딥러닝 기초 개념 소개

KUBIG 12기 이나윤

INDEX

2021년 여름방학 딥러닝 분반 2주차

01

딥러닝 전체 구조
및 학습 과정
+
XOR, 인공신경망

02

- Activation 함수
- Weight Initialization

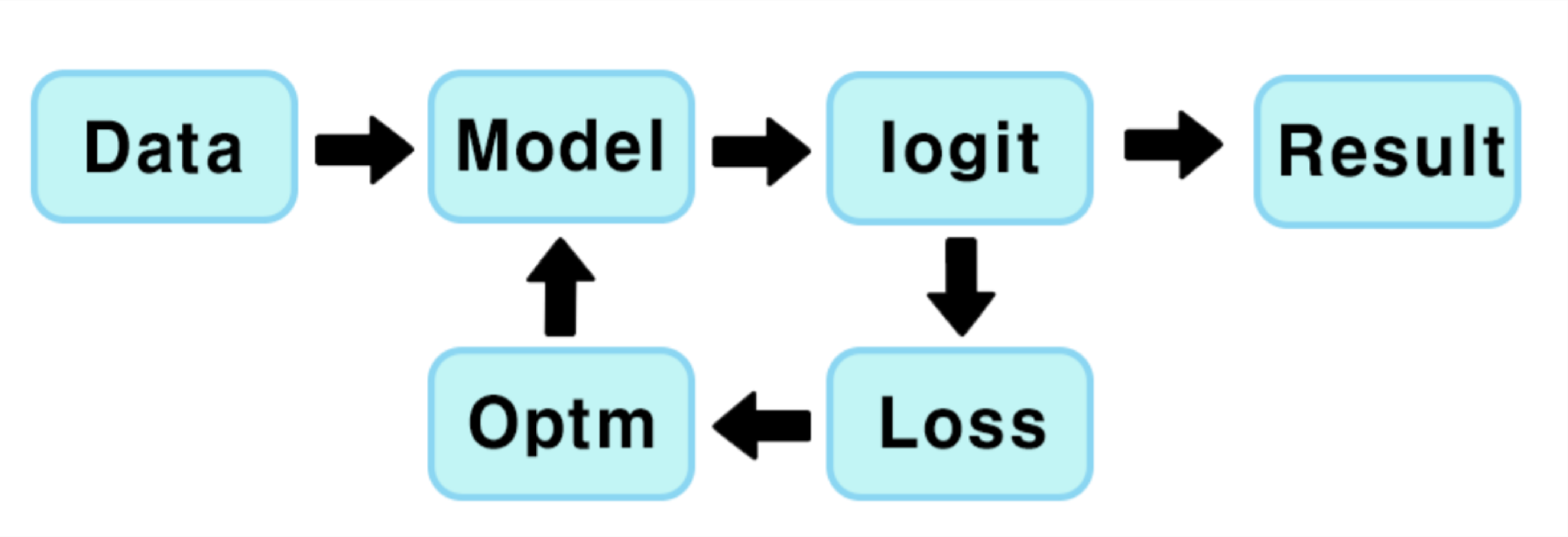
03

Overfitting
: Dropout,
Regularization
etc

04

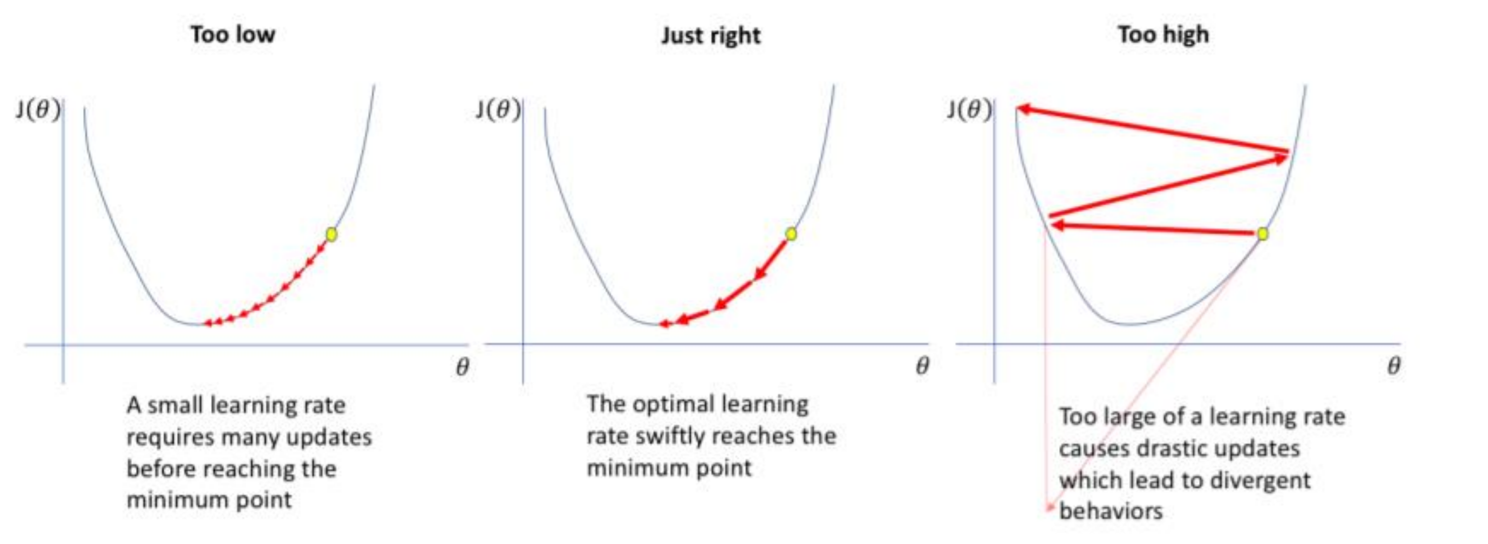
Optimizer

01. 딥러닝 전체 구조 및 학습 과정

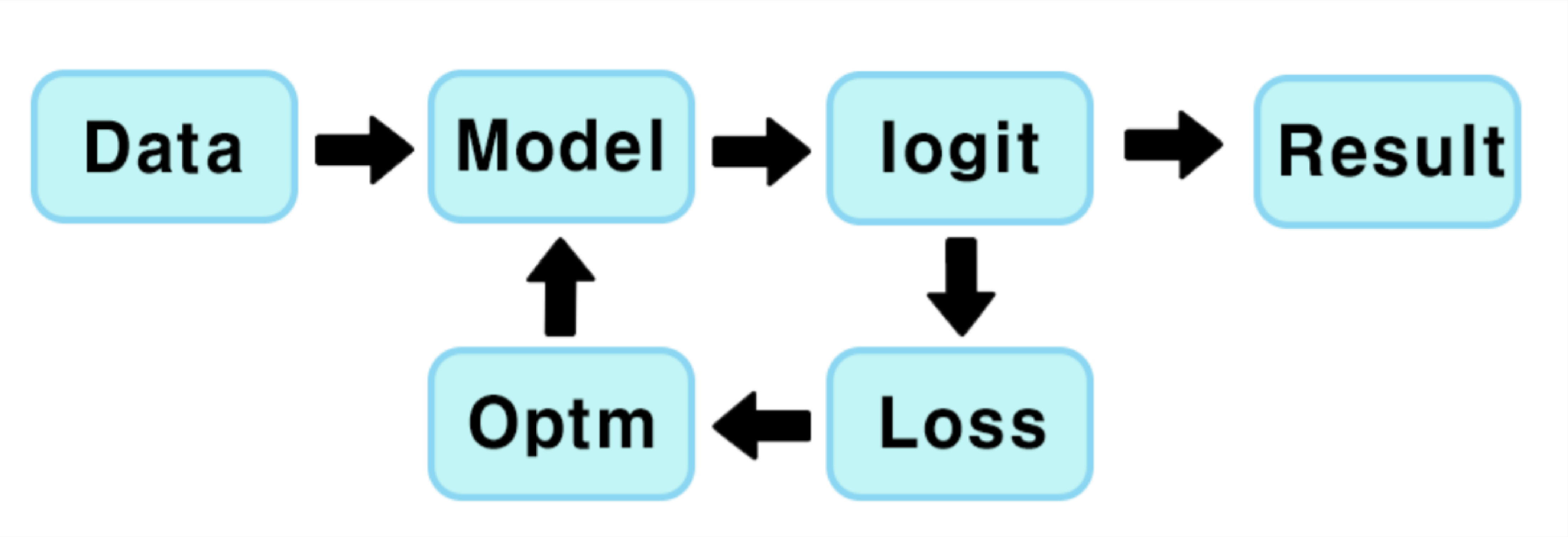


Batch size: 전체 training dataset을 여러 작은 그룹으로 나누었을 때, Batch size는 하나의 소그룹에 속하는 데이터 수

Learning rate: 기울기에 학습률 곱해 다음 지점 결정



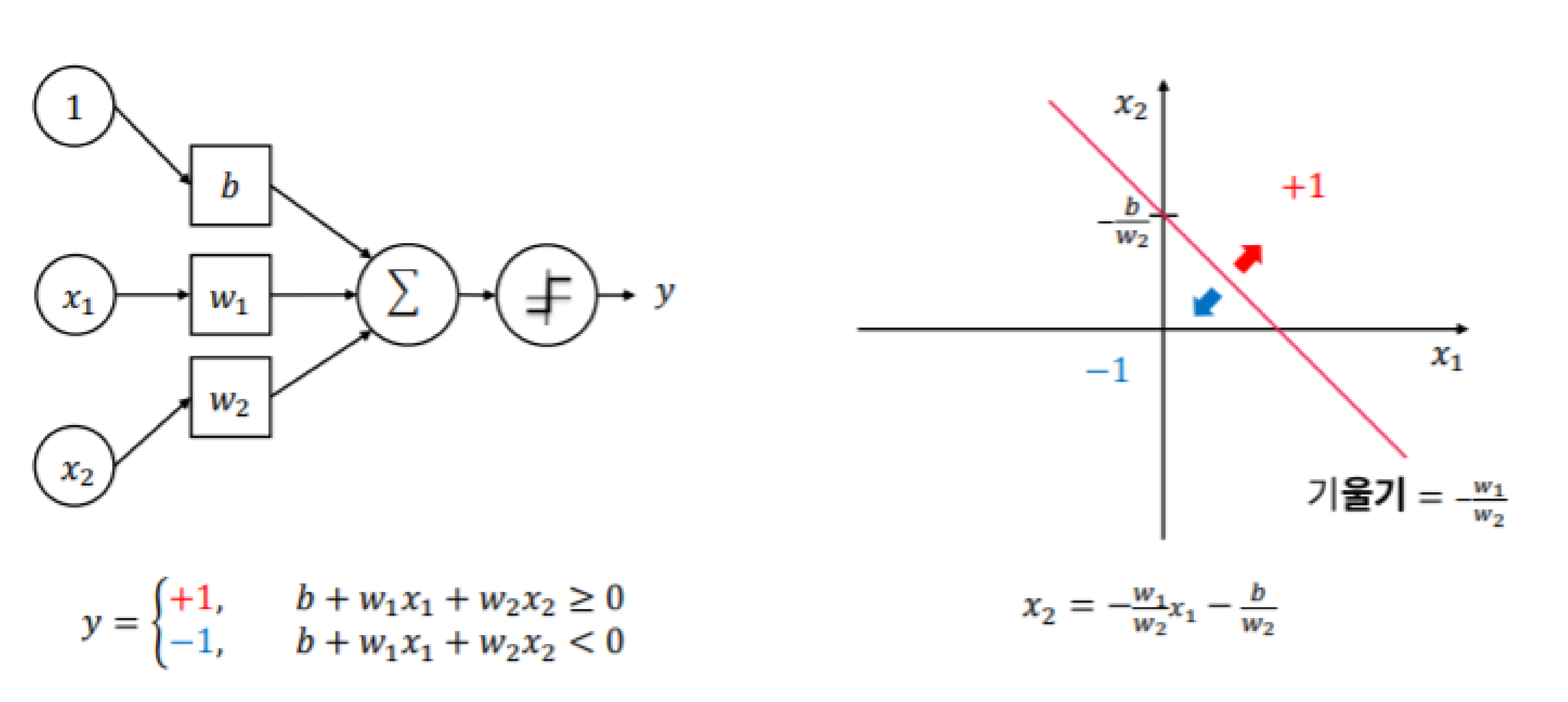
01. 딥러닝 전체 구조 및 학습 과정

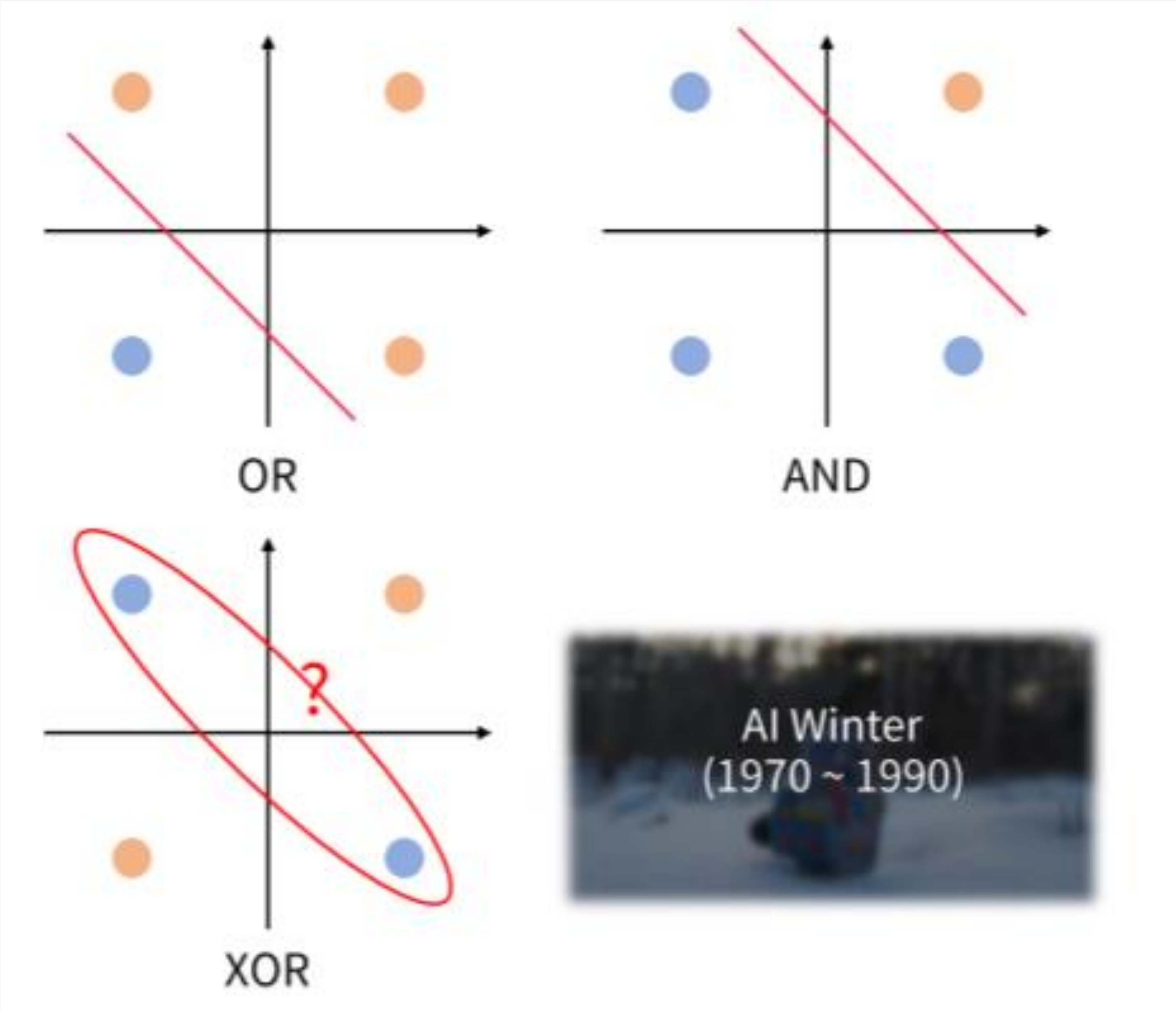


Test set: 학습에 전혀 관여하지 않고, 오직 '최종 성능' 을 평가하기 위해 사용.

Validation set: 학습을 시키지 않지만, 학습에 '관여'함.

01. XOR 문제 + 퍼셉트론

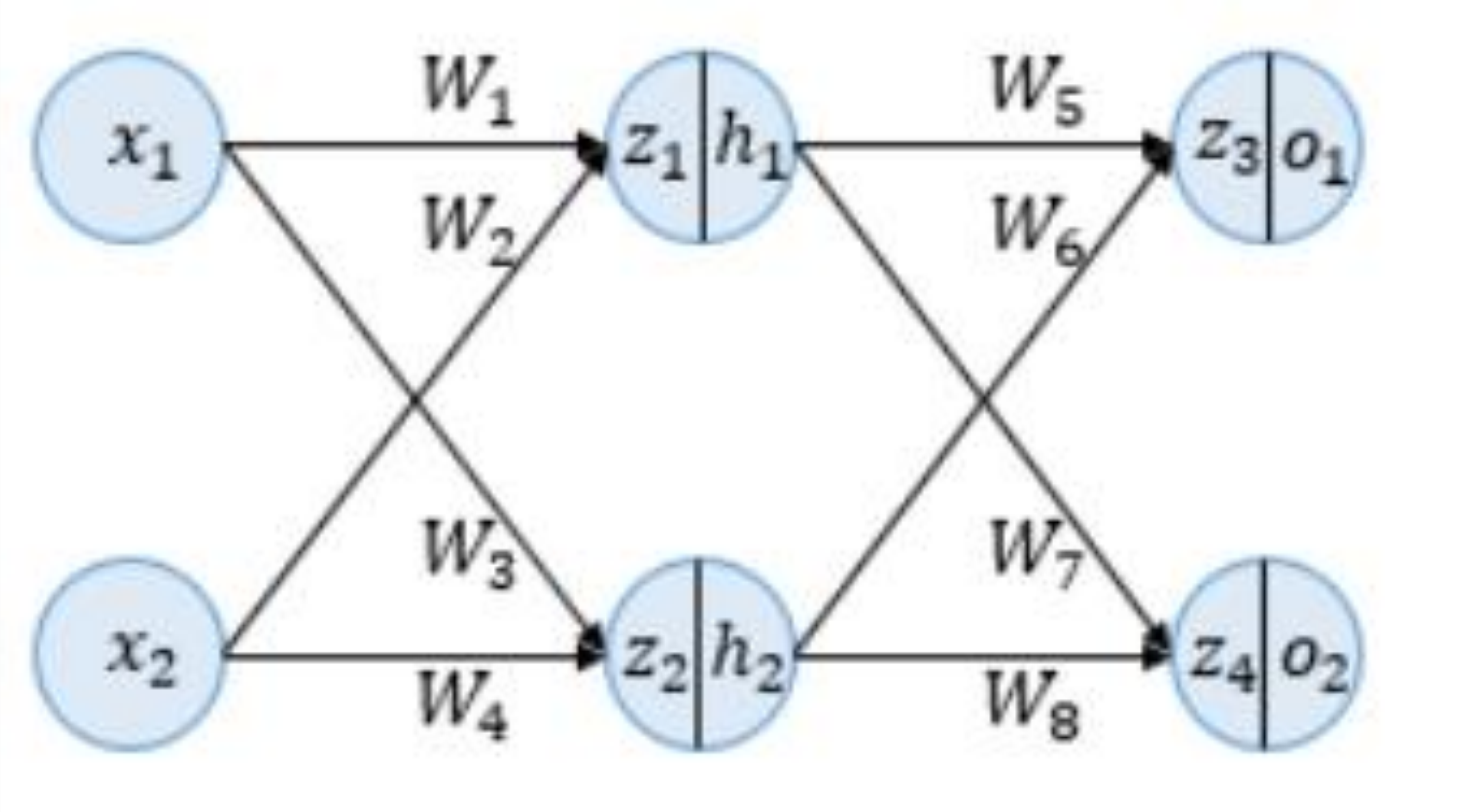




01. 인공신경망

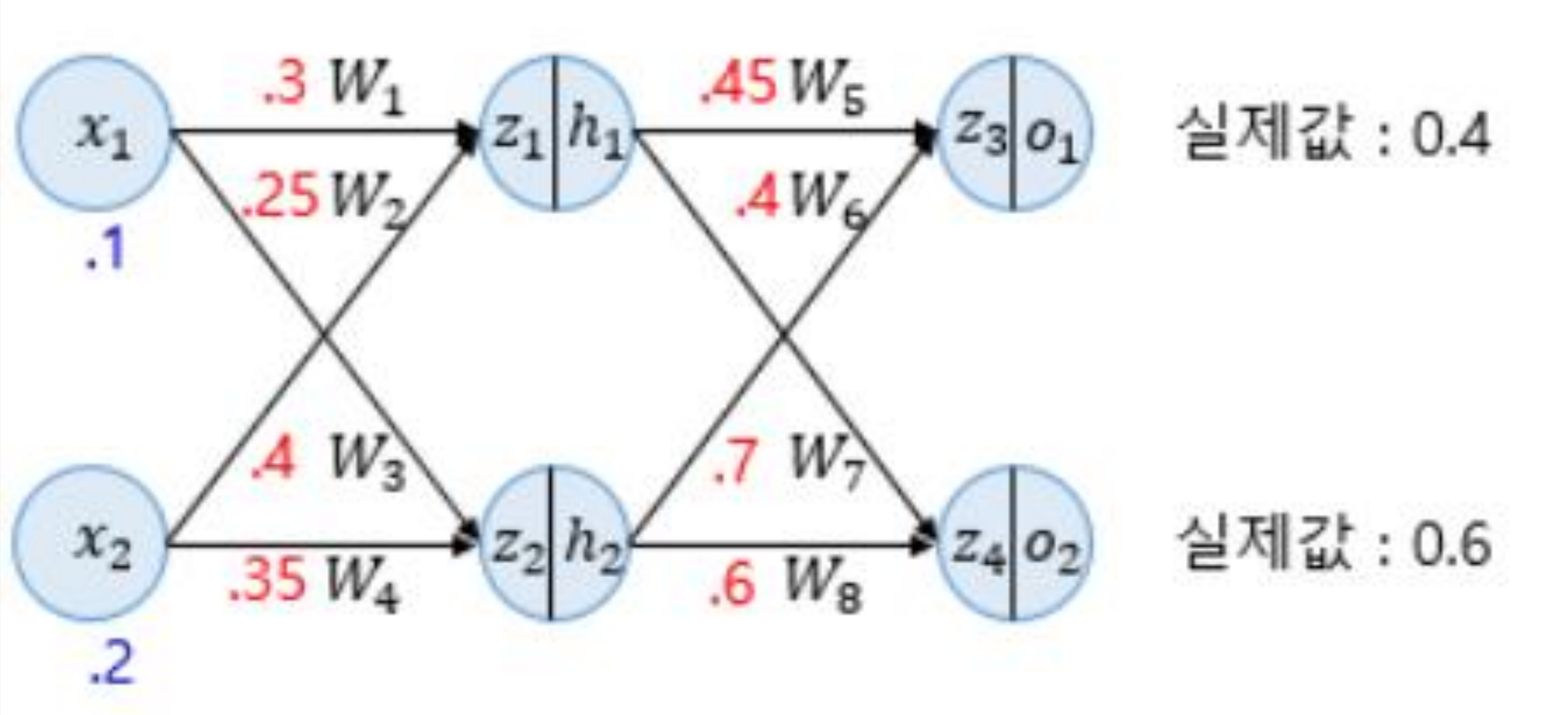
인공신경망: 뉴런이 모여 서로 연결된 형태

신경망의 기능: 출력 계층의 activation function에 의해 결정.



- Identity function : 회귀
- Sigmoid function: 이진 분류
- Softmax function: 다중 분류

01. 인공신경망 (Forward Propagation)



$$E_{o1} = \frac{1}{2}(target_{o1} - output_{o1})^2 = 0.02193381$$

$$E_{o2} = \frac{1}{2}(target_{o2} - output_{o2})^2 = 0.00203809$$

$$E_{total} = E_{o1} + E_{o2} = 0.02397190$$

$$z_1 = W_1x_1 + W_2x_2 = 0.3\times0.1 + 0.25\times0.2 = 0.08$$

$$z_2 = W_3x_1 + W_4x_2 = 0.4\times0.1 + 0.35\times0.2 = 0.11$$

$$z_3 = W_5h_1 + W_6h_2 = 0.45\times h_1 + 0.4\times h_2 = 0.44498412$$

$$z_4 = W_7h_1 + W_8h_2 = 0.7\times h_1 + 0.6\times h_2 = 0.68047592$$

$$h_1 = sigmoid(z_1) = 0.51998934$$

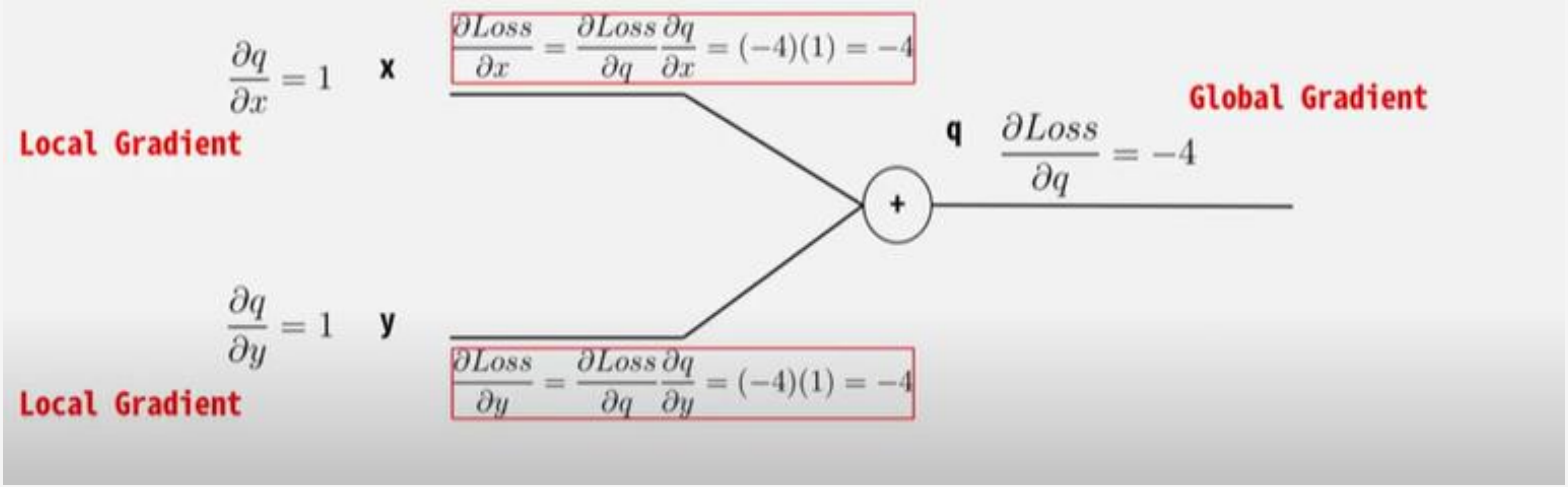
$$h_2 = sigmoid(z_2) = 0.52747230$$

$$o_1 = sigmoid(z_3) = 0.60944600$$

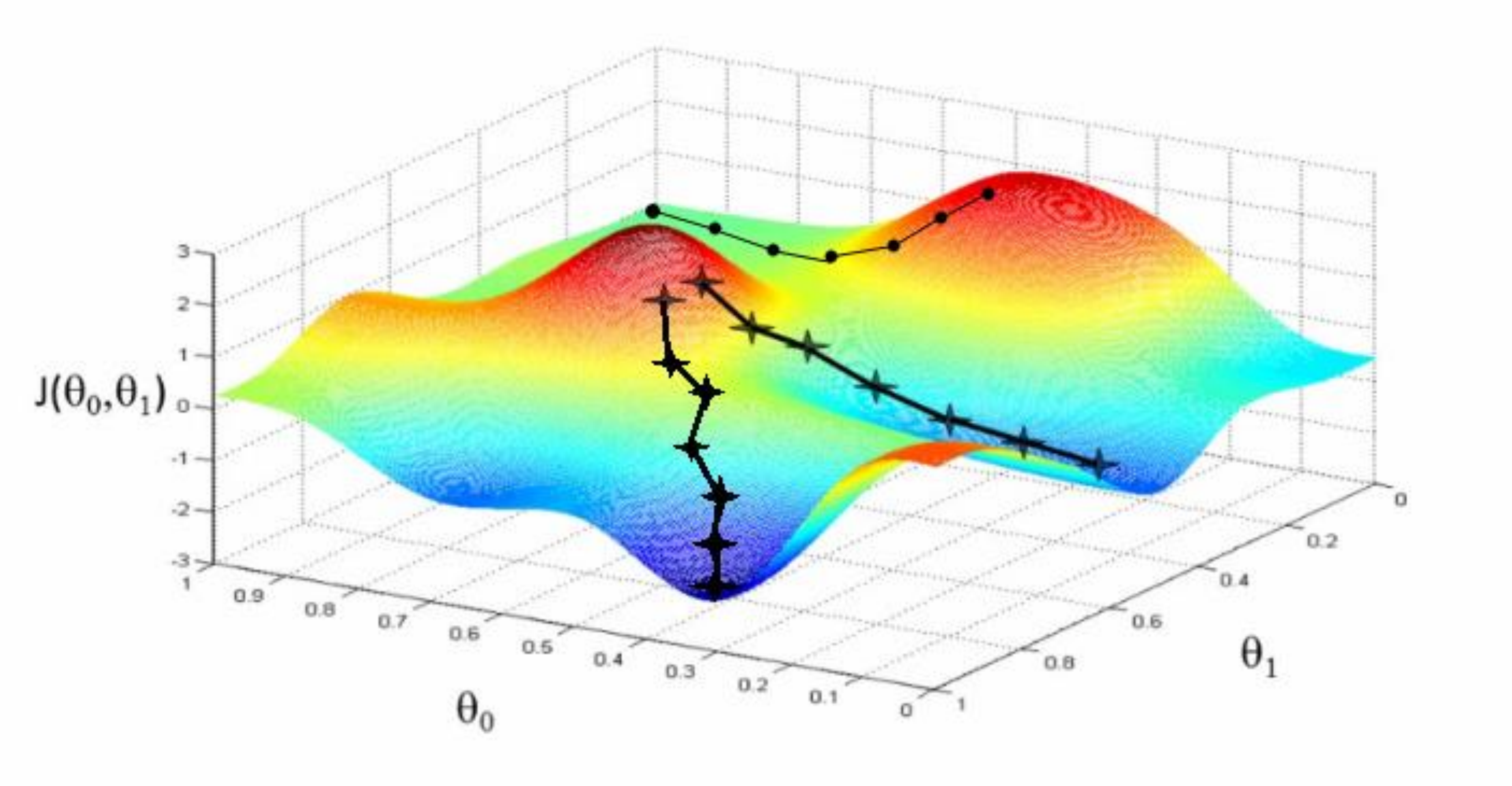
$$o_2 = sigmoid(z_4) = 0.66384491$$

01. 인공신경망 (BackPropagation)

-> Chain rule을 활용하여 Local Gradient * Global Gradient를 미분 값을 계산.



02. Weight Initialization



-> 첫 위치가 어디가 좋을까?

: Weight Initialization

02. Weight Initialization

- 상수 기반 초기화 (Zeros, Ones, Constant)

: 정해진 숫자를 기반으로 weight 값 초기화

: 모든 weight들이 동일한 값으로 초기화되어 모든 weight들이 동일한 출력값을 냄.

- 선형 대수 기반 초기화 (Orthogonal, Identity)

Orthogonal: 직교 행렬이 되게끔, weight 값 초기화

Identity: 항등행렬로 weight 값 초기화

- 확률분포 기반 초기화 (RandomUniform, RandomNormal, TruncatedNormal)

: 특정한 확률분포에 기반하여 랜덤한 값을 추출하여 weight 값 초기화

02. Weight Initialization

- 분산 조정 기반 초기화 (Xavier, Lecun, He)

: 실제로 딥러닝 모델들에서 가장 많이 사용하는 방법들

: 확률분포를 기반으로 추출한 값으로 weigth를 초기화 하되, 확률 분포의 분산을 weight별로 동적 조절

: 분산 조정 시, 각 weight에 input으로 들어오는 텐서의 차원(fan in)과

결과 값으로 출력하는 텐서의 차원(fan out) 사용

02. Weight Initialization

- Lecun weight Initialization (keras lecun_uniform/ lecun_normal)

: 초기 CNN인 lenet으로 유명한 Yann Lecun 교수님이 제안한 기법

: 기본적으로 uniform/ normal 분포에서 추출한 random 값으로 weight를 초기화하되,
이 확률분포를 fan in 으로 조절하자는 idea

$$lecun\ unifom:\ unif(-limit,\ +limit),\ limit=\sqrt{\frac{3}{fan\ in}}$$

$$lecun\ normal:\ normal(mean=0,\ stddev),\ stddev=\sqrt{\frac{1}{fan\ in}}$$

-> 신경망에 들어오는 input 크기인 fan in이 커질 수록 초기화 값의 분산을 작게 만들자

02. Weight Initialization

- Xavier weight Initialization (keras glorot_uniform/glorot_normal)

: Xavier Glorot 교수님이 2010년 제안한 기법

: 기본적으로 uniform/ normal 분포에서 추출한 random 값으로 weight를 초기화하되,
이 확률분포를 fan in과 fan out으로 조정하자는 idea

$$glorot\ unifom:\ unif(-limit,\ +limit),\ limit=\sqrt{\frac{6}{fan\ in+fan\ out}}$$

$$glorot\ normal:\ normal(mean=0,\ stddev),\ stddev=\sqrt{\frac{2}{fan\ in+fan\ out}}$$

-> fan in, fan out을 모두 고려하여 확률 분포를 조정

-> sigmoid, tanh activation function을 사용할 때, 많이 사용함

-> Relu 함수에서 사용시 출력 값이 0으로 수렴

02. Weight Initialization

- He weight Initialization (keras he_uniform/ he_normal)

: Xaiver의 한계를 극복하기 위해 kaming he가 제안

: 기본적으로 uniform/ normal 분포에서 추출한 random 값으로 weight를 초기화하되,
이 확률분포를 fan in 으로 조절하자는 idea

$$he\ unifom:\ unif(-limit,\ +limit),\ limit = \sqrt{\frac{6}{fan\ in}}$$

$$he\ normal:\ normal(mean = 0,\ stddev),\ stddev = \sqrt{\frac{2}{fan\ in}}$$

-> Relu activation function을 사용할 때, 많이 사용함

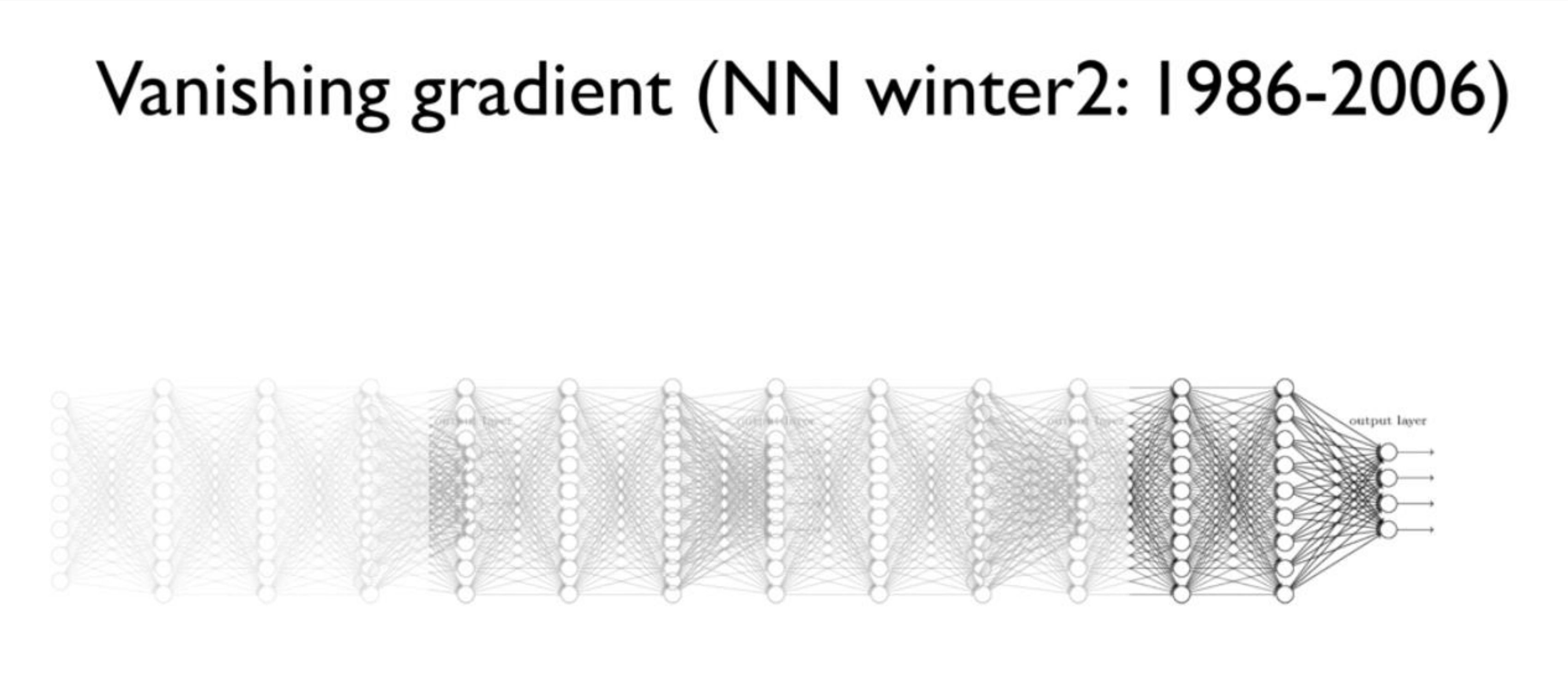
-> 깊은 CNN 신경망에서 잘 작동함 (최근 대부분 사용)

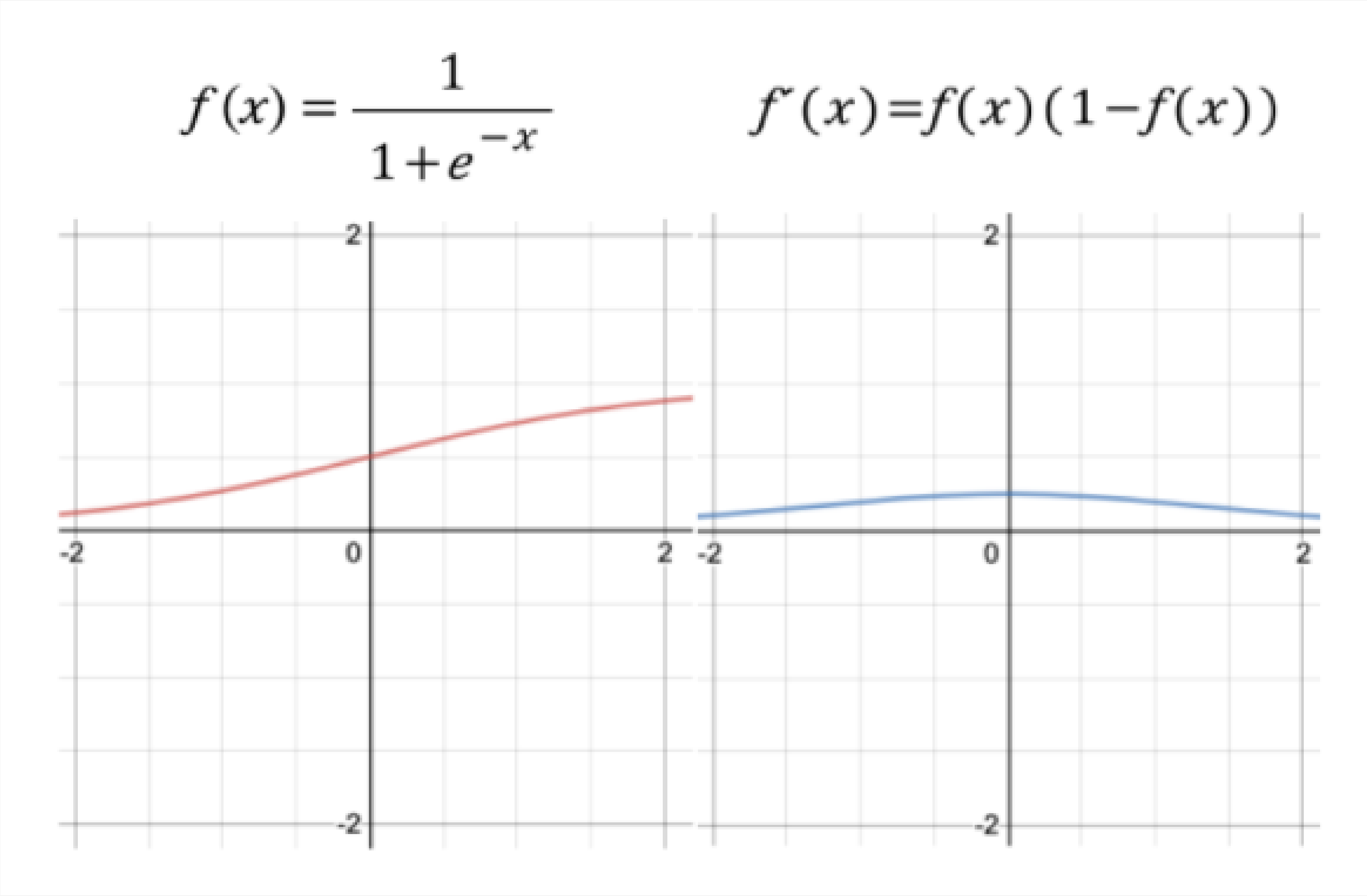
https://kjhov195.github.io/2020-01-07-weight_initialization/

02. Activation function

: 출력 값이 linear하지 않아, 선형분류기를 비선형 시스템으로 만들 수 있음. -> XOR 문제 해결

- Sigmoid





02. Activation function

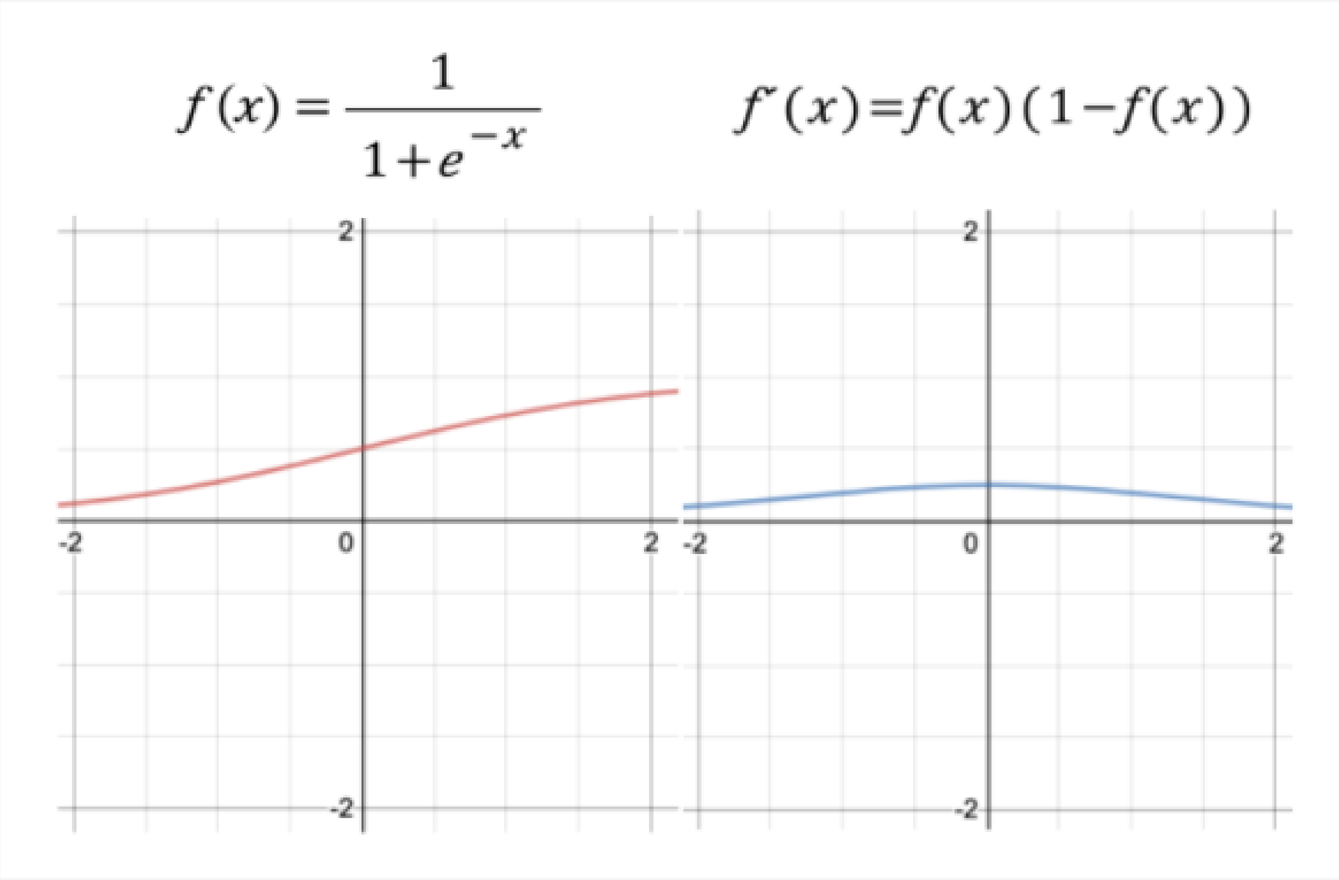
- Sigmoid

: 함수의 출력값을 [0,1]로 제한시킴.

: logistic regression, binary classification 등에 사용 O

: vanishing gradient 문제 발생할 가능성 O

(input 값이 너무 크거나 작아지면 기울기가 거의 0)

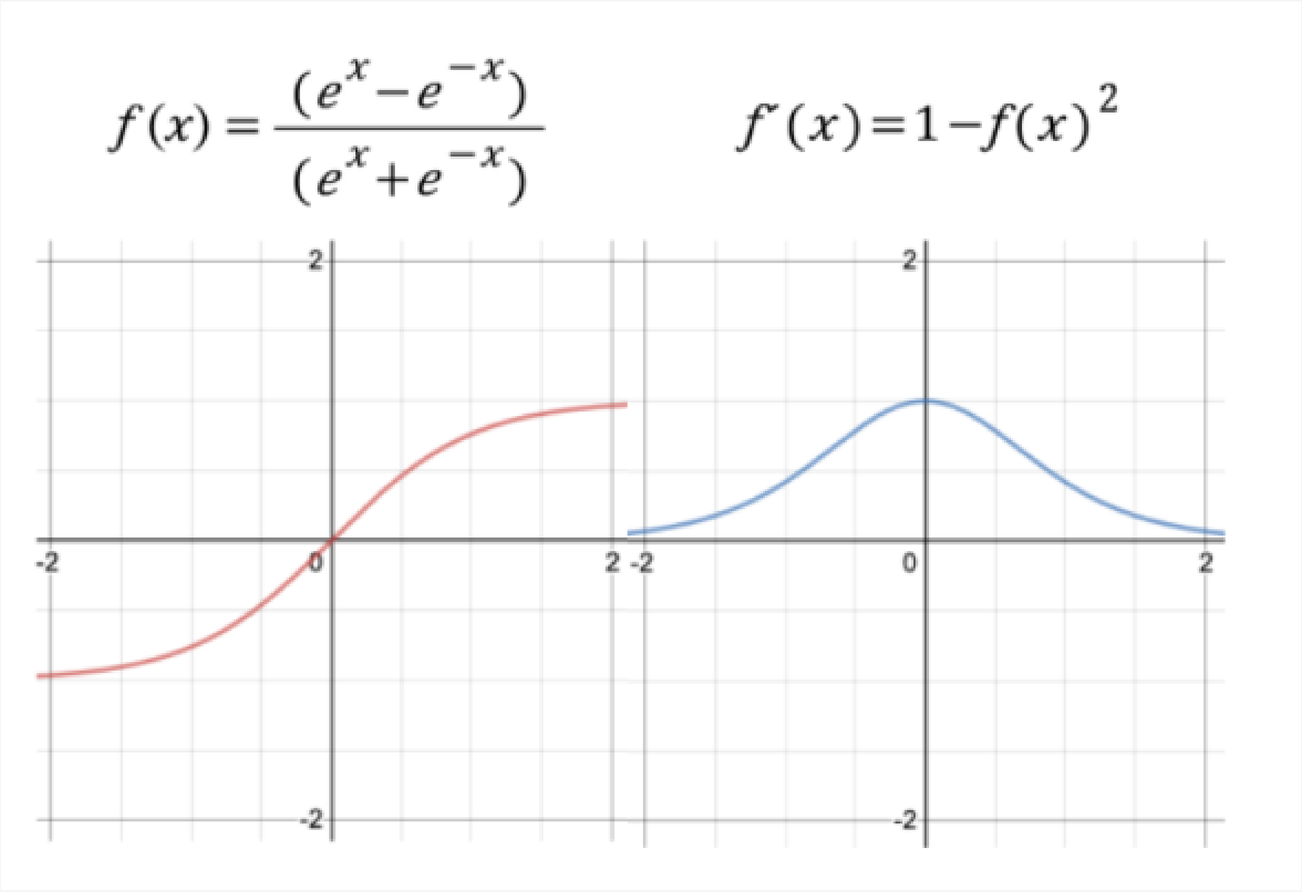


- tanh

: 함수의 출력값을 [-1,1]로 제한시킴.

: logistic regression, binary classification 등에 사용

: zero- centered 모양



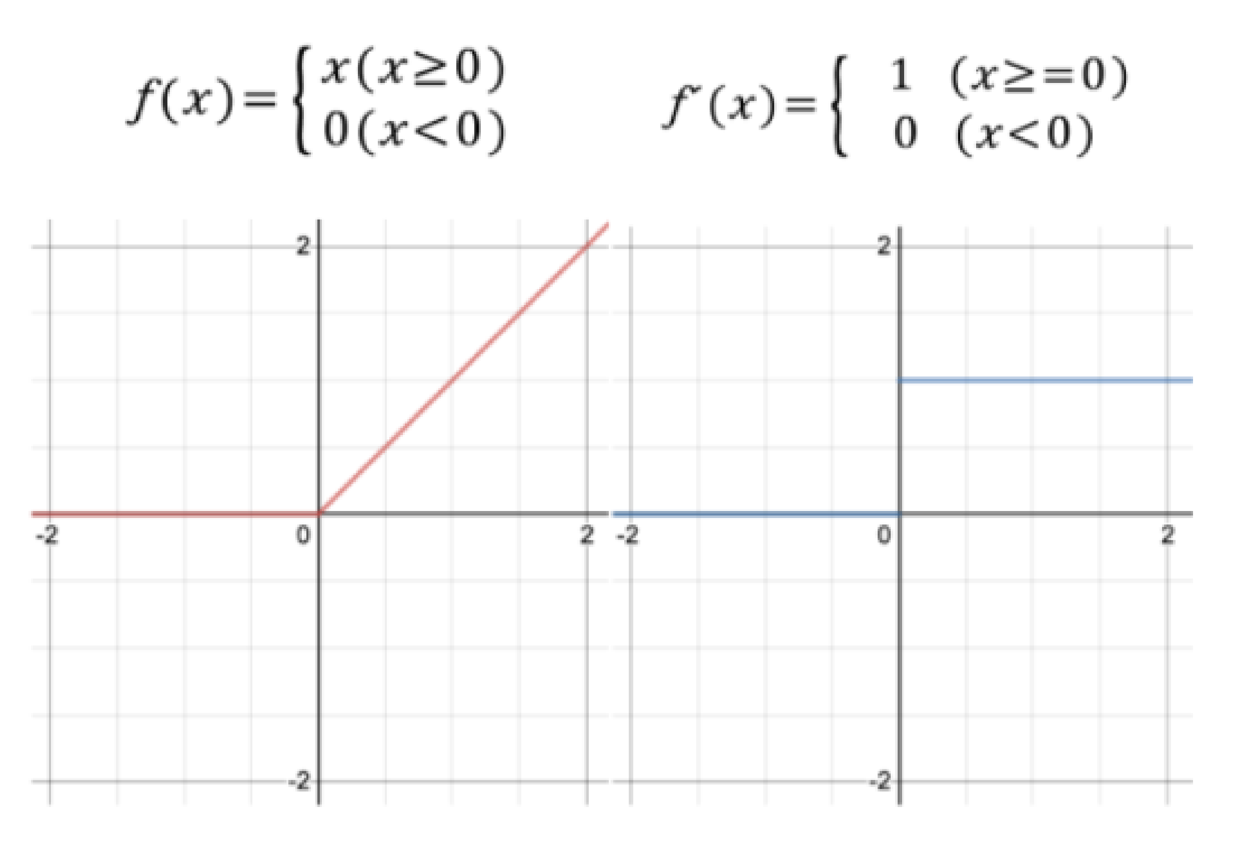
02. Activation function

- Relu

: 0 이하의 값은 다음 레이어에 전달 X, 0 이상의 값은 그대로 출력

: CNN 학습에 주로 사용

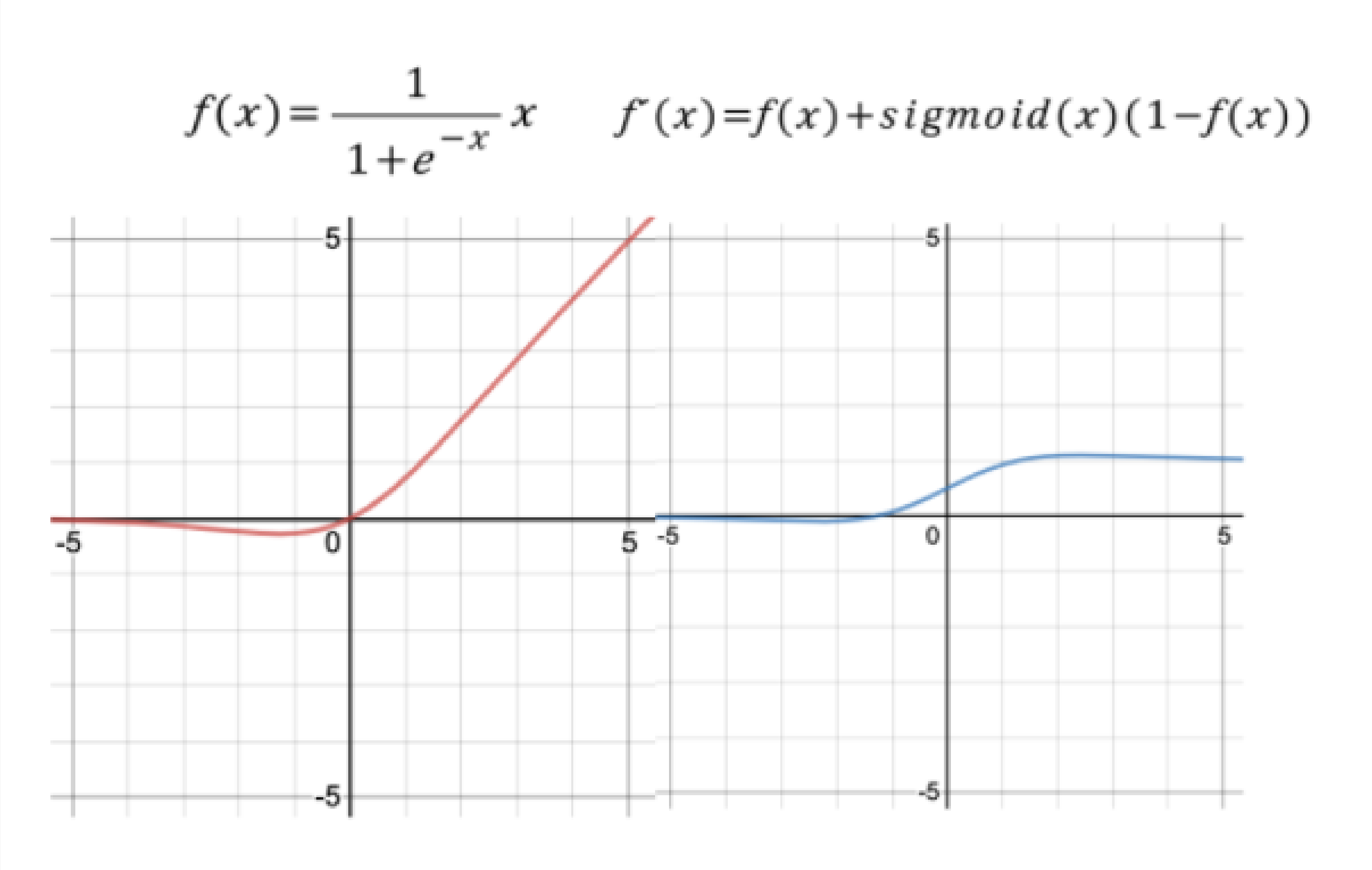
: 0 값을 다음 레이어에 전달하면 이후 출력값이 모두 0이 되는 dying Relu 현상 발생



- Swish

: Relu를 대체하기 위해 구글이 고안한 함수

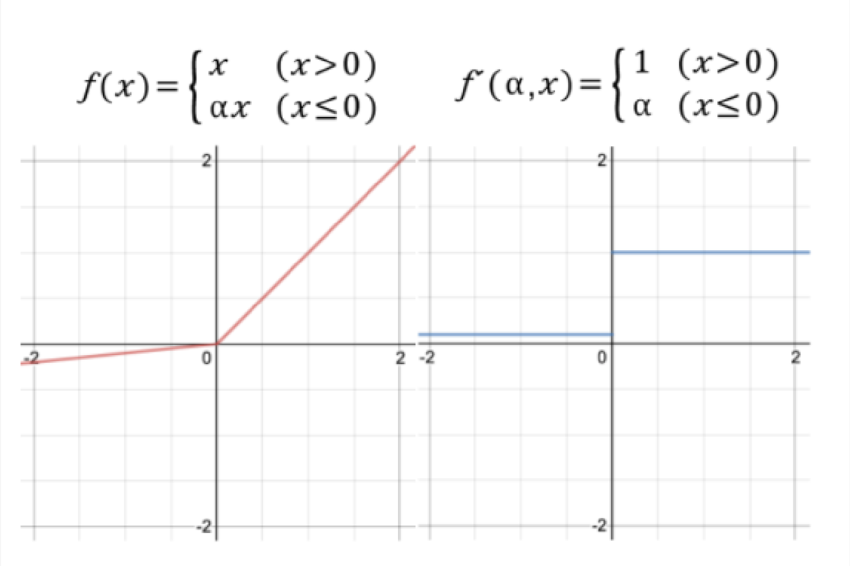
: 깊은 layer 학습시, ReLU보다 뛰어난 성능 보임



02. Activation function

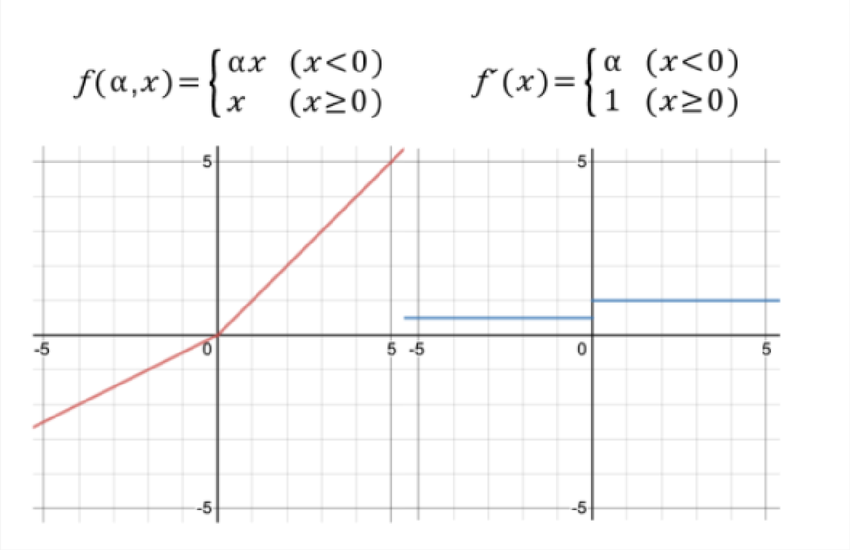
- LeakyRelu

: 입력값이 음수일때 완만한 선형 함수



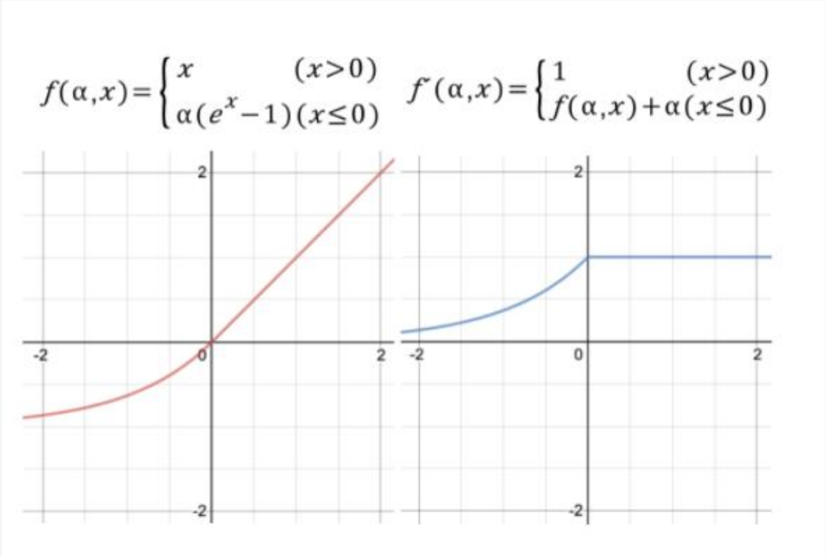
- PRelu

: LeakyRelu와 비슷하지만, 알파 값이 학습 가능한 parameter



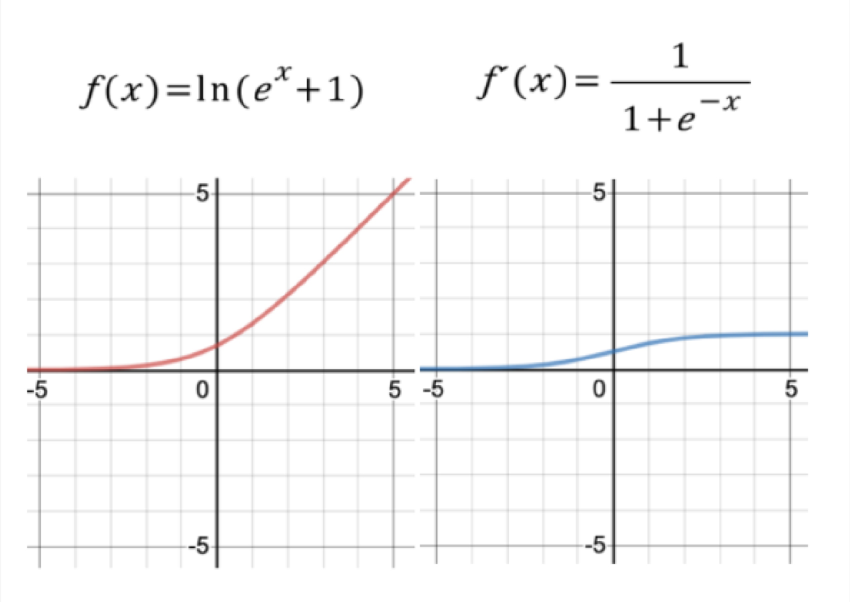
- ELU(exponential linear unit)

: 입력값이 음수일때 지수함수 사용

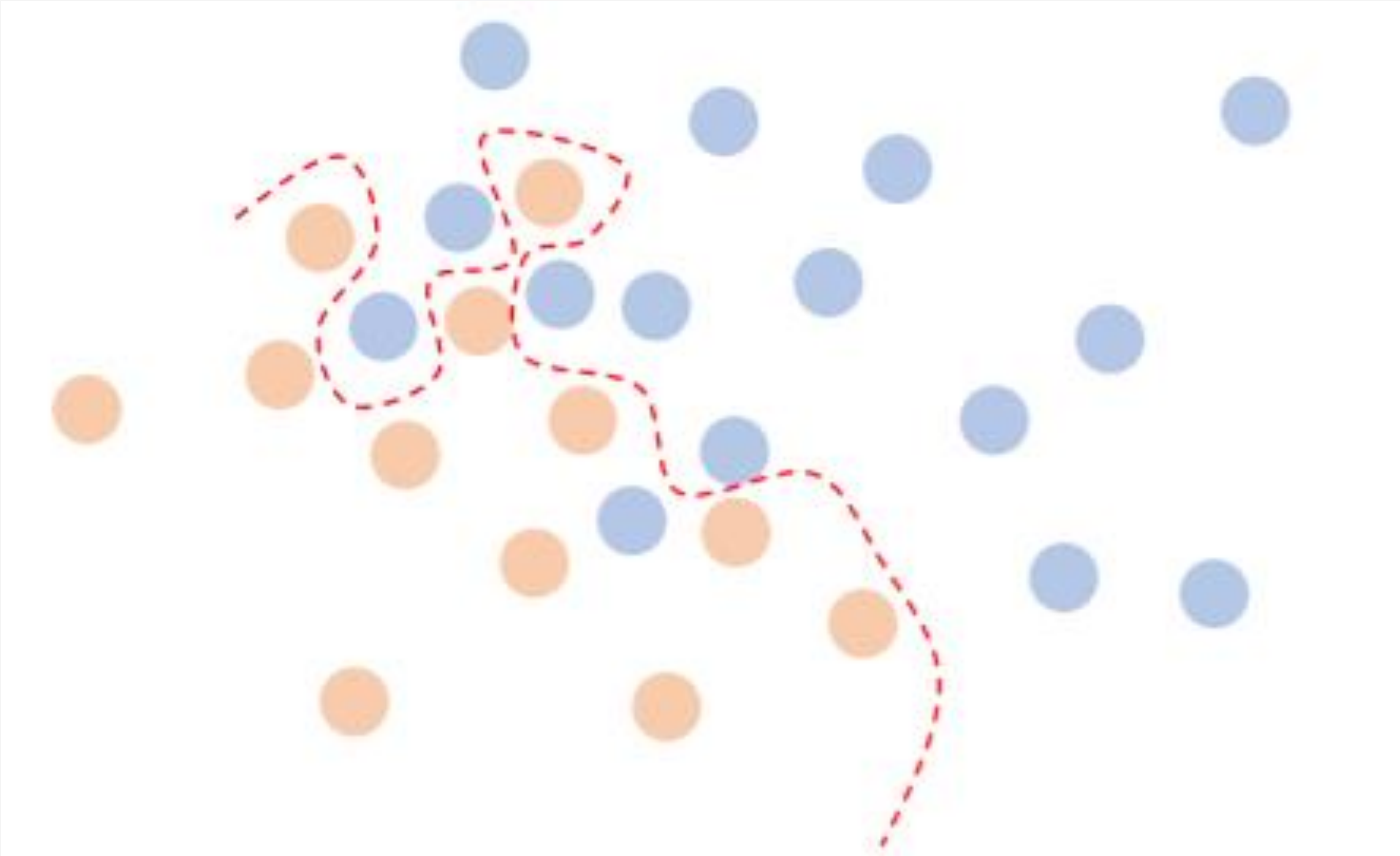


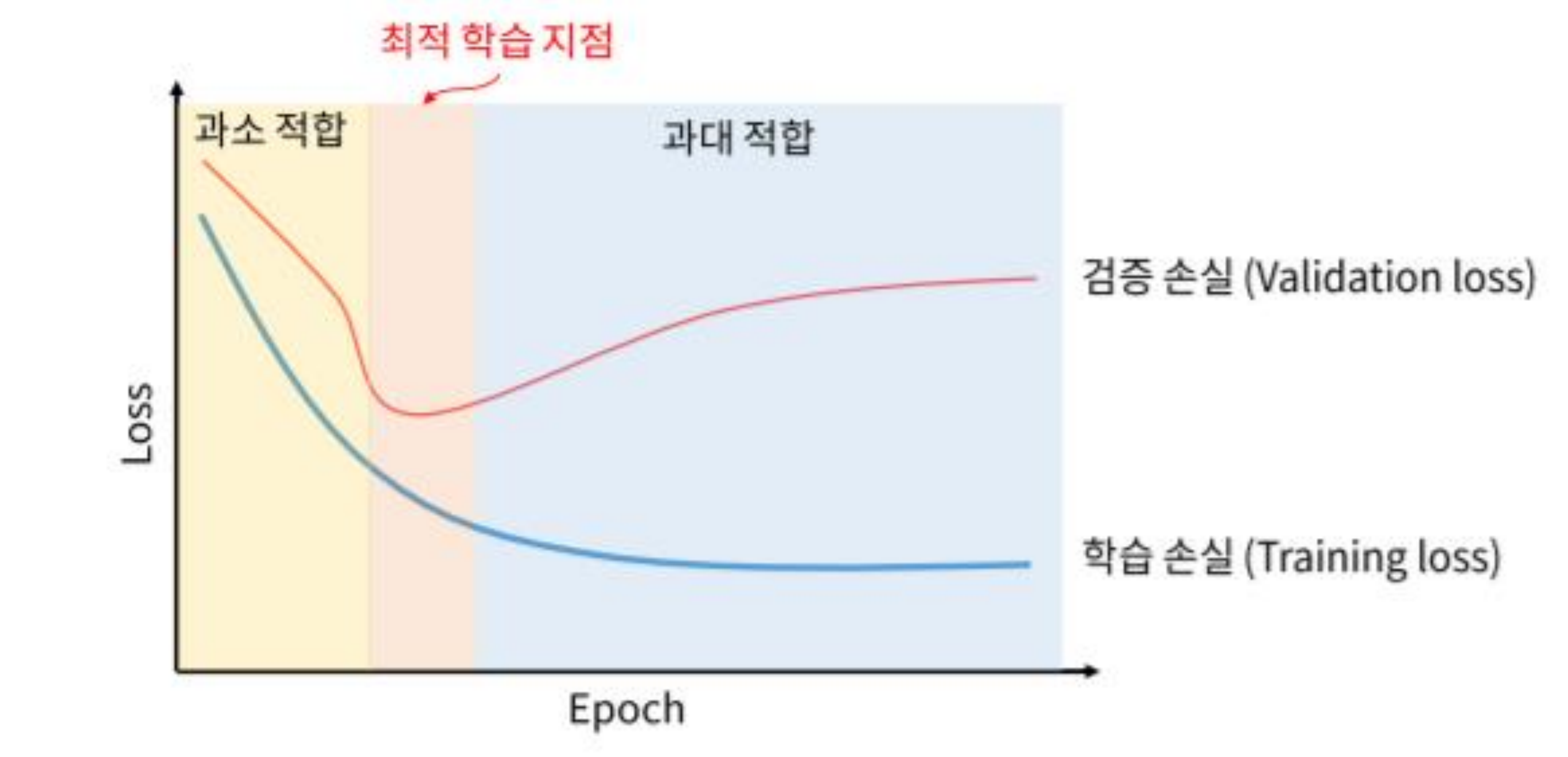
- Softplus

- : Relu를 부드럽게 깎아놓은 형태



03. Overfitting



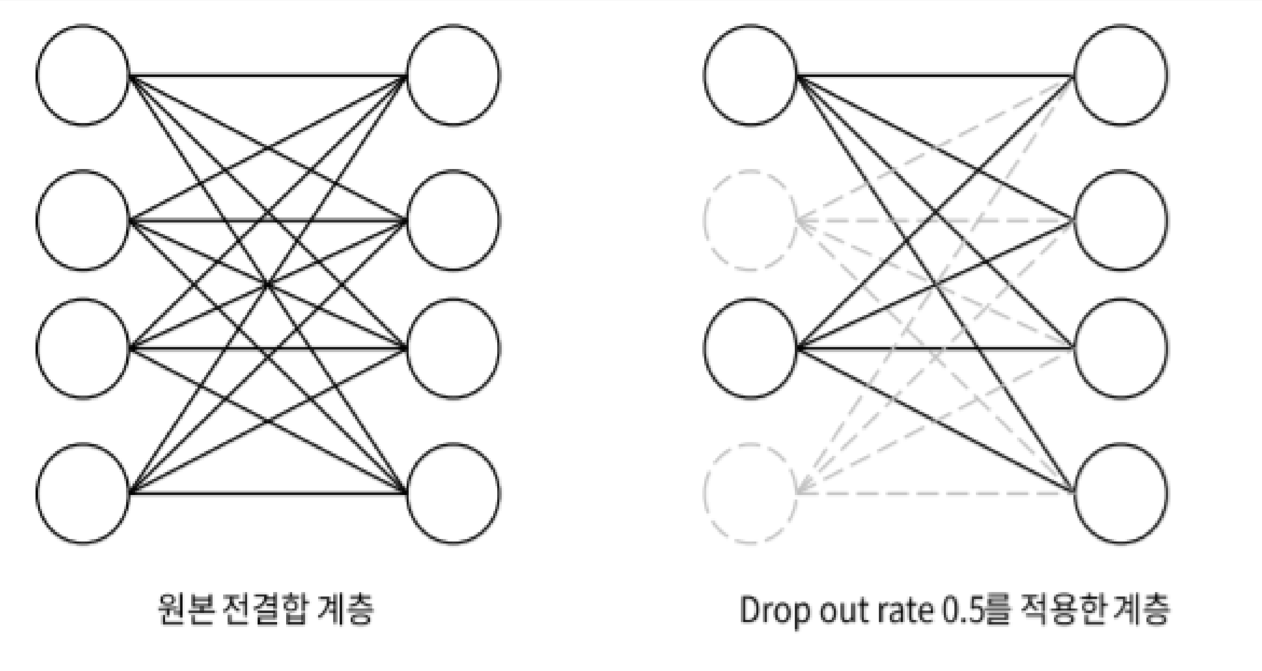


실제로 학습을 진행하다 보면, 학습 데이터에 과적합(Overfitting)되는 현상이 발생한다.

03. Overfitting

1. 데이터의 양을 늘리기 (Data Augmentation)

2. Dropout




```
model = Sequential()
model.add(Dense(256, input_shape=(max_words,), activation='relu'))
model.add(Dropout(0.5)) # 드롭아웃 추가. 비율은 50%
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5)) # 드롭아웃 추가. 비율은 50%
model.add(Dense(num_classes, activation='softmax'))
```

3. Early stopping

03. Overfitting

4. L1, L2 regularization

$$\text{Cost} = \text{Loss}(\text{Data}|\text{Model}) + \lambda\ \text{Complexity}(\text{Model})$$

- Loss: 학습 데이터에 대한 신뢰도가 높음. 학습 데이터에 속하지 않은 입력에 취약.

- Complexity: 모델의 복잡도 결정.

03. Overfitting

4. L1, L2 regularization

L-1 regularization (Lasso) :

가중치의 L-1 norm을 최소화하는 방법

$$Complexity(Model) = \frac{1}{N}\sum_i |w_i| = \|\mathbf{w}\|_1$$

L-2 regularization (Ridge) :

가중치의 L-2 norm을 최소화하는 방법

$$Complexity(Model) = \frac{1}{N}\sum_i \frac{1}{2}w_i^2 = \frac{1}{2}\|\mathbf{w}\|^2$$

$$\underset{\beta}{\text{minimize}} \left\{ \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 \right\} \quad \text{subject to} \quad \sum_{j=1}^{p} |\beta_j| \le s$$

$$\underset{\beta}{\text{minimize}} \left\{ \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 \right\} \quad \text{subject to} \quad \sum_{j=1}^{p} \beta_j^2 \le s,$$

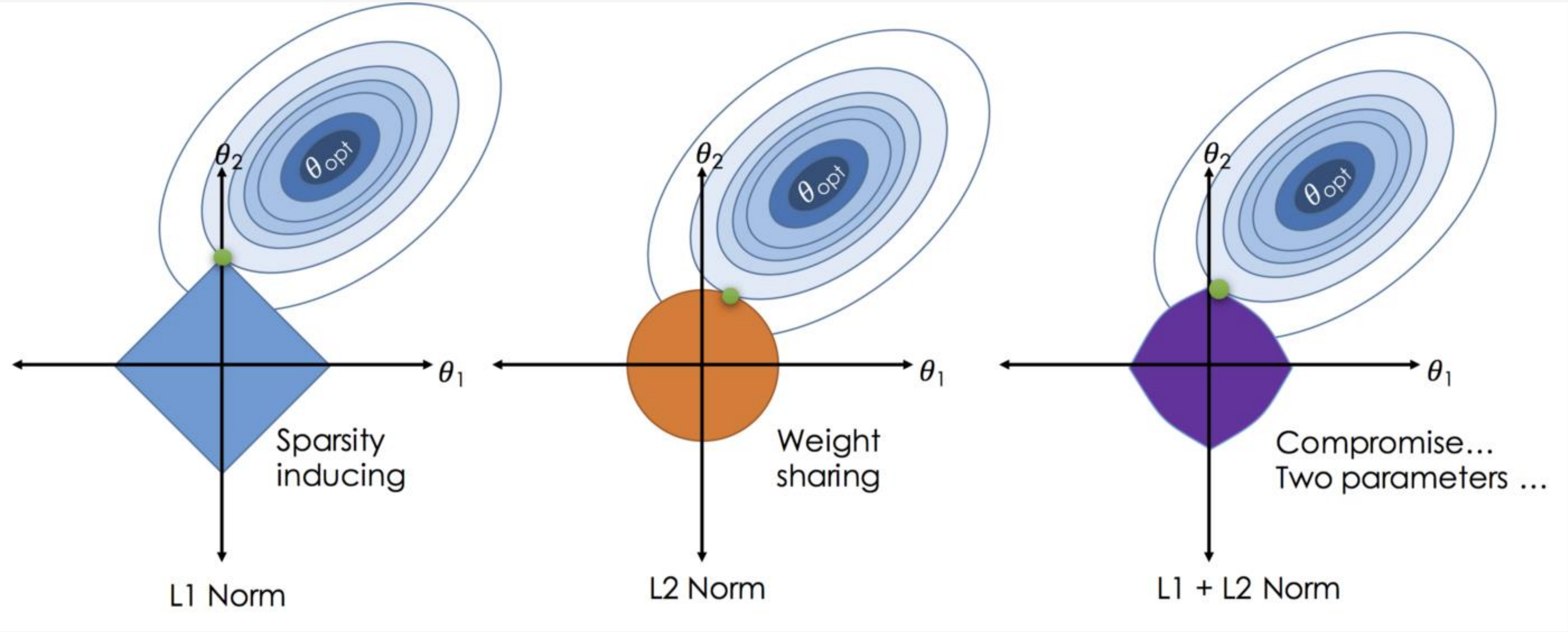
Elastic Net:

Lasso + Ridge

$$Elastic\ Net\ Regression = RSS(\beta) + \lambda_1 \sum_{j=1}^{p} \beta_j^2 + \lambda_2 \sum_{j=1}^{p} |\beta_j|$$

03. Overfitting

4. L1, L2 regularization

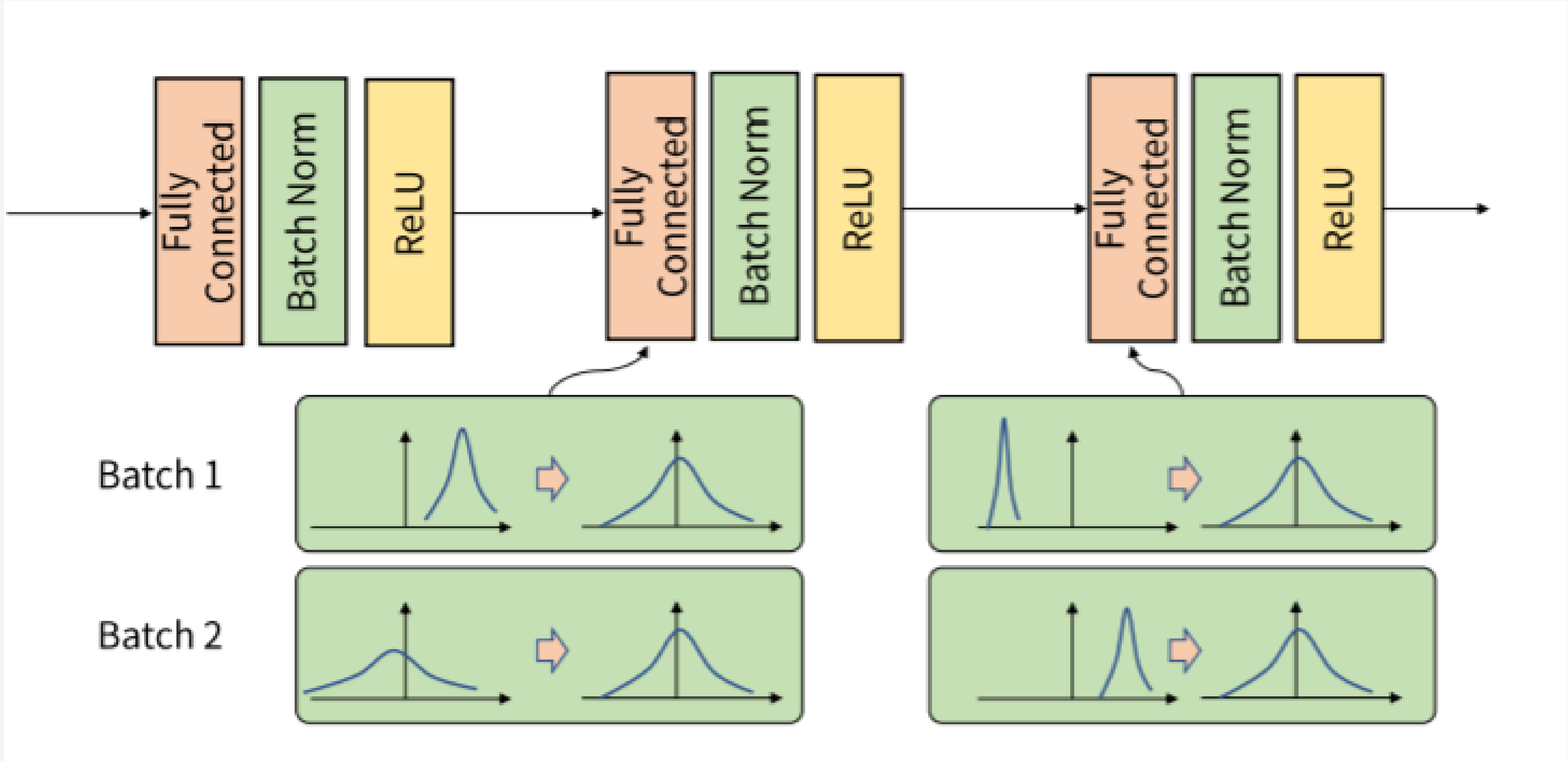


03. Overfitting

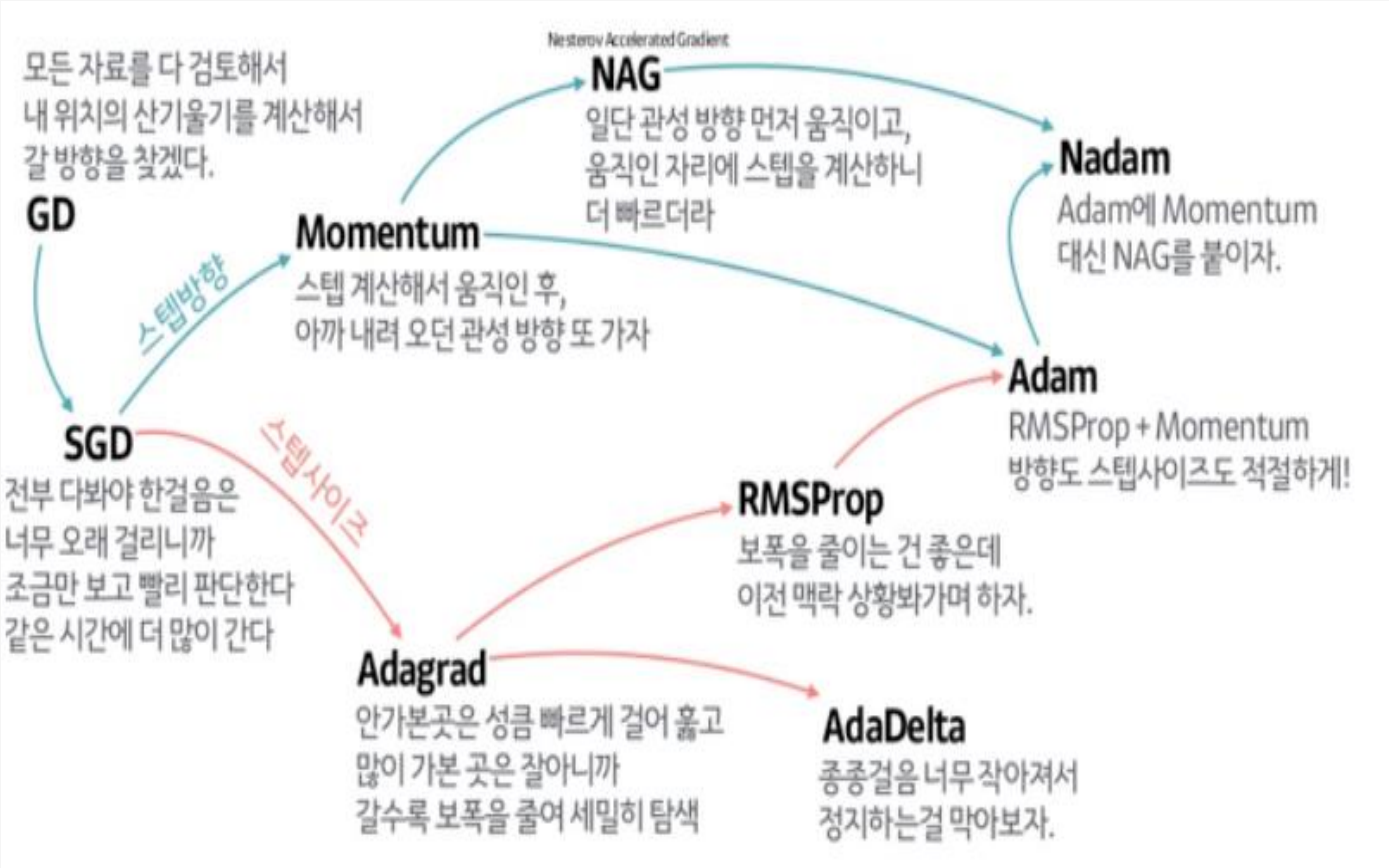
5. Batch Normalization

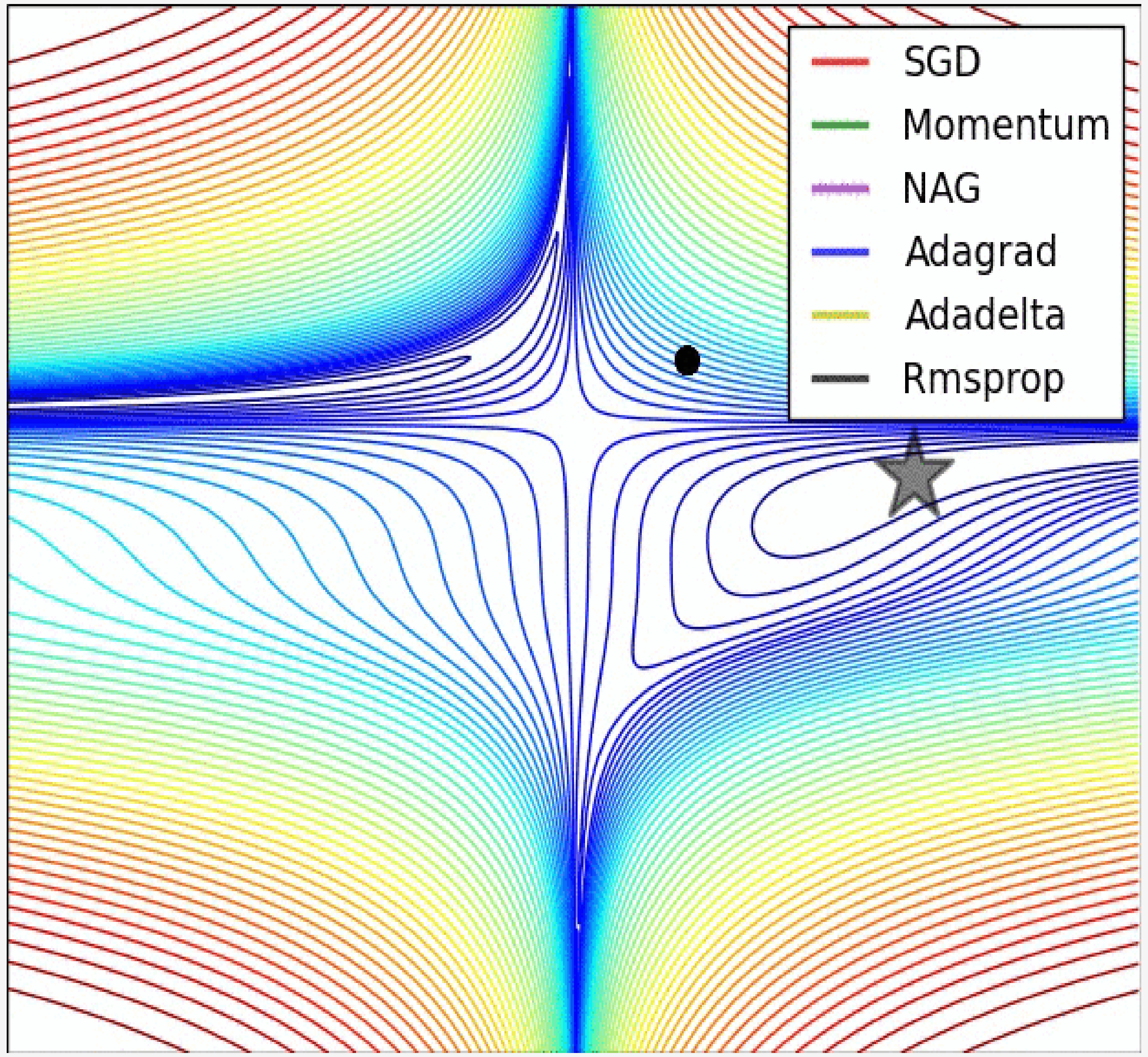
: Activation function의 출력 값을 normalize

-> activation value가 적절하게 분포된 값을 좋은 가중치의 초기값으로 봄.



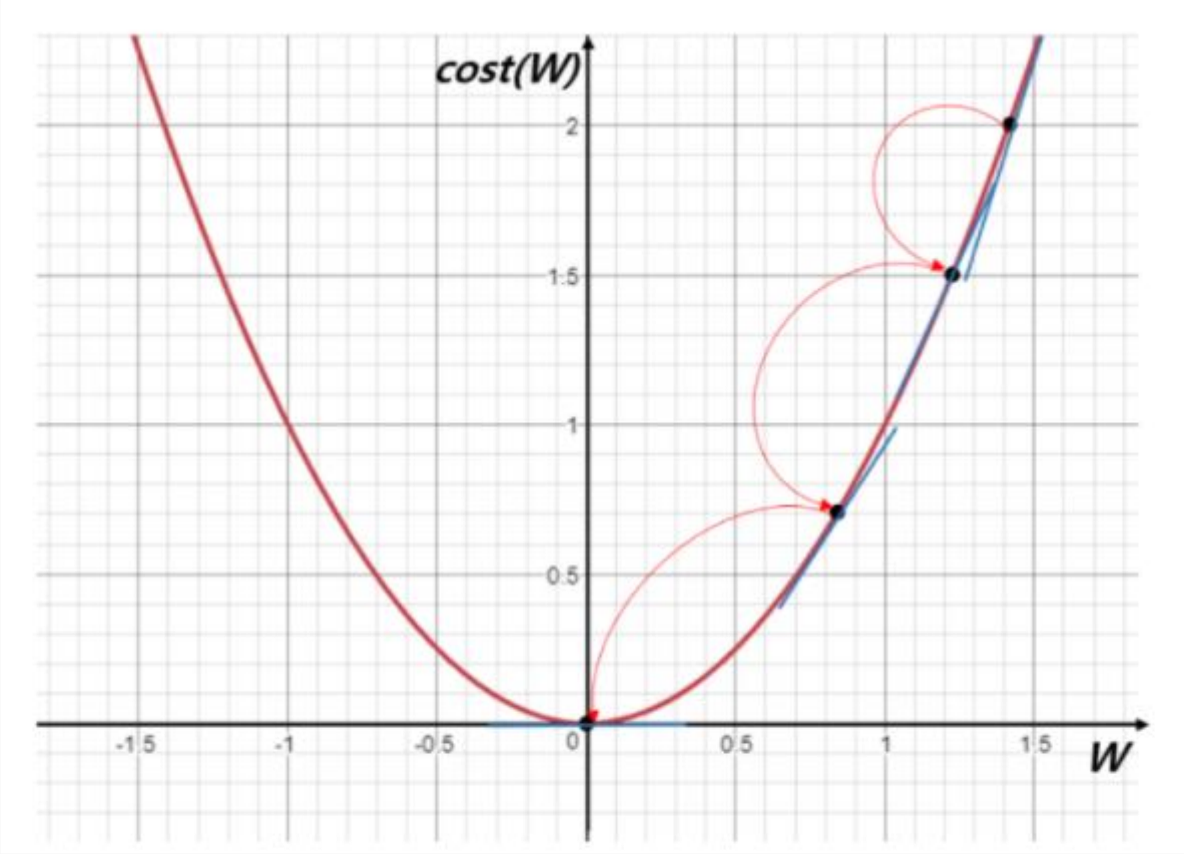
04. Optimizer





04. Optimizer (gradient)

Gradient descent (GD):
경사를 따라 내려가면서 W update



$$w^{+} = w - \boxed{\eta} * \boxed{\frac{\partial E}{\partial w}}$$

learning rate : 한번에 얼마나 학습할지

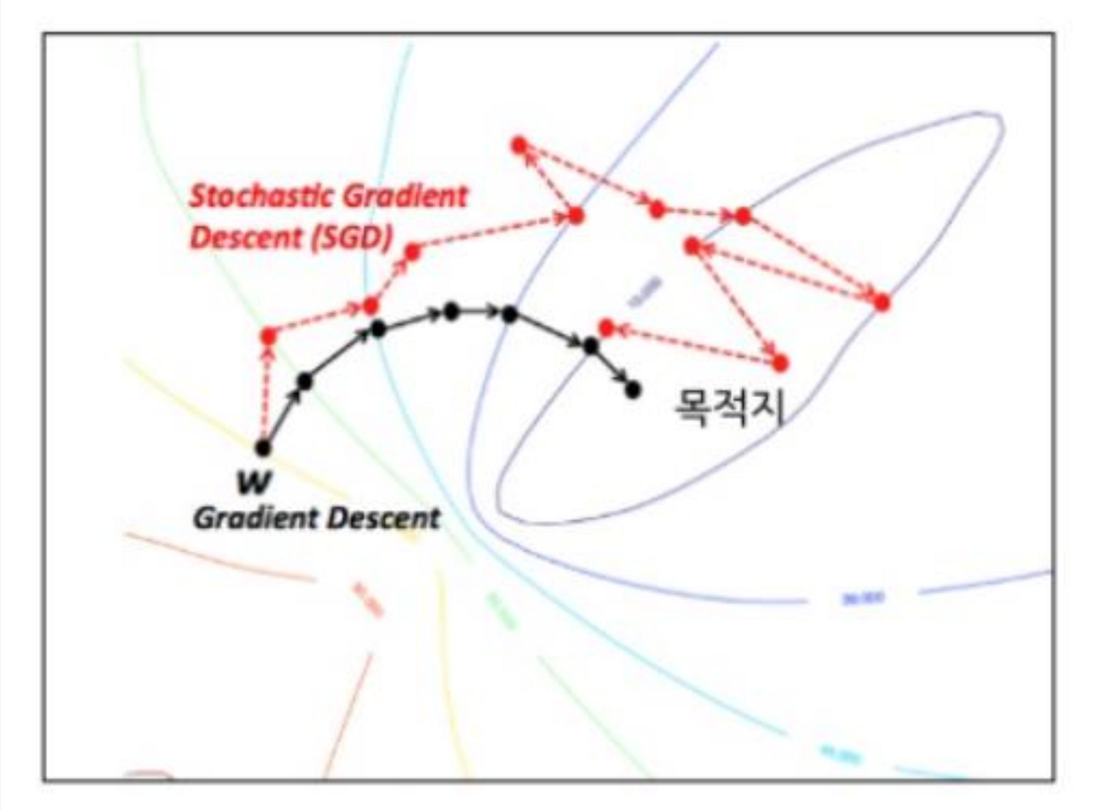
gradient : 어떤 방향으로 학습할지

Epoch

: 전체 training set이 신경망을 통과한 횟수

: 여러 번 반복해서 공부, but overfitting 위험 O

-> 1 epoch: 순전파, 역전파를 통해
신경망을 한 번 통과함



Iteration

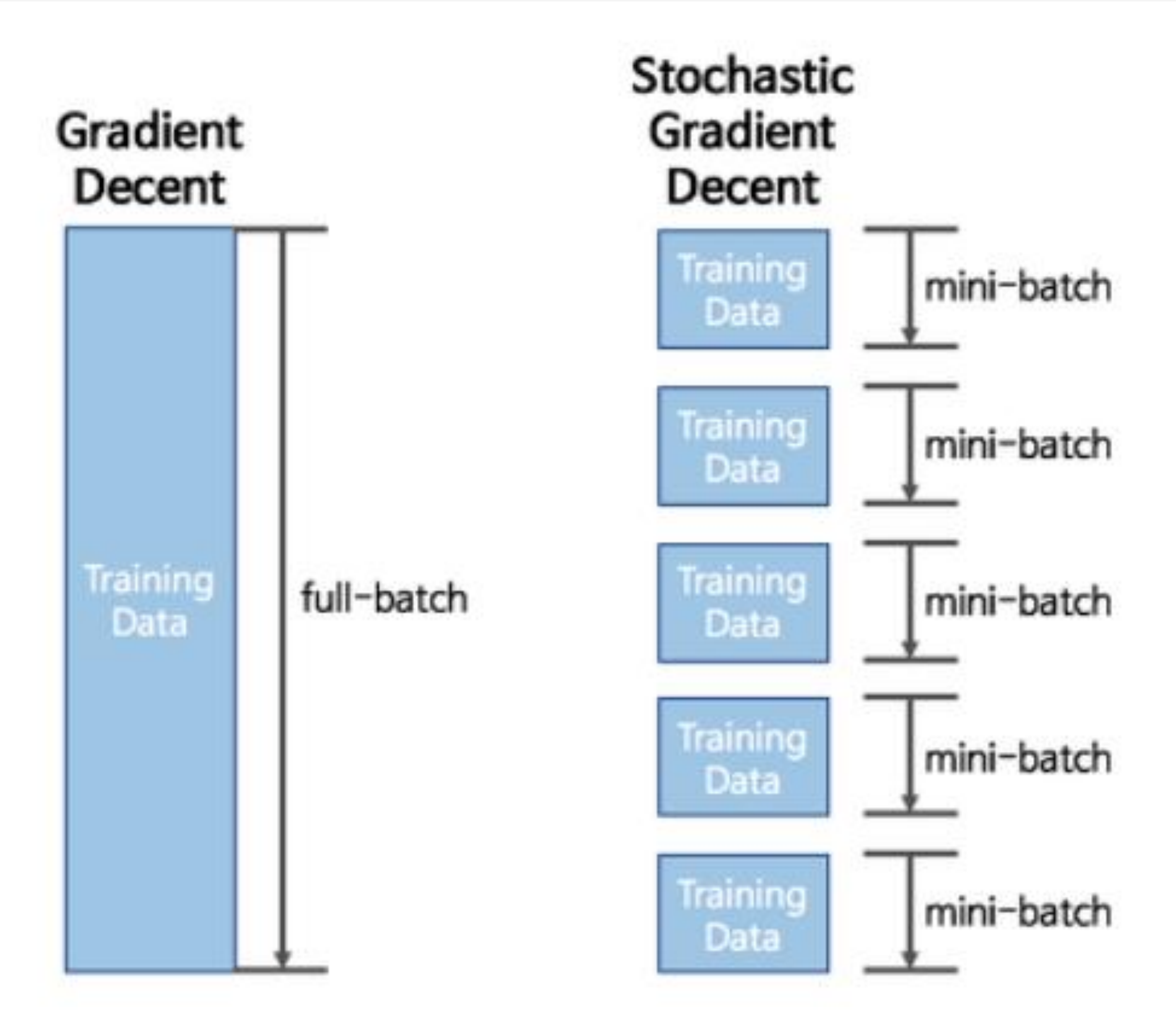
: 전체 데이터를 모델에 한 번 학습시키는데 필요한 배치 수

-> 1 epoch를 마치는데 필요한 parameter 업데이트 횟수

04. Optimizer (gradient)

SGD(stochastic Gradient Decent)

: mini-batch로 학습 진행
: 기울기가 0이 되는 지점에서 학습이 진행되지 X



Momentum:
현재 batch 뿐만 아니라, 이전의 batch 학습결과도 반영

$$v_1 \leftarrow -\eta \frac{\partial L}{\partial W_1} = -0.5 \ (\eta=0.1)$$
$$W \leftarrow W + v_1 = W - 0.5$$

$$v_2 \leftarrow \alpha v_1 - \eta \frac{\partial L}{\partial W_2} = -0.45 - 0.3 = -0.75 \ (\alpha = 0.9)$$
$$W \leftarrow W + v_2 = W - 0.75$$

04. Optimizer (learning rate)

AdaGrad:

학습을 통해 크게 변동이 있었던 가중치에 대해서는 학습률을 감소시키고, 학습을 통해 아직 가중치의 변동이 별로 없었던 가중치는 학습률을 증가시켜서 학습이 되게끔 함.

$$G_t = G_{t-1} + (\nabla_\omega J(\omega_t))^2 = \sum_{i=1}^{k} \nabla_{\omega_i} J(\omega_i)$$

$$\omega_{t+1} = \omega_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \cdot \nabla_\omega J(\omega_t)$$

RMSProp:

- 극점 근처에서 학습속도가 느려지는 문제 해결
- Local minimum에 수렴하는 문제 해결

$$G_t = \gamma G_{t-1} + (1-\gamma)(\nabla_\omega J(\omega_t))^2$$

$$\omega_{t+1} = \omega_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \cdot \nabla_\omega J(\omega_t)$$

04. Optimizer (Gradient + learning rate)

Adam : rmsprop+momentum

: 학습초반에 0으로 biased되는 문제 해결

먼저 초기화를 진행하고, Momentum과 RMSprop에서 사용한 v와 S를 지정해줍니다..

$$v_{dW} = 0, S_{dW} = 0, v_{db} = 0, S_{db} = 0$$

$$v_{dW} = \beta_1 v_{dW} + (1 - \beta_1)dW$$
$$v_{db} = \beta_1 v_{db} + (1 - \beta_1)db$$

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

또한, Momentum에서 소개한 Bias correction을 해주어야 합니다.

$$v_{dW}^{biascorr} = v_{dW}/(1 - \beta^t)$$
$$v_{db}^{biascorr} = v_{db}/(1 - \beta^t)$$
$$S_{dW}^{biascorr} = S_{dW}/(1 - \beta^t)$$
$$S_{db}^{biascorr} = S_{db}/(1 - \beta^t)$$

마지막으로 Momentum과 RMSprop의 가중치 업데이트 방식을 모두 사용하여 가중치 업데이트를 진행합니다.

$$W = W - \alpha v_{dW}^{biascorr} / \sqrt{S_{dW}^{biascorr} + \epsilon}$$
$$b = b - \alpha v_{db}^{biascorr} / \sqrt{S_{db}^{biascorr} + \epsilon}$$

Adam의 하이퍼 파라미터

α : learning rate

β_1 : 1차 moment, 대부분 0.9 (dw의 지수 가중 평균 계산)

β_2 : 2차 moment, 논문에서는 0.99 (dw^2과 db^2의 지수 가중 평균 계산)

ϵ : 논문에서는 0.10^{-8} (성능에 거의 영향 X)

THANK YOU

감사합니다.