4. Optimization : most important step in training

↳ loss를 최소화하는 parameter를 수학적으로 탐색하는 과정

Introduction

1. Optimization in deep learning

↳ training 할때에서 optimization 역할 大

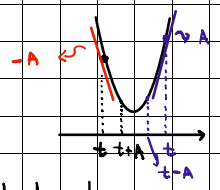
⇒ training error, object function을 감소시키는 문제

- most difficult optimization task in DL : training
  → important and expensive
- mainstream : stochastic gradient descent (→ first order method) and its variants
  + second-order methods, convex optimization

2. Derivatives and optimization order

- derivatives
  ① first derivatives (= gradient) ⇒ slope
  ② second derivatives (= Hessian) ⇒ curvature
- optimization
  ① first-order algorithms ⇒ use only gradients ⇒ $W \leftarrow W - \alpha \nabla J$
  ② second-order algorithms ⇒ also use Hessian matrix ⇒ $W \leftarrow W - \frac{f'}{f''}$

3. Critical points (= stationary points)



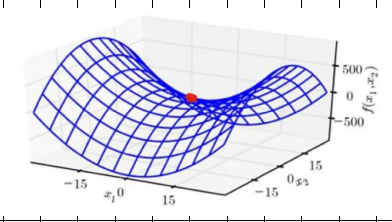
- points with zero slope : $\nabla_x f(x) = 0$
  → derivative gives no info about which direction to move
  → three types : maxima (- curvature), minima (+ curvature), saddle point
- a saddle points : contains both positive and negative curvature

$f(x) = x_1^2 - x_2^2 \longrightarrow \nabla f(2x_1, -2x_2) \Rightarrow \nabla f(0,0) = 0$

$H = \nabla^2 f = \begin{pmatrix} 2 & 0 \\ 0 & -2 \end{pmatrix} \Rightarrow$ eigenvalue : $\lambda_1 = +2, \lambda_2 = -2$

(+), (-) 섞여있음!

saddle point!

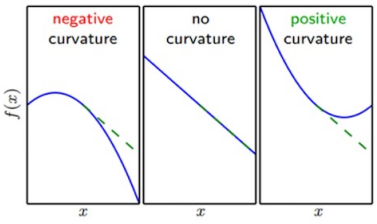


4. Use of second derivative

① to characterize critical points : min, max, saddle

② to measure curvature

③ to predict performance of an update in gradient based opti

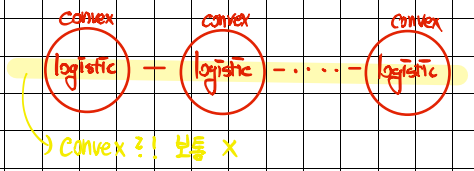


- negative curvature
  - ➢ f decreases faster than gradient predicts
- no curvature
  - ➢ gradient predicts the decrease correctly
- positive curvature
  - ➢ f decreases slower than gradient predicts (eventually increases)

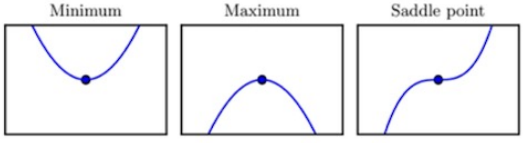
5. In deep learning

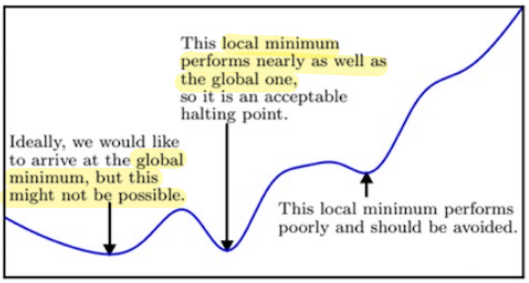
· our objective function has

⇒ many local minima + many saddle points surrounded by very flat regions



⇒ makes optimization very difficult ex) ~~> saddle point 근방 → gradient = 0 → training X



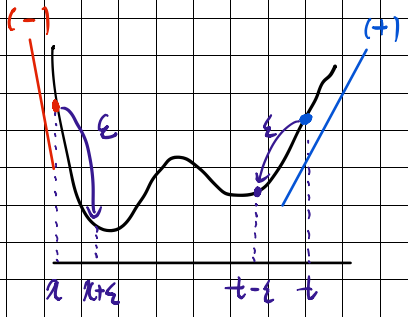


⇒ in high dim : Saddle points are much more common than local minima

Gradient-Based Optimization

- Gradient Descent and its limitations.

6. Method of gradient descent.



- derivative : useful for minimizing a function (objective function)

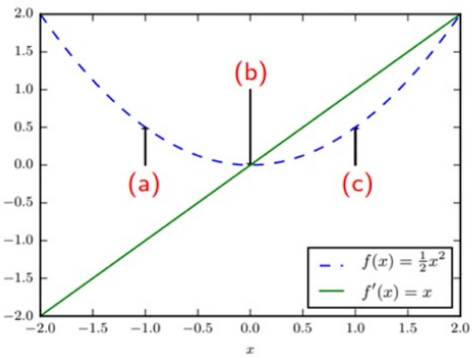
  ⇒ for small ϵ : $f(x - \epsilon \cdot \text{sign}[f'(x)]) < f(x)$

  new para old para

- We can thus reduce f(x) by

  ⇒ moving x in small steps with opposite sign of derivative.

  = method of gradient descent.



$$\underbrace{x'}_{\text{new}} = \underbrace{x}_{\text{old}} - \underbrace{\epsilon}_{\text{learning rate}} \nabla_x f(x)$$

(a) $x < 0$: $f'(x) < 0$
⇒ can decrease f by moving rightward

(b) $x = 0$: $f'(x) = 0$
⇒ gd halts here (global min)

(c) $x > 0$: $f'(x) > 0$
⇒ can decrease f by moving leftward

7. Sgd and its variants

- probably the most used optimization can obtain an unbiased estimate of gradient.

Algorithm 1 gradient descent

1: initialize θ
2: **while** stopping criterion not met **do**
3: sample m examples: $\mathbb{X}_m = \{(x^{(1)}, y^{(1)}), \ldots (x^{(m)}, y^{(m)})\}$
4: compute gradient estimate: $\hat{g} \leftarrow \frac{1}{m} \nabla_\theta \sum_{i=1}^{m} L(y^{(i)}, \hat{y}^{(i)})$ ▷ m forward props
5: apply update: $\theta \leftarrow \theta - \epsilon \hat{g}$ ▷ ϵ: learning rate
6: **end while**

$$\Rightarrow \nabla L(\theta) = \frac{1}{N} \sum_{i=1}^{N} \nabla l_i(\theta)$$

$$g_B(\theta) = \frac{1}{m} \sum_{i=1}^{m} \nabla l_i(\theta)$$

$$E_B[g_B(\theta)] = E_B\left[\frac{1}{m} \sum_{i=1}^{m} \nabla l_i(\theta)\right]$$

$$= \frac{1}{m} \sum_{i=1}^{m} E_B[\nabla l_i(\theta)]$$

$$= \frac{1}{m} \cdot m \cdot \nabla L(\theta)$$

$$= \nabla L(\theta)$$

- three variants (N: total # of examples)

① $m = 1$: sgd

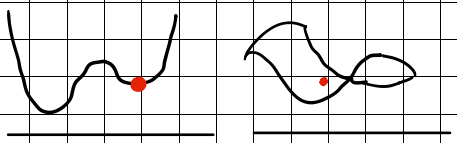
② $1 < m < N$: mini-batch sgd → 이론과 제일 많이 사용.

③ $N = m$: batch gradient descent

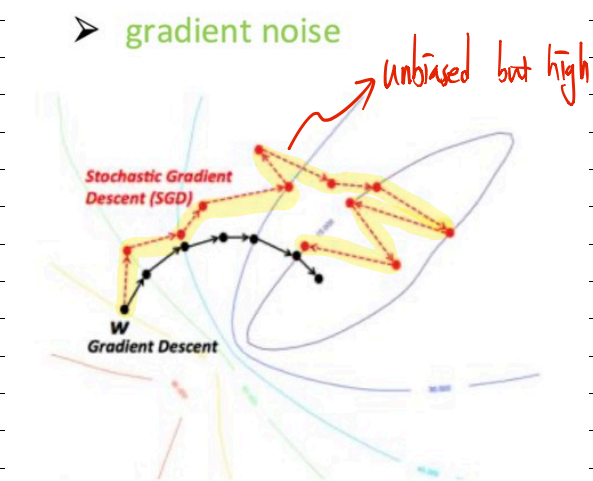
8. Properties of sgd : bad ones

- sgd may suffer in the following situations

① local minima / saddle points

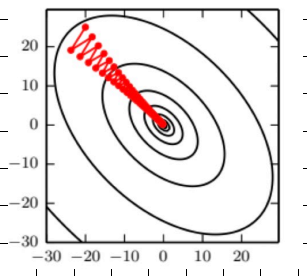


② gradient noise → because of minibatch ?? → poor condition caused ?? → EWMA... →



③ Zero gradient → gradient descent gets stuck

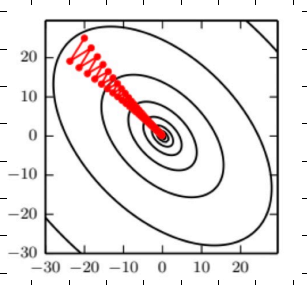
→ poor conditioning of H



9. Poor conditioning of H

↳ 두번 미분한 결과값을 담고 있는 matrix

- consider a point x in multiple dimensions : different second derivative for each direction.
- condition number of Hessian H at x
  - ⇒ measures how much the second derivatives differ from each other
  - ⇒ condition number of H at $X = \max_{ij} \left| \frac{\lambda_i}{\lambda_j} \right|$ (→ 가장 큰 값, → 가장 작은 값)
- when H has a large condition number (poorly conditioned)
  1. gradient descent performs poorly
  2. it is difficult to choose a good step size ϵ
     - ↳ N.M 에서 $\frac{f'}{f''}$ term 에서 f''를 계산할 때 결과값을 실제값으로 반영
     - but 구하는 실제값과 반영한 결과값은 fix된 ϵ에 의해.



- how to handle poor conditioning without directly considering H?

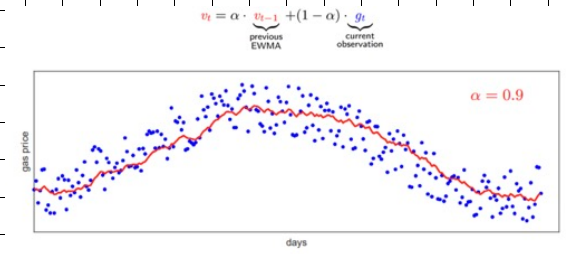
→ Exponetially Weighted Average

10. Exponentially weighted moving average (EWMA)

- given time series : $g_1, g_2 \cdots$
- EWMA defined as :

$$v_t = \begin{cases} g_1 & , t=1 \\ \alpha v_{t-1} + (1-\alpha) g_t & , t>1 \end{cases}$$

(αv_{t-1} : 과거 EWMA, $(1-\alpha) g_t$: 현재값)



$\alpha \in [0, 1]$: degree of weighting decrease (constant smoothing factor)

11. Properties of EWMA

- effective weighting decreases exponentially over time.

⇒ 과거의 값이 영향력(?)이 기하급수 감소

$$v_t = \alpha v_{t-1} + (1-\alpha) g_t$$

$$= \alpha[\alpha v_{t-2} + (1-\alpha) g_{t-1}] + (1-\alpha) g_t$$

$$\vdots$$

$$= \alpha^k v_{t-k} + (1-\alpha)\left[g_t + \alpha g_{t-1} + \alpha^2 g_{t-2} + \cdots + \alpha^{k-1} g_{t-k+1}\right]$$

⇒ weight exponentially decreases toward the past

⇒ this called "exponentially weighted"

- Recall $1 + \alpha + \alpha^2 + \cdots = \frac{1}{1-\alpha}$

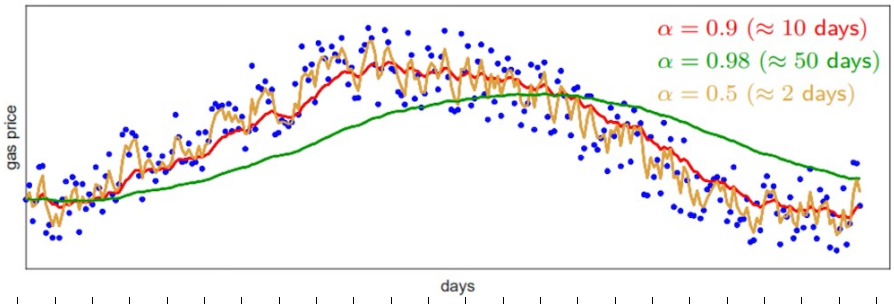
then $v_t = (1-\alpha)\left[g_t + \alpha g_{t-1} + \alpha^2 g_{t-2} + \alpha^3 g_{t-3} + \cdots\right]$

$$= \frac{g_t + \alpha g_{t-1} + \alpha^2 g_{t-2} + \cdots}{1 + \alpha + \alpha^2 + \cdots}$$

↳ effective number of observations

- higher α (= more weight to past, less weight to present)
  - ➢ smoother curve ← averaging over more days
  - ➢ shifted further ← averaging over larger window

⇒ curve adapts more _slowly_ to changes with more latency



12. Bias correction : Adam

⇒ 우리가 쓰는 MA : EWMA

$v_t = (1-\alpha) g_t + \alpha v_{t-1}$, α 값에 따라 과거의 값들이 존재하지 않을수록 커진다. 그래서 무시할 수 있는 것처럼 0으로 감쇠... $\longrightarrow 0$

ex) $v_t = (1-\alpha) \sum_{k=0}^{t-1} \alpha^k g_{t-k}$ ($\because$ stationary 하다라고 $E(g_t) = \mu$)

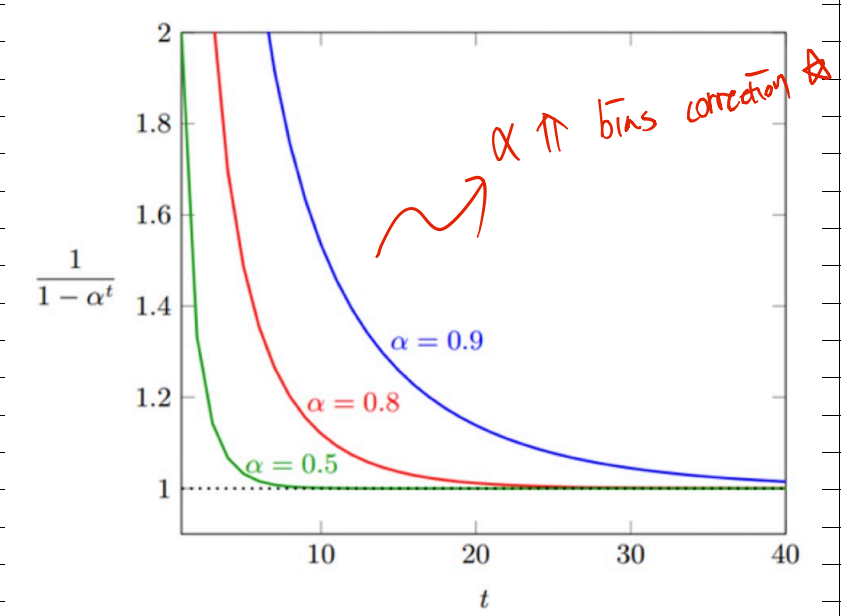
$E(v_t) = (1-\alpha) \mu \sum_{k=0}^{t-1} \alpha^k$ ($\because \sum_{k=0}^{t-1} \alpha^k = 1 + \alpha + \cdots + \alpha^{t-1} = \frac{1-\alpha^t}{1-\alpha}$)

$= (1-\alpha^t) \mu$

stationary 하기 때문에 μ가 나와야 하는데

$(1-\alpha^t)$ 만큼 biased된 값이 나옴!

그래서 $\hat{v}_t = \frac{v_t}{1-\alpha^t}$ 로 보정!



— Gradient Descent with Momentum

13. Method of momentum

- sgd : very popular but sometimes slow ⇒ unbiased but high variance (← gradient noise by minibatch & poor cond)
- Method of momentum ⇒ designed to accelerate learning
  especially in face of ① high curvature ⇒ Hessian λ_{max} ↑ ⇒ poor cond
  ② small but consistent gradients ⇒ saddle points
  ③ noisy gradients
  (high variance gradients ← ②, ③)
- Common algorithm
  → accumulates exponentially decaying moving average of past gradients
  ⇒ gradient descent + 과거 gradient의 EWMA
  → then continues to move in their direction.

14. Gradient descent with momentum

- Idea : compute EWMA of gradients and use it to update weights
  → gradient를 바로 쓰지 않고 과거 gradient의 EWMA를 활용
  → noise ↓, consistent한 direction gradient.

- let $g \triangleq \nabla_\theta J(\theta)$ (θ represents W and b altogether)

form 1
$$v \leftarrow \alpha v - \epsilon g, \quad \theta \leftarrow \theta + v \implies \theta \leftarrow \theta - \epsilon\left(g + \frac{\alpha}{-\epsilon} v\right)$$

form 2
$$v \leftarrow \alpha v + (1-\alpha) g, \quad \theta \leftarrow \theta - \epsilon v \implies^{3} \theta \leftarrow \theta - \tilde{\epsilon}\left(g + \frac{\alpha}{1-\alpha} v\right)$$

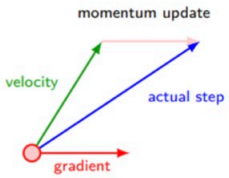
form 3
$$v \leftarrow \alpha v + g, \quad \theta \leftarrow \theta - \epsilon v \implies \theta \leftarrow \theta - \epsilon(g + \alpha v)$$

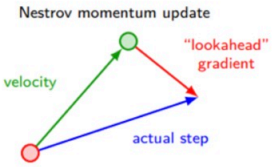
- bottom line: θ is updated by linear combination of gradient and velocity

$$\theta \leftarrow \theta - \epsilon\left(\underbrace{g}_{\text{gradient}} + \text{constant} \cdot \underbrace{v}_{\text{velocity}}\right)$$

15. Nesterov momentum

- difference from standard momentum:
  - _where_ gradient $g = \nabla_\theta J$ is evaluated





standard momentum	Nesterov momentum
g evaluated at current position θ (red circle)	g evaluated at "lookahead" position $\theta + \alpha v$ (green circle)

- Nesterov momentum update rule

: $v \leftarrow \alpha v - \varepsilon \nabla_\theta J(\theta + \alpha v)$

$\theta \leftarrow \theta + v$

⇒ gradient is evaluated after the current velocity is applied

→ interpreted as adding a correction factor to standard momentum

momentum의 현재속도를 gradient로 보정하는 느낌..?

↳ 보통 이거씀 optim.SGD(momentum=0.9, nesterov=True)

$v \leftarrow 0.9v - \varepsilon \nabla_\theta J(\theta + 0.9v)$

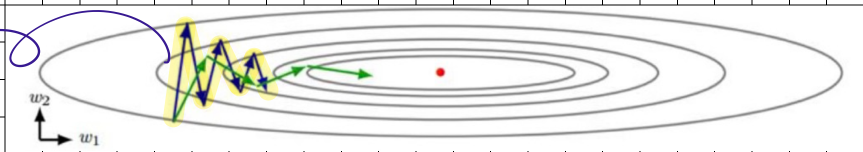
$\theta \leftarrow \theta + v$

— Pre parameter Adaptive Learning Rates : parameter 별로 다른 learning Rate 조정!

- optimization methods explained so far
  - → set learning rate ε globally and equally for all parameter : $w \leftarrow w - \varepsilon g$
- methods presented now : AdaGrab, RMSprop, Adam
  - → adaptively tune ε for each parameter

16. Motivation

- recall : limitation of gradient descent



gradient는 가장 빠르게 증가하는 방향이므로 & Minibatch?

→ goal : move horizontally

→ problem : huge vertical oscillations (→ cause by minibatch & curvature so high?)

→ solution : we want to ① slow down learning vertically
② speed up learning horizontally

→ how? without relying on H explicitly?

17. AdaGrab

- individually adapts learning rate of each direction (each parameter)
  - → steep direction (large $\frac{\partial J}{\partial \theta_j}$) : slow down learning
  - → gently sloped direction (small $\frac{\partial J}{\partial \theta_j}$) : speed up learning

→ 이렇게 가중치를 상황따라 다르게 어케 줄까? + 자동으로 not hyperparam

S : ① Adadelta : fix window
② EWMA

- adjusts learning rates per parameter

$$\varepsilon_j = \frac{\varepsilon}{\sqrt{\sum_{\text{all past iter}} (g_j \cdot g_j)}}$$

ε → Hyperparameter

ε_j → parameter which depends on past gradients (g^2)

→ learning rate of parameter θ_j

- net effect : greater process in more gently sloped direction

p : monotonically decreasing : too aggressive
⇒ stops learning too early!!

18. RMSProp (root-mean-square prop)

- modifies AdaGrad to perform better in (in DL) non-convex setting
  - → changes gradient accumulation to EWMA → mom ⇒ g 자체에 대한 EWMA
    RMS ⇒ g^2 (크기)에 대한 EWMA
- use of exponentially decaying average allows RMSprop to
  - → discard history from extreme past
  - ⇒ converge rapidly after finding a convex bowl

- comparison (r : accumulation variable)

① AdaGrad

$$r \leftarrow r + g \odot g$$

$$\Delta\theta \leftarrow -\frac{\varepsilon}{\sqrt{\delta + r}} \odot g$$

$$\theta_{new} \leftarrow \theta_{old} + \Delta\theta_{old}$$

② RMSprop

$$r_t \leftarrow \rho r_{t-1} + (1-\rho) g_t \odot g_t \quad (\because g_t = \nabla L(\theta_t))$$

$$\Delta\theta_t \leftarrow -\frac{\varepsilon}{\sqrt{\delta + r_t}} \odot g_t$$

$$\theta_{t+1} \leftarrow \theta_t + \Delta\theta_t$$

19. Adam (adaptive moment estimation)

- Idea : RMSProp + momentum with bias correction
- for each iteration

① compute gradient $g = \nabla L(\theta)$

② update first moment : $s \leftarrow \rho_1 s + (1-\rho_1) g$ → momentum

③ update second moment : $r \leftarrow \rho_2 r + (1-\rho_2) g \odot g$ → RMSProp

④ bias correction

① $\hat{s} \leftarrow \frac{s}{1-\rho_1^t}$ ② $\hat{r} \leftarrow \frac{r}{1-\rho_2^t}$

⑤ update parameter

$$\theta \leftarrow \theta - \varepsilon \frac{\hat{s}}{\sqrt{\hat{r} + \delta}}$$

- recommended values in the paper
  - learning rate ϵ : needs tuning (suggested default: 0.001)
  - for momentum ρ_1 : 0.9
  - for RMSProp ρ_2 : 0.999
  - for stability δ : 10^{-8}
- Adam: often works better than RMSProp
  - recommended as the default algorithm to use
  - alternative to Adam worth trying: sgd + Nestrov momentum

Algorithm 2 Adam optimizer

Require:
- step size ϵ
- exponential decay rates for moment estimates, ρ_1 and ρ_2 in $[0, 1)$
- small constant δ used for numerical stabilization
- initial parameters $\boldsymbol{\theta}$

initialize 1st and 2nd moment variables $s = 0$, $r = 0$
initialize time step $t = 0$
while stopping criterion not met **do**
  sample a minibatch $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$
  compute gradient: $g \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
  $t \leftarrow t + 1$
  update biased first moment estimate: $s \leftarrow \rho_1 s + (1 - \rho_1) g$
  update biased second moment estimate: $r \leftarrow \rho_2 r + (1 - \rho_2) g \odot g$
  correct bias in first moment: $\hat{s} \leftarrow \frac{s}{1-\rho_1^t}$
  correct bias in second moment: $\hat{r} \leftarrow \frac{r}{1-\rho_2^t}$
  compute update: $\Delta\boldsymbol{\theta} = -\epsilon \frac{\hat{s}}{\sqrt{\hat{r}} + \delta}$ (operations applied element-wise)
  apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$
end while