딥러닝 2단계 1-2주차
- 초기화와 최적화 -

이승목

2020.05.26.

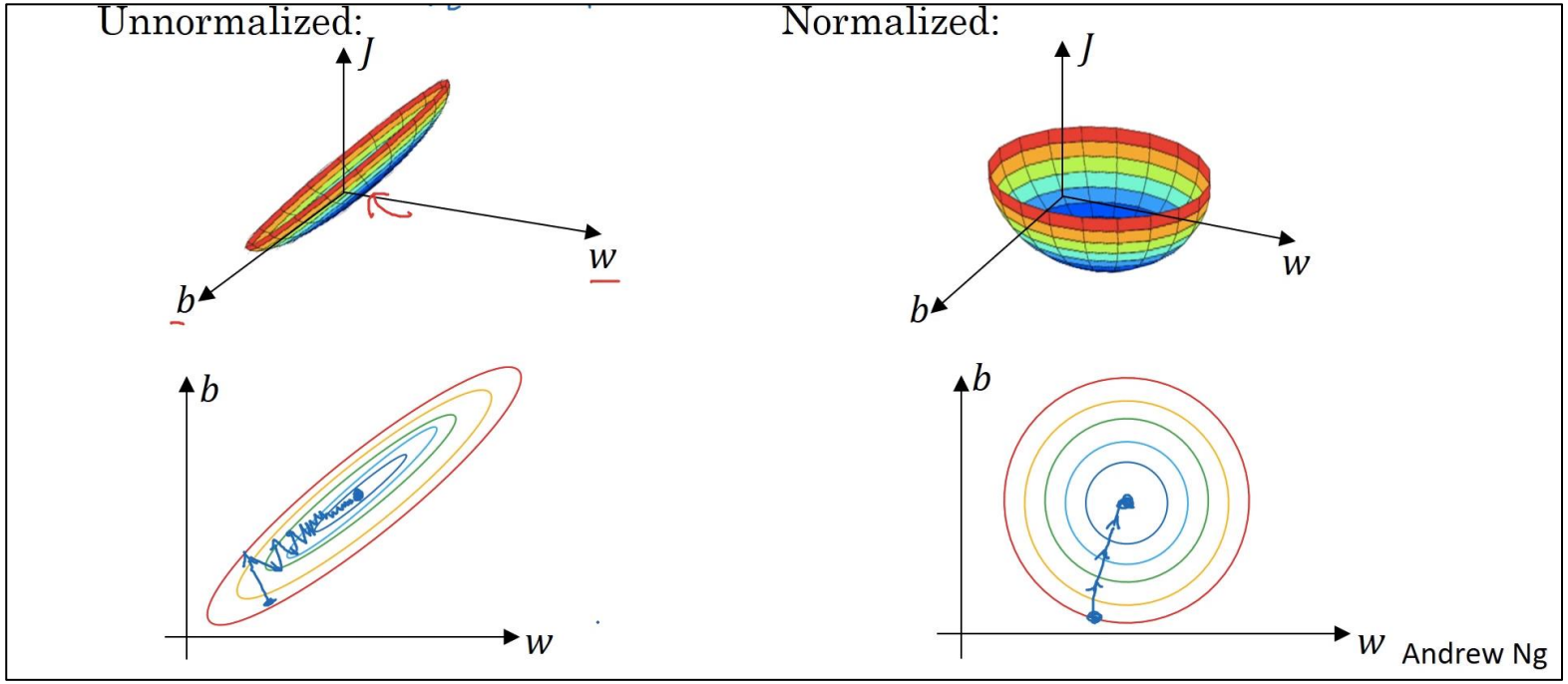
범위

- 초기화 (W1L09~14)

- 최적화 (W2L01~09)

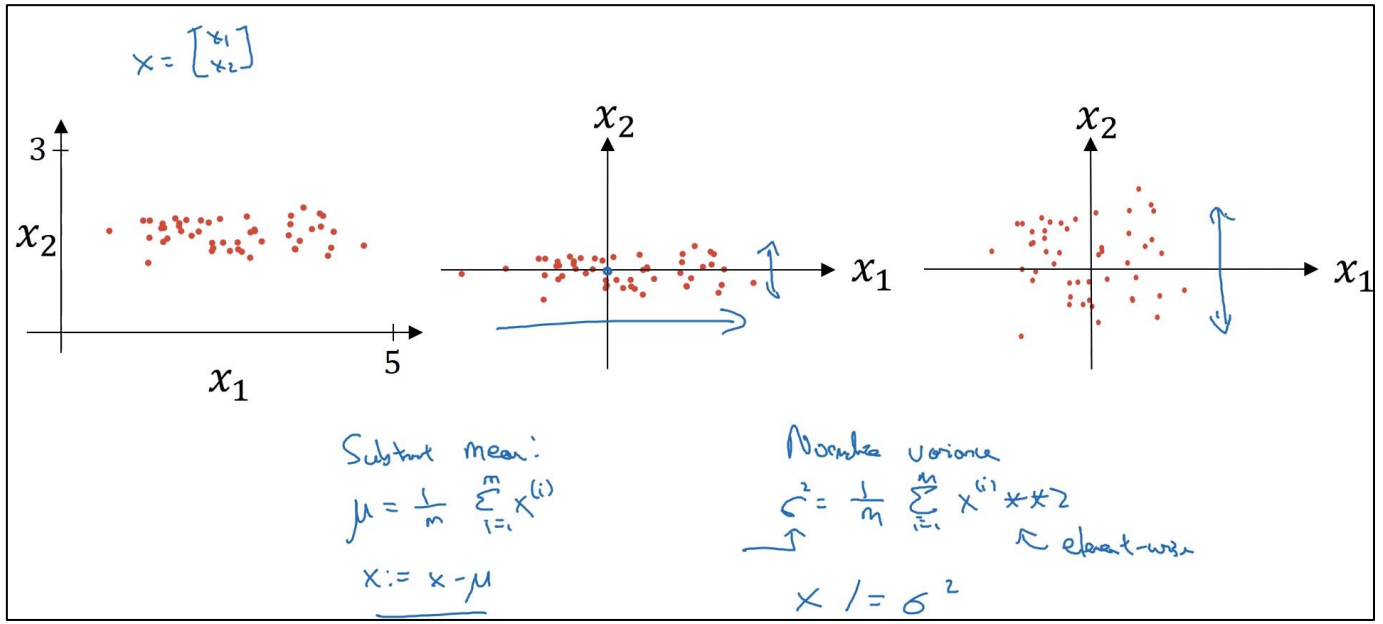
입력을 정규화하면 학습 속도가 빨라진다

- 입력을 정규화하면 weight가 어느 지점에서 시작되더라도 바로 최소점으로 향하게 된다.



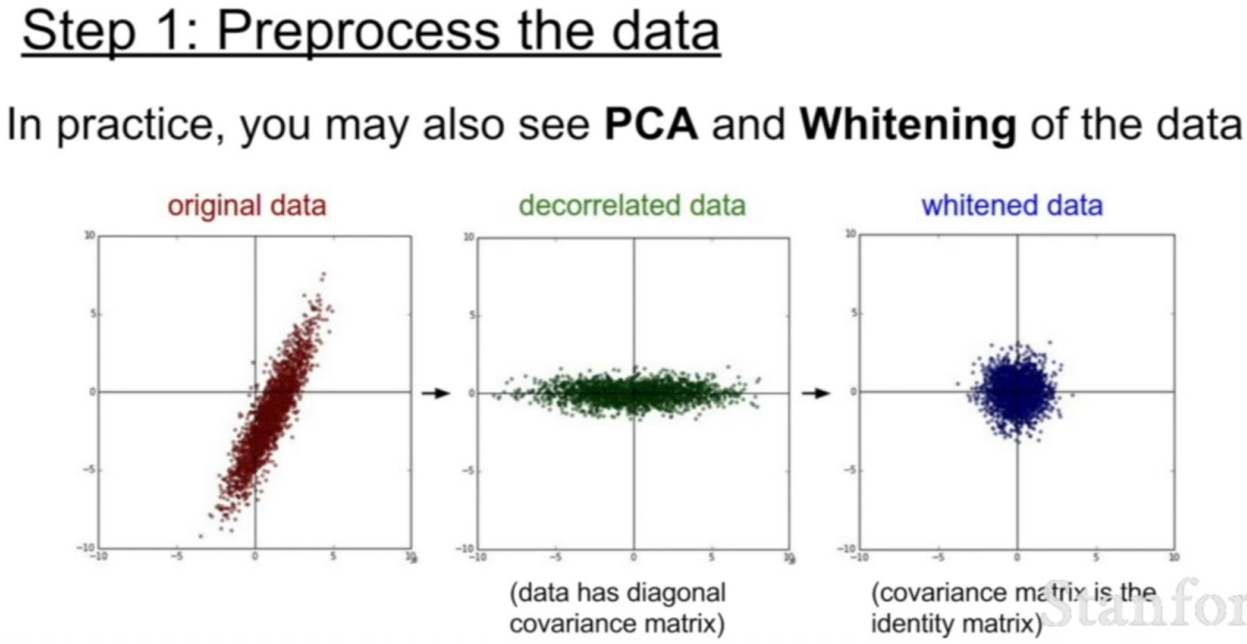
입력 정규화 방법

- 입력을 각각 ~N(0, 1)이 되도록.
- 이때 학습데이터에 적용한 평균과 분산 값을 테스트 데이터에도 똑같이 적용해야 함.



※ Preprocessing Data

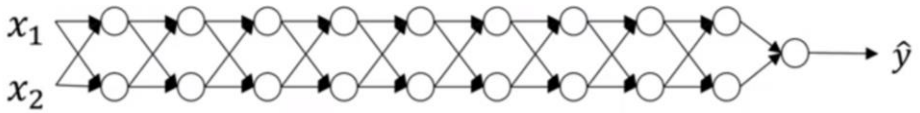
- PCA, Whitening 같은 고급 방법도 있음
  - CS231n
  - 문제에 따라 다른 정규화 방식을 사용하기도 한다.

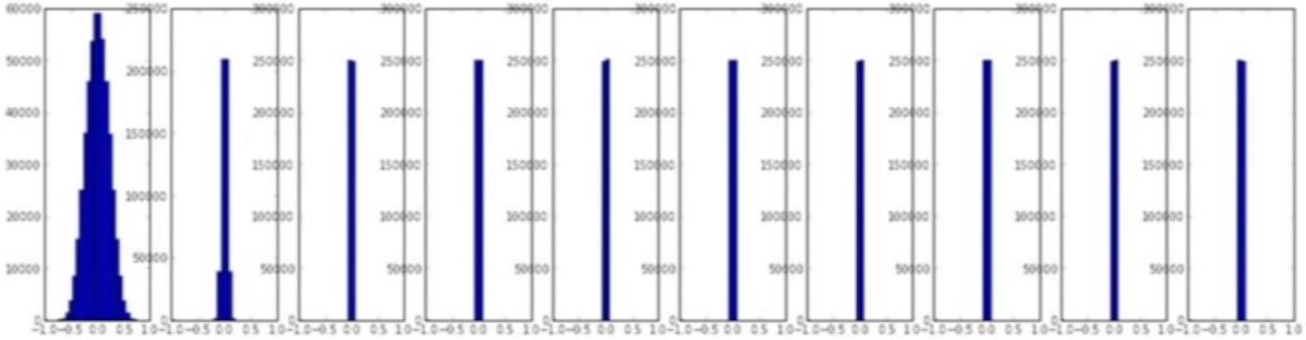


Gradient의 소실/발산

- 아주 깊은 네트워크에서는 출력값이나 gradient 값이 거의 사라지거나 발산해버리는 경우가 있다.
- 이는 학습을 느리게 만든다.

  -> weight의 초기값을 잘 설정해야 한다!





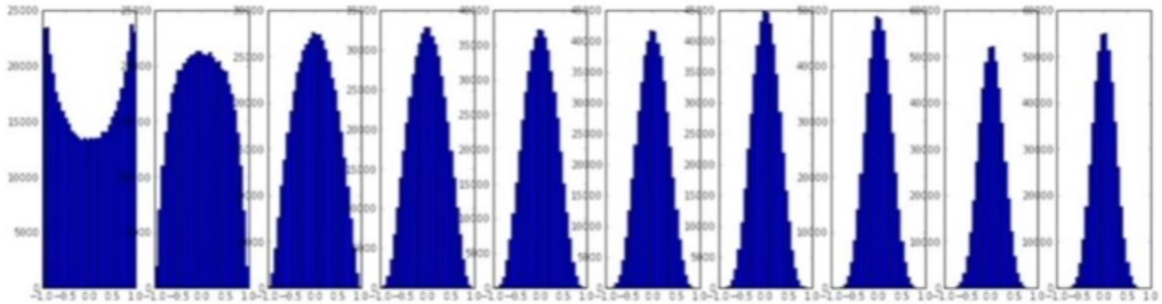
Gradient의 소실/발산 -> 가중치 초기화

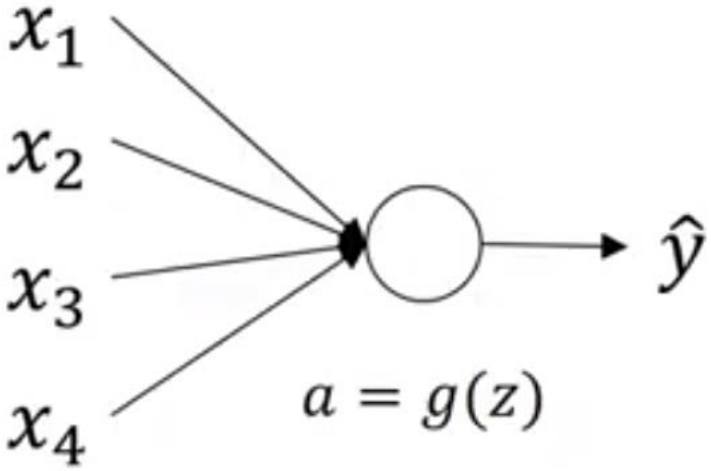
- Input x_i ~ 1
- Output = $\vec{w} \cdot \vec{x} = \sum_{i=1}^{n} w_i x_i$ ~ 1

  -> w_i ~ 1/n

  -> $w_i \sim \mathrm{N}\left(0, \frac{1}{n}\right)$: Xavier Initialization

  For ReLU, $w_i \sim \mathrm{N}\left(0, \frac{2}{n}\right)$ is preferred.





Optimizer 1: Mini Batch Gradient Descent

- Batch 경사하강법의 문제점: 한 번 업데이트하는데 모든 데이터를 다 봐야 해서 느리다.

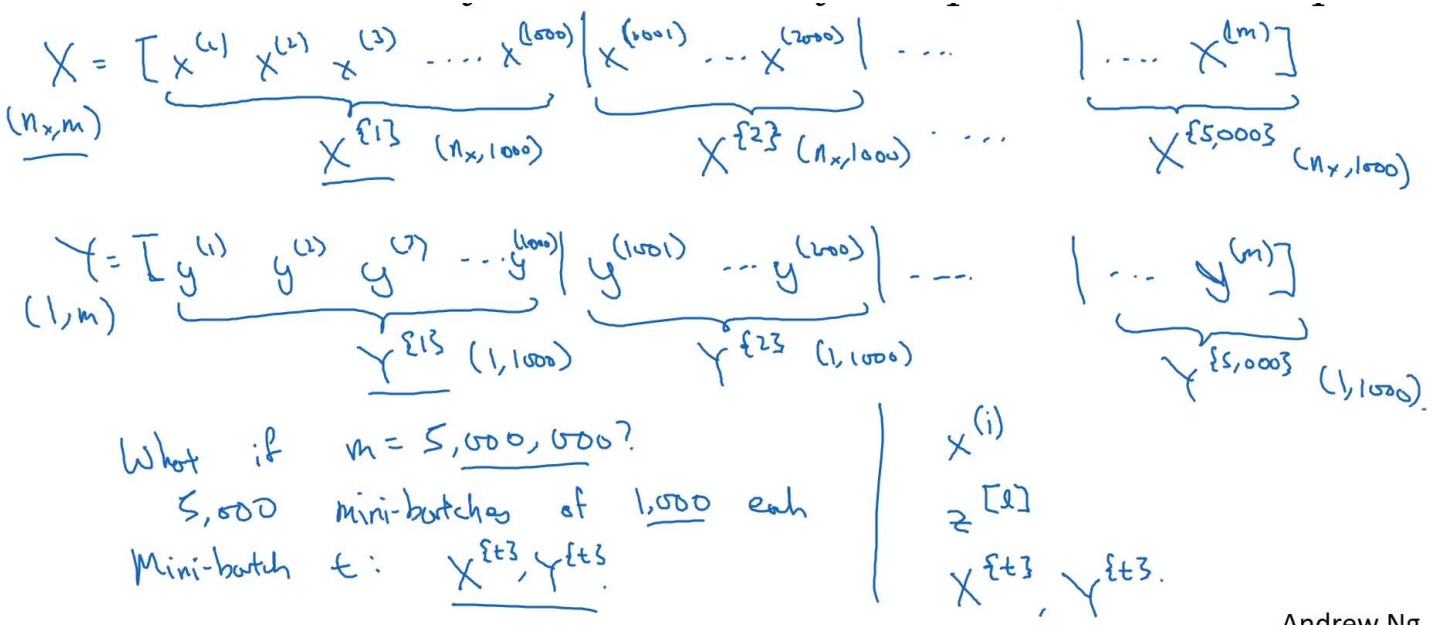
$X = [x^{(1)}\ x^{(2)}\ x^{(3)} \ldots \ldots\ x^{(m)}]$ (n_x, m)

$Y = [y^{(1)}\ y^{(2)}\ y^{(3)} \ldots \ldots\ y^{(m)}]$ $(1, m)$

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

Optimizer 1: Mini Batch Gradient Descent

- t번째 미니배치 $X^{\{t\}}, Y^{\{t\}}$ 에 대해 학습하고 업데이트를 진행
  -> 훨씬 빠르다!



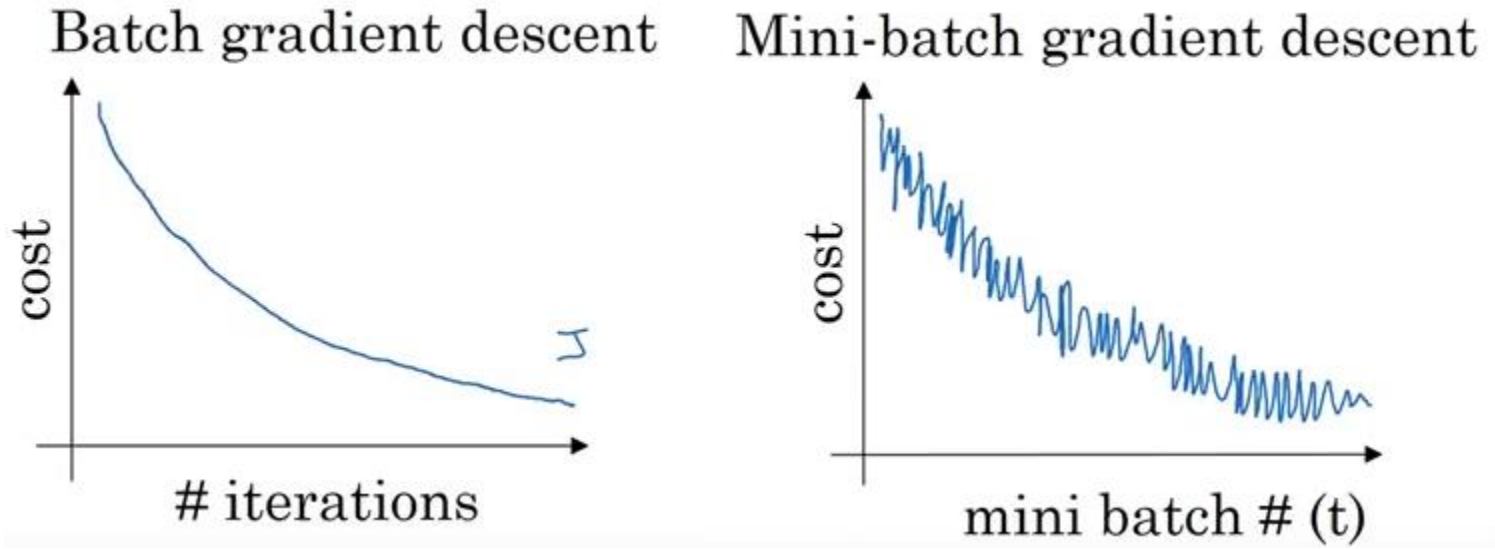
Andrew Ng

Optimizer 1: Mini Batch Gradient Descent

- t번째 미니배치 $X^{\{t\}}, Y^{\{t\}}$ 에 대해 학습하고 업데이트를 진행

  -> 훨씬 빠르다!

  But 노이즈 발생. 미니배치는 전체 배치에 대한 일종의 근사

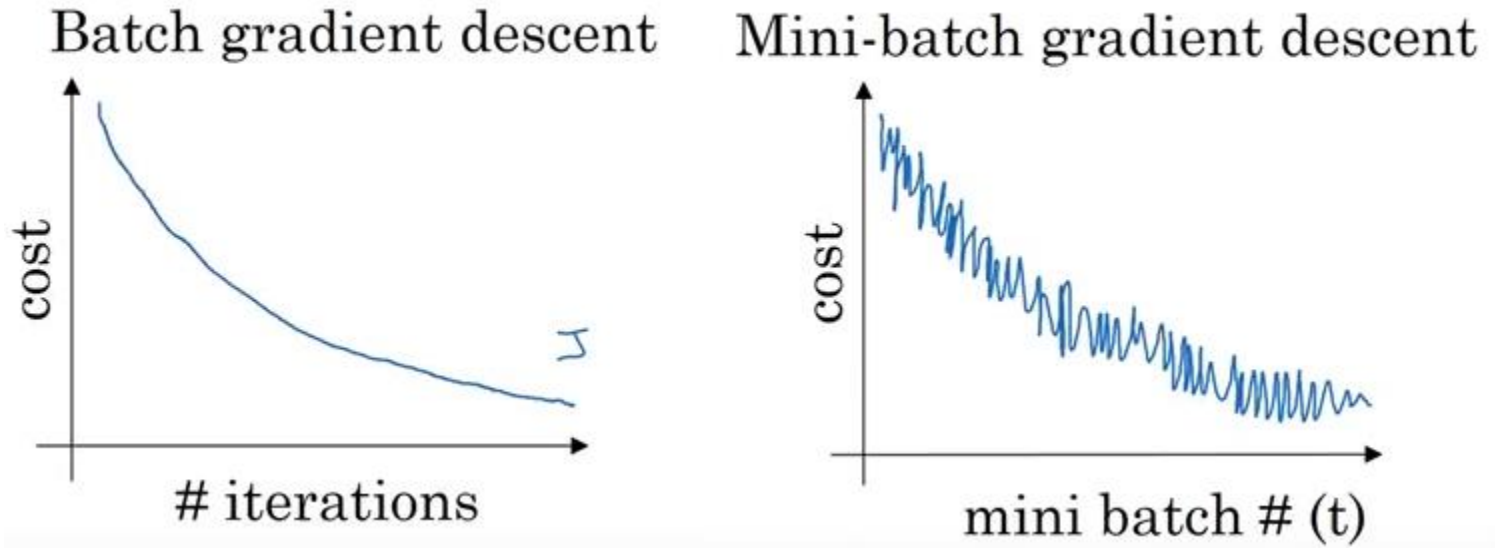


Optimizer 1: Mini Batch Gradient Descent

- t번째 미니배치 $X^{\{t\}}, Y^{\{t\}}$ 에 대해 학습하고 업데이트를 진행

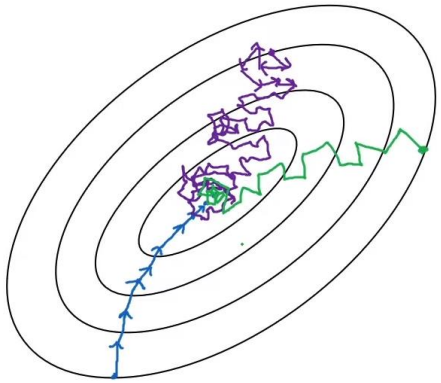
  -> 훨씬 빠르다!

  But 노이즈 발생. 미니배치는 전체 배치에 대한 일종의 근사

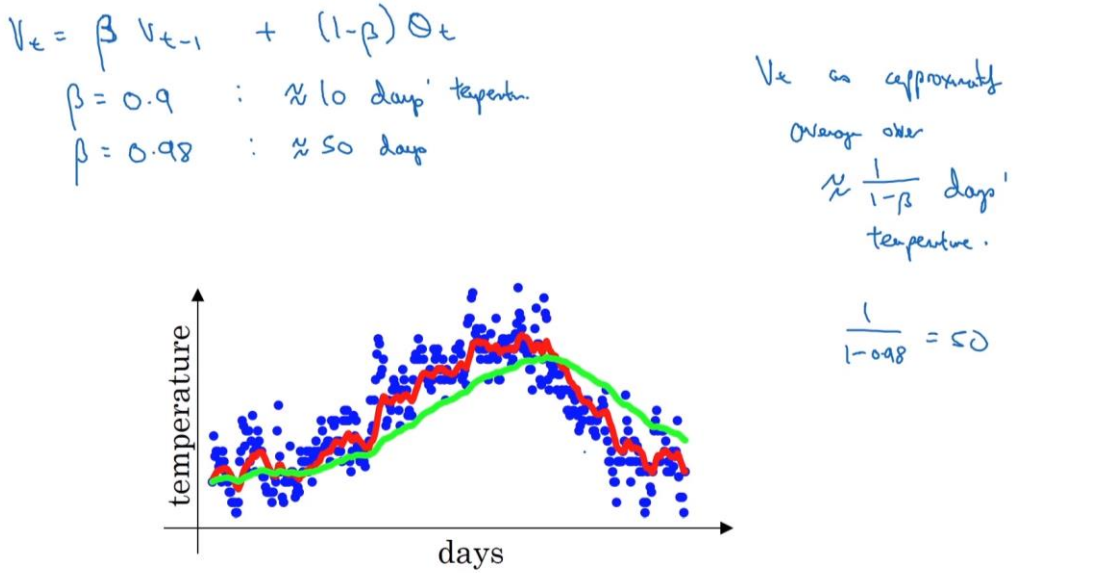


Optimizer 1: Mini Batch Gradient Descent

- 미니배치 사이즈 = m Gradient Descent
  - 한 번 업데이트하는데 너무 오래 걸림
  - $m \leq 2000$일 때 추천
- 미니배치 사이즈 = 1 Stochastic Gradient Descent
  - 벡터화의 이점을 얻을 수 없음
- 미니배치 사이즈 1~m Mini Batch Gradient Descent
  - 벡터화도 되면서 업데이트도 자주 됨
  - 보통 64, 128, 256, 512. 드물게 1024
  - CPU, GPU 메모리 크기에 맞는 값을 고른다.



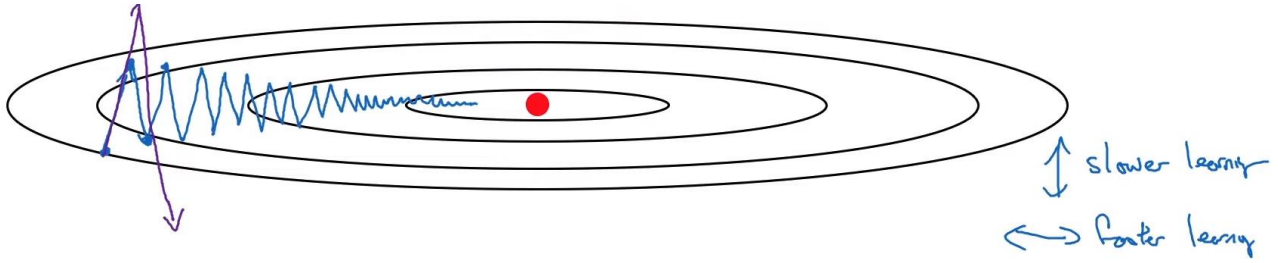
지수가중이동평균



- $v_0 = 0$
- $v_t = \beta v_{t-1} + (1-\beta)\theta_t$
- 메모리 소모가 적다!
  - v_t만 저장해두면 오랜 기간 동안의 평균을 근사할 수 있다.
- $\frac{v_t}{1-\beta^t}$을 사용하면 초반에 bias를 잡을 수 있다.
  - 어차피 수십~수천 번 학습할 거면 신경 안 써도 되긴 한다.

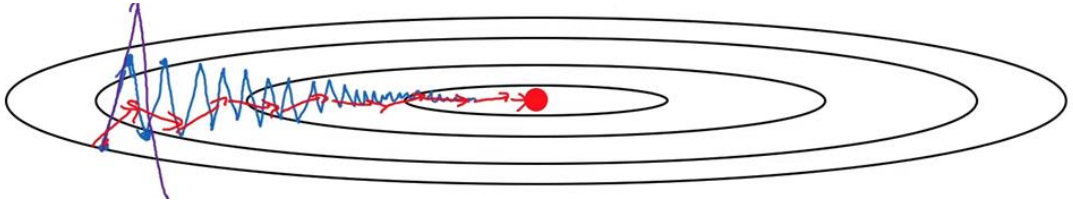
Optimizer 2: Momentum

- Gradient Descent의 문제점
  - 진동한다.
  - 정작 빨리 학습되어야 할 방향으로는 잘 나아가지 못한다.
  - Learning rate가 크면 발산한다.



Optimizer 2: Momentum

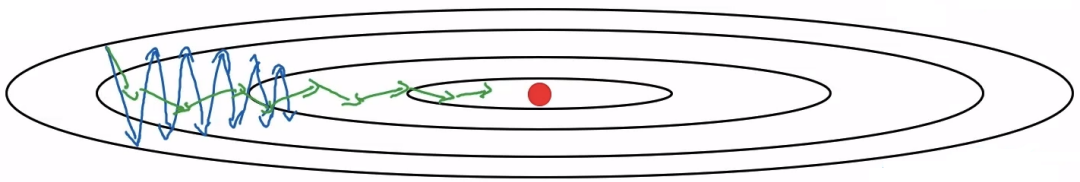
- Momentum 알고리즘
  - 평소처럼 미니배치마다 dw, db를 계산한다.
  - $V_{dw} = \beta V_{dw} + (1-\beta)dw$, $V_{db} = \beta V_{db} + (1-\beta)db$
  - $w -= \alpha V_{dw}$, $b -= \alpha V_{db}$



- V_{dw}: 속도, dw: 가속도 , β: 마찰
- 초매개변수: 학습률 $\alpha \sim 1e-3$, $\beta \sim 0.9$

Optimizer 3: RMS Prop

- Root mean square propagation 알고리즘
  - 평소처럼 미니배치마다 dw, db를 계산한다.
  - $S_{dw} = \beta_2 S_{dw} + (1-\beta_2)dw^2$, $S_{db} = \beta_2 S_{db} + (1-\beta_2)db^2$
  - $w -= \alpha \frac{dw}{\sqrt{S_{dw}+\epsilon}}$, $b -= \alpha \frac{db}{\sqrt{S_{db}+\epsilon}}$



- 많이 진동하는 쪽을 억제해서 학습 가속
- $\epsilon \sim 1e-8$: 0으로 나눠지지 않게 만들기 위한 안전장치.

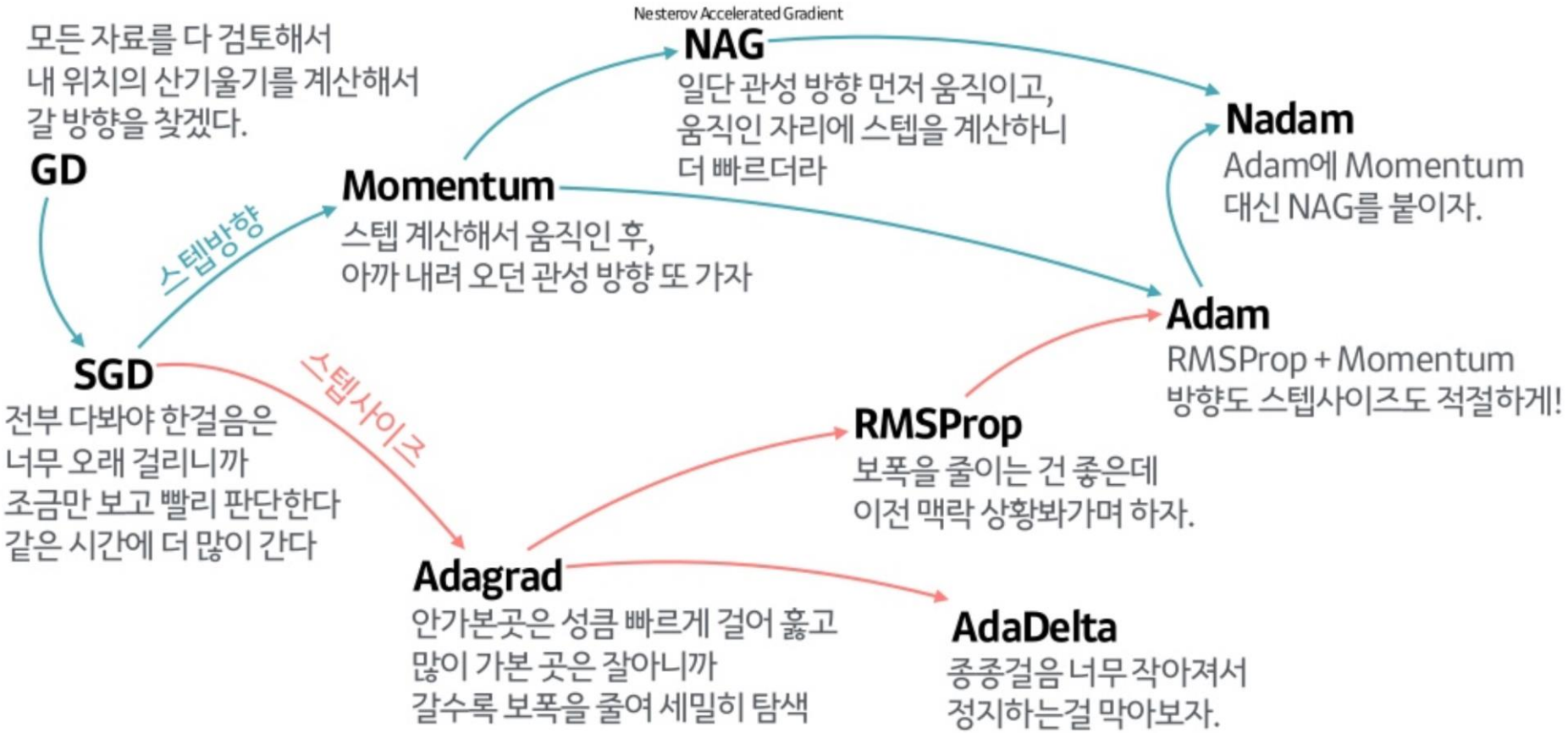
Optimizer 4: Adam

- Adaptive moment estimate 알고리즘
- Momentum + RMS Prop
  - 평소처럼 미니배치마다 dw, db를 계산한다.
  - $V_{dw} = \beta_1 V_{dw} + (1-\beta_1)dw$, $V_{db} = \beta_1 V_{db} + (1-\beta_1)db$
  - $S_{dw} = \beta_2 S_{dw} + (1-\beta_2)dw^2$, $S_{db} = \beta_2 S_{db} + (1-\beta_2)db^2$
  - $V_{dw}^{corrected} = \frac{V_{dw}}{1-\beta_1^t}$, $V_{db}^{corrected} = \frac{V_{db}}{1-\beta_1^t}$
  - $S_{dw}^{corrected} = \frac{S_{dw}}{1-\beta_2^t}$, $S_{db}^{corrected} = \frac{S_{db}}{1-\beta_2^t}$
  - $w -= \alpha \frac{V_{dw}^{corrected}}{\sqrt{S_{dw}^{corrected}+\epsilon}}$, $b -= \alpha \frac{V_{db}^{corrected}}{\sqrt{S_{db}^{corrected}+\epsilon}}$

Optimizer 4: Adam

- 초매개변수
  - 학습률 α : 튜닝 필요
  - $\beta_1 \sim 0.9$: 가끔 튜닝
  - $\beta_2 \sim 0.999$: 가끔 튜닝
  - $\epsilon \sim 10^{-8}$: 튜닝하지 않음

다양한 옵티마이저


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

학습률 감쇠

- 학습률을 점점 작게 만들자.
- 방법 1: $\alpha = \begin{cases} 0.001, & (epoch < 10) \\ 0.0001, & (epoch \geq 10) \end{cases}$
- 방법 2: $\alpha = \frac{k}{\sqrt{epoch}} \alpha_0$
- 방법 3: $\alpha = 0.95^{epoch} \alpha_0$