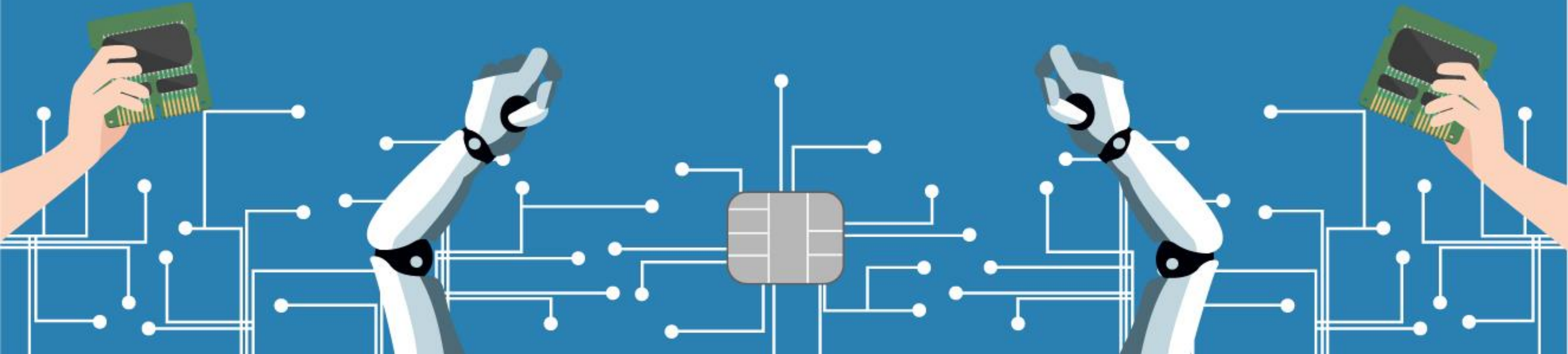
Kuggle

2020_02 Kuggle 정규세션_W10

2020.11.24



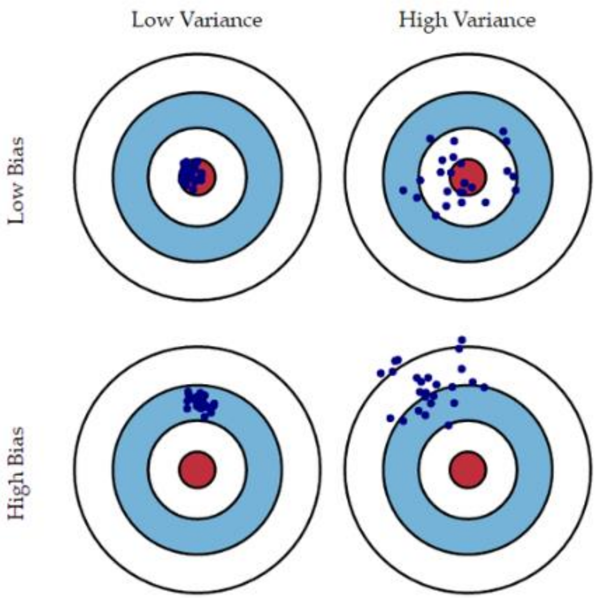
목차

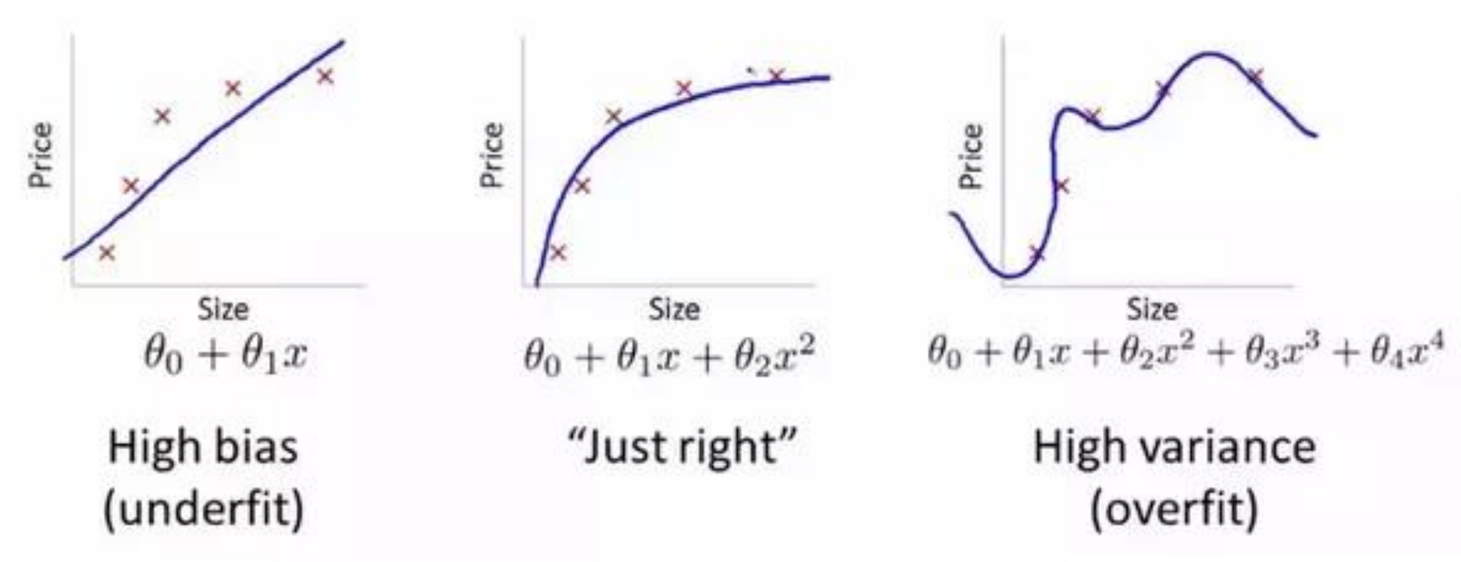
CONTENTS

01. Regularization

- Bias와 variance
- L2
- normalization
- Early stopping
- Data augmentation

1. Regularization – bias & variance

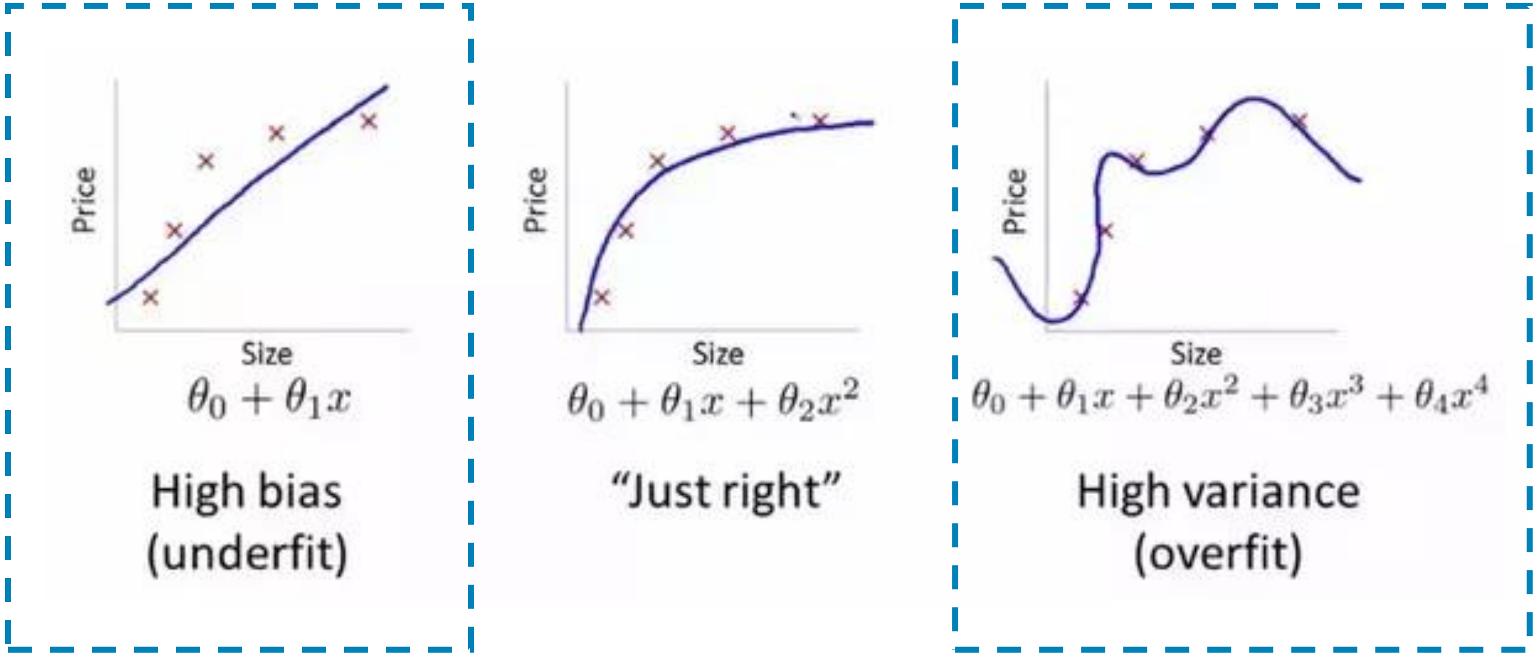




학습 과정에서 우리가 신경 써야 하는 두 가지, bias와 variance

- High bias : training set에서의 성능이 좋지 못한 경우 (underfitting)
- High variance : training set에서의 성능과 validation set에서의 성능차이가 심한 경우 (overfitting)

1. Regularization – bias & variance



High bias를 해결하는 방법

1. Layer의 개수를 늘린다.
2. Epoch을 늘린다.
3. 새로운 architecture를 시도해본다.

High variance를 해결하는 방법

1. Regularization 수행한다.
2. Training data를 늘린다.
3. 새로운 architecture를 시도해본다.

1. Regularization – L2 regularization

Regularization에는 다양한 방법이 존재합니다.

- L2 regularization
- Dropout
- Early stopping
- Data augmentation

L1 regularization on least squares:

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \sum_j \left(t(\mathbf{x}_j) - \sum_i w_i h_i(\mathbf{x}_j) \right)^2 + \boxed{\lambda \sum_{i=1}^{k} |w_i|}$$

L2 regularization on least squares:

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \sum_j \left(t(\mathbf{x}_j) - \sum_i w_i h_i(\mathbf{x}_j) \right)^2 + \boxed{\lambda \sum_{i=1}^{k} w_i^2}$$

DL에서 L1, L2 norm 수행할 때 기본 가정

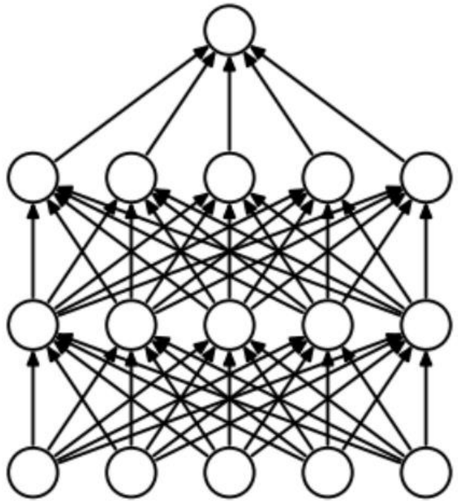
: Parameter의 값이 커질수록 over-fitting이 발생한다.

따라서 L1, L2 항을 추가하여 parameter의 크기를 제약하는 것!

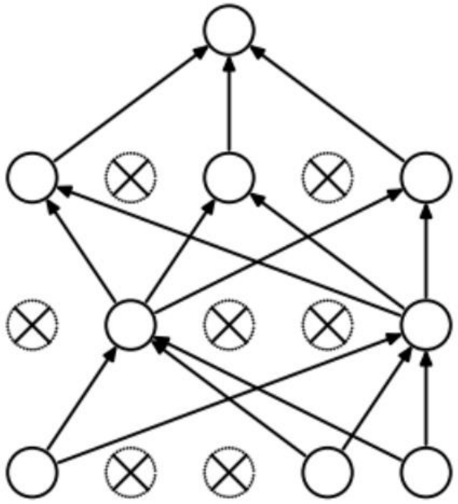
1. Regularization – dropout

Regularization에는 다양한 방법이 존재합니다.

- L2 regularization
- Dropout
- Early stopping
- Data augmentation



(a) Standard Neural Net



(b) After applying dropout.

Dropout은 "node를 꺼버리는" 기법입니다.

이 표현의 구체적인 의미는 activation function을 통과한 값 중 일부를 random하게 0으로 만들어준다는 의미입니다.
Random하게 만들어주는 비율은 hyper-parameter 입니다.

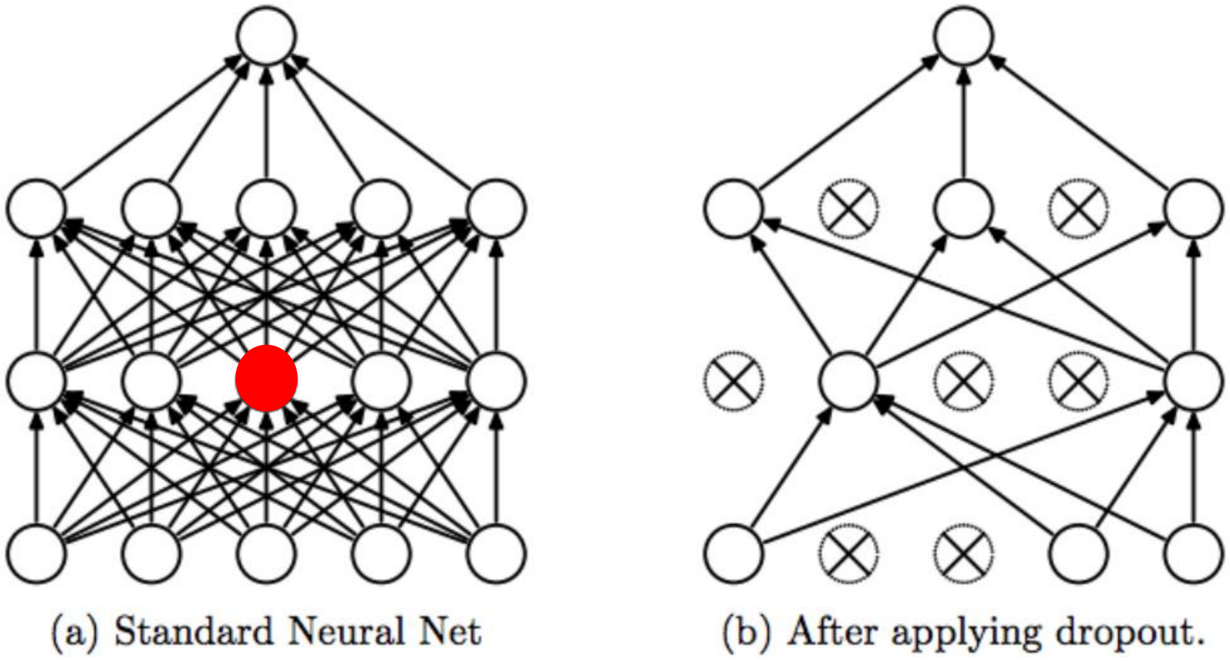
1. Regularization – dropout

Q1. dropout이 어떻게 overfitting을 방지할까?

A1. dropout은 random하게 node를 0으로 만들어버립니다.

이렇게 되면 예측 값이 특정 노드에 의존하는 현상을 줄일 수 있고,

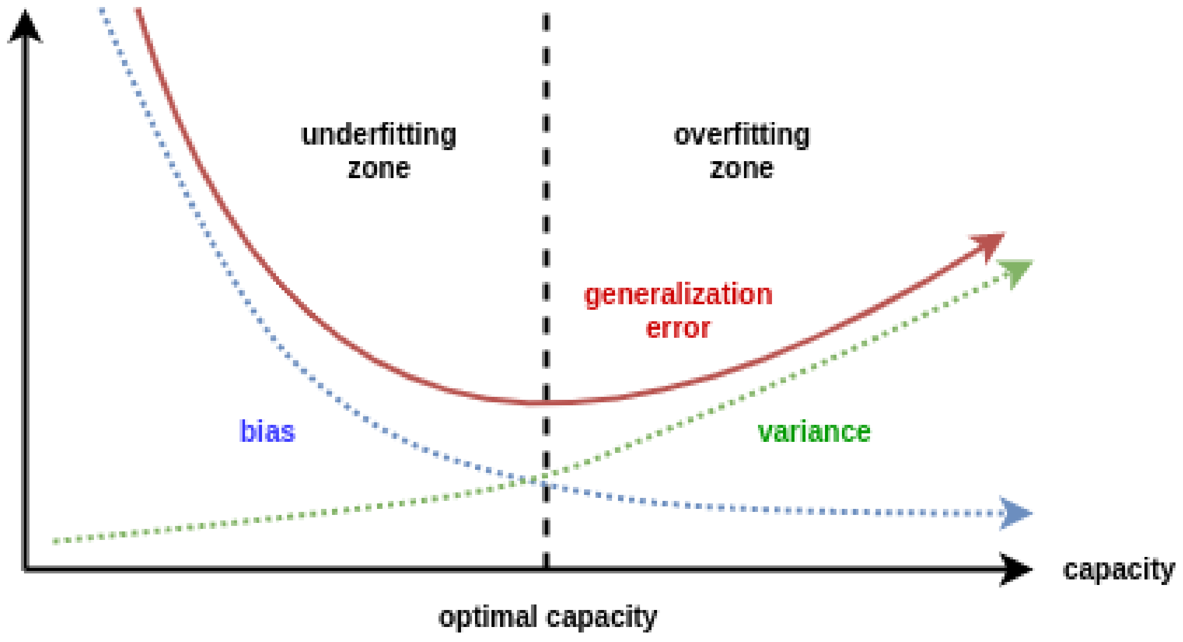
따라서 여러 노드에게 예측의 기여도를 분산시킴으로써 overfitting을 방지할 수 있습니다.



1. Regularization – early stopping

Regularization에는 다양한 방법이 존재합니다.

- L2 regularization
- Dropout
- Early stopping
- Data augmentation



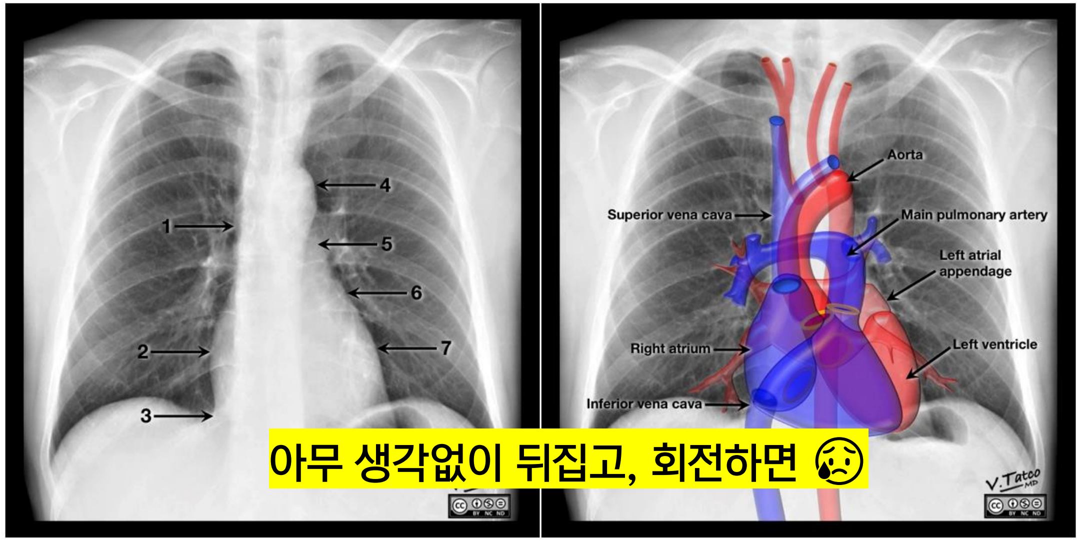
Validation set에서의 loss가 증가하는 시점에서 학습을 멈춘다.

1. Regularization – data augmentation

Regularization에는 다양한 방법이 존재합니다.

- L2 regularization
- Dropout
- Early stopping
- Data augmentation

회전, 뒤집기, 확대/축소, 밝기 변경 …



아무 생각없이 뒤집고, 회전하면 😥

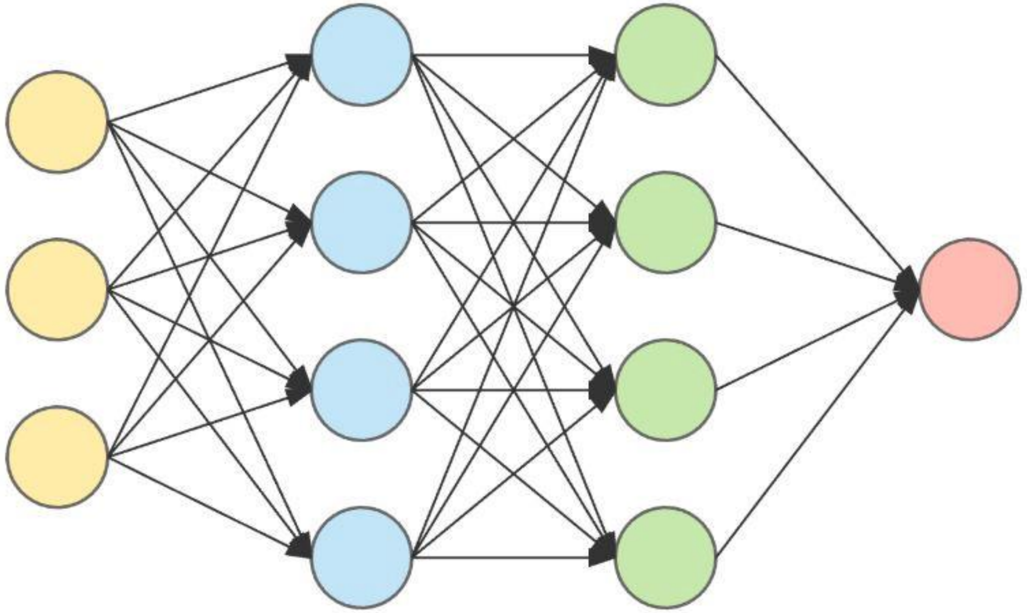
CONTENTS

02. Weight initialization

- Xavier
- He

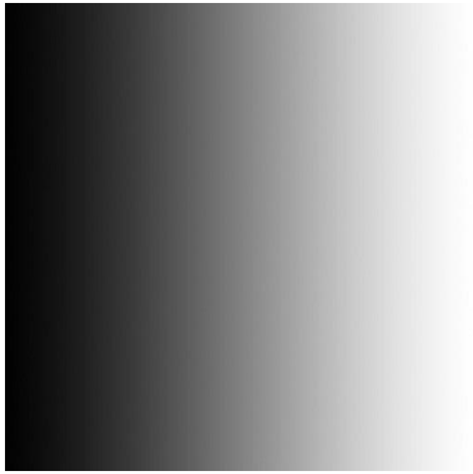
2. Weigh initialization

초기화 … ㅇㅏ무거나 … 대충 …

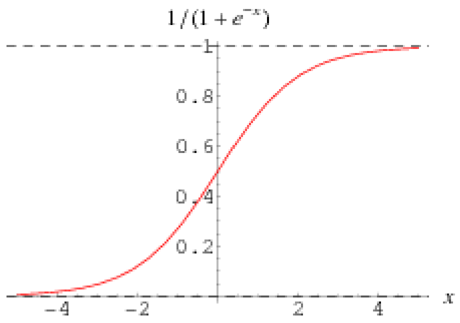


Gradient exploding

Gradient vanishing

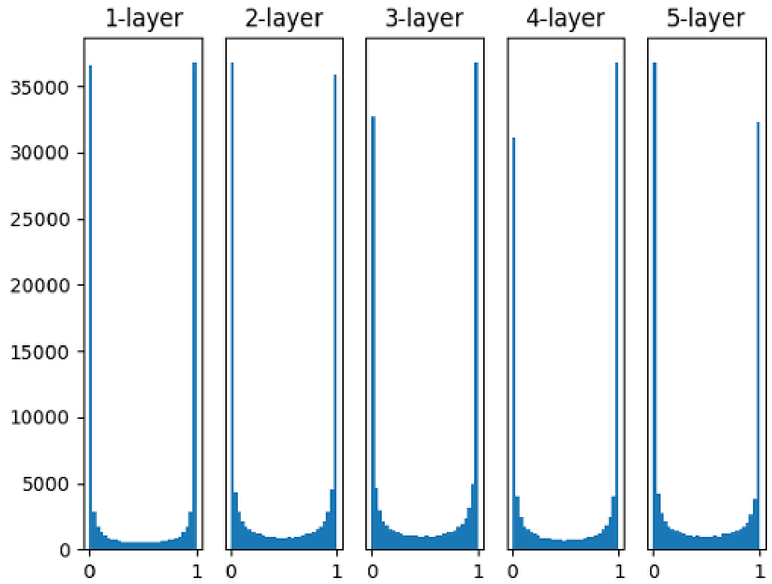


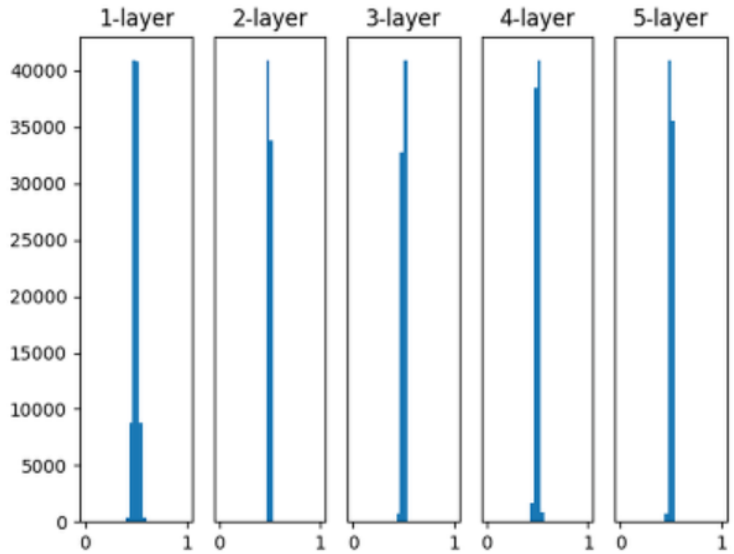
2. Weigh initialization



정규분포 따르도록 초기화 해봐 ~~~!

* activation function으로 sigmoid를 사용한 경우





분산이 큰 경우 z값이 크거나 작아져 출력 값이 0과 1에 치우침
-〉 gradient exploding

분산이 작은 경우 z값이 0 근처에 밀집
-〉 gradient vanishing

2. Weigh initialization

Summary of gradient descent

$$dz^{[2]} = a^{[2]} - y$$

$$dW^{[2]} = dz^{[2]} a^{[1]^T}$$

$$db^{[2]} = dz^{[2]}$$

$$dz^{[1]} = W^{[2]T} dz^{[2]} * g^{[1]\prime}(z^{[1]})$$

$$dW^{[1]} = dz^{[1]} x^T$$

$$db^{[1]} = dz^{[1]}$$

$$dZ^{[2]} = A^{[2]} - Y$$

$$dW^{[2]} = \frac{1}{m} dZ^{[2]} \boxed{A^{[1]^T}}$$

$$db^{[2]} = \frac{1}{m} np.sum(dZ^{[2]}, axis = 1, keepdims = True)$$

$$dZ^{[1]} = W^{[2]T} dZ^{[2]} * g^{[1]\prime} \boxed{(Z^{[1]})}$$

$$dW^{[1]} = \frac{1}{m} dZ^{[1]} X^T$$

$$db^{[1]} = \frac{1}{m} np.sum(dZ^{[1]}, axis = 1, keepdims = True)$$

Andrew Ng

초기 weight가 너무 작으면,

FC layer 통과한 값도 작고, Activation 통과한 값도 작아서

Gradient vanishing

초기 weight가 너무 크면,

FC layer 통과한 값도 작고, Activation 통과한 값도 작아서

Gradient exploding

2. Weigh initialization

두 아이디어 모두 노드의 수가 많아질수록 weight를 더 작게 만들어주는 방식으로 초기화

1. Xavier initialization
(tanh activation 사용 시 추천!)

$$W \sim N(0, Var(W))$$

$$Var(W) = \sqrt{\frac{2}{n_{in} + n_{out}}}$$

2. He initialization
(relu activation 사용 시 추천!)

$$W \sim N(0, Var(W))$$

$$Var(W) = \sqrt{\frac{2}{n_{in}}}$$

CONTENTS

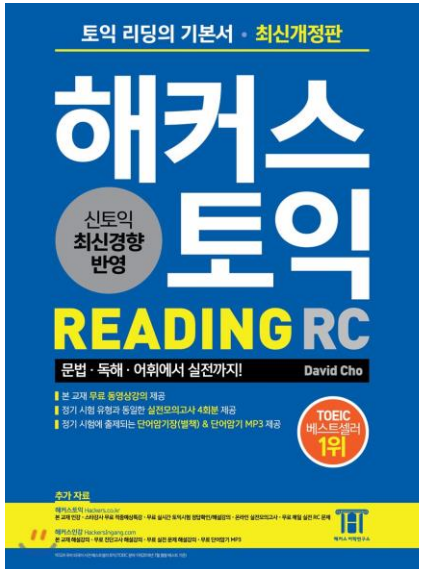
03. Optimizer

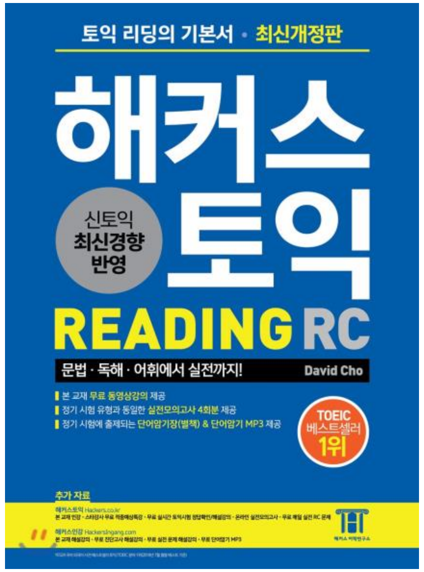
- Gradient Descent
- Batch Gradient Descent
- Stochastic Gradient Descent
- Momentum
- RMSProp
- Adam

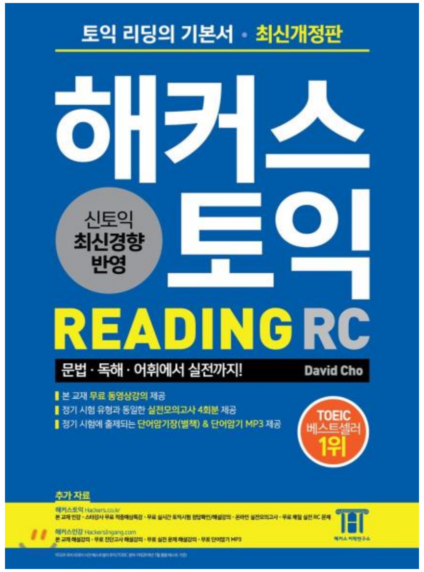
3. Optimizer

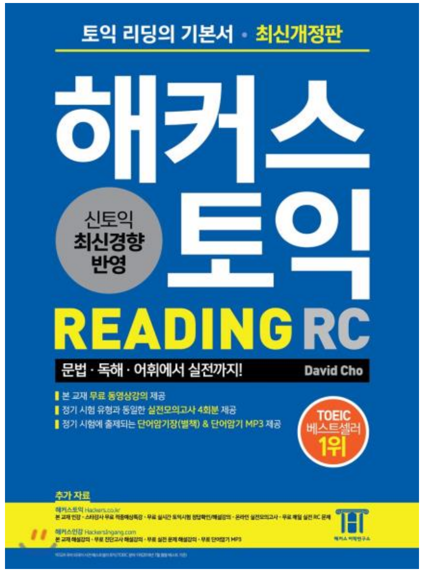
🤔 앗 잠깐1, iteration?

한 번 보고 ~~	두 번 보고 ~~	세 번 보고 ~~	네 번 보고 ~~





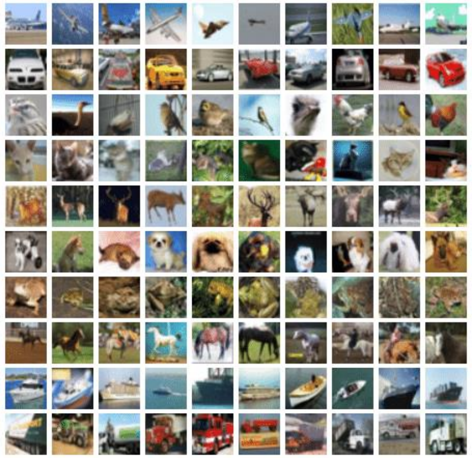




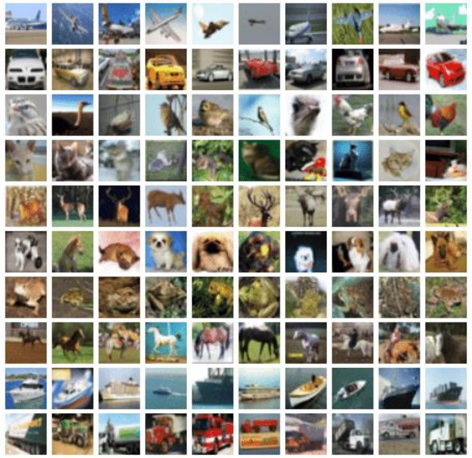
3. Optimizer

앗 잠깐1, iteration?

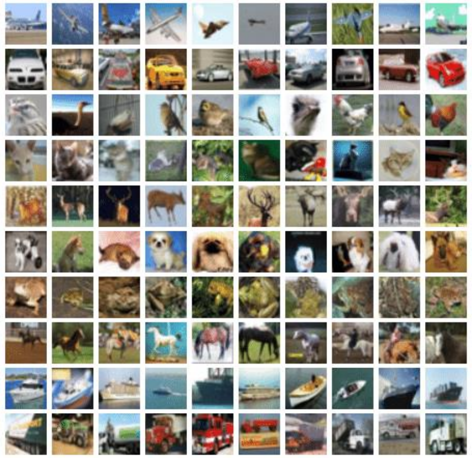
한 번 보고 ~~



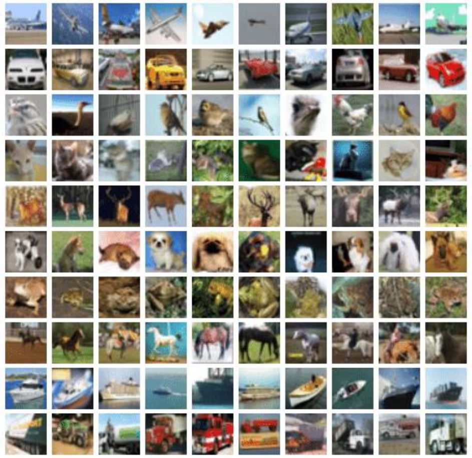
두 번 보고 ~~



세 번 보고 ~~



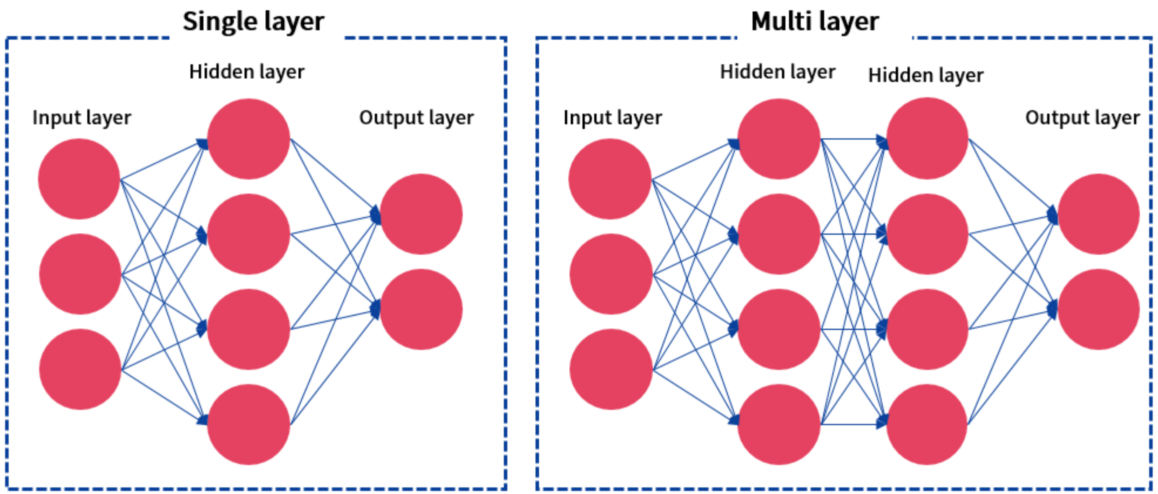
네 번 보고 ~~



```
for epoch in range(num_epoch):
        전체 데이터를 이용해 학습 진행
```

3. Optimizer

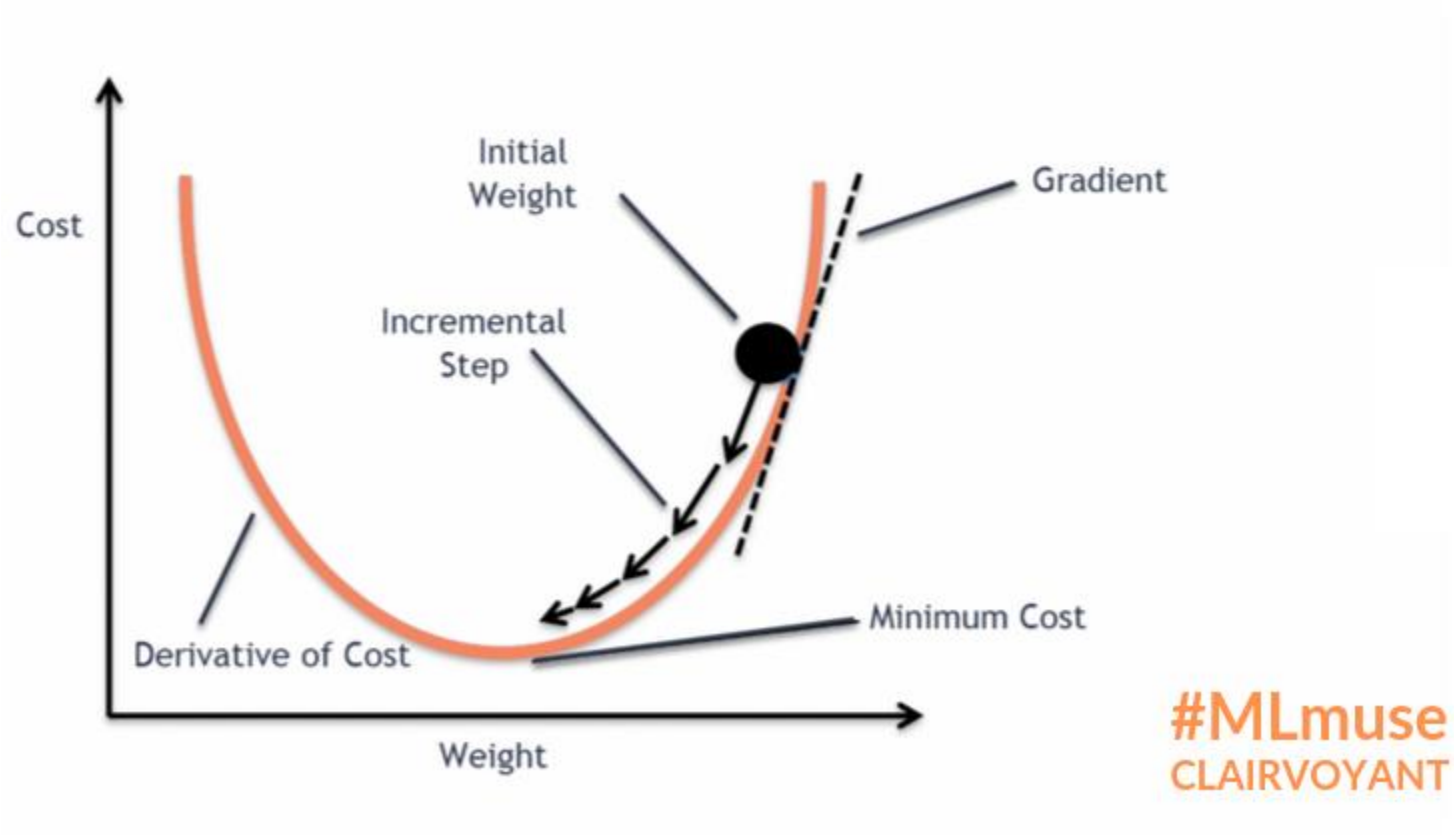
🤔 앗 잠깐2, vectorize?



For (x1, x2) in datas:
z1 = w11*x1+w12*x2+b1
z2 = w21*x1+w22*x2+b2
z3 = w31*x1+w32*x2+b3
z4 = w41*x1+w42*x2+b4
...

$$\begin{bmatrix} b1 & w11 & w12 \\ b2 & w21 & w22 \\ b3 & w31 & w32 \\ b4 & w41 & w42 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 \\ x11 & x21 & x31 \\ x12 & x22 & x32 \end{bmatrix}$$

3. Optimizer

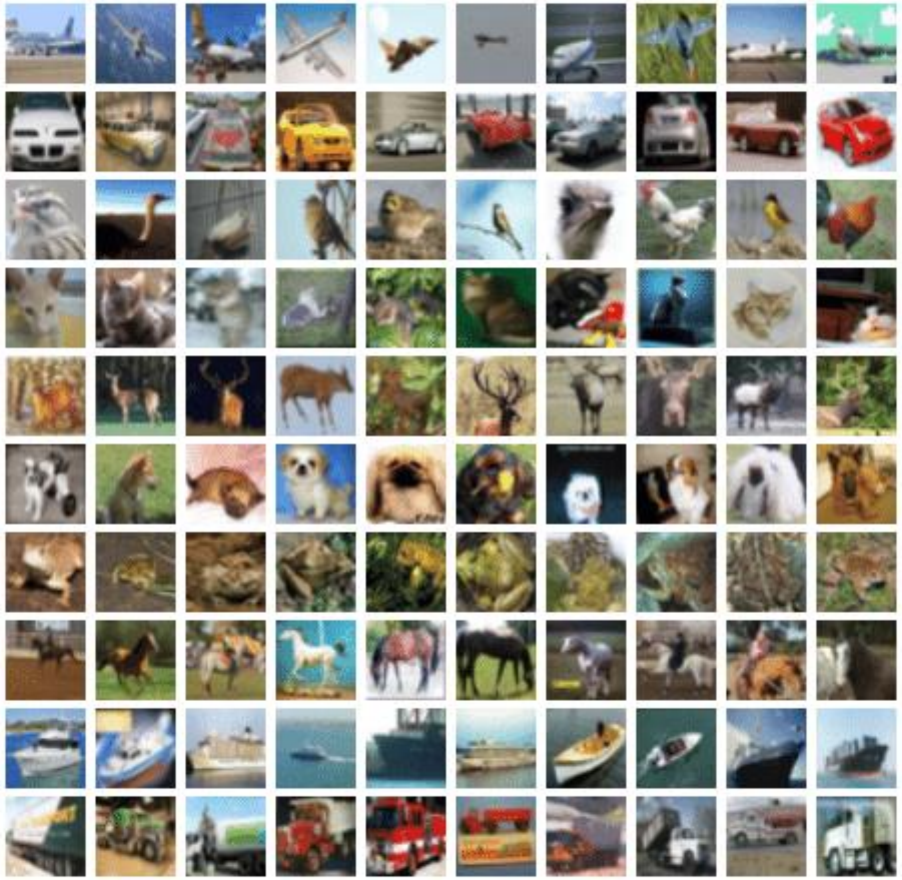


이 공식 너무 많이 보셨죠! 😉

$$W := W - \alpha \frac{\partial}{\partial W} cost(W)$$

3. Optimizer

전체 데이터셋



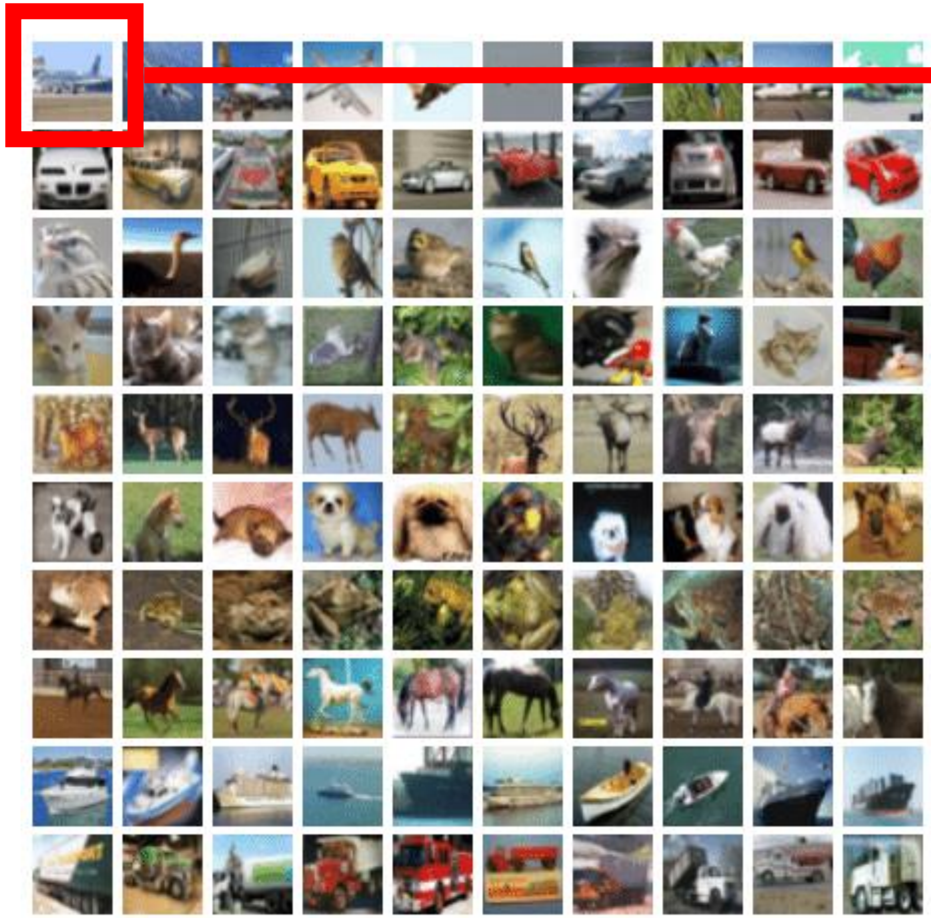
$$W := W - \alpha \frac{\partial}{\partial W} cost(W)$$

Gradient descent

- Vectorize 연산 가능하다
- 전체 데이터셋의 특징을 반영하여 업데이트
- Iteration 1번에 parameter update 1번

3. Optimizer

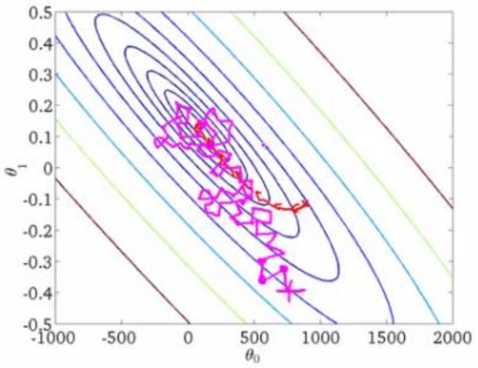
데이터 하나



$$W := W - \alpha \frac{\partial}{\partial W} cost(W)$$

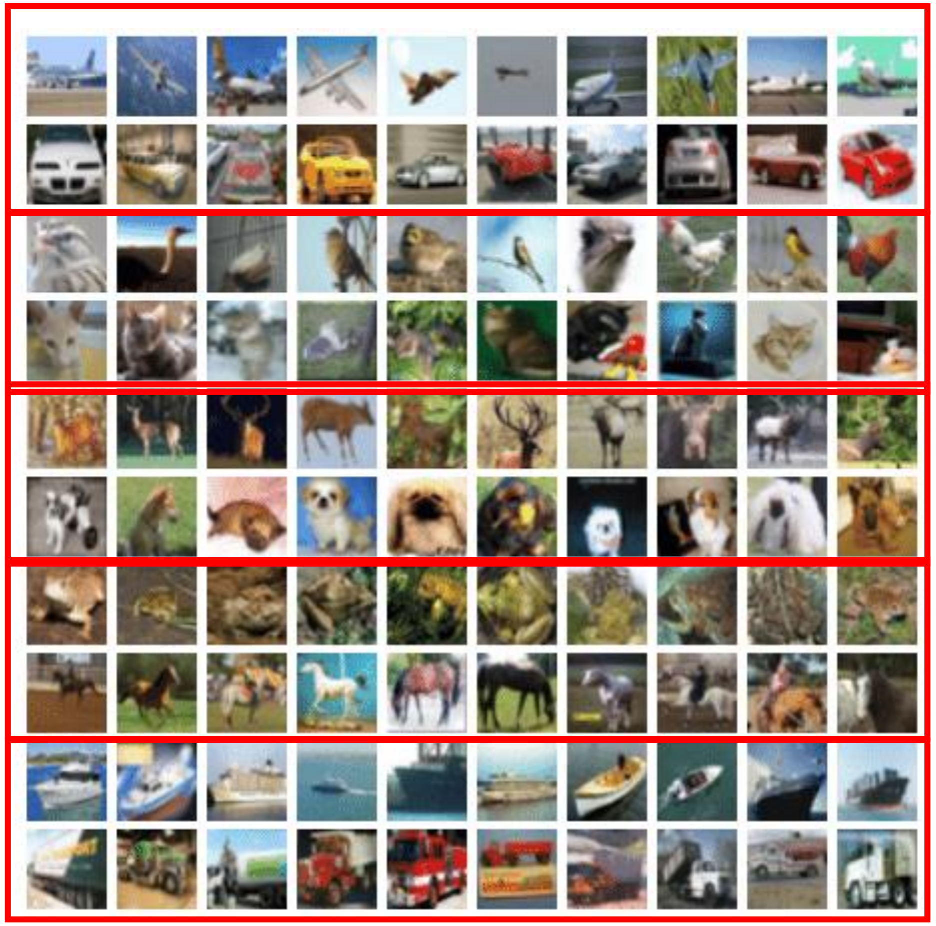
Stochastic gradient descent

- Vectorize 연산 불가능하다
- 지역적 특징을 너무 반영



3. Optimizer

배치 단위 학습



batch1

$$W := W - \alpha \frac{\partial}{\partial W} cost(W)$$

batch2

batch3

batch4

batch5

Batch gradient descent

- Vectorize 가능하다
- Iteration 1번에 여러 번 parameter update 가능하다
- -> 이거 사용하자!

3. Optimizer

$$W := W - \alpha \frac{\partial}{\partial W} cost(W)$$

Gradient descent

3. Optimizer

🤔 앗 잠깐3, exponentially weighted average?

Temperature in London

$\theta_1 = 40°F$ 4°C ←

$\theta_2 = 49°F$ 9°C

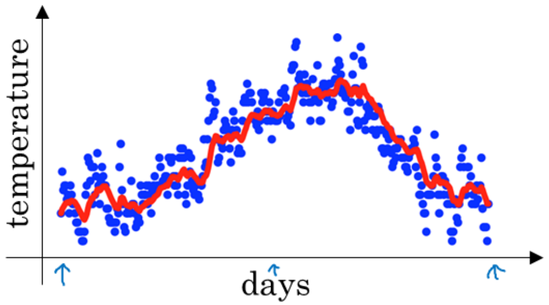
$\theta_3 = 45°F$ ⋮

⋮

$\theta_{180} = 60°F$ 15°C

$\theta_{181} = 56°F$ ⋮

⋮



$V_0 = 0$

$V_1 = 0.9 V_0 + 0.1 \theta_1$

$V_2 = 0.9 V_1 + 0.1 \theta_2$

$V_3 = 0.9 V_2 + 0.1 \theta_3$

⋮

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

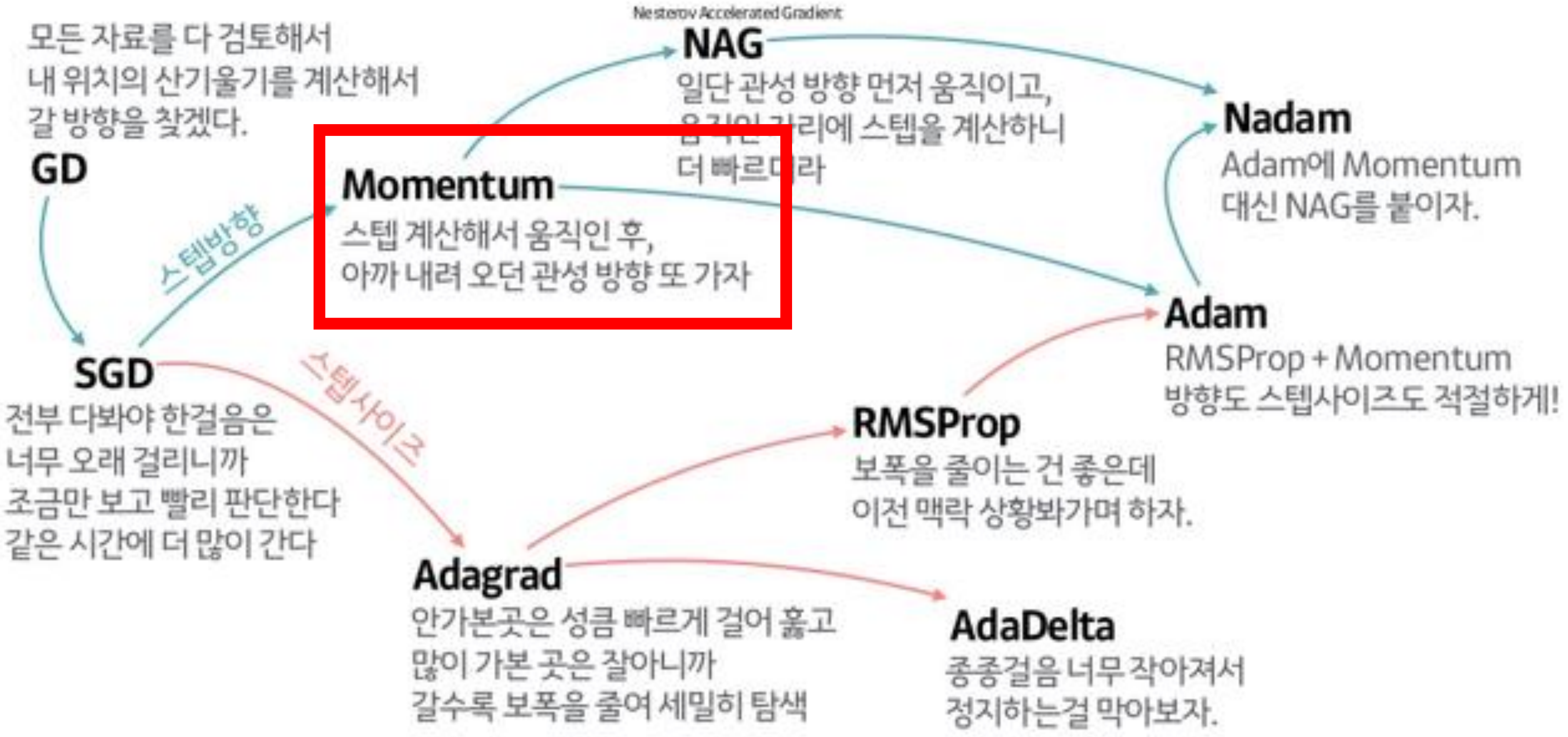
Andrew Ng

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

- V_t는 누적된 정보 (가중평균)
- theta_t는 현재 시점의 정보

- V_t는 1/(1-beta) 시점동안의 평균 정보를 가지고 있다고 보면 된다.
  - beta = 0.9 -> 지난 10일 동안의 가중평균
  - beta = 0.99 -> 지난 100일 동안의 가중평균

3. Optimizer

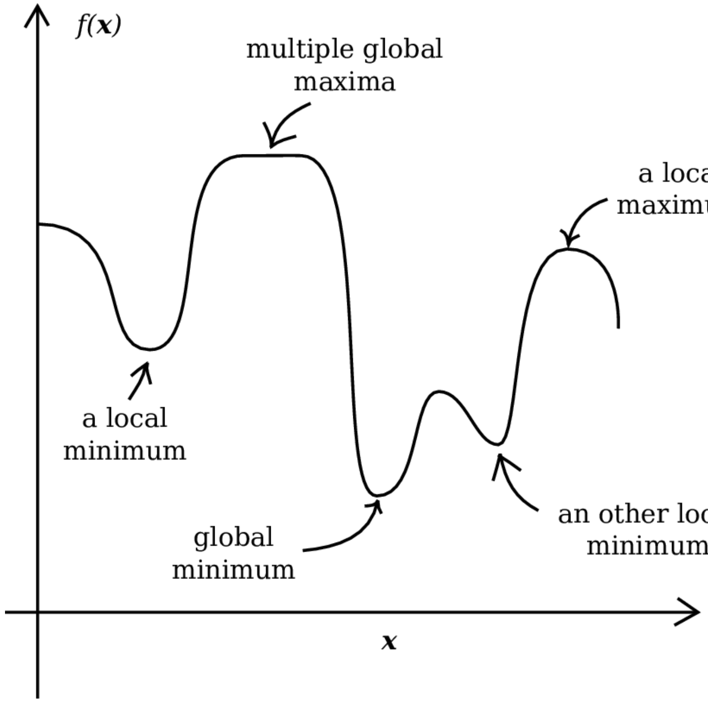


모든 자료를 다 검토해서
내 위치의 산기울기를 계산해서
갈 방향을 찾겠다.
GD
스텝방향
SGD
전부 다봐야 한걸음은
너무 오래 걸리니까
조금만 보고 빨리 판단한다
같은 시간에 더 많이 간다
스텝사이즈
Nesterov Accelerated Gradient
NAG
일단 관성 방향 먼저 움직이고,
움직인 자리에 스텝을 계산하니
더 빠르더라
Momentum
스텝 계산해서 움직인 후,
아까 내려 오던 관성 방향 또 가자
Nadam
Adam에 Momentum
대신 NAG를 붙이자.
Adam
RMSProp + Momentum
방향도 스텝사이즈도 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.

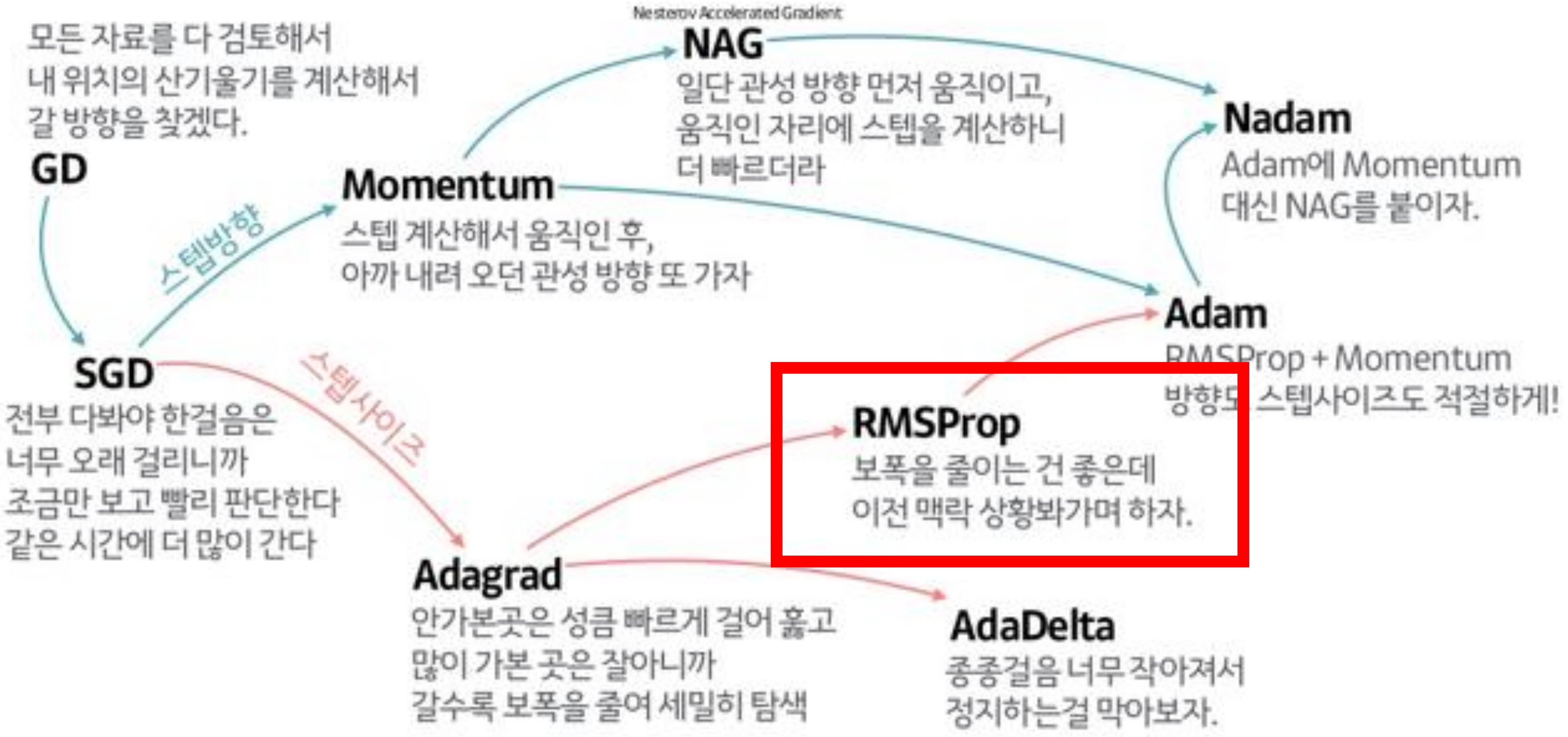
3. Optimizer

- momentum : 아까 내려오던 관성 방향으로 가자! (**방향 조절**)
  - $$VdW = \beta VdW + (1-\beta)dW$$
  - $$Vdb = \beta Vdb + (1-\beta)db$$
  - $$W := W - \alpha VdW$$
  - $$b := b - \alpha Vdb$$

일반적으로 beta는 0.9로 설정한다



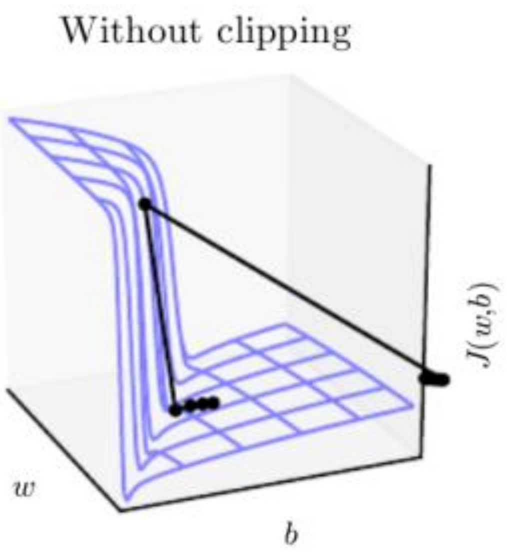
3. Optimizer

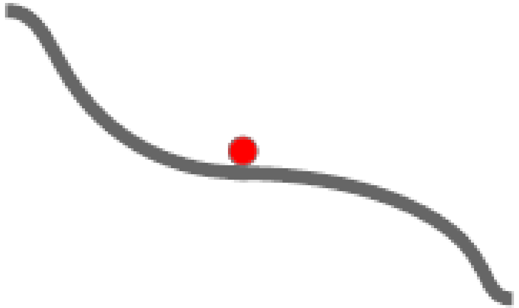


모든 자료를 다 검토해서
내 위치의 산기울기를 계산해서
갈 방향을 찾겠다.
GD
스텝방향
Momentum
스텝 계산해서 움직인 후,
아까 내려 오던 관성 방향 또 가자
Nesterov Accelerated Gradient
NAG
일단 관성 방향 먼저 움직이고,
움직인 자리에 스텝을 계산하니
더 빠르더라
Nadam
Adam에 Momentum
대신 NAG를 붙이자.
Adam
RMSProp + Momentum
방향도 스텝사이즈도 적절하게!
SGD
전부 다봐야 한걸음은
너무 오래 걸리니까
조금만 보고 빨리 판단한다
같은 시간에 더 많이 간다
스텝사이즈
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.

3. Optimizer

- RMSProp(<u>*R*</u>oot <u>*M*</u>ean <u>*S*</u>quare <u>*PROP*</u>agation) : 상황에 따라 step size 조절하자! **<u>(스텝 조절)</u>**
  - $$SdW = \beta SdW + (1-\beta)dW^2$$
  - $$Sdb = \beta Sdb + (1-\beta)db^2$$
  - $$W := W - \alpha \frac{dW}{\sqrt{SdW} + \epsilon}$$
  - $$b := b - \alpha \frac{db}{\sqrt{Sdb} + \epsilon}$$

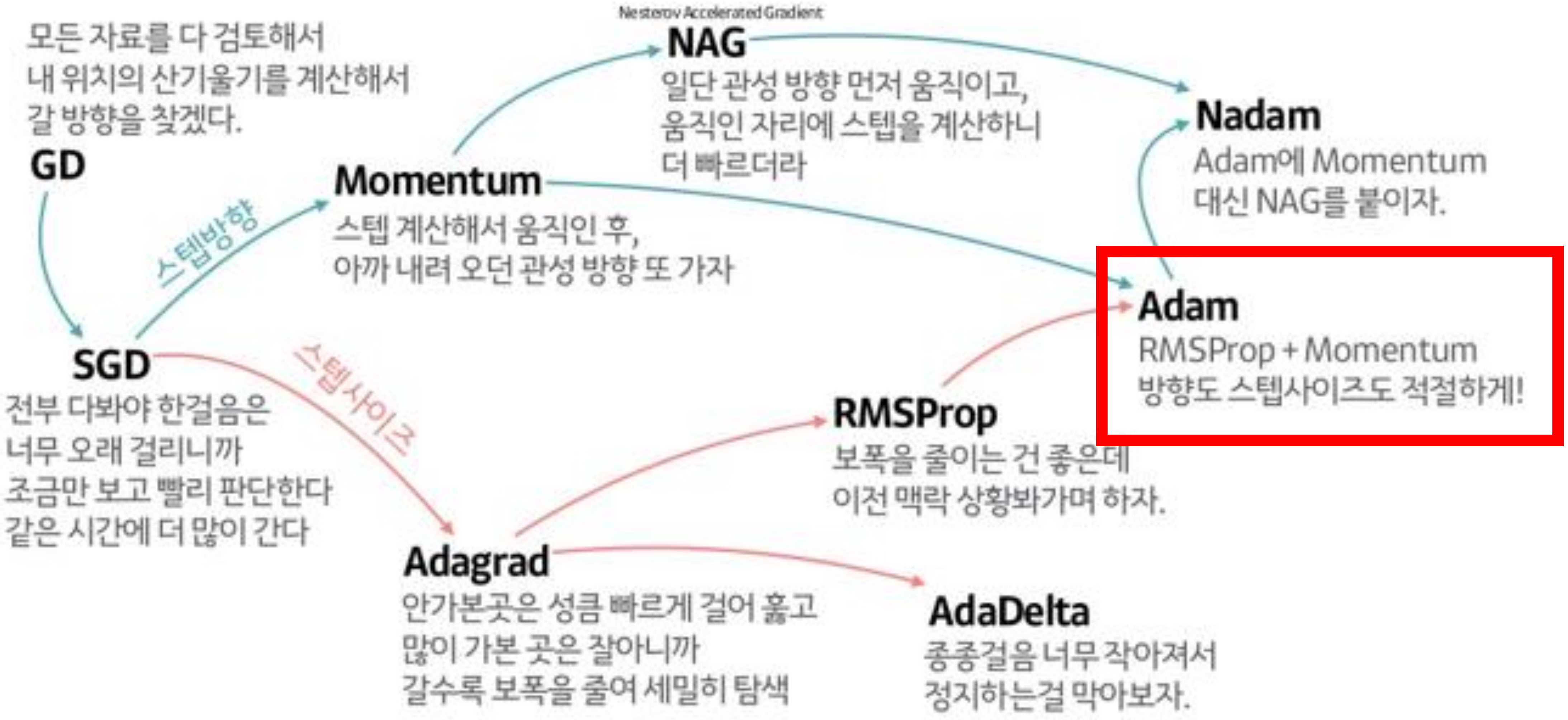




위의 제곱은 element-wise product

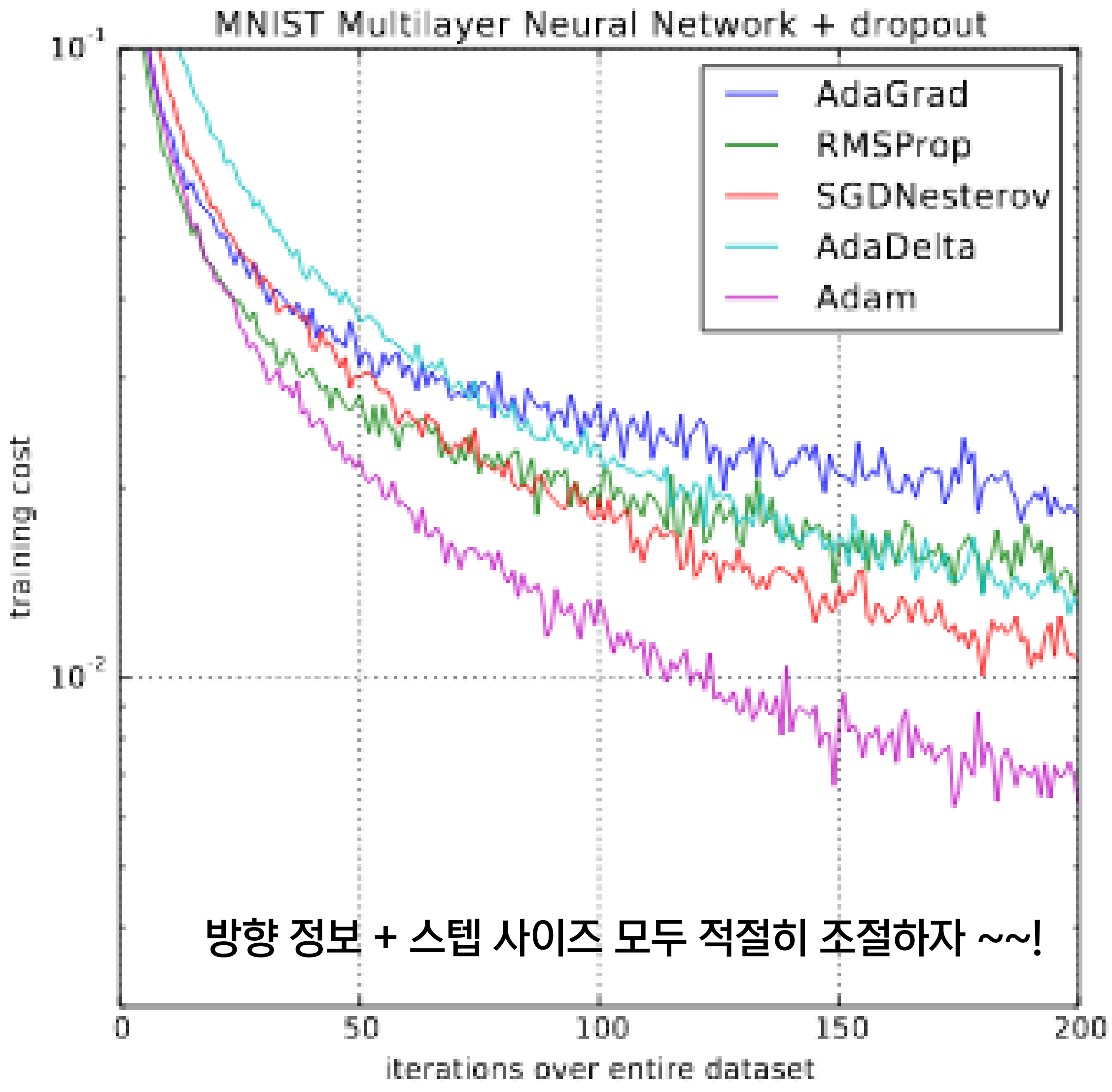
Beta는 0.999를 주로 사용

3. Optimizer


모든 자료를 다 검토해서
내 위치의 산기울기를 계산해서
갈 방향을 찾겠다.
GD
스텝방향
스텝사이즈
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
Nadam
Adam에 Momentum
대신 NAG를 붙이자.
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.

3. Optimizer

- Adam(ADAptive Momentum estimation) : momentum + RMSProp
  - $$VdW = \beta_1 VdW + (1-\beta_1)dW$$
  - $$Vdb = \beta_1 Vdb + (1-\beta_1)db$$
  - $$SdW = \beta_2 SdW + (1-\beta_2)dW^2$$
  - $$Sdb = \beta_2 Sdb + (1-\beta_2)db^2$$
  - --- bias correction---
  - $$VdW = \frac{VdW}{1-beta_1^t}$$
  - $$Vdb = \frac{Vdb}{1-beta_1^t}$$
  - $$SdW = \frac{SdW}{1-beta_2^t}$$
  - $$Sdb = \frac{Sdb}{1-beta_2^t}$$
  - -- parameter update --
  - $$W := W - \alpha\frac{VdW}{\sqrt{SdW}+\epsilon}$$
  - $$b := b - \alpha\frac{Vdb}{\sqrt{Sdb}+\epsilon}$$

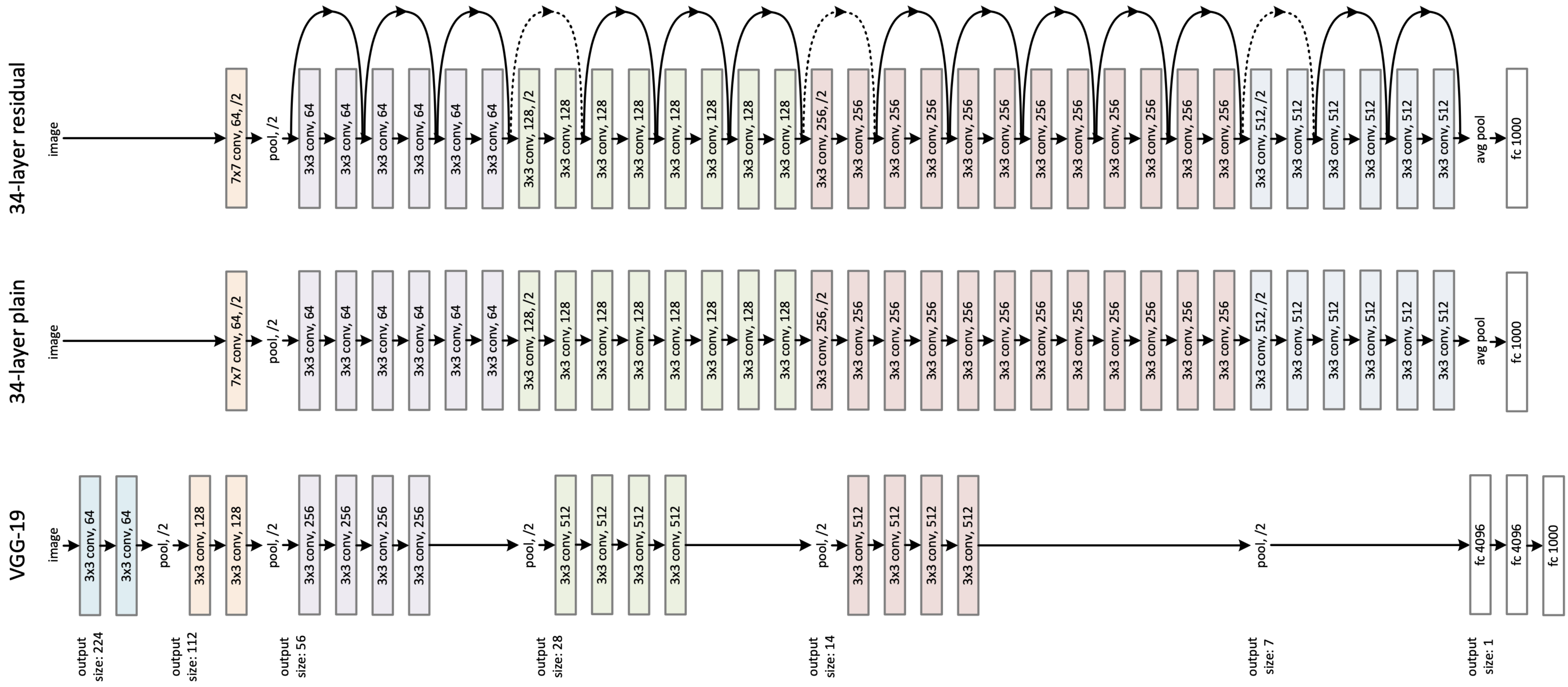


CONTENTS

04. Batch Normalization

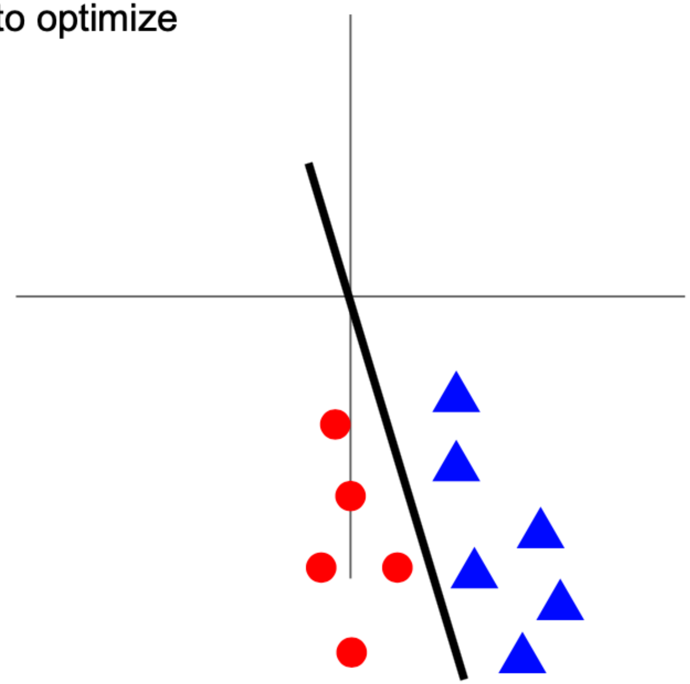
- Batch norm

4. Batch Normalization

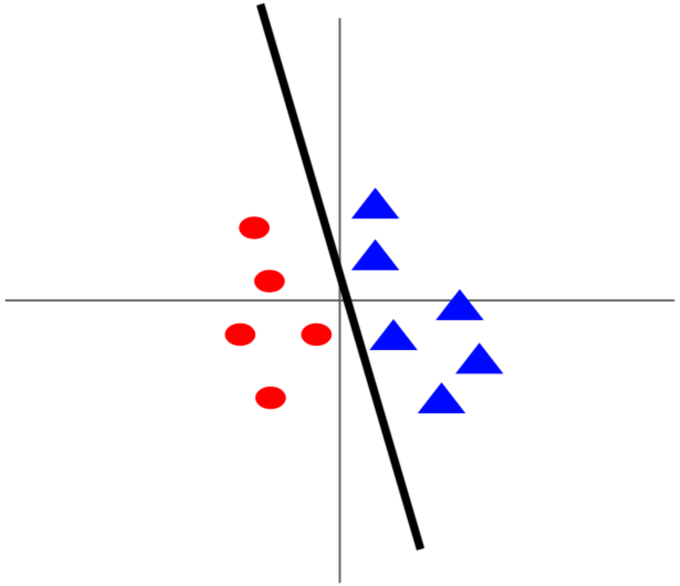


4. Batch Normalization

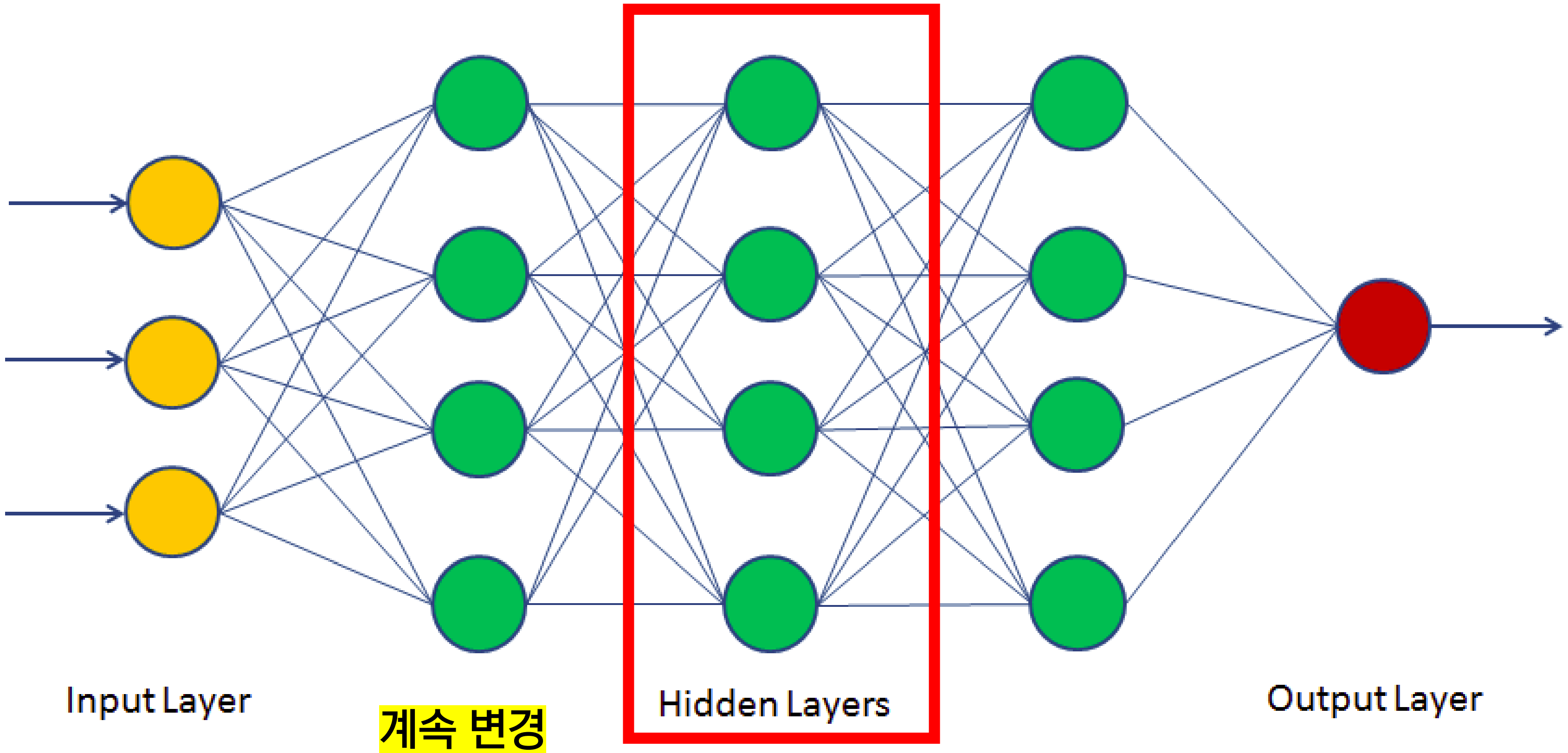
Before normalization: classification loss very sensitive to changes in weight matrix; hard to optimize



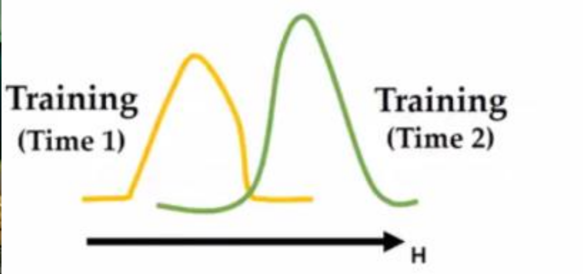
After normalization: less sensitive to small changes in weights; easier to optimize



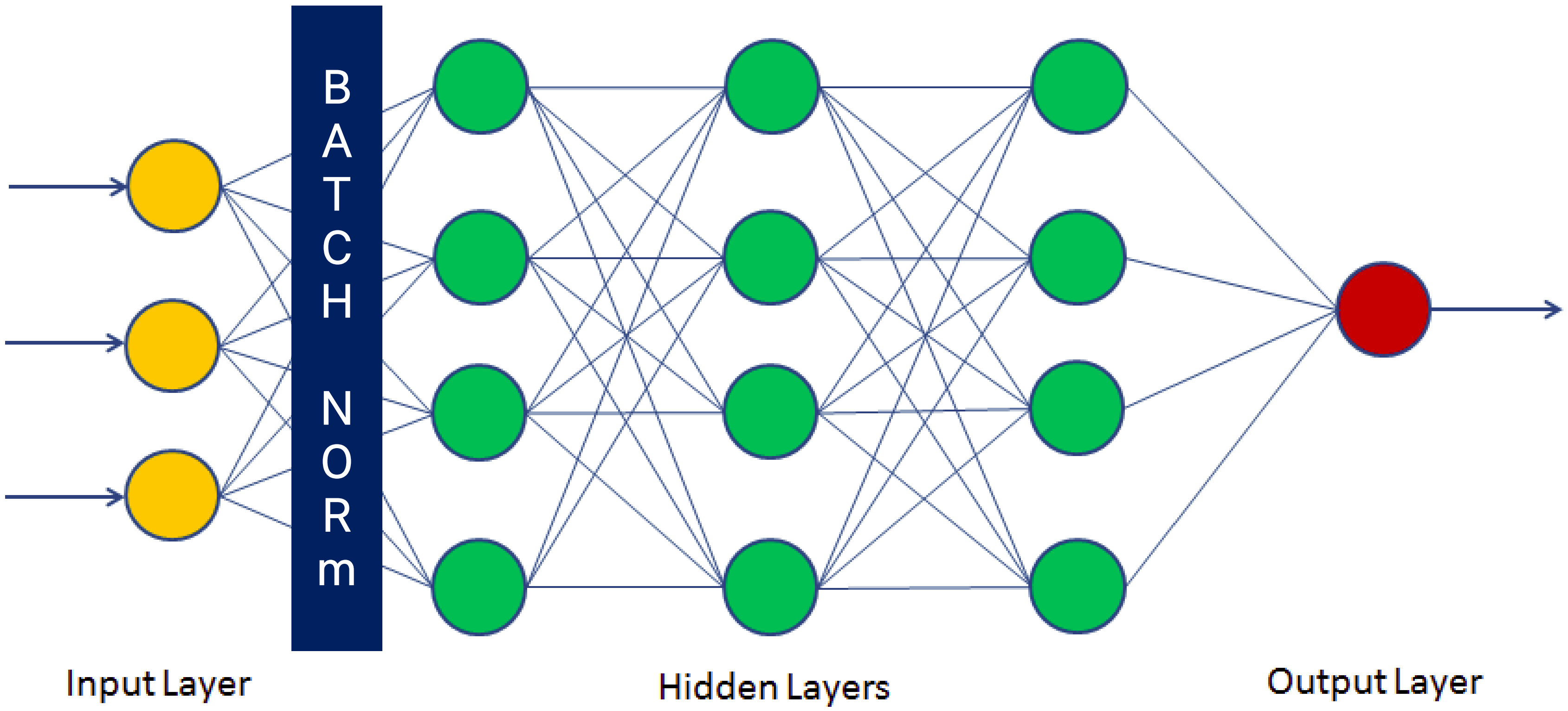
4. Batch Normalization



계속 변경



4. Batch Normalization



* Batch norm은 주로 FC – batch norm – activation에 위치

4. Batch Normalization

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1 \ldots m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

여기서 gamma와 beta는 모델에 의해서 학습되는 Parameter이다. (not hyper-parameter)

과제

☆ 과제는 8주차, 9주차, 10주차 내용 중 어려웠던 내용 공부하기 ☆

최소 분량은 따로 정해두지 않겠습니다.

자유롭게 어려웠던 내용에 대해 정리해주세요!

하나의 주제 (ex. Optimizer)를 고르셔서 정리하셔도 되고,

아니면 전체 흐름을 키워드 중심으로 정리하는 것도 좋습니다!

🎉쿠글 완주 성공🎉

감사합니다