2022년 3월 2일 machine learning course 소개

기초 수학

- Linear algebra: strang
- Vector calculus : marsden
- Mathematical analysis : Rudin
- Statistics : Berger
- numerical/matrix computations : Burden, Golub
- Numerical optimization : Nocedal

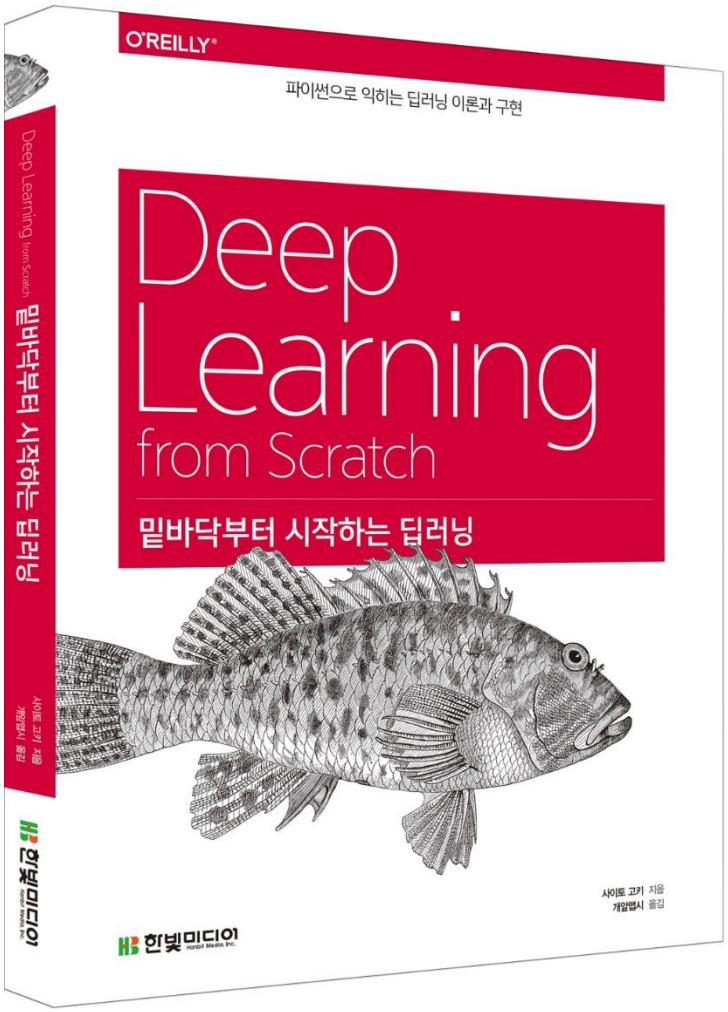
프로그래밍

- Python: 1학년
- 미적분, 선형대수, numpy, pandas, matplotlib, sklearn, http://www.scipy-lectures.org 전산수학 (2학년)
- machine learning algorithm: 머신러닝 (3학년 1학기)
- Tensorflow web page: 인공지능 (3학년 2학기)
- c++, cuda, image, 병렬분산 컴퓨팅 (4학년 2학기)

Projects

- Domain knowledge : speech recognition, image classification, language processing, medical application, law, etc.
- working knowledge, Papers
- open software, package

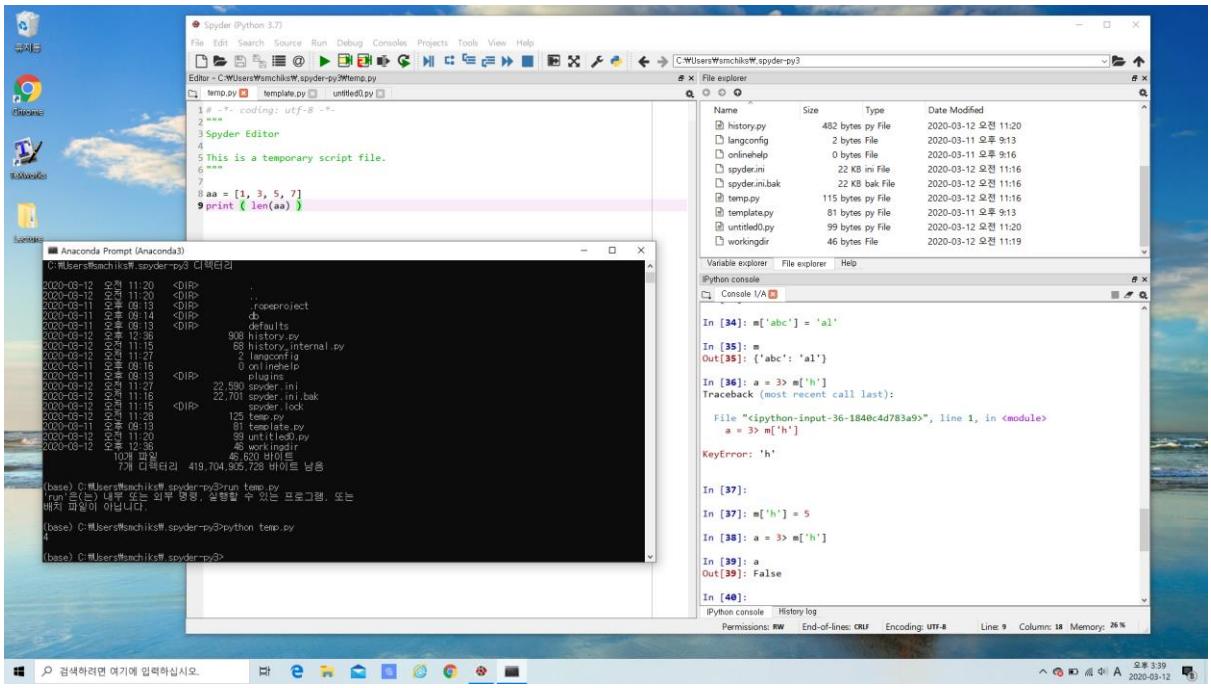
Deep Learning from scratch



실습 환경: https://www.anaconda.com/ 방문

소스코드: https://github.com/WegraLee/deep-learning-from-scratch

설치된 환경 보기



Python, numpy, scipy, http://www.scipy-lectures.org 복습

1.3 파이썬 인터프리터 : 가볍게 읽기

1.4 파이썬 스크립트 파일

1.4.2 클래스

```
class Man:
    def __init__(self, name):
        self.name = name
        print("Initilized!")

    def hello(self):
```

```
        print("Hello " + self.name + "!")

    def goodbye(self):
        print("Good-bye " + self.name + "!")

m = Man("David")
m.hello()
m.goodbye()
```

1.5 넘파이

numpy 의 배열 클래스 numpy.array

1.5.1, 1.5.2 넘파이 가져오기, 배열 생성하기

```
import numpy as np

x = np.array([1.0, 2.0, 3.0])

print (x)

print ( type(x) )
```

1.5.3 넘파이의 산술연산

```
import numpy as np

x = np.array([1.0, 2.0, 3.0])

y = np.array([2.0, 4.0, 6.0])

print (x+y, x-y, x*y, x/y)
```

1.5.4 넘파이의 N 차원 배열

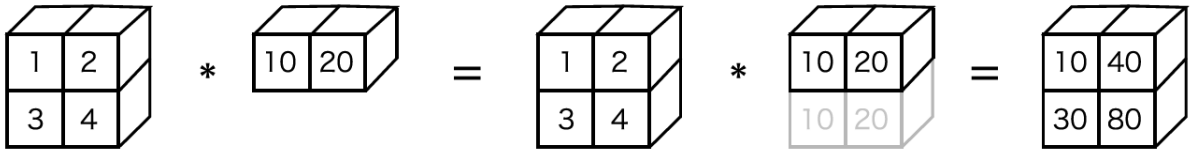
```
A = np.array([[1,2], [3,4]])
print (A, A.shape, A.dtype)
B = np.array([[3,0], [0,6]])
print (A+B)
print (A*B)
```

1 차원배열: 벡터, 2 차원 배열: 행렬, 일반화 텐서(tensor)

1.5.5 브로드캐스트 (broadcast)

형상이 다른 배열끼리 계산 가능: 단, 형상 확장 정보가 내재되야 함

```
A = np.array([[1,2], [3,4]])
B = np.array([10,20])
print (A*B)
```



1.5.6 원소 접근

인덱스는 0 부터

```
A = np.array([[51,55], [14,19], [0,4]])
print (A[0], A[0][1])

for row in A:
    print (row)

X = A.flatten()
print (X)
print (X[np.array([0,2,4])])

print (X>15)
print (X[X>15])
```

1.6 matplotlib

데이터 시각화

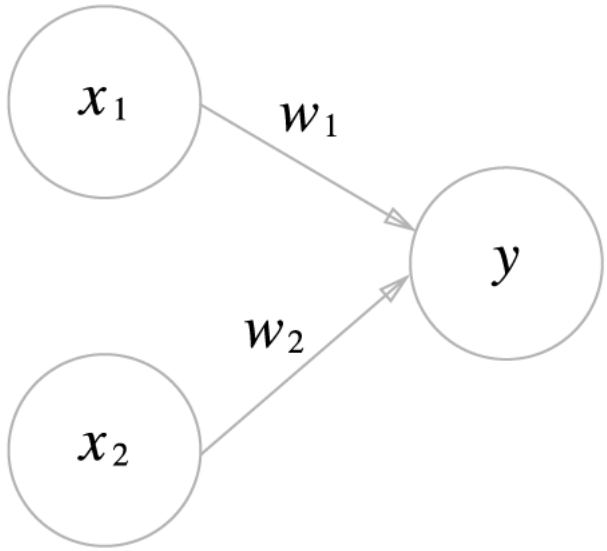
```
import numpy as np
import matplotlib.pyplot as plt
```

```
# 데이터 준비
x = np.arange(0, 6, 0.1) # 0 에서 6 까지 0.1 간격으로 생성
y1 = np.sin(x)
y2 = np.cos(x)
# 그래프 그리기
plt.plot(x, y1, label="sin")
plt.plot(x, y2, linestyle = "--", label="cos") # cos 함수는 점선
plt.xlabel("x") # x 축 이름
plt.ylabel("y") # y 축 이름
plt.title('sin & cos')
plt.legend()
plt.show()
```

이미지 표시

```
import matplotlib.pyplot as plt
from matplotlib.image import imread
img = imread('deep-learning-from-scratch-master/map.png') # 이미지 읽어오기
plt.imshow(img)
plt.show()
```

chapter 2　퍼셉트론 perceptron



x_1, x_2: 입력　y: 출력 w_1, w_2: 가중치

그림의 원을 뉴런, 노드

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

ϑ :임계값 세타 감각,신경 신호 인지/불인지, 합격/불합격, decision

가중치는 각 입력이 출력에 주는 영향력을 조절

2.2 단순한 논리 회로

2.2.1 AND 게이트

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

w_1, w_2, ϑ 조합으로 논리곱 표현 (0.5, 0.5, 0.7), (0.5, 0.5, 0.8), (1, 1, 1)

0.5*1 + 0.5*1 > 0.7, 0.5*0 + 0.5*1 < 0.7

이외에도 많은 조합 가능

2.2.2 NAND, OR

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

$(w_1, w_2, \vartheta) = (-0.5, -0.5, -0.7)$ 등 많은 조합

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

가중치는 어떻게? (1, 1, 0.5) (1, 2, 0.8)

적절한 가중치와 임계값으로 퍼셉트론이 논리회로 기능

신경망에서 이 값들을 정하는 작업이 학습

2.3 퍼셉트론 구현하기

2.3.1 간단한 구현부터: ϑ 가

2.3.2 가중치와 **편향** 도입 : b 로 변신 -> XW 행렬곱 구현

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

```
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


if __name__ == '__main__':
    for xs in [(0, 0), (1, 0), (0, 1), (1, 1)]:
        y = AND(xs[0], xs[1])
        print(str(xs) + " -> " + str(y))
```

예제: NAND, OR

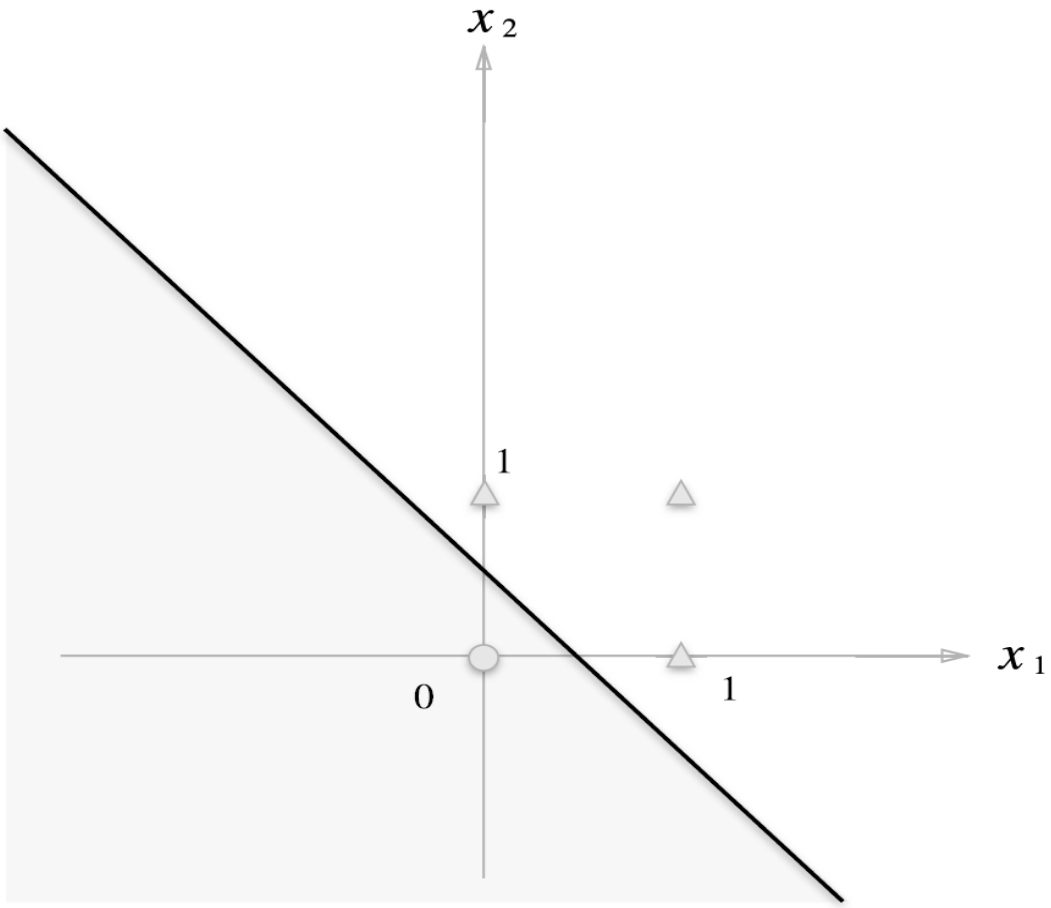
```
import numpy as np
def NAND(x1, x2):
    x = np.array([x1, x2])
    w = np.array([-0.5, -0.5])
    b = 0.7
    tmp = np.sum(w*x) + b
```

```
    if tmp <= 0:
        return 0
    else:
        return 1

def OR(x1, x2):
    x = np.array([x1, x2])
    w = np.array([0.5, 0.5])
    b = -0.2
    tmp = np.sum(w*x) + b
    if tmp <= 0:
        return 0
    else:
        return 1

if __name__ == '__main__':
    for xs in [(0, 0), (1, 0), (0, 1), (1, 1)]:
        y = NAND(xs[0], xs[1])
        print(str(xs) + " -> " + str(y))
    print()
    for xs in [(0, 0), (1, 0), (0, 1), (1, 1)]:
        y = OR(xs[0], xs[1])
            print(str(xs) + " -> " + str(y))
```

2.4 퍼셉트론의 한계

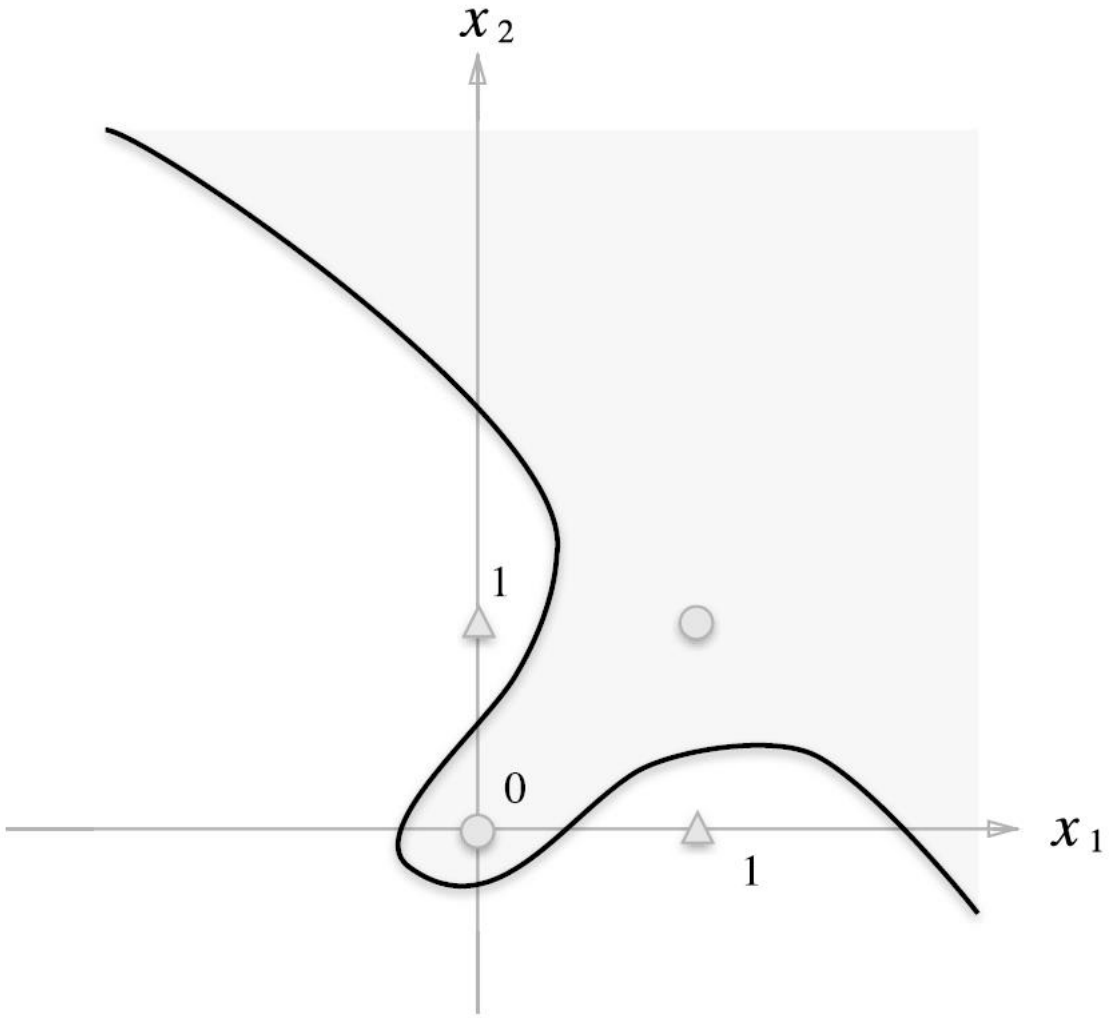
2.4.1 도전 XOR 게이트

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



2.4.2 선형과 비선형

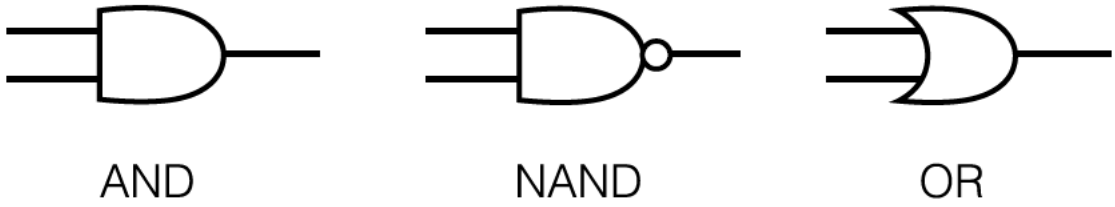
직선으로 나눌 수 없지만 곡선으로 나눌 수 있다.



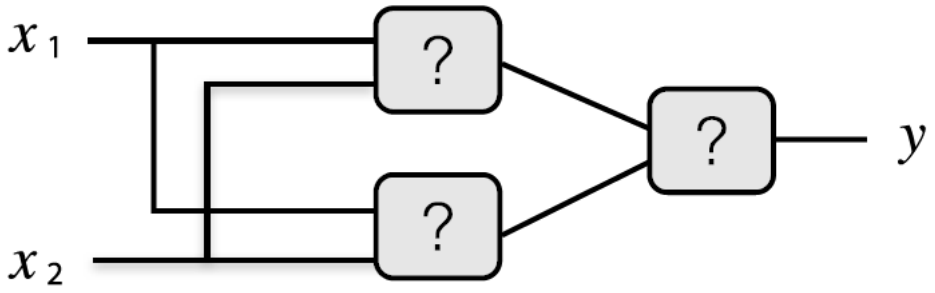
핵심은 직선 두개로 나눌 수 있다. -> 논리게이트 조합

2.5 다층 퍼셉트론이 출동한다면

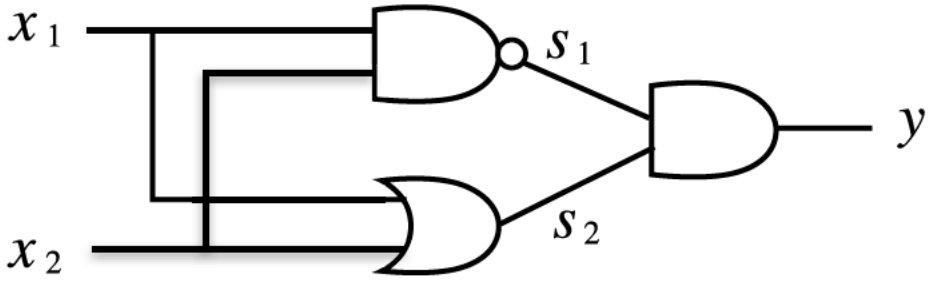
2.5.1 기존 게이트 조합하기



어떻게 조합하면 XOR



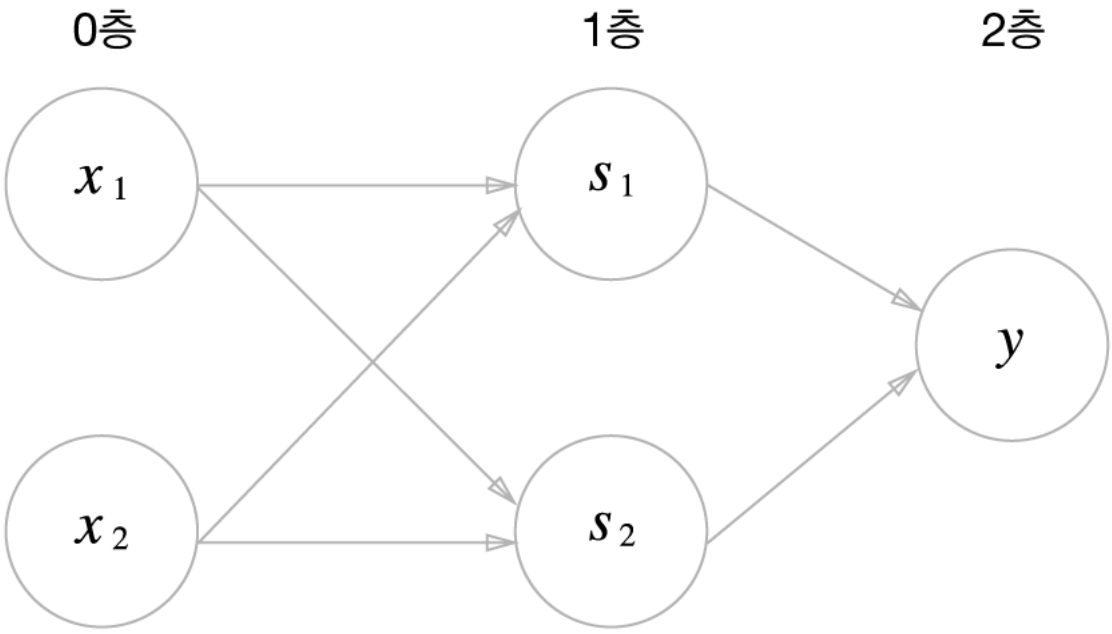
답



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

2.5.2 XOR 게이트 구현하기 : 디렉토리/import 주의

```
from and_gate import AND

from or_gate import OR

from nand_gate import NAND

def XOR(x1, x2):

    s1 = NAND(x1, x2)

    s2 = OR(x1, x2)

    y = AND(s1, s2)

    return y

if __name__ == '__main__':

    for xs in [(0, 0), (1, 0), (0, 1), (1, 1)]:

        y = XOR(xs[0], xs[1])

        print(str(xs) + " -> " + str(y))
```



단층 퍼셉트론으로 불가능한 것을 층을 하나 늘려 구현

2.6 NAND에서 컴퓨터까지

단순한 NAND소자에서 AND,OR 게이트,

➔ 반가산기, 전가산기

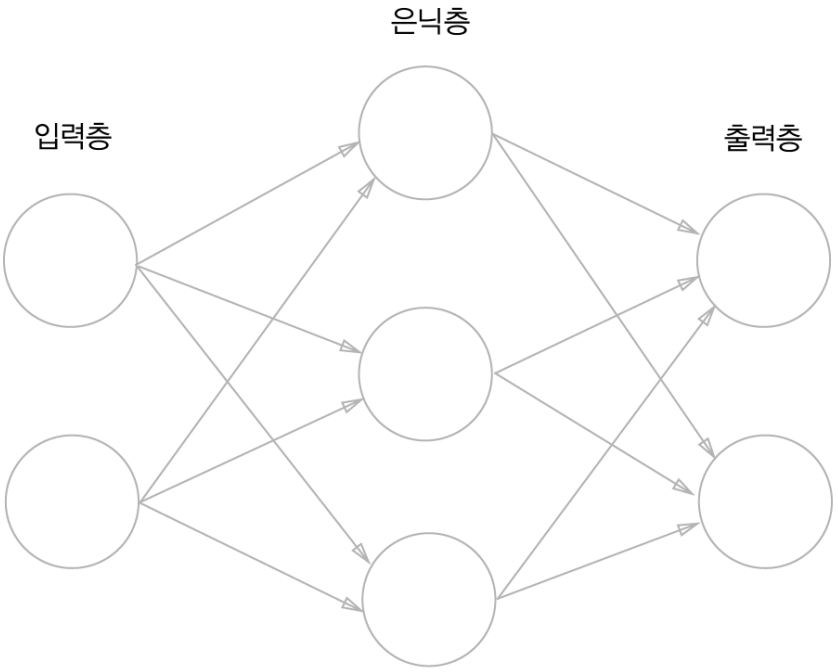
➔ 산술 논리 연산 장치, CPU

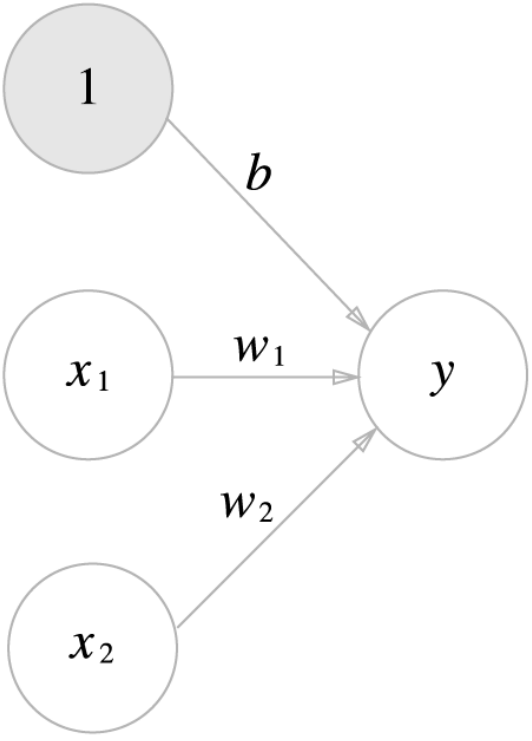
이책에서는 컴퓨터를 만들지는 않고, 뇌처리를 모방

3 월 16 일

chapter 3 신경망

3.1 퍼셉트론에서 신경망으로

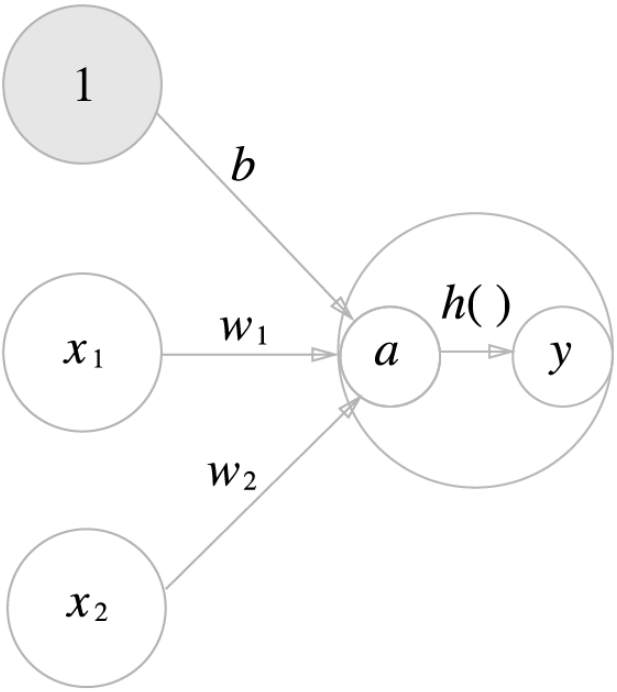




$$y = h(b + w_1 x_1 + w_2 x_2)$$

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

3.1.3 활성화 함수(activation function) 의 등장



3.2 활성화 함수

시그모이드 함수

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

```
import numpy as np
import matplotlib.pylab as plt
def sigmoid(x):
    return 1 / (1 + np.exp(-x))


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

x = np.arange(-5.0, 5.0, 0.1)
y1 = sigmoid(x)
y2 = step_function(x)

plt.plot(x, y1)
plt.plot(x, y2, 'k--')
plt.ylim(-0.1, 1.1) # y축 범위 지정
plt.show()
```

3.2.6 비선형 함수

선형이라면 h(X) = AX 형태

다층으로 h()를 여러 번 적용해도 h(h(h(X))) = AAAX 이므로

처음부터 h(X) = A^3X 한것과 같음

3.2.7 ReLU(Rectified Linear Unit)

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

```
import numpy as np
import matplotlib.pylab as plt
def relu(x):
    return np.maximum(0, x)

x = np.arange(-5.0, 5.0, 0.1)
y = relu(x)
plt.plot(x, y)
plt.ylim(-1.0, 5.5)
plt.show()
```

3.3 다차원 배열의 계산

```
import numpy as np
A=np.array([1,2,3,4,5])
print (A.shape, A.ndim)

A=np.array([[1,2,3], [4,5,6]])
```

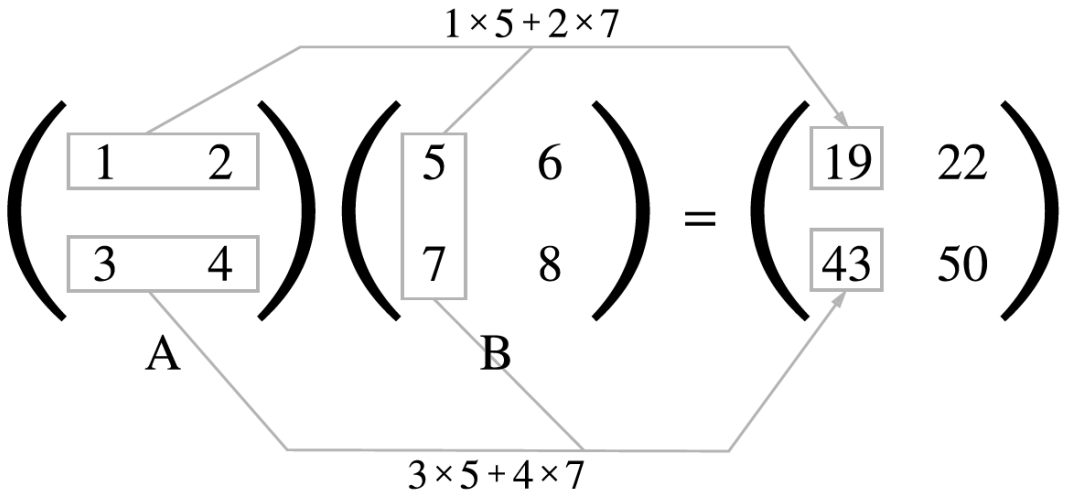
print (A.shape, A.ndim)

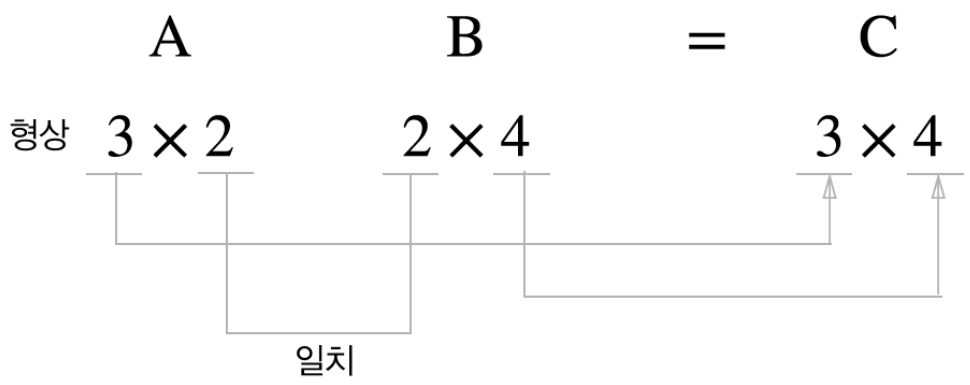
B=np.array([[1,2], [3,4], [5,6]])

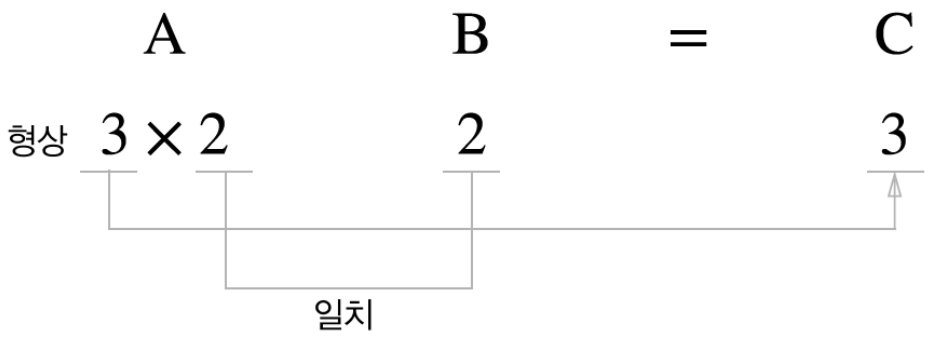
print (B.shape, B.ndim)

#A*B

print (np.dot(A,B))

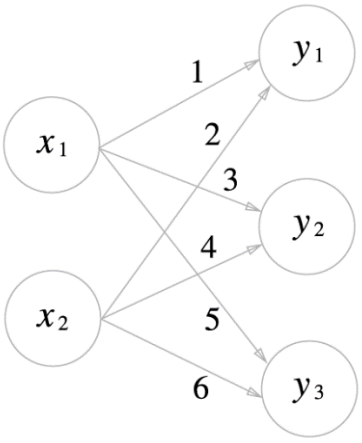







```
import numpy as np

A=np.array([ [1,2],[3,4],[5,6]])

B=np.array([ 7,8])

print ( np.dot(A,B))
```

3.3.3 신경망의 내적



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

X W = Y

2 2 × 3 3

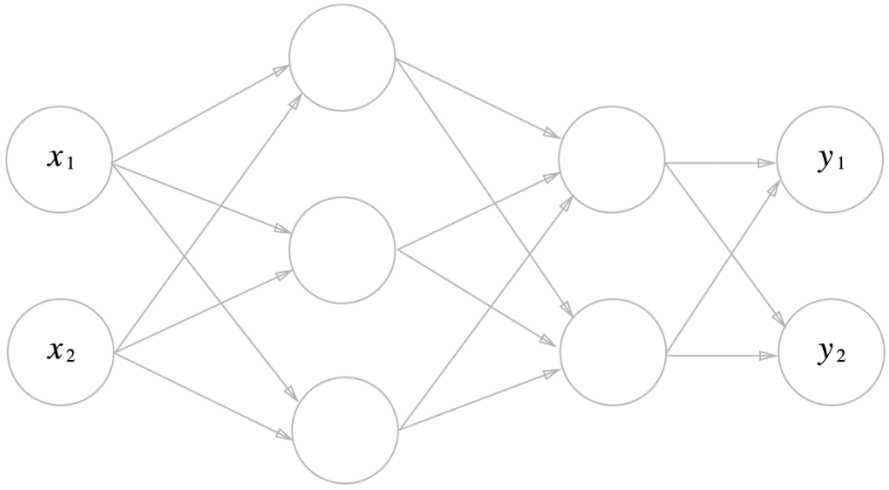
일치

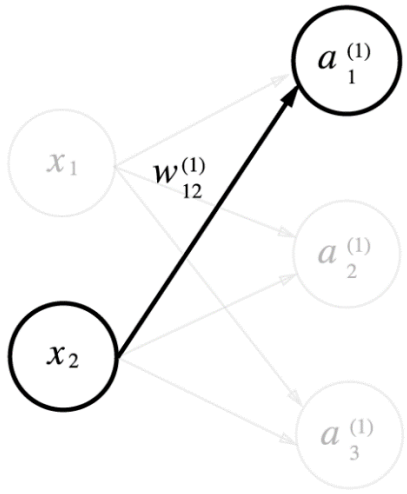
```
import numpy as np

X=np.array( [1,2] )

W=np.array([ [1,3,5], [2, 4, 6]])

print ( np.dot(X,W), np.dot(X,W).shape)
```

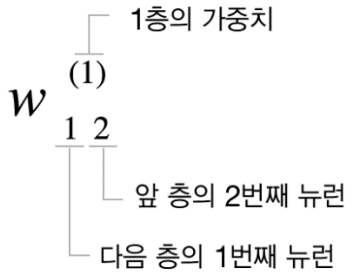
```
import numpy as np

X=np.array( [[1,2],[3,4]] )

W=np.array([ [1,3,5], [2, 4, 6]])

print ( np.dot(X,W), np.dot(X,W).shape)
```

학습한 행렬곱과 활성화함수를 사용하여 신경망 구현

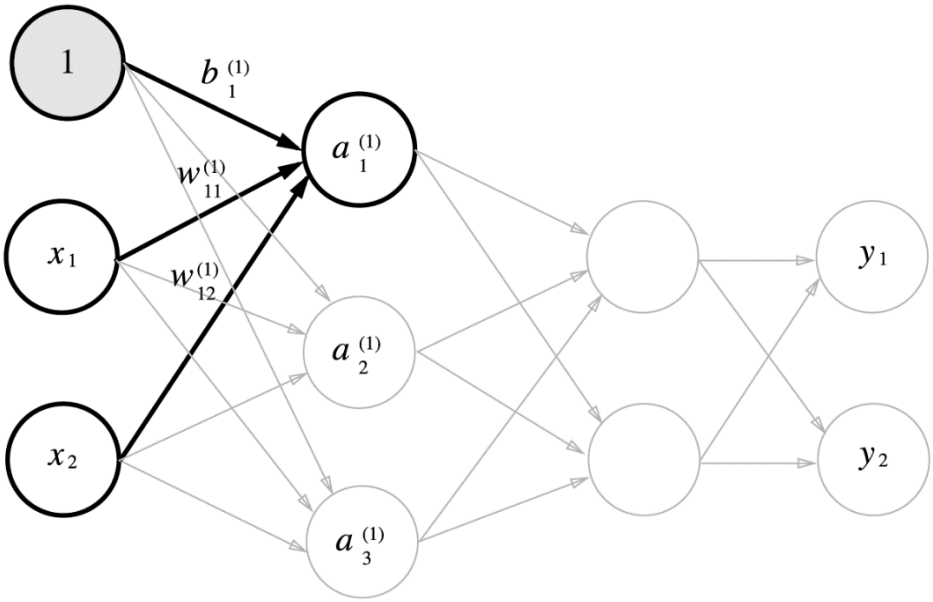
3.4 3층 신경망 구현하기







3.4.2 각 층의 신호 전달 구현하기



$$a_1^{(1)} = w_{11}^{(1)}x_1 + w_{12}^{(1)}x_2 + b_1^{(1)}$$

$$\mathbf{A}^{(1)} = \mathbf{X}\mathbf{W}^{(1)} + \mathbf{B}^{(1)}$$

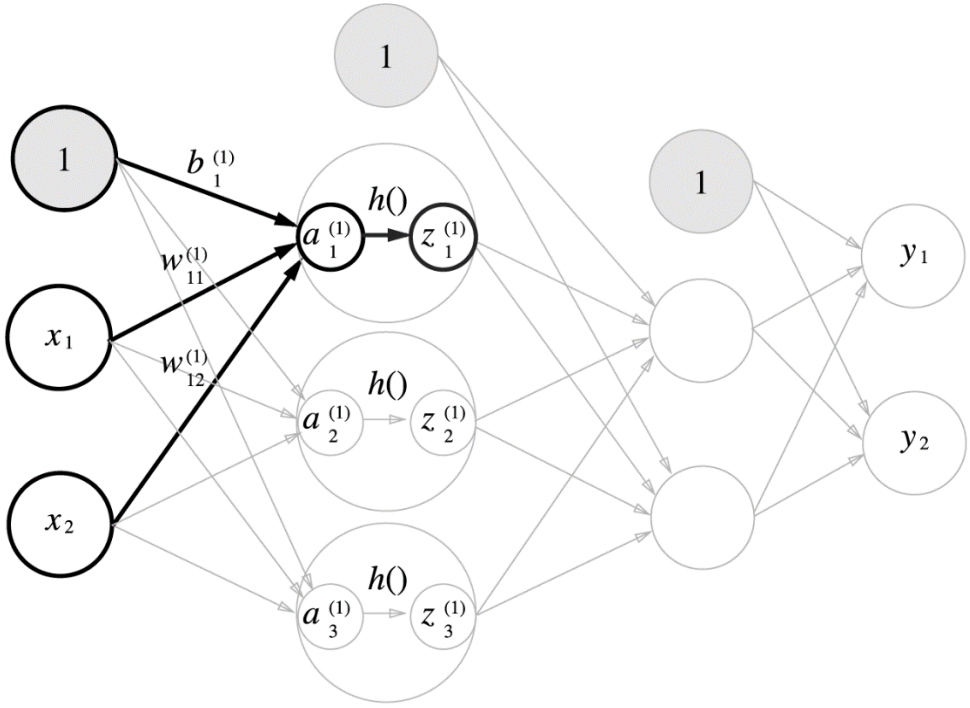
$$A^{(1)} = \left(a_1^{(1)}, a_2^{(1)}, a_3^{(1)}\right), X = (x_1, x_2), B^{(1)} = \left(b_1^{(1)}, b_2^{(1)}, b_3^{(1)}\right)$$

$$W^{(1)} = \begin{pmatrix} w_{11}^{(1)}, w_{21}^{(1)}, w_{31}^{(1)} \\ w_{12}^{(1)}, w_{22}^{(1)}, w_{32}^{(1)} \end{pmatrix}$$

```
import numpy as np

X=np.array( [1.0, 0.5] )

W1=np.array([ [0.1, 0.3, 0.5], [0.2, 0.4, 0.6]])

B1 = np.array( [0.1, 0.2, 0.3] )

print ( X.shape )

print ( W1.shape )
```

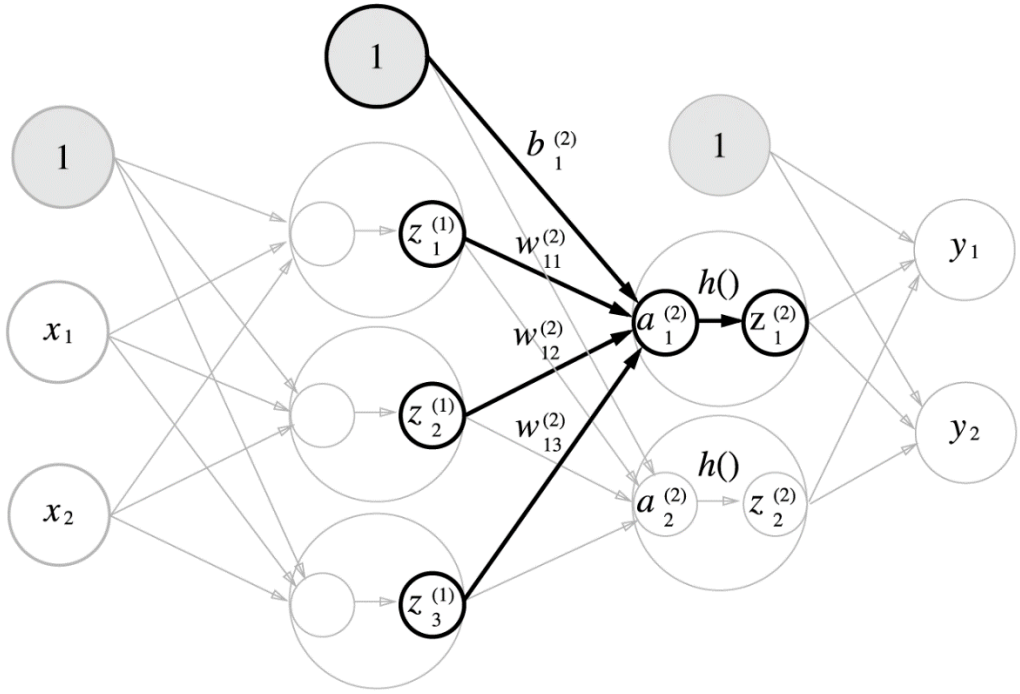
```
print ( B1.shape )
```

```
A1 = np.dot( X, W1 ) + B1
print ( A1.shape, A1 )
```




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

```
Z1 = sigmoid(A1)
print ( Z1.shape, Z1 )
```



W2=np.array([[0.1, 0.4], [0.2, 0.5], [0.3, 0.6]])

B2 = np.array([0.1, 0.2])

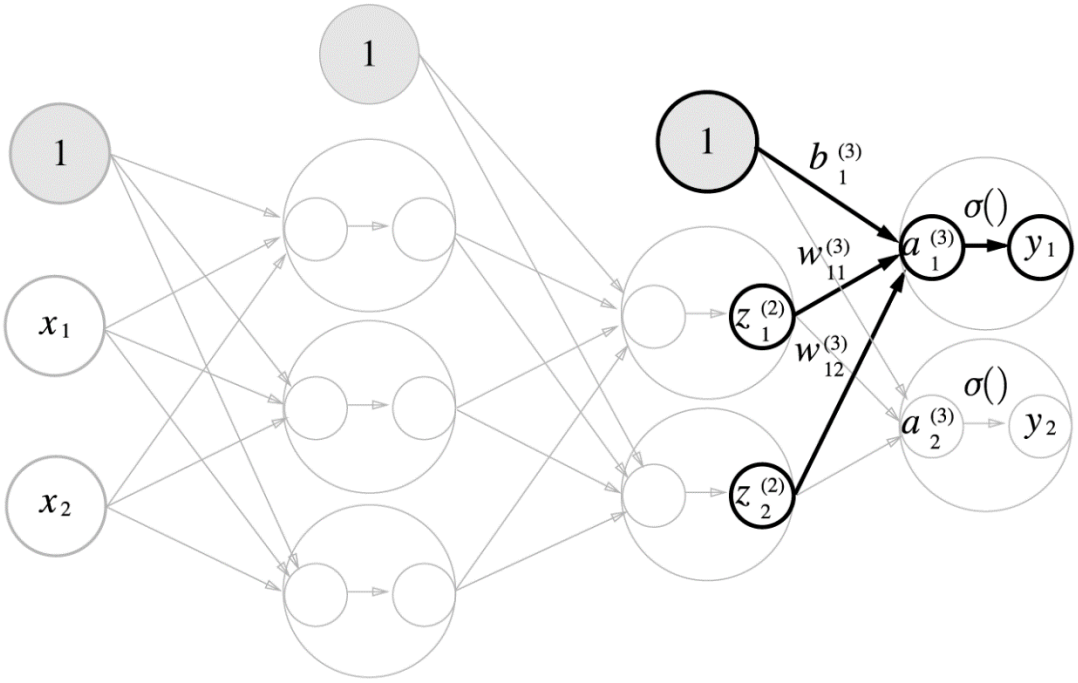
print (Z1.shape)

print (W2.shape)

print (B2.shape)

A2 = np.dot(Z1, W2) + B2

Z2 = sigmoid(A2)



W3=np.array([[0.1, 0.3], [0.2, 0.4]])

B3 = np.array([0.1, 0.2])

A3 = np.dot(Z2, W3) + B3

Y = A3

3.4.3 구현 정리

가중치 W와 편향 B들이 주어져 있고,

그림의 순서에 따라 행렬곱과 덧셈 수행, 활성화 함수 적용

3월 17일

3.5 출력층 설계하기

목적에 따라 출력층에서 사용하는 활성화 함수가 결정됨

분류 (classification) : 입력이 어떤 클래스에 속하는지 결정

예) 사진속 인물 분류

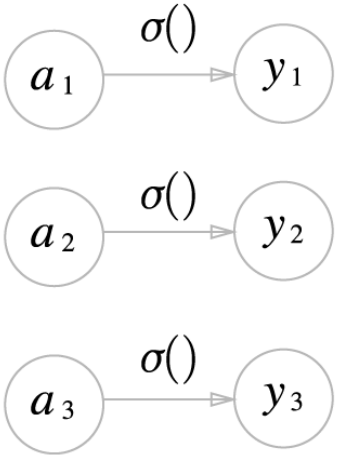
회귀 (regression) : 입력에서 연속적인 수치를 예측

예) 사진속 인물의 몸무게

분류에는 소프트맥스, 회귀에는 항등함수가 주로 사용됨

3.5.1 항등 함수와 소프트맥스 함수 구현하기

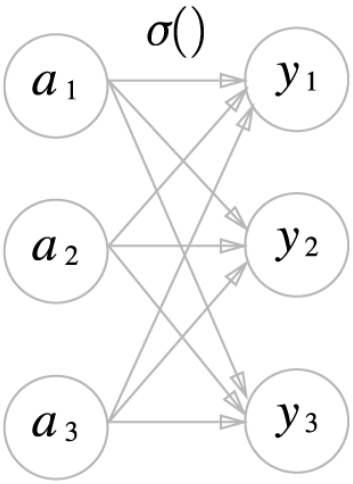
항등함수 (identity function)는 입력을 그대로 출력



소프트맥스 (softmax) 함수

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

여기서, n은 출력층의 뉴런 수, y_k는 k번째 출력




```
def softmax(a):
    y = np.exp(a) / np.sum(np.exp(a))
    return y
```

3.5.2 소프트맥스 함수 구현 시 주의점

오버플로($\exp(1000) = e^{1000}$)를 방지: exp( )가 커지지 않게 함

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

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

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

```
def softmax(a):
    c = np.max(a)
    exp_a = np.exp(a-c)
    y = exp_a / np.sum(exp_a)
    return y
```

3.5.3 소프트맥스 함수의 특징

$y_k = \frac{\exp(a_k)}{\sum\exp(a_k)}$ 를 모두 더하면 $\sum_{k=1}^{n}\frac{\exp(a_k)}{\sum\exp(a_k)} = \frac{\sum\exp(a_k)}{\sum\exp(a_k)} = 1$

$$0 \le y_k \le 1$$

$y_k = \frac{\exp(a_k)}{\sum\exp(a_k)}$를 입력을 k 클래스로 분류될 확률로 해석 가능

학습에는 y_k를 사용, 분류시에는 a_k사용해도 exp는 증가함수이므로 대소관계는 유지하며 계산량 감소

3.5.4 출력층의 뉴런 수 정하기

분류하려는 클래스 수: 숫자는 10개

3.6 손글씨 숫자 인식

파라미터 학습은 다음장, 지금은 가중치 존재 가정함

분류(예측, 신경망 순전파 forward propagation)

3.6.1 MNIST 데이터셋

훈련 이미지: 60,000, 시험 이미지: 10,000

28x28 회색조, 각 픽셀은 0에서 255까지의 값,

레이블은 '7', '2', '1'과 같은 숫자

dataset/mnist.py 에 load_mnist() 함수를 이용해 다운로드

```
import sys, os
sys.path.append(os.pardir) #부모 디렉터리의 파일 가져올 수 있도록 설정
import numpy as np
from dataset.mnist import load_mnist
from PIL import Image
```

```
def img_show(img):
    pil_img = Image.fromarray(np.uint8(img))
    pil_img.show()

(x_train, t_train), (x_test, t_test) = load_mnist(flatten=True, normalize=False)

img = x_train[0]
label = t_train[0]
print(label)  # 5

print(img.shape)  # (784,)
img = img.reshape(28, 28)  # 형상을 원래 이미지의 크기로 변형
print(img.shape)  # (28, 28)
```

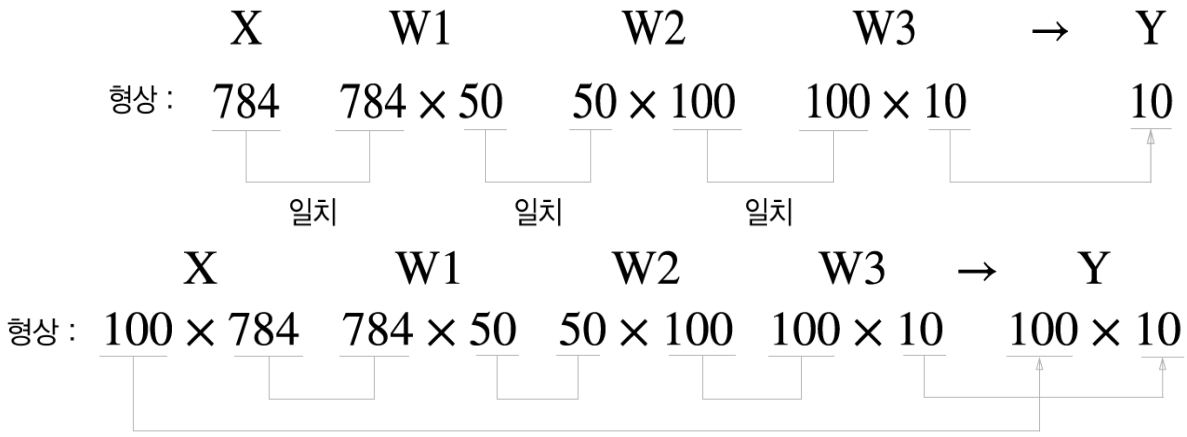
3.6.2 신경망의 추론 처리

ch03/neuralnet_mnist.py 읽기

path에 datataset, common 디렉토리가 접근 가능하도록 설정

현재 디렉토리에서 sample_weight.pkl 를 읽는다.

3.6.3 배치 처리 : 묶음 처리



ch03/ neuralnet_mnist_batch.py 읽기