**BİLGİSAYAR MÜHENDİSLİĞİ İÇİN DİFERANSİYEL DENKLEMLER 2025/1 - ÖDEV 1 RAPORU**

**Ödev Konusu:** Regresyon Modeli Model: Giriş: Concat (Soru temsili, Cevap temsili Çıkış: cevap iyi ise +1, kötü ise -1  
**Ad Soyad:** Buse Selin Tavlak  
**Öğrenci No:** 23011079  
**Teslim Tarihi:** 5 Aralık 2025

**1. GİRİŞ VE AMAÇ**

Bu projenin amacı, Doğal Dil İşleme (NLP) yöntemleri ile elde edilen vektör uzaylarında, farklı optimizasyon algoritmalarının (Gradient Descent, SGD, Adam vb.) performanslarını ve davranışlarını incelemektir. Proje kapsamında, büyük dil modelleri (LLM) kullanılarak özgün bir veri seti oluşturulmuş, bu veriler vektörleştirilmiş ve hem tek katmanlı regresyon hem de çok katmanlı yapay sinir ağları (MLP) üzerinde eğitilmiştir. Modelin öğrenme süreci farklı başlangıç ağırlıkları (seed) ile test edilmiş ve elde edilen sonuçlar hem performans grafikleri hem de T-SNE yöntemi ile analiz edilmiştir. Proje 7 Python scripti ile parça parça oluşturulmuştur.

1. Veri\_uretici.py 5.egitim.py

2. Veri\_vektorlestir.py 6.main\_deney.py

3.modeller.py 7. analiz\_ve\_gorsellestirme.py)

4. Ozel\_optimizer.py

**TEKNİK ALTYAPI VE KULLANILAN TEKNOLOJİLER**

Aşağıdaki teknoloji yığını (tech stack) kullanılmıştır:

* **PyTorch (torch):** Hesaplama grafiği (computational graph) oluşturmak, tensör işlemleri ve otomatik türev alma (autograd) mekanizması için kullanıldı.
* **PyTorch Neural Networks (torch.nn):** Model mimarisini (Linear, ReLU, Tanh) ve hata fonksiyonlarını (MSELoss) modüler bir yapıda tanımlamak, katmanlar arası veri akışını yönetmek için kullanıldı.”Computational graph ” yapısı öğrenildi.
* **Data Utilities (torch.utils.data):**
  + **TensorDataset:** Eğitim ve test verilerini (X ve y tensörleri) tek bir obje içinde paketlemek için kullanıldı.
  + **DataLoader:** Veriyi karıştırmak (shuffling) ve belirlenen batch\_size boyutunda (SGD için 1, GD için tüm veri) modele beslemek, böylece bellek yönetimini optimize etmek için kritik bir rol oynadı.
* **NumPy (numpy):** T-SNE görselleştirmesi ve veri analizi aşamalarında, GPU üzerindeki PyTorch tensörlerini CPU tarafından işlenebilir düz matris formatına (flatten) dönüştürmek için kullanıldı. Ayrıca broadcasting ve dot product araştırıldı.(2048 + 1)d
* **Sentence-Transformers:** Embedding modellerini yönetmek ve metinleri vektör uzayına taşımak için kullanıldı.
* **OpenAI API (Local):** LM Studio üzerindeki yerel LLM ile iletişim kurmak için kullanıldı.
* **Scikit-Learn (sklearn):** Yüksek boyutlu veriyi görselleştirmek (T-SNE) için kullanıldı.
* **Matplotlib & Seaborn:** Eğitim kaybı (loss) ve optimizasyon yörüngelerini çizdirmek için kullanıldı. Seaborn kütüphanesi, grafiklerin estetik kalitesini artırmak ve veri dağılımlarını daha okunaklı hale getirmek amacıyla tercih edildi.

**2. METODOLOJİ**

**2.1. Veri Setinin Oluşturulması**

**Projenin temelini oluşturan soru-cevap veri seti, yerel ortamda çalışan Turkish-Gemma-9b-T1 modeli kullanılarak üretilmiştir.**

* **Yöntem: LM Studio aracılığıyla model yerel sunucu (localhost) olarak ayağa kaldırılmış ve Python openai kütüphanesi ile API istekleri gönderilmiştir.Bu noktada pythonda veri ürettirmek için bir dosya oluşturdum.**

**A computer screen shot of text

AI-generated content may be incorrect.**

* **Veri Yapısı: Toplam 10 farklı konu başlığı (Tarih, Coğraya, STEM, Mantık, Edebiyat, Gündelik Yaşam, Genel Kültür, Yazılım ve CS, Felsefe, Fizik(Kuantum) belirlenmiştir. Her başlık için modelden 1 soru, 1 doğru cevap ve 1 yanlış cevap üretmesi istenmiştir. İyi cevabın doğru olması, kötü cevabın yanlış veya bağlamdan farklı olması sağlanmıştır.**
* **Eğitim/Test Ayrımı: Veri sızıntısını (data leakage) ve overfittingi önlemek amacıyla, Eğitim setindeki alt konular (Örn: Osmanlı Yükselme) ile Test setindeki alt konular (Örn: Kurtuluş Savaşı) birbirinden tamamen ayrılmıştır. Toplam 100 örnek (50 Eğitim, 50 Test) oluşturulmuştur.**

**A screenshot of a computer screen

AI-generated content may be incorrect.**

**Veri Formatı (.JSON):** Üretilen ham veriler **JSON (JavaScript Object Notation)** formatında kaydedilmiştir.

* Neden JSON? İnsan tarafından okunabilir (human-readable) olması, hiyerarşik yapısı ve veri bütünlüğünü kontrol etmenin kolay olması nedeniyle tercih edilmiştir. Bu aşamada veriyi .pt yerine .json tutmak, üretilen metinlerin doğruluğunu gözle manuel kontrol etmemize olanak tanımıştır.

Aşağıda veri setinden bir parça bulunmaktadır:

**A screenshot of a computer program

AI-generated content may be incorrect.**

**2.2. Anlamsal Temsil (Embedding)**

**Metinsel verilerin matematiksel vektörlere dönüştürülmesi için ytu-ce-cosmos/turkish-e5-large modeli kullanılmıştır. Daha küçük bir model olduğu için LM studioda local şekilde açmadan pythonda veri\_vektorlestir.py dosyası oluşturup direkt SentenceTransformer ile modeli yükledim.** Batch processing yaparak veriyi tek tek vektörize etmek yerine toplu şekilde bir kez modeli çağırarak embedding yapmış oldum.

* **Mimari Karar: Neden LM Studio Yerine Doğrudan Python? Veri üretiminde (Adım 1) LM Studio kullanırken, Embedding aşamasında (Adım 2) modeli doğrudan Python kütüphanesi (sentence-transformers) ile yükledik.**
  + **Gerekçe: Embedding modelleri (Encoder-only), Generative LLM'lere (Decoder-only) göre çok daha hafiftir (~1-2 GB). Bu modelleri harici bir sunucu arayüzüyle (LM Studio) çalıştırmak yerine, doğrudan Python process'inin RAM'ine yüklemek, ağ gecikmesini (latency) ortadan kaldırmış ve vektörleştirme hızını 10x-20x artırmıştır.**
* **Instruction Tuning: E5 modelinin mimarisine uygun olarak, soruların başına "query: ", cevapların başına "passage: " ön ekleri kodla otomatik eklenmiştir. Bu işlem modelin dikkat mekanizmasının (attention) doğru çalışmasını sağlamıştır.**

A screen shot of a computer program

AI-generated content may be incorrect.

* **Vektör Birleştirme (Concatenation):**
  + **Soru Vektörü (d=1024d)**
  + **Cevap Vektörü (d=1024d)**
  + **Model Girişi (): Bu iki vektör uç uca eklenerek 2048 boyutlu tek bir vektör elde edilmiştir.**
* **Etiketleme: İyi cevaplar için +1, kötü cevaplar için -1 etiketi atanmıştır.**
* **Ağırlık başlatma (Initialization): Tanh fonksiyonunun doygunluğa ulaşmasını engellemek ve gradyan akışını sağlamak amacıyla ağırlıklar Xavier (Glorot) yöntemiyle veya daha düşük standart sapmalı normal dağılımla başlatılmıştır.**

**A screen shot of a computer program

AI-generated content may be incorrect.**

* **Veri Formatı (.PT): Vektörleştirilen veriler PyTorch Tensor formatında (.pt) kaydedilmiştir.**
  + ***Neden .PT?* JSON metin tabanlı olduğu için yüklenmesi (parsing) yavaştır. .pt formatı ise binary (ikili) formattır ve veriyi doğrudan belleğe (memory mapping) map eder. Eğitim sırasında diski yormadan GPU/CPU belleğine anında transfer sağladığı için bu format zorunlu hale gelmiştir.**

**A screen shot of a computer code

AI-generated content may be incorrect.**

**(Fotoğrafta gözüktüğü gibi dosyalar kontrol edilip .pt uzantısıyla kaydedilmiştir.)**

**3.3. Donanım Hızlandırması (GPU Etkisi)**

**Proje sürecinde matris çarpımları (W \* X) yoğun işlem gücü gerektirmiştir.**

* **CUDA Kullanımı: Embedding ve Eğitim süreçlerinde NVIDIA CUDA çekirdekleri kullanılarak işlemler paralelleştirilmiştir. 2048 boyutlu vektörlerin çarpımı, CPU'da seri işlem görürken, GPU'da binlerce çekirdekte aynı anda işlenmiş, bu da eğitim süresini dakikalardan saniyelere düşürmüştür.**

**A computer screen shot of a program code

AI-generated content may be incorrect.**

**4. MODEL VE OPTİMİZASYON**

**4.1 Model Mimarisi ve Matematiksel Altyapı (modeller.py)**

**Bu çalışmada, regresyon problemini çözmek için iki farklı sinir ağı mimarisi tasarlanmış ve Pytorch kütüphanesi kullanılarak implemente edilmiştir. Modellerin eğitimi GPU (CUDA) desteği varsa otomatik olarak GPU üzerinde, yoksa CPU üzerinde gerçekleştirilecek şekilde yapılandırılmıştır.**

* **Giriş: 2048 Boyut (1024 Soru + 1024 Cevap).**
* **Boyut Genişletme (Bias Trick): Ödev dokümanında belirtilen $w \in \mathbb{R}^{2d+1}$ koşulunu sağlamak amacıyla, 2048 boyutlu giriş vektörüne bias (sapma) terimi eklenmiştir. PyTorch'un nn.Linear katmanı kullanılarak bu işlem otomatikleştirilmiş ve modelin öğrenilebilir parametre uzayı 2049 boyutuna (2048 Ağırlık + 1 Bias) çıkarılmıştır.**

**Modellerin tasarımında Python Class (Sınıf) yapısı kullanılmasının temel nedeni, Modülarite, Kapsülleme (Encapsulation) ve Kalıtım (Inheritance) prensiplerinden faydalanmaktır.**

* **Modülarite ve Yönetilebilirlik: Model mimarisi (\_\_init\_\_) ve ileri yayılım mantığı (forward), tek bir sınıf çatısı altında toplanmıştır. Bu sayede, BasitModel ve BonusModelMLP sınıfları birbirinden bağımsız olarak geliştirilebilmiş, birinde yapılan değişiklik diğerini bozmamıştır.**
* **PyTorch nn.Module Entegrasyonu: Sınıflar nn.Module'dan türetildiği (inherit) için, PyTorch'un to(device), parameters(), train(), eval() gibi güçlü yerleşik metodlarına otomatik erişim sağlanmıştır. Bu metodları elle yazmak yerine miras almak, kodun güvenilirliğini ve standardizasyonunu artırmıştır.**
* **Deneysel Tutarlılık (Reproducibility): Sınıf yapısı, her bir model örneği (instance) için bağımsız bir durum (state) tutmamıza olanak tanır. agirliklari\_baslat metodu sınıfa gömülerek, her deney öncesi modelin tıpatıp aynı başlangıç ağırlıklarına (seed) döndürülmesi garanti altına alınmıştır. Bu özellik olmadan, algoritmaların performans karşılaştırması adil (fair comparison) olamazdı.**

**A. Tek Katmanlı Regresyon Modeli (BasitModel)**

**Ödevin A kısmında istenen temel regresyon modeli, giriş vektörü ile hedef çıktı arasında doğrusal bir ilişki kurup, sonucu hiperbolik tanjant fonksiyonu ile sıkıştıran tek katmanlı bir yapıdır.**

* **Matematiksel Model:**

**Burada  giriş vektörünü (soru + cevap embedding),**

**ağırlık matrisini ve  sapma (bias) değerini temsil eder.**

* **Ağırlık Başlatma (Initialization): Modelin öğrenmeye başlayabilmesi için**

**ve  değerlerinin başlangıç noktası kritiktir. Aktivasyon fonksiyonu olarak Tanh kullanıldığı için, ağırlıklar Xavier (Glorot) Uniform dağılımı ile başlatılmıştır. Bu yöntem, gradyanların çok küçük (vanishing) veya çok büyük (exploding) olmasını engelleyerek, sinyalin ağ boyunca sağlıklı yayılmasını sağlar. Bias değerleri ise standart olarak 0 ile başlatılmıştır.**

* **Tekrarlanabilirlik (Seeding): Deneylerin güvenilirliği açısından, agirliklari\_baslat(seed) fonksiyonu ile her deney öncesinde ağırlıklar belirli bir tohum (seed) değeri kullanılarak sıfırlanmıştır. Bu sayede GD, SGD ve Adam algoritmaları tamamen aynı başlangıç koşullarında yarıştırılmıştır.**

**Aşağıda tek katmanlı regresyon modelimin python scripti halini inceleyebilirsiniz:**

**A screen shot of a computer program

AI-generated content may be incorrect.**

**A computer screen with text

AI-generated content may be incorrect.**

**Rastgelelik üretecinin başlangıç noktasıdır. Eğer seed'i 42 verirsen, bilgisayar her seferinde tıpatıp aynı "rastgele" sayı dizisini üretir.Böylece kodu bugün de çalıştırsam, yarın da çalıştırsam modelin hep aynı ağırlıklarla başlar.(Xavier (Glorot) Initialization kullanıyoruz (Tanh için uygundur))**

**B. Çok Katmanlı Algılayıcı (BonusModelMLP)**

**Modelin karmaşıklığını artırarak performans etkisini gözlemlemek amacıyla (Bonus kapsamında), gizli katman içeren bir MLP (Multi-Layer Perceptron) mimarisi tasarlanmıştır.**

* **Mimari:**
  1. **Giriş Katmanı: 2048 nöron (Embedding boyutu).**
  2. **Gizli Katman (Hidden Layer): 64 nöron. Aktivasyon fonksiyonu olarak ReLU () kullanılmıştır. ReLU, modelin doğrusal olmayan karmaşık ilişkileri öğrenmesini sağlar.** **Negatif değerleri sıfırlayarak seyreklik sağlar ve öğrenmeyi hızlandırır.**
  3. **Çıkış Katmanı: 1 nöron. Sonuç yine -1 ile +1 aralığında olması gerektiği için çıkışta Tanh kullanılmıştır.**

**Yani kısaca: Giriş (2048) -> Linear -> ReLU -> Gizli (64) -> Linear -> Tanh -> Çıkış (1).**

**OOP Katkısı:** BonusModelMLP sınıfı, BasitModel ile aynı arayüzü (interface) paylaşır. Bu sayede eğitim döngüsündeki (main\_deney.py) kodlarda hiçbir değişiklik yapmadan, sadece sınıf adını değiştirerek (Polimorfizm) model değiştirmek mümkün olmuştur.

**Bu süreçte optimizasyon döngülerini öğrendim. Ve modeller.py da ilkini kullandım. Bu döngünün yalnızca 1. Forward pass i modeller.py da kullandım. Geri kalanını ozel\_optimizer.py’da devam ettirdim.**

**Aşağıda MLP class’ımın kodları bulunmaktadır:**

**A screen shot of a computer program

AI-generated content may be incorrect.**

**Optimizasyon Döngüsü:**

* **1.İleri Yayılım (Forward Pass): Kısaca veriyi modele sokarak tahmini almak denebilir.**
* **ve koda göre**

**A computer code with text

AI-generated content may be incorrect.**

* **2. Calculate Loss: Hatayı hesapla.**
* **Loss = criterion(y\_pred, y\_real)**
* **3. Zero Gradients: Önceki adımlardan kalan türevleri temizle.(Türelerin birikmemesi lazım.)**
* **Optimizer.zero\_grad()**
* **Türevleri sıfırlama fonksiyonunu ozel\_optimizer.py da kendim yazdım.**
* **4. Backward Pass: Hatayı geriye doğru yayarak türevleri hesapla.**
* **Loss.backward()**
* **5. Optimizer Step: Ağırlıkların güncellendiği yer.**
* **Optimizer.step()**

**5.  Özel Optimizasyon Algoritmalarının İmplementasyonu ve Öğrenim Süreci**

**(ozel\_optimizer.py)**

**Projenin bu aşamasında, PyTorch kütüphanesinin sağladığı hazır optimizasyon araçlarını (torch.optim) kullanmak yerine, literatürdeki temel algoritmaların (GD, SGD, AdaGrad, RMSProp, Adam) çalışma mantığını derinlemesine kavramak amacıyla, bu algoritmalar sıfırdan matematiksel formülleri kullanılarak kodlanmıştır.**

* **Neden Kendi Kütüphanemiz? (Custom Implementation) Hazır fonksiyonlar (black-box), algoritmaların iç çalışma mantığını gizler. Algoritmaları sıfırdan sınıflar halinde yazmak, türev alma (differentiation) ve ağırlık güncelleme (weight update) süreçlerinin matematiksel arka planını tam anlamıyla kavramamı sağlamıştır. Ayrıca zaten dersimiz içeriğinde bu algoritmaları mantıklarıyla beraber öğremiştik.**
* **OOP ve Kalıtım Avantajı: Yazılan ozel\_optimizer.py modülü, tüm algoritmalar için ortak bir Base Class (Temel Sınıf) mimarisi üzerine kurulmuştur. Bu yapı sayesinde kod tekrarı önlenmiş ve nesne yönelimli programlama (OOP) prensipleri uygulanmıştır.**
  + **zero\_grad() gibi ortak fonksiyonlar sadece bir kez ana sınıfa yazılmış (DRY Prensibi), alt sınıflar ise sadece kendilerine özgü step() matematiklerini uygulamıştır. Bu yapı, kod tekrarını önlemiş ve temiz (clean) bir kod mimarisi sunmuştur.**
  + **Gradyan Yönetimi (zero\_grad): Her eğitim adımında (step) PyTorch, hesaplanan türevleri biriktirme (accumulation) eğilimindedir. Yazdığım zero\_grad fonksiyonu ile her iterasyon öncesi p.grad = None komutu uygulanarak bu hafıza manuel olarak temizlenmiş ve hatalı öğrenmenin önüne geçilmiştir.**

**A screen shot of a computer program

AI-generated content may be incorrect.**

* + **Hesaplama Grafiği Yönetimi (torch.no\_grad): Ağırlık güncelleme işlemi (w = w - lr \* grad), modelin ileri yayılım (forward) işleminin bir parçası değildir. Bu nedenle güncelleme adımları with torch.no\_grad(): bloğu içerisine alınarak, PyTorch'un bu matematiksel işlemleri türev zincirine (computational graph) eklemesi engellenmiştir.**
  + **In-Place Operasyonlar: Bellek verimliliğini artırmak için ağırlık tensörleri kopyalanarak değil, doğrudan bellek üzerindeki değerleri değiştirilerek (p.sub\_, p.add\_ gibi alt çizgi ile biten metodlar) güncellenmiştir.**

1. **SGD (Stochastic Gradient Descent):**
   * ***Matematik:***
   * ***Uygulama:* Türevi (grad) öğrenme oranı (lr) ile çarpıp mevcut ağırlıktan çıkararak en basit güncellemeyi yapar.**

**A computer screen with text and symbols

AI-generated content may be incorrect.**

1. **GD (Gradient Descent) İmplementasyon Stratejisi**

**Literatürde GD ve SGD arasındaki temel fark, ağırlık güncelleme formülü değil, gradyanın hesaplanmasında kullanılan veri sayısıdır.**

**SGD: Her adımda 1 örnek kullanır.**

**GD: Her adımda tüm eğitim setini () kullanır.**

**Bu nedenle, projede GD için ayrı bir Python sınıfı yazmak yerine, Veri Yükleyici (DataLoader) stratejisi optimize edilmiştir. OzelSGD sınıfı, batch\_size parametresi eğitim kümesinin toplam boyutuna () eşitlenerek çalıştırıldığında, matematiksel olarak Klasik Gradyan İnişi (GD) algoritmasına dönüşmektedir. Bu yöntemle, kod tekrarından kaçınılmış ve batch\_size hiperparametresinin algoritma davranışı üzerindeki etkisi doğrudan gözlemlenebilmiştir.**

1. **Adam (Adaptive Moment Estimation):**
   * ***Matematik:* En karmaşık yapıya sahip bu algoritma için, gradyanların hem 1. Momenti (Momentum - ) hem de 2. Momenti (Hız - ) hesaplanmıştır. Kodda özellikle başlangıç adımlarında momentlerin 0'a yakınsaması sorununu çözmek için Bias Correction (Sapma Düzeltme) formülleri () uygulanmıştır.**
   * ***Uygulama:* Sınıf yapısı sayesinde her parametreye ait geçmiş momentum (m) ve hız (v) değerleri, parametreye ait geçmiş momentum () ve hız () değerleri, parametrelerle birebir eşleşen self.m ve self.v listelerinde hafızada tutulabilmiştir. Her güncelleme adımında (step), parametreler indeksleri ile (enumerate) taranarak ilgili geçmiş veriye (momentlere) hızlıca erişilmiştir. Fonksiyonel programlamada bu "durum saklama" (statefulness) ve parametre-hafıza eşleştirmesi işlemi çok daha karmaşık olurdu.**

**A screen shot of a computer program

AI-generated content may be incorrect.**

1. **AdaGrad (Adaptive Gradient):**

**Çalışma Mantığı: Standart SGD'nin aksine, her parametre için sabit bir öğrenme hızı (learning rate) kullanmaz. Sık güncellenen parametrelerin öğrenme hızını düşürürken, seyrek güncellenenlerin hızını artırır.**

**İmplementasyon: Kodda self.sum\_squared\_gradients listesi oluşturularak, her adımda gradyanların kareleri (**

**G\_t=G\_(t-1)+g\_t^2) biriktirilmiştir. Güncelleme adımında ise efektif öğrenme hızı η/√(G\_t+ϵ) formülüyle hesaplanmıştır.**

**A computer screen shot of a program

AI-generated content may be incorrect.**

1. **RMSProp (Root Mean Square Propagation):**

**Çalışma Mantığı: AdaGrad'ın "öğrenme hızının çok çabuk sıfıra inmesi" problemini çözmek için tasarlanmıştır. Geçmiş gradyanların toplamı yerine hareketli ortalamasını (moving average) kullanır.**

**İmplementasyon: Kodda self.moving\_average listesi tutulmuş ve bir unutma faktörü (α=0.99) kullanılmıştır. Formül v\_t=αv\_(t-1)+(1-α)g\_t^2 şeklinde kodlanarak, sadece yakın geçmişteki gradyanların etkili olması sağlanmıştır. Bu sayede modelin eğitimi durmadan (learning rate vanish olmadan) devam edebilmiştir.**

**A computer screen with colorful text

AI-generated content may be incorrect.**

**Eğitim Döngüsü ve Veri Yönetimi (egitim.py)**

**Bu modül, hazırlanan veri setinin yüklenmesi, seçilen modelin (Basit veya MLP) başlatılması ve belirlenen özel optimizasyon algoritmasıyla eğitilmesinden sorumlu ana pipeline'ı (iş akışını) oluşturur. Geliştirme sürecinde PyTorch'un tensör operasyonları ve bellek yönetimi üzerine önemli tecrübeler edinilmiştir.**

**A. Veri Hazırlığı ve Tensör Boyutlandırma (verileri\_yukle)**

**Eğitim sürecinin en kritik adımlarından biri, verilerin PyTorch'un beklediği formata getirilmesidir.**

* **Tip Dönüşümü (.float()): Regresyon modelleri ondalıklı sayılarla çalıştığı için, yüklenen veriler açıkça float32 tipine dönüştürülmüştür. Bu yapılmadığında Double vs Float uyuşmazlığı hataları alınmaktadır.**
* **Boyutlandırma (.view(-1, 1)): Tek boyutlu etiket vektörleri (), matris çarpımı sırasında yayınlama (broadcasting) hatalarına yol açabilmektedir. Bu nedenle hedef vektörler (), (Batch\_Size, 1) boyutuna getirilerek model çıktısıyla birebir uyumlu hale getirilmiştir.**
* **Veri Kontrolü: Kod içerisine eklenen "Debug" satırları ile etiketlerin gerçekten -1 ve +1 olup olmadığı kontrol edilerek veri bütünlüğü garanti altına alınmıştır.**

**A screen shot of a computer program

AI-generated content may be incorrect.**

**B. Dinamik Eğitim Fonksiyonu (modeli\_egit)**

**Bu fonksiyon, farklı hiperparametre kombinasyonlarının (Algoritma, Batch Size, Learning Rate) tek bir yapı üzerinden test edilmesini sağlayan esnek bir yapıdadır.**

1. **Optimizer Seçimi (Factory Pattern): Kullanıcıdan gelen string ifadeye ("Adam", "SGD" vb.) göre ilgili özel optimizer sınıfı dinamik olarak başlatılmıştır.**

**A screen shot of a computer program

AI-generated content may be incorrect.**

1. **Eğitim Adımları: Her epoch'ta standart PyTorch eğitim döngüsü uygulanmıştır:**
   * **optimizer.zero\_grad(): Geçmiş gradyanlar temizlenmiştir.**
   * **loss.backward(): Hata geriye yayılmış ve türevler hesaplanmıştır.**
   * **optimizer.step(): Ağırlıklar, yazdığımız özel algoritmalarla güncellenmiştir.**

**A screen shot of a computer program

AI-generated content may be incorrect.**

1. **Kayıp Fonksiyonu (Loss) Takibi:**
   * **Eğitim verisi küçük (100 örnek) olduğu için modelin Overfitting (Aşırı Öğrenme) yaptığı gözlemlenmiştir. Bu nedenle modelin gerçekten öğrenip öğrenmediğini analiz edebilmek için grafiklerde "Test Loss" yerine "Average Training Loss" kaydedilmesi tercih edilmiştir.**

**C. Ağırlık Yörüngelerinin Kaydı (Part B İçin Hazırlık)**

**Ödevin B kısmındaki T-SNE görselleştirmesi için, eğitim sırasındaki ağırlık değişimlerinin kaydedilmesi gerekmektedir.**

* **Fonksiyon içerisinde, her epoch sonunda model.parameters() taranarak ilk katmanın ağırlıkları .data.view(-1).cpu().numpy() zinciri ile bellekten (GPU/CPU) alınıp NumPy dizisine dönüştürülmüş ve bir listede saklanmıştır. Bu sayede optimizasyon sürecinin "yolu" çizilebilir hale gelmiştir.**

**D. Öğrenilen Dersler**

**Bu modülün kodlanması sırasında:**

* **model.train() ve model.eval() modları arasındaki farkın (Dropout ve Batch Norm davranışları için) önemi kavranmıştır.**
* **with torch.no\_grad(): bloğunun, değerlendirme (inference) aşamasında hesaplama grafiğini (computational graph) oluşturmayarak bellek tasarrufu sağladığı ve işlemi hızlandırdığı deneyimlenmiştir.**
* **Batch Size'ın eğitim kararlılığı üzerindeki etkisi (GD vs SGD farkı) DataLoader yapılandırması üzerinden somut olarak gözlemlenmiştir.**

**Deney Yönetimi ve Hiperparametre Orkestrasyonu (main\_deney.py)**

**Bu modül, projenin en kapsamlı parçası olup; farklı optimizasyon algoritmalarının, değişen tohum (seed) değerlerinin ve model mimarilerinin tek bir otomatik döngü içerisinde test edilmesini ve sonuçların analiz için kaydedilmesini sağlar. Bu dosyanın geliştirilmesi sürecinde "deney tasarımı" ve "hiperparametre optimizasyonu" konularında kritik tecrübeler edinilmiştir.**

**A computer screen shot of a program

AI-generated content may be incorrect.**

**A. GD ve SGD Ayrımı (Batch Size Stratejisi)**

**Projenin en önemli teknik kazanımlarından biri, Gradyan İnişi (GD) ve Stokastik Gradyan İnişi (SGD) arasındaki farkın kod düzeyinde nasıl yönetildiğidir.**

* **Teori: GD ve SGD matematiksel olarak aynı güncelleme kuralını kullanır; fark, türevin hesaplandığı veri kümesinin büyüklüğüdür.**
* **Uygulama: GD için ayrı bir optimizer sınıfı yazmak yerine, OzelSGD sınıfı kullanılmış ancak BATCH\_SIZE\_CONFIG sözlüğü aracılığıyla GD için Batch Size = Tüm Veri Seti () olarak ayarlanmıştır. SGD için ise bu değer 1 tutulmuştur.**
* **Sonuç: Bu yaklaşım sayesinde, DataLoader yapısının esnekliği kullanılarak aynı algoritma motoru üzerinden iki farklı optimizasyon stratejisi (Batch vs Stochastic) simüle edilmiştir.**

**B. Hiperparametre Ayarlaması (Learning Rate Tuning)**

**Derin öğrenme modellerinin eğitiminde "Tek bir Learning Rate (LR) hepsine uyar" mantığının işlemediği, bu modüldeki deneylerle somut olarak görülmüştür. LR\_CONFIG sözlüğü, deneme-yanılma (empirical) yöntemiyle optimize edilmiştir:**

1. **GD (LR=0.5): Gradyanlar tüm veri seti üzerinden ortalaması alınarak hesaplandığı için gürültüsüz ancak küçük değerlerdedir. Modelin öğrenmesi için çok yüksek bir LR (0.5) gerektiği, aksi halde eğitimin çok yavaş ilerlediği tespit edilmiştir.**
2. **SGD (LR=0.01): Her adımda çok gürültülü güncellemeler yaptığı için, modelin savrulmasını önlemek adına daha düşük bir LR tercih edilmiştir.**
3. **Adam (LR=0.01): Adaptif bir algoritma olmasına rağmen, çok küçük LR değerlerinde (0.001) momentumun etkisiyle parametrelerin yerinde saydığı, değerin artırılmasıyla (0.01) performansın dramatik şekilde iyileştiği gözlemlenmiştir.**

**A screen shot of a computer program

AI-generated content may be incorrect.**

**C. Deneylerin Tekrarlanabilirliği ve Otomasyon**

**Bilimsel geçerlilik adına, her algoritma 5 farklı rastgele başlangıç noktası (Seed: 10, 20, 30, 40, 50) ile test edilmiştir.**

* **İç İçe Döngüler: Algoritmalar ve Seeds listeleri üzerinden kurulan yapı sayesinde, tek bir çalıştırmada**

**farklı eğitim senaryosu otomatik olarak gerçekleştirilmiştir.**

* **Hata Yönetimi (Try-Except): Uzun süren deneylerde tek bir seed'in hata vermesinin tüm süreci durdurmaması için try-except blokları kullanılarak sürecin dayanıklılığı (robustness) artırılmıştır.**

**A screen shot of a computer program

AI-generated content may be incorrect.**

* **Veri Saklama: Sonuçlar (Loss geçmişi ve Ağırlık değişimleri) torch.save ile disk üzerine .pt formatında serileştirilerek kaydedilmiştir. Bu yapı, eğitim süreci ile analiz/çizim sürecini birbirinden ayırarak (decoupling) daha modüler bir çalışma ortamı sağlamıştır.**

**A black background with yellow and orange text

AI-generated content may be incorrect.**

**D. Kazanımlar**

**Bu modül sayesinde:**

* **Büyük ölçekli deneylerin manuel olarak değil, parametrik konfigürasyonlarla yönetilmesi gerektiği öğrenilmiştir.**
* **Bir algoritmanın başarısının sadece matematiksel formülüne değil, ona atanan hiperparametrelere (özellikle Learning Rate ve Batch Size) ne kadar bağımlı olduğu deneysel olarak kanıtlanmıştır.**
* **Yazılım mühendisliği prensibi olan "Don't Repeat Yourself (DRY)" ilkesi, döngüler ve konfigürasyon sözlükleri kullanılarak uygulanmıştır.**

**Analiz ve Görselleştirme (analiz\_ve\_gorsellestirme.py)**

**Eğitim sürecinden elde edilen ham verilerin (.pt dosyaları) anlamlandırılması, algoritmaların performanslarının kıyaslanması ve yüksek boyutlu optimizasyon yörüngelerinin insan algısına uygun hale getirilmesi amacıyla bu modül geliştirilmiştir.**

**A. Performans Metriklerinin Görselleştirilmesi (performans\_ciz)**

**Algoritmaların başarısını tek bir boyut üzerinden değerlendirmek yanıltıcı olabileceğinden, analiz iki farklı eksen üzerinden kurgulanmıştır:**

1. **Epoch Bazlı Kararlılık (Sol Grafik):**
   * **Algoritmaların veri setini kaç kez taradığında (Epoch) ne kadar öğrendiğini gösterir.**
   * **Burada SGD ve Adam'ın, GD'ye kıyasla epoch başına daha hızlı hata düşürdüğü gözlemlenmiştir.**
   * ***Teknik Detay:* 5 farklı seed'den gelen verilerin ortalaması (np.mean) alınarak gürültüden arındırılmış genel eğilim çizdirilmiştir.**
2. **Maliyet/Güncelleme Bazlı Analiz (Sağ Grafik):**
   * **Bu grafik, hesaplama maliyetini temsil eder. X ekseni "Toplam Ağırlık Güncellemesi" olarak hesaplanmıştır.**
   * **Hesaplama Mantığı: Kod içerisindeki BATCH\_SIZES sözlüğü kullanılarak; GD için epoch başına 1 güncelleme, SGD için 50 güncelleme (Veri Sayısı kadar) hesaplanmıştır.**
   * **Logaritmik Ölçek: SGD'nin güncelleme sayısı (5000+) ile GD'nin güncelleme sayısı (100) arasında büyük fark olduğu için X ekseninde ax2.set\_xscale('log') kullanılarak logaritmik ölçeklendirme yapılmıştır.**

**B. Yüksek Boyutlu Yörüngelerin İndirgenmesi (tsne\_analizi\_yap)**

**Modelin ağırlık uzayı 2049 boyutludur (). Bu uzayda algoritmaların nasıl hareket ettiğini (zikzaklı mı, doğrusal mı) görebilmek için boyut indirgeme uygulanmıştır.**

1. **T-SNE (t-Distributed Stochastic Neighbor Embedding): PCA gibi lineer yöntemler yerine, karmaşık manfolt yapılarını koruyabilen T-SNE algoritması (sklearn.manifold.TSNE) tercih edilmiştir.**
2. **Veri Seyreltme (Subsampling): T-SNE işleminin hesaplama maliyetini düşürmek ve grafiği okunabilir kılmak için, ağırlık geçmişi w\_hist[::5] komutuyla her 5 adımda bir örneklenmiştir.**
3. **Görsel İşaretleyiciler: Yörüngenin yönünü ve gelişimini anlamak için:**
   * **Yeşil Kare: Başlangıç noktası (Rastgele ilk ağırlıklar).**
   * **Kırmızı Yıldız: Bitiş noktası (Eğitilmiş model).**
   * **Oklar: Hareket yönünü gösteren vektörler eklenmiştir.**

**C. Kazanımlar**

**Bu modülün geliştirilmesiyle:**

* **Bilimsel verilerin sunumunda eksen seçiminin (Epoch vs Update) yorumu nasıl değiştirdiği kavranmıştır.**
* **Yüksek boyutlu verilerin görselleştirilmesinde T-SNE hiperparametrelerinin (perplexity, learning\_rate) önemi deneyimlenmiştir.**
* **Matplotlib ve Seaborn kütüphaneleri ile yayın kalitesinde (publication-quality) akademik grafik üretme yetkinliği kazanılmıştır.**

**Aşağıda bu ödev için çıkardığım tüm grafikler ve açıklamalı kodum bulunmaktadır:**

**A screenshot of a computer program

AI-generated content may be incorrect.**

**A screen shot of a computer program

AI-generated content may be incorrect.**

**A computer screen with many colorful text

AI-generated content may be incorrect.**

**A screen shot of a computer program

AI-generated content may be incorrect.**

**A screen shot of a computer program

AI-generated content may be incorrect.**

**A black background with many small colored lines

AI-generated content may be incorrect.**

**A graph of different colored lines and dots

AI-generated content may be incorrect.A graph of a graph

AI-generated content may be incorrect.A graph with different colored lines

AI-generated content may be incorrect.A white grid with black and red triangles and green and red text

AI-generated content may be incorrect.**

**A graph of different colored lines

AI-generated content may be incorrect.**

**Karşılaşılan Zorluklar ve Çözüm: Overfitting İkilemi**

**Projenin ilk aşamalarında, modelin başarısını ölçmek için Test Hatası (Test Loss) metriği kullanılmıştır. Ancak yapılan deneylerde Loss değerinin 1.0 civarında takılı kaldığı ve düşmediği gözlemlenmiştir. Kodun ve türev hesaplamalarının doğruluğu defalarca kontrol edilmesine rağmen sonucun değişmemesi üzerine detaylı bir literatür taraması yapılmıştır.**

**Tespit: Eğitim kümesinde sadece 50 örnek bulunmasına karşın, modelin parametre sayısı 2049'dur. Bu durum, derin öğrenmede klasik bir problem olan Overfitting (Aşırı Öğrenme/Ezberleme) durumuna yol açmaktadır. Model, eğitim verisini ezberlemekte ancak görmediği test verisinde genelleme yapamamaktadır.**

**Çözüm: Algoritmaların matematiksel olarak çalışıp çalışmadığını (yakınsama yeteneklerini) doğrulamak amacıyla metrik olarak Eğitim Hatası (Training Loss) seçilmiştir. Bu değişikliğin ardından, algoritmaların hatayı hızla düşürdüğü ve optimizasyon sürecinin başarıyla işlediği kanıtlanmıştır.**

**Performans Grafiklerinin Analizi**

**Elde edilen sonuçlar iki farklı perspektiften incelenmiştir:**

**Şekil 1: Eğitim Kararlılığı ve Maliyeti Grafikleri  
A graph of different colored lines

AI-generated content may be incorrect.**

**A. Epoch Bazlı Analiz (Sol Grafik)**

**Bu grafik, "Algoritma veri setini kaç kez gördüğünde ne kadar öğreniyor?" sorusuna cevap verir.**

* **SGD (Mavi): En dikkat çekici performansı sergilemiştir. Epoch başına 50 kez (veri sayısı kadar) güncelleme yapması, kaybın (loss) logaritmik bir hızla düşmesini sağlamıştır. 100 epoch sonunda hatayı 0.35 seviyelerine indirmiştir.**
* **Adam (Kırmızı) ve RMSProp (Mor): Beklendiği üzere hızlı bir başlangıç yapmışlardır. Ancak bu spesifik veri setinde SGD'nin gürültülü yapısı, yerel minimumlardan kaçışta adaptif yöntemlerden daha etkili olmuş olabilir.**
* **GD (Yeşil): Çok daha yavaş ve lineer bir iniş sergilemiştir. Bu beklenen bir durumdur çünkü her epochta ağırlıkları sadece 1 kez güncellemektedir.**

**B. Hesaplama Maliyeti Analizi (Sağ Grafik)**

**Bu grafik, "Algoritma kaç adım attığında (işlem maliyeti) ne kadar öğreniyor?" sorusuna cevap verir.**

* **GD'nin Verimliliği: Grafikteki Yeşil çizgi (GD) en solda yer almaktadır. Bu, GD'nin çok az sayıda güncellemeyle (yaklaşık 100 adım) hatayı ciddi oranda düşürdüğünü gösterir. Her bir adımı "altın değerindedir" çünkü gürültüsüz, tam gradyan kullanır.**
* **SGD'nin Maliyeti: Mavi çizgi (SGD) sağ tarafa (logaritmik eksende**

**üzerine) uzanmaktadır. SGD'nin epoch başarısı yüksek olsa da, aynı hata oranına inmek için GD'den katbekat fazla işlem yapması (yaklaşık 5000 adım) gerekmiştir.**

**4. OPTİMİZASYON SÜRECİNİN GÖRSELLEŞTİRİLMESİ (T-SNE)**

**2049 boyutlu ağırlık uzayındaki hareketleri 2 boyuta indirgediğimiz T-SNE grafikleri, algoritmaların karakteristiğini net bir şekilde ortaya koymuştur. Her grafikte Yeşil Kare başlangıç noktasını, Kırmızı Yıldız ise 100 epoch sonundaki bitiş noktasını temsil eder.**

**1. SGD Yörüngeleri (Şekil: SGD T-SNE 2D)**

**SGD grafiğinde yörüngelerin oldukça uzun ve kavisli olduğu görülmektedir.**

* **Yorum: Stokastik yapısı gereği, her adımda rastgele bir örnek seçmesi, yörüngenin "zikzaklı" veya gürültülü olmasına neden olur. Ancak T-SNE grafiğinde görülen bu geniş yayılma, SGD'nin arama uzayını (search space) daha geniş taradığını ve global minimumu bulma şansının arttığını gösterir. Farklı seed'lerin (farklı renklerin) tamamen farklı yönlere gidip farklı yerel minimumlara oturması, loss yüzeyinin dışbükey (non-convex) olduğunu kanıtlar.**

**2. GD Yörüngeleri (Şekil: GD T-SNE 2D)**

**GD grafiğindeki çizgiler, SGD'ye kıyasla çok daha pürüzsüz ve "temiz"dir.**

* **Yorum: GD, tüm veri setinin ortalamasını kullandığı için gradyan yönü her zaman kesin ve nettir. Yörüngelerde kararsızlık veya titreme yoktur. Başlangıçtan bitişe kararlı bir şekilde ilerler. Ancak bu kararlılık, bazen en yakın yerel minimuma saplanıp kalmasına (saddle point) neden olabilir.**

**3. Adam ve RMSProp Yörüngeleri (Şekil: Adam/RMSProp T-SNE 2D)**

**Bu grafiklerdeki yörüngeler, momentum etkisinden dolayı daha "ok gibi" ve hedefe yöneliktir.**

* **Yorum: Adam algoritması, geçmiş gradyanların ivmesini (momentum) kullandığı için, SGD gibi savrulmaz. Başlangıçta hızla bir yöne karar verir ve o yöne doğru ivmelenir. T-SNE grafiğindeki yörüngelerin başlangıçta sert bir atılım yapıp sonra kendi etrafında kümelenmesi, algoritmanın hızlıca bir çukura inip orada ince ayar (fine-tuning) yapmaya çalıştığını gösterir.**

**4. AdaGrad Yörüngeleri (Şekil: AdaGrad T-SNE 2D)**

**AdaGrad yörüngelerinin diğerlerine göre daha kısa kaldığı veya sonlara doğru sıkıştığı görülmektedir.**

* **Yorum: AdaGrad'ın doğası gereği, biriken gradyan kareleri öğrenme hızını (Learning Rate) sürekli düşürür. Bu durum, eğitimin ilerleyen safhalarında modelin "erken durmasına" veya çok küçük adımlar atmasına neden olmuştur. Grafikteki hareketin sönümlenmesi bu teorik bilgiyi doğrulamaktadır.**

**5. SONUÇ VE KAZANIMLAR**

**Bu çalışma sayesinde:**

1. **Derin öğrenme modellerinde veri boyutu ile model kapasitesi arasındaki dengenin önemi ve overfitting kavramı deneysel olarak yaşanmıştır.**
2. **Loss fonksiyonunun düşmemesinin her zaman kod hatası olmadığı, bazen yanlış metriğe (Test Loss) bakmaktan kaynaklandığı öğrenilmiştir.**
3. **GD'nin hesaplama açısından verimli (az adım) ama yavaş; SGD'nin ise maliyetli (çok adım) ama hızlı öğrenen bir yapı olduğu grafiklerle ispatlanmıştır.**
4. **T-SNE analizi ile matematiksel formüllerin (Momentum, Adaptive LR) ağırlık uzayındaki fiziksel hareketlere (Pürüzsüzlük, Zikzak, Sönümlenme) nasıl dönüştüğü görselleştirilmiştir.**

**Kodumu anlattığım Youtube videom:**

**https://youtu.be/V\_MiZSI\_Yj8**