

# Attention과 AutoEncoder

연준모

1. Attention
2. AutoEncoder

# 1. Attention 정규화?

## Attention?

들어오는 데이터의 양은 엄청나게 많다

특히, 시퀀스 데이터는 입력 데이터가 많으면 먼저 들어온 정보가 소실된다는 것을 우리는 배웠다.(RNN)

Attention은 모든 정보를 동일하게 보지 않고, 현재 작업과 **관련된 중요한 정보에만 집중**하여  
가중치를 부여하는 방법이다

이는, 데이터 베이스 검색과 유사한데, 다음과 같은 값을 사용해서 계산하게 된다

**Query (Q)** : 내가 찾고자 하는 정보 (검색어)

**Key (K)** : 검색 대상이 되는 정보의 식별자 (데이터베이스의 인덱스)

**Value (V)** : 식별자에 해당하는 실제 정보 (데이터베이스의 내용)

Q가 모든 K와 얼마나 유사한지(관련 있는지) 계산하고, 이 유사도를 가중치로  
삼아 V들의 중합을 계산한다.

# 1. Attention 정규화?

## Attention?

일반적으로(특히 Attention이 주로 사용되는 Transformer에선) 행렬곱으로 이루어 진다

Q (Query 행렬):  $(N_q, d_k)$  -  $N_q$ 개의 쿼리 벡터

K (Key 행렬):  $(N_k, d_k)$  -  $N_k$ 개의 키 벡터

V (Value 행렬):  $(N_k, d_v)$  -  $N_k$ 개의 밸류 벡터

Transformer 모델에선 Self-Attention을 사용하게 되는데,(다음주 예정), 여기서는

$Q, K, V$ 가 모두 동일한 입력  $X$ 에 각각 다른 가중치 행렬  $W_Q, W_K, W_V$ 를 곱하여 생성된다.

$Q = XW_Q, K = XW_K, V = XW_V$ . 여기서  $W$  행렬들은  $X$ 를 다른 기저로 투영하는 선형 변환 이다.

# 1. Attention 정규화?

ONECLICK AI

## Attention?

식은 다음과 같다

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

비유

V = 도서관에 있는 모든 책의 내용 그 자체

K = 모든 책의 제목, 키워드

Q = 나의 리포트 주제 (고양이와 관련된 역사)

## 1단계: 유사도 계산

$$\text{Scores} = QK^T$$

**연산** :  $Q$  ( $N_q, d_k$ ) 행렬과  $K$ 의 전치(Transpose) 행렬  $K^T$  ( $d_k, N_k$ )의 곱.

**결과** :  $\text{scores}$  행렬 ( $N_q, N_k$ ).

이 행렬 곱의 결과  $\text{Scores}_{ij}$ 는  $i$ 번째 쿼리 벡터  $q_i$ 와  $j$ 번째 키 벡터  $k_j$ 의 내적이다.

$$\text{Scores}_{ij} = q_i \cdot k_j$$

벡터 내적  $a \cdot b = \|a\| \|b\| \cos(\theta)$ 는 두 벡터의 유사도(Similarity)를 측정한다  
 $q_i$ 와  $k_j$ 가 얼마나 정렬되어 있는지(방향이 비슷한지)를 나타낸다

# 1. Attention 정규화?

## Attention?

### 1단계: 유사도 계산

$$Scores = QK^T$$

**모든 Q와 모든 K를 한 번에 비교**하는 매우 효율적인 방법

세로 벡터 Q와 전치를 통해 가로가 된 가로벡터 K의 곱 => 모든 원소 조합에 대한 연산 가능

$$Scores_{ij} = q_i \cdot k_j \quad \text{내적을 해야 얼마나 유사한지 측정할 수 있다}$$

(내 주제 : 고양이와 관련된 역사) · (책 제목 : 고양이의 역사) => 높은 점수(90 점)

(내 주제 : 고양이와 관련된 역사) · (책 제목 : 군주론) => 낮은 점수(15점)

# 1. Attention 정규화?

## Attention?

2단계: 가중치 계산 with Softmax

$$A = \text{softmax}\left(\frac{\text{Scores}}{\sqrt{d_k}}\right)$$

연산 : 스케일링:  $\text{Scores}$  행렬의 모든 요소를  $\sqrt{d_k}$ 로 나눈다.

이는 내적 값이 너무 커져서 Softmax의 기울기가 0이 되는 것을 방지하기 위한 안정화 장치이다.

점수(90)가 너무 크면, Softmax가 "이 책 100%, 나머지 0%"처럼 너무 극단적인 결과가 나온다

$\sqrt{d_k}$  (키 벡터의 차원)라는 값으로 점수들을 나눠서, 점수 차이를 진정시키는 **안정화 장치**

(예: 90 → 9.0, 15 → 1.5)

**Softmax:** 스케일링된  $\text{Scores}$  행렬의 각 행에 대해 Softmax 를 적용한다

(9.0, 1.5) → **(85%, 15%)**

고양이의 역사 책에 85% 집중하고, 군주론 책에는 15%만 집중하라는 명확한 집중도 레시피가 만들어 진다

# 1. Attention 정규화?

## Attention?

2단계: 가중치 계산 with Softmax

결과:  $A$  (Attention Weight 행렬) ( $N_q, N_k$ ).

Softmax는 각 행의 값( $q_i$ 가 모든  $k_j$ 와 갖는 유사도 점수)을 0과 1 사이의 값으로 정규화하며, **각 행의 합이 1**이 되도록 만든다는 것을 우리는 이미 알고있다.

$A_{ij}$ 는  $i$ 번째 쿼리가  $j$ 번째 밸류에 **얼마나 집중해야 하는지**를 나타내는 **확률적 가중치**가 된다.



# 1. Attention 정규화?

## Attention?

### 3단계: 가중치 계산

$$\text{Output} = A V$$

**연산:**  $A$  ( $N_q, N_k$ ) 행렬과  $V$  ( $N_k, d_v$ ) 행렬의 곱.

$A$  : 2 단계에서 만든 집중도 레시피(Softmax 출력)

$V$  : 실제 책의 내용물

둘을 곱하므로 가중평균을 만든다

$$\text{Output}_i = \sum A_{ij} v_j$$

고양이의 역사 내용 85% + 군주론 15%

이는 고양이라는 단어의 의미를 군주론 관점까지 고려하여 풍부하게 재해석한 결과물이 나온다

# 1. Attention 정규화?

## Attention?

결과: *Output* 행렬 ( $N_q, d_v$ ).

이것이 어텐션의 최종 결과물이다.

*Output*의  $i$ 번째 행  $Output_i$ 는  $V$  행렬의 모든 행  $v_j$ (밸류 벡터)들의 가중 평균(Weighted Average)으로 계산된다.

$$Output_i = \sum_{j=1}^{N_k} A_{ij} v_j$$

즉,  $i$ 번째 쿼리( $q_i$ )에 대한 응답( $Output_i$ )은,  $q_i$ 와의 관련도( $A_{ij}$ )가 높은 밸류( $v_j$ )는 많이 가져오고 관련도가 낮은 밸류는 적게 가져와 모두 합친 **정보의 종합** 벡터이다.

## Attention?

### 결론

1. Q(내 질문)가 K(키워드)와 얼마나 비슷한지 점수를 매기고 (1단계:  $QK^T$ )
2. 이 점수를 집중할 비율(%)로 변환한 뒤 (2단계: Softmax)
3. 이 비율(A)대로 V(실제 내용물)들을 섞어서 새로운 종합 정보(Output)를 만드는 과정

## Attention?

### 결론

어텐션은 내적(Dot Product)을 통해 유사도를 측정하고, 이 유사도를 가중치 삼아 행렬 곱을 통해 가중합을 계산하는, 미분 가능한 정보 추출 메커니즘이다.  
모든 과정이 행렬 연산으로 구성되어 GPU에서 매우 효율적으로 병렬 처리될 수 있다.

정보의 병목 현상(치매)을 해결하고 **장기 의존성(Long-Term Dependency)** 문제를 극복하기 위해서 존재한다  
트랜스포머 등장 이후의 아키텍처에서 주로 사용한다.

다음 시간에, 이 어텐션의 발전형인 Self Attention, MHA, CLS 토큰 등을 알아볼 것이다.

또한, 실사용 시 SDPA, earge 가 있는데, 이 내용은 전부 트랜스포머 하면서 알아볼 것이다.

## 2. AutoEncoder 알아보기 어떻게 생겼을까?

ONECLICK AI

### AutoEncoder

데이터의 효율적인 압축 표현(Representation) 학습을 위한 비지도 학습이다  
대칭적인 U자형 구조와 스킵 연결(Skip Connections)을 핵심

인코더 (수축 경로)

입력 데이터  $x$ 를 원본보다 훨씬 저차원인 잠재 벡터  $z$ 로 압축

디코더 (확장 경로)

압축된  $z$ 로부터 원본  $x$ 와 최대한 유사한  $\hat{x}$ 를 복원

스킵 연결 (인코더, 디코더를 연결)

<https://dl.acm.org/doi/10.5555/1756006.1953039>

<https://arxiv.org/abs/2102.08012>

## 2. AutoEncoder 알아보기 어떻게 생겼을까?

ONECLICK AI

### AutoEncoder

핵심은 병목(Bottleneck) $z$ .

$z$ 가  $x$ 를 완벽히 복원할 수 있으려면,  $z$ 는  $x$ 의 가장 중요하고 본질적인 특징(essence)만을 담고 있게 된다

=> 단순 압축을 넘어 데이터의 기저에 깔린 매니폴드(manifold)를 학습하게 되는거다.

## 2. AutoEncoder 알아보기 어떻게 생겼을까?

ONECLICK AI

### Manifold?

한마디로 매니폴드는 "**데이터가 실제로 살아가는 숨겨진 저차원 공간**" 또는 데이터의 본질적인 규칙(Rule)

3D 공간 속 A4 용지를 구겨 만든 공이 있다 했을 때,  
구겨진 종이 표면이 바로 **2D 매니폴드**이다.

**데이터:** 우리가 가진 데이터는 이 구겨진 종이 위에만 찍힌 점들이다.

이게 뭘 말이나 3차원 공간의 데이터는 3개의 값을 가진다(H, W, C)  
데이터는 3D 공간에 흩어져 있는 것처럼 보이지만(고차원), 실제로는 2D 종이 표면(저차원 매니폴드)이라는 '규칙'을 따르고 있다.

## 2. AutoEncoder 알아보기 어떻게 생겼을까?

ONECLICK AI

### Manifold In AutoEncoder?

**입력 (x):** '구겨진 종이' 위의 한 점의 **3D 좌표** ( $x, y, z$ ).

(예: MNIST 이미지 784개 픽셀 값)

**인코더 (Encoder):** 이 3D 좌표를 보고, '구겨진 종이'를 짹 펴서 이 점이 원래 종이의 어디에 있었는지 2D 좌표 ( $u, v$ )를 찾아낸다.

**잠재 벡터 (z):** 인코더가 찾아낸 **핵심 2D 좌표** ( $u, v$ ). 이것이 바로 데이터의 '본질 (essence)' 이다.

**디코더 (Decoder):** 2D 좌표 ( $u, v$ )를 받고, 종이를 **원래대로 다시 구겨서** 이 점의 3D 좌표 ( $x, y, z$ )를 복원한다.

데이터가 어떤 '규칙'(매니폴드)을 따르는지 학습  
핵심 규칙'을 학습하고, 그것을 잠재 벡터  $z$ 에 담아내는 것



## 2. AutoEncoder 알아보기 어떻게 생겼을까?

ONECLICK AI

### AutoEncoder

- $\mathbf{x} \in R^d$ : 입력 벡터 (예: Flatten된 이미지)
- $\mathbf{z} \in R^k$ : 잠재 표현 (latent vector),  $\mathbf{z}$ 가 위치한 공간을 **잠재 공간(Latent Space)**이라 부른다.
- **병목 제약 (Bottleneck Constraint)**: 핵심은  $k \ll d$  (undercomplete)이다.
- 이는 모델이  $\mathbf{x}$ 를 단순히 복사하는 항등 함수( $\hat{\mathbf{x}} = \mathbf{x}$ )를 배우는 것을 방지하고,
- 유의미한 정보만 압축하도록 강제한다.
- $\hat{\mathbf{x}} \in R^d$ : 재구성 출력
- $\theta = \{\mathbf{W}_e^{(l)}, \mathbf{b}_e^{(l)}\}$ ,  $\phi = \{\mathbf{W}_d^{(m)}, \mathbf{b}_d^{(m)}\}$ : 인코더와 디코더의 학습 가능한 파라미터 (가중치 및 편향)

## 2. AutoEncoder 알아보기 어떻게 생겼을까?

ONECLICK AI

### AutoEncoder

#### 1. 인코더 입력 전 데이터 전처리

$$\mathbf{X} \in R^{C \times H \times W} \quad \text{-- flatten --} \rightarrow \mathbf{x} = \text{vec}(\mathbf{X}) \in R^d, \quad d = C \cdot H \cdot W$$

예: MNIST  $28 \times 28 \times 1 \rightarrow \mathbf{x} \in R^{784}$

이미지를 1자로 길게 편다

공간적 위계(spatial hierarchy)와 지역성(locality) 정보가 소실

## 2. AutoEncoder 알아보기 어떻게 생겼을까?

ONECLICK AI

### AutoEncoder

2. Encoder      **z**는 입력 데이터 **x**가 여러 개의 '압축 레이어(Dense)'를 통과하며 만들어진 **최종 요약 벡터**

L 개의 Dense 레이어가 들어가게 된다.

$$\mathbf{h}^{(0)} = \mathbf{x} \in \mathbb{R}^{d_0}, \quad d_0 = d$$

$$\mathbf{h}^{(1)} = \sigma_1 (\mathbf{W}_1 \mathbf{h}^{(0)} + \mathbf{b}_1) \in \mathbb{R}^{d_1}$$

$$\mathbf{h}^{(2)} = \sigma_2 (\mathbf{W}_2 \mathbf{h}^{(1)} + \mathbf{b}_2) \in \mathbb{R}^{d_2}$$

$\vdots$

$$\mathbf{z} = \mathbf{h}^{(L)} = \mathbf{W}_L \mathbf{h}^{(L-1)} + \mathbf{b}_L \in \mathbb{R}^k$$

- $\mathbf{W}_\ell \in \mathbb{R}^{d_\ell \times d_{\ell-1}}, d_\ell < d_{\ell-1} \rightarrow$  **차원 감소**
- $\sigma_\ell$  : ReLU, Tanh, Sigmoid 등의 **비선형 활성화 함수**
- 마지막 레이어는 **z**가 연속적인 값의 범위를 가질 수 있도록 **활성화 함수가 없거나(선형) Tanh**를 사용하기도 한다.

## 2. AutoEncoder 알아보기 어떻게 생겼을까?

ONECLICK AI

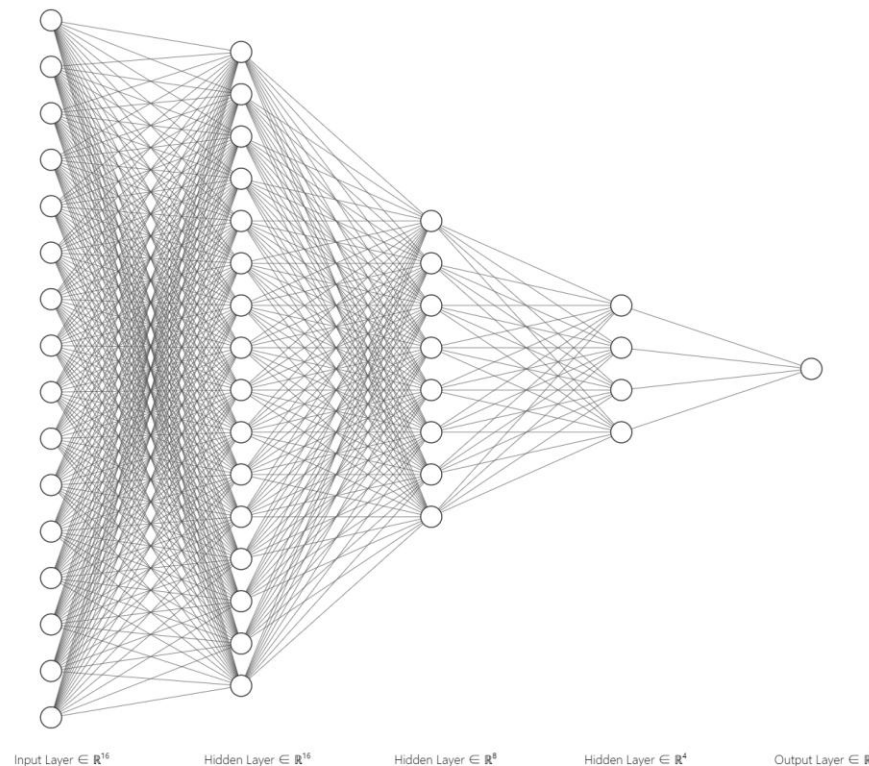
### AutoEncoder

#### 2. Encoder

? Dense Layer로 어떻게 차원을 축소하나요?

다음 레이어의 노드를 줄여버리면 된답니다!

Epoch가 진행되면서 좁은 길'을 통과하기 위해 스스로 가장 효율적인 형태(핵심 요약)로 변할 수밖에 없다



$z$ 는 '복사본'이 아닌 '가장 유의미한 정보'를 담도록 강제된다.  
"어떻게든 32개 숫자 안에 원본의 핵심을 담아 디코더에게 전달해야 하기 때문이다."

## 2. AutoEncoder 알아보기 어떻게 생겼을까?

ONECLICK AI

### AutoEncoder

#### 2. Encoder

#### 비선형성( $\sigma$ )이 왜 필수적인가?

만약 모든 활성화 함수  $\sigma$ 가 선형(즉,  $\sigma(x) = cx$ )이라면, L개의 레이어로 구성된 인코더  $f_\theta(\mathbf{x})$ 전체는  $\mathbf{W}_L \cdots \mathbf{W}_1 \mathbf{x} + \mathbf{b}'$  형태의 **단일 선형 변환**으로 축약된다

L개의 레이어를 쌓아도 **결국 1개의 레이어를 쓴 것과 똑같아 진다**

$$f_\theta(\mathbf{x}) = \mathbf{W}_E \mathbf{x} + \mathbf{b}_E \quad g_\phi(\mathbf{z}) = \mathbf{W}_D \mathbf{z} + \mathbf{b}_D$$

$$\hat{\mathbf{x}} = g_\phi(f_\theta(\mathbf{x})) = \mathbf{W}_D(\mathbf{W}_E \mathbf{x} + \mathbf{b}_E) + \mathbf{b}_D = (\mathbf{W}_D \mathbf{W}_E) \mathbf{x} + (\mathbf{W}_D \mathbf{b}_E + \mathbf{b}_D) = \mathbf{W}' \mathbf{x} + \mathbf{b}'$$

이는 손실 함수로 MSE를 사용할 경우,  
데이터를 가장 잘 표현하는 k차원의 초평면을 찾는 **PCA(주성분 분석)**와 동일해 진다.

비선형 활성화 함수는 AE가 PCA를 넘어

훨씬 더 복잡하고 강력한 비선형 매니폴드를 학습할 수 있게 하는 핵심이다

## 2. AutoEncoder 알아보기 어떻게 생겼을까?

ONECLICK AI

### AutoEncoder

#### 2. Decoder

$$\begin{aligned}\tilde{\mathbf{h}}^{(0)} &= \mathbf{z} \in \mathbb{R}^k \\ \tilde{\mathbf{h}}^{(1)} &= \sigma'_1 \left( \mathbf{W}'_1 \tilde{\mathbf{h}}^{(0)} + \mathbf{b}'_1 \right) \in \mathbb{R}^{d_{M-1}} \\ &\vdots \\ \hat{\mathbf{x}} &= \sigma_{\text{out}} \left( \mathbf{W}'_M \tilde{\mathbf{h}}^{(M-1)} + \mathbf{b}'_M \right) \in \mathbb{R}^d\end{aligned}$$

이 32개 숫자를 어떻게 256개로 펼칠지"에 대한 복원 노하우이며,  
이 역시 학습을 통해 얻어진다

**대칭 구조 (Mirrored Structure):** 일반적으로 디코더는 인코더의 구조를 역순으로 뒤집은 형태 ( $d \rightarrow 256 \rightarrow 32$  였으면  $32 \rightarrow 256 \rightarrow d$ )를 가진다.

**가중치 공유 (Tied Weights):** 때때로 디코더의 가중치를 인코더 가중치의 전치(transpose)로 설정하는 제약( $\mathbf{W}'_\ell = \mathbf{W}_{L-\ell+1}^T$ )을 주기도 한다. 이는 파라미터 수를 절반으로 줄여 과적합을 방지하는 데 도움이 된다.

## 2. AutoEncoder 알아보기 어떻게 생겼을까?

ONECLICK AI

### AutoEncoder

#### 2. Decoder

1차원 벡터를 2D 이미지로 복원 한 후,

$\sigma_{\text{out}}$ : 출력 활성화 함수. 입력  $x$ 의 스케일에 따라 결정된다. 같아야 비교할 수 있다.

$x \in [0,1]$  (예: 정규화된 이미지)  $\rightarrow$  **Sigmoid** (예: 정규화된 흑백)

$x \in [-1,1]$  (예: Tanh 정규화)  $\rightarrow$  **Tanh**

$x \in R$  (예: 일반적인 실수 데이터)  $\rightarrow$  **선형 (활성화 없음)** (예: -5.4, 102.1)

## 2. AutoEncoder 알아보기 어떻게 생겼을까?

ONECLICK AI

### AutoEncoder

$$\mathcal{L}(\mathbf{x}, \hat{\mathbf{x}}) = \begin{cases} \frac{1}{d} \|\mathbf{x} - \hat{\mathbf{x}}\|_2^2 = \frac{1}{d} \sum_{i=1}^d (x_i - \hat{x}_i)^2 & \text{MSE()} \\ -\frac{1}{d} \sum_{i=1}^d [x_i \log \hat{x}_i + (1 - x_i) \log(1 - \hat{x}_i)] & \text{BCE()} \end{cases}$$

사실상 틀린그림 찾기

1. **MSE (L2 Loss):** 재구성 오차가 **가우시안 분포(Gaussian distribution)**를 따른다고 가정하는 것과 같다. 즉,  $p(\mathbf{x}|\mathbf{z}) = \mathcal{N}(\mathbf{x}|g_\phi(\mathbf{z}), \sigma^2 \mathbf{I})$ 를 가정하고 Likelihood를 최대화하는 것과 같다. 이는 실수 값(real-valued) 데이터에 적합하다.

784개 모든 픽셀을 1:1로 비교해서 (원본 픽셀 - 복원 픽셀)<sup>2</sup>의 **평균 오류를 낸다**

일반적인 실수 값 데이터(예: 소리, 주가)에 적합



## 2. AutoEncoder 알아보기 어떻게 생겼을까?

ONECLICK AI

### AutoEncoder

2. **BCE (Cross-Entropy):** 입력 데이터  $\mathbf{x}$ 의 각 픽셀  $x_i$ 가 베르누이 분포(Bernoulli distribution)(즉, 0 또는 1)를 따른다고 가정하는 것과 같다.

이는 픽셀 값이  $[0, 1]$  범위 (확률로 해석)인 이진화된 이미지나 Sigmoid 출력에 적합하다.

픽셀 값이 0 아니면 1에 가까운 '확률'일 때 사용한다. (예: 흑백 이미지) "이 픽셀이 '흰색일 확률'이 0.9라고 예측했는데, 정답(원본)이 1(흰색)이네? 잘했어."라고 점수를 준다

Sigmoid를 사용한  $[0, 1]$  범위의 이미지에 적합

목적:  $\min_{\theta, \phi} \mathcal{L}(\mathbf{x}, g_{\phi}(f_{\theta}(\mathbf{x})))$

$\hat{\mathbf{x}} \approx \mathbf{x}$ 가 되도록  $\theta$ 와  $\phi$ 를 최적화 한다.

이는  $\mathbf{z}$ 가  $\mathbf{x}$ 의 **핵심 정보(압축된 표현)**를 성공적으로 보유해야 함을 의미한다.

## 2. AutoEncoder 알아보기 어떻게 생겼을까?

ONECLICK AI

### AutoEncoder

#### 역전파

손실 함수  $\mathcal{L}$ 을 최소화하기 위해,  $\mathcal{L}$ 을  $\theta$ 와  $\phi$ 의 모든 파라미터(가중치, 편향)로 편미분하여 그래디언트를 계산

1. 출력  $\rightarrow$  디코더 :  $\frac{\partial \mathcal{L}}{\partial \hat{\mathbf{x}}}$  (손실의 그래디언트)가 계산
2. 이 그래디언트는 디코더의 마지막 레이어  $\mathbf{W}'_{\mathbf{M}}$ 부터 역으로 전파되며  $\phi$  (디코더 가중치)를 업데이트

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}'_{\mathbf{M}}} = \frac{\partial \mathcal{L}}{\partial \hat{\mathbf{x}}} \frac{\partial \hat{\mathbf{x}}}{\partial \mathbf{W}'_{\mathbf{M}}}$$

## 2. AutoEncoder 알아보기 어떻게 생겼을까?

ONECLICK AI

### AutoEncoder

#### 역전파

3. 그래디언트는  $\mathbf{z}$ 까지 도달합니다.  $\frac{\partial \mathcal{L}}{\partial \mathbf{z}}$
4.  $\mathbf{z}$ 에 도달한 그래디언트는 이제 인코더를 역으로 통과하며  $\theta$  (인코더 가중치)를 업데이트 한다.

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}_L} = \frac{\partial \mathcal{L}}{\partial \mathbf{z}} \frac{\partial \mathbf{z}}{\partial \mathbf{W}_L}$$

**SGD/Adam**과 같은 옵티마이저는 이 그래디언트를 사용하여 파라미터를 갱신한다.

$$\theta \leftarrow \theta - \eta \frac{\partial \mathcal{L}}{\partial \theta}$$

$$\phi \leftarrow \phi - \eta \frac{\partial \mathcal{L}}{\partial \phi} \text{ (여기서 } \eta \text{는 학습률)}$$

## 2. AutoEncoder 알아보기 어떻게 생겼을까?

ONECLICK AI

### AutoEncoder

레이어	입력 (차원)	출력 (차원)	비고
Input	[1, 28, 28]	-	원본 이미지 (채널, 높이, 너비)
Flatten	[1, 28, 28]	[784]	[cite_start]2D 이미지를 1D 벡터로 펴 [cite: 101, 103]
<b>(Encoder)</b>			
Dense 1	784	256	[cite_start]첫 번째 은닉층 (차원 축소) [cite: 111]
ReLU 1	256	256	[cite_start]비선형 활성화 함수 [cite: 116]
Dense 2 (z)	256	32	[cite_start]z (잠재 벡터, 병목) [cite: 93, 115]
<b>(Decoder)</b>			
Dense 3	32	256	디코더 첫 번째 은닉층
ReLU 2	256	256	
Dense 4	256	784	원본 벡터 차원으로 복원
Reshape	[784]	[1, 28, 28]	1D 벡터를 2D 이미지로 재구성
Sigmoid	[1, 28, 28]	[1, 28, 28]	[cite_start] $\hat{X}$ (복원된 이미지) [cite: 158, 159]

## 2. AutoEncoder 알아보기 어떻게 생겼을까?

ONECLICK AI

### AutoEncoder

**예제: MNIST Dense AE (784 → 32 → 784)**

<https://huggingface.co/gihakkk/AutoEncoder/tree/main>

## 2. AutoEncoder 알아보기 어떻게 생겼을까?

ONECLICK AI

### AutoEncoder

아니 근데 그래서 결국 이미지인데 위치정보 소실되면 안 되는거 아닌가요?

맞아요 그래서 CNN AutoEncoder가 있습니다

<https://arxiv.org/abs/1611.09119>

## 2. AutoEncoder 알아보기 어떻게 생겼을까?

ONECLICK AI

### CNN AutoEncoder

항목	Dense AE	CNN AE
입력	1D 벡터 [B, D]	4D 텐서 [B, C, H, W]
공간 구조	소실(Flatten)	유지 및 활용
압축 방식	뉴런 수 감소(차원 수)	공간 해상도(H, W)감소 + 채널 증가(특징 추출)
파라미터	많음(Fully-Connected)	적음(Parameter Sharing)
핵심연산	행렬 곱	합성 곱

1. **파라미터 공유 (Parameter Sharing)** : 하나의 필터( $\mathbf{W}_k$ )가 이미지의 모든 위치를 순회하며 동일한 특징(예: 수직선)을 감지한다. 이로 인해 Dense AE보다 파라미터 수가 훨씬 적어 효율적이고 과적합 위험이 낮다.

2. **번역 불변성 (Translation Invariance)** : 물체가 이미지의 왼쪽 위에 있든 오른쪽 아래에 있든 동일한 필터가 활성화 된다. 이는 공간적 정보를 보존하는 데 매우 강력하다.

## 2. AutoEncoder 알아보기 어떻게 생겼을까?

ONECLICK AI

### CNN AutoEncoder

Dense에선 Flatten 했지만, 여기선 그런거 없이 바로 들어간다.  
바로 이미지가 Encoder로 들어가는거다

Encoder는 CNN 그 자체로,

우리가 매 주 하고 있는 [Conv + ReLU -> Maxpolling ]를 반복 -> Flatten -> Dense -> z

Decoder도 방식이 여러 개가 있는데,

저번주 U-Net 할 때 했던 **ConvTranspose**와

**Upsampling + Conv**가 있다.

[https://gihak111.github.io/ai/2025/11/13/Upsampling\\_Conv\\_upload.html](https://gihak111.github.io/ai/2025/11/13/Upsampling_Conv_upload.html)



## 2. AutoEncoder 알아보기 어떻게 생겼을까?

ONECLICK AI

### CNN AutoEncoder

#### 예제: MNIST CNN AE

<https://huggingface.co/gihakkk/AutoEncoder/tree/main>

제공된 표의 ConvT 레이어는 계산이 정확히 28x28로 떨어지도록 커널/스트라이드/패딩을 조정해야 한다

레이어	입력	출력	비고
Input	[1,28,28]	-	-
Conv1	[1,28,28]	[16,26,26]	3×3, pad=0, C: 1→16
ReLU	-	-	-
MaxPool	[16,26,26]	[16,13,13]	2×2, H/W: 26→13
Conv2	[16,13,13]	[32,11,11]	3×3, C: 16→32
ReLU	-	-	-
MaxPool	[32,11,11]	[32,5,5]	2×2, H/W: 11→5
Flatten	[32×5×5]	[800]	-
Dense	800	<b>32</b>	<b>z</b>
<b>(Decoder)</b>			
Dense	32	800	-
Reshape	[800]	[32,5,5]	-
ConvT1	[32,5,5]	[16,11,11]	3×3, stride=2, pad=1 (조정됨)
ConvT2	[16,11,11]	[1,28,28]	3×3, stride=3, pad=1 (조정됨)
Sigmoid	-	[1,28,28]	$\hat{\mathbf{X}}$

## 2. AutoEncoder 알아보기 어떻게 생겼을까?

ONECLICK AI

### CNN AutoEncoder

예제: MNIST CNN AE

```
C:\Users\yeonj\OneDrive\바탕 화면\AI 커리큘럼\6. Attention과 AutoEncoder>python CNN_AutoEncoder.py
원본 데이터(x) shape: torch.Size([64, 1, 28, 28])
압축된 잠재 벡터(z) shape: torch.Size([64, 32])
복원된 데이터(x_hat) shape: torch.Size([64, 1, 28, 28])
```

```
계산된 손실(BCELoss): 0.694730281829834
역전파 완료
```

```
C:\Users\yeonj\OneDrive\바탕 화면\AI 커리큘럼\6. Attention과 AutoEncoder>python Autoencoder.py
원본 데이터(x) shape: torch.Size([64, 784])
압축된 잠재 벡터(z) shape: torch.Size([64, 32])
복원된 데이터(x_hat) shape: torch.Size([64, 784])
```

```
계산된 손실(MSELoss): 0.9956663250923157
역전파 완료
```

## 2. AutoEncoder 알아보기 어떻게 생겼을까?

ONECLICK AI

### CNN AutoEncoder

어 근데 CNN AE 하는건 좋은데 오늘 Attention 했잖아요  
이건 AutoEncoder랑 안쓰나요?

그래서 Attention AutoEncoder가 있습니다.

궁금하다면, 아래 링크에서 확인해 주세요

<https://www.proquest.com/docview/2618269233?sourcetype=Scholarly%20Journals>

또한, 다양한 오토인코더가 궁금하다면?

[https://gihak111.github.io/ai/2025/11/13/many\\_Autoencoder\\_upload.html](https://gihak111.github.io/ai/2025/11/13/many_Autoencoder_upload.html)

감사합니다