Adam: Adaptif Moment Tahmini ile Derin Öğrenme Modellerini Optimize Etmek

1. Giriş: Optimizasyonun Derin Öğrenmedeki Yeri

Derin öğrenme modelleri, en basit tanımıyla, verilerdeki karmaşık desenleri öğrenmek için tasarlanmış matematiksel fonksiyonlar bütünüdür. Ancak bir modelin "öğrenmesi" tam olarak ne anlama gelir? Bu sorunun cevabı, optimizasyon sürecinin kalbinde yatar. Optimizasyon, bir modelin tahminlerini olabildiğince doğru hale getirmek için kendi iç parametrelerini sistematik olarak ayarlama işlemidir.

Maliyet Fonksiyonu: Modelin Karnesi

Bir modelin ne kadar "iyi" veya "kötü" performans gösterdiğini ölçmek için bir yola ihtiyacımız vardır. Bu ölçüm aracına **maliyet fonksiyonu** (loss function veya cost function) denir. Maliyet fonksiyonu, modelin tahminleri ile gerçek değerler arasındaki farkı, yani "hatayı" hesaplar. Yüksek bir maliyet değeri, modelin büyük hatalar yaptığını; düşük bir maliyet değeri ise tahminlerinin gerçeğe oldukça yakın olduğunu gösterir.

Dolayısıyla, bir modeli eğitmenin temel amacı, bu maliyet fonksiyonunun değerini mümkün olan en düşük seviyeye indirmektir. Bu minimum noktayı bulmamızı sağlayan algoritmalara ise **optimizasyon algoritmaları** denir.

Temel Başlangıç Noktası: Gradyan İnişi (Gradient Descent)

Optimizasyon algoritmalarının en temeli ve en bilineni **Gradyan İnişi** (Gradient Descent) algoritmasıdır. Bu algoritmanın çalışma mantığını basit bir analoji ile anlayabiliriz:

Gözleriniz bağlı bir şekilde dağlık bir arazide olduğunuzu ve vadinin en alçak noktasına inmeye çalıştığınızı hayal edin. Etrafınızı göremediğiniz için yapabileceğiniz en mantıklı şey, ayağınızın altındaki eğimi hissetmek ve en dik yokuş aşağı yöne doğru küçük bir adım atmaktır. Bu işlemi tekrar tekrar yaptığınızda, en sonunda vadinin tabanına ulaşırsınız.

Bu analojide:

- **Dağlık arazi**, maliyet fonksiyonunun oluşturduğu yüzeydir.
- **Bulunduğunuz konum**, modelin o anki parametre (ağırlıklar ve bias'lar) değerleridir.
- **Vadinin en alçak noktası**, ulaşmak istediğimiz minimum maliyet değeridir.

- **Ayağınızla hissettiğiniz en dik yokuş aşağı yön** ise maliyet fonksiyonunun gradyanı'dır. Gradyan, bir fonksiyonun en hızlı artış yönünü gösteren bir vektördür. Bu yüzden biz de tam tersi yönde, yani negatif gradyan yönünde ilerleyerek fonksiyonu minimize etmeye çalışırız.
- **Attığınız her adımın büyüklüğü** ise öğrenme oranı (learning rate) olarak bilinen bir hiperparametredir.

Gradyan İnişi, maliyeti düşürmek için parametreleri sürekli olarak negatif gradyan yönünde günceller. Ancak bu basit yaklaşım, özellikle karmaşık derin öğrenme modellerinde bazı zorluklarla karşılaşır. Bu zorluklar, Adam gibi daha gelişmiş optimizasyon algoritmalarının doğmasına zemin hazırlamıştır.

2. Klasik Optimizasyon Algoritmalarının Zorlukları

Gradyan İnişi, teoride oldukça mantıklı bir yaklaşım sunsa da, özellikle on binlerce hatta milyonlarca parametreye sahip olan modern derin öğrenme modellerinin karmaşık yapısında yetersiz kaldığı durumlar ortaya çıkar. Bu zorlukları anlamak, Adam gibi adaptif algoritmaların getirdiği çözümlerin değerini kavramamızı sağlar.

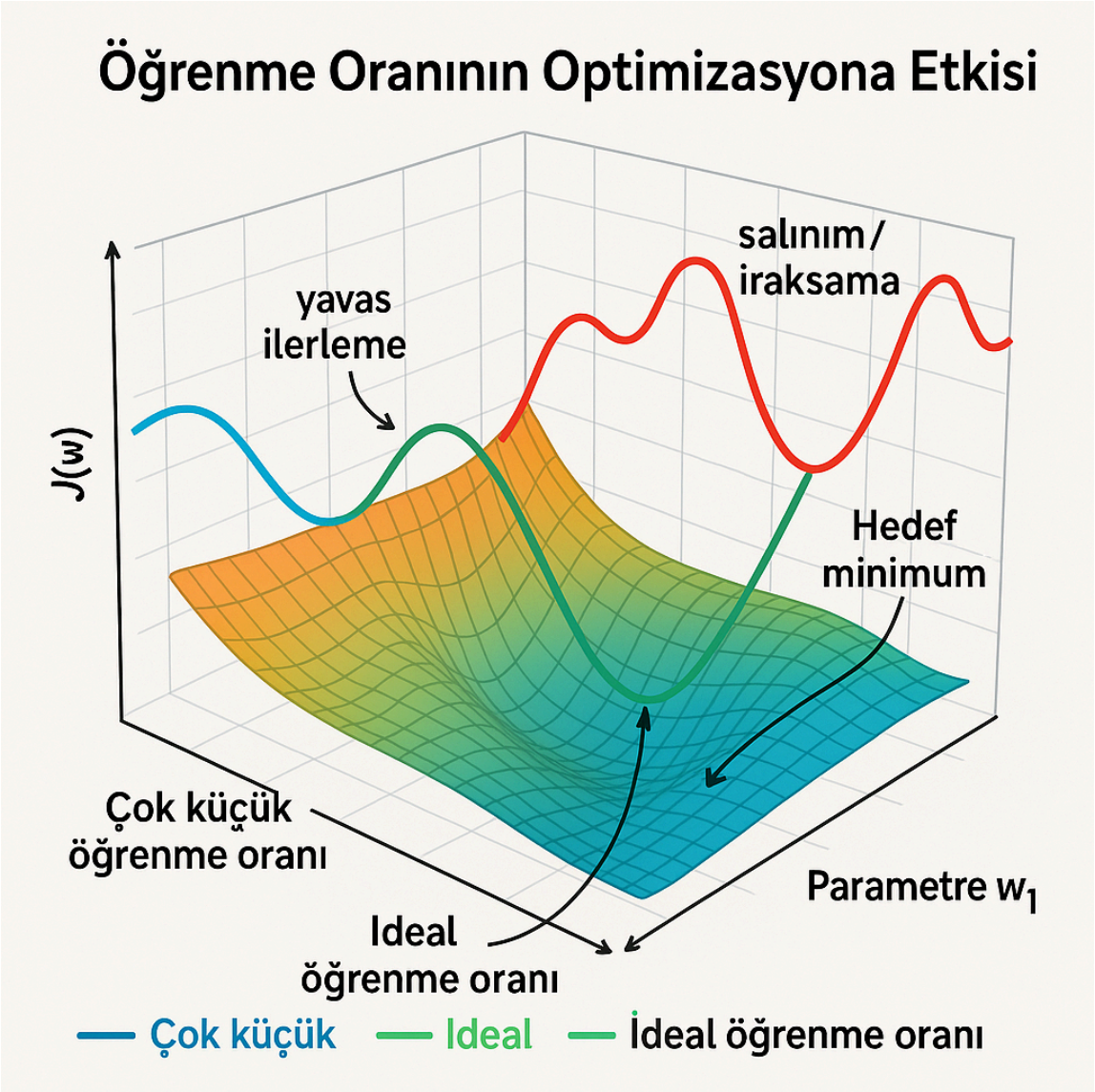
İşte en sık karşılaşılan zorluklar:

Öğrenme Oranının (Learning Rate) İkilemi

Önceki bölümde bahsettiğimiz gibi, öğrenme oranı, her adımda ne kadar "büyük" bir adım atacağımızı belirler. Bu parametrenin doğru seçimi kritik öneme sahiptir ve genellikle bir ikilem yaratır:

- **Çok Küçük Öğrenme Oranı**: Eğer adımlarımız çok küçük olursa, modelin öğrenme süreci aşırı yavaşlar. Minimum noktaya ulaşmak çok uzun sürebilir ve büyük veri setlerinde bu durum, eğitimin günler hatta haftalar sürmesi anlamına gelebilir.
- **Çok Büyük Öğrenme Oranı**: Eğer adımlarımız çok büyük olursa, minimum noktanın etrafında salınım yapmaya başlarız. Analojimize dönersek, vadinin en alçak noktasına yaklaşmak yerine, attığımız büyük adımlarla vadinin bir yakasından diğerine savruluruz. Bu durumda algoritma yakınsamakta zorlanır ve hatta maliyet fonksiyonu zamanla artarak tamamen dağılabilir (divergence).

İdeal öğrenme oranını bulmak, genellikle deneme-yanılma gerektiren zorlu bir süreçtir ve her problem için farklı bir değere ihtiyaç duyulabilir.



Yerel Minimumlar ve Eyer Noktaları (Local Minima and Saddle Points)

Derin öğrenme modellerinin maliyet yüzeyi, basit bir çanak gibi pürüzsüz değildir. Aksine, çok sayıda tepe, vadi ve düzlük içeren oldukça engebeli bir araziye benzer.

- **Yerel Minimum (Local Minimum)**: Gradyan İnişi, en yakınındaki vadiye (minimum noktaya) doğru ilerler. Ancak bu vadi, arazideki en alçak nokta, yani global minimum olmayabilir. Algoritma bir yerel minimuma ulaştığında, gradyan sıfır olacağı için daha fazla ilerleyemez ve oraya takılıp kalır. Bu, modelin potansiyel olarak ulaşabileceği en iyi performansa ulaşamaması anlamına gelir.
- **Eyer Noktası (Saddle Point)**: Özellikle çok boyutlu (çok parametreli) modellerde yerel minimumlardan daha sık rastlanan bir durumdur. Eyer noktası, bazı yönlerde bir minimum gibi görünürken, diğer yönlerde bir maksimum gibi görünen düz bir bölgedir. Bu noktalarda da gradyan sıfıra çok yakındır. Bu durum, algoritmanın

öğrenme hızının neredeyse durma noktasına gelmesine neden olur ve eğitimi ciddi şekilde yavaşlatır.

Tüm Parametreler İçin Tek Bir Öğrenme Oranı

Standart Gradyan İnişi, modeldeki tüm parametreler (ağırlıklar) için aynı öğrenme oranını kullanır. Ancak, her parametrenin güncellenme ihtiyacı aynı olmayabilir. Örneğin, bazı parametreler zaten ideale yakın bir değerde olabilir ve sadece küçük ayarlamalara ihtiyaç duyarken, diğerleri çok daha büyük adımlarla güncellenmelidir. Özellikle veride seyrek olarak bulunan özelliklerle ilişkili parametrelerin, sık görülen özelliklere göre farklı şekilde güncellenmesi gerekebilir. Herkese aynı "ayakkabı numarasını" vermeye çalışmak gibi, tek bir öğrenme oranı genellikle verimsizdir.

Bu temel zorluklar, araştırmacıları şu sorulara yöneltti: Öğrenme oranını her bir parametre için ayrı ayrı ve dinamik olarak ayarlayabilir miyiz? Yerel minimumlardan ve eyer noktalarından daha verimli bir şekilde kaçınabilir miyiz? Bu sorulara cevap arayışı, Momentum, RMSprop ve nihayetinde Adam gibi adaptif optimizasyon algoritmalarının geliştirilmesine yol açmıştır.

3. Adam'ın Ataları: Momentum ve RMSprop

Adam (Adaptive Moment Estimation) algoritması, isminden de anlaşılacağı gibi, kendisinden önce geliştirilen iki güçlü fikrin birleşiminden doğmuştur: **Momentum** ve **RMSprop**. Gradyan İnişi'nin zorluklarına farklı çözümler sunan bu iki yöntemi anlamak, Adam'ın neden bu kadar etkili olduğunu kavramanın anahtarıdır.

Momentum: İvme Kazanarak Engelleri Aşmak

Standart Gradyan İnişi'nin en büyük sorunlarından biri, her adımı bir öncekinden tamamen bağımsız olarak atmasıdır. Bu durum, özellikle dar ve uzun vadilerde (maliyet yüzeyinde) ilerlerken verimsiz salınımlara veya eyer noktalarında yavaşlamaya neden olur.

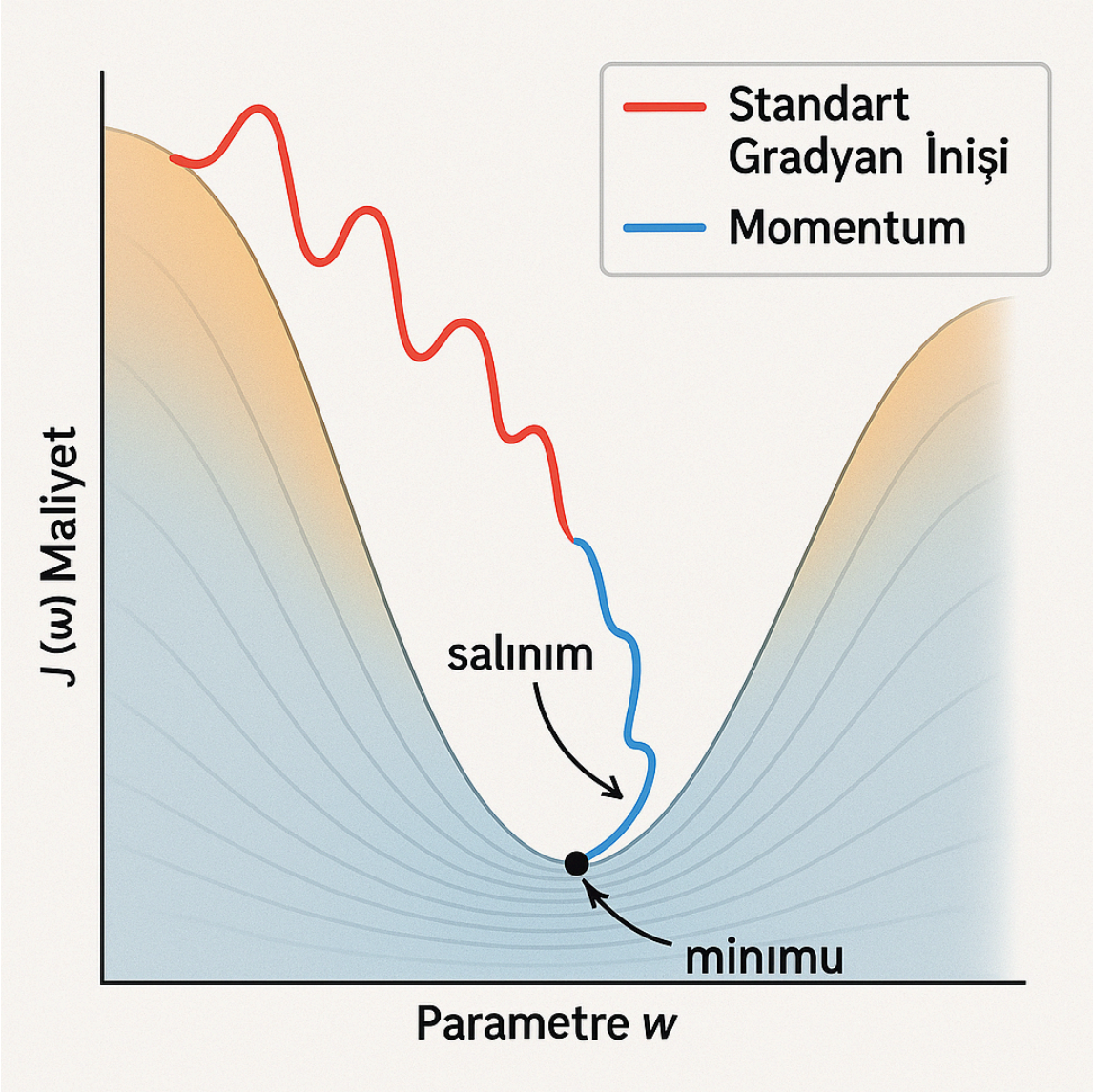
Momentum yöntemi, bu soruna fiziksel dünyadan bir ilhamla çözüm getirir: ivme. Yokuş aşağı yuvarlanan bir topu düşünün. Top, sadece o anki eğime göre değil, aynı zamanda o ana kadar biriktirdiği hıza (momentuma) göre de hareket eder. Bu birikmiş hız, topun küçük tümsekleri veya düzlükleri kolayca aşmasını sağlar.

Momentum optimizasyonda da tam olarak bu fikri uygular:

- Sadece mevcut gradyanı değil, aynı zamanda geçmiş adımlardaki gradyanların birikimli bir ortalamasını da hesaba katar. Bu birikimli ortalamaya "hız" (velocity) denir.

- Parametre güncellemesi, mevcut gradyan yönü ile bu "hız" vektörünün birleşimiyle yapılır.

Bu sayede, eğer gradyan adımları sürekli aynı yönde ilerliyorsa, hız artar ve algoritma o yönde ivmelenir. Eğer gradyanın yönü sürekli değişiyorsa (salınım durumu), zıt yönlerdeki güncellemeler birbirini sönümler ve salınım azalır. Bu yaklaşım, algoritmanın eyer noktalarından ve sığ yerel minimumlardan "hızını alarak" kurtulmasına yardımcı olur.



RMSprop: Her Parametreye Özel Öğrenme Oranı

Daha önce bahsettiğimiz gibi, tüm parametreler için tek bir global öğrenme oranı kullanmak verimsizdir. **RMSprop (Root Mean Square Propagation)**, bu sorunu çözmek için her bir parametreye özel, adapte olabilen bir öğrenme oranı mekanizması sunar.

RMSprop'un arkasındaki temel fikir şudur: Bir parametrenin gradyanı yakın geçmişte sürekli büyük değerler alıyorsa, muhtemelen çok fazla salınım yapıyordur, bu yüzden

onun öğrenme oranını (adım büyüklüğünü) azaltmalıyız. Eğer gradyanı sürekli küçükse, muhtemelen düz bir platoda yavaş ilerliyordur, bu yüzden onun öğrenme oranını artırarak hareketi hızlandırmalıyız.

Bunu başarmak için:

- Her bir parametre için geçmiş gradyanların karelerinin birikimli bir ortalamasını tutar.
- Parametre güncellemesi yapılırken, global öğrenme oranı bu ortalamanın karekköküne bölünür.

Sonuç olarak:

- Geçmişte büyük gradyanlara sahip bir parametre için payda büyüyecek ve etkin öğrenme oranı küçülecektir (adım yavaşlar).
- Geçmişte küçük gradyanlara sahip bir parametre için payda küçülecek ve etkin öğrenme oranı büyüyecektir (adım hızlanır).

Bu sayede RMSprop, her parametrenin kendi dinamiklerine göre öğrenme oranını otomatik olarak ayarlar.

Kısacası, Momentum gradyan güncellemelerine "ivme" katarken, RMSprop bu güncellemelerin "büyüklüğünü" her parametre için akıllıca ayarlar. Bir sonraki bölümde göreceğimiz gibi Adam, bu iki güçlü ve birbirini tamamlayan fikri tek bir optimizasyon algoritmasında birleştirir.

4. Adam (Adaptive Moment Estimation) Algoritması Derinlemesine Bakış

Önceki bölümlerde Gradyan İnişi'nin zorluklarını ve bu zorluklara çözüm olarak geliştirilen Momentum ve RMSprop yöntemlerini inceledik. Adam, bu iki güçlü yaklaşımı tek ve etkili bir çerçevede birleştirir. Adı, "Adaptif Moment Tahmini" anlamına gelen "Adaptive Moment Estimation" ifadesinin kısaltmasıdır ve bu isim, çalışma şeklini mükemmel bir şekilde özetler.

Adam, her bir parametre için öğrenme oranını ayrı ayrı adapte etmek üzere gradyanların hem birinci moment (ortalama) hem de ikinci moment (varyansın merkezileştirilmemiş hali) tahminlerini kullanır.

İki Farklı Momentin Gücü

Algoritmanın temelinde, geçmiş gradyanlardan gelen bilgiyi biriktiren iki ana değişken bulunur:

Birinci Moment (Ortalama, m_t): Bu, Momentum yöntemine benzer şekilde çalışır. Gradyanların üssel olarak azalan bir hareketli ortalamasını tutar. Bu değer, gradyanların yönü ve ivmesi hakkında bir fikir verir. Yani, "hız" vektörümüzdür.

İkinci Moment (Varyans, v_t): Bu ise RMSprop yöntemine benzer şekilde çalışır. Gradyanların karesinin üssel olarak azalan bir hareketli ortalamasını tutar. Bu değer, gradyanların büyüklüğünün (ne kadar değişken olduğunun) bir ölçüsüdür ve öğrenme oranını adapte etmek için kullanılır.

Algoritmanın Adımları ve Formülleri

Adam algoritması, her bir eğitim adımında (iterasyonda) aşağıdaki güncellemeleri gerçekleştirir. Öncelikle algoritmanın kontrol ettiği hiperparametreleri tanıyalım:

- (α): Öğrenme oranı (learning rate). Adım büyüklüğünü belirler. Genellikle `0.001` olarak başlanır.
- (β_1): Birinci moment için bozunma oranı. Momentum'un ne kadarının korunacağını belirler. Genellikle `0.9`.
- (β_2): İkinci moment için bozunma oranı. Geçmiş gradyan karelerinin ne kadarının hatırlanacağını belirler. Genellikle `0.999`.
- (ϵ): Sayısal kararlılık için kullanılan, sıfıra bölmeyi önleyen çok küçük bir sayı. Genellikle `1e-8`.

`t` anındaki bir iterasyonda, maliyet fonksiyonunun gradyanı (g_t) olarak hesaplandıktan sonra:

1. **Birinci moment (m_t) güncellenir:**

$$m_t = \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$$

2. **İkinci moment (v_t) güncellenir:**

$$v_t = \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$$

Sapma Düzeltmesi (Bias Correction): Adaleti Sağlamak

Yukarıdaki formüllerde bir sorun vardır. (m_0) ve (v_0) vektörleri sıfır olarak başlatıldığı için, eğitimin ilk adımlarında bu moment tahminleri sıfıra doğru yanlı (biased) olur. Özellikle (β_1) ve (β_2) değerleri 1'e yakın olduğunda bu etki daha belirgindir.

Adam'ın geliştiricileri bu sorunu çözmek için **sapma düzeltmesi** (bias correction) adımı eklemişlerdir. Bu adım, eğitimin başındaki yanlılığı giderir ve sonrasında etkisi azalarak kaybolur.

1. **Birinci moment için sapma düzeltmesi yapılır:**

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$

2. **İkinci moment için sapma düzeltmesi yapılır:**

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

Burada `t` iterasyon sayısıdır. `t` arttıkça, (β_1^t) ve (β_2^t) terimleri sıfıra yaklaşır; böylece bu düzeltmenin etkisi zamanla ortadan kalkar.

Nihai Güncelleme Kuralı

Tüm bu hesaplamalar yapıldıktan sonra, modelin parametreleri ((θ)) nihayet güncellenir:

5. **Parametreler güncellenir:**

$$\theta_t = \theta_{t-1} - \alpha \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

Bu son denklemi yorumlayalım: Parametreleri ((θ_{t-1})) güncellerken, öğrenme oranı (α) ile ölçeklendirilmiş bir adım atıyoruz. Adımın yönünü, düzeltilmiş birinci moment (yani momentum) ($\hat{m}_t$) belirliyor. Adımın büyüklüğüise parametreye özel olarak, düzeltilmiş ikinci momentin karekökü ($\sqrt{\hat{v}_t}$) ile bölünerek adapte ediliyor. Bu, RMSprop'tan gelen fikrin ta kendisidir.

Sonuç olarak Adam, Momentum'un hızlandırıcı etkisini ve RMSprop'un parametreye özel adaptif öğrenme oranı yeteneğini, başlangıçtaki yanlılığı düzelten bir mekanizma ile birleştirerek güçlü ve genel amaçlı bir optimizasyon algoritması ortaya koyar.

5. Adam'ın Avantajları ve Dezavantajları

Önceki bölümde Adam'ın matematiksel altyapısını ve çalışma mekanizmasını detaylıca inceledik. Peki bu algoritmayı pratikte bu kadar popüler ve güçlü kılan nedir? Her araç gibi Adam'ın da mükemmel olmadığını ve bazı durumlarda dikkatli kullanılması gerektiğini bilmek önemlidir.

Adam'ın Avantajları

Adam'ın derin öğrenme topluluğunda varsayılan (default) optimizasyon algoritması olarak kabul edilmesinin arkasında yatan birçok güçlü neden vardır:

1. Hızlı ve Etkili Yakınsama: Momentum'un ivmelenme yeteneği ile RMSprop'un adaptif öğrenme oranını birleştirmesi sayesinde, Adam genellikle minimum noktaya diğer optimizasyon algoritmalarından çok daha hızlı bir şekilde yakınsar. Bu, model eğitim sürelerini önemli ölçüde kısaltır.

2. Adaptif Öğrenme Oranı: Belki de en büyük avantajı budur. Her bir model parametresi için öğrenme oranını otomatik olarak ayarlar. Bu, bir veri bilimcinin en iyi öğrenme oranını bulmak için harcadığı zaman ve emekten tasarruf etmesini sağlar. Genellikle varsayılan hiperparametre değerleri ($\alpha=0.001$, $\beta_1=0.9$, $\beta_2=0.999$) çok çeşitli problemler için şaşırtıcı derecede iyi çalışır.

3. Bellek Verimliliği: Adam, algoritma karmaşıklığına rağmen bellek açısından oldukça verimlidir. Sadece her parametre için birinci ve ikinci moment tahminlerini (iki ek vektör) bellekte tutması yeterlidir. Bu, milyonlarca parametreye sahip devasa modeller için bile uygulanabilir olmasını sağlar.

4. Büyük Veri ve Parametreler İçin Uygunluk: Algoritmanın adım adım (iteratif) doğası ve verimliliği, onu çok büyük veri setleri ve yüksek boyutlu parametre uzayları (karmaşık sinir ağları) için son derece uygun hale getirir. Ayrıca, seyrek gradyanlara sahip problemlerle de (örneğin doğal dil işleme) iyi başa çıkar.

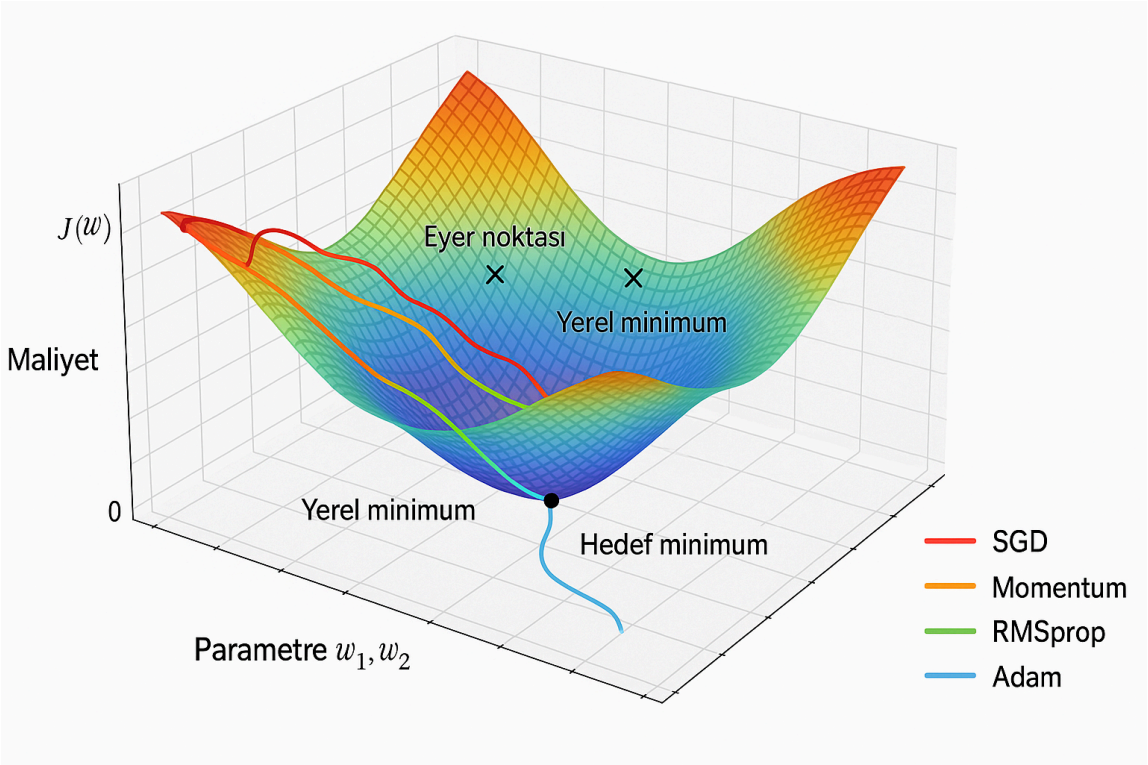
Dezavantajları ve Dikkat Edilmesi Gerekenler

Adam, birçok avantajına rağmen her derde deva bir çözüm değildir. Kullanırken akılda tutulması gereken bazı potansiyel zayıflıkları şunlardır:

1. Genelleme Performansı Üzerine Tartışmalar: Literatürdeki en yaygın eleştirilerden biri budur. Bazı araştırmalar, Adam'ın eğitim verisinde çok hızlı bir şekilde çok iyi bir sonuca (düşük maliyet değerine) ulaşmasına rağmen, modelin test verisi (daha önce görmediği veri) üzerindeki genelleme performansının, bazen dikkatlice ayarlanmış bir SGD+Momentum (Stokastik Gradyan İnişi ile Momentum) kadar iyi olmayabileceğini göstermiştir. Teoriye göre Adam, "keskin" minimumlara yakınsarken, SGD'nin "düz" minimumları bulma eğiliminde olması ve bu düz minimumların daha iyi genelleme yapması bu duruma neden olabilir.

2. Yakınsama Sorunları: Çok nadir de olsa, bazı spesifik ve karmaşık optimizasyon problemlerinde Adam'ın doğru çözüme yakınsamadığı, buna karşın daha basit algoritmaların yakınsadığı durumlar gözlemlenmiştir.

Sonuç olarak, Adam'ın sunduğu hız ve kullanım kolaylığı, onu çoğu derin öğrenme projesi için mükemmel bir başlangıç noktası ve genellikle de son durak yapar. Ancak, bir modelin performansının sınırlarını zorlarken veya genelleme konusunda sorunlar yaşandığında, SGD+Momentum gibi daha klasik bir algoritmayı denemek ve sonuçları karşılaştırmak değerli bir pratik olabilir.



6. Pratik Uygulama: Python ile Kod Örneği

Teoride Adam algoritmasının adımları karmaşık görünebilir, ancak pratikte Keras gibi yüksek seviyeli kütüphaneler sayesinde kullanımı son derece kolaydır. Şimdi, basit bir sinir ağı modelini eğitirken Adam'ı nasıl kullanacağımızı görelim.

Bu örnekte, sentetik bir veri seti oluşturup bu veriyi sınıflandırmak için küçük bir sinir ağı modeli kuracağız ve modelimizi **adam** optimizasyon algoritması ile derleyeceğiz.

In [ ]:
```
# 1. Gerekli kütüphaneleri içe aktarma
import tensorflow as tf
from tensorflow import keras
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
import numpy as np

# 2. Örnek bir veri seti oluşturma
# 1000 örnek, 20 öznitelik ve 2 sınıflı bir veri seti oluşturalım.
X, y = make_classification(n_samples=1000, n_features=20, n_informative=10,

# Veriyi eğitim ve test setlerine ayırma
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rar

# 3. Basit bir sinir ağı modeli tanımlama
model = keras.Sequential([
    keras.layers.Dense(32, activation='relu', input_shape=(20,)),
    keras.layers.Dense(16, activation='relu'),
    keras.layers.Dense(1, activation='sigmoid') # İkili sınıflandırma için s
])
```

```
# 4. Modeli derleme: Adam optimizasyon algoritmasını burada belirtiyoruz
# Adam'ı kullanmak için optimizer parametresine 'adam' stringini vermek yete
model.compile(optimizer='adam',
              loss='binary_crossentropy', # İkili sınıflandırma için kayıp f
              metrics=['accuracy'])

# Modelin özetini görüntüleme
model.summary()

# 5. Modeli eğitme
print("\nModel Eğitimi Başlıyor...")
history = model.fit(X_train, y_train, epochs=20, batch_size=32, validation_s
print("Model Eğitimi Tamamlandı.")

# 6. Modelin performansını test verisi üzerinde değerlendirme
print("\nModel Test Verisi Üzerinde Değerlendiriliyor...")
loss, accuracy = model.evaluate(X_test, y_test, verbose=0)
print(f"Test Kaybı (Loss): {loss:.4f}")
print(f"Test Doğruluğu (Accuracy): {accuracy:.4f}")
```

Gördüğünüz gibi, Adam algoritmasının tüm gücünü projemize dahil etmek için yapmamız gereken tek şey, `model.compile()` fonksiyonu içerisinde `optimizer='adam'` parametresini belirtmektir. Keras, arka planda birinci ve ikinci momentlerin hesaplanması, sapma düzeltmesi ve parametrelerin güncellenmesi gibi tüm karmaşık işlemleri bizim için otomatik olarak yönetir.

Eğer Adam'ın öğrenme oranı gibi hiperparametrelerini özelleştirmek isterseniz, string yerine bir optimizer nesnesi de kullanabilirsiniz. Örneğin, öğrenme oranını `0.0001` olarak değiştirmek için:

In [ ]:
```
# Alternatif kullanım: Hiperparametreleri özelleştirme
# from tensorflow.keras.optimizers import Adam
# custom_optimizer = Adam(learning_rate=0.0001)
# model.compile(optimizer=custom_optimizer, ...)
```

Bu pratik kullanım kolaylığı, Adam'ın neden derin öğrenme uygulamalarında bu kadar yaygın bir şekilde benimsendiğinin en büyük kanıtlarından biridir.

7. Sonuç

Bu makalede, modern derin öğrenmenin en temel optimizasyon algoritmalarından biri olan Adam'ı derinlemesine inceledik. Yolculuğumuza, optimizasyonun temel amacı olan maliyet fonksiyonunu minimize etme hedefi ve bu hedefe ulaşmak için kullanılan en temel yöntem olan Gradyan İnişi ile başladık. Ardından, bu klasik yaklaşımın yavaş yakınsama, yerel minimumlara takılma ve sabit öğrenme oranı gibi zorluklarını gördük.

Adam algoritmasının dehası, bu zorluklara çözüm olarak geliştirilen iki güçlü fikri — Momentum'un ivme kazandıran yaklaşımını ve RMSprop'un her parametreye özel adaptif

öğrenme oranı mekanizmasını — tek bir cohesive çerçevede birleştirmesinden gelir. Gradyanların hem birinci hem de ikinci moment tahminlerini kullanarak ve bu tahminleri başlangıçtaki yanlılık için düzelterek, Adam, karmaşık ve yüksek boyutlu maliyet yüzeylerinde verimli bir şekilde gezinebilen son derece etkili bir optimizatör haline gelmiştir.

Pratikteki kullanım kolaylığı ve genellikle varsayılan hiperparametrelerle bile sunduğu güçlü performans, Adam'ı sayısız derin öğrenme projesi için fiili standart ve mükemmel bir başlangıç noktası yapmıştır. Hız, bellek verimliliği ve adaptif doğası arasındaki denge, onu veri bilimcilerin ve makine öğrenmesi mühendislerinin araç setinde vazgeçilmez bir araç konumuna getirmiştir. Sonuç olarak Adam, sadece bir optimizasyon algoritması olmanın ötesinde, derin öğrenme alanındaki ilerlemeyi önemli ölçüde hızlandıran temel yapı taşlarından biri olarak yerini sağlamlaştırmıştır.

Kaynakça

- Kingma, Diederik P., and Jimmy Ba. "Adam: A Method for Stochastic Optimization". arXiv:1412.6980, 2014.
- Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. Deep Learning. MIT Press, 2016.
- Ruder, Sebastian. "An overview of gradient descent optimization algorithms". ruder.io, 2016.
- Stanford University. "CS231n: Neural Networks Part 3: Learning and Evaluation". cs231n.github.io, 2018.
- Orta Doğu Teknik Üniversitesi. "CENG 782: Deep Learning - Lecture 4: Optimization". ODTÜ Açık Ders Malzemeleri, 2021.
- Gezici, Gökhan. "Makine Öğrenmesinde Optimizasyon ve Optimizasyon Algoritmaları". Veri Bilimi Okulu, 2020.