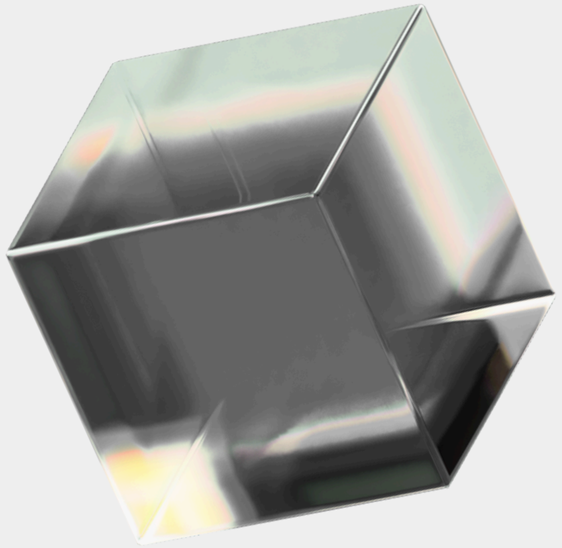
8.HAFTA

YAPAY SİNİR AĞLARI



DR. ÖĞR. ÜYESİ ALPER TALHA KARADENİZ

SAMSUN ÜNİVERSİTESİ

Yapay Sinir Ağlarında Optimizasyon Yöntemleri :

Yapay sinir ağları (YSA), veriden öğrenme sürecini **ağırlıkların güncellenmesi** üzerinden gerçekleştirir. Bu süreçte amaç, **hata (loss)** fonksiyonunu en aza indirmektir. Ancak sinir ağları doğrusal olmayan ve çok karmaşık yapılara sahip olduğu için, bu hatayı en aza indirgemek **optimizasyon yöntemleri** ile yapılır.

Bir yapay sinir ağında:

- **Amaç fonksiyonu (loss function)**: Modelin yaptığı tahmin ile gerçek değer arasındaki farkı ölçer.
- **Optimizasyon algoritmaları**: Bu hata fonksiyonunu minimuma indirmek için ağırlıkları günceller.
- **Zorluk**: Fonksiyonlar çoğu zaman çok boyutlu, doğrusal olmayan, lokal minimum ve saddle point gibi noktalar içerir. Bu nedenle optimizasyon yöntemleri kritik öneme sahiptir.

Optimizasyon Nedir?

Genel anlamda **optimizasyon**, bir problemin en iyi çözümünü bulma sürecidir.

Matematikte optimizasyonun amacı, bir **amaç fonksiyonunu (objective function)** en küçük (minimizasyon) ya da en büyük (maksimizasyon) hale getirmektir.

Örneğin:

- Bir üretim sürecinde maliyeti en aza indirmek,
- Bir şirketin karını en üst seviyeye çıkarmak,
- Bir yolun en kısa mesafesini bulmak,

hepsi birer optimizasyon problemidir.

Yapay sinir ağları bir **öğrenme modelidir**. Öğrenme sırasında ağın amacı, **girdi verilerinden doğru çıktıları üretmektir.**

Ama ağ ilk başta **rastgele ağırlıklarla** başlar ve yanlış tahminler yapar.

İşte burada **optimizasyon devreye girer**:

- Sinir ağının yaptığı hataları ölçmek için **loss function (hata fonksiyonu)** tanımlanır.
- Optimizasyon algoritmaları, bu hata fonksiyonunu **minimuma indirmek** için **ağırlıkları günceller**.

Yani yapay sinir ağlarında optimizasyonun görevi:

"Tahmin ile gerçek arasındaki farkı en aza indiren ağırlıkları bulmak."

Neden Optimizasyon Gereklidir?

Bir sinir ağı milyonlarca parametre (ağırlık ve bias) içerebilir. Bu parametrelerin en iyi değerlerini tek tek deneme-yanılma ile bulmak imkânsızdır.

Ayrıca hata fonksiyonu (loss function) genellikle **çok karmaşık, çok boyutlu ve doğrusal olmayan** bir fonksiyondur.

- **Global minimum** (gerçek en küçük nokta) vardır ama ulaşması zordur.
- Çoğu zaman **lokal minimum** (küçük çukurlar) ve **saddle point** (eğim olmayan düzlükler) bulunur.

Optimizasyon algoritmaları, bu zorluklara rağmen **ağırlıkları doğru yöne doğru iterek** modelin öğrenmesini sağlar.

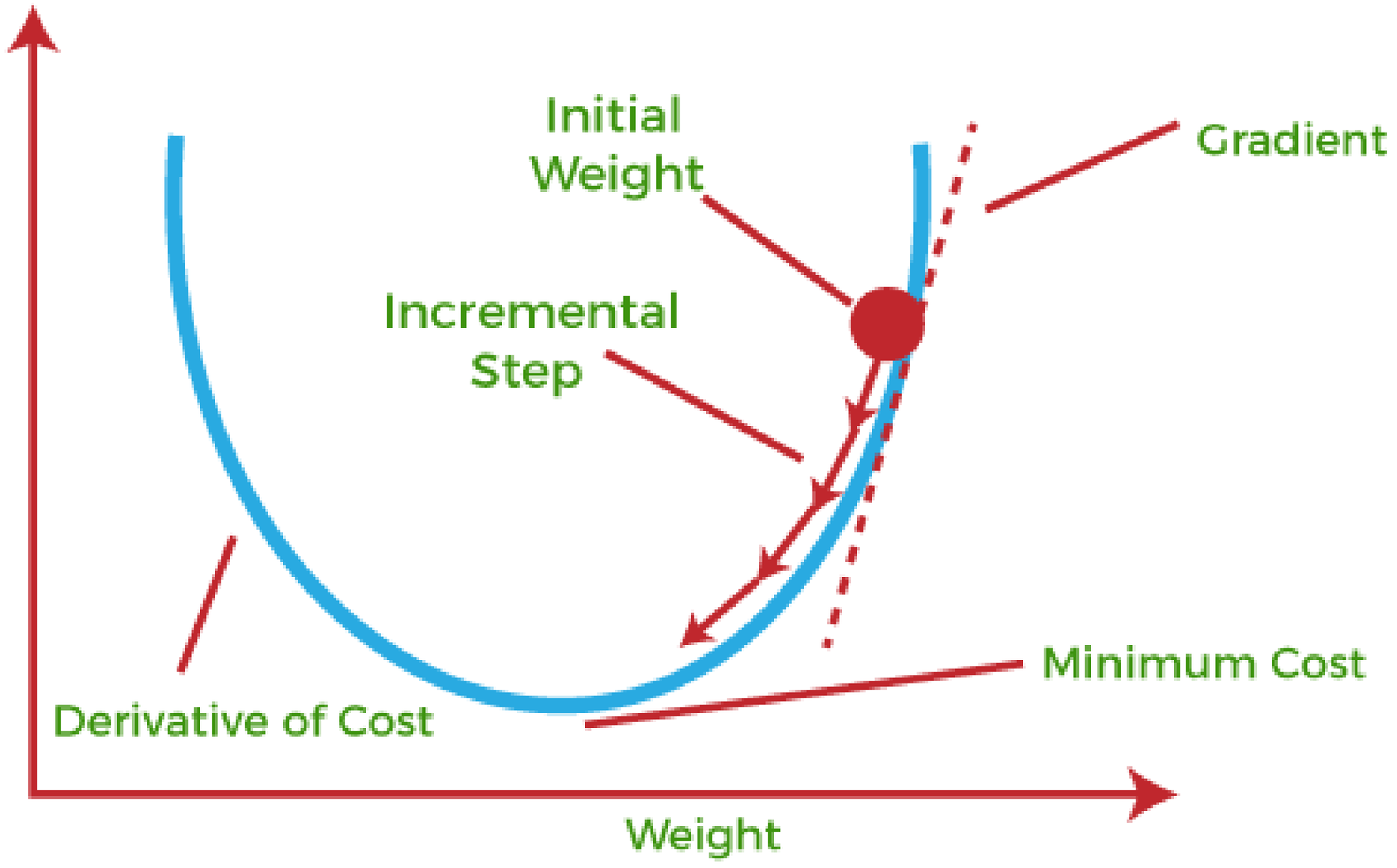
Optimizasyon Yöntemleri :

1. Gradient Descent (Gradyan İnişi)

En temel yöntemdir. Loss fonksiyonunun türevi alınır ve ağırlıklar en dik iniş yönünde güncellenir.

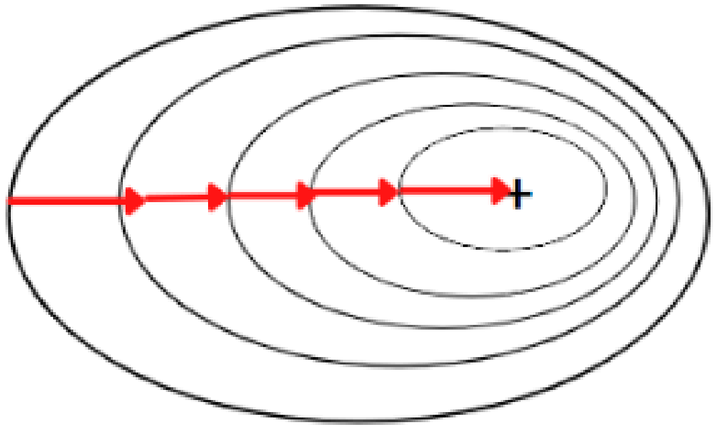
Çeşitleri:

- **Batch Gradient Descent**: Tüm veri setini kullanarak güncelleme yapar.
  - Avantaj: Kararlı öğrenme.
  - Dezavantaj: Çok yavaş (büyük veri setlerinde neredeyse imkânsız).
  - **Örnek**: Küçük bir veri setinde doğrusal regresyon modeli eğitirken kullanılabilir.

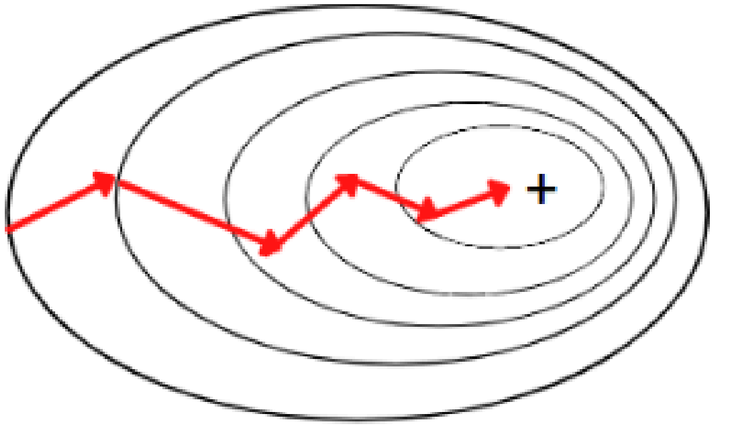

Initial
Weight
Gradient
Incremental
Step
Minimum Cost
Derivative of Cost
Weight

- **Stochastic Gradient Descent (SGD)**: Her bir örnek için güncelleme yapar.
  - Avantaj: Daha hızlıdır.
  - Dezavantaj: Gürültülü ve kararsız olabilir (loss grafiği zigzag yapar).
  - **Örnek**: Bir e-ticaret sitesinde ürün tavsiye sistemini eğitirken, milyonlarca kullanıcı verisi olduğunda tercih edilir.
- **Mini-Batch Gradient Descent**: Veri küçük gruplara bölünerek güncelleme yapılır.
  - Avantaj: Hem hız hem de kararlılık sağlar.
  - **Örnek**: Derin öğrenme modellerinde en sık kullanılan yöntemdir (örneğin CNN ile resim sınıflandırma).

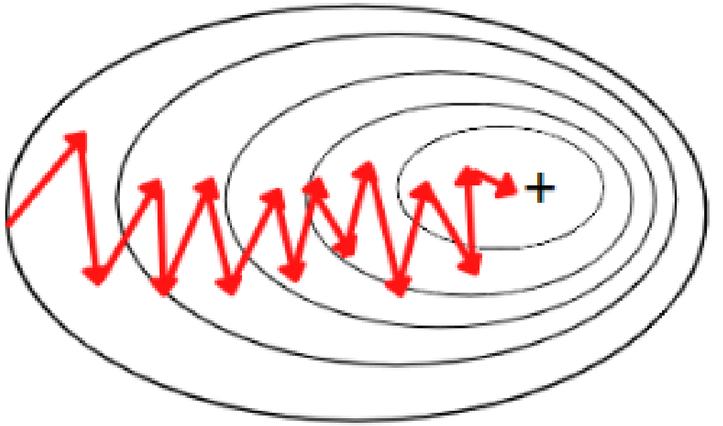
Batch Gradient Descent



Mini-Batch Gradient Descent



Stochastic Gradient Descent

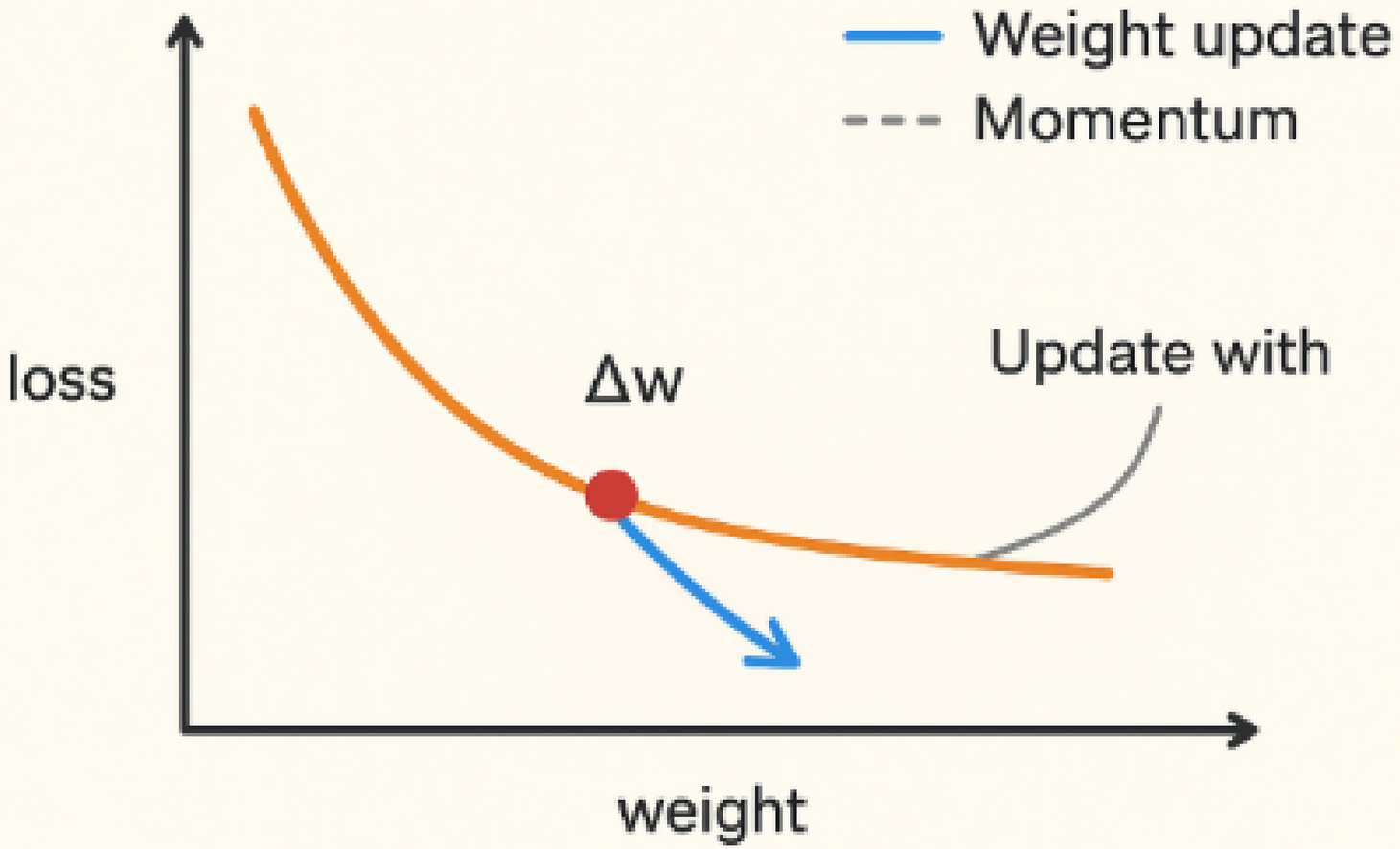


2. Momentum

SGD'deki "sallanma" problemini çözmek için geliştirilmiştir. Önceki güncellemelerden de yararlanarak hızlanır ve daha pürüzsüz bir şekilde minimuma yaklaşır.

- Avantaj: Yerel minimumlara takılmayı zorlaştırır, daha hızlı yakınsama sağlar.
- Dezavantaj: Hiperparametre (momentum katsayısı) doğru seçilmezse verimsiz olabilir.
- **Örnek**:
    - Bir vadide aşağıya yuvarlanan top gibi düşün. Top, küçük çukurlarda takılmaz, hız kazanarak büyük minimuma doğru gider.
    - Görüntü tanıma problemlerinde SGD yerine Momentumlu SGD kullanmak daha hızlı öğrenmeyi sağlar.

MOMENTUM IN OPTIMIZATION



3. Nesterov Accelerated Gradient (NAG)

Momentum'un geliştirilmiş versiyonudur. Normal momentumda güncelleme yaptıktan sonra gradient hesaplanır, NAG'da ise önce "ileride olacağımız nokta" hesaplanır, sonra gradient alınır.

- Avantaj: Daha akıllı adımlar atar, minimuma daha hızlı ve doğru ulaşır.
- Dezavantaj: Hesaplama biraz daha karmaşık.
- **Örnek:**
    - Koşu yapan bir atlet düşün. Atlet, dönemeçten önce hızını ayarlarsa virajı daha kolay alır. NAG da bu mantıkla "ilerideki durumu" hesaba katar.

4. Adagrad

Her parametreye ayrı bir öğrenme oranı atar. Sık güncellenen parametrelerin öğrenme oranı küçülür, nadiren güncellenenlerin öğrenme oranı büyük kalır.

- Avantaj: Özellikle seyrek verilerde çok etkilidir.
- Dezavantaj: Öğrenme oranı zamanla çok küçülüp öğrenme durabilir.
- **Örnek**:
  - **Doğal Dil İşleme (NLP)** görevlerinde, nadiren kullanılan kelimeler için öğrenmeyi hızlandırır.
  - Örneğin "the" gibi çok geçen kelimeler az güncellenir, ama "quantum" gibi nadir kelimeler daha fazla öğrenilir.

5. RMSProp

Adagrad'ın dezavantajını çözmek için geliştirilmiştir. Öğrenme oranı zamanla sıfıra inmez, çünkü ortalama kare gradientlerle güncelleme yapılır.

- Avantaj: Daha kararlı, sürekli öğrenmeye devam eder.
- Dezavantaj: Hiperparametre seçimi önemlidir.
- **Örnek**:
    - RNN (Recurrent Neural Network) ile **zaman serisi tahmini** yaparken tercih edilir.
    - Mesela hava sıcaklığı tahmini veya hisse senedi fiyat tahmini gibi sıralı verilerde çok başarılıdır.

6. Adam (Adaptive Moment Estimation)

En popüler optimizasyon yöntemidir. Momentum + RMSProp yöntemlerinin birleşimidir.

Hem geçmiş gradient bilgisini hem de adaptif öğrenme oranını kullanır.

- Avantaj: Çoğu problemde hızlı ve başarılıdır.
- Dezavantaj: Bazen overfitting (aşırı uyum) yapabilir.
- **Örnek**:
  - **Derin CNN (Convolutional Neural Network)** ile resim sınıflandırmada neredeyse her zaman ilk tercih.
  - Örneğin: MNIST el yazısı tanıma, ImageNet gibi büyük resim veri setlerinde Adam çok hızlı öğrenir.

7. AdamW

Adam'ın geliştirilmiş hali. Ağırlık çürümesi (weight decay) eklenmiştir, böylece modelin parametreleri gereksiz büyümez ve overfitting azalır.

- Avantaj: Adam'a göre genelleme başarısı daha iyidir.
- Dezavantaj: Biraz daha karmaşık.
- **Örnek**:
    - Doğal dil işleme (NLP) modellerinde (örneğin BERT, GPT gibi Transformer tabanlı modellerde) AdamW standart hale gelmiştir.
    - Çünkü bu modeller çok parametre içerir ve overfitting riski yüksektir.

8. Diğer Yöntemler

- **Nadam**: Adam + Nesterov momentum
- **Adadelta**: Adagrad'ın geliştirilmiş versiyonu
- **L-BFGS**: İkinci türev bilgisi kullanan yöntem, küçük veri setlerinde etkilidir

Hangi Yöntem Ne Zaman Kullanılır?

- **SGD + Momentum →**
  - Küçük veya orta boyutlu veri setlerinde, daha basit ağlarda tercih edilir.
  - **Örnek:** Küçük bir yapay sinir ağı ile konut fiyatı tahmini.
- **Adam →**
  - Büyük veri setleri, derin sinir ağları, hızlı öğrenme gerektiren durumlarda.
  - **Örnek:** CNN ile resim sınıflandırma, LSTM ile metin analizi.

- **AdamW →**
  - Çok parametreli derin öğrenme modellerinde overfitting'i önlemek için.
  - **Örnek**: Transformer tabanlı dil modelleri (GPT, BERT).
- **RMSProp →**
  - Zaman serisi, sıralı veriler (özellikle RNN'lerde).
  - **Örnek**: Ses tanıma, hisse senedi fiyat tahmini.

Öğrenme Oranı Zamanlamaları (Learning Rate Scheduling)

Yapay sinir ağlarının eğitiminde **öğrenme oranı (learning rate)**, modelin ağırlıklarının ne kadar büyük adımlarla güncelleneceğini belirler. Eğer öğrenme oranı çok yüksek seçilirse, model minimum noktayı atlayarak öğrenemez; çok düşük seçilirse de eğitim çok yavaş ilerler veya yerel minimumlarda takılır.

Bu sorunu çözmek için, eğitim sürecinde öğrenme oranını sabit tutmak yerine **dinamik olarak değiştirme** yöntemleri geliştirilmiştir. Bu yöntemlere **öğrenme oranı zamanlamaları (learning rate scheduling)** denir.

1. Sabit (Constant) Öğrenme Oranı

- En basit yaklaşımda, eğitim boyunca öğrenme oranı hep aynı kalır.
- Avantajı: Kolay ve hesaplama açısından basittir.
- Dezavantajı: Karmaşık problemlerde modelin öğrenmesini optimize etmek zordur. Eğitim başta iyi gidebilir ama daha sonra minimum noktaya yaklaşırken fazla adım atabilir.

2. Adım Azaltma (Step Decay)

- Belirli epoch'lardan (örneğin her 10 epoch) sonra öğrenme oranı belirli bir katsayı ile çarpılarak düşürülür.
- Örnek: Başlangıçta 0.1 olan öğrenme oranı, her 10 epoch'ta bir yarıya indirilebilir (0.1 → 0.05 → 0.025 → ...).
- Avantajı: Basit ve etkili bir yöntemdir.
- Dezavantajı: Ani düşüşler bazen öğrenmeyi bozabilir.

3. Üstel Azaltma (Exponential Decay)

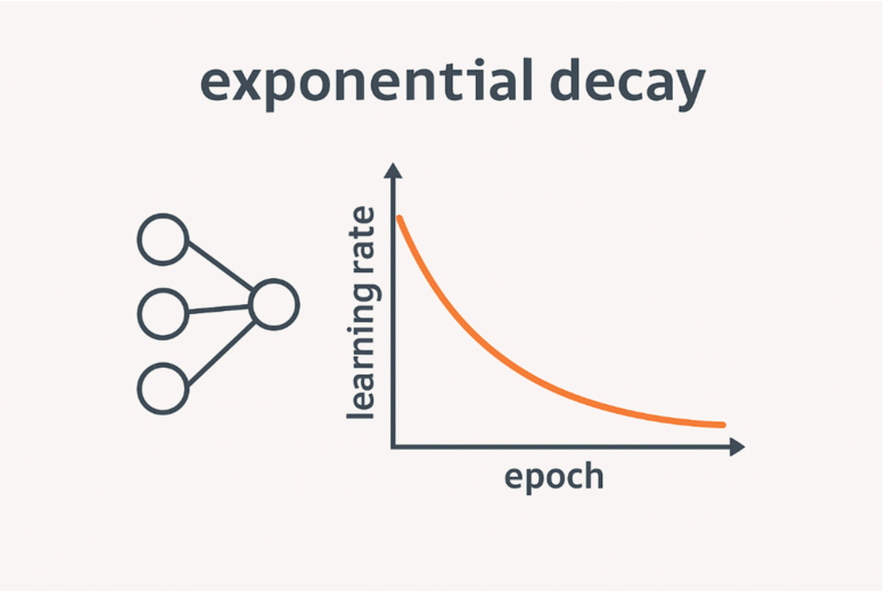
- Öğrenme oranı epoch ilerledikçe üstel bir şekilde küçülür.
- Formül:

$$\eta_t = \eta_0 \cdot e^{-kt}$$

Burada:

- η_t: t'inci adımda öğrenme oranı
- η_0: başlangıç öğrenme oranı
- k: azalma katsayısı

- Avantajı: Yumuşak ve sürekli bir azalma sağlar.



Soru :

Bir değer $V_0 = 100$ ile başlıyor. Her adımda %10 azalıyor

1. 1., 2. ve 5. adımda değer nedir?
2. **10.** adımda değer nedir?
3. Değer ilk kez 50'nin altına **kaçıncı adımda** iner?

Çözüm :

Genel kural:

Her adımda değer $V_n = 100 \times 0.9^n$.

1) 1., 2., 5. adım

- $V_1 = 100 \times 0.9 = \boxed{90}$
- $V_2 = 100 \times 0.9^2 = 100 \times 0.81 = \boxed{81}$
- $V_5 = 100 \times 0.9^5 = 100 \times 0.59049 \approx \boxed{59.05}$

2) 10. adım

- $V_{10} = 100 \times 0.9^{10} = 100 \times 0.348678\ldots \approx \boxed{34.87}$

3) İlk kez 50'nin altına iniş

İstiyoruz: $100 \times 0.9^n < 50 \Rightarrow 0.9^n < 0.5$.

- $0.9^6 \approx 0.5314 \Rightarrow 100 \times 0.5314 \approx 53.14$ (50'nin **üstünde**)
- $0.9^7 \approx 0.4783 \Rightarrow 100 \times 0.4783 \approx 47.83$ (50'nin **altında**)

Bu nedenle değer **ilk kez** $\boxed{7.\ \text{adımda}}$ 50'nin altına iner.

4. Kosinüs Azaltma (Cosine Annealing)

- Öğrenme oranı **kosinüs eğrisi**ne göre yavaşça azaltılır.
- Eğitim boyunca başlangıç değerinden minimum değere doğru dalgalanarak azalır.
- Avantajı: Çok hızlı öğrenmeyi engelleyip, minimum noktayı daha yumuşak bulmayı sağlar.

5. Döngüsel Öğrenme Oranı (Cyclical Learning Rate - CLR)

- Öğrenme oranı sabit olarak azalmaktansa, **belli aralıklarla artırılıp azaltılır.**
- Düşük değerlerden başlayıp, yavaşça artar, sonra tekrar azalır.
- Bu sayede model, farklı bölgeleri keşfedebilir ve daha iyi genelleme sağlar.
- Sıklıkla kullanılan bir versiyon: **One Cycle Policy** (1 döngüde önce yükselir, sonra azalır).

6. Performansa Dayalı Azaltma (Reduce on Plateau)

- Eğer modelin doğrulama kaybı (validation loss) belli bir süre boyunca iyileşmiyorsa, öğrenme oranı otomatik olarak düşürülür.
- Avantajı: Dinamik olduğu için gereksiz yere öğrenme oranını düşürmez, sadece ihtiyaç duyulunca devreye girer.

7. Adaptif Yöntemler (Adam, RMSProp, Adagrad vb.)

- Bu yöntemlerde öğrenme oranı **her parametre için farklı** ve adaptif olarak güncellenir.
- Örneğin **Adam optimizasyon algoritması**, momentum ve geçmiş gradyan bilgilerini kullanarak parametrelerin öğrenme hızını ayarlar.
- Bu yöntemler aslında "optimizer" sınıfına girse de, öğrenme oranı zamanlamasının bir türü olarak düşünülebilir.

Özetle :

- **Başlangıçta** yüksek öğrenme oranı kullanmak, modelin hızlı öğrenmesini sağlar.
- **Eğitimin ilerleyen aşamalarında** öğrenme oranını düşürmek, minimum noktanın daha hassas bulunmasına yardımcı olur.
- Bu nedenle çoğu durumda **öğrenme oranı zamanlamaları, sabit öğrenme oranına göre çok daha iyi sonuçlar** verir.

Uygulama: Keras optimizers parametreleriyle eğitim performans analizi

1. Gerekli Kütüphanelerin Import Edilmesi

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from datetime import datetime

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import Adam, SGD, RMSprop, Adagrad, Adadelta, Ada
max, Nadam
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import time

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

2. Veri Setinin Yüklenmesi ve Ön İşleme

```
# MNIST veri setini yükleme
(X_train, y_train), (X_test, y_test) = mnist.load_data()

# Veriyi normalize etme ve yeniden şekillendirme
X_train = X_train.reshape(-1, 784).astype('float32') / 255.0
X_test = X_test.reshape(-1, 784).astype('float32') / 255.0

# Etiketleri one-hot encoding yapma
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)

# Doğrulama seti oluşturma
X_train, X_val, y_train, y_val = train_test_split(
    X_train, y_train, test_size=0.1, random_state=42
)

print(f"Eğitim verisi: {X_train.shape}")
print(f"Doğrulama verisi: {X_val.shape}")
print(f"Test verisi: {X_test.shape}")
```

3. Model Mimarisi

```
def create_model():
    """Temel sinir ağı modeli oluşturma"""
    model = Sequential([
        Dense(512, activation='relu', input_shape=(784,)),
        Dropout(0.2),
        Dense(256, activation='relu'),
        Dropout(0.2),
        Dense(128, activation='relu'),
        Dropout(0.2),
        Dense(10, activation='softmax')
    ])
    return model

# Modeli kontrol etme
model = create_model()
model.summary()
```

4. Optimizör Konfigürasyonları

```
def get_optimizer_configurations():
    """Farklı optimizör konfigürasyonları"""
    return {
        'Adam_default': Adam(),
        'Adam_lr_0.01': Adam(learning_rate=0.01),
        'Adam_lr_0.001': Adam(learning_rate=0.001),
        'Adam_lr_0.0001': Adam(learning_rate=0.0001),
        'SGD_default': SGD(),
        'SGD_momentum': SGD(momentum=0.9),
        'SGD_nesterov': SGD(momentum=0.9, nesterov=True),
        'RMSprop_default': RMSprop(),
        'RMSprop_lr_0.001': RMSprop(learning_rate=0.001),
        'Adagrad_default': Adagrad(),
        'Adadelta_default': Adadelta(),
        'Adamax_default': Adamax(),
        'Nadam_default': Nadam()
    }

optimizers = get_optimizer_configurations()
```

5. Eğitim Fonksiyonu

```
def train_with_optimizer(optimizer, optimizer_name, epochs=20, batch_size=128):
    """Belirli bir optimizör ile model eğitme"""
    print(f"\n{'='*50}")
    print(f"Eğitim: {optimizer_name}")
    print(f"{'='*50}")

    # Yeni model oluştur
    model = create_model()

    # Modeli derle
    model.compile(
        optimizer=optimizer,
        loss='categorical_crossentropy',
        metrics=['accuracy']
    )

    # Zaman ölçümü
    start_time = time.time()

    # Modeli eğit
    history = model.fit(
        X_train, y_train,
        batch_size=batch_size,
        epochs=epochs,
        validation_data=(X_val, y_val),
        verbose=1
    )
```

```
    # Eğitim süresi
    training_time = time.time() - start_time

    # Test doğruluğu
    test_loss, test_accuracy = model.evaluate(X_test, y_test, verbose=0)

    return {
        'history': history.history,
        'training_time': training_time,
        'test_accuracy': test_accuracy,
        'test_loss': test_loss,
        'optimizer_name': optimizer_name
    }

# Tüm optimizörler için eğitim
results = {}
for opt_name, optimizer in optimizers.items():
    try:
        result = train_with_optimizer(optimizer, opt_name, epochs=15)
        results[opt_name] = result
    except Exception as e:
        print(f"{opt_name} ile eğitim hatası: {e}")
```

6. Sonuçları Analiz Etme ve Görselleştirme

```python
def plot_training_curves(results):
    """Eğitim eğrilerini görselleştirme"""
    fig, axes = plt.subplots(2, 2, figsize=(15, 12))
    axes = axes.flatten()

    metrics = ['accuracy', 'loss', 'val_accuracy', 'val_loss']
    titles = ['Eğitim Doğruluğu', 'Eğitim Kaybı',
              'Doğrulama Doğruluğu', 'Doğrulama Kaybı']

    for i, metric in enumerate(metrics):
        ax = axes[i]
        for opt_name, result in results.items():
            if metric in result['history']:
                ax.plot(result['history'][metric],
                        label=f"{opt_name}", linewidth=2)

        ax.set_title(titles[i], fontsize=14, fontweight='bold')
        ax.set_xlabel('Epok')
        ax.set_ylabel(metric.replace('_', ' ').title())
        ax.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
        ax.grid(True, alpha=0.3)

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

```python
# Eğri görselleştirme
plot_training_curves(results)

def create_comparison_dataframe(results):
    """Karşılaştırma tablosu oluşturma"""
    comparison_data = []

    for opt_name, result in results.items():
        comparison_data.append({
            'Optimizör': opt_name,
            'Test Doğruluğu': result['test_accuracy'],
            'Test Kaybı': result['test_loss'],
            'Eğitim Süresi (s)': result['training_time'],
            'Son Eğitim Doğruluğu': result['history']['accuracy'][-1],
            'Son Doğrulama Doğruluğu': result['history']['val_accuracy'][-1]
        })

    df = pd.DataFrame(comparison_data)
    df = df.sort_values('Test Doğruluğu', ascending=False)
    return df
```

```python
# Karşılaştırma tablosu
comparison_df = create_comparison_dataframe(results)
print("Optimizör Performans Karşılaştırması:")
print(comparison_df.to_string(index=False))

def plot_performance_comparison(comparison_df):
    """Performans karşılaştırması görselleştirme"""
    fig, axes = plt.subplots(2, 2, figsize=(15, 12))

    # Test doğruluğu
    axes[0, 0].barh(comparison_df['Optimizör'], comparison_df['Test Doğruluğu'])
    axes[0, 0].set_title('Test Doğruluğu Karşılaştırması')
    axes[0, 0].set_xlabel('Doğruluk')

    # Eğitim süresi
    axes[0, 1].barh(comparison_df['Optimizör'], comparison_df['Eğitim Süresi
(s)'])
    axes[0, 1].set_title('Eğitim Süresi Karşılaştırması (saniye)')
    axes[0, 1].set_xlabel('Süre (s)')
```

```
    # Son eğitim doğruluğu
    axes[1, 0].barh(comparison_df['Optimizör'], comparison_df['Son Eğitim Doğruluğ
u'])
    axes[1, 0].set_title('Son Eğitim Doğruluğu')
    axes[1, 0].set_xlabel('Doğruluk')

    # Son doğrulama doğruluğu
    axes[1, 1].barh(comparison_df['Optimizör'], comparison_df['Son Doğrulama Doğru
luğu'])
    axes[1, 1].set_title('Son Doğrulama Doğruluğu')
    axes[1, 1].set_xlabel('Doğruluk')

    plt.tight_layout()
    plt.show()

# Performans karşılaştırması
plot_performance_comparison(comparison_df)
```

7. Detaylı Analiz ve Rapor

```
def generate_performance_report(results, comparison_df):
    """Detaylı performans raporu oluşturma"""
    print("="*60)
    print("OPTİMİZÖR PERFORMANS ANALİZ RAPORU")
    print("="*60)

    # En iyi performans gösteren optimizör
    best_optimizer = comparison_df.iloc[0]
    print(f"\nEn İyi Performans: {best_optimizer['Optimizör']}")
    print(f"   Test Doğruluğu: {best_optimizer['Test Doğruluğu']:.4f}")
    print(f"   Eğitim Süresi: {best_optimizer['Eğitim Süresi (s)']:.2f} saniye")

    # İstatistiksel analiz
    print(f"\nİstatistikler:")
    print(f"   Ortalama Test Doğruluğu: {comparison_df['Test Doğruluğu'].mean():.4
f}")
    print(f"   Maksimum Test Doğruluğu: {comparison_df['Test Doğruluğu'].max():.4
f}")
    print(f"   Minimum Test Doğruluğu: {comparison_df['Test Doğruluğu'].min():.4
f}")
    print(f"   Standart Sapma: {comparison_df['Test Doğruluğu'].std():.4f}")

    ...
```

```
    # Öneriler
    print(f"\nÖneriler:")
    print("1. Adam optimizörü genellikle iyi bir başlangıç noktasıdır")
    print("2. Learning rate ayarı çok önemlidir - çok yüksek veya düşük olmamalı")
    print("3. Momentum, SGD için convergence'ı hızlandırabilir")
    print("4. Farklı problemler için farklı optimizörler daha iyi sonuç verebili
r")

# Rapor oluşturma
generate_performance_report(results, comparison_df)

# Sonuçları CSV olarak kaydetme
comparison_df.to_csv('optimizer_performance_comparison.csv', index=False)
print("\nSonuçlar 'optimizer_performance_comparison.csv' olarak kaydedildi")
```

8. Learning Rate Sensitivity Analizi

```python
def learning_rate_sensitivity_analysis():
    """Learning rate duyarlılık analizi"""
    learning_rates = [0.1, 0.01, 0.001, 0.0001, 0.00001]
    lr_results = {}

    for lr in learning_rates:
        optimizer = Adam(learning_rate=lr)
        result = train_with_optimizer(optimizer, f"Adam_lr_{lr}", epochs=10)
        lr_results[f"LR_{lr}"] = result

    # Görselleştirme
    plt.figure(figsize=(12, 6))

    for lr_name, result in lr_results.items():
        plt.plot(result['history']['val_accuracy'],
                 label=lr_name, linewidth=2)

    plt.title('Learning Rate Duyarlılık Analizi', fontsize=14, fontweight='bold')
    plt.xlabel('Epok')
    plt.ylabel('Doğrulama Doğruluğu')
    plt.legend()
    plt.grid(True, alpha=0.3)
    plt.show()

# Learning rate analizini çalıştırma (isteğe bağlı)
# learning_rate_sensitivity_analysis()
```