BITAmin 12기&13기 방학 3차세션

딥러닝 입문

5조
문유진 송규헌 이예린 홍성민

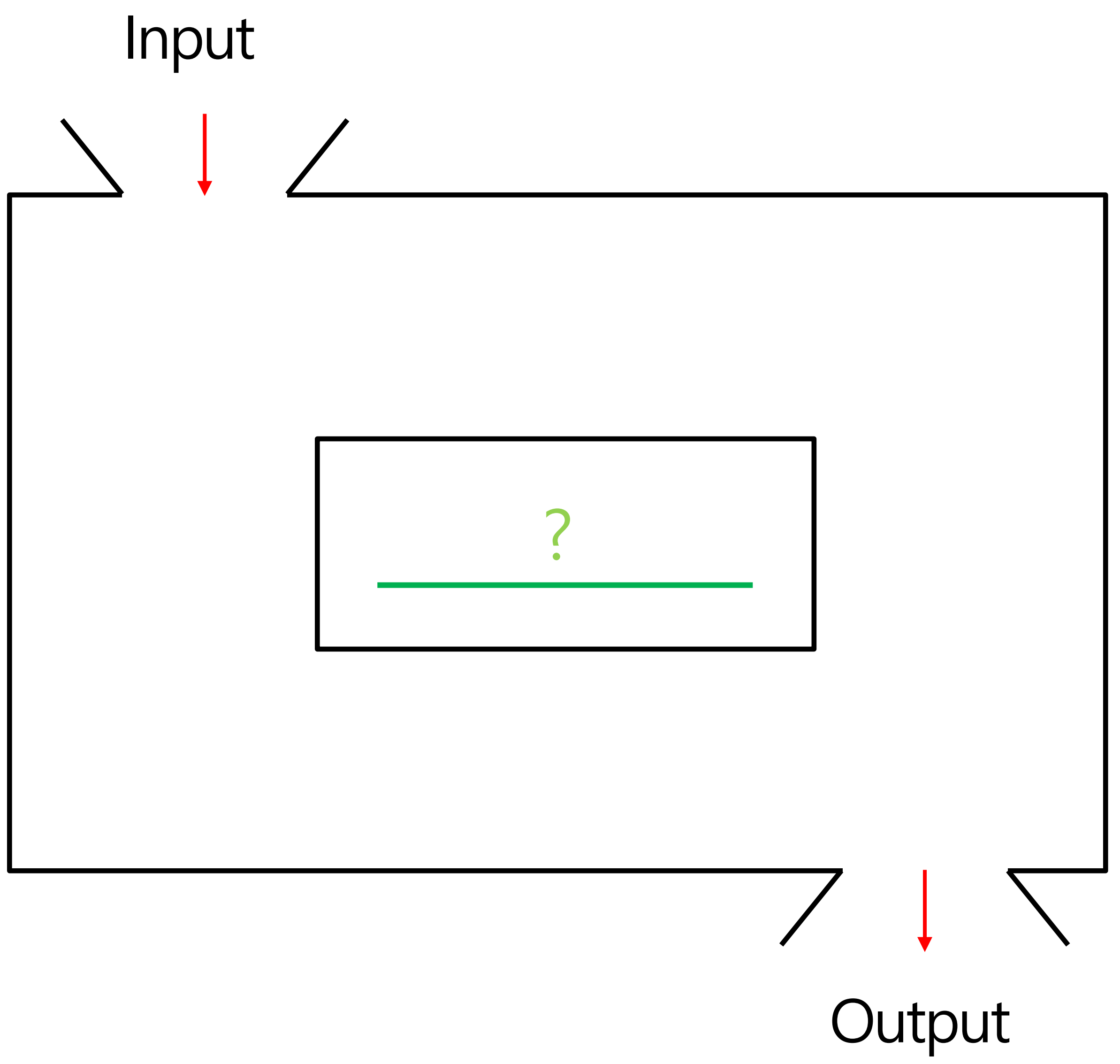
ML & DL , Perceptron

Machine Learning & Deep Learning

송규헌

머신러닝 개요

- 머신러닝이란?
  - 입력 데이터가 주어졌을 때 답을 유추해 줄 수 있는 최적의 _?_ 를 기계가 찾는 것



머신러닝 개요

- 머신러닝의 종류

- 기호주의

Input

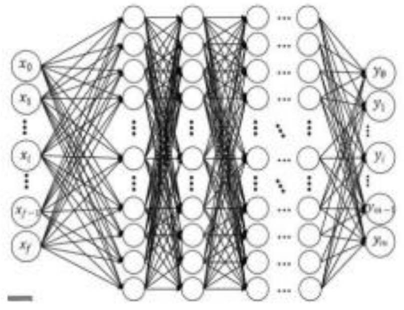
결정 트리 등

Output

- 연결주의

Input

신경망/딥러닝



Output

- 확률주의

Input

베이지안 통계

Output

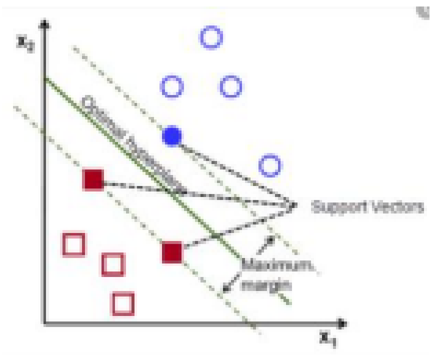
- 유전 알고리즘

Input

Output

- 유추주의

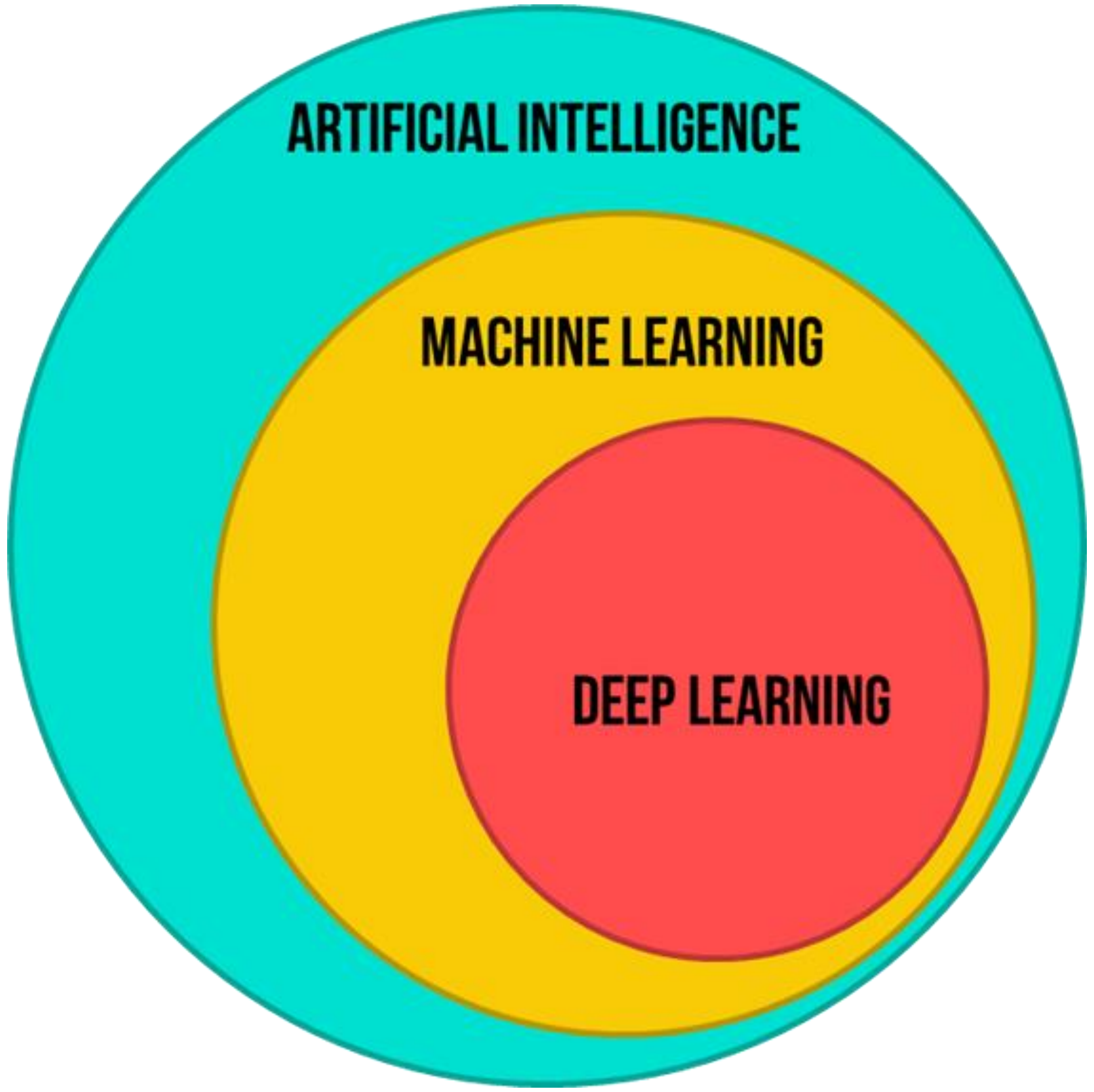
Input

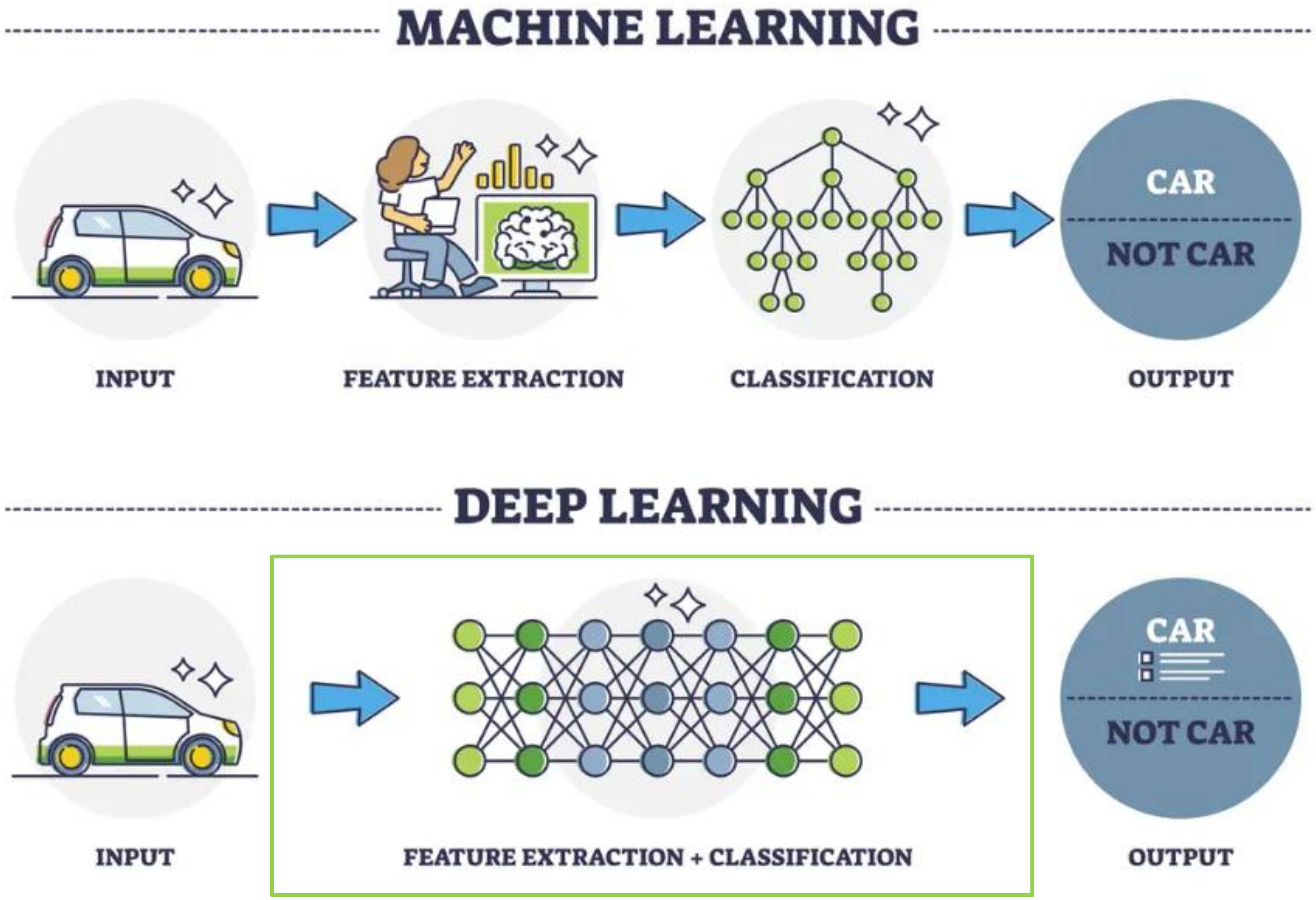


Output

머신러닝 개요

- 머신러닝과 딥러닝





???-to-??? 학습

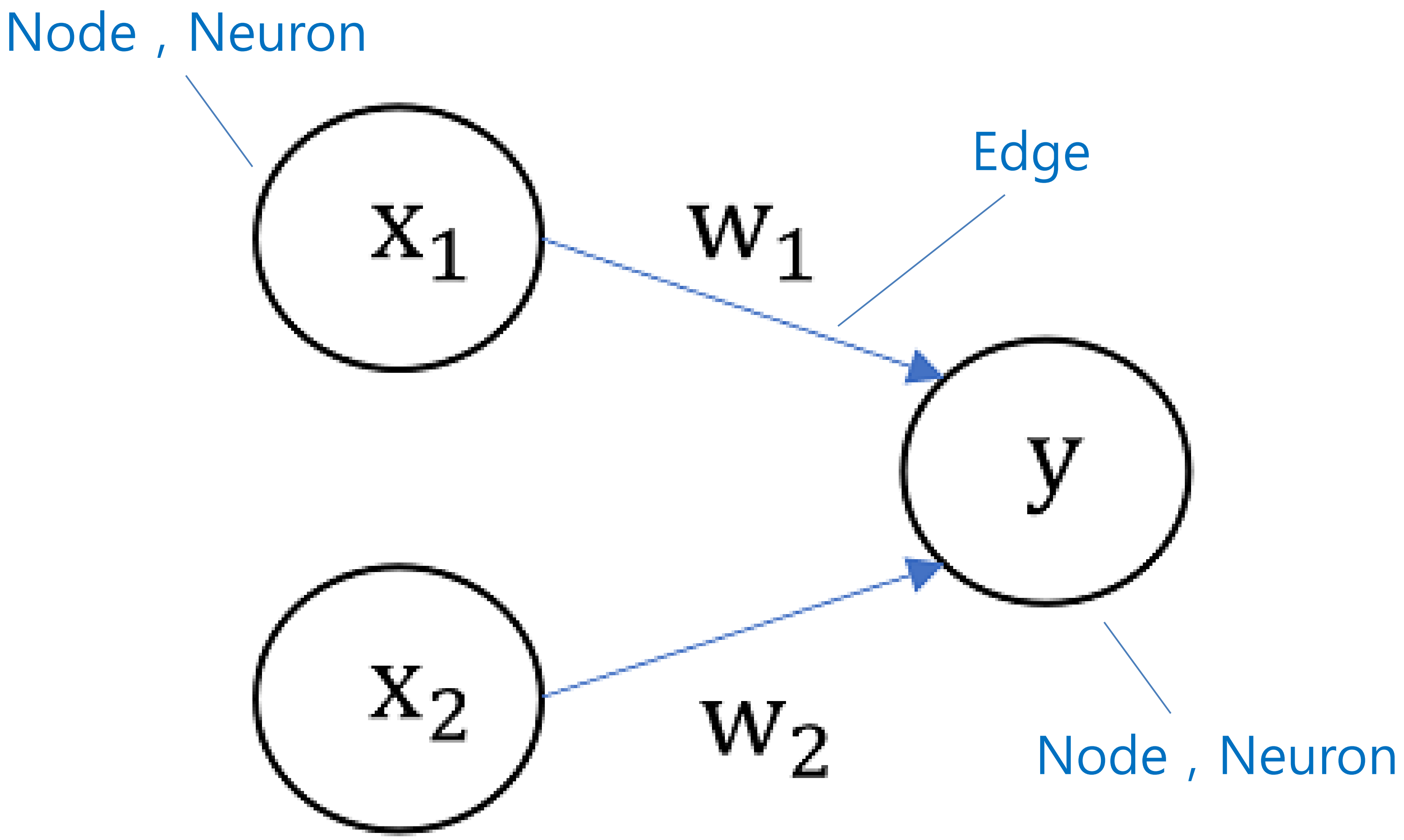
ML & DL , Perceptron

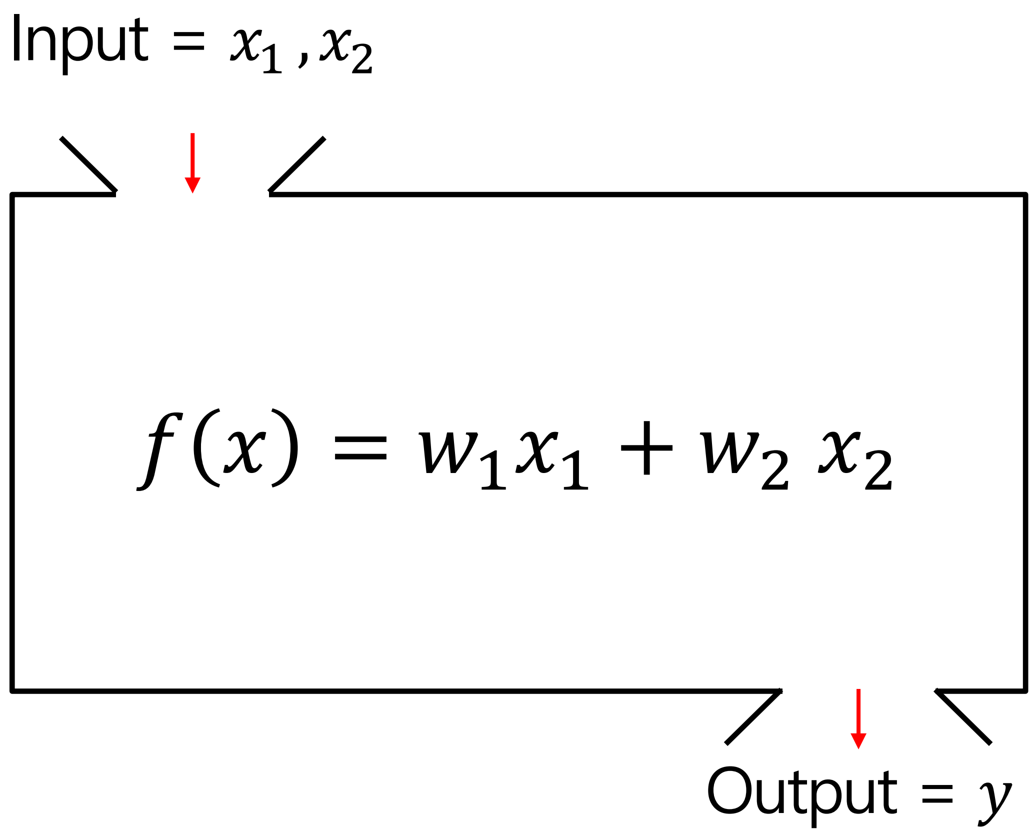
Perceptron

송규헌

Perceptron

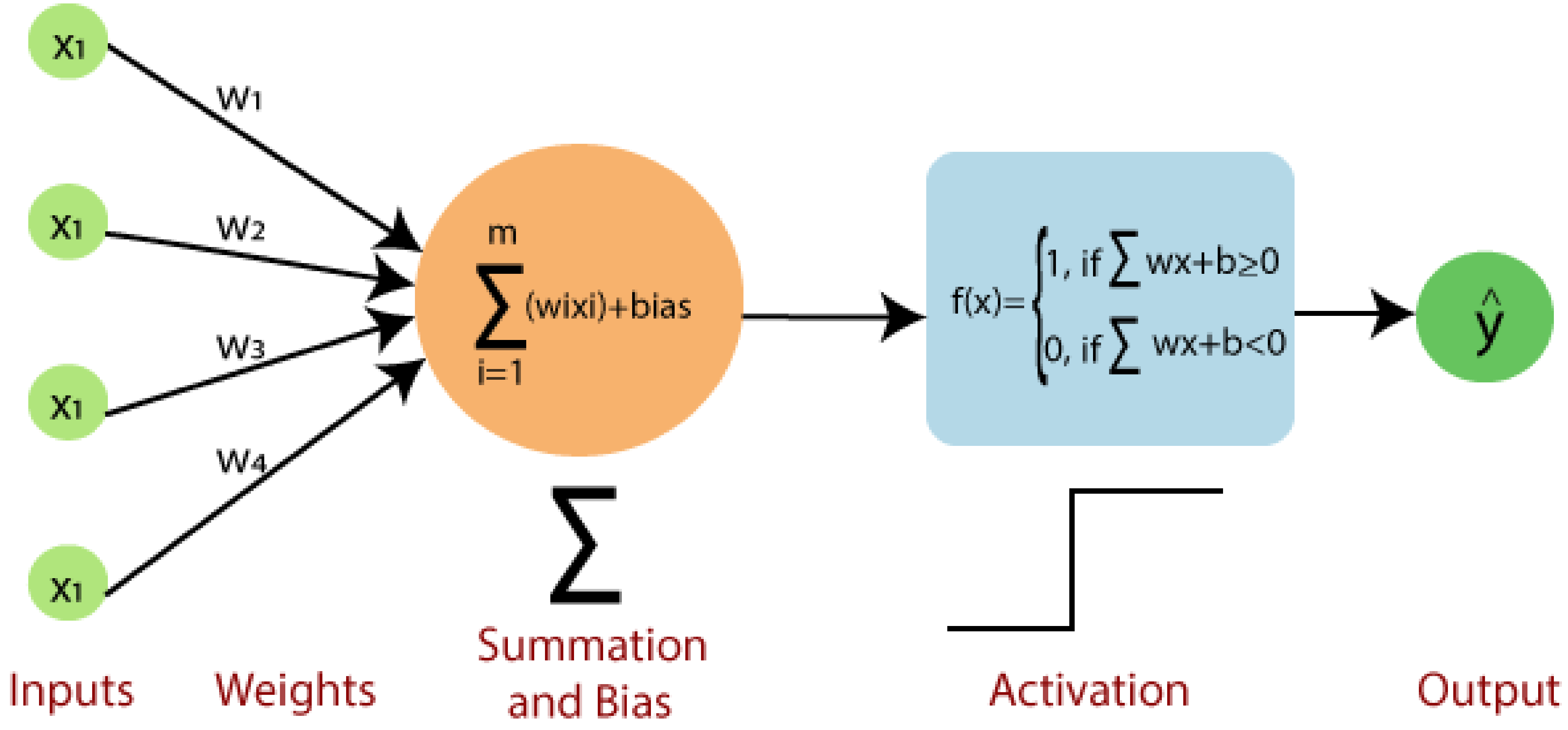
- Perceptron?
  - 다수의 신호를 입력받아 하나의 신호를 출력하는 알고리즘
  - 가장 단순한 형태의 신경망(Nueral Network) > Single Layer Perceptron





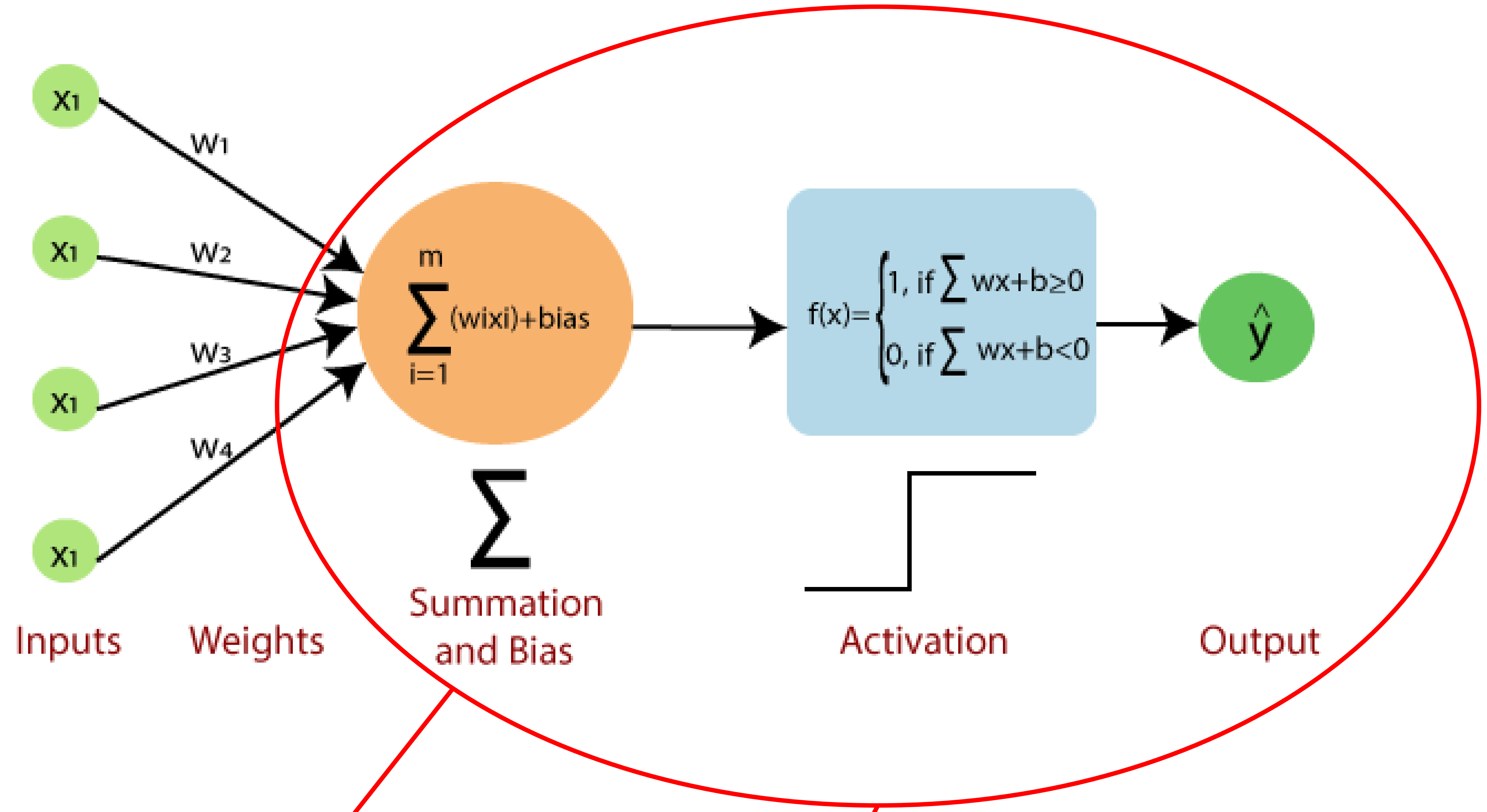
Perceptron

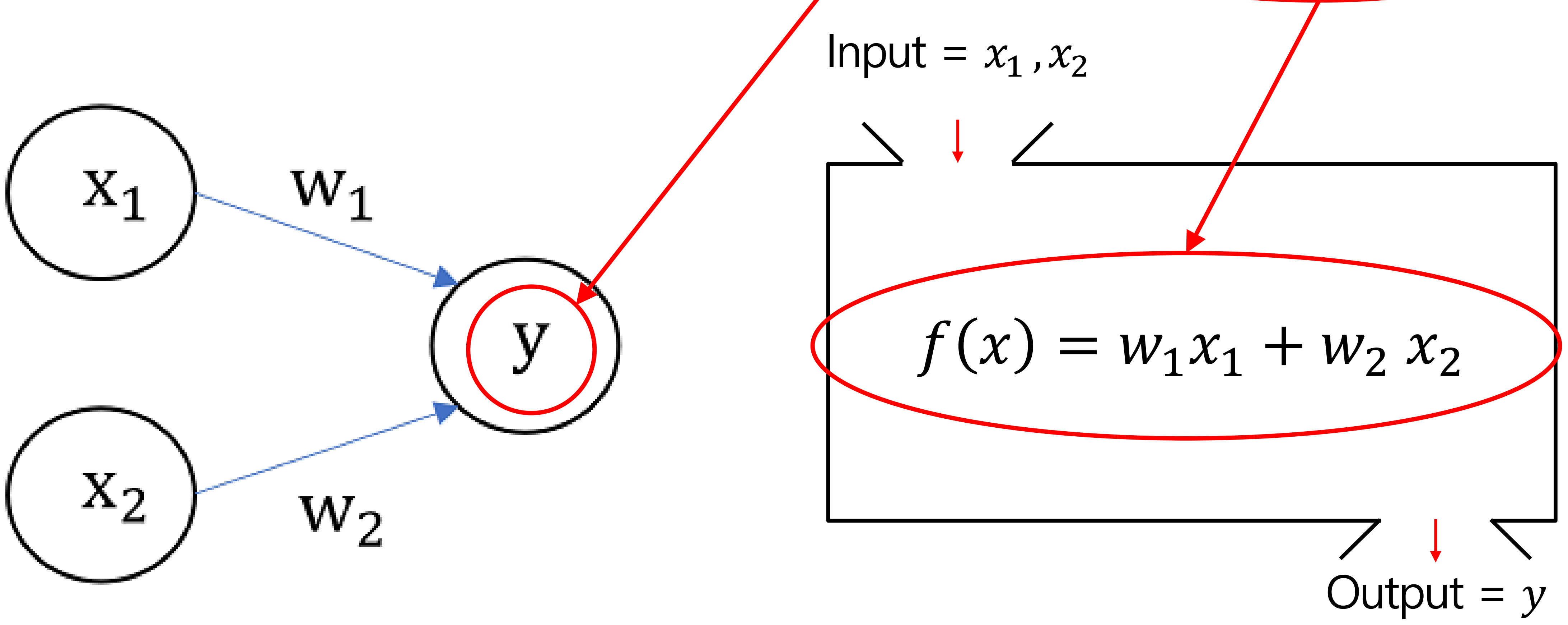
- Components of Perceptron
  - Inputs : 입력 신호
  - Weights : 가중치
  - Threshold & Bias : 편향
  - Activation Function : 활성화 함수
  - Output : 출력 신호



Perceptron

- Components of Perceptron
  - Inputs : 입력 신호
  - Weights : 가중치
  - Threshold & Bias : 편향
  - Activation Function : 활성화 함수
  - Output : 출력 신호





ML & DL , Perceptron

논리문제와 Perceptron

송규헌

논리문제

- 진리표

- AND

X1	X2	Y
0	0	0
0	1	0
1	0	0
1	1	1

- OR

X1	X2	Y
0	0	0
0	1	1
1	0	1
1	1	1

- NAND

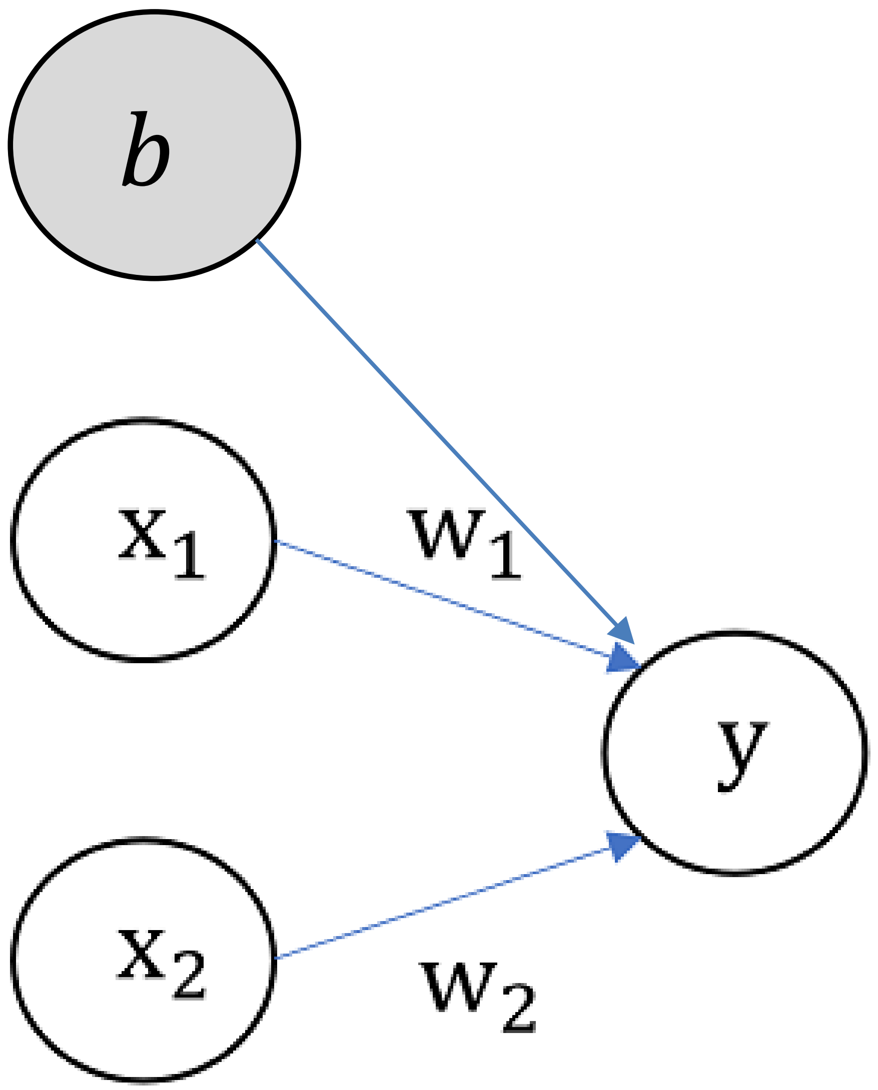
X1	X2	Y
0	0	1
0	1	1
1	0	1
1	1	0

- NOR

X1	X2	Y
0	0	1
0	1	0
1	0	0
1	1	0

AND 게이트

- SLP(Single Layer Perceptron)

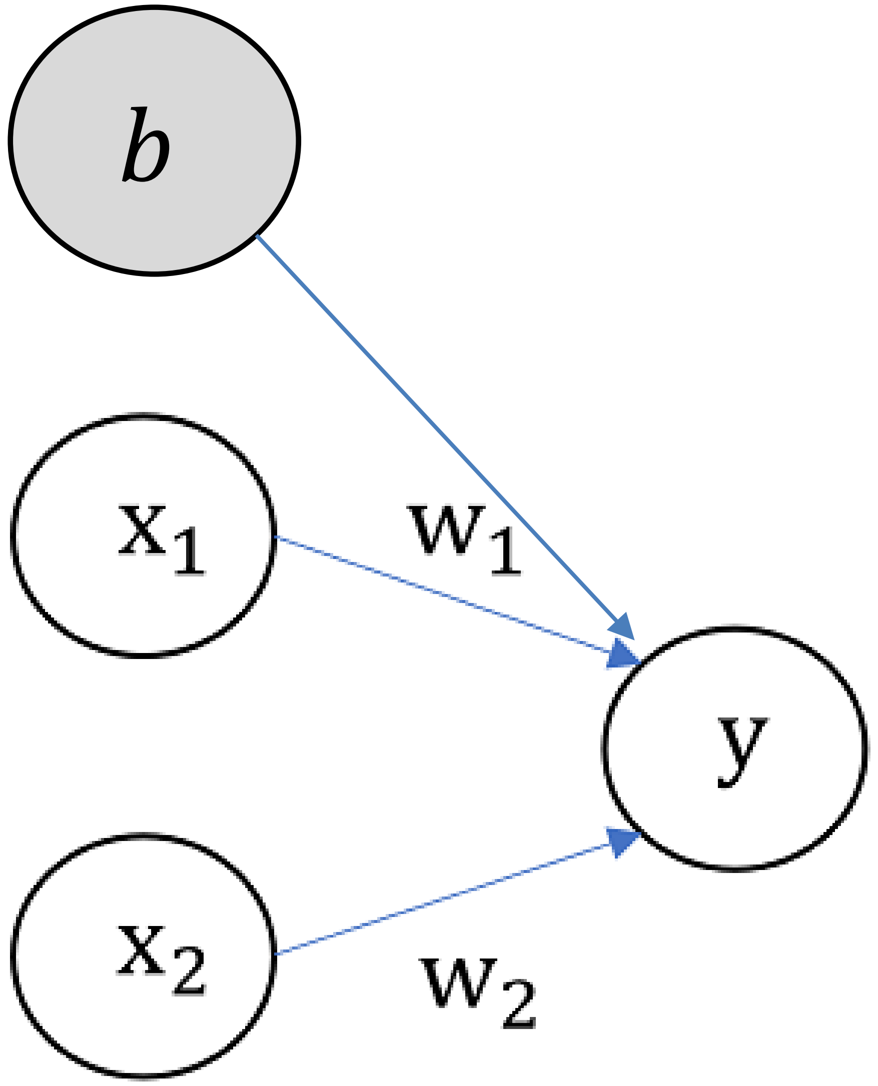


$$y = \begin{cases} 1 \ (w_1 x_1 + w_2 x_2 + b > 0) \\ 0 \ (w_1 x_1 + w_2 x_2 + b \le 0) \end{cases}$$

```
import numpy as np
def AND(x1,x2):
    x = np.array([x1,x2])
    w = np.array([0.5,0.5])
    b = -0.7
    y = np.sum(w*x) + b
    if y > 0: return 1;
    else: return 0;
cases = [[0,0],[0,1],[1,0],[1,1]]
for c in cases:
    x1,x2 = c
    result = AND(x1,x2)
    print(f'{x1} AND {x2} -> {result}')
```

AND 게이트

- SLP(Single Layer Perceptron)



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

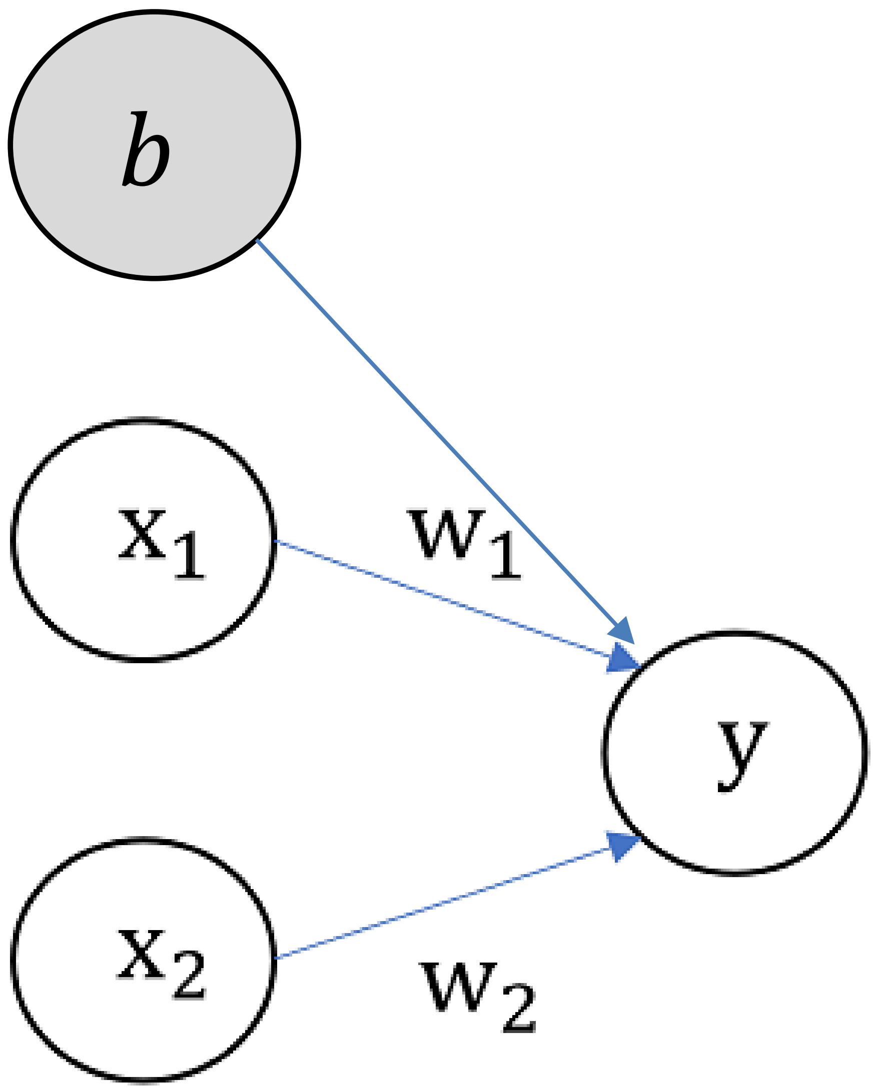
```
import numpy as np
def AND(x1,x2):
    x = np.array([x1,x2])
    w = np.array([0.5,0.5])
    b = -0.7
    y = np.sum(w*x) + b
    if y > 0: return 1;
    else: return 0;
cases = [[0,0],[0,1],[1,0],[1,1]]
for c in cases:
    x1,x2 = c
    result = AND(x1,x2)
    print(f'{x1} AND {x2} -> {result}')
```

Weighted Sum and Bias

Activation function (step function)

AND 게이트

- SLP(Single Layer Perceptron)



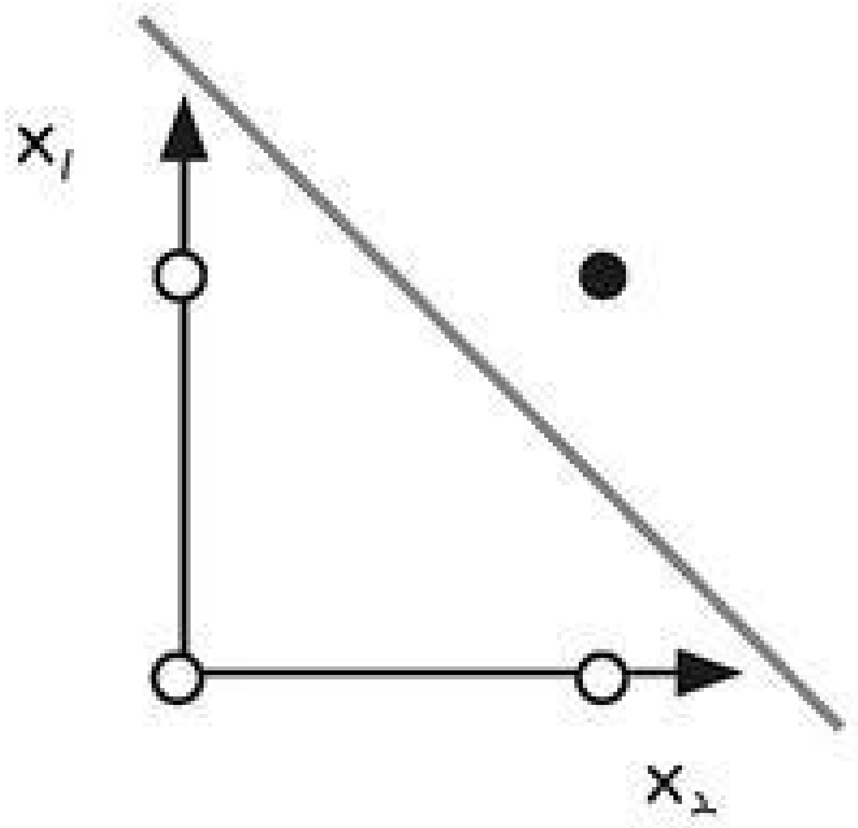
$$y = \begin{cases} 1 & (w_1x_1 + w_2x_2 + b > 0) \\ 0 & (w_1x_1 + w_2x_2 + b \leq 0) \end{cases}$$

```
import numpy as np
def AND(x1,x2):
    x = np.array([x1,x2])
    w = np.array([???,???])
    b = ???
    y = np.sum(w*x) + b
    if y > 0: return 1;
    else: return 0;
cases = [[0,0],[0,1],[1,0],[1,1]]
for c in cases:
    x1,x2 = c
    result = AND(x1,x2)
    print(f'{x1} AND {x2} -> {result}')
```

Learn Optimal

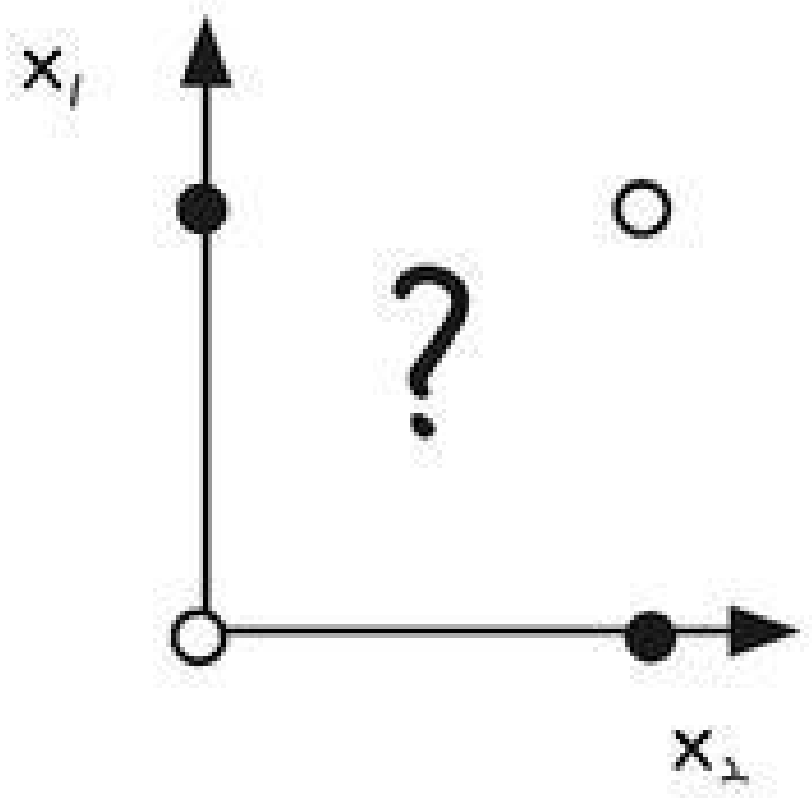
weights and bias

Limitation of SLP



- AND gate

X1	X2	Y
0	0	0
0	1	0
1	0	0
1	1	1



- XOR gate

X1	X2	Y
0	0	0
0	1	1
1	0	1
1	1	0

XOR, MLP

XOR 게이트와 MLP

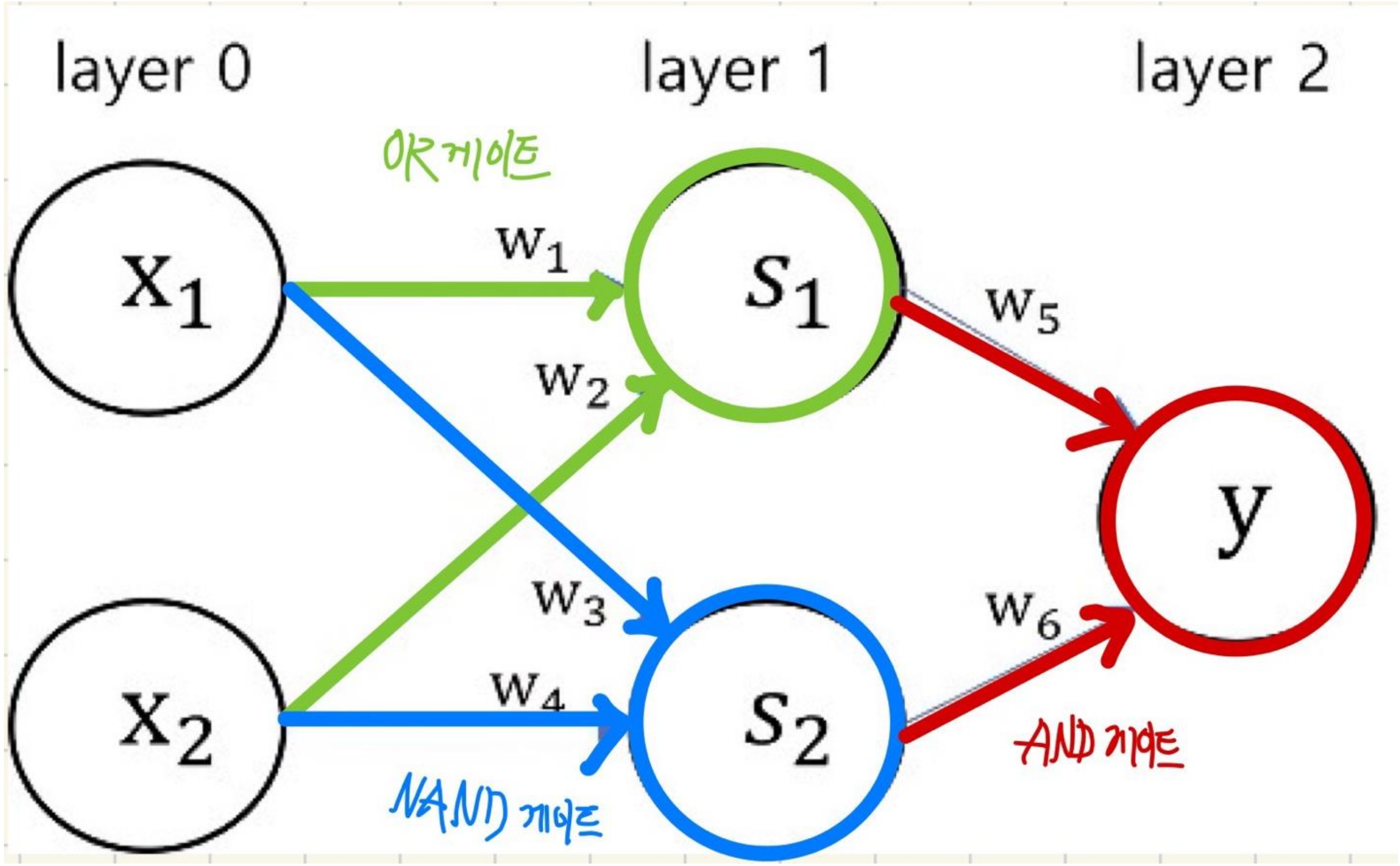
송규헌

XOR 게이트

- XOR 진리표

X1	X2	Y
0	0	0
0	1	1
1	0	1
1	1	0

- 도식화




```
import numpy as np
def AND(x1,x2):
    x = np.array([x1,x2])
    w = np.array([0.5,0.5])
    b = -0.7
    y = np.sum(w*x) + b
    if y > 0: return 1;
    else: return 0;
def OR(x1,x2):
    x = np.array([x1,x2])
    w = np.array([0.5,0.5])
    b = -0.2
    y = np.sum(w*x) + b
    if y > 0: return 1;
    else: return 0;
def NAND(x1,x2):
    x = np.array([x1,x2])
    w = np.array([-0.5,-0.5])
    b = 0.7
    y = np.sum(w*x) + b
    if y > 0: return 1;
    else: return 0;
                                layer  1
def XOR(x1,x2):
    s1 = NAND(x1,x2)
    s2 = OR(x1,x2)
    y = AND(s1,s2)              layer  2
    return y

cases = [[0,0],[0,1],[1,0],[1,1]]
for c in cases:
    x1,x2 = c
    result = XOR(x1,x2)
    print(f'{x1} XOR {x2} -> {result}')
```

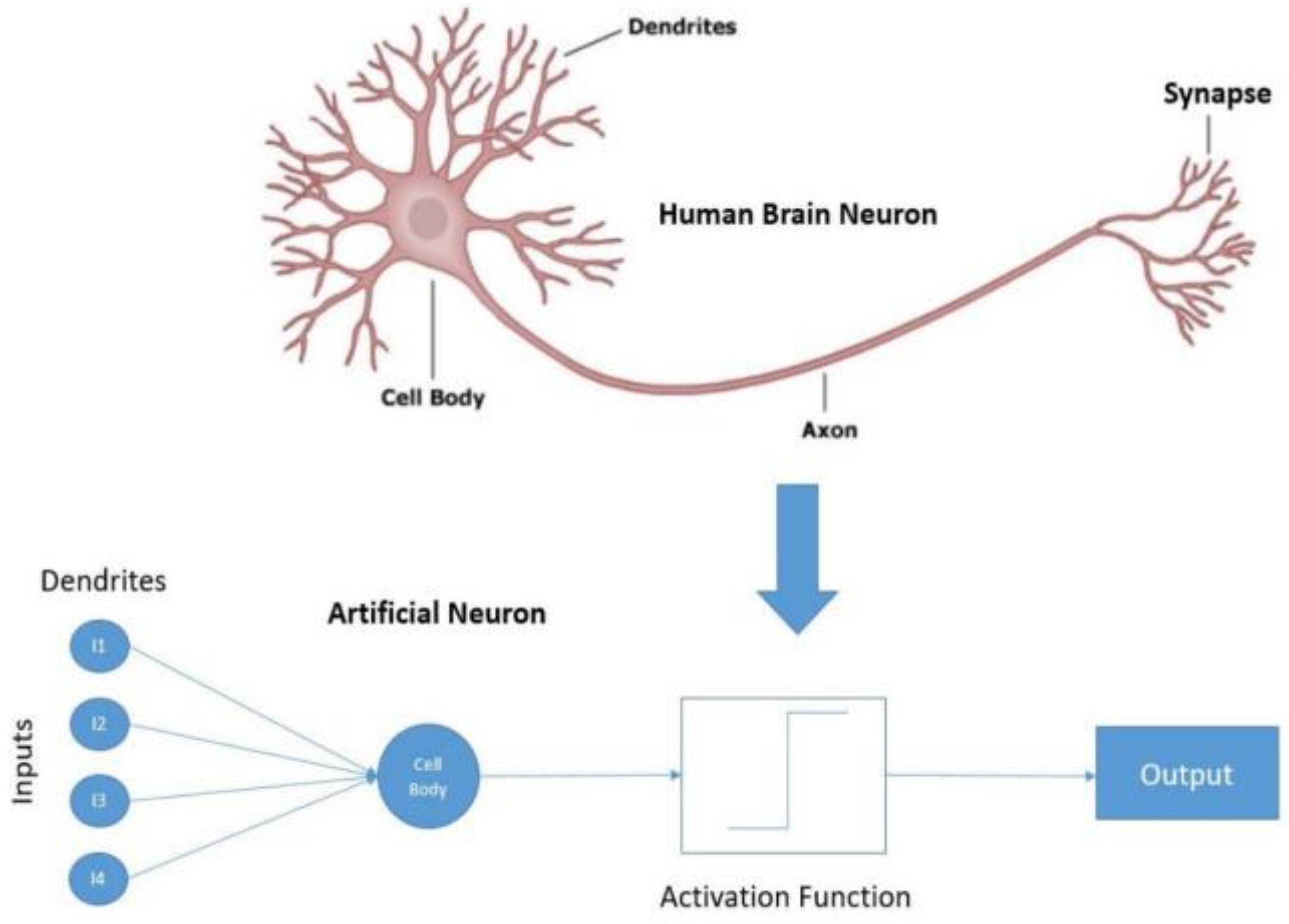
XOR , MLP

MLP : Multi-Layer Perceptron

송규헌

MLP

- 신경망 : 인간의 뇌 구조(neuron과 synapse)를 본딴 구조 = MLP, FFNN(Feed Forward Neural Network)



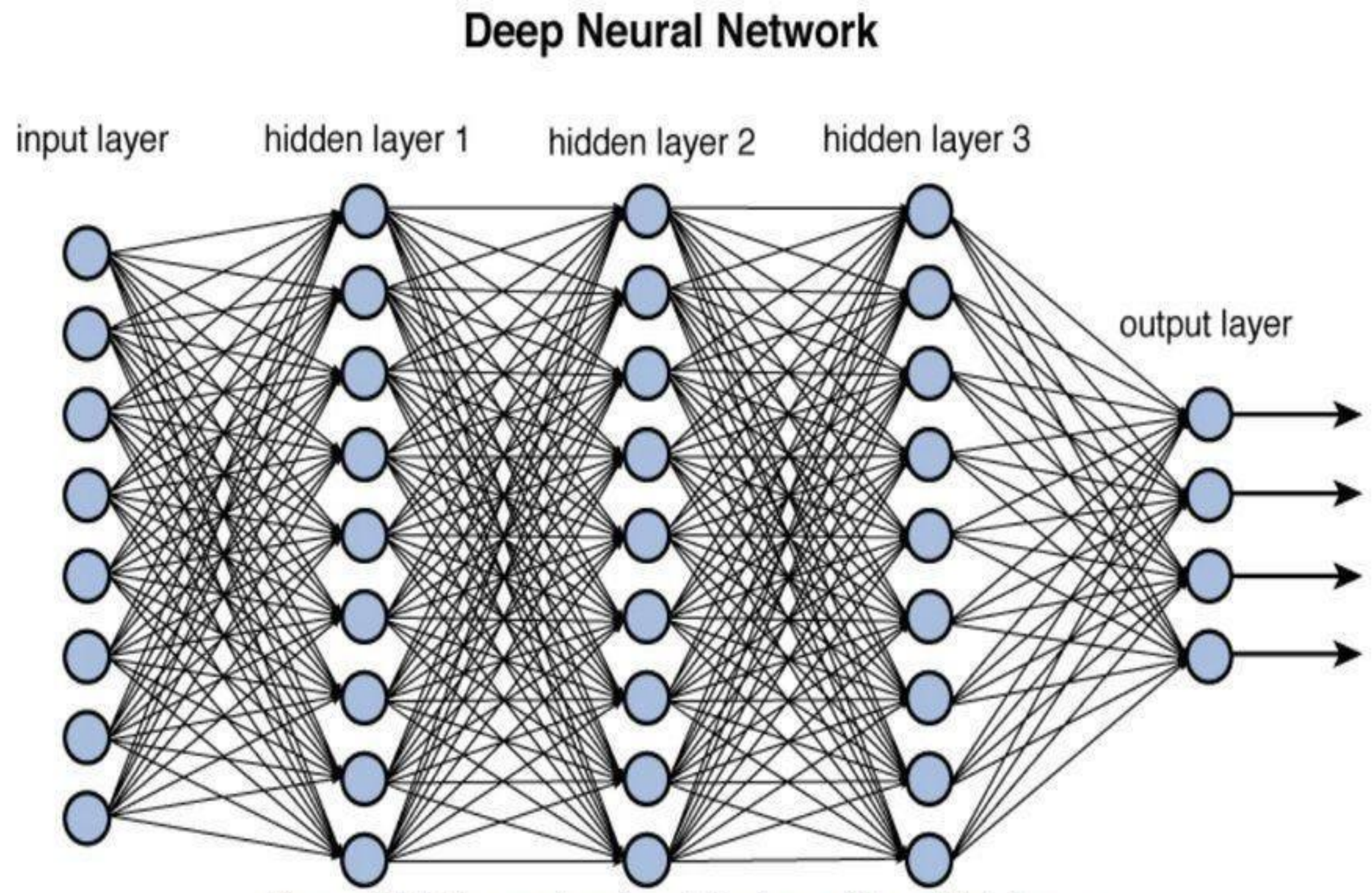
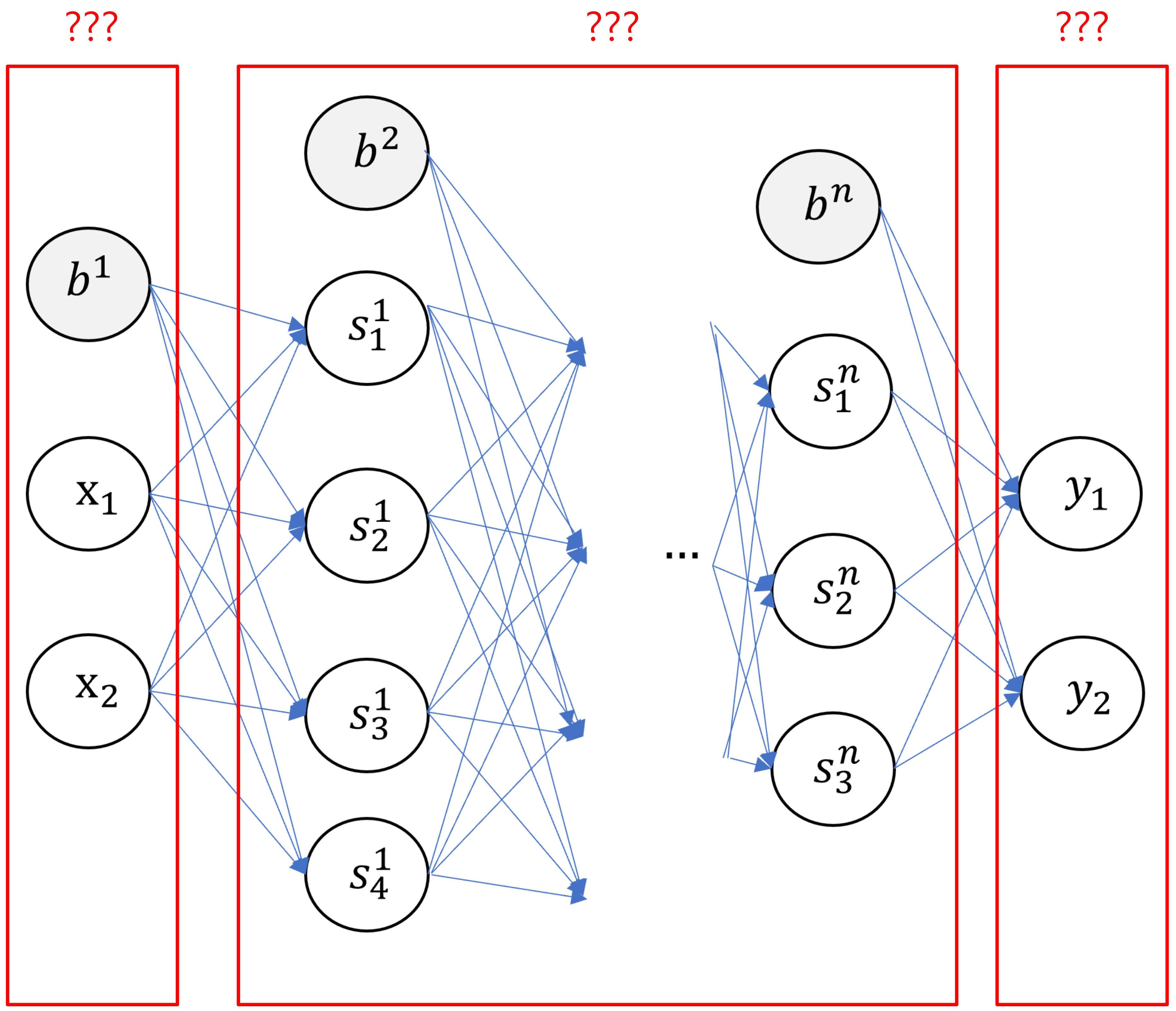


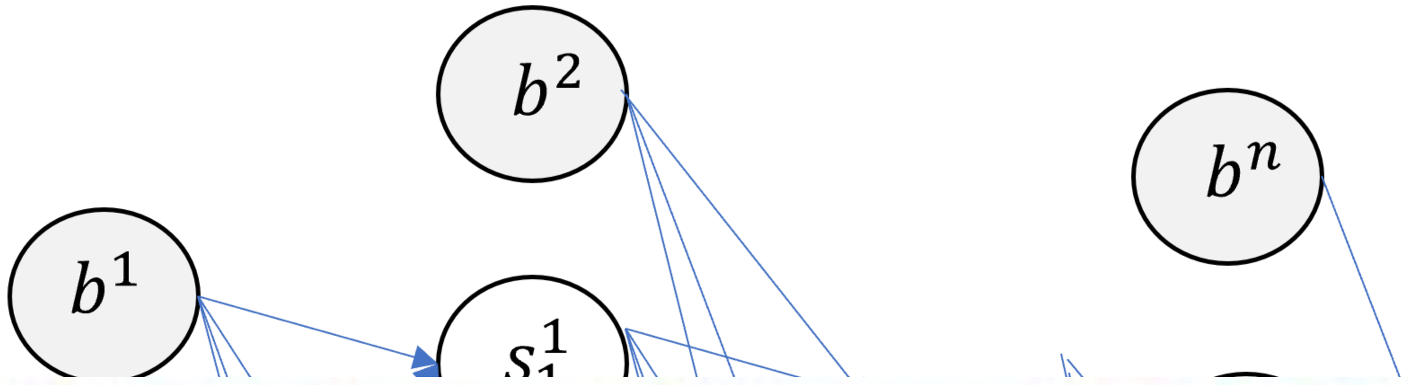
Figure 12.2 Deep network architecture with multiple layers.

MLP

- Components of MLP, FFNN, NN
  - Input layer
  - Bias
  - Hidden layer
    - └ Weights
    - └ Activation Function
  - Output layer

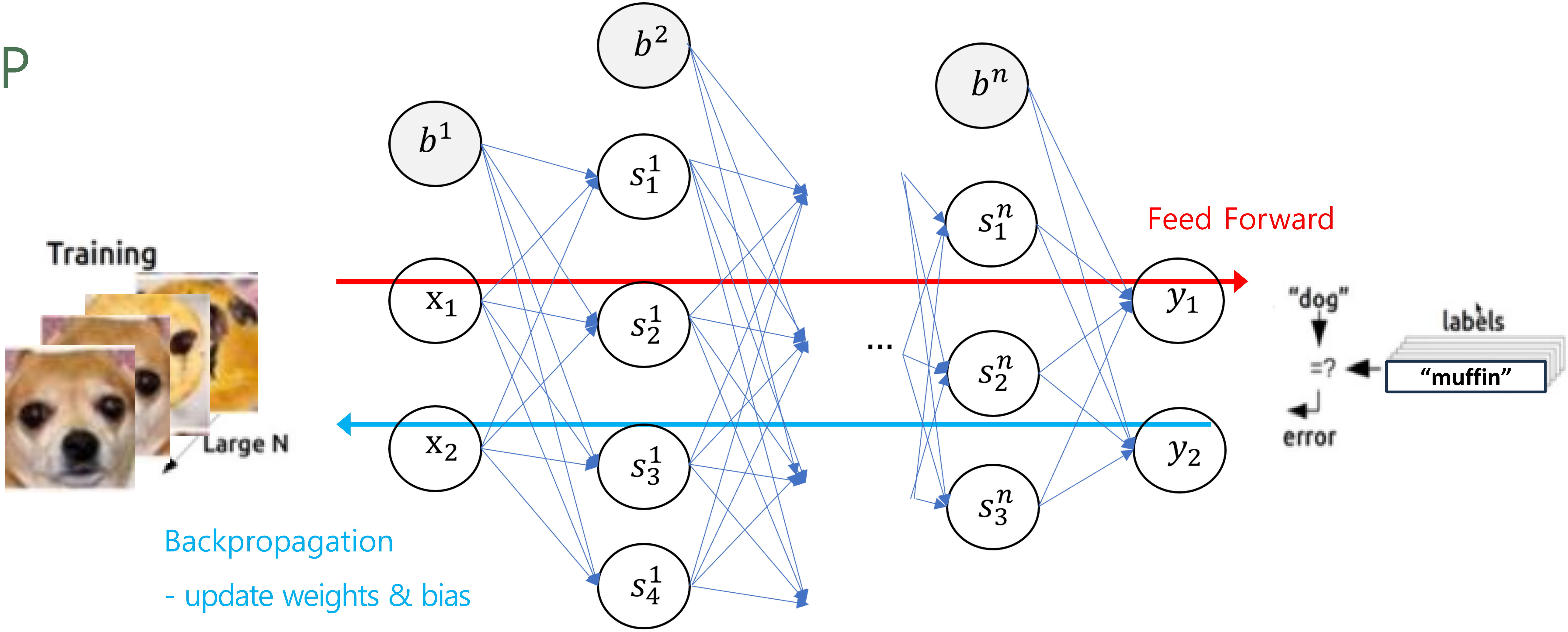


MLP



└ Backpropagation

MLP



- Train
  - └ Feed Forward
  - └ Backpropagation

iterate for the number of epochs

Gradient Descent

홍성민

◆ Multi-layer Perceptron (MLP)

- Gradient descent-based training
  - Weights (w) update

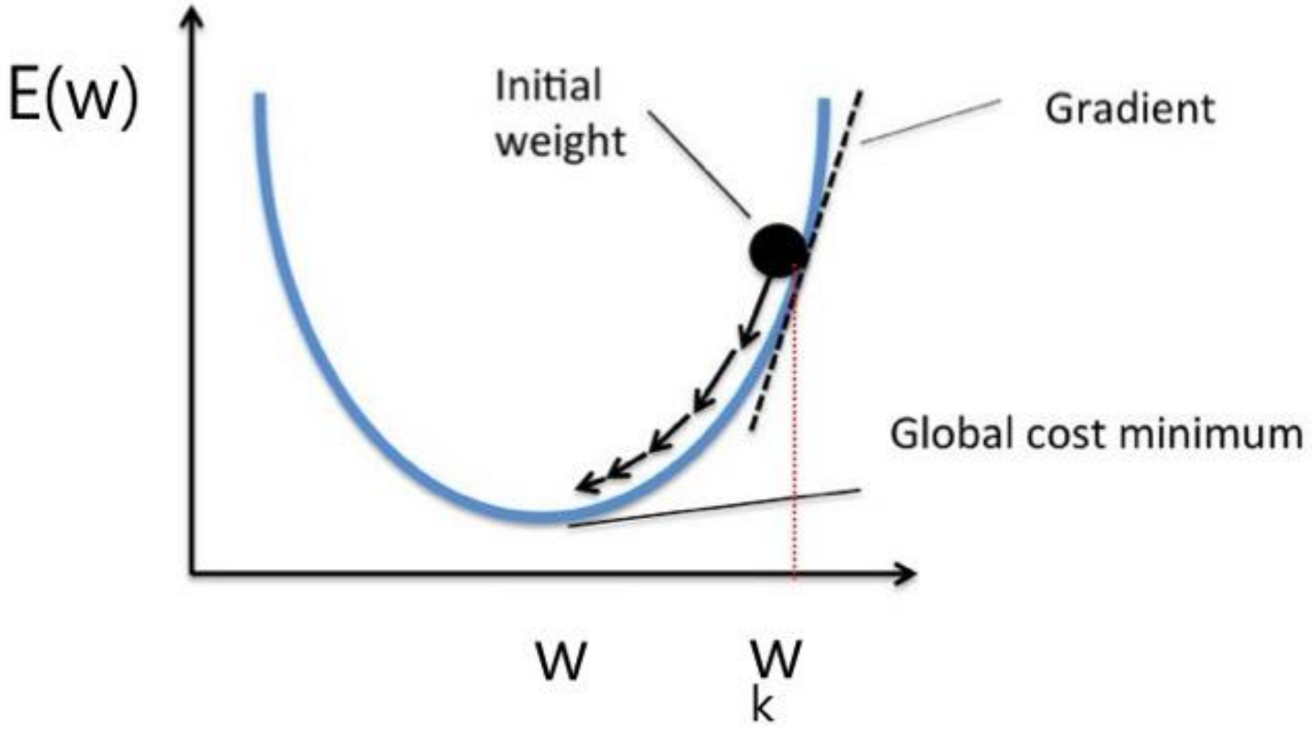
$$w \leftarrow w - \eta \nabla E(w)$$

Cost Function:
$E(W) = (t-y)^2/2$

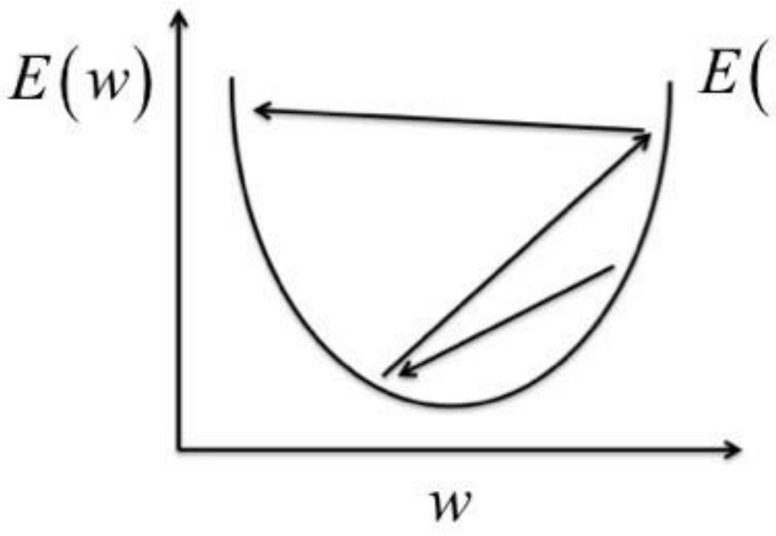
t : Ground-truth
y: Estimated Data

------- Linear approximation
: $f(w^k)+\nabla f(w^k)^T(w-w^k)$

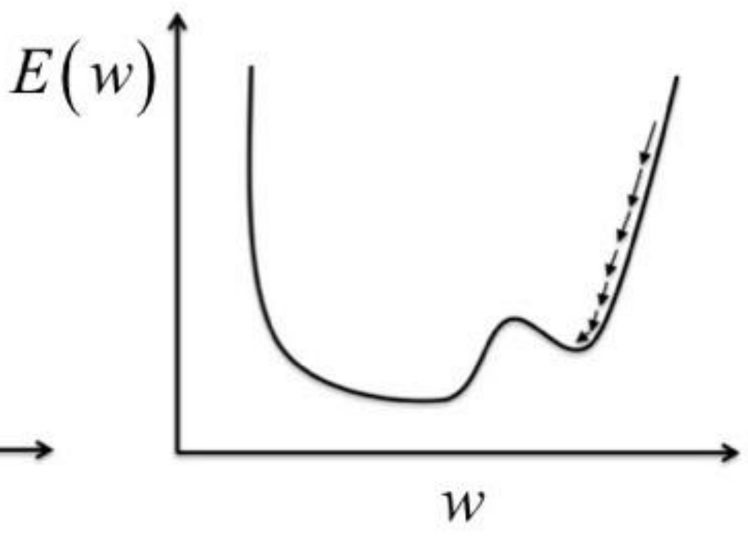
→ min $[f(w^k)+\nabla f(w^k)^T(w-w^k) + 1/2a \, \| w-w^k \|_2^2]$



  - Learning rate, η



Large η
: Fast but overshooting



Small η
: More stable but slow

◆ Gradient Descent

$$E = \frac{1}{2}(y-t)^2$$

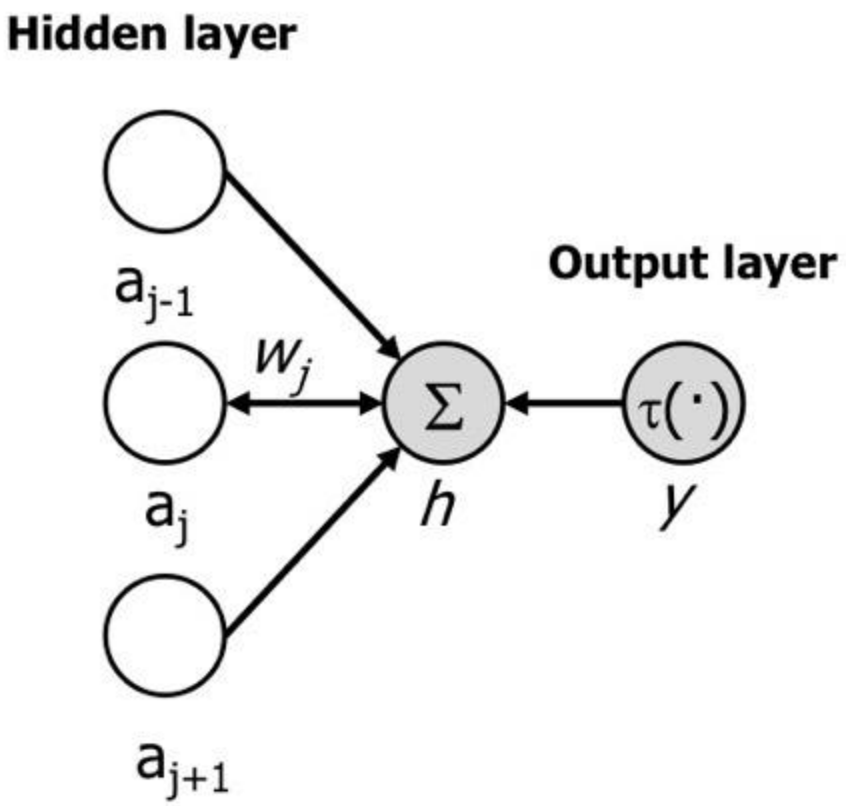
$$w_j \leftarrow w_j - \eta \frac{\partial E}{\partial w_j}$$

$$\frac{\partial E}{\partial w_j} = \underbrace{\boxed{\frac{\partial E}{\partial h}}}_{\text{Upstream gradient}} \underbrace{\boxed{\frac{\partial h}{\partial w_j}}}_{\text{Local gradient}} \qquad \frac{\partial h}{\partial w_j} = \frac{\partial\left[\sum_l w_l a_l\right]}{\partial w_j} = a_j$$

$$\frac{\partial E}{\partial h} = \boxed{\frac{\partial E}{\partial y}} \boxed{\frac{\partial y}{\partial h}}$$

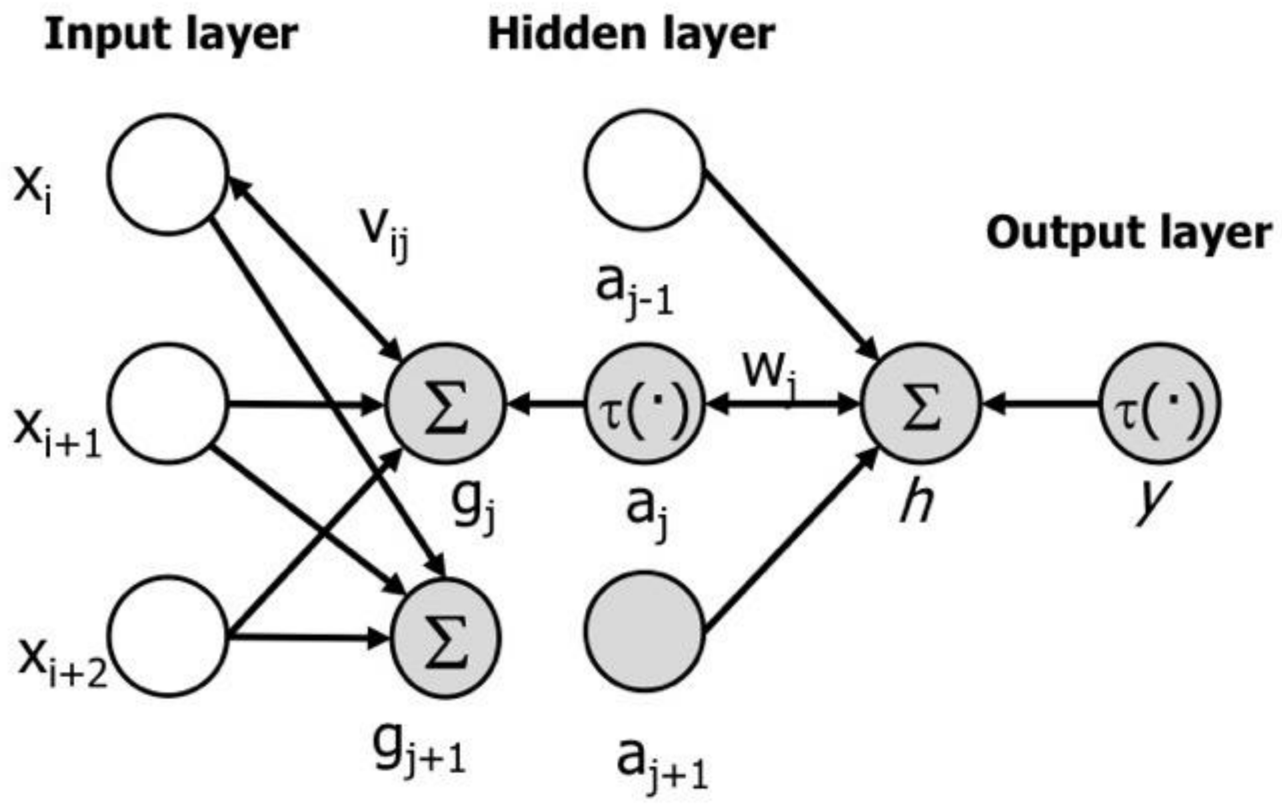
$$\frac{\partial E}{\partial y} = y - t \qquad \frac{\partial y}{\partial h} = \frac{\partial \tau(h)}{\partial h} = \tau(h)\left(1-\tau(h)\right) = y(1-y)$$

$$\therefore \frac{\partial E}{\partial w_j} = (y-t)\,y(1-y)\,a_j$$



◆ Gradient Descent

$$E = \frac{1}{2}(y-t)^2$$



$$v_{ij} \leftarrow v_{ij} - \eta \frac{\partial E}{\partial v_{ij}}$$

$$\frac{\partial E}{\partial v_{ij}} = \boxed{\frac{\partial E}{\partial g_j}} \boxed{\frac{\partial g_j}{\partial v_{ij}}}$$

$$\frac{\partial g_j}{\partial v_{ij}} = \frac{\partial \left[\sum_l x_l v_{lj} \right]}{\partial v_{ij}} = x_i$$

$$\frac{\partial E}{\partial g_j} = \boxed{\frac{\partial E}{\partial h}} \boxed{\frac{\partial h}{\partial g_j}}$$

$$\frac{\partial E}{\partial h} = (y-t)\,y(1-y)$$

앞 슬라이드 참조

$$\frac{\partial h}{\partial g_j} = \boxed{\frac{\partial h}{\partial a_j}} \boxed{\frac{\partial a_j}{\partial g_j}}$$

$$\frac{\partial a_j}{\partial g_j} = \frac{\partial \tau(g_j)}{\partial g_j} = \tau(g_j)\left(1-\tau(g_j)\right) = a_j(1-a_j)$$

$$\frac{\partial h}{\partial a_j} = \frac{\partial \left[\sum_l w_l a_l \right]}{\partial a_j} = w_j$$

$$\therefore \frac{\partial E}{\partial v_{ij}} = (y-t)\,y(1-y)\,w_j a_j (1-a_j)\,x_i$$

문제

Y = x^2 수식에서 y값이 최소가 되는 x 지점을 gradient descent 방법으로 찾으려고 한다. X 값을 3에서 시작하여 learning rate를 0.1로 설정한 후 2번 업데이트를 반복하였을 때, 결정되는 x값을 도출하세요.

- ◆ Multi-layer Perceptron (MLP)
  - Training – gradient descent
    - Suitable activation function

      (a) Step function: discontinuous → non-differentiable function

      (b) Sigmoid function: Differentiable function

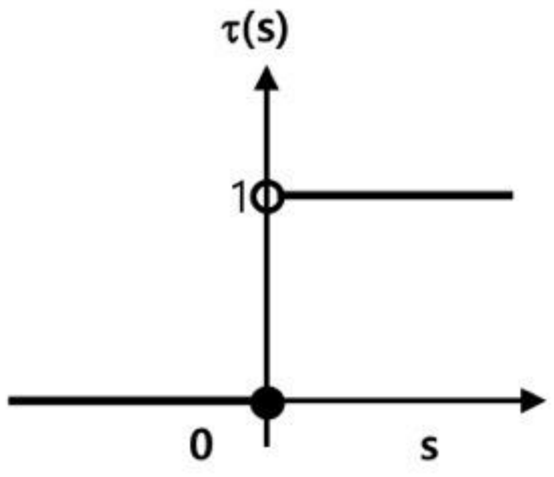
$$y = \tau(s) = \frac{1}{1+e^{-\beta s}}$$

$$\tau'(s) = \frac{\partial \tau(s)}{\partial s} = \frac{\partial\left(1+e^{-\beta s}\right)^{-1}}{\partial s} = -\left(1+e^{-\beta s}\right)^{-2}\left(e^{-\beta s}\right)(-\beta)$$

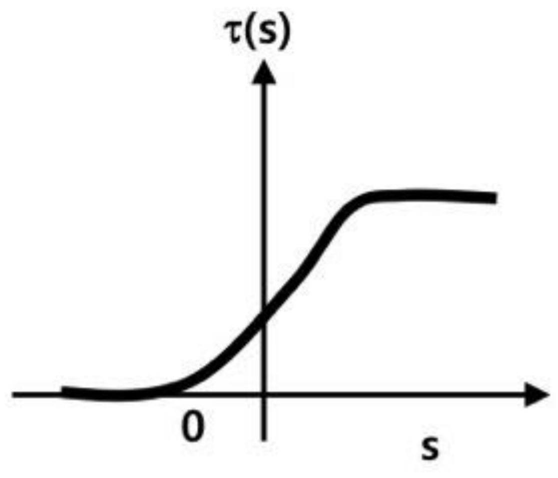
$$= \beta\left(\frac{e^{-\beta s}}{\left(1+e^{-\beta s}\right)^2}\right) = \beta\left(\frac{1}{\left(1+e^{-\beta s}\right)}\frac{e^{-\beta s}}{\left(1+e^{-\beta s}\right)}\right)$$

$$= \beta\left(\frac{1}{\left(1+e^{-\beta s}\right)}\left(1-\frac{1}{\left(1+e^{-\beta s}\right)}\right)\right) = \beta\tau(s)\left(1-\tau(s)\right)$$

$$= \beta y(1-y)$$

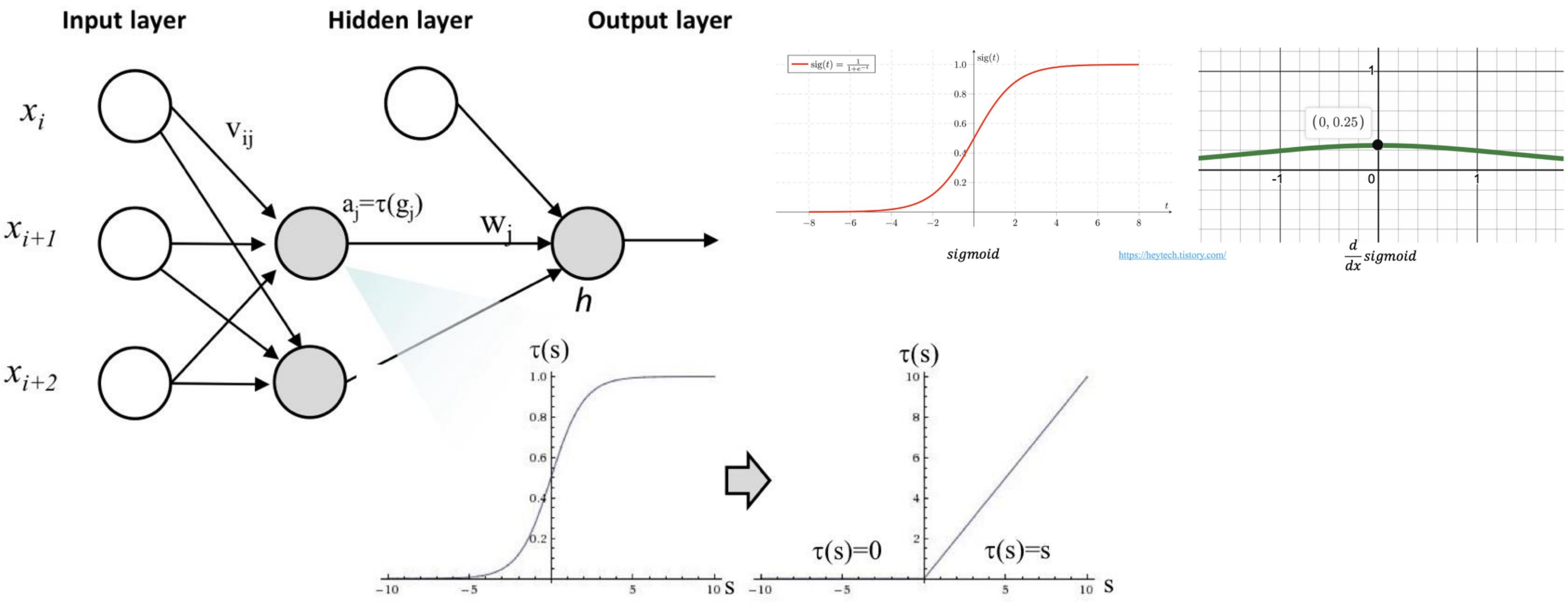


(a) Step Function τ(s)



(b) Sigmoid Function τ (s)

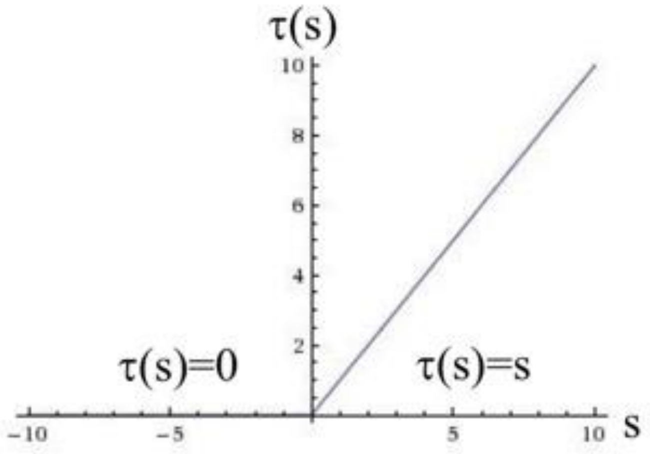
◆ Solving Vanishing Gradient Problem



Sigmoid Function $\tau(s)$

ReLU Function $\tau(s)$

- ReLU Function*
  - $\tau(s) = \max(0,s)$
    - $\tau'(s)=$
      $\begin{cases} 1 & \text{if } s>0 \\ 0 & \text{otherwise} \end{cases}$



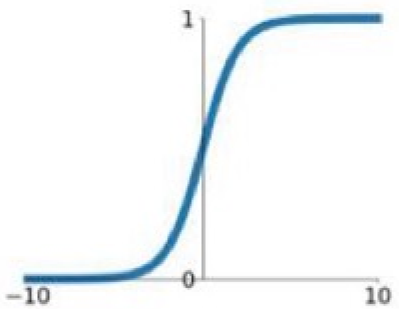
ReLU Function τ(s)

  - Advantages
    - Biological plausibility
    - Efficient gradient propagation: no vanishing gradient problem or exploding effect
    - Efficient computation: only comparison, addition and multiplication

◆ Activation function - squashing function

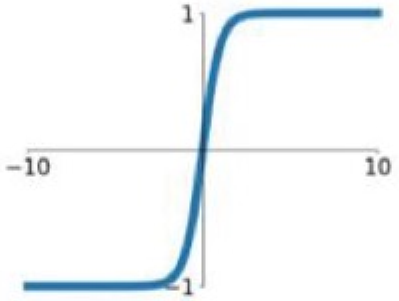
Sigmoid

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



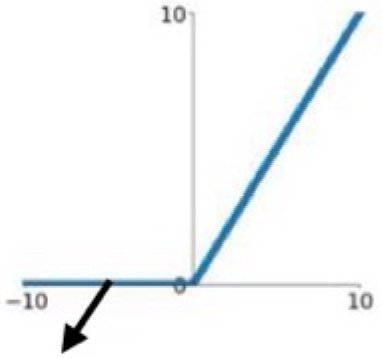
tanh

$\tanh(x)$



ReLU

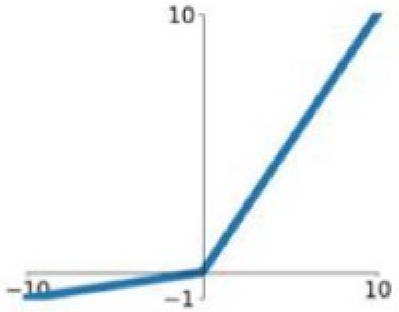
$\max(0, x)$



Only squashing for negative values

Leaky ReLU

$\max(0.1x, x)$

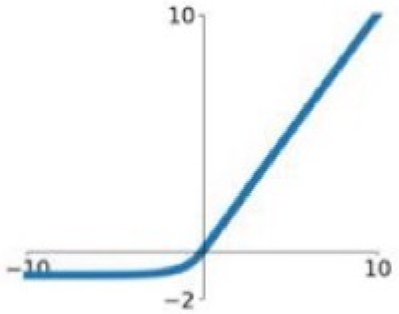


Maxout

$\max(w_1^T x + b_1, w_2^T x + b_2)$

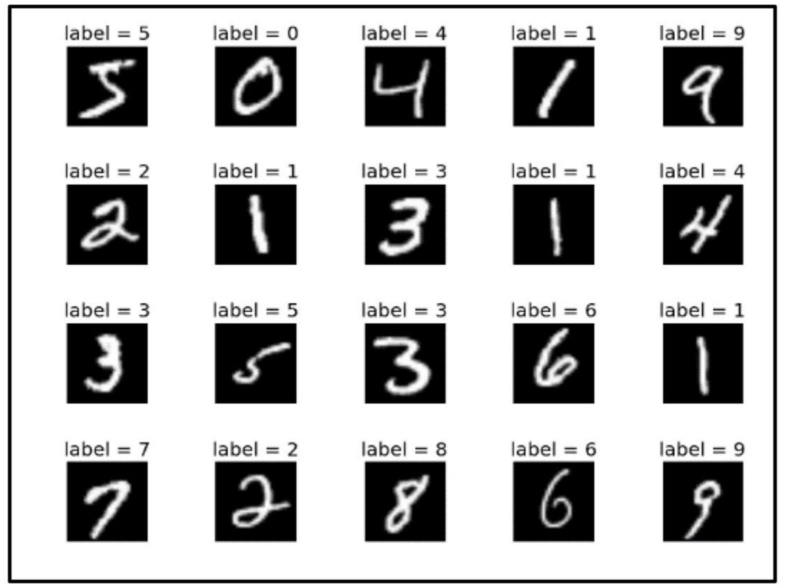
ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



ELU: Exponential Linear Unit

◆ Multi-layer Perceptron (MLP)* - Pytorch



MNIST dataset

```
import torch
import torchvision
import torch.nn.functional as F
from torchvision import transforms
from torch.utils.data.dataloader import DataLoader

# device
device = 'cuda' if torch.cuda.is_available() else 'cpu'
# device = 'cpu'
# Reproducibility
torch.manual_seed(123)
if device == 'cuda':
    torch.cuda.manual_seed_all(123)

trans = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.1307,), (0.3081,))
])
```

To.Tensor()는 (N, C, H, W) 형태의 tensor shape로 입력 데이터를 변환

N: the number of image
H,W,C: height, width, and channel

MNIST dataset은 grayscale 영상으로
To.Tensor() 적용 시 (60000, 28, 28)

Normalize(mean, standard deviation)
: 영상의 평균과 표준 편차를 통한 정규화
Color 영상인 경우
mean, std ∈ $\mathbb{R}^{1\times3}$

True: for train data
False: for test data

```
# Setup image set
train_X = torchvision.datasets.MNIST('./data', True, transform=trans, download=True)
test_X = torchvision.datasets.MNIST('./data', False, transform=trans, download=True)

# Setup data loader
train_loader = DataLoader(train_X, batch_size=64, shuffle=True, drop_last=True, pin_memory=True)
test_loader = DataLoader(test_X, batch_size=128, shuffle=False, drop_last=False, pin_memory=True)
```

Setup the image set
: the all of images and labels

Setup the data loader
: 모든 데이터를 batch size에 따라서 혹은 random하게 load하기 위해 loader 사용

shuffle : 입력 데이터의 무작위 호출을 위한 옵션
drop_last : 맨 마직 batch data를 생략 (일반적으로 훈련할 때 True, 테스트할 때 False)
pin_memory : 시스템 메모리의 직접적인 할당을 통한 CUDA 연산 효율성 증대
=GPU를 사용할 경우 일반적으로 True로 할당

◆ Multi-layer Perceptron (MLP)* - Pytorch

```
# Model
layer = torch.nn.Sequential(
   torch.nn.Flatten(), # one-dimensional 벡터로 변환
   torch.nn.Linear(in_features=784, out_features=256, bias=True),
   torch.nn.ReLU(),
   torch.nn.Linear(in_features=256, out_features=256, bias=True),
   torch.nn.ReLU(),
   torch.nn.Linear(in_features=256, out_features=10, bias=True),
).to(device)
print(layer)
```

훈련을 위한 model 정의

< 선언된 model의 print 결과 >

```
Sequential(
  (0): Flatten(start_dim=1, end_dim=-1)
  (1): Linear(in_features=784, out_features=256, bias=True)
  (2): ReLU()
  (3): Linear(in_features=256, out_features=256, bias=True)
  (4): ReLU()
  (5): Linear(in_features=256, out_features=10, bias=True)
)
```

Epoch: loader의 모든 image가 iterated

```
# Optimizer
optimizer = torch.optim.Adam(layer.parameters(), lr=0.001)

# Training
for epoch in range(15): # 총 15 epoch 훈련
  for idx, (images, labels) in enumerate(train_loader):
    # Change the data to cuda tensor and type
    images, labels = images.float().to(device), labels.long().to(device)

    # Extract output of single layer
    hypothesis = layer(images)

    # Calculate cross-entropy loss
    cost = F.cross_entropy(input=hypothesis, target=labels)

    # Gradient initialization
    optimizer.zero_grad()

    # Calculate gradient
    cost.backward()

    # Update parameters
    optimizer.step()

    # Calculate accuracy
    prob = hypothesis.softmax(dim=1)  # 0: column-wise, 1: row-wise
    pred = prob.argmax(dim=1)
    acc = pred.eq(labels).float().mean()
    if (idx+1) % 128 == 0:
      print(f'TRAIN-Iteration: {idx+1}, Loss: {cost.item()}, Accuracy: {acc.item()}')
```

◆ Multi-layer Perceptron (MLP)* - Pytorch

Result

```
TRAIN-Iteration: 128, Loss: 0.003724518697708845, Accuracy: 1.0
TRAIN-Iteration: 256, Loss: 0.00010883009963436052, Accuracy: 1.0
TRAIN-Iteration: 384, Loss: 0.0003785073640756309, Accuracy: 1.0
TRAIN-Iteration: 512, Loss: 0.026763420552015305, Accuracy:
0.984375
TRAIN-Iteration: 640, Loss: 8.215666457545012e-05, Accuracy: 1.0
TRAIN-Iteration: 768, Loss: 6.211748404894024e-05, Accuracy: 1.0
TRAIN-Iteration: 896, Loss: 0.005711937788873911, Accuracy: 1.0
TEST-Accuracy: 0.9776503443717957

Process finished with exit code 0
```

```python
# Evaluation
with torch.no_grad():
    acc = 0
    for idx, (images, labels) in enumerate(test_loader):
        images, labels = images.float().to(device), labels.long().to(device)

        # Extract output of single layer
        hypothesis = layer(images)

        # Calculate cross-entropy loss
        cost = F.cross_entropy(input=hypothesis, target=labels)

        # Calculate accuracy
        prob = hypothesis.softmax(dim=1)  # 0: column-wise, 1: row-wise
        pred = prob.argmax(dim=1)
        acc += pred.eq(labels).float().mean()
    print(f'TEST-Accuracy: {acc/len(test_loader)}')
```

References

- 밑바닥부터 시작하는 딥러닝

- 동국대 강의