

Finite sum of functions

- In ML optimization, objective function has the form:

$$f(x) = \frac{1}{n} \sum_{i=1}^n f_i(x)$$

n : training set size

$f_i(x)$: loss value for i.th training point according to parameter x

Bu slaytta, makine öğrenimi optimizasyonunda kullanılan bir hedef fonksiyonun formu anlatılmaktadır. Fonksiyon, eğitim setindeki her bir öğrenme noktasının kaybını temsil eden fonksiyonların bir ortalaması olarak ifade edilir. İşte detaylar:

- $f(x)$: Hedef fonksiyon. Bu fonksiyon, optimizasyon sırasında minimize edilmeye çalışılan bir değerdir.
- n : Eğitim setindeki örneklerin (veri noktalarının) sayısıdır.
- $f_i(x)$: i . eğitim noktası için hesaplanan kayıp (loss) değeridir. Burada x , modelin parametrelerini ifade eder.

Özetle, $f(x)$ fonksiyonu, tüm eğitim verileri üzerinden hesaplanan kayıp değerlerinin ortalamasını alarak bir genel performans ölçütü sunar. Bu, modelin eğitim verileri üzerinde ne kadar iyi performans gösterdiğini anlamak için kullanılır. Makine öğreniminde genellikle bu tür bir ortalama kayıp fonksiyonu, modelin parametrelerini ayarlamak için bir kılavuz olarak kullanılır.

Some examples of finite sum

- Training set: $i=1:n$ a_i : i .th input, b_i : i .th output
- x : parameters of a model to be trained
- Least squares:

$$\min_x \frac{1}{n} \sum_{i=1}^n (a_i^\top x - b_i)^2 = \min_x \frac{1}{n} \sum_{i=1}^n (f_i(x))^2$$

- Lasso:

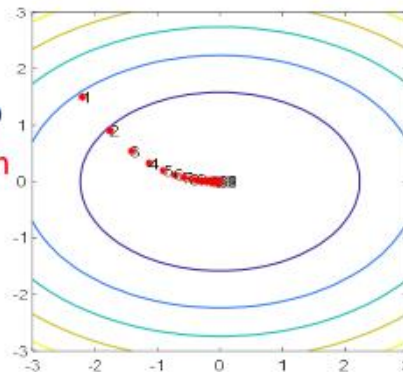
$$\min_x \frac{1}{n} \sum_{i=1}^n (a_i^\top x - b_i)^2 + \lambda \sum_{j=1}^d |x_j|$$

- ANN:

$$\min_x \frac{1}{n} \sum_{i=1}^n \text{loss}(\text{ANN}(x, a_i), b_i)$$

Gradient Descent

- $x_{t+1} = x_t - \eta_t \nabla f(x_t) = x_t - \eta_t \sum_{i=1}^n \nabla f_i(x_t)$
- $f(x_1, x_2) = x_1^2 + 2x_2^2$
- Learning rate (η_t) = 0.1
- it converges at 40.th step
- Code: [hessian_gradyan.m](#)





ChatGPT

Bu slaytlar, makine öğrenimi modellerini eğitmek için kullanılan çeşitli kayıp (loss) fonksiyonları ve bunların optimizasyonu için kullanılan bir yöntem olan Gradyan İnişi (Gradient Descent) hakkında bilgi vermektedir. İşte detaylar:

Üst Slayt:

- Eğitim seti i -inci girdi (a_i) ve çıktı (b_i) çiftlerinden oluşur.
- x : Eğitilmesi gereken modelin parametreleridir.
- En Küçük Kareler (Least Squares): Hedef fonksiyon, gözlemlenen çıktılar ile modelin tahminleri arasındaki farkların karelerinin ortalamasını alarak hesaplanır.
- Lasso: En Küçük Kareler metoduna ek olarak, modelin parametrelerinin mutlak değerlerinin toplamını da içerir, böylece modelin karmaşıklığını cezalandırır ve daha basit modeller oluşturmayı teşvik eder.
- Yapay Sinir Ağları (ANN): Hedef fonksiyon, modelin tahminleri ile gerçek değerler arasındaki kaybı, özel bir kayıp fonksiyonu (Loss) kullanarak hesaplar.

Alt Slayt:

- Gradyan İnişi: Modelin hedef fonksiyonunu minimize eden parametre değerlerini bulmak için kullanılan bir optimizasyon yöntemidir.
- $x_{t+1} = x_t - \eta \nabla F(x_t)$: Burada x_t mevcut adımdaki parametre değerleri, η öğrenme oranı ve $\nabla F(x_t)$ hedef fonksiyonunun gradyanıdır.
- $F(x_1, x_2) = x_1^2 + 2x_2^2$: Optimizasyonu yapılacak örnek bir fonksiyon.
- Öğrenme oranı (η) 0.1 olarak belirlenmiştir.
- Yöntem 40. adımda yakınsar.
- Kodun adı: hessian_gradyan.m olarak belirtilmiştir.

Slaytlar, bir optimizasyon teknikleri dersinin parçası olarak görünmektedir ve bu tekniklerin makine öğrenimi modellerini eğitmek için nasıl kullanılabileceğini örneklerle açıklamaktadır. Gradyan İnişi, modelin performansını en iyi hale getirecek şekilde parametrelerin nasıl güncelleneceğini gösteren bir yöntemdir.

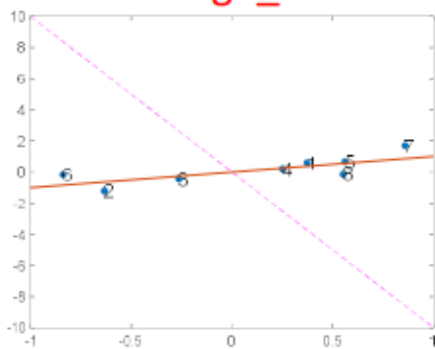


Large scale problems

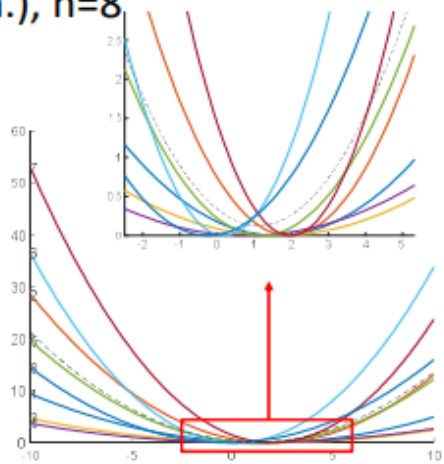
- Zillions of parametres (d)
- Zillions of data points (n)
- Each GD iteration requires $n \cdot d$ calculations

A simple model

- $\min_x \frac{1}{2n} \sum_{i=1}^n (a_i^T x - b_i)^2 = \min_x \frac{1}{2n} \sum_{i=1}^n (f_i(x))^2$
- a_i, b_i, x : scalar (1 dim.), $n=8$
- Code: `sgd_how.m`



Training points, - true model $b=1a$,
-- current model $b=-10a$



Corresponding $f_i(x)$ s = $(a_i x - b_i)^2$



ChatGPT

Bu slayt, büyük ölçekli problemlerde karşılaşılan durumları ve Gradyan İnişi (GD) yönteminin uygulanmasındaki zorlukları açıklıyor:

- "Zillions of parametres (d)": Burada "zillions" bir abartma olarak kullanılarak, çok büyük sayıda parametrenin olduğunu ifade ediyor. d ile gösterilen parametre sayısı, modelin karmaşıklığını ve boyutunu belirtir.
- "Zillions of data points (n)": Benzer şekilde, "zillions" ifadesi ile modelin eğitiminde kullanılan veri noktalarının (data points) sayısının çok fazla olduğu vurgulanıyor. n ile gösterilen veri noktası sayısı, eğitim setinin büyüklüğünü gösterir.
- "Each GD iteration requires $n*d$ calculations": Her bir Gradyan İnişi iterasyonunun, n veri noktası için her bir d parametre üzerinden hesaplamalar yapılmasını gerektirdiği anlatılıyor. Bu da, her iterasyonda n çarpı d sayıda hesaplama yapılması gerektiği anlamına gelir. Büyük ölçekli problemlerde bu, hesaplama yükünün çok büyük olabileceğini gösterir.

Kısacası, bu slayt büyük ölçekli makine öğrenimi problemlerinde, modeli eğitmek için gerekli olan hesaplama yükünün ne kadar büyük olabileceğine dikkat çekiyor.

— — — —



ChatGPT

Bu slaytta basit bir modelin nasıl kurulacağı ve eğitileceği üzerine bir örnek verilmektedir:

- Modelin amacı, $\frac{1}{n} \sum_{i=1}^n (a_i x - b_i)^2$ ifadesini minimize eden x değerini bulmaktır. Bu, verilen eğitim noktaları için en iyi tahmini yapacak olan x değerini bulmak anlamına gelir.
- a_i , b_i , ve x skaler (tek boyutlu) değerlerdir.
- n eğitim noktasının sayısını ifade eder ve burada $n = 8$ olarak belirtilmiştir.
- Kodun adı "sgd_how.m" olarak verilmiştir. Bu, Stokastik Gradyan İniş (Stochastic Gradient Descent) algoritmasını uygulayan bir MATLAB script dosyası olabilir.

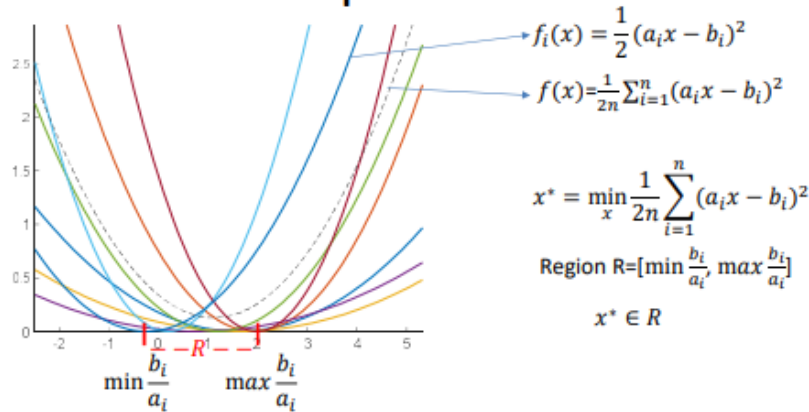
Slaytın alt kısmında iki grafik bulunmaktadır:

- Sol taraftaki grafik, eğitim noktalarını (mavi noktalar) ve modelin tahminlerini (mor çizgi) gösterir. Bu, gerçek modelin $b = 1a$ ile nasıl bir doğru çizdiğini ve şu anki modelin $b = 10a$ ile nasıl bir tahmin yaptığını gösterir.
- Sağ taraftaki grafik, farklı x değerleri için kayıp fonksiyonunun ($f(x)$) değerlerini gösterir. Kırmızı okla gösterilen yer, kayıp fonksiyonunun minimum değerine sahip olduğu x değerini işaret eder ve bu, modelin bu noktada en iyi tahminini yaptığı anlamına gelir.

Bu slayt, makine öğrenimi ve optimizasyon tekniklerinin eğitim sürecinin nasıl görselleştirilebileceğini örneklerle açıklamaktadır.



A simple model



Obverse that:

For all i , $\nabla f_i(x)$ and $\nabla f(x)$ have the same sign outside R

So, if we use $\nabla f_i(x)$, we guarantee the improvement outside R

In R , we get chaos ☹

Lets see

Stochastic Gradient Descent

- $x_{t+1} = x_t - \eta_t \nabla f_i(x_t)$
- $f(x_1, x_2) = x_1^2 + 2x_2^2$
- Learning rate (η_t) = 0.1
- it does not converge
- Code: `sgd.m`

Region R : an area (in 2 dim)

Outside R , GD and SGD are very similar

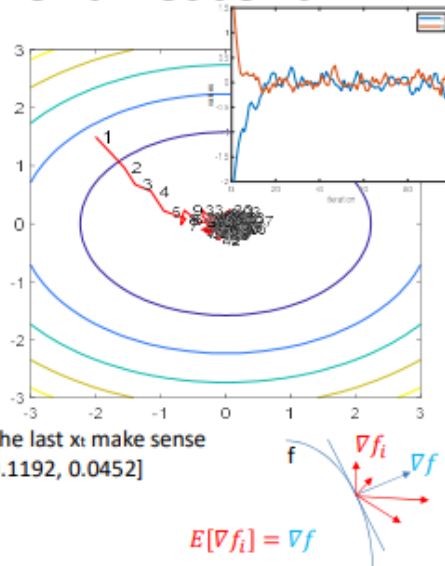
Inside R (near x^*), SGD fluctuates

Usage of $\text{mean}(x_t)$ or $\text{mean}(x_{t-k} \dots x_t)$ instead of the last x_t make sense

Here, the last $x_t = [-0.2692, 0.3260]$, $\text{mean}(x_t) = [-0.1192, 0.0452]$

Each GD iteration requires $d \cdot n$ calculations

Each SGD iteration requires d calculations ☺





ChatGPT

Bu slayt, bir optimizasyon probleminin çözümü için basit bir modelin nasıl kullanıldığını göstermektedir:

- $f(x) = \frac{1}{2}(a_i x - b_i)^2$ ifadesi, bir eğitim noktası için hesaplanan kayıp fonksiyonunu temsil eder.
- $F(x) = \frac{1}{2n} \sum_{i=1}^n (a_i x - b_i)^2$ ifadesi ise tüm eğitim noktaları için ortalama kayıp fonksiyonunu temsil eder.
- x^* sembolü, kayıp fonksiyonunu minimize eden x değerini ifade eder ve bu değer $x^* = \min_x \frac{1}{2n} \sum_{i=1}^n (a_i x - b_i)^2$ formülüyle bulunur.
- R bölgesi, a_i parametrelerinin tersinin minimumu ile b_i parametrelerinin tersinin maksimumu arasındaki aralığı ifade eder. Yani $R = [\min(\frac{b_i}{a_i}), \max(\frac{b_i}{a_i})]$.
- $x' \in R$ ifadesi, x' değerinin R bölgesinde olduğunu gösterir.
- Slaytta ayrıca, tüm a_i değerleri için $\frac{d}{dx} f(x)$ ve $F(x)$ 'in R dışında aynı işarete sahip olduğu ve bu nedenle $F(x)$ 'i kullanarak R dışındaki iyileştirmelerin garanti edilebileceği belirtilmiş. R içinde ise kaosu (belirsizliklerin) oluşabileceği ifade edilmiş.

Slayttaki grafikler, farklı a_i ve b_i değerlerine göre kayıp fonksiyonunun nasıl değiştiğini göstermektedir. Kırmızı, mavi ve yeşil çizgiler farklı parametre değerleri için kayıp fonksiyonlarını temsil ediyor. R bölgesi ve x^* değeri, bu kayıp fonksiyonlarının minimum değerlerinin nerede olduğunu göstermek için grafikte işaretlenmiş. Grafik ayrıca, belirli bir x değerindeki fonksiyon değerlerinin nasıl değişkenlik gösterebileceğini ve optimizasyon sürecinin nerede odaklanması gerektiğini de görselleştiriyor.

Örnek 1

Bu slayt, Stokastik Gradyan İnişi (Stochastic Gradient Descent - SGD) adlı bir optimizasyon algoritmasını açıklamaktadır ve bu algoritmanın uygulanışını gösteren bir örneği içermektedir:

- $x_{t+1} = x_t - \eta \nabla F(x_t)$: Bu formül, her iterasyonda parametrelerin nasıl güncellendiğini gösterir. Burada x_t , mevcut parametre değerlerini; $\nabla F(x_t)$, o parametrelerdeki kayıp fonksiyonunun gradyanını; ve η , öğrenme oranını ifade eder.
- $f(x_1, x_2) = x_1^2 + 2x_2^2$: Bu, optimizasyonu yapılan fonksiyondur.
- Öğrenme oranı (η) 0.1 olarak belirlenmiştir.
- "it does not converge": Algoritmanın bu örnek için yakınsamadığı, yani minimum bir değere ulaşmadığı belirtilmiştir.
- Kodun adı: "sgd.m" olarak verilmiştir ve bu, SGD algoritmasını uygulayan bir MATLAB script dosyası olabilir.

Slaytın alt kısmında şu bilgiler verilmiştir:

- R bölgesi, algoritmanın çalışma alanını ifade eder ve 2 boyutlu bir alanı temsil eder.
- R bölgesinin dışında SGD ve GD (Gradyan İnişi) birbirine çok benzer sonuçlar verirken, R bölgesinin içinde ve x^* 'a (optimum nokta) yakinken SGD'nin dalgalanmalar gösterdiği belirtilmiştir.
- `mean(x)` veya `mean(x...x)` kullanımının, son x değeri yerine mantıklı olduğu ifade edilmiştir ve son x değerlerinin bir örneği verilmiştir.
- Her GD iterasyonu $d \cdot n$ hesaplaması gerektirirken, her SGD iterasyonunun sadece d hesaplaması gerektirdiği vurgulanmıştır. Bu, SGD'nin her iterasyonunda tüm veri seti yerine tek bir veri noktası üzerinden hesaplama yaptığını gösterir.

Slaytta ayrıca bir grafik yer almaktadır, bu grafik SGD algoritmasının iterasyonları sırasında parametrelerin nasıl değiştiğini ve algoritmanın optimizasyon yüzeyinde nasıl hareket ettiğini görselleştirmektedir. Grafik, algoritmanın yakınsamadığı ve parametrelerin minimum nokta etrafında dalgalanmalar gösterdiğini gösterir, bu da belki algoritmanın doğru parametrelere ulaşmadığını veya öğrenme oranının çok yüksek olabileceğini gösterir.

Stabilization of SGD

- Gradient clipping[*]: rescale gradients so that their norm is at most a predetermined value (c).

if $\|\nabla f_i(x_t)\| \geq c$ then $\nabla f_i(x_t) = c \frac{\nabla f_i(x_t)}{\|\nabla f_i(x_t)\|}$

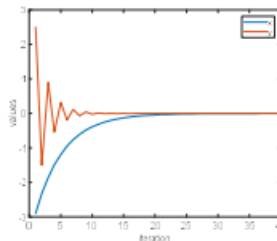
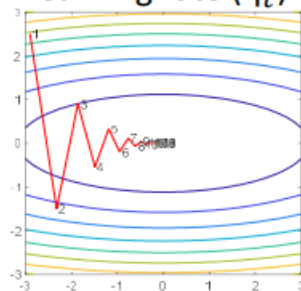
- Mini-batch: instead of one example, use k example. Reduce variance of $\nabla f_i(x_t)$

$$x_{t+1} = x_t - \frac{\eta_t}{|K_t|} \sum_{j \in K_t} \nabla f_j(x_t)$$

[*] <https://towardsdatascience.com/what-is-gradient-clipping-b8e815cdfb48>

Gradient Descent

- $f(x_1, x_2) = x_1^2 + 8x_2^2$
- Learning rate (η_t) = 0.1



GD wastes time. 2.order information helps but requires $d \times d$ calculations
There is an another way, mimic Hessian (quasi-Newton methods)
Observe all previous gradients



ChatGPT

Bu slayt, Stokastik Gradyan İnişi (SGD) algoritmasının kararlılığını artırmak için kullanılan iki yöntemi açıklamaktadır:

1. **Gradyan Kesme (Gradient Clipping):** Eğer bir gradyanın normu ($\|\nabla F(x_t)\|$) belirlenen bir eşik değerden (c) büyükse, gradyanı bu eşik değere göre yeniden ölçeklendirir. Bu, gradyanların çok büyük değerlere ulaşarak algoritmanın kararsız hale gelmesini önlemeye yardımcı olur. Yeniden ölçeklendirme işlemi, gradyanın normunu c ile bölüp, gradyan vektörüne bu değeri çarparak yapılır. Formül şöyle ifade edilir:

$$\nabla F(x_t) = \begin{cases} \nabla F(x_t) & \text{eğer } \|\nabla F(x_t)\| < c \\ c \cdot \frac{\nabla F(x_t)}{\|\nabla F(x_t)\|} & \text{eğer } \|\nabla F(x_t)\| \geq c \end{cases}$$

2. **Mini-Batch:** SGD'nin varyansını azaltmak için, tek bir örnek yerine, her adımda k sayıda örnek kullanılır. Bu, her iterasyonda gradyanların ortalamasını almak anlamına gelir ve algoritmanın daha kararlı olmasına yardımcı olur. Güncellenmiş parametreler şu formülle hesaplanır:

$$x_{t+1} = x_t - \frac{\eta_t}{|K_t|} \sum_{j \in K_t} \nabla f_j(x_t)$$

Burada K_t her adımda kullanılan örneklerin indekslerini temsil eder ve $|K_t|$ bu örneklerin sayısını ifade eder.

Slaytın alt kısmında, gradyan kesme hakkında daha fazla bilgi veren bir kaynağın linki yer almaktadır. Bu kaynak, gradyan kesme tekniğinin daha ayrıntılı bir açıklamasını içerir.



ChatGPT

Bu slayt, Gradyan İnişi (Gradient Descent - GD) algoritmasının kullanımını ve uygulandığı bir optimizasyon problemini açıklamaktadır:

- $f(x_1, x_2) = x_1^2 + 8x_2^2$: Bu, optimizasyonu yapılan iki değişkenli bir fonksiyondur. Fonksiyonun x_2 terimi x_1 terimine göre daha büyük bir ağırlığa sahip olduğu için, x_2 üzerindeki değişiklikler fonksiyon değerini daha fazla etkileyecektir.
- Öğrenme oranı (η) 0.1 olarak belirlenmiştir. Bu, her iterasyonda parametrelerin ne kadar değişeceğini belirleyen bir faktördür.
- Sol taraftaki grafik, optimizasyonun kontur haritasını ve GD algoritmasının parametre uzayında nasıl hareket ettiğini gösterir. Kırmızı oklar algoritmanın adımlarını temsil eder ve en düşük noktaya (minimuma) doğru ilerlemeyi gösterir.
- Sağ taraftaki grafik, fonksiyonun gradyanının nasıl azaldığını gösterir. Bu grafik, algoritmanın ilerleyişinin hızını ve her adımda gradyanın ne kadar değiştiğini gösterir.

Slaytta ayrıca şu bilgiler yer alır:

- Gradyan İnişi, ikinci dereceden bilgiyi (ikinci türevler) kullanmadığı için zaman kaybı olabilir. İkinci dereceden bilgi, yani Hessian matrisi kullanımı, optimizasyonun daha hızlı ilerlemesine yardımcı olabilir, fakat bu d^4 hesaplamalar gerektirir ki bu da çok büyük boyutlu problemlerde hesaplama yükünü artırır.
- Quasi-Newton yöntemleri gibi başka yöntemler de mevcuttur. Bu yöntemler, Hessian matrisinin bir taklitini kullanarak GD'nin verimliliğini artırmayı hedefler.
- Ayrıca, algoritmanın tüm önceki gradyanları gözlemlediği ve bu bilgiyi kullanarak daha etkili adımlar atabileceği belirtilmiştir.

Bu slayt, GD algoritmasının temel işleyişi ve potansiyel iyileştirmeleri hakkında bilgi sağlamaktadır.

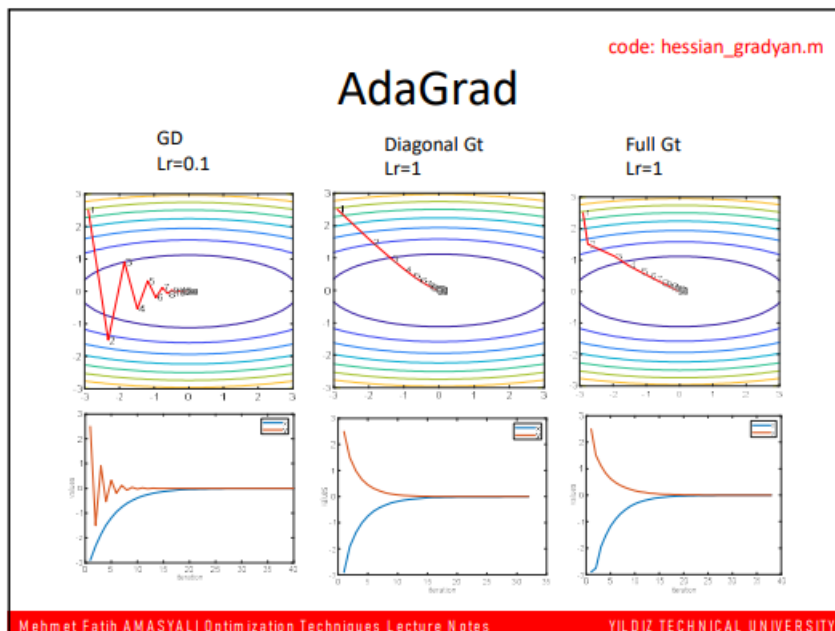
AdaGrad[*]: A quasi-Newton method

- $G_t = \sum_{j=1}^t \nabla f(x_j) \nabla f(x_j)^T$
- G_t (sum of outer product of all previous gradients) mimic Hessian
- G_t : $d \times d$ matrix
- 2 versions:
 - $x_{t+1} = x_t - \eta_t G_t^{-1/2} \nabla f(x_t)$
Full matrix inversion and square root requires $d \times d$ calculations
 - $x_{t+1} = x_t - \eta_t \text{diag}(G_t)^{-1/2} \nabla f(x_t)$
Diagonal matrix (vector in d dim) inversion and square root requires d calculations

[*] <http://www.jmlr.org/papers/volume12/duchi11a/duchi11a.pdf>

Mehmet Fatih AMASYALI Optimization Techniques Lecture Notes

YILDIZ TECHNICAL UNIVERSITY





ChatGPT

Bu slayt, AdaGrad adı verilen bir Quasi-Newton yöntemini anlatmaktadır. AdaGrad, makine öğreniminde gradyan tabanlı optimizasyon algoritmalarını geliştiren bir yöntemdir ve özellikle büyük ölçekli veri setleri için etkilidir.

- G_t : Algoritmanın t zaman adımında, önceki tüm gradyanların dış çarpımlarının toplamını ifade eder. Bu matris, Hessian matrisinin bir taklidini oluşturmak için kullanılır ve gradyanların birikimini temsil eder.
- G_t : $d \times d$ boyutunda bir matristir, burada d parametre sayısını ifade eder.
- İki güncelleme versiyonu vardır:
 1. $x_{t+1} = x_t - \eta G_t^{-1/2} \nabla F(x_t)$: Bu formülde, G_t matrisinin tersinin karekökü alınır ve bu işlem d^4 hesaplaması gerektirir.
 2. $x_{t+1} = x_t - \eta \text{diag}(G_t)^{-1/2} \nabla F(x_t)$: Bu formülde, yalnızca G_t matrisinin köşegeninin (diagonal) tersinin karekökü alınır, bu da yalnızca d hesaplaması gerektirir ve böylece hesaplama daha verimli hale gelir.

Slaytta, AdaGrad'ın daha detaylı bir açıklamasını içeren bir makalenin referansı da verilmektedir. Bu yöntem, her bir parametrenin öğrenme oranını onun alınan gradyanların büyüklüğüne göre otomatik olarak ayarlar, böylece bazı parametrelerin diğerlerine göre daha hızlı güncellenmesini sağlar. Bu, özellikle nadir özelliklerin sık özellikler kadar hızlı öğrenilmesini sağlar ve algoritmanın verimliliğini artırır.



ChatGPT

Bu slayt, AdaGrad optimizasyon algoritmasının Gradyan İnişi (Gradient Descent - GD) ile karşılaştırılmasını içeren üç farklı grafiği göstermektedir:

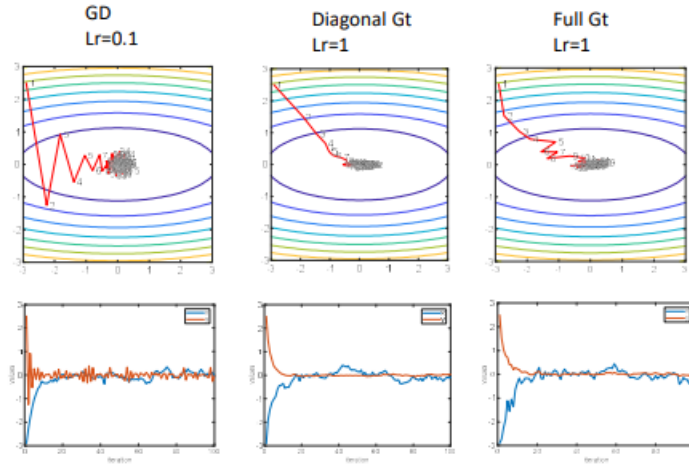
- Sol taraftaki ilk sütun, standart Gradyan İnişi algoritmasını ve öğrenme oranının $\eta = 0.1$ olduğu durumu gösteriyor. Bu grafikte, algoritmanın optimizasyon yüzeyinde nasıl ilerlediği ve minimuma (en düşük noktaya) nasıl ulaştığı gösteriliyor. Ayrıca, gradyanın büyüklüğünün zamanla nasıl azaldığını gösteren bir eğri var.
- Orta sütun, AdaGrad algoritmasının sadece köşegen elemanları (Diagonal G_t) kullanarak yapılan güncellemeleri gösteriyor ve öğrenme oranı $\eta = 1$ olarak belirlenmiş. Bu durumda, algoritma daha düzgün ve doğrudan minimuma ilerliyor gibi görünüyor.
- Sağ taraftaki sütun ise, AdaGrad'ın tam G_t matrisini kullanarak yapılan güncellemeleri gösteriyor ve burada da öğrenme oranı $\eta = 1$ olarak belirlenmiş. Bu grafik, algoritmanın minimuma doğru çok daha düz bir yoldan ilerlediğini gösteriyor.

Her sütunun altında, ilgili yöntemle ilerleme sürecinde gradyanın nasıl değiştiğini gösteren grafikler bulunmaktadır. Bunlar, optimizasyonun ilerleme hızını ve her bir adımda gradyanın nasıl değiştiğini gösterir.

Slaytın sağ üst köşesinde yer alan "code: hessian_gradyan.m" ifadesi, bu görselleştirmeleri üreten MATLAB kodunun dosya adını belirtir.

AdaGrad'ın, öğrenme oranını her parametre için adaptif olarak ayarlayarak daha hızlı ve stabil bir şekilde optimizasyon yapmasının avantajlarını gösteren bir örnektir. Her bir yöntemin farklı öğrenme oranları ve güncelleme stratejileriyle nasıl farklı davrandığını görsel olarak karşılaştırır.

Stochastic AdaGrad



ADAM[*]

Algorithm 1: *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. g_t^2 indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With β_1^t and β_2^t we denote β_1 and β_2 to the power t .

Require: α : Stepsize

Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates

Require: $f(\theta)$: Stochastic objective function with parameters θ

Require: θ_0 : Initial parameter vector

$m_0 \leftarrow 0$ (Initialize 1st moment vector)

$v_0 \leftarrow 0$ (Initialize 2nd moment vector)

$t \leftarrow 0$ (Initialize timestep)

while θ_t not converged **do**

$t \leftarrow t + 1$

$g_t \leftarrow \nabla_{\theta} f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)

$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)

$v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)

$\hat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)

$\hat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)

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

end while

return θ_t (Resulting parameters)

[*] Kingma, D. P., & Ba, J. Adam: A method for stochastic optimization. ICLR 2015, [arXiv:1412.6980](https://arxiv.org/abs/1412.6980).



ChatGPT

Bu görsel, Stokastik AdaGrad (Adaptive Gradient Algorithm) optimizasyon tekniğini anlatan bir sunum slaytını gösteriyor. Stokastik AdaGrad, makine öğreniminde sıklıkla kullanılan bir optimizasyon algoritmasıdır ve özellikle büyük ölçekli veri setlerinde etkilidir. Ana fikir, öğrenme oranını her bir parametre için özel olarak ayarlamaktır. Bu, sık kullanılan özellikler için öğrenme oranını azaltır ve daha az kullanılan özellikler için öğrenme oranını artırır.

Görselde üç farklı grafik grubu var:

- Sol tarafta "GD Lr=0.1" ile gösterilen grafikler, standart Gradient Descent (GD) algoritmasının öğrenme oranı 0.1 iken performansını gösteriyor.
- Ortadaki "Diagonal Gt Lr=1" ile gösterilen grafikler, AdaGrad'ın yalnızca gradyanın diyagonal bileşenlerini kullanarak performansını gösteriyor ve öğrenme oranı 1 olarak ayarlanmış.
- Sağ tarafta "Full Gt Lr=1" ile gösterilen grafikler ise AdaGrad'ın gradyanın tamamını kullanarak performansını gösteriyor ve yine öğrenme oranı 1.

Her grafik grubunda, yukarıdaki satırda algoritmanın optimizasyon yüzeyi üzerindeki hareketini gösteren kontur grafikler ve aşağıdaki satırda algoritmanın ilerlemesinin zamana göre değişimini gösteren grafikler bulunmaktadır.

Her bir kontur grafiğinde, algoritmanın minimum noktaya (genellikle bir optimizasyon probleminde hedeflenen) nasıl ilerlediği gösterilmektedir. Daha düzgün bir yol izleyen algoritma daha etkilidir. Alt grafiklerde, algoritmanın hedef fonksiyon değerindeki değişimi zamana göre gösterilmekte ve ideal olarak bu değerlerin zamanla azalması beklenir.

Bu slayt, Yıldız Teknik Üniversitesi'nde Mehmet Fatih Amasyalı tarafından hazırlanmış optimizasyon teknikleri ders notlarından bir parça olarak görünüyor.



ChatGPT

Bu görsel, Adam optimizasyon algoritmasının bir özetini içeren bir ders notu slaytını göstermektedir. Adam, "Adaptive Moment Estimation"ın kısaltmasıdır ve stokastik optimizasyon için popüler bir yöntemdir. Özellikle derin öğrenme uygulamalarında, büyük veri setleri ve yüksek boyutlu parametre uzayları ile çalışırken yaygın olarak kullanılır.

Adam algoritması, önceki gradyanların ortalama (ilk moment) ve varyansı (ikinci moment) gibi istatistiksel bilgileri hesaba katarak her bir parametre için adaptif öğrenme oranları uygular. Bu özellik, farklı parametrelerin farklı hızlarda güncellenmesini sağlar ve bu da genellikle daha hızlı ve daha kararlı bir yakınsamaya yol açar.

Algoritmanın adımları şu şekilde özetlenebilir:

1. Algoritmanın hiperparametreleri olan öğrenme oranı (α), ilk ve ikinci moment tahminlerinin üssel bozulma oranları (β_1 ve β_2), ve çok küçük bir sayı (ϵ) belirlenir.
2. Algoritma, parametrelerin (θ) ilk değerleriyle başlatılır.
3. İlk ve ikinci moment vektörleri sıfır vektörleriyle başlatılır.
4. Her eğitim adımında, gradyanlar hesaplanır ve ilk moment (m) ve ikinci moment (v) tahminleri güncellenir.
5. İlk ve ikinci momentler düzeltilir çünkü başlangıçta sıfırdan başladıkları için yanlılığa eğilimlidirler.
6. Düzeltilmiş momentler kullanılarak parametreler güncellenir.
7. Bu süreç, algoritma belirlenen bir durdurma kriterini karşılayana kadar devam eder.

Görselde algoritmanın kısaca özetlendiği pseudo-kod ve algoritmanın ilk olarak sunulduğu makalenin referansı bulunmaktadır: "Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. ICLR 2015." Bu çalışma, Adam algoritmasını derinlemesine incelemek isteyenler için başvuru kaynağı olarak kullanılabilir.

Sunum slaytı, Yıldız Teknik Üniversitesi'nde Mehmet Fatih Amasyalı tarafından hazırlanan optimizasyon teknikleri ders notlarından alınmıştır.



code: moving_avg.m

ADAM

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

Adagrad version of v_t : $v_t = v_{t-1} + g_t^2$

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

Adagrad version of m_t : $m_t = g_t$

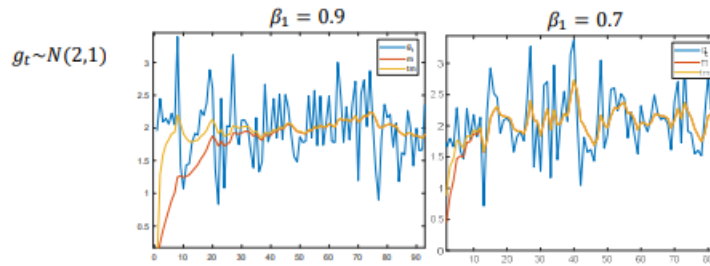
$$\hat{m}_t = m_t / (1 - \beta_1^t)$$

Corrections

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

Update

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

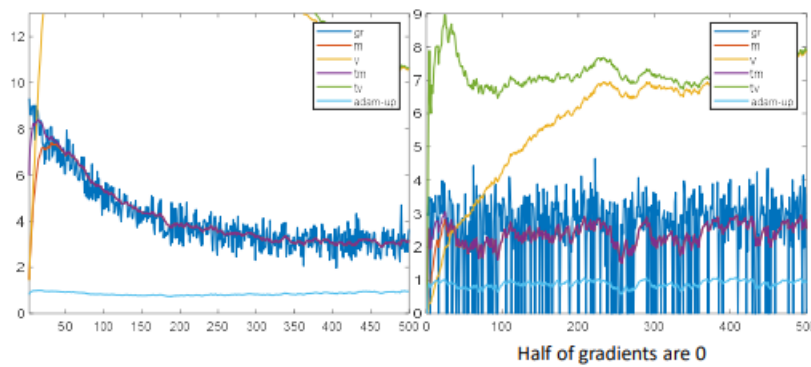


Mehmet Fatih AMASYALI Optimization Techniques Lecture Notes

YILDIZ TECHNICAL UNIVERSITY

Adam update

Robust to magnitude of gradients, noisy gradients (stochastic opt.), sparse gradients



Mehmet Fatih AMASYALI Optimization Techniques Lecture Notes

YILDIZ TECHNICAL UNIVERSITY

Bu görsel, ADAM optimizasyon algoritmasının matematiksel ifadelerini ve bu ifadelerin grafiksel temsillerini içeren bir ders notudur. ADAM, makine öğrenimi modellerinin parametrelerini optimize etmek için kullanılan bir yöntemdir ve bu slaytta anahtar bileşenleri açıklanmaktadır.

İşte görseldeki formüllerin Türkçe açıklaması:

- v_t : Zaman adımı t 'de ikinci moment tahmini (varyansın yaklaşık değeri). Bu, gradyanların karelerinin üssel hareketli ortalaması olarak hesaplanır. Adagrad algoritmasındaki karşılığı, gradyanların karelerinin toplamıdır.
- m_t : Zaman adımı t 'de ilk moment tahmini (gradyanın yaklaşık değeri). Bu, gradyanların üssel hareketli ortalaması olarak hesaplanır. Adagrad'da bu, sadece gradyanın kendisidir.
- Düzeltmeler: \hat{v}_t ve \hat{m}_t , başlangıçta yanlışlık gösterebilecek olan v_t ve m_t için düzeltilmiş değerlerdir. Bu düzeltmeler, algoritmanın daha hızlı ve doğru bir şekilde yakınsamasına yardımcı olur.
- Güncelleme (θ_t): Parametreler, düzeltilmiş ilk moment, düzeltilmiş ikinci moment ve öğrenme oranı kullanılarak güncellenir.

Algoritmanın performansını göstermek için iki grafik sunulmuştur. Her iki grafikte de, farklı β_1 değerleri kullanılarak ADAM algoritmasının nasıl farklı davrandığını gösteren simülasyonlar yer almaktadır. β_1 değeri, ilk moment tahmininin üssel hareketli ortalamasının ne kadar hızlı unutulacağını belirler. β_1 için yaygın bir değer 0.9'dur ve bu, önceki gradyanların güncel gradyan üzerindeki etkisini belirler. Grafikler, β_1 değerlerinin optimizasyon yoluna etkisini gösterir. Daha düşük bir β_1 değeri, algoritmanın daha reaktif olmasını ama aynı zamanda daha fazla dalgalanmaya eğilimli olmasını sağlar.

Bu slayt, Yıldız Teknik Üniversitesi'nde Mehmet Fatih Amasyalı tarafından verilen bir dersin notlarından alınmıştır ve ADAM algoritmasının temel prensiplerini öğrencilere anlatmayı amaçlar.



ChatGPT

Bu slaytta, Adam optimizasyon algoritmasının güncelleme mekanizmasının, gradyanların büyüklüğüne, gürültülü gradyanlara (stokastik optimizasyon için) ve seyrek gradyanlara karşı ne kadar dayanıklı olduğunu gösteren grafikler bulunmaktadır.

Slaytın üstünde "Adam update" başlığı bulunuyor. Bu, Adam algoritmasının parametre güncelleme adımlarını ifade ediyor. Adam algoritması, gradyanların değişken büyüklüklerine ve gürültüsüne iyi adapte olabilir ve seyrek gradyanlarla iyi çalışabilir. Seyrek gradyanlar, çoğunlukla sıfır olan ve yalnızca az sayıda elemanı sıfırdan farklı olan gradyanları ifade eder.

İki grafik mevcut:

- Sol taraftaki grafik, bir optimizasyon sürecinin ilerleyişini gösteriyor ve algoritmanın zamana karşı hata değerinin nasıl azaldığını gösteriyor. Hata değerinin zamanla azalması, algoritmanın başarılı bir şekilde yakınsadığını gösterir.
- Sağ taraftaki grafik, gradyanların yansının sıfır olduğu bir durumda algoritmanın performansını gösteriyor. Burada, algoritmanın, gradyanların sıfır olduğu noktalarda dahi parametreleri güncelleyebildiğini ve böylece seyrek veri setlerinde etkili olduğunu gösteren dalgalanmalar mevcut.

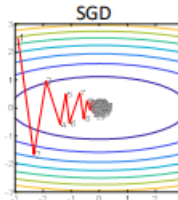
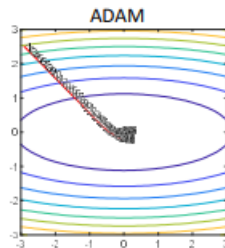
Her iki grafikte de x eksen, iterasyonları veya zaman adımlarını; y eksen ise hata değerini veya optimizasyonun o anki durumunu temsil ediyor. Grafikler, Adam algoritmasının farklı durumlardaki davranışını ve performansını gözlemlemek için kullanılıyor.

Bu slayt, Yıldız Teknik Üniversitesi'nde Mehmet Fatih Amasyalı tarafından hazırlanan optimizasyon teknikleri ders notlarının bir parçasıdır.



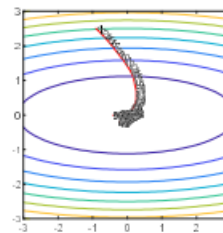
code: myADAM.m

ADAM in 2d



It directly goes to the optimal point.
Does it know the optimum point 😊

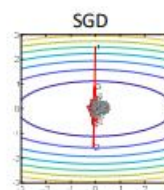
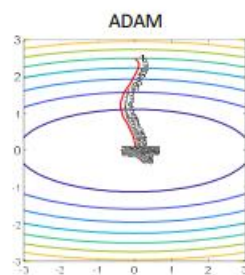
No 😊
Its update values are equal for all dimensions



Mehmet Fatih AMASYALI Optimization Techniques Lecture Notes

YILDIZ TECHNICAL UNIVERSITY

Disadvantage of ADAM



When if gradient is very close to 0 for a dimension,
it can make it bigger.

Mehmet Fatih AMASYALI Optimization Techniques Lecture Notes

YILDIZ TECHNICAL UNIVERSITY



ChatGPT

Bu slayt, iki boyutlu (2D) uzayda ADAM ve SGD (Stochastic Gradient Descent) optimizasyon algoritmalarının karşılaştırmalı performanslarını gösteren kontur grafiklerini içermektedir.

Sol üstte yer alan ADAM grafiği, algoritmanın optimizasyon yüzeyi üzerinde doğrudan en iyi (optimum) noktaya doğru ilerlediğini göstermektedir. ADAM algoritması, özellikle her bir parametre için adaptif öğrenme oranlarını ayarlayarak, optimizasyon sürecini hızlandırır ve daha stabil bir şekilde en iyi noktaya ulaşabilir.

Sağ üstteki SGD grafiği, SGD'nin daha dalgalı bir yol izlediğini ve optimum noktaya ADAM'a kıyasla daha az doğrudan bir yol izleyerek ulaştığını göstermektedir. Bu, SGD'nin sabit bir öğrenme oranı kullandığı ve bu yüzden tüm boyutlar boyunca aynı güncelleme değerlerine sahip olduğu anlamına gelir.

Altta yer alan büyük grafik, muhtemelen ADAM algoritmasının bir parametre uzayındaki davranışını tek bir durum için daha ayrıntılı olarak göstermektedir. Algoritmanın optimum noktaya doğru ilerlediğini, ancak optimum noktayı önceden "bilmediğini" vurgulamaktadır; yani algoritmanın her bir adımda sadece mevcut bilgilere dayanarak güncelleme yaptığını belirtmektedir.

Slaytın alt kısmında, bu görselin Yıldız Teknik Üniversitesi'nde Mehmet Fatih Amasyalı tarafından hazırlanan optimizasyon teknikleri ders notlarından alındığı ve bu slaytın bir parçası olduğunu belirten bir not bulunmaktadır. Ayrıca, ADAM algoritmasının MATLAB'te nasıl uygulanabileceğini gösteren bir kod parçasına ("myADAM.m") atıfta bulunmaktadır. Bu kod, ADAM algoritmasının MATLAB ortamında nasıl gerçekleştirilebileceği hakkında bir örnektir.

Buna buna tutunmuş ayınca



ChatGPT

Bu slayt, ADAM optimizasyon algoritmasının bir dezavantajını açıklamaktadır. Slaytta iki kontur grafiği yer almakta; sol tarafta ADAM ve sağ tarafta SGD (Stokastik Gradyan İnişi) algoritmasının performansları gösterilmektedir.

Slaytın başlığı "Disadvantage of ADAM" yani "ADAM'ın Dezavantajı" olarak belirtilmiştir. Açıklama kısmında, bir boyutta gradyanın sıfıra çok yakın olması durumunda, ADAM algoritmasının bu gradyanı büyütebileceği belirtilmiştir. Bu, ADAM'ın parametre güncellemelerinde gradyanların geçmiş karelerinin ortalamasını kullanmasından kaynaklanır. Eğer bir boyutta gradyan değeri çok düşükse, bu ortalama değer nispeten yüksek kalır ve bu durumda algoritma, o boyuttaki parametreyi beklenenden daha büyük bir oranda güncelleyebilir. Bu, algoritmanın o boyutta aşırı bir güncelleme yapmasına ve optimum noktayı kaçırmasına neden olabilir.

SGD grafiği, karşılaştırma amacıyla gösterilmiştir ve SGD'nin bu tür durumlarda daha stabil olabileceğini ima etmektedir.

Görselin altında, ders notlarının Yıldız Teknik Üniversitesi'nde Mehmet Fatih Amasyalı tarafından hazırlandığı ve bu bilginin ders notlarının bir parçası olduğu bilgisi verilmiştir.



Other quasi Newton methods

- Broyden–Fletcher–Goldfarb–Shanno (BFGS)
- Davidon–Fletcher–Powell (DFP)



ChatGPT

Bu görsel, kuazi-Newton yöntemlerinin iki örneğini listelemektedir:

1. **Broyden-Fletcher-Goldfarb-Shanno (BFGS)**: Bu yöntem, Newton'ın yönteminin bir yaklaşımıdır ve bir fonksiyonun ikinci türev bilgisini kullanmadan optimizasyon yapar. İkinci türev matrisine (Hessian matrisine) bir yaklaşım oluşturur ve bu sayede daha az hesaplama ile daha hızlı ilerler. Genellikle orta ölçekli optimizasyon problemlerinde tercih edilir.
2. **Davidon-Fletcher-Powell (DFP)**: Bu da kuazi-Newton yöntemlerinden biridir ve BFGS gibi Hessian matrisine bir yaklaşım oluşturur. Ancak, Hessian'ı güncelleme yöntemi BFGS'den farklıdır. DFP tarihsel olarak önemli olmasına rağmen, genellikle BFGS tarafından daha iyi sonuçlar verildiği için daha az tercih edilir.

Her iki yöntem de, özellikle büyük ölçekli olmayan optimizasyon problemlerinde, hesaplama maliyeti yüksek olan gerçek Hessian matrisi yerine yaklaşımlar kullanarak verimliliği artırmayı amaçlar.

Bu bilgiler, Yıldız Teknik Üniversitesi'nde Mehmet Fatih Amasyalı tarafından hazırlanan optimizasyon teknikleri ders notlarından alınmıştır.



