

# 딥러닝 기초(Fully Connected)

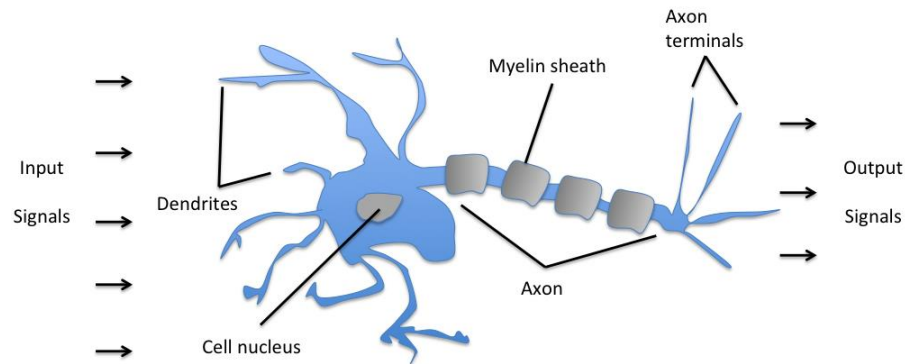
---

# Neural Network

---

# Perceptron

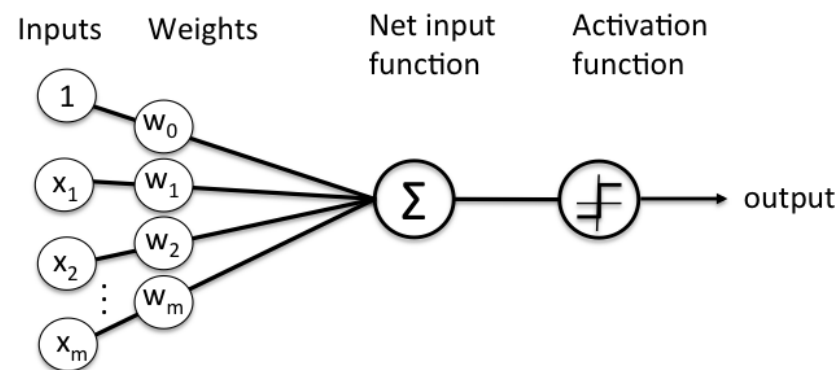
퍼셉트론은 1957년 로젠트발트가 고안한 알고리즘



Schematic of a biological neuron.

$$\phi(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

$$z = w_0x_0 + w_1x_1 + \dots + w_mx_m = \sum_{i=0}^m x_iw_i = w^T x$$



Schematic of Rosenblatt's perceptron.

# Perceptron

---

$$\phi(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

$$z = w_0x_0 + w_1x_1 + \dots + w_mx_m = \sum_{i=0}^m x_iw_i = \mathbf{w}^T \mathbf{x}$$

$$\vec{\mathbf{w}} = (w_0, \dots, w_d)^T = (0, \dots, 0)^T \in \mathbb{R}^{d+1} \quad \leftarrow \text{parameters} := \text{Initialize}()$$

**for**  $i := 1..M$ : iteration (epoch/time).  $M >$  sample size.  
(algorithm makes multiple passes over data.)

$$(\vec{\mathbf{x}}^{(t)}, y^{(t)}) := \text{ReceiveInstance}()$$

$$\hat{y}^{(t)} := \text{sign}(\vec{\mathbf{w}}^T \cdot \vec{\mathbf{x}}^{(t)}) \quad \leftarrow \hat{y}^{(t)} := \text{Predict}(\vec{\mathbf{x}}^{(t)}) \in \{-1, +1\}$$

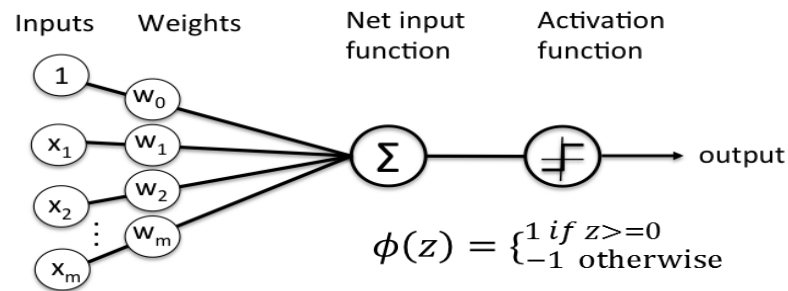
$$\text{if } (\hat{y}^{(t)} \neq y^{(t)}):$$
$$\vec{\mathbf{w}} := \vec{\mathbf{w}} + y^{(t)} \cdot \vec{\mathbf{x}}^{(t)}$$

parameters := Update(...)

**return**  $\vec{\mathbf{w}}$  return parameters

Attempting to minimize:

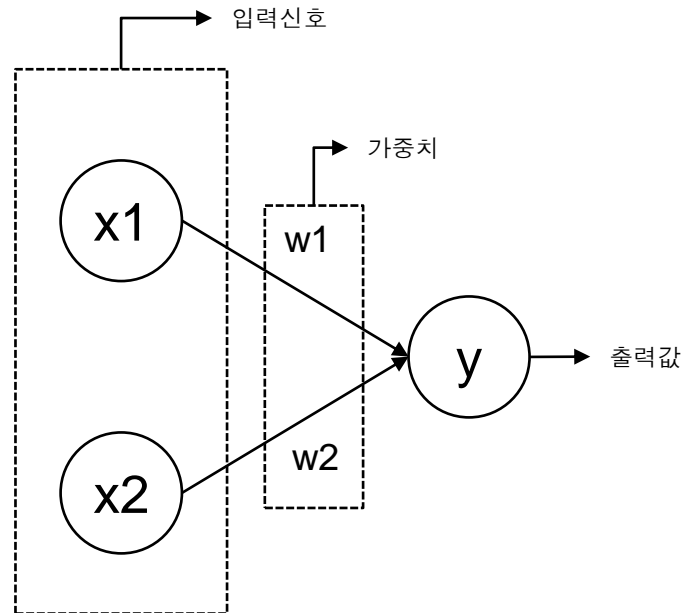
# Perceptron vs. Linear Regression



$$z = w_0x_0 + w_1x_1 + \dots + w_mx_m = \sum_{i=0}^m x_i w_i = w^T x$$

The diagram shows the linear regression equation  $Y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_p x_p + \varepsilon$  with several annotations. A red circle around  $Y$  is labeled "response, dependent variable, observation, 'y-variable'". A green circle around  $x_1$  is labeled "predictor, 'x-variable', independent variable, explanatory variable". An orange circle around  $\beta_2$  is labeled "coefficient". A blue bracket under the terms  $\beta_1 x_1 + \beta_2 x_2 + \dots + \beta_p x_p$  is labeled "linear predictor". A purple circle around  $\varepsilon$  is labeled "random error, 'noise'".

# Perceptron



$$y = \begin{cases} 0 & (w_1x_1 + w_2x_2 \leq \theta) \\ 1 & (w_1x_1 + w_2x_2 > \theta) \end{cases}$$

가중치  $w$ 는 전류에서 말하는 저항에 해당한다. 저항은 전류의 흐름을 억제하는 매개변수로 저항이 낮을수록 큰 전류가 흐른다. 하편 퍼셉트론의 가중치는 그 값이 클수록 강한 신호를 흘려보낸다.

# Perceptron

논리 회로

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

<AND>

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

<OR>

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

<NOR>

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

<NAND>

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

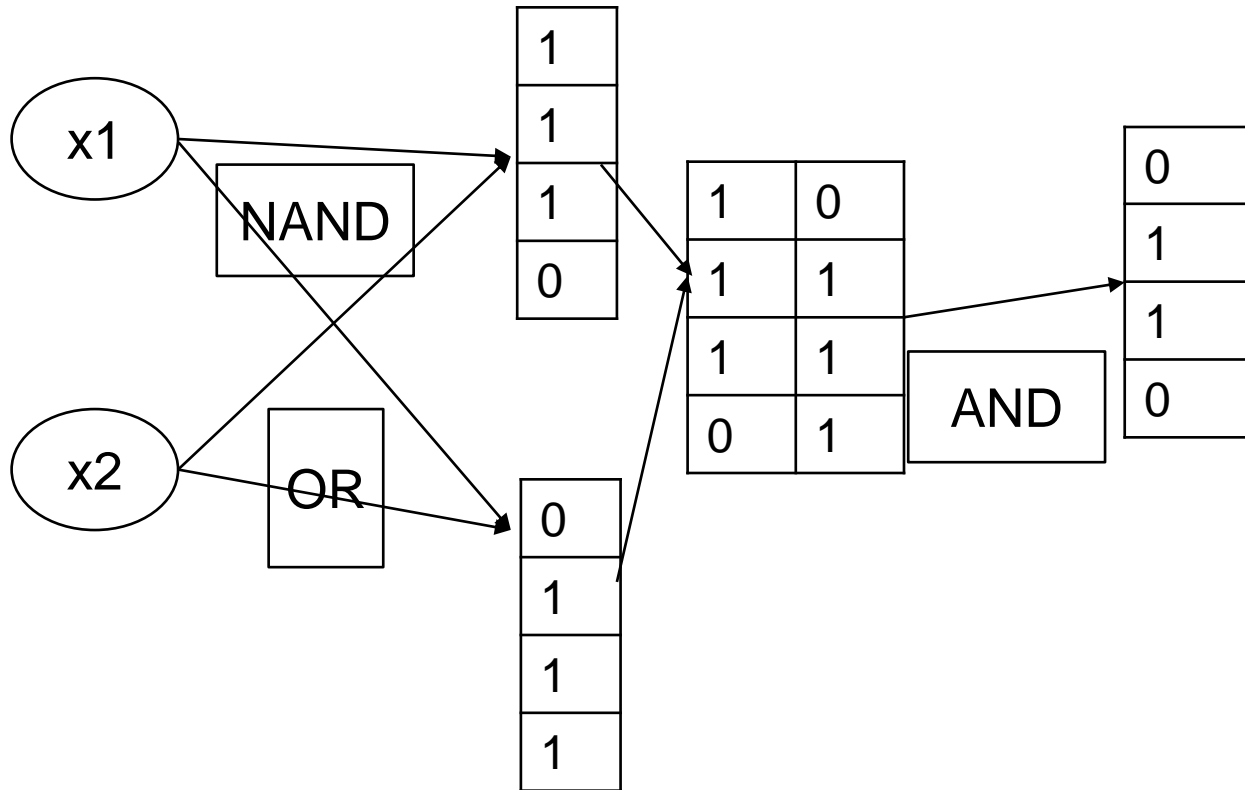
<XOR>

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

<XNOR>

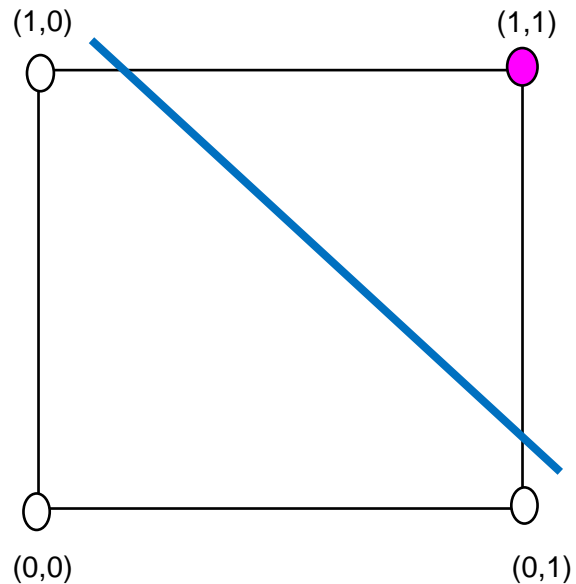
# Perceptron

---





# Perceptron AND

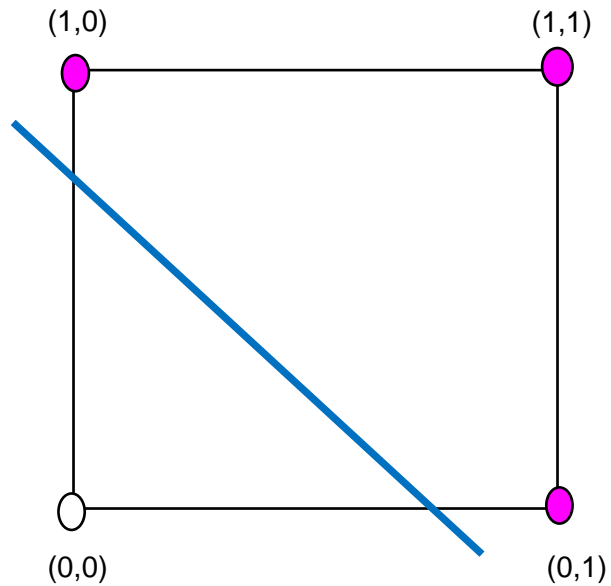


$w_1 = 0.5$   
 $w_2 = 0.5$   
 $\theta = 0.6$

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

$$y = \begin{cases} 0 & (w_1x_1 + w_2x_2 \leq \theta) \\ 1 & (w_1x_1 + w_2x_2 > \theta) \end{cases}$$

# Perceptron OR

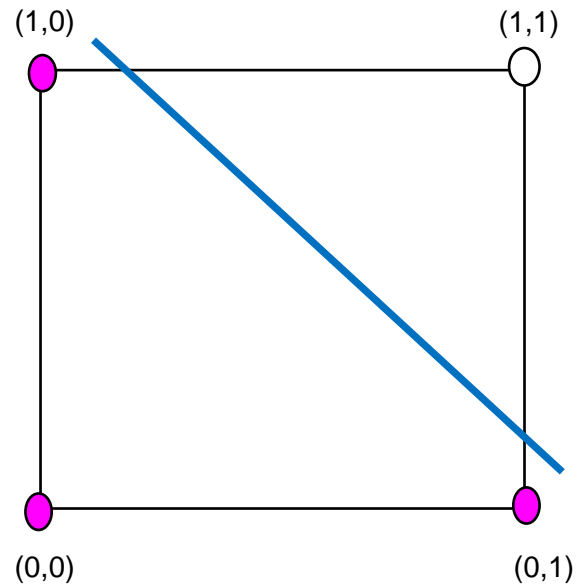


$w_1 = 0.5$   
 $w_2 = 0.5$   
 $\theta = 0.4$

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

$$y = \begin{cases} 0 & (w_1x_1 + w_2x_2 \leq \theta) \\ 1 & (w_1x_1 + w_2x_2 > \theta) \end{cases}$$

# Perceptron NAND



$w_1 = ?$

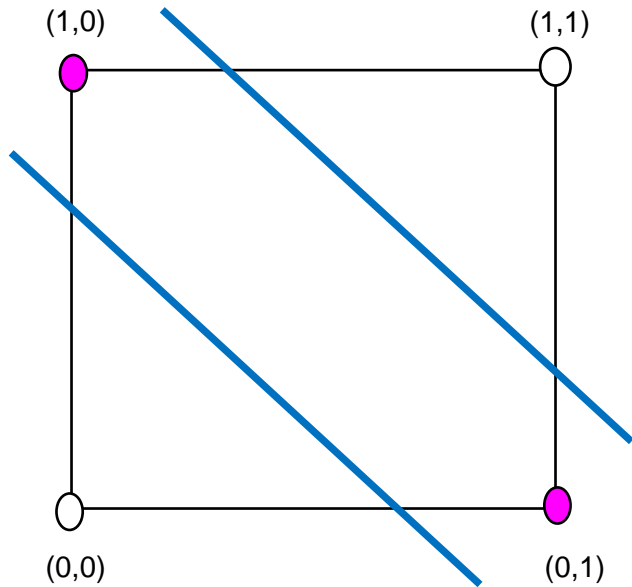
$w_2 = ?$

$\theta = ?$

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

$$y = \begin{cases} 0 & (w_1x_1 + w_2x_2 \leq \theta) \\ 1 & (w_1x_1 + w_2x_2 > \theta) \end{cases}$$

# Perceptron XOR



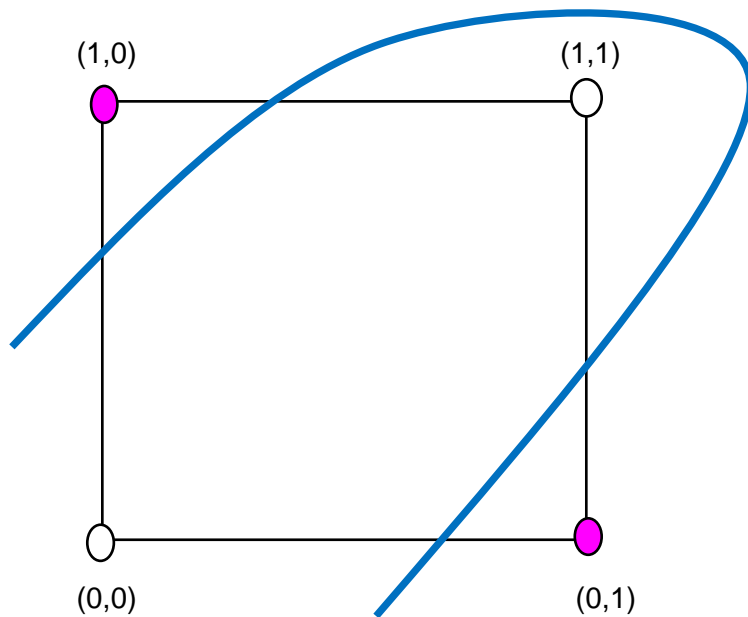
$x_1$	$x_2$	$y$
0	0	0
1	0	1
0	1	1
1	1	0

$w_1 = ?$   
 $w_2 = ?$   
 $\theta = ?$

존재 할까?

$$y = \begin{cases} 0 & (w_1x_1 + w_2x_2 \leq \theta) \\ 1 & (w_1x_1 + w_2x_2 > \theta) \end{cases}$$

# Perceptron XOR



$$y = \begin{cases} 0 & (w_1x_1 + w_2x_2 \leq \theta) \\ 1 & (w_1x_1 + w_2x_2 > \theta) \end{cases}$$

선형성을 만족 시키지  
못하고 비선형으로만 존재

# Perceptron

## XOR 구현

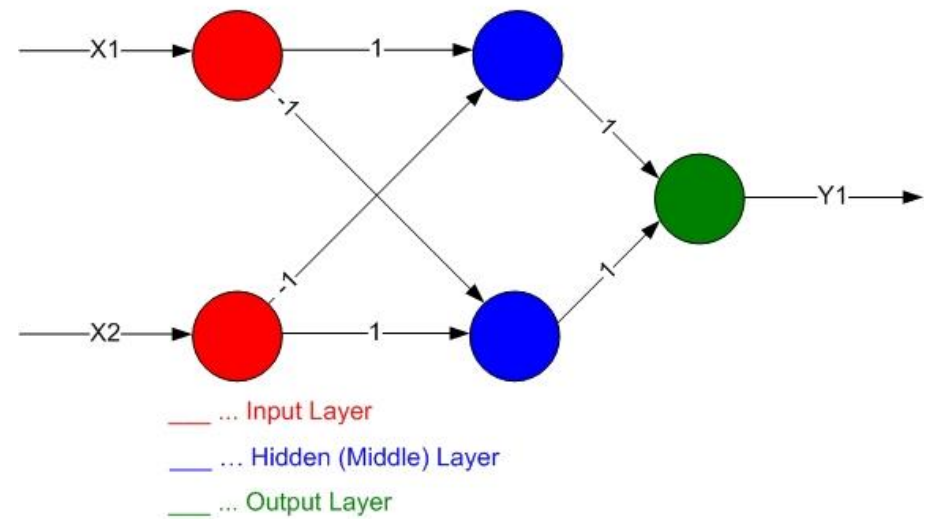
XOR

$x_1$	$x_2$	$s_1$	$s_2$	$y$
0	0	1	0	0
1	0	1	1	1
0	1	1	1	1
1	1	0	1	0

Diagram illustrating the implementation of XOR using NAND, AND, and OR gates:

- NAND**: Inputs  $x_1$  and  $x_2$  are connected to the inputs of a NAND gate.
- AND**: Inputs  $x_1$  and  $x_2$  are connected to the inputs of an AND gate.
- OR**: The outputs of the NAND and AND gates are connected to the inputs of an OR gate.

$$Y1 = \text{XOR}(X1, X2)$$



# Neural Network

---

## Perceptron 과 Neural Network(신경망)

### 퍼셉트론

비선형 같은 복잡한 함수도 표현 가능

컴퓨터가 수행하는 복잡한 처리도 표현할 수 있음

가중치를 설정하는 작업은 여전히 사람이 수동으로 조절

### 신경망

가중치 매개변수의 적절한 값을 데이터로부터 자동으로 학습

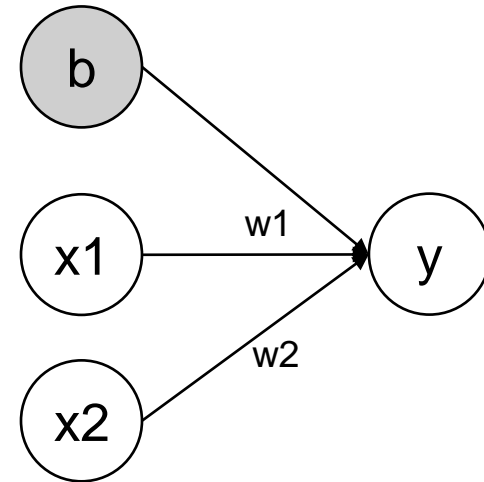
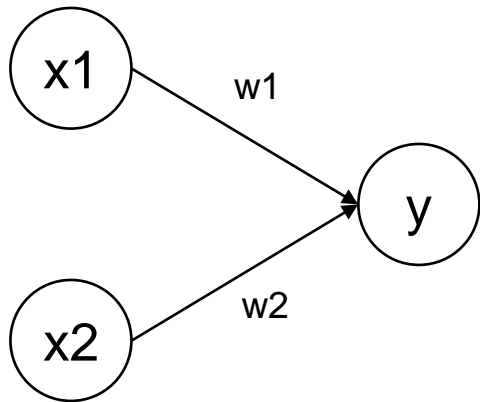
# Neural Network

## Perceptron 변환

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

⇒

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

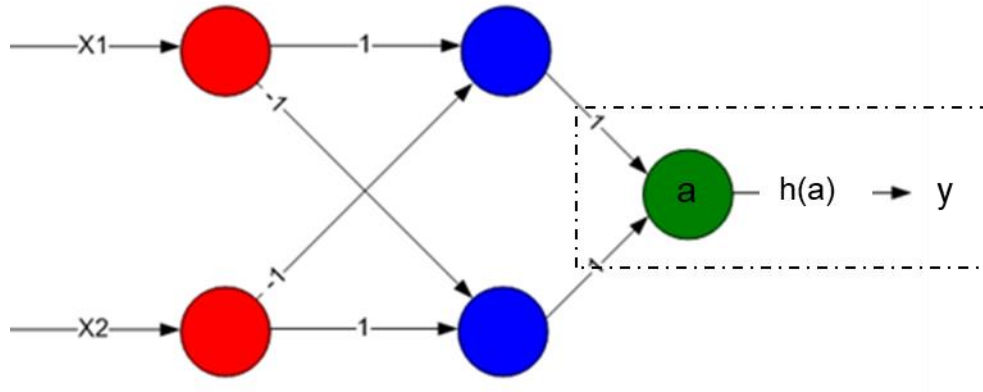


b는 편향을 나타내는 매개변수로 뉴런이 얼마나 쉽게 활성화 되는 지를 제어



# Neural Network

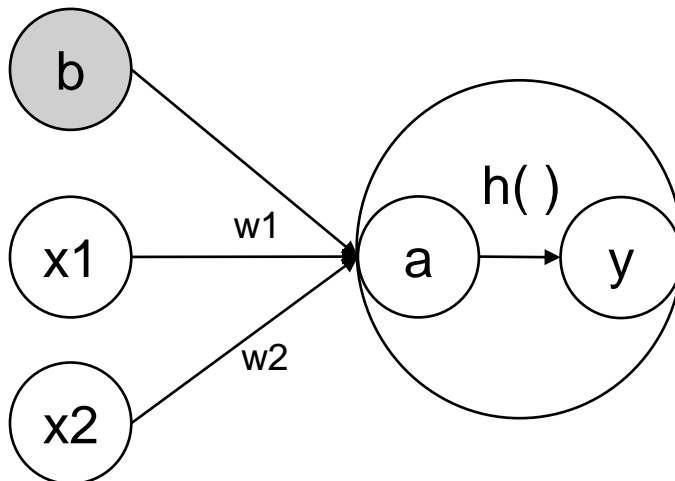
## 활성화 함수의 등장



$$a = w_1x_1 + w_2x_2 + b$$

$$y = h(a)$$

$h(a)$  는 입력 신호의 총합을 출력  
신호로 변환하는 함수이며 일반적으로  
**활성화** 함수라 함



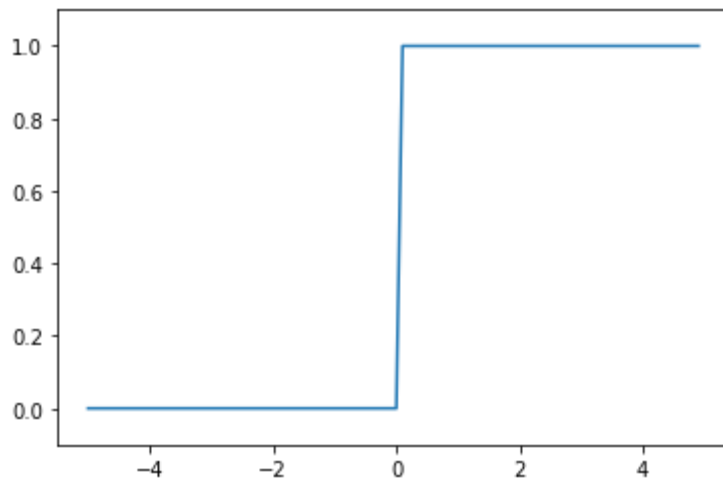
활성화 함수는 임계값을 경계로 출력을 바꿈

# Neural Network

---

## Step function (계단함수)

퍼셉트론의 활성화 함수



```
def step_function(x):  
    return np.array(x > 0, dtype=np.int)
```

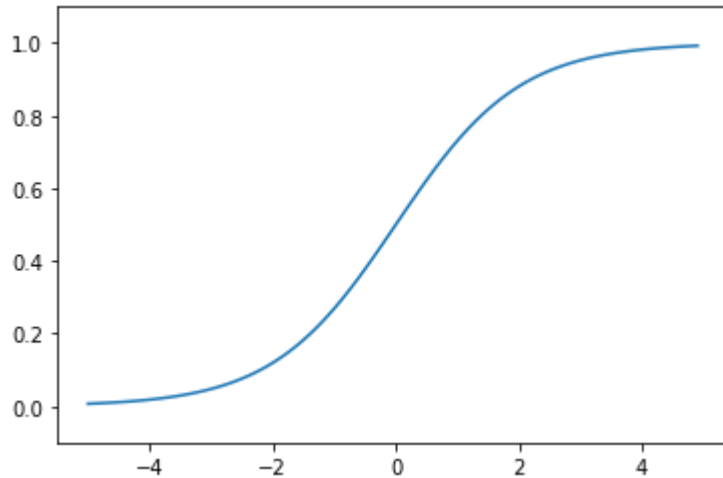
```
X = np.arange(-5.0, 5.0, 0.1)  
Y = step_function(X)  
plt.plot(X, Y)  
plt.ylim(-0.1, 1.1) # y축의 범위 지정  
plt.show()
```

# Neural Network

---

## Sigmoid function (S자 함수)

초기 신경망에서 유행했던 활성화 함수



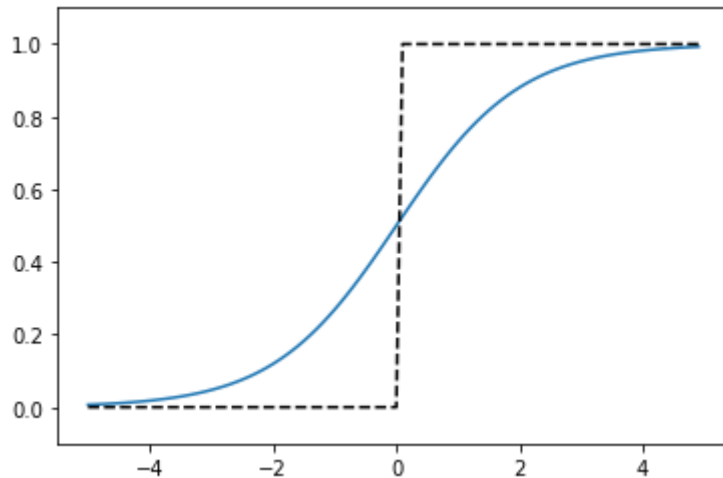
$$f(x) = \frac{1}{1 + e^{-x}}$$

```
def sigmoid(x):  
    return 1 / (1 + np.exp(-x))
```

```
X = np.arange(-5.0, 5.0, 0.1)  
Y = sigmoid(X)  
plt.plot(X, Y)  
plt.ylim(-0.1, 1.1)  
plt.show()
```

# Neural Network

## 계단함수와 시그모이드 함수 비교



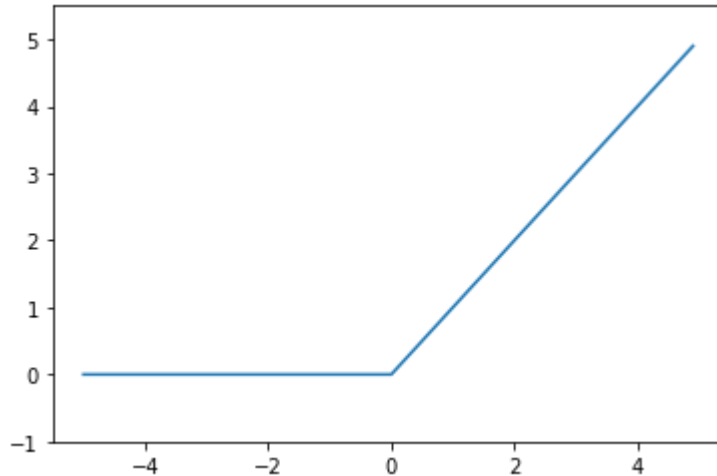
- ✓ 차이점  
매끄러움, 계단함수는 0을 기준으로 급격히 변하는 값을 출력(0,1), 시그모이드 함수는 천천히 변하는 값을 출력(연속된)
- ✓ 공통점  
0과 1사이의 값을 출력,  
**비선형 함수**

비선형 함수는 직선 하나로는 그릴 수 없는 함수를 뜻하며 신경망에서는 활성화 함수로 비선형 함수를 사용해야 함

선형 함수의 문제는 층을 아무리 깊게 해도 은닉층이 없는 네트워크로도 똑같은 기능을 할 수 있음  
즉, 은닉층이 없는 것과 같음  
다시, XOR문제 해결 불가능

# Neural Network

## ReLU(Rectified Linear Units)



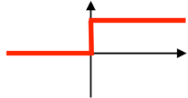
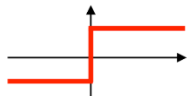
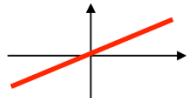
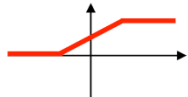
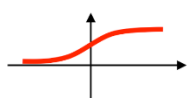
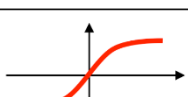
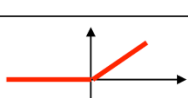
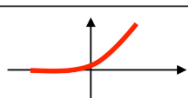
입력값이 0을 넘으면 그 입력값 그대로 출력하고 입력값이 0보다 작거나 같으면 0을 출력

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

예전에는 활성화 함수로 시그모이드 함수를 주로 사용했으나 최근에는 ReLU 함수를 주요 사용

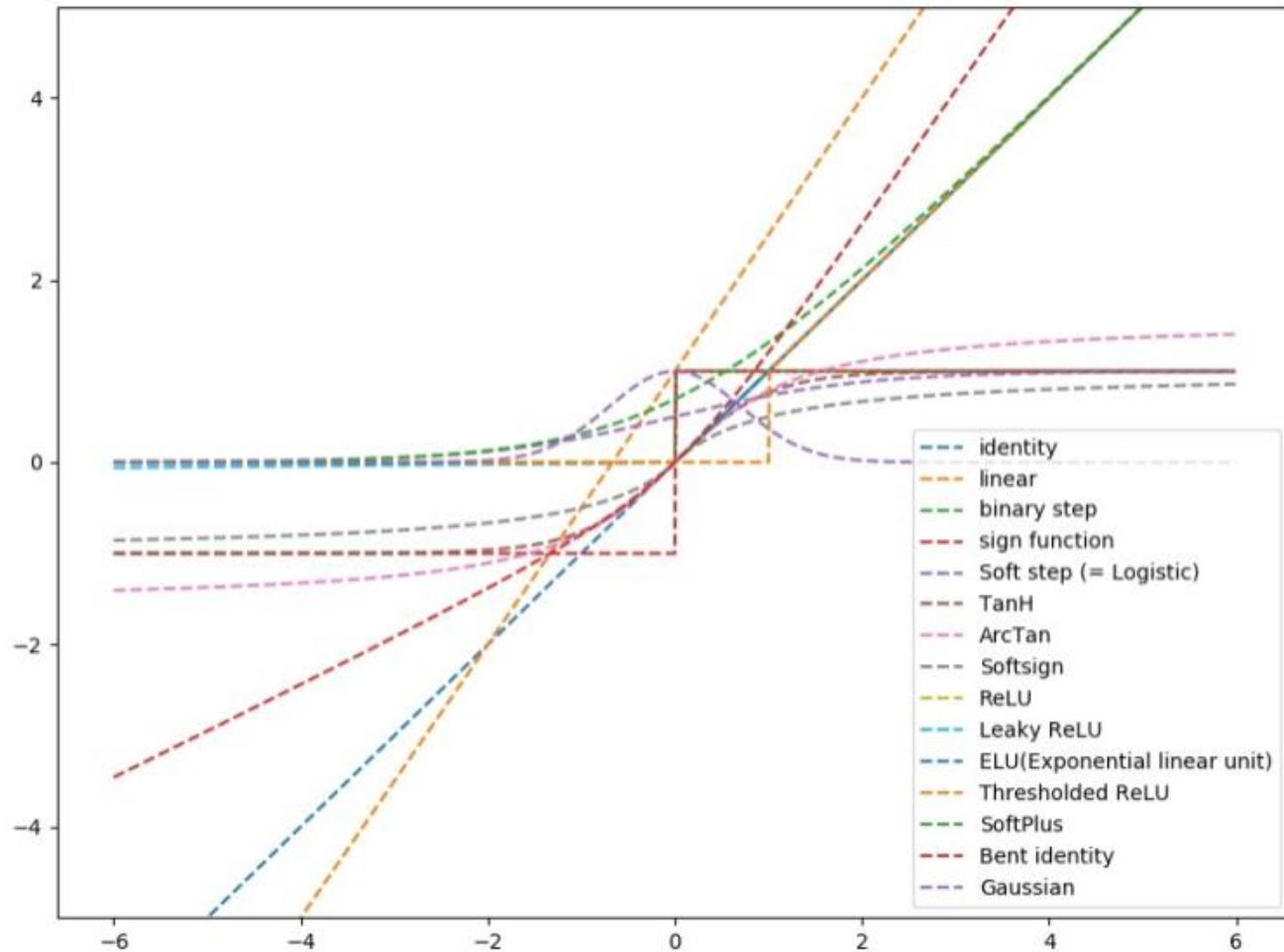
# Neural Network

## 여러가지 Activate Function

Activation function	Equation	Example	1D Graph
Unit step (Heaviside)	$\phi(z) = \begin{cases} 0, & z < 0, \\ 0.5, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Sign (Signum)	$\phi(z) = \begin{cases} -1, & z < 0, \\ 0, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Linear	$\phi(z) = z$	Adaline, linear regression	
Piece-wise linear	$\phi(z) = \begin{cases} 1, & z \geq \frac{1}{2}, \\ z + \frac{1}{2}, & -\frac{1}{2} < z < \frac{1}{2}, \\ 0, & z \leq -\frac{1}{2}, \end{cases}$	Support vector machine	
Logistic (sigmoid)	$\phi(z) = \frac{1}{1 + e^{-z}}$	Logistic regression, Multi-layer NN	
Hyperbolic tangent	$\phi(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	Multi-layer Neural Networks	
Rectifier, ReLU (Rectified Linear Unit)	$\phi(z) = \max(0, z)$	Multi-layer Neural Networks	
Rectifier, softplus	$\phi(z) = \ln(1 + e^z)$	Multi-layer Neural Networks	

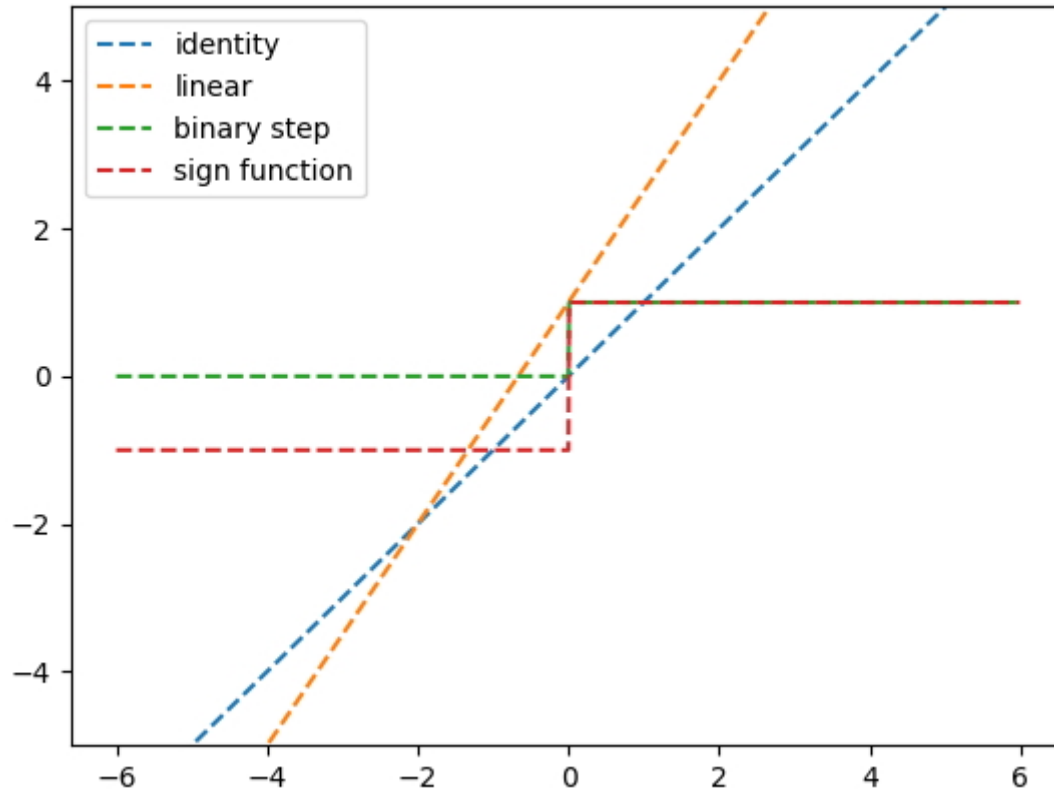
# Neural Network

## 여러가지 Activate Function



# Neural Network

## Activate Function – Linear, Step Function

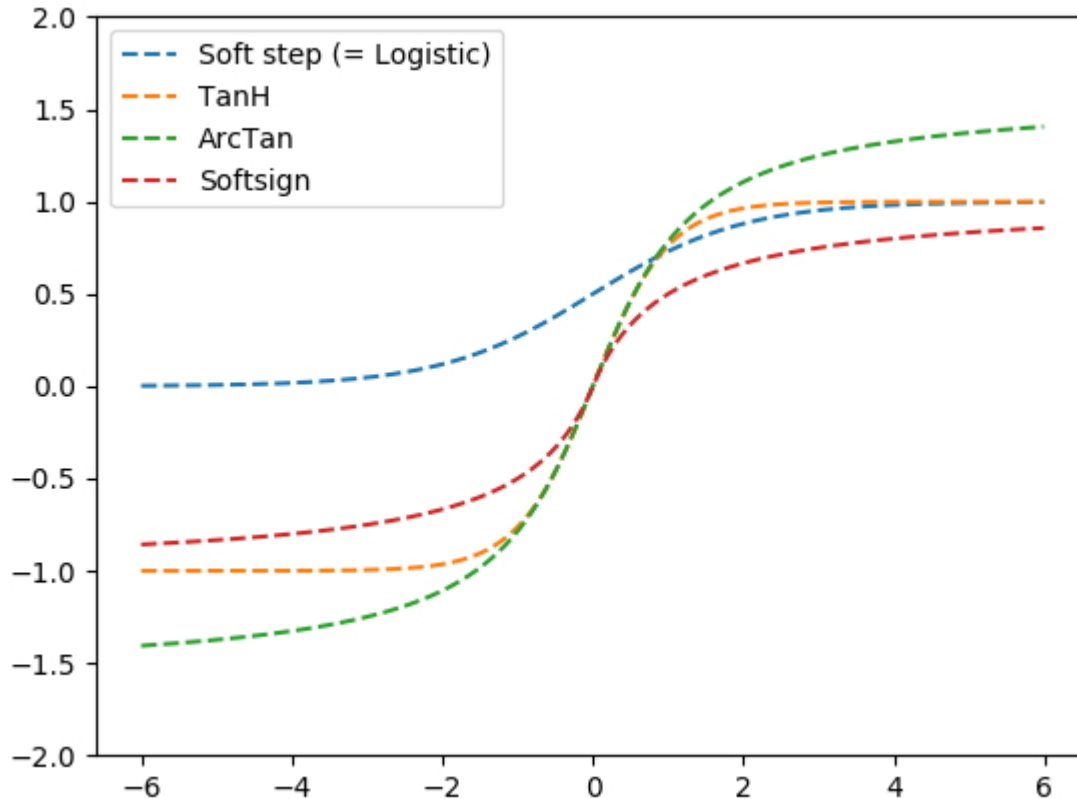


- 계산이 단조롭고 기울기가 정해져 있다
- 데이터 소실 문제가 발생하지 않는다.
- 데이터들이 선형성을 가진다.



# Neural Network

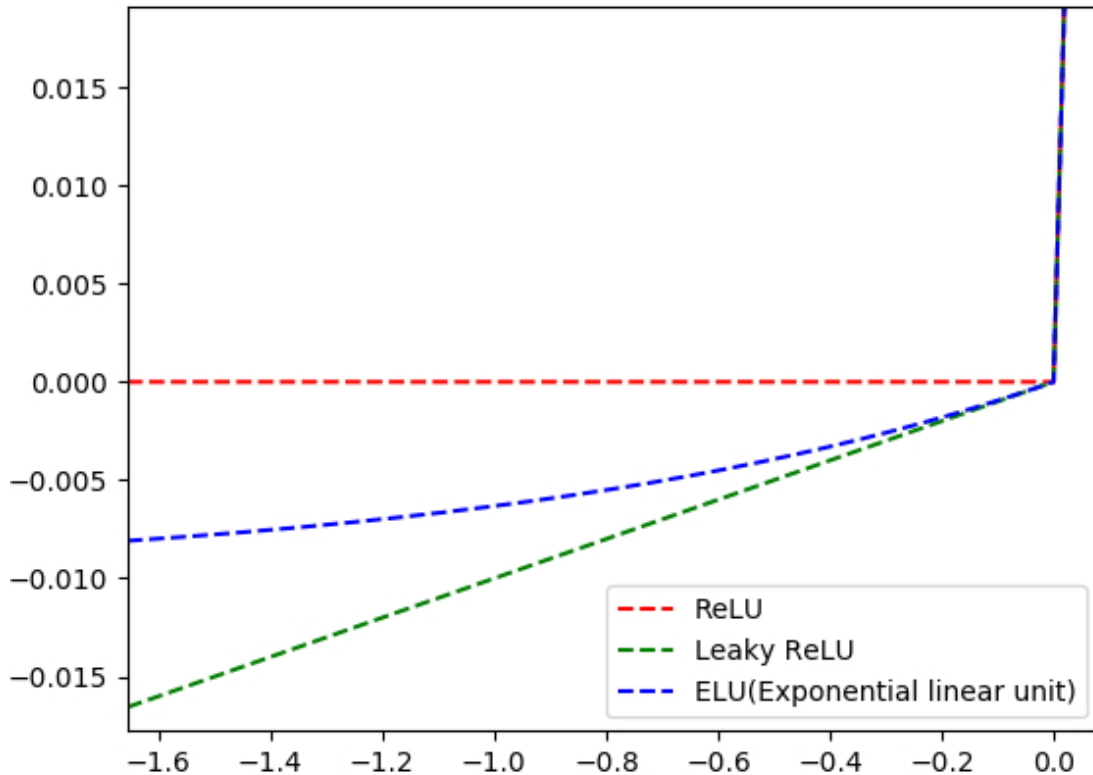
## Activate Function – Sigmoid 계열



- S 형태의 그래프
- 절대값의 크기가 커질수록 기울기 값이 0에 수렴
  - ✓ 기울기 소실의 문제점
- 비선형 데이터

# Neural Network

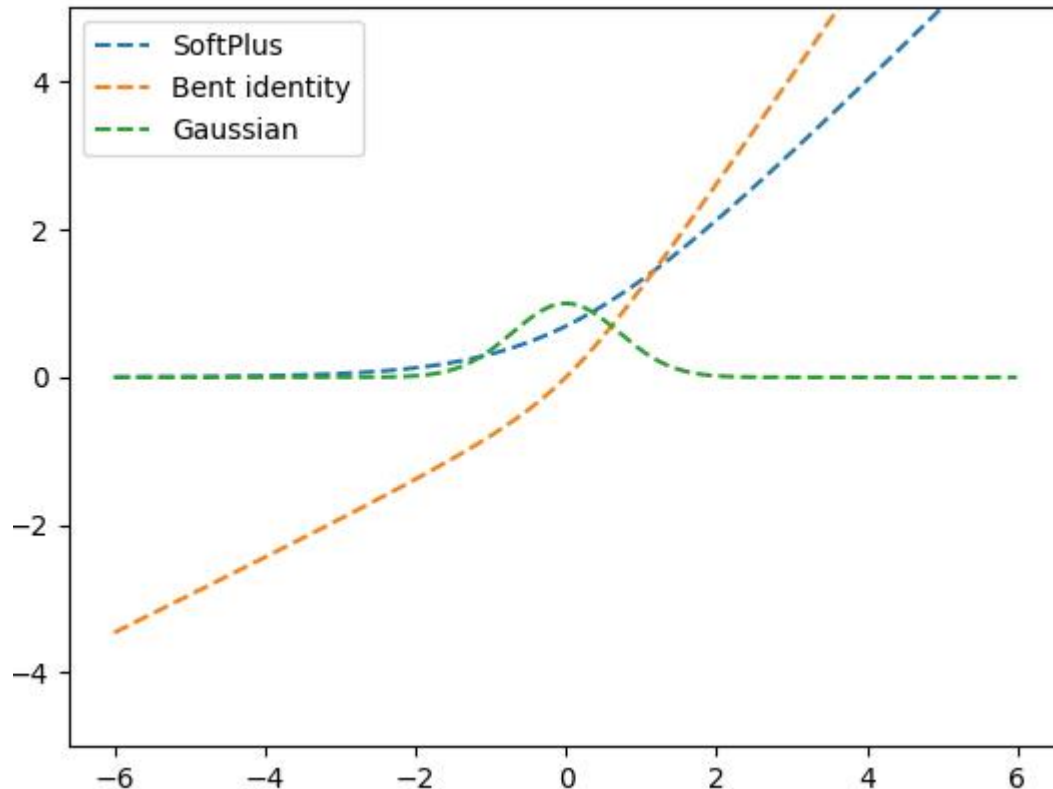
## Activate Function – ReLU 계열



- 기울기가 다른 두 직선의 결합 : 0을 기준으로 기울기 값이 두개로 갈린다.
- 가장 무난하고 Sigmoid 와 함께 많이 쓰이는 활성화 함수

# Neural Network

## Activate Function – 기타

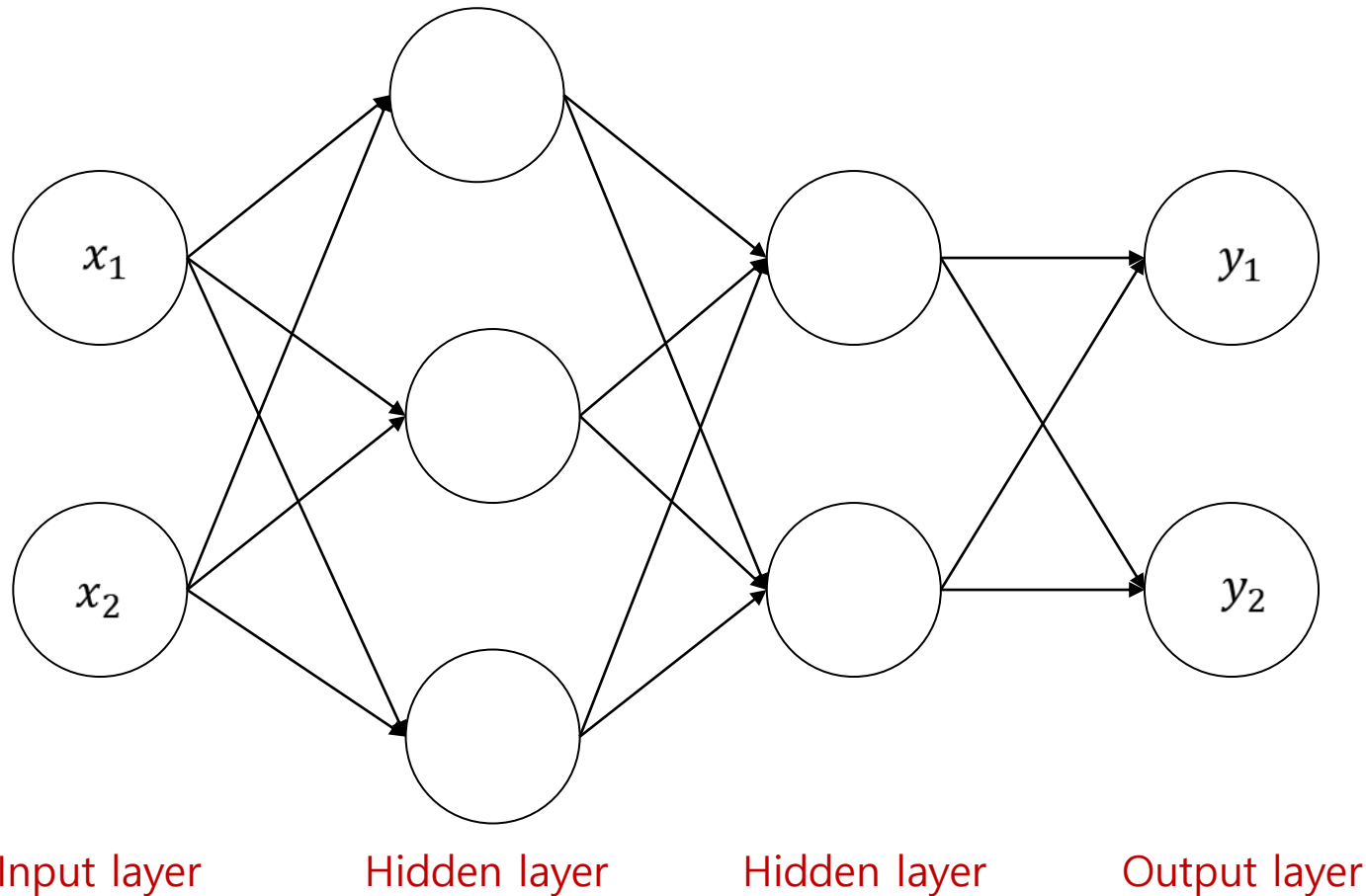


- 활성화 함수의 선택 기준 : 데이터 형태와 성질에 맞는 함수 선택
- 이 외에도 많은 활성화 함수가 존재하며, 적절한 함수를 찾지 못했을 시 직접 만들어 사용하여도 무방하다

# Neural Network

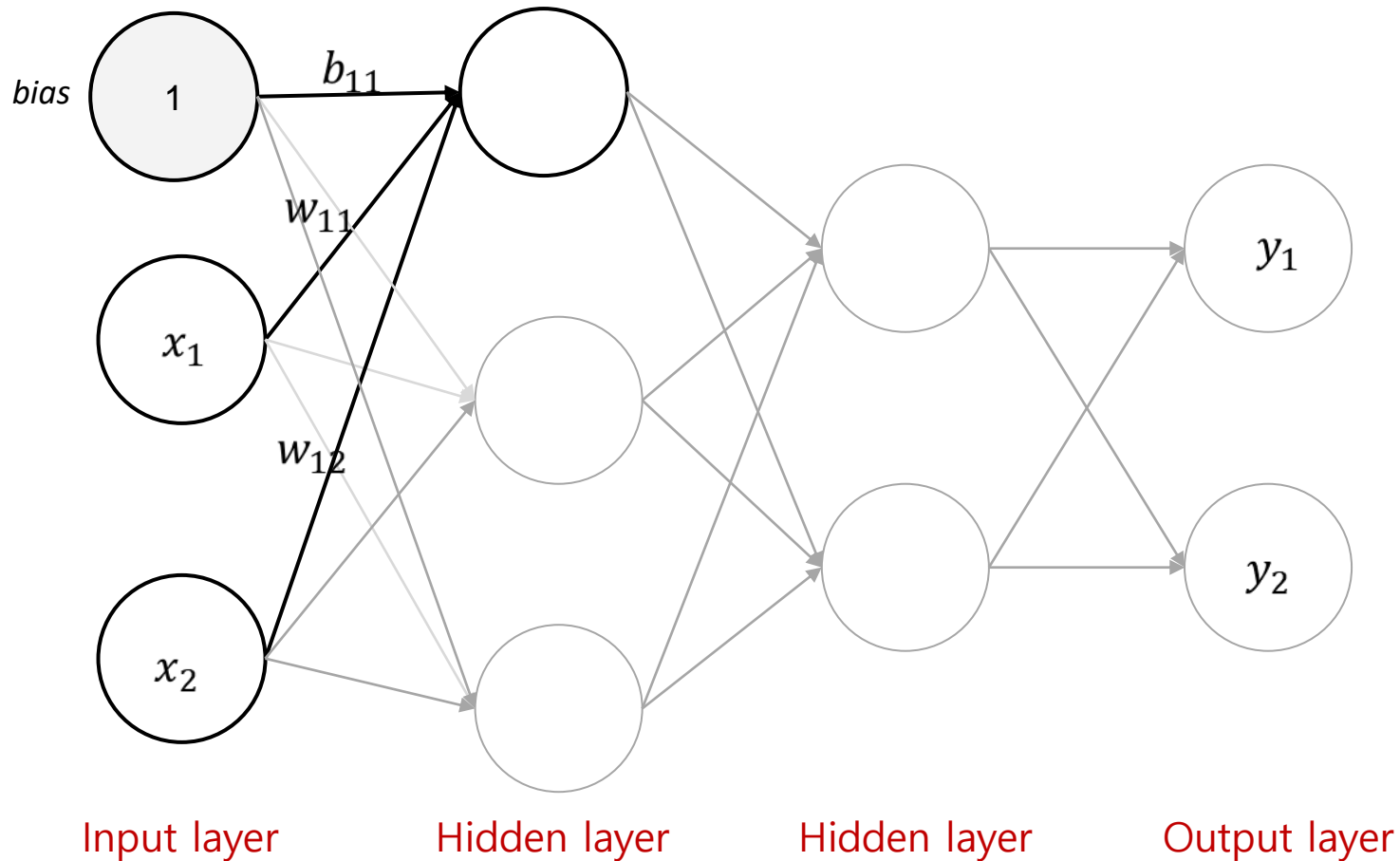
---

3층 신경망 : 입력층(0층 2), 은닉층(1층 3, 2층 2), 출력층(3층 2)



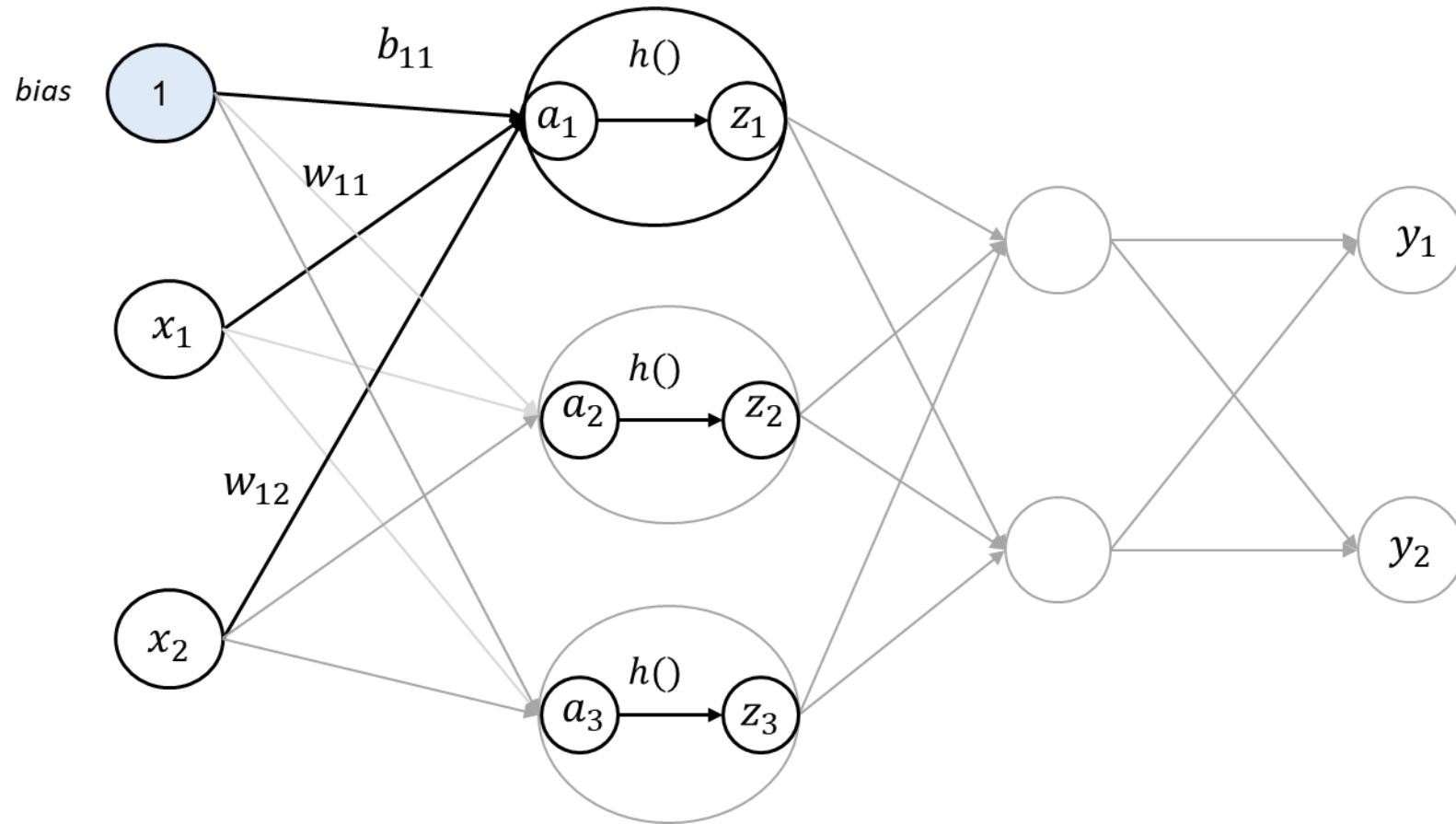
# Neural Network

입력층에서 1층으로 신호 전달



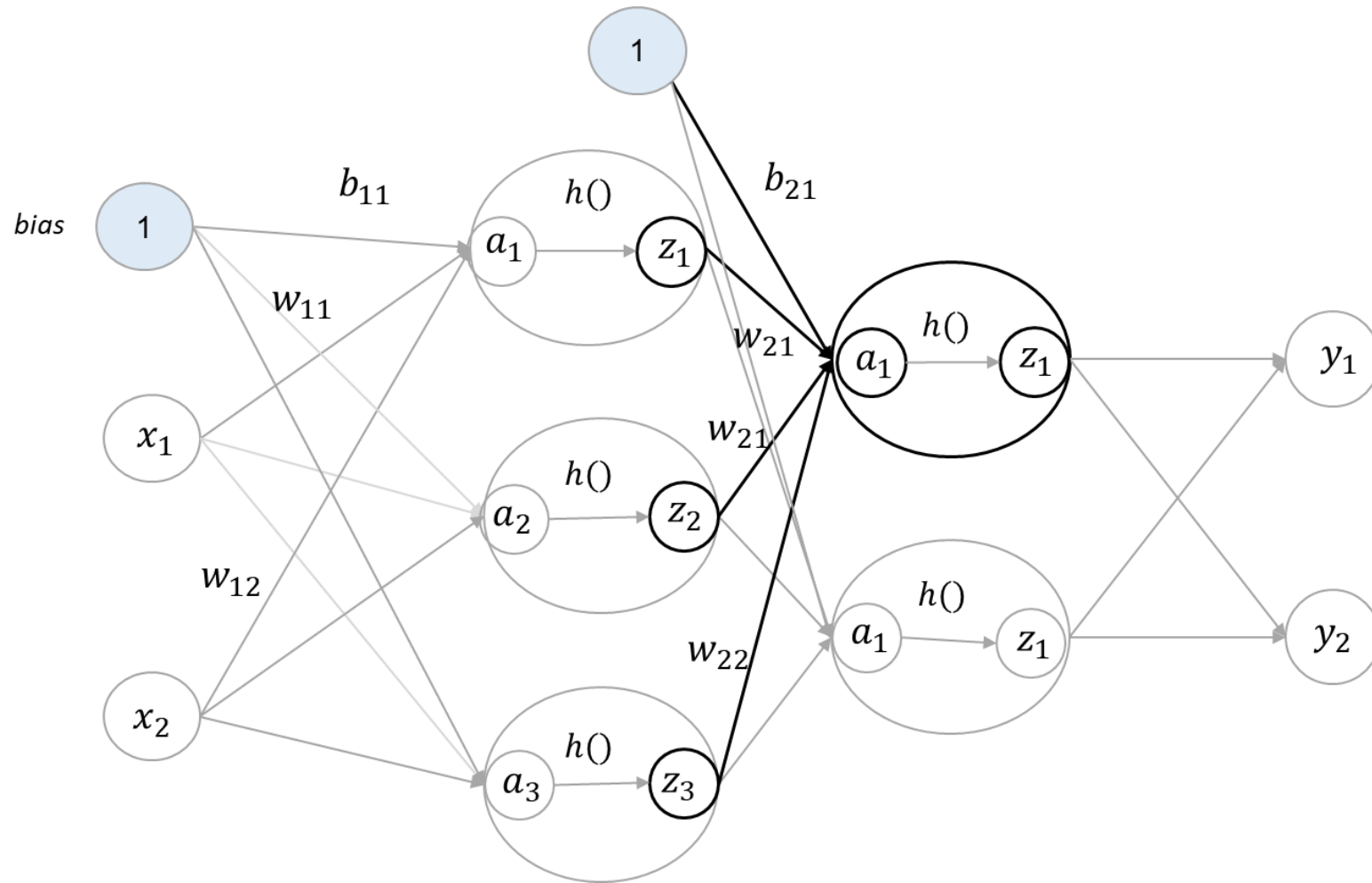
# Neural Network

입력층에서 1층으로 신호 전달



# Neural Network

2층에서 3층(출력층)으로 신호 전달

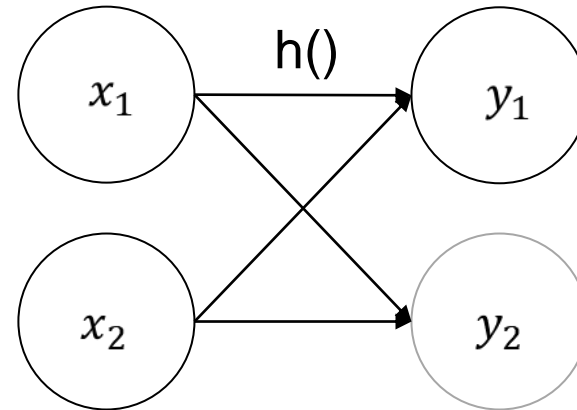
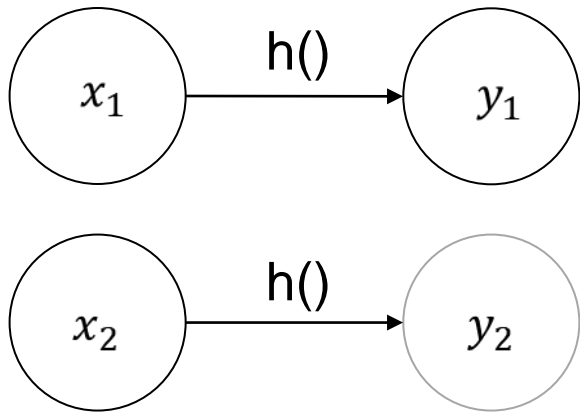


# Neural Network

---

## 출력층 설계

항등 함수(Identity Function)는 회귀에서 입력을 그대로 출력  
소프트맥스 함수(Softmax Function) 분류에서 확률값 으로 출력





# Neural Network

---

## Softmax Function

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

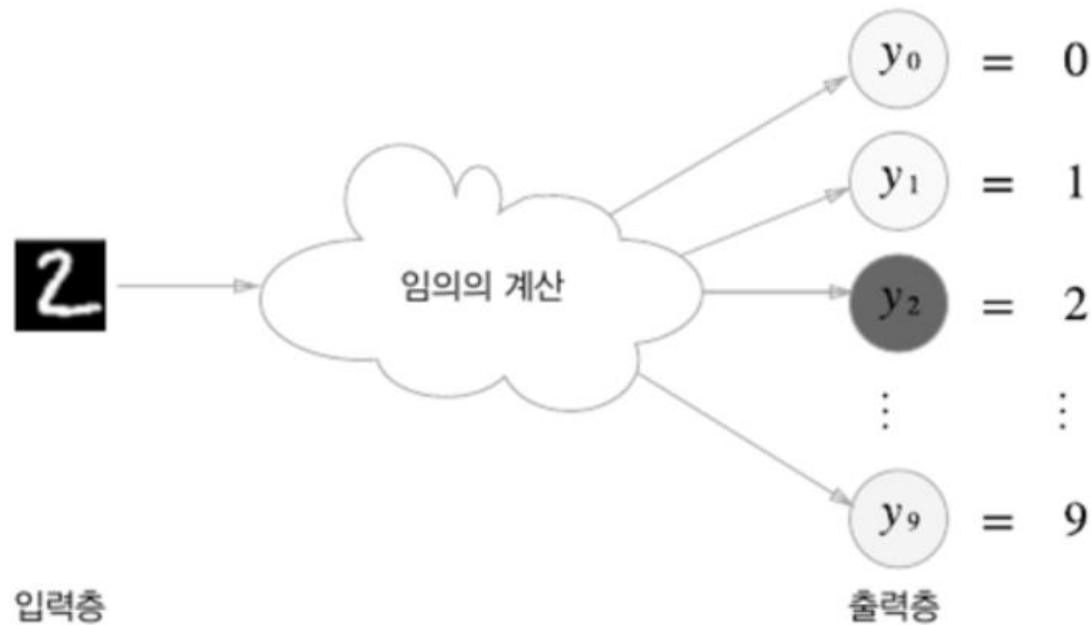
- Softmax는 입력받은 값을 출력으로 0~1사이의 값으로 모두 정규화하며 출력 값들의 총합은 항상 1이 되는 특성을 가진 함수이다
- 주로 마지막 Hidden-Output 층에 쓰인다
- 보통 타겟 값은 예측(Prediction)과 분류(Classification)으로 나뉘는데 분류의 성질을 가진 값의 활성화 함수로 주로 쓰인다.

# Neural Network

## Softmax Function

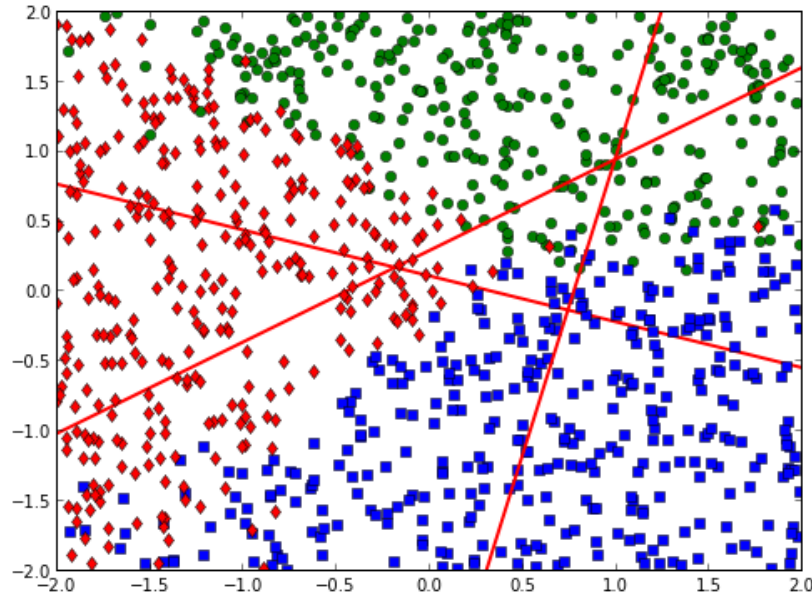
$$y_k = \frac{e^{a_k}}{\sum_{i=1}^n e^{a_i}}$$

$$\text{softmax}(z) = \left[ \frac{e^{z_1}}{e^{z_1} + e^{z_2} + e^{z_3}}, \frac{e^{z_2}}{e^{z_1} + e^{z_2} + e^{z_3}}, \frac{e^{z_3}}{e^{z_1} + e^{z_2} + e^{z_3}} \right] = [p_1, p_2, p_3]$$



# Neural Network

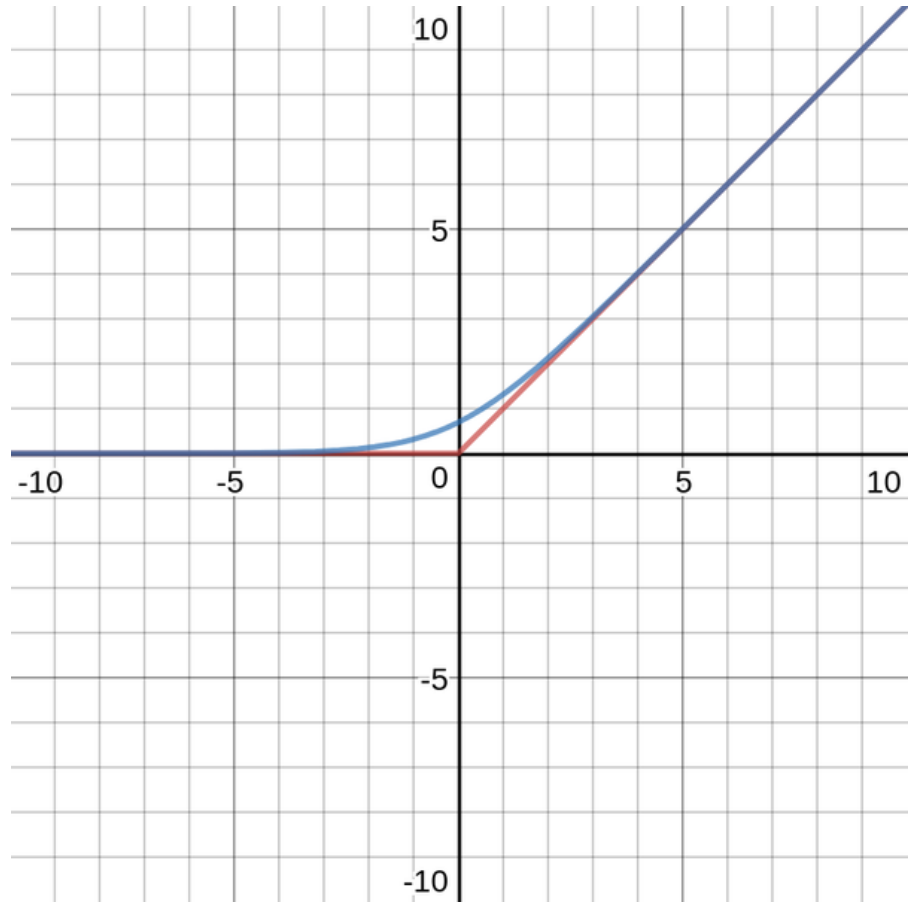
## Softmax Function



$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \text{softmax} \left( \begin{bmatrix} W_{1,1}x_1 + W_{1,2}x_2 + W_{1,3}x_3 + b_1 \\ W_{2,1}x_1 + W_{2,2}x_2 + W_{2,3}x_3 + b_2 \\ W_{3,1}x_1 + W_{3,2}x_2 + W_{3,3}x_3 + b_3 \end{bmatrix} \right)$$

# Neural Network

## Softmax Function



—  $\max(0, x)$  function  
—  $\text{softmax}(0, x)$  function

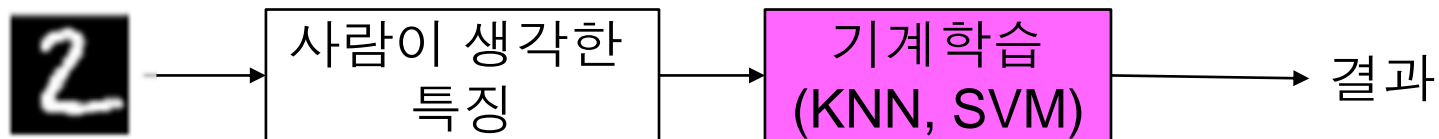
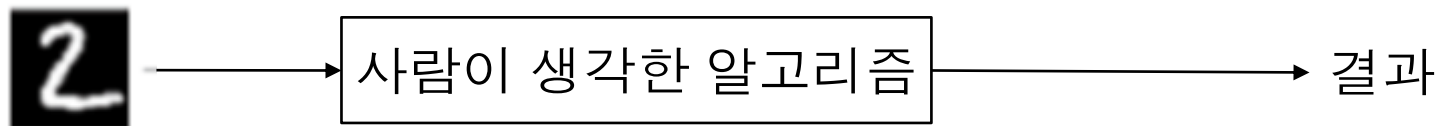
- Max의 근사값
- 0 근처에서 smooth curve 사용
- 미분 가능한 값으로 변경


# Neural Network Fit

## 신경망 학습

학습이란 훈련 데이터로부터 가중치 매개변수의 최적값을 자동으로 획득  
신경망이 학습하기 위해서는 지표가 필요 이를 손실함수라 함  
손실함수의 결과값을 가장 작게 만드는 가중치 매개변수를 찾는 것이 학습의 목표

딥러닝은 종단간 기계학습(end-to-end machine learning)



 는 사람이 개입하지 않음

# Neural Network Fit

---

## 손실 함수(Loss Function)

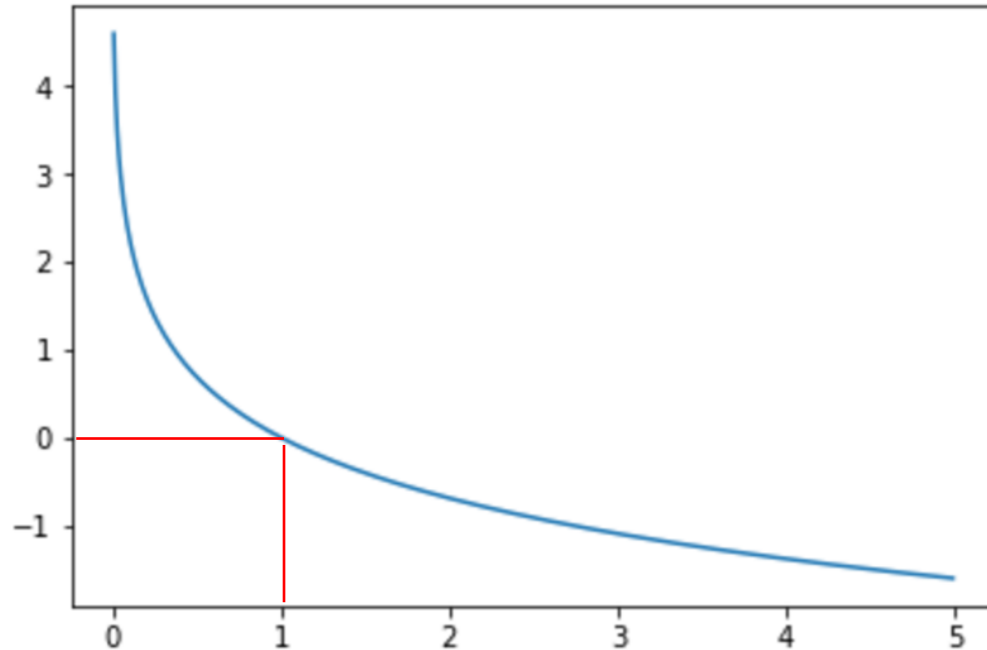
- 평균 제곱 오차(Mean Squared Error : MSE)
  - ✓ 계산이 간편하여 가장 많이 쓰이는 손실 함수  
( y : 신경망 출력 값, t : Target value)
- 교차 엔트로피 오차(Cross Entropy Error : CEE)
  - ✓ 기본적으로 분류(Classification) 문제에서 원-핫 인코딩(one-hot encoding)했을 경우에만 사용할 수 있는 loss function
  - ✓ 주로 SoftMax를 통해 활성화 된 Output 값과 함께 주로 쓰인다.

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

$$E = - \sum_k t_k \log y_k$$

# Neural Network Fit

교차 엔트로피 오차(Cross Entropy Error : CEE)



- $y == 1, E == 0$
- $y$ 가 1일 때(정확히 예측했을 때) cost 값은 0
- 그리고 0에 가까울 때(틀렸을 때) 큰 양수의 값

$$E = -\sum_k t_k \log y_k$$

$t_k$  : 정답 레이블  
 $y_k$  : 신경망의 출력

# Neural Network Fit

---

매개변수 최적화를 위해 정확도가 아닌 손실 함수를 사용하는 이유

- ✓ 학습의 궁극적 목적 = 정확도를 최대화 하는 매개변수( $W, b$ ) 값을 찾는 것
- ✓ 그런데 왜 정확도라는 지표를 사용하지 않고 손실함수의 값을 거치는 것일까?
- ✓ 이유는 신경망학습에서의 미분의 역할에 주목
- ✓ 매개변수의 손실함수 미분이란 가중치 매개변수의 값을 아주 조금 변화시켰을 경우 손실함수가 어떻게 변하느냐에 대한 의미

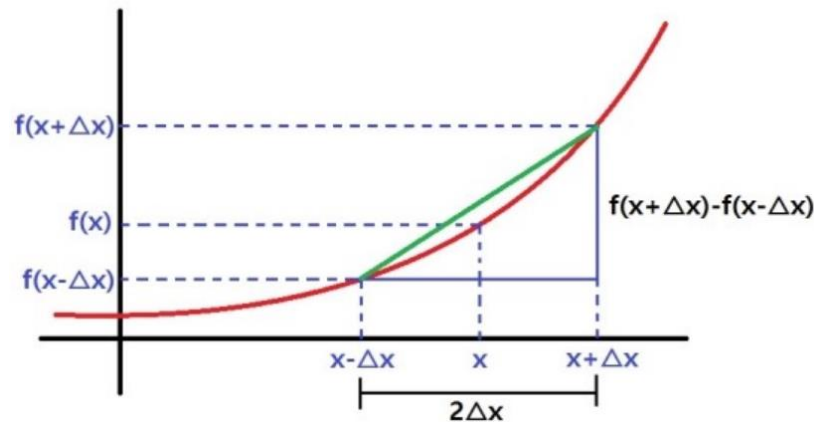


# Neural Network Fit

수치미분 - 아주 작은 차분으로 미분을 구하는 방법

$$\lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h} \quad \leftarrow \text{함수의 미분} \quad 1e-50$$

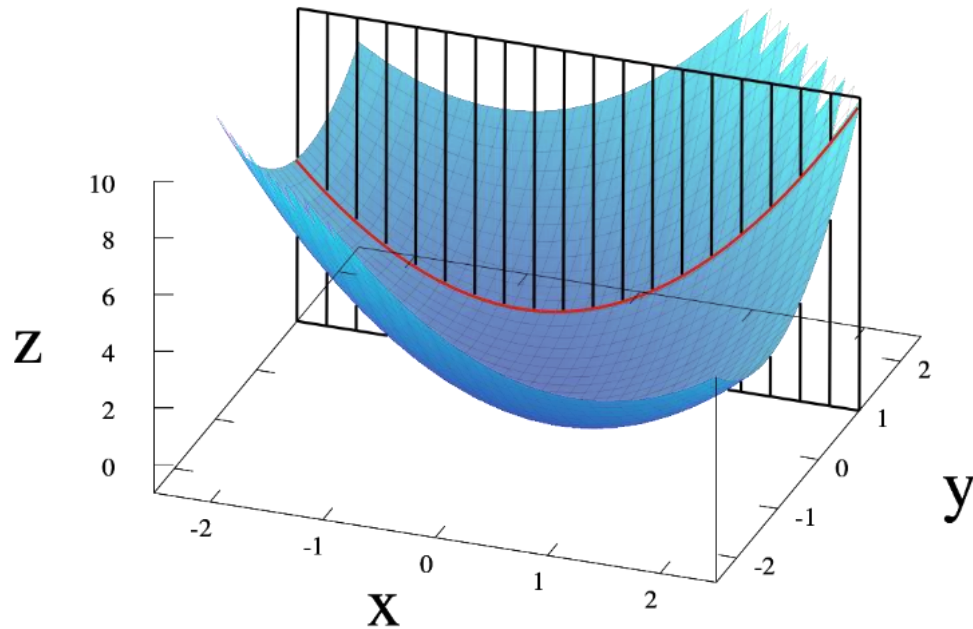
$$f'(x) \approx \frac{f(x+\Delta x) - f(x-\Delta x)}{2 \Delta x} \quad \leftarrow \text{수치 미분} \quad 1e-4$$



# Neural Network Fit

편미분(Partial differential)

$$z = x^2 + y^2$$

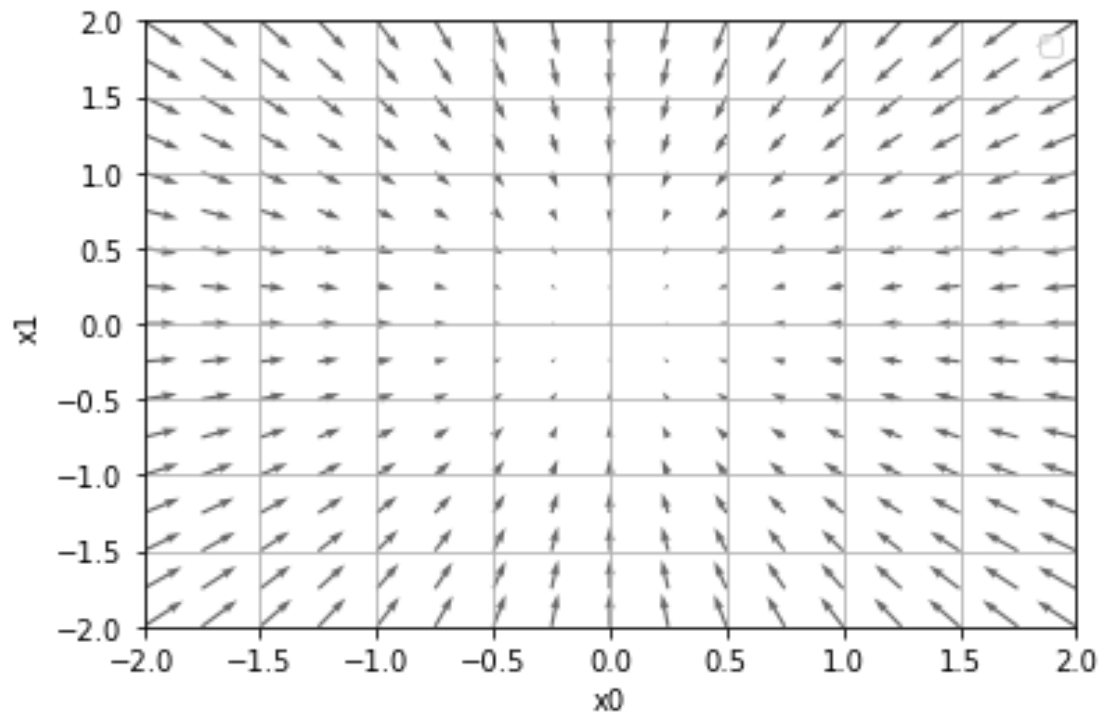


편미분은 변수가 하나인 미분과 마찬가지로 특정 장소의 기울기를 구함  
여러 변수 중 목표 변수 하나에 초점을 맞추고 다른 변수는 값을 고정  
수치 미분 함수를 적용하여 편미분을 구함

# Neural Network Fit

## 기울기(gradient)

모든 변수의 편미분값을 벡터로 정리한 것을 기울기로 표현

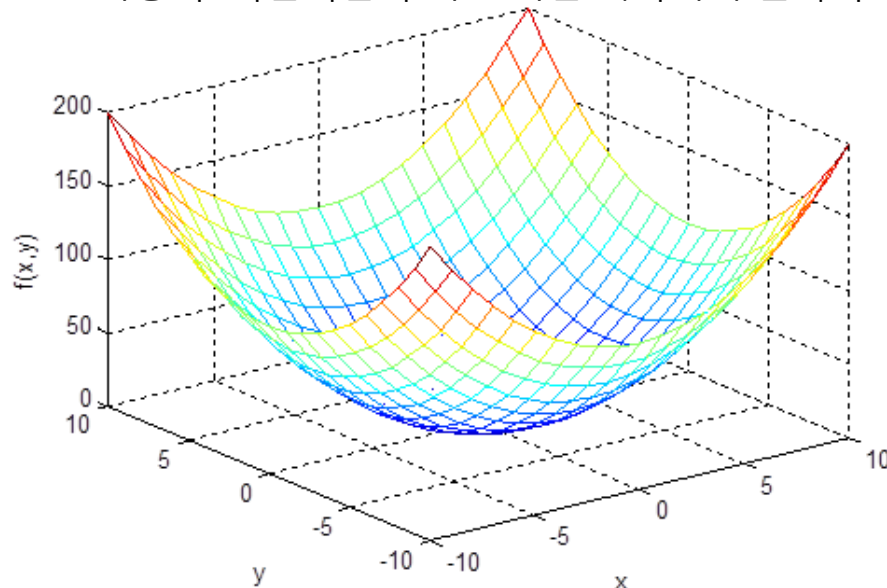


# Neural Network Fit

## 경사하강법(Gradient Descent)

최적의 매개변수를 찾기 위해 손실함수의 최솟값을 찾아가는 과정(학습)  
손실함수의 최솟값을 찾기 위해서는 손실함수의 복잡성에 기인하여 어려움이 존재  
기울기(경사)를 이용해 함수의 최솟값을 찾는 것이 경사하강법  
각 지점에서 함수의 값을 낮추는 방안을 제시하는 지표가 기울기  
복잡한 함수에서는 기울기가 가리키는 방향에 최솟값이 없는 경우가 대부분  
그러나 그 방향으로 가야 함수의 값을 줄일 수 있음  
기울기 정보가 유일한 단서임

경사하강법은 현 위치에서 기울어진 방향으로 일정거리만큼 이동  
이동 후 기울기를 구하고 이를 계속하여 반복 수행



$$\nabla f = \left( \frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \dots, \frac{\partial f}{\partial x_n} \right)$$

# Neural Network Fit

---

## 미니배치(Mini Batch)

모든 훈련 데이터를 대상으로 손실함수의 값을 구해야 함

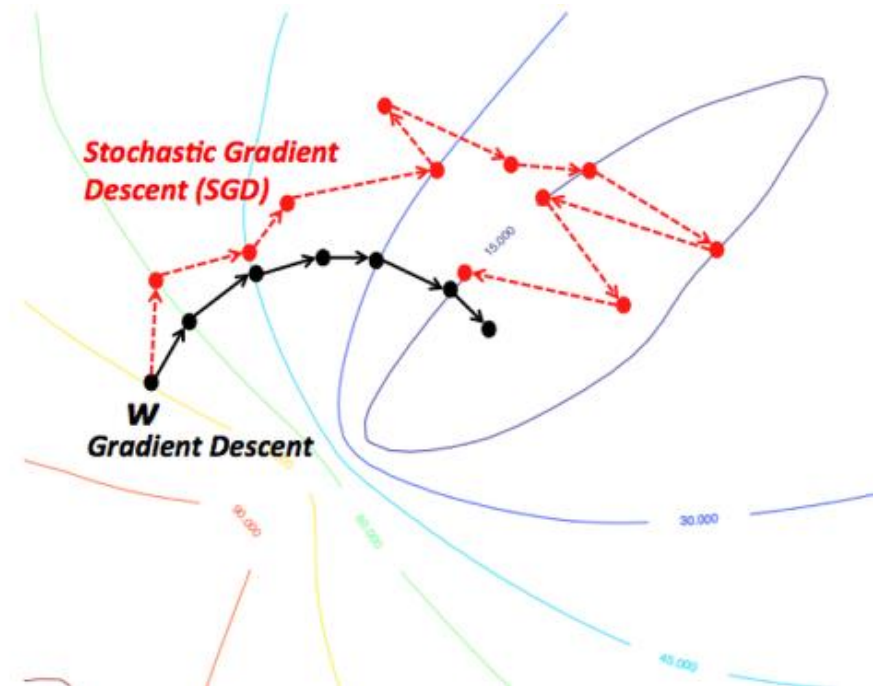
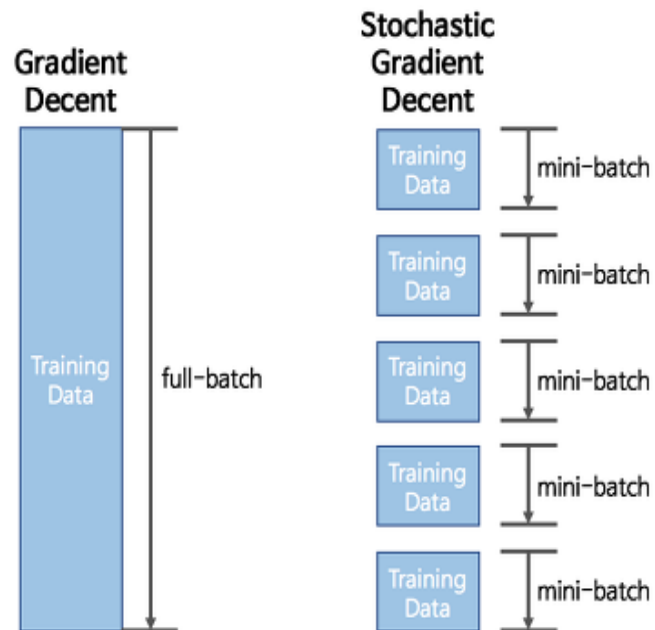
훈련 데이터 60000개에 대하여 훈련을 한다면 60000개의 손실함수 값을 구하여 평균을 구해야 통일된 지표를 얻을 수 있음

훈련 데이터 셋이 크다면 일부분만 추려 전체의 근사치로 이용 할 수 있음

이러한 학습 방법을 미니배치라고 함

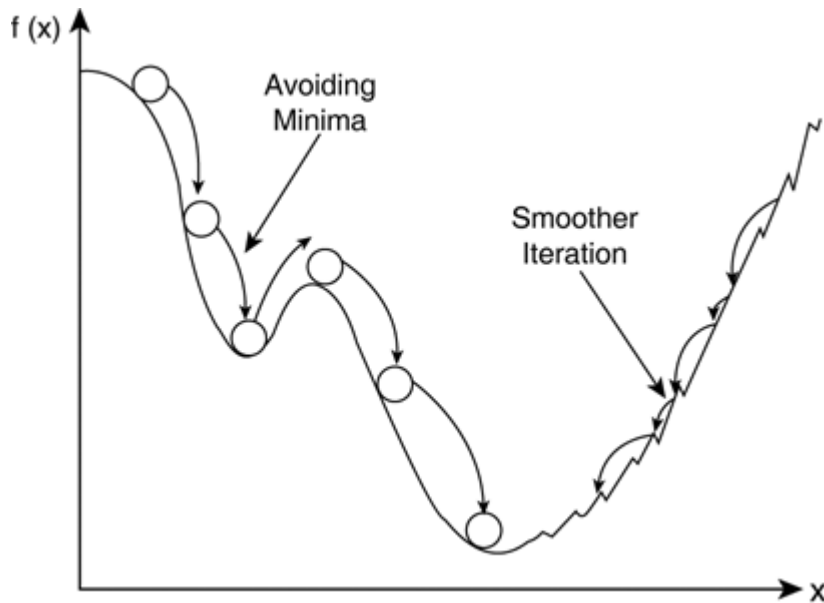
# Neural Network Fit

## Stochastic Gradient Descent(확률적 경사하강법)



# Neural Network Fit

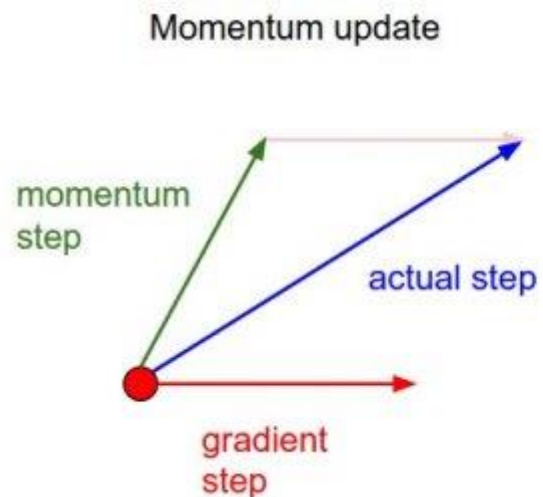
## Momentum



- $\alpha$  : Momentum term(얼마나 운동량을 더할 것인가)
- 과거에 얼마나 이동했는지에 대한 이동항  $v$ 를 기억하고 새로운 이동항을 구할 때 과거에 이동했던 관성만큼 곱해준 후 **gradient** 이동
- **Gradient**들의 지수평균을 이용하여 이동
- 이론적으로 “local minima” 를 피할 수 있다

# Neural Network Fit

## NAG(Nesterov Accelerator Gradient)



$$v = \alpha v - \eta \frac{\partial L}{\partial W}$$

$$W = W + v$$



$$v \leftarrow \alpha v - \eta \left( \frac{\partial L}{\partial W} + \alpha v \right)$$

$$W \leftarrow W + v$$



# Neural Network Fit

---

## AdaGrad

- $\mu$ (learning rate) : 너무 작으면 학습시간 길고 **overfitting** 가능성 높아짐, 너무 크면 발산하여 학습 안됨
- learning rate decay : 학습을 진행하면서 학습률을 점차 줄여가는 방법
- AdaGrad : 변수들을 **update**할 때 각각의 변수마다 **step size** 를 다르게 설정하는 방식  
“지금까지 많이 변화하지 않은 변수들은 **step size**를 크게 하고, 많이 변화한 변수들은 작게하자 ”

$$h \leftarrow h + \frac{\partial L^2}{\partial W}$$

$$W \leftarrow W + \eta \frac{1}{\sqrt{h}} \frac{\partial L}{\partial W}$$

# Neural Network Fit

## 신경망에서의 기울기

$$j \begin{matrix} & i \\ \begin{bmatrix} \theta_{ij} & \theta_{21} & \theta_{31} & \dots & \theta_{101} \\ \theta_{12} & \theta_{22} & \theta_{32} & \dots & \theta_{102} \\ \theta_{13} & \theta_{23} & \theta_{33} & \dots & \theta_{103} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \theta_{1\ 3072} & \theta_{2\ 3072} & \theta_{3\ 3072} & & \theta_{10\ 3072} \end{bmatrix} \end{matrix} \partial \theta_{ij}$$

$$\theta_{ij} = \theta_{ij} - \alpha \frac{\partial CE}{\partial \theta_{ij}}$$

ij 행렬을 W라 했을  
경우 가중치 행렬 W,  
손실함수 CE(Cross  
Entropy)의 편미분  
행렬을 조절하여  
손실함수의 최소값  
방향으로 업데이트

# Neural Network Fit

---

## Neural Network 학습

1. 가중치 행렬 Network 구축 (init network)
2. 손실함수 정의
3. 미니배치
4. 기울기 산출
5. 매개변수 갱신
6. 1~5 반복

# Neural Network Fit

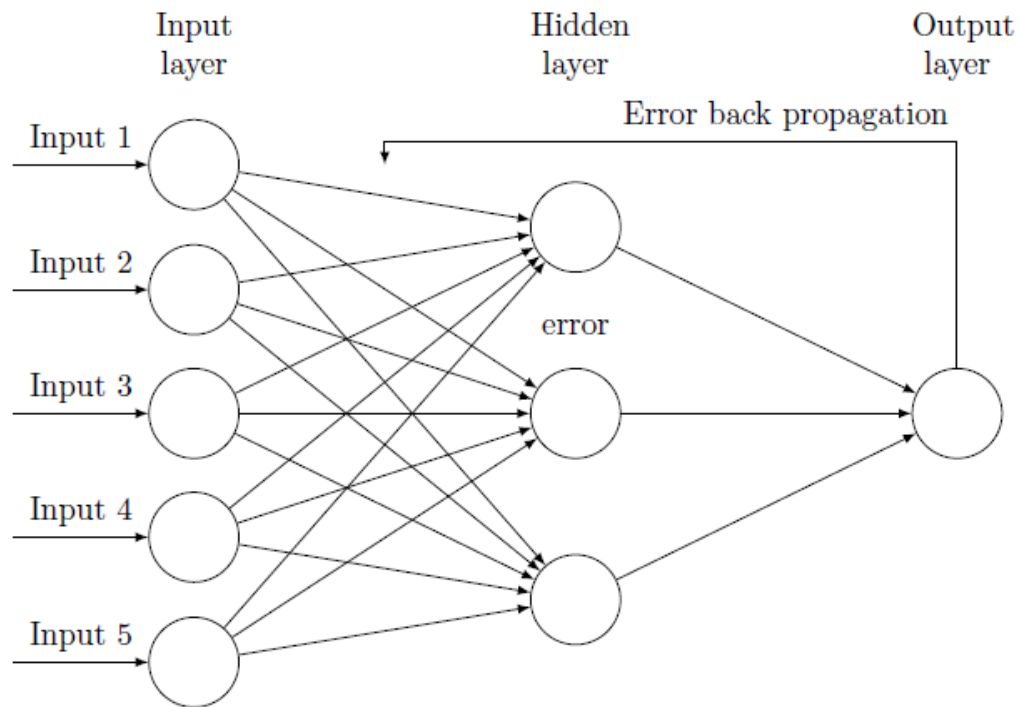
---

## Neural Network 학습(구현 목록)

1. 초기화 수행
2. 입력층, 은닉층, 출력층 구현
3. 예측함수 구현
4. 손실함수구현
5. 정확도 지표 함수 구현
6. 기울기 함수 구현
7. 미니배치 학습

# Back Propagation

Back Propagation(역전파)



*Geoffrey Hinton*

# Back Propagation

---

## 오차 역전파의 이해

- 신경망 학습 : 신경망을 학습하기 이해서는 가중치 매개변수의 기울기(가중치 매개변수에 대한 손실 함수의 기울기)를 최소화 하는 방법으로 수치 미분을 사용
- 수치미분의 장점 : 단순하기 때문에 이해와 구현이 쉬움
- 수치미분의 단점 : 계산 시간이 오래 걸림
- 오차역전파 : 가중치 매개변수의 기울기를 효율적으로 계산

## 오차 역전파의 이해를 위한 방법

- 수식을 통한 이해
- 계산 그래프(Computation Graph)를 통한 이해

※ Andrej Karphathy와 Fei-Fei Li 교수가 제안

# Back Propagation

---

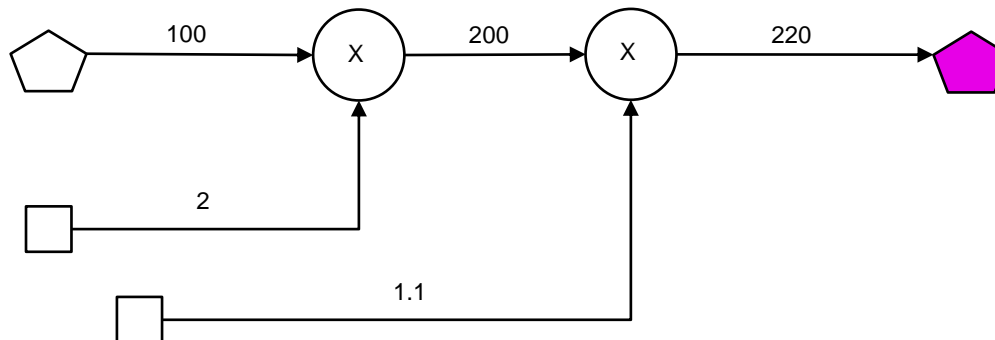
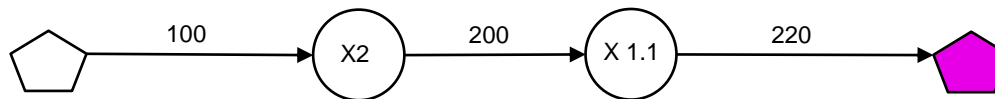
## 수식을 통한 이해

- 신경망 학습 : 신경망을 학습하기 이해서는 가중치 매개변수의 기울기(가중치 매개변수에 대한 손실 함수의 기울기)를 최소화 하는 방법으로 수치 미분을 사용
- 수치미분의 장점 : 단순하기 때문에 이해와 구현이 쉬움
- 수치미분의 단점 : 계산 시간이 오래 걸림
- 오차역전파 : 가중치 매개변수의 기울기를 효율적으로 계산

# Back Propagation

## Computational Graph(계산그래프)

- 계산 과정을 그래프로 나타낸 것
- 그래프는 복수의 노드와 에지로 표현
- 계산 그래프의 특징은 국소적 계산을 전파함으로 최종 결과를 획득
- 국소적 계산은 자신과 관련된 작은 범위만 계산

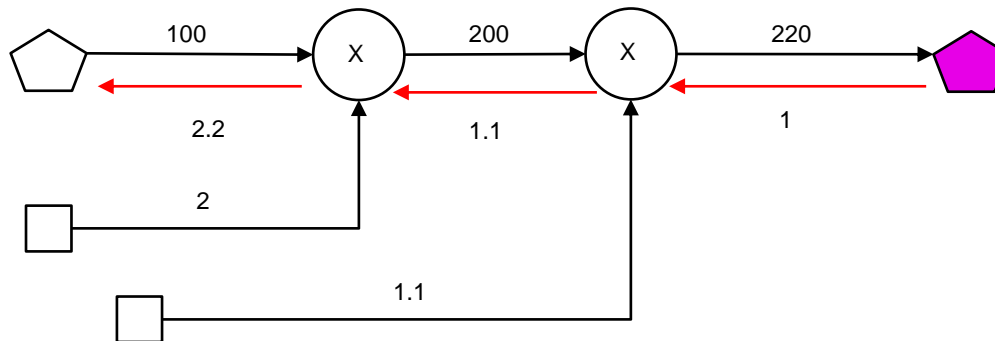




# Back Propagation

## 계산그래프를 사용하는 이유

- 국소적 계산을 함으로 전체가 아무리 복잡해도 각 노드에서는 단순한 계산에 집중하여 문제를 단순화
- 계산 그래프는 중간 계산 결과를 모두 보관 할 수 있음
- 역전파를 통해 미분을 효율적으로 계산할 수 있음



# Back Propagation

## 연쇄법칙(chain rule)

- 합성함수의 연쇄법칙 : 합성함수의 미분은 합성함수를 구성하는 각 함수의 곱으로 나타낼 수 있음

### Chain Rule

If  $f$  and  $g$  are both differentiable and  $F(x)$  is the composite function defined by  $F(x) = f(g(x))$  then  $F$  is differentiable and  $F'$  is given by the product

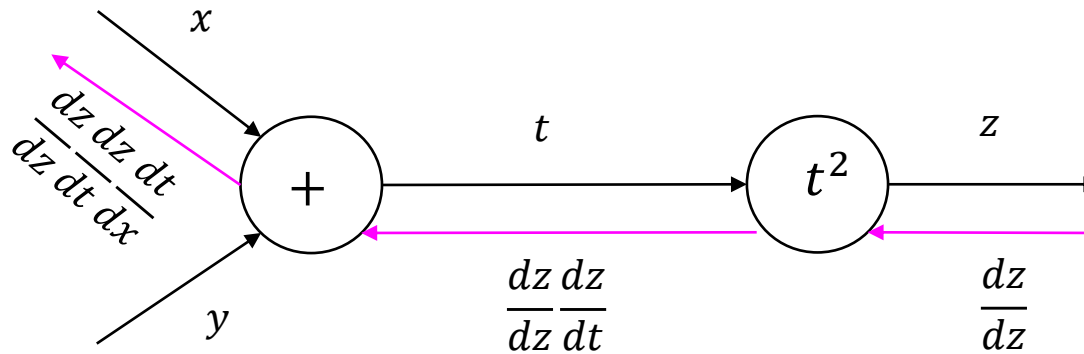
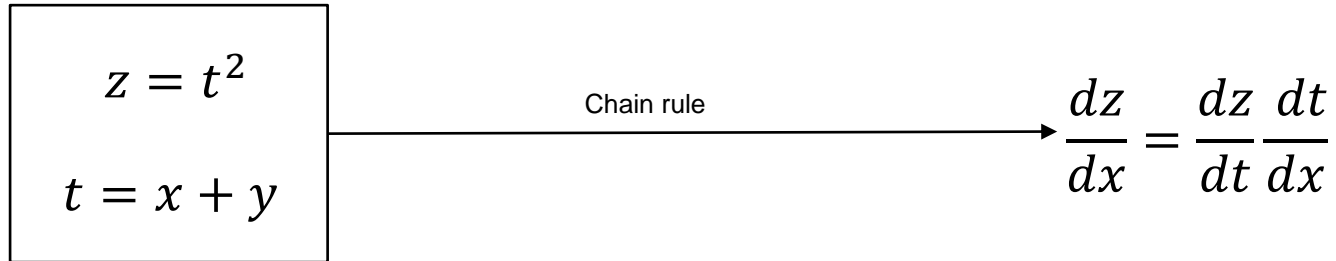
$$F'(x) = f'(g(x)) g'(x)$$

Differentiate  
outer function

Differentiate  
inner function

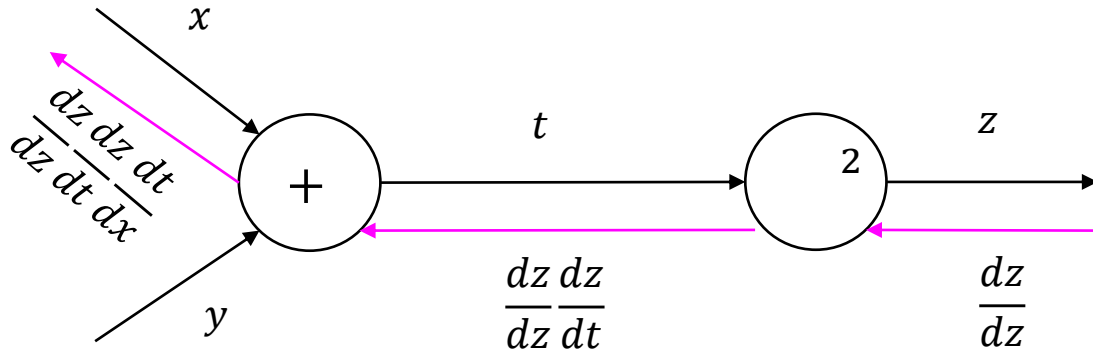
# Back Propagation

연쇄법칙과 계산 그래프



# Back Propagation

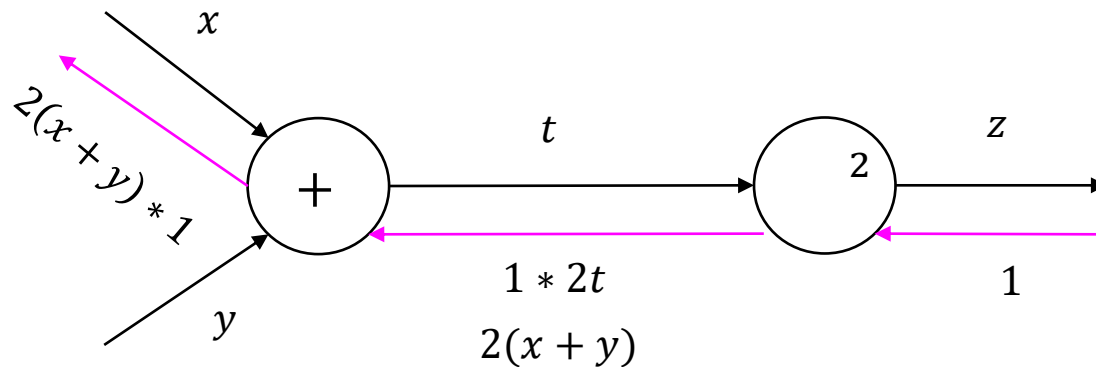
연쇄법칙과 계산 그래프



$$\frac{dz}{dz} = 1$$

$$\frac{dt}{dz} = 2t$$

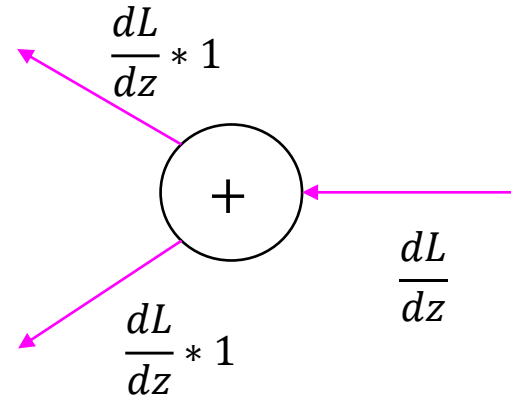
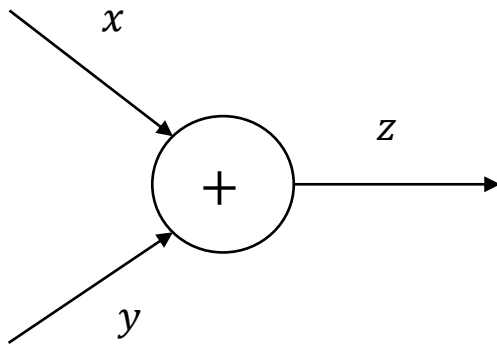
$$\frac{dt}{dx} = 1$$



# Back Propagation

## 덧셈 노드 역전파

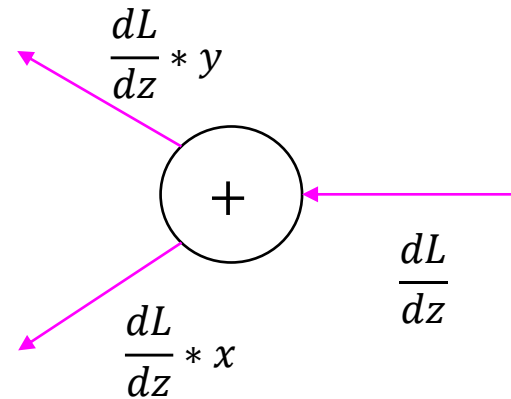
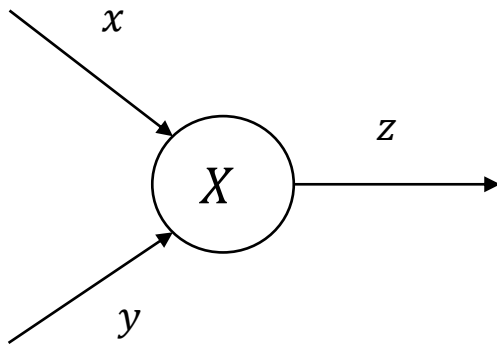
- 덧셈의 역전파는 상류의 값을 하류의 노드로 그대로 전달



# Back Propagation

## 곱셈 노드 역전파

- 곱셈의 역전파는 상류의 값에 순전파 때의 입력 신호들을 서로 바꾼 값을 곱해서 하류로 전달



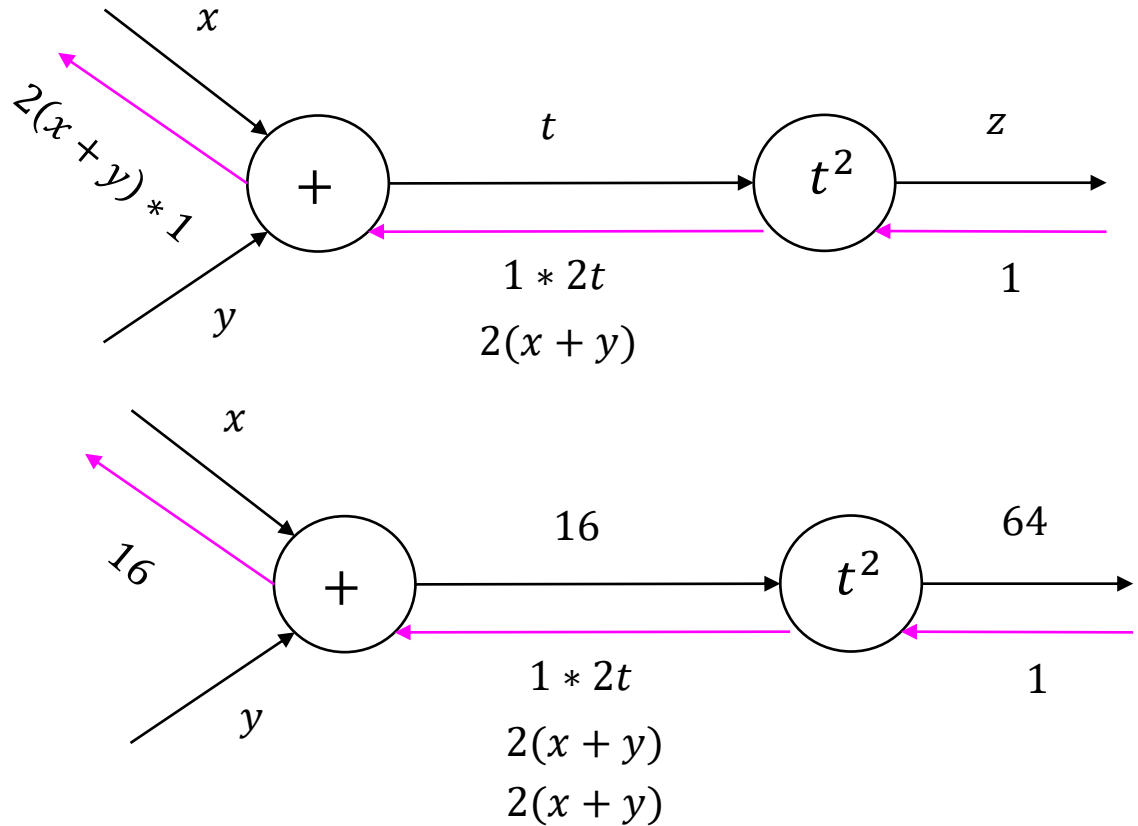
# Back Propagation

계산 그래프를 이용한 역전파 곱셈 계산

$$z = t^2$$
$$t = x + y$$

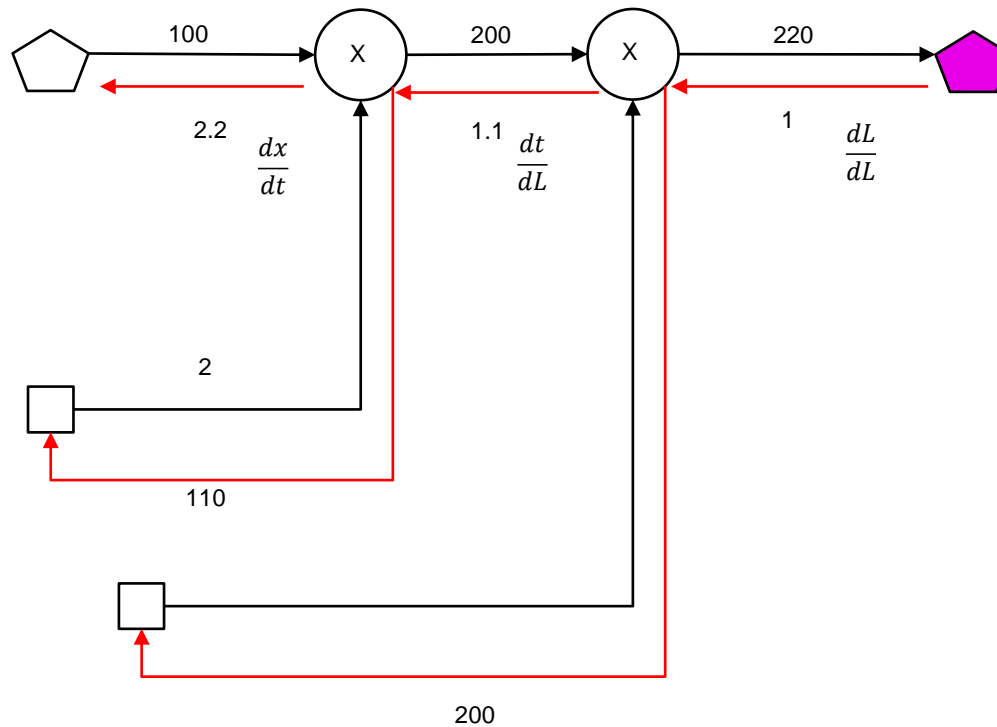
$$x = 3$$

$$y = 5$$



# Back Propagation

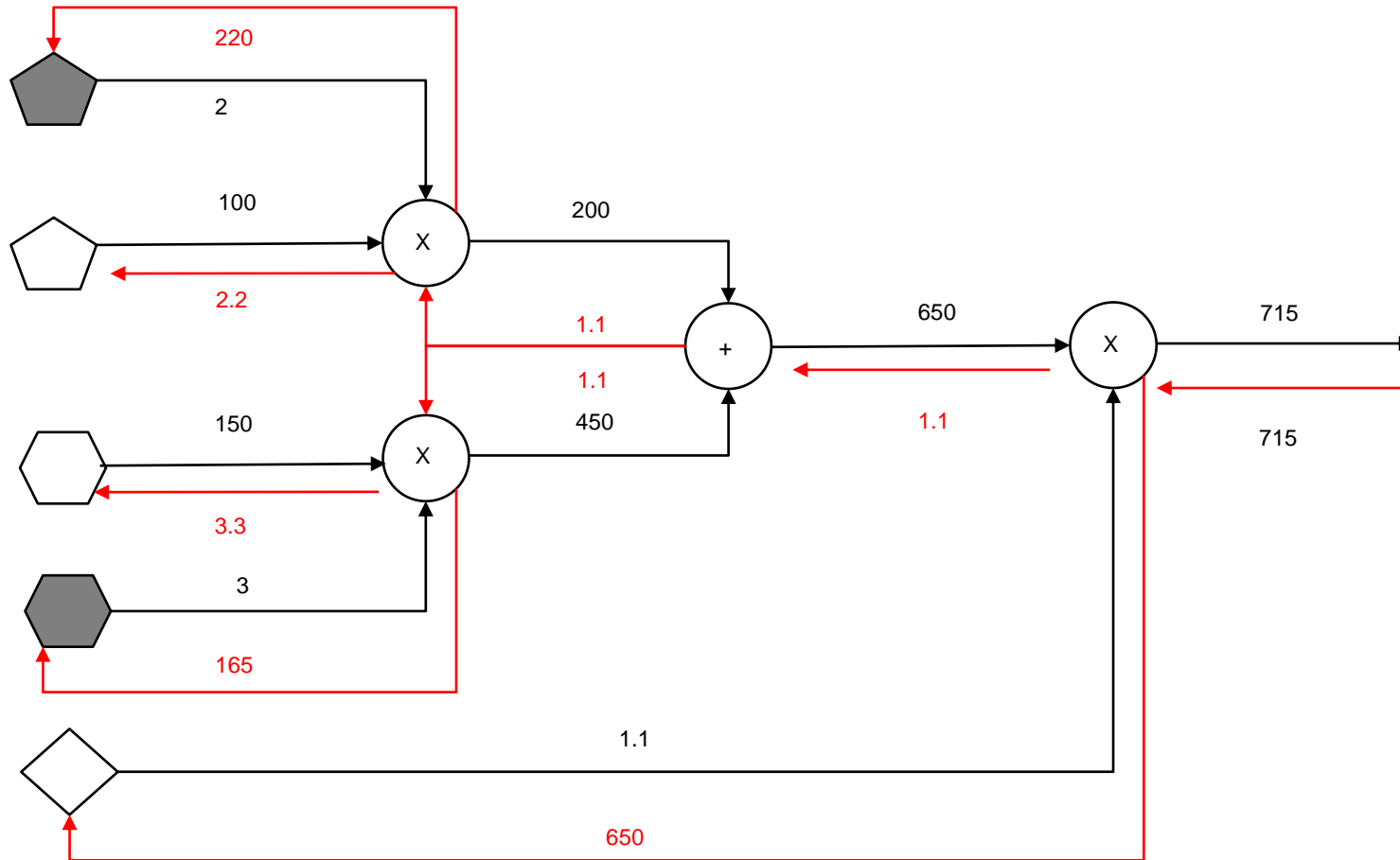
계산 그래프를 이용한 역전파 계산





# Back Propagation

계산 그래프를 이용한 역전파 계산

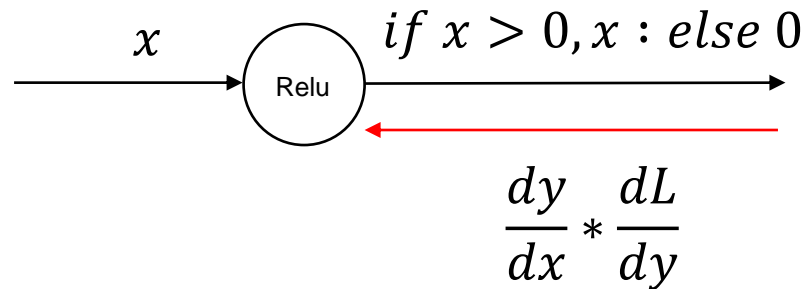


# Back Propagation

Relu계층 역전파

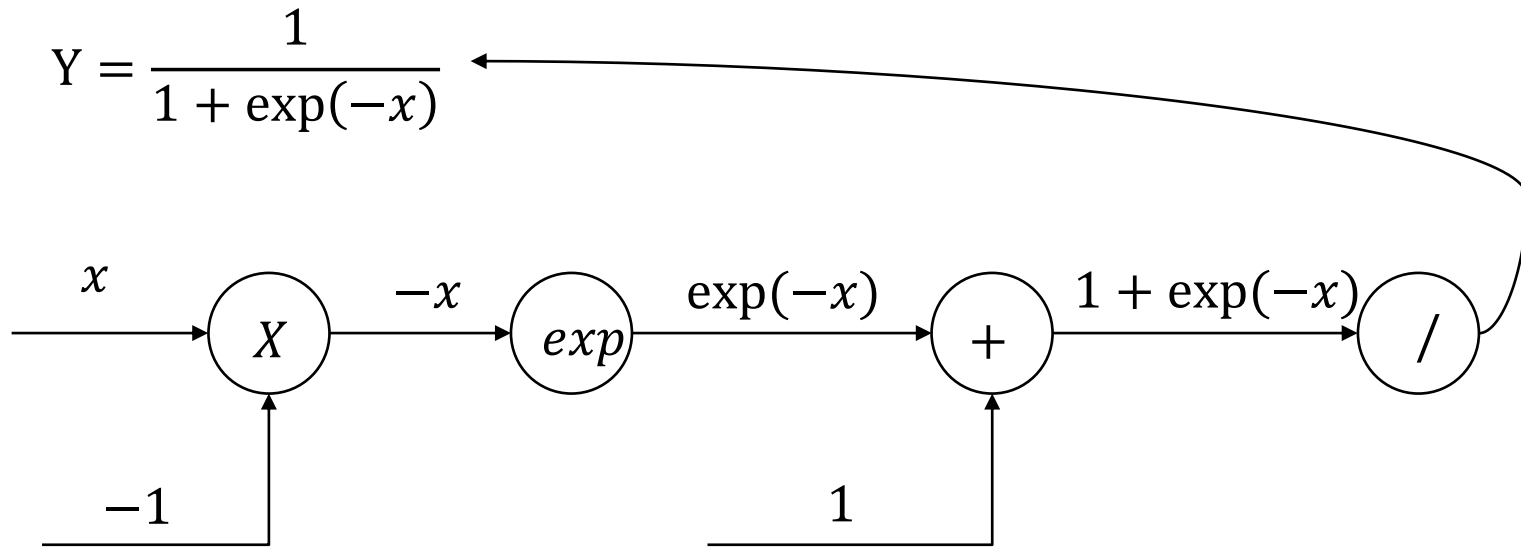
$$Y = \begin{cases} X & (X > 0) \\ 0 & (X \leq 0) \end{cases}$$

$$\frac{dy}{dx} = \begin{cases} 1 & (X > 0) \\ 0 & (X \leq 0) \end{cases}$$



# Back Propagation

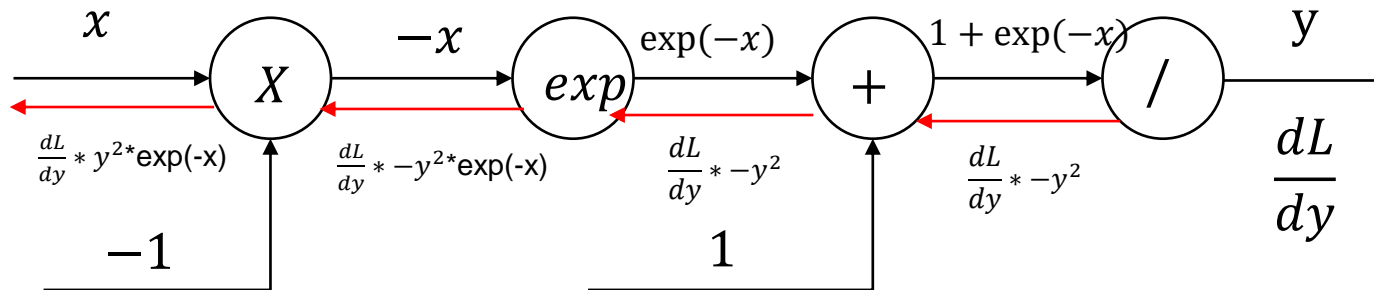
Sigmoid 계층 순전파



# Back Propagation

Sigmoid 계층 역전파

$$y = \frac{1}{x} \quad \frac{dy}{dx} = \frac{-1}{(x^2)}$$



$$\frac{dL}{dy} * y^2 * \exp(-x) = \frac{dL}{dy} * \frac{1}{(1 + \exp(-x))^2} * \exp(-x)$$

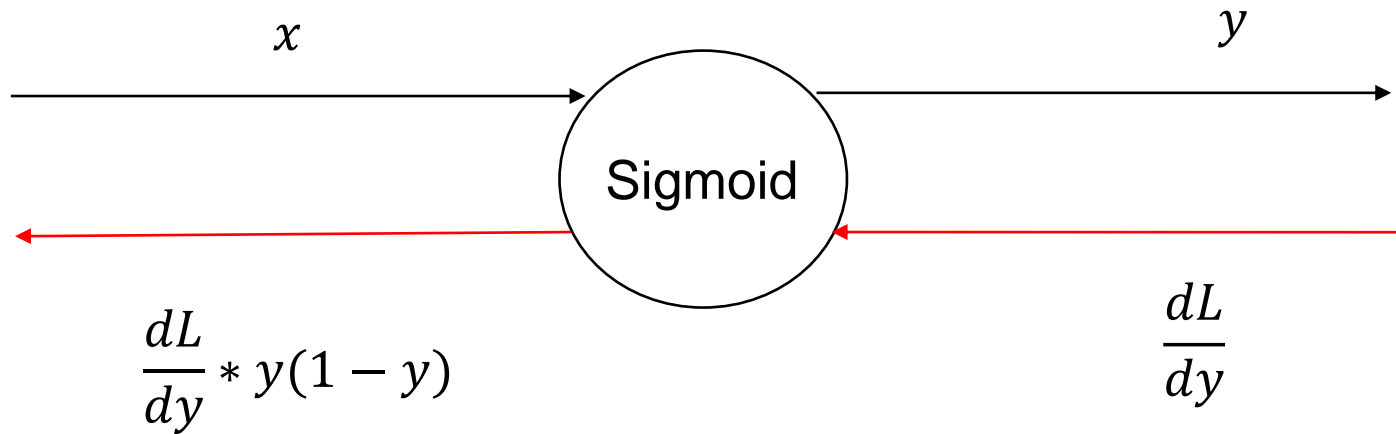
$$\frac{dL}{dy} * y^2 * \exp(-x) = \frac{dL}{dy} * \frac{1}{1 + \exp(-x)} * \frac{\exp(-x)}{1 + \exp(-x)}$$

$$\frac{dL}{dy} * y(1 - y)$$

# Back Propagation

---

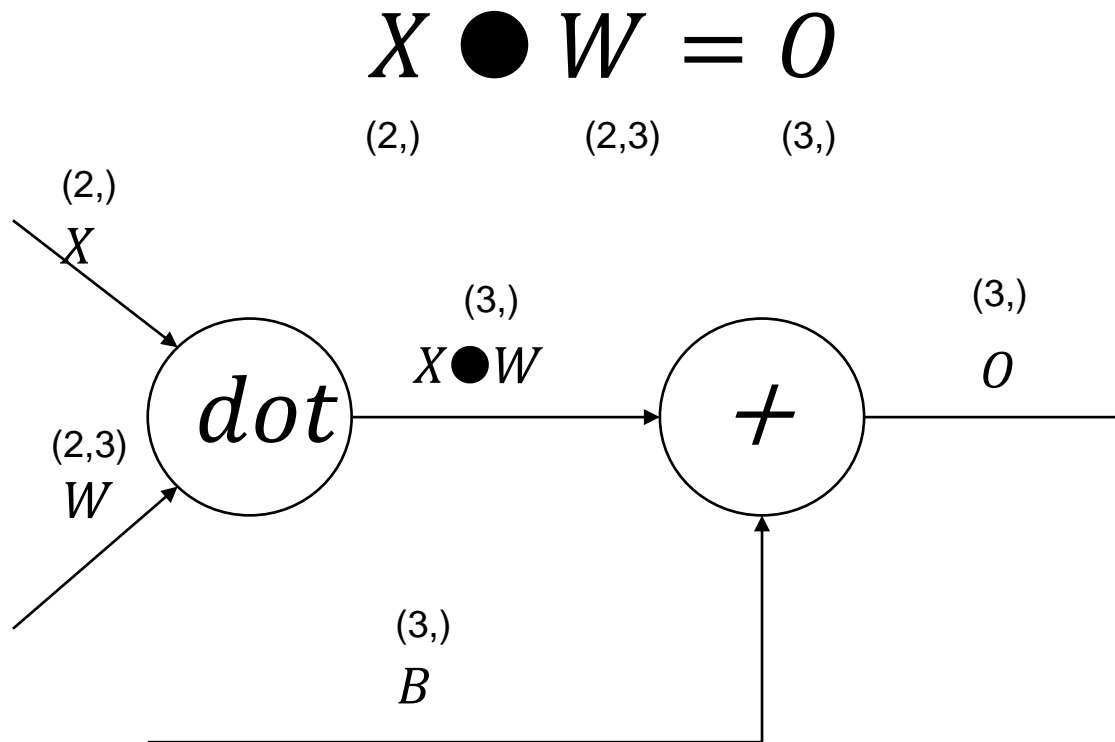
Sigmoid 계층 역전파



# Back Propagation

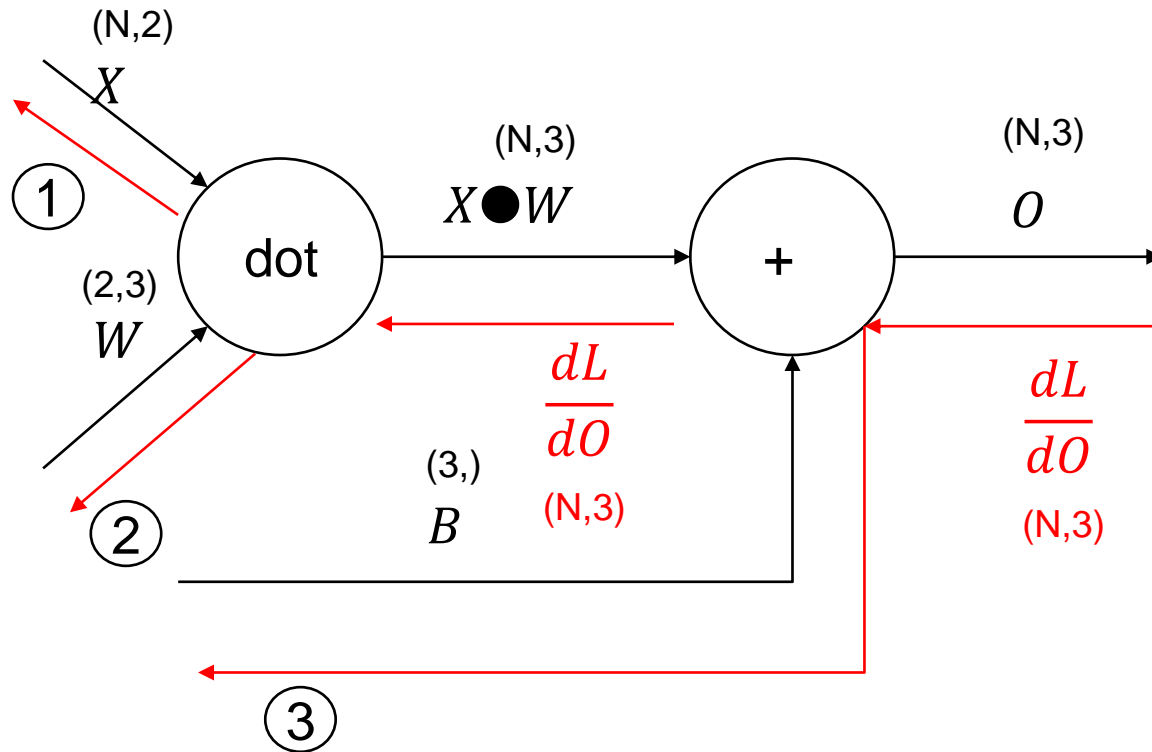
## Affine 계층 역전파

- Affine 변환 : 신경망의 순전파시 수행하는 행렬의 내적의 기하학적 표현 (Affine transformation)



# Back Propagation

Affine 계층 역전파



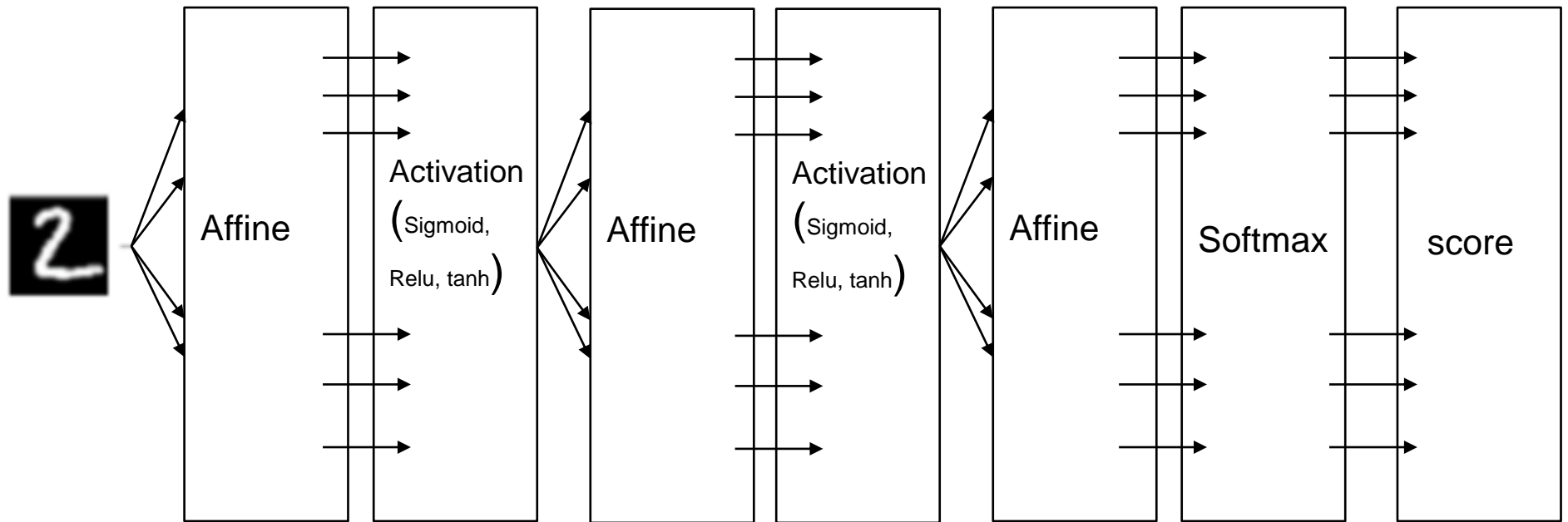
$$\textcircled{1} \quad \frac{dL}{dx} = \frac{dL}{dO} * W^t$$

$$\textcircled{2} \quad \frac{dl}{dW} = X^t * \frac{dL}{dO}$$

$$\textcircled{3} \quad \frac{dL}{dB} = \frac{dL}{dO}$$

# Back Propagation

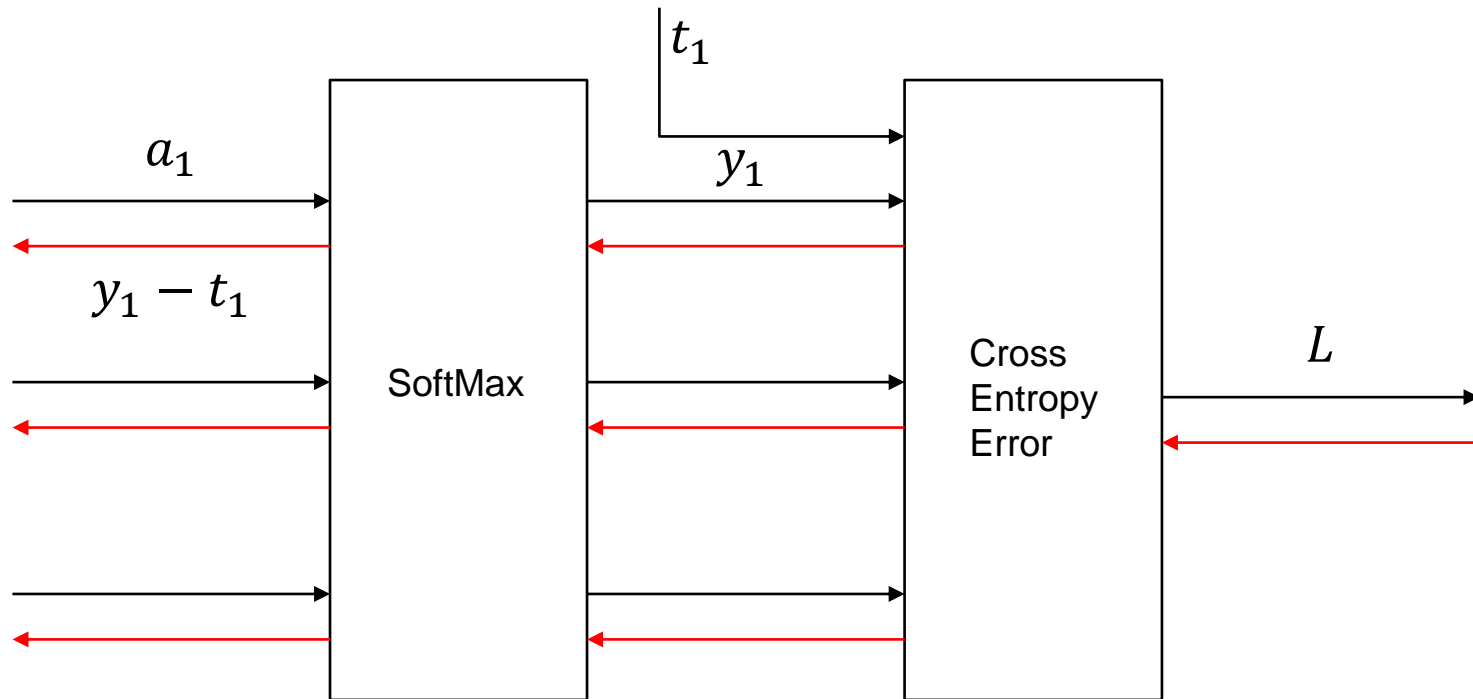
Softmax-with-Loss 계층 역전파





# Back Propagation

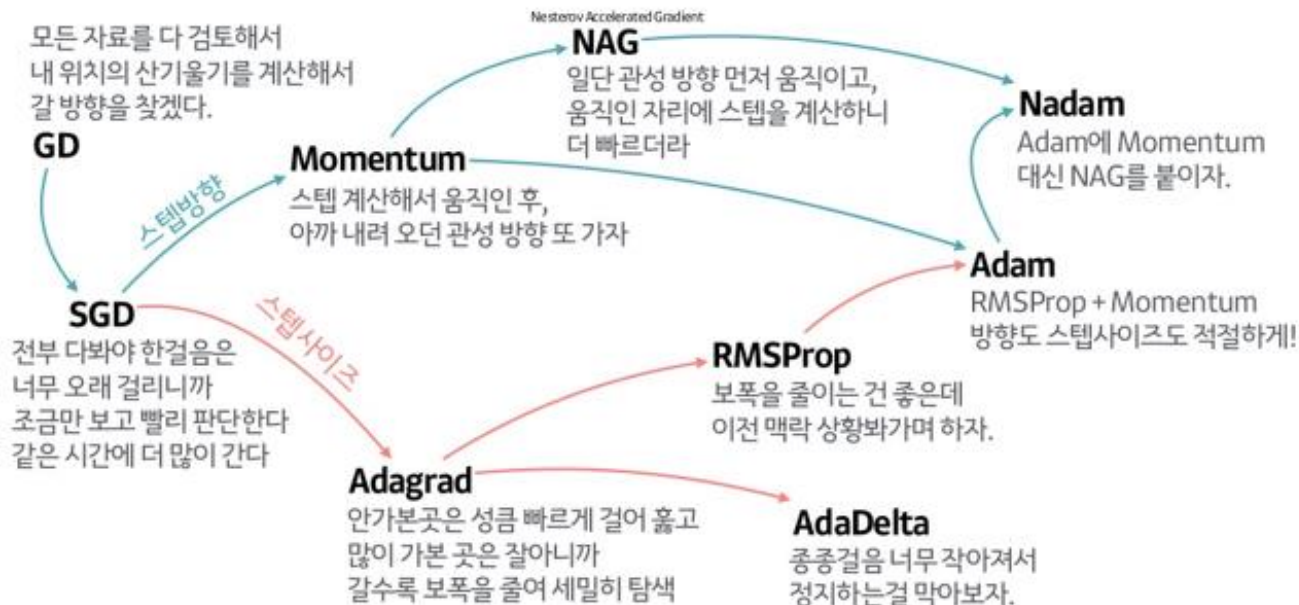
Softmax-with-Loss 계층 역전파



# Optimizer

## 매개변수(Parameter) 갱신

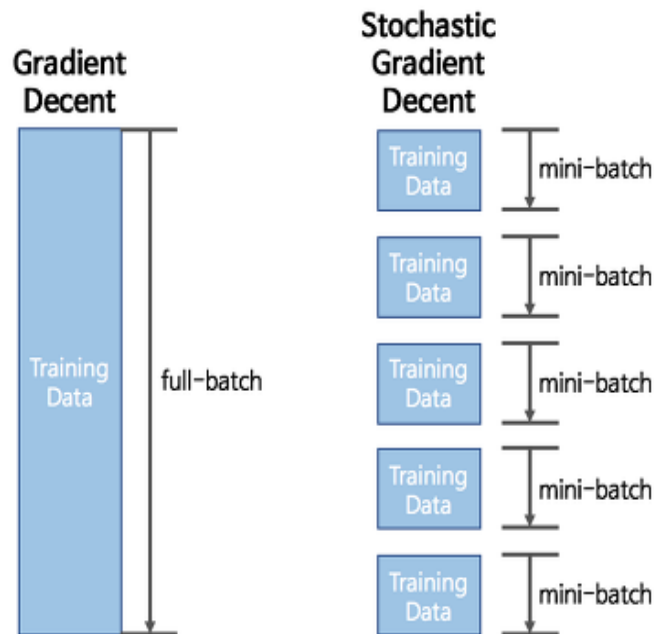
- 신경망 학습의 목적 : 손실함수의 값을 가능한 한 낮추는 매개변수를 탐색
- 옵티마이저 : 가중치 매개변수의 최적값을 탐색하여 최적화
- 매개변수 공간은 매우 복잡
- 매개변수의 값을 찾기 위해서는 기울기(Gradient)가 유일한 단서
- 기울기를 활용한 다양한 최적화 기법(Optimizer) 연구



# Optimizer

## SGD(Stochastic Gradient Descent) 확률적 경사하강법

- Gradient Descent 방법은 steepest descent 방법
- 함수 값이 낮아지는 방향으로 독립 변수 값을 변형시켜가면서 최종적으로는 최소 함수 값을 갖도록 하는 독립 변수 값을 찾는 방법



전체 데이터를 사용하는 것이 아니라, **랜덤하게 추출한 일부 데이터**를 사용하는 것  
따라서 **학습 중간 과정에서 결과의 진폭이 크고 불안정하며, 속도가 매우 빠르다.** 또한, 데이터 하나씩 처리하기 때문에 오차율이 크고 GPU의 성능을 모두 활용하지 못하는 단점을 가짐  
이러한 단점들을 보완하기 위해 나온 방법들이 Mini batch를 이용한 방법이며, 확률적 경사하강법의 노이즈를 줄이면서도 전체 배치보다 더 효율적

$$w := w - \eta * \frac{dx}{dL}$$

# Optimizer

---

## Momentum

- Momentum은 운동량을 뜻함
- Momentum 즉 관성을 이용하는 방법은 우리의 직관을 그대로 반영
- 현재 파라미터를 업데이트해 줄 때 이전 **gradient** 들도 계산해 포함하여 문제 해결
  - SGD 가 미끄럼틀 끝에서 **gradient** 가 0인 면을 만났을 경우
  - 미끄럼틀 빗면의 **gradient** 가 남아있어 SGD를 앞으로 밀어줌
- 계속 미끄러지는 것을 방지하기 위해 이전 **gradient**들의 영향력을 매 업데이트시  $r$  배씩 감소

$$v_t = rv_{t-1} + \eta g_{t-1}$$

$$\theta_{t+1} = \theta_t - v_t$$

# Optimizer

---

## AdaGrad

- $\mu$ (learning rate) : 너무 작으면 학습시간 길고 **overfitting** 가능성 높아짐, 너무 크면 발산하여 학습 안됨
- learning rate decay : 학습을 진행하면서 학습률을 점차 줄여가는 방법
- AdaGrad : 변수들을 **update**할 때 각각의 변수마다 **step size** 를 다르게 설정하는 방식  
“지금까지 많이 변화하지 않은 변수들은 **step size**를 크게 하고, 많이 변화한 변수들은 작게하자 ”

$$h \leftarrow h + \frac{\partial L^2}{\partial W}$$

$$W \leftarrow W + \eta \frac{1}{\sqrt{h}} \frac{\partial L}{\partial W}$$

# Optimizer

---

## RMSprop

- AdaGrad의 learning rate 가 작아지는 문제를 해결
- 적절한  $\gamma$  값으로 0.9,  $\eta$  값으로 0.001을 제안

$$E(g^2)_t = rE(g^2)_{t-1} + 0.1g_t^2$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E(g^2)_t}} g_t$$

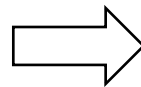
# Optimizer

## Adam(Adaptive Moment Estimation)

- RMSProp와 Momentum 기법을 합친 optimizer
- 모멘텀에서는 관성계수  $m$ 과 함께 계산된  $v_t$  로 파라미터를 업데이트 하지만 Adam에서는 기울기 값과 기울기의 제곱값의 지수이동평균을 활용하여 Step 변화량을 조절

1)  $E(g^2)_t, E(g)_t$ 를 업데이트에 사용  
 $m_t$  를 gradient에 대한 1차 moment  
 $v_t$  를 gradient에 대한 2차 moment

2) 가중평균식 (지수이동평균)  
 $m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$   
 $v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$



$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t}} \hat{m}_t$$

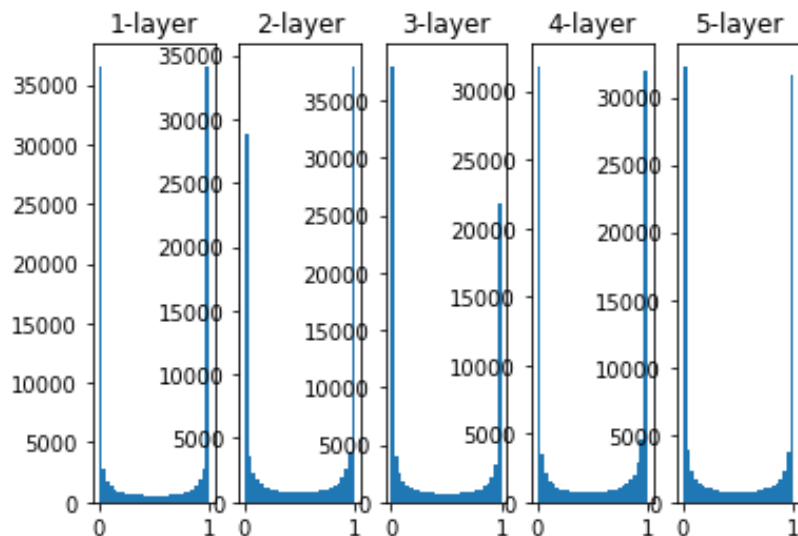
3) Bias Correct

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \quad \hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

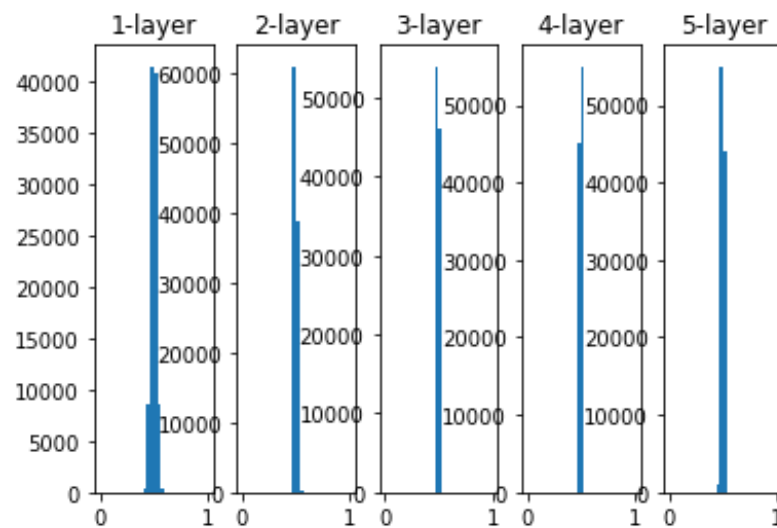
# 가중치 초기값

## Sigmoid 함수를 사용한 은닉층의 활성화 값 분포

- 각 층의 활성화 값은 적당히 고루 분포 되어야 함
- 층과 층 사이에 적당하게 다양한 데이터가 흐르게 해야 신경망 학습이 효율적으로 이루어짐
- 치우친 데이터가 흐르면 기울기 소실이나 표현력 제한 문제가 발생



표준편차가 1일경우 분포



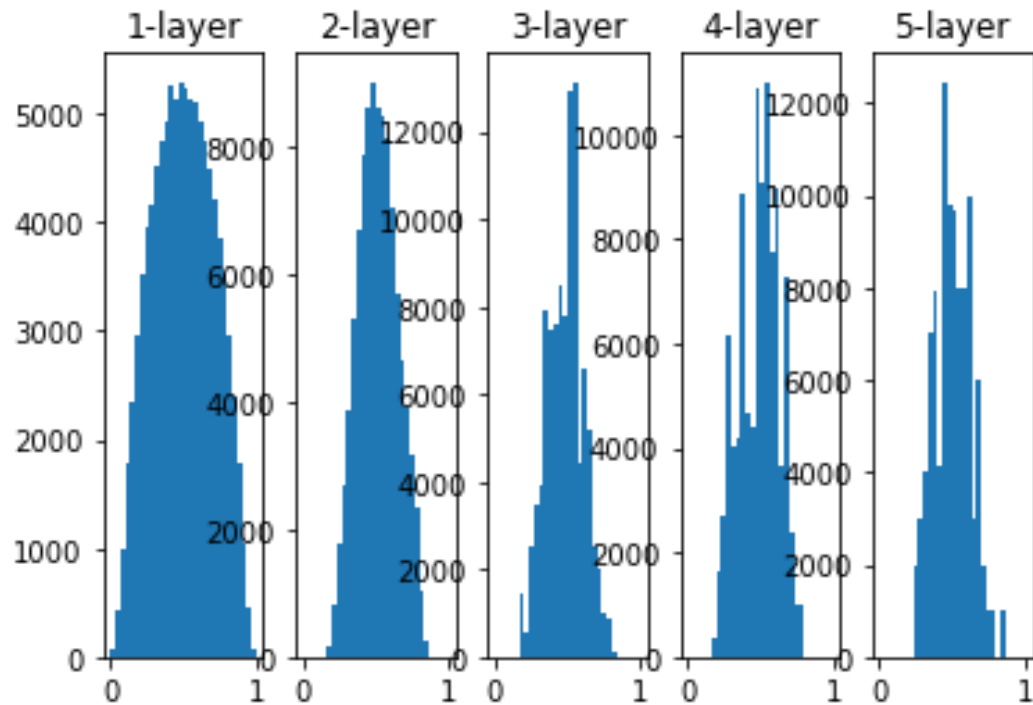
표준편차가 0.01일경우 분포



# 가중치 초기값

## Xavier 초기값

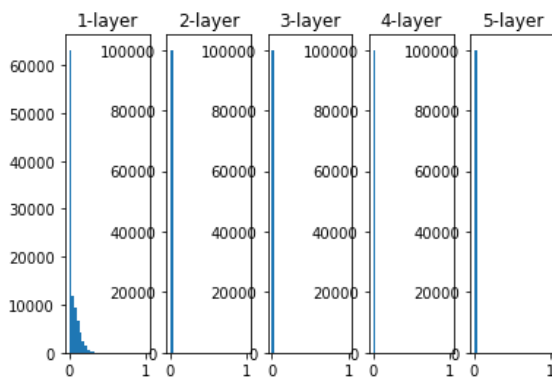
- 각 층의 가중치의 고른 분포를 위해 앞 계층의 노드가  $n$ 개라면 표준편차가  $\frac{1}{\sqrt{n}}$  인 분포를 사용



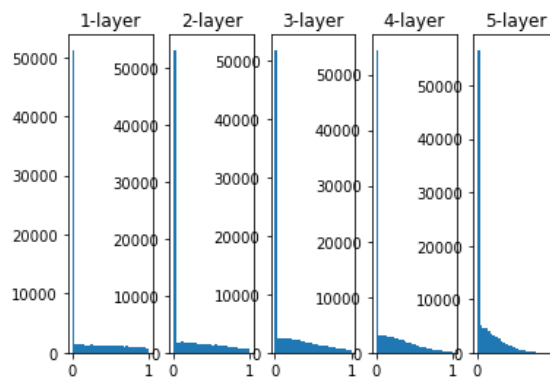
# 가중치 초기값

## He(Kaiming He) 초기값

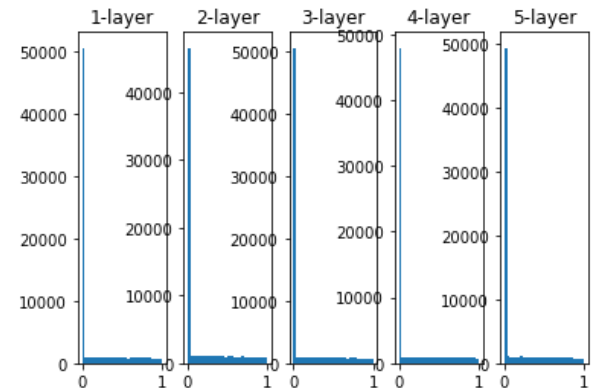
- Xavier 초기값은 활성화 함수가 선형인 함수에 적합
- Relu 함수는 음의 영역이 0이기 때문에 넓은 분포가 필요
- He 초기값은  $\sqrt{\frac{2}{n}}$  를 사용



표준편차 0.01



Xavier



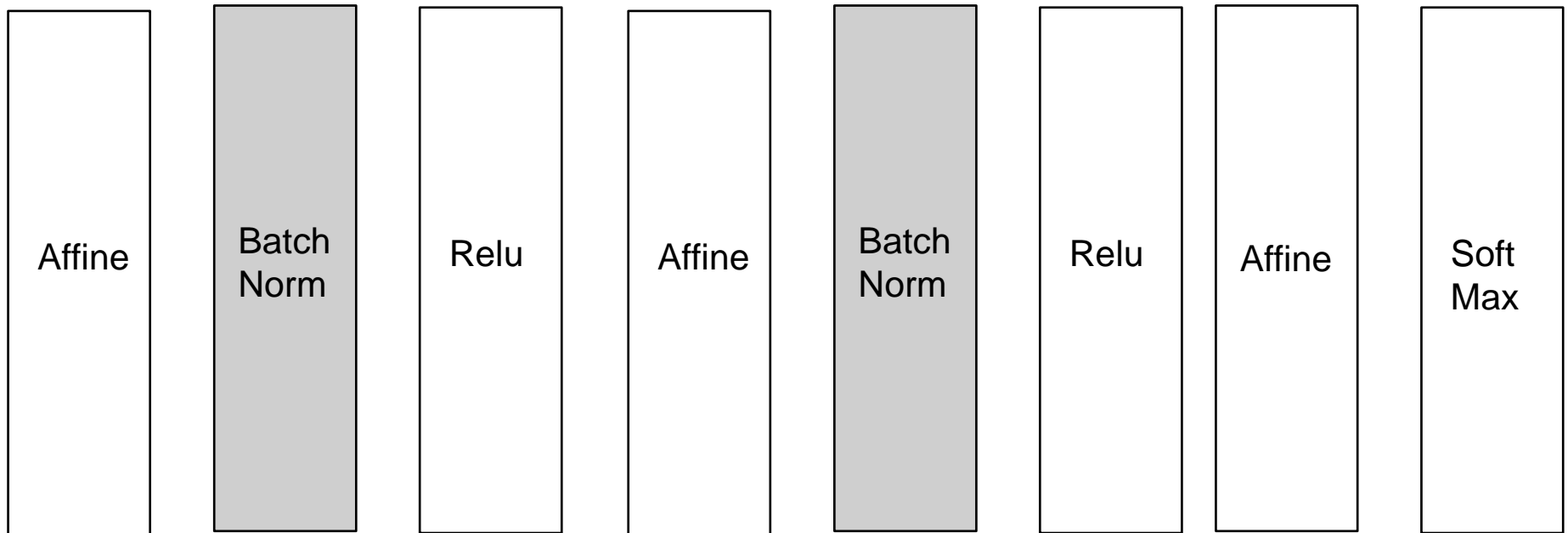
He

# 배치 정규화

---

## Batch Normalization(배치 정규화)

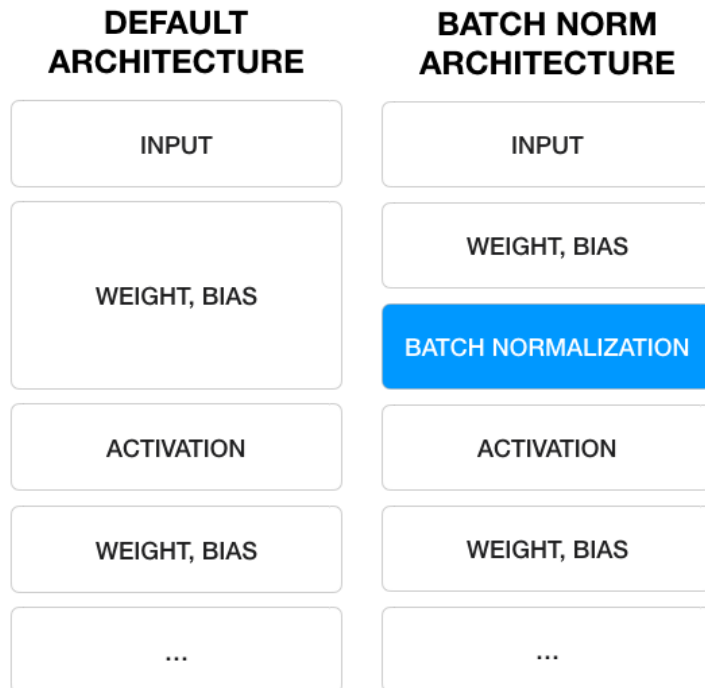
- 배치 정규화 : 각 층의 활성화값 분포를 강제로 퍼뜨리는 방법
- 학습속도의 개선
- 초기값에 크게 의존하지 않음
- 오버피팅 억제(Drop out 필요성 감소)



# 배치 정규화

## 배치 정규화를 하는 이유

- 빠른 학습속도
- 초기값에 대한 의존성 적음
- 과적합 방지



- **Covariate Shift**: 학습에 사용한 데이터와 테스트 데이터의 차이가 생기는 것
- **Internal Covariate Shift** : 각 레이어에서 업데이트된 가중치에 따라 생기는 차이
- 각 레이어의 입력데이터를 평균 0, 분산 1인 데이터로 변환하여 다음 레이어에 전달
- 각 **Batch Normalization** 마다 **scale, shift** 변환 수행

# 배치 정규화

---

## 배치 정규화 알고리즘

- 미니배치를 단위로 정규화
- 데이터 분포가 평균이 0, 분산이 1이 되도록 정규화
- 배치 정규화 계층마다 정규화된 데이터에 고유한 확대와 이동 변환 수행

$$\mu = \frac{1}{m} \sum x_i \quad \sigma^2 = \frac{1}{m} * \sum (x_i - \mu)^2 \quad x^- = \frac{x_i - \mu}{\sigma}$$

$$y_i = \gamma \hat{x}_i + B$$

# Over fitting

---

## Weight decay

- 오버피팅이 일어나는 경우
  - 매개변수가 많고 표현력이 높은 모델
  - 훈련데이터가 적은 모델
- 오버피팅이 발생하는 원인은 일반적으로 가중치 매개변수의 값이 클 경우 발생
- 가중치 감소(Weight decay)는 큰 가중치에 대해서 **Penalty**를 부과
- L2 법칙에 따라 페널티 부여

$$\frac{1}{2}\lambda w^2 = \text{가중치 감소}$$

$\lambda$ 는 정규화의 세기를 조절하는 하이퍼파라미터

# Over fitting

## Weight decay

- L1 Loss

$$L = \sum_{i=1}^n |y_i - f(x_i)|$$

Least absolute Errors(LAE)

- L2 Loss

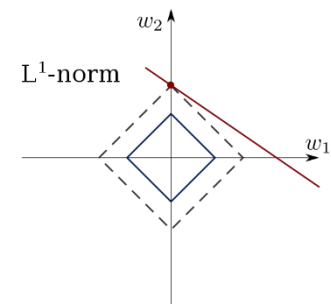
$$L = \sum_{i=1}^n (y_i - f(x_i))^2$$

Least squares error(LSE)

- outlier에 민감

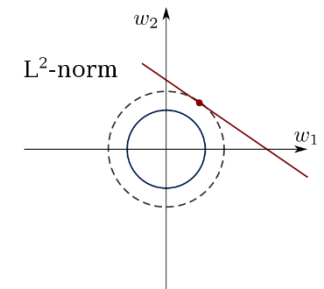
- L1 Regularization

$$Cost = \frac{1}{n} \sum_{i=1}^n \{L(y_i, \hat{y}_i) + \frac{\lambda}{2} |w|\}$$



- L2 Regularization

$$Cost = \frac{1}{n} \sum_{i=1}^n \{L(y_i, \hat{y}_i) + \frac{\lambda}{2} |w|^2\}$$

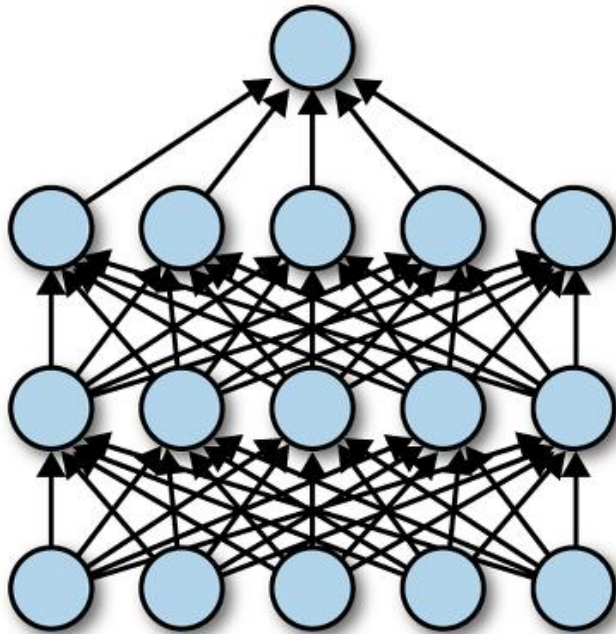


# Over fitting

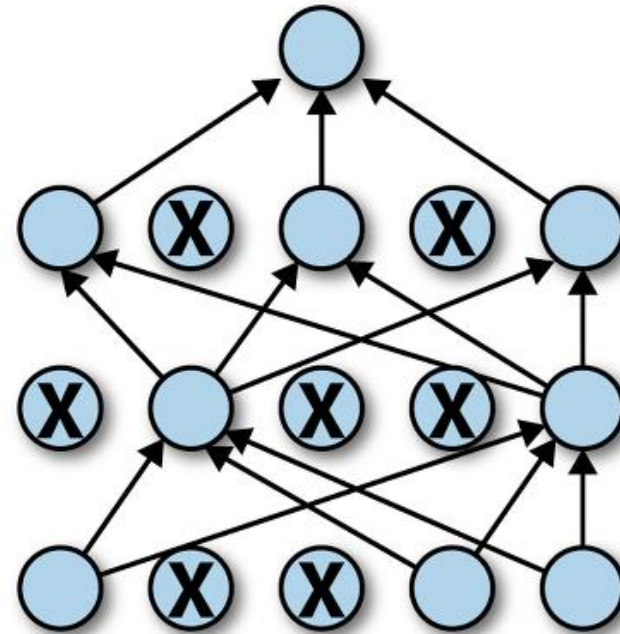
---

## Drop out

- Drop out ratio 지정
- 일반적으로 입력층과 출력층에 다르게 적용



(a) Standard Neural Net



(b) After applying dropout



# Over fitting

---

## Drop out과 Ensemble

- 머신러닝에서 Ensemble 을 애용
- 앙상블 학습은 개별적으로 학습시킨 여러 모델의 출력을 평균 내어 추론
- 신경망의 관점에서 같은 구조 혹은 비슷한 구조의 네트워크를 여러 개 준비하여 따로 학습 시키고 시험 때는 각 네트워크의 출력을 평균 내어 출력
- 앙상블 학습을 수행하면 신경망의 정확도가 개선되는 것은 실험적으로 증명
- 드롭아웃과 앙상블 학습은 유사

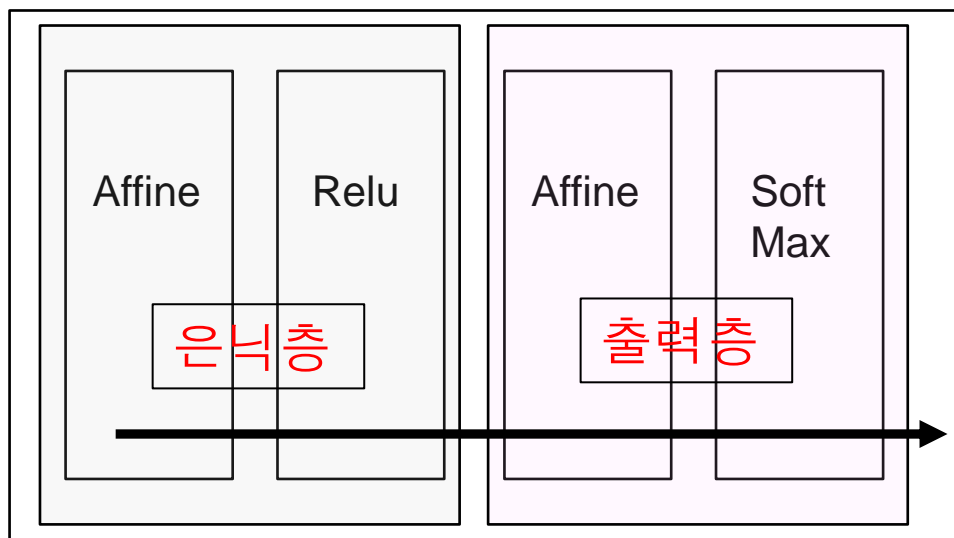
# **CNN (Convolution Neural Network)**

---

# 합성곱 신경망

## 완전연결계층

- CNN 은 이미지인식과 음성인식등 다양한 분야에서 사용
- 이미지 인식 분야에서는 **CNN** 이 거의 사용됨
- CNN은 Convolution Layer(합성곱 계층)과 Pooling Layer( 풀링 계층)으로 구성



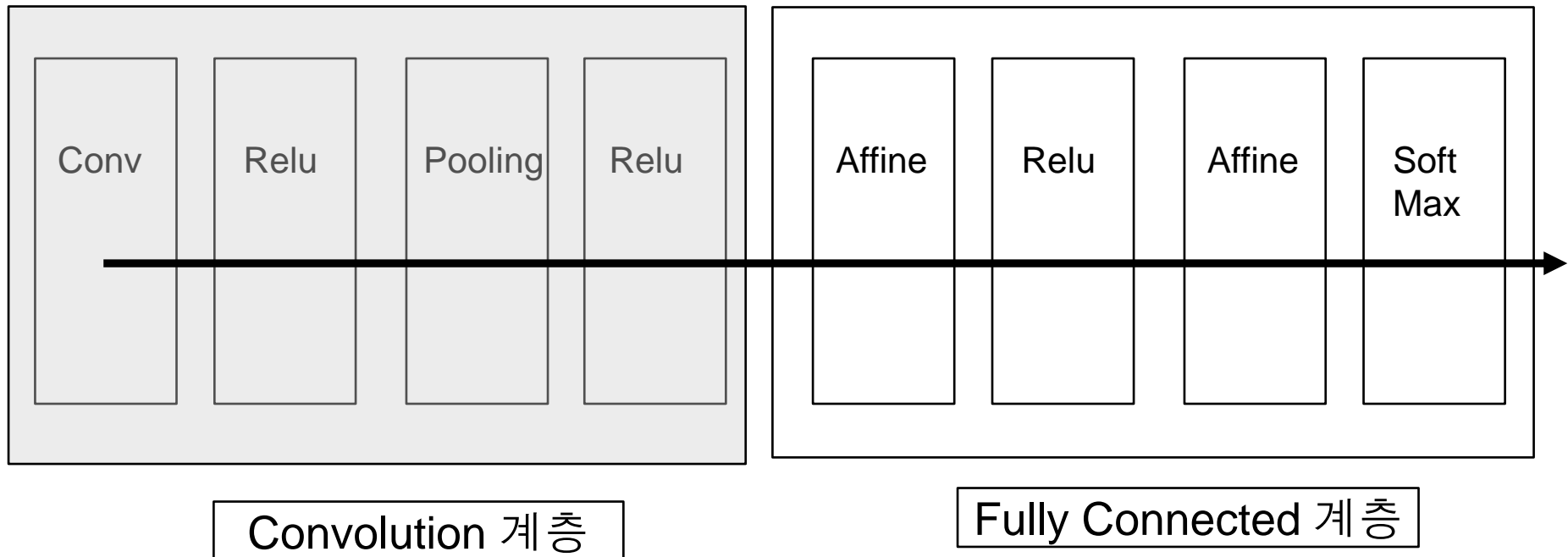
## Fully Connected Layer

완전연결 신경망은 Affine 계층 뒤에 활성화 함수 계층으로 구성된 은닉층과 Affine 계층과 Softmax 함수를 활성화(정규화)함수로 구성된 출력층으로 구성됨

# 합성곱 계층

## 완전연결 계층과 합성곱 계층

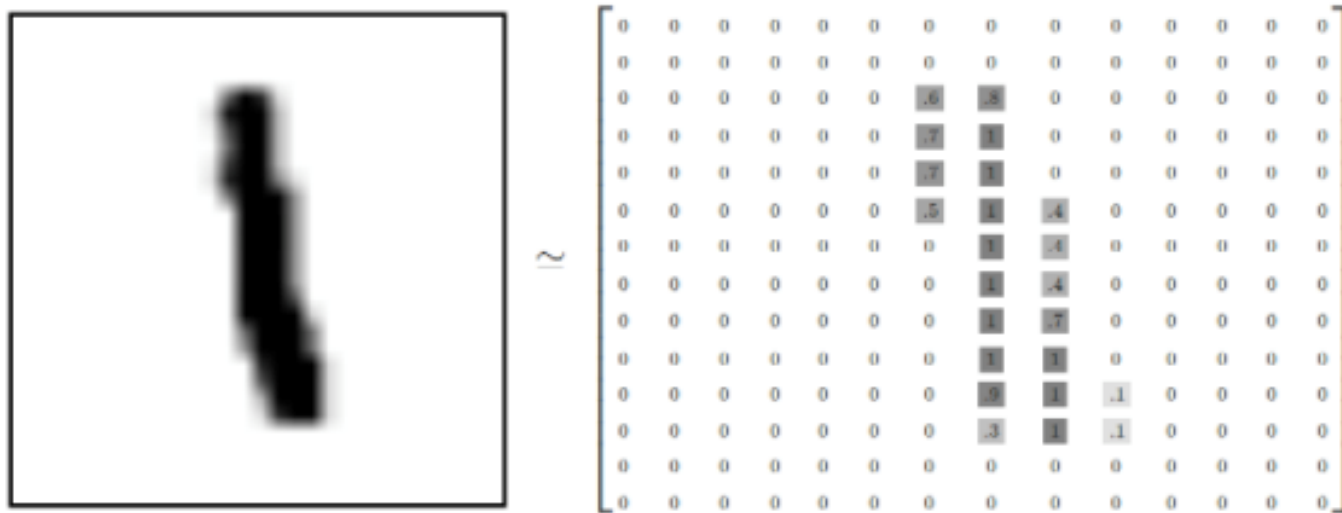
- CNN을 사용하면 완전연결 계층에 합성곱 계층을 추가하여 새로운 네트워크 구성 가능



# 합성곱 계층

## 합성곱 계층을 사용하는 이유

- 이미지는 일반적으로 세로\*가로\*채널(색상)으로 구성된 3차원 데이터
- 완전연결 계층에서는 3차원 데이터를 → 1차원으로 변환해야 함
- 완전연결 계층에서 데이터의 형상이 무너지는 상황 발생



[0,0,0, 0,0, 0,0, 0,0, 0,0,..... 0,0, 0,0, 0,0, 0,0, 0,0]

# 합성곱 연산

## 합성곱 연산 방법

- 합성곱 계층에서는 합성곱 연산을 처리
- 합성곱 연산은 특징을 추출(필터 혹은 커널)하기 수행

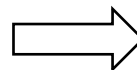
1	2	3	0
0	1	2	3
3	0	1	2
2	3	0	1

입력데이터

⊗

2	0	1
0	1	2
1	0	2

필터(커널)



15	16
6	15

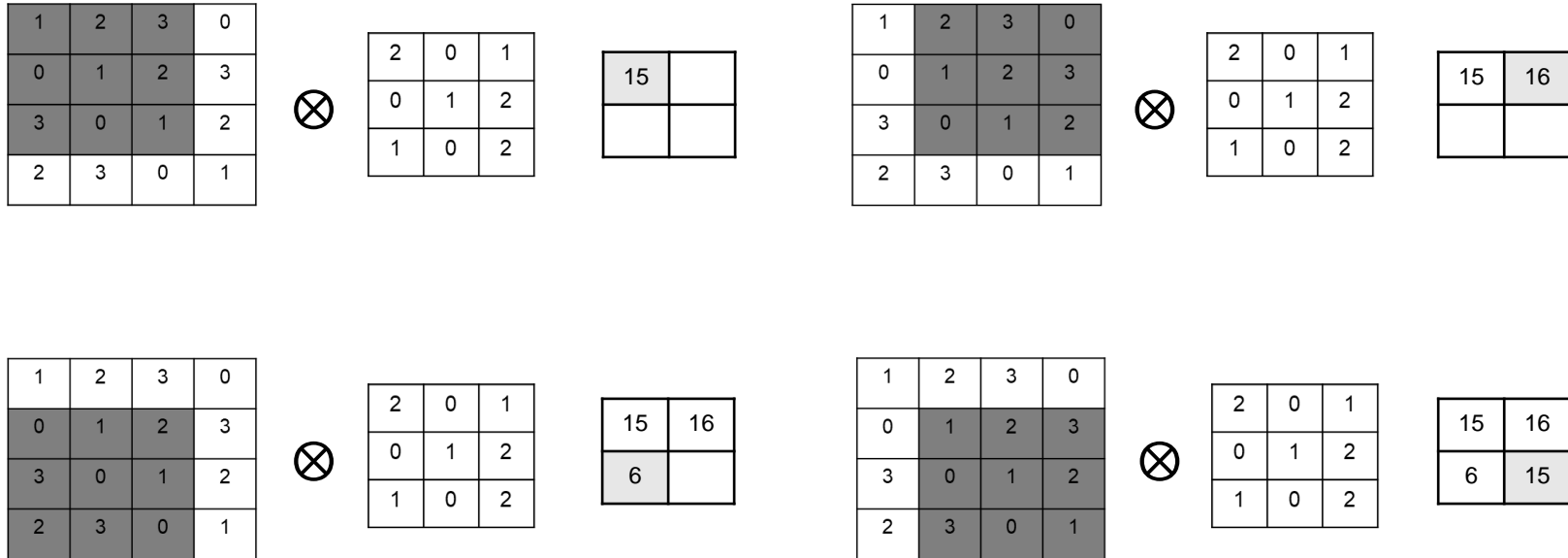
합성곱 연산은 필터(커널)의 윈도우를 일정 간격으로 이동해가며 입력 데이터에 적용하여 ⊗ 연산을 적용 하여 결과를 해당장소에 저장

※ ⊗(fused multiply add) : 단일 곱셈 누산

# 합성곱 연산

## FMA(Fused Multiply Add)

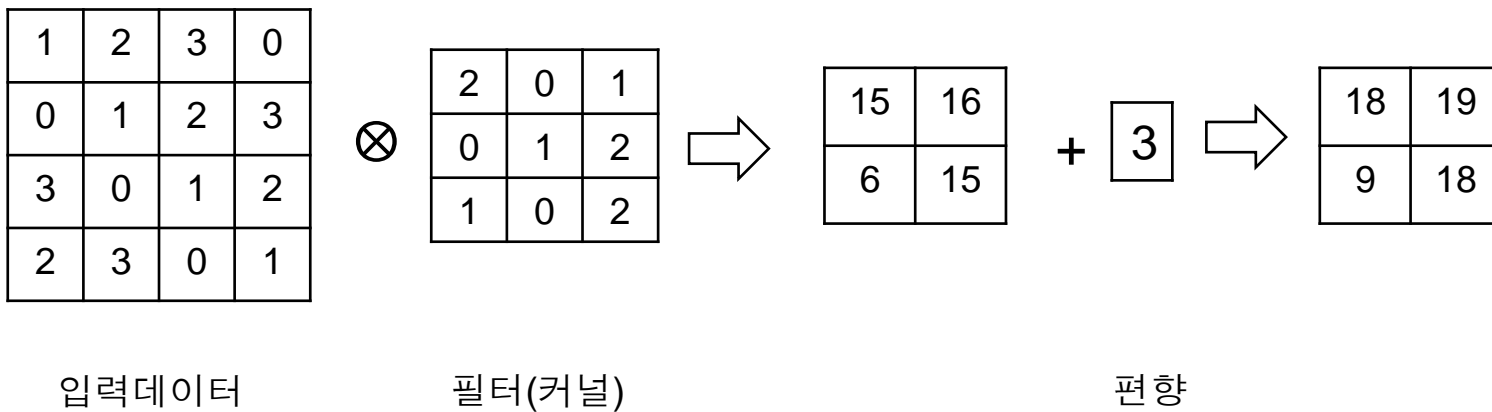
- FMA 연산은 필터의 윈도우에 대응하는 원소끼리 곱한 후 구 총합을 구함



# 합성곱 연산

## 합성곱 연산의 편향(bias)

- 합성곱 층에서도 편향은 존재
- 합성곱 연산의 편향은 필터를 적용한 원소에 편향을 합함





# 합성곱 연산

## 패딩(Padding)

- **Padding** : 합성곱 연산을 수행하기 전에 입력 데이터 주변을 특정 값(주로 0)으로 채움
- 패딩을 하는 목적은 출력 크기를 조절  
입력데이터(4x4)에 필터(3x3)를 적용시 출력(2x2) → 입력보다 2만큼 줄어들음  
합성곱 연산이 거듭 되면 출력값은 1로 수렴
- 출력값의 1로 수렴을 막고자 패딩 설정

1	2	3	0
0	1	2	3
3	0	1	2
2	3	0	1

입력데이터  
4x4

	1	2	3	0	
	0	1	2	3	
	3	0	1	2	
	2	3	0	1	

입력데이터(패딩:1)  
6x6

		1	2	3	0		
		0	1	2	3		
		3	0	1	2		
		2	3	0	1		

입력데이터(패딩:2)  
8x8

# Padding

## 패딩(Padding)

- **Padding** : 합성곱 연산을 수행하기 전에 입력 데이터 주변을 특정 값(주로 0)으로 채움
- 패딩을 하는 목적은 출력 크기를 조절  
입력데이터(4x4)에 필터(3x3)를 적용시 출력(2x2) → 입력보다 2만큼 줄어들음  
합성곱 연산이 거듭 되면 출력값은 1로 수렴
- 출력값의 1로 수렴을 막고자 패딩 설정

1	2	3	0
0	1	2	3
3	0	1	2
2	3	0	1

입력데이터  
4x4

	1	2	3	0	
	0	1	2	3	
	3	0	1	2	
	2	3	0	1	

입력데이터(패딩:1)  
6x6

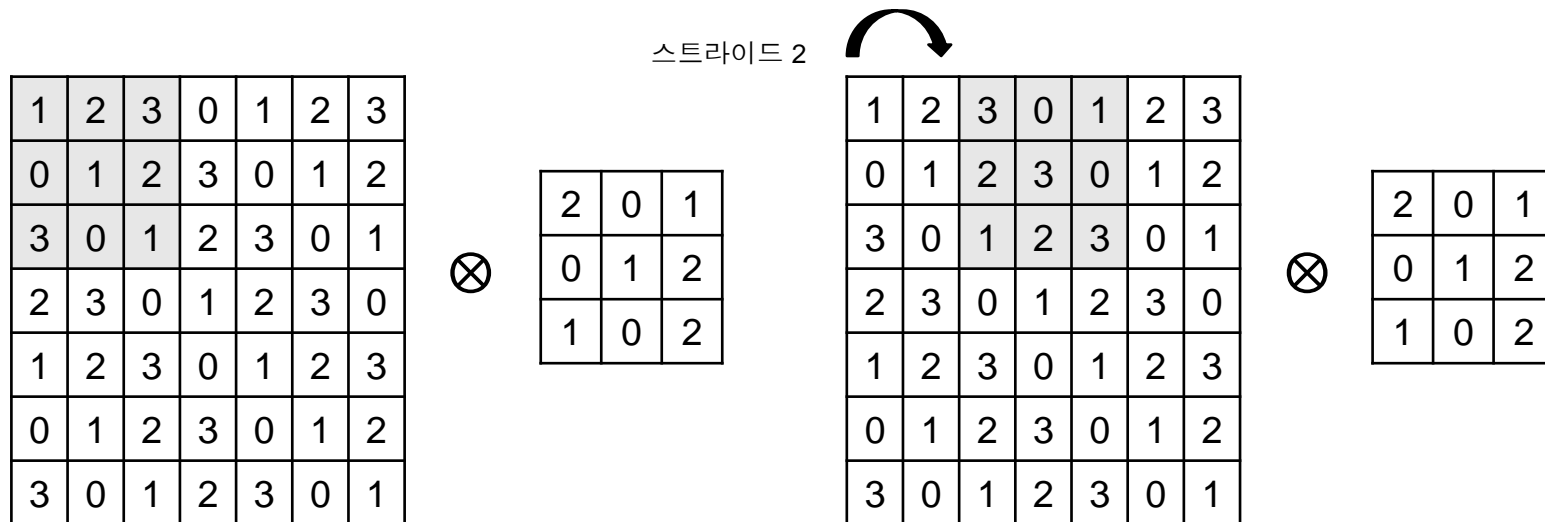
		1	2	3	0		
		0	1	2	3		
		3	0	1	2		
		2	3	0	1		

입력데이터(패딩:2)  
8x8

# Stride

## 스트라이드(Stride)

- Stride : 필터를 적용하는 위치의 간격



# Stride

## 스트라이드를 적용한 입출력 계산

- 입력데이터 Shape : Height, Width (H,W)
- 필터 Shape : Height, Width (FH, FW)
- 출력데이터 Shape : Height, Width (OH, OW)
- 패딩 : P
- 스트라이드 : S

$$OW = \frac{W+2P-FW}{S}+1$$

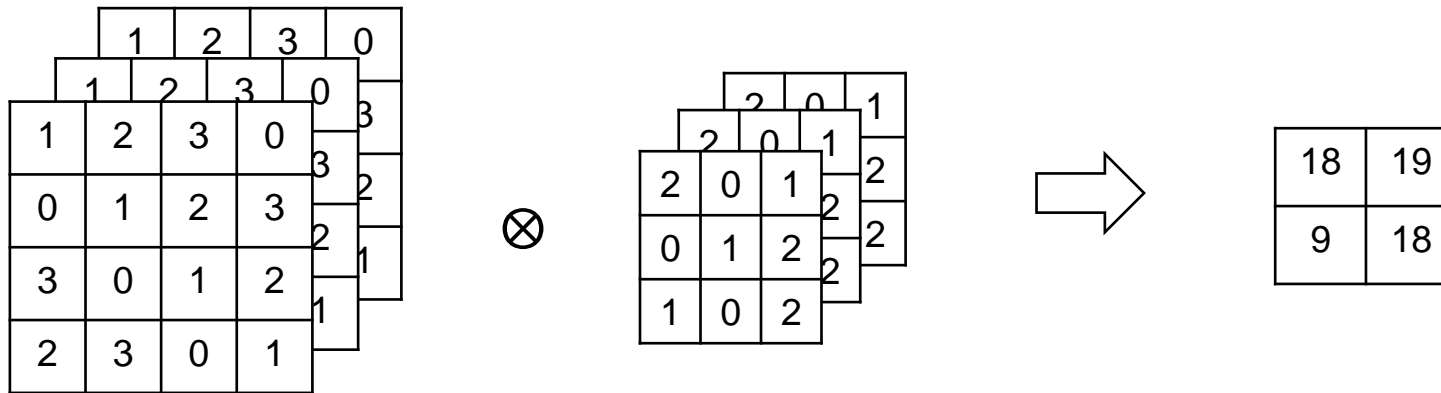
$$OH = \frac{H+2P-FH}{S}+1$$

스트라이드를 계산할 때  $\frac{H+2P-FH}{S}$   $\frac{W+2P-FW}{S}$  는 항상 정수로 나누어 져야 함

# 3차원 데이터의 합성곱

## 3차원 데이터의 합성곱 연산

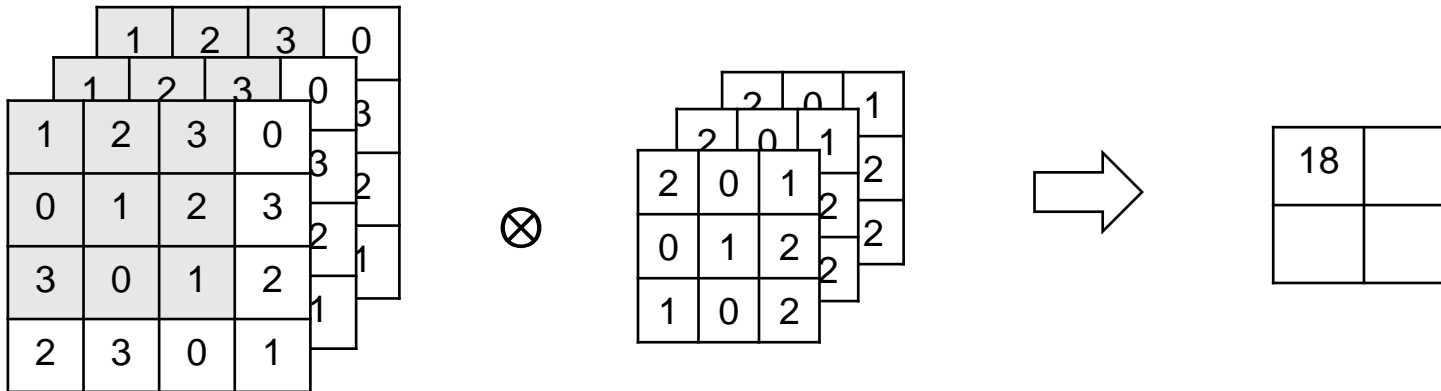
- 이미지데이터 Shape : Height, Width, Channel 구조



# 3차원 데이터의 합성곱

## 3차원 데이터의 합성곱 연산

- 3차원 합성곱 연산은 입력 데이터의 채널 수와 필터의 채널 수가 같아야 함

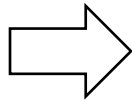


# Pooling Layer

## 풀링 연산

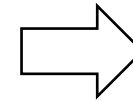
- 풀링은 세로 가로 방향의 공간을 줄이는 연산
- 풀링 연산은 **Max Pooling**, **Avg Pooling** 이 있음 (이미지 데이터에서는 주로 **Max Pooling** 이 사용됨)

1	2	3	0
0	1	2	3
3	0	1	2
2	3	0	1



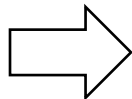
2	

1	2	3	0
0	1	2	3
3	0	1	2
2	3	0	1



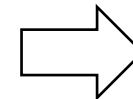
2	3

1	2	3	0
0	1	2	3
3	0	1	2
2	3	0	1



2	3
3	

1	2	3	0
0	1	2	3
3	0	1	2
2	3	0	1



2	3
3	2

# Pooling Layer

## 풀링 계층의 특징

- 풀링 계층은 학습해야 할 매개변수가 없음
- 채널수가 변하지 않음

		1	2	3	0	
	1	2	3	0		
1	2	3	0		3	
0	1	2	3		2	
3	0	1	2		1	
2	3	0	1			

입력데이터



		2	3	
	2	3		
2	3		2	
3	2			

출력데이터



# Pooling Layer

## 풀링 계층의 특징

- 입력의 변화에 영향을 적게 받음
- CNN에는 많은 convolution layer를 쌓기 때문에 많은 필터의 문제점
  - 필터가 많다는 것은 그만큼 feature map 들이 쌓이게 됨 (차원이 매우 커짐)
  - 파라메터가 많아 지면 학습 시 over fitting 발생
  - 차원을 감소시킬 방법 필요
- CNN 에서 차원을 줄여주는 레이어가 Pooling Layer

1	2	0	7	1	0
0	9	2	3	2	3
3	0	1	2	1	2
2	4	0	1	0	1
6	0	1	2	1	2
2	4	0	1	8	1

9	7
6	8

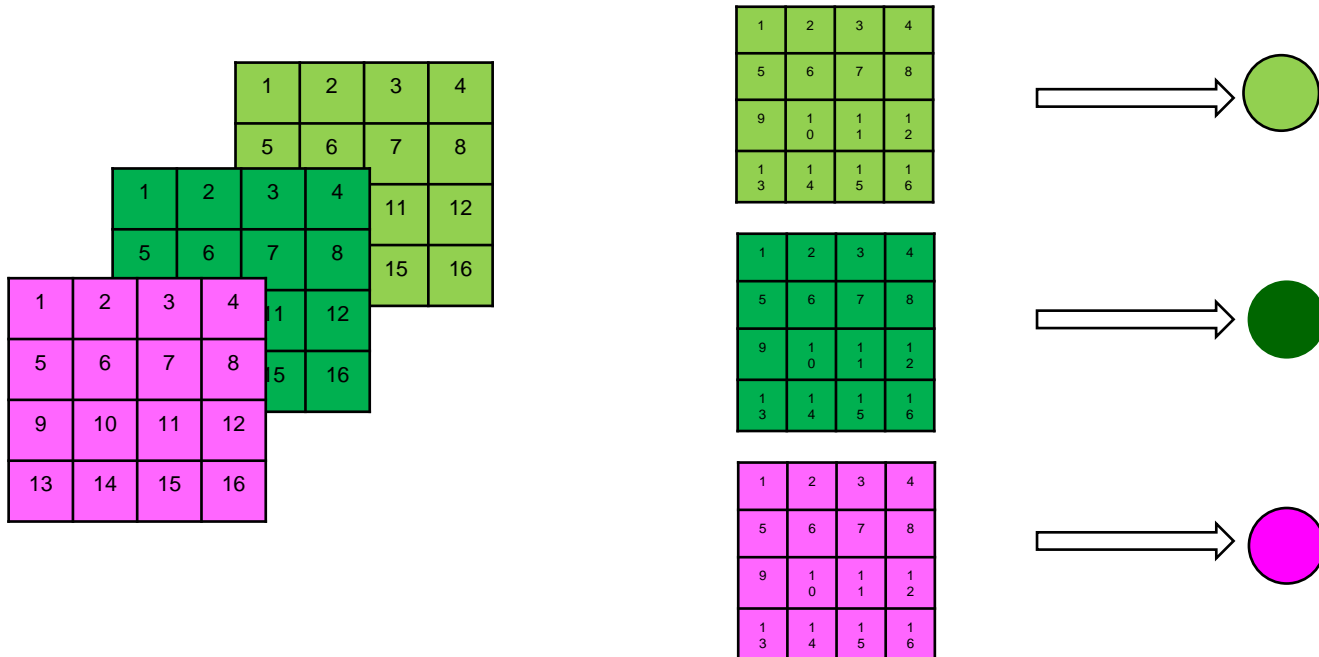
1	2	0	0	7	0
3	0	9	3	2	3
2	3	1	2	1	2
2	4	0	1	0	1
2	6	1	2	1	2
2	4	0	0	1	8

9	7
6	8

# CNN 예측

## 전층 연결 방법

- FC Layer와 연결하기 위해서는 1차원의 텐서 필요
- Flatten Layer : 여러차원의 텐서를 1차원으로 변환
- GlobalMax(Avg)Pooling1D Layer : Feature를 1차원 텐서(벡터)로 변환



# CNN 학습 시각화

---

## Conv Net 시각화 의미

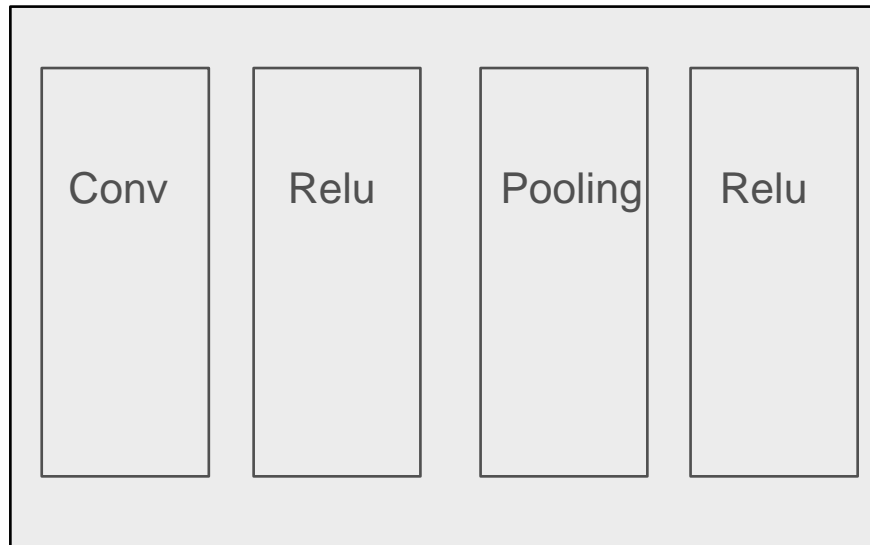
- 딥러닝 모델의 중간 계층의 의미
  - 일반적인 딥러닝 모델은 흔히 **black box**이라 표현
  - 학습된 표현에서 사람이 이해하기 쉬운 형태를 뽑거나 제시하기 어려움
- CNN에서 중간계층의 의미
  - 컨브넷의 표현은 시각적인 개념을 학습한 것이기 때문에 시각화에 의미가 있음
  - 2013년 부터 CNN의 중간계층의 시각화하고 해석하는 다양한 기법이 존재
- 컨브넷 중간층의 출력 시각화
  - 연속된 컨브넷 층이 입력을 어떻게 변형시키는지 이해 및 개별적인 컨브넷 필터의 의미를 파악
- 컨브넷 필터 시각화
  - 컨브넷의 필터가 찾으려는 시각적인 패턴과 개념을 이해
- 클래스 활성화에 대한 히트맵을 이미지에 시각화
  - 이미지의 어느 부분이 주어진 클래스에 속하는데 기여했는지 확인하고 이미지에서 객체 위치를 추정

# Simple CNN 구현

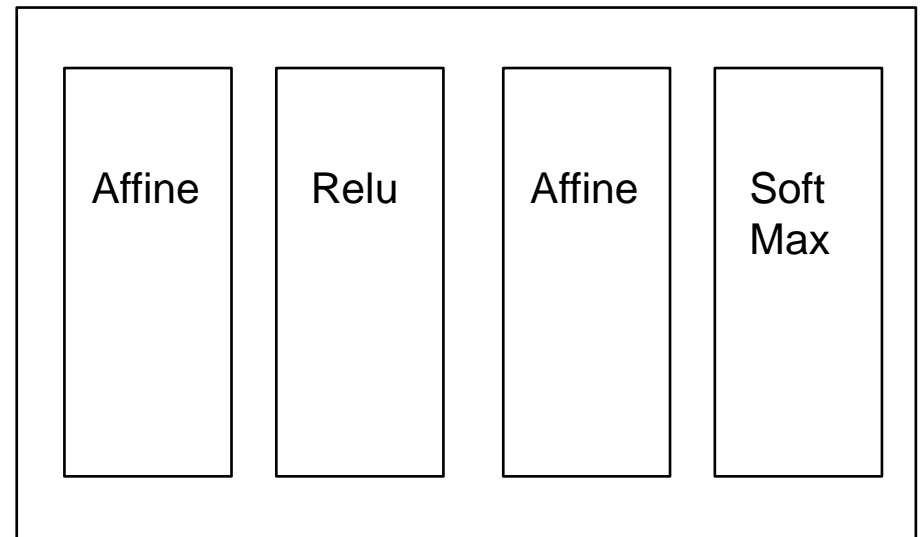
---

## Simple CNN 구현

- 하나의 Convolution Layer 를 구현하여 Fully Connecte Layer 로 출력



Convolution 계층

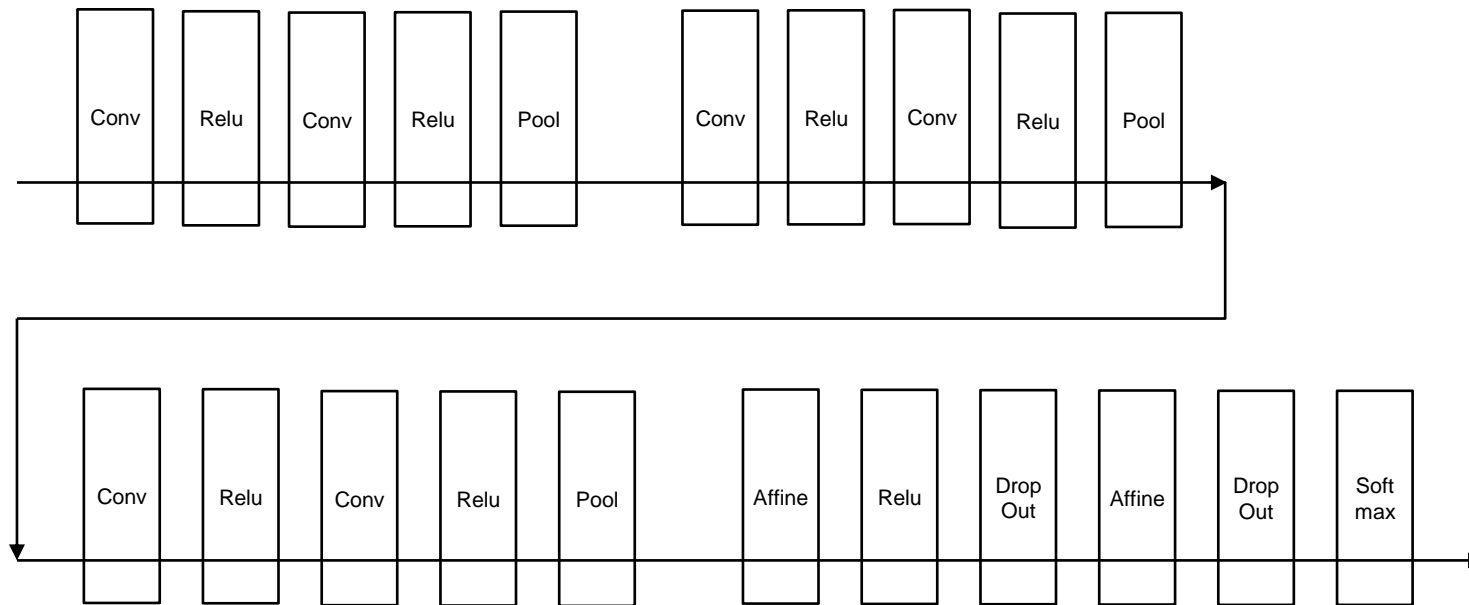


Fully Connected 계층

# Deep CNN 구현

## VGG CNN 구현

- Deep Convolution Layer 를 구현하여 Fully Connect Layer 로 출력



# CNN with keras

---

## 케라스 특징

- 모듈화 (Modularity)
  - 케라스에서 제공하는 모듈은 독립적이고 설정 가능하며, 가능한 최소한의 제약사항으로 서로 연결될 수 있음. 모델은 시퀀스 또는 그래프로 이러한 모듈들을 구성
  - 특히 신경망 층, 비용함수, 최적화, 초기화기법, 활성화함수, 정규화기법은 모두 독립적인 모듈이며, 새로운 모델을 만들기 위해 이러한 모듈을 조합
- 최소주의 (Minimalism)
  - 각 모듈은 짧고 간결
  - 모든 코드는 한 번 훑어보는 것으로도 이해가능
  - 단 반복 속도와 혁신성에는 다소 떨어질 수가 있음
- 쉬운 확장성
  - 새로운 클래스나 함수로 모듈을 아주 쉽게 추가 가능
  - 따라서 고급 연구에 필요한 다양한 표현 가능

# CNN with keras

---

## 케라스 구조

1. 데이터셋 생성하기
  1. 원본 데이터를 불러오거나 시뮬레이션을 통해 데이터를 생성
  2. 데이터로부터 훈련셋, 검증셋, 시험셋을 생성
  3. 딥러닝 모델의 학습 및 평가를 할 수 있도록 포맷 변환
2. 모델 구성하기
  1. 시퀀스 모델을 생성한 뒤 필요한 레이어를 추가
  2. 좀 더 복잡한 모델이 필요할 때는 케라스 함수 API를 사용
3. 모델 학습과정 설정하기
  1. 학습하기 전에 학습에 대한 설정을 수행
  2. 손실 함수 및 최적화 방법을 정의
  3. 케라스에서는 compile() 함수를 사용
4. 모델 학습시키기
  1. 훈련셋을 이용하여 구성한 모델로 학습.
  2. 케라스에서는 fit() 함수를 사용
5. 학습과정 살펴보기
  1. 모델 학습 시 훈련셋, 검증셋의 손실 및 정확도를 측정
  2. 반복횟수에 따른 손실 및 정확도 추이를 보면서 학습 상황을 판단
6. 모델 평가하기
  1. 준비된 시험셋으로 학습한 모델을 평가.
  2. 케라스에서는 evaluate() 함수를 사용
7. 모델 사용하기
  1. 임의의 입력으로 모델의 출력
  2. 케라스에서는 predict() 함수를 사용

# CNN with keras

---

## 케라스 사용 방법

- # 0. 사용할 패키지 불러오기
  - `from keras.utils import np_utils`
  - `from keras.datasets import mnist`
  - `from keras.models import Sequential`
  - `from keras.layers import Dense, Activation`
- # 1. 데이터셋 생성하기
  - `(x_train, y_train), (x_test, y_test) = mnist.load_data()`
  - `x_train = x_train.reshape(60000, 784).astype('float32') / 255.0`
  - `x_test = x_test.reshape(10000, 784).astype('float32') / 255.0`
  - `y_train = np_utils.to_categorical(y_train)`
  - `y_test = np_utils.to_categorical(y_test)`
- # 2. 모델 구성하기
  - `model = Sequential()`
  - `model.add(Dense(units=64, input_dim=28*28, activation='relu'))`
  - `model.add(Dense(units=10, activation='softmax'))`
- # 3. 모델 학습과정 설정하기
  - `model.compile(loss='categorical_crossentropy', optimizer='sgd', metrics=['accuracy'])`



# CNN with keras

---

## 케라스 사용 방법

- # 4. 모델 학습시키기
- `hist = model.fit(x_train, y_train, epochs=5, batch_size=32)`
- # 5. 학습과정 살펴보기
- `print('## training loss and acc ##')`
- `print(hist.history['loss'])`
- `print(hist.history['accuracy'])`
- # 6. 모델 평가하기
- `Loss_and_metrics = model.evaluate(x_test, y_test, batch_size=32)`
- `Print('## evaluation loss and_metrics ##')`
- `Print(loss_and_metrics)`
- # 7. 모델 사용하기
- `xhat = x_test[0:1]`
- `yhat = model.predict(xhat)`
- `print('## yhat ##')`
- `print(yhat)`

# CNN with keras

## 케라스 사용 방법

- Over fitting Stop을 위해 callback 함수 사용

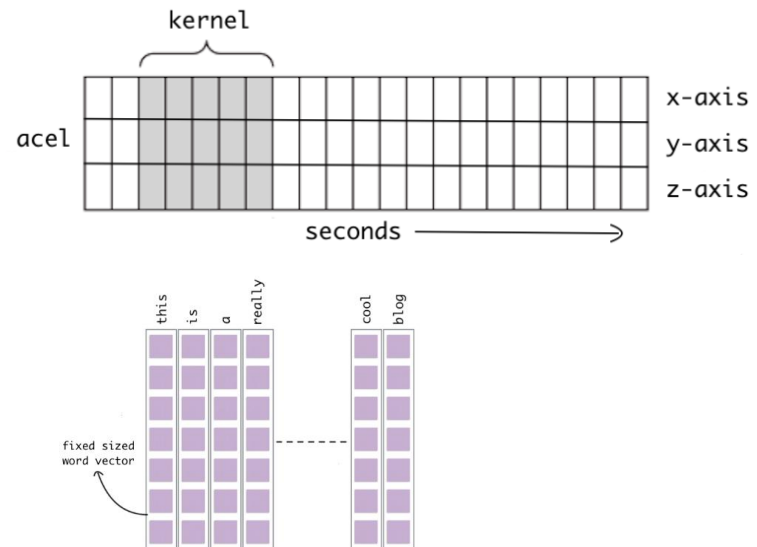
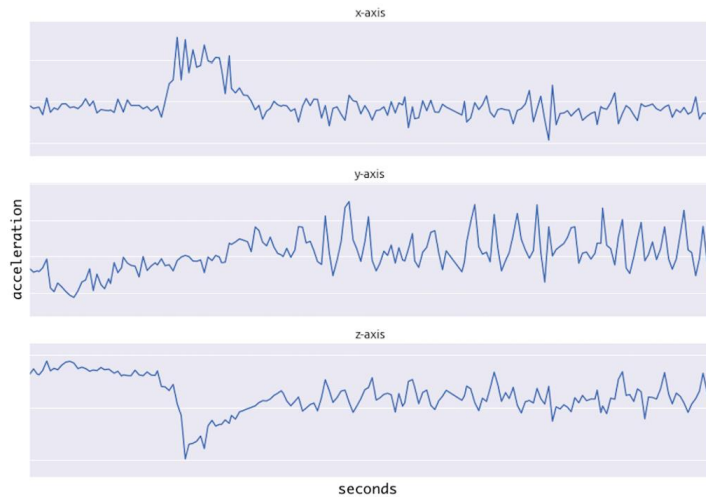
```
from keras.callbacks import EarlyStopping
early_stopping = EarlyStopping(monitor='val_loss', min_delta=0, patience=0, verbose=0, mode='auto')
model.fit(X_train, Y_train, epochs= 1000, callbacks=[early_stopping])
```

- monitor : 관찰하고자 하는 항목 'val\_loss'나 'val\_acc'가 주로 사용
- min\_delta : 개선되고 있다고 판단하기 위한 최소 변화 만약 변화량이 min\_delta보다 적은 경우에는 개선이 없다고 판단
- patience : 개선이 없다고 바로 종료하지 않고 개선이 없는 에포크를 얼마나 기다려 줄 것인 가를 지정 만약 10이라고 지정하면 개선이 없는 에포크가 10번째 지속될 경우 학습일 종료
- verbose : 얼마나 자세하게 정보를 표시할 것인가를 지정합니다. (0, 1, 2)
- mode : 관찰 항목에 대해 개선이 없다고 판단하기 위한 기준을 지정
  - 예를 들어 관찰 항목이 'val\_loss'인 경우에는 감소되는 것이 멈출 때 종료되어야 하므로, 'min'으로 설정
  - auto : 관찰하는 이름에 따라 자동으로 지정합니다.
  - min : 관찰하고 있는 항목이 감소되는 것을 멈출 때 종료합니다.
  - max : 관찰하고 있는 항목이 증가되는 것을 멈출 때 종료합니다.

# CNN with keras

## Conv1D

- Conv1D 적용방법



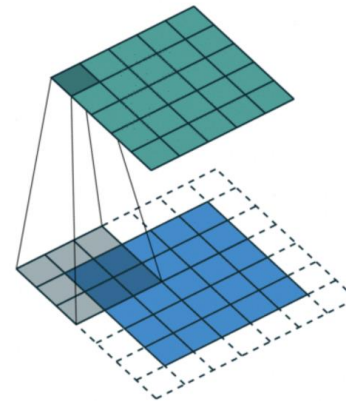
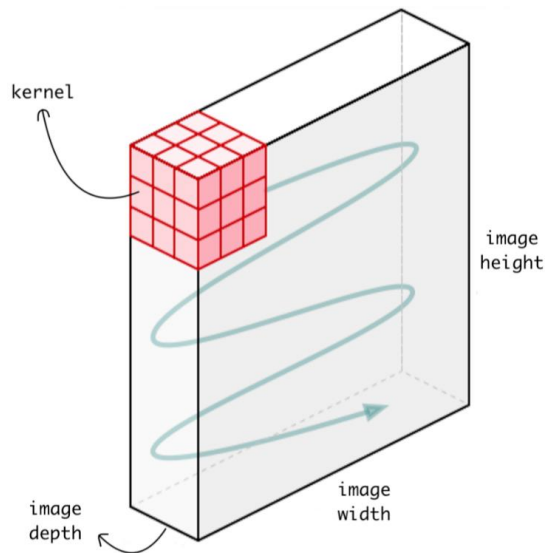
```
from keras.layers import Conv1D
model = Sequential()
model.add(Conv1D(100, kernel_size=3, padding='SAME', input_shape = (120, 3)))
model.summary()
```

# CNN with keras

---

## Conv2D

- Conv2D 적용방법

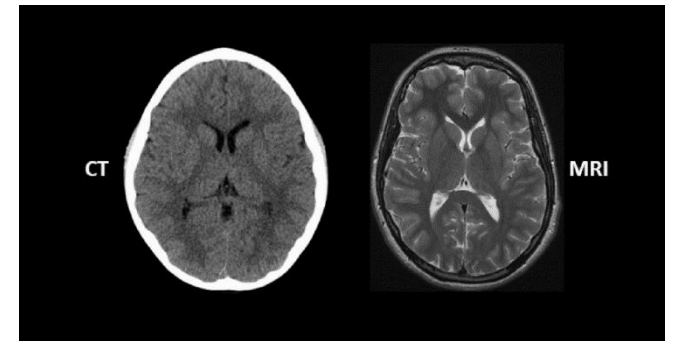
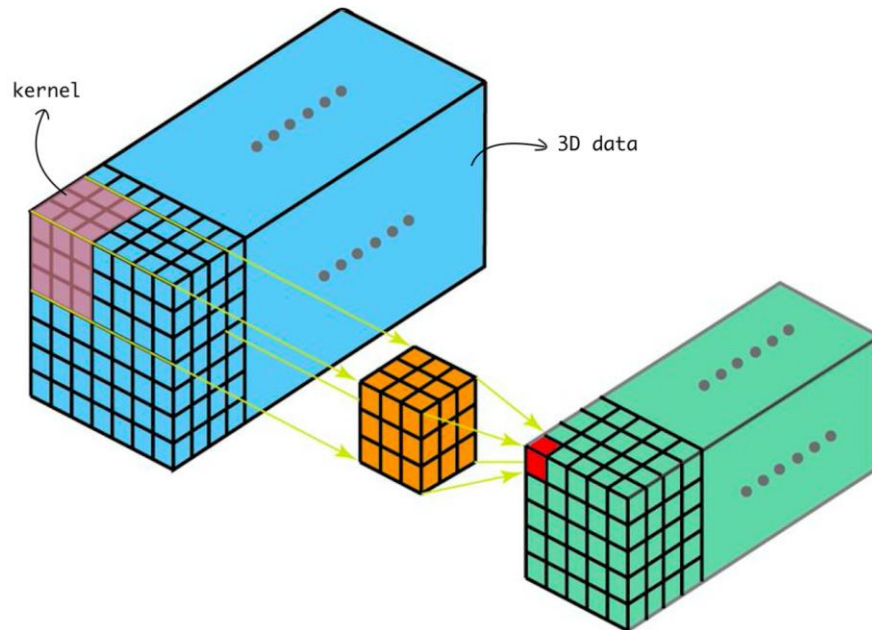


```
from keras.layers import Conv2D
from keras.models import Sequential
model = Sequential()
model.add(Conv2D(1, kernel_size=(3,3), input_shape = (128, 128, 3)))
model.summary()
```

# CNN with keras

## Conv3D

- Conv3D 적용방법



```
from keras.layers import Conv3D
model = Sequential()
model.add(Conv3D(100, kernel_size=(3,3,3), input_shape = (120, 120, 120, 3)))
model.summary()
```

# **RNN (Recurrent Neural Network)**

---

# 시계열(Time Series) 데이터 이해

---

## 시계열 형태

- 시계열 분석은 특정 대상의 시간에 따라 변하는 데이터를 사용하여 추세를 분석
- 불규칙 변동(irregular variation)
  - 시계열 자료에서 시간에 따른 규칙적인 움직임과 달리 어떤 규칙성이 없어 예측 불가능하고 우연적으로 발생하는 변동
- 추세 변동(trend variation)
  - 시계열 자료가 갖는 장기적인 변화 추세를 의미, **trend**는 장기간에 걸쳐 지속적으로 증가, 감소 혹은 일정한 상태(**stationary**)를 유지하려는 성향
- 순환 변동(cyclical variation)
  - 대체로 2~3년 정도의 일정한 기간을 주기로 순환적으로 나타나 변동
- 계절 변동(seasonal variation)
  - 시계열 자료에서 보통 계절적 영향과 사회적 관습에 따라 1년 주기로 발생하는 것을 의미 일반적으로 계절에 따라 순환하며 변동하는 특성

# 시계열 분석 모델

## 불규칙 시계열 데이터의 모델

- 시계열 데이터는 규칙적 시계열과 불규칙적 시계열로 나뉨
- 규칙적 시계열은 트렌드와 분산이 불변하는 데이터
- 불규칙적 시계열은 트렌드 혹은 분산이 변화하는 데이터
- 시계열 데이터에 특정한 기법이나 모델을 적용하여 규칙적 패턴을 찾거나 예측
- 불규칙적 시계열 데이터에 규칙성을 부여하는 방법으로 → AR, MA, ARMA, ARIMA

## AR 모델

- AR(AutoRegression) : 자기 회귀 모델은 이전 관측 값이 이후 값에 영향을 준다는 아이디어 대한 모형

$$\underbrace{z_t}_{\textcircled{1}} = \underbrace{\phi z_{t-1}}_{\textcircled{2}} + \underbrace{\phi z_{t-2}}_{\textcircled{2}} + \underbrace{\phi z_{t-p}}_{\textcircled{2}} + \underbrace{a_i}_{\textcircled{3}}$$

(1) 시계열 데이터의 현재 시점 (2)과거가 현재에 미치는 영향을 나타내는 모수( $\phi$ )에 시계열 데이터의 과거 시점을 곱함 (3)시계열 데이터의 오차항(백색잡음)



# 시계열 분석 모델

## MA 모델

- MA(Moving Average)(이동평균) 모델은 트렌드가 변화하는 상황에 적합한 회귀 모델

$$z_t = \underbrace{\theta_1 a_{t-1}}_{\textcircled{1}} + \underbrace{\theta_2 a_{t-2} + \theta_p a_{1-p}}_{\textcircled{2}} + \underbrace{a_t}_{\textcircled{3}}$$

(1) 시계열 데이터의 현재 시점 (2)는 매개변수 ( $\theta$ )에 시계열 데이터의 과거 시점의 오차를 곱함

(3) 시계열 데이터의 오차

※AR과 달리 이전상태에서 현재 상태를 추론 하는게 아닌 이전 데이터의 오차에서 현재 데이터의 상태를 추론

## ARMA 모델

- AR과 MA를 합한 모델

$$z_t = \phi_t z_{t-1} + \phi z_{t-2} + \phi z_{t-p} + a_i + \theta_1 a_{t-1} + \theta_2 a_{t-2} + \theta_p a_{1-p} + a_t$$

# 시계열 분석 모델

---

## ARIMA 모델

- ARIMA(AutoRegressive Integrated Moving Average)(자동 회귀 이동평균) 자기회귀와 이동평균을 함께 고려한 모델
- ARMA와 다른 점은 과거 데이터의 선형 관계 뿐만 아니라 추세까지 고려

p : 자기 회귀 차수 - 자기회귀 모형(AR)의 시차  
d : 차분 차수 - 차분누적(I) 횟수  
q : 이동 평균 차수 - 이동평균 모형(MA)의 시차

p와 q는 일반적으로  $p+q < 2$ ,  $p*q=0$  인 값을 사용 p와 q 중 하나는 0이므로 이는 시계열 데이터가 AR이나 MA중 하나의 경향만 가지기 때문

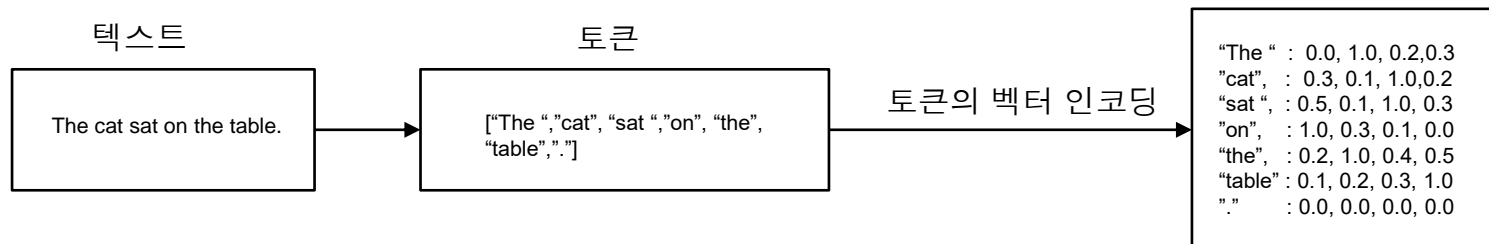
# TEXT와 시퀀스 데이터

## Text 벡터화

- 텍스트를 수치형 텐서로 변환하는 과정을 벡터화(Vectorizing)이라 함
- 텍스트를 단어로 나누고 각 단어를 하나의 벡터로 변환
- 텍스트를 문자로 나누고 각 문자를 하나의 벡터로 변환
- 텍스트에서 단어나 문자의 **n-gram** 을 추출하여 각 **n-gram** 을 하나의 벡터로 변환
  - n-gram은 연속된 단어나 문자의 그룹으로 텍스트에서 단어나 문자를 하나씩 이동하면서 추출

## Token

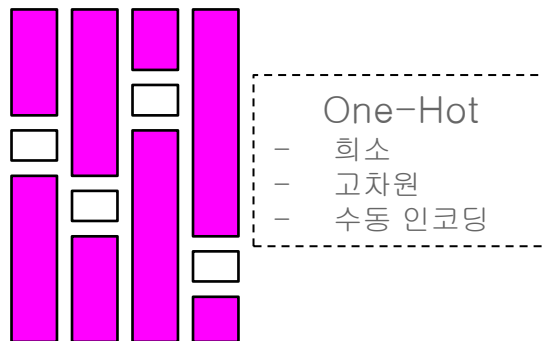
- 토큰(token) 텍스트를 나누는 단위
- 텍스트를 토큰으로 나누는 작업 및 과정을 토큰화(Tokenizing)이라 함



# 벡터화를 위한 인코딩

## One-Hot 인코딩과 Word 임베딩

- One-Hot 인코딩
  - 모든 단어에 고유한 정수 인덱스 부여
  - 정수 인덱스  $i$  를 크기가  $N$ (어휘 사전의 크기)인 이진 벡터로 변환
  - 변환된 벡터는  $i$  번째 원소만 1이고 나머지는 0
- Word 임베딩
  - 분류나 예측을 위한 모델 구현시 모델과 함께 단어 신경망을 구축하여 임베딩
  - 사전 훈련된 단어 임베딩 - 미리 계산된 단어 임베딩을 로드하여 사용



# Embedding Layer

---

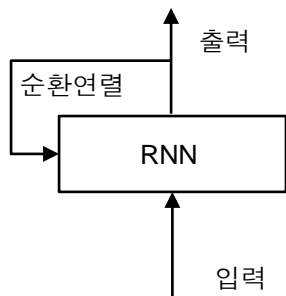
## Embedding Layer

- Embedding층 이해
  - 특정 단어를 나타내는 정수 인덱스를 밀집 벡터로 매핑하는 딕셔너리로 이해
  - 정수를 입력받아 내부 딕셔너리에서 이 정수에 연관된 벡터를 찾아 변환
- Embedding층 구성
  - Embedding 층은 크기가 (samples, sequence\_length)인 2D 정수 텐서를 입력으로 받음
  - Embedding 층은 크기가 (samples, sequence\_length, embedding\_dimensionality)인 3D 실수형 텐서를 반환

# RNN이해

## 순환 신경망과 다른 신경망과의 차이

- 완전 연결 네트워크나 컨브넷과의 특징은 메모리가 없음
- 컨브넷과 완전 연결 신경망에 주입되는 입력은 개별적으로 처리 → 입력간에 유지 상태가 없음
- FC나 CNN으로 시퀀스나 시계열 데이터를 처리 하려면 네트워크에 전체 시퀀스를 주입(Feedforward Network)

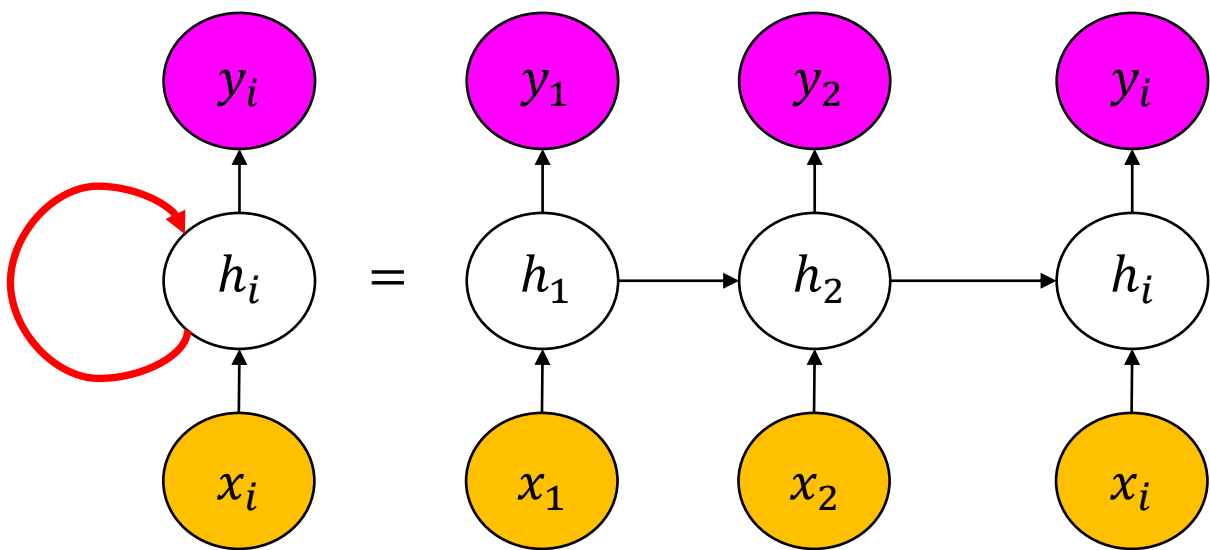


RNN은 크기가 (timestamps, input\_features)로 입력을 받음

# RNN

## 순환 신경망 (RNN)

- RNN 은 시간적으로 연속성이 있는 데이터를 처리하려고 고안
- Recurrent(반복되는)는 이전 은닉층이 현재 은닉층의 입력이 되면서 반복되는 순환 구조
- RNN 네트워크는 기억(Memory)를 갖음
- Memory 의 의미는 현재까지 입력 데이터를 요약한 정보



첫번째 입력이 들어오면  
첫번째 기억(h1)이 만들어  
지고 기존기억(h1)과 새로운  
입력(x2)을 참고하여 새기억  
(h2)를 만든다

# 순환 신경망 유형

## 입력과 출력에 따른 순환 신경망 유형

- One To One : 입력이 하나이고 출력이 하나인 구조
- One To Many : 입력이 하나이고 출력이 다수인 구조
- Many To One : 입력이 다수이고 출력이 하나인 구조
- Many To Many : 입력과 출력이 다수인 구조
- Synchronized Many To Many : 입력과 출력이 다수인 구조

유형별 사용 예

**일대일** – 순환이 없게 때문에 RNN 이라 말하기 어려우며 순방향 네트워크

**일대다** – 이미지를 입력해 이미지에 대한 설명을 문장으로 출력하는 이미지 캡션

**다대일** – 문장을 입력해 긍정/부정을 출력하는 감성 분석기

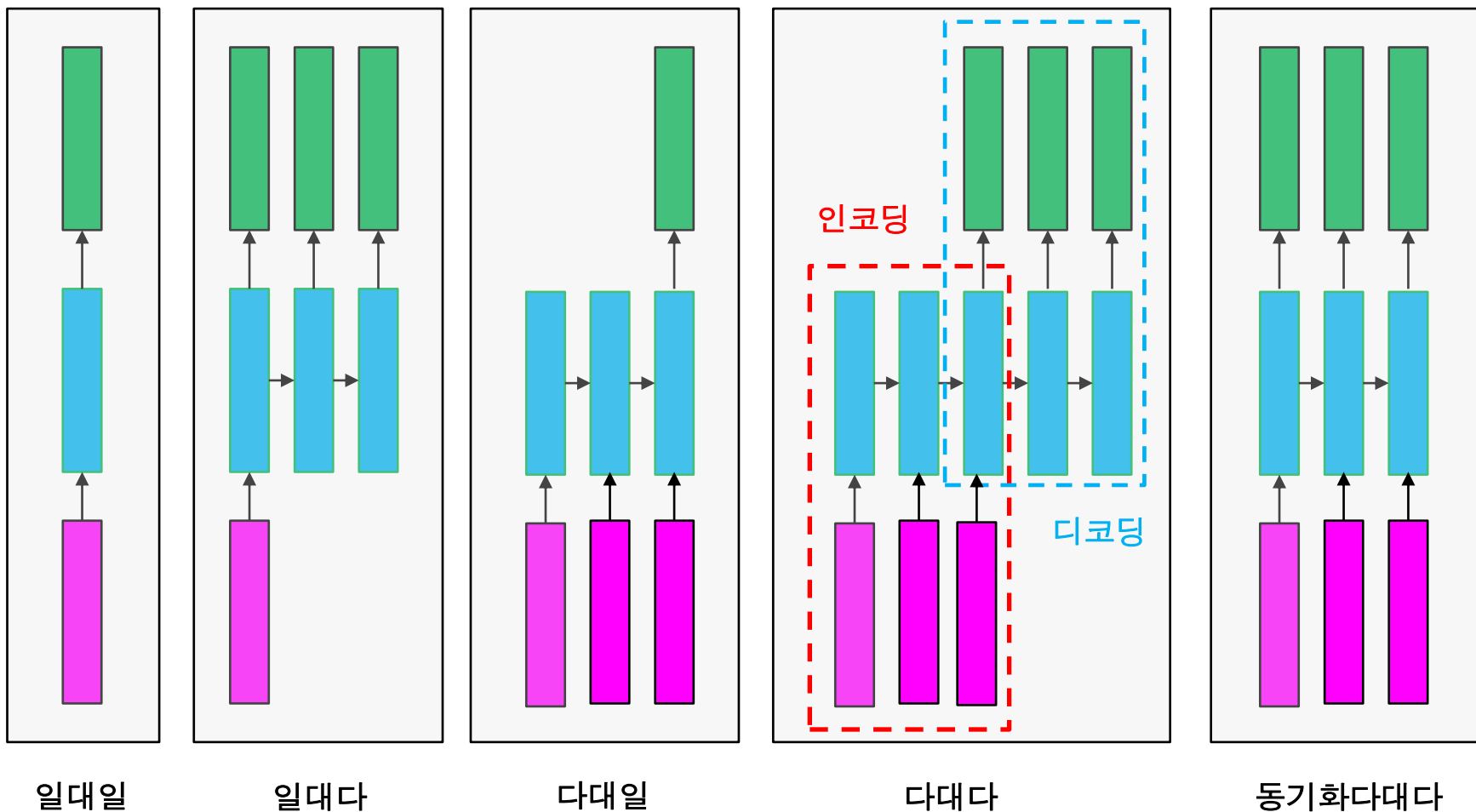
**다대다** – 언어를 번역하는 자동 번역기

**동기화된 다대다** – 문장에서 다음에 나올 단어를 예측하는 언어모델, 프레임 수준의 비디오 분류



# 순환 신경망 유형

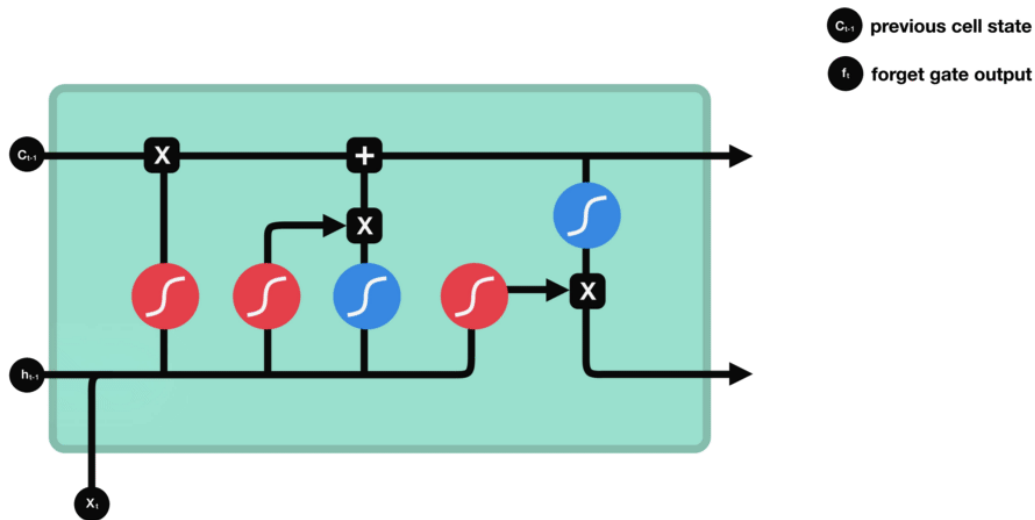
입력과 출력에 따른 순환 신경망 유형



# LSTM (Long Short-Term Memory)

## 망각 게이트

- LSTM 은 기울기 소실 문제를 해결하기 위해 망각 게이트, 입력 게이트, 출력 게이트를 은닉층의 각 뉴런에 추가
- Forget gate(망각 게이트)는 과거 정보를 어느 정도 기억할 지를 결정
- 과거 정보와 현재 데이터를 입력 받아 Sigmoid를 취한 후 그 값을 과거 정보에 곱함

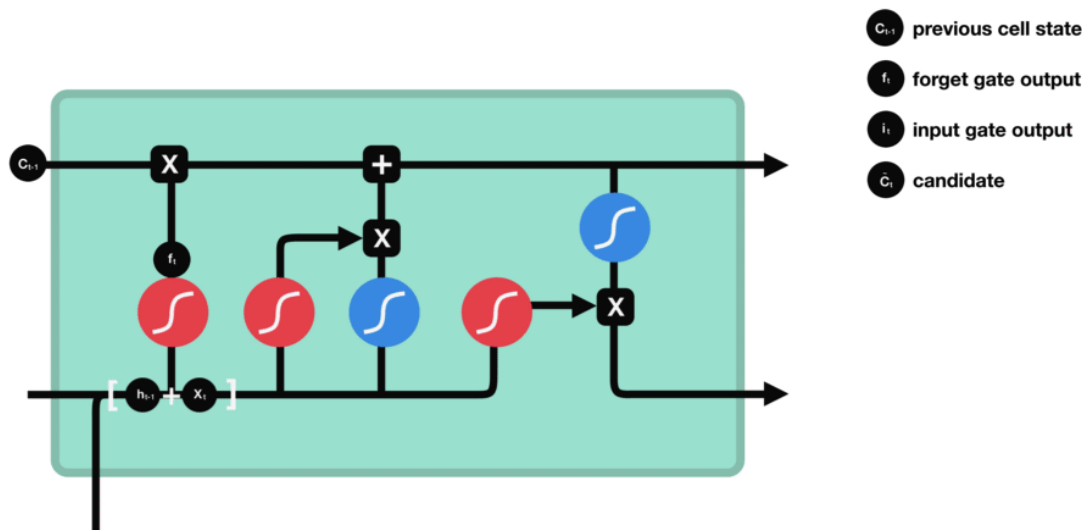


**Forget gate :** 전 단계에서 온 입력중 중요한것을 결정하고 keep함  
이전의 은닉 상태(hidden state) + 현재의 input data 가 합쳐져서 시그모이드 함수로 넘겨짐 출력이 0에 가까우면 안 중요하니까 잊어라, 1에 가까우면 중요하니 keep하라는 뜻

# LSTM 구조

## 입력 게이트

- input gate(입력 게이트)는 현재 정보를 기억
- 과거 정보와 현재 데이터의를 입력받아 **Sigmoid**와 **tanh**를 기반으로 현재 정보에 대한 보존량을 결정
- 계산한 값이 1이면 입력  $x$ 가 들어오도록 허용, 0이면 차단

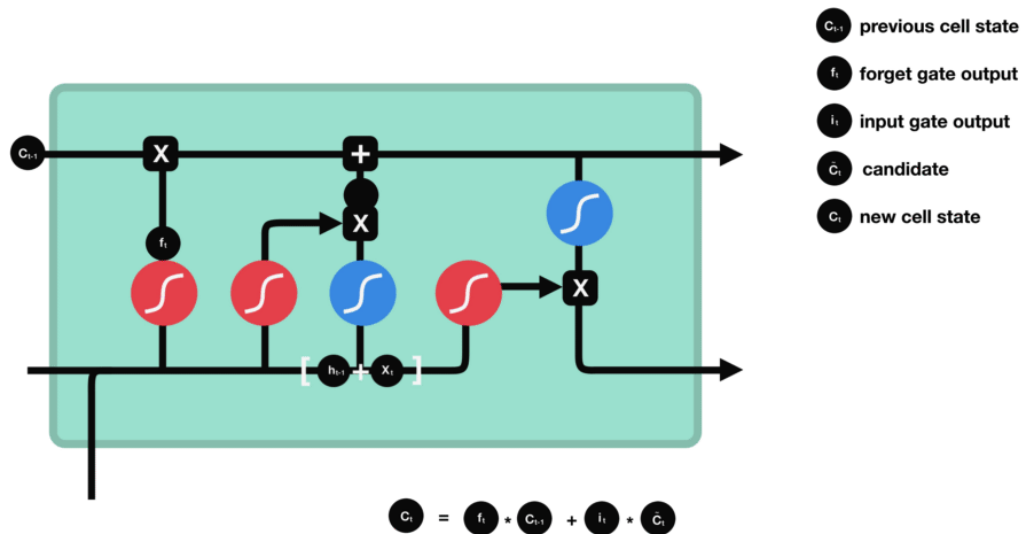


Input gate : 현재 단계에서 온 중요한게 뭔지 결정하고 add함  
- 1) [이전 은닉상태] + [현재 input data] 를 시그모이드 함수에 넘겨줌 : 잊을지 말지 결정  
- 2) [이전 은닉상태] + [현재 input data] 를 tanh 함수에 넘겨줌 : 출력을  $[-1,1]$  사이로 만들어 정규화  
- 1)과 2)를 곱하기  
- tanh에 모든 입력 정보가 있다면 시그모이드는 그 정보 중 중요한 것만 선별해줌 (안 중요하면 0으로 만들어 잊기)

# LSTM 구조

## 셀 상태

- 각 단계에 대한 hidden node(은닉 노드) 가 메모리 셀
- 총합(sum)을 사용하여 셀 값을 반영 하여 셀을 업데이트



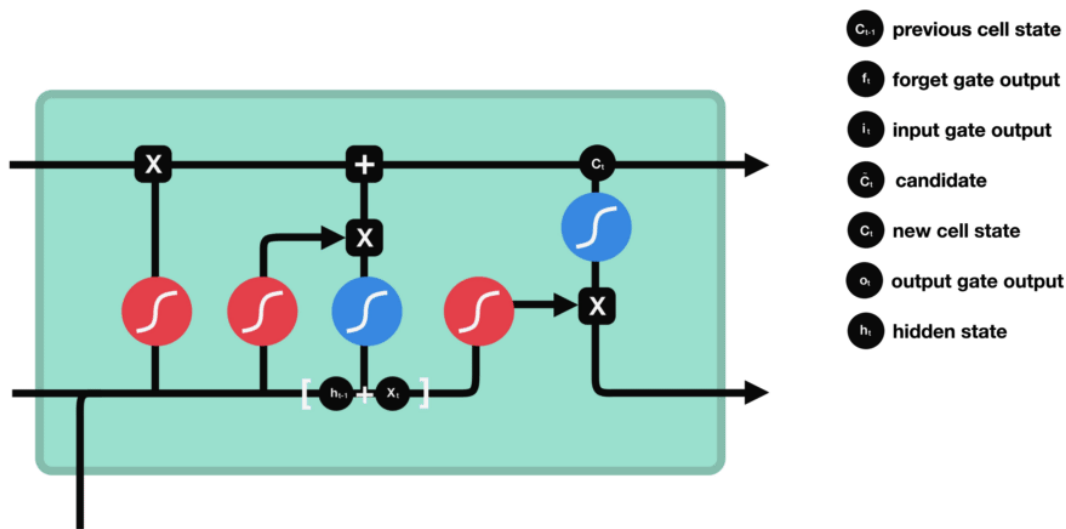
### Cell State 계산

기존 cell state \* [forget gate output] +  
[input gate output] = new cell state

# LSTM 구조

## 출력 게이트

- Output gate(출력 게이트)는 과거 정보와 현재 데이터를 사용하여 뉴런의 출력을 결정
- 이전 은닉상태와 t번째 입력을 고려해서 다음 은닉 상태를 계산
- 계산한 값이 1이면 의미 있는 결과로 최종 출력, 0이면 해당 연산 출력을 하지 않음



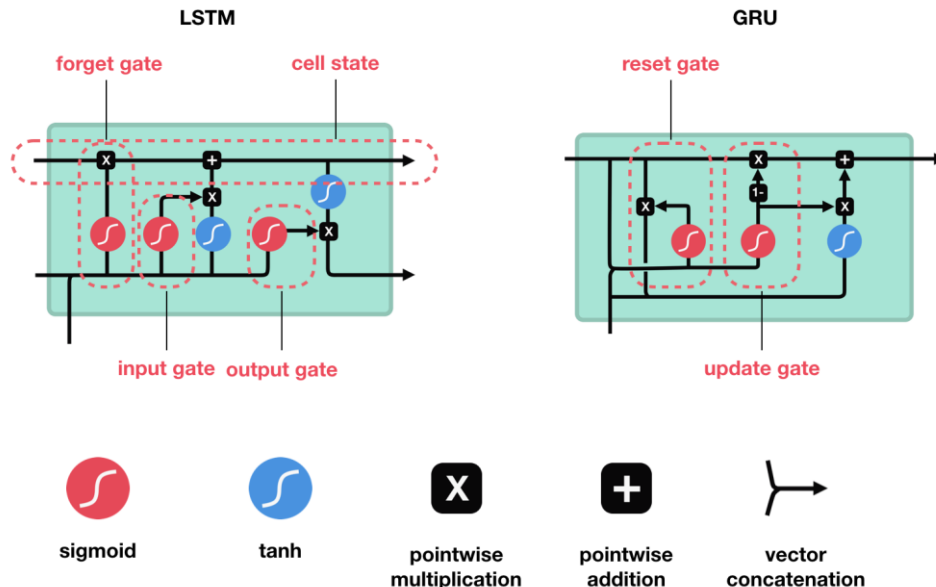
**Output gate : 새로운 은닉상태를 계산**

- 다음 은닉 계산
- 1) [이전 은닉상태] + [현재 input data] 를 시그모이드 함수에 넘겨줌
- 2) new cell state를 tanh 함수에 넘겨줌
- 1) 과 2)를 곱하여 새로운 은닉상태 만들기

# GRU (Gated Recurrent Unit)

## 게이트 순환 신경망(GRU) 구조

- GRU는 게이트 메커니즘이 적용된 RNN프레임 워크
- 하나의 게이트 컨트롤러가 망각 게이트와 입력 게이트를 제어
- 게이트 컨트롤러가 1을 출력 망각 게이트는 Open, 입력 게이트는 Close
- 게이트 컨트롤러가 0을 출력 망각 게이트는 Close, 입력 게이트는 Open
- 이전 기억이 저장될 때마다 단계별 입력은 삭제
- GRU는 출력 게이트가 없어 전체 상태 벡터가 매 단계마다 출력



### GRU

- LSTM의 cell state 대신 은닉 상태 사용
- 2개의 게이트 : Reset gate, Update gate

### Update gate

- LSTM의 Forget/Input gate와 유사함
- 어떤 정보를 버릴 지, 어떤 새 정보를 추가할지 결정

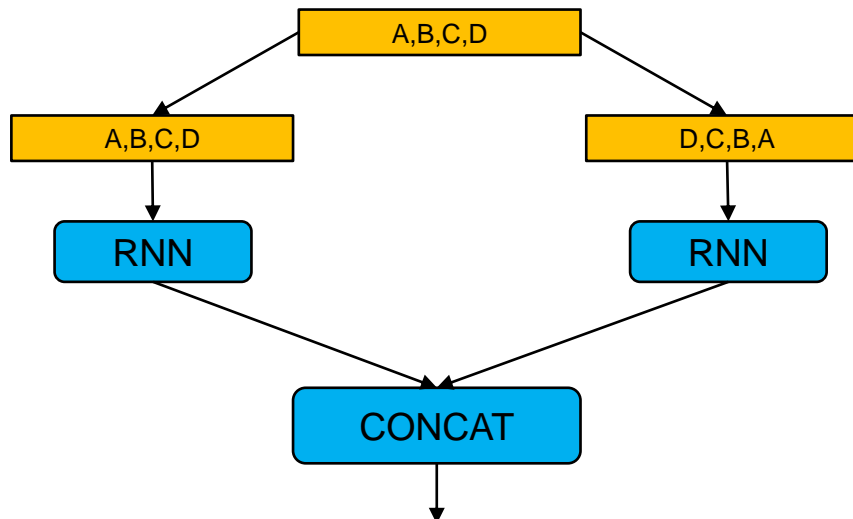
### Reset gate

- 과거 정보를 얼마나 버릴지 결정

# RNN Advanced

## 순환신경망 성능향상

- 순환 드롭아웃(recurrent dropout)
  - 순환 층에서 과대적합을 방지하기 위해 사용
- 스택킹 순환 층(stacking recurrent layer)
  - 네트워크의 표현 능력을 증가시키지만 계산 비용 많음
- 양방향 순환 층(bidirectional recurrent layer)
  - 순환 네트워크에 같은 정보를 다른 방향으로 주입하여 정확도를 높이고 기억을 오래 유지



# Keras 함수형 API

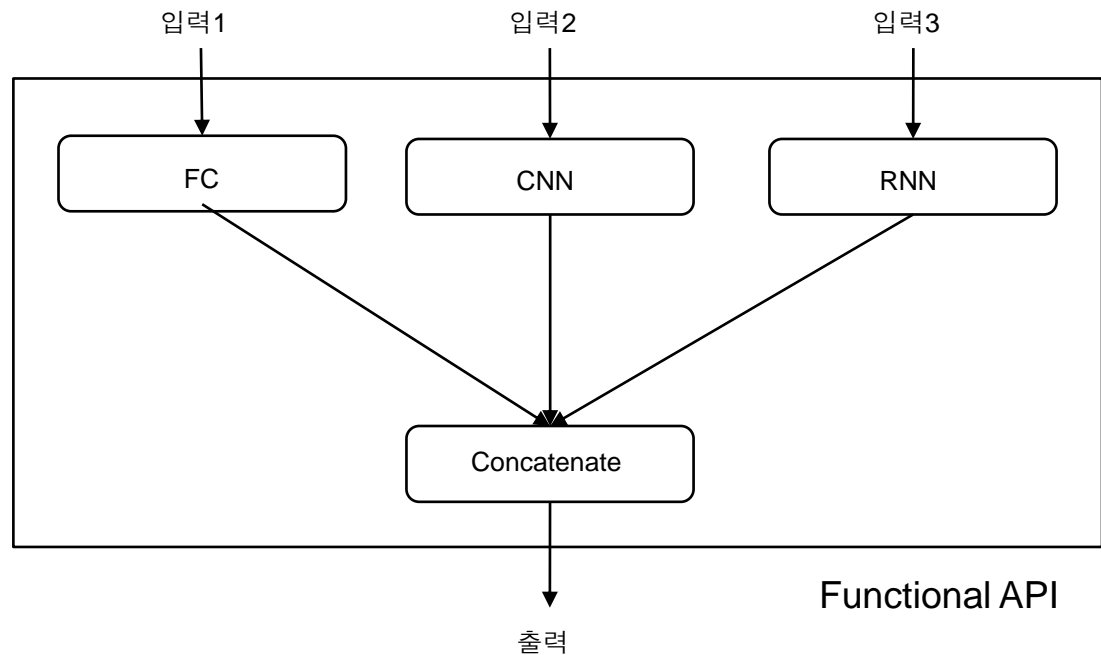
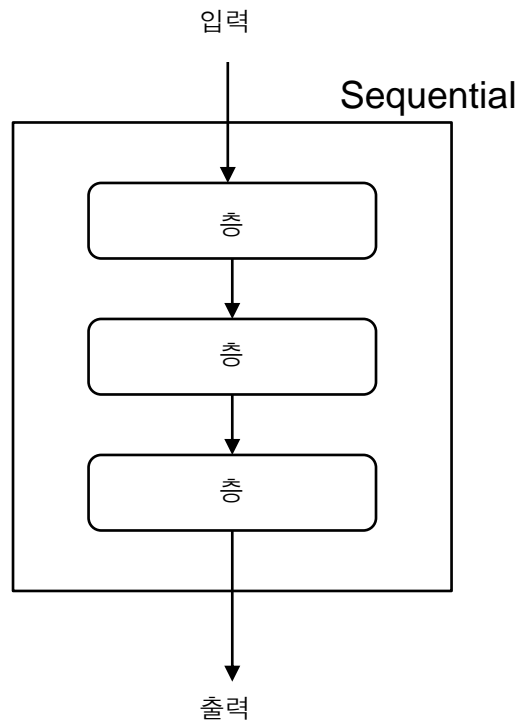
---



# 함수형 API

## Sequential모델과 함수형 API

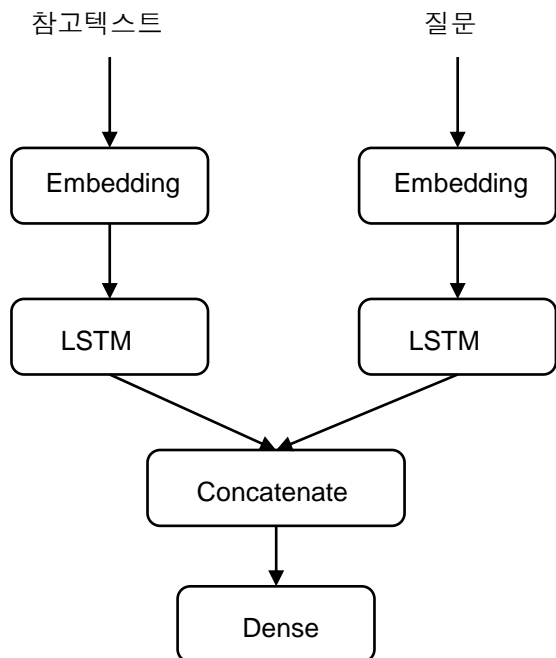
- Sequential 모델은 네트워크 입력과 출력이 하나
- 입력이 여러 개 필요하거나 출력이 여러 개 필요할 경우 함수형 API 사용



# 다중 입력 모델

## 다중 입력 모델

- 서로 다른 입력 가치를 합치기 위해 여러 텐서를 연결할 수 있는 층을 사용
- `keras.layers.add`, `keras.layers.concatenate` 등을 사용



```
from keras.models import Model
from keras import layers
from keras import Input
from keras.utils import plot_model

text_vocab_size = 10000
question_vocab_size = 10000
answer_vocab_size = 500

text_input = Input(shape=(None,), dtype="int32", name="text")
question_input = Input(shape=(None,), dtype="int32", name="question")
embedded_text = layers.Embedding(text_vocab_size, 64)(text_input)
embedded_question = layers.Embedding(question_vocab_size, 32)(question_input)

encoded_text = layers.LSTM(32)(embedded_text)
encoded_question = layers.LSTM(32)(embedded_question)

concated = layers.concatenate([encoded_text, encoded_question], axis=-1)

answer = layers.Dense(answer_vocab_size, activation="softmax")(concated)

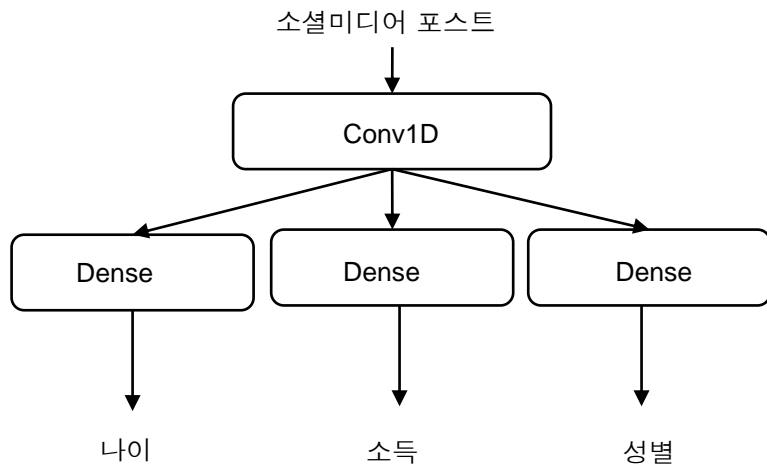
model = Model(inputs=[text_input, question_input], outputs=answer)

plot_model(model)
```

# 다중 입력 모델

## 다중 출력 모델

- 여러 속성을 동시에 예측하는 모델에 필요
- 다중 출력 모델을 훈련하기 위해서는 네트워크 출력마다 다른 손실 함수 사용



```
post_vocab_size = 50000
num_income_groups = 10
posts_input = Input(shape=(None,),dtype="int32",name="inputs")
embedded_posts = layers.Embedding(post_vocab_size,256)(posts_input)
x = layers.Conv1D(128,5,activation="relu")(embedded_posts)
x = layers.MaxPooling1D(5)(x)
x = layers.Conv1D(256,5,activation="relu")(x)
x = layers.Conv1D(256,5,activation="relu")(x)
x = layers.GlobalMaxPooling1D()(x)
x = layers.Dense(128,activation="relu")(x)

age_predict = layers.Dense(1,name="age")(x)
income_predict =
layers.Dense(num_income_groups,activation="softmax",name="income")(x)
gender_predict = layers.Dense(1,activation="sigmoid")(x)

model = Model(inputs=posts_input,
              outputs=[age_predict,income_predict,gender_predict])
plot_model(model)
```

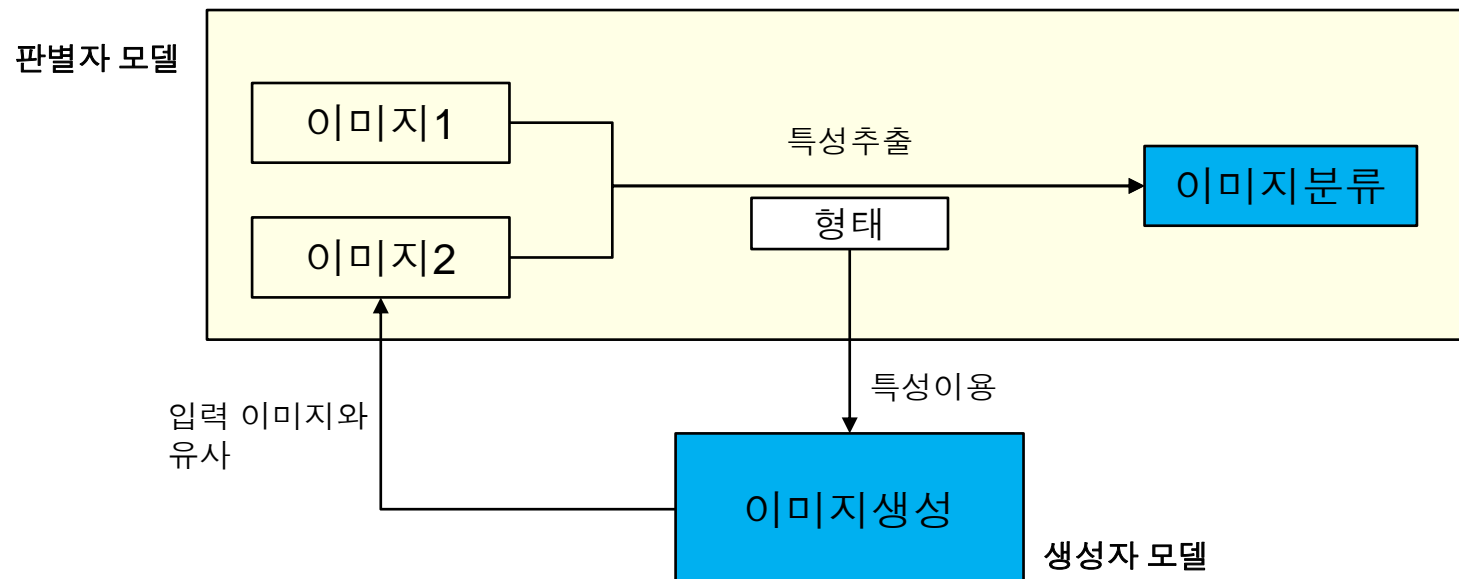
# Generative Model

---

# Generative Model

## 생성 모델 개념

- 생성 모델은 주어진 데이터를 학습하여 데이터 분포를 따르는 유사한 데이터를 생성
- 판별자 모델(Discriminative Model)은 convolution 신경망에서 이미지를 분류하는 모델
- 판별자 모델은 이미지를 대표하는 특성을 잘 찾는 것을 목표로 구성
- 생성자 모델(Generative Model)은 판별자 모델에서 추출한 특성들의 조합을 이용하여 새로운 이미지를 생성
- 생성 모델은 입력 이미지에 대한 데이터 분포를 학습하여 새로운 이미지를 생성

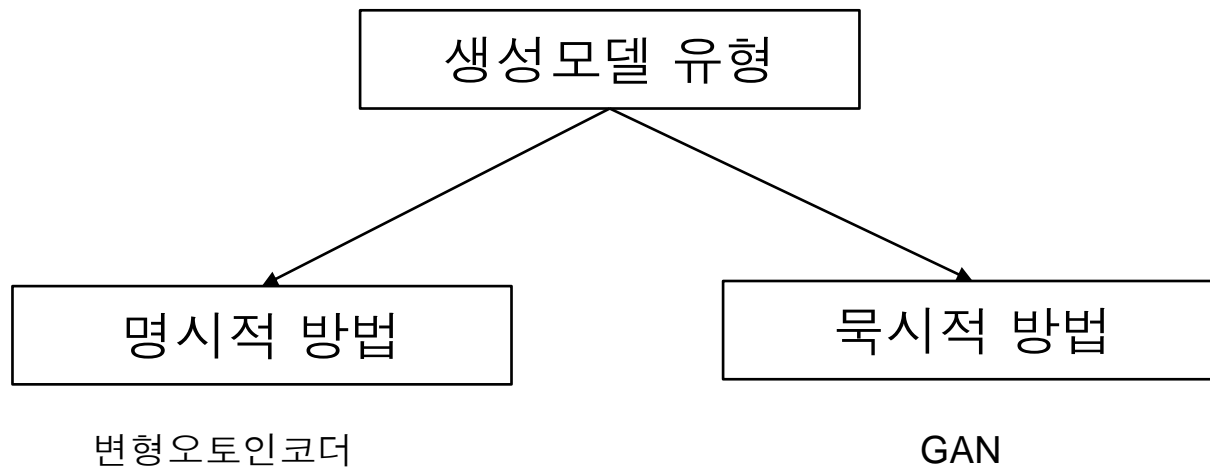


# 생성 모델 유형

---

## 생성 모델 유형

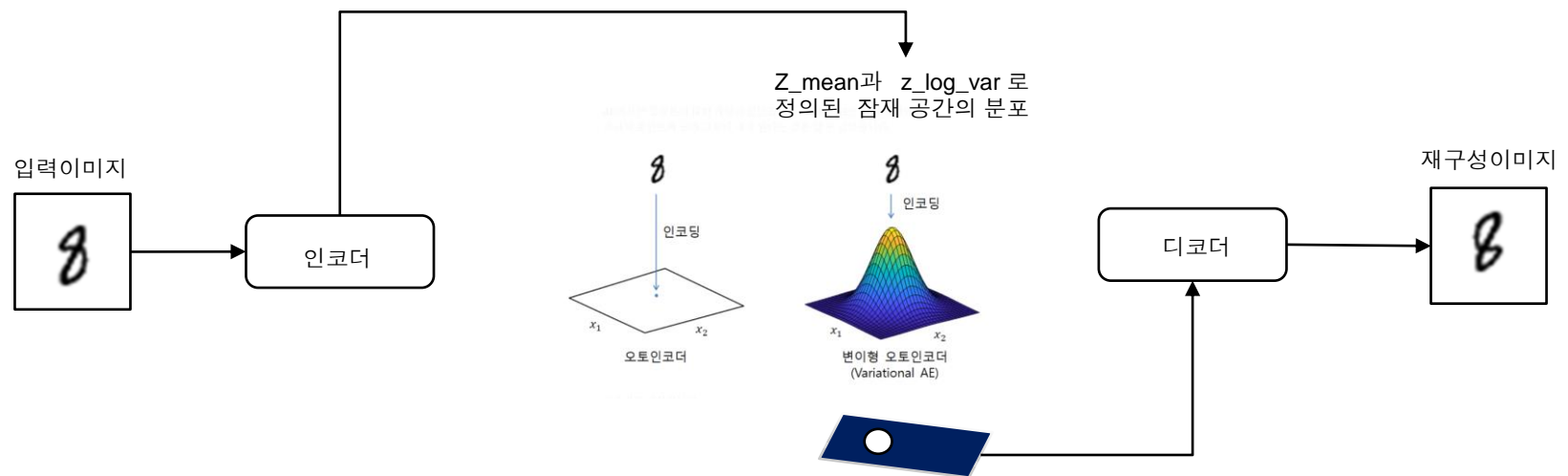
- 생성 모델의 유형은 명시적 방법과 묵시적 방법으로 구분
- 명시적 방법은 확률변수  $P(x)$ 를 정의하여 사용
- 명시적 방법의 대표적 모델 변형 오토인코더(Variational Autoencoder)
- 묵시적 방법의 대표적 모델 GAN(Generative Adversarial Network)



# VAE(변이형 오토 인코더)

## Variational AutoEncoder

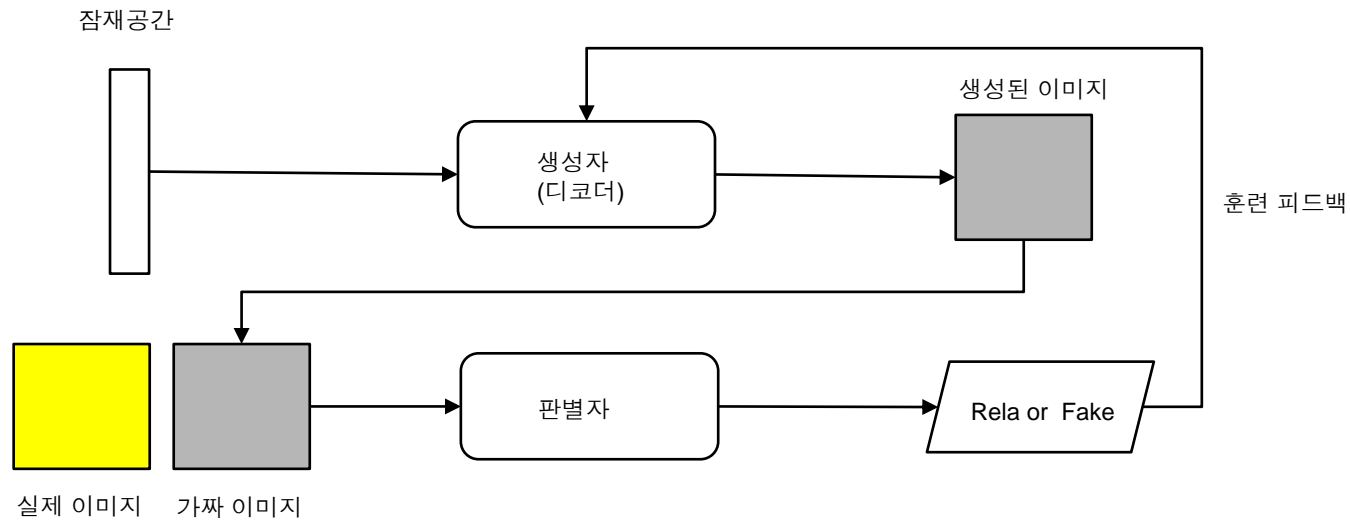
- 이미지 생성의 핵심아이디어는 각 포인트가 실제와 같은 이미지로 매핑할 수 있는 저차원 잠재공간의 표현을 만드는 것
- VAE는 이미지를 어떤 통계 분포의 파라미터로 변환
  - 평균과 분산 파라미터를 사용하여 이 분포에서 무작위로 하나의 샘플을 추출
  - 샘플을 디코딩하여 원본 입력으로 복원



# GAN

## 적대적 생성 신경망

- 생성된 이미지와 실제 이미지가 통계적으로 거의 구분이 되지 않도록 강제하여 실제 이미지 생성
- 위조범(생성자) 네트워크와 전문가(판별자) 네트워크를 경쟁 관계로 훈련
  - **Generator Network** : 랜덤벡터(잠재공간의 무작위한 포인터)를 입력 받아 이를 합성한 이미지로 디코딩
  - **Discriminator Network** : 이미지(실제나 합성한)를 입력 받아 훈련 세트에서 온 이미지인지 생성자에서 온 이미지 인지 판별





---

```
from keras.models import Model
from keras import layers
from keras import Input
from keras.utils import plot_model

text_vocab_size = 10000
question_vocab_size = 10000
answer_vocab_size = 500

text_input = Input(shape=(None,), dtype="int32", name="text")
question_input = Input(shape=(None,), dtype="int32", name="question")
embedded_text = layers.Embedding(text_vocab_size, 64)(text_input)
embedded_question = layers.Embedding(question_vocab_size, 32)(question_input)

encoded_text = layers.LSTM(32)(embedded_text)
encoded_question = layers.LSTM(32)(embedded_question)

concated = layers.concatenate([encoded_text, encoded_question], axis=-1)

answer = layers.Dense(answer_vocab_size, activation="softmax")(concated)

model = Model(inputs=[text_input, question_input], outputs=answer)

plot_model(model)
```