7. 미분 가능한 선형대수

1. 자코비 행렬 (Jaconian Matrix)

   자코비 행렬은 벡터 값 함수의 편미분으로 구성된 행렬 입니다.
   예를 들어, 함수
   $f(x) = [f_1(x), f_2(x), ..., f_m(x)]^T$가 있고, 벡터 $x = [x_1, x_2, ..., x_n]^T$ 일 때, 자코비 행렬 J는 다음과 같이 정의 됩니다.

   - 편미분으로 구성된 행렬

     편미분은 함수가 여러 변수에 의존할 때, 한 번 변수를 고정하고 다른 변수에 대한 변화율을 계산하는 방법입니다. 즉, 어떤 함수가 여러 변수로 이루어져 있다면 그 함수의 각 변수에 대해 따로 미분하는 것이 편미분 입니다.

     자코비 행렬은 여러 개의 함수로 이루어진 벡터 함수의 모든 변수에 대한 편미분 값들을 모안 놓은 행렬입니다. 즉, 각 함수가 특정 변수에 대해 어떻게 변하는지, 즉 기울기를 각 변수별로 계산한 결과들을 배열할 것입니다. 이 편미분들을 행렬 형태로 정리한 것이 자코비 행렬입니다.

     예시

     벡터 값 함

$$\mathbf{J} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \frac{\partial f_m}{\partial x_2} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}$$

   자코비 행렬은 다변수 함수의 기울기를 벡터 형식으로 나타낸 것으로, 딥러닝의 역전파 알고리즘에서 각 층의 기울기를 계산하는 데 사용됩니다.

2. 헤시안 행렬 (Hessian Matrix)

   헤시안 행렬은 2차 편미분으로 구성된 행렬입니다. 스칼라 값 함수 $f(x)$의 경우 헤시안 행렬은 다음과 같이 정의됩니다.

$$\mathbf{H} = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}$$

헤시안 행렬은 2차 미분 정보를 담고 있어, 최적화 과정에서 함수의 곡률을 파악하고 더 빠르게 수렴하는 데 도움이 됩니다. 뉴턴 방법(Newton's Method)과 같은 최적화 알고리즘에서 많이 사용됩니다.

```
import sympy as sp

# 변수와 함수 정의
x1, x2 = sp.symbols('x1 x2')
f1 = x1**2 + x2
f2 = x1 + sp.sin(x2)

# 벡터 함수 f
f = sp.Matrix([f1, f2])

# 변수 벡터 x
x = sp.Matrix([x1, x2])

# 자코비 행렬 계산
J = f.jacobian(x)

# 출력
sp.pprint(J)

#------------------------------------------------------------#

# 스칼라 함수 정의
f = x1**2 + x2**2 + x1 * x2

# 헤시안 행렬 계산
H = sp.hessian(f, x)
```

```
# 출력
sp.pprint(H)
```

```
import torch

# 입력 텐서 정의
x = torch.tensor([1.0, 2.0], requires_grad=True)

# 함수 정의
f1 = x[0]**2 + x[1]
f2 = x[0] + torch.sin(x[1])
f = torch.stack([f1, f2])  # 벡터 함수로 결합

# 자코비 행렬 계산
J = []
for i in range(f.shape[0]):
    grad_f = torch.autograd.grad(f[i], x, retain_graph=True, 
    J.append(grad_f)

J = torch.stack(J)  # 자코비 행렬로 변환

print("Jacobian Matrix:\n", J)

#-----------------------------------------------------------#

# 스칼라 함수 정의
f_scalar = x[0]**2 + x[1]**2 + x[0]*x[1]

# 헤시안 행렬 계산
H = []
for i in range(x.shape[0]):
    grad_fi = torch.autograd.grad(f_scalar, x, create_graph=T
    hessian_row = []
    for j in range(x.shape[0]):
        hess_ij = torch.autograd.grad(grad_fi[j], x, retain_g
        hessian_row.append(hess_ij)
    H.append(torch.stack(hessian_row))
```

```
H = torch.stack(H)  # 헤시안 행렬로 변환

print("Hessian Matrix:\n", H)
```