Introduction to Artificial Intelligence

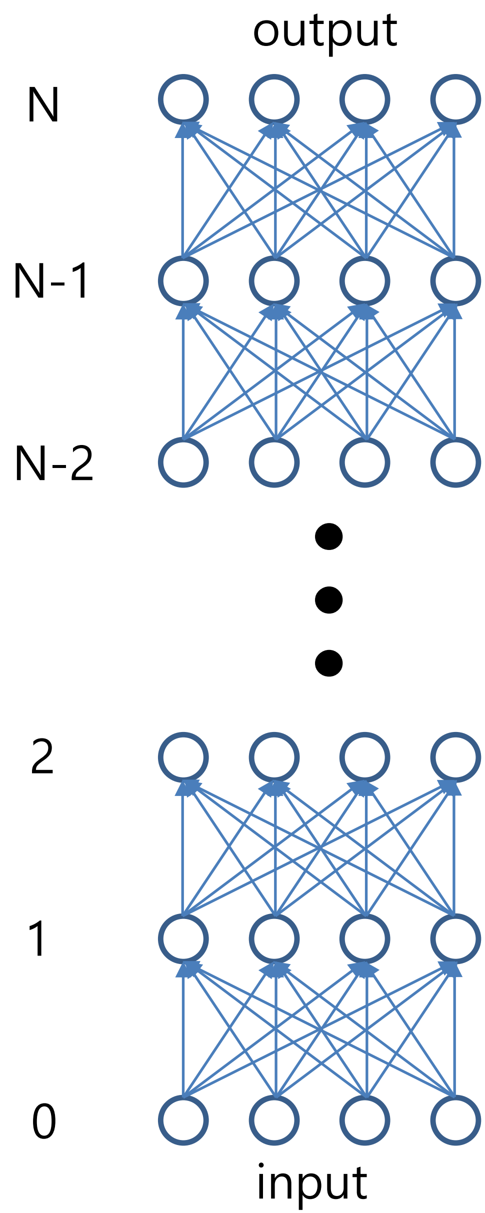
3. Multi-layer perceptron의 학습

2019

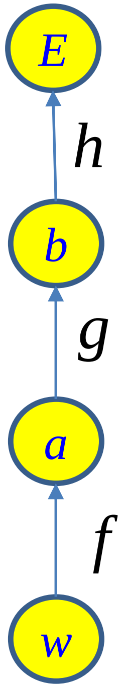
Ha-Jin Yu, Dept. of Computer Science, University of Seoul
서울시립대학교 컴퓨터과학부 유하진

HJYU@UOS.AC.KR

Multi-layer perceptron의 학습



Multi-layer perceptron의 학습 원리



$a = f(w)$

$b = g(a)$

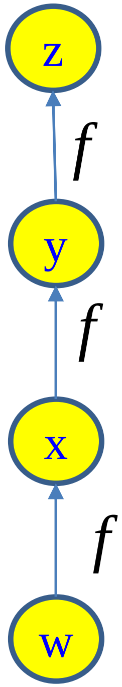
$E = h(b)$

$\Rightarrow$ $E = h(\,g(\,f(w)\,)\,)$

$$\frac{\partial E}{\partial w} = \frac{\partial E}{\partial b}\frac{\partial b}{\partial w} = \frac{\partial E}{\partial b}\frac{\partial b}{\partial a}\frac{\partial a}{\partial w}$$

chain rule

A computational graph that results in **repeated subexpressions** when computing the gradient.



$$z = f(y)$$
$$y = f(x)$$
$$x = f(w)$$

$$\Rightarrow \quad z = f(\, f(\, f(w)\,)\,)$$

중복!

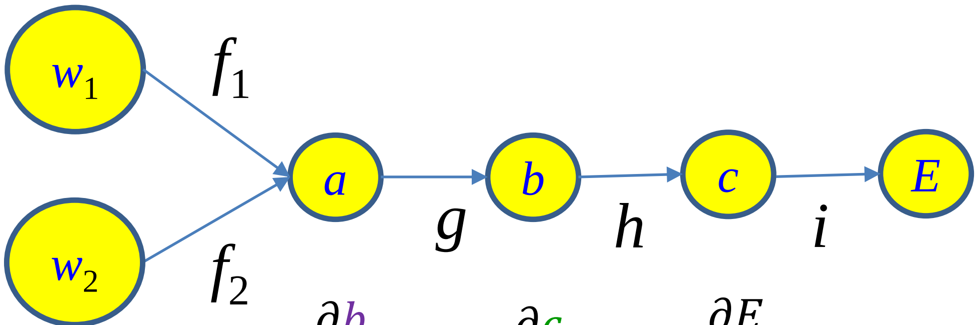
$$\frac{\partial z}{\partial w} = \boxed{\frac{\partial z}{\partial y}\frac{\partial y}{\partial x}}\frac{\partial x}{\partial w} \qquad \frac{\partial z}{\partial x} = \boxed{\frac{\partial z}{\partial y}\frac{\partial y}{\partial x}}$$

$$= f'(y)f'(x)f'(w)$$

$$= f'\Big(f\big(f(w)\big)\Big)\, f'\big(f(w)\big) f'(w)$$

- many subexpressions may be repeated several times within the overall expression for the gradient.

Back Propagation



$$\frac{\partial E}{\partial w_1} = \frac{\partial E}{\partial c}\frac{\partial c}{\partial b}\frac{\partial b}{\partial a}\frac{\partial a}{\partial w_1}$$

$$\frac{\partial E}{\partial w_2} = \frac{\partial E}{\partial c}\frac{\partial c}{\partial b}\frac{\partial b}{\partial a}\frac{\partial a}{\partial w_2}$$

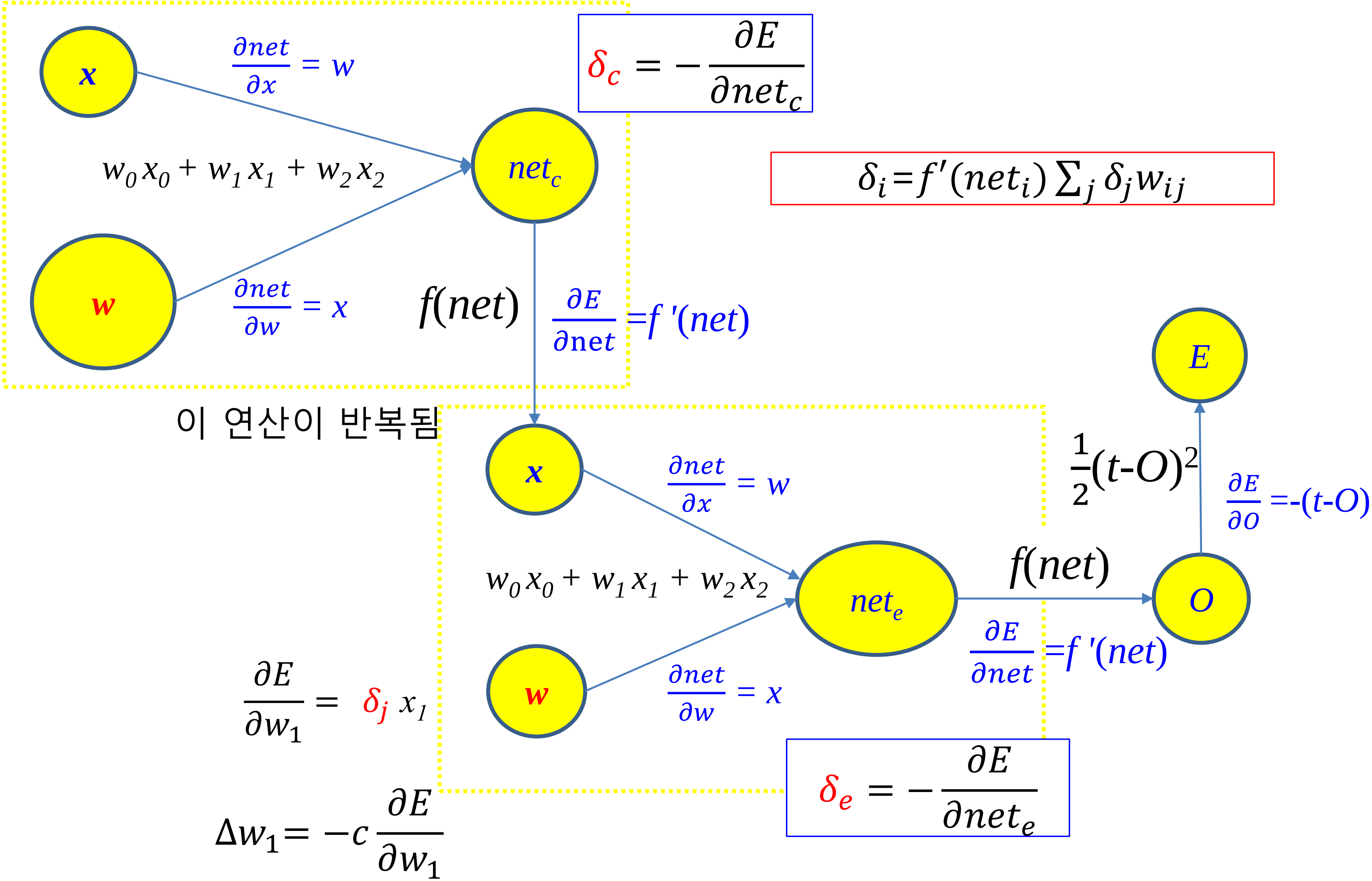
Duplicated!

define δ

$$\frac{\partial E}{\partial w_1} = \delta \frac{\partial a}{\partial w_1}$$

$$\frac{\partial E}{\partial w_2} = \delta \frac{\partial a}{\partial w_2}$$

← Dynamic programming



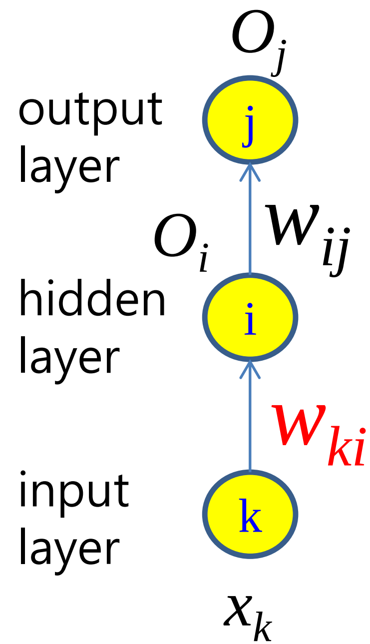
서울시립대학교
UNIVERSITY OF SEOUL

define

$$\delta_j = -\frac{\partial E}{\partial net_j}$$

$O_i = f(net_i)$

$$\frac{\partial O_i}{\partial net_i} = f'(net_i)$$



$$\frac{\partial E}{\partial w_{ki}} = \frac{\partial E}{\partial net_j}\frac{\partial net_j}{\partial O_i}\underbrace{\frac{\partial O_i}{\partial net_i}}_{\delta_i}\frac{\partial net_i}{\partial w_{ki}}$$

$$\frac{\partial net_j}{\partial O_i} = w_{ij}$$

$$\frac{\partial net_i}{\partial w_{ki}} = x_k$$

$$net_j = \sum_{i=0}^{n} O_i w_{ij}$$

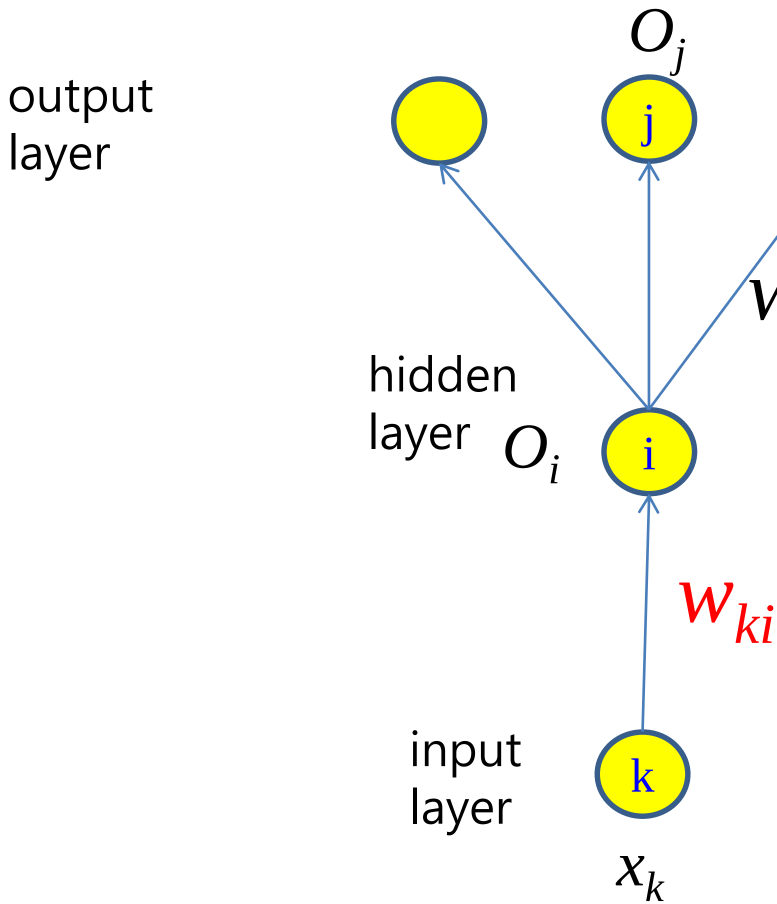
$$net_i = \sum_{i=0}^{n} x_k w_{ki}$$

$$\frac{\partial E}{\partial w_{ki}} = -\delta_j w_{ij} f'(net_i) x_k = -\delta_i x_k$$

$$\Delta w_{ki} = c f'(net_i) x_k \sum_j \delta_j w_{ij} = c\delta_i x_k$$

$$\delta_i = f'(net_i) \sum_j \delta_j w_{ij}$$

output이 여러 개 있으므로



$$\boxed{\delta_j = -\frac{\partial E}{\partial net_j}}$$

$$\boxed{\delta_i = f'(net_i) \sum_j \delta_j w_{ij}}$$

$$\frac{\partial E}{\partial w_{ij}^{n}} = \delta_j^{n} x_i^{n-1}$$

Backpropagation algorithm

$$\delta_j^N = \frac{\partial E}{\partial net_j^N} = \frac{\partial E}{\partial O_j}\frac{\partial O_j}{\partial net_j^N} = -(t_j - O_j)f'\left(net_j^N\right)$$

- Repeat for each $n = N, N\text{-}1, \ldots, 1$

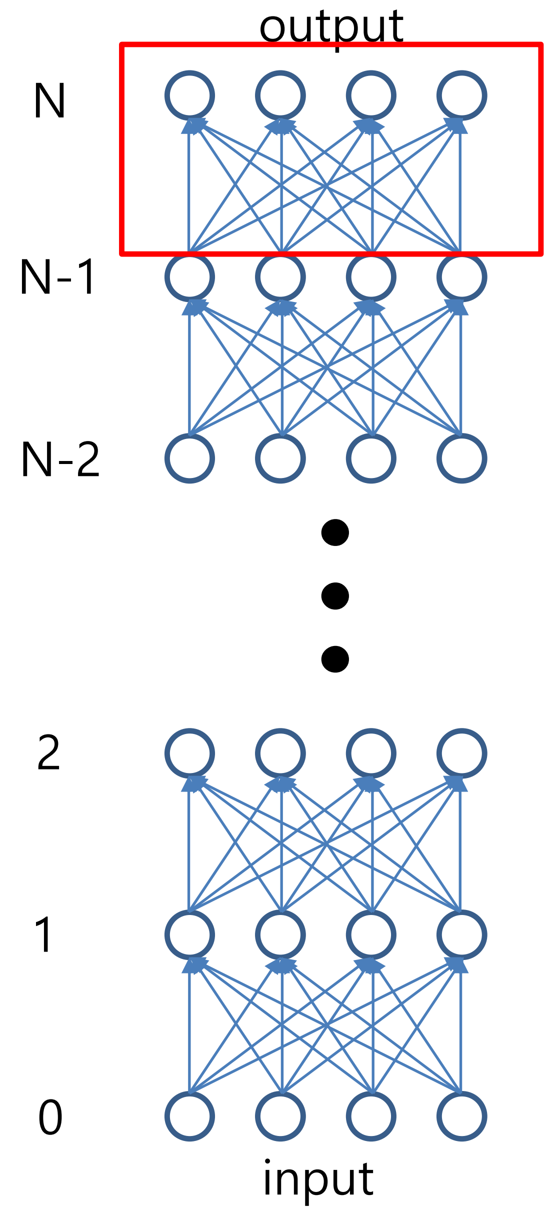
$$\frac{\partial E}{\partial w_{ij}^n} = \delta_j^n x_i^{n-1}$$

i = 0 … I

$$\Delta w_{ij}^n = -\mathrm{c}\frac{\partial E}{\partial w_{ij}^n} = -\mathrm{c}\delta_j^n x_i^{n-1}$$

$$\delta_j^{n-1} = f'(net_j^{n-1})\sum_j \delta_j^n w_{ij}^n$$

δ_i^n : layer n 에서 i번째 node 의 delta
net_j^n : layer n 에서 i번째 node 의 net
w_{ij}^n : : layer n-1 에서 i번째 node 와 layer n에서 j 번째 node 사이의 weight
x_i^{n-1}: layer n-1 에서 i번째 node 의 output



Procedure for backpropagation training

Initialize weights ← small random values.

While not stop

  Stop=true

  For each input vector

$$f(net_j) = f(\sum_{i=0}^{n} x_i w_i)$$

    Perform a forward sweep to find the actual output

    Obtain an error vector by comparing the actual and target output

    If the actual output is not within tolerance set STOP=FALSE

    Perform a backward sweep of the error vector $\boxed{\Delta w_{ki} = c f'(net_i) x_k \sum_j \delta_j w_{ij}}$

    Update weights

  End for

End while

Epoch : a complete cycle through all samples

(i.e. Each sample has been presented to the network)

LeCun's backpropagation algorithm

$$\boxed{\overline{\delta_j^N} = \frac{\partial E}{\partial o_i} = -(y_i - O_i)}$$

$$\boxed{\overline{\delta_j} = -\frac{\partial E}{\partial O_j}}$$

Repeat for each $n = N, N\text{-}1, \ldots, 1$

$$\delta_j^n = \overline{\delta_j^n} f'(net_j^n)$$

$$\frac{\partial E}{\partial w_{ij}^n} = \delta_j^n x_i^{n-1}$$

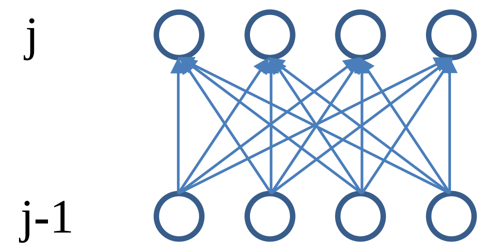
$$\Delta w_{ij}^n = -c \frac{\partial E}{\partial w_{ij}^n}$$

$$\overline{\delta_j^{n-1}} = \sum_j \delta_j^n w_{ij}^n$$

기존: 다음층에 $\delta_j^n = \overline{\delta_j^n} f'(net_i^n)$ 를 넘겨준다.

LeCun: $\overline{\delta_j^n}$ 만 넘겨준다.

→ Layer 별로 독립적인 계산 가능



Stopping criterion

- Training continues until …
  - 매 epoch 간의 평균 error 변화가 일정 값 이하 일때
    - example) a tolerance of 0.01
  - 또는 Training sample 에 대한 output 과 원하는 값의 차이가 일정 값 이하 일때
  - 또는 미리 정한 iteration 수
- If a network meets the tolerance during training → **converged**

Project #3 Multi-Layer perceptron 구현

- 실험 과제
(1) AND, OR, XOR 구분 실험
(2) 도우넛 모양 구분 실험 (아래 데이터 이용)

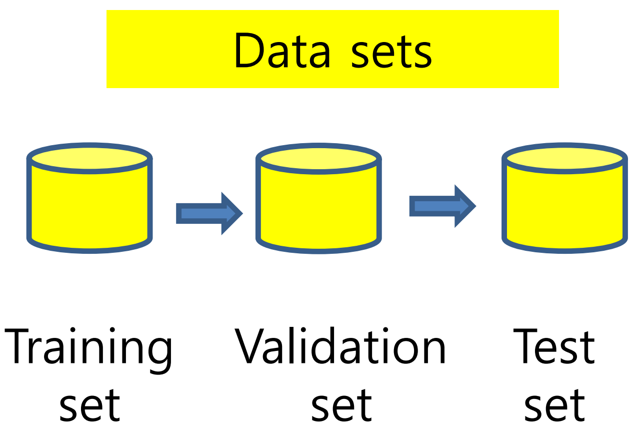
- Layer 수, Layer 당 node 수는 변수로 지정할 것.
- weight는 행렬 형식으로 파일에 저장
- Learning 과정을 그래프로 보여주기 (X1, X2 2차원 직선 그래프).
- 각 노드마다 직선을 그림으로 표시.
- iteration에 따른 Error 그래프
- 구현언어: C, C++
- 제출물: 프로그램, 결과 보고서
실행 10%, 출력 10, 주석 10, 완성도 25, 오류 10, 창의 10 보고서 25%

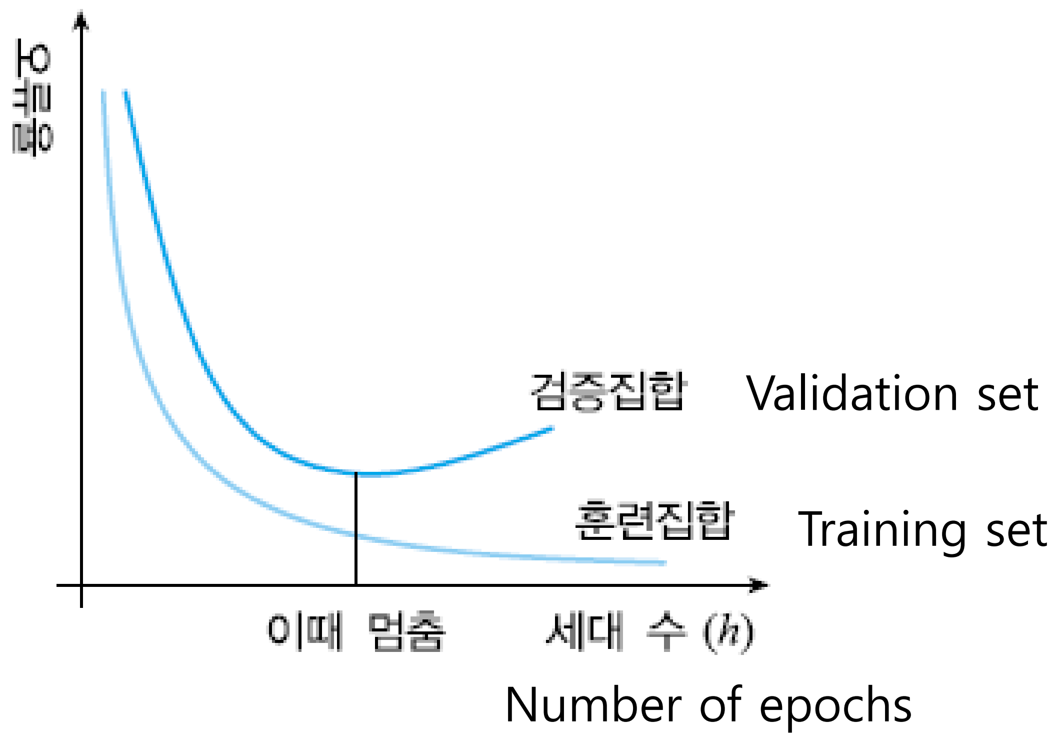
- 도우넛 모양 데이터

```
float train_set_x[][2] = {{0.,0.},
                          {0.,1.},
                          {1.,0.},
                          {1.,1.},
                          {0.5,1.},
                          {1.,0.5},
                          {0.,0.5},
                          {0.5,0.},
                          {0.5,0.5}} ;
float train_set_y[] = {0,0,0,0,0,0,0,0,1} ;
```

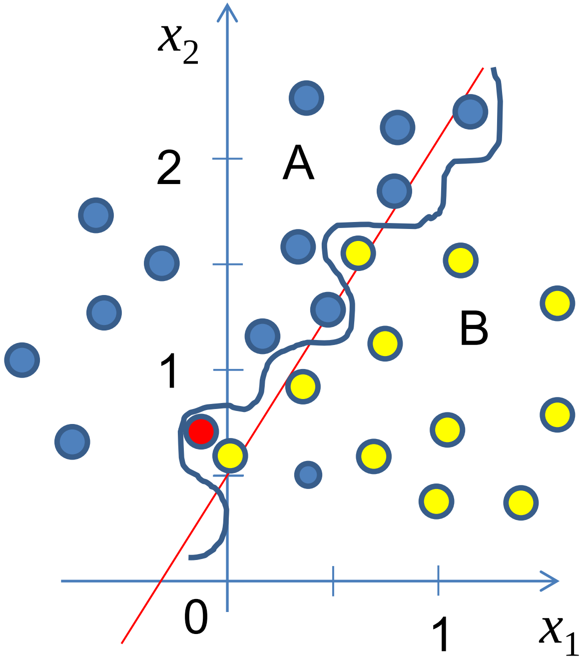
Generalization

- **Generalization** : training 에 사용되지 않은 data 에 대한 성능
- **Overfitting**: 모델 파라미터에 비하여 학습 데이터가 너무 적은 경우



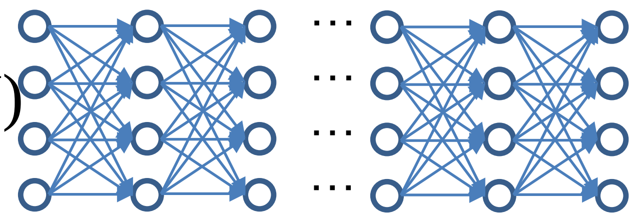


Overfitting



Neural Network Structures

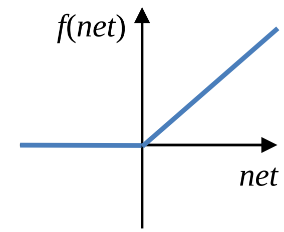
- Fully connected neural networks
- Convolutional neural networks (CNN)
- Recurrent neural networks (RNN)
- Restricted Boltzmann machine
- Gated
- Attention based
- Generative adversarial networks (GAN)
- Wavenet



Deep Neural Network 학습의 문제점

$$\frac{\partial E}{\partial w}=\frac{\partial E}{\partial h}\frac{\partial h}{\partial g}\cdots\cdots\cdots\frac{\partial a}{\partial f}\frac{\partial f}{\partial w}$$

- Vanishing gradient problem:
  - Gradient 가 단계를 지날 수록 작아진다
  - → Bottom-up layerwise unsupervised pre-training 으로 해결
  - → Rectified linear unit (ReLU)

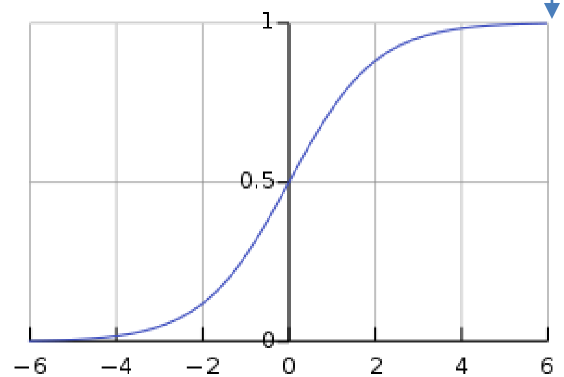


- Overfitting problem
  - Deep networks 가 shallow networks 보다 성능이 낮아짐
  - 대량의 labeled data 필요
  - → 대량의 unlabeled data 로 해결
  - → Dropout
- Local minima 에 빠진다 ?
  - → Unsupervised pre-training 으로 해결

Sigmoid 함수의 기울기

$$\boxed{\Delta w_{ki} = c f'(net_i) x_k \sum_j \delta_j w_{ij} = c\delta_i x_k}$$

f(net)



$f'(net_i)$ → 0

Δw_{ki} → 0

Update → 0

기울기 = 1

기울기 → 0