Introduction to Deep learning

Youngpyo Ryu

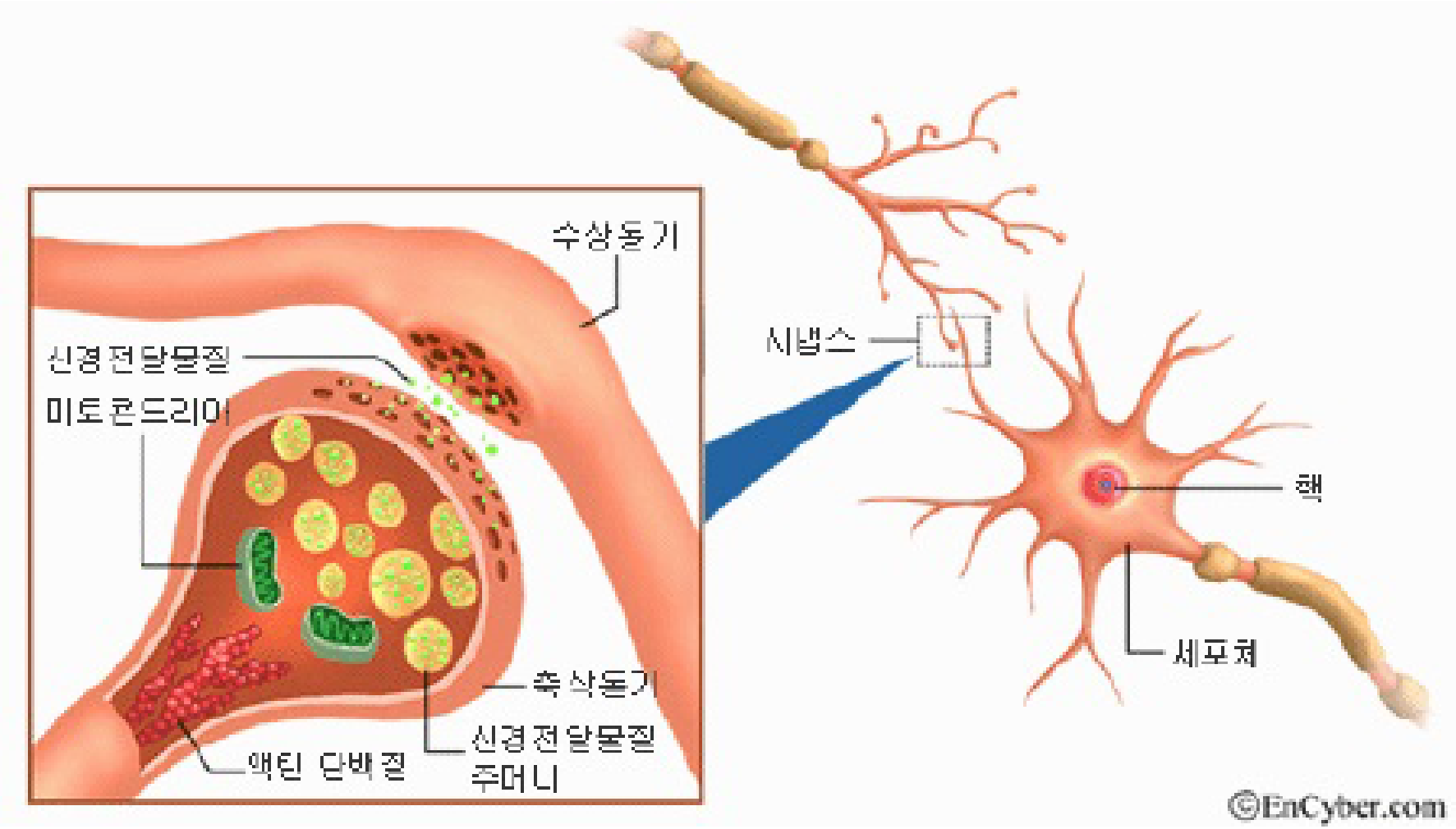
동국대학교 수학과대학원 응용수학 석사재학

youngpyoryu@dongguk.edu

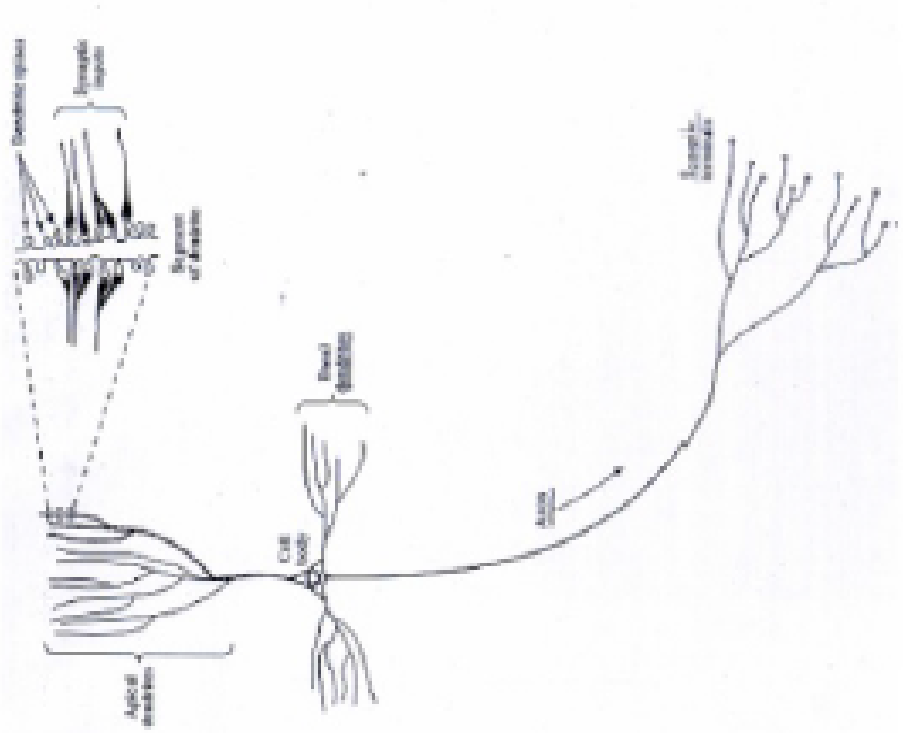
2018년 6월 30일

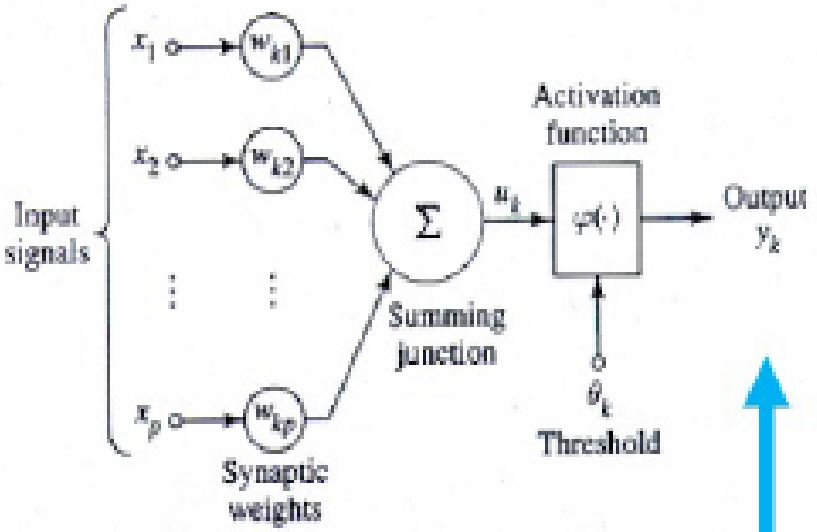
Overview

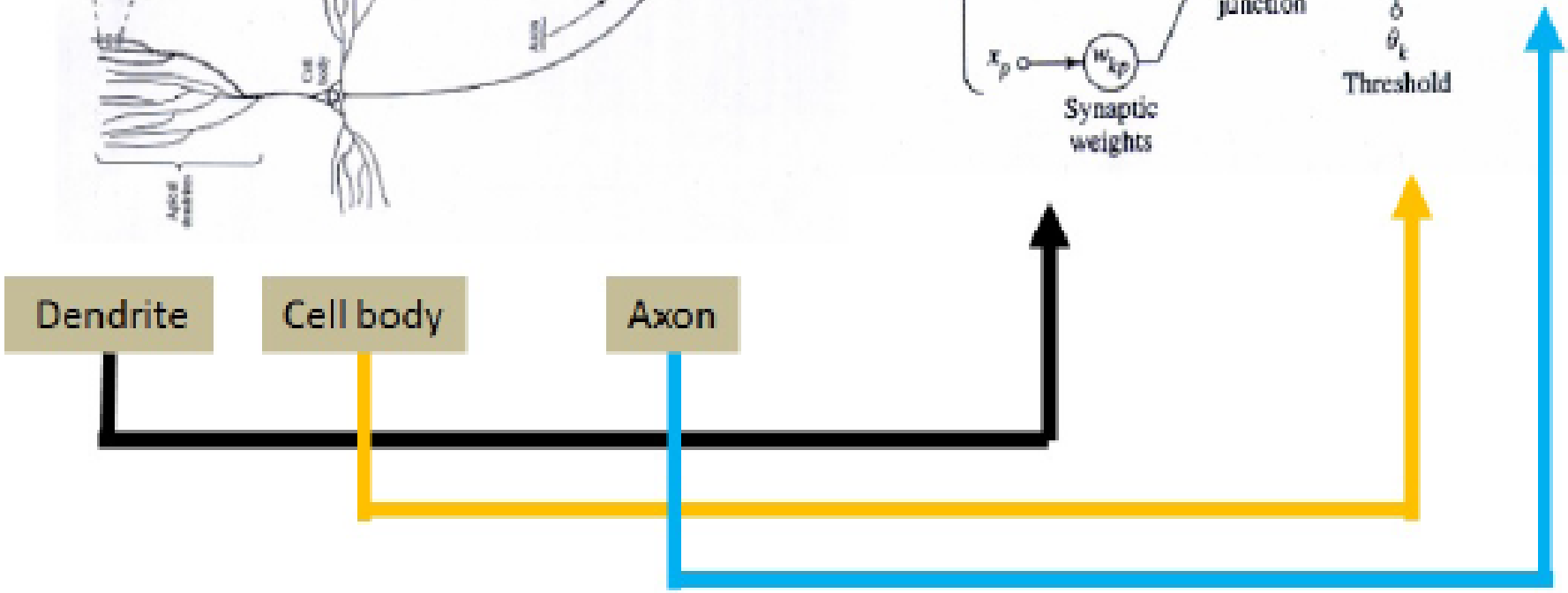
Neuron


수상돌기
신경전달물질
미토콘드리아
액틴 단백질
축삭돌기
신경전달물질
주머니
시냅스
핵
세포체
©EnCyber.com

Neuron




Input signals
Synaptic weights
Summing junction
Activation function
Threshold
Output


Dendrite
Cell body
Axon

Neuron

Quiz) 사람의 뉴런은 몇개 일까 ?

Neuron

Quiz) 사람의 뉴런은 몇개 일까 ?

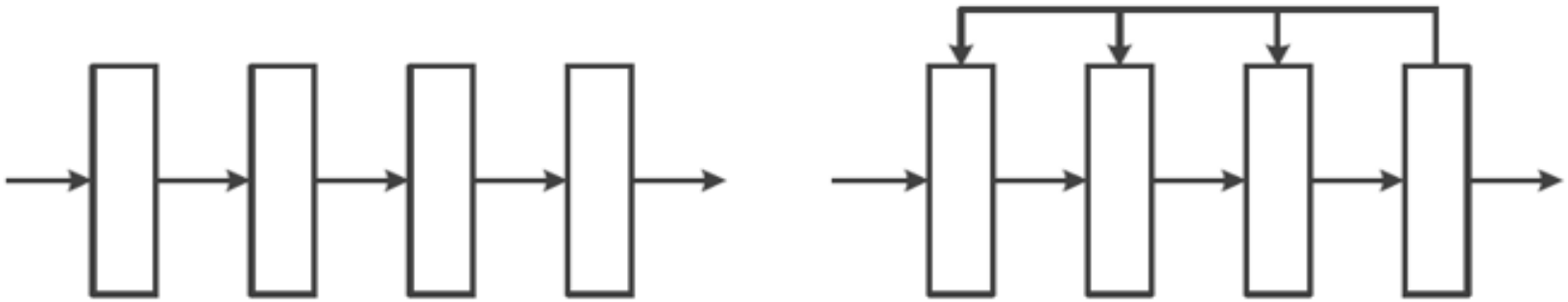
약 천억개 정도의 뉴런을 가지고 있다.

의식과 같은 뇌의 부분은 미지의 영역이지만, 많은 발전이 있었다.

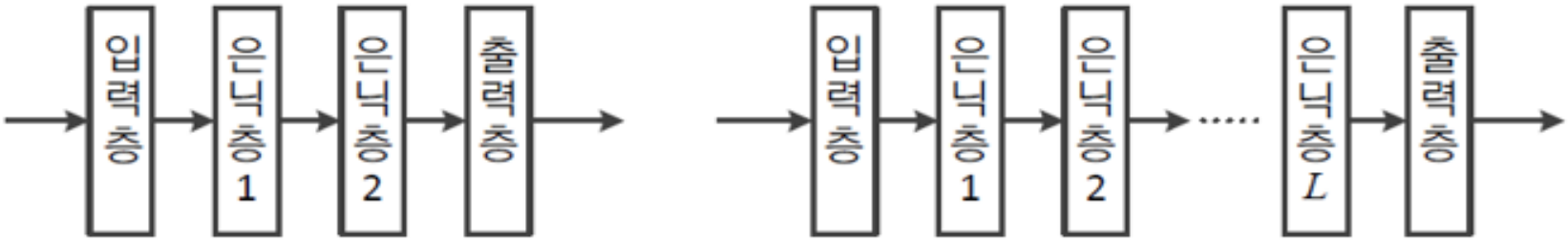
생물학적 뉴런 : ' 출력값 $=$ (상수 $\times$ 입력값) $+$ (또다른 상수) '

퍼셉트론

- 신경망에는 아주 다양한 모델이 존재함
  - 전방 신경망과 순환 신경망
  - 얕은 신경망과 깊은 신경망
  - 결정론 신경망과 스토캐스틱 신경망



(a) 전방 신경망과 순환 신경망



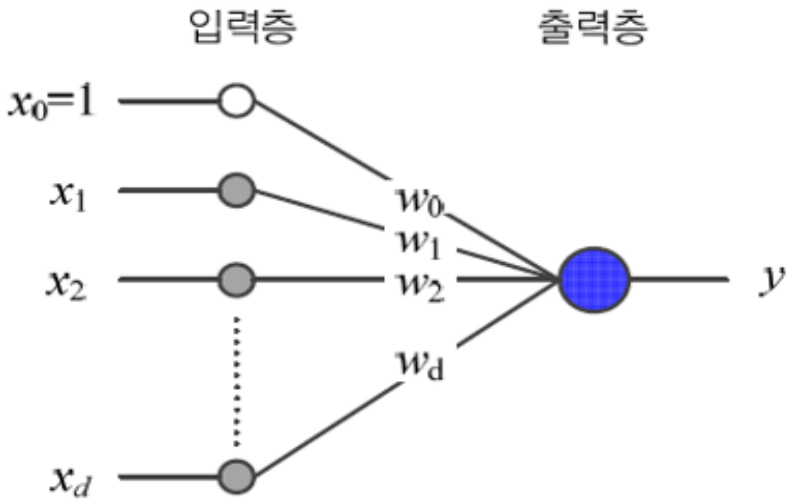
(b) 얕은 신경망과 깊은 신경망

그림 3-2 신경망의 종류

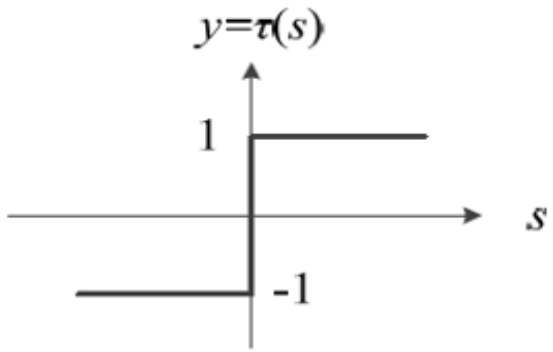
퍼셉트론의 구조

퍼셉트론의 구조

- 입력층과 출력층을 가짐
  - 입력층은 연산을 하지 않으므로 퍼셉트론은 단일 층 구조라고 간주
- 입력층의 i번째 노드는 특징 벡터 $\mathbf{x} = (x_1, x_2, \cdots, x_d)^{\mathrm{T}}$의 요소 x_i를 담당
- 항상 1이 입력되는 바이어스 노드
- 출력층은 한 개의 노드
- i번째 입력 노드와 출력 노드를 연결하는 에지는 가중치 w_i를 가짐



(a) 퍼셉트론의 구조



(b) 계단함수를 활성함수 $\tau(s)$로 이용함

그림 3-3 퍼셉트론의 구조와 동작

퍼셉트론의 동작

퍼셉트론의 동작

- 해당하는 특징값과 가중치를 곱한 결과를 모두 더하여 s를 구하고, 활성함수 τ를 적용함
- 활성함수 τ로 계단함수를 사용하므로 최종 출력 y는 +1 또는 -1

$$y = \tau(s)$$

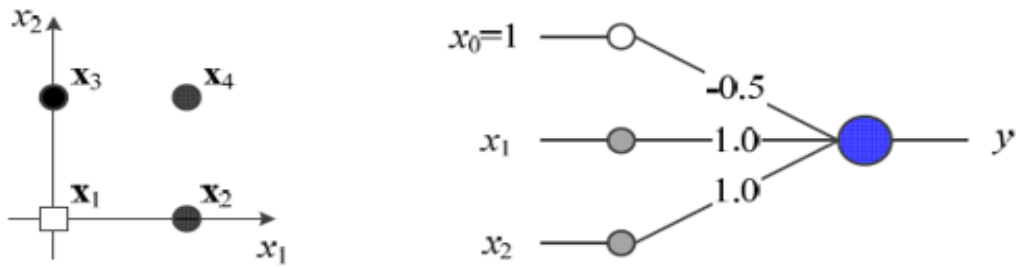
$$\text{이때 } s = w_0 + \sum_{i=1}^{d} w_i x_i, \qquad \tau(s) = \begin{cases} 1 & s \geq 0 \\ -1 & s < 0 \end{cases} \qquad (3.1)$$

퍼셉트론의 동작 예제

예제 3-1 **퍼셉트론의 동작**

2차원 특징 벡터로 표현되는 샘플을 4개 가진 훈련집합 $\mathbb{X} = \{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4\}$, $\mathbb{Y} = \{y_1, y_2, y_3, y_4\}$를 생각하자. [그림 3-4(a)]는 이 데이터를 보여준다.

$$\mathbf{x}_1 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, y_1 = -1, \quad \mathbf{x}_2 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, y_2 = 1, \quad \mathbf{x}_3 = \begin{pmatrix} 0 \\ 1 \end{pmatrix}, y_3 = 1, \quad \mathbf{x}_4 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, y_4 = 1$$



(a) 훈련집합 (b) 퍼셉트론

그림 3-4 OR 논리 게이트를 이용한 퍼셉트론의 동작 예시

샘플 4개를 하나씩 입력하여 제대로 분류하는지 확인해 보자.

$$\begin{aligned}
\mathbf{x}_1&: s = -0.5 + 0 * 1.0 + 0 * 1.0 = -0.5, && \tau(-0.5) = -1 \\
\mathbf{x}_2&: s = -0.5 + 1 * 1.0 + 0 * 1.0 = 0.5, && \tau(0.5) = 1 \\
\mathbf{x}_3&: s = -0.5 + 0 * 1.0 + 1 * 1.0 = 0.5, && \tau(0.5) = 1 \\
\mathbf{x}_4&: s = -0.5 + 1 * 1.0 + 1 * 1.0 = 1.5, && \tau(1.5) = 1
\end{aligned}$$

결국 [그림 3-4(b)]의 퍼셉트론은 샘플 4개를 모두 맞추었다. 이 퍼셉트론은 훈련집합을 100% 성능으로 분류한다고 말할 수 있다.

행렬표기

- 행렬 표기

$$s = \mathbf{w}^{\mathrm{T}}\mathbf{x} + w_0, \qquad \text{여기서 } \mathbf{x} = (x_1, x_2, \cdots, x_d)^{\mathrm{T}},\ \mathbf{w} = (w_1, w_2, \cdots, w_d)^{\mathrm{T}} \tag{3.2}$$

  - 바이어스 항을 벡터에 추가하면,

$$s = \mathbf{w}^{\mathrm{T}}\mathbf{x}, \qquad \text{여기서 } \mathbf{x} = (1, x_1, x_2, \cdots, x_d)^{\mathrm{T}},\ \mathbf{w} = (w_0, w_1, w_2, \cdots, w_d)^{\mathrm{T}} \tag{3.3}$$

  - 퍼셉트론의 동작을 식 (3.4)로 표현할 수 있음

$$y = \tau(\mathbf{w}^{\mathrm{T}}\mathbf{x}) \tag{3.4}$$

퍼셉트론의 동작

- [그림 3-4(b)]를 기하학적으로 설명하면,
  - 결정 직선 $d(\mathbf{x}) = d(x_1, x_2) = w_1 x_1 + w_2 x_2 + w_0 = 0$ → $x_1 + x_2 - 0.5 = 0$
    - w_1과 w_2는 직선의 방향, w_0은 절편을 결정
    - 결정 직선은 전체 공간을 +1과 -1의 두 부분공간으로 분할하는 분류기 역할

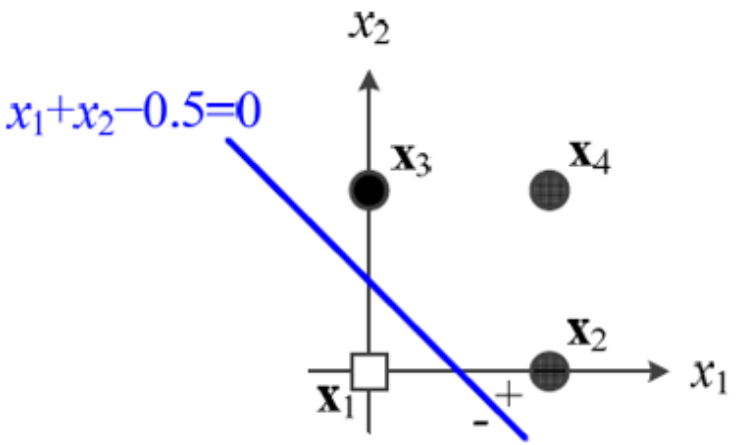


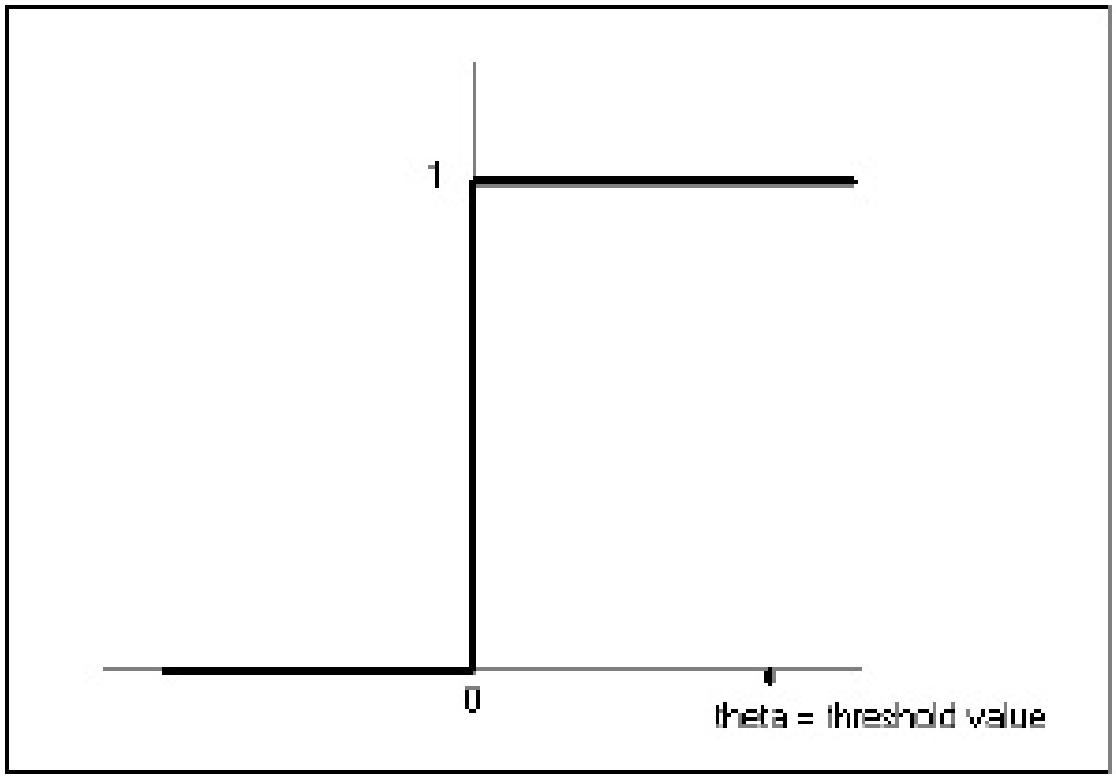
그림 3-5 [그림 3-4(b)]의 퍼셉트론에 해당하는 결정 직선

  - d차원 공간에서는 $d(\mathbf{x}) = w_1 x_1 + w_2 x_2 + \cdots + w_d x_d + w_0 = 0$
    - 2차원은 결정 직선decision line, 3차원은 결정 평면decision plane, 4차원 이상은 결정 초평면 decision hyperplane

Activation function

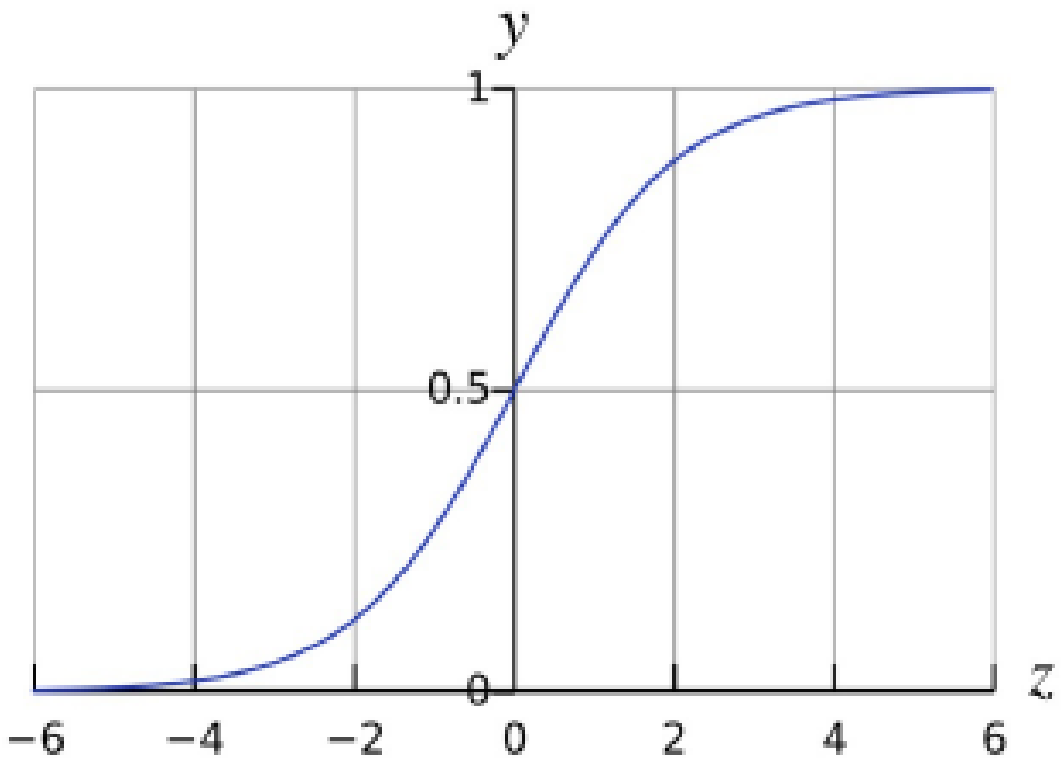
뉴런은 입력을 받았을 때 즉시 반응하지 않음 .

즉, 입력값이 어떤 분계점(Threshold)에 도달해야 출력이 발생.



Activation function

계단함수(step function)을 좀 더 개선하기 위해 sigmoid function을 이용.

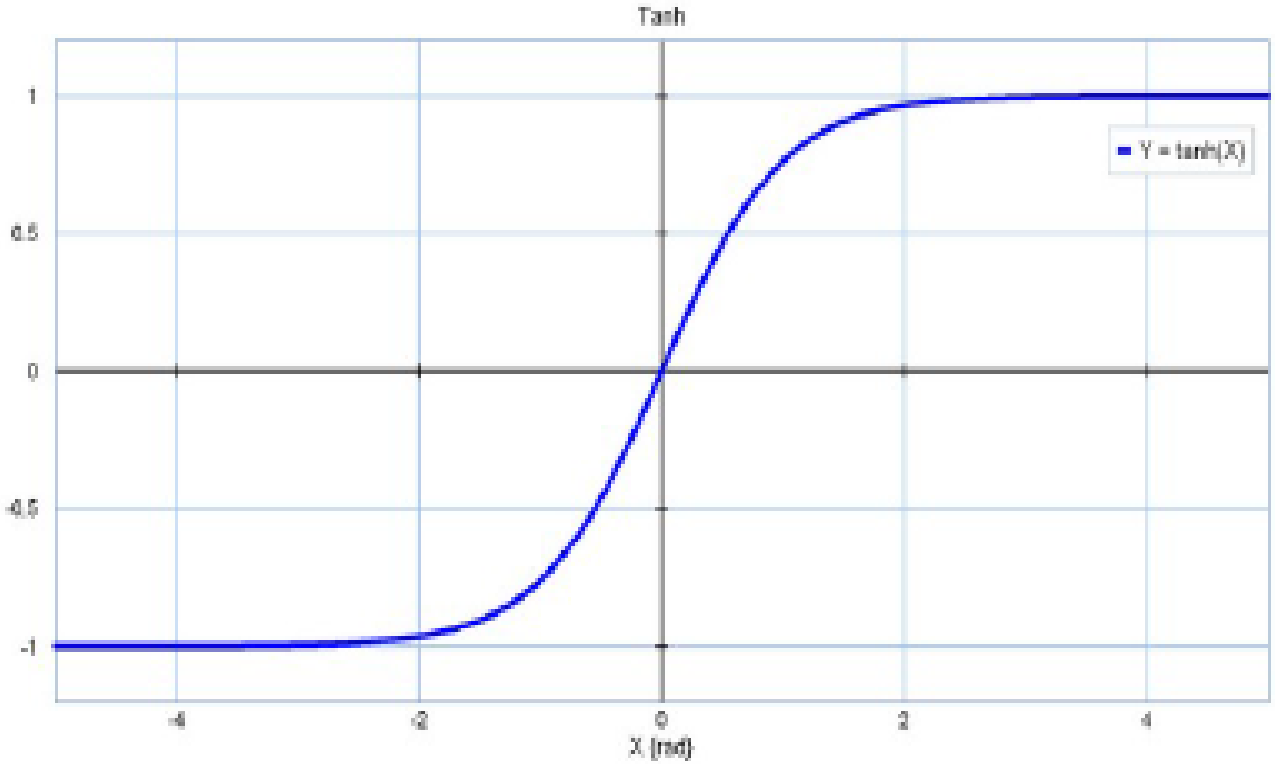


$$z = w_1 x_1 + w_2 x_2 + \ldots + w_n x_n$$

$$y = \frac{1}{1 + e^{-z}}$$

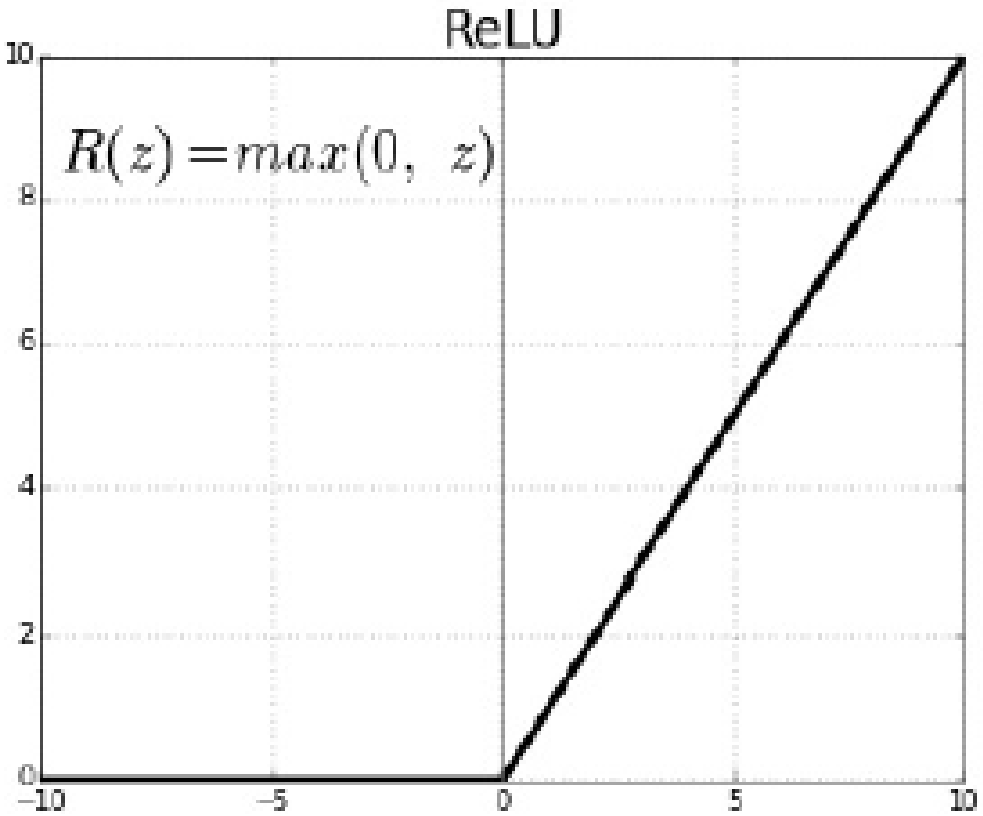
Activation function

sigmoid function의 범위를 확장하기 위해 tanh function을 사용.



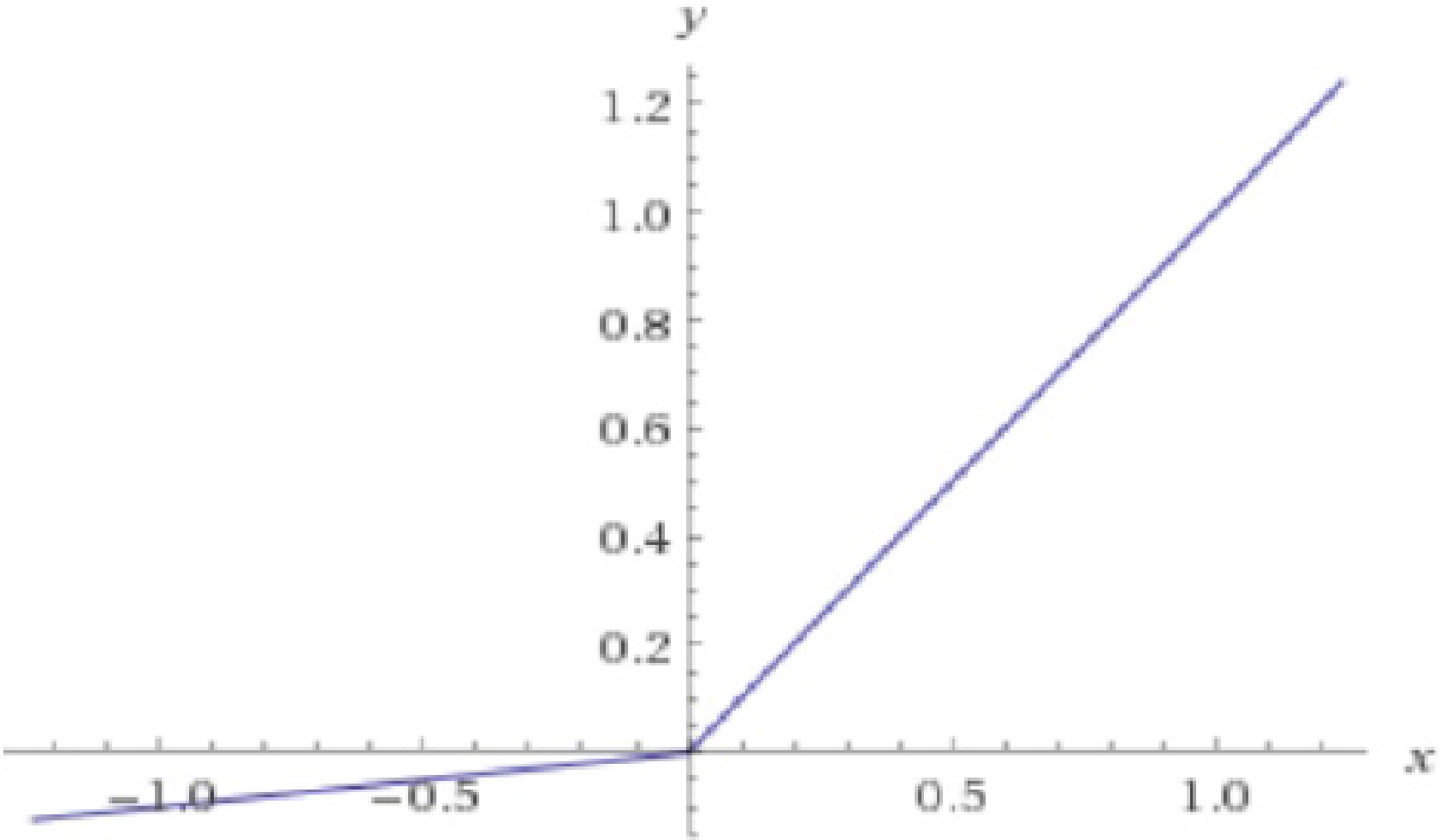
Activation function

y의 범위를 확장하기 위하여 RELU(ectified linear unit) function 사용



Activation function

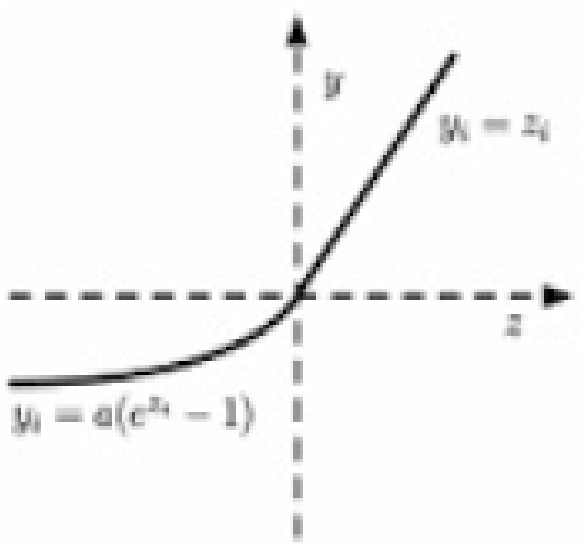
RELU함수의 약점을 보완하기 위해 leakly RElu



Activation function

exponential linear unitl(elu) function

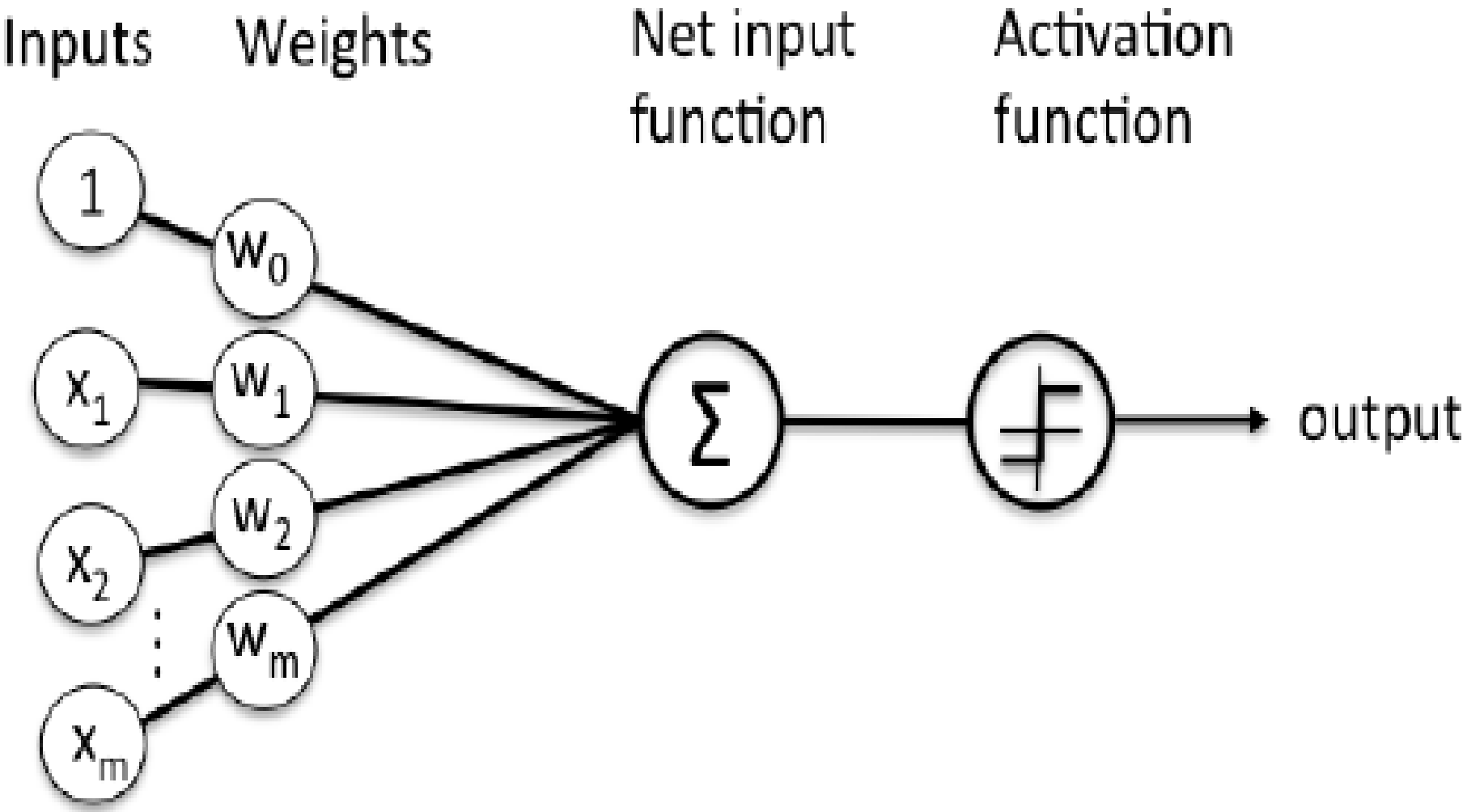
$$y_i = \begin{cases} z_i, & z_i \geq 0 \\ \alpha(\exp(z_i) - 1), & z_i \geq 0 \end{cases}$$



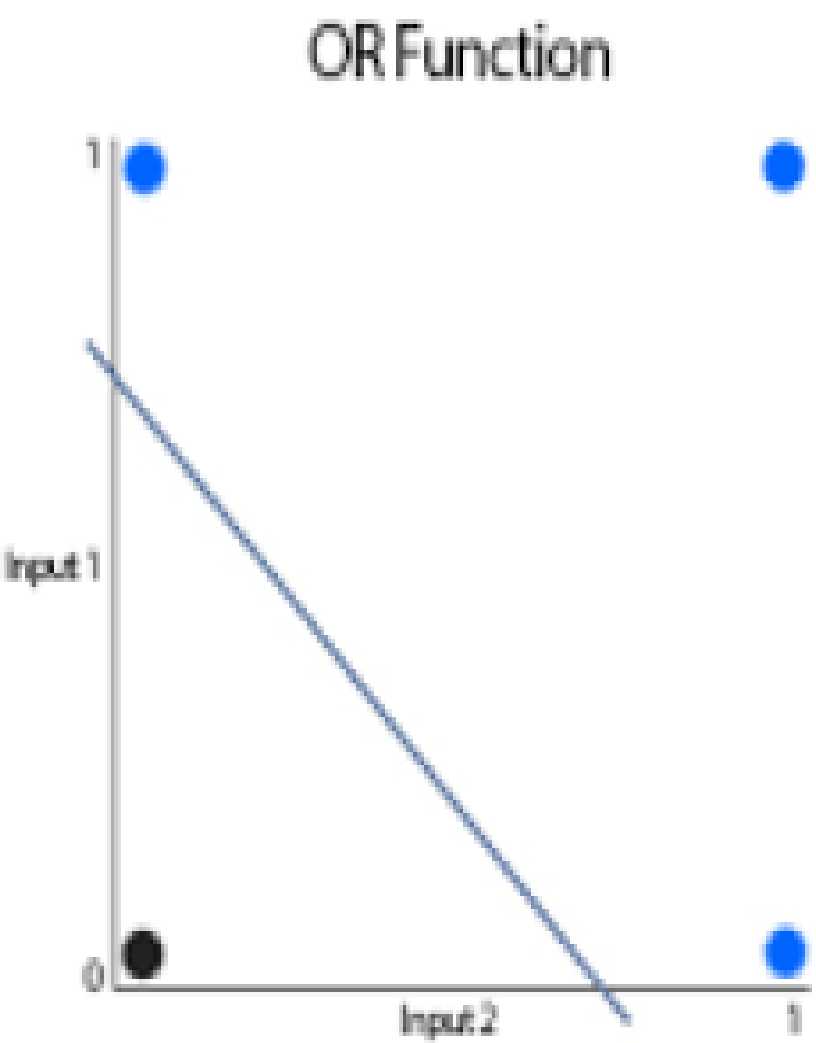
Activation function relation

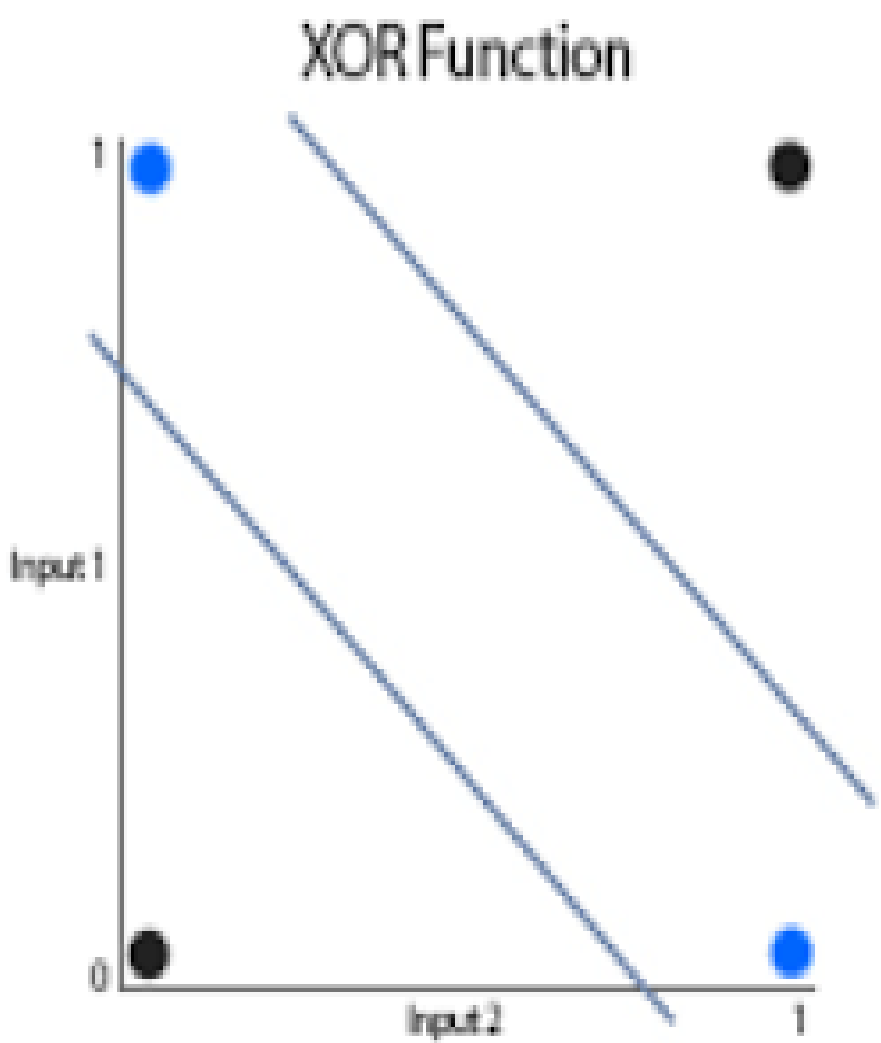
In geneal, ELU $>$ leaky RELU(and its variant) $>$ RELU $>$ tanh $>$ logistic

Perceptron


Inputs
Weights
Net input function
Activation function
1
x_1
x_2
x_m
w_0
w_1
w_2
w_m
Σ
output

단일 퍼셉트론의 한계


OR Function
1
0
Input 1
Input 2
1

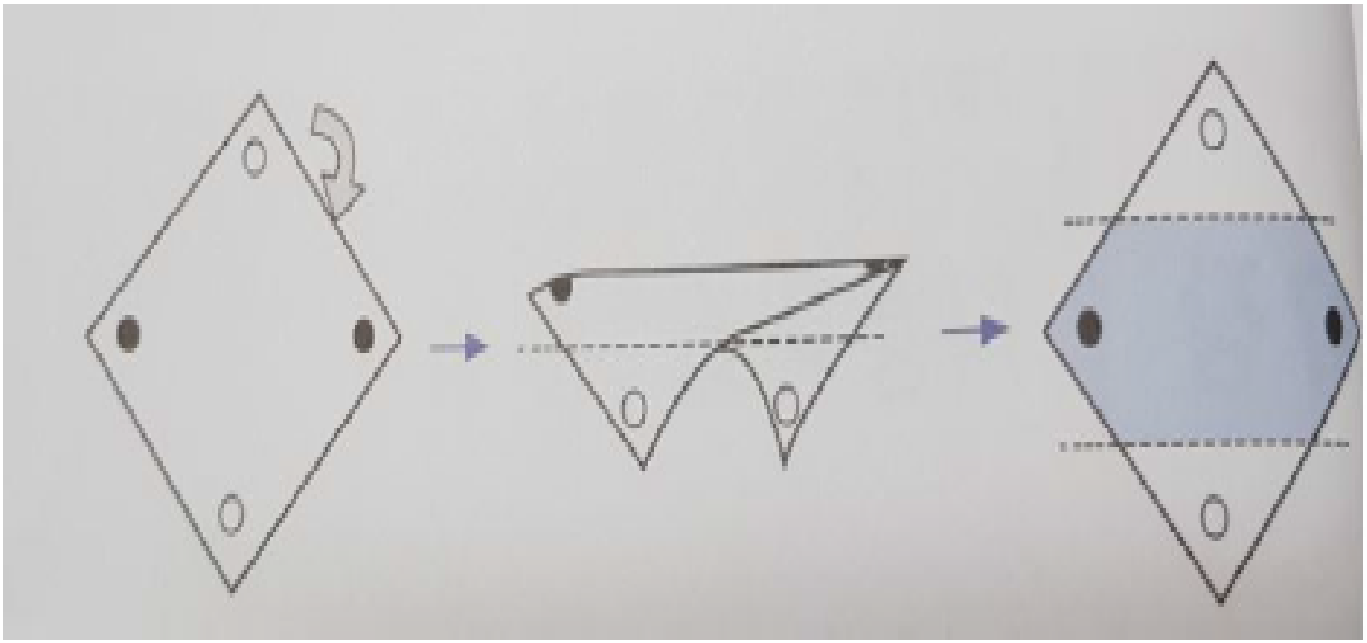

XOR Function
1
0
Input 1
Input 2
1

단일 퍼셉트론의 한계

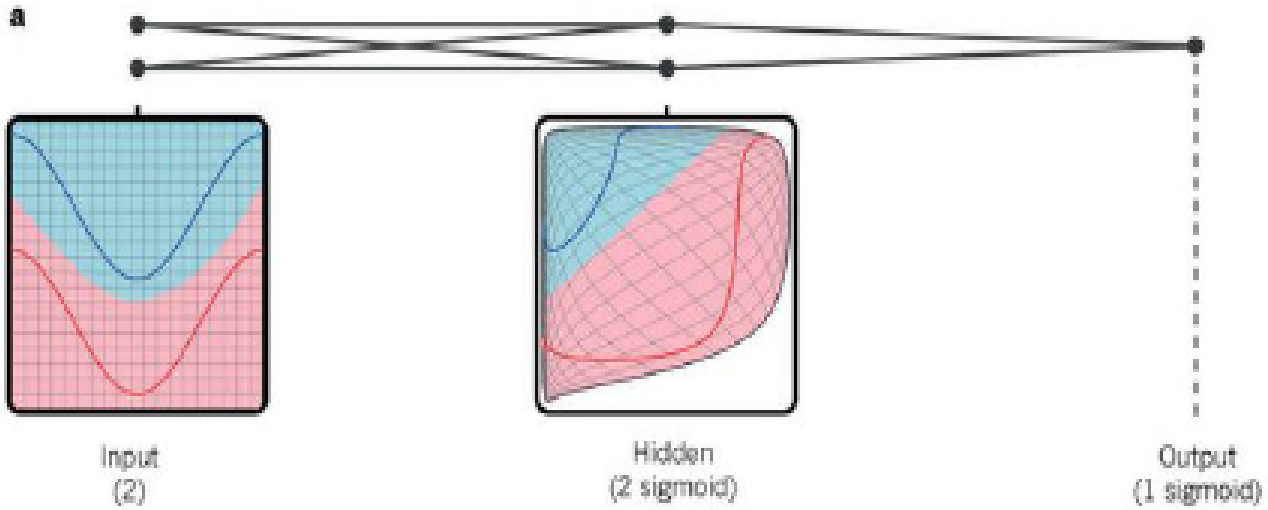
- AND와 OR 게이트는 직선을 그어 결괏값이 1인 값을 구별할 수 있었다.
- 하지만 XOR 게이트는 구별 불가
- 이는 인공지능 선구자였던 MIT의 마빈 민스키(Marvin Minsky)교수가 1969년에 발표한 ¡퍼셉트론즈(perceptrons)¿라는 논문에 나오는 내용
- '뉴런 → 신경망 → 지능'의 도식을 따라 '퍼셉트론 → 인공신경망 → 인공지능' 가능 하리라 꿈구던 당시 사람들은 이것이 생각처럼 쉽지 않다는 사실을 깨닫게 됨. 간단한 XOR문제조차 해결 할수 없기 때문. 10년 이후에 이 문제가 해결되는데, 이를 해결한 개념이 바로 다층 퍼셉트론(Multilayer perceptron)

단일 퍼셉트론의 한계

In XOR problem,

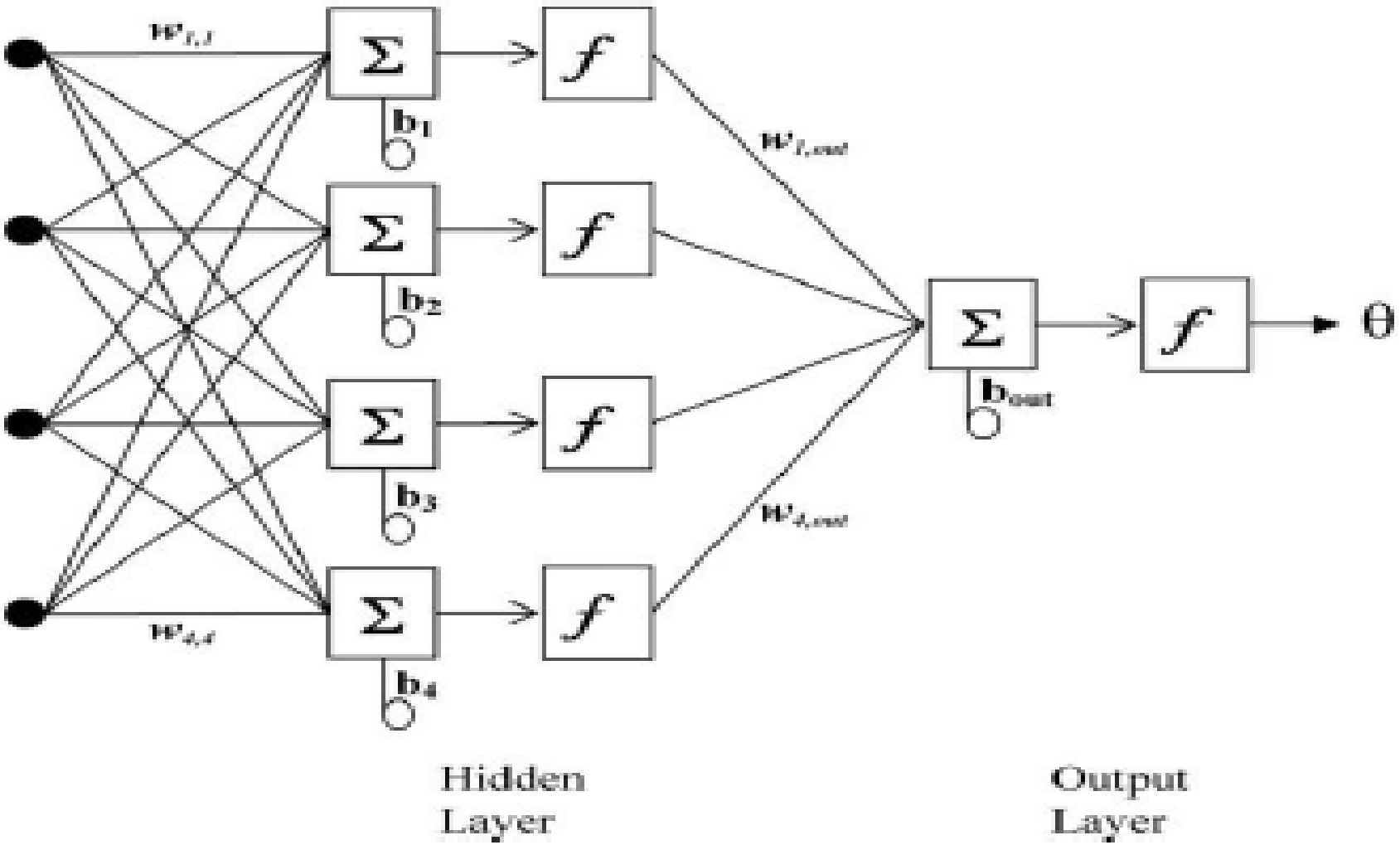


단일 퍼셉트론의 한계

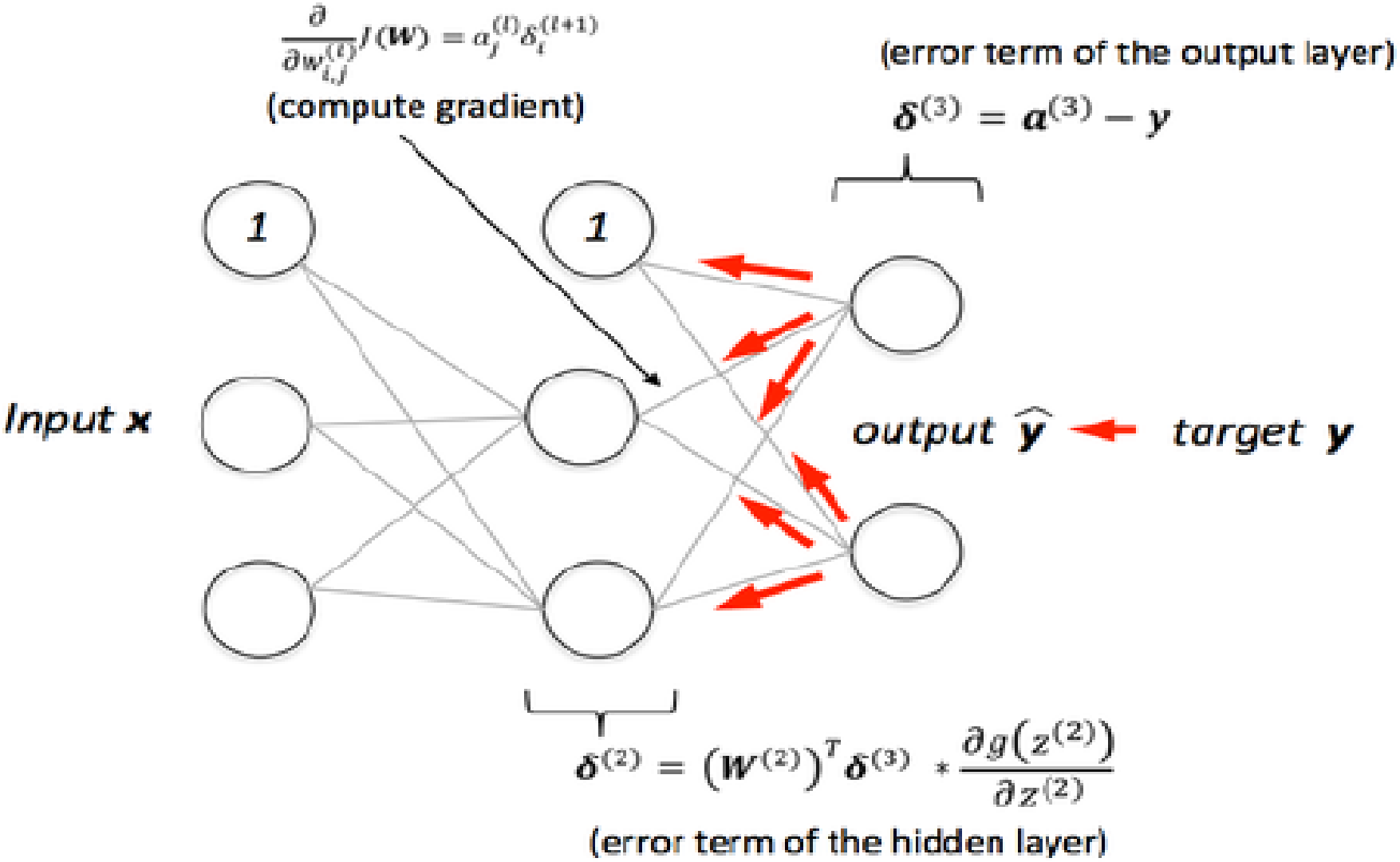


nature, Deep learning (Yann LeCun, Yoshua Bengio & Geoffrey Hinton)

다층 퍼셉트론



다층 퍼셉트론


$\frac{\partial}{\partial w_{i,j}^{(l)}} J(W) = a_j^{(l)} \delta_i^{(l+1)}$
(compute gradient)
(error term of the output layer)
$\delta^{(3)} = a^{(3)} - y$
1
1
Input x
output $\hat{y}$
target y
$\delta^{(2)} = (W^{(2)})^T \delta^{(3)} * \frac{\partial g(z^{(2)})}{\partial z^{(2)}}$
(error term of the hidden layer)

오차 역전파

- 임의의 가중치를 선언하고 최소제곱법을 이용해 오차를 구한 뒤 이 오차가 최소인 지점으로 계속하여 조금씩 이동.

- 이 오차가 최소가 되는 점(미분하였을때 기울기가 00이되는점)을 찾으면 그것이 우리가 알고자 하는 답이다.

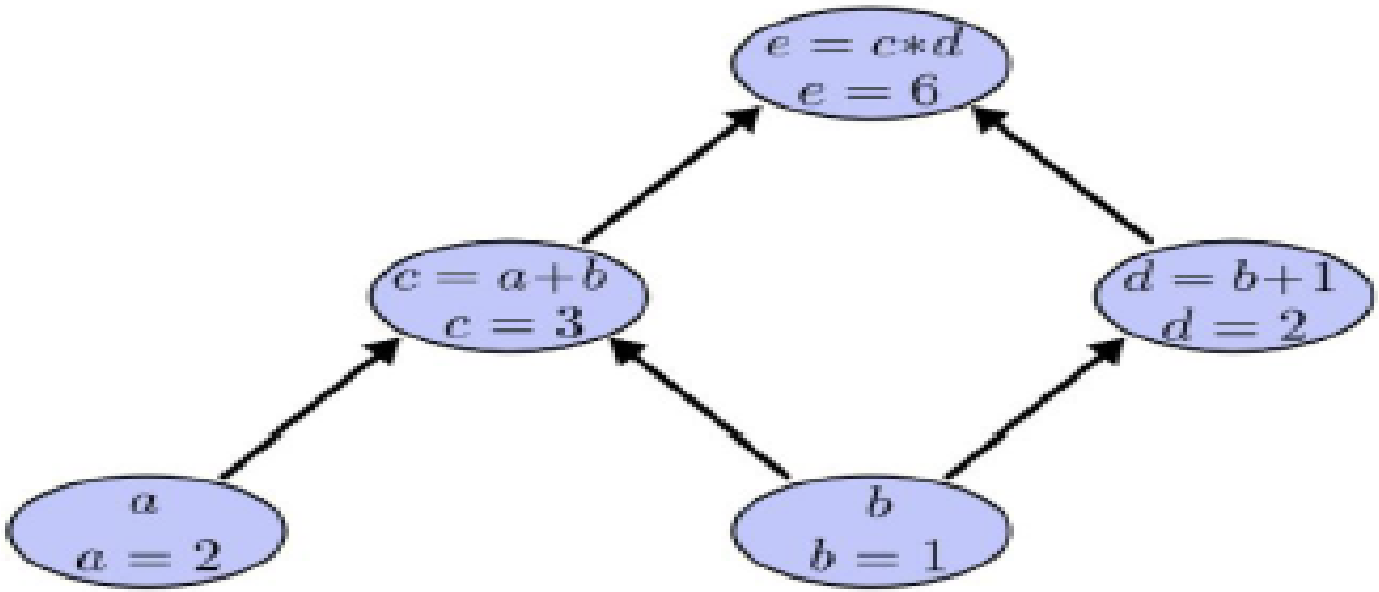
- 단층 퍼셉트론이 아닌 다층 퍼셉트론이므로, 계산이 좀 더 복잡해짐. 원리는 크게 다르지 않음.

Backpropargation introduction

- 역전파(Backpropagation)는 딥 모델을 컴퓨터연산으로 추적가능하게 훈련시키는 핵심 알고리즘
- 현대 뉴럴 네트워크를 위해서 기울기 하강(Gradien descent)으로 단순한 알고리즘에 비해 천만배 이상 빠르게 학습을 시킬 수 있게 됨.

Computational Graphs

계산 그래프들은 수학적인 표현을 생각하기 위한 멋진 방법.



Partial Differention

- 편미분 이란 ?

미분과 편미분은 모두 '미분하라'는 의미에서는 다를바가 없다. 그러나 여러가지 변수가 식에 있을 때는 모든 변수를 미분하는 것이 아니라, 우리가 원하는 한가지 변수만 미분하는 것을 말한다.

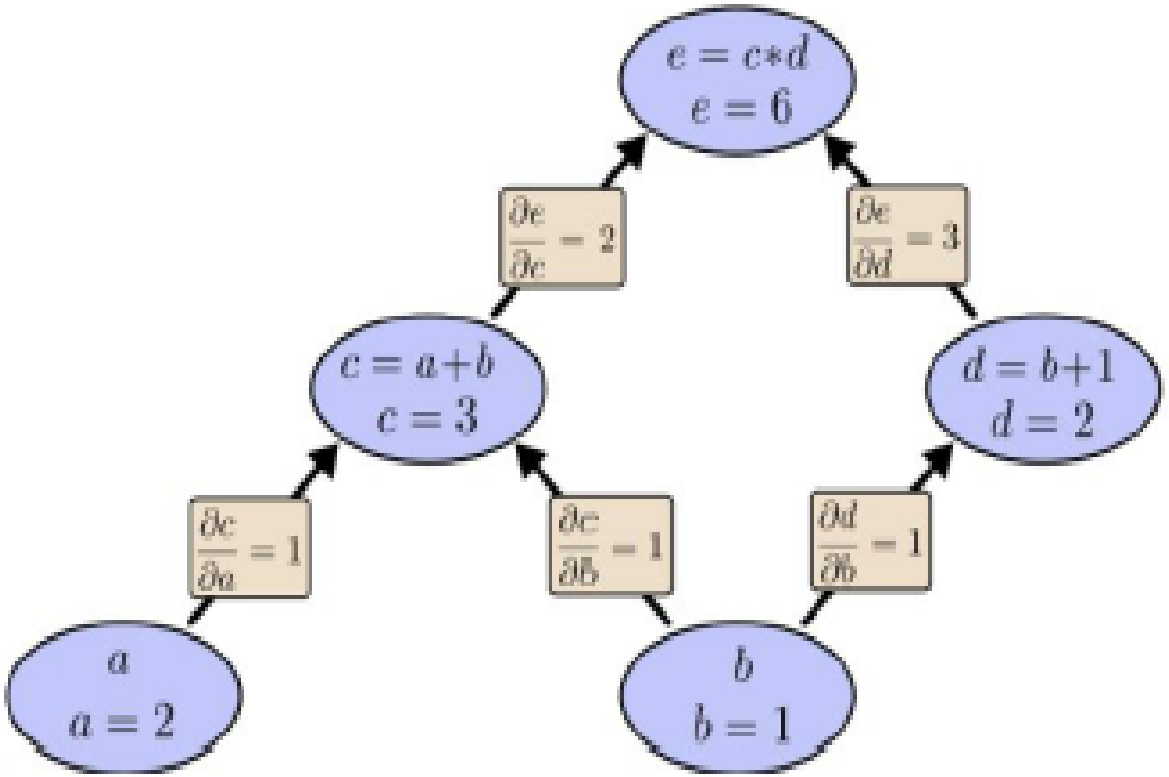
$$\text{수학 표현식} : \frac{\partial f}{\partial x}$$

ex) $f(x,y) = x^2 + xy + a$ (a는 상수) 일 때,

$$\frac{\partial f}{\partial x} = 2x + y$$

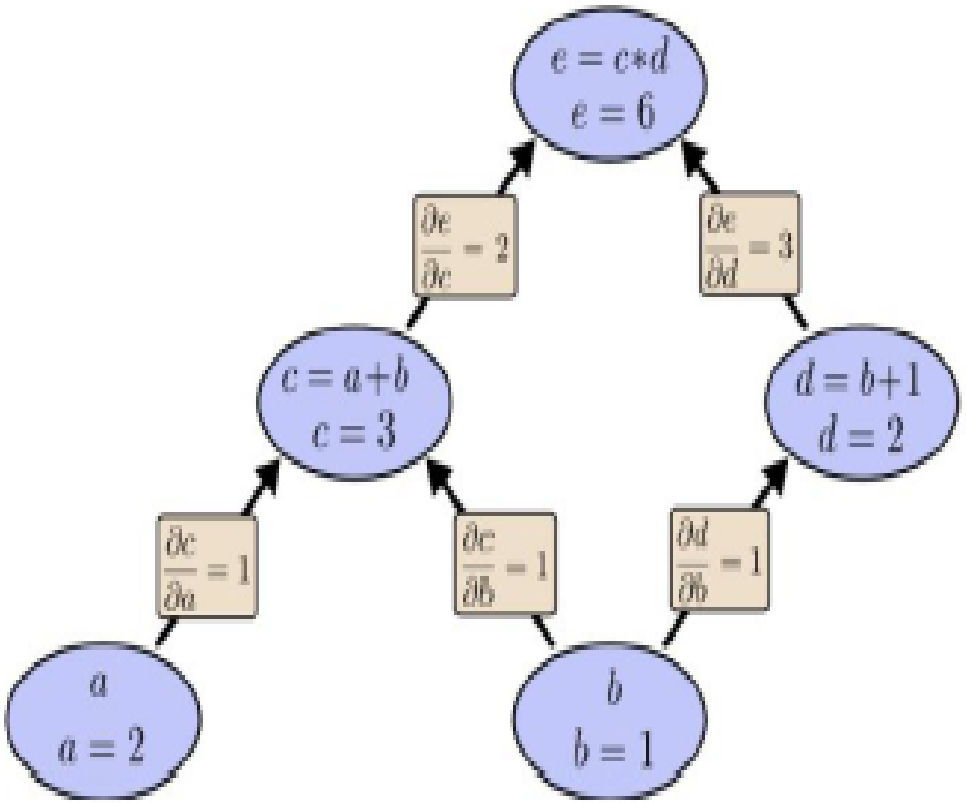
Partial Differention

- If one wants to understand derivatives in a computational graph, the key is to understand derivatives on the edges.



Derivatives on Computational Graphs

- If one wants to understand derivatives in a computational graph, the key is to understand derivatives on the edges.



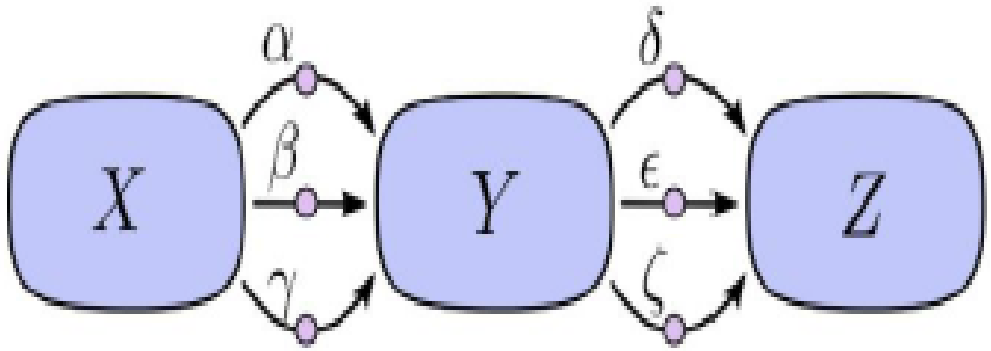
Derivatives on Computational Graphs

- 직접 연결되지 않은 노드가 서로 어떻게 영향을 미치는지 이해하려면 어떻게 해야될까?
- 먼저, e가 어떻게 영향을 받는지 생각해보자.
- 우리가 a의 1의 속도로 변경하면 c는 1의 속도로 변경되고, 차례로 c의 속도가 1로 변경되면 e가 2의 속도로 변경이 됩니다.
- 일반적인 규칙은 한 노드에서 다른 노드로가는 모든 가능한 경로를 합하여 경로의 각 가장자리에 있는 도함수(Derivates)을 함께 곱하는 것입니다.
- 예를 들어, b와 관련하여 e의 도함수(Derivates)를 얻으려면 다음을 얻을수 있다.

$$\frac{\partial e}{\partial b} = 1*2+1*3$$

- 이 일반적인 "경로를 통합 합계" 규칙은 Multivariate chain rule 에 대한 생각의 다른 방식일뿐.

Factoring paths



$$\frac{\partial Z}{\partial X} = \alpha\delta + \alpha\epsilon + \alpha\zeta + \beta\delta + \beta\epsilon + \beta\zeta + \gamma\delta + \gamma\epsilon + \gamma\zeta$$

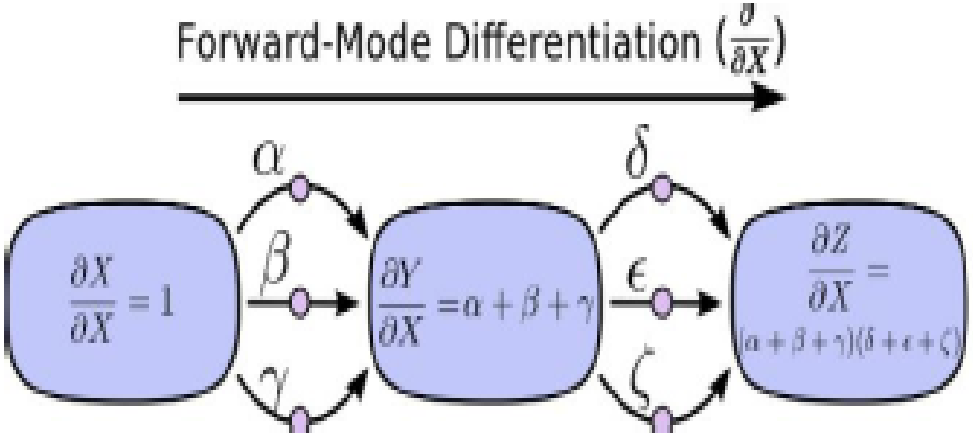
- 위의 경로는 9개 뿐이지만, 그래프가 복잡해질수록 경로 수가 기하급수적으로 늘어남
- 경로를 순진하게 합산하는 대신, 그것들을 고려하는 것이 훨씬 낫다.

Factoring paths

- $$\frac{\partial Z}{\partial X} = (\alpha + \beta + \gamma)(\delta + \epsilon + \zeta)$$
- 이것은 "순방향 미분(Forward-mode differentiaion)"와 "역방향 미분(Backward-mode differentiaion)"가 들어간 곳
- 경로를 인수 분해하여 합계를 효율적으로 계산하기 위한 알고리즘.
- 모든 경로를 명시적으로 합산하는 대신 모든 모드에서 경로를 다시 병합하여 동일한 합계를 더 효율적으로 계산
- 두 알고리즘은 각 노드에 정확히 한 번 씩 만지게 됨.

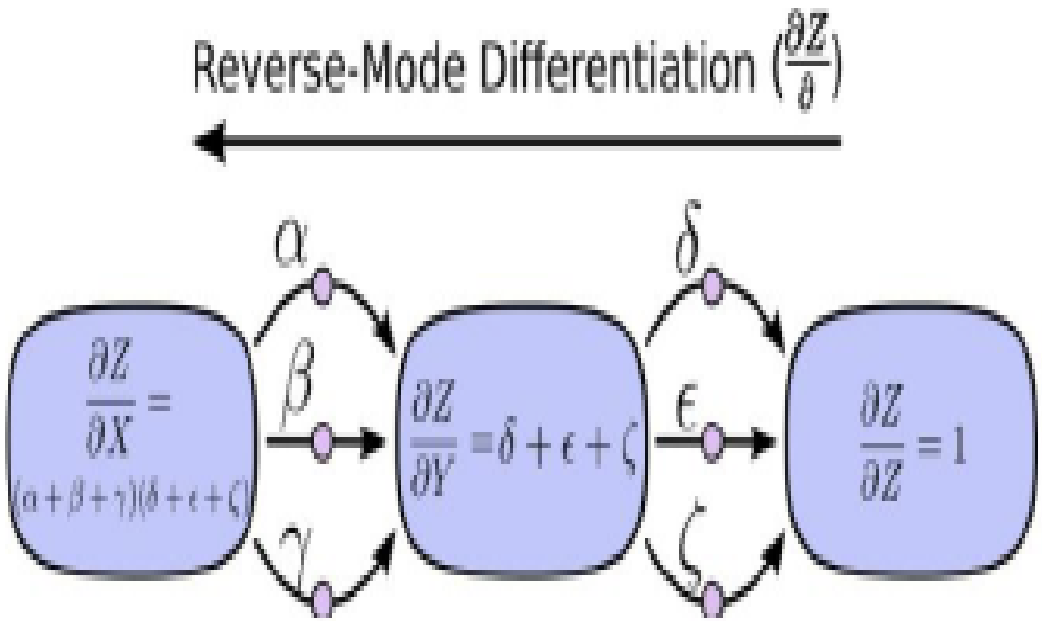
Factoring paths

- 순방향 미분(Forward-mode differentiation)은 그래프의 입력에서 시작하여 끝으로 이동.
- 모든 노드에서 입력되는 모든 경로를 합산을 하고, 각 경로는 입력이 해당 노드에 영향을 주는 한 가지 방법을 나타냄.
- 노드를 추가함으로써 노드가 입력의 영향을 받는 총합을 얻습니다. 이 노드는 도함수(Derivates)



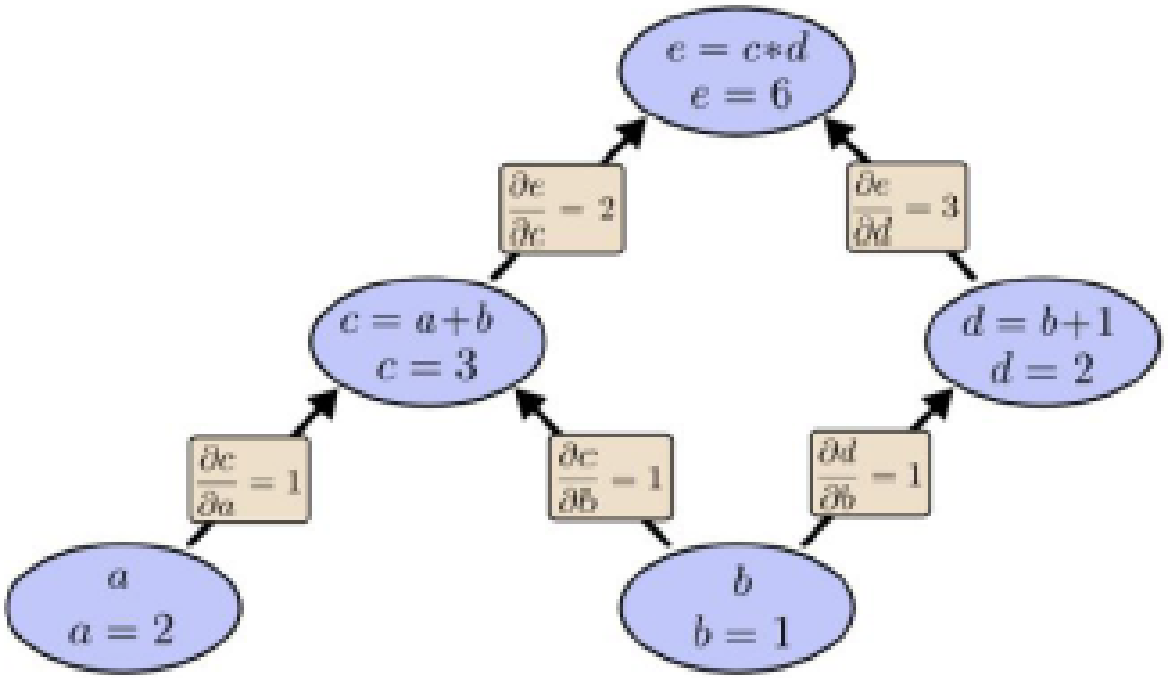
Factoring paths

- 반대로, 역방향 미분은 그래프 출력에서 시작하여 처음으로 이동
- 각 해당 노드에서 시작된 모든 경로를 병합



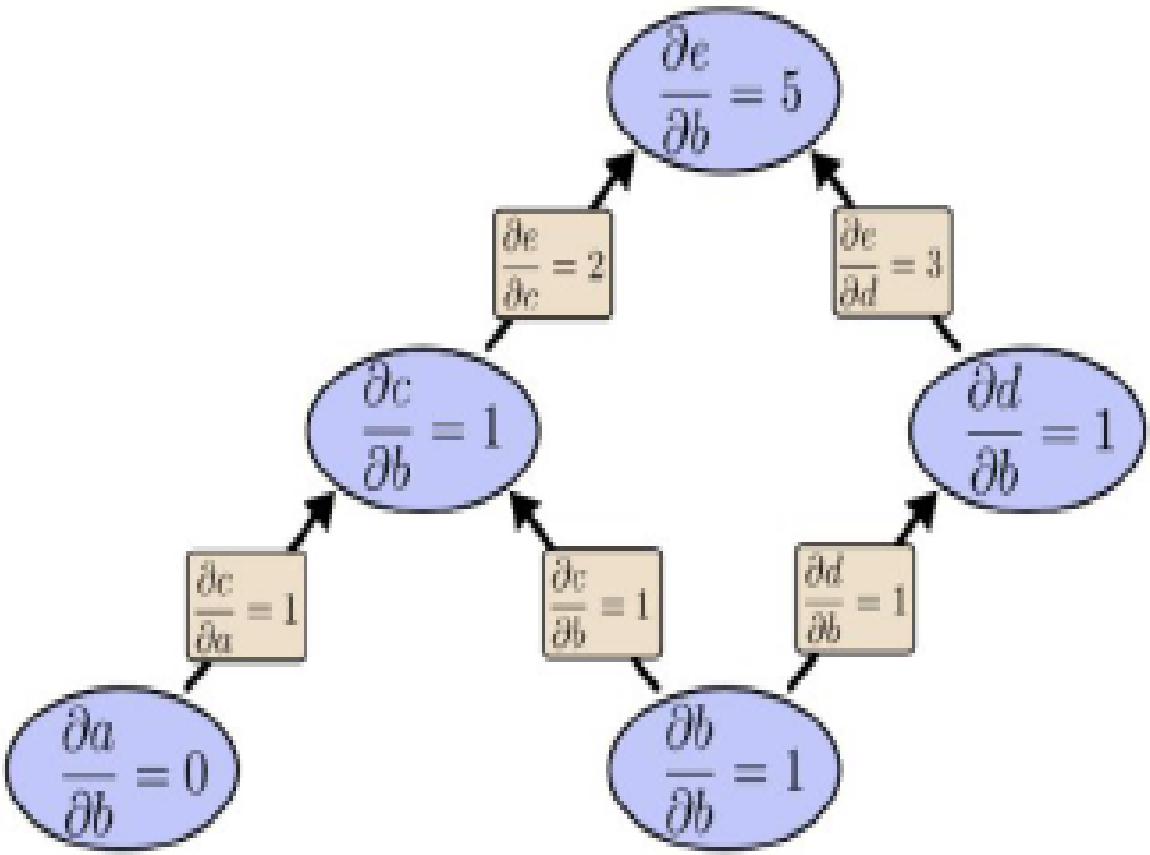
Computational Victories

- 이 시점에서, 왜 역방향 미분을 신경쓰고 있을지에 대한 고민이 생길 것입니다.
- 무슨 이점이 있을까?
- 원래 예제를 보면,



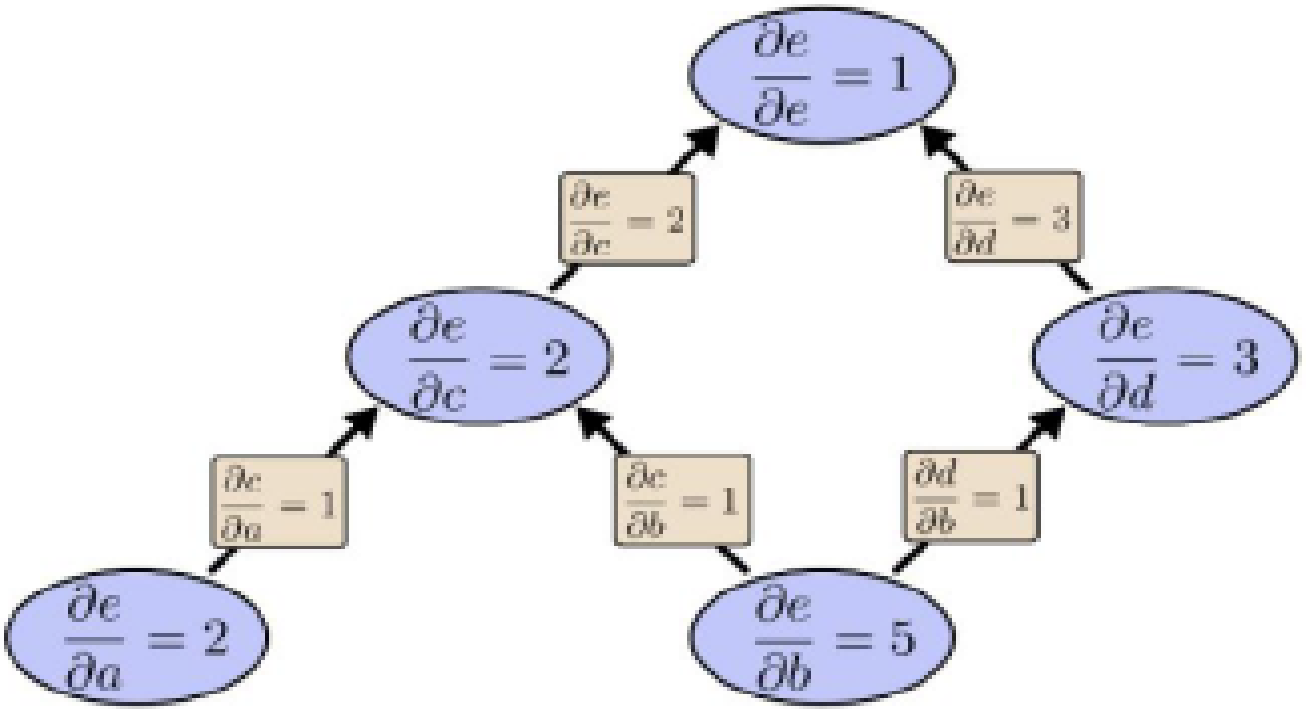
Computational Victories

- 우리는 b에서 순방향 미분을 사용할 수 있습니다. 이것은 우리에게 b에 관한 모든 노드의 미분을 제공



Computational Victories

- 입력 값 중 하나와 관련하여 출력 값의 미분 $\frac{\partial e}{\partial b}$ 를 계산
- 만약, e에서 역방향 미분을 한다면?? 이렇게하면 모든 노드와 관련하여 e의 도함수(Derivates)를 얻을 수 있습니다.



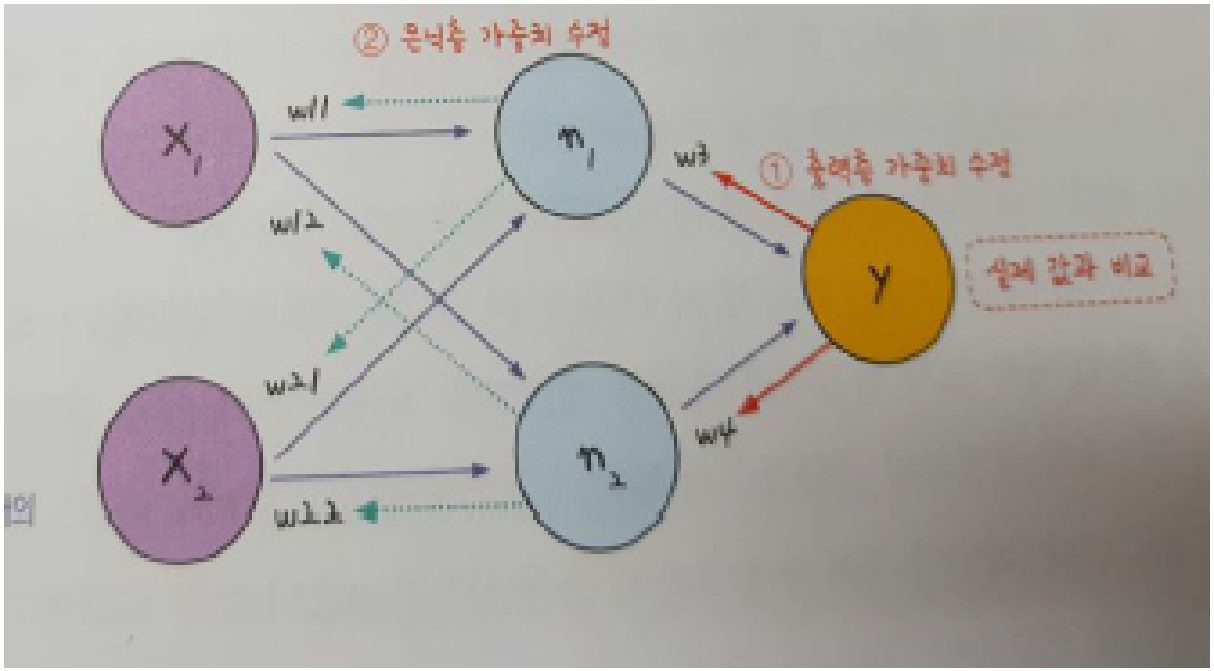
Computational Victories

- 역방향 미분이 모든 노드와 관련하여 e의 도함수(Derivates)를 제공한다고 말하면 모든 노드를 의미
- 양쪽 입력에 대해 e의 도함수인 $\frac{\partial e}{\partial a}$와 $\frac{\partial e}{\partial b}$를 얻을 수 있다.
- 순방향 미분은 단일 입력과 관련하여 산출물의 도함수를 나타내지만 역방향 미분은 우리에게 모든 것을 제공
- 단지 두가지 속도 향상 요인일 뿐이지만 백만개의 입력과 하나의 출력이 있는 함수를 생각해보자.
- 순방향 미분은 도함수를 얻기 위해 그래프를 100만 회를 통과해야 되지만, 역방향 미분을 통해 한꺼번에 모든것을 얻을수 있다.

Conclusion

- 신경망을 훈련 할때 우리는 매개변수(네트워크가 어떻게 행동하는지를 나타내는 숫자)의 함수로써 Loss(cost, 신경망이 얼마나 나쁜지를 나타내는 값을) 생각.
- 모든 매개 변수와 관련하여 비용의 파생어를 계산하여 Gradient Descent에 사용하려고 함
- 우리는 좀더 Gradient Desecent Optimzation Algoritms에 공부를 해보도록 하자.
- Stochastic gradient descent, Momentum, Nesterov accelerated graduent(NAG),Adagrad.Adadelta,RMSprop,Adaptive Moment Estimation(ADAM)

Backpropargation



오차 역전파 구동 방식은 다음과 같이 정리

1. 임의의 초기 가중치(w_1)를 준 뒤 (y_{out})를 계산
2. 계산 결과와 우리가 원하는 값 사이의 오차를 구함
3. 경사 하강법을 이용해 바로 앞 가중치를 오차가 작아지는 방향으로 업데이트를 함.
4. 1 3과정을 더이상 오차가 줄어들지 않을 때까지 반복.

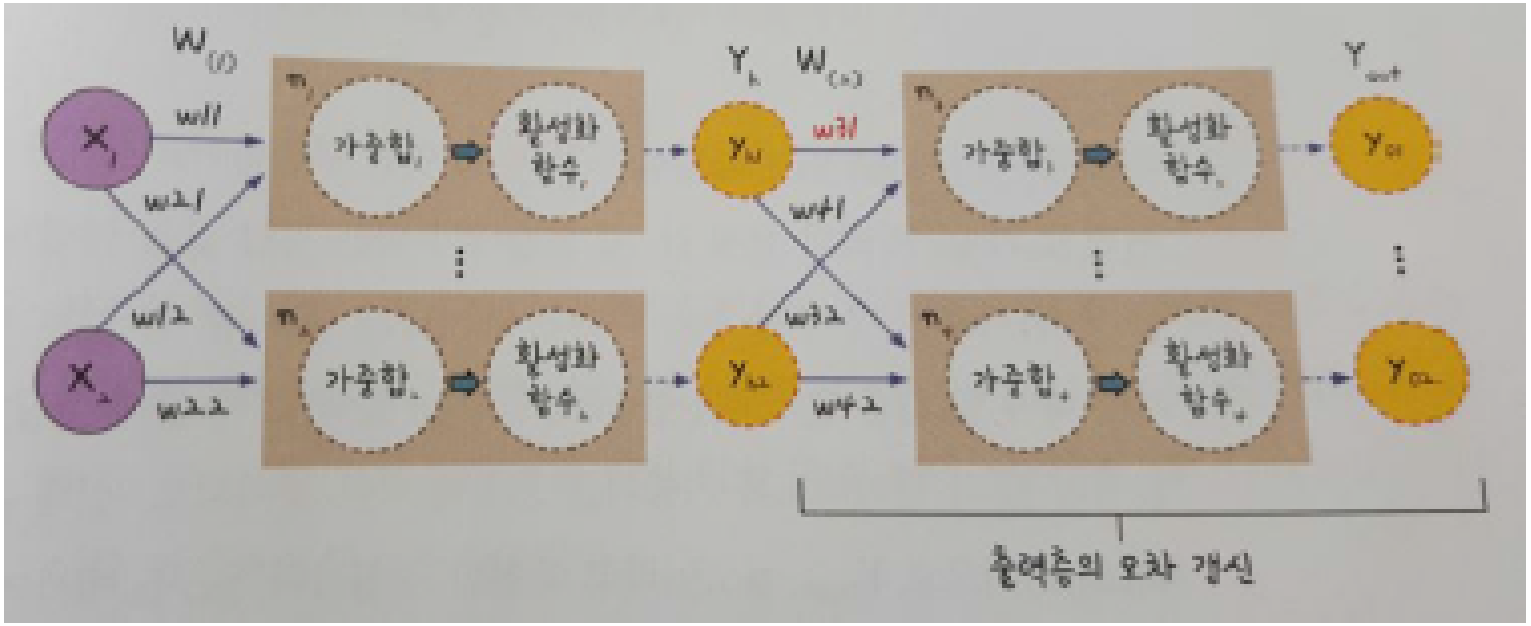
Backpropargation

- '오차가 작아지는 방향으로 업데이트를 한다

$\longrightarrow$ '미분값이 0에 가까워 지는 방향

$$W_{t+1} = W_t - \frac{\partial(error)}{\partial W}$$

Backpropargation



$$W_{31}(t+1) = W_{31}t - \frac{\partial(error\_Y_{out})}{\partial w_{31}}$$

w_{31}값은 이미 알고 있음
우리가 구해야 하는 값은

$$\frac{\partial(error\_Y_{out})}{\partial w_{31}}$$

오차공식

오차 Y_{out} 안에는 두개 (y_{o1}, y_{o2}) 의 출력값이 있음.

즉, $error\_Y_{out} = error\_y_{o1} + error\_y_{o2}$

여기서 $error_$과 $error\_y_{o2}$ 는 각각 앞서 배운 평균 제곱 오차를 이용해 구함.

$$error\_y_{o1} = \frac{1}{2}(y_{t1} - y_{o1})^2 \ , \ error\_y_{o2} = \frac{1}{2}(y_{t2} - y_{o2})^2$$

$$error\_Y_{out} = \frac{1}{2}(y_{t1} - y_{o1})^2 + \frac{1}{2}(y_{t2} - y_{o2})^2$$

오차공식

$$\frac{\partial(error\_Y_{out})}{\partial w_{31}} = \frac{\partial(error\_Y_{out})}{\partial y_{o1}} \cdot \frac{\partial y_{o1}}{\partial(Weightsum)_3} \cdot \frac{\partial(Weightsum)_3}{\partial w_{31}}$$

체인 룰은 연쇄법칙이라고 부름. '합성 함수'를 미분할 때의 계산 공식.
여기 함성 함수란, 함수 안에 또 다른 함수가 들어 있는 것을 말함.
$[f(g(x))]' = f'(g(x))g'(x)$ 다음과 같이 표시하면,

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial g}\frac{\partial g}{\partial x}$$

여기서,체인 룰, 즉 연쇄법칙이라고 부르는 이유가 나옴. 위의 식에서 dy 라고 하는 항이 분모와 분자로 고리처럼 연속적으로 이어져 나오기 때문.
만약, 합성함수 식이 3개라면?? $f(g(h(x)))'$ 일 때,
이를 미분하면,

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial g}\frac{\partial g}{\partial h}\frac{\partial h}{\partial x}$$

오차공식

$$\frac{\partial(error)Y_{out}}{\partial y_{o1}} = \frac{\partial(error\_y_{o1}) + \partial(error\_y_{o2})}{\partial y_{o1}}$$

$$= \frac{\partial \frac{1}{2}(y_{t1} - y_{o1})^2}{\partial y_{o1}} + \frac{\partial \frac{1}{2}(y_{t2} - y_{o2})^2}{\partial y_{o1}}$$

$$= \frac{\partial}{\partial y_{o1}} \frac{1}{2}(y_{t1} - y_{o1})^2$$

$$= (y_{t1} - y_{o1})(-1)$$

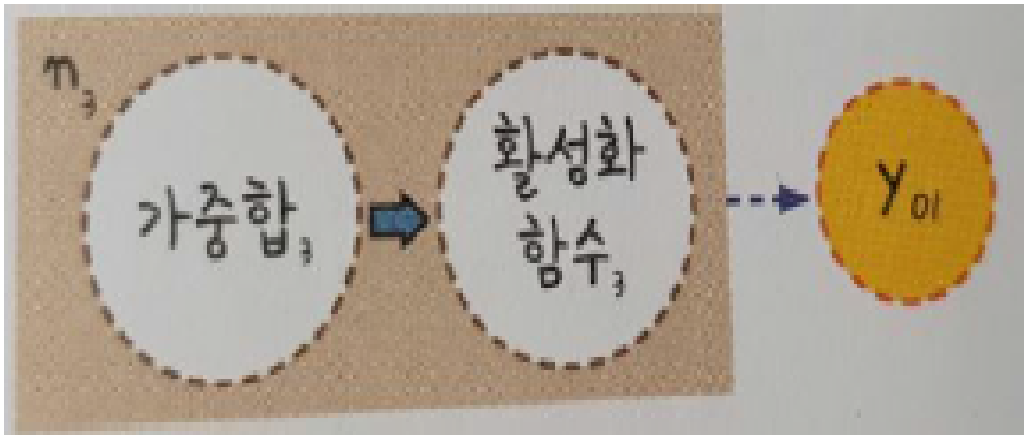
$$= (y_{o1} - y_{t1})$$

정리를 하면,

$$\frac{\partial(error\_Y_{out})}{\partial y_{o1}} = (y_{o1} - y_{t1})$$

오차공식

$$\frac{\partial y_{o1}}{\partial (Weightsum)_3}$$



$$\frac{\partial y_{o1}}{\partial (Weightsum)_3} = \text{활성화 함수}_3 \text{의 미분}$$

오차공식

활성화 함수는 여러가지있지만, 시그모이드 함수를 사용

$$\frac{1}{1+e^{-x}}$$

$$\frac{d\sigma(x)}{dx} = \sigma(x)\cdot(1-\sigma(x))$$

prove)
$$\begin{aligned}
\frac{d\sigma(x)}{dx} &= \frac{d}{dx}[\frac{1}{1+e^{-x}}] \\
&= \frac{d}{dx}(1+e^{-x})^{(-1)} \\
&= -(1+e^{-x})^{-2}(-e^{-x}) \\
&= \frac{(-e^{-x})}{(1+e^{-x})^2} \\
&= \frac{1}{(1+e^{-x})}\frac{-e^{-x}}{(1+e^{-x})} \\
&= \frac{1}{(1+e^{-x})}\frac{1-(1+e^{-x})}{(1+e^{-x})} \\
&= \sigma(x)\cdot(1-\sigma)
\end{aligned}$$

오차공식

$$\frac{\partial y_{o1}}{\partial (Weightsum)_3} = \text{활성화 함수}_3 \text{ 의 미분} = y_{o1} \cdot (1 - y_{o1})$$

$$\frac{\partial Weightsum_3}{\partial w_{31}}$$

$$wightsum_3 = n_1 + n_2 = w_{31} y_{h1} + w_{41} y_{h2} + 1(\text{bias})$$

$$\frac{\partial Weightsum_3}{\partial w_{31}} = y_{h1}$$

$$\frac{\partial (error\_Y_{out})}{\partial w_{31}} = \frac{\partial (error\_Y_{out})}{\partial y_{o1}} \cdot \frac{\partial y_{o1}}{\partial (Weightsum)_3} \cdot \frac{\partial (Weightsum)_3}{\partial w_{31}}$$

$$= (y_{o1} - y_{t1}) \cdot y_{o1}(1 - y_{o1}) \cdot y_{h1}$$

가중치 업데이트하기

In W_{31} 의 입장에서,

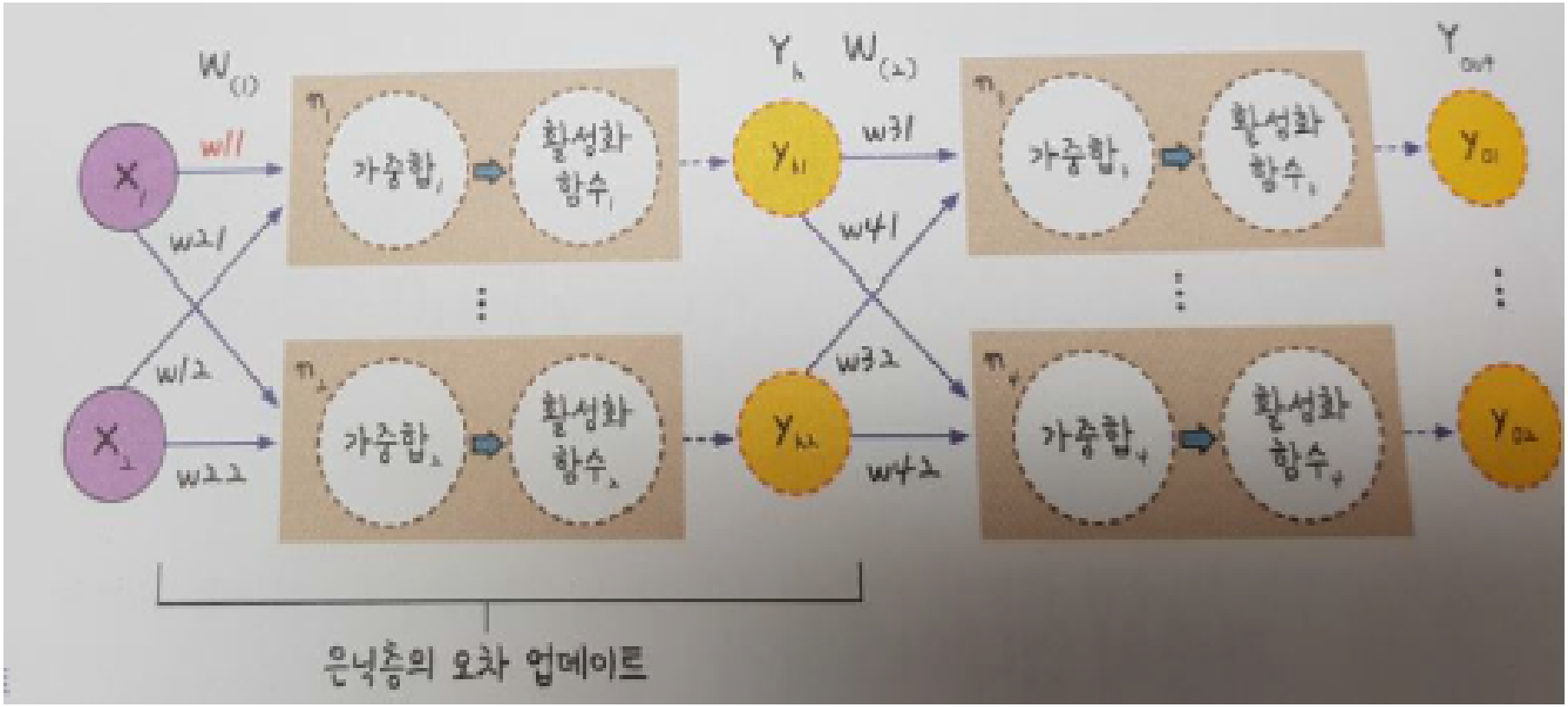
$$w_{31}(t+1) = w_{31}t - (y_{o1} - y_{t1}) \cdot y_{o1}(1 - y_{o1}) \cdot y_{h1}$$

이 형태는 다음 오차 업데이트 때도 반복하여 나타남. 따라서 이 식을 한 번 구해 놓으면 이후는 그대로 사용해서 오차를 구함.

이를 node3의 **델타(delta)식** 이라고 함

$$w_{31}(t+1) = w_{31}t - \delta y_{o1} \cdot y_{h1}, \quad \delta y_{o1} = (y_{o1} - y_{t1}) \cdot y_{o1}(1 - y_{o1})$$

가중치 업데이트하기



$$\ln W_{11} = W_{11}t - \frac{\partial(error\_Y_{out})}{\partial w_{11}}$$

가중치 업데이트하기

$$\frac{\partial(error\_Y_{out})}{\partial w_{11}} = \frac{\partial(error\_Y_{out})}{\partial y_{h1}} \cdot \frac{\partial y_{h1}}{\partial weightsum_1} \cdot \frac{\partial weightsum_1}{\partial w_{11}}$$

$$\frac{\partial y_{h1}}{\partial weightsum_1} \cdot \frac{\partial weightsum_1}{\partial w_{11}} = y_{h1}(1 - y_{h1}) \cdot x_1$$

$$\frac{\partial(error\_Y_{out})}{\partial y_{h1}} = \frac{\partial(error\_y_{o1}) + (error\_y_{o2})}{\partial y_{h1}} = \frac{\partial(error\_y_{o1})}{\partial y_{h1}} + \frac{\partial(error\_y_{o2})}{\partial y_{h1}}$$

a.

$$\frac{\partial(error\_y_{o1})}{\partial y_{h1}} = \frac{\partial errory_{o1}}{\partial y_{o1}} \cdot \frac{\partial y_{o1}}{\partial weighsum_3} \cdot \frac{\partial weightsum_3}{\partial y_{h1}}$$

$$\frac{\partial(error\_y_{o1})}{\partial y_{o1}} = (y_{o1} - y_{t1}) \ , \quad \frac{\partial y_{o1}}{\partial weightsum_3} = y_{o1} \cdot (1 - y_{o1})$$

$$\frac{\partial weightsum_3}{\partial y_{h1}} = W_{31}$$

가중치 업데이트하기

$$\frac{\partial(error\_y_{o1})}{\partial y_{h1}} = (y_{o1} - y_{t1}) \cdot y_{o1} \cdot (1 - y_{o1}) \cdot W_{31}$$

앞서 기억해 둔 델타식입니다. 간단히 표현하면,

$$\frac{\partial(error\_y_{o1})}{\partial y_{h1}} = \delta y_{o1} \cdot W_{31}$$

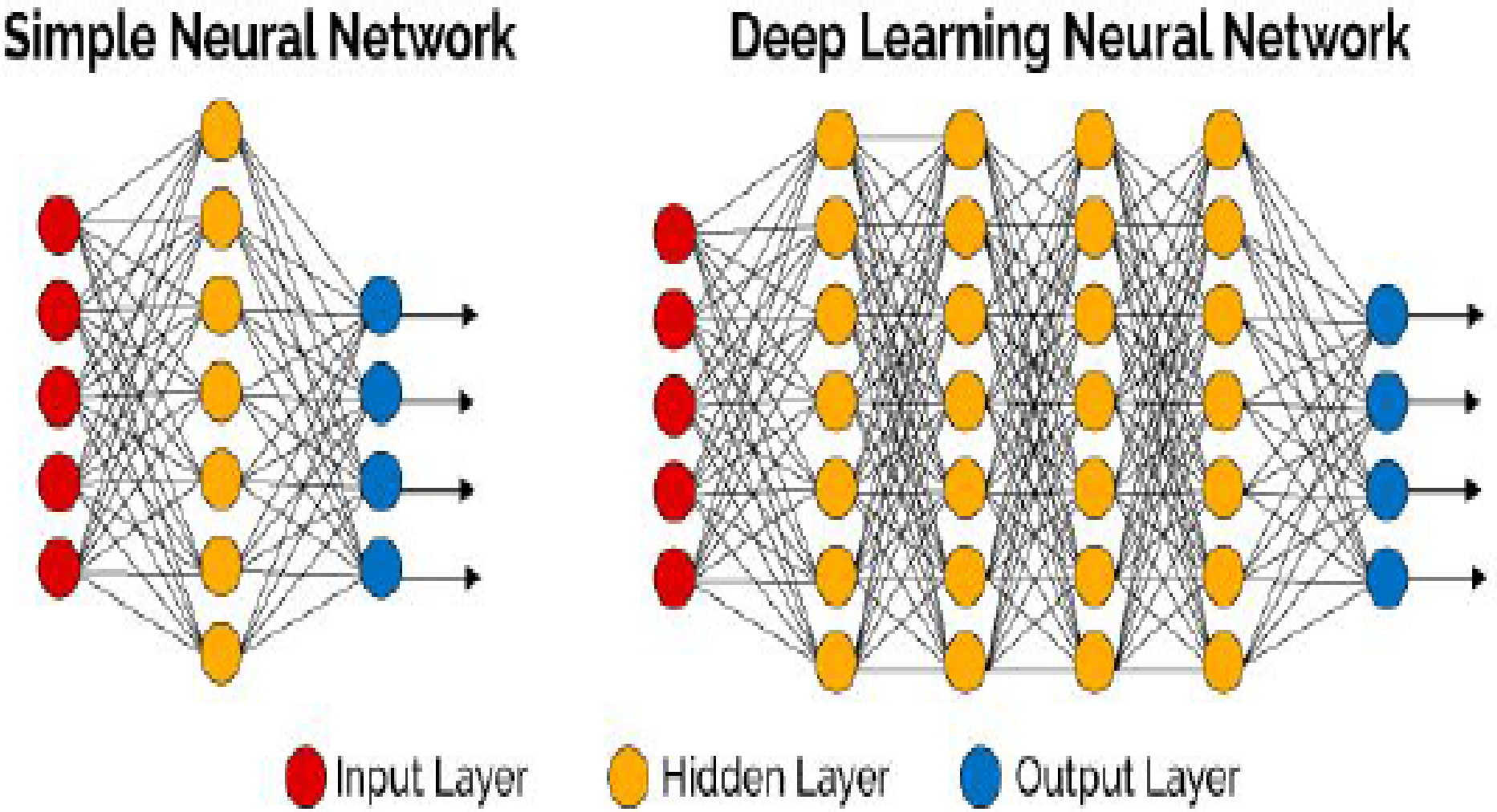
b. $$\frac{\partial(error\_y_{o2})}{\partial y_{h1}} = \frac{\partial(error\_y_{o2})}{\partial weightsum_4} \cdot \frac{\partial weightsum_4}{\partial y_{h1}}$$

$$= \frac{\partial(error\_y_{o2})}{\partial y_{o2}} \cdot \frac{\partial y_{o2}}{\partial weitghtsum_4} \cdot \frac{\partial weightsum_4}{\partial y_{h1}}$$

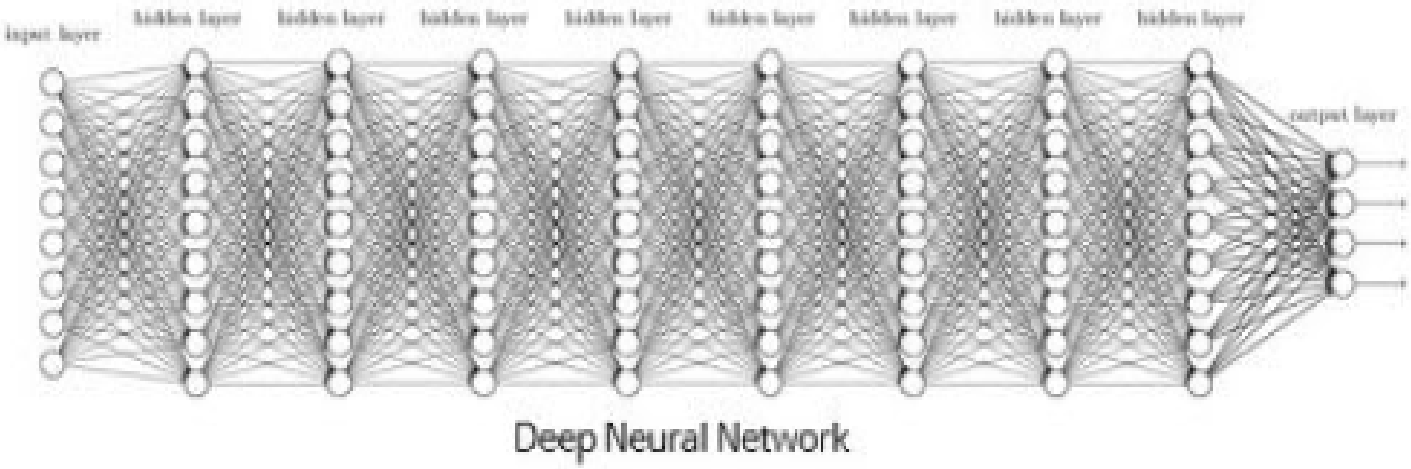
$$\frac{\partial(error)y_{o2}}{\partial y_{h1}} = (y_{o2} - y_{t2}) \cdot y_{o2}(1 - y_{o2}) \cdot W_{41}$$

$$\delta y_{o2} = (y_{o2} - y_{t2}) \cdot y_{o2}(1 - y_{o2})$$

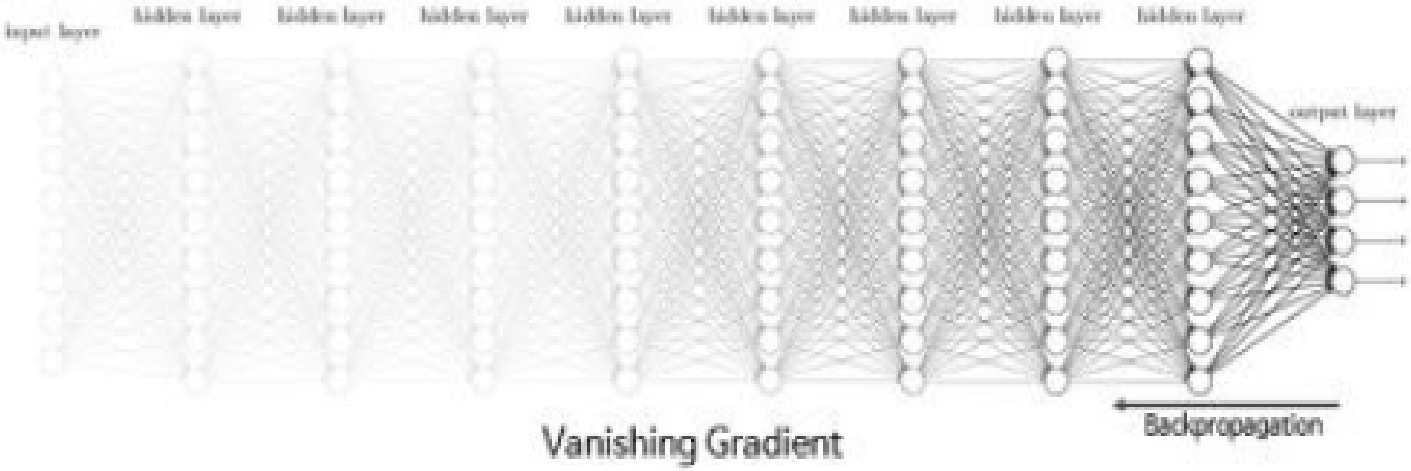
신경망에서 딥러닝


Simple Neural Network
Deep Learning Neural Network
Input Layer
Hidden Layer
Output Layer

신경망에서 딥러

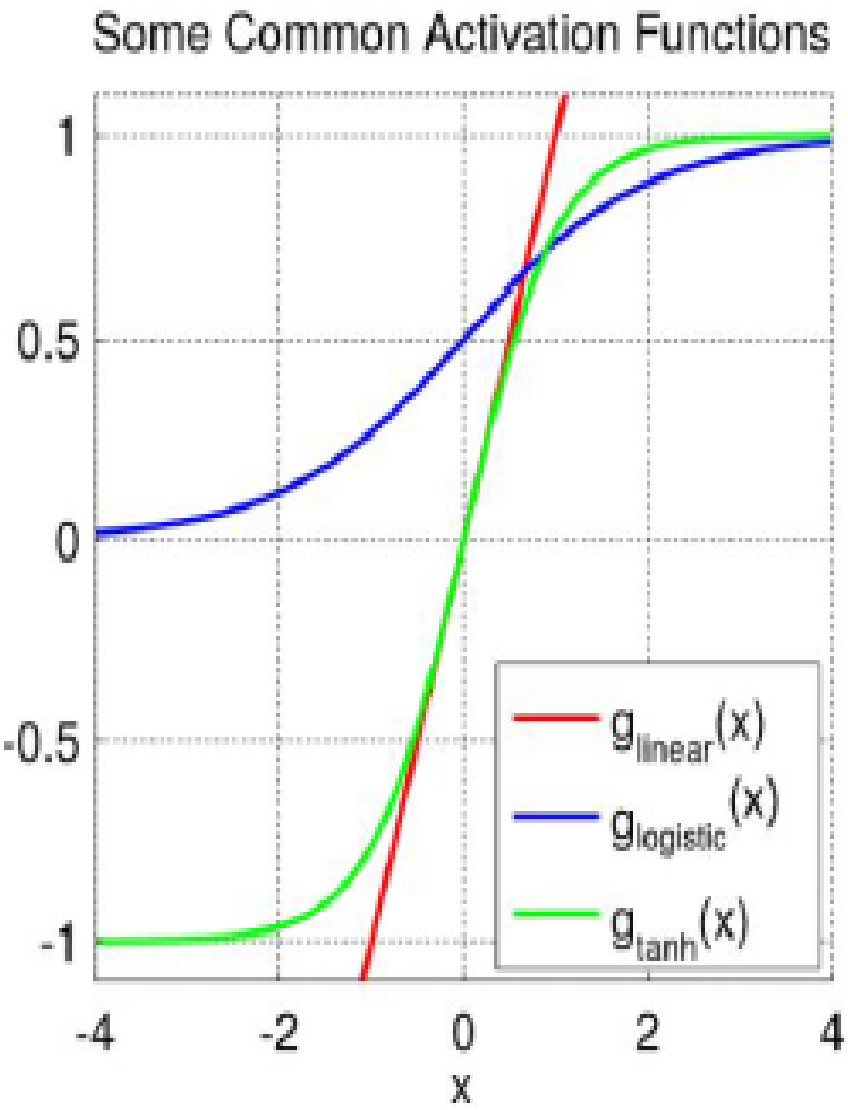


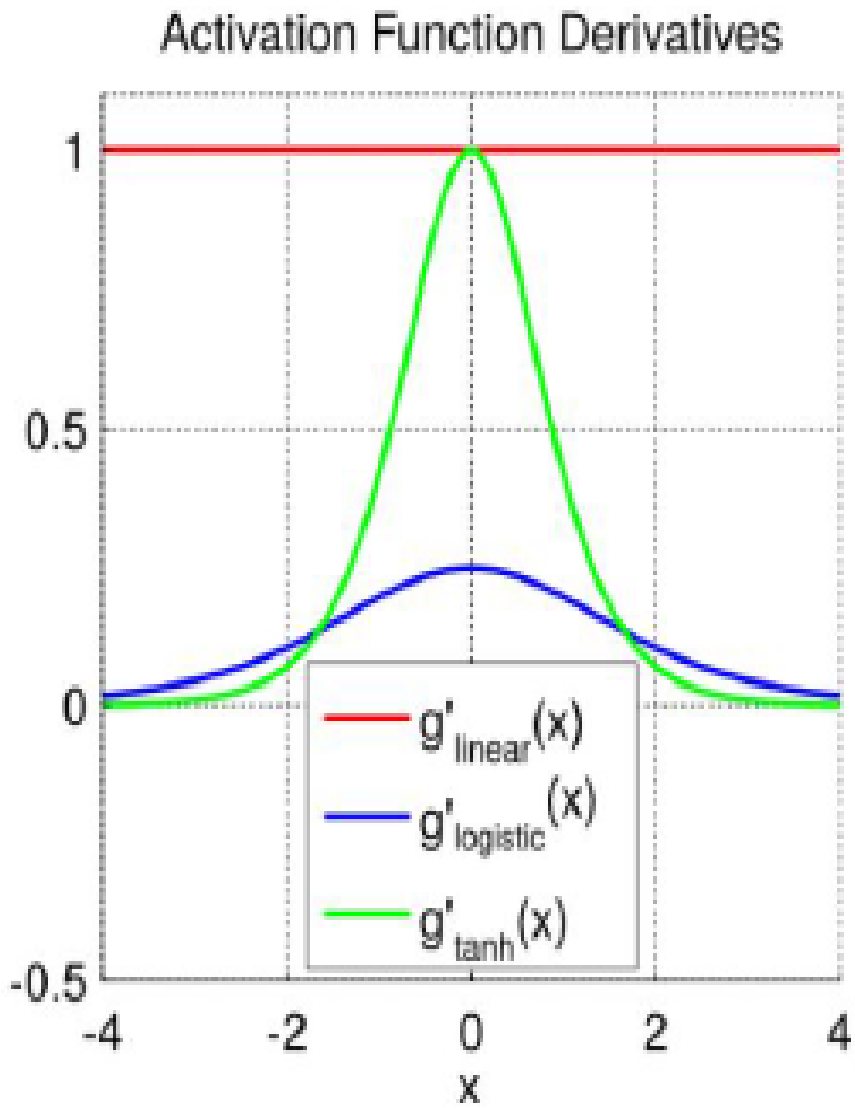
Deep Neural Network



Vanishing Gradient

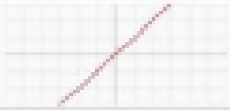
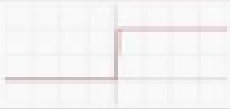
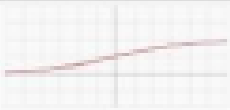
신경망에서 딥러닝





https://theclevermachine.files.wordpress.com/2014/09/mnet-error-functions2.png

신경망에서 딥러닝

Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1-f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2+1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU)[2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU)[3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1+e^x)$	$f'(x) = \frac{1}{1+e^{-x}}$

https://dwaithe.github.io/blog_20170508.html

고급 경사 하강법

- 속도와 정확도 문제를 해결하는 고급 경사하강법.

1. 확률적 경사 하강법(Stochastic Gradient Desecent)

· 전체 데이터를 사용하는 것이 아니라, 랜덤하게 추출한 일부 데이터를 사용

· 일부 데이터를 사용하므로 더 빨리 그리고 자주 업데이트 하는것이 가능해짐.

· 랜덤한 일부 데이터를 사용하는 만큼 확률적 경사 하강법은 중간 결과의 진폭이 크고 불안정해 보일수 있음.

$$W(t+1) = W(t) - \eta \frac{\partial error}{\partial W} \qquad but, x^{(i:i+n)}, \eta \text{는 학습률 ,}$$

x^i : training data

모멘텀(Momentum)

- 모멘텀이란 단어는 관성, 탄력, 가속도라는 뜻이다.
  모멘텀 SGD란 말 그대로 경사 하강법에 탄력을 더해 주는것.

- 경사 하강법과 마찬가지로 매번 기울기를 구하지만, 이를 통해 오차를 수정하기 전 바로 앞 수정 값과 방향$(+,-)$를 참고하여 같은 방향으로 일정한 비율만 수정되게 하는 방법.

$$V(t) = \gamma v(t-1) - \eta \frac{\partial error}{\partial W}$$
$$W(t+1) = W(t) + V(t)$$

네스테로프 모멘텀(Momentum)

- 모멘텀 SGD방식에 한 가지 옵션을 추가한 것.

즉, 그래디언트를 구할 때, 먼저 $\gamma v(t-1)$ 값을 더한 다음 계산하는 것
이단계를 추가함으로써 V(t)를 계산하기 전에 모멘텀 방법으로 인해 이동될 방향을 미리 예측하고 해당 바향으로 얼마간 미리 이동한 뒤 그래디언트를 계산하는 효과를 얻을 있음.
이 방법은 속도는 그대로 이면서, 이동을 실행하기 전 한단계를 미리 예측함으로써 불필요한 이동을 줄일수 있다.

$$V(t) = \gamma v(t-1) - \eta \frac{\partial error}{\partial (W + \gamma v(t-1))}$$

$$W(t+1) = W(t) + V(t)$$

아다그리드(Adagrad, Adaptive Gradient)

- 변수의 업데이트 횟수에 따라 학습률을 조절하는 옵션이 추가된 최적화 방법.

만일 학습 도중 어떤 변수가 자주 업데이트되었다면, 이 변수는 최적화 값에 더 가까워졌다고 가정할 수 있음. 이때 해당 변수에 대한 학습률을 줄여 좀 더 세밀한 업데이트를 해주면, 예측의 정확도가 높아짐.

$$G(t) = G(t-1) + \left[\frac{\partial error}{\partial W(t)}\right]^2$$

$$W(t+1) = W(t) + \eta \cdot \frac{1}{\sqrt{G(t)+\epsilon}} \cdot \frac{\partial error}{\partial W(t)}$$

아엠에스프롭(RMSprop)

- RMSprop은 아다그라드의 G(t)값이 무한히 커지는 것을 방지하고자 제안된 방법, 논문과 같은 형태로 발표된 다른 방법들과 달리 딜러닝의 대가인 제프리 힌튼 교수와 제자들이 코세라 수업에서 소개하였음.
이 방법은 한마디로 이동 지수의 평균을 이용한 방법입니다.
여기서 gamma 계수가 이전값과 수정 값 사이의 적용 비율을 조절해 줌.
따라서 이전 기울기의 영향을 억제하는 효과를 보이며, 아다그리드의 G(t) 값에 해당하는 부분이 급격히 변하는 것을 방지함

$$G(t) = \gamma \; G(t-1) + (1-\gamma)\left[\frac{\partial(error)}{\partial W(t)}\right]^2$$

$$W(t+1) = W(t) + \eta \cdot \frac{1}{\sqrt{G(t)+\epsilon}} \cdot \frac{\partial(error)}{\partial W(t)}$$

아담(Adam)

Adam은 알엠에스프롭에서 한 단계 더 업데이트 된 방법., 알엠에스프롭과 유사하게 G(t)의 값을 구하지만, 이보다 앞서 살펴본 모멘텀 SGD와도 유사하게 그래디언트를 제곱하지 않는 V(t)의 값 또한 사용함. 즉 두 방법의 장점을 취하는 방식

$$V(t) = \gamma_1 \ G(t-1) + (1-\gamma_1)\frac{\partial(error)}{\partial W(t)}$$

$$G(t) = \gamma_2 \ G(t-1) + (1-\gamma_2)\left[\frac{\partial(error)}{\partial W(t)}\right]^2$$

두 값을 다음과 같이 대입하면,

$$\hat{V}(t) = \frac{V(t)}{1-V_1^t}, \hat{G}(t) = \frac{G(t)}{1-V_2^t}$$

$$W(t+1) = W(t) - \eta \ \frac{\hat{G}(t)}{\sqrt{\hat{V}(t)+\epsilon}}$$

Reference

- 계산그래프로 역전파 이해하기 : https://brunch.co.kr/@chris-song/22
- Gradient Descent : https://brunch.co.kr/@chris-song/50
- 모두의 딥러닝, 한빛미디어
- 기계학습,한빛미디어(오일석)