

21.03.04

2021년 3월 4일 목요일 오전 9:49

딥러닝

머신러닝과 딥러닝의 차이

머신러닝으로 기계학습을 하려면 정형화된 데이터가 있어야 한다.

표 형태의 정형화 된 데이터를 만들려고 판다스를 이용했다.

이런 정형화 된 데이터를 만드는 것이 상당한 노동력이 들어간다.

그래서 딥러닝은 이미지를 신경에 보여주면 스스로 학습해서 그 이미지를 신경망이 알아볼 수 있는 상태가 된다.

딥러닝 기술이 현업에서 사용되는 예

1. 인천공항에서 컨테이너 검색대에 물건을 올리면 CNN으로 판별하여 위반되는 물품이 있는지 검사
2. 의료쪽에서 X-ray 사진과 의료영상 사진의 질병여부를 컴퓨터가 판별
3. 물품(의류)의 불량품을 판별하는 신경망 개발 + 인터페이스
4. 주가 예측 신경망
5. 외국어 번역하는 신경망
6. 인공지능 변호사
7. 음악 작곡 신경망(Gan)

넘파이(numpy) 배열 생성하기

numpy란 python언어에서 기본적으로 지원하지 않는 배열(array) 혹은 행렬(matrix)의 계산을 쉽게 해주는 라이브러리이다.

머신러닝에서 많이 사용하는 선형대수학에 관련된 수식들을 python에서 쉽게 프로그래밍 할 수 있게 해준다.

문제1. 아래의 행렬을 numpy로 만드시오.

```
array([[1, 2],  
       [3, 4]])
```

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

문제2. 위의 a 행렬의 각 요소에 숫자 5를 더하시오.

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

문제3. 아래의 두 행의 합을 구하시오.

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

numpy의 유용한 기능 브로드캐스트(broadcast)

넘파이에서는 형상이 다른 배열끼리도 계산할 수 있다.

아래의 경우처럼 숫자 10이 2×2 행렬로 확대된 후 연산이 이루어지는 것을 브로드 캐스트라고 한다.

문제4. 아래의 브로드 캐스트 기능을 numpy로 구현하시오.

$$\begin{array}{|c|c|} \hline 1 & 2 \\ \hline 3 & 4 \\ \hline \end{array} * \begin{array}{|c|} \hline 10 \\ \hline \end{array} = \begin{array}{|c|c|} \hline 1 & 2 \\ \hline 3 & 4 \\ \hline \end{array} * \begin{array}{|c|c|} \hline 10 & 10 \\ \hline 10 & 10 \\ \hline \end{array} = \begin{array}{|c|c|} \hline 10 & 20 \\ \hline 30 & 40 \\ \hline \end{array}$$

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

문제5. 아래의 그림 1-2도 브로드캐스트 되는지 테스트 하시오.

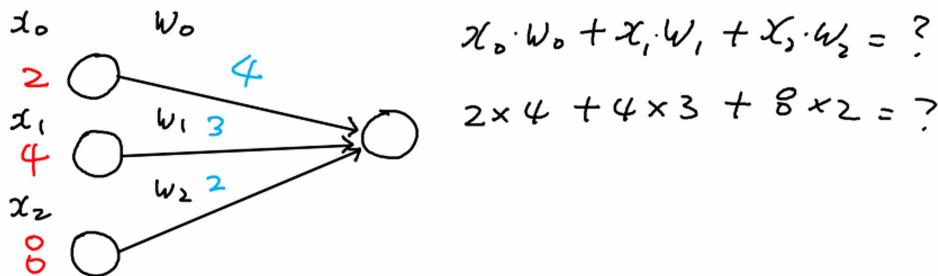
$$\begin{array}{|c|c|} \hline 1 & 2 \\ \hline 3 & 4 \\ \hline \end{array} * \begin{array}{|c|c|} \hline 10 & 20 \\ \hline \end{array} = \begin{array}{|c|c|} \hline 1 & 2 \\ \hline 3 & 4 \\ \hline \end{array} * \begin{array}{|c|c|} \hline 10 & 20 \\ \hline 10 & 20 \\ \hline \end{array} = \begin{array}{|c|c|} \hline 10 & 40 \\ \hline 30 & 80 \\ \hline \end{array}$$

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

```
b = np.array([10,20])
```

```
a*b
```

문제6. 아래의 신경망을 numpy 행렬로 구현하시오.

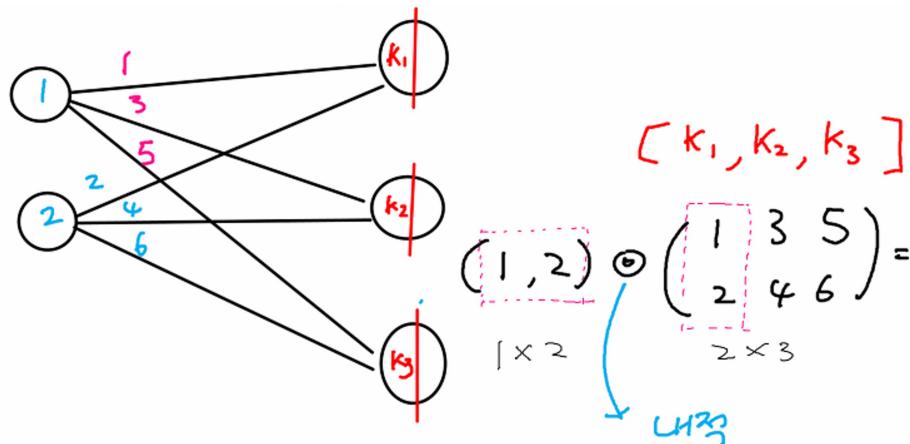


```
import numpy as np
```

```
x = np.array([2,4,8])  
w = np.array([4,3,2])
```

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

문제7. 아래의 신경망을 numpy 행렬로 구현하시오.

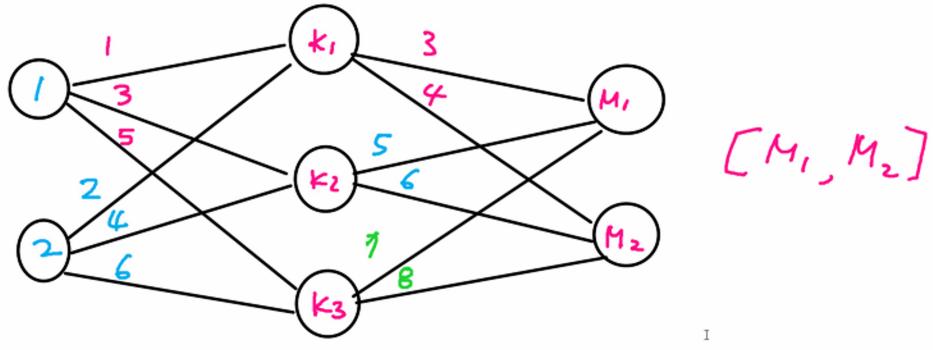


```
import numpy as np
```

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

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

문제8. 아래의 신경망을 numpy로 구현하시오.



```
import numpy as np
```

```
x = np.array([1,2]) # 0층 (입력층)
w1 = np.array([[1,3,5], [2,4,6]]) # 가중치 행렬1
k = np.dot(x, w1) # 1층 (은닉1층)
w2 = np.array([[3,4], [5,6], [7,8]]) # 가중치 행렬2
m = np.dot(k, w2) # 2층 (출력층)
```

m

넘파이(Numpy) N차원 배열

넘파이는 1차원 배열(1줄로 늘어선 배열) 뿐만 아니라 다차원 배열로도 작성할 수 있다.

ex)

```
import numpy as np
```

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

```
x.shape # (2, 2)
```

문제9. 아래와 같이 5x5 행렬을 생성하시오.

```
array([[1, 2, 3, 4, 5],
       [2, 4, 3, 2, 4],
       [3, 1, 4, 2, 1],
       [2, 7, 3, 5, 4],
       [1, 5, 6, 3, 1]])
```

```
import numpy as np
```

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

```
print (x)
x.shape
```

넘파이 배열은 N차원 배열을 작성할 수 있다.

1차원, 2차원, 3차원 배열처럼 원하는 차수의 배열을 만들 수 있다.

수학에서는 1차원 배열을 벡터(vector)라고 하고 2차원 배열을 행렬(matrix)라고 부른다.

또 벡터와 행렬을 일반화 한것을 텐서(tensor)라고 한다.

tensorflow : 행렬이 계산되면서 훌러간다.

1. 코드가 간결하다.
2. 신경망 구현에 필요한 모든 함수들이 내장되어 있다.
3. 속도가 아주 빠르다.
4. GPU를 사용할 수 있다.

문제10. 아래의 행렬의 내적을 구현하시오.

$\begin{bmatrix} 3 & 4 & 2 \end{bmatrix}$

$\begin{bmatrix} 4 & 1 & 3 \end{bmatrix}$

$\begin{bmatrix} 1 & 5 \end{bmatrix}$

$\begin{bmatrix} 2 & 3 \end{bmatrix}$

$\begin{bmatrix} 4 & 1 \end{bmatrix}$

```
import numpy as np
```

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

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

넘파이 원소 접근

numpy 배열안에 요소들에 대한 접근은 numpy를 이용하지 않았을 때 보다 훨씬 간단하게 구현할수 있다.

문제11. 아래의 리스트를 만들고 아래의 리스트에서 15이상인 숫자들만 출력하시오.

$[51, 55, 14, 19, 0, 4]$

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

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

문제12. 아래의 행렬식을 만들고 아래의 행렬의 요소에서 15이상인 것만 출력하시오.

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

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

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

문제13. 위의 예제를 numpy를 이용해서 구현하시오.

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

numpy를 이용하면 코딩을 어렵게 하지 않아도 된다. 쉽고 빠르게 원하는 것을 구현할 수 있다.

문제14. 아래의 행렬을 만들고 아래의 행렬에서 3 이상인 것만 출력하시오.

```
[[5.1 3.5 1.4 0.2]  
 [4.9 3. 1.4 0.2]  
 [4.7 3.2 1.3 0.2]  
 [4.6 3.1 1.5 0.2]]
```

```
import numpy as np  
  
a = np.array ([[5.1,3.5,1.4,0.2], [4.9,3.,1.4,0.2], [4.7,3.2,1.3,0.2], [4.6,3.1,1.5,0.2]])  
  
print (a[ a >= 3])
```

Matplotlib로 그래프 그리기

딥러닝 실험에서는 그래프 그리기와 데이터 시각화가 중요하다.

matplotlib는 그래프를 그리기 위한 라이브러리이다.

matplotlib를 이용하면 그래프 그리기가 쉬워진다.

신경망에서 사용하는 그래프는 주로 라인 그래프 이다.

정확도가 점점 올라가는지, 에러(오차)가 점점 떨어지는지 확인할 때 유용하다.

ex)

파이썬으로 산포도 그래프 그리기

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

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

```
plt.scatter(x, y, color='red', s = 80) # s : 점사이즈  
plt.show()
```

문제15. 위의 그래프를 라인 그래프로 그리시오.

```
import numpy as np  
import matplotlib.pyplot as plt  
  
x = np.array([0,1,2,3,4,5,6,7,8,9])  
y = np.array([9,8,7,9,8,3,2,4,3,4])
```

```
plt.plot(x, y, color='red') # scatter : 산포도, plot : 라인  
plt.show()
```

한글 폰트 지정

<https://cafe.daum.net/oracleoracle/SgRM/27>

문제16. 위의 그래프에 제목을 '신경망 오차 그래프' 라고 하시오.

```
import numpy as np  
import matplotlib.pyplot as plt  
from matplotlib import font_manager, rc  
  
font_path = "c:\\\\data\\\\malgun.ttf" #폰트파일의 위치 실습파일안에있음  
font_name = font_manager.FontProperties(fname=font_path).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, color='red') # scatter : 산포도, plot : 라인  
plt.title('신경망 오차 그래프')  
plt.show()
```

★ 1장 내용 정리

1. 딥러닝이란
 - 여러 비선형 변환 기법의 조합을 통해 높은 수준의 추상화를 시도하는 기계학습 알고리즘의 집합
 - 추상화 : 다량의 데이터나 복잡한 자료들 속에서 핵심적인 내용 또는 기능을 요약하는 기법
 - 딥러닝이 부각된 이유
 - o 기존 인공신경망 모델의 문제점인 기울기 소실이 해결되었다.
 - o 강력한 GPU연산을 활용하여 하드웨어 연산속도를 높였다.
 - o 빅데이터의 등장과 SNS의 활용이 증가하여 학습에 필요한 데이터 확보가 가능해졌다.
2. 신경망을 구현할 때 numpy 사용 하는 이유
 - 행렬 연산을 빠르게 할 수 있는 기능이 numpy에 내장되어 있는데 신경망에 데이터를 입력하면 행렬연산이 일어나고 맨 마지막으로 확률이 출력이 된다. 확률이 마지막에 출력되기 위해서는 다차원 행렬 계산을 해야한다.
3. numpy의 주요 기능 중 broadcast
 - 형상이 다른 배열끼리 계산을 스마트하게 할 수 있게 해주는 numpy의 기능
4. 딥러닝 실험을 위해 시각화가 필요한 이유
 - 신경망이 제대로 학습이 되고 있는지 확인하려면 학습과정을 시각화 해봐야 한다.

퍼셉트론(perceptron)

- 인간의 뇌세포 하나를 컴퓨터로 구현한 것
- 1957년 프랑크 로젠틀라트가 퍼셉트론을 고안했다.
- 사람의 뇌의 동작을 전기 스위치 온/오프로 흉내낼 수 있다는 이론을 증명

퍼셉트론

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

특정 자극이 있다면 그 자극이 어느 역치 이상이 되어야 세포가 반응한다.

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

단순한 논리 회로

1. AND 게이트
2. OR 게이트
3. NAND 게이트
4. XOR 게이트

AND 게이트

그림 2-2 참고

문제17. 위의 AND 게이트를 위한 데이터 행렬을 numpy로 구현하시오.

```
import numpy as np

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

print (X.shape)
print (y.shape)
```

문제18. AND 게이트 퍼셉트론 함수를 생성하시오.

```
def AND(x1, x2):
    w1, w2, theta = 0.5, 0.5, 0.7
    tmp = x1*w1 + x2*w2 # 입력값과 가중치 곱의 총합
    if tmp <= theta: # 입력값과 가중치의 총합이 임계치를 넘지 않을 때
        return 0
    elif tmp > theta: # 입력값과 가중치의 총합이 임계치를 넘을 때
        return 1

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

문제19. OR 게이트 퍼셉트론 함수를 생성하시오.

```
def OR(x1, x2):
    w1, w2, theta = 0.5, 0.5, 0.3
    tmp = x1*w1 + x2*w2
    if tmp <= theta:
        return 0
    elif tmp > theta:
```

```
return 1

print (OR(0,0))
print (OR(1,0))
print (OR(0,1))
print (OR(1,1))
```

문제20. 17번 문제에서 만든 X행렬에서 1행 1열에 데이터를 가져오시오.

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

```
import numpy as np

X = np.array([[0,0], [0,1], [1,0], [1,1]])

print (X[0, 0])
```

문제21. 위에서 만든 퍼셉트론 함수 AND를 가지고 입력값 X에 데이터를 입력받아 아래와 같이 결과를 출력하시오.

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

참조 :

```
print (AND(X[0,0],X[0,1]))
print (AND(X[1,0],X[1,1]))
print (AND(X[2,0],X[2,1]))
print (AND(X[3,0],X[3,1]))
```

```
import numpy as np

X = np.array([[0,0], [0,1], [1,0], [1,1]])

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

for i in range (len(X)):
    print (AND(X[i,0],X[i,1]))
```

문제22. NAND 게이트 함수를 생성하고 구현하시오.
(그림 2-3)

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

```

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

print (NAND(0,0))
print (NAND(1,0))
print (NAND(0,1))
print (NAND(1,1))

```

XOR 게이트

eXclusive OR 게이트

AND, OR, NAND는 단층 신경망으로 구현이 되는데 XOR 게이트는 단층으로는 구현이 되지 않는다.

20년 후에 다층 퍼셉트론으로 비선형 영역을 분류했다.

XOR 게이트 : 그림 2-5

XOR 게이트 그래프 시각화 : 그림 2-7

다층 퍼셉트론으로 XOR 게이트 해결 : 그림 2-12

문제23. 그림 2-11에 나오는 그림으로 입력값 데이터를 받아서 XOR 게이트 결과가 출력되게 하시오.

가중치와 편향의 도입

파라미터 : 가중치, 편향

하이퍼 파라미터 : 러닝 레이트, 가중치 감소(decay), 층수와 뉴런의 갯수

가중치 : 입력신호가 결과에 주는 영향력(중요도)를 조절하는 매개변수

편향 : 뉴런이 얼마나 쉽게 활성화(결과로 1을 출력) 하느냐를 조정하는 매개변수

ex)

OR 게이트처럼 입력신호가 x_1 과 x_2 값을 받는 경우에 편향(x_0)이 없다면 target을 분류하는 직선은 무조건 원점을 통과해야만 하기 때문에 제대로 분류할 수 없게 된다.

문제24. 이번에는 편향을 넣어서 AND게이트 함수를 생성하시오.

```
import numpy as np
```

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

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

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

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

문제25. OR게이트 함수를 생성하시오.

```
import numpy as np

def OR(x1, x2):
    x = np.array([x1,x2])
    w = np.array([0.5,0.5]) # 실제로는 w와 b를 학습해서 알아낸다.
    b = -0.4
    tmp = np.sum(w*x) + b

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

print (OR(0,0))
print (OR(1,0))
print (OR(0,1))
print (OR(1,1))
```

문제26. NAND게이트 함수를 만들고 XOR 게이트 함수를 만들어서 아래의 결과가 출력되게 하시오.

```
import numpy as np
```

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

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

def XOR(x1, x2):
    s1 = NAND(x1, x2)
```

```
s2 = OR(x1, x2)
y = AND(s1, s2)
return y
```

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

21.03.05

2021년 3월 5일 금요일 오전 9:39

복습

- 신경망 내부에서 행렬 연산을 편하게 하기 위해 사용되는 모듈인 numpy 모듈
- 2장의 퍼셉트론을 구현하기 위해서 함수 4개를 생성 (AND, OR, NAND, XOR)
 - 단층 신경망 : AND, OR, NAND
 - 다층 신경망 : XOR

기계가 직접 학습해서 가중치(w)와 바이어스(b)를 알아내는 방법

ex1)

아래의 두 행렬을 생성하시오.

x

```
[[ -1  0  0]
 [-1  1  0]
 [-1  0  1]
 [-1  1  1]]
```

w

```
[[ 0.3]
 [ 0.4]
 [ 0.1]]
```

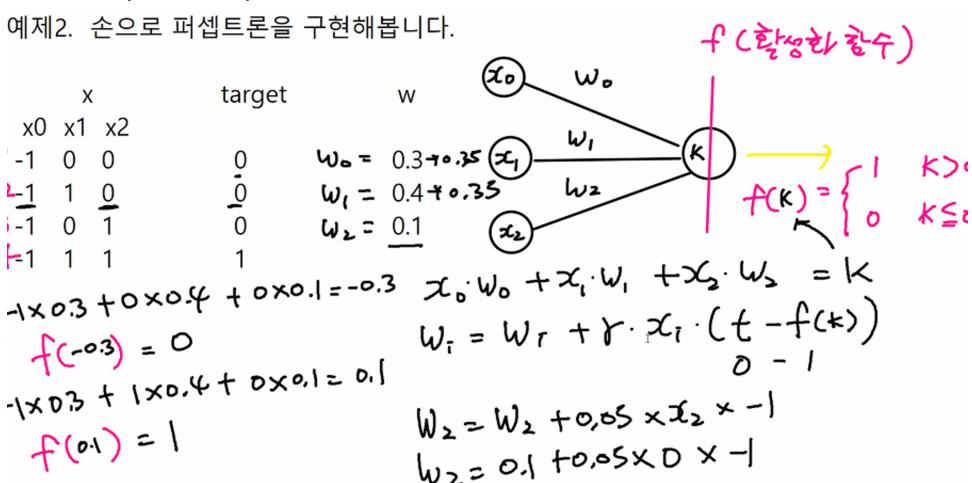
```
import numpy as np
```

```
x = np.array([[-1,0,0], [-1,1,0], [-1,0,1], [-1,1,1]])
w = np.array([[0.3], [0.4], [0.1]])
```

ex2)

손으로 퍼셉트론 구현

예제2. 손으로 퍼셉트론을 구현해봅니다.



문제27. 문제24번의 AND게이트 함수를 가져와서 위에서 구한 가중치와 바이어스를 대입해서 TARGET(정답)을 잘 예측하는지 테스트 하시오.

```
import numpy as np
```

```
def AND(x1, x2):
    x = np.array([x1,x2])
    w = np.array([0.35,0.1])
    b = -0.35
    tmp = np.sum(w*x) + b

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

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

and 게이트 데이터 분류

■ and 게이트 데이터는 어떻게 분류가 되는 것인가?

$$b = -0.35, w_1 = 0.35, w_2 = 0.1$$

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

$$-0.35 + 0.35 \cdot x_1 + 0.1 \cdot x_2 = 0$$

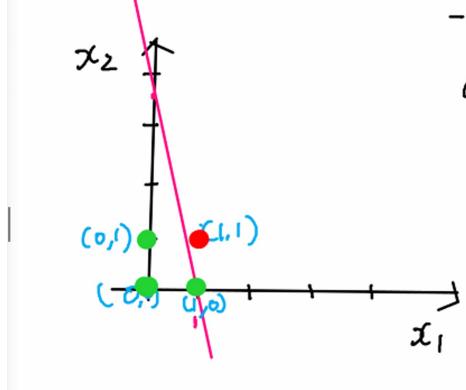
$$0.1 \cdot x_2 = -0.35 \cdot x_1 + 0.35$$

$$10 \cdot x_2 = -35 \cdot x_1 + 35$$

$$2 \cdot x_2 = -7 \cdot x_1 + 7$$

$$x_2 = -\frac{7}{2} \cdot x_1 + \frac{7}{2}$$

$$x_2 = -3.5 \cdot x_1 + 3.5$$



ex3)

위의 ex1에서 w를 전치 시키고 x와 w.T를 브로드캐스트에 의해서 곱한다.

x	w	$x_0 \cdot w_0 + x_1 \cdot w_1 + x_2 \cdot w_2 = ?$
-1 0 0	0.3 0.4 0.1	$-1 \times 0.3 + 0 \times 0.4 + 0 \times 0.1 = ?$
-1 1 0	0.3 0.4 0.1	$-1 \times 0.3 + 1 \times 0.4 + 0 \times 0.1 = ?$
-1 0 1	0.3 0.4 0.1	$-1 \times 0.3 + 0 \times 0.4 + 1 \times 0.1 = ?$
-1 1 1	0.3 0.4 0.1	$-1 \times 0.3 + 1 \times 0.4 + 1 \times 0.1 = ?$

방법 1

```
import numpy as np
```

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

```
w = np.array([[0.3], [0.4], [0.1]])
target = np.array([[0], [0], [0], [1]])

print (w.T)
print (np.sum(x[0] * w.T))
print (np.sum(x[1] * w.T))
print (np.sum(x[2] * w.T))
print (np.sum(x[3] * w.T))
```

방법 2

```
import numpy as np
```

```
x = np.array([[-1,0,0], [-1,1,0], [-1,0,1], [-1,1,1]])
w = np.array([[0.3], [0.4], [0.1]])
target = np.array([[0], [0], [0], [1]])
```

```
for i in range(len(x)):
    print (np.sum(x[i] * w.T))
```

ex4)

입력값과 가중치의 곱의 합을 계산하는 predict 함수를 만드시오.

```
import numpy as np
```

```
x = np.array([[-1,0,0], [-1,1,0], [-1,0,1], [-1,1,1]])
w = np.array([[0.3], [0.4], [0.1]])
target = np.array([[0], [0], [0], [1]])
```

```
def predict(x, w):
    a = np.sum(x * w.T)
    return a
```

```
for i in range(len(x)):
    print (predict(x[i], w))
```

ex5)

입력되는 값이 0보다 작거나 같으면 0을 출력하고, 입력되는 값이 0보다 크면 1을 출력하는 step 함수를 생성하시오.

```
def step_function(x):
    if x <= 0:
        return 0
    else:
        return 1

print (step_function(0.3)) # 1
print (step_function(-2)) # 0
```

ex6)

위에서 출력한 k 값을 step_function 함수에 넣고 값을 출력하시오.

```
import numpy as np
```

```

x = np.array([[-1,0,0], [-1,1,0], [-1,0,1], [-1,1,1]])
w = np.array([[0.3], [0.4], [0.1]])
target = np.array([[0], [0], [0], [1]])

def step_function(x):
    if x <= 0:
        return 0
    else:
        return 1

def predict(x, w):
    a = np.sum(x * w.T)
    return step_function(a)

for i in range(len(x)):
    print (predict(x[i], w))

```

ex7)

위에서 만든 예측값과 target값과의 차이를 구하시오.

```

import numpy as np

x = np.array([[-1,0,0], [-1,1,0], [-1,0,1], [-1,1,1]])
w = np.array([[0.3], [0.4], [0.1]])
target = np.array([[0], [0], [0], [1]])

def step_function(x):
    if x <= 0:
        return 0
    else:
        return 1

def predict(x, w):
    a = np.sum(x * w.T)
    return step_function(a)

for i in range(len(x)):
    cost = target[i] - predict(x[i], w)
    print (cost)

```

ex8)

위의 cost가 0이 아니면 cost를 출력하시오.

```

import numpy as np

x = np.array([[-1,0,0], [-1,1,0], [-1,0,1], [-1,1,1]])
w = np.array([[0.3], [0.4], [0.1]])
target = np.array([[0], [0], [0], [1]])

def step_function(x):
    if x <= 0:
        return 0
    else:
        return 1

```

```

    return 1

def predict(x, w):
    a = np.sum(x * w.T)
    return step_function(a)

for i in range(len(x)):
    cost = target[i] - predict(x[i], w)
    if cost != 0:
        print (cost)

```

ex9)
cost가 0이 아닐때는 가중치가 갱신되도록 하시오.

```

import numpy as np

x = np.array([[-1,0,0], [-1,1,0], [-1,0,1], [-1,1,1]])
w = np.array([[0.3], [0.4], [0.1]])
target = np.array([[0], [0], [0], [1]])

def step_function(x):
    if x <= 0:
        return 0
    else:
        return 1

def predict(x, w):
    a = np.sum(x * w.T)
    return step_function(a)

for i in range(len(x)):
    cost = target[i] - predict(x[i], w)
    if cost != 0:
        w = w + np.array([0.05 * x[i] * cost]).T
    print (w)

```

문제28. 위의 코드를 수정해서 맨 마지막에 갱신된 w 값만 출력되게 하시오.

```

import numpy as np

x = np.array([[-1,0,0], [-1,1,0], [-1,0,1], [-1,1,1]])
w = np.array([[0.3], [0.4], [0.1]])
target = np.array([[0], [0], [0], [1]])

def step_function(x):
    if x <= 0:
        return 0
    else:
        return 1

def predict(x, w):
    a = np.sum(x * w.T)

```

```

return step_function(a)

for i in range(len(x)):
    cost = target[i] - predict(x[i], w)
    if cost != 0:
        w = w + np.array([0.05 * x[i] * cost]).T
print (w)

```

문제29. 비용이 0이 되고 더 이상 비용이 발생하지 않았을 때의 가중치가 출력되게 하시오.

```

import numpy as np

x = np.array([[-1,0,0], [-1,1,0], [-1,0,1], [-1,1,1]])
w = np.array([[0.3], [0.4], [0.1]])
target = np.array([[0], [0], [0], [1]])

def step_function(x):
    if x <= 0:
        return 0
    else:
        return 1

def predict(x, w):
    a = np.sum(x * w.T)
    return step_function(a)

for j in range(5):
    sum1 = 0
    for i in range( len(x) ):
        cost = target[i] - predict(x[i], w)
        if cost != 0:
            w = w + np.array( [ 0.05 * x[i] * cost ] ).T
        elif cost==0:
            continue
        sum1 += abs(cost)
    print (sum1, w.T)

```

문제30. 위의 코드에서 비용(cost)이 0이 되는 시점에 break해서 loop문을 종료하고 변경된 가중치가 출력되게 하시오.

```

import numpy as np

x = np.array([[-1,0,0], [-1,1,0], [-1,0,1], [-1,1,1]])
w = np.array([[0.3], [0.4], [0.1]])
target = np.array([[0], [0], [0], [1]])

def step_function(x):
    if x <= 0:
        return 0
    else:
        return 1

```

```

def predict(x, w):
    a = np.sum(x * w.T)
    return step_function(a)

for j in range(5):
    sum1 = 0
    for i in range(len(x)):
        cost = target[i] - predict(x[i], w)
        if cost != 0:
            w = w + np.array([0.05 * x[i] * cost]).T
        elif cost == 0:
            continue
        sum1 += abs(cost)
    if sum1 == 0:
        break
print(sum1, w.T)

```

문제31. 위의 코드에서 `for j in range(5)`를 무한루프문으로 변경하시오.

```

import numpy as np

x = np.array([[-1,0,0], [-1,1,0], [-1,0,1], [-1,1,1]])
w = np.array([[0.3], [0.4], [0.1]])
target = np.array([[0], [0], [0], [1]])

def step_function(x):
    if x <= 0:
        return 0
    else:
        return 1

def predict(x, w):
    a = np.sum(x * w.T)
    return step_function(a)

while True:
    sum1 = 0
    for i in range(len(x)):
        cost = target[i] - predict(x[i], w)
        if cost != 0:
            w = w + np.array([0.05 * x[i] * cost]).T
        elif cost == 0:
            continue
        sum1 += abs(cost)
    if sum1 == 0:
        break
print(sum1, w.T)

```

문제32. 위의 코드를 아래와 같이 실행되도록 함수로 생성하시오.

```
print (perceptron_1957(x, w, targetw))
```

가중치 : [[0.4 0.3 0.1]]

```
import numpy as np
```

```
x = np.array([[-1,0,0], [-1,1,0], [-1,0,1], [-1,1,1]])  
w = np.array([[0.3], [0.4], [0.1]])  
target = np.array([[0], [0], [0], [1]])
```

```
def step_function(x):  
    if x <= 0:  
        return 0  
    else:  
        return 1
```

```
def predict(x, w):  
    a = np.sum(x * w.T)  
    return step_function(a)
```

```
def perceptron_1957(x, w, target):  
    while True:  
        sum1 = 0  
        for i in range( len(x) ):  
            cost = target[i] - predict(x[i], w)  
            if cost != 0:  
                w = w + np.array( [ 0.05 * x[i] * cost ] ).T  
            elif cost==0:  
                continue  
            sum1 += abs(cost)  
        if sum1 == 0:  
            break  
    print ('가중치 :', w.T)
```

```
perceptron_1957(x, w, target)
```

문제33. 위의 perceptron_1957함수에 OR게이트 데이터와 라벨을 입력하고 가중치를 출력하시오.

```
import numpy as np
```

```
x = np.array([[-1,0,0], [-1,1,0], [-1,0,1], [-1,1,1]])  
w = np.array([[0.3], [0.4], [0.1]])  
target = np.array([[0], [1], [1], [1]])
```

```
perceptron_1957(x, w, target)
```

★ 2장 내용 정리

1. 퍼셉트
- 인간의 뇌의 뉴런 세포를 컴퓨터로 흉내낸 것

2. 퍼셉트론의 종류
 - a. 단층 퍼셉트론 : AND 게이트, OR 게이트, NAND 게이트
 - b. 다층 퍼셉트론 : XOR 게이트
3. 인공신경망에서 최종적으로 산출해야 할 값
 - 파라미터 (가중치와 바이어스)

신경망

저자가 만들어온 가중치를 설정해서 필기체 데이터를 인식하는 3층 신경망 생성

신경망안에 들어가는 함수들 소개

1. 활성화 함수
 - a. 계단 함수
 - b. 시그모이드 함수
 - c. 렐루 함수
2. 출력층 함수
 - a. 항등 함수 (회귀분석)
 - b. 소프트맥스 함수 (분류)
3. 오차 함수

계단함수

숫자 0과 1을 리턴하는 함수

입력값 x가 0보다 작거나 같으면 0을 리턴하고,

입력값 x가 0보다 크면 1을 리턴한다.

```
ex1)
def step_function(x):
    if x <= 0:
        return 0
    else:
        return 1
```

`x_data = np.array([-1, 0, 1])` # 신경망의 데이터인 numpy array 형태로 구성

```
print (step_function(x_data)) # 예러
print (step_function(3)) # 1
```

위에서 만든 `step_function`에는 넘파이 배열을 넣을 수 없다.

```
ex2)
numpy 배열을 넣을 수 있도록 step_function 함수를 재생성한다.
```

```

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

x_data = np.array([-1, 0, 1]) # 신경망의 데이터인 numpy array 형태로 구성

print (step_function(x_data))
print (step_function(np.array([3])))

```

신경망에서 훌러가는 모든 데이터는 numpy array형태의 다차원 데이터 이므로 신경망 내에서 쓰여질 활성화 함수도 numpy array형태의 데이터를 받아서 처리할 수 있도록 생성되어져야 한다.

문제34. 위에서 만든 step_function을 이용해서 시각화 하시오.

```

import matplotlib.pyplot as plt
import numpy as np

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

x = np.arange (-5, 5, 0.1) # -5부터 5까지 0.1간격으로 숫자를 출력
y = step_function(x)

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

```

시그모이드 함수

문제35. 시그모이드 함수를 파이썬으로 구현하시오.

```

import numpy as np

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

x_data = np.array([-0.5, 1.0, 2.0])

print (sigmoid(x_data))

```

문제36. 시그모이드 함수를 시각화 하시오.

```

import matplotlib.pyplot as plt
import numpy as np

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

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

```

```
y = sigmoid(x)
```

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

시그모이드 함수의 유래

<https://cafe.daum.net/oracleoracle/Sedp/195>

오즈비율 함수 : 실패할 확률 대비 성공할 확률을 구하는 함수

$$- P / (1 - p)$$

로짓 함수 : 오즈비율 함수에 로그를 사용한 함수

$$- \log(P / (1 - p))$$

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

$$- h(x) = 1 / (1 + \exp(-x))$$

문제37. 오즈비율 함수 그래프를 그리시오.

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

```
def odds_ratio(x):
    return x / (1 - x)
```

```
x = np.arange (0, 1, 0.01)
y = odds_ratio(x)
```

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

문제38. 로짓함수를 그래프로 시각화 하시오.

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

```
def logit(x):
    return np.log(x / (1 - x))
```

```
x = np.arange (0.01, 1, 0.01)
y = logit(x)
```

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

활성화 함수 : 입력 신호의 총합을 출력 신호로 변환하는 함수

1. 시그모이드 함수

- import numpy as np

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

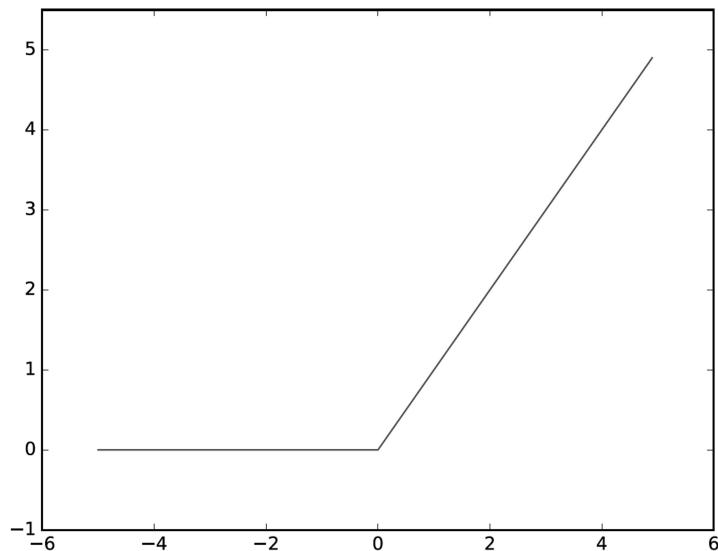
- 시그모이드 함수의 문제점 : 층이 깊어지면 기울기 소실 문제가 발생하여 학습이 잘 되지 않는다.

2. 렐루 함수

ReLU함수 (Rectified Linear Unit)

정류는 전기회로 쪽 용어로, 예를 들어 반파정류회로는 +/- 가 반복되는 교류에서 - 흐름을 차단하는 회로이다.

ReLU함수 그래프는 x가 0이하 일때를 차단하여 아무런 값도 출력하지 않는다.



ex1)

ReLU함수를 생성하시오.

```
import numpy as np
```

```
def relu(x):
    return np.maximum(0, x) # 0과 x 값중에 큰 값을 출력

print (relu(-5)) # 0
```

```
print (relu(0.1)) # 0.1
```

문제39. 위의 relu함수를 그래프로 시각화 하시오.

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

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

x = np.arange(-5, 5, 0.1)
y = relu(x)

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

다차원 배열계산

다차원 배열도 그 기본은 '숫자의 집합'이다. 숫자를 한줄로 늘어선 것이나 정사각형으로 늘어 놓은것이나 전부 다 차원 배열이라고 한다.

ex1)

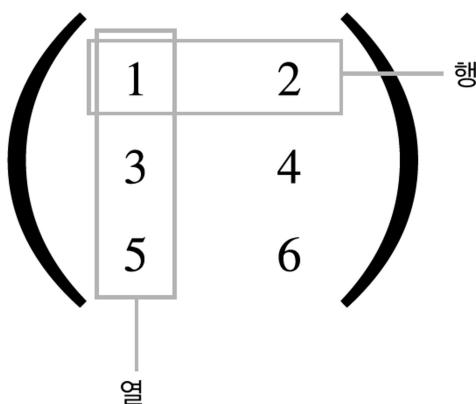
1차원 배열 만들기

```
import numpy as np
```

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

```
print (np.ndim(a)) # 차원을 확인
```

문제40. 아래의 그림의 3×2 행렬을 만들고 차원을 확인하시오.



```
import numpy as np
```

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

```
print (a)
```

```
print (np.ndim(a))
```

문제41. 3차원 배열을 만드시오.

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

문제42. 아래의 2차원 배열에서 숫자 3을 출력하시오.

```
import numpy as np  
a = np.array([[1,2], [3,4], [5, 6]])  
  
print (a[1,0])
```

문제43. 아래의 3차원 배열에서 숫자 3을 출력하시오.

```
import numpy as np  
a = np.array([[[1,2], [3,4]], [[5, 6], [7,8]]])  
  
import numpy as np  
  
a = np.array([[[1,2], [3,4]], [[5, 6], [7,8]]])  
  
print (a[1,0,0])
```

행렬의 내적(행렬 곱)

P.79 그림 3-11 참고

위의 내적을 구현하는 3가지

1. import numpy as np

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

2. import numpy as np

```
x = np.matrix([[1,2], [3,4]])  
y = np.matrix([[5,6], [7,8]])  
  
print (x * y)
```

3. import numpy as np

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

```
print (x @ y)
```

array와 matrix의 기능 차이

array는 다차원으로 나타낼 수 있는데 matrix는 2차원 밖에 나타낼 수 없다.

행렬곱(내적) 시 주의사항

P.81 그림 3-12 참고

다차원 배열을 곱하려면 두 행렬의 대응하는 차원의 원소 수를 일치 시켜야 한다.

신경망 내적

P.82 그림 3-14 참고

■ 신경망 내적 (p82)

$$x_1 = 1, x_2 = 2$$

그림 3-14 를 방정식으로 나타내면 ?

$$\begin{aligned}
 x_1 \cdot 1 + x_2 \cdot 2 &= y^1 \\
 x_1 \cdot 3 + x_2 \cdot 4 &= y^2 \\
 x_1 \cdot 5 + x_2 \cdot 6 &= y^3
 \end{aligned}
 \rightarrow (1, 2) \odot \begin{pmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{pmatrix} = (y_1, y_2, y_3)$$

1x2 ⊙ 2x3

문제44. 아래의 신경망의 출력 y 값을 구하시오.

```
import numpy as np
```

```
x = np.array([4,5,7,2])
w = np.array([[8,21,1], [4,5,9], [6,34,4], [12,2,5]])
y = np.dot (x, w)
```

```
print (y)
```

3층 신경망 구현하기

P.83 그림 3-15 참고

ex1)

입력층에서 은닉1층

```
import numpy as np
```

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

```
x = np.array([1,2])
w1 = np.array([[1,3,5], [2,4,6]])
y = np.dot(x, w1)
y_hat = sigmoid(y)

print (y_hat)
```

ex2)
은닉1층에서 은닉2층

```
import numpy as np

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

# 0층
x = np.array([1,2])

# 1층
w1 = np.array([[1,3,5], [2,4,6]])
a1 = np.dot(x, w1)
a1_hat = sigmoid(a1)

# 2층
w2 = np.array([[3,4], [5,6], [7,8]])
a2 = np.dot(a1_hat, w2)
a2_hat = sigmoid(a2)

print (a2_hat)
```

문제45. 위의 신경망을 출력층까지 구현하시오.
(출력층의 k값 까지)

```
import numpy as np

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

# 0층
x = np.array([1,2])

# 1층
w1 = np.array([[1,3,5], [2,4,6]])
a1 = np.dot(x, w1)
a1_hat = sigmoid(a1)

# 2층
w2 = np.array([[3,4], [5,6], [7,8]])
a2 = np.dot(a1_hat, w2)
a2_hat = sigmoid(a2)
```

```
# 출력층  
w3 = np.array([[4,5], [6,7]])  
y = np.dot(a2_hat, w3)  
  
print (y)
```

출력층 함수

출력층의 함수는 그동안 훈련했던 확률들의 숫자를 취합해서 결론을 내주는 함수

신경망으로 구현하고자 하는 문제

1. 회귀 : 항등함수

ex)

R을 활용한 머신러닝때 콘크리트 강도를 예측하는 머신러닝 모델을 생성

회귀를 통한 콘크리트 강도 예측

독립변수 : 콘크리트 재료

종속변수 : 콘크리트 강도

2. 분류 : 소프트맥스 함수

입력값을 받아서 확률백터로 출력하는 함수

ex)

폐 사진을 통해 정상인지 폐결절인지 분류

수화동작을 글로 출력하는 신경망

출력층 함수인 소프트맥스 함수 생성

P.91 식 3-10 참고

위의 식을 파이썬으로 구현하면 에러가 난다.

지수함수는 쉽게 큰 값을 출력하기 때문에 컴퓨터는 큰값을 출력하게 되면 overflow 문제가 발생한다.

위의 수학식을 컴퓨터로 구현할 수 있도록 개선하면 아래의 식이 된다.

P.93 식 3-11 참고

이를 해결하기 위해서 입력신호중에 최댓값을 이용해서 입력신호 값들의 최대값으로 각각의 요소의 값을 빼준다.

ex1)

```
import numpy as np
```

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

```
print (np.exp(a)) # [inf inf inf]

ex2)
import numpy as np

a = np.array([1010, 1000, 990])
c = np.max(a)
minus = a - c

print (np.exp(minus))
```

위의 코드를 이용해서 소프트맥스 함수 구현하기

```
import numpy as np

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

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

print (softmax(a))
```

문제46. 위의 결과 리스트 요소들을 더하면 1이 출력되는지 확인하시오.

```
import numpy as np

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

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

print (np.sum(softmax(a)))
```

문제47. 위의 결과 리스트 요소들중 어떤게 가장 큰 값인지 인덱스 번호로 출력하시오.

```
import numpy as np

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

```

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

print (np.argmax(softmax(a)))
# np.argmax : numpy리스트의 요소 중 가장 큰 값의 인덱스 번호 출력

```

문제48. 위에서 만든 소프트맥스 함수를 위에서 만든 3층 신경망 출력층에 k값의 활성화 함수로 하시오.

```
import numpy as np
```

```

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

```

```
# 0층
x = np.array([1,2])
```

```
# 1층
w1 = np.array([[1,3,5], [2,4,6]])
a1 = np.dot(x, w1)
a1_hat = sigmoid(a1)
```

```
# 2층
w2 = np.array([[3,4], [5,6], [7,8]])
a2 = np.dot(a1_hat, w2)
a2_hat = sigmoid(a2)
```

```
# 출력층
w3 = np.array([[4,5], [6,7]])
y = np.dot(a2_hat, w3)
y_hat = softmax(y)

print (y_hat)
```

문제49. 위의 3층 신경망 코드에서 w1, w2, w3 가중치를 하나로 모아서 심플한 코드로 작성하시오.

딕셔너리를 활용

파이썬의 자료형 5가지

1. 문자형
2. 숫자형
3. 리스트형
4. 딕셔너리형 : 키와 값으로 구성되어 있는 자료구조
5. 듀플

```
import numpy as np

def sigmoid(x):
    return 1 / (1 + np.exp(-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():
    network = {} # 비어있는 딕셔너리 생성
    network['W1'] = np.array([[1,3,5], [2,4,6]])
    network['W2'] = np.array([[3,4], [5,6], [7,8]])
    network['W3'] = np.array([[4,5], [6,7]])
    return network

# 가중치 값을 불러온다.
network = init_network()
w1, w2, w3 = network['W1'], network['W2'], network['W3']

# 0층
x = np.array([1,2])

# 1층
a1 = np.dot(x, w1)
a1_hat = sigmoid(a1)

# 2층
a2 = np.dot(a1_hat, w2)
a2_hat = sigmoid(a2)

# 출력층
y = np.dot(a2_hat, w3)
y_hat = softmax(y)

print (y_hat)
```

문제50. 위의 sigmoid, softmax, init_network 함수코드를 common.py라는 이름으로 메모장에 저장하고 파이썬의 워킹디렉토리에 저장하시오.

```
import numpy as np
from common import init_network, sigmoid, softmax

# 가중치 값을 불러온다.
network = init_network()
w1, w2, w3 = network['W1'], network['W2'], network['W3']

# 0층
x = np.array([1,2])

# 1층
a1 = np.dot(x, w1)
a1_hat = sigmoid(a1)

# 2층
a2 = np.dot(a1_hat, w2)
a2_hat = sigmoid(a2)

# 출력층
y = np.dot(a2_hat, w3)
y_hat = softmax(y)

print(y_hat)
```

위와 같이 코드를 작성하게 되면 common.py에 계속 코드를 추가하면 된다.

필기체 데이터를 신경망에 로드하기

필요한 파일2가지

1. 필기체 데이터(dataset.zip)
2. 저자가 이미 만들어 놓은 가중치와 바이어스 (sample_weight.pkl)

mnist 데이터를 숫자 0~9 까지의 숫자 이미지로 구성되어있고 훈련 데이터가 6만장, 테스트 데이터가 1만장으로 준비되어 있다.

P.94 그림 3-24 참고

<https://cafe.daum.net/oracleoracle/SgRM/85>

mnist 데이터를 파이썬으로 로드하기

1. 저자가 제공하고 있는 dataset 폴더를 주피터의 워킹 디렉토리로 복사한다.
2. 주피터 노트북에서 아래와 같이 코드를 작성한다.

```
import sys, os
sys.path.append(os.pardir)
from dataset.mnist import load_mnist

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

print(x_train.shape)
```

- load_mnist 함수는 필기체 데이터를 불러오는 함수
- normalize : 이미지의 픽셀값을 0.0~1.0 사이의 값으로 정규화 할지를 정한다.
- flatten : 입력이미지를 1차원 배열로 만들지 정한다.

문제51. 테스트 데이터는 몇장이 있는지 확인하시오.

```
import sys, os
sys.path.append(os.pardir)
from dataset.mnist import load_mnist

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

print(x_train.shape) # (60000, 784)
print(x_test.shape) # (10000, 784)
```

문제52. 훈련 데이터의 첫번째 필기체의 숫자가 무엇인지 정답을 출력해서 알아내시오.

```
import sys, os
sys.path.append(os.pardir)
from dataset.mnist import load_mnist

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

print(t_train[0]) # 5
```

문제53. 훈련 데이터의 첫번째 필기체 데이터를 2차원으로 해서 시각화 하시오.

아나콘다 프롬프트 창에서 Pillow 모듈을 설치한다.

- pip install Pillow

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

```

from PIL import Image

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

img = x_train[0]
print(img.shape) # (784,)
img = img.reshape(28, 28)

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

img_show(img)

```

문제54. mnist 데이터를 flatten 시키지 않고 출력하시오.

```

import sys, os
sys.path.append(os.pardir)
from dataset.mnist import load_mnist
import numpy as np
from PIL import Image # 이미지를 시각화 하기 위한 모듈

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

img = x_train[0]
print(img.shape) # (1, 28, 28)
img = img.reshape(28, 28)

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

img_show(img)

```

문제55. 아이린 사진을 파이썬에서 시각화 하시오.

```

from PIL import Image
import numpy as np
import matplotlib.pyplot as plt

img = Image.open('C:\\\\Users\\\\KJM\\\\아이린.jpg')
img_pixel = np.array(img) # 이미지를 numpy array로 변환
plt.imshow(img_pixel) # 이미지 시각화

print(img_pixel.shape) # (500, 500, 3), (가로, 세로, 색조)

```

문제56. 아이린 사진에서 red 부분의 행렬을 취하고 red부분만 이미지로 시각화 하시오.

```
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt

img = Image.open('C:\\\\Users\\\\KJM\\\\아이린.jpg')
img_pixel = np.array(img)

print (img_pixel[:, :, 0]) # red 부분 행렬만 출력
img_pixel[:, :, 1] = 0 # green 부분을 전부 0으로 변경
img_pixel[:, :, 2] = 0 # blue 부분을 전부 0으로 변경
plt.imshow(img_pixel)
plt.show()
```

문제57. 아이린 사진에서 green 부분만 시각화 하시오.

```
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt

img = Image.open('C:\\\\Users\\\\KJM\\\\아이린.jpg')
img_pixel = np.array(img)

print (img_pixel[:, :, 1]) # green 부분 행렬만 출력
img_pixel[:, :, 0] = 0 # red 부분을 전부 0으로 변경
img_pixel[:, :, 2] = 0 # blue 부분을 전부 0으로 변경
plt.imshow(img_pixel)
plt.show()
```

문제58. 인터넷에서 원하는 사진을 받아서 파이썬으로 시각화 하는 데 red, green, blue중에 하나로만 시각화 하시오.

21.03.09

2021년 3월 9일 화요일 오전 9:44

문제59. 아이린 사진을 흑백으로 변경하시오.

컬러사진 : RGB(3가지 색깔)

흑백사진 : Gray(1가지 색깔)

<https://cafe.daum.net/oracleoracle/SgRM/104>

```
j= 'C:\\Users\\KJM\\\\아이린.jpg'
```

```
import numpy as np # 신경망에 사진을 입력할 때 숫자행렬로 입력해야하기 때문에 필요
```

```
import matplotlib.pyplot as plt # 사진을 파이썬에서 시각화하기 위해 필요
```

```
import matplotlib.image as mpimg # 사진을 불러와서 숫자로 변환해주기 위해 필요
```

```
def rgb2gray(rgb): # 흑백으로 색깔을 변경하기 위한 함수
```

```
    return np.dot(rgb[:, :, :], [0.299, 0.587, 0.114])
```

```
img = mpimg.imread(j) # 사진을 숫자로 변경
```

```
gray = rgb2gray(img) # 컬러를 흑백으로 변환
```

```
#print (gray.flatten().shape) # 1차원으로 변경
```

```
plt.imshow(gray, cmap = plt.get_cmap('gray'))  
plt.show()
```

문제60. 어제 마지막 문제로 풀었던 사진을 흑백으로 변경하고 1차원으로 flatten 시키시오.

```
j= 'C:\\data\\\\lena2.png'
```

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

```
def rgb2gray(rgb):
```

```
    return np.dot(rgb[:, :, :], [0.299, 0.587, 0.114])
```

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

```
gray = rgb2gray(img)
```

```
#gray.flatten().shape
```

```
plt.imshow(gray, cmap = plt.get_cmap('gray'))
```

```
plt.show()
```

필기체 데이터를 인식하는 3층 신경망 구현하기

P.88 그림3-20 참고

common.py의 init_network 함수를 수정한다.

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

def softmax(a):
    import numpy as np
    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():
    import numpy as np
    network = {}
    network['W1'] = np.array([[1,3,5], [2,4,6]])
    network['W2'] = np.array([[3,4], [5,6], [7,8]])
    network['W3'] = np.array([[4,5], [6,7]])
    network['b1'] = np.array([0.1, 0.2, 0.3])
    network['b2'] = np.array([0.1, 0.2])
    network['b3'] = np.array([0.1, 0.2])
    return network
```

3층 신경망 코드

```
import numpy as np
from common import init_network, sigmoid, softmax

# 가중치 값을 불러온다.
network = init_network()
w1, w2, w3 = network['W1'], network['W2'], network['W3']
b1, b2, b3 = network['b1'], network['b2'], network['b3']

# 0층
x = np.array([1,2])

# 1층
a1 = np.dot(x, w1) + b1
a1_hat = sigmoid(a1)

# 2층
a2 = np.dot(a1_hat, w2) + b2
a2_hat = sigmoid(a2)
```

```
# 출력층  
y = np.dot(a2_hat, w3) + b3  
y_hat = softmax(y)  
  
print (y_hat)
```

피클파일

신경망을 학습시켜서 만들어 놓은 가중치를 파일로 생성하는 방법

- pickle을 이용한다.

ex1)

pickle 파일을 생성하는 예제

```
import pickle
```

```
a = ['a', 'b', 'c'] # 학습 시켜서 만든 가중치 값
```

```
with open('C:\data\\a.pkl', 'wb') as f:  
    pickle.dump(a, f)
```

ex2)

```
with open('C:\data\\a.pkl', 'rb') as f:  
    data = pickle.load(f)
```

```
print (data)
```

저자가 만든 sample_weight.pkl 로드

```
import pickle
```

```
def init_network():  
    with open('C:\data\\sample_weight.pkl', 'rb') as f:  
        network = pickle.load(f)  
    return network  
  
network = init_network()  
print (network.keys()) # dict_keys(['b2', 'W1', 'b1', 'W2', 'W3', 'b3'])  
print (network['W1'].shape) # (784, 50) 은 닉1층의 노드 수  
print (network['W2'].shape) # (50, 100) 은 닉2층의 노드 수  
print (network['W3'].shape) # (100, 10) 출력층의 노드 수
```

필기체 데이터를 불러오는 코드

```
import numpy as np  
from common import init_network, sigmoid, softmax  
from dataset.mnist import load_mnist  
  
(x_train, t_train), (x_test, t_test) = load_mnist(flatten=True, normalize=True,
```

```
    one_hot_label=False)
```

```
print (t_train[0:10])
```

필기체 데이터를 3층 신경망으로 예측값 출력하기

```
import numpy as np
from common import init_network, sigmoid, softmax
from dataset.mnist import load_mnist
```

1. 데이터를 불러온다.

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

2. 가중치와 바이어스 값을 불러온다.(저자가 미리 학습시킨 값)

```
network = init_network()
w1, w2, w3 = network['W1'], network['W2'], network['W3']
b1, b2, b3 = network['b1'], network['b2'], network['b3']
```

3. 신경망을 구성한다.

0층

```
x = x_train[0:10] # 10개의 데이터를 구성한다.
```

1층

```
a1 = np.dot(x, w1) + b1
a1_hat = sigmoid(a1)
```

2층

```
a2 = np.dot(a1_hat, w2) + b2
a2_hat = sigmoid(a2)
```

출력층

```
y = np.dot(a2_hat, w3) + b3
y_hat = softmax(y)
```

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

문제61. 위의 예측한 10장 중에 실제로 몇개를 맞췄는지 확인하시오.

```
import numpy as np
from common import init_network, sigmoid, softmax
from dataset.mnist import load_mnist
```

1. 데이터를 불러온다.

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

2. 가중치와 바이어스 값을 불러온다.(저자가 미리 학습시킨 값)

```
network = init_network()
w1, w2, w3 = network['W1'], network['W2'], network['W3']
```

```
b1, b2, b3 = network['b1'], network['b2'], network['b3']
```

3. 신경망을 구성한다.

0층

```
x = x_train[0:10] # 10개의 데이터를 구성한다.
```

1층

```
a1 = np.dot(x, w1) + b1
```

```
a1_hat = sigmoid(a1)
```

2층

```
a2 = np.dot(a1_hat, w2) + b2
```

```
a2_hat = sigmoid(a2)
```

출력층

```
y = np.dot(a2_hat, w3) + b3
```

```
y_hat = softmax(y)
```

```
print(np.argmax(y_hat, axis=1)) # 예측값
```

```
print(t_train[0:10]) # 실제 정답
```

문제62. 훈련 데이터 총 100장을 훌려보내고 100개중에 몇개를 맞추는지 확인하시오.

```
import numpy as np
from common import init_network, sigmoid, softmax
from dataset.mnist import load_mnist
```

1. 데이터를 불러온다.

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

2. 가중치와 바이어스 값을 불러온다.(저자가 미리 학습시킨 값)

```
network = init_network()
```

```
w1, w2, w3 = network['W1'], network['W2'], network['W3']
```

```
b1, b2, b3 = network['b1'], network['b2'], network['b3']
```

3. 신경망을 구성한다.

0층

```
x = x_train[0:100]
```

1층

```
a1 = np.dot(x, w1) + b1
```

```
a1_hat = sigmoid(a1)
```

2층

```
a2 = np.dot(a1_hat, w2) + b2
```

```
a2_hat = sigmoid(a2)
```

```

# 출력층
y = np.dot(a2_hat, w3) + b3
y_hat = softmax(y)

a = np.argmax(y_hat, axis=1)
b = t_train[0:100]

print (sum(a==b))

```

함수 생성

P.100 코드 참고

```

import numpy as np
from common import init_network, sigmoid, softmax
from dataset.mnist import load_mnist

# 1. 데이터를 불러온다.
def get_data():
    (x_train, t_train), (x_test, t_test) = load_mnist(flatten=True, normalize=True,
                                                    one_hot_label=False)
    return x_test, t_test

# 2. 가중치와 바이어스 값을 불러와서 3층 신경망에 흘려보내는 함수
def predict(network, x):
    #network = init_network()
    w1, w2, w3 = network['W1'], network['W2'], network['W3']
    b1, b2, b3 = network['b1'], network['b2'], network['b3']

    # 신경망을 구성한다.
    # 0층
    #x = x_train[0:100]

    # 1층
    a1 = np.dot(x, w1) + b1
    a1_hat = sigmoid(a1)

    # 2층
    a2 = np.dot(a1_hat, w2) + b2
    a2_hat = sigmoid(a2)

    # 출력층
    y = np.dot(a2_hat, w3) + b3
    y_hat = softmax(y)

    return y_hat

```

3. 위에서 만든 get_data함수와 predict함수로 실행하는 코드

```

x, t = get_data() # 테스트 데이터와 테스트 데이터의 정답을 불러오는 코드
network = init_network() # 저자가 만든 가중치와 바이어스를 불러오는 코드

```

```

accuracy_cnt = 0
for i in range(len(x)):
    y = predict(network, x[i])
    p = np.argmax(y)
    if p == t[i]:
        accuracy_cnt += 1

print ('정확도 :', accuracy_cnt/len(x))

```

문제63. 위의 코드에서 1000장만 테스트하여 정확도를 확인하시오.

```

import numpy as np
from common import init_network, sigmoid, softmax
from dataset.mnist import load_mnist

# 1. 데이터를 불러온다.
def get_data():
    (x_train, t_train), (x_test, t_test) = load_mnist(flatten=True, normalize=True,
                                                    one_hot_label=False)
    return x_test, t_test

# 2. 가중치와 바이어스 값을 불러와서 3층 신경망에 흘려보내는 함수
def predict(network, x):
    #network = init_network()
    w1, w2, w3 = network['W1'], network['W2'], network['W3']
    b1, b2, b3 = network['b1'], network['b2'], network['b3']

    # 신경망을 구성한다.
    # 1층
    a1 = np.dot(x, w1) + b1
    a1_hat = sigmoid(a1)

    # 2층
    a2 = np.dot(a1_hat, w2) + b2
    a2_hat = sigmoid(a2)

    # 출력층
    y = np.dot(a2_hat, w3) + b3
    y_hat = softmax(y)

    return y_hat

# 3. 위에서 만든 get_data함수와 predict함수로 실행하는 코드
x, t = get_data()
network = init_network()

accuracy_cnt = 0
for i in range(1000):
    y = predict(network, x[i])
    p = np.argmax(y)
    if p == t[i]:
        accuracy_cnt += 1

```

```
print ('정확도 :', accuracy_cnt/1000)
```

문제64. 훈련데이터 6만장의 정확도를 확인하시오.

```
import numpy as np
from common import init_network, sigmoid, softmax
from dataset.mnist import load_mnist

# 1. 데이터를 불러온다.
def get_data():
    (x_train, t_train), (x_test, t_test) = load_mnist(flatten=True, normalize=True,
                                                    one_hot_label=False)
    return x_train, t_train

# 2. 가중치와 바이어스 값을 불러와서 3층 신경망에 흘려보내는 함수
def predict(network, x):
    #network = init_network()
    w1, w2, w3 = network['W1'], network['W2'], network['W3']
    b1, b2, b3 = network['b1'], network['b2'], network['b3']

    # 신경망을 구성한다.
    # 1층
    a1 = np.dot(x, w1) + b1
    a1_hat = sigmoid(a1)

    # 2층
    a2 = np.dot(a1_hat, w2) + b2
    a2_hat = sigmoid(a2)

    # 출력층
    y = np.dot(a2_hat, w3) + b3
    y_hat = softmax(y)

    return y_hat

# 3. 위에서 만든 get_data함수와 predict함수로 실행하는 코드
x, t = get_data()
network = init_network()

accuracy_cnt = 0
for i in range(len(x)):
    y = predict(network, x[i])
    p = np.argmax(y)
    if p == t[i]:
        accuracy_cnt += 1

print ('정확도 :', accuracy_cnt/len(x))
```

문제65. 위의 코드에 1번과 2번 부분을 클래스로 생성하시오.

(Three_nn)

```
class Three_nn():
    import numpy as np
    from common import sigmoid, softmax
    from dataset.mnist import load_mnist

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

    # 1. 데이터를 불러온다.
    def get_data(self):
        (x_train, t_train), (x_test, t_test) = load_mnist(flatten=True, normalize=True,
                                                       one_hot_label=False)
        return x_train, t_train

    # 2. 가중치와 바이어스 값을 불러와서 3층 신경망에 흘려보내는 함수
    def predict(self, network, x):
        #network = init_network()
        w1, w2, w3 = network['W1'], network['W2'], network['W3']
        b1, b2, b3 = network['b1'], network['b2'], network['b3']

        # 신경망을 구성한다.
        # 1층
        a1 = np.dot(x, w1) + b1
        a1_hat = sigmoid(a1)

        # 2층
        a2 = np.dot(a1_hat, w2) + b2
        a2_hat = sigmoid(a2)

        # 출력층
        y = np.dot(a2_hat, w3) + b3
        y_hat = softmax(y)

        return y_hat

n1 = Three_nn()

# 3. 위에서 만든 get_data함수와 predict함수로 실행하는 코드
x, t = n1.get_data()
network = n1.init_network()
accuracy_cnt = 0

for i in range(len(x)):
    y = predict(network, x[i])
    p = np.argmax(y)
    if p == t[i]:
        accuracy_cnt += 1
```

```
print ('정확도 :', accuracy_cnt/len(x))
```

문제66. 훈련 데이터의 첫번째 데이터를 3층 신경망에 넣고 예측값을 출력하시오.

```
n1 = Three_nn()  
  
x, t = n1.get_data()  
network = n1.init_network()  
  
result = n1.predict(network, x[0])  
  
print (np.argmax(result))  
print (t[0])
```

문제67. 아이린 사진을 필기체를 인식하는 3층 신경망에 넣으면 어떻게 출력되는지 확인하시오.

차원이 일치하지 않아서 실행되지 않는다.

배치처리

훈련 데이터 6만장을 한번에 신경망에 넣고 학습을 시키게 되면 컴퓨터가 메모리 사용량이 초과하게 되어서 수행이 되지 않는다.

예를 들면 책을 볼 때 한페이지씩 보듯이 컴퓨터도 마찬가지로 메모리가 허용하는 내에서 여러 페이지를 학습 할 수 있도록 해주는 것

배치처리를 해야 학습이 더 빠르게 되고 컴퓨터로 학습이 가능한 상태가 된다.

문제68. 훈련데이터 6만장을 한번에 predict에 넣지 않고 100개씩 넣고 예측해서 정확도 600개가 a라는 리스트에 담기게 하시오.

```
n1 = Three_nn()  
  
# 3. 위에서 만든 get_data함수와 predict함수로 실행하는 코드  
x, t = n1.get_data()  
network = n1.init_network()  
a = []  
batch_size = 100  
accuracy_cnt = 0  
  
for i in range(0, len(x), batch_size): # 0, 100, 200, ...  
    y = predict(network, x[i:i+batch_size]) # x[0:100], x[100:200], ...
```

```
y_hat = np.argmax(y, axis=1) # 100개의 예측 숫자들 출력  
a.append(sum(y_hat == t[i:i+batch_size])/100) # 예측100개와 정답100개를 비교해서 정확도 계산  
  
print (a)  
print (np.mean(a))
```

★ 3장 내용 정리

1. 신경망 안에 들어가는 함수들(시그모이드, 렐루, 소프트맥스)을 파이썬으로 생성
2. 저자가 만든 가중치, 피클파일을 이용하여 3층 신경망 구현
3. 배치처리로 데이터를 신경망에 흘려보내는 코드 구현
 - 신경망이 학습할 때는 데이터 전체를 한번에 학습 할 수 없기 때문에 조금씩 학습해서 전체를 모두 학습하게 하는 것이 배치처리 이다.

신경망학습

(우리가 직접 신경망의 가중치와 바이어스를 생성)

신경망을 학습 시키기 위해서 알아야하는 내용

1. 오차함수 : 신경망의 어느 부분이 잘못되었는지 깨닫게 해주는 함수
2. 미니배치 : 학습할 때 데이터를 한번에 신경망에 넣는게 아니라 조금씩 신경망에 넣고 학습 시키는 것
3. 수치미분 : 오차함수의 기울기를 구해서 기울기만큼 가중치를 갱신해주는 역할을 할때 필요

오차함수

예상값과 실제값과의 오차를 신경망에 역전파 시켜주기 위해서 필요한 함수

시그모이드 함수, 렐루 함수, 소프트맥스 함수, 오차 함수

1. 평균제곱 오차 함수 (mean squared error) : 회귀분석시 사용
2. 교차엔트로피 오차 함수 (cross entropy error) : 분류시 사용

평균제곱 오차 함수

P.112 식 4-1 참고

```
import numpy as np
```

```

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

y = [0.1, 0.05, 0.6, 0.0, 0.05, 0.1, 0.0, 0.1, 0.0, 0.0] # 예측 숫자
t = [0, 0, 1, 0, 0, 0, 0, 0, 0, 0] # 정답 숫자

print (mean_squared_error(y, t)) # 0.09750000000000003

```

문제69. 숫자7로 예측한 결과와 정답숫자 2와의 오차를 구하시오.

```

import numpy as np

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

y = [0.1, 0.05, 0.1, 0.0, 0.05, 0.1, 0.0, 0.6, 0.0, 0.0] # 예측 숫자 7
t = [0, 0, 1, 0, 0, 0, 0, 0, 0, 0] # 정답 숫자 2

print (mean_squared_error(y, t)) # 0.5975

```

위의 오차 0.59는 오차가 작아서 분류문제를 해결하하기 위해서는 더 큰 오차로 응답해 줘야 한다. 그래서 필요한 함수가 바로 교차 엔트로피 함수이다.

교차엔트로피 함수

P.114 식 4-2 참고

```

import numpy as np

def cross_entropy_error(y, t):
    return - np.sum(t * np.log(y))

y = [0.1, 0.05, 0.1, 0.0, 0.05, 0.1, 0.0, 0.6, 0.0, 0.0] # 예측 숫자 7
t = [0, 0, 1, 0, 0, 0, 0, 0, 0, 0] # 정답 숫자 2

print (cross_entropy_error(y, t)) # nan

```

log 함수에 숫자 0이 들어갔기 때문에 결과에 nan이 출력된다.
log에 숫자 0이 들어가면 마이너스 무한대가 된다.

마이너스 무한대가 되지 않게 아주 작은 값(delta)을 더한다.

```

import numpy as np

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

```

```
y = [0.1, 0.05, 0.1, 0.0, 0.05, 0.1, 0.0, 0.6, 0.0, 0.0] # 예측 숫자 7  
t = [0, 0, 1, 0, 0, 0, 0, 0, 0, 0] # 정답 숫자 2  
  
print (cross_entropy_error(y, t)) # 2.302584092994546
```

평균제곱 오차 함수는 오차가 0.59인데 교차엔트로피 함수는 오차가 2.3으로 훨씬 큰 오차를 출력한다. 분류문제를 풀 때는 교차엔트로피 오차 함수를 사용해야 한다.

미니배치(P.115)

훈련 데이터 중에 일부만 골라서 학습하는 방법

6만장 중에 100장씩 신경망에 입력하여 학습시간을 줄인다.

100장을 추출할 때 복원추출, 비복원추출 할 것인가는 크게 중요하지 않고 결국 여러 번 반복해서 학습하다보면 학습이 된다.

문제70. 숫자 0~60000의 숫자 중에서 무작위로 10개를 출력하시오.

```
import numpy as np  
  
print (np.random.choice(np.arange(60001), 10))
```

문제71. 숫자 0~60000의 숫자 중에서 100개를 랜덤 추출하시오.

```
import numpy as np  
  
print (np.random.choice(np.arange(60001), 100))
```

문제72. 3장에서 마지막으로 작성한 필기체 숫자 예측하는 전체코드를 가져와서 100개를 랜덤추출하여 예측하게 할 수 있도록 코드를 수정하시오.

```
n1 = Three_nn()  
  
# 3. 위에서 만든 get_data함수와 predict함수로 실행하는 코드  
x, t = n1.get_data()  
network = n1.init_network()  
a = []  
batch_size = 100  
accuracy_cnt = 0  
  
for i in range(0, len(x), batch_size):  
    batch_mask = np.random.choice(len(x), batch_size)  
    y = predict(network, x[batch_mask])  
    y_hat = np.argmax(y, axis=1)
```

```
a.append(sum(y_hat == t[batch_mask])/100)
```

```
print (a)
```

수치미분(P.121)

가중치를 갱신해주기 위해서 미분이 필요하다.

가중치 = 가중치 - 기울기

P.121 식 4-4 참고

진정한 미분

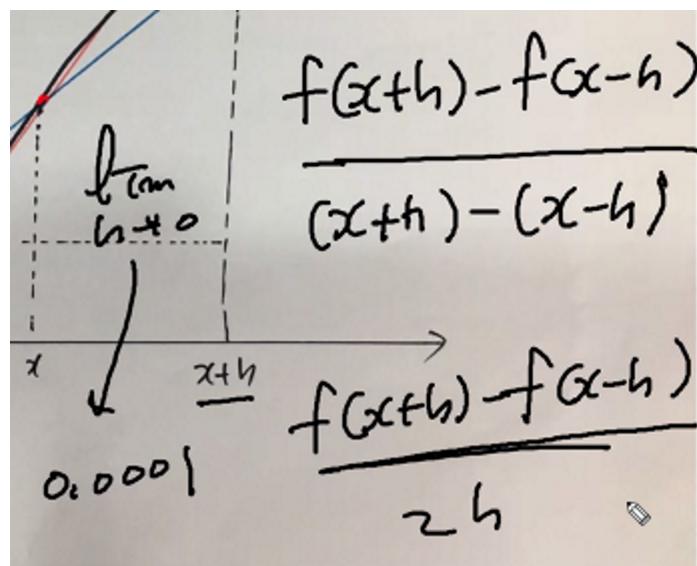
```
def numerical_diff(f, x):
    h = 10e-50
    return (f(x + h) - f(x)) / h

def f(x):
    return 2*x**2 + 2

print (numerical_diff(f, 4)) # 0.0
```

진정한 미분을 컴퓨터로 구현하면 분모가 0이 되기 때문에 계산이 되지 않는다.

도함수



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

```
def f(x):
    return 2*x**2 + 2
```

```
print (numerical_diff(f, 4)) # 15.999999999998238
```

문제73. 위에서 만든 미분함수를 이용하여 아래의 함수를 미분해서 기울기를 구하는데 $x = 6$ 일때의 미분 계수를 구하시오.

$$y = 3x^4 + 2x^3 + 6x^2 + 7$$

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

def f(x):
    return 3*x**4 + 2*x**3 + 6*x**2 + 7

print (numerical_diff(f, 6)) # 2880.0000007368
```

복습

1. numpy 사용법
 - 신경망에서 예측하는 모든 수학식이 행렬 계산 (numpy가 행렬연산을 고속으로 처리하는 모듈)
2. 퍼셉트론
 - 인공신경망의 원리 : 가중치를 갱신하는 작업
 - 3. 3층 신경망 구현 (저자가 만든 가중치를 세팅하여 신경망 구성)
 - 4. 2층 신경망 구현 (직접 필기체 데이터를 학습시켜서 신경망 구성)
 - a. 오차함수
 - b. 미니배치
 - c. 수치미분

진정한 접선으로 기울기를 구하는 것은 컴퓨터로 구현할 수 없어서 할선으로 기울기를 구해야 하는데, 그러면 아주 작은 오차가 발생하지만 그 정도 오차는 허용이 가능하다.

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

편미분 (P.125)

$$z = x^2 + y^2$$

변수가 2개 이상인 함수를 미분할 때 미분 대상 변수외에 나머지 변수를 상수처럼 고정시켜 미분하는것을 편미분이라고 한다.

문제74. 아래의 수학식을 손으로 편미분해서 $x_1 = 4, x_2 = 3$ 일때 기울기를 구하시오.

$$f = 2x_1^2 + 3x_2^2 + 4$$

$$\frac{\partial f}{\partial x_1} = 4x_1 \quad \text{기울기 } 16$$

$$\frac{\partial f}{\partial x_2} = 6x_2 \quad \text{기울기 } 18$$

문제75. 아래의 수학식을 오차함수로 생성하시오.

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

```
import numpy as np

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

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

print (loss_func(x))
```

문제76. 위의 loss_func() 함수를 $x_0 = 3, x_1 = 4$ 에서 x_0 에 대해 편미분했을때의 기울기를 구하시오.

```
import numpy as np

def numerical_diff(f, x):
    h = 0.0001
    return (f(x + h) - f(x - h)) / (2 * h)

def function_tmp1(x0):
    return x0**2 + 4**2

print (numerical_diff(function_tmp1, 3))
```

문제77. 위의 loss_func() 함수를 $x_0 = 3, x_1 = 4$ 에서 x_1 에 대해 편미분했을때의 기울기를 구하시오.

```

import numpy as np

def numerical_diff(f, x):
    h = 0.0001
    return (f(x + h) - f(x - h)) / (2 * h)

def function_tmp2(x1):
    return 3**2 + x1**2

print (numerical_diff(function_tmp1, 4))

```

위의 편미분 방법은 나머지 하나를 직접 상수화 시켜서 강제로 구현한 코드이므로 파이썬으로 알아서 편미분 되도록 함수를 다시 생성해야 한다.

편미분하는 numerical_gradient 함수 만들기 (P.127)

```

import numpy as np

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

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

def numerical_gradient(f, x):
    h = 1e-4
    grad = np.zeros_like(x) # x와 형상이 같은 배열 [0, 0]을 생성

    for i in range(x.size): # 0, 1
        tmp_val = x[i] # x[0], 3.0이 tmp_val에 담긴다.
        x[i] = tmp_val + h # [3.0001, 4.0]
        fxh1 = f(x) # 3.0001**2 + 4.0**2 = 25.00060001

        x[i] = tmp_val - h # [2.9999, 4.0]
        fxh2 = f(x) # 2.9999**2 + 4.0**2 = 24.99940001

        grad[i] = (fxh1 - fxh2) / (2*h) # 6.0000000000378 # [6.0, 4.0]
        x[i] = tmp_val

    return grad

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

```

문제78. 위의 함수를 방급한 것처럼 다시 디버깅하는데 i가 1일때를 디버깅하시오.

```

import numpy as np

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

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

```

```

def numerical_gradient(f, x):
    h = 1e-4
    grad = np.zeros_like(x) # x와 형상이 같은 배열 [0, 0]을 생성

    for i in range(x.size): # 0, 1
        tmp_val = x[i] # x[1], 4.0이 tmp_val에 담긴다.
        x[i] = tmp_val + h # [3.0, 4.0001]
        f_xh1 = f(x) # 3.0**2 + 4.0001**2 = 25.00080001

        x[i] = tmp_val - h # [3.0, 3.9999]
        f_xh2 = f(x) # 3.0**2 + 3.9999**2 = 24.99920001

        grad[i] = (f_xh1 - f_xh2) / (2*h) # 7.99999999999119 # [6.0, 7.99]
        x[i] = tmp_val

    return grad

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

```

경사하강법 (P.129)

위에서 만든 numerical_gradient 함수는 산에서 내려오기 위해서 내가 서있는 곳에서 어느쪽으로 가야 산 아래로 내려갈 수 있는지 내가 서있는 곳의 기울기를 구하는 함수이다.

가중치 = 가중치 - 기울기

위의 식을 loop문으로 반복수행해서 기울기가 0이 되서 가중치가 변경이 되지 않는 시점까지 수행해서 최적의 가중치를 알아낸다.

가중치 = 가중치 - 학습률 * 기울기

학습률(learning rate) : 한번의 학습으로 얼마만큼 매개변수를 갱신할지를 결정하는 하이퍼 파라미터

학습률은 개발자가 0.01이나 0.001등과 같이 알아서 정해줘야하는데 일반적으로 이 값이 너무 크거나 작으면 gloval minimum을 찾아갈 수 없다.

문제79. 위에서 만든 numerical_gradient 함수를 이용해서 경사하강을 하는 gradient_descent 함수를 생성하시오.

```

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
```

문제80. P.132처럼 함수 $f(x_0, x_1) = x_0^{**2} + x_1^{**2}$ 함수를 오차함수로 두고 처음 지점을 $[-3.0, 4.0]$ 이라고 하고 계속 경사하강을 하여 최소지점인 $[0, 0]$ 으로 경사하강되게 하시오.

```
import numpy as np

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

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

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

gradient_descent(loss_func, init_x, lr=0.1, step_num=100)
```

문제81. 러닝레이트를 0.1로 하지 않고 너무 작은 값을 주어서 최소지점에 도달하지 못하는지 확인하시오.

설현 사진 resize하고 흑백처리(신경망에 입력되기 전 데이터 구성)

<https://cafe.daum.net/oracleoracle/SgRM/150>

1. c:\data10 폴더 밑에 있는 파일들의 이름을 불러온다.

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

path = "c:\\\\data10"
file_list = os.listdir(path) # path에 지정된 위치에 있는 파일들의 이름을 불러온다.
#file_list # ['a.jpg'] 리스트형식
```

2. 설현사진을 128x128로 resize 한다.

```
for k in file_list: # 리스트 안에 있는 파일들을 하나씩 빼내는 코드
    img = cv2.imread(path + '\\' + k) # 설현사진을 숫자행렬로 변경
    width, height = img.shape[:2] # 설현사진 숫자 행렬에서 가져온다.
    resize_img = cv2.resize(img, (128 , 128), interpolation=cv2.INTER_CUBIC)
    cv2.imwrite('c:\\\\data11\\\\resize\\\\' + k, resize_img) # resize한 이미지를 저장
```

3. 설현사진을 흑백으로 변경한다.

```
j= 'c:\\\\data11\\\\resize\\\\a.jpg'

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

img = mpimg.imread(j)
gray = rgb2gray(img)
plt.imshow(gray, cmap = plt.get_cmap('gray'))
plt.show()
```

4. 설현 사진을 mnist 신경망에 넣는다.

```
# 1차원으로 변경
x = gray.flatten()
x.shape

import numpy as np
import pickle
from common import sigmoid, softmax
from dataset.mnist import load_mnist
```

1. 파이썬 기초 ---> 2. 함수를 생성 ----> 3. 함수로 클래스를 생성

```
class Three_nn():
    import numpy as np
    import pickle
    from common import sigmoid, softmax
    from dataset.mnist import load_mnist

    def init_network(self):
        import pickle
        with open("c:\\\\data\\\\sample_weight.pkl", "rb") as f:
            network = pickle.load(f)
        return network
```

1. 데이터를 불러옵니다. (얀르쿤 교수님이 만든 필기체 데이터)

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

2. 가중치와 바이어스 값을 불러와서 3층 신경망에 흘려보내는 함수

```
def predict(self, network, x):
    #network = init_network()
    w1, w2, w3 = network['W1'], network['W2'], network['W3']
    b1, b2, b3 = network['b1'], network['b2'], network['b3']
```

```

# 3. 신경망을 구성합니다.

# 0층
# x = x_train[0:100] # 일단 10개의 필기체 데이터를 구성합니다.

# 1층
y = np.dot(x,w1) + b1
y_hat = sigmoid(y)

# 2층
z = np.dot(y_hat, w2) + b2
z_hat = sigmoid(z)

# 3층
k = np.dot(z_hat, w3) + b3
k_hat = softmax(k)
return k_hat

```

n1 = Three_nn() # 객체화 시킨다. 설계도 가지고 제품을 만든다.

```

#x, t = n1.get_data() # 테스트 데이터와 테스트 데이터의 정답을 불러오는 코드
network = n1.init_network() # 저자가 만들어온 가중치와 바이어스를 불러오는 코드

result = n1.predict( network, x )
print (np.argmax(result) )

```

문제82. 다른 설현사진을 신경망에 넣고 어떤 숫자가 나오는지 확인하시오.

1. c:\\data10 폴더 밑에 있는 파일들의 이름을 불러온다.

```

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

path = "c:\\\\data10"
file_list = os.listdir(path) # path에 지정된 위치에 있는 파일들의 이름을 불러온다.
file_list # ['a.jpg'] 리스트형식

```

2. 설현사진을 128x128로 resize 한다.

```

for k in file_list: # 리스트 안에 있는 파일들을 하나씩 빼내는 코드
    img = cv2.imread(path + '\\'+ k) # 설현사진을 문자행렬로 변경
    width, height = img.shape[:2] # 설현사진 숫자 행렬에서 가져온다.
    resize_img = cv2.resize(img, (28 , 28), interpolation=cv2.INTER_CUBIC)
    cv2.imwrite('c:\\\\data11\\\\resize\\\\' + k, resize_img) # resize한 이미지를 저장

plt.imshow(resize_img) # resize된 사진을 시각화
plt.show()

```

```
# 3. 설현 사진을 흑백으로 변경한다.
```

```
j= 'c:\\data11\\resize\\b.jpg'
```

```
def rgb2gray(rgb):
```

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

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

```
gray = rgb2gray(img)
```

```
plt.imshow(gray, cmap = plt.get_cmap('gray'))
```

```
plt.show()
```

```
# 4. 설현 사진을 mnist 신경망에 넣는다.
```

```
# 1차원으로 변경
```

```
x = gray.flatten()
```

```
x.shape
```

```
import numpy as np
```

```
import pickle
```

```
from common import sigmoid, softmax
```

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

```
# 1. 파이썬 기초 ---> 2. 함수를 생성 ----> 3. 함수로 클래스를 생성
```

```
class Three_nn():
```

```
    import numpy as np
```

```
    import pickle
```

```
    from common import sigmoid, softmax
```

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

```
    def init_network(self):
```

```
        import pickle
```

```
        with open("c:\\data\\sample_weight.pkl", "rb") as f:
```

```
            network = pickle.load(f)
```

```
        return network
```

```
# 1. 데이터를 불러옵니다. ( 얀르쿤 교수님이 만든 필기체 데이터)
```

```
def get_data(self):
```

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

```
    return x_train, t_train
```

```
# 2. 가중치와 바이어스 값을 불러와서 3층 신경망에 흘려보내는 함수
```

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

```
    #network = init_network()
```

```
    w1, w2, w3 = network['W1'], network['W2'], network['W3']
```

```
    b1, b2, b3 = network['b1'], network['b2'], network['b3']
```

```
# 3. 신경망을 구성합니다.
```

```
# 0층
```

```
# x = x_train[0:100] # 일단 10개의 필기체 데이터를 구성합니다.
```

```
# 1층
```

```
y = np.dot(x,w1) + b1
```

```

y_hat = sigmoid(y)
# 2층
z = np.dot(y_hat, w2) + b2
z_hat = sigmoid(z)
# 3층
k = np.dot(z_hat, w3) + b3
k_hat = softmax(k)
return k_hat

```

`n1 = Three_nn()` # 객체화 시킨다. 설계도 가지고 제품을 만든다.

```

#x, t = n1.get_data() # 테스트 데이터와 테스트 데이터의 정답을 불러오는 코드
network = n1.init_network() # 저자가 만들어온 가중치와 바이어스를 불러오는 코드

result = n1.predict( network, x )
print (np.argmax(result) )

```

편미분을 하는 `gradient_descent` 함수를 이용해서 2층 신경망 구현

저자가 만든 소스 코드내에 common이라는 폴더가 있는데 이 폴더를 주피터 워킹 디렉토리에 가져다 두고 기존에 있던 common.py는 common1.py로 이름을 변경한다.

common 폴더(패키지) 안에는 functions.py, gradient.py 도 있는데 functions.py에는 신경망에 필요한 함수들이 들어가 있고 gradient.py에는 편미분하는 함수 numerical_gradient 함수가 들어가 있다.

2층 신경망 만들기

```

import sys, os
sys.path.append(os.pardir) # 부모디렉토리의 파일들을 가져올 수 있도록 설정
import numpy as np
from common.functions import softmax, cross_entropy_error
from common.gradient import numerical_gradient

z = np.array([0.1, 0.9])

print (softmax(z))

```

ex1)

가중치 행렬을 2x3으로 랜덤으로 숫자를 생성해서 생성하시오.

```

import numpy as np

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

```

ex2)

아래의 입력 데이터를 1x2 행렬로 만들고 위에서 만든 가중치 행렬과 내적하시오.

```
import numpy as np

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

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

ex3)

설현 사진을 가중치 행렬과 내적해서 나온 결과 행렬을 softmax함수에 넣어서 확률을 출력하시오.

```
import numpy as np
```

```
x = np.array([0.6, 0.9]) # 설현 사진
W = np.random.randn(2, 3)
z = (np.dot(x, W))
y = softmax(z)

print (y)
```

ex4)

위의 출력된 확률을 정답과 함께 오차함수에 넣어서 오차를 구하시오.

```
import numpy as np
```

```
x = np.array([0.6, 0.9]) # 설현 사진
np.random.seed(1) # seed값 설정
W = np.random.randn(2, 3)
t = np.array([0, 1, 0])
z = (np.dot(x, W))
y = softmax(z)
loss = cross_entropy_error(y, t)

print (loss)
```

ex5)

비용함수를 생성해서 비용함수를 미분하시오.

```
f = lambda w : net.loss(x, t) # 비용함수 생성
dW = numerical_gradient(f, net.W)

print (dW) # 기울기 출력
```

ex6)

1층 신경망 전체코드를 구현 (P.135)

```
import sys, os
```

```

sys.path.append(os.pardir) # 부모디렉토리의 파일들을 가져올 수 있도록 설정
import numpy as np
from common.functions import softmax, cross_entropy_error
from common.gradient import numerical_gradient

class simpleNet:
    # 가중치 행렬 생성
    def __init__(self): # 설계도로 제품을 만들 때 바로실행되는 함수
        self.W = np.random.randn(2, 3) # 랜덤으로 가중치 행렬을 생성

    # 입력값을 받아서 가중치 행렬과 내적한 결과를 출력
    def predict(self, x): # 설현 사진을 입력해서 예측하는 함수
        return np.dot(x, self.W)

    # 오차 출력
    def loss(self, x, t):
        z = self.predict(x) # 설현 사진을 넣어서 z값을 출력
        y = softmax(z) # z값을 받아서 확률벡터를 출력
        loss = cross_entropy_error(y, t) # 확률벡터와 정답을 넣어서 오차를 출력
        return loss

x = np.array([0.6, 0.9]) # 설현 사진
t = np.array([0, 0, 1]) # [아이린, 설현, 아이유]
net = simpleNet() # 클래스(2층 신경망 설계도)로 객체 생성
f = lambda w : net.loss(x, t) # 비용함수 생성
dW = numerical_gradient(f, net.W)

print(dW) # 기울기 출력

```

가중치가 2x3행렬이니까 기울기도 2x3행렬로 나와야 가중치에서 기울기를 출력할 수 있다.

문제83. 아래의 입력 데이터와 target(정답)을 simplenet에 입력하고 오차를 출력하시오.

```

x = np.array([0.8, 0.2])
t = np.array([0, 0, 1])

x = np.array([0.8, 0.2])
t = np.array([0, 0, 1])
net = simpleNet() # 클래스(2층 신경망 설계도)로 객체 생성

print(net.loss(x, t))

```

문제84. simpleNet()클래스에 있는 가중치 행렬 W를 출력하시오.

```
net = simpleNet()
```

```
print (net.W)
```

문제85. simpleNet()클래스에 있는 predict 함수에 아래의 입력 데이터를 넣고 결과를 출력하시오.

```
x = np.array([0.9, 0.6])  
net = simpleNet()
```

```
print (net.predict(x))
```

필기체 데이터를 학습 시키는 2층 신경망 전체 코드(P.138)

C:\Users\KJM\Desktop\딥러닝\deep-learning-from-scratch-master\ch04\two_layer_net.py

문제86. simpleNet 클래스를 객체화 시켜서 실행하는데 이 신경망에 설현 사진을 입력할 수 있도록 가중치 행렬의 shape를 수정하고 설현 사진을 입력하여 확률벡터를 출력하시오.

21.03.11

2021년 3월 11일 목요일 오전 9:46

2층 신경망 구현하기

4장 전체코드 수행

<https://cafe.daum.net/oracleoracle/SgRM/162>

문제87. numerical_gradient 함수를 실행하여 기울기를 출력하시오.

```
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)
(x_train, t_train), (x_test, t_test) = load_mnist(normalize=True, one_hot_label=True)
network.numerical_gradient(x_train[:100,:], t_train[:100,:])
```

문제88. 10 에폭이 아니라 20에폭돌게 코드를 수정하고 시각화 하시오.

```
# 하이퍼파라미터
iters_num = 12000 # 반복 횟수를 적절히 설정
train_size = x_train.shape[0]
batch_size = 100 # 미니배치 크기
learning_rate = 0.1 # 학습률

train_loss_list = []
train_acc_list = []
test_acc_list = []
```

학습시킨 가중치 파일로 생성

pickle 파일을 생성하는 예제

```
ex1)
import pickle
```

```
params = [3.1, 4.3, 5.3] # 가중치  
  
with open('c:\\data\\weight7.pkl', 'wb') as f:  
    pickle.dump(params, f)
```

문제89. 4장에서 만든 2층 신경망의 가중치와 바이어스를 pickle 파일로 내리시오.

```
# 4. pickle 파일로 내리기  
import pickle  
  
with open('c:\\data\\mnist_weight.pkl', 'wb') as f:  
    pickle.dump(network.params, f)
```

문제90. 3장에서 사용한 3층 신경망 코드를 2층 신경망으로 변경하고 위의 pickle 파일을 셋팅해서 필기체를 분류할 수 있게 하시오.

<https://cafe.daum.net/oracleoracle/SgRM/176>

★ 4장 내용 정리

1. 신경망을 학습에 필요한 3가지
 - a. 오차함수 (항등함수, 교차엔트로피 함수)
 - b. 수치미분
 - c. 미니배치
2. 미분함수를 파이썬으로 구현
3. 편미분 함수를 파이썬으로 구현
4. 2층 신경망을 구현 (학습 가능한 신경망)
5. 신경망의 가중치를 pickle 파일 내리기

오차 역전파

수치 미분으로 신경망 학습을 하면 너무 느리기 때문에 오차 역전파를 이용해서 학습한다.

계산 그래프

순전파와 역전파에 계산과정을 그래프로 나타내는 방법

계산 그래프의 장점

- 국소적 계산을 할 수 있다.

국소적 계산(P.150 그림 5-4)

- 전체에서 어떤일이 벌어지든 상관없이 자신과 관련된 정보만으로 원하는 결과를 얻어 낼 수 있는 계산방식

계산 그래프로 문제를 해결하는 이유

전체가 아무리 복잡해도 각 노드에서 단순한 계산에 집중하여 문제를 단순화 시킬 수 있다.

큰 문제를 해결하는 방법이 큰 문제는 작은 문제들이 여러개로 뭉여 있는 것이므로 작은 문제들을 하나씩 해결하면서 큰 문제를 해결하는 방법(다이나믹 프로그래밍)

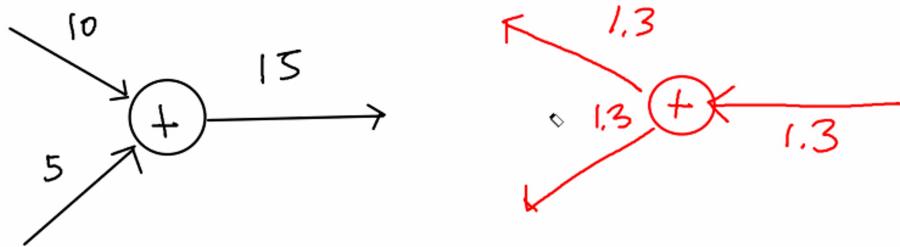
1. 입력값
2. 활성화 함수
 - a. 시그모이드 함수
 - b. 렐루 함수
3. 출력층 함수
 - a. 소프트맥스 함수 (분류)
 - b. 항등 함수 (회귀)
4. 오차 함수
 - a. 교차엔트로피 함수 (분류)
 - b. 평균제곱 오차 (회귀)

덧셈 계산 그래프

P.156 그림 5-9 참고

덧셈노드의 역전파는 상류에서 흘러왔던 값이 그대로 흘러간다.

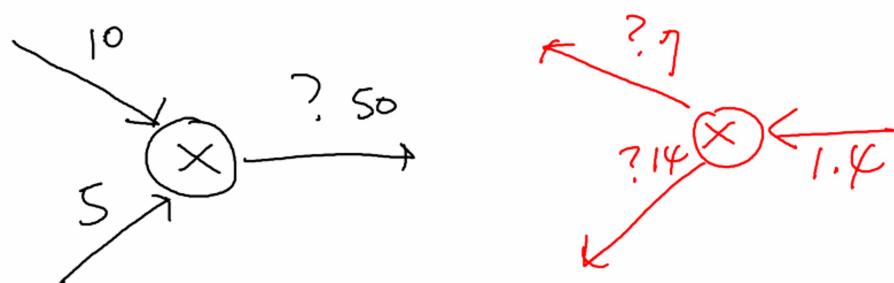
문제91. 아래의 덧셈노드 그래프의 순전파 값과 역전파 값을 적으시오.



곱셈 노드의 순전파와 역전파

P.158 그림 5-12 참고

문제92. 아래의 곱셈노드의 순전파 값과 역전파 값을 각각 적으시오.



문제93. 곱셈계층을 파이썬으로 구현하시오.

(P.161)

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

    def forward(self, x, y): # 순전파
        self.x = x
        self.y = y
        out = x * y

        return out

    def backward(self, dout): # 역전파
        dx = dout * self.y
        dy = dout * self.x

        return dx, dy
```

문제94. 위에서 만든 곱셈 클래스를 객체화 시켜서 아래의 사과가격을 구하시오.

```

apple = 100
apple_num = 2

apple = 100
apple_num = 2

class MulLayer:
    def __init__(self):
        self.x = None
        self.y = None

    def forward(self, x, y): # 순전파
        self.x = x
        self.y = y
        out = x * y

        return out

    def backward(self, dout): # 역전파
        dx = dout * self.y
        dy = dout * self.x

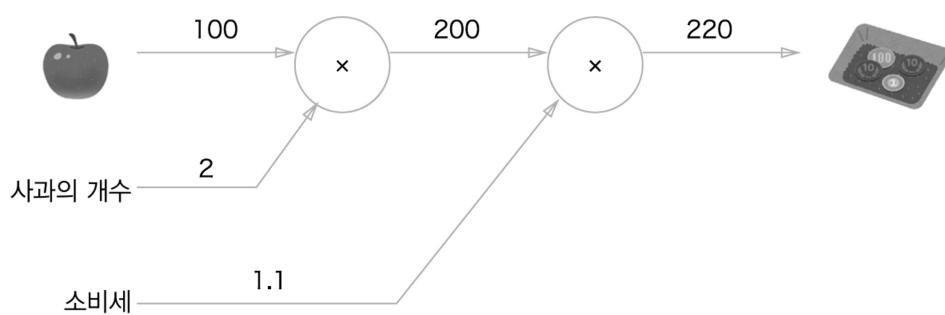
        return dx, dy

mul_apple_layer = MulLayer()
apple_price = mul_apple_layer.forward(apple, apple_num)

print (apple_price)

```

문제95. 곱셈계층 클래스를 객체화 시켜서 아래의 그림 5-2를 계산 하시오.



```

apple = 100
apple_num = 2
tax = 1.1

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

mul_apple_layer = MulLayer()
apple_price = mul_apple_layer.forward(apple, apple_num)
price = mul_apple_layer.forward(apple_price, tax)

print (price)

```

문제96. 덧셈 계층 클래스를 파이썬으로 구현하시오.
(P.163)

```

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

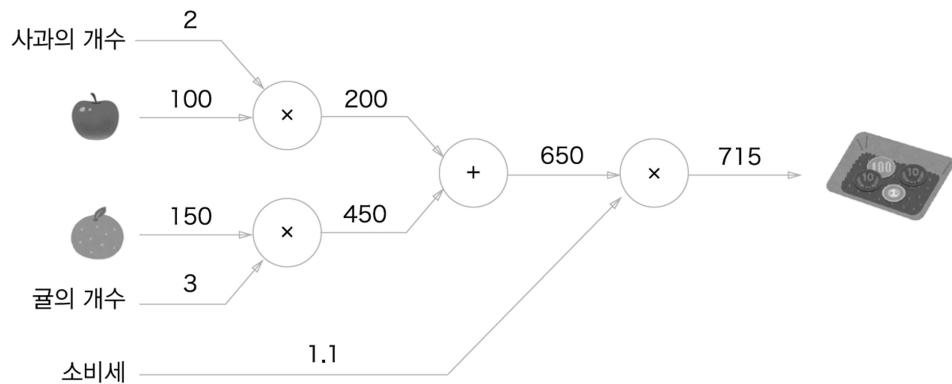
        return out

    def backward(self, dout):
        dx = dout
        dy = dout

        return dx, dy

```

문제97. 위에서 만든 곱셈클래스와 덧셈 클래스를 이용해서 그림 5-3의 계산 그래프를 구현하고 최종 값을 계산하시오.



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

```
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
```

```
mul_apple_layer = MulLayer()
mul_orange_layer = MulLayer()
```

```

add_layer = AddLayer()

apple_price = mul_apple_layer.forward(apple, apple_num)
orange_price = mul_orange_layer.forward(orange, orange_num)
apple_price_tax = mul_apple_layer.forward(apple_price, tax)
orange_price_tax = mul_orange_layer.forward(orange_price, tax)
price = add_layer.forward(apple_price_tax, orange_price_tax)

print (price)

```

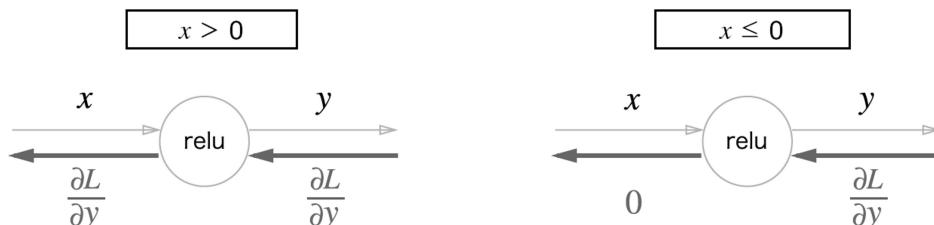
활성화 함수 계층 구현하기(P.165)

1. 계산 그래프
 - 덧셈 그래프
 - 곱셈 그래프
 - 렐루 함수 그래프
 - 시그모이드 함수 그래프
 - 교차엔트로피 함수 그래프(부록)
 - 소프트맥스 함수 그래프(부록)

계산 그래프를 보면서 순전파, 역전파 함수를 생성

렐루 함수를 위한 계산 그래프(P.165)

0보다 큰 값이 입력 되면 그 값을 그대로 출력하고 0이거나 작은 값을 입력하면 0이 출력되는 함수



렐루함수를 구현하려면 알아야하는 문법2 |

1. copy 모듈 사용법
2. $x[x \leq 0]$ 의 의미

copy 모듈 사용법

```

ex1)
a = [1, 2, 3]
b = a

```

```

print (a) # [1, 2, 3]
print (b) # [1, 2, 3]

```

```
a[1] = 6
```

```
print (a) # [1, 6, 3]
print (b) # [1, 6, 3]
```

```
ex2)
a = [1, 2, 3]
b = a.copy()
```

```
print (a) # [1, 2, 3]
print (b) # [1, 2, 3]
```

```
a[1] = 6
```

```
print (a) # [1, 6, 3]
print (b) # [1, 2, 3]
```

x[x <= 0]의 의미

```
import numpy as np
```

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

```
print (x)
```

```
mask = (x <= 0)
```

```
print (mask)
```

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

```
print (out)
```

문제98. 책 166 페이지의 ReLU 클래스를 생성하시오.

21.03.15

2021년 3월 15일 월요일 오전 9:41

정리

- 1장. numpy 사용방법
- 2장. 퍼셉트론
- 3장. 3층 신경망
- 4장. 2층 신경망
- 5장. 오차 역전파를 이용한 2층 신경망
- 6장. 오버피팅과 언더피팅을 막는 방법
- 7장. CNN
- 8장. 딥러닝의 역사 (텐서 플로우, 파이토치)

sigmoid계층(P.167)

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

ex1)

시그모이드 함수의 순전파를 파이썬 코드로 구현하시오.

```
import numpy as np

class Sigmoid:
    def __init__(self):
        self.out = None

    def forward(self, x):
        out = 1 / (1+np.exp(-x))
        self.out = out

    return out
```

문제99. 위의 Sigmoid 클래스를 객체화 시켜서 아래의 데이터를 forward함수에 넣고 실행하시오.

```
import numpy as np

class Sigmoid:
    def __init__(self):
        self.out = None
```

```

def forward(self, x):
    out = 1 / (1+np.exp(-x))
    self.out = out

    return out

x = np.array([24, 32, 5])

sigmoid = Sigmoid()

sigmoid.forward(x)

```

시그모이드 함수를 파이썬으로 날코딩하기(backward)

(P.167 ~ P.169)

```

import numpy as np

class Sigmoid:
    def __init__(self):
        self.out = None

    def forward(self, x):
        out = 1 / (1+np.exp(-x))
        self.out = out

        return out

    def backward(self, dout):
        dx = dout * (1.0 - self.out) * self.out

        return dx

```

문제100. 위에서 만든 시그모이드 클래스를 객체화 시켜서 순전파와 역전파를 각각 구현하시오.

```

import numpy as np

class Sigmoid:
    def __init__(self):
        self.out = None

    def forward(self, x):
        out = 1 / (1+np.exp(-x))
        self.out = out

        return out

    def backward(self, dout):
        dx = dout * (1.0 - self.out) * self.out

        return dx

sim = Sigmoid()

```

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

dout = np.array([[2.0, 3.0], [-3.0, -4.0]])
print(sim.backward(dout))
```

텐서 플로우 2.0 환경구성

<https://cafe.daum.net/oracleoracle/Sedp/287>

텐서플로우를 이용해서 sigmoid 함수에 데이터를 넣고 계산

```
import tensorflow as tf

x = tf.constant([24.0, 32.0, 5.0]) # 3개의 상수값을 생성(실수형)

tf.math.sigmoid(x) # 위의 3개의 숫자를 sigmoid 함수에 흘려보낸다.
```

문제

5장의 오차 역전파를 구현할 파이썬 코드 구현

1. relu 함수
2. sigmoid 함수
3. Affine 계층
4. softmax 함수
5. 오차 함수

Affine 계층

신경망의 순전파때 수행하는 행렬의 내적을 기하학에서는 어파인 변환이라고 한다.
그래서 신경망에서 입력값과 가중치의 내적의 합에 바이어스를 더하는 층을 Affine 계층이라고 해서 구현한다.

지금까지의 계산 그래프는 노드 사이에 스칼라값이 흘렀는데 이에 반해 이번에는 행렬이 흐르고 있어서 Affine 계층 구현이 필요하다.

Affine 계층을 이해하기 위한 행렬의 종류 설명

1. 전치 행렬
2. 역행렬

<https://cafe.daum.net/oracleoracle/Sedp/205>

<https://cafe.daum.net/oracleoracle/Sedp/206>

문제102. 책 175 페이지에 나오는 Affine 계층 클래스를 생성하시오.

```
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
        out = np.dot(x, self.W) + self.b

        return out

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

affine = Affine(W, b)

affine.forward(x)
```

순전파는 위와 같이 구현하고 역전파는 계산그래프를 그려서 도함수를 확인하고 도함수로 backward 함수를 구현하면 된다.

문제103. 아래의 batch로 데이터를 신경망에 입력했을 때의 순전파를 구현하시오.

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

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
        out = np.dot(x, self.W) + self.b

        return out
```

```

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

affine1 = Affine(W, b)

affine1.forward(x)

```

Affine 계층의 역전파 계산그래프

```

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
        out = np.dot(x, self.W) + self.b

        return out

    def backward(self, dout):
        dx = np.dot(dout, self.W.T)
        self.dW = np.dot(self.x.T, dout)
        self.db = np.sum(dout, axis = 0)

        return dx

```

Affine 계층의 역활은 신경망에서 층을 하나 구성하는 부분이다.

P.182

Tensorflow 2.x 에서 퍼셉트론 구현

<https://cafe.daum.net/oracleoracle/Sedp/492>



텐씨플로우
2.x 버전...

딥러닝 5장의 mnist 신경망 (필기체 분류 신경망)을 텐씨플로우 2.0으로 구현

순서

1. mnist 데이터를 불러온다.

2. 훈련 데이터의 일부를 검증 데이터로 구성한다.
3. 모델에 데이터 입력하기 전에 정규화를 한다. (데이터 전처리)
4. 모델에 입력할 정답을 one hot encoding 한다.
5. 모델을 구성한다.
6. 모델 과정을 설정한다.
7. 모델을 학습시킨다.
8. 학습 된 결과 모델을 확인한다.

C:\Users\KJM\Desktop\딥러닝\tensorflow_mnist데이터.ipynb

21.03.16

2021년 3월 16일 화요일 오전 9:45

5장 복습

수치미분이 너무 느리기 때문에 오차역전파를 사용해서 기울기를 구해 가중치를 갱신해줘야 한다. 그런데 오차 역전파를 이해하기 쉽도록 그림으로 코딩 전에 이론을 이해하는 과정이 계산 그래프이다.

1. 계산그래프를 통해 덧셈 그래프와 곱셈 그래프를 이해
2. ReLU 함수 클래스 생성
3. sigmoid 함수 계산 그래프, 클래스 생성 (순전파, 역전파)
4. Affine 계층 (행렬의 내적) 계산 그래프, 클래스 생성
5. 텐서플로우 2.x 으로 3층 신경망 구현

문제109. 3층 신경망에 활성화 함수를 relu로 했을 때의 테스트 데이터의 정확도는 0.975 인데 sigmoid 함수를 사용했을 때의 정확도를 확인하시오.

5. 신경망 모델을 구성한다.

```
model = Sequential()  
model.add(Dense(100, activation = 'sigmoid', input_shape = (784, )))  
model.add(Dense(50, activation = 'sigmoid'))  
model.add(Dense(10, activation = 'softmax'))
```

0.9762

문제110. 위의 결과를 시각화해서 선능을 확인하시오.

<https://cafe.daum.net/oracleoracle/SgRM/217>

시각화

```
import matplotlib.pyplot as plt  
  
his_dict = history.history  
loss = his_dict['loss']  
val_loss = his_dict['val_loss'] # 검증 데이터가 있는 경우 'val_' 수식어가 붙습니다.  
  
epochs = range(1, len(loss) + 1)  
fig = plt.figure(figsize = (10, 5))  
  
# 훈련 및 검증 손실 그리기
```

```

ax1 = fig.add_subplot(1, 2, 1)
ax1.plot(epochs, loss, color = 'blue', label = 'train_loss')
ax1.plot(epochs, val_loss, color = 'orange', label = 'val_loss')
ax1.set_title('train and val loss')
ax1.set_xlabel('epochs')
ax1.set_ylabel('loss')
ax1.legend()

acc = his_dict['acc']
val_acc = his_dict['val_acc']

# 훈련 및 검증 정확도 그리기
ax2 = fig.add_subplot(1, 2, 2)
ax2.plot(epochs, acc, color = 'blue', label = 'train_acc')
ax2.plot(epochs, val_acc, color = 'orange', label = 'val_acc')
ax2.set_title('train and val acc')
ax2.set_xlabel('epochs')
ax2.set_ylabel('acc')
ax2.legend()

plt.show()

```

시각화를 해서 오버피팅이 일어나고 있는 것을 확인 가능하다.

문제111. 위의 신경망은 3층 신경망인데 4층으로 신경망을 구성하시오.

5. 신경망 모델을 구성한다.

```

model = Sequential()
model.add(Dense(100, activation = 'sigmoid', input_shape = (784, )))
model.add(Dense(50, activation = 'sigmoid'))
model.add(Dense(50, activation = 'sigmoid'))
model.add(Dense(10, activation = 'softmax'))

```

tensorflow 를 사용하지 않고 3층 신경망 구현하기

<https://cafe.daum.net/oracleoracle/Sedp/235>



오차 역전
파로 구...

4장의 수치미분과 5장의 오차역전파의 차이

<https://cafe.daum.net/oracleoracle/Sedp/202>

수치미분은 오차함수를 바로 가중치로 편미분해서 기울기를 구했지만 이렇게 계산하면 시간

이 오래 걸린다. 그래서 합성함수 미분처럼 연쇄법칙에 의해서 신경망 안에 들어가는 각각의 함수들의 도함수를 구해서 도함수를 backward에 넣고 기울기가 역전파되어서 훌러가게 하면 그 기울기로 가중치를 갱신하는 것이 수치미분으로 했을 때 보다 훨씬 빠르다.

P.182~P.183

5장의 전체코드를 텐서플로우, 파이토치로 구현할 수 있어야 한다.

학습 관련 기술들

1. underfitting을 방지하는 방법

- 고급 경사 하강법의 종류
- 가중치 초기값 설정
- 배치 정규화

2. overfitting을 방지하는 방법

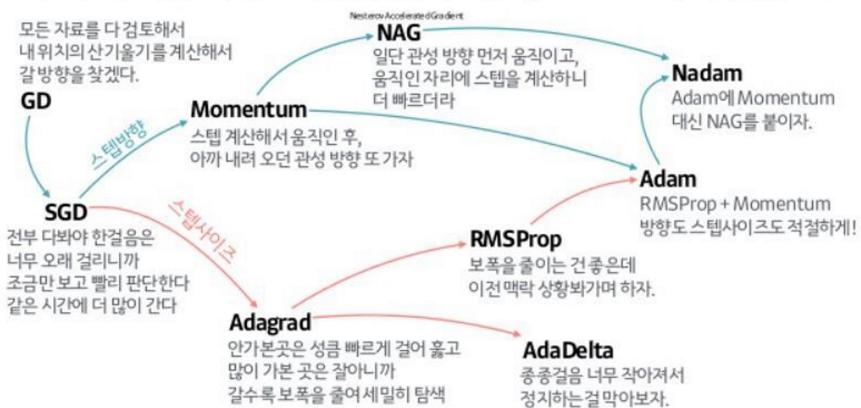
- 드롭아웃
- L2 정규화
- 엘리스텝 (keras의 기능)

고급 경사하강법 정의

<https://cafe.daum.net/oracleoracle/SgRM/214>

<https://cafe.daum.net/oracleoracle/Sedp/236>

산내려오는 작은 오솔길 잘찾기(Optimizer)의 발달 계보



출처: 하용호, 자습해도 모르겠던 딥러닝, 머리속에 인스톨 시켜드립니다

1. GD(Gradient Descent) : 학습 데이터를 다 입력하고 한걸음 이동하는 경사하강법

P.132 그림 4-10 참고

P.194 그림 6-3 참고

GD의 단점 : 학습 데이터를 신경망에 다 입력해서 한 걸음 이동하므로 시간이 많이 걸린다.

2. SGD(Stochastic Gradient Descent) : GD의 단점을 개선한 경사 하강법, Stochastic(확률적)경사하강법으로 복원추출 미니배치하여 경사가 기울어진 방향으로 학습해 나가는 방법

P.194 그림 6-3 참고

문제112. 5장 전체 코드(텐서플로우 2.0으로 구현)의 경사하강법을 SGD로 구현해서 수행하고 테스트 데이터 정확도와 시각화를 확인하시오.

6. 신경망 모델을 설정한다.

```
model.compile(optimizer='SGD',
              loss = 'categorical_crossentropy',
              metrics=['acc'])
```

SGD의 단점 : Local minimum에 잘 빠진다.

3. Momentum : 관성을 이용해서 local minima에서 빠져나갈 수 있도록 경사하강하는 방법

P.196 그림 6-5 참고

모멘텀은 tensorflow2.0에는 따로 없지만 Adam과 같은 다른 경사하강법 종류에 모멘텀의 장점이 구현 되어져 있다.

4. Adagrad : Learning Rate(학습률)를 자동조절되게 하는 경사하강법 처음에는 학습률을 크게하고 나중에는 학습률을 작게하여 점차 줄여가는 경사하강법으로 각각의 매개변수 (W_1, W_2, W_3)에 맞춤형 값을 만들어준다.

P.198 그림 6-6 참고

문제113. Adagrad로 3층 신경망의 경사하강법을 변경하고 정확도를 확인하시오.

6. 신경망 모델을 설정한다.

```
model.compile(optimizer='Adagrad',
              loss = 'categorical_crossentropy',
              metrics=['acc'])
```

5. Adam : momentum의 장점(관성) + Adagrade의 장점(러닝 레이트 자동 조절)을 살린 경사 하강법

문제114. 3층 신경망의 경사하강법을 Adam으로 하고 학습 시키시오.

6. 신경망 모델을 설정한다.

```
model.compile(optimizer='Adam',
              loss = 'categorical_crossentropy',
              metrics=['acc'])
```

6. RMSprop : Adagrad의 단점을 살려서 목표지점에 도달할 때 이전 맥락을 살펴서 러닝레이트를 조절한다.

문제115. RMSprop으로 경사하강법을 변경하여 정확도를 출력하시오.

6. 신경망 모델을 설정한다.

```
model.compile(optimizer='RMSprop',
              loss = 'categorical_crossentropy',
              metrics=['acc'])
```

가중치 초기값 설정(P.202)

랜덤으로 생성되는 가중치 W의 초기값을 어떻게 선정하느냐에 따라 학습이 잘 될 수도 있고 잘 되지 않을 수도 있다. 학습이 잘 이루어지려면 가중치 초기값들의 분포가 정규분포 형태를 이루어야 한다.

그런데 np.random.randn은 가우시안 정규분포(평균이 0이고 표준편차가 1)를 따르는 난수를 생성한다.

<https://cafe.daum.net/oracleoracle/Sedp/238>

1. 생성되는 가중치의 표준편차가 너무 큰 경우

- 시험문제가 너무 어려우면 아주 잘하는 학생들과 아주 못하는 학생들로 점수가 나뉜다.

1 * np.random.randn(784, 50) # W1 생성

P.204 그림 6-10 참고

2. 생성되는 가중치의 표준편차가 너무 작은 경우
 - 시험문제가 너무 쉬우면 학생들의 점수가 평균에 가까워진다.

0.01 * np.random.randn(784, 50)

P.205 그림 6-11 참고

다른 표준편차를 지정하는 방법 2가지

1. Xavier 초기값 설정 : Sigmoid함수와 사용
2. He 초기값 설정 : Relu 함수와 사용

파이썬 날코딩(He 초기값)

```
def __init__(self, input_size, hidden_size, output_size, weight_init_std=0.01):  
    # 가중치 초기화  
    self.params = {}  
    self.params['W1'] = np.sqrt(2/input_size) * np.random.randn(input_size, hidden_size)  
    self.params['b1'] = np.zeros(hidden_size)  
    self.params['W2'] = np.sqrt(2/input_size) * np.random.randn(hidden_size, output_size)  
    self.params['b2'] = np.zeros(output_size)
```

텐서플로우 2.0 (He 초기값)

```
# 5. 신경망 모델을 구성한다.  
model = Sequential()  
initializer = tf.keras.initializers.GlorotNormal()  
model.add(Dense(100, activation = 'relu', kernel_initializer = initializer, input_shape = (784, )))  
model.add(Dense(50, activation = 'relu', kernel_initializer = initializer))  
model.add(Dense(10, activation = 'softmax', kernel_initializer = initializer))
```

배치 정규화(batch normalization)

<https://cafe.daum.net/oracleoracle/Sedp/239>

가중치 초기화 값을 적절히 설정하면 각 층의 활성화 값의 분포가 적당히 퍼지는 효과를 보이는데 신경망이 깊어지고 학습이 반복되다 보면 각 층의 활성화 값의 분포가 정규성을 잃어버리는 현상이 발생하게 된다. 그래서 이 정규성을 계속해서 유지 시키도록 '강제화' 하는 방법이 배치정규화이다.

텐서플로우 2.0

```

# 1. 필요한 패키지 import
import tensorflow as tf
from tensorflow.keras.datasets.mnist import load_data
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, BatchNormalization
from tensorflow.keras.utils import to_categorical

# 5. 신경망 모델을 구성한다.
model = Sequential()
initializer = tf.keras.initializers.GlorotNormal()
model.add(Dense(100, activation = 'relu', kernel_initializer = initializer, input_shape = (784, )))
model.add(BatchNormalization())
model.add(Dense(50, activation = 'relu', kernel_initializer = initializer))
model.add(BatchNormalization())
model.add(Dense(10, activation = 'softmax', kernel_initializer = initializer))

```

overfitting을 방지하는 방법

드롭아웃

L2 정규화

얼리스탑 (keras의 기능)

드롭아웃(dropout)

오버피팅을 억제하기 위해서 뉴런을 임의로 삭제하면서 학습시키는 방법

P.219 그림 6-22 참고

텐서플로우 2.0

```

# 1. 필요한 패키지 import
import tensorflow as tf
from tensorflow.keras.datasets.mnist import load_data
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, BatchNormalization, Dropout
from tensorflow.keras.utils import to_categorical
from sklearn.model_selection import train_test_split

# 5. 신경망 모델을 구성한다.
model = Sequential()
initializer = tf.keras.initializers.GlorotNormal()
model.add(Dense(100, activation = 'relu', kernel_initializer = initializer, input_shape = (784, )))
model.add(BatchNormalization())
model.add(Dropout(0.2))
model.add(Dense(50, activation = 'relu', kernel_initializer = initializer))
model.add(BatchNormalization())
model.add(Dropout(0.2))
model.add(Dense(10, activation = 'softmax', kernel_initializer = initializer))

```

얼리스탑

1. tensorflow + keras
2. pytorch

훈련 데이터의 정확도와 validation 데이터의 정확도가 같아지는 부분에서 멈춘다.

<https://cafe.daum.net/oracleoracle/SgRM/229>

텐서플로우 2.0

7. 신경망 모델을 훈련시킨다.

```
from tensorflow.python.keras.callbacks import EarlyStopping

early_stopping = EarlyStopping()

history = model.fit(x_train, y_train,
                     epochs = 30,
                     batch_size = 100,
                     validation_data = (x_val, y_val),
                     callbacks=[early_stopping])
```

L2 정규화(가중치 감소)

학습하는 과정에서 큰 가중치에 대해서는 그에 상응하는 큰 패널티를 부여하여 오버피팅을 억제하는 방법

텐서플로우 2.0

5. 신경망 모델을 구성한다.

```
model = Sequential()
initializer = tf.keras.initializers.GlorotNormal()
model.add(Dense(100, kernel_regularizer='l2', activation = 'relu',
               kernel_initializer = initializer, input_shape = (784, )))
model.add(BatchNormalization())
model.add(Dropout(0.2))

model.add(Dense(50, activation = 'relu', kernel_initializer = initializer))
model.add(BatchNormalization())
model.add(Dropout(0.2))

model.add(Dense(10, activation = 'softmax', kernel_initializer = initializer))
```

★ 6장 내용 정리

1. 언더피팅을 막는 방법
 - a. 고급 경사 감소법
 - b. 가중치 초기값 설정
 - c. 배치 정규화

2. 오버피팅을 막는 방법

- a. 드롭아웃
- b. 엘리스탭
- c. L2 정규화

포토폴리오 준비

1. 스크롤링한 사진
2. 신경망 코드
3. 신경망에 사진을 로드하는 코드

예제1. test에서 사진을 불러오시오.

```
import os

test_image = 'C:\\\\Users\\\\KJM\\\\test'

def image_load(path):
    file_list = os.listdir(path)
    return file_list

print (image_load(test_image))
```

예제2. 위의 결과에서 .jpg는 빼고 숫자만 출력되게 하시오.

```
import os
import re

test_image = 'C:\\\\Users\\\\KJM\\\\test'

def image_load(path):
    file_list = os.listdir(path)
    file_name = []
    for i in file_list:
        a = int(re.sub('[^0-9]', "", i)) # i가 숫자가 아니면 null로 변경
        file_name.append(a)

    return file_name

print (image_load(test_image))
```

문제116. 위의 결과가 정렬되서 출력되게 하시오.

```
import os
import re

test_image = 'C:\\\\Users\\\\KJM\\\\test'
```

```
def image_load(path):
    file_list = os.listdir(path)
    file_name = []
    for i in file_list:
        a = int(re.sub('[^0-9]', "", i)) # i가 숫자가 아니면 null로 변경
        file_name.append(a)

    return sorted(file_name)

print (image_load(test_image))
```

언더피팅과 오버피팅을 줄이는 방법

1. 언더피팅을 줄이는 방법

- 고급 경사감소법 (SGD, momentum, Adagrade, Adam)
- 가중치 초기값 설정
- 배치 정규화 : 층이 깊어져도 중치가 계속 정규분포를 유지할 수 있도록 잡아주는 역할

2. 오버피팅을 줄이는 방법

- 드롭아웃
- L2
- 얼리스탑

예제4. 문제116의 결과에서 jpg 가 붙어서 출력되게 하시오.

```
import os
import re

test_image = 'C:\\\\Users\\\\KJM\\\\test'

def image_load(path):
    file_list = os.listdir(path)
    file_name = []
    for i in file_list:
        a = int(re.sub('[^0-9]', "", i)) # i가 숫자가 아니면 null로 변경
        file_name.append(a)
    file_name.sort()

    file_res=[]
    for j in file_name:
        file_res.append('%d.jpg' %j)

    return file_res

print (image_load(test_image))
```

예제5. 이미지의 이름 앞에 절대경로가 붙게 하시오.

```
import os
import re

test_image = 'C:\\\\Users\\\\KJM\\\\test'
```

```

def image_load(path):
    file_list = os.listdir(path)
    file_name = []
    for i in file_list:
        a = int(re.sub('[^0-9]', "", i)) # i가 숫자가 아니면 null로 변경
        file_name.append(a)
        file_name.sort()

    file_res=[]
    for j in file_name:
        file_res.append('%s\\%d.jpg' %(path,j))

    return file_res

print (image_load(test_image))

```

예제6. opencv를 이용해서 이미지를 numpy array 형태의 숫자로 변환하시오.

```

import os
import re
import cv2
import numpy as np

test_image = 'C:\\Users\\KJM\\test'

def image_load(path):
    file_list = os.listdir(path)
    file_name = []
    for i in file_list:
        a = int(re.sub('[^0-9]', "", i)) # i가 숫자가 아니면 null로 변경
        file_name.append(a)
        file_name.sort()

    file_res = []
    for j in file_name:
        file_res.append('%s\\%d.jpg' %(path,j))

    image = []
    for k in file_res:
        img = cv2.imread(k)
        image.append(img)

    return np.array(image)

print (image_load(test_image))

```

신경망에 이미지를 넣으려면 2개의 함수가 있어야 한다.

1. image_load : 이미지를 숫자로 변환해주는 함수
2. label_load : 정답 라벨을 one hot encoding 해주는 함수

예제7. 훈련 데이터의 라벨을 train_label.csv로 생성하시오.

(1~9500 을 1로 하고 9501~1900을 0으로 하시오.)

```
path = 'C:\\\\Users\\\\KJM\\\\train_label.csv'

file = open (path, 'w')

for i in range(0, 9500):
    file.write (str(1) + '\\n')
for i in range (0, 9500):
    file.write (str(0) + '\\n')

file.close()
```

예제8. 테스트 데이터의 라벨을 test_label.csv로 생성하시오.

```
path = 'C:\\\\Users\\\\KJM\\\\test_label.csv'

file = open (path, 'w')

for i in range(0, 500):
    file.write (str(1) + '\\n')
for i in range (0, 500):
    file.write (str(0) + '\\n')

file.close()
```

예제9. train_label.csv의 내용을 불러오는 함수를 생성하시오.

```
import csv

train_label = 'C:\\\\Users\\\\KJM\\\\train_label.csv'

def label_load(path):
    file = open (path)
    labldata = csv.reader(file)
    labellist = []
    for i in labldata:
        labellist.append(i)

    return labellist

print (label_load(train_label))
```

예제10. 위의 결과가 숫자로 출력되게 하시오.

```
import csv

train_label = 'C:\\\\Users\\\\KJM\\\\train_label.csv'

def label_load(path):
    file = open (path)
    labldata = csv.reader(file)
    labellist = []
    for i in labldata:
        labellist.append(i)

    return labellist
```

```

labellist.append(i)
label = np.array(labellist) # 리스트를 numpy array로 변환
label = label.astype(int) # 숫자 변환

return label

print (label_load(train_label))

```

예제11. 위의 숫자를 **one hot encoding** 하기 위해 아래의 코드를 연습하시오.

```

import numpy as np

print (np.eye(10)[4]) # 숫자 4가 one hot encoding 된다.

```

예제12. **label_load** 함수의 결과가 **one hot encoding** ← 결과로 출력되게 하시오.

```

import csv
import numpy as np

train_label = 'C:\\\\Users\\\\KJM\\\\train_label.csv'

def label_load(path):
    file = open (path)
    labldata = csv.reader(file)
    labellist = []
    for i in labldata:
        labellist.append(i)
    label = np.array(labellist) # 리스트를 numpy array로 변환
    label = label.astype(int) # 숫자 변환
    label = np.eye(2)[label]

    return label

print (label_load(train_label))

```

예제13. 위의 결과는 3차원으로 출력되는데 신경망에서 라벨로 사용하려면 2차원이여야 하므로 차원을 2차원으로 축소하시오.

```

import csv
import numpy as np

train_label = 'C:\\\\Users\\\\KJM\\\\train_label.csv'

def label_load(path):
    file = open (path)
    labldata = csv.reader(file)
    labellist = []
    for i in labldata:
        labellist.append(i)
    label = np.array(labellist) # 리스트를 numpy array로 변환

```

```

label = label.astype(int) # 숫자 변환
label = np.eye(2)[label] # one hot encoding
label = label.reshape(-1, 2) # 차원 축소

return label

print (label_load(train_label))

```

예제14. 위에서 만든 함수2개를 **loader_leaf.py**로 저장하시오.

예제15. 훈련 데이터를 256x256에서 32x32로 resize 하시오.

```

import loader_leaf

train_image = 'C:\\\\Users\\\\KJM\\\\train'
test_image = 'C:\\\\Users\\\\KJM\\\\test'
train_label = 'C:\\\\Users\\\\KJM\\\\train_label.csv'
test_label = 'C:\\\\Users\\\\KJM\\\\test_label.csv'

print (loader_leaf.image_load(train_image).shape)
print (loader_leaf.image_load(test_image).shape)
print (loader_leaf.label_load(train_label).shape)
print (loader_leaf.label_load(test_label).shape)

import cv2
import os
import numpy as np

path = 'C:\\\\Users\\\\KJM\\\\train'
file_list = os.listdir(path) # path에 지정된 위치에 있는 파일들의 이름을 불러온다.
file_list # ['b.jpg']

import numpy as np

for k in file_list: # 리스트 안에 있는 파일들을 하나씩 빼내는 코드
    img = cv2.imread(path + '\\\\' + k) # 수지 사진을 숫자행렬로 변경합니다.
    #
    width, height = img.shape[2] # 수지 사진 숫자 행렬에서 가로, 세로 가져온다.
    resize_img = cv2.resize(img, (32 , 32), interpolation=cv2.INTER_CUBIC)
    cv2.imwrite('C:\\\\Users\\\\KJM\\\\train\\\\train_resize\\\\' + k, resize_img) # resize 한 이미지를 저장합니다.

plt.imshow(resize_img) # resize 한 수지 사진을 시각화 해라 ~
plt.show()

```

예제16. 테스트 데이터를 256x256에서 32x32로 resize 하시오.

```

import cv2
import os
import numpy as np

path = 'C:\\\\Users\\\\KJM\\\\test'
file_list = os.listdir(path) # path에 지정된 위치에 있는 파일들의 이름을 불러온다.
file_list # ['b.jpg']

import numpy as np

for k in file_list: # 리스트 안에 있는 파일들을 하나씩 빼내는 코드
    img = cv2.imread(path + '\\\\' + k) # 수지 사진을 숫자행렬로 변경합니다.
    #
    width, height = img.shape[:2] # 수지 사진 숫자 행렬에서 가로, 세로 가져온다.
    resize_img = cv2.resize(img, (32 , 32), interpolation=cv2.INTER_CUBIC)
    cv2.imwrite('C:\\\\Users\\\\KJM\\\\test\\\\test_resize\\\\' + k, resize_img) # resize 한 이미지를 저장합니다.

plt.imshow(resize_img) # resize 한 수지 사진을 시각화 해라 ~
plt.show()

```

CNN

합성곱 신경망 : convolution 층과 pooling 층을 포함하는 신경망

기존방법 : 사진 그대로 입력하지 않고 1차원으로 flatten 시켜서 입력

CNN : 사진을 그대로 입력

CNN을 이용하지 않았을 때의 기존층의 문제점

형상을 무시하고 모든 입력 데이터를 동등한 뉴런으로 취급하기 때문에 이미지가 갖는 본질적인 패턴을 읽지 못한다.

<https://cafe.daum.net/oracleoracle/Sedp/199>

해결방법

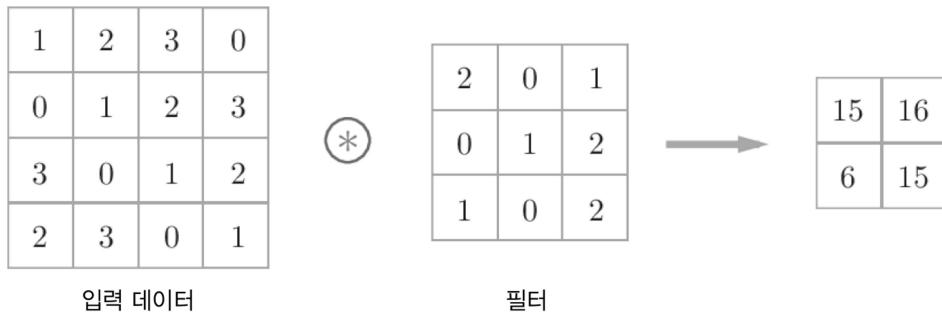
원본 이미지를 가지고 여러개의 feature map을 만들어서 데이터를 여러 개 생성한다.

합성곱 연산을 컴퓨터로 구현하는 방법

원본 이미지 한장을 filter 100개로 합성곱 연산을 하면 feature map이 100개가 생성된다.

합성곱의 역할 : 이미지의 특징을 잡아내는 역할을 한다.

문제117. 아래의 그림의 합성곱을 수행하기 위한 입력 데이터와 필터를 만드시오.

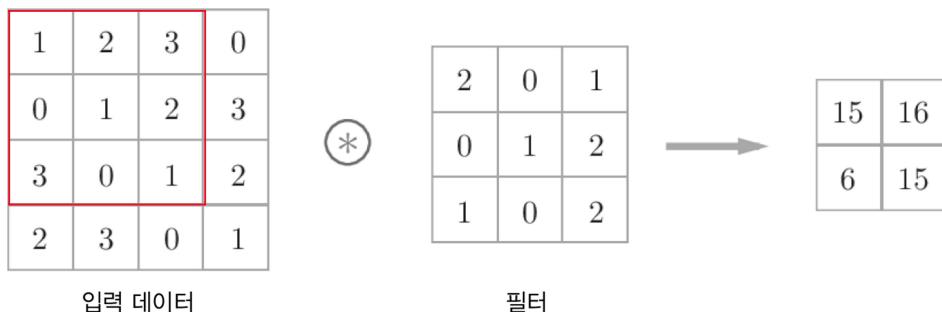


```
import numpy as np

x = np.array([[1, 2, 3, 0], [0, 1, 2, 3], [3, 0, 1, 2], [2, 3, 0, 1]])
ft = np.array([2, 0, 1, 0, 1, 2, 1, 0, 2]).reshape(3, 3)

print(x)
print(ft)
```

문제118. 입력 이미지 x에서 아래의 3x3 영역만 가지고 오시오.



```
import numpy as np

x = np.array([[1, 2, 3, 0], [0, 1, 2, 3], [3, 0, 1, 2], [2, 3, 0, 1]])
ft = np.array([2, 0, 1, 0, 1, 2, 1, 0, 2]).reshape(3, 3)

print(x[0:3, 0:3])
```

문제119. 위에서 가져온 영역의 원소들과 filter의 원소들의 곱셈을 하시오.

```
import numpy as np

x = np.array([[1, 2, 3, 0], [0, 1, 2, 3], [3, 0, 1, 2], [2, 3, 0, 1]])
ft = np.array([2, 0, 1, 0, 1, 2, 1, 0, 2]).reshape(3, 3)
```

```
print (x[0:3, 0:3] * ft)
```

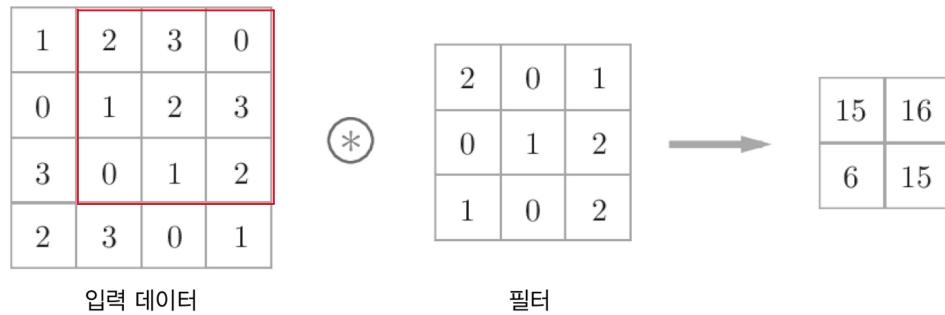
문제120. 위의 곱셈을 한 결과의 원소들의 합을 구하시오.

```
import numpy as np

x = np.array([[1, 2, 3, 0], [0, 1, 2, 3], [3, 0, 1, 2], [2, 3, 0, 1]])
ft = np.array([2, 0, 1, 0, 1, 2, 1, 0, 2]).reshape(3, 3)

print ((x[0:3] * ft).sum())
```

문제121. 원본 이미지에서 아래의 영역과 필터를 곱한 원소들의 합을 구하시오.



```
import numpy as np
```

```
x = np.array([[1, 2, 3, 0], [0, 1, 2, 3], [3, 0, 1, 2], [2, 3, 0, 1]])
ft = np.array([2, 0, 1, 0, 1, 2, 1, 0, 2]).reshape(3, 3)
```

```
print ((x[0:3, 1:4] * ft).sum())
```

문제122. 결과적으로 나와야하는 4개의 숫자를 모두 출력하시오.

```
import numpy as np
```

```
x = np.array([[1, 2, 3, 0], [0, 1, 2, 3], [3, 0, 1, 2], [2, 3, 0, 1]])
ft = np.array([2, 0, 1, 0, 1, 2, 1, 0, 2]).reshape(3, 3)
```

```
print ((x[0:3, 0:3] * ft).sum())
print ((x[0:3, 1:4] * ft).sum())
print ((x[1:4, 0:3] * ft).sum())
print ((x[1:4, 1:4] * ft).sum())
```

문제123. 위의 4개의 값을 하드코딩하지 않고 이중 루프문으로 수행하시오.

```
import numpy as np
```

```

x = np.array([[1, 2, 3, 0], [0, 1, 2, 3], [3, 0, 1, 2], [2, 3, 0, 1]])
ft = np.array([2, 0, 1, 0, 1, 2, 1, 0, 2]).reshape(3, 3)

for i in range (2):
    for j in range (2):
        print ((x[i : 3+i, j: 3+j] * ft).sum())

```

문제124. 위에서 출력된 숫자를 a라는 비어있는 리스트에 append시키시오.

```

import numpy as np

x = np.array([[1, 2, 3, 0], [0, 1, 2, 3], [3, 0, 1, 2], [2, 3, 0, 1]])
ft = np.array([2, 0, 1, 0, 1, 2, 1, 0, 2]).reshape(3, 3)
a = []

for i in range (2):
    for j in range (2):
        a.append ((x[i : 3+i, j: 3+j] * ft).sum())

print (a)

```

문제125. 위의 a 리스트의 shape를 2x2로 변경하시오.

```

import numpy as np

x = np.array([[1, 2, 3, 0], [0, 1, 2, 3], [3, 0, 1, 2], [2, 3, 0, 1]])
ft = np.array([2, 0, 1, 0, 1, 2, 1, 0, 2]).reshape(3, 3)
a = []

for i in range (2):
    for j in range (2):
        a.append ((x[i : 3+i, j: 3+j] * ft).sum())

b = np.array(a).reshape(2, 2)

print (b)

```

문제126. 위의 결과에서 편향 3을 더하시오.

```

import numpy as np

x = np.array([[1, 2, 3, 0], [0, 1, 2, 3], [3, 0, 1, 2], [2, 3, 0, 1]])
ft = np.array([2, 0, 1, 0, 1, 2, 1, 0, 2]).reshape(3, 3)
a = []

for i in range (2):
    for j in range (2):
        a.append ((x[i : 3+i, j: 3+j] * ft).sum())

```

```
b = np.array(a).reshape(2, 2)
```

```
print (b + 3)
```

3차원 합성곱

<https://cafe.daum.net/oracleoracle/Sedp/282>

<https://cafe.daum.net/oracleoracle/Sedp/326>

문제127. 3차원 합성곱을 하기전에 x2 입력 이미지에서 red 행렬만 출력하시오.

```
import numpy as np
```

```
x2 = np.array([[[1,2,3,0], # --> red 행렬  
               [0,1,2,3],  
               [3,0,1,2],  
               [2,3,0,1]],
```

```
[[2,3,4,1], # --> green 행렬  
 [1,2,3,4],  
 [4,1,2,3],  
 [3,4,1,2]],
```

```
[[3,4,5,2], # --> blue 행렬  
 [2,3,4,5],  
 [5,2,3,4],  
 [4,5,2,3]]])
```

```
f2 = np.array([[[2,0,1],  
                [0,1,2],  
                [1,0,2]],
```

```
[[3,1,2],  
 [1,2,3],  
 [2,1,3]],
```

```
[[4,2,3],  
 [2,3,4],  
 [3,2,4]]])
```

```
print (x2[0])
```

문제128. 3차원 합성곱을 진행하시오.

```
import numpy as np
```

```

x2 = np.array([[[1,2,3,0], # --> red 행렬
               [0,1,2,3],
               [3,0,1,2],
               [2,3,0,1]],

               [[2,3,4,1], # --> green 행렬
                [1,2,3,4],
                [4,1,2,3],
                [3,4,1,2]],

               [[3,4,5,2], # --> blue 행렬
                [2,3,4,5],
                [5,2,3,4],
                [4,5,2,3]]])

f2 = np.array([[[[2,0,1],
                 [0,1,2],
                 [1,0,2]],

                 [[3,1,2],
                  [1,2,3],
                  [2,1,3]],

                 [[4,2,3],
                  [2,3,4],
                  [3,2,4]]]])

a = []

for k in range(3):
    for i in range(2):
        for j in range(2):
            a.append ((x2[k, i : 3+i, j: 3+j] * f2[k]).sum())

b = np.array(a).reshape(3, 4)

c = np.sum(b, axis = 0)

print (c.reshape(2, 2))

```

3차원 원본 이미지에서 filter를 거쳐 2차원 feature map을 생성한다.

문제129. 유럽.png를 아래의 3차원 필터로 합성곱 하기 위하여 유럽.png를 numpy array 형태의 숫자로 변환 하시오.

<https://cafe.daum.net/oracleoracle/Sedp/358>

```

from PIL import Image
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

```

```
img = Image.open ('유럽.png')
img_pixel = np.array(img)

print (img_pixel.shape)
```

문제130. 유럽 이미지랑 합성곱할 필터를 아래와 같이 랜덤으로 생성하시오.

```
ft3 = np.random.rand(5, 5, 3)

print (ft3)

plt.imshow(ft3)
plt.show()
```

문제131. 문제 128번에서 사용한 3차원 합성곱 코드를 가져와서 3 차원 유럽 사진에 3차원 필터와 합성곱 하여 feature map을 생성하시오.

```
row = 499 - 5 + 1
col = 756 - 5 + 1

a = []

for i in range(row):
    for j in range(col):
        b=0
        for k in range(3):
            b += np.sum(img_pixel [i: 5 + i, j:5+j, k] * ft3[:, :, k] )
        a.append(b)

c = np.array(a).reshape(495,752)
```

문제132. 유럽사진의 피쳐맵 시각화를 올리시오.

```
row = 499 - 5 + 1
col = 756 - 5 + 1

a = []

for i in range(row):
    for j in range(col):
        b=0
        for k in range(3):
            b += np.sum(img_pixel [i: 5 + i, j:5+j, k] * ft3[:, :, k] )
        a.append(b)
```

```
c = np.array(a).reshape(495,752)
```

```
plt.imshow(c)
```

21.03.18

2021년 3월 18일 목요일 오전 9:43

패딩

패딩이란 합성곱 연산을 수행하기 전에 입력 데이터 주변을 특정값으로 채워 늘리는 것

패딩을 하지 않을 경우 data의 공간 크기는 합성곱 계층을 지날 때마다 작아지게 되므로 가장 자리 정보들이 사라지게 되는 문제가 발생하게 된다.

예제1

아래의 2x2 행렬을 제로패딩 1해서 4x4 행렬로 만드시오.

```
[[18 19]
 [ 9 18]]
```

```
import numpy as np

x = np.array ([[18, 19], [9, 18]])

x_pad = np.pad (x, pad_width=1, mode='constant', constant_values=0)

print (x_pad)
```

문제133. 아래의 행렬을 제로패딩 1한 후의 결과를 확인하시오.

```
import numpy as np

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

x_pad = np.pad (x, pad_width=1, mode='constant', constant_values=0)

print (x_pad)
```

패딩을 사용 목적

원본 이미지가 convolution 층을 지날 때마다 같은 형상으로 유지 시키는 것

원본 입력 이미지와 같은 형상으로 피쳐맵이 출력되기 위한 패딩 값

파이썬 날코딩 : padding = (직접 계산)

텐서플로우 코드 : padding = same

계산 공식

$$OH = \frac{H + 2P - FH}{S} + 1$$

$$OW = \frac{W + 2P - FW}{S} + 1$$

스트라이드

합성곱할 때 원본 이미지를 필터로 이동하면서 피쳐맵을 생성하는 과정에서 이동을 몇칸으로 할지 결정하는 것

텐서플로우 2.0 구현

```
model.add(Conv2D(32, (3, 3), padding='same', input_shape=x_train.shape[1:]))
```

32 : 뉴런의 개수

(3, 3) : 필터 사이즈(가로, 세로)

padding='same' : 입력이미지와 출력이미지의 사지으로 같게 한다.

input_shape=x_train.shape[1:] : 입력층의 뉴런의 갯수

파이썬 날코딩 (P.251)

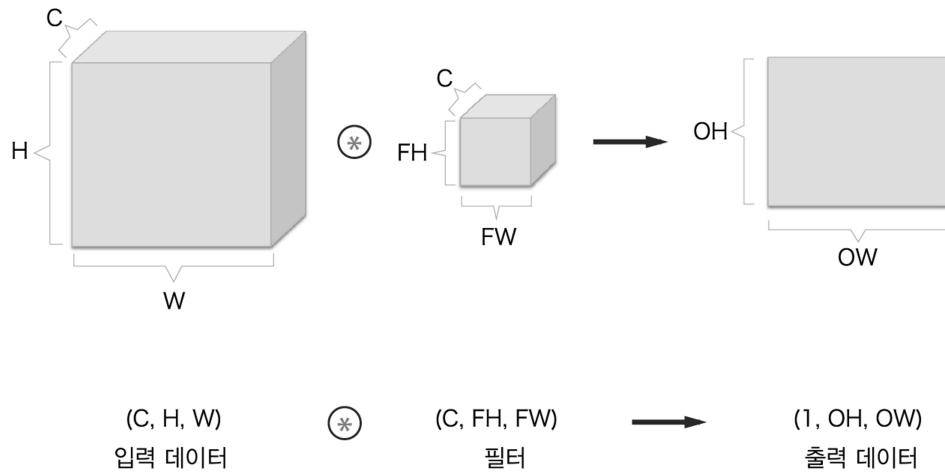
```
conv_params = {'filter_num':30, 'filter_size':5, 'pad':0, 'stride':1}
```

컴볼루션 층에서 하는 작업은 원본이미지를 가지고 여러개의 비슷한 이미지를 만들어내는 작업이다. 비슷한 이미지들이 바로 feature map이다. feature map의 갯수는 필터 갯수와 동일하게 생성된다.

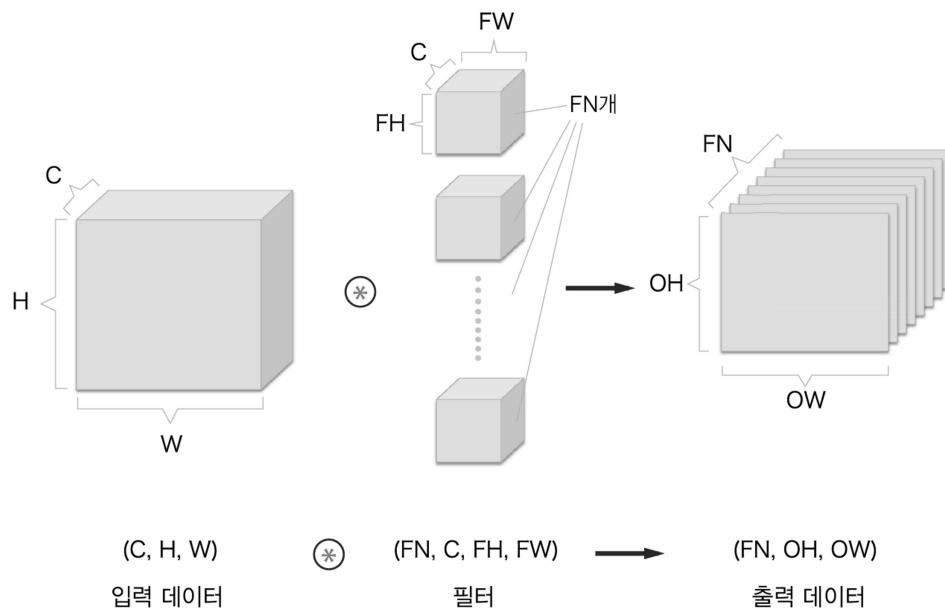
블럭으로 생각하기(P.237)

3차원 합성곱 연산은 데이터와 필터를 직육면체 블럭으로 생각하면 쉽다.

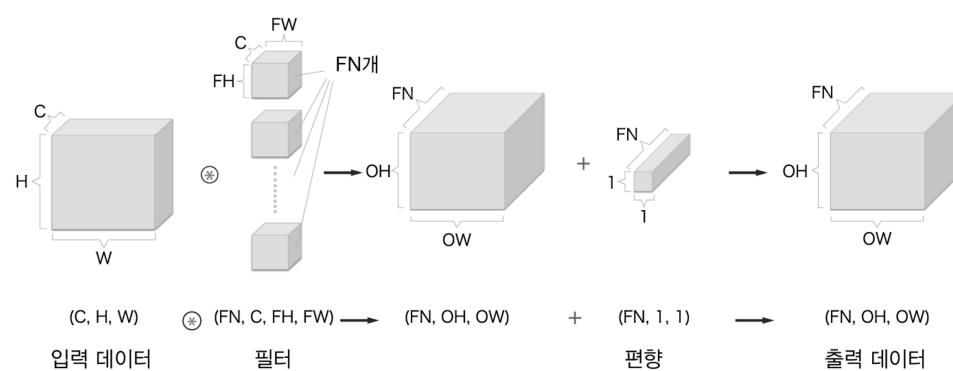
블럭은 아래의 그림과 같은 3차원 직육면체이다.



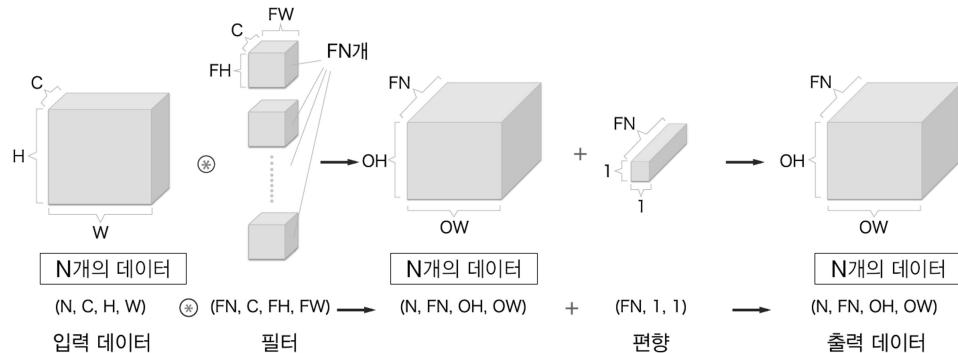
설현 사진 RGB컬러 사진 1장을 RGB필터로 합성곱해서 1개의 feature map을 출력하는 그림



설현 사진 1장에 필터 FN개를 합성곱하고 FN개의 피쳐맵을 출력하는 그림



설현 사진 1장에 필터 FN개를 합성곱하고 FN개의 편향을 더해서 FN개의 피쳐맵을 출력하는 그림



미니배치의 개수 N개 만큼 설현사진을 입력해서 필터 FN개와 합성곱하여 미니배치 개수 N개 만큼 피쳐맵을 출력하는 그림

RGB설현 사진 1장을 100개의 RGB필터로 합성곱하면 100개의 피쳐맵이 생기는데 100개의 설현 사진이 100개의 RGB 필터와 각각 합성곱하면 피쳐맵이 10000장이 된다.

합성곱 계층을 이해하기 위한 im2col 함수를 사용하는 단계

예제3. 설현 사진 10장을 im2col 함수에 넣어서 2차원 행렬로 변환하시오.

<https://cafe.daum.net/oracleoracle/Sedp/351>

```
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

import numpy as np

x1 = np.random.rand(10, 3, 7, 7)
col = im2col(x1, 5, 5, stride=1, pad=0)
print(col.shape) # (90, 75)

```

im2col 함수 : 신경망에서 합성곱을 진행하는데 입력되는 4차원 데이터를 2차원 데이터로 차원 축소해서 2차원 필터와 내적해서 합성곱하게 하는 함수

<https://cafe.daum.net/oracleoracle/SgRM/264>

```

model.add(Conv2D(32, (3, 3)))
# 3x3 의 필터 32개를 원본 이미지(100, 3, 32, 32)에 합성곱하여 32개의 피쳐맵 생성

```

4차원 필터를 2차원으로 변경하는 방법

예제1. 아래의 5x5 행렬의 RGB필터 10개를 3차원으로 변경하시오.

(10, 3, 5, 5)

```

import numpy as np

filter = np.random.rand(10, 3, 5, 5) # 4차원 필터 생성
print (filter.reshape(10, 3, -1).shape)

```

예제2. 위의 4차원 행렬을 2차원 행렬로 변경하시오.

```

import numpy as np

filter = np.random.rand(10, 3, 5, 5) # 4차원 필터 생성
print (filter.reshape(10, -1).shape)

```

원본 이미지를 2차원으로 변경 : im2col

필터를 2차원으로 변경 : reshape의 -1

풀링(pooling)층의 역할

원본 이미지 > Conv > Relu > Pooling > fully connected(완전 연결 계층)

Conv : 원본 이미지의 특징을 잡아내는 피쳐맵을 필터의 갯수가 맞춰 생성하는 층

Pooling : convolution 층이 망친 그림들을 피쳐맵 이미지의 각 부분에서 대표값들을 뽑아 사이즈가 작은 이미지를 만드는 역할 (이미지가 선명해진다.)

```
model.add(Conv2D(32, (3, 3)))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
```

풀링층의 역할 : convolution층이 이미지의 특징을 잡아내는 역할을 한다면 pooling층은 feature map 이미지를 선명하게 만드는 역할을 한다.

풀링의 종류 3가지

1. 최대 풀링 : convolution 데이터에서 가장 큰값을 대표값으로 선정
 - 이미지를 선명하게 만드는 효과
2. 평균 풀링 : convolution 데이터에서 모든값의 평균값을 대표값으로 선정
 - 이미지를 부드럽게 하는 효과
3. 확률적 풀링 : convolution 데이터에서 임의 확률로 한개를 선정

<https://cafe.daum.net/oracleoracle/Sedp/349>

예제1. 파이썬으로 아래의 행렬을 만드시오.

```
[[21  8  8 12]
 [12 19  9  7]
 [ 8 10  4  3]
 [18 12  9 10]]
```

```
import numpy as np
```

```
x = np.array([21, 8, 8, 12, 19, 9, 7, 8, 10, 4, 3, 18, 12, 9, 10]).reshape(4, 4)
```

```
print (x)
```

예제2. 위의 행렬에서 max pooling을 시도해서 아래의 결과를 출력하시오.

```
import numpy as np

x = np.array([21, 8, 8, 12, 12, 19, 9, 7, 8, 10, 4, 3, 18, 12, 9, 10]).reshape(4, 4)
x2 = x.reshape(1, 1, 4, 4)
a = im2col(x2, 2, 2, 2, 0) # (x2, 필터가로, 필터세로, 스트라이드, 패딩)
d = np.max(a, axis=1)

print (d.reshape(2, 2))
```

설현과 수지 사진 분류 신경망 만들기

<https://cafe.daum.net/oracleoracle/SgRM/268>

21.03.22

2021년 3월 22일 월요일 오전 9:55

구글 코랩을 이용해서 GPU를 사용하여 이파리 데이터 분류하기

1. 구글 코랩 가입
2. 새 노트를 열어 GPU를 사용할 수 있도록 설정
3. 카페에서 2개의 파일을 내려받는다.
 - <https://cafe.daum.net/oracleoracle/SgRM/274>
4. 이파리분류.ipynb 파일을 코랩에서 연다.

코랩에서 터미널창 열기

```
!pip install kora
from kora import console
console.start()
```

신경망 코드

<https://cafe.daum.net/oracleoracle/SgRM/280>



tensorflo...
신경망



fashion_...



fashion_...

문제. 아래의 코드를 참고해서 배치정규화 코드를 삽입하고 에폭을 100
에폭으로 돌리시오.

```
model.add(Conv2D(32, (3, 3), padding='same', input_shape=x_train.shape[1:]))
model.add(BatchNormalization())
model.add(Activation('relu'))
```

21.03.23

2021년 3월 23일 화요일 오전 9:46

딥러닝 복습

1. numpy 사용법
2. 퍼셉트론
3. 3층 신경망 구현 (학습x)
4. 2층 신경망 구현 (학습o) - 수치미분을 이용
5. 2층 신경망 구현 (학습o) - 오차역전파를 이용
6. 언더피팅과 오버피팅을 방지하는 방법
7. CNN층
8. 딥러닝 역사를 텐서플로우 2.0을 이용해서 구현

딥러닝을 발전 시킨 신경망

1. CNN을 사용한 신경망 구현 (fashion mnist)
2. VGG 신경망

신경망 사용시 구현한 기타 기능들

1. 이미지 증식시키기
2. 케라스의 콜백 기능

CNN을 사용한 신경망 구현 (제규어와 얼룩말사진을 분류)

1. 제규어와 얼룩말 데이터 2000장을 받는다.
2. GPU를 사용하는지 확인한다.
3. drive를 마운트 시킨다.
4. train_resize2.zip 과 test_resize2.zip을 /content/gdrive 밑에 올린다.
5. 얼룩말과 제규어 사진을 가지고 홈페이지 구현을 위한 신경망 모델 생성하는 코드 수행
 - <https://cafe.daum.net/oracleoracle/SgRM/285>

tensorflow 2.0 장점

1. 즉시 실행모드로 실행할 수 있다. (파이썬 처럼 실행이 가능하다.)
2. 유명한 신경망의 모델설계와 가중치 파일을 분류하고자 하는 신경망 코드에 쉽게 가져올 수 있다. (전이학습)
3. 이미지를 증식시키고 증식 시킨 이미지들을 쉽게 신경망에 입력할 수 있다.
4. EarlyStopping 기능을 구현 할 수 있다.

EarlyStopping 기능

모델 학습시 지정 된 기간동안 모니터링하는 평가지표에서 성능향상이 일어나지 않는 경우

학습을 중단한다.

주로 많이 사용하는 콜백인자는 다음과 같다.

ex)

```
EarlyStopping(monitor = 'val_loss', patience = 2, verbose = 0, mode = 'auto')
```

monitor : 모니터링 할 평가지표를 설정한다. (오차 : val_loss, 정확도 : val_acc)

verbose : 콜백의 수행과정 노출여부를 결정한다.

- 0 : 아무런 표시 x
- 1 : 프로그래스 바(progress bar)를 출력
- 2 : 에폭마다 수행과정을 출력

patience : 지정한 수만큼의 기간에서 평가지표의 향상이 일어나지 않을 경우 학습을 중단한다. patient = 5 는 5번은 평가지표의 향상이 일어나지 않아도 학습을 중단하지 않는다.

구현 :

```
callbacks = [EarlyStopping(monitor = 'val_loss', patience = 2, verbose = 0, mode = 'auto')]
```

```
model.fit(x_train, y_train,  
          batch_size = 32,  
          validation_data = (x_val, y_val),  
          epochs = 10,  
          callbacks = callbacks)
```

R Shiny를 이용해서 신경망을 쉽게 활용하는 방법

<https://cafe.daum.net/oracleoracle/SgRM/285>

21.03.24

2021년 3월 24일 수요일 오전 9:42

데이터 시각화 자동화 코드를 이해하기 위한 5단계

<https://cafe.daum.net/oracleoracle/Sefi/39>



샤이니자동
화

홈페이지 구현을 위한 샤이니 ui의 기능들

<https://cafe.daum.net/oracleoracle/SgRM/293>

데이터 시각화 화면을 홈페이지로 만들기

<https://cafe.daum.net/oracleoracle/Sefi/31>

데이터 분석 그래프 시각화 홈페이지 구현 최종

<https://cafe.daum.net/oracleoracle/SgRM/294>

```
library(rsconnect)
```

```
rsconnect::setAccountInfo(name='web-pg',
                           token='4A23CBE1E0092232807E72618A587990',
                           secret='LuBelv2baQVJNthLPq4iQmd4eeJwiexVy+S+dFHE')
```

```
rsconnect::deployApp('c:\\yys277',appName = "myapp2777")
```

21.03.25

2021년 3월 25일 목요일 오전 9:44

머신러닝을 구현한 사이니 코드

<https://cafe.daum.net/oracleoracle/SgRM/298>



머신러닝까
지 다 구...

R shiny의 큰 틀

1. ui <- { 화면 구현 }
 - 1.1 sidebar
 - 1.2 body
2. server <- { 데이터를 분석하고 결과를 출력 구현 }
 - 2.1 테이블 결과 출력 코드
 - 2.2 summary와 box plot
 - 2.3 통계구현 (회귀분석, 의사결정트리)
 - 2.4 머신러닝 구현 (jrip, knn, 신경망 등)
 - 2.5 그래프 그리는 코드 (원형, 막대, 산포도, 박스, 그래프 등)

ui.R 과 sever.R 만들기

1. C 드라이브 밑에 KJM300 폴더를 만든다.
2. KJM300 폴더 밑에 ui.R, sever.R 을 만든다.



sever



ui

3. 아래의 코드를 실행한다.

ui.R 과 sever.R 만들기 최종

<https://cafe.daum.net/oracleoracle/SgRM/317>

경로 : C:\KJM300

21.03.26

2021년 3월 26일 금요일 오전 9:45

동영상을 object detection 하는 방법

<https://cafe.daum.net/oracleoracle/SgRM/319>

2. 동영상 내에서 특정사물을 분류하기 위한 가중치(1000개 클래스를 분류)가 필요 (yolov3.weights)
1. 코덱(현재 os에 동영상이 플레이 될 수 있도록 인코딩, 디코딩하는 코드)을 다운로드 받는다.
3. 동영상 파일이 필요
4. 1000개의 클래스를 분류하는 가중치를 신경망에 셋팅해서 동영상에서 사물을 detection 하는 코드 (detect.py)

얼굴 디텍션 코드 정리

<https://cafe.daum.net/oracleoracle/SgRM/188>

21.03.29

2021년 3월 29일 월요일 오전 9:44

구글 코랩에서 수행되게 수정한 전체 코드

<https://cafe.daum.net/oracleoracle/SgRM/328>

환경 구성 : <https://github.com/heartkilla/yolo-v3>

코드 구현 : [Build a Face Detection Model on a Video using Python \(analyticsvidhya.com\)](https://www.analyticsvidhya.com/article/build-a-face-detection-model-on-a-video-using-python/)

사진 한장을 신경망에 넣고 학습해서 그 사진에 해당하는 인물의 얼굴을 영상에서 찾는 코드

최종 코드

<https://cafe.daum.net/oracleoracle/SgRM/329>