5-1. 극값 찾기 : 최대/최소값, 정류점 (Stationary Points)

미적분에서 최적화 문제와 관련된 중요한 개념인 극값, 최대/최소값 그리고 정류점은 딥러닝의 학습과정에서 특히 손실 함수(Loss Function)를 최소화하는데 핵심적인 역할을 합니다.

1. 극값(Extrema)

   극값은 함수의 최대값 또는 최소값을 말합니다. 주어진 구간에서 함수가 가장 큰 값이나 가장 작은 값을 가질 때, 그 지점을 극대값(Maximum) 혹은 극소값(Minimum)이라고 합니다.

   극값을 찾으려면 함수의 도함수(미분)를 이용하여 함수가 증가하는 구간과 감소하는 구간을 분석합니다.

2. 최대/최소값 (Maxima/Minima)

   함수 $f(x)$가 정의된 범위에서 최대값은 $f(x)$가 가질 수 있는 가장 큰 값이고, 최소값은 가장 작은 값입니다.

   함수가 최대값이나 최소값을 가질때의 점은 보통 도함수 $f'(x)$가 0이 되는 지점에서 찾을 수 있습니다.

3. 정류점 (Stationary Points)

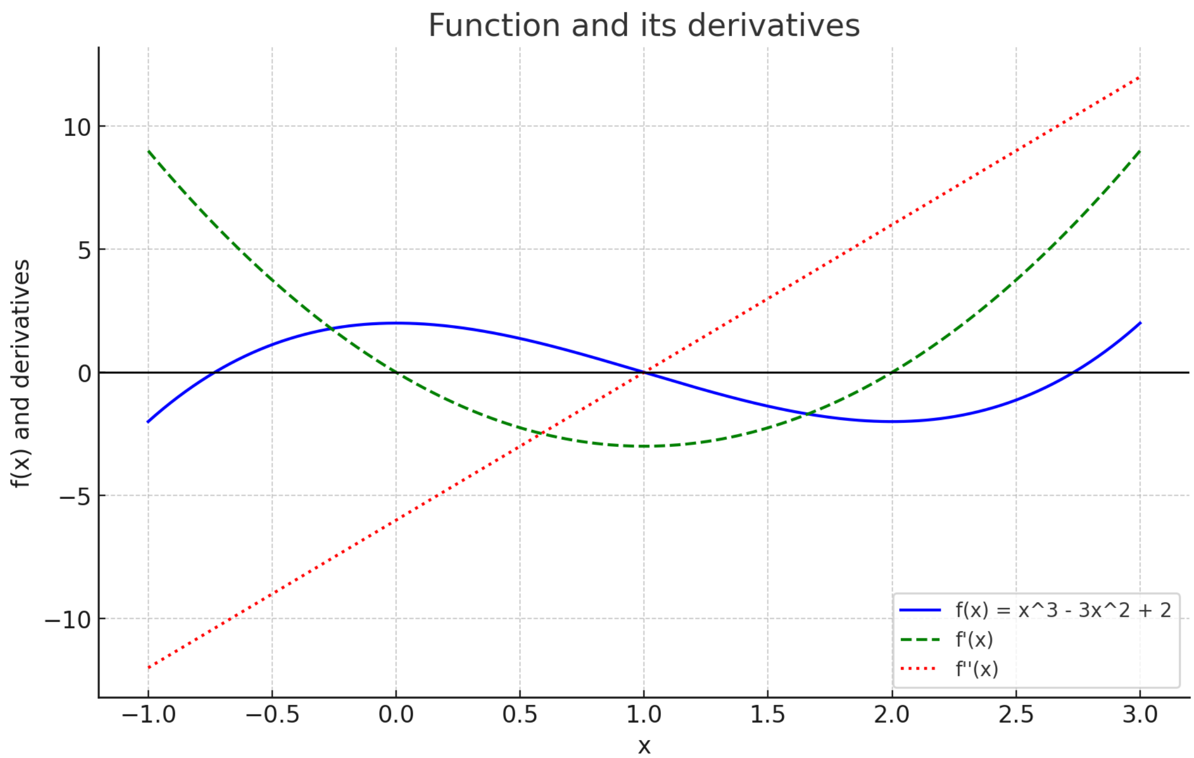
   정류점은 함수의 도함수 값이 0인 지점을 의미합니다. 즉, $f'(x) = 0$이 되는 점입니다. 이 정류점에서는 함수가 더이상 증가하거나 감소하지 않기 때문에 함수의 그래프에서 수평한 접선을 가집니다.

   모든 정류점이 극값이 되는 것은 아닙니다. 어떤 정류점에서는 함수가 증가 또는 감소하다가 잠시 멈춘 후 계속 증가하거나 감소할 수 있습니다.

4. 딥러닝에서의 활용

딥러닝 모델을 학습시킬 때는 손실 함수를 최소화 하는것이 목표입니다. 모델의 예측 값과 실제 값 사이의 오차를 나타내는 손실 함수는 최적화 알고리즘에 의해 극소값을 찾게 됩니다. 이때 사용되는 기법이 경사 하강법 입니다.

경사 하강법에서는 손실 함수의 도함수(즉, 기울기)를 계산하여, 기울기가 0에 가까워지는 지점, 즉 정류점을 찾아 손실을 최소화합니다. 실제로는 손실 함수가 복잡한 고차원 공간에서 정의되기 때문에 다수의 정류점이 있을 수 있고, 딥러닝에서는 보통 전역 최소값(Global Minimum) 또는 지역 최소값(Local Minimum)을 찾으려 합니다.



- 위 그래프는 함수 $f(x) = x^3 - 3x^2 + 2$, 그 도함수 $f'(x)$, 그리고 2차 도함수 $f''(x)$의 그래프를 나타냅니다. 파란선은 원래 함수 $f(x)$, 초록색 점섬은 1차 도함수 $f'(x)$, 빨간색 점선은 2차 도함수 $f''(x)$입니다. 도함수가 0이 되는 점들이 정류점이며, 이곳에서 극값을 찾을 수 있습니다.

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

```
# 함수 정의: f(x) = x^3 - 3x^2 + 2
def f(x):
    return x**3 - 3*x**2 + 2

# 1차 도함수: f'(x)
def f_prime(x):
    return 3*x**2 - 6*x

# 2차 도함수: f''(x)
def f_double_prime(x):
    return 6*x - 6

# x 범위 설정
x = np.linspace(-1, 3, 400)

# 함수값 계산
y = f(x)

# 1차 도함수와 2차 도함수 값 계산
y_prime = f_prime(x)
y_double_prime = f_double_prime(x)

# 그래프 그리기
plt.figure(figsize=(10, 6))
plt.plot(x, y, label="f(x) = x^3 - 3x^2 + 2", color='blue')
plt.plot(x, y_prime, label="f'(x)", color='green', linestyle=
plt.plot(x, y_double_prime, label="f''(x)", color='red', line
plt.axhline(0, color='black',linewidth=1)
plt.title("Function and its derivatives")
plt.xlabel("x")
plt.ylabel("f(x) and derivatives")
plt.legend()
plt.grid(True)
plt.show()

# 경사 하강법 구현 (더 작은 학습률과 기울기 절단 적용)
def gradient_descent(learning_rate=0.001, epochs=100, clip_va
    x = np.random.randn()  # 초기값 무작위 설정
```

```
    for i in range(epochs):
        grad = f_prime(x)  # 기울기 계산 (f'(x))

        # 기울기 절단 적용 (clip_value 이상의 기울기는 clip_value로
        if grad > clip_value:
            grad = clip_value
        elif grad < -clip_value:
            grad = -clip_value

        x = x - learning_rate * grad  # 업데이트
        print(f"Epoch {i+1}: x = {x}, f(x) = {f(x)}")

# 경사 하강법 실행
gradient_descent()
```