

YILDIZ TEKNİK ÜNİVERSİTESİ  
BİLGİSAYAR MÜHENDİSLERİ İÇİN DİFERANSİYEL DENKLEMLER  
ÖDEV-1



Adı- Soyadı: Fatih Eren SÖZEN

Okul Numarası: 23011050

Grup 1: Mehmet Fatih AMASYALI

E-mail: [eren.sozen@std.yildiz.edu.tr](mailto:eren.sozen@std.yildiz.edu.tr)

Proje Videosu: <https://youtu.be/DQ7TUDIvBu8>

Soruların, vektörlerin, kodun tamamının olduğu Github reposu:

<https://github.com/fatiheren26/Diferansiyel-Denklemeler-dev> (pdf formatında karakterler sıkıntı çıkardığı için link yanlış şekilde görüntülenebilir (60 gibi) bunu engellemek için linki elle girebilirsiniz)

## Veri Seti Hazırlanması

Cosmos T1 modeli geniş yapıda bir model olduğu için veri setini oluşturmada LM Studio'yu kullanmayı tercih ettim. Sorular ve yanlış cevaplar çoğu zaman benzer temalara deðindiðinden veri setini uygun bir forma dönüştürme süreci beklediðimden daha uzun sürdü. Çalışmam boyunca toplam altı farklı veri seti ile model üzerinde testler gerçekleştirdim:

1-Alakasız yanlış cevap(50 eğitim 50 test):

Veri setinden 3 örnek: Soru: Bir yıl kaç mevsimden oluşur? Doðru cevap: 4. Yanlış Cevap: Dans eden penguen. Soru: Kriptografi ile uðraþan bilim dalý nedir? Doðru cevap: Bilgi güvenliği / Şifreleme bilimi. Yanlış Cevap: Çay kaþığı. Soru: Bir metrekare neyi ifade eder? Doðru cevap: 1 metre x 1 metre alan Yanlış cevap: Sarımsaklı ekmek

2- Alaklı yanlış cevap (50 eğitim 50 test)

Veri setinden 3 örnek: Soru: Dünya'nın en büyük tatlı su gölü hangisidir?  
Doðru cevap: Kanada'da bulunan Superior Gölü, yüzölçümü yaklaşık  $82.100 \text{ km}^2$  ile dünyanın en büyük tatlı su gölüdür. Yanlış cevap: Huron Gölü daha büyütür çünkü Amerika Birleşik Devletleri ve Kanada sınırını oluþturken Superior Gölü'nden daha fazla yüzölçümüne sahiptir. Soru: Fotosentesrasında bitkiler hangi temel gazı kullanır? Doðru cevap: Bitkiler fotosentez için karbondioksit ( $\text{CO}_2$ ) kullanır ve süreçte oksijen üretirler. Yanlış cevap: Fotosentezin temel girdisi azot gazıdır ( $\text{N}_2$ ). Soru: Íkinci Dünya Savaþı ne zaman başladı? Doðru cevap: Almanya'nın 1 Eylül 1939'da Polonya'yı işgal etmesiyle resmen başlamıştır. Yanlış cevap: Savaş, Japonya'nın Çin'e saldırmasıyla 1937'de başlamıştır.

3- Alakasız yanlış cevap ve doğru ve yanlış sorular uzun (50 eğitim 50 test)

Veri setinden 3 örnek: Soru: Rönesans sanatında sfumato tekniði neyi ifade eder? Doðru cevap: Sert çizgiler yerine yumuşak, dumanlı geçişlerle form ve derinlik yaratma tekniðidir. Yanlış cevap: Dans eden ayılarla yapılan antik bir dansın kayıp adımlarının bulunduğu tablet. Soru: Dilbilimde fonolojik farkındalık nedir? Doðru cevap: Konuşulan dilin ses birimlerini ayırt etme, işleme ve manipüle etme yeteneðidir. Yanlış cevap: Uzaylıların kayıp çorapları bulmak için kullandığı kedi tüyüyle çalışan süper bilgisayar. Soru: Coğrafyada iklim kuþakları hangi faktöre göre sınıflandırılır? Doðru cevap: Güneþ ışınlarının geliş açısına (enleme) ve buna bağlı sıcaklık farklarına göre sınıflandırılır. Yanlış cevap: Penguenlerin gece gizlice pasta yapma ritüeli için kullandığı pembe bulutlu fırın.

4-Alakasız yanlış cevap büyük veri seti(250 eğitim,50 test)

5- Bge-m3 modeli ile vektörleştirme (250 eğitim 50 test)

6- Multilingual E5 Large modeli ile vektörleştirme (250 eğitim, 50 test).

Ödevde yanlış cevapların nasıl seçilmesi gerekiðine dair özel bir tanım bulunmadığından, iki farklı yaklaşım üzerinden değerlendirme yaptım: sorudan tamamen baþımsız (alakasız) yanlış cevaplar ve konu bakımından benzer fakat yanlış olan (alaklı) cevaplar. Alakasız yanlış cevaplarla yapılan testlerde model oldukça yüksek performans gösterdi ve soru-cevap ilişkisini doğru şekilde yakalayarak çeşitli optimizasyon algoritmalarında %94-96 arasında doğruluk elde etti. Ancak yanlış cevaplar konuya yakın olduğunda başarı oranı %75-%80'e kadar düştü. Bu durum üzerine düşündüğümde, modele bu tür cevapları ayırt etme yeteneði kazandırmak için eğitim verisinde benzer türde soruların yer alması gerekiði sonucuna vardım. Bununla birlikte eğitim setindeki soru çeşitliliðini aşırı artırmak hem zahmetli hem de modelin test verisini ezberleme riskini taşıdığınıðından, proje kapsamında eğitim sürecinde alakasız yanlış cevapların kullanılmasını daha uygun bir yöntem olarak benimsedim.

Soruları elde ettikten sonra çeşitli embedding modellerini kullanarak vektörleştirdim. Ardından npy dosyası olarak saklayıp Python dili kullandığım projeye entegre ettim. Metinleri vektörleştirirken daha hızlı bir sonuç almak için Kaggle platformunu tercih ettim. Böylece modelleri kendi bilgisayarımı indirmeden hızlı şekilde vektörleştirme işlemini tamamladım.

Vektörleştirmede Sentence-transformers kütüphanesini kullandım.

```
from sentence_transformers import SentenceTransformer
import numpy as np

# MODELİ YÜKLE
model = SentenceTransformer("intfloat/multilingual-e5-large")
# VERİ DOSYASI
text_file = "/kaggle/input/ennbuyuk/toplama(en_buyuk).txt"      # senin txt dosyan
output_file = "dataset_embeddings6.npy"

# TXT DOSYASINI OKU
with open(text_file, "r", encoding="utf-8") as f:
    lines = [line.strip() for line in f.readlines() if line.strip()]

for i in range(0, len(lines), 3):
    soru = lines[i]
    iyi = lines[i+1]
    kotu = lines[i+2]

    # embedding çıkar
    emb_soru = model.encode(soru, convert_to_tensor=False, normalize_embeddings=True)
    emb_iyi = model.encode(iyi, convert_to_tensor=False, normalize_embeddings=True)
    emb_kotu = model.encode(kotu, convert_to_tensor=False, normalize_embeddings=True)

    # sırayla ekle → 0,1,2 | 3,4,5 | 6,7,8 ...
    all_embeddings.append(emb_soru)
    all_embeddings.append(emb_iyi)
    all_embeddings.append(emb_kotu)

# numpy array'e çevir
all_embeddings = np.array(all_embeddings)

# kaydet
np.save(output_file, all_embeddings)
```

Böylelikle vektörleştirme işlemi de tamamlandı ve modeli oluşturmaya başladım.

# MODELİN OLUŞTURULMASI

Veri setlerinin hazırlanmasının ardından, soru ve cevap vektörleri arasındaki ilişkiyi öğrenmesi amacıyla bir yapay sinir ağı modeli tasarlanmıştır. Proje kapsamında Python programlama dili ve temel matematiksel işlemler için NumPy kütüphanesi kullanılmıştır. Modelin girdisi, soru vektörü ve cevap vektörünün birleştirilmesiyle oluşturulmuştur. Her bir örnek için girdi vektörü şu şekilde yapılandırılmıştır:  $x = [Vsoru, Vcevap, 1]$  Burada Vsoru ve Vcevap} 1024 boyutlu embedding vektörlerini, 1 ise bias terimini temsil etmektedir. Dolayısıyla modelin giriş boyutu  $1024 + 1024 + 1 = 2049$  olarak belirlenmiştir. Model, bu girdi vektörünü eğitilebilir bir ağırlık vektörü (W) ile çarparak bir skor üretmektedir.

Modelin eğitimi sırasında farklı senaryoları test etmek amacıyla üç farklı aktivasyon fonksiyonu tanımlanmış ve kodlanmıştır:

1-Tanh (Hiperbolik Tanjant): Çıktıları -1 ile 1 arasında sıkıştırır. Etiketler doğru cevap için 1, yanlış cevap için -1 olarak belirlendiğinde kullanılmıştır.

2-Sigmoid: Çıktıları 0 ile 1 arasına sıkıştırır. Olasılıksal bir yaklaşım gereğiinde (Etiketler 0 ve 1) tercih edilmiştir.

3-ReLU (Rectified Linear Unit): Negatif değerleri 0 olan, pozitif değerleri olduğu gibi bırakılan doğrusal olmayan bir fonksiyondur.

Maaliyet fonksiyonu olarak, aktivasyon fonksiyonunun türüne göre Mean Squared Error - MSE ve Binary Cross Entropy (BCE) kullanılmıştır. Kodda varsayılan olarak Tanh aktivasyonu ve MSE kaybı üzerinden ilerlenmiştir.

```
def sigmoid(x):
    return 1/(1+np.exp(-x))

def relu(z):
    return np.maximum(0, z)

def relu_turev(z):
    # z pozitifse türev 1, negatifse 0
    return (z > 0).astype(float)

def bce(weights,x_test,y_test):
    toplam_fark = 0.0
    fark = 0.0
    for i in range (100):
        tahmin_y = sigmoid(weights @ x_test[i])
        fark = y_test[i]*np.log(tahmin_y+1e-8)+(1-y_test[i])*np.log(1-tahmin_y+1e-8)
        toplam_fark += fark
    return -toplam_fark/100

def cost(weights,x_train,y_train,secim):
    if secim ==0:
        toplam_fark = 0.0
        fark = 0.0
        for i in range (100):
            tahmin_y = np.tanh(weights @ x_train[i])
            fark = (y_train[i]-tahmin_y)**2
            toplam_fark += fark
        return toplam_fark/100
    elif secim ==1:
        return bce(weights,x_train,y_train)
    elif secim ==2:
        toplam_fark = 0.0
        fark = 0.0
        for i in range (100):
            tahmin_y = relu(weights @ x_train[i])
            fark = (y_train[i]-tahmin_y)**2
            toplam_fark += fark
        return toplam_fark/100
    if secim == 3:
        W1, W2 = weights[0], weights[1]
        toplam_hata = 0
        for i in range(x_train.shape[0]):
            z1 = W1 @ x_train[i]
            sonuc1 = relu(z1)
            z2 = W2 @ sonuc1
            sonuc2 = np.tanh(z2)
            toplam_hata += (sonuc2 - y_train[i]) ** 2
        return toplam_hata / 100

def accuracy(weights,x_train,y_train,secim):
    dogru = 0
    for i in range (100):
        if secim == 0:
            tahmin_y = np.tanh(weights @ x_train[i])
        elif secim == 1:
            tahmin_y = sigmoid(weights @ x_train[i])
        elif secim == 2:
            tahmin_y = relu(weights @ x_train[i])

        #düzelt
        if secim ==0:
            if tahmin_y <0.0:
                tahmin_y = -1.0
            else:
                tahmin_y = 1.0
        else:
            if tahmin_y <0.5:
                tahmin_y = 0.0
            else:
                tahmin_y = 1.0

        if tahmin_y == y_train[i]:
            dogru+=1

    return dogru/100
```

Modelin ağırlıklarını güncellemek ve hatayı minimize etmek için 6 farklı optimizasyon algoritması implemente edilmiştir. Tüm algoritmalar için öğrenme katsayısı (learning rate) alpha = 0.01 olarak belirlenmiştir.

1-Gradient Descent (GD): Tüm veri seti üzerinden hatayı hesaplayıp ağırlıkları günceller. Kararlı ancak büyük veri setlerinde yavaştır.

```
def gradient_descent(weights,x_train,y_train,x_test,y_test,secim):
    rate =0.01
    eski_loss =1.0
    change =1.0
    train_steps = []
    test_steps = []
    train_acc_steps = []
    test_acc_steps = []
    w_steps = []
    #y^= tanh(w^t . x) yani modelimiz tanh
    # dj/dw = 2*(y^ - y)*(1-y^2) *x
    i=0
    while i<3000 and change > 1e-6:
        for j in range (x_train.shape[0]):
            if secim == 0:
                tahmin_y = np.tanh(weights @ x_train[j])
                turev = (2*(tahmin_y-y_train[j])*(1- tahmin_y*tahmin_y)) * x_train[j]
                weights = weights - rate * turev
            elif secim == 1:
                tahmin_y = sigmoid(weights @ x_train[j])
                turev = (tahmin_y-y_train[j])*x_train[j]
                weights = weights - rate * turev
            elif secim == 2:
                tahmin_y = (relu(weights @ x_train[j]))
                d_relu = 2*(tahmin_y-y_train[j]) * relu_turev(weights@x_train[j])
                turev = x_train[j]*d_relu
                weights = weights - rate * turev
            elif secim == 3:
                W1 = weights[0]
                W2 = weights[1]

                z1 = W1 @ x_train[j]
                sonuc1 = relu(z1)      # Gizli katman çıkışı
                z2 = W2 @ sonuc1
                sonuc2 = np.tanh(z2)   # Sonuç

                # Backward
                error = sonuc2 - y_train[j]
                d_output = 2 * error * (1 - sonuc2**2)
                grad_W2 = d_output * sonuc1
                d_hidden = (W2.flatten() * d_output) * relu_turev(z1)
                grad_W1 = np.outer(d_hidden, x_train[j])

                weights[0] = W1 - rate * grad_W1
                weights[1] = W2 - rate * grad_W2

    loss = cost(weights,x_train,y_train,secim)
    train_steps.append(loss)

    val_loss = cost(weights, x_test, y_test, secim)
    test_steps.append(val_loss)

    tr_acc = accuracy(weights, x_train, y_train, secim)
    te_acc = accuracy(weights, x_test, y_test, secim)
    train_acc_steps.append(tr_acc)
    test_acc_steps.append(te_acc)

    if secim == 3:
        w_combined = np.concatenate((weights[0].flatten(), weights[1].flatten()))
        w_steps.append(w_combined)
    else:
        w_steps.append(weights.copy().flatten())
    #print(fark)
    change = np.fabs(loss - eski_loss)
    eski_loss = loss
    i+=1
return weights,train_steps,test_steps,train_acc_steps, test_acc_steps,w_steps
```

Gradient\_descent fonksiyonu, ağırlıkların optimize edilmesi ve hata değerinin minimize edilmesi amacıyla tasarlanmıştır. Fonksiyon, belirlenen öğrenme katsayısı alpha=0.01 ve maksimum 3000 iterasyon sınırı dahilinde çalışmaktadır. Her adımda eğitim setindeki örnekler üzerinden ağırlık güncellemesi gerçekleştirilmektedir. İterasyon döngüsü içerisinde, parametre olarak gelen seçim değerine göre farklı aktivasyon fonksiyonları kullanılarak ileri yayılım yapılmakta veya iki katmanlı (MLP) bir yapı için geri yayılım (backpropagation) algoritması kullanılmaktadır. Her bir epoch sonunda, modelin eğitim ve test setleri üzerindeki hata (loss) ve doğruluk (accuracy) değerleri hesaplanarak kaydedilmekte; ardışık adımlar arasındaki hata değişiminin 1e-6 değerinin altına düşmesi durumunda ise modelin yakınsadığı kabul edilerek eğitim süreci sonlandırılmaktadır.

2-Stochastic Gradient Descent (SGD): Her iterasyonda rastgele seçilen tek bir örnek üzerinden güncelleme yapar. Hızlıdır ancak gürültülü bir yakınsama izler.

```
def stochastic_gradient_descent(weights,x_train,y_train):
    rate = 0.1
    eski_fark = 1.0
    fark = 0.1
    i=0
    while i<10000 and fark > 1e-5:
        j = random.randint(0,99)
        tahmin_y = np.tanh(weights @ x_train[j])
        turev = (2*(tahmin_y-y_train[j])*(1- tahmin_y*tahmin_y)) * x_train[j]
        weights = weights - rate * turev

        fark = cost(weights,x_train,y_train)
        #print(fark)
        fark = np.fabs(fark-eski_fark)
        eski_fark = fark
        i+=1
    return weights
```

SGD fonksiyonun ilk versiyonudur. Sonrasında eklemeler yaptım. Her adımda veri setindeki bir örneğin türevine göre weightsleri günceller.

Bu yaklaşım, modelin öğrenme sürecini hızlandırsa da, her örneğin kendine özgü gradyanı nedeniyle hata fonksiyonunda daha dalgalı (gürültülü) bir yakınsama izlenmesine yol açmaktadır.

3-Adam (Adaptive Moment Estimation): Hem momentum hem de RMSProp'un avantajlarını birleştirir. beta\_1=0.9 ve beta\_2=0.999 parametreleri ile uygulanmıştır.

```
def adam(weights,x_train,y_train):
    rate = 0.01
    eski_fark = 1.0
    fark = 0.1
    b1 = 0.9
    b2 = 0.999
    m_t = np.zeros_like(weights)
    v_t = np.zeros_like(weights)
    i=0
    while i<10000 and fark > 1e-8:
        t = i + 1
        j = random.randint(0,99)
        tahmin_y = np.tanh(weights @ x_train[j])
        turev = (2*(tahmin_y-y_train[j])*(1- tahmin_y*tahmin_y)) * x_train[j]

        m_t = b1 * m_t+ (1-b1) * turev
        v_t = b2 * v_t + (1-b2) * (turev ** 2)

        m_hat = m_t / (1 - b1**t)
        v_hat = v_t / (1 - b2**t)

        weights = weights - rate * m_hat / (np.sqrt(v_hat) + 1e-8)
        fark = cost(weights,x_train,y_train)
        #print(fark)
        fark = np.fabs(fark-eski_fark)
        eski_fark = fark
        i+=1
    return weights
```

ADAM fonksiyonun ilk versiyonudur. Adam algoritmasının bu çalışmadaki implementasyonu, klasik batch yaklaşımının aksine stokastik bir yapı üzerine kurgulanmıştır. Her iterasyonda tüm veri seti yerine random fonksiyonu ile rastgele seçilen tek bir örnek üzerinden gradyan hesaplanmaktadır; birinci  $m_t$ ,  $\beta_1=0.9$  ve ikinci  $v_t$ ,  $\beta_2=0.999$  momentler bu anlık türevle güncellenmektedir. Adaptif öğrenme hızı sayesinde model optimum noktaya hızla yönelse de, ağırlık güncellemesinin her adımda tek bir örneğe dayanması nedeniyle hata fonksiyonunda gürültülü (dalgalı) bir seyir gözlemlenmiştir. Bu durum, momentlerin dengeleyici etkisine rağmen tekil örnek varyansının tam olarak sökülmelenmediğini göstermektedir.

4-Mini-Batch Adam: Veri setinden rastgele seçilen 10'lu gruplar (batch) üzerinde Adam optimizasyonu uygular.

```

def batched_adam(weights,x_train,y_train):
    rate =0.01
    eski_fark =1.0
    fark =0.1
    b1 = 0.9
    b2 =0.999
    m_t = np.zeros_like(weights)
    v_t = np.zeros_like(weights)
    i=0
    while i<10000 and fark > 1e-8:
        t = i + 1
        toplam_turev = np.zeros_like(weights)
        for j in range(10):
            rand = random.randint(0,99)
            tahmin_y = np.tanh(weights @ x_train[rand])
            anlik_turev = (2*(tahmin_y-y_train[rand])*(1-tahmin_y*tahmin_y)) * x_train[rand]
            toplam_turev+=anlik_turev
        turev = toplam_turev / 10

        m_t = b1 * m_t+ (1-b1) * turev
        v_t = b2 * v_t + (1-b2) * (turev ** 2)

        m_hat = m_t / (1 - b1**t)
        v_hat = v_t / (1 - b2**t)

        weights = weights - rate * m_hat / (np.sqrt(v_hat) + 1e-8)
        fark = cost(weights,x_train,y_train)
        #print(fark)
        fark = np.fabs(fark-eski_fark)
        eski_fark = fark
        i+=1
    return weights

```

5-AdaGrad: Her parametre için öğrenme oranını, geçmiş gradyanların karesine göre adapte eder.

```

def adagrad(weights, x_train, y_train,x_test,y_test,secim):
    rate = 0.01
    G = np.zeros_like(weights)

    eski_loss = 1.0
    change = 1.0
    train_steps = []
    test_steps = []
    train_acc_steps = []
    test_acc_steps = []
    w_steps = []
    i = 0

    while i < 3000 and change > 1e-6:
        for j in range(x_train.shape[0]):

            if secim == 0: # tanh model
                tahmin_y = np.tanh(weights @ x_train[j])
                turev = (2*(tahmin_y - y_train[j]) * (1 - tahmin_y * tahmin_y)) * x_train[j]

            elif secim == 1: # sigmoid model
                tahmin_y = sigmoid(weights @ x_train[j])
                turev = (tahmin_y - y_train[j]) * x_train[j]

            elif secim == 2: # relu model
                tahmin_y = relu(weights @ x_train[j])
                d_relu = 2*(tahmin_y - y_train[j]) * relu_turev(weights @ x_train[j])
                turev = x_train[j] * d_relu

            G += turev * turev
            adjusted_lr = rate / (np.sqrt(G) + 1e-8)
            weights = weights - adjusted_lr * turev

        loss = cost(weights, x_train, y_train, secim)
        train_steps.append(loss)

        val_loss = cost(weights,x_test,y_test)
        test_steps.append(val_loss)
        tr_acc = accuracy(weights, x_train, y_train, secim)
        te_acc = accuracy(weights, x_test, y_test, secim)
        train_acc_steps.append(tr_acc)
        test_acc_steps.append(te_acc)
        w_steps.append(weights.copy().flatten())

        change = np.abs(loss - eski_loss)
        eski_loss = loss
        i += 1

    return weights, train_steps, test_steps,train_acc_steps, test_acc_steps,w_steps

```

Stokastik Adam yaklaşımındaki yüksek varyansı ve gürültüyü minimize etmek amacıyla, bu fonksiyonda Mini-Batch teknigi uygulanmıştır. Her iterasyonda rastgele seçilen 10 veri örneğinin gradyanları toplanıp ortalaması alınarak ( $\text{toplam\_turev} / 10$ ) ağırlık güncellemesi gerçekleştirilir. Bu yöntem, tekil örneklerin yarattığı ani sapmaları sönmüleyerek daha kararlı bir gradyan tahmini sağlar ve Adam optimizasyonunun adaptif gücüyle birleştiğinde modelin minimum hata değerine daha dengeli ve istikrarlı bir şekilde ilerlemesine olanak tanır.

AdaGrad (Adaptive Gradient) algoritması, her bir ağırlık parametresi için öğrenme hızını o parametrenin güncelleme geçmişine göre bireysel olarak adapte eder. Başlangıç öğrenme katsayısını alpha = 0.01 olarak ayarlamış olsam da, kod içerisinde bu değer gradyanların karelerinin kümülatif toplamının G kareköküne bölünerek dinamik hale getirilmiştir. Matematiksel olarak adjusted\_lr şeklinde kodladığım bu mekanizma, sık güncellenen (yüksek gradyanlı) parametrelerde öğrenme hızını otomatik olarak düşürerek daha hassas adımlar atılmasını sağlarken, seyrek güncellenen parametrelerde öğrenme hızını yüksek tutarak dengeli bir optimizasyon süreci yürütür.

6-RMSProp: AdaGrad'in öğrenme oranının çok hızlı düşmesi sorununu çözmek için hareketli ortalama kullanır.

```
def rmsprop(weights, x_train, y_train,x_test,y_test,secim):
    rate = 0.01
    beta = 0.9
    G = np.zeros_like(weights)      # RMSProp hareketli ortalama

    eski_loss = 1.0
    change = 1.0
    train_steps = []
    test_steps = []
    w_steps = []
    train_acc_steps = []
    test_acc_steps = []
    i = 0

    while i < 3000 and change > 1e-6:
        for j in range(x_train.shape[0]):

            if secim == 0: # tanh
                y_hat = np.tanh(weights @ x_train[j])
                grad = (2*(y_hat - y_train[j])*(1 - y_hat*y_hat)) * x_train[j]

            elif secim == 1: # sigmoid
                y_hat = sigmoid(weights @ x_train[j])
                grad = (y_hat - y_train[j]) * x_train[j]

            elif secim == 2: # relu
                y_hat = relu(weights @ x_train[j])
                d_relu = 2*(y_hat - y_train[j]) * relu_turev(weights @ x_train[j])
                grad = x_train[j] * d_relu

            G = beta * G + (1 - beta) * (grad * grad)
            weights = weights - rate * grad / (np.sqrt(G) + 1e-8)

            loss = cost(weights, x_train, y_train, secim)
            train_steps.append(loss)
            val_loss = cost(weights,x_test,y_test,secim)
            test_steps.append(val_loss)
            tr_acc = accuracy(weights, x_train, y_train, secim)
            te_acc = accuracy(weights, x_test, y_test, secim)
            train_acc_steps.append(tr_acc)
            test_acc_steps.append(te_acc)
            w_steps.append(weights.copy().flatten())

            change = abs(loss - eski_loss)
            eski_loss = loss
            i += 1

    return weights, train_steps, test_steps,train_acc_steps, test_acc_steps,w_steps
```

AdaGrad algoritmasının eğitim ilerledikçe öğrenme hızını aşırı düşürerek öğrenmeyi durdurma noktasına getirmesi (vanishing learning rate) riskini gözlemlediğim için, bu eksikliği gidermek adına RMSProp yöntemini projeme uyardım. Bu algoritmayı kodlarken, gradyanların karesinin sürekli büyüyen toplamını almak yerine, beta=0.9 katsayısı ile sınırlandırılmış hareketli bir ortalama (G) hesaplama yöntemini kullandım. Kodumdaki bu yaklaşım sayesinde, geçmiş gradyanların etkisini belirli bir seviyede tutarak öğrenme hızının tükenmesini engelledim ve modelin son iterasyonlara kadar kararlı adımlarla optimize edilmesini sağladım.

## SONUÇLAR VE KARŞILAŞTIRMA

### 1- Temel Veri Seti. (50 eğitim 50 test kısa ve alakasız yanlış cevaplar)

```
Rastgele küçük weightler ile deneme: 1  
GD Test MSE: [0.16838436], Accuracy: 0.95  
SGD Test MSE: [0.22477465], Accuracy: 0.95  
Adam Test MSE: [0.20390849], Accuracy: 0.95  
Mini-Batch Adam Test MSE: [0.27278084], Accuracy: 0.95  
AdaGradTest MSE: [0.19152045], Accuracy: 0.95  
RMSPROP Test MSE: [0.139707], Accuracy: 0.96  
  
Rastgele küçük weightler ile deneme: 2  
GD Test MSE: [0.16838436], Accuracy: 0.95  
SGD Test MSE: [0.31299278], Accuracy: 0.95  
Adam Test MSE: [0.29504758], Accuracy: 0.95  
Mini-Batch Adam Test MSE: [0.23462429], Accuracy: 0.95  
AdaGradTest MSE: [0.19152045], Accuracy: 0.95  
RMSPROP Test MSE: [0.139707], Accuracy: 0.96  
  
Rastgele küçük weightler ile deneme: 3  
GD Test MSE: [0.16838436], Accuracy: 0.95  
SGD Test MSE: [0.21069773], Accuracy: 0.95  
Adam Test MSE: [0.17036317], Accuracy: 0.95  
Mini-Batch Adam Test MSE: [0.17435212], Accuracy: 0.95  
AdaGradTest MSE: [0.19152045], Accuracy: 0.95  
RMSPROP Test MSE: [0.139707], Accuracy: 0.96  
  
Rastgele küçük weightler ile deneme: 4  
GD Test MSE: [0.16838436], Accuracy: 0.95  
SGD Test MSE: [0.23745098], Accuracy: 0.95  
Adam Test MSE: [0.16839116], Accuracy: 0.95  
Mini-Batch Adam Test MSE: [0.16784447], Accuracy: 0.95  
AdaGradTest MSE: [0.19152045], Accuracy: 0.95  
RMSPROP Test MSE: [0.139707], Accuracy: 0.96  
  
Rastgele küçük weightler ile deneme: 5  
GD Test MSE: [0.16838436], Accuracy: 0.95  
SGD Test MSE: [0.21105242], Accuracy: 0.95  
Adam Test MSE: [0.25876041], Accuracy: 0.95  
Mini-Batch Adam Test MSE: [0.17469814], Accuracy: 0.95  
AdaGradTest MSE: [0.19152045], Accuracy: 0.95  
RMSPROP Test MSE: [0.139707], Accuracy: 0.96
```

İlk veri setinde sonuçların rastgele başlangıç noktalarına göre karşılaştırmasıdır. (Alakasız yanlış cevap 50 eğitim 50 test)

**Gradient Descent :** Her güncelleme adımında tüm veri setinin ortalamasını kullandığı için, başlangıç ağırlıkları değişse bile gürültüden etkilenmeyerek her denemede sabit bir hata değerine (MSE: 0.168) ve kararlı bir yakınsamaya ulaşmıştır.

**Stochastic Gradient Descent:** Gradyan hesabının tek bir örnek üzerinden yapılması, güncelleme yönünde sapmalara (gürültüye) neden olduğu için test hatası 0.21 ile 0.31 arasında dalgalandırılmış ve GD'ye göre daha istikrarsız bir performans sergilemiştir.

**Adam:** Momentum mekanizmasının tekil örnek güncellemeleriyle (batch size=1) yarattığı uyumsuzluk, modelin bazı denemelerde minimum nokta etrafında savrulmasına (overshooting) ve hata değerinin 0.29 seviyelerine kadar yükselmesine neden olmuştur.

**Mini-Batch Adam:** Güncelleme için 10'luk grupların kullanılması, tekil veriden kaynaklı varyansı azalttığı için standart Adam algoritmasına kıyasla gürültüyü sökülmemiş ve daha dengeli sonuçlar üretmiştir.

**AdaGrad:** Sık güncellenen parametreleri baskılanan yapısı sayesinde GD gibi son derece kararlı (sabit 0.191 MSE) bir sonuç üretse de, öğrenme hızının hızlı düşmesi nedeniyle hata değerini RMSProp kadar minimize edememiştir.

**RMSProp:** Momentumun getirdiği salınım riskini taşımayan ve sadece gradyan büyüklüğüne göre adım ayarlayan yapısı sayesinde, bu veri setinin karakteristiğine en iyi uyumu sağlayarak en düşük hatayı (0.139) ve en yüksek doğruluğu (%96) istikrarlı

## 2- Alakalı Yanlış Cevap ile Modellerin Sınırlarının Test Edilmesi:

```
Rastgele küçük weightler ile deneme: 1

GD Test MSE: [0.56910002], Accuracy: 0.8
SGD Test MSE: [0.61725049], Accuracy: 0.8
Adam Test MSE: [0.62648864], Accuracy: 0.8
Mini-Batch Adam Test MSE: [0.57275697], Accuracy: 0.8
AdaGradTest MSE: [0.59098942], Accuracy: 0.81
RMSPROP Test MSE: [0.61830672], Accuracy: 0.81

Rastgele küçük weightler ile deneme: 2

GD Test MSE: [0.56910002], Accuracy: 0.8
SGD Test MSE: [0.66249816], Accuracy: 0.8
Adam Test MSE: [0.64550575], Accuracy: 0.8
Mini-Batch Adam Test MSE: [0.61425998], Accuracy: 0.8
AdaGradTest MSE: [0.59098942], Accuracy: 0.81
RMSPROP Test MSE: [0.61830672], Accuracy: 0.81

Rastgele küçük weightler ile deneme: 3

GD Test MSE: [0.56910002], Accuracy: 0.8
SGD Test MSE: [0.60092777], Accuracy: 0.8
Adam Test MSE: [0.63665208], Accuracy: 0.8
Mini-Batch Adam Test MSE: [0.58851062], Accuracy: 0.8
AdaGradTest MSE: [0.59098942], Accuracy: 0.81
RMSPROP Test MSE: [0.61830672], Accuracy: 0.81

Rastgele küçük weightler ile deneme: 4

GD Test MSE: [0.56910002], Accuracy: 0.8
SGD Test MSE: [0.61527072], Accuracy: 0.8
Adam Test MSE: [0.60757772], Accuracy: 0.8
Mini-Batch Adam Test MSE: [0.61108845], Accuracy: 0.8
AdaGradTest MSE: [0.59098942], Accuracy: 0.81
RMSPROP Test MSE: [0.61830672], Accuracy: 0.81

Rastgele küçük weightler ile deneme: 5

GD Test MSE: [0.56910002], Accuracy: 0.8
SGD Test MSE: [0.62238997], Accuracy: 0.8
Adam Test MSE: [0.61498688], Accuracy: 0.8
Mini-Batch Adam Test MSE: [0.58159996], Accuracy: 0.8
AdaGradTest MSE: [0.59098942], Accuracy: 0.81
RMSPROP Test MSE: [0.61830672], Accuracy: 0.81
```

"Alakalı Yanlış Cevap" senaryosunu içeren veri seti üzerinde yapılan testlerde , tüm optimizasyon algoritmalarının benzer şekilde %80 doğruluk ve 0.57-0.62 MSE bandında takılı kaldığı gözlemlenmiştir. Bu deneylerde algoritmalar arası performans farkı minimize olmuş, Gradient Descent veya gelişmiş yöntemler (Adam, RMSProp) sonucu anlamlı ölçüde değiştirememiştir.

Bu durumun temel nedeni, veri setindeki yanlış cevapların konu ve içerik bakımından doğru cevaba anlamsal olarak çok yakın olmasıdır. Vektör uzayında birbirine çok yakın konunan bu örnekler, modelin karar sınırını net bir şekilde çizmesini engellemiştir ve öğrenme sürecinin erken bir aşamada tıkanmasına (underfitting) yol açmıştır. Problemin optimizasyon tekniğinden ziyade veri setinin ayırtılabilirliği ile ilgili olması nedeniyle, algoritmaların yakınsama davranışlarını ve performans farklarını daha net kıyaslayabilmek adına bu veri seti projenin ana eğitim sürecinde tercih etmedim.

### 3- Detaylı Doğru ve Yanlış Cevap: (En Az 8 kelime cevap uzunluğu.)

```
Rastgele küçük weightler ile deneme: 1  
  
GD Test MSE: [0.13620004], Accuracy: 0.96  
SGD Test MSE: [0.1780175], Accuracy: 0.96  
Adam Test MSE: [0.13904938], Accuracy: 0.96  
Mini-Batch Adam Test MSE: [0.12462104], Accuracy: 0.96  
AdaGradTest MSE: [0.13070992], Accuracy: 0.96  
RMSProp Test MSE: [0.13734147], Accuracy: 0.96  
  
Rastgele küçük weightler ile deneme: 2  
  
GD Test MSE: [0.13620004], Accuracy: 0.96  
SGD Test MSE: [0.18347985], Accuracy: 0.96  
Adam Test MSE: [0.14519506], Accuracy: 0.96  
Mini-Batch Adam Test MSE: [0.12486963], Accuracy: 0.96  
AdaGradTest MSE: [0.13070992], Accuracy: 0.96  
RMSProp Test MSE: [0.13734147], Accuracy: 0.96  
  
Rastgele küçük weightler ile deneme: 3  
  
GD Test MSE: [0.13620004], Accuracy: 0.96  
SGD Test MSE: [0.18554411], Accuracy: 0.96  
Adam Test MSE: [0.15879359], Accuracy: 0.96  
Mini-Batch Adam Test MSE: [0.12716698], Accuracy: 0.96  
AdaGradTest MSE: [0.13070992], Accuracy: 0.96  
RMSProp Test MSE: [0.13734147], Accuracy: 0.96  
  
Rastgele küçük weightler ile deneme: 4  
  
GD Test MSE: [0.13620004], Accuracy: 0.96  
SGD Test MSE: [0.22089405], Accuracy: 0.96  
Adam Test MSE: [0.15944358], Accuracy: 0.96  
Mini-Batch Adam Test MSE: [0.12712267], Accuracy: 0.96  
AdaGradTest MSE: [0.13070992], Accuracy: 0.96  
RMSProp Test MSE: [0.13734147], Accuracy: 0.96  
  
Rastgele küçük weightler ile deneme: 5  
  
GD Test MSE: [0.13620004], Accuracy: 0.96  
SGD Test MSE: [0.18550736], Accuracy: 0.96  
Adam Test MSE: [0.14864838], Accuracy: 0.96  
Mini-Batch Adam Test MSE: [0.12661205], Accuracy: 0.96  
AdaGradTest MSE: [0.13070992], Accuracy: 0.96  
RMSProp Test MSE: [0.13734147], Accuracy: 0.96
```

Vektörleştirme aşamasında metin uzunluğunun ayırt ediciliğe etkisini gözlemlemek adına, "Uzun Metinli Alakasız Yanlış Cevap" veri setini hazırladım. Temel bekłentim, kelime sayısını artırarak vektörlerin uzayda daha keskin konumlara yerleşmesini sağlamak ve böylece modelin karar sınırlarını netleştirmekti.

Elde ettiğim sonuçları incelediğimde, Mini-Batch Adam algoritmasının 0.124 MSE deðeriyle hatayı en iyi minimize eden yöntem olduğunu, buna karşın SGD'nin tekil örnek varyansı sebebiyle 0.22 seviyelerine kadar çıkan hatasıyla en gürültülü sonucu verdığını gördüm. GD, AdaGrad ve RMSProp ise 0.13 bandında oldukça stabil bir çizgi izleyerek gürültüden etkilenmediklerini kanıtladılar.

Ancak, tüm algoritmaların %96 doğruluk oranında doyuma (saturation) ulaşması, bu veri setinin kısa cevaplı versiyona kıyasla dramatik bir iyileştirme sağlamadığını ortaya koydu. Metinleri uzatmanın getirdiği işlem yüküne karşılık model başarısında sadece marjinal bir artış gözlemediğim için, projenin nihai eğitim veri kümesi olarak tercih etmedim.

#### 4- Büyük Veri Seti (250 eğitim 50 test):

```
Rastgele küçük weightler ile deneme: 1

GD Test MSE: [0.10173956], Accuracy: 0.96
SGD Test MSE: [0.30647595], Accuracy: 0.96
Adam Test MSE: [0.25335464], Accuracy: 0.96
Mini-Batch Adam Test MSE: [0.11478518], Accuracy: 0.96
AdaGradTest MSE: [0.14008217], Accuracy: 0.95
RMSPROP Test MSE: [0.08815953], Accuracy: 0.97

Rastgele küçük weightler ile deneme: 2

GD Test MSE: [0.10173956], Accuracy: 0.96
SGD Test MSE: [0.21358381], Accuracy: 0.96
Adam Test MSE: [0.37512698], Accuracy: 0.96
Mini-Batch Adam Test MSE: [0.14464214], Accuracy: 0.96
AdaGradTest MSE: [0.14008217], Accuracy: 0.95
RMSPROP Test MSE: [0.08815953], Accuracy: 0.97

Rastgele küçük weightler ile deneme: 3

GD Test MSE: [0.10173956], Accuracy: 0.96
SGD Test MSE: [0.28158596], Accuracy: 0.96
Adam Test MSE: [0.26080319], Accuracy: 0.96
Mini-Batch Adam Test MSE: [0.18065443], Accuracy: 0.96
AdaGradTest MSE: [0.14008217], Accuracy: 0.95
RMSPROP Test MSE: [0.08815953], Accuracy: 0.97

Rastgele küçük weightler ile deneme: 4

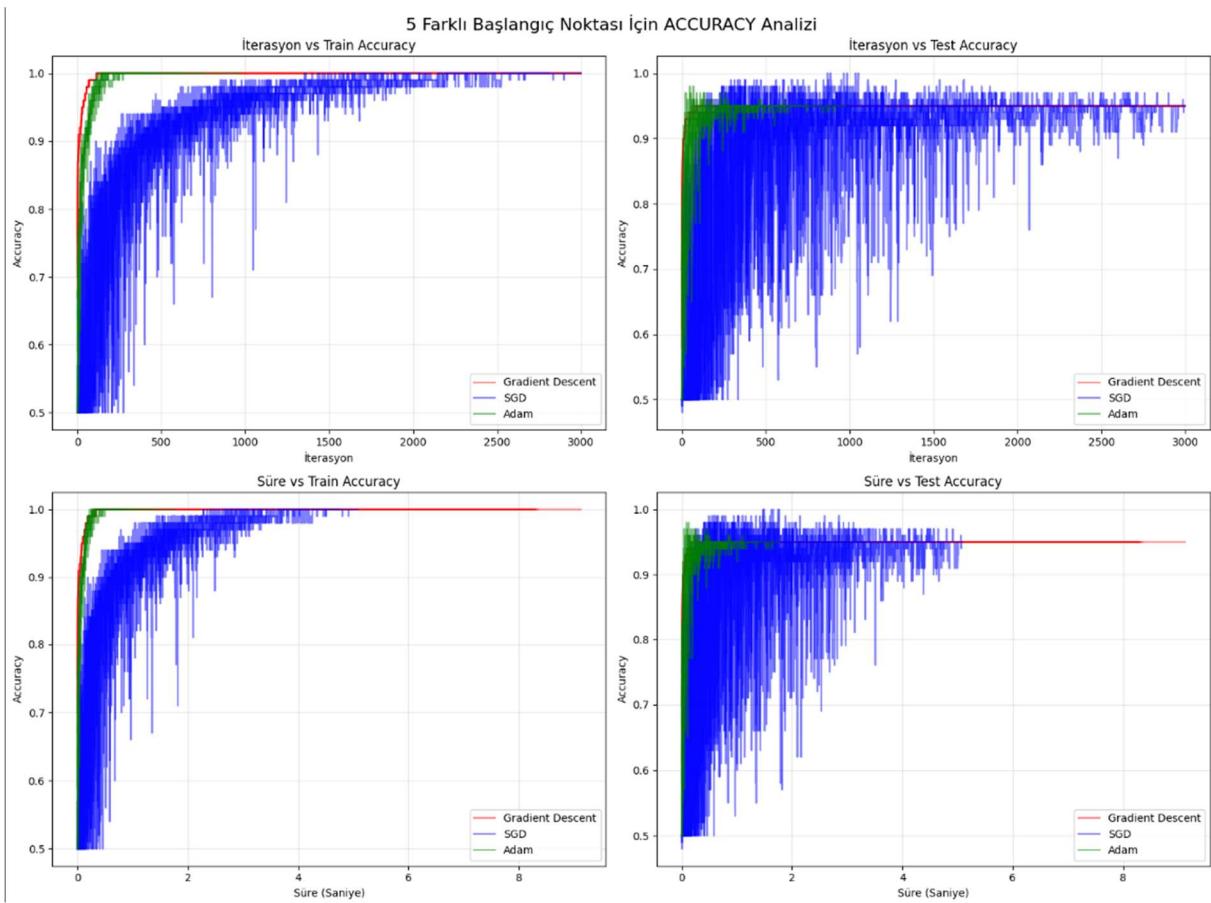
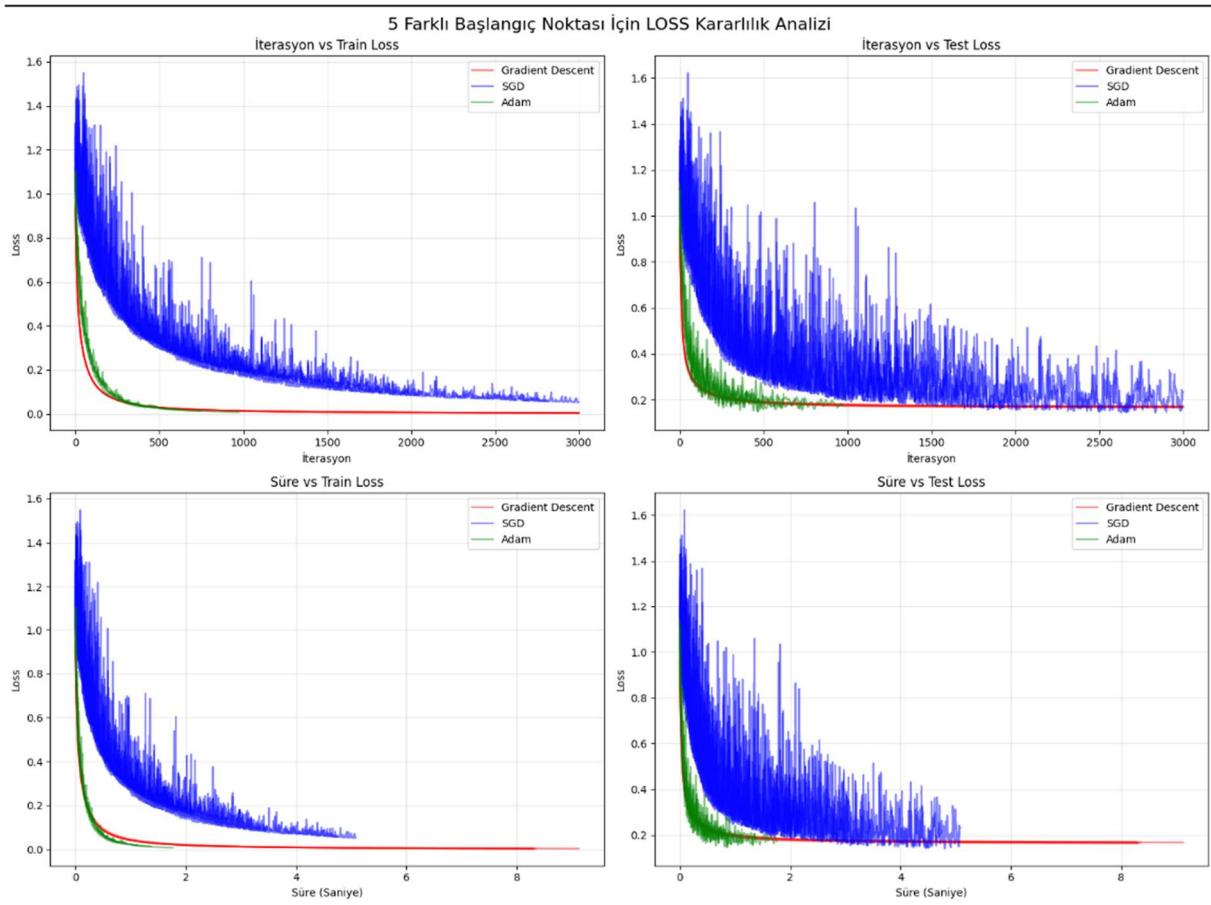
GD Test MSE: [0.10173956], Accuracy: 0.96
SGD Test MSE: [0.24591043], Accuracy: 0.96
Adam Test MSE: [0.27481821], Accuracy: 0.96
Mini-Batch Adam Test MSE: [0.11423963], Accuracy: 0.96
AdaGradTest MSE: [0.14008217], Accuracy: 0.95
RMSPROP Test MSE: [0.08815953], Accuracy: 0.97

Rastgele küçük weightler ile deneme: 5

GD Test MSE: [0.10173956], Accuracy: 0.96
SGD Test MSE: [0.2861626], Accuracy: 0.96
Adam Test MSE: [0.19177617], Accuracy: 0.96
Mini-Batch Adam Test MSE: [0.14990932], Accuracy: 0.96
AdaGradTest MSE: [0.14008217], Accuracy: 0.95
RMSPROP Test MSE: [0.08815953], Accuracy: 0.97
```

Modelin genelleştirme kapasitesini ölçmek ve sonuçların istatistiksel güvenilirliğini artırmak amacıyla, eğitim veri setindeki örnek sayısını 50'den 250'ye çıkararak deneyleri tekrarladım. Genişletilmiş veri setiyle elde edilen sonuçlar incelendiğinde, veri miktarındaki artışın özellikle RMSProp algoritması üzerinde çarpıcı bir pozitif etki yarattığı görülmüştür. RMSProp, bu veri setinde 0.088 gibi proje genelindeki en düşük MSE değerine ve %97 doğruluk oranına ulaşarak, veri yoğunluğu arttıkça optimizasyon kararlılığının da güçlendiğini kanıtlamıştır.

Gradient Descent (GD) 0.10 MSE ile istikrarlı yapısını korurken, tekil örnek üzerinden çalışan SGD ve Adam algoritmalarındaki gürültülü dalgalanmaların (MSE ~0.25-0.37) devam ettiği, ancak Mini-Batch yönteminin bu gürültüyü başarıyla sönmeyerek (MSE ~0.11) GD'ye yakın bir performans sergilediği gözlemlenmiştir. Artan veri hacminin modelin karar sınırlarını daha hassas öğrenmesini sağladığını ve elde edilen sonuçların 50 örneklik sete kıyasla çok daha kayda değer olduğunu değerlendirdiğimden, projenin devamında ve nihai modelin oluşturulmasında bu genişletilmiş veri setini (250 Eğitim - 50 Test) kullanmaya karar verdim.



3. Veri setinde elde edilen Loss ve Başarı grafikleri 5 farklı başlama noktası için algoritmaların yapılarının öğrenme sürecine etkisini net bir şekilde ortaya koymaktadır:

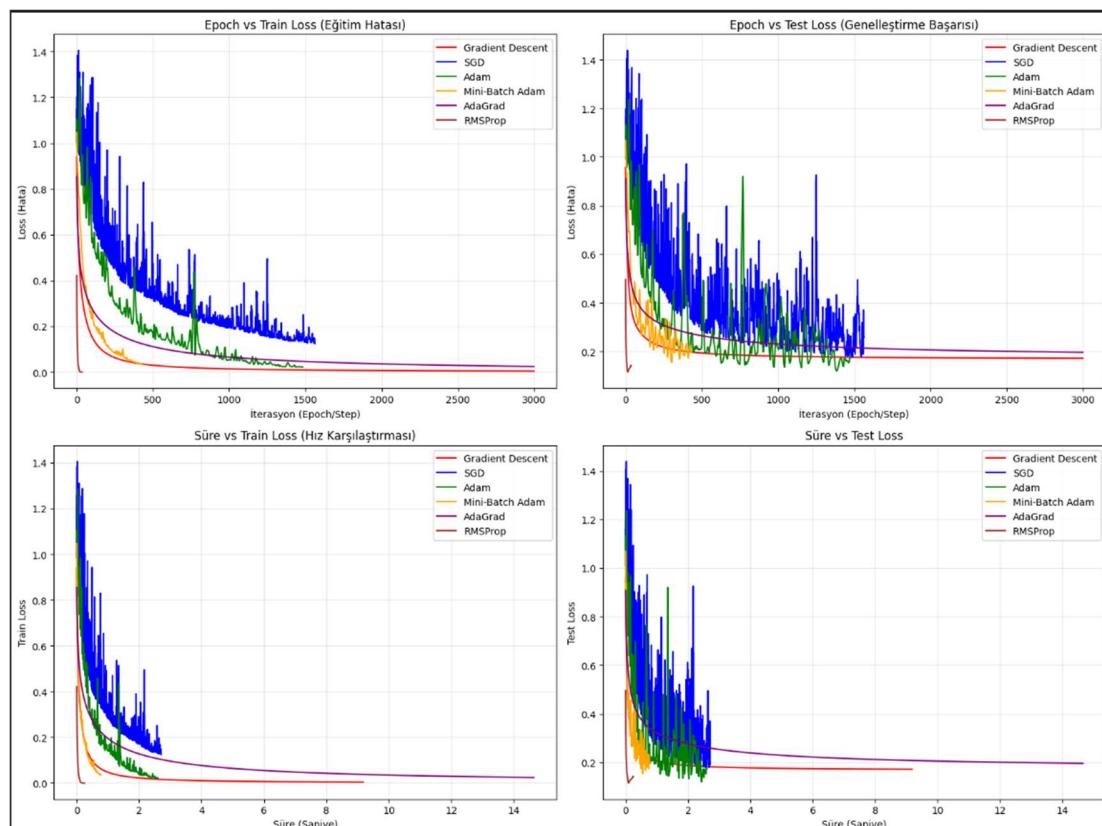
1. Gradient Descent (GD) - Kararlılık: Grafiklerde kırmızı çizgi ile gösterilen Gradient Descent, bekleniği üzere en pürüzsüz ve kararlı öğrenme eğrisini sergilemiştir. Her adımda tüm veri setinin gradyan ortalamasını alması sebebiyle, ağırlık güncellemeleri gürültüden arınmıştır. Loss değerindeki düşüş monotonik ve istikrarlıdır; herhangi bir ani sıçrama veya dalgalanma gözlemlenmemiştir. Bu durum, GD'nin güvenilirliğini kanıtlasa da, lokal minimumlara takılma riskini de beraberinde getirmektedir.

2. Stochastic Gradient Descent (SGD) - Yüksek Varyans ve "Şans" Faktörü: Mavi çizgi ile temsil edilen SGD, grafiğin en dikkat çekici ve gürültülü (dalgalı) davranışını sergilemektedir. Algoritmanın her iterasyonda rastgele tek bir örneğe göre yön değiştirmesi, Loss grafiğinde kalın bir "zizkak" örüntüsü oluşturmuştur.

Özellikle Başarı (Accuracy) grafiği incelendiğinde, SGD'nin bazı iterasyonlarda anlık olarak %100 (1.0) başarıya ulaştığı, ancak hemen ardından tekrar düştüğü görülmektedir. Bu durum, modelin o an gerçekten mükemmel öğrendiğini değil; rastgele seçilen örneğin ağırlıkları şans eseri doğru konuma savurduğunu göstermektedir. SGD'nin bu yüksek varyanslı yapısı, modelin global minimum etrafında sürekli salınım yapmasına (osilasyon) ve GD kadar kararlı bir duruma yerleşememesine neden olmaktadır.

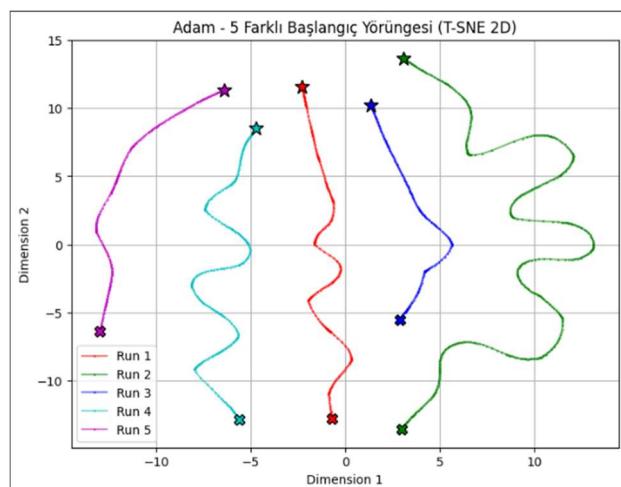
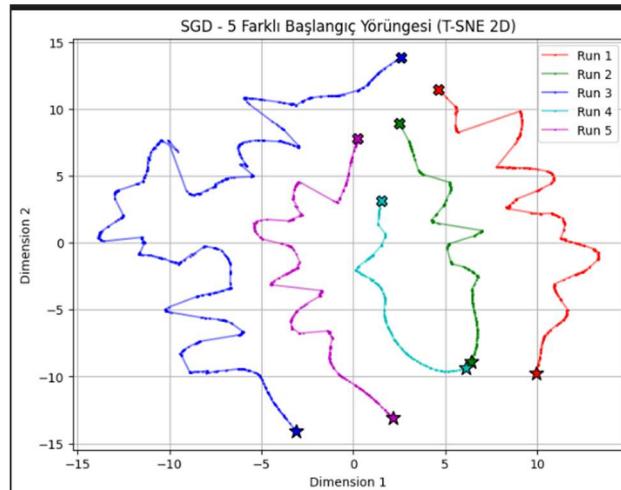
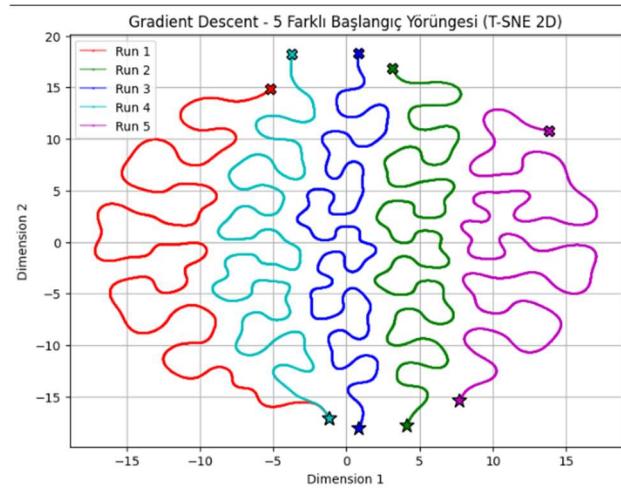
3. Batched-Adam: Yeşil çizgi ile gösterilen Adam algoritması, Momentum mekanizması sayesinde SGD'ye kıyasla gürültüyü kısmen sönmülemeyi başarmış olsa da, GD kadar pürüzsüz bir çizgi izlememiştir. Başlangıçtaki hızlı düşüşü, adaptif öğrenme hızının avantajını gösterse de küçük batch kullanımından kaynaklı dalgalanmalar devam etmiştir.

Sonuç olarak grafikler; Gradient Descent'in en kararlı ve güvenilir yakınsamayı sağladığını, SGD'nin ise hızlı hareket etmesine rağmen ağırlık uzayında çok fazla savrulduğunu ve başarısının yüksek standart sapma içерdiğini doğrulamaktadır.



3. Veri setindeki bu karşılaştırma 6 optimizasyon algoritması da bulunmaktadır. Önceki grafiklerden farklı olarak bunda ADAM, AdaGrad ve RMSProp da bulunmaktadır. GD ve AdaGrad yeterince yakınsamadığını düşündüğü için sınır olan 3000.epoch a kadar parametreleri güncelliyor. Diğerleri ise yakınsayarak eğitimden çıkıyor. Grafiklerden de gözüktüğü üzere en iyi performansı RMSProp'ta elde ettim. Hem kısa sürede yakınsıyor hem de MSE, doğruluk gibi parametreleri diğer algoritmalarla göre çok iyi.

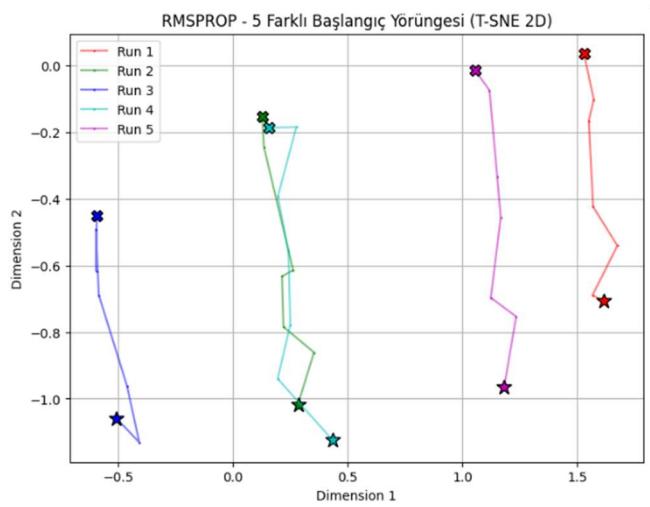
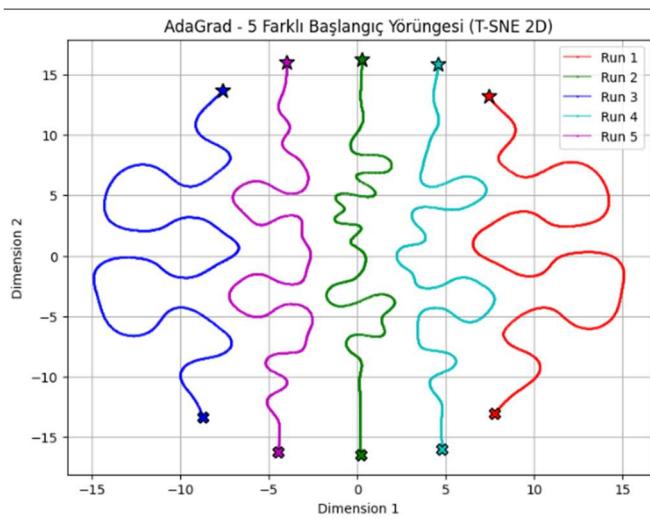
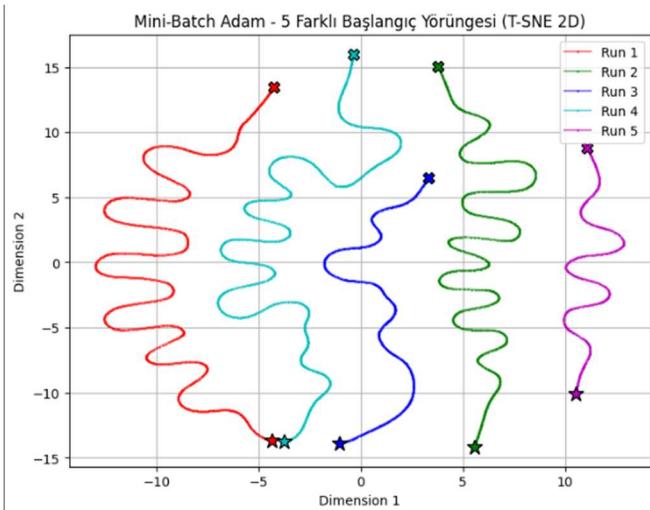
## T-SNE GRAFİKLERİ KARŞILAŞTIRMASI



Her adımda veri setinin tamamını kullandığı için güncellemleri deterministik ve gürültüsüzdür. Grafikteki pürüzsüz ancak kıvrımlı yapı, algoritmanın gürültüden değil, kayıp yüzeyinin kendi doğal eğiminden kaynaklanan "gerçek" vadileri takip ettiğini gösterir. Yörüngenin temiz olması, algoritmanın her adımda matematiksel olarak en dik iniş yönünü tam bir kesinlikle hesapladığının kanıdır.

SGD, momentum içermediği ve her adımda verinin sadece bir örneğe baktığı için gürültülü güncellemler yapar. Bu durum, parametre uzayında "sarhoş yürüyüşü"ne neden olur. Grafikteki keskin dönüşüler ve düzensizlikler, algoritmanın anlık veri gürültüsüne tepki verisini, ancak yine de genel bir yöne (minimuma) gitmeye çalıştığını gösterir.

Tek örnek kullanmasına (yani gürültülü olmasına) rağmen SGD'den çok daha pürüzsüz ve uzun yörüngeler çizer. Bunun nedeni içeridiği Momentum mekanizmasıdır. Momentum, tek örnekten gelen anlık gürültüyü söñümler (yumuşatır) ve geçmiş adımların hızını korur. Grafikteki kıvrımlı ancak akıcı yapı, algoritmanın gürültüye rağmen hız kazanabildiğini ve SGD'ye kıyasla yerel minimumlardan daha kolay sıyrılarak daha geniş bir alanı taradığını gösterir.



Hem momentumu hem de mini-batch (çoklu örnek) yapısını kullandığı için hem hızlı hem de nispeten daha kararlıdır. Grafikteki belirgin, ritmik ve "yılanvari" (kırımlı) salınımlar, momentumun kazandırdığı hız ile mini-batch varyansının dengesini temsil eder. Bu yoğun kıvrımlar bir hata değil; algoritmanın loss yüzeyindeki vadilerin şekline momentum sayesinde mükemmel uyum sağladığını ve en derin noktaya (düşük MSE) ulaşmak için yüzeyin geometrisini agresif bir şekilde kullandığını gösterir.

Geçmiş gradyanların karesini kümülatif olarak toplayarak öğrenme hızını sürekli düşürür. Grafikteki yörüngelerin birbirinden çok net ayırmış, simetrik ve "yaprak" benzeri düzenli yapısı, algoritmanın adımlarını giderek küçültüğünü (decay) ve çok temkinli ilerlediğini gösterir. Bu yapı, algoritmanın gürültüye kapılmadan, belirlediği yolda inatçı ve kararlı bir şekilde (ancak yavaşlayarak) ilerlediğinin görsel kanıdır.

Gradyanların karesinin hareketli ortalamasını alarak öğrenme hızını adapte eder. Grafikteki yörüngelerin diğerlerine göre daha dik veya ani yön değişimleri içermesi (bazı koşularda düzensiz sıçramalar), algoritmanın yüzeydeki eğim değişikliklerine çok hızlı adapte olmasından kaynaklanır. t-SNE uzayında görülen bu karakteristik, algoritmanın sık sık değişen eğimlere karşı agresif bir frenleme veya hızlanması mekanizmasını uyguladığını işaret eder.

## BAŞKA ANLAMSAL MODELLERİN KARŞILAŞTIRILMASI

### 1-BGE-M3 Modeli:

```
Rastgele küçük weightler ile deneme: 1  
  
GD Test MSE: [0.16873069], Accuracy: 0.99  
SGD Test MSE: [0.19757123], Accuracy: 0.99  
Adam Test MSE: [0.19610263], Accuracy: 0.99  
Mini-Batch Adam Test MSE: [0.15676175], Accuracy: 0.99  
AdaGradTest MSE: [0.13936991], Accuracy: 0.97  
RMSPROP Test MSE: [0.20516874], Accuracy: 0.93  
  
Rastgele küçük weightler ile deneme: 2  
  
GD Test MSE: [0.16873069], Accuracy: 0.99  
SGD Test MSE: [0.28947842], Accuracy: 0.99  
Adam Test MSE: [0.17305852], Accuracy: 0.99  
Mini-Batch Adam Test MSE: [0.14283684], Accuracy: 0.99  
AdaGradTest MSE: [0.13936991], Accuracy: 0.97  
RMSPROP Test MSE: [0.20516874], Accuracy: 0.93  
  
Rastgele küçük weightler ile deneme: 3  
  
GD Test MSE: [0.16873069], Accuracy: 0.99  
SGD Test MSE: [0.18463117], Accuracy: 0.99  
Adam Test MSE: [0.27948318], Accuracy: 0.99  
Mini-Batch Adam Test MSE: [0.13182662], Accuracy: 0.99  
AdaGradTest MSE: [0.13936991], Accuracy: 0.97  
RMSPROP Test MSE: [0.20516874], Accuracy: 0.93  
  
Rastgele küçük weightler ile deneme: 4  
  
GD Test MSE: [0.16873069], Accuracy: 0.99  
SGD Test MSE: [0.22667682], Accuracy: 0.99  
Adam Test MSE: [0.2237062], Accuracy: 0.99  
Mini-Batch Adam Test MSE: [0.14121362], Accuracy: 0.99  
AdaGradTest MSE: [0.13936991], Accuracy: 0.97  
RMSPROP Test MSE: [0.20516874], Accuracy: 0.93  
  
Rastgele küçük weightler ile deneme: 5  
  
GD Test MSE: [0.16873069], Accuracy: 0.99  
SGD Test MSE: [0.18308921], Accuracy: 0.99  
Adam Test MSE: [0.19221907], Accuracy: 0.99  
Mini-Batch Adam Test MSE: [0.16644418], Accuracy: 0.99  
AdaGradTest MSE: [0.13936991], Accuracy: 0.97  
RMSPROP Test MSE: [0.20516874], Accuracy: 0.93
```

BGE-M3 modeli ile oluşturulan veri seti, vektör uzayında mükemmel bir lineer ayırtılabilirlik sunmuştur. Bu sayede Gradient Descent, SGD ve Adam algoritmalarının tamamı %99 doğruluk oranına ulaşarak, vektörler arasındaki sınırları kolayca tespit edebilmiştir.

Ancak önceki veri setlerinde en iyi performansı gösteren RMSProp, bu modelde %93 doğrulukla en zayıf sonucu vermiştir. Bu durumun iki temel nedeni olabilir:

1-Adım Büyüklüğü Uyumsuzluğu: BGE-M3'ün düşük varyanslı gradyanları, RMSProp'un paydasındaki  $v_t$  değerini küçülterek adım büyüğünü orantısız şekilde artırmış; bu da algoritmanın minimum noktayı sürekli pas geçmesine (overshooting) neden olmuş olabilir.

2-Momentum Eksikliği: Mevcut öğrenme hızı ( $rate=0.01$ ) bu veri dağılımı için fazla agresif kalmış olabilir. Adam algoritması momentum mekanizmasıyla bu hızı dengelerken, momentumlu RMSProp savrulmaya devam etmiştir.

Sonuç: Bu deney, "evrensel en iyi algoritma" olmadığını; başarının veri dağılımı ile algoritma karakteristiğinin uyumuna bağlı olduğunu göstermiştir. Zorlu setlerde RMSProp'un adaptif gücü öne çıkarken, iyi ayırtılmış setlerde GD ve Adam'ın kararlılığı daha üstün sonuç vermiştir.

## 2- Multilingual E5 Large Modeli

```
Rastgele küçük weightler ile deneme: 1

GD Test MSE: [0.16839887], Accuracy: 0.99
SGD Test MSE: [0.21902677], Accuracy: 0.99
Adam Test MSE: [0.15213577], Accuracy: 0.99
Mini-Batch Adam Test MSE: [0.15149874], Accuracy: 0.99
AdaGradTest MSE: [0.13911998], Accuracy: 0.97
RMSPROP Test MSE: [0.20557217], Accuracy: 0.93

Rastgele küçük weightler ile deneme: 2

GD Test MSE: [0.16839887], Accuracy: 0.99
SGD Test MSE: [0.24167482], Accuracy: 0.99
Adam Test MSE: [0.17213234], Accuracy: 0.99
Mini-Batch Adam Test MSE: [0.15696994], Accuracy: 0.99
AdaGradTest MSE: [0.13911998], Accuracy: 0.97
RMSPROP Test MSE: [0.20557217], Accuracy: 0.93

Rastgele küçük weightler ile deneme: 3

GD Test MSE: [0.16839887], Accuracy: 0.99
SGD Test MSE: [0.18000354], Accuracy: 0.99
Adam Test MSE: [0.15295799], Accuracy: 0.99
Mini-Batch Adam Test MSE: [0.1285615], Accuracy: 0.99
AdaGradTest MSE: [0.13911998], Accuracy: 0.97
RMSPROP Test MSE: [0.20557217], Accuracy: 0.93

Rastgele küçük weightler ile deneme: 4

GD Test MSE: [0.16839887], Accuracy: 0.99
SGD Test MSE: [0.20243839], Accuracy: 0.99
Adam Test MSE: [0.17845533], Accuracy: 0.99
Mini-Batch Adam Test MSE: [0.17363498], Accuracy: 0.99
AdaGradTest MSE: [0.13911998], Accuracy: 0.97
RMSPROP Test MSE: [0.20557217], Accuracy: 0.93

Rastgele küçük weightler ile deneme: 5

GD Test MSE: [0.16839887], Accuracy: 0.99
SGD Test MSE: [0.20748825], Accuracy: 0.99
Adam Test MSE: [0.17380562], Accuracy: 0.99
Mini-Batch Adam Test MSE: [0.15016409], Accuracy: 0.99
AdaGradTest MSE: [0.13911998], Accuracy: 0.97
RMSPROP Test MSE: [0.20557217], Accuracy: 0.93
```

Alternatif bir gelişmiş vektörleştirme modeli olan Multilingual E5 Large ile yapılan testler, BGE-M3 ile elde edilen bulgularla büyük ölçüde paralellik göstermektedir . Modelin sunduğu yüksek lineer ayırtırılabilirlik sayesinde Gradient Descent, SGD ve Adam algoritmaları %99 doğruluk oranına ulaşarak neredeyse kusursuz bir performans sergilemiştir. RMSProp algoritması ise bu vektör uzayında da benzer bir karakteristikle %93 seviyesinde kalarak, daha önce analiz edilen gradyan ölçüği hassasiyetini bir kez daha doğrulamıştır. Sonuçlar, her iki gelişmiş embedding modelinin de problemi algoritmalar için oldukça çözülebilir hale getirdiğini kanıtlamaktadır.

## FARKLI AKTİVASYON FONKSİYONLARININ TEST EDİLMESİ

1- Sigmoid:

```
Rastgele küçük weightler ile deneme: 1  
  
GD Test MSE: [0.34514501], Accuracy: 0.92  
SGD Test MSE: [0.59353552], Accuracy: 0.92  
Adam Test MSE: [0.5105181], Accuracy: 0.92  
Mini-Batch Adam Test MSE: [0.49779114], Accuracy: 0.92  
AdaGradTest MSE: [0.49508236], Accuracy: 0.69  
RMSPROP Test MSE: [0.48920098], Accuracy: 0.76  
  
Rastgele küçük weightler ile deneme: 2  
  
GD Test MSE: [0.34514501], Accuracy: 0.92  
SGD Test MSE: [0.51991299], Accuracy: 0.92  
Adam Test MSE: [0.45858649], Accuracy: 0.92  
Mini-Batch Adam Test MSE: [0.4781314], Accuracy: 0.92  
AdaGradTest MSE: [0.49508236], Accuracy: 0.69  
RMSPROP Test MSE: [0.48920098], Accuracy: 0.76  
  
Rastgele küçük weightler ile deneme: 3  
  
GD Test MSE: [0.34514501], Accuracy: 0.92  
SGD Test MSE: [0.581479], Accuracy: 0.92  
Adam Test MSE: [0.52718032], Accuracy: 0.92  
Mini-Batch Adam Test MSE: [0.5438569], Accuracy: 0.92  
AdaGradTest MSE: [0.49508236], Accuracy: 0.69  
RMSPROP Test MSE: [0.48920098], Accuracy: 0.76  
  
Rastgele küçük weightler ile deneme: 4  
  
GD Test MSE: [0.34514501], Accuracy: 0.92  
SGD Test MSE: [0.51751562], Accuracy: 0.92  
Adam Test MSE: [0.47902927], Accuracy: 0.92  
Mini-Batch Adam Test MSE: [0.5021936], Accuracy: 0.92  
AdaGradTest MSE: [0.49508236], Accuracy: 0.69  
RMSPROP Test MSE: [0.48920098], Accuracy: 0.76  
  
Rastgele küçük weightler ile deneme: 5  
  
GD Test MSE: [0.34514501], Accuracy: 0.92  
SGD Test MSE: [0.53786069], Accuracy: 0.92  
Adam Test MSE: [0.49749001], Accuracy: 0.92  
Mini-Batch Adam Test MSE: [0.50160097], Accuracy: 0.92  
AdaGradTest MSE: [0.49508236], Accuracy: 0.69  
RMSPROP Test MSE: [0.48920098], Accuracy: 0.76
```

Modelin çıktılarını  $[0, 1]$  aralığına sıkıştırarak olasılıksal bir yaklaşım sunan Sigmoid Aktivasyon Fonksiyonu ile yapılan testler, Tanh'a kıyasla başarısı düşüktür. RMSProp algoritmasında ise gradyan hassasiyeti probleminden dolayı tekrar düşük çıkmıştır. AdaGrad'da da benzer sorun görülmektedir. Onun dışındaki geleneksel algoritmalarla ise accuracy metriği diğer algoritmala göre kötü fakat hala kabul edilebilir seviyededir.

## 2-RELU

```
Rastgele küçük weightler ile deneme: 1

GD Test MSE: [0.06457413], Accuracy: 0.94
SGD Test MSE: [0.11716006], Accuracy: 0.94
Adam Test MSE: [0.4652679], Accuracy: 0.94
Mini-Batch Adam Test MSE: [0.41184564], Accuracy: 0.94
AdaGradTest MSE: [0.44034743], Accuracy: 0.9
RMSProp Test MSE: [5.17491649], Accuracy: 0.74

Rastgele küçük weightler ile deneme: 2

GD Test MSE: [0.06457413], Accuracy: 0.94
SGD Test MSE: [0.06161936], Accuracy: 0.94
Adam Test MSE: [0.26120025], Accuracy: 0.94
Mini-Batch Adam Test MSE: [0.28747811], Accuracy: 0.94
AdaGradTest MSE: [0.44034743], Accuracy: 0.9
RMSProp Test MSE: [5.17491649], Accuracy: 0.74

Rastgele küçük weightler ile deneme: 3

GD Test MSE: [0.06457413], Accuracy: 0.94
SGD Test MSE: [0.05694124], Accuracy: 0.94
Adam Test MSE: [0.63305727], Accuracy: 0.94
Mini-Batch Adam Test MSE: [0.16283304], Accuracy: 0.94
AdaGradTest MSE: [0.44034743], Accuracy: 0.9
RMSProp Test MSE: [5.17491649], Accuracy: 0.74

Rastgele küçük weightler ile deneme: 4

GD Test MSE: [0.06457413], Accuracy: 0.94
SGD Test MSE: [0.05888559], Accuracy: 0.94
Adam Test MSE: [0.12207165], Accuracy: 0.94
Mini-Batch Adam Test MSE: [0.08864288], Accuracy: 0.94
AdaGradTest MSE: [0.44034743], Accuracy: 0.9
RMSProp Test MSE: [5.17491649], Accuracy: 0.74

Rastgele küçük weightler ile deneme: 5

GD Test MSE: [0.06457413], Accuracy: 0.94
SGD Test MSE: [0.11829943], Accuracy: 0.94
Adam Test MSE: [0.46626818], Accuracy: 0.94
Mini-Batch Adam Test MSE: [0.24660904], Accuracy: 0.94
AdaGradTest MSE: [0.44034743], Accuracy: 0.9
RMSProp Test MSE: [5.17491649], Accuracy: 0.74
```

Negatif değerleri sıfırlayarak hesaplama maliyetini düşüren ReLU Aktivasyon Fonksiyonu ile yapılan testlerde, genel başarı oranı oldukça yüksek ve tatmin edici seviyelerdedir. Ancak RMSProp algoritması, bu fonksiyon altında da gradyan kararsızlığı yaşayarak %74 doğrulukta kalmış ve oldukça yüksek bir hata oranı (MSE) üretmiştir. AdaGrad %90 seviyesinde kalarak lider grubun bir adım gerisinde dursa da; GD, SGD ve Adam gibi diğer tüm algoritmalarla %94 doğruluk oranında birleşilerek istikrarlı ve başarılı bir sonuç elde edilmiştir.

## İKİ KATMANLI MLP

İlk katman RELU ikinci katman Tanh olacak şekilde iki katmanlı mlp modeli kurdum. Model, Girdi Katmanı, 10 nöronlu tek bir Gizli Katman (Hidden Layer) ve çıkış Katmanından oluşur.

```
elif secim == 3: # MLP (2 Katmanlı)
    W1 = weights[0]
    W2 = weights[1]

    z1 = W1 @ x_train[j]
    sonuc1 = relu(z1)
    z2 = W2 @ sonuc1
    sonuc2 = np.tanh(z2)

    # Backward
    error = sonuc2 - y_train[j]
    d_output = 2 * error * (1 - sonuc2**2)

    grad_W2 = d_output * sonuc1
    d_hidden = (W2.flatten() * d_output) * relu_turev(z1)
    grad_W1 = np.outer(d_hidden, x_train[j])

    toplam_grad_W1 += grad_W1
    toplam_grad_W2 += grad_W2

if secim == 3:
    weights[0] = weights[0] - rate * (toplam_grad_W1 / N)
    weights[1] = weights[1] - rate * (toplam_grad_W2 / N)
```

Bu modeli yalnızca GD algoritması ile deneme şansı buldum. Daha fazla zamanım olsaydı farklı aktivasyon fonksiyonları, farklı optimizasyon algoritmaları ile sonuçları test etmek isterdim. Şu anda elde ettiğim sonuç Test MSE: [0.1686702] seviyesinde.

## KENDİ BONUSUM

Bu çalışmanın en özgün ve performans artırıcı adımlarından biri, soru ve cevap embedding'leri arasındaki anlamsal ilişkiyi modele örtük olarak bırakmak yerine, açık bir nitelik olarak sunmak olmuştur. Modern multimodal mimarilerin (örneğin CLIP) metin-görüntü hizalama başarısından ilham alarak, vektörler arasındaki yönsele uyumu temsil eden Cosine Similarity değerini hesapladım ve bunu modele 2050. boyut olarak entegre ettim. Vektörler halihazırda normalize edildiği için, bu işlem hesaplama maliyeti düşük olan Dot Product (Skaler Çarpım) ile gerçekleştirilmiştir.

```
if secim ==3:
    iyi_similarity = np.dot(soru, iyi)
    x_train[index] = np.concatenate([soru, iyi, [1.0],[iyi_similarity]])
else:
    x_train[index] = np.concatenate([soru, iyi, [1.0]])
```

```

if secim ==3:
    kotu_similarity = np.dot(soru,kotu)
    x_train[index] = np.concatenate([soru, kotu, [1.0],[kotu_similarity]])
else:
    x_train[index] = np.concatenate([soru, kotu, [1.0]])

```

Bu yaklaşım, yüksek boyutlu vektör uzayında modelin yönünü bulmasını sağlayan bir çipa görevi görmüştür. Model, salt embedding değerleri üzerinden ilişkiyi türetmeye çalışmak yerine, bu ek boyut sayesinde anlamsal yakınlığı doğrudan bir sinyal olarak algılamıştır. Sonuçlar, bu hibrit özellik çıkarma yönteminin optimizasyon yüzeyini pürüzsüzleştirdiğini, eğitimin stabilitesini dramatik ölçüde artırdığını ve tüm deneyler arasında en düşük hata oranını sağladığını kanıtlamaktadır. Bu bulgu, derin öğrenme modellerine geometrik ön bilgi sağlamaının performansı artırmadaki kritik rolünü ortaya koymaktadır.

```

Rastgele küçük weightler ile deneme: 1

GD Test MSE: [0.01977872], Accuracy: 1.0
SGD Test MSE: [0.04385753], Accuracy: 1.0
Adam Test MSE: [0.15273926], Accuracy: 1.0
Mini-Batch Adam Test MSE: [0.08087122], Accuracy: 1.0
AdaGradTest MSE: [0.08915316], Accuracy: 0.98
RMSPROP Test MSE: [0.03846456], Accuracy: 0.98

Rastgele küçük weightler ile deneme: 2

GD Test MSE: [0.01977872], Accuracy: 1.0
SGD Test MSE: [0.05269949], Accuracy: 1.0
Adam Test MSE: [0.11208964], Accuracy: 1.0
Mini-Batch Adam Test MSE: [0.06979032], Accuracy: 1.0
AdaGradTest MSE: [0.08915316], Accuracy: 0.98
RMSPROP Test MSE: [0.03846456], Accuracy: 0.98

Rastgele küçük weightler ile deneme: 3

GD Test MSE: [0.01977872], Accuracy: 1.0
SGD Test MSE: [0.04570317], Accuracy: 1.0
Adam Test MSE: [0.59295096], Accuracy: 1.0
Mini-Batch Adam Test MSE: [0.08898819], Accuracy: 1.0
AdaGradTest MSE: [0.08915316], Accuracy: 0.98
RMSPROP Test MSE: [0.03846456], Accuracy: 0.98

Rastgele küçük weightler ile deneme: 4

GD Test MSE: [0.01977872], Accuracy: 1.0
SGD Test MSE: [0.04083744], Accuracy: 1.0
Adam Test MSE: [0.17330278], Accuracy: 1.0
Mini-Batch Adam Test MSE: [0.09418251], Accuracy: 1.0
AdaGradTest MSE: [0.08915316], Accuracy: 0.98
RMSPROP Test MSE: [0.03846456], Accuracy: 0.98

Rastgele küçük weightler ile deneme: 5

GD Test MSE: [0.01977872], Accuracy: 1.0
SGD Test MSE: [0.03023466], Accuracy: 1.0
Adam Test MSE: [0.15318754], Accuracy: 1.0
Mini-Batch Adam Test MSE: [0.10572544], Accuracy: 1.0
AdaGradTest MSE: [0.08915316], Accuracy: 0.98
RMSPROP Test MSE: [0.03846456], Accuracy: 0.98

```

Proje Videosu: <https://youtu.be/DQ7TUDIvBu8>

Soruların, vektörlerin, kodun tamamının olduğu Github reposu:  
<https://github.com/fatiheren26/Diferansiyel-Denklemler-dev>