Derin Öğrenme
Deep Learning

Hazırlayan: M. Ali Akcayol
Gazi Üniversitesi
Bilgisayar Mühendisliği Bölümü

Bu dersin sunumları, “Deep Learning, I. Goodfellow, Y. Bengio and A. Courville, MIT Press, 2016.” kitabı kullanılarak hazırlanmıştır.

İçerik

- Derin modellerin eğitimi için optimizasyon
- Yapay sinir ağı optimizasyonundaki zorluklar
- Temel algoritmalar
- Parametre başlatma stratejileri
- Adaptif öğrenme oranı algoritmaları

Derin modellerin eğitimi için optimizasyon

- Yapay sinir ağı modellerinin optimizasyonu oldukça zordur.
- Bir yapay sinir ağı probleminin çözümü bazen yüzlerce makineyle aylarca sürebilir.
- Yapay sinir ağı optimizasyonunda $J(\theta)$ **maliyet fonksiyonunu minimum yapan** θ **parametreleri bulunur.**
- Genellikle, performans ölçütü P optimize edilerek dolaylı olarak J minimum yapılır.
- Eğitim veri kümesi için maliyet fonksiyonu minimum yapılır.

$$J(\boldsymbol{\theta}) = \mathbb{E}_{(\boldsymbol{x},\mathrm{y})\sim\hat{p}_{\mathrm{data}}} L(f(\boldsymbol{x};\boldsymbol{\theta}), y)$$

- Tüm veriler için maliyet fonksiyonunun minimum yapılması amaçlanır. m örnek sayısını göstermektedir.

$$J^*(\boldsymbol{\theta}) = \mathbb{E}_{(\boldsymbol{x},\mathrm{y})\sim p_{\mathrm{data}}} L(f(\boldsymbol{x};\boldsymbol{\theta}), y)$$

$$\mathbb{E}_{\boldsymbol{x},\mathrm{y}\sim\hat{p}_{\mathrm{data}}(\boldsymbol{x},y)}[L(f(\boldsymbol{x};\boldsymbol{\theta}), y)] = \frac{1}{m}\sum_{i=1}^{m} L(f(\boldsymbol{x}^{(i)};\boldsymbol{\theta}), y^{(i)})$$

Derin modellerin eğitimi için optimizasyon

- Eğitim veri kümesinin tamamını kullanan optimizasyon algoritmaları **batch** veya **deterministik** gradyan metot olarak adlandırılır.
- Sadece bir tane örnek veriyi kullanan optimizasyon algoritmaları **online** veya **stokastik** metot olarak adlandırılır.
- Derin öğrenme algoritmalarının çoğu birden çok ancak tüm eğitim veri kümesinden daha az örnek veri kullanır.
- Bu tür algoritmalar **minibatch** veya **minibatch stokastik** metot olarak adlandırılır.

Derin modellerin eğitimi için optimizasyon

- $J(\boldsymbol{X})$ maliyet fonksiyonunu minimum yapacak güncelleme değeri minibatch $\boldsymbol{X}$ için hesaplanır.
- Genelleme hatası ve gradyan aşağıdaki şekilde ifade edilir:

$$J^*(\boldsymbol{\theta}) = \sum_{\boldsymbol{x}} \sum_{y} p_{\text{data}}(\boldsymbol{x}, y) L(f(\boldsymbol{x}; \boldsymbol{\theta}), y)$$

$$\boldsymbol{g} = \nabla_{\boldsymbol{\theta}} J^*(\boldsymbol{\theta}) = \sum_{\boldsymbol{x}} \sum_{y} p_{\text{data}}(\boldsymbol{x}, y) \nabla_{\boldsymbol{\theta}} L(f(\boldsymbol{x}; \boldsymbol{\theta}), y)$$

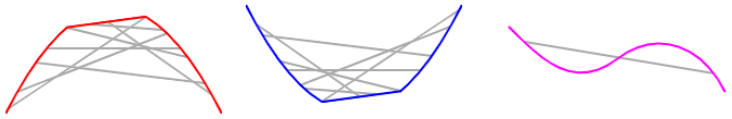
$$\hat{\boldsymbol{g}} = \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_{i} L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), y^{(i)})$$

İçerik

- Derin modellerin eğitimi için optimizasyon
- Yapay sinir ağı optimizasyonundaki zorluklar
- Temel algoritmalar
- Parametre başlatma stratejileri
- Adaptif öğrenme oranı algoritmaları

Yapay sinir ağı optimizasyonundaki zorluklar

- Optimizasyon genellikle çok zordur ve uzun sürer.
- Makine öğrenmesinde **amaç fonksiyonu ve kısıtlar** optimizasyon problemini **convex yapacak şekilde tasarlanır.**

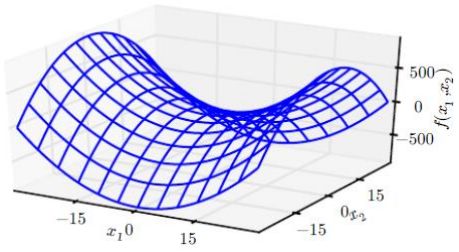


concav fonksiyon convex fonksiyon non-convex fonksiyon

- **Genellikle,** yapay sinir ağı eğitiminde **non-convex durumla karşılaşılır.**
- Convex optimizasyon probleminde lokal minimum aynı zamanda global minimumdur.
- Non-convex fonksiyonlarda birden fazla lokal minimum olabilir.
- Derin öğrenme modellerinde çok sayıda lokal minimum olabilir.

Yapay sinir ağı optimizasyonundaki zorluklar

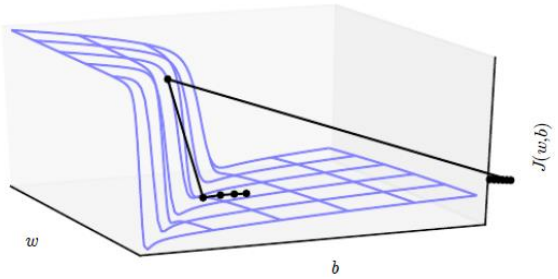
- Yüksek boyutlu non-convex fonksiyonlar sıfır gradyana sahip olan **saddle point'e (eyer noktası) sahip olabilir.**
- Bir eyer noktasında Hessian matris **hem pozitif hem de negatif** eigenvalue'ya sahiptir (hem maksimum hem de minimum nokta vardır.).



- Düşük boyutlu uzaylarda **local minimum yaygındır.**
- Yüksek boyutlu uzaylarda **saddle point yaygındır.**
- $f : \mathbb{R}^n \to \mathbb{R}$ fonksiyonu için saddle point sayısının local minimum sayısına oranının üstel artması beklenir.

Yapay sinir ağı optimizasyonundaki zorluklar

- Birçok yapay sinir ağı uçurumlara benzer **aşırı dik bölgelere sahiptir.**
- Bunun nedeni **çok sayıda büyük değere sahip ağırlığın birbiriyle çarpılmasıdır.**



- Bu durumda **gradyan güncelleme parametreleri çok büyük oranda değişir.**
- Klasik **gradient descent algoritması** parametreler için **büyük bir değişim değeri hesaplar.**

İçerik

- Derin modellerin eğitimi için optimizasyon
- Yapay sinir ağı optimizasyonundaki zorluklar
- Temel algoritmalar
- Parametre başlatma stratejileri
- Adaptif öğrenme oranı algoritmaları

Temel algoritmalar

Stochastic gradient descent

- Derin öğrenmede en çok kullanılan optimizasyon algoritmasıdır.

Algorithm 8.1 Stochastic gradient descent (SGD) update at training iteration k

Require: Learning rate ϵ_k.
Require: Initial parameter $\boldsymbol{\theta}$
  while stopping criterion not met **do**
    Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient estimate: $\hat{\boldsymbol{g}} \leftarrow +\frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon \hat{\boldsymbol{g}}$
  end while

- SGD algoritmasının en önemli parametresi öğrenme oranıdır.
- **Öğrenme oranı iterasyon sayısı arttıkça azaltılır** (ϵ_k).

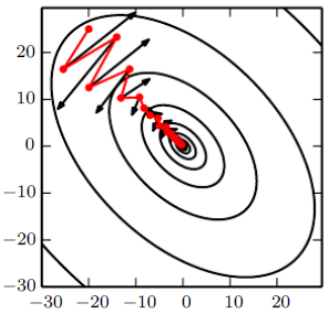
$$\epsilon_k = (1 - \alpha)\epsilon_0 + \alpha\epsilon_\tau \qquad \alpha = \frac{k}{\tau}$$

- ϵ iterasyon sonra genellikle τ sabit bırakılır.

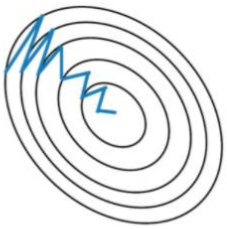
Temel algoritmalar

Momentum

- **Stochastic gradient descent** çok popüler bir optimizasyon algoritması olmasına rağmen **bazen oldukça yavaştır.**
- **Momentum** metodu **öğrenmeyi hızlandırmak için** kullanılır.



Stochastic Gradient Descent **withhout** Momentum



Stochastic Gradient Descent **with** Momentum

$$\boldsymbol{v} \leftarrow \alpha\boldsymbol{v} - \epsilon\nabla_{\boldsymbol{\theta}} \left(\frac{1}{m} \sum_{i=1}^{m} L(\boldsymbol{f}(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)}) \right) \qquad \alpha \in [0, 1)$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$$

Temel algoritmalar

Momentum

- Momentum ile stochastic gradient descent algoritması

Algorithm 8.2 Stochastic gradient descent (SGD) with momentum

Require: Learning rate ϵ, momentum parameter α.
Require: Initial parameter $\boldsymbol{\theta}$, initial velocity $\boldsymbol{v}$.
  while stopping criterion not met **do**
    Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient estimate: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    Compute velocity update: $\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \boldsymbol{g}$
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$
  end while

Temel algoritmalar

Nesterov momentum

- Nesterov momentum'da parametreler **önce mevcut değişim hızıyla değiştirilir.**
- Ardından standart momentum'daki gibi tekrar değiştirilir.

$$\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \nabla_{\boldsymbol{\theta}} \left[\frac{1}{m} \sum_{i=1}^{m} L\left(\boldsymbol{f}(\boldsymbol{x}^{(i)}; \boldsymbol{\theta} + \alpha \boldsymbol{v}), \boldsymbol{y}^{(i)} \right) \right]$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$$

Algorithm 8.3 Stochastic gradient descent (SGD) with Nesterov momentum

Require: Learning rate ϵ, momentum parameter α.
Require: Initial parameter $\boldsymbol{\theta}$, initial velocity $\boldsymbol{v}$.
  while stopping criterion not met **do**
    Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding labels $\boldsymbol{y}^{(i)}$.
    Apply interim update: $\tilde{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta} + \alpha \boldsymbol{v}$
    Compute gradient (at interim point): $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\tilde{\boldsymbol{\theta}}} \sum_i L(f(\boldsymbol{x}^{(i)}; \tilde{\boldsymbol{\theta}}), \boldsymbol{y}^{(i)})$
    Compute velocity update: $\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \boldsymbol{g}$
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$
  end while

İçerik

- Derin modellerin eğitimi için optimizasyon
- Yapay sinir ağı optimizasyonundaki zorluklar
- Temel algoritmalar
- Parametre başlatma stratejileri
- Adaptif öğrenme oranı algoritmaları

Parametre başlatma stratejileri

- Derin öğrenme **eğitim algoritmaları iteratif çalışır.**
- Iterasyonların başlaması için **kullanıcının bir başlangıç noktası belirlemesi gerekir.**
- Derin öğrenme algoritmalarının çoğu **başlangıç noktası seçiminden etkilenir.**
- Seçilen başlangıç noktası algoritmanın hiçbir zaman istenen sonuca yakınsamamasına neden olabilir.
- Seçilen başlangıç noktası algoritmanın istenen sonuca yakınsasa bile çok uzun süre gerekmesine neden olabilir.
- Farklı birimlerin (gizli katmanlar) ağırlıkları farklı şekilde başlatılmalıdır. Aynı değere sahip olmamalıdır.
- Her birimin farklı değerler alması için **parametreler rastgele başlatılır.**

Parametre başlatma stratejileri

- Yüksek boyutlu uzayda **yüksek entropy dağılımına sahip rastgele başlatmanın hesaplama maliyeti düşüktür.**
- Genellikle, **bias değerleri sabit alınır** ve **ağırlık değerleri rastgele başlatılır.**
- Tüm ağırlık değerleri genellikle **Gauss veya uniform dağılımla** rastgele atanır.
- Büyük başlangıç değerleri, doğrusal fonksiyona sahip düğümlerde verinin hem ileri geçişte hem de geri yayılımda kaybolmasına engel olur.
- Çok büyük başlangıç değerleri ise değerlerde hem ileri geçişte hem de geri yayılımda patlayan değerlere neden olur.
- **Recurrent neural network'lerde büyük ağırlık değerleri kaos'a neden olur.**

Parametre başlatma stratejileri

Xavier initialization

- m tane giriş için aşağıdaki gibi ifade edilir.

$$U(-\frac{1}{\sqrt{m}}, \frac{1}{\sqrt{m}})$$

- Burada, U uniform dağılımı gösterir.

Normalized initialization

- Glorot ve Bengio'nun önerdiği normalize edilmiş başlatma aşağıdaki şekilde ifade edilir.

$$\mathrm{W}_{i,j} \sim U\left(-\sqrt{\frac{6}{m+n}}, \sqrt{\frac{6}{m+n}}\right)$$

- Burada, m giriş sayısını, n çıkış sayısını ve U uniform dağılımı gösterir.

İçerik

- Derin modellerin eğitimi için optimizasyon
- Yapay sinir ağı optimizasyonundaki zorluklar
- Temel algoritmalar
- Parametre başlatma stratejileri
- Adaptif öğrenme oranı algoritmaları

Adaptif öğrenme oranı algoritmaları

Delta-bar-delta algoritması

- **Öğrenme oranı her bir parametre için** kısmi türev değerine göre **değiştirilir.**
- Eğer kayıp fonksiyonunun model parametrelerine göre **kısmi türevi, parametre işaretini aynı yönde bırakıyorsa**, öğrenme oranı **artırılır.**
- Eğer kayıp fonksiyonunun kısmi türevi parametre işaretini **değiştiriyorsa**, öğrenme oranı **azaltılır.**
- Bu kural full batch optimizasyonda da kullanılabilir.

Adaptif öğrenme oranı algoritmaları

AdaGrad algoritması

- AdaGrad algoritması ile öğrenme oranı her bir parametre için **tüm geçmiş değerlerinin toplamının karekökü ile orantılı olarak değiştirilir.**
- **Kısmi türev değeri büyük olan** bir parametre için öğrenme oranı **hızlı azalır, küçük olan** için **azalma yavaştır.**
- Başlangıçtan itibaren gradyan değerlerinin toplanarak alınması erken ve fazla düşüşe neden olabilir.
- AdaGrad algoritması bazı derin öğrenme modellerinde başarılı olmuştur ancak **hepsinde başarılı değildir.**

Adaptif öğrenme oranı algoritmaları

AdaGrad algoritması

Algorithm 8.4 The AdaGrad algorithm

Require: Global learning rate ϵ

Require: Initial parameter $\boldsymbol{\theta}$

Require: Small constant δ, perhaps 10^{-7}, for numerical stability

Initialize gradient accumulation variable $\boldsymbol{r} = \mathbf{0}$

while stopping criterion not met **do**

Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.

Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$

Accumulate squared gradient: $\boldsymbol{r} \leftarrow \boldsymbol{r} + \boldsymbol{g} \odot \boldsymbol{g}$

Compute update: $\Delta\boldsymbol{\theta} \leftarrow -\frac{\epsilon}{\delta + \sqrt{\boldsymbol{r}}} \odot \boldsymbol{g}$. (Division and square root applied element-wise)

Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$

end while

Adaptif öğrenme oranı algoritmaları

RMSProp algoritması

- RMSProp algoritması AdaGrad algoritmasındaki gradyan toplama işlemini değiştirmiştir.
- AdaGrad, convex fonksiyonlarda hızlı yakınsama için tasarlanmıştır.
- AdaGrad, non-convex fonksiyonlarda uygulandığında lokal convex bir bölgeye ulaşabilmektedir.
- RMSProp algoritması **uzun süreli geçmişi atarak convex bir bölgeye hızlı yakınsama yapabilir.**
- RMSProp algoritması bir **hiperparametre (ρ) kullanarak önceki değeri ile gradyan karesi arasında ağırlıklandırma yapar.**
- RMSProp, derin sinirsel ağlarda etkin bir optimizasyon algoritmasıdır.

Adaptif öğrenme oranı algoritmaları

RMSProp algoritması

Algorithm 8.5 The RMSProp algorithm

Require: Global learning rate ϵ, decay rate ρ.
Require: Initial parameter $\boldsymbol{\theta}$
Require: Small constant δ, usually 10^{-6}, used to stabilize division by small numbers.
Initialize accumulation variables $\boldsymbol{r} = 0$
while stopping criterion not met **do**
  Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
  Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
  Accumulate squared gradient: $\boldsymbol{r} \leftarrow \rho \boldsymbol{r} + (1 - \rho) \boldsymbol{g} \odot \boldsymbol{g}$
  Compute parameter update: $\Delta \boldsymbol{\theta} = -\frac{\epsilon}{\sqrt{\delta + \boldsymbol{r}}} \odot \boldsymbol{g}$. ($\frac{1}{\sqrt{\delta + \boldsymbol{r}}}$ applied element-wise)
  Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta \boldsymbol{\theta}$
end while

Adaptif öğrenme oranı algoritmaları

RMSProp algoritması ve Nesterov momentum

Algorithm 8.6 RMSProp algorithm with Nesterov momentum

Require: Global learning rate ϵ, decay rate ρ, momentum coefficient α.
Require: Initial parameter $\boldsymbol{\theta}$, initial velocity $\boldsymbol{v}$.
  Initialize accumulation variable $\boldsymbol{r} = \boldsymbol{0}$
  while stopping criterion not met **do**
    Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute interim update: $\tilde{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta} + \alpha \boldsymbol{v}$
    Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\tilde{\boldsymbol{\theta}}} \sum_i L(f(\boldsymbol{x}^{(i)}; \tilde{\boldsymbol{\theta}}), \boldsymbol{y}^{(i)})$
    Accumulate gradient: $\boldsymbol{r} \leftarrow \rho \boldsymbol{r} + (1 - \rho) \boldsymbol{g} \odot \boldsymbol{g}$
    Compute velocity update: $\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \frac{\epsilon}{\sqrt{\boldsymbol{r}}} \odot \boldsymbol{g}$. ($\frac{1}{\sqrt{\boldsymbol{r}}}$ applied element-wise)
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$
  end while

Adaptif öğrenme oranı algoritmaları

Adam (adaptive moments) algoritması

- Adam, **birinci ve ikinci derece gradyan** değerlerini kullanır.

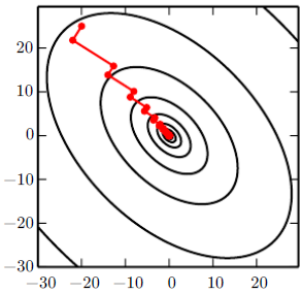
Algorithm 8.7 The Adam algorithm

Require: Step size ϵ (Suggested default: 0.001)
Require: Exponential decay rates for moment estimates, ρ_1 and ρ_2 in $[0, 1)$. (Suggested defaults: 0.9 and 0.999 respectively)
Require: Small constant δ used for numerical stabilization. (Suggested default: 10^{-8})
Require: Initial parameters $\boldsymbol{\theta}$
  Initialize 1st and 2nd moment variables $\boldsymbol{s} = \boldsymbol{0}$, $\boldsymbol{r} = \boldsymbol{0}$
  Initialize time step $t = 0$
  while stopping criterion not met **do**
    Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    $t \leftarrow t + 1$
    Update biased first moment estimate: $\boldsymbol{s} \leftarrow \rho_1 \boldsymbol{s} + (1 - \rho_1) \boldsymbol{g}$
    Update biased second moment estimate: $\boldsymbol{r} \leftarrow \rho_2 \boldsymbol{r} + (1 - \rho_2) \boldsymbol{g} \odot \boldsymbol{g}$
    Correct bias in first moment: $\hat{\boldsymbol{s}} \leftarrow \frac{\boldsymbol{s}}{1 - \rho_1^t}$
    Correct bias in second moment: $\hat{\boldsymbol{r}} \leftarrow \frac{\boldsymbol{r}}{1 - \rho_2^t}$
    Compute update: $\Delta \boldsymbol{\theta} = -\epsilon \frac{\hat{\boldsymbol{s}}}{\sqrt{\hat{\boldsymbol{r}}} + \delta}$ (operations applied element-wise)
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta \boldsymbol{\theta}$
  end while

Adaptif öğrenme oranı algoritmaları

Conjugate gradients

- Conjugate gradients, **arama yönündeki her doğrunun bir öncekine dik olmasını garanti eder.**



- İterasyon t için $\boldsymbol{d_t}$ aşağıdaki şekilde hesaplanır (β_t sabit katsayı):

$$\boldsymbol{d}_t = \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) + \beta_t \boldsymbol{d}_{t-1}$$

- Eğer $\boldsymbol{d}_t^\top \boldsymbol{H} \boldsymbol{d}_{t-1} = 0$ ise, $\boldsymbol{d_t}$ ve $\boldsymbol{d_{t-1}}$ conjugate olarak ifade edilir.

$$\boldsymbol{H} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}}^2 \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$$

Adaptif öğrenme oranı algoritmaları

Conjugate gradients

- β_t katsayısının belirlenmesi için iki popüler yöntem vardır:

Fletcher-Reeves:

$$\beta_t = \frac{\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_t)^\top \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_t)}{\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_{t-1})^\top \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_{t-1})}$$

Polak-Ribière:

$$\beta_t = \frac{(\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_t) - \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_{t-1}))^\top \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_t)}{\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_{t-1})^\top \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_{t-1})}$$

- $\boldsymbol{k}$-boyutlu bir parametre uzayında conjugate gradients metodu en çok $\boldsymbol{k}$ çizgi aramasıyla minimuma ulaşır.

Adaptif öğrenme oranı algoritmaları

Conjugate gradients

Algorithm 8.9 The conjugate gradient method

Require: Initial parameters $\boldsymbol{\theta}_0$
Require: Training set of m examples
Initialize $\boldsymbol{\rho}_0 = \mathbf{0}$
Initialize $g_0 = 0$
Initialize $t = 1$
while stopping criterion not met **do**
  Initialize the gradient $\boldsymbol{g}_t = \mathbf{0}$
  Compute gradient: $\boldsymbol{g}_t \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
  Compute $\beta_t = \frac{(\boldsymbol{g}_t - \boldsymbol{g}_{t-1})^\top \boldsymbol{g}_t}{\boldsymbol{g}_{t-1}^\top \boldsymbol{g}_{t-1}}$ (Polak-Ribière)
  (Nonlinear conjugate gradient: optionally reset β_t to zero, for example if t is a multiple of some constant k, such as $k = 5$)
  Compute search direction: $\boldsymbol{\rho}_t = -\boldsymbol{g}_t + \beta_t \boldsymbol{\rho}_{t-1}$
  Perform line search to find: $\epsilon^* = \text{argmin}_\epsilon \frac{1}{m} \sum_{i=1}^{m} L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}_t + \epsilon \boldsymbol{\rho}_t), \boldsymbol{y}^{(i)})$
  (On a truly quadratic cost function, analytically solve for ϵ^* rather than explicitly searching for it)
  Apply update: $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \epsilon^* \boldsymbol{\rho}_t$
  $t \leftarrow t + 1$
end while