

머신러닝/딥러닝을 위한

# 수치미분

— 수치미분 코드• 예제 —

# 수치미분 1차 버전 - numerical derivative

- 수치미분은 수학적공식을 쓰지 않고 C / 파이썬 등을 이용하여, 주어진 입력 값이 미세하게 변할 때 함수 값  $f$  는 얼마나 변하는지를 계산하는 해주는 것을 지칭

① 미분 하려는 함수  $f(x)$  정의

② 극한(lim) 개념을 구현하기 위해  $\Delta x$  는 작은 값으로 설정

③ 분자 / 분모 구현

$$f'(x) = \frac{df(x)}{dx} = \lim_{\Delta x \rightarrow 0} \frac{f(x+\Delta x) - f(x)}{\Delta x} = \lim_{\Delta x \rightarrow 0} \frac{f(x+\Delta x) - f(x-\Delta x)}{2\Delta x}$$

- 수치미분 구현 (1차 버전)

①  $f$  는 미분 하려는 함수. 외부에서 def, lambda 등으로 정의됨

$x$  는 미분 값을 알고자 하는 입력 값, 즉 미세하게 변하는 입력 값

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

```
    delta_x = 1e-4
```

```
    return (f(x+delta_x) - f(x-delta_x)) / (2*delta_x)
```

③ 분자 / 분모


② lim 에 해당되는 작은 값

# 수치미분 1차 버전 – numerical derivative


[예제 1] 함수  $f(x) = x^2$  에서 미분계수  $f'(3)$  을 구하기. 즉,  $x=3$  에서 값이 미세하게 변할 때, 함수  $f$  는 얼마나 변하는지 계산하라는 의미

$f'(3)$  계산 과정 (참고:  $f'(x) = 2x$ )


$$f'(3) = \lim_{\Delta x \rightarrow 0} \frac{f(3+\Delta x) - f(3-\Delta x)}{2\Delta x}$$

  $\Delta x$  는  $10^{-4}$  대입

$$f'(3) = \frac{f(3+1e^{-4}) - f(3-1e^{-4})}{2 * 1e^{-4}}$$

  $f(x) = x^2$

$$f'(3) = \frac{(3+1e^{-4})^2 - (3-1e^{-4})^2}{2 * 1e^{-4}}$$

 result

$$f'(3) = 6.0$$

```
import numpy as np

def my_func1(x):

    return x**2

f = lambda x : my_func1(x)

def numerical_derivative(f, x):

    delta_x = 1e-4

    return (f(x+delta_x) - f(x-delta_x)) / (2*delta_x)

result = numerical_derivative(f, 3)
print("result ==", result)

result = numerical_derivative(my_func1, 3)
print("result ==", result)

result == 6.000000000012662
result == 6.000000000012662
```

# 수치미분 1차 버전 - numerical derivative

[예제 2] 함수  $f(x) = 3xe^x$  를 미분한 함수를  $f'(x)$  라고 할 경우,  $f'(2)$  을 구하기.  
x=2 에서 값이 미세하게 변할 때, 함수 f 는 얼마나 변하는지 계산하라는 의미

## 수치 미분

```
import numpy as np

def my_func2(x):

    return 3*x*(np.exp(x))

f = lambda x : my_func2(x)

def numerical_derivative(f, x):

    delta_x = 1e-4

    return (f(x+delta_x) - f(x-delta_x)) / (2*delta_x)

result = numerical_derivative(f, 2)
print("result ==", result)

result = numerical_derivative(my_func2, 2)
print("result ==", result)

result == 66.50150507518049
result == 66.50150507518049
```

## 수학공식 검증

$$f(x) = 3xe^x \text{ 미분}$$



$$f'(x) = 3e^x + 3xe^x$$



x = 2 대입

$$f'(2) = 3e^2 + 3 \cdot 2e^2$$



```
print("3*exp(2) + 3*2*exp(2) == ", end='')
print(3*np.exp(2) + 3*2*np.exp(2))
```

```
3*exp(2) + 3*2*exp(2) == 66.50150489037586
```

## 수치미분 최종 버전 – numerical derivative

- 입력 변수가 하나 이상인 다 변수 함수의 경우, 입력변수는 서로 독립적이기 때문에 수치미분 또한 변수의 개수만큼 개별적으로 계산하여야 함

[예]  $f(x, y) = 2x + 3xy + y^3$  라면, 입력 변수  $x, y$  두 개 이므로  $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}$  각각 수치미분 수행.

$f'(1.0, 2.0)$  값을 계산하기 위해서는,

$\Rightarrow x = 1.0$  에서의 미분계수는 변수  $y = 2.0$  을 상수로 대입하여  $\frac{\partial f(x, 2)}{\partial x}$  를 수행

$$f(x, 2) = 2x + 6x + 8$$

$\Rightarrow y = 2.0$  에서의 미분계수 또한 변수  $x = 1.0$  인 상수로 대입하여  $\frac{\partial f(1, y)}{\partial y}$  를 수행

$$f(1, y) = 2 + 3y + y^3$$

[insight]  $f(x, y) = 2x + 3xy + y^3$  ,인 경우  $f'(1.0, 2.0) = (8.0, 15.0)$  직관적 이해

$\Rightarrow x = 1.0$  에서 미분 값을 구한다는 것은,  $y$  값은  $2.0$  으로 고정한 상태에서,  $x = 1.0$  을 미세하게 변화시킬 때  $f(x, y)$  는 얼마나 변화는지 알아보겠다는 의미. 즉,  $y = 2.0$  으로 고정된 상태에서  $x = 1.0$  을 미세하게 변화시키면  $f(x, y)$  는  $8.0$  만큼 변한다는 의미

$\Rightarrow y = 2.0$  에서 미분 값을 구한다는 것은,  $x$  값은  $1.0$  으로 고정한 상태에서,  $y = 2.0$  을 미세하게 변화시킬 때  $f(x, y)$  는 얼마나 변화는지 알아보겠다는 의미. 즉,  $x = 1.0$  으로 고정된 상태에서  $y = 2.0$  을 미세하게 변화시키면  $f(x, y)$  는  $15.0$  만큼 변한다는 의미

## 수치미분 최종 버전 – numerical derivative

```
import numpy as np

def numerical_derivative(f, x):
    delta_x = 1e-4
    grad = np.zeros_like(x)

    it = np.nditer(x, flags=['multi_index'], op_flags=['readwrite'])

    while not it.finished:
        idx = it.multi_index

        tmp_val = x[idx]
        x[idx] = float(tmp_val) + delta_x
        fx1 = f(x)    # f(x+delta_x)

        x[idx] = tmp_val - delta_x
        fx2 = f(x)    # f(x-delta_x)
        grad[idx] = (fx1 - fx2) / (2*delta_x)

        x[idx] = tmp_val
        it.iternext()

    return grad
```

소스출처(재구성): <https://github.com/WegraLee/deep-learning-from-scratch/tree/master/common>

## 수치미분 예제 - 1변수 함수 $f(x) = x^2$ , $f'(3.0)$

```
import numpy as np

def numerical_derivative(f, x):    # 수치미분 debug version
    delta_x = 1e-4
    grad = np.zeros_like(x)
    print("debug 1. initial input variable =", x)
    print("debug 2. initial grad =", grad)
    print("=====")

    it = np.nditer(x, flags=['multi_index'], op_flags=['readwrite'])

    while not it.finished:
        idx = it.multi_index

        print("debug 3. idx = ", idx, ", x[idx] = ", x[idx])

        tmp_val = x[idx]
        x[idx] = float(tmp_val) + delta_x
        fx1 = f(x)    # f(x+delta_x)

        x[idx] = tmp_val - delta_x
        fx2 = f(x)    # f(x-delta_x)
        grad[idx] = (fx1 - fx2) / (2*delta_x)

        print("debug 4. grad[idx] = ", grad[idx])
        print("debug 5. grad = ", grad)
        print("=====")

        x[idx] = tmp_val
        it.iternext()

    return grad
```

# 입력변수 1 개의 함수  $f(x) = x^2$

```
def func1(input_obj):
```

```
    x = input_obj[0]
```

```
    return x**2
```

#  $x = 3.0$  에서의 편미분 값

```
numerical_derivative( func1, np.array([3.0]) )
```

```
debug 1. initial input variable = [3.]
```

```
debug 2. initial grad = [0.]
```

```
=====
```

```
debug 3. idx = (0,) , x[idx] = 3.0
```

```
debug 4. grad[idx] = 6.0000000000012662
```

```
debug 5. grad = [6.]
```

```
=====
```

```
array([6.])
```

# 수치미분 예제 - 2변수 함수 $f(x, y) = 2x + 3xy + y^3$ , $f'(1.0, 2.0)$

```
import numpy as np

def numerical_derivative(f, x):    # 수치미분 debug version
    delta_x = 1e-4
    grad = np.zeros_like(x)
    print("debug 1. initial input variable =", x)
    print("debug 2. initial grad =", grad)
    print("=====")

    it = np.nditer(x, flags=['multi_index'], op_flags=['readwrite'])

    while not it.finished:
        idx = it.multi_index

        print("debug 3. idx = ", idx, ", x[idx] = ", x[idx])

        tmp_val = x[idx]
        x[idx] = float(tmp_val) + delta_x
        fx1 = f(x)    # f(x+delta_x)

        x[idx] = tmp_val - delta_x
        fx2 = f(x)    # f(x-delta_x)
        grad[idx] = (fx1 - fx2) / (2*delta_x)

        print("debug 4. grad[idx] = ", grad[idx])
        print("debug 5. grad = ", grad)
        print("=====")

        x[idx] = tmp_val
        it.iternext()

    return grad
```

```
# 입력변수 2 개의 함수 f(x, y) = 2x + 3xy + y^3
def func1(W):

    x = W[0]
    y = W[1]

    return ( 2*x + 3*x*y + np.power(y,3) )

f = lambda W : func1(W)

# (x,y) = (1.0, 2.0) 에서의 편미분 값
W = np.array([1.0, 2.0])

numerical_derivative( f, W )

debug 1. initial input variable = [1. 2.]
debug 2. initial grad = [0. 0.]
=====
debug 3. idx = (0,) , x[idx] = 1.0
debug 4. grad[idx] = 7.999999999990237
debug 5. grad = [8. 0.]
=====
debug 3. idx = (1,) , x[idx] = 2.0
debug 4. grad[idx] = 15.000000010019221
debug 5. grad = [ 8.          15.00000001]
=====

array([ 8.          , 15.00000001])
```



# 수치미분 예제 - 4변수 함수 $f(w, x, y, z) = wx + xyz + 3w + zy^2$ , $f'(1.0, 2.0, 3.0, 4.0)$

```
import numpy as np

def numerical_derivative(f, x):    # 수치미분 debug version
    delta_x = 1e-4
    grad = np.zeros_like(x)
    print("debug 1. initial input variable =", x)
    print("debug 2. initial grad =", grad)
    print("=====")

    it = np.nditer(x, flags=['multi_index'], op_flags=['readwrite'])

    while not it.finished:
        idx = it.multi_index

        print("debug 3. idx = ", idx, ", x[idx] = ", x[idx])

        tmp_val = x[idx]
        x[idx] = float(tmp_val) + delta_x
        fx1 = f(x)    # f(x+delta_x)

        x[idx] = tmp_val - delta_x
        fx2 = f(x)    # f(x-delta_x)
        grad[idx] = (fx1 - fx2) / (2*delta_x)

        print("debug 4. grad[idx] = ", grad[idx])
        print("debug 5. grad = ", grad)
        print("=====")

        x[idx] = tmp_val
        it.iternext()

    return grad
```

```
# 입력변수 4 개의 함수
# f(w,x,y,z) = wx + xyz + 3w + zy^2
# input_obj 는 행렬
def func1(input_obj):

    w = input_obj[0, 0]
    x = input_obj[0, 1]
    y = input_obj[1, 0]
    z = input_obj[1, 1]

    return ( w*x + x*y*z + 3*w + z*np.power(y,2) )

# 입력을 2X2 행렬로 구성함
input = np.array([ [1.0, 2.0], [3.0, 4.0] ])

numerical_derivative( func1, input )

debug 1. initial input variable = [[1. 2.]
 [3. 4.]]
debug 2. initial grad = [[0. 0.]
 [0. 0.]]
=====
debug 3. idx = (0, 0) , x[idx] = 1.0
debug 4. grad[idx] = 5.000000000023874
debug 5. grad = [[5. 0.]
 [0. 0.]]
=====
debug 3. idx = (0, 1) , x[idx] = 2.0
debug 4. grad[idx] = 13.00000000000523
debug 5. grad = [[ 5. 13.]
 [ 0.  0.]]
=====
debug 3. idx = (1, 0) , x[idx] = 3.0
debug 4. grad[idx] = 32.00000000006753
debug 5. grad = [[ 5. 13.]
 [32.  0.]]
=====
debug 3. idx = (1, 1) , x[idx] = 4.0
debug 4. grad[idx] = 15.00000000000568
debug 5. grad = [[ 5. 13.]
 [32. 15.]]
=====

array([[ 5., 13.],
       [32., 15.]])
```

w 변수  
(0,0)  
↓  
x 변수  
(0,1)  
↓  
y 변수  
(1,0)  
↓  
z 변수  
(1,1)