

딥러닝

2018년 8월 13일 월요일 오후 4:37

목차

1. 이 책을 보기 위해 필요한 파이썬 기본 문법
2. 퍼셉트론 (신경망의 하나의 세포를 컴퓨터로 구현)
3. 신경망 (신경망의 활성화 함수, 3층 신경망 생성)
4. 신경망 학습 (손실(오차) 함수, 수치 미분, 경사하강법, 학습알고리즘 구현)
5. 오차 역전파 (계산 그래프, 연쇄법칙, 역전파, 단순한 계층 구현)
6. 신경망을 학습시키는 여러 기술들 소개 (경사하강법의 종류, 배치, 정규화, 드롭아웃)
7. CNN (합성곱 신경망)
사진을 신경망에 입력해서 이 사진이 어떤 사진인지 컴퓨터가 맞히는 방법을 구현
8. 딥러닝의 역사

+ 텐서플로를 이용한 신경망 구현
+ 강화학습

목표 : 이미지를 분류할 수 있는 신경망을 구현

1. 폐결절 사진 VS 정상 폐 사진 구별
---> segmentation 으로 mri 사진 분류
2. 제조업에서 만드는 제품들의 불량 여부 확인
ex. 스노우 보드 : 정상 스노우보드 vs 기스가 있는 스노우 보드
옷감 : 정상 옷감 vs 불량 옷감 구분

인공지능의 눈 ? cnn

- 사진속의 사람, 동물 등을 구별 할 수 있다.
- 다음 카카오 로드뷰에 사람 얼굴이나 차 번호등을 개인정보 보호법상 반드시 모자이크 처리를 해야하는데 너무 많은 양이라 사람이 일일이 할 수 없다.

ex. 007 영상 --> object detection, yolo 기술
mri 사진 --> segmentation

인공지능의 입 ? rnn

1. 파이썬 기본 문법

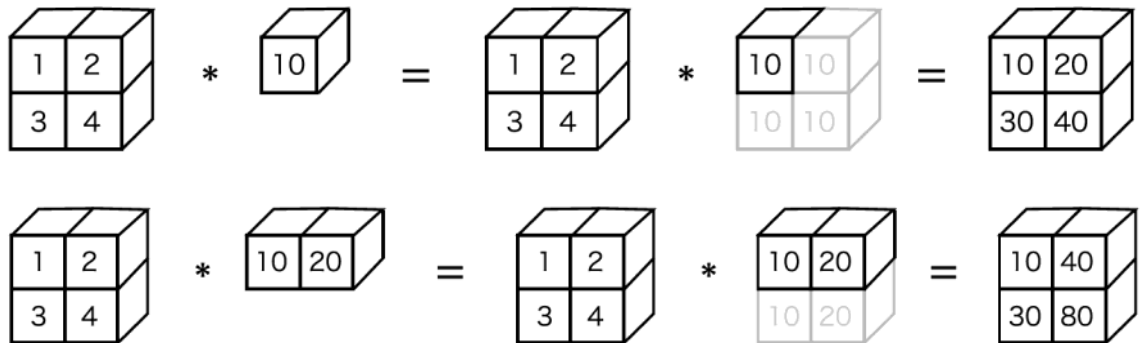
2018년 8월 14일 화요일 오전 10:25

■ numpy 란?

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

■ numpy의 broadcast 기능 (p39)

형상이 다른 배열끼리도 계산할 수 있다.



■ matplotlib 사용법 (p41)

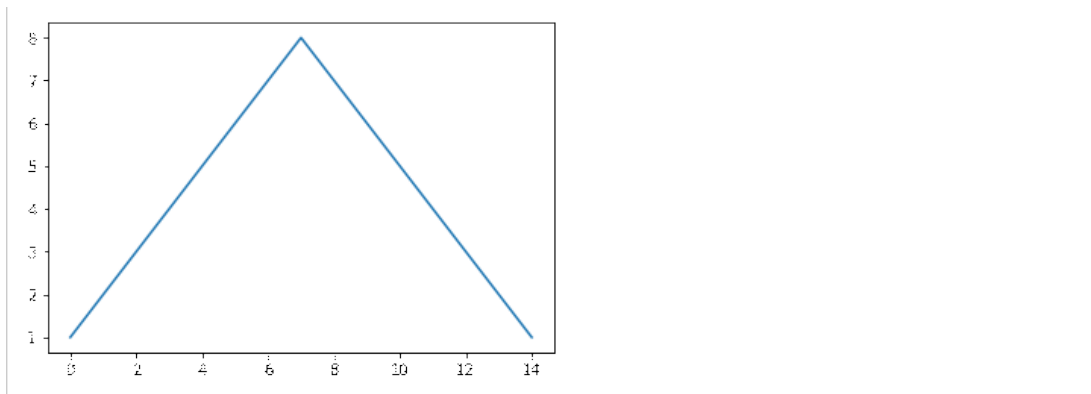
딥러닝 실험에서는 그래프 그리기와 데이터 시각화가 중요하다. matplotlib는 그래프를 그리기 위한 라이브러리이다.

matplotlib를 이용하면 그래프를 그리기가 쉬워진다.

예제(1) 간단 그래프 그려보기

```
import matplotlib.pyplot as plt

plt.figure() # 하나의 화면에 여러개의 그래프를 그릴 때 필요함 (여기선 생략가능)
plt.plot([1,2,3,4,5,6,7,8,7,6,5,4,3,2,1])
plt.show()
```

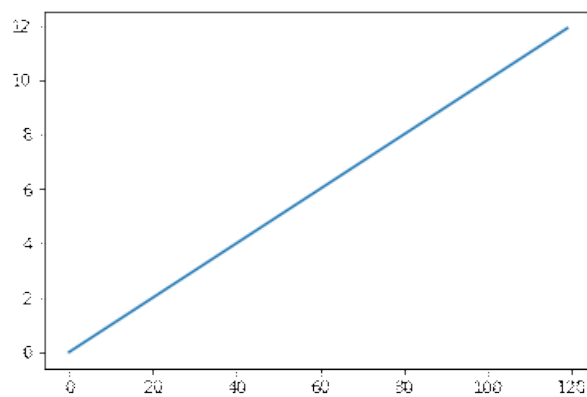


예제(2) 넘파이 배열을 이용해서 그래프 그리기

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

```
t=np.arange(0,12,0.1)
```

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



문제 1. 아래의 행렬을 numpy로 만드시오.

```
1 2
3 4
```

```
import numpy as np
```

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

****결과**

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

문제 2. 위의 행렬 a 각 요소에 5를 더한 값을 출력 하시오.

```
import numpy as np
```

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

```
print(a+5)
```

****결과**

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

문제 3. 아래의 배열 원소들의 평균 값을 출력 하시오.

```
import numpy as np

a = np.array([1,2,4,5,5,7,10,13,18,21])
print('평균 :',np.mean(a))
print('중앙값 :',np.median(a))
print('최대값 :',np.max(a))
print('최소값 :',np.min(a))
print('표준편차 :',np.std(a))
print('분산 :',np.var(a))
```

****결과**

```
평균 : 8.6
중앙값 : 6.0
최대값 : 21
최소값 : 1
표준편차 : 6.437390775772433
분산 : 41.44
```

문제 4. 아래의 행렬식을 numpy로 구현 하시오.

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

```
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)
```

****결과**

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

문제 5. 아래의 그림의 브로드 캐스트를 numpy로 구현 하시오.

```
import numpy as np

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

```
print(a*b)
```

****결과**

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

| | |
|--------------|--|
| 문제 6. | 아래의 행렬식을 numpy로 구현하고 아래의 요소에서 15이상인 것만 출력 하시오. 51 55 14 19 0 4 결과 : [51 55 19] |
|--------------|--|

```
import numpy as np
```

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

****결과**

```
[51 55 19]
```

| | |
|--------------|--|
| 문제 7. | 아래의 행렬식을 numpy로 구현하고 아래의 요소에서 15이상인 것만 출력 하시오. 51 55 14 19 0 4 결과 : [51 55 19] |
|--------------|--|

```
import numpy as np
```

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

##또 다른 방법

```
import numpy as np
```

```
a=np.array([[51,55],[14,19],[0,4]])  
a=a.flatten() # 1차원 배열로 변환
```

```
b=[]  
for i in a:  
    if i > 15:  
        b.append(i)  
print(b)
```

numpy 사용하지 않았을 때

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

```
b=[]  
for i in range(len(a)):
```

```

for j in range(len(a[i])):
    if a[i][j]>=15:
        b.append(a[i][j])
print(b)

```

****결과**

[51 55 19]

문제 8. 아래의 행렬의 합을 numpy를 이용하지 않고 구현해 보시오.

```

a = [[1,3,7],[1,0,0]]
b = [[0,0,5],[7,5,0]]

c=[]
c2=[]
for i in range(len(a)):
    for j in range(len(a[i])):
        c.append(a[i][j]+b[i][j])
    c2.append(c)
    c=[]
print(c2)

```

****결과**

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

문제 9. numpy의 브로드 캐스트를 사용한 연산을 numpy를 이용하지 않는 방법으로 구현 하시오.

```

1 2    10 20
3 4    *
-----> 브로드 캐스트 하면 ?
1 2    10 20    10 40
3 4 * 10 20 = 30 80

```

```

a=[[1,2],[3,4]]
b=[[10,20]]

while 1:
    if (len(a[0])> len(b)):
        b.append(b[0])
    else:
        break

c=[];c2=[]
for i in range(len(a)):
    for j in range(len(a[i])):
        c.append(a[i][j]*b[i][j])
    c2.append(c)
    c=[]
print(c2)

```

****결과**

[[10, 40], [30, 80]]

문제 11. 아래의 브로드 캐스트를 numpy를 이용하지 않고 파이썬으로 구현 하시오.

```
a=[[1,2],[3,4]]
b=[[10]]

for i in range(len(a)):
    if (len(a)> len(b)):
        b.append(b[i])
    elif (len(a[i])>len(b[i])):
        b[i].append(b[i][0])
    else:
        break

#print(b)
c=[];c2=[]
for i in range(len(a)):
    for j in range(len(a[i])):
        c.append(a[i][j]*b[i][j])
    c2.append(c)
    c=[]
print(c2)
```

****결과**

```
[[10, 10], [10, 10]]
[[10, 20], [30, 40]]
```

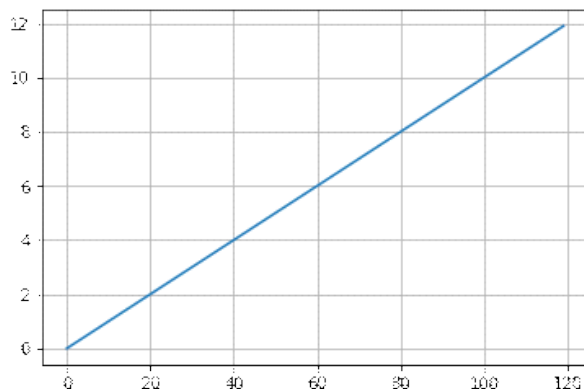
문제 12. 위의 그래프에 grid(격자)를 추가 하시오.

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

t=np.arange(0,12,0.1)
print(t)

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

****결과**

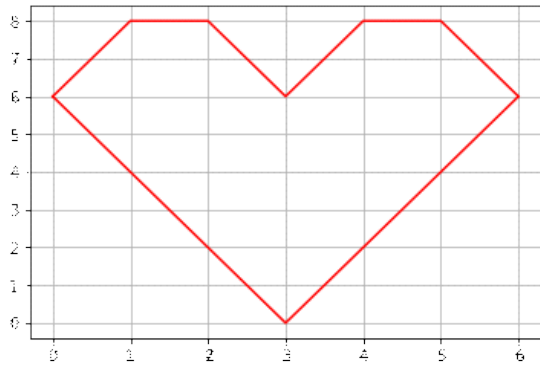


문제 13. 하트모양 그래프를 그리시오.

```
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()
```

****결과**



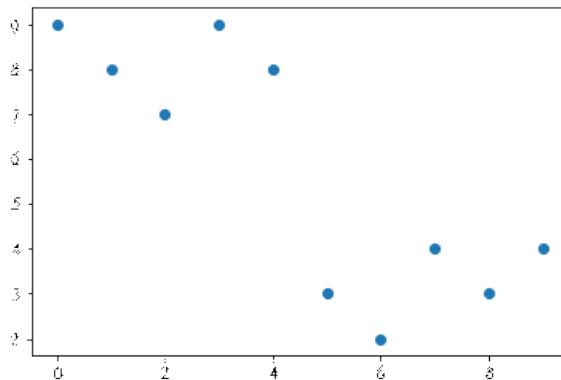
문제 14. 아래의 numpy 배열로 산포도 그래프를 그리시오.

```
import numpy as np
```

```
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()
```

****결과**



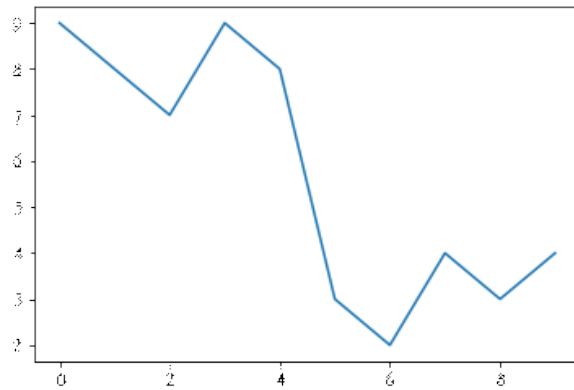
문제 15. 아래의 numpy 배열로 라인 그래프를 그리시오.

```
import numpy as np
```

```
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()
```


****결과**



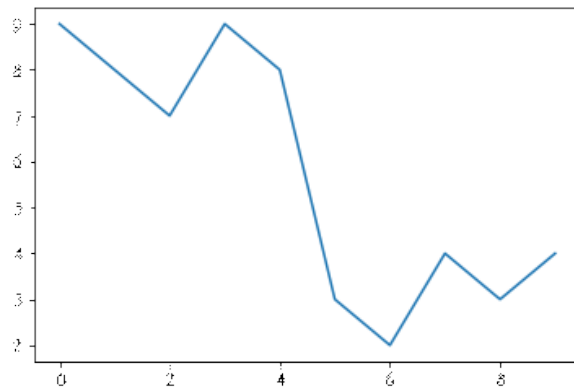
문제 15. 아래의 numpy 배열로 라인 그래프를 그리시오.

```
import numpy as np
```

```
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()
```

****결과**



문제 16. 위의 그래프에 x축의 라벨을 월이라고 하고 y축을 집값으로 라벨을 붙여서 출력 하시오.

```
import numpy as np
```

```
# 한글 폰트 설정
```

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

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

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

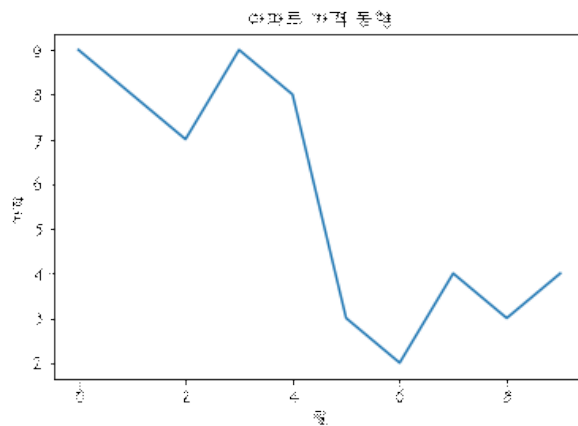
```
plt.xlabel('월')
```

```
plt.ylabel('가격')
```

```
plt.title('아파트 가격 동향')
```

```
plt.show()
```

**결과



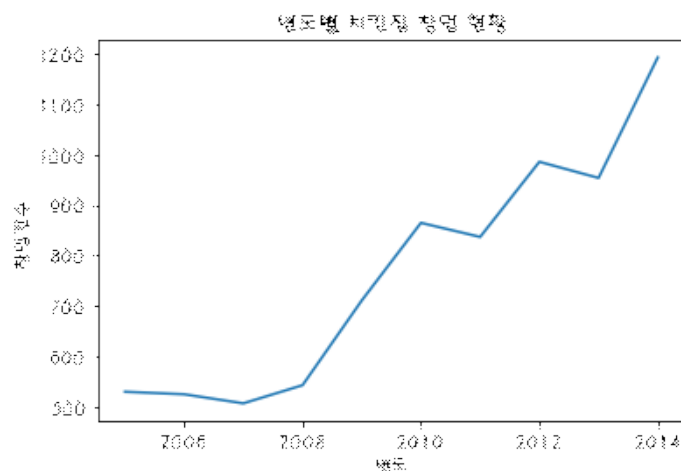
문제 17. 치킨집 년도별 창업 건수를 가지고 라인 그래프를 그리시오.

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

chi = np.loadtxt('C:\\data\\창업건수.csv',skiprows=1,unpack=True,delimiter=',')
#설명 unpack = False면 csv 파일을 그대로 읽어오는 것
#unpack = True 면 csv파일을 pivot해서 읽어오는 것이다.

plt.plot(chi[0],chi[4])
plt.xlabel('년도')
plt.ylabel('창업건수')
plt.title('년도별 치킨집 창업 현황')
plt.show()
```

**결과



문제 18. 치킨 폐업 현황을 위의 그래프에 겹치게 해서 출력 하시오.

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

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

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

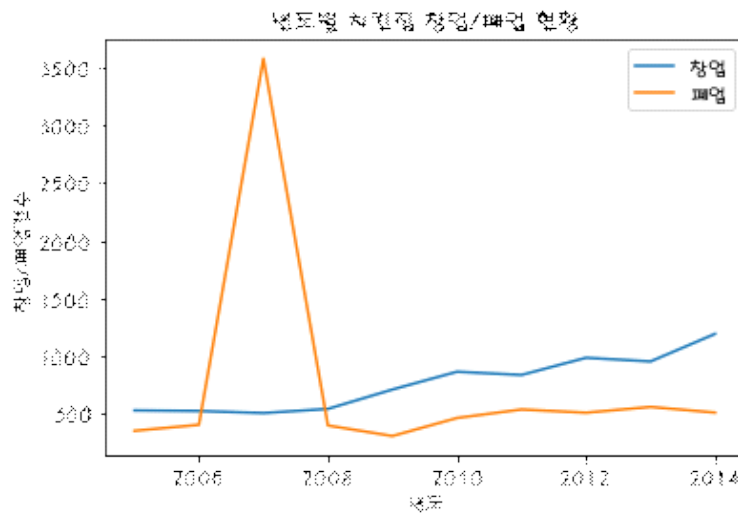
#설명 unpack = False면 csv 파일을 그대로 읽어오는 것

#unpack = True 면 csv파일을 pivot해서 읽어오는 것이다.

```
plt.figure()
plt.plot(chi[0],chi[4],label='창업')
plt.plot(chi2[0],chi2[4],label='폐업')
plt.legend()
plt.xlabel('년도')
plt.ylabel('창업/폐업건수')
plt.title('년도별 치킨집 창업/폐업 현황')

plt.show()
```

**결과

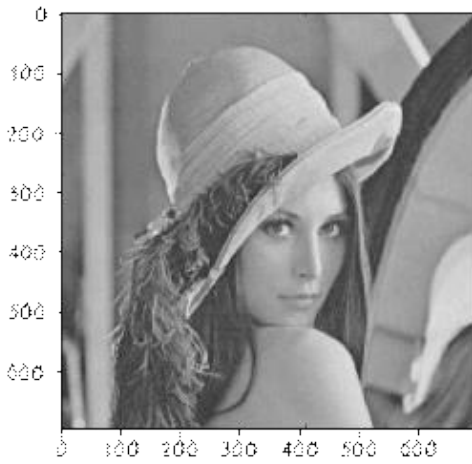


문제 19. 책 44페이지의 이미지 표시를 파이썬으로 구현 하시오.

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

```
img = imread('c:\\data\\lena.png')
plt.imshow(img)
plt.show()
```

**결과

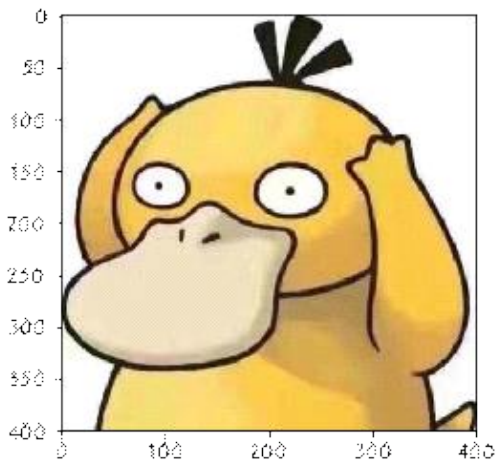


문제 20. jpg 사진을 파이썬에서 출력 하시오.

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

img = imread('c:\\data\\gora.jpg')
plt.imshow(img)
plt.show()
```

****결과**



문제 21. 아래의 4개의 딕셔너리를 이용해서 입력값과 가중치의 곱의 합을 구하는 함수를 x_w_sum 이라는 이름으로 생성 하시오.

```
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}
all_d=[input1_dic,input2_dic,input3_dic,input4_dic]

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

def x_w_sum(input_d, w_d):
    return input_d['x0']*w_d['w0']+input_d['x1']*w_d['w1']+input_d['x2']*w_d['w2']

for i in all_d:
```

```
print(x_w_sum(i,w_dic))
```

****결과**

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

| | |
|--------|---|
| 문제 22. | 위의 결과가 0보다 크거나 같으면 1을 출력하고 0보다 작으면 0을 출력하는 함수를 active_func()로 생성 하시오. |
|--------|---|

```
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}  
all_d=[input1_dic,input2_dic,input3_dic,input4_dic]  
  
w_dic={'w0':0.3,'w1':0.4,'w2':0.1}  
  
def x_w_sum(input_d, w_d):  
    return input_d['x0']*w_d['w0']+input_d['x1']*w_d['w1']+input_d['x2']*w_d['w2']  
  
def active(num):  
    if num >= 0: return 1  
    else : return 0  
  
rst=[]  
for i in all_d:  
    print(x_w_sum(i,w_dic))  
    rst.append(active(x_w_sum(i,w_dic)))  
print(rst)
```

****결과**

```
-0.3  
-0.19999999999999998  
0.10000000000000003  
0.20000000000000004  
[0, 0, 1, 1]
```

| | |
|--------|---|
| 문제 23. | 위의 코드를 이용해서 오차가 없는 가중치를 구하는 코드를 작성 하시오. |
|--------|---|

```
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}  
all_d=[input1_dic,input2_dic,input3_dic,input4_dic]  
  
w_dic={'w0':0.3,'w1':0.4,'w2':0.1}  
  
def x_w_sum(input_d, w_d):  
    return input_d['x0']*w_d['w0']+input_d['x1']*w_d['w1']+input_d['x2']*w_d['w2']
```

```
def active(num):
    if num >= 0: return 1
    else : return 0

rst=[]
for i in all_d:
    print(x_w_sum(i,w_dic))
    rst.append(active(x_w_sum(i,w_dic)))
print(rst)
```

****결과**

```
-0.3
-0.19999999999999998
0.10000000000000003
0.20000000000000004
[0, 0, 1, 1]
```

| | |
|--------|---|
| 문제 24. | (점심시간문제) t -k 의 차이 . 즉, 오차가 있다가 0이 되는 부분이 어디인지 알아내는 라인 그래프를 그리시오. |
|--------|---|

```
import matplotlib.pyplot as plt

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

def active_func2(x, y):
    k = x['x0'] * y['w0'] + x['x1'] * y['w1'] + x['x2'] * y['w2']

    return k

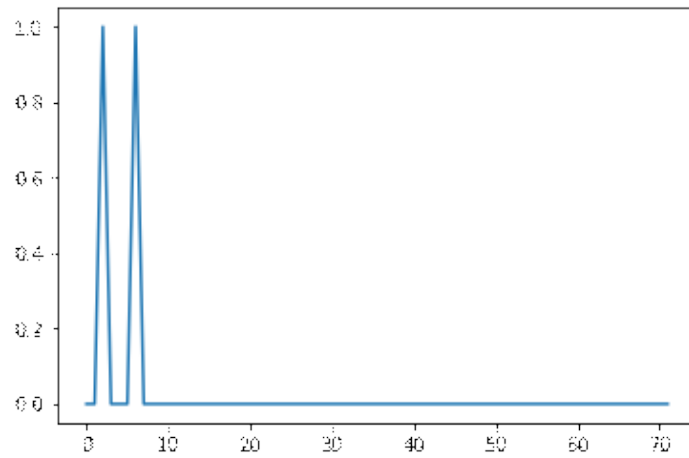
input1_dic = {'x0': -1, 'x1': 0, 'x2': 0, 't': 0}
input2_dic = {'x0': -1, 'x1': 0, 'x2': 1, 't': 0}
input3_dic = {'x0': -1, 'x1': 1, 'x2': 0, 't': 0}
input4_dic = {'x0': -1, 'x1': 1, 'x2': 1, 't': 1}
w_dic = {'w0': 0.3, 'w1': 0.4, 'w2': 0.1}
res = 0

y=[]
for h in range(18):

    for i in range(1, 5): # input1, input2, input3, input4
        input_dic = eval('input%s_dic' % i)
        w_sum = abs(input_dic['t'] - active_func(input_dic, w_dic))
        y.append(w_sum)
    for j in range(3): # w0, w1, w2
        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 )
```

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

**결과

[illegible]

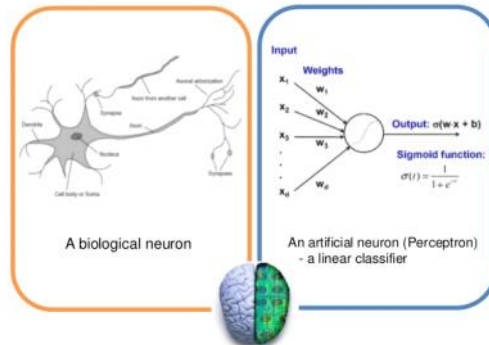
2. 퍼셉트론

2018년 8월 14일 화요일 오후 3:19

2.1 퍼셉트론 기초 개념

퍼셉트론은 다수의 신호(Input)을 입력받아서 하나의 신호(Output)을 출력한다. 이는 뉴런이 전기신호를 내보내 정보를 전달하는 것과 비슷해 보인다. 그리고 뉴런의 수상돌기나 축삭돌기처럼 신호를 전달하는 역할을 퍼셉트론에서는 weight가 그 역할을 한다. 가중치라고 부르는 이 weight는 각각의 입력신호에 부여되어 입력신호와 곱셈을 하고 신호의 총합이 정해진 임계값(θ ; theta, 세타)을 넘었을 때 1을 출력한다. (이를 뉴런의 활성화activation 으로도 표현) 넘지 못하면 0 또는 -1을 출력한다.

Biological neuron and Perceptrons



각 입력신호에는 고유한 weight가 부여되며 weight가 클수록 해당 신호가 중요하다고 볼 수 있다.

여기서 기계학습이 하는 일은 이 weight(입력을 조절하니 매개변수로도 볼 수 있음)의 값을 정하는 작업이라고 할 수 있다. 학습 알고리즘에 따라 방식이 다를 뿐 이 weight를 만들어내는 것이 학습이라는 차원에서는 모두 같다고 할 수 있다.

$$w_1x_1 + w_2x_2 + \dots + w_nx_n > \theta \quad \text{Output} \quad \Rightarrow \quad 1$$

$$w_1x_1 + w_2x_2 + \dots + w_nx_n \leq \theta \quad \Rightarrow \quad 0$$

• 머신러닝의 종류 3가지

| 구분 | 설명 | 해당 알고리즘 |
|-------|---|---|
| 지도학습 | 훈련 데이터와 정답을 가지고 데이터를 분류/예측하는 함수를 만들어내는 기계학습의 한 방법 | 분류 : Knn, 나이브베이지, 결정트리, rule-base 알고리즘, 서포트 벡터머신 회귀 (예측) : 선형회귀, 신경망 |
| 비지도학습 | 정답 없이 훈련 데이터만 가지고 데이터로부터 숨겨진 패턴/규칙을 탐색하는 기계학습의 한 방법 | 클러스터링 : k-means, 연관규칙 (아프리오 알고리즘) |
| 강화학습 | 어떤 환경에서 정의된 에이전트가 현재의 상태를 인식하여 선택 가능한 행동들 중 보상을 최대화 하는 행동 혹은 행동 순서를 선택하는 방법 | |

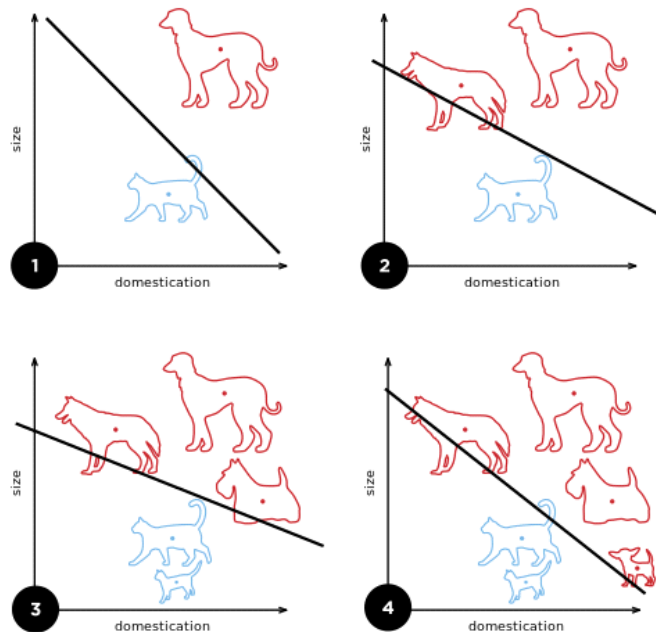
2.2 퍼셉트론 학습방법

처음에는 임의로 설정된 weight로 시작한다.

학습 데이터를 퍼셉트론 모형에 입력하며 분류가 잘못됐을 때 weight를 개선해 나간다.

weight를 개선해 나간다는 의미는 우리가 수학 문제를 잘못 풀었을 때 선생님이 다시 풀어오라고 하면 정답에 맞게 풀기 위해서 다시 풀고 다시 풀고 하다가 정답을 맞추는 것과 비슷하다. 그래서 학습이라고 부른다.

퍼셉트론은 모든 학습 데이터를 정확히 분류시킬 때까지 학습이 진행되기 때문에 학습 데이터가 선형적으로 분리될 수 있을 때 적합한 알고리즘이다. 이와 관련된 퍼셉트론 모형이 가지는 한계는 후에 설명하겠다. 선형분류는 아래와 같이 선으로 분류하는 것을 의미한다. 학습이 반복될수록 선의 기울기가 달라지는 것을 볼 수 있다. 학습을 하면서 weight가 계속 조정(adjust)되는 것이다.



2.3 가중치와 편향

앞의 퍼셉트론 수식에서 나오는 세타 θ 를 $-b$ 로 치환하여 좌변으로 넘기면

$$b + w_1x_1 + w_2x_2 < 0 \Rightarrow 0$$

$$b + w_1x_1 + w_2x_2 \geq 0 \Rightarrow 1$$

과 같이 되며 여기에서 b 를 **편향(bias)**라고 할 수 있다. 기계학습 분야에서는 모델이 학습 데이터에 **과적합(overfitting)**되는 것을 방지하는 것이 중요하다. 여기서 과적합이라 함은 모델이 엄청 유연해서 학습 데이터는 귀신같이 잘 분류하지만, 다른 데이터를 넣어봤을 때는 제대로 성능을 발휘하지 못하는 것을 말한다. 어느 데이터를 넣어도 일반적으로 잘 들어맞는 모델을 만드는 것이 중요하다.

따라서 앞에서 설명했듯이 편향은 θ (theta)로 학습 데이터(input)이 가중치와 계산되어 넘어야 하는 임계점으로 이 값이 높으면 높을 수록 그만큼 분류의 기준이 엄격하다는 것을 의미한다. 그래서 편향이 높을 수록 모델이 간단해지는 경향이 있으며 (변수가 적고 더 일반화 되는 경우) 오히려 **과소적합(underfitting)**의 위험이 발생하게 된다. 반대로 편향이 낮을수록 한계점이 낮아 데이터의 허용범위가 넓어지는 만큼 학습 데이터에만 잘 들어맞는 모델이 만들어질 수 있으며 모델이 더욱 복잡해질 것이다. 허용범위가 넓어지는 만큼 필요 없는 노이즈가 포함될 가능성도 높다.

가중치(weight)는 입력신호가 결과 출력에 주는 영향도를 조절하는 매개변수이고, **편향(bias)**은 뉴런(또는 노드; x 를 의미)이 얼마나 쉽게 **활성화**(1로 출력; activation)되느냐를 조정하는(adjust) 매개변수이다.

2.4 퍼셉트론의 한계점

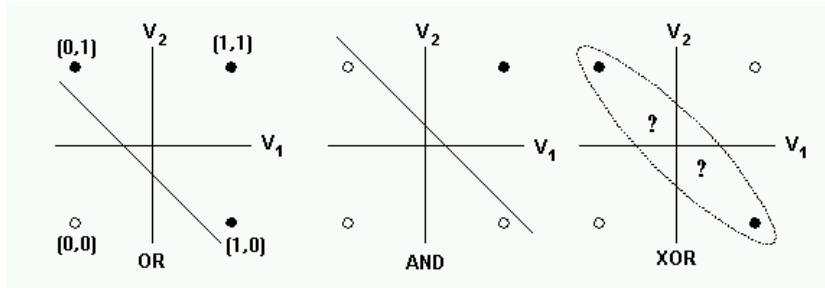
퍼셉트론이 인공지능 분야에서 센세이션을 불러일으켰고 연구 과제도 이쪽으로 몰렸으나 이것이 가지는 한계점이 밝혀지면서 한동안 소외 받는 이론이 되었다. 퍼셉트론의 한계는 선형으로 분류를 할 수 있지만 XOR와 같이 선형 분류만 가능하며 비선형 분류는 불가능하다는 점이다.

XOR 논리는 exclusive(배타적) 논리연산이다. 아래의 진리표를 보면, x_1 과 x_2 중 어느 한쪽이 1일 때만 1을 출력한다.

| x_1 | x_2 | y |
|-------|-------|-----|
| 0 | 0 | 0 |

| | | |
|---|---|---|
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 1 | 0 |

아래의 그림을 보면 XOR에서는 선형으로(직선 하나로) 분류가 불가능함을 알 수 있습니다.

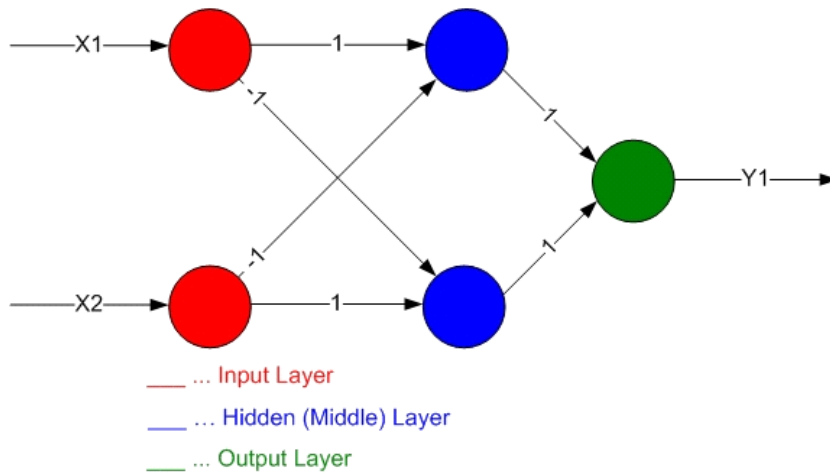


퍼셉트론의 한계를 간략히 말하면, 직선 하나로 나눈 영역만 표현할 수 있어 XOR과 같은 데이터 형태는 분류가 불가능하다는 한계가 있다.

2.5 다층 퍼셉트론을 통한 한계 극복

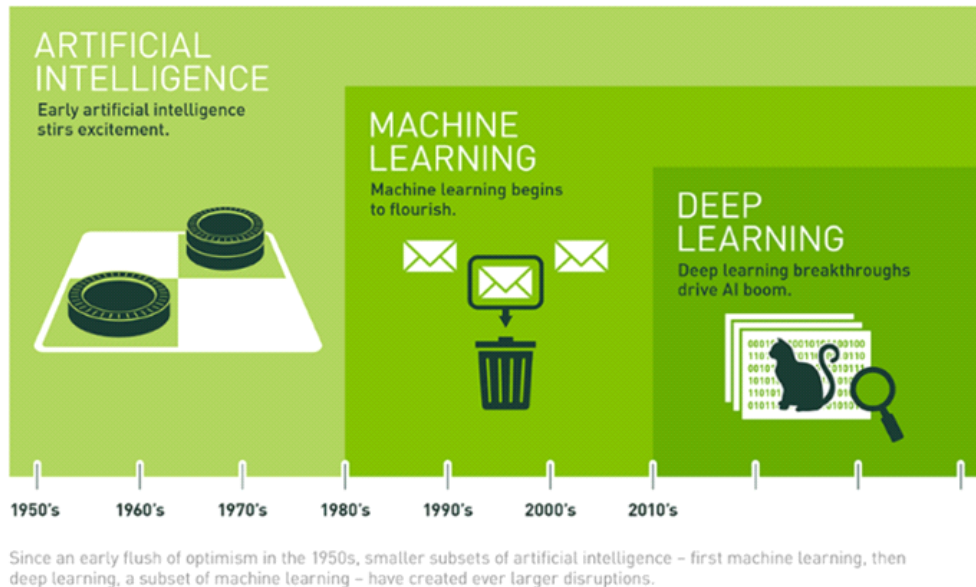
단일 퍼셉트론으로는 XOR을 분류할 수 없지만, 다층 퍼셉트론을 만들면 이를 극복할 수 있습니다. 다층(multi-layer)이라는 말은 하나의 퍼셉트론에 또 다른 퍼셉트론을 덧붙인다는 의미로 볼 수 있다. 단층 퍼셉트론이 비선형 영역을 분리할 수 없다는 것이 문제이며 다층으로 할 경우 비선형으로 이를 해결할 수 있다.

$$Y1 = \text{XOR}(X1, X2)$$



이런식으로 층을 겹겹이 쌓아나가면서 선형 분류만으로는 풀지 못했던 문제를 비선형적으로 풀 수 있게 된다.

2.6 수업 필기



--> 머신러닝의 한 분류인 딥러닝 (이미지를 기계가 직접 인식해 학습)

퍼셉트론 -> 신경망

인간의 뇌 세포중에 하나를 컴퓨터로 구현해 봄 (뉴런)

1943년에 미국의 신경외과 의사인 워런 맥컬록에 의해서 발단이 되었음.

1957년에 프랑크 로젠 블라트가 퍼셉트론 알고리즘을 고안했다.

사람의 뇌의 동작을 전기 스위치의 온/오프로 흉내 낼 수 있다는 이론을 증명 했다.

즉, 인간의 신경세포 하나를 흉내 냈는데

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

특정 자극이 있다면 그 자극이 어느 역치 이상일 경우만 세포가 반응한다.

ex. 짜게 먹는 사람은 자기가 평소에 먹는 만큼 음식이 짜지 않으면 싱겁다고 느낀다. (역치 이하의 자극은 무시)

싱겁게 먹는 사람이 짜게 먹기 시작하면, 오랜 시간이 지난 후 예전에 먹던 싱거운 음식에 만족하지 못한다 (역치가 올라감)

퍼셉트론은 자극에 반응하는 뉴런을 하나 만들.

뉴런의 개수

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

■ 퍼셉트론 역사

1. 1957년 : 프랑크 블라트가 퍼셉트론 알고리즘을 고안했다.
2. 1969년 : 퍼셉트론은 단순 선형분류기에 불과하다. wh? xor 분류도 못한다고 단정을 지음 (침체기에 빠짐)
3. 1970년대 중반 : 중요한 논문이 발표 (역전파 알고리즘 =다층 퍼셉트론)

당시의 컴퓨터 연산으로는 이 이론을 구현하기가 어려웠다

4. 1986년 : 은닉층을 갖는 다층 퍼셉트론 + 오류 역전파 학습 알고리즘 구현
5. 오늘날 -- 신경망을 통해서 구현하고자 하는 목표 ? 분류

- 불꽃사진을 보고 불꽃의 종을 분류한다.

입력신호가 뉴런에 보내질때는 각각의 고유한 가중치가 곱해진다.

$w_1 \cdot x_1 + w_2 \cdot x_2 \leq \text{임계값} \rightarrow 0$ (신호가 안흐른다.)

$w_1 \cdot x_1 + w_2 \cdot x_2 > \text{임계값} \rightarrow 1$ (신호가 흐른다.)

뉴런에서 보내온 신호의 총합이 정해진 합계(임계값)을 넘어 설 때만 1을 출력한다.

퍼셉트론은 n개의 이진수가 하나의 뉴런을 통과해서 가중의 합이 0보다 크면 활성화 되는 간단한 신경망이다.

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

■ 단층 신경망과 다층 신경망의 차이

1958년 로젠블라트가 퍼셉트론을 제안했다. 1959년 민스키가 기존 퍼셉트론의 문제점을 지적했는데, XOR 분류를 못한다는 점이었고 이로 인해 인공지능의 겨울기가 시작되었다.

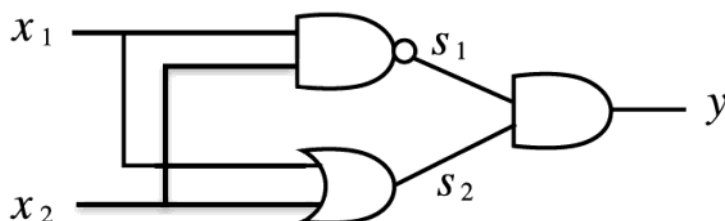
XOR 게이트는 단층 신경망으로 구현이 안되서 다층 신경망으로 구현해야 하는 것이었다.

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

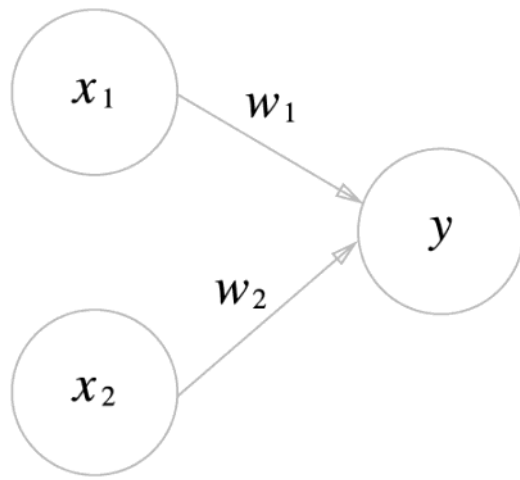
- xor

| x1 | x2 | t |
|----|----|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

| x1 | x2 | or 게이트 | nand게이트 | or & nand |
|----|----|--------|---------|-----------|
| 0 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 1 | 1 |
| 0 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 0 | 0 |



문제 26. 아래 그림을 파이썬으로 구현하시오.



```
import numpy as np
```

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

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

```
print(x)
```

```
print(w)
```

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

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

****결과**

```
[0 1]
```

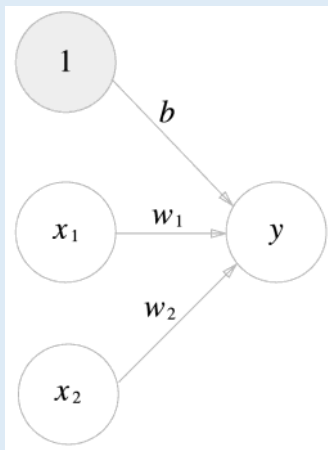
```
[0.5 0.5]
```

```
[0. 0.5]
```

```
0.5
```

문제 27.

위의 식에 책 52쪽에 나오는 편향을 더하여 완성한 아래의 식을 파이썬으로 구현 하시오.



$b(\text{편향}) = -0.7$

$x_1 : 0, x_2 : 1$

$w_1 : 0.5, w_2 : 0.5$

```
import numpy as np
```

```
b=np.array([-0.7])
```

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

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

```
print(x)
```

```
print(w)
```

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

****결과**

```
[0 1]
[0.5 0.5]
[0. 0.5]
[-0.2]
```

문제 28. and 게이트를 파이썬으로 구현 하시오.
(파이썬 함수를 생성하는데 함수 이름은 AND 이다)

| x_1 | x_2 | y |
|-------|-------|-----|
| 0 | 0 | 0 |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 1 | 1 |

```
import numpy as np
```

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

```
def and_func(n1,n2):
    if n1*w[0]+ n2*w[1] <= 0.7:
        return 0
    else : return 1
```

```
print(and_func(0,0))
print(and_func(0,1))
print(and_func(1,0))
print(and_func(1,1))
```

****결과**

```
0
0
0
1
```

문제 29. 위에서 만든 AND 퍼셉트론 함수를 이용해서 아래의 INPUT DATA를 이용해서 출력 결과를 for loop문으로 한번에 출력 하시오.

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

```
import numpy as np
```

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

```
def and_func(n1,n2):
    if n1*w[0]+ n2*w[1] <= 0.7:
```

```

        return 0
    else : return 1

inputData=np.array([[0,0],[0,1],[1,0],[1,1]])
print('-----AND 퍼셉트론-----')
for i,j in inputData:
    print(i,j,'==>' ,and_func(i,j))

**결과
-----AND 퍼셉트론-----
0 0 ==> 0
0 1 ==> 0
1 0 ==> 0
1 1 ==> 1

```

문제 30. 문제 29번 코드를 가지고 좀 수정해서 OR 게이트 함수를 생성해서 아래와 같이 출력 하시오.

```

import numpy as np

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

def OR_func(n1,n2):
    if n1*w[0]+ n2*w[1] <= 0:
        return 0
    else : return 1

inputData=np.array([[0,0],[0,1],[1,0],[1,1]])
print('-----OR 퍼셉트론-----')
for i,j in inputData:
    print(i,j,'==>' ,OR_func(i,j))

**결과
-----OR 퍼셉트론-----
0 0 ==> 0
0 1 ==> 1
1 0 ==> 1
1 1 ==> 1

```

문제 31. 문제 30번 코드를 가지고 좀 수정해서 NAND 게이트 함수를 생성해서 아래와 같이 출력 하시오.

```

import numpy as np

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

def NAND_func(n1,n2):
    if n1*w[0]+ n2*w[1] <= -0.7:
        return 0
    else : return 1

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

**결과

```

-----NAND 퍼셉트론-----

0 0 ==> 1

0 1 ==> 1

1 0 ==> 1

1 1 ==> 0

문제 32. xor 을 함수로 생성 하시오.

```
import numpy as np
```

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

```
def OR_func(n1,n2):
```

```
    if n1*w[0]+ n2*w[1] <= -0.7:
```

```
        return 0
```

```
    else : return 1
```

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

```
print('-----NAND 퍼셉트론-----')
```

```
for i,j in inputData:
```

```
    print(i,j,'==>' ,OR_func(i,j))
```

****결과**

-----NAND 퍼셉트론-----

0 0 ==> 1

0 1 ==> 1

1 0 ==> 1

1 1 ==> 0

3. 신경망

2018년 8월 16일 목요일 오후 4:21

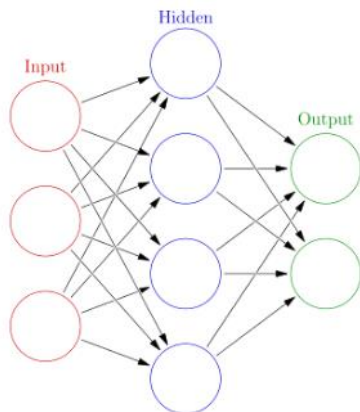
3.1 신경망과 퍼셉트론 비교

퍼셉트론이 하나의 뉴런 단위로 다루어진다면 각 뉴런이 모여 하나의 뇌가 되는 것과 같은 신경망은 퍼셉트론의 하위요소일 것이다. 단층 퍼셉트론이 하나의 나무와 같이 신경망은 나무가 모여 숲을 이룬 것과 같은 느낌으로 이해하면 된다.

퍼셉트론이라는 말을 어떻게 정의하느냐에 따라 가리키는 것이 달라진다. 일반적으로는 단층 퍼셉트론은 step function(임계값을 넘었었을 때 출력을 1로 하는 함수)을 활성화 함수로 사용한 모델을 가리킨다. 다층 퍼셉트론은 층이 여러개이며 sigmoid function을 활성화함수로 사용하는 네트워크를 가리킨다.

| 구분 | 층수 | 활성화 함수 |
|---------|----|---|
| 단층 퍼셉트론 | 단층 | 임계값을 넘었을 때 1을 출력하는 step function 활성화 함수 사용 |
| 다층 퍼셉트론 | 다층 | Sigmoid 활성화 함수 사용 |

신경망은 아래와 같이 왼쪽부터 Input(입력층), Hidden(은닉층), Output(출력층)으로 표현할 수 있다. 은닉층은 양쪽의 입력층과 출력층과는 달리 우리 눈에는 보이지 않기 때문에 (내부적으로 돌아가고 있는 Black Box와도 같다. 모르기 때문에) 'Hidden(은닉)'이라고 한다. 퍼셉트론과는 크게 달라 보이지 않는다.



*퍼셉트론과 신경망의 차이점 ?

1. 퍼셉트론 ?

원하는 결과를 출력하도록 **사람이 직접 가중치를 정해줘야 한다.**

ex. w_1, w_2 , 임계값을 사람이 정해 주었다.

2. 신경망?

가중치 매개변수의 적절한 값을 **기계가 데이터로 부터 학습해서 자동으로 알아낸다.**

ex. 오전에 만들었던 알고리즘문제 12번

3.2 활성화 함수(Activation Function)

- 활성화 함수란? : 신호의 총합을 받아서 다음 신호로 내보낼지 말지를 결정하는 함수

■ 신경망에서 사용되는 활성화 함수

| 구분 | 설명 | 함수 구현 |
|----------|--|---|
| 계단 함수 | 0 또는 1의 값을 출력하는 함수 ex. 고양이 사진 --> 계단함수--> 1 or 0 (맞다 아니다.) | <pre>def step_func(x): y=x>0 return y.astype(np.int)</pre> |
| 시그모이드 함수 | 0~1 사이의 실수(확률)를 출력하는 함수 $h(x) = \frac{1}{1 + \exp(-x)}$ ex. 고양이 사진 --> 시그모이드 --> 0.21 (21%확률로 고양이 사진이다) | <pre>import numpy as np def sigmoid(x): return 1/(1+np.exp(-x))</pre> |
| RELU 함수 | 입력이 0을 넘으면 그 입력 값을 그대로 출력하는 함수, 0 이하의 값은 0을 출력. | <pre>def Relu(x): return np.maximum(x,0)</pre> |

■ 시그모이드 함수 식이 아래와 같이 나온 이유

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

통계학에서 성공할 확률이 실패할 확률보다 얼마나 큰지를 나타내는 오즈 비율이라는 값이 있다.

오즈비율 = 성공 / 실패 = $p / (1-p)$

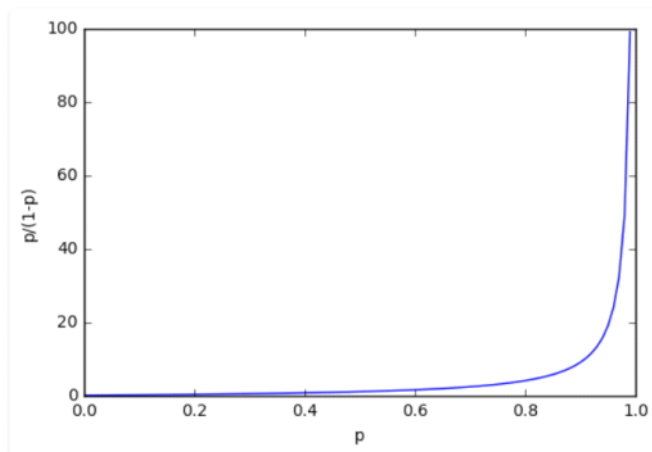


그림 7. $\left(\frac{p}{1-p}\right)$ 그래프

성공할 확률 p 를 0에서 1사이의 값으로 나타내면 실패할 확률은 $1-p$ 이다.

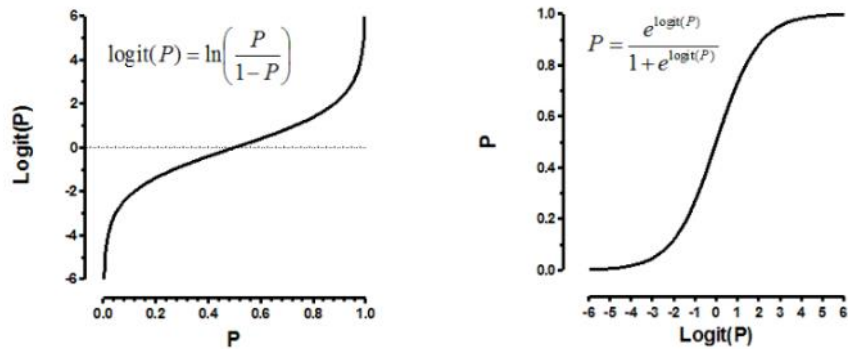
위의 그래프에서 p (성공할 확률)가 1에 가까우면 오즈 비율(성공할 확률이 실패할 확률보다 얼마나 큰지 나타낸 값)이 급격히 커져버리는 현상이 발생한다.

----> 급격히 오즈비율이 커져버리는 현상을 방지하기 위해서 이 함수에 log를 취한게 logit 함수이다.

p 가 0.5일때는 0이 되고 p 가 1일때는 무한히 큰 양수, p 가 0일때는 무한히 큰 음수를 가지는 특성이 있다.

$$\log(p / (1-p)) = wx + b$$

이 로그함수를 뉴런의 출력값에 따라 확률 p 를 구하기 쉽도록 지수함수 형태로 바꾸면 ,



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

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

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

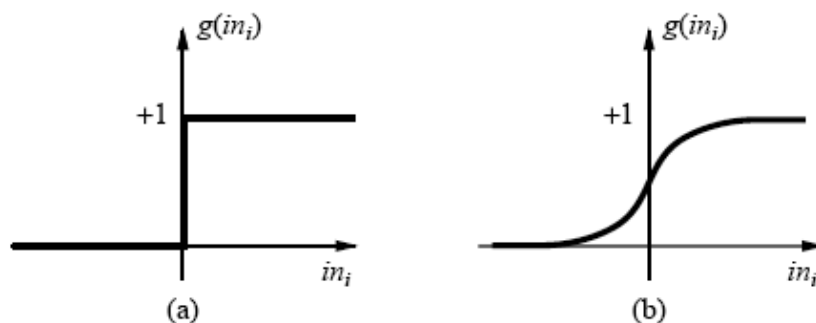
$$P = \frac{1}{1 + e^{-(wx+b)}} = \frac{1}{1 + e^{-x}}$$

$$p = 1 / (1 + e^{-(wx+b)})$$

$$= 1 / (1 + e^{-x})$$

a. 계단함수 step function

퍼셉트론은 활성화 함수로 step function(계단 함수)를 이용한다. 특정 임계값을 넘기면 활성화되는 함수이다. 아래 왼쪽 (a)가 계단 함수이다. 0에서 멈추어있다. 어느 기점에서 1로 바뀐다.



b. 시그모이드 함수 sigmoid function

앞서 말했듯이 신경망에서 주로 이용하는 활성화 함수는 시그모이드 함수이다. 아래는 시그모이드 함수를 나타낸 식이다. e 는 자연상수로 2.7192...의 값을 갖는 실수 이다. 계단함수에 비해 완만한 곡선 형태로 비선형이다. 특정 경계를 기준으로 출력이 확 바뀌어버리는 계단함수와는 달리 시그모이드 함수는 완만하게 매끄럽게 변화하는데 이 매끄러움이 신경망 학습에서 중요하며 활성화 함수로 시그모이드 함수를 사용하는 이유이기도 하다.

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

시그모이드 함수는 값을 실수형으로 가지는 것을 볼 수 있다. 시그모이드 함수의 매끄러움은 가중치 값을 전달할 때 좀 더 부드럽게 양을 조절해서 전달할 수 있다는 점이 계단 함수와 다른 점이다.

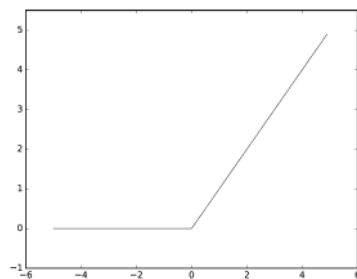
둘다 비선형인 점은 동일하다.

c. Relu (Rectified Linear Unit) 함수 // Rectified : 정류된

Relu는 입력이 0을 넘으면 그 입력을 그대로 출력하고 0 이하면 0을 출력하는 함수

신경망 학습이 잘되어서 현업에서 주로 사용하는 함수

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



** 그렇다면 왜 비선형 함수를 사용해야 하는가?

선형함수를 사용했을 때는 은닉층을 사용하는 이점이 없기 때문이다. 다시 말해 선형함수를 여러층으로 구성한다 하더라도 이는 선형함수를 세번 연속 반복한 것에 지나지 않는다는 의미와 같기 때문이다. $y = ax$ 라는 선형함수가 있다고 한다면 이 것을 3층으로 구성하면 $y = a(a(a(x)))$ 와 동일한 것으로 이는 $y = a^3(x)$ 와 같다. 굳이 은닉층 없이 선형함수로 네트워크를 구성하는 것은 의미가 없다는 뜻입니다.

3.3 출력층 함수

신경망은 **분류(classification)**와 **회귀(regression)** 문제에 모두 활용될 수 있다. 어떤 문제냐에 따라 활성화 함수가 달라질 뿐이다. 분류는 어떤 사람이 사기를 쳤는지(1), 안 쳤는지(0) 예측하는 것이고, 회귀는 사기당한 금액이 얼마(\$10,000)였는지 예측하는 문제이다. 둘다 크게 보면 **예측(prediction)**이다.

어떤 상황에 어떤 활성화 함수를 사용해야 하는가

출력 부분에서의 활성화 함수는 문제 상황에 따라 다를 것이다. 일반적으로

회귀 --> 항등 함수 (출력 값을 그대로 반환하는 함수) identity function

분류(0/1) --> 시그모이드 함수 sigmoid function

분류(multiple) --> 소프트맥스 함수 softmax function

소프트맥스 함수(Softmax Function)는 아주아주 중요한 개념이므로 다른 포스팅에서 조금 더 설명해보겠다. 0~1의 실수 값을 출력하는 소프트맥스는 그 성질 때문에 분류하는 결과에 대한 확률값으로 해석되기도 하여 분류 문제를 확률적으로 풀어볼 수


```

# 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)

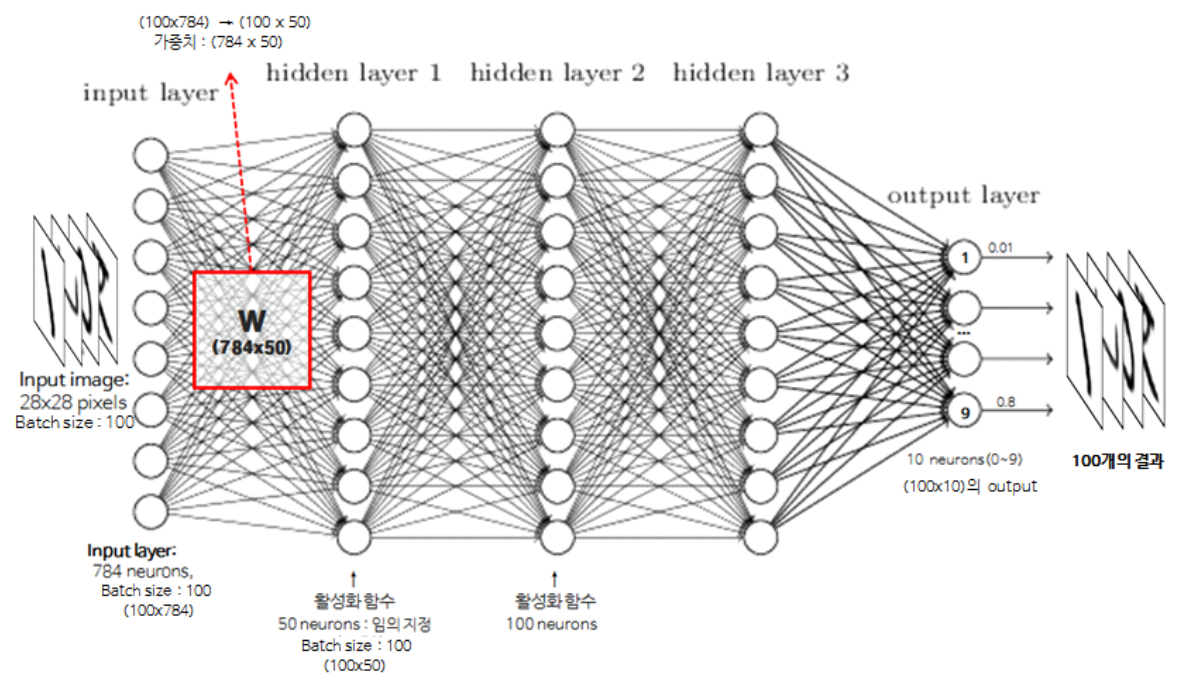
img = x_train[0]
label = t_train[0]
print(label) # 5

print(img.shape) # (784,)
img = img.reshape(28, 28)

img_show(img)

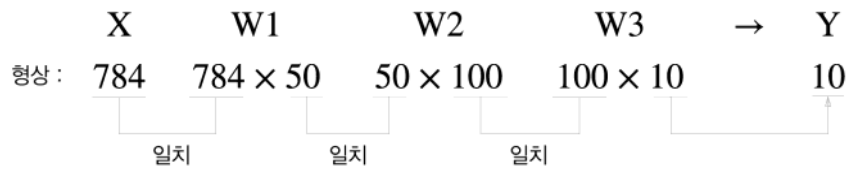
```

■ 신경망에 데이터 입력 시 배치(batch)로 입력하는 방법

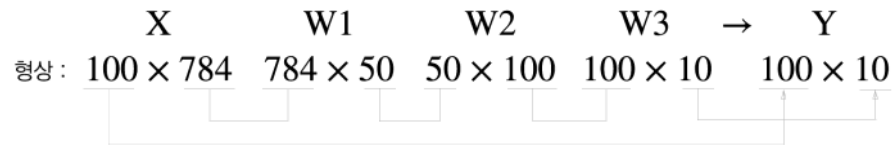


배치처리 : 이미지를 한 장씩 처리하는 것이 아니라 여러장을 한 번에 처리.

**이미지를 한 장씩 처리할 때



**이미지를 한꺼번에 여러장 처리할 때



예제(1) :

```
import numpy as np
print(np.exp(1)) #2.71828
print(np.exp(100)) #2.6881171418161356e+43
```

****결과**

```
2.718281828459045
2.6881171418161356e+43
```

문제 32. 파이썬으로 계단 함수를 만드시오.

```
def step_func(data):
    rst=[]
    for i in data:
        if i > 0 :
            rst.append(1)
        else:
            rst.append(0)
    return rst
x_data=np.array([-1,0,1])

print(step_func(x_data))

##다른 방법

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

y=x_data > 0
#print(y) # False False True
print(y.astype(np.int))

**결과
[0, 0, 1]
```

문제 33. 계단함수를 파이썬으로 시각화 하시오.

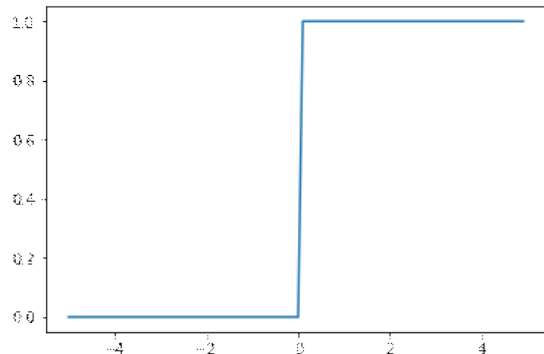
```
x=np.arange(-5,5,0.1)
```

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

```
y=step_func(x)
```

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

****결과**



문제 34. a.csv 에 있는 데이터가 삼성전자, 엘지전자 처럼 회사명이 들어있고 b.csv엔 삼성전자윤진민, 엘지전자백광흠, 등의 회사명+직원명이 들어가 있다. b.csv에 a.csv의 회사명이 있을 경우 그 회사명만 모두 출력 하시오.

```
import pandas as pd
c_name=pd.read_csv('company_name.txt',sep='\\t')
emp=pd.read_csv('emp_plus_company.txt',sep='\\t')
l=[]
for i in c_name['name']:
    for j in emp['text']:
        if i in j:
            l.append(i)
print(pd.Series(l).unique())
```

****결과**

['현대자동차', '삼성전기', '엘지전자', '삼성전자']

문제 35. 시그모이드 함수를 파이썬으로 구현 하시오.

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

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

****결과**

[0.73105858 0.88079708]

문제 36. 시그모이드 함수의 그래프를 그리시오.

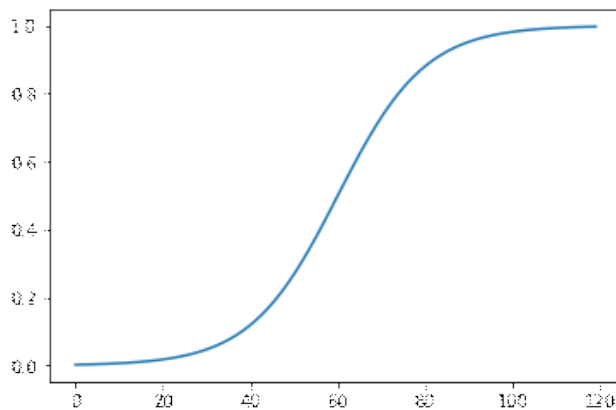
```
import numpy as np
import matplotlib.pyplot as plt
```

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

```
x=np.arange(-6,6,0.1)
```

```
plt.plot(sigmoid(x))
```

****결과**



문제 37. Relu 함수를 생성 하시오.

```
def Relu(x):
    return max(x*1,0)
```

```
print(Relu(3))
```

****결과**

3

문제 38. numpy를 이용해서 Relu 함수를 생성 하시오.

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

```
y=np.array([1,2,3,-4])
print(Relu(y))
```

****결과**

[1 2 3 0]

문제 39. Relu 함수의 그래프를 그리시오.

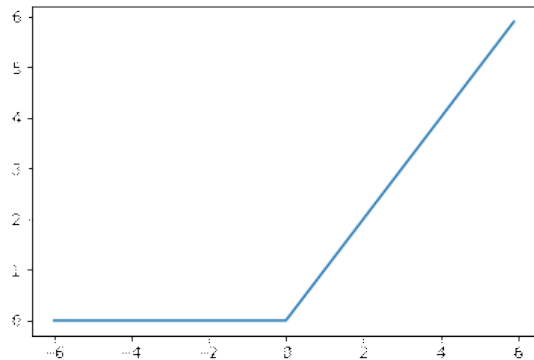
```
def Relu(x):
```

```
return np.maximum(x,0)
```

```
x=np.arange(-6,6,0.1)
```

```
plt.plot(x,Relu(x))
```

****결과**



#행렬의 내적

문제 40. 아래의 두개의 행렬의 내적을 numpy로 구현 하시오.

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix} = \begin{pmatrix} 19 & 22 \\ 43 & 50 \end{pmatrix}$$

A B

$1 \times 5 + 2 \times 7$ $3 \times 5 + 4 \times 7$

```
import numpy as np
```

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

```
b = np.array([[5,6],[7,8]])
```

```
print(np.dot(a,b))
```

****결과**

```
[[19 22]
```

```
 [43 50]]
```

문제 41. 아래의 행렬곱(행렬내적)을 파이썬으로 구현하시오.

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

```
import numpy as np
```

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

```
b = np.array([[5,6],[7,8],[9,10]])
```

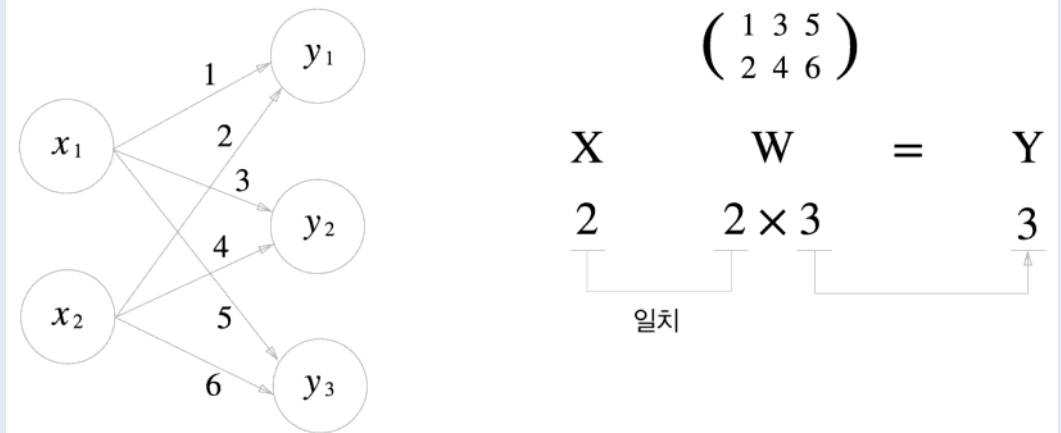
```
print(np.dot(a,b))
```

****결과**

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

#신경망의 내적

문제 42. 위의 신경망의 x_1, x_2 가 1과 2이면 y_1, y_2 가 무엇이 되는가?



$$x_1 \cdot 1 + x_2 \cdot 2 = y_1$$

$$x_1 \cdot 3 + x_2 \cdot 4 = y_2$$

$$x_1 \cdot 5 + x_2 \cdot 6 = y_3$$

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

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

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

```
def func(x,w):
    return np.dot(x,w)
```

```
print(func(input_d,weight_d))
```

****결과**

```
[ 5 11 17]
```

문제 43. 위의 신경망에서 만들어진 가중의 총합인 y 값을 시그모이드 함수를 통과 시켜서 나온 y_{hat} 을 출력 하시오.

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

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

```
def func(x,w):
    return np.dot(x,w)
```

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

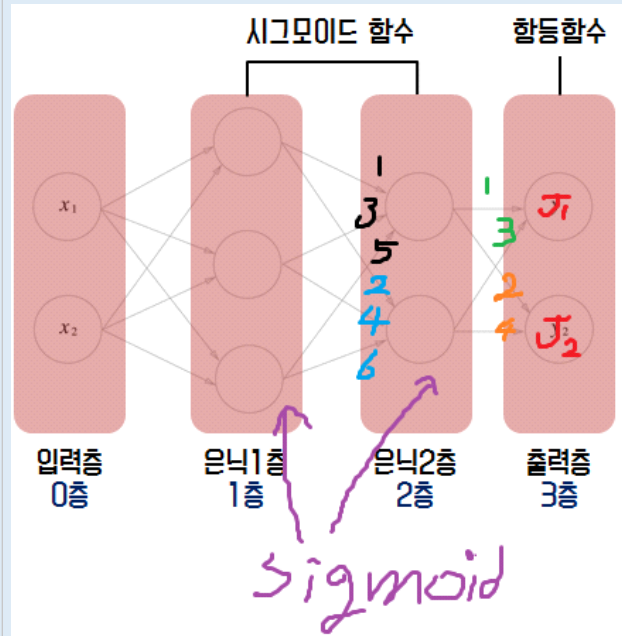
```
print(sigmoid(func(input_d,weight_d)))
```

****결과**

```
[0.99330715 0.9999833 0.99999996]
```

문제 44.

3층 신경망을 구현하고 J1, J2 값을 구하시오.



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

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

```
w2_d=np.array([[1,2],[3,4],[5,6]])
```

```
w3_d=np.array([[1,2],[3,4]])
```

```
def func(x,w):
```

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

```
def sigmoid(x):
```

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

```
print(func(sigmoid(func(sigmoid(func(input_d,weight_d)),w2_d)),w3_d))
```

****결과**

```
[3.99985709 5.99972663]
```

문제 45.

아래의 입력값과 가중치를 이용하여 신경망 1층에서 나온 출력값인 [9 13 17]을 출력 하시오.

```
import numpy as np
```

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

```
x = np.matrix([[1,2]])
```

```
w1 = np.matrix([[1,3,5], [4,5,6]])
```

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

```
w3 = np.matrix([[1,3],[2,4]])
```

#1층

```
y1= np.dot(x,w1)
print(y1)
```

****결과**

```
[[ 9 13 17]]
```

문제 46.

1층의 시그모이드 함수를 통과한 결과를 y_hat 변수에 담아 출력 하시오.

```
import numpy as np
from matplotlib import font_manager, rc
```

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

#1층

```
y1= np.dot(x,w1)
print(y1)
```

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

```
y_hat=sigmoid(y1)
print(y_hat)
```

****결과**

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

문제 47.

2층의 시그모이드 함수 통과하기 전 y2 값을 출력 하시오.

```
import numpy as np
from matplotlib import font_manager, rc
```

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

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

#1층

```

y1= np.dot(x,w1)
y1_hat=sigmoid(y1)

#2층
y2 = np.dot(y1_hat,w2)
print(y2)

**결과
[[ 8.99986962 11.99974392]]

```

문제 48. 2층에서 시그모이드 함수 통과한 후의 y2_hat을 출력 하시오.

```

import numpy as np
from matplotlib import font_manager, rc

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

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

#1층
y1= np.dot(x,w1)
y1_hat=sigmoid(y1)

#2층
y2 = np.dot(y1_hat,w2)
y2_hat=sigmoid(y2)
print(y2_hat)

**결과
[[0.99987659 0.99999385]]

```

문제 49. 출력층(3층)의 항등함수에 값이 입력되기 전의 y3 값을 출력 하시오.

```

import numpy as np
from matplotlib import font_manager, rc

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

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

```

```

#1층
y1= np.dot(x,w1)
y1_hat=sigmoid(y1)

#2층
y2 = np.dot(y1_hat,w2)
y2_hat=sigmoid(y2)

#3층
y3=np.dot(y2_hat,w3)
print(y3)

**결과
[[2.9998643  6.99960519]]

```

문제 50. 입력값을 받아 그대로 출력하는 항등함수를 identity_function 이라는 이름으로 생성 하시오.

```

import numpy as np
from matplotlib import font_manager, rc

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

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

def identity_function(x):
    return x

#1층
y1= np.dot(x,w1)
y1_hat=sigmoid(y1)

#2층
y2 = np.dot(y1_hat,w2)
y2_hat=sigmoid(y2)

#3층
y3=np.dot(y2_hat,w3)
print(y3)

**결과
[[2.9998643  6.99960519]]

```

문제 51. 위에서 만든 항등함수를 통과한 결과인 y3_hat의 결과를 출력하시오. (3층 신경망 전체 코드)

```

import numpy as np
from matplotlib import font_manager, rc

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

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

def identity_function(x):
    return x

#1층
y1= np.dot(x,w1)
y1_hat=sigmoid(y1)

#2층
y2 = np.dot(y1_hat,w2)
y2_hat=sigmoid(y2)

#3층
y3=np.dot(y2_hat,w3)
print(y3)

**재귀버전

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

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

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

print(ANN(x,w0,3))

**결과
[[2.9998643  6.99960519]]

```

| | |
|---------------|--|
| 문제 52. | 위의 코드중에 가중치에 해당하는 코드들을 init_network()라는 함수로 생성해서 가중치를 딕셔너리에서 가져올 수 |
|---------------|--|

있도록 하시오.

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

network=init_network() # 함수를 호출하면 network 딕셔너리를 생성
print(network['W1'])

**결과
[[1 3 5]
 [4 5 6]]
```

문제 53. 1,2,3층 코드를 하나의 함수로 생성하시오.

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

def identity_function(x):
    return x

def forward(network, x):
    W1, W2, W3 = network['W1'], network['W2'], network['W3']
    # 1층
    y1 = np.dot(x, W1)
    y1_hat = sigmoid(y1)

    # 2층
    y2 = np.dot(y1_hat, W2)
    y2_hat = sigmoid(y2)

    # 3층
    y3 = np.dot(y2_hat, W3)
    y3_hat = identity_function(y3)
    return y3_hat

def init_network():
    network ={}

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

    return network
```

```

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

```

****결과**

```

[[1 3 5]
 [4 5 6]]

```

문제 54. 입력한 값이 1,2가 아니라 4,5 일때의 위의 3층 신경망을 통과한 결과를 출력 하시오.

```

# layers_3.py 생성

import numpy as np

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

def identity_function(x):
    return x

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

    # 1층
    y1 = np.dot(x, W1)
    y1_hat = sigmoid(y1)

    # 2층
    y2 = np.dot(y1_hat, W2)
    y2_hat = sigmoid(y2)

    # 3층
    y3 = np.dot(y2_hat, W3)
    y3_hat = identity_function(y3)
    return y3_hat

def init_network():
    network = {}

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

    return network

```

```

import layers_3

```

```

network = layers_3.init_network()

```

```
x = layers_3.np.matrix([[1, 2]])
print(layers_3.forward(network, x))
```

****결과**

```
[[3.99985815 5.9997286 ]]
```

#softmax 함수

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

```
import numpy as np
```

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

```
print(np.exp(a)/np.sum(np.exp(a))) # 오버플로우 문제가 발생 e^1010, e^1000, e^990 ..
```

****결과**

```
[nan nan nan]
```

```
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:5: RuntimeWarning: invalid value encountered in true_divide
  """
```

문제 56.아래의 리스트에서 가장 큰 max 값을 출력 하시오.

```
import numpy as np
```

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

```
print(np.max(a))
```

****결과**

```
1010
```

문제 57.아래의 리스트에서 각 요소를 가장 큰 요소로 뺀 값이 얼마인지 출력 하시오.

```
import numpy as np
```

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

```
print(a-np.max(a))
```

****결과**

```
[ 0 -10 -20]
```

문제 58.위의 [0 -10 -20]을 자연상수 e의 제곱으로 해서 출력된 결과가 무엇인지 확인 하시오.

```
import numpy as np
```

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

```
print(a-np.max(a))
```

```
print(np.exp(a-np.max(a))/np.sum(np.exp(a-np.max(a))))
```

****결과**

```
[ 0 -10 -20]
```

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

문제 59. 위의 코드를 이용해서 softmax 함수를 생성하기 위한 분자식을 구현 하시오.

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

```
def softmax(x):
```

```
    max_v=np.max(x)
```

```
    minus_x=x-max_v
```

```
    np_exp=np.exp(minus_x)
```

```
    return np_exp
```

```
print(softmax(a))
```

****결과**

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

문제 60. softmax 함수를 분모까지 포함해서 완전히 완성하시오.

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

```
def softmax(x):
```

```
    max_v=np.max(x)
```

```
    minus_x=x-max_v
```

```
    exp_x=np.exp(minus_x)
```

```
    sum_exp_x=np.sum(exp_x)
```

```
    return exp_x/sum_exp_x
```

```
print(softmax(a))
```

****결과**

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

문제 61. 문제 53번 코드의 항등함수를 softmax함수로 바꾸시오.

```
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(x):
    max_v=np.max(x)
    minus_x=x-max_v
    exp_x=np.exp(minus_x)
    sum_exp_x=np.sum(exp_x)
    return exp_x/sum_exp_x

def forward(network, x):
    W1, W2, W3 = network['W1'], network['W2'], network['W3']
    # 1층
    y1 = np.dot(x, W1)
    y1_hat = sigmoid(y1)

    # 2층
    y2 = np.dot(y1_hat, W2)
    y2_hat = sigmoid(y2)

    # 3층
    y3 = np.dot(y2_hat, W3)
    y3_hat = identity_function(y3)
    return y3_hat

def init_network():
    network = {}

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

    return network

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

```

****결과**

```
[[0.11921653 0.88078347]]
```

| | |
|---------------|--|
| 문제 62. | 아래의 코드에 data set 패키지의 mnist.py에 load_mnist라는 함수가 있는지 확인 하시오. |
|---------------|--|

```

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)
# (훈련데이터, 라벨) (테스트데이터, 라벨)

```

```
print (len(t_train))
```

```
def load_mnist(normalize=True, flatten=True, one_hot_label=False):
    """MNIST 데이터셋 읽기

    Parameters
    -----
    normalize : 이미지의 픽셀 값을 0.0~1.0 사이의 값으로 정규화할지 정한다.
    one_hot_label :
        one_hot_label이 True면, 레이블을 원-핫(one-hot) 배열로 돌려준다.
        one-hot 배열은 예를 들어 [0,0,1,0,0,0,0,0,0,0]처럼 한 원소만 1인 배열이다.
    flatten : 입력 이미지를 1차원 배열로 만들지를 정한다.

    Returns
    -----
    (훈련 이미지, 훈련 레이블), (시험 이미지, 시험 레이블)
    """
    if not os.path.exists(save_file):
        init_mnist()

    with open(save_file, 'rb') as f:
        dataset = pickle.load(f)

    if normalize:
        for key in ('train_img', 'test_img'):
            dataset[key] = dataset[key].astype(np.float32)
            dataset[key] /= 255.0

    if one_hot_label:
        dataset['train_label'] = _change_one_hot_label(dataset['train_label'])
        dataset['test_label'] = _change_one_hot_label(dataset['test_label'])

    if not flatten:
        for key in ('train_img', 'test_img'):
            dataset[key] = dataset[key].reshape(-1, 1, 28, 28)

    return (dataset['train_img'], dataset['train_label']), (dataset['test_img'], dataset['test_label'])
```

****결과**

60000

| | |
|---------------|----------------------------|
| 문제 63. | x_train 데이터를 print 확인 하시오. |
|---------------|----------------------------|

```
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)
```

****결과**

```
[[0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [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 확인 하시오.

```
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))
```

****결과**

```
60000
```

문제 65. x_train 데이터의 6만개의 이미지 중에서 제일 첫 번째 이미지 한개를 출력 하시오.

```
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('라벨', t_train[0])
```

****결과**

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

라벨 5

딥러닝 페이지 49

| | |
|---------------|--------------------|
| 문제 67. | np 배열의 차원을 확인 하시오. |
|---------------|--------------------|

```
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]]

np_lst = np.array(x)
print(np_lst.ndim)
```

****결과**
2

| | |
|---------------|-----------------------|
| 문제 68. | 위의 리스트를 1차원으로 변경 하시오. |
|---------------|-----------------------|

```
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]]
```

```

np_lst = np.array(x)
print(np_lst.ndim)

np_lst = np_lst.flatten()
print(np_lst.ndim)

```

****결과**

```

2
1

```

문제 69. mnist의 훈련 데이터 (x_train)의 차원이 어떻게 되는지 확인 하시오.

```

import sys, os
import numpy as np

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)

np_lst = np.array(x_train)

print(np_lst.ndim)

```

****결과**

```

2
가로 784, 세로 6만개
[ [0,0,0,2,0,...]
  [0,0,...]
  [0,0,...]
  ....
]
```

문제 70. flatten을 false로 설정하고 훈련 데이터의 모양을 확인 하시오.

```

import sys, os
import numpy as np

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)
```

****결과**

```
(60000, 1, 28, 28)
```

```
# 전체장 수, 흑백, 가로, 세로
```

| | |
|---------------|--|
| 문제 71. | t_train[0]가 어떤 숫자인지 확인하는데, one-hot encoding 했을 때와 안했을 때의 차이를 확인 하시오. |
|---------------|--|

```
import sys, os
```

```
import numpy as np
```

```
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, one_hot_label=False)
```

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

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

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

****결과**

```
5
```

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

| | |
|---------------|-------------------------|
| 문제 72. | x_train[0] 요소를 시각화 하시오. |
|---------------|-------------------------|

```
# 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()
```

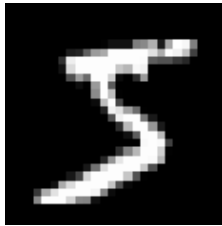
```
(x_train, t_train), (x_test, t_test) = load_mnist(flatten=True, normalize=False)
```

```
img = x_train[0]
```

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

```
img_show(img)
```

****결과**



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

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

```
x,y = get_data()
```

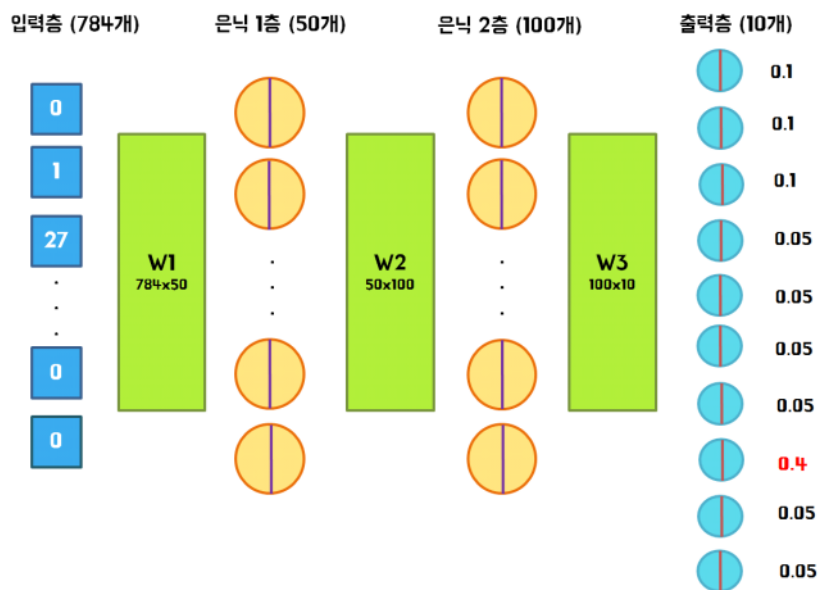
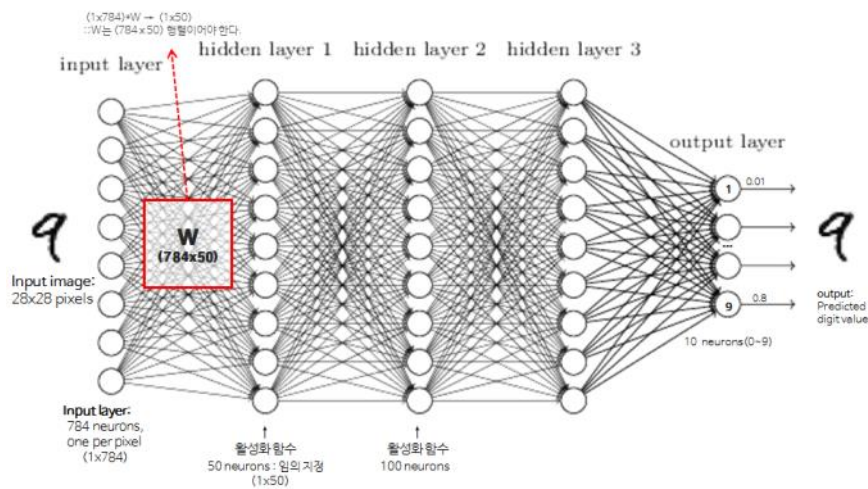
```
print(x)  
print(y)  
print(x.shape)  
print(len(y))
```

****결과**

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

(10000, 784) ----> 784개 요소 값을 가진 리스트가 1만개 있다.
10000

문제 74. 테스트 데이터 10000장을 3층 신경망으로 입력했을 때의 그림을 그리시오.



문제 75. mnist 필기체 데이터를 위해 신경망에서 사용할 가중치(weight)와 바이어스(bias)가 셋팅되어 있는 sample_weight.pkl을 로드하는 함수를 init_network()라는 이름으로 생성하여라.

```

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)

**결과
(784, 50)
  
```

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

```

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

def softmax(x):
    max_v=np.max(x)
    minus_x=x-max_v
    exp_x=np.exp(minus_x)
    sum_exp_x=np.sum(exp_x)
    return exp_x/sum_exp_x

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

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

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

    a1=np.dot(x,W1)+b1
    z1=sigmoid(a1)

    a2=np.dot(z1,W2)+b2
    z2=sigmoid(a2)

    a3=np.dot(z2,W3)+b3
    y=softmax(a3)

    return y

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

```

****결과**

```

[8.4412408e-05 2.6350606e-06 7.1549352e-04 1.2586251e-03 1.1727943e-06
 4.4990764e-05 1.6269318e-08 9.9706501e-01 9.3744702e-06 8.1831077e-04]

```

| | |
|---------------|--|
| 문제 77. | A4지에 필기체 숫자를 쓰고 사진을 찍어서 png 또는 jpg로 만든 후에 numpy 배열로 변환해서 위의 3층 신경망에 넣고 잘 예측하는지 확인 하시오. |
|---------------|--|

```

def sigmoid(num):

```

```

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

def softmax(x):
    max_v=np.max(x)
    minus_x=x-max_v
    exp_x=np.exp(minus_x)
    sum_exp_x=np.sum(exp_x)
    return exp_x/sum_exp_x

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

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

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

    a1=np.dot(x,W1)+b1
    z1=sigmoid(a1)

    a2=np.dot(z1,W2)+b2
    z2=sigmoid(a2)

    a3=np.dot(z2,W3)+b3
    y=softmax(a3)

    return y

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

```

****결과**

```

[8.4412408e-05 2.6350606e-06 7.1549352e-04 1.2586251e-03 1.1727943e-06
 4.4990764e-05 1.6269318e-08 9.9706501e-01 9.3744702e-06 8.1831077e-04]

```

문제 78. D:\WWnumber33.jpg 파일을 (28,28,3) ---> (28,28,1)로 변경 하시오.

```

import cv2
j = 'c:\wwdata\wwnumber33.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] # grayscale로 변환
print(img.shape)
```

****결과**

(28, 28, 3)

(28, 28)

| | |
|---------------|---|
| 문제 79. | 흑백으로 변경한 숫자 3을 3층 신경망에 넣고 숫자 3을 컴퓨터가 맞추는지 확인 하시오. |
|---------------|---|

```
import cv2
j = 'c:\data\mnist\mnist\train\33.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) = W
    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(np.where(y==max(y)))

**결과
(array([3], dtype=int64),)

```

| | |
|---------------|--|
| 문제 80. | 아래 10개의 원소를 갖는 x라는 리스트를 만들고 x 리스트에 가장 큰 원소가 몇 번째 인덱스인지 알아내시오. x=[0.05, 0.01, 0.02, 0.02, 0.1, 0.2, 0.3, 0.4 ,0.05 ,0.04] |
|---------------|--|

```

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)

**결과
7

```

문제 81.

위의 코드를 수정해서 `print(np.where(y==max(y)))` 대신에 `np.argmax`로 수정해서 결과를 출력 하시오.

```
import cv2
j = 'c:\data\11\11number33.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) = W
    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(np.argmax(y,axis=0))

```

****결과**

3 # 4번째 인덱스의 값이 제일 큰 값을 가진다.

| | |
|---------------|--|
| 문제 82. | 아래의 리스트를 numpy array 리스트로 변환하고, shape를 확인 하시오. |
|---------------|--|

```

x = [[0.1,0.05,0.1,0.0,0.05,0.1,0.0,0.1,0.0,0.0],

      [0.1,0.05,0.2,0.0,0.05,0.1,0.0,0.6,0.0,0.0],

      [0.0,0.05,0.3,0.0,0.05,0.1,0.0,0.6,0.0,0.0],

      [0.0,0.05,0.4,0.0,0.05,0.0,0.0,0.5,0.0,0.0],

      [0.0,0.05,0.5,0.0,0.05,0.0,0.0,0.4,0.0,0.0],

      [0.0,0.05,0.6,0.0,0.05,0.0,0.0,0.3,0.0,0.0],

      [0.0,0.05,0.7,0.0,0.05,0.0,0.0,0.2,0.0,0.0],

      [0.0,0.1,0.8,0.0,0.1,0.0,0.0,0.2,0.0,0.0],

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

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

```

****결과**

(9, 10)

문제 83. 위의 10,10 행렬에서 각 행에서의 가장 큰 원소가 몇번째에 존재하는지 출력 하여라.

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

```
x=np.array(x)
print(np.argmax(x,axis=1))
```

****결과**

```
[0 7 7 7 2 2 2 2 2]
```

문제 84. (점심시간문제) 각 라인별로 A4지에 숫자를 하나 필기체로 적고 그 숫자를 3층 신경망에 넣었을때 컴퓨터가 숫자를 잘 인식하는지 확인하시오.

```
j = 'c:\\data\\number1.jpg'
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

def rgb2gray(rgb):
    return np.dot(rgb[...,:3], [0.299, 0.587, 0.114])

img = mpimg.imread(j)
gray = rgb2gray(img)
a = np.array(gray)
x= a.flatten()
print(x.shape)

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)

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.argmax(y, axis=0))

**결과
(784,)
1

```

문제 85. 텍스트 데이터 하나 $x[34]$ 의 필기체의 라벨이 무엇인지 확인 하시오.

```

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(t[34])
```

****결과**

7

| | |
|---------------|---|
| 문제 86. | 테스트 데이터 하나의 x[34]의 필기체를 신경망에 넣고 신경망이 예측한 것과 라벨이 서로 일치 하는지 확인 하시오. |
|---------------|---|

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import sys, os
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) = W
    load_mnist(normalize=True, flatten=True, one_hot_label=False)
    return x_test, t_test

network = init_network()
y = predict(network,x[34])
print('예측 :',np.argmax(y), ' 실제 :',t[34])

```

****결과**

예측 : 7 실제 : 7

| | |
|---------------|--|
| 문제 87. | 테스트 데이터 10000장을 for loop문으로 전부 3층 신경망에 넣고 10000장 중에 몇장을 3층 신경망이 맞추는지 확인 하시오. |
|---------------|--|

```

import cv2
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import sys, os
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) = W
    load_mnist(normalize=True, flatten=True, one_hot_label=False)
    return x_test, t_test

network = init_network()
y = predict(network, x)

print('예측 :', np.argmax(y, axis=1), ' 실제 :', t)

true=0
false=0
for i in range(len(np.argmax(y, axis=1))):
    if np.argmax(y, axis=1)[i] == t[i]:
        true+=1
    else :
        false+=1

print('성공 :', true, ' 실패 :', false)
print(true/(true+false))

**결과
예측 : [7 2 1 ... 4 5 6] 실제 : [7 2 1 ... 4 5 6]
성공 : 9352 실패 : 648
0.9352

```

| | |
|---------------|------------------------|
| 문제 88. | 위의 문제에서 정확도도 출력하게 하시오. |
|---------------|------------------------|

```

import cv2
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import sys, os

```

```

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

network = init_network()
y = predict(network, x)

print('예측 :', np.argmax(y, axis=1), ' 실제 :', t)

true=0

```

```

false=0
for i in range(len(np.argmax(y,axis=1))):
    if np.argmax(y,axis=1)[i] == t[i]:
        true+=1
    else :
        false+=1

```

```

print('성공 :',true, ' 실패 :',false)
print('정확도 :',true/(true+false))

```

****결과**

예측 : [7 2 1 ... 4 5 6] 실제 : [7 2 1 ... 4 5 6]
 성공 : 9352 실패 : 648
 정확도 : 0.9352

| | |
|---------------|--|
| 문제 89. | 아래의 결과를 출력 하시오. [0,1,2,3,4,5,6,7,8,9] |
|---------------|--|

```

lst = list(range(10))
print(lst)

```

****결과**

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

| | |
|---------------|---------------------------------|
| 문제 90. | 아래와 같이 결과를 출력 하시오. [0,3,6,9] |
|---------------|---------------------------------|

```

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

```

****결과**

[0, 3, 6, 9]

| | |
|---------------|---------------------------------|
| 문제 91. | 아래의 리스트에서 최대값의 원소의 인덱스를 출력 하시오. |
|---------------|---------------------------------|

```

lst = list(range(0,10,3))
print(np.argmax(lst))

```

****결과**

3

| | |
|---------------|---------------------------|
| 문제 92. | 아래의 행렬을 numpy 배열로 생성 하시오. |
|---------------|---------------------------|

```

import numpy as np

array = 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(array)

```

****결과**

```
[[0.1 0.8 0.1]
 [0.3 0.1 0.6]
 [0.2 0.5 0.3]
 [0.8 0.1 0.1]]
```

문제 93. numpy의 argmax를 이용해서 아래의 행렬에서 행 중에 최대값 원소의 인덱스를 출력 하시오.

```
import numpy as np

array = np.array([[0.1,0.8,0.1],[0.3,0.1,0.6],[0.2,0.5,0.3],[0.8,0.1,0.1]])
print(np.argmax(array,axis=1))
```

****결과**

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

문제 94. 아래의 2개의 리스트를 만들고 서로 같은 자리에 같은 숫자가 몇개가 있는지 출력 하시오.
[2,1,3,5,1,4,2,1,1,0]

```
arr1=[2,1,3,5,1,4,2,1,1,0]
arr2=[2,1,3,4,5,4,2,1,1,2]

cnt =0
for i,j in zip(arr1,arr2):
    if i==j:
        cnt+=1

print(cnt)
```

****결과**

```
7
```

문제 95. 아래의 리스트를 x라는 변수에 담고 앞에 5개의 숫자만 출력 하시오.

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

print(x[:5])

** numpy를 이용한 방법
arr1=np.array([2,1,3,5,1,4,2,1,1,0])
arr2=np.array([2,1,3,4,5,4,2,1,1,2])

print(np.sum(arr1==arr2))
```

****결과**

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

| | |
|---------------|--|
| 문제 95. | 아래의 리스트를 x라는 변수에 담고 앞에 5개의 숫자만 출력 하시오. |
|---------------|--|

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

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

****결과**

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

| | |
|---------------|--|
| 문제 96. | 아래의 숫자들을 출력 하시오. 100, 200, 300, 400 ... , 10000 |
|---------------|--|

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

****결과**

```
100
```

```
200
```

```
300
```

```
400
```

```
...
```

```
9900
```

```
10000
```

| | |
|---------------|---------------------------------------|
| 문제 97. | mnist의 훈련 데이터를 100개씩 가져오는 코드를 작성 하시오. |
|---------------|---------------------------------------|

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

****결과**

```
100
```

```
200
```

```
300
```

```
400
```

```
...
```

```
9900
```

```
10000
```

| | |
|---------------|---|
| 문제 98. | 100개 가지고 온 훈련데이터를 predict 함수에 입력해서 예측 숫자 100개를 출력하는 코드를 작성 하시오. |
|---------------|---|

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

****결과**

```
100
```

```
200
```

```
300
```

400
...
9900
10000

문제 99. 위의 코드를 수정해서 예측 숫자 100개와 실제 숫자 100개를 출력하게끔 파이썬 코드를 작성 하시오.

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

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

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

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

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

cnt=0
batch_size=100
for i in range(0,10000,100):
    x_batch = x[i:i+batch_size]
    y_batch=predict(network,x_batch)
    p = np.argmax(y_batch,axis=1)
    cnt+=np.sum(p==t[i:i+batch_size])
    print('p :',p)
    print('t :',t[i:i+batch_size])
print(cnt)

**결과
p : [7 2 1 0 4 1 4 9 6 9 0 6 9 0 1 5 9 7 3 4 9 6 6 5 4 0 7 4 0 1 3 1 3 6 7 2 7
 1 2 1 1 7 4 2 3 5 1 2 4 4 6 3 5 5 6 0 4 1 9 5 7 8 9 3 7 4 2 4 3 0 7 0 2 9
 1 7 3 2 9 7 7 6 2 7 8 4 7 3 6 1 3 6 4 3 1 4 1 7 6 9]

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

1 7 3 2 9 7 7 6 2 7 8 4 7 3 6 1 3 6 9 3 1 4 1 7 6 9]

...

| | |
|----------------|---|
| 문제 100. | 아래의 100개씩 뽑은 예상 숫자와 실제 숫자가 서로 얼마나 일치하는지 숫자를 count 해서 아래와 같은 결과를 출력 하시오. |
|----------------|---|

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

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

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

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

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

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

****결과**

9352

4장. 신경망 학습

2018년 8월 23일 목요일 오후 2:22

4.1 손실함수

손실함수는 신경망을 학습할 때 학습 상태에 대해 측정하는 하나의 지표로 사용한다. 신경망의 가중치 매개변수들이 스스로 특징을 찾아 가기에 이 가중치 값의 최적이 될 수 있도록 해야 하며 잘 찾아가고 있는지 볼 때 손실 함수를 보는 것이다.

1) 평균제곱오차 Mean Squared Error

예측하는 값이랑 실제 값의 차이(error)를 제곱하여 평균을 낸 것이 평균제곱오차이다.

예측 값과 실제 값의 차이가 클수록 평균제곱오차의 값도 커진다는 것은 이 값이 작을 수록 예측력이 좋다고 할 수 있다.

**평균제곱오차 Mean Squared Error

```
def mean_squared_error(y, t):  
    return 0.5*np.sum((y-t)**2)  
  
#정답은 2  
t = [0,0,1,0,0,0,0,0,0,0]  
  
y1 = [0.1, 0.05, 0.6, 0.0, 0.05, 0.1, 0.0, 0.1, 0.0, 0.0]  
print(mean_squared_error(np.array(y1), np.array(t)))  
  
y2 = [0.1, 0.05, 0.1, 0.0, 0.05, 0.1, 0.0, 0.6, 0.0, 0.0]  
print(mean_squared_error(np.array(y2), np.array(t)))
```

**결과

```
0.0975  
0.5975
```

2) 교차 엔트로피 오차 Cross Entropy Error

교차엔트로피는 로그의 밑이 e인 자연로그를 예측 값에 씌워서 실제 값과 곱한 후 전체 값을 합한 후 음수로 변환한다. 실제 값이 원핫인코딩(one-hot encoding; 더미변수처럼 1~9까지 범주로 했을 때 정답이 2일 경우 2에는 '1'을 나머지 범주에는 '0'으로) 방식 일 경우에는 2를 제외한 나머지는 무조건 0이 나오므로 실제 값일 때의 예측 값에 대한 자연로그를 계산하는 식이 된다. 실제 값이 2인데 0.6으로 예측했다면 교차 엔트로피 오차는 $-\log(1*0.6) = -\log 0.6$ 이 된다. $= 0.51$
교차 엔트로피는 출력이 1일 때 0이 되며 x가 커질수록 0에 가까워지고 x가 작아질수록(0에 가까워질수록) 값이 작아진다 (음의방향).

```
def cross_entropy_error(y,t):  
    delta = 1e-7  
    return -np.sum(t * np.log(y+delta))  
  
t = [0,0,1,0,0,0,0,0,0,0]  
y1 = [0.1, 0.05, 0.6, 0.0, 0.05, 0.1, 0.0, 0.1, 0.0, 0.0]  
print(cross_entropy_error(np.array(y1), np.array(t)))  
y2 = [0.1, 0.05, 0.1, 0.0, 0.05, 0.1, 0.0, 0.6, 0.0, 0.0]
```



```
print(cross_entropy_error(np.array(y2), np.array(t)))
```

****결과**

0.510825457099

2.30258409299

#

평균제곱오차에서 계산한 것과 동일하게 처음의 예측이 더 잘못았음을 교차엔트로피 오차로 확인할 수 있다.

손실함수 종류

| 구분 | 수식 |
|--|--|
| 평균 제곱 오차 (mean squared error, MSE) | $E = \frac{1}{2} \sum_k (y_k - t_k)^2$ |
| 교차 엔트로피 오차 (Cross entropy error, CEE) | $E = - \sum_k t_k \log y_k$ |

4.2 미니배치 학습

앞의 교차 엔트로피 오차에서 구한 것은 하나의 케이스(instance)에 대한 손실함수를 구한 것으로 만약 mnist 데이터와 같이 약 6만개 정도의 케이스에 대해서 학습을 할 때는 교차 엔트로피 오차를 6만번 계산해야되게 된다.

만약 실제 빅데이터 환경에 간다면 6만이 아니라 6백만, 6천만, 6억개의 데이터가 될 수도 있으며 일일이 계산한다면 굉장히 오랜 시간이 걸리게 될 것이다.

따라서 한번에 하나만 계산하는게 아니라 일부를 조금씩 가져와서 전체의 '근사치'로 이용하여 일부분 계속 사용하여 학습을 수행하는 방법을 이용한다. 그 일부를 미니배치 mini-batch 라고 한다. 훈련 데이터에서 일부를 무작위로 뽑아 학습하는 것은 미니배치 학습이다.

미니배치는 무작위로 추출하는 것으로 표본을 무작위로 샘플링하는 것과 개념적으로 유사하다.

"훈련 데이터 중에 일부분 골라서 학습하는 방법"

--> 표본을 뽑아서 학습 시킨다.

복원 추출이든 비복원 추출이든 mnist의 경우 100장씩 배치 처리한다면 100장씩 600번을 훈련시키면 그게 1epoch이다. 여러번 에폭이 반복되면서 비용함수의 global minima로 찾아가게 된다.

■ 미니배치 처리에 맞게끔 교차 엔트로피 함수를 구성하는 방법

```
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) )
```

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

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

```
    return -np.sum( t * np.log(y+delta/y.shape[0]) )
```

■ 수치미분 (p121)

진정한 미분은 컴퓨터로 구현하기 어렵기 때문에 중앙 차분 오차가 발생하지만 컴퓨터로 미분을 구현하기 위해서는 수치미분을 사용해야한다.

```
def numerical_diff(f,x):
```

```
    h = 1e-4      # 0.0001 -->컴퓨터로는 극한 값을 구하기 어려우므로 h를 10^-4 값을 사용하면 좋은 결과가 나온다고 알려져있다.
```

```
    return (f(x+h) - f(x))/h # 컴퓨터로는 미분을 하지 못하니 도함수를 구현해야 한다.
```

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

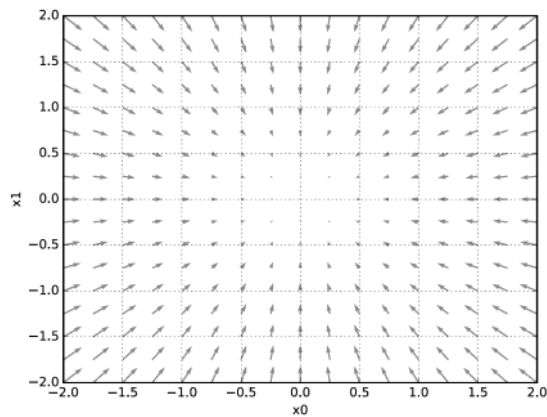
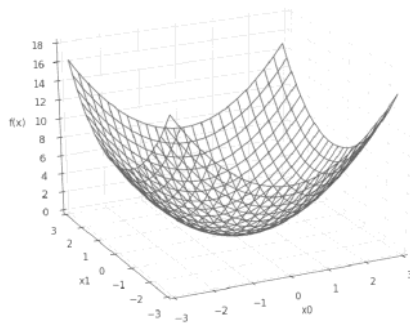
```
def numerical_diff(f,x):
```

```
    h=1e-4 # 0.0001
```

```
    return (f(x+h) - f(x-h)) / (2*h) # 함수를 미분하는 수치미분 함수
```

■ 편미분 (p125)

변수가 2개 이상인 함수를 미분할 때 미분 대상 변수 외에 나머지 변수를 상수 처럼 고정 시켜 미분하는 것을 편미분 이라고 한다.



| | |
|---------|--|
| 문제 101. | <p>평균제곱 오차 함수를 책 112 페이지 식을 보고 mean_squared_error 라는 함수 이름으로 만들고 아래와 같이 구현 하시오.</p> <pre>t=[0,0,1,0,0,0,0,0,0] # 실제값</pre> <pre>y=[0.1,0.05,0.6,0.0,0.05,0.1,0.0,0.1,0.0,0.0] #예측값</pre> |
|---------|--|

| |
|--|
| <pre>import numpy as np</pre> <pre>t=[0,0,1,0,0,0,0,0,0] # 실제값</pre> <pre>y=[0.1,0.05,0.6,0.0,0.05,0.1,0.0,0.1,0.0,0.0] #예측값</pre> |
|--|

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

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

****결과**

0.097500000000000003

문제 102. 아래의 확률벡터를 평균제곱오차 함수를 이용해서 target(실제값)과 예측값의 오차율이 어떻게 되는지 for loop문으로 한 번에 알아내시오.

```
import numpy as np

t=[0,0,1,0,0,0,0,0,0] # 실제값
y=[0.1,0.05,0.6,0.0,0.05,0.1,0.0,0.1,0.0,0.0] #예측값

def mean_squared_error(y,t):
    return np.sum(np.square(y-t))/2

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

****결과**

0.097500000000000003

문제 103. 밑이 자연상수이고 진수가 0.6인 로그 값을 출력 하시오.

```
import numpy as np

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

****결과**

-0.5108256237659907

문제 104. 밑수가 10이고 진수가 0.6인 로그의 로그값을 출력 하시오.

```
import numpy as np

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

또는

```
import math
print(math.log(0.6, 10))
```

****결과**

-0.5108256237659907

문제 105. mean square error 함수의 공식을 가지고 0.1과 0.9 확률의 오차를 각각 출력 하시오.

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

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

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

print(mean_squared_error(x, t))
print(mean_squared_error(y, t))
```

****결과**

0.81

0.009999999999999998

문제 106. 밑수가 자연상수이고 진수가 0인 로그의 로그값은 얼마인지 출력 하시오.

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

****결과**

-inf

```
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: RuntimeWarning: divide by zero encountered in log
  """Entry point for launching an IPython kernel.
```

#마이너스 무한대 출력

-inf 가 출력되면 컴퓨터로 수치연산을 할 수 없다.

문제 107. cross entropy error 함수를 아래의 공식을 보고 생성 하시오.

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

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

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

****결과**

0.1053604045467214

****설명**

delta를 더해주는 이유는 np.log() 함수에 0을 입력하면 마이너스 무한대를 뜻하는 -inf가 되어 더 이상 계산을 진행 할 수 없기 때문이다. 그래서 아주 작은 값을 더해서 진수가 0이 되지 않도록 한다.

문제 108. 60000 숫자중에 무작위로 10개를 출력 하시오.

```
import numpy as np

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

****결과**

```
[18285 37292 10249 58912 24908 23764 21844 31705 12158 35151]
```

문제 109. mnist의 테스트 데이터 10000장 중에 랜덤으로 100장을 추출하는 코드를 작성 하시오.

```
import numpy as np

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

****결과**

```
[18285 37292 10249 58912 24908 23764 21844 31705 12158 35151]
```

문제 109+ 문제 100번을 비복원 추출에서 복원추출을 하게끔 수정 하시오.

```
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(np.arange(10000),100) # 10000개 중 10개를 랜덤하게 받음. (복원추출)

    x_batch = x[batch_mask] #복원추출
    t_batch = t[batch_mask]

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

**결과
10000 개 중에서 일치하는 수 : 9336 #복원 추출이기 때문에 실행할 때마다 결과가 바뀐다.

```

문제 110. 위 코드를 5에폭 돌게 코드를 수정 하시오.

```

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(np.arange(10000),100) # 10000개 중 10개를 랜덤하게 받음. (복원추출)

    x_batch = x[batch_mask] #복원추출
    t_batch = t[batch_mask]

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

**결과
10000 개 중에서 일치하는 수 : 9336   #복원 추출이기 때문에 실행할 때마다 결과가 바뀐다.

```

| | |
|----------------|---|
| 문제 111. | <p>아래의 비용함수를 수치미분 함수로 미분을 하는데, $x=4$ 에서의 미분계수(기울기)를 구하시오.</p> <p>$y=x^2 + 4^2$ (비용함수)</p> |
|----------------|---|

```

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

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

print(numerical_diff(loss2,4))

```

****결과**

7.999999999999119 #진정한 미분이었다면 8이 나왔지만 중앙차분 오차가 발생한다.

| | |
|----------------|--------------------------------|
| 문제 112. | 문제 111번의 비용함수를 아래와 같이 시각화 하시오. |
|----------------|--------------------------------|

```

import matplotlib.pyplot as plt
import numpy as np
def numerical_diff(f,x):
    h=1e-4 # 0.0001
    return (f(x+h) - f(x-h)) / (2*h) # 함수를 미분하는 수치미분 함수

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

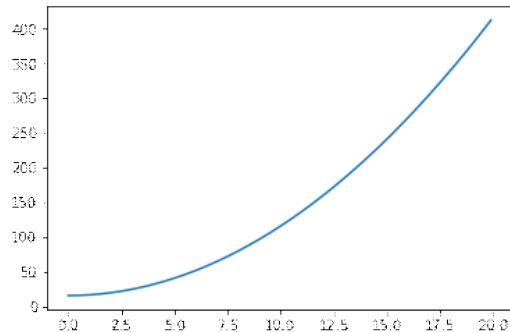
x=np.arange(0,20,0.1)

```

```
y=[]
for i in x:
    y.append(loss2(i))
```

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

****결과**



문제 113. 아래의 그래프를 구글에서 시각화 하시오.

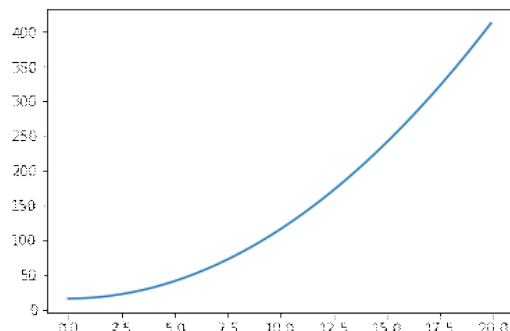
```
import matplotlib.pyplot as plt
import numpy as np
def numerical_diff(f,x):
    h=1e-4 # 0.0001
    return (f(x+h) - f(x-h)) / (2*h) # 함수를 미분하는 수치미분 함수

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

x=np.arange(0,20,0.1)
y=[]
for i in x:
    y.append(loss2(i))

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

****결과**



문제 114. $f(x_0, x_1) = x_0^w + x_1^2$ 함수를 편미분하는데, $x_0=3$, $x_1=4$ 일 때, x_0 에 대해서 편미분 하시오.


```
import matplotlib.pyplot as plt
import numpy as np
def numerical_diff(f,x):
    h=1e-4 # 0.0001
    return (f(x+h) - f(x-h) ) / (2*h) # 함수를 미분하는 수치미분 함수

def samplefunc1(x):
    return x**2+4**2

print(numerical_diff(samplefunc1,3))
print(numerical_diff(samplefunc1,4))
```

****결과**

6.000000000000378
7.999999999999119

문제 115. $f(x_0, x_1) = x_0^2 + x_1^2$ 함수를 편미분 하는데 $x_0=3$, $x_1=4$ 일 때 x_1 에 대해서 편미분 하시오.

```
import matplotlib.pyplot as plt
import numpy as np
def numerical_diff(f,x):
    h=1e-4 # 0.0001
    return (f(x+h) - f(x-h) ) / (2*h) # 함수를 미분하는 수치미분 함수

def samplefunc1(x):
    return x**2+4**2

def samplefunc2(x):
    return 3**2+x**2

print(numerical_diff(samplefunc2,3))
print(numerical_diff(samplefunc2,4))
```

****결과**

6.000000000000378
7.999999999999119

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

```
import numpy as np

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

****결과**

[0. 0.]

문제 117. 위에서는 $f(x_0, x_1) = x_0^2 + x_1^2$ 함수를 편미분하는 것을 각각 수행했는데 이번에는 편미분이 한 번에 수행되게끔 하는 코드를 작성 하시오.
책 127p 아래에 나오는 numerical_gradient 함수를 직접 만드시오.

```
import numpy as np
```

```

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

def numerical_gradient(x,y):
    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)

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

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

    return grad

```

****결과**

[0. 0.]

| | |
|----------------|--|
| 문제 118. | 아래의 x0, x1 지점에서의 기울기를 각각 구하시오. x0, x1 [3.0, 4.0] [0.0, 2.0] [3.0, 0.0] |
|----------------|--|

```

x = np.array([3.0, 4.0])
x1 = np.array([0.0, 2.0])
x2 = np.array([3.0, 0.0])

import numpy as np

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

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

x = np.array([3.0, 4.0])
x1 = np.array([0.0, 2.0])
x2 = np.array([3.0, 0.0])

print ( numerical_gradient( loss_func, x ) ) #[6. 8.]
print ( numerical_gradient( loss_func, x1 ) ) #[0. 4.]
print ( numerical_gradient( loss_func, x2 ) ) #[6. 0.]

```

****결과**

[0. 0.]

문제 119. 위에서 만든 numerical_gradient 함수를 100번 수행되게 해서 global minuma에 도달하게끔 하는 코드를 구현 하시오.

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

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

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

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

print(gradient_descent(loss_func,init_x,0.1,100))
```

****결과**

[6.11110793e-10 8.14814391e-10]

문제 120. 러닝 레이트(학습률)을 크게 주고 학습을 하면 결과가 어떻게 나오는지 확인하고 러닝 레이트 학습률을 적게 주고 학습을 하면 결과가 어떻게 나오는가?

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

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

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

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

print(gradient_descent(loss_func,init_x.copy(),100,100))
print(gradient_descent(loss_func,init_x.copy(),0.0001,100))
```

****결과**

[2.67498081e+14 -1.25119902e+12]

[2.94059014 3.92078685]

■ 학습이 스스로 되는 3층 신경망 구현

문제 121. 2x3 행렬을 생성하는데 값은 랜덤 값으로 생성되게 하시오.

```
import numpy as np

w=np.random.rand(2,3)
```

```
print(w)
```

****결과**

```
[[0.20299822 0.26098418 0.64834628]
 [0.96236855 0.75517662 0.3546467 ]]
```

문제 122. 위에서 구한 w 값으로 아래의 입력 값과 행렬의 내적을 출력 하시오.

```
import numpy as np
```

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

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

****결과**

```
[0.77906922 0.77599472 0.93134846]
```

문제 123. 문제 121번 코드를 __init__ 라는 함수로 생성 하시오.

```
import numpy as np
```

```
def __init__():
    w=np.random.rand(2,3)
```

문제 124. 위에서 만든 가중치와 아래의 입력값과 행렬 내적을 하는 함수를 predict 란 이름으로 생성 하시오.

```
import numpy as np
```

```
def __init__():
```

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

```
def predict(x,w):
    return np.dot(x,w)
```

```
x = np.array([0.6, 0.9]) # 1 x 2 행렬
w = __init__()
print ( predict(x,w))
```

****결과**

```
[-0.13593501 -1.59435611  1.05499264]
```

문제 125. 위의 predict에서 나온 결과를 softmax 출력층 함수에 통과시켜 출력 하시오.

```
import numpy as np
```

```
def __init__():
```

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

```

return w

def predict(x,w):
    return np.dot(x,w)

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

x = np.array([0.6, 0.9]) # 1 x 2 행렬
w = __init__()
print ( softmax(predict(x,w)))

**결과
[0.14353739 0.42711401 0.42934861]

```

문제 125. 위의 predict에서 나온 결과를 softmax 출력층 함수에 통과시켜 출력 하시오.

```

import numpy as np

def __init__():

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

def predict(x,w):
    return np.dot(x,w)

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

x = np.array([0.6, 0.9]) # 1 x 2 행렬
w = __init__()
print ( softmax(predict(x,w)))

**결과
[0.14353739 0.42711401 0.42934861]

```

문제 126. 위에서 출력한 소프트맥스 함수 결과와 아래의 target 값과의 오차를 출력하기 위해 cross_entropy_error 함수를 통과한 결과를 출력 하시오.

```

import numpy as np

def __init__():

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

def predict(x,w):
    return np.dot(x,w)

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

def cross_entropy_error(y,t):

```

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

```
x = np.array([0.6, 0.9]) # 1 x 2 행렬
w = __init__()
```

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

****결과**

```
[0.14353739 0.42711401 0.42934861]
```

문제 127. 위의 3가지 함수를 다 사용한 simpleNet 이라는 클래스를 책 134 페이지를 보고 생성하고 135 페이지를 보며 클래스가 잘 생성 되었는지 확인 하시오.

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

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

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

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

    def loss(self, x, t):
        y = self.softmax(self.predict(x))
        return (self.cross_entropy_error(y, t))
```

문제 128. TwoLayerNet 클래스로 객체를 생성하고 w1,w2,b1,b2 의 행렬의 모양을 확인 하시오.

```
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

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)

**결과
w1 (784, 50)
w2 (50, 10)
b1 (50,)
b2 (10,)

```

| | |
|----------------|---|
| 문제 129. | TwoLayerNet 클래스로 객체를 생성하고 predict 메소드를 실행해서 나온 결과의 행렬의 shape를 출력 하시오. |
|----------------|---|

```

import numpy as np
from dataset.mnist import load_mnist
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()
TN=TwoLayerNet(input_size=784,hidden_size=50,output_size=10)
print(TN.predict(x[:100]).shape)

**결과
(100, 10)

```

| | |
|----------------|---|
| 문제 130. | TwoLayerNet 클래스로 객체를 생성하고 crossentropyError 함수를 통과한 결과를 출력 하시오. |
|----------------|---|

```

# 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 #媛?蹂듭쌔
        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

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

x,t=get_data()
TN=TwoLayerNet(input_size=784,hidden_size=50,output_size=10)
print(TN.predict(x[:100]).shape)

print(TN.loss(x[:100],t[:100]))

**결과
(100, 10)
6.916611891004159

```

■ accuracy 함수의 이해

" 예상한 숫자와 실제 숫자를 비교해서 정확도를 출력하는 함수 "

```

def accuracy(self, x, t) :
    y = self.predict(y, axis=1)    # 10개짜리 확률 벡터(10개의 합이 1인) 100개 출력 (100x10 matrix)
                                    ex. [[0.1,0.05,0.0.1 .... ], [0.3, 0.03. 0.24, ...], ....]
    y = np.argmax(t, axis=1)       # 각 행(10개마다)의 최대 값을 가진 요소의 인덱스 반환 100개
                                    ex. [3 0 1 5 6 .... 3]
    t = np.argmax(t, axis=1)       # 라벨 값을 가짐 ex. [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 훈련 데이터와 타겟 데이터를 입력 했을때의 정확도를 출력 하시오. |
|---------|---|

```
# 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

import numpy as np
import matplotlib.pyplot as plt
from dataset.mnist import load_mnist

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]

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

(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.accuracy(x_train[:100], t_train[:100]))

**결과
0.11

```

■ numerical_gradient

" 비용함수와 가중치 또는 바이어스를 입력받아 기울기를 출력하는 함수 "

```

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

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

    return grads

```

■ 비용함수 생성 방법

```

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)

```

```

** 결과

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

```

■ lambda 표현식이란?

" 여러줄의 코드를 딱 한줄로 만들어주는 인자 = 이름없는 함수 "

```

def hap(x,y):
    return x+y

```

```

print( hap(3,4))

```

위의 코드를 lambda 표현식으로 표현하면?

```

print((lambda x,y : x+y)(10,20))

```

| | |
|----------------|---|
| 문제 132. | 아래의 비용함수를 lambda 표현식으로 한 줄로 변경해서 출력 하시오. <pre> def f(w): return network.loss(x,t) </pre> |
|----------------|---|

```

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

```

| | |
|----------------|---|
| 문제 133. | TwoLayerNet 클래스를 가지고 학습 시키는 코드를 완성시키시오. |
|----------------|---|

문제133. TwoLayerNet 클래스를 가지고 학습 시키는 코드를 완성시키시오 !

```

# 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 = [] #[6.9 6.7 6.5 .....] 오차를 에폭마다 입력
train_acc_list = [] # [0.23 0.24 0.25 ...] train 데이터의 정확도를 에폭마다 입력

```

```

test_acc_list = [] # test 데이터의 정확도를 에폭마다 입력

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

for i in range(iters_num): # 0:10000 랜덤으로 100개씩 뽑는 작업을 1만번 수행
    # 미니배치 획득
    batch_mask = np.random.choice(train_size, batch_size)
                                #60000 , 100 : 0~60000 의 숫자 중 100개를 랜덤 생성
    x_batch = x_train[batch_mask] # 100 x 784
    t_batch = t_train[batch_mask] # 100 x 10

    # 기울기 계산
    grad = network.numerical_gradient(x_batch, t_batch) # 수치 미분을 이용한 기울기
    #grad = network.gradient(x_batch, t_batch) # 오차 역전파를 이용한 기울기

    # 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()

```

| | |
|----------------|---|
| 문제 134. | <p>수치미분이 아닌 오차 역전파를 이용한 신경망 학습으로 2층 신경망 코드를 구현하시오.</p> <ol style="list-style-type: none"> 1. common 패키지를 working directory에 가져다 둔다. 2. 오차 역전파를 이용한 신경망 코드를 돌린다. |
|----------------|---|

문제133. TwoLayerNet 클래스를 가지고 학습 시키는 코드를 완성시키시오 !

```

# 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 = [] # [6.9 6.7 6.5 .....] 오차를 에폭마다 입력
train_acc_list = [] # [0.23 0.24 0.25 ...] train 데이터의 정확도를 에폭마다 입력
test_acc_list = [] # test 데이터의 정확도를 에폭마다 입력

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

for i in range(iters_num): # 0:10000 랜덤으로 100개씩 뽑는 작업을 1만번 수행
    # 미니배치 획득
    batch_mask = np.random.choice(train_size, batch_size)
    # 60000, 100 : 0~60000의 숫자 중 100개를 랜덤 생성
    x_batch = x_train[batch_mask] # 100 x 784
    t_batch = t_train[batch_mask] # 100 x 10

    # 기울기 계산
    grad = network.numerical_gradient(x_batch, t_batch) # 수치 미분을 이용한 기울기
    # grad = network.gradient(x_batch, t_batch) # 오차 역전파를 이용한 기울기

    # 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()

```

```

# 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

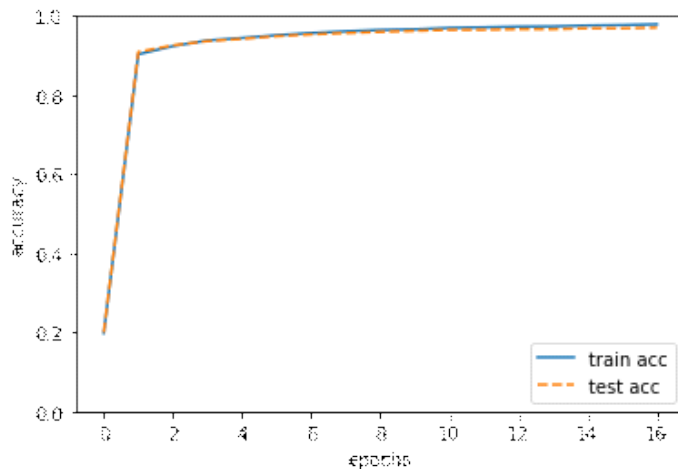
if i % iter_per_epoch == 0: # 600 번마다 정확도 쌓는다.

    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()

```



1장. 신경망을 구현하기 위해 알아야하는 파이썬 문법

- numpy 사용법
- matplotlib 그래프 그리는 사용법

2장. 퍼셉트론

신경망 안에 뉴런 1개를 컴퓨터로 구현

- 퍼셉트론 종류 2가지
 1. 단층 퍼셉트론
 2. 다층 퍼셉트론

단층 퍼셉트론의 한계를 다층 퍼셉트론으로 해결했는데 그 한계가 무엇이었는가?

" xor 게이트를 단층 퍼셉트론으로는 구현이 불가능해서 다층 퍼셉트론으로 해결했다."
"

3장. 신경망 구현

퍼셉트론 -----> 신경망

and 게이트

or 게이트

nand 게이트

xor 게이트

결과를 보려면 w를 사람이 직접 정해줬어야 했는데,

신경망의 w를 랜덤으로 지정해주기만 하고 컴퓨터가 알아서 w를 알아낸다.

- 신경망에 들어가는 활성화 함수

" 신호를 보낼지 안보낼지를 결정하는 함수 "

1. 계단함수 : 숫자 0과 1을 리턴하는 함수
2. 시그모이드 함수 : 숫자 0~1 사이의 실수를 리턴하는 함수 (확률을 출력하는 함

수)

3. 렐루 함수 : 입력되는 값이 0보다 크면 그 값을 그대로 출력하고 0 이하면 0을 리턴하는 함수

• 시그모이드 함수의 유래

| 구분 | 설명 |
|----------|---------------------------------------|
| 오즈비율 그래프 | 실패할 확률 대비 성공할 확률 |
| 로짓함수 | 오즈비율 함수에 로그를 씌움 |
| 시그모이드 함수 | 로짓함수를 신경망에서 확률(p)을 계산하기 편하도록 지수형태로 바꿈 |

numpy 함수

2018년 8월 14일 화요일

오전 11:13

#numpy 함수 모음

| 함수명 | 설명 | 예시 |
|--------------|---|---|
| np.mean(x) | #평균 값 n차원 배열&매트릭스 요소들 값의 평균 값을 리턴 | <pre>a = np.array([1,2,4,5,5,7,10,13,18,21]) np.mean(a) >> 8.6</pre> |
| np.median(x) | #중앙 값 n차원 배열&매트릭스 요소들 값의 중앙 값을 리턴 | <pre>a = np.array([1,2,4,5,5,7,10,13,18,21]) np.median(a) >> 6.0</pre> |
| np.max(x) | #최대 값 n차원 배열&매트릭스 요소들 값의 최대 값을 리턴 | <pre>a = np.array([1,2,4,5,5,7,10,13,18,21]) np.max(a) >> 21</pre> |
| np.min(x) | #최소 값 n차원 배열&매트릭스 요소들 값의 최소 값을 리턴 | <pre>a = np.array([1,2,4,5,5,7,10,13,18,21]) np.min(a) >> 1</pre> |
| np.std(x) | #표준편차 n차원 배열&매트릭스 요소들 값의 표준편차를 리턴 **옵션 ddof = 1을 주면 분모가 n-1 (표준분산) 아무것도 안주면 분모 n (모분산) 모분산 : 편차 ² / n , 표준분산 : 편차 ² /n-1 | <pre>a = np.array([1,2,4,5,5,7,10,13,18,21]) np.std(a) >>6.437390775772433</pre> |
| np.var(x) | #분산 n차원 배열&매트릭스 요소들 값의 표준편차를 리턴 | <pre>a = np.array([1,2,4,5,5,7,10,13,18,21]) np.var(a) >>41.44</pre> |
| x.flatten() | #1차원 배열로 변환 n차원 배열 -> 1차원 배열로 변환 | <pre>a=np.array([[1,2,3],[4,5,6]]) b=a.flatten() print(b) >>[1 2 3 4 5 6]</pre> |
| a.astype(타입) | #배열의 자료형을 변환 **타입 - np.int : 정수형 - np.float : 실수형 - np.str : 문자형 등... | <pre>a=np.array([1,2,3]) b=a.astype(np.str) c=a.astype(np.bool) print(b) print(c) >>['1' '2' '3'] >>[True True True]</pre> |
| np.exp(x) | #자연상수 e의 x승 반환 | <pre>print(np.exp(1)) >>2.718281828459045</pre> |
| np.arange(x) | #범위 반환 np.arange([start],end,[증가값]) | <pre>a=np.arange(1,10,2) print(a) >>[1 3 5 7 9]</pre> |
| np.dim(x) | #배열의 차원수 반환 | <pre>a=np.array([[1,2,3],[4,5,6]]) print(np.ndim(a)) >>2</pre> |
| x.shape | #배열의 행,열 반환 배열의 행, 열을 튜플로 반환한다. - 2차원 배열일 때, (3,2) 를 반환 했다면 3x2 배열이란 뜻. | <pre>a=np.array([[1,2,3],[4,5,6]]) print(a.shape) >>(2,3) a=np.array([[[1,2,3,4],[5,6,7,8]],[[11,22,</pre> |

| | | |
|------------------|--|---|
| | | <pre> a=np.array([[[1,2,3,4],[5,6,7,8]],[[11,22,33,44],[55,66,77,88]],[[5,3,2,9],[4,5,3,0]]]) [[[1 2 3 4] [5 6 7 8]] [[11 22 33 44] [55 66 77 88]] [[5 3 2 9] [4 5 3 0]]] print(a.shape) (3, 2, 4) </pre> |
| np.dot(x,y) | #두 행렬간의 곱(내적) **행렬의 형상이 맞지 않으면 오류발생 ! ex. (3x2)행렬 x (4x2)행렬 => 에러발생 | <pre> a=np.array([[1,2],[3,4]]) b=np.array([[5,6],[7,8]]) c=np.dot(a,b) print(c) >> [[19 22] [43 50]] </pre> |
| np.maximum(x,y) | # 두 값 중 큰 값을 반환 | <pre> np.maximum(0,4) >>4 </pre> |
| np.argmax(x) | #배열의 요소 중 가장 큰 값의 인덱스 반환 **옵션 axis = 0 or 1 : 0은 행 중에 가장 큰 값, 1인 열 단위로 가장 큰 값 (2차원 이상일때 사용) ex. np.argmax(x,axis = 1) | <pre> a=np.array([3,4,1,2]) print(np.argmax(a)) </pre> |
| np.argmin(x) | #배열의 요소 중 가장 작은 값의 인덱스 반환 | <pre> a=np.array([3,4,1,2]) print(np.argmin(a)) </pre> |
| np.zeros_like(x) | #x와 형상이 같고 그 원소가 0으로 채워진 배열을 만들어 반환 | |
| 조건주기 | # 배열에서 특정 값 이상인 값만 출력하기 | <pre> a=np.array([[51,55],[14,19],[0,4]]) print(a[a>15]) >> 51, 55, 19 </pre> |
| np.sqrt | #제곱근을 계산 (= 루트, **1/2) | <pre> print(np.sqrt(2)) >>1.4142135623730951 </pre> |
| np.square | #제곱을 계산 (= **2) | <pre> print(np.square(2)) >>4 </pre> |
| np.sum(x) | # 행렬을 더한다. **옵션 axis = 0 (열끼리 더함) axis= 1(행끼리 더함) | <pre> import numpy as np b = np.array([[[1,1,1],[2,2,2]]) np.sum(b,axis=0) >>>array([3, 3, 3]) </pre> |

Matplotlib 함수

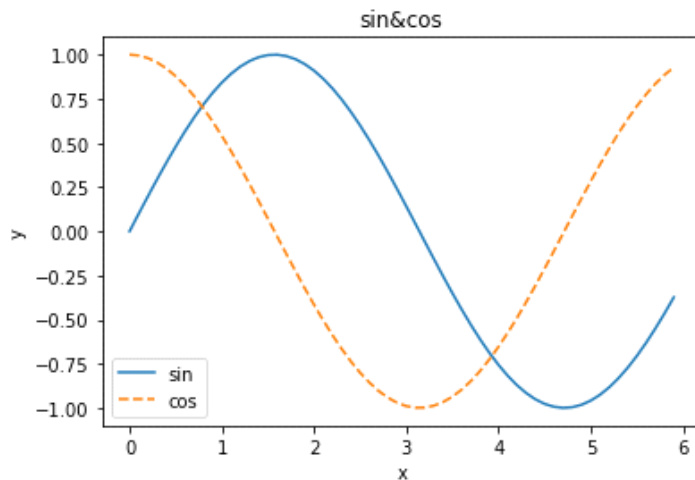
2018년 8월 15일 수요일 오후 3:36

예제 (1) matplotlib 그래프 그리기

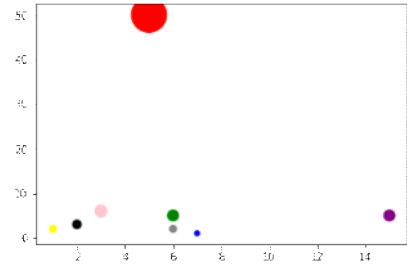
```
import matplotlib.pyplot as plt
import numpy as np

x=np.arange(0,6,0.1)
y1=np.sin(x)
y2=np.cos(x)

plt.plot(x,y1,label='sin')
plt.plot(x,y2,linestyle='--',label='cos')
plt.xlabel("x")
plt.ylabel("y")
plt.title('sin&cos')
plt.legend()
plt.show()
```



| 함수명 | 설명 | 예시 |
|-------------------|--|---|
| plt.plot(x,y) | #라인그래프 생성 x,y축을 가진 그래프를 생성한다. **옵션 - label = '그래프이름' # legend() 함수와 함께 쓴다. - linestyle = '--' # 줄을 점선으로 그린다. | import matplotlib.pyplot as plt import numpy as np x=np.arange(0,6,0.1) y1=np.sin(x) y2=np.cos(x) plt.plot(x,y1,label='sin') |
| plt.scatter(x,y) | #산포도 그래프 생성 | plt.scatter(x,y) |
| plt.hist(x) | #히스토그램 그래프 생성 **옵션 - bins = 숫자 # ? | plt.hist(x) |
| plt.bar(x) | | |
| [옵션] s (sizes) | # 그래프의 크기조절 ex) 산포도 그래프에서 각 점의 크기에 비례하게 사이즈를 주고 싶으면 | import matplotlib.pyplot as plt import numpy as np |

| | | |
|--------------------------------|--|---|
| | <p>s 옵션에 y축데이터[np 배열]*숫자를 준다. $s = y * 20$</p>  | <pre>x=np.array([5,6,7,1,2,3,6,15]) y=np.array([50,5,1,2,3,6,2,5]) print(y*2) plt.scatter(x,y,s=y*20)</pre> |
| [옵션] c | <p>#그래프의 색 설정 배열을 통해 순서대로 줄 수 도 있고, 딕셔너리를 통해 매칭을 시켜줄 수 있다.</p>  | <pre>col=['red','green','blue','yellow','black','pink','gray','purple'] plt.scatter(x,y,s=y*20,c=col)</pre> |
| plt.xlabel(x) plt.ylabel(x) | # x, y축 제목 설정 | plt.xlabel("x") plt.ylabel("y") |
| plt.title(x) | #그래프의 상단 위치한 제목 설정 | plt.title('sin&cos') |
| plt.legend() | #그래프의 범례 출력 | plt.legend() |
| plt.show() | # 생성된 그래프를 보여줌 | plt.show() |
| plt.grid() | #그래프의 배경에 세로,가로 줄 생성 | plt.grid() |
| plt.figure([x]) | <p># 그래프를 두 개이상 다른 윈도우 창에 출력 plt.figure(1)과 같이 정의하면 그 뒤의 그래프는 1번 그래프에 그려진다.</p>  <p>**옵션</p> <ul style="list-style-type: none"> - dpi=숫자 : 그래프의 해상도 조절 - facecolor ='색' : 그래프 외곽의 색깔 설정 | <pre>plt.figure(1,facecolor='green') plt.scatter(x,y,s=y*20,c=col) plt.figure(2) plt.plot(y,x)</pre> |
| 한글깨짐방지 | #matplotlib 한글깨짐 방지 | <pre>### matplotlib 한글깨짐 방지 from matplotlib import font_manager, rc font_name = font_manager.FontProperties(fname=" c:/Windows/Fonts/malgun.ttf").get_name() rc('font', family=font_name)</pre> |

Linear Regression

2018년 8월 16일 목요일 오전 11:11

1. regression이란?

직역하면 회귀. 그래프상에 무작위하게 흩어져 있는 데이터의 경향을 찾아 선으로 나타내는 것
선이 직선형태일 경우 **linear regression**이라고 하며 여기서는 예측(prediction)의 한 방법을 뜻함.
○

예제 1. 아래와 같은 데이터가 주어졌다고 하면,

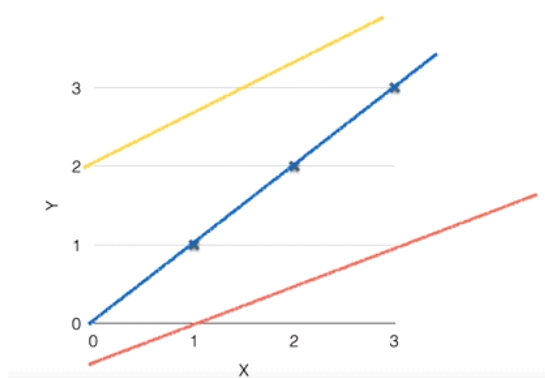
Regression (data)

| x | y |
|---|---|
| 1 | 1 |
| 2 | 2 |
| 3 | 3 |

사람의 경우 직관적으로 $y=x$ 라는 방정식을 세울 수 있다.

이러한 직관을 컴퓨터에 주입해 기계적인 방식으로 regression이 가능하게 하는 것이 목적.

(Linear) Hypothesis



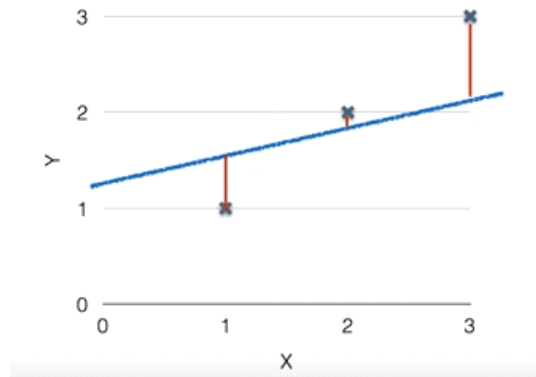
주어진 데이터를 표현하는 여러 가설(Hypothesis)이 존재함 (각 직선)

직선이므로 $H(x) = Wx + b$ 로 표현가능

W 는 weight (가중치), b 는 bias (편향)을 뜻한다.

이 가설에서 **가장 잘 들어 맞는 w (가중치)** 를 찾아보자.

** '가장 잘 들어 맞는'의 정의는?



각 데이터와 가설간의 y값 차이를 구한 평균 = 오차

즉, $H(x) - y$ 의 값이 가장 작을 때를 의미한다.

하지만 이는 +도 되고 -되므로 이를 보완하기 위해 제곱을 해준다. 제곱을 하므로서 부호문제 해결과 더 큰 차이를 boosting하는 효과가 있다.

** 부호 문제를 해결하기 위해 절대값을 씌우면 안되나?

- > 절대값을 씌우게 되면 함수가 연속적이지 않다.
- > 함수가 연속적이지 않다는 것은 미분을 할 수 없다는 뜻.
- > 기울기를 구하기 위해선 미분을 할 수 있어야 한다.

따라서 $H(x) - y$ (오차)를 구하고 이를 나누어 평균을 낸 수식은 아래와 같다.

$$cost(W, b) = \frac{1}{m} \sum_{i=1}^m (H(x^{(i)}) - y^{(i)})^2$$

이를 cost function 또는 loss function 이라고도 한다.

cost가 0에 갈 수록 가설과 데이터가 정확히 일치 한다는 뜻이다.

cost함수에 들어가는 인자는 W,b 인데 이를 미세하게 조정해가며 cost함수 결과를 0에 가까워지게 해주어야 한다.

■ cost가 0에 가까워 지는 점을 어떻게 찾는가?

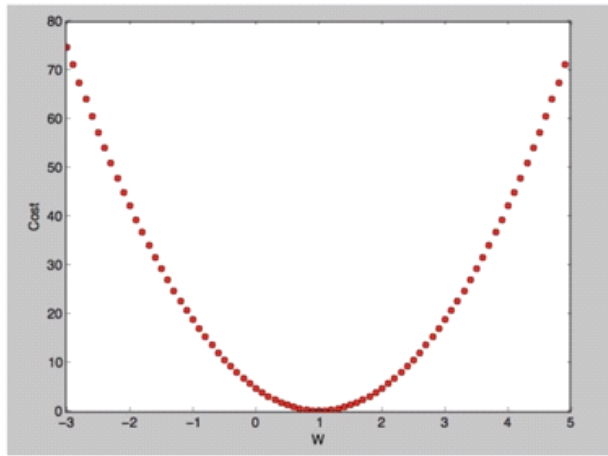
일단 b는 없고 $H(x) = Wx$ 라고 가정하면,

$$W := W - \alpha \frac{\partial}{\partial W} cost(W)$$

위의 식을 사용해서 cost가 0에 가까워 지는 점을 찾는다.

cost 함수의 'W'에 대해 미분한 값 => 기울기에 해당

원래 W에서 기울기를 빼줌



ex. w(가중치)가 -1일 경우 --> cost는 20이 되고 오른쪽으로 나아 가야함

w가 -1일때의 기울기는 음수가 나온다. 아까 w에서 기울기를 빼준다고 했는데 기울기가 음수 이므로 w가 오른쪽으로 이동하게 됨.

이렇게 w에 대한 미분값을 통해 cost의 저점을 찾아가는 방식을 Gradient Descent 방식이라고 함.

$$W := W - \alpha \frac{\partial}{\partial W} \frac{1}{2m} \sum_{i=1}^m (Wx^{(i)} - y^{(i)})^2$$

$$W := W - \alpha \frac{1}{2m} \sum_{i=1}^m 2(Wx^{(i)} - y^{(i)})x^{(i)}$$

$$W := W - \alpha \frac{1}{m} \sum_{i=1}^m (Wx^{(i)} - y^{(i)})x^{(i)}$$

일반적으로 cost함수에 1/2을 곱해줘서, 미분했을 때 값이 깔끔하게 정리되게 함

어차피 상수를 곱해주는 건 별 의미없음

위에서 마지막 식이 GD 식에 해당함.

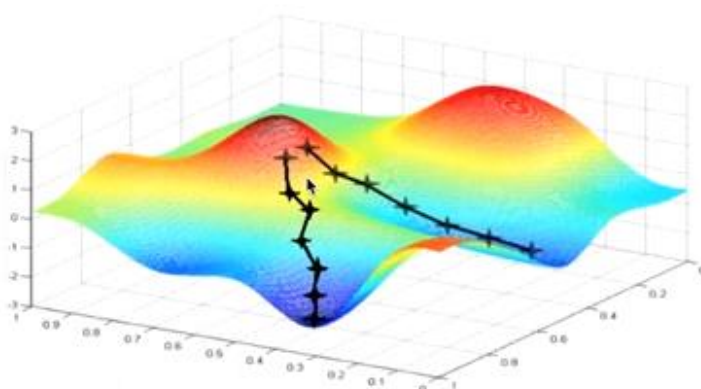
:= 는 대입한다는 뜻

저기서 α 는 얼마만큼의 강도로 이동할 지를 결정짓는 계수에 해당

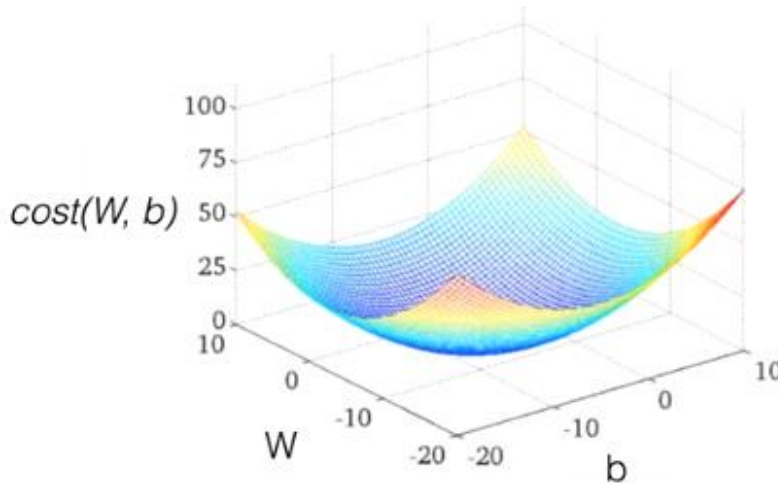
learning rate라 함

α 가 너무 크면 지나치게 많이 이동하므로 W가 0에 수렴하지 않고 발산하게 됨

보통 0.1을 기본으로 잡고 점점 줄여나감



그런데 W 와 b 에 따른 cost값이 위와 같이 표현된다고 하면
 어디에서 시작해서 내려가느냐에 따라서 도착지점이 달라짐
 즉 greedy하게 진행하므로 global minimum이 아닌 local minimum에 빠질 수 있음
 이것이 GD의 단점.



따라서 **cost함수의 형태가 위와 같이 convex function으로 나타날 때만 GD가 제대로 먹힘**

convex는 볼록하다는 뜻참고로 오목한 함수는 concave function이라고 하는데 좀 헷갈릴 수 있음.
 그냥 오목 볼록 따지지 말고
 아래로 둥글면 convex
 위로 둥글면 concave라고 알아두면 됨

이제 여기서 수 많은 의문점들이 쏟아져나온다..
 항상 convex 함수로 나타내어 지는가?
 convex한지 아닌지 어떻게 알아내는가? 등등

예제처럼 차원이 낮은 간단한 경우에는 알아내는 것이 쉽지만
 차원이 높아지는 경우 cost 함수의 형태를 가늠하기 힘들어짐
 이것이 현재의 연구주제라고 하니 계속 파고들면 끝도 없으므로
 일단은 linear regression의 개념과 gradient descent의 개념만 짚고 넘어가자

+

y 가 단일 x 에 대한 1차식으로 나타나는 경우 linear regression이 먹힌다.

예를 들면 pc방 사용시간(x)에 따른 pc방 요금(y)

그러나 현실의 데이터 상관관계는 이렇게 단순하지만은 않다.

x_1, x_2, x_3, \dots 가 모여서 y 가 결정되는 경우 (Multiple linear regression)

또는 x 와 y 의 관계가 n 차 식으로 나타나는 경우(Nonlinear regression)

이외에도 regression의 종류는 많다.

따라서 linear regression은 그냥 방법중에 하나일뿐이며 모든 데이터 모델에 적용가능한 것은 아니다.

5. 오차역전파

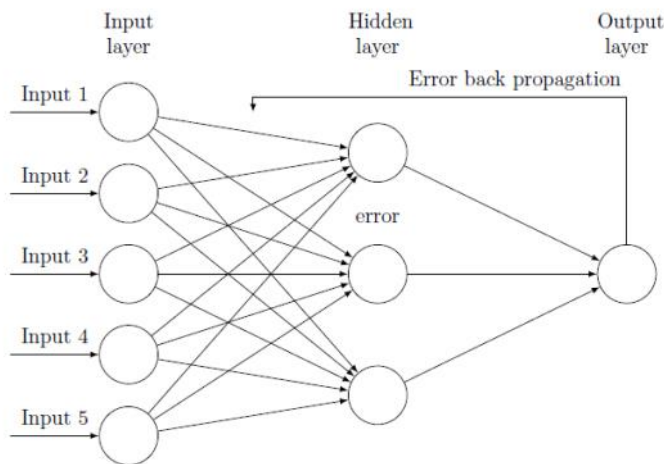
2018년 8월 25일 토요일 오후 6:40

1. 오차 역전파 Error Backpropagation

Input에서 Output으로 가중치를 업데이트하면서 활성화 함수를 통해서 결과값을 가져오는 것까지 배웠다. 이렇게 쪽 오는 것을 순전파(foward)라고 하며 말 그대로 앞쪽으로 input 값을 전파, 보내는 것이라고 보면 된다. 하지만 우리가 임의로 한 번 순전파 했다고 출력 값이 정확하지는 않을 것이다. 우리가 임의로 설정한 가중치 값이 input에 의해서 한 번 업데이트 되긴 했지만 많은 문제가 있을 수 있다.

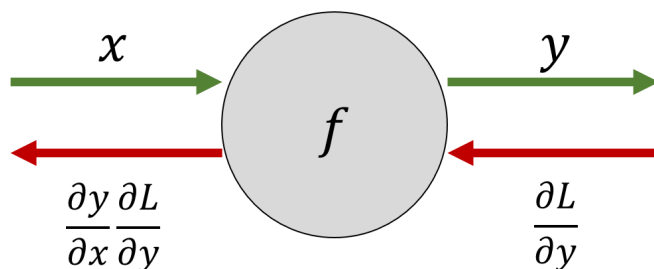
역전파란 즉, 전파된 오차를 사용해서 함수의 경사(기울기)를 계산할 수 있다.

역전파 방법은 결과 값을 통해서 다시 역으로 input 방향으로 오차를 다시 보내며 가중치를 재업데이트 하는 것이다. 물론 결과에 영향을 많이 미친 노드(뉴런)에 더 많은 오차를 돌려줄 것이다.



위의 그림을 보면 Input이 들어오는 방향(순전파)으로 output layer에서 결과 값이 나온다. 결과값은 오차(error)를 가지게 되는데 역전파는 이 오차(error)를 다시 역방향으로 hidden layer와 input layer로 오차를 다시 보내면서 가중치를 계산하면서 output에서 발생했던 오차를 적용시키는 것이다.

한 번 돌리는 것을 1 epoch 주기라고 하며 epoch를 늘릴 수록 가중치가 계속 업데이트(학습)되면서 점점 오차가 줄어 나가는 방법이다.



위의 그림에서 $\frac{\partial L}{\partial y} \frac{\partial y}{\partial x} \frac{\partial L}{\partial y}$ 의 의미에 주목할 필요가 있다. 지금은 예시이기 때문에 노드를 하나만 그렸지만, 실제 뉴럴네트워크는 이러한 노드가 꽤 많은 큰 계산그래프이다. 이 네트워크는 최종적으로는 정답과 비교한 뒤 Loss를 구한다.

우리의 목적은 뉴럴네트워크의 오차를 줄이는 데 있기 때문에, 각 파라미터별로 Loss에 대한 그래디언트를 구한 뒤 그래디언트들이 향한 쪽으로 파라미터들을 업데이트한다. $\frac{\partial L}{\partial y} \frac{\partial y}{\partial x} \frac{\partial L}{\partial y}$ 는 y 에 대한 Loss의 변화량, 즉 Loss로부터 흘러들어온 그래디언트라고 이해

하면 좋을 것 같다.

이제는 현재 입력값 x 에 대한 Loss의 변화량, 즉 $\partial L / \partial x$ 를 구할 차례이다. 이는 **미분의 연쇄법칙(chain rule)**에 의해 다음과 같이 계산할 수 있다.

$$\partial L / \partial x = \partial y / \partial x \partial L / \partial y = \partial y / \partial x \partial L / \partial y$$

이미 설명했듯 $\partial L / \partial y$ 는 Loss로부터 흘러들어온 그래디언트이다. $\partial y / \partial x$ 는 현재 입력값에 대한 현재 연산결과의 변화량, 즉 **로컬 그래디언트(Local Gradient)**이다.

다시 말해 현재 입력값에 대한 Loss의 변화량은 Loss로부터 흘러들어온 그래디언트에 로컬 그래디언트를 곱해서 구한다는 이야기이다. 이 그래디언트는 다시 앞쪽에 배치돼 있는 노드로 역전파된다.

■ 오차 역전파법

"오차 함수를 미분해서 기울기를 가지고 가중치와 바이어스를 갱신하여 최적의 가중치와 바이어스를 지닌 신경망을 만드는게 목표 "

장점 : 구현이 간단하다.

단점 : 시간이 오래 걸린다.

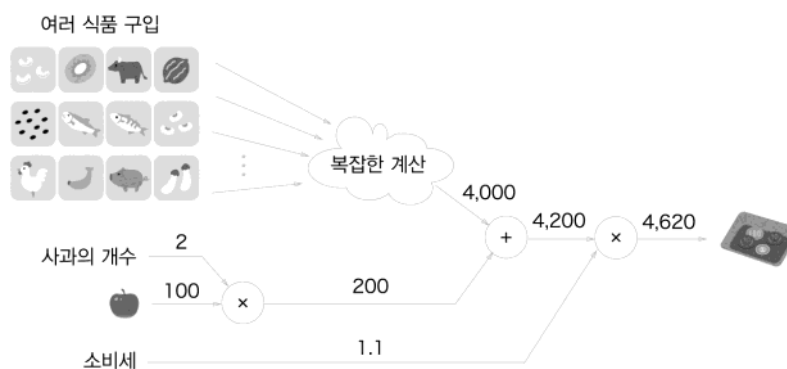
■ 계산 그래프

" 순전파와 역전파에 계산 과정을 그래프로 나타내는 방법 "

- 계산 그래프의 장점이 무엇인가?

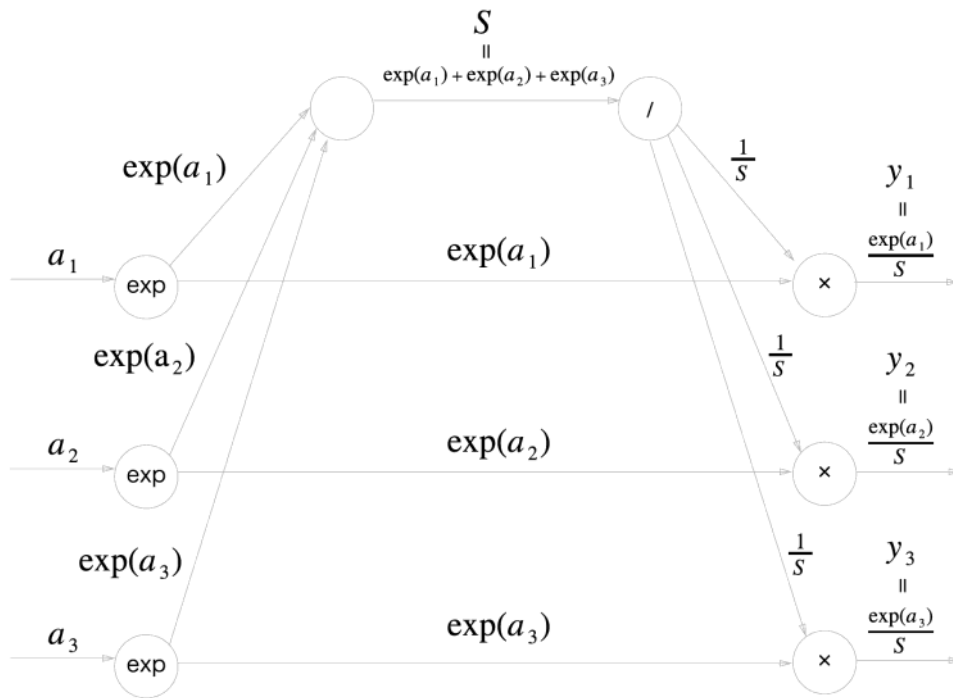
국소적 계산을 할 수 있다.

국소적 계산이란 전체에 어떤일이 벌어지든 상관없이 자신과 관계된 정보만으로 다음 결과를 출력 할 수 있다는 것이다. -->전체가 아무리 복잡해도 각 노드에서 단순한 계산에 집중하여 문제를 단순화 시킬 수 있다.

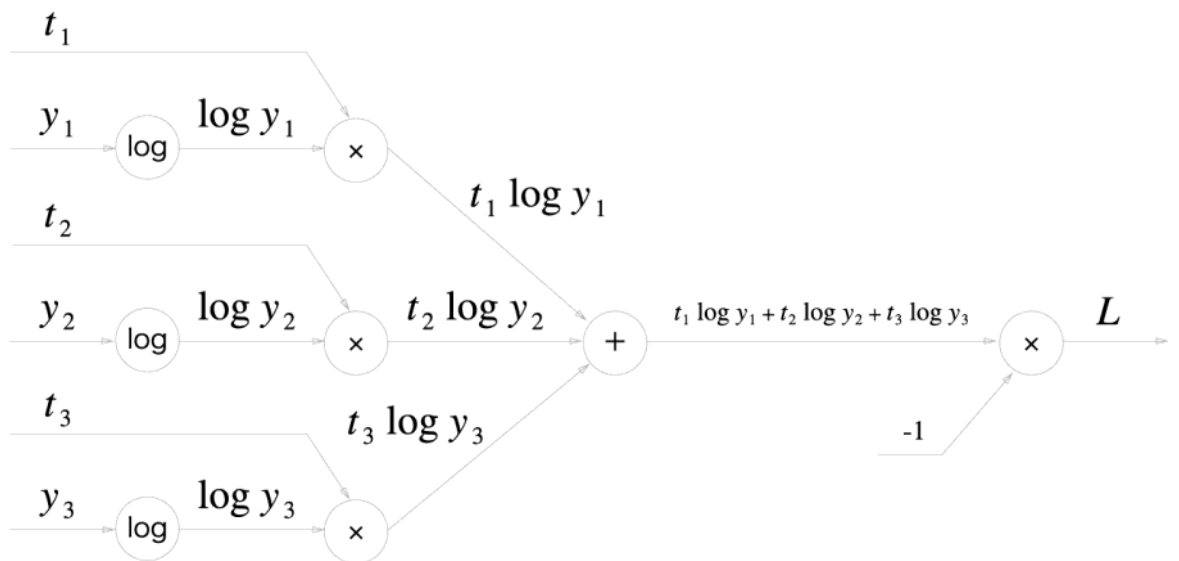


4000원이라는 숫자가 어떻게 계산되었느냐와는 상관없이 사과가 어떻게 200원이 되었는가만 신경쓰면 된다는 것이 국소적 계산이다.

**soft max 함수 계산 그래프



****교차 엔트로피 함수 계산 그래프**



■ 활성화 함수 계층 구현하기

예제(1) : mask = (x<=0) 소스 이해하기

relu 클래스를 이해하려면 두 가지를 알아야 한다.

1. copy의 의미
 2. $x[x \leq 0]$ 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 # true 인 곳에만 0을 넣는다.
print(out)

```

****결과**

```

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

```

■ OrderedDict() 함수의 이해

orderDict은 그냥 dictionary와는 다르게 입력되서 데이터 뿐만 아니라 입력된 순서까지 같아야 동일한 것으로 판단한다..

예제 (1) : 일반 딕셔너리 사용 했을 때 순서가 같은지?

```

import collections

print (' dict :')

d1 = {}
d1['a']='A'
d1['b']='B'
d1['c']='C'
d1['d']='D'
d1['e']='E'

d2 = {}
d2['e']='E'
d2['d']='D'
d2['c']='C'
d2['b']='B'
d2['a']='A'

print (d1)
print (d2)
print (d1 == d2) # 딕셔너리는 순서가 없기 때문에 True

```

****결과**

```

dict :
{'a': 'A', 'b': 'B', 'c': 'C', 'd': 'D', 'e': 'E'}
{'e': 'E', 'd': 'D', 'c': 'C', 'b': 'B', 'a': 'A'}
True

```

예제 (2) : OrderedDict는 순서가 존재하는지?

```

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) # OrderedDict는 순서가 있기 때문에 False

```

****결과**

```

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 되기 때문에 orderedDict 함수를 사용해야 한다.**

#순전파

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

#역전파

소프트맥스 --> Affine2 계층 --> 시그모이드 --> Affine1 계층

■ 오차 역전파를 이용한 2층 신경망 전체 코드

클래스 이름 : TwoLayerNet

1. 가중치와 바이어스를 초기화 하는 함수 (__init__)
2. 순전파를 진행하는 함수 (predict)
3. 비용 (오차)를 출력하는 함수 (loss)
4. 정확도를 출력하는 함수 (accuracy)
5. 오차역전파를 진행하는 함수 (gradient)

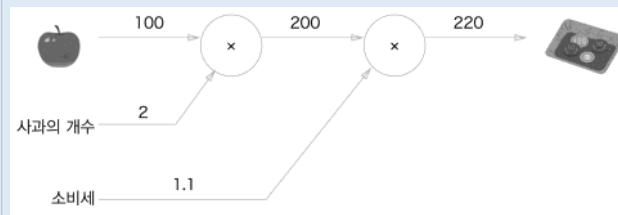
순전파(predict) ----->

<----- 역전파(gradient) 비용 (loss) 정확도 (accuracy)

가중치, 바이어스 (__init__) 갱신

문제 136.

위에서 만든 곱셈 클래스를 객체화 시켜서 사과 가격의 총 가격을 구하시오.



```
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
        dy = dout * self.x
```

```
    return dx, dy
```

```
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)
```

****결과**

```
220.00000000000003
```

문제 137.

덧셈 계층을 파이썬으로 구현 하시오.

```
class AddLayer:
```

```
    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          # 덧셈 노드는 상류값을 여과없이 하류로 흘려보냄
```

```
    dy = dout
```

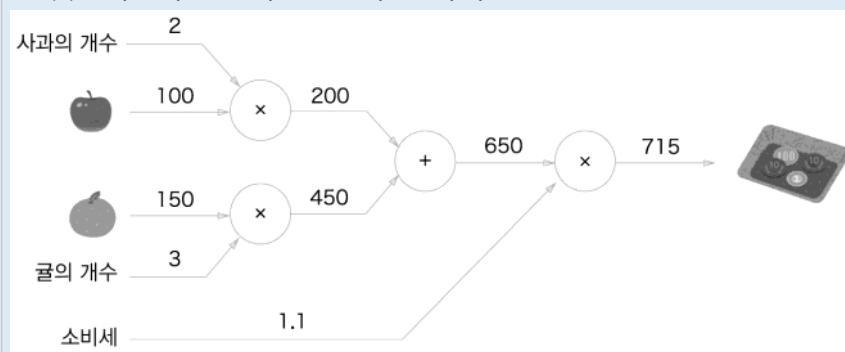
```
    return dx, dy
```

**결과

220.00000000000003

문제 138.

층을 4개로 해서 사과와 귤의 총 가격을 구하는 책의 149쪽의 그림 5-3의 신경망을 위에서 만든 덧셈계층과 곱셈계층으로 구현 하시오.



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
```

```
    dy = dout * self.x
```

```
    return dx, dy
```

class AddLayer:

```
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
    dy = dout

    return dx, dy

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

mul_apple_layer = MulLayer()
mul_orange_layer = MulLayer()
add_apple_orange_layer = AddLayer()
mul_tax_layer = MulLayer()

apple_p=mul_apple_layer.forward(apple,apple_num)
orange_p=mul_orange_layer.forward(orange,orange_num)
all_p=add_apple_orange_layer.forward(apple_p,orange_p)

total_p = mul_tax_layer.forward(all_p,tax)

print(total_p)

**결과
715.0000000000001

```

| | |
|----------------|---|
| 문제 139. | 문제 138번의 순전파로 출력한 과일 가격의 총합 신경망의 역전파를 구현 하시오. |
|----------------|---|

```

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
        dy = dout * self.x

        return dx, dy

class AddLayer:
    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
    dy = dout

    return dx, dy

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

#순전파
mul_apple_layer = MulLayer()
mul_orange_layer = MulLayer()
add_apple_orange_layer = AddLayer()
mul_tax_layer = MulLayer()

apple_p=mul_apple_layer.forward(apple,apple_num)
orange_p=mul_orange_layer.forward(orange,orange_num)
all_p=add_apple_orange_layer.forward(apple_p,orange_p)
total_p = mul_tax_layer.forward(all_p,tax)

print(total_p)

#역전파
d_price=1 #역전파 시작은 1
d_all_price, d_tax = mul_tax_layer.backward(d_price)
d_apple_price,d_orange_price=add_apple_orange_layer.backward(d_all_price)
d_apple,d_apple_num=mul_apple_layer.backward(d_apple_price)
d_orange,d_orange_num=mul_orange_layer.backward(d_orange_price)

print(d_apple_num,d_apple,d_orange,d_orange_num,d_tax )

**결과
715.0000000000000001
110.0000000000000001 2.2 3.300000000000000003 165.0 650

```

| | |
|---------|---------------------------|
| 문제 140. | 책 166p의 Relu 클래스를 생성 하시오. |
|---------|---------------------------|

```

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
        dy = dout * self.x

```

```

        return dx, dy

class AddLayer:
    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
        dy = dout

        return dx, dy

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

#순전파
mul_apple_layer = MulLayer()
mul_orange_layer = MulLayer()
add_apple_orange_layer = AddLayer()
mul_tax_layer = MulLayer()

apple_p=mul_apple_layer.forward(apple,apple_num)
orange_p=mul_orange_layer.forward(orange,orange_num)
all_p=add_apple_orange_layer.forward(apple_p,orange_p)
total_p = mul_tax_layer.forward(all_p,tax)

print(total_p)

#역전파
d_price=1 #역전파 시작은 1
d_all_price, d_tax = mul_tax_layer.backward(d_price)
d_apple_price,d_orange_price=add_apple_orange_layer.backward(d_all_price)
d_apple,d_apple_num=mul_apple_layer.backward(d_apple_price)
d_orange,d_orange_num=mul_orange_layer.backward(d_orange_price)

print(d_apple_num,d_apple,d_orange,d_orange_num,d_tax )

**결과
715.00000000000001
110.00000000000001 2.2 3.3000000000000003 165.0 650

```

| | |
|----------------|---|
| 문제 141. | Relu 클래스를 객체화 시켜서 아래의 입력 데이터 x값을 받아서 출력되는 순전파 행렬을 출력 하시오. |
|----------------|---|

```

class Relu :
    def __init__(self):

```

```

self.mask=None

def forward(self, x):
    self.mask = (x<=0)
    out = x.copy()
    out[self.mask]=0
    return out

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

    return dx

rl= Relu()
print(rl.forward(x))

**결과
[[1. 0.]
 [0. 3.]]

```

| | |
|----------------|--|
| 문제 142. | Relu 클래스를 객체화 시켜서 아래의 입력 데이터 x 값을 받아서 출력되는 역전파 행렬을 출력 하시오. |
|----------------|--|

```

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

    def forward(self, x):
        self.mask = (x<=0)
        out = x.copy()
        out[self.mask]=0
        return out

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

        return dx

rl= Relu()
fw=rl.forward(x)
print(rl.backward(fw))

**결과
[[1. 0.]
 [0. 3.]]

```

| | |
|----------------|--|
| 문제 144. | 아래의 두 행렬의 내적과 바이어스를 더한 값이 무엇인지 파이썬으로 구현 하시오. |
|----------------|--|

$$y = (x \bullet w) + b$$

```
import numpy as np

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

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

****결과**

[6 13 20]

문제 145. 위의 순전파의 역전파된 최종값인 dX와 dW를 각각 구하시오.

```
import numpy as np

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

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

print('dX:', np.dot(dy, W.T))
print('dW:', np.dot(X.T, dy))
```

****결과**

dX: [[22 28]]

dW: [[1 2 3]

[2 4 6]]

문제 146. 아래의 행렬을 입력받아 결과를 출력하는 forward 함수를 생성 하시오.

$$y = (x \bullet w) + b$$

```
import numpy as np

def forward(x,w,b):
    return np.dot(x,w)+b

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))
```

****결과**

[[6 12 18]]

문제 147. 지금 위의 코드는 mnist로 치면 딱 한 장의 이미지만 입력되는 코드이다. 아래와 같이 batch로 처리할 수 있도록 2장의 이미지로 처리되는 코드로 구현 하시오.

```
import numpy as np
```

```
def forward(x,w,b):
    return np.dot(x,w)+b

X = np.array([[1,2],[3,4]]) # [1,2] , [3,4] 각각 하나
W = np.array([[1, 3, 5], [2, 4, 6]])
B = np.array([[1, 1, 1]])

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

****결과**

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

문제 148. 문제 147번에 대한 역전파 함수를 backward라는 이름으로 생성 하시오.

```
import numpy as np

def forward(x,w,b):
    return np.dot(x,w)+b

def backward(x,w,dy):
    dx = np.dot(dy, w.T)
    dw = np.dot(x.T, dy)
    print('dx (입력값에 대한 역전파) : ',dx)
    print(('dw(가중치에 대한 역전파) : ',dw))
```

```
X = np.array([[1,2],[3,4]]) # 2x2
W = np.array([[1, 3, 5], [2, 4, 6]]) # 2x3
B = np.array([[1, 1, 1]])
dY = np.array([[1,1,1],[2,2,2]])
```

```
backward(X,W,dY)
```

****결과**

```
dx (입력값에 대한 역전파) : [[ 9 12]
 [18 24]]
('dw(가중치에 대한 역전파) : ', array([[ 7,  7,  7],
 [10, 10, 10]]))
```

문제 149. 아래 행렬의 열을 더한 결과를 아래와 같이 출력 하시오.

```
import numpy as np
b = np.array([[1,1,1],[2,2,2]])

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

****결과**

```
array([3, 3, 3])
```

문제 149+ 아래의 backward 함수에 바이어스에 대한 역전파를 출력하는 코드를 추가 하시오.


```
def backward(x,w,dy):
    dx = np.dot(dy, w.T)
    dw = np.dot(x.T, dy)
    db = np.sum(dy,axis=0)

    print('입력값에 대한 역전파(dx) :',dx)
    print('가중치에 대한 역전파(dw) :',dw)
    print('바이어스에 대한 역전파(db) :',db)

X = np.array([[1,2],[3,4]]) # 2x2
W = np.array([[1, 3, 5], [2, 4, 6]]) # 2x3
B = np.array([[1, 1, 1]])
dY = np.array([[1,1,1],[2,2,2]])

backward(X,W,dY)
```

****결과**

```
입력값에 대한 역전파(dx) : [[ 9 12]
 [18 24]]
가중치에 대한 역전파(dw) : [[ 7  7  7]
 [10 10 10]]
바이어스에 대한 역전파(db) : [3 3 3]
```

문제 150. 책 175페이지 아래에 나오는 Affine 클래스를 구현 하시오.

```
import numpy as np

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

    def forward(self, x):
        self.x = x
        y = np.dot(x, self.W) + self.b

        return y

    def backward(self, dy):
        dx = np.dot(dy, self.W.T)
        dW = np.dot(self.x.T, dy)
        db = np.sum(dy, axis=0)

        return dx, dW, db

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

affine1 = Affine(w, b) # 가중치와 바이어스를 가지고
# 신경망 객체 affine1 생성
```

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

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

**결과

```
[[ 6 12 18]
```

```
 [12 26 40]]
```

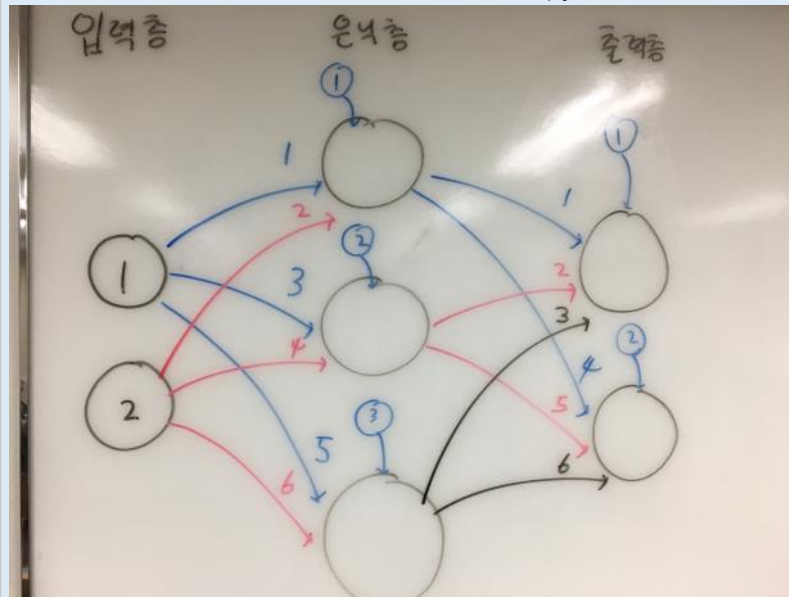
```
(array([[ 9, 12],
```

```
      [18, 24]]), array([[ 7,  7,  7],
```

```
      [10, 10, 10]]), array([3, 3, 3]))
```

문제 151.

아래 그림의 신경망의 x, w_1, w_2, b_1, b_2 의 numpy 배열을 만드시오.



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

```
w1 = np.array([[1,3,5],[2,4,6]])
```

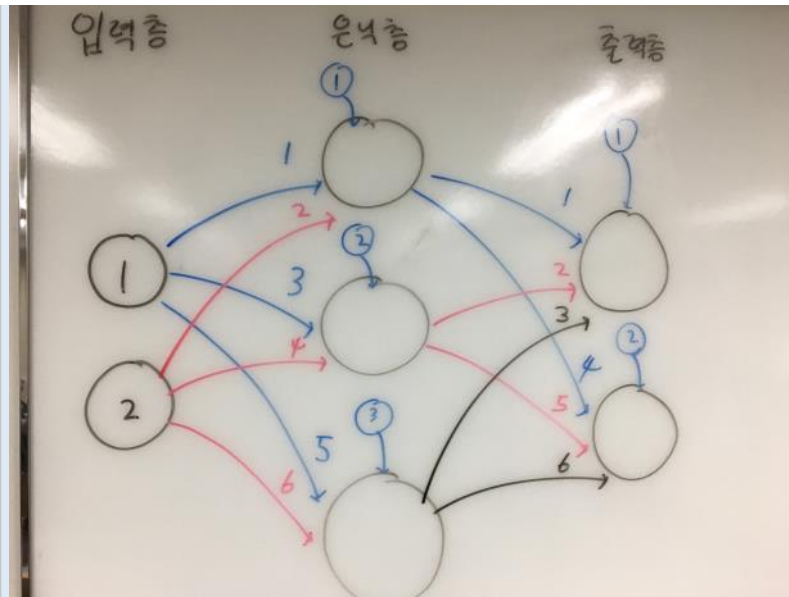
```
w2 = np.array([[1,4],[2,5],[3,6]])
```

```
b1 = np.array([[1,2,3]])
```

```
b2 = np.array([[1,2]])
```

문제 152.

Affine 클래스를 이용해서 2층 신경망의 순전파를 구현하시오.



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

    def forward(self, x):
        self.x = x
        y = np.dot(x, self.W) + self.b
        return y

    def backward(self, dy):
        dx = np.dot(dy, self.W.T)
        dW = np.dot(self.x.T, dy)
        db = np.sum(dy, axis=0)
        return dx, dW, db

x = np.array([[1,2]])
w1 = np.array([[1,3,5],[2,4,6]])
w2 = np.array([[1,4],[2,5],[3,6]])
b1 = np.array([[1,2,3]])
b2 = np.array([[1,2]])

layer_1 = Affine(w1,b1)
layer_2 = Affine(w2,b2)

r1 = layer_1.forward(x)
r2 = layer_2.forward(r1)

print(r2)

**결과
[[ 93 211]]
```

문제 153. 위의 2층 신경망의 역전파를 Affine 클래스를 이용해서 구현 하시오.
아래 backward에 입력할 dy는 위의 순전파의 결과인 r2 행렬을 그대로 사용하시오.

```
class Affine:
```

```

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

def forward(self, x):
    self.x = x
    y = np.dot(x, self.W) + self.b
    return y

def backward(self, dy):
    self.dx = np.dot(dy, self.W.T)
    self.dW = np.dot(self.x.T, dy)
    self.db = np.sum(dy, axis=0)
    return self.dx

x = np.array([[1,2]])
w1 = np.array([[1,3,5],[2,4,6]])
w2 = np.array([[1,4],[2,5],[3,6]])
b1 = np.array([[1,2,3]])
b2 = np.array([[1,2]])

layer_1 = Affine(w1,b1)
layer_2 = Affine(w2,b2)

r1 = layer_1.forward(x)
r2 = layer_2.forward(r1)

print('r1 : ',r1) #순전파 결과
print('r2 : ',r2,end='\n\n') #순전파 결과

l2_back_rst = layer_2.backward(r2)
print(l2_back_rst)
l1_back_rst = layer_1.backward(l2_back_rst)

print('db1 : ', l2_back_rst)
print('dx : ',l1_back_rst)
print('dw1 : ',layer_1.dW)

**결과
r1 : [[ 6 13 20]]
r2 : [[ 93 211]]
[[ 937 1241 1545]]
db1 : [[ 937 1241 1545]]
dx : [[12385 16108]]
dw1 : [[ 937 1241 1545]
[1874 2482 3090]]

```

| | |
|----------------|--|
| 문제 154. | 위의 2층 신경망의 순전파를 은닉층에 활성화 함수로 Relu를 추가해서 구현 하시오. (어제 만들었던 Relu 클래스를 이용해서 구현) |
|----------------|--|

```

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

    def forward(self, x):
        self.x = x
        y = np.dot(x, self.W) + self.b
        return y

    def backward(self, dy):
        self.dx = np.dot(dy, self.W.T)
        self.dW = np.dot(self.x.T, dy)
        self.db = np.sum(dy, axis=0)
        return self.dx

x = np.array([[1,2]])
w1 = np.array([[1,3,5],[2,4,6]])
w2 = np.array([[1,4],[2,5],[3,6]])
b1 = np.array([[1,2,3]])
b2 = np.array([[1,2]])

layer_1 = Affine(w1,b1)
l1_relu= Relu()

layer_2 = Affine(w2,b2)
l2_relu= Relu()

r1 = l1_relu.forward(layer_1.forward(x))
r2 = l2_relu.forward(layer_2.forward(r1))

print('r1 : ',r1) #순전파 결과
print('r2 : ',r2,end='\n\n') #순전파 결과

l2_back_rst = layer_2.backward(l2_relu.backward(r2))
l1_back_rst = layer_1.backward(l1_relu.backward(l2_back_rst))

print('db1 : ', l2_back_rst)
print('dx : ',l1_back_rst)
print('dw1 : ',layer_1.dW)

**결과
r1 : [[ 6 13 20]]
r2 : [[ 93 211]]
db1 : [[ 937 1241 1545]]
dx : [[12385 16108]]
dw1 : [[ 937 1241 1545]
[1874 2482 3090]]

```

| | |
|----------------|-------------------------------------|
| 문제 155. | Softmax-with-Loss 계층을 클래스로 구현해 보시오. |
|----------------|-------------------------------------|

```

class SoftmaxWithLoss:
    def __init__(self):

```

```

self.loss=None
self.y=None
self.t=None

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 156. 위에서 만든 softmaxwithloss클래스를 객체화 시켜서 아래의 x(입력값), t(target value)를 입력해서 순전파의 오차율을 출력하고 역전파도 출력 하시오.

```

import numpy as np

class SoftmaxWithLoss:
    def __init__(self):
        self.loss=None
        self.y=None
        self.t=None

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

    def cross_entropy_error(self,y,t):
        delta = 1e-7 #아주 작은 수
        return -np.sum(t*np.log(y+delta))

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

t = np.array([0,0,1,0,0,0,0,0,0]) # 숫자2

x = np.array([0.01,0.01,0.01,0.01,0.01,0.01,0.05,0.3,0.1,0.5])

x2 = np.array([0.01,0.01,0.9,0.01,0.01,0.01,0.01,0.01,0.01,0.02])

sfm = SoftmaxWithLoss()
y1=sfm.forward(x,t)
print(y1)
x1_back= sfm.backward(y1)

```

```
print(x1_back,end='\n\n')
```

```
sfm2 = SoftmaxWithLoss()
y2=sfm2.forward(x2,t)
print(y2)
x2_back= sfm2.backward(y2)
print(x2_back)
```

****결과**

2.4072798663898216

[0.00900598 0.00900598 -0.09099402 0.00900598 0.00900598 0.00900598
 0.00937352 0.01203584 0.00985412 0.01470061]

1.5475681948007376

[0.0087373 0.0087373 -0.07872354 0.0087373 0.0087373 0.0087373
 0.0087373 0.0087373 0.0087373 0.00882511]

문제 158. 책 181p부터 183p 까지 나오는 TwoLayerNet 클래스를 생성 하시오.

```
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 TwoLayersNet:
```

```
#가중치와 바이어스를 랜덤으로 초기화하는 함수
```

```
def __init__(self, input_size, hidden_size, output_size, weight_init_std=0.02):
```

```
    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
```

```

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

#정확도 구하는 함수
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

#수치 미분하는 함수
def numerical_gradient(self, x, t):
    loss_W = lambda W: self.loss(x, t)

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

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)

    grads = {}
    grads['W1'] = self.layers['Affine1'].dW
    grads['b1'] = self.layers['Affine1'].db
    grads['W2'] = self.layers['Affine1'].dW
    grads['b2'] = self.layers['Affine1'].db

    return grads

tolayernet1=TwoLayersNet(input_size=2, hidden_size=50, output_size=2, weight_init_std=0.02)
x = np.array([[1,2]])
t= np.array([[0,1]])
print(tolayernet1.predict(x))
print(tolayernet1.gradient(x,t))

**결과
[[ 0.00055182 -0.00106967]]
{'W1': array([[ 0.         ,  0.         ,  0.         ,  0.01091851, -0.00194719,
                0.         ,  0.         ,  0.         ,  0.         , -0.00358164,
                0.         ,  0.00037814,  0.         , -0.00550006, -0.00167984,
                0.00540526,  0.         ,  0.         ,  0.         ,  0.         , -0.01040314,

```



```

0.      , 0.      , 0.      , 0.00920718, 0.      ,
0.      , -0.00565664, -0.01436085, 0.      , 0.      ,
0.      , 0.      , 0.      , 0.00276956, -0.00780347,
0.00734415, -0.0055347 , 0.00279121, 0.      , 0.      ,
-0.00979602, 0.      , 0.      , 0.      , 0.02487008,
0.01671949, 0.      , 0.00873867, -0.00219583, 0.00332157],
[ 0.      , 0.      , 0.      , 0.02183702, -0.00389437,
0.      , 0.      , 0.      , 0.      , -0.00716328,
0.      , 0.00075628, 0.      , -0.01100012, -0.00335969,
0.01081051, 0.      , 0.      , 0.      , -0.02080629,
0.      , 0.      , 0.      , 0.01841435, 0.      ,
0.      , -0.01131328, -0.0287217 , 0.      , 0.      ,
0.      , 0.      , 0.      , 0.00553912, -0.01560695,
0.0146883 , -0.01106939, 0.00558242, 0.      , 0.      ,
-0.01959205, 0.      , 0.      , 0.      , 0.04974015,
0.03343898, 0.      , 0.01747734, -0.00439165, 0.00664315]]), 'b1': array([ 0.      , 0.      ,
0.      , 0.01091851, -0.00194719,
0.      , 0.      , 0.      , 0.      , -0.00358164,
0.      , 0.00037814, 0.      , -0.00550006, -0.00167984,
0.00540526, 0.      , 0.      , 0.      , -0.01040314,
0.      , 0.      , 0.      , 0.00920718, 0.      ,
0.      , -0.00565664, -0.01436085, 0.      , 0.      ,
0.      , 0.      , 0.      , 0.00276956, -0.00780347,
0.00734415, -0.0055347 , 0.00279121, 0.      , 0.      ,
-0.00979602, 0.      , 0.      , 0.      , 0.02487008,
0.01671949, 0.      , 0.00873867, -0.00219583, 0.00332157]), 'W2': array([[ 0.      , 0.      ,
0.      , 0.01091851, -0.00194719,
0.      , 0.      , 0.      , 0.      , -0.00358164,
0.      , 0.00037814, 0.      , -0.00550006, -0.00167984,
0.00540526, 0.      , 0.      , 0.      , -0.01040314,
0.      , 0.      , 0.      , 0.00920718, 0.      ,
0.      , -0.00565664, -0.01436085, 0.      , 0.      ,
0.      , 0.      , 0.      , 0.00276956, -0.00780347,
0.00734415, -0.0055347 , 0.00279121, 0.      , 0.      ,
-0.00979602, 0.      , 0.      , 0.      , 0.02487008,
0.01671949, 0.      , 0.00873867, -0.00219583, 0.00332157],
[ 0.      , 0.      , 0.      , 0.02183702, -0.00389437,
0.      , 0.      , 0.      , 0.      , -0.00716328,
0.      , 0.00075628, 0.      , -0.01100012, -0.00335969,
0.01081051, 0.      , 0.      , 0.      , -0.02080629,
0.      , 0.      , 0.      , 0.01841435, 0.      ,
0.      , -0.01131328, -0.0287217 , 0.      , 0.      ,
0.      , 0.      , 0.      , 0.00553912, -0.01560695,
0.0146883 , -0.01106939, 0.00558242, 0.      , 0.      ,
-0.01959205, 0.      , 0.      , 0.      , 0.04974015,
0.03343898, 0.      , 0.01747734, -0.00439165, 0.00664315]]), 'b2': array([ 0.      , 0.      ,
0.      , 0.01091851, -0.00194719,
0.      , 0.      , 0.      , 0.      , -0.00358164,
0.      , 0.00037814, 0.      , -0.00550006, -0.00167984,
0.00540526, 0.      , 0.      , 0.      , -0.01040314,
0.      , 0.      , 0.      , 0.00920718, 0.      ,
0.      , -0.00565664, -0.01436085, 0.      , 0.      ,
0.      , 0.      , 0.      , 0.00276956, -0.00780347,
0.00734415, -0.0055347 , 0.00279121, 0.      , 0.      ,
-0.00979602, 0.      , 0.      , 0.      , 0.02487008,
0.01671949, 0.      , 0.00873867, -0.00219583, 0.00332157])}]

```

문제 159. 위의 2층 신경망 클래스를 mnist 데이터로 구현하는 최종 코드를 책 186p를 보고 작성 하시

```

# coding: utf-8
import sys, os

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

class TwoLayerNet:
    def __init__(self, input_size, hidden_size, output_size, weight_init_std=0.01):
        # 가중치 초기화
        self.params = {}
        self.params['W1'] = weight_init_std * np.random.randn(input_size, hidden_size)
        self.params['b1'] = np.zeros(hidden_size)
        self.params['W2'] = weight_init_std * np.random.randn(hidden_size, output_size)
        self.params['b2'] = np.zeros(output_size)

        # 계층 생성
        self.layers = OrderedDict()
        self.layers['Affine1'] = Affine(self.params['W1'], self.params['b1'])
        self.layers['Relu1'] = Relu()
        self.layers['Affine2'] = Affine(self.params['W2'], self.params['b2'])

        self.lastLayer = SoftmaxWithLoss()

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

        return x

    # x : 입력 데이터, t : 정답 레이블
    def loss(self, x, t):
        y = self.predict(x)
        return self.lastLayer.forward(y, t)

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

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

    # x : 입력 데이터, t : 정답 레이블
    def numerical_gradient(self, x, t):
        loss_W = lambda W: self.loss(x, t)

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

```

```

        return grads

def gradient(self, x, t):
    # forward
    self.loss(x, t)

    # backward
    dout = 1
    dout = self.lastLayer.backward(dout)

    layers = list(self.layers.values())
    layers.reverse()
    for layer in layers:
        dout = layer.backward(dout)

    # 결과 저장
    grads = {}
    grads['W1'], grads['b1'] = self.layers['Affine1'].dW, self.layers['Affine1'].db
    grads['W2'], grads['b2'] = self.layers['Affine2'].dW, self.layers['Affine2'].db

    return grads

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

network = TwoLayerNet(input_size=784, hidden_size=50, output_size=10)

# 하이퍼파라미터
iters_num = 10000 # 반복 횟수를 적절히 설정한다.
train_size = x_train.shape[0] # 60000 개
batch_size = 100 # 미니배치 크기
learning_rate = 0.1

train_loss_list = []
train_acc_list = []
test_acc_list = []

# 1에폭당 반복 수
iter_per_epoch = max(train_size / batch_size, 1)
print(iter_per_epoch) # 600

for i in range(iters_num): # 10000
    # 미니배치 획득 # 랜덤으로 100개씩 뽑아서 10000번을 수행하니까 백만번
    batch_mask = np.random.choice(train_size, batch_size) # 100개 씩 뽑아서 10000번 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()

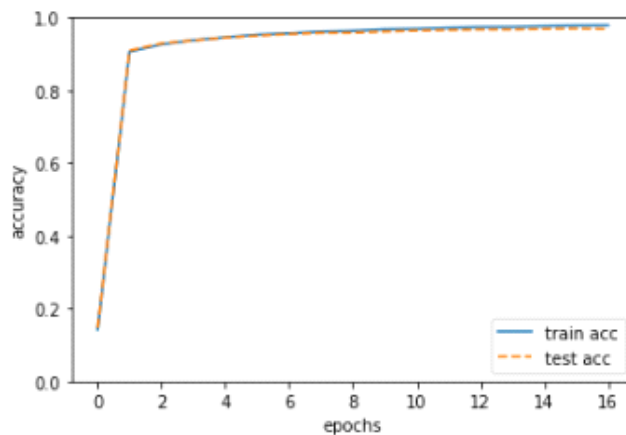
**결과

```

```

600.0
(60000, 784)
train acc, test acc | 0.14245, 0.1471
(60000, 784)
train acc, test acc | 0.9063833333333333, 0.9094
(60000, 784)
train acc, test acc | 0.92625, 0.9291
(60000, 784)
train acc, test acc | 0.93795, 0.9373
(60000, 784)
train acc, test acc | 0.94595, 0.9452
(60000, 784)
train acc, test acc | 0.9532666666666667, 0.9504
(60000, 784)
train acc, test acc | 0.9567833333333333, 0.955
(60000, 784)
train acc, test acc | 0.9616, 0.9581
(60000, 784)
train acc, test acc | 0.9634666666666667, 0.9583
(60000, 784)
train acc, test acc | 0.9678666666666667, 0.9617
(60000, 784)
train acc, test acc | 0.9692166666666666, 0.9641
(60000, 784)
train acc, test acc | 0.9717833333333333, 0.9658
(60000, 784)
train acc, test acc | 0.9740666666666666, 0.9672
(60000, 784)
train acc, test acc | 0.9745333333333334, 0.9672
(60000, 784)
train acc, test acc | 0.9768, 0.9687
(60000, 784)
train acc, test acc | 0.9785333333333334, 0.9698
(60000, 784)
train acc, test acc | 0.9792333333333333, 0.9684

```



문제 160. 위의 신경망은 2층 신경망인데 3층 신경망으로 변경 하시오.

```
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_1, hidden_size_2, output_size, weight_init_std=0.01):
    # 가중치 초기화
    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)

    # 계층 생성
    self.layers = OrderedDict()
    self.layers['Affine1'] = Affine(self.params['W1'], self.params['b1'])
    self.layers['Relu1'] = Relu()
    self.layers['Affine2'] = Affine(self.params['W2'], self.params['b2'])
    self.layers['Relu2'] = Relu()
    self.layers['Affine3'] = Affine(self.params['W3'], self.params['b3'])

    self.lastLayer = SoftmaxWithLoss()

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

    return x

# x : 입력 데이터, t : 정답 레이블
def loss(self, x, t):
    y = self.predict(x)
    return self.lastLayer.forward(y, t)

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

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

# x : 입력 데이터, t : 정답 레이블
def numerical_gradient(self, x, t):
    loss_W = lambda W: self.loss(x, t)

    grads = {}
    grads['W1'] = numerical_gradient(loss_W, self.params['W1'])
    grads['b1'] = numerical_gradient(loss_W, self.params['b1'])
    grads['W2'] = numerical_gradient(loss_W, self.params['W2'])
    grads['b2'] = numerical_gradient(loss_W, self.params['b2'])
    grads['W3'] = numerical_gradient(loss_W, self.params['W3'])
    grads['b3'] = numerical_gradient(loss_W, self.params['b3'])

    return grads

def gradient(self, x, t):
    # forward
    self.loss(x, t)

    # backward
    dout = 1
    dout = self.lastLayer.backward(dout)

```

```

layers = list(self.layers.values())
layers.reverse()
for layer in layers:
    dout = layer.backward(dout)

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

return grads

# 데이터 읽기
(x_train, t_train), (x_test, t_test) = load_mnist(normalize=True, one_hot_label=True)
network = TwoLayerNet(input_size=784, hidden_size_1=50, hidden_size_2=100, output_size=10)

# 하이퍼파라미터
iters_num = 10000 # 반복 횟수를 적절히 설정한다.
train_size = x_train.shape[0] # 60000 개
batch_size = 100 # 미니배치 크기
learning_rate = 0.1

train_loss_list = []
train_acc_list = []
test_acc_list = []

# 1에폭당 반복 수
iter_per_epoch = max(train_size / batch_size, 1)
print(iter_per_epoch) # 600

for i in range(iters_num): # 10000
    # 미니배치 획득 # 랜덤으로 100개씩 뽑아서 10000번을 수행하니까 백만번
    batch_mask = np.random.choice(train_size, batch_size) # 100개 씩 뽑아서 10000번 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', 'W3', 'b3'):
        network.params[key] -= learning_rate * grad[key]

```

```

# 학습 경과 기록
loss = network.loss(x_batch, t_batch)
train_loss_list.append(loss) # cost 가 점점 줄어드는것을 보려고

# 1에폭당 정확도 계산 # 여기는 훈련이 아니라 1에폭 되었을때 정확도만 체크

if i % iter_per_epoch == 0: # 600 번마다 정확도 쌓는다.
    print(x_train.shape) # 60000,784
    train_acc = network.accuracy(x_train, t_train)
    test_acc = network.accuracy(x_test, t_test)
    train_acc_list.append(train_acc) # 10000/600 개 16개 # 정확도가 점점 올라감
    test_acc_list.append(test_acc) # 10000/600 개 16개 # 정확도가 점점 올라감
    print("train acc, test acc | " + str(train_acc) + ", " + str(test_acc))

# 그래프 그리기
markers = {'train': 'o', 'test': 's'}
x = np.arange(len(train_acc_list))
plt.plot(x, train_acc_list, label='train acc')
plt.plot(x, test_acc_list, label='test acc', linestyle='--')
plt.xlabel("epochs")
plt.ylabel("accuracy")
plt.ylim(0, 1.0)
plt.legend(loc='lower right')
plt.show()

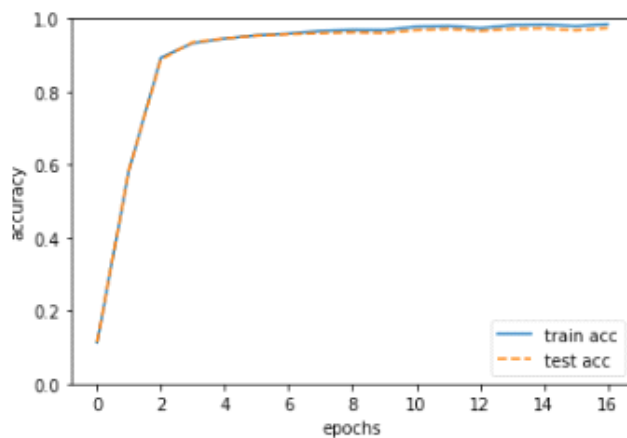
```

****결과**


```

600.0
(60000, 784)
train acc, test acc | 0.11236666666666667, 0.1135
(60000, 784)
train acc, test acc | 0.5845166666666667, 0.5851
(60000, 784)
train acc, test acc | 0.89315, 0.8881
(60000, 784)
train acc, test acc | 0.9330666666666667, 0.9356
(60000, 784)
train acc, test acc | 0.9456333333333333, 0.946
(60000, 784)
train acc, test acc | 0.9542, 0.9533
(60000, 784)
train acc, test acc | 0.9588666666666666, 0.9571
(60000, 784)
train acc, test acc | 0.96595, 0.9697
(60000, 784)
train acc, test acc | 0.9690666666666666, 0.9621
(60000, 784)
train acc, test acc | 0.9683333333333334, 0.9604
(60000, 784)
train acc, test acc | 0.9781, 0.9678
(60000, 784)
train acc, test acc | 0.9803333333333333, 0.9715
(60000, 784)
train acc, test acc | 0.9740166666666666, 0.9658
(60000, 784)
train acc, test acc | 0.9824666666666667, 0.9716
(60000, 784)
train acc, test acc | 0.9839333333333333, 0.9729
(60000, 784)
train acc, test acc | 0.97985, 0.9675
(60000, 784)
train acc, test acc | 0.98495, 0.9737

```



문제 161. 위의 3층 신경망의 활성화함수가 Relu 일때와 sigmoid 일때의 차이를 테스트 하시오. (위는

```
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_1, hidden_size_2, output_size, weight_init_std=0.01):
    # 가중치 초기화
    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)

    # 계층 생성
    self.layers = OrderedDict()
    self.layers['Affine1'] = Affine(self.params['W1'], self.params['b1'])
    self.layers['sigmoid1'] = Sigmoid()
    self.layers['Affine2'] = Affine(self.params['W2'], self.params['b2'])
    self.layers['sigmoid2'] = Sigmoid()
    self.layers['Affine3'] = Affine(self.params['W3'], self.params['b3'])

    self.lastLayer = SoftmaxWithLoss()

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

    return x

# x : 입력 데이터, t : 정답 레이블
def loss(self, x, t):
    y = self.predict(x)
    return self.lastLayer.forward(y, t)

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

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

# x : 입력 데이터, t : 정답 레이블
def numerical_gradient(self, x, t):
    loss_W = lambda W: self.loss(x, t)

    grads = {}
    grads['W1'] = numerical_gradient(loss_W, self.params['W1'])
    grads['b1'] = numerical_gradient(loss_W, self.params['b1'])
    grads['W2'] = numerical_gradient(loss_W, self.params['W2'])
    grads['b2'] = numerical_gradient(loss_W, self.params['b2'])
    grads['W3'] = numerical_gradient(loss_W, self.params['W3'])
    grads['b3'] = numerical_gradient(loss_W, self.params['b3'])

    return grads

def gradient(self, x, t):
    # forward
    self.loss(x, t)

    # backward
    dout = 1
    dout = self.lastLayer.backward(dout)

```

```

layers = list(self.layers.values())
layers.reverse()
for layer in layers:
    dout = layer.backward(dout)

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

return grads

# 데이터 읽기
(x_train, t_train), (x_test, t_test) = load_mnist(normalize=True, one_hot_label=True)
network = TwoLayerNet(input_size=784, hidden_size_1=50, hidden_size_2=100, output_size=10)

# 하이퍼파라미터
iters_num = 10000 # 반복 횟수를 적절히 설정한다.
train_size = x_train.shape[0] # 60000 개
batch_size = 100 # 미니배치 크기
learning_rate = 0.1

train_loss_list = []
train_acc_list = []
test_acc_list = []

# 1에폭당 반복 수
iter_per_epoch = max(train_size / batch_size, 1)
print(iter_per_epoch) # 600

for i in range(iters_num): # 10000
    # 미니배치 획득 # 랜덤으로 100개씩 뽑아서 10000번을 수행하니까 백만번
    batch_mask = np.random.choice(train_size, batch_size) # 100개 씩 뽑아서 10000번 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', 'W3', 'b3'):
        network.params[key] -= learning_rate * grad[key]

# 학습 경과 기록
loss = network.loss(x_batch, t_batch)
train_loss_list.append(loss) # cost 가 점점 줄어드는것을 보려고

```

1에폭당 정확도 계산 # 여기는 훈련이 아니라 1에폭 되었을때 정확도만 체크

if i % iter_per_epoch == 0: # 600 번마다 정확도 쌓는다.

print(x_train.shape) # 60000,784

train_acc = network.accuracy(x_train, t_train)

test_acc = network.accuracy(x_test, t_test)

train_acc_list.append(train_acc) # 10000/600 개 16개 # 정확도가 점점 올라감

test_acc_list.append(test_acc) # 10000/600 개 16개 # 정확도가 점점 올라감

print("train acc, test acc | " + str(train_acc) + ", " + str(test_acc))

그래프 그리기

markers = {'train': 'o', 'test': 's'}

x = np.arange(len(train_acc_list))

plt.plot(x, train_acc_list, label='train acc')

plt.plot(x, test_acc_list, label='test acc', linestyle='--')

plt.xlabel("epochs")

plt.ylabel("accuracy")

plt.ylim(0, 1.0)

plt.legend(loc='lower right')

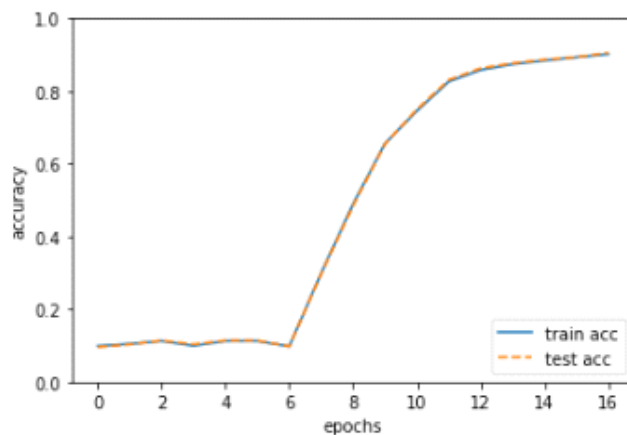
plt.show()

****결과**

```

600.0
(60000, 784)
train acc, test acc | 0.09863333333333334, 0.0958
(60000, 784)
train acc, test acc | 0.10441666666666667, 0.1028
(60000, 784)
train acc, test acc | 0.11236666666666667, 0.1135
(60000, 784)
train acc, test acc | 0.0993, 0.1032
(60000, 784)
train acc, test acc | 0.11236666666666667, 0.1135
(60000, 784)
train acc, test acc | 0.11236666666666667, 0.1135
(60000, 784)
train acc, test acc | 0.09753333333333333, 0.0974
(60000, 784)
train acc, test acc | 0.29946666666666666, 0.2974
(60000, 784)
train acc, test acc | 0.4896333333333333, 0.4849
(60000, 784)
train acc, test acc | 0.6558, 0.6546
(60000, 784)
train acc, test acc | 0.7458333333333333, 0.7496
(60000, 784)
train acc, test acc | 0.8263, 0.8308
(60000, 784)
train acc, test acc | 0.8583666666666666, 0.863
(60000, 784)
train acc, test acc | 0.87405, 0.8764
(60000, 784)
train acc, test acc | 0.8839666666666667, 0.8865
(60000, 784)
train acc, test acc | 0.89335, 0.8935
(60000, 784)
train acc, test acc | 0.9018666666666667, 0.9057

```



문제 162. 다시 Relu로 변환하고 노드수를 늘리면 정확도가 올라가는지 확인 하시오.
 기존 : 입력층 (784)----> 은닉1층(50) ----> 은닉2층 (100)----> 출력층(10)
 변환 : 입력층(784) ----> 은닉1층(100) ----> 은닉2층 (100)----> 출력층(10)

```

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_1, hidden_size_2, output_size, weight_init_std=0.01):
        # 가중치 초기화
        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)

        # 계층 생성
        self.layers = OrderedDict()
        self.layers['Affine1'] = Affine(self.params['W1'], self.params['b1'])
        self.layers['Relu1'] = Relu()
        self.layers['Affine2'] = Affine(self.params['W2'], self.params['b2'])
        self.layers['Relu2'] = Relu()
        self.layers['Affine3'] = Affine(self.params['W3'], self.params['b3'])

        self.lastLayer = SoftmaxWithLoss()

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

        return x

    # x : 입력 데이터, t : 정답 레이블
    def loss(self, x, t):
        y = self.predict(x)
        return self.lastLayer.forward(y, t)

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

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

    # x : 입력 데이터, t : 정답 레이블
    def numerical_gradient(self, x, t):
        loss_W = lambda W: self.loss(x, t)

        grads = {}
        grads['W1'] = numerical_gradient(loss_W, self.params['W1'])
        grads['b1'] = numerical_gradient(loss_W, self.params['b1'])
        grads['W2'] = numerical_gradient(loss_W, self.params['W2'])
        grads['b2'] = numerical_gradient(loss_W, self.params['b2'])
        grads['W3'] = numerical_gradient(loss_W, self.params['W3'])
        grads['b3'] = numerical_gradient(loss_W, self.params['b3'])

        return grads

    def gradient(self, x, t):
        # forward
        self.loss(x, t)

        # backward

```

```

dout = 1
dout = self.lastLayer.backward(dout)

layers = list(self.layers.values())
layers.reverse()
for layer in layers:
    dout = layer.backward(dout)

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

return grads

# 데이터 읽기
(x_train, t_train), (x_test, t_test) = load_mnist(normalize=True, one_hot_label=True)
network = TwoLayerNet(input_size=784, hidden_size_1=100, hidden_size_2=100, output_size=10)

# 하이퍼파라미터
iters_num = 10000 # 반복 횟수를 적절히 설정한다.
train_size = x_train.shape[0] # 60000 개
batch_size = 100 # 미니배치 크기
learning_rate = 0.1

train_loss_list = []
train_acc_list = []
test_acc_list = []

# 1에폭당 반복 수
iter_per_epoch = max(train_size / batch_size, 1)
print(iter_per_epoch) # 600

for i in range(iters_num): # 10000
    # 미니배치 획득 # 랜덤으로 100개씩 뽑아서 10000번을 수행하니까 백만번
    batch_mask = np.random.choice(train_size, batch_size) # 100개 씩 뽑아서 10000번 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', 'W3', 'b3'):
        network.params[key] -= learning_rate * grad[key]

# 학습 경과 기록
loss = network.loss(x_batch, t_batch)
train_loss_list.append(loss) # cost 가 점점 줄어드는것을 보려고

```

1에폭당 정확도 계산 # 여기는 훈련이 아니라 1에폭 되었을때 정확도만 체크

```
if i % iter_per_epoch == 0: # 600 번마다 정확도 쌓는다.  
    print(x_train.shape) # 60000,784  
    train_acc = network.accuracy(x_train, t_train)  
    test_acc = network.accuracy(x_test, t_test)  
    train_acc_list.append(train_acc) # 10000/600 개 16개 # 정확도가 점점 올라감  
    test_acc_list.append(test_acc) # 10000/600 개 16개 # 정확도가 점점 올라감  
    print("train acc, test acc | " + str(train_acc) + ", " + str(test_acc))
```

그래프 그리기

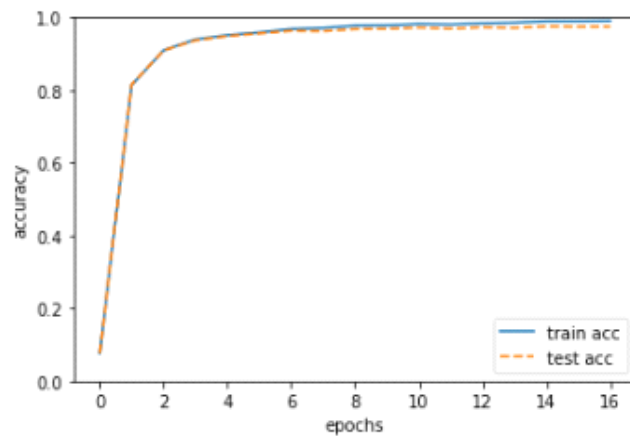
```
markers = {'train': 'o', 'test': 's'}  
x = np.arange(len(train_acc_list))  
plt.plot(x, train_acc_list, label='train acc')  
plt.plot(x, test_acc_list, label='test acc', linestyle='--')  
plt.xlabel("epochs")  
plt.ylabel("accuracy")  
plt.ylim(0, 1.0)  
plt.legend(loc='lower right')  
plt.show()
```

****결과**


```

600.0
(60000, 784)
train acc, test acc | 0.0768, 0.0766
(60000, 784)
train acc, test acc | 0.8140666666666667, 0.8156
(60000, 784)
train acc, test acc | 0.9094, 0.9091
(60000, 784)
train acc, test acc | 0.9397, 0.9369
(60000, 784)
train acc, test acc | 0.9514666666666667, 0.9488
(60000, 784)
train acc, test acc | 0.9586833333333333, 0.9561
(60000, 784)
train acc, test acc | 0.9682166666666666, 0.9632
(60000, 784)
train acc, test acc | 0.9709166666666667, 0.9622
(60000, 784)
train acc, test acc | 0.9774833333333334, 0.9679
(60000, 784)
train acc, test acc | 0.9782166666666666, 0.9686
(60000, 784)
train acc, test acc | 0.9819833333333333, 0.972
(60000, 784)
train acc, test acc | 0.9806166666666667, 0.9691
(60000, 784)
train acc, test acc | 0.9839833333333333, 0.9727
(60000, 784)
train acc, test acc | 0.9856166666666667, 0.9712
(60000, 784)
train acc, test acc | 0.9896166666666667, 0.9756
(60000, 784)
train acc, test acc | 0.9896833333333334, 0.9747
(60000, 784)
train acc, test acc | 0.9904166666666666, 0.9748

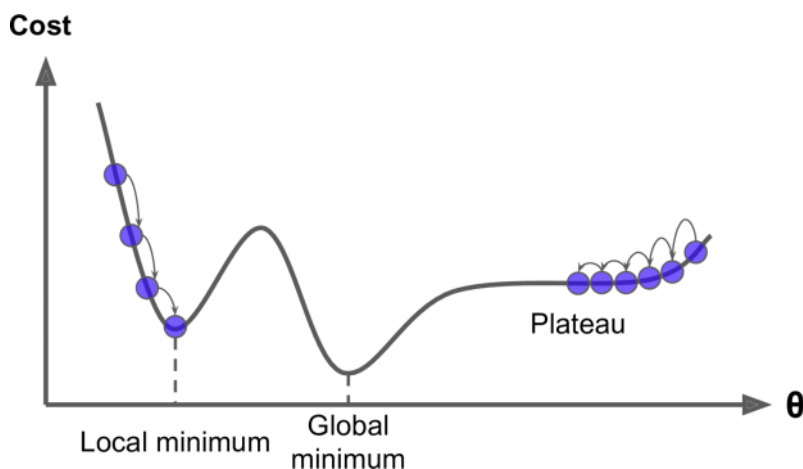
```



2. 경사하강법 Gradient Descent

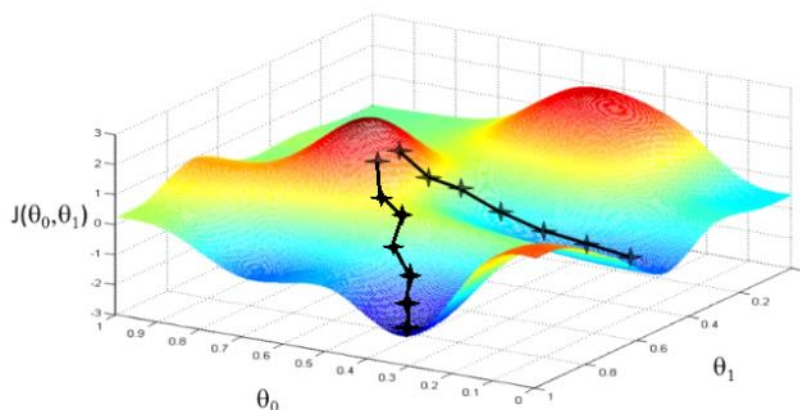
경사하강법이란, 최적의 가중치와 편향을 찾기 위해서 학습을 진행하며 손실함수가 최솟값을 갖도록 하는 것이다. 가중치의 차원이 커질수록 이 손실함수의 최솟값을 갖는 지점을 찾는 것은 어려워진다. 기울기를 이용해서 함수의 최솟값이 어디에 있는지 찾는 것이며 이를 경사하강법이라고 한다.

기울기를 통해서 손실함수의 최솟값이 그쪽에 있는지 보는 것이며 최소한의 방향을 나타낸다. 최솟값이 되는 지점에서는 기울기가 0이 된다. 주의할 것은 기울기가 0이라고 반드시 그 지점이 최솟값이라고 할 수 없다. 경사하강법에서 발생할 수 있는 문제이기도 하다.



위의 그림을 보면 plateau라는 지점에서 기울기가 0이 되어 최솟값이 아님에도 최솟값으로 인식하고 학습이 중지된 상태다.

이러한 문제를 피하기 위해서는 서로 다른 초기값으로 주어 산을 내려가게 하는 방법으로 신경망의 경우에는 weight의 초기 값을 다르게 주면서 경사하강법을 활용해본다는 의미이다.



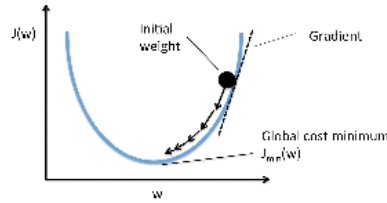
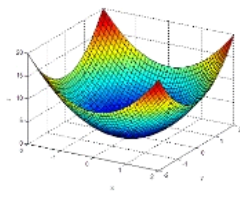
아래 그림과 같이 수학적으로 설명하면 낮은 쪽의 방향을 찾기 위해 오차함수를 현재 위치에서 미분한다.

한 발 내려간 시점에서 등산객이 자기가 산을 제대로 내려가고 있는지 주변 경사(기울기)를 보면서 판단하듯이 기울기가 음수인지? 양수인지? 확인하면서 최저지점으로 내려가는 거라고 보면 되지 않을까 싶다.

경사하강법(Gradient Descent)

- 오차함수의 낮은 지점을 찾아가는 최적화 방법
- 낮은 쪽의 방향을 찾기 위해 오차함수를 현재 위치에서 미분함

$$J = \frac{1}{2m} \sum_{i=1}^m (y - \hat{y})^2, \quad \nabla J = \frac{1}{m} (y - \hat{y})$$



18

경사하강법의 특징

- ♦ 함수의 최저점을 구하기 좋은 방법으로 신경망과 같이 계산해야 하는 양이 많고(선형대수학) 접근하기가 복잡한 수식에서 잘 작동
- ♦ 데이터가 불완전해도 유도리 있게 작동

Learning Rate

아래는 학습률 Learning Rate라고 하며 한 번 학습할 때 얼마만큼 학습해야 하는지 학습 양을 의미하며 한 번의 학습량으로 학습한 이후에 가중치 매개변수가 갱신된다.

$$\alpha(t) = \alpha(0)(1.0 - t/rlen)$$

학습률 값은 미리 0.01, 0.001과 같이 특정 값을 정해두어야 하며 일반적으로 이 값이 너무 크거나 작으면 적합한 지점으로 찾아가기가 어렵다. 신경망 학습에서는 보통 이 학습률 값을 변경하면서 올바르게 학습하고 있는지를 확인한다. 학습률이 너무 크면 큰 값을 반환하고, 너무 작으면 거의 갱신되지 않고 학습이 끝나버린다.

학습률은 하이퍼파라미터 hyperparameter라고 부르며 가중치와 편향 같은 신경망의 매개변수와는 다르다. 가중치 등은 훈련용 데이터 셋과 학습 알고리즘에 따라 자동적으로 생성되는 것임에 비해서 학습률 같은 하이퍼파라미터는 사람이 수동적으로 설정해야 하기 때문에 시험을 통해서 가장 적절한 값을 찾아가야 한다.

6.1 가중치(W) 매개변수 갱신

2018년 8월 26일 일요일 오후 1:56

목차

1. 고급 경사 하강법 : underfitting을 해결하기 위한 방법
 - SGD
 - Momentum
 - AdaGrade
 - Adam
2. 가중치 초기화값 설정 : underfitting을 해결하기 위한 방법
3. 배치 정규화 : underfitting을 해결하기 위한 방법
4. Dropout 기법 : overfitting을 해결하기 위한 방법

딥러닝은 깊은 신경망 계층을 말하며 수 많은 모듈의 조합으로 이루어진 것이라고 볼 수 있다. 따라서 모듈과 함수가 많은 만큼 각각에 대한 하이퍼파라미터(hyper-parameter)가 많다. 하이퍼파라미터를 어떻게 설정하느냐에 따라서 학습의 결과가 천차만별로 달라질 수 있다.

신경망 모델의 학습과 그 결과에 따른 손실함수의 값을 최소화하는 방향으로 하이퍼파라미터의 값을 찾는 것이 최적화의 목표라고 할 수 있다. 하이퍼파라미터 매개변수의 수 n 만큼 가능한 매개변수의 조합이 $n \times n$ 만큼 증가하므로 이중에서 최적의 조합을 찾는 것은 쉽지 않다. 다양한 방법으로 자동으로 찾고자 하지만 여전히 이는 연구자의 주관이나 직관, 경험 등에 의존하고 있다.

딥러닝 모델 학습에서 많이들 중요성을 간과하는 부분이 이 최적화 부분이다. 신경망을 생성하면서 이 최적화 문제에 대해서 끊임 없이 고민해야 한다. 이는 Input으로 넣는 데이터와의 직접적인 연결성도 있으며 하이퍼파라미터에 따른 모델의 변화를 어느정도 이해하고 예측하여 튜닝 과정을 할 수 있어야 하기 때문이다. 눈가리고 모래에서 진주찾는 식으로 적절한 하이퍼파라미터 값을 찾는 것은 좋지 못한 방법이다.

6.0 경사 하강법 (Gradient Descent)

카카오 면접질문

GD (Gradient Descent)는 가장 기본적인 뉴럴넷의 학습 방법으로 전체 데이터에 대한 에리(비용) 함수를 Weight과 미분하여 각 W(가중치) 파라미터에 대한 기울기를 이용하여 W(가중치)를 업데이트 하는 방법이다.

그런데 이 GD는 층이 많아질수록 parameter들의 복잡도가 늘어남에 따라 에리(비용) 함수가 더욱 복잡해져서 local minima에 빠지는 현상이 더욱 잘 발생하게 된다.

그래서 이문제를 해결하기 위해 나온것이 SGD (**Stochastic:확률적 Gradient Descent**)

**경사하강법의 단점

1. 계산량이 많아서 속도가 느리다.
2. local minima에 빠질 수 있다.

6.1 확률적 경사 하강법 (SGD : Stochastic Gradient Descent)

데이터를 미니배치로 무작위로 선정하여 수행하는 경사 하강법.

최적의 가중치 값을 구하기 위해서 앞에서는 미분을 통해 기울기를 구하여 가중치 값을 갱신하는 방법인 확률적 경사하강법(Stochastic Gradient Descent; SGD) 방법을 사용하였다. 이는 무작정 가중치 값을 랜덤으로 콕콕 찍어서 찾는 방식보다는 훨씬 스마트한 방법이다.

SGD는 손실함수의 기울기를 계산하여서 이 기울기 값에 학습률(Learning Rate)를 계산하여 이 결과 값으로 기존의 가중치 값을 갱신한다.

전체 data를 가지고 계산하면 계산량이 너무 커서 속도가 느려지므로 전체 data 중 랜덤하게 추출한 1/N의 데이터만을 사용해서 훨씬 빠르게 가중치를 업데이트 하는 방법이다. (복원추출) --> GD 보단 빠름

```
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]
```

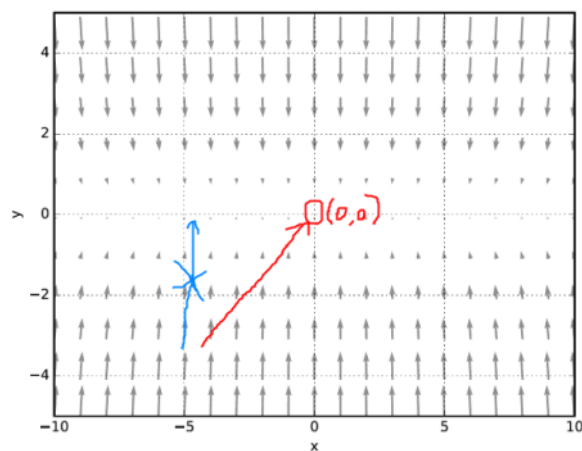
****SGD는 구현하기 쉽고 GD보단 빠르다는 장점을 가짐**

(매우 큰 데이터 셋을 효율적으로 처리하는 장점 = SGD가 한 번에 하나씩 훈련 샘플을 독립적으로 처리 --> 온라인 학습에 잘 어울린다.)

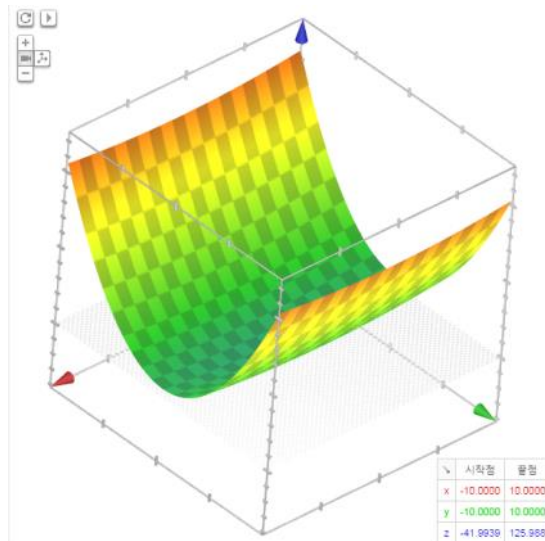
■ 확률적 경사 하강법(Stochastic Gradient Descent)의 단점

경사하강법은 무작정 기울어진 방향으로 이동하는 방식이기 때문에 탐색경로가 비효율적이어서 한참을 탐색하게 된다.

또한 SGD는 방향에 따라서 기울기 값이 달라지는 경우에 적합하지 않는데 최소값의 방향으로만 움직이기 때문에 본래의 최저점으로 기울기의 방향이 좀처럼 향하지 않게 되고 다른 방향을 가리키게 되어 비효율적이게 되기도 한다. (local minima에 빠질 수 있다.)



global minima 쪽으로 기울기 방향이 있는게 아니기 때문에 local minima에 빠지는 단점이 있다.



6.2 모멘텀 (Momentum)

모멘텀은 '운동량'을 의미한다. 기울기에서 속도의 개념이 추가된 것으로 고등학교 물리 시간을 떠올려보면 자세히는 아니지만 지상의 마찰력 때문에 물체의 이동속도가 점점 감소한다고 배웠던 기억이 어렴풋이 기억이 난다. 속도가 크게 나을수록 기울기가 크게 업데이트 되어 확률적 경사하강법이 가지는 단점을 보완할 수 있다.

| 기존의 SGD | Momentum |
|-------------------------|---|
| 가중치 = 가중치 - 러닝레이트 * 기울기 | 가중치 = 가중치 + 속도 [속도 : 0.9(마찰계수) * 속도 - 러닝레이트 * 기울기] 내리막 : 기울기 음수이면 속도 증가 오르막 : 기울기 양수이면 속도 감소 |

Momentum은 마찰력/공기저항 같은 것에 해당하며 기울기 업데이트 시 이 폭을 조절하는 역할을 한다. 이에 따라 속도 velocity가 변화한다.

```
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]
```

6.3 AdaGrad

신경망 학습에서의 학습률 learning rate의 값은 일종의 보폭으로 생각할 수 있는데 한 번 갱신하는 가중치의 값을 양을 결정한다. 학습률을 너무 작게하면 보폭이 너무 작아서 많은 걸음을 걸어야 하므로 학습 시간을 아주 길게 해야 한다. 반대로 너무 크게 하면 최적의 점을 계속 지나치게 된다.

학습률을 정하는 방법으로 **학습률 감소(learning rate decay)**가 있다고 한다. 학습을 진행하면서 점차 학습률을 줄여나가는 방법이다. 하지만 학습이 계속되면서 학습률이 0에 가까워져서 학습이 진행이 안되는 문제가 발생한다.

- 학습률 높으면 : 발산될 위험이 있지만 수렴속도가 빨라진다.
- 학습률 낮으면 : 발산될 위험은 없지만 local minima에 빠지거나 학습이 안될 수 있다.

| SGD | Adagrade |
|--|--|
| $W = W - \eta * \partial L / \partial W$ | $H = H + (\partial L / \partial W) \odot (\partial L / \partial W)$ 처음에는 학습률이 크다가 조금씩 감소 \odot 는 행렬의 원소별 곱셈을 의미하는데, h의 원소가 각각의 매개변수 원소 변화량에 의해 결정된다. --> 각각의 매개변수 원소가 갱신되는 값이 다르다. --> 많이 움직인 원소일 수록 누적 h의 값이 크니까 갱신정도가 그만큼 감소한다. |
| | $\text{SGD} : W \leftarrow W - \eta \frac{\partial L}{\partial W}$ $\text{Adagrad} : W \leftarrow W - \eta \frac{1}{\sqrt{h}} \frac{\partial L}{\partial W}$ $h \leftarrow h + \frac{\partial L}{\partial W} \odot \frac{\partial L}{\partial W}$ <p>(\odot : 행렬의 원소별 곱셈을 의미)</p> |

****가중치마다 학습률이 다르다 !!**

AdaGrad는 과거의 기울기 값을 제공해서 계속 더하는 식으로 학습률을 낮추는데 학습이 진행될수록 제공의 값으로 학습의 정도가 크게 떨어진다.

이를 개선하기 위해서 **RMSProp**이 사용된다. RMSProp은 과거의 모든 기울기를 균일하게 더하지 않고 새로운 기울기의 정보만 반영하도록 해서 학습률이 크게 떨어져 0에 가까워지는 것을 방지하는 방법이다.

```
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)
```

6.4 Adam

- Momentum의 장점 + Adagrade의 장점을 살린 경사 감소법

* 최근 딥러닝에서 가장 많이 사용하는 최적화 방법

<http://shuuki4.github.io/deep%20learning/2016/05/20/Gradient-Descent-Algorithm-Overview.html>

문제 163. SGD 클래스를 만들고 SGD 클래스를 이용해서 3층 신경망의 코드를 변경하시오. (P.191)

```
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)
```

****결과**

```
0.08951666666666666 0.0858
0.7032666666666667 0.7084
0.8729166666666667 0.8745
0.9048 0.9043
0.9250666666666667 0.923
0.9362833333333334 0.9335
0.9449333333333333 0.9452
0.9483666666666667 0.9449
0.9564166666666667 0.9542
0.9604333333333334 0.9558
0.96375 0.9586
0.96675 0.9599
0.96665 0.9608
0.97145 0.9634
0.9694833333333334 0.9599
0.9744666666666667 0.9639
0.9762666666666666 0.9659
```

문제 164. Momentum 클래스를 파이썬 코드로 구현하고 방금 만든 3층 신경망에 적용해서 정확도를 확인 하시오.

```
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]

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]
```

```
optimizer = Momentum() # 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)
```

****결과**

```
0.11246666666666667 0.1134
```

```
0.94675 0.9444
```

```
0.9625333333333334 0.9575
```

```
0.9716666666666667 0.9618
```

```
0.9754333333333334 0.967
```

```
0.9759 0.9654
```

```
0.9810833333333333 0.9715
```

```
0.9835666666666667 0.9686
```

```
0.9836 0.9712
```

```
0.9868666666666667 0.9711
```

```
0.9890666666666666 0.9741
```

```
0.9887 0.9714
```

```
0.9946166666666667 0.9784
```

```
0.9915166666666667 0.9747
```

```
0.9945333333333334 0.9768
```

```
0.9952833333333333 0.976
```

```
0.9966666666666667 0.9774
```

문제 165. Adagrade 경사 감소법으로 위의 3층 신경망을 학습 시킨 결과 정확도를 출력 하시오.

```
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]

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]

```

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() # 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)

```

****결과**

```

0.23548333333333332 0.2361

```

```

0.9315666666666667 0.9293

```

```

0.9457833333333333 0.9417

```

```

0.9549 0.9478

```

```

0.95955 0.9533

```

```

0.9610666666666666 0.9544

```

```

0.9651 0.9599

```

```

0.9668 0.9598

```

```

0.9688166666666667 0.96

```

```

0.97145 0.9629

```

```

0.9726333333333333 0.9635

```

```

0.9744833333333334 0.964

```

```

0.97575 0.9638

```

```

0.97535 0.9654

```



```
0.97665 0.9654
0.9775 0.9657
0.9789833333333333 0.9661
```

문제 166. adam 경사 감소법으로 위의 3층 신경망을 학습 시킨 결과 정확도를 출력 하시오.

```
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]

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]

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)

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() # 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)

```

****결과**

```

0.3276333333333333 0.3223
0.9533166666666667 0.9524
0.9661 0.9601
0.9735833333333334 0.9647
0.9816666666666667 0.9694
0.9826166666666667 0.9704
0.9841166666666666 0.9728
0.9876 0.9707
0.9845 0.9692
0.98645 0.9679
0.9887333333333334 0.9695
0.98865 0.9705
0.9901166666666666 0.968
0.9914666666666667 0.9724
0.9917166666666667 0.9722
0.9940333333333333 0.9743
0.9897333333333334 0.9714

```

6.2 가중치의 초깃값

2018년 8월 26일 일요일 오후 2:25

1. 가중치 초깃값을 0으로 하면?

가중치의 초깃값을 0으로 default를 주고 시작하면 올바른 학습을 기대하기 어렵다. 오차역전파에서 가중치의 값이 똑같이 갱신되기 때문이다. 가중치가 각각 영향이 있어야 하는데 고르게 되어버리는 상황이 발생하면 각각의 노드를 만든 의미를 잃어버리게 된다. 그래서 다른 샘플 코드나 예제를 보면 `np.random.randn()`을 사용하여 **무작위로 가중치를 설정**하는 것을 볼 수 있다.

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

1. 가중치 초깃값을 0으로 설정 ---> 학습 안됨

2. 표준편차가 1인 정규분포를 사용해 초깃값 설정 --->

- 표준편차가 작을수록 data가 평균에 가깝게 분포 (시험문제가 쉬우면 학생들의 점수가 평균에 가깝다)
- 구현 예 : `0.01 * np.random.randn(10,100)`
- 표준편차가 클수록 data가 더 많이 흩어져 있다. (시험문제가 어려우면 아주 잘하는 학생들과 아주 못하는 학생들로 점수가 나뉘진다.)

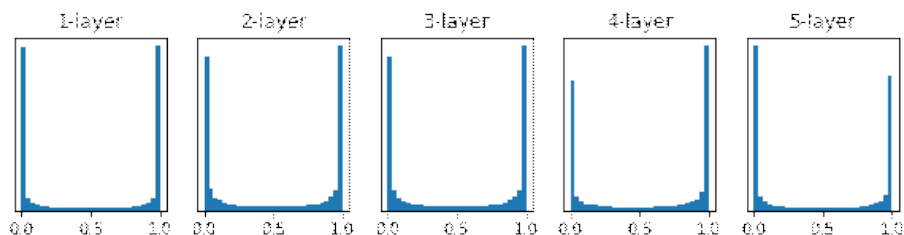
3. Xavier(사비에르) 초깃값 선정

- 표준편차가 $(1/n)^{1/2}$ 인 정규 분포로 초기화를 한다. (n은 앞층의 노드수)
- sigmoid와 짝꿍

2. Sigmoid 가중치 초깃값 설정 : Xavier

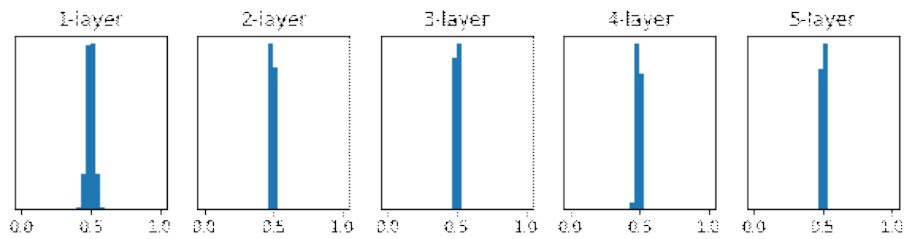
표준편차가 1

활성화함수가 시그모이드 함수인 경우 활성화 값이 0과 1에 주로 분포되어 있다. 이는 시그모이드 함수의 미분 값이 0에 가까워지도록 한다. 따라서 역전파의 기울기 값은 점점 작아지다가 사라지는 Gradient Vanishing 문제가 발생하며 층이 깊어질수록 기울기는 더 사라진다.



표준편차가 0.01

기울기가 사라지는 문제는 발생하지 않았으나 0.5 주변으로 값이 여전히 치우쳐있기 때문에 뉴런을 다수 사용한 이점이 없다. 뉴런 100개가 똑같은 값을 출력한다면 100개가 다 필요한 이유가 없다.

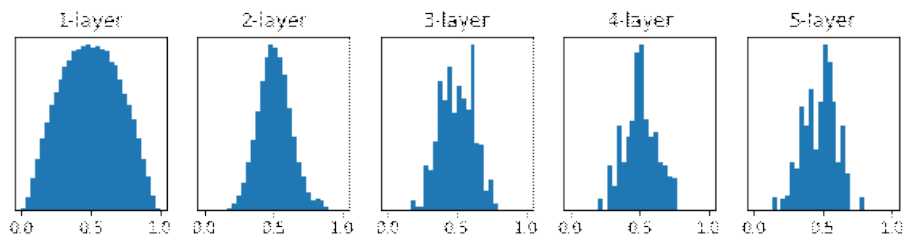


Xavier 초기값

Xavier Glorot & Yoshua Bengio의 논문에서 권장하는 가중치 초기값.

앞 층의 입력 노드 수에 더하여 다음 계층의 출력 노드 수를 함께 고려하여 초기값을 설정하는 방법

아래 그래프를 보면 앞의 두 방식보다 고르게 분포되어 학습이 효율적으로 이루어질 수 있음을 엿볼 수 있다. 시그모이드 함수를 사용했음에도 표현력의 문제가 발생하지 않는다.



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

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

x = np.random.randn(1000, 100)
node_num= 100
hidden_layer_size = 5
activations = {}

for i in range(hidden_layer_size):
    if i != 0:
        x = activations[i-1]

        w = np.random.randn(node_num, node_num) / np.sqrt(node_num)

        a = np.dot(x, w)
        z = sigmoid(a)
        activations[i] = z

fig = plt.figure(figsize=(10,2))
for i, a in activations.items():
    plt.subplot(1, len(activations), i+1)
    plt.title(str(i+1) + "-layer")
    plt.yticks([],[])
    plt.hist(a.flatten(), 30, range=(0,1))
plt.show()
```

3. ReLU 가중치 초기값 설정 : He

He 초기값은 ReLU에 특화된 초기값이다.

계층의 노드가 n 개 일 때 표준편차 $\sqrt{2/n}$ 인 정규분포를 사용

ReLU는 음의 영역은 0을 출력하는 활성화함수로 정규분포 표준편차를 사용하거나 Xavier를 사용할 경우 Hidden Layer의 계층이 깊어질수록 가중치의 값이 0으로 수렴하여 가중치가 사라지는 Gradient Vanishing 문제가 발생한다.

문제 167. 지금까지 구현한 3층 신경망의 가중치 초기값을 0으로 설정하고 신경망을 학습 시켜 보시오.

```
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 = 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]
```

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]
```

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)
```

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() # SGD 클래스를 객체화 시킨다.

```

for i in range(its_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)

```

****결과**

```

0.0993 0.1032
0.11236666666666667 0.1135
0.11236666666666667 0.1135
0.11236666666666667 0.1135
0.11236666666666667 0.1135
## 학습이 되지 않는다.

```

문제 168. 가중치 초기값을 xavier로 설정하고 학습이 잘되는지 확인 하시오.

```

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):
        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)

        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]

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]

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)
```

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() # 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)
```

****결과**

```
0.4083833333333333 0.4183
0.9619 0.9583
0.9661833333333333 0.9567
0.97615 0.9638
0.9824666666666667 0.9704
0.9788333333333333 0.9632
0.9849666666666667 0.9704
0.9830166666666666 0.9675
0.98625 0.9701
0.9860833333333333 0.9707
0.9854166666666667 0.9697
0.9848333333333333 0.9644
0.9914166666666666 0.9729
# 잘된다
```

문제 169. 지금까지 구현한 3층 신경망의 사중치 초기값을 He로 설정하고 정확도를 확인하시오.

```
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):
        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)

        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]

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]

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)

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() # 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)

```

****결과**

```

0.4083833333333333 0.4183
0.9619 0.9583
0.9661833333333333 0.9567
0.97615 0.9638
0.9824666666666667 0.9704
0.9788333333333333 0.9632
0.9849666666666667 0.9704
0.9830166666666666 0.9675
0.98625 0.9701
0.9860833333333333 0.9707
0.9854166666666667 0.9697
0.9848333333333333 0.9644
0.9914166666666666 0.9729
# 잘된다

```

6.3 배치 정규화

2018년 8월 26일 일요일 오후 2:50

신경망의 정확도를 높이기 위한 여러가지 방법 소개

1. 고급 경사하강법 -> underfitting 을 해결하기 위한 방법
2. 가중치 초기화값 설정 -> underfitting 을 해결하기 위한 방법
3. 배치 정규화 -> underfitting 을 해결하기 위한 방법

앞에서는 가중치 초기값을 적절히 설정하면 각 층의 활성화 값의 분포가 적당히 퍼지는 효과를 보았다.
적당히 퍼지게 되면 학습이 잘되고 정확도가 높아지는 것을 확인했다.

그런데 배치 정규화는 바로 각 층에서의 활성화 값이 적당히 분포되도록 강제로 조정하는 것을 말한다.

※ 배치 정규화의 필요성

뉴럴 네트워크는 파라미터가 많기 때문에 학습이 어렵다.

딥러닝의 경우는 hidden layer가 많아서 학습이 어렵다.

※ Why 어렵?

가중치의 조금한 변화가 가중되어서 쌓이면

히든 레이어가 많아질수록 출력되는 값의 변화가 크기 때문이다.

그림

만약에 변화없이 hidden_layer2 까지 안정된 가중치 값을 가지게 된다면 학습이 잘될 것이다.

그런데 가중치의 영향을 계속 받아서 hidden_layer2의 값이 아주 다른값이 된다면
기존값과는 다르기 때문에 어떻게 학습해야할지 모르게되는 상황이 발생한다.

그래서 나온 것이 배치정규화이다.

배치 정규화는 값이 **활성화 함수를 통과하기 전에 가중의 변화를 줄이는 것이 목표**이다.

가중의 합이 배치 정규화에 들어오게 되면 기존 값의 스케일을 줄여버리게 된다.

| | |
|---------|--|
| 문제 170. | 배치 정규화 클래스를 우리 3층 신경망에 구현하고 각층마다 아래와 같이 배치 정규화층을 구현 하시오. |
|---------|--|

```
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 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
    dmux = np.sum(dxc, axis=0)
    dx = dxc - dmux / 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):
    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.lastLayer = SoftmaxWithLoss()
    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 = 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]

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]

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)

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() # 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)

**결과
0.32981666666666665 0.3258
0.9684833333333334 0.9624
0.9753166666666667 0.9664
0.9830833333333333 0.9716
0.9845333333333334 0.9698
0.9845833333333334 0.9724
0.9889333333333333 0.9718
0.9906333333333334 0.9749
0.9909166666666667 0.9751
0.9922333333333333 0.9728
0.9925 0.9754
0.9939333333333333 0.9781
0.99335 0.9752

```

6.4 오버피팅 문제

2018년 9월 4일 화요일 오전 10:21

■ 오버 피팅을 억제하는 방법 2가지

1. 드롭아웃 (drop out)
2. 가중치 감소 (Weight Decay)

■ 드롭아웃 (Dropout) p219

" 오버피팅을 억제하기 위해서 뉴런을 임의로 삭제하면서 학습시키는 방법 "

고양이와 개 사진을 구분하는 신경망

고양이 사진 ===== 은닉 1층 ===== 고양이 점수

0 -----> 귀가 있어요 -----> 가중치 출력

0 -----> 꼬리가 있어요 -----> **아주 큰 가중치 출력** (사진에 꼬리가 안찍힌 경우 고양이로 분류못함)

0 -----> 고양이처럼 생긴 사람 같아요 -----> 가중치 출력

0 -----> 집게발을 가지고 있어요 -----> 가중치 출력

0 -----> 장난스럽게 보여요 -----> 가중치 출력

전문가가 많으면 오히려 오답이 나올 수 있다.

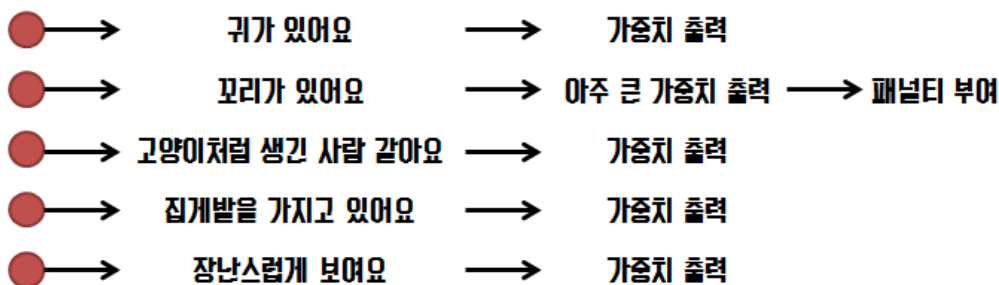
사공이 많으면 배가 산으로 가기 때문에 몇명의 전문가만 선별해서 반복적으로 같은 결과가 나오면 그것을 답으로 보겠다.

일반적으로 입력층에서 20~50% 정도, 은닉층에서는 50% 정도의 노드를 생략한다.

■ 가중치 Decay

학습 과정에서 큰 가중치에 대해서는 그에 상응하는 큰 패널티를 부여해서 오버피팅을 억제하는 방법

ex.



고양이 사진 ===== 은닉 1층 ===== 고양이 점수

0 -----> 귀가 있어요 -----> 가중치 출력

0 -----> 꼬리가 있어요 -----> **아주 큰 가중치 출력** (사진에 꼬리가 안찍힌 경우 고양이로 분류못함)

0 -----> 고양이처럼 생긴 사람 같아요 -----> 가중치 출력

0 -----> 집게발을 가지고 있어요 -----> 가중치 출력

0 ----> 장난스럽게 보여요 -----> 가중치 출력

■ 가중치 Decay 코드 분석

1. 모든 가중치의 각각의 손실 함수에 $1/2 * \lambda * w^2$ 를 더한다.

****오차 함수의 코드에서 수정해야할 부분**

```
weight_decay += 0.5 * self.weight_decay_lambda * np.sum(w**2)
return self.lastLayer.forward(y,t) + weight_decay
```

2. Gradient Descent 업데이트 과정에서 그동안 오차 역전파법에 따른 결과에 정규화 항을 미분한 값에 람다 * 가중치를 더한다.

****코드예제 :**

기존 코드 : $\text{grad}['W1'] = \text{미분값}$

변경된 코드 : $\text{grad}['W1'] = \text{미분값} + \lambda * \text{현재의 가중치}$

예제(1) : Dropout class 구현 코드

```
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

** 100개의 노드가 있을 때 dropout_ratio를 0.15로 설정했으면 몇개의 노드가 남을까?
    1. 85개
    2. 15개
책의 코드에서는 훈련할 때는 1번이고 테스트 할때는 2번이다.
```

문제 171. 지금까지 만들었던 3층 신경망 코드에서 dropout을 적용해서 오버피팅이 발생하지 않는지 확인하시오.

```
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 $\leftarrow 0.9(\text{마찰계수}) * \text{속도} - \text{러닝레이트} * \text{기울기}$

가중치 $\leftarrow \text{가중치} + \text{속도}$

optimizer = Momentum() # Momentum 클래스 객체화

Adagrad 클래스

class AdaGrad:

```
def __init__(self, lr=0.01):
    self.lr = lr
    self.h = None
```

```
def update(self, params, grads):
    if self.h is None:
        self.h = {}
    for key, val in params.items():
        self.h[key] = np.zeros_like(val)
```

```
for key in params.keys():
    self.h[key] += grads[key] * grads[key]
    params[key] -= self.lr * grads[key] / (np.sqrt(self.h[key]) + 1e-7)
```

optimizer = AdaGrad() # AdaGrad 클래스 객체화

Adam 클래스

class Adam:

```
def __init__(self, lr=0.001, beta1=0.9, beta2=0.999):
    self.lr = lr
    self.beta1 = beta1
    self.beta2 = beta2
    self.iter = 0
```

```

self.m = None
self.v = None

def update(self, params, grads):
    if self.m is None:
        self.m, self.v = {}, {}
        for key, val in params.items():
            self.m[key] = np.zeros_like(val)
            self.v[key] = np.zeros_like(val)

    self.iter += 1
    lr_t = self.lr * np.sqrt(1.0 - self.beta2 ** self.iter) / (1.0 - self.beta1 ** self.iter)

    for key in params.keys():

        self.m[key] += (1 - self.beta1) * (grads[key] - self.m[key])
        self.v[key] += (1 - self.beta2) * (grads[key] ** 2 - self.v[key])

        params[key] -= lr_t * self.m[key] / (np.sqrt(self.v[key]) + 1e-7)

optimizer = Adam() # Adam 클래스 객체화

for i in range(iters_num):
    batch_mask = np.random.choice(train_size, batch_size)
    x_batch = x_train[batch_mask]
    t_batch = t_train[batch_mask]

    for key in ('W1', 'b1', 'W2', 'b2', 'W3', 'b3'):
        grads = network.gradient(x_batch, t_batch)
        params = network.params
        optimizer.update(params, grads)

    loss = network.loss(x_batch, t_batch)
    train_loss_list.append(loss)

    if i % iter_per_epoch == 0:
        train_acc = network.accuracy(x_train, t_train)
        test_acc = network.accuracy(x_test, t_test)
        train_acc_list.append(train_acc)
        test_acc_list.append(test_acc)
        print(train_acc, test_acc)

```

****결과**

```

0.32981666666666665 0.3258
0.9684833333333334 0.9624
0.9753166666666667 0.9664
0.9830833333333333 0.9716
0.9845333333333334 0.9698
0.9845833333333334 0.9724
0.9889333333333333 0.9718
0.9906333333333334 0.9749
0.9909166666666667 0.9751
0.9922333333333333 0.9728
0.9925 0.9754
0.9939333333333333 0.9781
0.99335 0.9752

```

문제 172. 오버피팅을 억제하는 가중치 감소방법 중에 하나인 L2 정규화를 이용해서 3층 신경망을 구현 하시오.

```
import sys, os
```

```

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

##### 배치정규화 클래스 #####

class BatchNormalization:
    """
    http://arxiv.org/abs/1502.03167
    """

    def __init__(self, gamma, beta, momentum=0.9, running_mean=None, running_var=None):
        self.gamma = gamma
        self.beta = beta
        self.momentum = momentum
        self.input_shape = None # 합성곱 계층은 4차원, 완전연결 계층은 2차원

        # 시험할 때 사용할 평균과 분산
        self.running_mean = running_mean
        self.running_var = running_var

        # backward 시에 사용할 중간 데이터
        self.batch_size = None
        self.xc = None
        self.std = None
        self.dgamma = None
        self.dbeta = None

    def forward(self, x, train_flg=True):
        self.input_shape = x.shape
        if x.ndim != 2:
            N, C, H, W = x.shape
            x = x.reshape(N, -1)

        out = self.__forward(x, train_flg)

        return out.reshape(*self.input_shape)

    def __forward(self, x, train_flg):
        if self.running_mean is None:
            N, D = x.shape
            self.running_mean = np.zeros(D)
            self.running_var = np.zeros(D)

        if train_flg:
            mu = x.mean(axis=0)
            xc = x - mu
            var = np.mean(xc ** 2, axis=0)
            std = np.sqrt(var + 10e-7)
            xn = xc / std

            self.batch_size = x.shape[0]
            self.xc = xc
            self.xn = xn
            self.std = std
            self.running_mean = self.momentum * self.running_mean + (1 - self.momentum) * mu
            self.running_var = self.momentum * self.running_var + (1 - self.momentum) * var
        else:
            xc = x - self.running_mean
            xn = xc / ((np.sqrt(self.running_var + 10e-7)))

```

```

    out = self.gamma * xn + self.beta
    return out

def backward(self, dout):
    if dout.ndim != 2:
        N, C, H, W = dout.shape
        dout = dout.reshape(N, -1)

    dx = self.__backward(dout)

    dx = dx.reshape(*self.input_shape)
    return dx

def __backward(self, dout):
    dbeta = dout.sum(axis=0)
    dgamma = np.sum(self.xn * dout, axis=0)
    dxn = self.gamma * dout
    dxc = dxn / self.std
    dstd = -np.sum((dxn * self.xc) / (self.std * self.std), axis=0)
    dvar = 0.5 * dstd / self.std
    dxc += (2.0 / self.batch_size) * self.xc * dvar
    dmua = np.sum(dxc, axis=0)
    dx = dxc - dmua / self.batch_size

    self.dgamma = dgamma
    self.dbeta = dbeta

    return dx

##### 오버피팅을 억제하기 위한 Dropout 클래스 #####
class Dropout:
    """
    http://arxiv.org/abs/1207.0580
    """

    def __init__(self, dropout_ratio=0.15):
        self.dropout_ratio = dropout_ratio
        self.mask = None

    def forward(self, x, train_flg=True):
        if train_flg:
            self.mask = np.random.rand(*x.shape) > self.dropout_ratio
            return x * self.mask
        else:
            return x * (1.0 - self.dropout_ratio)

    def backward(self, dout):
        return dout * self.mask

class TwoLayersNet:
    def __init__(self, input_size, hidden_size_1, hidden_size_2, output_size, weight_init_std=1):
        self.weight_decay_lambda = 0.001
        self.params = {}
        self.params['W1'] = np.sqrt(2 / 784) * np.random.randn(input_size, hidden_size_1)
        self.params['b1'] = np.zeros(hidden_size_1)
        self.params['W2'] = np.sqrt(2 / 50) * np.random.randn(hidden_size_1, hidden_size_2)
        self.params['b2'] = np.zeros(hidden_size_2)
        self.params['W3'] = np.sqrt(2 / 100) * np.random.randn(hidden_size_2, output_size)
        self.params['b3'] = np.zeros(output_size)

        self.layers = OrderedDict()

        self.layers['Affine1'] = Affine(self.params['W1'], self.params['b1'])
        self.layers['BatchNorm1'] = BatchNormalization(gamma=1.0, beta=0.)

```



```

self.layers['Relu1'] = Relu()

self.layers['Affine2'] = Affine(self.params['W2'], self.params['b2'])
self.layers['BatchNorm2'] = BatchNormalization(gamma=1.0, beta=0.)
self.layers['Relu2'] = Relu()

self.layers['Affine3'] = Affine(self.params['W3'], self.params['b3'])

self.lastLayer = SoftmaxWithLoss()

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

    return x

# L2 정규화 적용 후

def loss(self, x, t): # x : (100, 1024), t : (100, 10)

    y = self.predict(x) # (100, 10) : 마지막 출력층을 통과한 신경망이 예측한 값

    weight_decay = 0

    for idx in range(1, 4):
        W = self.params['W' + str(idx)]

        weight_decay += 0.5 * 0.001 * np.sum(W ** 2)

    return self.lastLayer.forward(y, t) + weight_decay

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

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

    return accuracy

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

    grads = {}
    grads['W1'] = numerical_gradient(loss_W, self.params['W1'])
    grads['b1'] = numerical_gradient(loss_W, self.params['b1'])
    grads['W2'] = numerical_gradient(loss_W, self.params['W2'])
    grads['b2'] = numerical_gradient(loss_W, self.params['b2'])
    grads['W3'] = numerical_gradient(loss_W, self.params['W3'])
    grads['b3'] = numerical_gradient(loss_W, self.params['b3'])
    return grads

def gradient(self, x, t):
    self.loss(x, t)

    dout = 1
    dout = self.lastLayer.backward(dout)

    layers = list(self.layers.values())
    layers.reverse()
    for layer in layers:
        dout = layer.backward(dout)

    grads = {}

```

```

        for idx in range(1, 4):

            grads['W' + str(idx)] = self.layers['Affine' + str(idx)].dW + self.weight_decay_lambda * self.layers['Affine' + str(idx)].W

            grads['b' + str(idx)] = self.layers['Affine' + str(idx)].db

        return grads

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

network = TwoLayersNet(input_size=784, hidden_size_1=50, hidden_size_2=100, output_size=10)

iters_num = 10000
train_size = x_train.shape[0]
batch_size = 100
learning_rate = 0.1

train_loss_list = []
train_acc_list = []
test_acc_list = []

iter_per_epoch = max(train_size / batch_size, 1)

##### SGD 클래스 #####
class SGD:
    def __init__(self, lr=0.01):
        self.lr = lr

    def update(self, params, grads):
        for key in params.keys():
            params[key] -= self.lr * grads[key]

# optimizer = SGD() # SGD 클래스를 객체화 시킨다.

##### Momentum 클래스 #####
class Momentum:
    def __init__(self, lr=0.01, momentum=0.9):
        self.lr = lr
        self.momentum = momentum
        self.v = None

    def update(self, params, grads):
        if self.v is None:
            self.v = {}
            for key, val in params.items():
                self.v[key] = np.zeros_like(val)

        for key in params.keys():
            self.v[key] = self.momentum * self.v[key] - self.lr * grads[key]
            params[key] += self.v[key]

# 속도 ← 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)
```

****결과**

```
0.3626833333333336 0.365
0.9565333333333333 0.9562
0.9595 0.9534
0.9653333333333334 0.9645
0.9616666666666667 0.961
0.9637666666666667 0.9607
0.9676833333333333 0.9649
0.9603 0.9552
0.9660333333333333 0.9636
0.9686833333333333 0.9643
0.9676833333333333 0.9611
0.9675666666666667 0.9633
0.9684666666666667 0.9666
0.9686 0.9623
0.9572166666666667 0.9491
0.9637 0.9558
0.9658166666666667 0.9629
```

7. CNN

2018년 8월 28일 화요일 오전 10:48

CNN (Convolution neural network)

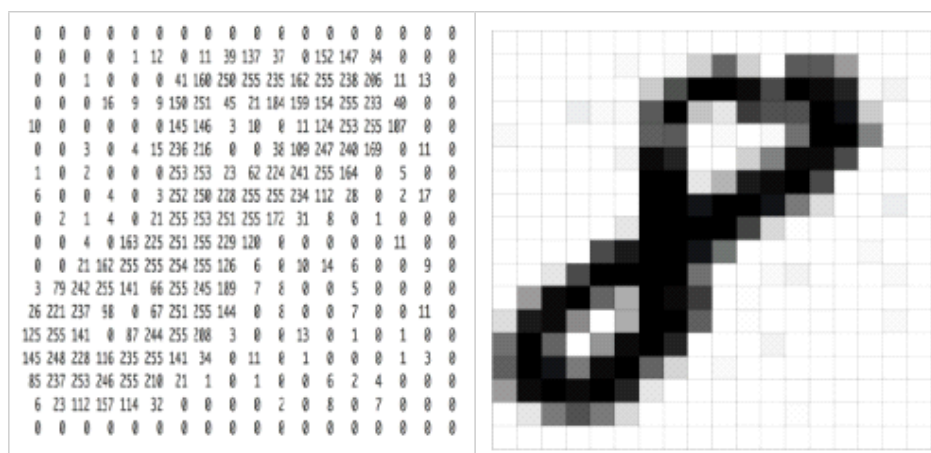
합성곱 신경망 ?

" Convolution 층과 pooling 층을 포함하는 신경망 "

기존 신경망과의 차이?

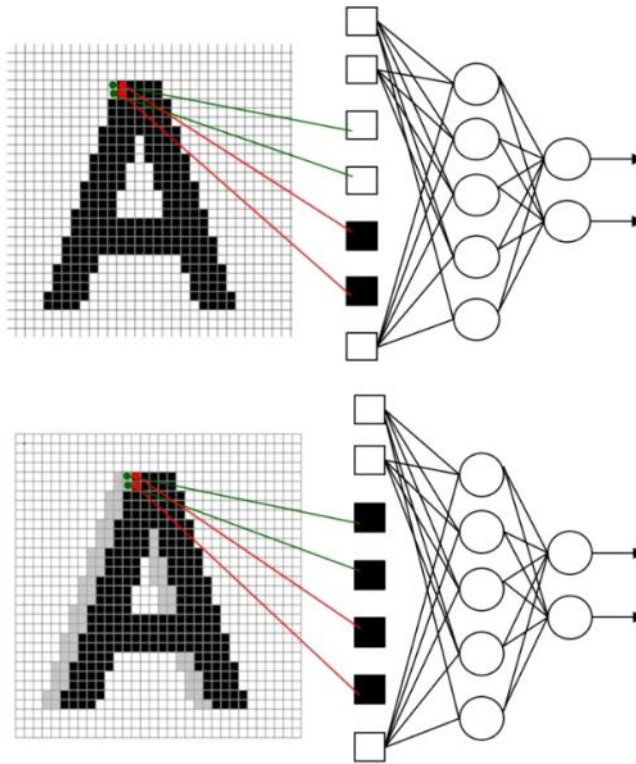
- 기존방법 : Affine ----> ReLu
- CNN : Conv ----> ReLu ----> Pooling

1. 아래의 숫자 8 필기체 784개의 픽셀값이 신경망에 입력이 되는데



2. CNN 을 이용하지 않았을때의 문제점은 ?

전체 글자에서 단지 2픽셀 값만 달라지거나 2픽셀씩 이동만 하더라도 새로운 학습 데이터로 처리를 해줘야 하는 문제점이 있다.



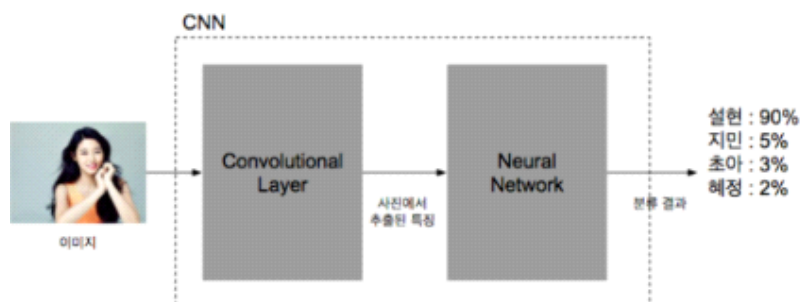
또한 글자의 크기가 달라지거나, 글자가 회전하거나, 글자에 변형 (distortion)이 조금만 생기더라도 새로운 학습 데이터를 넣어주지 않으면 좋은 결과를 기대하기 어렵다.



결론적으로 기존 multi - layerd neural network는 글자의 topology는 고려하지 않고, 말 그대로 raw data에 대해 직접적으로 처리하기 때문에 엄청나게 많은 학습 데이터를 필요로 하고, 또한 거기에 따른 학습 시간을 대가로 지불해야되는 문제점이있다.

3. 그래서 CNN을 사용하게 되면?

cnn은 전통적인 뉴럴 네트워크 앞에 여러 계층의 컨볼루션 계층을 붙인 모양이 되는데, 그 이유는 다음과 같다. cnn은 앞의 컨볼루션 계층을 통해서 입력 받은 이미지에 대한 특징(Feature)를 추출하게 되고, 이렇게 추출된 특징을 기반으로 기존의 뉴럴 네트워크를 이용하여 분류를 해내게 된다.



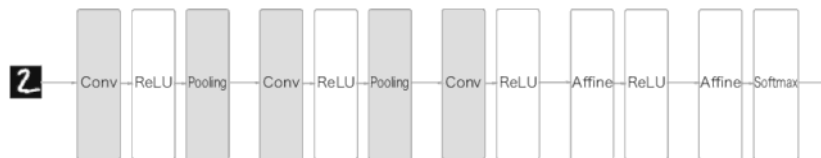
형상을 무시하고 모든 입력 데이터를 동등한 뉴런으로 취급하기 때문에 이미지가 갖는 본질적인 패턴을 읽지 못한다. ---> 그래서 합성곱이 필요하다.

결국 원본 이미지에서 조금만 모양이 달라져도 같은 이미지로 인식하지 못하는 문제를 합성곱이 해결해줄 수 있다.

** 어떻게 해결하는가?

원본이미지를 가지고 여러개의 feature map을 만들어서 분류하는 완전 연결 계층에 입력한다.

■ 합성곱 계층



"feature map을 만들고 그 feature map을 선명하게 해주는 층"

** 합성곱 연산 ? 이미지 3차원 (세로,가로,색상) data의 형상을 유지하면서 연산하는 작업

합성곱층 ----> 풀링층

** 합성곱층의 역할 ? 이미지에서 특징 (feature map)을 추출하는 작업

** pooling 층의 역할 ? 이미지를 더 선명하게

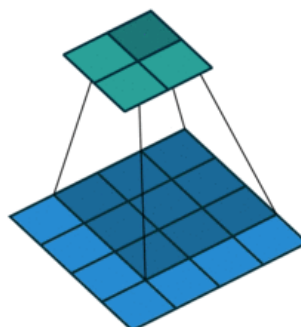
"입력 데이터에 필터를 적용한 것이 합성곱 연산이다. "

- 합성곱 연산을 컴퓨터로 구현하는 방법

$$\begin{array}{ccc}
 1 & 2 & 3 & 0 \\
 0 & 1 & 2 & 3 \\
 3 & 0 & 1 & 2 \\
 2 & 3 & 0 & 1
 \end{array}
 \odot
 \begin{array}{ccc}
 2 & 0 & 1 \\
 0 & 1 & 2 \\
 1 & 0 & 2
 \end{array}
 =
 \begin{array}{cc}
 15 & 16 \\
 6 & 15
 \end{array}$$

입력 데이터 (4,4) 필터 (3,3) (2,2)

■ Striding 1 의 Convolution 연산 (합성곱 연산)



** 출력크기 공식

입력크기를 (H,W), 필터크기를 (FH,FW),

출력크기를 (OH,OW), 패딩을 P, 스트라이드를 S라고 할 때 출력 크기의 공식

$$OH = \frac{H + 2P - FH}{S} + 1$$

$$OW = \frac{W + 2P - FW}{S} + 1$$

$$\text{----> } P(\text{패딩}) = (OH - 1) * S - H + FH / 2$$

■ 3차원의 합성곱 (p235)

이미지의 색깔이 보통은 흑백이 아니라 RGB 컬러이므로 RGB(Red,Green,Blue) 컬러에 대해서 합성곱을 해야한다.

■ 합성곱 총정리

합성곱 ? 이미지의 특징 (features amp)을 추출하는 과정
filter(가중치)를 이용해서 추출한다.

원본이미지 1장 * 필터 50개 = 50개의 feature map의 갯수

3. 합성곱 연산

" 합성곱층의 역할을 컴퓨터로 구현 "

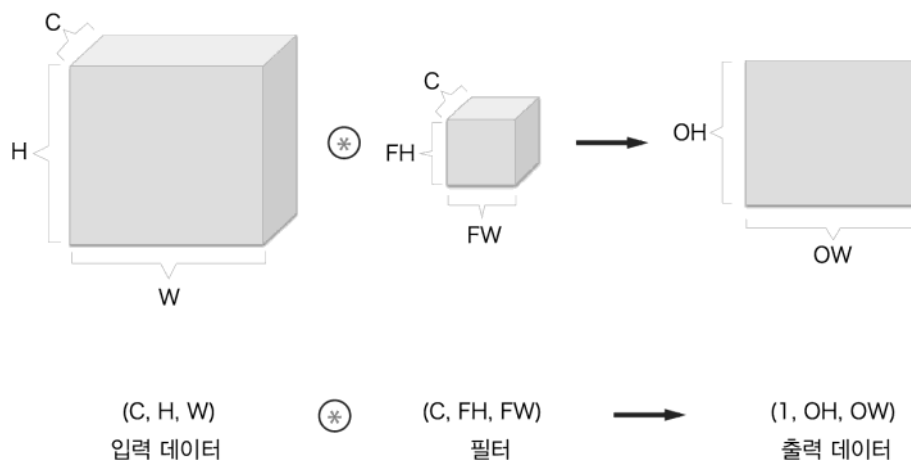
원본 이미지를 필터의 크기로 스트라이드 하면서 feature map을 출력하는 연산

4. 3차원 합성곱

" RGB 이미지를 RGB 필터로 합성곱 연산 "

■ 블록으로 생각하기 (p237)

3차원 합성곱 연산은 데이터와 필터를 직육면체 블록이라고 생각하면 쉽다.



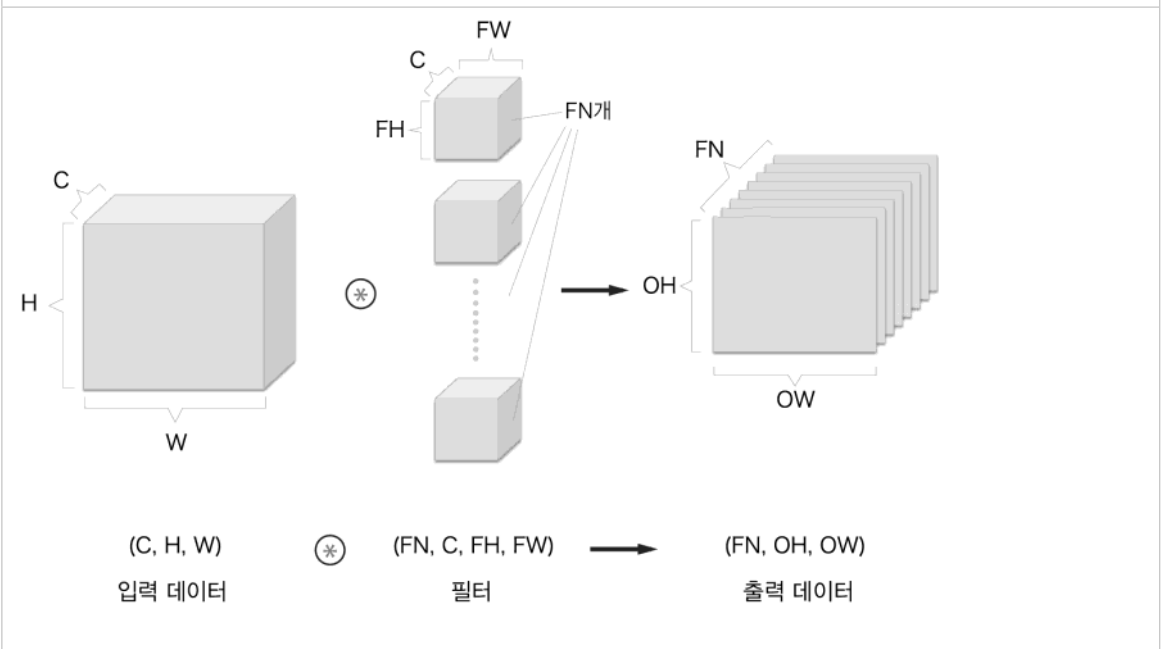
#필터 한 장당 feature map 한 장 나옴

위의 그림은 feature map이 한 개가 나오고 있는데 실제로는 아이린 사진 한 장에 대해서 여러개의 feature map이 필요하다. 여러개의 feature map을 출력하려면 어떻게 해야하는가?

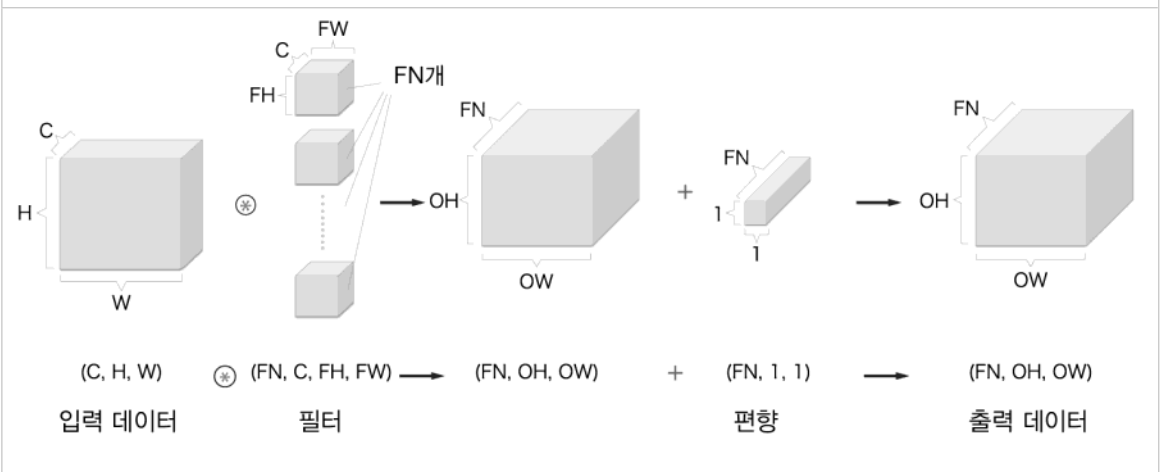
채널 수 : C
 높이 : H
 너비 : W
 FH : 필터 높이
 FW : 필터 너비

여러개의 feature map을 출력하려면 어떻게 해야하는가?

-->filter 의 갯수를 늘린다

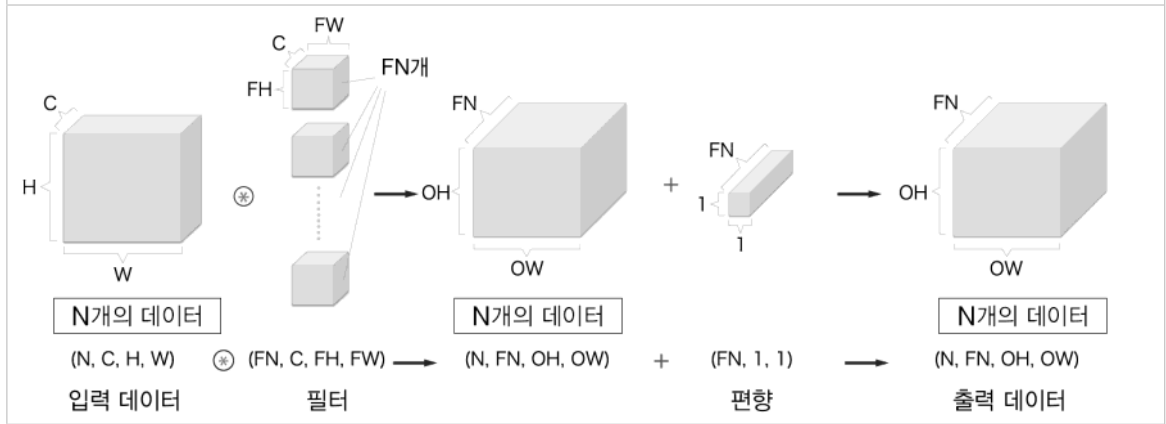


합성곱 연산에서도 편향이 쓰이므로 편향을 더하면 어떤 그림일까?



위의 그림은 이미지를 1장씩 넣어서 학습 시키는 것이므로 학습속도가 느리므로 여러장의

이미지를 한 번에 입력해서 학습시키면 (mini batch) 아래의 그림이 된다.

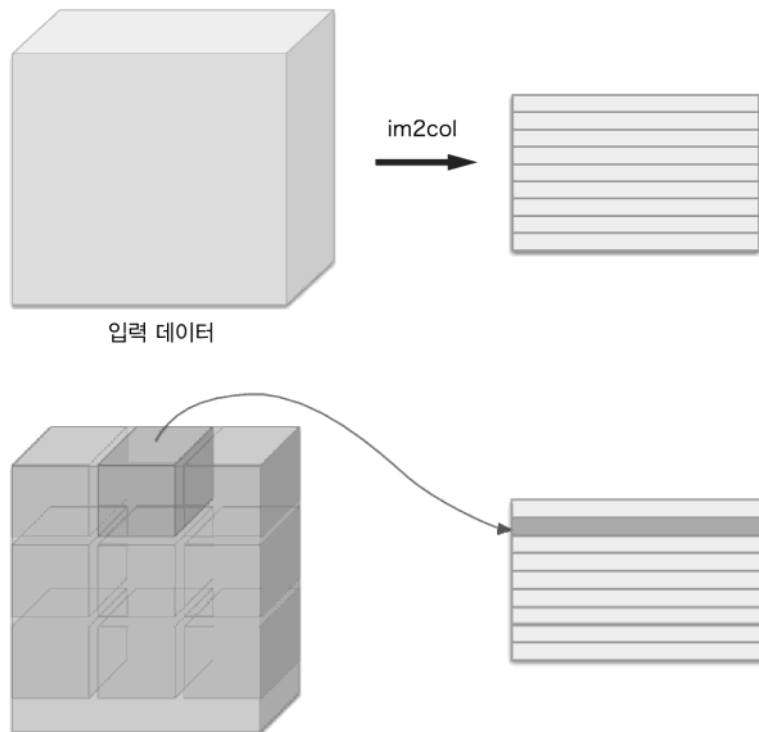


그러면 결국 합성곱 계층을 구현할 때 흘러가는 4차원 행렬이 연산되는데, 그러면 연산 속도가 느리므로 행렬 연산을 빠르게 하려면 4차원이 아니라 2차원으로 차원 축소를 해야한다.

그래서 필요한 함수 ?

" im2col 함수 " <---- p 234

4차원 -----> 2차원



합성곱 연산을 빠르게 진행하기 위해 im2col 함수를 이용했다.

--2차원으로 변경해야 할 행렬 2가지

1. 원본 이미지를 필터 사이즈에 맞게 2차원으로 변경한 행렬 (im2col 함수 사용)

원본이미지 (장 수, 채널, 높이, 너비) [4차원] ---- im2col함수 -----> (높이, 너비) [2차원]

2. 4차원 필터 행렬을 2차원으로 변경

원본 필터 (필터의 개수, 채널, 높이, 너비) ---- reshape함수 -----> (높이, 너비) [2차원]

■ convolution 클래스 내에서 일어나는 일

1. 원본 이미지를 im2col로 2차원 행렬로 변경 한다.
2. filter 를 reshape함수를 이용해서 2차원 행렬로 변경한다.
3. 2차원 행렬로 변경한 두 행렬을 내적한다.
4. 내적인 결과인 2차원 행렬을 다시 4차원으로 변경한다.

■ pooling

풀링층의 역할은 말 그대로 출력값에서 일부분만 취하는 기능이다.

convolution이 이렇게 저렇게 망쳐놓은 그림들을 각 부분에서 대표들을 뽑아 사이즈가 작은 이미지를 만드는 것이다. 마치 사진을 축소하면 해상도가 좋아지는 듯한 효과와 비슷하다.

풀링의 종류 3가지

1. 최대 풀링 : 컨볼루션 데이터에서 가장 큰 값을 대표값으로 선정한다.



2. 평균 풀링 : 컨볼루션 데이터에서 모든 값의 평균 값을 대표값으로 선정
3. 확률적 풀링 : 컨볼루션 데이터에서 임의 확률로 한 개를 선정

■ CNN 층의 구조

conv 층 -----> pooling 층 -----> fully connected 층

convolution : 이미지의 특징[feature map]을 추출하는 층

pooling 층 :

fully connected (완전 연결 계층 [Affine]) :

문제 173. 아래의 두 행렬을 만들고 합성곱 한 결과인 15를 파이썬으로 출력 하시오.

```
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]])

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

****결과**
[[2 0 3]
[0 1 4]
[3 0 2]]
15

문제 174. 아래의 4x4행렬에서 아래의 3x3 행렬만 추출 하시오.

```
[[1 2 3 0]
 [0 1 2 3]
 [3 0 1 2]
 [2 3 0 1]]
---->

[[1 2 3]
 [0 1 2]
 [3 0 1]]
```

```
import numpy as np

a = np.array([[1,2,3,0],[0,1,2,3],[3,0,1,2],[2,3,0,1]])

b = np.array([a[0][:3],a[1][:3],a[2][:3]])
print(b)
```

****결과**
[[1 2 3]
[0 1 2]
[3 0 1]]

문제 175. 아래의 행렬에서 아래의 결과 행렬을 추출 하시오.

```
[[1 2 3 0]
 [0 1 2 3]
 [3 0 1 2]
 [2 3 0 1]]
```

```
import numpy as np
```

```

a = np.array([[1,2,3,0],[0,1,2,3],[3,0,1,2],[2,3,0,1]])
print(a[0:3,0:3])
print()
print(a[0:3,1:4])
print()
print(a[1:4,0:3])
print()
print(a[1:4,1:4])

```

****결과**

```

[[1 2 3]
 [0 1 2]
 [3 0 1]]
[[2 3 0]
 [1 2 3]
 [0 1 2]]
[[0 1 2]
 [3 0 1]
 [2 3 0]]
[[1 2 3]
 [0 1 2]
 [3 0 1]]

```

문제 176.

아래의 4x4 행렬에서 아래의 결과를 출력 하시오.

| | | | |
|------------|---|----------|-----------------|
| [[1 2 3 0] | ◎ | [[2 0 1] | [15, 16, 6, 15] |
| [0 1 2 3] | | [0 1 2] | |
| [3 0 1 2] | | [1 0 2]] | |
| [2 3 0 1]] | | | |

```
import numpy as np
```

```

a = np.array([[1,2,3,0],[0,1,2,3],[3,0,1,2],[2,3,0,1]])
print(a,end='\n\n')
print(a[0:3,0:3],end='\n\n')
print(a[0:3,1:4],end='\n\n')
print(a[1:4,0:3],end='\n\n')
print(a[1:4,1:4],end='\n\n')

```

```

n=3
lst=[]
for i in range(len(a[0])):
    for j in range(len(a)):
        if i+n <= len(a) and j+n <= len(a[0]):
            lst.append(np.sum(a[i:i+n,j:j+n]))

print(lst)

```

****결과**

```

[[1 2 3 0]
 [0 1 2 3]

```

```
[3 0 1 2]
[2 3 0 1]]
[[1 2 3]
[0 1 2]
[3 0 1]]
[[2 3 0]
[1 2 3]
[0 1 2]]
[[0 1 2]
[3 0 1]
[2 3 0]]
[[1 2 3]
[0 1 2]
[3 0 1]]
[13, 14, 12, 13]
```

문제 177. 아래의 합성곱을 파이썬으로 구현 하시오.

```
import numpy as np

a = np.array([[1,2,3,0],[0,1,2,3],[3,0,1,2],[2,3,0,1]])
b = np.array([[2,0,1],[0,1,2],[1,0,2]])

n=3
lst=[]
for i in range(len(a[0])):
    for j in range(len(a)):
        if i+n <= len(a) and j+n <= len(a[0]):
            c= a[i:i+n,j:j+n] * b
            lst.append(np.sum(c))

print(lst)

**결과
[15, 16, 6, 15]
```

문제 178. 위에서 출력한 결과인 1차원 배열 [15, 16, 6, 15] 결과를 2x2 행렬로 변경 하시오.

```
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]])

result = []
for rn in range(len(a) - 2):
    for cn in range(len(a) - 2):
        result.append( np.sum(a[rn:rn+3,cn:cn+3] * filter) )

print (result)

result = np.array(result).reshape(2,2)
print(result)
```

****결과**

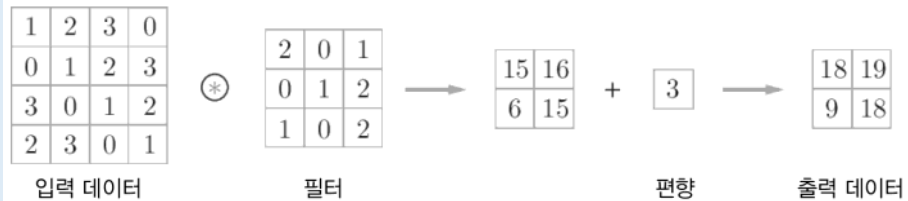
[15, 16, 6, 15]

[[15 16]

[6 15]]

문제 179.

아래의 그림의 convolution 연산을 파이썬으로 구현 하시오.



```
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]])
b=3
result = []
for rn in range(len(a) - 2):
    for cn in range(len(a) - 2):
        result.append( np.sum(a[rn:rn+3,cn:cn+3] * filter) )
```

```
print (result)
```

```
result = np.array(result).reshape(2,2)
print(result+b)
```

****결과**

[15, 16, 6, 15]

[[18 19]

[9 18]]

문제 180.

위에서 출력한 2x2 행렬에 제로패딩 1을 수행 하시오.

```
import numpy as np

result = np.array([[15,16],[6,15]])
result_pad = np.pad(result, pad_width=1,mode='constant',constant_values=0)

print(result_pad)
```

****결과**

[[0 0 0 0]

[0 15 16 0]

[0 6 15 0]

[0 0 0 0]]

문제 181.

4x4 행렬에 3x3 필터를 적용해서 결과로 4x4 행렬이 출력되게 하려면 제로패딩을 몇으로 해줘야 하는가?

```
import numpy as np

result = np.array([[15,16],[6,15]])
result_pad = np.pad(result, pad_width=1,mode='constant',constant_values=0)

print(result_pad)

**결과
[[ 0  0  0  0]
 [ 0 15 16  0]
 [ 0  6 15  0]
 [ 0  0  0  0]]
```

문제 182. 패딩을 답으로 해서 아래의 식을 풀어보시오.

$$OH = \frac{H + 2P - FH}{S} + 1$$

$$OW = \frac{W + 2P - FW}{S} + 1$$

입력크기를 (H,W), 필터크기를 (FH,FW),
출력크기를 (OH,OW), 패딩을 P, 슬라이드를 S라고 할 때 출력 크기의 공식

$$P = (OH - 1) * S - H + FH / 2$$

문제 183. 입력 이미지 4x4 행렬에 필터 3x3 행렬을 합성곱 한 출력 결과 행렬이 4x4 행렬이 되려면 패딩이 몇인지 출력 하시오.

$$P = (4 - 1) * 1 - 4 + 3 / 2$$

****출력결과**

1

문제 184. 위의 패딩 공식을 구현하는 파이썬 함수를 구현 하시오.

```
def padding(h,s,oh,fh):
    return (s*(oh-1)-h+fh)//2
```

문제 185. 입력행렬(6x6), 슬라이드 1, 출력행렬 (6x6), 필터행렬 (3x3)의 패딩이 어떻게 되는지 출력 하시오.

```
def padding(h,s,oh,fh):
    return (s*(oh-1)-h+fh)//2
```

```
print(padding(6,1,6,3))
```

****결과**

1

문제 186. 레드벨벳의 아이린 사진을 3차원 행렬로 변환하시오.


```

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:\\data\\irin.jpg')
img_pixel = np.array(img)
print(img_pixel)
print(img_pixel.shape)
plt.imshow(img_pixel)

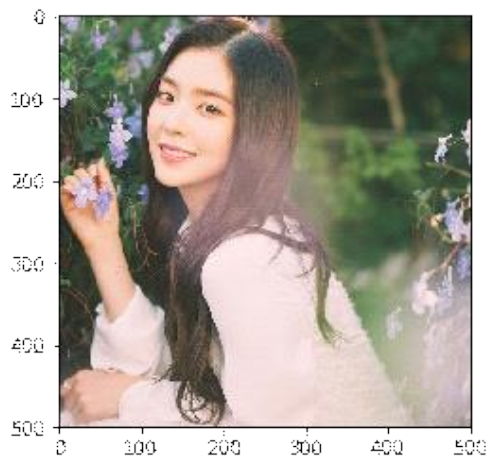
```

****결과**

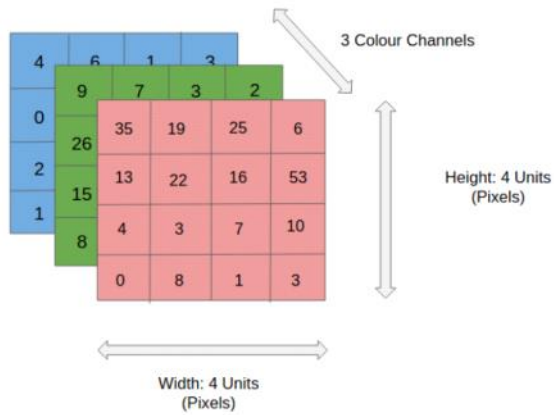
```

[[[ 79  96 104]
  [113 118 138]
  [147 137 172]
  ...
  [ 50  72  51]
  [ 50  72  51]
  [ 50  72  51]]
 [[101 118 128]
  [137 142 162]
  [169 159 194]
  ...
  [165 150 147]
  [176 160 160]
  [184 168 169]]]
(500, 500, 3)

```



##설명



레드,그린,블루의 3차원 배열로 이루어져 있다.

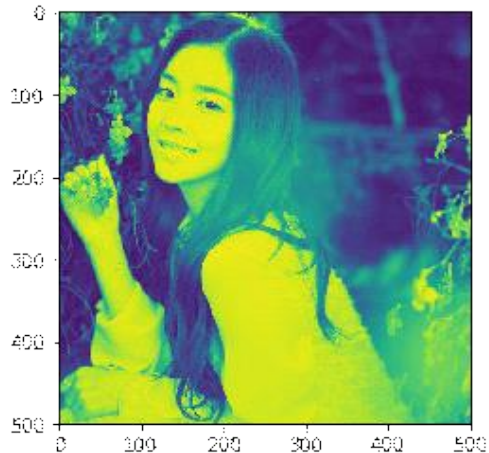
문제 187. 레드벨벳의 아이린 사진에서 Red 행렬만 출력하고 Red행렬만 시각화 하시오.

```
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

img = Image.open('c:\\data\\irin.jpg')

img_pixel = np.array(img)
plt.imshow(img_pixel[:, :, 0])
```

****결과**



##설명

plt.imshow(img_pixel[:, :, 0]) # 0: 레드, 1: 그린, 2: 블루

문제 188. 원하는 사진을 선택해서 RGB를 확인 하시오.

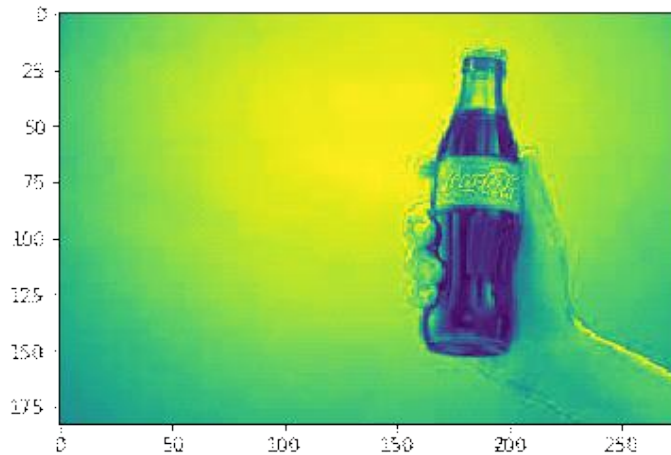
```
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
```

```
img = Image.open('c:\\data\\tt.jpg')
```

```
img_pixel = np.array(img)
print(img_pixel.shape)
plt.imshow(img_pixel[:, :, 0])
```

****결과**

(183, 275, 3)



문제 189. R,G,B 행렬을 이해하기 위한 아래의 numpy array 를 이해하시오 !

```
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], #레드

         [[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], #그린

         [[4, 2, 1, 2],
          [0, 1, 0, 4],
          [3, 0, 6, 2],
          [4, 2, 4, 5]] #블루
    ])

```

문제 190. 문제 189번 data행렬의 shape를 확인 하시오.

```
import numpy as np
```

```
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, 1],
[0, 1, 0, 4, 1],
[3, 0, 6, 2, 0],
[4, 2, 4, 5, 2],
[2, 0, 2, 2, 2]]
])

```

```
print(data.shape)
```

****결과**

(3, 5, 5)

문제 191. 위의 data 행렬에서 Red 행렬만 출력 하고 시각화 하시오.

```

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, 1],
[0, 1, 0, 4, 1],
[3, 0, 6, 2, 0],
[4, 2, 4, 5, 2],
[2, 0, 2, 2, 2]]
])

```

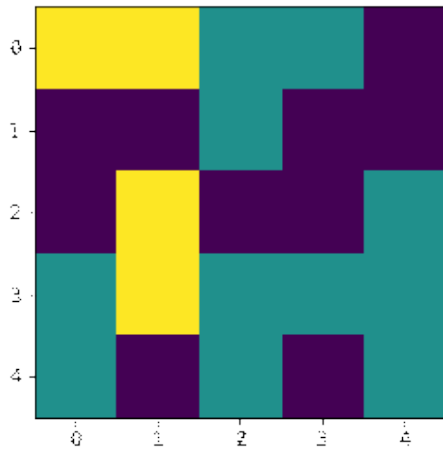
```

print(data[0])
plt.imshow(data[0])

```

****결과**





문제 192. 아래의 numpy array의 행렬을 확인 하시오.

```
import numpy as np

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

print(Filter)
print(Filter.shape)
```

****결과**

```
[[[ 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]]]
(1, 3, 9)
```

문제 193. 위의 행렬을 (3,3,3)행렬로 변환 하시오.

```
import numpy as np

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(Filter.shape)
```

****결과**

```
[[[ 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 (3,3,3) 행렬을 시각화 하시오.

```
from PIL import Image  
import numpy as np  
import matplotlib.pyplot as plt  
import matplotlib.image as mpimg  
  
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)
```

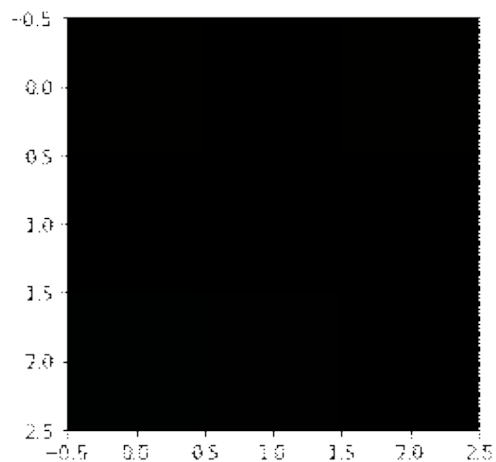
```
print(Filter)  
print(Filter.shape)
```

```
plt.imshow(Filter)
```

****결과**

```
[[[ 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)
```



0, -1 만 존재하므로 검정색만 출력된다.

문제 195. 위의 (3,3,3) 행렬의 Red, Green, Blue 행렬을 각각 시각화 하시오.

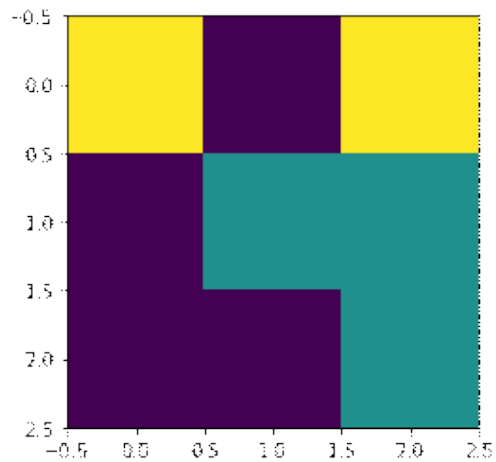
```
from PIL import Image  
import numpy as np  
import matplotlib.pyplot as plt
```

```
import matplotlib.image as mpimg

Filter = np.array([[[1,1,-1,-1,0,0,1,1,0],
                    [-1,-1,0,0,-1,1,0,-1,0],
                    [-1,1,1,-1,1,-1,0,0,-1]]]).reshape(3,3,3)

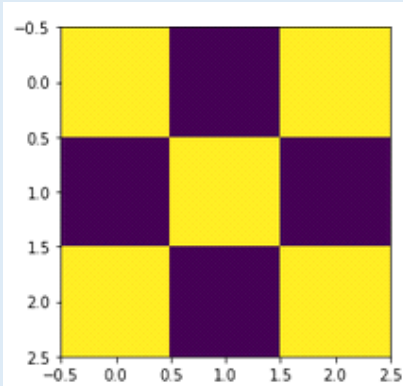
plt.imshow(Filter[:,0])
plt.show()
plt.imshow(Filter[:,1])
plt.show()
plt.imshow(Filter[:,2])
plt.show()
```

****결과**



문제 196.

아래의 필터를 생성 하시오.

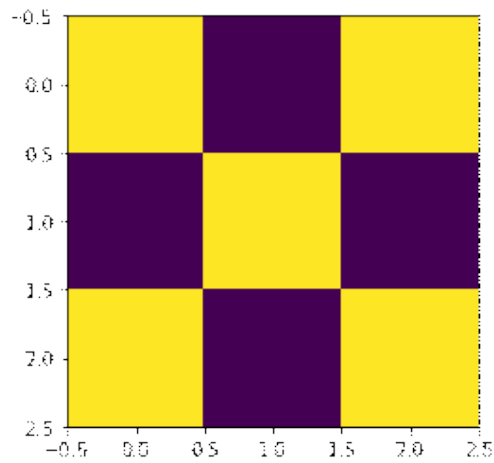


```
filter= np.array([[[1,-1,1],[-1,1,-1],[1,-1,1]])
print(filter.shape)
plt.imshow(filter[0,:])
plt.show()
```

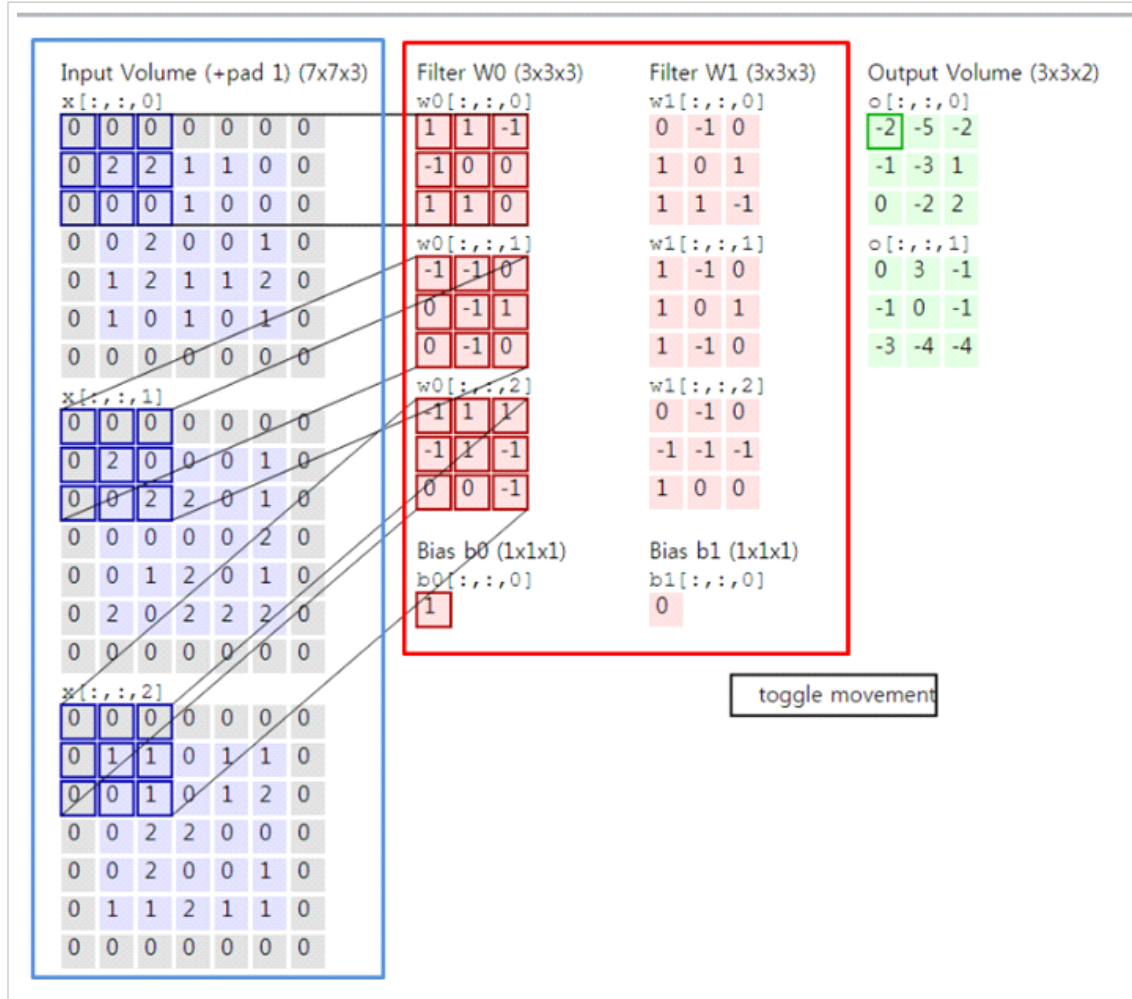
****결과**

(1, 3, 3)





문제 197. 아래의 원본 이미지인 data 행렬과 filter 행렬을 3차원 합성곱 하는 그림을 이해하시오.



문제 198. 아래의 원본 이미지의 zero 패딩 1한 결과를 출력 하시오.

```
data = np.array([
    [2, 2, 1, 1, 0],
    [0, 0, 1, 0, 0],
    [0, 2, 0, 0, 1],
    [1, 2, 1, 1, 1], # ---> Red
    [1, 0, 1, 0, 1],
    [2, 0, 0, 0, 1],
```



```

    [0, 2, 2, 0, 1],
    [0, 0, 0, 0, 2], # ----> Green
    [0, 1, 2, 0, 1],
    [2, 0, 2, 2, 2]],
    [[4, 2, 1, 2, 2],
    [0, 1, 0, 4, 1], # ----> Blue
    [3, 0, 6, 2, 1],
    [4, 2, 4, 5, 4],
    [0, 1, 2, 0, 1]]
])

```

```
data_pad = np.pad(data, pad_width=1, mode='constant', constant_values=0)[1:4]
```

```
#data_pad = np.pad(data, pad_width=((0,0),(1,1),(1,1)), mode='constant', constant_values=0) # 또 다른 방법
print(data_pad)
```

****결과**

```

[[[0 0 0 0 0 0]
  [0 2 2 1 1 0]
  [0 0 0 1 0 0]
  [0 0 2 0 0 1]
  [0 1 2 1 1 1]
  [0 1 0 1 0 1]
  [0 0 0 0 0 0]]
 [[0 0 0 0 0 0]
  [0 2 0 0 0 1]
  [0 0 2 2 0 1]
  [0 0 0 0 0 2]
  [0 0 1 2 0 1]
  [0 2 0 2 2 2]
  [0 0 0 0 0 0]]
 [[0 0 0 0 0 0]
  [0 4 2 1 2 2]
  [0 0 1 0 4 1]
  [0 3 0 6 2 1]
  [0 4 2 4 5 4]
  [0 0 1 2 0 1]
  [0 0 0 0 0 0]]]

```

문제 199. 아래의 코드를 이해하시오.

```

data = np.array(
    [
        [2, 2, 1, 1, 0],
        [0, 0, 1, 0, 0],
        [0, 2, 0, 0, 1],
        [1, 2, 1, 1, 1], # ---> Red
        [1, 0, 1, 0, 1]],
    [[2, 0, 0, 0, 1],
    [0, 2, 2, 0, 1],
    [0, 0, 0, 0, 2], # ----> Green

```

```

    [0, 1, 2, 0, 1],
    [2, 0, 2, 2, 2]],
    [[4, 2, 1, 2, 2],
    [0, 1, 0, 4, 1],    # ----> Blue
    [3, 0, 6, 2, 1],
    [4, 2, 4, 5, 4],
    [0, 1, 2, 0, 1]]
])

```

```

data_pad = np.pad(data, pad_width=((0,0),(1,1),(1,1)),mode='constant',constant_values=0) # 또 다른 방법
print(data_pad)

```

****결과**

```

[[[0 0 0 0 0 0]
  [0 2 2 1 1 0]
  [0 0 0 1 0 0]
  [0 0 2 0 0 1]
  [0 1 2 1 1 0]
  [0 1 0 1 0 1]
  [0 0 0 0 0 0]]
 [[0 0 0 0 0 0]
  [0 2 0 0 0 1]
  [0 0 2 2 0 1]
  [0 0 0 0 0 2]
  [0 0 1 2 0 1]
  [0 2 0 2 2 0]
  [0 0 0 0 0 0]]
 [[0 0 0 0 0 0]
  [0 4 2 1 2 2]
  [0 0 1 0 4 1]
  [0 3 0 6 2 1]
  [0 4 2 4 5 4]
  [0 0 1 2 0 1]
  [0 0 0 0 0 0]]

```

문제 200. 한 개의 zero 패딩한 Red 행렬에서 아래의 행렬만 추출 하시오.

```

data = np.array(
    [
        [[2, 2, 1, 1, 0],
         [0, 0, 1, 0, 0],
         [0, 2, 0, 0, 1],
         [1, 2, 1, 1, 1],    # ---> Red
         [1, 0, 1, 0, 1]],
        [[2, 0, 0, 0, 1],
         [0, 2, 2, 0, 1],
         [0, 0, 0, 0, 2],    # ----> Green
         [0, 1, 2, 0, 1],
         [2, 0, 2, 2, 2]],
        [[4, 2, 1, 2, 2],

```

```

[0, 1, 0, 4,1], # ----> Blue
[3, 0, 6, 2,1],
[4, 2, 4, 5,4],
[0, 1, 2, 0, 1]]
])

```

```
data_pad = np.pad(data, pad_width = ((0,0),(1,1),(1,1)), mode = 'constant', constant_values = 0)
```

```
print(data_pad[0,:3,:3])
```

```
print(data[0][0:3,0:3])
```

****결과**

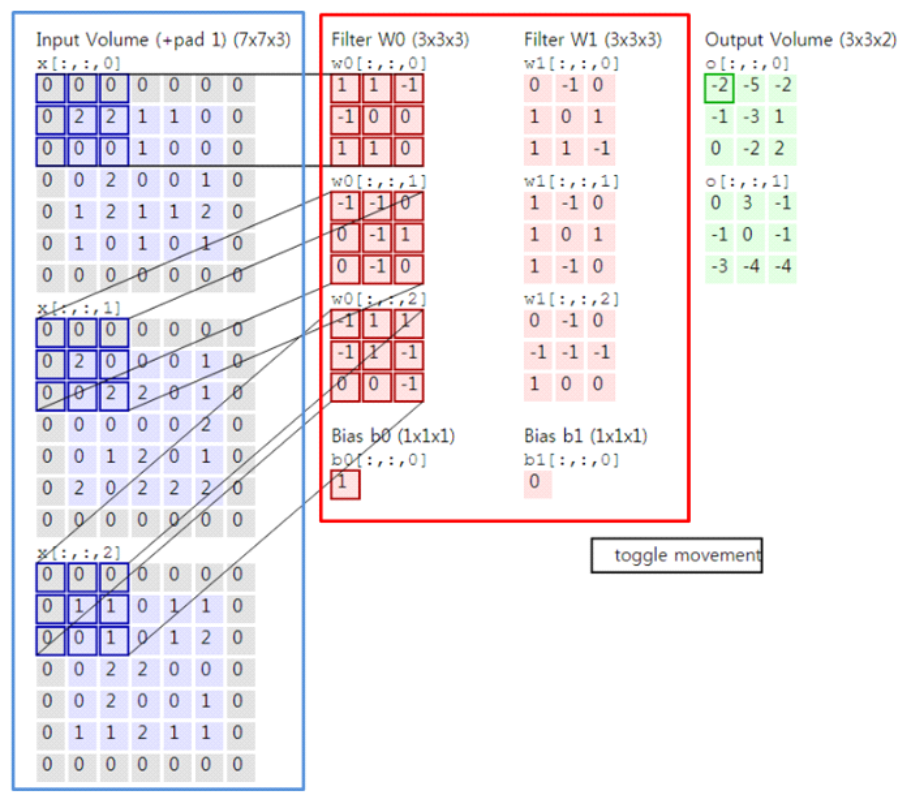
```
[[0 0 0]
```

```
[0 2 2]
```

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

문제 201.

위의 3x3 행렬의 R,G,B 행렬을 각각 출력하면 ?



```

[[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]]

```

```
import numpy as np
```

```

data = np.array(
    [
        [[2, 2, 1, 1, 0],
         [0, 0, 1, 0, 0],
         [0, 2, 0, 0, 1],
         [1, 2, 1, 1, 1], # ---> Red
         [1, 0, 1, 0, 1]],
        [[2, 0, 0, 0, 1],
         [0, 2, 2, 0, 1],
         [0, 0, 0, 0, 2], # ----> Green
         [0, 1, 2, 0, 1],
         [2, 0, 2, 2, 2]],
        [[1, 1, 0, 1, 1],
         [0, 1, 0, 1, 2], # ----> Blue
         [0, 2, 2, 0, 0],
         [0, 2, 0, 0, 1],
         [1, 1, 2, 1, 1]]
    ])

pad=np.pad(data,pad_width=((0,0),(1,1),(1,1)),
            mode='constant', constant_values=0)
print(pad[0,:3,:3])
print(pad[1,:3,:3])
print(pad[2,:3,:3])

```

****결과**

```

[[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 추출 하시오.

```

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,:,:])

```

****결과**

```

[[ 1  1 -1]
 [-1  0  0]
 [ 1  1  0]]

```

문제 203. 위의 3개의 행렬 중 원본 이미지의 Red 행렬과 아래의 filter의 Red 행렬과의 곱을 수행하시오.

| | | |
|----------|----------|---------|
| [[0 0 0] | 1 1 -1 | 0 0 0 |
| [0 2 2] | * -1 0 0 | = 0 0 0 |
| [0 0 0]] | 1 1 1 | 0 0 0 |

```
data = np.array(
[
    [[2, 2, 1, 1, 0],
     [0, 0, 1, 0, 0],
     [0, 2, 0, 0, 1],
     [1, 2, 1, 1, 1], # ---> Red
     [1, 0, 1, 0, 1]],
    [[2, 0, 0, 0, 1],
     [0, 2, 2, 0, 1],
     [0, 0, 0, 0, 2], # ----> Green
     [0, 1, 2, 0, 1],
     [2, 0, 2, 2, 2]],
    [[1, 1, 0, 1, 1],
     [0, 1, 0, 1, 2], # ----> Blue
     [0, 2, 2, 0, 0],
     [0, 2, 0, 0, 1],
     [1, 1, 2, 1, 1]]
])

pad=np.pad(data,pad_width=((0,0),(1,1),(1,1)),
            mode='constant', constant_values=0)

Filter=np.array([[[1,1,-1,-1,0,0,1,1,0], \
                  [-1,-1,0,0,-1,1,0,-1,0], \
                  [-1,1,1,-1,1,-1,0,0,-1]]]).reshape(3,3,3)

print(pad[0,:3,:3]*Filter[0,:,:])

**결과
[[0 0 0]
 [0 0 0]
 [0 0 0]]
```

문제 204. 아래의 원본 이미지 RGB 3개의 행렬과 아래의 필터 RGB 3개의 행렬을 각각 행렬곱 한 후 그 원소들을 다 합친 결과 숫자 하나를 출력 하시오. (3차원 합성곱 연산)

```
data = np.array(
[
    [[2, 2, 1, 1, 0],
     [0, 0, 1, 0, 0],
     [0, 2, 0, 0, 1],
     [1, 2, 1, 1, 1], # ---> Red
     [1, 0, 1, 0, 1]],
    [[2, 0, 0, 0, 1],
     [0, 2, 2, 0, 1],
     [0, 0, 0, 0, 2], # ----> Green
     [0, 1, 2, 0, 1],
     [2, 0, 2, 2, 2]],
    [[1, 1, 0, 1, 1],
     [0, 1, 0, 1, 2], # ----> Blue
     [0, 2, 2, 0, 0],
     [0, 2, 0, 0, 1],
     [1, 1, 2, 1, 1]]
])
```

```

))

pad=np.pad(data,pad_width=((0,0),(1,1),(1,1)),
           mode='constant', constant_values=0)

Filter=np.array([[[1,1,-1,-1,0,0,1,1,0], \
                  [-1,-1,0,0,-1,1,0,-1,0], \
                  [-1,1,1,-1,1,-1,0,0,-1]]]).reshape(3,3,3)

r=pad[0,:3,:3]*Filter[0,:,:]
b=pad[1,:3,:3]*Filter[1,:,:]
g=pad[2,:3,:3]*Filter[2,:,:]

rst = r+b+g
print(np.sum(rst))

```

****결과**

-3

문제 205. 1칸 스트라이드 한 원본 이미지의 3x3 행렬 RGB와 Filter RGB와의 합성곱을 구하시오.

$$\begin{bmatrix} 0 & 0 & 0 \\ 2 & 2 & 1 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 1 & -1 \\ -1 & 0 & 0 \\ 1 & 1 & 0 \end{bmatrix} = \begin{bmatrix} -2 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 0 & 0 \\ 2 & 0 & 0 \\ 0 & 2 & 2 \end{bmatrix} * \begin{bmatrix} -1 & -1 & 0 \\ 0 & -1 & 1 \\ 0 & -1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & -2 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix} * \begin{bmatrix} -1 & 1 & 1 \\ -1 & 1 & -1 \\ 0 & 0 & -1 \end{bmatrix} = \begin{bmatrix} -1 & 1 & 0 \\ -1 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

```

data = np.array(
[
    [2, 2, 1, 1, 0],
    [0, 0, 1, 0, 0],
    [0, 2, 0, 0, 1],
    [1, 2, 1, 1, 1], # ---> Red
    [1, 0, 1, 0, 1],
    [2, 0, 0, 0, 1],
    [0, 2, 2, 0, 1],
    [0, 0, 0, 0, 2], # ----> Green
    [0, 1, 2, 0, 1],
    [2, 0, 2, 2, 2]],
[[1, 1, 0, 1, 1],
 [0, 1, 0, 1, 2], # ----> Blue
 [0, 2, 2, 0, 0],
 [0, 2, 0, 0, 1],
 [1, 1, 2, 1, 1]]
)

pad=np.pad(data,pad_width=((0,0),(1,1),(1,1)),
           mode='constant', constant_values=0)

Filter=np.array([[[1,1,-1,-1,0,0,1,1,0], \
                  [-1,-1,0,0,-1,1,0,-1,0], \
                  [-1,1,1,-1,1,-1,0,0,-1]]]).reshape(3,3,3)

```

```

r=pad[0,:3,1:4]*Filter[0,:,:]
b=pad[1,:3,1:4]*Filter[1,:,:]
g=pad[2,:3,1:4]*Filter[2,:,:]

```

```

rst = r+b+g
print(np.sum(rst))

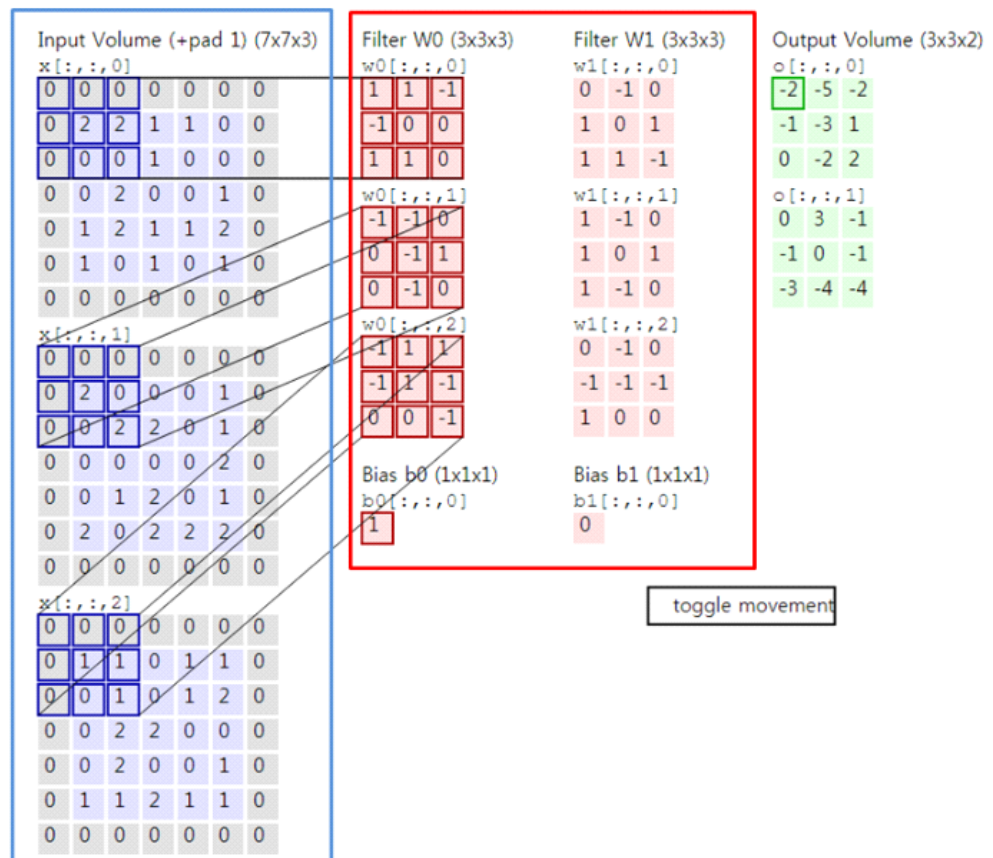
```

****결과**

-4

문제 206.

아래의 합성곱 연산 (25번 연산)을 하는 결과를 파이썬으로 구현 하시오.



```

data = np.array(
[
    [[2, 2, 1, 1, 0],
    [0, 0, 1, 0, 0],
    [0, 2, 0, 0, 1],
    [1, 2, 1, 1, 1], # ---> Red
    [1, 0, 1, 0, 1]],
    [[2, 0, 0, 0, 1],
    [0, 2, 2, 0, 1],
    [0, 0, 0, 0, 2], # ----> Green
    [0, 1, 2, 0, 1],
    [2, 0, 2, 2, 2]],
    [[1, 1, 0, 1, 1],
    [0, 1, 0, 1, 2], # ----> Blue
    [0, 2, 2, 0, 0],
    [0, 2, 0, 0, 1],
    [1, 1, 2, 1, 1]]
)

```

```

))

pad=np.pad(data,pad_width=((0,0),(1,1),(1,1)),
           mode='constant', constant_values=0)

Filter=np.array([[[1,1,-1,-1,0,0,1,1,0], \
                  [-1,-1,0,0,-1,1,0,-1,0], \
                  [-1,1,1,-1,1,-1,0,0,-1]]]).reshape(3,3,3)
rst=[]
for d in range(len(pad)):
    for i in range(len(pad[0])-len(Filter)+1):
        for j in range(len(pad[0])-len(Filter)+1):
            r=pad[d,i:i+len(Filter),j:j+len(Filter)]*Filter[d,:,:]
            rst.append(np.sum(r))

rst=np.array(rst).reshape(3,5,5)
print(rst,end='\n\n')

print(rst[0]+rst[1]+rst[2])

```

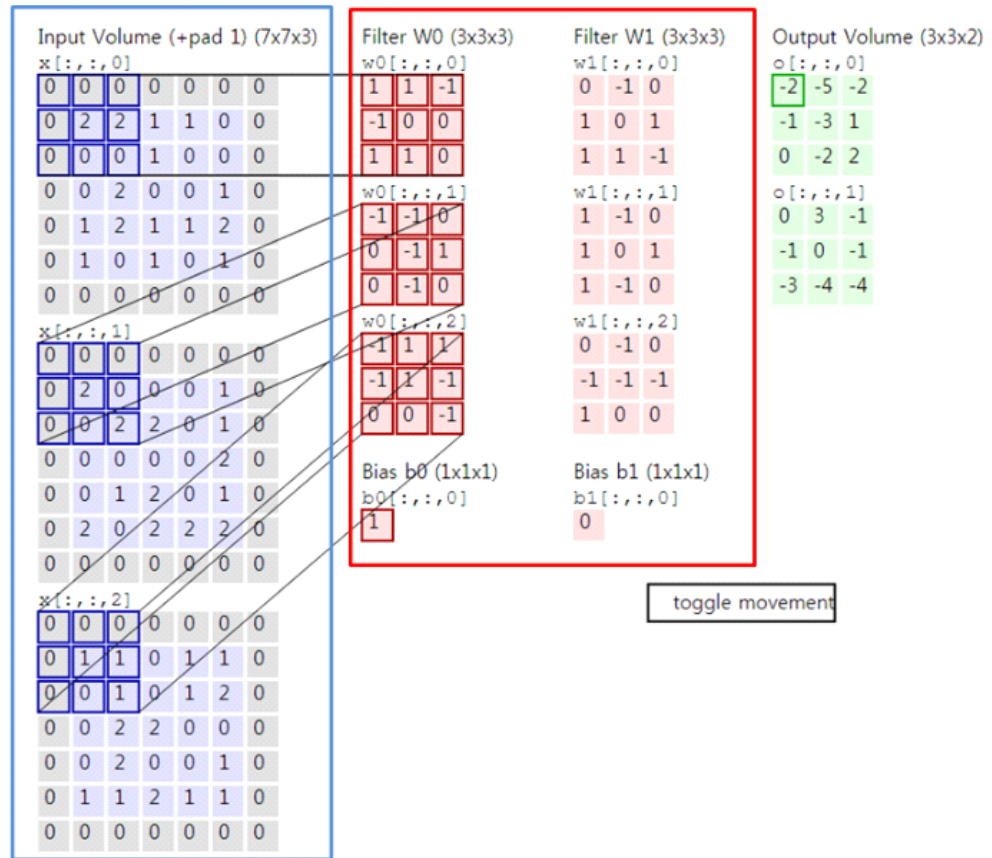
****결과**

```

[[[ 0 -2 -1  0 -1]
  [ 0 5  4  1  2]
  [ 1 2  2  3  2]
  [-1 2  1 -1  1]
  [-1 1  2  0  2]]
 [[-2 -2 -2  1 -2]
  [ 0 -2 -2  1 -4]
  [ 0 -3 -6  0 -4]
  [-1  1 -4 -1 -5]
  [-2  1 -3 -2 -3]]
 [[-1  0 -3 -2  0]
  [-1 -1 -2  1  1]
  [-3  1  0  0  1]
  [-1  4 -3 -4  1]
  [ 2  0 -2 -1  1]]]
 [[-3 -4 -6 -1 -3]
  [-1  2  0  3 -1]
  [-2  0 -4  3 -1]
  [-3  7 -6 -6 -3]
  [-1  2 -3 -3  0]]

```

| | |
|---------|---|
| 문제 207. | 아래와 같이 입력행렬과 필터행렬과 스트라이드와 패딩을 입력받아 출력 행렬의 shape를 출력하는 함수를 구하시오. |
|---------|---|



```
def out(x,f, s, p):

    FH = f.shape[-2]
    FW = f.shape[-1]
    H = x.shape[-2]
    W = x.shape[-1]

    OH = int(((H+2*p-FH)/s)+1)
    OW = int(((W+2*p-FW)/s)+1)

    pad=np.pad(x,pad_width=((0,0),(p,p),(p,p)),
               mode='constant', constant_values=0)
    rst=[]
    for d in range(len(pad)):
        for i in range(int((len(pad[0])-len(f))/s+1)):
            for j in range(int((len(pad[0])-len(f))/s+1)):
                r=pad[d,i:i+len(Filter),j:j+len(Filter)]*Filter[d,:,:]
                rst.append(np.sum(r))
    print('###shape : OH :',OH, ' OW :',OW)
    rst=np.array(rst)
    return rst.reshape(int(len(rst)/(OH*OW)),OH,OW)

data = np.array(
[
    [[2, 2, 1, 1, 0],
     [0, 0, 1, 0, 0],
     [0, 2, 0, 0, 1],
     [1, 2, 1, 1, 1], # ---> Red
     [1, 0, 1, 0, 1]],
    [[2, 0, 0, 0, 1],
     [0, 2, 2, 0, 1],
     [0, 0, 0, 0, 2], # ----> Green
     [0, 0, 0, 0, 2],
     [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], # ----> Blue
[0, 2, 2, 0, 0],
[0, 2, 0, 0, 1],
[1, 1, 2, 1, 1]]
])

```

```

Filter=np.array([[[[1,1,-1,-1,0,0,1,1,0],
[-1,-1,0,0,-1,1,0,-1,0],
[-1,1,1,-1,1,-1,0,0,-1]]]).reshape(3,3,3) # 개수, 행, 열

```

```

out(data,Filter,1,1)

```

****결과**

```

###shape : OH : 5 OW : 5
array([[[ 0, -2, -1, 0, -1],
[ 0, 5, 4, 1, 2],
[ 1, 2, 2, 3, 2],
[-1, 2, 1, -1, 1],
[-1, 1, 2, 0, 2]],
[[-2, -2, -2, 1, -2],
[ 0, -2, -2, 1, -4],
[ 0, -3, -6, 0, -4],
[-1, 1, -4, -1, -5],
[-2, 1, -3, -2, -3]],
[[-1, 0, -3, -2, 0],
[-1, -1, -2, 1, 1],
[-3, 1, 0, 0, 1],
[-1, 4, -3, -4, 1],
[ 2, 0, -2, -1, 1]]])

```

| | |
|----------------|--|
| 문제 208. | 설현 사진 50장과 아이린 사진 50장, 총 100장의 사진을 신경망에 입력해서 설현과 아이린을 구분(분류) 하는 신경망을 만든다고 할 때 RGB필터를 30개 사용하면 Feature map은 총 몇개 일까 ? |
|----------------|--|

100장 ----> 3000장

| | |
|----------------|--|
| 문제 209. | 칠판에 나온 아이린 사진 한 장의 3차원 행렬을 만드시오. (RGB 7x7 행렬 1장) |
|----------------|--|

```

import numpy as np

```

```

x1 = np.random.rand(1,3,7,7)
print(x1)
print(x1.shape)

```

****결과**

```

[[[[0.04146522 0.75111956 0.3096085 0.70458906 0.78363936 0.52925284
0.03027472]
[0.84720142 0.9510707 0.71125988 0.42433048 0.75146977 0.92061094
0.39863146]

```

```

[0.84406455 0.76879374 0.6406376 0.87873444 0.97485125 0.98397857
0.7300184 ]
[0.10790605 0.85568982 0.50554661 0.60595911 0.55054506 0.7559521
0.53092377]
[0.46822921 0.18695024 0.68807195 0.04881204 0.55263676 0.87514535
0.36401664]
[0.16108935 0.34637151 0.69821563 0.92129002 0.9331873 0.25258835
0.27808362]
[0.76620471 0.92373867 0.48886042 0.89476054 0.26297989 0.62433865
0.76431686]]
[[[0.20595574 0.10288318 0.36168794 0.81557768 0.90056976 0.2643665
0.08590275]
[0.84876066 0.13131874 0.15082708 0.77715371 0.42781956 0.44429663
0.16916134]
[0.38209403 0.81437037 0.71543096 0.20895893 0.26314861 0.73417797
0.48657193]
[0.45288346 0.68045069 0.40808002 0.74165353 0.64106329 0.462943
0.64699237]
[0.87769469 0.87687895 0.15368963 0.8452533 0.9300904 0.29249455
0.4777687 ]
[0.26382278 0.51740736 0.30171081 0.94858638 0.24544676 0.66503829
0.51625294]
[0.66949584 0.49436196 0.80933866 0.41411674 0.04888242 0.503647
0.85257683]]
[[[0.13118409 0.34162458 0.26272157 0.5251492 0.97501604 0.54128155
0.78062057]
[0.41335574 0.92593055 0.65730433 0.49590699 0.77932026 0.71341426
0.83236066]
[0.10497102 0.19541296 0.88130376 0.04252038 0.32205706 0.40305427
0.11580042]
[0.6650085 0.79719574 0.89253871 0.64035887 0.30685577 0.08802892
0.54560454]
[0.88878912 0.33629819 0.93316654 0.97246735 0.61050836 0.86916122
0.91330019]
[0.59210703 0.29721845 0.77380886 0.76259301 0.64428312 0.33293676
0.43756581]
[0.83783129 0.55232062 0.40546976 0.91759246 0.15869097 0.04373689
0.66651265]]]]
(1, 3, 7, 7)

```

문제 210. im2col 함수를 이용해서 아래의 4차원을 2차원 행렬로 변경 하시오.
(필터는 5x5의 RGB 행렬을 사용함)

```
import numpy as np
```

```
def im2col(input_data, filter_h, filter_w, stride=1, pad=0):
    """다수의 이미지를 입력받아 2차원 배열로 변환한다(평탄화).

    Parameters
    -----
    input_data : 4차원 배열 형태의 입력 데이터(이미지 수, 채널 수, 높이, 너비)
    filter_h : 필터의 높이
    filter_w : 필터의 너비
    stride : 스트라이드
    pad : 패딩

    Returns
    -----
    col : 2차원 배열
    """
    N, C, H, W = input_data.shape
    out_h = (H + 2 * pad - filter_h) // stride + 1
    out_w = (W + 2 * pad - filter_w) // stride + 1

    img = np.pad(input_data, [(0, 0), (0, 0), (pad, pad), (pad, pad)], 'constant')
    col = np.zeros((N, C, filter_h, filter_w, out_h, out_w))

    for y in range(filter_h):
        y_max = y + stride * out_h
        for x in range(filter_w):
            x_max = x + stride * out_w
            col[:, :, y, x, :, :] = img[:, :, y:y_max:stride, x:x_max:stride]

    col = col.transpose(0, 4, 5, 1, 2, 3).reshape(N * out_h * out_w, -1)
    return col

x1 = np.random.rand(1,3,7,7) # 입력데이터 (원본) 4차원
col = im2col(x1,5,5,stride=1,pad=0) # 입력데이터를 2차원 행렬로 변환
print(col.shape)

**결과
(9, 75) # 한장만 im2col 했을 때
```

| | |
|----------------|-----------------------|
| 문제 211. | 이미지 10장을 랜덤으로 생성 하시오. |
|----------------|-----------------------|

```
x10 = np.random.rand(10,3,7,7)
print(x10.shape)
```

****결과**
(10, 3, 7, 7)

| | |
|----------------|--|
| 문제 212. | 아이린 사진 10장을 im2col 함수에 넣어서 2차원 행렬로 변환 시키시오. (필터는 5x5의 RGB 행렬을 사용함) --> 입력 데이터(4차원)을 2차원 데이터로 변형 |
|----------------|--|

```
x10 = np.random.rand(10,3,7,7)
print(x10.shape)
```

```
col = im2col(x10,5,5,stride=1,pad=0)
print(col.shape)
```

****결과**

```
(10, 3, 7, 7)
(90, 75)
```

문제 213. mlist 데이터 100장을 im2col 함수에 넣었을 때, 나오는 출력 예상하시오.
(필터의 크기 : 5X5 RGB 채널, mlist 이미지 크기 : 28x28 흑백채널)

```
x10 = np.random.rand(10,3,7,7)
print(x10.shape)
col = im2col(x10,5,5,stride=1,pad=0)
print(col.shape)
```

****결과**

```
(100, 1, 28, 28)
(57600, 25)
```

문제 214. 아래의 filter를 생성하고 shape를 확인하고 전치 시키시오.

```
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) # (5,5,3)
```

****결과**

```
(5, 5, 3)
```

문제 215. Filter (3,5,5) 행렬을 (3,25) 행렬로 변경 하시오.

```
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) # (5,5,3)
a = Filter.T.reshape(3,-1)
print(a.shape)
```

****결과**

```
(5, 5, 3)
(3, 25)
```

문제 216. 아래의 4차원 행렬의 filter를 numpy의 random을 이용해서 만드시오.

```
f = np.random.rand(10,3,5,5)
print(f.shape)
```

****결과**

(10, 3, 5, 5)

문제 217. 아래의 4차원 행렬을 3차원으로 변경 하시오.
(10,3,5,5) [4차원] -----> (10,3,25) [3차원]

```
f = np.random.rand(10,3,5,5)
print(f.shape)
f = f.reshape(10,3,-1)
print(f.shape)
```

****결과**

(10, 3, 5, 5)

(10, 3, 25)

문제 218. 아래의 3차원 행렬을 2차원 행렬로 변경 하시오.

```
f = np.random.rand(10,3,5,5)
print(f.shape)
f = f.reshape(10,3,-1)
print(f.shape)
f = f.reshape(10,-1)
print(f.shape)
```

****결과**

(10, 3, 5, 5)

(10, 3, 25)

(10, 75)

문제 219. (10,75) ---> (75, 10)으로 변경 하시오.

```
f = np.random.rand(10,3,5,5)
print(f.shape)
f = f.reshape(10,3,-1)
print(f.shape)
f = f.reshape(10,-1)
print(f.shape)
print(f.T.shape)
```

****결과**

(10, 3, 5, 5)

(10, 3, 25)

(10, 75)

(75, 10)

문제 220. 책 246페이지에 나오는 Convolution 클래스를 생성 하시오.

```
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
        out = np.dot(col, col_W) + self.b

        out = out.reshape(N, out_h, out_w, -1).transpose(0,3,1,2)

        return out
```

문제 221. 위에서 만든 Convolution 클래스를 객체화 시켜서 칠판에 나온 convolution층을 구현 하시오.

```
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
        out = np.dot(col, col_W) + self.b

        out = out.reshape(N, out_h, out_w, -1).transpose(0,3,1,2) # 입력행렬의 모양으로 바꿔주기 위해서

        return out

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)

**결과
f.shape = (10, 10, 3, 3)
```

문제 222. 아래의 이미지를 손으로 최대풀링 하시오.

| | | | |
|----|----|---|----|
| 21 | 8 | 8 | 12 |
| 12 | 19 | 9 | 7 |
| 8 | 10 | 4 | 3 |
| 18 | 12 | 9 | 10 |

| | | | |
|----|----|---|----|
| 21 | 8 | 8 | 12 |
| 12 | 19 | 9 | 7 |
| 8 | 10 | 4 | 3 |
| 18 | 12 | 9 | 10 |

--->

| | |
|----|----|
| 21 | 12 |
| 18 | 10 |

문제 223. max_pooling 함수를 이용해서 4x4 행렬을 2x2 행렬로 변경 하시오.

```
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
```

```
x = np.array([[21,8,8,12],
              [12,19,9,7],
              [8,10,4,3],
              [18,12,9,10]])
```

```
print(x,end='\n\n')
```

```
max_p = max_pooling(x)
print(max_p)
```

****결과**

```
[[21  8  8 12]
 [12 19  9  7]
 [ 8 10  4  3]
 [18 12  9 10]]
[[21 12]
 [18 10]]
```

문제 224. 책 249 페이지의 Pooling 클래스 생성 하시오.

```
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)

    col = im2col(x, self.pool_h, self.pool_w, self.stride, self.pad)
    col = col.reshape(-1, self.pool_h * self.pool_w)

    out = np.max(col,axis=1)

    out = out.reshape(N, out_h, out_w, C).transpose(0,3,1,2)

    return out

```

문제 225. mnist (28 x 28) 데이터가 convolution 층을 통과했을 때 출력이미지의 사이즈를 알아내시오.
(필터 사이즈 : 5x5 , stride : 1, padding : 0)

$$OH = \frac{H + 2P - FH}{S} + 1$$

$$OW = \frac{W + 2P - FW}{S} + 1$$

----> P(패딩) = (OH -1) * S -H +FH / 2

$$oh = (28 + 2*0 - 5) / 1 + 1 = 24$$

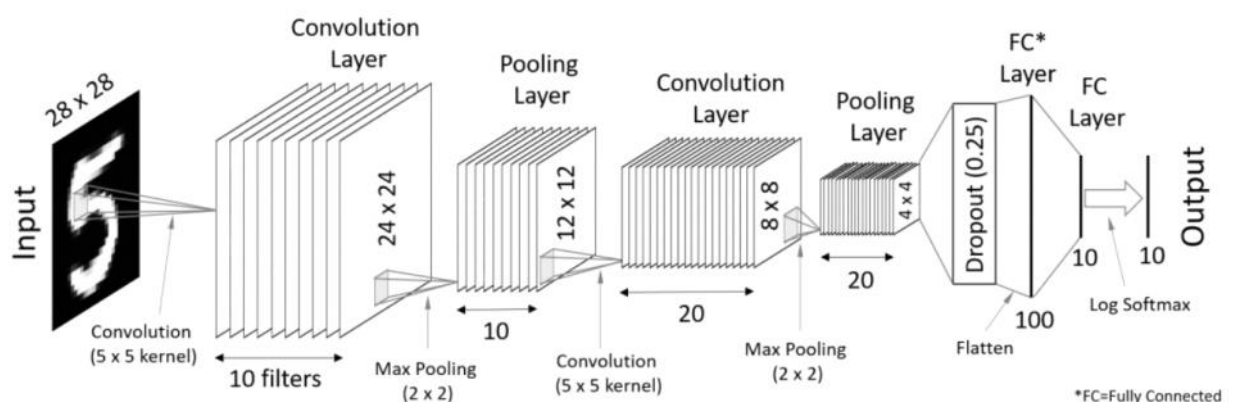
$$ow = (28 + 2*0 - 5) / 1 + 1 = 24$$

$$oh \times ow = 24 \times 24$$

문제 226. 입력 이미지 (24x24) 행렬이 풀링층을 통과했을 때 출력되는 이미지의 크기가 어떻게 되는가?
(pool_h : 2, stride : 2)

11.5인데 int(11.5) 이므로 12가 된다. # 파이썬을 짰수로 반올림

문제 227. mnist 데이터를 cnn으로 구현했을때의 신경망을 그림으로 확인 하시오.



문제 228. 칠판에 그린 CNN 구현코드를 구현 하시오.

```
# 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

```

```

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)

```

```

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 신경망에 가중치 초기화 선정인 Xavier를 적용해서 테스트 하시오. |
|---------|---|

- 1. Xavier 적용

```
(60000, 1, 28, 28)
```

```
train acc, test acc | 0.9863833333333333, 0.9833
```

```
(60000, 1, 28, 28)
```

```
train acc, test acc | 0.9877166666666667, 0.9843
```

| | |
|---------|------------------------------------|
| 문제 230. | 위의 cnn을 이용한 3층 신경망에 배치정규화를 적용 하시오. |
|---------|------------------------------------|

```
# 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'])

```

층이 깊어질 수록 배치정규화의 필요성이 커진다. (가중치 값들이 퍼트려져 있는 것을 강제화 하기위해)

| | |
|---------|--|
| 문제 231. | 위의 cnn을 이용한 3층 신경망에 합성곱층과 pooling 층을 하나 더 추가하시오. |
|---------|--|

conv --> relu --> pooling --> conv --> relu --> pooling --> affine1 --> relu --> affine2 --> softmax