SESSION # 12

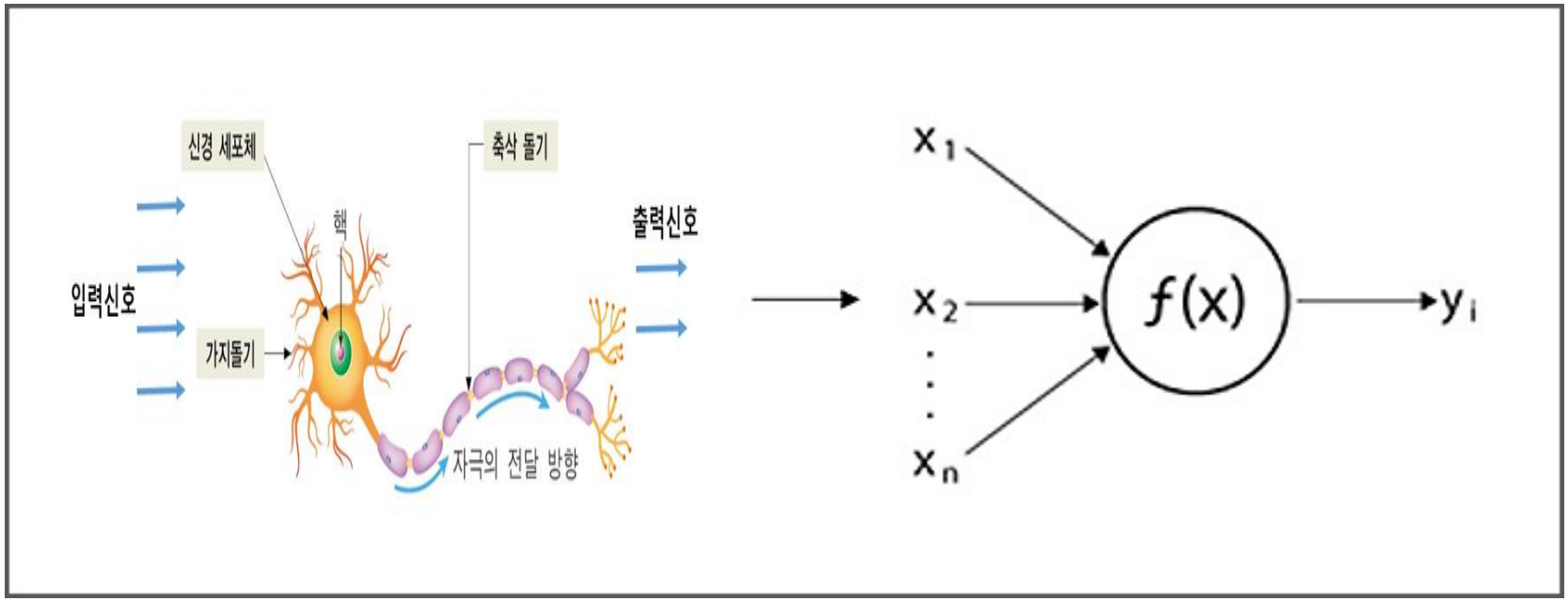
By Team 4
@MJ, Jay, Winnie, Elly
Date _ 2017.08.26

CONTENTS

01 신경망 Neural Networks

인공 신경 (Artificial Neuron)

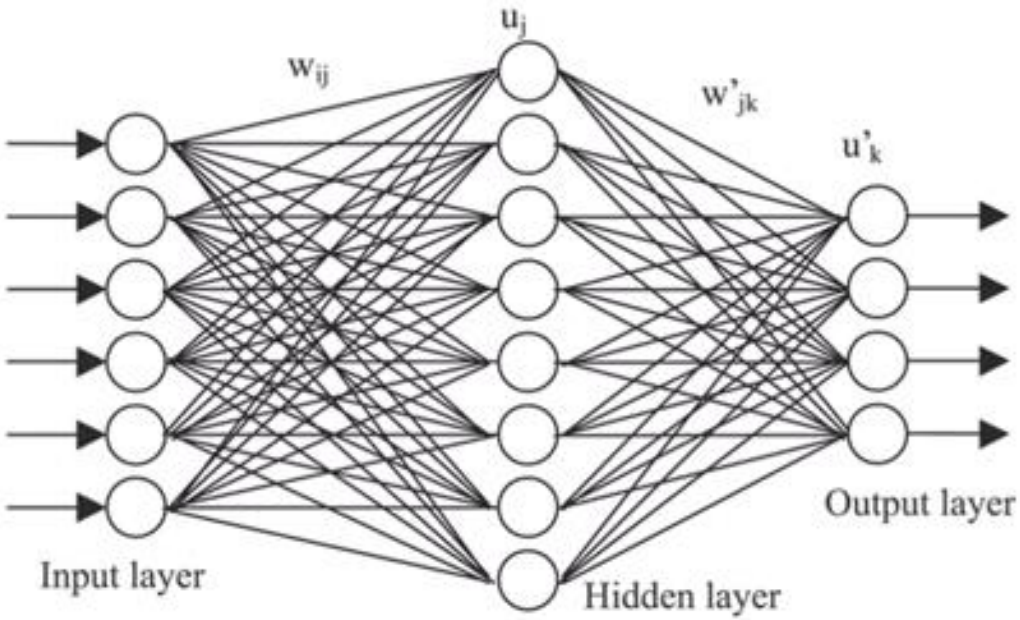
- 우리의 두뇌 속에는 천억개 정도의 뉴런이 복잡하게 연결되어있음
- 각 뉴런은 단순하게 동작
- 다른 뉴런들로부터 입력받은 신호를 어떻게 처리하여 어떻게 내보낼지는 각 뉴런마다 다름
- 이러한 신경해부학적 사실을 토대로 하여 고안된 간단한 연산기능만을 갖는 처리기



01 신경망 Neural Networks

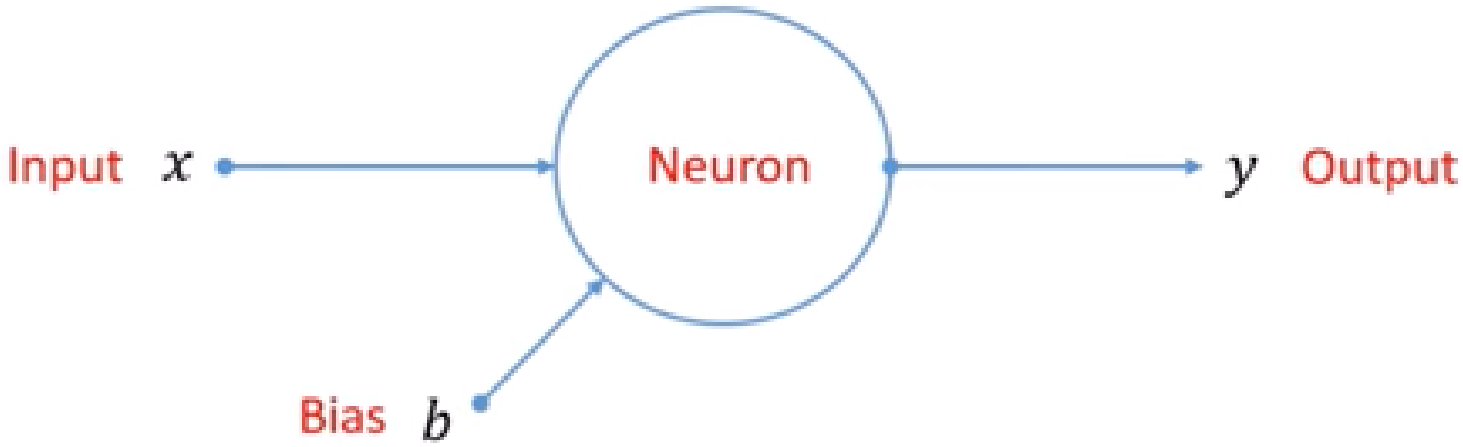
인공 신경망 (Artificial Neural Networks)

- '뉴런의 집합'인 뇌를 묘사한 예측 모델
- 단순한 연산기능을 가지고 있는 수많은 인공뉴런이 서로 연결되어 정보를 저장하고 처리
- 영문필기 인식, 얼굴 인식 등 다양한 문제에 활용
- 딥러닝은 복잡한 인공신경망을 어떻게 훈련시킬 것인가에 대한 문제



01 신경망 Neural Networks

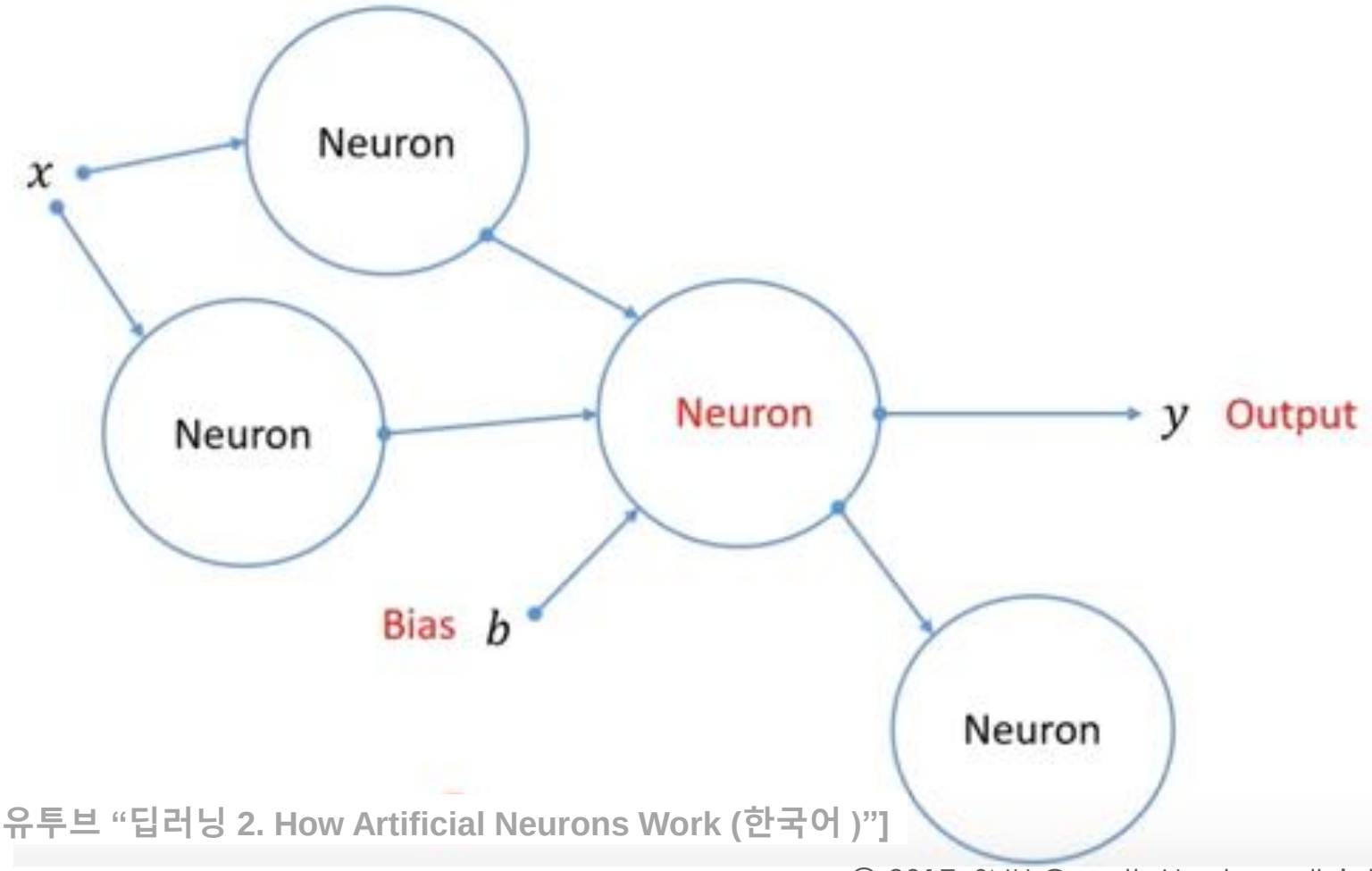
인공 신경 (Artificial Neuron)



[출처 :유투브 “딥러닝 2. How Artificial Neurons Work (한국어)”]

01 신경망 Neural Networks

인공 신경망 (Artificial Neural Networks)

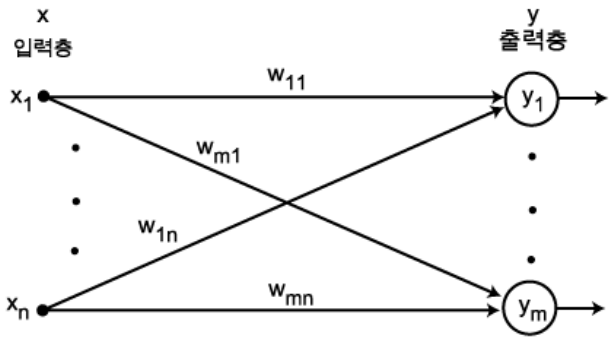


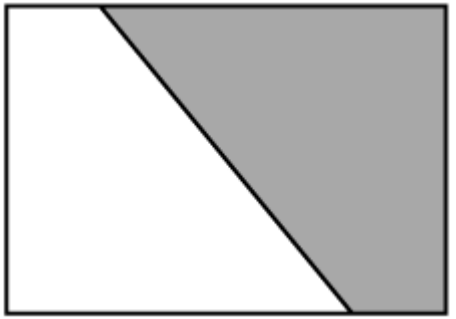
[출처 :유투브 “딥러닝 2. How Artificial Neurons Work (한국어)”]

01 신경망 Neural Networks

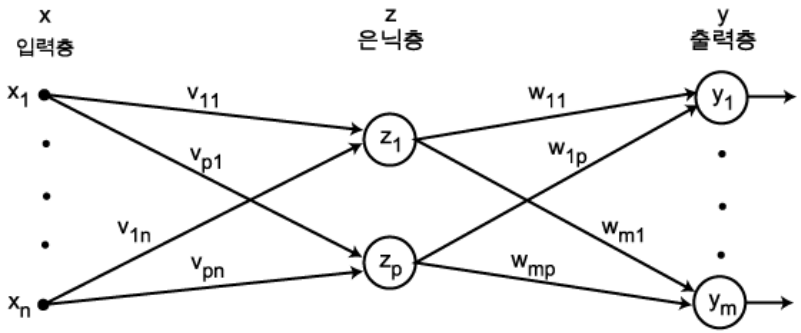
인공 신경망의 분류 - 계층수

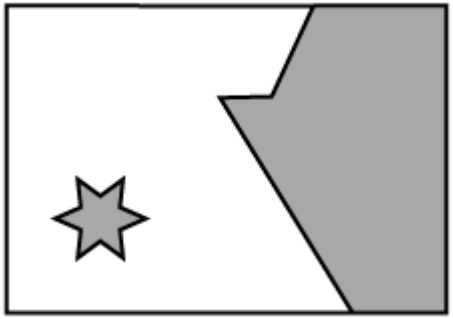
1) 단층 신경망





2) 다층 신경망

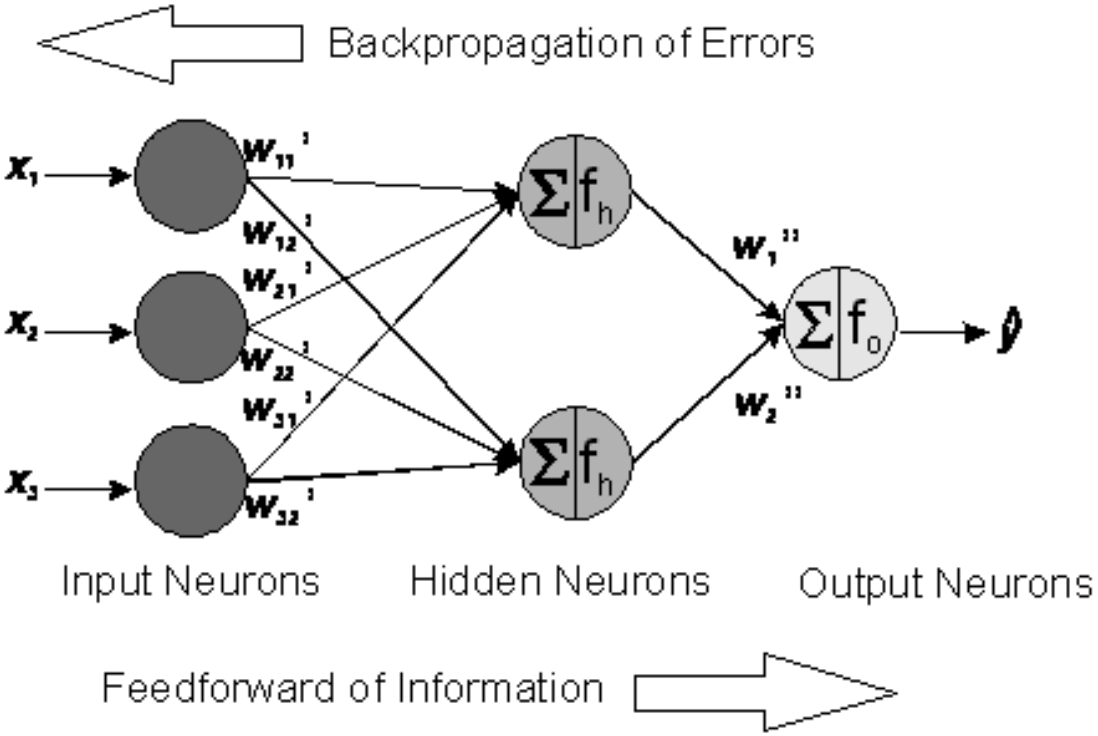




01 신경망 Neural Networks

인공 신경망의 분류 - 작동 방향

2) 역전도 신경망

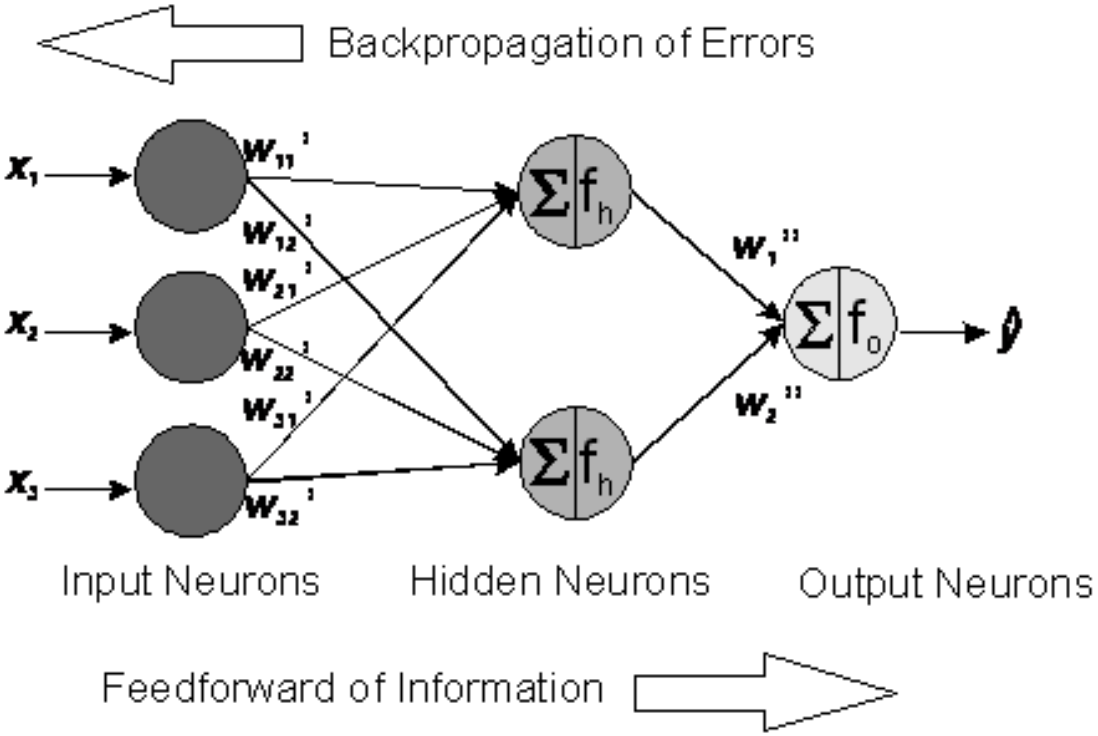


1) 순방향 신경망

01 신경망 Neural Networks

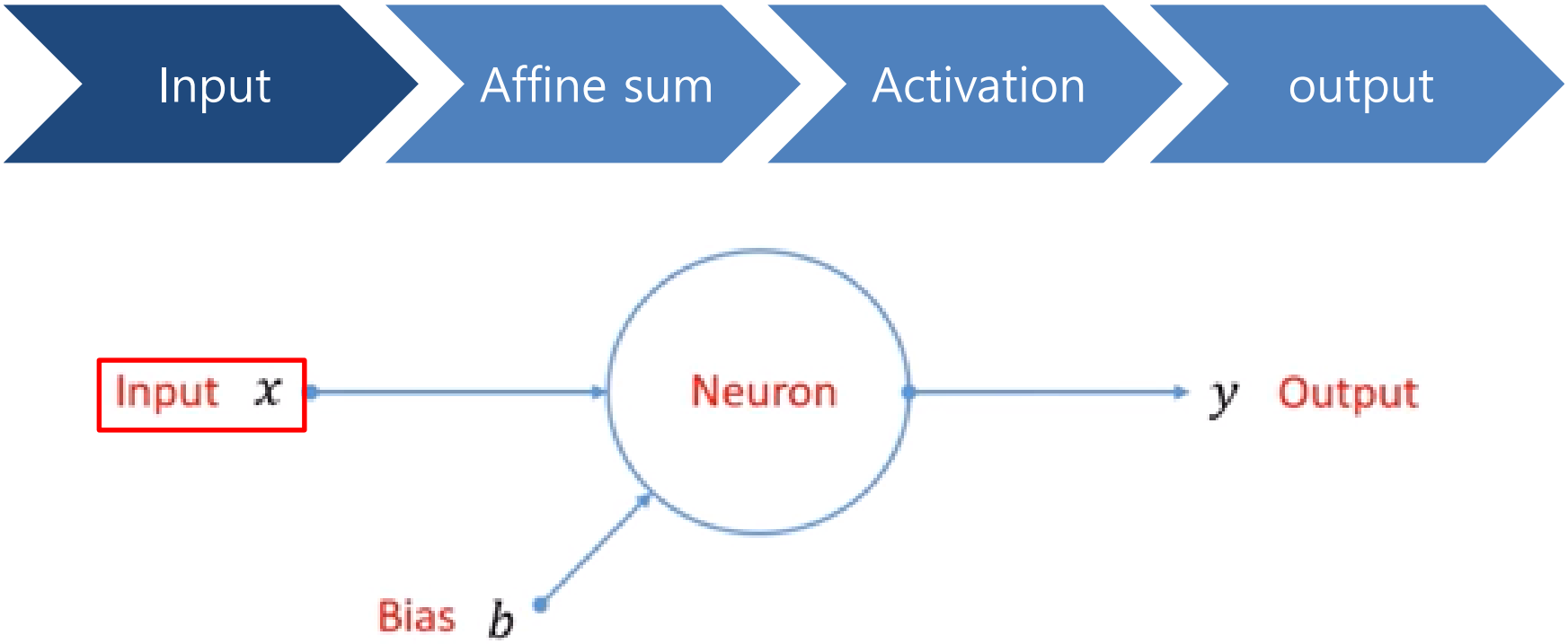
인공 신경망의 분류 - 작동 방향

2) 역전도 신경망



1) 순방향 신경망

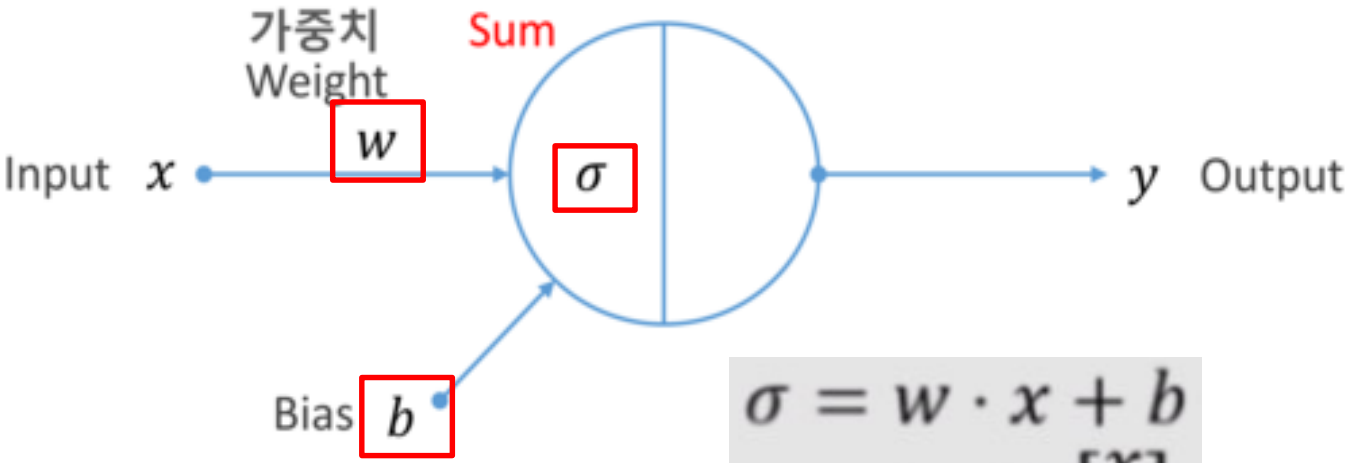
02 순방향 신경망 Feed-forward



[출처 :유투브 “딥러닝 2. How Artificial Neurons Work (한국어)”]

02 순방향 신경망 Feed-forward

Input → Affine sum → Activation → output

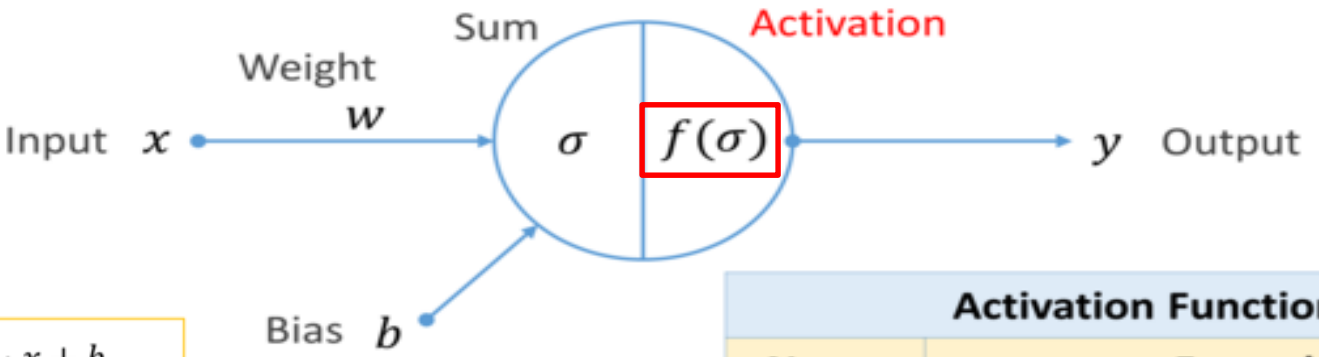


$$\sigma = w \cdot x + b = [w\ b] \begin{bmatrix} x \\ 1 \end{bmatrix}$$

[출처 :유투브 "딥러닝 2. How Artificial Neurons Work (한국어)"]

02 순방향 신경망 Feed-forward

Input → Affine sum → Activation → output



$$\sigma = w \cdot x + b$$

$$f(\sigma) = f(w \cdot x + b)$$

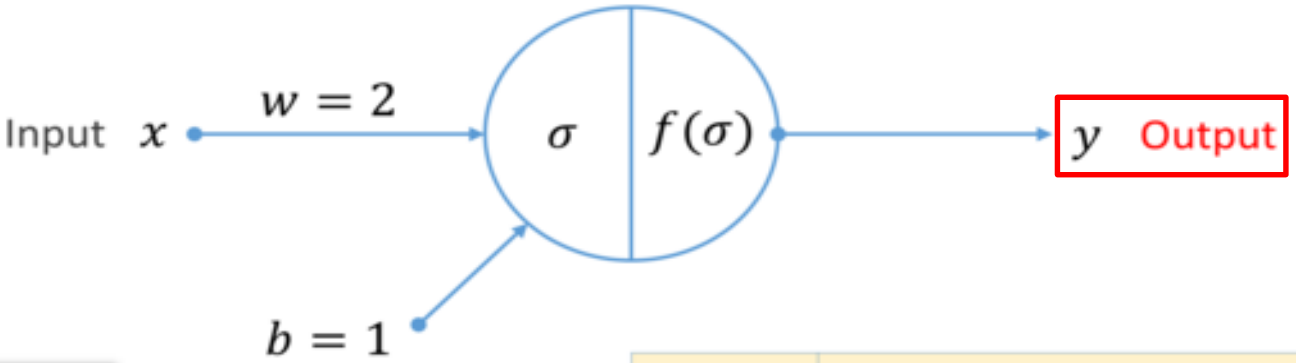
*ReLU: Rectified Linear Unit

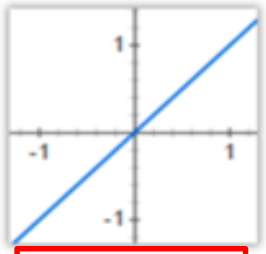
Activation Functions	
Name	**Formula**
Identity	$f(x) = x$
ReLU	$f(x) = \max(x, 0)$
Sigmoid	$f(x) = \frac{1}{1 + e^{-x}}$

[출처 :유투브 "딥러닝 2. How Artificial Neurons Work (한국어)"]

02 순방향 신경망 Feed-forward

Input → Affine sum → Activation → output





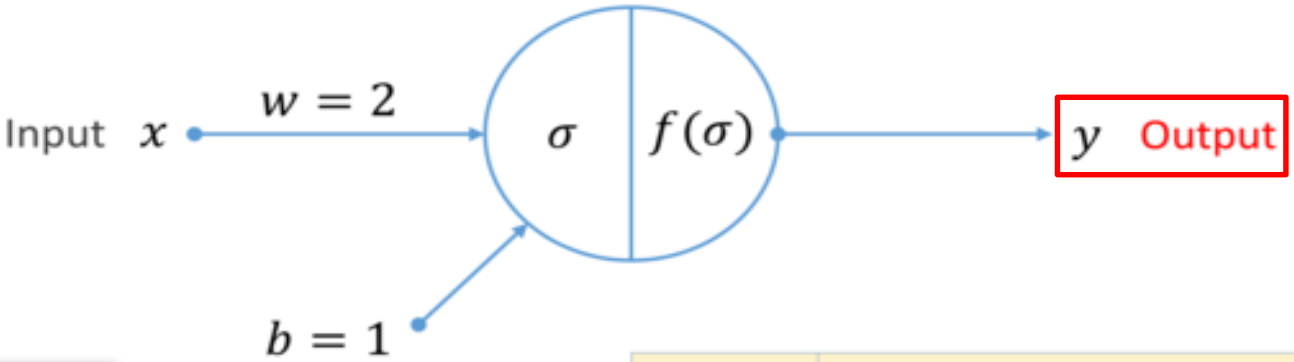
$f(x) = x$

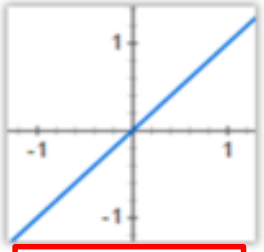
Input x	Output y
0	?
1	
2	

[출처 :유투브 "딥러닝 2. How Artificial Neurons Work (한국어)"]

02 순방향 신경망 Feed-forward

Input → Affine sum → Activation → output





$f(x) = x$

Input x	Output y
0	$y = f(2 \cdot 0 + 1) = 1$
1	$y = f(2 \cdot 1 + 1) = 3$
2	$y = f(2 \cdot 2 + 1) = 5$

[출처 :유투브 "딥러닝 2. How Artificial Neurons Work (한국어)"]

02 순방향 신경망 Feed-forward

순방향 신경망 파이썬 코드

```
def sigmoid(t):
    ?
```

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

```
def neuron_output(weights, inputs):
    ?
```

```
def feed_forward(neural_network, input_vector):
    ?
```

02 순방향 신경망 Feed-forward

순방향 신경망 파이썬 코드

```
def sigmoid(t):
    return 1 / (1 + math.exp(-t))
```

Import math

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

```
def neuron_output(weights, inputs):
    ?
```

```
def feed_forward(neural_network, input_vector):
    ?
```

02 순방향 신경망 Feed-forward

순방향 신경망 파이썬 코드

```
def sigmoid(t):
    return 1 / (1 + math.exp(-t))
```

Import math

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

```
def neuron_output(weights, inputs):
    return sigmoid(np.dot(weights, inputs))
```

신경망은 층(list)의 뉴런(list)의 가중치(list)로 표현

```
def feed_forward(neural_network, input_vector):
    ?
```

02 순방향 신경망 Feed-forward

순방향 신경망 파이썬 코드

```
def sigmoid(t):
    return 1 / (1 + math.exp(-t))
```

Import math

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

```
def neuron_output(weights, inputs):
    return sigmoid(np.dot(weights, inputs))
```

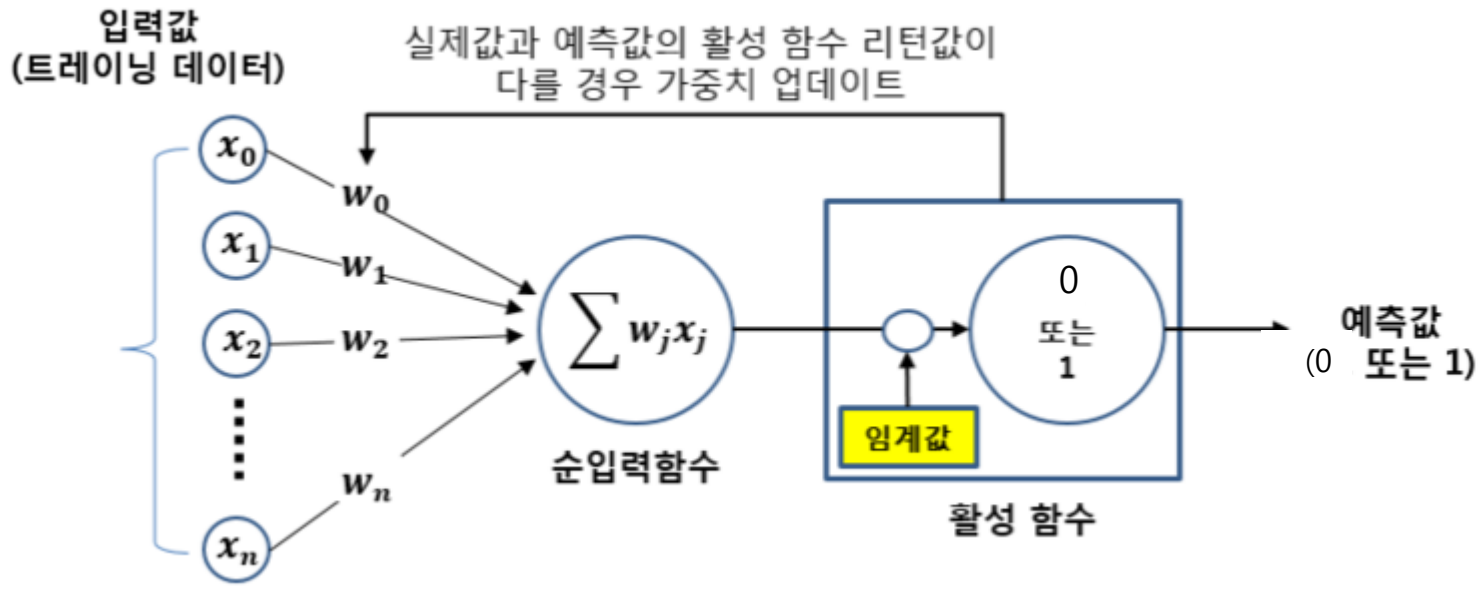
신경망은 층(list)의 뉴런(list)의 가중치(list)로 표현

```
def feed_forward(neural_network, input_vector):
    outputs = []  # 결과값을 담을 빈 리스트 생성
    for layer in neural_network:  # 한 층별로 계산
        input_with_bias = input_vector + [1]  # Bias 추가
        output = [neuron_output(neuron, input_with_bias) for neuron in layer]
                                                          # 계산
        outputs.append(output) # 빈 리스트에 저장
        input_vector = output # 이번 층의 결과값이 다음 층의 입력값이 됨
    return outputs
```

03 퍼셉트론 Perceptron

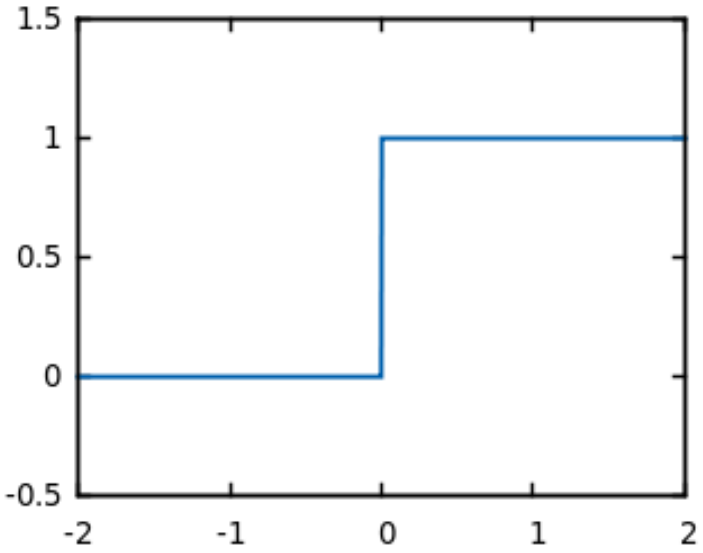
퍼셉트론 (Perceptron)

- 가장 간단한 순방향 신경망 구조
- N개의 이진수가 하나의 뉴런을 통과해서 가중합이 0보다 크거나 같으면 활성화



03 퍼셉트론 Perceptron

퍼셉트론 활성화 함수 - 계단 함수




```
def step_function(x):
    return ?
```

```
def ?
```

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

numpy.dot

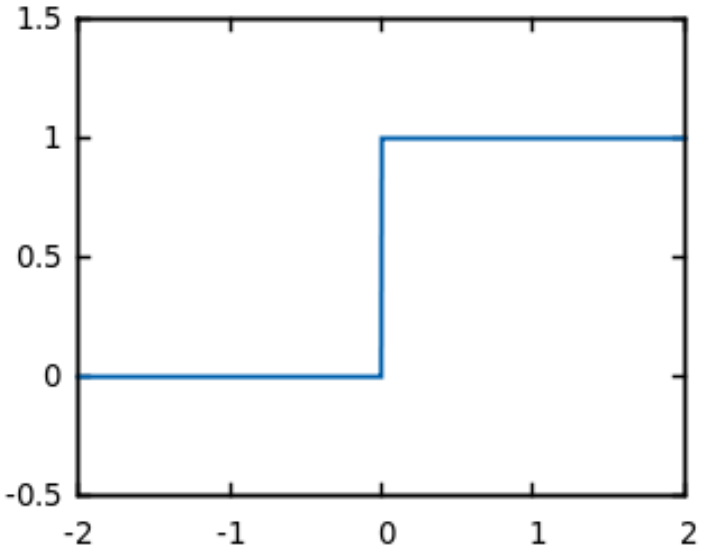
numpy.**dot**(*a*, *b*, *out=None*)

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*:

03 퍼셉트론 Perceptron

퍼셉트론 활성화 함수 - 계단 함수




```
def step_function(x):
    return 1 if x >= 0 else 0
```

```
def perceptron_output(weights, bias, x):
    '''퍼셉트론이 활성화되면 1 아니면 0을 반환 '''
    ?
```

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

numpy.dot

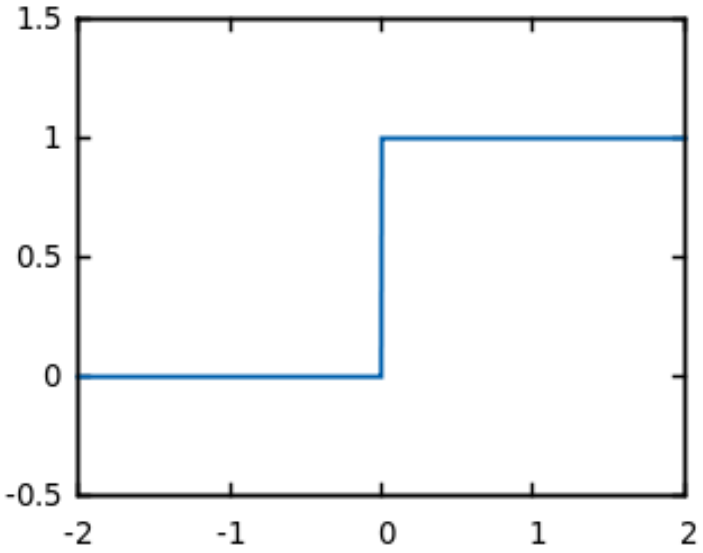
numpy. **dot** (*a*, *b*, *out=None*)

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*:

03 퍼셉트론 Perceptron

퍼셉트론 활성화 함수 - 계단 함수




```
def step_function(x):
    return 1 if x >= 0 else 0
```

```
def perceptron_output(weights, bias, x):
    '''퍼셉트론이 활성화되면 1 아니면 0을 반환 '''
    calculation = np.dot(weights, x) + bias
    return step_function(calculation)
```

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

numpy.dot

numpy. **dot** (*a*, *b*, *out=None*)

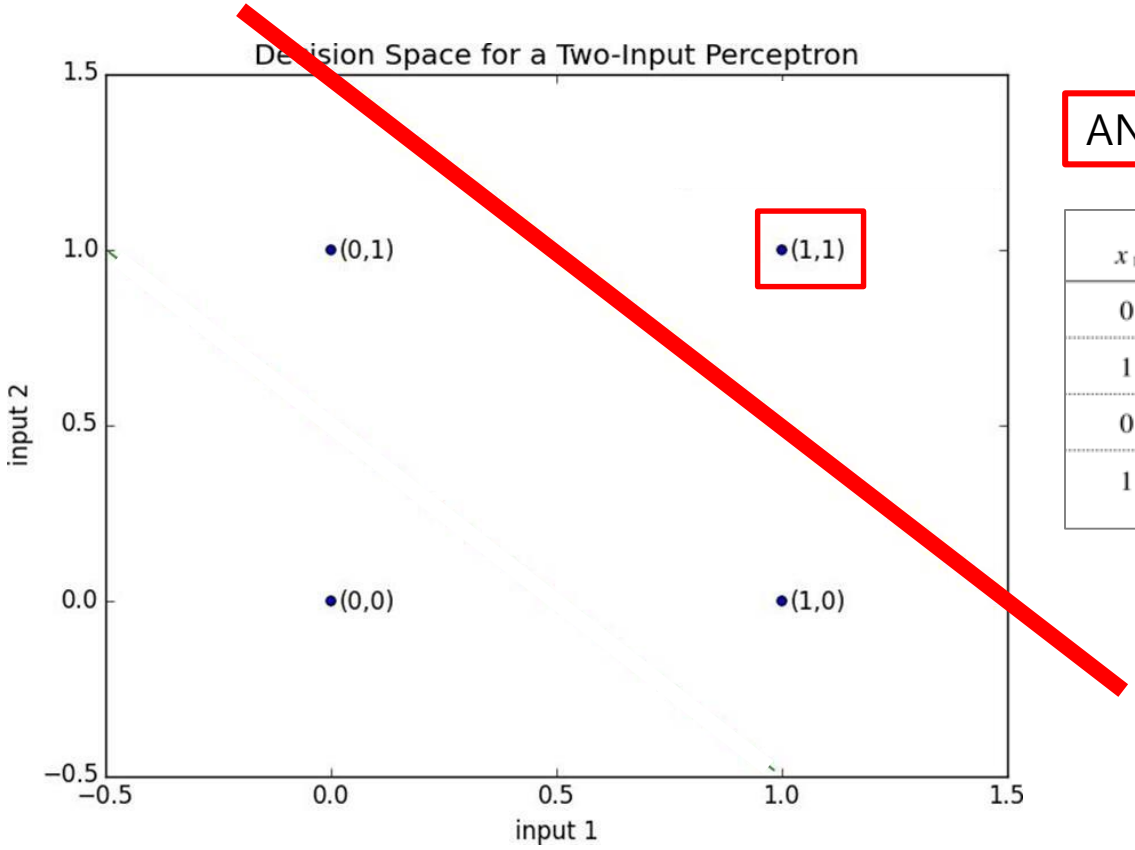
Dot product of two arrays.

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*:

03 퍼셉트론 Perceptron

퍼셉트론에서 사용되는 논리 연산

- 가중치만 잘 선택하면 여러 가지 간단한 문제 해결 가능



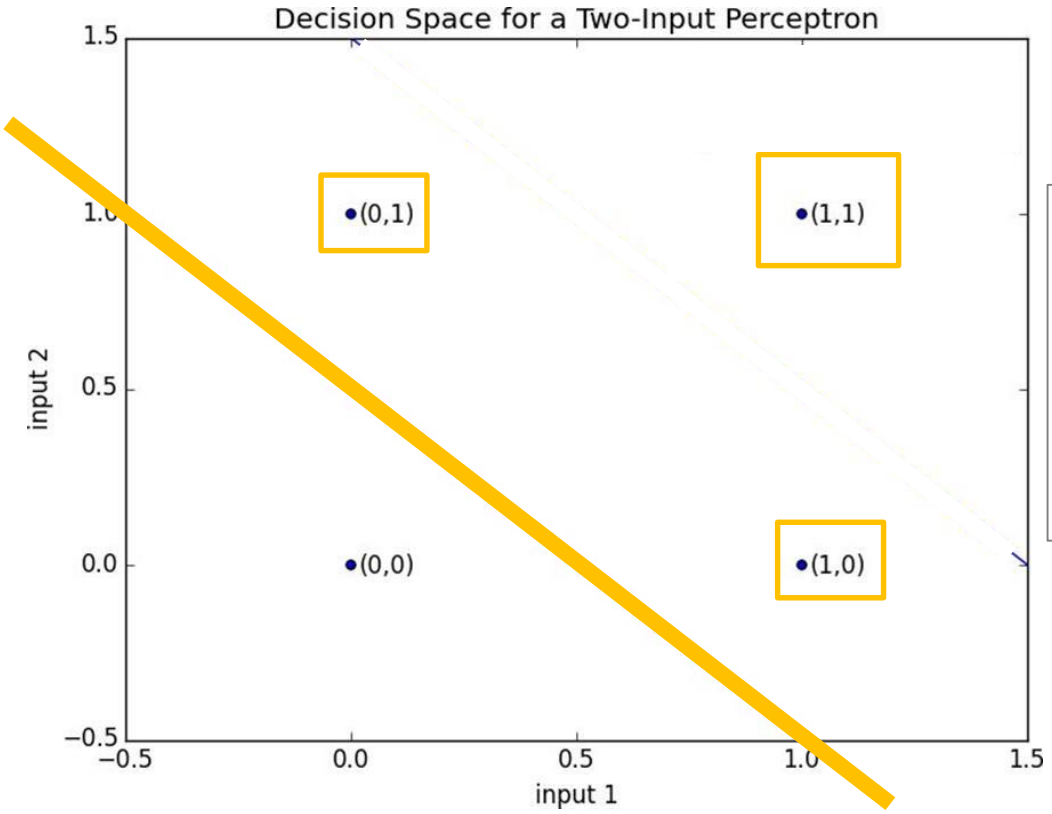
AND gate

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

03 퍼셉트론 Perceptron

퍼셉트론에서 사용되는 논리 연산

- 가중치만 잘 선택하면 여러 가지 간단한 문제 해결 가능



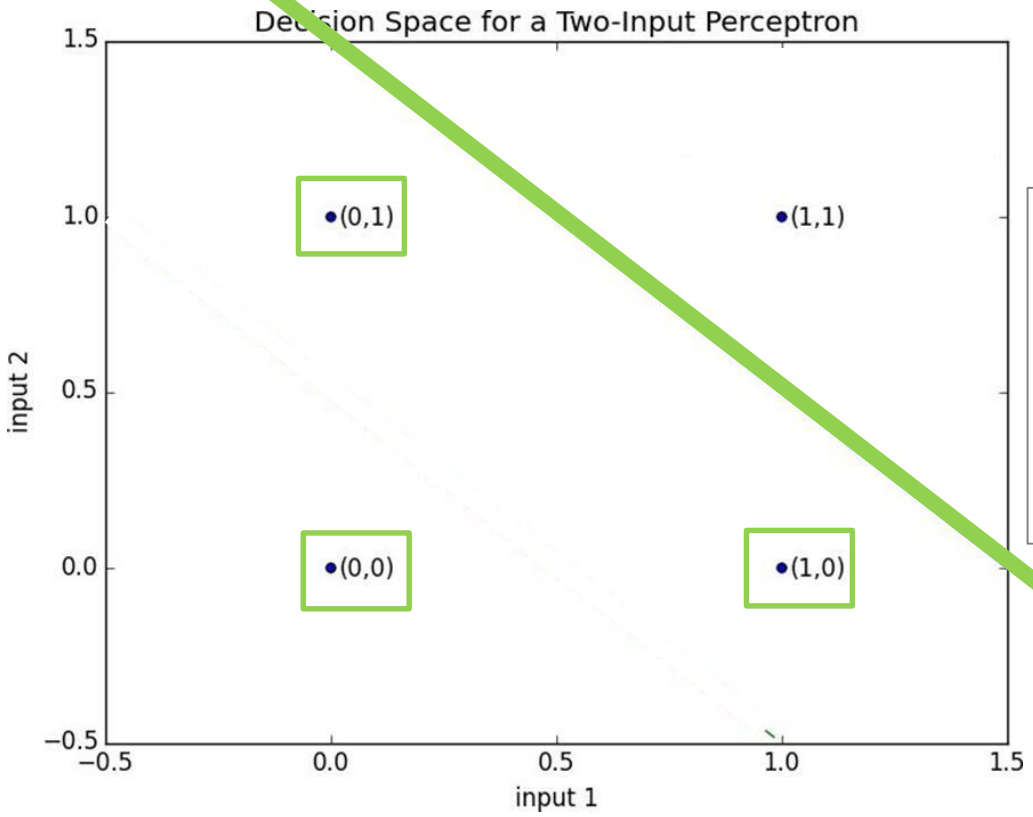
OR gate

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

03 퍼셉트론 Perceptron

퍼셉트론에서 사용되는 논리 연산

- 가중치만 잘 선택하면 여러 가지 간단한 문제 해결 가능



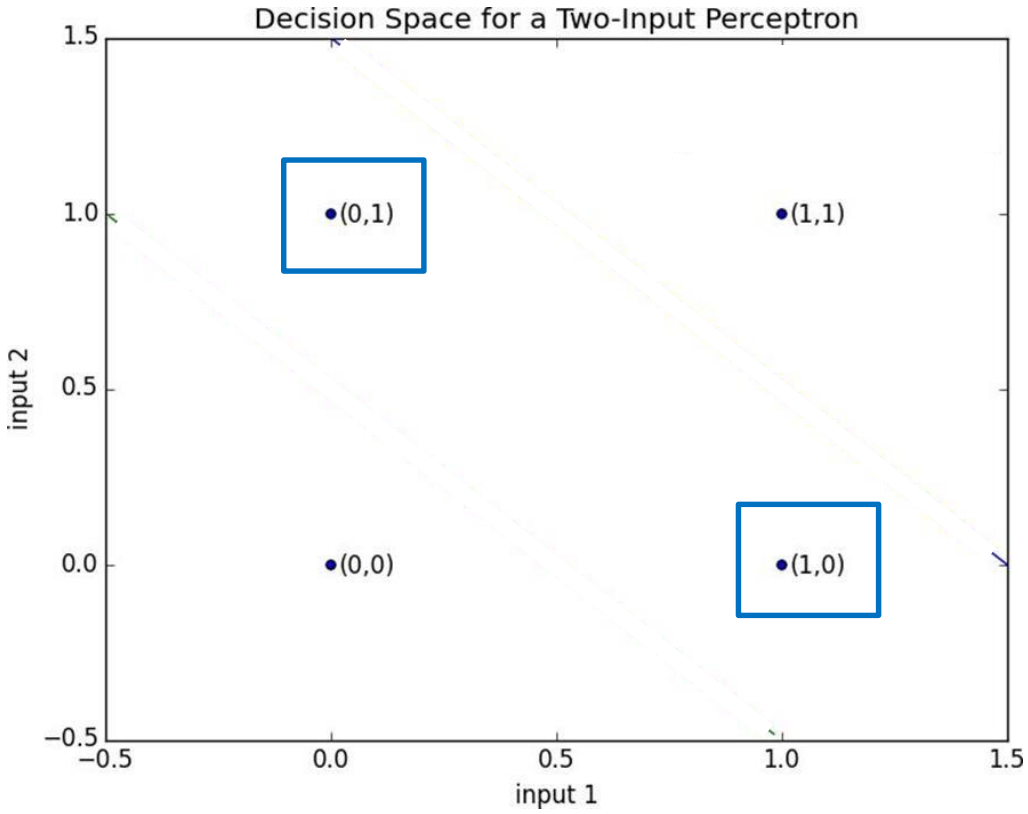
NAND gate

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

03 퍼셉트론 Perceptron

단층 퍼셉트론의 한계

- 단층으로 해결되지 않는 문제는 다층 퍼셉트론으로 해결



XOR gate?

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

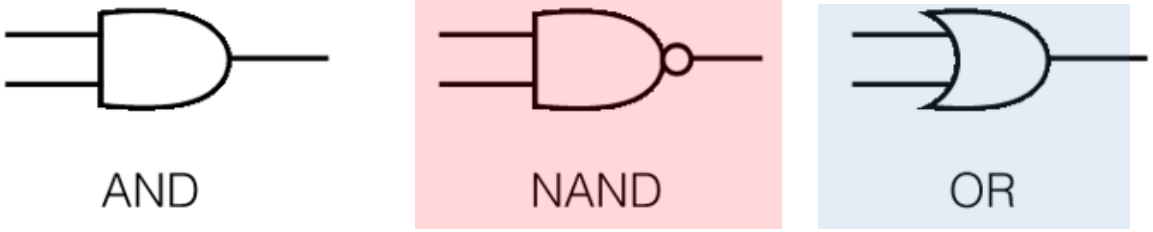
03 퍼셉트론 Perceptron

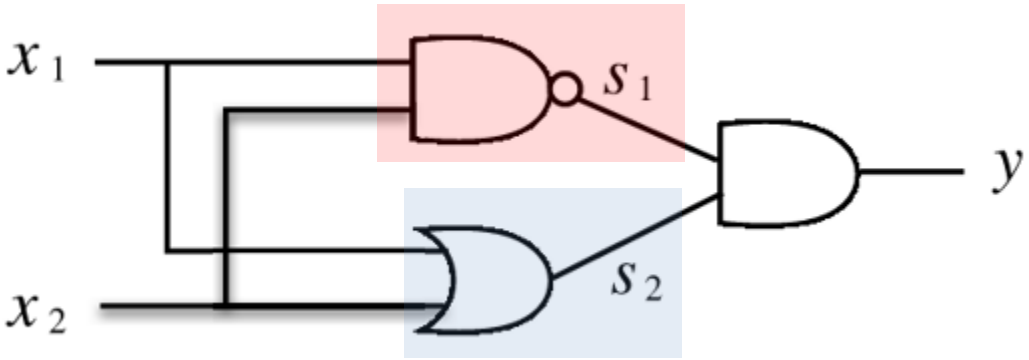
단층 퍼셉트론의 한계

- 단층으로 해결되지 않는 문제는 다층 퍼셉트론으로 해결

XOR(x1, x2) = AND(NAND(x1, x2), OR(x1, x2))

XOR gate?





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

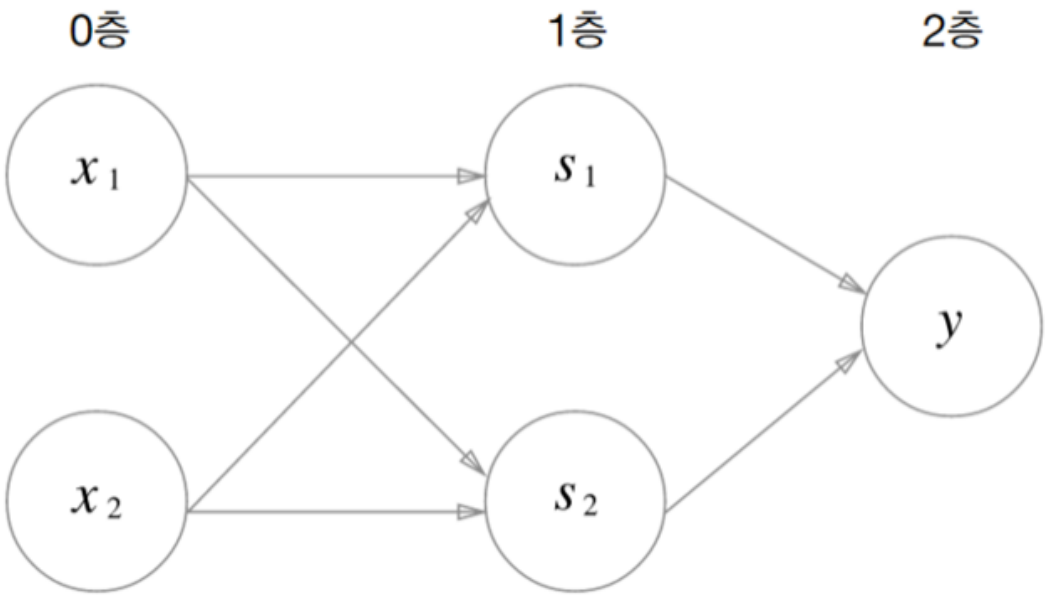
03 퍼셉트론 Perceptron

단층 퍼셉트론의 한계

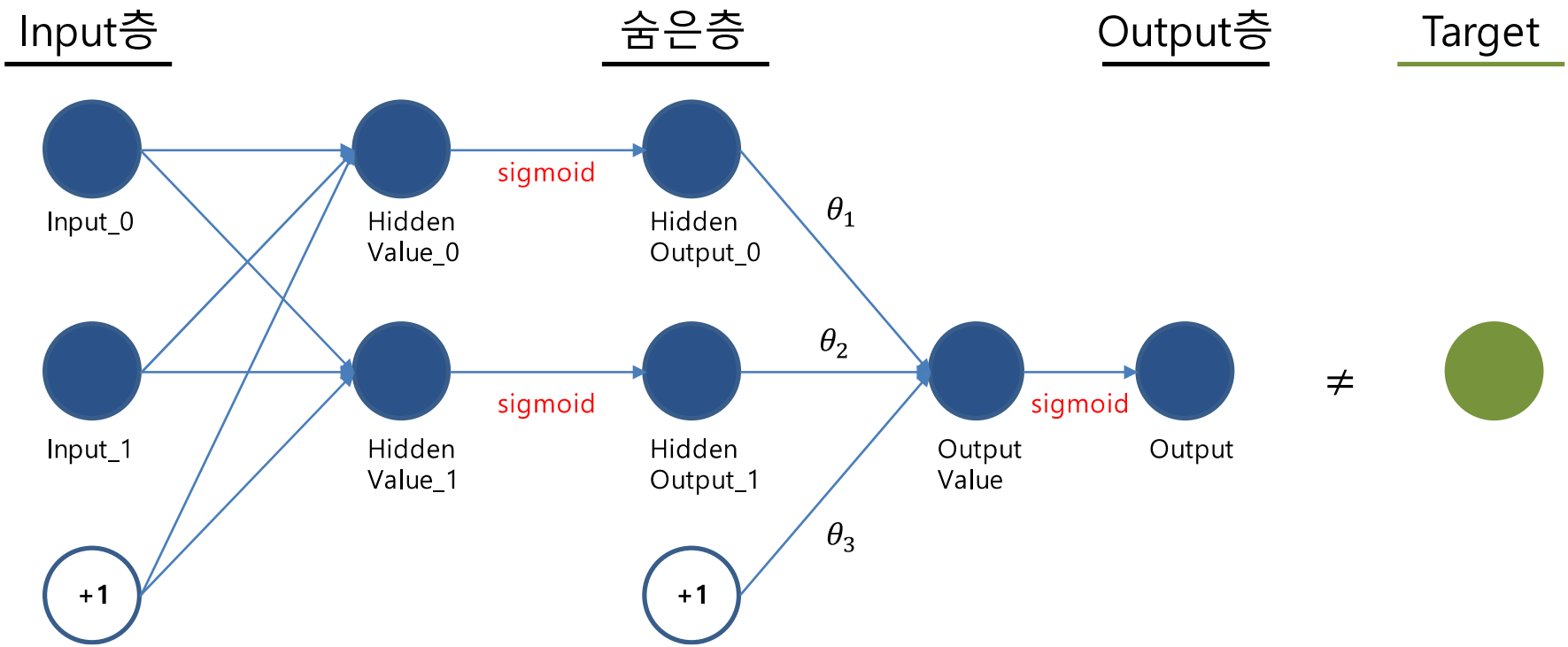
- 단층으로 해결되지 않는 문제는 다층 퍼셉트론으로 해결 -> ***은닉층***

XOR(x1, x2) = AND(NAND(x1, x2), OR(x1, x2))

XOR gate?



04 Backpropagation – FP



Cost Function?

04 Backpropagation

Layer들이 도입되면서 생기는 의문점

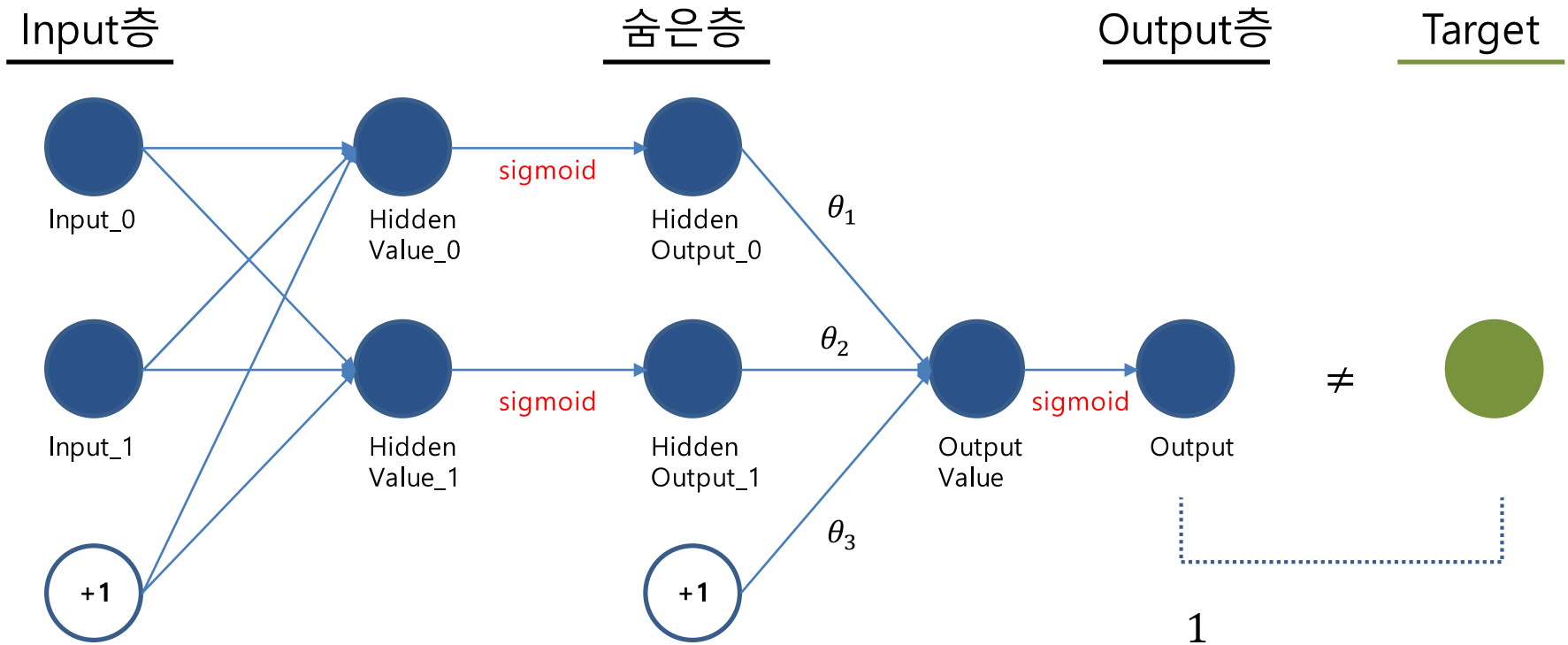
- Cost Function은 도둬췌 어떤 모습일까

- 어떻게 모든 Theta에 대해 최적화 시킬 것인가

  각 Theta가 1 만큼 바뀔 때,
  전체 Cost Function의 변화는 어떻게 되는가...

  Theta 각각의 영향력(Gradient)은 어느 정도인가?

04 Backpropagation – BP



$$J = \frac{1}{2}(Target - Output)^2$$

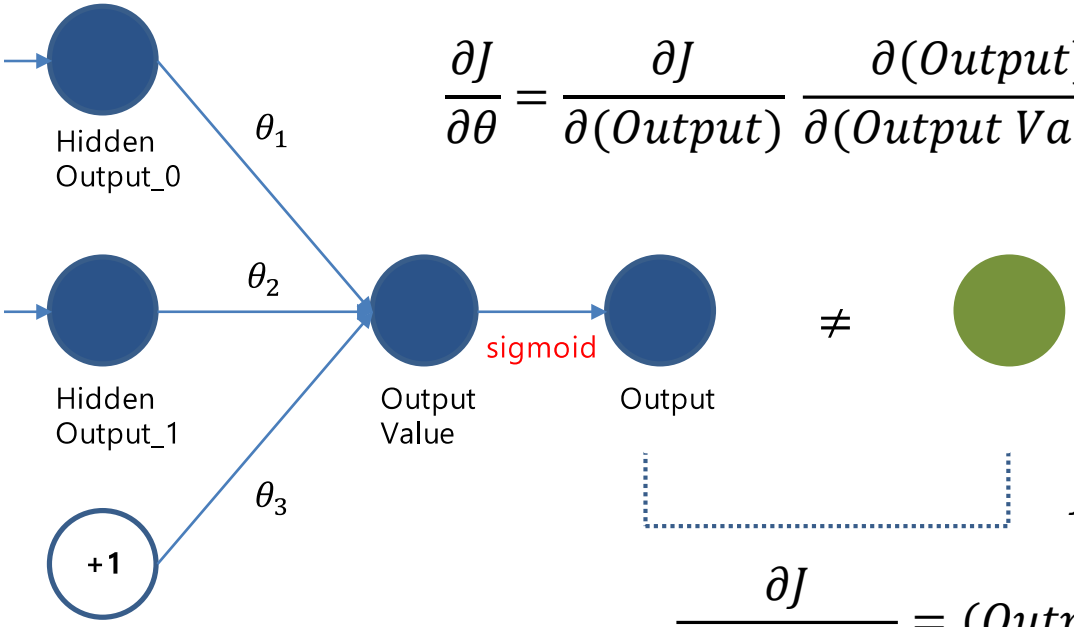
Output의 함수

알고싶은것: $\frac{\partial}{\partial\theta}J$

$$= \frac{\partial J}{\partial(Output)} \frac{\partial(Output)}{\partial(Output\ Value)} \frac{\partial(Output\ Value)}{\partial\theta}$$

04 Backpropagation – BP

$$Output\ Value = \theta_1 * Hidden\ Output0 + \cdots$$



$$\frac{\partial J}{\partial \theta} = \frac{\partial J}{\partial (Output)} \frac{\partial (Output)}{\partial (Output\ Value)} \frac{\partial (Output\ Value)}{\partial \theta}$$

$$\neq$$

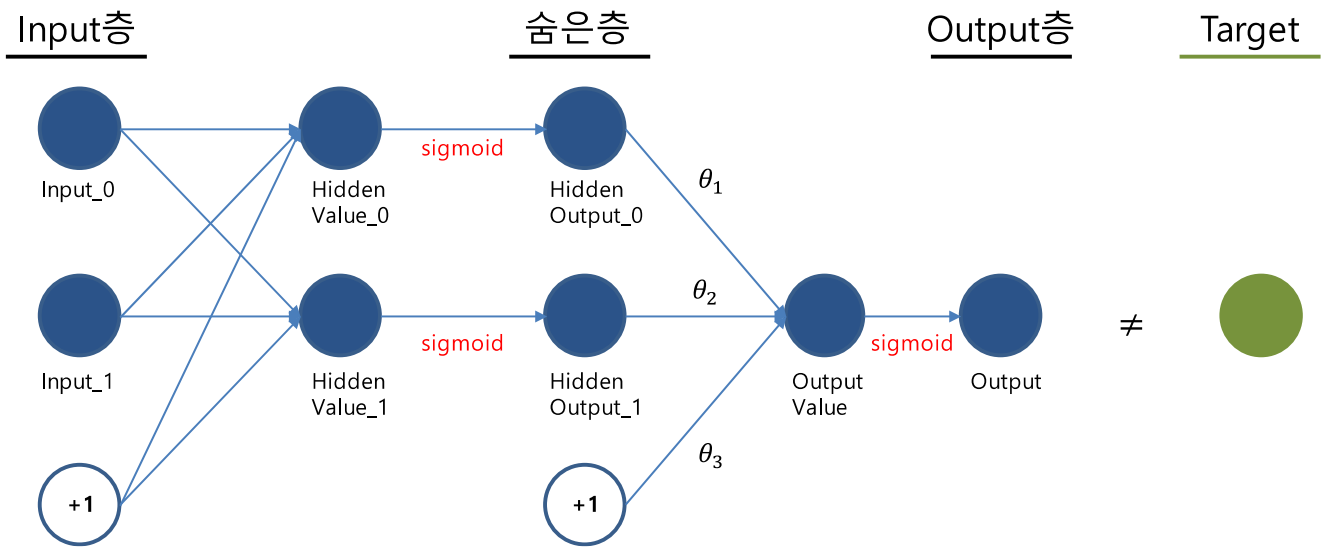
$$J = \frac{1}{2}(Target - Output)^2$$

$$\frac{\partial J}{\partial (Output)} = (Output - Target)$$

$$\frac{\partial J}{\partial (Output)} \frac{\partial (Output)}{\partial (Output\ Value)} = (Output - Target) * Output * (1 - Output)$$

Sigmoid 미분

04 Backpropagation – BP



$$\frac{\partial J}{\partial \theta} = \frac{\partial J}{\partial (Output)} \frac{\partial (Output)}{\partial (Output\ Value)} \frac{\partial (Output\ Value)}{\partial \theta}$$

$$= (Output - Target) * Output * (1 - Output) * Hidden\ Output$$

$$\theta = \theta - \frac{\partial J}{\partial \theta}$$

04 Backpropagation

코딩

```
def feed_forward(neural_network, input_vector):
    outputs = []
    for layer in neural_network:
        input_with_bias = input_vector + [1]
        output = [neuron_output(neuron, input_with_bias) for neuron in layer]
        outputs.append(output)
        input_vector = output
    return outputs
```

04 Backpropagation

코딩

$$\frac{\partial J}{\partial (Output)} \frac{\partial (Output)}{\partial (Output\ Value)} = (Output - Target) * Output * (1 - Output)$$

$$\frac{\partial J}{\partial \theta} = (Output - Target) * Output * (1 - Output) * Hidden\ Output$$

$$\theta = \theta - \frac{\partial J}{\partial \theta}$$

```
def backpropagate(network, input_vector, targets):
    hidden_outputs, outputs = feed_forward(network, input_vector)

    #Step 1

    output_deltas = [output * (1-output) * (output-target)
                     for output, target in zip(outputs, targets)]

    for i, output_neuron in enumerate(network[-1]):
        for j, hidden_output in enumerate(hidden_outputs + [1]):
            output_neuron[j] -= output_deltas[i] * hidden_output

    #Step 2

    hidden_deltas = [hidden_output * (1-hidden_output) * np.dot(output_deltas, [n[i] for n in output_layer])
                     for i, hidden_output in enumerate(hidden_outputs)]

    for i, hidden_neuron in enumerate(network[0]):
        for j, input in enumerate(input_vector + [1]):
            hidden_neuron[j] -= hidden_deltas[i] * input
```

04 Backpropagation

코딩

```
def backpropagate(network, input_vector, targets):
    hidden_outputs, outputs = feed_forward(network, input_vector)

    #Step 1

    output_deltas = [output * (1-output) * (output-target)
                     for output, target in zip(outputs, targets)]

    for i, output_neuron in enumerate(network[-1]):
        for j, hidden_output in enumerate(hidden_outputs + [1]):
            output_neuron[j] -= output_deltas[i] * hidden_output

    #Step 2

    hidden_deltas = [hidden_output * (1-hidden_output) * np.dot(output_deltas, [n[i] for n in output_layer])
                     for i, hidden_output in enumerate(hidden_outputs)]

    for i, hidden_neuron in enumerate(network[0]):
        for j, input in enumerate(input_vector + [1]):
            hidden_neuron[j] -= hidden_deltas[i] * input
```

$$\frac{\partial J}{\partial \theta} = \frac{\partial J}{\partial (Output)} \frac{\partial (Output)}{\partial (Output\ Value)} \times \frac{\partial (Output\ Value)}{\partial (Hidden\ Output)} \frac{\partial (Hidden\ Output)}{\partial (Hidden\ Value)} \frac{\partial (Hidden\ Value)}{\partial \theta}$$

04 Backpropagation

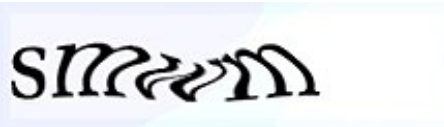
코딩

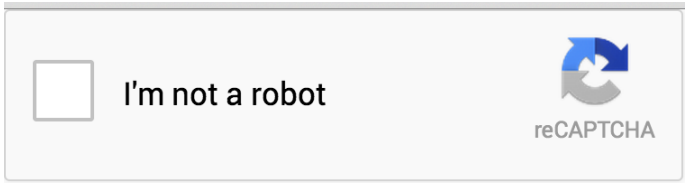
```
def backpropagate(network, input_vector, targets):
    hidden_outputs, outputs = feed_forward(network, input_vector)

    #Step 1

    output_deltas = [output * (1-output) * (output-target)
                     for output, target in zip(outputs, targets)]

    for i, output_neuron in enumerate(network[-1]):
        for j, hidden_output in enumerate(hidden_outputs + [1]):
            output_neuron[j] -= output_deltas[i] * hidden_output

    #Step 2

    hidden_deltas = [hidden_output * (1-hidden_output) * np.dot(output_deltas, [n[i] for n in output_layer])
                     for i, hidden_output in enumerate(hidden_outputs)]

    for i, hidden_neuron in enumerate(network[0]):
        for j, input in enumerate(input_vector + [1]):
            hidden_neuron[j] -= hidden_deltas[i] * input
```

05 실습 – CAPTCHA 깨기

Captcha 소개 및 실습 목표

- Captch : Completely Automated Public Turing test to tell Computers and Humans Apart (완전 자동화된 사람과 컴퓨터 판별)
- 어떠한 사용자가 실제 사람인지 컴퓨터 프로그램인지를 구별하기 위해 사용되는 방법
- 사람은 구별할 수 있지만 컴퓨터는 구별하기 힘들게 의도적으로 비틀거나 덧칠한 그림을 주고 그 그림에 쓰여 있는 내용을 물어보는 방법이 자주 사용됨





숫자 이미지가 주어졌을 때 컴퓨터는 정말로 숫자를 맞출 수 없을까??

→ Goal: 인공신경망 모델을 이용해 숫자에 대한 captcha 테스트를 통과해보자!

05 실습 – CAPTCHA 깨기

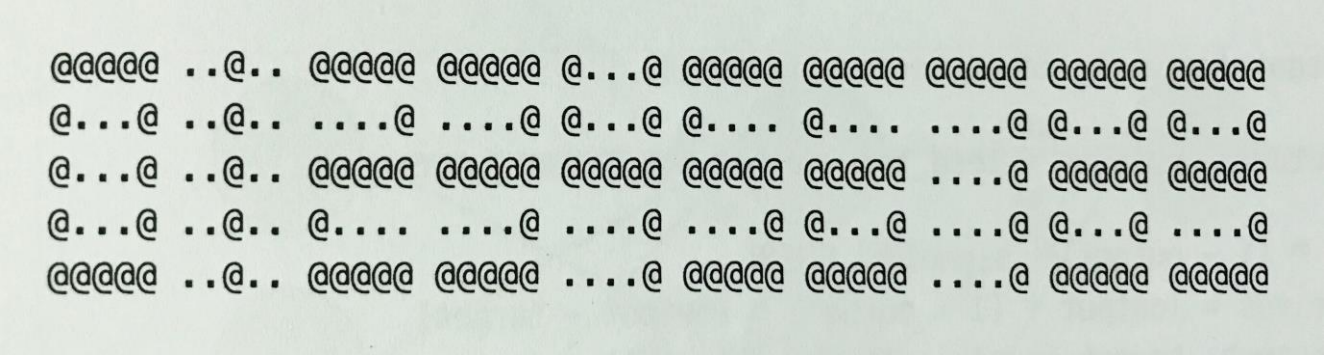
Flow chart

Input과 output 데이터 정의	첫 뉴런으로 들어갈 **입력 변수**와 마지막 뉴런으로부터 나올 **출력 변수**를 정의
▼	
은닉층과 출력층 설계	각 뉴런이 입력 값에 대해 가질 weight와 bias를 임의로 설정
▼	
Backpropagation	Backpropagation 알고리즘의 반복을 통해 적절한 weight와 bias 학습
▼	
Predict	숫자 이미지를 받아 예측하는 함수 구성

05 실습 – CAPTCHA 깨기

1. 입력값과 출력값 list 만들기

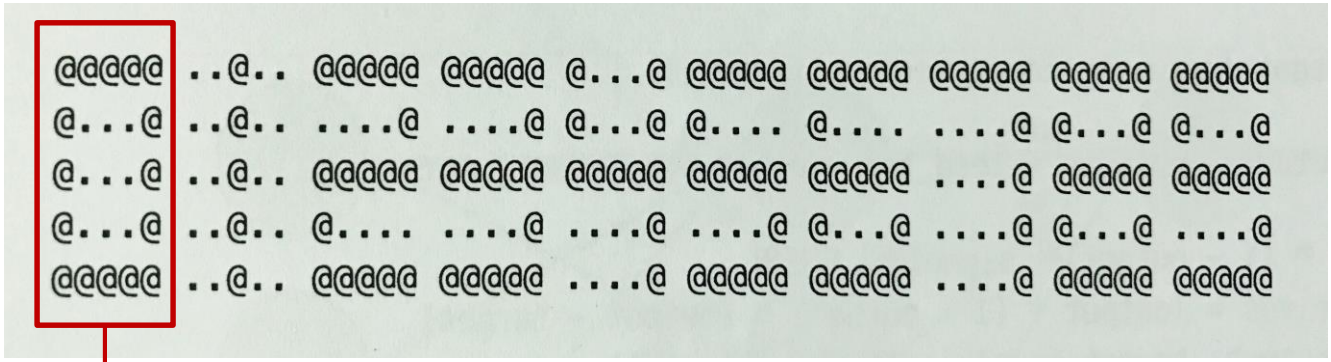
- 입력으로 들어오는 숫자는 아래와 같이 각각 5X5 사이즈의 이미지로 되어 있다고 가정



05 실습 – CAPTCHA 깨기

1. 입력값과 출력값 list 만들기

- 입력으로 들어오는 숫자는 아래와 같이 각각 5X5 사이즈의 이미지로 되어 있다고 가정



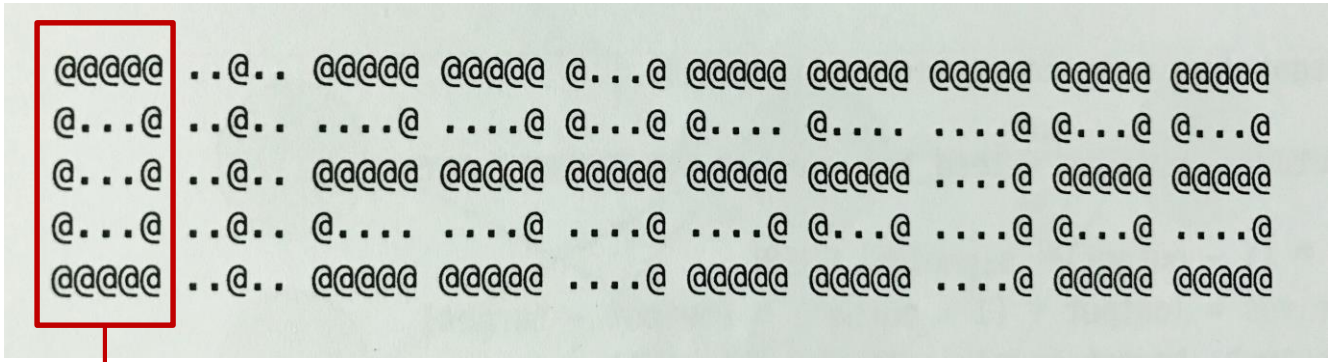

```
[1,1,1,1,1,
 1,0,0,0,1,
 1,0,0,0,1,
 1,0,0,0,1,
 1,1,1,1,1]
```

- 파이썬이 처리할 수 있도록 각 숫자들을 길이가 25이고 각 원소의 값이 1 또는 0인 벡터로 변환하자.
- 값이 1일 때는 해당 픽셀이 이미지에 포함되었다는 뜻
- 이것이 개별 input의 형태!

05 실습 – CAPTCHA 깨기

1. 입력값과 출력값 list 만들기

- 입력으로 들어오는 숫자는 아래와 같이 각각 5X5 사이즈의 이미지로 되어 있다고 가정




```
[1,1,1,1,1,
 1,0,0,0,1,
 1,0,0,0,1,
 1,0,0,0,1,
 1,1,1,1,1]
```

- 파이썬이 처리할 수 있도록 각 숫자들을 길이가 25이고 각 원소의 값이 1 또는 0인 벡터로 변환하자.
- 값이 1일 때는 해당 픽셀이 이미지에 포함되었다는 뜻
- 이것이 개별 input의 형태!

- Input들의 list인 inputs를 만든다

05 실습 – CAPTCHA 깨기

1. 입력값과 출력값 list 만들기

- Input들의 list인 inputs를 만든다 (...노가다;;)
- 이것이 **학습 데이터**인 셈

복붙하세요 →

```
inputs = [[1,1,1,1,1,
          1,0,0,0,1,
          1,0,0,0,1,
          1,0,0,0,1,
          1,1,1,1,1],
         [0,0,1,0,0,
          0,0,1,0,0,
          0,0,1,0,0,
          0,0,1,0,0,
          0,0,1,0,0],
         [1,1,1,1,1,
          0,0,0,0,1,
          1,1,1,1,1,
          1,0,0,0,0,
          1,1,1,1,1],
         [1,1,1,1,1,
          0,0,0,0,1,
          1,1,1,1,1,
          0,0,0,0,1,
          1,1,1,1,1],
         [1,0,0,0,1,
          1,0,0,0,1,
          1,1,1,1,1,
          0,0,0,0,1,
          0,0,0,0,1],
         [1,1,1,1,1,
          1,0,0,0,0,
          1,1,1,1,1,
          0,0,0,0,1,
          1,1,1,1,1],
         [1,1,1,1,1,
          1,0,0,0,0,
          1,1,1,1,1,
          1,0,0,0,1,
          1,1,1,1,1],
         [1,1,1,1,1,
          0,0,0,0,1,
          0,0,0,0,1,
          0,0,0,0,1,
          0,0,0,0,1],
         [1,1,1,1,1,
          1,0,0,0,1,
          1,1,1,1,1,
          1,0,0,0,1,
          1,1,1,1,1],
         [1,1,1,1,1,
          1,0,0,0,1,
          1,1,1,1,1,
          0,0,0,0,1,
          1,1,1,1,1]]
```

05 실습 – CAPTCHA 깨기

1. 입력값과 출력값 list 만들기

- Input들의 list인 inputs를 만든다 (...노가다;;)
- 이것이 **학습 데이터**인 셈

복붙하세요 →

```
inputs = [[1,1,1,1,1,
           1,0,0,0,1,
           1,0,0,0,1,
           1,0,0,0,1,
           1,1,1,1,1],
          [0,0,1,0,0,
           0,0,1,0,0,
           0,0,1,0,0,
           0,0,1,0,0,
           0,0,1,0,0],
          [1,1,1,1,1,
           0,0,0,0,1,
           1,1,1,1,1,
           1,0,0,0,0,
           1,1,1,1,1],
          [1,1,1,1,1,
           0,0,0,0,1,
           1,1,1,1,1,
           0,0,0,0,1,
           1,1,1,1,1],
          [1,0,0,0,1,
           1,0,0,0,1,
           1,1,1,1,1,
           0,0,0,0,1,
           0,0,0,0,1],
          [1,1,1,1,1,
           1,0,0,0,0,
           1,1,1,1,1,
           0,0,0,0,1,
           1,1,1,1,1],
          [1,1,1,1,1,
           1,0,0,0,0,
           1,1,1,1,1,
           1,0,0,0,1,
           1,1,1,1,1],
          [1,1,1,1,1,
           0,0,0,0,1,
           0,0,0,0,1,
           0,0,0,0,1,
           0,0,0,0,1],
          [1,1,1,1,1,
           1,0,0,0,1,
           1,1,1,1,1,
           1,0,0,0,1,
           1,1,1,1,1],
          [1,1,1,1,1,
           1,0,0,0,1,
           1,1,1,1,1,
           0,0,0,0,1,
           1,1,1,1,1]]
```

One-hot encoding

- 10개의 입력값에 대한 출력값이 필요
- 숫자 4에 대한 출력값은 [0, 0, 0, 0, 1, 0, 0, 0, 0, 0]인 식이다.

```
targets = [[1 if i == j else 0 for i in range(10)]
           for j in range(10)]
```

- range(10)은 0부터 10 미만의 숫자를 포함하는 range 객체를 만들어 줌

05 실습 – CAPTCHA 깨기

1. 입력값과 출력값 list 만들기

- Input들의 list인 inputs를 만든다 (...노가다;;)
- 이것이 **학습 데이터**인 셈

복붙하세요 →

```
inputs = [[1,1,1,1,1,
           1,0,0,0,1,
           1,0,0,0,1,
           1,0,0,0,1,
           1,1,1,1,1],
          [0,0,1,0,0,
           0,0,1,0,0,
           0,0,1,0,0,
           0,0,1,0,0,
           0,0,1,0,0],
          [1,1,1,1,1,
           0,0,0,0,1,
           1,1,1,1,1,
           1,0,0,0,0,
           1,1,1,1,1],
          [1,1,1,1,1,
           0,0,0,0,1,
           1,1,1,1,1,
           0,0,0,0,1,
           1,1,1,1,1],
          [1,0,0,0,1,
           1,0,0,0,1,
           1,1,1,1,1,
           0,0,0,0,1,
           0,0,0,0,1],
          [1,1,1,1,1,
           1,0,0,0,0,
           1,1,1,1,1,
           0,0,0,0,1,
           1,1,1,1,1],
          [1,1,1,1,1,
           1,0,0,0,0,
           1,1,1,1,1,
           1,0,0,0,1,
           1,1,1,1,1],
          [1,1,1,1,1,
           0,0,0,0,1,
           0,0,0,0,1,
           0,0,0,0,1,
           0,0,0,0,1],
          [1,1,1,1,1,
           1,0,0,0,1,
           1,1,1,1,1,
           1,0,0,0,1,
           1,1,1,1,1],
          [1,1,1,1,1,
           1,0,0,0,0,
           1,1,1,1,1,
           0,0,0,0,1,
           1,1,1,1,1]]
```

- 10개의 입력값에 대한 출력값이 필요
- 숫자 4에 대한 출력값은 [0, 0, 0, 0, 1, 0, 0, 0, 0, 0]인 식이다.

```
targets = [[1 if i == j else 0 for i in range(10)]
           for j in range(10)]
```

- range(10)은 0부터 10 미만의 숫자를 포함하는 range 객체를 만들어 줌

```
print(targets)
```

```
[[1, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 1, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 1, 0, 0, 0, 0,
0, 0, 0], [0, 0, 0, 1, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 1, 0, 0, 0, 0, 0], [0, 0, 0, 0,
0, 1, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 1, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 1, 0, 0], [0,
0, 0, 0, 0, 0, 0, 0, 1, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 1]]
```

05 실습 – CAPTCHA 깨기

1. 입력값과 출력값 list 만들기

- Input들의 list인 inputs를 만든다 (...노가다;;)
- 이것이 **학습 데이터**인 셈

복붙하세요 →

```
inputs = [[1,1,1,1,1,
1,0,0,0,1,
1,0,0,0,1,
1,0,0,0,1,
1,1,1,1,1],
[0,0,1,0,0,
0,0,1,0,0,
0,0,1,0,0,
0,0,1,0,0,
0,0,1,0,0],
[1,1,1,1,1,
0,0,0,0,1,
1,1,1,1,1,
1,0,0,0,0,
1,1,1,1,1],
[1,1,1,1,1,
0,0,0,0,1,
1,1,1,1,1,
0,0,0,0,1,
1,1,1,1,1],
[1,0,0,0,1,
1,0,0,0,1,
1,1,1,1,1,
0,0,0,0,1,
0,0,0,0,1],
[1,1,1,1,1,
1,0,0,0,0,
1,1,1,1,1,
0,0,0,0,1,
1,1,1,1,1],
[1,1,1,1,1,
1,0,0,0,0,
1,1,1,1,1,
1,0,0,0,1,
1,1,1,1,1],
[1,1,1,1,1,
0,0,0,0,1,
0,0,0,0,1,
0,0,0,0,1,
0,0,0,0,1],
[1,1,1,1,1,
1,0,0,0,1,
1,1,1,1,1,
1,0,0,0,1,
1,1,1,1,1],
[1,1,1,1,1,
1,0,0,0,1,
1,1,1,1,1,
0,0,0,0,1,
1,1,1,1,1]]
```

- range(10)은 0부터 10 미만의 숫자를 포함하는 range 객체를 만들어 줌

```
targets = [[1 if i == j else 0 for i in range(10)]
           for j in range(10)]
```

- e.g.)targets[4]: 숫자 4의(정확한) 이미지를 입력 받았을 때 출력값

```
print(targets[4])
```

```
[0, 0, 0, 0, 1, 0, 0, 0, 0, 0]
```

05 실습 – CAPTCHA 깨기

1. 입력값과 출력값 list 만들기

- Input들의 list인 inputs를 만든다 (...노가다;;)
- 이것이 **학습 데이터**인 셈

복붙하세요 →

```
inputs = [[1,1,1,1,1,
1,0,0,0,1,
1,0,0,0,1,
1,0,0,0,1,
1,1,1,1,1],
[0,0,1,0,0,
0,0,1,0,0,
0,0,1,0,0,
0,0,1,0,0,
0,0,1,0,0],
[1,1,1,1,1,
0,0,0,0,1,
1,1,1,1,1,
1,0,0,0,0,
1,1,1,1,1],
[1,1,1,1,1,
0,0,0,0,1,
1,1,1,1,1,
0,0,0,0,1,
1,1,1,1,1],
[1,0,0,0,1,
1,0,0,0,1,
1,1,1,1,1,
0,0,0,0,1,
0,0,0,0,1],
[1,1,1,1,1,
1,0,0,0,0,
1,1,1,1,1,
0,0,0,0,1,
1,1,1,1,1],
[1,1,1,1,1,
1,0,0,0,0,
1,1,1,1,1,
1,0,0,0,1,
1,1,1,1,1],
[1,1,1,1,1,
0,0,0,0,1,
0,0,0,0,1,
0,0,0,0,1,
0,0,0,0,1],
[1,1,1,1,1,
1,0,0,0,1,
1,1,1,1,1,
1,0,0,0,1,
1,1,1,1,1],
[1,1,1,1,1,
1,0,0,0,1,
1,1,1,1,1,
0,0,0,0,1,
1,1,1,1,1]]
```

- 10개의 입력값에 대한 출력값이 필요
- 숫자 4에 대한 출력값은 [0, 0, 0, 0, 1, 0, 0, 0, 0, 0]인 식이다.

```
targets = [[1 if i == j else 0 for i in range(10)]
           for j in range(10)]
```

▶주의

```
targets = [1 if i == j else 0 for i in range(10) for j in range(10)]
print(targets)
```

```
[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 1]
```

05 실습 – CAPTCHA 깨기

2. 은닉층과 출력층 설계

- 나중에 backpropagation으로 반복 학습시킬 것이므로 우선 각 뉴런의 weight와 bias를 임의로 설정한다
- random 모듈: 난수를 생성해준다

```
random.seed(0)
```

매번 동일한 결과를 얻도록 시드 설정

05 실습 – CAPTCHA 깨기

2. 은닉층과 출력층 설계

- 나중에 backpropagation으로 반복 학습시킬 것이므로 우선 각 뉴런의 weight와 bias를 임의로 설정한다
- random 모듈: 난수를 생성해준다

```
random.seed(0)
```

매번 동일한 결과를 얻도록 시드 설정

```
print(random.random())
print(random.random())
```

```
0.4098806093887075
0.6112019178475728
```

```
print(random.random())
print(random.random())
```

```
0.8057382560448987
0.8722274929743393
```

```
random.seed(0)
print(random.random())
print(random.random())
```

```
0.8444218515250481
0.7579544029403025
```

```
0.8444218515250481
0.7579544029403025
```

```
random.seed()
print(random.random())
```

```
0.672198015230881
```

```
0.7600232582181359
```

05 실습 – CAPTCHA 깨기

2. 은닉층과 출력층 설계

- 나중에 backpropagation으로 반복 학습시킬 것이므로 우선 각 뉴런의 weight와 bias를 임의로 설정한다
- random 모듈: 난수를 생성해준다

```
random.seed(0)
```

매번 동일한 결과를 얻도록 시드 설정

```
input_size = 25
num_hidden = 5
output_size = 10
```

입력 변수와 출력 변수의 길이는 앞서 설정한 바와 같고,
은닉층은 5개 뉴런으로 구성된다고 설정함

05 실습 – CAPTCHA 깨기

2. 은닉층과 출력층 설계

- 나중에 backpropagation으로 반복 학습시킬 것이므로 우선 각 뉴런의 weight와 bias를 임의로 설정한다
- random 모듈: 난수를 생성해준다

```
random.seed(0)
```

매번 동일한 결과를 얻도록 시드 설정

```
input_size = 25
num_hidden = 5
output_size = 10
```

입력 변수와 출력 변수의 길이는 앞서 설정한 바와 같고,
은닉층은 5개 뉴런으로 구성된다고 설정함

Input_size = 25 인 이유 : 숫자를 5 * 5의 픽셀로 생각
Output_size = 10 인 이유 : (숫자 4에 대한 출력값예시 참조)

05 실습 – CAPTCHA 깨기

2. 은닉층과 출력층 설계

- 나중에 backpropagation으로 반복 학습시킬 것이므로 우선 각 뉴런의 weight와 bias를 임의로 설정한다
- random 모듈: 난수를 생성해준다

```
random.seed(0)
```

매번 동일한 결과를 얻도록 시드 설정

```
input_size = 25
num_hidden = 5
output_size = 10
```

입력 변수와 출력 변수의 길이는 앞서 설정한 바와 같고,
은닉층은 5개 뉴런으로 구성된다고 설정함

은닉층의 각 뉴런은 각 입력 값에 대한 weight와 bias를 갖고 있음

```
hidden_layer = [[random.random() for _ in range(input_size + 1)]
                for _ in range(num_hidden)]
```

05 실습 – CAPTCHA 깨기

2. 은닉층과 출력층 설계

- 나중에 backpropagation으로 반복 학습시킬 것이므로 우선 각 뉴런의 weight와 bias를 임의로 설정한다
- random 모듈: 난수를 생성해준다

은닉층의 각 뉴런은 각 입력 값에 대한 weight와 bias를 갖고 있음

```
hidden_layer = [[random.random() for _ in range(input_size + 1)]
                for _ in range(num_hidden)]
```

'Weight(=input의 크기) + bias' 크기

Ex.

```
xor_network = [
    [[20, 20, -30], #hidden layer
    [20, 20, -10]],
    [[-60, 60, -30]]] #output layer

xor_network
```

05 실습 – CAPTCHA 깨기

2. 은닉층과 출력층 설계

- 나중에 backpropagation으로 반복 학습시킬 것이므로 우선 각 뉴런의 weight와 bias를 임의로 설정한다
- random 모듈: 난수를 생성해준다

```
random.seed(0)
```

매번 동일한 결과를 얻도록 시드 설정

```
input_size = 25
num_hidden = 5
output_size = 10
```

입력 변수와 출력 변수의 길이는 앞서 설정한 바와 같고,
은닉층은 5개 뉴런으로 구성된다고 설정함

은닉층의 각 뉴런은 각 입력 값에 대한 weight와 bias를 갖고 있음

```
hidden_layer = [[random.random() for _ in range(input_size + 1)]
                for _ in range(num_hidden)]
```

출력층의 각 뉴런은 각 은닉층의 뉴런에 대한 weight와 bias를 갖고 있음

```
output_layer  = [[random.random() for _ in range(num_hidden + 1)]
                 for _ in range(output_size)]
```

05 실습 – CAPTCHA 깨기

2. 은닉층과 출력층 설계

- 나중에 backpropagation으로 반복 학습시킬 것이므로 우선 각 뉴런의 weight와 bias를 임의로 설정한다
- random 모듈: 난수를 생성해준다

```
random.seed(0)
```

매번 동일한 결과를 얻도록 시드 설정

```
input_size = 25
num_hidden = 5
output_size = 10
```

입력 변수와 출력 변수의 길이는 앞서 설정한 바와 같고,
은닉층은 5개 뉴런으로 구성된다고 설정함

은닉층의 각 뉴런은 각 입력 값에 대한 weight와 bias를 갖고 있음

```
hidden_layer = [[random.random() for _ in range(input_size + 1)]
                for _ in range(num_hidden)]
```

출력층의 각 뉴런은 각 은닉층의 뉴런에 대한 weight와 bias를 갖고 있음

```
output_layer  = [[random.random() for _ in range(num_hidden + 1)]
                 for _ in range(output_size)]
```

```
network = [hidden_layer, output_layer]
```

이 신경망은 초기에 임의의 weight로 시작

05 실습 – CAPTCHA 깨기

3. backpropagation

- 앞서 정의한 함수 backpropagate 함수 이용
- 충분히 많이 반복하여 weight와 bias가 수렴하도록 함

```
for _ in range(10000):
    for input_vector, target_vector in zip(inputs, targets):
        backpropagate(network, input_vector, target_vector)
```

05 실습 – CAPTCHA 깨기

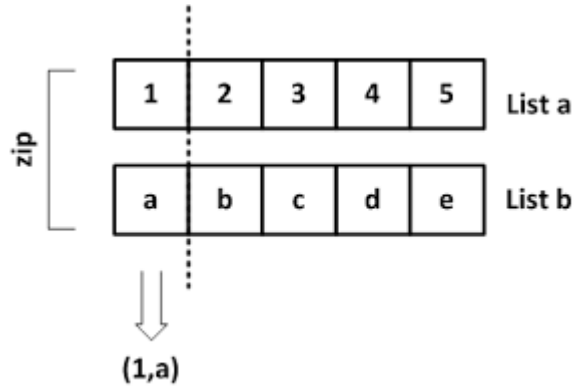
3. backpropagation

- 앞서 정의한 함수 backpropagate 함수 이용
- 충분히 많이 반복하여 weight와 bias가 수렴하도록 함

```
for _ in range(10000):
    for input_vector, target_vector in zip(inputs, targets):
        backpropagate(network, input_vector, target_vector)
```

```
a = [1,2,3,4,5]
b = ['a','b','c','d','e']

for x,y in zip (a,b):
    print (x,y)
```

```
1 a
2 b
3 c
4 d
5 e
```



05 실습 – CAPTCHA 깨기

3. backpropagation

- 앞서 정의한 함수 backpropagate 함수 이용
- 충분히 많이 반복하여 weight와 bias가 수렴하도록 함

```
for _ in range(10000):
    for input_vector, target_vector in zip(inputs, targets):
        backpropagate(network, input_vector, target_vector)
```

```
inputs = [[1,1,1,1,1,
           1,0,0,0,1,
           1,0,0,0,1,
           1,0,0,0,1,
           1,1,1,1,1],
          [0,0,1,0,0,
          0,0,1,0,0,
          0,0,1,0,0,
          0,0,1,0,0,
          0,0,1,0,0],
```

학습 [1, 0, 0, 0, 0, 0, 0, 0, 0, 0]

학습 [0, 1, 0, 0, 0, 0, 0, 0, 0, 0]

05 실습 – CAPTCHA 깨기

3. backpropagation

- 앞서 정의한 함수 backpropagate 함수 이용
- 충분히 많이 반복하여 weight와 bias가 수렴하도록 함

```
for _ in range(10000):
    for input_vector, target_vector in zip(inputs, targets):
        backpropagate(network, input_vector, target_vector)
```

```
def backpropagate(network, input_vector, targets)
```

05 실습 – CAPTCHA 깨기

4. predict함수 정의

- 앞서 정의한 함수 feed_forward를 이용해 predict 함수 정의

```
def predict(input):
    return feed_forward(network, input)[-1]
```

05 실습 – CAPTCHA 깨기

4. predict함수 정의

- 앞서 정의한 함수 feed_forward를 이용해 predict 함수 정의

```
def predict(input):
    return feed_forward(network, input)[-1]
```

```
predict(inputs[7])
```

```
[0.025293040717071412,
 1.6105842756391263e-05,
 1.4496195001149663e-10,
 0.017996127865903346,
 0.0008338720925808351,
 6.678999609815583e-10,
 2.9703032623788618e-08,
 0.9675758404419063,
 2.0420644649978637e-08,
 7.346756337389539e-08]
```

05 실습 – CAPTCHA 깨기

4. predict함수 정의

- 앞서 정의한 함수 feed_forward를 이용해 predict 함수 정의

```
def predict(input):
    return feed_forward(network, input)[-1]
```

```
predict([0,1,1,1,0,
         0,0,0,1,1,
         0,1,1,1,0,
         0,0,0,1,1,
         0,1,1,1,0])
```

```
[1.622475111166703e-07,
 9.379259432484909e-09,
 1.2872506068346948e-08,
 0.049726380599453036,
 5.215721492386159e-09,
 0.0035635187426415224,
 1.4339147274385235e-07,
 1.2431443116495094e-05,
 0.0015511494180063025,
 0.880016330182578]
```

05 실습 – CAPTCHA 깨기

4. predict함수 정의

- 한계: (솔직히) 정확하지 않다

05 실습 – CAPTCHA 깨기

4. predict함수 정의

- 한계: (솔직히) 정확하지 않다

```
predict([1,1,1,1,1,
         0,0,0,1,0,
         0,0,1,0,0,
         0,1,0,0,0,
         1,0,0,0,0])
```

```
[2.0749460096055353e-13,
 0.9610523563825943,
 0.0678425332755213,
 0.010757368124947621,
 0.004815204943377475,
 2.418697712177461e-05,
 3.0362886996831454e-08,
 9.194101370956578e-06,
 9.8953376511838e-12,
 3.806024928287174e-06]
```

05 실습 – CAPTCHA 깨기

4. predict함수 정의

- 보완점:
  - 학습 데이터의 크기를 더 크게 한다
  - 5 * 5 가 아닌, 7 * 7 등 더 큰 픽셀로 숫자를 이미지화 한다.

Thank you !