

Derin Öğrenme Modellerinin GPU Eğitimi Analizine Giriş

Doç. Dr. Erdem Akagündüz



23 Aralık 2025



Doç. Dr. Erdem Akagündüz
akaerdem@metu.edu.tr

- **Akademik Görev:**
 - ODTÜ Enformatik Enstitüsü,
Modelleme ve Simülasyon ABD Öğretim Üyesi.
- **Uzmanlık Alanları:**
 - Bilgisayarla Görü
 - Makine Öğrenmesi ve Örüntü Tanıma
 - Sinyal İşleme tabanlı Derin Öğrenme
- **Araştırma Laboratuvarı:**
 - ODTÜ Enformatik Enstitüsü
Uygulamalı Yapay Zeka Araştırma Laboratuvarı
(AIRLab) yöneticisi
 - <https://airlab-ii.metu.edu.tr/>

- Derin Öğrenme Eğitiminin Anatomisi
- GPU Mekanığı

Bölüm 1: Derin Öğrenme Eğitiminin Anatomisi

Derin sinir ağı eğitim döngüsü nedir, hangi parçalardan oluşur?

Derin Öğrenme “Eğitim” Döngüsü Nedir?

• Optimizasyon Problemi:

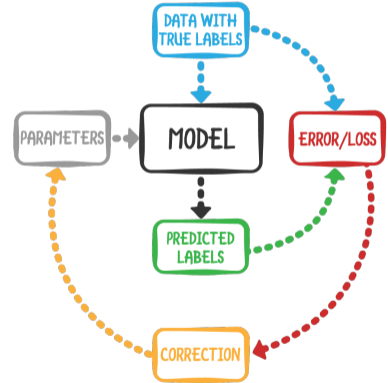
- Eğitimi, yüksek boyutlu parametre uzayında bir “en düşük hata” arayışı olarak tanımlıyoruz.

• Temel Bileşenlerin Etkileşimi:

- **Model** $f(x, \theta)$: Girdi verisini işleyen mimari.
- **Kayıp Fonksiyonu** $L[f(x, \theta), g]$: Başarıyı sayısallaştıran metrik.
- **Optimizer**: Hatayı azaltmak için parametreleri güncelleyen algoritma.

• Döngüsel Süreç:

- Veri → Tahmin → Hata → Güncelleme.



İleri Yayılım (Forward Pass)

• Tahmin Mekanizması:

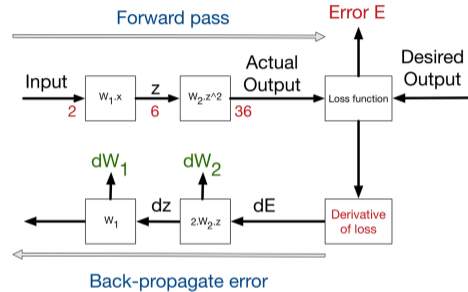
- Girdi verisinin katmanlar arasından geçerek çıktıya dönüşümü.

• Hesaplama Yükü:

- Temel işlem: $y = \sigma(W \cdot x + b)$
- Devasa boyutlu **Matris Çarpımları**
- Aktivasyon fonksiyonları (ReLU, Softmax vb.).

• GPU Analizi İçin Kritik Notlar:

- **Bellek:** Ara katmanlardaki aktivasyon değerleri, bellekte tutulur.



Kayıp Fonksiyonu: Başarıyı Ölçmek

• Hatanın Sayısallaştırılması:

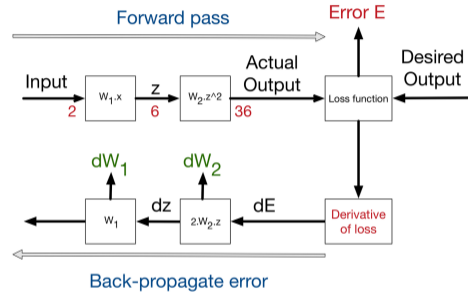
- Modelin tahmini (\hat{y}) ile gerçek etiket (y) arasındaki farkın tek bir skaler değere indirgenmesi.

• Temel Metrikler:

- **MSE (Mean Squared Error):** Regresyon problemleri için.
- **Cross-Entropy:** Sınıflandırma problemleri için (Olasılık dağılımı farkı).

• Neden Bu Sayıya Muhtacız?

- **Geri Yayılımın Başlangıcı:** Gradyan hesaplaması için gereken türev zinciri bu noktadan başlar.
- **Skaler Dönüşüm:** GPU'daki devasa matrislerin tek bir sayıya indirgenmesi süreci.



Geri Yayılım (Backpropagation)

• Zincir Kuralı (Chain Rule):

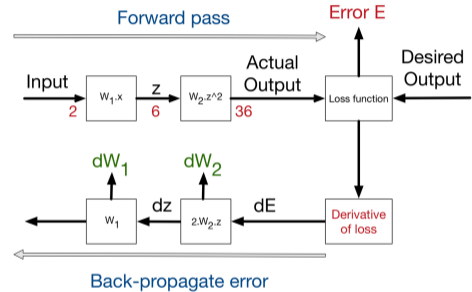
- Kayıp değerinden (L) başlayarak her bir parametreye (w) doğru türev akışı: $\frac{\partial L}{\partial w}$.

• Hesaplama Yoğunluğu:

- İleri yayılımın yaklaşık **2-3 katı** daha fazla matris işlemi gerektirir.
- Gradyanların hesaplanması ve saklanması.

• GPU Analizi İçin Kritik Not:

- GPU çekirdeklerinin (CUDA Cores) en yoğun çalıştığı aşamadır.
- Veri tipi (FP32 vs. FP16) burada hızı doğrudan etkiler.



Optimizer: Karar Verici

● Parametre Güncelleme:

- Gradyanları kullanarak ağırlıkların (W) revize edilmesi:

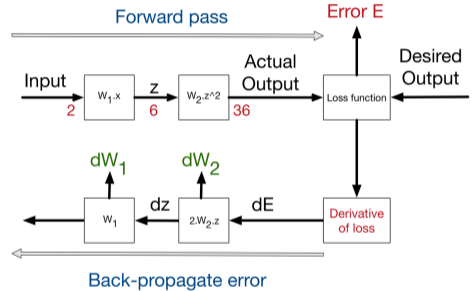
$$W_{new} = W_{old} - \eta \cdot \nabla L(W)$$

● Popüler Algoritmalar:

- **SGD:** Basit ve kararlı gradyan inişi.
- **Adam / RMSProp:** Adaptif öğrenme oranları ile daha hızlı yakınsama.

● Öğrenme Oranı (Learning Rate - η):

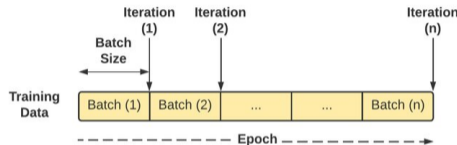
- "Ne kadar büyük adımlar atmamız?"
- Çok büyük η : Divergence (Iraksama).
- Çok küçük η : Yavaş eğitim / Yerel minimuma takılma.



Batch ve Epoch: Veri Yönetimi

• Batch Size: Neden tüm veriyi vermiyoruz?

- **Donanım Kısıtı:** Veri setinin tamamı GPU belleğine (VRAM) sığmaz.
- **Algoritmik Avantaj:** "Gürültülü gradyanlar" yerel minimumlardan kurtulmaya yardımcı olur.



• Iteration vs. Epoch:

- **Iteration:** Bir adet batch'in işlenmesi.
- **Epoch:** Tüm veri setinin modelden bir kez geçmesi.

• GPU Analizi İçin Kritik Not:

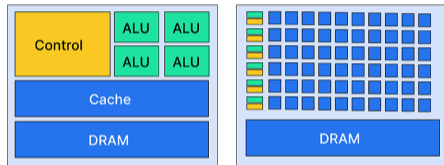
- **Parallelleştirme:** Batch size ne kadar büyükse, GPU o kadar verimli (paralel) çalışır.
- **Darboğaz:** Çok küçük batch size, GPU çekirdeklerinin boşa kalmasına neden olur.

Bölüm 2: GPU Mekaniği

Bir modelin GPU üzerinde adım adım nasıl çalıştığını anlamak

Neden GPU? (CPU vs. GPU)

- **CPU (Latans Odaklı):**
 - Az sayıda ama çok güçlü çekirdek.
 - Karmaşık mantıksal işlemler ve seri komutlar.
- **GPU (Throughput Odaklı):**
 - Binlerce (örneğin A100'de 6912) küçük CUDA çekirdeği.
 - **Massively Parallel:** Aynı anda binlerce basit matris işlemini yapma yeteneği.
- **Derin Öğrenme İçin Anlamı:**
 - Bir katmandaki tüm nöronların hesaplamaları aynı anda (paralel) yapılabilir.



CPU

GPU

Neden Conv Layer ve GPU İkilisi?

• Yerel Bağımsızlık (Locality):

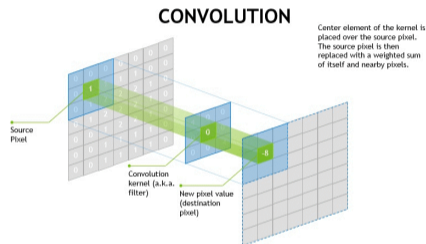
- Görüntünün sol üst köşesindeki konvolüsyon işlemi, sağ alt köşedekinden tamamen bağımsızdır.
- **GPU Sonucu:** Binlerce pencere (patch) aynı anda, farklı çekirdeklerde hesaplanabilir.

• Ağırlık Paylaşımı (Weight Sharing):

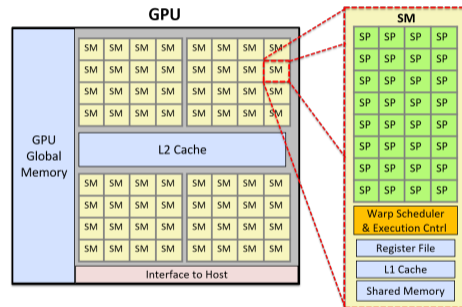
- Aynı filtre (kernel) tüm görüntü üzerinde gezdirilir.
- **GPU Sonucu:** Filtre katsayıları GPU'nun "Constant" veya "Shared" belleğine bir kez yazılır, tüm çekirdekler oradan okur.

• Matris Çarpımına Dönüşüm

- GPU'lar matris çarpım için tasarlanmış makinelerdir.



- **VRAM (Video RAM):**
 - GPU'nun ana deposudur. Model ağırlıkları ve veri batch'leri burada saklanır.
 - Kapasitesi (örn. 24GB, 80GB) model büyüklüğünü belirler.
- **SM (Streaming Multiprocessor):**
 - GPU'nun temel hesaplama birimidir. Bir GPU'da çok sayıda SM bulunur.
- **Çekirdek (SP) Tipleri**
 - **CUDA Cores:** Genel matematiksel işlemler.
 - **Tensor Cores:** Derin öğrenmeye özel, 4x4 matris çarpmak için özelleşmiş çekirdekler.



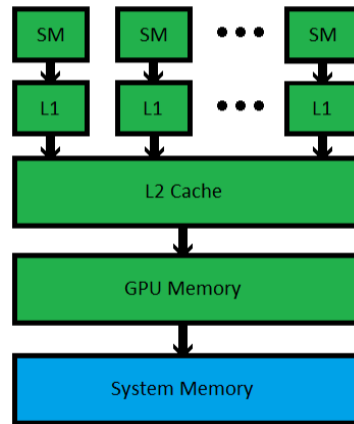
GPU Bellek Hiyerarşisi: Hız vs. Kapasite

• Hiyerarşik Yapı (Piramit):

- **Registers (Yazmaçlar):** En hızlı, çekirdeğin tam içinde (birkaç bayt).
- **L1 / Shared Memory:** SM içinde, çok düşük gecikme (birkaç KB).
- **L2 Cache:** Tüm SM'ler için ortak tampon (birkaç MB).
- **VRAM (Global Memory):** Ana depo, en yavaş ama en büyük (GB'larca).

• Altın Kural: Veri Hareketi Maliyetlidir

- Hesaplama hızı, bellekten veri çekme hızından kat kat fazladır.
- **Memory Bound:** Çekirdeklerin veri beklediği durum.



GPU İşleyişi: SM, Core ve Thread

- **Thread (İş Parçacığı):**
 - En küçük iş birimi. (Örn: Bir pikselin çarpım işlemi).
 - Binlerce thread, **Thread Block**'lar halinde gruplanır.
- **SM (Streaming Multiprocessor):**
 - Donanımsal "Atölye". Her SM, bir veya birden fazla Thread Block'u aynı anda misafir eder.
 - **Shared Memory** bu atölyenin içindeki ortak tezgahdır.
- **CUDA Core (Çekirdek):**
 - SM içindeki "İşçiler". Bir SM'de onlarca çekirdek bulunur.

figs/S12 - GPU_Hierarchy_Threads

Bir Model GPU'da Nasıl Başlar?

- **1. Adım: Modelin Transferi**
 - Modelin ağırlıkları (weights) CPU RAM'den GPU VRAM'ine kopyalanır. (`model.to('cuda')`)
- **2. Adım: Veri Hazırlığı**
 - Girdi batch'i GPU belleğine taşınır.
- **3. Adım: Kernel Launch (Çekirdek Başlatma)**
 - CPU, GPU'ya bir "komut dizisi" gönderir.
 - **Kritik:** Bu transferler zaman alır.

