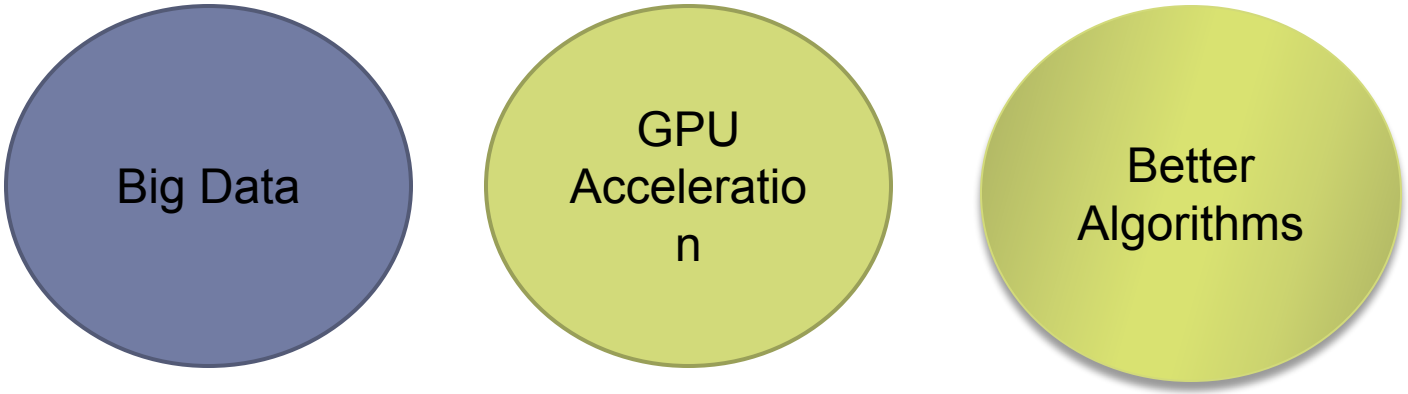
딥러닝

딥러닝이 주목받게 된 이유

- 많은 사람들이 딥러닝이 주목받은 이유를 3가지로 설명합니다.

1. 쉽게 빅데이터를 구할 수 있게 된 환경
2. GPU로 인한 컴퓨팅 파워의 발전
3. 새로운 딥러닝 알고리즘의 개발



딥러닝의 발전요인 - 빅데이터 / GPU / 알고리즘

빅데이터

• 머신러닝 알고리즘이 잘 학습되기 위해서는 대량의 데이터가 필요합니다.
• 인터넷이 등장하고 인간 행동의 많은 부분이 웹상에서 이루어지면서 대량의 데이터를 구하는 것이 쉬워졌고, 대량의 데이터, 빅데이터를 이용해서 머신러닝 알고리즘을 학습시킬 수 있는 환경이 만들어졌습니다.

GPU

• 빅데이터의 처리가 가능한 컴퓨터의 연산 속도가 뒷받침 되어야 합니다. 빅데이터를 처리하기 위해 1년 이상의 학습 시간이 필요하다면 아무리 성능이 좋은 알고리즘일지라도 현실의 문제를 해결하는데 무리가 있을 것입니다. 최근 10년 사이에 GPU를 이용한 병렬 연산 처리 기술이 발달하면서 대량의 데이터를 빠른 시간에 처리할 수 있게 되었습니다.

딥러닝이 좋은 성능을 보이는 이유

• 딥러닝 알고리즘은 어떻게 다른 머신 러닝 기법들 보다 좋은 성능을 보여줄 수 있을까? 딥러닝 알고리즘은 인공신경망을 깊게(Deep) 쌓아 올립니다. 깊게 쌓아 올림으로 얻는 효과는 **데이터의 특징을 단계별로 학습 할 수 있다**는 점입니다.

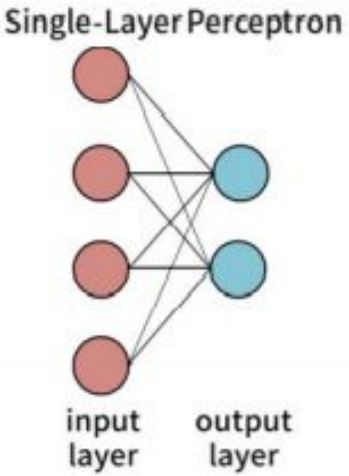
• 가장 대표적인 것이 이미지 분류 문제를 수행할 때 깊은 인공신경망의 각 층들이어떤 특징을 학습하는지 보여줍니다. 깊은 인공신경망의 낮은 층은 이미지의 픽셀, 선 등의 저차원 특징을 학습합니다. (CNN)

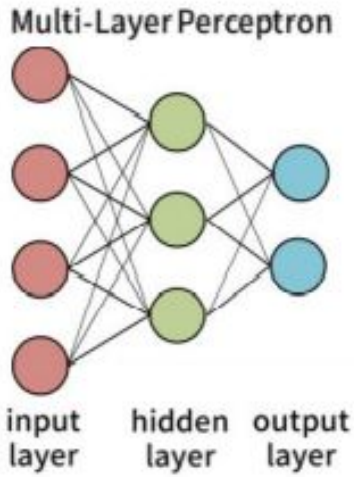
• 데이터의 특징을 단계별로 학습하기 때문에 딥러닝을 표현학습이라고도 부릅니다. 데이터의 특징을 잘 나타낼 수 있는 표현을 학습하는 것은 딥러닝과 머신러닝알고리즘의 핵심입니다. 데이터의 특징을 잘 학습하면 학습한 특징을 이용해서알고리즘이 더 좋은 성능을 낼 수 있습니다.

Artificial Neural Network (ANN)

Artificial Neural Network
Perceptron을 여러 개 연결한 것

Artificial Neural Network의 학습
우리가 원하는 목표를 달성하기 위한 weight값들을 찾아내는 과정

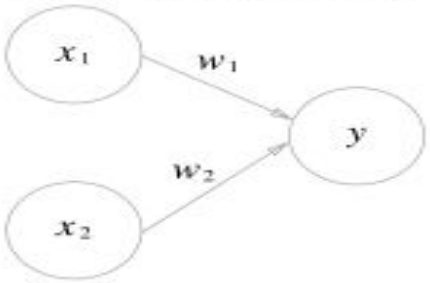




퍼셉트론

- 퍼셉트론이란
  - 퍼셉트론 은 다수의 신호를 입력으로 받아 하나의 신호를 출력.
  - 퍼셉트론 신호도 흐름을 만들고 정보를 앞으로 전달.
  - 다만, 실제 전류와 달리 퍼셉트론 신호는 '흐른다/안 흐른다(1 이나 0)'의 두 가지 값을 가진다.

그림 2-1 입력이 2개인 퍼셉트론



$$y = \begin{cases} 0\ (w_1x_1 + w_2x_2 \le \theta) \\ 1\ (w_1x_1 + w_2x_2 > \theta) \end{cases} \quad [식\ 2.1]$$

퍼셉트론

- 단순한 논리 회로

AND 게이트의 진리표

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

NAND 게이트의 진리표

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

OR 게이트의 진리표

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

NAND 게이트를 표현하려면
AND 게이트를 구현하는 매개변수의 부호를
모두 반전하기만 하면 NAND 게이트가 된다

퍼셉트론

- 퍼셉트론 구현하기
  - 매개변수 w1 , w2 , theta 는 함수 안에서 초기화
  - 가중치를 곱한 입력의 총합이 임계값을 넘으면 1 을 반환하고 그 외에는 0 을 반환.

```
def AND (x1,x2) :
   w1,w2,theta = 0.5,0.5,0.7
   tmp = x1*w1 + x2 * w2
   if tmp <= theta :
      return 0
   elif tmp > theta :
      return 1
print("AND(0,0)=",AND(0,0))
print("AND(0,1)=",AND(0,1))
print("AND(1,0)=",AND(1,0))
print("AND(1,1)=",AND(1,1))
```

```
AND(0,0)= 0
AND(0,1)= 0
AND(1,0)= 0
AND(1,1)= 1
```

퍼셉트론

- 가중치 와 편향
  - b 를 편향 bias 이라하며 w1 과 w2 는 가중치 weight 라 한다.

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

```
import numpy as np

x = np.array([0,1])
w = np.array([0.5,0.5])
b = -0.7
print(w*x)
print(np.sum(w*x))
print(np.sum(w*x) + b)
```

```
[0.  0.5]
0.5
-0.19999999999999996
```

퍼셉트론

- 가중치 와 편향
  - w1 과 w2 는 각 입력 신호가 결과에 주는 영향력을 조절하는 매개변수
  - 편향은 뉴런이 얼마나 쉽게 활성화(결과로 1 을 출력) 하느냐를 조정하는 매개변수.

```
import numpy as np
def AND(x1,x2) :
   x = np.array([x1,x2])
   w = np.array([0.5,0.5])
   b = -0.7
   tmp = np.sum(w*x) + b
   if tmp <= 0 :
      return 0
   else :
      return 1
```

```
print("AND(0,0)=",AND(0,0))
print("AND(0,1)=",AND(0,1))
print("AND(1,0)=",AND(1,0))
print("AND(1,1)=",AND(1,1))
```

```
AND(0,0)= 0
AND(0,1)= 0
AND(1,0)= 0
AND(1,1)= 1
```

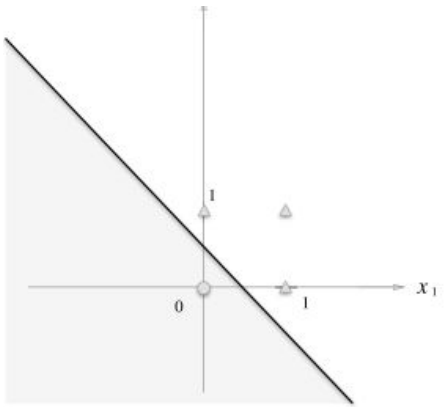
퍼셉트론

- 퍼셉트론의 한계
  - OR 게이트는 가중치 매개변수가 (b , w 1 , w 2) = (-0.5 , 1.0 , 1.0)일 때

OR 게이트의 진리표

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

$$y = \begin{cases} 0\ (-0.5 + x_1 + x_2 \le 0) \\ 1\ (-0.5 + x_1 + x_2 > 0) \end{cases}$$

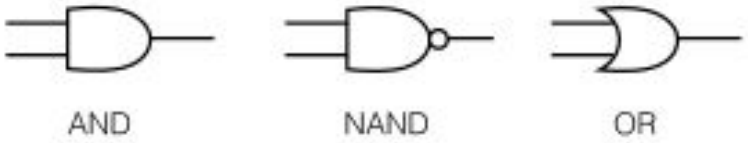


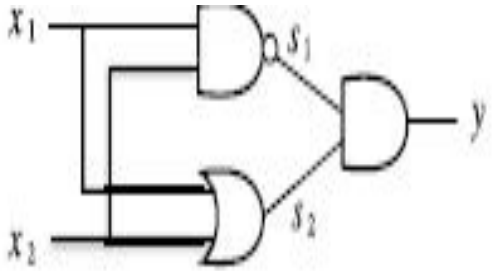
퍼셉트론의 시각화 : 회색 영역은 0 을 출력하는 영역이며, 전체 영역은 OR 게이트의 성질을 만족한다.

퍼셉트론

- 기존 게이트 *조합하기*
  - NAND 의 출력을 s1 , OR의 출력을 s2 로 해서 진리표를 XOR 를 만들 수 있다

AND , NAND , OR 게이트 기호





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

퍼셉트론

- XOR 게이트 구현하기

```
import numpy as np

def AND(x1,x2) :
    x = np.array([x1,x2])
    w = np.array([0.5,0.5])
    b = -0.7
    tmp = np.sum(w*x) + b
    if tmp <= 0 :
        return 0
    else :
        return 1

))
```

```
def OR(x1,x2) :
    x = np.array([x1,x2])
    w = np.array([1.0,1.0])
    b = -0.5
    tmp = np.sum(w*x) + b
    if tmp <= 0 :
        return 0
    else :
        return 1
```

퍼셉트론

- XOR 게이트 구현하기

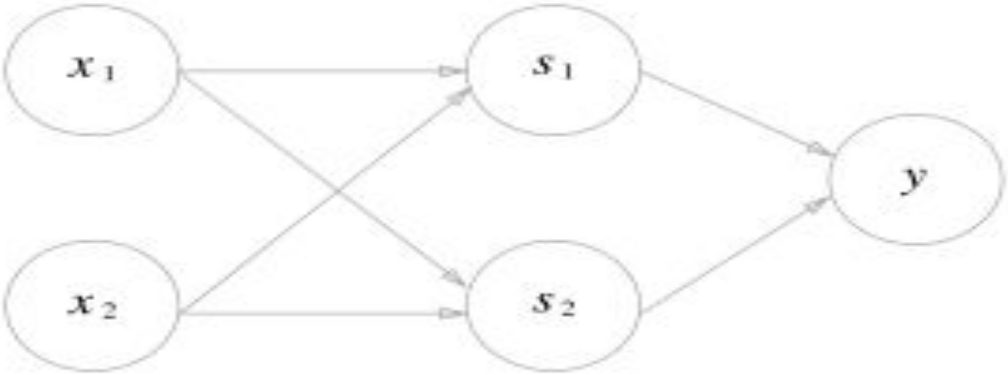
```
def NAND(x1,x2) :
    x = np.array([x1,x2])
    w = np.array([0.5,0.5])
    b = -0.7
    tmp = np.sum(w*x) + b
    if tmp <= 0 :
        return 1
    else :
        return 0

def XOR(x1,x2) :
    s1 = NAND(x1,x2)
    s2 = OR(x1,x2)
    y = AND(s1,s2)
    return y
```

```
print("XOR(0,0)=",XOR(0,0))
print("XOR(0,1)=",XOR(0,1))
print("XOR(1,0)=",XOR(1,0))
print("XOR(1,1)=",XOR(1,1))
```

```
XOR(0,0)= 0
XOR(0,1)= 1
XOR(1,0)= 1
XOR(1,1)= 0
```

퍼셉트론

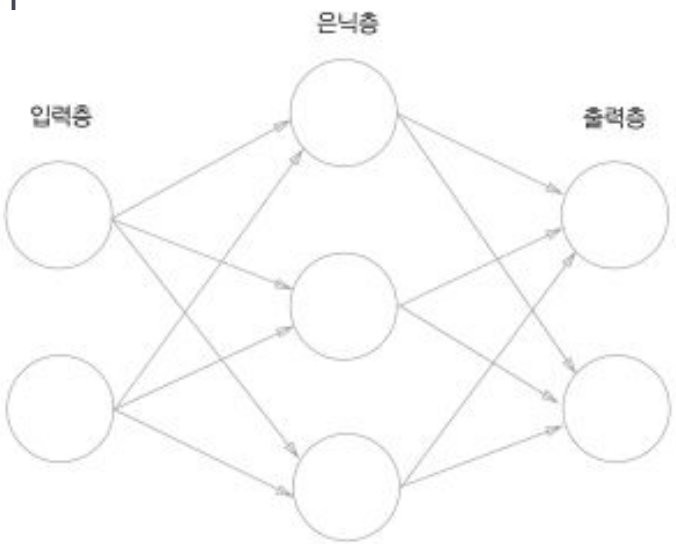
- XOR 게이트 구현하기



2 층 구조를 사용해 퍼셉트론으로 XOR 게이트를 구현할 수 있게 되었다. 이처럼 퍼셉트론은 층을 쌓아(깊게 하여) 더 다양한 것을 표현할 수 있다.

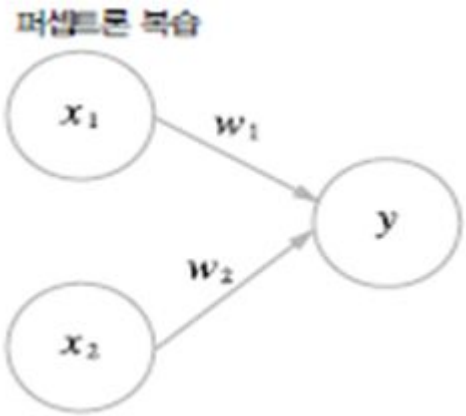
신경망

- 신경망
  - 가장 왼쪽 줄을 입력층 , 맨 오른쪽 줄을 출력층 , 중간 줄을 은닉층 이라고 함.
  - 은닉층의 뉴런은 사람 눈에는 보이지 않는다.
  - 입력층에서 출력층방향으로 차례로 0 층, 1 층, 2 층이라 하겠다'



신경망

- 활성화 함수
  - 입력 신호의 총합을 출력 신호로 변환하는 함수를 활성화 함수 activation function 라 한다.
  - '활성화'라는 이름이 말해주듯 활성화 함수는입력 신호의 총합이 활성화를 일으키는지를 정하는 역할
  - 가중치가 곱해진 입력 신호의 총합을 계산하고, 그합을 활성화 함수에 입력해 결과를 내는 2 단계로 처리 된다.



$$a = b + w_1 x_1 + w_2 x_2 \quad [식 3.4]$$

$$y = h(a) \quad [식 3.5]$$

신경망

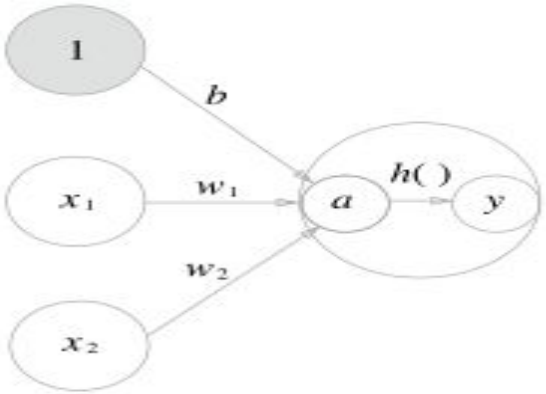
- 활성화 함수
  - 가중치가 달린 입력 신호와 편향의 총합을 계산하고, 이를 a 라 한다.
  - a 를 함수 h ()에 넣어 y 를 출력.

$$a = b + w_1 x_1 + w_2 x_2$$

$$y = h(a)$$

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

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



신경망

- 활성화 함수의 등장
  - 입력 신호의 총합을 출력 신호로 변환하는 함수를 일반적으로 활성화 함수activation function라 한다

왼쪽은 일반적인 뉴런,
오른쪽은 활성화 처리 과정을 명시한 뉴런
(*a*는 입력 신호의 총합, *h*( )는 활성화 함수,*y*는 출력)

신경망

- 시그모이드 함수

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

신경망에서는 활성화 함수로 시그모이드 함수를 이용하여
신호를 변환하고, 그 변환된 신호를다음 뉴런에 전달

신경망

- 계산 함수 구현하기
  - 넘파이 배열에 부등호 연산을 수행하면 배열의 원소 각각에 부등호 연산을 수행한 bool 배열이 생성.
  - 이 예에서는 배열 x의 원소 각각이 0보다 크면 True로, 0 이하면 False로 변환 한 새로운 배열 y가 생성

```
import numpy as np
x = np.array([-1.0,1.0,2.0])
print(x)
y = x > 0
print(y)
```

```
[-1.  1.  2.]
[False  True  True]
```

신경망

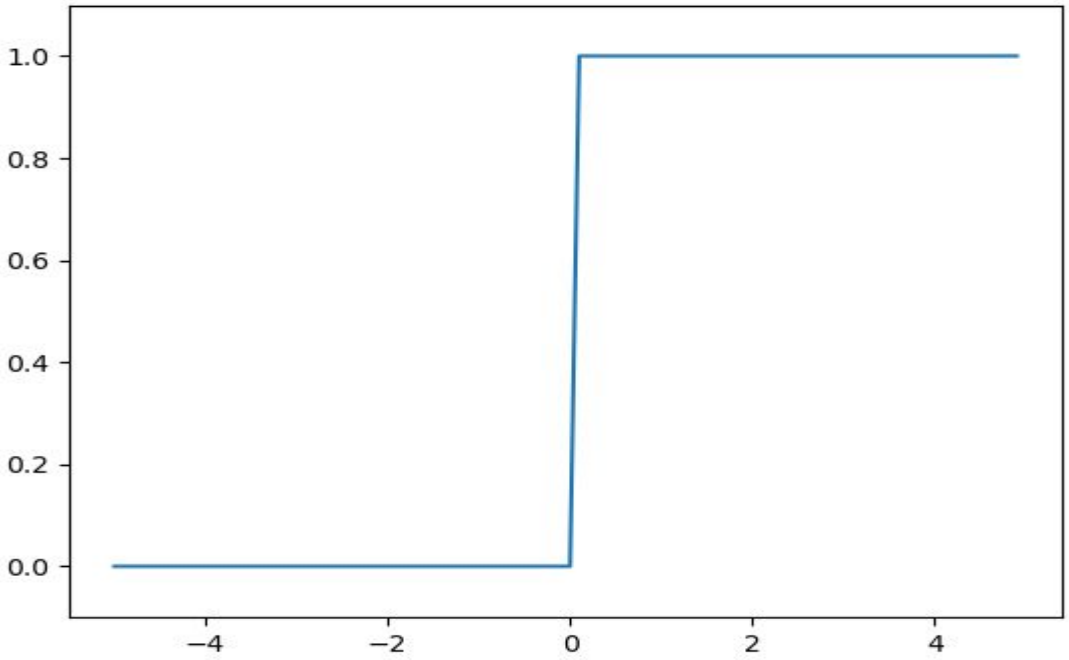
- 계단 함수의 그래프
  - np.arange(-5.0, 5.0, 0.1)은 -5.0에서 5.0 전까지 0.1 간격의 넘파이 배열을 생성.
  - [-5.0, -4.9, ..., 4.9]를 생성.step_function ( )은 인수로 받은 넘파이 배열의 원소 각각을 인수로 계단 함수 실행해, 그 결과를 다시 배열로 만들어 돌려준다

```
import numpy as np
import matplotlib.pyplot as plt

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)
print(y)
plt.plot(x,y)
plt.ylim(-0.1,1.1)
plt.show()
```

신경망

- 계단 함수의 그래프

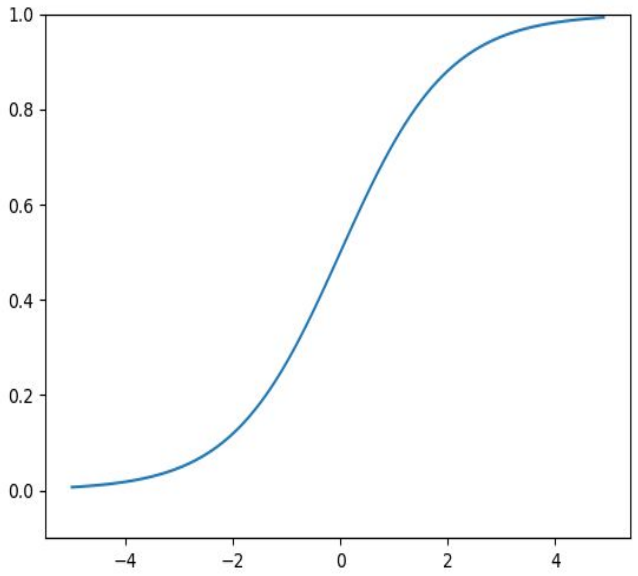


신경망

시그모이드 함수

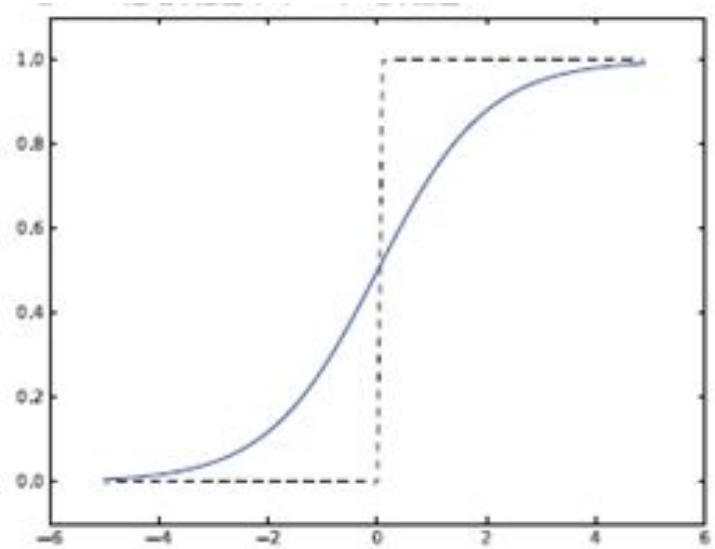
```
import numpy as np
import matplotlib.pyplot as plt

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

x = np.array([-1.0,1.0,2.0])
sigmoid(x)

t = np.array([1.0,2.0,3.0])
print(1.0 + t)
print(1.0/t)

x = np.arange(-5.0,5.0,0.1)
y = sigmoid(x)
plt.plot(x,y)
plt.ylim(-0.1,1,1)
plt.show()
```

구현한 sigmoid 함수에서도 np.exp(-x)가 넘파이 배열을 반환하기 때문에 1 / (1 + np.exp(-x))도 넘파이 배열의 각 원소에 연산을 수행한 결과를 내어 준다



신경망

- 시그모이드 함수와 계단 함수 비교
  - 시그모이드 함수는 부드러운 곡선이며 입력에 따라 출력이 연속적으로 변화.
  - 한편, 계단 함수는 0을 경계로 출력이 갑자기 바뀐다.
  - 시그모이드 함수의 이 매끈함이 신경망 학습에서 아주 중요

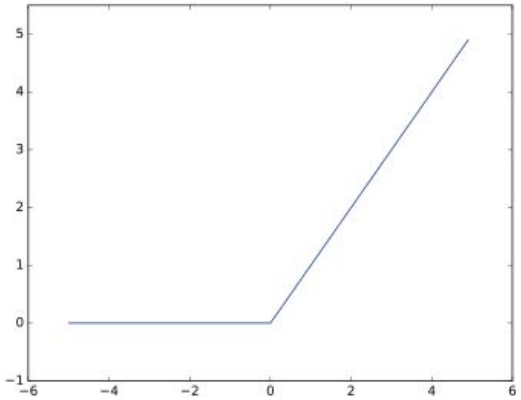


신경망

- 비선형 함수
  - 계단 함수와 시그모이드 함수의 중요한 공통점은 모두는비선형 함수.
    - 시그모이드 함수는 곡선, 계단 함수는 계단처럼 구부러진 직선으로 나타나며, 동시에 비선형 함수로 분류.
  - 활성화 함수를 설명할 때 비선형 함수와 선형 함수라는 용어가 자주 등장. 함수란 어떤 값을입력하면 그에 따른 값을 돌려주는 '변환기' 임.
  - 변환기에 무언가 입력했을 때 출력이 입력의 상수배 만큼 변하는 함수를 선형 함수라고 한다.
    - 수식으로는 f (x) = ax + b 이고, 이때 a 와 b 는 상수이다.
    - 선형 함수는 곧은 1 개의 직선이 된다.
  - 비선형 함수는 문자 그대로 '선형이 아닌' 함수입니다. 즉, 직선1 개로는 그릴 수 없는 함수를 말함.

신경망

- ReLU 함수
  - 넘파이의 maximum 함수를 사용.
  - maximum 은 두 입력 중 큰 값을 선택해 반환하는 함수.
  - 시그모이드 함수를 활성화 함수로 사용했으나, 나중에 주로ReLU 함수를 사용



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

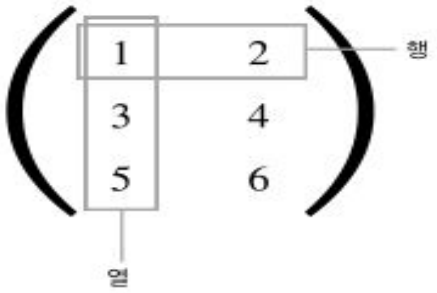
```
def relu(x) :
    return np.maximum(0,x)
```

신경망

다차원 배열

```
import numpy as np
A = np.array([1,2,3,4])
print(A)
np.ndim(A) #배열의 차원
A.shape  #배열의 형태. 튜플로 리턴

A.shape[0]

B = np.array([[1,2],[3,4],[5,6]])
print(B)
np.ndim(B)
B.shape
```

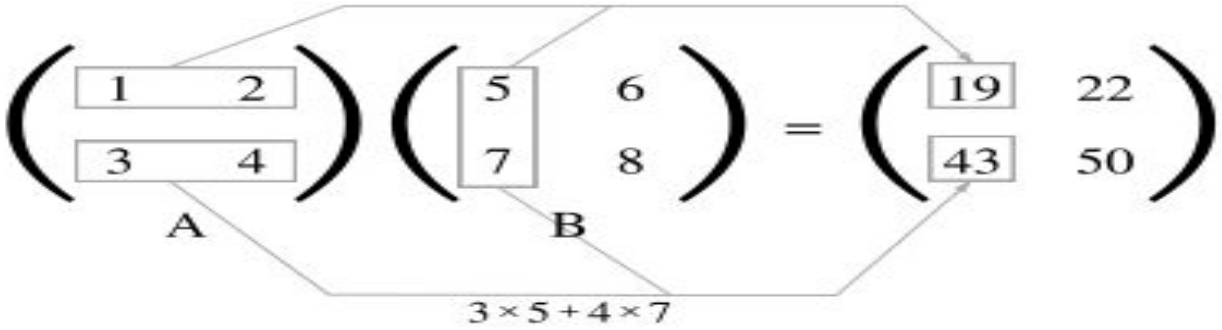
2 차원 배열(행렬)의 행(가로)과 열(세로)



파이썬의 인덱스는 0 부터 시작
2 차원 배열은 행렬 matrix 이라고 함
배열의 가로 방향을 행 row , 세로 방향을 열 column 이라고 한다

신경망

- 행렬의 곱

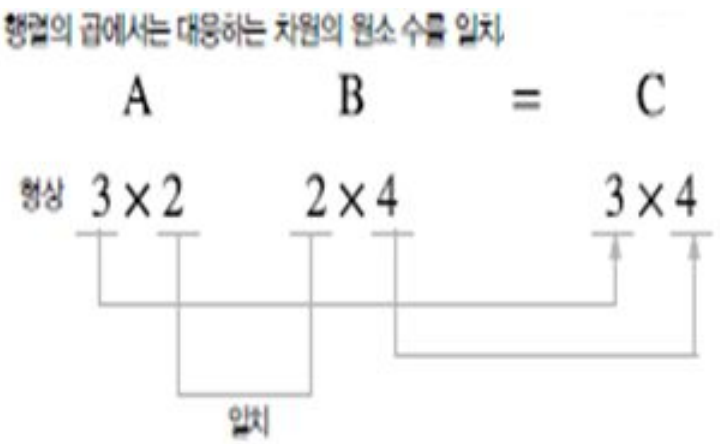



```
>>> A = np.array([[1,2], [3,4]])
>>> A.shape
(2, 2)
>>> B = np.array([[5,6], [7,8]])
>>> B.shape
(2, 2)
>>> np.dot(A, B)
array([[19, 22],
       [43, 50]])
```

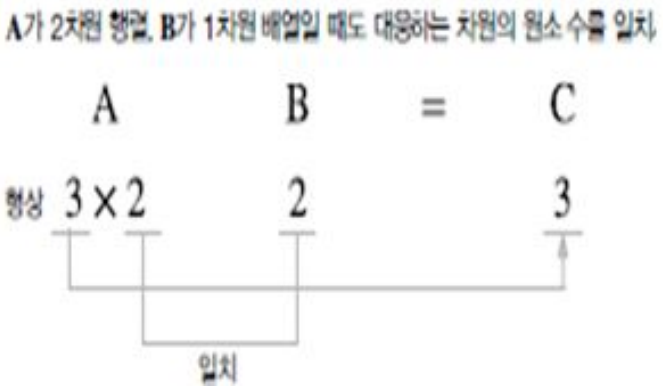
신경망

- 행렬의 곱

```
>>> A = np.array([[1,2,3], [4,5,6]])
>>> A.shape
(2, 3)
>>> B = np.array([[1,2], [3,4], [5,6]])
>>> B.shape
(3, 2)
>>> np.dot(A, B)
array([[22, 28],
       [49, 64]])
```

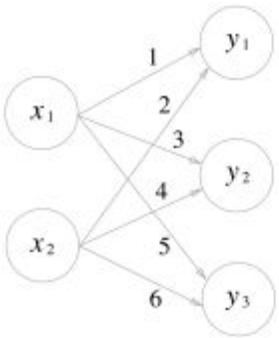



```
>>> C = np.array([[1,2], [3,4]])
>>> C.shape
(2, 2)
>>> A.shape
(2, 3)
>>> np.dot(A, C)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: shapes (2,3) and (2,2) not aligned: 3 (dim 1) != 2 (dim 0)
```



신경망

신경망에서의 행렬 곱



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

$$X \quad W \quad = \quad Y$$

$$2 \quad 2 \times 3 \quad 3$$

일치

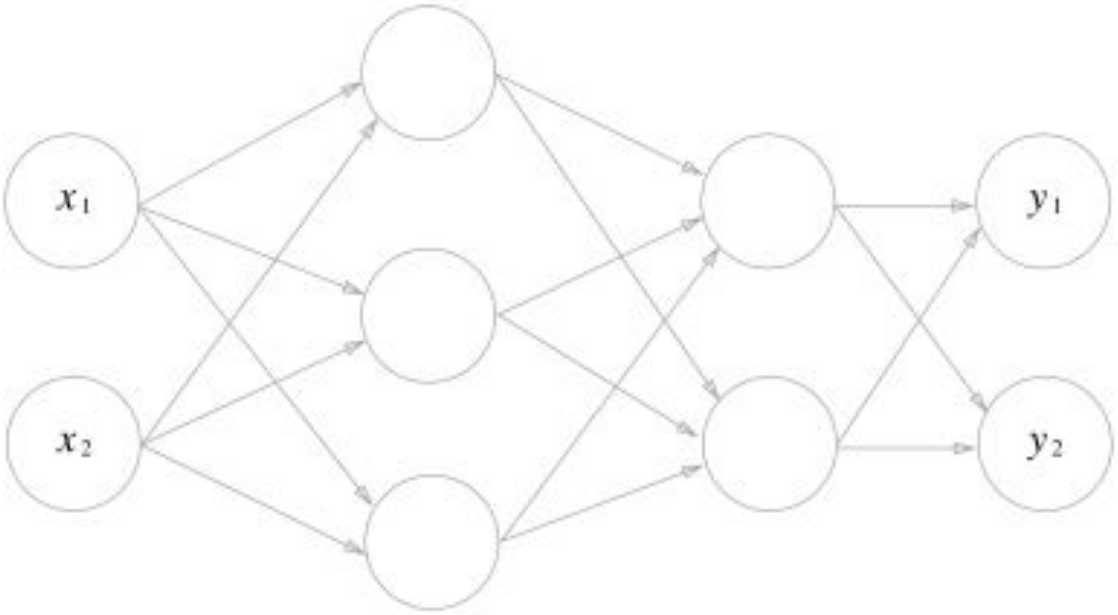
이 구현에서도 X , W , Y 의 형상을 주의해서 보세요. 특히 X 와 W 의 대응하는 차원의 원소 수가 같아야 한다

```
>>> X = np.array([1, 2])
>>> X.shape
(2,)
>>> W = np.array([[1, 3, 5], [2, 4, 6]])
>>> print(W)
[[1 3 5]
 [2 4 6]]
>>> W.shape
(2, 3)
>>> Y = np.dot(X, W)
>>> print(Y)
[ 5  11  17]
```

다차원 배열의 스칼라곱을 구해주는 np . dot 함수를 사용시 결과 Y 를 계산. Y 의 원소가 100 개든 1 , 000 개든 한 번의 연산으로 계산

신경망

- 신경망 구현하기



3 층 신경망 : 입력층(0 층)은 2 개, 첫 번째 은닉층(1 층)은 3 개, 두 번째 은닉층(2 층)은 2 개, 출력층(3 층)은 2개의 뉴런으로 구성