# 4주차

**Optimization Algorithms**

모델을 빠르게 훈련시키는 것이 매우 중요함

→ 좋은 최적화 알고리즘 찾기 : 효율성 향상

## ♥ Mini-batch gradient descent

- 벡터화

  m개의 샘플에 대한 계산을 효율적으로 만들어준다.

  명시적인 반복문 없이도 훈련 세트를 진행할 수 있도록 한다.

- m이 매우 크다면?

  ⇒ 훈련 세트를 더 작은 훈련 세트들(미니 배치)로 나누기

- **배치 경사 하강법**

  일반적인 경사 하강법

  모든 훈련 세트를 동시에 진행시키는 방법

- **미니 배치 경사 하강법**

  전체 훈련 세트를 한 번에 진행시키지 않고 하나의 미니배치 $X^t, Y^t$ 를 동시에 진행시키는 알고리즘

  반복문 돌리기(한 단계의 경사 하강법 구현)

⇒ 원하는 만큼 계속 거의 수렴할 때까지 훈련 세트 반복시키기

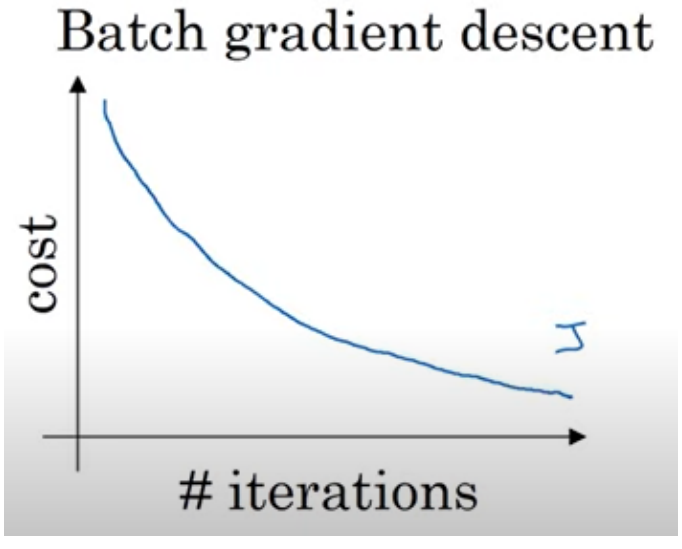
⇒ 훈련 세트가 많다면 미니배치 경사 하강법이 훨씬 빠름

## ♥ Understanding mini-batch gradient descent

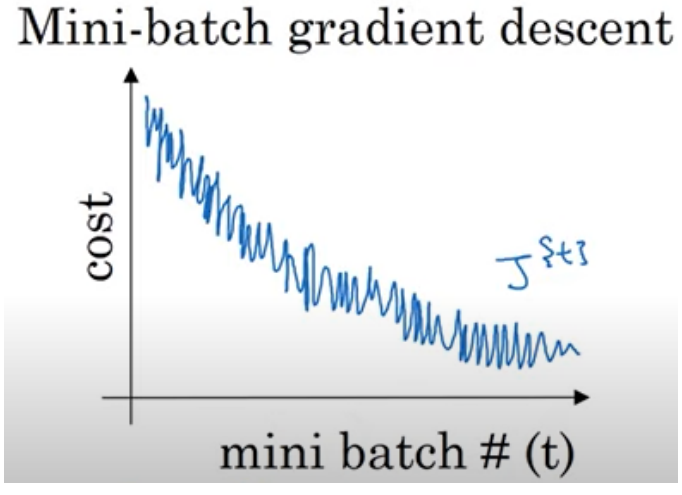
미니 배치 하강법의 구현

알고리즘의 수행 속도 높이기

- 배치 경사 하강법에서는 모든 반복마다 비용이 감소해야한다.




- 미니 배치 경사 하강법에서는 비용의 전체적인 흐름은 감소하나 약간의 노이즈가 발생될 수 있다.




    - 잘못 표시된 샘플이 있다든지의 이유로 진동이 일어날 수 있다.

- 미니배치의 크기
    - 훈련 세트의 크기: m
        - **미니배치 크기 = m**

          ⇒ 배치 경사 하강법, 하나의 미니 배치( $X^1, Y^1$ )만을 갖게 됨

          ⇒ 상대적으로 노이즈가 적고, 상대적으로 큰 단계

          ⇒ 한 반복에서 너무 오랜 시간이 걸린다. (매우 큰 훈련 세트)
        - **미니배치 크기 = 1**

          ⇒ 확률적 경사 하강법, 각각의 샘플이 하나의 미니배치

          ⇒ 모든 반복에서 하나의 훈련 샘플로 경사 하강법 실행

          ⇒ 극단적으로 노이즈가 많을 수 있지만, 평균적으로는 좋은 방향

          잘못된 방향일수도 있다.(절대 수렴 X)

          ⇒ 하나의 샘플만 처리한 뒤 계속 진행할 수 있어 간단하다.

          ⇒ 노이즈는 작은 학습률을 사용해 줄일 수 있다.

          ⇒ 벡터화에서 얻을 수 있는 속도 향상을 잃게 된다.  - 비효율적
        - 따라서 **1 < 미니배치 크기 < m** 선택

          ⇒ 너무 크거나 작지 않을 때 가장 빠른 학습을 제공

          ⇒ 많은 벡터화를 얻는다.

          ⇒ 전체 훈련 세트가 진행되기를 기다리지 않고 진행할 수 있다.

          (더 일관되게 전역의 최솟값으로 향하는 경향)

          ⇒ 매우 작은 영역에서 항상 정확하게 수렴하거나 진동하게 된다.

학습률을 낮추는 방법으로 해결 가능

- 미니 배치의 크기 가이드라인
  1. 작은 훈련 세트(m ≤ 2000)

     배치 경사 하강법 사용 ⇒ m

     ⇒ 전체 훈련 세트를 빠르게 진행할 수 있다.
  2. 큰 훈련 세트

     전형적인 미니 배치 크기 ⇒ 64 ~ 512

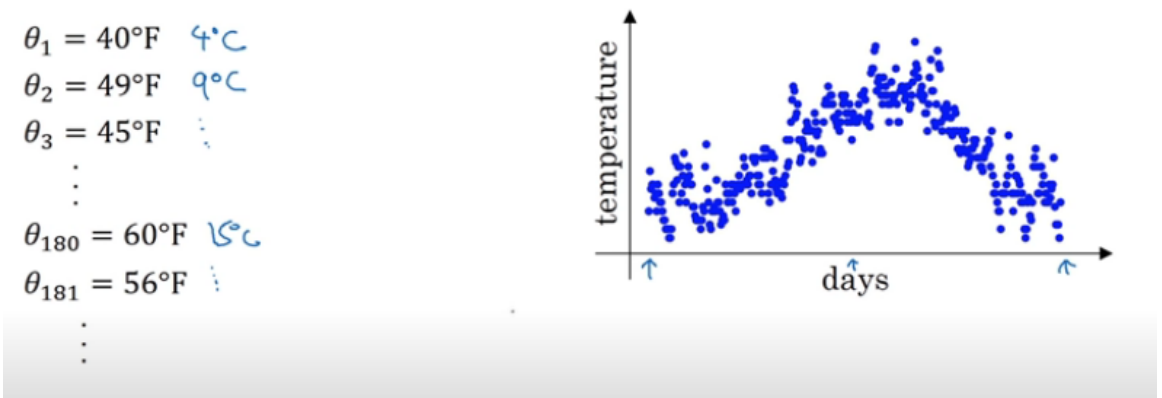
     ⇒ 2의 제곱인 것이 코드를 빠르게 실행시켜 준다.
  3. 모든 X, Y가 CPU와 GPU 메모리에 맞는지 확인하기

     ⇒ 맞지 않으면 성능이 갑자기 떨어지고 훨씬 나빠짐

## ♥ Exponentially Weighted Averages (지수 가중 평균)

== **Exponentially weighted moving average**

경사 하강법보다 빠른 최적화 알고리즘을 알기 위해 필요한 개념(주요 구성요소)




⇒ 노이즈 존재

Local average or Moving average 계산?

1. $V_0 = 0$ 초기화
2. $V_1 = 0.9\ V_0 + 0.1\ \theta_1$
3. …
4. $V_t = 0.9\ V_{t-1} + 0.1\ \theta_t$

⇒ 일별 기온의 exponentially weighted averages

- Exponentially weighted averages

  $V_t = \beta\ V_{t-1} + (1 - \beta)\ \theta_t$

  $V_t$ : $\frac{1}{1-\beta}$ days의 평균 기온

  ⇒ $\beta = 0.9$ (red) : 10 days average temperature

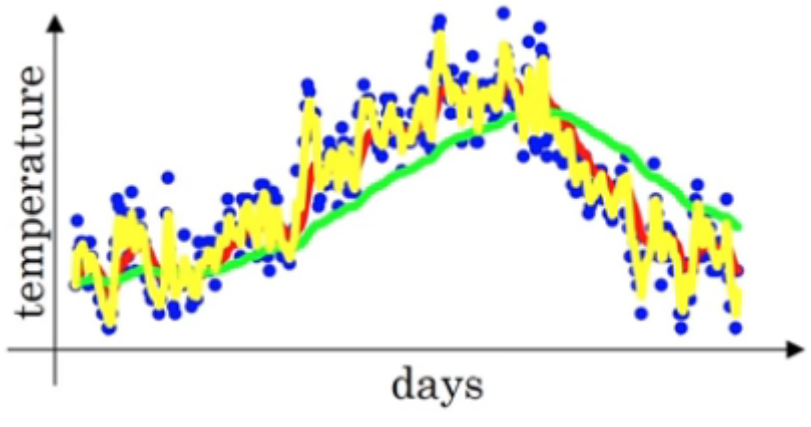
  ⇒ $\beta = 0.98$ (green) : 50 days

  : 기온이 올라가거나 내려갈 때 더 느리게 적응

⇒ $\beta = 0.5$ (yellow) : 2 days

: 노이즈가 많고 이상치에 더 민감

: 기온 변화에 빠르게 적응




$\beta$ 값이 클수록 선이 더 부드러워 진다. (더 많은 날짜의 기온의 평균을 이용하기 때문)

⇒ 클수록 이전 값에 많은 가중치, 현재의 기온에는 작은 가중치를 준다.

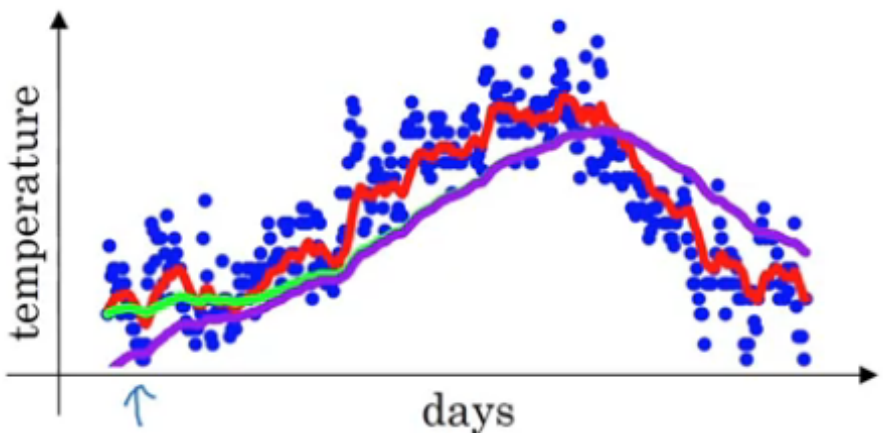
## ♥ Understanding Exponentially Weighted Averages

## ♥ Bias Correction of Exponentially Weighted Averages

편향 보정→ 평균을 더 정확하게 계산하기

$\frac{V_t}{1-\beta^t} = \beta\, V_{t-1} + (1-\beta)\, \theta_t$

⇒ 초반부 지수 가중 평균 값 보정,,




보라색 그래프 → 초록색 그래프

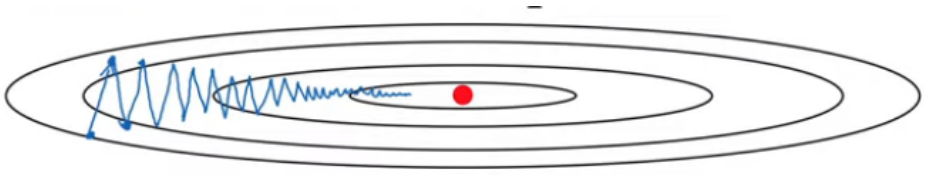
## ♥ Gradient Descent With Momentum

**Algorithm called momentum**

- 기본 아이디어 : 경사에 대한 지수 가중 평균 계산 → 가중치 업데이트

- Momentum ?

Gradient Descent의 이동 방향과 속도가 유지될 수 있도록 경향성 추가



⇒ 랜덤한 진동

- 최종 수렴까지 더 오래 걸림,
- learning rate를 함부로 키울 수 없는 문제 (오버슈팅으로 인해 발산할 수 있기 때문)

- **Gradient Descent with Momentum**

$v_{dW} = \beta\, v_{dW} + (1-\beta)dW$
$v_{db} = \beta\, v_{db} + (1-\beta)db$
$W = W - \alpha\, v_{dW},\ b = b - \alpha\, v_{db}$

⇒ 경사 하강법의 단계를 부드럽게 만들어 줌

⇒ 수직 방향에서는 훨씬 더 작은 진동, 수평 방향에서는 더 빠른 움직임

⇒ 직선의 길을 가거나 진동을 줄일 수 있게 한다.

## ♥ RMSProp

**root mean square prop**

⇒ 수직 방향에서의 업데이트는 감소, 수평 방향은 계속 나아가게 한다.

⇒ 큰 학습률을 사용해 빠르게 학습(속도 증가), 수직 방향으로 발산하지 않는다.

⇒ (진동을 줄이는 효과가 있다)

## ♥ Adam Optimization Algorithm

**RMSProp + Momentum**

⇒ 두 가지를 동시에 사용해 진동을 방지, minimum으로 빠르게 수렴할 수 있게끔 한다.

## ♥ Learning Rate Decay

**학습률 감쇠시키기**

⇒ 훈련 속도를 빠르게

1. Epoch 마다 감소

$\alpha = \frac{1}{(1+decayrate * epochnum)} * \alpha_0$

2. 지수적 감쇠

$\alpha = 0.95^{epochnum} * \alpha_0$

3. 제곱근 사용

$$\alpha = \frac{k}{\sqrt{epochnum}} * \alpha_0$$

4. 이산 계단