딥러닝기초

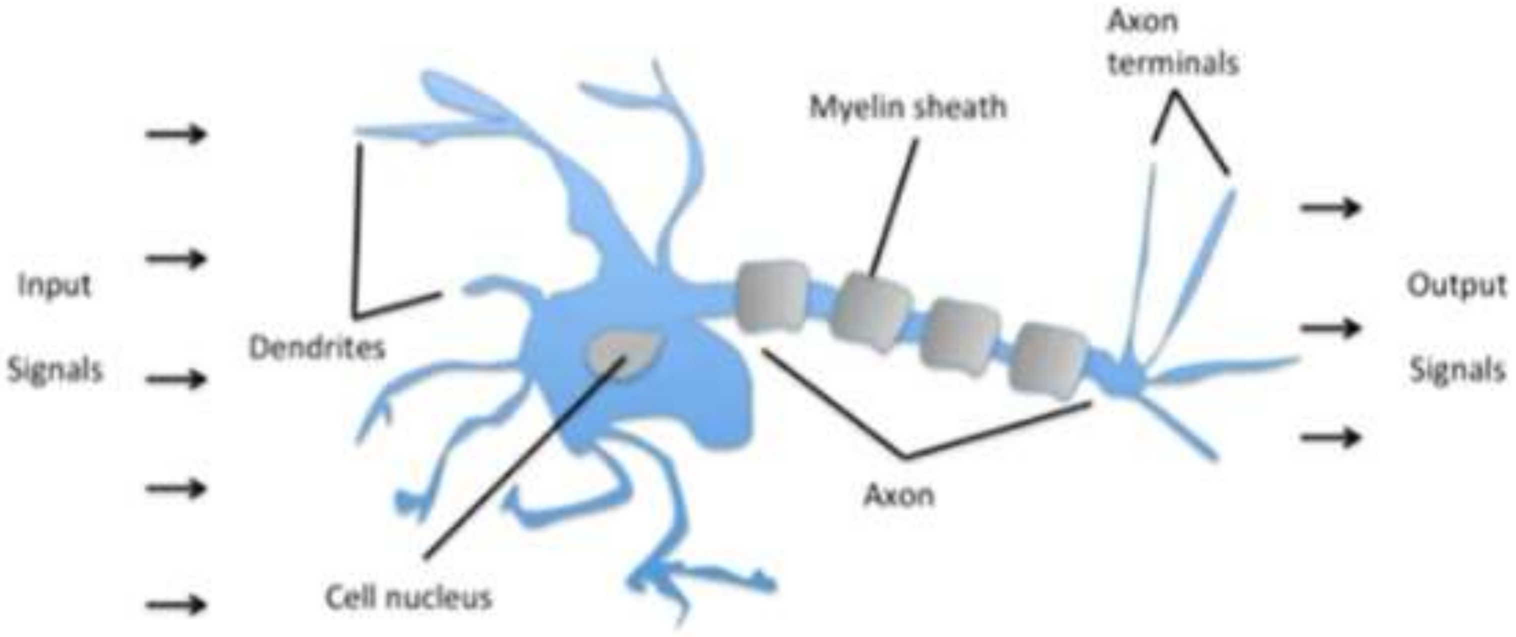
BOAZ BASE 분석 세션

16기 분석 김지원

딥러닝의 시작

딥러닝의 시작

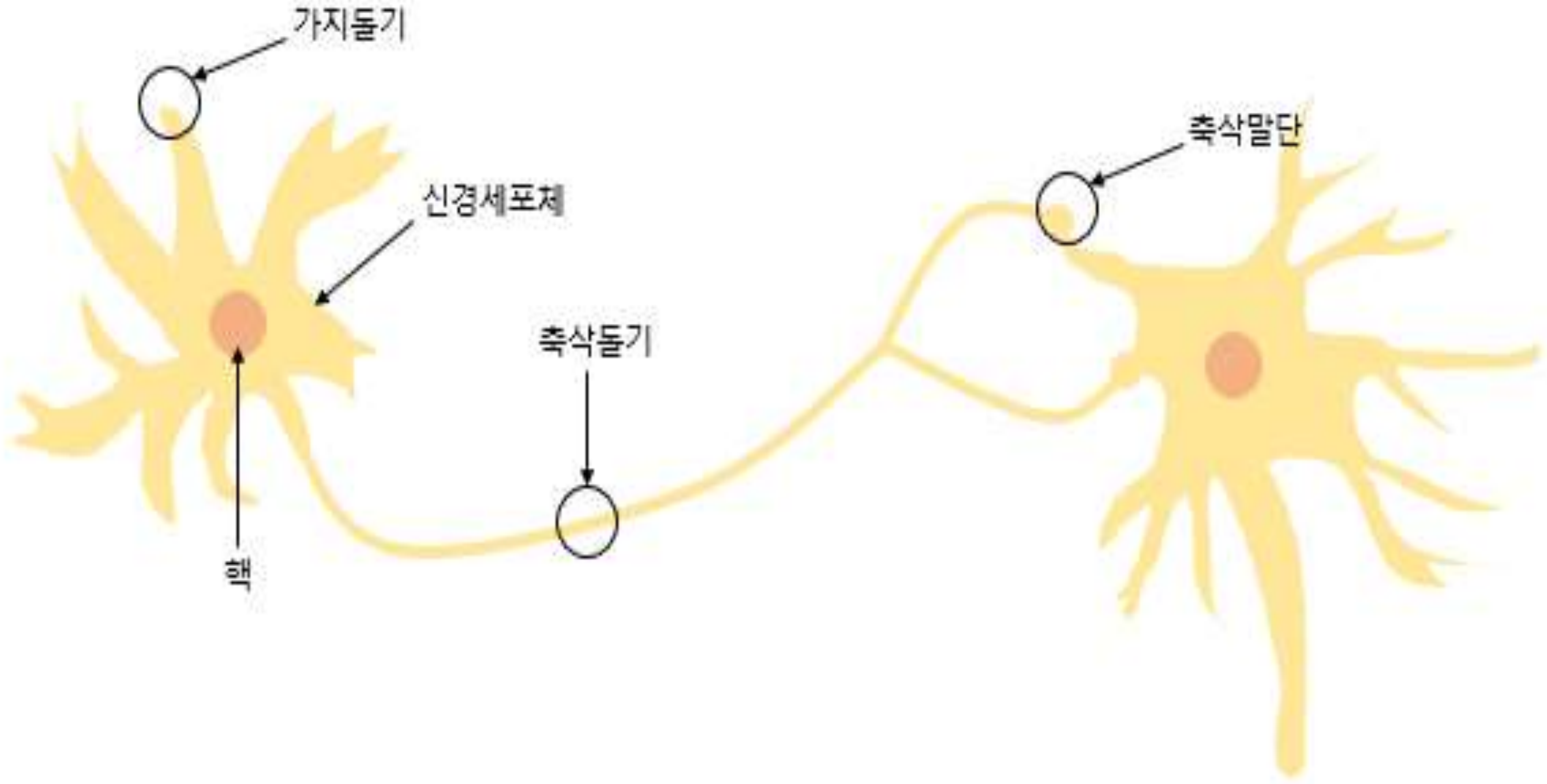
Ultimate dream: thinking machine



Schematic of a biological neuron.

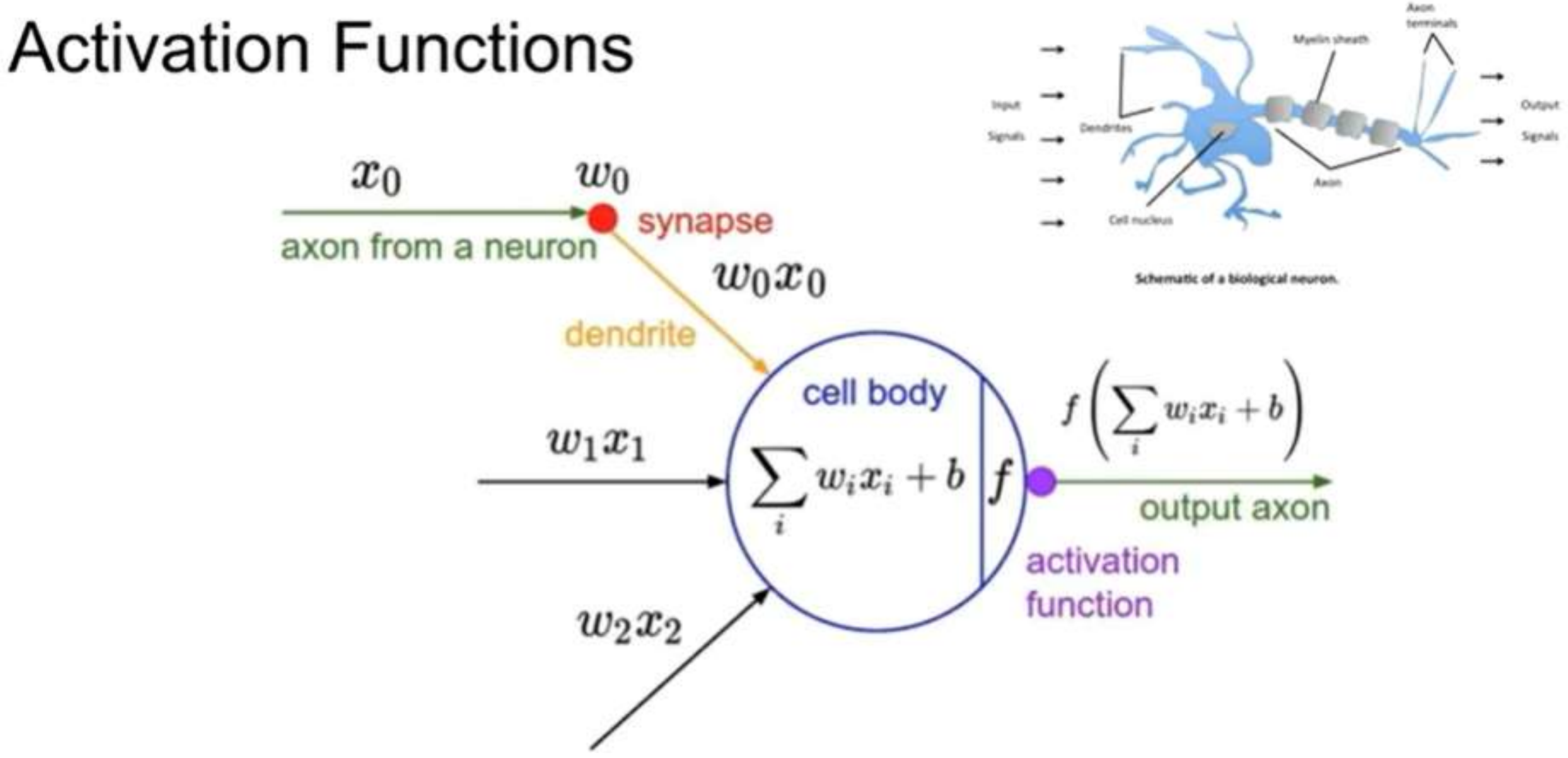
퍼셉트론

신경 세포 뉴런의 동작



퍼셉트론이란?

입력 값에 대해 가중치를 적용한 뒤, 결과를 전달하는 방식

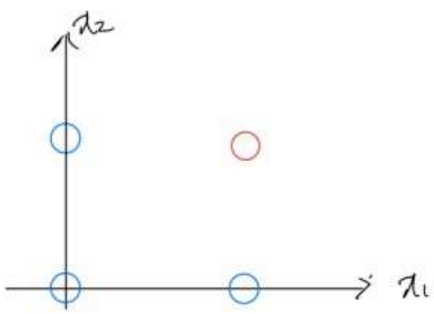


단순한 논리회로

적절하게 매개변수만 조절해주면 모두 표현 가능하다!

AND

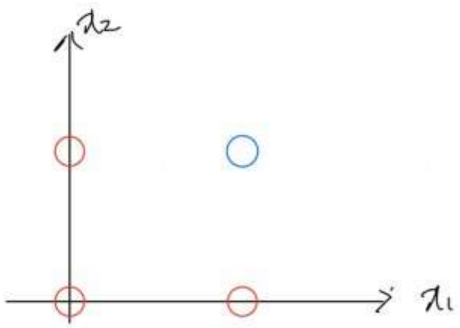
x_1	x_2	y
0	0	0
1	0	0
0	1	0
1	1	1



$$(w_1, w_2, \theta) = (0.5, 0.5, 0.7)$$

NAND

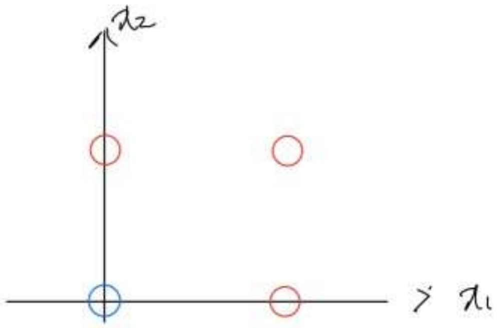
x_1	x_2	y
0	0	1
1	0	1
0	1	1
1	1	0



$$(w_1, w_2, \theta) = (-0.5, -0.5, -0.7)$$

OR

x_1	x_2	y
0	0	0
1	0	1
0	1	1
1	1	1

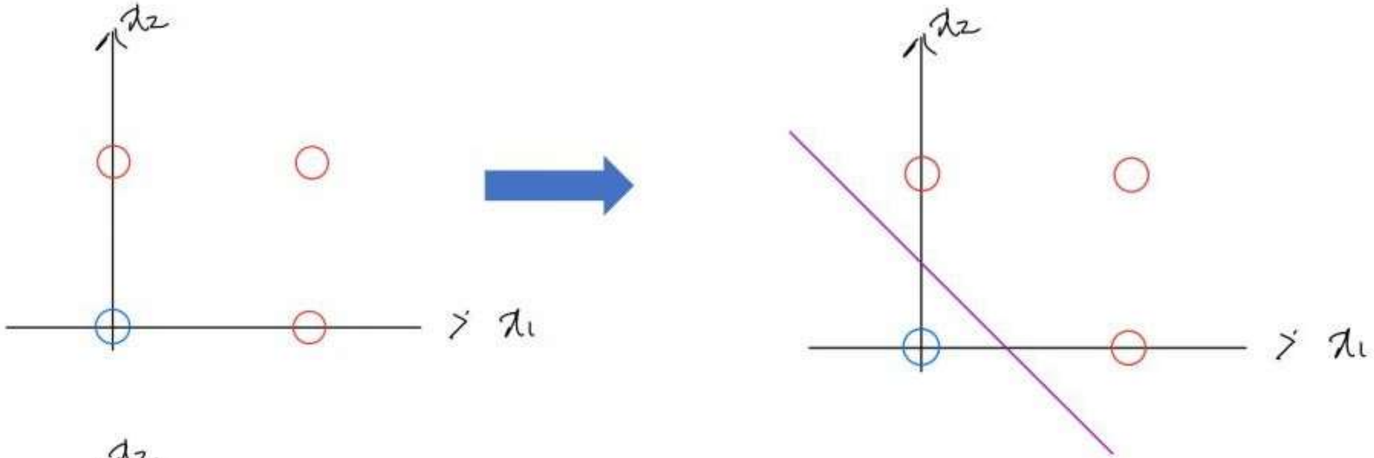


$$(w_1, w_2, \theta) = (0.5, 0.5, 0.2)$$

퍼셉트론의 한계

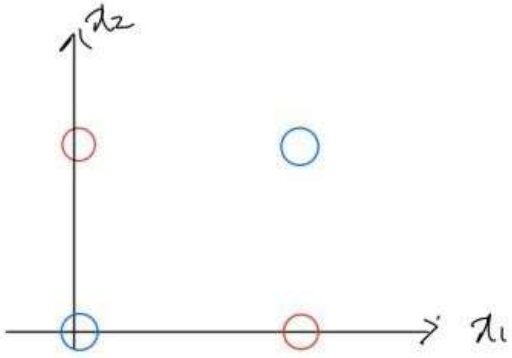
OR

x_1	x_2	y
0	0	0
1	0	1
0	1	1
1	1	1



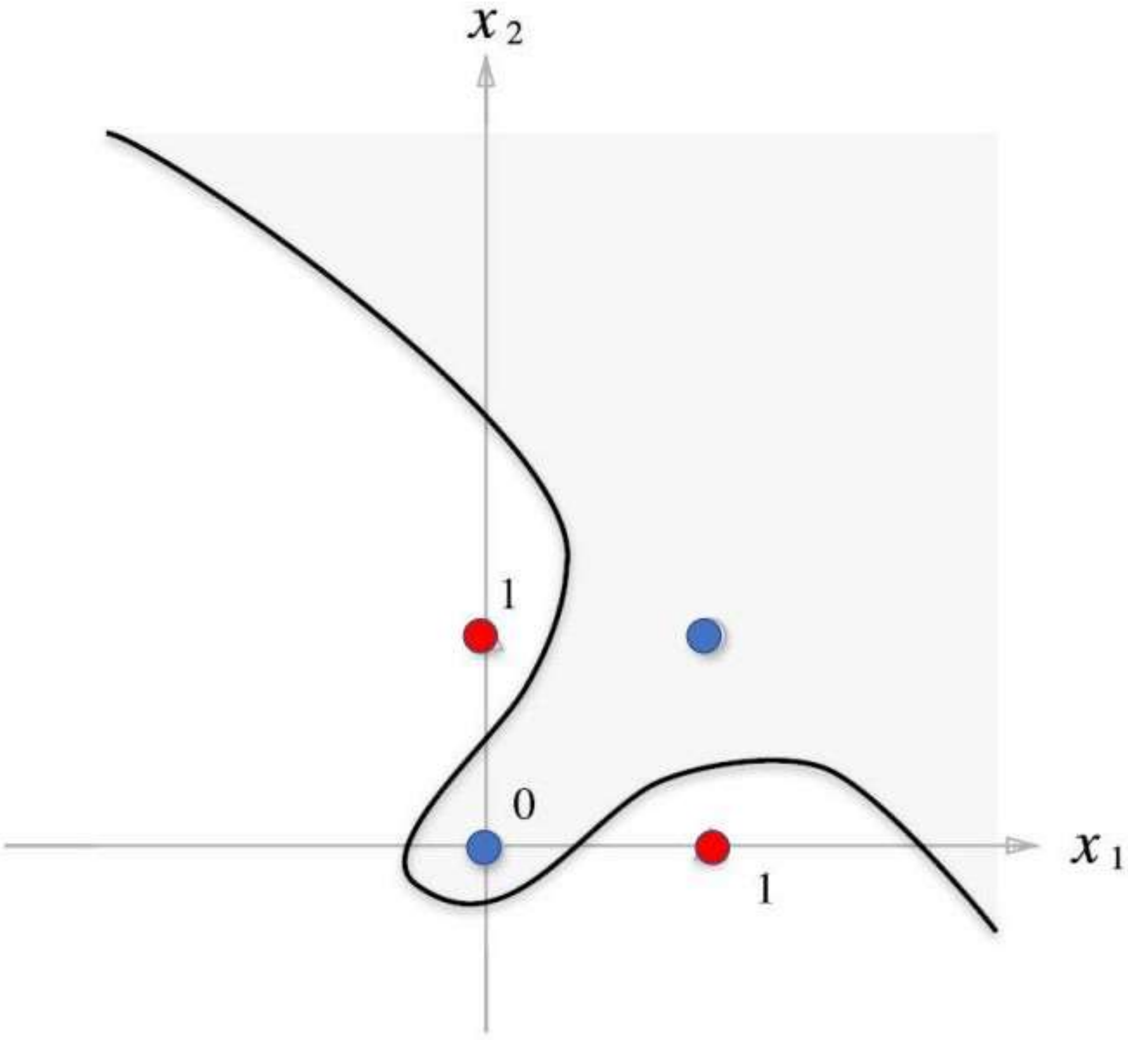
XOR

x_1	x_2	y
0	0	0
1	0	1
0	1	1
1	1	0



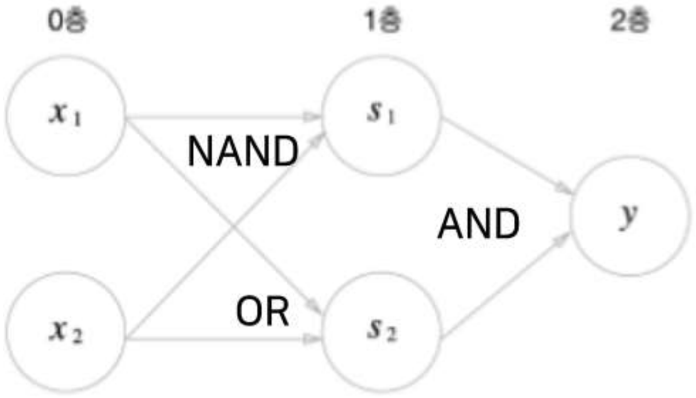
$$y = \begin{cases} 0 & (-0.5 + x_1 + x_2 \leq 0) \\ 1 & (-0.5 + x_1 + x_2 > 0) \end{cases}$$

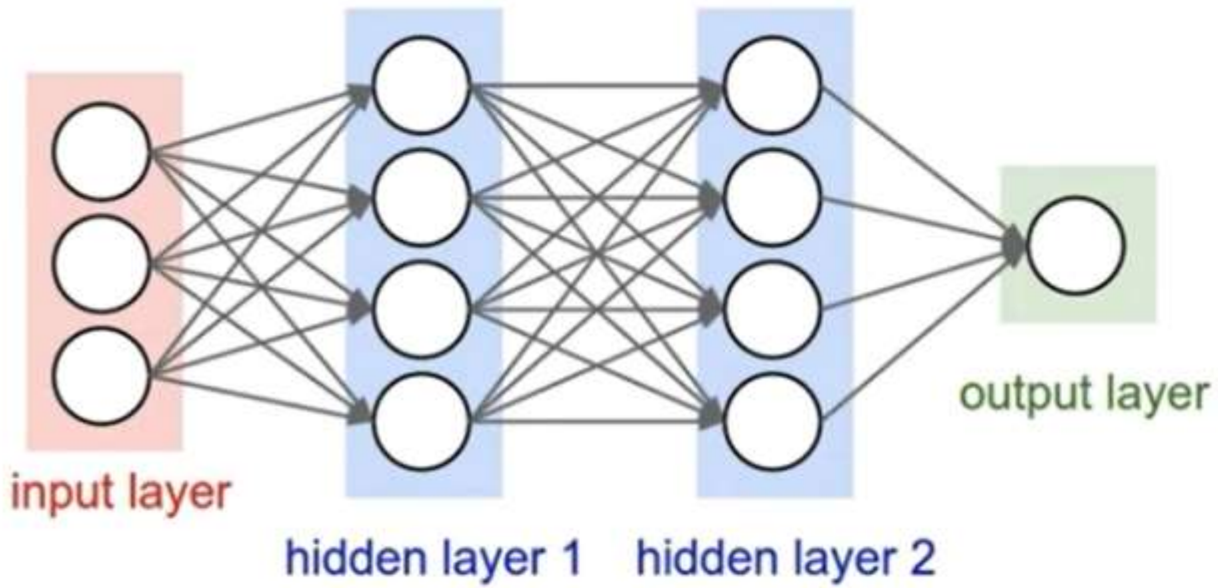
퍼셉트론의 한계



다층 퍼셉트론 MLP

- 퍼셉트론 여러개를 이어 붙여 층을 쌓는다!

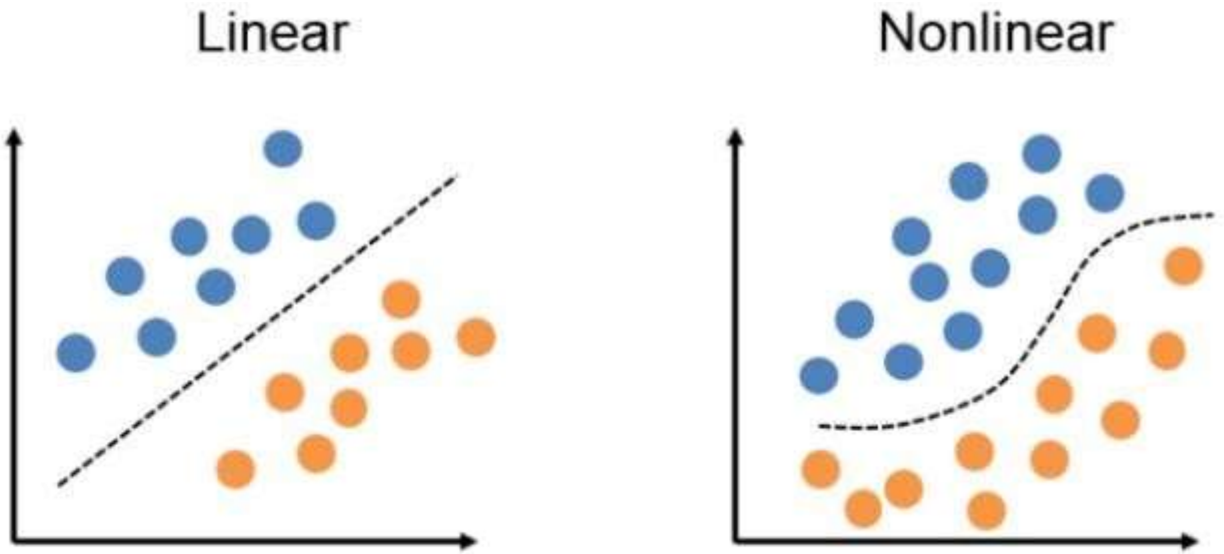




선형 분류만으로는 풀지 못했던 문제를 비선형적으로 해결
(hidden layer 가 2개 이상이면 신경망이라고 한다.)

다층 퍼셉트론

- 다층 레이어
- 선형성 극복
- 다양한 결과 가능

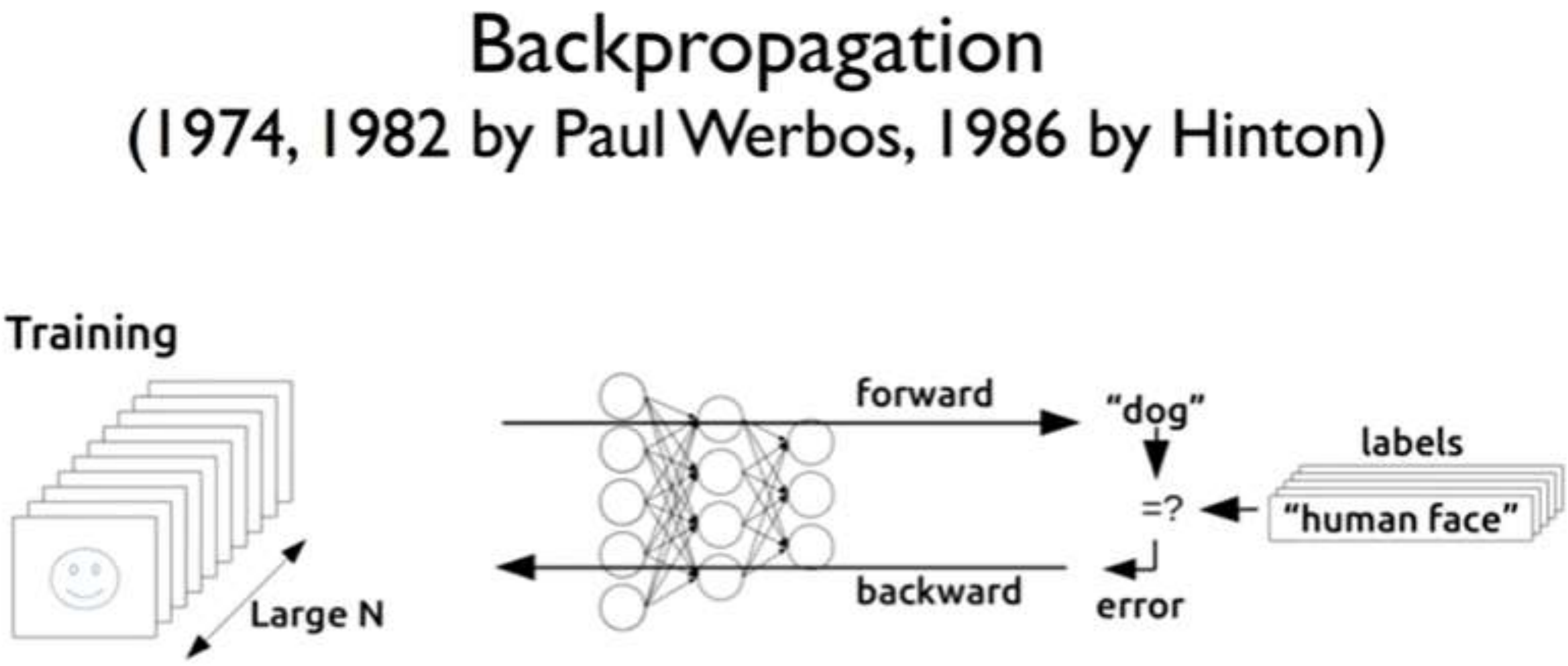


- But MLP는 각 layer의 W, bias를 학습할 수가 없었음.
(Backpropagation이 나오기 전까지)

역전파

backpropagation

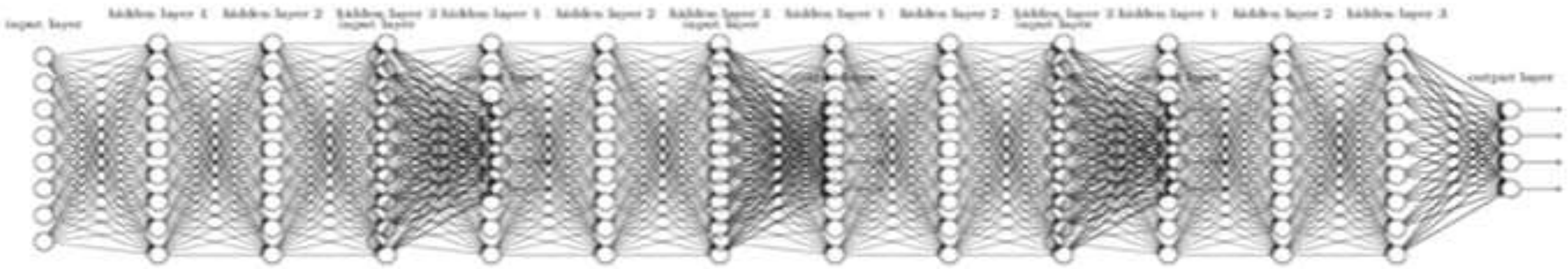
오차 역전파


Backpropagation
(1974, 1982 by Paul Werbos, 1986 by Hinton)
Training
Large N
forward
backward
"dog"
=?
error
labels
"human face"

오차 역전파

A BIG problem

- Backpropagation just did not work well for normal neural nets with many layers
- Other rising machine learning algorithms: SVM, RandomForest, etc.
- **1995** "Comparison of Learning Algorithms For Handwritten Digit Recognition" by LeCun et al. found that this new approach worked better



오차 역전파

Breakthrough

in 2006 and 2007 by Hinton and Bengio

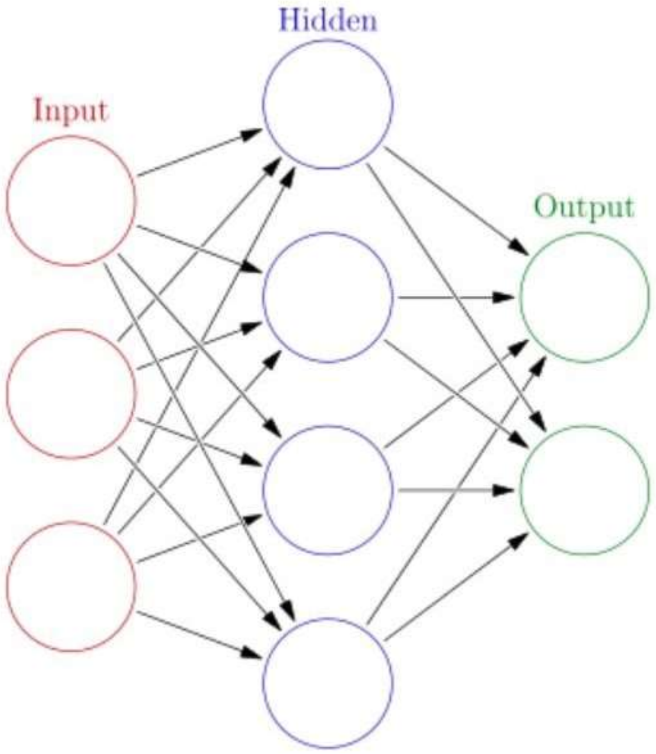
- Neural networks with many layers really could be trained well, *if the weights are initialized in a clever way* rather than randomly.
- Deep machine learning methods are more efficient for difficult problems than shallow methods.
- Rebranding to ***Deep Nets***, ***Deep Learning***

역전파 계산

신경망 학습

구성

- 입력층 (Input)
- 은닉층 (Hidden Layer)
- 출력층 (Output)



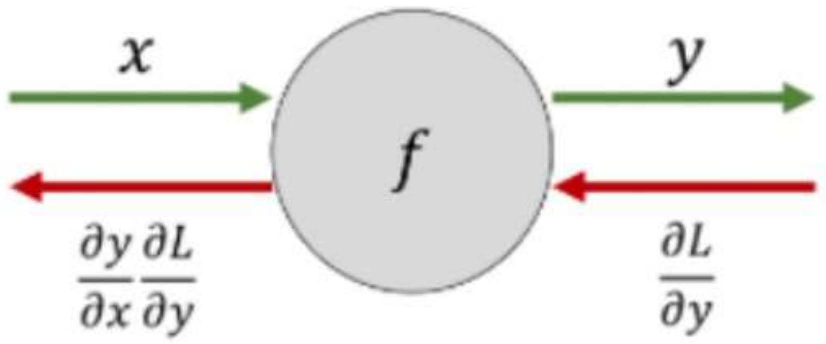
신경망 학습

Forward propagation -> computational node를 순서대로 방문하면서 loss값을 계산한다!

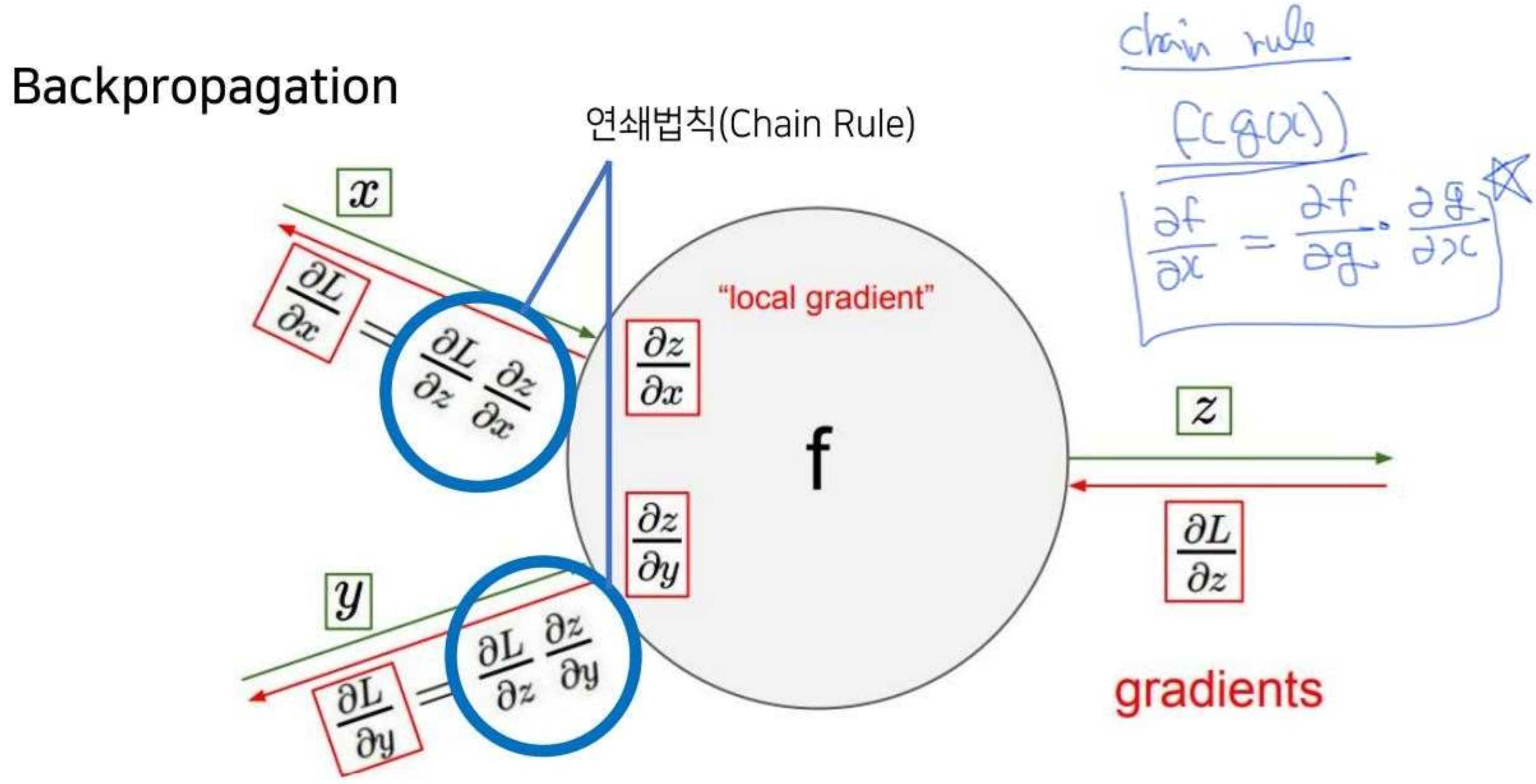
- Input에서 Output 방향으로 결과 값을 내보내는 과정

Backpropagation ->loss값을 최소화하는 가중치 w를 찾기 위해 backpropagation을 한다!

- 결과 값을 통해 역으로 Input 방향으로 오차를 다시 보내며 가중치 업데이트

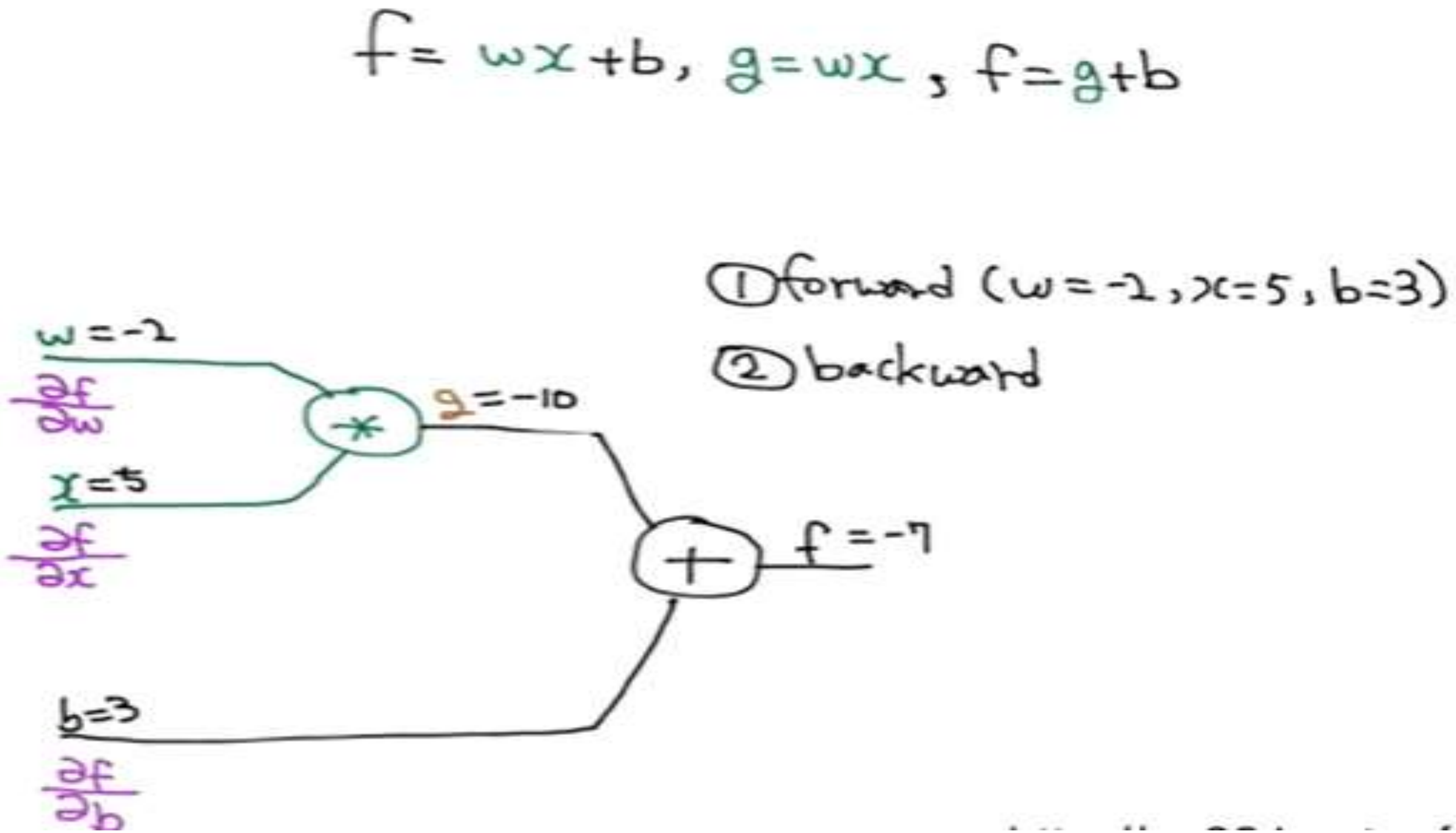


신경망 학습



Backpropagation
연쇄법칙(Chain Rule)
"local gradient"
gradients
chain rule
f(g(x))
$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial g} \cdot \frac{\partial g}{\partial x}$

신경망 학습



$f = wx + b, \; g = wx, \; f = g + b$
① forward ($w=-2, x=5, b=3$)
② backward
$w=-2$
$\frac{\partial f}{\partial w}$
$x=5$
$\frac{\partial f}{\partial x}$
$g=-10$
$f=-7$
$b=3$
$\frac{\partial f}{\partial b}$

신경망 학습

Back propagation (chain rule)

$f = wx + b$, $g = wx$, $f = g + b$

$\frac{\partial f}{\partial g} = 1$, $\frac{\partial f}{\partial b} = 1$

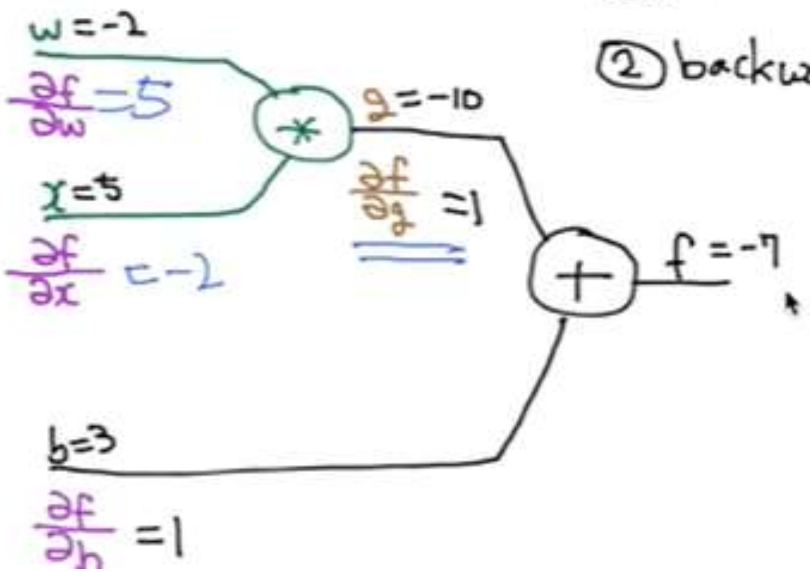
$\frac{\partial g}{\partial w} = x$, $\frac{\partial g}{\partial x} = w$

$$\frac{\partial f}{\partial w} = \frac{\partial f}{\partial g} \cdot \frac{\partial g}{\partial w} = 1 * 5 = 5$$

$$\frac{\partial f}{\partial x} = \underbrace{\frac{\partial f}{\partial g}}_{=1} \underbrace{\frac{\partial g}{\partial x}}_{=-2} = -2$$

① forward ($w=-2$, $x=5$, $b=3$)

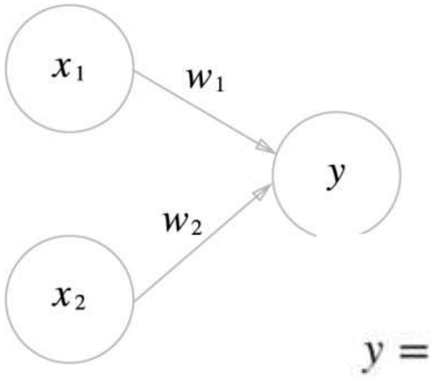
② backward



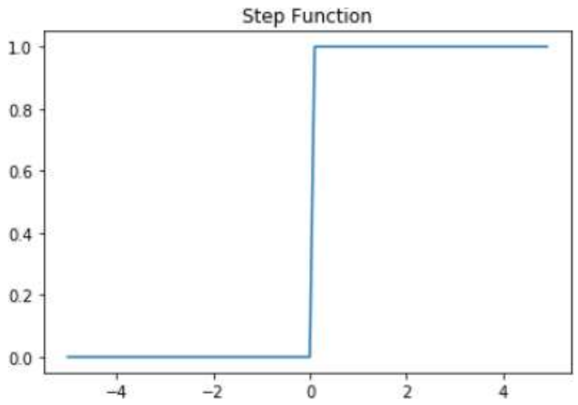
신경망

퍼셉트론에서 신경망으로

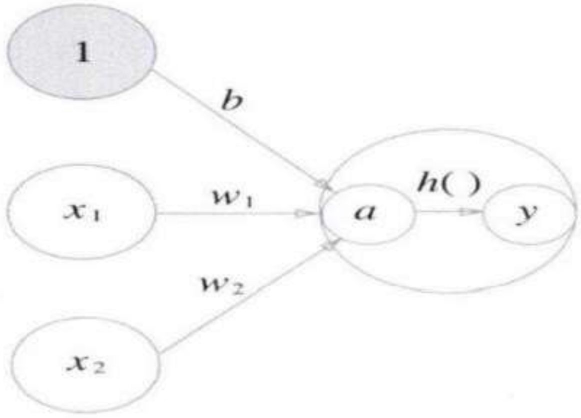
퍼셉트론



$$y = \begin{cases} 0 & (w_1 x_1 + w_2 x_2 \leq \theta) \\ 1 & (w_1 x_1 + w_2 x_2 > \theta) \end{cases}$$



신경망



$$y = h(b + w_1 x_1 + w_2 x_2)$$

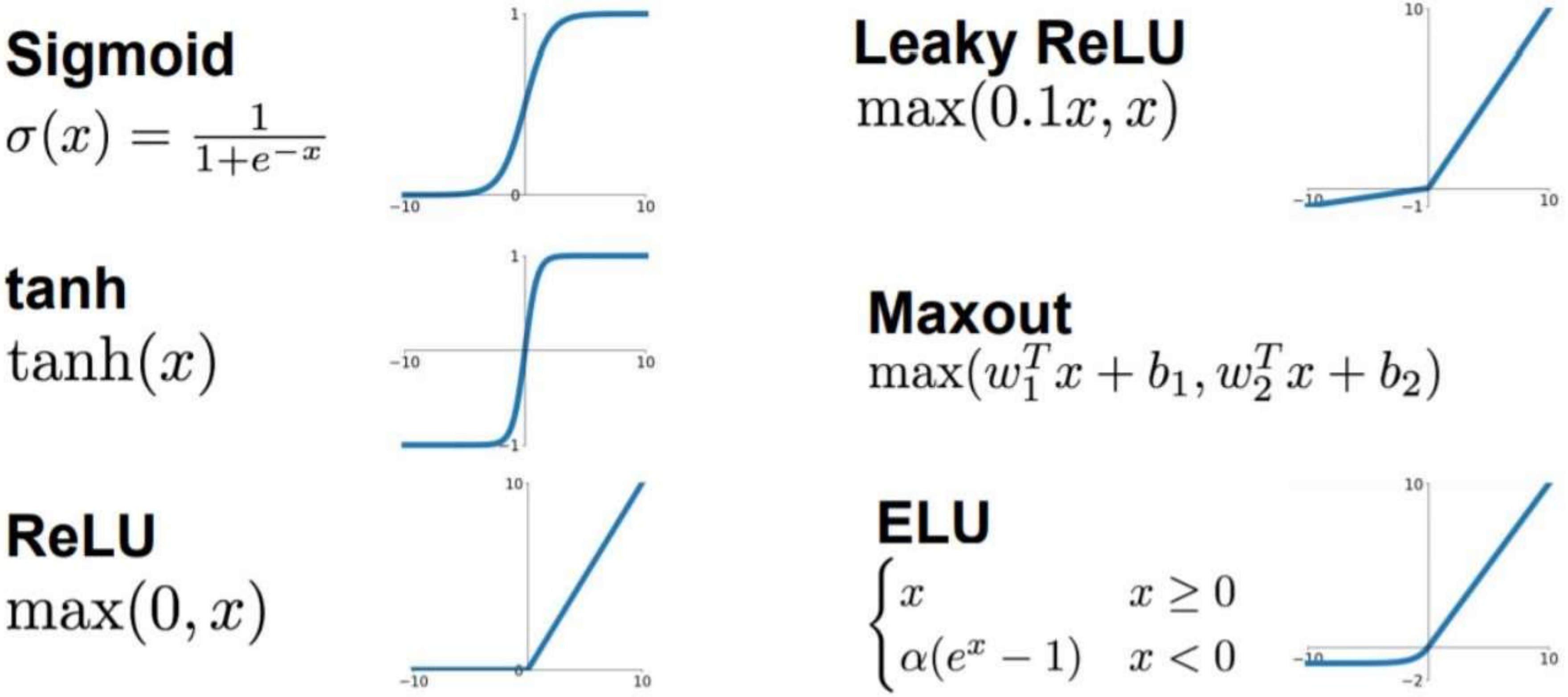
$$h(x) = \begin{cases} 0 \ (x \leq 0) \\ 1 \ (x > 0) \end{cases}$$

활성화 함수!

활성화 함수

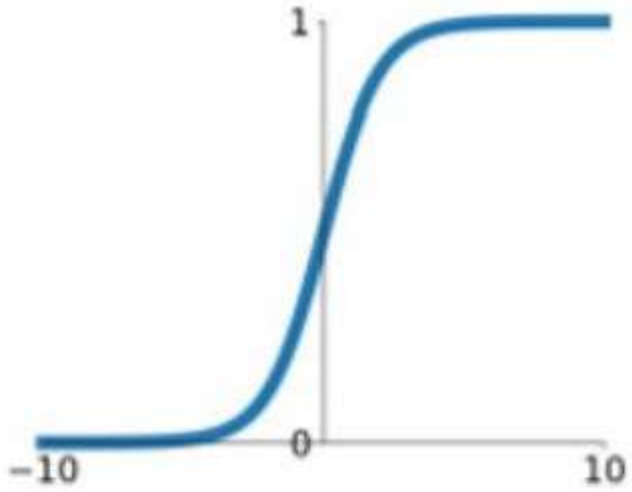
노드에 들어오는 값들에 대해 다음 레이어로 전달하기 전 비선형 함수를 사용

사용 이유 : 선형 함수를 사용할 경우 층을 깊게하는 의미가 줄어든다. 딥러닝에서 활성화 함수는 비선형 함수라고 생각하면 된다.



활성화 함수

- **sigmoid**



Sigmoid

Sigmoid 함수

- 결과 값이 (0,1)사이로 출력
- 극단의 값은 대부분 0 또는 1
- 신경망 초기에 사용

단점

- Gradient Vanishing 발생
- 함수 값 중심이 0이 아님 -> zig zag 현상
- Exp 사용으로 비용 많이 발생

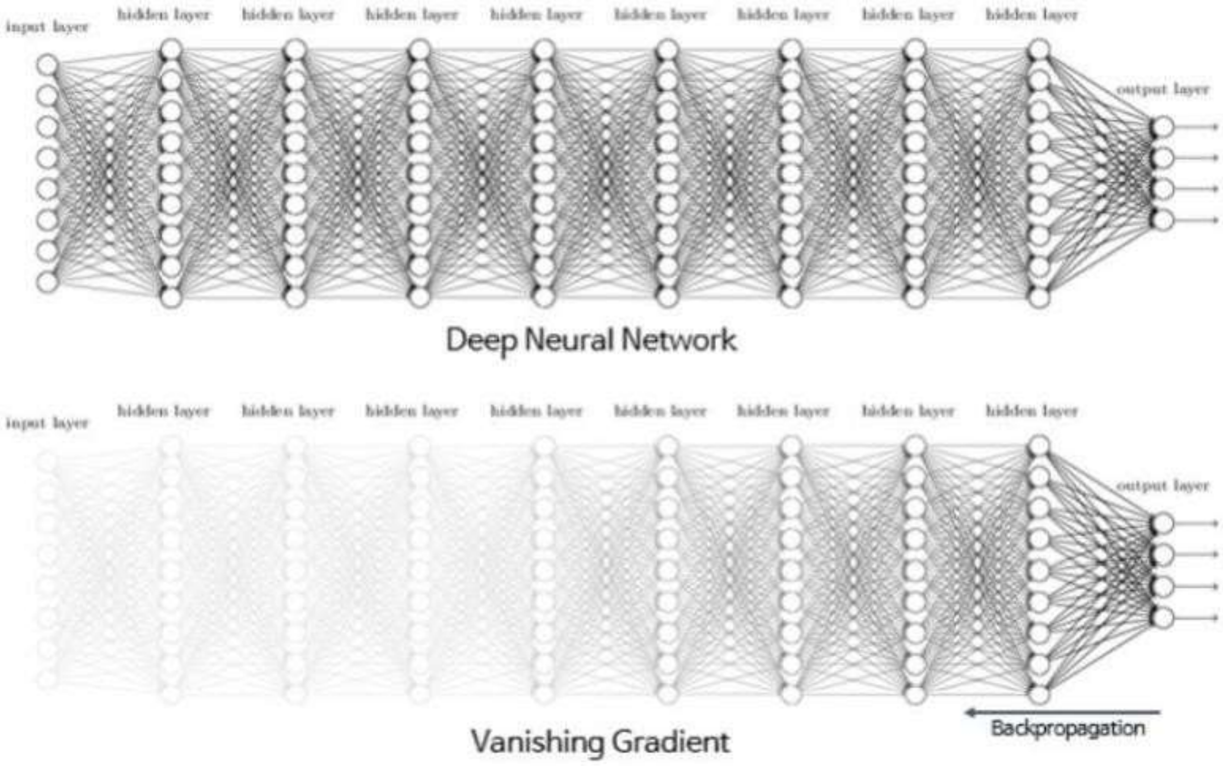
활성화 함수

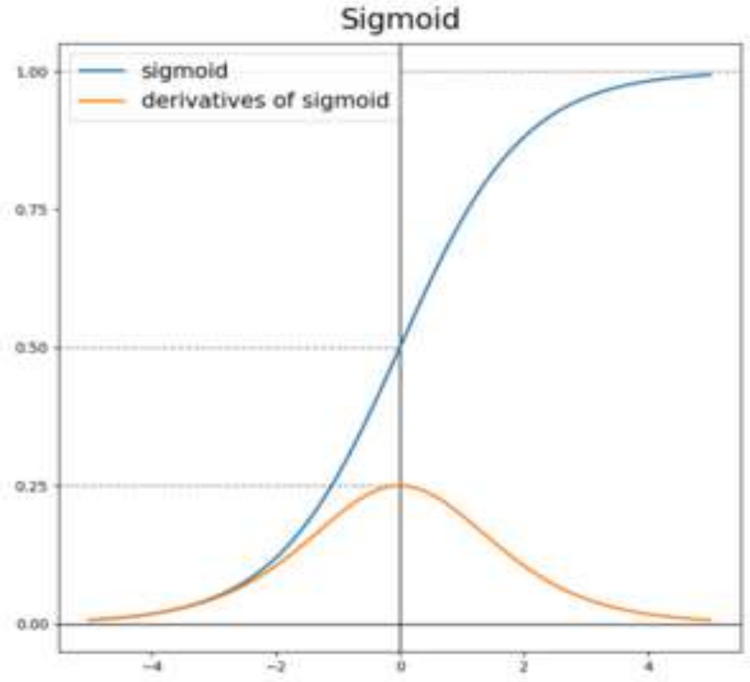
- **Gradient Vanishing? Gradient 항이 사라지는 문제**

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

learning rate : 한번에 얼마나 학습할지

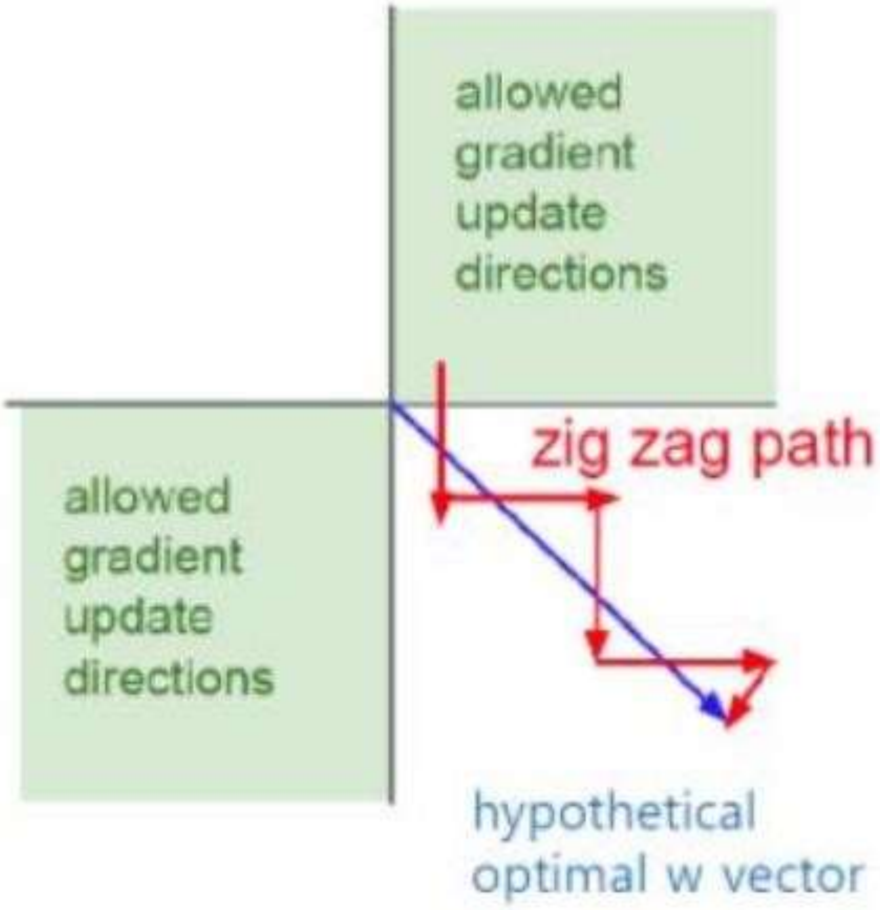
gradient : 어떤 방향으로 학습할지





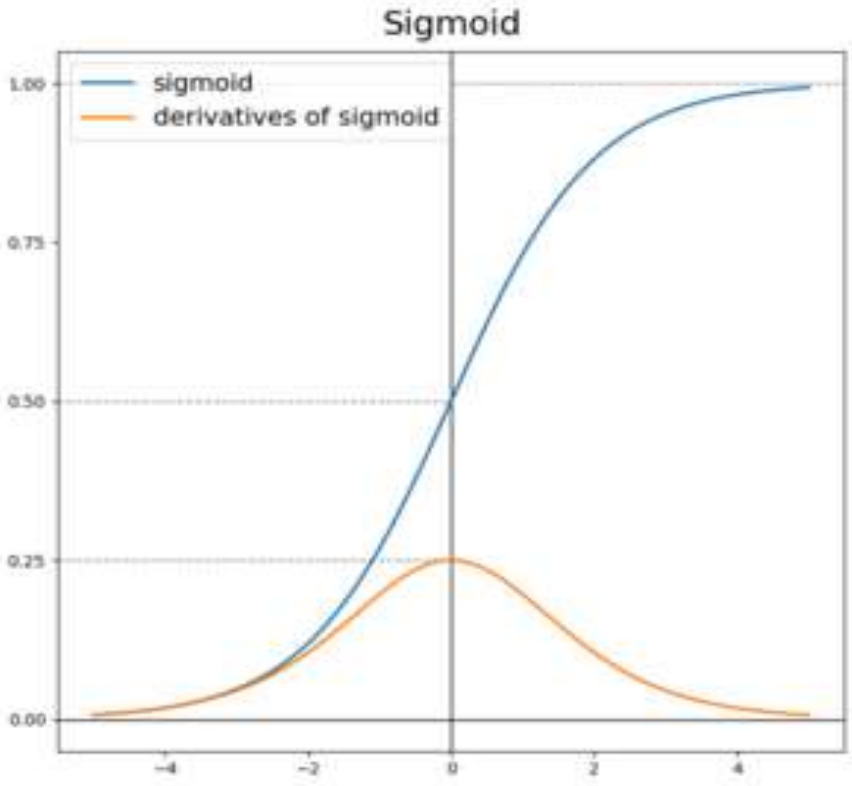
활성화 함수

- **지그재그 현상?**



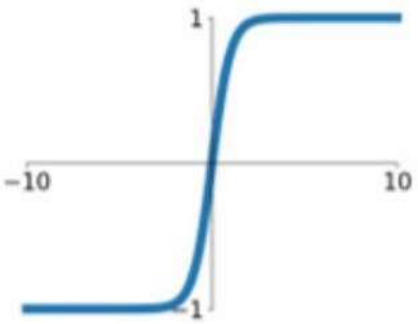
함수 값이 모두 양수

-> Loss가 가장 낮은 지점을 찾아 가는데 정확한 방향으로 가지 못하고 지그재그로 수렴



활성화 함수

- **tanh(x)**



tanh(x)

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

tahn 함수

- Sigmoid와 유사
- 결과 값이 (-1,1)사이로 출력
- Zigzag현상이 덜하다

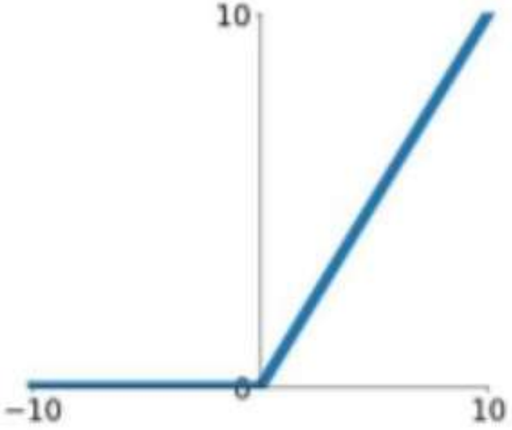
단점

- Gradient Vanishing 발생

활성화 함수

- **ReLU**

$$h(x) = \begin{cases} x & (x > 0) \\ 0 & (x \le 0) \end{cases}$$



ReLU

$$\boldsymbol{f(x) = max(0, x)}$$

ReLU 함수

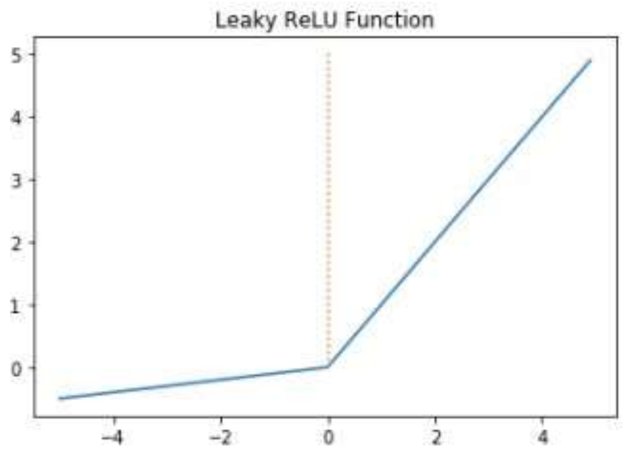
- 현재 가장 많이 사용하는 함수
- 학습이 빠름
- 연산 비용이 크지않고 구현이 간단

단점

- zig zag 현상 발생
- X<0 값들로 뉴런이 죽을 수도 있음

활성화 함수

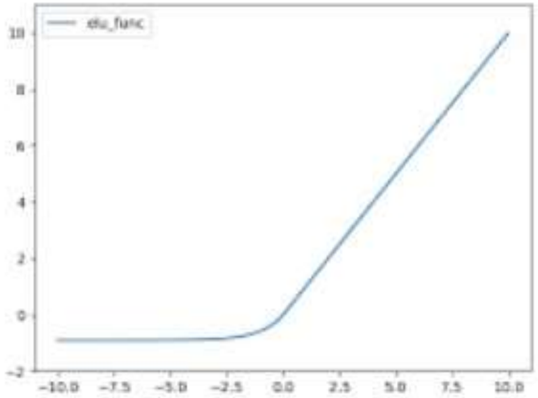
Leaky ReLU



$$\max(0.01, x)$$

ReLU 음수부분 개선

ELU



$$\begin{cases} x & x > 0 \\ \alpha(e^{x} - 1) & x \leq 0 \end{cases}$$

ReLU 특성 공유
gradient vanishing 극복

Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

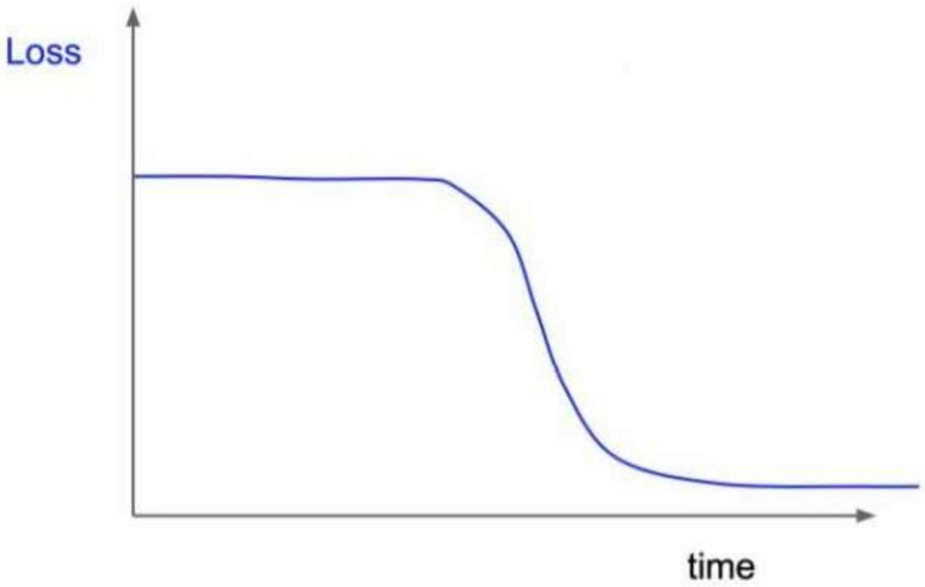
연결된 두 뉴런 값 중 큰 값 이용
연산량이 많이 필요

Weight initialization

weight initialization

초기 가중치 설정 (weight initialization)

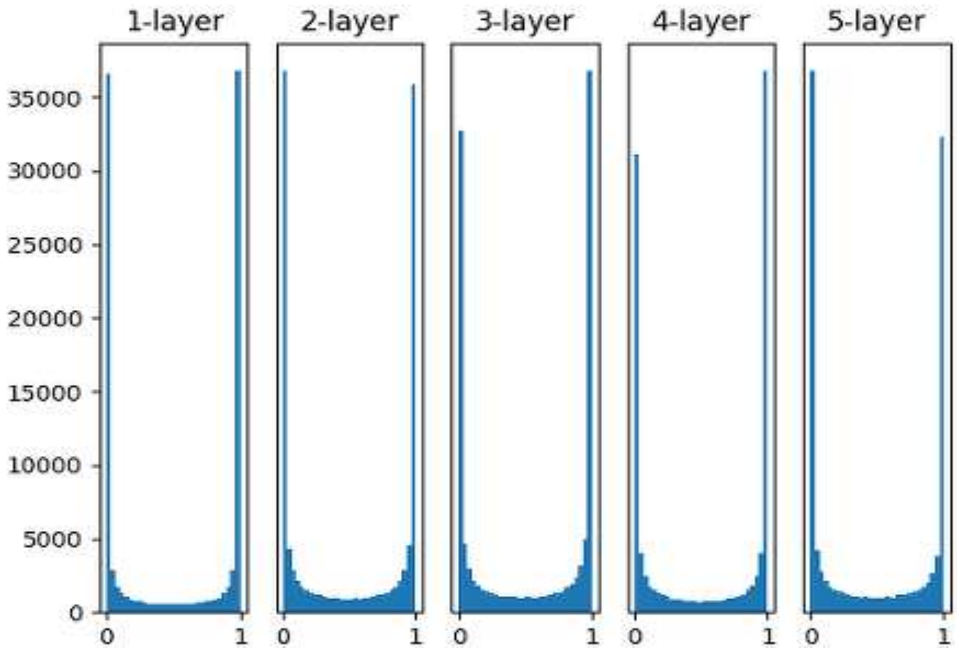
- 딥러닝 학습에 있어 초기 가중치 설정은 매우 중요한 역할
- 가중치를 잘못 설정할 경우 기울기 소실 문제나 표현력의 한계를 갖는 등 여러 문제를 야기하게 된다.
- 또한 딥러닝의 학습의 문제가 non-convex 이기 때문에 초기값을 잘못 설정할 경우 local minimum에 수렴할 가능성이 커지게 된다.



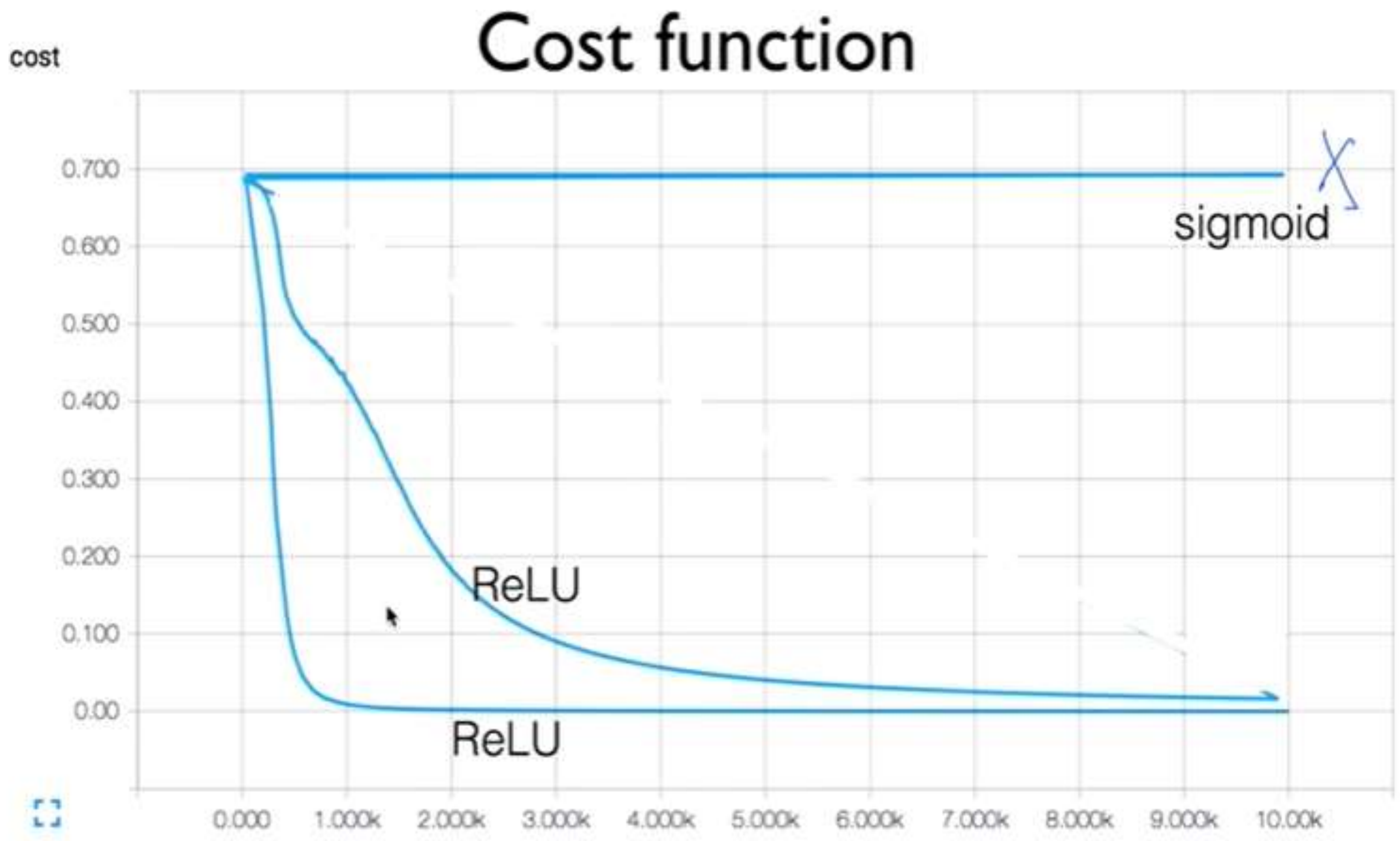
weight initialization

ex) sigmoid

- 표준 편차가 1일 경우, 값이 0과 1에 분포
- Gradient Vanishing 문제

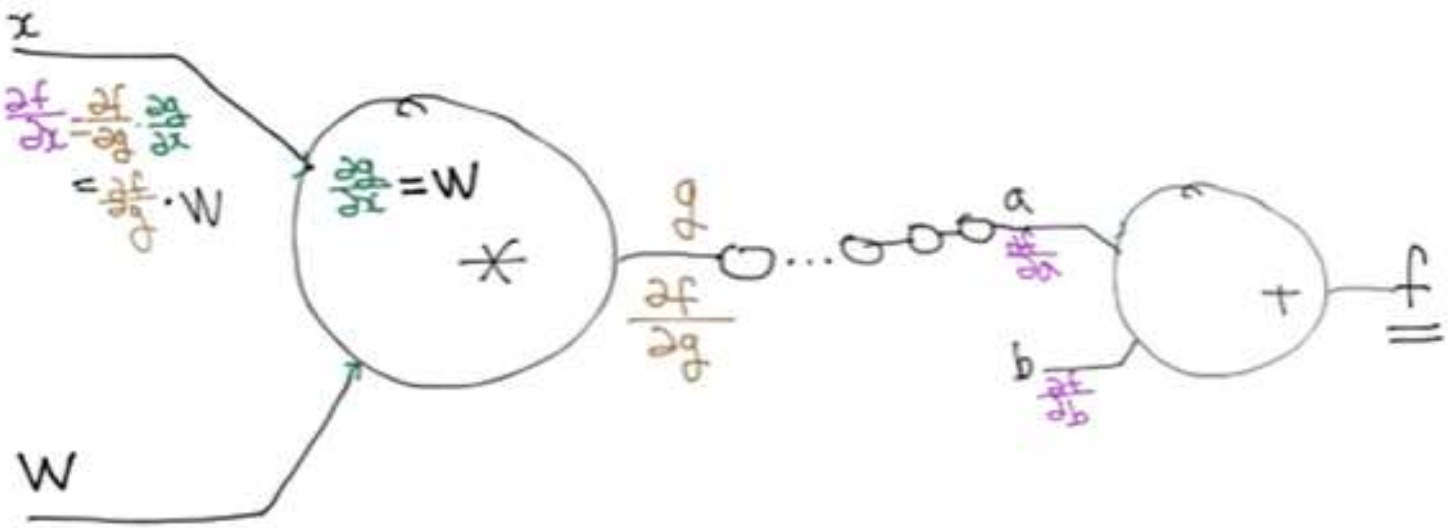


weight initialization



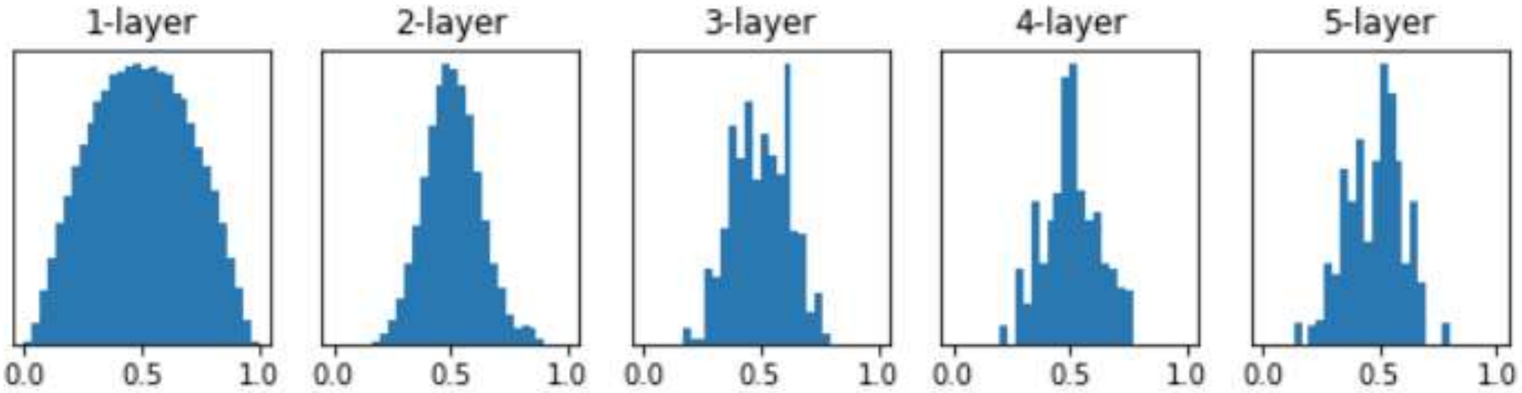
weight initialization

Set all initial weights to 0



weight initialization

- **1. Xavier Initialization**



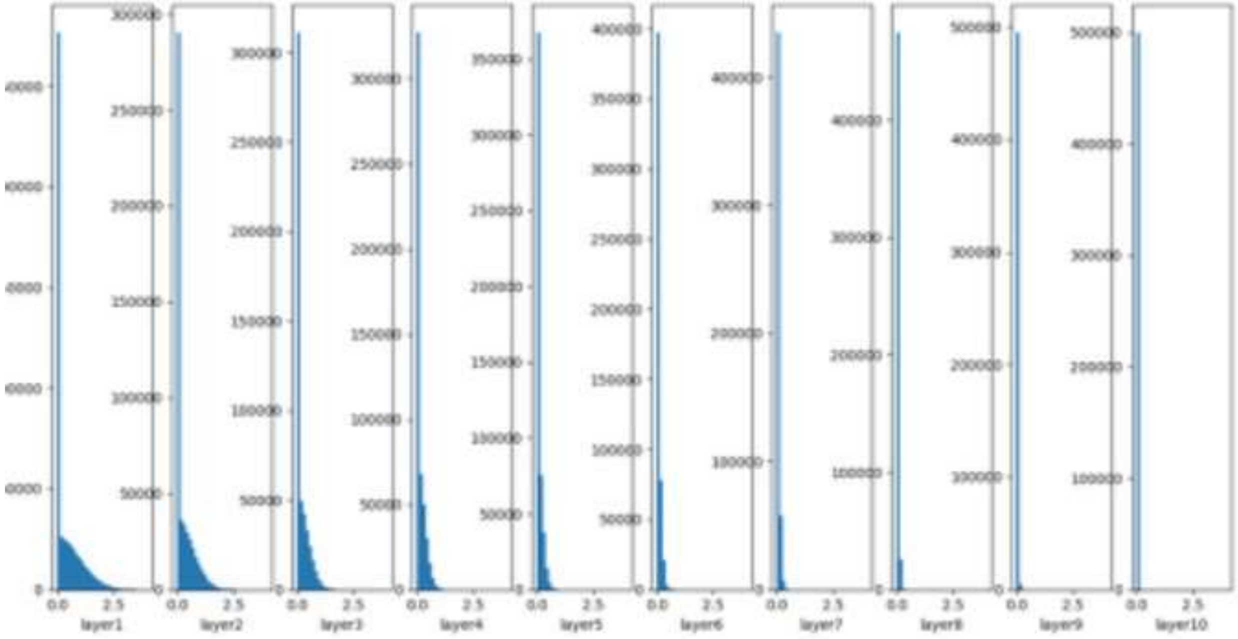
표준 정규 분포를 입력 개수의 표준편차로 나누기

np.random.randn(n_input, n_output)/sqrt(n_input)

Sigmoid 또는 tanh 함수 사용시 주로 사용

weight initialization

- 1. Xavier Initialization



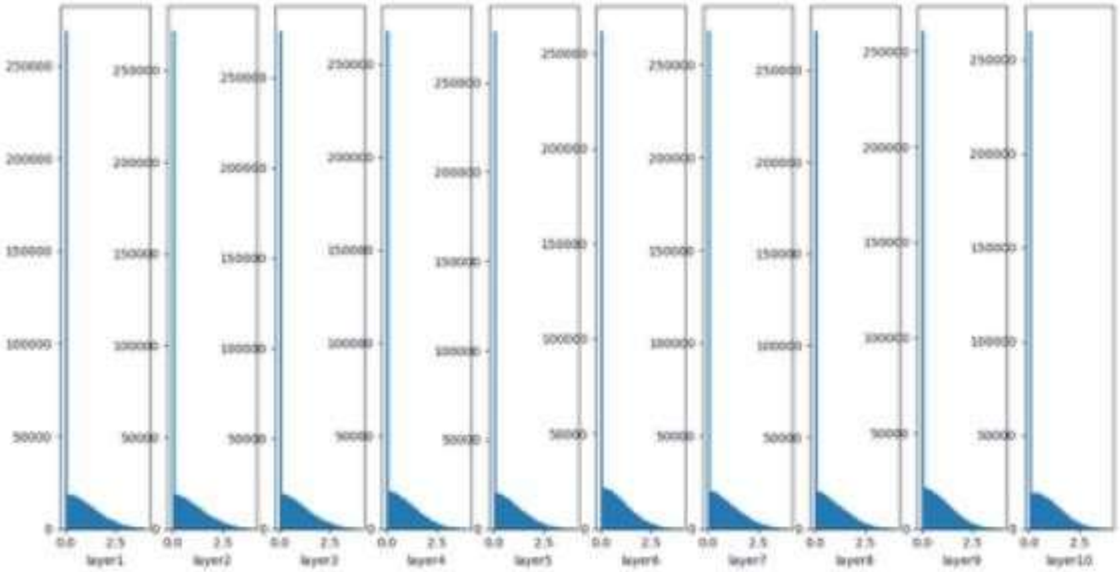
ReLU함수
+
Xavier Initialization

↓

점차 0으로 수렴

weight initialization

- **2.He Initialization**

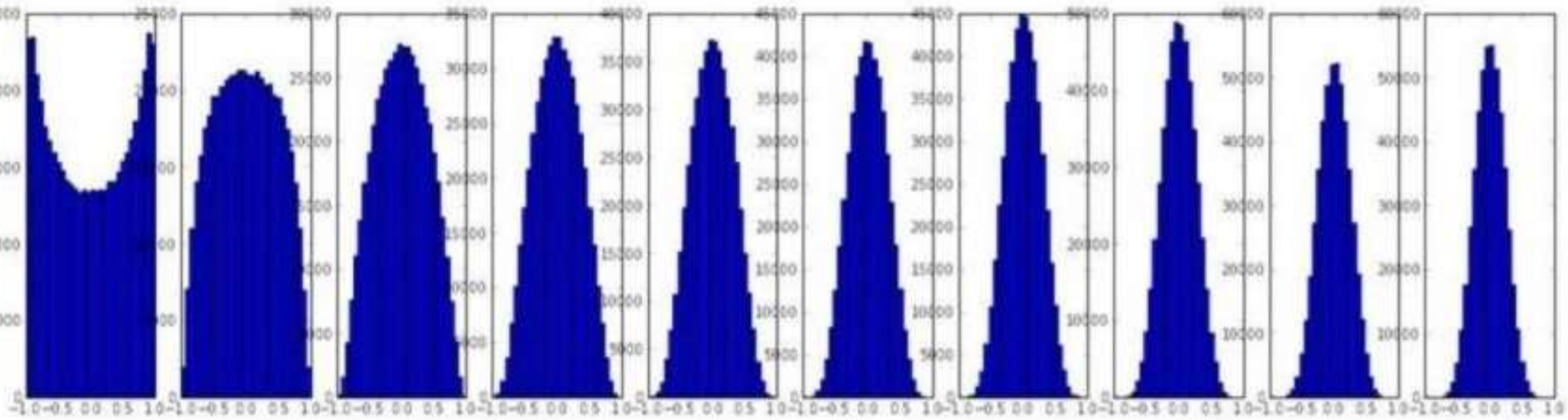


표준 정규 분포를 입력 개수의 절반의 제곱근으로 나누기
np.random.randn(n_input, n_output)/sqrt(n_input/2)

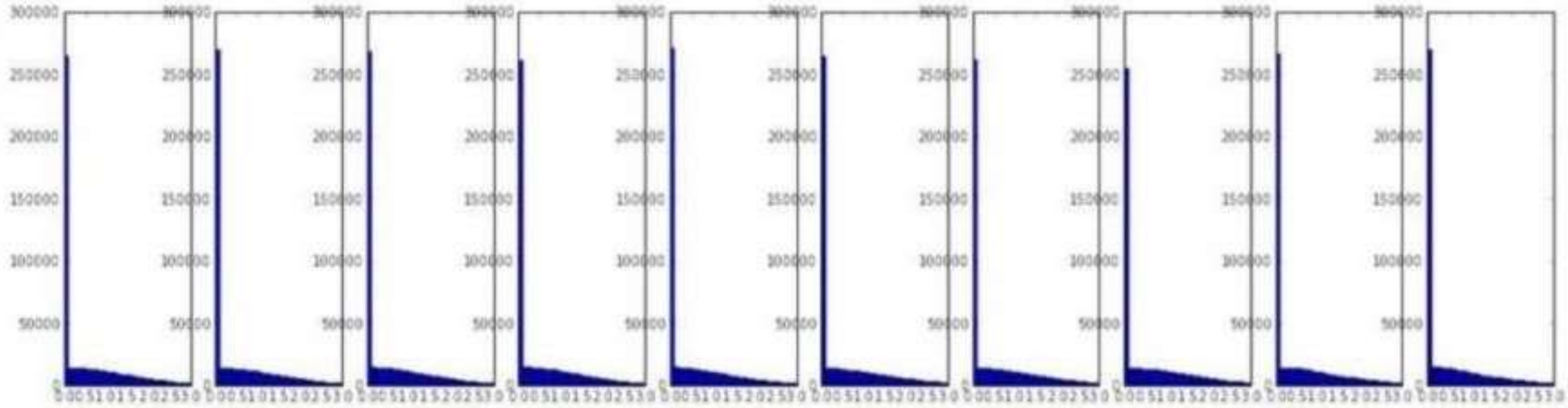
ReLu계열 함수에 사용

weight initialization

Xavier Initializer
나머지 sigmoid tanh



He Initializer
RELU family들



bias initialization

- 가중치 초기화 뿐만 아니라 편향(bias) 초기값 또한 초기값 설정 또한 중요하다.
- 보통의 경우에는 Bias는 0으로 초기화 하는 것이 일반적이다. ReLU의 경우 0.01과 같은 작은 값으로 b를 초기화 하는 것이 좋다는 보고도 있지만 모든 경우는 아니라 일반 적으로는 0 으로 초기화 하는 것이 효율적이다.

Loss Function

Loss Function

Suppose: 3 training examples, 3 classes.
With some W the scores $f(x, W) = Wx$ are:



cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**

Loss Function

$$L_i = \sum_{j \neq y_i} \begin{cases} 0 & \text{if } s_{y_i} \geq s_j + 1 \\ s_j - s_{y_i} + 1 & \text{otherwise} \end{cases}$$
$$= \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

Li 수식

Loss Function

Softmax Classifier (Multinomial Logistic Regression)

$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$

unnormalized probabilities

cat	**3.2**		**24.5**		**0.13**	→ L_i = -log(0.13) = **0.89**
car	5.1	exp →	164.0	normalize →	0.87	
frog	-1.7		0.18		0.00	

unnormalized log probabilities

probabilities

Loss Function

평균 제곱 오차
(Mean Square Error)

$$E = \frac{1}{2}\sum_{k}(y_k - t_k)^2$$

- 계산 간편해서 가장 많이 사용

교차 엔트로피 오차
(Cross Entropy Error)

$$E = -\sum_{k} t_k \log y_k$$

- 분류 문제에서 사용

Loss Function

그 외 손실함수

Probabilistic losses

- BinaryCrossentropy class
- CategoricalCrossentropy class
- SparseCategoricalCrossentropy class
- Poisson class
- binary_crossentropy function
- categorical_crossentropy function
- sparse_categorical_crossentropy function
- poisson function
- KLDivergence class
- kl_divergence function

Regression losses

- MeanSquaredError class
- MeanAbsoluteError class
- MeanAbsolutePercentageError class
- MeanSquaredLogarithmicError class
- CosineSimilarity class
- mean_squared_error function
- mean_absolute_error function
- mean_absolute_percentage_error function
- mean_squared_logarithmic_error function
- cosine_similarity function
- Huber class
- huber function
- LogCosh class
- log_cosh function

Hinge losses for "maximum-margin" classification

- Hinge class
- SquaredHinge class
- CategoricalHinge class
- hinge function
- squared_hinge function
- categorical_hinge function

Batch Normalization

Batch Normalization

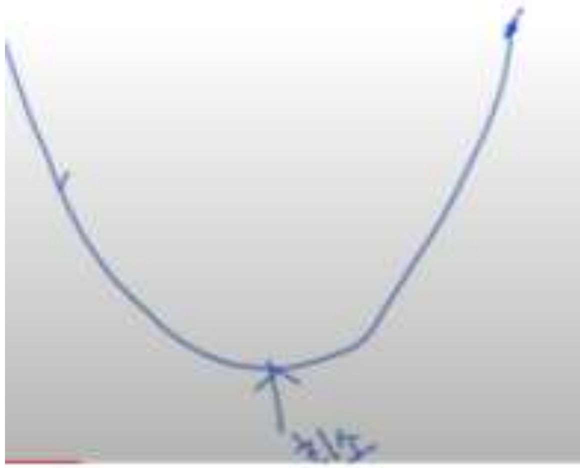
Cost function

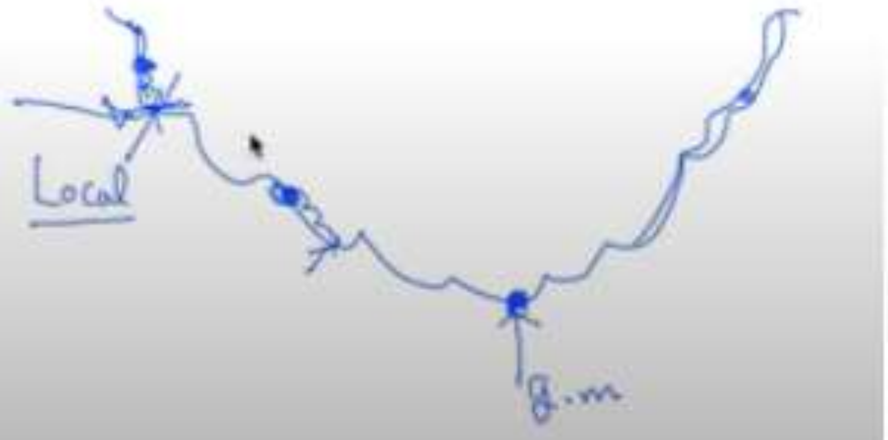
$$cost(W,b) = \frac{1}{m}\sum_{i=1}^{m}(H(x^{(i)}) - y^{(i)})^2$$

0< ~ <1

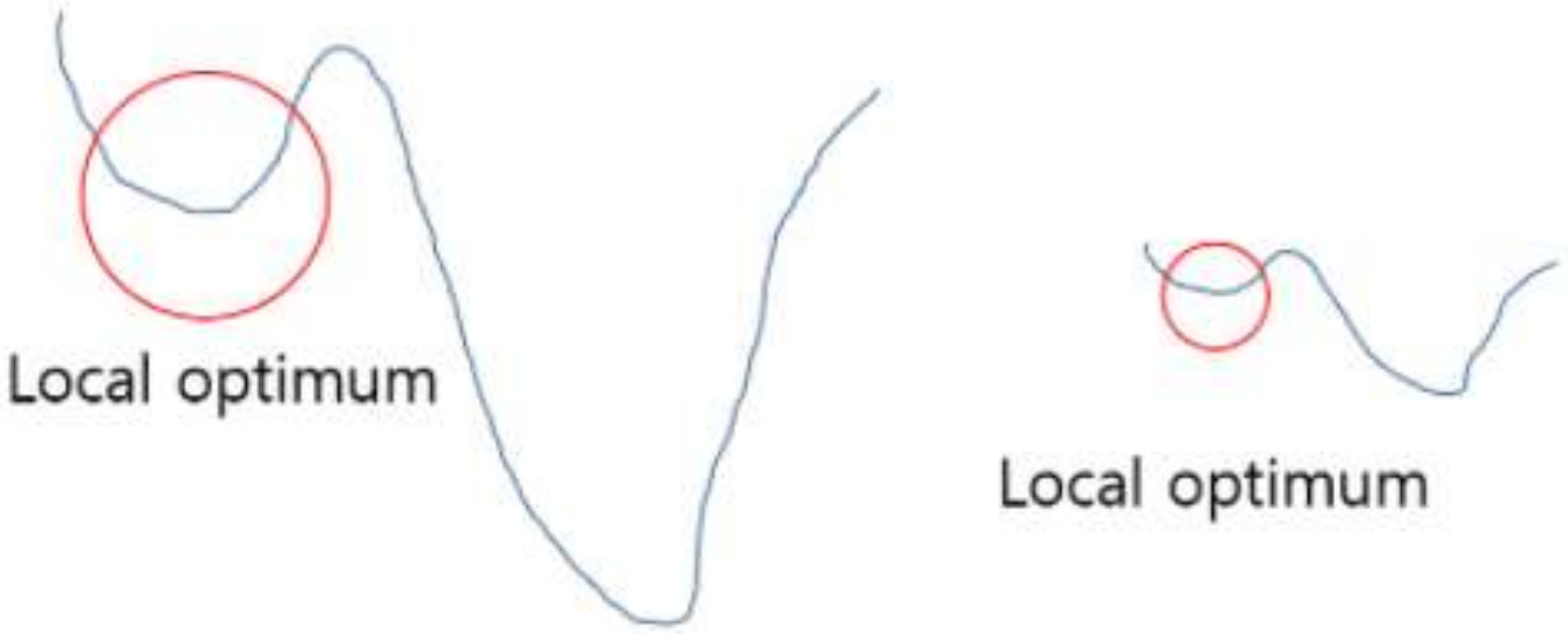
$$H(x) = Wx + b$$

$$H(X) = \frac{1}{1+e^{-W^T X}}$$





Batch Normalization

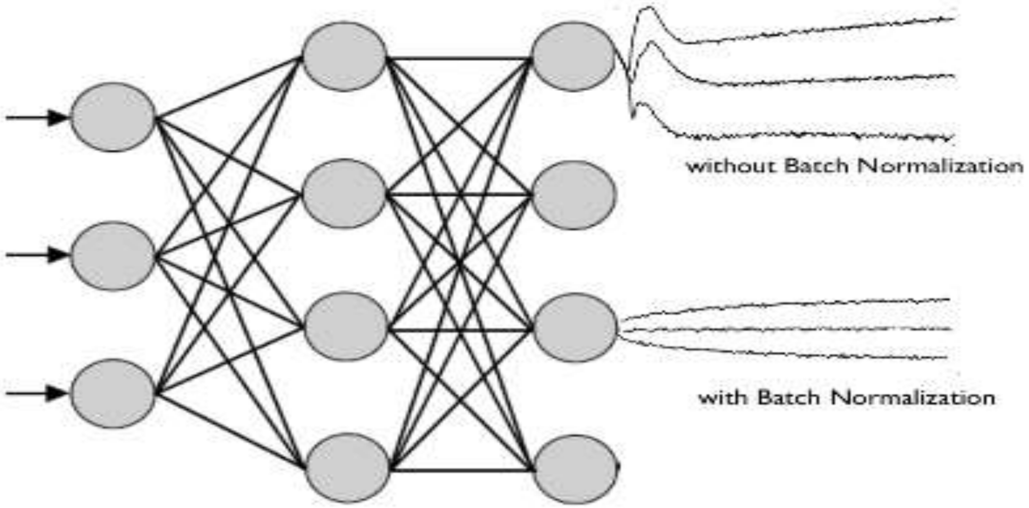


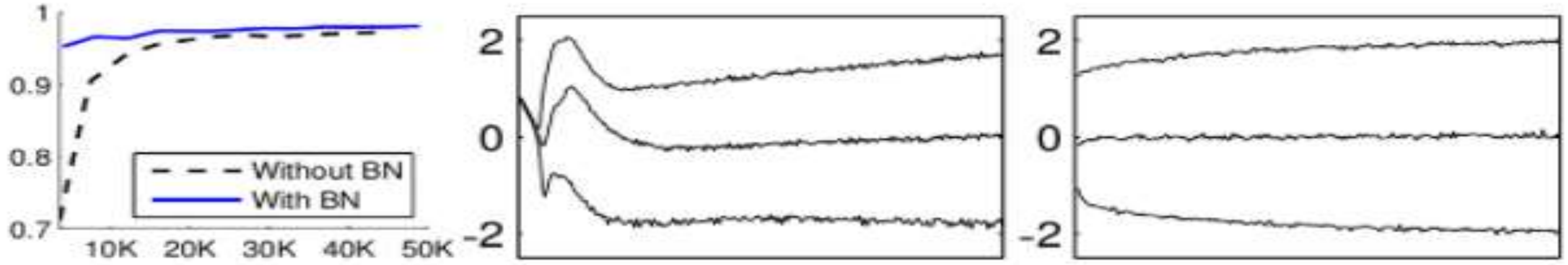
(좌) Normalization 적용 전 / (우) Normalization 적용 후

Batch Normalization

Covariate Shift : 이전 레이어의 파라미터 변화로 인하여 현재 레이어의 입력의 분포가 바뀌는 현상

Internal Covariate Shift : 레이어를 통과할 때 마다 Covariate Shift 가 일어나면서 입력의 분포가 약간씩 변하는 현상





Batch Normalization

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

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

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

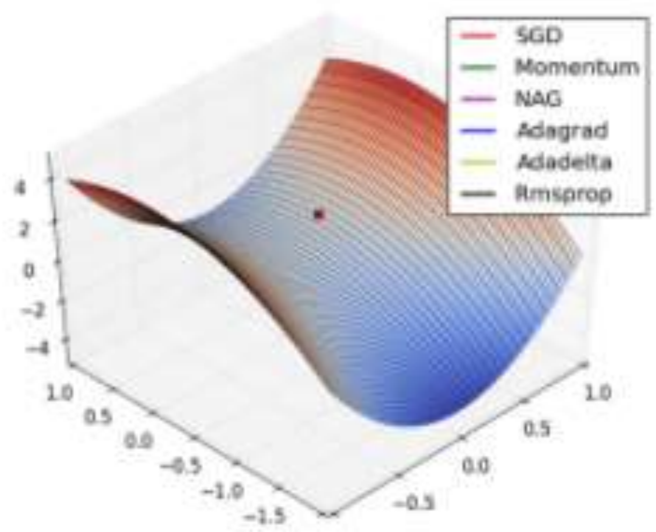
$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Optimizer

Optimizer

Training model = Optimization problem

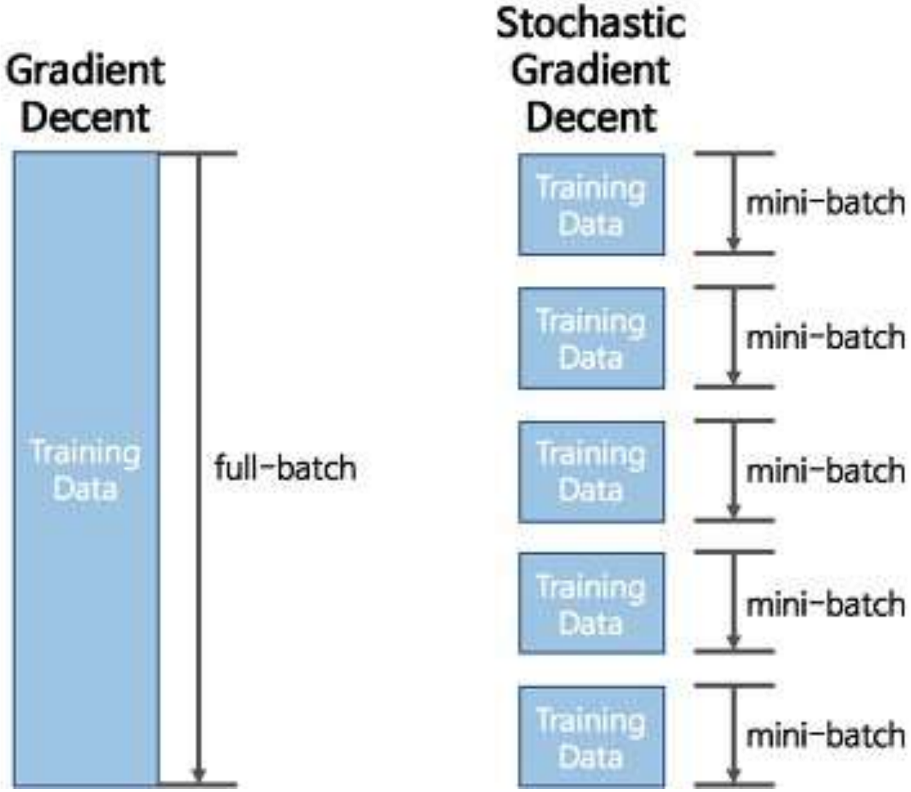
- Loss function이 보여주는 것 : weight(가중치)가 얼마나 잘 설정되어 있는가



weight의 업데이트 =	에러 낮추는 방향 (decent)	X	한발자국 크기 (learning rate)	X	현 지점의 기울기 (gradient)
$-\gamma\nabla F(\mathbf{a}^{\mathbf{n}})$	$-$		γ		$\nabla F(\mathbf{a}^{\mathbf{n}})$
			보폭		방향

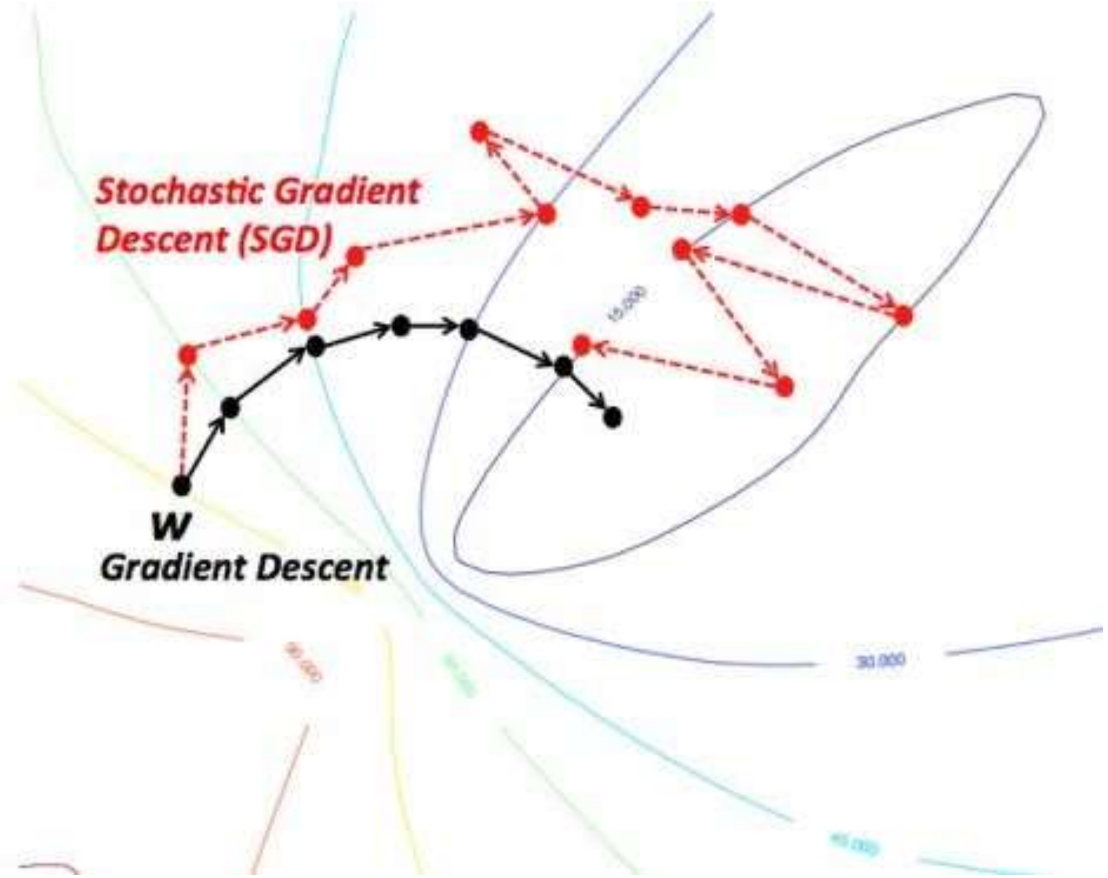
Optimizer

- **SGD(Stochastic Gradient Descent)**
- **조금만 훑어보고(Mini batch) 빠르게 가자!**

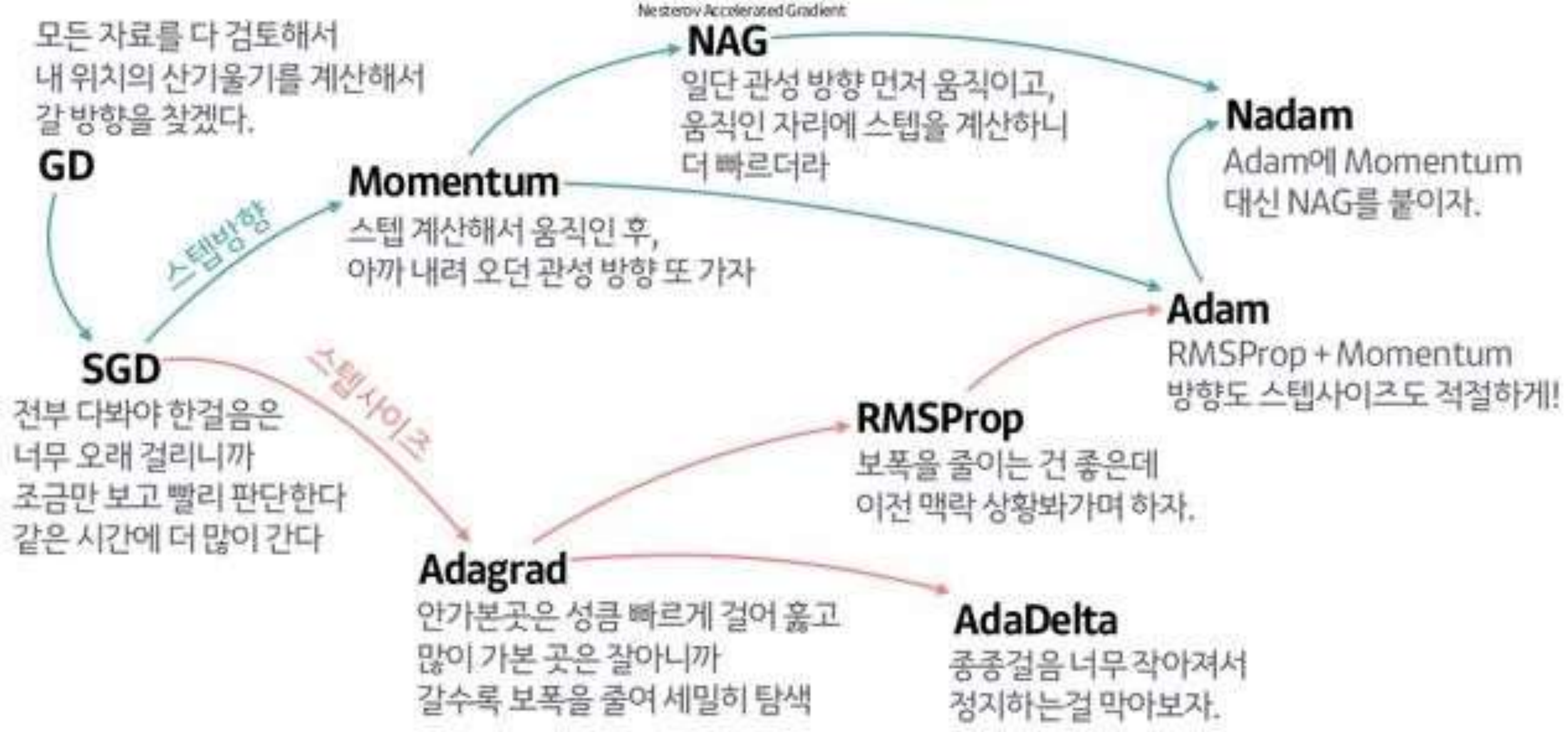


Optimizer

- **SGD(Stochastic Gradient Descent)**
- **조금만 훑어보고(Mini batch) 빠르게 가자!**



Optimizer

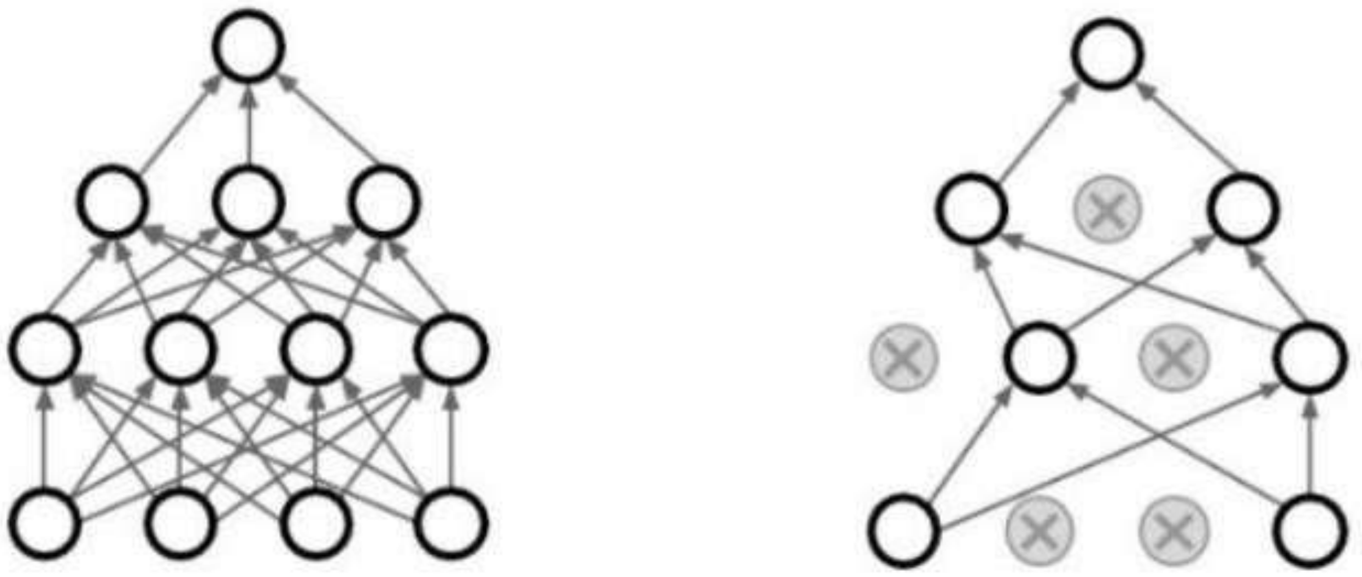

Nesterov Accelerated Gradient
NAG
일단 관성 방향 먼저 움직이고,
움직인 자리에 스텝을 계산하니
더 빠르더라
모든 자료를 다 검토해서
내 위치의 산기울기를 계산해서
갈 방향을 찾겠다.
GD
Momentum
스텝 계산해서 움직인 후,
아까 내려 오던 관성 방향 또 가자
스텝방향
Nadam
Adam에 Momentum
대신 NAG를 붙이자.
Adam
RMSProp + Momentum
방향도 스텝사이즈도 적절하게!
SGD
전부 다봐야 한걸음은
너무 오래 걸리니까
조금만 보고 빨리 판단한다
같은 시간에 더 많이 간다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

Regularizer - Dropout

오버피팅

Regularizer - Dropout

Drop Out : 일정 노드를 랜덤하게 버려 버리기 Forward pass시에 일부 뉴런의 출력을 0으로 만들어 버림 (할 때마다 0이 되는 뉴런은 바껴!)



Regularizer - Dropout

Dropout이 왜 좋은데?

- 뉴런 하나당 하나의 특성만을 학습하는데 랜덤으로 노드를 죽이게 되면 살아 남은 노드들이하나의 노드에만 의존 하지 않고, 다른 특징까지 학습하려 하면 서 overfitting을 막을 수 있음
- - forward pass 시마다 랜덤하게 dropout 시키기 때문에 앙상블의 효과를 볼 수 있음