



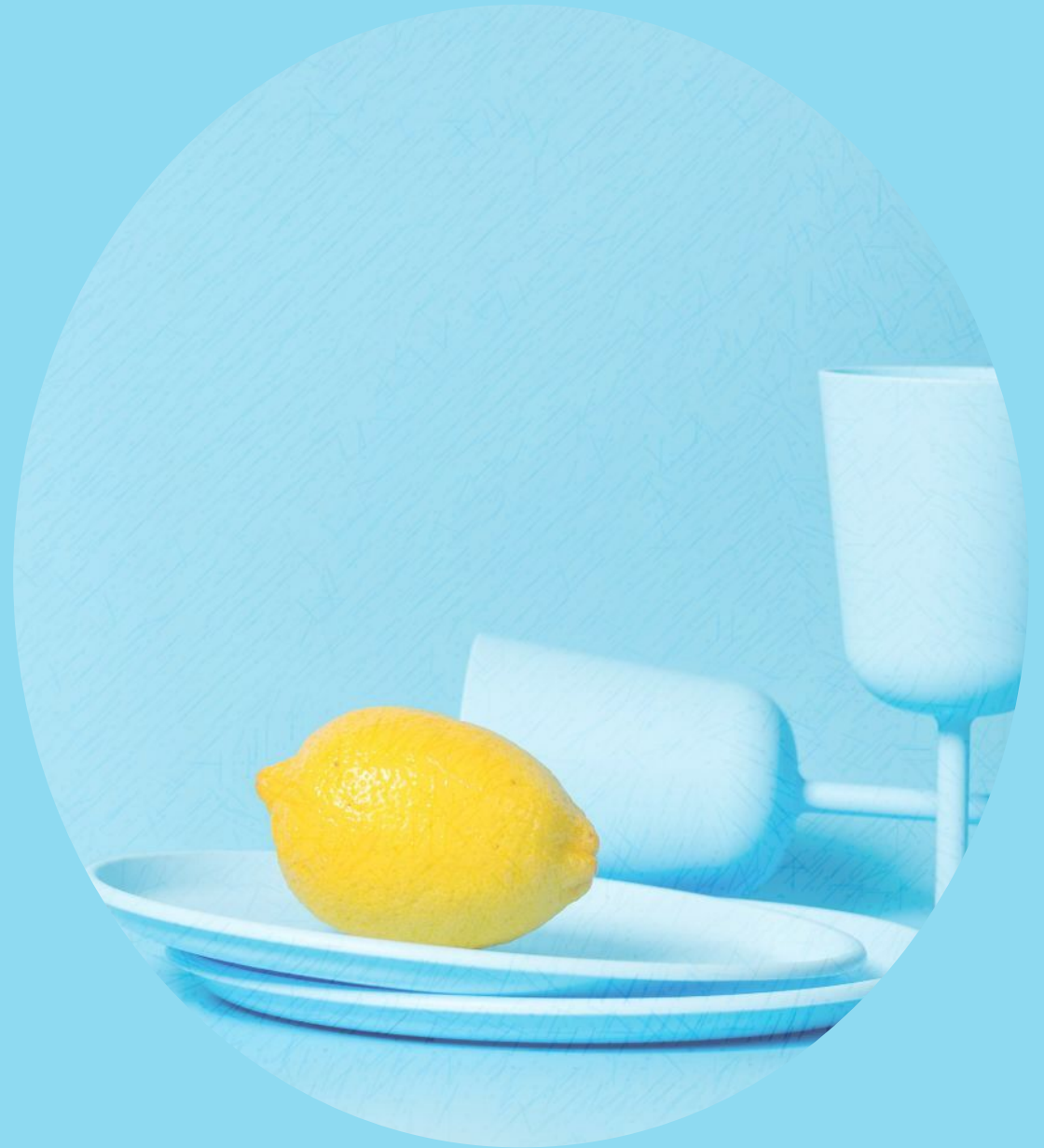
모두의 딥러닝 4팀

신경망의 이해



목차

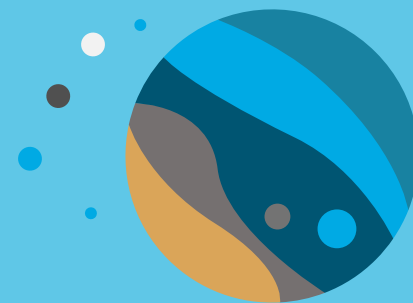
- 1 퍼셉트론과 다층 퍼셉트론
- 2 오차 역전파의 개념
- 3 심화 학습 - 오차 업데이트
- 4 고급 경사 하강법





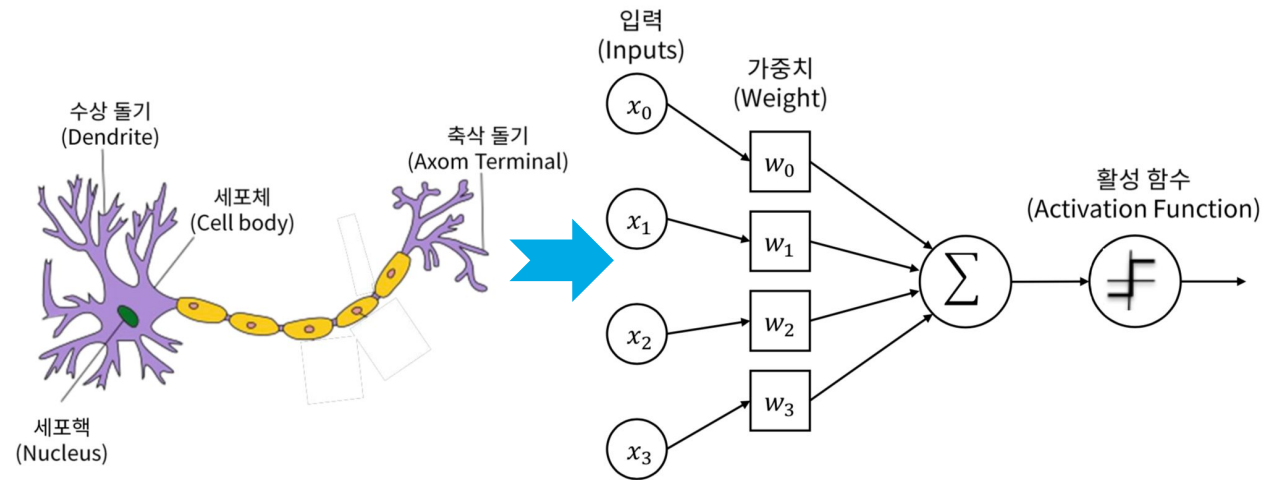
Part 1

퍼셉트론과 다층 퍼셉트론



퍼셉트론

뉴런과 유사하게 신경망의 기본 구조 또한



**여러 층의 퍼셉트론을 서로 연결 시키고 복잡하게 조합하여
주어진 입력 값에 대한 판단을 하게하는 것**

가중치, 가중합, 바이어스, 활성화 함수

$$y = ax + b \rightarrow y = wx + b$$

a : 기울기, b = y 절편

w : 가중치, b = 바이어스

가중치(w) : 다음 노드로 넘길 때 적용하는 중요도

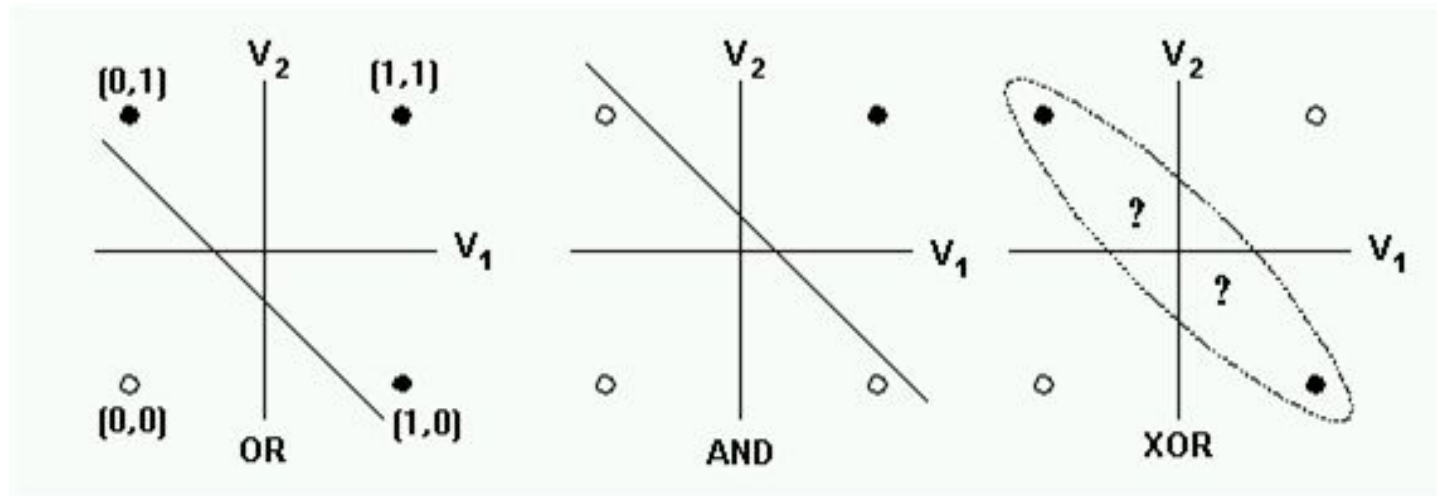
바이어스(b) : 뉴런의 활성화 정도를 조절하는 매개변수

$$y = wx + b$$

이 식의 y 값 = 가중합

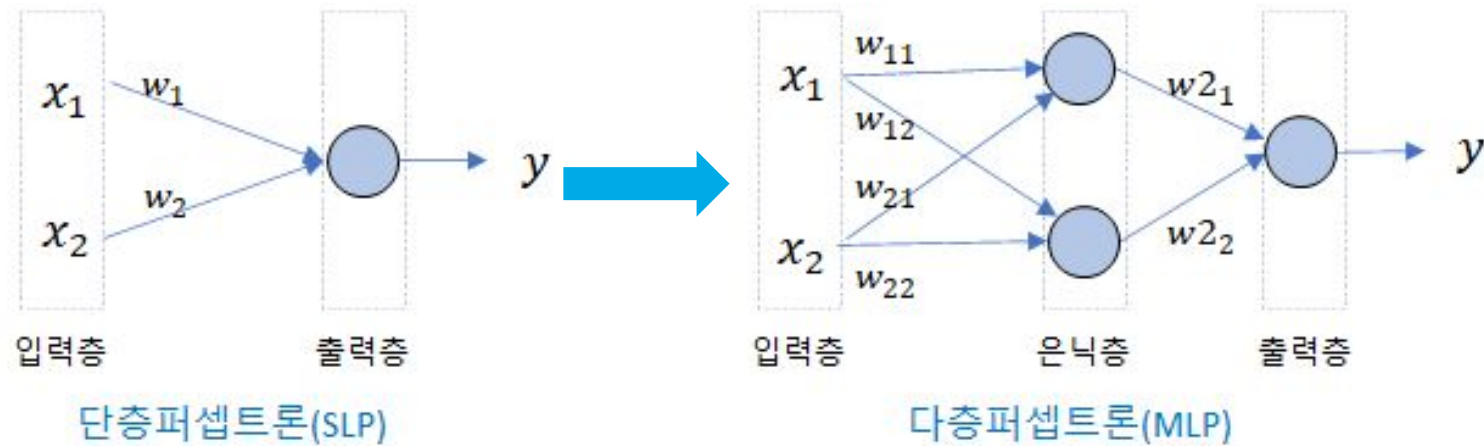
활성화 함수 : 가중합 결과를 1 또는 0으로 판단하는 함수

퍼셉트론의 한계



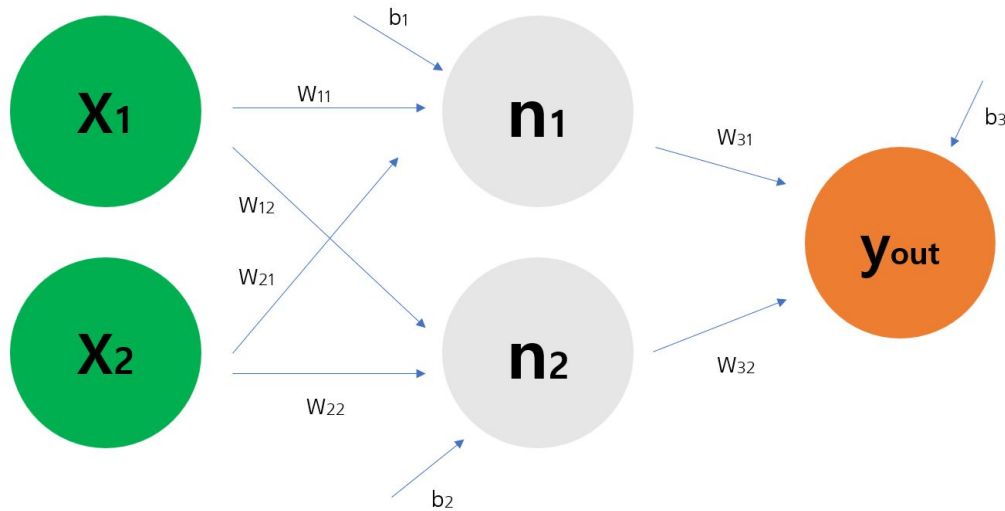
AND게이트와 OR게이트는 한 직선으로 흰점과 검은점을 구별할 수 있지만,
XOR 게이트는 불가능하다.

다층 퍼셉트론



2차원 평면에서 해결되지 않은 문제들이 3차원에서는 해결되는 것처럼
(단층)퍼셉트론이 해결하지 못하는 문제들은 **다층 퍼셉트론**이 해결할 수 있게됨

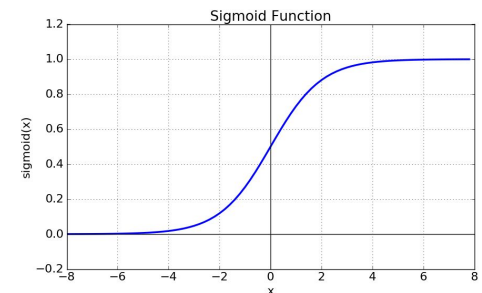
다층 퍼셉트론의 구조



1 퍼셉트론이 입력받은 값과 각각 자신의 가중치(W), 바이어스(b)를 **은닉층**으로 보낸다.

2 은닉층에 모인 값은 **시그모이드 함수**를 이용해서 최종값을 도출하고 결과로 보낸다

$$y = \frac{1}{1 + e^{-(ax+b)}}$$



XOR 문제 해결 - 코딩으로 구현

1

```
import numpy as np
# 주어진 가중치를 2차원 배열로 입력
w11 = np.array([-2,-2])
w12 = np.array([2,2])
w2 = np.array([1,1])

#주어진 바이어스를 입력
b1 = 3
b2= -1
b3 = -1
```

2

```
#퍼셉트론 함수
def MLP(x,w,b):
    #입력값과 2차원 배열 형식으로 되어있는 가중치의 곱에 바이어스를 더함
    y= np.sum(w*x) + b

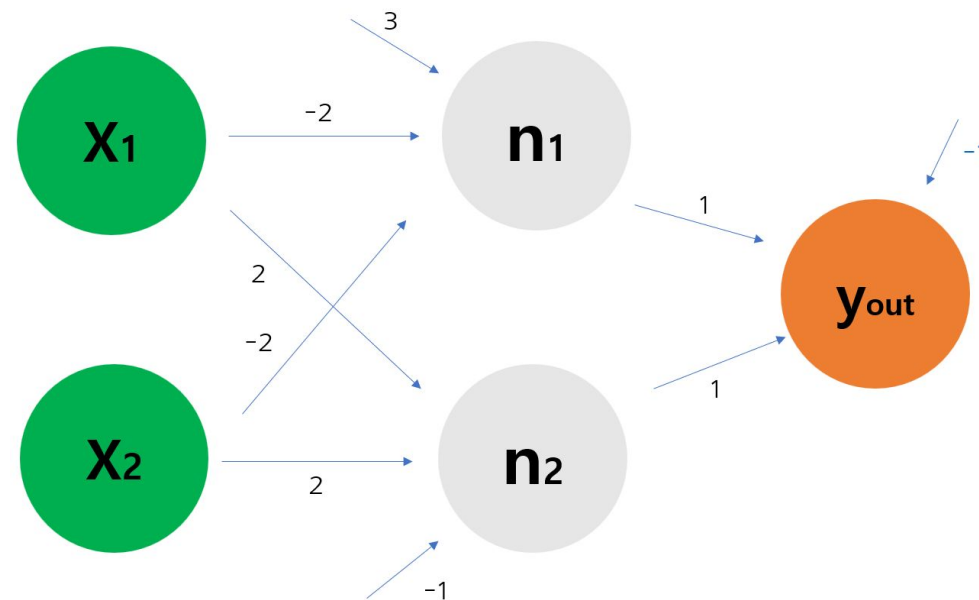
    if y<=0 :
        return 0
    else :
        return 1
```

XOR 문제 해결

가중치와 바이어스 값을 미리 정해둠

$$W(1) = \begin{bmatrix} -2 & 2 \\ -2 & 2 \end{bmatrix} \quad B(1) = \begin{bmatrix} 3 \\ -1 \end{bmatrix}$$

$$W(2) = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \quad B(2) = [-1]$$



가중치와 바이어스 값을 대입한 구조

XOR 문제 해결 - 코딩으로 구현

3

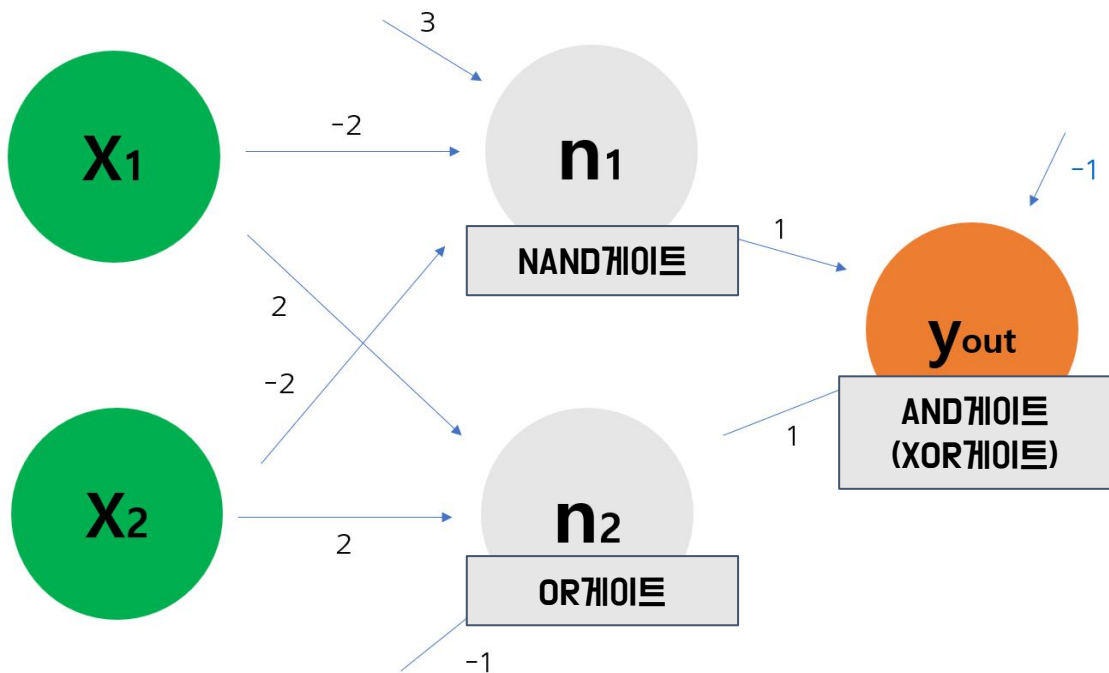


```
def NAND(x1,x2) : #AND게이트와 반댓값 출력
    return MLP(np.array([x1,x2]), w11, b1)

def OR (x1, x2):
    return MLP(np.array([x1,x2]),w12, b2)

def AND (x1, x2):
    return MLP(np.array([x1,x2]),w2, b3)

def XOR (x1, x2):
    return AND(NAND(x1, x2), OR(x1,x2))
```



XOR 문제 해결 - 코딩으로 구현

4

```
if __name__ == '__main__': # 메인함수 선언
    for x in [(0,0), (1,0), (0,1), (1,1)] :
        y = XOR(x[0],x[1])
        print("입력 값:" + str(x) + "출력 값 : " + str(y))
```

```
입력 값:(0, 0)출력 값 :0
입력 값:(1, 0)출력 값 :1
입력 값:(0, 1)출력 값 :1
입력 값:(1, 1)출력 값 :0
```

Tf로 은닉층 설계 (실습문제!)



4. 경사하강법 실습문제

```
model = Sequential()  
model.add(Dense(1, input_dim = 2, activation = 'linear'))
```

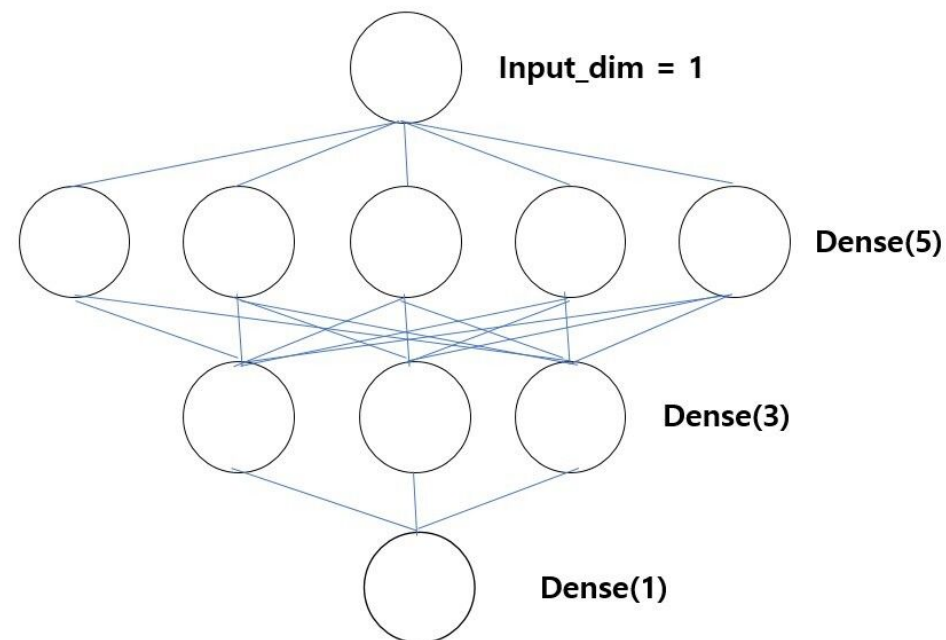
첫번째 숫자 : 노드 수

Input_dim : 입력 노드 수

activation : 활성화 함수 종류

EX

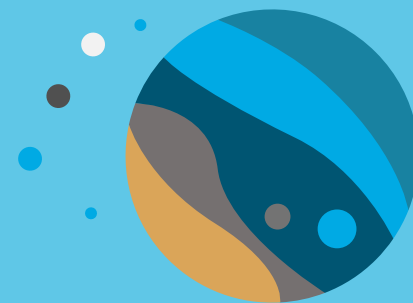
```
model = Sequential()  
model.add(Dense(5, input_dim=1, activation='sigmoid'))  
model.add(Dense(3, activation='sigmoid'))  
model.add(Dense(1, activation='sigmoid'))
```



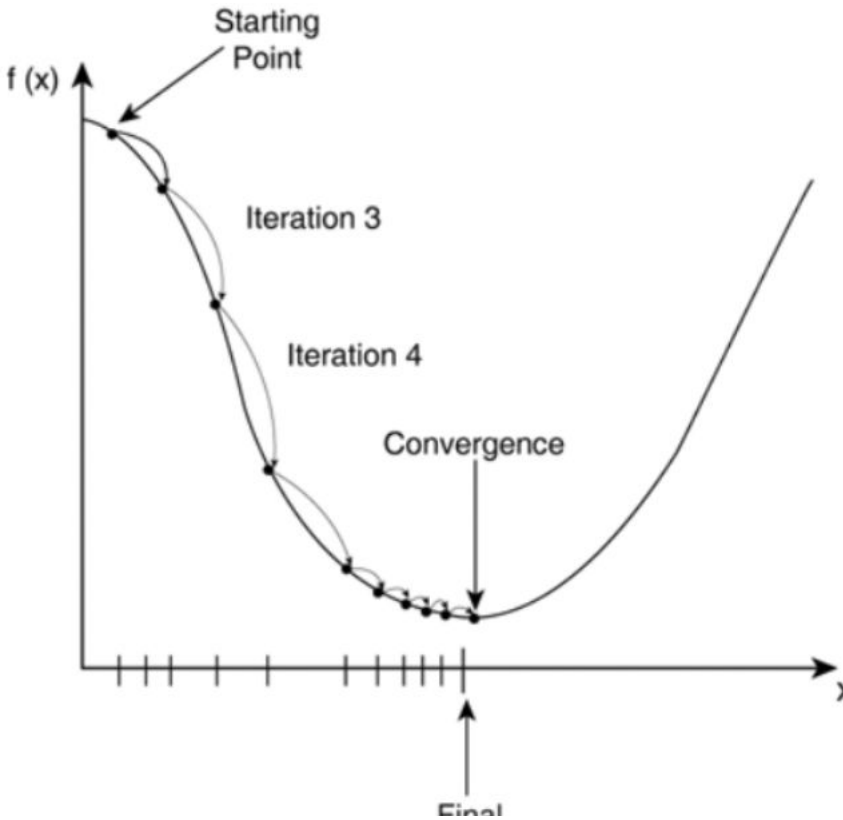


Part 2

오차 역전파의 개념



경사 하강법과의 유사점



[경사 하강법]

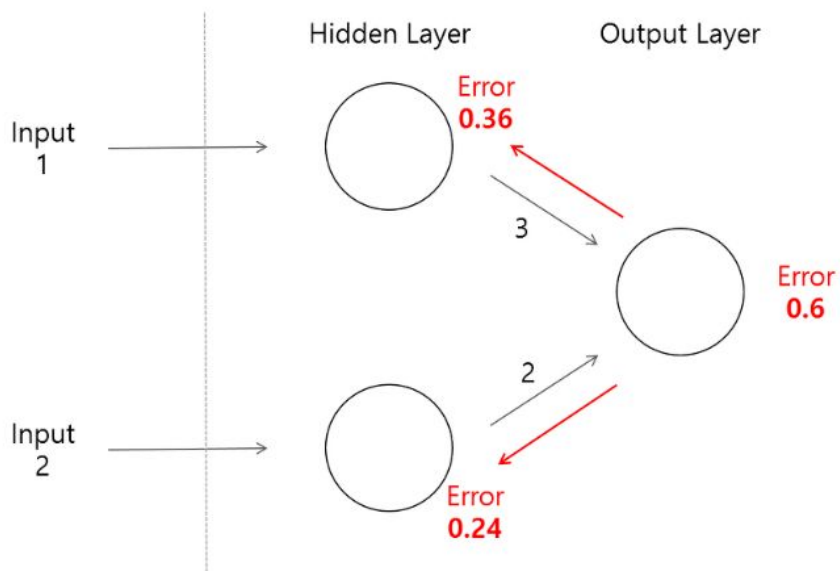
1. 임의의 직선을 설정
2. 실제값과 예측값 사이 오차 측정
3. 오차가 작은 지점으로 이동시킴
4. 기울기와 y절편을 업데이트



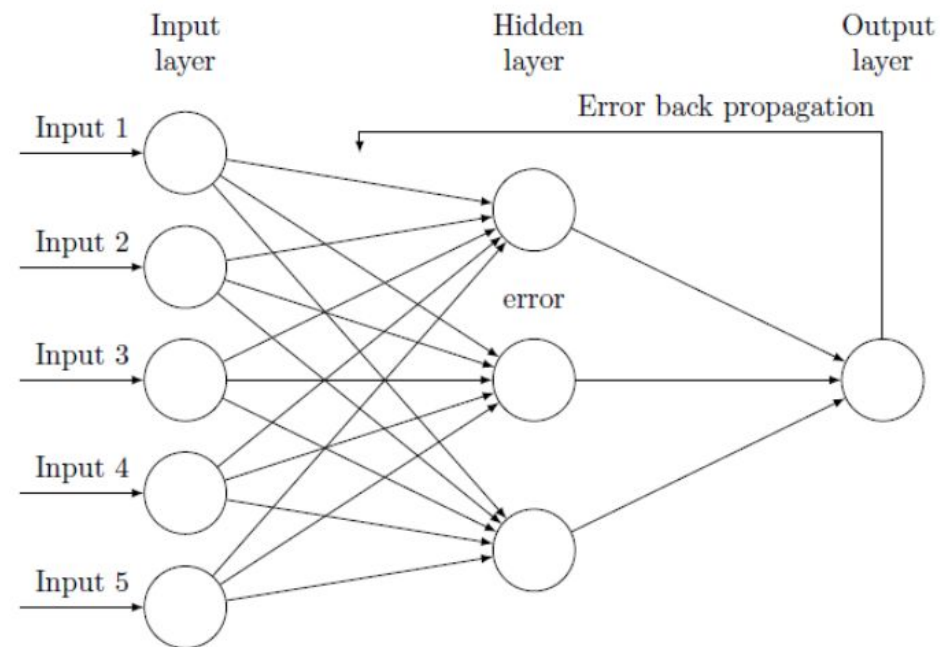
XOR 문제를 해결할 때는 임의의 값을 지정해줌.

이에 대해 가중치와 바이어스에 대한 오차를 확인하고
최소가 되는 지점까지 수정이 필요

경사 하강법과의 차이점



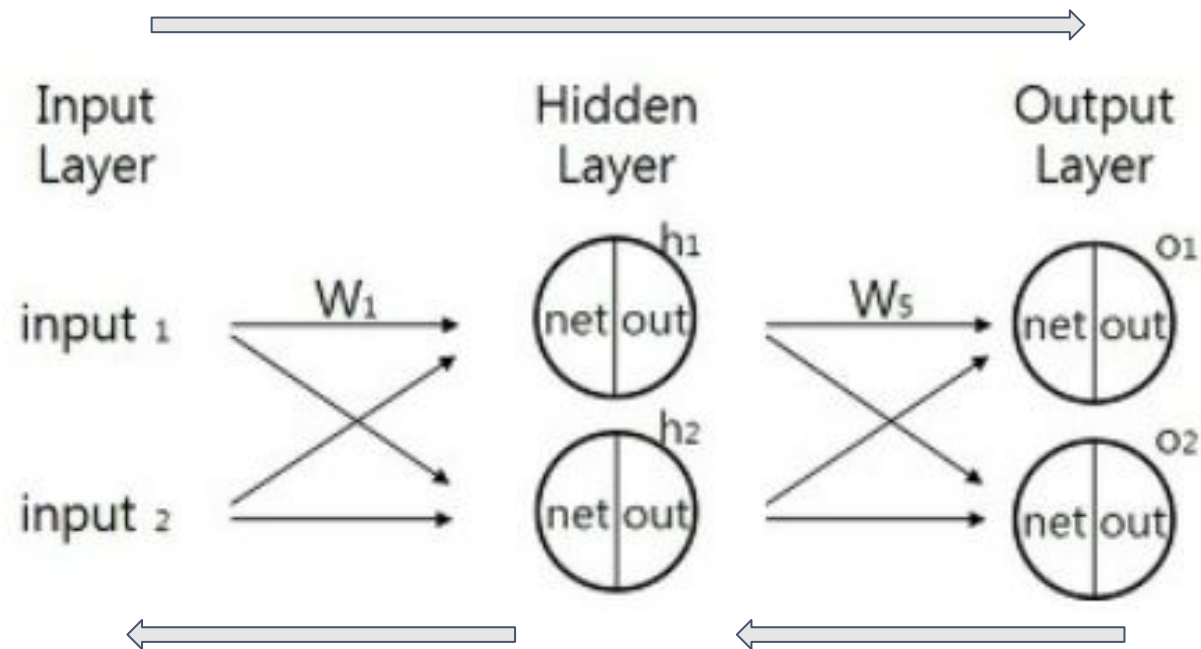
단일 퍼셉트론 - 경사 하강법



다층 퍼셉트론 - 오차 역전파

오차 역전파의 구동 방식

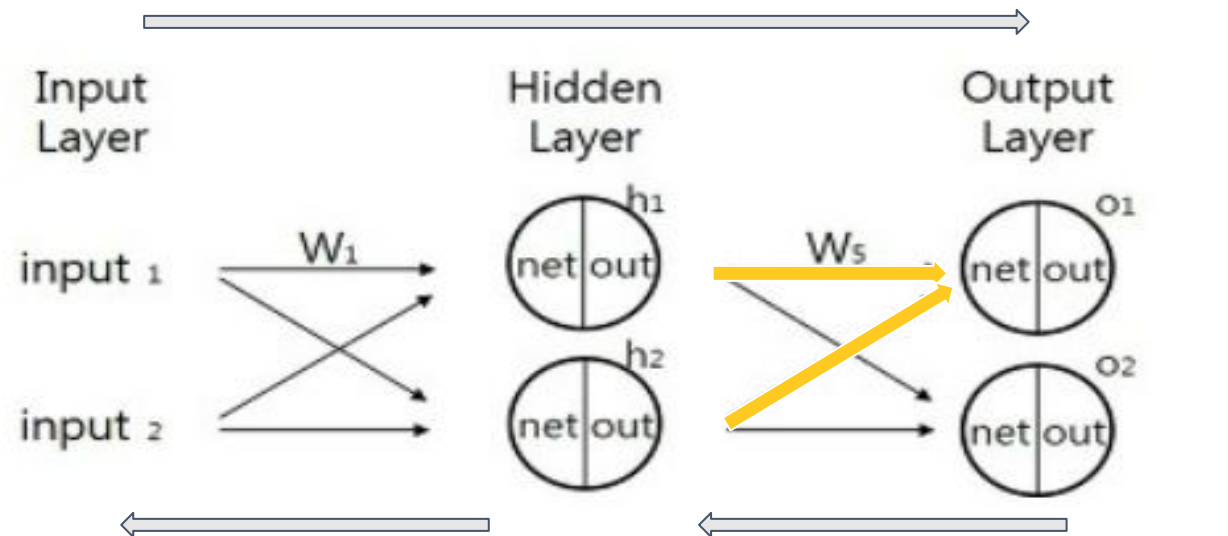
1. 순전파



역전파 2 : 은닉층 오차 업데이트

역전파 1 : 출력층 오차 업데이트

오차 역전파의 구동 방식



역전파 2 : 은닉층 오차
업데이트

역전파 1 : 출력층 오차 업데이트

수식

수식

$$(\delta y_{o1} \cdot W_{31} + \delta y_{o2} \cdot W_{41}) y_{h1} (1 - y_{h1}) \cdot x_1$$

$$(y_{o1} - y_{\text{실제 값}}) \cdot y_{o1} (1 - y_{o1}) \cdot y_{h1}$$

오차 역전파 계산법

오차 역전파의 목표 !

-> 오차가 작아지는 방향으로 업데이트하기

-> 수학적으로 표현하면 x축이 가중치, y축이 오차인 그래프의 기울기가 0이 되는 부분의 가중치!

$$W(t+1) = W_t - \frac{\partial \text{오차}}{\partial W}$$

오차 역전파 계산법

오차 공식 -> **평균 제곱 오차** 이용!

$$\begin{aligned} \text{오차 } y_{01} &= \frac{1}{2}(y_{t1} - y_{01})^2 \\ \text{오차 } y_{02} &= \frac{1}{2}(y_{t2} - y_{02})^2 \\ \Rightarrow \text{오차 } y_{out} &= \frac{1}{2}(y_{t1} - y_{01})^2 + \frac{1}{2}(y_{t2} - y_{02})^2 \end{aligned}$$

출력값 : y_{01}, y_{02}

실제값 : y_{t1}, y_{t2}

-> **전체 오차**

오차 역전파 계산법

$$W(t+1) = W_t -$$

1. 오차 공식 활용
2. 시그모이드 함수

$$\begin{aligned} \frac{d}{dx} \sigma(x) &= \frac{d}{dx} \left[\frac{1}{1+e^{-x}} \right] \\ &= [1+e^{-x}]^{-1} \\ &= \frac{1}{1+e^{-x}} \cdot \left(1 - \frac{1}{1+e^{-x}}\right) \\ &= \sigma(x) \cdot (1 - \sigma(x)) \end{aligned}$$

우선 각각의 층에서는
입력값을 넣어 가중합을
만드는 단계와 활성화 함수를
통해 출력값을 내는 단계로
이루어집니다.

우리가 구해야 하는 식!!

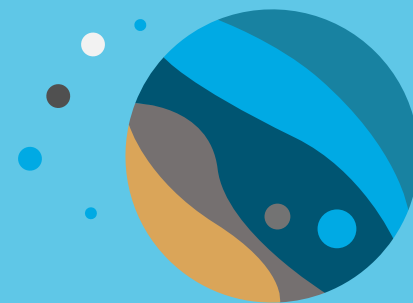
$$\begin{aligned} \frac{\partial \text{err}_{Y_{out}}}{\partial w_1} &= \frac{\partial \text{err}_{Y_{out}}}{\partial y_{01}} \cdot \frac{\partial y_{01}}{\partial \text{가중합}} \cdot \frac{\partial \text{가중합}}{\partial w_1} \\ &\quad \downarrow \mathbf{1} \qquad \qquad \downarrow \mathbf{2} \qquad \qquad \downarrow \mathbf{3} \\ &= (y_{01} - y_{t1}) \cdot y_{01}(1 - y_{01}) \cdot y_{h1} \end{aligned}$$

3. 가중합 공식 $\rightarrow w*y_1 + w*y_2 + b$ 이용



Part 3

심화학습 - 오차 업데이트



가중치 업데이트

출력층 $\rightarrow w_{31}(t+1) = w_{31}(t) - \frac{\delta^2 \text{차 } Y_{out}}{\delta w_{31}}$

체인룰 \downarrow

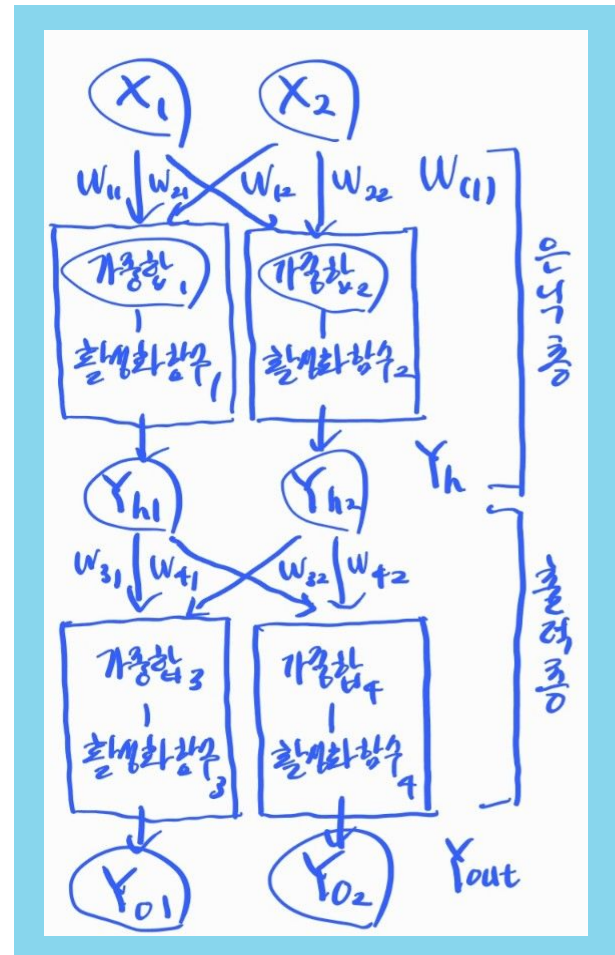
$$= \frac{\delta^2 \text{차 } Y_{out}}{\delta y_{01}} \cdot \frac{\delta y_{01}}{\delta \text{가중합}_3} \cdot \frac{\delta \text{가중합}_3}{\delta w_{31}}$$

\downarrow \downarrow \downarrow
 $= (y_{01} - y_{t1}) \quad y_{01}(1 - y_{01}) \quad y_{h1}$

$$w_{31}(t+1) = w_{31}(t) - \boxed{(y_{01} - y_{t1}) \cdot y_{01}(1 - y_{01}) \cdot y_{h1}}$$

\therefore 다음 업데이트 때도 변복래미 나타날
 \Rightarrow 델타식^{*} (δy) 라 함

$$= w_{31}(t) - \delta y \cdot y_{h1}$$



은닉층 오차 업데이트하기

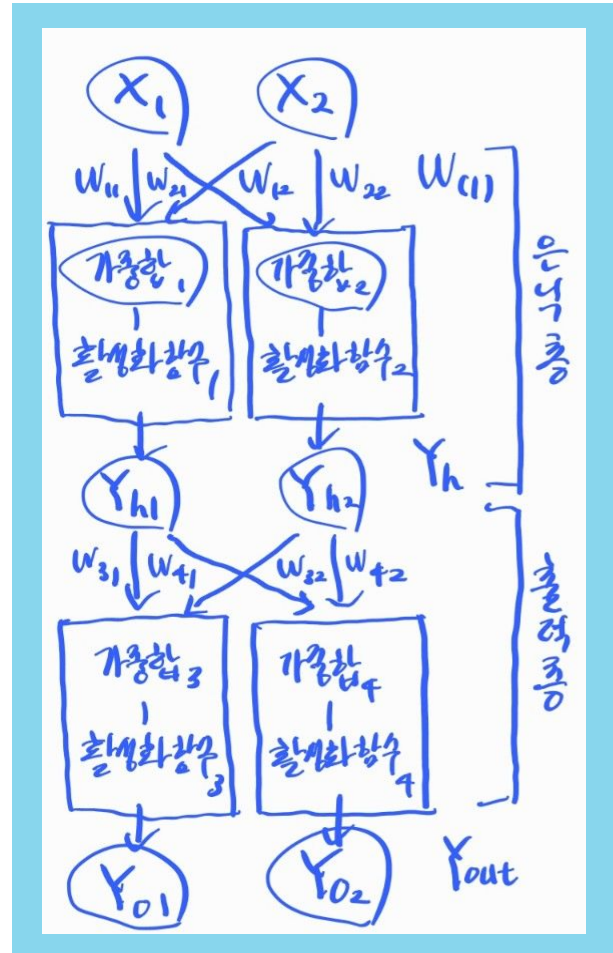
$$\begin{aligned} \text{출력층} \rightarrow w_{31}(t+1) &= w_{31}(t) - \frac{\delta \text{오차 } Y_{out}}{\delta w_{31}} \\ &= w_{31}(t) - (y_{o1} - y_{t1}) \cdot y_{h1} \cdot (1 - y_{h1}) \cdot y_{h1} \end{aligned}$$

$$\begin{aligned} \text{은닉층} \rightarrow w_{11}(t+1) &= w_{11}(t) - \frac{\delta \text{오차 } Y_{out}}{\delta w_{11}} \\ &= w_{11}(t) - (\delta y_{o1} \cdot y_{o1} + \delta y_{o2} \cdot y_{o2}) \cdot y_{h1} \cdot (1 - y_{h1}) \cdot x_1 \end{aligned}$$

$$\frac{\delta \text{오차 } Y_{out}}{\delta w_{11}} = \underbrace{\frac{\delta \text{오차 } Y_{out}}{\delta y_{h1}}}_{\text{계산결과 동일}} \cdot \underbrace{\frac{\delta y_{h1}}{\delta \text{가중합}_1}}_{\text{계산결과 동일}} \cdot \frac{\delta \text{가중합}_1}{\delta w_{11}}$$

$$\frac{\delta \text{오차 } Y_{out}}{\delta y_{h1}} = \delta (y_{o1} + y_{o2}) = \frac{\delta \text{오차 } y_{o1}}{\delta y_{h1}} + \frac{\delta \text{오차 } y_{o2}}{\delta y_{h1}}$$

$$\frac{\delta \text{오차 } Y_{out}}{\delta y_{h1}} = \underbrace{\frac{\delta \text{오차 } y_{o1}}{\delta y_{h1}}}_a + \underbrace{\frac{\delta \text{오차 } y_{o2}}{\delta y_{h1}}}_b \rightarrow y_{h1} \text{이 아닌 } y_{h1} \text{에 대해 미분} \rightarrow \text{들려와야 해 X}$$



은닉층 오차 업데이트하기

$$a) \frac{\partial 오차_{y_{o1}}}{\partial y_{h1}} = \frac{\partial 오차_{y_{o1}}}{\partial 가중합_3} \cdot \frac{\partial 가중합_3}{\partial y_{h1}}$$

$$a-1) - \frac{\partial 오차_{y_{o1}}}{\partial 가중합_3} = \frac{\partial 오차_{y_{o1}}}{\partial y_{o1}} \cdot \frac{\partial y_{o1}}{\partial 가중합_3}$$

오차 $y_o = \frac{1}{2} (y_{ti} - y_{oi})^2$ 편미분.

$$\therefore y_{oi} - y_{ti}$$

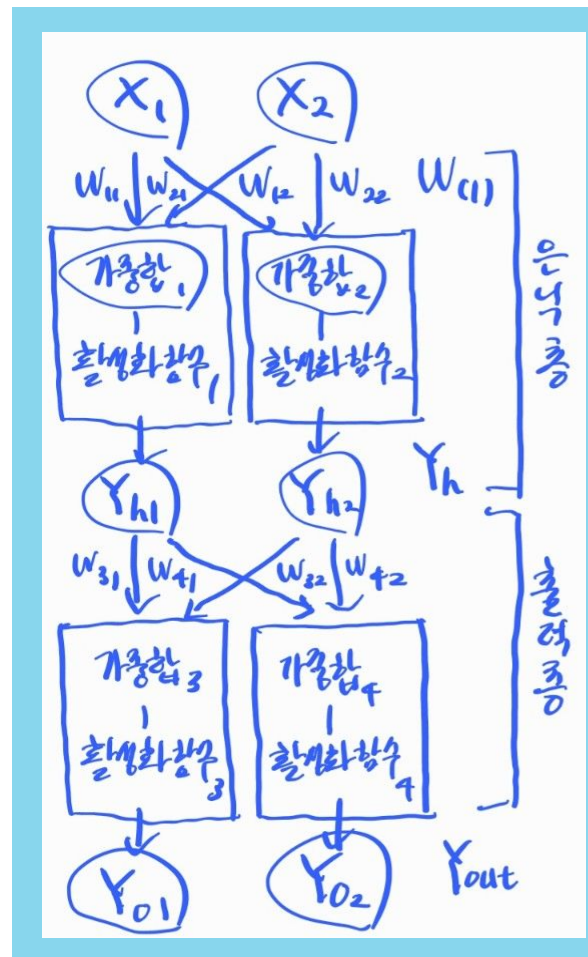
시그모이드 함수의 미분

$$\therefore y_{oi} \cdot (1 - y_{oi})$$

$$\rightarrow (y_{oi} - y_{ti}) \cdot y_{oi} \cdot (1 - y_{oi})$$

a-2) - 가중합₃ 식 편미분 = w_{31} .

$$\therefore a) \frac{\partial 오차_{y_{o1}}}{\partial y_{h1}} = (y_{o1} - y_{t1}) \cdot y_{o1} \cdot (1 - y_{o1}) \cdot w_{31} = \delta y_{o1} \cdot w_{31}$$



은닉층 오차 업데이트하기

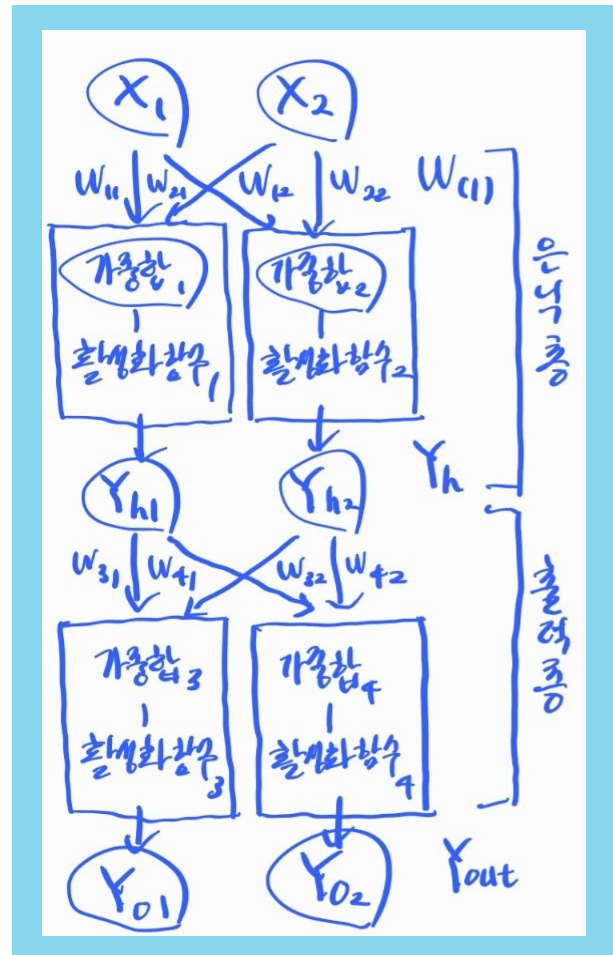
$$\textcircled{b) \frac{\delta \text{오차 } y_{02}}{\delta y_{n1}} = \frac{\delta \text{오차 } y_{02}}{\delta \text{가중합}_4} \cdot w_{41}}$$

$$\hookrightarrow \frac{\delta \text{오차 } y_{02}}{\delta y_{02}} \cdot \frac{\delta y_{02}}{\text{가중합}_4} \Rightarrow \textcircled{a-1} \text{과정과 동일}$$

$$\therefore \textcircled{b) \frac{\delta \text{오차 } y_{02}}{\delta y_{n1}} = (y_{02} - y_{t2}) \cdot y_{02}(1 - y_{02}) w_{41} = \delta y_{02} \cdot w_{41}}$$

$$\frac{\delta \text{오차 } Y_{out}}{\delta w_{11}} = \frac{\delta \text{오차 } Y_{out}}{\delta y_{h1}} \cdot \frac{\delta y_{h1}}{\delta \text{가중합}_1} \cdot \frac{\delta \text{가중합}_1}{\delta w_{11}}$$

$$= (\delta y_{01} w_{31} + \delta y_{02} w_{41}) y_{h1} (1 - y_{h1}) x_1$$



델타 식

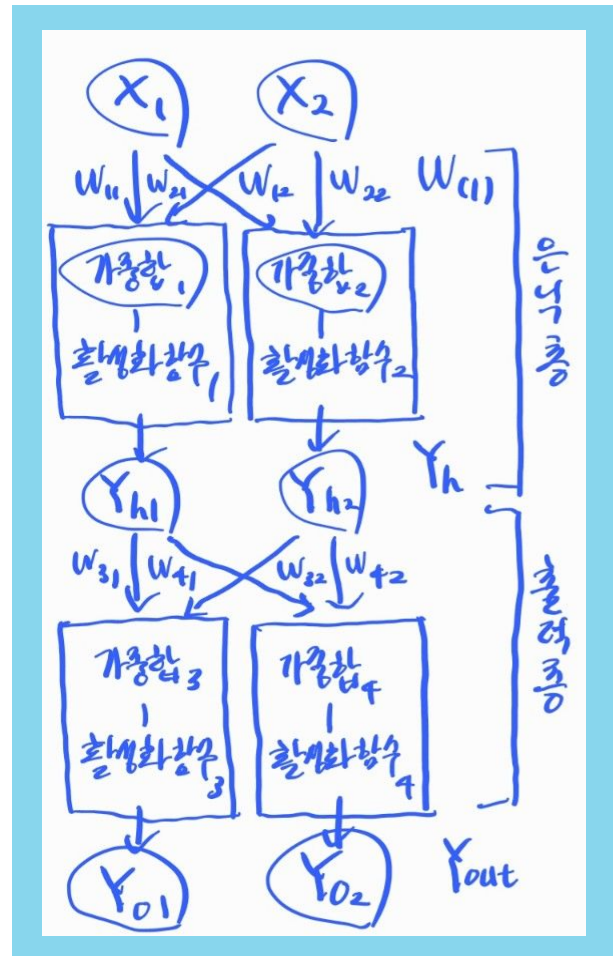
출력층의 오차 전달 = $(y_{oi} - y_{oi}) \cdot y_{oi}(1 - y_{oi}) \cdot y_{ni}$

은닉층의 " = $(\delta y_{oi} \cdot y_{oi} + \delta y_{o2} \cdot y_{o2}) y_{ni}(1 - y_{ni}) \cdot x_1$

$y(1-y)$ 의 형태는 동일 / $y_{ni} - y_{ni}$, $\delta y_{oi} \cdot y_{oi} + \delta y_{o2} \cdot y_{o2}$ 둘다 오차 전달.

$(\delta x_1) \cdot y(1-y)$ 의 형태로 일반화됨 $\rightarrow \delta h$ 로 일반화, (은닉층의 델타식)

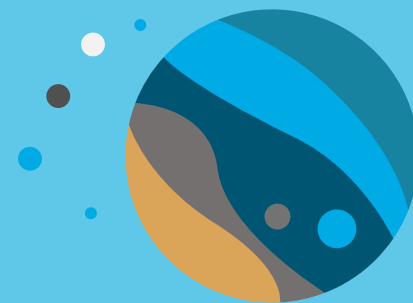
$\therefore W_{11}(t+1) = W_{11}(t) - \delta h \cdot x_1$





Part 4

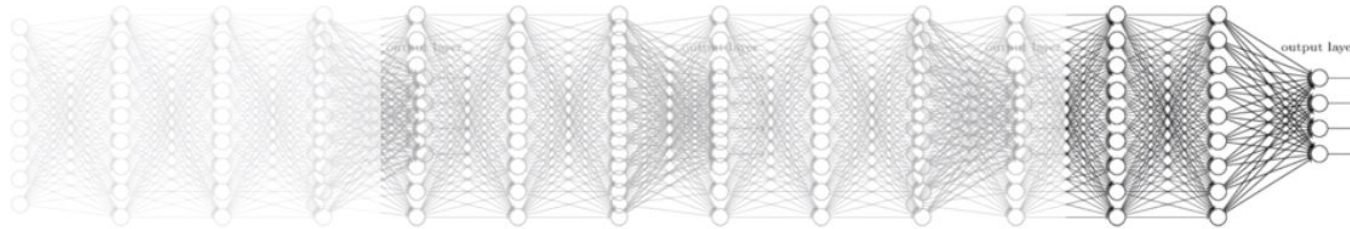
고급 경사 하강법



0. 기울기 소실 문제

기울기 소실(vanishing gradient) 문제: 층이 늘어나며 역전파를 통해 전달되는 기울기의 값이 점점 작아져 맨 처음 층까지 전달되지 않는 문제

Vanishing gradient (NN winter2: 1986-2006)

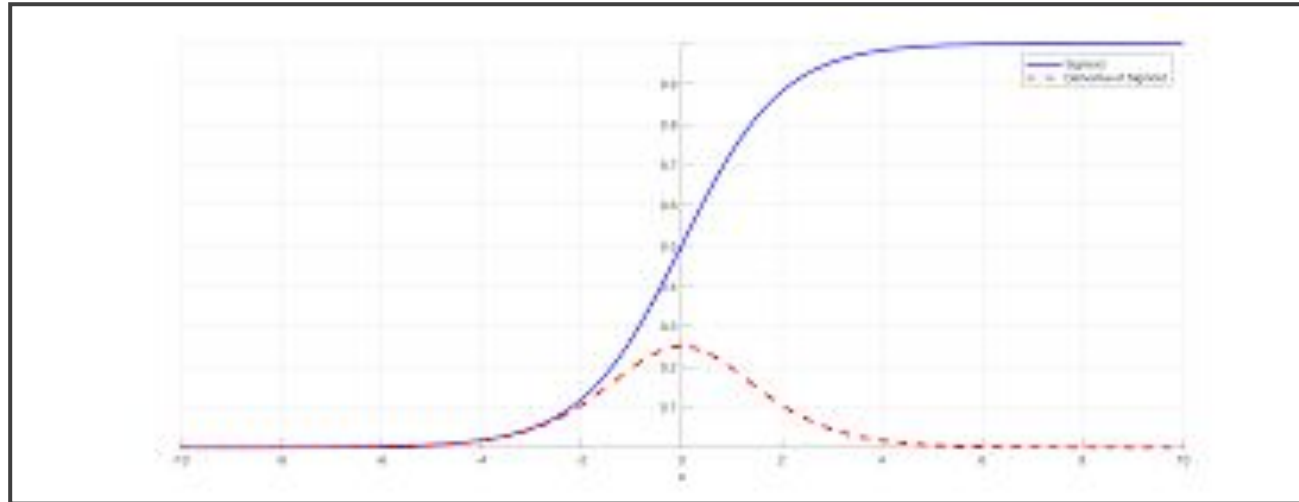


0. 기울기 소실 문제

시그모이드 함수의 특성 때문

시그모이드 함수: 0에서 1사이의 함수이며, 값이 들어왔을 때, 0~1 사이의 값을 반환,
계단 함수의 부드러운 s형태와 유사

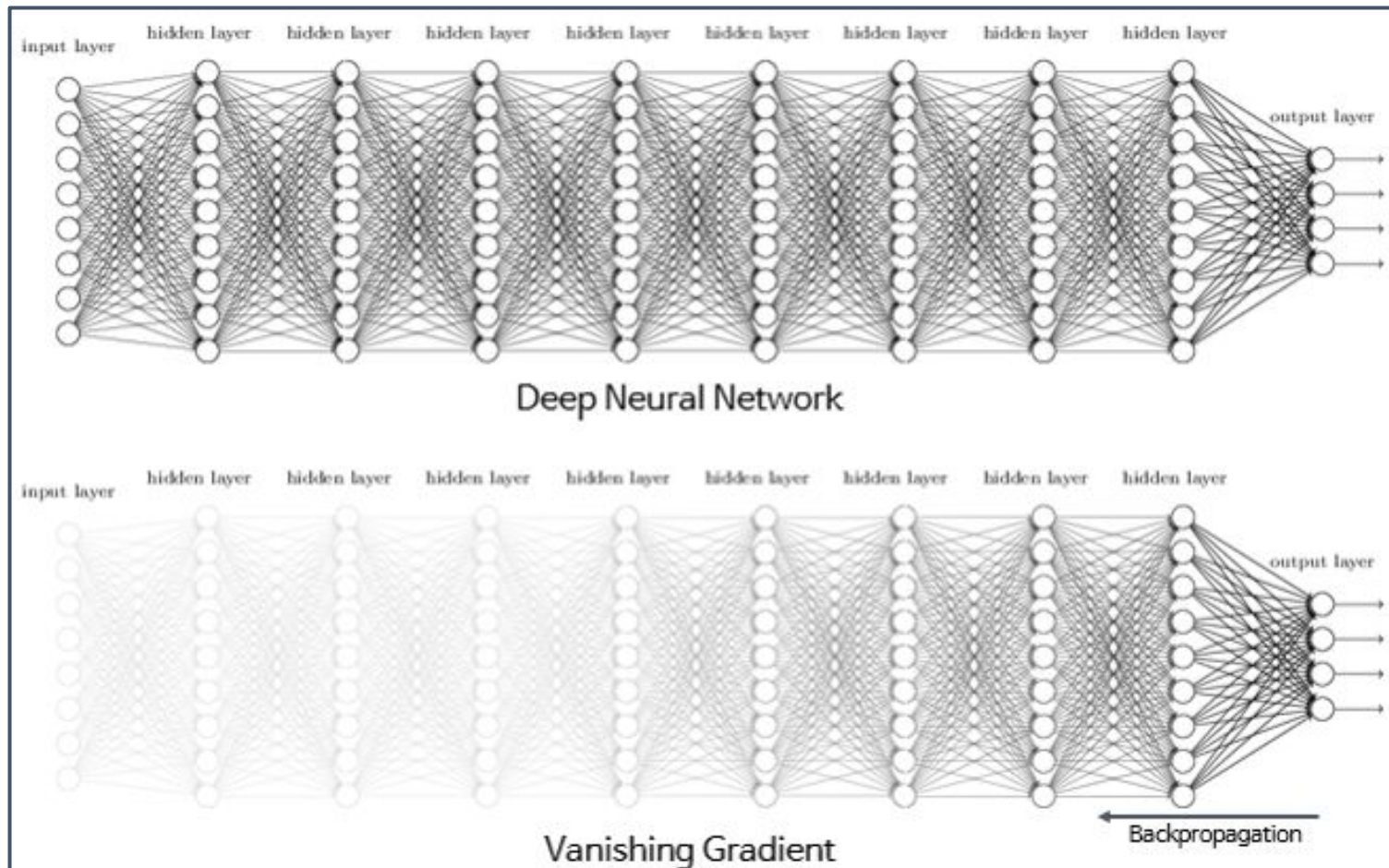
$$f(x) = \sigma(x) = \frac{1}{1 + e^{-x}}$$



0. 기울기 소실 문제

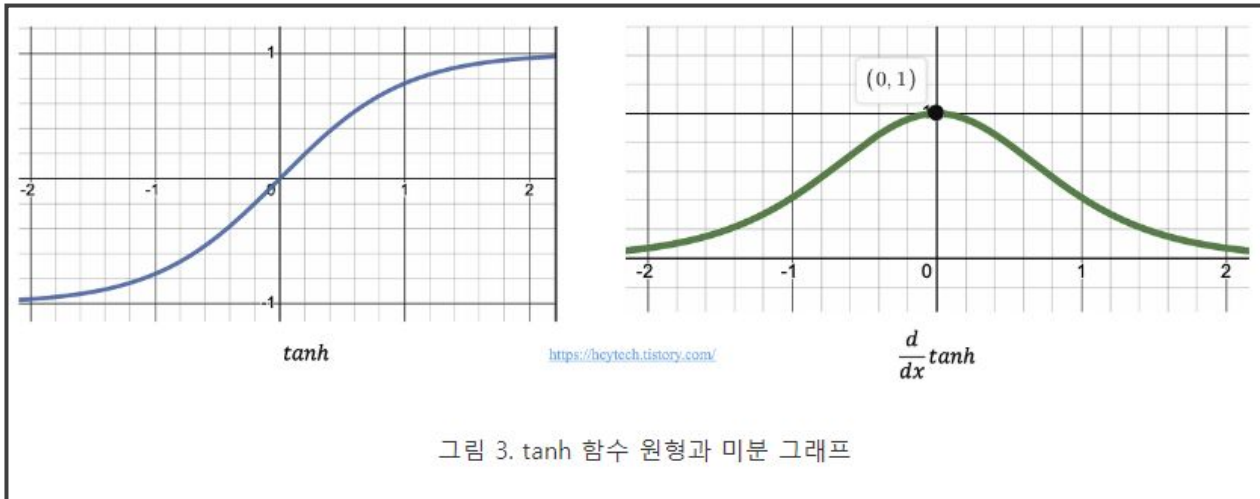
- 역전파 과정에서 Sigmoid 함수의 미분값이 거듭 곱해지면 출력층과 멀어질수록 Gradient 값이 매우 작아짐
- e(exponential)는 컴퓨터가 계산할 때 정확한 값이 아닌 근사값으로 계산해야 되기 때문에 역전파 과정에서 점차 학습 오차까지 증가
- 결국 Sigmoid 함수를 활용하면 모델 학습이 제대로 이루어지지 않음

0. 기울기 소실 문제

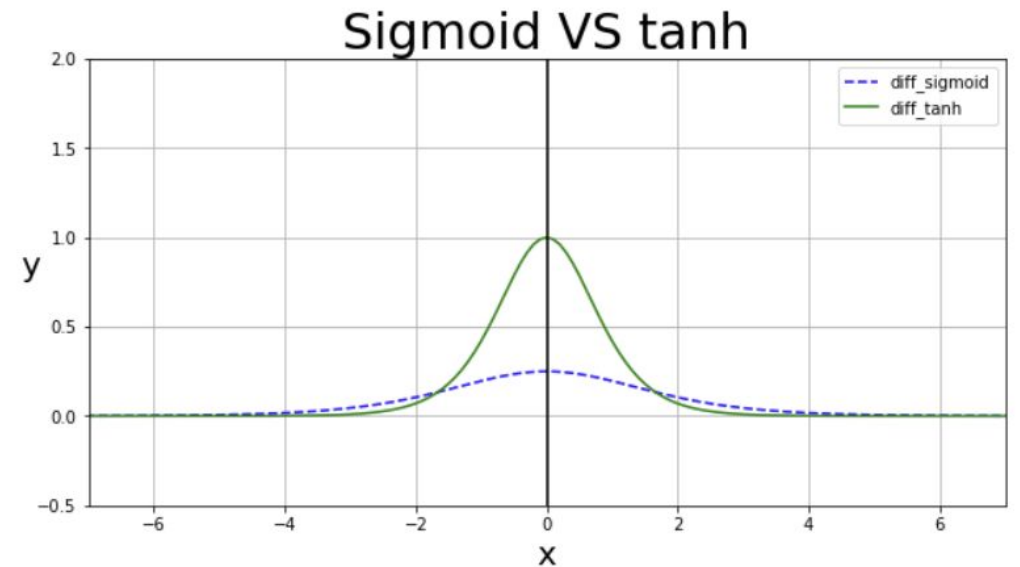


0. 여러 활성화 함수의 도입

1) 하이퍼볼릭 탄젠트 함수(Hyperbolic Tangent, tanh)



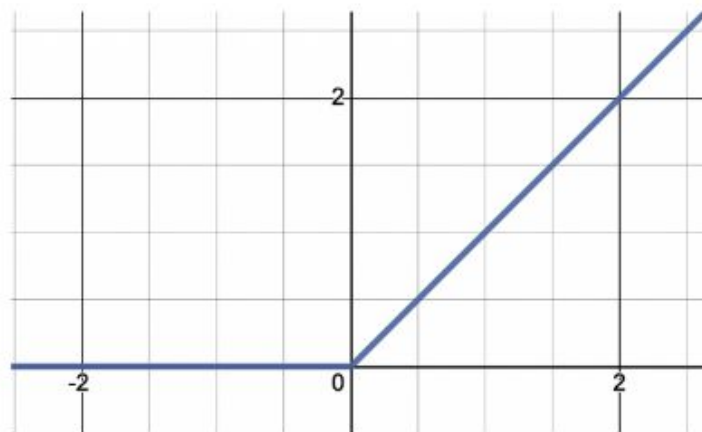
	시그모이드 함수	하이퍼볼릭 탄젠트 함수
범위	0 ~ 1	-1 ~ 1
중앙값	0.5	0
미분 최대값	0.3	1



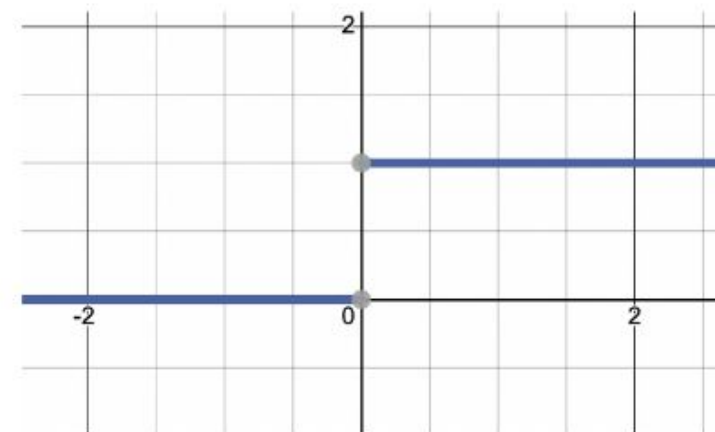
0. 여러 활성화 함수의 도입

2) 렐루(Rectified Linear Unit, ReLU)

$$f(x) = \begin{cases} 0 & \text{for } x \leq 0 \\ x & \text{for } x > 0 \end{cases}$$



ReLU

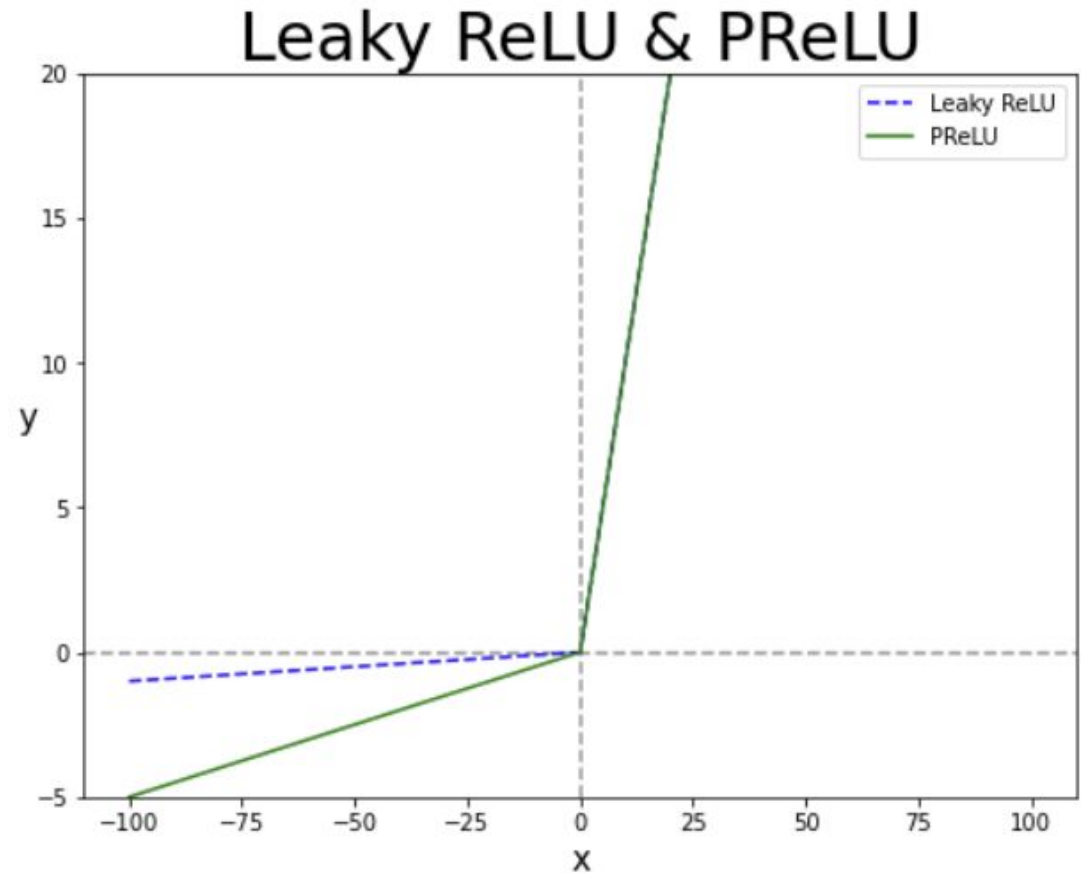
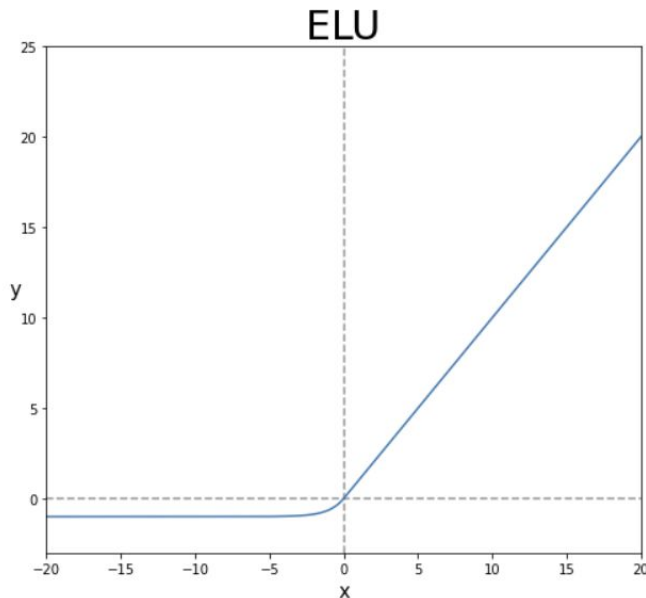


$\frac{d}{dx} \text{ReLU}$

<https://heytech.tistory.com/>

0. 여러 활성화 함수의 도입

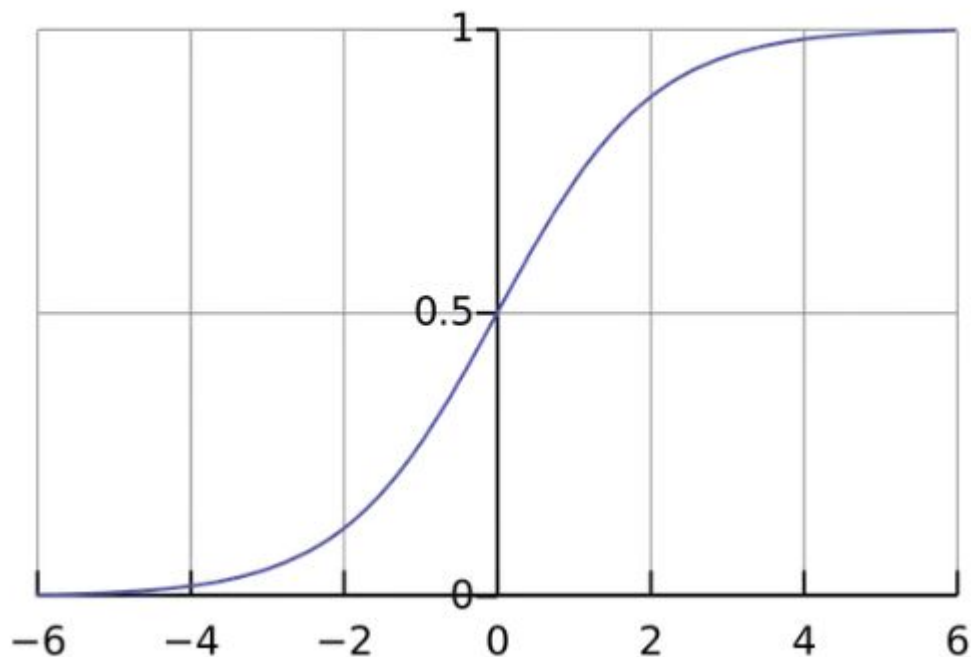
1. 리키 렐루(Leaky ReLU, LReLU)
2. 파라미터 렐루(Parameter ReLU, PReLU)
3. ELU(Exponential Linear Unit)
4. SELU(Scaled ELU)











0. 여러 활성화 함수의 도입

3) 소프트맥스(Softmax)

$$\text{Softmax}(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$$



0. 여러 활성화 함수의 도입

Name	Plot	Equation	Derivative (with respect to x)	Range	Order of continuity	Monotonic	Derivative Monotonic
Identity		$f(x) = x$	$f'(x) = 1$	$(-\infty, \infty)$	C^∞	Yes	Yes
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$	$\{0, 1\}$	C^{-1}	Yes	No
Logistic (a.k.a. Soft step)		$f(x) = \frac{1}{1 + e^{-x}}$	$f'(x) = f(x)(1 - f(x))$	$(0, 1)$	C^∞	Yes	No
TanH		$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$	$(-1, 1)$	C^∞	Yes	No
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2 + 1}$	$\left(-\frac{\pi}{2}, \frac{\pi}{2}\right)$	C^∞	Yes	No
Softsign [7][8]		$f(x) = \frac{x}{1 + x }$	$f'(x) = \frac{1}{(1 + x)^2}$	$(-1, 1)$	C^1	Yes	No
Rectified linear unit (ReLU)[9]		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$[0, \infty)$	C^0	Yes	Yes
Leaky rectified linear unit (Leaky ReLU)[10]		$f(x) = \begin{cases} 0.01x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0.01 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$(-\infty, \infty)$	C^0	Yes	Yes

고급 경사 하강법

경사 하강법의 장점: 정확하게 가중치를 찾아감

경사 하강법의 단점: 한 번 업데이트할 때마다 전체 데이터를 미분해야 하므로 계산량이 매우 많음

→ 고급 경사 하강법의 등장

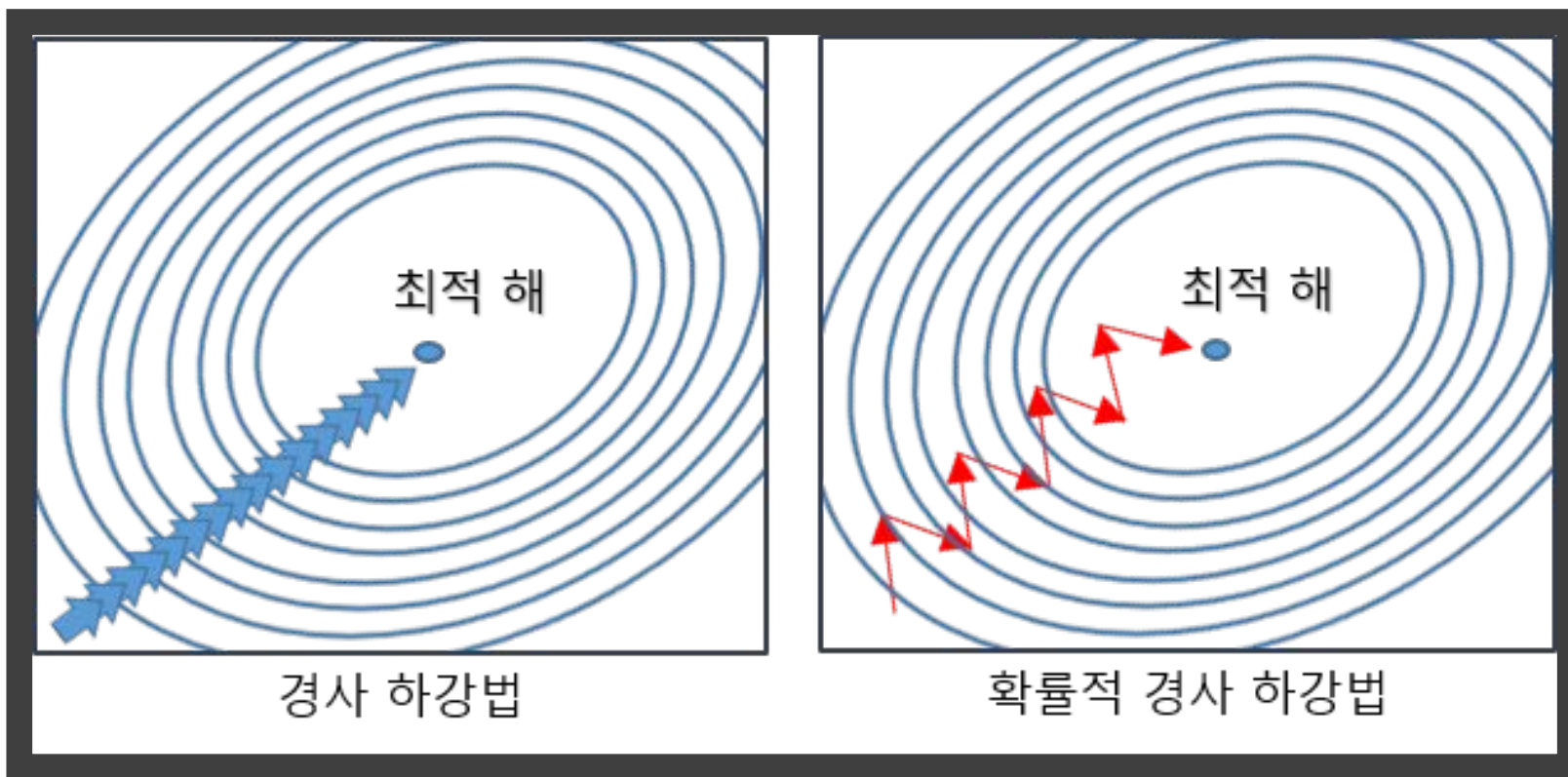
확률적 경사 하강법 (Stochastic Gradient Descent, SGD)

경사 하강법의 불필요하게 많은 계산량을 속도를 느리게 할 뿐만 아니라, 최적 해를 찾기 전에 최적화 과정이 멈출 수도 있음

-> 확률적 경사 하강법 (Stochastic Gradient Descent, SGD)

- 랜덤하게 추출한 일부 데이터 사용 → 더 빨리, 자주 업데이트 가능
- 중간 결과의 진폭이 크고 불안정해 보일 수도 있지만 속도가 확연히 빠르면서도 최적 해에 근사한 값을 찾아냄

확률적 경사 하강법 (Stochastic Gradient Descent, SGD)



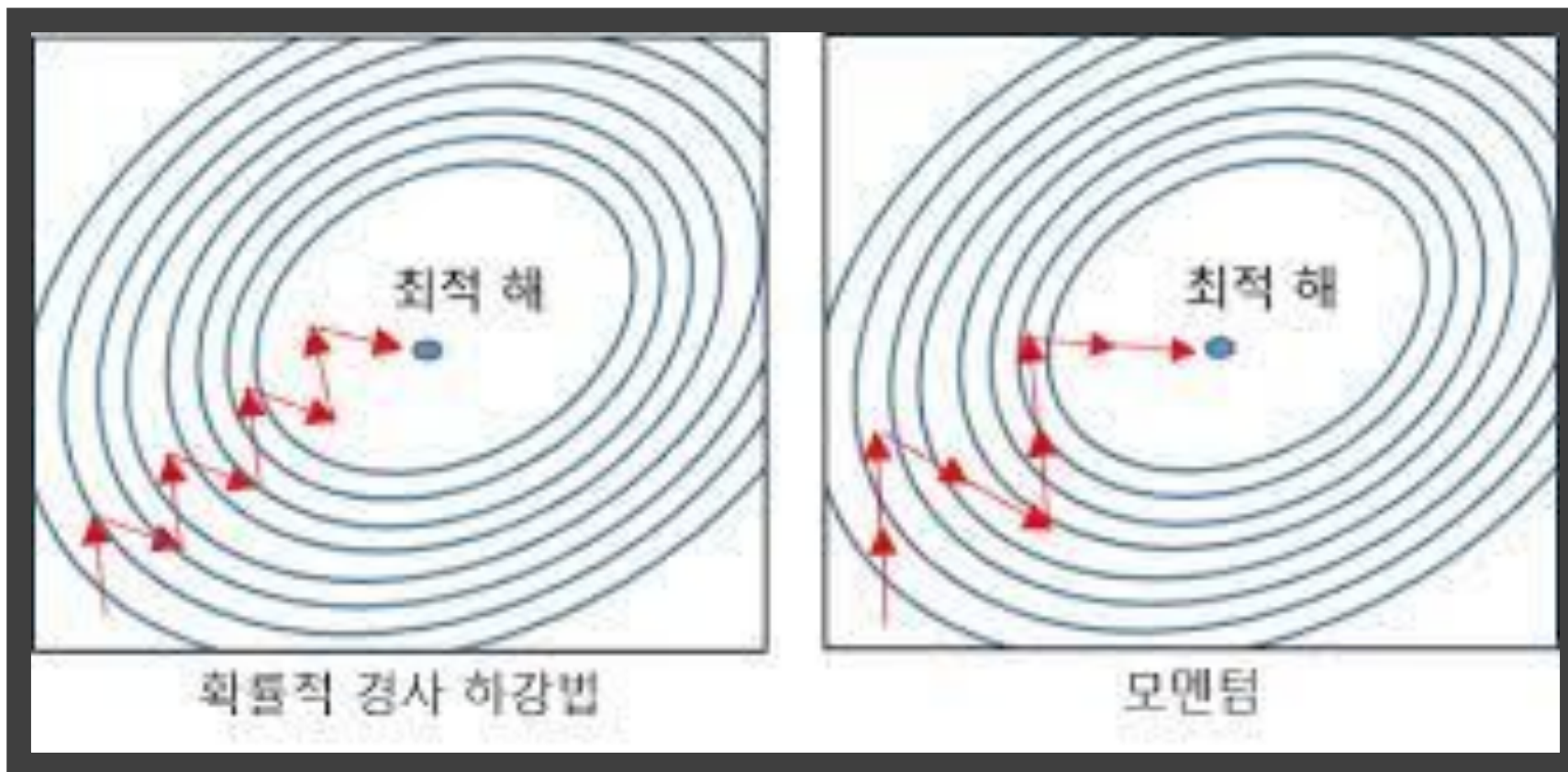
모멘텀SGD

모멘텀(momentum): '관성, 탄성, 가속도'

-> **모멘텀 SGD**: 경사하강법에 탄력을 더해 주는 것

- 같은 방향으로 일정한 비율만 수정
- 이전 이동 값을 고려하여 일정 비율만큼만 다음 값을 결정하므로 관성의 효과를 냄

모멘텀SGD



고급 경사 하강법

고급 경사 하강법	개요	효과	케라스 사용법
확률적 경사 하강법(SGD)	랜덤하게 추출한 일부 데이터를 사용해 더 빨리, 자주 업데이트를 하게 하는 것	속도 개선	keras.optimizers.SGD(lr = 0.1) 케라스 최적화 함수를 이용
모멘텀(Momentum)	관성의 방향을 고려해 진동과 폭을 줄이는 효과	정확도 개선	keras.optimizers.SGD(lr = 0.1, momentum = 0.9) 모멘텀 계수를 추가
네스테로프 모멘텀(NAG)	모멘텀이 이동시킬 방향으로 미리 이동해서 그래디언트를 계산, 불필요한 이동을 줄이는 효과	정확도 개선	keras.optimizers.SGD(lr = 0.1, momentum = 0.9, nesterov = True) 네스테로프 옵션을 추가

아다그라드 (Adagrad)

변수 업데이트 횟수에 따라 학습률을 조절하는 옵션이 추가된 방법

가중치(W) 벡터의 하나의 값($w[i]$)

많이 변화하지 않은 변수들은 학습률을 크게하여 빠르게 loss 값을 줄이고
반대로 많이 변화한 변수들은 최적값에 근접하였을 것으로 가정해
학습률을 적게 하여 작은 크기로 이동해 세밀한 값을 조정

같은 데이터가 여러 번 학습되는 학습모델에 유용

ex) 언어와 관련된 word2vec 이나 GloVe 에서 학습 단어의 등장확률에 따라
변수의 사용 비율이 차이나게 되어 많이 등장한 단어는 가중치를 적게 수정하고
적게 등장한 단어는 가중치를 많이 수정한다.

아다그라드 (Adagrad)

$$g_t = g_{t-1} + (\nabla f(x_{t-1}))^2$$

g_t : t 번째 time step까지의 기울기 누적 크기

$$x_t = x_{t-1} - \frac{\eta}{\sqrt{g_t + \epsilon}} \cdot \nabla f(x_{t-1})$$

ϵ : 분모가 0이 되는 것을 방지하기 위한 작은 값 $\approx 10^{-6}$

η : 학습률(Learning rate)

t 번째 time step까지 기울기를 누적인 값(g_t)를 계산하여
이 값의 제곱근 역수를 t 번째 time step에서의 x_t 의 η 에 곱함

이 때, g_t 가 0인 경우 값이 무한대로 발산할 수 있기 때문에 ϵ 을 같이 더해줌

아다그라드 (Adagrad)

#파이썬 소스 코드

```
g += gradient**2  
weight[i] += -lr * gradient / (np.sqrt(g) + e)
```

#Tensorflow 소스 코드

```
optimizer = tf.train.AdagradOptimizer(lr = 0.01).minimize(loss)
```

#Keras 소스 코드

```
keras.optimizers.Adagrad(lr = 0.01, epsilon = 1e - 6)
```

아주 작은 상수를 의미,
0으로 나뉘지는 것을 방지

아다그라드 (Adagrad)

장점

학습률을 직접 조절하지 않아도 됨
각 변수마다 적절한 학습률을 자동으로 설정

단점

학습이 오래 지속되면 변수의 update가 되지 않음
 g_t 값이 계속 증가하기 때문에 학습률을 나누는데 g_t 값이 사용되어
매개변수가 최적 값에 근사하지 않았음에도 학습률을 계속 작게 만들어 학습이 정체

알엠에스프롭 (RMSProp)

학습이 진행될수록 학습률이 극단적으로 감소하는
아다그라드(Adagrad)의 문제점을 보완

아다그라드(Adagrad)와 마찬가지로 변수별로 학습률을 조절하는데
기울기 업데이트 방법에서 차이가 있음

gradient 제공에 더해지는 가중치가
지수승으로 변화하기 때문에
지수가중평균이라 부름

기울기를 단순히 같은 비율로 누적하지 않고 지수가중이동평균을 활용해
최근 경로의 gradient는 많이 반영되고 오래된 경로의 gradient는 작게 반영

매번 새로운 gradient 제공의 비율을 반영하여 평균을 업데이트하는 방식

알엠에스프롭 (RMSProp)

$$g_t = \gamma g_{t-1} + (1 - \gamma)(\nabla f(x_{t-1}))^2$$

g_t : t 번째 time step까지의 기울기 누적 크기

γ : 지수이동평균의 업데이트 계수

$$x_t = x_{t-1} - \frac{\eta}{\sqrt{g_t + \epsilon}} \cdot \nabla f(x_{t-1})$$

ϵ : 분모가 0이 되는 것을 방지하기 위한 작은 값 $\approx 10^{-6}$

η : 학습률

t 번째 time step의 누적 기울기(g_t)는 이전 time step까지의 누적 기울기(g_{t-1})에 γ 를 곱해 점차 작게 만들어 주고, 새로운 gradient에는 $(1-\gamma)$ 를 곱한 값을 더하여 업데이트

이 값의 제곱근 역수를 t 번째 time step에서의 x_t 의 η 에 곱함

이 때 g_t 가 0인 경우 값이 무한대로 발산할 수 있기 때문에 이를 방지하기 위해 ϵ 을 더해줌

알엠에스프롭 (RMSProp)

#파이썬 소스 코드

```
g = gamma * g + (1 - gamma) * gradient**2  
weight[i] += -lr * gradient / (np.sqrt(g) + e)
```

#Tensorflow 소스 코드

```
optimizer = tf.train.RMSPropOptimizer(lr = 0.01, decay = 0.9, momentum = 0.0, epsilon = 1e - 10).minimize(cost)
```

#Keras 소스 코드

```
keras.optimizers.RMSprop(lr = 0.001, rho = 0.9, epsilon = None, decay = 0.0)
```

장점

학습률을 직접 조절하지 않아도 됨

각 변수마다 적절한 학습률을 자동으로 설정

아다그라드(Adagrad) 보다 학습을 오래할 수 있음

아다델타 (Adadelta)

알엠에스프롭(RMSProp) 과 비슷하게 아다그라드(Adagrad) 에서 생기는 학습률이 극단적으로 감소하는 문제점을 보완

과거 기울기의 제곱합을 전부 누적하지 않고, 지수평균을 이용한다는 점은 알엠에스프롭(RMSProp) 과 동일하지만 학습률을 지수평균을 이용한 변화량을 사용하는 것이 특징

#Tensorflow 소스 코드

```
optimizer = tf.train.adadeltoptimizer(lr = 0.001, rho = 0.95, epsilon = 1e - 08).minimize(loss)
```

#Keras 소스 코드

```
keras.optimizers.Adadelta(lr = 1.0, rho = 0.95, epsilon = None, decay = 0.0)
```

아담 (Adam)

모멘텀(Momentum) 과 알엠에스프롭(RMSProp) 의 장점을 결합한 알고리즘으로
학습의 방향과 크기를 모두 개선하여 최근 딥러닝에서 가장 많이 사용

#Tensorflow 소스 코드

```
optimizer = tf.train.AdamOptimizer(lr = 0.001, beta1 = 0.9, beta2 = 0.999, epsilon = 1e - 08).minimize(loss)
```

#Keras 소스 코드

```
keras.optimizers.Adam(lr = 0.001, beta_1 = 0.9, beta_2 = 0.999, epsilon = None, decay = 0.0, amsgrad = False)
```

아담 (Adam)

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) \nabla f(x_{t-1})$$

β_1 : Momentum의 지수이동평균 ≈ 0.9

$$g_t = \beta_2 g_{t-1} + (1 - \beta_2) (\nabla f(x_{t-1}))^2$$

β_2 : RMSProp의 지수이동평균 ≈ 0.999

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}, \hat{g}_t = \frac{g_t}{1 - \beta_2^t}$$

\hat{m}, \hat{g} : 학습 초기 시 m_t, g_t 가 0이 되는 것을 방지하기 위한 보정 값

ϵ : 분모가 0이 되는 것을 방지하기 위한 작은 값 $\approx 10^{-8}$

$$x_t = x_{t-1} - \frac{\eta}{\sqrt{\hat{g}_t + \epsilon}} \cdot \hat{m}_t$$

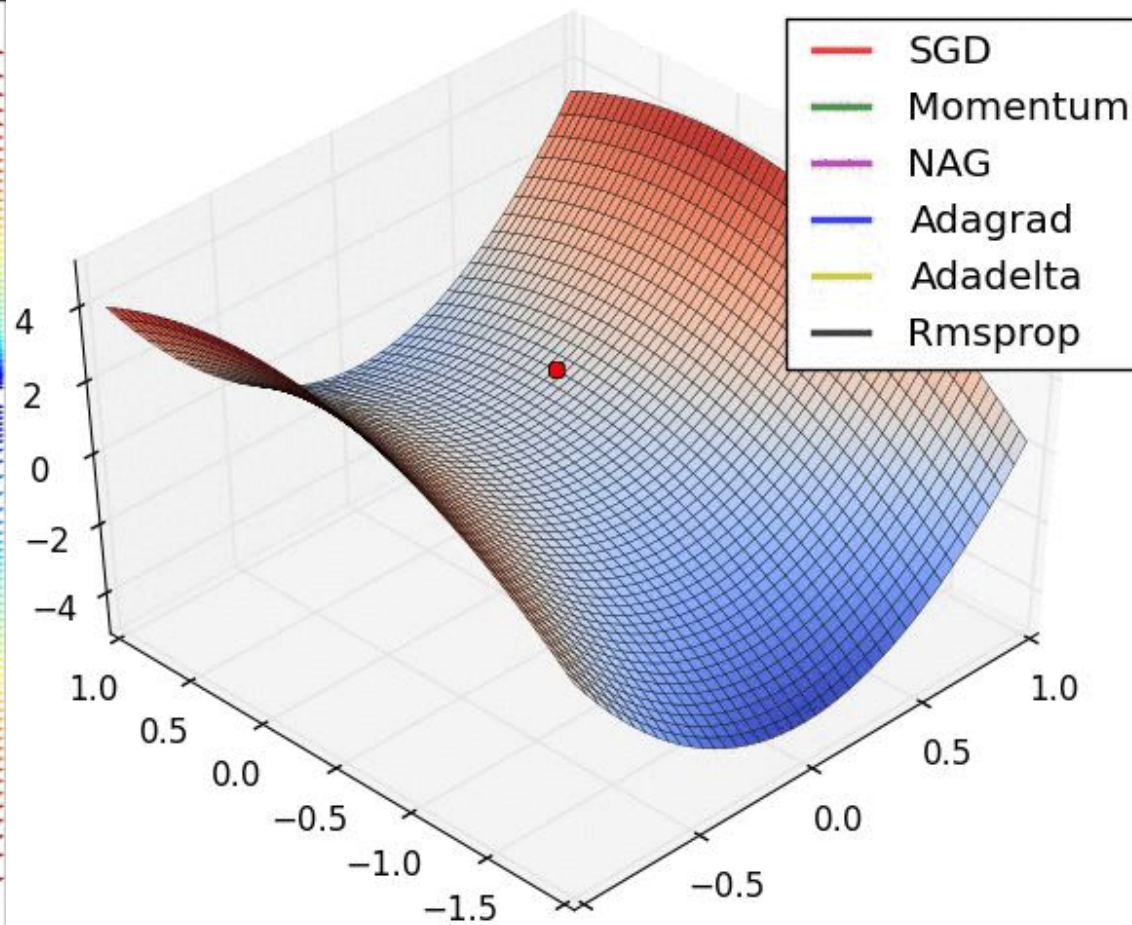
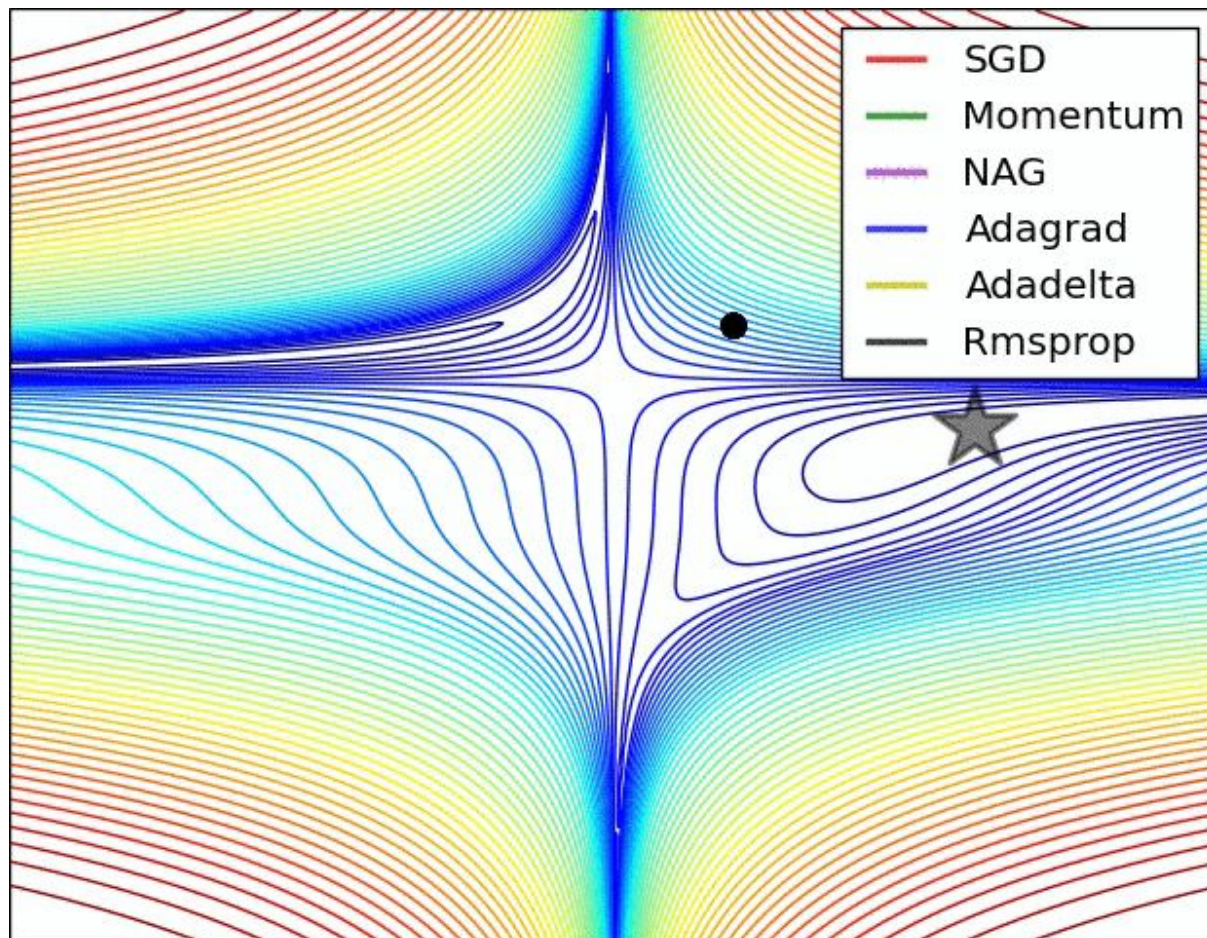
η : 학습률 ≈ 0.001

m_t 와 g_t 는 각각 모멘텀(momentum) 과 알엠에스프롭(RMSProp) 에서 사용한 수식과 동일

지수이동평균을 사용하여 최근 경로의 gradient는 많이 반영하고
오래된 경로의 gradient는 작게 반영

밑의 식은 초기값이 0에 가까워지는 편향의 문제를 해결하기 위해 m_t 와 x_t 를 구함 : 편향 보정

고급경사하강법 수렴



감사합니다

