00 목차

2018년 9월 27일 목요일 오후 6:03

01 딥러닝을 배우는 이유

2018년 8월 13일 월요일 오후 4:34

교재

밑바닥부터 시작하는 딥러닝

CNN 이란

인공지능의 눈? CNN

사진 속에서 사람, 동물 등을 구별할 수 있다

다음 카카오 로드뷰에 사람 얼굴이나 차 번호 등을 개인정보 보호법 상 반드시 모자이크 처리를 해야하는데 너무 많아서 이것을 사람이 일일이 할 수가 없다. 따라서 컴퓨터에게 모자이크 처리하라고 시켜야 한다. 그 때 쓴다.

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

개요

1장. 파이썬 기본 문법
2장. 퍼셉트론 (신경망의 하나의 세포를 컴퓨터로 구현)
3장. 신경망 (신경망의 활성화 함수, 3층 신경망 생성)
4장. 신경망 학습 (손실(오차) 함수, 수치 미분, 경사하강법, 학습알고리즘 구현)
5장. 오차역전파 (계산그래프, 연쇄법칙, 역전파, 단순한 계층 구현)
6장. 신경망을 학습시키는 여러 기술들 소개 (경사하강법의 종류, 배치정규화, 드롭아웃)
7장. CNN (합성곱 신경망) - 사진을 신경망에 입력해서 이 사진이 어떤 사진인지 컴퓨터가 알아맞히게 하는 방법을 구현
8장. 딥러닝의 역사
↓
텐서플로우를 이용해서 신경망을 구현
↓

강화학습

목표: "이미지를 분류할 수 있는 신경망을 구현"

1. 폐결절 사진 vs 정상 폐사진 구별 (데이터 충분)
   -> segmentation 으로 MRI 사진 분류
2. 제조업에서 만드는 제품들의 불량 여부 확인
   ex) 스노우보드 : 정상 스노우보드 vs 기스가 있는 스노우보드
   ex) 옷감 (천) : 정상 옷감 vs 기스가 있는 옷감

---

02 Numpy

2018년 8월 13일 월요일 오후 4:34

Table of Contents

넘파이 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.

넘파이(numpy) 기본문법

문제 1) 아래의 행렬을 numpy 로 만드시오

보기)

```
1 2
3 4
```

답)

```
import numpy as np

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

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

문제 2) 위의 a 행렬의 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]]
```

문제 3) 아래의 배열의 원소들의 평균값을 출력하시오

답)

```
import numpy as np

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

8.6

추가문제)

```
import numpy as np

a = np.array([1,2,4,5,5,7,10,13,18,21])
print(np.mean(a))
print(np.median(a))
print(np.max(a))
print(np.min(a))
print(np.std(a))
print(np.var(a))
```

8.6
6.0
21
1
6.437390775772433
41.44

문제 4) 아래의 행렬식을 numpy로 구현하시오

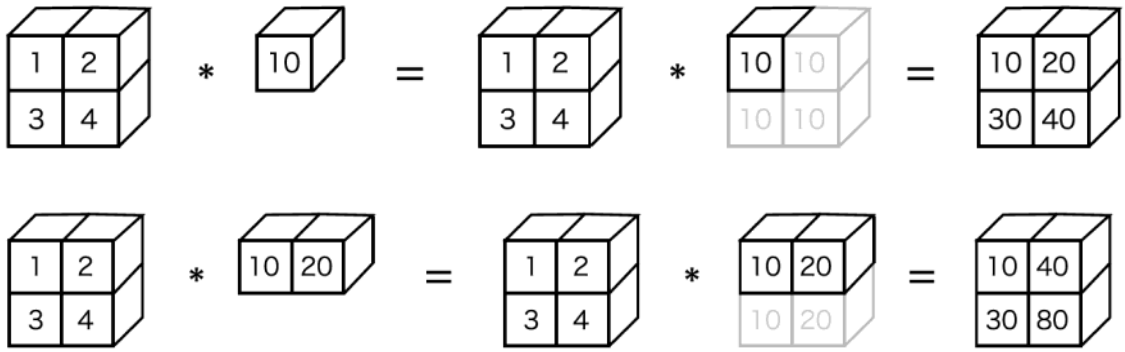
보기)

```
[[ 1  3 12]
 [ 8  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)
```

numpy 의 broadcast 기능



문제 5) 위의 그림 중 두번째 그림의 브로드캐스트를 numpy 로 구현하시오

답)

```
import numpy as np

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

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

문제 6) 아래의 행렬식을 numpy 로 구현하고 아래의 요소에서 15 이상인 것만 출력하시오

보기)

```
[[51 55]
 [14 19]
 [ 0  4]]
```

답)

```
import numpy as np

a = np.array([[51,55], [14,19], [0,4]])

# 방법 1
print('방법1:',a[a>=15])

# 방법 2
a = a.flatten() # 1차원 배열로 변환
# print(a)
b = []
for i in a:
    if i >= 15 :
        b.append(i)
print('방법2:',b)
```

방법1: [51 55 19]
방법2: [51, 55, 19]

문제 7) 위의 문제를 numpy 를 이용하지 말고 구현하시오

보기)

```
[[51 55]
 [14 19]
 [ 0  4]]
```

답)

```
m = [[51,55], [14,19], [0,4]]

row_len = len(m) # 행의 개수
col_len = len(m[0]) # 열의 개수

m2 = []
for i in range(row_len):
    for j in range(col_len):
        if m[i][j] >= 15:
            m2.append(m[i][j])
print(m2)
```

[51, 55, 19]

문제 8) 위의 문제를 numpy 를 이용하지 말고 구현하시오

보기)

```
1 3 7        0 0 5        1 3 12
1 0 0   +    7 5 0   =    8 5 0
```

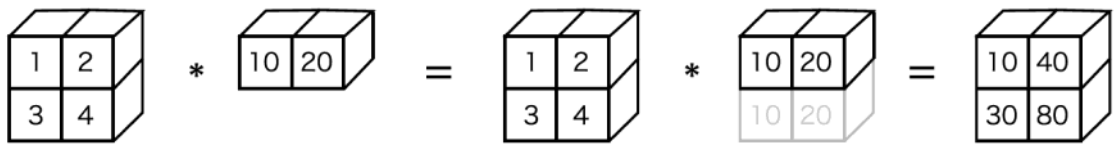
답)

```
m = [[1,3,7], [1,0,0]]
m2 = [[0,0,5], [7,5,0]]

print(m)
print(m2)

res = [[0,0,0], [0,0,0]]   # np.zeros([3,3]) 로 쉽게 만들 수도 있다.
row_len = len(m)
col_len = len(m[0])

for i in range(row_len):
    for j in range(col_len):
        res[i][j] = m[i][j] + m2[i][j]
print(res)
```

[[1, 3, 7], [1, 0, 0]]
[[0, 0, 5], [7, 5, 0]]
[[1, 3, 12], [8, 5, 0]]

문제 9) numpy 의 브로드캐스트를 사용한 연산을 numpy 를 이용하지 않고 구현해보시오

보기)



결과 : [[10,40], [30, 80]]

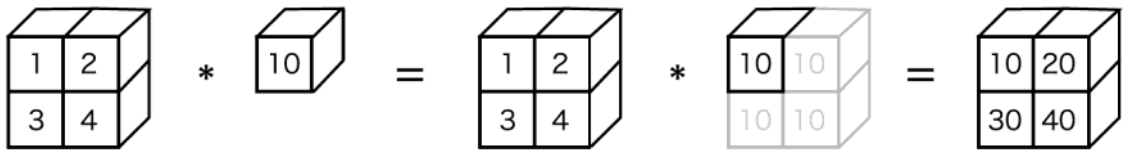
답)

```
m = [[1,2], [3,4]]
m2 = [[10,20]]

print(m)
print(m2)
```

```
res = [[0,0], [0,0]]            # np.array([2,2]) 로 쉽게 생성 가능
row_len = len(m)
col_len = len(m[0])

for i in range(row_len):
    for j in range(col_len):
        res[i][j] = m[i][j] * m2[0][j]
print(res)
```

```
[[1, 2], [3, 4]]
[[10, 20]]
[[10, 40], [30, 80]]
```

문제 10) (점심시간 문제) 아래의 브로드캐스트를 numpy 이용하지 않고 파이썬으로 구현하시오



답)

```
m = [[1,2], [3,4]]
m2 = [[10]]

print(m)
print(m2)

res = [[0,0], [0,0]]            # np.array([2,2]) 로 쉽게 생성 가능
row_len = len(m)
col_len = len(m[0])

for i in range(row_len):
    for j in range(col_len):
        res[i][j] = m[i][j] * m2[0][0]
print(res)
```

---

03 Matplotlib

2018년 8월 13일 월요일 오후 4:34

Table of Contents

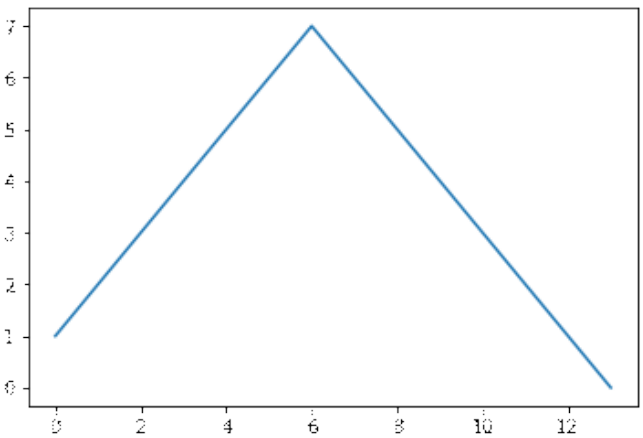
p.41

딥러닝 실험에서는 그래프 그리기와 데이터 시각화가 중요하다

matplotlib 는 그래프를 그리기 위한 라이브러리이고, 이를 이용하면 그래프 그리기가 쉬워진다

예제)

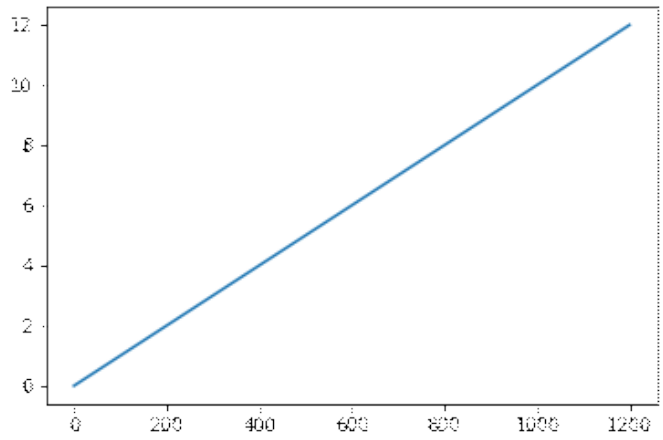
```
import matplotlib.pyplot as plt

plt.figure()     # 하나의 화면에 여러 개의 그래프를 그릴 때 필요하기 때문에 없어도 무방
plt.plot([1,2,3,4,5,6,7,6,5,4,3,2,1,0])
plt.show()
```



라인그래프

예제 2) 넘파이 배열을 이용해서 그래프 그리기

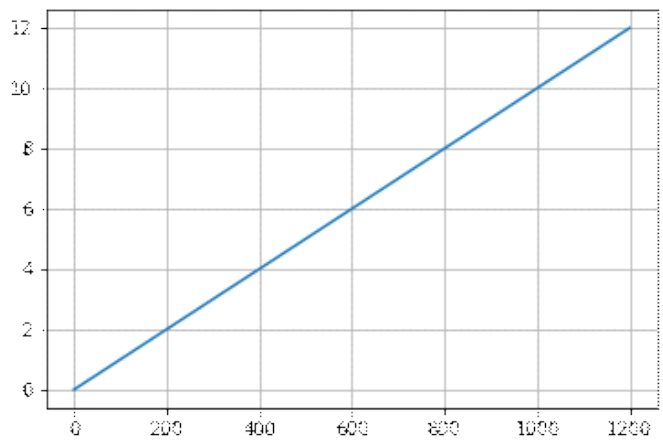
```
import matplotlib.pyplot as plt
import numpy as np

t = np.arange(0, 12, 0.01)

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



문제 11) 위의 그래프에 grid(격자)를 추가하시오

답)

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

t = np.arange(0, 12, 0.01)

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

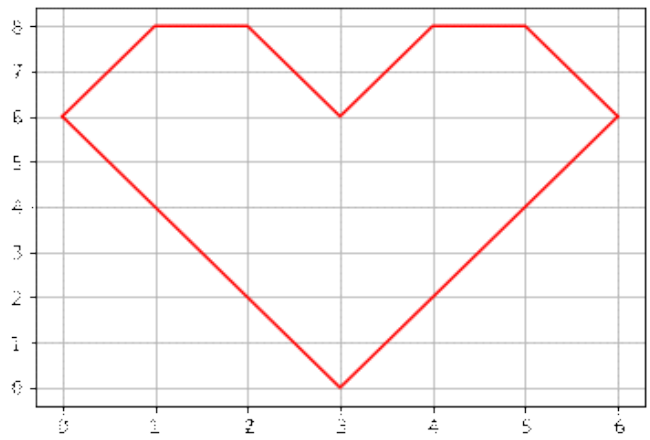


문제 12) 현수가 원섭이에 대한 마음을 시각화하시오

답)

```
import matplotlib.pyplot as plt

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

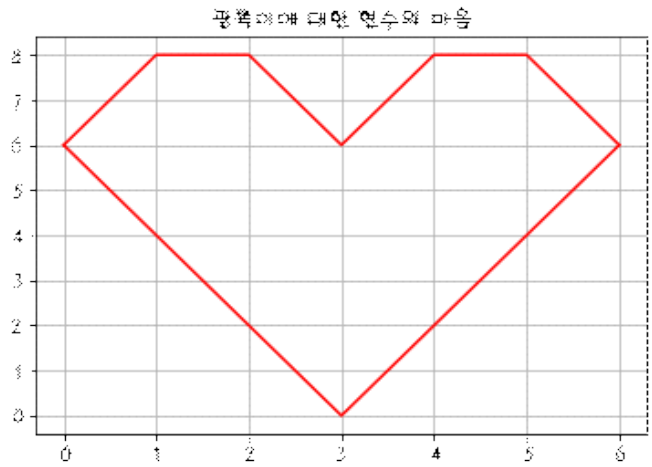
```
plt.plot([6, 8, 8, 6, 8, 8, 6], color = 'red')
plt.grid()
plt.show()
```



문제 13) 위의 그래프에서 한글로 제목을 붙이시오

답)

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

# 한글 폰트 설정
font_name = font_manager.FontProperties(fname="C:/Windows/Fonts/H2PORM.TTF").get_name()
rc('font', family=font_name)

# 그래프 그리기
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.title('광록이에 대한 현수의 마음')
plt.show()
```

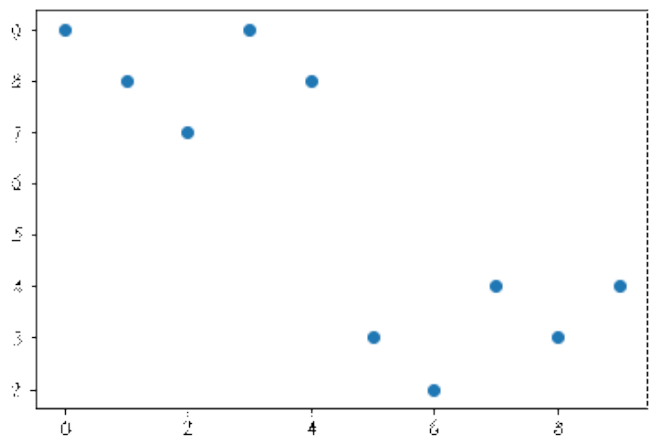


산포도그래프

문제 14) 아래의 numpy 배열로 산포도그래프를 그리시오

답)

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

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()
```

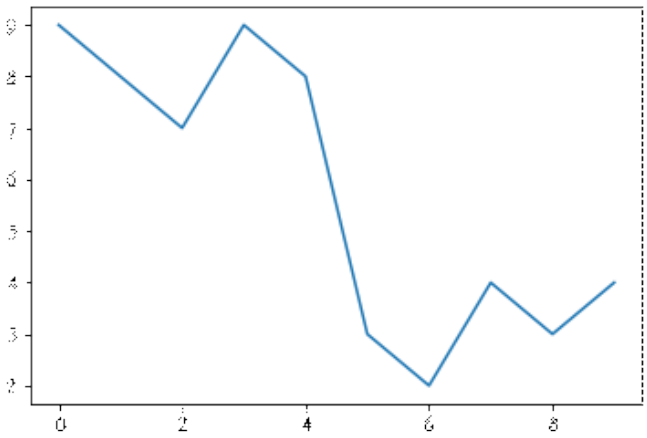


문제 15) 위의 산포도 그래프를 라인그래프로 그리시오

답)

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

```
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()
```



문제 16) 위의 그래프에 x축을 "월" 이라고 하고, y축을 "집값" 으로 라벨을 달으시오

답)

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

# 한글 폰트 설정
font_name = font_manager.FontProperties(fname="C:/Windows/Fonts/MALGUN.TTF").get_name()
rc('font', family=font_name)

# 데이터 준비
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])

# 그래프 그리기
plt.plot(x,y)
plt.title("The trend of apartment's price")
plt.xlabel("month")
plt.ylabel("price of apartment")
plt.show()
```

문제 17) 치킨집 연도별 창업건수를 가지고 라인그래프를 그리시오

답)

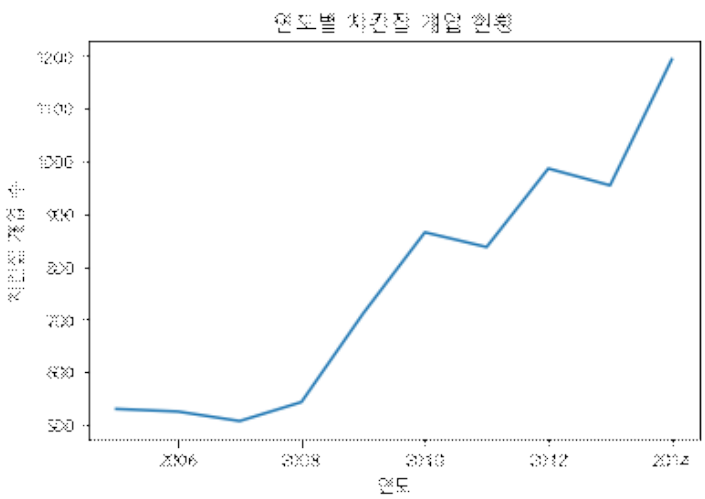
```
import numpy as np
from matplotlib import pyplot as plt

chi = np.loadtxt('C:₩₩Users₩₩kwang₩₩iCloudDrive₩₩Python₩₩Python_data₩₩창업건수.csv',
                 skiprows=1,
                 unpack=True,
                 delimiter=',')

chi2 = np.loadtxt('C:₩₩Users₩₩kwang₩₩iCloudDrive₩₩Python₩₩Python_data₩₩창업건수.csv',
                 skiprows=1,
                 unpack=False,
                 delimiter=',')

print(chi,"₩n₩n", 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.]]
```

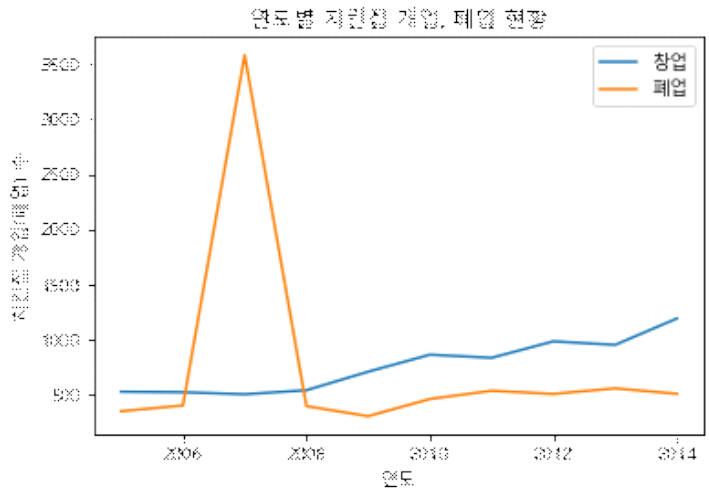
위의 두 결과를 비교해 보면 연도가 한 리스트 요소에 담기려면 위의 방법으로 해야한다.

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

chi = np.loadtxt('C:₩₩Users₩₩kwang₩₩iCloudDrive₩₩Python₩₩Python_data₩₩창업건수.csv',
                 skiprows=1,
                 unpack=True,
                 delimiter=',')

print(chi)
x = chi[0]  # x축으로 리스트에서 연도를 가져옴
y = chi[4]  # y축으로 리스트에서 치킨 집에 해당하는 5번째 컬럼을 가져옴

plt.plot(x, y)
plt.xlabel("연도")
plt.ylabel("치킨집 개업 수")
plt.title("연도별 치킨집 개업 현황")
plt.show()
```



문제 18) 아래와 같이 치킨집의 연도별 창업과 폐업 수를 겹치게 하여 그리시오

결과)



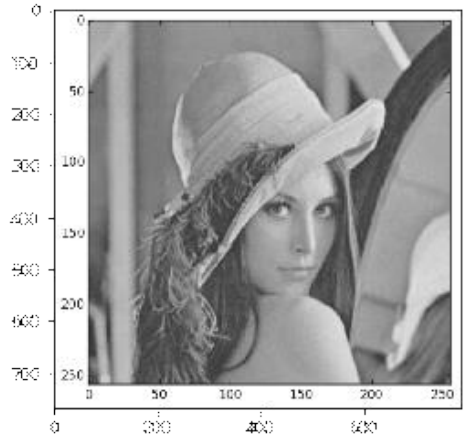
답)

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

chi = np.loadtxt('C:₩₩Users₩₩kwang₩₩iCloudDrive₩₩Python₩₩Python_data₩₩창업건수.csv',
                 skiprows=1,
                 unpack=True,
                 delimiter=',')

chi_close = np.loadtxt('C:₩₩Users₩₩kwang₩₩iCloudDrive₩₩Python₩₩Python_data₩₩폐업건수.csv',
                 skiprows=1,
                 unpack=True,
                 delimiter=',')

x = chi[0]     # x축으로 리스트에서 연도를 가져옴
y = chi[4]     # y축으로 리스트에서 치킨 집에 해당하는 5번째 컬럼을 가져옴
a = chi_close[0]
b = chi_close[4]

plt.figure(figsize=(6,4))
plt.plot(x, y, label = "창업")
plt.plot(a, b, label = "폐업")
plt.xlabel("연도")
plt.ylabel("치킨집 개업(폐업) 수")
plt.title("연도별 치킨집 개업, 폐업 현황")
plt.legend()
plt.show()
```

문제 19) 교재 44페이지의 이미지 표시를 파이썬으로 구현하시오

답)

```
import matplotlib.pyplot as plt
from matplotlib.image import imread

img = imread('C:₩₩Users₩₩kwang₩₩iCloudDrive₩₩Python₩₩Python_data₩₩lena.png')

plt.imshow(img)
plt.show()
```



문제 20) 고양이 사진을 파이썬에서 출력하시오

답)

```
import matplotlib.pyplot as plt
from matplotlib.image import imread

img = imread('C:₩₩Users₩₩kwang₩₩iCloudDrive₩₩Python₩₩Python_data₩₩cat.jpg')

plt.imshow(img)
plt.show()
```

04 퍼셉트론

2018년 8월 13일 월요일 오후 4:34

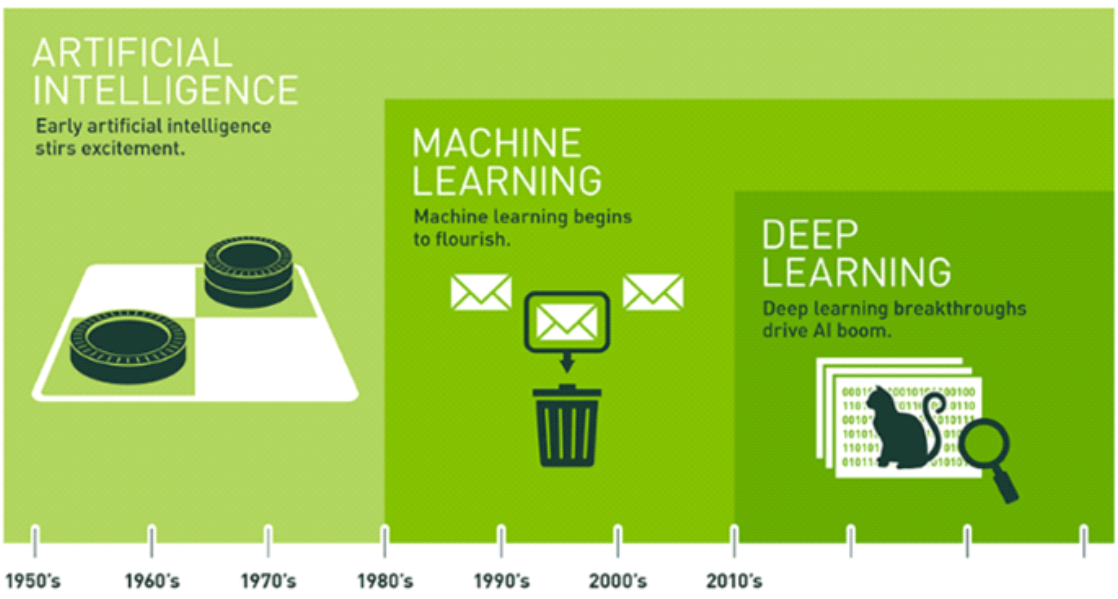
Table of Contents

머신러닝의 종류 3가지

1. 지도학습 : 정답이 있는 상태에서 학습
   예: KNN, decision tree, naivebaisys, 규칙기반, 회귀분석, 신경망

2. 비지도 학습 : 정답이 없는 상태에서 학습
   예: k-means

3. 강화학습 : 보상을 통해서 학습 데이터를 만들며 학습



Since an early flush of optimism in the 1950s, smaller subsets of artificial intelligence – first machine learning, then deep learning, a subset of machine learning – have created ever larger disruptions.

머신러닝의 한 부류인 딥러닝 (이미지를 기계가 직접 인식해서 학습)

↓

퍼셉트론 → 신경망

퍼셉트론: 인간의 뇌세포 중에 하나를 컴퓨터로 구현해 봄

1943년에 미국의 신경외과 의사인 워렌 멕컬록에 의해서 발단이 되었고
1957년에 프랑크 로젠 블라트가 퍼셉트론 알고리즘을 고안했다

사람의 뇌의 동작을 전기 스위치의 온/오프로 흉내낼 수 있다는 이론을 증명을 했다.

간단히 말해, 인간의 신경세포 하나를 흉내를 낸 것이다.

고등학교 생물 시간에 배운 3가지 용어?

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

역치

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

예: 짜게 먹는 사람은 자기가 평소에 먹는 만큼 음식이 짜지 않으면 싱겁다고 느낀다
(역치 이하의 자극 무시)
싱겁게 먹는 사람이 짜게 먹기 싫가하면 오랜시간 지나면 예전에 먹던 싱거운 음식에 만족하지 못한다
(역치가 올라감)

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.

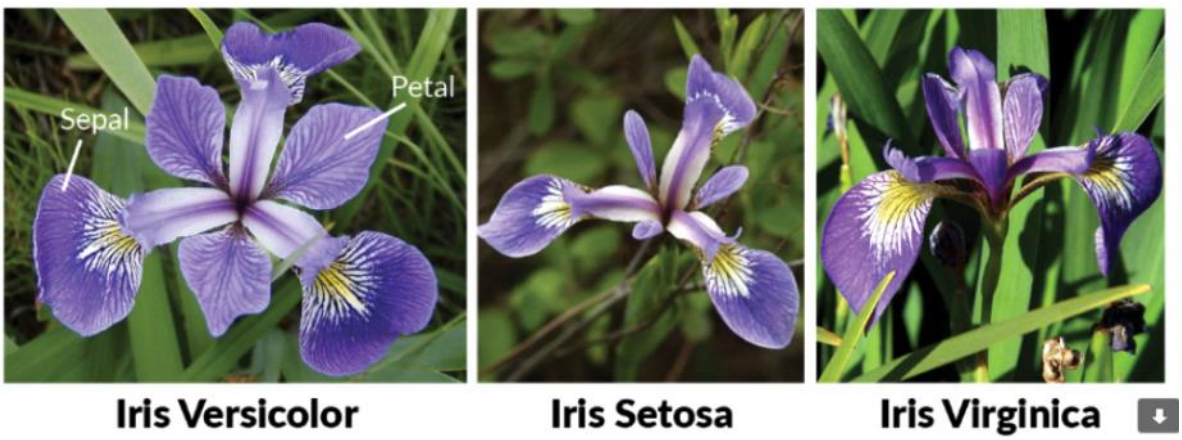
퍼셉트론의 역사

1943년에 미국의 신경외과 의사인 워렌 멕컬록에 의해서 발단이 되었고
1957년에 프랑크 로젠 블라트가 퍼셉트론 알고리즘을 고안했다
1969년 : 퍼셉트론은 단순 선형분류기에 불과하다
왜냐하면 ? XOR 분류도 못한다고 단정을 지음
침체기에 빠짐 (인공지능의 겨울기)
1970년 중반 : 중요한 논문이 발표 - 역전파 알고리즘(다층 퍼셉트론)

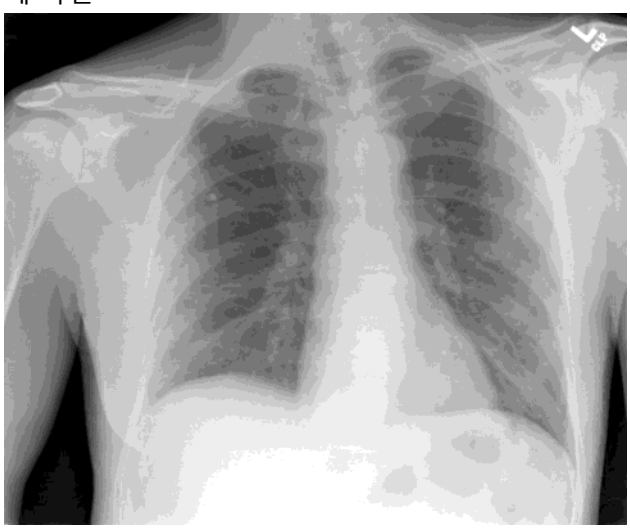
당시의 컴퓨터 연산으로는 이 이론을 구현하기가 어려웠음
1986년 : 은닉층을 갖는 다층 퍼셉트론 + 오류역전파학습 알고리즘 구현
오늘날 : 신경망을 통해서 구현하고자 하는 못표? 분류
붓꽃사진을 보고 붓꽃의 종을 분류한다

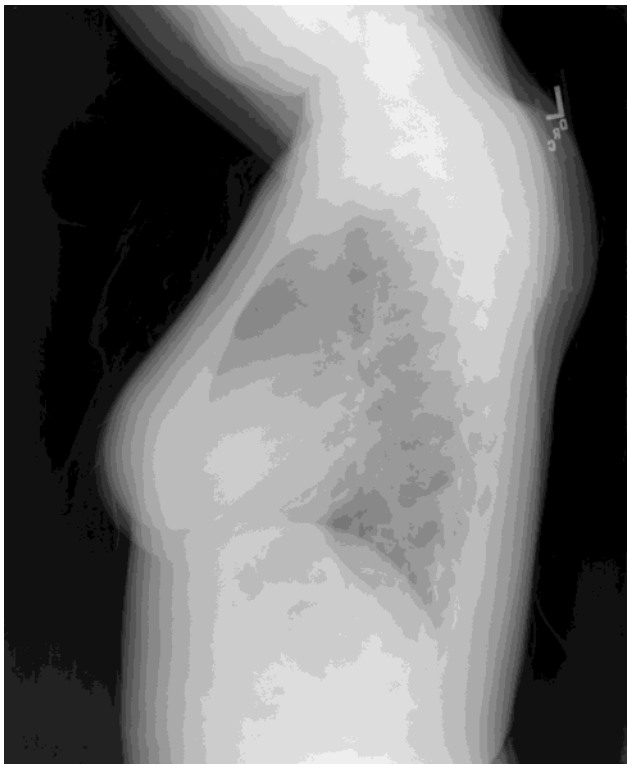
A	B	C	D	E
5.1	3.5	1.4	0.2	Iris-setosa
4.9	3	1.4	0.2	Iris-setosa
4.7	3.2	1.3	0.2	Iris-setosa
4.6	3.1	1.5	0.2	Iris-setosa
5	3.6	1.4	0.2	Iris-setosa
5.4	3.9	1.7	0.4	Iris-setosa
4.6	3.4	1.4	0.3	Iris-setosa
5	3.4	1.5	0.2	Iris-setosa
4.4	2.9	1.4	0.2	Iris-setosa
4.9	3.1	1.5	0.1	Iris-setosa

원래는 위의 액셀 파일처럼 일일이 붓꽃의 길이, 꽃받침의 길이 등등을 적어서 학습시켰지만
사실 사진을 툭 주고 학습하라고 하는 게 훨~~씬 편하다

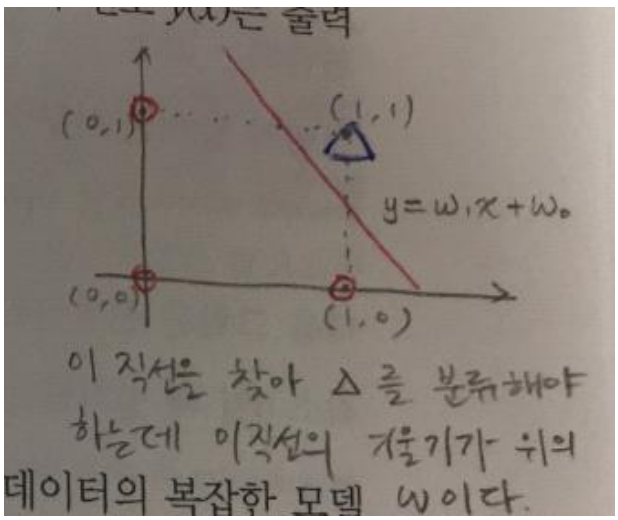


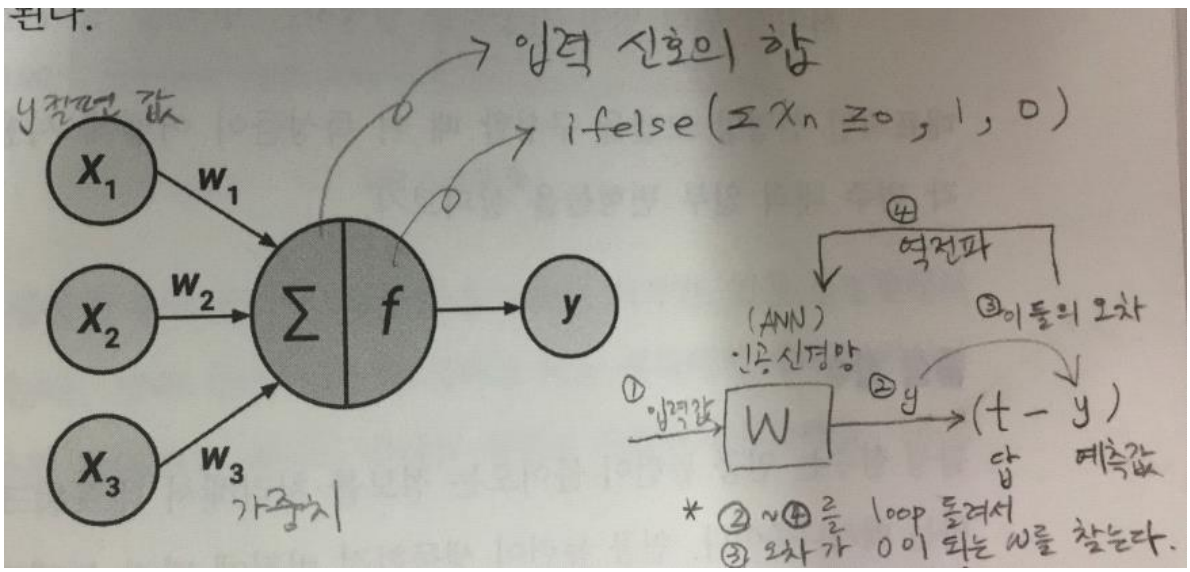
폐 사진



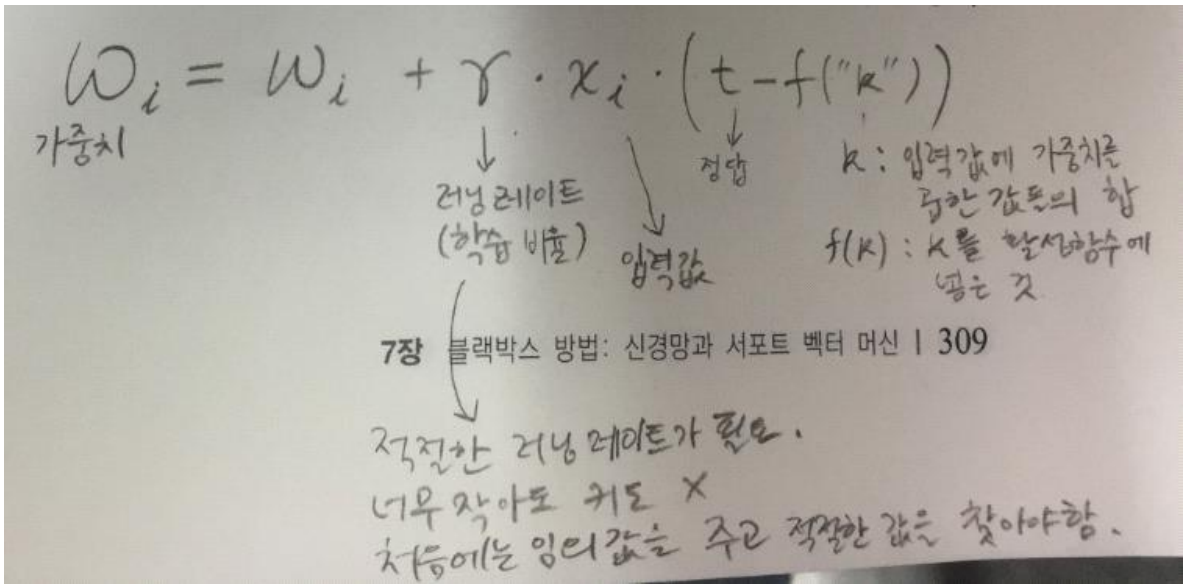


단층 퍼셉트론


(0,1)
(1,1)
y=w₁x+w₀
(0,0)
(1,0)
이 직선을 찾아 △를 분류해야
하는데 이 직선의 기울기가 위의
데이터의 복잡한 모델 w이다.


입력 신호의 합
y 실제 값
ifelse(Σxn ≥0, 1, 0)
X1
X2
X3
W1
W2
W3
가중치
Σ
f
y
④
역전파
(ANN)
인공신경망
③이들의 오차
①입력값
W
②y
(t - y)
합
예측값
* ② ~ ④를 loop 돌려서
③ 오차가 0이 되는 W를 찾는다.

이 직선의 기울기 즉, W을 구해서 세모를 분류해내는 것이 핵심이다.



P. 309

And	x_1	x_2	t
입력1	0	0	0
입력2	0	1	0
입력3	1	0	0
입력4	1	1	1

① 임의로 값들을 준다.

$w_0 = 0.3 \rightarrow 0.35$

$w_1 = 0.4 \rightarrow 0.35$

$w_2 = 0.1$

$x_0 = -1$

② 입력 1일 때,

$x_0 \cdot w_0 + x_1 \cdot w_1 + x_2 \cdot w_2 = k$

$-1 \cdot 0.3 + 0 \cdot 0.4 + 0 \cdot 0.1 = k$

$k = -0.3$

$f(k) = 0$

$w_i = w_i + \underline{\gamma \cdot x_i} \cdot \frac{(t - f("k"))}{0 - 0}$

$0.05 \cdot x_i \cdot 0$ (0)

$w_i = w_i$

∴ 가중치의 변화가 없다는 것을 확인.

③ 입력 2 일 때,

$-1 \cdot 0.3 + 0 \cdot 0.4 + 1 \cdot 0.1 = k$

$k = -0.2$

$f(k) = 0$

$w_i = w_i + \underset{0.05 \cdot x_i}{\gamma \cdot x_i} \cdot \underset{0}{(t - f("k"))}$ (0)

$w_i = w_i$

∴ 가중치의 변화가 없다는 것을 확인.

④ 입력 3인 때,

$-1 \cdot 0.3 + 1 \cdot 0.4 + 0.01 = k$

$k = 0.1$

$f(k) = 1$

$w_i = w_i + \gamma \cdot x_i \cdot (t - f("k"))$

$0.05 \cdot x_i \cdot (0 - 1)$

$w_i = w_i + (-0.05 x_i)$

가중치 값의 변화가 있다.

1) $i = 1$ 일 때, $w_1 = w_1 + 0.05 \cdot x_1 \cdot (0-1)$

$= 0.4 + 0.05 \times 1 \times (-1)$

$= 0.4 - 0.05$

$= 0.35$

2) $i = 2$ 일 때, $w_2 = \underset{0.1}{w_2} + 0.05 \cdot \underset{0}{x_2} \cdot (0-1)$

$= 0.1$

3) $i = 0$ 일 때, $w_0 = w_0 + 0.05 \cdot x_0 \cdot (0-1)$

$= 0.3 + 0.05 \times (-1) \times (-1)$

$= 0.35$

⇒ ①에 update.

⑤ 입력 4 일 때,

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

$k = 0.1$

$f(k) = 1$

$w_i = w_i + \gamma \cdot x_i \cdot (t - f(k))$

$1 - 1$

∴ 가중치의 변화가 없다는 것을 확인

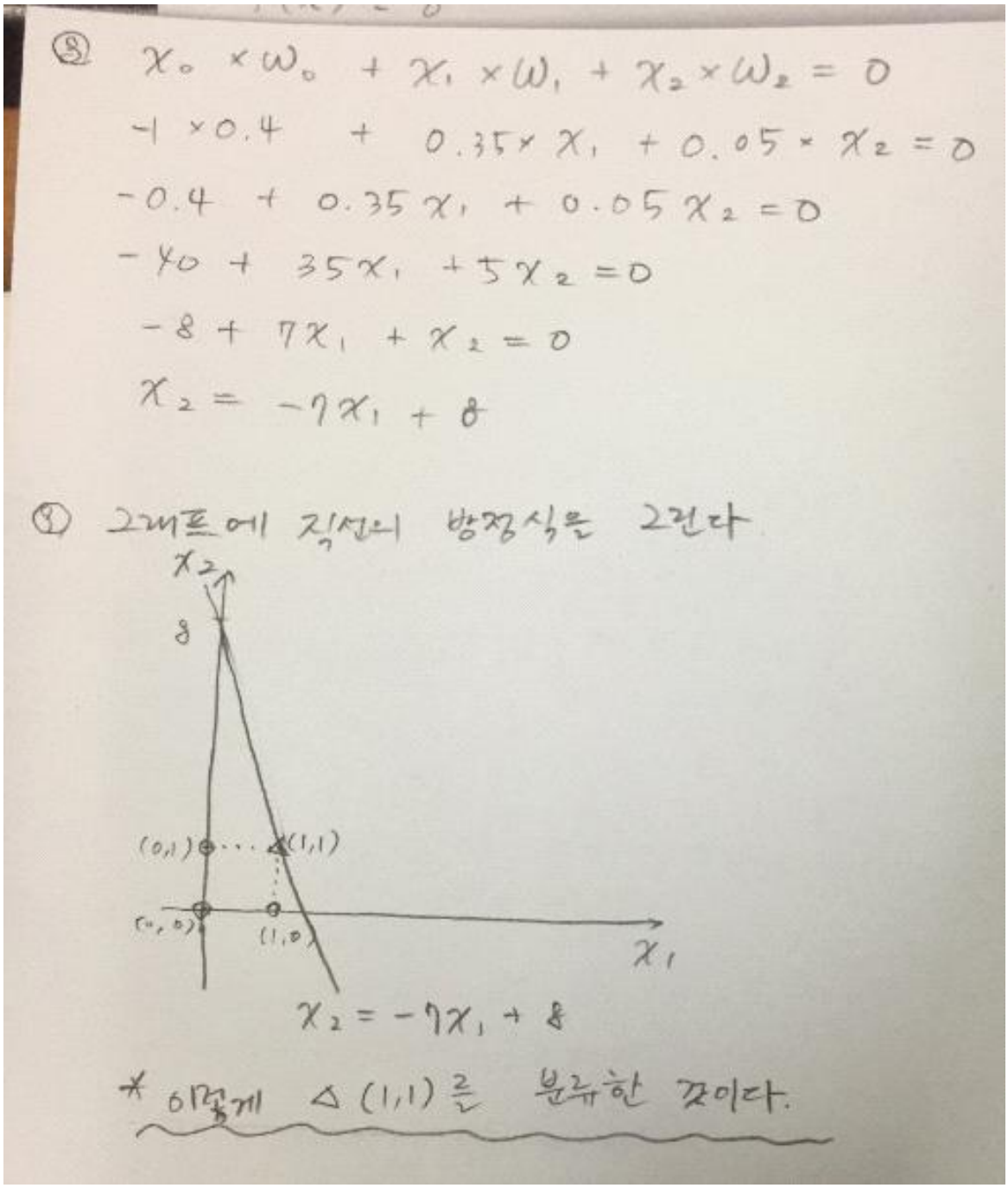
⑥ 다시 ②로 가서 ①의 값들의 변화가 없을 때까지 돌린다.

⑦ 그러면 최종적으로 $w_0 = 0.4$

$w_1 = 0.35$

$w_2 = 0.05$

$x_0 = -1$



문제 21) 아래의 4개의 딕셔너리를 이용해서 입력 값과 가중치의 곱의 합을 구하는 함수를 x_w_sum 이라는 이름으로 생성하시오

결과)

```
-0.3
-0.19999999999999998
0.10000000000000003
0.20000000000000004
```

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 = w_dic['w0'] * input_dic['x0'] + w_dic['w1'] *input_dic['x1'] + w_dic['w2'] * input_dic['x2']

    return k

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

문제 22) 위의 결과가 0보다 크거나 같으면 1을 출력하고 0보다 작으면 0을 출력하는 함수를 active_func()으로 생성하시오

결과)

```
0
0
1
1
```

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}

# x와 w의 합 함수 생성
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


# 활성화 함수
def activate_func():
    for i in range(1, 5):
        a = eval('input' + str(i) + '_dic')
        b = x_w_sum(a, w_dic)
        if b >= 0:
```

```
            print(1)
        else:
            print(0)

activate_func()
```

문제 23) 위의 코드를 이용해서 오차가 없는 가중치를 구하는 코드를 작성하시오

결과)

```
{'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}


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

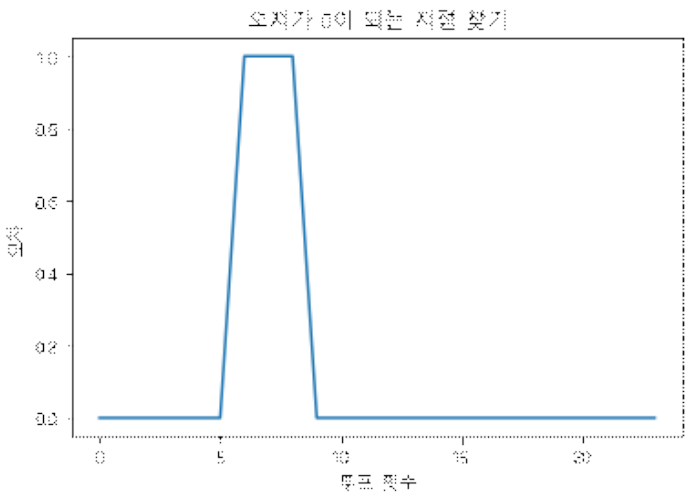
답)

```
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}

for m 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 )
        print("₩n")
```

문제 25) (점심시간 문제) 일단 위의 코드를 답글로 올리고 라인 그래프를 그리는데 t - k 의 차이 즉, 오차가 있다가 0이 되는 부분이 어디인지 알아내는 라인 그래프를 그리시오 (y축이 오차, x축이 loop 횟수)

결과)

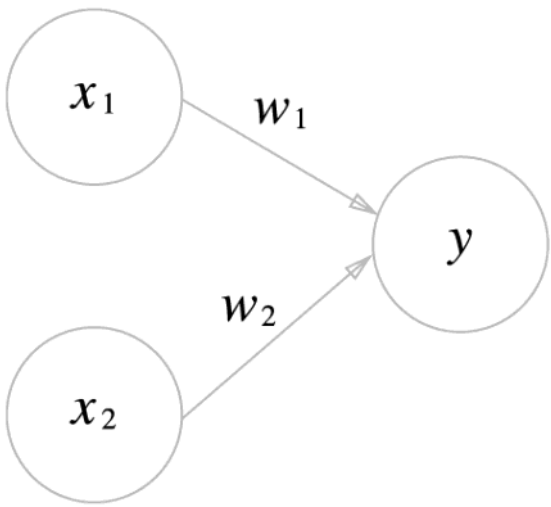


답)

```
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 = []
```

```
for m in range(2):
    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]
                res.append(abs(input_dic['t'] - active_func(input_dic, w_dic)))
                print( w_dic )
            print("₩n")


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

# 한글 폰트 설정
font_name = font_manager.FontProperties(fname="C:/Windows/Fonts/MALGUN.TTF").get_name()
rc('font', family=font_name)

# 그래프 그리기
plt.plot(res)
plt.title("오차가 0이 되는 지점 찾기")
plt.xlabel("루프 횟수")
plt.ylabel("오차")
plt.show()
```

교재에서 설명하는 퍼셉트론



입력신호의 연결강도가 가중치인데 가중치의 값이 클수록 강한 신호이다.
입력신호가 뉴런에 보내질 때는 각가의 고유한 가중치가 곱해진다.

w1*x1 + w2*x2 <= 임계값(세타) -----> 0 (신호가 흐르지 않는다)
w1*x1 + w2*x2 > 임계값(세타) -----> 1 (신호가 흐른다)

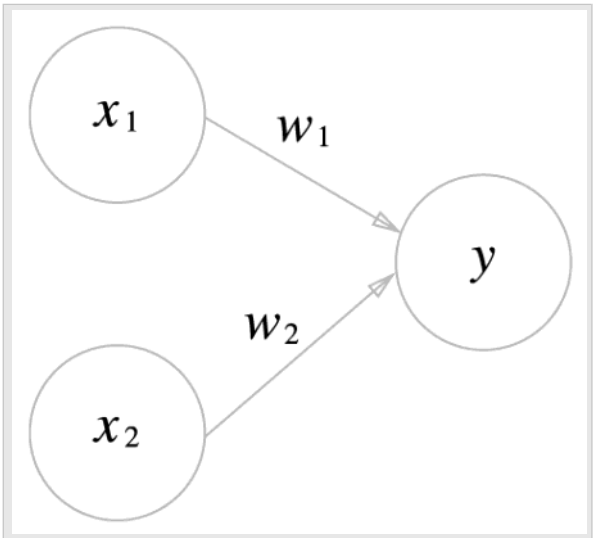
뉴런에서 보내온 신호의 총합이 정해진 한계(임계값)을 넘어설 때만 1을 출력한다.

퍼셉트론은 n개의 2진수가 하나의 뉴런을 통과해서 가중의 합이 0보다 크면 활성화되는 가장 간단한 신경망이다.

퍼셉트론을 학습시키는 방법은 간단한데, 보통 목표치를 정해주고 현재 계산한 값이 목표치와 다르면 그만큼의 오차를 다시 퍼셉트론에 반영해서 오차를 줄여나가는 방법이다.

문제 26) 아래의 식을 파이썬으로 구현하시오

그림)



보기)

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

답)

```
import numpy as np

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

print(sum(x*w))
```

```
[0 1]
[0.5 0.5]
```

0.5

문제 27) 위의 식에 책 52쪽에 나오는 편향을 더해 완성한 아래의 식을 파이썬으로 구현하시오

보기)

```
b + x1 * w1 + x2 * w2 = ?
```

답)

```
import numpy as np

x = np.array([0,1])
w = np.array([0.5,0.5])
b = -0.7

print(sum(x*w) + b)
```

-0.19999999999999996

AND 게이트

문제 28) AND 게이트를 파이썬으로 구현하시오

보기)

그림 2-2 AND 게이트의 진리표

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

답)

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

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

```
0
0
0
1
```

문제 29) 위에서 만든 AND 퍼셉트론 함수를 이용해서 아래의 inputdata를 이용해서 출력 결과를 for loop 문으로 한번에 출력하시오

답)

```
import numpy as np

inputData = np.array([[0,0], [0,1], [1,0], [1,1]])
print("------ AND 퍼셉트론 ------")

for i in inputData:
    print(str(i) , "==>" , AND(i[0],i[1]))

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

```
------ AND 퍼셉트론 ------
[0 0] ==> 0
[0 1] ==> 0
[1 0] ==> 0
[1 1] ==> 1
```

퍼셉트론 게이트 3가지

```
1. AND
2. OR
3. NAND
```

4. XOR : exclusive OR 라는 뜻으로 둘 중에 하나만 1일 때 1이 된다.

OR 게이트

문제 30) 문제 29번 코드를 가지고 좀 수정해서 OR 게이트 함수를 생성해서 아래와 같이 출력하시오

답)

```
def OR(x1, x2):
    w1, w2, theta = 0.5, 0.5, 0.4
    tmp = x1*w1 + x2*w2
    if tmp <= theta:
        return 0
    elif tmp > theta:
        return 1

import numpy as np

inputData = np.array([[0,0], [0,1], [1,0], [1,1]])
print("------ OR 퍼셉트론 ------")

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

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

NAND 게이트

문제 31) 문제 29번 코드를 가지고 좀 수정해서 NAND 게이트 함수를 생성해서 아래와 같이 출력하시오

답)

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

import numpy as np
```

```
inputData = np.array([[0,0], [0,1], [1,0], [1,1]])
print("------ NAND 퍼셉트론 ------")

for i in inputData:
    print(str(i) , "==>" , NAND(i[0],i[1]))
```

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

XOR 게이트

문제 32) 문제 29번 코드를 가지고 좀 수정해서 NAND 게이트 함수를 생성해서 아래와 같이 출력하시오

답)

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

import numpy as np

inputData = np.array([[0,0], [0,1], [1,0], [1,1]])
print("------ NAND 퍼셉트론 ------")

for i in inputData:
    print(str(i) , "==>" , NAND(i[0],i[1]))
```

단층 신경망과 다층 신경망의 차이

1958년 로젠블래트가 퍼셉트론을 제안을 했다.

1959년 민스키가 기존 퍼셉트론의 문제점을 지적했는데 XOR 분류를 못한다는 문제점을 지적하고
인공지능의 겨울기가 시작되었다.

즉, XOR 게이트는 단층신경망으로는 구현이 안되는 것이었다.
XOR 게이트는 다층 신경망으로 구현해야 하는 것이다.

1. 단층: 입력층 ---> 출력층
2. 다층: 얕은 신경망 : 입력층 --> 은닉층 --> 출력층
　　깊은 신경망(deep learning) : 입력층 --> 은닉층들 --> 출력층

1. XOR 게이트

XOR	x1	x2	t
	0	0	0
	1	0	1
	0	1	1
	1	1	0

2. 중간층을 넣어서

XOR	x1	x2	OR게이트 결과 (A)	NAND 결과 (B)	A와 B의 AND 결과
	0	0	0	1	0
	1	0	1	1	1
	0	1	1	1	1
	1	1	1	0	0

문제 33) XOR 함수를 생성하여 아래와 같이 결과가 나오도록 하시오

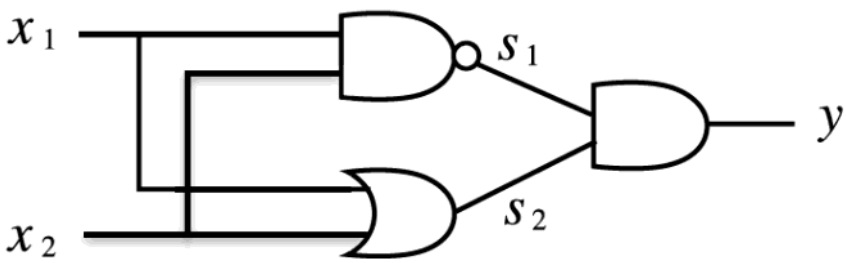
결과)

```
------ XOR 퍼셉트론 ------
[0 0] ==> 0
[0 1] ==> 1
[1 0] ==> 1
[1 1] ==> 0
```

답)

```
def OR(x1, x2):
    w1, w2, theta = 0.5, 0.5, 0.4
    tmp = x1*w1 + x2*w2
    if tmp <= theta:
        return 0
    elif tmp > theta:
        return 1

def NAND(x1, x2):
```

```
    w1, w2, theta = -0.5, -0.5, -0.7
    tmp = x1*w1 + x2*w2
    if tmp <= theta:
        return 0
    elif tmp > theta:
        return 1

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

def XOR(x1, x2):
    A = OR(x1, x2)
    B = NAND(x1, x2)
    return AND(A, B)


import numpy as np

inputData = np.array([[0,0], [0,1], [1,0], [1,1]])
print("------ XOR 퍼셉트론 ------")

for i in inputData:
    print(str(i) , "==>" , XOR(i[0],i[1]))
```



---

05 신경망

2018년 8월 13일 월요일 오후 4:34

Table of Contents

퍼셉트론과 신경망의 차이점

1. 퍼셉트론은 원하는 결과를 출력하도록 사람이 직접 가중치를 정해줘야 한다

2. 신경망은 가중치 매개변수의 적절한 값을 기계가 데이터로부터 학습해서 자동으로 알아낸다

신경망에 들어가는 함수

1. 은닉층에 들어가는 함수 : # 신호를 확률로 내보내는 역할
    - 계단함수
    - 시그모이드 함수
    - 렐루 함수

2. 출력층에 들어가는 함수 : # 은닉층에서 보내온 확률을 모아 개인지 고양인지 결정
    - 항등함수 (회귀분석)
    - 소프트 맥스 함수 (분류 문제)

3. 배치 처리하는 방법

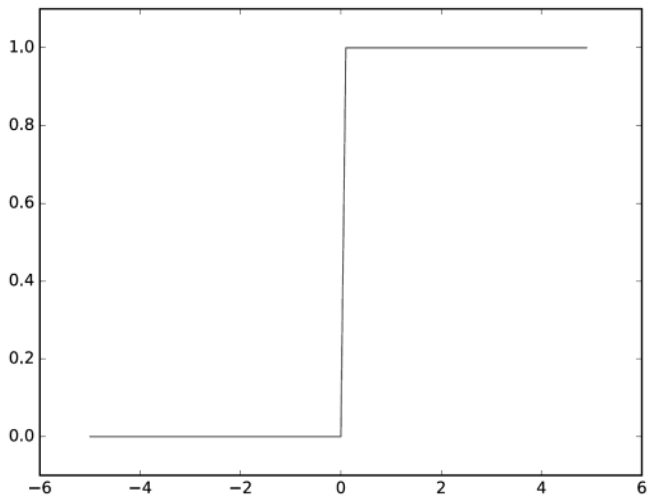
신경망 안에 은닉층에 들어가는 활성화함수 3가지

1. 계단 함수: 출력 신호가 0 또는 1의 값을 출력하는 함수
2. 시그모이드 함수: 0 ~ 1 사이의 실수를 출력하는 함수
3. 렐루 함수: 입력이 0을 넘으면 그 입력값을 그대로 출력하고 0 이하면 0을 출력하는 함수

계단 함수

문제 32) 파이썬으로 계단 함수를 만드시오

보기)



답)

```
def step_func(a):
    y = a > 0
    return y.astype(np.int)

x_data = np.array([-1,0,1])
print(step_func(x_data))
```

[0 0 1]

설명)

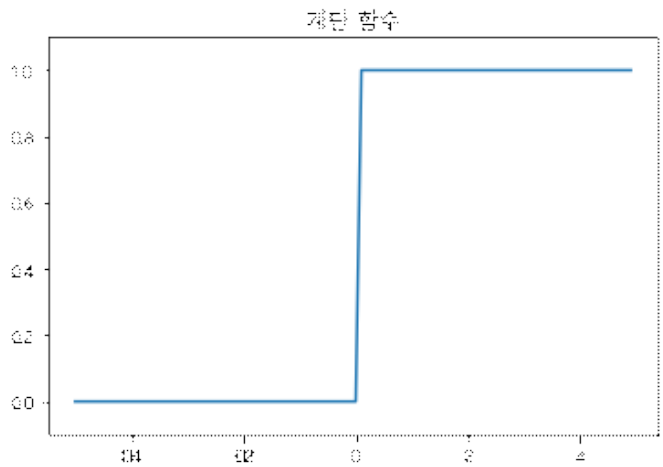
계단함수의 경우 주로 단층퍼셉트론에서 쓰인다

문제 33) 파이썬으로 계단 함수를 시각화하시오

답)

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

```
from matplotlib import font_manager, rc

# 계단함수 생성
def step_func(a):
    y = a > 0
    return y.astype(np.int)

x_data = np.arange(-5,5,0.1)
result = step_func(x_data)

# 한글 폰트 설정
font_name = font_manager.FontProperties(fname="C:/Windows/Fonts/MALGUN.TTF").get_name()
rc('font', family=font_name)

# 그래프 그리기
plt.plot(x_data, result)
plt.title("계단 함수")
plt.ylim(-0.1, 1.1)
plt.show()
```

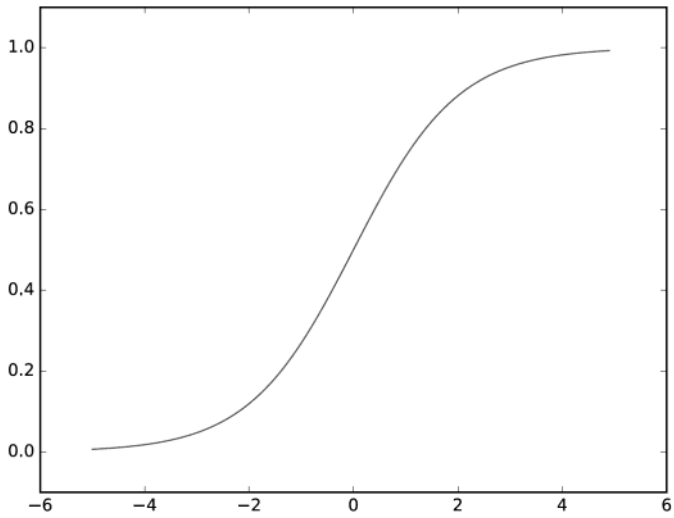


시그모이드 함수

계단 함수 (단층 퍼셉트론) vs 시그모이드 함수 (다층 퍼셉트론)

차이점
: 계단 함수는 숫자 1과 0만 출력하는데, 시그모이드 함수는 0~1의 실수 출력
    예) f(가중의 합) = 0.8834

공통점
: 둘다 0~1의 데이터만 출력한다는 점



시그모이드 함수식

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

문제 35) 시그모이드 함수를 파이썬으로 구현하시오

답)

```
import numpy as np

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

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

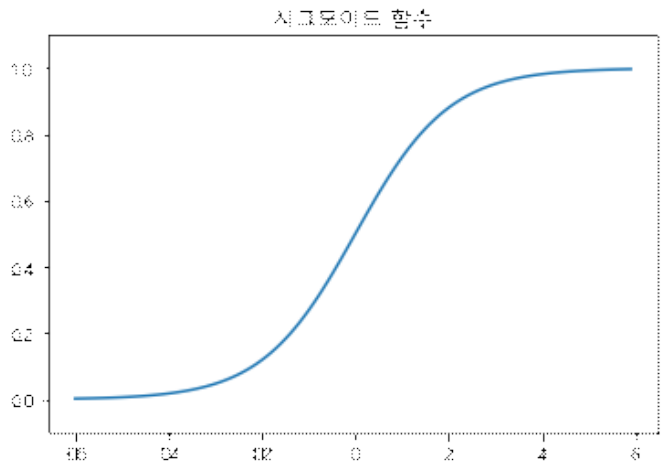
[0.73105858 0.88079708]

문제 36) 시그모이드 함수를 파이썬으로 그래프를 그려 시각화하시오

답)

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

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

```
x_data = np.arange(-6, 6, 0.1)
result = sigmoid(x_data)



# 한글 폰트 설정
font_name = font_manager.FontProperties(fname="C:/Windows/Fonts/MALGUN.TTF").get_name()
rc('font', family=font_name)



# 그래프 그리기
plt.plot(x_data, result)
plt.title("시그모이드 함수")
plt.ylim(-0.1, 1.1)
plt.show()
```



sigmoid 함수의 식이 왜 아래와 같은지 설명하시오

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

통계학에서 성공할 확률이 실패할 확률보다 얼마나 큰지를 나타내는 오즈비율이라는 값이 있다.

오즈 비율 = (성공 확률) / (실패 확률)

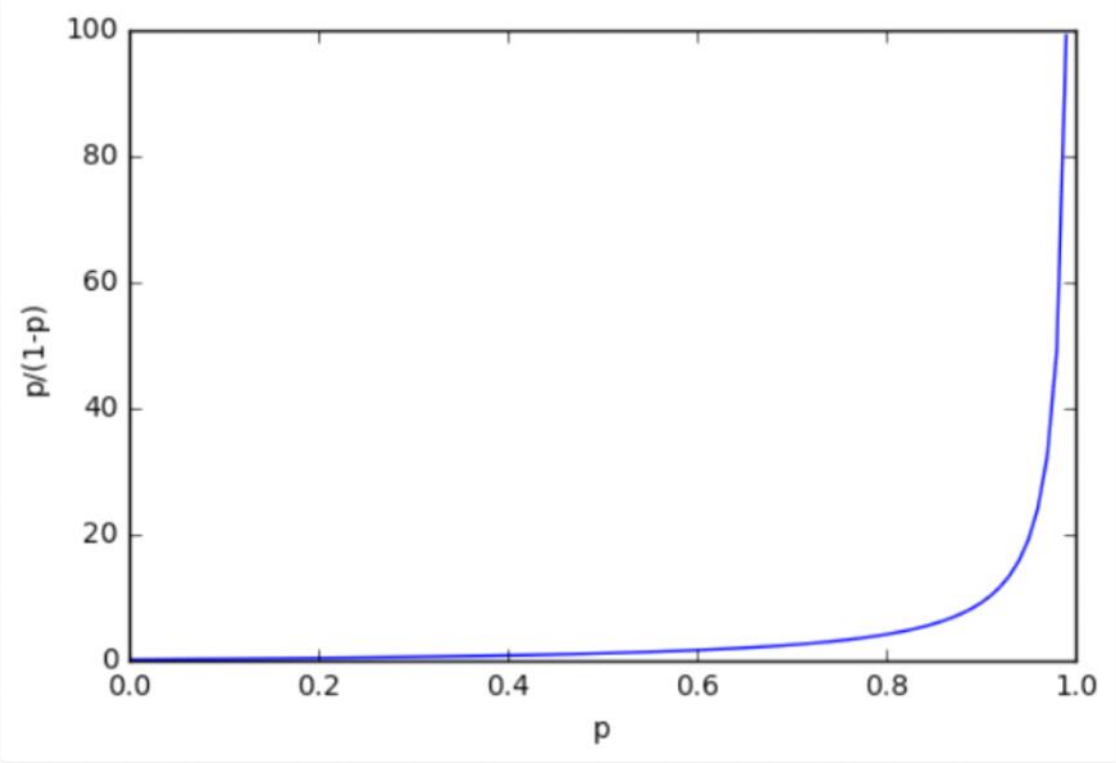


그림 7. $(\frac{p}{1-p})$ 그래프

성공할 확률 P를 0~1 사이의 값으로 나타내면 실패할 확률은 1-P이다

위의 그래프에서 P(성공할 확률) 가 1에 가까우면 오즈비율 (성공할 확률이 실패할 확률보다 얼마나 큰지) 값이 급격히 커져버리는 현상이 발생한다.

급격히 커져버리는 현상을 방지하기 위해서 이 함수에 로그를 취한게 logit 함수이다.

그래프는 아래와 같다

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

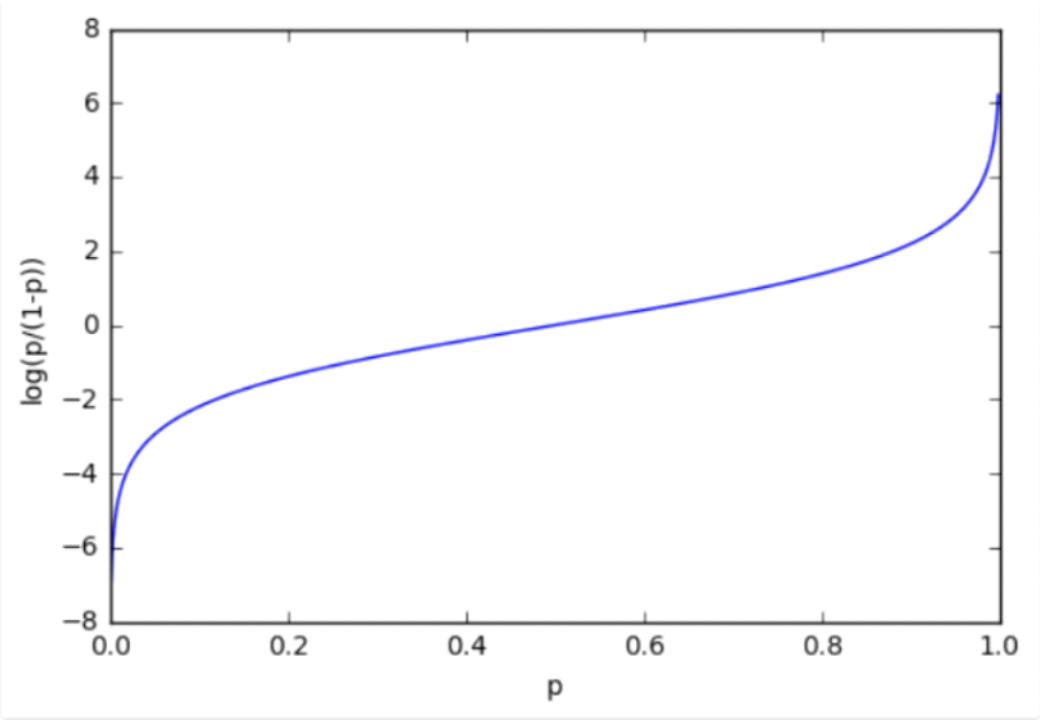


그림 8. $log(\frac{p}{1-p})$ 그래프

P가 0.5 일 때 실패할 확률 대비 성공할 확률이 0이 된다

P = 0.5 일 때는, 0이 되고
P = 1 일 때는, 무한히 큰 양수,
P = 0 일 때는, 무한히 큰 음수를 가지는 특징이 있다.

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

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

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

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

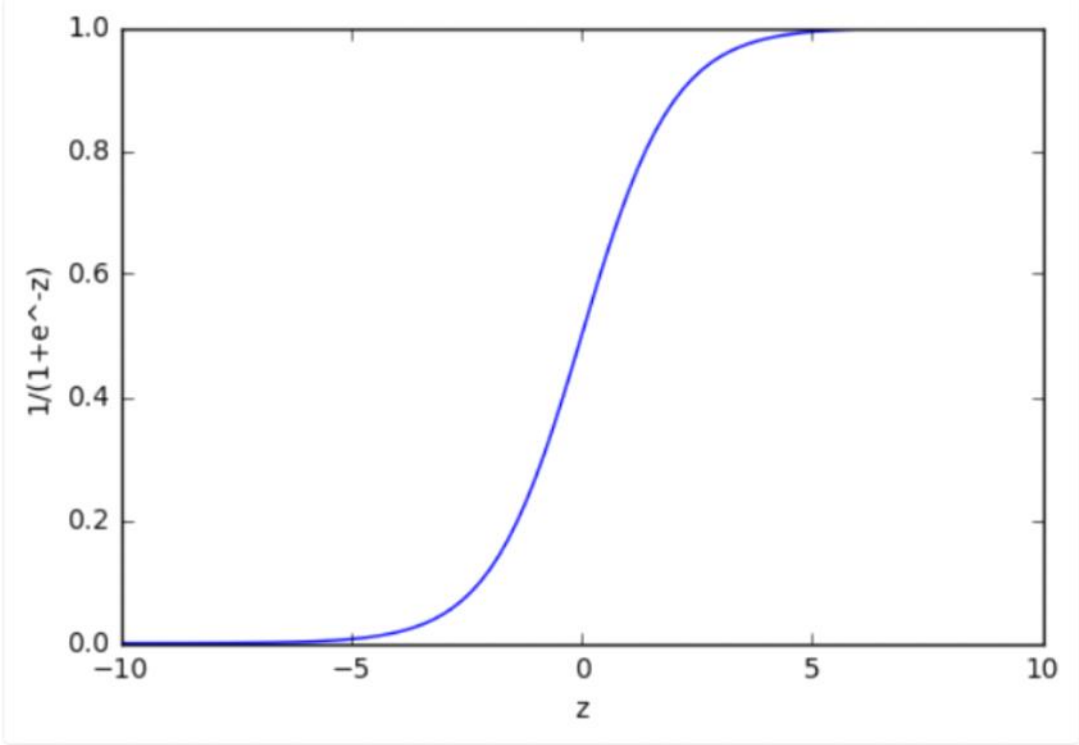
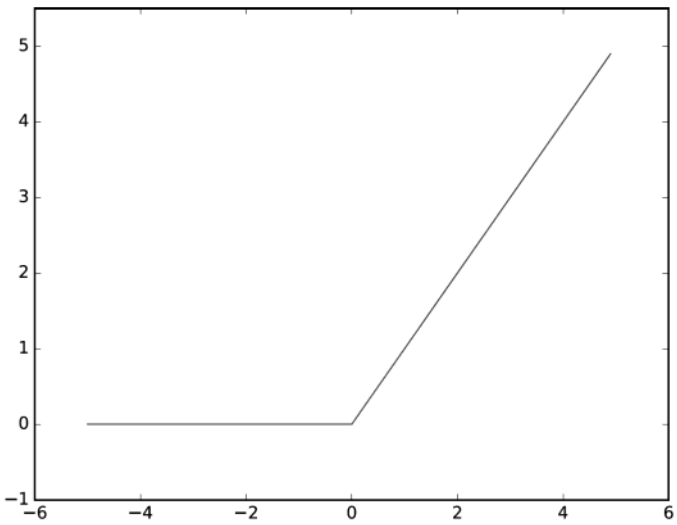


그림 9. $\frac{1}{1+e^{-z}}$ 그래프

렐루 함수 (Rectified Linear Unit)



Relu 는 입력이 0을 넘으면 그 입력을 그대로 출력하고 0 이하면 0을 출력하는 함수

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

문제 37) Relu 함수를 파이썬으로 구현하시오

답)

```
# 렐루 함수 생성
def Relu(x):
    if x > 0 :
        return x
    else :
        return 0

print(Relu(0.3))
print(Relu(-2))
```

0.3
0

문제 38) numpy 를 이용해서 Relu 함수를 파이썬으로 구현하시오

답)

```
import numpy as np

# 렐루 함수 생성
def Relu(x):
    return np.maximum(0, x)

print(Relu(0.3))
print(Relu(-2))
```

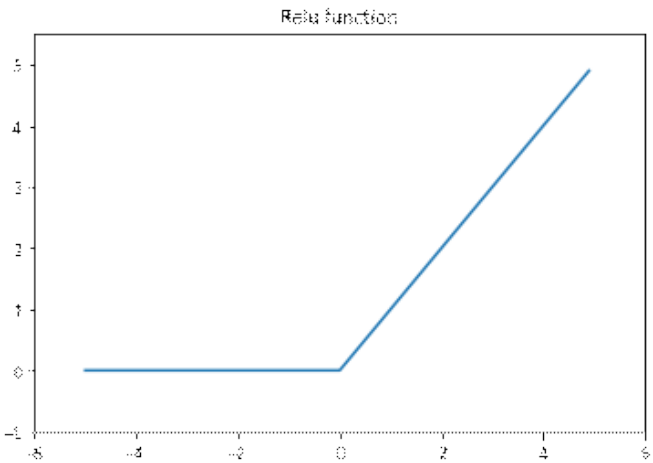
0.3
0

문제 38) Relu 함수를 파이썬으로 시각화하시오

답)

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

# 렐루 함수 생성
def Relu(x):
    return np.maximum(0, x)
```

```
x_data = np.arange(-5, 5, 0.1)
result = Relu(x_data)



# 한글 폰트 설정
font_name = font_manager.FontProperties(fname="C:/Windows/Fonts/calibri.ttf").get_name()
rc('font', family=font_name)



# 그래프 그리기
plt.plot(x_data, result)
plt.title("Relu function")
plt.xlim(-6,6)
plt.ylim(-1,5.5)
plt.show()
```



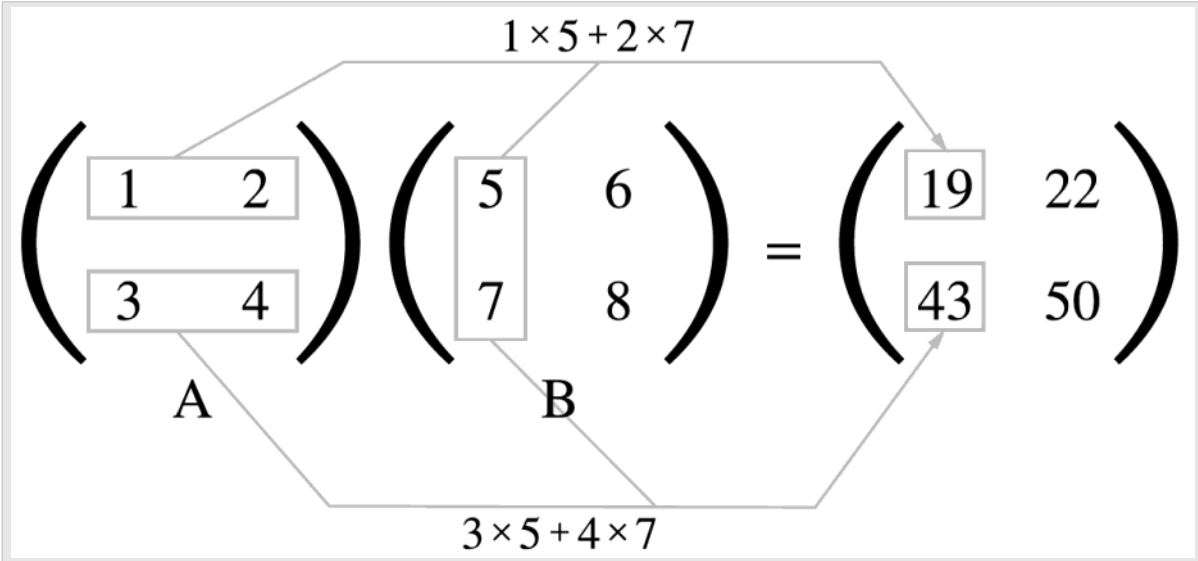
다차원 배열의 계산

넘파이로 행렬을 만들고 차원 수 확인과 몇 행 몇 열인지 확인하는 방법

```
import numpy as np

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

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

행렬의 내적



문제 40) 아래의 두 개의 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]]
```

np.matrix 로 구현하는 방법

```
import numpy as np

A = np.matrix([[1,2], [3,4]])
B = np.matrix([[5,6], [7,8]])

print(A*B)
```

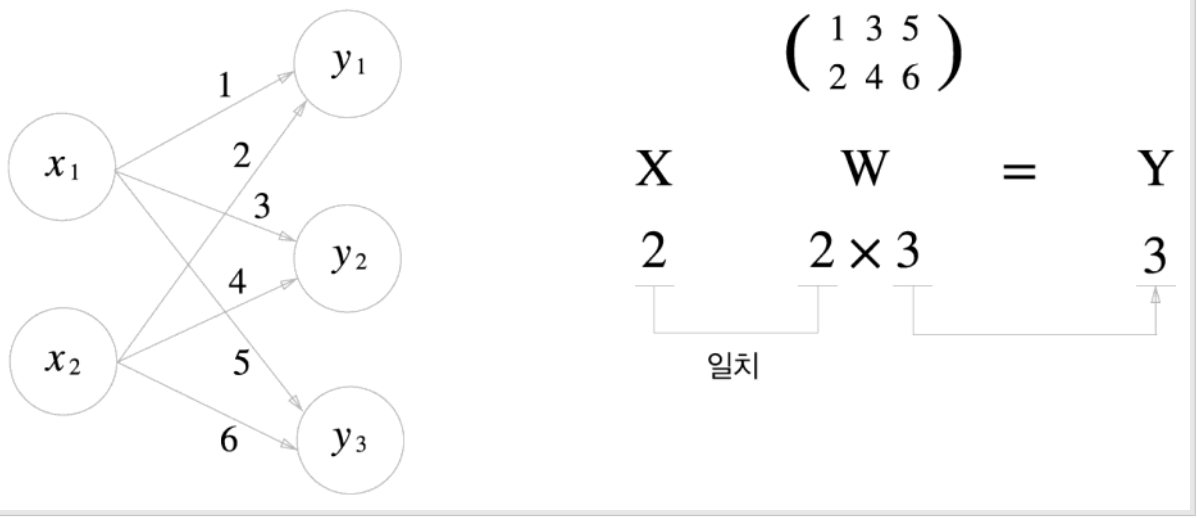
문제 41) 아래의 행렬 곱(내적)을 파이썬으로 구현하시오

답)

```
import numpy as np

A = np.matrix([[1,2,3], [4,5,6]])
B = np.matrix([[5,6], [7,8], [9,10]])

print(A*B)
```

```
[[ 46  52]
 [109 124]]
```

신경망의 내적



문제 42) 위의 신경망의 x1 과 x2 가 1과 2면 y1 과 y2 의 값은 무엇인가?

보기)

```
x1*1 + x2*2 = y1
x1*3 + x2*4 = y2
x1*5 + x2*6 = y3

(x1 x2) ( 1 3 5 )  = (y1 y2 y3)
        ( 2 4 6 )
```

답)

```
import numpy as np

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

```
print(np.dot(x, w))
```

[5 11 17]

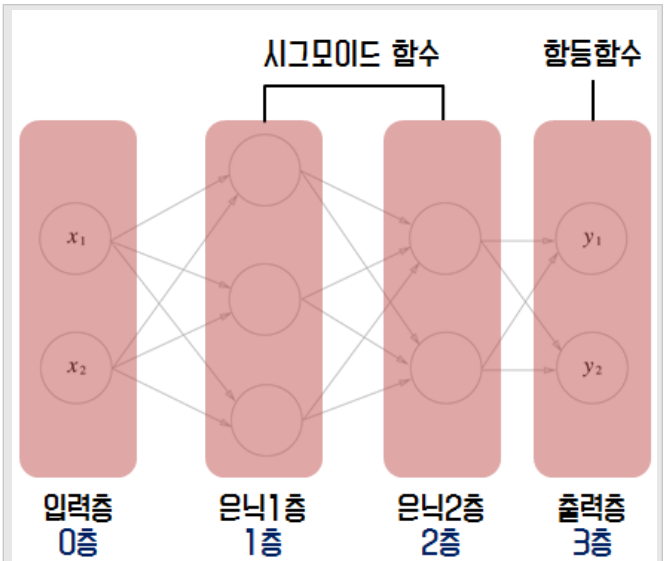
문제 43) 위의 신경망에서 만들어진 가중의 총합인 y값을 시그모이드 함수를 통과 시켜서 나온 y_hat 을 출력하시오

답)

```
import numpy as np

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

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

y = np.dot(x, w)
y_hat = sigmoid(y)
print(y_hat)
```

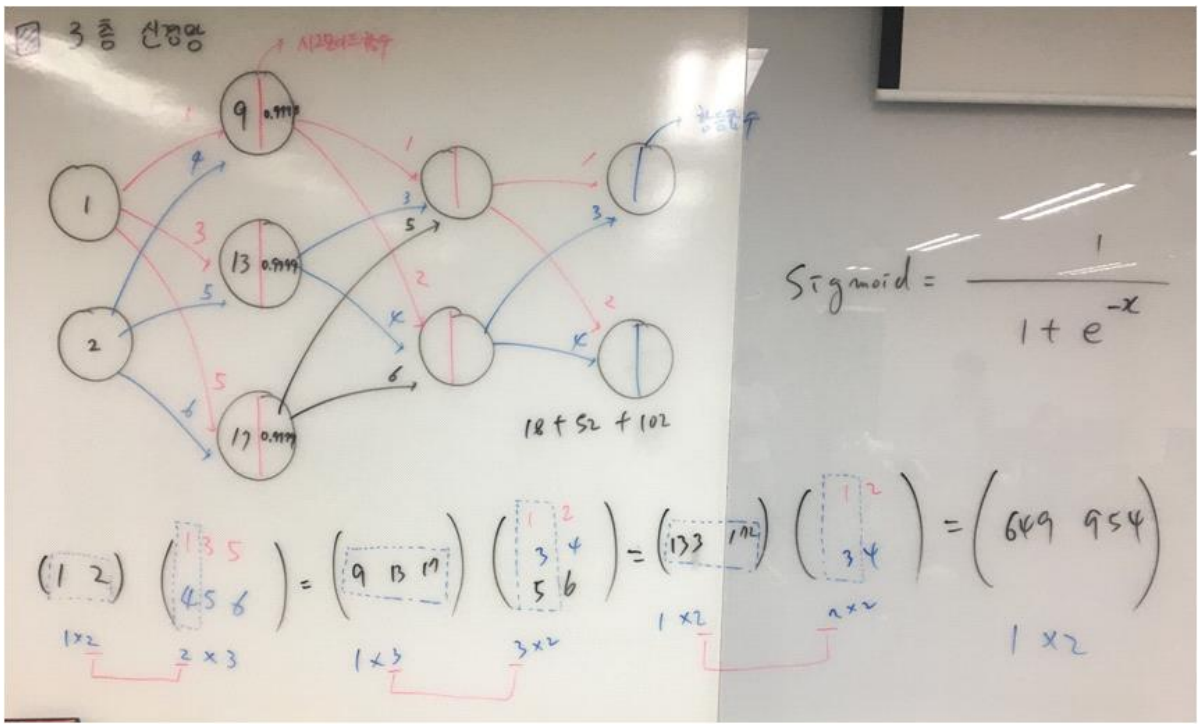
[0.99330715 0.9999833 0.99999996]

3층 신경망 구현하기



문제 44) 칠판에 나온 3층 신경망을 구현하고 J1, J2 값을 구하시오 (3층 신경망 코드)

칠판)



답)

```
#####################################################
# 3층 신경망 전체 코드
#####################################################
import numpy as np

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

# 항등 함수
def identity_func(x):
    return x


# 입력층
x = np.array([1,2])

# 입력층의 입력치와 가중치를 곱해준다
w = np.array([[1,3,5], [4,5,6]])
y = np.dot(x, w)


# 은닉 1층
y_hat = sigmoid(y)
```

```
# 은닉 1층의 결과와 가중치 곱
w2 = np.array([[1,2], [3,4], [5,6]])
y2 = np.dot(y_hat, w2)


# 은닉 2층
y2_hat = sigmoid(y2)


# 출력층
w3 = np.array([[1,2], [3,4]])
y3 = np.dot(y2_hat, w3)


# 출력층(항등함수 통과 후 결과 도출)
print(identity_func(y3))
```

[3.99985709 5.99972663]

시그모이드 함수의 유래

```
오즈비율 그래프 --> 로짓 함수 --> 시그모이드 함수
```

오즈비율 그래프

```
실패할 확률 대비 성공할 확률
```

로짓 함수

```
오즈비율 함수에 로그를 사용한 함수
```

시그모이드 함수

```
로짓함수를 신경망에서 P(확률)값을 계산하기 편하도록 지수형태로 바꾼 함수
```

문제 45) 아래의 입력값과 가중치를 이용해서 신경망 1층에서 나온 출력값인 [9 13 17] 을 출력하시오

답)

```
import numpy as np

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

```
y = np.dot(x, w)
print(y)
```

[9 13 17]

문제 46) 위의 결과에서 시그모이드 함수를 만들어 통과하도록 만드시오

답)

```
import numpy as np

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

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

y = np.dot(x, w)
y_hat = sigmoid(y)
print(y_hat)
```

[0.99987661 0.99999774 0.99999996]

문제 49) 출력층(3층)의 항등함수에 값이 입력되기 전의 y3 값을 출력하시오

답)

```
# 출력층
w3 = np.array([[1,2], [3,4]])
y3 = np.dot(y2_hat, w3)



# 출력층(항등함수 통과 후 결과 도출)
print(y3)
```

출력층 함수

```
출력층의 함수는 그동안 흘러왔던 확률들의 숫자를 취합해서 결론을 내줘야 하는 함수

신경망으로 구현하고자 하는 문제가
1. 회귀면? 항등함수
예: 독립변수(콘크리트 재료: 자갈 200kg, 시멘트 2포대), 종속변수(콘크리트 강도)
```

2. 분류면? 소프트맥스 함수
예: 정상 폐사진 vs 폐결절 사진

문제 50) 입력값을 받아 그대로 출력하는 항등함수를 identity_function 이라는 이름으로 생성하시오

답)

```
# 항등 함수
def identity_func(x):
    return x
```

문제 51) 항등함수를 통과시킨 최종 결과를 출력하시오

답)

```
# 출력층
w3 = np.array([[1,2], [3,4]])
y3 = np.dot(y2_hat, w3)



# 출력층(항등함수 통과 후 결과 도출)
print(y3)
```

문제 52) 위의 코드 중에 가중치에 해당하는 코드들을 init_network() 라는 함수로 생성해서 가중치를 딕셔너리에서 가져올 수 있도록 하시오

답)

```
# 가중치 받아오는 함수
def init_network():
    network = {}
    network['W1'] = np.array([[1,3,5], [4,5,6]])
    network['W2'] = np.array([[1,2], [3,4], [5,6]])
    network['W3'] = np.array([[1,2], [3,4]])

    return network

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

문제 53) 1층, 2층, 3층 코드를 하나의 함수로 생성하시오
(함수 이름: forward(network, x))

답)

```
##################################################
# 3층 신경망 전체 코드
##################################################
import numpy as np

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

# 항등 함수
def identity_func(x):
    return x



# 가중치 받아오는 함수
def init_network():
    network = {}
    network['W1'] = np.array([[1,3,5], [4,5,6]])
    network['W2'] = np.array([[1,2], [3,4], [5,6]])
    network['W3'] = np.array([[1,2], [3,4]])

    return network



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

    # 입력층의 입력치와 가중치를 곱해준다
    y = np.dot(x, W1)

    # 은닉 1층
    y_hat = sigmoid(y)

    # 은닉 1층의 결과와 가중치 곱
    y2 = np.dot(y_hat, W2)

    # 은닉 2층
    y2_hat = sigmoid(y2)
```

```
    # 출력층
    y3 = np.dot(y2_hat, W3)

    # 출력층(항등함수 통과 후 결과 도출)
    y3_hat = identity_func(y3)

    return y3_hat


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

[3.99985815 5.9997286]

문제 54) 입력 값이 1, 2 가 아니라 4, 5 일 때의 위의 3층 신경망을 통과한 결과를 출력하시오

답)

```
################################################
# 3층 신경망 전체 코드
################################################
import numpy as np

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

# 항등 함수
def identity_func(x):
    return x


# 가중치 받아오는 함수
def init_network():
    network = {}
    network['W1'] = np.array([[1,3,5], [4,5,6]])
    network['W2'] = np.array([[1,2], [3,4], [5,6]])
    network['W3'] = np.array([[1,2], [3,4]])

    return network
```

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

    # 입력층의 입력치와 가중치를 곱해준다
    y = np.dot(x, W1)

    # 은닉 1층
    y_hat = sigmoid(y)

    # 은닉 1층의 결과와 가중치 곱
    y2 = np.dot(y_hat, W2)

    # 은닉 2층
    y2_hat = sigmoid(y2)

    # 출력층
    y3 = np.dot(y2_hat, W3)

    # 출력층(항등함수 통과 후 결과 도출)
    y3_hat = identity_func(y3)

    return y3_hat


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

소프트맥스(softmax) 함수

분류를 위한 출력층 함수인데 0~1 사이의 숫자를 출력하는 함수

공식)

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

소프트맥스 함수는 지수 함수를 사용하는데 이 지수함수가 쉽게 아주 큰 값을 내뱉는다
e (스위스의 수학자 오일러가 발견한 무리) 의 10승은 20,000이 넘고
e의 100승은 40개가 넘고 e의 1000승은 무한대를 뜻하는 inf가 출력이 되어 돌아오므로 컴퓨터로

계산을 할 수가 없다.

예)

```
import numpy as np
print(np.exp(100))
```

2.6881171418161356e+43

로그함수는 수를 작게 만들고 지수함수는 수를 크게 만든다

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

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

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

소프트맥스 함수 공식에서 상수 C 를 분모, 분자에 곱해주었다

문제 55) 아래의 리스트를 무리수 (자연상수)의 제곱으로 해서 계산한 값이 무엇인가?

보기)

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

답)

```
import numpy as np

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

[inf inf inf]

문제 56) 아래의 리스트에서 가장 큰 값을 출력하시오

보기)

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

답)

```
import numpy as np

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

문제 57) 아래의 리스트에서 각 요소를 아래의 리스트에서 가장 큰 값으로 뺀 값이 얼마인지 출력하시오

보기)

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

답)

```
import numpy as np

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

[0 -10 -20]

설명)

```
가장 큰 수로 빼면 0보다 같거나 음수의 값으로 나온다!
```

문제 58) 위의 [0 -10 -20] 을 자연상수 e의 제곱으로 해서 출력된 결과가 무엇인지 확인하시오

보기)

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

답)

```
import numpy as np

a = np.array([1010, 1000, 990])
C = np.max(a)
res = a - C
```

```
print(np.exp(np.exp(res)))
```

[2.71828183 1.0000454 1.   ]

문제 59) 위의 코드를 이용해서 softmax 함수를 생성하기 위한 분자식을 구현하시오

보기)

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

답)

```
import numpy as np

def softmax(a):
    c = np.max(a)
    minus = a - c
    np_exp = np.exp(minus)
    return np_exp

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

[1.00000000e+00 4.53999298e-05 2.06115362e-09]

문제 60) softmax 함수를 분모까지 포함해서 완전히 완성하시오

보기)

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

답)

```
import numpy as np

def softmax(a):
    c = np.max(a)
    minus = a - c
    exp_a = np.exp(minus)  # 분자
    sum_exp_a = np.sum(exp_a)  # 분모
    return exp_a / sum_exp_a

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

[9.99954600e-01 4.53978686e-05 2.06106005e-09]

문제 61) 문제 53번 코드에 항등 함수를 소프트맥스 함수로 바꾸시오

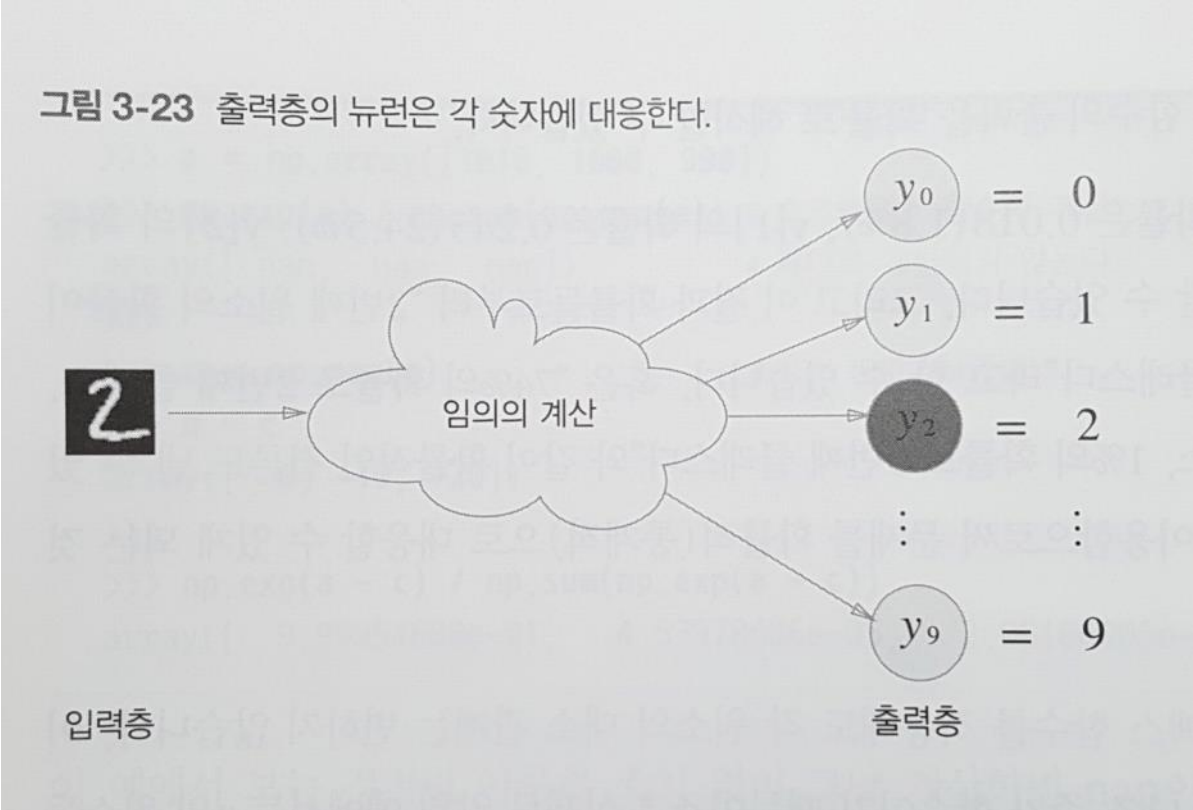
답)

```
################################################
# 3층 신경망 전체 코드
################################################
import numpy as np

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

# 항등 함수
def identity_func(x):
    return x

# 소프트맥스 함수
def softmax(a):
    c = np.max(a)
    minus = a - c
    exp_a = np.exp(minus)       # 분자
    sum_exp_a = np.sum(exp_a)     # 분모
    return exp_a / sum_exp_a


# 가중치 받아오는 함수
def init_network():
    network = {}
    network['W1'] = np.array([[1,3,5], [4,5,6]])
    network['W2'] = np.array([[1,2], [3,4], [5,6]])
    network['W3'] = np.array([[1,2], [3,4]])

    return network


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

    # 입력층의 입력치와 가중치를 곱해준다
```

```
    y = np.dot(x, W1)

    # 은닉 1층
    y_hat = sigmoid(y)

    # 은닉 1층의 결과와 가중치 곱
    y2 = np.dot(y_hat, W2)

    # 은닉 2층
    y2_hat = sigmoid(y2)

    # 출력층
    y3 = np.dot(y2_hat, W3)

    # 출력층(항등함수 통과 후 결과 도출)
    y3_hat = softmax(y3)

    return y3_hat


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

[0.11921653 0.88078347]

교재 96페이지



신경망
학습 >> 소프트맥스 함수를 사용
테스트 >> 소프트 맥스 함수를 사용하지 X

왜냐하면, 어차피 결과가 같기 때문에 지수 함수 계산에 드는 자원 낭비를 줄이기 위해

출력층의 뉴런 수 정하기

데이터 종류	출력층의 뉴런 수
사진 데이터로 개인지 고양이인지만 분류	2개
정상 폐사진, 질병 폐사진 분류	2개
MNIST 데이터 (필기체 숫자 0~9 사진)	

MNIST (손글씨 필기체) 데이터를 파이썬으로 로드하는 방법

1. 책 소스코드와 데이터를 다운로드 받는다.
https://github.com/WegraLee/deep-learning-from-scratch

2. dataset 이라는 폴더를 워킹 디렉토리에 가져다 둔다
(실행하는 소스가 있는 디렉토리)
C:₩Users₩kwang₩iCloudDrive₩Python

3. 아래의 파이썬 코드를 실행해서 필기체 데이터 하나를 시각화 한다

코드

```
# coding: utf-8

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



def img_show(img):       # 이미지 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)
img = x_train[0]
label = t_train[0]
print(label)  # 5
print(img.shape)  # (784,)

img = img.reshape(28, 28)
img_show(img)
```

문제 62) 아래의 코드의 dataset 패키지의 mnist.py 에 load_mnist 라는 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))
```

설명)

```
파이참으로는 간단하게 확인할 수 있지만 쥬피터는 방법을 모름.
```

문제 63) x_train 데이터를 print 하여 확인하시오

답)

```
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 (x_train)
```

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

문제 64) x_train 데이터가 6만장이 맞는지 확인하시오

답)

```
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(x_train))
```

60000

문제 65) x_train 6만 개의 이미지 중에서 제일 첫번째 이미지 한개를 출력하시오

답)

```
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=False, normalize=False)

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

문제 66) 아래의 리스트를 numpy array 로 바꾸시오

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]]
```

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
print(np.array(x))
```

문제 67) 위의 결과의 차원을 확인하시오

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)
```

문제 68) 위의 결과를 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

문제 69) MNIST의 훈련 데이터 (x_train)의 차원이 어떻게 되는지 확인하시오

답)

```
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 (x_train[0])
print(t_train.ndim)
```

1

문제 70) flatten을 False 로 설정하고

답)

```
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=False, normalize=False)

print (x_train.shape)
```

(60000, 1, 28, 28)

설명)

```
60000: 전체 장수
1: 흑백
28: 가로
28: 세로
```

컬러면: RGB 값으로 해서 3이 출력됨

문제 71) t_train[0] 가 어떤 숫자인지 확인하는데 one_hot encoding을 했을 때와 안했을 때의 차이를 확인하시오

답)

```
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, one_hot_label=True)

print (t_train[0])
```

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

설명)

one_hot_label=False 로 주면 5 라고 나온다

문제 72) x_train[0] 요소를 시각화 하시오 (img_show 함수를 이용하여)

답)

```
# coding: utf-8

import sys, os

sys.path.append(os.pardir)

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

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

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

```
img = x_train[0]
img = img.reshape(28, 28)
img_show(img)
```

문제 73) mnist 데이터를 가져오는 코드를 가지고 아래의 get_data() 함수를 생성하고 아래와 같이 실행되게 하시오

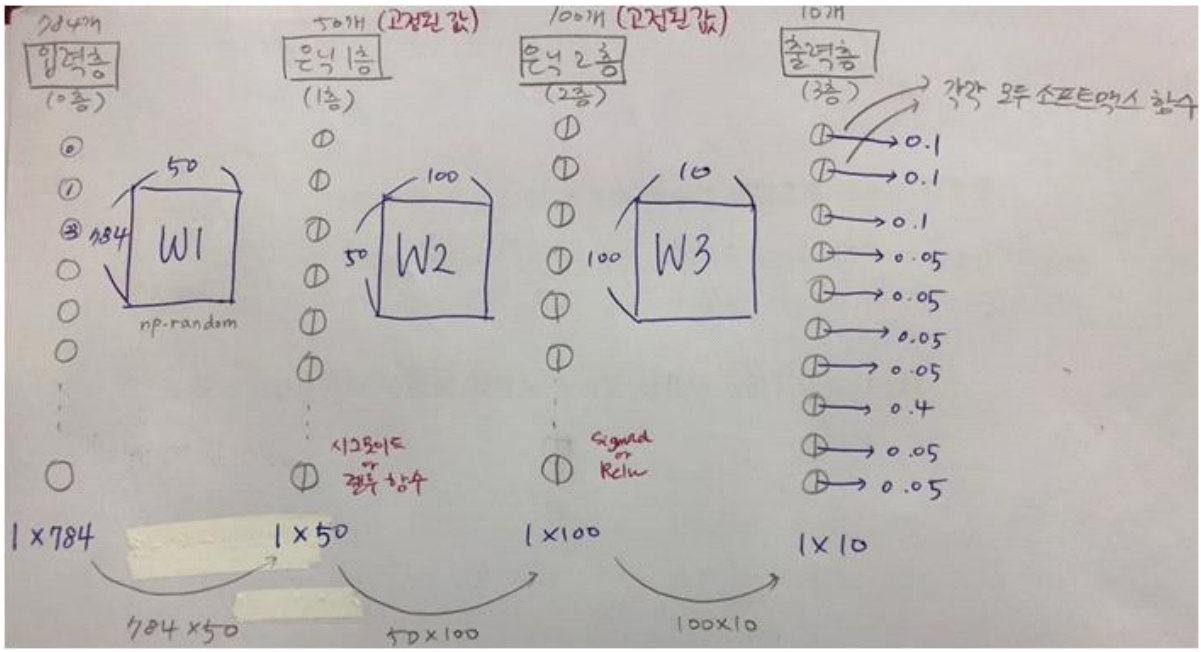
답)

```
# coding: utf-8

import sys, os

sys.path.append(os.pardir)

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

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

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



x, t = get_data()
print(x.shape)
print(len(t))
img = x_train[0]
img = img.reshape(28, 28)
img_show(img)
```

```
(10000, 784)
10000
```

문제 74) 테스트 데이터 10000장을 3층 신경망으로 입력했을때의 그림을 손으로 그리시오

답)



문제 75) mnist 필기체 데이터를 위해 신경망에서 사용할 가중치 (weight) 와 bias 가 셋팅되어 있는 sample_weight.pkl 을 로드하는 함수를 init_network() 라는 이름으로 생성하시오 (p.100)

답)

```
import pickle

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,)
```

문제 76) mnist 데이터 셋에서 숫자 하나를 입력하면 그 숫자가 어떤 숫자인지 예측하는 predict 라는 함수를 책 100페이지를 보고 만드시오

답)

```
import sys, os

sys.path.append(os.pardir)

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

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

# 소프트맥스 함수
def softmax(a):
    c = np.max(a)
    minus = a - c
    exp_a = np.exp(minus)       # 분자
    sum_exp_a = np.sum(exp_a)     # 분모
    return exp_a / sum_exp_a

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

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
    z3 = sigmoid(a3)
    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]

문제 77) A4에 필기체 숫자를 쓰고 사진을 찍어서 png 또는 jpg 로 만든 후에 numpy 배열로 변환하시오

답)
A4에 숫자를 쓰고 찍은 사진을 그림판으로 28x28 픽셀로 바꾸고 numpy 배열로 바꾸기

```
import cv2
j = 'C:/python_test_data/number3.jpg'
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg



img = mpimg.imread(j)
print(img.shape)
```

문제 78) 위의 코드에서 number3.jpg 파일을 (28, 28, 1) 로 변경하시오 (grayscale로 변환하는 작업)

답)

```
import cv2
j = 'C:/python_test_data/number3.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]   # graysacle 로 변환
print(img.shape)
```

(28, 28, 3)
(28, 28)

문제 79) 흑백으로 변경한 숫자 3을 3층 신경망에 넣고 숫자 3을 컴퓨터가 맞추는지 확인하시오

답)

```
import cv2
j = 'C:/python_test_data/number3.jpg'
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import sys, os

sys.path.append(os.pardir)

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



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



def identity_function(x):
    return x



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

```
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

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


import matplotlib.image as img
img=img.imread(j)
img=img[:,:,2] #grayscale로 변환

img1=img.flatten()
network=init_network()
y=predict(network,img1)
print(y)
print(np.where(y==max(y)))
```

문제 80) 아래의 10개의 원소를 갖는 x라는 리스트를 만들고 x리스트에 가장 큰 원소가 몇번째 인덱스인지 알아내시오

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

문제 81) 문제 79번의 코드에서 np.where 코드 대신에 np.argmax 로 수행해서 결과를 출력하시오

답)

```
img1=img.flatten()
network=init_network()
y=predict(network,img1)
print(y)
y2 = np.argmax(y, axis=0)
print(y2)
```

[5.1594954e-02 1.1992999e-04 3.7974499e-02 7.3731309e-01 1.8067822e-05
 7.8464113e-02 4.5326524e-05 8.5749791e-04 9.2930138e-02 6.8237475e-04]
3

문제 82) 아래의 리스트를 numpy array 리스트로 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],
     [0.0,0.05,0.9,0.0,0.05,0.0,0.0,0.0,0.0,0.0]]
```

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],
      [0.0,0.05,0.9,0.0,0.05,0.0,0.0,0.0,0.0,0.0]]

y = np.array(x)
print(y.shape)
```

(10, 10)

문제 83) 위의 (10, 10) 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],
      [0.0,0.05,0.9,0.0,0.05,0.0,0.0,0.0,0.0,0.0]]

y = np.array(x)
print(y.shape)
print(np.argmax(y, axis=1))
# axis=0 : 열에서 가장 큰 수, axis=1 : 행에서 가장 큰 수, axis 인수를 안쓰면 전체에서 가장 큰 수
```

문제 85) 테스트 데이터 하나 x[34] 의 필기체가 라벨이 무엇인지 확인하시오

답)

```
import cv2
j = 'C:/python_test_data/1.jpg'
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
```

```
import sys, os

sys.path.append(os.pardir)

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


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


def identity_function(x):
    return x


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

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

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



x, t = get_data()
print(x.shape)
print(t[34])
```

```
(10000, 784)
7
```

문제 86) 테스트 데이터의 하나인 x[34]의 필기체를 신경망에 넣고 신경망이 예측한 것과 라벨이 서로 일치하는지 확인하시오

답)

```
import cv2
j = 'C:/python_test_data/1.jpg'
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import sys, os

sys.path.append(os.pardir)

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



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

def identity_function(x):
```

```
    return x

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

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

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



x, t = get_data()
network=init_network()
y=predict(network,x[34])
print('예측결과:', np.argmax(y, axis=0))
print('실제결과:',t[34])
```

예측결과: 7
실제결과: 7

문제 87, 88) 테스트 데이터 10000장을 for loop 문으로 전부 3층 신경망에 넣고 10000장 중에 몇장을 3층 신경망이 맞추는지 확인하고 정확도도 확인하시오

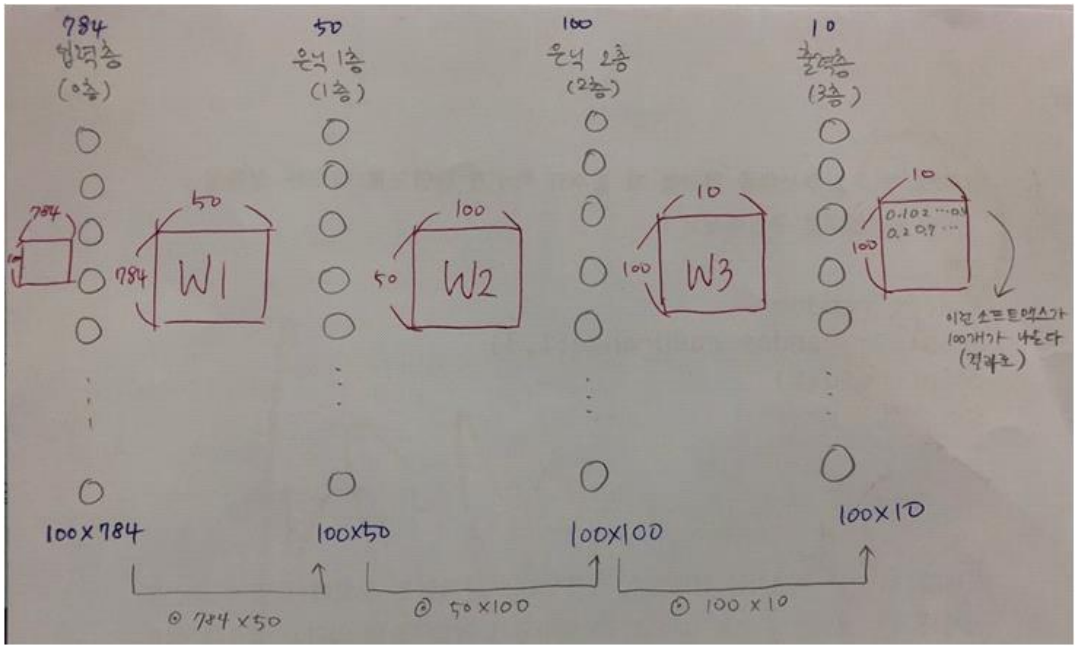
답)

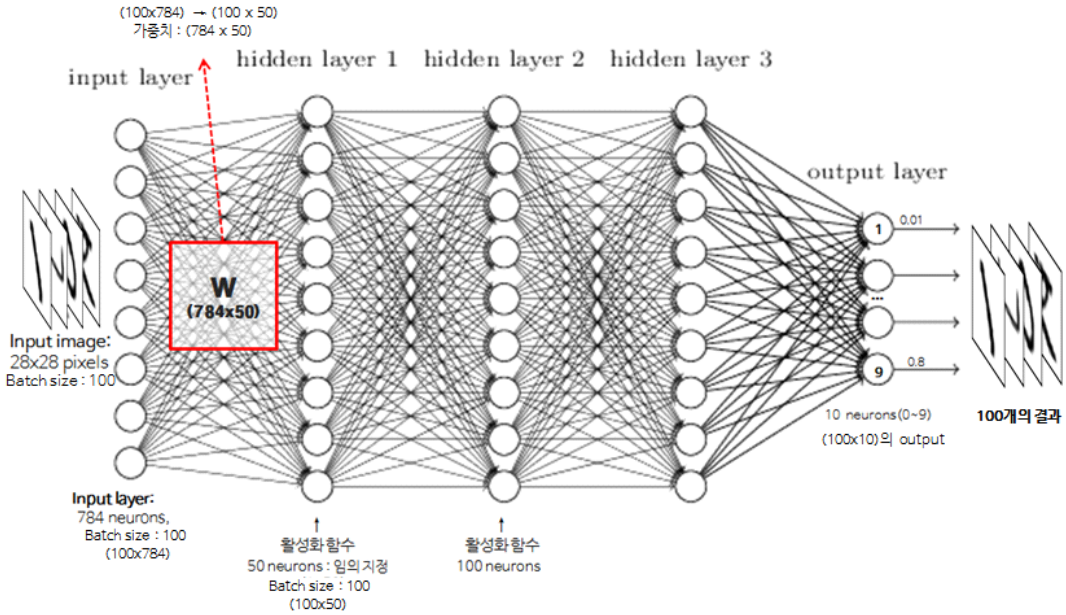
```
import cv2
j = 'C:/python_test_data/1.jpg'
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import sys, os

sys.path.append(os.pardir)

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



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

def identity_function(x):
    return x

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

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

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



x, t = get_data()
network=init_network()

cnt = 0
x_size = len(x)
for i in range(x_size):
    a = predict(network, x[i])
    if t[i] == np.argmax(a, axis=0):
        cnt += 1
print('전체 10000장 중', cnt, '개를 맞췄습니다.')
print('정확도:', round(cnt/x_size * 100, 2), '%')
```

전체 10000장 중 9352 개를 맞췄습니다.
정확도: 93.52 %

신경망에 데이터 입력 시 배치(batch)로 입력하는 방법





배치(batch)

이미지를 한장씩 처리하는게 아니라 여러장을 한번에 처리

이미지를 한장 씩 처리한 신경망

밑바닥부터 시작하는 딥러닝 교재 p.103 그림 3-26

이미지를 100장 씩 처리한 신경망

밑바닥부터 시작하는 딥러닝 교재 p.103 그림 3-27

문제 89) 아래의 결과를 출력하시오

보기)

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

답)

```
x = list(range(10))
print(x)
```

문제 90) 아래의 결과를 출력하시오

보기)

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

답)

```
x = list(range(0, 10, 3))
print(x)
```

문제 91) 아래의 리스트에서 최대값의 원소의 인덱스를 출력하시오

보기)

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

답)

```
import numpy as np

x = list(range(0, 10, 3))
print(x)

# 최대값의 인덱스 뽑기
y = np.argmax(x)
print(y)
```

문제 92) 아래의 행렬을 numpy 배열로 만드시오

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

x = 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(x)
```

답 2)

```
import numpy as np

x = 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(x)
y = x.reshape(4, 3)
print(y)
```

문제 93) numpy의 argmax를 이용해서 아래의 행렬에서 행 중에 최대값 원소의 인덱스를 출력하시오

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

x = 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(x)

print('열 별 최대값:', np.argmax(x, axis=0))
print('행 별 최대값:', np.argmax(x, axis=1))
```

```
[[0.1 0.8 0.1]
 [0.3 0.1 0.6]
```

```
 [0.2 0.5 0.3]
 [0.8 0.1 0.1]]
열 별 최대값: [3 0 1]
행 별 최대값: [1 2 1 0]
```

문제 94) 아래의 2개의 리스트를 만들고 서로 같은 자리에 같은 숫자가 몇 개가 있는지 출력하시오

보기)

```
x = [2,1,3,5,1,4,2,1,1,0]
y = [2,1,3,4,5,4,2,1,1,2]
```

답)

```
import numpy as np

x = [2,1,3,5,1,4,2,1,1,0]
y = [2,1,3,4,5,4,2,1,1,2]

cnt = 0
for i in range(len(x)):
    if x[i] == y[i]:
        cnt += 1
print(cnt)
```

7

Numpy 이용한 방법

```
import numpy as np

x = [2,1,3,5,1,4,2,1,1,0]
y = [2,1,3,4,5,4,2,1,1,2]
a = np.array(x)
b = np.array(y)

print(np.sum(a == b))
```

7

행렬 np.array도 가능

```
import numpy as np

x = 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]])

y = np.array([[0.2,0.8,0.1],
                [0.3,0.1,0.6],
                [0.2,0.4,0.3],
                [0.8,0.1,0.1]])

print(x == y)
```

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

문제 95) 아래의 리스트를 x라는 변수에 담고 앞에 5개의 숫자만 출력하시오

답)

```
x = [7,3,2,1,6,7,7,8,2,4]
print(x[:5])
```

문제 96) 아래와 같이 나오도록 하시오

보기)

```
0 ~ 100
100 ~ 200
200 ~ 300
.
.
.
9900 ~ 10000
```

답)

```
import numpy as np

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

문제 97) mnist 의 훈련 데이터를 100개 씩 가져오는 코드를 작성하시오

답)

```
x, t = get_data()

batch_size = 100
for i in range(0, 10000, 100):    # 0 ~ 100 100 ~ 200 , ... , 9800 ~ 9900, 9900 ~ 10000
    x_batch = x[i:i+batch_size]
    print(x_batch)
```

문제 98) 100개 가지고 온 훈련데이터를 predict 함수에 입력해서 예측 숫자 100개를 출력하는 코드를 작성하시오

답)

```
x, t = get_data()

batch_size = 100
for i in range(0, 10000, 100):    # 0 ~ 100 100 ~ 200 , ... , 9800 ~ 9900, 9900 ~ 10000
    x_batch = x[i:i+batch_size]
    y_batch = predict(network, x_batch) # 100개 짜리가 신경망으로 들어옴
    p = np.argmax(y_batch, axis = 1)
print(p)
```

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

문제 99) 위의 코드를를 수정해서 예측 숫자 100개와 실제 숫자 100개를 출력하도록 파이썬 코드를 작성하시오

답)

```
x, t = get_data()

batch_size = 100
for i in range(0, 10000, 100):    # 0 ~ 100 100 ~ 200 , ... , 9800 ~ 9900, 9900 ~ 10000
    x_batch = x[i:i+batch_size]
    t_batch = t[i:i+batch_size]
    y_batch = predict(network, x_batch) # 100개 짜리가 신경망으로 들어옴
    p = np.argmax(y_batch, axis = 1)   # 행 별로 최대값을 뽑는다.
    print(np.sum(np.array(p) == np.array(t_batch)))
```

문제 100) 아래의 100개 씩 뽑은 예상 숫자와 실제 숫자가 서로 얼마나 일치하는지 숫자를 count 해서 아래와 같은 결과를 출력하시오

답)

```
x, t = get_data()

cnt = 0
batch_size = 100
for i in range(0, 10000, 100):    # 0 ~ 100 100 ~ 200 , ... , 9800 ~ 9900, 9900 ~ 10000
    x_batch = x[i:i+batch_size]
    t_batch = t[i:i+batch_size]
    y_batch = predict(network, x_batch) # 100개 짜리가 신경망으로 들어옴
    p = np.argmax(y_batch, axis = 1)   # 행 별로 최대값을 뽑는다.
    cnt += np.sum(np.array(p) == np.array(t_batch))
print('예측숫자와 실제숫자가 일치하는 개수:', cnt)
print('정확도:', round(cnt/len(x) * 100, 2), '%')
```

예측숫자와 실제숫자가 일치하는 개수: 9352
정확도: 93.52 %

---

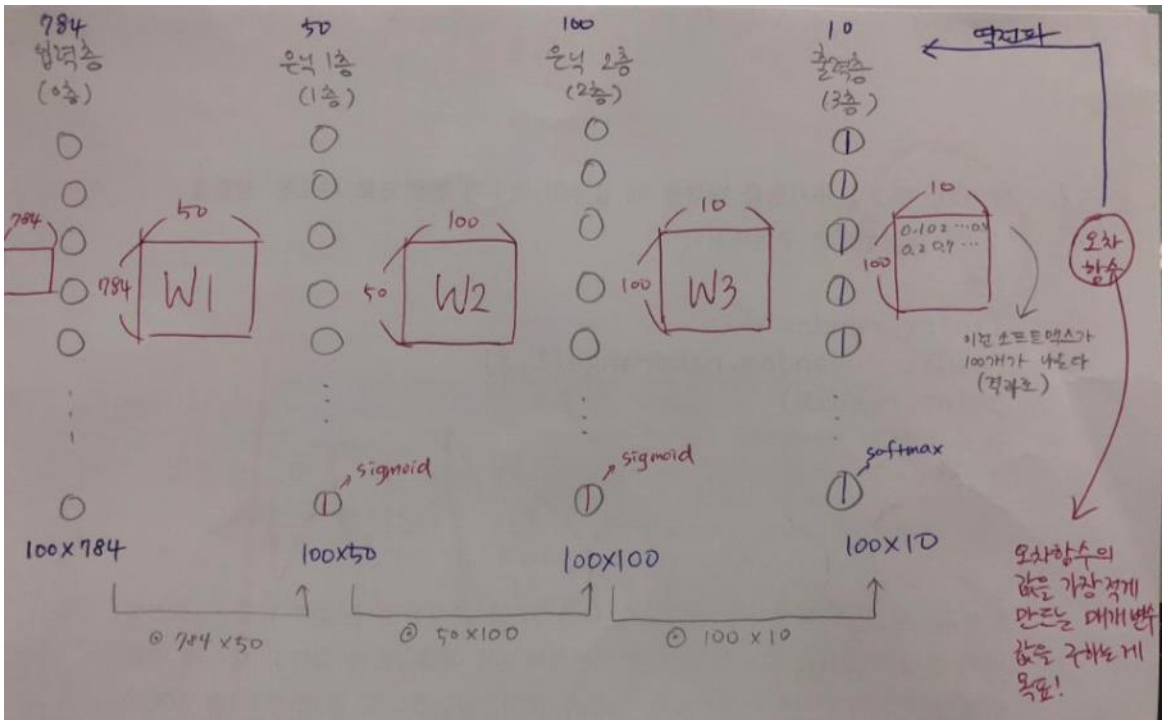
06 신경망 학습

2018년 8월 13일 월요일 오후 4:34

Table of Contents

학습목표

" 신경망의 가중치와 bias를 학습시키는 방법을 배우는 챕터 "



cost function의 결괏값을 가장 작게 만드는 가중치 매개변수를 찾는 것이 이 학습의 목표

비용(손실) 함수의 종류 2가지

1. Mean Squared Error, MSE (평균 제곱 오차)

2. Cross Entropy Error, CEE (교차 엔트로피 오차)

사용하는 경우

1. 회귀 : 항등 함수의 경우
   Mean Squared Error, MSE (평균 제곱 오차)

2. 분류 : 소프트맥스 함수의 경우
   Cross Entropy Error, CEE (교차 엔트로피 오차)

Mean Squared Error, MSE (평균 제곱 오차)

문제 101) 평균제곱 오차 함수를 책 112페이지 식을 참고해서 함수를 만들어 구현하시오

답)

```
import numpy as np

def mean_squared_error(y, t):
    return 0.5 * np.sum((y-t)**2)
```

```
t = [0, 0, 1, 0, 0, 0, 0, 0, 0, 0]    # target
y = [0.1, 0.05, 0.6, 0.0, 0.05, 0.1, 0.0, 0.1, 0.0, 0.0]    # predict
print(mean_squared_error(np.array(y), np.array(t)))
```

0.09750000000000003

문제 102) 아래의 확률 벡터를 Mean Squared Error 함수를 이용해서 target과 predict 값의 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)

```
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]

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

```
    y = eval('y' + str(i))
    print('y' + str(i), '의 오차율:', round(mean_squared_error(np.array(y), np.array(t))*100, 2), '%')
```

y1 의 오차율: 42.25 %
y2 의 오차율: 51.25 %
y3 의 오차율: 43.25 %
y4 의 오차율: 30.75 %
y5 의 오차율: 20.75 %
y6 의 오차율: 12.75 %
y7 의 오차율: 6.75 %
y8 의 오차율: 5.0 %
y9 의 오차율: 0.75 %

문제 103) 밑이 자연상수이고 진수가 0.6인 로그 값을 출력하시오. e (자연상수, 무리수)

답)

```
import numpy as np

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

-0.5108256237659907

문제 104) 밑수가 10이고 진수가 0.6인 로그의 로그 값을 출력하시오

답)

```
# 방법 1
import math
print(math.log(0.6, 10))

# 방법 2
import numpy as np
print(np.log10(0.6))
```

-0.22184874961635637
-0.2218487496163564

문제 105) mean square error 함수의 공식을 가지고 0.1과 0.9 확률의 오차를 각각 출력하시오

답)

```
x = np.array([0, 0, 0.1, 0, 0, 0, 0, 0, 0, 0.9])  # 예측값이 숫자 9를 예측하는 확률 벡터
```

```
y = np.array([0, 0, 0.9, 0, 0, 0, 0, 0, 0, 0.1])  # 예측값이 숫자 2를 예측하는 확률 벡터
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))
print(mean_squared_error(y, t))
```

0.81
0.009999999999999998

Cross Entropy Error, CEE (교차 엔트로피 오차)

공식

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

문제 106) 밑수가 자연상수이고 진수가 0인 로그의 로그값은 얼마인지 출력하시오

답)

```
np.log(0)
```

-inf

문제 107) Cross Entropy Error 함수를 아래의 공식을 보고 생성하시오
(함수이름: cross_entropy_error)

보기)

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

답)

```
def cross_entropy_error(y, t):
    delta = 1e-7
    return -np.sum(t * np.log(y+delta))
            # log0 이 -inf 로 나오기 때문에 계산하기 위해 아주 작은 숫자를 더해주는 작업
```

```
y = np.array([0, 0, 0.9, 0, 0, 0, 0, 0, 0, 0.1])  # 예측값이 숫자 2를 예측하는 확률 벡터
t = np.array([0, 0, 1, 0, 0, 0, 0, 0, 0, 0])

print(cross_entropy_error(y, t))
```

설명)

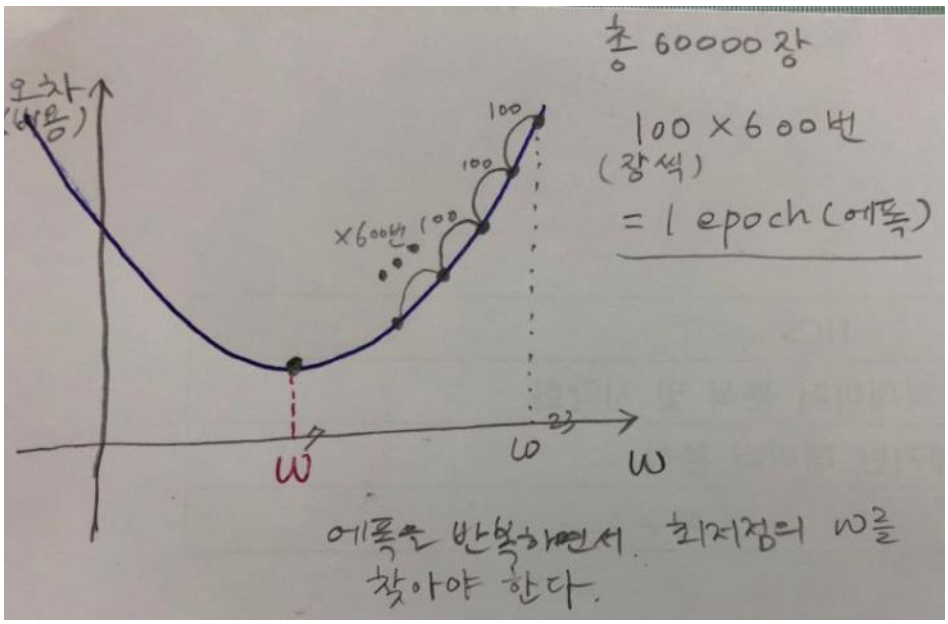
책 115쪽 위의 코드에서 delta를 더하고 있는 이유는 np.log() 함수에 0을 입력하면 마이너스 무한대를 뜻하는 -inf가 되어 더 이상 계산을 진행할 수 없기 때문입니다.

아주 작은 값을 더해서 절대 0이 되지 않도록, 즉 마이너스 무한대가 발생하지 않도록 조치한 것이다.

미니배치 학습

훈련 데이터 중에 일부만 골라서 학습하는 방법

표본을 뽑아서 학습시킨다



복원추출이든 비복원추출이든

문제 108) 60000 숫자 중에 무작위로 10개를 출력하시오

답)

```
import numpy as np

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

[45821 21272 43202 45735 34025 5620 2036 45211 59237 58987]

문제 109) MNIST의 테스트 데이터 10000장 중에 랜덤으로 100장을 추출하는 코드를 작성하시오

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(len(t), 100)
    x_batch = x[batch_mask]  # 복원 추출 코드
    t_batch = t[batch_mask]  # 복원 추출 코드
#     x_batch = x[i:i+batch]  # 비복원 추출 코드
#     t_batch = t[i:i+batch]  # 비복원 추출 코드
    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('정확도:', round(cnt/len(x) * 100, 2), '%')
```

문제 110) 위의 코드를 5 에폭(epoch) 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

for j in range(5):
    for i in range(0, 10000, 100):
        batch_mask = np.random.choice(len(t), 100)
        x_batch = x[batch_mask]   # 복원 추출 코드
        t_batch = t[batch_mask]   # 복원 추출 코드
    #     x_batch = x[i:i+batch]  # 비복원 추출 코드
    #     t_batch = t[i:i+batch]  # 비복원 추출 코드
        y = predict(network,x_batch)
        z = np.argmax(y, axis=1)
        cnt += (sum(z==t_batch))
print(len(x)*5, '개 중에서 일치하는 수 :', cnt)
print('정확도:', round(cnt/(len(x)*5) * 100, 2), '%')
```

50000 개 중에서 일치하는 수 : 46700
정확도: 93.4 %

미니배치 처리에 맞도록 Cross Entropy Error 함수를 구성하는 방법

미니배치하기 전의 교차엔트로피 함수

```
def cross_entropy_error(y, t):
    delta = 1e-7
    return -np.sum(t * np.log(y+delta))
            # log0 이 -inf 로 나오기 때문에 계산하기 위해 아주 작은 숫자를 더해주는 작업



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(cross_entropy_error(y, t))
```

100장 씩 미니배치한 후의 교차엔트로피 함수

```
def cross_entropy_error(y, t):
    delta = 1e-7
    return -np.sum(t * np.log(y+delta)) / y.shape[0]
            # log0 이 -inf 로 나오기 때문에 계산하기 위해 아주 작은 숫자를 더해주는 작업

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],
                [0.9, 0, 0.1, 0, 0, 0, 0, 0, 0, 0.1],
                                :
                                :   100개
                                :
                [0.1, 0, 0.9, 0, 0, 0, 0, 0, 0, 0.1]])
```

```
def cross_entropy_error(y, t):
    delta = 1e-7
    return -np.sum(t * np.log(y+delta)) / y.shape[0]
            # log0 이 -inf 로 나오기 때문에 계산하기 위해 아주 작은 숫자를 더해주는 작업

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],
                [0.9, 0, 0.1, 0, 0, 0, 0, 0, 0, 0.1],
                                :
                                :   100개
                                :
                [0.1, 0, 0.9, 0, 0, 0, 0, 0, 0, 0.1]])
```

SGD (Stochastic Gradient Descent) - 확률적 경사 하강법

100장을 랜덤으로 무작위로(미니배치로) 선정(수집)해서 신경망 학습을 시키는 경사하강법

이 경사하강법으로 매개변수를 갱신하는데 이때 Cost Function 을 최저점으로 하는 점을 찾기 위해 기울기를 찾아야 한다.

이 때 필요한게 바로 '미분' 이다

수치미분

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

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

위의 미분 공식을 파이썬 함수로 구현

```
def numerical_diff(f, x):
    h = 10e-50  # 0에 아주 가까운 숫자 (0.0000..1)
    return (f(x+h) - f(x)) / h
```

위의 코드의 문제점

너무 작은 값을 이용하면 컴퓨터로 계산하는 데 문제가 된다는 것이다.

아래와 같이 해보면 알 수 있다.

```
import numpy as np

np.float32(1e-50)
```

0.0

따라서 10의 -4제곱 정도의 값을 사용하면 좋은 결과가 나온다고 알려져 있다

```
def numerical_diff(f, x):
    h = 1e-4  # 0.0001
    return (f(x+h) - f(x)) / h
```

또 문제점이 하나 더 있다. 위의 공식은 진정한 미분(접선)을 하지 못한다.
따라서, 수치 미분(할선 공식)을 써야 한다.

```
def numerical_diff(f, x):
    h = 1e-4  # 0.0001
    return (f(x+h) - f(x-h)) / (2*h)
```

문제 111) 아래의 비용함수를 수치미분 함수로 미분 하는데 x=4 에서의 미분계수(기울기)를 구하시오

보기)

```
y = x**2 + 4**2 (비용함수)
```

답)

```
# 수치미분 함수
def numerical_diff(f, x):
    h = 1e-4  # 0.0001
    return (f(x+h) - f(x-h)) / (2*h)

# 비용함수
def loss_func(x):
    return x**2 + 4**2

print(numerical_diff(loss_func, 4))
```

7.999999999999119

문제 112) 문제 111번의 비용함수를 아래와 같이 시각화 하시오

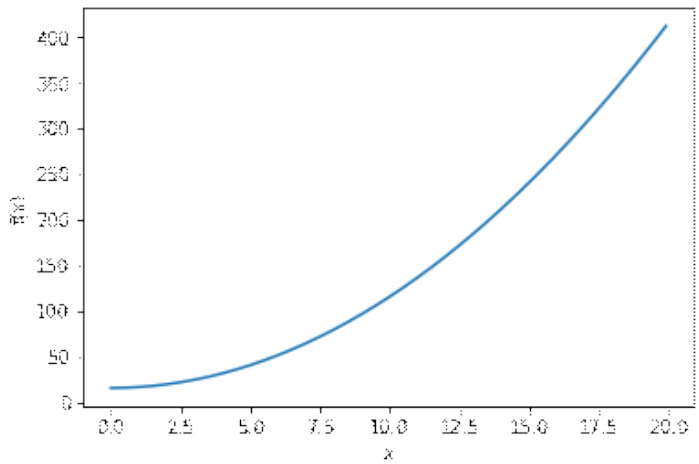
보기)

```
y = x**2 + 4**2 (비용함수)
```

답)

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

# 수치미분 함수
def numerical_diff(f, x):
    h = 1e-4  # 0.0001
    return (f(x+h) - f(x-h)) / (2*h)

# 비용함수
def loss_func(x):
    return x**2 + 4**2



x = np.arange(0.0, 20.0, 0.1)
```

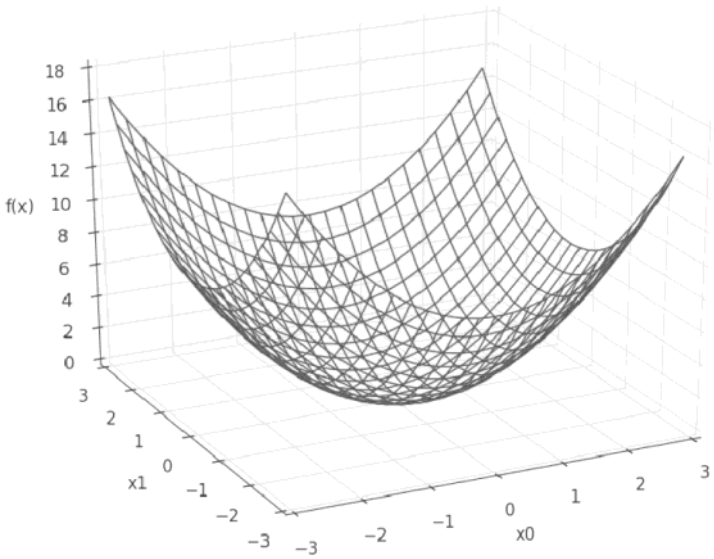
```
y = loss_func(x)
plt.xlabel("x")
plt.ylabel("f(x)")
plt.plot(x, y)
plt.show()
```



문제 113) 아래의 식을 구글에서 시각화 하시오

보기)

```
f(x0, x1) = x0**2 + x1**2
```



답)

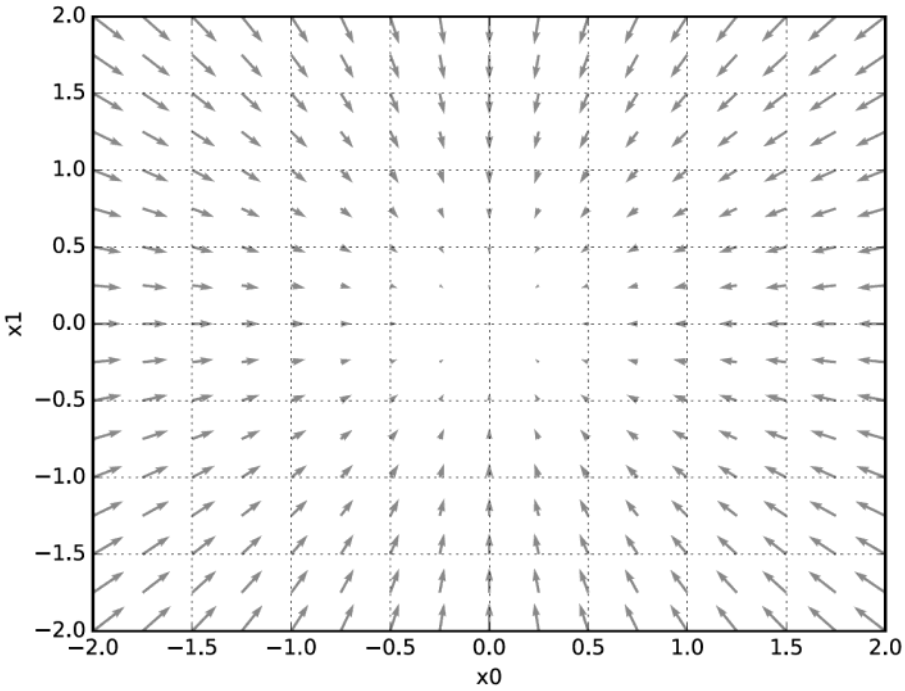
```
구글 홈페이지에 z=x^2+y^2 라고 검색한다.
```

편미분

```
변수가 2개 이상인 함수를 미분할 때 미분 대상 변수 외에 나머지 변수를 상수처럼 고정시켜 미분하
```

는 것을 편미분이라고 한다

f(x0, x1) = x0**2 + x1**2
f(w0, w1) = w0**2 + w1**2



문제 114) f(x0, x1) = x0^2 + x1^2 함수를 편미분하는데 x0=3, x1=4 일 때, 편미분하시오

답)

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

# 수치미분 함수
def numerical_diff(f, x):
    h = 1e-4  # 0.0001
    return (f(x+h) - f(x-h)) / (2*h)

# 비용함수
def loss_func(x):
    return x**2 + 4**2

numerical_diff(loss_func, 3)
```

6.00000000000378

문제 115) f(x0, x1) = x0^2 + x1^2 함수를 편미분하는데 x0=3, x1=4 일 때, x1에 대해 편미분하시오

답)

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

# 수치미분 함수
def numerical_diff(f, x):
    h = 1e-4  # 0.0001
    return (f(x+h) - f(x-h)) / (2*h)

# 비용함수
def loss_func(x):
    return 3**2 + x**2

numerical_diff(loss_func, 4)
```

7.999999999999119

이렇게 일일이 비용함수를 바꿔가면서 하는 것은 말이 안된다.
만약에 W(차수)가 100차면 100번 바꿔야 한다

문제 116) 아래의 numpy 배열과 같은 구조의 행렬이 출력되는데 값은 0으로 출력되게 하시오

보기)

```
x = np.array([3.0, 4.0])
```

결과가 [0, 0] 나오도록

답)

```
import numpy as np

x = np.array([3.0, 4.0])
print(np.zeros_like(x))
```

[0. 0.]

기울기

문제 117) 위에서는 f(x0, x1) = x0^2 + x1^2 함수를 편미분하는 것을 각각 수행했는데 이번에는

편미분이 한번에 수행되게끔 하는 코드를 작성하시오

보기)

책 p.127의 아래에 나오는 numerical_gradient 함수를 만드시오

답)

```
# 수치미분 함수
def numerical_diff(f, x):
    h = 1e-4  # 0.0001
    return (f(x+h) - f(x-h)) / (2*h)

# 비용함수
def loss_func(x):
    return np.sum(x**2)

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

    for idx in range(x.size):
        tmp_val = x[idx]

        # f(x+h) 계산
        x[idx] = tmp_val + h
        fxh1 = f(x)

        # f(x-h) 계산
        x[idx] = tmp_val - h
        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.]

문제 118) 아래의 x0, x1 지점에서의 기울기를 각각 구하시오

보기)

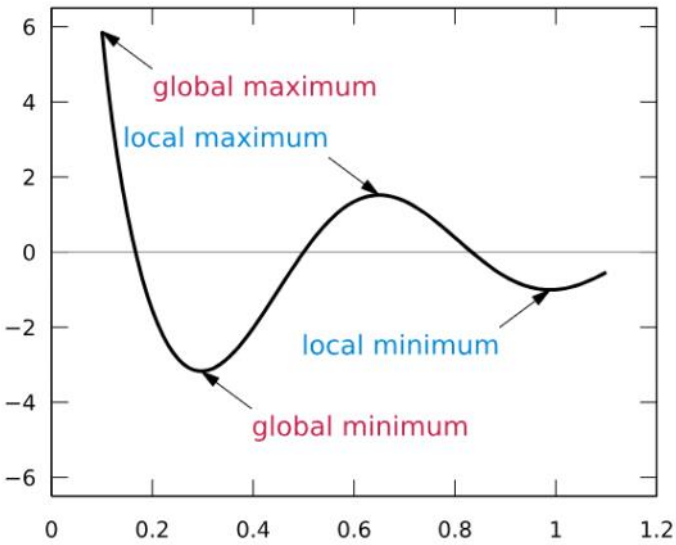
```
[3.0, 4.0] -----> [6.0, 8.0]
[0.0, 2.0] ----->
[3.0, 0.0] ----->
```

답)

```
# 수치미분 함수
def numerical_diff(f, x):
    h = 1e-4  # 0.0001
    return (f(x+h) - f(x-h)) / (2*h)

# 비용함수
def loss_func(x):
    return np.sum(x**2)

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

    for idx in range(x.size):
        tmp_val = x[idx]

        # f(x+h) 계산
        x[idx] = tmp_val + h
        fxh1 = f(x)

        # f(x-h) 계산
        x[idx] = tmp_val - h
        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])))
print(numerical_gradient(loss_func, np.array([0.0, 2.0])))
print(numerical_gradient(loss_func, np.array([3.0, 0.0])))
```

```
[6. 8.]
[0. 4.]
[6. 0.]
```

경사법(경사 하강법) - learning rate



문제 119) 위에서 만든 numerical_gradient 함수를 100번 수행되게 해서 global minima에 도달하게끔 하는 코드를 구현하시오

답)

```
# 수치미분 함수
def numerical_diff(f, x):
    h = 1e-4  # 0.0001
    return (f(x+h) - f(x-h)) / (2*h)

# 비용함수
def loss_func(x):
    return np.sum(x**2)

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

    for idx in range(x.size):
        tmp_val = x[idx]

        # f(x+h) 계산
        x[idx] = tmp_val + h
        fxh1 = f(x)

        # f(x-h) 계산
```

```
        x[idx] = tmp_val - h
        fxh2 = f(x)

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

    return grad

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

init_x = np.array([3.0, 4.0])
print(gradient_descent(loss_func, init_x, 0.1, 100))
```

[6.11110793e-10 8.14814391e-10]

문제 120) lr(Learning Rate)을 크게 주면 어떻게, 작게 주면 어떻게 결과가 나오는지 확인하시오

보기)

```
lr = 10
lr =  1e-10
```

답)

```
# 수치미분 함수
def numerical_diff(f, x):
    h = 1e-4  # 0.0001
    return (f(x+h) - f(x-h)) / (2*h)

# 비용함수
def loss_func(x):
    return np.sum(x**2)

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

```
    for idx in range(x.size):
        tmp_val = x[idx]

        # f(x+h) 계산
        x[idx] = tmp_val + h
        fxh1 = f(x)

        # f(x-h) 계산
        x[idx] = tmp_val - h
        fxh2 = f(x)

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

    return grad

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

init_x = np.array([3.0, 4.0])
print(gradient_descent(loss_func, init_x, 10, 100))

init_x = np.array([3.0, 4.0])
print(gradient_descent(loss_func, init_x, 1e-10, 100))
```

```
[ 2.58983747e+13 -1.29524862e+12]
[2.99999994 3.99999992]
```

문제 121) 2 x 3 행렬을 생성하는데 값은 랜덤 값으로 생성되게 하시오

답)

```
import numpy as np

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

```
[[-1.15815693 -1.03718435  0.03311024]
```

```
 [ 0.04157162 -1.4255968    0.49998371]]
```

문제 122) 위에서 구한 w값으로 아래의 입력값과 행렬의 내적을 출력하시오

보기)

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

답)

```
import numpy as np

x = np.array([0.6, 0.9])  # 1 x 2 행렬
w = np.random.randn(2, 3)  # 2 x 3 행렬

np.dot(x, w)
```

```
array([-0.83951829, -0.74994459,  0.45038604])
```

문제 123) 가중치를 랜덤으로 생성하는 __init__ 함수를 생성하시오

답)

```
x = np.array([0.6, 0.9])
import numpy as np

# 가중치 랜덤으로 생성
def __init__():
    w = np.random.rand(2, 3)
    return w

__init__()
```

```
[[0.47499735 0.26928766 0.04306079]
 [0.88458809 0.93261392 0.40295721]]
```

문제 124) 위에서 만든 가중치와 아래의 입력값과 행렬 내적을 하는 함수를 predict 로 생성하시오

답)

```
x = np.array([0.6, 0.9])
import numpy as np
```

```
# 가중치 랜덤으로 생성
def __init__():
    w = np.random.rand(2, 3)
    return w

# 랜덤으로 생성한 가중치와 입력값 x를 내적
def predict(x, w):
    return np.dot(x, w)

predict(x, w)
```

```
array([-0.83951829, -0.74994459,  0.45038604])
```

문제 125) 위의 predict에서 나온 결과를 softmax 출력층 함수에 통과시킨 결과가 무엇인가?

답)

```
import numpy as np
x = np.array([0.6, 0.9])



# 가중치 랜덤으로 생성
def __init__():
    w = np.random.rand(2, 3)
    return w

# 랜덤으로 생성한 가중치와 입력값 x를 내적
def predict(x, w):
    return np.dot(x, w)

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

a = predict(x, w)
print(softmax(a))
```

```
[0.1746375  0.19100242 0.63436008]
```

문제 126) 위에서 출력한 소프트맥스 함수의 결과와 아래의 target 값과의 오차를 출력하기 위해

cross_entropy_error 함수를 생성하고 통과한 결과를 출력하시오

답)

```
import numpy as np

# 가중치 랜덤으로 생성
def __init__():
    w = np.random.rand(2, 3)
    return w

# 랜덤으로 생성한 가중치와 입력값 x를 내적
def predict(x, w):
    return np.dot(x, w)

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

# CEE 생성 (Cross Entropy Error)
def cross_entropy_error(y, t):
    delta = 1e-7
    return -np.sum(t * np.log(y+delta))



x = np.array([0.6, 0.9])
t = np.array([0, 0, 1])
a = predict(x, w)
b = softmax(a)
print(cross_entropy_error(b, t))
```

0.4551383843637972

문제 126) 위에서 출력한 소프트맥스 함수의 결과와 아래의 target 값과의 오차를 출력하기 위해 cross_entropy_error 함수를 생성하고 통과한 결과를 출력하시오

답)

```
import numpy as np

# 가중치 랜덤으로 생성
def __init__():
    w = np.random.rand(2, 3)
```

```
    return w

# 랜덤으로 생성한 가중치와 입력값 x를 내적
def predict(x, w):
    return np.dot(x, w)

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

# CEE 생성 (Cross Entropy Error)
def cross_entropy_error(y, t):
    delta = 1e-7
    return -np.sum(t * np.log(y+delta))



x = np.array([0.6, 0.9])
t = np.array([0, 0, 1])
w = __init__()
y = predict(x, w)
softmax_y = softmax(y)
print(cross_entropy_error(b, softmax_y))
```

문제 127) 위의 3가지 함수를 다 사용한 simplenet 이라는 클래스를 책 134 페이지를 보고 생성하고 135페이지를 보며 클래스가 잘 생성되었는지 확인하시오 (마지막 문제)

답)

```
import numpy as np

class simpleNet:
    # 가중치 랜덤으로 생성
    def __init__(self):
        self.W = np.random.rand(2, 3)

    # 랜덤으로 생성한 가중치와 입력값 x를 내적
    def predict(self, x):
        return np.dot(x, self.W)

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

```
    # CEE 생성 (Cross Entropy Error)
    def cross_entropy_error(self, x, t):
        delta = 1e-7
        return -np.sum(t * np.log(x+delta))


net = simpleNet()
print('가중치 매개변수:₩n', net.W)

x = np.array([0.6, 0.9])
p = net.predict(x)
print('₩n입력값 x와 가중치 w를 곱해준 값:₩n', p)

print('₩n최대값의 인덱스:₩n', np.argmax(p))

t = np.array([0, 0, 1]) # 정답 레이블
softmax_p = net.softmax(p) # p에 소프트맥스 함수 통과한 값

print('₩n결과:₩n', net.cross_entropy_error(softmax_p, t))
```

```
가중치 매개변수:
 [[0.35196232 0.24128007 0.72909494]
 [0.46175271 0.11353967 0.33078632]]

입력값 x와 가중치 w를 곱해준 값:
 [0.62675483 0.24695374 0.73516465]

최대값의 인덱스:
 2

결과:
 0.9206741471247873
```

이전까지 배웠던 딥러닝 수업 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)

오차(비용) 함수를 미분하기 위해 수치미분 함수 사용

3층 신경망 구현을 위한 클래스 생성

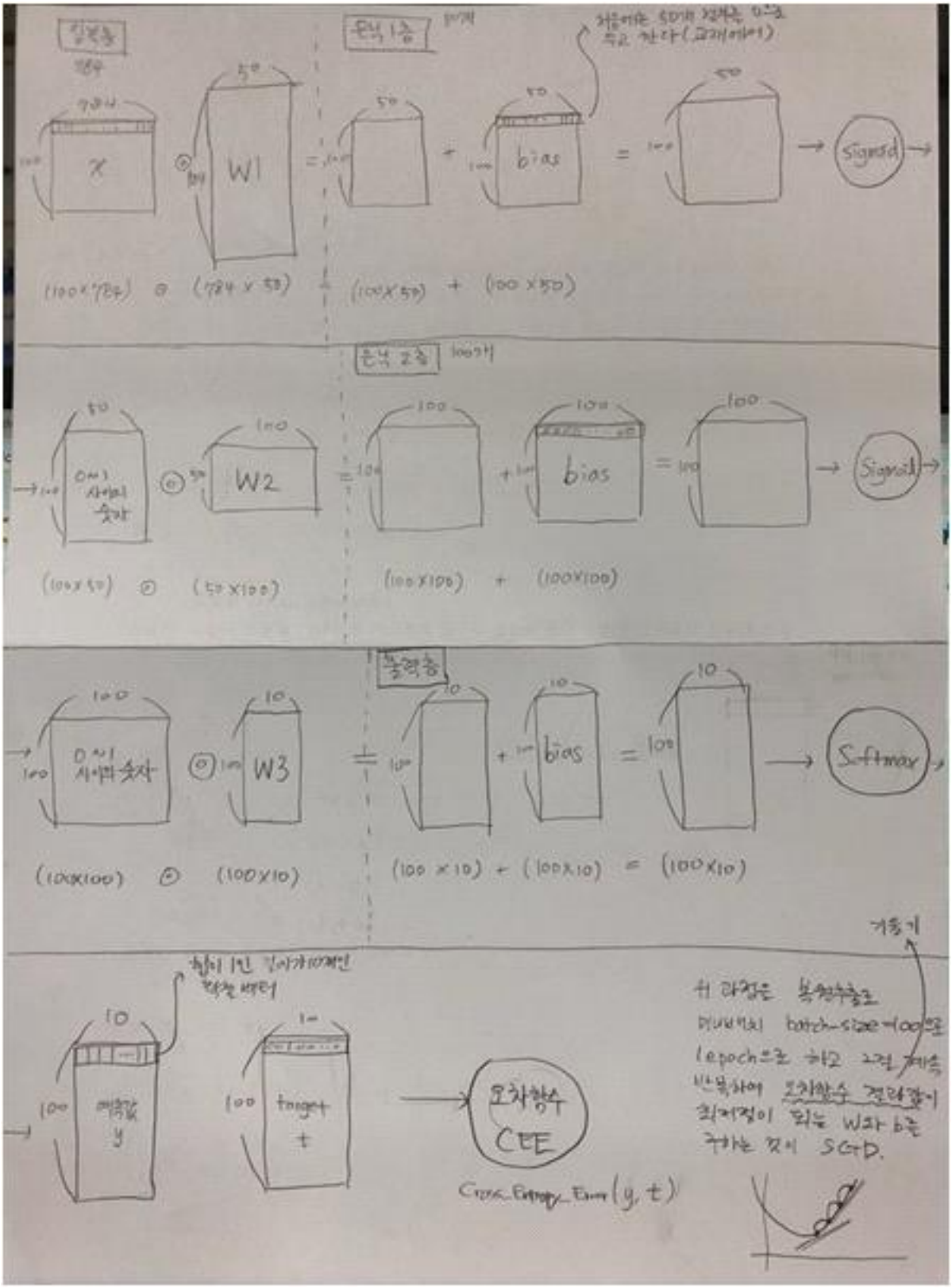
학습목표 : "오차함수를 미분해서 기울기를 가지고 가중치와 바이어스를 갱신하여 최적의 가중치와 바이어스를 지닌 신경망을 만드는게 목표"

5장. 3층 신경망 학습 (오차역전파를 이용해서 학습시킴)

6장. 신경망의 정확도를 올리기 위한 여러가지 방법들

7장. CNN으로 신경망 구현

3층 신경망 구조 손필기 캡처



수치미분을 이용한 2층 신경망 코드

```
# 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 crossEntropyError(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  # 嫒?蹂듭쌥
        it.iternext()

    return grad


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)
        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 crossEntropyError(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)

# 하이퍼파라미터
iters_num = 10000  # 반복 횟수를 적절히 설정한다.
train_size = x_train.shape[0]
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):
    # 미니배치 획득
    batch_mask = np.random.choice(train_size, batch_size)
    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)

    # 1에폭당 정확도 계산
    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 | " + 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()
```

위의 코드 분해해서 이해하기

문제 128) TwoLayerNet 클래스로 객체를 생성하고 W1, W2, b1, b2 의 행렬의 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
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 crossEntropyError(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  # 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  # 嫒?蹂듭쌥
        it.iternext()

    return grad



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)
        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 crossEntropyError(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,)

문제 129) TwoLayerNet 클래스의 객체를 생성하고 predict 메소드를 실행하여 나온 결과의 행렬의 shape를 출력하시오

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=50, output_size=10)
network.predict(x_train)[:100].shape
```

(100, 10)

문제 130) TwoLayerNet 클래스의 객체를 생성하고 crossEntropyError(y, t) 메소드를 실행해서 나온 출력결과인 오차를 출력하시오

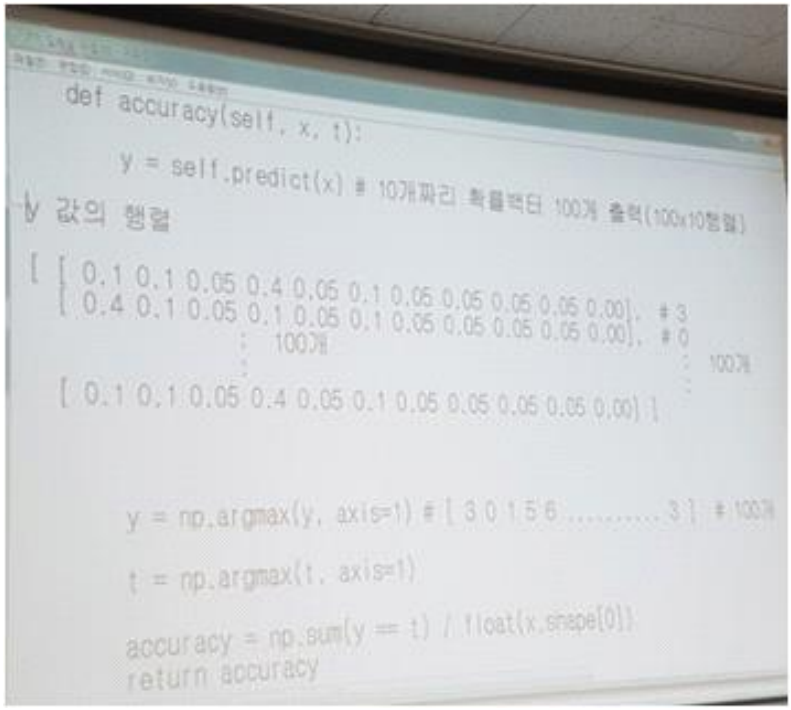
답)

```
# 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=50, output_size=10)
y = network.predict(x_train)[:100]
t = t_train[:100]
print('오차:', crossEntropyError(y, t))
```

오차: 13.246389060227672

accuracy 함수의 이해

예상한 숫자와 실제 숫자를 비교해서 정확도를 출력하는 함수



예제:

```
def accuracy(self, x, t):
        y = self.predict(x) # 10개짜리 확률백터 100개 출력(100x10행렬)

- y 값의 행렬

 [ [ 0.1 0.1 0.05 0.4 0.05 0.1 0.05 0.05 0.05 0.05 0.00],  # 3
   [ 0.4 0.1 0.05 0.1 0.05 0.1 0.05 0.05 0.05 0.05 0.00],  # 0
                :  100개                                    :  100개
                :                                          :
   [ 0.1 0.1 0.05 0.4 0.05 0.1 0.05 0.05 0.05 0.05 0.00] ]

- target 의 행렬 (one hot encoding)

           [ [ 0 0 1 0 0 0 0 0 0 0  ],
             [ 1 0 0 0 0 0 0 0 0 0  ],
                       :
                       :     100개
             [ 0 0 1 0 0 0 0 0 0 0  ] ]

        y = np.argmax(y, axis=1) # [ 3 0 1 5 6 ..... 3 ]  # 100개
        t = np.argmax(t, axis=1) # [ 2 0 2 1 3 .....3 ]  # 100개
        accuracy = np.sum(y == t) / float(x.shape[0])
        return accuracy
```

문제 131) TwoLayerNet 클래스로 객체를 생성하고 accuracy(x, t) 메소드를 실행해서 100장

훈련데이터와 100장의 target 데이터를 입력했을 때의 정확도를 출력하시오

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=50, output_size=10)
y = x_train[:100]
t = t_train[:100]
print('오차:', network.loss(y, t))

# 정확도
print('정확도:', network.accuracy(y, t))
```

오차: 6.905222086271415
정확도: 0.19

numerical_gradient 함수

비용함수와 가중치 또는 Bias를 입력받아 기울기를 출력하는 함수

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 것입니다.
numerical_gradien(f, x) 내부에서 f(x)를 실행하는데 그와의 일관성을 위해서 f(W)를 정의한 것
```

예제)

```
여기서 정의한 f(W) 함수의 W는 더미로 만든 것입니다.
numerical_gradien(f, x) 내부에서 f(x)를 실행하는데 그와의 일관성을 위해서 f(W)를 정의한 것
```

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 common.functions import *
```

```
#from 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 crossEntropyError(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  # 嫒?蹂듭쌥
        it.iternext()

    return grad


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)
        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 crossEntropyError(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(network.loss(x_train[:100], t_train[:100]))

# 하이퍼파라미터
iters_num = 10000  # 반복 횟수를 적절히 설정한다.
train_size = x_train.shape[0]
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)


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)
```

lambda 표현식 이란?

```
# 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 common.functions import *
#from 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 crossEntropyError(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  # 嫄?蹂듭썝
        it.iternext()

    return grad


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)
        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 crossEntropyError(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(network.loss(x_train[:100], t_train[:100]))

# 하이퍼파라미터
iters_num = 10000  # 반복 횟수를 적절히 설정한다.
train_size = x_train.shape[0]
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)



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)
```

문제 132) 아래의 비용함수를 lambda 표현식으로 한줄로 변경해서 출력하시오

보기)

```
def f(W):
    return  network.loss(x,t)
```

답)

```
f = lambda W: network.loss(x,t)
```

문제 133) TwoLayerNet 클래스를 가지고 학습 시키는 코드를 완성시키시오

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=50, output_size=10)
```

```
# 하이퍼파라미터
iters_num = 10000  # 반복 횟수를 적절히 설정한다.
train_size = x_train.shape[0]   # x_train : (60000, 784)
batch_size = 100  # 미니배치 크기
learning_rate = 0.1

# 그래프 그릴때 필요한 데이터를 담을 리스트 변수 3개를 선언
train_loss_list = [] # 훈련데이터의 손실
train_acc_list = []  # 훈련데이터의 정확도
test_acc_list = []   # 테스트데이터의 정확도

# 1에폭당 반복 수
iter_per_epoch = max(train_size / batch_size, 1)
                    # 60000 / 100 = 600 => max(600, 1) = 600

for i in range(iters_num):
    # 미니배치 획득  # 랜덤으로 100개 씩 뽑는 작업을 10000번 수행
    batch_mask = np.random.choice(train_size, batch_size)
    # 0 ~ 60000 의 숫자 중에 100개를 랜덤으로 생성
    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)
    # 오차역전파를 이용한 기울기 구하는 함수 (빠름, 나중에 배울 예정)

    # 매개변수 갱신
    for key in ('W1', 'b1', 'W2', 'b2'):
        network.params[key] -= learning_rate * grad[key]

    # 학습 경과 기록   # 손실(비용)이 점점 줄어드는 것을 보기 위해
    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()
```

문제 134) 수치미분이 아닌 오차 역전파를 이용한 신경망 학습으로 2층 신경망 코드를 구현하시오

답)

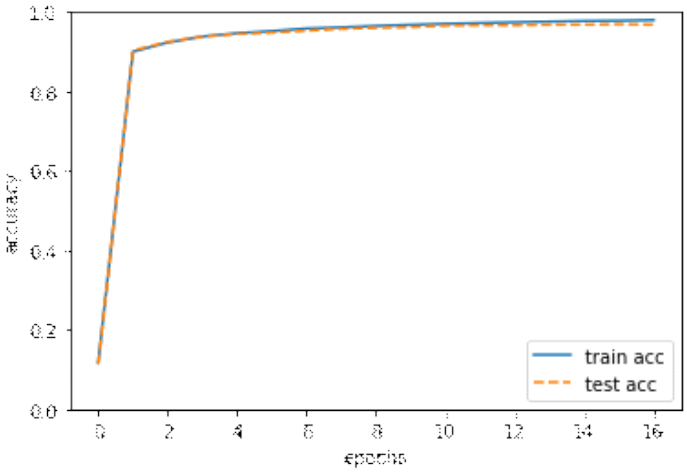
```
# coding: utf-8
import sys, os
sys.path.append(os.pardir)  # 부모 디렉터리의 파일을 가져올 수 있도록 설정
import numpy as np
from common.layers import *
from common.gradient import numerical_gradient
from collections import OrderedDict
import matplotlib.pyplot as plt
from dataset.mnist import load_mnist

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)

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.
        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 | 0.11691666666666667, 0.1131
train acc, test acc | 0.8996833333333333, 0.9028
train acc, test acc | 0.9230333333333334, 0.9249
train acc, test acc | 0.9380666666666667, 0.9376
train acc, test acc | 0.9465666666666667, 0.9445
train acc, test acc | 0.9517833333333333, 0.9481
train acc, test acc | 0.9583166666666667, 0.9528
train acc, test acc | 0.96115, 0.9574
train acc, test acc | 0.9648166666666667, 0.9601
train acc, test acc | 0.9676333333333333, 0.9618
train acc, test acc | 0.9701833333333333, 0.9656
train acc, test acc | 0.9721666666666666, 0.9666
train acc, test acc | 0.9739333333333333, 0.9663
train acc, test acc | 0.9757333333333333, 0.9677
train acc, test acc | 0.9772666666666666, 0.9678
train acc, test acc | 0.9779333333333333, 0.9692
train acc, test acc | 0.9795333333333334, 0.9679
```


1.0
0.8
0.6
0.4
0.2
0.0
accuracy
0
2
4
6
8
10
12
14
16
epochs
train acc
test acc

---

07 오차역전파법

2018년 8월 13일 월요일 오후 4:34

Table of Contents

역전파 란?

신경망 학습 처리에서 최소화되는 함수의 경사를 효율적으로 계산하기 위한 방법으로
"오류 역전파" 가 있다

비용함수의 기울기를 계산하는 방법

1. 수치미분
2. 오류역전파

오차역전파 란?

순전파 : 입력층 → 은닉층 → 출력층
역전파 : 입력층 ← 은닉층 ← 출력층

여기서 역전파를 시키는 것이 오류(오차) 입니다

출력층부터 차례대로 역방향으로 따라 올라가 각 층에 있는 노드의 오차를 계산할 수 있다.
각 노드의 오차를 계산하면 그 오차를 사용해서 함수의 기울기를 계산할 수 있다.

"즉 전파된 오차를 이용해서 가중치를 조정한다"
↓
"오류(오차) 역전파"

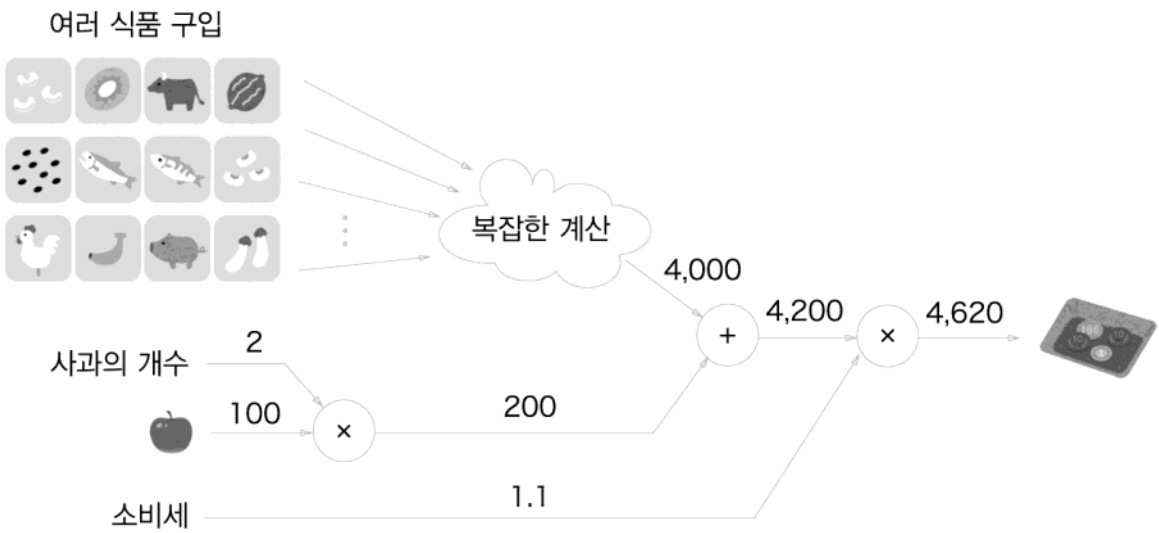
계산그래프

순전파와 역전파에 계산 과정을 그래프로 나타내는 방법

계산그래프의 장점

국소적 계산이 가능하다.

국소적 계산이란 전체에 어떤 일이 벌어지든 상관없이 자신과 관계된 정보만으로 다음 결과를 출력할 수 있다는 것이다.



4000원 이라는 숫자가 어떻게 계산되었느냐와는 상관없이 사과가 어떻게 200원이 되었는가만 신경쓰면 된다는 것이 국소적 계산이다

왜 계산 그래프로 문제를 해결 하는가?

전체가 아무리 복잡해도 각 노드에서 단순한 계산에 집중하여 문제를 단순화 시킬 수 있다.
(수치미분으로 하면 오~~~래 걸리기 때문에)

실제로 계산 그래프를 사용하는 가장 큰 이유는?

역전파를 통해서 미분을 효율적으로 계산할 수 있는 점에 있다

예)
사과값이 '아주 조금' 올랐을때 '지불금액'이 얼마나 증가하는지 알고 싶은 경우

∂ 지불금액

∂ 사과값

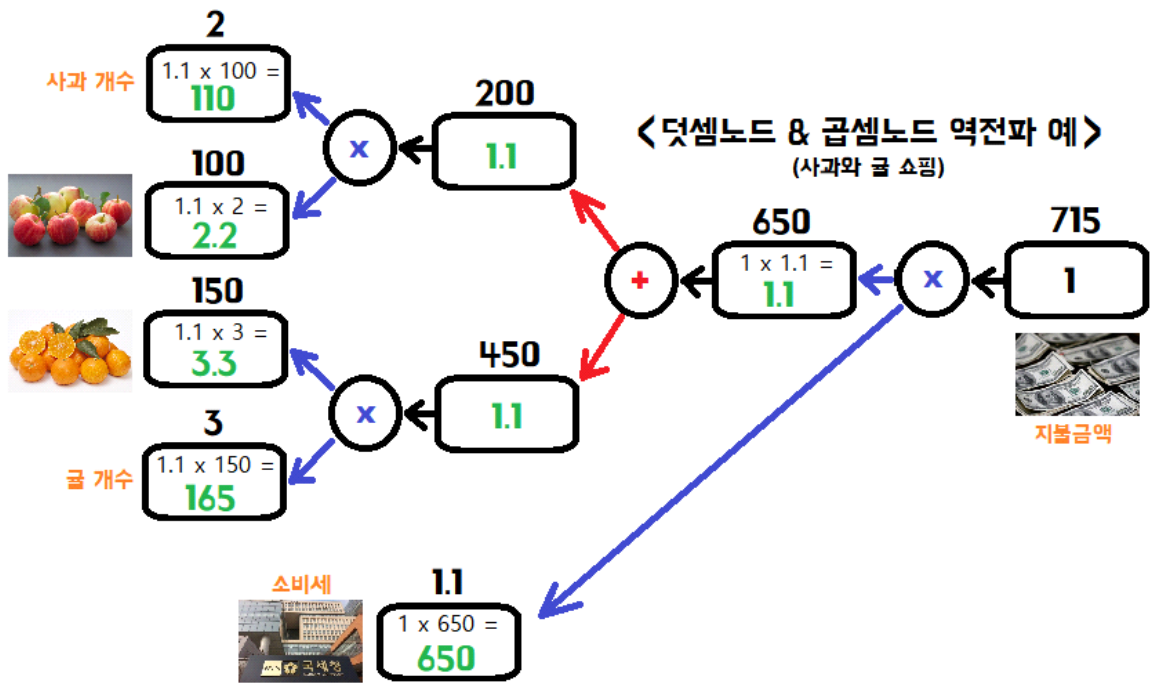
지불금액을 사과값으로 편미분하면 알 수 있다

편미분하는 방법 : 계산 그래프로 역전파를 이용한다

결과 : 사과값이 1원이 오르면 최종 금액은 2.2원이 오른다

도함수 [derivative, 導函數]

함수 y=f(x)을 미분하여 얻은 함수 f'(x)를 말한다. 일반적으로 f(x)의 미계수 또는 미분계수라고도 한다. x의 함수로 본 것을 **도함수**라고 하고 f'(x)로 나타낸다.



곱셈계층 파이썬으로 구현

문제 135) 곱셈계층을 파이썬으로 구현하시오

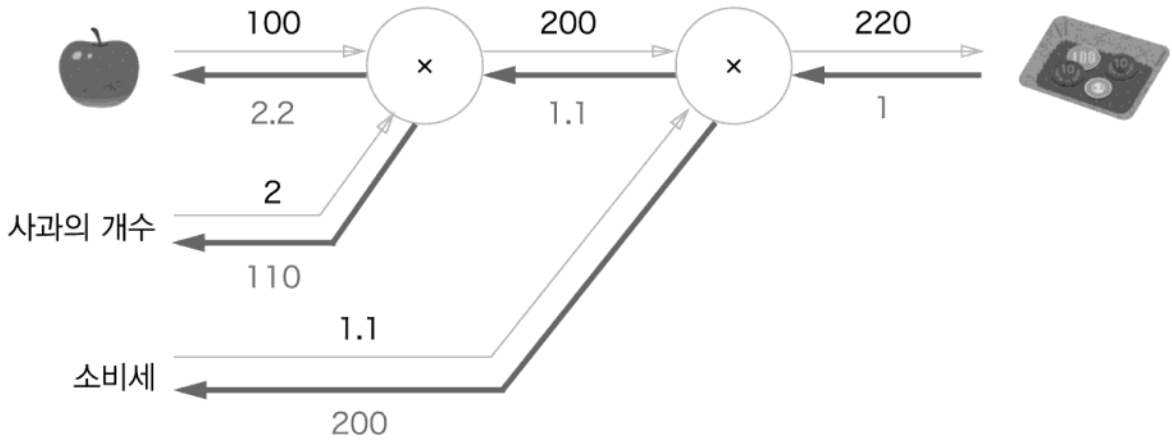
답)

```
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
```

문제 136) 위에서 만든 곱셈 클래스를 객체화 시켜서 사과 가격의 총 가격을 구하시오

보기)



답)

```
apple = 100
apple_num = 2
tax = 1.1

# 계층들
mul_apple_layer = MulLayer()
mul_tax_layer = MulLayer()

# 순전파
apple_price = mul_apple_layer.forward(apple, apple_num)
price = mul_tax_layer.forward(apple_price, tax)

print(price)
```

덧셈계층 파이썬으로 구현

문제 137) 덧셈계층을 파이썬으로 구현하시오

답)

```
class AddLayer:
```

```
    def __init__(self):
        self.x = None
        self.y = None

    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
```

문제 138) 층을 4개로 해서 사과와 귤의 총 가격을 구하는 책 148쪽의 그림 5-3의 신경망을 위에서 만든 곱셈계층과 덧셈계층으로 구현하시오

답)

```
apple = 100
apple_num = 2
orange = 150
orange_num = 3
tax = 1.1

# 계층 생성
mul_apple_layer = MulLayer()
mul_orange_layer = MulLayer()
add_apprange_layer = AddLayer()
tax_layer = MulLayer()

# 순전파
apple_price = mul_apple_layer.forward(apple, apple_num)
orange_price = mul_orange_layer.forward(orange, orange_num)
apprange_price = add_apprange_layer.forward(apple_price, orange_price)
final_price = tax_layer.forward(apprange_price, tax)

print(final_price)
```

715.0000000000001

문제 139) 문제 138번의 순전파로 출력한 과일 가격의 총합 신경망의 역전파를 구현하시오

답)

```
apple = 100
apple_num = 2
orange = 150
orange_num = 3
tax = 1.1

# 계층 생성
mul_apple_layer = MulLayer()
mul_orange_layer = MulLayer()
add_apprange_layer = AddLayer()
tax_layer = MulLayer()

# 순전파
apple_price = mul_apple_layer.forward(apple, apple_num)
orange_price = mul_orange_layer.forward(orange, orange_num)
apprange_price = add_apprange_layer.forward(apple_price, orange_price)
final_price = tax_layer.forward(apprange_price, tax)
# print(final_price)  # 순전파 결과

# 역전파
dprice = 1
dapprange_price, dtax = mul_tax_layer.backward(dprice)
dapple_price, dorange_price = add_apprange_layer.backward(dapprange_price)
dapple, dapple_num = mul_apple_layer.backward(dapple_price)
dorange, dorange_num = mul_orange_layer.backward(dorange_price)

# 역전파 결과
print('모든 과일 가격을 역전파한 값:', dapprange_price)
print('소비세를 역전파한 값:', dtax)

print('사과 가격 역전파 값:', dapple_price)
print('귤 가격 역전파 값:', dorange_price)

print('사과 단가 역전파 값:', dapple)
print('사과 개수 역전파 값:', dapple_num)

print('귤 단가 역전파 값:', dorange)
print('귤 개수 역전파 값:', dorange_num)
```

모든 과일 가격을 역전파한 값: 1.1
소비세를 역전파한 값: 200

사과 가격 역전파 값: 1.1
귤 가격 역전파 값: 1.1
사과 단가 역전파 값: 2.2
사과 개수 역전파 값: 110.00000000000001
귤 단가 역전파 값: 3.3000000000000003
귤 개수 역전파 값: 165.0

활성화 함수 계층 구현하기

1. Relu 계층

" 0보다 큰 값이 입력되면 그 값을 그대로 출력하고 0 이거나 0보다 작은 값이 입력되면 0을 출력하는 함수 "

문제 140) 책 166페이지의 Relu 클래스를 생성하시오

답)

```
class Relu:
    def __init__(self):
        self.mask = None

    def forward(self, x):
        self.mask = (x <= 0)  # x 에서 0보다 작은 값들에 True 들어감
        out = x.copy()
        out[self.mask] = 0  # out 에서 0보다 작은 값들 0 으로 바꿔줌

        return out

    def backward(self, dout):
        dout[self.mask] = 0
        dx = dout

        return dx
```

설명)
Relu 클래스를 이해하기 위해 2가지를 알아야 한다
1. copy의 의미 :

다른 주소값을 갖는 객체를 생성하여 값을 넣어주는 것 (수정 시 같이 수정되지 않도록 함)

2. x[x<=0] 의 의미 :

```
import numpy as np
```

```
x = np.array([[1.0, -0.5], [-2.0, 3.0]])
print(x)

mask = ( x<=0 )
print(mask)

out = x.copy()
out[mask] = 0
print(out)
```

```
[[ 1.  -0.5]
 [-2.   3. ]]
[[False  True]
 [ True False]]
[[1. 0.]
 [0. 3.]]
```

문제 141) Relu 클래스를 객체화 시켜 아래의 입력 데이터 x값을 받아 순전파 행렬을 출력하시오

답)

```
import numpy as np

x = np.array([[1.0, -0.5], [-2.0, 3.0]])
relu_test = Relu()
a = relu_test.forward(x)
print(a)
```

```
[[1. 0.]
 [0. 3.]]
```

문제 142) Relu 클래스를 객체화 시켜 아래의 입력 데이터 x값을 받아 역전파 행렬을 출력하시오

답)

```
import numpy as np

x = np.array([[1.0, -0.5], [-2.0, 3.0]])
relu_test = Relu()
a = relu_test.forward(x)
b = relu_test.backward(a)

print(a)
```

```
print(b)
```

```
[[1. 0.]
 [0. 3.]]
[[1. 0.]
 [0. 3.]]
```

2. Sigmoid 계층

문제 143) 책 170페이지의 sigmoid 클래스를 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 * (1.0 - self.out) * self.out

        return dx
```

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)

오차(비용) 함수를 미분하기 위해 수치미분 함수 사용
3층 신경망 구현을 위한 클래스 생성
학습목표 : "오차함수를 미분해서 기울기를 가지고 가중치와 바이어스를 갱신하여 최적의 가중치와 바이어스를 지닌 신경망을 만드는게 목표"

5장. 3층 신경망 학습 (오차역전파를 이용해서 학습시킴)
계산 그래프를 왜 사용하는지?
덧셈, 곱셈 계산 그래프
렐루 함수 계산그래프 - 파이썬으로 구현
시그모이드 함수 계산그래프 - 파이썬으로 구현
Affine 계층 구현 (순전파, 역전파 코드)
소프트맥스, 교차엔트로피 계산그래프

6장. 3층 신경망의 정확도를 올리기 위한 여러가지 방법들

7장. CNN으로 신경망 구현

수치 미분의 장단점

장점: 구현이 쉽다
단점: 학습이 너무 느리다

그래서 오차역전파를 이용한 학습 방법을 쓴다

Affine/Softmax 계층 구현하기

Affine 계층 (p.170)

신경망의 순전파 때 수행하는 행렬의 내적은 기하학에서는 어파인 변환 이라고합니다.
그래서 신경망에서 입력값과 가중치의 내적의 합에 바이어스를 더하는 그 층을 Affine 계층이라고 한다.

지금까지의 계산그래프는 노드 사이에 '스칼라값' 이 흘렀는데 이에 반해 이번에는 '행렬' 이 흐르고 있다

문제 144) 아래의 두 행렬의 내적과 바이어스를 더한 값이 무엇인지 파이썬으로 구현하시오

보기)

```
X = [[1 2]]
W = [[1 3 5]
     [2 4 6]]
b = [[1 2 3]]
```

답)

```
import numpy as np

X = np.array([[1,2]])
W = np.array([[1,3,5], [2,4,6]])
b = np.array([[1,2,3]])
Y = np.dot(X, W) + b
Y
```

array([[6, 13, 20]])

문제 145) 위의 순전파의 역전파된 최종값인 dX와 dW를 각각 구하시오

답)

```
import numpy as np

X = np.array([[1,2]])
W = np.array([[1,3,5], [2,4,6]])
b = np.array([[1,2,3]])
X_W_dot = np.dot(X, W)
print('X dot Y:', X_W_dot)
Y = X_W_dot + b
print('Y:', Y)

dY = np.array([[1,2,3]])
dX = np.dot(dY, W.T)
print('dX:', dX)
dW = np.dot(X.T, dY)
print('dW:', dW)
```

X dot Y: [[5 11 17]]
Y: [[6 13 20]]
dX: [[22 28]]
dW: [[1 2 3]
 [2 4 6]]

문제 146) 아래의 행렬을 입력받아 결과를 출력하는 forward 함수를 생성하시오

보기)

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

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

print(forward(X, W, b))
```

답)

```
def forward(X, W, b):
    out = np.dot(X, W) + b
    return out
```

[[6 12 18]]

문제 147) 지금 위의 코드는 MNIST 로 치면 딱 한장의 이미지만 입력되는 코드이다. 아래와 같이 batch 로 처리할 수 있도록 2장의 이미지로 처리되는 코드로 forward를 구현하시오

보기)

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

print(forward(X, W, b))
```

답)

```
def forward(X, W, b):
    out = np.dot(X, W) + b
    return out
```

문제 148) 문제 147번에 대한 역전파 함수를 backward 라는 이름으로 생성하시오

보기)

```
X = np.array([[1,2], [3,4]])        # (2, 2)
W = np.array([[1,3,5], [2,4,6]])   # (2, 3)
b = np.array([[1,1,1]])             # (1, 3)
dY = np.array([[1,1,1], [2,2,2]])  # (2, 3)
```

답)

```
import numpy as np

# 순전파
def forward(X, W, b):
```

```
    out = np.dot(X, W) + b
    return out

# 역전파
def backward(X, W, dY):
    dX = np.dot(dY, W.T)
    print('dX(입력값에 대한 역전파):₩n', dX)
    dW = np.dot(X.T, dY)
    print('dW(가중치에 대한 역전파):₩n', dW)


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

backward(X, W, dY)
```

```
dX(입력값에 대한 역전파):
 [[ 9 12]
 [18 24]]
dW(가중치에 대한 역전파):
 [[ 7  7  7]
 [10 10 10]]
```

문제 149) 아래의 행렬의 열을 더한 결과를 아래와 같이 출력하시오

보기)

```
b = np.array([[1,1,1], [2,2,2]])              # (2, 3)
```

답)

```
b = np.array([[1,1,1], [2,2,2]])              # (2, 3)

db = np.sum(b, axis=0)
print(db)
```

```
[3 3 3]
```

문제 150) 아래의 backward 함수에 바이어스에 대한 역전파를 출력하는 코드를 추가하시오

보기)

```
# 역전파
def backward(X, W, dY):
    dX = np.dot(dY, W.T)
    print('dX(입력값에 대한 역전파):₩n', dX)
    dW = np.dot(X.T, dY)
    print('dW(가중치에 대한 역전파):₩n', dW)
```

답)

```
b = np.array([[1,1,1], [2,2,2]])            # (2, 3)

db = np.sum(b, axis=0)
print(db)
```

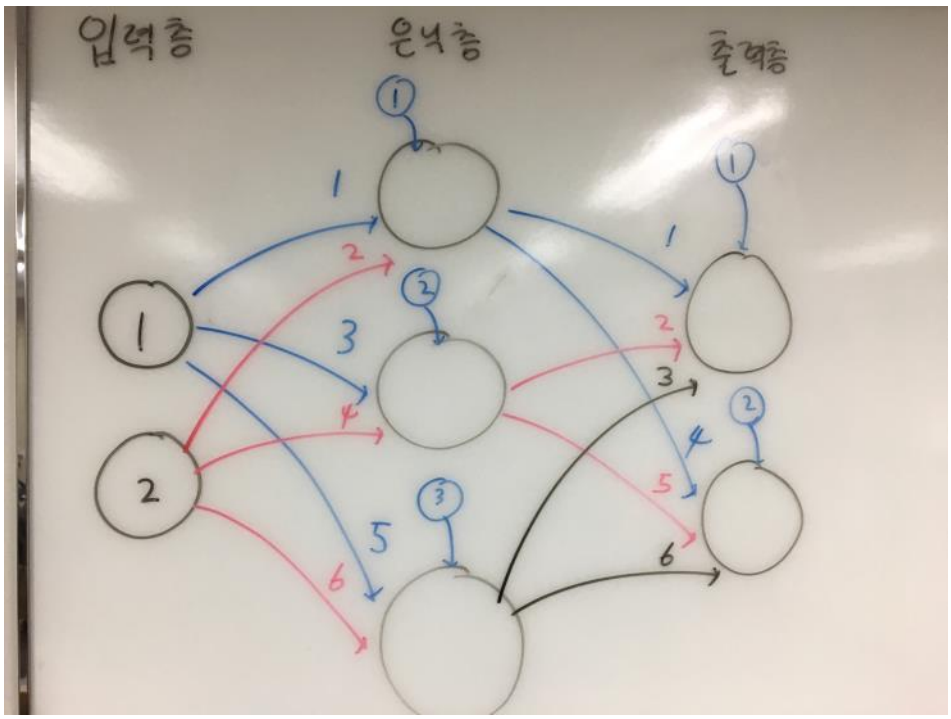
문제 151) 책 175 페이지 아래에 나오는 Affine 클래스를 구현하시오

답)

```
class Affine:
    def __init__(self, W, b):
        self.W = W
        self.b = b
        self.x = None
        self.dW = None
        self.db = None

    # 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


affine1 = Affine(W, b)  # 가중치와 바이어스로 신경망 객체 affine1을 생성한 것
```

```
print(affine1.forward(x))
print(affine1.backward(dY))
```

```
[[ 6 12 18]
 [12 26 40]]
[[ 9 12]
 [18 24]]
```



문제 152) 위의 그림의 신경망의 x, W1, W2, b1, b2의 numpy 배열을 만드시오

답)

```
x = np.array([[1,2]])                  # (1, 2)
W1 = np.array([[1,3,5], [2,4,6]])      # (2, 3)
W2 = np.array([[1,4]. [2,5], [3,6]])   # (3, 2)
b1 = np.array([[1,2,3]])               # (1, 3)
b2 = np.array([[1,2]])                 # (1, 2)
```

문제 153) Affine 클래스를 이용해서 2층 신경망의 순전파를 구현하시오

답)

```
layer_1 = Affine(W1, b1)
layer_2 = Affine(W2, b2)
out1 = layer_1.forward(x)
out2 = layer_2.forward(out1)
print(out2)
```

[[93 211]]

문제 154) 위의 2층 신경망의 역전파를 Affine 클래스를 이용해서 구현하시오
dY는 위의 순전파의 결과인 out2 행렬을 그대로 사용하시오

답)

```
import numpy as np

class Affine:
    def __init__(self, W, b):
        self.W = W
        self.b = b
        self.x = 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)
        dW = np.dot(self.x.T, dY)
        db = np.sum(dY, axis=0)

        return dx, dW, db


x = np.array([[1,2]])                  # (1, 2)
W1 = np.array([[1,3,5], [2,4,6]])      # (2, 3)
W2 = np.array([[1,4], [2,5], [3,6]])   # (3, 2)
b1 = np.array([[1,2,3]])               # (1, 3)
b2 = np.array([[1,2]])                 # (1, 2)

layer_1 = Affine(W1, b1)
layer_2 = Affine(W2, b2)
out1 = layer_1.forward(x)
out2 = layer_2.forward(out1)

dx2, dW2, db2 = layer_2.backward(out2)
```

```
dx1, dW1, db1 = layer_1.backward(dx2)

print(dx1)
print(dW1)
print(db1)
```

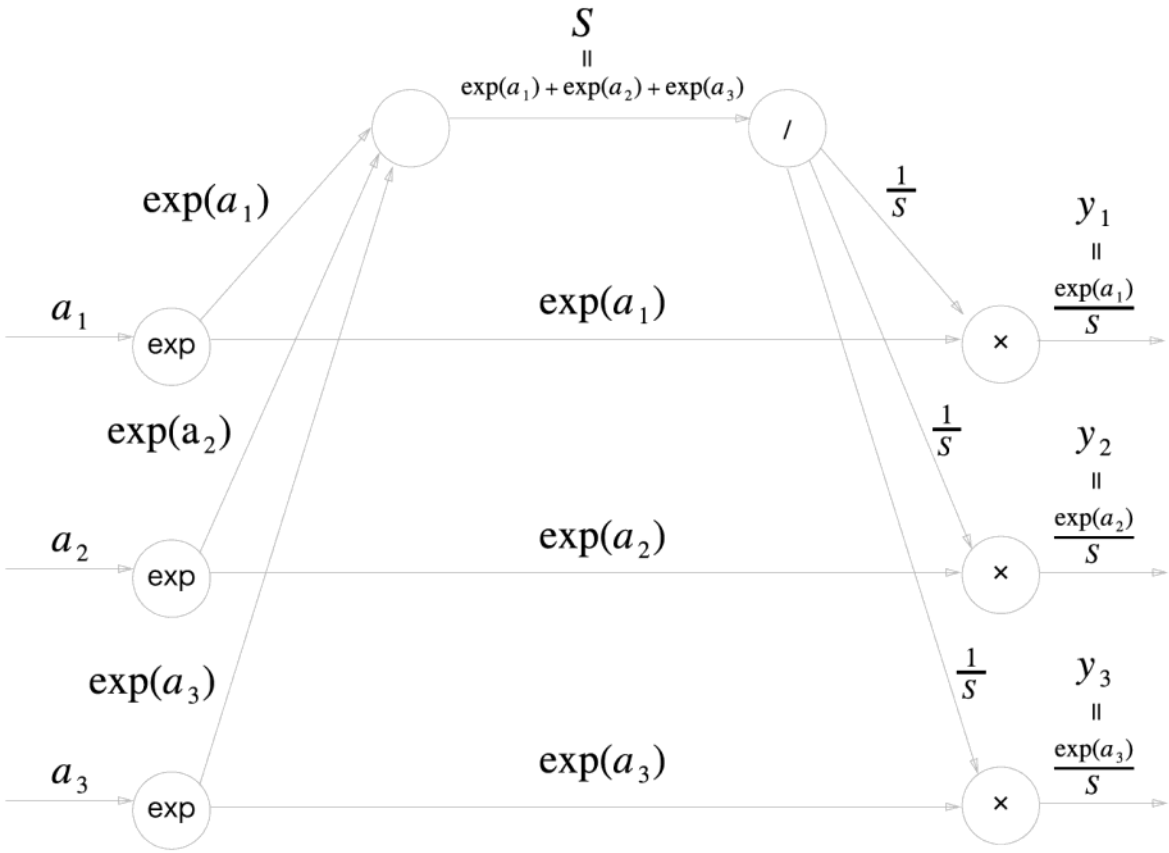
문제 155) 위의 2층 신경망의 순전파를 은닉층에 활성화함수로 ReLU 를 추가해서 구현하시오
(어제 만들었던 Relu 클래스를 가져와서 구현하시오)

답)

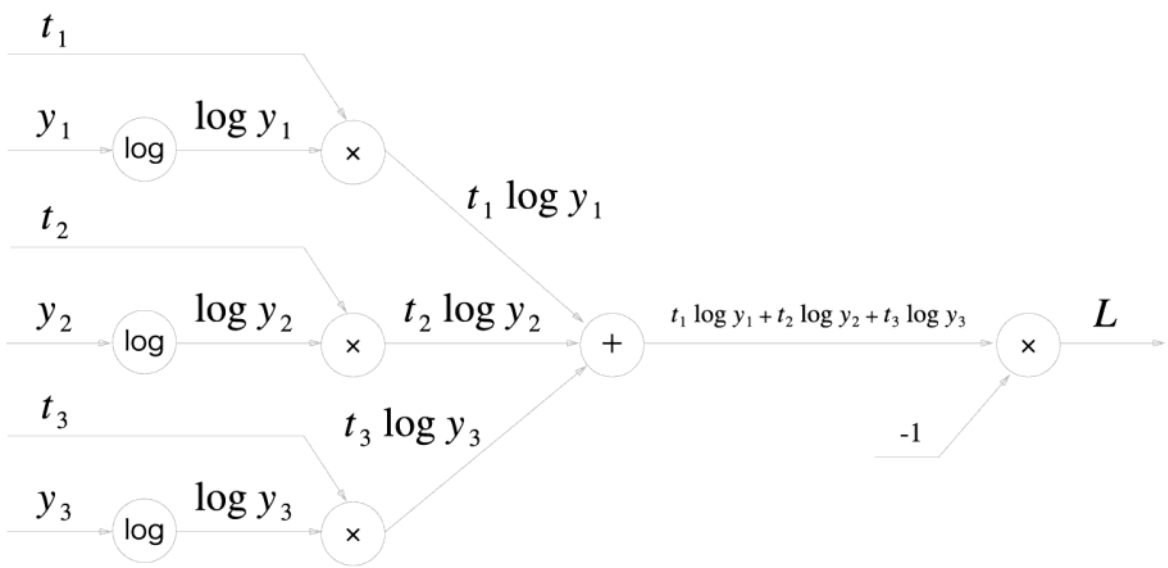
```
x = np.array([[1,2]])                  # (1, 2)
W1 = np.array([[1,3,5], [2,4,6]])      # (2, 3)
W2 = np.array([[1,4], [2,5], [3,6]])   # (3, 2)
b1 = np.array([[1,2,3]])               # (1, 3)
b2 = np.array([[1,2]])                 # (1, 2)

layer_1 = Affine(W1, b1)
relu_1 = Relu()
layer_2 = Affine(W2, b2)
out1 = layer_1.forward(x)
r_out1 = relu_1.forward(out1)
out2 = layer_2.forward(x1)
r_out2 = relu_1.forward(out2)

print(y)
```

소프트맥스 함수 계산그래프

Softmax 계층의 계산 그래프 (순전파만)



Cross Entropy Error 계층의 계산 그래프 (순전파만)



문제 156) 책 179페이지에 나오는 class SoftmaxWithLoss 함수를 구현하시오

답)

```
def softmax(a):
    c = np.max(a)
    minus = a - c
    np_exp = np.exp(minus)
    return np_exp
```

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

class SoftmaxWithLoss:
    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 = cross_entropy_error(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
```

문제 157) 위에서 만든 SoftmaxWithLoss 클래스를 객체화 시켜서 아래의 x(입력값), t(target value)를 입력해서 순전파의 오차율을 출력하고 역전파도 출력하시오

보기)

```
t = np.array([0,0,1,0,0,0,0,0,0,0]) # 숫자 2
x1 = 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, 0.05, 0.3, 0.1, 0.02])
```

답)

```
soft_with_loss1 = SoftmaxWithLoss()
print(soft_with_loss1.forward(x2, t))
print(soft_with_loss1.backward())
```

OrderDict() 함수의 이해

OrderDict은 그냥 dictionary와는 다르게 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)
```

```
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:
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 되어야 하기 때문에 orderedDict 함수를 사용해야 한다

순전파 :
입력값 --> Affine1 계층 --> 시그모이드 --> Affine2 계층 --> 소프트맥스

역전파:
소프트맥스 --> Affine2 계층 --> 시그모이드 --> Affine1 계층
```

오차역전파를 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. 오차역전파를 진행하는 함수 (gradient)
```

문제 158) 책 181~183페이지에 나오는 클래스 생성하시오 (점심시간 문제)

아래 코드를 위한 클래스들

```
# 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

# Relu 클래스
class Relu:
    def __init__(self):
        self.mask = None

    def forward(self, x):
        self.mask = (x <= 0)  # x 에서 0보다 작은 값들에 True 들어감
        out = x.copy()
        out[self.mask] = 0  # out 에서 0보다 작은 값들 0 으로 바꿔줌

        return out

    def backward(self, dout):
        dout[self.mask] = 0
        dx = dout

        return dx

# softmax 함수
def softmax(a):
    c = np.max(a)
    minus = a - c
    np_exp = np.exp(minus)
    return np_exp

# CEE 함수
def cross_entropy_error(y, t):
    delta = 1e-7
```

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

# SoftmaxWithLoss 클래스
class SoftmaxWithLoss:
    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 = cross_entropy_error(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
```

답)

```
import sys, os
sys.path.append(os.pardir)
import numpy as np
from common.layers import *
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)

        # 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):
        # 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['Affine2'].dW
        grads['b2'] = self.layers['Affine2'].db

        return grads

twolayernet1 = 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(twolayernet1.predict(x))
print(twolayernet1.gradient(x, t))
```

문제 159, 160) 3층 신경망 클래스를 구현하시오

답)

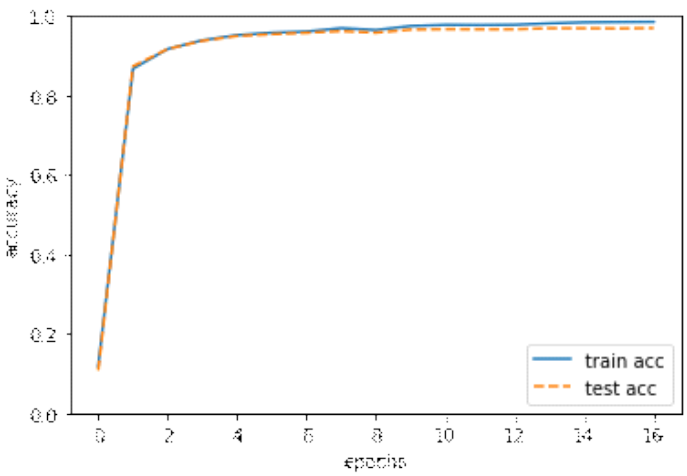
```
import sys, os
sys.path.append(os.pardir)
import numpy as np
from dataset.mnist import load_mnist
from common.layers import *
from common.gradient import numerical_gradient
from collections import OrderedDict

class TwoLayersNet:
    def __init__(self, input_size, hidden_size_1, hidden_size_2, output_size, weight_init_std = 0.02):
        self.params = {}
        self.params['W1'] = weight_init_std*np.random.randn(input_size, hidden_size_1)
        self.params['b1'] = np.zeros(hidden_size_1)
        self.params['W2'] = weight_init_std*np.random.randn(hidden_size_1, hidden_size_2)
        self.params['b2'] = np.zeros(hidden_size_2)
        self.params['W3'] = weight_init_std*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['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

    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

    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 = {}
        grads['W1'] = self.layers['Affine1'].dW
        grads['b1'] = self.layers['Affine1'].db
        grads['W2'] = self.layers['Affine2'].dW
        grads['b2'] = self.layers['Affine2'].db
        grads['W3'] = self.layers['Affine3'].dW
        grads['b3'] = self.layers['Affine3'].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)

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]

    grad = network.gradient(x_batch, t_batch)

    for key in ('W1', 'b1', 'W2', 'b2', 'W3', 'b3'):
        network.params[key] -= learning_rate * grad[key]

    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)

# 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()
```

```
0.11805 0.1077
0.8671166666666666 0.8732
0.9168333333333333 0.9168
0.9387 0.9373
0.9517 0.9495
0.9577666666666667 0.954
0.96065 0.9575
0.9692833333333334 0.9618
0.9645333333333334 0.9577
0.9747 0.9656
0.9776333333333334 0.9667
0.9771666666666666 0.9663
0.9777833333333333 0.9664
0.9814 0.9691
0.9838666666666667 0.969
0.9846333333333334 0.9685
0.9849833333333333 0.9695
```



문제 161) 위의 3층 신경망의 정확도가 활성화 함수가 Relu 일 때와 활성화 함수가 Sigmoid 일 때의 차이를 테스트 하시오

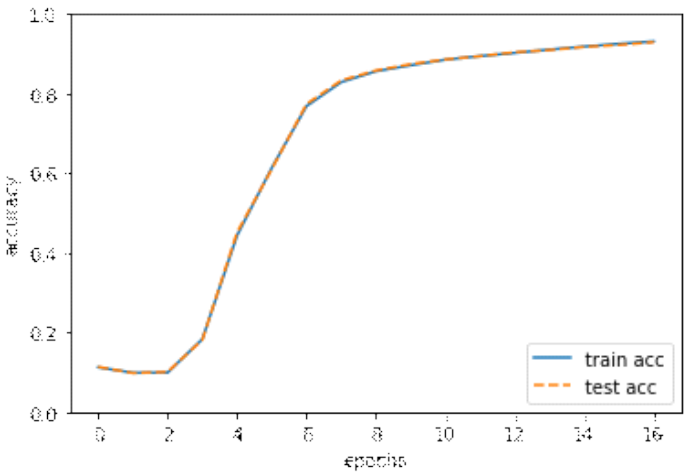
답)

```
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

class TwoLayersNet:
    def __init__(self, input_size, hidden_size_1, hidden_size_2, output_size, weight_init_std = 0.02):
        self.params = {}
        self.params['W1'] = weight_init_std*np.random.randn(input_size, hidden_size_1)
        self.params['b1'] = np.zeros(hidden_size_1)
        self.params['W2'] = weight_init_std*np.random.randn(hidden_size_1, hidden_size_2)
        self.params['b2'] = np.zeros(hidden_size_2)
        self.params['W3'] = weight_init_std*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['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()

    def predict(self, x):
        for layer in self.layers.values():
            x = layer.forward(x)

        return x

    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

    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 = {}
        grads['W1'] = self.layers['Affine1'].dW
        grads['b1'] = self.layers['Affine1'].db
        grads['W2'] = self.layers['Affine2'].dW
        grads['b2'] = self.layers['Affine2'].db
        grads['W3'] = self.layers['Affine3'].dW
        grads['b3'] = self.layers['Affine3'].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)

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]

    grad = network.gradient(x_batch, t_batch)

    for key in ('W1', 'b1', 'W2', 'b2', 'W3', 'b3'):
        network.params[key] -= learning_rate * grad[key]

    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)

# 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()
```

0.11236666666666667 0.1135
0.09871666666666666 0.098
0.09915 0.1009
0.1833 0.1815
0.44361666666666666 0.4515
0.6136666666666667 0.6104
0.7685833333333333 0.7739
0.8288166666666666 0.8331
0.8560666666666666 0.8581
0.8710666666666667 0.8739
0.8851666666666667 0.8858
0.8942 0.894
0.9021333333333333 0.9039
0.9106166666666666 0.9093
0.91815 0.9174
0.9251666666666667 0.9222
0.9310333333333334 0.9292



문제 162) 다시 Relu로 변환하고 노드 수를 늘리면 정확도가 올라가는지 확인하시오

보기)

기존: 입력층(784) → 은닉1층(50) → 은닉2층(100) → 출력층(10)

변경: 입력층(784) → 은닉1층(100) → 은닉2층(100) → 출력층(10)

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

class TwoLayersNet:
    def __init__(self, input_size, hidden_size_1, hidden_size_2, output_size, weight_init_std = 0.02):
        self.params = {}
        self.params['W1'] = weight_init_std*np.random.randn(input_size, hidden_size_1)
        self.params['b1'] = np.zeros(hidden_size_1)
        self.params['W2'] = weight_init_std*np.random.randn(hidden_size_1, hidden_size_2)
        self.params['b2'] = np.zeros(hidden_size_2)
        self.params['W3'] = weight_init_std*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['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

    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

    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 = {}
        grads['W1'] = self.layers['Affine1'].dW
        grads['b1'] = self.layers['Affine1'].db
        grads['W2'] = self.layers['Affine2'].dW
        grads['b2'] = self.layers['Affine2'].db
        grads['W3'] = self.layers['Affine3'].dW
        grads['b3'] = self.layers['Affine3'].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 = 100, 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)

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]

    grad = network.gradient(x_batch, t_batch)

    for key in ('W1', 'b1', 'W2', 'b2', 'W3', 'b3'):
        network.params[key] -= learning_rate * grad[key]

    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)

# 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()
```

```
0.15006666666666665 0.1521
0.8915 0.8897
0.9240666666666667 0.9256
0.9387 0.9411
0.9599166666666666 0.9578
0.96635 0.9597
0.9705 0.9608
0.9756833333333333 0.9678
0.978 0.9693
0.9803 0.9694
```

0.9811166666666666 0.968

0.9854833333333334 0.973

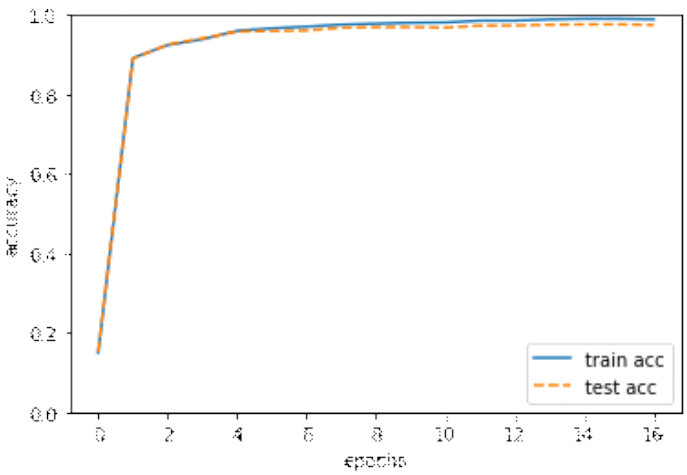
0.9855 0.9733

0.9888166666666667 0.9754

0.99035 0.9763

0.9903 0.9764

0.9889166666666667 0.9747



---

08 학습 관련 기술들

2018년 8월 13일 월요일 오후 4:34

Table of Contents

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)

오차(비용) 함수를 미분하기 위해 수치미분 함수 사용

3층 신경망 구현을 위한 클래스 생성

학습목표 : "오차함수를 미분해서 기울기를 가지고 가중치와 바이어스를 갱신하여 최적의 가중치와 바이어스를 지닌 신경망을 만드는게 목표"

5장. 3층 신경망 학습 (오차역전파를 이용해서 학습시킴)

계산 그래프를 왜 사용하는지?
덧셈, 곱셈 계산 그래프
렐루 함수 계산그래프 - 파이썬으로 구현
시그모이드 함수 계산그래프 - 파이썬으로 구현
Affine 계층 구현 (순전파, 역전파 코드)
소프트맥스, 교차엔트로피 계산그래프

6장. 3층 신경망의 정확도를 올리기 위한 여러가지 방법들

- 언더피팅을 막기 위한 방법
    1. 고급 경사 감소법 (SGD, Momentum, Adagrade, Adam)
    2. 가중치 초기화 설정
    3. 배치 정규화

- 오버피팅을 막기 위한 방법들
    1. 드롭아웃
    2. 가중치 감소(decay)

7장. CNN으로 신경망 구현

언더피팅을 막기 위한 방법들

고급 경사 감소법의 종류

1. SGD
2. Momentum
3. Adagrade
4. Adam

1. SGD (Stochastic Gradient Descent)

Gradient Descent 의 단점

1. 계산량이 많아서 속도가 느리다
2. local minima 에 빠질 수 있다

GD 는 가장 기본적인 Neural Net 의 학습 방법으로 전체 데이터에 대한 비용 함수를 Weight 과 미분하여 각 W(가중치) 파라미터에 대한 기울기를 이용하여 W(가중치) 를 업데이트 하는 방법이다

그런데 이 GD는 층이 많아질 수록 파라미터들의 복잡도가 늘어남에 따라 비용 함수가 더욱 복잡해져서 local minima 에 빠지는 현상이 더욱 잘 발생하게 된다

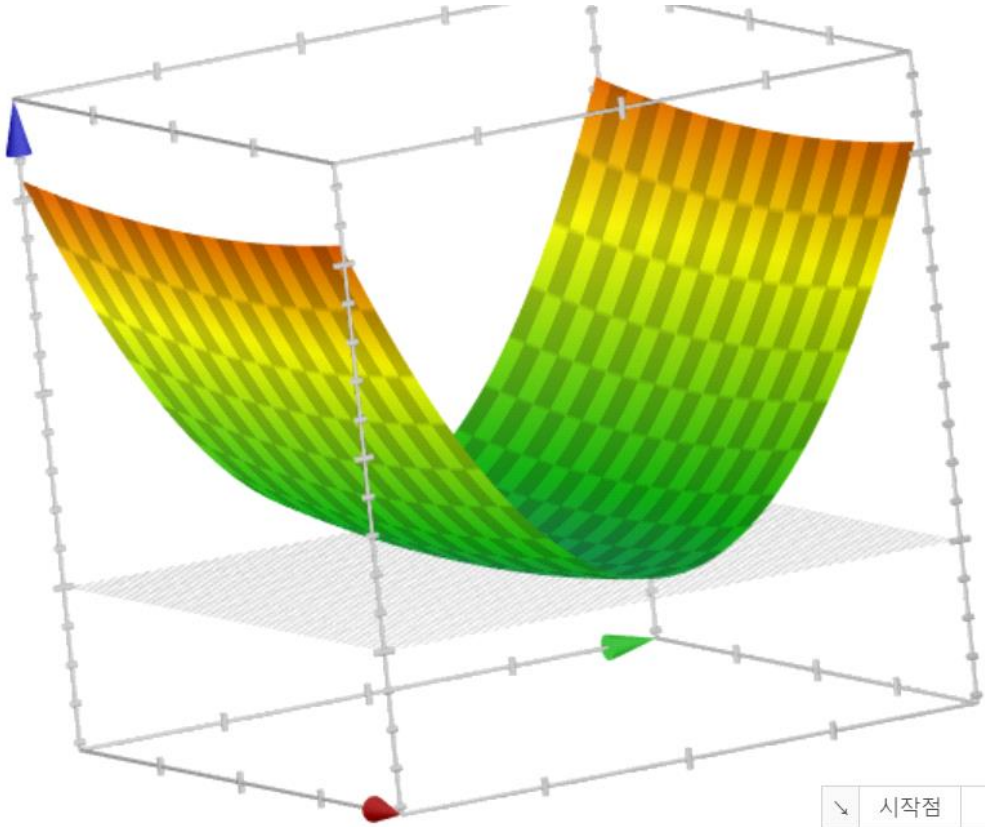
그래서 해결하고자 나온 방법이

SGD (Stochastic Gradient Descent)

확률적

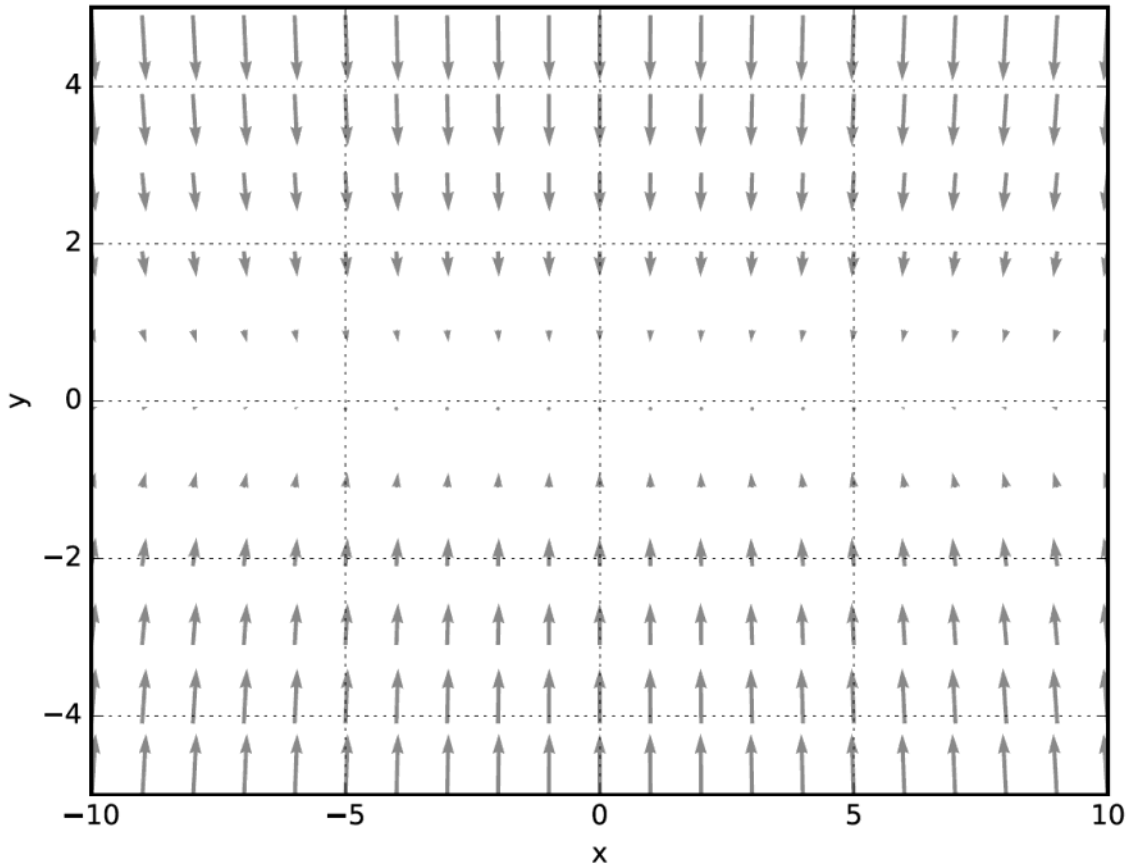
전체 데이터를 가지고 계산을 하면 계산량이 너무 커서 속도가 느려지므로 전체 데이터 중 랜덤하게 복원추출하여 더 빠르게 가중치를 최적화 하는 방법이다

구글에 z = x^2/20 + y^2 라고 검색하면 아래와 같은 결과가 나온다



↘	시작점	끝점
x	-10.0000	10.0000
y	-10.0000	10.0000
z	-41.9939	125.988

위 그림을 위에서 보면 아래와 같은 형태가 된다



위의 그림처럼 Global Minima 쪽으로 기울기 방향이 있는게 아니기 때문에 Local Minima에 빠지고 Global Minima 쪽으로 가지 못하는 단점이 있다.

2. Momentum (운동량)

기존 SGD : 가중치 ← 가중치 - 러닝레이트 * 기울기

Momentum : 속도 ← 0.9 (마찰계수) * 속도 - 러닝레이트 * 기울기
가중치 ← 가중치 + 속도
내리막 : 기울기 음수이면 속도가 증가
오르막 : 기울기 양수이면 속도가 감소

기울어진 방향으로 물체가 가속되는 관성의 원리를 적용

*관성

정지해 있는 물체는 계속 정지해 있으려 하고 운동하는 물체는 계속 운동하려는 성질
(예: 100미터 달리기 결승점에서 갑자기 멈춰지지 않는 현상)

3. Adagrade 경사 감소법

러닝레이트(학습률)이 학습이 되면서 자체적으로 조정이 되는 경사 감소법

처음에는 크게 학습하다가 조금씩 작게 학습을 진행한다

Learning Rate ↑	발산될 위험은 있지만 수렴속도가 빨라진다
Learning Rate ↓	발산될 위험은 없지만 Local Minima 에 빠지거나 학습이 안될 수 있다

$$\text{SGD}: \mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\partial L}{\partial \mathbf{W}}$$

$$\text{Adagrad}: \mathbf{W} \leftarrow \mathbf{W} - \eta \frac{1}{\sqrt{\mathbf{h}}} \frac{\partial L}{\partial \mathbf{W}}$$

$$\mathbf{h} \leftarrow \mathbf{h} + \frac{\partial L}{\partial \mathbf{W}} \odot \frac{\partial L}{\partial \mathbf{W}}$$

$(\odot$: 행렬의 원소별 곱셈을 의미$)$

1. 처음에는 학습률이 크다가 조금씩 감소

2. ◎ 는 행렬의 원소 별 곱셈을 의미하는데
   h의 원소가 각각의 매개변수 원소 변화량에 의해 결정이 된다
   ↓
  "각각의 매개변수 원소가 갱신되는 값이 다르다"
   ↓
  "많이 움직인 원소일수록 누적 h의 값이 크니까 갱신정도가 그만큼 감소한다"

4. Adam 경사 감소법

Momentum 의 장점 + Adagrade 의 장점을 살린 경사 하강법
  (가속도, 관성) (각 매개변수마다 학습률 조정)

" 최근 딥러닝에서 가장 많이 사용하는 최적화 방법 "

문제 163) SGD 클래스를 만들고 SGD 클래스를 이용해서 3층 신경망의 코드를 변경하시오

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

class TwoLayersNet:
    def __init__(self, input_size, hidden_size_1, hidden_size_2, output_size, weight_init_std = 0.02):
        self.params = {}
        self.params['W1'] = weight_init_std*np.random.randn(input_size, hidden_size_1)
        self.params['b1'] = np.zeros(hidden_size_1)
        self.params['W2'] = weight_init_std*np.random.randn(hidden_size_1, hidden_size_2)
        self.params['b2'] = np.zeros(hidden_size_2)
        self.params['W3'] = weight_init_std*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['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

    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

    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 = {}
        grads['W1'] = self.layers['Affine1'].dW
        grads['b1'] = self.layers['Affine1'].db
        grads['W2'] = self.layers['Affine2'].dW
        grads['b2'] = self.layers['Affine2'].db
        grads['W3'] = self.layers['Affine3'].dW
        grads['b3'] = self.layers['Affine3'].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 = 50, 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)


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.

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)
```

```
0.103 0.1029
0.7969333333333334 0.8038
0.8774 0.8778
0.9073 0.9038
0.9244666666666667 0.9226
0.9385833333333333 0.9351
0.9404333333333333 0.9375
0.9527833333333333 0.9488
0.9546333333333333 0.9481
0.96205 0.956
0.9646 0.9578
```

0.96695 0.9598
0.9693333333333334 0.9629
0.96845 0.9589
0.9749 0.9626
0.9756833333333333 0.964
0.9780333333333333 0.9661

문제 164) Momentum 클래스를 파이썬 코드로 구현하고 방금 만든 3층 신경망에 적용해서 정확도를 확인하시오

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

class TwoLayersNet:
    def __init__(self, input_size, hidden_size_1, hidden_size_2, output_size, weight_init_std = 0.02):
        self.params = {}
        self.params['W1'] = weight_init_std*np.random.randn(input_size, hidden_size_1)
        self.params['b1'] = np.zeros(hidden_size_1)
        self.params['W2'] = weight_init_std*np.random.randn(hidden_size_1, hidden_size_2)
        self.params['b2'] = np.zeros(hidden_size_2)
        self.params['W3'] = weight_init_std*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['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
```

```
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

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 = {}
    grads['W1'] = self.layers['Affine1'].dW
    grads['b1'] = self.layers['Affine1'].db
    grads['W2'] = self.layers['Affine2'].dW
    grads['b2'] = self.layers['Affine2'].db
    grads['W3'] = self.layers['Affine3'].dW
    grads['b3'] = self.layers['Affine3'].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 = 50, 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 ← 0.9(마찰계수) * 속도 - 러닝레이트 * 기울기
        # 가중치 ← 가중치 + 속도

optimizer = Momentum()  # Momentum 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)
```

```
0.103 0.1146
0.9433333333333334 0.9409
0.97005 0.9631
0.97405 0.9662
0.9783833333333334 0.9702
0.9785833333333334 0.9659
0.97965 0.9709
0.9833833333333334 0.9697
0.9813333333333333 0.968
0.9892 0.9748
0.9869 0.9723
0.9890166666666667 0.9757
0.9886166666666667 0.9734
```

0.98725 0.9693
0.9928166666666667 0.9762
0.99405 0.9763
0.9942333333333333 0.9728

문제 165) Adagrade 경사 하강법으로 위의 3층 신경망을 학습시킨 결과 정확도를 출력하시오

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

class TwoLayersNet:
    def __init__(self, input_size, hidden_size_1, hidden_size_2, output_size, weight_init_std = 0.02):
        self.params = {}
        self.params['W1'] = weight_init_std*np.random.randn(input_size, hidden_size_1)
        self.params['b1'] = np.zeros(hidden_size_1)
        self.params['W2'] = weight_init_std*np.random.randn(hidden_size_1, hidden_size_2)
        self.params['b2'] = np.zeros(hidden_size_2)
        self.params['W3'] = weight_init_std*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['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

    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

    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 = {}
        grads['W1'] = self.layers['Affine1'].dW
        grads['b1'] = self.layers['Affine1'].db
        grads['W2'] = self.layers['Affine2'].dW
        grads['b2'] = self.layers['Affine2'].db
        grads['W3'] = self.layers['Affine3'].dW
        grads['b3'] = self.layers['Affine3'].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 ← 0.9(마찰계수) * 속도 - 러닝레이트 * 기울기
        # 가중치 ← 가중치 + 속도

# optimizer = Momentum()  # Momentum 클래스 객체화


###################### Adagrade 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


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)
```

```
0.4133 0.4278
0.9382666666666667 0.9357
0.9472666666666667 0.9457
0.9546666666666667 0.9495
0.9575333333333333 0.9497
0.9622666666666667 0.9573
0.9651666666666666 0.9562
0.9662166666666666 0.958
0.96805 0.9592
0.97085 0.9613
0.9708 0.9603
0.9743 0.964
0.9743666666666667 0.9652
0.9762666666666666 0.9653
0.9769666666666666 0.9651
0.9784666666666667 0.9653
0.97875 0.9667
```

문제 166) Adam클래스를 생성하고 위의 3층 신경망에 Adam경사 감소법을 적용해서 3층 신경망을 학습시키시오

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

class TwoLayersNet:
    def __init__(self, input_size, hidden_size_1, hidden_size_2, output_size, weight_init_std = 0.02):
        self.params = {}
        self.params['W1'] = weight_init_std*np.random.randn(input_size, hidden_size_1)
        self.params['b1'] = np.zeros(hidden_size_1)
        self.params['W2'] = weight_init_std*np.random.randn(hidden_size_1, hidden_size_2)
        self.params['b2'] = np.zeros(hidden_size_2)
        self.params['W3'] = weight_init_std*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['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

    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

    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 = {}
        grads['W1'] = self.layers['Affine1'].dW
        grads['b1'] = self.layers['Affine1'].db
        grads['W2'] = self.layers['Affine2'].dW
        grads['b2'] = self.layers['Affine2'].db
        grads['W3'] = self.layers['Affine3'].dW
        grads['b3'] = self.layers['Affine3'].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 ← 0.9(마찰계수) * 속도 - 러닝레이트 * 기울기
        # 가중치 ← 가중치 + 속도

# optimizer = Momentum()  # Momentum 클래스 객체화


###################### Adagrade 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)
```

가중치의 초깃값

가중치의 초깃값을 적절히 설정하면 각 층의 활성화 값의 분포가 적당히 퍼지는 효과가 발생한다
적당히 퍼지게 되면 학습이 잘되고 정확도가 높아진다

가중치 초깃값을 선정하는 방법

1. 가중치 초깃값을 0으로 선정 → 좋지 않은 방법 (학습 안됨)

2. 표준편차가 1인 정규분포를 사용해 초깃값을 선정
- 표준편차가 작을수록 데이터가 평균에 가깝다
구현 예) 0.01 * np.random.randn(10, 100)

- 표준편차가 클수록 데이터가 더 많이 흩어져 있다.
(시험 문제가 어려우면 아주 잘하는 학생들과 못하는 학생들로 나뉜다.
구현 예) 0.01 * np.random.randn(10, 100)

3. Xavier (사비에르) 초깃값 선정
- 표준편차가 루트 1/n 인 정규분포로 초기화를 한다 (n은 앞층의 노드 수)
- 시그모이드와 짝꿍

4. He 초깃값 선정
- 표준편차가 루트 2/n 인 정규분포로 초기화 (n은 앞층의 노드 수)
- 렐루와 짝꿍

1. 가중치 초깃값을 0으로 선정

문제 167) 지금까지 구현한 3층 신경망의 가중치 초깃값을 0으로 설정하고 신경망을 학습시켜 보시오

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

class TwoLayersNet:
    def __init__(self, input_size, hidden_size_1, hidden_size_2, output_size, weight_init_std = 0):
        self.params = {}
        self.params['W1'] = weight_init_std*np.random.randn(input_size, hidden_size_1)
        self.params['b1'] = np.zeros(hidden_size_1)
        self.params['W2'] = weight_init_std*np.random.randn(hidden_size_1, hidden_size_2)
        self.params['b2'] = np.zeros(hidden_size_2)
        self.params['W3'] = weight_init_std*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['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

    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

    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 = {}
        grads['W1'] = self.layers['Affine1'].dW
        grads['b1'] = self.layers['Affine1'].db
        grads['W2'] = self.layers['Affine2'].dW
        grads['b2'] = self.layers['Affine2'].db
        grads['W3'] = self.layers['Affine3'].dW
        grads['b3'] = self.layers['Affine3'].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 ← 0.9(마찰계수) * 속도 - 러닝레이트 * 기울기
        # 가중치 ← 가중치 + 속도

# optimizer = Momentum()  # Momentum 클래스 객체화


###################### Adagrade 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)
```

3. Xavier (사비에르) 초깃값 선정

문제 168) 가중치의 초깃값을 사비에르로 설정하고 학습이 잘되는지 확인하시오

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

class TwoLayersNet:
    def __init__(self, input_size, hidden_size_1, hidden_size_2, output_size, weight_init_std = 1):
        self.params = {}
        self.params['W1'] = 1/np.sqrt(784)*np.random.randn(input_size, hidden_size_1)
        self.params['b1'] = np.zeros(hidden_size_1)
        self.params['W2'] = 1/np.sqrt(50)*np.random.randn(hidden_size_1, hidden_size_2)
        self.params['b2'] = np.zeros(hidden_size_2)
        self.params['W3'] = 1/np.sqrt(100)*np.random.randn(hidden_size_2, output_size)
        self.params['b3'] = np.zeros(output_size)
```

0.2356 0.2274

0.9622166666666667 0.9584
0.9727333333333333 0.9668
0.9715666666666667 0.9615
0.9788666666666667 0.9666
0.9831666666666666 0.9668
0.98555 0.9701
0.9849166666666667 0.9702
0.9867666666666667 0.9711
0.9808333333333333 0.9645

4. He 초깃값 선정

문제 169) 지금까지 구현한 3층 신경망의 가중치 초깃값을 He로 설정하고 정확도 확인하시오

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

class TwoLayersNet:
    def __init__(self, input_size, hidden_size_1, hidden_size_2, output_size, weight_init_std = 1):
        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)
```

배치 정규화

(p.210)

앞에서는 가중치 초깃값을 적절히 설정하면 각 층의 활성화 값의 분포가 적당히 퍼지는 효과를 보았다.

적당히 퍼지게 되면 학습이 잘되고 정확도가 높아지는 것을 확인했다.

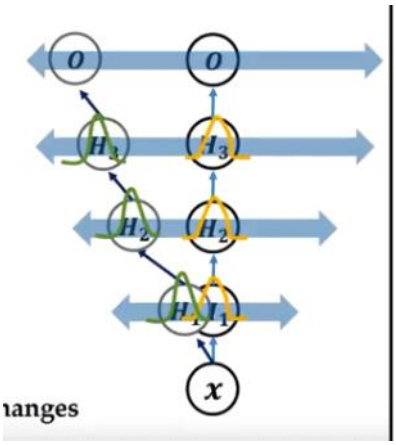
그런데 배치정규화는 바로 각 층에서의 활성화 값이 적당히 분포 되도록 강제로 조정하는 것을 말한다

그림 (6장) 배치 정규화의 필요성

Neural Network 는 파라미터가 많기 때문에 학습이 어렵다.
딥러닝 같은 경우는 히든 레이어가 많아서 학습이 어렵다.

어려운 이유 ?

가중치의 조그마한 벼노하가 가중되어서 쌓이면 히든 레이어가 많아질수록 출력되는 값의 변화가 크기 때문이다



만약에 변화없이 히든 레이어2 까지 안정된 가중치 값을 가지게 된다면 학습이 잘될 것이다.
그런데 가중치의 영향을 계속 받아서 히든레이어2 의 값이 아주 다른 값이 된다면 기존값과는 다르기 때문에 어떻게 학습해야 할지 모르게 되는 상황이 발생하게 된다

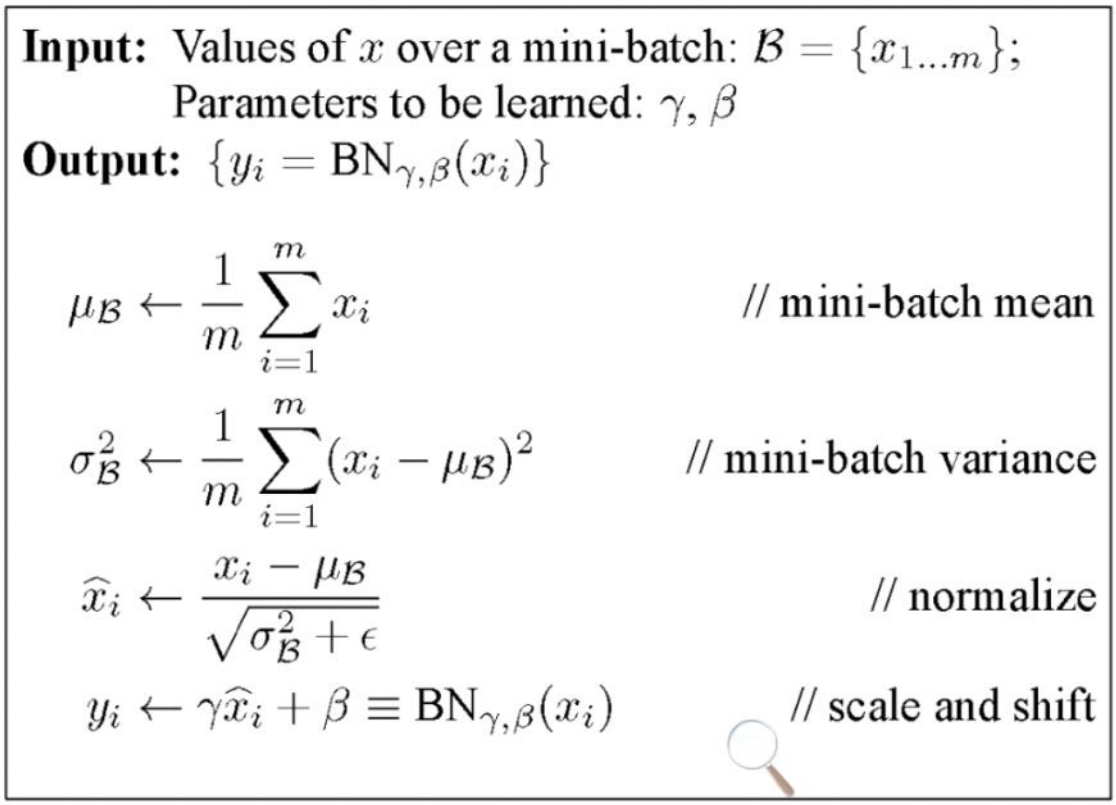
그래서 나온 게 배치정규화이다. 배치정규화는 값이 활성화 함수를 통과하기 전에 가중의 변화를 줄이는 것이 목표이다.
가중의 합이 배치정규화에 들어오게 되면 기존 값의 스케일을 줄여버리게 된다

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()
```

정규화 방법

실제값에서 평균을 뺀 값에 표준편차를 나눈다



배치정규화는 위의 식을 코드화한 것이다

문제 170) 배치정규화 클래스를 우리 3층 신경망에 구현하고 각 층마다 아래와 같이 배치 정규화 층을 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
        dmu = np.sum(dxc, axis=0)
        dx = dxc - dmu / self.batch_size

        self.dgamma = dgamma
        self.dbeta = dbeta

        return dx


class TwoLayersNet:
    def __init__(self, input_size, hidden_size_1, hidden_size_2, output_size, weight_init_std = 1):
        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

    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

    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 = {}
        grads['W1'] = self.layers['Affine1'].dW
        grads['b1'] = self.layers['Affine1'].db
        grads['W2'] = self.layers['Affine2'].dW
        grads['b2'] = self.layers['Affine2'].db
        grads['W3'] = self.layers['Affine3'].dW
        grads['b3'] = self.layers['Affine3'].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 ← 0.9(마찰계수) * 속도 - 러닝레이트 * 기울기
        # 가중치 ← 가중치 + 속도

# optimizer = Momentum()  # Momentum 클래스 객체화


###################### Adagrade 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)
```

```
0.3045333333333333 0.297
0.9697166666666667 0.9611
0.9763833333333334 0.9659
0.9813833333333334 0.9702
0.9861666666666666 0.9695
0.9889833333333333 0.9731
0.9892666666666666 0.9741
0.9911166666666666 0.9747
0.9908833333333333 0.9753
0.994 0.9753
0.99385 0.9751
0.99515 0.9749
0.9944 0.9769
0.99565 0.9772
0.9954666666666667 0.9764
0.9962 0.9758
0.9964 0.9783
```

오버피팅을 억제하기 위한 방법들

```
1. 드롭아웃 (dropout)
2. 가중치 감소 (Weight Decay)
```

드롭아웃 (Dropout)

```
오버피팅을 억제하기 위해서 뉴런을 임의로 삭제하면서 학습시키는 방법

고양이와 개 사진을 구분하는 신경망
```

고양이 사진을 은닉1층을 통과시켜 고양이 점수를 주는 신경망이 있다고 가정하자

귀가 있으면 몇점,
꼬리가 있으면 몇점,
고양이처럼 생긴 사람 같으면 몇점,
집게발을 가지고 있으면 몇점,
:
:
이런 식으로 은닉1층을 통과 시키는데 만약 고양이 꼬리가 안나온 사진이라면 이때 꼬리가 없을 때 낮은 점수는 주는 경우를 없애주어야 한다.

즉, 너무 많은 조건을 주면 오답이 나올 수가 있다는 말이다.

따라서 몇 명의 전문가만 선별하여 반복적으로 같은 결과가 나오면 그것을 답으로 보겠다는 것이다.

노드 제거

따라서 노드를 제거하여 몇 명의 전문가만 선별하는데,

일반적으로, 입력층에서는 20~50% 정도, 은닉층에서는 50% 정도의 노드를 생략한다

Dropout class 구현 코드

```
class Dropout:
    """
    http://arxiv.org/abs/1207.0580
    """
    def __init__(self, dropout_ratio=0.5):
        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
```

* Dropout 클래스의 키가 되는 부분

self.mask = np.random.rand(*x.shape) > self.dropout_raion

100개의 노드가 있을 때 dropout_raion를 0.15를 줬으면 몇 개의 노드가 남을까?

1번. 85개
2번. 15개

코드에서는 훈련할 때 1번, 테스트 할 때는 2번을 사용한다

문제 171) 지금까지 만들었던 3층 신경망 코드에 dropout 을 적용해서 오버피팅이 발생하지 않는지 확인하시오

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
        dmu = np.sum(dxc, axis=0)
        dx = dxc - dmu / 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.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['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

    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

    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 = {}
        grads['W1'] = self.layers['Affine1'].dW
        grads['b1'] = self.layers['Affine1'].db
        grads['W2'] = self.layers['Affine2'].dW
        grads['b2'] = self.layers['Affine2'].db
        grads['W3'] = self.layers['Affine3'].dW
        grads['b3'] = self.layers['Affine3'].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 ← 0.9(마찰계수) * 속도 - 러닝레이트 * 기울기
        # 가중치 ← 가중치 + 속도

# optimizer = Momentum()  # Momentum 클래스 객체화


###################### Adagrade 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)
```

```
0.2898 0.2864
0.9475833333333333 0.9453
0.9602166666666667 0.9543
0.9638166666666667 0.9556
0.96585 0.9549
0.9686333333333333 0.959
0.9723333333333334 0.9618
0.9729166666666667 0.9606
0.9729 0.9625
0.9756 0.9623
0.9773166666666666 0.9626
0.9766166666666667 0.9633
0.9772166666666666 0.9623
0.9781333333333333 0.9635
0.9795833333333334 0.9639
0.9799666666666667 0.9641
0.9793833333333334 0.9637
```

가중치 감소 (Weight Decay)

학습과정에서 큰 가중치에 대해서는 그에 상응하는 큰 패널티를 부여하여 오버피팅을 억제하는 방법

예) 고양이와 개의 사진을 구분하는 신경

```
귀가 있으면                → 가중치
꼬리가 있으면              → 아주 큰 가중치
장난스럽게 보이면          → 가중치
집게발을 가지고 있으면  → 가중치
:
:
```

Weight Decay 라는 것은 위에 입김이 쎈 사공 즉, 꼬리가 있으면 고양이다 라는 노드에 아주 큰 가중치에 패널티를 부여하는 방법이다.

Weight Decay 코드 분석

모든 가중치의 각각의 손실 함수에 1/2 * 람다 * 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
                    ↓
                  오차
```

2. GD 업데이트 과정에서 그동안 오차역전파법에 따른 결과에 정규화 항을 미분한 값에 람다 * 가중치를 더한다

```
코드 예제:

기존 코드 : grad['W1'] = 미분값
변경된 코드 : grad['W1'] = 미분값 + 람다 * 현재의 가중치
```

L2 정규화 적용하는 코드

```
#1. 비용함수를 수정

# L2 정규화 적용 전
# def loss(self, x, t):  # x : (100, 1024), t : (100, 10)
#     y = self.predict(x)  # (100, 10) : 마지막 출력층을 통과한 신경망이 예측한 값
#     return self.lastLayer.forward(y, t) # 마지막 계층인 SoftmaxWithLoss 계층에 대해 forward 수행



# L2 정규화 적용 후
self.weight_decay_lambda = 0.001    <--- __init__ 에 추가해야할 인스턴스 변수

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, 6):
        W = self.params['W' + str(idx)]
        weight_decay += 0.5 * self.weight_decay_lambda * np.sum(W ** 2)
    return self.lastLayer.forward(y, t) + weight_decay


#2. 기울기 계산하는 코드 변경

# L2 정규화 적용 전
# 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
# grads['W4'], grads['b4'] = self.layers['Affine4'].dW, self.layers['Affine4'].db
# grads['W5'], grads['b5'] = self.layers['Affine5'].dW, self.layers['Affine5'].db


# L2 정규화 적용 후
grads['W1'] = self.layers['Affine1'].dW + self.weight_decay_lambda * self.layers['Affine1'].W
grads['b1'] = self.layers['Affine1'].db


# FOR 문으로 한번에 작성
grads = {}
  for idx in range(1, 6):
        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
```

문제 172) 우리 3층 신경망 코드에 L2 정규화를 적용해보시오

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
        dmu = np.sum(dxc, axis=0)
        dx = dxc - dmu / 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 ← 0.9(마찰계수) * 속도 - 러닝레이트 * 기울기
        # 가중치 ← 가중치 + 속도


# optimizer = Momentum()  # Momentum 클래스 객체화


############################ Adagrade 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)
```

0.42473333333333335 0.4376
0.9557166666666667 0.9526
0.9588166666666667 0.9537

---

09 합성곱 신경망 (CNN)

2018년 8월 13일 월요일 오후 4:34

Table of Contents

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)

오차(비용) 함수를 미분하기 위해 수치미분 함수 사용

3층 신경망 구현을 위한 클래스 생성

학습목표 : "오차함수를 미분해서 기울기를 가지고 가중치와 바이어스를 갱신하여 최적의 가중치와 바이어스를 지닌 신경망을 만드는게 목표"

5장. 3층 신경망 학습 (오차역전파를 이용해서 학습시킴)

계산 그래프를 왜 사용하는지?
덧셈, 곱셈 계산 그래프
렐루 함수 계산그래프 - 파이썬으로 구현
시그모이드 함수 계산그래프 - 파이썬으로 구현
Affine 계층 구현 (순전파, 역전파 코드)
소프트맥스, 교차엔트로피 계산그래프

6장. 3층 신경망의 정확도를 올리기 위한 여러가지 방법들

- 언더피팅을 줄이기 위한 방법
    1. 고급 경사 감소법 (SGD, Momentum, Adagrade, Adam)
    2. 가중치 초기화 설정
    3. 배치 정규화

- 오버피팅을 줄이기 위한 방법들
    1. 드롭아웃
    2. 가중치 감소(decay)

7장. CNN으로 신경망 구현

기존층의 문제점
합성곱 계층의 개념과 파이썬으로 구현
    1. im2col 함수 이용 (이미지 → 행렬)
    2. reshape 의 -1 이용 (4차원 행렬(가중치) → 2차원)
풀링 계층의 개념과 파이썬으로 구현

합성곱 신경망 이란?

Convolution 층과 pooling 층을 포함하는 신경망

기존 신경망과의 차이

기존방법	Affine → Relu
CNN	Conv → Relu → pooling

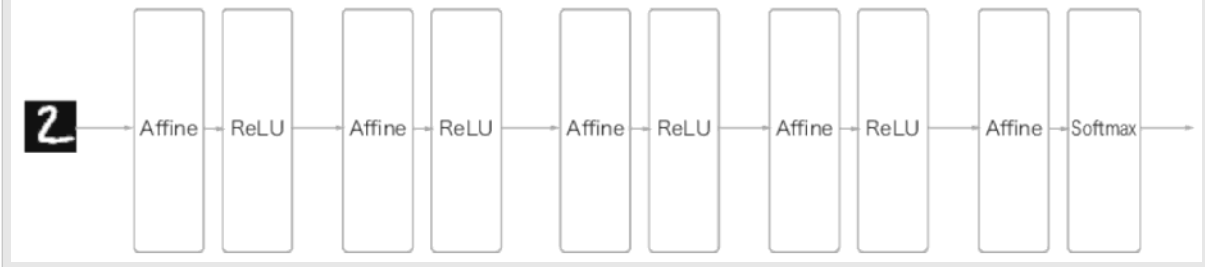
기존에 구현했던 완전 연결 계층의 문제점

데이터의 형상이 무시된다

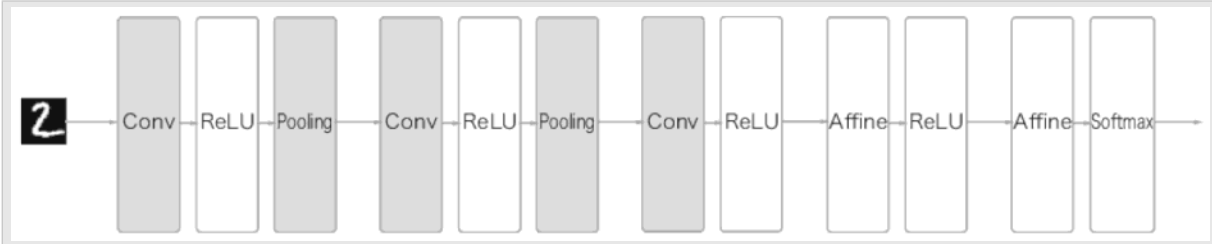
그림으로 설명

http://cafe.daum.net/oracleoracle/SWD1/12

기존 신경망



합성곱 신경망



합성곱 계층 (Convolution Layer)

합성곱 계층의 역할

" feature map 을 만들고 그 feature map 을 선명하게 해주는 층 "

CNN을 이용하지 않은 기존층의 문제점

필기체 데이터를 28 x 28 = 784 의 1차원 데이터로 변경을 해서 784개의 데이터를 첫 Affine 계층에 입력한게 기존 방법이다

따라서, 형상을 무시하고 모든 입력 데이터를 동등한 뉴런으로 취급하기 때문에 이미지가 갖는 본질적인 패턴을 읽지 못한다. (위의 카페에서 그림으로 이해할 것)

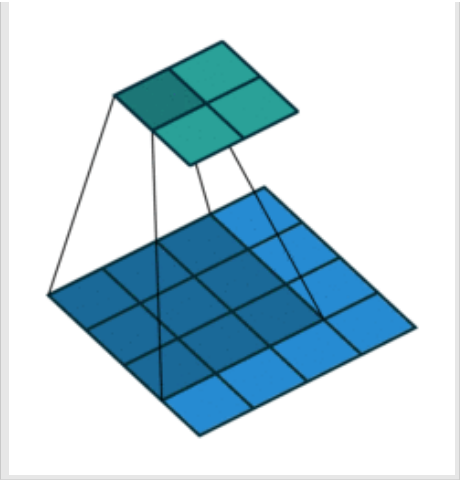
그래서 합성곱 계층이 필요하다

결국 원본 이미지에서 조금만 모양이 달라져도 같은 이미지로 인식하지 못하는 문제를 합성곱이 해결해 줄 수 있다

어떻게 해결하는가?

원본이미지를 가지고 여러 개의 feature map 을 만들어서 분류하는 완전 연결 계층에 입력한다

합성곱 연산

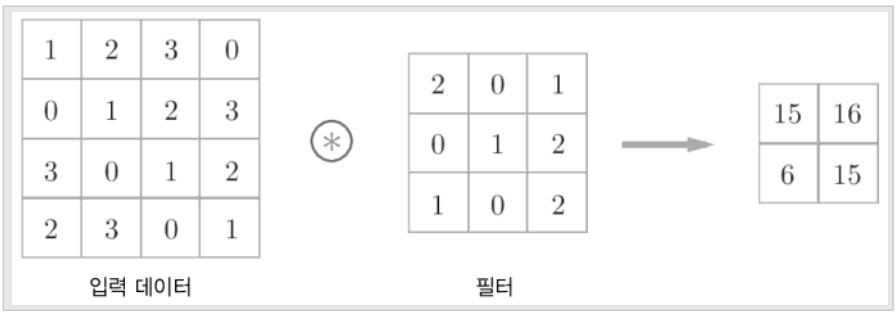


합성곱 연산 이란?

이미지 3차원 (세로, 가로, 색상) data 의 형상을 유지하면서 연산하는 작업

" 입력 데이터에 필터를 적용한 것이 합성곱 연산이다 "

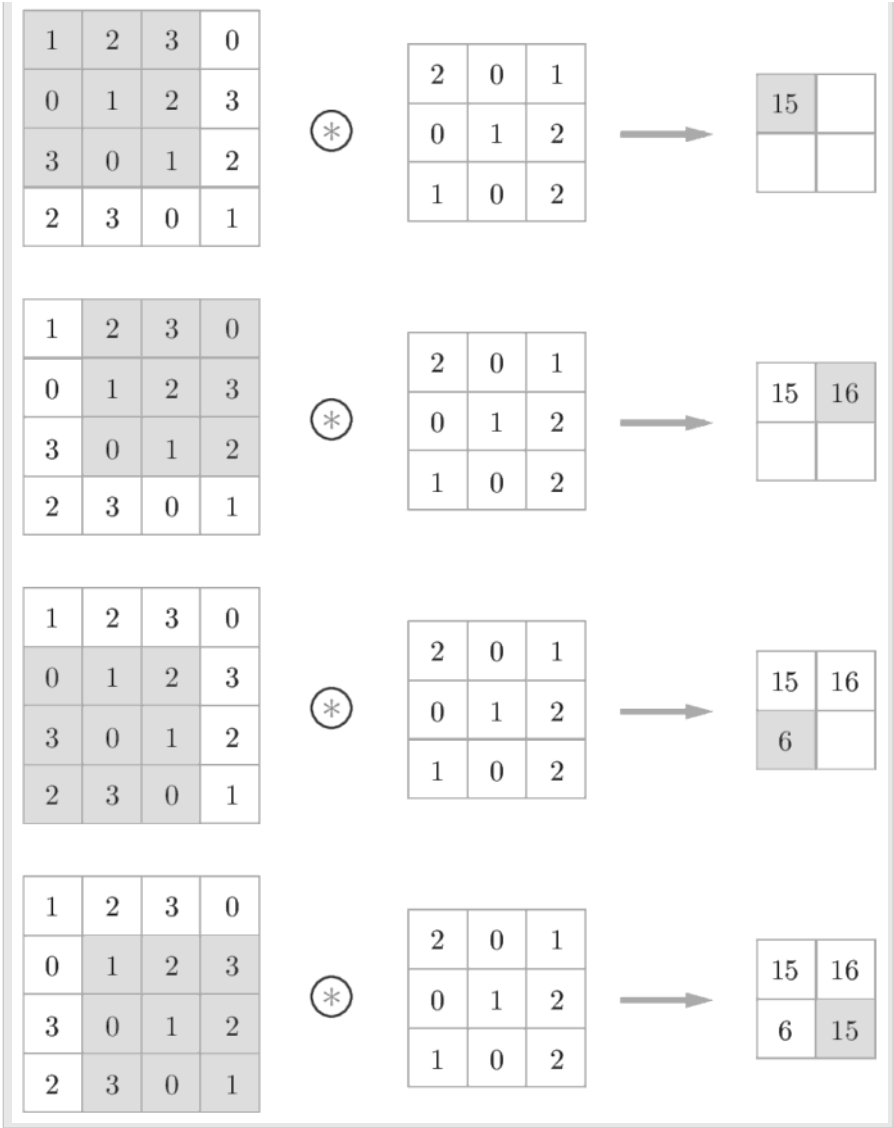
합성곱 연산을 컴퓨터로 구현하는 방법



위의 식 계산 과정

```
1 2 3          2 0 1          2 0 3
0 1 2    ◎     0 1 2    =     0 1 4    =    15
3 0 1          1 0 2          3 0 2
```

(1,1)은 (1,1) 끼리 (1,2)은 (1,2) 끼리 각자 곱한 결과를 모두 더한다 → 그게 (1,1) 이 된다



문제 173) 아래의 두 행렬을 만들고 합성곱한 결과인 15를 파이썬으로 출력하시오

보기)

```
1 2 3          2 0 1          2 0 3
0 1 2    ◎     0 1 2    =     0 1 4    =    15
3 0 1          1 0 2          3 0 2
```

답)

```
import numpy as np

A = np.array([[1,2,3],
              [0,1,2],
              [3,0,1]])

B = np.array([[2,0,1],
              [0,1,2],
              [1,0,2]])
```

```
AB = A * B
np.sum(AB)
```

15

문제 174) 아래의 4 x 4 행렬에서 아래의 3 x 3 행렬만 추출하시오

보기)

```
1  2  3  0
0  1  2  3                  1  2  3
3  0  1  2          →       0  1  2
2  3  0  1                  3  0  1
```

답)

```
import numpy as np

A = np.array([[1,2,3,0],
              [0,1,2,3],
              [3,0,1,2],
              [2,3,0,1]])

A[:3, :3]
```

```
array([[1, 2, 3],
       [0, 1, 2],
       [3, 0, 1]])
```

문제 175) 아래의 행렬에서 아래의 결과 행렬을 출력하시오

보기)

```
1  2  3  0
0  1  2  3                  2  3  0
3  0  1  2          →       1  2  3
2  3  0  1                  0  1  2
```

답)

```
import numpy as np

A = np.array([[1,2,3,0],
```

```
            [0,1,2,3],
            [3,0,1,2],
            [2,3,0,1]])

A[0:3, 1:4]
```

```
array([[2, 3, 0],
       [1, 2, 3],
       [0, 1, 2]])
```

문제 176) 아래의 4 x 4 행렬에서 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

A = np.array([[1,2,3,0],
              [0,1,2,3],
              [3,0,1,2],
              [2,3,0,1]])

row_size = len(A[0])
col_size = len(A)
result = []

for i in range(row_size-2):
    for j in range(col_size-2):
        result.append(np.sum(A[i:i+3, j:j+3]))

print(result)
```

```
[13, 14, 12, 13]
```

문제 177) 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

A = np.array([[1,2,3,0],
              [0,1,2,3],
              [3,0,1,2],
              [2,3,0,1]])

filter = np.array([[2,0,1],
                   [0,1,2],
                   [1,0,2]])

row_size = len(A[0])
col_size = len(A)
result = []

for i in range(row_size-2):
    for j in range(col_size-2):
        result.append(np.sum(A[i:i+3, j:j+3]*filter))

print(result)
```

[15, 16, 6, 15]

문제 178) 위에서 출력한 결과인 1차원 배열인 [15, 16, 6, 15] 결과를 2 x 2 행렬로 변경하시오

답)

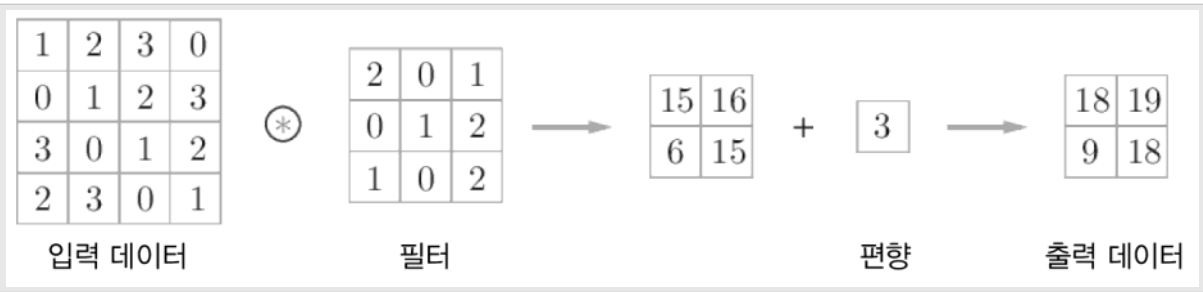
```
import numpy as np

A = np.array([[1,2,3,0],
              [0,1,2,3],
              [3,0,1,2],
              [2,3,0,1]])

filter = np.array([[2,0,1],
                   [0,1,2],
                   [1,0,2]])
```

```
row_size = len(A[0])
col_size = len(A)
result = []

for i in range(row_size-2):
    for j in range(col_size-2):
        result.append(np.sum(A[i:i+3, j:j+3]*filter))

print(np.reshape(result, (2,2)))
```

```
[[15 16]
 [ 6 15]]
```

문제 179) 아래의 그림의 convolution 연산을 파이썬으로 구현하시오

보기)



답)

```
import numpy as np

A = np.array([[1,2,3,0],
              [0,1,2,3],
              [3,0,1,2],
              [2,3,0,1]])

filter = np.array([[2,0,1],
                   [0,1,2],
                   [1,0,2]])

row_size = len(A[0])
col_size = len(A)
result = []

for i in range(row_size-2):
    for j in range(col_size-2):
```

```
        result.append(np.sum(A[i:i+3, j:j+3]*filter))

res = np.reshape(result, (2,2))
bias = 3

print(res + bias)
```

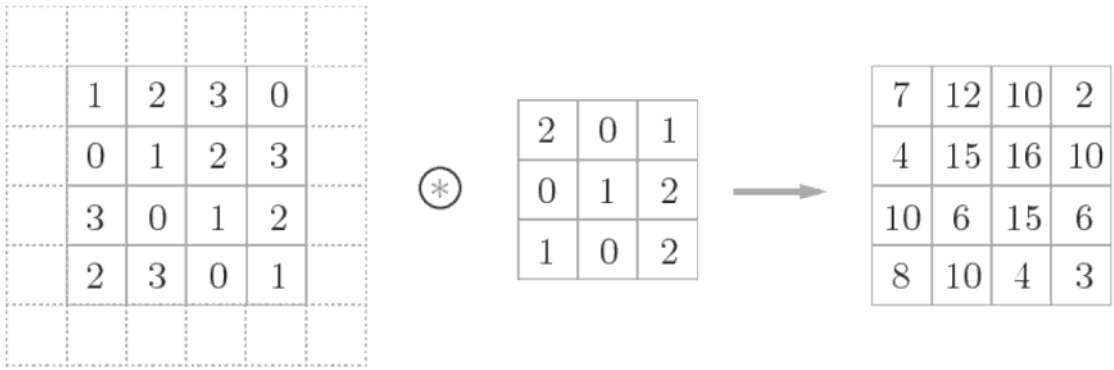
[[18 19]
 [9 18]]

패딩 (Padding)

" 합성곱 연산을 수행하기 전에 입력 데이터 주변을 특정값으로 채워 늘리는 것을 말한다 "

패딩이 필요한 이유

패딩을 하지 않을 경우 data의 공간 크기는 합성곱 계층을 지날 때마다 작아지게 되므로 가장 자리 정보들이 사라지게 되는 문제가 발생하기 때문에 패딩을 사용한다



문제 180) 위에서 출력한 2 x 2 행렬에 제로패딩 1을 수행하시오

답)

```
import numpy as np

r = np.array([[15,16], [6,15]])
r_pad = np.pad(r, pad_width=2, mode='constant', constant_values=0)

print(r_pad)
```

[[0 0 0 0 0 0]
 [0 0 0 0 0 0]

```
[ 0  0 15 16  0  0]
[ 0  0  6 15  0  0]
[ 0  0  0  0  0  0]
[ 0  0  0  0  0  0]]
```

문제 181) 4 x 4 행렬에 3 x 3 필터를 적용해서 결과로 4 x 4 행렬이 출력되게 하려면 제로 패딩을 몇을 해줘야 하는가?

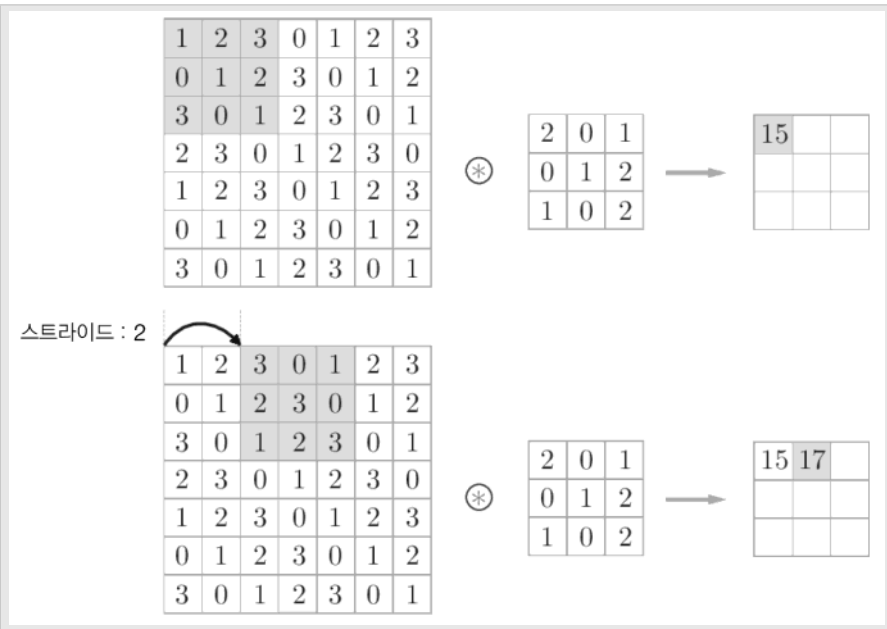
답)

```
import numpy as np

r = np.array([[15,16], [6,15]])
r_pad = np.pad(r, pad_width=2, mode='constant', constant_values=0)

print(r_pad)
```

스트라이드 (Stride)

" 필터를 적용하는 위치의 간격 "

만약 스트라이드를 2로 설정하면 아래와 같이 나온다



교재 p. 234 공식

$$OH = \frac{H + 2P - FH}{S} + 1$$

$$OW = \frac{\mathrm{W} + 2P - FW}{S} + 1$$

문제 182) 패딩을 답으로 해서 위의 식을 풀어보시오

답)

```
P = (OH-1)S - (1-F)H
    --------------------
              2
```

문제 183) 입력 이미지 4 x 4 행렬에 필터 3 x 3 행렬을 합성곱한 결과 행렬이 4 x 4 행렬이 되려면 패딩이 몇인지 출력하시오

답)

```
P = ((4 - 1) * 1 - 4 + 3) / 2 = 1
```

문제 184) 위의 패딩 공식을 구현하는 파이썬 함수를 생성하시오

답)

```
def padding(H, S, OH, FH):
    P = ((OH-1)*S - (H-FH)) / 2
    return P

print(padding(4, 1, 4, 3))
```

1.0

문제 185) 입력행렬(6x6), 스트라이드 1, 출력행렬(6x6), 필터행렬(3x3) 의 패딩이 어떻게 되는지 출력하시오

답)

```
def padding(H, S, OH, FH):
    P = ((OH-1)*S - (H-FH)) / 2
    return P
```

```
print(padding(6, 1, 6, 3))
```

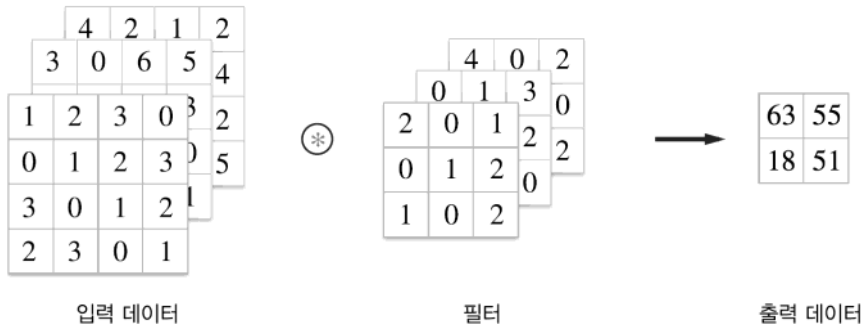
1.0

3차원 데이터의 합성곱 연산

(p. 235)

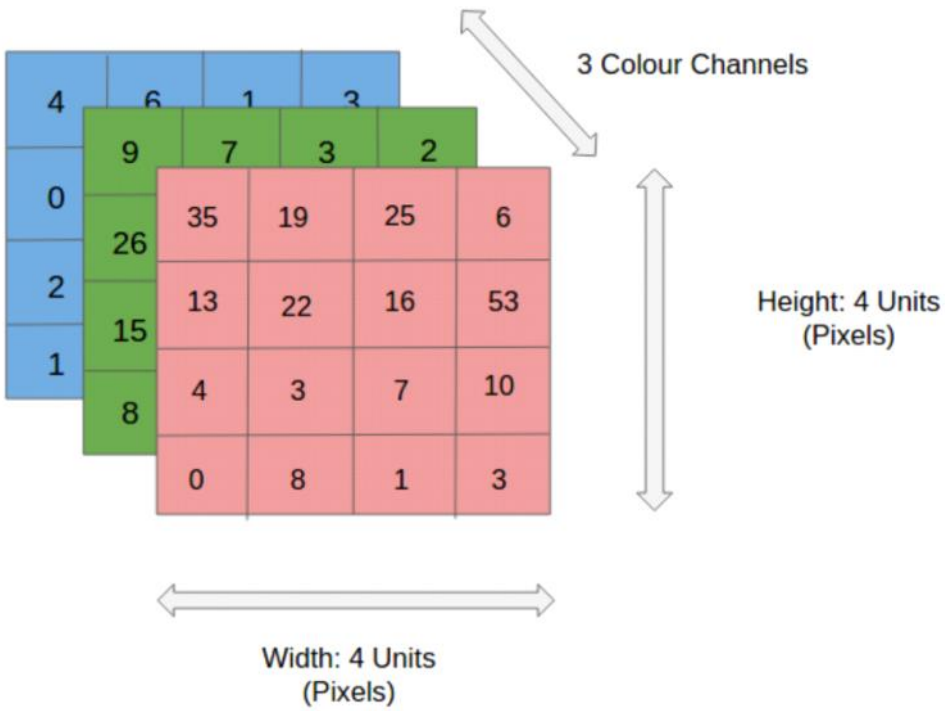
이미지의 색깔이 보통은 흑백이 아니라 RGB 컬러 이므로 RGB(Red, Green, Blue) 컬러에 대해서 합성곱을 해야한다.

아래의 그림은 3차원 데이터의 합성곱 연산의 예이다.



입력 데이터에 R, G, B 각각 3개의 행렬이 입력된다.
이제까지는 하나의 행렬에 필터를 입혔다.

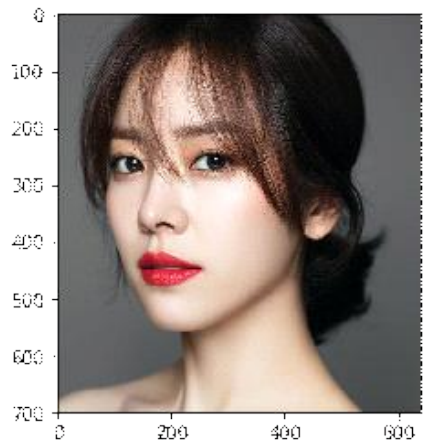
먼저 RGB에 대해 이해를 해야한다



문제 186) 한지민 사진을 3차원 행렬로 변환하시오

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_test_data/hanjimin.jpg')
img_pixel = np.array(img)
plt.imshow(img_pixel)
```



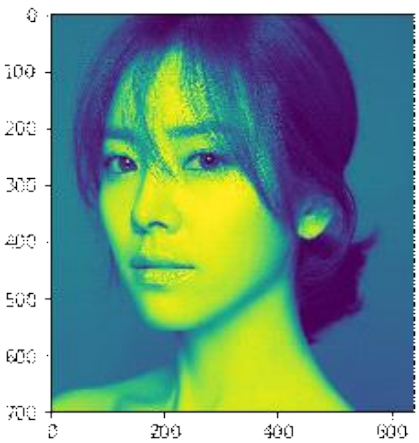
문제 187) 한지민 사진에서 Red 행렬만 출력하고 Red 행렬만 시각화 하시오

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_test_data/hanjimin.jpg')
img_pixel = np.array(img)
plt.imshow(img_pixel[:,:,0])
print(img_pixel[:,:,0])
```

```
[[ 99  99  99 ...  92  92  92]
```

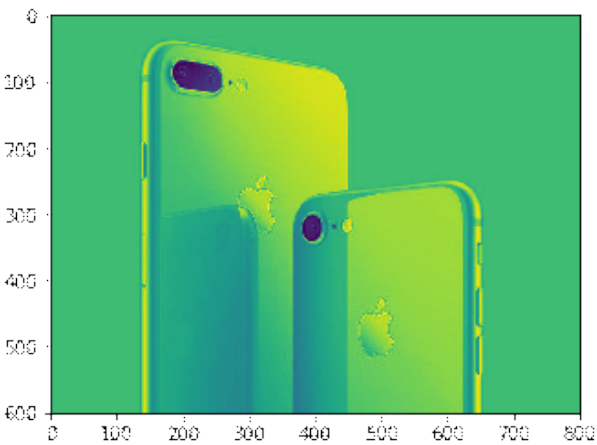
```
[ 95  95  95 ...  86  86  86]
[ 93  93  93 ...  86  86  86]
...
[206 206 207 ...  72  72  72]
[207 207 208 ...  72  72  72]
[208 209 210 ...  72  72  72]]
```



문제 188) 원하는 사진을 직접 내려받아 RGB 이미지를 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_test_data/red_iphone.JPG')
img_pixel = np.array(img)
plt.imshow(img_pixel[:,:,0])
print(img_pixel[:,:,0])
```

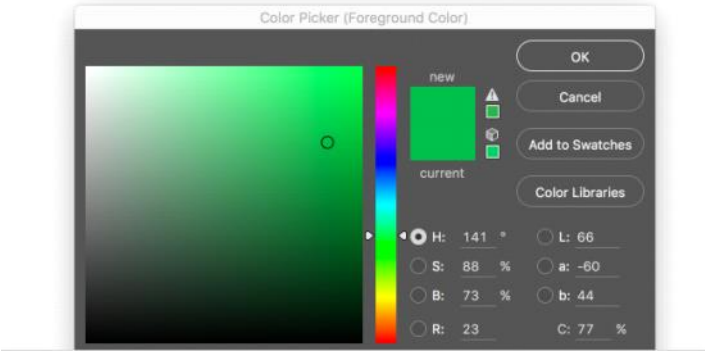


설명)

RGB 말고 HSB 라는 개념이 있는데 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.



문제 189) RGB 행렬을 이해하기 위한 아래의 numpy array 를 이해하시오

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]],
                 [[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]],
                 [[4, 2, 1, 2, 2],
                  [0, 1, 0, 4, 2],
                  [3, 0, 6, 2, 2],
                  [4, 2, 4, 5, 2],
                  [3, 4, 1, 0, 0]]
                ])
```

문제 190) 문제 189번의 data 행렬의 shape를 확인하시오

답)

```
data.shape
```

(3, 5, 5)

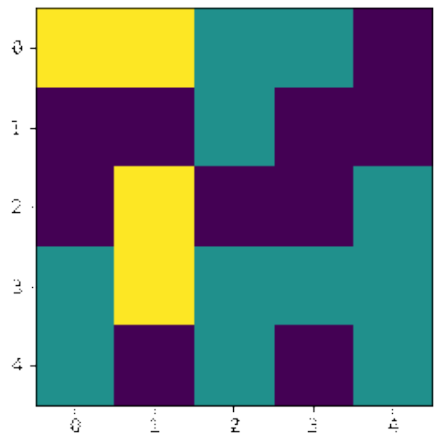
설명)

```
(3, 5, 5)
(색상, 가로, 세로)

print(img_pixel[:,:,0])
                (가로, 세로, 색상)
```

문제 191) 위의 data 행렬에서 Red 행렬만 출력하고 시각화도 하시오

답)

```
data[0, :, :]

plt.imshow(data[0,:,:])
```



문제 192) 아래의 numpy array의 행렬을 확인하시오

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]]  ])

Filter.shape
```

(1, 3, 9)

문제 193) 위의 행렬을 (3,3,3) 행렬로 변환하시오

답)

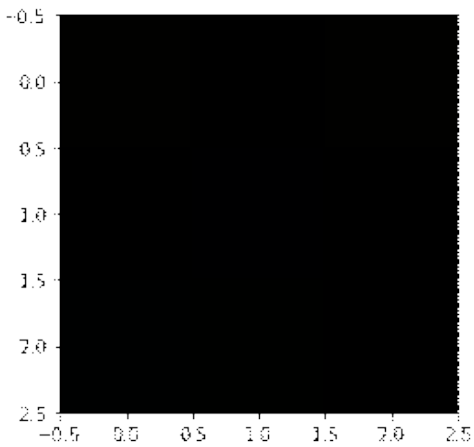
```
Filter=np.array([
    [[1,1,-1,-1,0,0,1,1,0],
     [-1,-1,0,0,-1,1,0,-1,0],
     [-1,1,1,-1,1,-1,0,0,-1]]  ]).reshape(3,3,3)

print(Filter)
print("₩n₩n", Filter.shape)
```

```
[[[ 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]]]


 (3, 3, 3)
```

문제 194) 위의 행렬을 시각화하시오

답)

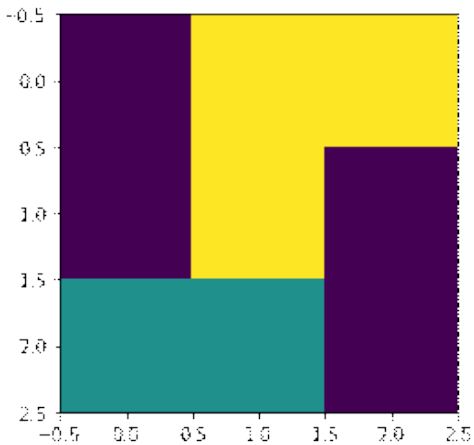
```
plt.imshow(Filter)
```



문제 195) 위의 행렬 Filter 에 Red, Green, Blue 행렬을 각각 시각화 하시오

답)

```
plt.imshow(Filter[0, :, :])
plt.imshow(Filter[1, :, :])
plt.imshow(Filter[2, :, :])
```

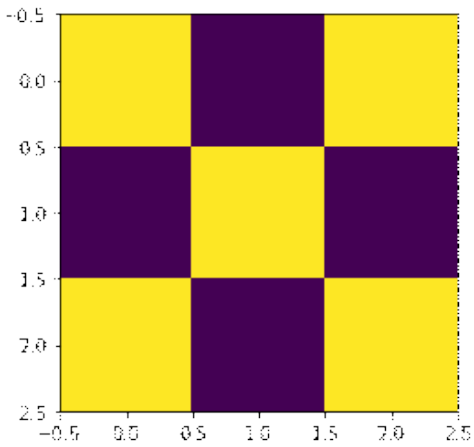


문제 196) 아래의 코드를 실행해보시오

답)

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

print(a)
plt.imshow(a[0, :, :])
plt.show
```

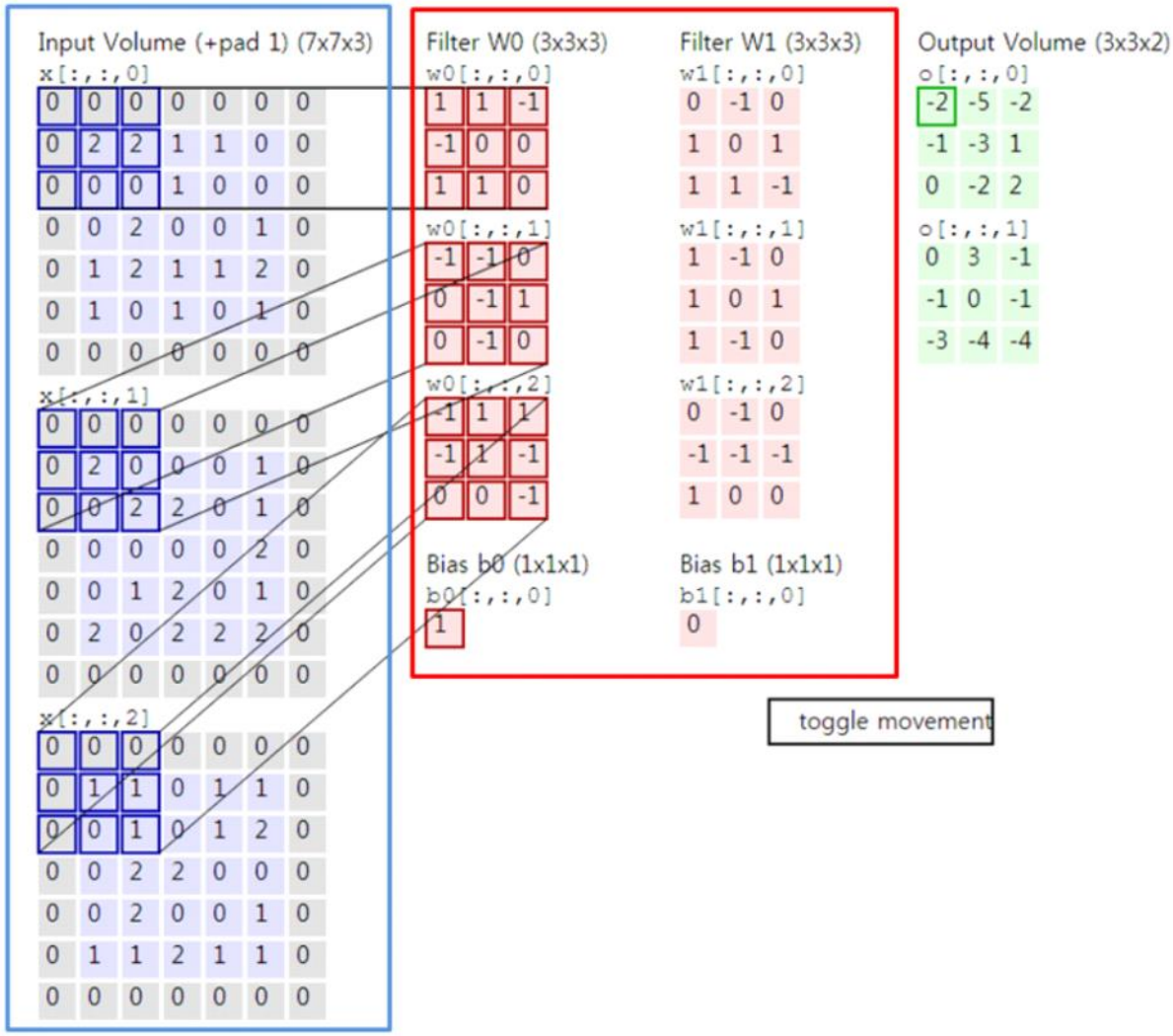
```
[[[1 0 1]
  [0 1 0]
  [1 0 1]]]
```



문제 197) 아래의 원본 이미지인 data 행렬과 filter 행렬을 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]],
                 [[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]],
                 [[4, 2, 1, 2, 2],
                  [0, 1, 0, 4, 2],
                  [3, 0, 6, 2, 2],
                  [4, 2, 4, 5, 2],
                  [3, 4, 1, 0, 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)
```



문제 198) 아래의 원본이미지의 zero 패딩 1을 한 결과를 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]],
                 [[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]],
                 [[4, 2, 1, 2, 2],
                  [0, 1, 0, 4, 2],
                  [3, 0, 6, 2, 2],
                  [4, 2, 4, 5, 2],
                  [3, 4, 1, 0, 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)



npad = ((0,0),(1,1),(1,1))

data_pad = np.pad(data, pad_width=npad, mode='constant',constant_values=0)
print(data_pad)
```

문제 201) 위의 3x3 행렬의 RGB 행렬을 각각 출력하면?

답)

```
print(data_pad[0, :3, :3])
print(data_pad[1, :3, :3])
print(data_pad[2, :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]]
```

문제 202) 아래의 Filter 에서 아래의 행렬을 추출하시오

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[0, :, :])
```

```
[[ 1   1 -1]
 [-1   0   0]
 [ 1   1   0]]
```

문제 203) 위의 3개의 행렬 중 원본 이미지의 Red 행렬과 아래의 Filter의 Red 행렬과의 합성곱을 수행하시오

답)

```
data_red = data_pad[0, :3, :3]
F_red = Filter[0, :, :]

print(data_red * F_red)
```

```
[[0 0 0]
 [0 0 0]
 [0 0 0]]
```

문제 204) 원본이미지의 RGB 3개의 행렬과 아래의 필터 RGB 3개의 행렬을 각각 행렬곱 한 후 그 원소들을 다 합칠 결과 숫자 하나를 출력하시오

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)

```
rst = 0

for i in range(len(data_pad)):
    data_RGB = data_pad[i, :3, :3]
    F_RGB = Filter[i, :, :]
    rst += np.sum(data_RGB * F_RGB)

print(rst)
```

문제 205) 1칸 스트라이드 한 원본이미지의 3x3 행렬 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

결과 :   -4
```

답)

```
rst = 0

for i in range(len(data_pad)):
    data_RGB = data_pad[i, :3, 1:4]
    F_RGB = Filter[i, :, :]
    rst += np.sum(data_RGB * F_RGB)

print(rst)
```

문제 206) 아래의 합성곱 연산 (25번 연산) 을 하는 결과를 파이썬으로 구현하시오
(스트라이드:1 이므로 결과가 5x5 행렬이 나와야한다)

답)

```
rst = []

for j in range(len(data_pad[0])-2):
    for k in range(len(data_pad[0])-2):
        rst.append(np.sum(data_pad[:, k:k+3, j:j+3] * Filter[:, :, :]))

rst = np.array(rst)
rst = rst.reshape(5, 5)
rst
```

```
array([[-3, -1, -2, -3, -1],
       [-4,  2,  0,  7,  2],
       [-6,  0, -4, -6, -3],
       [-1,  3,  3, -6, -3],
       [-3, -1, -1, -3,  0]])
```

패딩과 Feature map의 shape과의 관계

이미지의 특징(feature map) 을 추출하는 과정
filter(가중치) 를 이용해서 추출한다

원본이미지 1장 * 필터 50개 = feature map의 개수 (50개)
7 x 7 3 x 3 7 x 7

패딩을 2로 해주면 입력값과 동일한 shape의 feature map이 출력된다

Convolution 층을 통과한 출력 이미지의 사이즈 계산방법

문제 207) 아래와 같이 입력 행렬과 필터 행렬과 스트라이드와 패딩을 입력받아 출력 행렬의 shape을 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, 2], [1, 0, 1, 0, 1]],
              [[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]],
              [[1, 1, 0, 1, 1], [0, 1, 0, 1, 2], [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)
```

참고 공식)

$$OH = \frac{H + 2P - FH}{S} + 1$$

$$OW = \frac{\mathrm{W} + 2P - FW}{S} + 1$$

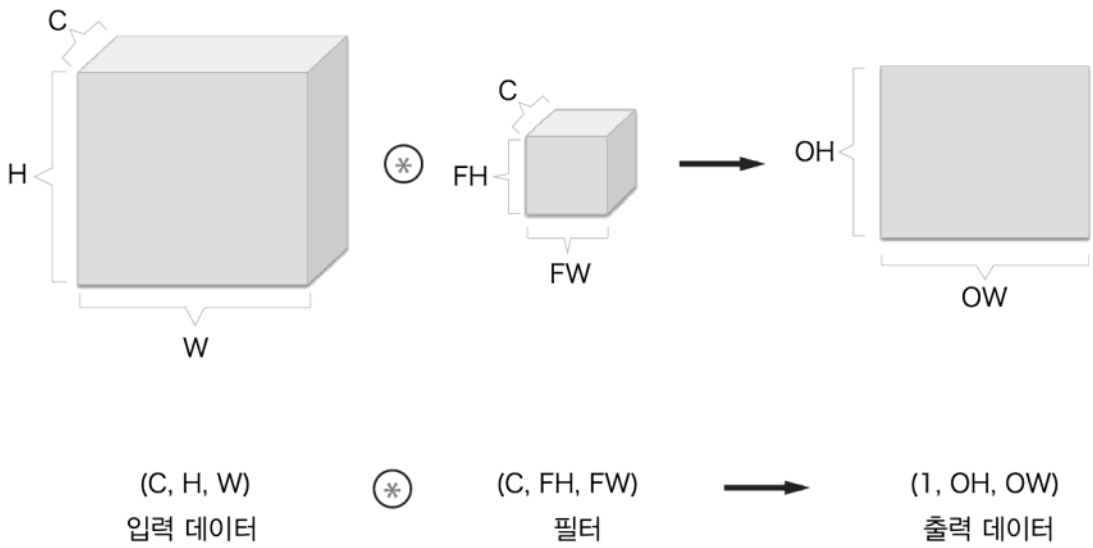
답)

```
a = np.array([[[2, 2, 1, 1, 0], [0, 0, 1, 0, 0], [0, 2, 0, 0, 1], [1, 2, 1, 1, 2], [1, 0, 1, 0, 1]],
              [[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]],
              [[1, 1, 0, 1, 1], [0, 1, 0, 1, 2], [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 output(data, Filter, S, P):
    H = len(data)
    W = len(data[0])
    FH = len(Filter)
    FW = len(Filter[0])

    return (H + 2*P - FH / S) + 1

output(a, Filter, 1, 2)
```

블록으로 생각하기

" 3차원 합성곱연산은 데이터와 필터를 직육면체 블록이라고 생각하면 쉽다"
블록은 3차원 직육면체 (채널, 높이, 너비)로 구성됨
필터의 채널(C), 필터의 높이(FH), 필터의 너비(FW)

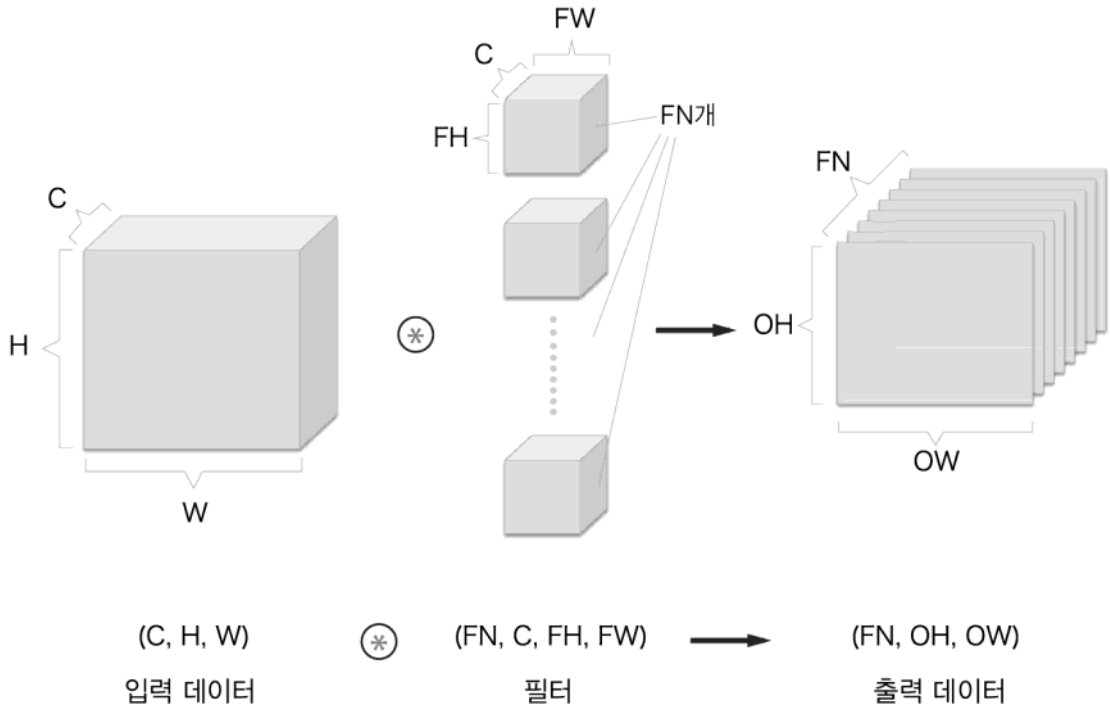


설명)

아이린 사진 한장 (RGB) 를 RGB 필터로 합성곱해서 2차원 출력행렬 (feature map) 1장 출력

위의 그림은 feature map이 한 개가 나오고 있는데 실제로는 아이린 사진 1장에 대해서 여러 개의 feature map이 필요하다

즉, Filter의 개수를 늘려야 한다



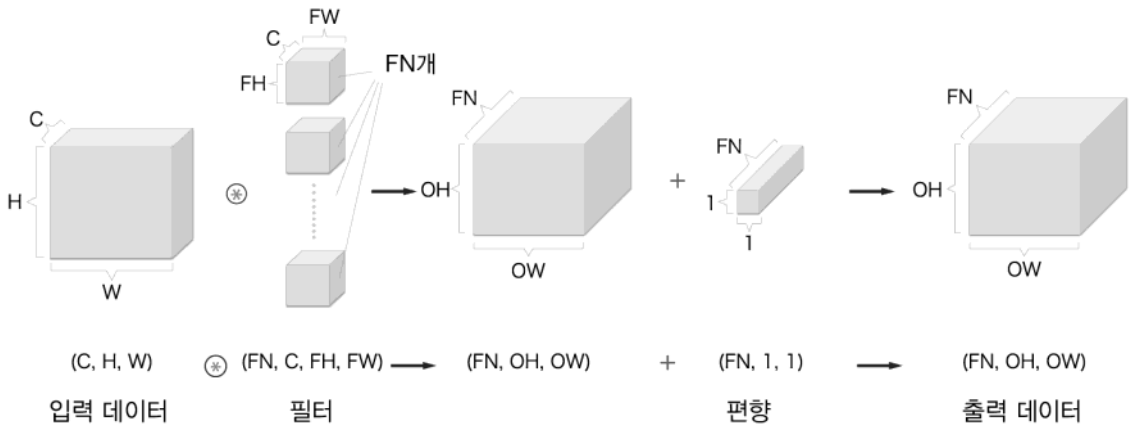
문제 208) 설현 사진 50장과 아이린 50장, 총 100장의 사진을 신경망에 입력해서 설현과 아이린을 구분하는 신경망을 만든다고 할 때 RGB 필터를 30개를 사용하면 출력 feature map 이 총 몇 개일까?

답)

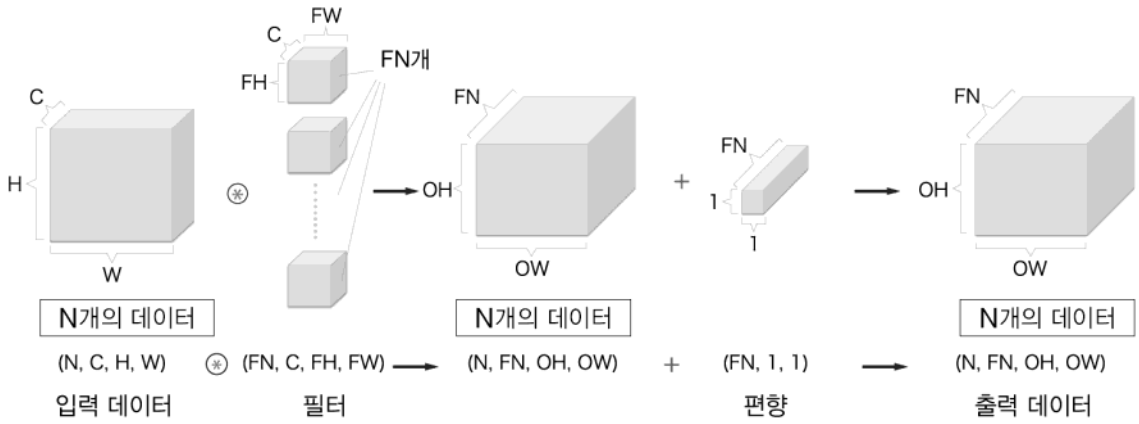
사진 당 30개의 필터를 사용해서 30개의 feature map 이 나오므로 총 3000장이 나온다

배치 처리

합성곱 연산에서도 편향이 쓰이므로 편향을 더하면 어떤 그림일까?



위의 그림은 이미지를 1장 씩 넣어서 학습시키는 것이므로 학습 속도가 느리므로
여러 장의 이미지를 한번에 입력해서 학습 (미니배치) 시키면 아래의 그림이 된다
↓
처리 효율 증가



결국 합성곱 계층을 구현할 때 흘러가는 행렬이 4차원 행렬이 연산이 될 것인데 그러면 연산 속도가 느리므로 행렬 계산을 할 때 행렬 연산을 빠르게 하려면 4차원이 아니라 2차원으로 차원 축소가 필요하다

그래서, 필요한 함수가 im2col 함수이다 (4차원 행렬을 2차원으로 바꿔줌)

풀링 계층

풀링은 세로, 가로 방향의 공간을 줄이는 연산

풀링 계층의 특징

1. 학습해야 할 매개변수가 없다
2. 채널 수가 변하지 않는다
3. 입력의 변화에 영향을 적게 받는다

풀링의 종류

1. 최대풀링 : 컨볼루션 데이터에서 가장 큰 값을 대표값으로 선정
2. 평균풀링 : 컨볼루션 데이터에서 모든 값의 평균 값을 대표값으로 선정
3. 확률적 풀링 : 컨볼루션 데이터에서 임의 확률로 한 개를 선정

합성곱/풀링 계층 구현하기

4차원 배열

문제 209) 칠판에 나온 아이린 사진이라 생각하고 한장의 4차원 행렬을 만드시오
(RGB의 7 x 7 행렬 1장)

답)

```
import numpy as np


x1= np.random.rand(1,3,7,7)
```

```
# print(x1)
print(x1.shape)
```

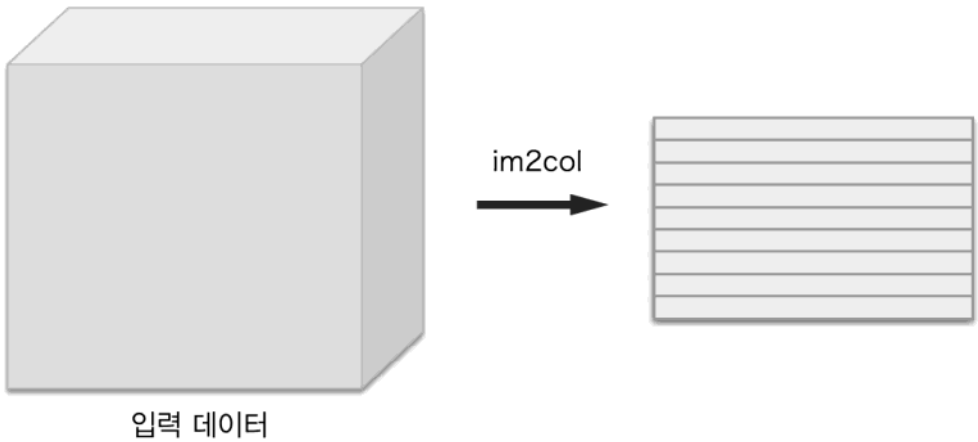
(1, 3, 7, 7)

im2col로 데이터 전개하기

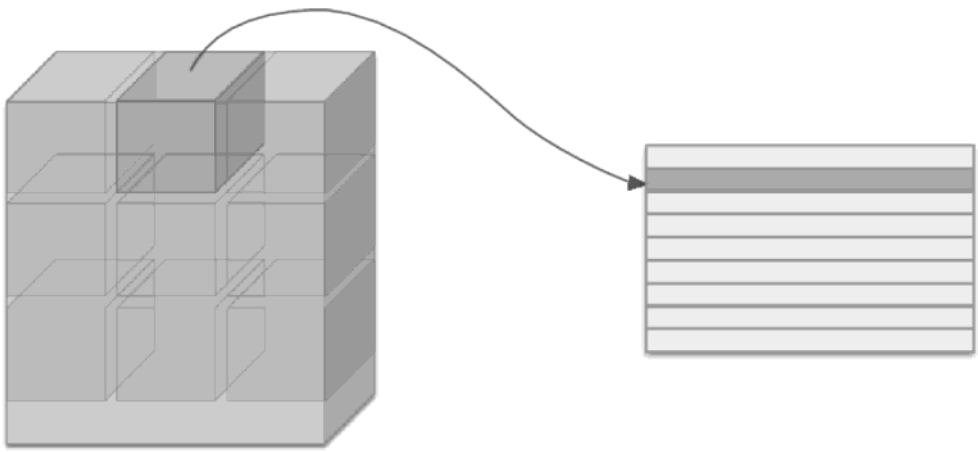
합성곱 연산을 곧이곧대로 구현하려면 for 문을 겹겹이 사용해야 한다.
하지만 넘파이에서 원소에 접근할 때 for 문을 사용하지 않는 것이 성능 면에서 바람직합니다.

im2col 함수 란?

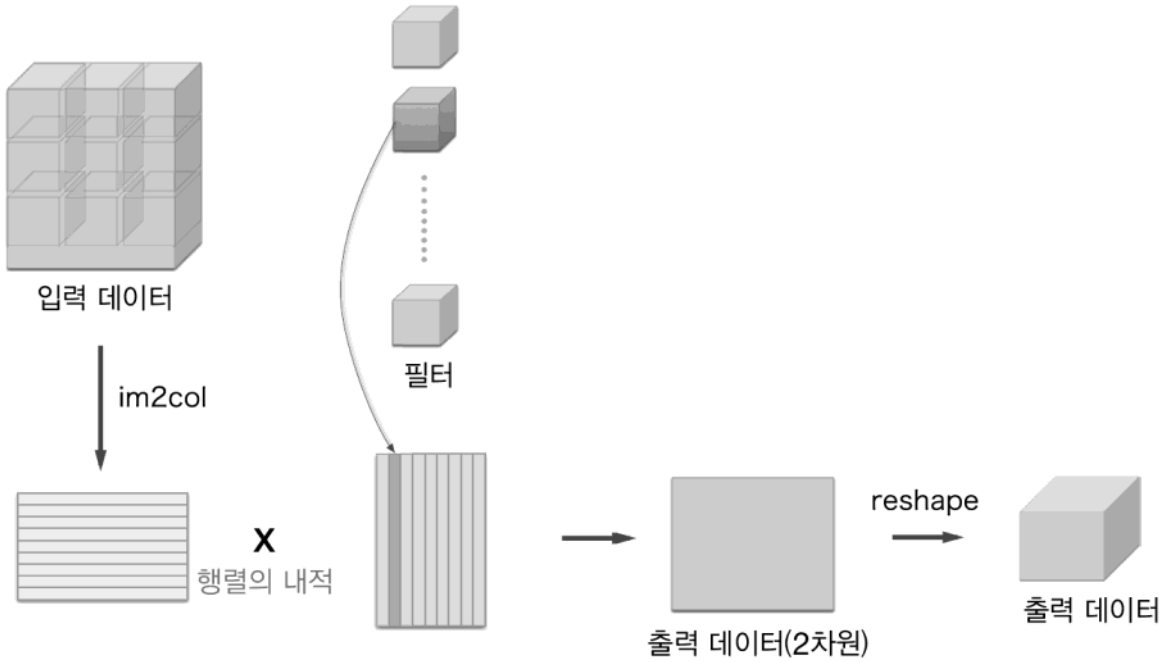
입력 데이터를 필터링 (가중치 계산) 하기 좋게 전개하는(펼치는) 함수
4차원 → 2차원 행렬



아래 그림처럼 3차원 블록을 한 줄로 늘어놓는다

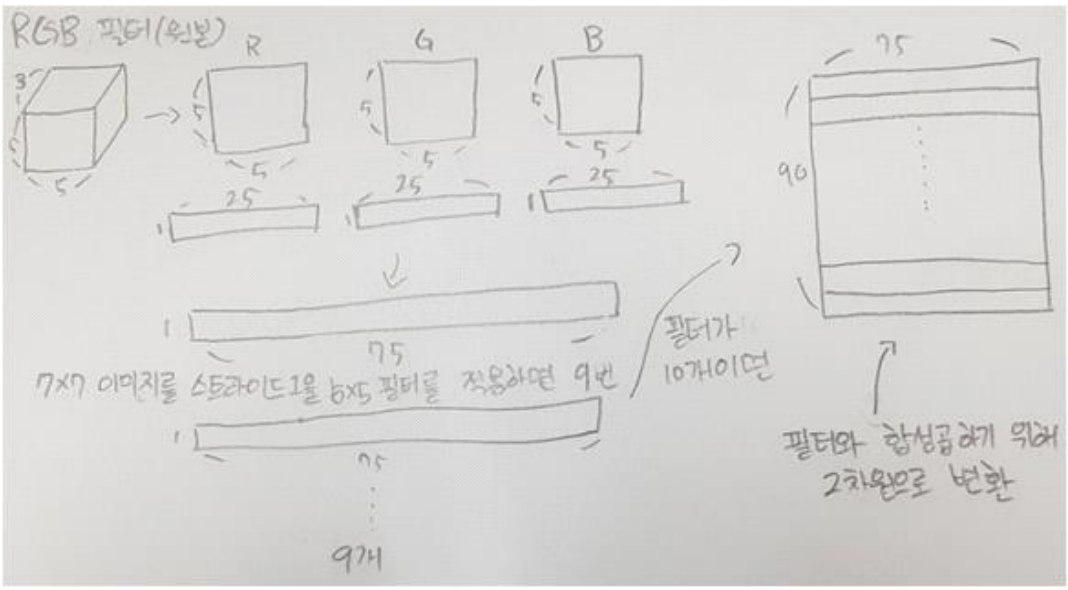


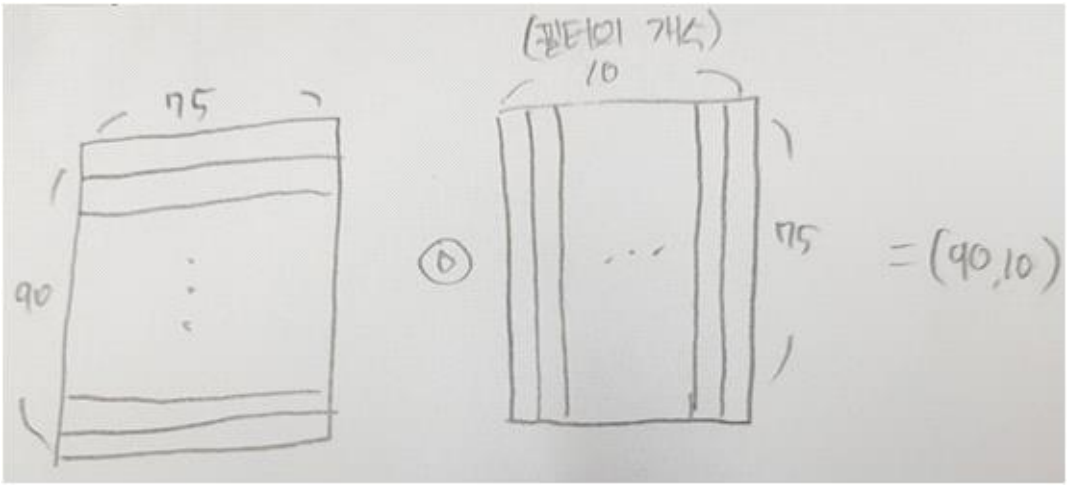
im2col : image to column (이미지를 행렬로 변환)



위 그림을 보면,
im2col로 입력 데이터를 전개한다 → 필터(가중치)를 1열로 전개한다 → 두 행렬 내적 계산

4차원 원본 7 x 7 행렬을 필터 5 x 5 행렬과 합성곱 하기 위해 2차원으로 변환하는 방법





합성곱 계층 구현하기

문제 210) common/util.py 안에 im2col 함수를 사용해서 아래 4차원 행렬을 2차원으로 변경하시오

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



a = np.random.rand(1,3,7,7)

col = im2col(a,5,5,stride=1,pad=0)
print(col.shape)
```

(9, 75)

문제 211) 아이린 사진 10장을 랜덤으로 생성하시오

답)

```
i_lyn = np.random.rand(10, 3, 7, 7)
```

문제 212) 아이린 사진 10장을 im2col 함수의 넣어서 2차원 행렬로 변환하시오 (Filter : 5 x 5)

보기)

```
4차원          →        2차원
(10, 3, 7, 7)             (90, 75)
```

답)

```
i_lyn = np.random.rand(10, 3, 7, 7)

col = im2col(i_lyn, 5, 5, stride=1, pad=0)
print(col.shape)
```

(90, 75)

설명)

```
(9, 75) 짜리가 10개가 아래에 붙기 때문에 (90, 75) 가 된다
```

문제 213) MNIST 데이터 100장을 합성곱하기 편하게 im2col 함수에 넣었을때 나오는 출력 형상을 예상하시오 (Filter : 5 x 5 흑백 채널)

보기)

```
4차원          →          2차원
(100, 1, 28, 28)          ( ? )
```

답)

```
a = np.random.rand(100, 1, 28, 28)
col = im2col(a, 5, 5, stride=1, pad=0)

print(col.shape)
```

(57600, 25)

설명)

```
4차원 행렬을 2차원으로 변경해서 합성곱 연산을 빠르게 진행하려고 im2col 함수를 이용했다
```

2차원으로 변경해야 할 행렬 2가지

```
1. 원본 이미지를 필터 사이즈에 맞게 2차원으로 변경
(10, 3, 7, 7)          →          (90, 75)
            im2col 함수 사용해서

2. 4차원 필터 행렬을 2차원으로 변경
(10, 3, 5, 5)          →          (75, 10)
       reshape 함수의 -1 을 이용해서
```

문제 214) 아래의 Filter 를 생성하고 shape 를 확인하고 전치 시키시오

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)
```

답)

```
import numpy as np
```

```
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)

문제 215) Filter (3, 5, 5) 행렬을 (3, 25) 행렬로 변경하시오

답)

```
import numpy as np

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)
a = Filter.T.reshape(3, -1)
print(a.shape)
```

(5, 5, 3)
(3, 25)

문제 216) 아래의 4차원 행렬의 Filter 를 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)
```

문제 217) 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)

print(filter2.reshape(10, 3, -1).shape)
```

문제 218) 아래의 3차원 행렬을 2차원 행렬로 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)

문제 219) 위 코드에서 (10, 75)를 (75, 10) 으로 전치하시오

답)

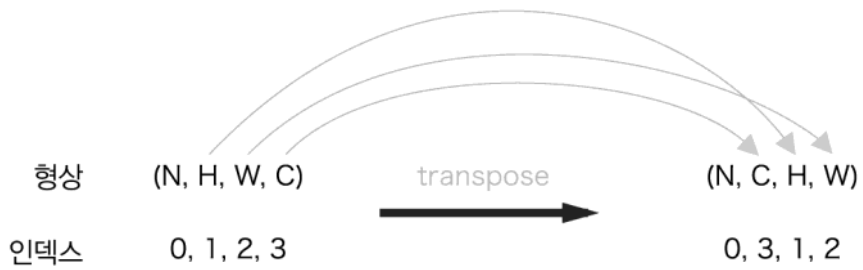
```
print(b.T.shape)
```

(75, 10)

문제 220) 책 246 페이지에 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
        print('out.shape:', out.shape)

        out = out.reshape(N, out_h, out_w, -1).transpose(0, 3, 1, 2)

        return out
```

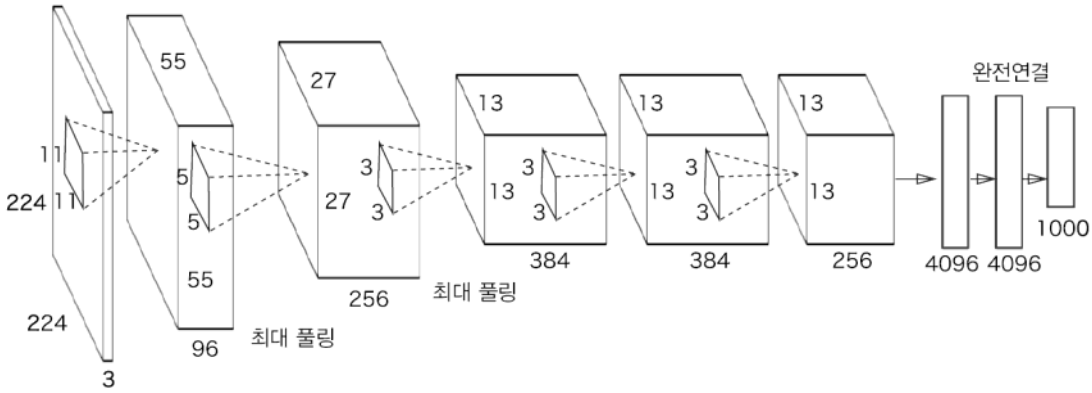


문제 221) 위에서 만든 Convolution 클래스를 객체화 시켜서 Convolution 층을 구현하시오

답)

```
x1 = np.arange(1470).reshape(10,3,7,7)
W1 = np.arange(750).reshape(10,3,5,5)
b1 = 1
```

```
conv = Convolution(W1,b1)
f = conv.forward(x1)
print ('f.shape=', f.shape)
```



Convolution 클래스내에서 일어나는 일

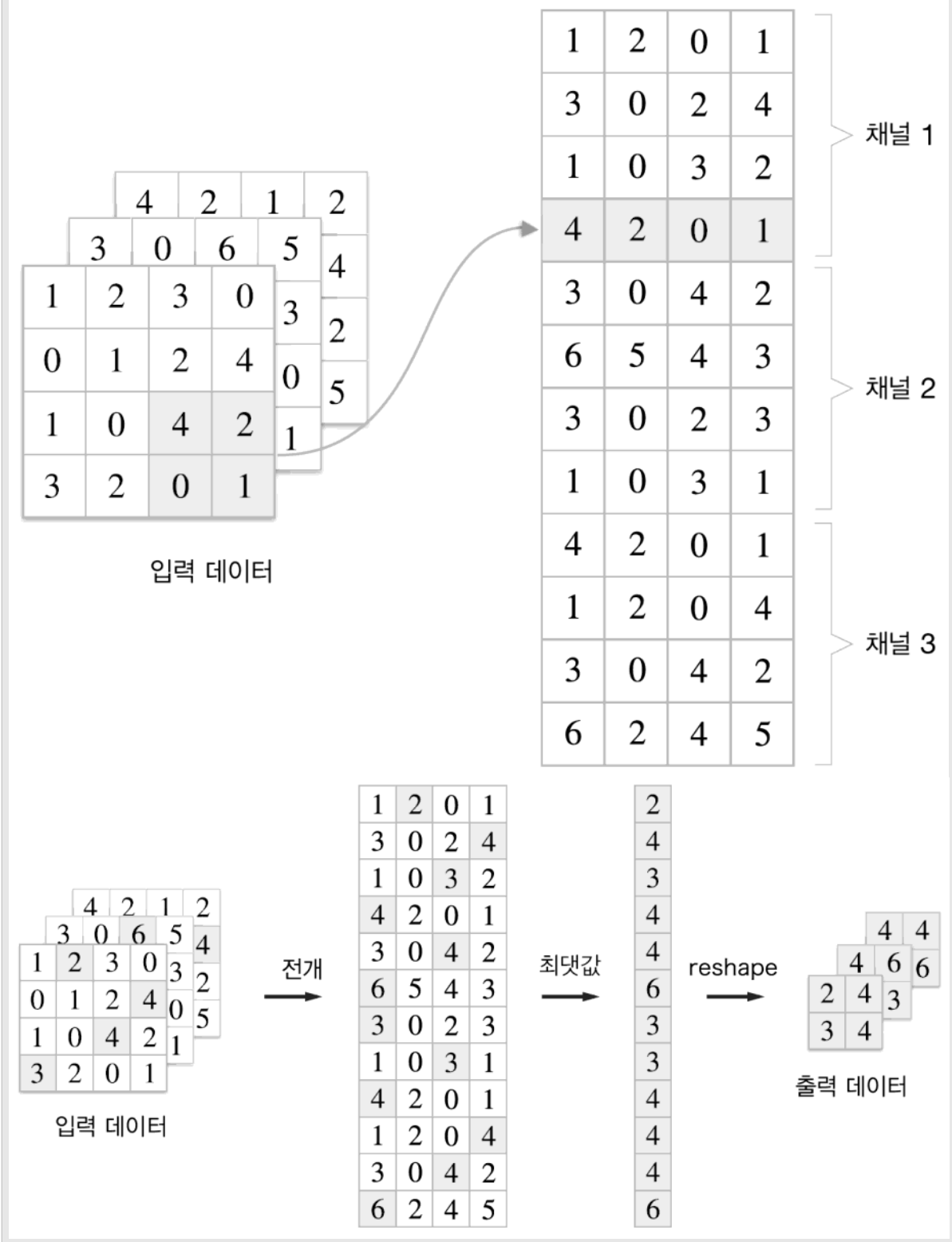
1. 원본이미지를 im2col로 2차원 행렬로 변경한다.
2. filter를 reshape로 2차원 행렬로 변경한다.
3. 2차원 행렬로 변경한 두 행렬을 내적한다.
4. 내적한 결과 2차원 행렬을 다시 4차원으로 변경한다.

CNN 층의 구조

conv → pooling → fully connected

풀링 계층 구현하기

최대 풀링이 일어나는 과정



문제 222) 아래의 이미지를 손으로 최대 풀링하시오

보기)

```
21  8  8  12
12 19  9   7
8   10 4   3
18 12  9  10
```

답)

```
21  12
18  10
```

문제 223) max_pooling 함수를 이용해서 4x4 행렬을 2x2 행렬로 변경하시오

max_pooling 함수 생성)

```
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
```

답)

```
a = np.random.rand(4,4)
max_pooling(a)
```

```
array([[0.70330599, 0.93562072],
       [0.70889038, 0.96730929]])
```

```
In [22]: a = np.random.rand(4,4)
         a
Out[22]: array([[0.62495867, 0.562403  , 0.6478887 , 0.16934186],
                [0.02336976, 0.8067352 , 0.57278934, 0.56714878],
                [0.05857559, 0.64014559, 0.93050781, 0.46696343],
                [0.79052221, 0.96882288, 0.28643938, 0.7705385 ]])

In [23]: max_pooling(a)
Out[23]: array([[0.8067352 , 0.6478887 ],
                [0.96882288, 0.93050781]])
```

문제 224) 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)

        # 입력 데이터 전개 (1)
        col = im2col(x, self.pool_h, self.pool_w, self.stride, self.pad)
        col = col.reshape(-1, self.pool_h*self.pool_w)

        # 행별 최댓값 (2)
        out = np.max(col, axis=1)

        # 적절한 모양으로 성형 (3)
        out = out.reshape(N, out_h, out_w, C).transpose(0, 3, 1, 2)

        return out
```

Convolution 층을 통과한 출력 이미지의 사이즈 계산방법

$$OH = \frac{H + 2P - FH}{S} + 1$$

$$OW = \frac{W + 2P - FW}{S} + 1$$

문제 225) MNIST (28 x 28) 데이터가 Convolution 층을 통과했을 때 출력이미지의 사이즈를 알아내시오 (Filter 사이즈 : 5 x 5, Stride : 1, Padding : 0)

답)

```
OH = (28 + 2*0 - 5 / 1) + 1 = 24

24 x 24
```

문제 226) 위의 출력 이미지가 풀링 계층의 입력 이미지 (24 x 24) 행렬로 통과할 때 결과는 어떻게 되는가? (pool_h : 2, stride : 2)

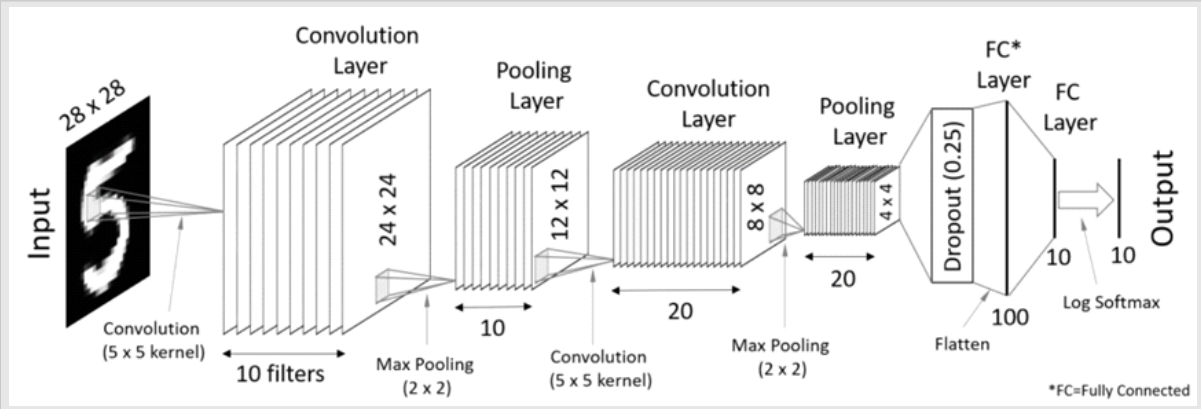
답)

```
H = 24
```

```
pool_h = 2
stride = 2
out_h = int(1 + (H - pool_h) /  stride)
out_h
```

문제 227) MNIST 데이터를 CNN으로 구현했을때의 신경망을 그림으로 확인하시오

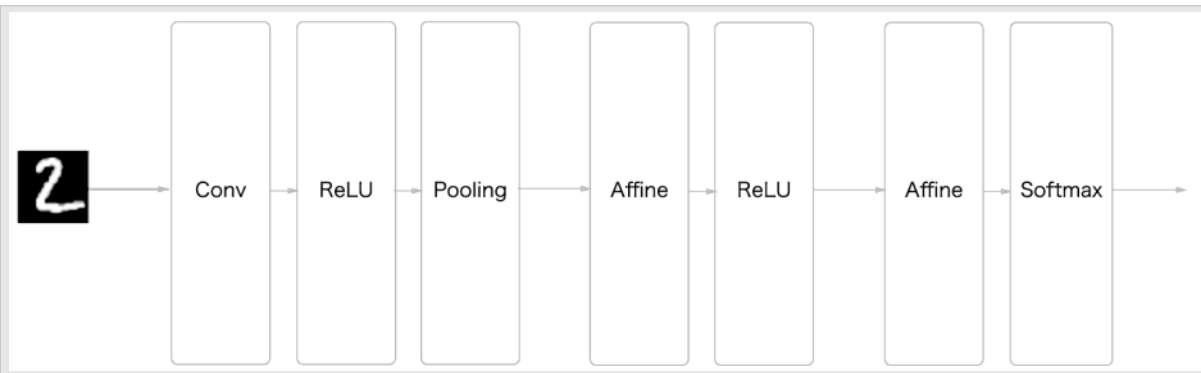
답)



이 부분 필터 : 2장이어서 10장에서 20장이 된 것

CNN 구현하기

단순한 CNN의 네트워크 구성



문제 228) 위의 간단한 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 초깃값'으로 설정
        'sigmoid'나 'xavier'로 지정하면 'Xavier 초깃값'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
        pool_output_size = int(filter_num * (conv_output_size / 2) * (conv_output_size / 2))

        # 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



# 최적화 시킨 파라미터를 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)



# 저장한 파라미터 로드
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))


# 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()
```

문제 229) 위의 CNN 신경망 코드에 가중치 초기화 선정인 Xavier 를 적용해서 테스트하시오

답)

```
- 1. Xavier 적용

# 가중치 초기화
        self.params = {}
```

```
        self.params['W1'] = np.sqrt(1/input_dim[1]) * ₩
                            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)



(60000, 1, 28, 28)
train acc, test acc | 0.15733333333333333, 0.1607
(60000, 1, 28, 28)
train acc, test acc | 0.96, 0.9607
(60000, 1, 28, 28)
train acc, test acc | 0.97245, 0.9702
```

문제 230) 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()
```

문제 231) 위의 CNN을 이용한 3층 신경망에 합성곱 계층과 풀링 계층을 하나 더 추가하시오

답)
