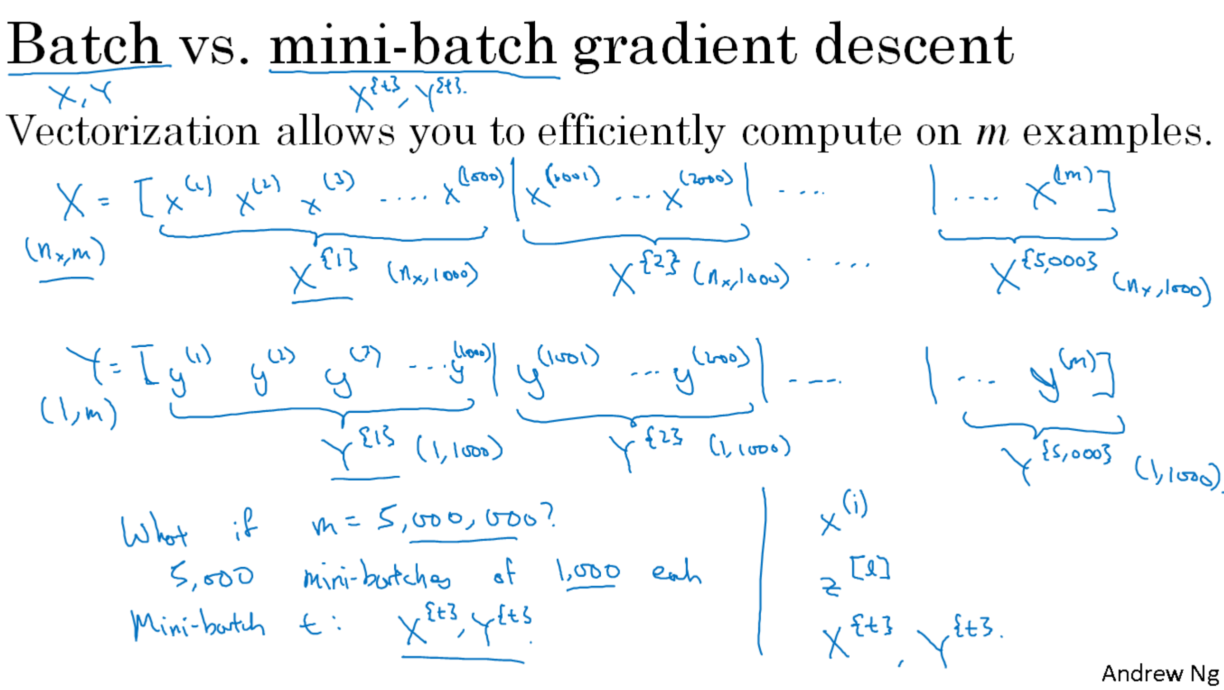
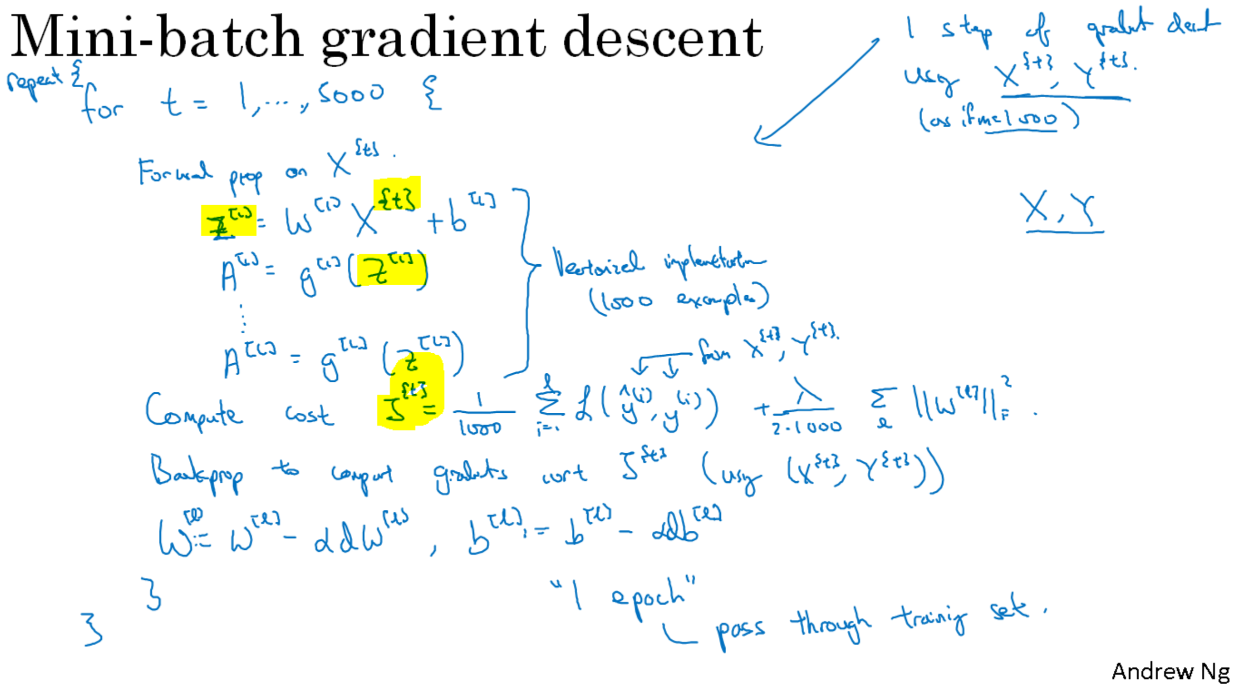
[2주차] Mini-batch gradient descent



여기서는 중괄호 t를 이용해서 mini bacthes들을 인덱싱 합니다.



미니 배치 기울기 강하를 트레이닝 세트에서 실행하기 위해서는, t=1 에서 5000까지 실행합니다. 각각 1000개씩 가지고 있는 미니 배치가 5000개 있었죠. for loop 내부에서는 무엇을 할 것이냐면, xt, yt를 이용해서 기울기 강하의 한 단계를 도입할 것입니다. 이것은 마치 트레이닝 세트의 크기가 1000개의 예시가 있는 것과 비슷합니다.

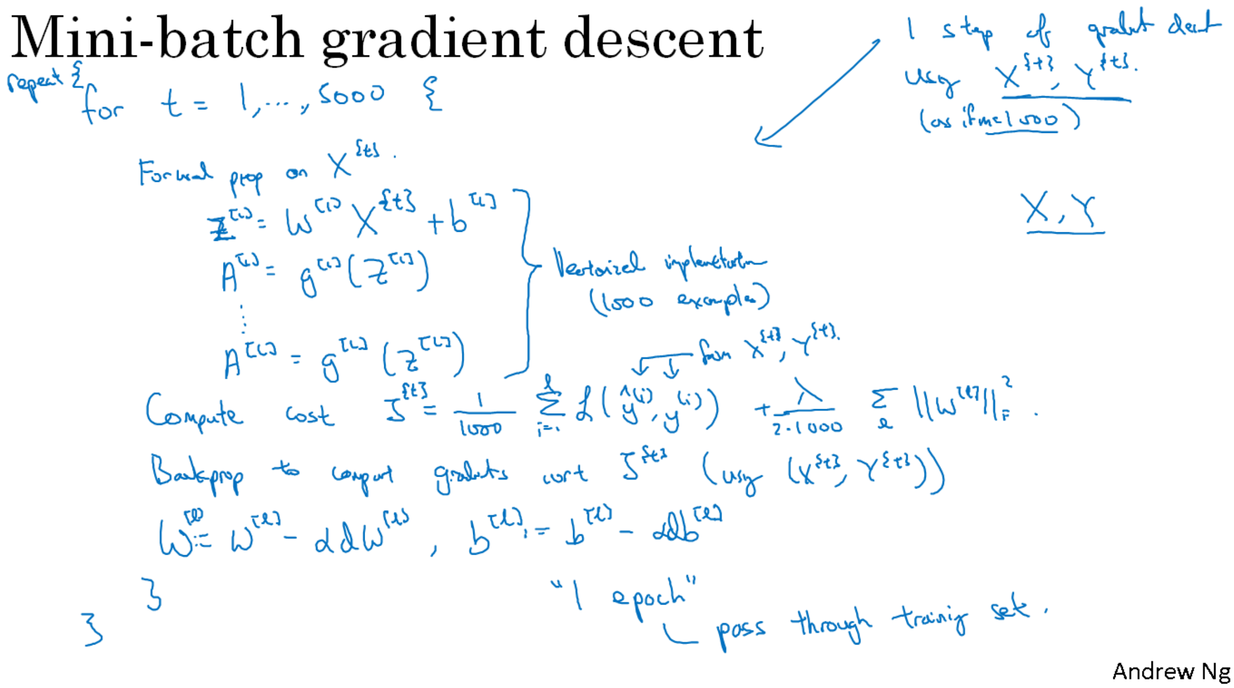
여러분이 별개의 for loop을 1000개의 예시에 갖게 하는 것이 아닌, **벡터화를 통해 1000개의 예시를 한번에 처리하는 것**과 같습니다.

이전에는 여기가 그냥 X였죠? 하지만 이제 여러분은 전체 트레이닝 세트를 처리하는 것이 아니라, 첫번째 미니 배치만 처리하는 것입니다. 그렇기 때문에 여기가 xT가 되죠. 미니 배치 T를 처리하는 경우에 말이죠.

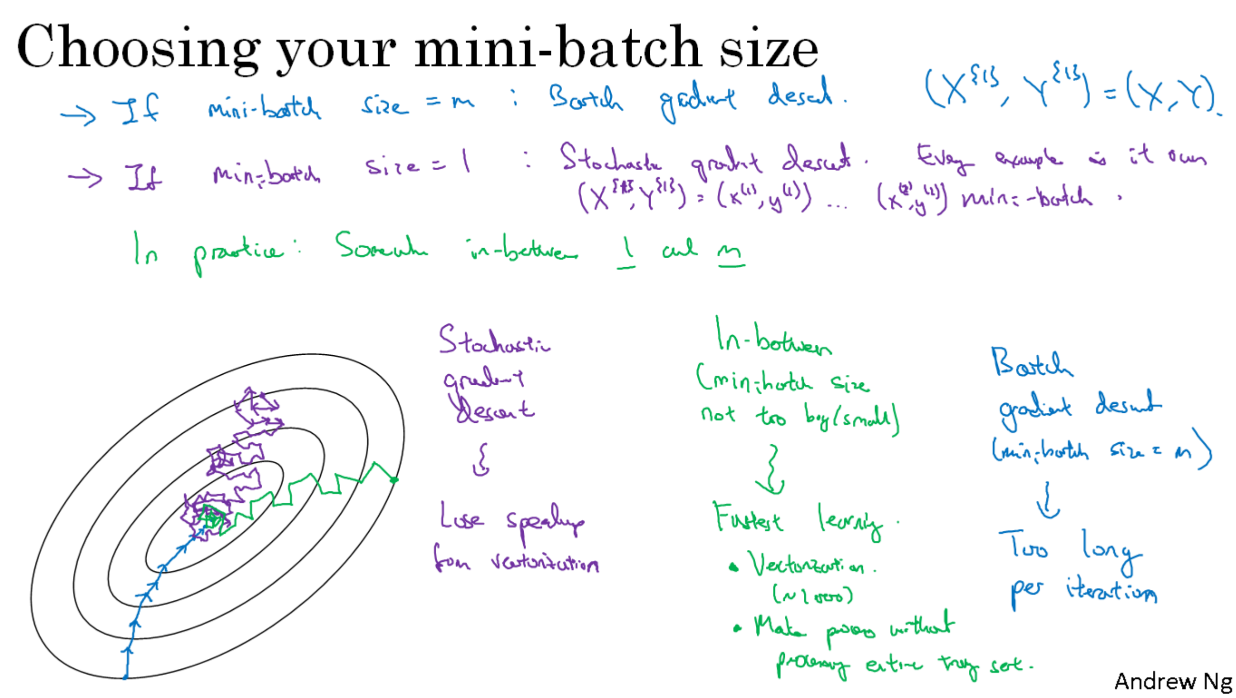
그러면 G1의 Z1의 값을 갖게 되겠죠. 여기는 대문자 Z죠, 벡터화 도입이기 때문입니다. 이어서 AL값이 될 때까지 진행하는데요, 여기는 GL의 ZL입니다. 그리고 이 값이 예측 값이 되는 것입니다.

다음으로는, **J 비용함수를 산출**합니다. 여기서는 이 함수를 1 /1000으로 적을 것인데요, 그 이유는 1000이 여러분의 작은 트레이닝 세트의 크기이기 때문입니다.

여기 제가 쓴 코드는 1 epoch 트레이닝을 한다고 표현하기도 하는데요, epoch이라는 용어는 트레이닝 세트로 1번 통과한다는 뜻입니다. 반면에 **배치 기울기 강하에서는 한번 트레이닝을 통과하는 것이 오로지 한번의 기울기 강하 절차를 밟게 해줍니다.**



조금 더 noisy한 이유는 X{1}, Y{1}이 조금 더 쉬운 미니 배치여서 비용이 조금 더 낮기 때문에 그럴 수 있는데요, 하지만 우연으로 X{2},Y{2}가 어려운 미니 배치일 수도 있죠. mislabel 된 예시도 있을 수도 있겠죠, 이런 경우 비용이 조금 더 높겠습니다.

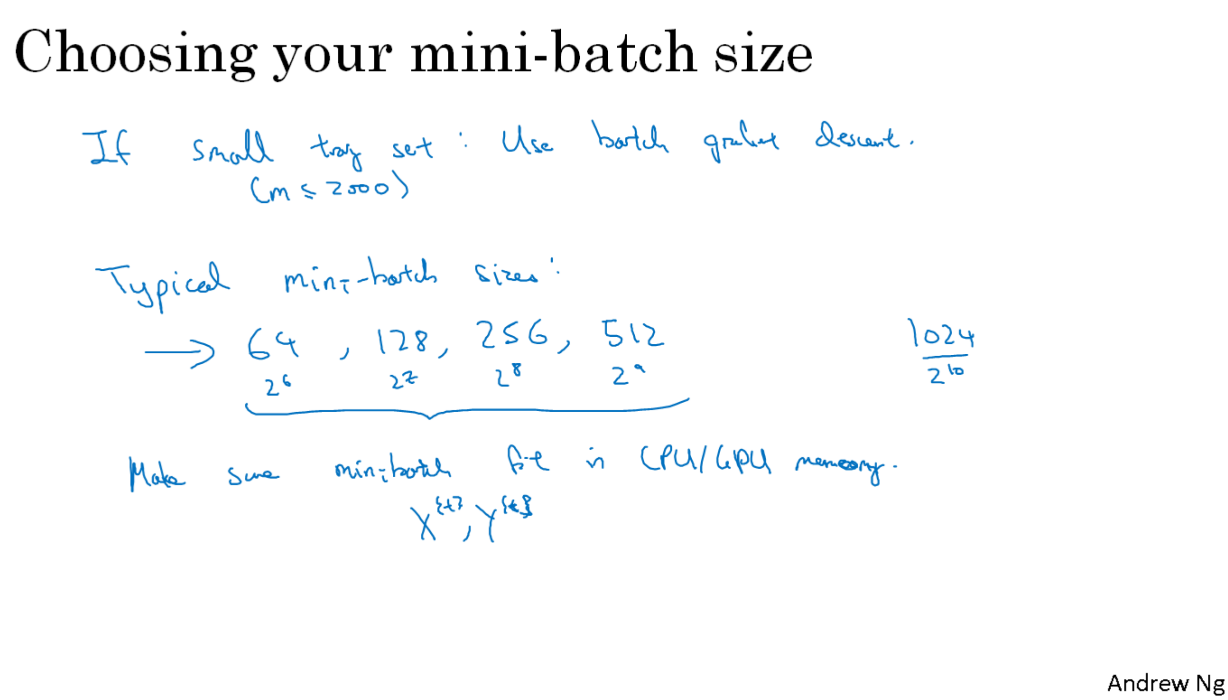


1. 미니 배치 사이즈를 m으로 설정하는 것은 단순히 배치 기울기 강하를 줍니다.
2. stochastic 기울기 강하라는 알고리즘에서 각각의 예시는 그들의 미니 배치 입니다.

이럴 경우에는, 첫번째 미니 배치를 봅니다. 즉, X{1}, Y{1}인데요, 그렇지만 미니 배치 사이즈가 1인 경우, 첫번째 트레이닝 예시 밖에 없는데요, 즉 첫번째 트레이닝 샘플로 기울기 강하를 가져야 합니다. 다음으로 두번째 미니 배치를 봅니다. 두번째는 그냥 두번째 트레이닝 샘플인데요, 이 값에 기울기 강하 step를 적용합니다. 그리고 이어서 세번째 트레이닝 예시도 그렇게 합니다. 하나의 트레이닝 샘플씩 보는 것입니다.

배치 기울기 강하 는 여기쯤에서 시작할텐데요, 이 경우, 낮은 noise와 큰 step를 가질 수 있을 것입니다. 그러면 계속 최소값을 향할 수 있을 것입니다. stochastic 기울기 강하 와는 반대로, 다른 점을 한번 지정해보겠습니다. 그러면 한번의 iteration마다 한개의 트레이닝 예시로 gradient descent를 하는 것인데요,

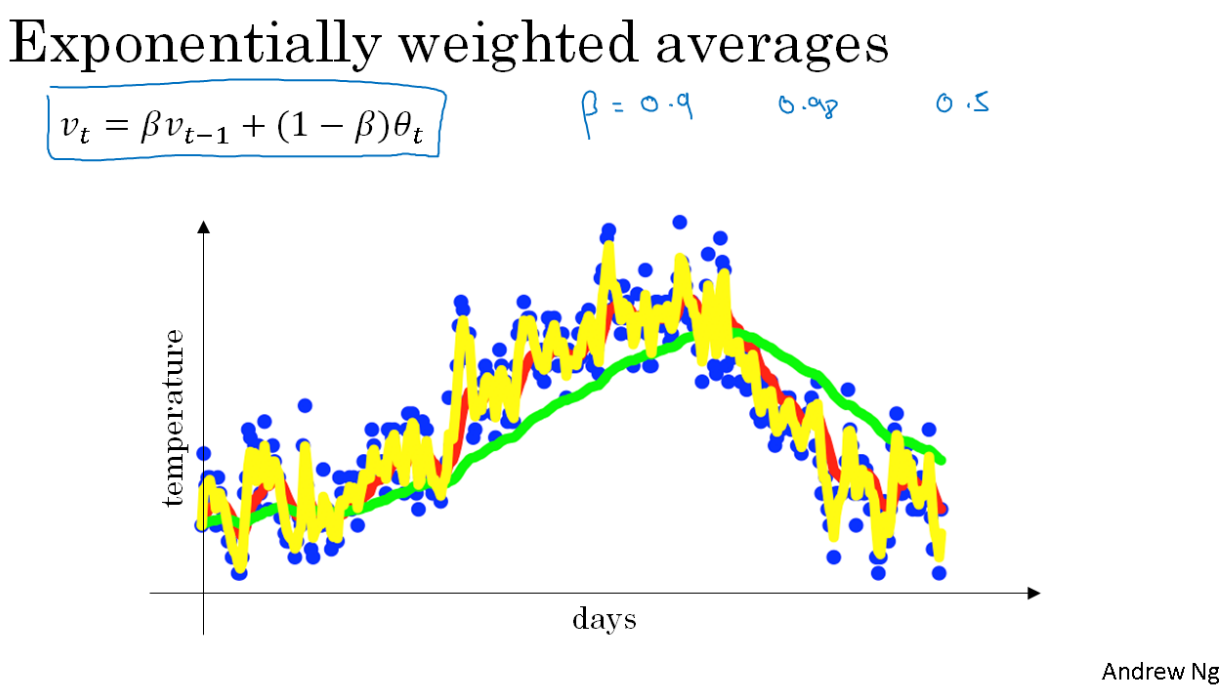
그러면 만약 미니 배치 사이즈가 m이 아니고, 1이 아니며, 그 사이 값을 가지려면, 어떤 것을 고를 수 있을까요?

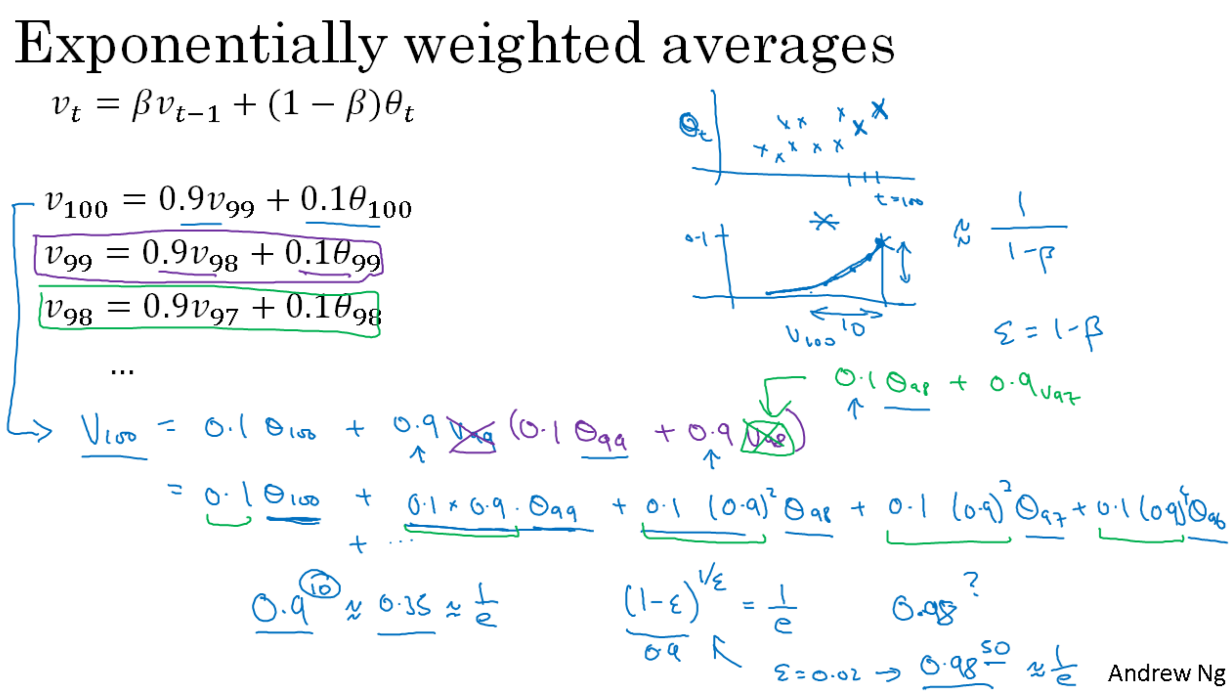


조금 더 큰 트레이닝 세트의 경우, 전형적인 미니 배치 사이즈는 64에서 512가 대표적인데요, 컴퓨터 메모리가 배치되어 있는 방식과 접속되는 방법에 따라, 가끔씩 그 코드가 2의 지수값을 가질때 더 빨리 실행됩니다. 64는 그러면 2의 6승이구여, 2의 7승, 2의 8승, 2의 9승, 저는 그래서 자주 미니 배치 사이즈를 어떤값의 2승으로 도입합니다. 이번 비디오에서는 제가 미니 배치 사이즈 1000을 사용했는데요, 만약 정말로 그렇게 하고 싶은 경우엔, 저는 여러분께서 1024를 쓰시는 것을 추천 드립니다. 이 값은 2의 10승이죠. 조금 흔하진 않지만, 미니 배치 사이즈가 1024인게 있긴 합니다. 여기 이 범위의 미니 배치 사이즈가 조금 더 흔하긴 합니다. 마지막 팁으로는,

모든 X{t}, Y{t} 값이 CPU/GPU 메모리에 들어가게 하도록 하는 것입니다.

지수가중 평균법



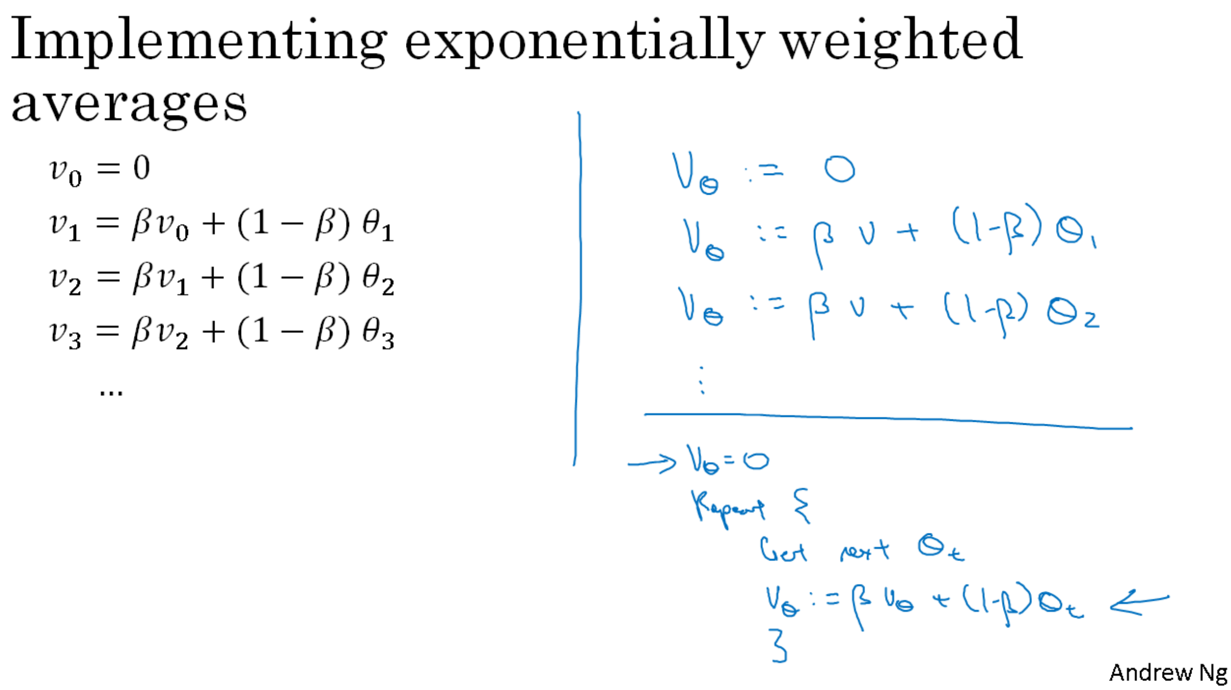


일별 평균 기온을 어떻게 산출하는지 한번 이해해보도록 하겠습니다. 여기는 아까봤던 그 공식인데요, 베타의 값을 0.9로 설정하고, 이 값에 상응하는 몇개의 식을 더 적어보겠습니다. 만약에 T의 값이 0에서 1로 그리고 2에서 3으로 T가 늘어날때의 값을 분석하는 반면에 여기서는 **T의 값이 감소할때 의 경우를 적었습니다**. 여기 첫번째 식을 보겠습니다. V100이 실제로는 어떤 의미인지 보겠습니다. V100은 어떻게 되냐면, 여기 2개의 항을 서로 맞바꿔 볼텐데요, 여기는 0.1 곱하기 쎄타 100에 더하기 0.9 곱하기 전날의 기온을 해줍니다. 그러면 V99는 또 어떻게 되죠? 여기 식에 이 식을 대입해보겠습니다. 이 값은 그냥 0.1 곱하기 쎄타 99, 그리고 다시 2개의 항을 서로 맞바꿨습니다. 더하기 0.9 곱하기 V98입니다. 그러면 또 V98은 또 어떻게 되죠? 자, 여기서 구할 수 있는데요, 이곳에다가 대입시키면 됩니다.

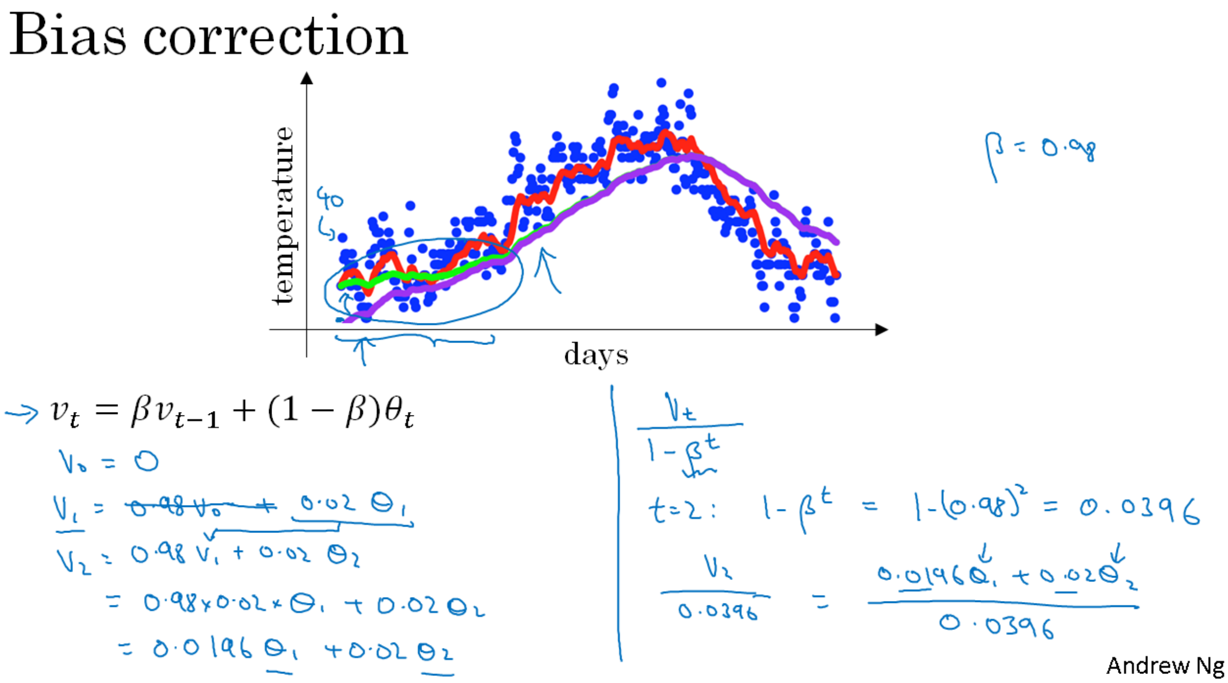
이런 방법으로 합을 구하면 되고, 쎄타100에 대한 가중평균치 인데요, 그 해의 100일간의 온도를 계산한 V100 까지 반영하여 현재 시점의 기온을 뜻합니다. 이것은 쎄타 100과 쎄타 99 쎄타 98, 쎄타 97, 쎄타 96 등등의 합을 나타내는 값입니다

나중에 제가 설명을 드리겠지만, 여기에 있는 모든 계수들을 더하면 1이 되거나 1과 매우 근접한 값을 갖게됩니다. **편이보정**(bias correction)이라는 값과 이것에 대한 내용은 다음 비디오에서 다루겠지만, 이런 점 때문에 지수 가중 평균이라고 합니다.

베타가 0.9인 경우, 이것은 마치 여러분이 직전 10일 간의 기온만 집중하여 지수 가중 평균을 구하는 것과 같은데요, 약 10일 정도 이후 가중치가 현재 날짜 가중치 3분의 1보다 약간 더 감소하기 때문에 그렇습니다. 반면에 베타의 값이 0.98이였다고 하면, 그러면, 0.98의 몇승을 해야 이 값이 매우 작은 값이 될까요? 알고보니 0.98의 50승이면 대략 1/e와 비슷한 값이 됩니다. 그렇기 때문에 가중치는 첫 50일에는 1/e 보다 큰 값이 되었다가, 그 후엔, 꽤 빠른 속도로 감소할 것입니다.



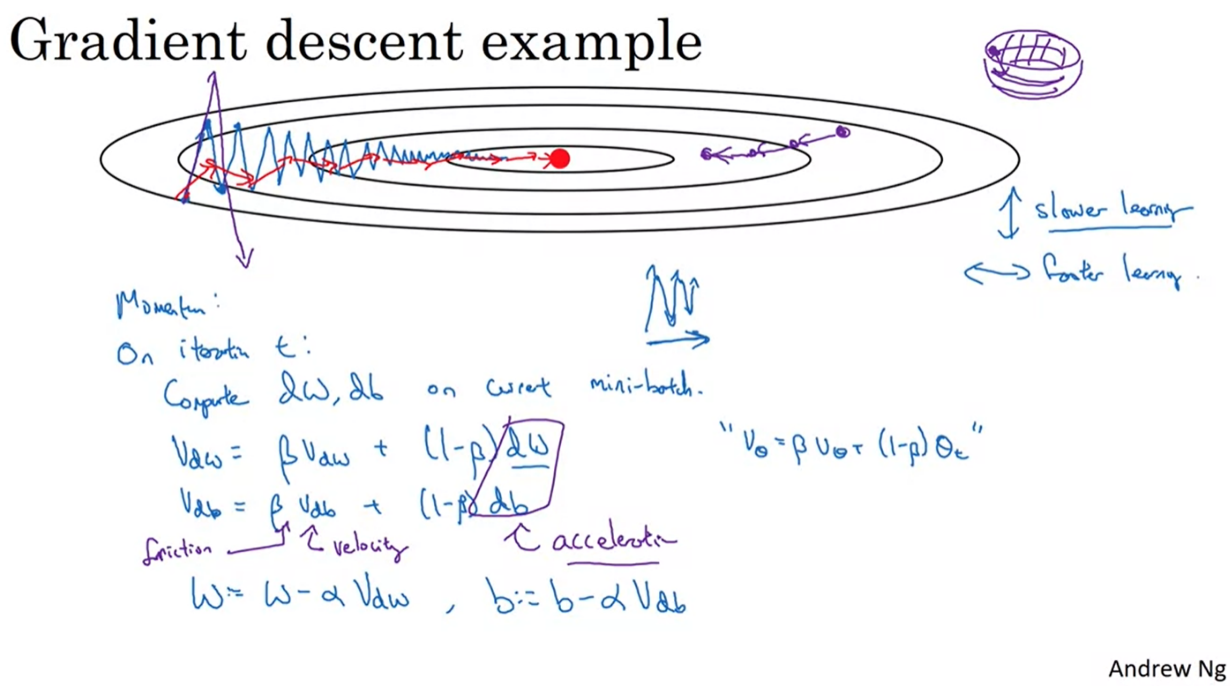
V를 초기화한 값을 0이라 하고, 첫번째 날은, V의 값을 베타 곱하기 V 더하기 1 빼기 베타 곱하기 쎄타 1로 설정할 것입니다. 그 다음날에는, V를 업데이트하여 베타 곱하기 V 더하기 1빼기 베타, 곱하기 쎄타2를 할 것입니다. 이렇게 계속 이어지겠죠. 어떤 것은 이런 V 아래첨자 쎄타로 표기할텐데요, 쎄타인 파라미터에 대해 지수 가중 평균을 구하고 있다는 의미를 나타냅니다.

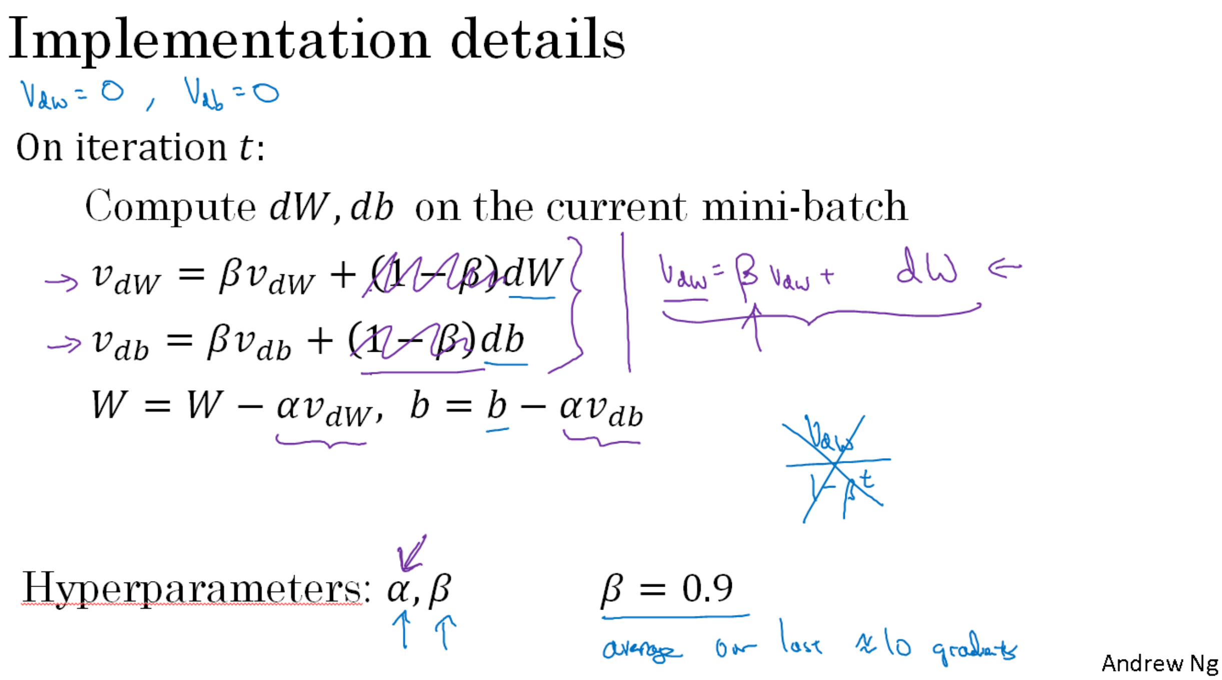


이 그래프는 0.98인 경우입니다. 베타가 0.98일때 말이죠. 대신 이렇게 생긴 보라색 곡선을 갖게됩니다. 여러분도 보이시겠지만 보라색 곡선은 굉장히 낮은데서 시작합니다. 이것이 끼치는 영향을 보겠습니다. **우리가 이동평균을 구현할 때, v0 = 0으로 초기화합니다.** 그리고 v1 = 0.98 v0 + 0.02 쎄타 1 이 됩니다. 그러나 v0은 0이기 때문에 이 항은 그냥 사라지게 됩니다. v1은 그러면 0.02 곱하기 쎄타1입니다. 그렇기 때문에 첫째 날의 온도가, 예를 들어, 화씨 40도 인 경우, v1은 0.02 곱하기 40 이므로 8이 됩니다. 그래서, 훨씬 더 작은 값이 여기서 나오게 됩니다. 그러면 첫째 날 온도로 좋은 추정치가 아니죠. v2는 0.98 곱하기 v1, 더하기 0.02 곱하기 쎄타 2 이고, v1은 이것을 아래로 대입해서 곱하면 v2는 0.98 곱하기 0.02 곱하기 쎄타 1 더하기 0.02 곱하기 쎄타2가 됩니다. 그러면 0.0196 쎄타 1 더하기 0.02 쎄타 2 이고요. 그리고 다시 쎄타1과 쎄타2가 양수라 가정하면, v2는 쎄타1이나 쎄타2보다 훨씬 작은 값이 될 것 입니다. **그래서, v2는 그 해의 첫 이틀간 기온에 대한 좋은 추정치가 아닙니다.**

이 추정치를 더 낫게 보완하는 방법이 있는데요, 더 정확하게 만들어 주는 것인데. 특히 추정 초반 부분에 대한 것입니다. **Vt 대신에 Vt 나누기 1 빼기 베타의 t 승을 사용**하는 것인데, t는 이 시점의 현재 데이터 입니다. 구체적인 예를 살펴보죠. t의 값이 2일 때, 1 빼기 베타의 t승은 1-0.98 제곱이고, 이 값은 0.0396입니다. 그러므로 두째 날의 기온 추정치는 v2 나누기 0.0396이며, 이 값은 0.0196 곱하기 쎄타1 더하기 0.02 곱하기 쎄타 2가 됩니다. 보시면, 여기 이 둘에 분모로 0.0396으로 하고, 이것이 쎄타1과 쎄타2의 가중평균이 되서, 이 바이어스를 제거됩니다. 그러므로 t의 값이 커지면서 베타의 t승은 0으로 수렴하고, t 값이 충분히 커

지면, 바이어스 보정 영향은 없게 됩니다. 그러므로 t의 값이 큰 경우, 보라색 선과 초록색 선은 훨씬 많이 겹치게 됩니다. 이 추정의 초반부에는 추정치가 서서히 맞춰 들어 갈 때, 바이어스 보정을 통해서 기온의 더 나은 추정치를 얻을 수 있고요. 바이어스 보정으로 보라색 선이 초록색 선으로 가게 됩니다.

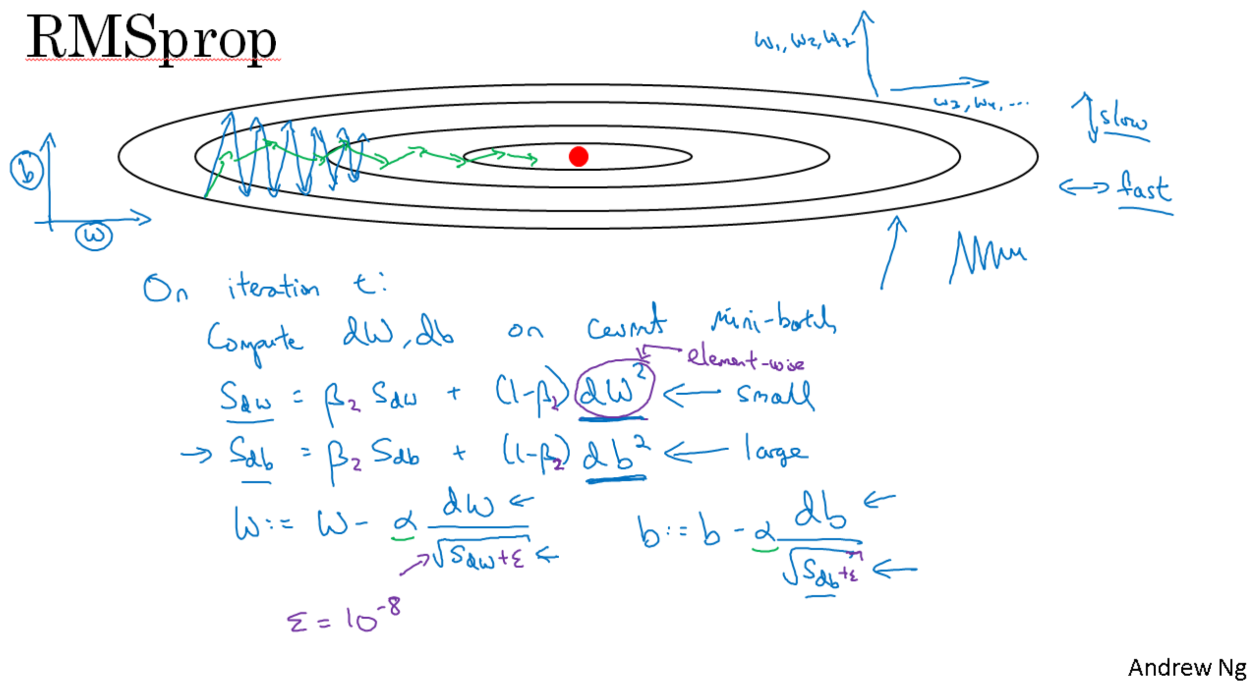




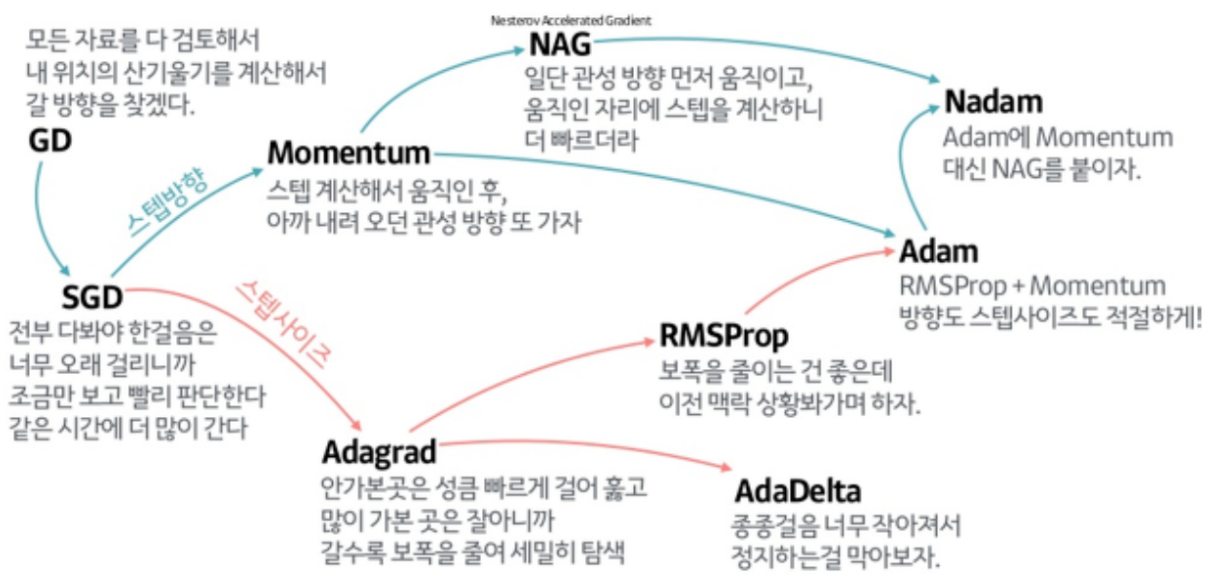
마지막으로 여러분이 기울기 강하 와 모멘텀에 대한 학술을 읽으시면, 이 항이 생략되는 것을 보실텐데요 1 빼기 베타 부분말이죠. 그렇게해서 vdW = 베타 vdw 더하기 dW로 남게됩니다. 여기 보라색 버전을 쓰면서 나타나는 최종 효과는 vdW가 1 빼기 1베타로 스케일 된다는 것입니다. 또는, 더 정확히 얘기하면 1 나누기 1-베타로 말입니다. 그러므로 여러분이 기울기 강하 갱신을 할때 알파의 값은 그에 상응하는 1 나누기 1 빼기 베타로 변해야 합니다.

RMSprop

기울기 강하를 도입하는 경우엔, 세로로 아주 큰 변동이 일어날 수 있습니다. 가로줄로 나아가려고 할때 말이죠.



RMSprop
slow
fast
On iteration t:
Compute dW, db on current mini-batch
element-wise
S_dW = β2 S_dW + (1-β2) dW^2 ← small
S_db = β2 S_db + (1-β2) db^2 ← large
W := W - α dW / √(S_dW+ε)
b := b - α db / √(S_db+ε)
ε = 10^-8
Andrew Ng



모든 자료를 다 검토해서
내 위치의 산기울기를 계산해서
갈 방향을 찾겠다.
GD
Nesterov Accelerated Gradient
NAG
일단 관성 방향 먼저 움직이고,
움직인 자리에 스텝을 계산하니
더 빠르더라
Nadam
Adam에 Momentum
대신 NAG를 붙이자.
Momentum
스텝 계산해서 움직인 후,
아까 내려 오던 관성 방향 또 가자
스텝방향
SGD
전부 다봐야 한걸음은
너무 오래 걸리니까
조금만 보고 빨리 판단한다
같은 시간에 더 많이 간다
스텝사이즈
Adam
RMSProp + Momentum
방향도 스텝사이즈도 적절하게!
RMSProp
보폭을 줄이는 건 좋은데
이전 맥락 상황봐가며 하자.
Adagrad
안가본곳은 성큼 빠르게 걸어 훑고
많이 가본 곳은 잘아니까
갈수록 보폭을 줄여 세밀히 탐색
AdaDelta
종종걸음 너무 작아져서
정지하는걸 막아보자.

`RMSProp`

- AdaGrad는 스텝이 많이 진행되면 누적치 h_n이 너무 커져서 학습률이 너무 작아져 학습이 거의 되지 않는 문제가 있음
- 이를 보완하기 위해 RMSProp은 Adagrad와 달리 기울기를 단순 누적 x
- AdaGrad보다 최근 값을 더 잘 반영하기 위해 최근 값과 이전 값에 각각 가중치를 주어 계산하는 방법
- **지수 이동 평균** 사용
- 쉽게 말하면, 내분인데, 더 중요한 변수에 내분점을 가까이 찍어서 가중치를 더 주는 것
- γ가 크다→빨간점에 더 가까이→ 과거 중요하게 생각
- γ가 작다→$(1-\gamma)$가 크다→파란점에 더 가까이→ 현재 중요하게 생각

$$\mathbf{h}_n = \gamma \boxed{\mathbf{h}_{n-1}} + (1-\gamma)\boxed{\nabla f(\mathbf{x}_n) \odot \nabla f(\mathbf{x}_n)}, \qquad \mathbf{h}_{-1} = \mathbf{0}$$

Adam : momentom + RMSprop

Adam optimization algorithm

$V_{dw}=0,\ S_{dw}=0.\quad V_{db}=0,\ S_{db}=0$

On iteration t:

Compute dW, db using current mini-batch

$V_{dw}=\beta_1 V_{dw}+(1-\beta_1)dW, \quad V_{db}=\beta_1 V_{db}+(1-\beta_1)db \leftarrow$ "momentum" β_1

$S_{dw}=\beta_2 S_{dw}+(1-\beta_2)dW^2, \quad S_{db}=\beta_2 S_{db}+(1-\beta_2)db \leftarrow$ "RMSprop" β_2

yhat = np.array([.9, 0.2, 0.1, .4, .9])

$V_{dw}^{corrected}=V_{dw}/(1-\beta_1^t), \quad V_{db}^{corrected}=V_{db}/(1-\beta_1^t)$

$S_{dw}^{corrected}=S_{dw}/(1-\beta_2^t), \quad S_{db}^{corrected}=S_{db}/(1-\beta_2^t)$

$$W := W-\alpha\frac{V_{dw}^{corrected}}{\sqrt{S_{dw}^{corrected}}+\varepsilon} \qquad b := b-\alpha\frac{V_{db}^{corrected}}{\sqrt{S_{db}^{corrected}}+\varepsilon}$$

Andrew Ng

미분을 계산할텐데요. dw와 db를 현재 미니 배치를 이용해 계산하고 보통, 미니 배치 기울기 강하를 이용하면 되고요.

그리고 모멘텀, 지수 가중 평균을 쓸텐데요. **Vdw = ß인데, 이제부터는 이 값을 ß1**이라고 쓰고, **RmsProp의 하이퍼 파라미터 ß는 ß2**로 구분해서 사용하겠습니다.

자, 이것은 모멘텀을 구현할 때 했던 것과 완전히 동일한데요 유일한 차이는 하이퍼 파라미터 ß 대신에 ß1이라고 부른 점입니다. 유사하게 Vdb 는 이렇게, 1 빼기 ß1 곱하기 db죠. 그리고 RmsProp도 업데이트 하는데요. 이제 하이퍼 파라미터는 ß2죠. 여기는 플러스 1 빼기 ß2 dw^2 여기서 제곱은 미분 dw를 원소별로 제곱하는 것이고요.

일반적으로 Adam 구현에는, 편향보정도 같이 합니다. 그러므로 v corrected 를 사용할텐데, Corrected(보정된)는 편향 보정된 것을 뜻 합니다. dw는 vdw 나누기 1 빼기 ß1 의 t승, t회째 반복 루프를 진행한 경우에 말이죠. 비슷하게, vdb corrected는 vdb 나누기 1 빼기 ß1 의 t승입니다. 그리고 또 비슷하게, S에도 편향보정을 적용해서 즉, sdw 나누기 1 빼기 ß2 의 T승, 그리고 sdb corrected는 sdb 나누기 1 빼기 ß2 의 T승입니다. 마지막으로 업데이트를 진행합니다. 그러면 W는 W 빼기 알파 곱하기, 만약 모멘텀만 사용하는 경우 Vdw를 사용하거나, Vdw corrected가 될 수도 있겠죠. 이제 RmsProp에 해당하는 부분도 추가하는데요. 그래서, Sdw corrected + 앱실론의 루트로 나눠줍니다. 마찬가지로, 비슷한 공식으로 b 또한 업데이트 될텐데요. Vdb corrected 나누기 루트 S corrected db 더하기 앱실론입니다. 이 알고리즘은 기울기 강하에 모멘텀 효과와 함께 RmsProp을 같이 결합한 것입니다.

Hyperparameters choice:

$\rightarrow \alpha$: needs to be tune

$\rightarrow \beta_1 : 0.9 \quad \rightarrow (dw)$

$\rightarrow \beta_2 : 0.999 \quad \rightarrow (dw^2)$

$\rightarrow \varepsilon : 10^{-8}$

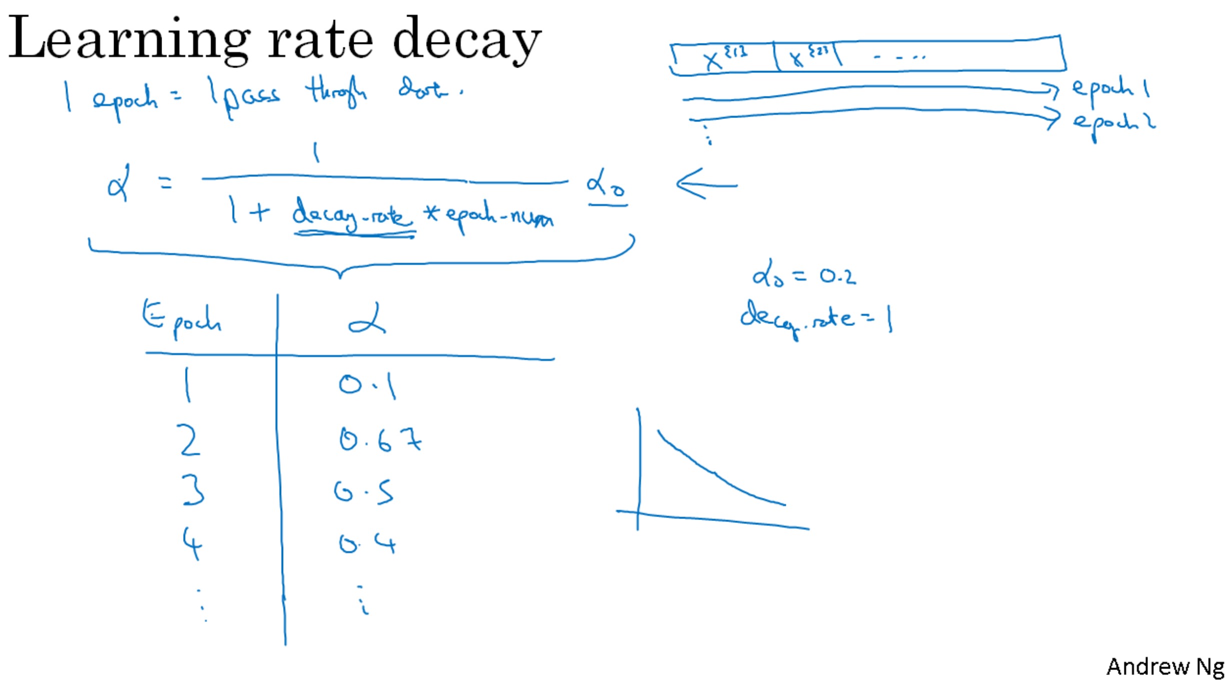
Adam: Adaptive moment estimation

Adam Coates

Andrew Ng

이 알고리즘은 몇개의 하이퍼 파라미터들이 있는데요. 러닝 속도인 하이퍼 파라미터 알파α는 여전히 중요하고, 보통 튜닝이 필요합니다. 그러므로 여러분이 다양한 범위의 값을 시도해서 어떤 값이 적합한지 찾아야 합니다. ß1의 가장 흔한 설정 값은 0.9입니다. 이것은 이동 평균값인데요. dw의 가중평균값인데요, 이것은 모멘텀 같은 성분이고, 하이퍼 파라미터 ß2는, Adam 논문의 저자, Adam 알고리즘 발명자는 0.999를 권장하였습니다. 이것은 dw^2과 또, db^2의 이동평균 (지수 가중 평균)들을 계산하는 것입니다. 그리고, 앱실론ε 값 선택은 그리 중요하지 않습니다.

Adam은 "적응 모멘트 추정" (Adaptive Moment Estimation)을 뜻합니다. **ß1은 미분의 평균값을 산출하는데요. 이것을 1차 모멘트**라고 합니다. 그리고 **ß2는 dW^2의 지수이동평균에 사용되는데, 이것은 2차 모멘트**라고 합니다. 이런 이유에서 adaptive moment estimation 이라는 이름이 나오게 됐습니다.

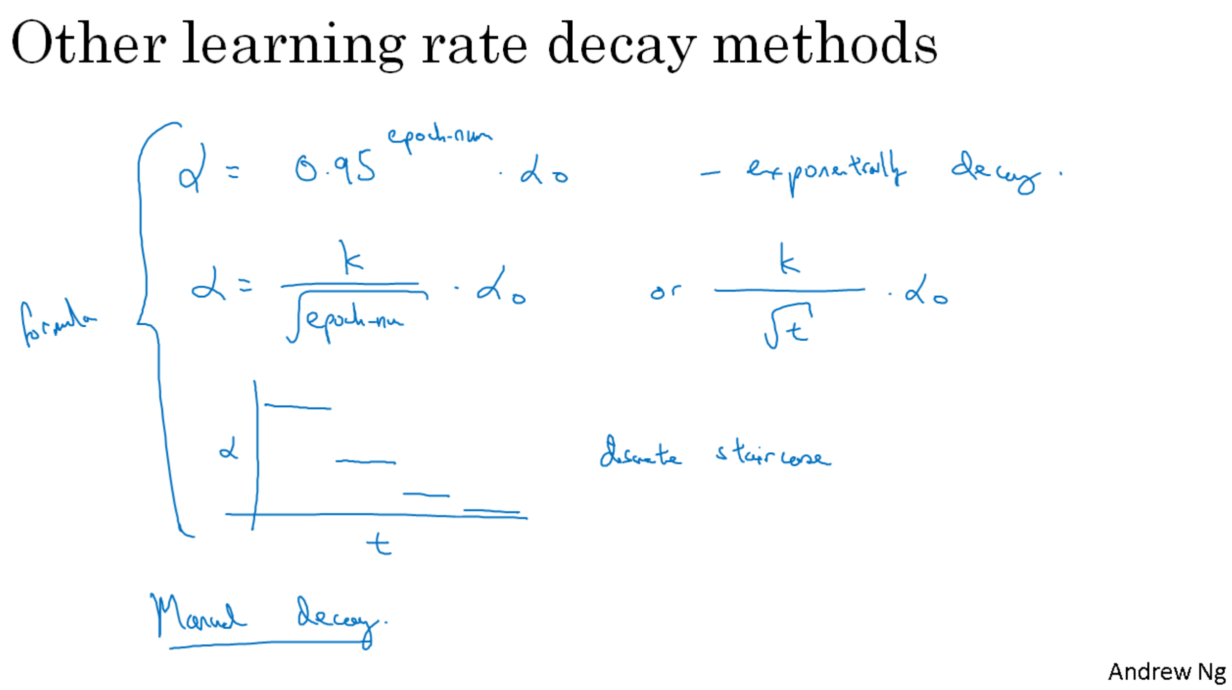


학습 알고리즘을 가속화시키는데 도움이 될 수 있는 것 중 하나는 러닝 레이트를 시간이 가면서 천천히 줄여나가는 것입니다. 이것을 learning rate decay (학습 속도 감쇠법)라고 합니다.

여러 에폭(epoch)을 진행해 보면 즉, 데이터에 여러번 패스를 돌리면 α0 는 0.2 이고, 감쇠비(decay-rate)는 1이면, 그럼 첫번 에폭(epoch)에서 α는 1/ (1 + 1 * α0)입니다. 학습 속도는 0.1이 되겠죠. 이 공식에 넣어보면 나오고요. 감쇠비(decay-rate)는 1이고, 에폭 값(epoch-num)은 1일 때 말이죠. 두번째 에폭(epoch)에서는 러닝 속도가 0.67로 감쇠되었습니다.

세번째에서는 0.5, 네번째에서는 0.4. 등등.

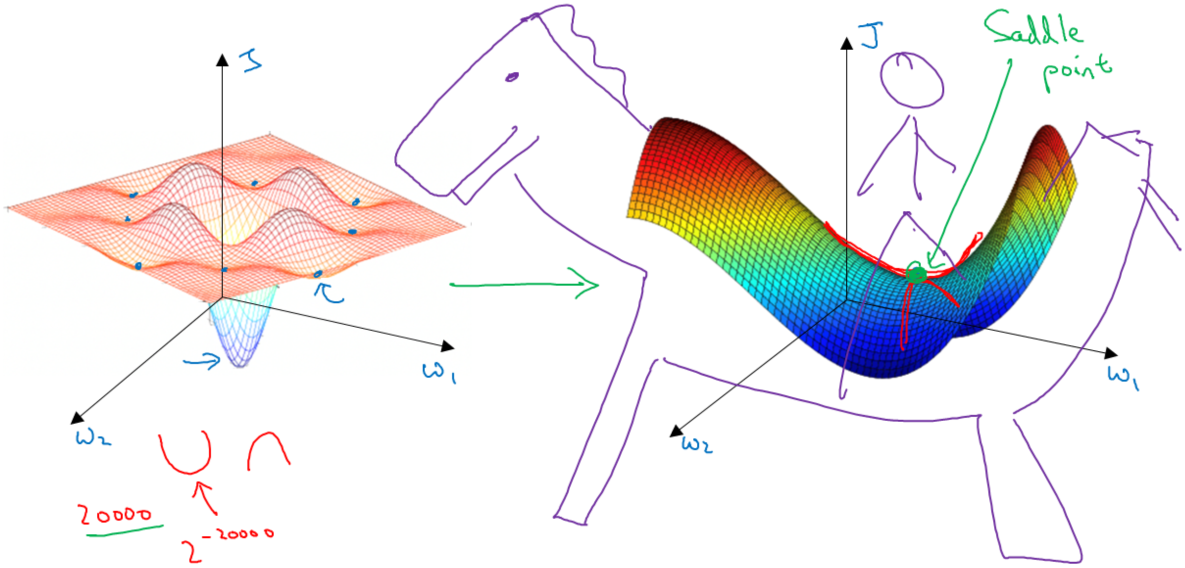
학습 속도는 점진적으로 작아지는 것이 감이 오죠.



사람들이 쓰는 또 다른 공식은 α = 어떤 상수 k / 루트 에폭값에 곱하기 α0. 또는, 또 하나의 하이퍼 파라미터죠. 어떤 상수 k에 나누기 루트 미니 배치의 수 t 곱하기 α0 입니다.

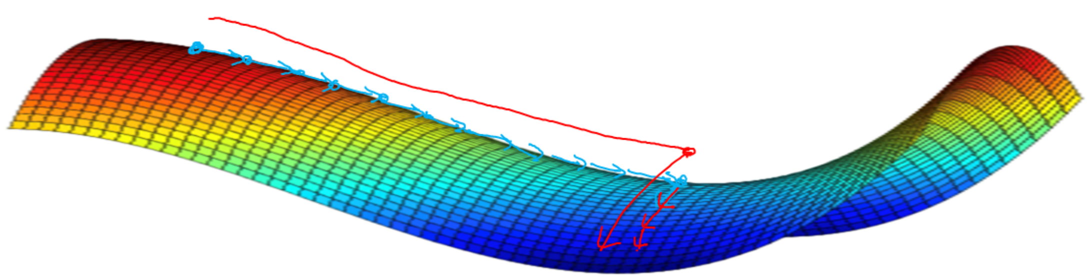
가끔 사람들이 하는 한 가지 방법은 **수작업식 감쇠(manual decay)**입니다.

Local optima in neural networks



Andrew Ng

Problem of plateaus



- Unlikely to get stuck in a bad local optima
- Plateaus can make learning slow

Andrew Ng

국소 최적값이 문제가 되지 않는다면, 어떤 것이 문제가 될까요? plateau 가 러닝속도를 저하시킬 수 있는데요, plateau는 함수 기울기의 값이 0에 근접한 긴 범위를 이야기 합니다.

momentum 도는 RmsProp 또는 Adam 과 같은 알고리즘이 도움을 줄 수 있는 부분입니다. 이런 plateau와 같은 시나리오 경우, 가장 정교한 observation 알고리즘인 Adam과 같은 알고리즘이 plateau를 빠져나오는 속도를 높혀줄 수 있습니다.