미분(derivative)

수학에서 derivative는 미분을 의미하며, 함수의 특정 지점에서의 순간 변화율을 나타내는 도함수를 구하는 과정을 말합니다.

이는 그래프에서 해당 지점에 접하는 직선의 기울기와 같으며, 함수의 변화 정도를 측정하는 데 사용됩니다

미분 & 경사하강법 (Gradient Descent)
인공신경망(AI 모델)을 학습시킨다는 것은 곧 **손실 함수(L)의 최솟값**을 찾는 것입니다. 이 최솟값을 찾기 위해 사용하는 핵심 알고리즘이 바로 경사 하강법(Gradient Descent)이며, 경사 하강법은 **미분**에 전적으로 의존합니다.

부성순강사(bakpak@empas.com)

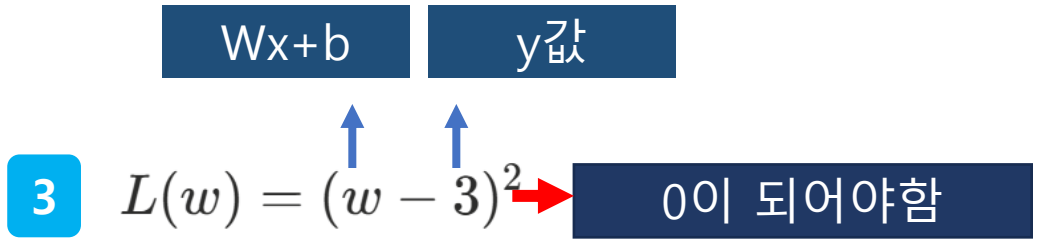
기계학습(Machine Learning)이란? - 기울기와 변화량 이해하기

ML(기계학습이란)

"기울기(=미분값)"을 이용해서 Loss L(w)를
0 또는 최소로 만드는 방향으로 계속 움직이는 과정

- **미분**은 **곡선 위의 한 점에서 접선(tangent line)의 기울기**를 구하기

단계	내용	핵심 키워드
1	데이터 입력 (x) → 예측값 계산 (ŷ)	모델, 가중치 (w)
2	예측과 실제 차이 계산	손실함수 (L(w))
3	그 손실이 얼마나, 어느 방향으로 변하는지 계산	**미분(기울기)**
4	기울기의 **반대 방향**으로 가중치 조정	**경사하강법** (Gradient Descent)
5	반복해서 (L(w)) 가 최소가 되면 → 모델 학습 완료	**최적화** (Optimization)



L(w): 함수값의 변화량, 값이 얼마나 변했는가?

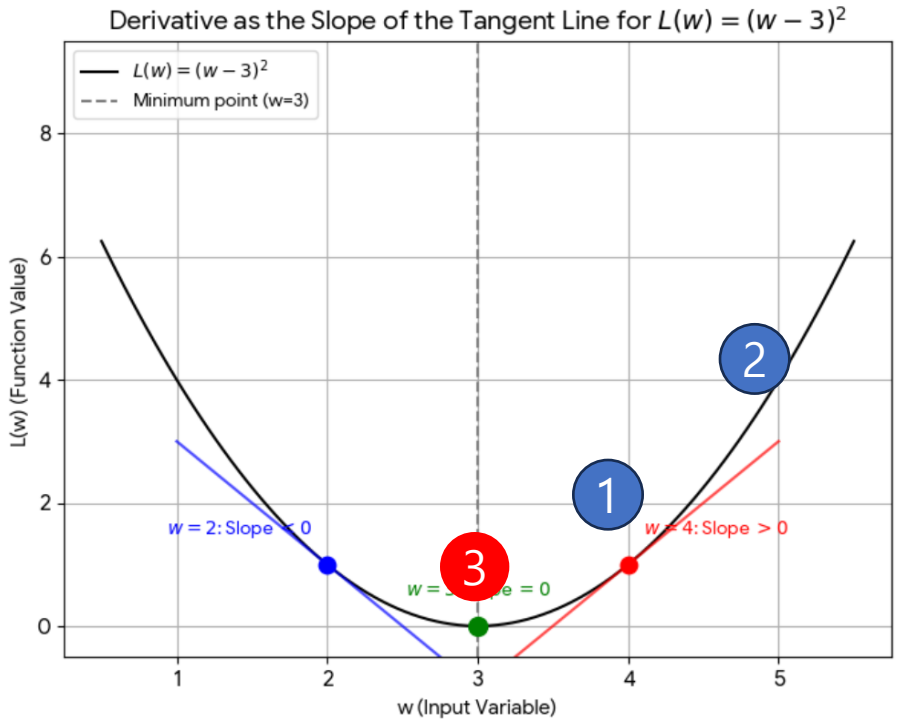
손실 함수(Loss Function) 로 해석하면,

- **모델이 얼마나 틀렸는가(오차)** 를 수치로 표현하는 식
- 따라서 학습의 목표는 이 손실 함수의 **값을 최소화(Minimize)** 하는,
  즉 **손실이 가장 작은 지점(최솟값, Minimum)** 을 찾는 것입니다.

부성순 bakpak@empas.com

기계학습(Machine Learning)이란? - 기울기와 변화량 이해하기

*참고: 미분



3 $L(w) = (w-3)^2$

양의 기울기 (Positive Slope)

- **지점**: w=4(빨간색)
- **접선**: 오른쪽 위로 **상승**하는 기울기입니다.
- **의미**: w를 조금 더 증가시키면(오른쪽으로 가면),
  함수 값 L(w)는 **증가**합니다.
  즉, 이 지점에서는 함수가 상승하는 중

W가 4일때 L(w)=(4-3)=**1** (양수는 상승함을뜻함) → 1
W를 조금더 증가 5 L(w)=(5-3)=4 → 2

영의 기울기 (Zero Slope) 3

- **지점**: w=3 (녹색, 최솟값)
- **접선**: 완전히 **수평**한 기울기입니다.
- 의미: 이 지점에서 함수는 더 이상 증가하지도,
  감소하지도 않습니다. 잠시 멈춘 상태이며,
  이는 함수의 최솟값 (또는 최댓값)이 되는 지점입니다

부성순 bakpak@empas.com

기계학습(Machine Learning)이란? - 순간변화율(미분)

평균변화율(average rate of change)

$$\text{평균변화율} = \frac{L(w_2) - L(w_1)}{w_2 - w_1}$$

평균변화율은 "두 점 사이의 기울기"입니다.

- $w_1 = 3$
- $w_2 = 4$

라고 하면,

평균 변화율: 1

$$L(w) = (w-3)^2$$

$$L(4) = (4-3)^2 = 1, \quad L(3) = (3-3)^2 = 0$$

$$\frac{L(4) - L(3)}{4-3} = \frac{1-0}{1} = 1$$

3에서 4로 갈 때 이 함수의 평균적인 변화 속도(기울기)는 1입니다.

평균변화율 → 미분(순간변화율)

"평균"이 아니라 "순간"의 변화를 알고 싶다고 할때 즉, $w = 3$ **바로 근처에서의 변화율**.

함수 f(x)에서 두 점 x와 x+h를 잡았을 때의 평균 변화율 공식

h는 두 점 사이의 간격(거리) $\text{평균 변화율} = \frac{f(x+h) - f(x)}{(x+h) - x} = \frac{f(x+h) - f(x)}{h}$

극한을 통한 미분 (순간기울기)

순간기울기, 즉 **미분**은 이 간격 h를 **0으로 극한을 취하는** 과정을 통해 얻어집니다.

평균 변화율의 개념을 **극한**을 통해 **순간의 변화율**로 **진화**

• **아이디어**: 두 점이 점점 가까워져서 **하나의 점**이 되는 순간의 기울기를 구하는 것입니다.

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

인공신경망 학습 관점에서의 의미 : 인공신경망에서 평균 변화율이 아닌 미분을 사용하는 이유는 바로 **극도로 정확한 방향 정보**가 필요하기 때문입니다.

→ **평균 변화율**: 넓은 구간의 평균적인 정보를 주므로, 최솟값 근처에서 **정확도가 떨어져** 최적점을 지나치거나 맴돌 수 있습니다.

→ **미분**: h→0 극한을 통해 **현재 바로 그 지점에서의** 완벽한 방향을 알려주므로, 경사 하강법이 **가장 효율적이고 안정적으로** 최솟값에 도달하도록 돕습니다.

부성순 bakpak@empas.com

기계학습(Machine Learning)이란? - 순간변화율(미분), 도함수

도함수 (Derivative Function)란?

- **도함수**는 미분(순간기울기)을 구하는 과정을 거쳐 도출된 함수라는 의미입니다.
- 원래 함수 f(x)에 대해, 모든 x 값에서의 **순간기울기**를 계산할 수 있도록 만들어진 **새로운 함수**입니다

$$L(w) = (w-3)^2$$

$$L(w) = (w-3)^2 = (w-3)(w-3)$$

$$= w^2 - 6w + 9$$

∴ 함수 $L(w) = w^2 - 6w + 9$를 미분하여 도함수

$$\frac{dL}{dw} = L'(w) = \frac{d}{dw}(w^2 - 6w + 9)$$

1. 덧셈/뺄셈 미분 규칙 적용

미분은 각 항에 대해 독립적으로 적용됩니다.

$$L'(w) = \frac{a}{dw}(w^2) - \frac{a}{dw}(6w) + \frac{a}{dw}(9$$

3. 최종 도함수 결합

각 항의 미분 결과를 합쳐서 도함수를 완성합니다:

$$L'(w) = 2w - 6 + 0$$

$$\mathbf{L'(w) = 2w - 6}$$

2. 각 항별 미분 (중간 과정)

- **첫 번째 항:** w^2
  - **규칙:** 멱의 법칙 $\left(\frac{d}{dx}(x^n) = nx^{n-1}\right)$
  - **계산:** $2 \cdot w^{2-1} = \mathbf{2w}$
- **두 번째 항:** $-6w$
  - **규칙:** 상수배 법칙 $\left(\frac{d}{dx}(cx) = c\right)$
  - **계산:** $-6 \cdot 1 = \mathbf{-6}$
- **세 번째 항:** $+9$
  - **규칙:** 상수 법칙 $\left(\frac{d}{dx}(c) = 0\right)$
  - **계산:** $\mathbf{0}$

부성순 bakpak@empas.com

기계학습(Machine Learning)이란? - 순간변화율(미분), 도함수

$$L'(w) = 2w - 6 + 0 \qquad \mathbf{L'(w) = 2w - 6}$$

```python
import torch

# 1. 초기 설정
w = torch.tensor(4.0, requires_grad=True)

# 2. 기울기 계산
# L(w) = (w - 3)^2
L = (w - 3) ** 2

# 3. 역전파(Backpropagation) 실행
# 도함수 실행
# L.backward()를 호출하면 L이 w에 대해
# 미분된 값(기울기)값 +2가
#  w.grad에 저장됩니다.

L.backward()

print(f'L(w)=w2-w1: {L.item():.6f}')
print(f'업데이트전 w값:  {w.item()}')
print(f'도함수적용(미분): {w.grad.item()}')

```

미분계산용 변수로 정의
False하면 일반상수

```
L(w)=w2-w1: 1.000000
업데이트전 w값:  4.0
도함수적용(미분): 2.0
```

$$L'(4) = 2(4) - 6 = 8 - 6 = +\mathbf{2}$$
$$L'(3) = 2(3) - 6 = 6 - 6 = \mathbf{0}$$

도함수($L'(w)$)의 결과를 보고, 모델이 $-$ 방향이나 $+$ 방향으로 $\mathbf{w}$ 값(가중치)을 **'알아서' 조정**하는 것이 바로 경사 하강법(Gradient Descent) 의 기본 원리입니다.

새로운 w = 기존 w − (학습률 × 도함수 값)

- **도함수 값:** $+2$ (현재 위치의 기울기)
- **w의 변화량:** -2 (최솟값을 향해 움직인 거리)

$+2$라는 도함수 값을 이용하여 -2만큼 이동

W값4 -2 =>2로변경됨

```python
import torch.optim as optim
learning_rate = 1.0  # 학습률을 1.0으로 설정
optimizer = optim.SGD([w], lr=learning_rate)
optimizer.step()

print(f'역전파된 w값:  {w.item()}')

```

```
역전파된 w값:  2.0
```

부성순 bakpak@empas.com

기계학습(Machine Learning)이란? - 순간변화율(미분), 도함수

(참고: requires_grad=False로 했을때)

```python
import torch

# 1. 초기 설정
w = torch.tensor(4.0, requires_grad=False)

# 2. 기울기 계산
# L(w) = (w - 3)^2
L = (w - 3) ** 2

# 3. 역전파(Backpropagation) 실행
# 도함수 실행
# L.backward()를 호출하면C LOI w에 대해
# 미분된 값(기울기)값 +2가
#  w.grad에 저장됩니다.

L.backward()

print(f'L(w)=w2-w1: {L.item():.6f}')
print(f'업데이트전 w값:  {w.item()}')
print(f'도함수적용(미분): {w.grad.item()}')

```

```
---------------------------------------------------------------------------
RuntimeError                              Traceback (most recent call last)
/tmp/ipython-input-2079245367.py in <cell line: 0>()
     14 #  w.grad에 저장됩니다.
     15
---> 16 L.backward()
     17
     18 print(f'L(w)=w2-w1: {L.item():.6f}')

                              2 frames
/usr/local/lib/python3.12/dist-packages/torch/autograd/graph.py in _engine_run_backward(t_outputs, *args, **kwargs)
    827         unregister_hooks = _register_logging_hooks_on_whole_graph(t_outputs)
    828     try:
--> 829         return Variable._execution_engine.run_backward(  # Calls into the C++ engine to run the backward pass
    830             t_outputs, *args, **kwargs
    831         )  # Calls into the C++ engine to run the backward pass

RuntimeError: element 0 of tensors does not require grad and does not have a grad_fn
```

미분의 역할을 수식으로 표현

1️⃣ 방향 결정: 최적의 경로 찾기

손실함수(loss function) $L(w)$ 가 있을 때,
현재 위치 w 에서의 **기울기(미분값)** 은 다음과 같습니다.

$\frac{dL}{dw}$ 이 값은 함수의 **오르막 방향**을 알려줍니다.
즉, 손실이 **증가하는 방향**입니다.

- η: 학습률(learning rate, 이동 크기 조절)
- $\frac{dL}{dw} > 0$: 오르막길 → w 감소시켜야 함
- $\frac{dL}{dw} < 0$: 내리막길 → w 증가시켜야 함

따라서 손실을 **줄이기 위해(내리막길로 내려가기 위해)**
-> **미분값의 반대 방향**으로 이동해야 합니다.

$$w_{\text{new}} = w_{\text{old}} - \eta \frac{dL}{dw}$$

2️⃣ 업데이트 크기 결정: 학습 속도 조절

미분의 **절댓값(크기)** 은 언덕의 가파름을 의미합니다.

$\left|\frac{dL}{dw}\right|$ 이 클수록 언덕이 가파르고, $\left|\frac{dL}{dw}\right|$ 이 작을수록 평평한 지역입니다.

$$\Delta w = -\eta \frac{dL}{dw}$$

- $|\frac{dL}{dw}|$ **이 크면 →** $|\Delta w|$ 도 커져서 빠르게 이동
- $|\frac{dL}{dw}|$ **이 작으면 →** $|\Delta w|$ 도 작아져서 미세하게 조정

자연상수와 함수

- **Exponential Function**의 약자입니다. 수학에서는 다음과 같이 정의됩니다.

$$\exp(\mathbf{x}) = \mathbf{e}^{\mathbf{x}} \quad e\text{는 약 } 2.71828..$$

- 미분했을 때 **자기 자신이 되는** 매우 특별한 함수 $\frac{d}{dx}\exp(x) = \exp(x)$

이러한 **단순한 미분 특성** 덕분에, exp 함수는 시그모이드(Sigmoid)나 소프트맥스(Softmax)와 같은 인공신경망의 핵심 활성화 함수와 손실 함수에 널리 사용되어 기울기(미분값) 계산을 효율적으로 만듭니다.

부성순강사(bakpak@empas.com)

미분과 자연상수

인공신경망에서 미분은 학습의 핵심이므로, **미분이 잘되는 함수**가 필요합니다.
시그모이드 함수는 $\sigma(x) = \frac{1}{1+e^{-x}}$ 형태로, 자연상수 e를 사용해 **부드럽고 연속적인 미분값**을 제공합니다.
이 덕분에 신경망이 가중치 w를 조정할 때, **기울기(gradient)** 를 안정적으로 계산할 수 있습니다.

만약 자연상수 e를 사용하지 않는다면,
함수의 **변화율(기울기)** 이 일정하지 않거나 불연속적으로 변해 **매끄러운 미분 계산이 어려워집니다.**
그 결과, 신경망이 학습 중에 **가중치를 안정적으로 조정하지 못하고** 최적점을 찾기 어려워집니다.

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

e^{-x} 때문에 큰 음수 → 0 근처

큰 양수 → 1 근처로 **부드럽게 눌러주는** 효과

입력 x	계산식	출력 $\sigma(x)$
-2	$\frac{1}{1+e^{2}}$	0.119
0	$\frac{1}{1+e^{0}}$	0.500
2	$\frac{1}{1+e^{-2}}$	0.881

$$\text{Softmax}(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$$

부성순 bakpak@empas.com

미분과 자연상수- (참고) 자연상수 출력값 확인

```python
import matplotlib.pyplot as plt

# Given x values
x_vals = [-6, -3, -1, 0, 1, 3, 6]
y_vals = [math.exp(x) for x in x_vals]

# Print values for the user
for x, y in zip(x_vals, y_vals):
    print(f"x = {x:>2}, exp(x) = {y}")

# Plot
plt.figure()
plt.plot(x_vals, y_vals, marker='o')
plt.title("exp(x) for selected values")
plt.xlabel("x")
plt.ylabel("exp(x)")
plt.grid(True)
plt.show()
```

STDOUT/STDERR

```
x = -6, exp(x) = 0.0024787521766663585
x = -3, exp(x) = 0.049787068367863944
x = -1, exp(x) = 0.36787944117144233
x =  0, exp(x) = 1.0
x =  1, exp(x) = 2.718281828459045
x =  3, exp(x) = 20.085536923187668
x =  6, exp(x) = 403.4287934927351
```

미분과 자연상수- exp 자연상수는 미분해도 같음 확인

```python
import torch
import torch.optim as optim

# 1. 초기 설정
w = torch.tensor(4.0, requires_grad=True)  # w = 1부터 시작

# 2. 손실함수 정의 : L = exp(w)
L = torch.exp(w)

# 3. 역전파 수행 (dL/dw 계산)
L.backward()
print(f'L(w)=w2-w1: {L.item():.6f}')
print(f'업데이트전 w값:  {w.item()}')
print(f'도함수적용(미분): {w.grad.item()}')


# 4. 경사하강법으로 w 업데이트
learning_rate = 1.0
optimizer = optim.SGD([w], lr=learning_rate)
optimizer.step()   # w ← w - lr * grad

print(f"업데이트 후 w값: {w.item():.6f}")

```

```
L(w)=w2-w1: 54.598148
업데이트전 w값:  4.0
도함수적용(미분): 54.598148345947266
업데이트 후 w값: -50.598148
```

```python
import torch
import torch.optim as optim

# 1. 초기 설정
w = torch.tensor(4.0, requires_grad=True)

# 2. 손실함수 정의 : L = exp(w)
#L = torch.exp(w)
L=w

# 3. 역전파 수행 (dL/dw 계산)
L.backward()
print(f'L(w)=w2-w1: {L.item():.6f}')
print(f'업데이트전 w값:  {w.item()}')
print(f'도함수적용(미분): {w.grad.item()}')


# 4. 경사하강법으로 w 업데이트
learning_rate = 1.0
optimizer = optim.SGD([w], lr=learning_rat
optimizer.step()   # w ← w - lr * grad

print(f"업데이트 후 w값: {w.item():.6f}")

```

```
L(w)=w2-w1: 4.000000
업데이트전 w값:  4.0
도함수적용(미분): 1.0
업데이트 후 w값: 3.000000
```

미분과 자연상수- exp 자연상수는 미분해도 같음 확인

인공신경망의 많은 부분(시그모이드, 소프트맥스 등)은
e^x 형태의 함수 사용

역전파는 "오차가 얼마나 영향을 미쳤는지"를
미분으로 거꾸로 전달하는 과정

그런데 어떤 함수는 미분할 때마다 모양이 바뀌어 복잡해지지만,

e^x**는 미분해도 그대로** e^x**이므로** 계산이 깔끔하고,

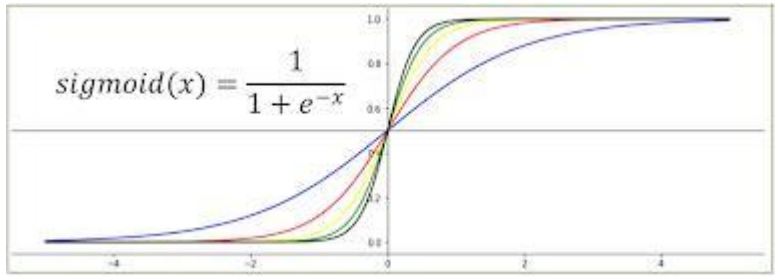
오차를 따라 매끄럽게 파라미터를 조정가능함.

함수	미분 결과	특징
x^2	$2x$	모양이 달라짐
$\sin(x)$	$\cos(x)$	모양이 바뀜
e^x	e^x	모양이 안 바뀜!! 💡

부성순 bakpak@empas.com

시그모이드함수

- **시그모이드 함수 공식:** $sigmoid(x) = \frac{1}{1 + e^{-x}}$
- **특징:** 시그모이드 함수는 미분 가능하며, 주로 로지스틱 회귀(Logistic Regression)와 같은 분류 문제에 사용됩니다.
- **그래프 해석:** 그래프는 여러 시그모이드 함수를 보여주며, 각 곡선은 기울기가 다른 것을 나타냅니다.
- **비선형 함수:** 시그모이드 함수는 직선 하나로 표현할 수 없는 비선형 함수에 속합니다.



부성순강사(bakpak@empas.com)

시그모이드 기본 계산

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

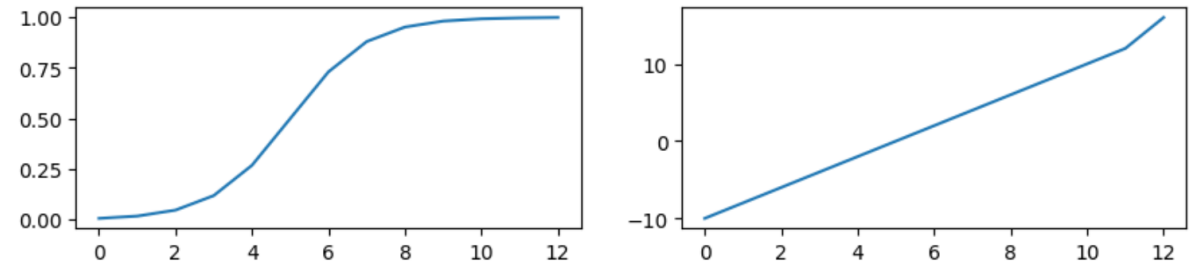
입력 x	계산식	출력값
-2	$\frac{1}{1+e^{-(-2)}} = \frac{1}{1+e^{2}} = \frac{1}{1+7.389}$	**0.119**
0	$\frac{1}{1+e^{0}} = \frac{1}{1+1}$	**0.5**
2	$\frac{1}{1+e^{-2}} = \frac{1}{1+0.135}$	**0.881**

$x = -2 \Rightarrow 0.12, \quad x = 0 \Rightarrow 0.5, \quad x = 2 \Rightarrow 0.88$

음수는 0에 가까워지고, 양수는 1에 가까워집니다.

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

x =np.array([-5,-4,-3,-2,-1,0,1,2,3,4,5,6,8])
y_linear=2*x
y_sigmoid= 1 / (1 + np.exp(-x)) # 무조건 0~1사이값이됨
print(y_sigmoid.min(), '~',y_sigmoid.max())

plt.figure(figsize=(10,2))

plt.subplot(1,2,1)
plt.plot(y_sigmoid)
plt.subplot(1,2,2)
plt.plot(y_linear)
```

```
0.0066928509242848554 ~ 0.9996646498695336
[<matplotlib.lines.Line2D at 0x78bf17750440>]
```



부성순 bakpak@empas.com

시그모이드 - 기울기소실

시그모이드 함수의 극한 현상: 기울기 소실 (Vanishing Gradient)

"정규화(Normalization)" 가 필요함

입력 z의 값이 **매우 크거나 매우 작을 때** 미분값(기울기)이 **거의 0에 수렴**하는 극한 현상이 발생합니다.
이것이 바로 학습을 방해하는 **기울기 소실**(Vanishing Gradient) 문제입니다.

```python
import torch
import torch.nn.functional as F

# 시그모이드 함수와 그 미분 (σ'(x) = σ(x)(1 - σ(x)))
x = torch.tensor([-20., -4., -2., 0., 2., 4., 20.])

# 1. 시그모이드 값
y = torch.sigmoid(x)

# 2. 미분값(기울기)
grad = y * (1 - y)

# 결과 출력
for i, (val, g) in enumerate(zip(y, grad)):
    print(f"x={x[i].item():>5} → sigmoid={val.item():.6f},  gradient={g.item():.8f}")

```

```
x=-20.0 → sigmoid=0.000000,  gradient=0.00000000
x= -4.0 → sigmoid=0.017986,  gradient=0.01766271
x= -2.0 → sigmoid=0.119203,  gradient=0.10499358
x=  0.0 → sigmoid=0.500000,  gradient=0.25000000
x=  2.0 → sigmoid=0.880797,  gradient=0.10499363
x=  4.0 → sigmoid=0.982014,  gradient=0.01766273
x= 20.0 → sigmoid=1.000000,  gradient=0.00000000
```

방법	역할
Batch Normalization	각 배치의 입력을 평균 0, 분산 1로 조정
Layer Normalization	각 레이어별로 입력을 스케일 조정
Weight Initialization	처음 가중치를 작게 설정해서 x 값이 폭주하지 않게 함

시그모이드 - 기울기소실

1. 문제의 원인: 미분의 범위 제한

시그모이드 함수 sigma(z)는 아무리 복잡한 값을 넣어도 그 결과가 **0과 1 사이**로 압축됩니다. 이 함수의 **미분값(기울기)은 최대** 0.25를 절대 넘을 수 없습니다.

현상: 기울기의 증발

신경망이 깊어질수록, 학습을 위해 앞쪽 층으로 기울기 정보가 전달될 때 연쇄 법칙(Chain Rule)에 따라 이 작은 미분값들이 계속 **곱해집니다**

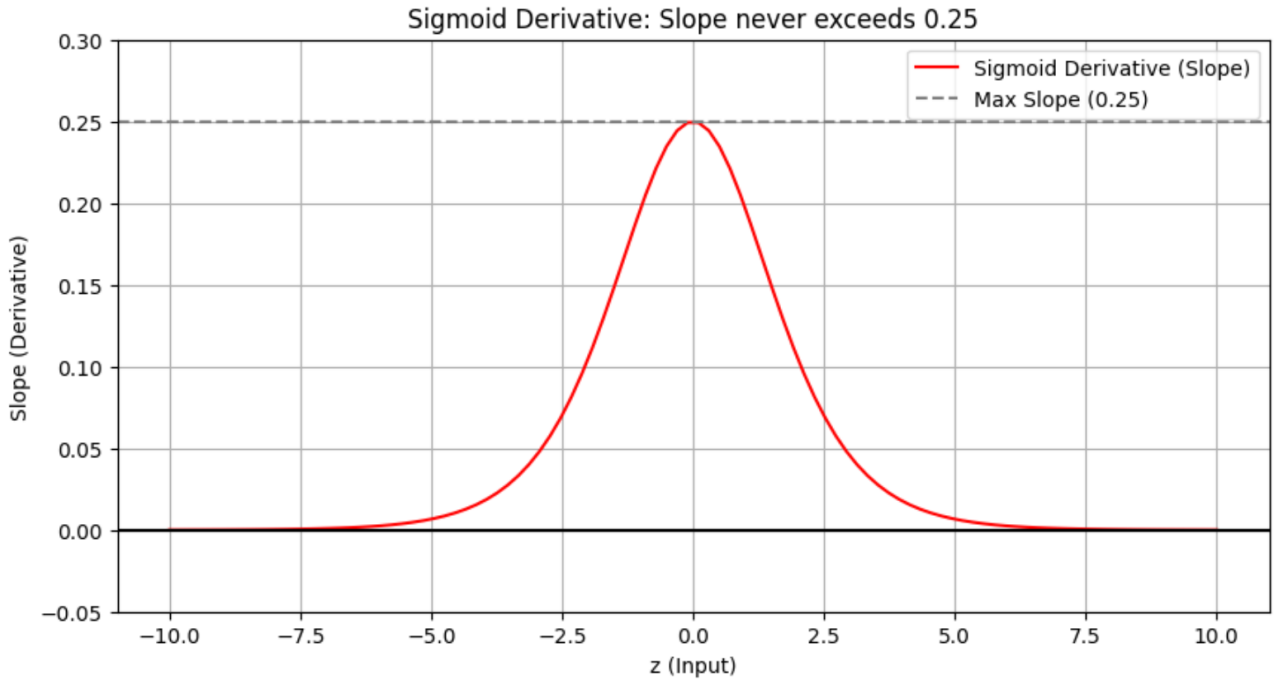
$$\text{기울기} \approx \mathbf{0.25} \times \mathbf{0.25} \times \mathbf{0.25} \times \cdots$$

이처럼 0.25 이하의 작은 숫자들이 여러 번 곱해지면, 최종적으로 기울기 정보는 **거의 0에 가까워지면서 사라져버립니다.**

부성순 bakpak@empas.com

시그모이드 - 기울기소실

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

# 시그모이드 함수 정의 (파이토치 사용)
def sigmoid(z):
    return torch.sigmoid(z)

# 미분값(기울기) 계산 함수 정의
def calculate_slope(z_value):
    # 1. z 텐서 정의 (미분 추적 활성화)
    z = torch.tensor(z_value, requires_grad=True)

    # 2. 시그모이드 함수 계산
    s = sigmoid(z)

    # 3. 역전파(backward)로 기울기 계산
    s.backward()

    # 4. 기울기 값 반환
    return z.grad.item()

# 1. z 값 범위 설정 (넓은 범위: -10부터 10까지)
z_values = np.linspace(-10, 10, 100)

# 2. 각 z 값에 대한 기울기 계산
slopes = [calculate_slope(z) for z in z_values]

```



계산된 최대 기울기 값: 0.2494

부성순 bakpak@empas.com