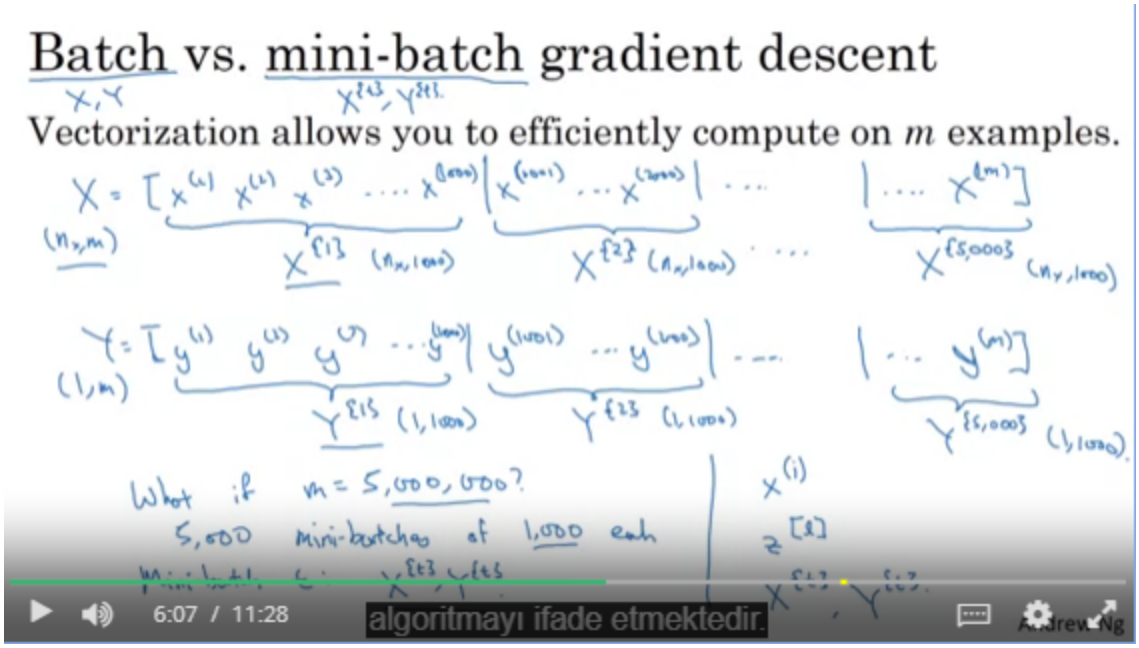
Week - 2

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 kullanark gradient descent uygularız. Eğer eğitim veriniz çok büyükse bunu küçük eğitim setlerine böleriz buna mini-batch olarak adlandırırz. Minibatchler X^{1} olarak gösterilir. Batch gradient descent tüm eğitim örneği için aynı anda uyguladığımız methodur.



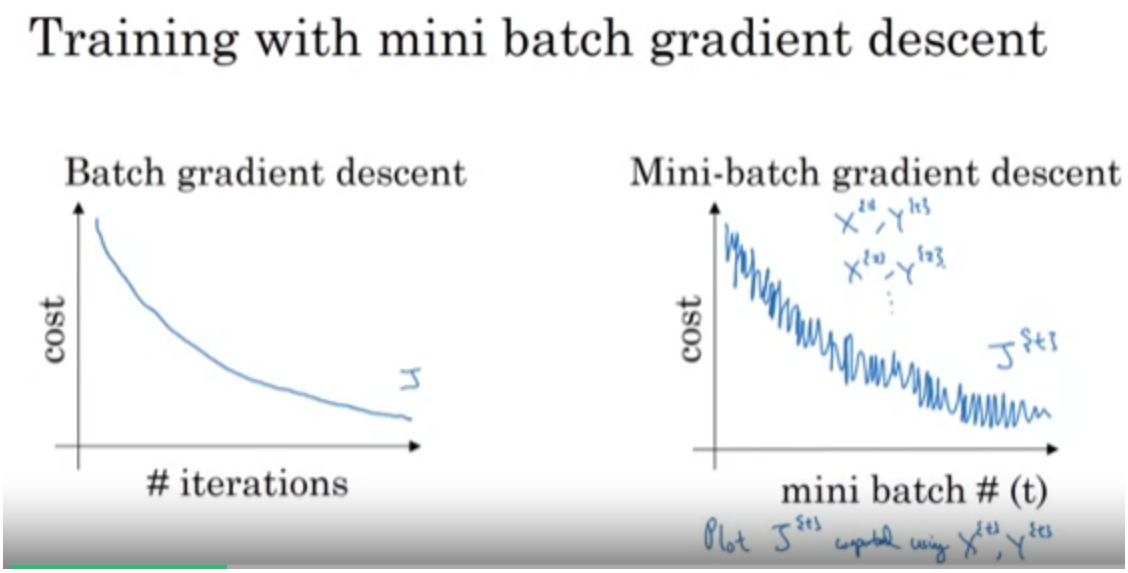
Mini-batch gradient descent'i eğitim setlerinizde çalıştırmak için, for t = 1, ...5000 olur. For döngsünün içinde yapmanız gereken gradyan inişin bir adımını uygulamaktır.

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.

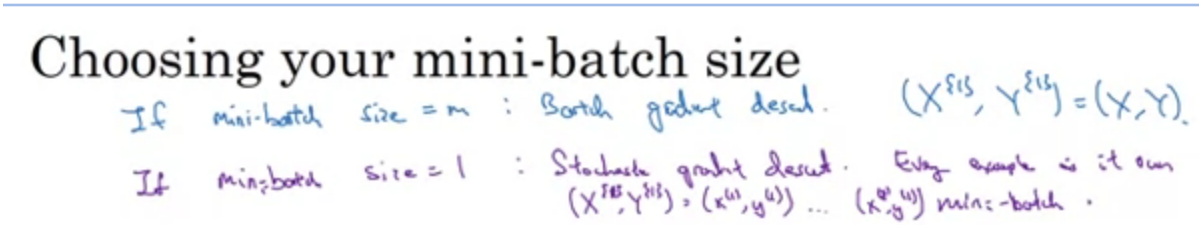
Understanding Mini-batch Gradient Descent

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

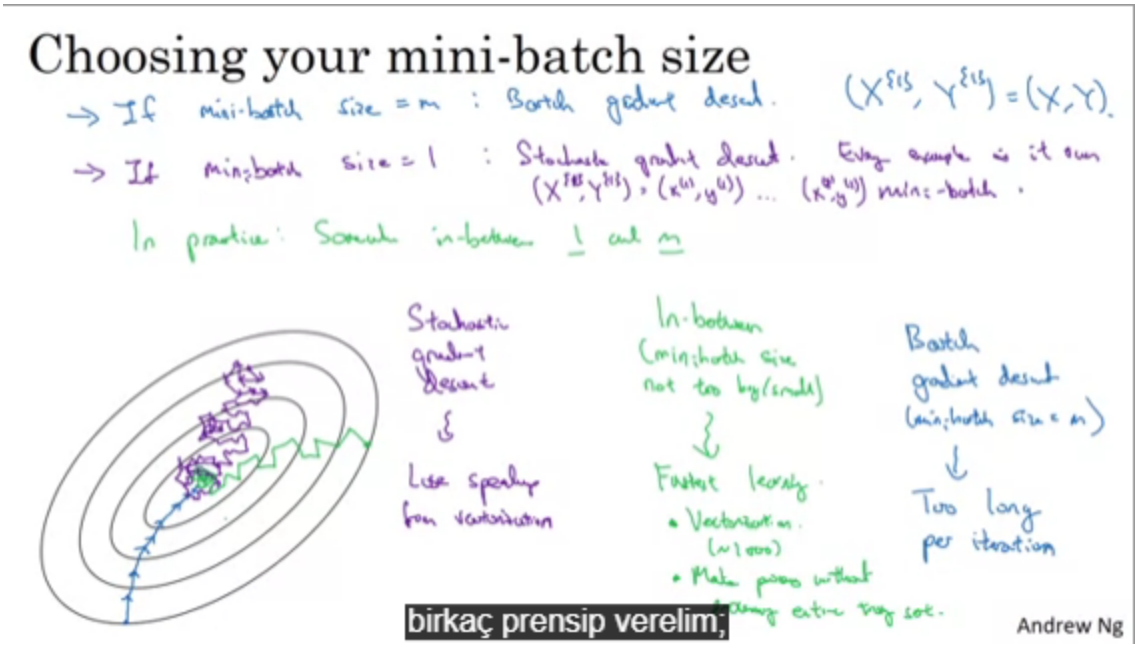
Mini batch gradient descenette ise, eğer cost fonksiyonunuzun gelişimini çizerseniz, bu durumda her yinelemede cost fonksiyonunuz azalmayabilir. Özellikle, her yinelemede, X(t), Y(t)'yi işlerken, eğer cost fonksiyon J(t)'yi çizerseniz, ki buda X ve Y'yi kullanarak çiziliyor. Bu durumda her yinelemede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



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



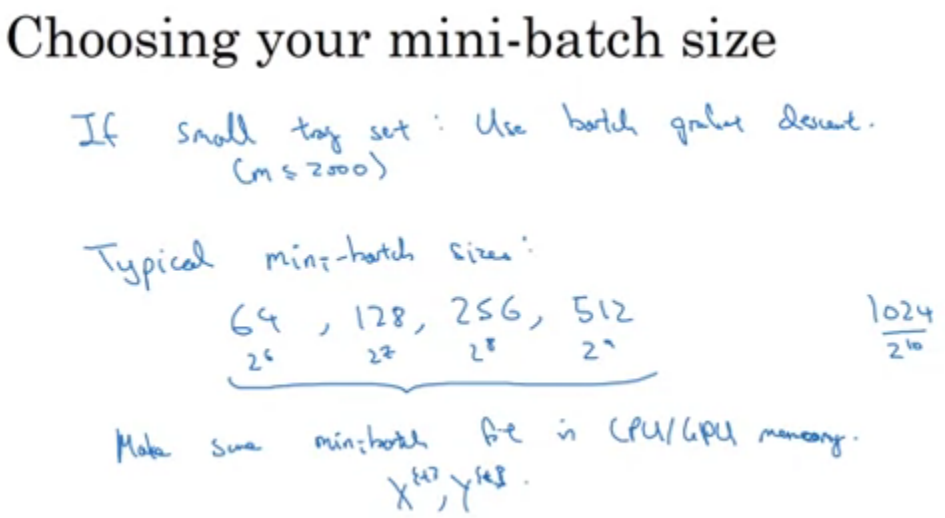
Pratikte, kullanacağınız minibatch size iki uç arasında bir yerde olacaktır. (1-m).Batch, veri büyükse bizi yavaşlatır. Skolastik ise tek vectorizasyon işleminden bizi kısatlar ve zamanımızı çalar.



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 vdieoda 1000 yaptık ama onu 104 olarak görün. Altı çizilen değerler yaygındır.



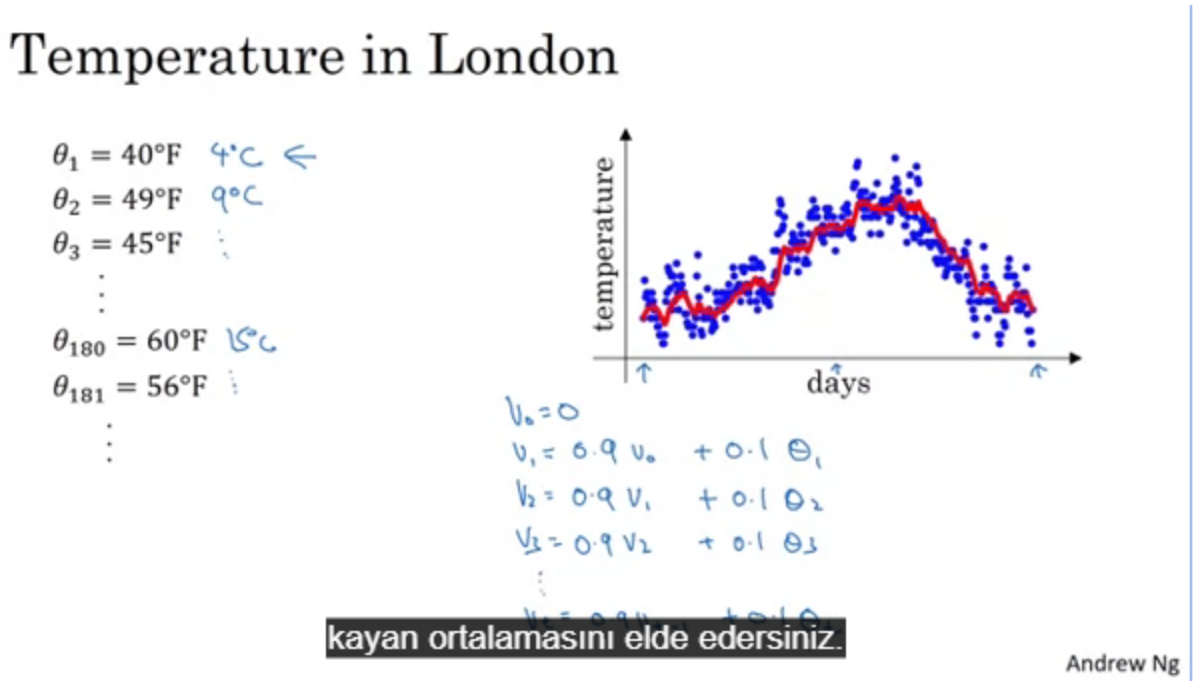
Exponentially Weighted Averages

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

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

Vnin başlangıç değeri 0 olsun. Takip eden her gün, değerini 0.9 ile ağırlandırarak ortalamasını alarak, 0.9xönceki değer + 0.1xGününIsısı şekilde devam eder.

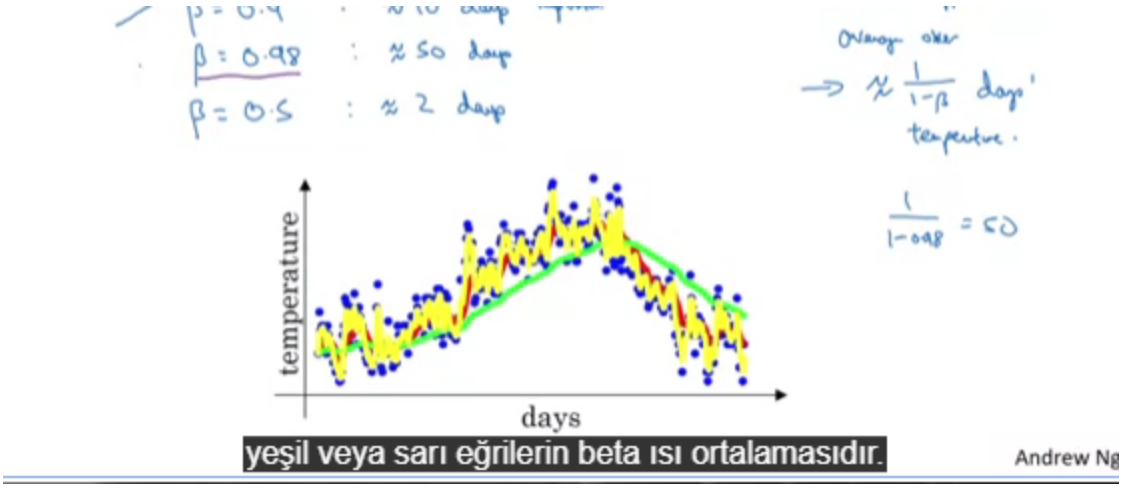
Elde edilen V değerleri grafike kırmızı çizgi ile gösteriliyor ve exponentially weighted avargaes(üstel ağırlıklı ortalama)yı buluyoruz.



Vt = betaxVt-1 + (1-beta)Qt beta = 0.9 bulunur.

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

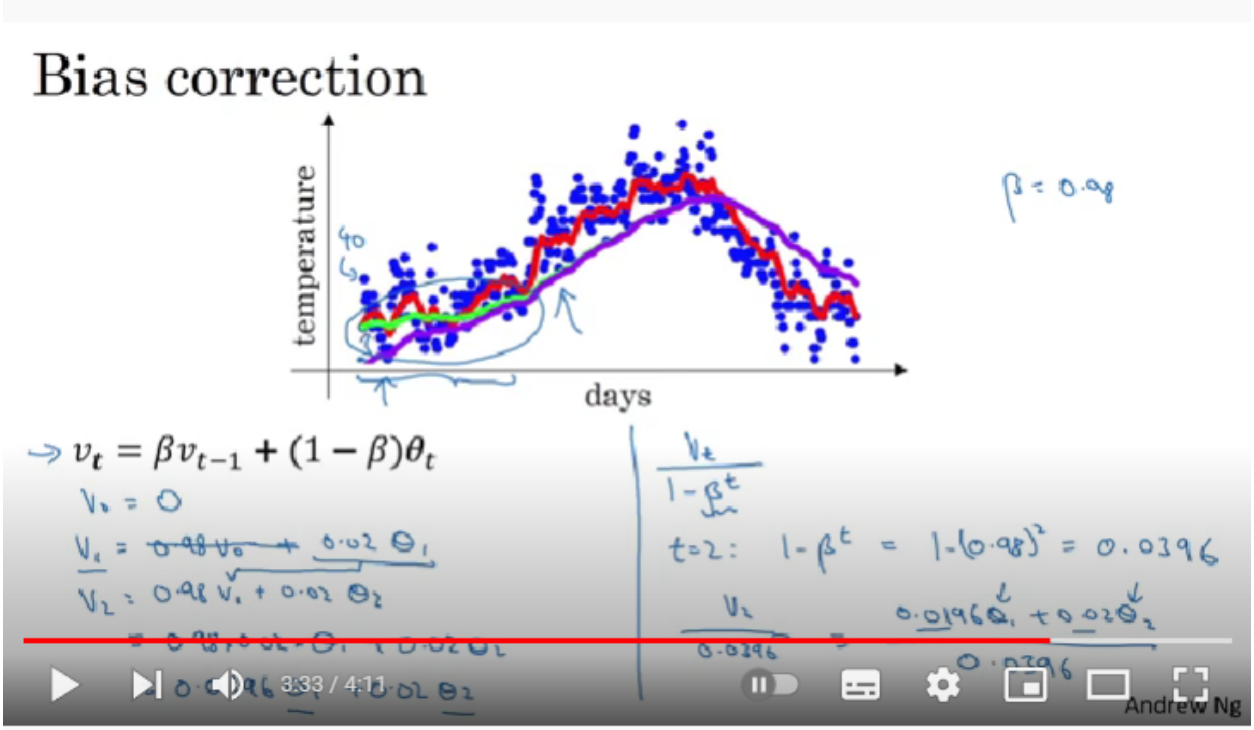
bea 0.5 olrsa 2 güne bakarsanız ve sarı çizgi oldukça gürültülü olur(noise)



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. V0 = 0 başlattığınızdan V1 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 eksende salınım istemezsiniz fakat yatay eksende öğrenmenin daha hızlı olmasını istersiniz. İşte burada momentum ile gradient descent devreye giriyor. Her yinelemede sırasında dw,db gibi klasik türevler hesaplanır.

Vdw = BetaVdw + (1-Beta)dW hesaplamasına benzer.

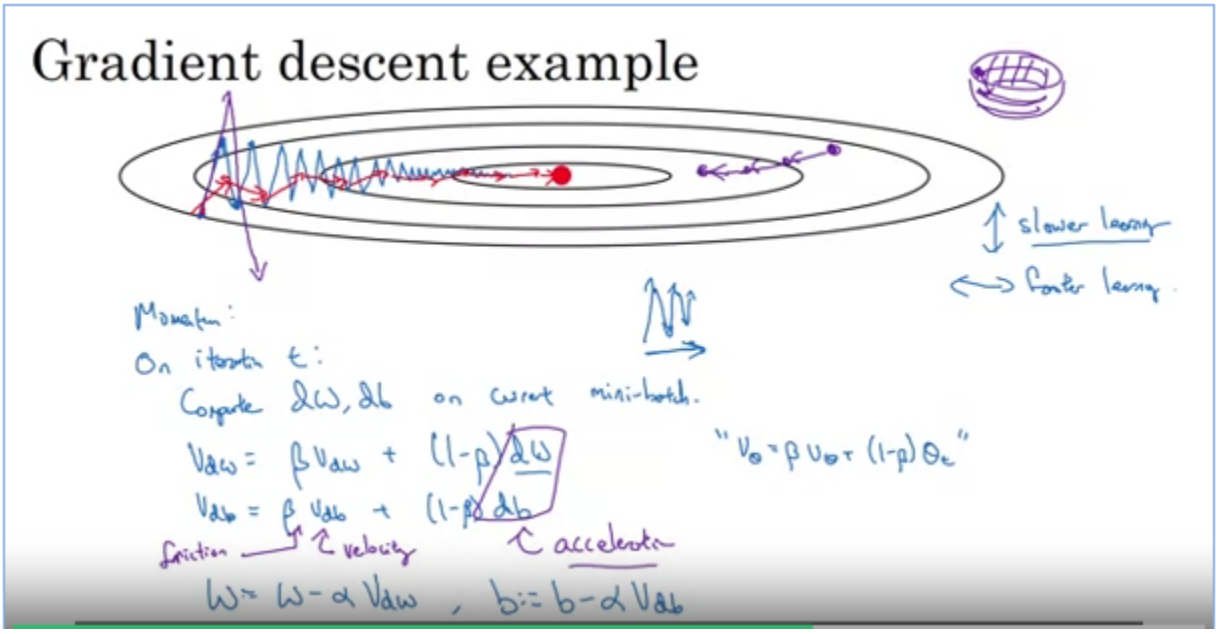
Aynı şekilde, Vdb = BetaVdb + (1-beta)db

alfa = öğrenme oranları çarpımı(learning rate)

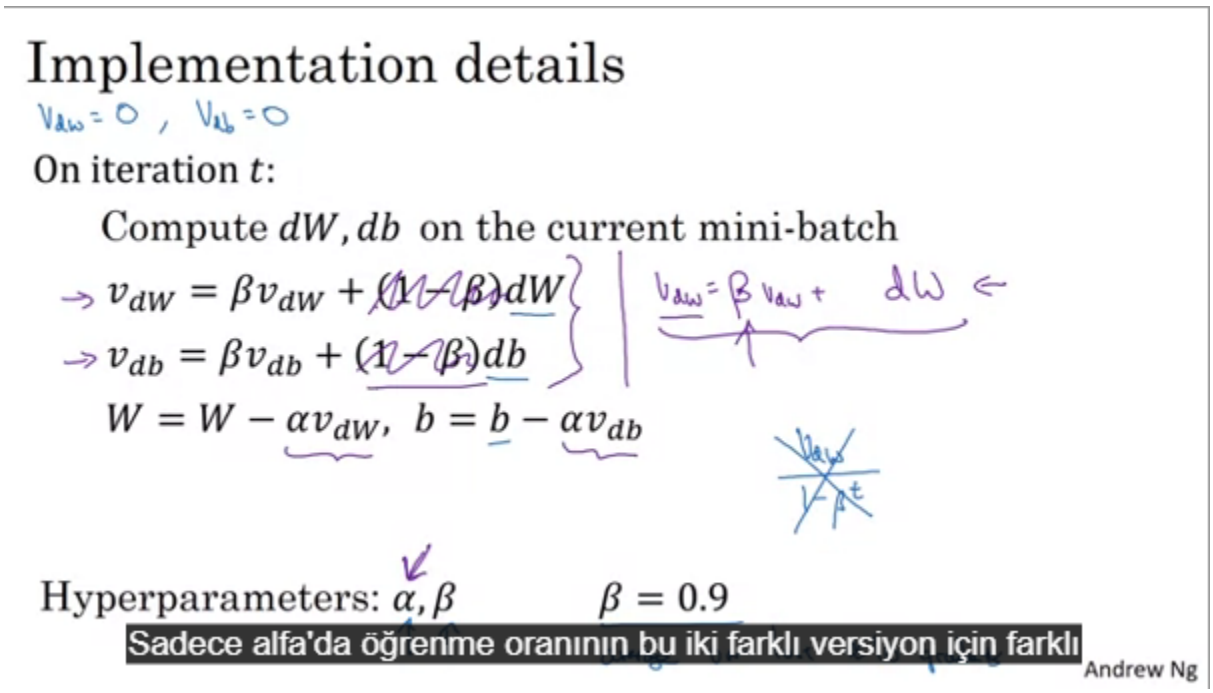
W = W - alfaVdw b = b-alfaVdb

Bu durum gradient descent basamaklarını yavaşlatır.

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



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 yayın değeri 0.9dur. Vdw ve vdb sıfırlar matrisi olarak başlatılır. Boyuutları aynıdır. 1-beta değeri oldukça 1e yakın olduğu için kullanılmadığı alanlar olabilir. Bu sadece alfanın öğrenme oranını etkiler.



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

Sdw = BetaSdw + (1-Beta)dW^2

Sdb = BetaSdb + (1-Beta)db^2

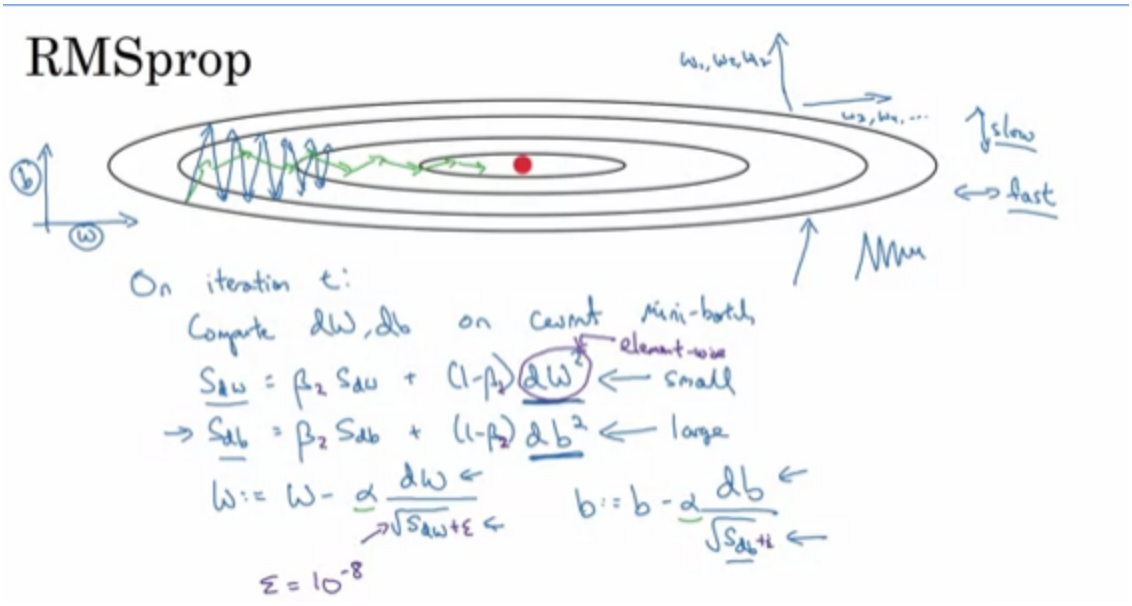
Takiben, RMSprop paremetreleri şu şekilde günceller: W, W-a(öğrenme hızı) olarak güncellenir ve W = W-alfaXdw/kökiçindeSdw olur.

b = b-alfaXdb/kökiçindeSdb Peki nasıl çaılışır.

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.

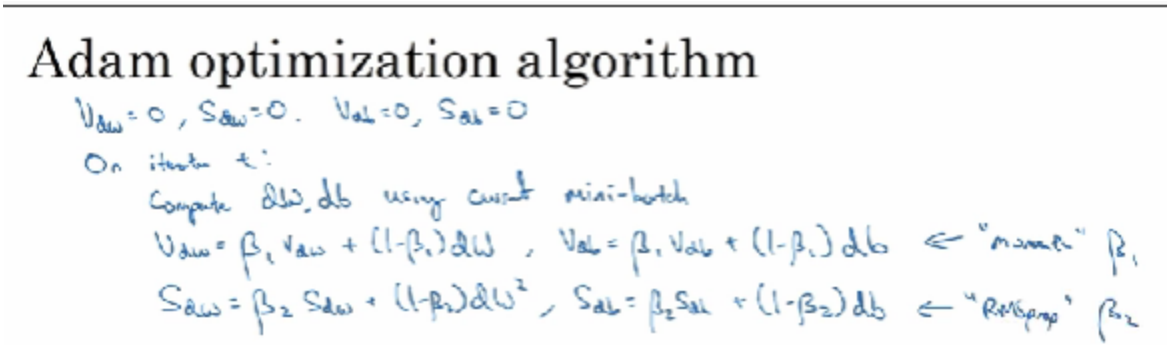
Paydaya epsilon eklenir sıfırdan uzak olması için.



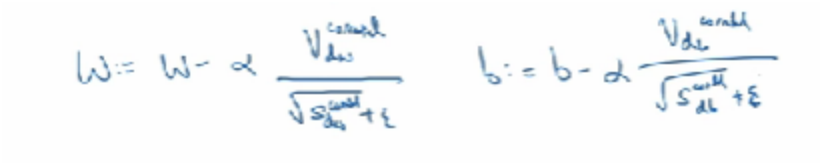
Adam Optimization Algorithm

Temelinde momentum ve RMSprop'u birlikte kullanır.

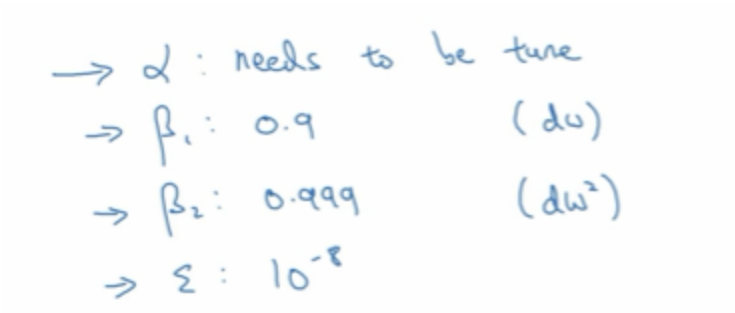
Beta1 momentum, Beta2 RMSprop'undur.



Güncellenmiş w değerine göre ikisinin birleşimi şu şekilde gözükür.



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



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

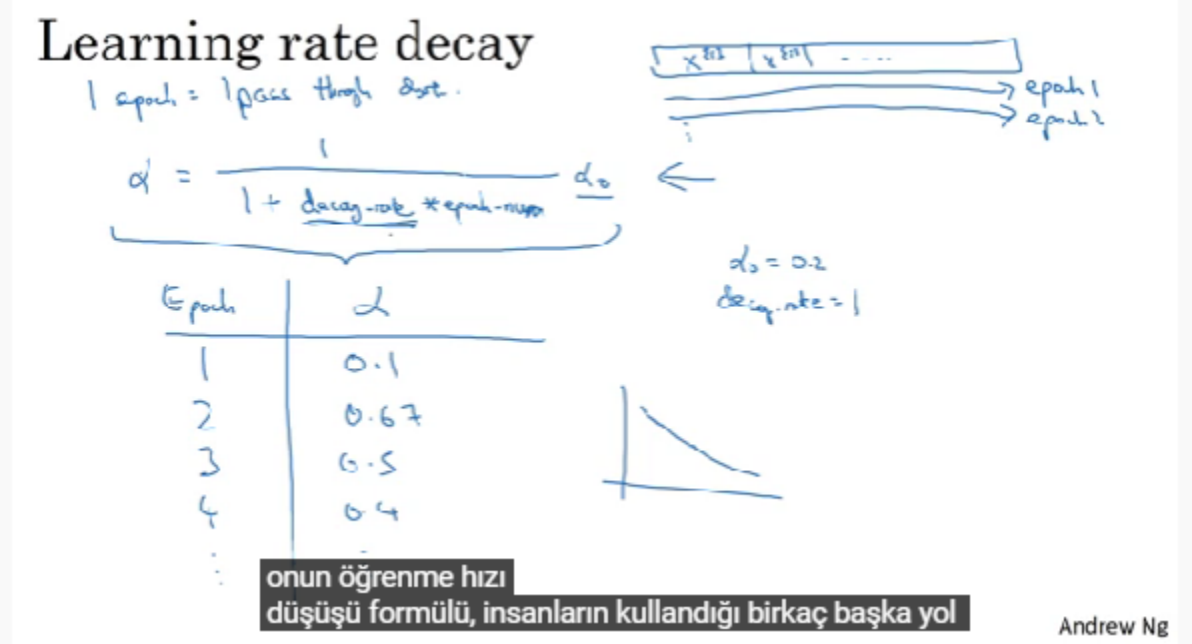
The formula for learning rate decay is:

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

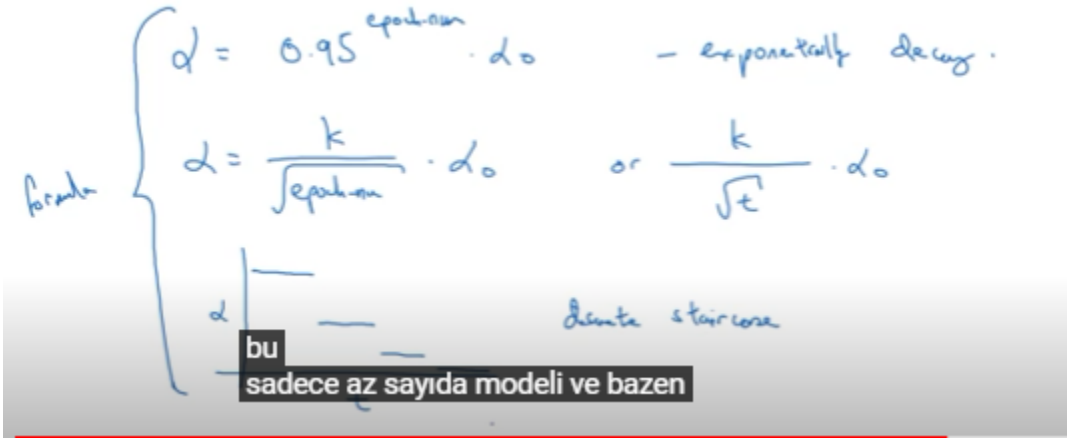
Learning Rate Decay

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.

Tüm minibatchlerde yani tüm eğitim setinde turlamasına epoch denir.



Başka rate decay methotlarda vardır.

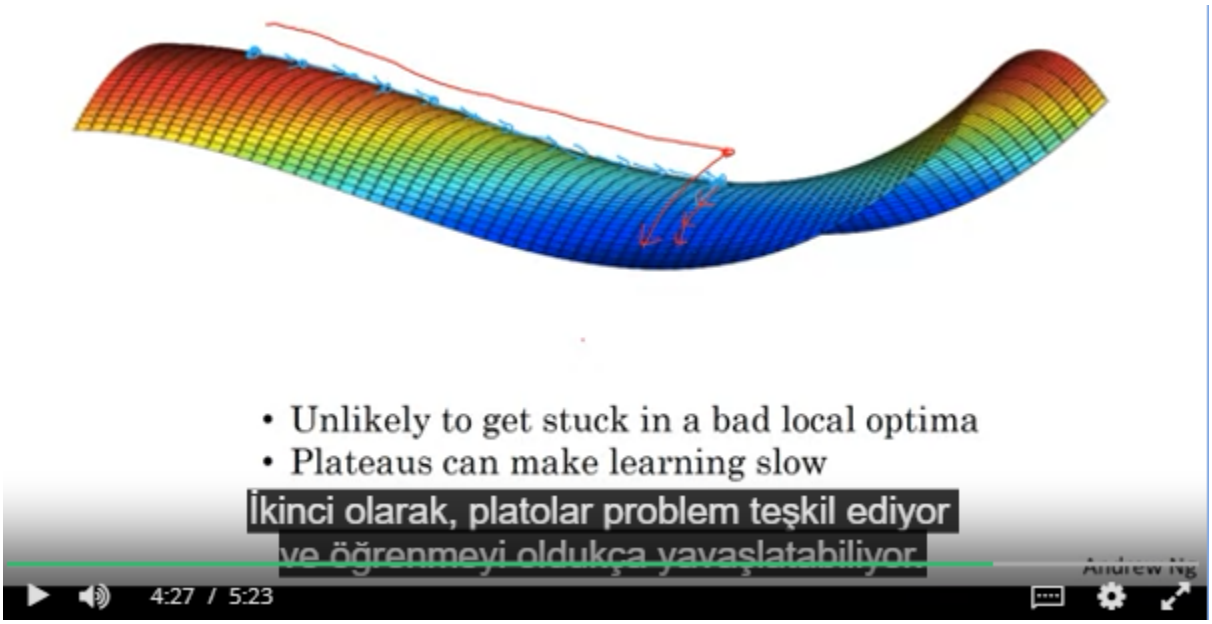


The Problem of Local Optima

Yerel optimum nedir doğrusu nedir?

Ekranda bir sürü yerel optimum gözürüyoruz. Gradient descent ya da herhangi bir optimiazasyon 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 gözükebilir. 20bin parametrenin olduğu durumda eyer noktasıyla karşılaşma ihtimaliniz çok yüksek.



Bu durumda adama, rmsprop ve momentum işimize yarıyor.