



Yıldız Teknik Üniversitesi

BLM2642 Ödev 1

Emir Utku Özgen
23011032
utku.ozgen@std.yildiz.edu.tr
<https://www.youtube.com/watch?v=ywfKpzvqfkM>

Ödev Başlığı: Bilgisayar Mühendisliği için Diferansiyel Denklemler Dersi- Ödev 1:Optimizasyon Algoritmalarının Analizi

Giriş: Ödevin amacı anlamsal temsil verileri üzerinden eğitilen Lineer Regresyon modelinde Gradient Descent(GD), Stochastic Gradient Descent(SGD) ve ADAM algoritmalarının performansını ve ağırlık uzayındaki davranışlarını incelemektedir. Modelin eğitimi için gerekli verileri ytucecosmos/Turkish-Gemma-9b-T1 dil modelini kullanarak üretildi. Model eğitimi için 50 tane soru ve bu her 50 soru için bir iyi ve kötü cevap üretirerek toplam da 100 tane eğitim kümesi verisi, 100 tane de test kümesi verisi elde edildi. Her veri soru ve cevap çifti olarak etiketlenmiş ve her iyi cevap için +1 ve her kötü cevap için -1 etiketi kullanılmıştır. Veri üretiminden sonra tüm soru cevap metinlerinin vektörleştirilmesi işleminde ytucecosmos/turkish-e5-large modeli kullanarak d boyutlu anlamsal vektörler üretilmiştir. Modelin nihai girdisi(x), soru temsili ve cevap temsilinin birleştirilmesi ile elde edilmiştir. Son olarak regresyon modeli w verileri kullanarak çıktıyı çıktı = $\tan(w^*x)$ formülüyle tahmin eder, buradaki w ise $2d+1$ boyutlu öğrenilecek veri vektördür. Bu model üzerinde 5 farklı başlangıç w değeri için optimizasyon algoritmaları karşılaştırılmıştır ve w yörüngeleri t-SEN ile görselleştirilmiştir.

A Kısmı Raporu:

1) Verileri Elde Edilmesi

Öncelikle modeli yerelde çalıştmak için LM studio uygulaması kurarak aşağıda belirttiğim promptu modele verdim:

Sen, yapay zeka modelinin cevap kalitesi sınıflandırması için özel olarak tasarlanmış ,soru-cevap tabanlı CSV verileri üreten bir sistem modülüsün.

Çıktı Kuralları:

1. Üretim sırasında iç düşüncelerinden asla çıktıda görünümeyecek; <think> etiketleri kesinlikle kaldırılmalıdır.

2. Yalnızca ve sadece CSV satırları üretilecektir: yorum, analiz, başlık, istatistik veya ek metin üretimi yasaktır.

3. Formatın kesinliği sağlanmalıdır : soru, cevap, etiket.

4. Eğitim kümesi için 50 benzersiz soru ve her soruya karşılık 1 iyi (+1) ile 1 kötü(-1) cevap olmak üzere toplam 100 satır oluşturulacaktır.

5. Test kümesi için, eğitimde kullanılanlardan farklı 50 yeni soru ve bunlara karşılık 1 iyi(+1) ile 1 kötü(-1) cevap olmak üzere toplam 100 satır oluşturulacaktır.

6. Boş satır, liste numarası, kod bloğu veya JSON formatı kullanılmayacaktır.

Ardından tüm verileri Python dilinde Pandas kütüphanesi kullanarak DataFrame haline getirdim. Ardından oluşturduğum DataFrameleri .csv uzantılı şekilde bilgisayara kaydettim. Yaptığım işlemin kodunun fotoğrafını aşağıya bırakıyorum.



```
all_data_list = [
    # --- Eski Veriler ---
    {"Soru": "Python'da f-string nasıl tanımlanır?", "Cevap": "f'metin' ve bir değişken eklenebilir.", "Etiket": 1},
    {"Soru": "Python'da f-string nasıl tanımlanır?", "Cevap": "f' metin ' kullanılmaz", "Etiket": -1},
    {"Soru": "OSI modelinin 4. katmanı nedir?", "Cevap": "TCP protokolünü kapsar.", "Etiket": 1},
    {"Soru": "OSI modelinin 4. katmanı nedir?", "Cevap": "DNS protokollerini yönetir.", "Etiket": -1},
    {"Soru": "HTML etiketleri nasıl kapanır?", "Cevap": "</tag> şeklinde kapanır.", "Etiket": 1},
    {"Soru": "HTML etiketleri nasıl kapanır?", "Cevap": "<tag> ile kapanmaz", "Etiket": -1},
    {"Soru": "SQL'de LEFT JOIN ne işe yarar?", "Cevap": "Sola ait tüm kayıtları döndürür.", "Etiket": 1},
    {"Soru": "SQL'de LEFT JOIN ne işe yarar?", "Cevap": "Sağdaki tabloya filtre uygular", "Etiket": -1},
    {"Soru": "Veri madenciliğinde karar ağacı nasıl oluşturulur?", "Cevap": "Olasılık ve safsızlık ölçütlerine göre dallanır.", "Etiket": 1},
    {"Soru": "Veri madenciliğinde karar ağacı nasıl oluşturulur?", "Cevap": "Rastgele ağac düğümleri seçilerek oluşur", "Etiket": -1},
    {"Soru": "TCP bağlantı nasıl kurulur?", "Cevap": "Üç yönlü el sıkışma ile başlar.", "Etiket": 1},
    {"Soru": "TCP bağlantı nasıl kurulur?", "Cevap": "Tek bir SYN paketi gönderilir.", "Etiket": -1},
    {"Soru": "Kriptografide hash fonksiyonu nedir?", "Cevap": "Tek yönlü matematsel dönüşümür.", "Etiket": 1},
    {"Soru": "Kriptografide hash fonksiyonu nedir?", "Cevap": "Şifreleme için kullanılır", "Etiket": -1},
    {"Soru": "Python'da list comprehension örneği verin?", "Cevap": "[x*x for x in range(5)] şeklinde oluşturulur.", "Etiket": 1},
    {"Soru": "Python'da list comprehension örneği verin?", "Cevap": "[x*x for x in range(5)] şeklinde olusur", "Etiket": -1},
    {"Soru": "HTML5'te form elemanları nasıl etiketlenir?", "Cevap": "type='email'" gibi attribute'lar ile.", "Etiket": 1},
    {"Soru": "HTML5'te form elemanları nasıl etiketlenir?", "Cevap": "class='input' eklenir", "Etiket": -1},
    {"Soru": "Matris çarpımı için koşul nedir?", "Cevap": "İlk matris sütun sayısı = ikinci satır sayısı.", "Etiket": 1},
    {"Soru": "Matris çarpımı için koşul nedir?", "Cevap": "Her iki matrisin boyutu eşit olmalı", "Etiket": -1},
    {"Soru": "Python'da decorator nasıl yazılır?", "Cevap": "def @decorator(func): şeklinde tanımlanır.", "Etiket": 1},
    {"Soru": "Python'da decorator nasıl yazılır?", "Cevap": "fonksiyonun önüne eklenir", "Etiket": -1},
    {"Soru": "SQL'de transaction ne işe yarar?", "Cevap": "Veri bütünlüğü sağlar.", "Etiket": 1},
    {"Soru": "SQL'de transaction ne işe yarar?", "Cevap": "Sorguları hızlandırır.", "Etiket": -1},
    {"Soru": "Matematikte limit tanımı?", "Cevap": "Fonksiyonun yakınsamasıdır.", "Etiket": 1},
    {"Soru": "Matematikte limit tanımı?", "Cevap": "sonsuzdaki değerdir", "Etiket": -1},
    {"Soru": "HTML5'te semantik etiketler nelerdir?", "Cevap": "<header>,<nav> gibi elementler.", "Etiket": 1},
    {"Soru": "HTML5'te semantik etiketler nelerdir?", "Cevap": "<div> ve <span> kullanılmaz", "Etiket": -1},
    {"Soru": "Kimya molar kütle nedir?", "Cevap": "Atom ağırlıklarının toplamıdır.", "Etiket": 1},
    {"Soru": "Kimya molar kütle nedir?", "Cevap": "molekülün kütlesi ile aynıdır", "Etiket": -1},
    {"Soru": "Fizik Newton'un 3. yasası?", "Cevap": "Etki-tepkİYE eşittir.", "Etiket": 1},
    {"Soru": "Fizik Newton'un 3. yasası?", "Cevap": "Kuvvetler zit yönlüdür", "Etiket": -1},
```

Bu kodda da öncelikle Pandas Kütüphanesini import ettim. Ardından elimdeki bütün soru cevap ve etiketleri yukarıda gördüğünüz formatta bir datalist elde ettim. Ardından verileri .csv uzantılı halde kaydettim.



```
import os

# Masaüstü yol (MacOS için)
desktop_path = os.path.expanduser("~/Desktop")
dataset_path = os.path.join(desktop_path, "dataset")

# Eğer klasör yoksa oluştur
if not os.path.exists(dataset_path):
    os.makedirs(dataset_path)

CSV_DOSYA_ADI = 'odev_veri_seti_50_soru.csv'

try:
    df.to_csv(CSV_DOSYA_ADI, index=False, encoding='utf-8')
    print(f"✅ Başarılı! Dosya '{CSV_DOSYA_ADI}' adıyla Colab ortamına kaydedildi.")
    print("Bu dosyayı Mac'inize indirmek için sol menüdeki 'Dosyalar' simgesine tıklayıp dosyayı bulup indirebilirsiniz.")

except Exception as e:
    print(f"❌ KAYIT HATASI: {e}")
```

Train kümesini bitirdikten sonra ise aynı işlemleri test kümesi içinde yaparak .csv uzantılı şekilde kaydettim. Test kümesi için yazdığım kodların bir kısmını aynı şekilde aşağıya bırakıyorum.

```
new_data_list = [
    # PYTHON / ETKİLEŞİMİ
    {"Soru": "Python'da Java sınıfı nasıl oluşturulur?", "Cevap": "public class Ad { ... } şeklinde tanımlanır.", "Etiket": -1},
    {"Soru": "Python'da Java sınıfı nasıl oluşturulur?", "Cevap": "java -c komutu ile kullanılır.", "Etiket": -1},

    # BİYOLOJİ / KİMYA / FİZİK
    {"Soru": "Rönesans hangi ülkede başladı?", "Cevap": "Rönesans İtalya'da başladı", "Etiket": 1},
    {"Soru": "Rönesans hangi ülkede başladı?", "Cevap": "Rönesans Almanya'da başladı.", "Etiket": -1},
    {"Soru": "Fizikte momentum korunumu ne demektir?", "Cevap": "Fnet = 0 iken değişmez.", "Etiket": 1},
    {"Soru": "Fizikte momentum korunumu ne demektir?", "Cevap": "Kütle x hız çarpımıdır.", "Etiket": -1},
    {"Soru": "Kımyada nükleofiller ne işe yarar?", "Cevap": "Elektron verici moleküller ba\u011flar.", "Etiket": 1},
    {"Soru": "Kımyada nükleofiller ne işe yarar?", "Cevap": "Asitlerin proton kaynağıdır.", "Etiket": -1},

    # MATEMATİK / LİNEER CEBİR
    {"Soru": "Lineer cebirde rank nedir?", "Cevap": "Vektör uzayının boyutudur.", "Etiket": 1},
    {"Soru": "Lineer cebirde rank nedir?", "Cevap": "Matrisin satır/sütun sayısıdır.", "Etiket": -1},
    {"Soru": "Matematiğe Riemann hipotezi nedir?", "Cevap": "asal sayıların da\u011flımını \u0131\u011fklar.", "Etiket": 1},
    {"Soru": "Matematiğe Riemann hipotezi nedir?", "Cevap": "2'den büyük tüm asallar için gecerlidir.", "Etiket": -1},

    # PROGRAMLAMA MANTIĞI / SQL
    {"Soru": "Python'da decorator neden kullanılır?", "Cevap": "Fonksiyon davranışını de\u011flistirir.", "Etiket": 1},
    {"Soru": "Python'da decorator neden kullanılır?", "Cevap": "Parametre eklemek içindir.", "Etiket": -1},
    {"Soru": "SQL'de aggregate fonksiyon örnekleri?", "Cevap": "COUNT(), SUM() gibi fonksiyonlardır.", "Etiket": 1},
    {"Soru": "SQL'de aggregate fonksiyon örnekleri?", "Cevap": "WHERE koşulu ile filtreler.", "Etiket": -1},]

# DataFrame'i olu\u011flurma
df_new_set = pd.DataFrame(new_data_list)

# CSV dosyası olarak kaydetme (Güvenli, yerel \u0131\u011fzma dizinine)
csv_file_path = 'test_veri_seti.csv'
df_new_set.to_csv(csv_file_path, index=False, encoding='utf-8')

print(f"\u2713 Yeni veri setiniz (Toplam {df_new_set.shape[0]} satır) \u0131\u011fzıyla '{csv_file_path}' dosyasına kaydedildi.")
print("\nOlu\u011flurulan DataFrame'in ilk 5 Satırı:")
print(df_new_set.head())

# Yeni veri setiniz (Toplam 100 satır) \u0131\u011fzıyla 'test_veri_seti.csv' dosyasına kaydedildi.
```

2) Elde Edilen Verilerin Anlamsal Temsili

Verileri .csv uzantılı dosya şeklinde elde edildikten sonra verilerin anlamsal temsili için ytucecosmos/turkish-e5-large modelini kullanarak hem test hem de train kümeleri için soru cevap ve etiketlerin vektörlerini .npy uzantılı dosya şeklinde elde ettim. Aşağıya yukarıda bahsettiğim işlemin kodunu bırakıyorum.

```
# Veri setlerini yükleyin
try:
    train_df = pd.read_csv("train.csv")
    test_df = pd.read_csv("test.csv")
    print("✅ Veri setleri başarıyla yüklendi.")
except FileNotFoundError as e:
    print("❌ Hata: Dosya bulunamadı: {e}")
    exit()

# Model Adı ve Cihaz Ayarı
MODEL_NAME = "ytu-ce-cosmos/turkish-e5-large"
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"\n💡 Kullanılacak Cihaz: {device}")

# =====#
# 1. MODEL VE TOKENIZER YÜKLEME
# =====#

try:
    print(f"\n💡 {MODEL_NAME} modeli yükleniyor...")
    tokenizer = AutoTokenizer.from_pretrained(MODEL_NAME)
    model = AutoModel.from_pretrained(MODEL_NAME)
    model.to(device).eval() # Modeli değerlendirme (eval) moduna al
    print("✅ Model yüklemeye tamamlandı.")
except Exception as e:
    print("❌ Model yüklenirken bir hata oluştu: {e}")
    exit()

def embed_texts(texts: list) -> np.ndarray:
    """
    E5 modelini kullanarak metin listesinden anlamsal vektörler üretir.
    """
    embeddings_list = []
    # Veriyi daha küçük parçalara (batch) ayırarak işlem hızını artırırız
    BATCH_SIZE = 64
```

```
def embed_texts(texts: list) -> np.ndarray:
    """
    E5 modelini kullanarak metin listesinden anlamsal vektörler üretir.
    """
    embeddings_list = []
    # Veriyi daha küçük parçalara (batch) ayırarak işlem hızını artırırız
    BATCH_SIZE = 64

    for i in tqdm(range(0, len(texts), BATCH_SIZE), desc="Vektör Üretimi"):
        batch_texts = texts[i:i + BATCH_SIZE]

        # Tokenizasyon
        tokenized_inputs = tokenizer(
            batch_texts,
            padding=True,
            truncation=True,
            return_tensors="pt",
            max_length=512
        ).to(device)

        with torch.no_grad():
            model_output = model(**tokenized_inputs)

        # [CLS] token'ının çıktısını al
        embeddings = model_output.last_hidden_state[:, 0]

        # L2 normalizasyon
        embeddings = torch.nn.functional.normalize(embeddings, p=2, dim=1)

        embeddings_list.append(embeddings.cpu().numpy())

    return np.concatenate(embeddings_list, axis=0)

# =====#
```

```

# =====#
# 2. VECTÖR ÜRETİMİ
# =====#
start_time = time.time()

# 2.1. Eğitim Verisi Vektörleri
print("\n\x26gt; Eğitim (Train) verisi için vektörler üretiliyor...")
train_texts = train_df['Soru'].tolist() + train_df['Cevap'].tolist()
train_embeddings_all = embed_texts(train_texts)

N_train = len(train_df)
train_query_embeddings = train_embeddings_all[:N_train]
train_passage_embeddings = train_embeddings_all[N_train:]

print(f"\x26gt; Train Vektör Boyutları: Soru {train_query_embeddings.shape}, Cevap {train_passage_embeddings.shape}")

# 2.2. Test Verisi Vektörleri
print("\n\x26gt; Test verisi için vektörler üretiliyor...")
test_texts = test_df['Soru'].tolist() + test_df['Cevap'].tolist()
test_embeddings_all = embed_texts(test_texts)

N_test = len(test_df)
test_query_embeddings = test_embeddings_all[:N_test]
test_passage_embeddings = test_embeddings_all[N_test:]

print(f"\x26gt; Test Vektör Boyutları: Soru {test_query_embeddings.shape}, Cevap {test_passage_embeddings.shape}")

end_time = time.time()
print(f"\n\x26gt; Toplam Vektör Üretim Süresi: {end_time - start_time:.2f} saniye.")

# =====#
# 3. VECTÖRLERİ VE ETİKETLERİ KAYDETME (EKLENEN KISIM)
# =====#

```

```

try:
    y_train_raw = train_df[ETIKET_SUTUN_ADI].values.astype(np.float32)
    train_labels = np.where(y_train_raw == 0, -1.0, 1.0)
    train_labels = train_labels.reshape(-1, 1) # (N, 1) boyutuna getir
    print(f"\x26gt; Train Label Boyutu (+1/-1): {train_labels.shape}")
except KeyError:
    print(f"\x26gt; Hata: Train dosyasında '{ETIKET_SUTUN_ADI}' sütunu bulunamadı.")
    exit()

# Test Etiketleri: 0 -> -1.0 ve 1 -> 1.0 dönüşümü
try:
    y_test_raw = test_df[ETIKET_SUTUN_ADI].values.astype(np.float32)
    test_labels = np.where(y_test_raw == 0, -1.0, 1.0)
    test_labels = test_labels.reshape(-1, 1) # (N, 1) boyutuna getir
    print(f"\x26gt; Test Label Boyutu (+1/-1): {test_labels.shape}")
except KeyError:
    print(f"\x26gt; Hata: Test dosyasında '{ETIKET_SUTUN_ADI}' sütunu bulunamadı.")
    exit()

# 3.2. NumPy dosyası olarak kaydetme
#
# Vektörleri kaydetme
np.save("train_query_embeddings.npy", train_query_embeddings)
np.save("train_passage_embeddings.npy", train_passage_embeddings)
np.save("test_query_embeddings.npy", test_query_embeddings)
np.save("test_passage_embeddings.npy", test_passage_embeddings)

# Etiketleri kaydetme (YENİ EKLENEN)
np.save("train_labels.npy", train_labels)
np.save("test_labels.npy", test_labels)

print("\x26gt; Tüm anlamsal vektörler ve etiketler başarıyla kaydedildi.")

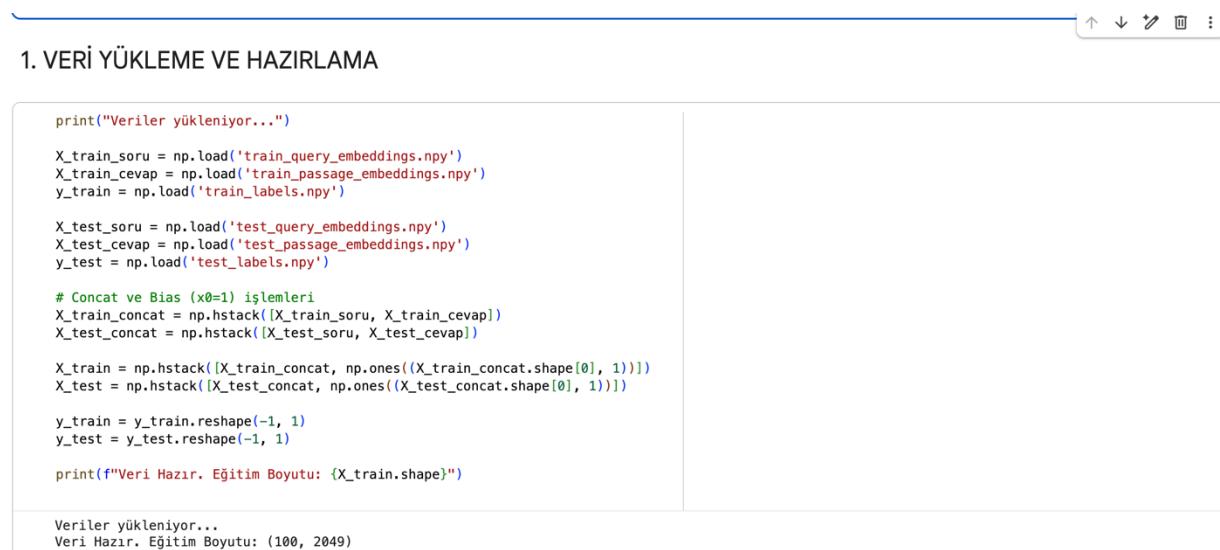
```

3)Optimizasyon Algoritmalarının Karşılaştırılması

Bu ödev tasarlanan ikili sınıflandırma modelinin eğitiminde kullanılan Gradient Descent(GD), Stochastic Gradient Descent(SGD) ve ADAM algoritmaların performansını ve başlangıç koşullarına olan duyarlılığını karşılaştırmayı amaçlamaktadır. Deneyler her algoritmanın 5 farklı W değeri üzerinden başlattılarak uygulanmıştır. Bu yaklaşım, algoritmaların kayıp yüzeyindeki farklı başlangıç noktalarından minimuma ne kadar etkin ve tutarlı bir şekilde

yakınsadığını gözlemlememizi sağlamıştır. Karşılaştırma hem eğitim süresi hem de güncelleme sayısı olmak üzere iki farklı yatay eksen kriterleri üzerinden yapılmıştır. Dikey eksende ise algoritmaların yakınsama kalitesini ölçmek amacıyla temel parametreler olan Eğitim Kaybı/Başarısı ve Test Kaybı/Başarısı kullanılmıştır. Elde edilen sonuçlar Süre vs.Kayıp/Başarı ve Epoch vs. Kayıp/Başarı olmak üzere dört farklı grafik serisi halinde gösterilmiştir.

Şimdi tek tek bu kısım için yazdığım kodları aşağıya bırakıyorum.



```
print("Veriler yükleniyor...")

X_train_soru = np.load('train_query_embeddings.npy')
X_train_cevap = np.load('train_passage_embeddings.npy')
y_train = np.load('train_labels.npy')

X_test_soru = np.load('test_query_embeddings.npy')
X_test_cevap = np.load('test_passage_embeddings.npy')
y_test = np.load('test_labels.npy')

# Concat ve Bias (x0=1) işlemleri
X_train_concat = np.hstack([X_train_soru, X_train_cevap])
X_test_concat = np.hstack([X_test_soru, X_test_cevap])

X_train = np.hstack([X_train_concat, np.ones((X_train_concat.shape[0], 1))])
X_test = np.hstack([X_test_concat, np.ones((X_test_concat.shape[0], 1))])

y_train = y_train.reshape(-1, 1)
y_test = y_test.reshape(-1, 1)

print(f"Veri Hazır. Eğitim Boyutu: {X_train.shape}")

Veriler yükleniyor...
Veri Hazır. Eğitim Boyutu: (100, 2049)
```

İlk olarak modelin optimizasyon karşılaştırmasına geçmeden önceki son adım olan, girdi temsillerini ve etiketleri yüklemek ve modelin beklediği formata getirmektir. Bu adım için öncelikle Soru temsilleri,Cevap temsilleri ve etiketler hem eğitim hem de test kümeleri için ayrı ayrı yüklenmiştir. Ardından modelin girdi vektörü oluşturmak amacıyla Soru ve Cevap temsilleri. Yatay olarak birleştirilmiştir ve böylece d boyutundan 2d boyutuna ulaşılmıştır. Bu birleştirilmiş vektörlere, modelin $w \cdot x + b$ işlemini tek bir matris çarpımıyla gerçekleştirebilmesi için Bias terimi eklenmiştir. Bu, vektörün sonuna sabit bir 1 değerinden oluşan bir sütunun eklenmesiyle yapılmış ve nihai girdi matrisi X 'in boyutu ($\text{Örnek sayısı}) \times (2d+1)$ olarak belirlenmiştir. Son olarak, etiket vektörlerleri y 'ler, olası boyutlandırma hatalarını önlemek adına matematiksel işlemler için uygun olan sütun vektörü $(N, 1)$ formatına dönüştürüülerek modelin eğitime hazır hale gelmesi sağlanmıştır.

2. MODEL VE YARDIMCI FONKSİYONLAR

```
❶ def tanh(x): return np.tanh(x)
❷ def predict(X, w): return tanh(np.dot(X, w))
❸ def calculate_loss(y_true, y_pred): return np.mean((y_true - y_pred) ** 2)
❹ def calculate_accuracy(y_true, y_pred): return np.mean(np.where(y_pred >= 0, 1, -1) == y_true)

❺ def get_gradients(X_batch, y_batch, w):
    z = np.dot(X_batch, w)
    p = tanh(z)
    error = y_batch - p
    derivative = 1 - p**2
    # GD için bu ortalama (mean) işlemi gradyanı küçültür, bu yüzden GD'de yüksek LR gereklidir.
    grad = -2 * np.dot(X_batch.T, error * derivative) / X_batch.shape[0]
    return grad
```

Bu kod paketi, basit bir ikili sınıflandırma modelinin temelini oluşturur: tanh aktivasyon fonksiyonu, modelin çıktısını $[-1, 1]$ arasına sıkıştırır; predict fonksiyonu, girdiyi (X) ağırlıklarla (w) çarpıp tanh'tan geçirerek tahmini yapar; calculate_loss fonksiyonu, tahmin ve gerçek değerler arasındaki farkın karesinin ortalaması olan MSE(Mean Squared Error) hesaplayarak modelin hmasını ölçer; calculate_accuracy ise tahminleri +1 veya -1 olarak sınıflandırıp doğru eşleşme oranını bulur. En kritik fonksiyon olan get_gradients MSE kaybının ağırlıklara göre türevi olan gradyanı hesaplamak için ters yayılımın kuralını kullanır ve bu sonucu batch büyülüğüne bölgerek ortalamasını alır. Bu bölme işlemi gradyanın büyülüğünü kasıtlı olarak küçültüğü için, ağırlıkların öğrenme sırasında yeterince hareket etmesi için Gradient Descent algoritmasında daha yüksek bir öğrenme oranı kullanılması gereklidir.

3. EĞİTİM DÖNGÜSÜ

```
▶ def train_optimizer(X_train, y_train, X_test, y_test, w_init, optimizer, epochs, lr):
    w = w_init.copy()
    if w.ndim == 1: w = w.reshape(-1, 1)

    # w yörungesini kaydetmek için yeni liste
    trajectory = [] # <-- Yörunge Listesi Eklendi!

    history = {'time': [], 'epoch': [], 'train_loss': [], 'test_acc': [], 'w_trajectory': trajectory} # <-- Söz
    start_time = time.time()

    # Adam parametreleri
    m = np.zeros_like(w); v = np.zeros_like(w)
    beta1, beta2, epsilon = 0.9, 0.999, 1e-8
    t_adam = 0
    N = X_train.shape[0]

    def record_metrics(current_epoch):
        # ... (Metrik Kayıt kodları aynı) ...
        curr_time = time.time() - start_time
        tr_loss = calculate_loss(y_train, predict(X_train, w))
        te_acc = calculate_accuracy(y_test, predict(X_test, w))

        history['epoch'].append(current_epoch)
        history['time'].append(curr_time)
        history['train_loss'].append(tr_loss)
        history['test_acc'].append(te_acc)

    def record_w(): # <-- Yörunge Kaydetme Fonksiyonu
        trajectory.append(w.flatten().copy())

    record_w()
```

Bu kod, ikili sınıflandırma modelinin ağırlıklarını (w) Gradient Descent (GD), Stochastic Gradient Descent (SGD), ve Adam olmak üzere üç farklı optimizasyon algoritması kullanarak eğiten temel bir eğitim döngüsünü tanımlar. Fonksiyon, başlangıç ağırlıkları (w) eğitim ve test verilerini (X, Y) ve öğrenme oranını (lr) alarak, belirtilen epoch sayısı boyunca modeli döngüye sokar. Her epoch başında ve sonunda modelin performansını ölçmek için eğitim kaybı (Loss) ve test doğruluğu (Accuracy) verileri hesaplanıp history sözlüğüne kaydedilir. GD modunda, tüm eğitim kümesi kullanılarak hesaplanan gradyanla ağırlıklar tek adımda güncellenirken; SGD ve ADAM modlarında, veri kümesi üzerinde rastgele karıştırılmış tek bir örnek (mini-batch boyutu 1) kullanılarak gradyan hesaplanır ve ağırlıklar daha sık güncellenir. Özellikle ADAM algoritması, gradyanların birinci ve ikinci dereceden tahminlerini (momentum m ve kare moment v) takip eden uyarlanabilir bir yöntem uygulayarak, öğrenme oranını dinamik olarak ayarlayıp daha verimli bir yakınsama hedefler. Tüm optimizasyon yöntemlerinde, ağırlıkların eğitim sırasındaki değerlerinin ($w_trajectory$) kaydedilmesi, farklı algoritmaların kayıp yüzeyi üzerindeki yürüme yollarını görsel olarak analiz etme imkanı sunan kritik bir özelliklektir.

Öncelikle yardımcı fonksiyonları açıklıyorum:

```
def record_metrics(current_epoch):
    # ... (Metrik kayıt kodları aynı) ...
    curr_time = time.time() - start_time
    tr_loss = calculate_loss(y_train, predict(X_train, w))
    te_acc = calculate_accuracy(y_test, predict(X_test, w))

    history['epoch'].append(current_epoch)
    history['time'].append(curr_time)
    history['train_loss'].append(tr_loss)
    history['test_acc'].append(te_acc)
```

Bu record_metrics adlı iç içe fonksiyon, modelin eğitim döngüsündeki belirli bir andaki performansını ölçmek ve kaydetmek için kritik bir yardımcı araçtır. Fonksiyon, öncelikle eğitimin başlangıcından bu yana geçen toplam süreyi(curr_time) hesaplar ve mevcut epoch numarasını ile birlikte history sözlüğüne kaydeder. Ardından, modelin anlık ağırlıkları (w) kullanılarak, harici olarak tanımlanmış predict fonksiyonu ile hem eğitim hem de test verileri üzerinde tahminler yapılır. Bu tahminler ve gerçek etiketler kullanılarak eğitim kaybı(tr_loss) calculate_loss ile ve test doğruluğu (te_acc) calculate_accuracy ile hesaplanır. Son olarak, hesaplanan bu iki temel parametre (tr_loss ile te_acc), eğitim ilerlemesini takip etmek amacıyla history sözlüğündeki ilgili listelere eklenir; böylece, eğitimin sonunda modelin öğrenme hızı ve genelleme yeteneği hakkında ayrıntılı bir zaman serisi analizi yapılabilir.

```
▶ def train_optimizer(X_train, y_train, X_test, y_test, w_init, optimizer, epochs, lr):
    w = w_init.copy()
    if w.ndim == 1: w = w.reshape(-1, 1)

    # w yörungesini kaydetmek için yeni liste
    trajectory = [] # <-- Yörunge Listesi Eklendi!

    history = {'time': [], 'epoch': [], 'train_loss': [], 'test_acc': [], 'w_trajectory': trajectory} # <-- Söz
    start_time = time.time()

    # Adam parametreleri
    m = np.zeros_like(w); v = np.zeros_like(w)
    beta1, beta2, epsilon = 0.9, 0.999, 1e-8
    t_adam = 0
    N = X_train.shape[0]
```

Bu kod blogu, train_optimizer fonksiyonun eğitim döngüsünü başlatmadan önceki temel hazırlıklarını içerir: Öncelikle, orijinal başlangıç ağırlıklarının (w_init) bir kopyası alınır ve matris işlemlerine uygun olması için gerekiyorsa iki boyutlu bir yapıya dönüştürülür. Ardından, modelin ilerlemesini izlemek üzere bir metrik takip sistemi kurulur; bu sistem, ağırlık yörungesini (trajectory) kaydetmek için boş bir liste ve zaman, kayıp, doğruluk gibi tüm sonuçları tutacak olan sözlüğünü içerir ve süreç başlangıç zamanı kaydedilir. Son olarak, seçilmesi durumunda kullanılacak olan Adam optimizasyon algoritmasına ait parametreler (beta1,beta2,epsilon) tanımlanır ve gradyan hareketli ortalamalarını tutacak m ve v vektörleri ile adım sayacı (T_adam)sıfır olarak başlatılır, böylece eğitim döngüsü için tüm altyapı hazır hale getirilmiş olur.

```

for epoch in range(epochs):
    record_metrics(epoch)

    if optimizer == 'GD':
        grad = get_gradients(X_train, y_train, w)
        w -= lr * grad
        record_w() # <-- w kaydedildi

    elif optimizer == 'SGD' or optimizer == 'Adam':
        indices = np.arange(N)
        np.random.shuffle(indices)
        for i in indices:
            xi = X_train[i:i+1]; yi = y_train[i:i+1]
            grad = get_gradients(xi, yi, w)

        if optimizer == 'SGD':
            w -= lr * grad
            record_w() # <-- w kaydedildi

        elif optimizer == 'Adam':
            t_adam += 1
            m = beta1 * m + (1 - beta1) * grad
            v = beta2 * v + (1 - beta2) * (grad ** 2)
            m_hat = m / (1 - beta1 ** t_adam)
            v_hat = v / (1 - beta2 ** t_adam)
            w -= lr * m_hat / (np.sqrt(v_hat) + epsilon)
            record_w() # <-- w kaydedildi

    # Son epoch sonrası son w'yu ve metrikleri kaydet
    record_metrics(epochs)

    # w_trajectory'yi history sözlüğünden ayırip ekliyoruz
    history['w_trajectory'] = trajectory
return history

```

Bu döngü train_optimizer fonksiyonunun en önemli yeridir. Belirtilen epoch sayısı boyunca modelin ağırlıklarını (w) güncelleyerek öğrenmeyi gerçekleştirir. Her epoch başında record_metrics ile performans kaydedilir ve optimizasyon algoritmasına (GD,SGD,Adam) göre bir güncelleme yolu izlenir. GD seçildiğinde, tüm eğitim veri setinin gradyanı kullanılarak ağırlıklar tek bir adımda güncellenir; SGD veya ADAM seçildiğinde ise, veri seti rastgele karıştırılır ve döngü her bir örnek üzerinden ilerler. SGD her örnek sonrası standart gradyan kuralını uygularken;ADAM algoritması, gradyanların hareketli ortalamalarını (m, v) takip eder, adım sayacı (t_{adam}) ile yanılık düzeltmesi uygular ve ağırlıkları uyarlanabilir bir öğrenme hızı ile günceller. Her güncelleme sonrası record_w ağırlıkların kayıp yüzeyindeki yörünge titizlikle kaydedilir. Döngü tamamlandığında record_metrics ile son metrikler eklenir ve yörünge verisi dahil olmak üzere tüm eğitim geçmişini içeren history sözlüğü döndürülür.

Ayrıca tek tek GD,SGD ve ADAM algoritmalarının matematiğinden bahsedeyim:

1.Gradient Descent(GD): GD, tüm eğitim veri setini kullanarak kayıp fonksiyonunun eğimini (gradyanını) hesaplar ve ağırlıkları bu eğimin zıt yönünde günceller. Amacı, kayıp yüzeyindeki en dik iniş yönünü takip etmektir.

Matematiksel formülü: $x_{new} = x_{old} - lr * df$

$X_{new} = \text{yeni } x$

$X_{old} = \text{eski } x$

Lr=learning rate

Df=türev

2. Stochastic Gradient Descent(SGD): SGD, GD algoritmasının her ağırlık güncellemesi için tüm veri setini kullanma gerekliliğini ortadan kaldırarak daha hızlı hareket eden bir türevidir. Matematiksel olarak, GD'den farklı olarak, SGD her güncelleme adımında kayıp fonksiyonunun gradyanını tüm veri setine göre değil, yalnızca rastgele seçilmiş tek bir veri örneğine göre hesaplar. Bu yaklaşım, gradyan tahmininin daha gürültülü olmasına yol açar; yani, her adım minimuma giden en doğru yönü göstermese de, çok sık güncelleme yapıldığı için SGD, GD'den çok daha hızlı bir ilerleme kaydeder ve büyük veri setlerinde tercih edilir. Matematiksel formülü: $x_{\text{new}} = x_{\text{old}} - lr * df$

Formül çok benzer ama SGD de türev sadece bir veri örneğine göre hesaplanıyor.

3. ADAM: Adam algoritması, SGD üzerine inşa edilmiş, uyarlanabilir bir öğrenme oranı mekanizmasıdır. SGD'nin aksine, Adam sadece gradyanın yönünü değil, aynı zamanda her bir parametrenin geçmişteki gradyan hareketini takip eder. Bunu yapmak için iki temel kavramı kullanır: Birinci Moment (m) ve İkinci Moment(v). Birinci moment, gradyanların üstel hareketli ortalamasını (tutarken; ikinci moment, gradyanların karelerinin hareketli ortalamasını tutar. Bu momentler, algoritmanın başlangıcında sıfıra yakın olduklarından oluşan yanılılığı düzelttilir. Son olarak, Adam, her bir parametre için öğrenme oranını (v_t) kareköküne bölerek dinamik olarak ayarlar ve böylece daha büyük adımlar atılması gereken yerlerde daha hızlı, kayıp yüzeyinin düz olduğu yerlerde ise daha yavaş ve istikrarlı bir yakınsama sağlar. Bu özelliği, Adam'ı çoğu makine öğrenimi görevi için varsayılan ve oldukça verimli bir optimizasyon algoritması yapar.

Matematiksel formülü : $m_t = B_1 * m_{(t-1)} + (1-B_1) * g_t$
 $v_t = B_2 * v_{(t-1)} + (1-B_2) * g_t * g_t$

$$\hat{m}_t = m_t / (1 - \beta_1^t)$$

$$\hat{v}_t = v_t / (1 - \beta_2^t)$$

$$\theta_t = \theta_{t-1} - \alpha \hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon)$$

4. DENEYLERİ ÇALIŞTIRMA

```
▶ epochs = 300
num_experiments = 5
input_dim = X_train.shape[1]

learning_rates = {
    'GD': 0.1, # GD için daha küçük LR önerilir
    'SGD': 0.1,
    'Adam': 0.01
}

results = {'GD': [], 'SGD': [], 'Adam': []}
trajectories = {'GD': [], 'SGD': [], 'Adam': []}
optimizers = ['GD', 'SGD', 'Adam']

print(f"Deneyler başlıyor ({epochs} epoch)...")

for i in range(num_experiments):
    print(f"Deney {i+1}/{num_experiments}...")
    # 5 farklı ilk w değeri, tüm optimizasyonlar için aynı başlangıç noktasını kullanır
    w_init = np.random.randn(input_dim, 1) * 0.5

    for opt in optimizers:
        hist = train_optimizer(X_train, y_train, X_test, y_test, w_init, opt, epochs, lr=learning_rates[opt])

        # Yörüngeyi ayıร ve metrikleri kaydet
        w_traj = hist.pop('w_trajectory')
        results[opt].append(hist)
        trajectories[opt].append(w_traj)

print("✅ Deneyler tamamlandı. Ortalama sonuçlar hesaplanıyor...")
```

Bu kısım ise ödevin en önemli kısmı çünkü bütün algoritmaları çalıştırduğumuz kısım. Bu kod bloğu, farklı optimizasyon algoritmalarının (GD, SGD, ADAM) performansını karşılaştırmak için tasarlanmış, çoklu çalıştırmadan oluşan bir deney çerçevesini kurar ve yürütür. Öncelikle, eğitim için gerekli sabitler (epochs=300, num_experiments=5, input_dim) ve her bir algoritma için belirlenmiş öğrenme oranları tanımlanır. Deney sonuçları ve ağırlıkların hareket yolları (trajectories) ise optimizasyon algoritmalarına göre gruplandırılmak üzere sözlükler halinde başlatılır. Ana döngü num_experiments kadar tekrarlanır ve her yeni deneyde tüm algoritmalar için başlangıç noktası olarak yeni ve rastgele bir ağırlık vektörü belirlenir. Bu başlangıç noktasından hareketle, iç döngü sırayla her bir optimizasyon algoritmasını (GD, SGD, ADAM) çağırır; her çağrıda, train_optimizer fonksiyonu ilgili öğrenme hızıyla çalıştırılır ve döndürülen metrikler (hist) ile ağırlık yörüneleri (w_traj) ilgili sözlüklerine eklenir. Bu çoklu çalışma yapısı, algoritmaların performansını tek bir başlangıç noktasına bağımlı kalmadan, daha güvenilir ve istatistiksel olarak anlamlı bir şekilde karşılaştırmayı amaçlar.

```
# Ortalama sonuçlar (Ödev A Grafikleri için)
avg_results = {}
for opt in results:
    avg_results[opt] = {
        'num_epochs': np.array(results[opt][0]['epoch']),
        'avg_train_loss': np.mean([r['train_loss'] for r in results[opt]], axis=0),
        'avg_test_accuracy': np.mean([r['test_acc'] for r in results[opt]], axis=0),
        'elapsed_time_sec': np.mean([r['time'] for r in results[opt]], axis=0)
    }
```

Bu kod bloğu, daha önce 5 farklı başlangıç noktasıyla çalıştırılan tüm optimizasyon deneylerinin sonuçlarını toplayarak, rastgelelikten kaynaklanan sapmaları gidermek ve güvenilir ortalama performans eğrileri oluşturmak amacıyla tasarlanmıştır. avg_results sözlüğü, döngü aracılığıyla her bir algoritma (GD, SGD, ADAM) için ayrı ayrı yapılandırılır. Her algoritmanın tüm deneyleri (results[opt]) gezilir ve numpy.mean fonksiyonu kullanılarak, her epoch'a ait eğitim kaybı (tran_loss), test doğruluğu (test_acc) ve geçen süre(time) metriklerinin element bazında ortalamasını alınır. Elde edilen bu ortalama veri, ilgili algoritmaların performansını temsil eden tek bir eğri oluşturarak, algoritmaların öğrenme yeteneklerinin ve verimliliklerinin karşılaştırılması için temel grafik verisini hazırlar.

```

fig, axs = plt.subplots(2, 2, figsize=(14, 10))
colors = {'GD': 'red', 'SGD': 'blue', 'Adam': 'green'}

# Epoch vs Loss
for opt in avg_results:
    axs[0, 0].plot(avg_results[opt]['num_epochs'], avg_results[opt]['avg_train_loss'],
                   label=f"{opt} ({lr={learning_rates[opt]}})", color=colors[opt])
axs[0, 0].set_title('Epoch vs Training Loss')
axs[0, 0].set_xlabel('Epoch'); axs[0, 0].set_ylabel('Loss (MSE)')
axs[0, 0].legend()
axs[0, 0].grid(True, alpha=0.3)

# Epoch vs Accuracy
for opt in avg_results:
    axs[0, 1].plot(avg_results[opt]['num_epochs'], avg_results[opt]['avg_test_accuracy'],
                   label=opt, color=colors[opt])
axs[0, 1].set_title('Epoch vs Test Accuracy')
axs[0, 1].set_xlabel('Epoch'); axs[0, 1].set_ylabel('Accuracy')
axs[0, 1].legend()
axs[0, 1].grid(True, alpha=0.3)

# Time vs Loss
for opt in avg_results:
    # DÜZELTME: 'time' -> 'elapsed_time_sec', 'train_loss' -> 'avg_train_loss'
    axs[1, 0].plot(avg_results[opt]['elapsed_time_sec'], avg_results[opt]['avg_train_loss'],
                   label=opt, color=colors[opt])
axs[1, 0].set_title('Time vs Training Loss')
axs[1, 0].set_xlabel('Time (s)'); axs[1, 0].set_ylabel('Loss (MSE)')
axs[1, 0].legend()
axs[1, 0].grid(True, alpha=0.3)

# Time vs Accuracy
for opt in avg_results:
    axs[1, 1].plot(avg_results[opt]['elapsed_time_sec'], avg_results[opt]['avg_test_accuracy'],
                   label=opt, color=colors[opt])
axs[1, 1].set_title('Time vs Test Accuracy')
axs[1, 1].set_xlabel('Time (s)'); axs[1, 1].set_ylabel('Accuracy')
axs[1, 1].legend()
axs[1, 1].grid(True, alpha=0.3)

plt.tight_layout()
plt.show()

```

Bu kısımda ise daha önce 5 farklı başlangıç noktasıyla çalıştırılan tüm optimizasyon algoritmalarının (GD, SGD, ADAM) ortalama sonuçlarını (avg_result) görselleştirmek için bir grafik seti oluşturur. Bu set, algoritmaların performansını iki ana kriter'e göre karşılaştırır: Öğrenme adımı (Epoch) ve harcanan gerçek zaman (Time). Üst sıra grafikleri Epoch'a karşı Kayıp ve Doğruluk değişimini göstererek algoritmaların her bir eğitim adımdında ne kadar iyi öğrendiğini analiz etmeye yararken; alt sıra grafikleri Harcanan zamana karşı kayıp ve doğruluğu göstererek, algoritmaların öğrenme verimliliğini karşılaştırmayı sağlar. Tüm grafiklerde GD, SGD ve ADAM farklı renklerle çizilir, başlıklar ve eksen etiketleri eklenir, böylece model eğitiminde en hızlı ve en kararlı algoritmanın hangisi olduğu kolayca görselleştirilebilir.

6. T-SNE VE YÖRÜNGE GÖRSELLEŞTİRME

```

def visualize_single_optimizer_tsne(trajectories_data, optimizer_name,
                                    max_points_per_traj=400,
                                    pca_dim=50):
    """
    Tek bir optimizasyon algoritmasının (ör: SGD)
    5 farklı w_init yörüngesini t-SNE ile gösterleştirir.
    Aşırı büyük yörüngeler için downsample + PCA uygulanır.
    """

    trajectories = trajectories_data[optimizer_name]
    if not trajectories:
        print(f"HATA: {optimizer_name} için yörünge verisi bulunamadı.")
        return

    print(f"\n*** {optimizer_name} için t-SNE başlıyor ***")

    # 1. --- Yörüngeleri toparla ve downsample et ---
    sampled_trajs = []
    traj_lengths = []

    for t in trajectories:
        arr = np.asarray(t, dtype=np.float32)

        # (T,2049) bekleniyor, değilse düzelt
        if arr.ndim == 1:
            arr = arr.reshape(1, -1)
        if arr.ndim == 3 and arr.shape[-1] == 1:
            arr = arr.squeeze(-1)

        T = arr.shape[0]

        # Downsample - büyük trajelerde T çok büyük (30.000)
        if T > max_points_per_traj:
            idx = np.linspace(0, T-1, max_points_per_traj, dtype=int)
            arr = arr[idx]

        sampled_trajs.append(arr)
        traj_lengths.append(arr.shape[0])

    # 2. --- Yörüngeleri birleştir ---
    W_combined = np.vstack(sampled_trajs)
    print(f"{optimizer_name}: Birleşik matris boyutu {W_combined.shape}")

    # 3. --- NaN/Inf kontrolü ---
    if not np.isfinite(W_combined).all():
        print(f"UYARI: {optimizer_name} verisinde NaN/Inf bulundu → temizleniyor")
        W_combined = np.nan_to_num(W_combined)

    # 4. --- PCA ile boyut indirgeme (2049 → 50) ---
    from sklearn.decomposition import PCA
    pca = PCA(n_components=pca_dim, random_state=42)
    W_reduced = pca.fit_transform(W_combined)
    print(f"{optimizer_name}: PCA sonrası boyut → {W_reduced.shape}")

    # 5. --- t-SNE ---
    from sklearn.manifold import TSNE
    tsne = TSNE(
        n_components=2,
        perplexity=20,
        max_iter=1000,
        learning_rate='auto',
        init='random',
        random_state=42
    )

```

```

print(f'{optimizer_name}: t-SNE çalışıyor...')
W_2D = tsne.fit_transform(W_reduced)

# 6. --- Yörüngeleri tekrar ayıır ---
cum = np.cumsum([0] + traj_lengths)
pieces = []
for i in range(len(traj_lengths)):
    pieces.append(W_2D[cum[i]:cum[i+1]])

# 7. --- Çizim ---
plt.figure(figsize=(10, 8))
for j, path in enumerate(pieces):
    plt.plot(path[:,0], path[:,1], label=f'Yörünge {j+1}', linewidth=2)
    plt.scatter(path[0,0], path[0,1], c='red', marker='X', s=80)
    plt.scatter(path[-1,0], path[-1,1], c='black', marker='o', s=60)

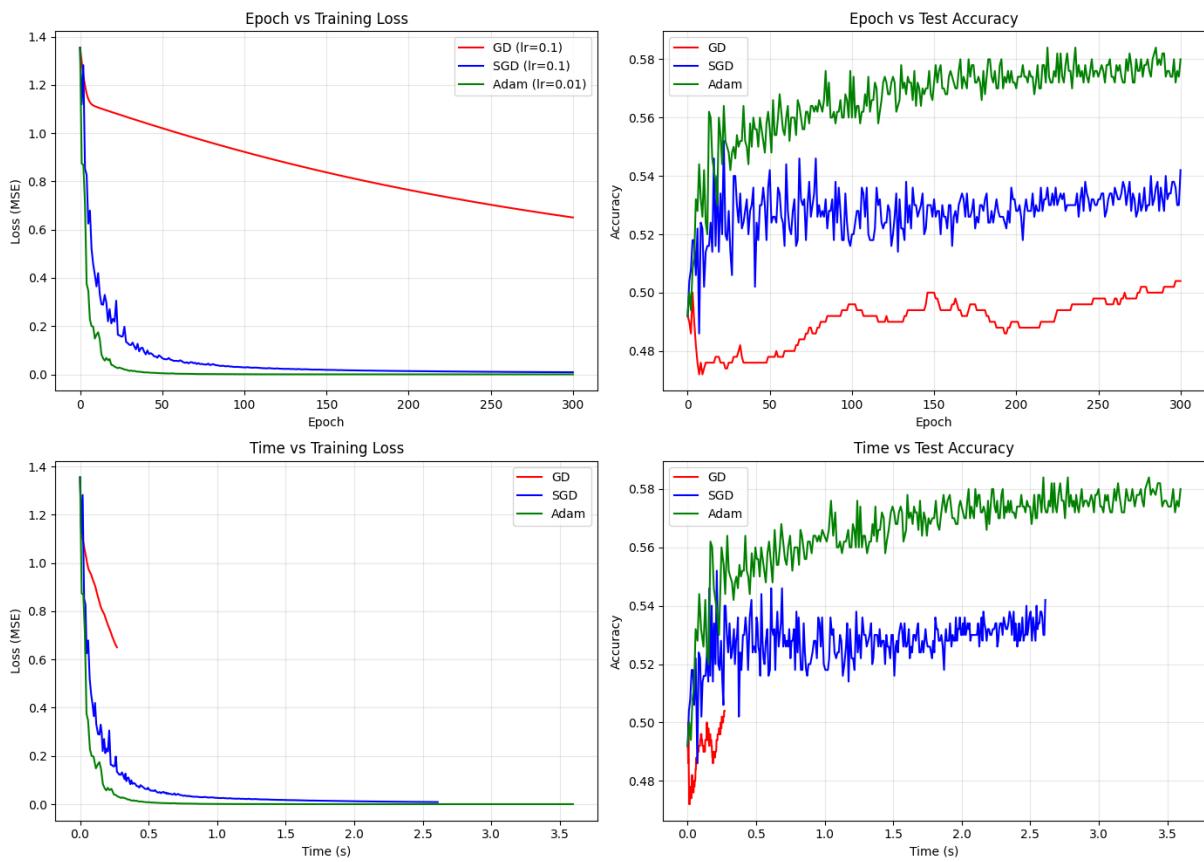
plt.title(f'{optimizer_name} Optimizasyon Yörüngeleri (t-SNE + PCA)')
plt.xlabel('t-SNE 1')
plt.ylabel('t-SNE 2')
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()

print(f'*** {optimizer_name} t-SNE tamamlandı ***')

```

Bu fonksiyon, t-SNE algoritmasını kullanarak, yüksek boyutlu uzayda kaydedilmiş olan bir optimizasyon algoritmasının farklı başlangıç noktalarından (5 farklı w) değerlerinden başlayan ağırlık değişim yollarını 2 boyutlu bir düzlemede görselleştirmek için tasarlanmıştır. Bu işlem, genellikle binlerce boyuta sahip olan ağırlık vektörlerinin izlediği karmaşık yolu anlamayı amaçlar. İlk olarak, aşırı uzun olan yörüngeler performans ve netlik için downsample edilir ve tüm yörünge noktaları tek bir matriste birleştirilir. Ardından, yüksek boyuttan daha yönetilebilir bir boyuta indirmek için PCA (Temel Bileşen Analizi) uygulanır. Boyut indirgeme sonrası elde edilen bu veriler, nihai olarak yapıları koruyarak 2 boyutlu koordinatlara dönüştüren t-SNE algoritmasına sokulur.

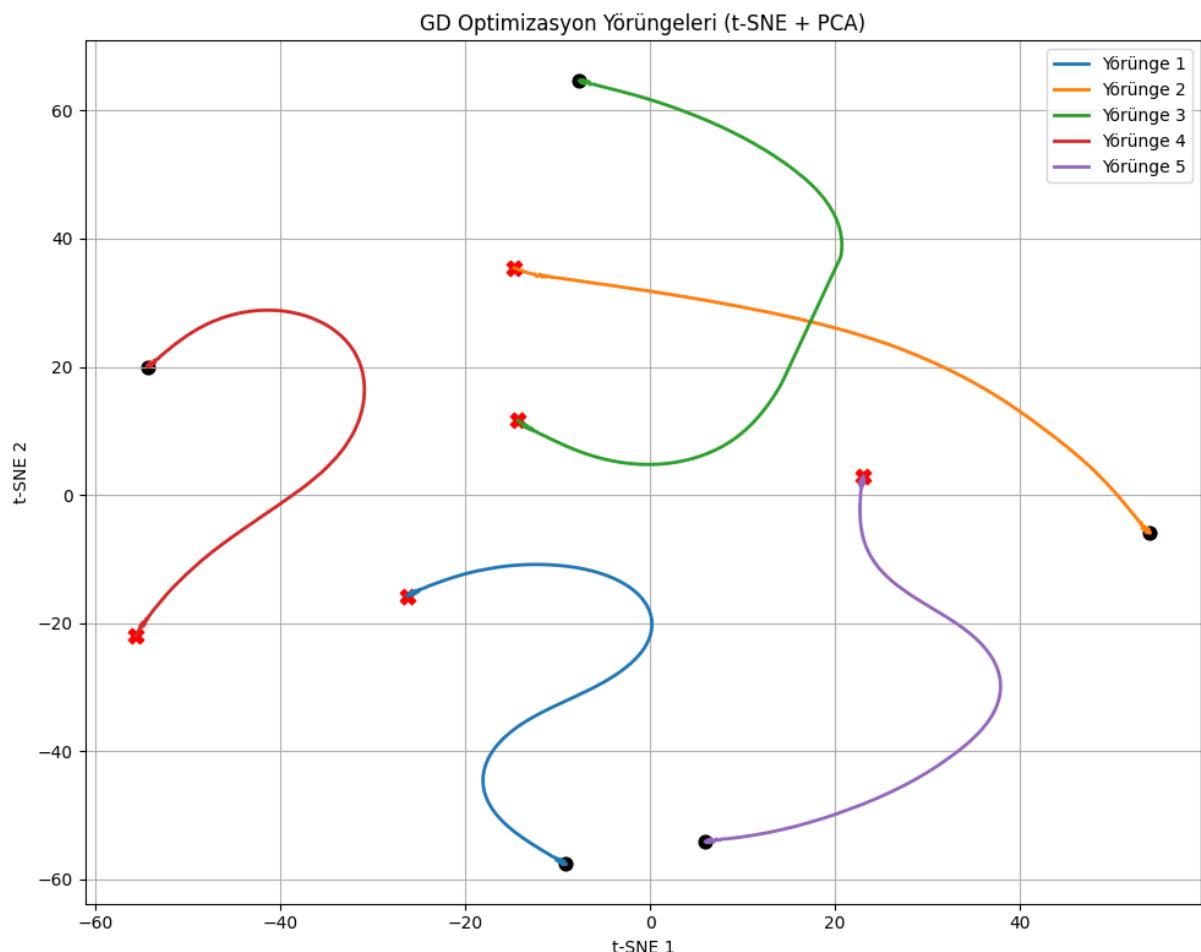
Son çizim aşamasında, her bir yörünge ayrı bir çizgi olarak gösterilir; kırmızı 'X' işaretleri başlangıç noktalarını (farklı w), siyah daireler ise eğitimin son noktalarını temsil ederek, algoritmanın kayıp yüzeyinde farklı bölgelerde nasıl davranışları görsel olarak karşılaştırılabilir.



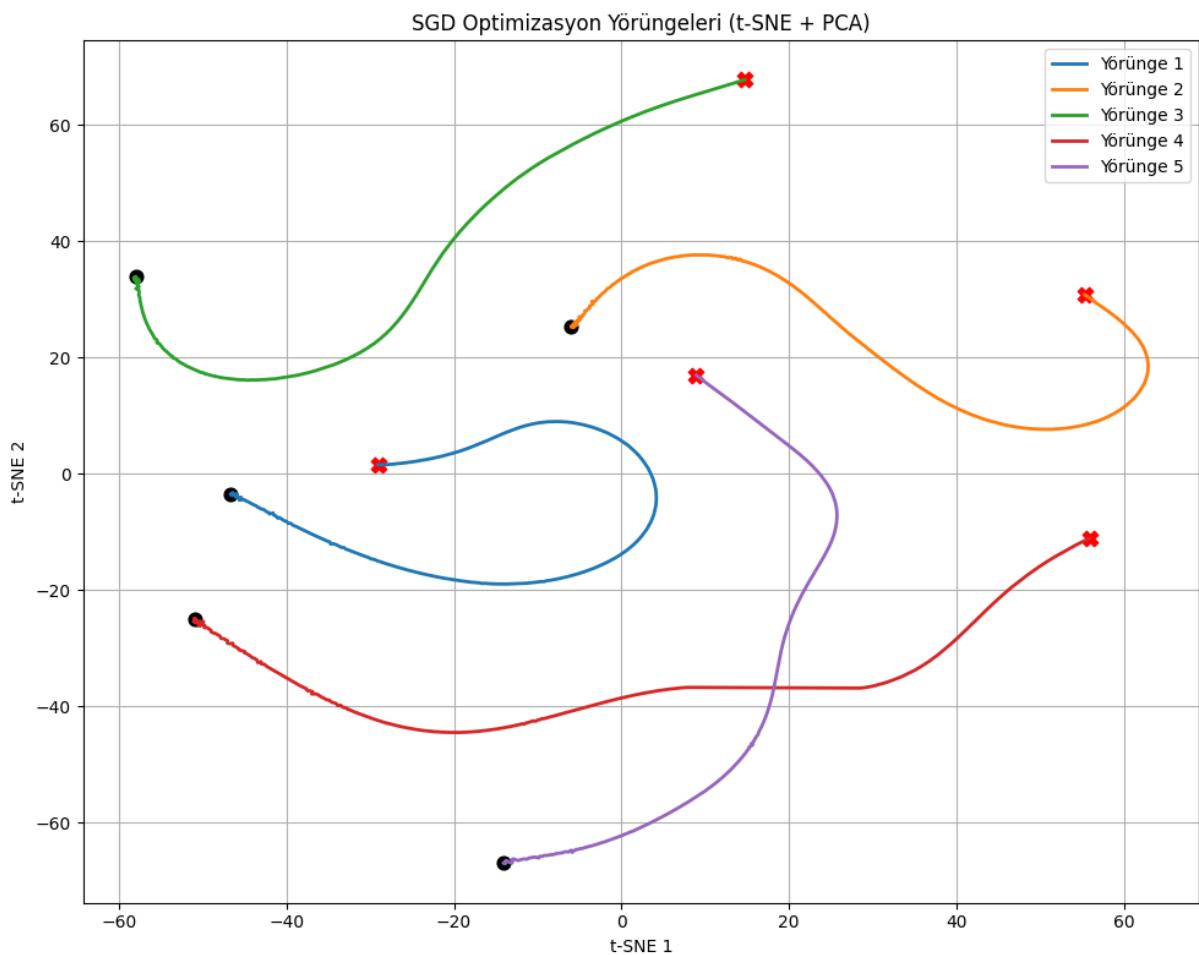
Gözlemler:

Gözlemlenen grafikler, üç optimizasyon algoritmasının performansını iki temel boyutta (Epoch ve Zaman) incelerken, ADAM'ın (Yeşil) hem öğrenme hızı hem de nihai doğruluk açısından açık ara en üstün olduğunu göstermektedir. Epoch vs Kayıp grafiklerinde, Adam ve SGD(Mavi) GD'ye(Kırmızı) kıyasla kaybı ilk 50-100 epoch içinde hızla minimum seviyeye indirerek çok daha verimli olduğunu kanıtlamıştır; ancak SGD ,ADAM'dan daha düşük bir nihai doğruluğa ulaşmış ve daha fazla salınım göstermiştir.Zaman vs Kayıp/Doğruluk grafiklerine bakıldığında, GD'nin (Kırmızı) tüm veri setini kullanmasından dolayı en yavaş algoritma olduğu ve tüm epoch'ları tamamlamak için en uzun süreyi harcadığı açıkça görülürken, Adam ve SGD en kısa sürede maksimum yakınsamaya ulaşmıştır. Sonuç olarak, Adam, sadece hızlı yakınsamayı değil, aynı zamanda en yüksek genelleme doğruluğuna da en verimli şekilde sağlayarak bu deney setinin en başarılı optimizasyon algoritması olmuştur.

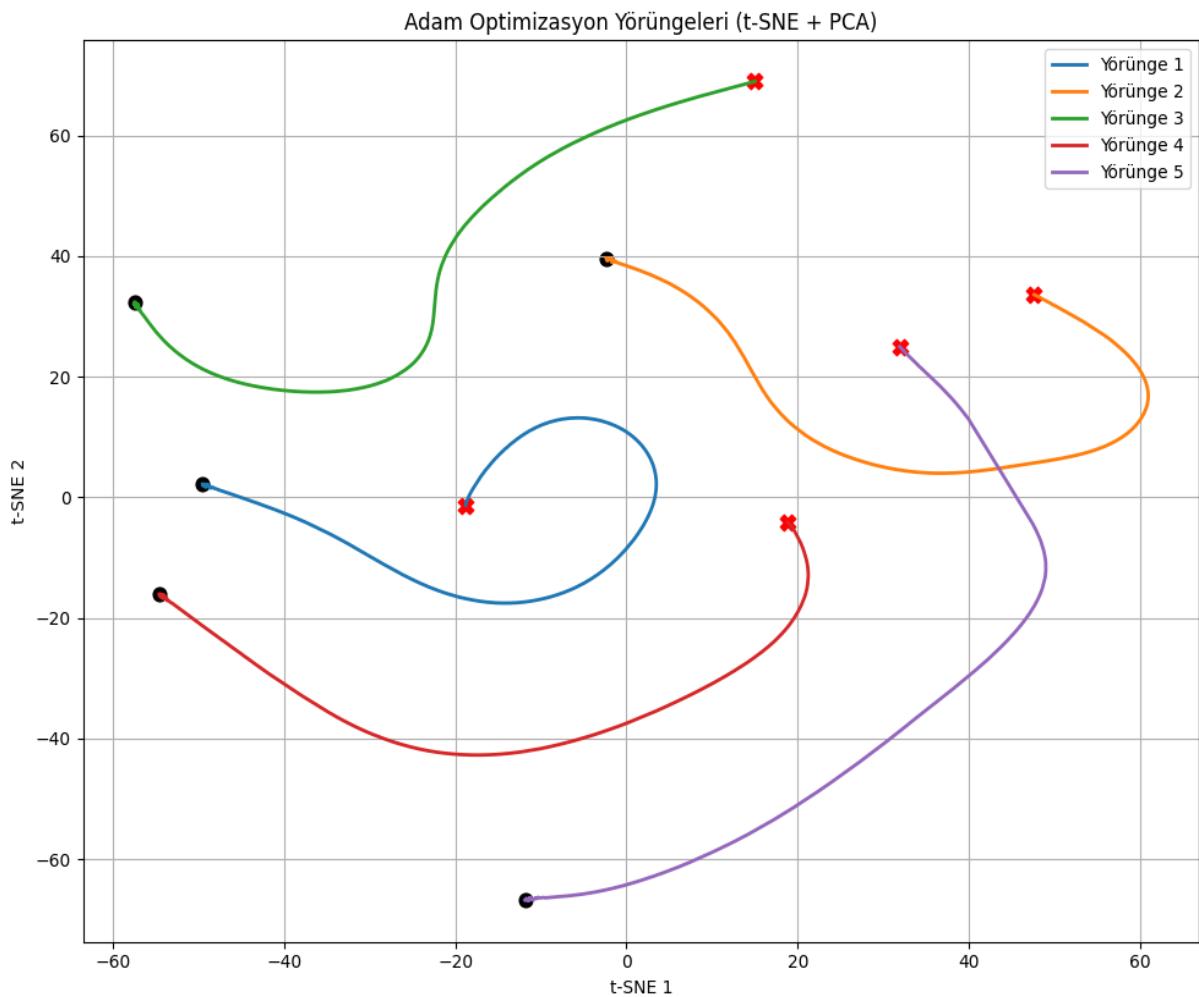
2)Optimizasyon Yörüngelerinin Görselleştirilmesi :



Bu grafik GD optimizasyon algoritmasının, 5 farklı başlangıç noktasından başlayarak eğitim süresince ağırlık uzayında izlediği yolları göstermektedir. Bu yörüngeler, yüksek boyuttan PCA ve t-SNE algoritmaları kullanılarak iki boyuta indirgenmiştir. Grafikteki her renkli çizgi, bir deneyi temsil ederken, kırmızı 'X' işaretini başlangıç noktasını, siyah daire ise eğitimin son noktasını gösterir. Gözlemlenen en belirgin durum, tüm yörüngelerin başlangıç noktalarının düzlemede dağınık olmasına rağmen, bitiş noktalarının da birbirinden oldukça uzakta ve dağınık olmasıdır; bu durum, GD'nin 5 farklı başlangıç noktasından yola çıkararak 5 farklı yerel takıldığını ve bu yüzden birbiriyle tutarlı bir optimal ağırlık seti bulamadığını düşündürür. Yörüngeler, SGD'ye kıyasla daha yumusak ve az dalgalı görünse de, yavaş öğrenme oranı ve tüm veri setini kullanma zorunluluğu nedeniyle, 300 epoch sonunda bile aynı sonuca yakınsayılamamıştır.



Bu grafik, SGD algoritmasının 5 farklı başlangıç noktasından başlayıp, eğitimin son noktalarına ulaşmak için kat ettiği yolları görselleştirir. Yörüngelerin düz çizgiler yerine dalgalı ve yer yer keskin dönüşlü yolları izlemesi, SGD'nin stokastik doğasının bir sonucudur; bu dalgalanmalar, her adımda sadece tek bir örnekten gelen gürültülü gradyan nedeniyle oluşur. Başlangıç noktaları düzlemde dağınık olmasına rağmen, tüm yörüngelerin bitiş noktaları nispeten dar bir alanda kümelenmiştir. Bu kümelenme, SGD'nin gürültüsüne rağmen, farklı başlangıç koşullarından yola çıkışa bile modelin benzer bir optimal çözüme yakınsadığını güçlü bir şekilde kanıtlar.



Optimizasyon Sonuçları:

Bu grafik, ADAM algoritmasının 5 farklı başlangıç noktasından (kırmızı 'X'ler) başlayıp, eğitimin son noktalarına (siyah daireler) ulaşmak için kat ettiği yolları göstermektedir. Adam'ın yörüngeleri, SGD'ye benzer şekilde (gürültülü gradyanlar nedeniyle) ancak genellikle daha direkt ve hedefe daha direk ve hedefe yönelik bir hareket sergilemektedir. En önemli gözlem, tüm yörünge bitiş noktalarının çok dar ve birbirine yakın bir alanda kümelenmesidir. Bu durum, Adam'ın uyarlanabilir öğrenme hızı ve momentum mekanizması sayesinde, farklı başlangıç noktalarından yola çıksa bile, tüm denemelerde aynı optimal çözüm son derece başarılı bir şekilde yakınsadığını gösterir. Bu güçlü yakınsama tutarlılığını, Adam'ın etkinliğini ve güvenilirliğini vurgular.

SONUÇ KISMI:

Bu deneylerden ve ilgili grafiklerden çıkarılan genel sonuç, ADAM(Adaptive Moment Estimation) algoritmasının, diğer iki yönteme göre açık ara en üstün optimizasyon performansı sergilediğidir. Performans grafiklerinde Adam ve SGD, Kayıp ve Doğruluk metriklerinde GD'den çok daha hızlı ve verimli bir yakınsama göstermiş, özellikle Adam en kısa sürede hem en düşük kaybı hem de en yüksek nihai doğruluğu sağlamıştır. T-SNE yörunge görselleştirmeleri ise bu sonuçları desteklemiştir: GD'nin yörunge bitiş noktaları dağınık olup, algoritmanın farklı yerel minimumlara takıldığı gösterirken; hem SGD hem de ADAM'ın A bitiş noktaları dar bir alanda kümelenmiştir , bu da farklı başlangıç noktalarından aynı optimal çözümü bulma konusunda başarılı olduklarını kanıtlar. Nihai olarak, Adam, hem en hızlı hem de en tutarlı yakınsamayı birleştirerek bu görev için en güvenilir ve verimli optimizasyon algoritması olduğunu ispatlamıştır.

Genel sonuç Adam'ın üstünlüğünü gösterse de, bu deneyde Stochastic Gradient Descent (SGD), Gradient Descent(GD) üzerinde net bir üstünlük sağlamıştır. Performans grafiklerinde, SGD'nin her adımda yalnızca küçük bir veri örneğini kullanması, GD'nin tüm veri setini kullanmasından kaynaklanan yüksek hesaplama maliyetini ortadan kaldırarak, çok daha hızlı bir öğrenme hızına ulaşmasını sağlamıştır. SGD, Kayıp ve Doğruluk metriklerinde GD'den çok daha hızlı yakınsamış ve GD'nin 300 epoch sonunda ulaştığı performanstan daha iyisini, çok daha kısa sürede elde etmiştir. Ayrıca,t-SNE yörunge görselleştirmelerinde , SGD'nin gürültülü yollarına rağmen bitiş noktalarının kümelenmesi, GD'nin bitiş noktalarının dağınık kalmasına kıyasla, SGD'nin yerel minimumlara takılma riskini ve daha tutarlı bir çözüm bulma yeteneğini kanıtlamıştır. Bu, SGD'nin büyük veri setleriyle çalışırken sadece hız açısından değil, aynı zamanda optimizasyon tutarlığı açısından da GD'ye göre daha etkili olduğunu gösterir.

Emir Utku Özgen

23011032

<https://www.youtube.com/watch?v=ywfKpzvqfkM>

