WEEK 2. Optimization Algorithms

• Mini-batch gradient descent

- Deep learning은 highly empirical process, 빅데이터에서 잘 작동
  ⇒ 좋은 최적화 알고리즘으로 speed-up 이 필요

- Batch vs mini-batch gradient descent
  ⇒ Vectorization은 m example에 대한 효율적인 계산을 가능하게 해주지만,
  매우 큰 m에 대해서는 느려질 수 있음

Batch gradient descent	Mini-batch gradient descent
훈련샘플의 모든 batch로 훈련	전체 훈련세트 (X, Y)를 한번에 진행하지 않고 하나의 mini-batch $(X^{\{t\}}, Y^{\{t\}})$를 동시에 진행
모든 훈련 샘플을 gradient descent 전에 processing 해야 함	모든 training sample을 processing 하기 전에 gradient descent를 진행시킬 수 있음
1 epoch ⇒ one gradient descent step	1 epoch → 5,000 gradient descent (반복적인 학습)

- What if $m = 5{,}000{,}000$?

$X = [\underbrace{X^{(1)} X^{(2)} \cdots X^{(1000)}}_{X^{\{1\}}} \underbrace{X^{(1001)} \cdots X^{(2000)}}_{X^{\{2\}}} \cdots \underbrace{\cdots X^{(m)}}_{X^{\{5{,}000\}}}]$
(n_x, m) $(n_x, 1000)$

$Y = [\underbrace{Y^{(1)} Y^{(2)} \cdots Y^{(1000)}}_{Y^{\{1\}}} \underbrace{Y^{(1001)} \cdots Y^{(2000)}}_{Y^{\{2\}}} \cdots \underbrace{\cdots Y^{(m)}}_{Y^{\{5{,}000\}}}]$
$(1, m)$ $(1, 1000)$

$Y^{\{1\}}$ (1,1000) $Y^{\{2\}}$ $Y^{\{5,000\}}$

1 mini batch — batch size

* Notation

$a^{[l]\{t\}(i)}$ ⇒ i^{th} samples activation at l^{th} layer of t^{th} mini batch

- How does it work?

⇒ 1 step of gradient descent using $X^{\{t\}}, Y^{\{t\}}$ (as if $m = 1,000$)

repeat { for $t = 1, \ldots, 5000$ {

Forward prop on $X^{\{t\}}$

$$Z^{[1]} = W^{[1]} X^{\{t\}} + b^{[1]}$$
$$A^{[1]} = g^{[1]}(Z^{[1]})$$
$$\vdots$$
$$A^{[l]} = g^{[l]}(Z^{[l]})$$

Vectorized implementation (1,000 examples)

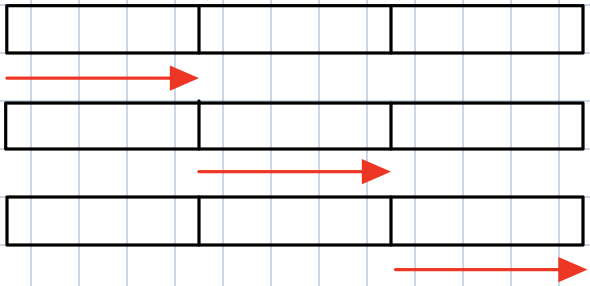
Compute cost $J^{\{t\}} = \frac{1}{1000} \sum_{i=1}^{1000} \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2 \cdot 1000} \sum_l \|W^{[l]}\|_F^2$

Backprop to compute gradients w.r.t $J^{\{t\}}$ (using $(X^{\{t\}}, y^{\{t\}})$)

$W^{[l]} := W^{[l]} - \alpha dW^{[l]}$

$b^{[l]} := b^{[l]} - \alpha db^{[l]}$

} → one-epoch : a single pass through training set (full batch / min-batch)



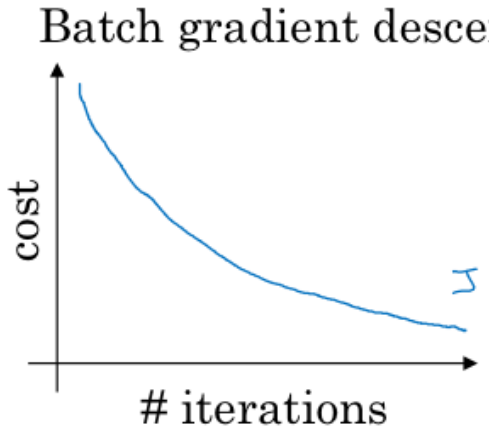
⇒ 1 epoch

⇒ 1 epoch

⇒ 1 epoch

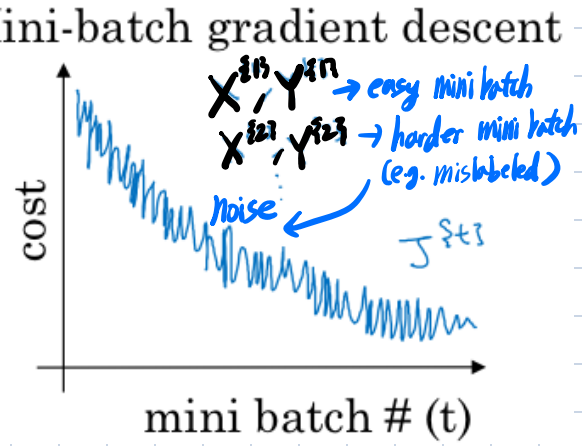
• Understanding mini-batch gradient descent

— Training with mini batch gradient descent



모든 반복에서 전체 샘플 사용 &
각각의 반복마다 비용 감소

⇒ 모든 반복마다 감소해야 함
한 번이라도 올라가면 learning rate가
너무 크지는 않은지 점검 필요



모든 반복에서 서로 다른 mini-batch
이용한 훈련

⇒ 모든 반복마다 반드시 J 감소X + noise
(but, 낮아지는 경향)

— Choosing your mini-batch size

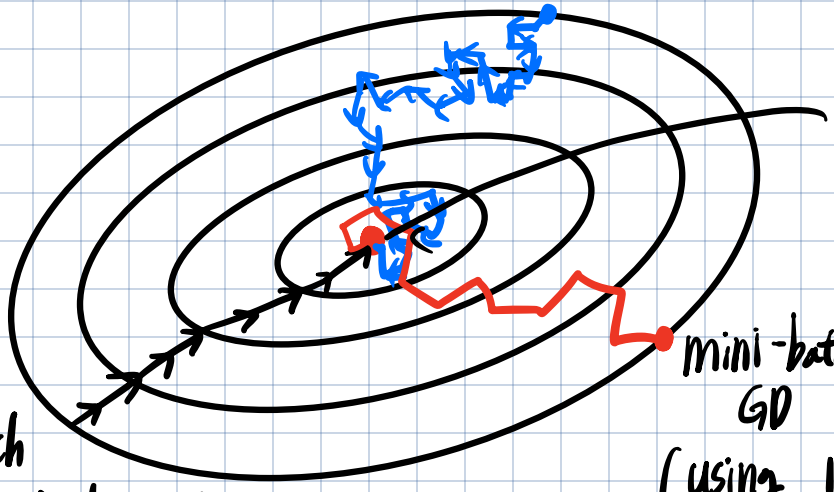
⇒ If mini-batch size is...

size = m	1< size <m	size = 1
Batch Gradient Descent		Stochastic Gradient descent
각 반복마다 huge training set을 process 해야 함 (X, Y)	한 번에 batch-size 만큼의 training sample로 진행	한 번에 하나의 training sample로 경사하강법
too long per iter	Fastest learning ① Vectorization (~1000) ② Make progress w.o. processing entire training set	① Lose benefits of Vectorization ② Will never converge ③ 방향은 맞지만 최솟값으로

③ always converge / oscillate within very small region

straight forward 하게 가지는 않는다.

Stochastic GD (using 1 training sample)



각은 지점에서 oscillate하는 것은 smaller learning rate로 noise를 줄여줄 수 있음.

mini-batch GD (using $1 < n < m$ training samples per iteration)

batch Gradient descent (using entire training samples per iteration)

- Exponentially weighted (moving) averages
  - Basic concept before learning optimization algorithms that are faster than gradient descent.
  - Temperature in London

$\theta_1 = 40°F$

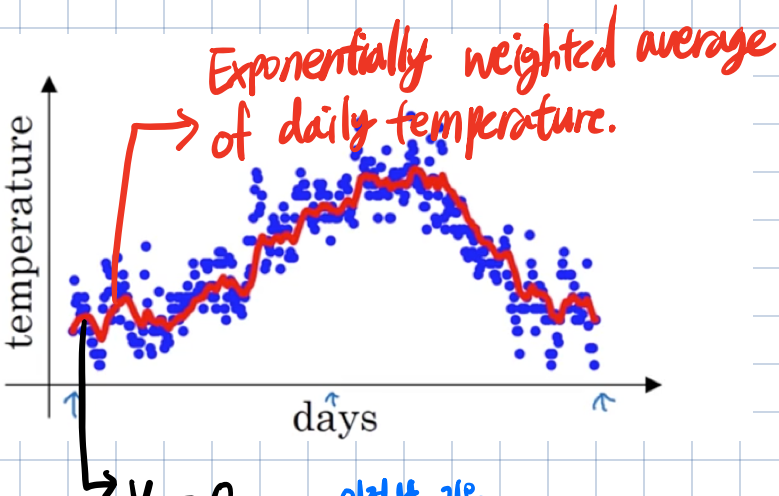
$\theta_2 = 49°F$

$\theta_3 = 45°F$

$\vdots$

$\theta_{180} = 60°F$

$\theta_{181} = 56°F$



$V_0 = 0$

$V_1 = 0.9V_0 + 0.1\theta_1$ (이전 값, 오늘 값)

$V_2 = 0.9V_1 + 0.1\theta_2$

$\vdots$

$V_t = 0.9V_{t-1} + 0.1\theta_t$

- Formula of exponentially weighted averages

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

↳ β를 조정하면 약간씩 다른 효과를 얻는다.

parameter

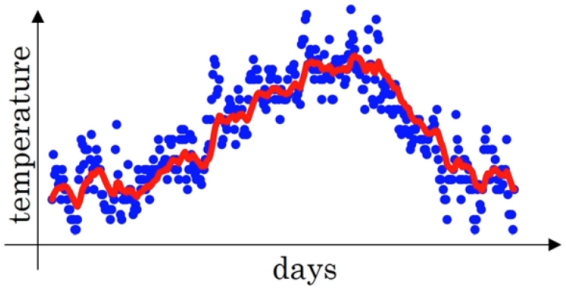
$\beta = 0.9$: $\approx$ 10 days temp

$\beta = 0.98$: $\approx$ 50 days temp ← V_t는 약 $\approx \frac{1}{1-\beta}$ days의 temp를 가지고 평균을 냄

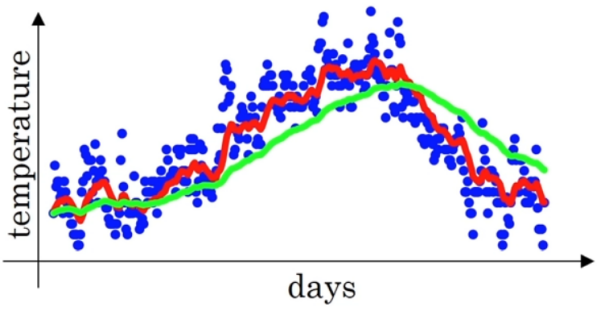
$\beta = 0.5$: $\approx$ 2 days temp

⇓

β가 1에 가까울수록 이전값에 가중치를 둔다. ∴ 현재에 변화가 생기면 trend 값 또 따라감

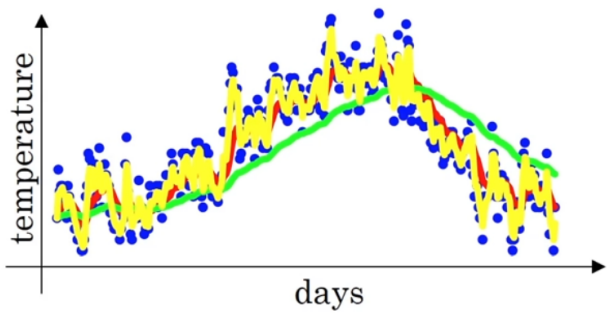


When $\beta = 0.9$ (10 days)



When $\beta = 0.98$ (50 days)

⇒ Smoother line but the curve is shifted to right (기온이 바뀌면 느리게 반응)



When $\beta = 0.5$ (2 days)

⇒ Much more noisy & susceptible to the outliers (fast adaptation)

- Understanding exponentially weighted averages

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

$\Rightarrow V_{100} = 0.9 V_{99} + 0.1 \theta_{100}$

$V_{99} = 0.9 V_{98} + 0.1\theta_{99}$

$V_{98} = 0.9 V_{99} + 0.1\theta_{98}$

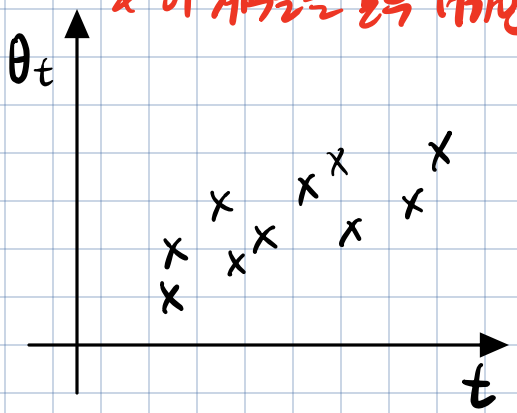
$\vdots$

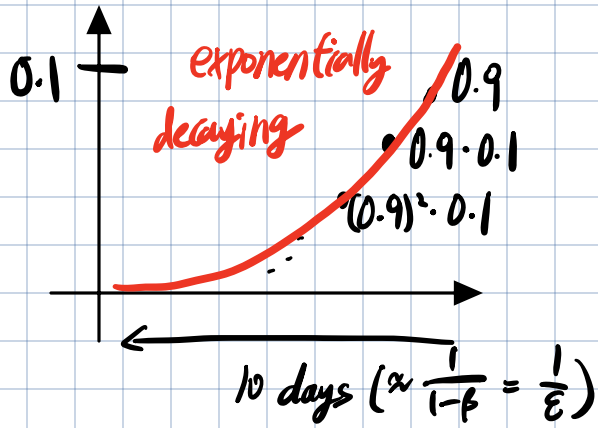
$V_{100} = 0.1\theta_{100} + 0.9(0.1\theta_{99} + 0.9 V_{98})$

$= 0.1\theta_{100} + 0.1\times0.9\,\theta_{99} + (0.9)^3 \cdot 0.1\theta_{97} + 0.1\cdot(0.9)^4\theta_{96}$

⇒ 일일 온도에 기하학적으로 감소하는 합을 곱한걸 모두 더하고 있다.

& 이 계수들을 모두 더하면 ≈ 1 (bias correction)





⇒ 10일 뒤에는 가중치가 현재의 가중치의 $\frac{1}{3}$로 줄어든다.

— Implementing exponentially weigted averages

$V_0 = 0$

$V_1 = \beta V_0 + (1-\beta)\theta_1$

$V_2 = \beta V_1 + (1-\beta)\theta_2$

$V_3 = \beta V_2 + (1-\beta)\theta_3$

$\vdots$

$V_\theta = 0$

Repeat {

Get next θ_t

$V_\theta := \beta V_\theta + (1-\beta)\theta_t$

(Overwrite the recent value)

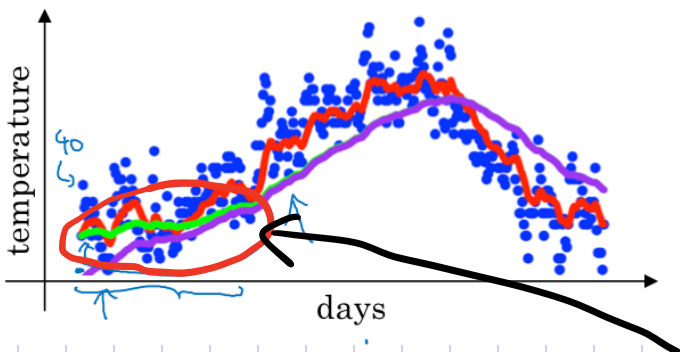
— Advantages of exponentially weighted average

① It takes very little memory & cheap computation

② One line code

• Bias Correction in exponentially weighted average

- Estimate of initial phase



green: theoritical
purple: actual implementation

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

$V_0 = 0 \rightarrow 0$

$V_1 = 0.98 V_0 + 0.02\theta_1$

$V_2 = 0.98 V_1 + 0.02\theta_2 < \theta_1$ or θ_2

⇒ 최기값이 0이기 때문에 처음에는 온도를 잘 추정하지 못하기 때문에 최대한 값을 보정해주어야 한다.

- Bias correction

$$\frac{V_t}{1-\beta^t} \Rightarrow \frac{V_2}{1-(0.98)^2} = \frac{0.0196\theta_1 + 0.02\theta_2}{0.0396}$$

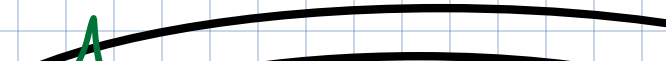
t가 커질수록 $1-\beta^t$는 0에 가까워지므로 시간이 지날수록 보정효과↓ (but, 시간이 어느정도 지나면 보정 없어도 추정을 잘한다.)

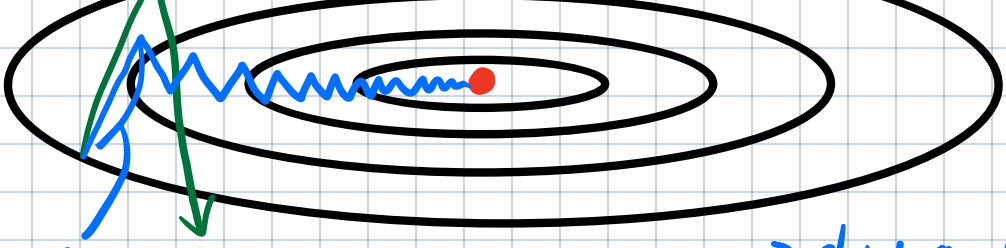
• Gradient descent with momentum

- min-batch를 이용하는 경우 training set의 subset만 보기 때문에 update direction에 variance & oscillation toward convergence

⇒ compute an exponentially weighted average of gradients for updating weights to smooth out direction.

- Gradient descent example





1 iter of mini batch of gradient descent

bigger learning rate makes 'overshooting'

→ slow gradient descent

⇒ larger learning rate is not allowed

⇒ We want vertically slow learning ↕
horizontally fast learning ↔

- Gradient descent with momentum

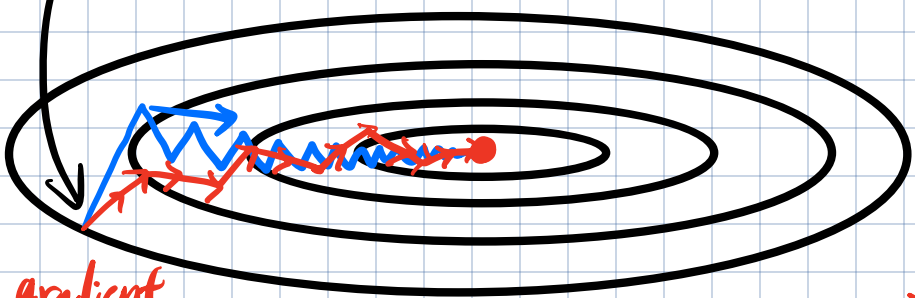
On iter t:

Compute dw, db on current mini-batch

$V_{dw} = \beta V_{dw} + (1-\beta)\, dw$

$V_{db} = \beta V_{db} + (1-\beta)\, db$

$w := w - \alpha V_{dw}$, $b := b - \alpha V_{db}$



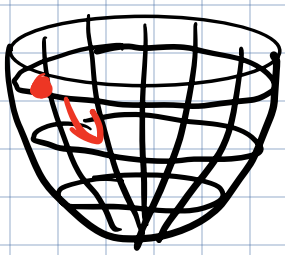
gradient descent with momentum

average out gradients & smooth out the steps of gradient descent

⇒ 수직방향의 진동이 0에 가깝고 수평방향으로는 큰 값으로 평균이 만들어짐

- Bowl-shaped function

⇒ bowl-shaped function을 최적화한다고 했을 때



$V_{dw} = \beta V_{dw} + (1-\beta)\, dw$

$V_{db} = \beta V_{db} + (1-\beta)\, db$

① friction ($\beta < 1$)
② Velocity
③ Acceleration
→ momentum

- Implementation details

$V_{dw} = 0,\ V_{db} = 0$ (same dimension with dW)

On iteration t:

Compute dW, db on the current mini-batch.

$V_{dW} = \beta v_{dw} + (1-\beta) dW$

$V_{db} = \beta v_{db} + (1-\beta) db$

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

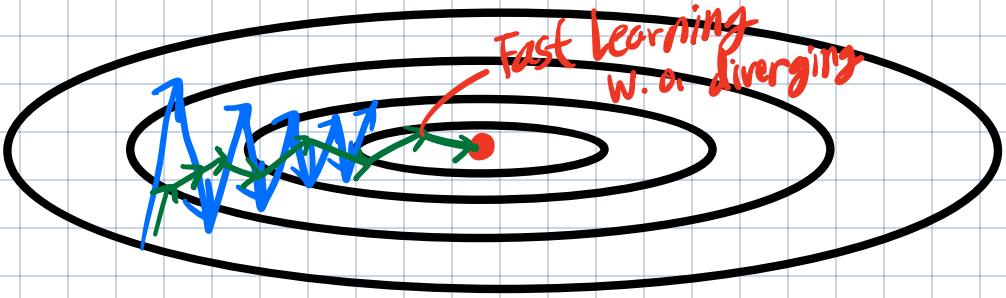
⇒ Hyperparameters : α, β ↑ 0.9 is common

⇒ 보통 10번 정도 후에 이동평균 계산이 충분히 진행되어 bias X

∴ $\frac{V_{dw}}{1-\beta^t}$ (bias correction)은 쓰지 않는다.

- RMSprop

- to speed up gradient descent



⇒ On iteration t:

Compute dW, db on current mini-batch

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

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

speed up ← $w := w - \alpha \frac{dw}{\sqrt{S_{dw}} + \varepsilon}$ $b := b - \alpha \frac{db}{\sqrt{S_{db}} + \varepsilon}$ ⇒ slow down

(horizontal) ↑ small (vertical) ↑ large

※ ε ⇒ make sure to not divide by 0, commonly 10^{-8}

- Advantages
  1. Similar to momentum (관성)
  2. Allow us to use larger learning rate ⇒ speed up gradient descent
  3. Work well in wide Neural Network.

- **Adam optimization algorithm (Adapter moment estimation)**
  - Wide NN에서는 optimization이 잘 작동하지 않을 수 있다.
  - Momentum + RMS prop

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

On iter t:

Compute dw, db using current mini-batch

momentum β_1 ⇒ $V_{dw} = \beta_1 V_{dw} + (1-\beta_1) dw,\ V_{db} = \beta_1 V_{db} + (1-\beta_1) db$

RMS prop β_2 ⇒ $S_{dw} = \beta_2 V_{dw} + (1-\beta_2) dw^2,\ V_{db} = \beta_2 V_{db} + (1-\beta_2) db^2$

bias correction

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

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

$$w := w - \alpha \frac{V_{dw}^{corrected}}{\sqrt{S_{dw}} + \varepsilon},\quad b := b - \alpha \frac{V_{db}^{corrected}}{\sqrt{S_{db}} + \varepsilon}$$

⇒ Commonly used for Neural Network

- Hyperparameter choice

- α : needs to be tuned
- β_1 : 0.9 $\longrightarrow$ (dw) $\Rightarrow$ momentum-like term (1st moment)
- β_2 : 0.999 $\longrightarrow$ (dw^2) (2nd moment)
- ε : 10^{-8}

- **Learning rate decay**
  - to slowly reduce learning rate
    - ⇒ If you're implementing mini-batch gradient descent with very small mini-batch (e.g. 64, 128...)

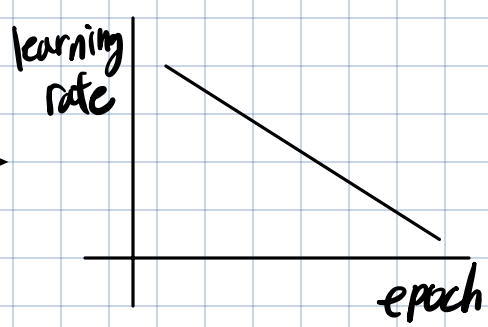
  - Decide learning rate
    - ⇒ reduce learning rate as epoch increases

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

need to be tuned

⇒ if $\alpha_0 = 0.2$, decay-rate = 1,

Epoch	α
1	1
2	0.67
3	0.5
⋮	⋮



- Other learning rate methods

① exponential decay

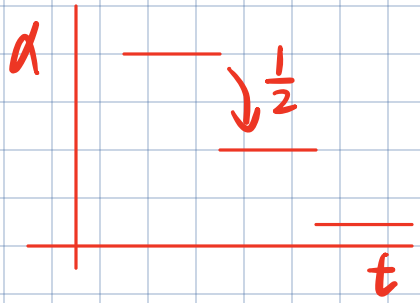
$$\alpha = 0.95^{\text{epoch-num}} \cdot \alpha_0$$

↳ this should be < 1

②

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

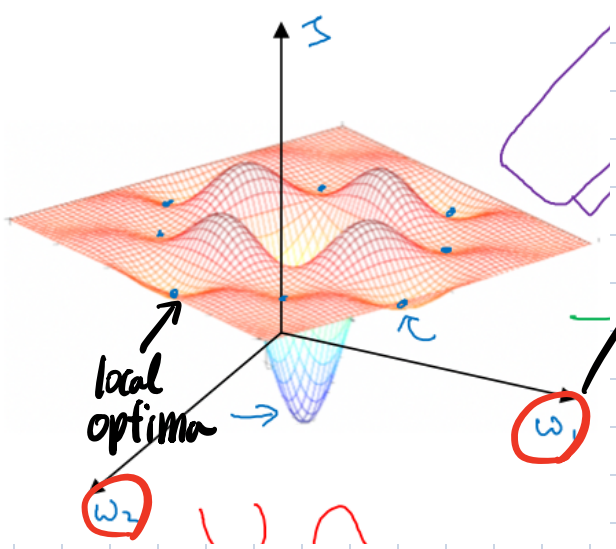
③ discrete stair case



④ manual decay (when training # is small)

- The problem of local optima

- Local optima in neural networks



실제로는 훨씬 고차원 (n 차원)

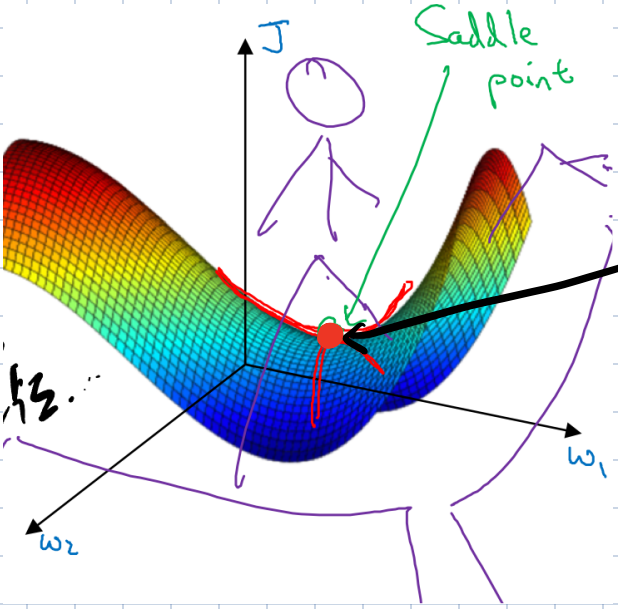
⇒ 모든 n개의 방향에서 볼록/오목 함수여야하기 때문에 가능성이 적다.

e.g. 20000 차원에서 local optima가 만들어지는 확률

$= 2^{-20000}$

- Saddle point

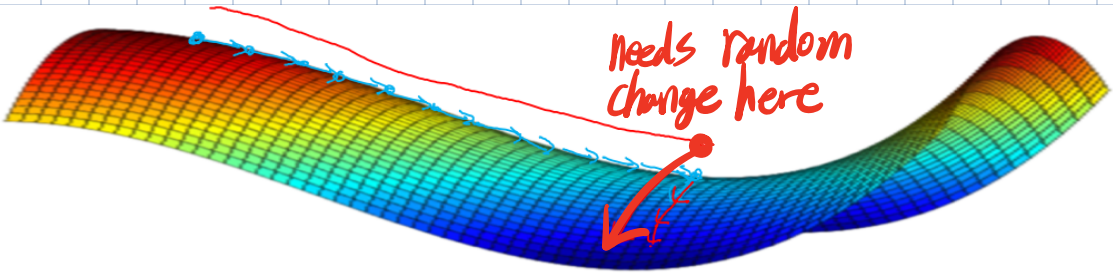
⇒ 실제로는 말 안장과 같은 모양이다.



0 gradient

⇒ 고차원에서는 local optima보다 saddle point가 되기 쉽다.

- Problem of plateaus: region where derivatives are close to zero for a long time



① Unlikely to get stuck in a bad local optima in high dimension

② Plateaus can make learning slow.

⇒ Momentum, RMS prop or Adam can speed up exiting plateaus