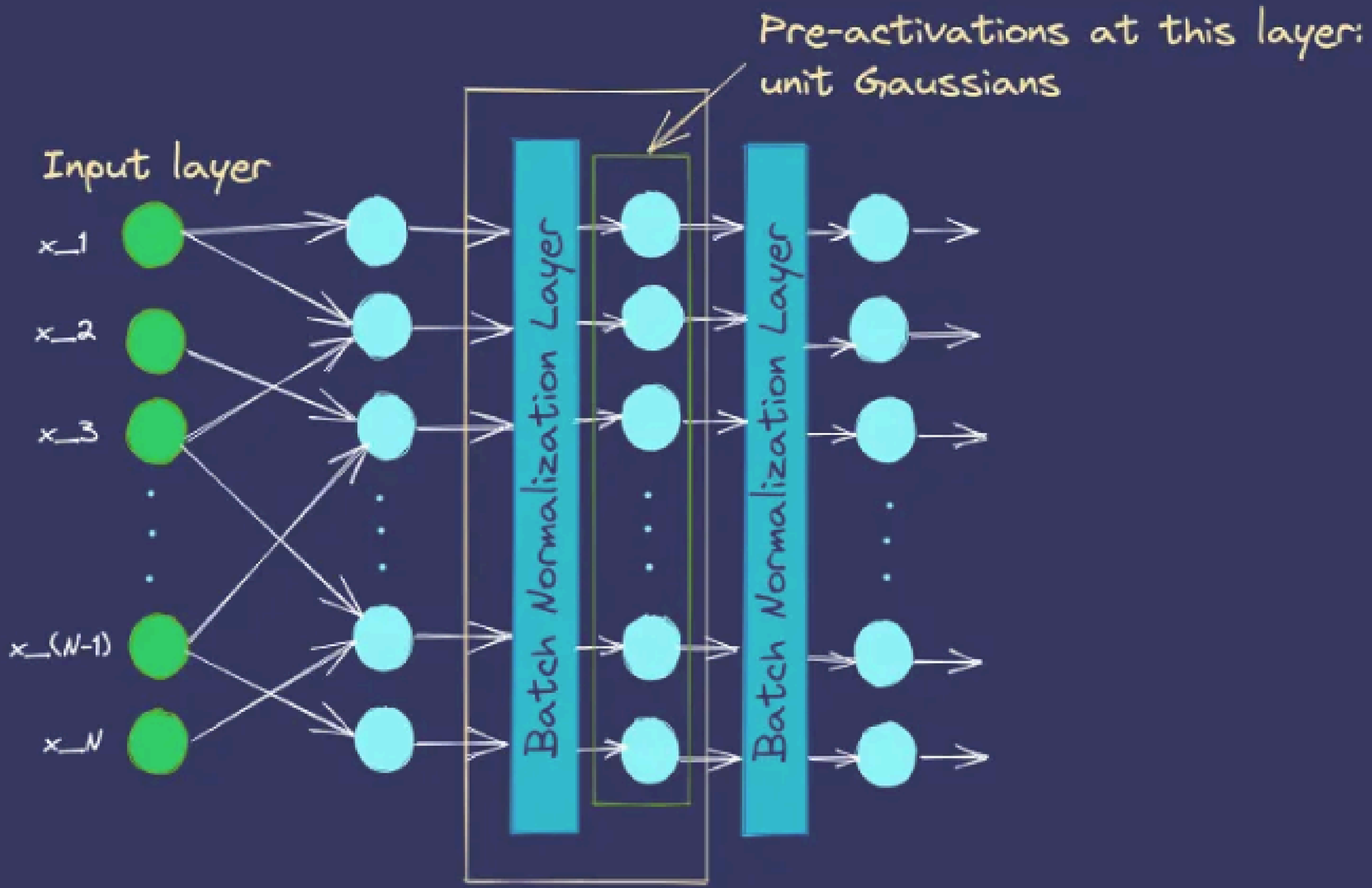
- BatchNormalization, **her katmandaki girişlerin normalizasyonunu (ortalama=0, varyans=1) yaparak** ağırlık güncellemelerini stabilize eden bir katmandır.

## Avantajları

- Ağın **daha hızlı ve kararlı öğrenmesini sağlar**
- Aktivasyonların aşırı büyümesini veya küçülmesini engeller (vanishing/exploding gradient sorununu azaltır)
- Daha yüksek öğrenme oranları kullanılmasına izin verir

## Kullanım Alanları

- Derin ağlarda gizli katmanlar arasında
- Convolutional ve dense katmanlarla birlikte


Pre-activations at this layer:
unit Gaussians
Input layer
x_1
x_2
x_3
x_(N-1)
x_N
Batch Normalization Layer
Batch Normalization Layer

# 3. Dropout Katmanı

- Dropout, eğitim sırasında bazı nöronları **rastgele devre dışı bırakan** bir katmandır.
- Amaç: Ağın **genelleme yeteneğini artırmak ve aşırı öğrenmeyi (overfitting) önlemektir**

## Çalışma Prensibi

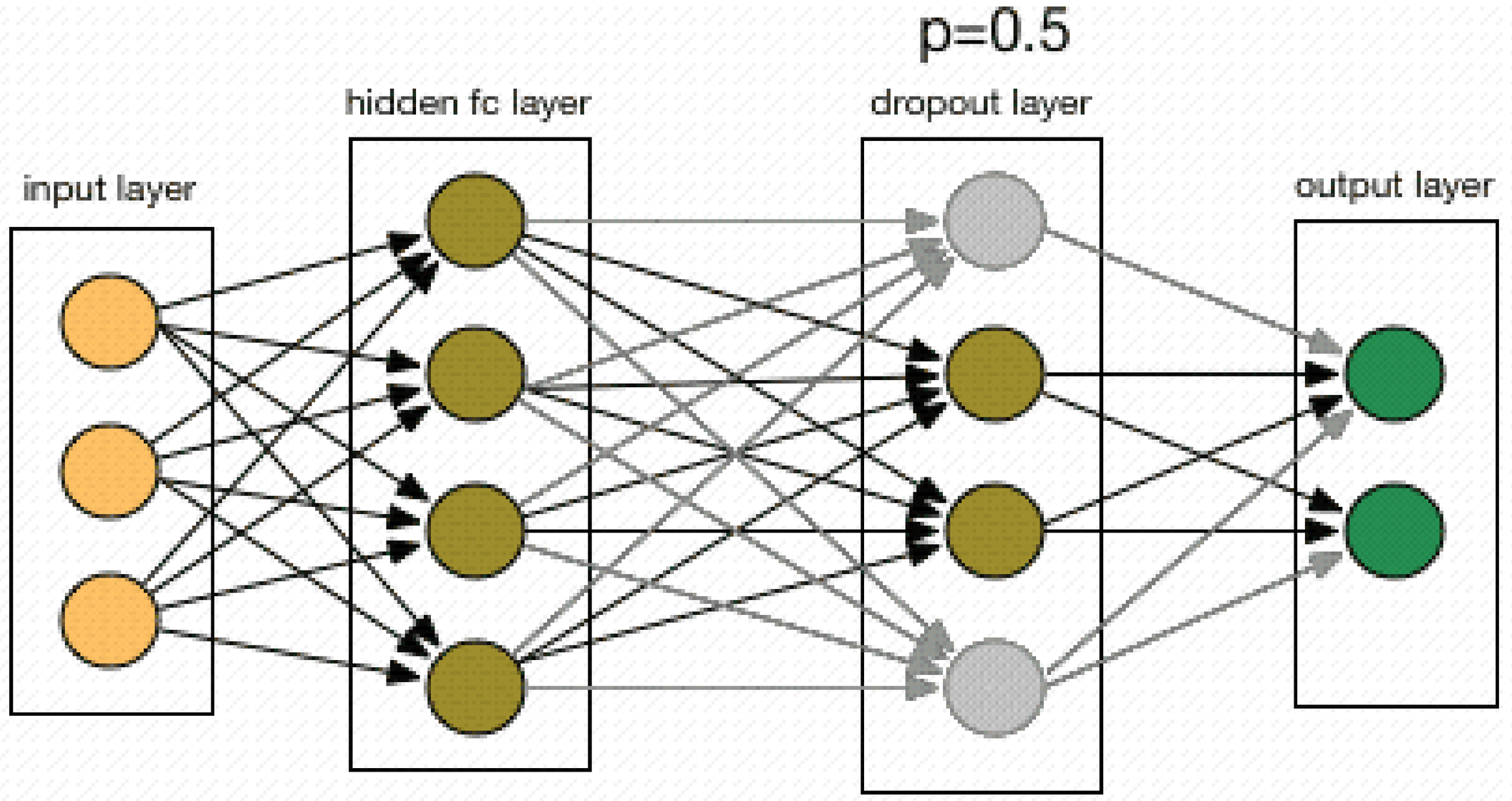
- Eğitim sırasında belirli bir olasılıkla nöronlar geçici olarak kapatılır
- Test aşamasında tüm nöronlar aktif şekilde kullanılır ve ağırlıklar buna göre ölçeklenir

## Avantajları

- Overfitting'i azaltır
- Modelin farklı nöron kombinasyonlarını öğrenmesini sağlar
- Daha sağlam ve genelleyici modeller oluşturur

## Kullanım Alanları

- Gizli katmanlarda sık kullanılır
- Özellikle veri seti küçük veya model çok büyükse faydalıdır


p=0.5
hidden fc layer
dropout layer
input layer
output layer

# Uygulama: Gelişmiş DNN mimarisi ile sınıflandırma örneği

```
# Temel kütüphaneler
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers, models
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Veri işleme ve değerlendirme
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, confusion_matrix

# Görselleştirme ayarları
plt.style.use('seaborn-v0_8')
sns.set_palette("husl")
%matplotlib inline
```

# Veri Kümesinin Yüklenmesi ve İncelenmesi

```
# Fashion MNIST veri setini yükleme
(X_egitim, y_egitim), (X_test, y_test) = keras.datasets.fashion_mnist.load_data()

# Veri seti hakkında bilgi
print("╔═══════════════════════════════════════════════════╗")
print("║           VERİ SETİ BİLGİLERİ                     ║")
print("╠═══════════════════════════════════════════════════╣")
print(f"║ Eğitim verisi şekli:    {X_egitim.shape}       ║")
print(f"║ Test verisi şekli:      {X_test.shape}         ║")
print(f"║ Sınıf sayısı:           {len(np.unique(y_egitim))}           ║")
print("╚═══════════════════════════════════════════════════╝")

# Sınıf isimleri
sinif_isimleri = [
    'Tişört/Üst', 'Pantolon', 'Kazak', 'Elbise', 'Ceket',
    'Sandalet', 'Gömlek', 'Spor Ayakkabı', 'Çanta', 'Bilekte Bot'
]
```

# Veri Ön İşleme

```
# Veriyi normalize etme (0-1 arası)
X_egitim_normal = X_egitim.astype('float32') / 255.0
X_test_normal = X_test.astype('float32') / 255.0

# Veriyi düzleştirme (28x28 -> 784 piksel)
X_egitim_duz = X_egitim_normal.reshape(-1, 784)
X_test_duz = X_test_normal.reshape(-1, 784)

# Doğrulama seti için ayırma
X_egitim_son, X_dogrulama, y_egitim_son, y_dogrulama = train_test_split(
    X_egitim_duz, y_egitim, test_size=0.2, random_state=42, stratify=y_egitim
)
```

# Model Mimarisi Oluşturma

```
def derin_sinir_agi_olustur(girdi_boyutu, sinif_sayisi):
    """
    Derin sinir ağı modeli oluşturur

    Args:
        girdi_boyutu: Giriş verisinin boyutu
        sinif_sayisi: Çıkış sınıf sayısı

    Returns:
        Derlenmiş keras modeli
    """
    model = models.Sequential([
        # Giriş katmanı
        layers.Dense(256, activation='relu', input_shape=(girdi_boyutu,),
                     kernel_initializer='he_normal'),
        layers.BatchNormalization(),
        layers.Dropout(0.3),

        # 1. Gizli katman
        layers.Dense(128, activation='relu', kernel_initializer='he_normal'),
        layers.BatchNormalization(),
        layers.Dropout(0.3),
```

```python
        # 2. Gizli katman
        layers.Dense(64, activation='relu', kernel_initializer='he_normal'),
        layers.BatchNormalization(),
        layers.Dropout(0.2),

        # Çıkış katmanı
        layers.Dense(sinif_sayisi, activation='softmax')
    ])

    return model

# Modeli oluşturma
girdi_boyutu = 784
sinif_sayisi = 10
model = derin_sinir_agi_olustur(girdi_boyutu, sinif_sayisi)

# Model özetini gösterme
print("╔═══════════════════════════════════════════════╗")
print("║              MODEL MİMARİSİ                   ║")
print("╚═══════════════════════════════════════════════╝")
model.summary()
```

# Model Derleme

```
# Optimizasyon parametreleri
optimizer = keras.optimizers.Adam(
    learning_rate=0.001,
    beta_1=0.9,
    beta_2=0.999,
    epsilon=1e-07
)

# Modeli derleme
model.compile(
    optimizer=optimizer,
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy',
             keras.metrics.SparseTopKCategoricalAccuracy(k=3, name='top_3_accuracy')]
)

print("    Model başarıyla derlendi!")
```

# Model Eğitimi

```
# Callback'ler
cagri_listesi = [
    keras.callbacks.EarlyStopping(
        monitor='val_loss',
        patience=10,
        restore_best_weights=True,
        verbose=1
    ),
    keras.callbacks.ReduceLROnPlateau(
        monitor='val_loss',
        factor=0.5,
        patience=5,
        min_lr=1e-7,
        verbose=1
    )
]
```

```python
# Model eğitimi
print("    Model eğitimi başlıyor...")
tarihce = model.fit(
    X_egitim_son, y_egitim_son,
    batch_size=128,
    epochs=50,
    validation_data=(X_dogrulama, y_dogrulama),
    callbacks=cagri_listesi,
    verbose=1
)

print("    Model eğitimi tamamlandı!")
```

# Model Değerlendirme

```
# Test setinde değerlendirme
test_sonuc = model.evaluate(X_test_duz, y_test, verbose=0)
test_kaybi, test_dogruluk, test_top3_dogruluk = test_sonuc

print("╔═══════════════════════════════════════════════════╗")
print("║                TEST SONUÇLARI                     ║")
print("╠═══════════════════════════════════════════════════╣")
print(f"║ Test Kaybı:            {test_kaybi:.4f}                ║")
print(f"║ Test Doğruluğu:        {test_dogruluk:.4f}                ║")
print(f"║ Top-3 Doğruluk:        {test_top3_dogruluk:.4f}                ║")
print("╚═══════════════════════════════════════════════════╝")

# Tahminler
y_tahmin = model.predict(X_test_duz)
y_tahmin_sinif = np.argmax(y_tahmin, axis=1)
```

# Görselleştirme ve Analiz

```
# Eğitim geçmişi görselleştirme
plt.figure(figsize=(15, 5))

# Doğruluk grafiği
plt.subplot(1, 2, 1)
plt.plot(tarihce.history['accuracy'], label='Eğitim Doğruluğu', linewidth=2)
plt.plot(tarihce.history['val_accuracy'], label='Doğrulama Doğruluğu', linewidth=2)
plt.title('Model Doğruluğu', fontsize=14, fontweight='bold')
plt.xlabel('Epok', fontsize=12)
plt.ylabel('Doğruluk', fontsize=12)
plt.legend()
plt.grid(True, alpha=0.3)

# Kayıp grafiği
plt.subplot(1, 2, 2)
plt.plot(tarihce.history['loss'], label='Eğitim Kaybı', linewidth=2)
plt.plot(tarihce.history['val_loss'], label='Doğrulama Kaybı', linewidth=2)
plt.title('Model Kaybı', fontsize=14, fontweight='bold')
plt.xlabel('Epok', fontsize=12)
plt.ylabel('Kayıp', fontsize=12)
plt.legend()
plt.grid(True, alpha=0.3)

plt.tight_layout()
plt.show()
```

```python
# Karışıklık matrisi
plt.figure(figsize=(12, 10))
cizim = confusion_matrix(y_test, y_tahmin_sinif)
sns.heatmap(cizim, annot=True, fmt='d', cmap='Blues',
            xticklabels=sinif_isimleri, yticklabels=sinif_isimleri)
plt.title('Karışıklık Matrisi', fontsize=16, fontweight='bold')
plt.xlabel('Tahmin Edilen Etiket', fontsize=12)
plt.ylabel('Gerçek Etiket', fontsize=12)
plt.xticks(rotation=45)
plt.yticks(rotation=0)
plt.show()

# Sınıflandırma raporu
print("    Sınıflandırma Raporu:")
print(classification_report(y_test, y_tahmin_sinif,
                            target_names=sinif_isim_labels, digits=4))
```

# Örnek Tahminlerin Görselleştirilmesi

```
# Rastgele örneklerle tahmin görselleştirme
plt.figure(figsize=(15, 10))
ornek_sayisi = 12

for i in range(ornek_sayisi):
    plt.subplot(3, 4, i + 1)
    rastgele_indeks = np.random.randint(0, len(X_test))

    # Görüntüyü göster
    plt.imshow(X_test[rastgele_indeks], cmap='gray')

    # Gerçek ve tahmin bilgileri
    gercek_etiket = sinif_isimleri[y_test[rastgele_indeks]]
    tahmin_etiket = sinif_isimleri[y_tahmin_sinif[rastgele_indeks]]
    tahmin_olasilik = np.max(y_tahmin[rastgele_indeks])
```

```
    # Başlık rengini belirle (doğru/yanlış)
    renk = 'green' if y_test[rastgele_indeks] == y_tahmin_sinif[rastgele_indeks] else
'red'

    plt.title(f'Gerçek: {gercek_etiket}\nTahmin: {tahmin_etiket}\nOlasılık: {tahmin_ol
asilik:.2f}',
              color=renk, fontsize=9)
    plt.axis('off')

plt.suptitle('Örnek Tahminler - Yeşil: Doğru, Kırmızı: Yanlış',
             fontsize=16, fontweight='bold')
plt.tight_layout()
plt.show()
```

# Model Kaydetme

```
# Modeli kaydetme
model.save('fashion_mnist_derin_sinir_agi_modeli.h5')
print("   Model 'fashion_mnist_derin_sinir_agi_modeli.h5' olarak kaydedildi!")

# Model mimarisini kaydetme
with open('model_mimarisi.json', 'w') as dosya:
    dosya.write(model.to_json())
print("   Model mimarisi 'model_mimarisi.json' olarak kaydedildi!")
```

# Beklenen çıktılar :

## Eğitim Süreci Çıktısı

```
Model eğitimi başlıyor...
Epoch 1/50
375/375 [==============================] - 5s 10ms/step - loss: 0.6241 - accuracy: 0.7821 - top_3_accuracy:
0.9601 - val_loss: 0.4521 - val_accuracy: 0.8402 - val_top_3_accuracy: 0.9765 - lr: 0.0010
Epoch 2/50
375/375 [==============================] - 4s 9ms/step - loss: 0.4321 - accuracy: 0.8483 - top_3_accuracy:
0.9782 - val_loss: 0.3987 - val_accuracy: 0.8589 - val_top_3_accuracy: 0.9821 - lr: 0.0010
...
Epoch 15/50
375/375 [==============================] - 4s 9ms/step - loss: 0.3124 - accuracy: 0.8876 - top_3_accuracy:
0.9889 - val_loss: 0.3210 - val_accuracy: 0.8854 - val_top_3_accuracy: 0.9891 - lr: 0.0010

Epoch 00015: ReduceLROnPlateau reducing learning rate to 0.0005000000237487257.
...
Epoch 00025: Early stopping conditioned met. Restoring model weights from the end of the best epoch: 15.
```

# Test Sonuçları Çıktısı

```
╔═══════════════════════════════════════════════════╗
║                TEST SONUÇLARI                     ║
╠═══════════════════════════════════════════════════╣
║ Test Kaybı:              0.3456                   ║
║ Test Doğruluğu:          0.8764                   ║
║ Top-3 Doğruluk:          0.9872                   ║
╚═══════════════════════════════════════════════════╝

313/313 [==============================] - 1s 2ms/step
```

# Sınıflandırma Raporu Çıktısı

Sınıflandırma Raporu:

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| Tişört/Üst | 0.8456 | 0.8560 | 0.8508 | 1000 |
| Pantolon | 0.9876 | 0.9760 | 0.9818 | 1000 |
| Kazak | 0.8234 | 0.8450 | 0.8341 | 1000 |
| Elbise | 0.8967 | 0.9010 | 0.8988 | 1000 |
| Ceket | 0.8345 | 0.8560 | 0.8451 | 1000 |
| Sandalet | 0.9765 | 0.9650 | 0.9707 | 1000 |
| Gömlek | 0.7456 | 0.7120 | 0.7284 | 1000 |
| Spor Ayakkabı | 0.9567 | 0.9450 | 0.9508 | 1000 |
| Çanta | 0.9678 | 0.9780 | 0.9729 | 1000 |
| Bilekte Bot | 0.9654 | 0.9620 | 0.9637 | 1000 |
| accuracy | | | 0.8964 | 10000 |
| macro avg | 0.9000 | 0.8996 | 0.8997 | 10000 |
| weighted avg | 0.9000 | 0.8996 | 0.8997 | 10000 |