

Week - 2

Optimization Algorithms

Mini-batch Gradient Descent

Vektörleşmenin bize sağladığı fayda, m sayıdaki bütün örnekler üzerinde etkin bir hesap yapmanızdır. Bu durum bütün eğitim kümesi üzerinde belirli bir formül olmadan işlem yapmanızı sağlar. Bu yüzden bütün eğitim verilerimizi X adındaki büyük matrisimizin içine ekleyeceğiz. Eğitim setiniz çok büyükse gradient descent sizi yavaşlatabilir yani tüm eğitim setini kullanarak gradient descent uygularız. Eğer eğitim veriniz çok büyükse bunu küçük eğitim setlerine böleriz buna mini-batch olarak adlandırırız. Minibatchler $X^{[1]}$ olarak gösterilir. Batch gradient descent tüm eğitim örneği için aynı anda uyguladığımız methodur.

Batch vs. mini-batch gradient descent

X, Y $X^{[1]}, Y^{[1]}$

Vectorization allows you to efficiently compute on m examples.

$$X = \begin{bmatrix} x^{(1)} & x^{(1)} & x^{(1)} & \dots & x^{(1000)} & | & x^{(1001)} & \dots & x^{(2000)} & | & \dots & | & \dots & x^{(m)} \end{bmatrix}$$

(n_x, m) $X^{[1]} (n_x, 1000)$ $X^{[2]} (n_x, 1000)$ $X^{[5,000]} (n_x, 1000)$

$$Y = \begin{bmatrix} y^{(1)} & y^{(1)} & y^{(1)} & \dots & y^{(1000)} & | & y^{(1001)} & \dots & y^{(2000)} & | & \dots & | & \dots & y^{(m)} \end{bmatrix}$$

$(1, m)$ $Y^{[1]} (1, 1000)$ $Y^{[2]} (1, 1000)$ $Y^{[5,000]} (1, 1000)$

What if $m = 5,000,000$?

5,000 Mini-batches of 1,000 each

Mini-batch $X^{[1]}, Y^{[1]}$

6:07 / 11:28 algoritmayı ifade etmektedir. Andrew Ng

Mini-batch gradient descent'i eğitim setlerinizde çalıştırmak için, for $t = 1, \dots, 5000$ olur.

For döngüsünün içinde yapmanız gereken gradyan inişin bir adımını uygulamaktır.

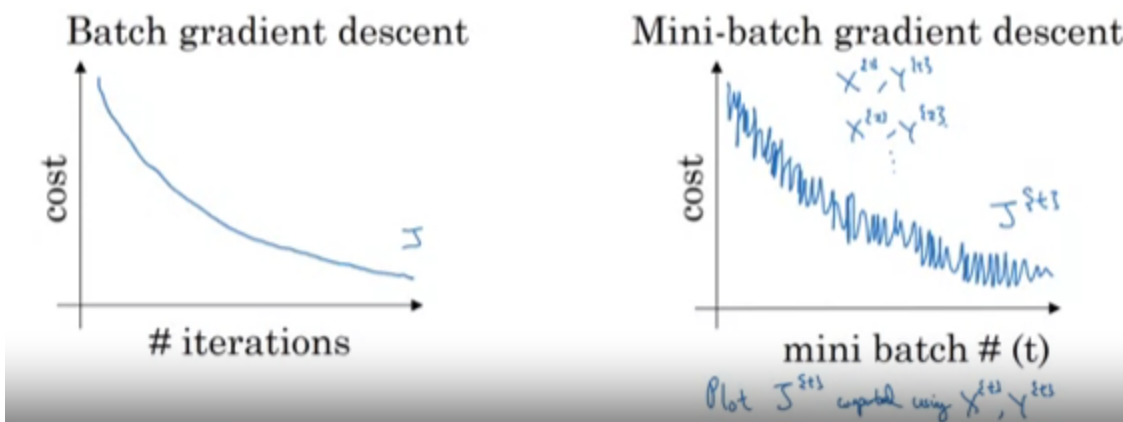
Yaptığımız şey genel olarak aynı, sadece bunu X, Y üzerine yapmak yerine $X^{[t]}, Y^{[t]}$ 'ye göre yapıp $J^{[t]}$ 'ye göre gradyanları hesaplıyoruz.

Understanding Mini-batch Gradient Descent

Batch gradient descentte, J fonksiyonu her yinelemede azalma gösterir. Fakat bir yukarı çıkma varsa ters giden bir şeyler vardır. Öğrenme hızınız yüksek olabilir.

Mini batch gradient descenette ise, eğer cost fonksiyonunuzun gelişimini çizerseniz, bu durumda her yinelemede cost fonksiyonunuz azalmayabilir. Özellikle, her yinelemede, $X(t)$, $Y(t)$ 'yi işlerken, eğer cost fonksiyon $J(t)$ 'yi çizerseniz, ki buda X ve Y 'yi kullanarak çiziliyor. Bu durumda her yinelemede farklı bir eğitim setinde ya da gerçekten farklı bir minibatchte eğitim sağlıyorsunuz gibi olacaktır. Dolayısıyla cost fonksiyonu çizerseniz, parazitli bir iniş olacaktır.

Training with mini batch gradient descent



Seçmeniz gereken parametrelerden biri ise minibatch boyutudur. Diyelim ki m eğitim seti büyüklüğü l sun, bir uçta eğer minibatch boyutunu m yaparsanız batch gradient descent olur. Eğer minibatch size = 1 verirsiniz stokastik gradient descent olur. İçlerinde 1 veri bulunur ve baştan iloc1 şeklinde ilerler. Örneklerle tek tek bakar.

Choosing your mini-batch size

If mini-batch size = m : Batch gradient descent. $(X^{(1)}, Y^{(1)}) = (X, Y)$.
If mini-batch size = 1 : Stochastic gradient descent. Every example is its own $(X^{(1)}, Y^{(1)}) = (x^{(1)}, y^{(1)}) \dots (x^{(m)}, y^{(m)})$ mini-batch.

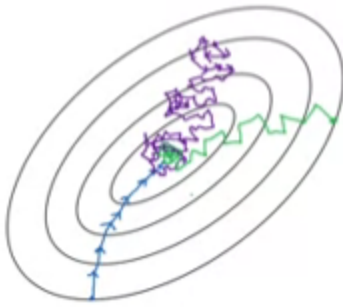
Pratikte, kullanacağınız minibatch size iki uç arasında bir yerde olacaktır. (1-m). Batch, veri büyükse bizi yavaşlatır. Skolastik ise tek vectorizasyon işleminden bizi kısıtlar ve zamanımızı çalar.

Choosing your mini-batch size

→ If mini-batch size = m : Batch gradient descent. $(X^{(1)}, Y^{(1)}) = (X, Y)$

→ If mini-batch size = 1 : Stochastic gradient descent. Every example is its own mini-batch. $(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}) \dots (x^{(n)}, y^{(n)})$ mini-batch.

In practice: Search in-between 1 and m



Stochastic
gradient
descent
↓
Less sensitive
from initialization

In-between
(mini-batch size
not too big/small)

Fastest learning

- Vectorization

- ($n \times 1000$)

- Make progress without

making extra trig calc.

Batch
gradient descent
(mini-batch size = m)

↓
Too long
per iteration

birkaç prensip verelim,

Andrew Ng

1-m arasında olması gerekiyor fakat bunu nasıl seçeceğiz?

Eğer, küçük bir eğitim setiniz varsa batch gradient descent yöntemi kullanılır.

Eğer büyükse, 64ten 512ye kadar kullanılır. 2nin üstel sayıları olması mantıklıdır. Önceki videoda 1000 yaptık ama onu 104 olarak görün. Altı çizilen değerler yaygındır.

Choosing your mini-batch size

If small toy set : Use batch gradient descent.
($m \leq 2000$)

Typical mini-batch sizes:

64 , 128 , 256 , 512 $\frac{1024}{2^{10}}$
 2^6 2^7 2^8 2^9

Make sure mini-batch fits in CPU/GPU memory.
 $x^{(i)}, y^{(i)}$.

Exponentially Weighted Averages

Gradient Descentten daha hızlı çalışan optimizasyon algoritmalarına göz atalım. Fakat önce exponentially weighted averages(üstel ağırlıklı ortalama)'nın ne olduğunu bilmeniz gerekmektedir.

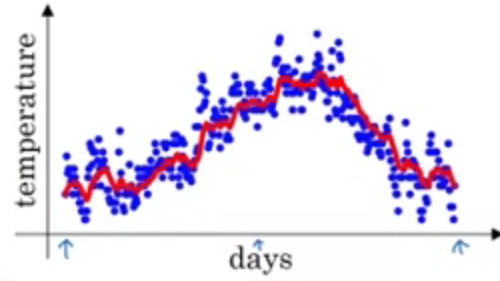
Aşağıdaki Londra sıcaklık verisinin grafiğinde biraz gürültülü olduğundan ısının yerel ortalaması ya kayan ortalaması gibi gittiği yönü bulmak biraz zordur. Ne yapabiliriz?

Vnin başlangıç değeri 0 olsun. Takip eden her gün, değerini 0.9 ile ağırlandırarak ortalamasını alarak, $0.9 \times \text{önceki değer} + 0.1 \times \text{Günün Isısı}$ şekilde devam eder.

Elde edilen V değerleri grafike kırmızı çizgi ile gösteriliyor ve exponentially weighted averages(üstel ağırlıklı ortalama'yı) buluyoruz.

Temperature in London

$$\begin{aligned}\theta_1 &= 40^\circ\text{F} & 4^\circ\text{C} \leftarrow \\ \theta_2 &= 49^\circ\text{F} & 9^\circ\text{C} \\ \theta_3 &= 45^\circ\text{F} & \vdots \\ &\vdots & \\ \theta_{180} &= 60^\circ\text{F} & 15^\circ\text{C} \\ \theta_{181} &= 56^\circ\text{F} & \vdots \\ &\vdots & \end{aligned}$$



$$\begin{aligned}V_0 &= 0 \\ V_1 &= 0.9 V_0 + 0.1 \theta_1 \\ V_2 &= 0.9 V_1 + 0.1 \theta_2 \\ V_3 &= 0.9 V_2 + 0.1 \theta_3 \\ &\vdots \\ V_t &= 0.9 V_{t-1} + 0.1 \theta_t\end{aligned}$$

kayan ortalamasını elde edersiniz.

Andrew Ng

$V_t = \beta V_{t-1} + (1-\beta)Q_t$ beta = 0.9 bulunur.

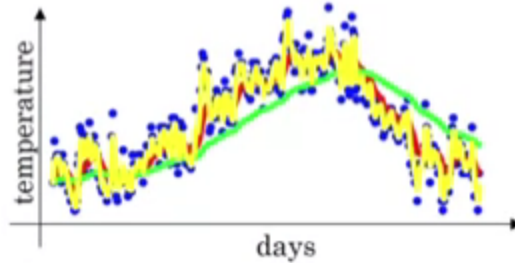
Betaya 1'e yakın değer verelim. 0.98 olarak. Daha fazla günlere bakarsınız çizgi yeşil oldu.

beta 0.5 olursa 2 güne bakarsanız ve sarı çizgi oldukça gürültülü olur(noise)

$$\begin{aligned}\beta &= 0.9 & \approx 10 \text{ days} \\ \beta &= 0.98 & \approx 50 \text{ days} \\ \beta &= 0.5 & \approx 2 \text{ days}\end{aligned}$$

$$\text{Ortalama okur} \rightarrow \approx \frac{1}{1-\beta} \text{ days' temperature.}$$

$$\frac{1}{1-0.98} = 50$$



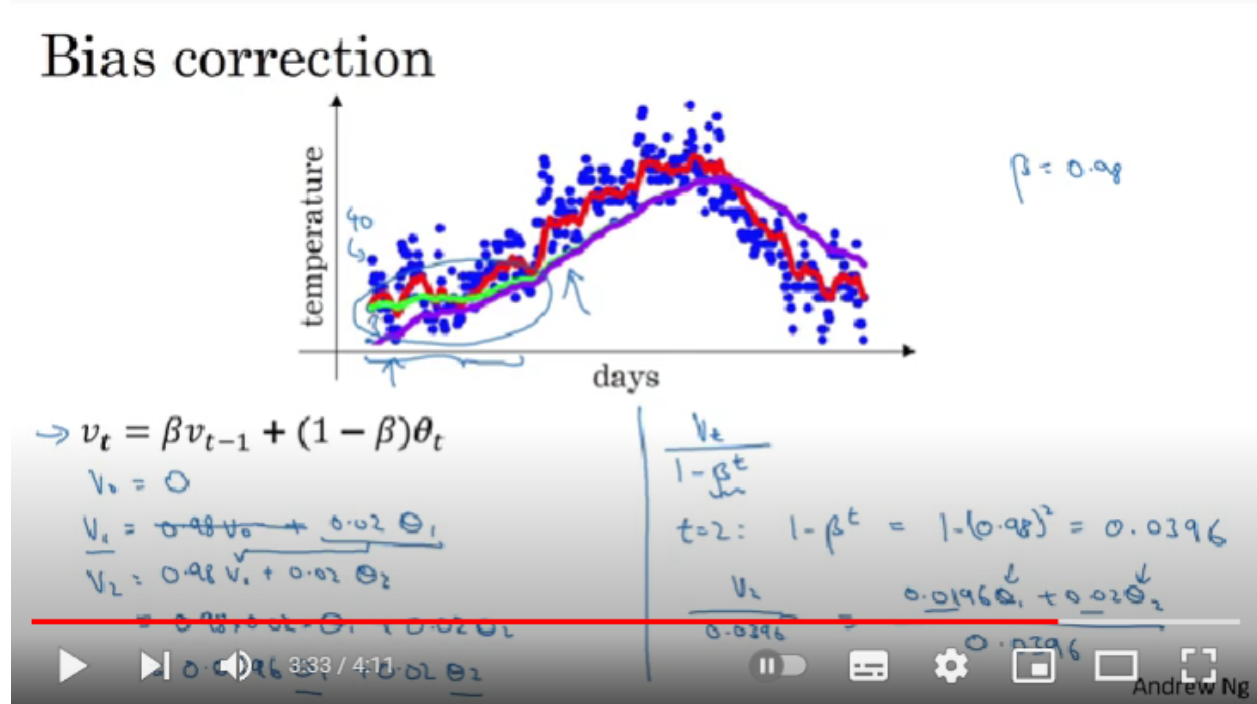
yeşil veya sarı eğrilerin beta ısı ortalamasıdır.

Andrew Ng

Bias Correction in Exponentially Weighted Averages

Bias ile daha teknik ve daha iyi çalıştığına göz atalım. Gerçekte 0.98 için yeşil çizgi yerine moru elde edersiniz. Mor eğrinin düşük yeşile göre düşük başladığına dikkat

edin. $V_0 = 0$ başlattığınızdan V_1 değeri oldukça küçük başlar ve bu iyi bir tahmin değildir ve haliyle diğer V değerlerini de etkilemektedir. T büyük olduğunda yeşil ve mor çizgi hemen hemen ölçüşür.



Gradient Descent with Momentum

Ekranda maliyet fonksiyonumuz var ve kırmızı nokta yerel minimum noktamızdır. Yukarı aşağı salınımlar gradient descenti yavaşlatır ve sizin daha yüksek öğrenme oranı kullanmanızı önler. Dikey ekseninde salınım istemezsiniz fakat yatay ekseninde öğrenmenin daha hızlı olmasını istersiniz. İşte burada momentum ile gradient descent devreye giriyor. Her yinelemede sırasında dw, db gibi klasik türevler hesaplanır.

$V_{dw} = \text{Beta} V_{dw} + (1 - \text{Beta}) dw$ hesaplamasına benzer.

Aynı şekilde, $V_{db} = \text{Beta} V_{db} + (1 - \text{beta}) db$

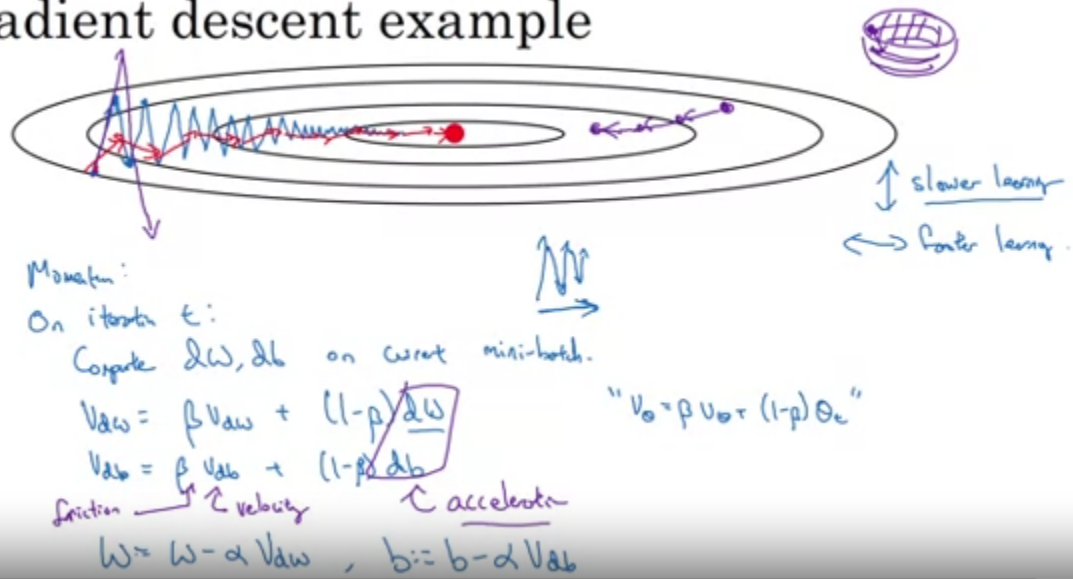
α = öğrenme oranları çarpımı (learning rate)

$W = W - \alpha V_{dw}$ $b = b - \alpha V_{db}$

Bu durum gradient descent basamaklarını yavaşlatır.

Sonuç olarak daha küçük salınım hareketleriyle, yatayda daha hızlı hareket etmeye yatkın oluyoruz.

Gradient descent example



Son olarak bunu nasıl uygulayacağımız hakkında detayları konuşalım. İşte algoritma ve bu yüzden şimdi learning rate oranının ağırlı ortalanızı kontrol eden alfa ile birlikte beta parametrelerine sahipsiniz. Betanın en yaygın değeri 0.9dur. V_{dw} ve v_{db} sıfırlar matrisi olarak başlatılır. Boyutları aynıdır. $1-\beta$ değeri oldukça 1e yakın olduğu için kullanılmadığı alanlar olabilir. Bu sadece alfanın öğrenme oranını etkiler.

Implementation details

$$V_{dw} = 0, V_{db} = 0$$

On iteration t :

Compute dW, db on the current mini-batch

$$\begin{aligned} \rightarrow v_{dw} &= \beta v_{dw} + (1-\beta) dW \\ \rightarrow v_{db} &= \beta v_{db} + (1-\beta) db \end{aligned} \quad \left| \quad v_{dw} = \beta v_{dw} + dW \leftarrow$$

$$W = W - \alpha v_{dw}, \quad b = b - \alpha v_{db}$$

$$\frac{v_{dw}}{1-\beta^t}$$

Hyperparameters: α, β $\beta = 0.9$

Sadece alfa'da öğrenme oranının bu iki farklı versiyon için farklı

Andrew Ng

RMSprop

Root mean square prop'u temsil eder ve bu da gradient descenti hızlandırabilir. Nasıl kullanıldığına bakalım.

On iteration t:

her zaman yaptığımız güncel turev dw,db on minibatch

$$S_{dw} = \beta_2 S_{dw} + (1-\beta_2) dW^2$$

$$S_{db} = \beta_2 S_{db} + (1-\beta_2) db^2$$

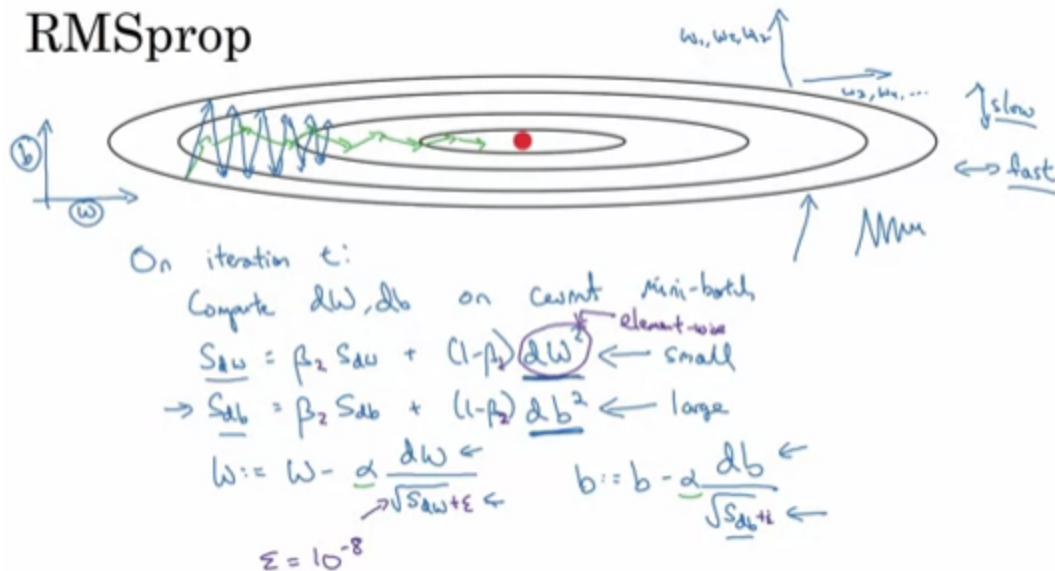
Takiben, RMSprop parametreleri şu şekilde günceller: W, W-a(öğrenme hızı) olarak güncellenir ve $W = W - \alpha \frac{dw}{\sqrt{S_{dw}}}$ olur.

$b = b - \alpha \frac{db}{\sqrt{S_{db}}}$ Peki nasıl çalışır.

Yatay doğrultuda gradient descent hızlı olsun istiyoruz. Dikey doğrultuda salınım istemiyoruz. Fonksiyon dikey doğrultuda(b), yatay doğrultuda(w) olduğundan w'yi küçük b'yi büyük isteriz.

Yeşil çizgi RMSprop olur ve böylece daha büyük bir learning rate ve dikey doğrultuda ıraksama yaşamadan daha hızlı öğrenmenizi sağlayacak.

Paydaya epsilon eklenir sıfırdan uzak olması için.



Adam Optimization Algorithm

Temelinde momentum ve RMSprop'u birlikte kullanır.

Beta1 momentum, Beta2 RMSprop'undur.

Adam optimization algorithm

$V_{dw} = 0, S_{dw} = 0, V_{db} = 0, S_{db} = 0$

On iteration t :

Compute dw, db using current mini-batch

$V_{dw} = \beta_1 V_{dw} + (1 - \beta_1) dw, V_{db} = \beta_1 V_{db} + (1 - \beta_1) db \leftarrow \text{"momentum"} \beta_1$

$S_{dw} = \beta_2 S_{dw} + (1 - \beta_2) dw^2, S_{db} = \beta_2 S_{db} + (1 - \beta_2) db^2 \leftarrow \text{"RMSprop"} \beta_2$

Güncellenmiş w değerine göre ikisinin birleşimi şu şekilde gözükür.

$$w := w - \alpha \frac{V_{dw}}{\sqrt{S_{dw}} + \epsilon} \quad b := b - \alpha \frac{V_{db}}{\sqrt{S_{db}} + \epsilon}$$

Hiperparametrelerde ise learning rate(alfa) ayarlanmalıdır. Beta1 0.9, Beta2 ise 0.999 tavsiye edilir. Epsilon seçimi ise çok mühim değildir. 10^{-8} önerilir.

→ α : needs to be tune
→ β_1 : 0.9 (dw)
→ β_2 : 0.999 (dw^2)
→ ϵ : 10^{-8}

Learning Rate Decay

Learning Rate Decay

Please note that in the next video, at time 3:35, the values for alpha should be:

Epoch 1: alpha 0.1

Epoch 2: alpha 0.067

Epoch 3: alpha 0.05

Epoch 4: alpha 0.04

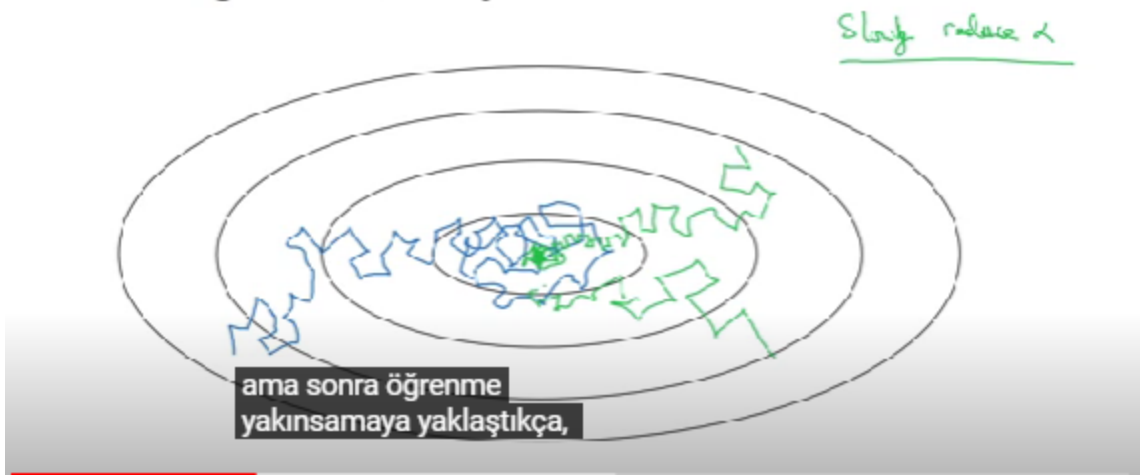
The formula for learning rate decay is:

$$\alpha = \frac{1}{1 + \text{decayRate} \times \text{epochNumber}} \alpha_0$$

Learning Rate Decay

Zamanla öğrenme hızını düşürmek en iyi alfa seçimidir. Minibatchlerde sabit alfa kullanmak gürültülü bir etki yaratabilir. Yerel minimuma yaklaştıkça alfa küçülür ve daha dar bir bölgede salınım yaparsınız. Öğrenmenin ilk adımında büyük adımlar atmayı göze alabilirsiniz.

Learning rate decay



Tüm minibatchlerde yani tüm eğitim setinde turlamasına epoch denir.

Learning rate decay

1 epoch = 1 pass through data.

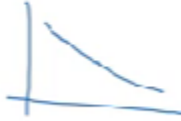
$$\alpha = \frac{1}{1 + \text{decay-rate} * \text{epoch-number}} \cdot \alpha_0$$



$$\alpha_0 = 0.2$$

$$\text{decay-rate} = 1$$

Epoch	α
1	0.1
2	0.67
3	0.5
4	0.4
...	...



onun öğrenme hızı

düşüşü formülü, insanların kullandığı birkaç başka yol

Andrew Ng

Başka rate decay methodlarda vardır.

for example

$$\alpha = 0.95^{\text{epoch-number}} \cdot \alpha_0 \quad - \text{exponential decay.}$$

$$\alpha = \frac{k}{\sqrt{\text{epoch-number}}} \cdot \alpha_0 \quad \text{or} \quad \frac{k}{\sqrt{t}} \cdot \alpha_0$$

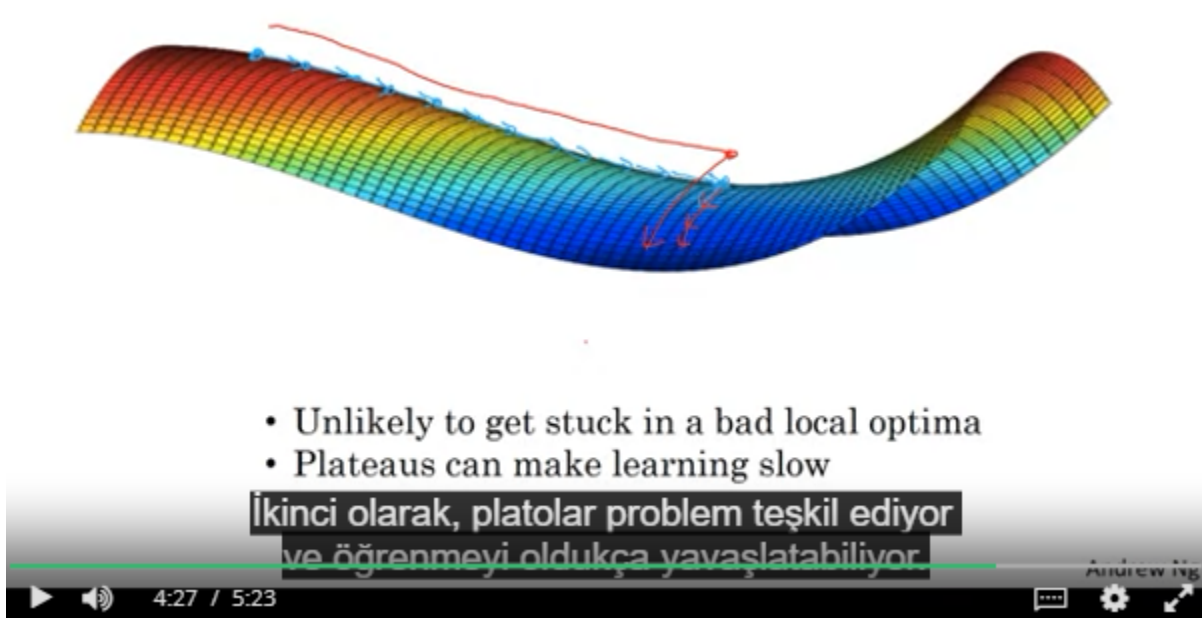
bu sadece az sayıda modeli ve bazen

The Problem of Local Optima

Yerel optimum nedir doğrusu nedir?

Ekranda bir sürü yerel optimum gözürüyoruz. Gradient descent ya da herhangi bir optimizasyon methodu yerel optimumların birinde sıkışıp kalabilir. Cost fonksiyonundaki sıfır gradyanın çoğu noktası eyer noktalardır. Çok boyutlu uzaylarda yerel minimum eyer noktası olur.

Küçük boyutlu eksenlerde parametrelerin az olduğu yerlerde yerel minimum soldaki gibi görülebilir. 20bin parametrenin olduğu durumda eyer noktasıyla karşılaşma ihtimaliniz çok yüksek.



Bu durumda adama, rmsprop ve momentum işimize yarıyor.