05. 경사법 (Local descent)

아래쪽으로 하강 시킨다.

다변량 함수의 최소값 찾기

일변량 $\xrightarrow[\text{분}]{\text{미}}$ 도함수

: 구간 결정 후 함수값 비교해 최소값 존재하는 짧은 구간 결정

다변량 $\xrightarrow[\text{분}]{\text{미}}$ 기울기, 경사도 (gradient)

: gradient 이용해 최소값 있다고 생각되는 아래쪽으로 design point(계획점) 변경 해나가기 by. 반복 계산

국소 하강법 (Local descent)

- 국소 모형 (Local model)

- 최소값을 찾기 위해 주어진 점에서 계산을 수행하는 모델
- 목적 함수 값을 작게 해주는 방향으로 이동시키는데 사용됨
  → 주어진 점에서의 기울기 (경사도) 값
- 근사 최소값에 수렴할 때 까지 계획점 (design point) 갱신

⇒ 최소값 위치 모르고, 최소값이라 생각하는 값 결정
그 점(좌표)에서 방향 도함수 찾음

- 하강 방향 반복법 (descent direction iteration)

- 국소 모형 이용해 목적함수 값을 작게 하는 방향으로 적절한 단계값 (step size) 이용해 계획점 x를 반복적으로 갱신

① design point 결정
② gradient 계산
③ gradient 방향으로 단계값 크기만큼 계획점 이동
: 목적함수 값 작아질 때까지 반복 수행

- 하강 방향법 : 계산 알고리즘

① 초기 계획점 $x^{(1)}$ 결정 ← 사전 정보, 함수 특징 이해하고 있다는 가정
② K번째 반복에서 계획점 $x^{(k)}$의 종료조건 확인
③ $x^{(k)}$ 에서의 경사도 (헷세 행렬) 정보를 이용하여
하강 방향 $d^{(k)}$ 결정 cf. 경우에 따라 $\|d\| = 1$
급격하게 변하는 문제 완화
④ 단계값 (학습률) $\alpha^{(k)}$ 결정
하이퍼 파라미터
⑤ (k+1) 번째 계획점 갱신

$$: x^{(k+1)} \leftarrow x^{(k)} + \alpha^{(k)} d^{(k)}$$

결정된 값

선탐색 / 근사 선탐색 으로 결정
관심있는 계산

$d^{(k)}$
: gradient (directional derivate) 에 정보 추가
→ 헷세 행렬 이용해 curvature information 활용

→ K번째 반복에 대해서의 목적함수 값과 (k+1)번째 목적함수 값 비교
값 변화 적으면 (k+1)번째 목적함수 값을 최소값의 근사값으로!

- 선 탐색 : 단계값 (step size) 탐색
- 학습률 (learning rate) : $\alpha \equiv \alpha^{(k)}$, 하이퍼 파라미터
  학습률 고정되면 목적함수 커지는 경우 생길 수 있음
- 쇠퇴 단계 인수 (decaying step factor)

$: \alpha^{(k)} = \alpha^{(1)} \gamma^{k-1}$, $\gamma \in (0,1]$ → 반복 수행할수록 α 감소
기계학습, machine learning

국소 하강법 (Local descent)

- 선탐색 (line Search) : 단계값 탐색

- 목적 함수 : $\underset{\alpha}{\text{minimize}}\ f(x + \alpha d)$ 계획점(design point) 하강 방향
- 주어진 계획점 x와 하강 방향 d에 대하여 목적함수 f를 최소화 하는 α를 찾는 문제
- 일변량 탐색법 : α라는 일변량 미지수 최소화

→ α의 최소값 있다고 생각되는 구간 결정 알고리즘 사용가능

bracket-minimum, trifold, 피보나치, Golden Section … 등

- 알고리즘

```
function line_search(f, x, d)
    objective = α -> f(x + α*d)
    a, b = bracket_minimum(objective)
    α = minimize(objective, a, b)
    return x + α*d
end
```

초기값 설정

↳ trifold, 피보나치, 황금분할 탐색법 모두 이용가능

→ f : 목적함수. 주어진 것

x : 계획점. local model 이나가 정해짐

d : directional vector, local model에서 gradient 정해짐.

→ α에 의해 Gradient (d)만큼 이동시키고 나면 목적함수 값 작아짐. $x+\alpha \cdot d$로 α에 의해 찾은 값이 목적함수 작게 해주는 방향으로 이동된 국소모형 계획점으로 판단.

ex) $f(x_1, x_2, x_3) = \sin(x_1 x_2) + \exp(x_2 + x_3) - x_3$

$x = [1, 2, 3]$ $\quad d = [0, -1, -1]$

$\rightarrow \underset{\alpha}{\text{minimize}}\ \sin((1+0\alpha)(2-\alpha)) + \exp((2-\alpha)+(3-\alpha)) - (3-\alpha)$

$= \underset{\alpha}{\text{minimize}}\ \sin(2-\alpha) + \exp(5-2\alpha) + \alpha - 3$

Golden Search Method, trifold … 이용해 α값 찾기

$\alpha \approx 3.127$, $x = [1, -1.126, -0.126]$

↳ $x + \alpha \cdot d$인 다음 계획점

⇒ line Search Algorithm은 결국 단계값 α를 일변량 최적화 문제로 전환해 매번 계획점 갱신될 때 마다 최적화 문제 수행해 계산량 多

line search algorithm

- bracket minimum
- minimize(최적화 방법)

```
import numpy as np
from IPython.display import Image

def bracket_minimum(f,x,s=1E-2,k=2.0):
    print('bracket_minimum')
    a,ya=x,f(x)
    b,yb=a+s, f(a+s)
    print('init: (a:%.4f, b:%.4f) (ya:%.4f, yb:%.4f)' %(a,b,ya,yb))
    if yb>ya:
        a,b=b,a
        ya,yb=yb,ya
        s=-s
    while True:
        c,yc=b+s,f(b+s)
        print('step: (a:%.4f, b:%.4f, c:%.4f) (ya:%.4f, yb:%.4f, yc:%.4f)'%(a,b,c,ya,yb,yc))
        if yc>yb:
            print(' ')
            return (a,c) if a<c else (c,a)
        else:
            a,ya,b,yb=b,yb,c,yc
            s*=k
```

하이퍼 파라미터 : X, S, K

X : 계획점 , S : 학습률

```
def golden_section_search(f,x,epsilon=1E-6):
    print('golend_section_search')
    a,b=bracket_minimum(f,x)
    print('init: (a:%.4f, b:%.4f)' %(a,b))
    distance=abs(a-b)
    psi=0.5*(1.+np.sqrt(5))
    rho=psi**(-1)
    d=rho*b+(1.-rho)*a
    yd=f(d)
    i=1
    while distance>epsilon:
        c=rho*a+(1.-rho)*b
        yc=f(c)
        if yc<yd:
            b,d,yd=d,c,yc
        else:
            a,b=b,c
        pa,pb=(a,b) if a<b else (b,a)
        print('%d:(a:%.4f, b:%.4f)' %(i,pa,pb))
        distance=abs(pa-pb)
        i+=1
    a,b=(a,b) if a<b else (b,a)
    x=0.5*(a+b)
    y=f(x)
    print('golden_section_search Algorithm finish...!')
    print(' ')
    return x,y
```

→ 초기값에 따라 bracket_min 값 달라지기 때문에 최소값 다르게 구해질 수 있다.

```
def line_search(f,x,d):
    def obj(alpha):
        return f(x+alpha*d) #이 값이 작은 alpha를 찾아보자
    alpha,_=golden_section_search(obj,0.) #y는 사용 안할 꺼야! _로 지정
    return alpha,x+alpha*d

def f(x):
    y=np.sin(x[0]*x[1])+np.exp(x[1]+x[2])-x[2]
    return y

x=np.asarray([1,2,3])
d=np.asarray([0,-1,-1])

print(line_search(f,x,d))
```

```
golend_section_search
bracket_minimum
init: (a:0.0000, b:0.0100) (ya:146.3225, yb:143.3978)
step: (a:0.0000, b:0.0100, c:0.0200) (ya:146.3225, yb:143.3978, yc:140.5312)
step: (a:0.0100, b:0.0200, c:0.0400) (ya:143.3978, yb:140.5312, yc:134.9678)
step: (a:0.0200, b:0.0400, c:0.0800) (ya:140.5312, yb:134.9678, yc:124.4890)
step: (a:0.0400, b:0.0800, c:0.1600) (ya:134.9678, yb:124.4890, yc:105.8941)
step: (a:0.0800, b:0.1600, c:0.3200) (ya:124.4890, yb:105.8941, yc:76.5712)
step: (a:0.1600, b:0.3200, c:0.6400) (ya:105.8941, yb:76.5712, yc:39.8823)
step: (a:0.3200, b:0.6400, c:1.2800) (ya:76.5712, yb:39.8823, yc:10.4124)
step: (a:0.6400, b:1.2800, c:2.5600) (ya:39.8823, yb:10.4124, yc:-0.0843)
step: (a:1.2800, b:2.5600, c:5.1200) (ya:10.4124, yb:-0.0843, yc:2.1037)

init: (a:1.2800, b:5.1200)
1:(a:1.2800, b:3.6533)
2:(a:2.1865, b:3.6533)
3:(a:2.7467, b:3.6533)
4:(a:2.7467, b:3.3070)
5:(a:2.9607, b:3.3070)
6:(a:2.9607, b:3.1747)
7:(a:3.0425, b:3.1747)
8:(a:3.0930, b:3.1747)
9:(a:3.0930, b:3.1435)
10:(a:3.1123, b:3.1435)
11:(a:3.1123, b:3.1316)
12:(a:3.1197, b:3.1316)
13:(a:3.1242, b:3.1316)
14:(a:3.1242, b:3.1288)
15:(a:3.1260, b:3.1288)
16:(a:3.1260, b:3.1277)
17:(a:3.1266, b:3.1277)
18:(a:3.1266, b:3.1273)
19:(a:3.1269, b:3.1273)
20:(a:3.1269, b:3.1271)
21:(a:3.1270, b:3.1271)
22:(a:3.1270, b:3.1271)
23:(a:3.1270, b:3.1271)
24:(a:3.1270, b:3.1271)
25:(a:3.1270, b:3.1271)
26:(a:3.1270, b:3.1271)
27:(a:3.1270, b:3.1270)
28:(a:3.1270, b:3.1270)
29:(a:3.1270, b:3.1270)
30:(a:3.1270, b:3.1270)
31:(a:3.1270, b:3.1270)
32:(a:3.1270, b:3.1270)
golden_section_search Algorithm finish...!

(3.1270454770468, array([ 1.        , -1.12704548, -0.12704548]))
```

Golden Section Search에서 나온 값. a, b가 굉장히 작아짐 : 중간값은 3.127045~ 새로운 계획점은 (1, -1.13, -0.13)

- line search는 bracket_minimum 도 계산 여러번 하고
- golden_section_search 또 시행
- 매번 계획점 갱신할 때 마다 하는 것이 계산적으로 비효율적

-> alpha를 크게 해놓고 조건이 만족될 때 까지 적절히 줄여나가자

-> backtracking line search algorithm

국소 하강법 (Local descent)

- 근사 선 탐색 : Armijo 조건 (충분 감소 조건)

- 매 반복에서 선 탐색을 수행하는 것이 아니라 보다 적은 단계값 이용해 반복 수행
  ↳ 반복마다 α값을 최적화 문제로 바꿔 계산.
- 하강법은 항상 하강하므로 목적함수를 작게 할 수 있다면 적절히 작은 α 만으로도 충분

→ α 크게 해 design point 이동시켜 목적함수 값 구하기
이전 단계의 목적함수보다 크면 α값 $1/2$, 계획점 이동

→ 몇 번 α를 변경해주면 이전 단계의 목적함수보다 작게 해주는 계획점 결정해주는 α 찾을 수 있어!

- 충분 감소 조건

: 목적함수를 충분히 작게 해주는 단계값(α)이 갖춰야 할 조건

① $f(x^{(k+1)}) \le f(x^{(k)}) + \beta \cdot \alpha \cdot \boxed{\nabla_{d^{(k)}} f(x^{(k)})}$ 방향도함수 >0

$f(x^{(k+1)}) \le f(x^{(k)}) + \beta \cdot \alpha \cdot \boxed{\nabla f(x^{(k)})^T \cdot d^{(k)}}$

② $\beta \in [0, 1]$: $\beta = 1 \times 10^{-4}$ 로 많이 결정함

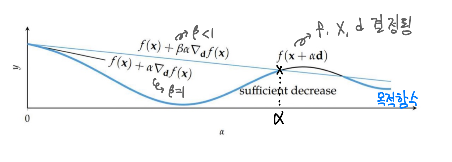
→ $f(x^{(k)})$: 결정된 값

$\beta \cdot \alpha \cdot \nabla_{d^{(k)}} f(x^{(k)})$ 는 step size α를 정해주는 식과 같음

이 때의 α가 목적함수 이전보다 더 작게 해줌

$\beta = 0$: $f(x^{(k+1)}) = f(x^{(k)})$, 어떤 감소도 받아들이겠다!

⇒ α를 큰 값 주고 부등식 만족할 때까지 갱신



① 무조건 α 큰 값 찾기 : 목적 함수 값 크면 줄여나감

② 어느 순간, 줄어든 α에 의해 목적함수 줄어든다.

③ 목적함수는 틀림없이 줄어드니 그 값을 단계값으로 사용

- Algorithm (backtracking_line_search 라고 부름)

계획점에서의 함수 값
계획점에서의 gradient 값

```
function backtracking_line_search(f, ∇f, x, d, α; p=0.5, β=1e-4)
    y, g = f(x), ∇f(x)
    while f(x + α*d) > y + β*α*(g·d)
        α *= p
    end
    α
end
```

Armijo 조건
↳ α 만큼 이동시킨 것이 이전 목적함수보다 커지면 α 줄이기

1보다 작은 값이니까 α 줄어든다.

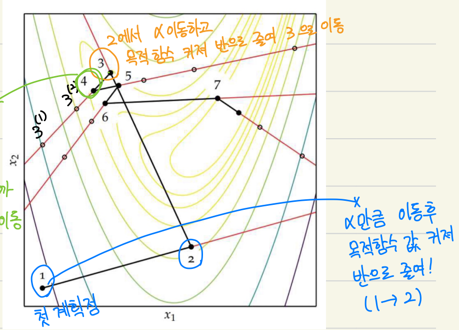
→ 인자 설명

f : 목적함수, ∇f : gradient, x : 계획점,
d : velocity, α : 최대 α값, p : 줄어든 비율
β : Armijo 조건에서 사용하게 될 값.

→ backtracking line search

장 계산량 ↓ 단 필요 정보 ↑ (ex. gradient)

- Rosenbrock function (=banana function)



⇒ Aromijo 조건 이용해 다변량에서도 최소값 찾을 수 O

ex) $f(x_1, x_2) = x_1^2 + x_1 x_2 + x_2^2$

계획점 $x = [1,2]$, $d = [-1, -1]$ velocity , $\alpha = 10$ 이동시킬 최대값

$p = 0.5$, $\beta = 1 \times 10^{-4}$

줄이는 비율

backtracking line search

```
def backtracking_line_search(f,gradient,x,d,alpha,p=0.5,beta=1E-4):

    y,g =f(x),gradient
    i=1

    while f(x+alpha*d) > y+beta*alpha*np.dot(g,d):
        #np.dot(g,d) = d^t d -> 벡터 내적 ; g.T @ d로 계산 가능
        alpha*=p
        print('%d: alpha=%.4f'%(i,alpha))
        i+=1

    return alpha

def f(x):
    y=x[0]**2+x[0]*x[1]+x[1]**2
    return y


def pdf0(x):
    return 2*x[0]+x[1]

def pdf1(x):
    return 2*x[1]+x[0]

x=np.array([1,2])
d=np.array([-1,-1])
gradient=np.array([pdf0(x),pdf1(x)])

alpha=10

alpha=backtracking_line_search(f,gradient,x,d,alpha)
x=x+alpha*d
print(x)
```

$\alpha = 10$ 반으로 줄여도 Armijo 조건 만족X
2.5 까지 줄여 Armijo 조건 만족
$\alpha = 2.5$ 로 fix.

```
1: alpha=5.0000
2: alpha=2.5000
[-1.5 -0.5]
```

계획점 $x = [1,2]$ 에서 2.5 만큼 줄여 $[-1.5, -0.5]$로 이동