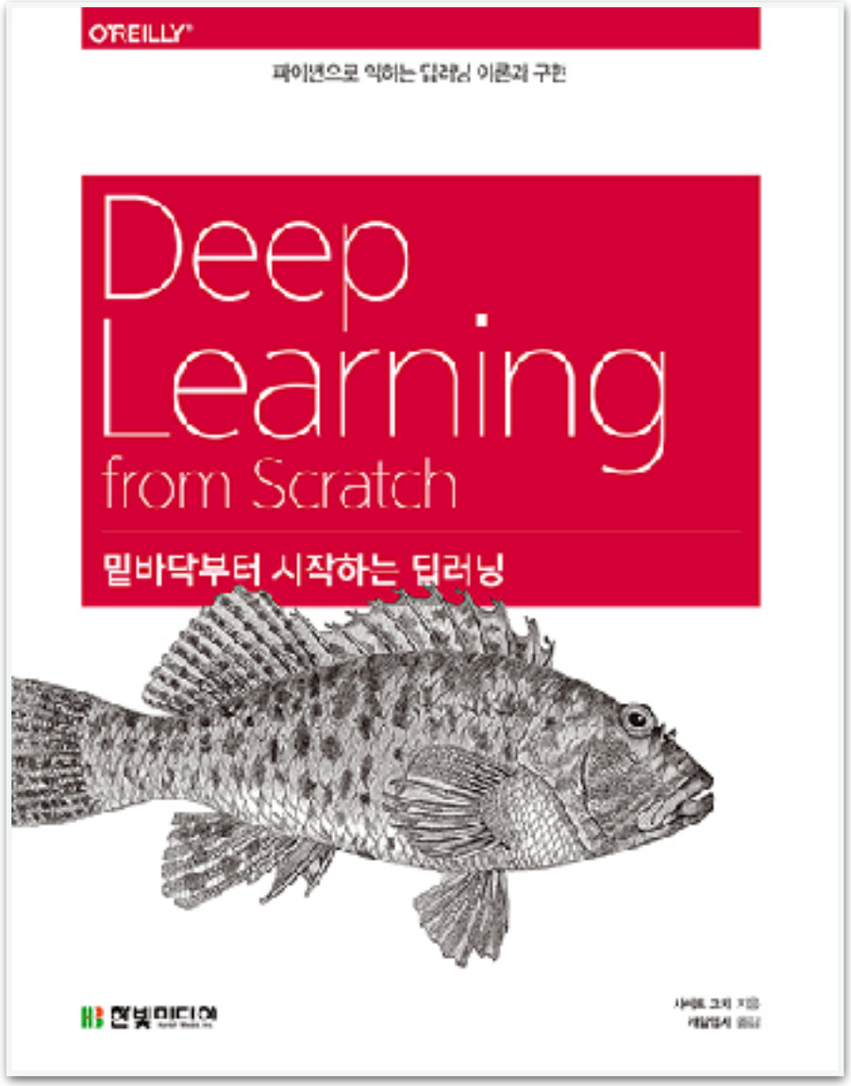
DL의 이론적 배경

밑바닥.. (물고기) Ch2-5

Namun Cho/ mailto:namun@snu.ac.kr

주제

- 퍼셉트론 Ch2
- 신경망 Ch3
- 신경망학습 Ch4
- 오차역전파법 Ch5



Namun Cho/ mailto:namun@snu.ac.kr

Perceptron

물고기 Ch2

Namun Cho/ mailto:namun@snu.ac.kr

Model of Neuron

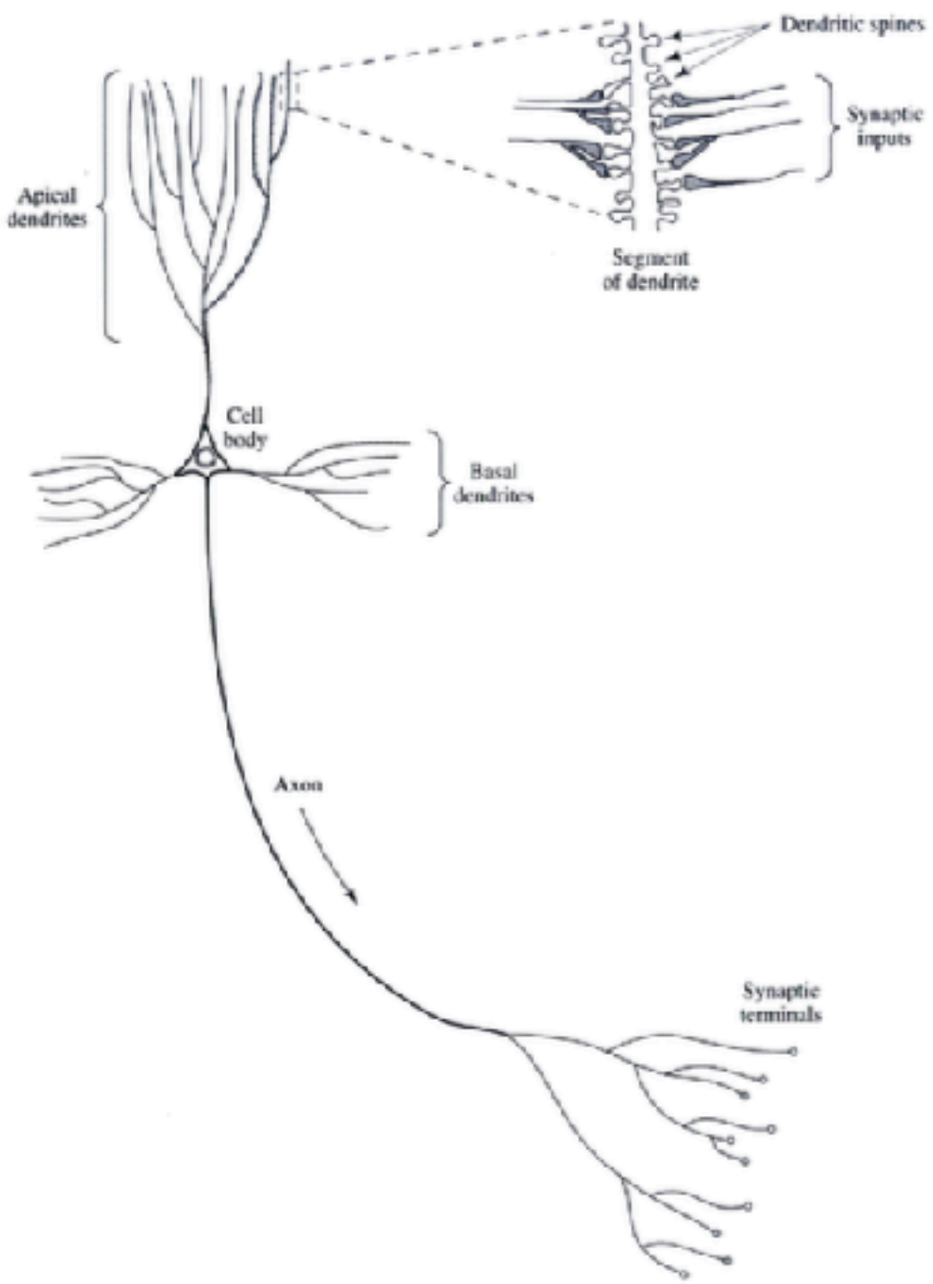
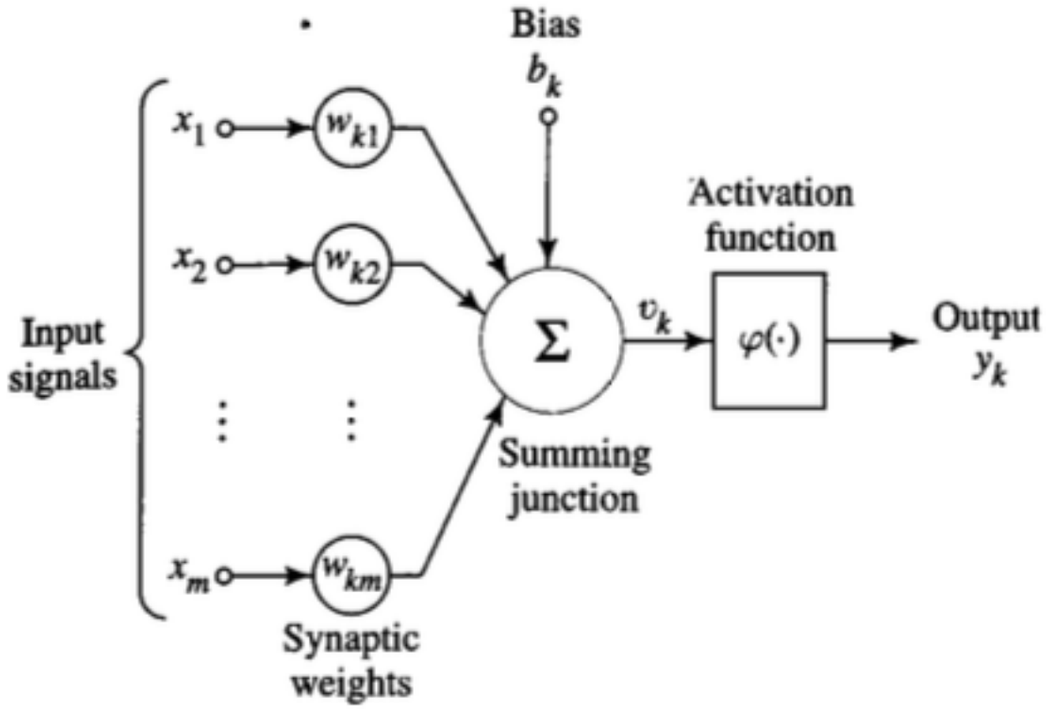
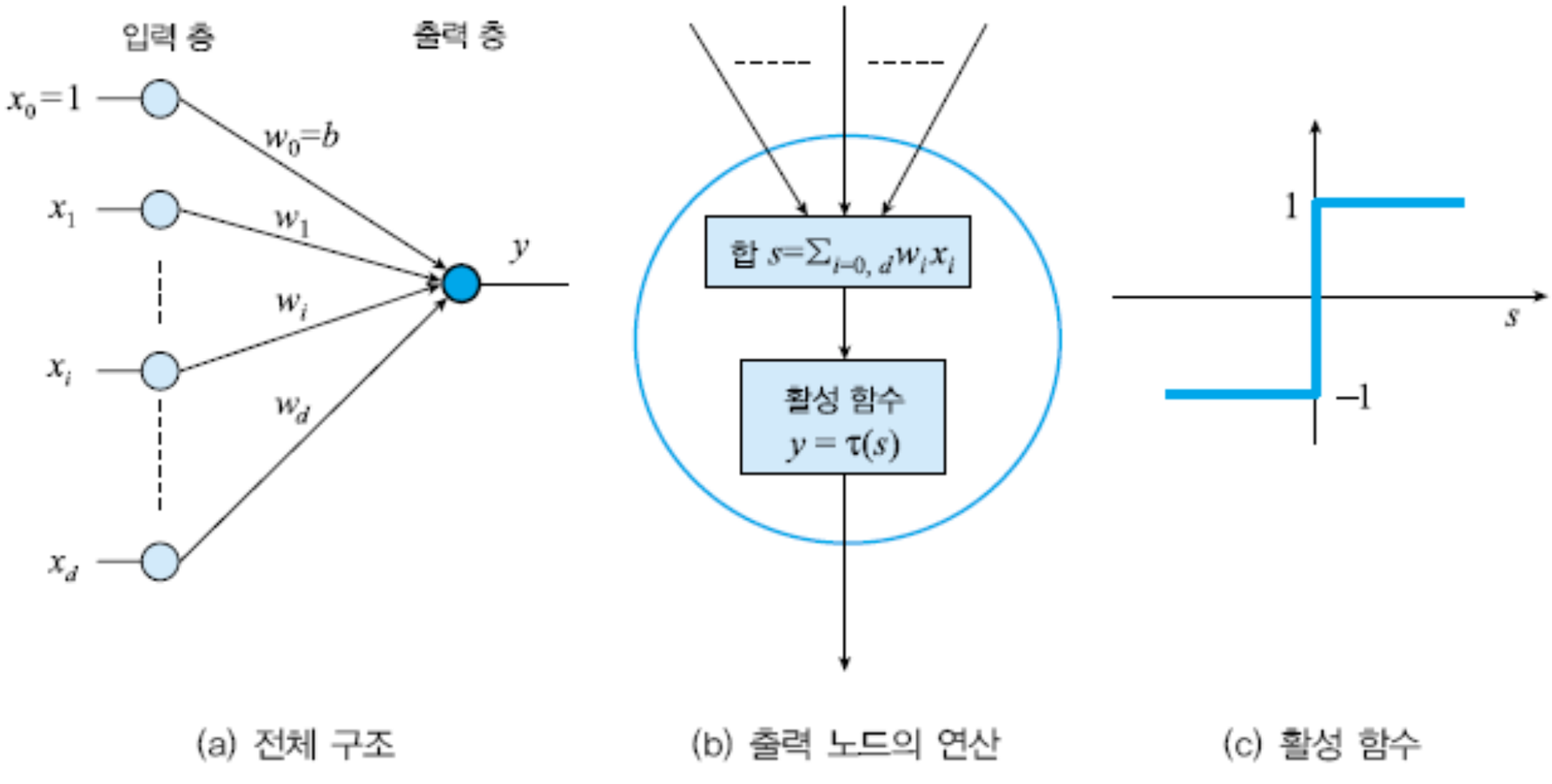


FIGURE 1.2 The pyramidal cell.



퍼셉트론 Perceptron

- 신경망의 수학적 모형



(a) 전체 구조 (b) 출력 노드의 연산 (c) 활성 함수

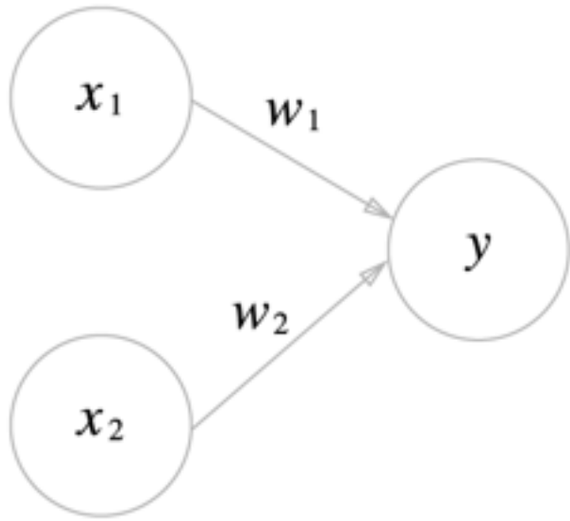
그림 4.2 퍼셉트론의 구조

출처: 패턴인식

Namun Cho/ mailto:namun@snu.ac.kr

퍼셉트론

- 원: 뉴런 (node, or vertex)
- 선: 시냅스 (link, or edge)
- x1, x2: 입력
- w1, w2: 가중치
- θ: 임계값
- y: 출력



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

입력2 → 출력1 퍼셉트론의 예

- 퍼셉트론으로 논리연산을 할 수 있음
  - AND
  - NAND
  - OR
- 위 역할을 수행할 수 있는 (w1,w2,θ) 벡터를 찾는 문제
  - AND: (0.5,0.5,0.7)
  - NAND: (-1,-1,-1)
  - OR: (0.5,0.5,0.3)

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

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

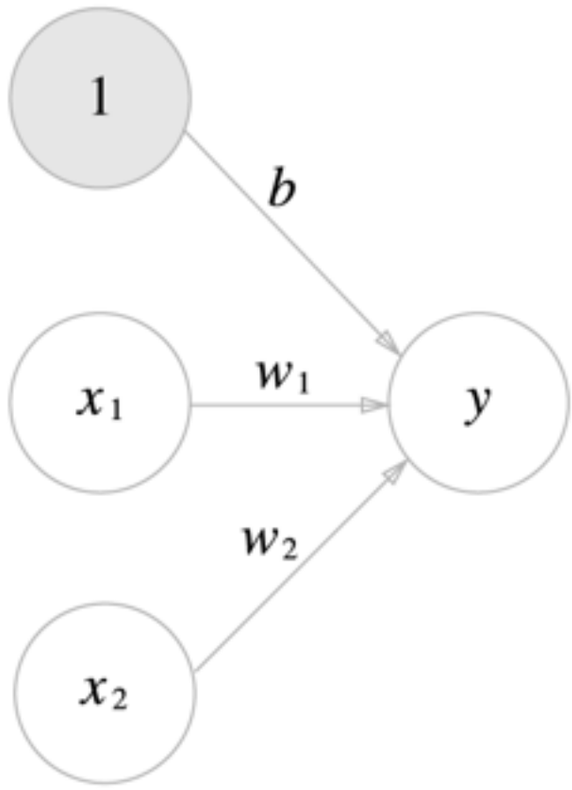
x_1	x_2	y
0	0	0
1	0	1
0	1	1
1	1	1

임계값 → bias (편향)

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

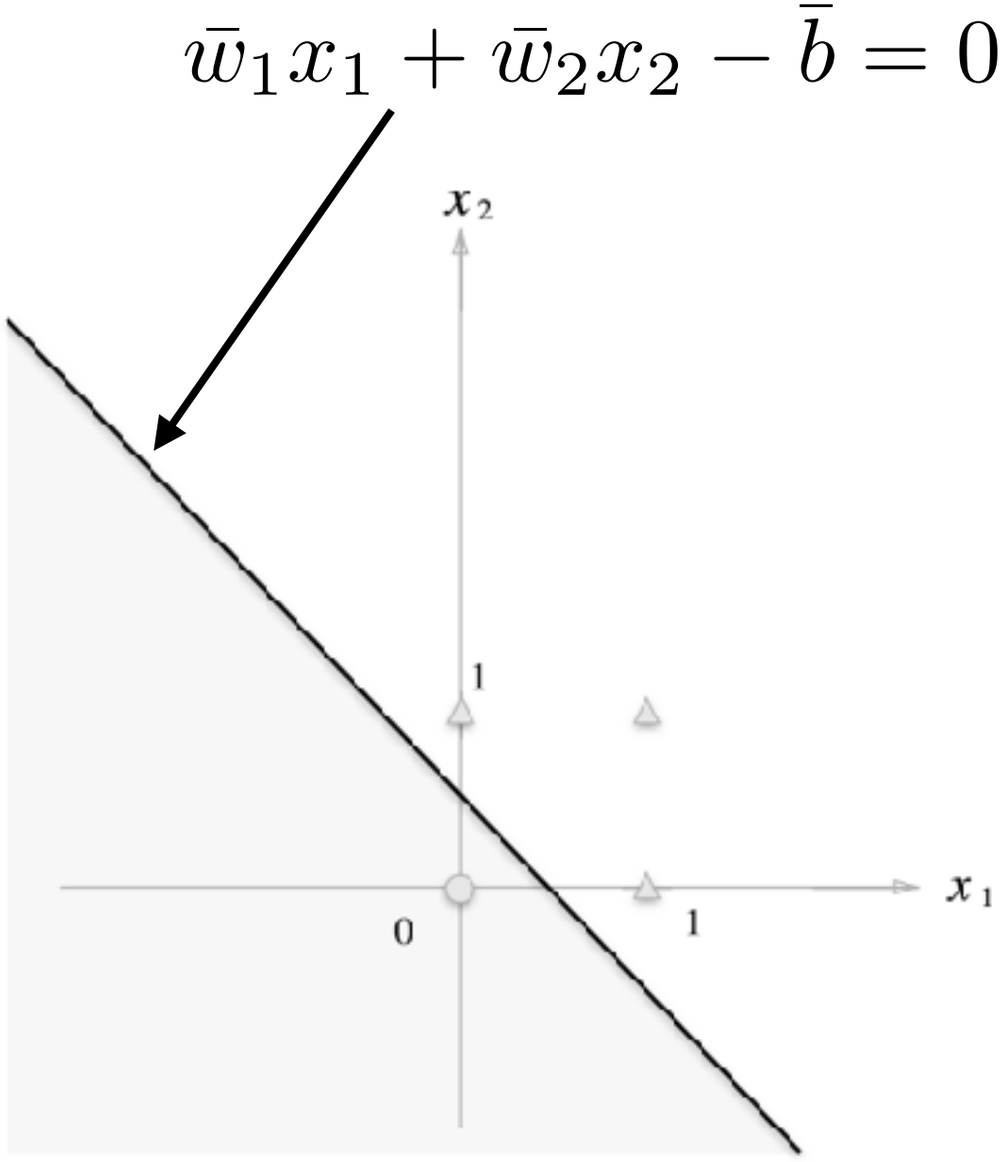
$$y = \begin{cases} 0 & (b + w_1 x_1 + w_2 x_2 \le 0) \\ 1 & (b + w_1 x_1 + w_2 x_2 > 0) \end{cases}$$

- b := -θ



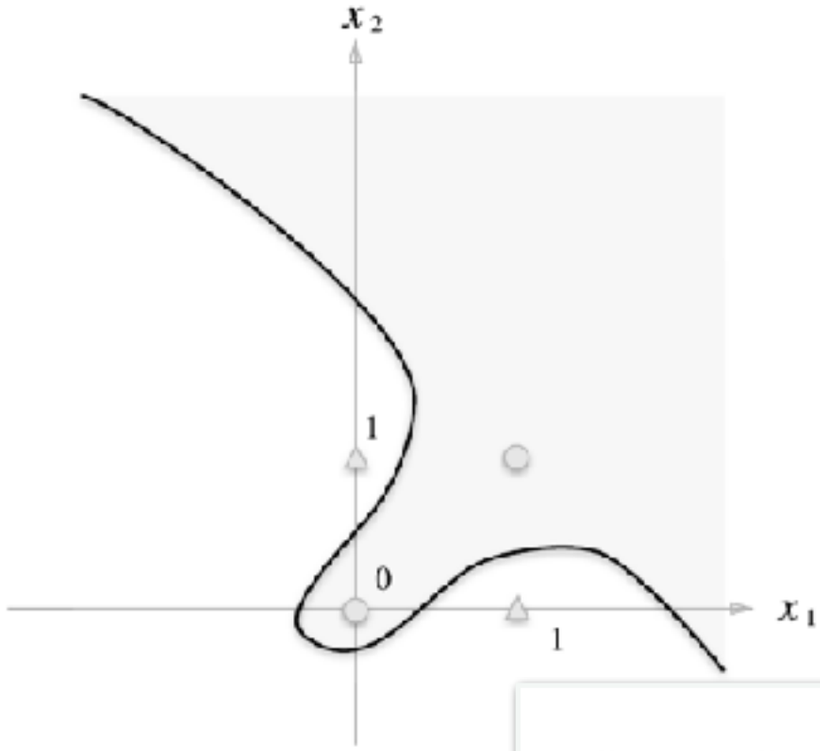
기하학적 의미

- 단일 퍼셉트론은 선형 구분 선을 의미함
  - 입력수 = 차원수
  - 즉, 입력 퍼셉트론이 2개인 경우 2차원 선형 구분선 (즉, 직선)을 의미
  - n개인 경우 n차원 버젼의 평면으로된 구분면 (hyperplane)



다층 퍼셉트론 (MLP)
Multi-Layer Perceptron

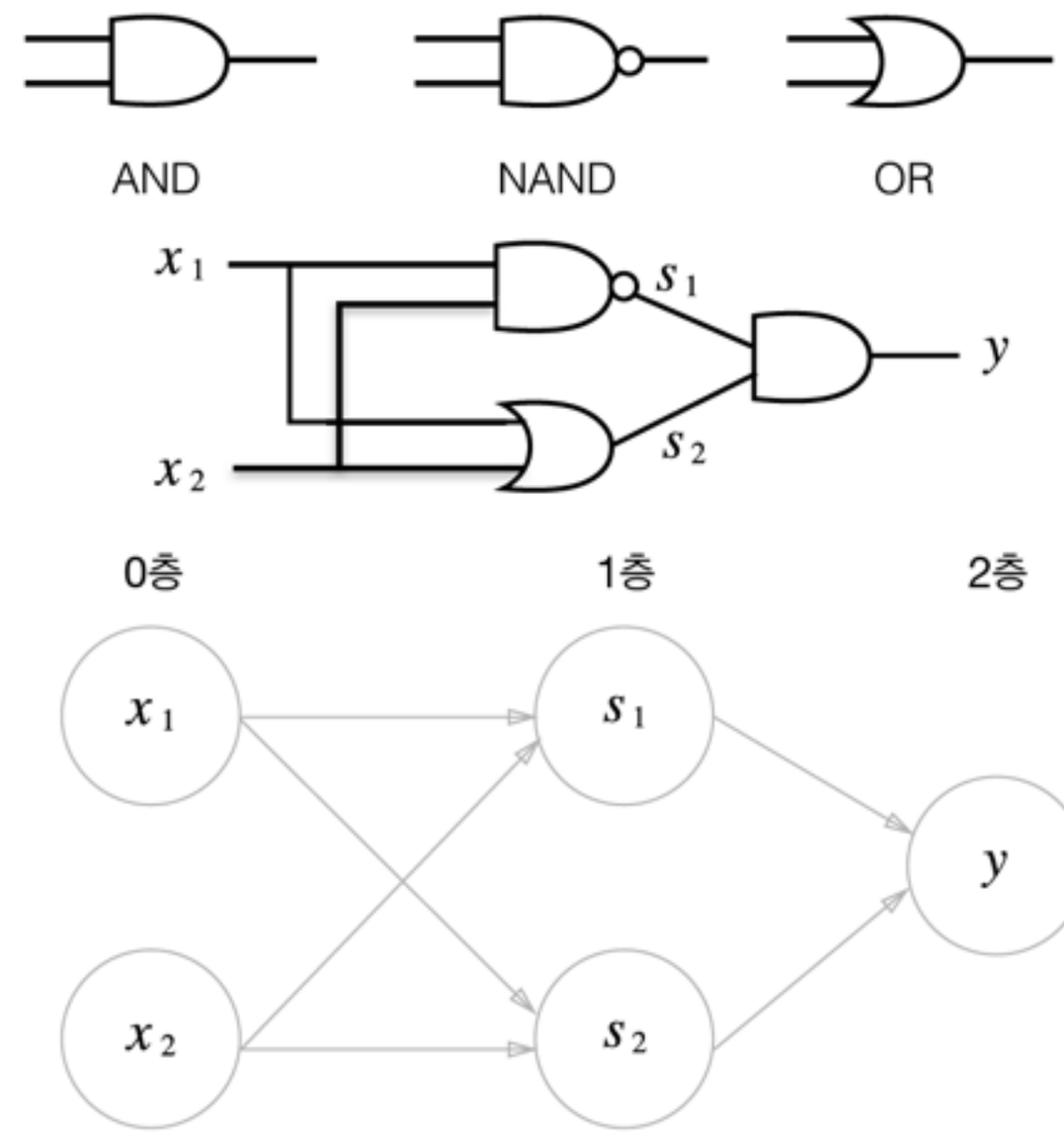
- XOR의 경우 output을 가르는 선형 구분선을 만들 수 없음
- 하지만 AND, NAND, OR 세 퍼셉트론을 조합하여 XOR을 구현할 수 있음!



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

XOR by MLP

- XOR (x1, x2):
  - s1 = NAND(x1,x2)
  - s2 = OR(x1,x2)
  - return AND(s1,s2)
- 2 Layer Perceptron
- 활성함수들의 합성함수의 개념과 동등함



NANDs can do everything

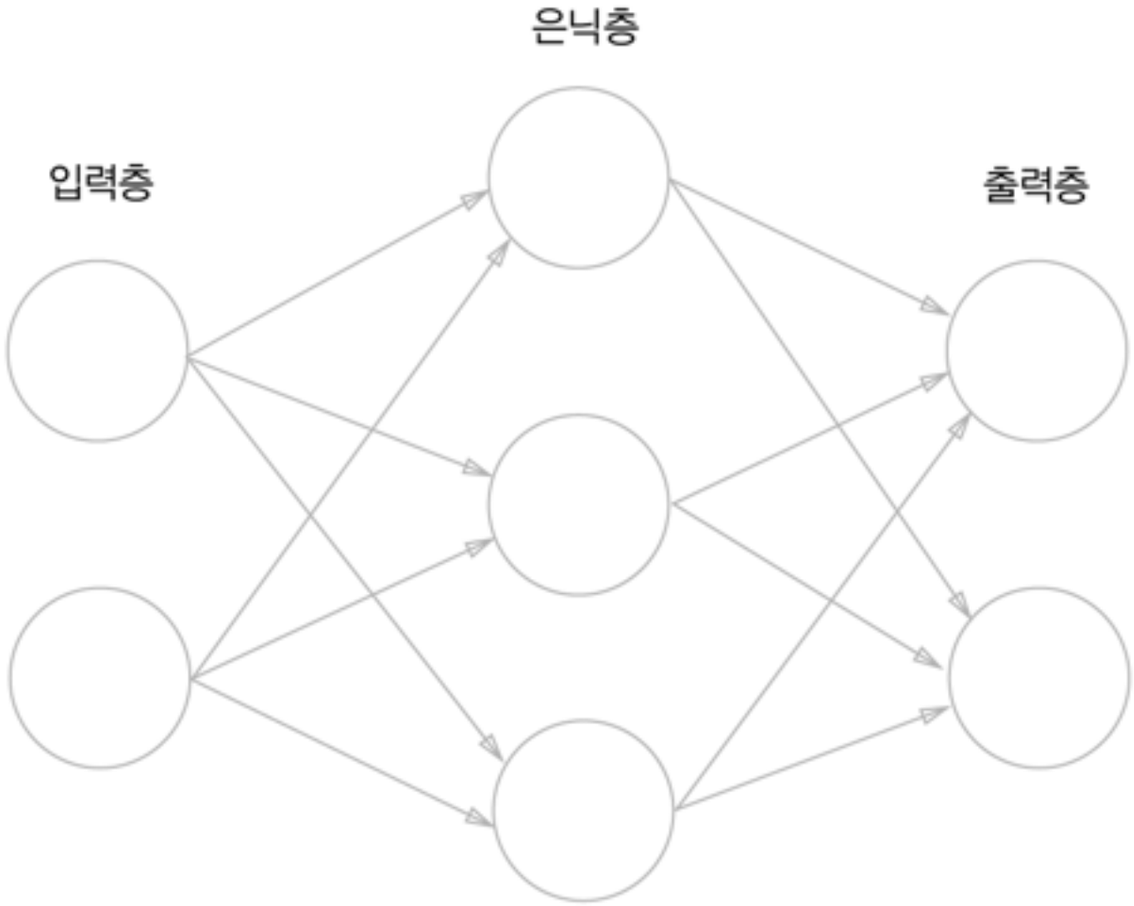
- NAND의 유한한 합성으로 컴퓨터의 모든 연산을 재현할 수 있음
  - The Elements of Computing Systems, MIT Press 2005
- Sigmoid 함수를 활성함수로 사용할 경우 임의의 함수를 2층 퍼셉트론으로 표현할 수 있음이 수리적으로 증명되어 있음 (이론적으로는!)
  - 현실적으로는 weight 를 찾는 것이 어려운 문제

NN: Neural Networks
신경망

물고기 Ch3

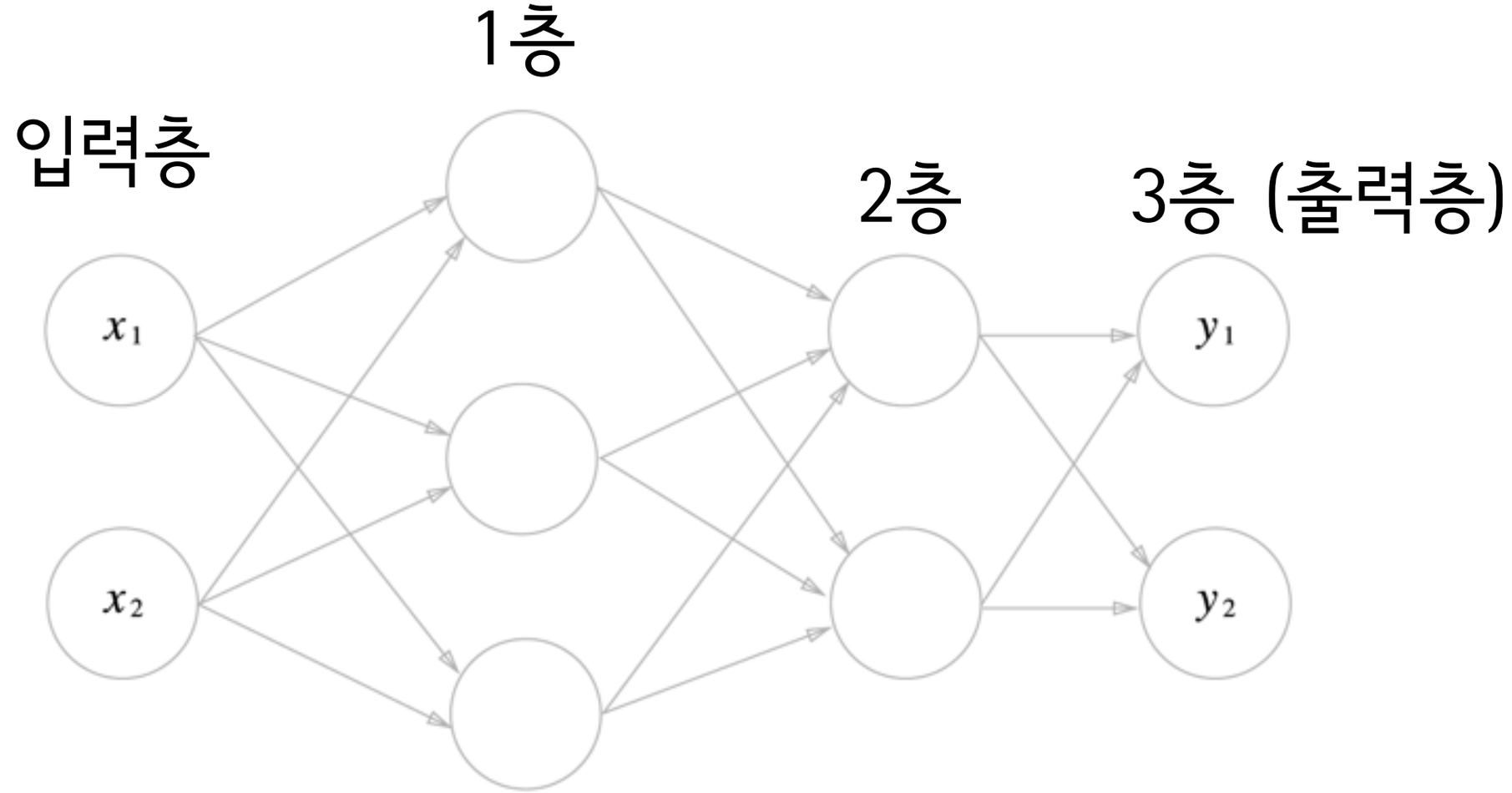
Namun Cho/ mailto:namun@snu.ac.kr

신경망 구조



- input layer: 입력 (data)
- hidden layer: 연산
- output layer: 결과

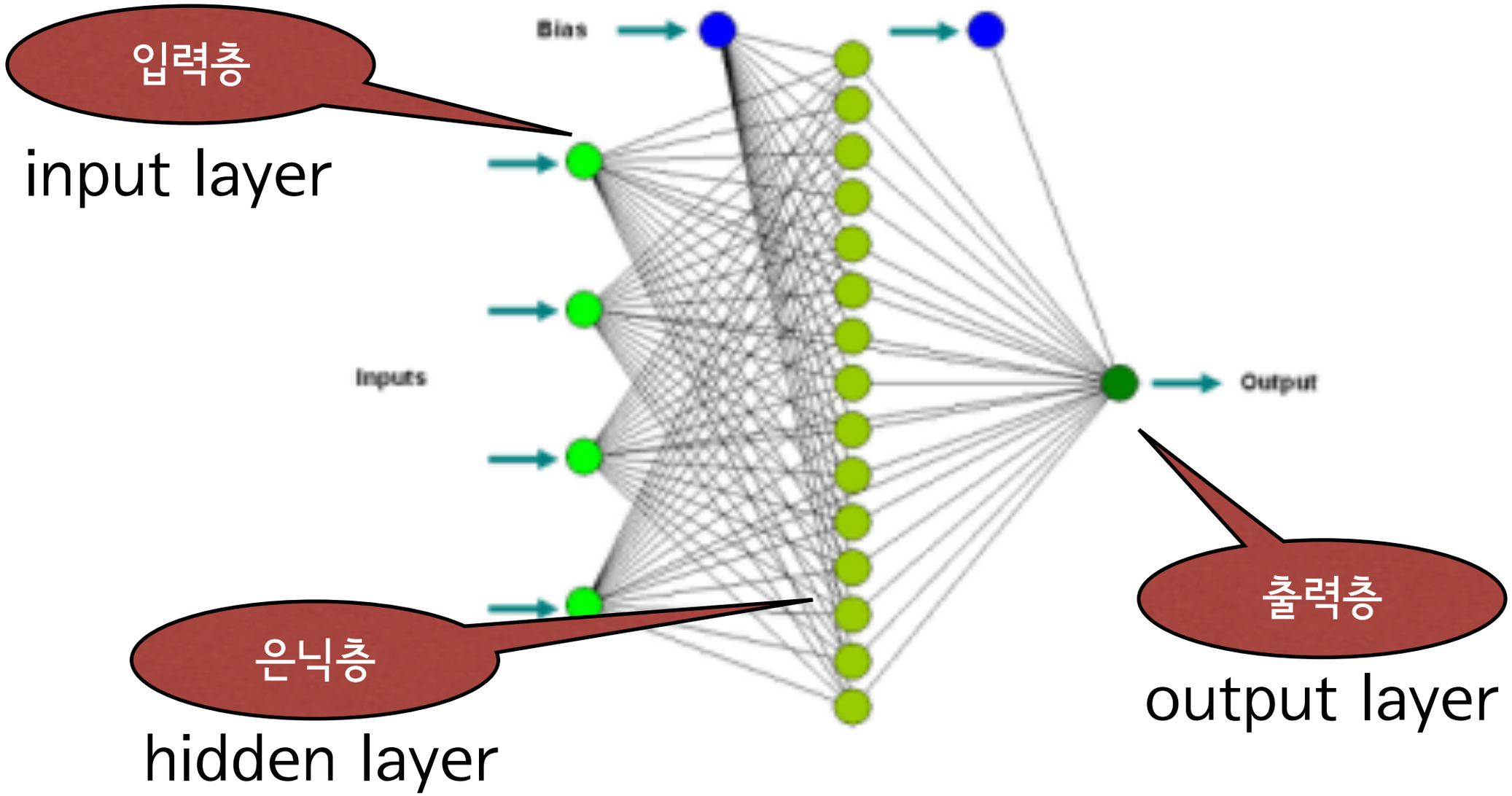
3층 신경망



사이토 고키 (2017)

Namun Cho/ mailto:namun@snu.ac.kr

신경망
Neural Networks



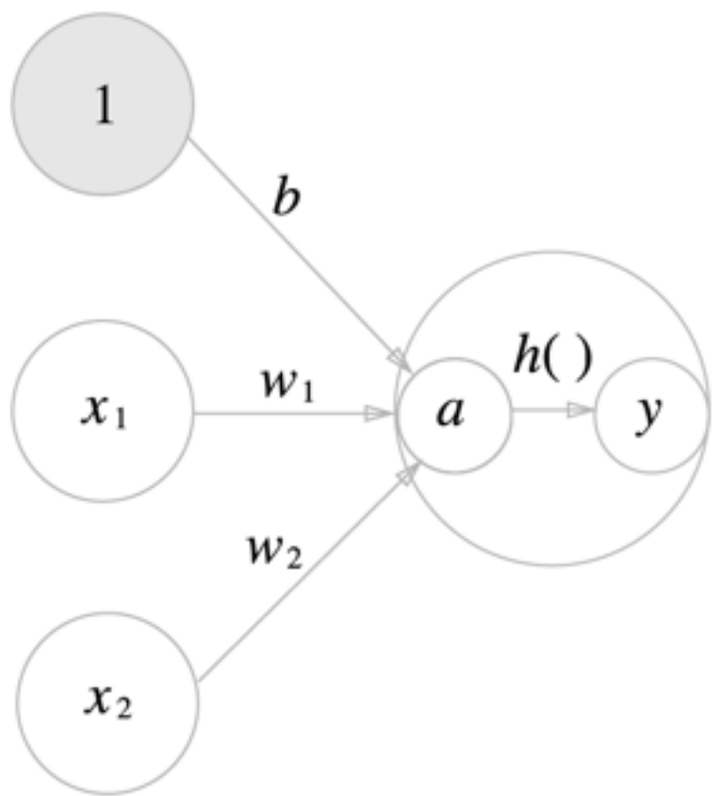
http://neuromastersoftware.com/neural-network-theory-introduction/

Activation Function

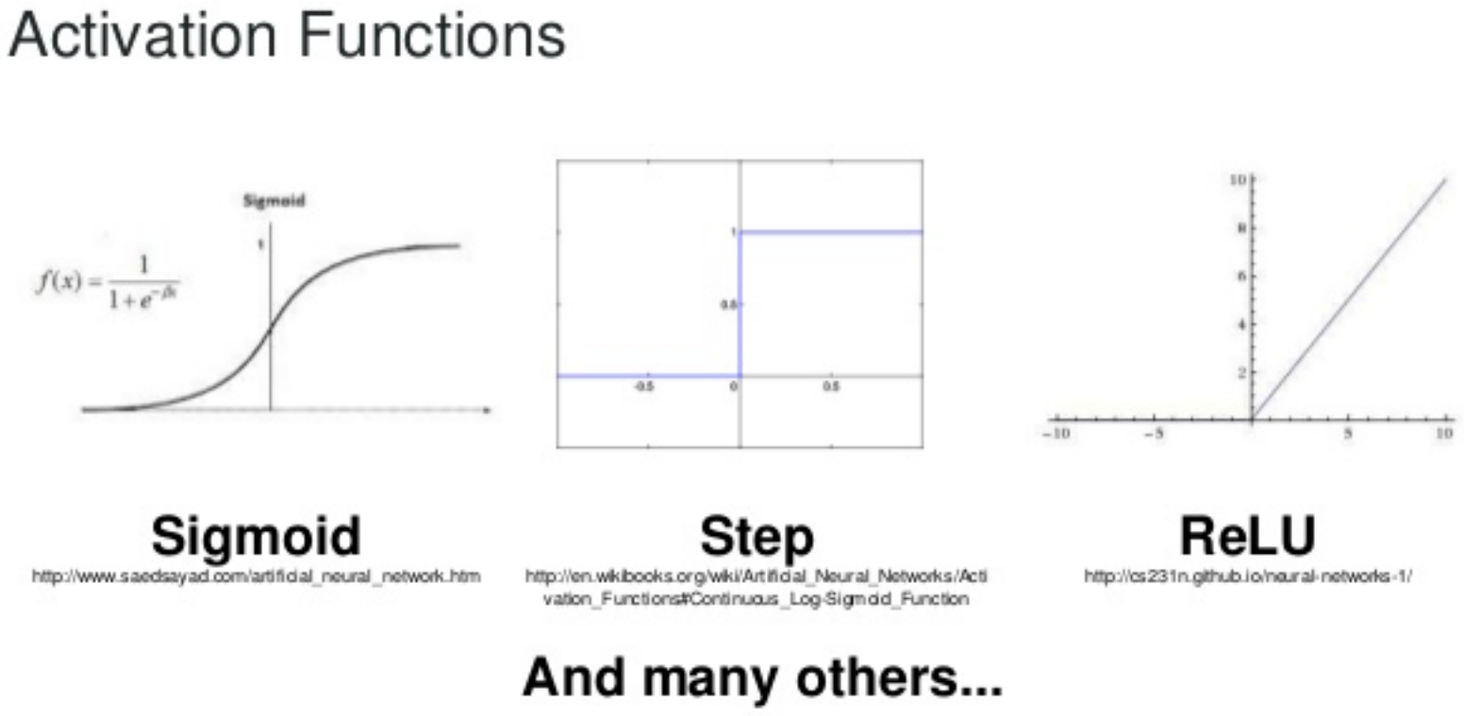
- h(x)
  - x: input으로 들어온 신호의 가중합
  - h: 가중합을 통해 activation 여부를 결정할 함수
- 다양한 activation 함수들 존재
  - step function
  - sigmoid function
  - relu function 등

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

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



활성함수

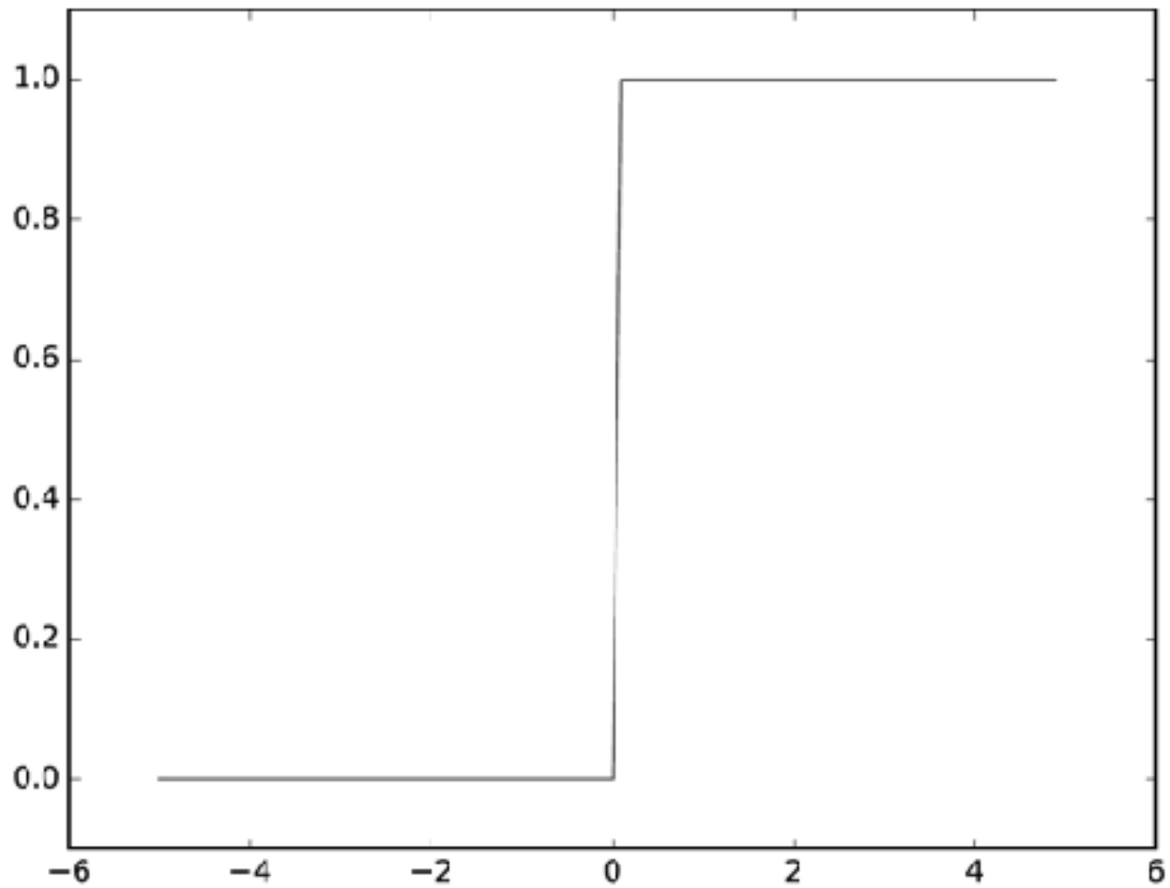


https://image.slidesharecdn.com/usuconference-deeplearning-160418191119/95/introduction-to-deep-learning-7-638.jpg?cb=1461006739

Namun Cho/ mailto:namun@snu.ac.kr

Step Function

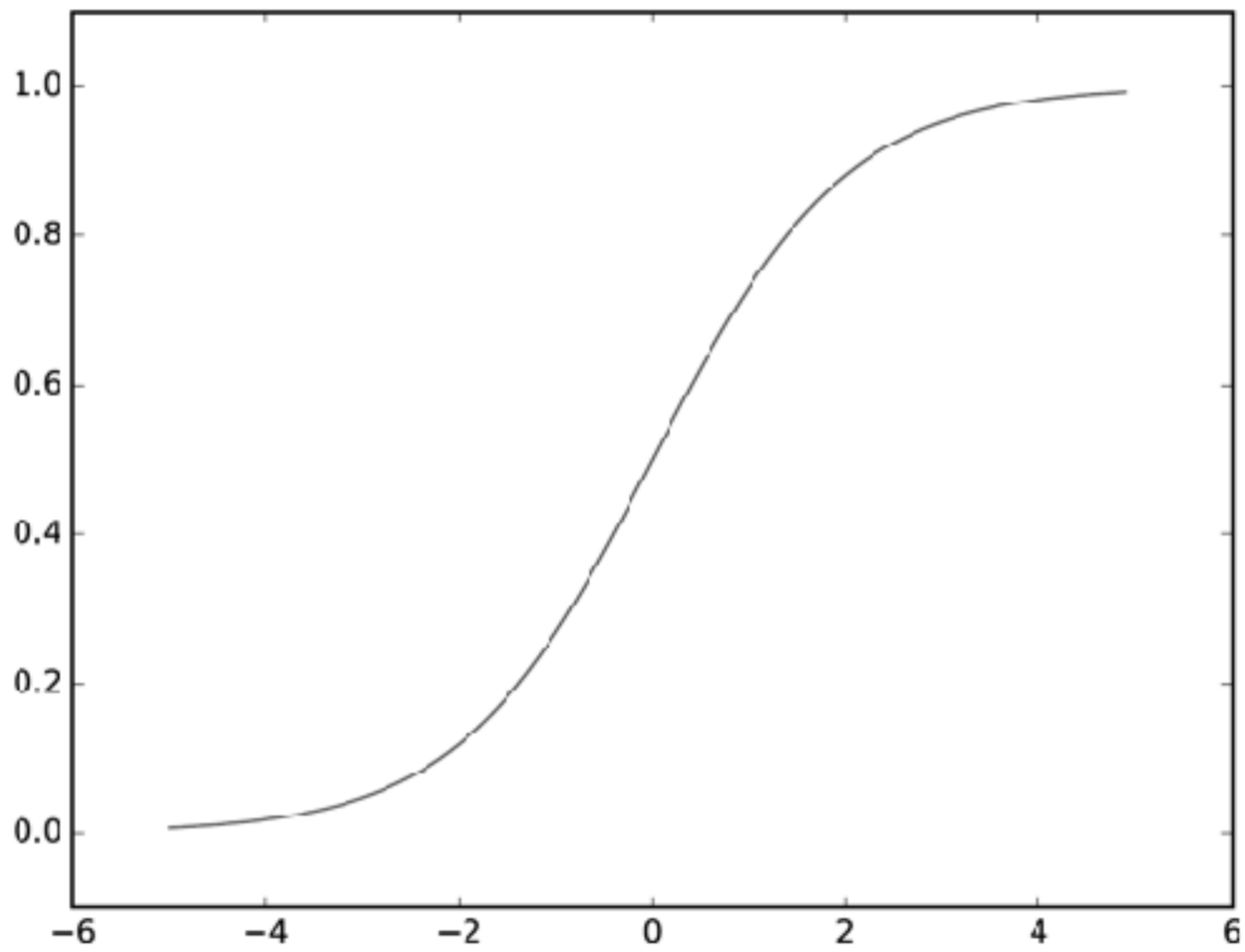
- 가로축 변수: 입력가중합
- 세로축 변수: 출력값
- 각 노드는 0 혹은 1의 값만을 발생시킴
- 미분 불가능
  - 계산적으로 좋지 않은 속성
  - 역함수 없음



Sigmoid Function

- S - shaped graph
- 각 노드는 0~1 사이의 실수를 발생시킴
- 미분 가능함!
  - 계산적으로 매우 좋은 특성임
- 특정 상황에서 step function보다 나을 수 있음
  - 미분 불가능한 함수는 다루기가 어려움

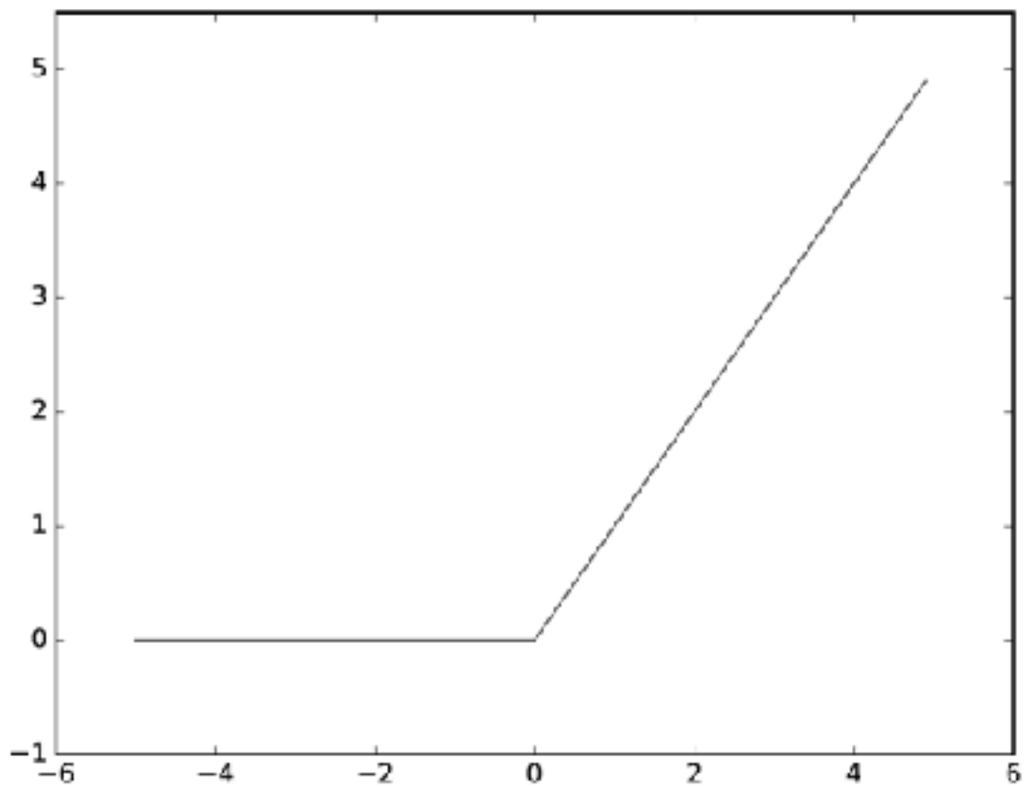
$$h(x) = \frac{1}{1 + \exp(-x)}$$



Rectified Linear Unit :ReLU Function

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

- 0 이하일 경우 무시
- 0 이상일 경우 입력값을 그대로 전달
- 0에서 미분 불가능
- 제한적인 미분 가능성 (0 이상) 존재



N차원 배열 (tensor?)

- 벡터: 숫자를 1차원 직선으로 배열한 것
- 행렬: 숫자를 2차원 직사각형으로 배열한 것
- 3차원 배열: 숫자를 3차원 직육면체로 배열한 것
- N차원 배열: 숫자를 N차원 직육면체(hyper cube)로 배열한 것
- Numpy 구현:
  - np.array(…)

Inner Product

$$\mathbf{u} \bullet \mathbf{v} := \sum_{i}^{n} u_i v_i \in \mathbb{R}^1$$

- Vector Inner Product
  - 잘 알고 있는 그것.
- Matrix Inner Product
  - 2D: 잘 알고 있는 그것
  - nD: 잘 모를 그것..

$$\mathbf{A} = \begin{pmatrix} A_{11} & A_{12} & \cdots & A_{1m} \\ A_{21} & A_{22} & \cdots & A_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ A_{n1} & A_{n2} & \cdots & A_{nm} \end{pmatrix} = \begin{pmatrix} \mathbf{a}_1 \\ \mathbf{a}_2 \\ \vdots \\ \mathbf{a}_n \end{pmatrix},$$

$$\mathbf{B} = \begin{pmatrix} B_{11} & B_{12} & \cdots & B_{1p} \\ B_{21} & B_{22} & \cdots & B_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ B_{m1} & B_{m2} & \cdots & B_{mp} \end{pmatrix} = \begin{pmatrix} \mathbf{b}_1 & \mathbf{b}_2 & \cdots & \mathbf{b}_p \end{pmatrix}$$

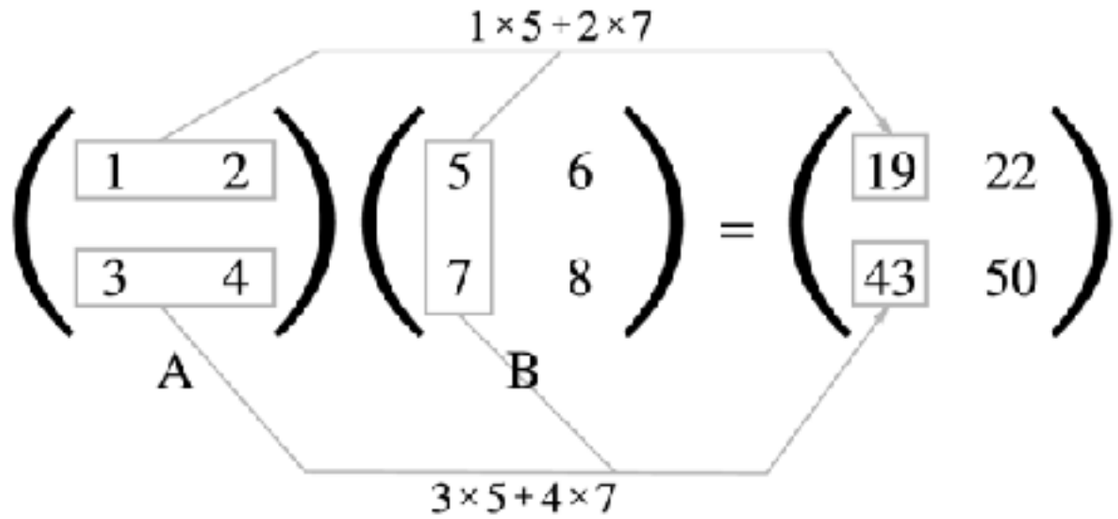
$$\mathbf{AB} = \begin{pmatrix} \mathbf{a}_1 \\ \mathbf{a}_2 \\ \vdots \\ \mathbf{a}_n \end{pmatrix} \begin{pmatrix} \mathbf{b}_1 & \mathbf{b}_2 & \cdots & \mathbf{b}_p \end{pmatrix} = \begin{pmatrix} (\mathbf{a}_1 \cdot \mathbf{b}_1) & (\mathbf{a}_1 \cdot \mathbf{b}_2) & \cdots & (\mathbf{a}_1 \cdot \mathbf{b}_p) \\ (\mathbf{a}_2 \cdot \mathbf{b}_1) & (\mathbf{a}_2 \cdot \mathbf{b}_2) & \cdots & (\mathbf{a}_2 \cdot \mathbf{b}_p) \\ \vdots & \vdots & \ddots & \vdots \\ (\mathbf{a}_n \cdot \mathbf{b}_1) & (\mathbf{a}_n \cdot \mathbf{b}_2) & \cdots & (\mathbf{a}_n \cdot \mathbf{b}_p) \end{pmatrix}$$

numpy.**dot**(*a*, *b*, *out=None*)¶

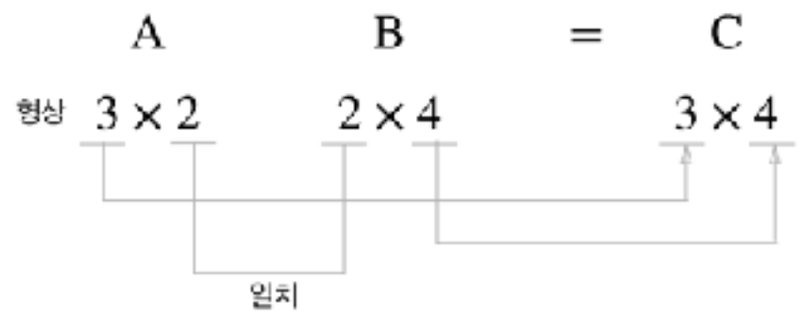
Dot product of two arrays.
For 2-D arrays it is equivalent to matrix multiplication, and for 1-D arrays to inner product of vectors (without complex conjugation). For N dimensions it is a sum product over the last axis of *a* and the second-to-last of *b*:

```
dot(a, b)[i,j,k,m] =
sum(a[i,j,:] * b[k,:,m])
```

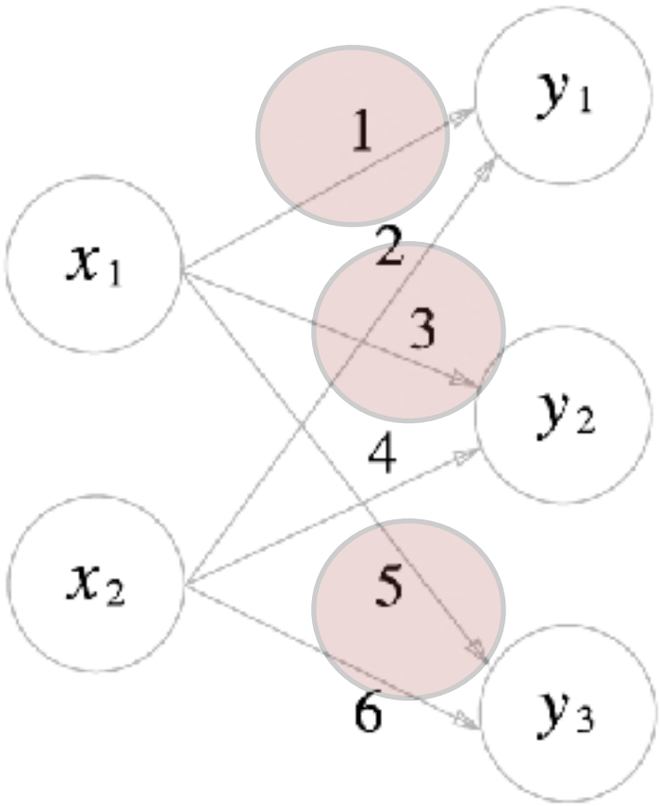
Matrix Inner Product: Example



- Numpy: np.dot(A,B)
- (m × n) · (n × p) = (m × p)



신경망 ⇒ 선형대수



$$\begin{pmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{pmatrix}$$

$$\mathrm{X} \qquad \mathrm{W} \qquad = \qquad \mathrm{Y}$$

$$2 \qquad 2 \times 3 \qquad\qquad 3$$

$$(1 \times 2) \bullet (2 \times 3) = (1 \times 3)$$

일치

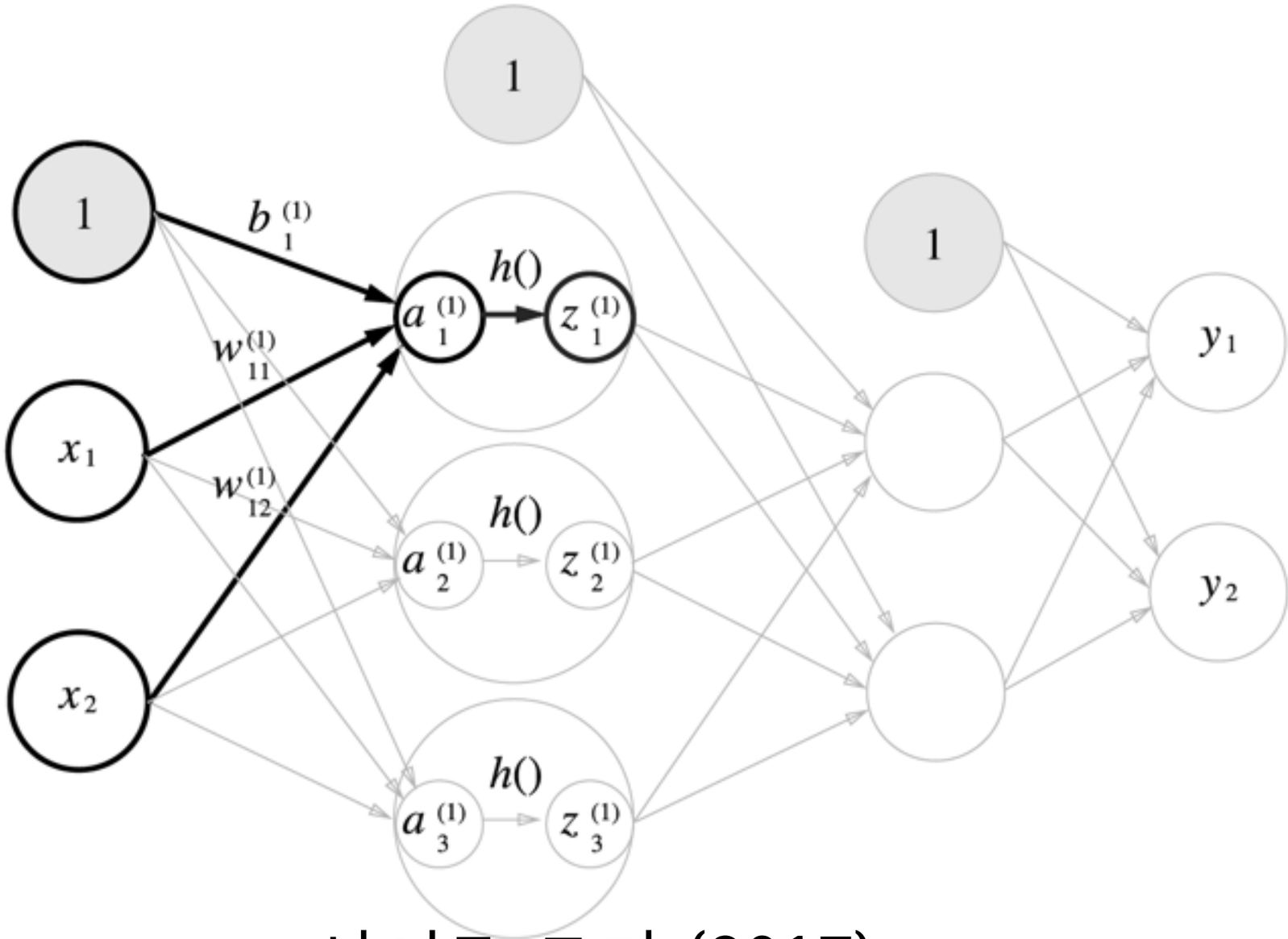
$$\begin{pmatrix} x_1 & x_2 \end{pmatrix} \bullet \begin{pmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{pmatrix} = \begin{pmatrix} y_1 & y_2 & y_3 \end{pmatrix}$$

사이토 고키 (2017)

Namun Cho/ mailto:namun@snu.ac.kr

$$A^{(1)} = X \bullet W^{(1)} + B^{(1)}$$

$$\begin{pmatrix} a_1^{(1)} & a_2^{(1)} & a_3^{(1)} \end{pmatrix} = \begin{pmatrix} x_1 & x_2 \end{pmatrix} \bullet \begin{pmatrix} w_{11}^{(1)} & w_{12}^{(1)} & w_{13}^{(1)} \\ w_{21}^{(1)} & w_{22}^{(1)} & w_{23}^{(1)} \end{pmatrix} + \begin{pmatrix} b_1^{(1)} & b_2^{(1)} & b_3^{(1)} \end{pmatrix}$$

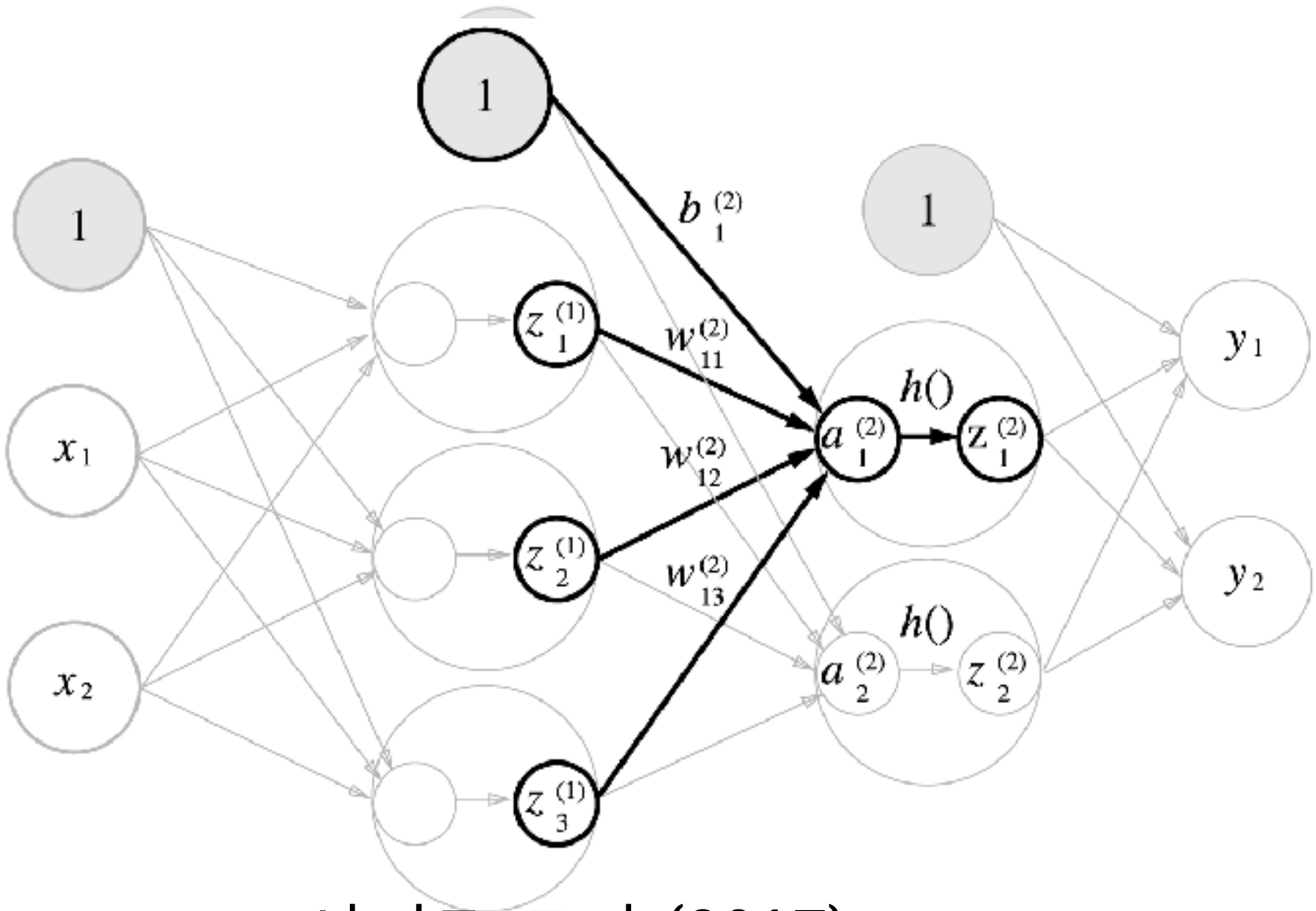


사이토 고키 (2017)

Namun Cho/ mailto:namun@snu.ac.kr

$$A^{(1)} = X \bullet W^{(1)} + B^{(1)}$$

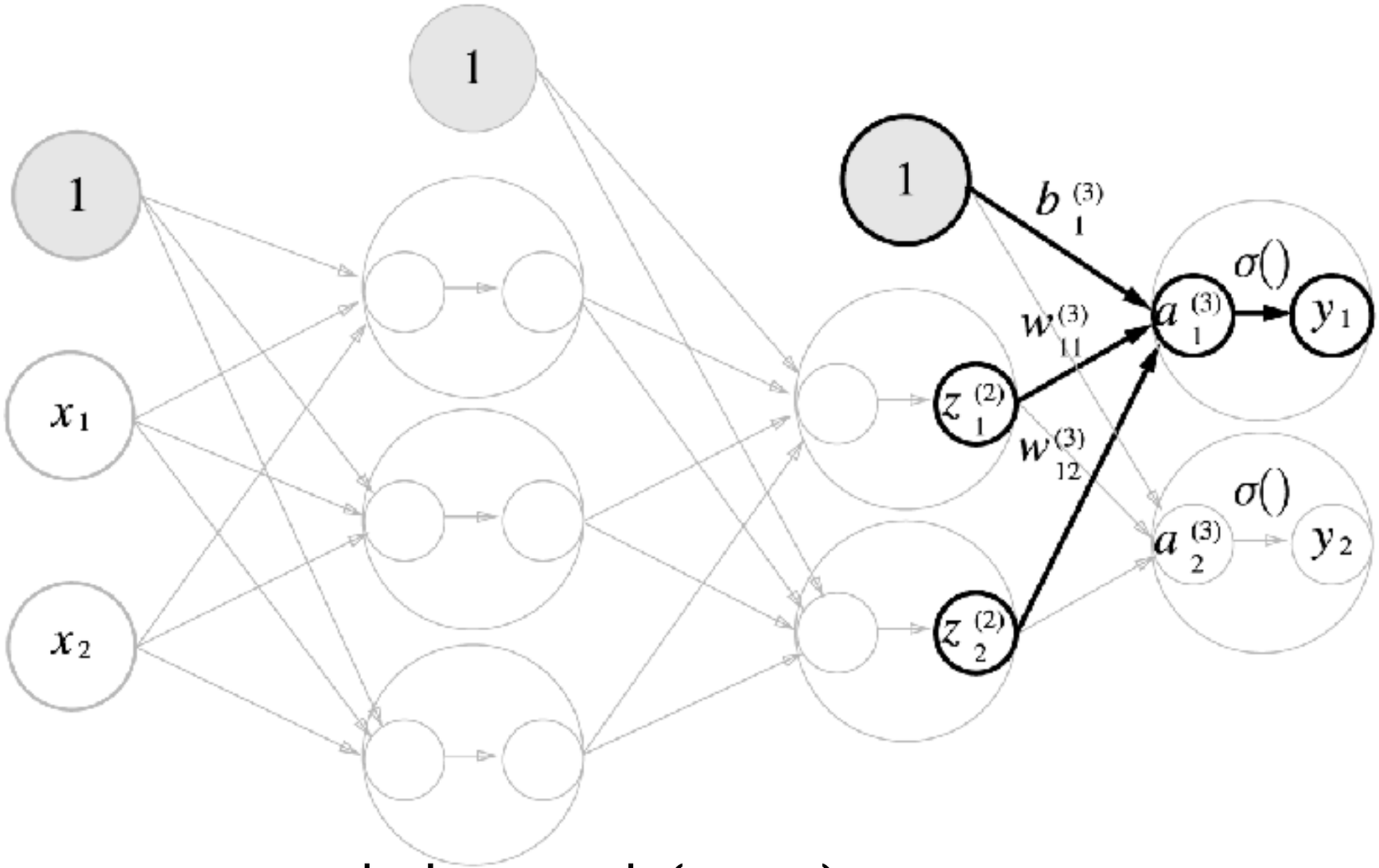
$$\begin{pmatrix} a_1^{(1)} & a_2^{(1)} & a_3^{(1)} \end{pmatrix} = \begin{pmatrix} x_1 & x_2 \end{pmatrix} \bullet \begin{pmatrix} w_{11}^{(1)} & w_{12}^{(1)} & w_{13}^{(1)} \\ w_{21}^{(1)} & w_{22}^{(1)} & w_{23}^{(1)} \end{pmatrix} + \begin{pmatrix} b_1^{(1)} & b_2^{(1)} & b_3^{(1)} \end{pmatrix}$$



사이토 고키 (2017)

$$A^{(1)} = X \bullet W^{(1)} + B^{(1)}$$

$$\begin{pmatrix} a_1^{(1)} & a_2^{(1)} & a_3^{(1)} \end{pmatrix} = \begin{pmatrix} x_1 & x_2 \end{pmatrix} \bullet \begin{pmatrix} w_{11}^{(1)} & w_{12}^{(1)} & w_{13}^{(1)} \\ w_{21}^{(1)} & w_{22}^{(1)} & w_{23}^{(1)} \end{pmatrix} + \begin{pmatrix} b_1^{(1)} & b_2^{(1)} & b_3^{(1)} \end{pmatrix}$$



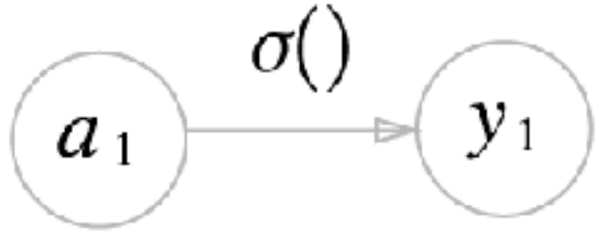
사이토 고키 (2017)

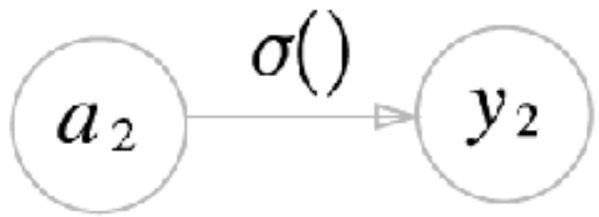
출력층의 활성화함수

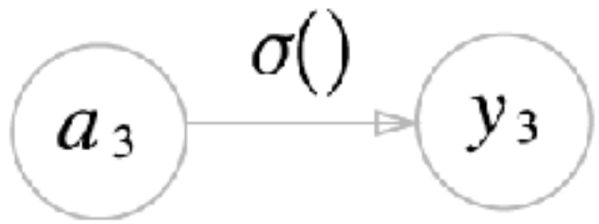
- 기계학습 문제에 따라 달라짐
  - 회귀 (regression) ⇒ identity function
    - 출력층의 수: 추정하고자 하는 벡터의 차원수
  - 분류 (classification) ⇒ softmax function
    - 출력층의 수: 분류하고자 하는 것의 갯수

Identity function

- $\sigma(x) = x$



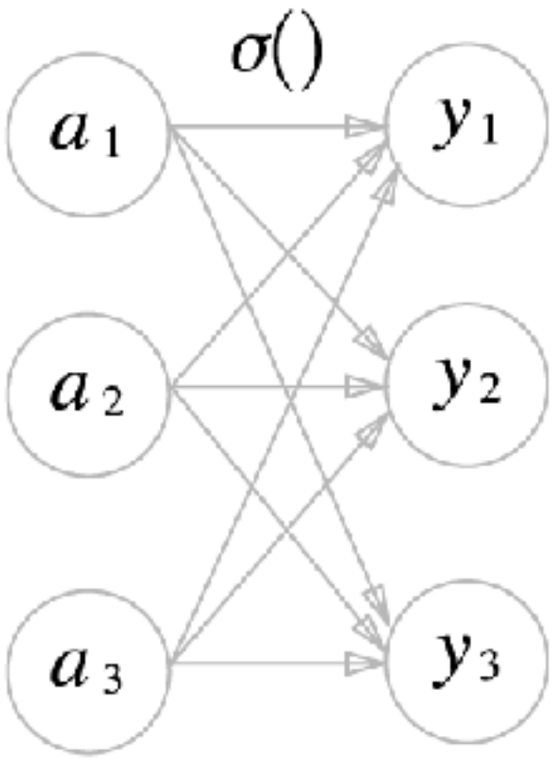




Softmax function

- 출력층의 모든 값(yk)의 합이 1
  - 0 ≤ yk ≤ 1
  - **Σ**yk = 1
    - 각 출력의 확률로 해석 가능
    - 지수이기 때문에 계산시 불안정한 문제 발생 (overflow 문제)
- 계산량 부담으로 추론단계에서는 그냥 max(yk)를 사용하기도 함
  - 학습 단계에서는 softmax 사용

$$y_k = \frac{\exp(a_k)}{\sum_{i=1}^{n} \exp(a_i)}$$



Resolving Overflow

- def softmax(a):
- c = np.max(a)
- exp_a = np.exp(a-c)
- sum_exp_a = np.sum(exp_a)
- return exp_a/ sum_exp_a

$$y_k = \frac{\exp(a_k)}{\sum_{i=1}^{n}\exp(a_i)} = \frac{C\exp(a_k)}{C\sum_{i=1}^{n}\exp(a_i)}$$

$$= \frac{\exp(a_k + \log C)}{\sum_{i=1}^{n}\exp(a_i + \log C)}$$

$$= \frac{\exp(a_k + C')}{\sum_{i=1}^{n}\exp(a_i + C')}$$

신경망 학습

물고기 Ch4

Namun Cho/ mailto:namun@snu.ac.kr

오차역전파법
Backpropagation

물고기 Ch5

Namun Cho/ mailto:namun@snu.ac.kr