

# 목차

2018년 11월 27일 화요일    오후 6:21

## - 딥러닝

### - 1장. 이 책을 이해 하기 위한 파이썬 기본 문법

- numpy 란?
- numpy에 broadcast 기능

### - 2장 퍼셉트론

- AND게이트
- OR게이트
- NAND게이트
- XOR게이트

### - 3장 신경망

- 퍼셉트론과 신경망의 차이점?
- 신경망안에 들어가는 활성화 함수 3가지
  - 계단함수
  - 시그모이드 함수
  - Relu 함수
- 다차원 배열의 계산
- 행렬의 내적
- 신경망의 내적
- 3층 신경망 구현하기
  - 소프트맥스 함수
- 신경망에 데이터 입력시 배치(batch)로 입력하는 방법

### - 4장 신경망 학습

- 평균 제곱 오차
- 교차 엔트로피 비용 함수
- 미니 배치 학습
- 미니배치 처리에 맞도록 교차 엔트로피 함수를 구성하는 방법
- 확률적 경사 하강법(SGD Stochastic Gradient Decent)
- 최소 비용의 가중치를 알아내기 위해 학습을 시키는가? (미분)
- 신경망 학습을 시키기 위한 방법

- 학습이 스스로 되는 3층 신경망 구현

## - 5장 오차역전파를 이용한 3층신경망 학습

- 계산그래프
- 활성화 함수 계층 구현하기
- 배치 정규화
- 오버피팅을 억제하는 방법 2가지
  - 1. 드롭아웃(dropout)
  - 2. 가중치 감소

## - 7장 CNN(Convolution neural network)

- 합성곱 계층
- 패딩
- 3차원의 합성곱
- 합성곱 총정리
- 블록으로 생각하기
- 풀링(pooling)
  - 풀링의 종류 3가지
- 최대 풀링을 어떻게 진행하는지 그림으로 설명

## - 신경망 함수들

## 인공지능의 눈? cnn

사진속에 사람, 동물등을 구별 할 수 있다.

다음 카카오 로드뷰에 사람 얼굴이나 차 번호등을 개인정보 보호법상 반드시 모자이크 처리를 해야하는데 사람이 일일이 할 수 없다. 컴퓨터에게 모자이크 처리하라고 시켜야 한다.

### 데이터를 분석한다는 것은?

회사에 돈을 벌어 주겠다

### 의료 영상 데이터를 분석한다는 것은?

환자를 살리는 일이 된다.

007 영상 : object detection, yolo 기술

mri 사진 : segmentation

## 인공지능의 입? rnn

### 목차

1장. 파이썬 기본 문법

2장. 퍼셉트론

3장. 신경망 (신경망의 활성화 함수, 3층 신경망 생성)

4장. 신경망 학습( 손실(오차) 함수, 수치 미분, 경사 하강법, 학습 알고리즘 구현)

5장. 오차 역전파(계산 그래프, 연쇄법칙, 역전파, 단순한 계층 구현)

6장. 신경망을 학습 시키는 여러 기술들 소개( 경사 하강법의 종류, 배치 정규화, 드롭아웃)

7장. CNN (합성곱 신경망) 사진을 신경망에 입력해서 이 사진이 어떤 사진인지 컴퓨터가 알아맞히게 하는

방법 구현

8장. 딥러닝의 역사

↓

텐서 플로우를 이용해서 신경망 구현

↓

강화학습

### 목표 : 이미지르 분류할 수 있는 신경망을 구현

1. 폐결절 사진 vs 정상 폐사진 구별

2. 제조업에서 만드는 제품들의 불량 여부 확인

ex) 스노우보드 : 정상 스노우보드 vs 기스가 있는 스노우보드

옷감(천) : 정상 옷감 vs 기스가 있는 옷감

# 1장. 이 책을 이해 하기 위한 파이썬 기본 문법

2018년 8월 14일 화요일 오전 9:48

- **numpy 란?**

- 파이썬 언어에서 기본적으로 지원하지 않는 배열(array) 혹은 행렬(matrix)의 계산을 쉽게 해주는 라이브러리
- 머신러닝에서 많이 사용하는 선형대수학에 관련된 수식들을 파이썬에서 쉽게 프로그래밍 할 수 있게 해준다.

**문제1. 아래의 행렬을 numpy로 만드시오.**

보기)

```
1 2
3 4
```

답)

```
import numpy as np
a=np.array([[1,2],[3,4]])
print(a)
```

```
[[1 2]
 [3 4]]
```

**문제2. 위의 a행렬의 각 요소에 5를 더한 값을 출력하시오.**

답)

```
import numpy as np
a=np.array([[1,2],[3,4]])
print(a+5)
```

```
[[6 7]
 [8 9]]
```

**문제3. 아래의 배열의 원소들의 평균값을 출력하시오.**

보기)

```
a=np.array( [1,2,4,5,5,7,10,13,18,21] )
```

답)

```
import numpy as np
a=np.array( [1,2,4,5,5,7,10,13,18,21] )
print(np.mean(a))
```

```
8.6
```

```
print(np.median(a)) #중앙
```

```
print(np.max(a)) #최대
```

```
print(np.min(a)) #최소
```

```
print(np.std(a)) #표준편차
```

```
print(np.var(a)) #분산
```

문제4. 아래의 행렬식을 numpy로 구현하시오.

보기)

```
1 3 7      0 0 5
1 0 0      +  7 5 0 = ?
```

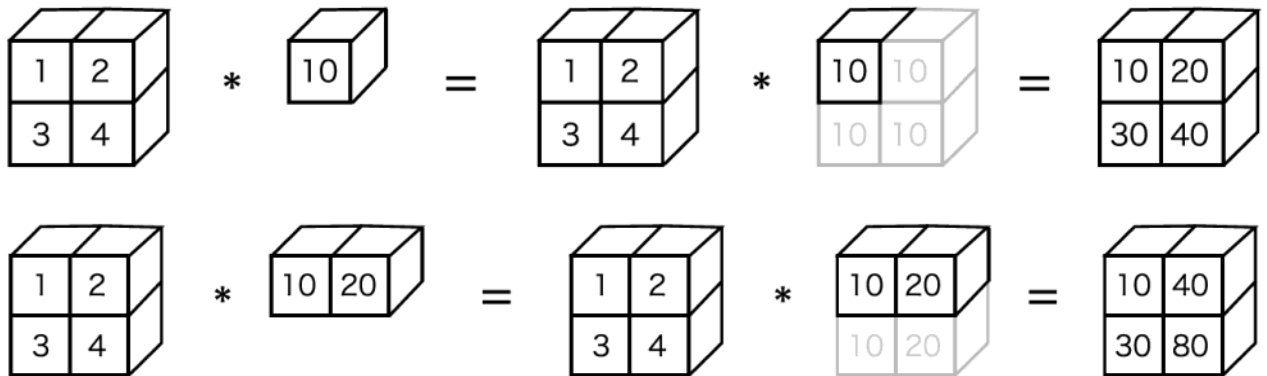
답)

```
import numpy as np
a=np.array( [[1,3,7],[1,0,0]] )
b=np.array( [[0,0,5],[7,5,0]] )
print(a+b)
```

```
[[ 1  3  7]
 [ 1  0  0]
 [ 8  5  0]]
```

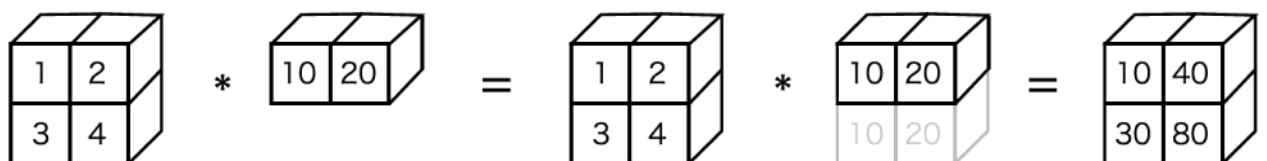
#### • numpy에 broadcast 기능

ex)



문제5. 아래 그림을 브로드 캐스트를 numpy로 구현하시오.

보기)



답)

```
import numpy as np
a=np.array( [[1,2],[3,4]] )
b=np.array( [[10, 20]] )
print(a*b)
```

```
[[10 40]
 [30 80]]
```

문제6. 아래의 행렬식을 numpy로 구현하고 아래의 요소에서 15이상인 것만 출력하시오.

보기)

```
51 55
14 19
```

0 4

결과)

[51 55 19]

답)

```
import numpy as np
a=np.array( [[51,55],[14,19],[0,4]] )
print(a[a>=15])
```

[51 55 19]

a=a.flatten() #일차원 배열로 변환

문제7. 문제6번을 numpy를 이용하지 않고 구현하시오.

답)

문제8. 아래의 행렬의 합을 numpy를 이용하지 않고 구현 해보시오.

보기)

```
1 3 7 + 0 0 5
1 0 0   7 5 0 =
```

답)

```
a=[[1,3,7],[1,0,0]]
b=[[0,0,5],[7,5,0]]
# print(a)
# print(b)
res=[[0,0,0],[0,0,0]]

for i in range(len(a)):
    for j in range(len(a[0])):
        res[i][j]=a[i][j]+b[i][j]

print(res)
```

[[1, 3, 12], [8, 5, 0]]

문제9. numpy의 브로드캐스트를 사용한 연산을 numpy를 이용하지 않는 방법으로 구현하시오.

보기)

```
1 2 * 10 20
3 4
↓ 브로드캐스트
10 40
30 80
```

답1)

```
a=[[1,2],[3,4]]
b=[[10,20]]
res=[[0,0],[0,0]] #numpy로 만드는 방법 np.zeros([2,2])
```

```

for i in range(len(a)):
    for j in range(len(b[0])):
        res[i][j]=a[i][j]*b[0][j]

print(res)

```

답2)

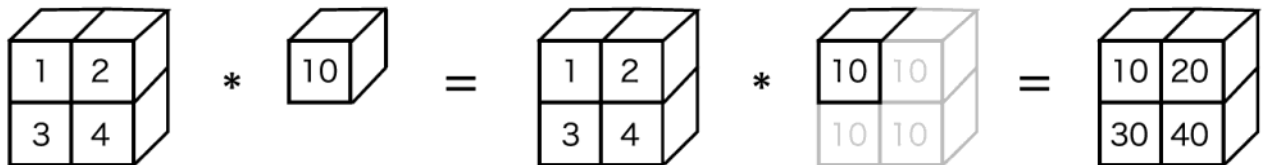
```

a=[[1,2],[3,4]]
b=[10,20]
d=[]
for i in a:
    c=[]
    for j,h in zip(i,b):
        c.append(j*h)
    d.append(c)
print(d)

```

문제10. (점심시간) 아래의 브로드 캐스트를 numpy를 이용하지 않고 파이썬으로 구현하시오.

보기)



답1)

```

a=[[1,2],[3,4]]
b=[[10]]
res=[[0,0],[0,0]] #numpy로 만드는 방법 np.zeros([2,2])

```

```

for i in range(len(a)):
    for j in range(len(res)):
        res[i][j]=a[i][j]*b[0][0]

```

```

print(res)
[[ 10, 20], [ 30, 40]]

```

답2)

## • matplotlib 사용법

"딥 러닝 실험에서는 그래프 그리기와 데이터 시각화가 중요하다."

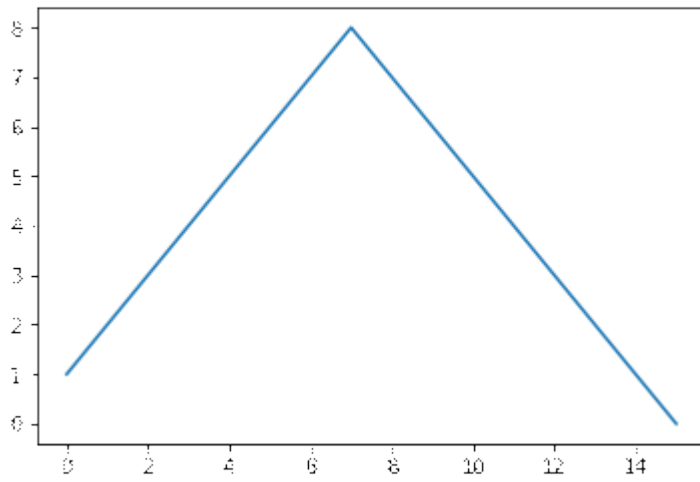
"그래프를 그리기 위한 라이브러리"

예제1)

```

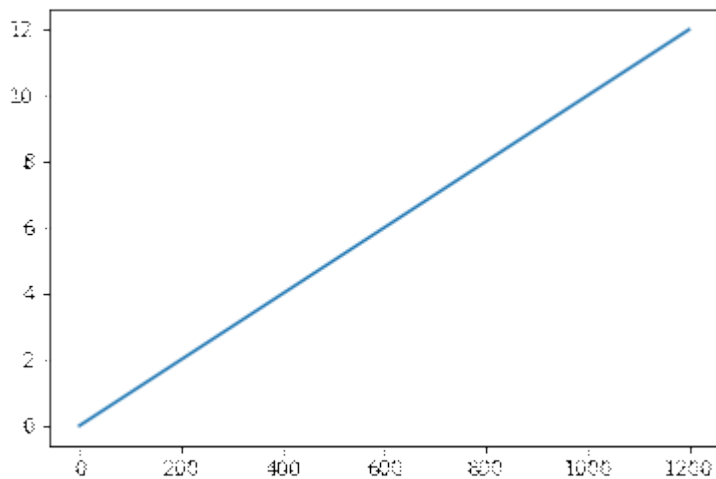
import matplotlib.pyplot as plt
#plt.figure() #여러개의 그래프를 한화면에 그릴때 필요
plt.plot([1,2,3,4,5,6,7,8,7,6,5,4,3,2,1,0])
plt.show()

```



### 예제2) 넘파이 배열을 이용해서 그리기

```
import matplotlib.pyplot as plt
import numpy as np
t=np.arange(0, 12, 0.01) #0부터 12까지 0.01씩
print(t)
plt.plot(t)
plt.show()
```



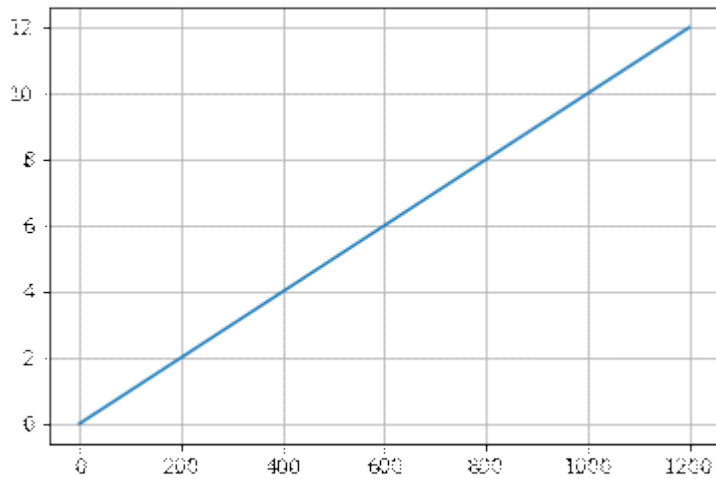
문제11. 위의 그래프에 **grid(격자)**를 추가하시오

답)

```
import matplotlib.pyplot as plt
import numpy as np
t=np.arange(0, 12, 0.01) #0부터 12까지 0.01씩

plt.plot(t)
plt.grid()
plt.show()
```





문제12. 현수형이 광록이형 에대한 마음을 시각화하시오.

답)

```
import matplotlib.pyplot as plt
```

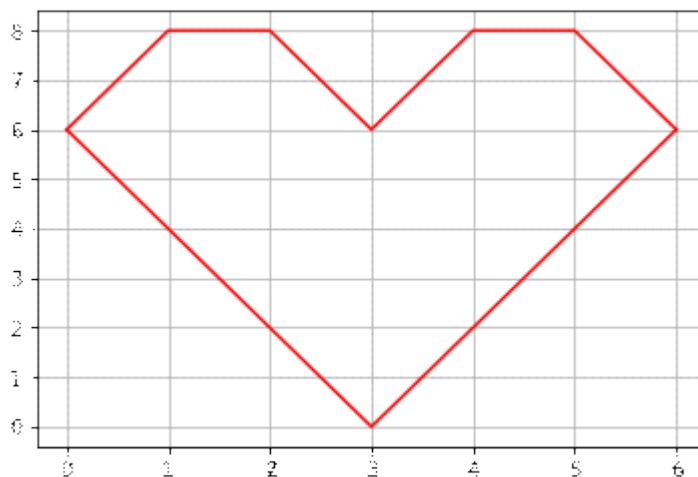
```
plt.figure()
```

```
plt.plot([6, 4, 2, 0, 2, 4, 6], color = 'red')
```

```
plt.plot([6, 8, 8, 6, 8, 8, 6], color = 'red')
```

```
plt.grid()
```

```
plt.show()
```



문제13. 위의 그래프에 제목을 붙이시오.

보기)

제목 : 현수형 -> 광록이형

답)

```
import matplotlib.pyplot as plt
```

```
from matplotlib import font_manager, rc
```

```
font_name =
```

```
font_manager.FontProperties(fname="C:/Windows/Fonts/H2PORM.TTF").get_name()
```

```
rc('font', family=font_name)
```

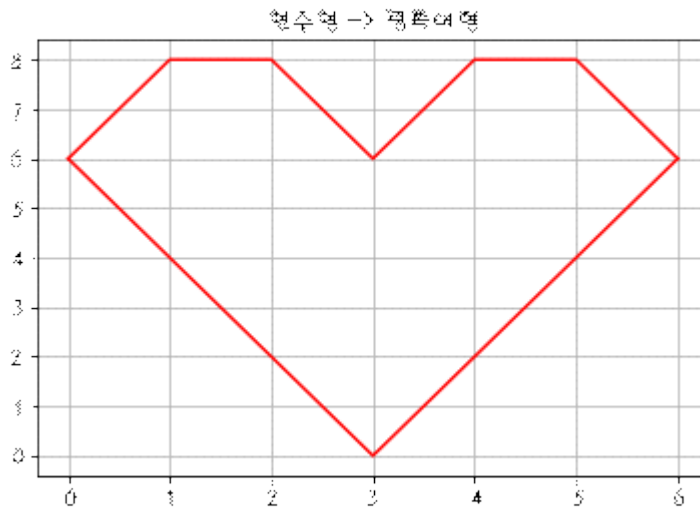
```
plt.plot([6, 4, 2, 0, 2, 4, 6], color = 'red')
```

```
plt.plot([6, 8, 8, 6, 8, 8, 6], color = 'red')
```

```
plt.grid()
```

```
plt.title("현수형 -> 광록이형")
```

plt.show()



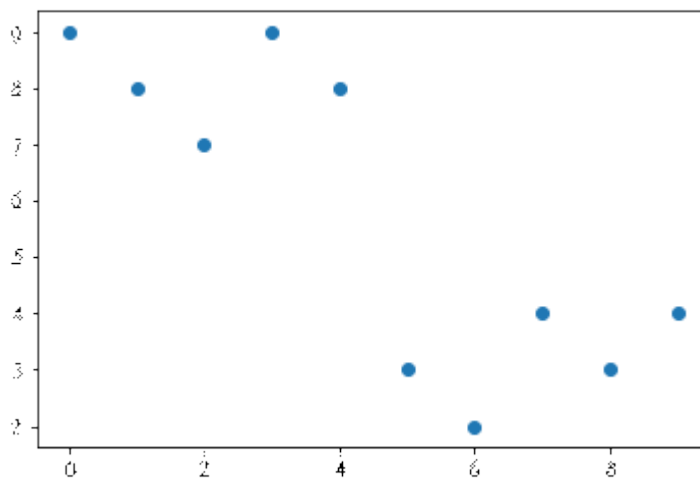
문제14. 아래의 numpy 배열로 산포도 그래프를 그리시오.

답)

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import font_manager, rc
```

```
x=np.array([0,1,2,3,4,5,6,7,8,9])
y=np.array([9,8,7,9,8,3,2,4,3,4])
```

```
plt.scatter(x,y)
plt.show()
```



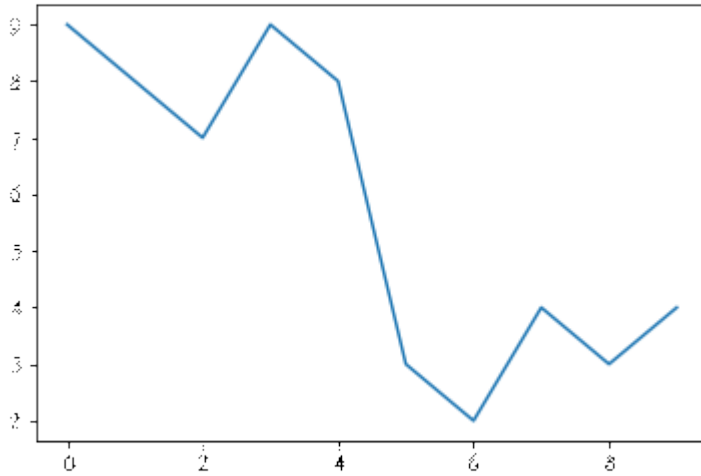
문제15. 문제14번 그래프를 라인그래프로 그리시오.

답)

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import font_manager, rc
```

```
x=np.array([0,1,2,3,4,5,6,7,8,9])
y=np.array([9,8,7,9,8,3,2,4,3,4])
```

```
plt.plot(x,y)
plt.show()
```



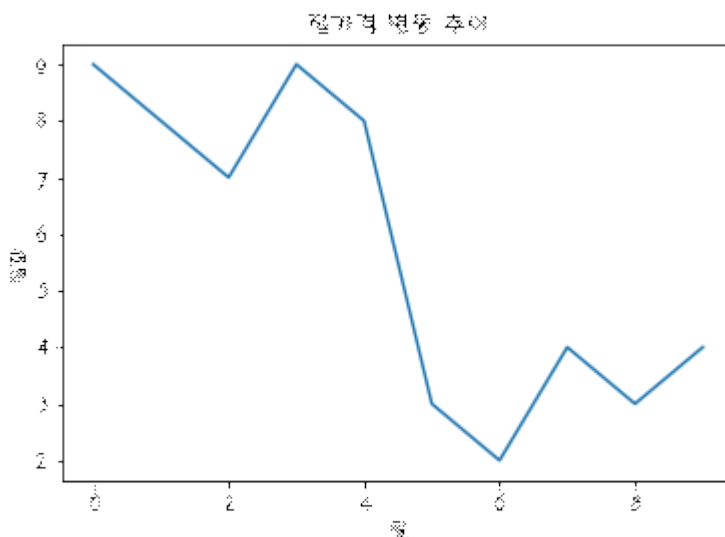
문제16. 위의 그래프에 x축을 월 이라고하고, y축을 집값으로 라벨을 붙이시오.

답)

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import font_manager, rc
```

```
x=np.array([0,1,2,3,4,5,6,7,8,9])
y=np.array([9,8,7,9,8,3,2,4,3,4])
```

```
plt.plot(x,y)
plt.xlabel('월')
plt.ylabel('집값')
plt.title("집가격 변동 추이")
plt.show()
```



문제17. 치킨집 년도별 창업건수를 가지고 라인 그래프를 그리시오.

답)

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

```
chi = np.loadtxt('C:\\python_data\\창업건수.csv', skiprows=1, unpack=True, delimiter=',')
```

#설명 : unpack=False면 csv 파일을 그대로 읽어오는 것

unpack=True면 csv파일을 pivot해서 읽어 온다.

```
#chi = np.loadtxt('C:\\python_data\\창업건수.csv', skiprows=1, unpack=True, delimiter=',')
#print(chi, '\n')
#chi2 = np.loadtxt('C:\\python_data\\창업건수.csv', skiprows=1, unpack=False, delimiter=',')
#print(chi2)
```

```
[[2005., 2006., 2007., 2008., 2009., 2010., 2011., 2012., 2013., 2014.]
 [2196., 2028., 1802., 1691., 1826., 1798., 1688., 1767., 1965., 1980.]
 [1034., 950., 1036., 1127., 1086., 1105., 1199., 1183., 1432., 1870.]
 [ 540., 577., 620., 561., 645., 669., 736., 753., 839., 1095.]
 [ 530., 525., 507., 543., 711., 865., 837., 986., 954., 1193.]
 [ 454., 483., 575., 772., 845., 1291., 1671., 1847., 2287., 3053.]
 [5994., 5504., 6148., 6036., 6577., 6689., 6900., 7082., 7708., 9772.]
 [ 635., 591., 544., 525., 627., 553., 638., 687., 769., 1272.]]

[[2005., 2196., 1034., 540., 530., 454., 5994., 635.]
 [2006., 2028., 950., 577., 525., 483., 5504., 591.]
 [2007., 1802., 1036., 620., 507., 575., 6148., 544.]
 [2008., 1691., 1127., 561., 543., 772., 6036., 525.]
 [2009., 1826., 1086., 645., 711., 845., 6577., 627.]
 [2010., 1798., 1105., 669., 865., 1291., 6689., 553.]
 [2011., 1688., 1199., 736., 837., 1671., 6900., 638.]
 [2012., 1767., 1183., 753., 986., 1847., 7082., 687.]
 [2013., 1965., 1432., 839., 954., 2287., 7708., 769.]
 [2014., 1980., 1870., 1095., 1193., 3053., 9772., 1272.]]
```

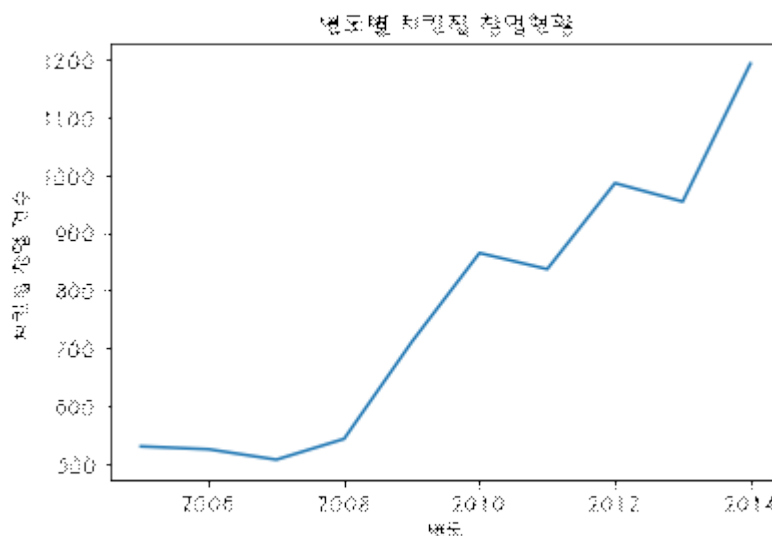
T

chi

F

chi

```
plt.plot(x,y)
plt.title("년도별 치킨집 창업현황")
plt.xlabel('년도')
plt.ylabel('치킨집 창업 건수')
plt.show()
```



문제18. 치킨집 폐업현황도 위의 그래프에 겹치게해서 출력하시오.

힌트)  
plt.figure(figsize=(6,4))

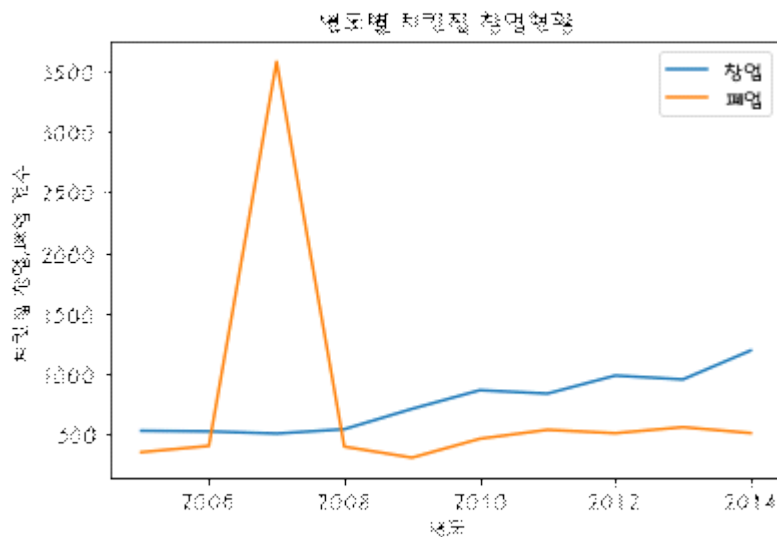
답)

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

```
chi = np.loadtxt('C:\\python_data\\창업건수.csv', skiprows=1, unpack=True, delimiter=',')
chi2 = np.loadtxt('C:\\python_data\\폐업건수.csv', skiprows=1, unpack=True, delimiter=',')
```

```
x1=chi[0]
y1=chi[4]
x2=chi2[0]
y2=chi2[4]
```

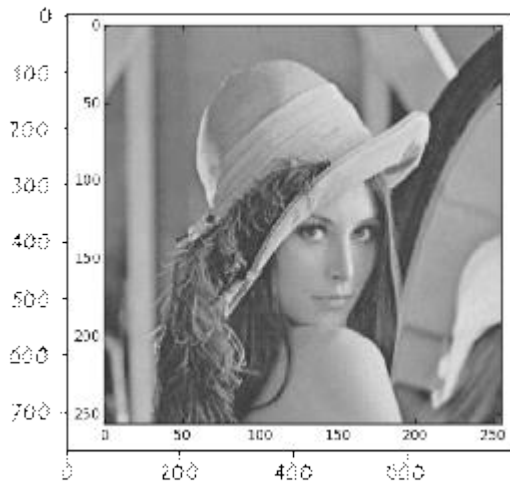
```
plt.figure(figsize=(6,4)) #그래프 크기 6x4
plt.plot(x1,y1,label='창업')
plt.plot(x2,y2, label='폐업')
plt.title("년도별 치킨집 창업현황")
plt.xlabel('년도')
plt.ylabel('치킨집 창업,폐업 건수')
plt.legend()
plt.show()
```



문제19. 이미지 표시를 파이썬으로 구현하시오.

답)

```
import matplotlib.pyplot as plt
from matplotlib.image import imread
img = imread('C:\\python_data\\aa.png')
plt.imshow(img)
plt.show
```



문제20. 고양이 사진을 파이썬에서 출력하시오.

답)

```
import matplotlib.pyplot as plt
from matplotlib.image import imread
img = imread('C:\\python_data\\cat.jpg')
plt.imshow(img)
plt.show
```

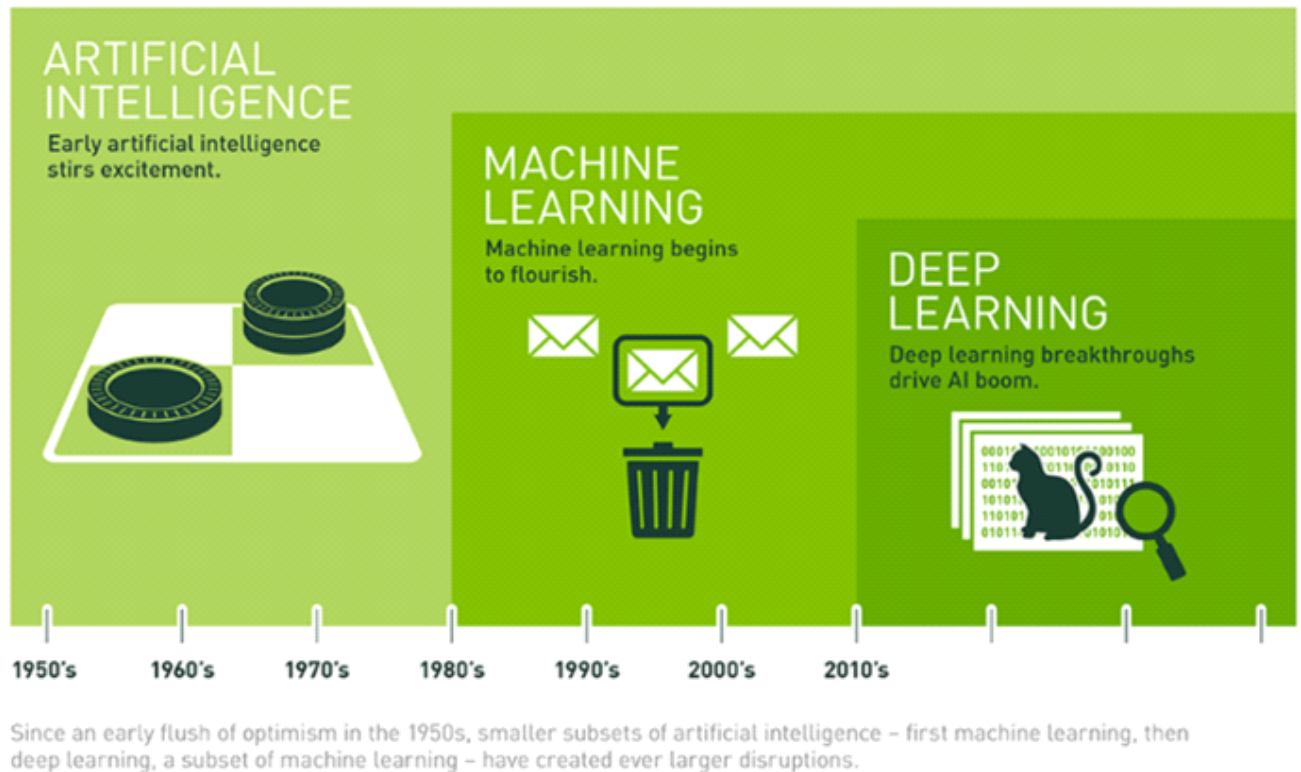


## 2장 퍼셉트론

2018년 8월 14일 화요일    오후 3:16

### 머신러닝 종류 3가지

1. 지도 학습 : 정답이 있는 상태에서 학습
2. 비지도 학습 : 정답이 없는 상태에서 학습
3. 강화 학습 : 보상을 통해서 빅데이터를 만들며 학습



### 머신러닝의 한 부류인 딥러닝( 이미지를 기계가 직접 인식해서 학습)



퍼셉트론 → 신경망

### 퍼셉트론 :인간의 뇌세포중에 하나를 컴퓨터로 구현해 봄

1943년에 미국의 신경외과 의사인 워렌 맥컬록에 의해서 발단이 되었음.

1957년에 프랑크 로젠 블라트가 퍼셉트론 알고리즘을 고안.

인간의 신경세포 하나를 흉내

1. 자극(stimulus)
2. 반응(response)
3. 역치(threshold)

"특정 자극이 있다면 그 자극이 어느 역치 이상이어야지만 세포가 반응한다."

## \* 뉴런의 개수

1. 사람 : 850 억개
2. 고양이 : 10억개
3. 쥐 : 7천 5백만개
4. 바퀴벌레: 몇백만개
5. 하루살이 : 지금 현재까지 나온 최첨단 인공지능의 뉴런수보다 많다.

### 퍼셉트론 역사

1957년에 프랑크 로젠 블라트가 퍼셉트론 알고리즘을 고안을 했다.

1969년 : 퍼셉트론은 단순 선형분류기에 불과하다.

왜냐면 ? xor 분류도 못한다고 단정을 지음

침체기에 빠짐 (인공진의 기울기)

1970년 중반 : 중요한 논문이 발표 : 역전파 알고리즘(다층 퍼셉트론)

1986년 : 은닉층을 갖는 다층 퍼셉트론 + 오류 역전파 학습 알고리즘 구현

오늘날

### 신경망을 통해서 구현하고자 하는 목표? 분류

- 붓꽃사진을 보고 붓꽃의 종을 분류 한다.

	A	B	C	D	E
	5.1	3.5	1.4	0.2	Iris-setosa
	4.9	3	1.4	0.2	Iris-setosa
	4.7	3.2	1.3	0.2	Iris-setosa
	4.6	3.1	1.5	0.2	Iris-setosa
	5	3.6	1.4	0.2	Iris-setosa
	5.4	3.9	1.7	0.4	Iris-setosa
	4.6	3.4	1.4	0.3	Iris-setosa
	5	3.4	1.5	0.2	Iris-setosa
	4.4	2.9	1.4	0.2	Iris-setosa
	4.9	3.1	1.5	0.1	Iris-setosa

원래는 위의 엑셀 파일처럼 일일이 붓꽃의 길이, 꽃받침의 길이 등등을 적어서 학습시켰지만 사실 사진을 톡 주고 학습하라고 하는게 훨~~~얼씬 편하다.





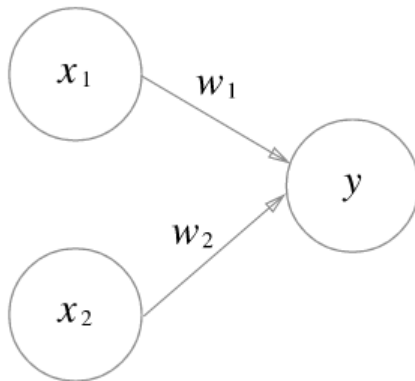
**Iris Versicolor**



**Iris Setosa**



**Iris Virginica**



입력신호의 연결강도가 가중치인데 가중치의 값이 클수록 강한 신호이다.

입력신호가 뉴런에 보내질때는 각각의 고유한 가중치가 곱해진다.

$(w_1 \cdot x_1 + w_2 \cdot x_2 \leq \theta) \rightarrow 0$  (신호가 안흐른다)

$(w_1 \cdot x_1 + w_2 \cdot x_2 > \theta) \rightarrow 1$  (신호가 흐른다)

뉴런에서 보내온 신호의 총합이 정해진 임계값을 넘어설때만 1을 출력

퍼셉트론은 n개의 이진수가 하나의 뉴런을 통과해서 가중의 합이 0보다 크면 활성화 되는 가장

간단한 신경망이다.

퍼셉트론을 학습 시키는 방법은 간단한데, 보통 목표치를 정해주고 현재 계산한 값이 목표치와 다르

면 그 만큼의 오차를 다시 퍼셉트론에 반영해서 오차를 줄여나가는 방법이다.

and 게이트

	x1	x2	t
입력1	0	0	0
입력2	0	1	0
입력3	1	0	0
입력4	1	1	1

$$w_0=0.3$$

$$w_1=0.4$$

$$w_2=0.1$$

$$x_0=-1$$

$$w_i = w_i + r * x_i * (t - f(k)) \quad \#r: \text{러닝레이트}$$

### 입력 1일때

$$K = -1 \times 0.3 + 0 \times 0.4 + 0 \times 0.1 = -0.3$$

$$f(-0.3) = 0$$

$$w_i = w_i + 0.05 \times (0 - 0) \times x_i$$

=>  $t - f(k) = 0$  에 변화가 없다

### 입력 2일때

$$K = -1 \times 0.3 + 0 \times 0.4 + 1 \times 0.1 = -0.2$$

$$f(-0.2) = 0$$

$$w_i = w_i + 0.05 \times (0 - 0) \times x_i$$

=> 변화 없음

### 입력 3일때

$$K = -1 \times 0.3 + 1 \times 0.4 + 0 \times 0.1 = 0.1$$

$$f(0.1) = 1$$

$$w_i = w_i + 0.05 \times (0 - 1) \times x_i$$

$$w_i = w_i - 0.05 * x_i$$

=> 변화가 있다

### i=0일때

$$w_0 = w_0 - 0.05 \times x_0$$

$$w_0 = 0.3 - 0.05 \times -1 = 0.35$$

### i=1일때

$$w_1 = w_1 - 0.05 \times x_1$$

$$w_1 = 0.4 - 0.05 \times 1 = 0.35$$

### i=2일때

$$w_2 = w_2 - 0.05 \times x_2$$

$$w_2 = 0.1 - 0.05 \times 0 = 0.1$$

### 입력 4일때

$$K = -1 \times 0.35 + 1 \times 0.35 + 1 \times 0.1 = 0.1$$

$$f(0.1) = 1$$

$$t - f(0.1) = 1 - 1 = 0 \text{ 이라 변화 없음}$$

쭉쭉쭉하면

최종적으로

$$w_0=0.4$$

$$w_1=0.35$$

$$w_2=0.05$$

$$x = -1$$

문제21. 아래의 4개의 딕셔너리를 이용해서 입력값과 가중치의 곱을 합을 구하는 함수를 x\_w\_sum이라는

이름으로 생성하시오.

보기)

```
input1_dic = {'x0':-1, 'x1':0, 'x2':0, 't':0}
input2_dic = {'x0':-1, 'x1':0, 'x2':1, 't':0}
input3_dic = {'x0':-1, 'x1':1, 'x2':0, 't':0}
input4_dic = {'x0':-1, 'x1':1, 'x2':1, 't':1}
w_dic={'w0':0.3, 'w1':0.4, 'w2':0.1}
```

```
print(x_w_sum(input_dic, w_dic)
```

결과)

```
-0.3
-0.2
.
.
```

답)

```
input1_dic = {'x0':-1, 'x1':0, 'x2':0, 't':0}
input2_dic = {'x0':-1, 'x1':0, 'x2':1, 't':0}
input3_dic = {'x0':-1, 'x1':1, 'x2':0, 't':0}
input4_dic = {'x0':-1, 'x1':1, 'x2':1, 't':1}
w_dic={'w0':0.3, 'w1':0.4, 'w2':0.1}
```

```
def x_w_sum(input_dic,w_dic):
    k=input_dic['x0']*w_dic['w0'] + input_dic['x1']*w_dic['w1'] + input_dic['x2']*w_dic['w2']
    return k
```

```
for i in range(1,5):
    input_dic = eval('input' + str(i) + '_dic')
    print(input_dic)
    print ( x_w_sum(input_dic, w_dic) )
```

```
{ 'x0': -1, 'x1': 0, 'x2': 0, 't': 0}
-0.3
{ 'x0': -1, 'x1': 0, 'x2': 1, 't': 0}
-0.19999999999999998
{ 'x0': -1, 'x1': 1, 'x2': 0, 't': 0}
0.10000000000000003
{ 'x0': -1, 'x1': 1, 'x2': 1, 't': 1}
0.20000000000000004
```

문제22. 위의 결과가 0보다 크거나 같으면 1을 출력 0보다 작으면 0을 출력하는 함수를 active\_func()으로

생성하시오.

답)

```
input1_dic = {'x0':-1, 'x1':0, 'x2':0, 't':0}
input2_dic = {'x0':-1, 'x1':0, 'x2':1, 't':0}
input3_dic = {'x0':-1, 'x1':1, 'x2':0, 't':0}
input4_dic = {'x0':-1, 'x1':1, 'x2':1, 't':1}
```

```

w_dic={'w0':0.3, 'w1':0.4, 'w2':0.1}

def active_func(input_dic,w_dic):
    k=input_dic['x0']*w_dic['w0'] + input_dic['x1']*w_dic['w1'] + input_dic['x2']*w_dic['w2']
    if k>=0:
        return 1
    else:
        return 0

for i in range(1,5):
    input_dic = eval('input' + str(i) + '_dic')
    print(input_dic)
    print ( active_func(input_dic, w_dic) )

{'x0': -1, 'x1': 0, 'x2': 0, 't': 0}
0
{'x0': -1, 'x1': 0, 'x2': 1, 't': 0}
0
{'x0': -1, 'x1': 1, 'x2': 0, 't': 0}
1
{'x0': -1, 'x1': 1, 'x2': 1, 't': 1}
1

```

**문제23.** 위의 코드를 이용해서 오차가 없는 가중치를 구하는 코드를 작성하시오.

힌트)

```

for j in range(3):
    w_dic['w%s' % j] = w_dic['w%s' % j] + \
        0.05 * (input_dic['t'] - active_func(input_dic, w_dic)) * input_dic['x%s' % j]
    print( w_dic )

```

답)

```

def active_func(x, y):
    k = x['x0'] * y['w0'] + x['x1'] * y['w1'] + x['x2'] * y['w2']

    if k >= 0:
        k = 1
    else:
        k = 0
    return k

input1_dic = {'x0': -1, 'x1': 0, 'x2': 0, 't': 0}
input2_dic = {'x0': -1, 'x1': 0, 'x2': 1, 't': 0}
input3_dic = {'x0': -1, 'x1': 1, 'x2': 0, 't': 0}
input4_dic = {'x0': -1, 'x1': 1, 'x2': 1, 't': 1}
w_dic = {'w0': 0.3, 'w1': 0.4, 'w2': 0.1}
res = 0

for i in range(1, 5):
    input_dic = eval('input%s_dic' % i)
    for j in range(3):
        w_dic['w%s' % j] = w_dic['w%s' % j] + \
            0.05 * (input_dic['t'] - active_func(input_dic, w_dic)) * input_dic['x%s' % j]
    print( w_dic )

```

```
{'w0': 0.3, 'w1': 0.4, 'w2': 0.1}
{'w0': 0.3, 'w1': 0.4, 'w2': 0.1}
{'w0': 0.3, 'w1': 0.4, 'w2': 0.1}
{'w0': 0.3, 'w1': 0.4, 'w2': 0.1}
{'w0': 0.3, 'w1': 0.4, 'w2': 0.1}
{'w0': 0.3, 'w1': 0.4, 'w2': 0.1}
{'w0': 0.35, 'w1': 0.4, 'w2': 0.1}
{'w0': 0.35, 'w1': 0.35000000000000003, 'w2': 0.1}
{'w0': 0.35, 'w1': 0.35000000000000003, 'w2': 0.1}
{'w0': 0.35, 'w1': 0.35000000000000003, 'w2': 0.1}
{'w0': 0.35, 'w1': 0.35000000000000003, 'w2': 0.1}
{'w0': 0.35, 'w1': 0.35000000000000003, 'w2': 0.1}
```

**문제24. 위의 코드를 18번 반복 수행하시오.**

답)

```
def active_func(x, y):
    k = x['x0'] * y['w0'] + x['x1'] * y['w1'] + x['x2'] * y['w2']

    if k >= 0:
        k = 1
    else:
        k = 0
    return k

input1_dic = {'x0': -1, 'x1': 0, 'x2': 0, 't': 0}
input2_dic = {'x0': -1, 'x1': 0, 'x2': 1, 't': 0}
input3_dic = {'x0': -1, 'x1': 1, 'x2': 0, 't': 0}
input4_dic = {'x0': -1, 'x1': 1, 'x2': 1, 't': 1}
w_dic = {'w0': 0.3, 'w1': 0.4, 'w2': 0.1}
res = 0

for k in range(18):
    for i in range(1, 5):
        input_dic = eval('input%s_dic' % i)
        for j in range(3):
            w_dic['w%s' % j] = w_dic['w%s' % j] + \
                0.05 * (input_dic['t'] - active_func(input_dic, w_dic)) * input_dic['x%s' % j]
        print( w_dic )
```

**문제25 (점심시간) 일단 위의 코드를 답글로 올리고 라인 그래프를 그리는데 t-k의 차이 즉 오차가 있다가**

**0이되는 부분이 어디인지 알아내는 라인 그래프를 그리시오.**

**(y축이 오차, x축 루프 횟수)**

답)

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import font_manager, rc

x = np.array([[ -1,0,0],[ -1,0,1],[ -1,1,0],[ -1,1,1]])
y = np.array([[0],[0],[0],[1]])

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

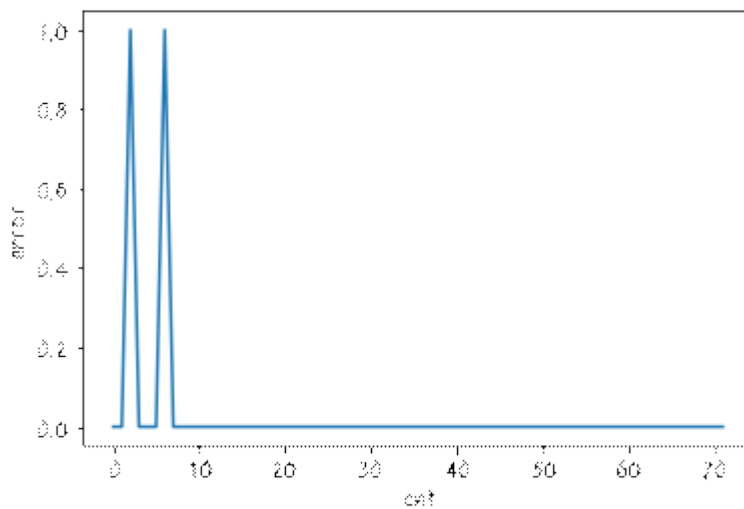
```

def pcntrain(x,y,r,repeat):
    w = np.array([[0.3], [0.4], [0.1]])
    cnt=0
    error=[]
    while True:
        for j in range(4): #입력
            k = w[0]*x[j][0] + w[1]*x[j][1] + w[2]*x[j][2]
            error.append(abs(y[j]-step_function(k)))
            if y[j]-step_function(k) != 0:
                for h in range(3):
                    w[h]=w[h]+r*x[j][h]*(y[j]-step_function(k))
        cnt+=1
        if cnt==repeat:
            break
    print(w)
    plt.plot(error)
    plt.xlabel('cnt')
    plt.ylabel('error')

    plt.show()

```

pcntrain(x,y,0.05,18



)

## • 퍼셉트론 게이트 4가지

### 1. AND 게이트

x1	x2	y
0	0	0
1	0	0
0	1	0
1	1	1

### 2. OR 게이트

--	--	--

x1	x2	y
0	0	0
1	0	1
0	1	1
1	1	1

### 3. NAND 게이트

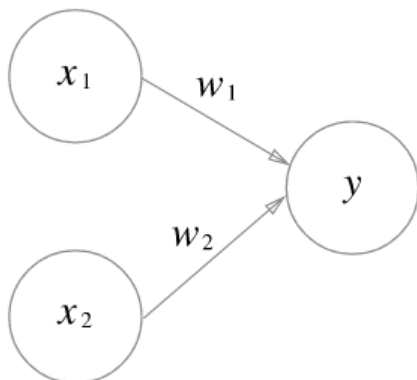
x1	x2	y
0	0	1
1	0	1
0	1	1
1	1	0

### 4. XOR 게이트

x1	x2	y
0	0	0
1	0	1
0	1	1
1	1	0

문제26. 아래의 식을 파이썬으로 구현하시오. (numpy를 이용해서)

보기)



$$x1 * w1 + x2 * w2 = ?$$

x1 : 0

x2 : 1

w1 : 0.5

w2 : 0.5

답)

```
import numpy as np
```

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

0.5

문제27. 위의 식에 책52쪽에 나오는 편향을 더해서 완성한 아래의 식을 파이썬으로 구현하시오.

보기)

$b+x_1*w_1 + x_2*w_2 = ?$

b(편향) : -0.7

x1 : 0

x2 : 1

w1 : 0.5

w2 : 0.5

답)

```
import numpy as np
```

```
x=np.array([0,1])
w=np.array([0.5,0.5])
b=-0.7
print(np.sum(x*w)+b)
-0.19999999999999996
```

## AND게이트

문제28. and게이트를 파이썬으로 구현하시오.

답)

```
import numpy as np
```

```
def AND(x1,x2):
    x=np.array([x1,x2])
    w=np.array([0.5,0.5])
    theta=0.7
    y=np.sum(x*w)
    if y>theta:
        return 1
    elif y<=theta:
        return 0
```

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



```
0
0
0
1
```

문제29. 위에서 AND퍼셉트론 함수를 이용해서 아래의 inputdata를 이용해서 출력 결과를 for loop문으로

한번에 출력하시오.

보기)

```
inputData=np.array([[0,0],[0,1],[1,0],[1,1]])
```

답)

```
import numpy as np
```

```
def AND(x1,x2):
```

```
    x=np.array([x1,x2])
```

```
    w=np.array([0.5,0.5])
```

```
    theta=0.7
```

```
    y=np.sum(x*w)
```

```
    if y>theta:
```

```
        return 1
```

```
    elif y<=theta:
```

```
        return 0
```

```
inputData=np.array([[0,0],[0,1],[1,0],[1,1]])
```

```
print('----AND-----퍼셉트론')
```

```
for i in inputData:
```

```
    print(str(i), ">=", str(AND(i[0],i[1])) ) )
```

---

```
----AND-----퍼셉트론
```

```
[0 0] => 0
```

```
[0 1] => 0
```

```
[1 0] => 0
```

```
[1 1] => 1
```

## OR게이트

문제30. 문제29번 코드를 수정해서 OR게이트 함수를 만드시오.

답)

```
import numpy as np
```

```
def OR(x1,x2):
```

```
    x=np.array([x1,x2])
```

```
    w=np.array([0.5,0.5])
```

```
    theta=0.4
```

```
    y=np.sum(x*w)
```

```
    if y>theta:
```

```
        return 1
```

```
    elif y<=theta:
```

```
        return 0
```

```
inputData=np.array([[0,0],[0,1],[1,0],[1,1]])
```

```

print('----OR----퍼셉트론')
for i in inputData:
    print(str(i), ">=", str(OR(i[0],i[1])) ) )

----OR----퍼셉트론
[0 0] => 0
[0 1] => 1
[1 0] => 1
[1 1] => 1

```

## NAND게이트

**문제31. NAND 퍼셉트론 함수를 만드시오.**

```

답)
import numpy as np

def NAND(x1,x2):
    x=np.array([x1,x2])
    w=np.array([-0.5,-0.5])
    theta= -0.7
    y=np.sum(x*w)
    if y>theta:
        return 1
    elif y<=theta:
        return 0

inputData=np.array([[0,0],[0,1],[1,0],[1,1]])
print('----NAND-----퍼셉트론')
for i in inputData:
    print(str(i), ">=", str(NAND(i[0],i[1])) ) )

----NAND-----퍼셉트론
[0 0] => 1
[0 1] => 1
[1 0] => 1
[1 1] => 0

```

## XOR게이트

### • 단층 신경망과 다층 신경망의 차이

1958년 로젠블라트가 퍼셉트론을 제안을 했다.

1959년 민스키가 기존 퍼셉트론의 문제점을 지적했는데 XOR분류를 못한다는 문제점을 지적하고

인공지능의 겨울기가 시작되었다.

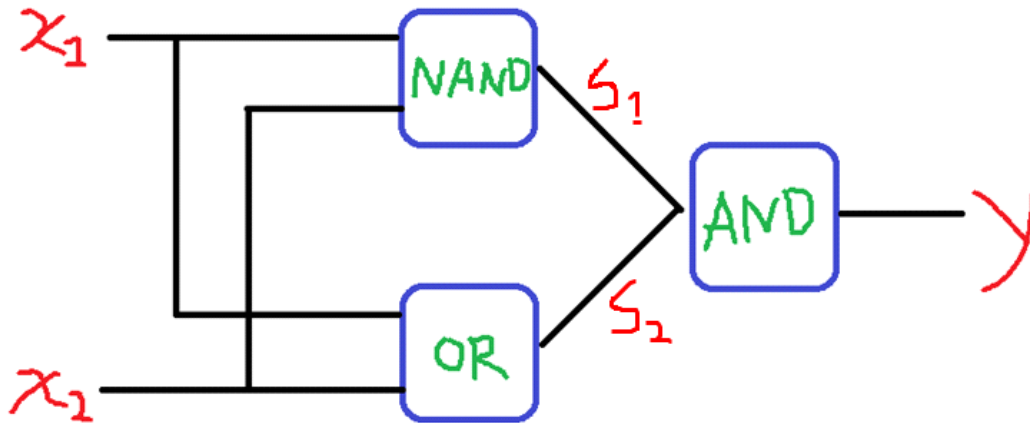
즉, XOR게이트는 다층 신경망으로 구현해야 한다.

1. 단층 : 입력층 ----> 출력층

2. 다층 : 얇은 신경망 : 입력층 --> 은닉층 --> 출력층

깊은 신경망(depp learning) : 입력층 --> 은닉층들 --> 출력층

## XOR 게이트 만드는 방법



문제32. XOR게이트 함수로 생성하시오.

답)

```
import numpy as np
```

```
def OR(x1,x2):
    x=np.array([x1,x2])
    w=np.array([0.5,0.5])
    theta=0.4
    y=np.sum(x*w)
    if y>theta:
        return 1
    elif y<=theta:
        return 0
```

```
def NAND(x1,x2):
    x=np.array([x1,x2])
    w=np.array([-0.5,-0.5])
    theta= -0.7
    y=np.sum(x*w)
    if y>theta:
        return 1
    elif y<=theta:
        return 0
```

```
def AND(x1,x2):
    x=np.array([x1,x2])
    w=np.array([0.5,0.5])
    theta=0.7
    y=np.sum(x*w)
    if y>theta:
        return 1
    elif y<=theta:
        return 0
```

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

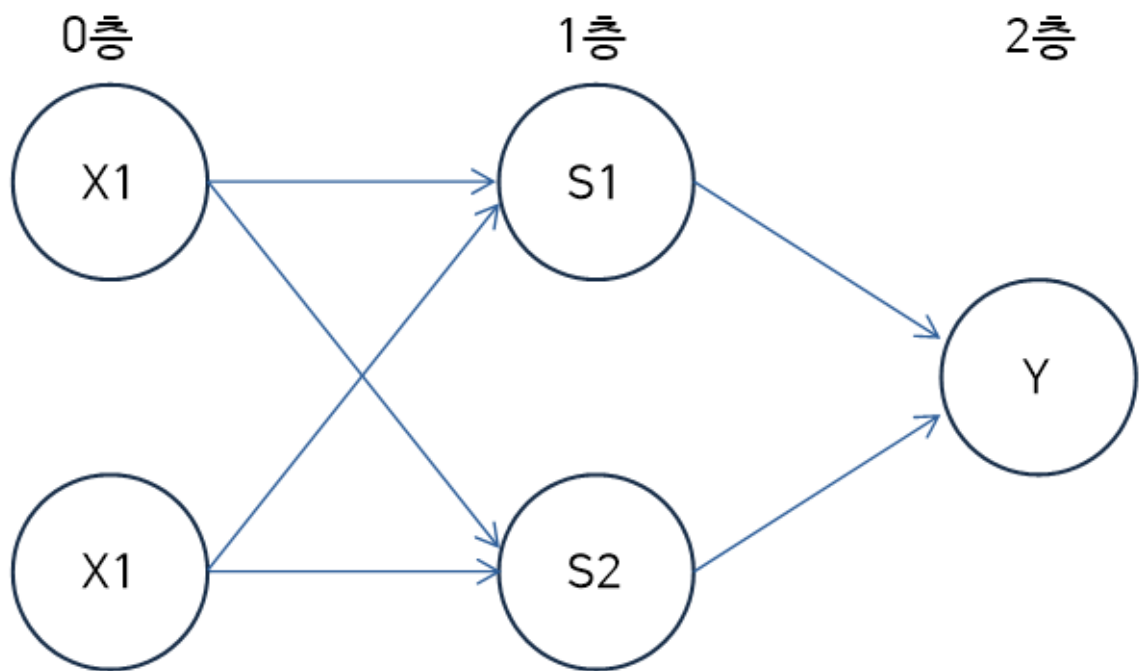
```

inputData=np.array([[0,0],[0,1],[1,0],[1,1]])
print('----XOR-----퍼셉트론')
for i in inputData:
    print(str(i), ">=>", str(XOR(i[0],i[1])) )
----XOR-----퍼셉트론
[0 0] => 0
[0 1] => 1
[1 0] => 1
[1 1] => 0

```

설명 :

### XOR 다층 네트워크 구조



### 3장 신경망

2018년 8월 16일 목요일    오후 4:15

#### 퍼셉트론과 신경망의 차이점?

퍼셉트론 : 원하는 결과를 출력하도록 사람이 직접 가중치르 정해줘야 한다.

ex)	w1	w2	임계값
and	0.5	0.5	0.7
nand	-0.5	-0.5	-0.7
or	0.5	0.5	0.3

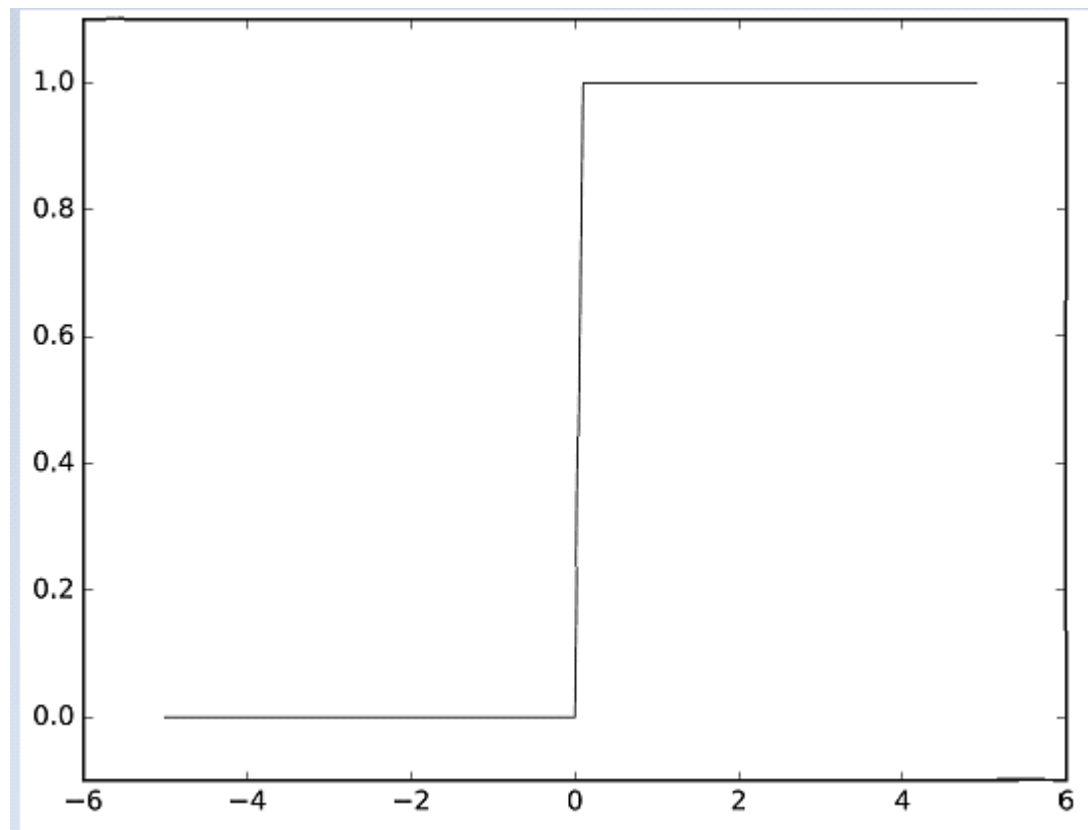
신경망 : 가중치 매개 변수의 적절한 값을 기계가 데이터로부터 학습해서 자동으로 알아낸다.

ex) 필수 알고리즘에서 만들었던 문제 17번

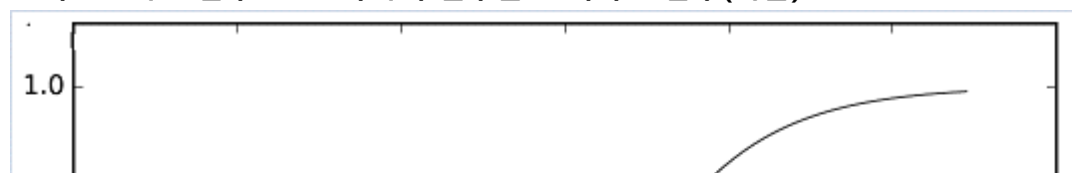
#### 신경망안에 들어가는 활성화 함수 3가지

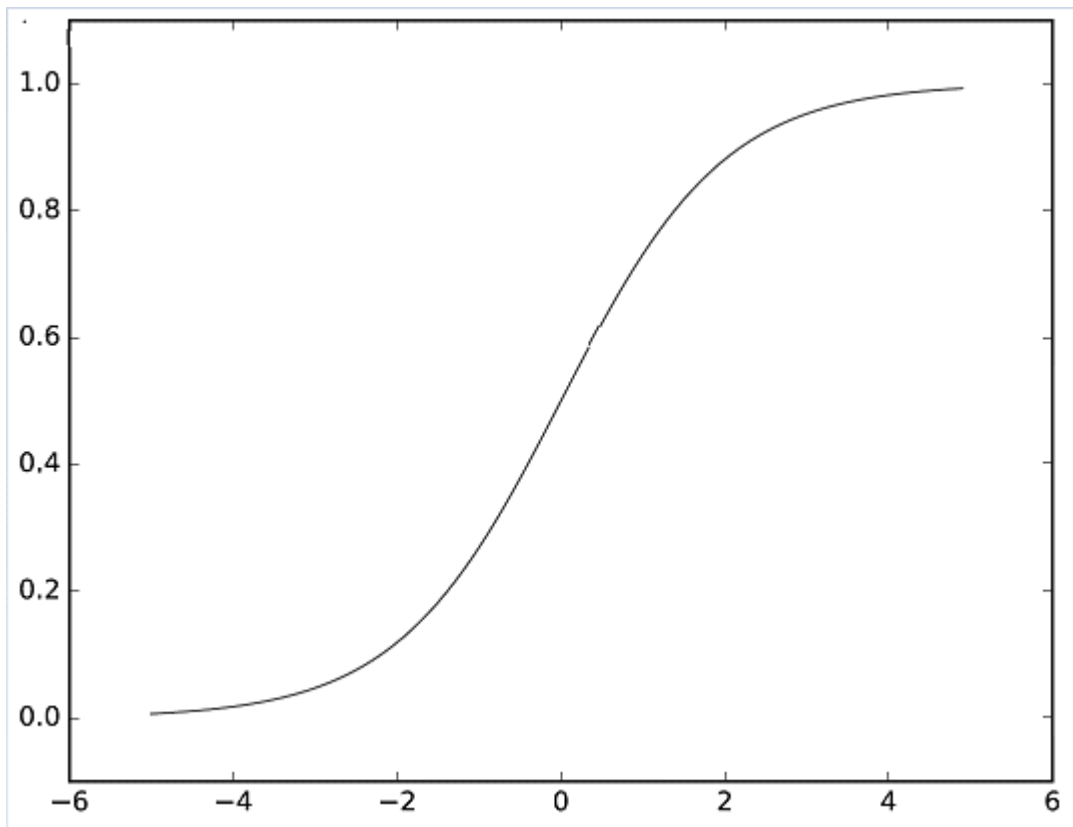
"활성화 함수는 신호의 총합을 받아서 다음 신호로 내보낼지 말지를 결정하는 함수"

##### 1. 계단 함수 : 0 또는 1의 값을 출력하는 함수

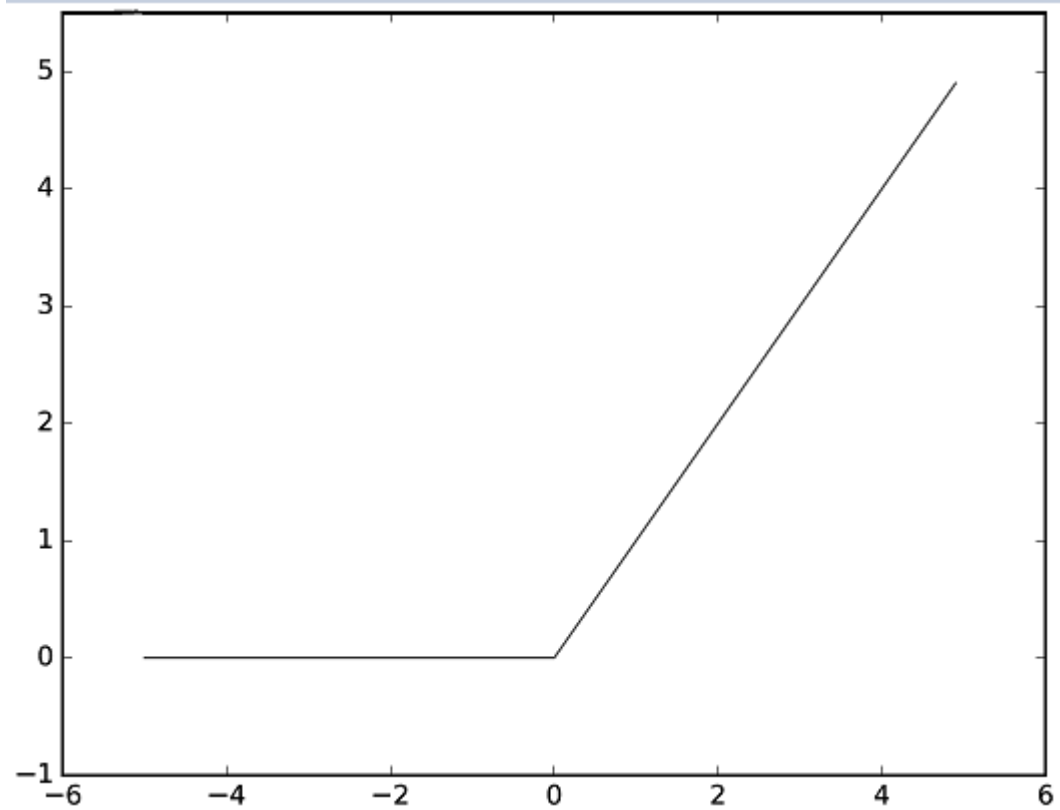


##### 2. 시그모이드 함수 : 0~1사이의 실수를 출력하는 함수(확률)





3. 렐루 함수 : 입력이 0을 넘으면 그 입력값을 그대로 출력하고 0이하면 0을 출력



### 계단함수

문제33. 파이썬으로 계단함수를 만드시오.

답1)  
import numpy as np  
x\_data=np.array([-1,0,1])

```
def step_function(x):
    if x>0:
        return 1
    elif x<=0:
        return 0
for i in x_data:
    print(step_function(i))
0
0
1
```

답2)

```
import numpy as np
x_data=np.array([-1,0,1])

def step_function(x):
    y= x>0
    return y.astype(np.int)

print(step_function(x_data))
```

**문제34. 계단함수를 파이썬으로 시각화 하시오.**

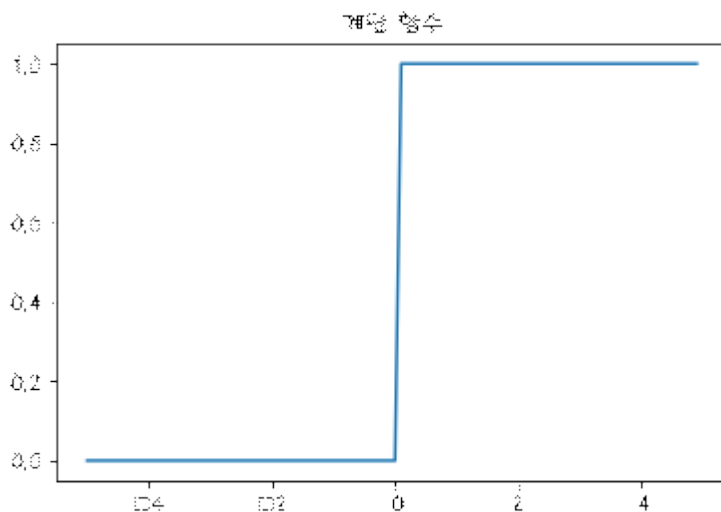
답)

```
import matplotlib.pyplot as plt
from matplotlib import font_manager, rc

x=np.arange(-5,5,0.1)
def step_function(x):
    y= x>0
    return y.astype(np.int)

y=step_function(x)

plt.plot(x,y)
plt.title('계단 함수')
plt.show()
```



**문제34. 아래의 csv2개를 생성하고, company\_name.csv의 기업명이 emp\_plus\_company.csv에 존재하는**

것만 추출하시오.

보기)

company\_name.csv

emp\_plus\_company.csv

name

text

삼성전자

삼성전자신현수

현대자동차

엘지전전자김광록

삼성전기

삼성전기김원섭

기업은행

현대자동차김건태

엘지전자

삼성전자이유진

삼성자동차이상민

기업은행김지우

결과)

삼성전자

현대자동차

삼성전기

엘지전자

답)

```
import pandas as pd
```

```
file = open("C://python_data//company_name.csv")
```

```
file = pd.read_csv(file)
```

```
file2 = open("C://python_data//emp_plus_company.csv")
```

```
file2 = pd.read_csv(file2)
```

```
print(file)
```

```
print(file2)
```

```
a=[]
```

```
for i in file['name']:
```

```
    for j in file2['text']:
```

```
        if i in j:
```

```
            a.append(i)
```

```
print(pd.Series(a).unique())
```

## 시그모이드 함수

### 계단함수(단층) vs 시그모이드 함수(다층)

- 공통점 : 0과 1사이의 데이터만 출력하고 확률이 출력된다.
- 차이점 : 계단함수는 0 or 1을 출력하는데, 시그모이드는 0과 1사이의 실수를 출력.

## 시그모이드 함수식

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

$\exp(-x)$  :  $e(\text{자연상수})^{-x}$



문제35. 시그모이드 함수를 파이썬으로 구현하시오.

보기)

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

답)

```
import numpy as np  
x=np.array([1.0,2.0])  
def sigmoid(x):  
    return 1/(1+np.exp(-x))  
  
print(sigmoid(x))  
[0.73105858 0.88079708]
```

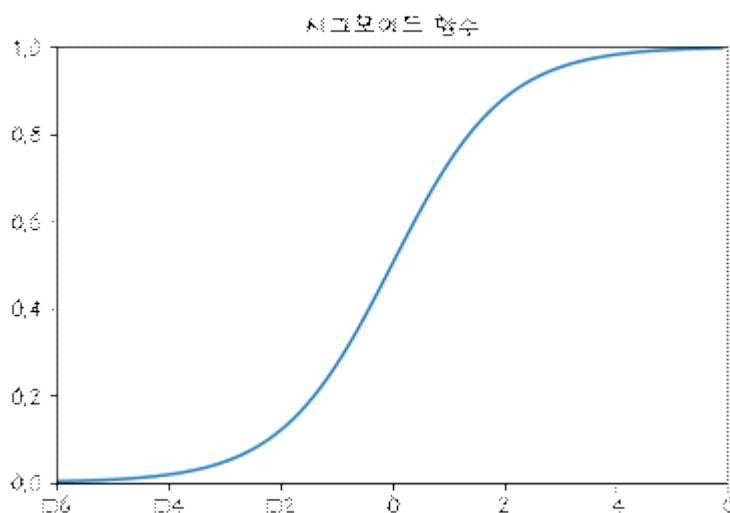
문제36. 시그모이드 함수의 그래프를 그리시오.

답)

```
import numpy as np  
import matplotlib.pyplot as plt  
from matplotlib import font_manager, rc
```

```
x=np.arange(-6.0, 6.0, 0.1)  
def sigmoid(x):  
    return 1/(1+np.exp(-x))
```

```
y= sigmoid(x)  
plt.plot(x,y)  
plt.xlim(-6,6)  
plt.ylim(0,1,1)  
plt.title('시그모이드 함수')  
plt.show()
```



- sigmoid 함수의 식이 왜 아래와 같은지 설명하시오.

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

통계학에서 성공할 확률이 실패할 확률보다 얼마나 큰지를 나타내는 **오즈비율**이라는

값이 있다.

$$\text{오즈비율} = \frac{\text{성공}}{\text{실패}} = \frac{P}{1-P}$$

$\frac{P}{1-P}$  그래프

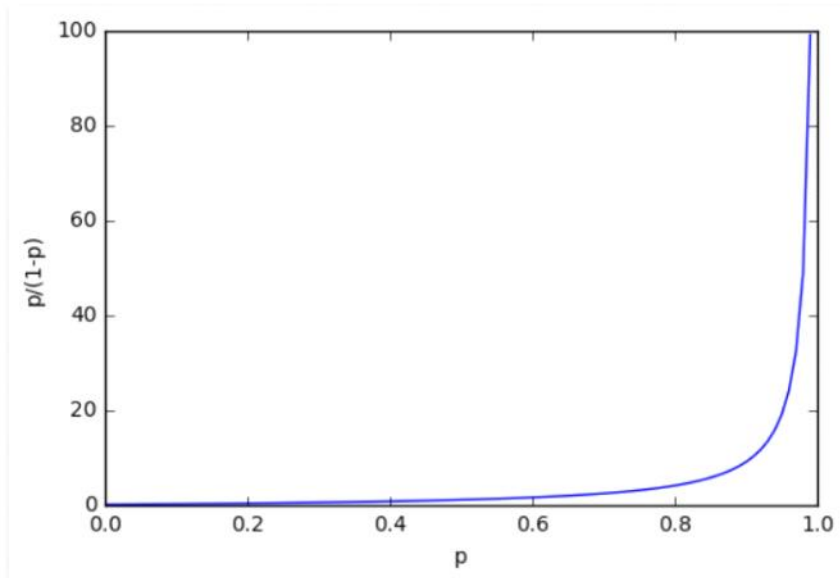


그림 7.  $\left(\frac{p}{1-p}\right)$  그래프

성공할 확률(P)을 0에서 1사이의 값으로 나타내면 실패할 확률(1-P)이다.

위의 그래프처럼 P가 1에 가까우면 오즈비율값이 급격히 커져버리는 현상이 발생한다.

급격히 커져버리는 현상을 방지하기 위해서 이 함수에 로그를 취한게 logit함수 이다.

그래프는 아래와 같다.

$$\text{logit}(p) = \log\left(\frac{p}{1-p}\right)$$

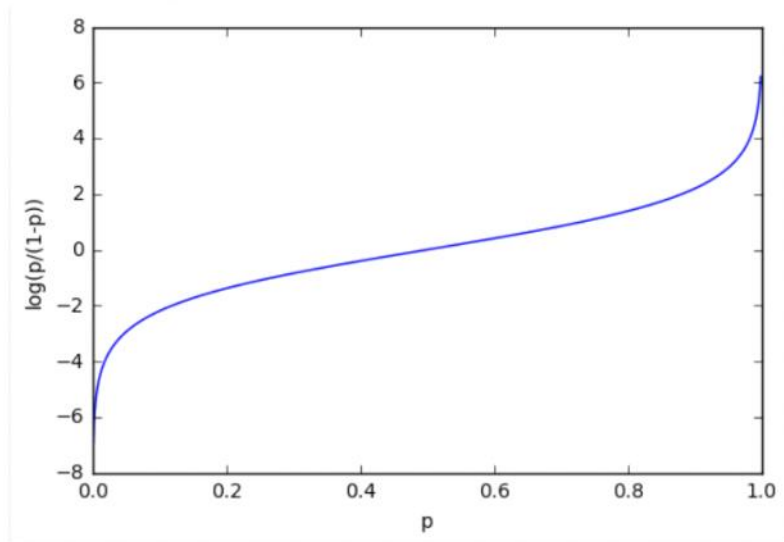


그림 8.  $\log\left(\frac{p}{1-p}\right)$  그래프

p가 0.5일때 실패할 확률 대비 성공할 확률이 0이 된다.

p가 0.5일때는 0이되고,

p가 1일때는 무한히 큰 양수

p가 0일때는 무한히 큰 음수를 가지는 특징이 있다.

$$\log\left(\frac{P}{1-P}\right) = wx+b$$

이 로그함수를 뉴런의 출력값에 따라 확률 P를 구하기 쉽도록 지수함수 형태로 바꾸면

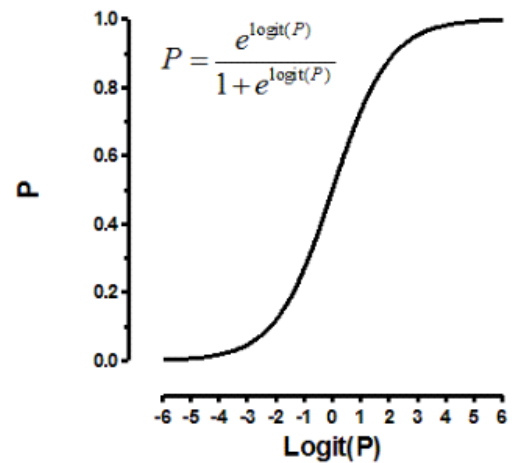
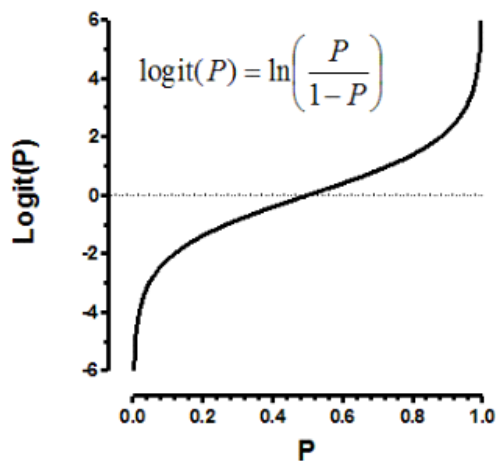
$$P = \frac{1}{1+e^{-(wx+b)}} = \frac{1}{1+e^{-x}}$$

로그함수 -> 지수함수 바꾸는 식

$$\ln\left(\frac{P}{1-P}\right) = a + bX$$

$$\frac{P}{1-P} = e^{a+bX}$$

$$P = \frac{e^{a+bX}}{1 + e^{a+bX}}$$



### Relu 함수

"Relu는 입력이 0을 넘으면 그 입력을 그대로 출력하고 0이하 0을 출력하는 함수"

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

문제37. Relu 함수를 생성하시오.

보기)

```
print(relu(-2)) # 0
print(relu(0.3)) # 0.3
```

답)

```
def relu(x):
    if x>0:
```

```

    return x
else:
    return 0
print(relu(-2))
print(relu(0.3))

0
0.3

```

문제38. numpy를 이용해서 relu함수를 생성하시오.

답)

```

import numpy as np

def relu(x):
    return np.maximum(0,x) #0과 x값중에 큰값을 출력해라
print(relu(-2))
print(relu(0.3))

0
0.3

```

문제39. Relu함수의 그래프를 그리시오.

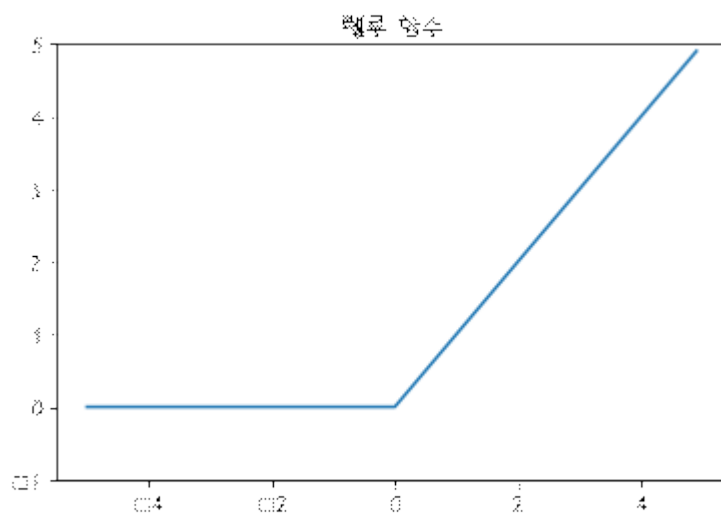
답)

```

import numpy as np

def relu(x):
    return np.maximum(0,x) #0과 x값중에 큰값을 출력해라
x=np.arange(-5,5,0.1)
y=relu(x)
plt.plot(x,y)
plt.ylim(-1,5)
plt.title('렐루 함수')
plt.show()

```



다차원 배열의 계산

ex)

```

import numpy as np

```

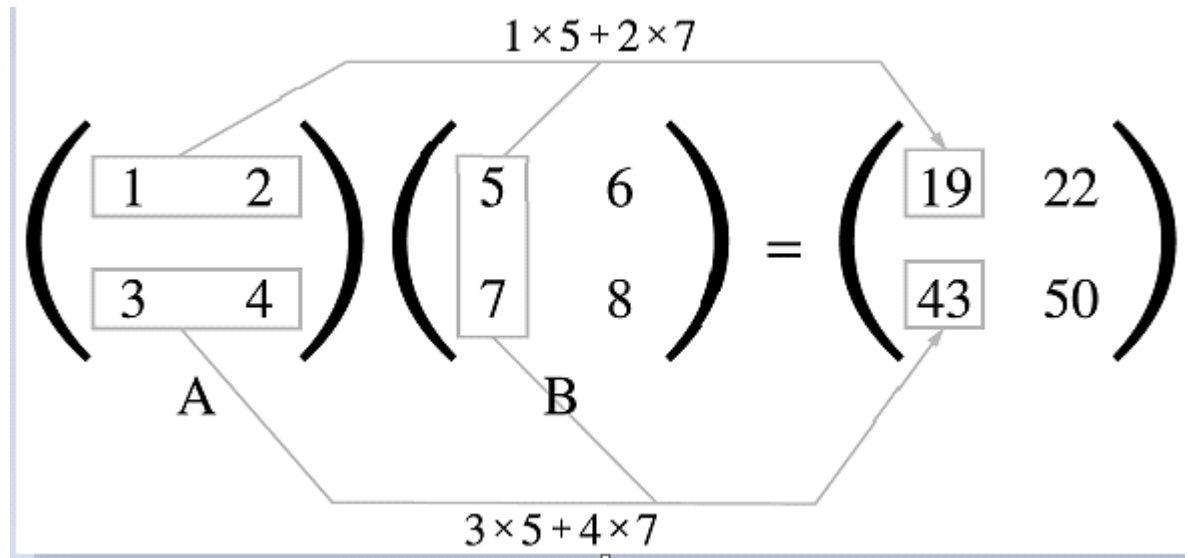
```

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

```

[[1 2]  
 [3 4]  
 [5 6]]  
 2  
 (3, 2)

### 행렬의 내적



문제40. 위의 두개의 행렬의 내적을 numpy로 구현하시오.

답)

```

import numpy as np
a=np.array([[1,2],[3,4]])
b=np.array([[5,6],[7,8]])

print(np.dot(a,b))

```

[[19 22]  
 [43 50]]

답2)

```

import numpy as np
a=np.matrix([ [1,2],[3,4] ])
b=np.matrix([ [5,6],[7,8] ])

print(a*b)

```

[[19 22]  
 [43 50]]

문제41. 아래의 행렬곱(내적)을 파이썬으로 구현하시오.

보기)

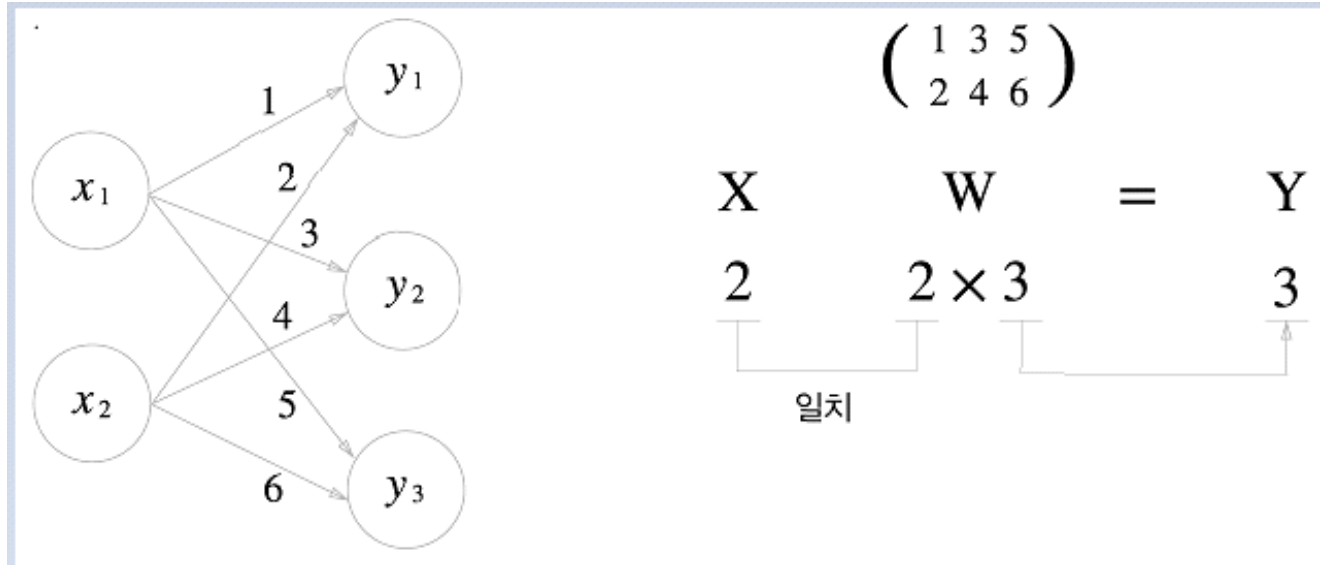
$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix} * \begin{pmatrix} 5 & 6 \\ 7 & 8 \\ 9 & 10 \end{pmatrix} =$$

답)  
import numpy as np  
a=np.array([[1,2,3],[4,5,6]])  
b=np.array([[5,6],[7,8],[9,10]])

print(np.dot(a,b))

[[ 46 52]  
[109 124]]

## 신경망의 내적



문제42. 위의 신경망의  $x_1$ 과  $x_2$ 가 1과 2이면  $y_1$ 과  $y_2$ 가 무엇이 되는가?

보기)  
 $y_1 = x_1 * 1 + x_2 * 2$   
 $y_2 = x_1 * 3 + x_2 * 4$   
 $y_3 = x_1 * 5 + x_2 * 6$

$$(x_1 \ x_2) * \begin{pmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{pmatrix} = ?$$

답)  
import numpy as np  
a=np.array([[1,2]])  
b=np.array([[1,3,5],[2,4,6]])

print(np.dot(a,b))

[[ 5 11 17]]

문제43. 위의 신경망에서 만들어진 가중의 총합인  $y$ 값을 sigmoid함수를 통과 시켜서 나온  $y_{\text{hat}}$ 을 출력

보기)  
[0.2 0.5 0.8]

```

답)
import numpy as np

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

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

res=np.dot(a,b)

print(res)
print(sigmoid(res))

```

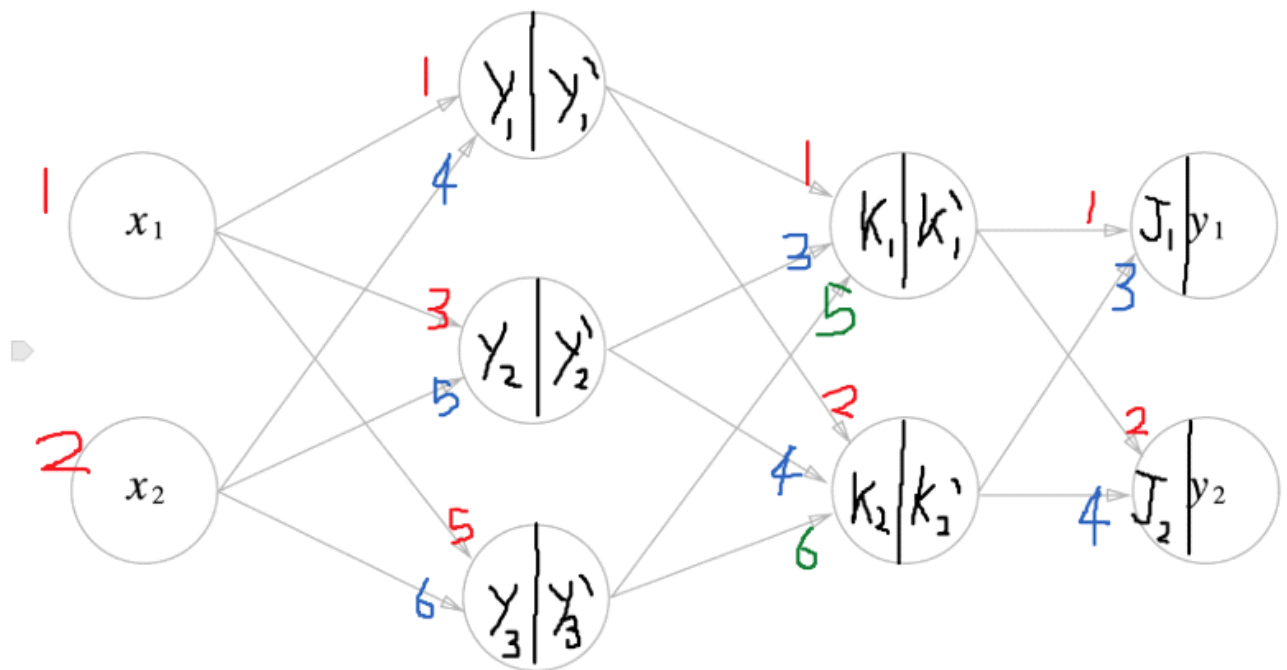
---

```

[[ 5 11 17]]
[[0.99330715 0.9999833 0.99999996]]

```

### 3층 신경망 구현하기



문제44. (마지막) 위의 3층 신경망을 구현하고 J1과 J2값을 구하시오.

```

답)
import numpy as np

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

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

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

```



```

#1층
res=np.dot(a,b)
print(res)
#1층_hat
y_hat=sigmoid(res)
print(y_hat,'\n')

#2층
res2=np.dot(y_hat,c)
print(res2)
#2층_hat
k_hat=sigmoid(res2)
print(k_hat,'\n')

#출력층
res3=np.dot(k_hat,d)
print(res3)
#출력층_hat
j_hat=sigmoid(res3)
print(j_hat)
[ 9 13 17]
[0.99987661 0.99999774 0.99999996]

[ 8.99986962 11.99974392]
[0.99987659 0.99999385]

[3.99985815 5.9997286 ]
[0.98201128 0.99752671]

```

**문제45. 아래의 코드를 이용해서 신경망 1층에서 나온 출력값은 9 13 17을 출력하시오.**

```

보기)
import numpy as np

def sigmoid(num):
    rst = ( 1/( 1+ np.exp(-num) ) )
    return(rst)

x = np.matrix([[1,2]])
w1 = np.matrix([[1,3,5], [4,5,6]])
w2 = np.matrix([[1,2],[3,4],[5,6]])
w3 = np.matrix([[1,2],[3,4]])

y1_hat = []
# 은닉 1층으로!
for i in x:
    y1 = x * w1
    y1_hat.append(sigmoid(y1))
print(y1)
print(y1_hat)

y2_hat = []
# 은닉 2층으로!

```

```

for i in x:
    y2 = y1_hat * w2
    y2_hat.append(sigmoid(y2))

y3_hat = []
# 출력층으로!
for i in x:
    y3 = y2_hat * w3
    y3_hat.append(y3)

print(y3_hat)
[[ 9 13 17]]
[matrix([[0.99987661, 0.99999774, 0.99999996]])]
[matrix([[3.99985815, 5.9997286 ]])]

```

---

문제46. 1층의 시그모이드 함수를 통과한 결과를 y\_hat으로 출력하시오.

답)

```

import numpy as np

def sigmoid(num):
    rst = ( 1/( 1+ np.exp(-num) ) )
    return(rst)

x = np.matrix([[1,2]])
w1 = np.matrix([[1,3,5], [4,5,6]])
w2 = np.matrix([[1,2],[3,4],[5,6]])
w3 = np.matrix([[1,2],[3,4]])

y1_hat = []
# 은닉 1층으로!
for i in x:
    y1 = x * w1
    y1_hat.append(sigmoid(y1))
print(y1)
print(y1_hat)
y2_hat = []
# 은닉 2층으로!
for i in x:
    y2 = y1_hat * w2
    y2_hat.append(sigmoid(y2))

y3_hat = []
# 출력층으로!
for i in x:
    y3 = y2_hat * w3
    y3_hat.append(y3)

print(y3_hat)

[[ 9 13 17]]
[matrix([[0.99987661, 0.99999774, 0.99999996]])]
[matrix([[3.99985815, 5.9997286 ]])]

```

---

문제47. 2층의 시그모이드 함수 통과하기 전 y2값을 출력하시오.

답)

```
import numpy as np

def sigmoid(num):
    rst = ( 1/( 1+ np.exp(-num) ) )
    return(rst)

x = np.matrix([[1,2]])
w1 = np.matrix([[1,3,5], [4,5,6]])
w2 = np.matrix([[1,2],[3,4],[5,6]])
w3 = np.matrix([[1,2],[3,4]])

y1_hat = []
# 은닉 1층으로!
for i in x:
    y1 = x * w1
    y1_hat.append(sigmoid(y1))

y2_hat = []
# 은닉 2층으로!
for i in x:
    y2 = y1_hat * w2
    y2_hat.append(sigmoid(y2))
print(y2)
print(y2_hat)

y3_hat = []
# 출력층으로!
for i in x:
    y3 = y2_hat * w3
    y3_hat.append(y3)

print(y3_hat)
```

---

```
[[ 8.99986962 11.99974392]]
[matrix([[0.99987659, 0.99999385]])]
[matrix([[3.99985815, 5.9997286 ]]])]
```

문제48. 2층에서 시그모이드 함수 통과한 후의 y2\_hat을 출력하시오.

답)

```
import numpy as np

def sigmoid(num):
    rst = ( 1/( 1+ np.exp(-num) ) )
    return(rst)

x = np.matrix([[1,2]])
w1 = np.matrix([[1,3,5], [4,5,6]])
w2 = np.matrix([[1,2],[3,4],[5,6]])
w3 = np.matrix([[1,2],[3,4]])

y1_hat = []
```

```

# 은닉 1층으로!
for i in x:
    y1 = x * w1
    y1_hat.append(sigmoid(y1))

y2_hat = []
# 은닉 2층으로!
for i in x:
    y2 = y1_hat * w2
    y2_hat.append(sigmoid(y2))
print(y2)
print(y2_hat)

y3_hat = []
# 출력층으로!
for i in x:
    y3 = y2_hat * w3
    y3_hat.append(y3)

print(y3_hat)

```

---

```

[[ 8.99986962 11.99974392]]
[matrix([[0.99987659, 0.99999385]])]
[matrix([[3.99985815, 5.9997286 ]])]

```

**문제49. 출력층(3층)의 항등함수에 값이 입력되기 전에 y3값을 출력하시오.**

답)

```

import numpy as np

def sigmoid(num):
    rst = ( 1/( 1+ np.exp(-num) ) )
    return(rst)

x = np.matrix([[1,2]])
w1 = np.matrix([[1,3,5], [4,5,6]])
w2 = np.matrix([[1,2],[3,4],[5,6]])
w3 = np.matrix([[1,2],[3,4]])

y1_hat = []
# 은닉 1층으로!
for i in x:
    y1 = x * w1
    y1_hat.append(sigmoid(y1))
# print(y1)
# print(y1_hat)

y2_hat = []
# 은닉 2층으로!
for i in x:
    y2 = y1_hat * w2
    y2_hat.append(sigmoid(y2))
# print(y2)
# print(y2_hat)

```

```

y3_hat = []
# 출력층으로!
for i in x:
    y3 = y2_hat * w3
    y3_hat.append(sigmoid(y3))
print(y3)
print(y3_hat)

```

---

```

[[3.99985815 5.9997286 ]]
[matrix([[0.98201128, 0.99752671]])]

```

## • 출력층 함수

"출력층의 함수는 그동안 흘러왔던 확률들의 숫자를 취합해서 결론을 내줘야하는 함수"

신경망으로 구현하고자 하는 문제가

### 1. 회귀면 ? **항등함수**

예 : 독립변수(콘크리트 재료 : 자갈 200kg, 시멘트 2포데, ...)

종속변수(콘크리트 강도)

### 2. 분류면 ? **소프트맥스 함수**

예 : 정살 폐사진 vs 폐결절 사진

**문제50. 입력값을 받아 그대로 출력하는 항등함수를 identity\_function이라는 이름으로 생성 하시오.**

답)

```

def identity_function(x):
    return(x)

```

**문제51. 위에서 만든 항등함수를 통과한 결과인 y3\_hat의 결과를 출력하시오.**

답)

```

import numpy as np

def sigmoid(num):
    rst = ( 1/( 1+ np.exp(-num) ) )
    return(rst)

x = np.matrix([[1,2]])
w1 = np.matrix([[1,3,5], [4,5,6]])
w2 = np.matrix([[1,2],[3,4],[5,6]])
w3 = np.matrix([[1,2],[3,4]])

y1_hat = []
# 은닉 1층으로!
for i in x:
    y1 = x * w1
    y1_hat.append(sigmoid(y1))
# print(y1)
# print(y1_hat)

```

```

y2_hat = []
# 은닉 2층으로!
for i in x:
    y2 = y1_hat * w2
    y2_hat.append(sigmoid(y2))
# print(y2)
# print(y2_hat)

y3_hat = []
# 출력층으로!
for i in x:
    y3 = y2_hat * w3
    y3_hat.append(sigmoid(y3))
print(y3)
print(y3_hat)

```

---

```

[[ 3.99985815  5.9997286 ]]
[matrix([[0.98201128, 0.99752671]])]

```

### ###위의 결과를 재귀함수로 만들기!!!!!!

```

답)
import numpy as np
x = np.matrix([[1,2]])
w0= np.matrix([[1,3,5], [4,5,6]])
w1 = np.matrix([[1,2],[3,4],[5,6]])
w2 = np.matrix([[1,2],[3,4]])

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

def ANN(x,w,depth,now=1):
    y=np.dot(x,w)
    if now!=depth:
        w=eval('w'+str(now))
        return ANN(sigmoid(y),w,depth,now+1)
    else: return y

print(ANN(x,w0,3)) #3= 층수
print(sigmoid(ANN(x,w0,3)))

```

---

```

[[ 3.99985815  5.9997286 ]]
[[ 0.98201128  0.99752671 ]]

```

문제52. 위의 코드중에 가중치에 해당하는 코드들을 init\_network()라는 함수로 생성해서 가중치를

딕셔너리에서 가져올 수 있도록 하시오.

```

답)
def init_network():
    network={}

    network['W1']=np.matrix([[1,3,5],[4,5,6]])
    network['W2']=np.matrix([[1,2],[3,4],[5,6]])
    network['W3']=np.matrix([[1,2],[3,4]])

```

```

    return network

network=init_network()
print(network['W1'])
[[1 3 5]
 [4 5 6]]

```

위의 결과로 코드를 수정하시오

```

답)
import numpy as np

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

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

#1층
res=np.dot(x,W1)
print(res)
#1층_hat
y_hat=sigmoid(res)
print(y_hat,'\n')

#2층
res2=np.dot(y_hat,W2)
print(res2)
#2층_hat
k_hat=sigmoid(res2)
print(k_hat,'\n')

#출력층
res3=np.dot(k_hat,W3)
print(res3)
#출력층_hat
j_hat=sigmoid(res3)
print(j_hat)

def init_network():
    network={}

    network['W1']=np.matrix([[1,3,5],[4,5,6]])
    network['W2']=np.matrix([[1,2],[3,4],[5,6]])
    network['W3']=np.matrix([[1,2],[3,4]])
    return network

network=init_network()
W1, W2, W3 = network['W1'], network['W2'], network['W3']

```

문제53. 1층, 2층, 3층 코드를 하나의 함수로 생성하시오.( forward(network, x) )

```

답)
#층함수

```

```
def forward(network, x):
    W1, W2, W3 = network['W1'], network['W2'], network['W3']

    #1층
    res=np.dot(x,W1)
    print(res)
    #1층_hat
    y_hat=sigmoid(res)
    print(y_hat,'\n')

    #2층
    res2=np.dot(y_hat,W2)
    print(res2)
    #2층_hat
    k_hat=sigmoid(res2)
    print(k_hat,'\n')

    #출력층
    res3=np.dot(k_hat,W3)
    print(res3)
    #출력층_hat
    j_hat=sigmoid(res3)
    return j_hat
```

#### #가중치 함수

```
def init_network():
    network={}

    network['W1']=np.matrix([[1,3,5],[4,5,6]])
    network['W2']=np.matrix([[1,2],[3,4],[5,6]])
    network['W3']=np.matrix([[1,2],[3,4]])
    return network
```

#### #구현코드


```
network=init_network()
x = np.matrix([[1,2]])
print(forward(network,x))

[[ 9 13 17]]
[[0.99987661 0.99999774 0.99999996]]

[[ 8.99986962 11.99974392]]
[[0.99987659 0.99999385]]

[[3.99985815 5.9997286 ]]
[[0.98201128 0.99752671]]
```

문제54. 입력값이 1,2가 아니라 4, 5 일때의 위의 3층신경망을 통과한 결과를 출력하시오.  
답)

 jupyter layers\_3.py ✓ 몇 초 전

File Edit View Language



File Edit View Language

```

1  •#층함수
2  def forward(network, x):
3      W1, W2, W3 = network['W1'], network['W2'], network['W3']
4
5      #1층
6      res=np.dot(x,W1)
7      print(res)
8      #1층_hat
9      y_hat=sigmoid(res)
10     print(y_hat, '\n')
11
12     #2층
13     res2=np.dot(y_hat,W2)
14     print(res2)
15     #2층_hat
16     k_hat=sigmoid(res2)
17     print(k_hat, '\n')
18
19     #출력층
20     res3=np.dot(k_hat,W3)
21     print(res3)
22     #출력층_hat
23     j_hat=sigmoid(res3)
24     return j_hat
25
26     #가중치 함수
27     def init_network():
28         network={}
29
30         network['W1']=np.matrix([[1,3,5],[4,5,6]])
31         network['W2']=np.matrix([[1,2],[3,4],[5,6]])
32         network['W3']=np.matrix([[1,2],[3,4]])
33         return network
34

```

```

import layers_3
network=init_network()
x = np.matrix([[4,5]])
print(forward(network,x))

[[24 37 50]]
[[1. 1. 1.]]

[[ 9. 12.]]
[[0.99987661 0.99999386]]

[[3.99985817 5.99972863]]
[[0.98201128 0.99752671]]

```

## 신경망에서 들어가는 함수

1. 은닉층에 들어가는 함수:

- 계단함수
- 시그모이드 함수
- 렐루 함수

2. 출력층에 들어가는 함수:

- 항등함수 (회귀분석)
- 소프트 맥스 함수(분류문제)

### 소프트맥스 함수

"분류를 위한 출력층 함수 : 0~1 숫자를 출력하는 함수"

소프트 맥스 함수 공식

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

소프트 맥스 함수는 지수 함수를 사용하는데 지수 함수라는 것이 쉽게 아주 큰 값을 내뱉는다.

e의10승은 2만이 넘고 e의 100승은 0이 40개가 넘고, e의 1000승은 무한대(inf)를 뜻해서 컴퓨터로는 계산할 수 없다.

예 :

```
import numpy as np
print(np.exp(1))
print(np.exp(1000))
```

```
2.718281828459045
inf
```

$$\begin{aligned}
 y_k &= \frac{\exp(a_k)}{\sum_{i=1}^n \exp(a_i)} = \frac{C \exp(a_k)}{C \sum_{i=1}^n \exp(a_i)} \\
 &= \frac{\exp(a_k + \log C)}{\sum_{i=1}^n \exp(a_i + \log C)} \\
 &= \frac{\exp(a_k + C')}{\sum_{i=1}^n \exp(a_i + C')}
 \end{aligned}$$

여기서 C'을 선택할때 대체로 입력된 값중에 제일 큰것으로 골라준다.

문제55. 아래의 리스트를 무리수(자연상수)에 제곱으로 해서 계산한 값이 무엇인가?

보기)

```
a=np.array([1010,1000,990])
```

결과)

$e^{1010}, e^{1000}, e^{990}$

답)

```
import numpy as np
a=np.array([1010,1000,990])
print(np.exp(a))
[inf inf inf]
```

문제56. 아래의 리스트에서 가장 큰 max값을 출력하시오.

보기)

```
a=np.array([1010,1000,990])
```

답)

```
import numpy as np
a=np.array([1010,1000,990])
c=np.max(a)
print(c)
1010
```

문제57. 아래의 리스트에서 각 요소를 아래의 리스트에서 가장 큰값으로 뺀값이 얼마인지 출력하시오.

```
보기)
a=np.array([1010,1000,990])
```

```
답)
import numpy as np
a=np.array([1010,1000,990])
```

```
c=np.max(a)
print(a-c)
_____
[  0 -10 -20]
```

문제58. 위의 결과를 자연상수 e의 제곱으로 해서 출력된 결과가 무엇인지 확인하시오.

```
답)
import numpy as np
a=np.array([1010,1000,990])

c=np.max(a)

print(np.exp(a-c))
[1.00000000e+00 4.53999298e-05 2.06115362e-09]
```

문제59. 위의 코드를 이용해서 softmax함수를 생성하기 위한 분자식을 구현하시오.

```
답)
import numpy as np
a=np.array([1010,1000,990])
def softmax(a):
    c=np.max(a)
    np_exp=np.exp(a-c)
    return np_exp

print(softmax(a))
_____
[1.00000000e+00 4.53999298e-05 2.06115362e-09]
```

문제60. softmax함수로 분모까지 포함해서 완전히 완성하시오.

```
답)
import numpy as np
a=np.array([1010,1000,990])
def softmax(a):
    c=np.max(a)
    np_exp=np.exp(a-c)
    np_sum=np.sum(np_exp)
    res=np_exp/np_sum
    return res

print(softmax(a))
[9.99954600e-01 4.53978686e-05 2.06106005e-09]
```

문제61. 문제53번 코드에 항등함수를 softmax함수로 변경하시오.

답)

```
def sigmoid(num):
    rst = ( 1/( 1+ np.exp(-num) ) )
    return(rst)
def identity_function(x):
    return(x)

def forward(network, x):
    W1, W2, W3 = network['W1'], network['W2'], network['W3']

    #1층
    res=np.dot(x,W1)
    print(res)
    #1층_hat
    y_hat=sigmoid(res)
    print(y_hat,'\n')

    #2층
    res2=np.dot(y_hat,W2)
    print(res2)
    #2층_hat
    k_hat=sigmoid(res2)
    print(k_hat,'\n')

    #출력층
    res3=np.dot(k_hat,W3)
    print(res3)
    #출력층_hat
    j_hat=softmax(res3)
    return j_hat

#가중치 함수
def init_network():
    network={}

    network['W1']=np.matrix([[1,3,5],[4,5,6]])
    network['W2']=np.matrix([[1,2],[3,4],[5,6]])
    network['W3']=np.matrix([[1,2],[3,4]])
    return network

def softmax(a):
    c=np.max(a)
    np_exp=np.exp(a-c)
    np_sum=np.sum(np_exp)
    res=np_exp/np_sum
    return res

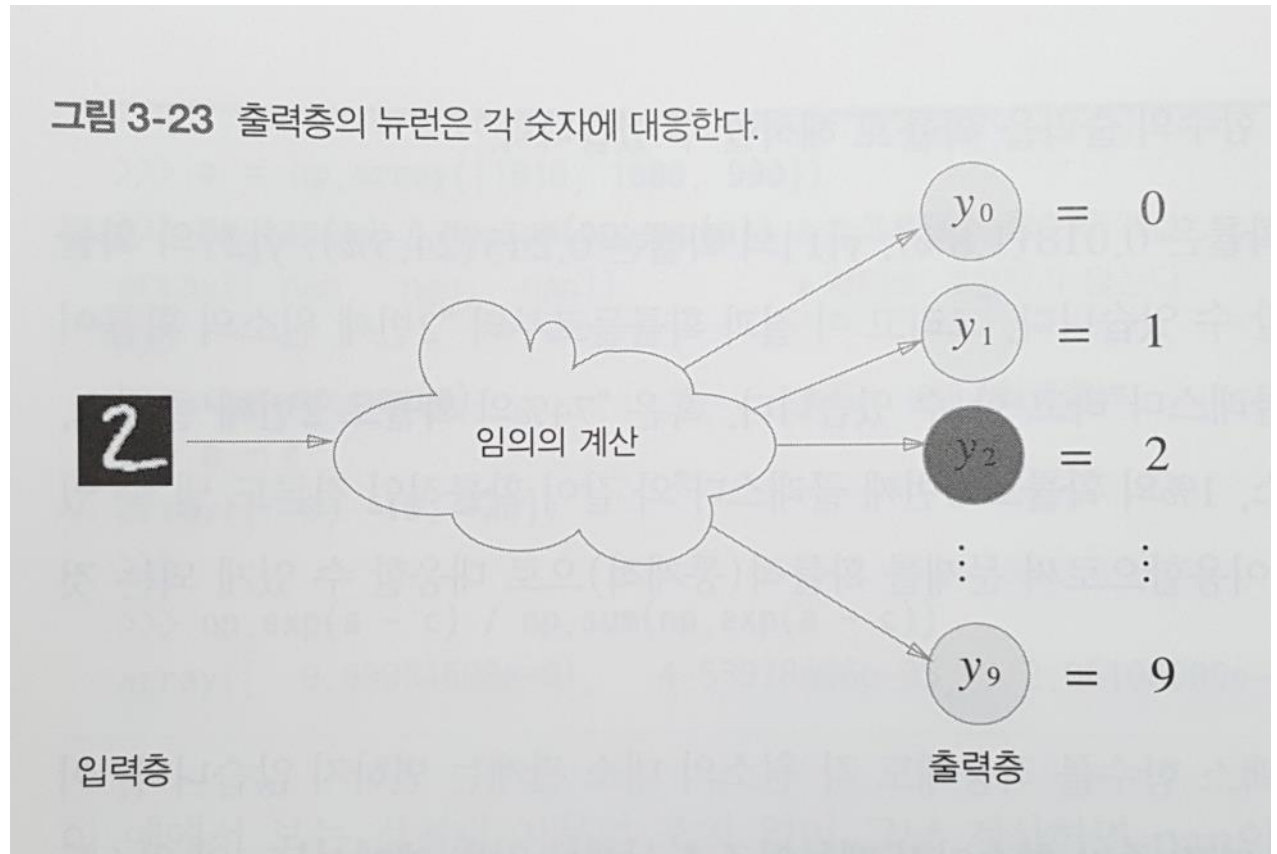
#구현코드
network=init_network()
x = np.matrix([[1,2]])
print(forward(network,x))
```

```
[[ 9 13 17]]  
[[0.99987661 0.99999774 0.99999996]]
```

```
[[ 8.99986962 11.99974392]]  
[[0.99987659 0.99999385]]
```

```
[[3.99985815 5.9997286 ]]  
[[0.11921653 0.88078347]]
```

두개를 더하면 1이다!!



학습 --> 소프트 맥스 함수를 사용o

테스트 --> 소프트 맥스 함수를 사용x

why? 자원 낭비르 줄이기 위해

## • 출력층의 뉴런수 정하기

출력층 뉴런수?

1. 이 사진이 개와 고양이만 분류 --> 2개
2. 이 사진이 정상폐 사진, 질병 폐사진 분류 --> 2개
3. mnist 데이터(필기체 숫자0~9까지 사진) --> 10개

## • mnist(손글씨 필기체) 데이터를 파이썬으로 로드하는 방법

1. 책 소스코드와 데이터를 다운로드 받는다.
2. dataset이라는 폴더를 워킹 디렉토리에 가져다 둔다.  
(C:\Users\Kim won sub\Anaconda3)

3. 아래의 파이썬 코드를 실행해서 필기체 데이터 하나를 시각화 한다.

4. 아래의 코드를 실행한다.

```
-----
# coding: utf-8
import sys, os

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

def img_show(img): #이미지를 출력하는 함수
    pil_img = Image.fromarray(np.uint8(img)) #파이썬 이미지 객체로 변환
    pil_img.show()

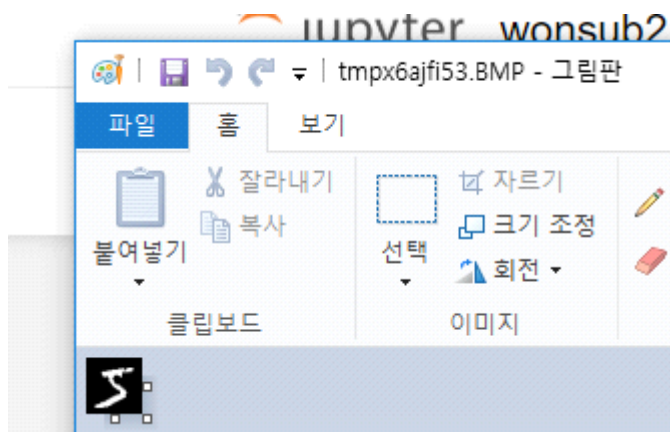
(x_train, t_train), (x_test, t_test) = load_mnist(flatten=True, normalize=False)
#(훈련데이터, 훈련데이터라벨),(테스트데이터,테스트 데이터 라벨)
img = x_train[0]
label = t_train[0]
print(label) # 5

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

img_show(img)
-----
```

결과)

```
5
( 784, )
( 28, 28)
```



문제62. 아래 코드의 dataset패키지의 mnist.py에 load\_mnist라는 함수가 있는지 확인하시오.

```
보기)
import sys, os

sys.path.append(os.pardir)
```

```
import numpy as np
from dataset.mnist import load_mnist
from PIL import Image

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

print(len(t_train))
```

답)

파이참에서는 ctrl키를 누르고 클릭하면 확인 할 수있다.

### 문제63. x\_train데이터를 print로 확인하시오.

답)

```
# coding: utf-8
import sys, os

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

def img_show(img): #이미지를 출력하는 함수
    pil_img = Image.fromarray(np.uint8(img)) #파이썬 이미지 객체로 변환
    pil_img.show()

(x_train, t_train), (x_test, t_test) = load_mnist(flatten=True, normalize=False)
#(훈련데이터, 훈련데이터라벨),(테스트데이터,테스트 데이터 라벨)
print(x_train)
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)
```

```
img_show(img)
[ [0 0 0 ... 0 0 0]
  [0 0 0 ... 0 0 0]
  [0 0 0 ... 0 0 0]
  ...
  [0 0 0 ... 0 0 0]
  [0 0 0 ... 0 0 0]
  [0 0 0 ... 0 0 0]
  _
```

### 문제64. x\_train데이터가 6만장이 맞는지 확인하시오.

답)

```
# coding: utf-8
import sys, os

sys.path.append(os.pardir) # 부모 디렉터리의 파일을 가져올 수 있도록 설정
```



```

import numpy as np
from dataset.mnist import load_mnist
from PIL import Image

def img_show(img): #이미지를 출력하는 함수
    pil_img = Image.fromarray(np.uint8(img)) #파이썬 이미지 객체로 변환
    pil_img.show()

(x_train, t_train), (x_test, t_test) = load_mnist(flatten=True, normalize=False)
#(훈련데이터, 훈련데이터라벨),(테스트데이터,테스트 데이터 라벨)
print(len(x_train))
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)

img_show(img)
60000
5
(784,)
(28, 28)

```

**문제65. x\_train 6만개의 이미지 중에서 제일 첫번째 이미지 한 개를 출력하시오.**

```

답)
# coding: utf-8
import sys, os

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

def img_show(img): #이미지를 출력하는 함수
    pil_img = Image.fromarray(np.uint8(img)) #파이썬 이미지 객체로 변환
    pil_img.show()

(x_train, t_train), (x_test, t_test) = load_mnist(flatten=False, normalize=False)
#(훈련데이터, 훈련데이터라벨),(테스트데이터,테스트 데이터 라벨)
print(x_train[0])
print(t_train[0])
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)

```

```
[[[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0 0 3 18 18 18 126 136 175 26 166 255 247 127 0 0 0 0]
[0 0 0 0 0 0 0 0 0 30 36 94 154 170 253 253 253 253 253 225 172 253 242 195 64 0 0 0 0]
[0 0 0 0 0 0 0 0 49 238 253 253 253 253 253 253 253 251 93 82 82 56 39 0 0 0 0 0]
[0 0 0 0 0 0 0 0 18 219 253 253 253 253 253 198 182 247 241 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 80 156 107 253 253 205 11 0 43 154 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 14 1 154 253 90 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 139 253 190 2 0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 11 190 253 70 0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 35 241 225 160 108 1 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 81 240 253 253 119 25 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 45 186 253 253 150 27 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0 0 16 93 252 253 187 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 249 253 249 64 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 46 130 183 253 253 207 2 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 39 148 229 253 253 253 250 182 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 24 114 221 253 253 253 253 201 78 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 23 66 213 253 253 253 253 198 81 2 0 0 0 0 0 0 0]
[0 0 0 0 0 0 18 171 219 253 253 253 253 195 80 9 0 0 0 0 0 0 0 0]
[0 0 0 0 55 172 226 253 253 253 253 244 133 11 0 0 0 0 0 0 0 0 0]
[0 0 0 0 136 253 253 253 212 135 132 16 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]]]
```

```
x = [[0.1,0.05,0.1,0.0,0.05,0.1,0.0,0.1,0.0,0.0],
      [0.1,0.05,0.2,0.0,0.05,0.1,0.0,0.6,0.0,0.0],
      [0.0,0.05,0.3,0.0,0.05,0.1,0.0,0.6,0.0,0.0],
      [0.0,0.05,0.4,0.0,0.05,0.0,0.0,0.5,0.0,0.0],
      [0.0,0.05,0.5,0.0,0.05,0.0,0.0,0.4,0.0,0.0],
      [0.0,0.05,0.6,0.0,0.05,0.0,0.0,0.3,0.0,0.0],
      [0.0,0.05,0.7,0.0,0.05,0.0,0.0,0.2,0.0,0.0],
      [0.0,0.1,0.8,0.0,0.1,0.0,0.0,0.2,0.0,0.0],
      [0.0,0.05,0.9,0.0,0.05,0.0,0.0,0.0,0.0,0.0]]
```

```
x = [[0.1,0.05,0.1,0.0,0.05,0.1,0.0,0.1,0.0,0.0],
      [0.1,0.05,0.2,0.0,0.05,0.1,0.0,0.6,0.0,0.0],
      [0.0,0.05,0.3,0.0,0.05,0.1,0.0,0.6,0.0,0.0],
      [0.0,0.05,0.4,0.0,0.05,0.0,0.0,0.5,0.0,0.0],
      [0.0,0.05,0.5,0.0,0.05,0.0,0.0,0.4,0.0,0.0],
      [0.0,0.05,0.6,0.0,0.05,0.0,0.0,0.3,0.0,0.0],
      [0.0,0.05,0.7,0.0,0.05,0.0,0.0,0.2,0.0,0.0],
      [0.0,0.1,0.8,0.0,0.1,0.0,0.0,0.2,0.0,0.0],
      [0.0,0.05,0.9,0.0,0.05,0.0,0.0,0.0,0.0,0.0]]
```

```
x = [[0.1,0.05,0.1,0.0,0.05,0.1,0.0,0.1,0.0,0.0],
      [0.1,0.05,0.2,0.0,0.05,0.1,0.0,0.6,0.0,0.0],
      [0.0,0.05,0.3,0.0,0.05,0.1,0.0,0.6,0.0,0.0],
      [0.0,0.05,0.4,0.0,0.05,0.0,0.0,0.5,0.0,0.0],
      [0.0,0.05,0.5,0.0,0.05,0.0,0.0,0.4,0.0,0.0],
```

```

[0.0,0.05,0.6,0.0,0.05,0.0,0.0,0.3,0.0,0.0],
[0.0,0.05,0.7,0.0,0.05,0.0,0.0,0.2,0.0,0.0],
[0.0,0.1,0.8,0.0,0.1,0.0,0.0,0.2,0.0,0.0],
[0.0,0.05,0.9,0.0,0.05,0.0,0.0,0.0,0.0,0.0]]
import numpy as np
a=np.array(x)
print(a.ndim)

2

```

**문제68. 위의 리스트를 1차원으로 변경하시오.**

```

답)
x = [[0.1,0.05,0.1,0.0,0.05,0.1,0.0,0.1,0.0,0.0],
      [0.1,0.05,0.2,0.0,0.05,0.1,0.0,0.6,0.0,0.0],
      [0.0,0.05,0.3,0.0,0.05,0.1,0.0,0.6,0.0,0.0],
      [0.0,0.05,0.4,0.0,0.05,0.0,0.0,0.5,0.0,0.0],
      [0.0,0.05,0.5,0.0,0.05,0.0,0.0,0.4,0.0,0.0],
      [0.0,0.05,0.6,0.0,0.05,0.0,0.0,0.3,0.0,0.0],
      [0.0,0.05,0.7,0.0,0.05,0.0,0.0,0.2,0.0,0.0],
      [0.0,0.1,0.8,0.0,0.1,0.0,0.0,0.2,0.0,0.0],
      [0.0,0.05,0.9,0.0,0.05,0.0,0.0,0.0,0.0,0.0]]
import numpy as np
a=np.array(x)
print(a.ndim)
a=a.flatten()
print(a.ndim)

2
1

```

**문제69. mnist의 훈련데이터 (x\_train)의 차원이 어떻게 되는지 확인하시오.**

```

답)
# coding: utf-8
import sys, os

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

def img_show(img): #이미지를 출력하는 함수
    pil_img = Image.fromarray(np.uint8(img)) #파이썬 이미지 객체로 변환
    pil_img.show()

(x_train, t_train), (x_test, t_test) = load_mnist(flatten=True, normalize=False)
#(훈련데이터, 훈련데이터라벨),(테스트데이터,테스트 데이터 라벨)
print(x_train.ndim)
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)

```

```
img_show(img)
2
5
(784,)
(28, 28)
```

문제70. flatten을 False로 설정하고, 훈련데이터의 모양을 확인하시오.

```
답)
# coding: utf-8
import sys, os
sys.path.append(os.pardir) # 부모 디렉터리의 파일을 가져올 수 있도록 설정
import numpy as np
from dataset.mnist import load_mnist
from PIL import Image

def img_show(img): #이미지를 출력하는 함수
    pil_img = Image.fromarray(np.uint8(img)) #파이썬 이미지 객체로 변환
    pil_img.show()

(x_train, t_train), (x_test, t_test) = load_mnist(flatten=False, normalize=False)
#(훈련데이터, 훈련데이터라벨),(테스트데이터,테스트 데이터 라벨)
print(x_train.shape)
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)

img_show(img)
(60000, 1, 28, 28)
5
(1, 28, 28)
(28, 28)

설명 : 전체 장수, 흑백(1), 가로,세로
만약 컬러면 RGB값으로 3이 출력
```

문제71. t\_train[0]가 어떤 숫자인지 확인하는데 one hot encoding 했을때와 안했을때의 차이를 확인하시오.

```
답)
# coding: utf-8
import sys, os
sys.path.append(os.pardir) # 부모 디렉터리의 파일을 가져올 수 있도록 설정
import numpy as np
from dataset.mnist import load_mnist
```

```
from PIL import Image
```

```
def img_show(img): #이미지를 출력하는 함수
```

```
    pil_img = Image.fromarray(np.uint8(img)) #파이썬 이미지 객체로 변환
    pil_img.show()
```

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

```
#(훈련데이터, 훈련데이터라벨),(테스트데이터,테스트 데이터 라벨)
```

```
print(t_train[0])
```

```
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)
```

```
img_show(img)
```

**False일때**

```
5
```

```
5
```

```
(1, 28, 28)
```

```
(28, 28)
```

**True일때**

```
[0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
```

```
[0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
```

```
(1, 28, 28)
```

```
(28, 28)
```

**문제72. x\_train[0] 요소를 시각화 하시오.(img\_show함수를 사용)**

답)

```
# coding: utf-8
```

```
import sys, os
```

```
sys.path.append(os.pardir) # 부모 디렉터리의 파일을 가져올 수 있도록 설정
```

```
import numpy as np
```

```
from dataset.mnist import load_mnist
```

```
from PIL import Image
```

```
def img_show(img): #이미지를 출력하는 함수
```

```
    pil_img = Image.fromarray(np.uint8(img)) #파이썬 이미지 객체로 변환
    pil_img.show()
```

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

```
#(훈련데이터, 훈련데이터라벨),(테스트데이터,테스트 데이터 라벨)
```

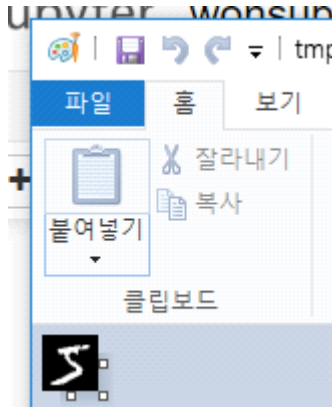
```

print(t_train[0])
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 (중요★★★★★)
print(img.shape) # (28, 28)

img_show(img)

```



문제73. mnist데이터를 가져오는 코드를 가지고 아래의 get\_data()함수를 생성하고 아래와 같이 실행되도록

하십시오.

```

보기)
x,t = get_data()
print(x)
print(t)

```

```

답)
# coding: utf-8
import sys, os

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

```

```

def img_show(img): #이미지를 출력하는 함수
    pil_img = Image.fromarray(np.uint8(img)) #파이썬 이미지 객체로 변환
    pil_img.show()

def get_data():
    (x_train, t_train), (x_test, t_test) = load_mnist(flatten=True, normalize=True,
one_hot_label=False)
    #(훈련데이터, 훈련데이터라벨),(테스트데이터,테스트 데이터 라벨)
    return x_test, t_test
x,t=get_data()
print(x)
print(t)

```

```

[[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]
[7 2 1 ... 4 5 6]

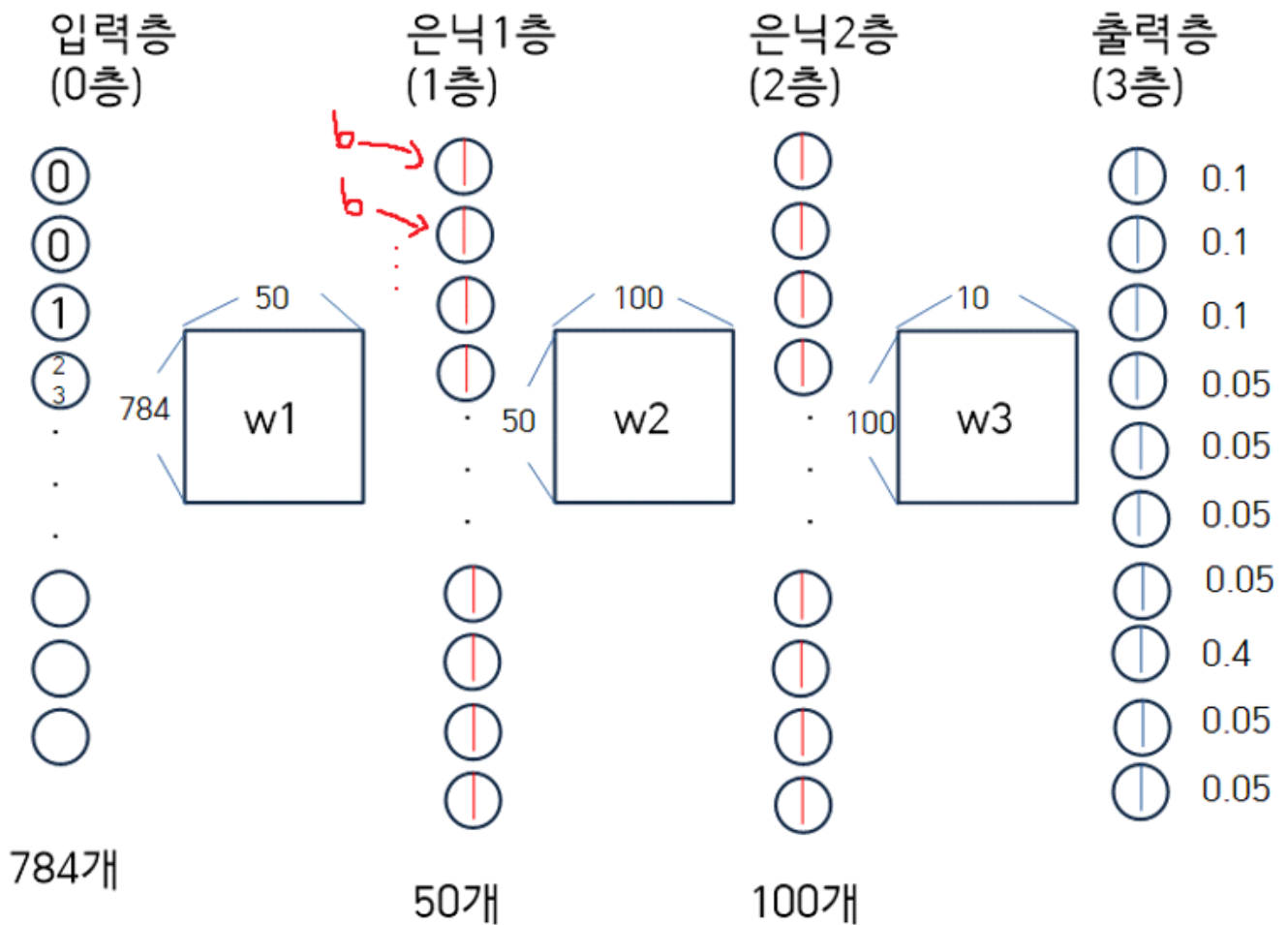
print(x.shape)
print(len(t))

(10000, 784)
10000

```

문제74. 테스트 데이터 10000장을 3층 신경망으로 입력했을때의 그림을 그리시오.(꼭 이해하기!!)

답)



$$1 \times 784 \odot 784 \times 50 = 1 \times 50 \odot 50 \times 100 = 1 \times 100 \odot 100 \times 10$$

문제75. mnist 필기체 데이터를 위해 신경망에서 사용할 가중치(weight)와 바이어스(bias)가 셋팅되어있는

sample\_weight.pkl을 로드하는 init\_network() 함수를 생성하시오.

답)

```
import pickle
```

```
def get_data():
    (x_train, t_train), (x_test, t_test) = load_mnist(flatten=True, normalize=True,
one_hot_label=False)
    #(훈련데이터, 훈련데이터라벨),(테스트데이터,테스트 데이터 라벨)
    return x_test, t_test
def init_network():
    with open("sample_weight.pkl", 'rb') as f:
        network=pickle.load(f)
    return network
network=init_network()
print(network["W1"].shape)
print(network["W2"].shape)
print(network["W3"].shape)
print(network["b1"].shape)
print(network["b2"].shape)
print(network["b3"].shape)
```

```
(784, 50)
(50, 100)
(100, 10)
(50, )
(100, )
(10, )
```

**문제76. mnist 데이터 셋에서 숫자 하나를 입력하면 그 숫자가 어떤 숫자인지 예측하는 predict라는 함수를**

**생성하시오.**

답)

```
import pickle
```

```
def sigmoid(num):
    rst = ( 1/( 1+ np.exp(-num) ) )
    return(rst)

def softmax(a):
    c=np.max(a)
    np_exp=np.exp(a-c)
    np_sum=np.sum(np_exp)
    res=np_exp/np_sum
    return res

def get_data():
    (x_train, t_train), (x_test, t_test) = load_mnist(flatten=True, normalize=True,
one_hot_label=False)
    #(훈련데이터, 훈련데이터라벨),(테스트데이터,테스트 데이터 라벨)
    return x_test, t_test
def init_network():
    with open("sample_weight.pkl", 'rb') as f:
        network=pickle.load(f)
    return network
```



```

def predict(network,x):
    W1, W2, W3 = network['W1'],network['W2'],network['W3']
    b1, b2, b3 = network['b1'],network['b2'],network['b3']

    a1 = np.dot(x,W1)+b1
    z1 = sigmoid(a1)
    a2 = np.dot(z1,W2)+b2
    z2 = sigmoid(a2)
    a3 = np.dot(z2,W3)+b3
    y = softmax(a3)

    return y
x,t = get_data()
network=init_network()
y=predict(network,x[0])
print(y)

```

---

```

[8.4412488e-05 2.6350631e-06 7.1549421e-04 1.2586262e-03 1.1727954e-06
 4.4990808e-05 1.6269318e-08 9.9706501e-01 9.3744793e-06 8.1831159e-04]

```

설명 : x[0]이 7이니까 7번째가 제일 크다.

문제77. A4지에 필기체 숫자를 쓰고 사진을 찍어서 png또는 ipg로 만든후에 numpy 배열로 변환해서 위

의3층 신경망에 넣고 잘 예측하는지 확인하시오.

답)

```

import sys, os
sys.path.append(os.pardir)
import numpy as np
from dataset.mnist import load_mnist
import pickle
import numpy as np

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

def softmax(a):
    return np.exp(a-np.max(a))/np.sum(np.exp(a-np.max(a)))

def get_data():
    (x_train, t_train), (x_test, t_test) = \
    load_mnist(flatten=True, normalize=False, one_hot_label=False)
    return x_test, t_test

def forward(w,x,now=1):
    y=np.dot(x,w)
    if now!=3:
        w=eval('W'+str(now+1))
        return forward(w,sigmoid(y),now+1)
    else: return softmax(y)

def init_network():

```

```

with open('sample_weight.pkl','rb') as f:
    network=pickle.load(f)
    return network

def predict(network,x,now=1):
    W1,W2,W3=network['W1'],network['W2'],network['W3']
    b1,b2,b3=network['b1'],network['b2'],network['b3']
    w='W'+str(now)
    b=eval('b'+str(now))
    y=np.dot(x,network[w])+b
    if now!=3:
        return predict(network,sigmoid(y),now+1)
    else: return softmax(y)

import matplotlib.image as img
img=img.imread('C:\\python_data\\3.jpg')
img=img[:, :,2] #grayscale로 변환
img1=img.flatten()
network=init_network()
y=predict(network,img1)
print(np.where(y==max(y)))
(array([3], dtype=int64),)

```

**설명 :**

컬러 사진(red, green, blue)은 구분을 하기 어려워서 흑백으로 변환해줘야한다.

**문제78. 'C:\\python\_data\\3.jpg' 파일을(28,28,3) ---> (28,28,1)로 변경하시오.(grayscale)**

답)

```

import cv2
j = 'C:\\python_data\\3.jpg'
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

```

```

img = mpimg.imread(j)
print(img.shape)
img=img[:, :,2]
print(img.shape)
(28, 28, 3)
(28, 28)

```

**문제79. 흑백으로 변경한 숫자3을 3층 신경망에 넣고 숫자3을 컴퓨터가 맞추는지 확인하시오.**

답)

```

import sys, os
sys.path.append(os.pardir)
import numpy as np
from dataset.mnist import load_mnist
import pickle
import numpy as np

```

```

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

def softmax(a):
    return np.exp(a-np.max(a))/np.sum(np.exp(a-np.max(a)))

def get_data():
    (x_train, t_train), (x_test, t_test) = \
    load_mnist(flatten=True, normalize=False, one_hot_label=False)
    return x_test, t_test

def forward(w,x,now=1):
    y=np.dot(x,w)
    if now!=3:
        w=eval('W'+str(now+1))
        return forward(w,sigmoid(y),now+1)
    else: return softmax(y)

def init_network():
    with open('sample_weight.pkl','rb') as f:
        network=pickle.load(f)
        return network

def predict(network,x,now=1):
    W1,W2,W3=network['W1'],network['W2'],network['W3']
    b1,b2,b3=network['b1'],network['b2'],network['b3']
    w='W'+str(now)
    b=eval('b'+str(now))
    y=np.dot(x,network[w])+b
    if now!=3:
        return predict(network,sigmoid(y),now+1)
    else: return softmax(y)

import matplotlib.image as img
img=img.imread('C:\\python_data\\3.jpg')
img=img[:, :,2] #grayscale로 변환
img1=img.flatten()
network=init_network()
y=predict(network,img1)
print(y)
print(np.where(y==max(y)))
[4.4447958e-02 1.4861076e-04 4.7776058e-02 4.7793990e-01 4.9663293e-05
 8.5526943e-02 1.0935562e-04 3.3125555e-04 3.4241295e-01 1.2573689e-03]
(array([3], dtype=int64),)

```

**문제80. 아래의 10개의 원소를 갖는 x라는 리스트를 만들고 x리스트에 가장 큰 원소가 몇번째 인덱스인지**

**알아내시오.(argmax)**

보기)

x=[0.05, 0.01, 0.02, 0.02, 0.1, 0.2, 0.3, 0.4, 0.05, 0.04]

답)

import numpy as np

```
x=[0.05, 0.01, 0.02, 0.02, 0.1, 0.2, 0.3, 0.4, 0.05, 0.04]
y=np.argmax(x,axis=0)
print(y)
7
```

문제81. 문제79번 코드를 수정해서 np.where대신에 np.argmax를 사용하시오.

```
답)
import sys, os
sys.path.append(os.pardir)
import numpy as np
from dataset.mnist import load_mnist
import pickle
import numpy as np

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

def softmax(a):
    return np.exp(a-np.max(a))/np.sum(np.exp(a-np.max(a)))

def get_data():
    (x_train, t_train), (x_test, t_test) = \
    load_mnist(flatten=True, normalize=False, one_hot_label=False)
    return x_test, t_test

def forward(w,x,now=1):
    y=np.dot(x,w)
    if now!=3:
        w=eval('W'+str(now+1))
        return forward(w,sigmoid(y),now+1)
    else: return softmax(y)

def init_network():
    with open('sample_weight.pkl','rb') as f:
        network=pickle.load(f)
    return network

def predict(network,x,now=1):
    W1,W2,W3=network['W1'],network['W2'],network['W3']
    b1,b2,b3=network['b1'],network['b2'],network['b3']
    w='W'+str(now)
    b=eval('b'+str(now))
    y=np.dot(x,network[w])+b
    if now!=3:
        return predict(network,sigmoid(y),now+1)
    else: return softmax(y)

import matplotlib.image as img
img=img.imread('C:\\python_data\\3.jpg')
img=img[:, :,2] #grayscale로 변환
img1=img.flatten()
network=init_network()
y=predict(network,img1)

print(y)
```

```
print(np.argmax(y,axis=0))
```

```
[4.4447958e-02 1.4861076e-04 4.7776058e-02 4.7793990e-01 4.9663293e-05  
8.5526943e-02 1.0935562e-04 3.3125555e-04 3.4241295e-01 1.2573689e-03]
```

3

문제82. 아래의 리스트를 numpy array리스트로 변환하고 shape를 확인하시오.

보기)

```
x = [[0.1,0.05,0.1,0.0,0.05,0.1,0.0,0.1,0.0,0.0],  
      [0.1,0.05,0.2,0.0,0.05,0.1,0.0,0.6,0.0,0.0],  
      [0.0,0.05,0.3,0.0,0.05,0.1,0.0,0.6,0.0,0.0],  
      [0.0,0.05,0.4,0.0,0.05,0.0,0.0,0.5,0.0,0.0],  
      [0.0,0.05,0.5,0.0,0.05,0.0,0.0,0.4,0.0,0.0],  
      [0.0,0.05,0.6,0.0,0.05,0.0,0.0,0.3,0.0,0.0],  
      [0.0,0.05,0.7,0.0,0.05,0.0,0.0,0.2,0.0,0.0],  
      [0.0,0.1,0.8,0.0,0.1,0.0,0.0,0.2,0.0,0.0],  
      [0.0,0.05,0.9,0.0,0.05,0.0,0.0,0.0,0.0,0.0],  
      [0.0,0.05,0.6,0.0,0.05,0.0,0.0,0.3,0.0,0.0] ]
```

답)

```
import numpy as np
```

```
import numpy as np
```

```
x = [[0.1,0.05,0.1,0.0,0.05,0.1,0.0,0.1,0.0,0.0],  
      [0.1,0.05,0.2,0.0,0.05,0.1,0.0,0.6,0.0,0.0],  
      [0.0,0.05,0.3,0.0,0.05,0.1,0.0,0.6,0.0,0.0],  
      [0.0,0.05,0.4,0.0,0.05,0.0,0.0,0.5,0.0,0.0],  
      [0.0,0.05,0.5,0.0,0.05,0.0,0.0,0.4,0.0,0.0],  
      [0.0,0.05,0.6,0.0,0.05,0.0,0.0,0.3,0.0,0.0],  
      [0.0,0.05,0.7,0.0,0.05,0.0,0.0,0.2,0.0,0.0],  
      [0.0,0.1,0.8,0.0,0.1,0.0,0.0,0.2,0.0,0.0],  
      [0.0,0.05,0.9,0.0,0.05,0.0,0.0,0.0,0.0,0.0],  
      [0.0,0.05,0.6,0.0,0.05,0.0,0.0,0.3,0.0,0.0]]  
x2=np.array(x)  
print(x2.shape)  
(10, 10)
```

문제83. 위에 10,10열 행렬에서 각 행에서의 가장 큰 원소가 몇번째 있는지 출력하시오.

답)

```
import numpy as np
```

```
x = [[0.1,0.05,0.1,0.0,0.05,0.1,0.0,0.1,0.0,0.0],  
      [0.1,0.05,0.2,0.0,0.05,0.1,0.0,0.6,0.0,0.0],  
      [0.0,0.05,0.3,0.0,0.05,0.1,0.0,0.6,0.0,0.0],  
      [0.0,0.05,0.4,0.0,0.05,0.0,0.0,0.5,0.0,0.0],  
      [0.0,0.05,0.5,0.0,0.05,0.0,0.0,0.4,0.0,0.0],  
      [0.0,0.05,0.6,0.0,0.05,0.0,0.0,0.3,0.0,0.0],  
      [0.0,0.05,0.7,0.0,0.05,0.0,0.0,0.2,0.0,0.0],  
      [0.0,0.1,0.8,0.0,0.1,0.0,0.0,0.2,0.0,0.0],  
      [0.0,0.05,0.9,0.0,0.05,0.0,0.0,0.0,0.0,0.0],  
      [0.0,0.05,0.6,0.0,0.05,0.0,0.0,0.3,0.0,0.0]]  
x2=np.array(x)  
print(x2.shape)
```

```

print(np.argmax(x2,axis=0)) #axis=0은 열중에서 가장 큰 것
print(np.argmax(x2,axis=1)) #axis=1은 행에서 가장 큰 것, 안쓰면 전체 행렬중에 가장 큰
것
(10, 10)
[0 7 8 0 7 0 0 1 0 0]
[0 7 7 7 2 2 2 2 2 2]

```

**문제84. (점심시간) 각 라인별로 A4지에 숫자를 하나 필기체로 적고 그 숫자를 3층 신경망에 넣었을때**

**컴퓨터가 숫자를 인식하는지 확인하시오.**

답)

```

import sys, os
sys.path.append(os.pardir)
import numpy as np
from dataset.mnist import load_mnist
import pickle
import numpy as np

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

def softmax(a):
    return np.exp(a-np.max(a))/np.sum(np.exp(a-np.max(a)))

def get_data():
    (x_train, t_train), (x_test, t_test) = \
        load_mnist(flatten=True, normalize=False, one_hot_label=False)
    return x_test, t_test

def forward(w,x,now=1):
    y=np.dot(x,w)
    if now!=3:
        w=eval('W'+str(now+1))
        return forward(w,sigmoid(y),now+1)
    else: return softmax(y)

def init_network():
    with open('sample_weight.pkl','rb') as f:
        network=pickle.load(f)
    return network

def predict(network,x,now=1):
    W1,W2,W3=network['W1'],network['W2'],network['W3']
    b1,b2,b3=network['b1'],network['b2'],network['b3']
    w='W'+str(now)
    b=eval('b'+str(now))
    y=np.dot(x,network[w])+b
    if now!=3:
        return predict(network,sigmoid(y),now+1)
    else: return softmax(y)

import matplotlib.image as img
img=img.imread('C:\python_data\4.jpg')

```

```

img=img[:, :, 2] #grayscale로 변환
img1=img.flatten()
network=init_network()
y=predict(network,img1)

print(y)
print(np.argmax(y,axis=0))

```

**문제85. 테스트 데이터 하나 x[34]의 필기체가 라벨이 무엇인지 확인하시오.**

답)

```

import sys, os
sys.path.append(os.pardir)
import numpy as np
from dataset.mnist import load_mnist
import pickle
import numpy as np

def get_data():
    (x_train, t_train), (x_test, t_test) = \
        load_mnist(flatten=True, normalize=False, one_hot_label=False)
    return x_test, t_test

x,t=get_data()
print(x.shape)
print(t[34])
(10000, 784)
7

```

**문제86. 테스트 데이터 하나인 x[34]의 필기체를 신경망에 넣고 신경망이 예측하는 것과 라벨이 서로 일치**

**하는지 확인하시오.**

구현결과)

예측 : 7, 실제 : 7

답)

```

import sys, os
sys.path.append(os.pardir)
import numpy as np
from dataset.mnist import load_mnist
import pickle
import numpy as np

def get_data():
    (x_train, t_train), (x_test, t_test) = \
        load_mnist(flatten=True, normalize=False, one_hot_label=False)
    return x_test, t_test

def predict(network,x,now=1):
    W1,W2,W3=network['W1'],network['W2'],network['W3']
    b1,b2,b3=network['b1'],network['b2'],network['b3']
    w='W'+str(now)

```

```

b=eval('b'+str(now))
y=np.dot(x,network[w])+b
if now!=3:
    return predict(network,sigmoid(y),now+1)
else: return softmax(y)

x,t=get_data()
print(x.shape)
print(t[34])
y=predict(network,x[34])
print(y)
p=np.argmax(y)
print("예측 : ",p, "실제 : ",t[34])

```

---

```

(10000, 784)
7
[8.3728571e-07 1.0406419e-05 1.4742499e-03 5.1291432e-04 1.0659765e-07
 1.8104543e-06 9.3316788e-10 9.9765539e-01 4.4168819e-06 3.3993935e-04]
예측 : 7 실제 : 7

```

**문제87. 테스트 데이터 10000장을 for loop문으로 전부 3층 신경망에 넣고 10000장중에 몇장을 맞추는지**

**확인하시오.**

답)

```

import sys, os
sys.path.append(os.pardir)
import numpy as np
from dataset.mnist import load_mnist
import pickle
import numpy as np

def get_data():
    (x_train, t_train), (x_test, t_test) = \
        load_mnist(flatten=True, normalize=True, one_hot_label=False)
    return x_test, t_test

def predict(network,x,now=1):
    W1,W2,W3=network['W1'],network['W2'],network['W3']
    b1,b2,b3=network['b1'],network['b2'],network['b3']
    w='W'+str(now)
    b=eval('b'+str(now))
    y=np.dot(x,network[w])+b
    if now!=3:
        return predict(network,sigmoid(y),now+1)
    else: return softmax(y)

cnt=0
x,t=get_data()
network=init_network()
for i in range(len(x)):
    y=predict(network,x[i])
    p=np.argmax(y)
    if p == t[i]:

```



```

        cnt+=1
    print(cnt)
9352

```

문제88. 9352개가 아니라 아래와 같이 정확도로 출력하시오,

```

답)
import sys, os
sys.path.append(os.pardir)
import numpy as np
from dataset.mnist import load_mnist
import pickle
import numpy as np

def get_data():
    (x_train, t_train), (x_test, t_test) = \
        load_mnist(flatten=True, normalize=True, one_hot_label=False)
    return x_test, t_test

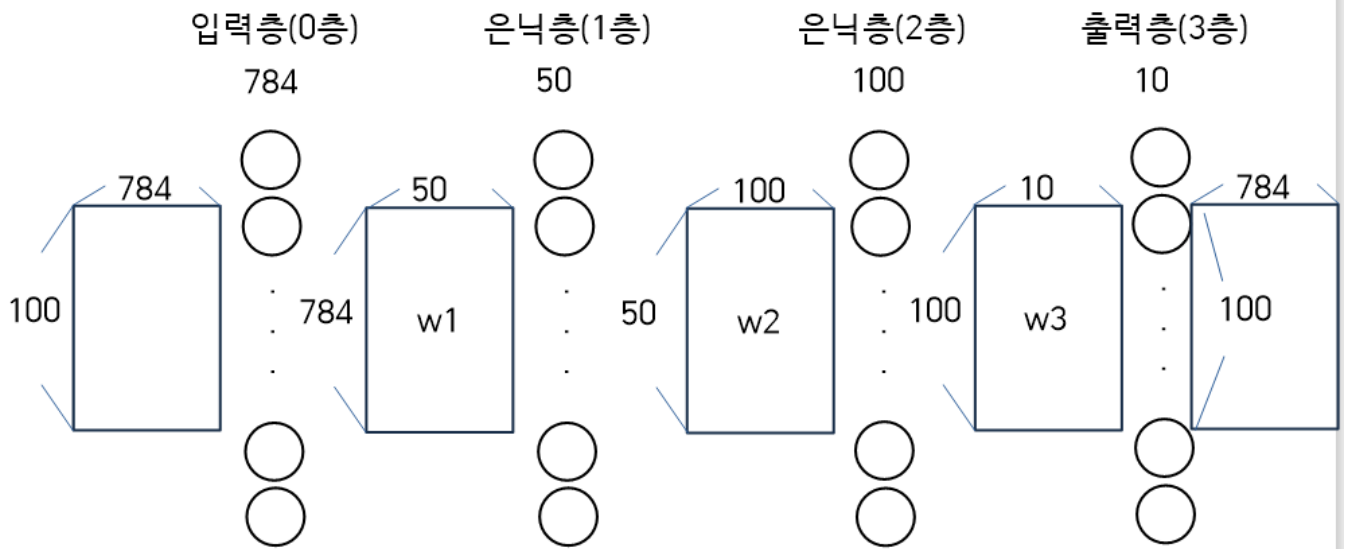
def predict(network, x, now=1):
    W1, W2, W3 = network['W1'], network['W2'], network['W3']
    b1, b2, b3 = network['b1'], network['b2'], network['b3']
    w = 'W' + str(now)
    b = eval('b' + str(now))
    y = np.dot(x, network[w]) + b
    if now != 3:
        return predict(network, sigmoid(y), now+1)
    else: return softmax(y)

cnt=0
x, t = get_data()
network = init_network()
for i in range(len(x)):
    y = predict(network, x[i])
    p = np.argmax(y)
    if p == t[i]:
        cnt+=1
print(len(x), '개 중에 ', cnt, '개를 맞췄음')
print('정확도=', round(cnt/len(x)*100, 2), '%')
10000 개 중에  9352 개를 맞췄음
정확도= 93.52 %

```

신경망에 데이터 입력시 배치(batch)로 입력하는 방법

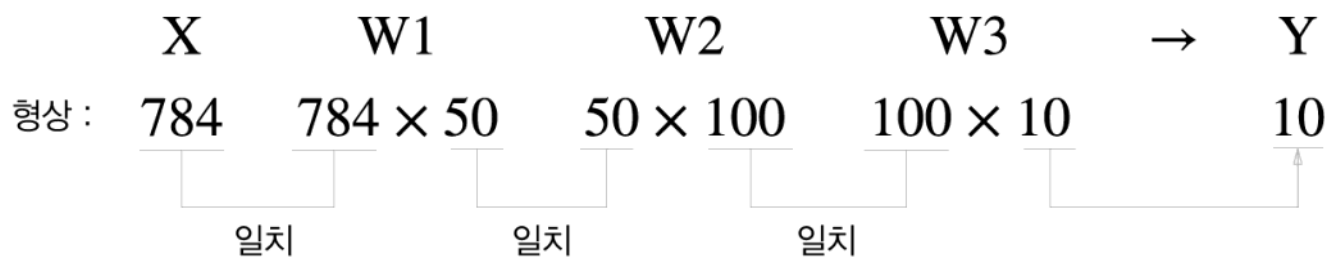
**3층 신경망 배치처리 그림**



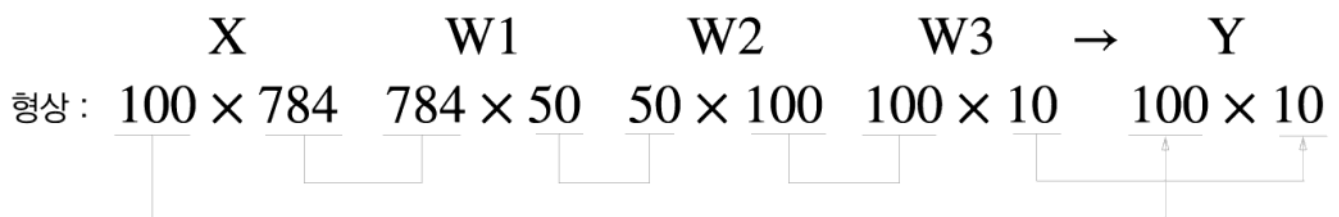
$$100 \times 784 \quad 784 \times 50 = 100 \times 50 \odot 50 \times 100 = 100 \times 100 \odot 100 \times 10 = 100 \times 1$$

" 이미지를 한장씩 처리하는게 아니라 여러장을 한번에 처리 "

이미지를 한장씩 처리한 신경망



이미지를 100장씩 처리한 신경망



문제89. 아래의 결과를 출력하시오.

보기)

[0,1,2,3,4,5,6,7,8,9]

답)

y=list(range(0,10))

print(y)

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

문제90.아래와 같이 결과를 출력하시오.

보기)

```
[0,3,6,9]
```

답)

```
y=list(range(0,10,3))
```

```
print(y)
```

```
[0, 3, 6, 9]
```

문제91. 아래의 리스트에서 최대값의 원소의 인덱스를 출력하시오.

보기)

```
[0,3,6,9]
```

답)

```
import numpy as np
```

```
y=list(range(0,10,3))
```

```
print(np.argmax(y))
```

```
3
```

문제92. 아래의 행렬을 numpy배열로 생성하시오.

보기)

```
0.1 0.8 0.1
```

```
0.3 0.1 0.6
```

```
0.2 0.5 0.3
```

```
0.8 0.1 0.1
```

답)

```
import numpy as np
```

```
a=np.array([[0.1,0.8,0.1],[0.3,0.1,0.6],[0.2,0.5,0.3],[0.8,0.1,0.1]])
```

```
print(a)
```

```
[[0.1 0.8 0.1]
```

```
 [0.3 0.1 0.6]
```

```
 [0.2 0.5 0.3]
```

```
 [0.8 0.1 0.1]]
```

문제93. numpy의 argmax를 이용해서 아래의 행렬에서 행중에 최대값 원소의 인덱스를 출력하시오,

답)

```
import numpy as np
```

```
a=np.array([[0.1,0.8,0.1],[0.3,0.1,0.6],[0.2,0.5,0.3],[0.8,0.1,0.1]])
```

```
print(np.argmax(a, axis=1))
```

```
[1 2 1 0]
```

문제94. 아래의 2개의 리스트를 만들고 서로 같은 자리에 같은 숫자가 몇 개가 있는지 출력하시오.

보기)

```
[2,1,3,5,1,4,2,1,1,0]
```

```
[2,1,3,4,5,4,2,1,1,2]
```

결과)

7

답1)

```
a=[2,1,3,5,1,4,2,1,1,0]
```

```
b=[2,1,3,4,5,4,2,1,1,2]
```

```
cnt=0
```

```
c=len(a)
```

```
for i in range(c):
```

```
    if a[i]==b[i]:
```

```
        cnt+=1
```

```
print(cnt)
```

답2) numpy이용

```
a=[2,1,3,5,1,4,2,1,1,0]
```

```
b=[2,1,3,4,5,4,2,1,1,2]
```

```
x=np.array(a)
```

```
y=np.array(b)
```

```
print(x==y)
```

```
print(np.sum(x==y))
```

```
[ True  True  True False False  True  True  True  True False]
```

7

문제95. 아래의 리스트를 x라는 변수에 담고 앞에 5개의 숫자만 출력하시오.

보기)

```
[7,3,2,1,6,7,7,8,2,4]
```

결과)

```
[7,3,2,1,6]
```

답)

```
x=[7,3,2,1,6,7,7,8,2,4]
```

```
print(x[0:5])
```

```
[7, 3, 2, 1, 6]
```

문제96. 0~100, 100~200, 200~300, ... 9900~10000에서 100,200,300 ... 10000을 출력하시오.

답)

```
for i in range(0,10001,100):
```

```
    print(i)
```

```
0
100
200
300
400
500
600
700
800
900
1000
1100
1200
1300
1400
1500
1600
```

**문제97. mnist의 훈련 데이터를 100개씩 가져오는 코드를 작성하시오.**

```
보기)
import sys, os
sys.path.append(os.pardir)
import numpy as np
from dataset.mnist import load_mnist
import pickle
import numpy as np

def get_data():
    (x_train, t_train), (x_test, t_test) = \
        load_mnist(flatten=True, normalize=True, one_hot_label=False)
    return x_test, t_test

x, t = get_data()
batch_size=100
for i in range(0,10000,100):
    x_batch=x[i:i+batch_size]
    print(x_batch)
[[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]
[[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
 [0 0 0      0 0 0 1
```

**해결될때는 아래코드를 보자**

```
for i in range(0,10000,100):
    #x_batch=x[i:i+batch_size]
    x_batch=(i,i+batch_size)
```

```

print(x_batch)
(0, 100)
(100, 200)
(200, 300)
(300, 400)
(400, 500)

(8000, 8000),
(9600, 9700)
(9700, 9800)
(9800, 9900)
(9900, 10000)

```

**문제98.** 100개 가지고 온 훈련데이터를 predict함수에 입력해서 예측 숫자 100개를 출력하는 코드를 작성

하시오.

답)

```

import sys, os
sys.path.append(os.pardir)
import numpy as np
from dataset.mnist import load_mnist
import pickle
import numpy as np

def get_data():
    (x_train, t_train), (x_test, t_test) = \
        load_mnist(flatten=True, normalize=True, one_hot_label=False)
    return x_test, t_test

def predict(network, x, now=1):
    W1, W2, W3 = network['W1'], network['W2'], network['W3']
    b1, b2, b3 = network['b1'], network['b2'], network['b3']
    w = 'W' + str(now)
    b = eval('b' + str(now))
    y = np.dot(x, network[w]) + b
    if now != 3:
        return predict(network, sigmoid(y), now+1)
    else: return softmax(y)

x, t = get_data()
batch_size = 100
for i in range(0, 10000, 100):
    x_batch = x[i:i+batch_size]
    y_batch = predict(network, x_batch)
    p = np.argmax(y_batch, axis=1)
    print(p)

```

```
[7 2 1 0 4 1 4 9 6 9 0 6 9 0 1 5 9 7 3 4 9 6 6 5 4 0 7 4 0 1 3 1 3 6 7 2 7
 1 2 1 1 7 4 2 3 5 1 2 4 4 6 3 5 5 6 0 4 1 9 5 7 8 9 3 7 4 2 4 3 0 7 0 2 9
 1 7 3 2 9 7 7 6 2 7 8 4 7 3 6 1 3 6 4 3 1 4 1 7 6 9]
[6 0 5 4 9 9 2 1 9 4 8 7 3 9 7 4 4 4 9 2 5 4 7 6 4 9 0 5 8 5 6 6 5 7 8 1 0
 1 6 4 6 7 3 1 7 1 8 2 0 9 9 9 5 5 1 5 6 0 3 4 4 6 5 4 6 5 4 5 1 4 4 7 2 3
 2 7 1 8 1 8 1 8 5 0 8 9 2 5 0 1 1 1 0 9 0 3 1 6 4 2]
[3 6 1 1 1 3 9 5 2 9 4 5 9 3 9 0 3 5 5 5 7 2 2 7 1 2 8 4 1 7 3 3 8 7 7 9 2
 2 4 1 5 8 8 7 2 5 0 2 4 2 4 1 9 5 7 7 2 8 2 0 8 5 7 7 9 1 8 1 8 0 3 0 1 9
 9 4 1 8 2 1 2 9 7 5 9 2 6 4 1 5 4 2 9 2 0 4 0 0 2 8]
[6 7 1 2 4 0 2 7 4 3 3 0 0 5 1 9 6 5 2 5 7 7 9 3 0 4 2 0 7 1 1 2 1 5 3 3 9
 7 8 6 3 4 1 3 8 1 0 5 1 3 1 5 0 6 1 8 5 1 9 9 4 6 7 2 5 0 6 5 6 3 7 2 0 8
```

문제99. 위의 코드를 수정해서 예측 숫자100개와 실제 숫자 100개를 출력하도록 파이썬 코드를 작성.

```
답)
import sys, os
sys.path.append(os.pardir)
import numpy as np
from dataset.mnist import load_mnist
import pickle
import numpy as np

def get_data():
    (x_train, t_train), (x_test, t_test) = \
        load_mnist(flatten=True, normalize=True, one_hot_label=False)
    return x_test, t_test

def predict(network, x, now=1):
    W1, W2, W3 = network['W1'], network['W2'], network['W3']
    b1, b2, b3 = network['b1'], network['b2'], network['b3']
    w = 'W' + str(now)
    b = eval('b' + str(now))
    y = np.dot(x, network[w]) + b
    if now != 3:
        return predict(network, sigmoid(y), now+1)
    else: return softmax(y)

x, t = get_data()
batch_size = 100
for i in range(0, 10000, 100):
    x_batch = x[i:i+batch_size]
    y_batch = predict(network, x_batch)
    p = np.argmax(y_batch, axis=1)
    t_batch = t[i:i+batch_size]
    print(t_batch)
```

```
[7 2 1 0 4 1 4 9 5 9 0 6 9 0 1 5 9 7 3 4 9 6 6 5 4 0 7 4 0 1 3 1 3 4 7 2 7
 1 2 1 1 7 4 2 3 5 1 2 4 4 6 3 5 5 6 0 4 1 9 5 7 8 9 3 7 4 6 4 3 0 7 0 2 9
 1 7 3 2 9 7 7 6 2 7 8 4 7 3 6 1 3 6 9 3 1 4 1 7 6 9]
[6 0 5 4 9 9 2 1 9 4 8 7 3 9 7 4 4 4 9 2 5 4 7 6 7 9 0 5 8 5 6 6 5 7 8 1 0
 1 6 4 6 7 3 1 7 1 8 2 0 2 9 9 5 5 1 5 6 0 3 4 4 6 5 4 6 5 4 5 1 4 4 7 2 3
 2 7 1 8 1 8 1 8 5 0 8 9 2 5 0 1 1 1 0 9 0 3 1 6 4 2]
[3 6 1 1 1 3 9 5 2 9 4 5 9 3 9 0 3 6 5 5 7 2 2 7 1 2 8 4 1 7 3 3 8 8 7 9 2
 2 4 1 5 9 8 7 2 3 0 4 4 2 4 1 9 5 7 7 2 8 2 6 8 5 7 7 9 1 8 1 8 0 3 0 1 9
 9 4 1 8 2 1 2 9 7 5 9 2 6 4 1 5 8 2 9 2 0 4 0 0 2 8]
[4 7 1 2 4 0 2 7 4 3 3 0 0 3 1 9 6 5 2 5 9 2 9 3 0 4 2 0 7 1 1 2 1 5 3 3 9
 7 8 6 5 6 1 3 8 1 0 5 1 3 1 5 5 6 1 8 5 1 7 9 4 6 2 2 5 0 6 5 6 3 7 2 0 8
 8 5 4 1 1 4 0 3 3 7 6 1 6 2 1 9 2 8 6 1 9 5 2 5 4 4]
```

문제100. p,t리스트에서 얼마나 일치하는지 숫자를 count해서 아래와 같은 결과를 출력하시오.

결과)

9352

답)

```
import sys, os
sys.path.append(os.pardir)
import numpy as np
from dataset.mnist import load_mnist
import pickle
import numpy as np

def get_data():
    (x_train, t_train), (x_test, t_test) = \
        load_mnist(flatten=True, normalize=True, one_hot_label=False)
    return x_test, t_test

def predict(network, x, now=1):
    W1, W2, W3 = network['W1'], network['W2'], network['W3']
    b1, b2, b3 = network['b1'], network['b2'], network['b3']
    w = 'W' + str(now)
    b = eval('b' + str(now))
    y = np.dot(x, network[w]) + b
    if now != 3:
        return predict(network, sigmoid(y), now+1)
    else: return softmax(y)

cnt=0
x, t = get_data()
batch_size=100
for i in range(0, 10000, 100):
    x_batch = x[i:i+batch_size]
    y_batch = predict(network, x_batch)
    p = np.argmax(y_batch, axis=1)
    t_batch = t[i:i+batch_size]
    #print(t_batch)
    cnt += np.sum(p == t_batch)
    #print(np.sum(p == t_batch))
print(cnt)
9352
```





## 4장 신경망 학습

2018년 8월 23일 목요일 오전 11:11

### 신경망 학습

"신경망의 가중치와 bias를 학습시키는 방법을 배우는 챕터"

### 평균 제곱 오차

$$\frac{1}{n} \sum_{i=1}^n (y_i - t_i)^2$$

문제101. 평균제곱 오차 함수를 책 112페이지 식을보고 `mean_squared_error` 라는 함수 이름으로 만들고

아래와 같이 구현하시오.

보기)

```
y=[0.1,0.05,0.6,0.0,0.05,0.1,0.0,0.1,0.0,0.0] #예측
```

```
t=[0,0,1,0,0,0,0,0,0,0] #실제
```

답)

```
import numpy as np
```

```
y=[0.1,0.05,0.6,0.0,0.05,0.1,0.0,0.1,0.0,0.0] #예측
```

```
t=[0,0,1,0,0,0,0,0,0,0] #실제
```

```
n=2
```

```
def mean_squared_error(y,t):  
    return 1/n*np.sum((y-t)**2)
```

```
print(mean_squared_error(np.array(y),np.array(t)))
```

```
0.09750000000000003
```

문제102. 아래의 확률벡터를 평균제곱오차 함수를 이용해서 `target(실제값)`과 `예측값`의 오차율이 어떻게

되는지 `for loop`문으로 한번에 알아내시오.

보기)

```
t = [0,0,1,0,0,0,0,0,0,0] # 숫자2
```

```
y1 = [0.1,0.05,0.1,0.0,0.05,0.1,0.0,0.1,0.0,0.0]
```

```
y2 = [0.1,0.05,0.2,0.0,0.05,0.1,0.0,0.6,0.0,0.0]
```

```

y3 = [0.0,0.05,0.3,0.0,0.05,0.1,0.0,0.6,0.0,0.0]
y4 = [0.0,0.05,0.4,0.0,0.05,0.0,0.0,0.5,0.0,0.0]
y5 = [0.0,0.05,0.5,0.0,0.05,0.0,0.0,0.4,0.0,0.0]
y6 = [0.0,0.05,0.6,0.0,0.05,0.0,0.0,0.3,0.0,0.0]
y7 = [0.0,0.05,0.7,0.0,0.05,0.0,0.0,0.2,0.0,0.0]
y8 = [0.0,0.1,0.8,0.0,0.1,0.0,0.0,0.2,0.0,0.0]
y9 = [0.0,0.05,0.9,0.0,0.05,0.0,0.0,0.0,0.0,0.0]

```

결과)

y1 의 오차율 : 32%

y2 의 오차율 : 43%

:

답)

```
import numpy as np
```

```
t=[0,0,1,0,0,0,0,0,0,0] #실제
```

```
y1 = [0.1,0.05,0.1,0.0,0.05,0.1,0.0,0.1,0.0,0.0]
```

```
y2 = [0.1,0.05,0.2,0.0,0.05,0.1,0.0,0.6,0.0,0.0]
```

```
y3 = [0.0,0.05,0.3,0.0,0.05,0.1,0.0,0.6,0.0,0.0]
```

```
y4 = [0.0,0.05,0.4,0.0,0.05,0.0,0.0,0.5,0.0,0.0]
```

```
y5 = [0.0,0.05,0.5,0.0,0.05,0.0,0.0,0.4,0.0,0.0]
```

```
y6 = [0.0,0.05,0.6,0.0,0.05,0.0,0.0,0.3,0.0,0.0]
```

```
y7 = [0.0,0.05,0.7,0.0,0.05,0.0,0.0,0.2,0.0,0.0]
```

```
y8 = [0.0,0.1,0.8,0.0,0.1,0.0,0.0,0.2,0.0,0.0]
```

```
y9 = [0.0,0.05,0.9,0.0,0.05,0.0,0.0,0.0,0.0,0.0]
```

```
def mean_squared_error(y,t):
    return 1/n*np.sum((y-t)**2)
```

```
for i in range(1,10):
```

```
    y=eval('y'+str(i))
```

```
    print('y', i, '의 오차율 : ', round(mean_squared_error(np.array(y),np.array(t))*100,4),'%')
```

```
y 1 의 오차율 : 42.25 %
```

```
y 2 의 오차율 : 51.25 %
```

```
y 3 의 오차율 : 43.25 %
```

```
y 4 의 오차율 : 30.75 %
```

```
y 5 의 오차율 : 20.75 %
```

```
y 6 의 오차율 : 12.75 %
```

```
y 7 의 오차율 : 6.75 %
```

```
y 8 의 오차율 : 5.0 %
```

```
y 9 의 오차율 : 0.75 %
```

## 교차 엔트로피 비용 함수

문제103. 밑이 자연상수이고 진수가 0.6인 로그값을 출력하시오.

답)

```
import numpy as np
```

```
print(np.log(0.6))
```

-0.5108256237659907

문제104. 밑수가 10이고 진수가 0.6인 로그의 로그값을 출력하시오.

```
답)
import numpy as np

print(np.log10(0.6))
-0.2218487496163564
```

문제105. mean\_square error 함수의 공식을 가지고 0.1과 0.9 확률의 오차를 각각 출력하시오.

```
보기)
확률 : 0.1    확률 : 0.9
오차 : ?      오차 : ?
t=[0,0,1,0,0,0,0,0,0,0] #실제
y=[0,0,0.1,0,0,0,0,0,0.9] #예측
```

```
답)
x = np.array([0, 0, 0.1, 0, 0, 0, 0, 0, 0, 0.9])
y = np.array([0, 0, 0.9, 0, 0, 0, 0, 0, 0, 0.1])
t = np.array([0, 0, 1, 0, 0, 0, 0, 0, 0, 0])

def mean_squared_error(x, t):
    import numpy as np
    return np.sum((x-t)**2)/2

print(mean_squared_error(x, t)) #확률 0.1
print(mean_squared_error(y, t)) #확률 0.9
0.81
0.009999999999999998
```

교차 엔트로피 공식

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

*target*  
*확률벡터에 로그*

MSE 함수와 비교

```
print(mean_squared_error(x, t)) #확률 0.1
print(mean_squared_error(y, t)) #확률 0.9
```

```
0.81
0.009999999999999998
```

```
: print(-np.log(0.1), -np.log(0.9))
```

```
2.3025850929940455 0.10536051565782628
```

설명 : MSE보다 교차 엔트로피가 더큰 오차를 낸다.

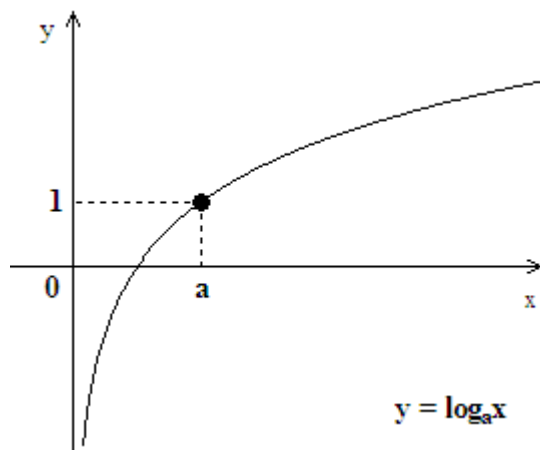
문제106. 밑수가 자연상수이고 진수가 0인 로그의 로그값은 얼마인지 출력하시오.

답)

```
import numpy as np
print(np.log(0))
```

```
-inf
```

-무한대가 나오는데 그래프를 보면 이해가 간다.



수치연산이 불가능해 진다.

문제107. cross entropy error 함수를 아래의 공식을 보고 생성하시오.

보기)

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

```
y = np.array([0, 0, 0.9, 0, 0, 0, 0, 0, 0, 0.1])
t = np.array([0, 0, 1, 0, 0, 0, 0, 0, 0, 0])
print(CEE(y,t))
```

답)

```
y = np.array([0, 0, 0.9, 0, 0, 0, 0, 0, 0, 0.1])
t = np.array([0, 0, 1, 0, 0, 0, 0, 0, 0, 0])
def CEE(y,t):
    return -np.sum(t*np.log(y))
```

```
print(CEE(y,t))
```

```
nan
```

$\log(0)$ 이 -무한대여서 계산이 안되서 아주 작은 숫자를 더해줘야한다.

그래서 아주작은( $1e-7$ ) 값을 더해서 절대 0이 되지 않도록 해준다.

```
y = np.array([0, 0, 0.9, 0, 0, 0, 0, 0, 0, 0.1])
```

```
t = np.array([0, 0, 1, 0, 0, 0, 0, 0, 0, 0])
```

```
def CEE(y,t):
```

```
    delta=1e-7
```

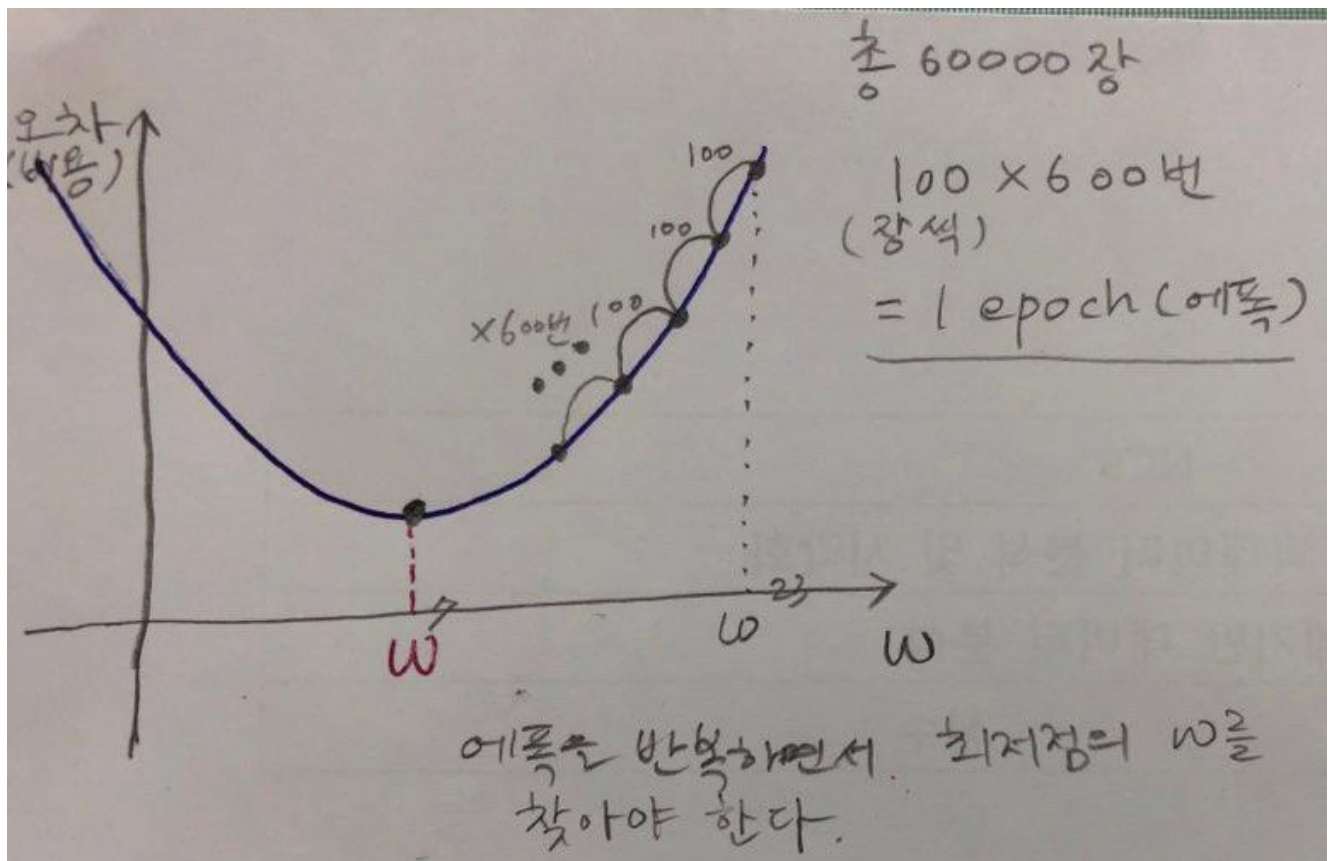
```
    return -np.sum(t*np.log(y+delta))
```

```
print(CEE(y,t))
```

```
0.1053604045467214
```

## 미니 배치 학습

학습속도를 높이기 위해서 사용한다.



문제108. 60000 숫자 중에 무작위로 10개를 출력하시오.

답)

```
import numpy as np
```

```
print(np.random.choice(np.arange(60000), 10))
```

```
[ 8082 44873 9567 5173 46878 20668 22293 19969 8162 35237]
```

문제109. mnist의 테스트 데이터 10000장중에 랜덤으로 100장을 추출하는 코드를 작성하시오.

(복원추출되게 코드를 수정하시오)

답)

```
from dataset.mnist import load_mnist
import pickle
import numpy as np
```

# 시그모이드 함수 생성

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

# 소프트맥스 함수 생성

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

# 테스트 데이터 가져오는 함수

```
def get_data():
    (x_train, t_train), (x_test, t_test) = load_mnist(normalize=True, flatten=True,
one_hot_label=False)
    return x_test, t_test
```

# 가중치와 bias값을 가져오는 함수

```
def init_network():
    with open("sample_weight.pkl", 'rb') as f:
        network = pickle.load(f)
    return network
```

# 숫자를 분류하는 신경망 함수

```
def predict(network, x):
    W1, W2, W3 = network['W1'], network['W2'], network['W3']
    b1, b2, b3 = network['b1'], network['b2'], network['b3']

    y1 = np.dot(x, W1) + b1
    y1_hat = sigmoid(y1)
    y2 = np.dot(y1_hat, W2) + b2
    y2_hat = sigmoid(y2)
    y3 = np.dot(y2_hat, W3) + b3
    y3_hat = softmax(y3)

    return y3_hat
```

# 실행코드

```
network=init_network()
x, t = get_data()
batch = 100
cnt = 0
```

```
for i in range(0, 10000, 100):
    batch_mask=np.random.choice(10000,100)
    x_batch = x[batch_mask]#복원 추출코드
    t_batch = t[batch_mask]#복원 추출코드
```

```

#x_batch = x[i:i+batch]#비복원 추출코드
#t_batch = t[i:i+batch]#비복원 추출코드
y = predict(network,x_batch)
z = np.argmax(y, axis=1)
cnt += (sum(z==t_batch))
print(len(x), '개 중에서 일치하는 수 :', cnt)

```

---

10000 개 중에서 일치하는 수 : 9344

---

10000 개 중에서 일치하는 수 : 9334

**돌릴때마다 결과가 다르게 나온다.**

**문제110. 위의 코드를 5 에폭 돌게 코드를 수정하시오.**

```

답)
from dataset.mnist import load_mnist
import pickle
import numpy as np

# 시그모이드 함수 생성
def sigmoid(x):
    return 1/(1+np.exp(-x))

# 소프트맥스 함수 생성
def softmax(x):
    c = np.max(x)
    return np.exp(x-c)/np.sum(np.exp(x-c))

# 테스트 데이터 가져오는 함수
def get_data():
    (x_train, t_train), (x_test, t_test) = load_mnist(normalize=True, flatten=True,
one_hot_label=False)
    return x_test, t_test

# 가중치와 bias값을 가져오는 함수
def init_network():
    with open("sample_weight.pkl", 'rb') as f:
        network = pickle.load(f)
    return network

# 숫자를 분류하는 신경망 함수
def predict(network, x):
    W1, W2, W3 = network['W1'], network['W2'], network['W3']
    b1, b2, b3 = network['b1'], network['b2'], network['b3']

    y1 = np.dot(x, W1) + b1
    y1_hat = sigmoid(y1)
    y2 = np.dot(y1_hat, W2) + b2
    y2_hat = sigmoid(y2)
    y3 = np.dot(y2_hat, W3) + b3
    y3_hat = softmax(y3)

```



```

return y3_hat

# 실행코드
network=init_network()
x, t = get_data()
batch = 100
cnt = 0
epoch=5
for j in range(epoch):
    for i in range(0, 10000, 100):
        batch_mask=np.random.choice(10000,100)
        x_batch = x[batch_mask]#복원 추출코드
        t_batch = t[batch_mask]#복원 추출코드

        #x_batch = x[i:i+batch]#비복원 추출코드
        #t_batch = t[i:i+batch]#비복원 추출코드
        y = predict(network,x_batch)
        z = np.argmax(y, axis=1)
        cnt += (sum(z==t_batch))
    print(len(x), '개 중에서 일치하는 수 :', cnt)
    print(print('정확도 : ',cnt/100,'%'))
    cnt=0

```

```

10000 개 중에서 일치하는 수 : 9377
정확도 : 93.77 %
None
10000 개 중에서 일치하는 수 : 9336
정확도 : 93.36 %
None
10000 개 중에서 일치하는 수 : 9350
정확도 : 93.5 %
None
10000 개 중에서 일치하는 수 : 9350
정확도 : 93.5 %
None
10000 개 중에서 일치하는 수 : 9334
정확도 : 93.34 %
None

```

## 미니배치 처리에 맞도록 교차 엔트로피 함수를 구성하는 방법

### - 미니배치 처리하기전 교차 엔트로피 함수

```

def CEE(y,t):
    delta=1e-7
    return -np.sum(t*np.log(y+delta))

```

### - 100장씩 미니배치한후의 교차 엔트로피 함수

```

y = np.array([[0, 0, 0.9, 0, 0, 0, 0, 0, 0, 0.1],
              [0.9, 0, 0.1, 0, 0, 0, 0, 0, 0, 0.1],
              .
              . #100개
              [0, 0, 0.9, 0, 0, 0, 0, 0, 0, 0.1]])

```

```
def CEE_MB(y,t):
    delta=1e-7
    return -np.sum(t*np.log(y+delta))/y.shape[0]
```

### 확률적 경사 하강법(SGD Stochastic Gradient Decent)

" 100장을 랜덤으로 무작위로 선정(수집)해서 신경망 학습을 시키는 경사 하강법 "

### 최소 비용의 가중치를 알아내기 위해 학습을 시키는가? (미분)

" 미분을 사용한다 "

### 수치미분

" 진정한 미분은 컴퓨터로 구현하기 어렵기 때문에 중앙 차분 오차가 발생하지만 컴퓨터로 미분을 구현하기 위해서는 수치미분을 사용해야 한다. "

$$f'(x) = \lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h}$$

미분 공식을 파이썬 함수로 구현해서 실행을 해본다.

답)

```
def numerical_diff(f,x):
    h=10e-50
    return (f(x+h)-f(x)) / h
numerical_diff()
```

위의 함수에 문제점은 h가 컴퓨터로 계산시 소수점 8자리 이하는 생략이 되어서 오류가 발행한다.

```
import numpy as np
print(np.float32(1e-50)) #0.0
```

```
def numerical_diff(f,x):
    h=1e-4 # 0.0001, 컴퓨터로는 극한값을 구하기 어려우므로
    return (f(x+h)-f(x)) / h
#컴퓨터로는 미분을 하지 못하니 도함수를 구현해야 한다.
```

### 접선공식

### 할선 공식

$$(\lim)_{h \rightarrow 0} \frac{f(x+h)-f(x)}{h} \longrightarrow \lim_{h \rightarrow 0} \frac{f(x+h)-f(x-h)}{(x+h)-(x-h)}$$

↓

$$\lim_{h \rightarrow 0} \frac{f(x+h) - f(x-h)}{2h}$$

함수를 미분하는 수치미분 함수

```
def numerical_diff(f,x):
    h=1e-4
    return (f(x+h)-f(x-h)) / (2*h)
```

**문제111. 아래의 비용함수를 수치미분 함수로 미분을 하는데  $x=4$  에서의 미분계수(기울기)를 구하시오.**

보기)

$y=x^2+4^2$  (비용함수)

답)

```
import numpy as np
print(np.float32(1e-50)) #0.0
```

```
def numerical_diff(f,x):
    h=1e-4
    return (f(x+h)-f(x-h)) / (2*h)
```

```
def loss2(x):
    return (x**2 + 4**2)
```

```
print(numerical_diff(loss2,4))
7.999999999999999
```

**문제112. 문제111번의 비용함수를 아래와 같이 시각화 하시오.**

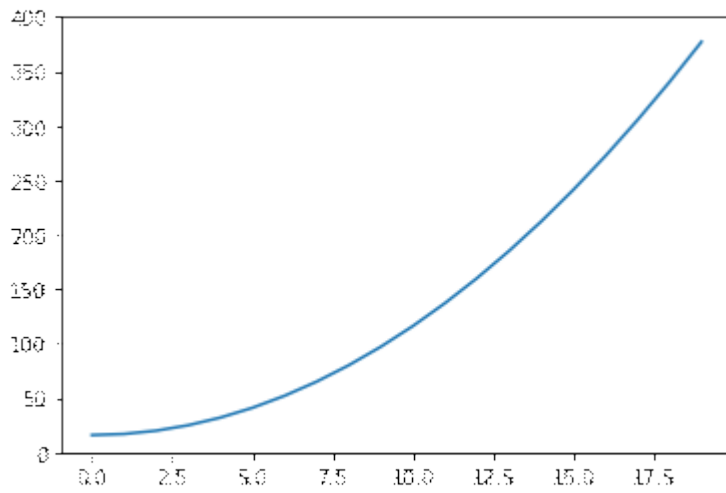
보기)

```
def loss2(x):
    return (x**2 + 4**2)
```

답)

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

```
def loss2(x):
    return (x**2 + 4**2)
x=np.arange(0,20,1)
y=loss2(x)
plt.ylim(0,400)
plt.plot(x,y)
plt.show()
```



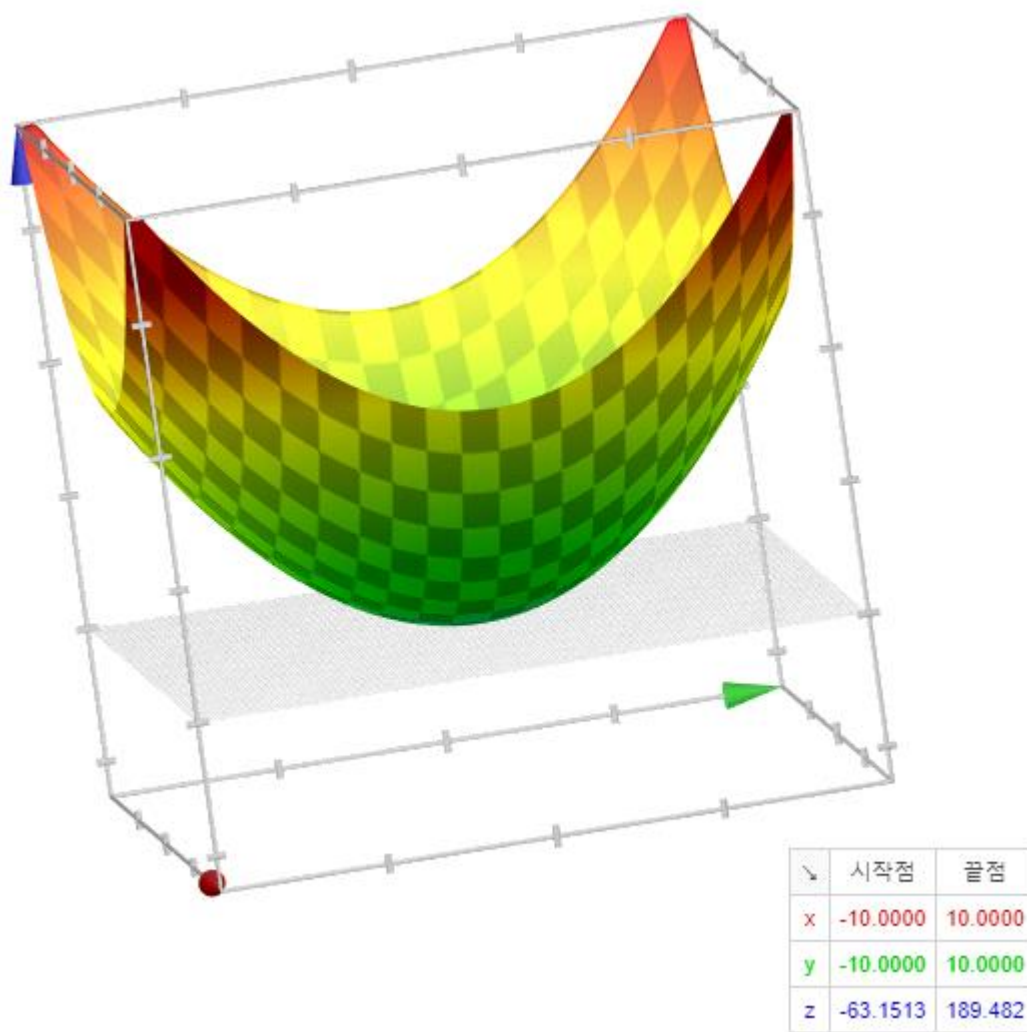
문제113. 아래의 식을 구글에서 시각화 하시오.

보기)

$$f(x_0, x_1) = x_0^2 + x_1^2$$

답)

구글에서  $z=x^2+y^2$  을 검색



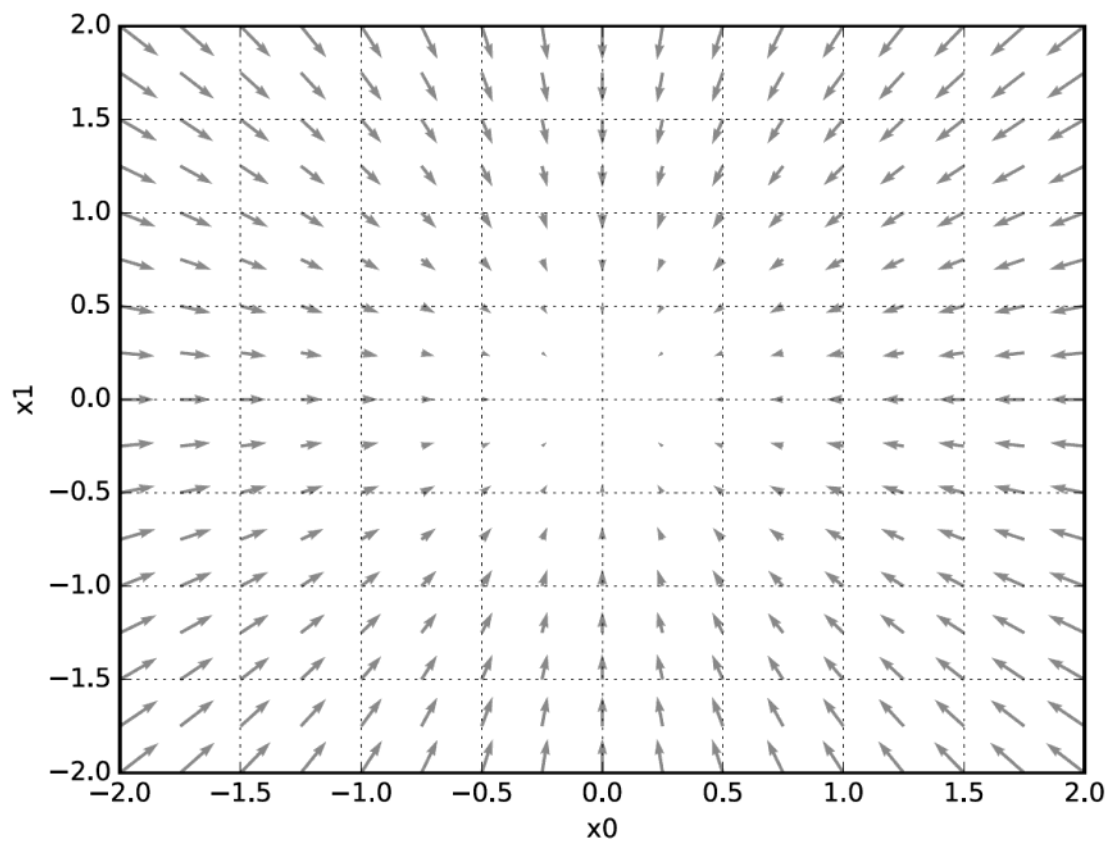
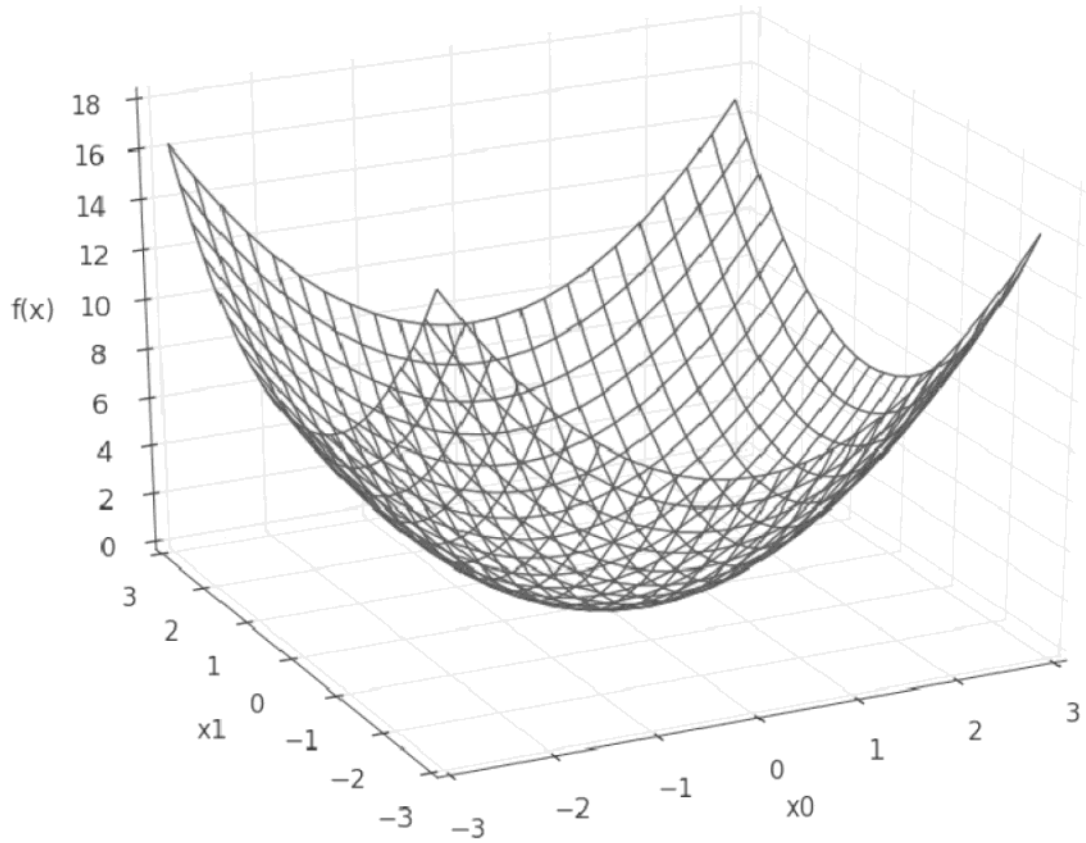
## 편미분

" 변수가 2개 이상인 함수를 미분할 때 미분 대상 변수 외에 나머지 변수를 상수 처럼

## 고정시켜 미분

하는 것 "

예 :  $f(x_0, x_1) = x_0^2 + x_1^2$  을  $f(w_0, w_1) = w_0^2 + w_1^2$ 으로 생각하자.



문제114.  $f(x_0, x_1) = x_0^2 + x_1^2$  함수를 편미분하는데  $x_0=3$ ,  $x_1=4$ 일때  $x_0$ 에 대해서 편미분하시오.

```
답)
import numpy as np

def numerical_diff(f,x):
    h=1e-4
    return (f(x+h)-f(x-h)) / (2*h)

def sample_f1(x):
    return x**2 + 4**2

print(numerical_diff(sample_f1,3))
6.0000000000000378
```

문제115.  $x_1$ 에 대해서 편미분 하시오.

```
답)
import numpy as np

def numerical_diff(f,x):
    h=1e-4
    return (f(x+h)-f(x-h)) / (2*h)

def sample_f1(x):
    return x**2 + 4**2

def sample_f2(x):
    return 3**2 + x**2

print(numerical_diff(sample_f2,4))
7.999999999999919
```

이렇게하면 w값들이 여러 개인데 하나 하나씩 구하기가 어렵다. 그러므로 한번에 알아내는 함수를 만들자.

문제116. 아래의 numpy 배열과 같은 구조의 행렬이 출력되는데 값은 0으로 출력하시오.

```
보기)
x=np.array([3.0, 4.0])

결과)
[0, 0]

답)
import numpy as np
x=np.array([3.0, 4.0])
print(np.zeros_like(x))
[0.  0.]
```

문제117. 어...위에서는  $f(x_0, x_1) = x_0^2 + x_1^2$  함수를 편미분하는 것을 각각 수행했는데 이번에는 편미분이

한번에 수행되도록 하는 코드를 작성하시오.

답)

```
import numpy as np
```

```
def numerical_diff(f,x):  
    h=1e-4 #0.00001  
    return (f(x+h)-f(x-h)) / (2*h)
```

```
def loss_func(x):  
    return x[0]**2 + x[1]**2
```

```
def numerical_gradient(f,x):  
    h=1e-4 #0.00001  
    grad=np.zeros_like(x) #x와 형상이 같은 배열을 생성
```

```
    for idx in range(x.size): #0, 1  
        tmp_val = x[idx] #x[0]
```

```
        #f(x+h) 계산
```

```
        x[idx]=tmp_val + h #3.00001
```

```
        fxh1=f(x) #3.000101^2 + 4^2 = 25.00060001
```

```
        #f(x-h) 계산
```

```
        x[idx]=tmp_val - h #3 - 0.0001 = 2.9999^2 + 4^2 = 24.99940001
```

```
        fxh2=f(x)
```

```
        grad[idx] = (fxh1-fxh2) / (2*h)
```

```
        x[idx] = tmp_val
```

```
    return grad
```

```
print(numerical_gradient(loss_func, np.array([3.0, 4.0])))
```

```
[6. 8.]
```

문제118. 아래의 x0,x1 지점에서의 기울기를 각각 구하시오.

보기)

[3.0, 4.0] --> [6.0, 8.0]

[0.0, 2.0] --> ?

[3.0, 0.0] --> ?

답)

```
print(numerical_gradient(loss_func, np.array([0.0, 2.0])))
```

```
print(numerical_gradient(loss_func, np.array([3.0, 0.0])))
```

```
[0. 4.]
```

```
[6. 0.]
```

문제119. 위에서 만든 numerical\_gradient 함수를 100번 수행해서 global minima에 도달하도록 하는

코드를 구현하시오.

답)

```
def loss_func(x):
    return x[0]**2 + x[1]**2

def numerical_gradient(f,x):
    h=1e-4 #0.00001
    grad=np.zeros_like(x) #x와 형상이 같은 배열을 생성

    for idx in range(x.size): #0, 1
        tmp_val = x[idx] #x[0]

        #f(x+h) 계산
        x[idx]=tmp_val + h #3.00001
        fxh1=f(x) #3.00010|^2 + 4^2 = 25.00060001

        #f(x-h) 계산
        x[idx]=tmp_val - h #3 - 0.0001 = 2.9999|^2 + 4^2 = 24.99940001
        fxh2=f(x)

        grad[idx] = (fxh1-fxh2) / (2*h)
        x[idx] = tmp_val
    return grad

init_x = np.array([3.0, 4.0])

def gradient_descent(f, init_x, lr=0.01, step_num=100):
    x=init_x

    for i in range(step_num):
        grad = numerical_gradient(f,x)
        x -= lr * grad
    return x
print(gradient_descent(loss_func, init_x,0.1,100))

[6.11110793e-10  8.14814391e-10]
```

문제120. 러닝 레이트(학습률)을 크게 주고 학습을 하면 결과가 어떻게 나오는지, 작게주면 어떻게

나오는지 확인 하시오.

보기)

학습률 작게 = lr : 10

=> global minima에 도달하지 못하고 발산.

학습률 크게 = lr : 1e-10

=> global minima에 도달하지 못하고 중간에 멈춤

답)

```
def loss_func(x):
    return x[0]**2 + x[1]**2
```



```

def numerical_gradient(f,x):
    h=1e-4 #0.00001
    grad=np.zeros_like(x) #x와 형상이 같은 배열을 생성

    for idx in range(x.size): #0, 1
        tmp_val = x[idx] #x[0]

        #f(x+h) 계산
        x[idx]=tmp_val + h #3.00001
        fxh1=f(x) #3.00010|^2 + 4^2 = 25.00060001

        #f(x-h) 계산
        x[idx]=tmp_val - h #3 - 0.0001 = 2.9999|^2 + 4^2 = 24.99940001
        fxh2=f(x)

        grad[idx] = (fxh1-fxh2) / (2*h)
        x[idx] = tmp_val
    return grad

init_x = np.array([3.0, 4.0])

def gradient_descent(f, init_x, lr=0.01, step_num=100):
    x=init_x

    for i in range(step_num):
        grad = numerical_gradient(f,x)
        x -= lr * grad
    return x

print(gradient_descent(loss_func, init_x.copy(),10,100))

#init_x = np.array([3.0, 4.0])
print(gradient_descent(loss_func, init_x.copy(),1e-10,100))
[ 2.58983747e+13 -1.29524862e+12]
[2.99999994 3.99999992]

```

**설명 :** 위 프린트할때 x값이 주소값을 가져와서 사용해서 x값이 변경이 되므로 복사해서 가져와야지

연속으로 프린트를 할 수 있다.

```

ex)
def ex(y):
    x=y
    x-=1
    print(x)

init_x = np.array([3.0,4.0])

ex(init_x)
print(init_x)

init_x = np.array([3.0,4.0])

```

```
ex(init_x.copy())
print(init_x)
```

```
[2. 3.]
[2. 3.]
[2. 3.]
[3. 4.]
```

## 신경망 학습을 시키기 위한 방법

$$W1 - \frac{\partial \text{오차함수}}{\partial w_1}$$

## 학습이 스스로 되는 3층 신경망 구현

문제121. 2x3 행렬을 생성하는데 값은 랜덤값으로 생성되도록 하시오.

```
답)
import numpy as np

w = np.random.randn(2,3)
print(w)

[[-0.9292536  0.07881325 -0.95219595]
 [-0.28405154 -0.01206966  0.57126594]]
```

문제122. 위에서 구한 w값으로 아래의 입력값과 행렬의 내적을 출력하시오.

```
보기)
x=np.array([0.6, 0.9])

답)
import numpy as np

x=np.array([0.6, 0.9])
w = np.random.randn(2,3)
print(np.dot(x,w))

[-1.0621508  1.09758136 -0.36838826]
```

문제123. 문제121번 코드를 \_\_init\_\_ 라는 함수로 생성하시오.

```
답)
def __init__():
    import numpy as np
    W = np.random.randn(2,3)
    print(W)

__init__()

[[ 0.51832747 -1.74422386  0.15411538]
 [-0.42714409  0.212828   -0.78844815]]
```

문제124. 위에서 만든 가중치와 아래의 입력값과 행렬 내적을 하는 함수를 predict으로 생성하시오.

```
답)
x=np.array([0.6, 0.9])
```

```

def __init__():
    import numpy as np
    W = np.random.randn(2,3)
    return W
W=__init__()
def predict(x,w):
    return np.dot(x,w)

print(predict(x,W))
[0.54322783 0.50427274 0.91728171]

```

**문제125. 위의 predict에서 나온 결과를 softmax 출력층 함수에 통과시킨 결과가 무엇인가?**

답)

```

x=np.array([0.6, 0.9])
def __init__():
    import numpy as np
    W = np.random.randn(2,3)
    return W
W=__init__()
def predict(x,w):
    return np.dot(x,w)

def softmax(a):
    import numpy as np
    c=np.max(a)
    np_exp=np.exp(a-c)
    np_sum=np.sum(np_exp)
    res=np_exp/np_sum
    return res

print(softmax(predict(x,W)))
[0.06479373 0.92684552 0.00836075]

```

**문제126. 위에서 출력한 softmax함수 거로가와 아래의 target값과의 오차를 출력하기 위해 cross\_entroyp\_eroor 함수를 통과한 결과를 출력하시오.**

답)

```

x=np.array([0.6, 0.9])
def __init__():
    import numpy as np
    W = np.random.randn(2,3)
    return W
W=__init__()
def predict(x,w):
    return np.dot(x,w)

def softmax(a):
    import numpy as np
    c=np.max(a)
    np_exp=np.exp(a-c)
    np_sum=np.sum(np_exp)
    res=np_exp/np_sum
    return res

```

```

def CEE(y,t):
    delta=1e-7
    return -np.sum(t*np.log(y+delta))

t=np.array([0,0,1])
z=predict(x,W)
y=softmax(z)

print(CEE(y,t))
1.772239685737383

```

문제127. 위의 3가지 함수를 다 사용한 simplenet이라는 클래스를 책 134페이지를 보고 생성하고, 135페

이지를 보며 클래스가 잘 생성되었는지 확인하시오.

답)

```

import sys, os
sys.path.append(os.pardir)
import numpy as np

class simpleNet:
    def __init__(self):
        self.W = np.random.randn(2,3)

    def predict(self,x):
        return np.dot(x,self.W)

    def softmax(self,a):
        import numpy as np
        c=np.max(a)
        np_exp=np.exp(a-c)
        np_sum=np.sum(np_exp)
        res=np_exp/np_sum
        return res

    def loss(self,x,t):
        z=self.predict(x)
        y=softmax(z)
        loss=CEE(y,t)
        return loss

    def CEE(self, y,t):
        delta=1e-7
        return -np.sum(t*np.log(y+delta))

    def numerical_gradient(self,f,x):
        h=1e-4 #0.00001
        grad=np.zeros_like(x) #x와 형상이 같은 배열을 생성

        for idx in range(x.size): #0, 1
            tmp_val = x[idx] #x[0]

            #f(x+h) 계산
            x[idx]=tmp_val + h #3.00001

```

```

fxh1=f(x) #3.00010|^2 + 4^2 = 25.00060001

#f(x-h) 계산
x[idx]=tmp_val - h #3 - 0.0001 = 2.9999^2 + 4^2 = 24.99940001
fxh2=f(x)

grad[idx] = (fxh1-fxh2) / (2*h)
x[idx] = tmp_val
return grad

net=simpleNet()
print('가중치 매개변수 : \n',net.W,'\n')
x=np.array([0.6, 0.9]) #입력 데이터
p=net.predict(x)
print(p,'\n')
print('최댓값의 인덱스 : \n',np.argmax(p))

t=np.array([0,0,1]) #정답 레이블
print('기울기(손실함수 미분) : ',net.loss(x,t))
가중치 매개변수 :
[[-0.48628246 -0.08716079  0.76900084]
 [-0.10150768 -0.63540666 -1.84042669]]

[-0.38312639 -0.62416246 -1.19498352]

최댓값의 인덱스 :
0
기울기(손실함수 미분) :  1.61378912745599

```

**문제128.** TwoLayerNet클래스로 객체를 생성하고 W1, W2, b1, b2의 행렬의 모양을 확인하십시오.

```

답)
#2층 신경망
class TwoLayerNet:
    def __init__(self, input_size, hidden_size, output_size, weight_init_std=0.01):
        self.params = {}

        self.params['W1'] = weight_init_std * np.random.randn(input_size, hidden_size) #랜덤 가
중치 생성(784x50)
        self.params['b1'] = np.zeros(hidden_size)
        self.params['W2'] = weight_init_std * np.random.randn(hidden_size, output_size)
        self.params['b2'] = np.zeros(output_size)

    def predict(self, x):
        W1, W2 = self.params['W1'], self.params['W2']
        b1, b2 = self.params['b1'], self.params['b2']

        a1 = np.dot(x, W1) + b1
        z1 = sigmoid(a1)
        a2 = np.dot(z1, W2) + b2
        y = softmax(a2)

```

```

        return y

    def loss(self, x,t):
        y = self.predict(x)

        return CEE(y, t)

    def accuracy(self, x, t):
        y = self.predict(x)
        y = np.argmax(y, axis=1)
        t = np.argmax(t, axis=1)

        accuracy = np.sum(y == t) / float(x.shape[0])
        return accuracy

    def numerical_gradient(self,x,t):
        loss_W = lambda W: self.loss(x,t)

        grads = {}
        grads['W1'] = numerical_gradient(loss_W, self.params['W1'])
        grads['b1'] = numerical_gradient(loss_W, self.params['b1'])
        grads['W2'] = numerical_gradient(loss_W, self.params['W2'])
        grads['b2'] = numerical_gradient(loss_W, self.params['b2'])

        return grads

# 데이터 읽기
(x_train, t_train), (x_test, t_test) = load_mnist(normalize=True, one_hot_label=True)

network = TwoLayerNet(input_size=784, hidden_size=50, output_size=10)
print('w1',network.params.get('W1').shape)
print('w2',network.params.get('W2').shape)
print('b1',network.params.get('b1').shape)
print('b2',network.params.get('b2').shape)
w1 (784, 50)
w2 (50, 10)
b1 (50,)
b2 (10,)

```

**문제129. TwoLayerNet클래스로 객체를 생성하고 predict메소드를 실행해서 나온 결과의 행렬의 shape를**

**출력하시오.**

답)

```

# coding: utf-8
import sys, os

```

```

sys.path.append(os.pardir) # 부모 디렉터리의 파일을 가져올 수 있도록 설정
import numpy as np
import matplotlib.pyplot as plt
from dataset.mnist import load_mnist
#from prac.common.functions import *
#from prac.common.gradient import numerical_gradient

```

```

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

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

def CEE(y, t):
    delta = 1e-7 #아주 작은 값 (y가 0인 경우 -inf 값을 예방)
    return -np.sum(t*np.log(y+delta)) / y.shape[0]

# def numerical_gradient(f,x):
#     h = 1e-4
#     grad = np.zeros_like(x)
#     #
#     for idx in range(x.size):
#         tmp_val = x[idx]
#         x[idx] = tmp_val + h
#         fxh1 = f(x) # f(x+h) 를 계산
#         #
#         x[idx] = tmp_val - h
#         fxh2 = f(x) # f(x-h) 를 계산
#         #
#         grad[idx] = (fxh1 - fxh2) / (2*h)
#         x[idx]= tmp_val # 값 복원
#     #
#     return grad

def numerical_gradient(f, x):
    h = 1e-4 # 0.0001
    grad = np.zeros_like(x)

    it = np.nditer(x, flags=['multi_index'], op_flags=['readwrite'])
    while not it.finished:
        idx = it.multi_index
        tmp_val = x[idx]
        x[idx] = float(tmp_val) + h
        fxh1 = f(x) # f(x+h)

        x[idx] = tmp_val - h
        fxh2 = f(x) # f(x-h)
        grad[idx] = (fxh1 - fxh2) / (2 * h)

        x[idx] = tmp_val #媛?蹂듭쌔
        it.iternext()

    return grad

```

## #2층 신경망

```
class TwoLayerNet:
```

```
    def __init__(self, input_size, hidden_size, output_size, weight_init_std=0.01):
```

```
        self.params = {}
```

```
        self.params['W1'] = weight_init_std * np.random.randn(input_size, hidden_size) #랜덤 가
```

```
    중치 생성(784x50)
```

```
        self.params['b1'] = np.zeros(hidden_size)
```

```
        self.params['W2'] = weight_init_std * np.random.randn(hidden_size, output_size)
```

```
        self.params['b2'] = np.zeros(output_size)
```

```
    def predict(self, x):
```

```
        W1, W2 = self.params['W1'], self.params['W2']
```

```
        b1, b2 = self.params['b1'], self.params['b2']
```

```
        a1 = np.dot(x, W1) + b1 #100x784  $\odot$  784x50 = 100x50+100x50 = 100x50
```

```
        z1 = sigmoid(a1) #100x50 행렬인데 0~1사이의 숫자로 변경
```

```
        a2 = np.dot(z1, W2) + b2 #100x50  $\odot$  50x10 = 100x10+100x10 = 100x10
```

```
        y = softmax(a2) #100x10 의 10개의 확률벡터가 생성
```

```
        return y
```

```
    def loss(self, x,t):
```

```
        y = self.predict(x)
```

```
        return CEE(y, t)
```

```
    def accuracy(self, x, t):
```

```
        y = self.predict(x)
```

```
        y = np.argmax(y, axis=1)
```

```
        t = np.argmax(t, axis=1)
```

```
        accuracy = np.sum(y == t) / float(x.shape[0])
```

```
        return accuracy
```

```
    def numerical_gradient(self,x,t):
```

```
        loss_W = lambda W: self.loss(x,t)
```

```
        grads = {}
```

```
        grads['W1'] = numerical_gradient(loss_W, self.params['W1'])
```

```
        grads['b1'] = numerical_gradient(loss_W, self.params['b1'])
```

```
        grads['W2'] = numerical_gradient(loss_W, self.params['W2'])
```

```
        grads['b2'] = numerical_gradient(loss_W, self.params['b2'])
```

```
        return grads
```

```
# 데이터 읽기
```

```
def get_data():
```

```
    (x_train, t_train), (x_test, t_test) = load_mnist(normalize=True, one_hot_label=True)
```

```
    network = TwoLayerNet(input_size=784, hidden_size=50, output_size=10)
```

```
    return x_test, t_test
```

```
x,t=get_data()
```



```
print((network.predict(x[:100]).shape))
(100, 10)
```

**문제130. TwoLayerNet클래스로 객체를 생성하고 CEE메소드를 실행해서 결과(오차)를 출력 하시오.**

```
답)
# coding: utf-8
import sys, os

sys.path.append(os.pardir) # 부모 디렉터리의 파일을 가져올 수 있도록 설정
import numpy as np
import matplotlib.pyplot as plt
from dataset.mnist import load_mnist
#from prac.common.functions import *
#from prac.common.gradient import numerical_gradient

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

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

def CEE(y, t):
    delta = 1e-7 #아주 작은 값 (y가 0인 경우 -inf 값을 예방)
    return -np.sum(t*np.log(y+delta)) / y.shape[0]

# def numerical_gradient(f,x):
#     h = 1e-4
#     grad = np.zeros_like(x)
#     #
#     for idx in range(x.size):
#         tmp_val = x[idx]
#         x[idx] = tmp_val + h
#         fxh1 = f(x) # f(x+h) 를 계산
#         #
#         x[idx] = tmp_val - h
#         fxh2 = f(x) # f(x-h) 를 계산
#         #
#         grad[idx] = (fxh1 - fxh2) / (2*h)
#         x[idx]= tmp_val # 값 복원
#     #
#     return grad

def numerical_gradient(f, x):
```

```

h = 1e-4 # 0.0001
grad = np.zeros_like(x)

it = np.nditer(x, flags=['multi_index'], op_flags=['readwrite'])
while not it.finished:
    idx = it.multi_index
    tmp_val = x[idx]
    x[idx] = float(tmp_val) + h
    fxh1 = f(x) # f(x+h)

    x[idx] = tmp_val - h
    fxh2 = f(x) # f(x-h)
    grad[idx] = (fxh1 - fxh2) / (2 * h)

    x[idx] = tmp_val #媛?蹂듭쌔
    it.iternext()

return grad

```

## #2층 신경망

```

class TwoLayerNet:
    def __init__(self, input_size, hidden_size, output_size, weight_init_std=0.01):
        self.params = {}

        self.params['W1'] = weight_init_std * np.random.randn(input_size, hidden_size) #랜덤 가
중치 생성(784x50)
        self.params['b1'] = np.zeros(hidden_size)
        self.params['W2'] = weight_init_std * np.random.randn(hidden_size, output_size)
        self.params['b2'] = np.zeros(output_size)

    def predict(self, x):
        W1, W2 = self.params['W1'], self.params['W2']
        b1, b2 = self.params['b1'], self.params['b2']

        a1 = np.dot(x, W1) + b1 #100x784 ⊙ 784x50 = 100x50+100x50 = 100x50
        z1 = sigmoid(a1) #100x50 행렬인데 0~1사이의 숫자로 변경

        a2 = np.dot(z1, W2) + b2 #100x50 ⊙ 50x10 = 100x10+100x10 = 100x10
        y = softmax(a2) #100x10 의 10개의 확률벡터가 생성

        return y

    def loss(self, x, t):
        y = self.predict(x)

        return CEE(y, t)

    def accuracy(self, x, t):
        y = self.predict(x)
        y = np.argmax(y, axis=1)
        t = np.argmax(t, axis=1)

        accuracy = np.sum(y == t) / float(x.shape[0])
        return accuracy

```

```

def numerical_gradient(self,x,t):
    loss_W = lambda W: self.loss(x,t)

    grads = {}
    grads['W1'] = numerical_gradient(loss_W, self.params['W1'])
    grads['b1'] = numerical_gradient(loss_W, self.params['b1'])
    grads['W2'] = numerical_gradient(loss_W, self.params['W2'])
    grads['b2'] = numerical_gradient(loss_W, self.params['b2'])

    return grads

# 데이터 읽기
def get_data():
    (x_train, t_train), (x_test, t_test) = load_mnist(normalize=True, one_hot_label=True)
    network = TwoLayerNet(input_size=784, hidden_size=50, output_size=10)
    return x_test, t_test

x,t=get_data()
print(network.loss(x_train[:100], t_train[:100]))
6.910148887209832

```

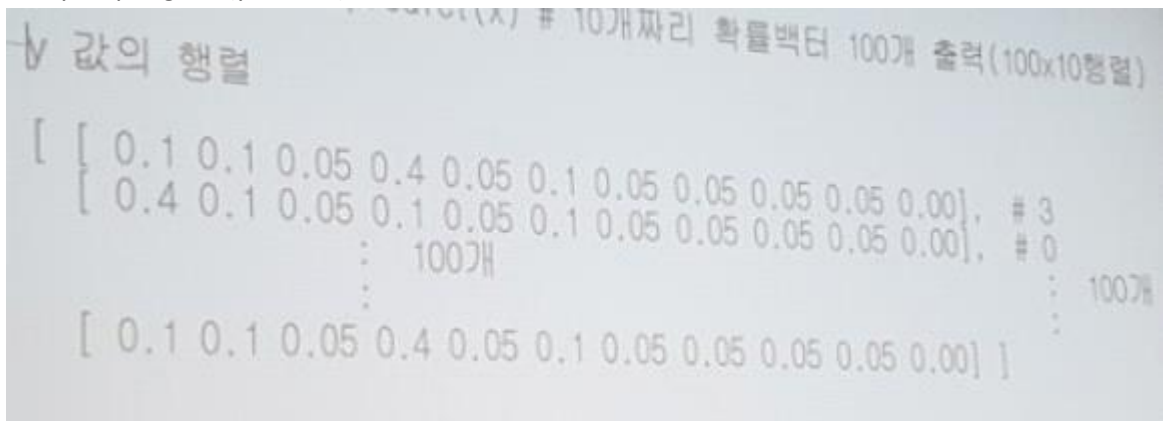
### accuracy 함수의 이해

" 예상한 숫자와 실제 숫자를 비교해서 정확도를 출력하는 함수 "

```

def accuracy(self, x, t):
    y = self.predict(x) #10개(열)짜리 확률벡터 100개(행) 출력(100x10행렬)
    y = np.argmax(y, axis=1) #100행중에 가장 큰 인덱스값 100개

```



```

t = np.argmax(t, axis=1) #원핫인코딩 되어있는 100개중 가장 큰 인덱스값 100개

```

```

accuracy = np.sum(y == t) / float(x.shape[0]) #100개중 같으면 T, 아니면 F인데 T인걸
sum/(100)
return accuracy

```

문제131. TwoLayerNet클래스로 객체를 생성하고 accuracy(x,f) 메소드를 실행해서 100장 테스트 훈련

데이터를 입력 했을때의 정확도를 출력하시오.

답)

```

def get_data():
    (x_train, t_train), (x_test, t_test) = load_mnist(normalize=True, one_hot_label=True)

```

```

network = TwoLayerNet(input_size=784, hidden_size=50, output_size=10)
return x_test, t_test

x,t=get_data()
print(network.accuracy(x[:100],t[:100]))
0.1

```

## numerical\_gradient 함수

"비용함수와 가중치 또는 바이어스를 입력받아 기울기를 출력하는 함수"

```

def numerical_gradient(self,x,t):
    loss_W = lambda W: self.loss(x,t)

    grads = {}
    grads['W1'] = numerical_gradient(loss_W, self.params['W1'])
    grads['b1'] = numerical_gradient(loss_W, self.params['b1'])
    grads['W2'] = numerical_gradient(loss_W, self.params['W2'])
    grads['b2'] = numerical_gradient(loss_W, self.params['b2'])

    return grads

```

## 비용함수 생성 방법(p.135)

```

network = TwoLayerNet(input_size=784, hidden_size=50, output_size=10)
def f(W):
    return network.loss(x,t)

```

설명 : 여기서 정의한 f(W)함수의 W는 더미로 만든 것이다.

numerical\_gradient(f,x) 내부에서 f(x)를 실행하는데, 그와의 일관성을 위해 f(w)를 정의 한것.

예제)

```

# 데이터 읽기
network = TwoLayerNet(input_size=784, hidden_size=50, output_size=10)
(x_train, t_train), (x_test, t_test) = load_mnist(normalize=True, one_hot_label=True)
x = x_train[:100]
t = t_train[:100]

```

```

def f(W):
    return network.loss(x,t)

```

```

W1_grad = numerical_gradient(f, network.params.get('W1'))
print(W1_grad)

```

```

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

```

## lambda 표현식

" 여러줄의 코드를 한 줄로 만들어주는 인자 "

```
ex)
def hap(x,y):
    return x+y
print(hap(10,20))

↓

print( (lambda x,y:x+y)(10,20) )
```

문제132. 아래의 비용함수를 lambda 표현식으로 한줄로 변경해서 출력하시오.

```
보기)
def f(W):
    return network.loss(x,t)
```

```
답)
f=lambda W :network.loss(x,t)
```

문제133. TwoLayerNet클래스를 가지고 학습 시키는 코드를 완성 시키시오.

```
답)
# coding: utf-8
import sys, os

sys.path.append(os.pardir) # 부모 디렉터리의 파일을 가져올 수 있도록 설정
import numpy as np
import matplotlib.pyplot as plt
from dataset.mnist import load_mnist
#from prac.common.functions import *
#from prac.common.gradient import numerical_gradient

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

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

def CEE(y, t):
    delta = 1e-7 #아주 작은 값 (y가 0인 경우 -inf 값을 예방)
    return -np.sum(t*np.log(y+delta)) / y.shape[0]

# def numerical_gradient(f,x):
#     h = 1e-4
#     grad = np.zeros_like(x)
#     #
#     for idx in range(x.size):
#         tmp_val = x[idx]
#         x[idx] = tmp_val + h
```

```

#    fxh1 = f(x) # f(x+h) 를 계산
#
#    x[idx] = tmp_val - h
#    fxh2 = f(x) # f(x-h) 를 계산
#
#    grad[idx] = (fxh1 - fxh2) / (2*h)
#    x[idx] = tmp_val # 값 복원
#
#    return grad

```

```

def numerical_gradient(f, x):
    h = 1e-4 # 0.0001
    grad = np.zeros_like(x)

    it = np.nditer(x, flags=['multi_index'], op_flags=['readwrite'])
    while not it.finished:
        idx = it.multi_index
        tmp_val = x[idx]
        x[idx] = float(tmp_val) + h
        fxh1 = f(x) # f(x+h)

        x[idx] = tmp_val - h
        fxh2 = f(x) # f(x-h)
        grad[idx] = (fxh1 - fxh2) / (2 * h)

        x[idx] = tmp_val #媛?蹂듭쌔
        it.iternext()

    return grad

```

## #2층 신경망

```

class TwoLayerNet:
    def __init__(self, input_size, hidden_size, output_size, weight_init_std=0.01):
        self.params = {}

        self.params['W1'] = weight_init_std * np.random.randn(input_size, hidden_size) #랜덤 가
중치 생성(784x50)
        self.params['b1'] = np.zeros(hidden_size)
        self.params['W2'] = weight_init_std * np.random.randn(hidden_size, output_size)
        self.params['b2'] = np.zeros(output_size)

    def predict(self, x):
        W1, W2 = self.params['W1'], self.params['W2']
        b1, b2 = self.params['b1'], self.params['b2']

        a1 = np.dot(x, W1) + b1 #100x784 ⊙ 784x50 = 100x50+100x50 = 100x50
        z1 = sigmoid(a1) #100x50 행렬인데 0~1사이의 숫자로 변경

        a2 = np.dot(z1, W2) + b2 #100x50 ⊙ 50x10 = 100x10+100x10 = 100x10
        y = softmax(a2) #100x10 의 10개의 확률벡터가 생성

        return y

```

```

def loss(self, x,t):
    y = self.predict(x)

    return CEE(y, t)

def accuracy(self, x, t):
    y = self.predict(x) #10개(열)짜리 확률벡터 100개(행) 출력(100x10행렬)
    y = np.argmax(y, axis=1) #100행중에 가장큰 인덱스값 100개

    t = np.argmax(t, axis=1) #원핫인코딩 되있는 100개중 가장큰 인덱스값 100개

    accuracy = np.sum(y == t) / float(x.shape[0]) #100개중 같으면 T, 아니면 F인데 T인걸
sum/x.shape[0](100)
    return accuracy

def numerical_gradient(self,x,t): #클래스 밖에 같은 이름 함수가 있음
    loss_W = lambda W: self.loss(x,t)

    grads = {}
    grads['W1'] = numerical_gradient(loss_W, self.params['W1'])
    grads['b1'] = numerical_gradient(loss_W, self.params['b1'])
    grads['W2'] = numerical_gradient(loss_W, self.params['W2'])
    grads['b2'] = numerical_gradient(loss_W, self.params['b2'])

    return grads

# 데이터 읽기
network = TwoLayerNet(input_size=784, hidden_size=50, output_size=10)
(x_train, t_train), (x_test, t_test) = load_mnist(normalize=True, one_hot_label=True)
x = x_train[:100]
t = t_train[:100]

f=lambda W :network.loss(x,t)

W1_grad = numerical_gradient( f, network.params.get('W1') )
print (W1_grad)

#하이퍼파라미터
iters_num = 10000 # 반복 횟수를 적절히 설정한다.
train_size = x_train.shape[0] #60000x784
batch_size = 100 # 미니배치 크기
learning_rate = 0.1

#그래프 그릴때 데이터를 담을 리스트 변수 3개 선언
train_loss_list = [] #훈련 데이터 손실 (점점 작아짐)
train_acc_list = [] #훈련 데이터 정확도 (점점 높아짐)
test_acc_list = [] #테스트 데이터 정확도 (점점 높아짐)

# 1에폭당 반복 수

```

```

iter_per_epoch = max(train_size / batch_size, 1)
# 600          60000/100= 600

for i in range(iters_num):
    # 미니배치 획득

    batch_mask = np.random.choice(train_size, batch_size) #0~60000의 숫자중에 100개를 랜
   덤으로 생성
    x_batch = x_train[batch_mask] #100x784
    t_batch = t_train[batch_mask] #100x10

    # 기울기 계산
    grad = network.numerical_gradient(x_batch, t_batch) #수치미분을 이용한 기울기 구하기
    #grad = network.gradient(x_batch, t_batch) #오차 역전파를 이용한 기울기 구하기

    # 매개변수 갱신 #손실(비용)이 점점 줄어드는것을 보기위해
    for key in ('W1', 'b1', 'W2', 'b2'):
        network.params[key] -= learning_rate * grad[key]

    # 학습 경과 기록
    loss = network.loss(x_batch, t_batch)
    train_loss_list.append(loss)

    # 1에폭당 정확도 계산
    if i % iter_per_epoch == 0: #600/600, 1200/600, 1800/600, ...
        train_acc = network.accuracy(x_train, t_train)
        test_acc = network.accuracy(x_test, t_test)
        train_acc_list.append(train_acc)
        test_acc_list.append(test_acc)
        print("train acc, test acc | " + str(train_acc) + ", " + str(test_acc))

# 그래프 그리기
markers = {'train': 'o', 'test': 's'}
x = np.arange(len(train_acc_list))
plt.plot(x, train_acc_list, label='train acc')
plt.plot(x, test_acc_list, label='test acc', linestyle='--')
plt.xlabel("epochs")
plt.ylabel("accuracy")
plt.ylim(0, 1.0)
plt.legend(loc='lower right')
plt.show()

```

---

```

train acc, test acc | 0.11236666666666667, 0.1135

```

**첫번째 정확도는 금방? 나오지만 두번째 정확도는 엄청 오래걸린다!!!!**

그래서 오차역전파를 이용한다.

**문제134. 수치미분이 아닌 오차 역전파를 이용한 신경망 학습으로 2층 신경망 코드를 구현 하시오.**

답)

1. common 패키지를 working directory에 가져다 둔다.



2. 오차 역전파를 이용한 신경망 코드를 돌린다.

```
# coding: utf-8
import sys, os

sys.path.append(os.pardir) # 부모 디렉터리의 파일을 가져올 수 있도록 설정
import numpy as np
from common.layers import *
from common.gradient import numerical_gradient
from collections import OrderedDict
import matplotlib.pyplot as plt
from dataset.mnist import load_mnist

class TwoLayerNet:
    def __init__(self, input_size, hidden_size, output_size, weight_init_std=0.01):
        # 가중치 초기화
        self.params = {}
        self.params['W1'] = weight_init_std * np.random.randn(input_size, hidden_size)
        self.params['b1'] = np.zeros(hidden_size)
        self.params['W2'] = weight_init_std * np.random.randn(hidden_size, output_size)
        self.params['b2'] = np.zeros(output_size)

        # 계층 생성
        self.layers = OrderedDict()
        self.layers['Affine1'] = Affine(self.params['W1'], self.params['b1'])
        self.layers['Relu1'] = Relu()
        self.layers['Affine2'] = Affine(self.params['W2'], self.params['b2'])

        self.lastLayer = SoftmaxWithLoss()

    def predict(self, x):
        for layer in self.layers.values():
            x = layer.forward(x)
        return x

    # x : 입력 데이터, t : 정답 레이블
    def loss(self, x, t):
        y = self.predict(x)
        return self.lastLayer.forward(y, t)

    def accuracy(self, x, t):
        y = self.predict(x)
        y = np.argmax(y, axis=1)
        if t.ndim != 1: t = np.argmax(t, axis=1)

        accuracy = np.sum(y == t) / float(x.shape[0])
        return accuracy

    # x : 입력 데이터, t : 정답 레이블
    def numerical_gradient(self, x, t):
        loss_W = lambda W: self.loss(x, t)
        grads = {}
        grads['W1'] = numerical_gradient(loss_W, self.params['W1'])
        grads['b1'] = numerical_gradient(loss_W, self.params['b1'])
        grads['W2'] = numerical_gradient(loss_W, self.params['W2'])
```

```

        grads['b2'] = numerical_gradient(loss_W, self.params['b2'])
        return grads

def gradient(self, x, t):
    # forward
    self.loss(x, t)

    # backward
    dout = 1
    dout = self.lastLayer.backward(dout)
    layers = list(self.layers.values())
    layers.reverse()
    for layer in layers:
        dout = layer.backward(dout)

    # 결과 저장
    grads = {}
    grads['W1'], grads['b1'] = self.layers['Affine1'].dW, self.layers['Affine1'].db
    grads['W2'], grads['b2'] = self.layers['Affine2'].dW, self.layers['Affine2'].db
    return grads

# 데이터 읽기
(x_train, t_train), (x_test, t_test) = load_mnist(normalize=True, one_hot_label=True)
network = TwoLayerNet(input_size=784, hidden_size=50, output_size=10)

# 하이퍼파라미터
iters_num = 10000 # 반복 횟수를 적절히 설정한다.
train_size = x_train.shape[0] # 60000 개
batch_size = 100 # 미니배치 크기
learning_rate = 0.1

train_loss_list = []
train_acc_list = []
test_acc_list = []

# 1에폭당 반복 수
iter_per_epoch = max(train_size / batch_size, 1)

for i in range(iters_num): # 10000
    # 미니배치 획득 # 랜덤으로 100개씩 뽑아서 10000번을 수행하니까 백만번
    batch_mask = np.random.choice(train_size, batch_size) # 100개 씩 뽑아서 10000번 백만
    번
    x_batch = x_train[batch_mask]
    t_batch = t_train[batch_mask]

    # 기울기 계산
    # grad = network.numerical_gradient(x_batch, t_batch)

    grad = network.gradient(x_batch, t_batch)

    # 매개변수 갱신
    for key in ('W1', 'b1', 'W2', 'b2'):

```

```

network.params[key] -= learning_rate * grad[key]

# 학습 경과 기록
loss = network.loss(x_batch, t_batch)
train_loss_list.append(loss) # cost 가 점점 줄어드는것을 보려고

# 1에폭당 정확도 계산 # 여기는 훈련이 아니라 1에폭 되었을때 정확도만 체크
if i % iter_per_epoch == 0: # 600 번마다 정확도 쌓는다.
    train_acc = network.accuracy(x_train, t_train)
    test_acc = network.accuracy(x_test, t_test)
    train_acc_list.append(train_acc) # 10000/600 개 16개 # 정확도가 점점 올라감
    test_acc_list.append(test_acc) # 10000/600 개 16개 # 정확도가 점점 올라감
    print("train acc, test acc | " + str(train_acc) + ", " + str(test_acc))

# 그래프 그리기
markers = {'train': 'o', 'test': 's'}
x = np.arange(len(train_acc_list))
plt.plot(x, train_acc_list, label='train acc')
plt.plot(x, test_acc_list, label='test acc', linestyle='--')
plt.xlabel("epochs")
plt.ylabel("accuracy")
plt.ylim(0, 1.0)
plt.legend(loc='lower right')
plt.show()

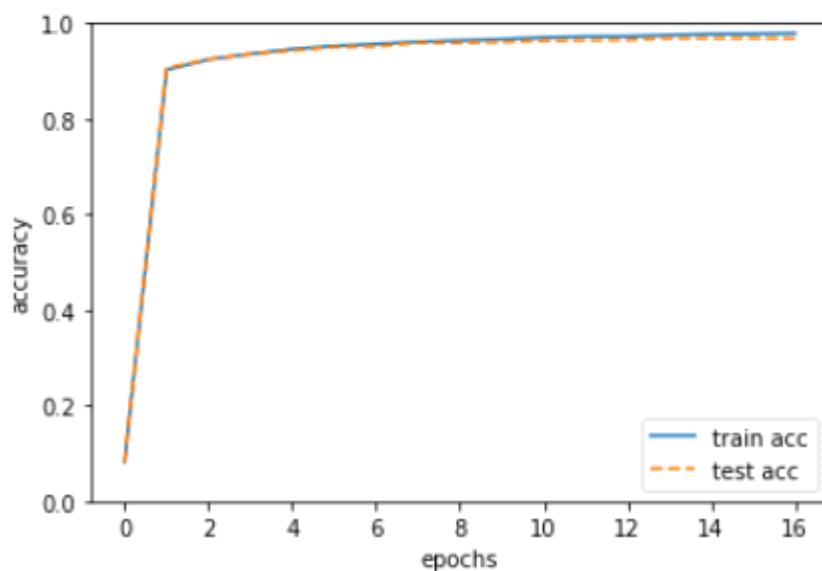
```

---

```

train acc, test acc | 0.0821, 0.0816
train acc, test acc | 0.9022833333333333, 0.9064
train acc, test acc | 0.9244333333333333, 0.9246
train acc, test acc | 0.9358333333333333, 0.9355
train acc, test acc | 0.9458833333333333, 0.9428
train acc, test acc | 0.95255, 0.9504
train acc, test acc | 0.9567333333333333, 0.9514
train acc, test acc | 0.9613666666666667, 0.959
train acc, test acc | 0.9641333333333333, 0.9589
train acc, test acc | 0.9667333333333333, 0.9608
train acc, test acc | 0.9704, 0.9635
train acc, test acc | 0.9723333333333334, 0.9646
train acc, test acc | 0.97265, 0.9653
train acc, test acc | 0.9754833333333334, 0.9688
train acc, test acc | 0.97735, 0.9687
train acc, test acc | 0.9777166666666667, 0.9691
train acc, test acc | 0.9791, 0.9681

```



## 5장 오차역전파를 이용한 3층신경망 학습

2018년 8월 28일 화요일    오후 4:13

### 역전파란?

"신경망 학습 처리에서 최소화되는 함수의 경사를 효율적으로 계산하기 위한 방법으로 '오류 역전파'가 있다.

- 함수의 경사(기울기)를 계산하는 방법?

1. 수치미분 : 성능이 너무너무너무너무 느리다.
2. 오류 역전파

순전파 : 입력층 --> 은닉층 --> 출력층

역전파 : 출력층 <-- 은닉층 <-- 입력층

여기서 역전파를 시키는 것이 오류(오차) 이다.

출력층부터 차례대로 역방향으로 따라 올라가 각 층에 있는 노드의 오차를 계산할 수 있다.

각 노드의 오차를 계산하면 그 오차를 사용해서 함수의 경사(기울기)를 계산할 수 있다.

"즉 전파된 오차를 이용해서 가중치를 조정한다."

↓

"오류(오차) 역전파"

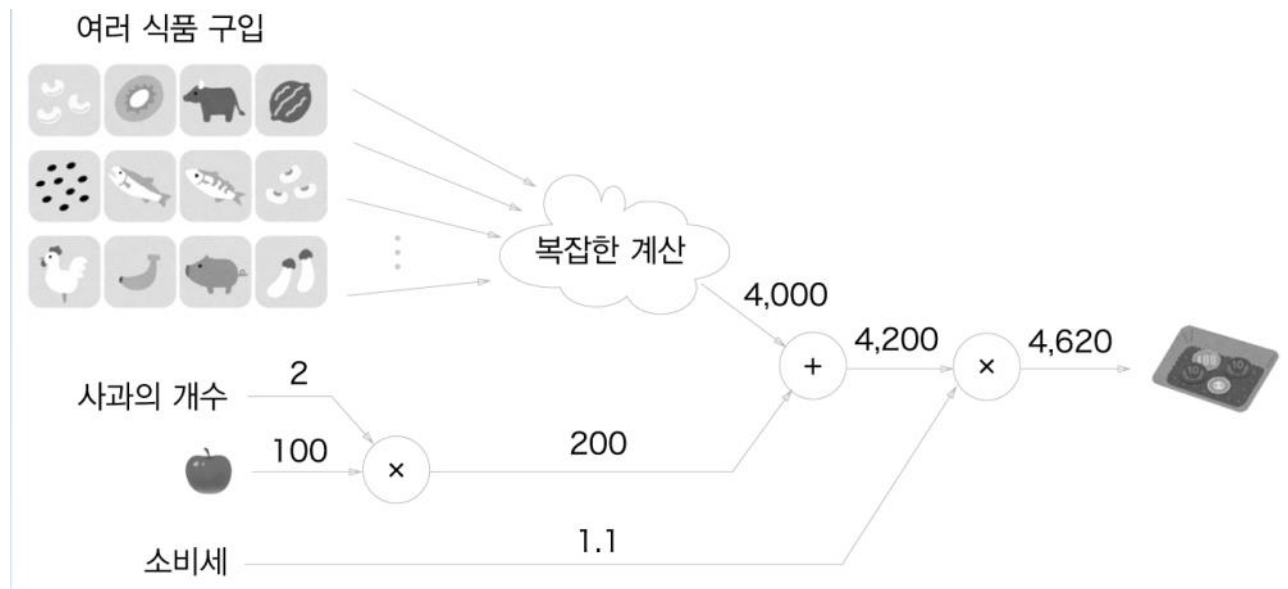
### 계산그래프

"순전파와 역전파에 계산 과정을 그래프로 나타내는 방법"

#### 계산 그래프의 장점

- 국소적 계산을 할 수 있다.

국소적 계산이란? 전체의 어떤일이 벌어지든 상관없이 자신과 관계된 정보만으로 다음 결과를 출력.



4000원이라는 숫자가 어떻게 계산되었느냐와는 상관없이 사과가 어떻게 200원이 되었는가?만  
 신경쓰면 된다는 것이 국소적 계산이다.

## 배운 내용 복습

### 1장. 파이썬 기본 문법

numpy  
 matplotlib

### 2장. 퍼셉트론

단층 (입력층 -> 출력층)  
 다층 (입력층 -> 은닉층 -> 출력층)

### 3장. 3층 신경망 구현 (이미 최적화 되어 있는 가중치와 바이어스를 이용해서)

활성화 함수 (계단, 시그모이드, 렐루)  
 출력층 함수 (항등, 소프트맥스)

### 4장. 3층 신경망 학습 (수치미분을 이용해서 학습시킴)

오차(비용) 함수 (평균제곱오차, 교차엔트로피)  
 비용함수를 미분하기 위해 수치미분 함수  
 3층 신경망 구현을 위한 클래스 생성  
**장점 :** 구현이 간단.  
**단점 :** 시간이 오래 걸린다.

### 5장. 3층 신경망 학습 (오차역전파를 이용해서 학습시킴)

구현하는 내용을 쉽게 이해하기 위해서 책에서 소개하는 방법?

" 계산 그래프 "

### 6장. 신경망의 정확도를 올리기 위한 여러가지 방법들

# • 왜 계산 그래프로 문제를 해결 하는가?

" 전체가 아무리 복잡해도 각 노드에서 단순한 계산에 집중하여 문제를 단순화 시킬 수 있다. "

# • 실제로 계산 그래프를 사용하는 가장 큰 이유는?

" 역전파를 통해서 미분을 효율적으로 계산할 수 있는점에 있다."

↓

"사과값이 '아주 조금' 올랐을때 '지불금액'이 얼마나 증가하는지 알고 싶은 경우"

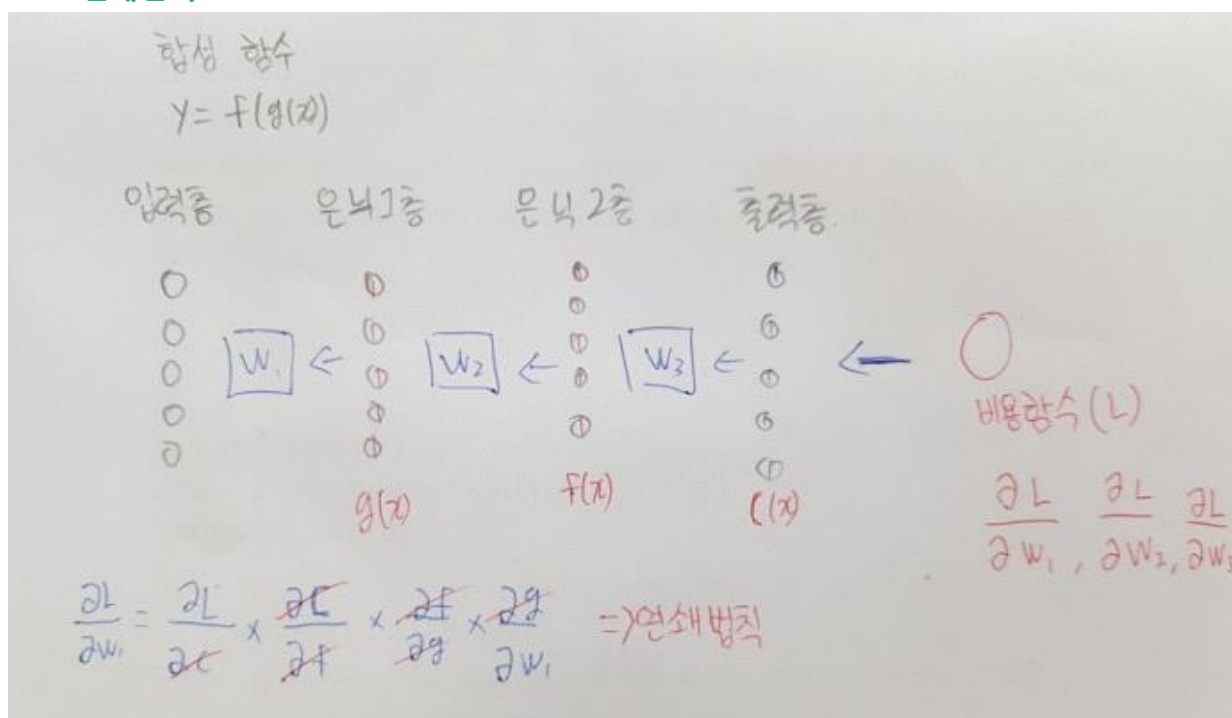
-----

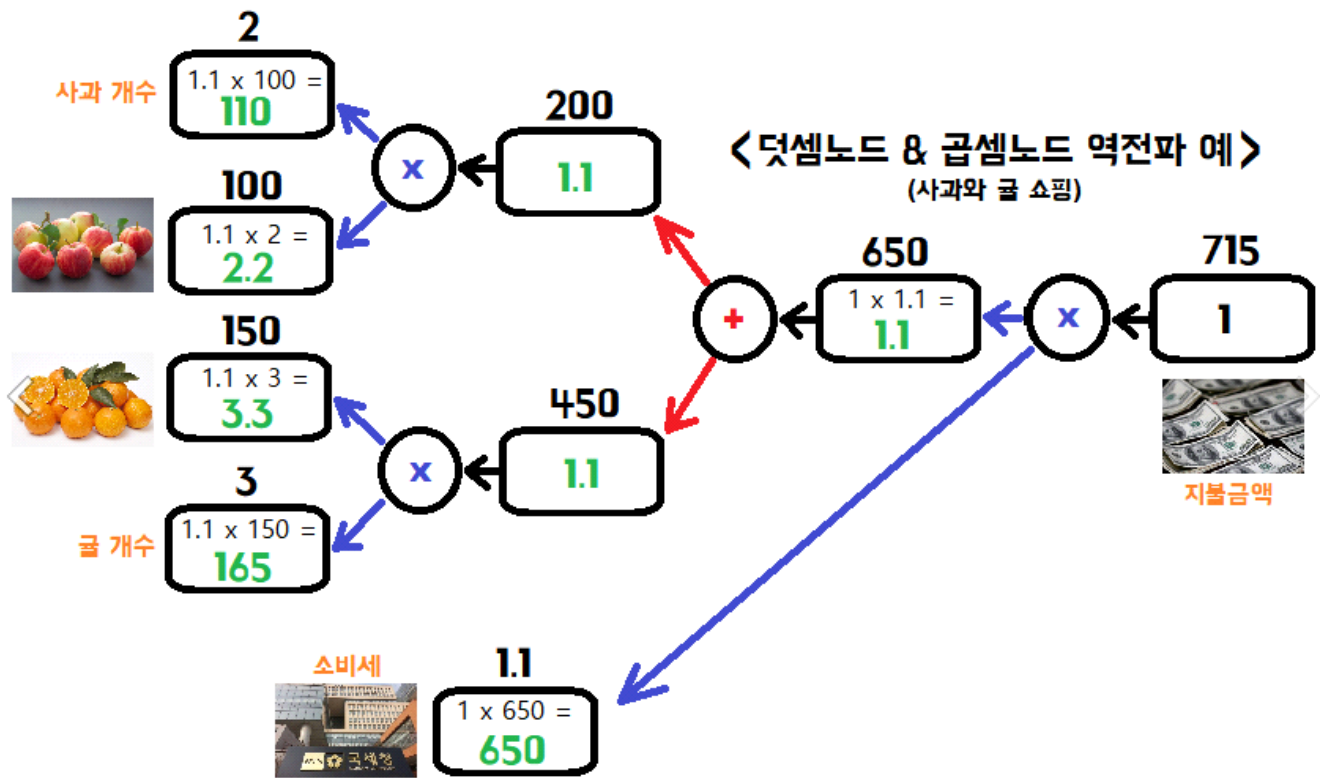
지불금액을 사과값으로 편미분하면 알 수 있다.

↓ 계산 그래프 역전파를 이용하면 되는데

사과값이 1원이 오르면 최종 금액은 2.2원이 오른다.

## 연쇄법칙





문제135. 곱셈계층을 파이썬으로 구현하시오.(p.161)

답)

```

class MulLayer:
    def __init__(self):
        self.x=None
        self.y=None

    def forward(self, x,y):
        self.x=x
        self.y=y
        out=x*y
        return out

    def backward(self, dout):
        dx=dout*self.y #x와 y를 바꾼다.
        dy=dout*self.x
        return dx, dy
    
```

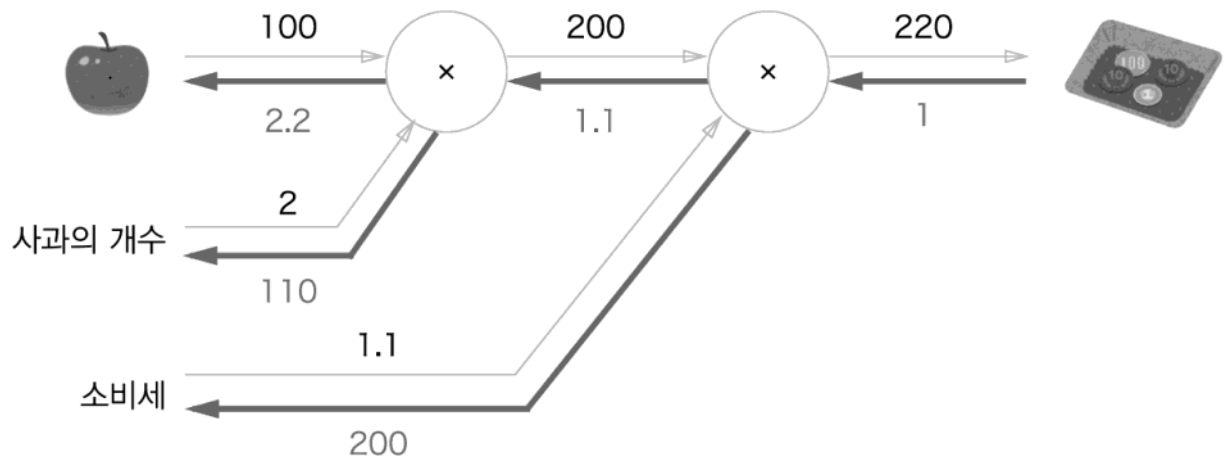
문제136. 위에서 만든 곱셈 클래스를 객체화 시켜서 사과 가격의 총 가격을 구하시오.

보기)

```

apple=100
apple_num=2
tax=1.2
    
```





답)

```
class MulLayer:
    def __init__(self):
        self.x=None
        self.y=None

    def forward(self, x,y):
        self.x=x
        self.y=y
        out=x*y
        return out

    def backward(self, dout):
        dx=dout*self.y #x와 y를 바꾼다.
        dy=dout*self.x
        return dx, dy
```

```
apple=100
apple_num=2
tax=1.1
```

```
m_a_l=MulLayer()
m_t_l=MulLayer()
```

```
apple_price=m_a_l.forward(apple, apple_num)
price=m_t_l.forward(apple_price,tax)
```

```
print(price)
220.00000000000003
```

문제137. 덧셈 계층을 파이썬으로 구현하시오.

답)

```
class AddLayer:
    def __init__(self):
        pass # 초기화가 필요없어서 사용안함

    def forward(self, x,y):
        out=x+y
        return out
```

```
def backward(self, dout):
    dx=dout*1 #덧셈노드는 상류값을 여과없이 하류로 흘려보냄
    dy=dout*1
    return dx, dy
```

문제138. 층을 4개로해서 사과와 귤의 총 가격을 구하는 책p149의 그림5-3의 신경망을 위에서 만든 덧셈

계층과 곱셈 계층으로 구현하시오.

보기)

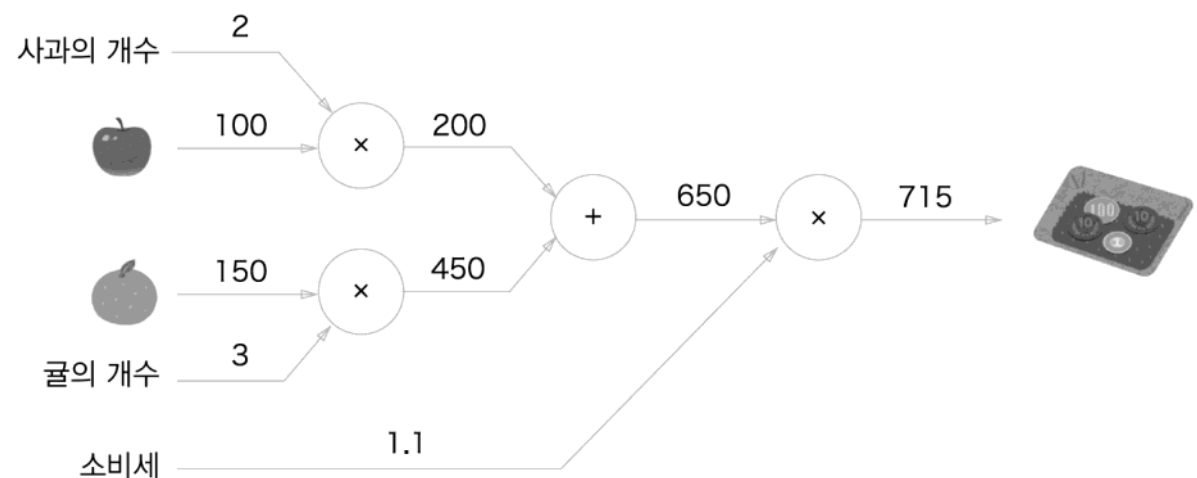
사과가격 :100원

사과개수 : 2개

귤 가격 : 150원

귤 개수 : 3개

소비세 : 1.1



답)

```
class MulLayer:
    def __init__(self):
        self.x=None
        self.y=None

    def forward(self, x,y):
        self.x=x
        self.y=y
        out=x*y
        return out

    def backward(self, dout):
        dx=dout*self.y #x와 y를 바꾼다.
        dy=dout*self.x
        return dx, dy

class AddLayer:
    def __init__(self):
        pass # 초기화가 필요없어서 사용안함

    def forward(self, x,y):
        out=x+y
```

```

        return out

    def backward(self, dout):
        dx=dout*1
        dy=dout*1
        return dx, dy

apple=100
apple_num=2
orange=150
orange_num=3
tax=1.1

mul_a_l=MulLayer()
mul_o_l=MulLayer()
add_a_o_l=AddLayer()
mul_t_l=MulLayer()

apple_price=mul_a_l.forward(apple, apple_num)
orange_price=mul_o_l.forward(orange,orange_num)
all_price=add_a_o_l.forward(apple_price,orange_price)
price=mul_t_l.forward(all_price,tax)

print(price)
715.0000000000001

```

문제139. 문제138번의 순전파로 출력한 과일 가격의 총합 신경망의 역전파를 구현하시오.  
답)

## 활성화 함수 계층 구현하기

### 1. Relu 계층

"0보다 큰값이 입력되면 그값을 그대로 출력 하고 0이거나 0보다 작으면 0을 출력"  
순전파 함수

역전파 함수

문제140. p.166의 Relu 클래스를 생성하시오.

답)

```

class Relu:
    def __init__(self):
        self.mask=None

    def forward(self,x):
        self.mask=(x<=0)
        out=x.copy()
        out[self.mask]=0

    def backward(self,dout):
        dout[self.mask]=0
        dx=dout
        return dx

```

**설명 : Relu 클래스를 이해하려면 두가지를 알아야 한다.**

1. copy의 의미

2.  $x[x \leq 0]$ 의 의미

ex)

```
import numpy as np
```

```
import copy
```

```
x=np.array([[1.0,-0.5],[-2.0, 3.0]])
```

```
print(x)
```

```
mask=(x<=0) #0보다 작으면 T 아니면 F
```

```
print(mask)
```

```
out=x.copy()
```

```
out[mask]=0 #T인곳에만 0을 넣겠다
```

```
print(out)
```

```
[[ 1.  -0.5]
 [-2.   3. ]]
[[False  True]
 [ True False]]
[[1.  0.]
 [0.  3.]]
```

**문제141. Relu클래스를 객체화 시켜서 아래의 입력 데이터 x값을 받아서 출력되는 순전파 행렬을**

**출력하시오.**

보기)

```
x=np.array([[1.0,-0.5],[-2.0, 3.0]])
```

답)

```
class Relu:
```

```
    import numpy as np
```

```
    import copy
```

```
    def __init__(self):
```

```
        self.mask=None
```

```
    def forward(self,x):
```

```
        self.mask=(x<=0)
```

```
        out=x.copy()
```

```
        out[self.mask]=0
```

```
        return out
```

```
    def backward(self,dout):
```

```
        dout[self.mask]=0
```

```
        dx=dout
```

```
        return dx
```

```
x=np.array([[1.0,-0.5],[-2.0, 3.0]])
```

```
relu=Relu()
```

```
res=relu.forward(x)
```

```
print(res)
```

```
[[1. 0.]  
 [0. 3.]]
```

문제142. Relu클래스를 객체화 시켜서 아래의 입력 데이터 x값을 받아서 출력되는 역전파 행렬을

출력하시오.

보기)

```
x=np.array([[1.0,-0.5],[-2.0, 3.0]])
```

답)

```
relu=Relu()
```

```
res1=relu.forward(x)
```

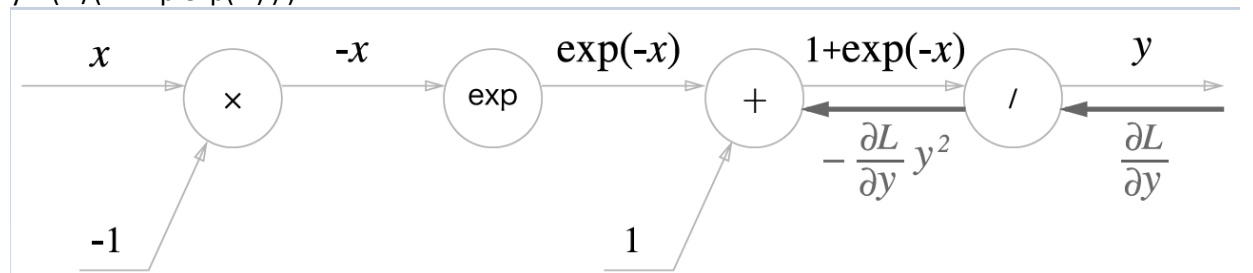
```
res2=relu.backward(res1)
```

```
print(res2)
```

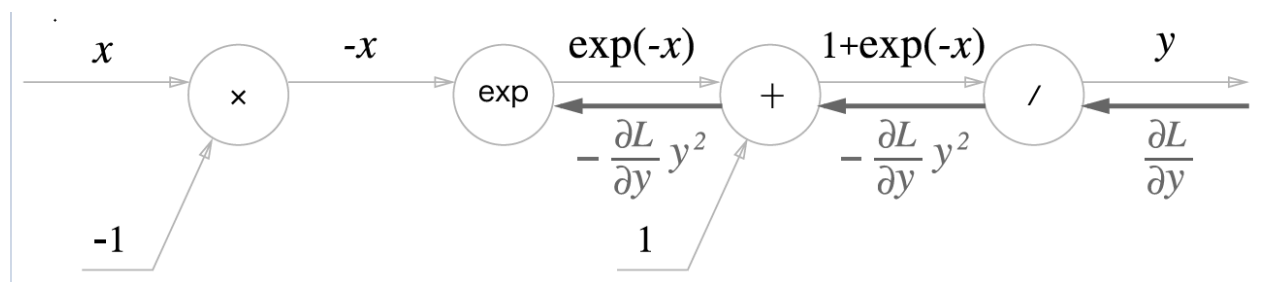
```
[[1. 0.]  
 [0. 3.]]
```

## 2. Sigmoid 계층

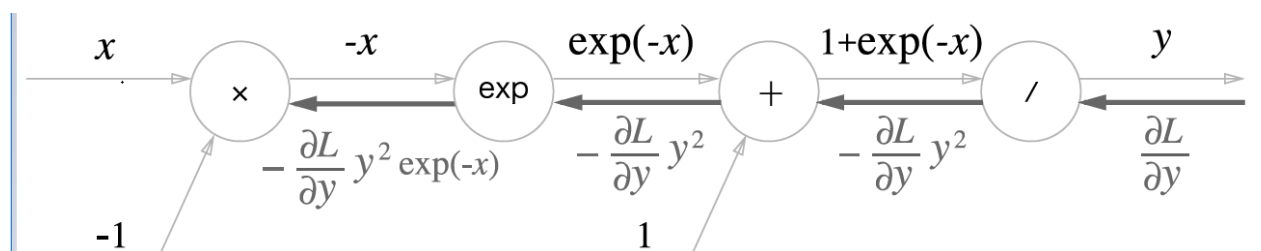
```
y = ( 1/( 1+ np.exp(-x) ) )
```

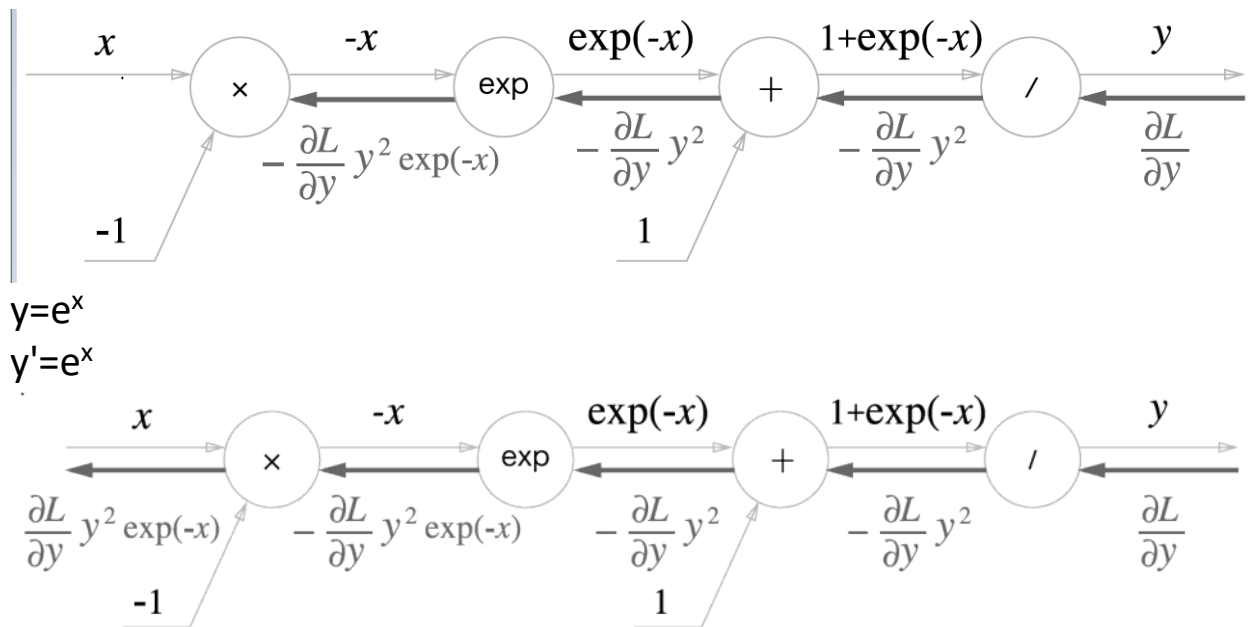


$$y = \frac{1}{x}$$
$$y' = -\frac{1}{x^2} = -y^2$$



덧셈 노드라 그대로





문제143. Sigmoid클래스를 생성하시오.

답)

```

class Sigmoid:
    def __init__(self):
        self.out=None

    def forward(self,x):
        out=1/(1+np.exp(-x))
        self.out=out
        return out

    def backward(self,dout):
        dx=dout*self.out*(1-self.out)
        return dx

```

문제146. 아래의 행렬을 입력받아 결과를 출력하는 forward 함수를 생성하시오.

보기)

```

y = (x ⊙ w) + b
x=np.array([[1,2]])
w=np.array([[1,3,5],[2,4,6]])
b=np.array([[1,1,1]])

```

답)

```

import numpy as np

x=np.array([[1,2]])
w=np.array([[1,3,5],[2,4,6]])
b=np.array([[1,1,1]])

def forward(x,w,b):
    return np.dot(x,w)+b
print(forward(x,w,b))

```

문제147. 지금위의 코드는 mnist로 치면 딱 한장의 이미지만 입력되는 코드이다. 아래와 같

## 이 batch로 처리

할 수 있도록 2장의 이미지로 처리되는 코드로 forward로 구현하시오.

보기)

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

답)

```
import numpy as np
```

```
x=np.array([[1,2],[3,4]]) #2(2장)x2
w=np.array([[1,3,5],[2,4,6]]) #2x3
b=np.array([[1,1,1]])
```

```
def forward(x,w,b):
    return np.dot(x,w)+b
print(forward(x,w,b))
```

문제148. 문제147번에 대한 역전파 함수를 backward라는 이름으로 생성하시오.

보기)

```
x=np.array([[1,2],[3,4]])
w=np.array([[1,3,5],[2,4,6]])
b=np.array([[1,1,1]])
dy=np.array([[1,1,1],[2,2,2]])
```

답)

```
import numpy as np
```

```
x=np.array([[1,2],[3,4]]) #2(2장)x2
w=np.array([[1,3,5],[2,4,6]]) #2x3
b=np.array([[1,1,1]])
dy=np.array([[1,1,1],[2,2,2]])
```

```
def forward(x,w,b):
    return np.dot(x,w)+b
```

```
def backward(x,w,dy):
    dx = np.dot(dy,w.T)
    dw = np.dot(x.T,dy)
    print('입력값에 대한 역전파(dx): \n', dx)
    print('가중치에 대한 역전파(dw): \n', dw)
```

```
backward(x,w,dy)
```

```
입력값에 대한 역전파(dx):
[[ 9 12]
 [18 24]]
가중치에 대한 역전파(dw):
[[ 7  7  7]
 [10 10 10]]
```

문제149. 아래의 행렬의 열을 더한 결과를 아래와 같이 출력하시오.

```
보기)
b=np.array([[1,1,1],[2,2,2]])
```

```
답)
import numpy as np

b=np.array([[1,1,1],[2,2,2]])
db=np.sum(b,axis=0)
print(db)

[3 3 3]
```

**문제150. 아래의 backward함수에 바이어스에 대한 역전파를 출력하는 코드를 출력하시오.**

```
보기)
def backward(x,w,dy):
    dx = np.dot(dy,w.T)
    dw = np.dot(x.T,dy)
    db=?

    print('입력값에 대한 역전파(dx): \n', dx)
    print('가중치에 대한 역전파(dw): \n', dw)
    print('바이어스에 대한 역전파(db):\n', db)
```

```
답)
import numpy as np

x=np.array([[1,2],[3,4]]) #2(2장)x2
w=np.array([[1,3,5],[2,4,6]]) #2x3
b=np.array([[1,1,1]])
dy=np.array([[1,1,1],[2,2,2]])

def forward(x,w,b):
    return np.dot(x,w)+b

def backward(x,w,dy):
    dx = np.dot(dy,w.T)
    dw = np.dot(x.T,dy)
    db = np.sum(dy,axis=0)
    print('입력값에 대한 역전파(dx): \n', dx)
    print('가중치에 대한 역전파(dw): \n', dw)
    print('바이어스에 대한 역전파(db):\n', db)
```

```
backward(x,w,dy)
입력값에 대한 역전파(dx):
[[ 9 12]
 [18 24]]
가중치에 대한 역전파(dw):
[[ 7  7  7]
 [10 10 10]]
바이어스에 대한 역전파(db):
[3 3 3]
```



문제151. 책175p 아래의 나오는 affine 클래스를 구현하시오.

답)

```
class Affine:
    def __init__(self, W, b):
        self.W = W
        self.b = b
        self.x = None
        self.dW = None
        self.db = None

    def forward(self,x):
        self.x = x
        y = np.dot(x, self.W)+self.b
        return y

    def backward(self,dy):
        dx=np.dot(dy, self.W.T)
        self.dW=np.dot(self.x.T, dy)
        self.db=np.sum(dy, axis=0)
        return dx

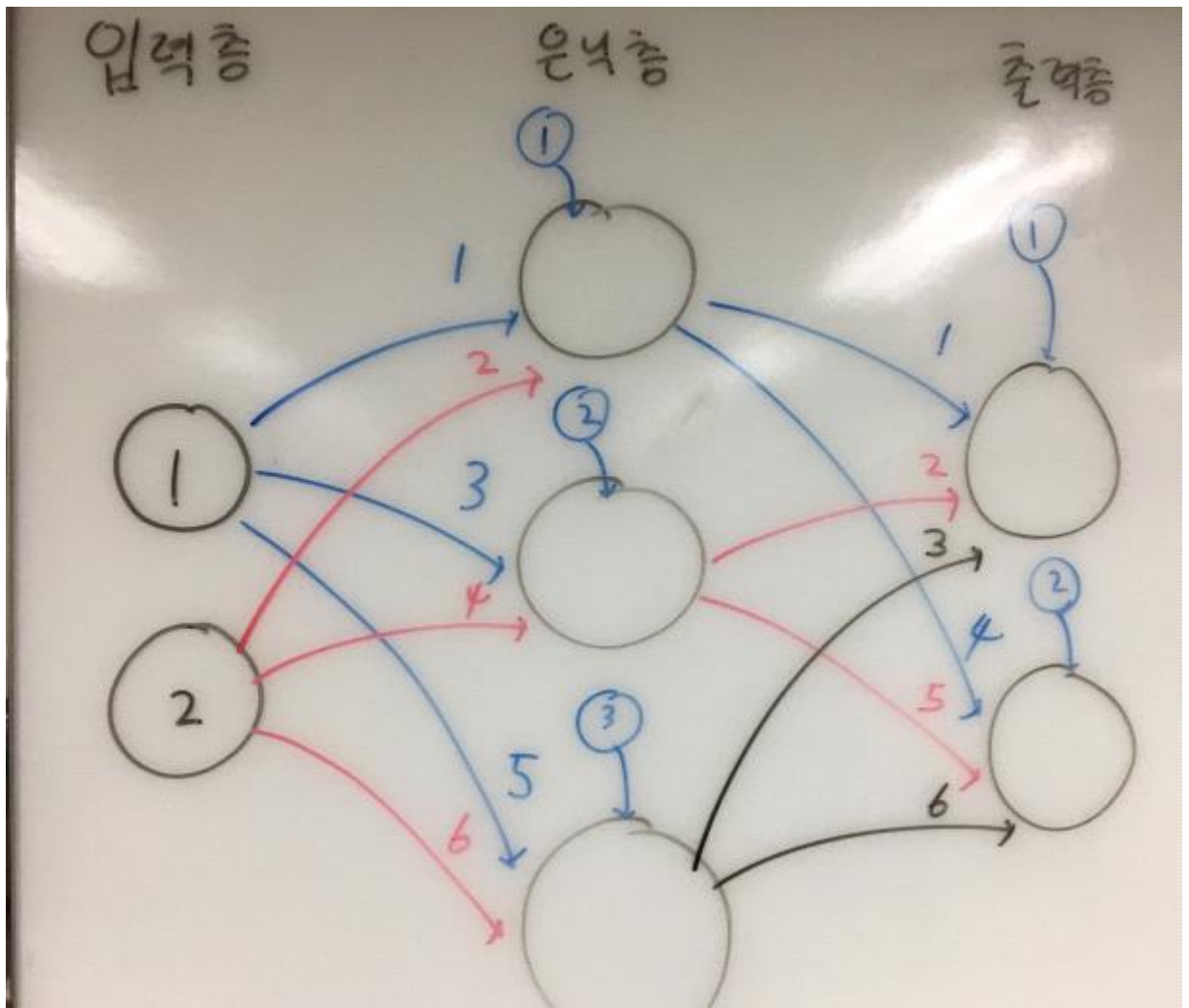
x=np.array([[1,2],[3,4]]) #2(2장)x2
w=np.array([[1,3,5],[2,4,6]]) #2x3
b=np.array([[1,1,1]])
dy=np.array([[1,1,1],[2,2,2]])

affine1=Affine(w,b)
print(affine1.forward(x))
print(affine1.backward(dy))

[[ 6 12 18]
 [12 26 40]]
[[ 9 12]
 [18 24]]
```

문제152. 아래의 그림의 신경망의 x,w1,w2,b1,b2의 numpy배열을 만드시오.

보기)



답)

```
import numpy as np
```

```
x = np.array([[1,2]])
```

```
w1= np.array([[1,3,5],[2,4,6]])
```

```
w2=np.array([[1,4],[2,5],[3,6]])
```

```
b1= np.array([[1,2,3]])
```

```
b2=np.array([[1,2]])
```

문제153. Affine클래스를 이용해서 2층 신경망의 순전파를 구현하시오.

답)

```
import numpy as np
```

```
x = np.array([[1,2]])
```

```
w1= np.array([[1,3,5],[2,4,6]])
```

```
w2=np.array([[1,4],[2,5],[3,6]])
```

```
b1= np.array([[1,2,3]])
```

```
b2=np.array([[1,2]])
```

```
class Affine:
```

```
    def __init__(self, W, b):
```

```
        self.W = W
```

```
        self.b = b
```

```
        self.x = None
```

```
        self.dW = None
```

```

self.db = None

def forward(self,x):
    self.x = x
    y = np.dot(x, self.W)+self.b
    return y

def backward(self,dy):
    dx=np.dot(dy, self.W.T)
    self.dW=np.dot(self.x.T, dy)
    self.db=np.sum(dy, axis=0)
    return dx

layer_1=Affine(w1,b1)
layer_2=Affine(w2,b2)

a1=layer_1.forward(x)
a2=layer_2.forward(a1)
print(a2)

```

[[ 93 211]]

**문제154.** 위의 2층 신경망의 역전파를 Affine 클래스를 이용해서 구현하시오. 아래 backward에 입력할 dy

는 위의 순전파의 결과인 a2행렬을 그대로 사용하시오.

결과)

```

dx : [ [12385 16108] ]
dw1 : [ [937 1241 1545
        1874 2482 3090] ]
dw2 : [ [937 1241 1545] ]

```

답)

```

import numpy as np

x = np.array([[1,2]])
w1= np.array([[1,3,5],[2,4,6]])
w2=np.array([[1,4],[2,5],[3,6]])
b1= np.array([1,2,3])
b2=np.array([1,2])

class Affine:
    def __init__(self, W, b):
        self.W = W
        self.b = b
        self.x = None
        self.dW = None
        self.db = None

    def forward(self,x):
        self.x = x
        y = np.dot(x, self.W)+self.b
        return y

    def backward(self,dy):
        dx=np.dot(dy, self.W.T)

```

```

        dW=np.dot(self.x.T, dy)
        db=np.sum(dy, axis=0)
        return dx, dW, db

layer_1=Affine(w1,b1)
layer_2=Affine(w2,b2)

a1=layer_1.forward(x)
a2=layer_2.forward(a1)
print(a2)

dx2, dw2, db2 = layer_2.backward(a2)

dx1, dw1,db1 = layer_1.backward(dx2)

print(dx1)
print(dw1)
print(db1)
[[ 93 211]]
[[12385 16108]]
[[ 937 1241 1545]
 [1874 2482 3090]]
[ 937 1241 1545]

```

문제155. 위의 2층 신경망의 순전파를 은닉층에 활성화 함수로 Relu를 추가해서 구현하시오.

(Relu클래스를 가져와서)

답)

```

import numpy as np

x = np.array([[1,2]])
w1= np.array([[1,3,5],[2,4,6]])
w2=np.array([[1,4],[2,5],[3,6]])
b1= np.array([[1,2,3]])
b2=np.array([[1,2]])

class Affine:
    def __init__(self, W, b):
        self.W = W
        self.b = b
        self.x = None
        self.dW = None
        self.db = None

    def forward(self,x):
        self.x = x
        y = np.dot(x, self.W)+self.b
        return y

    def backward(self,dy):
        dx=np.dot(dy, self.W.T)
        dW=np.dot(self.x.T, dy)
        db=np.sum(dy, axis=0)
        return dx, dW, db

```

```

class Relu:
    import numpy as np
    import copy
    def __init__(self):
        self.mask=None

    def forward(self,x):
        self.mask=(x<=0)
        out=x.copy()
        out[self.mask]=0
        return out

    def backward(self,dout):
        dout[self.mask]=0
        dx=dout
        return dx

layer_1=Affine(w1,b1)
layer_2=Affine(w2,b2)
relu=Relu()

a1=layer_1.forward(x)
relu_1=relu.forward(a1)
a2=layer_2.forward(relu_1)
relu_2=relu.forward(a2)

print(relu_2)
[[ 93 211]]

```

**문제156. 책179p에 나오는 class softmaxWithLoss 클래스를 생성하시오.**

답)

```

class SMWL:
    def __init__(self):
        self.loss = None # 손실
        self.y = None #softmax의 출력
        self.t = None #정답 레이블(원-핫 벡터)

    def forward(self,x,t):
        self.t=t
        self.y=softmax(x)
        self.loss = CEE(self.y, self.t)
        return self.loss

    def backward(self, dout=1):
        batch_size=self.t.shape[0]
        dx = (self.y-self.t)/batch_size
        return dx

```

**문제157. 위에서만든 SMWL클래스를 객체화 시켜서 아래의 x(입력값), t(target value)를 입력해서 순전파  
의오차율을 출력하고 역전파도 출력하시오.**

```

보기)
t=np.array([0,0,1,0,0,0,0,0,0])
x=np.array([0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.05, 0.3, 0.1, 0.5])
x2=np.array([0.01, 0.01, 0.9, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.02])

답)
import numpy as np

def CEE(y,t):
    delta=1e-7
    return -np.sum(t*np.log(y+delta))

def softmax(x):
    c = np.max(x)
    return np.exp(x-c)/np.sum(np.exp(x-c))

class SMWL:
    def __init__(self):
        self.loss = None # 손실
        self.y = None #softmax의 출력
        self.t = None #정답 레이블(원-핫 벡터)

    def forward(self,x,t):
        self.t=t
        self.y=softmax(x)
        self.loss = CEE(self.y, self.t)
        return self.loss

    def backward(self, dout=1):
        batch_size=self.t.shape[0]
        dx = (self.y-self.t)/batch_size
        return dx

t=np.array([0,0,1,0,0,0,0,0,0])
x=np.array([0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.05, 0.3, 0.1, 0.5])
x2=np.array([0.01, 0.01, 0.9, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.02])

smwl1=SMWL()
smwl2=SMWL()

print('순전파 오차율:', smwl1.forward(x,t))
print('역전파 오차율:', smwl1.backward())

print('순전파 오차율:', smwl1.forward(x2,t))
print('역전파 오차율:', smwl1.backward())
순전파 오차율: 2.4072798663898216
역전파 오차율: [ 0.00900598  0.00900598 -0.09099402  0.00900598  0.00900598  0.00900598
 0.00937352  0.01203584  0.00985412  0.01470061]
순전파 오차율: 1.5475681948007376
역전파 오차율: [ 0.0087373  0.0087373 -0.07872354  0.0087373  0.0087373  0.0087373
 0.0087373  0.0087373  0.0087373  0.00825111]

```

## OrderDict() 함수의 이해

"orderDict은 그냥 dictionary와는 틀리게 입력된 데이터 뿐만 아니라 입력된 순서까지 같아야 동일한 것으로 판단한다."

예제:

```
import collections
print('dict:')
```

```
d1={}
d1['a']="A"
d1['b']="B"
d1['c']="C"
d1['d']="D"
d1['e']="E"
```

```
d2={}
d2['e']="E"
d2['d']="D"
d2['c']="C"
d2['b']="B"
d2['a']="A"
```

```
print(d1)
print(d2)
```

```
print(d1==d2)#딕셔너리는 순서가 달라고 key와 value만 같아도 T
```

```
dict :
{'a': 'A', 'b': 'B', 'c': 'C', 'd': 'D', 'e': 'E'}
{'e': 'E', 'd': 'D', 'c': 'C', 'b': 'B', 'a': 'A'}
True
```

```
import collections
print('OrderedDict :')
d1=collections.OrderedDict()
d1['a']="A"
d1['b']="B"
d1['c']="C"
d1['d']="D"
d1['e']="E"
```

```
d2=collections.OrderedDict()
d2['e']="E"
d2['d']="D"
d2['c']="C"
d2['b']="B"
d2['a']="A"
```

```
print(d1)
print(d2)
```

```
print(d1==d2)# OrderedDict() 함수는 key와 value값이 같지만 순서가 다르면 F
```

```
OrderedDict :
OrderedDict([('a', 'A'), ('b', 'B'), ('c', 'C'), ('d', 'D'), ('e', 'E')])
OrderedDict([('e', 'E'), ('d', 'D'), ('c', 'C'), ('b', 'B'), ('a', 'A')])
```

```
OrderedDict :
OrderedDict([('a', 'A'), ('b', 'B'), ('c', 'C'), ('d', 'D'), ('e', 'E')])
OrderedDict([('e', 'E'), ('d', 'D'), ('c', 'C'), ('b', 'B'), ('a', 'A')])
False
```

순전파 순서의 반대로 역전파가 되어야하기 때문에 `OrderedDict()` 함수를 사용해야 한다.

**순전파 :**

입력값 --> Affine1 계층 --> 시그모이드 --> Affine2 계층 --> 소프트맥스

**역전파:**

소프트맥스 --> Affine2 계층 --> 시그모이드 --> Affine1 계층

## 오차 역전파를 이용한 2층 신경망 전체 코드

**클래스 이름 : TwoLayerNet**

1. 가중치와 바이어스를 초기화 하는 함수(`__init__`)
2. 순전파를 진행하는 함수(`predict`)
3. 비용(오차)를 출력하는 함수(`loss`)
4. 정확도를 출력하는 함수(`accuracy`)
5. 오차 역전파를 진행하는 함수(`gradient`)

**문제158. 책 181~183p에 TwoLayerNet 클래스를 생성하시오.**

답)

```
import sys, os
sys.path.append(os.pardir)
import numpy as np

from common.layers import * #common에 나머지 클래스들이 들어있다.
from common.gradient import numerical_gradient
from collections import OrderedDict

class TwoLayerNet:
    def __init__(self, input_size, hidden_size, output_size, weight_init_std=0.01):
        #가중치 초기화
        self.params={}
        self.params['W1']=weight_init_std * np.random.randn(input_size, hidden_size)
        self.params['b1']=np.zeros(hidden_size)
        self.params['W2']=weight_init_std * np.random.randn(hidden_size, output_size)
        self.params['b2']=np.zeros(output_size)

        #계층생성
        self.layers=OrderedDict()
        self.layers['Affine1']=Affine(self.params['W1'], self.params['b1'])
        self.layers['Relu1']=Relu()
        self.layers['Affine2']=Affine(self.params['W2'], self.params['b2'])

        self.lastLayer=SMWL()
```



```

def predict(self,x):
    for layer in self.layers.values():
        x=layer.forward(x)
    return x

#x: 입력데이터, t:정답 레이블
def loss(self,x,t):
    y=self.predict(x)
    return self.lastLayer.forward(y,t)

def accuracy(self,x,t):
    y=self.predict(x)
    y=np.argmax(y, axis=1)
    if t.ndim != 1: t = np.argmax(t,axis=1)

    accuracy=np.sum(y==t) / float(x.shape[0])
    return accuracy

#x: 입력데이터, t:정답 레이블
def numerical_gradient(self, x, t):
    loss_W=lambda W:self.loss(x,t)

    grads={}
    grads['W1']=numerical_gradient(loss_W, self.params['W1'])
    grads['b1']=numerical_gradient(loss_W, self.params['b1'])
    grads['W2']=numerical_gradient(loss_W, self.params['W2'])
    grads['b2']=numerical_gradient(loss_W, self.params['b2'])
    return grads

def gradient(self, x, t):
    #순전파
    self.loss(x,t)

    #역전파
    dout=1
    dout=self.lastLayer.backward(dout)

    layers=list(self.layers.values())
    layers.reverse()
    for layer in layers:
        dout=layer.backward(dout)

    #결과 저장
    grads={}
    grads['W1']=self.layers['Affine1'].dW
    grads['b1']=self.layers['Affine1'].db
    grads['W2']=self.layers['Affine1'].dW
    grads['b2']=self.layers['Affine1'].db

    return grads

tolayernet1=TwoLayerNet(input_size=2, hidden_size=50, output_size=2, weight_init_std=
0.02)
x = np.array([[1,2]])

```

```

t= np.array([[0,1]])
print(tolayernet1.predict(x))
print(tolayernet1.gradient(x,t))

```

**문제159.** 위의 2층 신경망 클래스를 mnist 데이터로 구현하는 최종 코드를 책186p를 보고 작성하시오.

```

답)
# coding: utf-8
import sys, os

sys.path.append(os.pardir) # 부모 디렉터리의 파일을 가져올 수 있도록 설정
import numpy as np
from common.layers import *
from common.gradient import numerical_gradient
from collections import OrderedDict
import matplotlib.pyplot as plt
from dataset.mnist import load_mnist

class TwoLayerNet:
    def __init__(self, input_size, hidden_size, output_size, weight_init_std=0.01):
        # 가중치 초기화
        self.params = {}
        self.params['W1'] = weight_init_std * np.random.randn(input_size, hidden_size)
        self.params['b1'] = np.zeros(hidden_size)
        self.params['W2'] = weight_init_std * np.random.randn(hidden_size, output_size)
        self.params['b2'] = np.zeros(output_size)

        # 계층 생성
        self.layers = OrderedDict()
        self.layers['Affine1'] = Affine(self.params['W1'], self.params['b1'])
        self.layers['Relu1'] = Relu()
        self.layers['Affine2'] = Affine(self.params['W2'], self.params['b2'])

        self.lastLayer = SoftmaxWithLoss()

    def predict(self, x):
        for layer in self.layers.values():
            x = layer.forward(x)

        return x

# x : 입력 데이터, t : 정답 레이블
def loss(self, x, t):
    y = self.predict(x)
    return self.lastLayer.forward(y, t)

def accuracy(self, x, t):
    y = self.predict(x)
    y = np.argmax(y, axis=1)
    if t.ndim != 1: t = np.argmax(t, axis=1)

    accuracy = np.sum(y == t) / float(x.shape[0])
    return accuracy

```

```

# x : 입력 데이터, t : 정답 레이블
def numerical_gradient(self, x, t):
    loss_W = lambda W: self.loss(x, t)

    grads = {}
    grads['W1'] = numerical_gradient(loss_W, self.params['W1'])
    grads['b1'] = numerical_gradient(loss_W, self.params['b1'])
    grads['W2'] = numerical_gradient(loss_W, self.params['W2'])
    grads['b2'] = numerical_gradient(loss_W, self.params['b2'])

    return grads

def gradient(self, x, t):
    # forward
    self.loss(x, t)

    # backward
    dout = 1
    dout = self.lastLayer.backward(dout)

    layers = list(self.layers.values())
    layers.reverse()
    for layer in layers:
        dout = layer.backward(dout)

    # 결과 저장
    grads = {}
    grads['W1'], grads['b1'] = self.layers['Affine1'].dW, self.layers['Affine1'].db
    grads['W2'], grads['b2'] = self.layers['Affine2'].dW, self.layers['Affine2'].db

    return grads

# 데이터 읽기
(x_train, t_train), (x_test, t_test) = load_mnist(normalize=True, one_hot_label=True)
network = TwoLayerNet(input_size=784, hidden_size=50, output_size=10)

# 하이퍼파라미터
iters_num = 10000 # 반복 횟수를 적절히 설정한다.
train_size = x_train.shape[0] # 60000 개
batch_size = 100 # 미니배치 크기
learning_rate = 0.1

train_loss_list = []
train_acc_list = []
test_acc_list = []

# 1에폭당 반복 수
iter_per_epoch = max(train_size / batch_size, 1)
print(iter_per_epoch) # 600

for i in range(iters_num): # 10000
    # 미니배치 획득 # 랜덤으로 100개씩 뽑아서 10000번을 수행하니까 백만번
    batch_mask = np.random.choice(train_size, batch_size) # 100개 씩 뽑아서 10000번 백만

```

번

```
x_batch = x_train[batch_mask]
t_batch = t_train[batch_mask]

# 기울기 계산
#grad = network.numerical_gradient(x_batch, t_batch)
grad = network.gradient(x_batch, t_batch)

# 매개변수 갱신
for key in ('W1', 'b1', 'W2', 'b2'):
    network.params[key] -= learning_rate * grad[key]

# 학습 경과 기록
loss = network.loss(x_batch, t_batch)
train_loss_list.append(loss) # cost 가 점점 줄어드는것을 보려고

# 1에폭당 정확도 계산 # 여기는 훈련이 아니라 1에폭 되었을때 정확도만 체크
if i % iter_per_epoch == 0: # 600 번마다 정확도 쌓는다.
    print(x_train.shape) # 60000,784
    train_acc = network.accuracy(x_train, t_train)
    test_acc = network.accuracy(x_test, t_test)
    train_acc_list.append(train_acc) # 10000/600 개 16개 # 정확도가 점점 올라감
    test_acc_list.append(test_acc) # 10000/600 개 16개 # 정확도가 점점 올라감
    print("train acc, test acc | " + str(train_acc) + ", " + str(test_acc))

# 그래프 그리기
markers = {'train': 'o', 'test': 's'}
x = np.arange(len(train_acc_list))
plt.plot(x, train_acc_list, label='train acc')
plt.plot(x, test_acc_list, label='test acc', linestyle='--')
plt.xlabel("epochs")
plt.ylabel("accuracy")
plt.ylim(0, 1.0)
plt.legend(loc='lower right')
plt.show()
```

**문제160. 위의 신경망을 3층 신경망으로 변경하시오.**

답)

```
# coding: utf-8
import sys, os

sys.path.append(os.pardir) # 부모 디렉터리의 파일을 가져올 수 있도록 설정
import numpy as np
from common.layers import *
from common.gradient import numerical_gradient
from collections import OrderedDict
import matplotlib.pyplot as plt
from dataset.mnist import load_mnist

class TwoLayerNet:
    def __init__(self, input_size, hidden_size, hidden_size2, output_size, weight_init_std=0.01):
```

```

# 가중치 초기화
self.params = {}
self.params['W1'] = weight_init_std * np.random.randn(input_size, hidden_size)
self.params['b1'] = np.zeros(hidden_size)
self.params['W2'] = weight_init_std * np.random.randn(hidden_size, hidden_size2)
self.params['b2'] = np.zeros(hidden_size2)
self.params['W3'] = weight_init_std * np.random.randn(hidden_size2, output_size)
self.params['b3'] = np.zeros(output_size)

# 계층 생성
self.layers = OrderedDict()
self.layers['Affine1'] = Affine(self.params['W1'], self.params['b1'])
self.layers['Relu1'] = Relu()
self.layers['Affine2'] = Affine(self.params['W2'], self.params['b2'])
self.layers['Relu2'] = Relu()
self.layers['Affine3'] = Affine(self.params['W3'], self.params['b3'])

self.lastLayer = SoftmaxWithLoss()

def predict(self, x):
    for layer in self.layers.values():
        x = layer.forward(x)

    return x

# x : 입력 데이터, t : 정답 레이블
def loss(self, x, t):
    y = self.predict(x)
    return self.lastLayer.forward(y, t)

def accuracy(self, x, t):
    y = self.predict(x)
    y = np.argmax(y, axis=1)
    if t.ndim != 1: t = np.argmax(t, axis=1)

    accuracy = np.sum(y == t) / float(x.shape[0])
    return accuracy

# x : 입력 데이터, t : 정답 레이블
def numerical_gradient(self, x, t):
    loss_W = lambda W: self.loss(x, t)

    grads = {}
    grads['W1'] = numerical_gradient(loss_W, self.params['W1'])
    grads['b1'] = numerical_gradient(loss_W, self.params['b1'])
    grads['W2'] = numerical_gradient(loss_W, self.params['W2'])
    grads['b2'] = numerical_gradient(loss_W, self.params['b2'])
    grads['W3'] = numerical_gradient(loss_W, self.params['W3'])
    grads['b3'] = numerical_gradient(loss_W, self.params['b3'])
    return grads

def gradient(self, x, t):
    # forward
    self.loss(x, t)

```

```

# backward
dout = 1
dout = self.lastLayer.backward(dout)

layers = list(self.layers.values())
layers.reverse()
for layer in layers:
    dout = layer.backward(dout)

# 결과 저장
grads = {}
grads['W1'], grads['b1'] = self.layers['Affine1'].dW, self.layers['Affine1'].db
grads['W2'], grads['b2'] = self.layers['Affine2'].dW, self.layers['Affine2'].db
grads['W3'], grads['b3'] = self.layers['Affine3'].dW, self.layers['Affine3'].db
return grads

# 데이터 읽기
(x_train, t_train), (x_test, t_test) = load_mnist(normalize=True, one_hot_label=True)
network = TwoLayerNet(input_size=784, hidden_size=50, hidden_size2=100, output_size=10)

# 하이퍼파라미터
iters_num = 10000 # 반복 횟수를 적절히 설정한다.
train_size = x_train.shape[0] # 60000 개
batch_size = 100 # 미니배치 크기
learning_rate = 0.1

train_loss_list = []
train_acc_list = []
test_acc_list = []

# 1에폭당 반복 수
iter_per_epoch = max(train_size / batch_size, 1)
print(iter_per_epoch) # 600

for i in range(iters_num): # 10000
    # 미니배치 획득 # 랜덤으로 100개씩 뽑아서 10000번을 수행하니까 백만번
    batch_mask = np.random.choice(train_size, batch_size) # 100개 씩 뽑아서 10000번 백만
    번
    x_batch = x_train[batch_mask]
    t_batch = t_train[batch_mask]

    # 기울기 계산
    #grad = network.numerical_gradient(x_batch, t_batch)
    grad = network.gradient(x_batch, t_batch)

    # 매개변수 갱신
    for key in ('W1', 'b1', 'W2', 'b2', 'W3', 'b3'):
        network.params[key] -= learning_rate * grad[key]

# 학습 경과 기록
loss = network.loss(x_batch, t_batch)
train_loss_list.append(loss) # cost 가 점점 줄어드는것을 보려고

```

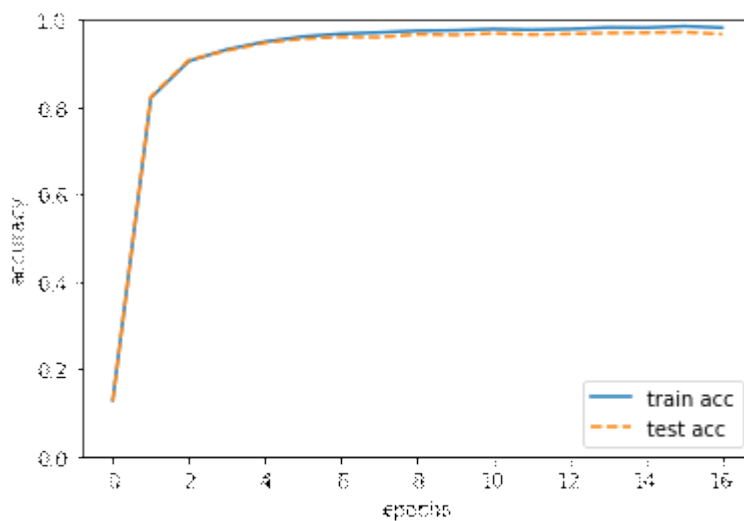
```

# 1에폭당 정확도 계산 # 여기는 훈련이 아니라 1에폭 되었을때 정확도만 체크
if i % iter_per_epoch == 0: # 600 번마다 정확도 쌓는다.
    print(x_train.shape) # 60000,784
    train_acc = network.accuracy(x_train, t_train)
    test_acc = network.accuracy(x_test, t_test)

    train_acc_list.append(train_acc) # 10000/600 개 16개 # 정확도가 점점 올라감
    test_acc_list.append(test_acc) # 10000/600 개 16개 # 정확도가 점점 올라감
    print("train acc, test acc | " + str(train_acc) + ", " + str(test_acc))

# 그래프 그리기
markers = {'train': 'o', 'test': 's'}
x = np.arange(len(train_acc_list))
plt.plot(x, train_acc_list, label='train acc')
plt.plot(x, test_acc_list, label='test acc', linestyle='--')
plt.xlabel("epochs")
plt.ylabel("accuracy")
plt.ylim(0, 1.0)
plt.legend(loc='lower right')
plt.show()
train acc, test acc | 0.9845, 0.9721

```



**문제161.** 위의 3층 신경망의 정확도가 활성화 함수가 Relu일때와 활성화 함수가 sigmoid일  
때의 차이를 테

스트 하시오.

보기)

3층 Relu : 98, 97

답)

class TwoLayerNet:

```

def __init__(self, input_size, hidden_size, hidden_size2, output_size, weight_init_std=0.01):
    # 가중치 초기화
    self.params = {}
    self.params['W1'] = weight_init_std * np.random.randn(input_size, hidden_size)
    self.params['b1'] = np.zeros(hidden_size)
    self.params['W2'] = weight_init_std * np.random.randn(hidden_size, hidden_size2)

```

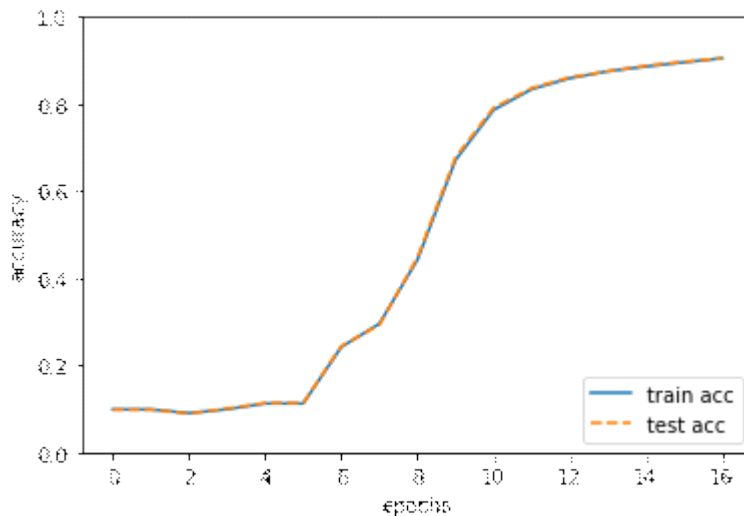
```

self.params['b2'] = np.zeros(hidden_size2)
self.params['W3'] = weight_init_std * np.random.randn(hidden_size2, output_size)
self.params['b3'] = np.zeros(output_size)

# 계층 생성
self.layers = OrderedDict()
self.layers['Affine1'] = Affine(self.params['W1'], self.params['b1'])
self.layers['Sigmoid1'] = Sigmoid()
self.layers['Affine2'] = Affine(self.params['W2'], self.params['b2'])
self.layers['Sigmoid2'] = Sigmoid()
self.layers['Affine3'] = Affine(self.params['W3'], self.params['b3'])

self.lastLayer = SoftmaxWithLoss()
train acc, test acc | 0.9048, 0.9048

```



문제162. 다시 Relu로 바꾸고 노드수를 늘리면 정확도가 올라가는지 확인하시오.

보기)

기존은 784, 50, 100, 10 이다.

입력층 --> 은닉1층 --> 은닉2층 --> 출력층

784            100            100            10

으로 변경

답)

# 데이터 읽기

```

(x_train, t_train), (x_test, t_test) = load_mnist(normalize=True, one_hot_label=True)
network = TwoLayerNet(input_size=784, hidden_size=100, hidden_size2=100, output_size=10)

```

```

train acc, test acc | 0.9906833333333333, 0.9745

```

-----6장

문제166. Adam클래스를 생성하고 위의 3층 신경망에 Adam경사 감소법을 적용해서 3층 신경망을 학습



시킵시오.

답)

# coding: utf-8

import sys, os

sys.path.append(os.pardir) # 부모 디렉터리의 파일을 가져올 수 있도록 설정

import numpy as np

from common.layers import \*

from common.gradient import numerical\_gradient

from collections import OrderedDict

import matplotlib.pyplot as plt

from dataset.mnist import load\_mnist

class TwoLayerNet:

def \_\_init\_\_(self, input\_size, hidden\_size, hidden\_size2, output\_size, weight\_init\_std=0.01):

# 가중치 초기화

self.params = {}

self.params['W1'] = weight\_init\_std \* np.random.randn(input\_size, hidden\_size)

self.params['b1'] = np.zeros(hidden\_size)

self.params['W2'] = weight\_init\_std \* np.random.randn(hidden\_size, hidden\_size2)

self.params['b2'] = np.zeros(hidden\_size2)

self.params['W3'] = weight\_init\_std \* np.random.randn(hidden\_size2, output\_size)

self.params['b3'] = np.zeros(output\_size)

# 계층 생성

self.layers = OrderedDict()

self.layers['Affine1'] = Affine(self.params['W1'], self.params['b1'])

self.layers['Relu1'] = Relu()

self.layers['Affine2'] = Affine(self.params['W2'], self.params['b2'])

self.layers['Relu2'] = Relu()

self.layers['Affine3'] = Affine(self.params['W3'], self.params['b3'])

self.lastLayer = SoftmaxWithLoss()

def predict(self, x):

for layer in self.layers.values():

x = layer.forward(x)

return x

# x : 입력 데이터, t : 정답 레이블

def loss(self, x, t):

y = self.predict(x)

return self.lastLayer.forward(y, t)

def accuracy(self, x, t):

y = self.predict(x)

y = np.argmax(y, axis=1)

if t.ndim != 1: t = np.argmax(t, axis=1)

accuracy = np.sum(y == t) / float(x.shape[0])

return accuracy

# x : 입력 데이터, t : 정답 레이블

```

def numerical_gradient(self, x, t):
    loss_W = lambda W: self.loss(x, t)

    grads = {}
    grads['W1'] = numerical_gradient(loss_W, self.params['W1'])
    grads['b1'] = numerical_gradient(loss_W, self.params['b1'])
    grads['W2'] = numerical_gradient(loss_W, self.params['W2'])
    grads['b2'] = numerical_gradient(loss_W, self.params['b2'])
    grads['W3'] = numerical_gradient(loss_W, self.params['W3'])
    grads['b3'] = numerical_gradient(loss_W, self.params['b3'])
    return grads

def gradient(self, x, t):
    # forward
    self.loss(x, t)

    # backward
    dout = 1
    dout = self.lastLayer.backward(dout)

    layers = list(self.layers.values())
    layers.reverse()
    for layer in layers:
        dout = layer.backward(dout)

    # 결과 저장
    grads = {}
    grads['W1'], grads['b1'] = self.layers['Affine1'].dW, self.layers['Affine1'].db
    grads['W2'], grads['b2'] = self.layers['Affine2'].dW, self.layers['Affine2'].db
    grads['W3'], grads['b3'] = self.layers['Affine3'].dW, self.layers['Affine3'].db
    return grads

class Adam:
    def __init__(self, lr=0.001, beta1=0.9, beta2=0.999):
        self.lr = lr
        self.beta1 = beta1
        self.beta2 = beta2
        self.iter = 0
        self.m = None
        self.v = None

    def update(self, params, grads):
        if self.m is None:
            self.m, self.v = {}, {}
            for key, val in params.items():
                self.m[key] = np.zeros_like(val)
                self.v[key] = np.zeros_like(val)

        self.iter += 1
        lr_t = self.lr * np.sqrt(1.0 - self.beta2 ** self.iter) / (1.0 - self.beta1 ** self.iter)

        for key in params.keys():

            self.m[key] += (1 - self.beta1) * (grads[key] - self.m[key])
            self.v[key] += (1 - self.beta2) * (grads[key] ** 2 - self.v[key])

```

```

        params[key] -= lr_t * self.m[key] / (np.sqrt(self.v[key]) + 1e-7)

optimizer=Adam()
# 데이터 읽기
(x_train, t_train), (x_test, t_test) = load_mnist(normalize=True, one_hot_label=True)
network = TwoLayerNet(input_size=784, hidden_size=50, hidden_size2=100, output_size=10)

# 하이퍼파라미터
iters_num = 10000 # 반복 횟수를 적절히 설정한다.
train_size = x_train.shape[0] # 60000 개
batch_size = 100 # 미니배치 크기
learning_rate = 0.1

train_loss_list = []
train_acc_list = []
test_acc_list = []

# 1에폭당 반복 수
iter_per_epoch = max(train_size / batch_size, 1)
print(iter_per_epoch) # 600

for i in range(iters_num):
    batch_mask = np.random.choice(train_size, batch_size)
    x_batch = x_train[batch_mask]
    t_batch = t_train[batch_mask]

    for key in ('W1', 'b1', 'W2', 'b2', 'W3', 'b3'):
        grads = network.gradient(x_batch, t_batch)
        params = network.params
        optimizer.update(params, grads)

    loss = network.loss(x_batch, t_batch)
    train_loss_list.append(loss)

    if i % iter_per_epoch == 0:
        train_acc = network.accuracy(x_train, t_train)
        test_acc = network.accuracy(x_test, t_test)
        train_acc_list.append(train_acc)
        test_acc_list.append(test_acc)
        print(train_acc, test_acc)

```

## • 가중치 초기화 값 설정

" 가중치 초기값을 적절히 설정하면 각 층의 활성화 값의 분포가 적당히 퍼지는 효과가 발생한다.

적당히 퍼지게 되면 학습이 잘되고 정확도가 높아진다. "

가중치 초기값을 선정하는 방법 5가지

### 1. 가중치 초기값을 0으로 선정 -----> "학습 안됨"

예제)

문제167. 지금까지 구현 3층 신경망의 가중치 초기값을 0 으로 설정하고 신경망을 학습 시켜 보시오.

답)

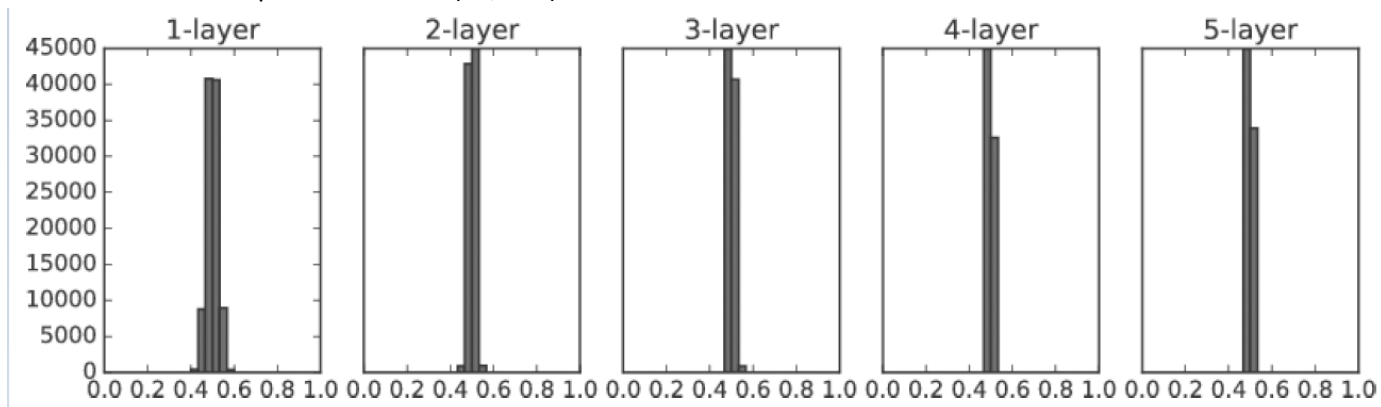
```
class TwoLayerNet:
    def __init__(self, input_size, hidden_size, hidden_size2, output_size, weight_init_std=0):
        # 가중치 초기화
        self.params = {}
        self.params['W1'] = weight_init_std * np.random.randn(input_size, hidden_size)
        self.params['b1'] = np.zeros(hidden_size)
        self.params['W2'] = weight_init_std * np.random.randn(hidden_size, hidden_size2)
        self.params['b2'] = np.zeros(hidden_size2)
        self.params['W3'] = weight_init_std * np.random.randn(hidden_size2, output_size)
        self.params['b3'] = np.zeros(output_size)
```

결과 : 안된다!!!!!!!

## 2. 표준편차가 1인 정규분포를 사용해 초기값을 선정

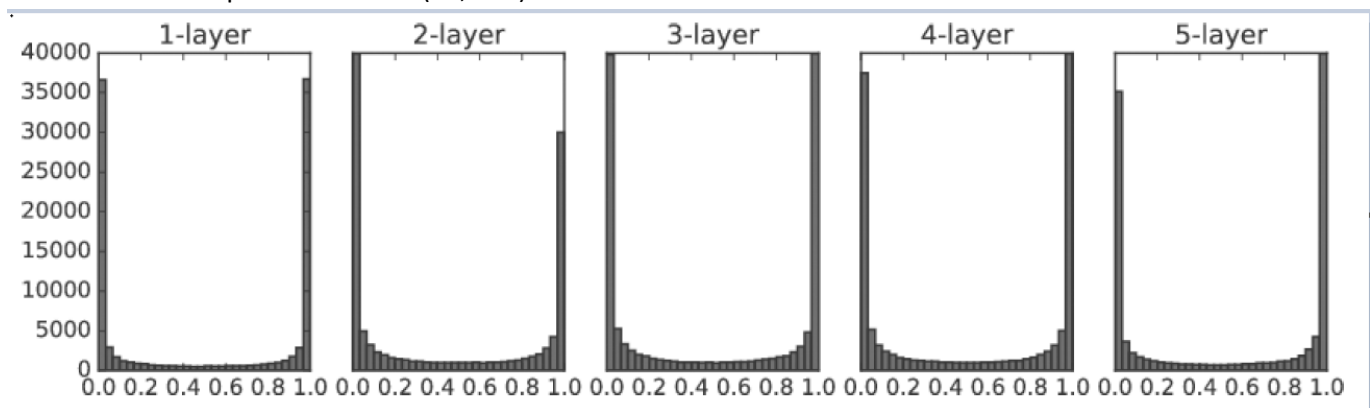
- 표준편차가 1인 정규분포를 사용해 초기값을 선정(시험문제가 쉬우면 학생들 점수가 평균에 가깝다)

구현예:  $0.01 * \text{np.random.randn}(10, 100)$



-표준편차가 클수록 data가 더 많이 흩어져 있다.(시험문제가 어려우면 아주 잘하는 학생들과 못하는 학생들로 나뉘진다)

구현예:  $1 * \text{np.random.randn}(10, 100)$



## 3. Xavier(사비에르) 초기값 설정

- 표준편차가  $\sqrt{\frac{1}{n}}$ 인 정규 분포로 초기화 한다. (n은 앞층의 노드수)

- sigmoid와 짝궁

문제168. 가중치 초기값을 Xavier로 설정하고 학습이 잘되는지 확인하시오.

보기)

```
weight_init_std= 1/np.sqrt(50)
```

답)

```
class TwoLayerNet:
```

```
    def __init__(self, input_size, hidden_size, hidden_size2, output_size, weight_init_std=1): #
```

여기서 사용안함

```
    # 가중치 초기화
```

```
    self.params = {}
```

```
    self.params['W1'] = np.sqrt(1/784) * np.random.randn(input_size, hidden_size)
```

```
    self.params['b1'] = np.zeros(hidden_size)
```

```
    self.params['W2'] = np.sqrt(1/50) * np.random.randn(hidden_size, hidden_size2)
```

```
    self.params['b2'] = np.zeros(hidden_size2)
```

```
    self.params['W3'] = np.sqrt(1/100) * np.random.randn(hidden_size2, output_size)
```

```
    self.params['b3'] = np.zeros(output_size)
```

```
    # 계층 생성
```

```
    self.layers = OrderedDict()
```

```
    self.layers['Affine1'] = Affine(self.params['W1'], self.params['b1'])
```

```
    self.layers['Relu1'] = Relu()
```

```
    self.layers['Affine2'] = Affine(self.params['W2'], self.params['b2'])
```

```
    self.layers['Relu2']=Relu()
```

```
    self.layers['Affine3'] = Affine(self.params['W3'], self.params['b3'])
```

```
    self.lastLayer = SoftmaxWithLoss()
```

#### 4. He 초기값 선정

- 표준편차가  $\frac{2}{\sqrt{n}}$ 인 정규 분포로 초기화 (n은 앞층의 노드수)

-Relu 와 짝궁

문제169. 지금까지 구현한 3층 신경망의 가중치 초기값을 He로 설정하고 정확도를 확인하시오.

보기)

```
weight_init_std=np.sqrt(2/50)
```

답)

```
class TwoLayerNet:
```

```
    def __init__(self, input_size, hidden_size, hidden_size2, output_size, weight_init_std=1): #
```

여기서 사용안함

```
    # 가중치 초기화
```

```
    self.params = {}
```

```
    self.params['W1'] = np.sqrt(2/784) * np.random.randn(input_size, hidden_size)
```

```
    self.params['b1'] = np.zeros(hidden_size)
```

```
    self.params['W2'] = np.sqrt(2/50) * np.random.randn(hidden_size, hidden_size2)
```

```

self.params['b2'] = np.zeros(hidden_size2)
self.params['W3'] = np.sqrt(2/100) * np.random.randn(hidden_size2, output_size)
self.params['b3'] = np.zeros(output_size)

# 계층 생성
self.layers = OrderedDict()
self.layers['Affine1'] = Affine(self.params['W1'], self.params['b1'])
self.layers['Relu1'] = Relu()
self.layers['Affine2'] = Affine(self.params['W2'], self.params['b2'])
self.layers['Relu2'] = Relu()
self.layers['Affine3'] = Affine(self.params['W3'], self.params['b3'])

self.lastLayer = SoftmaxWithLoss()

```

## 배치 정규화

" 앞에서는 가중치 초기값을 적절히 설정하면 각 층의 활성화 값의 분포가 적당히 퍼지는 효과를 보았다. "

적당히 퍼지게 되면 학습이 잘되고 정확도가 높아지는것을 확인했다.

그런데 배치정규화는 바로 각 층에서의 활성화 값이 적당히 분포 되도록 강제로 조정하는 것을 말한다.

### 그림(6장) 배치 정규화의 필요성

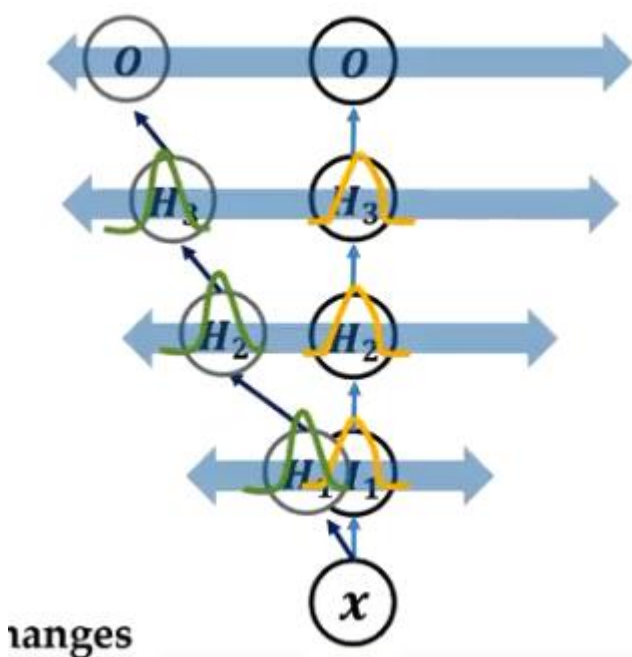
뉴럴 네트워크는 파라미터가 많기 때문에 학습이 어렵다.

딥러닝 같은 경우는 레이어가 많아서 학습이 어렵다.

### 이게 왜 학습이 어렵냐면?

가중치의 조그만 변화가 가중되어서 쌓이면 히든 레이어가 많아 질수록 출력되는 값의 변화가 크기

때문이다.



만약에 변화없이 히든 레이어2까지 안정된 가중치 값을 가지게 된다면 학습이 잘 될

것이다.

그런데 가중치의 영향을 계속 받아서 히든 레이어2의 값이 아주 다른값이 된다면 기존 값과는

다르기 때문에 어떻게 학습해야할지 모르게되는 상황이 발생하게 된다.

그래서 나온게 배치 정규화이다. 배치 정규화는 값이 활성화 함수를 통과하기전에 가중의 변화를 줄이는것이 목표이다.

가중의 합이 배치 정규화에 들어오게 되면 기존 값의 스케일을 줄여버리게 된다.

#계층 생성

```
self.layers = OrderedDict()
```

```
self.layers['Affine1'] = Affine(self.params['W1'], self.params['b1'])
self.layers['BatchNorm1'] = BatchNormalization(gamma=1.0, beta=0.)
self.layers['Relu1'] = Relu()
```

```
self.layers['Affine2'] = Affine(self.params['W2'], self.params['b2'])
self.layers['BatchNorm2'] = BatchNormalization(gamma=1.0, beta=0.)
self.layers['Relu2'] = Relu()
```

```
self.layers['Affine3'] = Affine(self.params['W3'], self.params['b3'])
self.lastLayer = SoftmaxWithLoss()
```

배치 정규화 식

**Input:** Values of  $x$  over a mini-batch:  $\mathcal{B} = \{x_{1...m}\};$

Parameters to be learned:  $\gamma, \beta$

**Output:**  $\{y_i = \text{BN}_{\gamma, \beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^m x_i \quad // \text{ mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2 \quad // \text{ mini-batch variance}$$

$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \quad // \text{ normalize}$$

$$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv \text{BN}_{\gamma, \beta}(x_i) \quad // \text{ scale and shift}$$

문제170. 배치 정규화 클래스를 우리 3층 신경망에 구현하고 각층마다 아래와 같이 배치 정규화층을 구현

하시오.

보기)

class BatchNormalization:

"""

<http://arxiv.org/abs/1502.03167>

"""

def \_\_init\_\_(self, gamma, beta, momentum=0.9, running\_mean=None, running\_var=None):

self.gamma = gamma

self.beta = beta

self.momentum = momentum

self.input\_shape = None # 합성곱 계층은 4차원, 완전연결 계층은 2차원

# 시험할 때 사용할 평균과 분산

self.running\_mean = running\_mean

self.running\_var = running\_var

# backward 시에 사용할 중간 데이터

self.batch\_size = None

self.xc = None

self.std = None

self.dgamma = None

self.dbeta = None

def forward(self, x, train\_flg=True):

self.input\_shape = x.shape

if x.ndim != 2:

N, C, H, W = x.shape

x = x.reshape(N, -1)

out = self.\_\_forward(x, train\_flg)

return out.reshape(\*self.input\_shape)

def \_\_forward(self, x, train\_flg):

if self.running\_mean is None:

N, D = x.shape

self.running\_mean = np.zeros(D)

self.running\_var = np.zeros(D)

if train\_flg:

mu = x.mean(axis=0)

xc = x - mu

var = np.mean(xc\*\*2, axis=0)

std = np.sqrt(var + 10e-7)

xn = xc / std

self.batch\_size = x.shape[0]

self.xc = xc

self.xn = xn

self.std = std

self.running\_mean = self.momentum \* self.running\_mean + (1-self.momentum) \* mu

self.running\_var = self.momentum \* self.running\_var + (1-self.momentum) \* var

else:

xc = x - self.running\_mean

xn = xc / ((np.sqrt(self.running\_var + 10e-7)))

out = self.gamma \* xn + self.beta



```

return out

def backward(self, dout):
    if dout.ndim != 2:
        N, C, H, W = dout.shape
        dout = dout.reshape(N, -1)

    dx = self.__backward(dout)

    dx = dx.reshape(*self.input_shape)
    return dx

def __backward(self, dout):
    dbeta = dout.sum(axis=0)
    dgamma = np.sum(self.xn * dout, axis=0)
    dxn = self.gamma * dout
    dxc = dxn / self.std
    dstd = -np.sum((dxn * self.xc) / (self.std * self.std), axis=0)
    dvar = 0.5 * dstd / self.std
    dxc += (2.0 / self.batch_size) * self.xc * dvar
    dm_u = np.sum(dxc, axis=0)
    dx = dxc - dm_u / self.batch_size

    self.dgamma = dgamma
    self.dbeta = dbeta

    return dx

```

답)

```

# coding: utf-8
import sys, os

```

```

sys.path.append(os.pardir) # 부모 디렉터리의 파일을 가져올 수 있도록 설정
import numpy as np
from common.layers import *
from common.gradient import numerical_gradient
from collections import OrderedDict
import matplotlib.pyplot as plt
from dataset.mnist import load_mnist

```

```

class TwoLayerNet:
    def __init__(self, input_size, hidden_size, hidden_size2, output_size, weight_init_std=1):
        # 가중치 초기화
        self.params = {}
        self.params['W1'] = np.sqrt(2/784) * np.random.randn(input_size, hidden_size)
        self.params['b1'] = np.zeros(hidden_size)
        self.params['W2'] = np.sqrt(2/50) * np.random.randn(hidden_size, hidden_size2)
        self.params['b2'] = np.zeros(hidden_size2)
        self.params['W3'] = np.sqrt(2/100) * np.random.randn(hidden_size2, output_size)
        self.params['b3'] = np.zeros(output_size)

        # 계층 생성
        self.layers = OrderedDict()
        self.layers['Affine1'] = Affine(self.params['W1'], self.params['b1'])
        self.layers['Relu1'] = Relu()

```

```

self.layers['BatchNorm1'] = BatchNormalization(gamma=1.0, beta=0.)

self.layers['Affine2'] = Affine(self.params['W2'], self.params['b2'])
self.layers['Relu2']=Relu()
self.layers['BatchNorm2'] = BatchNormalization(gamma=1.0, beta=0.)

self.layers['Affine3'] = Affine(self.params['W3'], self.params['b3'])

self.lastLayer = SoftmaxWithLoss()

def predict(self, x):
    for layer in self.layers.values():
        x = layer.forward(x)

    return x

# x : 입력 데이터, t : 정답 레이블
def loss(self, x, t):
    y = self.predict(x)
    return self.lastLayer.forward(y, t)

def accuracy(self, x, t):
    y = self.predict(x)
    y = np.argmax(y, axis=1)
    if t.ndim != 1: t = np.argmax(t, axis=1)

    accuracy = np.sum(y == t) / float(x.shape[0])
    return accuracy

# x : 입력 데이터, t : 정답 레이블
def numerical_gradient(self, x, t):
    loss_W = lambda W: self.loss(x, t)

    grads = {}
    grads['W1'] = numerical_gradient(loss_W, self.params['W1'])
    grads['b1'] = numerical_gradient(loss_W, self.params['b1'])
    grads['W2'] = numerical_gradient(loss_W, self.params['W2'])
    grads['b2'] = numerical_gradient(loss_W, self.params['b2'])
    grads['W3'] = numerical_gradient(loss_W, self.params['W3'])
    grads['b3'] = numerical_gradient(loss_W, self.params['b3'])
    return grads

def gradient(self, x, t):
    # forward
    self.loss(x, t)

    # backward
    dout = 1
    dout = self.lastLayer.backward(dout)

    layers = list(self.layers.values())
    layers.reverse()
    for layer in layers:
        dout = layer.backward(dout)

    # 결과 저장

```

```

grads = {}
grads['W1'], grads['b1'] = self.layers['Affine1'].dW, self.layers['Affine1'].db
grads['W2'], grads['b2'] = self.layers['Affine2'].dW, self.layers['Affine2'].db
grads['W3'], grads['b3'] = self.layers['Affine3'].dW, self.layers['Affine3'].db
return grads

```

class Adam:

```

def __init__(self, lr=0.001, beta1=0.9, beta2=0.999):
    self.lr = lr
    self.beta1 = beta1
    self.beta2 = beta2
    self.iter = 0
    self.m = None
    self.v = None

```

def update(self, params, grads):

```

    if self.m is None:
        self.m, self.v = {}, {}
    for key, val in params.items():
        self.m[key] = np.zeros_like(val)
        self.v[key] = np.zeros_like(val)

```

```

    self.iter += 1

```

```

    lr_t = self.lr * np.sqrt(1.0 - self.beta2 ** self.iter) / (1.0 - self.beta1 ** self.iter)

```

```

    for key in params.keys():

```

```

        self.m[key] += (1 - self.beta1) * (grads[key] - self.m[key])
        self.v[key] += (1 - self.beta2) * (grads[key] ** 2 - self.v[key])

```

```

        params[key] -= lr_t * self.m[key] / (np.sqrt(self.v[key]) + 1e-7)

```

optimizer=Adam()

# 데이터 읽기

```

(x_train, t_train), (x_test, t_test) = load_mnist(normalize=True, one_hot_label=True)
network = TwoLayerNet(input_size=784, hidden_size=50, hidden_size2=100, output_size=10)

```

class BatchNormalization:

```

    """

```

<http://arxiv.org/abs/1502.03167>

```

    """

```

```

def __init__(self, gamma, beta, momentum=0.9, running_mean=None, running_var=None):

```

```

    self.gamma = gamma

```

```

    self.beta = beta

```

```

    self.momentum = momentum

```

```

    self.input_shape = None # 합성곱 계층은 4차원, 완전연결 계층은 2차원

```

```

    # 시험할 때 사용할 평균과 분산

```

```

    self.running_mean = running_mean

```

```

    self.running_var = running_var

```

```

    # backward 시에 사용할 중간 데이터

```

```

    self.batch_size = None

```

```

    self.xc = None

```

```

    self.std = None

```

```

self.dgamma = None
self.dbeta = None

def forward(self, x, train_flg=True):
    self.input_shape = x.shape
    if x.ndim != 2:
        N, C, H, W = x.shape
        x = x.reshape(N, -1)
    out = self.__forward(x, train_flg)
    return out.reshape(*self.input_shape)

def __forward(self, x, train_flg):
    if self.running_mean is None:
        N, D = x.shape
        self.running_mean = np.zeros(D)
        self.running_var = np.zeros(D)

    if train_flg:
        mu = x.mean(axis=0)
        xc = x - mu
        var = np.mean(xc**2, axis=0)
        std = np.sqrt(var + 10e-7)
        xn = xc / std

        self.batch_size = x.shape[0]
        self.xc = xc
        self.xn = xn
        self.std = std
        self.running_mean = self.momentum * self.running_mean + (1-self.momentum) * mu
        self.running_var = self.momentum * self.running_var + (1-self.momentum) * var
    else:
        xc = x - self.running_mean
        xn = xc / ((np.sqrt(self.running_var + 10e-7)))

    out = self.gamma * xn + self.beta
    return out

def backward(self, dout):
    if dout.ndim != 2:
        N, C, H, W = dout.shape
        dout = dout.reshape(N, -1)

    dx = self.__backward(dout)

    dx = dx.reshape(*self.input_shape)
    return dx

def __backward(self, dout):
    dbeta = dout.sum(axis=0)
    dgamma = np.sum(self.xn * dout, axis=0)
    dxn = self.gamma * dout
    dxc = dxn / self.std
    dstd = -np.sum((dxn * self.xc) / (self.std * self.std), axis=0)
    dvar = 0.5 * dstd / self.std
    dxc += (2.0 / self.batch_size) * self.xc * dvar
    dmu = np.sum(dxc, axis=0)
    dx = dxc - dmu / self.batch_size

```

```

        self.dgamma = dgamma
        self.dbeta = dbeta

    return dx

# 하이퍼파라미터
iters_num = 10000 # 반복 횟수를 적절히 설정한다.
train_size = x_train.shape[0] # 60000 개
batch_size = 100 # 미니배치 크기
learning_rate = 0.1

train_loss_list = []
train_acc_list = []
test_acc_list = []

# 1에폭당 반복 수
iter_per_epoch = max(train_size / batch_size, 1)
print(iter_per_epoch) # 600

for i in range(iters_num):
    batch_mask = np.random.choice(train_size, batch_size)
    x_batch = x_train[batch_mask]
    t_batch = t_train[batch_mask]

    for key in ('W1', 'b1', 'W2', 'b2', 'W3', 'b3'):
        grads = network.gradient(x_batch, t_batch)
        params = network.params
        optimizer.update(params, grads)

    loss = network.loss(x_batch, t_batch)
    train_loss_list.append(loss)

    if i % iter_per_epoch == 0:
        train_acc = network.accuracy(x_train, t_train)
        test_acc = network.accuracy(x_test, t_test)
        train_acc_list.append(train_acc)
        test_acc_list.append(test_acc)
        print(train_acc, test_acc)

```

## 오버피팅을 억제하는 방법 2가지

1. 드롭아웃(dropout)
2. 가중치 감소 (Weight decay)

### 1. 드롭아웃(dropout)

" 오버 피팅을 억제하기 위해서 뉴런을 임의로 삭제하면서 학습시키는 방법"  
ex)

고양이와 개 사진을 구분하는 신경망

고양이 사진          은닉1층          고양이 점수  
o(입력노드) ----> 귀가 있다.          ---->

o(입력노드) ----> 꼬리가 있어요 -----> (삭제)  
 o(입력노드) ----> 고양이처럼 생긴 사람 같아요 -----> (삭제)  
 o(입력노드) ----> 집게발을 가지고 있어요 ----->  
 o(입력노드) ----> 장난스럽게 보여요 ----->

**전문가가 많으면 오히려 오답이 나올 수 있다.**

즉, 사공이 많으면 배가 산으로 가기 때문에 몇 명의 전문가만 선별해서 반복적으로 같은 결과가 나오면 그것을 답으로 보겠다.

일반적으로 입력층에서 20%~50%정도 은닉층에서는 50%정도의 노드를 생략한다.

### Dropout class 구현코드

```
class Dropout:
    """
    http://arxiv.org/abs/1207.0580
    """

    def __init__(self, dropout_ratio=0.15): #15%를 삭제
        self.dropout_ratio = dropout_ratio
        self.mask = None

    def forward(self, x, train_flg=True):
        if train_flg:
            self.mask = np.random.rand(*x.shape) > self.dropout_ratio
            return x * self.mask
        else:
            return x * (1.0 - self.dropout_ratio)

    def backward(self, dout):
        return dout * self.mask
```

### 설명

self.mask = np.random.rand(\*x.shape) > self.dropout\_ratio

**100개의 노드가 있을때 dropout\_ratio를 0.15를 줬으면 몇 개의 노드가 남을까?**

1번. 85개의 노드가 남는다.

2번. 15개가 남는다.

**코드에서는 훈련할때 1번, 테스트 할때는 2번이 사용한다.**

**문제171. 지금까지 만들었던 3층 신경망 코드에 dropout을 적용해서 오버피팅이 발생하지 않는지 확인하시오.(50%)**

답)

# coding: utf-8

import sys, os

sys.path.append(os.pardir) # 부모 디렉터리의 파일을 가져올 수 있도록 설정

import numpy as np

from common.layers import \*

```

from common.gradient import numerical_gradient
from collections import OrderedDict
import matplotlib.pyplot as plt
from dataset.mnist import load_mnist

class TwoLayerNet:
    def __init__(self, input_size, hidden_size, hidden_size2, output_size, weight_init_std=1):
        # 가중치 초기화
        self.params = {}
        self.params['W1'] = np.sqrt(2/784) * np.random.randn(input_size, hidden_size)
        self.params['b1'] = np.zeros(hidden_size)
        self.params['W2'] = np.sqrt(2/50) * np.random.randn(hidden_size, hidden_size2)
        self.params['b2'] = np.zeros(hidden_size2)
        self.params['W3'] = np.sqrt(2/100) * np.random.randn(hidden_size2, output_size)
        self.params['b3'] = np.zeros(output_size)

        # 계층 생성
        self.layers = OrderedDict()
        self.layers['Affine1'] = Affine(self.params['W1'], self.params['b1'])
        self.layers['BatchNorm1'] = BatchNormalization(gamma=1.0, beta=0.)
        self.layers['Relu1'] = Relu()
        self.layers['Dropout1']=Dropout()

        self.layers['Affine2'] = Affine(self.params['W2'], self.params['b2'])
        self.layers['BatchNorm2'] = BatchNormalization(gamma=1.0, beta=0.)
        self.layers['Relu2']=Relu()
        self.layers['Dropout2']=Dropout()

        self.layers['Affine3'] = Affine(self.params['W3'], self.params['b3'])

        self.lastLayer = SoftmaxWithLoss()

    def predict(self, x):
        for layer in self.layers.values():
            x = layer.forward(x)

        return x

    # x : 입력 데이터, t : 정답 레이블
    def loss(self, x, t):
        y = self.predict(x)
        return self.lastLayer.forward(y, t)

    def accuracy(self, x, t):
        y = self.predict(x)
        y = np.argmax(y, axis=1)
        if t.ndim != 1: t = np.argmax(t, axis=1)

        accuracy = np.sum(y == t) / float(x.shape[0])
        return accuracy

    # x : 입력 데이터, t : 정답 레이블
    def numerical_gradient(self, x, t):
        loss_W = lambda W: self.loss(x, t)

```

```

grads = {}
grads['W1'] = numerical_gradient(loss_W, self.params['W1'])
grads['b1'] = numerical_gradient(loss_W, self.params['b1'])
grads['W2'] = numerical_gradient(loss_W, self.params['W2'])
grads['b2'] = numerical_gradient(loss_W, self.params['b2'])
grads['W3'] = numerical_gradient(loss_W, self.params['W3'])
grads['b3'] = numerical_gradient(loss_W, self.params['b3'])
return grads

def gradient(self, x, t):
    # forward
    self.loss(x, t)

    # backward
    dout = 1
    dout = self.lastLayer.backward(dout)

    layers = list(self.layers.values())
    layers.reverse()
    for layer in layers:
        dout = layer.backward(dout)

    # 결과 저장
    grads = {}
    grads['W1'], grads['b1'] = self.layers['Affine1'].dW, self.layers['Affine1'].db
    grads['W2'], grads['b2'] = self.layers['Affine2'].dW, self.layers['Affine2'].db
    grads['W3'], grads['b3'] = self.layers['Affine3'].dW, self.layers['Affine3'].db
    return grads

class Adam:
    def __init__(self, lr=0.001, beta1=0.9, beta2=0.999):
        self.lr = lr
        self.beta1 = beta1
        self.beta2 = beta2
        self.iter = 0
        self.m = None
        self.v = None

    def update(self, params, grads):
        if self.m is None:
            self.m, self.v = {}, {}
            for key, val in params.items():
                self.m[key] = np.zeros_like(val)
                self.v[key] = np.zeros_like(val)

        self.iter += 1
        lr_t = self.lr * np.sqrt(1.0 - self.beta2 ** self.iter) / (1.0 - self.beta1 ** self.iter)

        for key in params.keys():

            self.m[key] += (1 - self.beta1) * (grads[key] - self.m[key])
            self.v[key] += (1 - self.beta2) * (grads[key] ** 2 - self.v[key])

            params[key] -= lr_t * self.m[key] / (np.sqrt(self.v[key]) + 1e-7)

optimizer=Adam()

```



```

# 데이터 읽기
(x_train, t_train), (x_test, t_test) = load_mnist(normalize=True, one_hot_label=True)
network = TwoLayerNet(input_size=784, hidden_size=50, hidden_size2=100, output_size=10)

class BatchNormalization:
    """
    http://arxiv.org/abs/1502.03167
    """
    def __init__(self, gamma, beta, momentum=0.9, running_mean=None, running_var=None):
        self.gamma = gamma
        self.beta = beta
        self.momentum = momentum
        self.input_shape = None # 합성곱 계층은 4차원, 완전연결 계층은 2차원

        # 시험할 때 사용할 평균과 분산
        self.running_mean = running_mean
        self.running_var = running_var

        # backward 시에 사용할 중간 데이터
        self.batch_size = None
        self.xc = None
        self.std = None
        self.dgamma = None
        self.dbeta = None

    def forward(self, x, train_flg=True):
        self.input_shape = x.shape
        if x.ndim != 2:
            N, C, H, W = x.shape
            x = x.reshape(N, -1)
        out = self.__forward(x, train_flg)
        return out.reshape(*self.input_shape)

    def __forward(self, x, train_flg):
        if self.running_mean is None:
            N, D = x.shape
            self.running_mean = np.zeros(D)
            self.running_var = np.zeros(D)

        if train_flg:
            mu = x.mean(axis=0)
            xc = x - mu
            var = np.mean(xc**2, axis=0)
            std = np.sqrt(var + 10e-7)
            xn = xc / std

            self.batch_size = x.shape[0]
            self.xc = xc
            self.xn = xn
            self.std = std
            self.running_mean = self.momentum * self.running_mean + (1-self.momentum) * mu
            self.running_var = self.momentum * self.running_var + (1-self.momentum) * var
        else:
            xc = x - self.running_mean
            xn = xc / ((np.sqrt(self.running_var + 10e-7)))

```

```

        out = self.gamma * xn + self.beta
        return out

def backward(self, dout):
    if dout.ndim != 2:
        N, C, H, W = dout.shape
        dout = dout.reshape(N, -1)

    dx = self.__backward(dout)

    dx = dx.reshape(*self.input_shape)
    return dx

def __backward(self, dout):
    dbeta = dout.sum(axis=0)
    dgamma = np.sum(self.xn * dout, axis=0)
    dxn = self.gamma * dout
    dxc = dxn / self.std
    dstd = -np.sum((dxn * self.xc) / (self.std * self.std), axis=0)
    dvar = 0.5 * dstd / self.std
    dxc += (2.0 / self.batch_size) * self.xc * dvar
    dmu = np.sum(dxc, axis=0)
    dx = dxc - dmu / self.batch_size

    self.dgamma = dgamma
    self.dbeta = dbeta

    return dx

class Dropout:
    """
    http://arxiv.org/abs/1207.0580
    """

    def __init__(self, dropout_ratio=0.5): #50%를 삭제
        self.dropout_ratio = dropout_ratio
        self.mask = None

    def forward(self, x, train_flg=True):
        if train_flg:
            self.mask = np.random.rand(*x.shape) > self.dropout_ratio
            return x * self.mask
        else:
            return x * (1.0 - self.dropout_ratio)

    def backward(self, dout):
        return dout * self.mask

# 하이퍼파라미터
iters_num = 10000 # 반복 횟수를 적절히 설정한다.
train_size = x_train.shape[0] # 60000 개
batch_size = 100 # 미니배치 크기
learning_rate = 0.1

train_loss_list = []

```

```

train_acc_list = []
test_acc_list = []

# 1에폭당 반복 수
iter_per_epoch = max(train_size / batch_size, 1)
#print(iter_per_epoch) # 600

for i in range(iters_num):
    batch_mask = np.random.choice(train_size, batch_size)
    x_batch = x_train[batch_mask]
    t_batch = t_train[batch_mask]

    for key in ('W1', 'b1', 'W2', 'b2', 'W3', 'b3'):
        grads = network.gradient(x_batch, t_batch)
        params = network.params
        optimizer.update(params, grads)

    loss = network.loss(x_batch, t_batch)
    train_loss_list.append(loss)

    if i % iter_per_epoch == 0:
        train_acc = network.accuracy(x_train, t_train)
        test_acc = network.accuracy(x_test, t_test)
        train_acc_list.append(train_acc)
        test_acc_list.append(test_acc)
        print(train_acc, test_acc)

```

## 2. 가중치 감소

" 학습과정에서 큰 가중치에 대해서는 그에 상응하는 큰 페널티를 부여하여 오버피팅을 억제하는 방법 "

예: 고양이와 개 사진을 구분하는 신경망

고양이 사진          은닉1층

고양이 점수

$\alpha$ (입력노드) ----> 귀가 있다.          ----> 가중치 출력

$\alpha$ (입력노드) ----> 꼬리가 있어요          -----> 아주 큰 가중치 출력

$\alpha$ (입력노드) ----> 집게발을 가지고 있어요 ----> 가중치 출력

$\alpha$ (입력노드) ----> 장난스럽게 보여요 ----> 가중치 출력

## 가중치 decay코드 분석

" 모든 가중치의 각각의 손실 함수에  $\frac{1}{2} \times \text{람다} \times w^2$ 을 더한다."

### 1. 오차 함수의 코드에서 수정해야 할 부분

```

weight_decay += 0.5 * self.weight_decay_lambda * np.sum(w**2)
return self.lastLayer.forward(y,t) + weight_decay

```

### 2. gradient Descent 업데이트 과정에서 그동안 오차 역전파법에 따른 결과에 정규화 항을 미분한 값에

람다 \*가중치를 더한다.

코드예제:

기존코드 :  $\text{grad}['W1'] = \text{미분값}$

변경된 코드 :  $\text{grad}['W1'] = \text{미분값} + \lambda * \text{현재의 가중치}$

**문제172. 오버피팅을 억제하는 가중치 감소 방법중에 하나의 L2정규화를 이용해서 3층 신경망을 구현하시오.**

답)

```
import sys, os
```

```
sys.path.append(os.pardir)
import numpy as np
from dataset.mnist import load_mnist
from common.layers import *
from common.gradient import numerical_gradient
from collections import OrderedDict
```

```
##### 배치정규화 클래스 #####
```

```
class BatchNormalization:
```

```
    """
```

```
    http://arxiv.org/abs/1502.03167
```

```
    """
```

```
    def __init__(self, gamma, beta, momentum=0.9, running_mean=None,
running_var=None):
```

```
        self.gamma = gamma
```

```
        self.beta = beta
```

```
        self.momentum = momentum
```

```
        self.input_shape = None # 합성곱 계층은 4차원, 완전연결 계층은 2차원
```

```
        # 시험할 때 사용할 평균과 분산
```

```
        self.running_mean = running_mean
```

```
        self.running_var = running_var
```

```
        # backward 시에 사용할 중간 데이터
```

```
        self.batch_size = None
```

```
        self.xc = None
```

```
        self.std = None
```

```
        self.dgamma = None
```

```
        self.dbeta = None
```

```
    def forward(self, x, train_flg=True):
```

```
        self.input_shape = x.shape
```

```
        if x.ndim != 2:
```

```
            N, C, H, W = x.shape
```

```
            x = x.reshape(N, -1)
```

```
        out = self.__forward(x, train_flg)
```

```
        return out.reshape(*self.input_shape)
```

```
    def __forward(self, x, train_flg):
```

```
        if self.running_mean is None:
```

```
            N, D = x.shape
```

```

        self.running_mean = np.zeros(D)
        self.running_var = np.zeros(D)

    if train_flg:
        mu = x.mean(axis=0)
        xc = x - mu
        var = np.mean(xc ** 2, axis=0)
        std = np.sqrt(var + 10e-7)
        xn = xc / std

        self.batch_size = x.shape[0]
        self.xc = xc
        self.xn = xn
        self.std = std
        self.running_mean = self.momentum * self.running_mean + (1 - self.momentum) *
mu        self.running_var = self.momentum * self.running_var + (1 - self.momentum) * var
    else:
        xc = x - self.running_mean
        xn = xc / ((np.sqrt(self.running_var + 10e-7)))

    out = self.gamma * xn + self.beta
    return out

def backward(self, dout):
    if dout.ndim != 2:
        N, C, H, W = dout.shape
        dout = dout.reshape(N, -1)

    dx = self.__backward(dout)

    dx = dx.reshape(*self.input_shape)
    return dx

def __backward(self, dout):
    dbeta = dout.sum(axis=0)
    dgamma = np.sum(self.xn * dout, axis=0)
    dxn = self.gamma * dout
    dxc = dxn / self.std
    dstd = -np.sum((dxn * self.xc) / (self.std * self.std), axis=0)
    dvar = 0.5 * dstd / self.std
    dxc += (2.0 / self.batch_size) * self.xc * dvar
    dmua = np.sum(dxc, axis=0)
    dx = dxc - dmua / self.batch_size

    self.dgamma = dgamma
    self.dbeta = dbeta

    return dx

```

##### 오버피팅을 억제하기 위한 Dropout 클래스 #####

class Dropout:

"""

<http://arxiv.org/abs/1207.0580>

"""

```

def __init__(self, dropout_ratio=0.15):
    self.dropout_ratio = dropout_ratio
    self.mask = None

def forward(self, x, train_flg=True):
    if train_flg:
        self.mask = np.random.rand(*x.shape) > self.dropout_ratio
        return x * self.mask
    else:
        return x * (1.0 - self.dropout_ratio)

def backward(self, dout):
    return dout * self.mask

class TwoLayersNet:
    def __init__(self, input_size, hidden_size_1, hidden_size_2, output_size, weight_init_std=1):
        self.weight_decay_lambda = 0.001
        self.params = {}
        self.params['W1'] = np.sqrt(2 / 784) * np.random.randn(input_size, hidden_size_1)
        self.params['b1'] = np.zeros(hidden_size_1)
        self.params['W2'] = np.sqrt(2 / 50) * np.random.randn(hidden_size_1, hidden_size_2)
        self.params['b2'] = np.zeros(hidden_size_2)
        self.params['W3'] = np.sqrt(2 / 100) * np.random.randn(hidden_size_2, output_size)
        self.params['b3'] = np.zeros(output_size)

        self.layers = OrderedDict()

        self.layers['Affine1'] = Affine(self.params['W1'], self.params['b1'])
        self.layers['BatchNorm1'] = BatchNormalization(gamma=1.0, beta=0.)
        self.layers['Relu1'] = Relu()

        self.layers['Affine2'] = Affine(self.params['W2'], self.params['b2'])
        self.layers['BatchNorm2'] = BatchNormalization(gamma=1.0, beta=0.)
        self.layers['Relu2'] = Relu()

        self.layers['Affine3'] = Affine(self.params['W3'], self.params['b3'])

        self.lastLayer = SoftmaxWithLoss()

    def predict(self, x):
        for layer in self.layers.values():
            x = layer.forward(x)

        return x

# L2 정규화 적용 후

def loss(self, x, t): # x : (100, 1024), t : (100, 10)

    y = self.predict(x) # (100, 10) : 마지막 출력층을 통과한 신경망이 예측한 값

```

```

weight_decay = 0

for idx in range(1, 4):
    W = self.params['W' + str(idx)]

    weight_decay += 0.5 * 0.001 * np.sum(W ** 2)

return self.lastLayer.forward(y, t) + weight_decay

def accuracy(self, x, t):
    y = self.predict(x)
    y = np.argmax(y, axis=1)
    if t.ndim != 1: t = np.argmax(t, axis=1)

    accuracy = np.sum(y == t) / float(x.shape[0])

    return accuracy

def numerical_gradient(self, x, t):
    loss_W = lambda W: self.loss(x, t)

    grads = {}
    grads['W1'] = numerical_gradient(loss_W, self.params['W1'])
    grads['b1'] = numerical_gradient(loss_W, self.params['b1'])
    grads['W2'] = numerical_gradient(loss_W, self.params['W2'])
    grads['b2'] = numerical_gradient(loss_W, self.params['b2'])
    grads['W3'] = numerical_gradient(loss_W, self.params['W3'])
    grads['b3'] = numerical_gradient(loss_W, self.params['b3'])
    return grads

def gradient(self, x, t):
    self.loss(x, t)

    dout = 1
    dout = self.lastLayer.backward(dout)

    layers = list(self.layers.values())
    layers.reverse()
    for layer in layers:
        dout = layer.backward(dout)

    grads = {}

    for idx in range(1, 4):

        grads['W' + str(idx)] = self.layers['Affine' + str(idx)].dW + self.weight_decay_lambda *
self.layers['Affine' + str(idx)].W

        grads['b' + str(idx)] = self.layers['Affine' + str(idx)].db

    return grads

(x_train, t_train), (x_test, t_test) = load_mnist(normalize=True, one_hot_label=True)

network = TwoLayersNet(input_size=784, hidden_size_1=50, hidden_size_2=100,
output_size=10)

```

```

iters_num = 10000
train_size = x_train.shape[0]
batch_size = 100
learning_rate = 0.1

train_loss_list = []
train_acc_list = []
test_acc_list = []

iter_per_epoch = max(train_size / batch_size, 1)

##### SGD 클래스 #####
class SGD:
    def __init__(self, lr=0.01):
        self.lr = lr

    def update(self, params, grads):
        for key in params.keys():
            params[key] -= self.lr * grads[key]

# optimizer = SGD() # SGD 클래스를 객체화 시킨다.

##### Momentum 클래스 #####
class Momentum:
    def __init__(self, lr=0.01, momentum=0.9):
        self.lr = lr
        self.momentum = momentum
        self.v = None

    def update(self, params, grads):
        if self.v is None:
            self.v = {}
            for key, val in params.items():
                self.v[key] = np.zeros_like(val)

        for key in params.keys():
            self.v[key] = self.momentum * self.v[key] - self.lr * grads[key]
            params[key] += self.v[key]

# 속도  $\leftarrow 0.9(\text{마찰계수}) * \text{속도} - \text{러닝레이트} * \text{기울기}$ 
# 가중치  $\leftarrow \text{가중치} + \text{속도}$ 

# optimizer = Momentum() # Momentum 클래스 객체화

##### Adagrad 클래스 #####
class AdaGrad:
    def __init__(self, lr=0.01):
        self.lr = lr

```



```

self.h = None

def update(self, params, grads):
    if self.h is None:
        self.h = {}
        for key, val in params.items():
            self.h[key] = np.zeros_like(val)

    for key in params.keys():
        self.h[key] += grads[key] * grads[key]
        params[key] -= self.lr * grads[key] / (np.sqrt(self.h[key]) + 1e-7)

# optimizer = AdaGrad() # AdaGrad 클래스 객체화

##### Adam 클래스 #####
class Adam:

    def __init__(self, lr=0.001, beta1=0.9, beta2=0.999):
        self.lr = lr
        self.beta1 = beta1
        self.beta2 = beta2
        self.iter = 0
        self.m = None
        self.v = None

    def update(self, params, grads):
        if self.m is None:
            self.m, self.v = {}, {}
            for key, val in params.items():
                self.m[key] = np.zeros_like(val)
                self.v[key] = np.zeros_like(val)

        self.iter += 1
        lr_t = self.lr * np.sqrt(1.0 - self.beta2 ** self.iter) / (1.0 - self.beta1 ** self.iter)

        for key in params.keys():
            self.m[key] += (1 - self.beta1) * (grads[key] - self.m[key])
            self.v[key] += (1 - self.beta2) * (grads[key] ** 2 - self.v[key])

            params[key] -= lr_t * self.m[key] / (np.sqrt(self.v[key]) + 1e-7)

optimizer = Adam() # Adam 클래스 객체화

for i in range(iters_num):
    batch_mask = np.random.choice(train_size, batch_size)
    x_batch = x_train[batch_mask]
    t_batch = t_train[batch_mask]

    for key in ('W1', 'b1', 'W2', 'b2', 'W3', 'b3'):
        grads = network.gradient(x_batch, t_batch)
        params = network.params
        optimizer.update(params, grads)

```

```
loss = network.loss(x_batch, t_batch)
train_loss_list.append(loss)

if i % iter_per_epoch == 0:
    train_acc = network.accuracy(x_train, t_train)
    test_acc = network.accuracy(x_test, t_test)
    train_acc_list.append(train_acc)
    test_acc_list.append(test_acc)
    print(train_acc, test_acc)
```

## 7장 CNN(Convolution neural network)

2018년 9월 4일 화요일 오전 11:31

### 합성곱 신경망 ?

" Convolution 층과 pooling층을 포함하는 신경망 "

기존 신경망과의 차이?

- 기존방법 : Affine ---> Relu
- CNN : Conv ---> Relu ---> Pooling
- 기존에 구현했던 완전 연결 계층의 문제점?
  - " 데이터의 형상이 무시된다 "

합성곱층의 역할?

이미지의 특징을 추출한다.

cnn을 이용하지 않은 기존층의 문제점

필기체 -->  $28 \times 28 = 784$ 의 1차원 데이터로 변경을해서 784개의 데이터를 첫 affine 계층의 입력한 것.

↓

형상을 무시하고 모든 입력 데이터를 동등한 뉴런으로 취급해서 이미지가 갖는 본질적인 패턴을 읽지 못한다.

↓

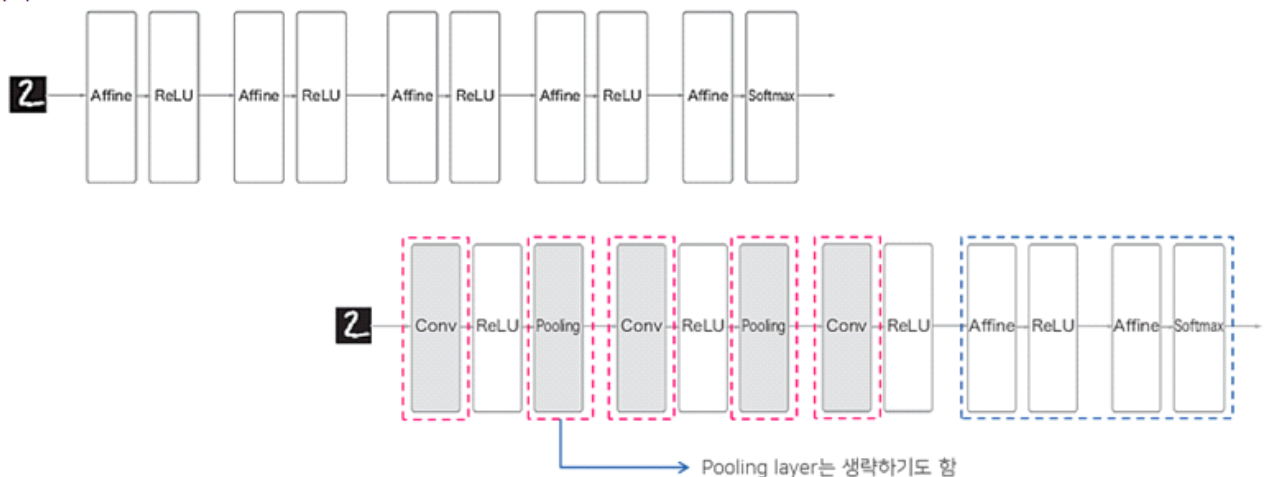
그래서 합성곱 계층이 필요하다!!

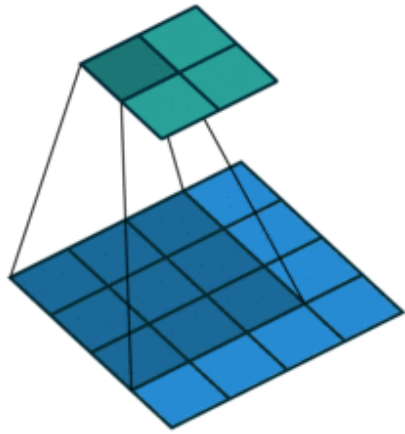
원본 이미지에서 조금만 모양이 달라져도 같은 이미지로 인식하지 못하는 문제를 합성곱이 해결

어떻게 해결을 하는가? 원본 이미지를 가지고 여러 개의 feature map을 만들어서 분류하는 완전 연결 계층에 입력한다.

### 합성곱 계층

" feature map을 만들고 그 feature map을 선명하게 해주는 층"





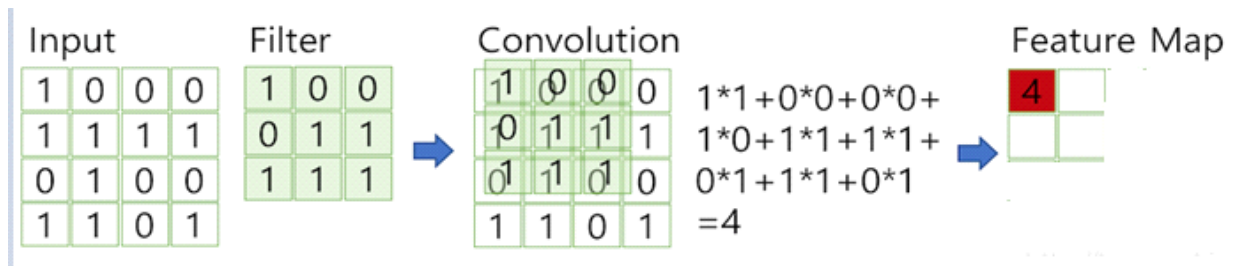
합성곱 연산? 이미지 3차원(세로, 가로, 색상) data의 형상을 유지하면서 연산하는 작업

" 입력 데이터에 필터를 적용한 것."

합성곱 연산을 컴퓨터로 구현하는 방법

```

1 2 3 0
0 1 2 3      2 0 1
3 0 1 2  ⊙  0 1 2 =  15 16
2 3 0 1      1 0 2      6 15
(입력 데이터)   (필터)
  
```



문제173. 아래 두 행렬을 만들고 합성곱한 결과를 출력하시오.

```

보기)
1 2 3      2 0 1
0 1 2  ⊙  0 1 2 =  15
2 0 1      1 0 2
  
```

```

답)
import numpy as np

aa=np.array([[1,2,3],[0,1,2],[3,0,1]])
ff=np.array([[2,0,1],[0,1,2],[1,0,2]])

print(np.sum(aa*ff))
  
```

문제174. 아래의 4x4행렬에서 아래의 3x3행렬만 추출하시오.

```

보기)
1 2 3 0
  
```

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

답)

```
import numpy as np
```

```
aa=np.array([[1,2,3,0],[0,1,2,3],[3,0,1,2],[2,3,0,1]])
```

```
ff=np.array([[2,0,1],[0,1,2],[1,0,2]])
```

```
print(aa[:3,:3])
```

```
[[1 2 3]
 [0 1 2]
 [3 0 1]]
```

문제175. 아래의 행렬에서 아래의 결과 행렬을 추출하시오.

보기)

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

답)

```
import numpy as np
```

```
aa=np.array([[1,2,3,0],[0,1,2,3],[3,0,1,2],[2,3,0,1]])
```

```
ff=np.array([[2,0,1],[0,1,2],[1,0,2]])
```

```
print(aa[:3,1:4])
```

```
[[2 3 0]
 [1 2 3]
 [0 1 2]]
```

```
print(aa)
```

```
print(aa[:3,:3])
```

```
print(aa[:3,1:4])
```

```
print(aa[1:4,:3])
```

```
print(aa[1:4,1:4])
```

```

[[1 2 3 0]
 [0 1 2 3]
 [3 0 1 2]
 [2 3 0 1]]
[[1 2 3]
 [0 1 2]
 [3 0 1]]
[[2 3 0]
 [1 2 3]
 [0 1 2]]
[[0 1 2]
 [3 0 1]
 [2 3 0]]
[[1 2 3]
 [0 1 2]
 [3 0 1]]

```

문제176. 아래의 4x4행렬에서 아래의 결과를 출력하시오.

보기)

```

1 2 3 0
0 1 2 3
3 0 1 2 ----> [13, 14, 12, 13]
2 3 0 1

```

답)

```

import numpy as np

aa=np.array([[1,2,3,0],[0,1,2,3],[3,0,1,2],[2,3,0,1]])

ff=np.array([[2,0,1],[0,1,2],[1,0,2]])

res=[]

for i in range(len(aa)-2):
    for j in range(len(aa)-2):
        res.append(np.sum(aa[i:i+3,j:j+3]))
print(res)

```

---

```

[13, 14, 12, 13]

```

문제177. 아래의 합성곱을 파이썬으로 구현하시오.

보기)

```

1 2 3 0
0 1 2 3      2 0 1
3 0 1 2  ⊗  0 1 2 =  [15, 16, 6, 15]
2 3 0 1      1 0 2

```

답)

```

import numpy as np

aa=np.array([[1,2,3,0],[0,1,2,3],[3,0,1,2],[2,3,0,1]])

```

```
ff=np.array([[2,0,1],[0,1,2],[1,0,2]])

res=[]

for i in range(len(aa)-2):
    for j in range(len(aa)-2):
        res.append(np.sum(aa[i:i+3,j:j+3]*ff))
print(res)
[15, 16, 6, 15]
```

문제178. 위에 결과를 2x2행렬로 변환하시오.

답)

```
import numpy as np

aa=np.array([[1,2,3,0],[0,1,2,3],[3,0,1,2],[2,3,0,1]])

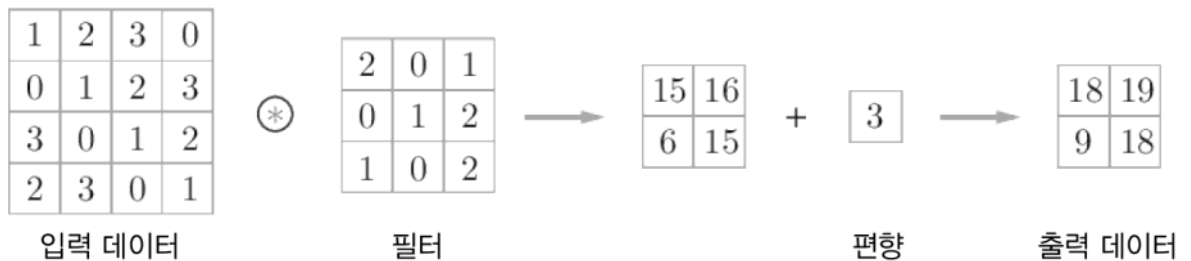
ff=np.array([[2,0,1],[0,1,2],[1,0,2]])

res=[]

for i in range(len(aa)-2):
    for j in range(len(aa)-2):
        res.append(np.sum(aa[i:i+3,j:j+3]*ff))
print(np.reshape(res,(2,2)))
[[15 16]
 [ 6 15]]
```

문제179. 아래의 그림의 convolution 연산을 파이썬으로 구현하시오.

보기)



답)

```
import numpy as np

aa=np.array([[1,2,3,0],[0,1,2,3],[3,0,1,2],[2,3,0,1]])

ff=np.array([[2,0,1],[0,1,2],[1,0,2]])

res=[]

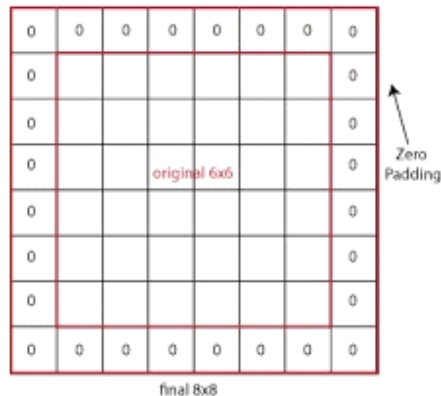
for i in range(len(aa)-2):
    for j in range(len(aa)-2):
        res.append(np.sum(aa[i:i+3,j:j+3]*ff)+3)
print(np.reshape(res,(2,2)))
```

패딩

" 합성곱 연산을 수행하기전에 입력 데이터 주변을 특정값으로 채워 늘리는 것. "

### - 패딩이 필요한 이유?

패딩을 하지 않을 경우 data의 공간크기는 합성곱 계층을 지날때 마다 작아지게 되므로 가장자리 정보들이 사라지게 되는 문제가 발생하기 때문에 패딩을 사용한다.



0	0	0	0	0	0
0	35	19	25	6	0
0	13	22	16	53	0
0	4	3	7	10	0
0	9	8	1	3	0
0	0	0	0	0	0

문제180. 위에서 출력한 2x2행렬에 제로 패딩 1을 수행하시오.

답)

```
import numpy as np
res=np.array([[15,16],[6,15]])
```

```
res_pad=np.pad(res,pad_width=1, mode='constant', constant_values=0)
print(res_pad)
```

```
[[ 0  0  0  0]
 [ 0 15 16  0]
 [ 0  6 15  0]
 [ 0  0  0  0]]
```

문제181. 4x4행렬에 3x3필터를 적용해서 결과로 4x4행렬이 출력되게 하려면 제로패딩을 몇을 해줘야 하는가?

공식)



입력크기를 (H, W), 필터크기를(FH, FW), 출력크기를(OH, OW), 패딩을P, 스트라이드를S라고 할때

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

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

답)2

문제182. 패딩을 답으로 해서 위의 식을 풀어보시오.

답)

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

문제183. 입력 이미지 4x4 행렬에 필터 3x3행렬을 합성곱 한 출력 결과 행렬이 4x4행렬이 되려면 패딩이 몇인지 출력하시오.

답)

$$P = \frac{1(4-1) + 3 - 4}{2} = 1$$

문제184. 위의 패딩 공식을 파이썬 함수를 생성하시오.

답)

```
def padding(H,S,OH,FH):  
    return (S*(OH-1)+FH-H)/2
```

```
print(padding(4,1,4,3))
```

```
1.0
```

문제185. 입력행렬(6x6), 스트라이드1, 출력행렬(6x6), 필터행렬(3x3)의 패딩이 몇인지 출력하시오.

답)

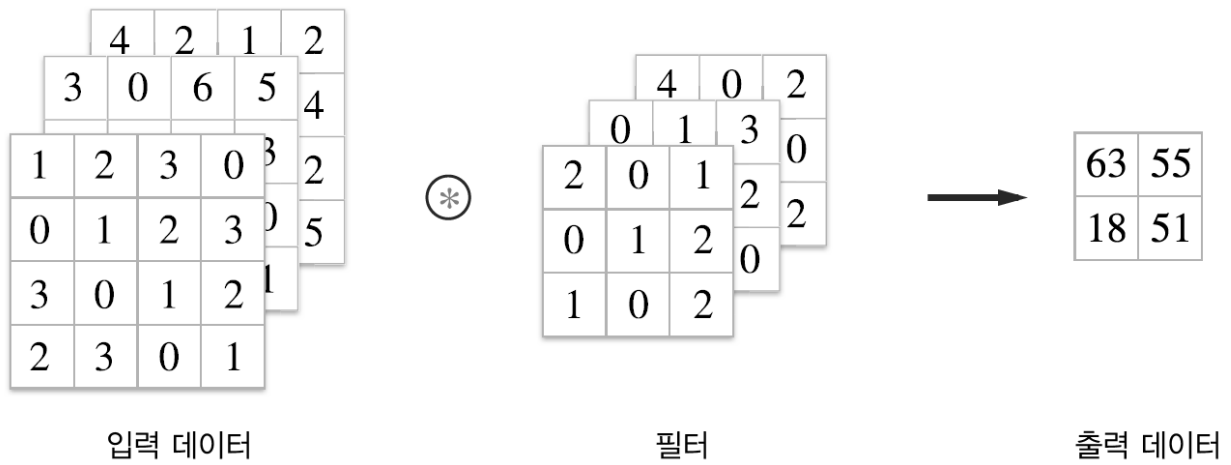
```
def padding(H,S,OH,FH):  
    return (S*(OH-1)+FH-H)/2
```

```
print(padding(6,1,6,3))
```

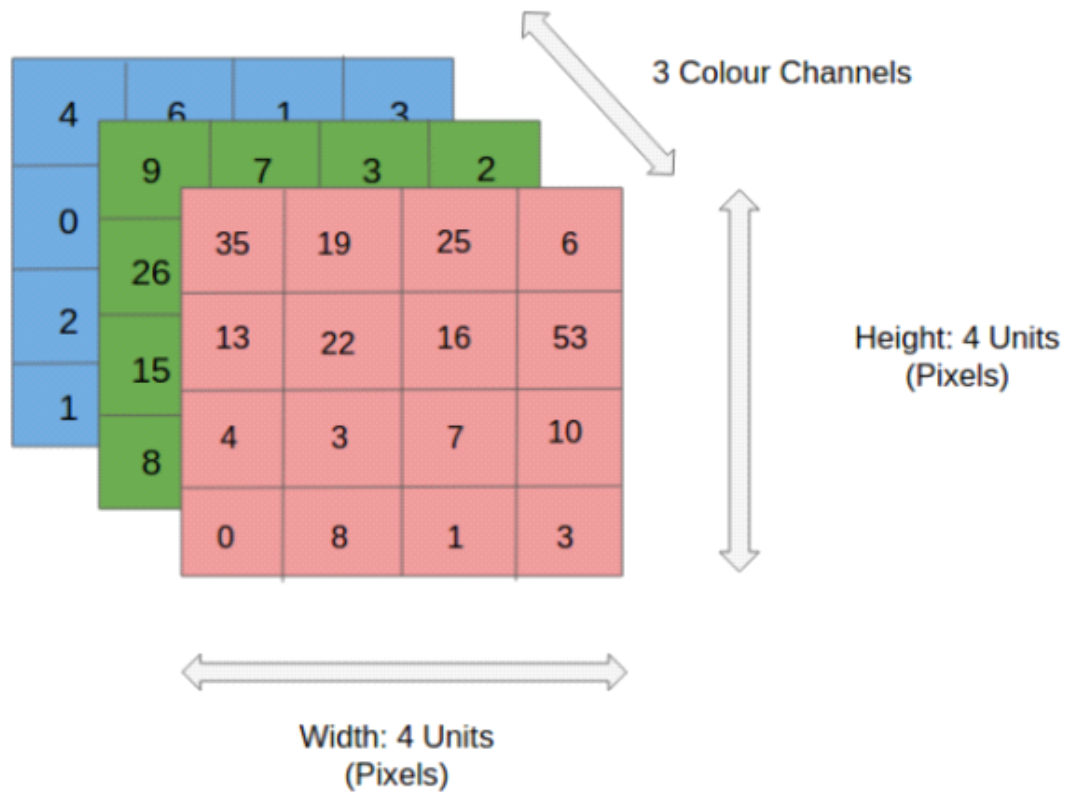
```
1.0
```

### 3차원의 합성곱

" 이미지의 색깔이 보통은 흑백이 아니라 RGB컬러이므로 RGB(Red, Green, Blue) 컬러에 대해서 합성곱을 해야한다. "



■ 3차원 합성곱을 이해하기 위한 그림



문제187. 아이린 사진에서 Red행렬만 출력하고 시각화하시오.

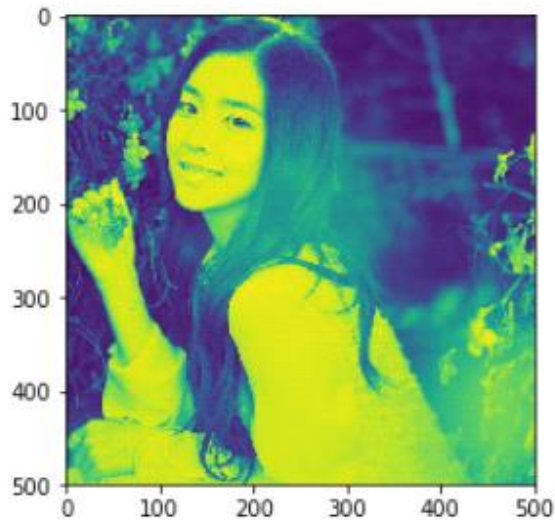
답)

```
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
```

##5.1. 원본 이미지 불러오기

```
img = Image.open('C:\\python_data\\이린이.jpg')
img_pixel = np.array(img)
plt.imshow(img_pixel[:, :, 0]) #이미지
print(img_pixel[:, :, 0]) #행렬
```

```
[[ 79 113 147 ... 50 50 50]
 [101 137 169 ... 51 51 50]
 [116 152 181 ... 49 49 49]
 ...
 [219 219 222 ... 163 173 184]
 [222 222 224 ... 163 174 184]
 [222 223 225 ... 165 176 184]]
```



설명 : [:,:,0] = [가로,세로,색상] (0:Red, 1:Green, 2:Blue)

문제188. 여러분들이 원하는 사진을 직접 내려받아 R,G,B를 확인해보시오.

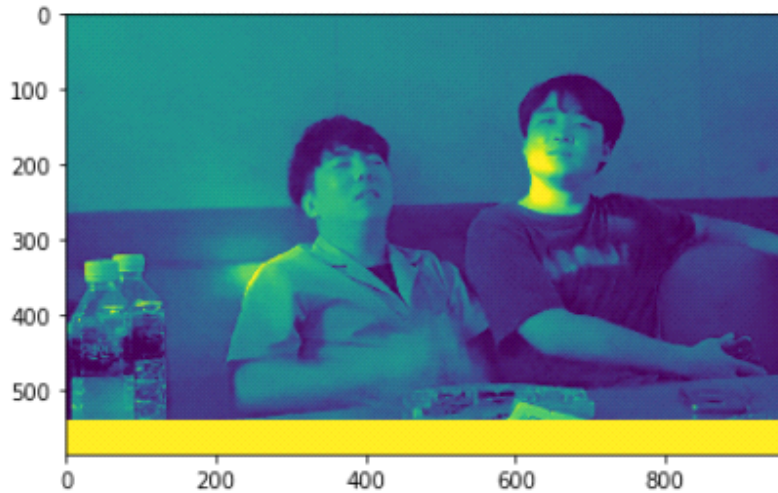
답)

```
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
```

##5.1. 원본 이미지 불러오기

```
img = Image.open('C:\\python_data\\해찬이형ㄹ.png')
img_pixel = np.array(img)
plt.imshow(img_pixel[:,:,:0])
print(img_pixel[:,:,:0])
```

```
[[134 135 135 ... 80 80 80]
 [135 135 135 ... 79 79 79]
 [135 135 135 ... 80 79 79]
 ...
 [255 255 255 ... 255 255 255]
 [255 255 255 ... 255 255 255]
 [255 255 255 ... 255 255 255]]
```

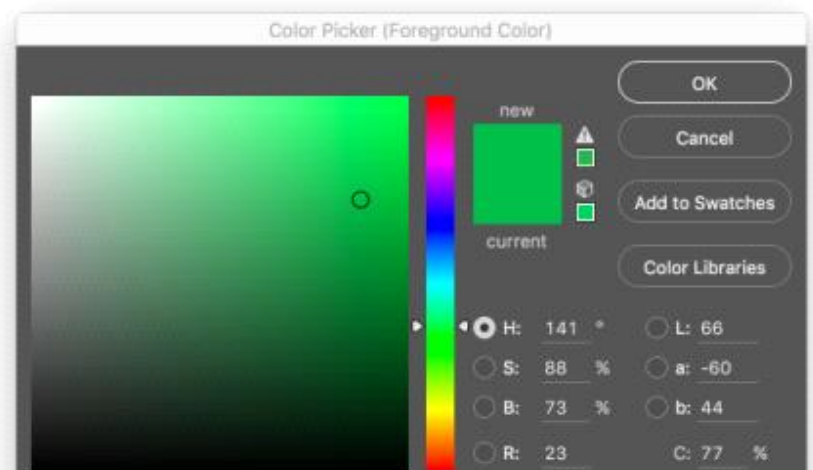


**설명 : Red가 초록색으로 나오는 이유?**

RGB 값을 HSB 로 변환하면 기본적으로 초록색이라고 하네요. 그러니까 우리가 보는 빨간색, 초록색, 파란색이 기본적으로 초록색인 것이지요.

## HSB의 탄생

1, 2, 3번은 쉽게 답을 맞혔을 것이다. 1번은 Red가 최대고, Green과 Blue는 없는 상태이므로 당연히 빨간색일 것이다. 2, 3번도 같은 식으로 각각 초록, 파랑이라고 답할 수 있다. 반면 4번은 무슨 색인지 유추해내기 어렵다. 이렇게 RGB 값은 직관적으로 이해하기 어려우므로 HSB라는 호환체계가 1970년경에 고안되었다. 그럼 다시 4번 문제로 돌아가서 RGB(23, 185, 79)를 HSB로 변환해 보면 HSB(141, 88, 73)이다. 바로 명 채도가 약간 낮은 초록색이라는 것을 알 수 있다.



문제189. R,G,B 행렬을 이해하기 위해서 아래의 numpy array를 이해하시오.

```
보기)
data = np.array(
    [
        [[2, 2, 1, 1, 0],
         [0, 0, 1, 0, 0],
         [0, 2, 0, 0, 1],
         [1, 2, 1, 1, 1],
         [1, 0, 1, 0, 1]], #-->Red
        [[2, 0, 0, 0, 1],
         [0, 2, 2, 0, 1],
         [0, 0, 0, 0, 2],
         [0, 1, 2, 0, 1],
         [2, 0, 2, 2, 2]], #-->Green
        [[4, 2, 1, 2, 1],
         [0, 1, 0, 4, 2],
         [3, 0, 6, 2, 3],
         [4, 2, 4, 5, 4],
         [0, 1, 2, 0, 1]] #-->Blue
    ]
)
```

문제190. 문제189번의 data행렬의 shape를 확인하시오.

```
보기)
print(data.shape)
(3, 5, 5)

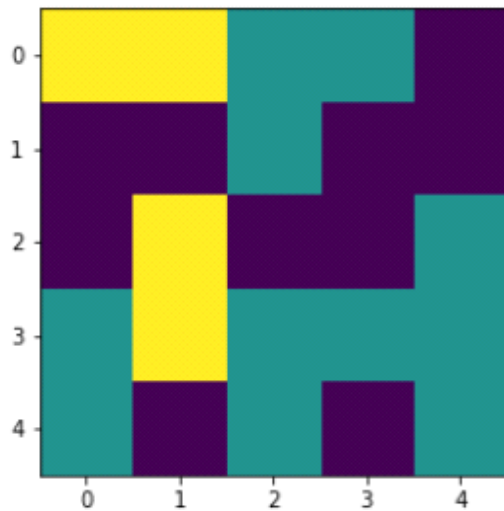
색상, 가로,세로
```

문제191. 위의 data행렬에서 Red행렬만 출력하시오.

```
답)
data = np.array(
    [
        [[2, 2, 1, 1, 0],
         [0, 0, 1, 0, 0],
         [0, 2, 0, 0, 1],
         [1, 2, 1, 1, 1],
         [1, 0, 1, 0, 1]], #-->Red
        [[2, 0, 0, 0, 1],
         [0, 2, 2, 0, 1],
         [0, 0, 0, 0, 2],
         [0, 1, 2, 0, 1],
         [2, 0, 2, 2, 2]], #-->Green
        [[4, 2, 1, 2, 1],
         [0, 1, 0, 4, 2],
         [3, 0, 6, 2, 3],
         [4, 2, 4, 5, 4],
         [0, 1, 2, 0, 1]] #-->Blue
    ]
)
print(data[0,:,:])
plt.imshow(data[0,:,:])
```

```
[[2 2 1 1 0]
 [0 0 1 0 0]
 [0 2 0 0 1]
 [1 2 1 1 1]
 [1 0 1 0 1]]
```

<matplotlib.image.AxesImage at 0x15a5f566b70>



문제192. 아래의 numpy array의 행렬을 확인하시오.

```
보기)
Filter=np.array([
    [[1,1,-1,-1,0,0,1,1,0],
     [-1,-1,0,0,-1,1,0,-1,0],
     [-1,1,1,-1,1,-1,0,0,-1]]
])

print(Filter.shape)
print(Filter.ndim) #3차원

(1, 3, 9)
3
```

문제193. 위에 행렬을 (3,3,3)행렬로 변환하시오.

```
답)
Filter=np.array([
    [[1,1,-1,-1,0,0,1,1,0],
     [-1,-1,0,0,-1,1,0,-1,0],
     [-1,1,1,-1,1,-1,0,0,-1]]
]).reshape(3,3,3)

print(Filter.shape)
print(Filter)
```

```

(3, 3, 3)
[[[ 1  1 -1]
   [-1  0  0]
   [ 1  1  0]]

  [[-1 -1  0]
   [ 0 -1  1]
   [ 0 -1  0]]

  [[-1  1  1]
   [-1  1 -1]
   [ 0  0 -1]]]

```

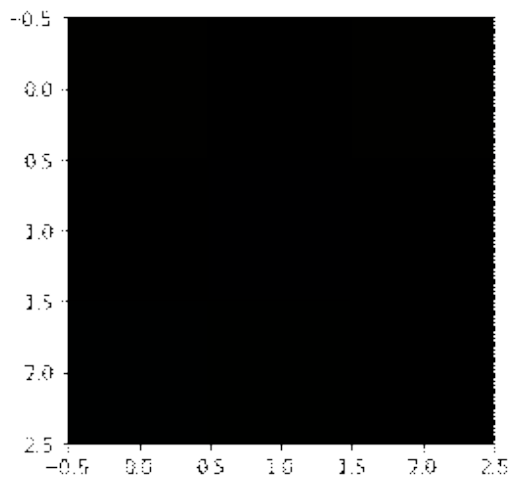
문제194. 위에 (3,3,3)행렬을 시각화 하시오.

답)

```

Filter=np.array([
    [[1,1,-1,-1,0,0,1,1,0],
     [-1,-1,0,0,-1,1,0,-1,0],
     [-1,1,1,-1,1,-1,0,0,-1]]
]).reshape(3,3,3)
plt.imshow(Filter)

```



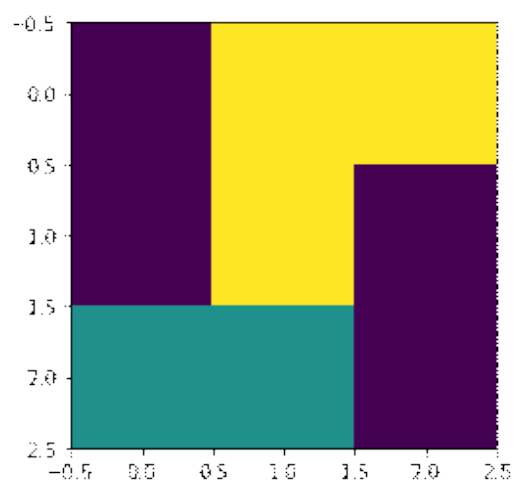
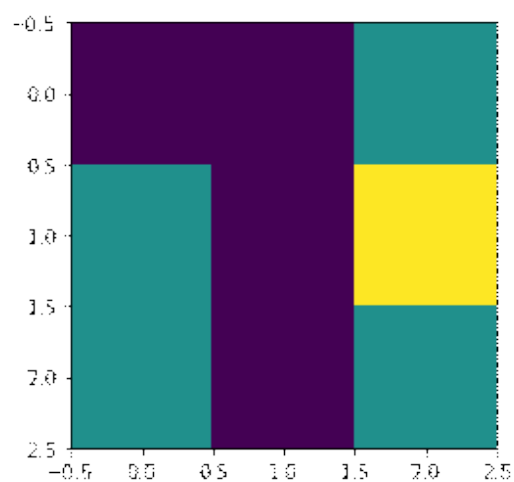
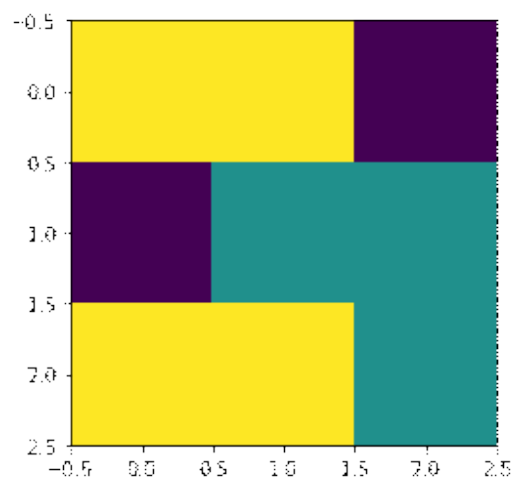
문제195. 위에 (3,3,3) 행렬의 R, G, B 행렬을 각각 시각화 하시오.

답)

```

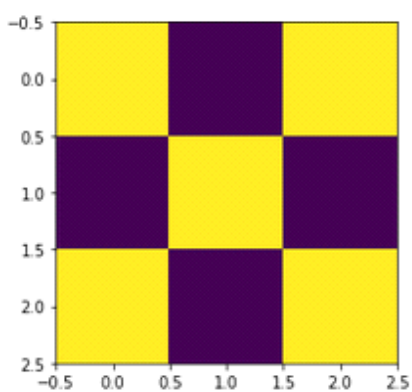
plt.imshow(Filter[0,:]) #레드
plt.imshow(Filter[1,:]) #그린
plt.imshow(Filter[2,:]) #블루

```



문제196. 아래의 필터를 생성하시오.

보기)



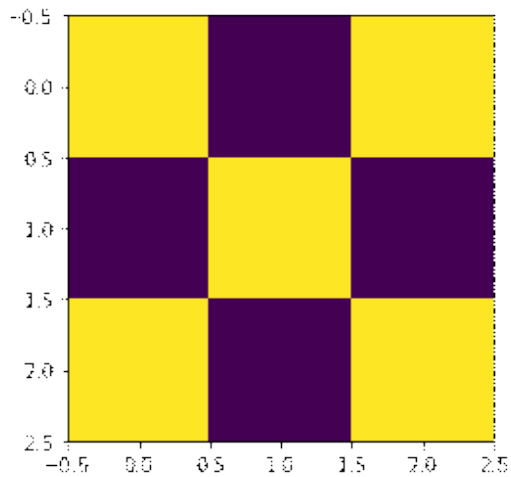


1 : 노랑  
 -1 : 보라  
 0 : 초록

답)

```
Filter=np.array([[[1,0,1],[0,1,0],[1,0,1]])
```

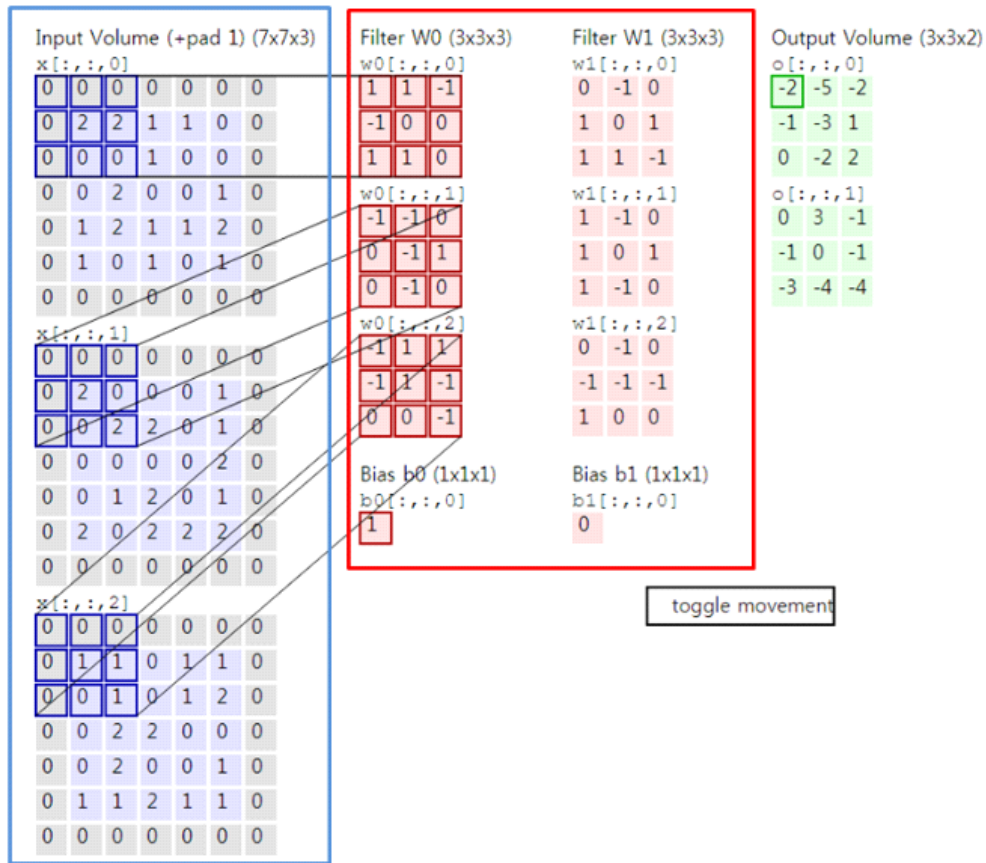
```
plt.imshow(Filter[0,:,:])
```



문제197. 아래의 원본 이미지인 data행렬과 Filter행렬을 3차원 합성곱 하기 위한 그림을 이해하시오.

보기)

```
data = np.array(
[
  [[2, 2, 1, 1, 0],
   [0, 0, 1, 0, 0],
   [0, 2, 0, 0, 1],
   [1, 2, 1, 1, 1],
   [1, 0, 1, 0, 1]], #-->Red
  [[2, 0, 0, 0, 1],
   [0, 2, 2, 0, 1],
   [0, 0, 0, 0, 2],
   [0, 1, 2, 0, 1],
   [2, 0, 2, 2, 2]], #-->Green
  [[4, 2, 1, 2, 1],
   [0, 1, 0, 4, 2],
   [3, 0, 6, 2, 3],
   [4, 2, 4, 5, 4],
   [0, 1, 2, 0, 1]] #-->Blue
])
print(data[0,:,:])
```



문제198. 아래의 원본이미지의 zero padding 1을 한 결과를 출력하시오.

보기)

```
data = np.array([
    [[2, 2, 1, 1, 0],
     [0, 0, 1, 0, 0],
     [0, 2, 0, 0, 1],
     [1, 2, 1, 1, 1],
     [1, 0, 1, 0, 1]], #-->Red
    [[2, 0, 0, 0, 1],
     [0, 2, 2, 0, 1],
     [0, 0, 0, 0, 2],
     [0, 1, 2, 0, 1],
     [2, 0, 2, 2, 2]], #-->Green
    [[4, 2, 1, 2, 1],
     [0, 1, 0, 4, 2],
     [3, 0, 6, 2, 3],
     [4, 2, 4, 5, 4],
     [0, 1, 2, 0, 1]] #-->Blue
])
```

답1)

```
data_pad=np.pad(data,pad_width=1, mode='constant', constant_values=0)[1:4]
print(data_pad)
```

답2)

```
data_pad=np.pad(data,pad_width=((0,0),(1,1),(1,1)), mode='constant', constant_values=0)
```

설명 : R,G,B 각각 제로패딩을 해줘야한다. 안써주면 전체 행렬에도 제로 패딩이 들어간다.

문제199. 198.번의 답2)의 코드를 이해하시오.

답)

(0,0),(1,1),(1,1)

(0,0):행렬의 위, 아래 개수 (차원의 개수)

(1,1): 행렬 안에 위, 아래 개수

(1,1): 행렬 안에 왼쪽, 오른쪽 개수

문제200. 한개 zero 패딩한 Red 행렬에서 아래의 행렬만 추출하시오 !

보기)

```
[[[0 0 0 0 0 0]
  [0 2 2 1 1 0]
  [0 0 0 1 0 0]
  [0 0 2 0 1 0]
  [0 1 2 1 1 1]
  [0 1 0 1 0 1]
  [0 0 0 0 0 0]]
```

답)

```
data_pad = np.pad(data, pad_width = ((0,0),(1,1),(1,1)), mode = 'constant', constant_values = 0)
```

```
print(data_pad[0,:3,:3])
```

```
[ [0 0 0]
  [0 2 2]
  [0 0 0]]
```

문제201. 위의 3x3 행렬의 R, G, B 행렬을 각각 출력하면?

보기)

```
[[2, 2, 1, 1, 0],
 [0, 0, 1, 0, 0],
 [0, 2, 0, 0, 1],
 [1, 2, 1, 1, 1], # ---> Red
 [1, 0, 1, 0, 1]],
 [[2, 0, 0, 0, 1],
 [0, 2, 2, 0, 1],
 [0, 0, 0, 0, 2], # ----> Green
 [0, 1, 2, 0, 1],
 [2, 0, 2, 2, 2]],
 [[1, 1, 0, 1, 1],
 [0, 1, 0, 1, 2], # ----> Blue
 [0, 2, 2, 0, 0],
 [0, 2, 0, 0, 1],
 [1, 1, 2, 1, 1]]
```

답)

```
print(pad[:, :3, :3])
```

---

```
[[[0 0 0]
   [0 2 2]
   [0 0 0]]

 [[0 0 0]
   [0 2 0]
   [0 0 2]]

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

---

**문제202. 아래의 Filter 에서 아래의 행렬을 추출하시오.**

보기)

```
Filter=np.array([[[1,1,-1,-1,0,0,1,1,0],
                  [-1,-1,0,0,-1,1,0,-1,0],
                  [-1,1,1,-1,1,-1,0,0,-1]]]).reshape(3,3,3)
```

답)

```
print(Filter[0,:,:])

[[ 1  1 -1]
 [-1  0  0]
 [ 1  1  0]]
```

**문제203. 위의 3개의 행렬중 Red 행렬과 아래의 filter 의 Red 행렬과 곱을 수행하시오 !**

보기)

```
[[0 0 0]   1 1 -1   0 0 0
 [0 2 2] * -1 0 0 = 0 0 0
 [0 0 0]   1 1 1   0 0 0
```

원본이미지|Red \* Filter의 Red = ?

답)

```
pad=np.pad(data,pad_width=((0,0),(1,1),(1,1)),
            mode='constant', constant_values=0)
```

```
Filter=np.array([[[1,1,-1,-1,0,0,1,1,0],
                  [-1,-1,0,0,-1,1,0,-1,0],
                  [-1,1,1,-1,1,-1,0,0,-1]]]).reshape(3,3,3)
```

```
print(pad[0,:3,:3]*Filter[0,:,:])
```

```
[[ 0  0  0]
 [ 0  0  0]
 [ 0  0  0]]
```

**문제204. 아래의 원본 이미지 RGB 3개의 행렬과 아래의 필터 RGB 3개의 행렬을 각각 행렬곱 한후 그 원소들을 다 합친 결과 숫자 하나를 출력하시오.(3차원 합성곱연산)**

보기)

```
[[0 0 0]   1 1 -1   0 0 0
 [0 2 2] * -1 0 0 = 0 0 0
 [0 0 0]   1 1 0   0 0 0
```

```

[[0 0 0]   -1 -1 0   0 0 0
[0 2 0]   * 0 -1 1 = 0 -2 0
[0 0 2]]   0 -1 0   0 0 0

[[0 0 0]   -1 1 1   0 0 0
[0 1 1]   * -1 1 -1 = 0 1 -1
[0 0 1]]   0 0 -1   0 0 -1

```

결과)

-3

답)

```

res=0
for i in range(3):
    res+=np.sum(pad[i,:3,:3]*Filter[i,:,:])
print(res)

```

문제205. 1칸 스트라이드 한 원본이미지의 3x3행렬 RGB와 Filter RGB와의 합성곱을 구하시오.

보기)

```

[[0 0 0]   1 1 -1   0 0 0
[2 2 1]   * -1 0 0 = -2 0 0
[0 0 1]]   1 1 0   0 0 0

```

```

[[0 0 0]   -1 -1 0   0 0 0
[2 0 0]   * 0 -1 1 = 0 0 0
[0 2 2]]   0 -1 0   0 -2 0

```

```

[[0 0 0]   -1 1 1   0 0 0
[1 1 0]   * -1 1 -1 = -1 1 0
[0 1 0]]   0 0 -1   0 0 0

```

답)

```

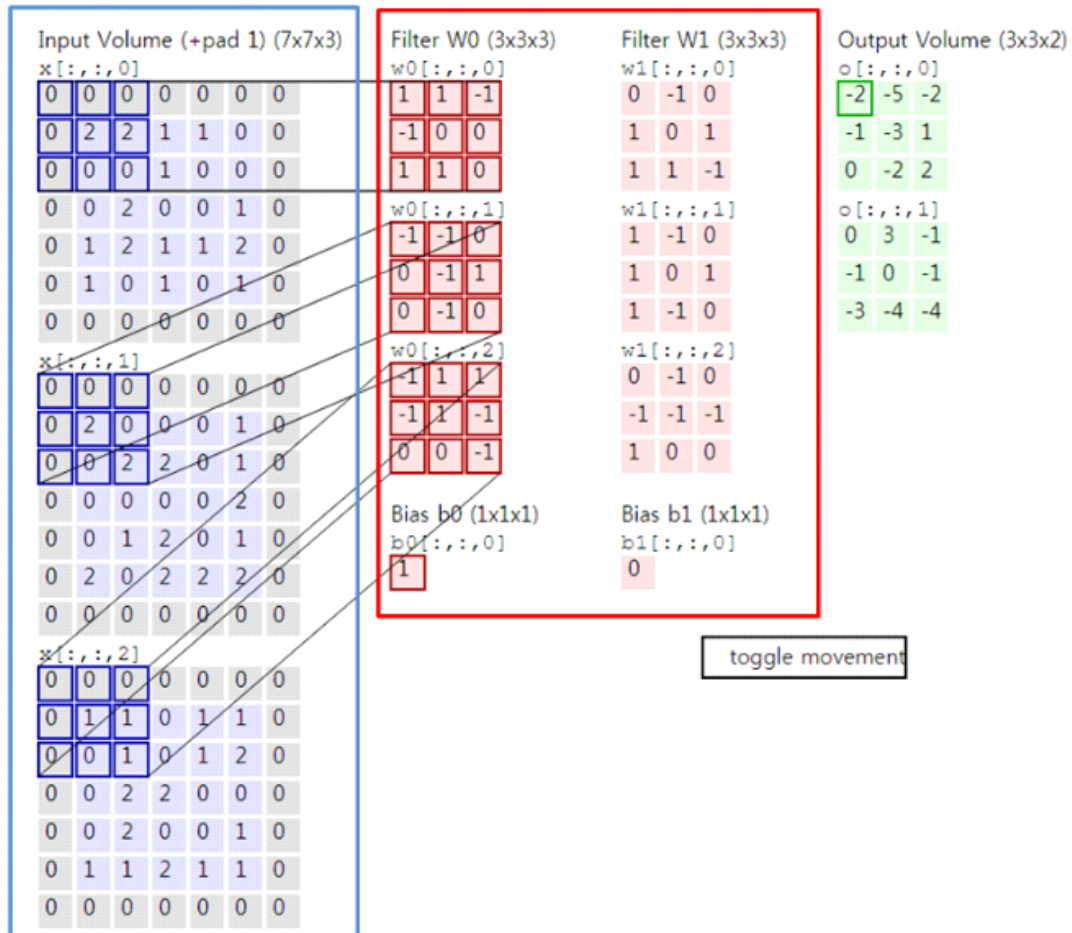
res=0
for i in range(3):
    res+=np.sum(pad[i,:3,1:4]*Filter[i,:,:])
print(res)
-4

```

문제206. 아래의 합성곱 연산(25번)을 하는 결과를 파이썬으로 구현하시오.

보기)

스트라이드 2일때 그림



답)

```
res=[]
for j in range(5):
    for k in range(5):
        res.append(np.sum(pad[:,j:j+3,k:k+3]*Filter[:, :, :]))
res=np.array(res)
res=res.reshape(5,5)
print(res)
```

설명:range(3)을 안해주는 이유가 애초에 RGB값을 전부 더하는거라 안적어도 된다.

## 합성곱 총정리

합성곱? 이미지의 특징(feature map)을 추출하는 과정)

filter(가중치)를 이용해서 추출

원본 이미지 1장 \* 필터50개 = feacture map의 개수(50개)

문제207. 아래와 같이 입력 행렬과 필터 행렬과 스트라이드와 패딩을 입력받아 출력 행렬의 shape를 출력하는 함수를 생성하시오.

```
보기)a=np.array(
    [[2, 2, 1, 1, 0],
     [0, 0, 1, 0, 0],
     [0, 2, 0, 0, 1],
     [1, 2, 1, 1, 1], # ---> Red
     [1, 0, 1, 0, 1]],
    [[2, 0, 0, 0, 1],
     [0, 2, 2, 0, 1],
```

```

[0, 0, 0, 0, 2], # ----> Green
[0, 1, 2, 0, 1],
[2, 0, 2, 2, 2]],
[[1, 1, 0, 1, 1],
[0, 1, 0, 1, 2], # ----> Blue
[0, 2, 2, 0, 0],
[0, 2, 0, 0, 1],
[1, 1, 2, 1, 1]])

Filter=np.array([[[[1,1,-1,-1,0,0,1,1,0],
                    [-1,-1,0,0,-1,1,0,-1,0],
                    [-1,1,1,-1,1,-1,0,0,-1]]]).reshape(3,3,3)
print(out(a,filter,1(스트라이드),1(패딩) ) )

```

```

답)
data = np.array(
[
    [[2, 2, 1, 1, 0],
    [0, 0, 1, 0, 0],
    [0, 2, 0, 0, 1],
    [1, 2, 1, 1, 1], # ---> Red
    [1, 0, 1, 0, 1]],
    [[2, 0, 0, 0, 1],
    [0, 2, 2, 0, 1],
    [0, 0, 0, 0, 2], # ----> Green
    [0, 1, 2, 0, 1],
    [2, 0, 2, 2, 2]],
    [[1, 1, 0, 1, 1],
    [0, 1, 0, 1, 2], # ----> Blue
    [0, 2, 2, 0, 0],
    [0, 2, 0, 0, 1],
    [1, 1, 2, 1, 1]]
])
Filter=np.array([[[[1,1,-1,-1,0,0,1,1,0],
                    [-1,-1,0,0,-1,1,0,-1,0],
                    [-1,1,1,-1,1,-1,0,0,-1]]]).reshape(3,3,3)

```

```

def out(data,F,S,P):
    data = len(data[0])
    F = len(F[0])

    return ((data + (2*P) - F) / S) + 1

print(out(data,Filter,1,1))

```

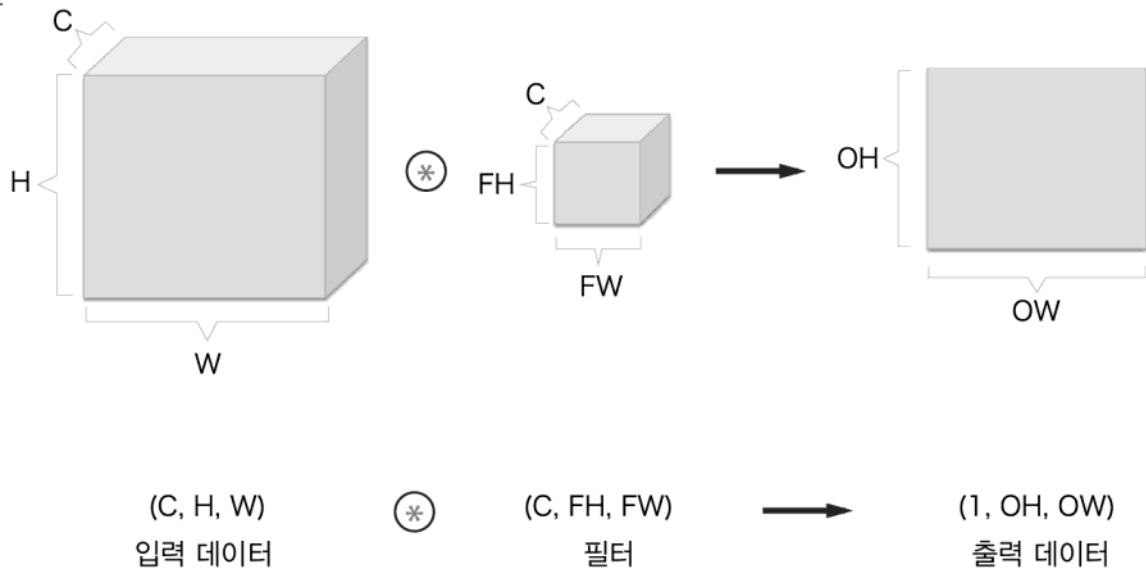
5.0

## 블록으로 생각하기

" 3차원 합성곱 연산은 데이터와 필터를 직육면체 블록이라고 생각하면 쉽다 "

블록은 3차원 직육면체(채널, 높이, 너비)로 구성됨

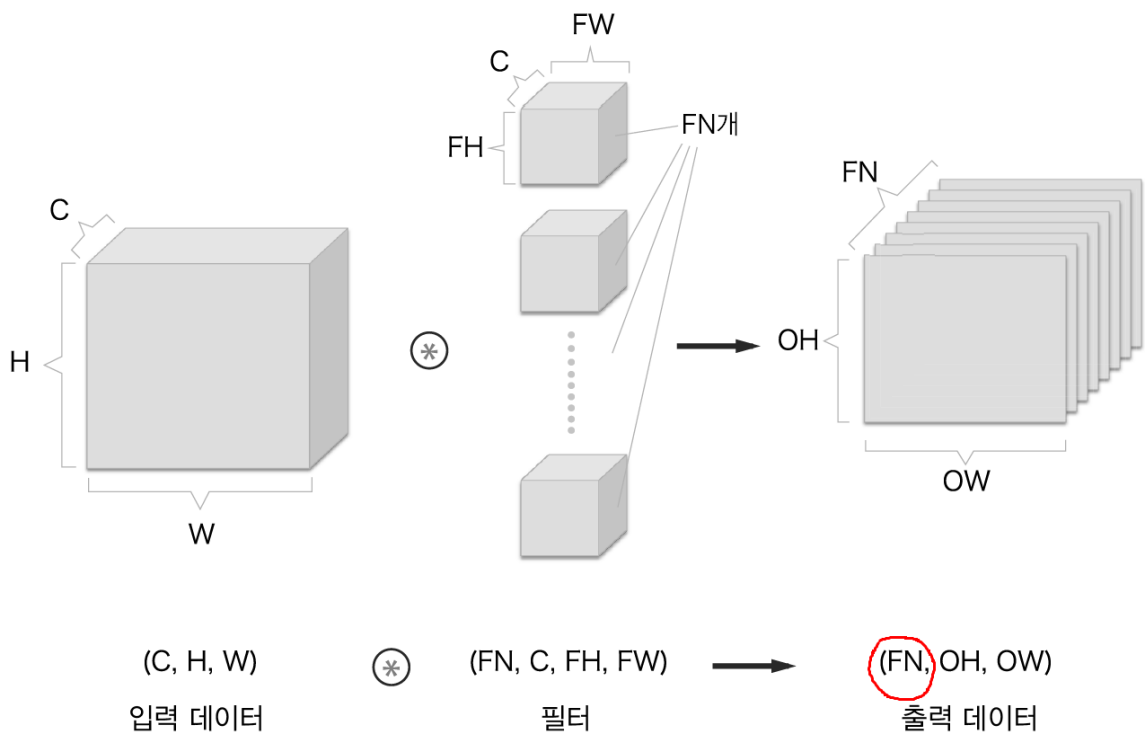
필터의 채널(C), 필터의 높이(FH), 필터의 너비(FW)



**설명 :** 아이린 사진 한장(RGB)를 RGB필터로 합성곱해서 2차원 출력행렬(feature map) 1장 출력 위의 그림은 feature map이 한 개가 나오고 있는데 실제로는 아이린사진 한장에 대해서 여러 개의 feature map이 필요하다.

**여러 개의 feature map이 출력하려면 어떻게 해야하는가?**

filter의 개수를 늘린다.



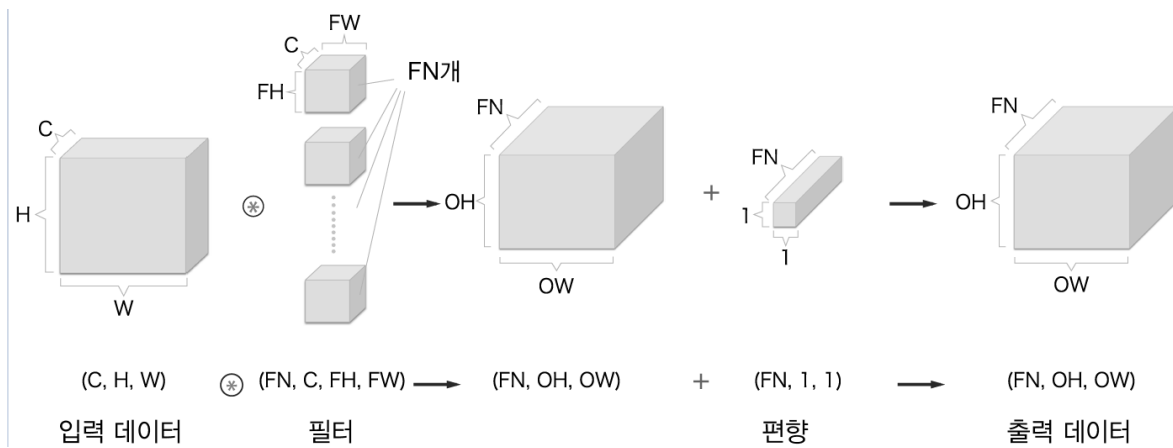
**문제208.** 설현사진 50장과 아이린 50장, 총100장의 사진을 신경망에 입력해서 설현과 아이린을 구분하는 신경망을 만든다고 할때 RGB 필터를 30개를 사용하면 출력 feature map이 총 몇 개 일까?

답)

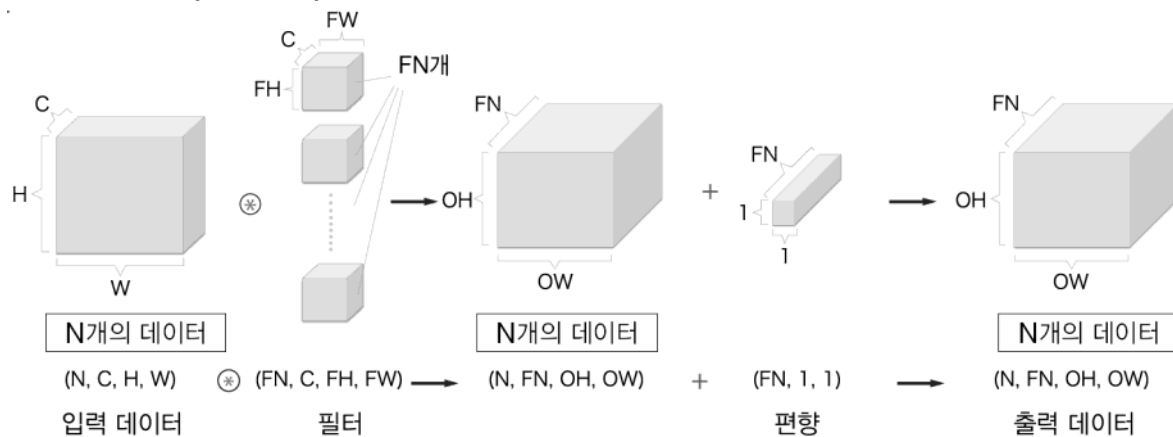
사진당 30개의 필터를 사용해서 30개의 feature map이 나오므로 총 3000장이 나온다.

합성곱 연산에서도 편향이 쓰이므로 편향을 더하면 어떤 그림일까?



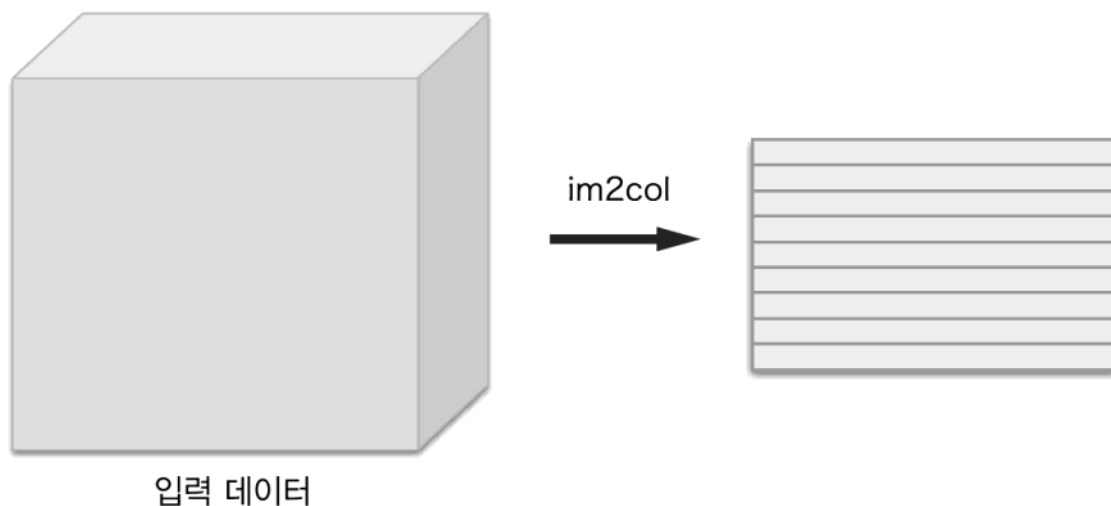


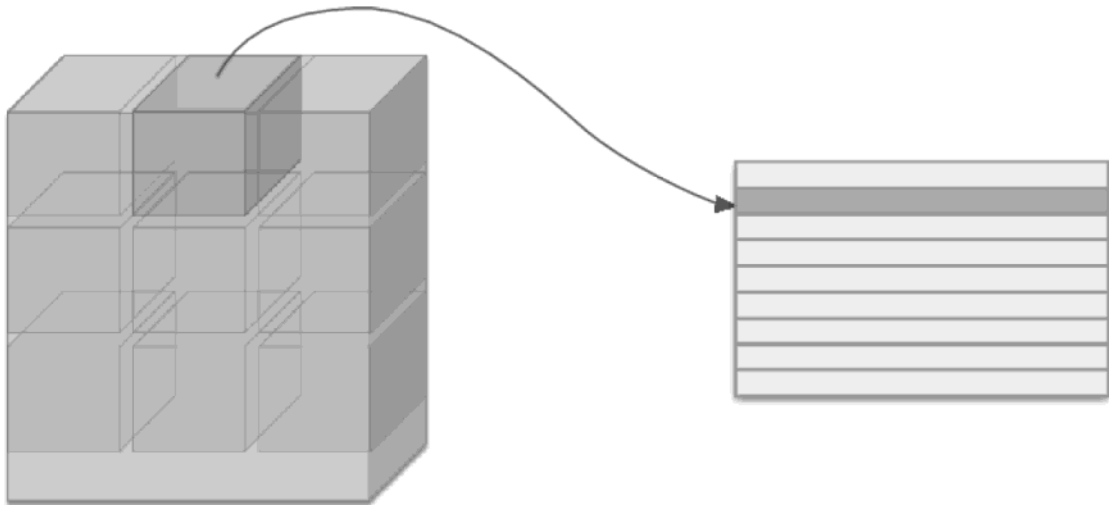
위의 그림은 이미지를 1장씩 넣어서 학습 시키는 것이므로 학습 속도가 느리므로 여러장의 이미지를 한번에 입력해서 학습(미니배치)시키면 아래의 그림이 된다.



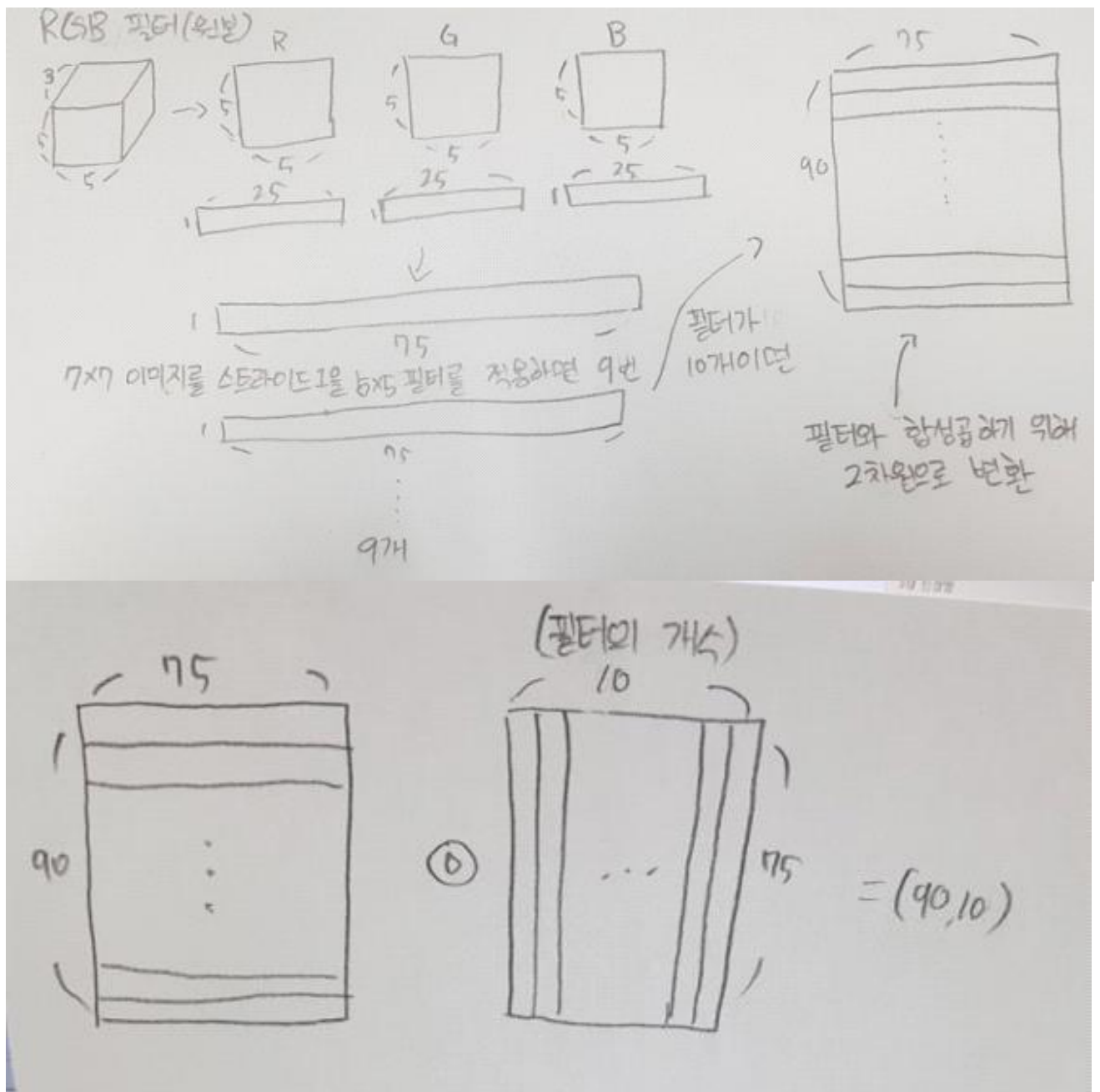
결국 합성곱 계층을 구현할때 흘러가는 행렬이 4차원 행렬이 연산이 될 것인데 그러면 연산속도가 느리므로 행렬 계산을 할때 행렬 연산을 빠르게 하려면 4차원이 아니라 2차원으로 차원 축소가 필요하다.

그래서, 필요한 함수가 im2col 함수이다. (4차원 ----> 2차원)





4차원 원본 7x7를 필터5x5와 합성곱 하기 위해 2차원으로 변환하는 방법



문제209. 칠판에 나온 아이린 사진이라 생각하고 한장의 3차원 행렬을 만드시오.(RGB의 7x7행렬 1장)

답)

```
import numpy as np
x1= np.random.rand(1,3,7,7)
print(x1)
print(x1.shape)
[[[[[0.71527961 0.78374904 0.97778614 0.46963129 0.39809458 0.54490154
      0.12392775]
      [0.28519725 0.70349217 0.84723895 0.25246234 0.20793327 0.65361657
      0.00184902]
      [0.64106858 0.89326104 0.16803063 0.22143187 0.74555975 0.07471184
      0.16067925]
      [0.16081585 0.44465189 0.08703073 0.92006692 0.24334741 0.58826449
      0.27063997]
      [0.06825168 0.60342276 0.85419573 0.75288124 0.45509926 0.17937953
      0.88161167]
      [0.46135953 0.98146419 0.61835436 0.62144005 0.09975371 0.20636637
      0.11903339]
      [0.15941074 0.22302632 0.68312768 0.39593387 0.07592954 0.8625928
      0.64305437]]
      [0.35136135 0.01264626 0.35875797 0.71621167 0.99091078 0.92042767
      0.01401373]
      [0.82405318 0.91242528 0.84173523 0.39432053 0.40755922 0.89010763
      0.87286826]
      [0.71294042 0.3587166 0.4983071 0.43078127 0.81358423 0.02329393
      0.33526389]
      [0.97146416 0.78307071 0.67254877 0.66577738 0.25483207 0.85507373
      0.41898835]
      [0.47546977 0.62878977 0.33900423 0.65410865 0.53587244 0.7961005
      0.20687199]
      [0.52356813 0.39988692 0.12150244 0.01269925 0.17607486 0.83788293
      0.15838076]
      [0.46264205 0.52532758 0.29345177 0.50831373 0.29703187 0.18060353
      0.40422919]]
      [0.84430791 0.81261396 0.02708515 0.08012923 0.44803159 0.01998091
      0.48709201]
      [0.59672429 0.42689116 0.42639461 0.78014367 0.1192418 0.74848875
      0.60410062]
      [0.03777136 0.09536662 0.38868719 0.17056413 0.35503144 0.99262665
      0.62630957]
      [0.05026218 0.50552517 0.55163998 0.97935244 0.10079732 0.23071303
      0.44665338]
      [0.7999119 0.81017178 0.75059439 0.4049688 0.64196453 0.25787766
      0.77894167]
      [0.28422276 0.37583749 0.15314497 0.33946921 0.78920063 0.4884014
      0.76078289]
      [0.39195299 0.65612697 0.12904195 0.43753384 0.59047922 0.03852708
      0.77221636]]]]
(1, 3, 7, 7)
```

문제210. im2col 함수를 이용해서 아래의 4차원을 2차원 행렬로 변경하시오.(filter는 5x5 RGB행렬을 사용)

보기)

4차원 -----> 2차원

(1,3,7,7)

(9,75)

답)

```
import numpy as np
```

```
def im2col(input_data, filter_h, filter_w, stride=1, pad=0):
```

```
    """다수의 이미지를 입력받아 2차원 배열로 변환한다(평탄화).
```

```
    Parameters
```

```
    -----
```

```
    input_data : 4차원 배열 형태의 입력 데이터(이미지 수, 채널 수, 높이, 너비)
```

```
    filter_h : 필터의 높이
```

```
    filter_w : 필터의 너비
```

```
    stride : 스트라이드
```

```
    pad : 패딩
```

```
    Returns
```

```
    -----
```

```
    col : 2차원 배열
```

```
    """
```

```
    N, C, H, W = input_data.shape
```

```
    out_h = (H + 2 * pad - filter_h) // stride + 1
```

```
    out_w = (W + 2 * pad - filter_w) // stride + 1
```

```
    img = np.pad(input_data, [(0, 0), (0, 0), (pad, pad), (pad, pad)], 'constant')
```

```
    col = np.zeros((N, C, filter_h, filter_w, out_h, out_w))
```

```
    for y in range(filter_h):
```

```
        y_max = y + stride * out_h
```

```
        for x in range(filter_w):
```

```
            x_max = x + stride * out_w
```

```
            col[:, :, y, x, :, :] = img[:, :, y:y_max:stride, x:x_max:stride]
```

```
    col = col.transpose(0, 4, 5, 1, 2, 3).reshape(N * out_h * out_w, -1)
```

```
    return col
```

```
x1= np.random.rand(1,3,7,7)
```

```
col = im2col(x1,5,5,stride=1,pad=0)
```

```
print(col1.shape)
```

```
(9, 75)
```

**문제211. 아이린 사진을 10장을 랜덤으로 생성하시오.**

답)

```
x1= np.random.rand(10,3,7,7)
```

**문제212. 아이린 사진 10장을 im2col 함수의 넣어서 2차원 행렬로 변환 하시오.(filter는 5x5 RGB행렬)**

보기)

4차원 -----> 2차원

(10,3,7,7)

(90,75)

답)

```
x1= np.random.rand(10,3,7,7)
```

```
col1 = im2col(x1,5,5,stride=1,pad=0)
```

```
print(col1.shape)
(90, 75)
```

문제213. mnist데이터 100장을 합성곱하기 편하도록 im2col 함수에 넣었을 때 나오는 출력 형상을 예상하시오.(filter의 크기 : 5x5 흑백 채널)

보기)

4차원 -----> 2차원  
(100,1,28,28) (??)

답)

```
x1= np.random.rand(100,1,28,28)
col1 = im2col(x1,5,5,stride=1,pad=0)
```

```
print(col1.shape)
```

28x28 행렬을 5x5필터로 스프라이드 1을적용하면 24x24=576 만큼 수행한다.

## 2차원으로 변경해야할 행렬 2가지?

1. 원본 이미지를 필터 사이즈에 맞게 2차원으로 변경한 행렬 (im2col함수 사용)

(10,3,7,7) ----->(90,75)

2. 4차원 필터 행렬을 2차원으로 변경 (reshape 함수의 -1이용)

(10,3,5,5)----->(75,10)

문제214. 아래의 filter를 생성하고 shape 를 확인하고 전치시키시오.

보기)

```
Filter = np.array([[[[255,255,255],[255,255,255],[0,0,0],[255,255,255],[255,255,255]],
                    [[255,255,255],[255,255,255],[0,0,0],[255,255,255],[255,255,255]],
                    [[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0]],
                    [[255,255,255],[255,255,255],[0,0,0],[255,255,255],[255,255,255]],
                    [[255,255,255],[255,255,255],[0,0,0],[255,255,255],[255,255,255]]], dtype = np.uint8)
print(Filter.shape)
print(Filter.T.shape)
```

```
(5, 5, 3)
(3, 5, 5)
```

문제215. filter (3,5,5) 행렬을 (3,25)행렬로 변환하시오.

답)

```
Filter = np.array([[[[255,255,255],[255,255,255],[0,0,0],[255,255,255],[255,255,255]],
                    [[255,255,255],[255,255,255],[0,0,0],[255,255,255],[255,255,255]],
                    [[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0]],
                    [[255,255,255],[255,255,255],[0,0,0],[255,255,255],[255,255,255]],
                    [[255,255,255],[255,255,255],[0,0,0],[255,255,255],[255,255,255]]], dtype = np.uint8)
```

```
a = Filter.T.reshape(3,-1) #reshape의 트릭
```

```
print(a.shape)
```

```
(3, 25)
```

설명 : 다차원 배열의 원소 수가 변환 후에도 똑같이 유지되도록 적절히 묶어줌

**문제216. 아래의 4차원 행렬의 Filter를 numpy의 random을 이용해서 만드시오.**

보기)  
(10,3,5,5)

답)  
import numpy as np  
Filter2=np.random.rand(10,3,5,5)  
print(Filter2.shape)  
(10, 3, 5, 5)

**문제217. 아래의 4차원 행렬을 3차원으로 변경하시오.**

보기)  
(10,3,5,5) ----->(10,3,25)

답)  
import numpy as np  
Filter2=np.random.rand(10,3,5,5)  
print(Filter2.shape)  
  
a=Filter2.reshape(10,3,-1)  
print(a.shape)  
(10, 3, 5, 5)  
(10, 3, 25)

**문제218. 아래의 3차원 행렬을 2차원행렬로 변경하시오.**

보기)  
(10,3,25) ---> (10,75)

답)  
import numpy as np  
Filter2=np.random.rand(10,3,5,5)  
print(Filter2.shape)  
a=Filter2.reshape(10,3,-1)  
print(a.shape)  
  
b=a.reshape(10,-1)  
print(b.shape)  
(10, 3, 5, 5)  
(10, 3, 25)  
(10, 75)

**위에 두과정을 한번에 할 수도 있다.**

a=Filter2.reshape(10,-1)  
print(a.shape)

**문제219. (10,75) ---> (75,10)으로 변경하시오.**

답)  
Filter2=np.random.rand(10,3,5,5)  
print(Filter2.shape)

```
a=Filter2.reshape(10,-1)
print(a.shape)
```

```
b=a.reshape(10,-1)
print(b.shape)
print(b.T.shape)
```

```
(10, 3, 5, 5)
(10, 75)
(10, 75)
(75, 10)
```

**문제220. 책 246페이지에 나오는 Convolution 클래스를 생성하시오.**

답)

```
class Convolution:
    def __init__(self, W, b, stride=1, pad=0):
        self.W=W
        self.b=b
        self.stride=stride
        self.pad=pad

    def forward(self,x):
        FN, C, FH, FW = self.W.shape
        N, C, H, W = x.shape
        out_h=int(1+(H+2*self.pad - FH) / self.stride)
        out_w=int(1+(W+2*self.pad - FW) / self.stride)

        col=im2col(x, FH, FW, self.stride, self.pad)
        col_W = self.W.reshape(FN, -1).T #필터 전개
        out = np.dot(col, col_W)+self.b

        out=out.reshape(N, out_h, out_w, -1).transpose(0,3,1,2)
        return out
```

**문제221. 위에서 만든 Convolution 클래스를 객체화 시켜서 칠판에 나온 convolution층을 구현하시오.**

답)

```
class Convolution:
    def __init__(self, W, b, stride=1, pad=0):
        self.W=W
        self.b=b
        self.stride=stride
        self.pad=pad

    def forward(self,x):
        FN, C, FH, FW = self.W.shape
        N, C, H, W = x.shape
        out_h=int(1+(H+2*self.pad - FH) / self.stride)
        out_w=int(1+(W+2*self.pad - FW) / self.stride)

        col=im2col(x, FH, FW, self.stride, self.pad)
        col_W = self.W.reshape(FN, -1).T #필터 전개
        out = np.dot(col, col_W)+self.b
        #print(out.shape) #(90,10)

        out=out.reshape(N, out_h, out_w, -1).transpose(0,3,1,2) #4차원으로 바꿔줌
```

```

return out

x1 = np.arange(1470).reshape(10,3,7,7)#10x3x7x7
W1 = np.arange(750).reshape(10,3,5,5)#10x3x5x5
b1 = 1

conv = Convolution(W1,b1)
f = conv.forward(x1)
print ('f.shape =', f.shape)

```

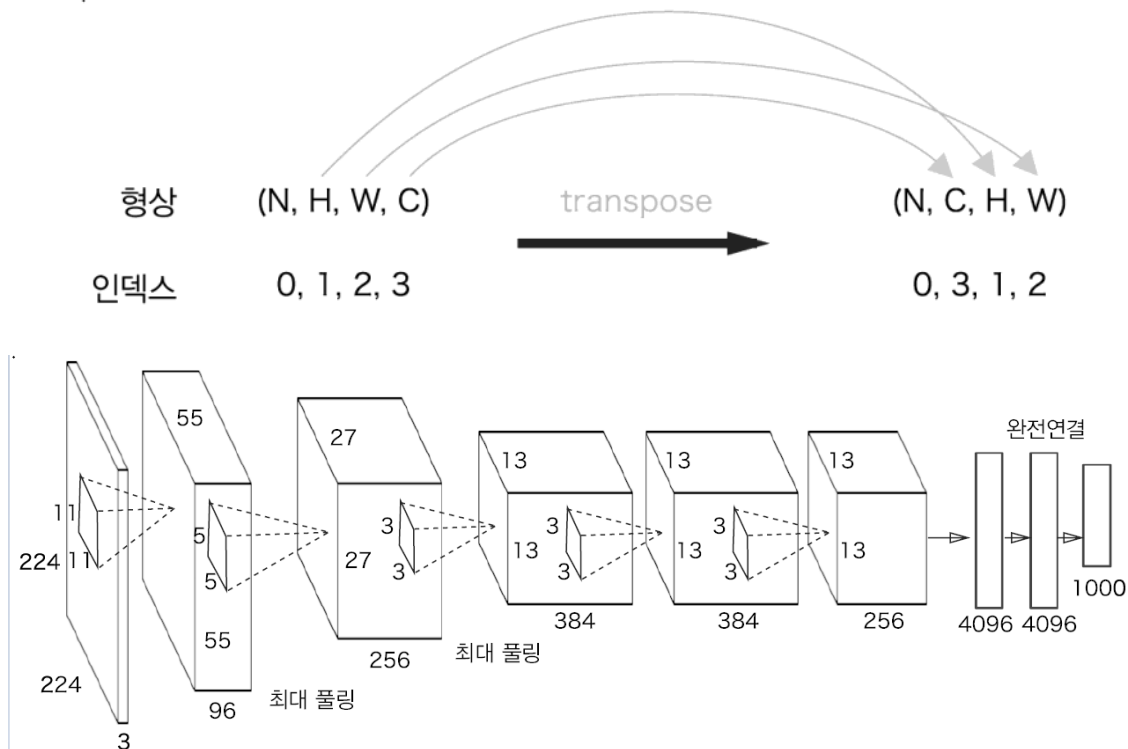
---

```

f.shape= (10, 10, 3, 3)

```

**설명 : transpose(0,3,1,2)의 의미**



## convolution 클래스내에서 일어나는 일

1. 원본이미지를 im2col로 2차원 행렬로 변경한다.
2. filter를 reshape로 2차원 행렬로 변경한다.
3. 2차원 행렬로 변경한 두 행렬을 내적한다.
4. 내적한 결과 2차원 행렬을 다시 4차원으로 변경한다.

conv ----->pooling
(이미지의 특징을                      추출한 feature map을
추출하는 층)                              선명하게 하는 층

## CNN층의 구조

conv ----> pooling ----> fully connected

## 풀링(pooling)



### 풀링 계층의 역할 : 출력 값에서 일부분만 취하는 기능

convolution이 이렇게 저렇게 망쳐놓은 그림들을 각 부분에서 대표들을 뽑아 사이즈가 작은 이미지를 만드는 것. 마치 사진을 축소하면 해상도가 좋아지는 듯한 효과와 비슷하다.

### 풀링의 종류 3가지

1. 최대 풀링 : convolution 데이터에서 가장 큰 값을 대표값으로 선정
2. 평균 풀링 : convolution 데이터에서 모든값의 평균값을 대표값으로 선정
3. 확률적 풀링 : convolution 데이터에서 임의 확률로 한 개를 선정

문제222. 아래의 이미지를 손으로 최대풀링하시오.

보기)

21	8	8	12
12	19	9	7
8	10	4	3
18	12	9	10

답)

21 12  
18 10

문제223. max\_pooling 함수를 이용해서 4x4행렬을 2x2행으로 변경하시오.

보기)

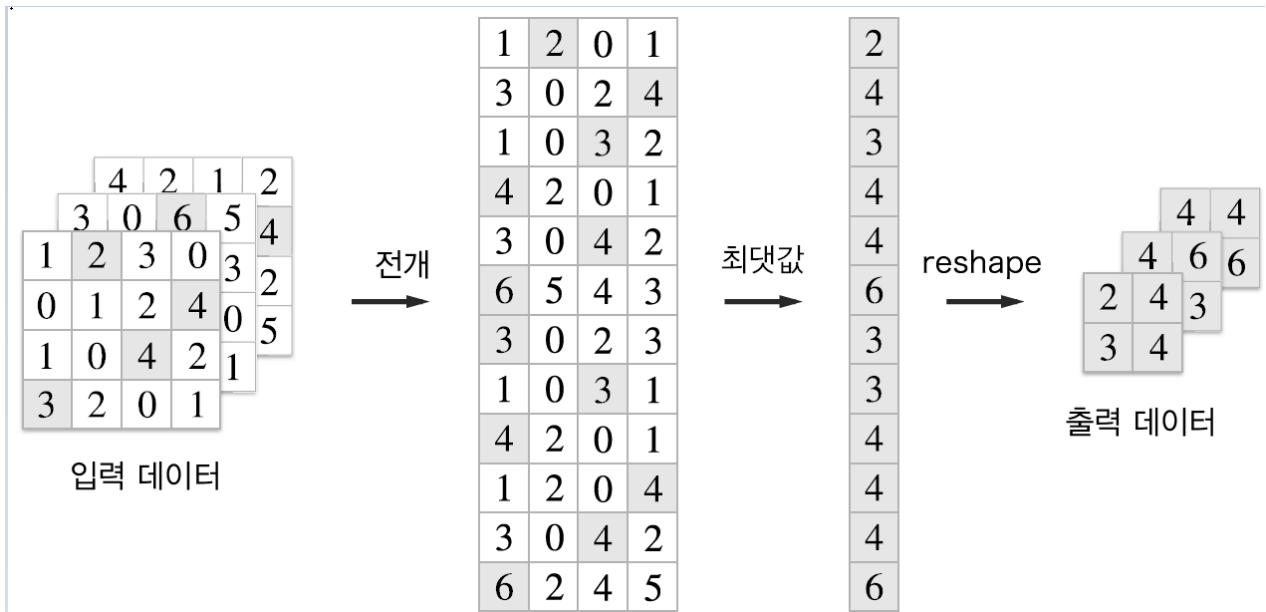
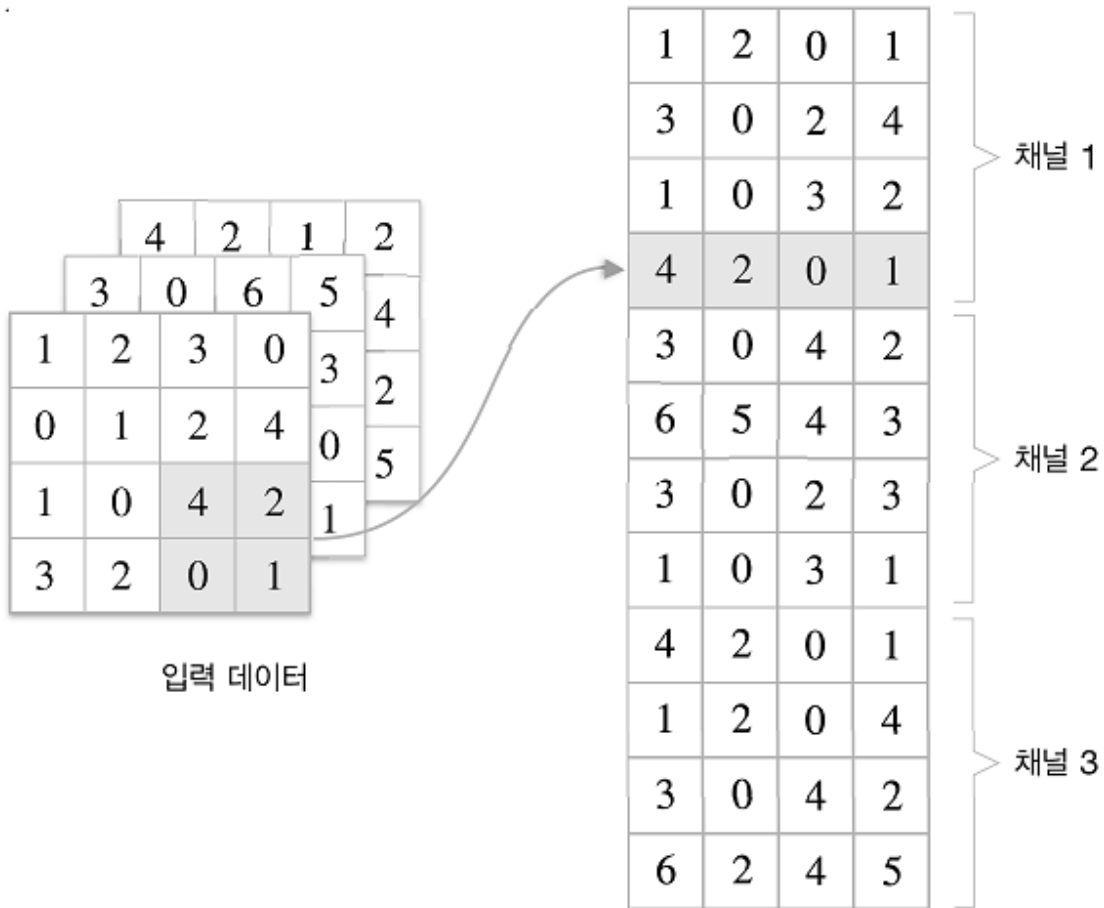
21	8	8	12
12	19	9	7
8	10	4	3
18	12	9	10

답)

```
import numpy as np
def max_pooling(array):
    res = []
    a = array.flatten()
    for i in range(0,12,2):
        if i==4 or i==6:
            continue
        temp=np.array([a[i:i+2], a[i+4:i+6]])
        res.append(np.max(temp))
    res = np.array(res).reshape(2,2)
    return res

sasa=np.array([[21,8,8,12],[12,19,9,7],[8,10,4,3],[18,12,9,10]])
print(max_pooling(sasa))
```

### 최대 풀링을 어떻게 진행하는지 그림으로 설명



문제224. 책 249 페이지의 Pooling 클래스를 생성하시오.

답)

```
class Pooling:
    def __init__(self, pool_h, pool_w, stride=1, pad=0):
        self.pool_h = pool_h
        self.pool_w = pool_w
        self.stride = stride
        self.pad = pad

    def forward(self, x):
        N, C, H, W = x.shape
```

```

out_h = int(1 + (H - self.pool_h) / self.stride)
out_w = int(1 + (W - self.pool_w) / self.stride)

#전개1
col = im2col(x, self.pool_h, self.pool_w, self.stride, self.pad)
col = col.reshape(-1, self.pool_h*self.pool_w)

#최댓값2
out = np.max(col, axis=1)

#성형3
out = out.reshape(N, out_h, out_w, C).transpose(0, 3, 1, 2)

return out

```

### convolution 층을 통과한 출력 이미지의 사이즈 계산방법?

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

OH : 출력 이미지의 높이

H : 입력이미지의 높이

P : 패딩

FH : 필터의 높이

문제225. Mnist(28x28) 데이터가 convolution층을 통과했을때 출력 이미지의 사이즈를 알아내시오.

(필터 사이즈 : 5x5, stride : 1, padding:0)

답)

$$(28 + 2*0 - 5) / 1 + 1 = 24$$

### Pooling층을 통과한 출력 이미지의 사이즈 계산 방법?

$$out\_h = \text{int}\left(\frac{1+(H-pool\_h)}{stride}\right)$$

문제226. 입력 이미지 (24x24) 행렬이 Pooling층을 통과했을때 출력되는 이미지의 크기가 어떻게 되는가?

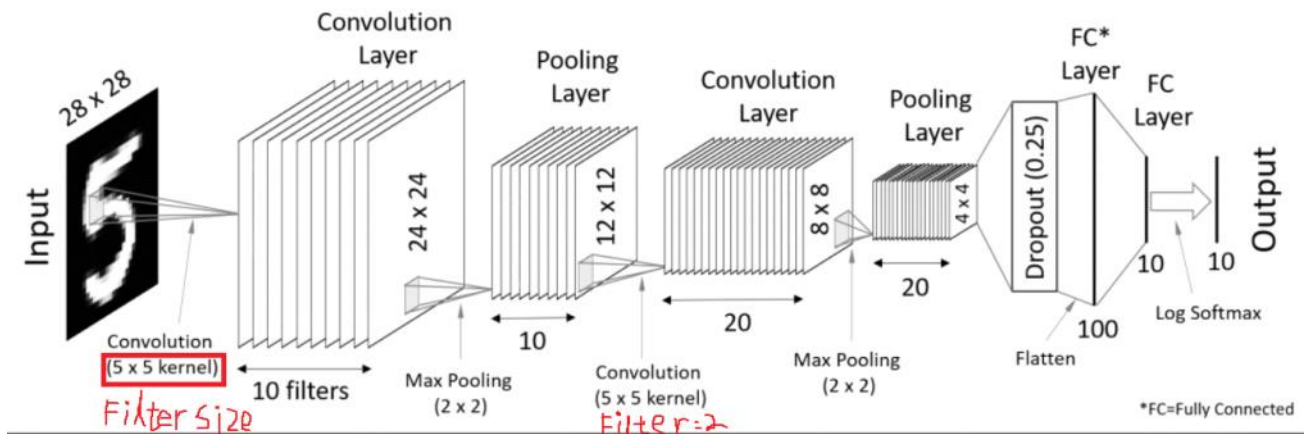
(pool\_h=2, stride=2)

답)

$$(1+(24-2)) / 2 = 11.5 \text{ 인데 } \text{int}(11.5) \text{이므로 } 12 \text{가 된다.}$$

문제227. mnist 데이터를 cnn으로 구현했을때의 신경망을 그림으로 확인하시오.

답)



문제228. 칠판에 그린 CNN 구현 코드를 구현하시오.

답)

```
# coding: utf-8
import sys, os
```

```
sys.path.append(os.pardir) # 부모 디렉터리의 파일을 가져올 수 있도록 설정
import pickle
import numpy as np
from collections import OrderedDict
from common.layers import *
from common.gradient import numerical_gradient
import matplotlib.pyplot as plt
from dataset.mnist import load_mnist
from common.trainer import Trainer
```

```
class SimpleConvNet:
```

```
    """단순한 합성곱 신경망
```

```
    conv - relu - pool - affine - relu - affine - softmax
```

```
    Parameters
```

```
    -----
    input_size : 입력 크기 ( MNIST의 경우엔 784 )
```

```
    hidden_size_list : 각 은닉층의 뉴런 수를 담은 리스트 ( e.g. [100, 100, 100] )
```

```
    output_size : 출력 크기 ( MNIST의 경우엔 10 )
```

```
    activation : 활성화 함수 - 'relu' 혹은 'sigmoid'
```

```
    weight_init_std : 가중치의 표준편차 지정 ( e.g. 0.01 )
```

```
    'relu'나 'he'로 지정하면 'He 초기값'으로 설정
```

```
    'sigmoid'나 'xavier'로 지정하면 'Xavier 초기값'으로 설정
```

```
    """
```

```
    def __init__(self, input_dim=(1, 28, 28),
```

```
                  conv_param={'filter_num': 30, 'filter_size': 5, 'pad': 0, 'stride': 1},
```

```
                  hidden_size=100, output_size=10, weight_init_std=0.01):
```

```
        filter_num = conv_param['filter_num']
```

```
        filter_size = conv_param['filter_size']
```

```
        filter_pad = conv_param['pad']
```

```
        filter_stride = conv_param['stride']
```

```
        input_size = input_dim[1]
```

```
        conv_output_size = (input_size - filter_size + 2 * filter_pad) / filter_stride + 1 #conv 공식
```

```

pool_output_size = int(filter_num * (conv_output_size / 2) * (conv_output_size / 2)) #pooling 공식

# 가중치 초기화
self.params = {}
self.params['W1'] = weight_init_std * \
    np.random.randn(filter_num, input_dim[0], filter_size, filter_size)
self.params['b1'] = np.zeros(filter_num)
self.params['W2'] = weight_init_std * \
    np.random.randn(pool_output_size, hidden_size)
self.params['b2'] = np.zeros(hidden_size)
self.params['W3'] = weight_init_std * \
    np.random.randn(hidden_size, output_size)
self.params['b3'] = np.zeros(output_size)

# 계층 생성
self.layers = OrderedDict()
self.layers['Conv1'] = Convolution(self.params['W1'], self.params['b1'],
    conv_param['stride'], conv_param['pad'])
self.layers['Relu1'] = Relu()
self.layers['Pool1'] = Pooling(pool_h=2, pool_w=2, stride=2)
self.layers['Affine1'] = Affine(self.params['W2'], self.params['b2'])
self.layers['Relu2'] = Relu()
self.layers['Affine2'] = Affine(self.params['W3'], self.params['b3'])

self.last_layer = SoftmaxWithLoss()

def predict(self, x):
    for layer in self.layers.values():
        x = layer.forward(x)

    return x

def loss(self, x, t):
    """손실 함수를 구한다.

    Parameters
    -----
    x : 입력 데이터
    t : 정답 레이블
    """
    y = self.predict(x)
    return self.last_layer.forward(y, t)

def accuracy(self, x, t, batch_size=100):
    if t.ndim != 1: t = np.argmax(t, axis=1)

    acc = 0.0

    for i in range(int(x.shape[0] / batch_size)):
        tx = x[i * batch_size:(i + 1) * batch_size]
        tt = t[i * batch_size:(i + 1) * batch_size]
        y = self.predict(tx)

```

```

        y = np.argmax(y, axis=1)
        acc += np.sum(y == tt)

    return acc / x.shape[0]

def numerical_gradient(self, x, t):
    """기울기를 구한다 (수치미분) .

    Parameters
    -----
    x : 입력 데이터
    t : 정답 레이블

    Returns
    -----
    각 층의 기울기를 담은 사전(dictionary) 변수
        grads['W1'], grads['W2'], ... 각 층의 가중치
        grads['b1'], grads['b2'], ... 각 층의 편향
    """
    loss_w = lambda w: self.loss(x, t)

    grads = {}
    for idx in (1, 2, 3):
        grads['W' + str(idx)] = numerical_gradient(loss_w, self.params['W' + str(idx)])
        grads['b' + str(idx)] = numerical_gradient(loss_w, self.params['b' + str(idx)])

    return grads

def gradient(self, x, t):
    """기울기를 구한다(오차역전파법).

    Parameters
    -----
    x : 입력 데이터
    t : 정답 레이블

    Returns
    -----
    각 층의 기울기를 담은 사전(dictionary) 변수
        grads['W1'], grads['W2'], ... 각 층의 가중치
        grads['b1'], grads['b2'], ... 각 층의 편향
    """
    # forward
    self.loss(x, t)

    # backward
    dout = 1
    dout = self.last_layer.backward(dout)

    layers = list(self.layers.values())
    layers.reverse()

```

```

for layer in layers:
    dout = layer.backward(dout)

# 결과 저장
grads = {}
grads['W1'], grads['b1'] = self.layers['Conv1'].dW, self.layers['Conv1'].db
grads['W2'], grads['b2'] = self.layers['Affine1'].dW, self.layers['Affine1'].db
grads['W3'], grads['b3'] = self.layers['Affine2'].dW, self.layers['Affine2'].db

return grads

#가중치와 바이너리 저장
def save_params(self, file_name="params.pkl"):
    params = {}
    for key, val in self.params.items():
        params[key] = val
    with open(file_name, 'wb') as f:
        pickle.dump(params, f)

#가중치와 바이너리 불러오기
def load_params(self, file_name="params.pkl"):
    with open(file_name, 'rb') as f:
        params = pickle.load(f)
    for key, val in params.items():
        self.params[key] = val

    for i, key in enumerate(['Conv1', 'Affine1', 'Affine2']):
        self.layers[key].W = self.params['W' + str(i + 1)]
        self.layers[key].b = self.params['b' + str(i + 1)]

# 데이터 읽기
(x_train, t_train), (x_test, t_test) = load_mnist(flatten=False)

# 시간이 오래 걸릴 경우 데이터를 줄인다.
# x_train, t_train = x_train[:5000], t_train[:5000]
# x_test, t_test = x_test[:1000], t_test[:1000]

max_epochs = 20

network = SimpleConvNet(input_dim=(1, 28, 28),
                        conv_param={'filter_num': 30, 'filter_size': 5, 'pad': 0, 'stride': 1},
                        hidden_size=100, output_size=10, weight_init_std=0.01)

# 매개변수 보존
network.save_params("params.pkl")
print("Saved Network Parameters!")

# 하이퍼파라미터
iters_num = 10000 # 반복 횟수를 적절히 설정한다.
train_size = x_train.shape[0] # 60000 개

```

```

batch_size = 100 # 미니배치 크기
learning_rate = 0.1
train_loss_list = []
train_acc_list = []
test_acc_list = []

# 1에폭당 반복 수
iter_per_epoch = max(train_size / batch_size, 1)
print(iter_per_epoch) # 600

for i in range(iters_num): # 10000
    # 미니배치 획득 # 랜덤으로 100개씩 뽑아서 10000번을 수행하니까 백만번
    batch_mask = np.random.choice(train_size, batch_size) # 100개 씩 뽑아서 10000번 백만번
    x_batch = x_train[batch_mask]
    t_batch = t_train[batch_mask]

    # 기울기 계산
    #grad = network.numerical_gradient(x_batch, t_batch)
    grad = network.gradient(x_batch, t_batch)
    # 매개변수 갱신

    for key in ('W1', 'b1', 'W2', 'b2'):
        network.params[key] -= learning_rate * grad[key]

    # 학습 경과 기록
    loss = network.loss(x_batch, t_batch)
    train_loss_list.append(loss) # cost 가 점점 줄어드는것을 보려고
    # 1에폭당 정확도 계산 # 여기는 훈련이 아니라 1에폭 되었을때 정확도만 체크

    if i % iter_per_epoch == 0: # 600 번마다 정확도 쌓는다.
        print(x_train.shape) # 60000, 784
        train_acc = network.accuracy(x_train, t_train)
        test_acc = network.accuracy(x_test, t_test)
        train_acc_list.append(train_acc) # 10000/600 개 16개 # 정확도가 점점 올라감
        test_acc_list.append(test_acc) # 10000/600 개 16개 # 정확도가 점점 올라감
        print("train acc, test acc | " + str(train_acc) + ", " + str(test_acc))

train acc, test acc | 0.9555, 0.9551
(60000, 1, 28, 28)
train acc, test acc | 0.9578, 0.9561

```

**오버피팅이 거의 안난다.**

**문제229. 위의 cnn 신경망에 가중치 초기화 선정인 Xavier를 적용해서 테스트 하시오.**

답)

```

def __init__(self, input_dim=(1, 28, 28),
              conv_param={'filter_num': 30, 'filter_size': 5, 'pad': 0, 'stride': 1},
              hidden_size=100, output_size=10, weight_init_std=0.01):
    filter_num = conv_param['filter_num']

```



```

filter_size = conv_param['filter_size']
filter_pad = conv_param['pad']
filter_stride = conv_param['stride']
input_size = input_dim[1]
conv_output_size = (input_size - filter_size + 2 * filter_pad) / filter_stride + 1 #conv공식
pool_output_size = int(filter_num * (conv_output_size / 2) * (conv_output_size / 2)) #pooling공식

# 가중치 초기화
self.params = {}
self.params['W1'] = np.sqrt(1/input_dim[1]) * \ #앞층의 노드수=n
    np.random.randn(filter_num, input_dim[0], filter_size, filter_size)
self.params['b1'] = np.zeros(filter_num)
self.params['W2'] = np.sqrt(1/pool_output_size) * \
    np.random.randn(pool_output_size, hidden_size)
self.params['b2'] = np.zeros(hidden_size)
self.params['W3'] = np.sqrt(1/hidden_size) * \
    np.random.randn(hidden_size, output_size)
self.params['b3'] = np.zeros(output_size)
train acc, test acc | 0.9935833333333334, 0.988
(60000, 1, 28, 28)
train acc, test acc | 0.9941, 0.9874
(60000, 1, 28, 28)
train acc, test acc | 0.9954833333333334, 0.9883

```

문제230. 위의 cnn을 이용한 3층 신경망에 배치정규화를 적용하시오.

답)

# 계층 생성

```

self.layers = OrderedDict()
self.layers['Conv1'] = Convolution(self.params['W1'], self.params['b1'],
    conv_param['stride'], conv_param['pad'])
self.layers['BatchNorm1']=BatchNormalization(gamma=1.0, beta=0.)
self.layers['Relu1'] = Relu()
self.layers['Pool1'] = Pooling(pool_h=2, pool_w=2, stride=2)
self.layers['Affine1'] = Affine(self.params['W2'], self.params['b2'])
self.layers['BatchNorm2']=BatchNormalization(gamma=1.0, beta=0.)
self.layers['Relu2'] = Relu()
self.layers['Affine2'] = Affine(self.params['W3'], self.params['b3'])

self.last_layer = SoftmaxWithLoss()

```

문제231. 위의 cnn을 이용한 3층 신경망에 합성곱층과 pooling 층을 하나더 추가하시오.

보기)

conv와 pooling층 추가

conv --> relu --> pooling --> conv --> relu --> pooling --> affine1 --> relu --> affine2 --> softmax

답)

# 신경망 함수들

2018년 8월 21일 화요일 오전 11:04

```
def sigmoid(num):
    rst = ( 1/( 1+ np.exp(-num) ) )
    return(rst)

def identity_function(x):
    return(x)

def softmax(a):
    c=np.max(a)
    np_exp=np.exp(a-c)
    np_sum=np.sum(np_exp)
    res=np_exp/np_sum
    return res

def forward(w,x,now=1):
    y=np.dot(x,w)
    if now!=3:
        w=eval('W'+str(now+1))
        return forward(w,sigmoid(y),now+1)
    else: return softmax(y)

def forward(x,w,b):
    return np.dot(x,w)+b
print(forward(x,w,b))

def predict(network,x,now=1):
    W1,W2,W3=network['W1'],network['W2'],network['W3']
    b1,b2,b3=network['b1'],network['b2'],network['b3']
    w='W'+str(now)
    b=eval('b'+str(now))
    y=np.dot(x,network[w])+b
    if now!=3:
        return predict(network,sigmoid(y),now+1)
    else: return softmax(y)

def MSE(y,t):
    return 1/n*np.sum((y-t)**2)

def CEE(y,t):
    delta=1e-7
    return -np.sum(t*np.log(y+delta))

def loss_func(x):
    return x[0]**2 + x[1]**2

def numerical_gradient(f,x):
    h=1e-4 #0.00001
    grad=np.zeros_like(x) #x와 형상이 같은 배열을 생성
```

```

for idx in range(x.size): #0, 1
    tmp_val = x[idx] #x[0]

    #f(x+h) 계산
    x[idx]=tmp_val + h #3.00001
    fxh1=f(x) #3.00010|^2 + 4^2 = 25.00060001

    #f(x-h) 계산
    x[idx]=tmp_val - h #3 - 0.0001 = 2.9999^2 + 4^2 = 24.99940001
    fxh2=f(x)

    grad[idx] = (fxh1-fxh2) / (2*h)
    x[idx] = tmp_val
return grad

init_x = np.array([3.0, 4.0])

def gradient_descent(f, init_x, lr=0.01, step_num=100):
    x=init_x

    for i in range(step_num):
        grad = numerical_gradient(f,x)
        x -= lr * grad
    return x

def __init__():
    import numpy as np
    W = np.random.randn(2,3)
    print(W)

class MulLayer:
    def __init__(self):
        self.x=None
        self.y=None

    def forward(self, x,y):
        self.x=x
        self.y=y
        out=x*y
        return out

    def backward(self, dout):
        dx=dout*self.y #x와 y를 바꾼다.
        dy=dout*self.x
        return dx, dy

class AddLayer:
    def __init__(self):
        pass # 초기화가 필요없어서 사용안함

    def forward(self, x,y):
        out=x+y
        return out

```

```
def backward(self, dout):
    dx=dout*1
    dy=dout*1
    return dx, dy
```

```
class Relu:
    import numpy as np
    import copy
    def __init__(self):
        self.mask=None
```

```
def forward(self,x):
    self.mask=(x<=0)
    out=x.copy()
    out[self.mask]=0
    return out
```

```
def backward(self,dout):
    dout[self.mask]=0
    dx=dout
    return dx
```

```
class Sigmoid:
    def __init__(self):
        self.out=None
```

```
def forward(self,x):
    out=1/(1+np.exp(-x))
    self.out=out
    return out
```

```
def backward(self,dout):
    dx=dout*self.out*(1-self.out)
    return dx
```

# Affine 클래스

```
class Affine:
    def __init__(self, W, b):
        self.W = W
        self.b = b
        self.x = None
        self.dW = None
        self.db = None
```

# 순전파

```
def forward(self, x):
    self.x = x
    Y = np.dot(x, self.W) + self.b

    return Y
```

# 역전파

```
def backward(self, dY):
    dx = np.dot(dY, self.W.T)
```

```

self.dW = np.dot(X.T, dY)
self.db = np.sum(dY, axis=0)

return dx

def padding(H,S,OH,FH):
    return (S*(OH-1)+FH-H)/2

def im2col(input_data, filter_h, filter_w, stride=1, pad=0):
    """다수의 이미지를 입력받아 2차원 배열로 변환한다(평탄화).

    Parameters
    -----
    input_data : 4차원 배열 형태의 입력 데이터(이미지 수, 채널 수, 높이, 너비)
    filter_h : 필터의 높이
    filter_w : 필터의 너비
    stride : 스트라이드
    pad : 패딩

    Returns
    -----
    col : 2차원 배열
    """
    N, C, H, W = input_data.shape
    out_h = (H + 2 * pad - filter_h) // stride + 1
    out_w = (W + 2 * pad - filter_w) // stride + 1

    img = np.pad(input_data, [(0, 0), (0, 0), (pad, pad), (pad, pad)], 'constant')
    col = np.zeros((N, C, filter_h, filter_w, out_h, out_w))

    for y in range(filter_h):
        y_max = y + stride * out_h
        for x in range(filter_w):
            x_max = x + stride * out_w
            col[:, :, y, x, :, :] = img[:, :, y:y_max:stride, x:x_max:stride]

    col = col.transpose(0, 4, 5, 1, 2, 3).reshape(N * out_h * out_w, -1)
    return col

```

# argmax

2018년 8월 22일 수요일    오후 3:34

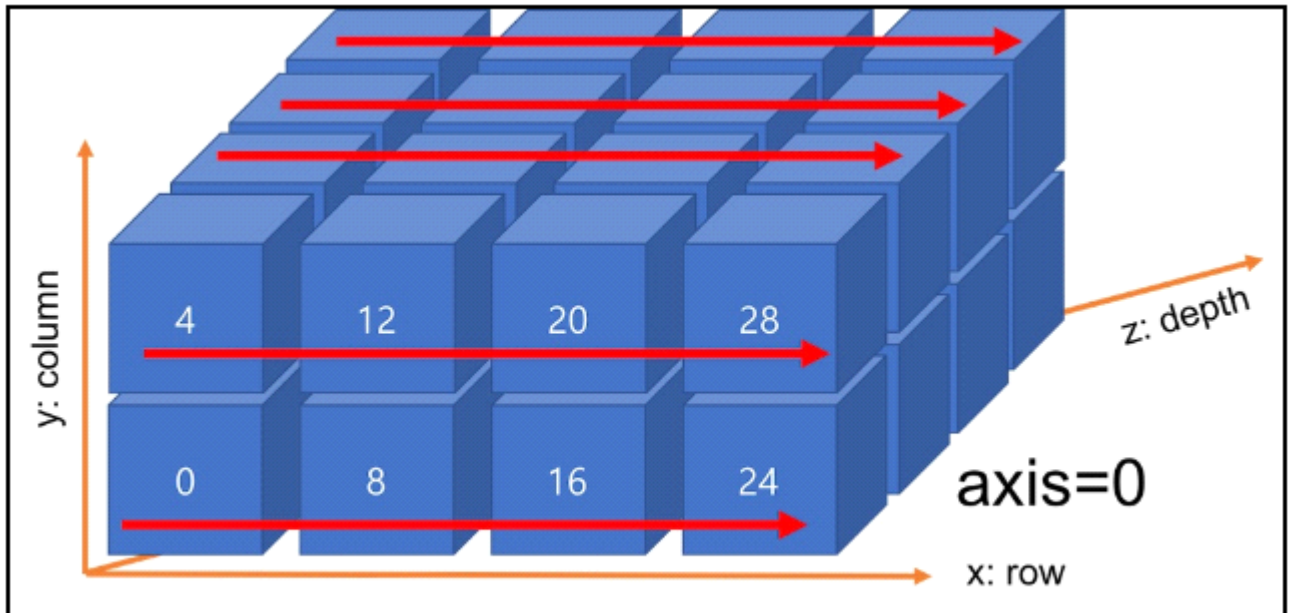


그림 4: 다차원 배열에서 axis=0의 합산 방향

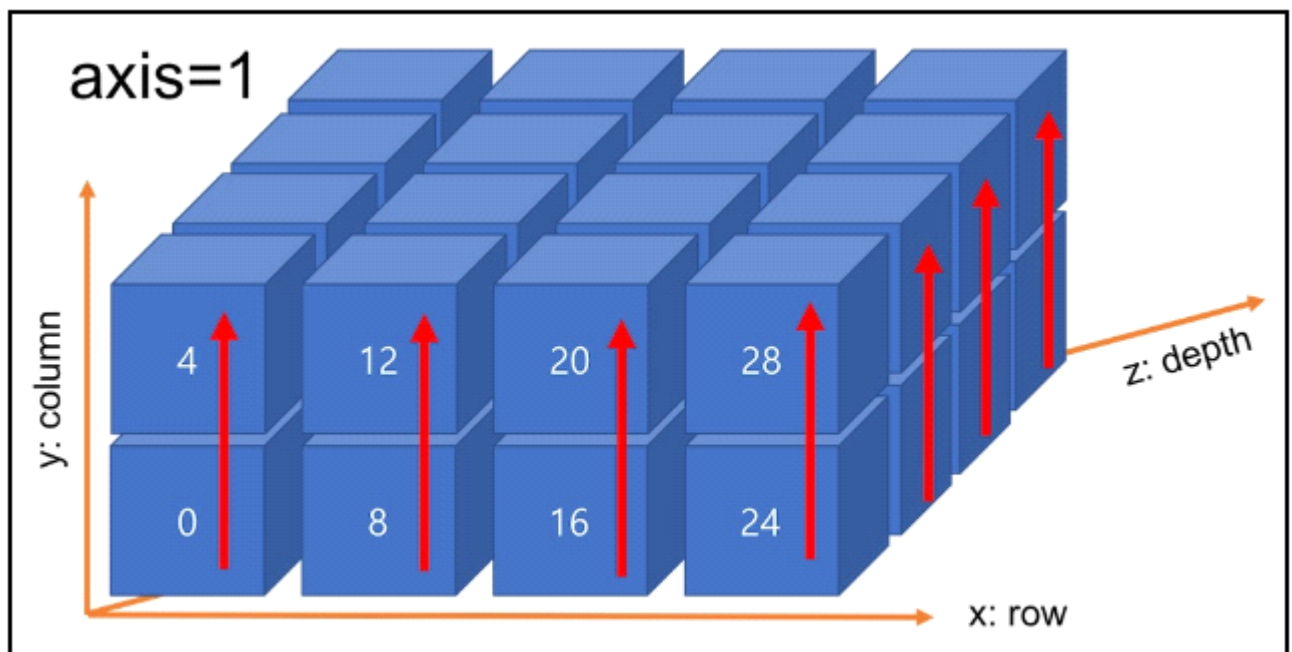


그림 6: 다차원 배열에서 axis=1의 합산 방향

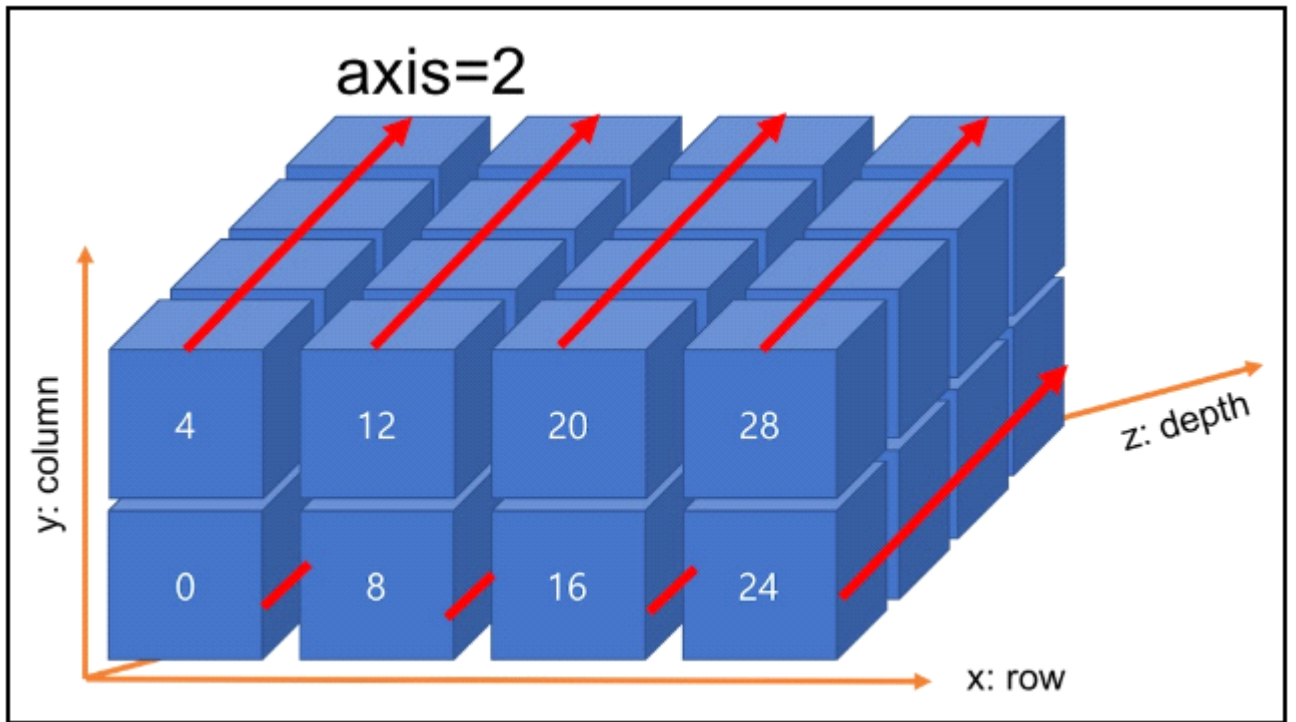


그림 8: 다차원 배열에서 axis=2의 합산 방향