경사하강법_순한맛

- 미분(differentiation)

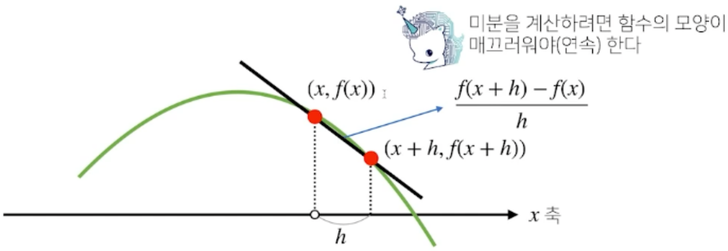
  $f'(x) = lim_{h \to 0} \frac{f(x+h)-f(x)}{h}$

  변수의 움직임에 따른 함수값의 변화를 측정하기 위한 도구

  최적화에서 제일 많이 사용하는 기법

  ```
  import sympy as sym
  from sympmy.abc import x
  sym.diff(sym.poly(x**2 + 2*x + 3), x)
  # Poly(2*x + 2, x, domain='ZZ')
  ```

- 함수 f의 주어진 점 $(x, f(x))$에서의 접선의 기울기를 구한다.

  h를 0으로 보내면 $(x, f(x))$에서 접선의 기울기로 수렴

- **증가** => 미분값 + // 음수, 왼쪽 / 양수, 오른쪽
- **감소** => 미분값 -
- **경사상승법(gradient ascent)**: 미분값 더함, 함수의 극대값 위치 구할 때 사용
- **경사하강법(fradient descent)**: 미분값 뺌, 함수의 극소값 위치 구함
- 경사상승/경사하강 모두 극값 도달 시, 움직임 멈춤

  => 목적함수 최적화 자동 끝남

```
# gradient: 미분을 계산하는 함수
# init: 시작점, lr: 학습률, eps: 알고리즘 종료조건

var = init
grad = gradient(var)
while (abs(grd) > eps): # 컴퓨터로 계산할 때, 미분이 정확히 0이 되는 것 불가능 => eps보다 작을 때의 종료 조건 필요
  var = var - lr * grad # lr은 미분을 통해 업데이트 속도 조절
  grad = gradient(var) # 종료조건 성립 전까지, 미분값 계속 업데이트
```

```python
import sympy as sym
from sympy.abc import x
import numpy as np

def func(val):
    fun=sym.poly(x**2) # f(x)=x^2
    return fun.subs(x,val),fun #subs(x,val)==f(val) , f(2)=2^2=4
def func_gradient(func,val):
    _,function=func(val)
    diff=sym.diff(function,x) #f'(x)=2x
    return diff.subs(x,val),diff #f'(2)=2*2=4
def gradient_descent(func,init_point,lr=1e-2,eps=1e-5):
    cnt=0
    val=init_point
    diff,_=func_gradient(func,init_point)
    while np.abs(diff) >eps:
        val=val-lr*diff # Gradient descent
        diff,_=func_gradient(func,val)
        cnt+=1

    print(f"함수 : {func(val)[1]} , 연산횟수 : {cnt}, 최소점 : ({val},{func(val)[0]})")
```

```python
gradient_descent(func,init_point=np.random.uniform(low=-2,high=2))
```

```
함수 : Poly(x**2, x, domain='ZZ') , 연산횟수 : 554, 최소점 : (-0.00000497881365721495,2.47885854332701E-11)
```

- **편미분(partial differentiation)**: 백터가 입력인 다변수 함수

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

  e_i는 i번째 값만 1이고 나머지는 0인 단위벡터

  $f(x,y) = x^2 + 2xy + 3 + cos(x+2y)$

  $\partial_x f(x,y) = 2x + wy - sin(x+2y)$ => y를 상수 취급, x만 미분

  x방향 움직임만 분석

  `sym.diff()` 이용

  주어진 개수만큼 편미분 가능

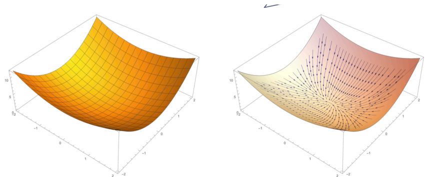
- 각 변수 별로 편미분 계산한 그레디언트(gradient) 벡터 이용해 경사하강/경사상승법에 사용

  $\nabla f = (\partial_{x1} f, \partial_{x2} f, \ldots, \partial_{xd} f)$

  ∇: nabla

  백터 ∇f를 사용해 변수 $x = (x_1, \ldots, x_d)$ 동시에 업데이트 가능

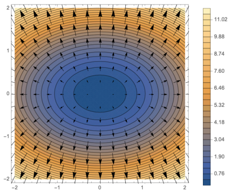
- 각 점 (x,y,z) 공간에서 $f(x,y)$ 표면을 따라 $-\nabla f$ 벡터를 그리면 다음과 같다.



$f(x,y) = x^2 + 2y^2$ $\quad$ $-\nabla f = -(2x, 4y)$

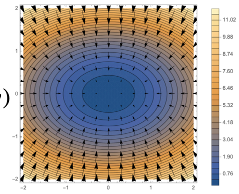
- 그레디언트 벡터 $\nabla f(x,y)$는 각 점 (x,y)에서 가장 빨리 증가하는 방향으로 흐름

$$f(x, y) = x^2 + 2y^2$$
$$\Rightarrow \nabla f = (2x, 4y)$$



- $-\nabla f$는 $\nabla(-f)$랑 같고 이는 각 점 에서 가장 빨리 감소하게 되는 방향과 같다

$$f(x, y) = x^2 + 2y^2$$
$$\Rightarrow -\nabla f = -(2x, 4y)$$



```
# gradient: 미분을 계산하는 함수
# init: 시작점, lr: 학습률, eps: 알고리즘 종료조건

var = init
grad = gradient(var)
while (norm(grd) > eps): # 벡터는 절대값 대신 노름(norm)을 계산해서 종료조건 설정
  var = var - lr * grad
  grad = gradient(var)
```

> **gradient**: 공간에 대한 기울기