ONECLICK AI

PINN

연준모

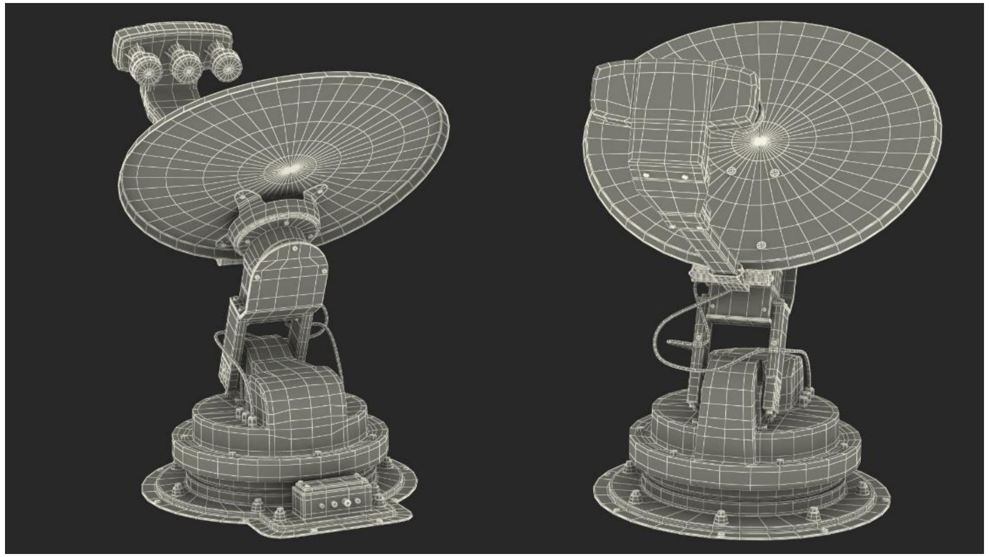
목차

1. 기존 해석법의 한계

전자기 해석의 현 주소

맥스웰 방정식을 수치적으로 풀기 위해

연속적인 물리계를 메쉬를 통해 근사한다.



$$E(r) \approx \sum a_n f_n(r)$$

$E(r)$: 구하고자 하는 전기장
$f_n(r)$: 기저 함수(Mesh)
a_n: 전기장 세기

기저함수: 연속적인 공간을 유한한 격자로 표현하였을 때 각 격자 조각 위에서 정의되는 단순한 형태의 함수

이 기저함수를 통해서 전기장의 세기를 구하는 것이 수치해석기법이다.

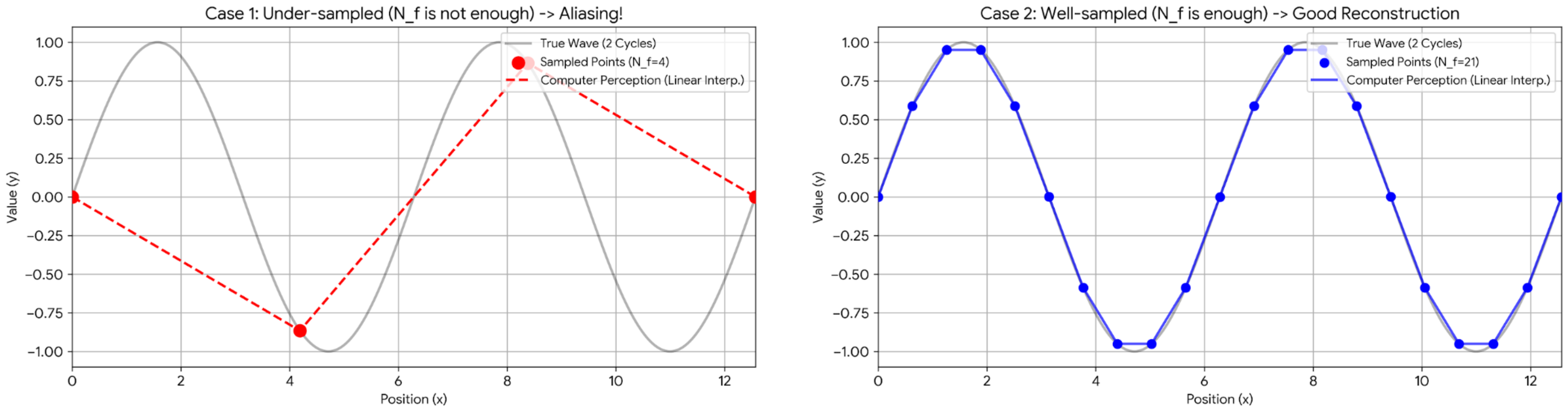
Jin, J. M. (2015). *The Finite Element Method in Electromagnetics* (3rd ed.). Wiley-IEEE Press.

Harrington, R. F. (1993). *Field Computation by Moment Methods*. Wiley-IEEE Press.

1. 기존 해석법의 한계

고주파 해석의 룰

위상오차를 최소화하기 위한 나이퀴스트 조건. $\frac{\lambda}{2} \leq \Delta x \leq \frac{\lambda}{10}$



파장 당 격자 수가 부족하면 위상 오차가 누적되어 원거리장 패턴이 완전히 틀어짐.

Chew, W. C., et al. (2001). *Fast and Efficient Algorithms in Computational Electromagnetics*. Artech House.

1. 기존 해석법의 한계

격자와 의존성

전기적 크기의 급증

파장이 짧아질수록, 동일한 물리적 크기의 안테나라도 해석해야 할 전기적 크기L/λ는 급증한다

안테나가 물리적으로 커져도 필요한 격자의 수는 증가한다

미지수 N의 폭발적 증가

3D 해석 시, 미지수 N은 주파수의 3제곱에 비례하여 증가 $N \propto f^3$.
예: 주파수가 10배 오르면, 계산량은 **1,000배** 이상 증가.

6G 시대에 들어가며 더 큰 안테나와 95GHz의 높은 주파수를 감당하기엔
기존의 mesh 방식은 부적절하다

Shephard, M. S., et al. (2005). "Mesh generation for domains with complex geometry." *Annual Review of Fluid Mechanics*, 37.

1. 기존 해석법의 한계

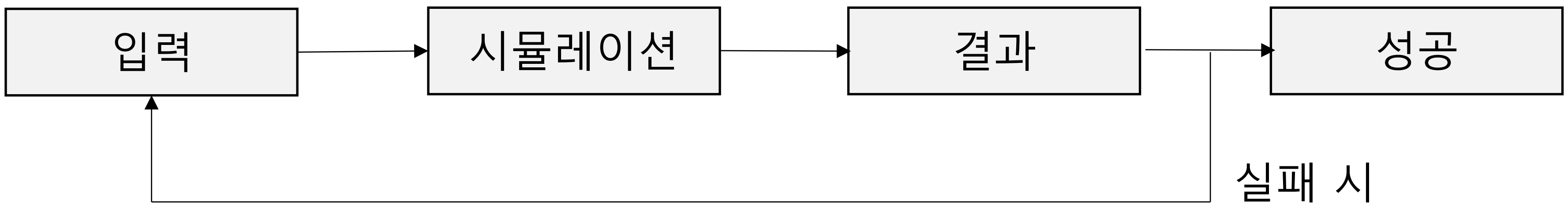
역설계의 난관

$E_{target} \rightarrow \epsilon(r)$

기존 상용 툴(Forward Solver)은 입력ϵ → 결과E 연산만 가능.

목표 성능E_{target}을 만족하는 구조ϵ를 도출하는 **직접적인 역연산 불가능.**

미분값을 알 수 없으므로, Parametric Sweep을 통해 값을 얻어 설계 시간이 기하급수적으로 증가

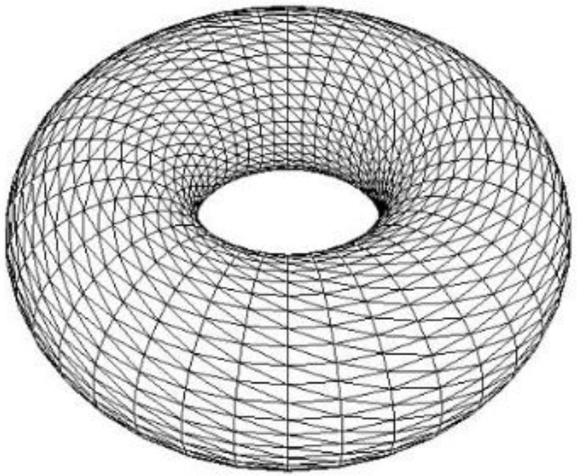


Rahmat-Samii, Y., & Michielssen, E. (1999). *Electromagnetic Optimization by Genetic Algorithms*. Wiley.

1. 기존 해석법의 한계

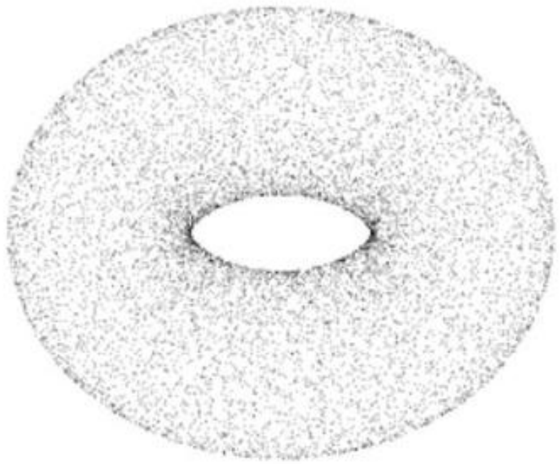
Paradigm Shift

Mesh 방식



격자점들이 서로 이웃과 연결되어
그 사이를 선형적으로 채우는 방식.
서로 구속된 구조이며, 모양을 조금
만 바꾸려 해도 전체를 다시 짜야 한
다.

Point 방식



점들은 서로 연결될 필요 없이
그냥 좌표로 존재한다

격자라는 족쇄가 사라졌기 때문에,
우리는 형상을 자유자재로 바꿀 수 있고,
무엇보다 미분을 통해 최적의 해를 찾아갈 수 있다.

PINN

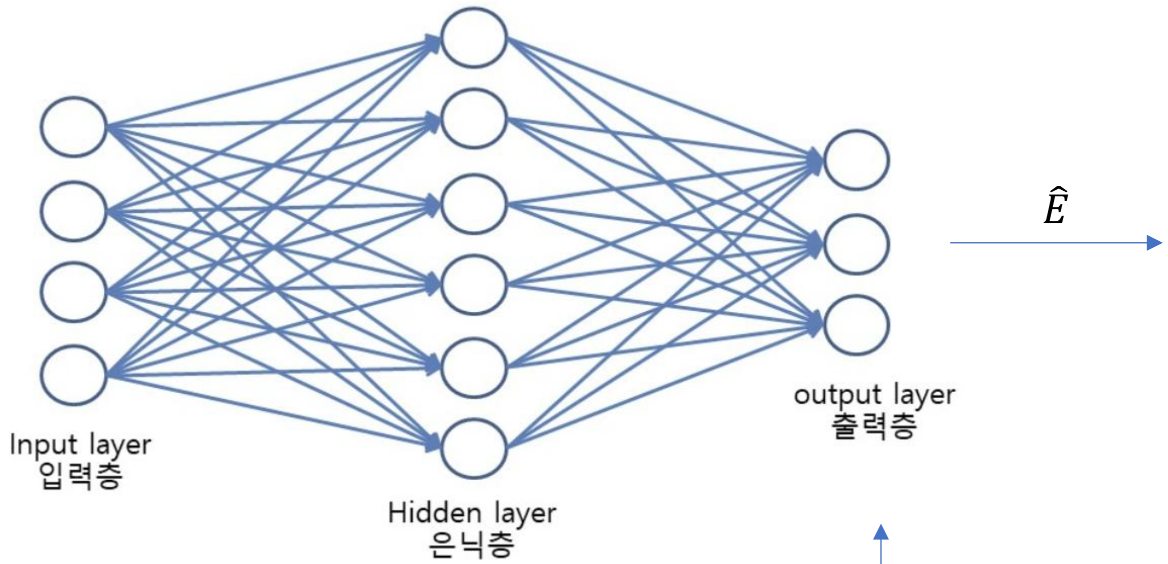
입력 데이터가 Point인 Mesh-free 방식

앞선 포인트 좌표들을 딥러닝 모델의 입력으로 한다

(x, y, z)

이 좌표 Point 의 수를 N_f라 한다.

Number of Collocation Points for the Residual function



오차를 줄이기 위한 Loss function에 만족하고자 하는 방정식을 넣는다.

학습이 진행되며 오차가 감소하면, 해당 좌표에서 방정식을 만족시키는 모델이 만들어 진다.

2. PINN

역문제 해결

기존 해석: 급전점 위치 (x_s, y_s)는 고정된 상수

PINN: 급전점 위치 (x_s, y_s)를 학습 가능한 변수로 설정.

학습은 오차를 줄이기 위한 방향으로 진행되기 때문에

급전점 같은 위치를 학습 가능한 변수해로 정의 해 주면 모델의 Loss가 줄어듦에 따라 올바른 급전점의 위치도 알 수 있다.

2. PINN

네트워크 아키텍처 상세

아키텍쳐는 내가 고를 수 있다.

MLP PINN : 제일 덜 블랙박스

CNN PINN : 이미지 형태의 공간 도메인

GNN PINN : 안테나나 로봇 관절 등 비정형 메쉬

RNN/LSTM PINN : 시간에 따라 변하는 동역학 시스템

Transformer PINN : 높은 성능과 범용성

현 시점에서는 무조건 Transformer PINN을 사용한다

2. PINN

지배 방정식 (Governing Equation)

우리는 보고싶은 물리정보가 어떤 건지에 따라 지배방정식을 정해야 한다

방사 노이즈(Radiated Emission)를 볼 때: **헬름홀츠 방정식**
과도 응답(Transient)이나 ESD를 볼 때: **시간 영역 파동 방정식**
케이블/전도 노이즈(Conducted Emission)를 볼 때: **텔레그래퍼 방정식**

이번에는, 헬름홀츠 방정식을 기준으로 진행 해 보자

2. PINN

ONECLICK AI

지배 방정식 (Governing Equation)

$$\nabla^2 u(\mathbf{x}) + k^2 u(\mathbf{x}) = f(\mathbf{x})$$

$\boldsymbol{u}(\mathbf{x})$ **(u):** 전기장 E_z의 진폭
$\boldsymbol{\nabla}^2$ **(라플라시안, ∇ 제곱):** 얼마나 꼬불꼬불한가를 뜻한다. 파동이 급격하게 변하는지, 완만하게 변하는지를 나타낸다.
k (파수): 얼마나 자주 진동하는가 이다. 높으면 고주파다.
$\boldsymbol{f}(\mathbf{x})$ **(Source):** 파동을 만들어내는 전류원. 전자기파를 발생기키는 급전 전류

파동의 꼬불거림($\nabla^2 u$)과 진동수($k^2 u$)를 더하면, 외부에서 가해진 힘(f)과 같아야 한다.

2. PINN

왜 헬름홀츠 방정식일까?

Time Domain: 스위치를 켜는 순간부터 전기가 퍼져나가는 모든 과정을 다 계산해야 한다. (t 변수 필요). 너무 무거워 진다.

Frequency Domain: 안테나 설계의 목적은 대부분 특정 주파수(공진 주파수)에서 신호가 안정적으로 나갈 때(Steady State)의 패턴을 보는 것이다.

Helmholtz: 파동 방정식에서 시간(t)을 상수($e^{j\omega t}$)로 가정하고 없애버린 식.
즉, 정지 상태의 파동 패턴을 계산하는 가장 효율적인 방정식이다.

헬름홀츠 방정식은 파동 방정식에서 시간 변수를 소거한 형태로, 불필요한 연산을 줄이고
우리가 원하는 **공간상의 방사 패턴**을 가장 효율적으로 학습할 수 있다.

2. PINN

Loss function

$\nabla^2 u(\mathbf{x}) + k^2 u(\mathbf{x}) = f(\mathbf{x})$

$f(\mathbf{x})$를 옆으로 넘기면, $\nabla^2 u(\mathbf{x}) + k^2 u(\mathbf{x}) - f(\mathbf{x}) = 0$이 된다.

이렇게 이항시킨 식을 Loss function에 넣으면, 정답지가 0이 되어
데이터로부터 자유로운 학습이 가능해 진다.

입력 데이터인 좌표에서의 전기장이 만족되게 된다.

2. PINN

Loss function

$\nabla^2 u + k^2 u = f$

∇^2 ⟶ 공간에 대한 2차 편미분의 합.

$$\nabla^2 u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2}$$

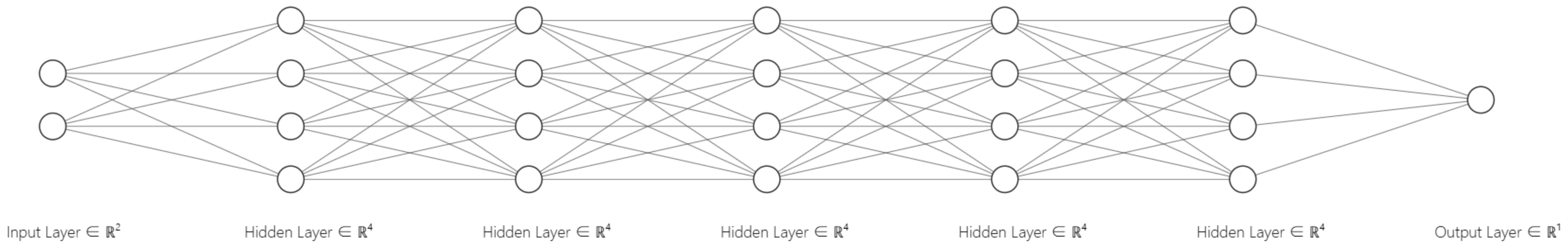
이걸 자세히 이해해 보자.

2. PINN

Loss function

AI는 수많은 층으로 구성된 거대한 합성함수이다.

$$u(x) = f_L(W_L \cdot f_{L-1}(\dots f_1(W_1 x + b_1)\dots) + b_L)$$



$$\nabla^2 u + k^2 u = f$$

AI의 출력을 Loss function의 지배방정식과 최대한 동일하게 만들어야 하는데,
헬름홀츠 방정식에는 2차 미분된 형태가 포함되어 있으므로 우리도 2차 미분을 해야 한다.

저 Loss function 을 통과한 값이 0이 아니면 틀린 것으로 그만큼의 Loss 가 발생한다고 생각하면 된다.

2. PINN

Loss function

$\mathcal{L} = \mathcal{L}_{\mathcal{PDE}} + \mathcal{L}_{\mathcal{BC}}$

전체 에러($\mathcal{L}$)는 물리 법칙을 어긴 에러($\mathcal{L}_{\mathcal{PDE}}$)와 경계 조건을 어긴 에러($\mathcal{L}_{\mathcal{BC}}$)의 합 이다.
그 외 더 넣고 싶으면 넣어도 되긴 한다.

$$\mathcal{L}_{\mathcal{PDE}} = \frac{1}{N_f}\sum_{i=1}^{N_f}\left|\nabla^2\hat{u} + k^2\hat{u} - f\right|^2$$

$\hat{\boldsymbol{u}}$**:** AI가 추측한 전기장.

$\boldsymbol{\nabla^2\hat{u} + k^2\hat{u} - f}$ **:** 앞서 본 지배 방정식. 이항하면 0 되어야 한다

이걸 모든 입력값에 대해 진행하고, 전체 N_f에 대한 평균을 구한다.

2. PINN

Loss function

$$\mathcal{L}_{\mathcal{BC}} = \frac{1}{N_b}\sum_{i=1}^{N_b}\left|\hat{u}\left(\mathbf{x}_{\mathbf{edge}}\right) - g\left(\mathbf{x}_{\mathbf{edge}}\right)\right|^2$$

$\hat{u}\left(\mathbf{x}_{\mathbf{edge}}\right)$: AI가 출력한 가장자리 좌표의 전기장 값
$g\left(\mathbf{x}_{\mathbf{edge}}\right)$: 물리적으로 정해진 가장자리 좌표의 전기장 값
(PEC : 0, Port : 1V/m 등) 하나하나 지정해 주어야 한다
N_b: 입력 좌표 중 경계면에 해당되는 좌표의 수

2. PINN

Loss function

PINN은 경계조건을 잘 못지킨다.

N_f를 찍는데, 경계조건에 찍혀있는 N_f보다 다른 면에 찍혀있는 N_f가 더 많다.

내부의 $\mathcal{L}_{\mathcal{PDE}}$는 2차 미분 등이 포함되어 있어 기울기가 아주 복잡하고 불안정하다.

반면 경계 $\mathcal{L}_{\mathcal{BC}}$는 상대적으로 단순하다.

이 둘이 겹치면,보통 복잡한 PDE 쪽으로 학습이 편향되거나, 반대로 PDE가 너무 어려워서 학습을 포기하고 0으로 수렴해버리는 문제가 발생한다

Wang, S., Teng, Y., & Perdikaris, P. (2021). Understanding and mitigating gradient pathologies in physics-informed neural networks. SIAM Journal on Scientific Computing.

2. PINN

Loss function

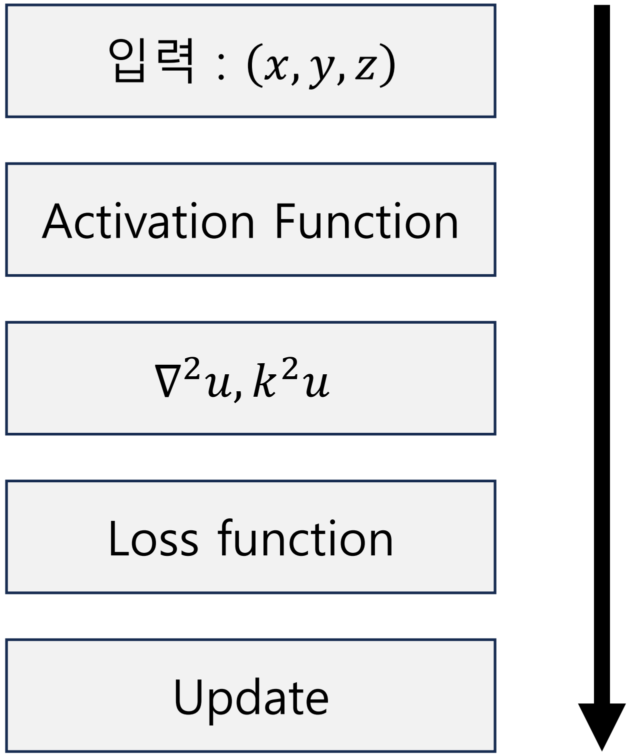
이를 해결하기 위해

Hard Weighting: 경계면의 점 개수N_b를 늘리거나, 경계 Loss에 큰 상수 가중치$\lambda_{BC} > 1$를 곱한다.

Soft Weighting (Advanced): 학습 중에 기울기를 분석하여 가중치를 자동으로 조절하는 알고리즘 (예: GradNorm, ReLoBRaLo) 적용.

2. PINN

Algorithm Flowchart



위와 같은 과정을 통해서 PINN은 지배 방정식을 만족하는
방향으로 강제되며 학습된다.

2. PINN

Algorithm Flowchart

지금까지는 아키텍쳐 내용 없이, PINN의 고유적인 특징만 알아보았다.

아키텍쳐에 따라 추가적으로 필요한 것이 다 다른데,

Transformer 아키텍쳐에 대해 알아본 후 이어서 보도록 하자

3. Transformer

Transformer

순서의 족쇄를 끊고, **문맥의 시대**를 열다

Attention is all you need

AI의 혁신

https://arxiv.org/abs/1706.03762

3. Transformer

Transformer

이전까지는 텍스트 처리 모델 중 RNN, LSTM이 최고였다.

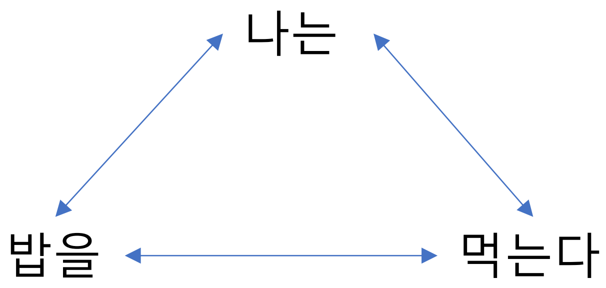
순차적 계산:
"나는 밥을 먹는다"라는 문장을 "나는" → "밥을" → "먹는다" 순서로, 단어를 **하나씩 순차적으로** 처리하였다.
t 시점의 계산은 t-1 시점의 계산이 끝나야만 가능했다.
이는 GPU의 강력한 병렬 처리 능력을 전혀 활용할 수 없어 학습 속도가 매우 느렸다.

치매 문제:
문장이 길어지면, "나는"이라는 첫 단어의 정보가 "먹는다"라는 끝 단어까지 전달될 때 쯤이면
정보가 희미해지거나 소실될 위험이 컸다.

3. Transformer

Transformer

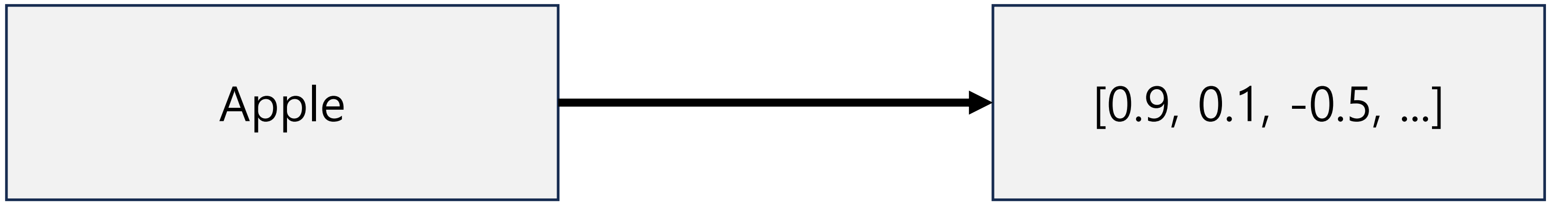
만약, 문장을 순차적으로 보지 않고, **모든 단어를 한꺼번에 펼쳐놓고**
각 단어가 다른 모든 단어와 **직접 상호작용**하게 만들면 어떨까?



순환(Recurrence) 구조를 아예 없애고, 오직 Attention 메커니즘만으로
더 빠르고 더 성능 좋은 모델을 만들 수 있다.

Transformer

입력될 데이터는 텍스트. 하지만, 컴퓨터는 고차원 백터 공간에서의 연산을 수행한다.
따라서 텍스트를 숫자로 만드는데, 이걸 토크나이징이라고 한다.



어텐션 메커니즘은 “A가 B를 때렸다 " 와 “B가 A를 때렸다 " 를 **똑같은 문장**으로 인식한다.
순서 정보가 사라졌기 때문이다.

https://gihak111.github.io/ai/2025/08/29/AI_basics_1_upload.html

Transformer

이 문제를 해결하기 위해, **위치 인코딩 (Positional Encoding)**을 진행한다

단어의 '의미' 벡터에, 단어의 '위치' 정보가 담긴 벡터를 더해준다.

$$\mathrm{InputVector} = \mathrm{TokenEmbedding} + \mathrm{PositionalEncoding}$$

논문에선 사인(sin)과 코사인(cos) 함수를 사용한다.

$$PE_{(pos,2i)} = \sin\left(pos/10000^{2i/d}\mathrm{model}\right)$$

$$PE_{(pos,2i+1)} = \cos\left(pos/10000^{2i/d}\mathrm{model}\right)$$

Pos : 0부터 시작하는 토큰의 위치(인덱스).

I : 512차원 임베딩 벡터 내의 차원 인덱스.

Transformer

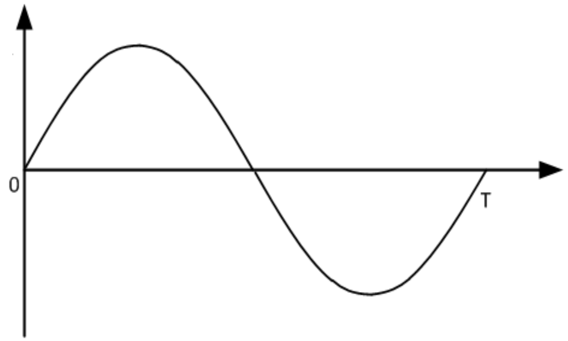
만일, 곱해지는 숫자가 이런식으로 크다면,

A가 * (1) + B를 * (2) + 때렸다 * (3)

뒤쪽의 값들와 앞쪽의 값 차이가 너무 커 불균형이 온다.

→ **값을 작게 유지하면서도$-1 \sim 1$, 위치를 구분할 방법이 필요하다.**

이를 위해 Sin 파를 이용한다.

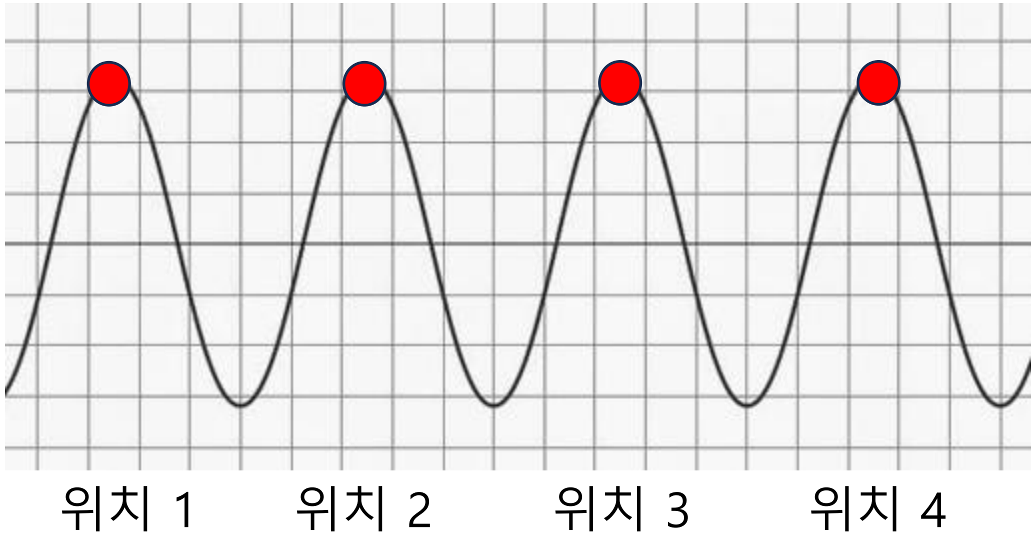


이러면 -1 ~ 1 사이로 위치를 알려줄 수 있다.

Transformer

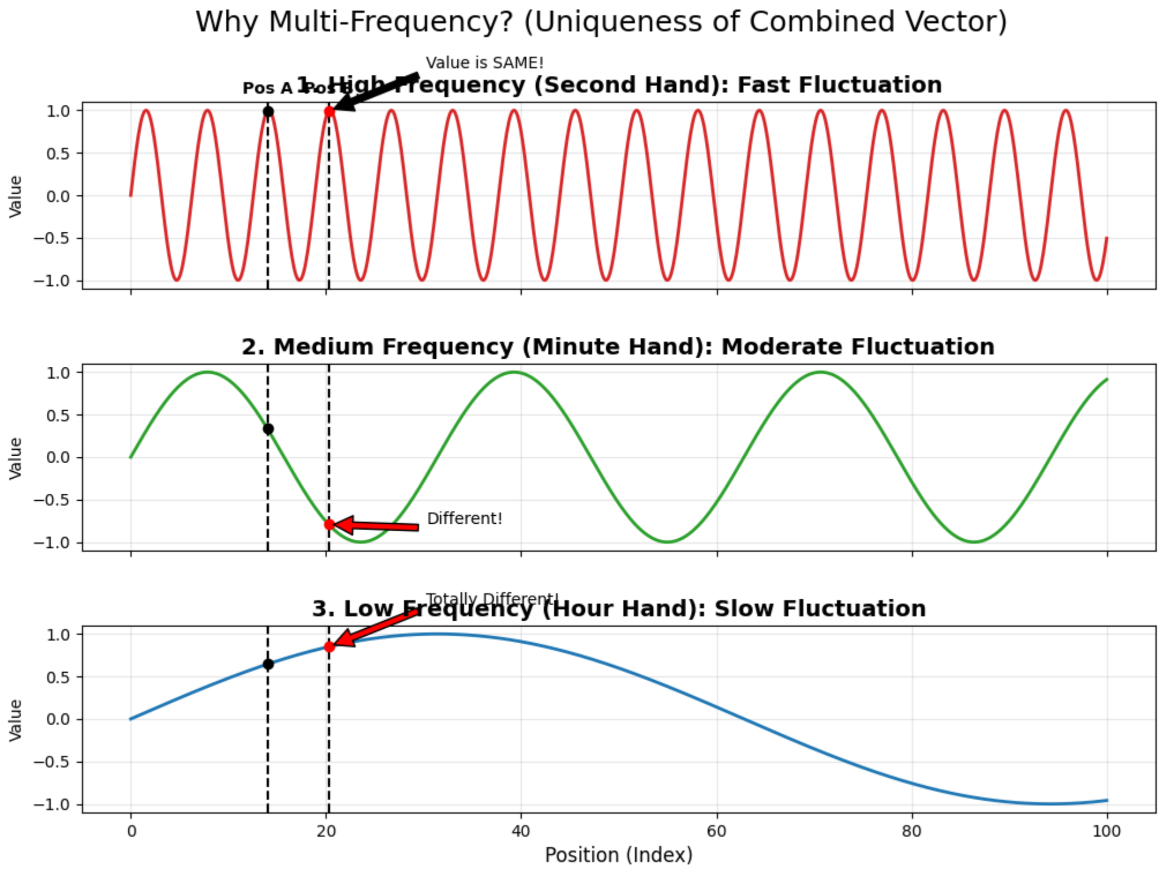
그렇다고 Sin 함수 하나만 쓰면, 값이 오르락내리락 반복되니까
여기가 1번 위치인지, 한 바퀴 돌아서 온 100번 위치인지 구분을 못 하게 된다.

때문에, 서로 주파수가 다른 함수 여러 개를 사용하게 된다.



→ 출력은 모두 같은 1

Transformer



그림에 있는 두 점은 모두 붉은 그래프에서는 1.0 이지만,
2, 3 일 때는 값이 다르다.

Pos 14 : [1.0, 0.3, 0.6]
Pos 20 : [1.0, -0.8, 0.8]

이렇게 하면 여러 입력이 아무리 길어도
다 다른 위치로 임베딩 할 수 있다.

+ 논문에서는 sin, cos 이렇게 두 개 사용하였다.

Transformer

따라서, 최종 출력은,

Apple 이 [0.9, 0.1, -0.5] 라 하고, Pos 14 : [1.0, 0.3, 0.6]라 했을 때,

$$Input = Embedding + PositionalEncoding$$

[0.9, 0.1, -0.5] + [1.0, 0.3, 0.6] = [1.9, 0.4, 0.1]

이렇게 된다. 이런 값들이 트랜스포머의 인코더 블록으로 들어가게 된다.

Self Attention

인코더의 목표는 **문맥을 깊게 이해**하는 것이다

"I went to the **bank**..."라는 문장에서 "bank"가 '은행'인지 '강둑'인지 알아내야 한다

이를 위한 **Self-Attention**

3. Transformer 4. Self Attention

Attention

Query (Q) : 내가 지금 당장 궁금한 것. (예: "머신러닝의 역사")
Key (K) : 도서관에 있는 모든 책의 '키워드' 또는 '색인'. (예: "머신러닝", "AI", "역사"...)
Value (V) : 책의 '실제 내용'.

어텐션은
(1) 내 Query로 모든 Key와 유사도를 비교하고,
(2) 유사도(가중치)가 높은 Key의 Value(실제 내용)를 가져와
(3) 가중합하여 나에게 꼭 맞는 '맞춤형 정보'를 만드는 과정이다

Self Attention

Self-Attention에서 Q, K, V는 모두 **자기 자신(입력 문장)**에서 나온다.

Query (Q) - 질문: bank인 나랑 친한 단어, 문맥상 어울리는 단어가 뭐야?

Key (K) - 확인: bank는 문장 내 다른 단어들의 꼬리표(Key)를 하나씩 확인한다.
I: "나(주어)" → (관련성 낮음)
went: "가다(동사)" → (강둑에도 갈 수 있고 은행에도 갈 수 있음. 관련성 중간)
money: "돈(목적어)" → (강둑에 돈을 입금하러 가진 않음. 관련성 매우 높음)

Value (V) – 정보와 결합: bank는 관련성 점수가 가장 높은 money의 정보를 강력하게 흡수한다.

→ bank는 money가 가진 "금융, 경제"라는 속성(Value)을 자신의 의미에 섞는다.

Self Attention

Q, K, V는 어디서 오는가?

원래의 입력 벡터(x)에서 학습 가능한 가중치 행렬W_Q, W_K, W_V를 곱해서 생성 된다.

$q_i = x_i \cdot W_Q$(내 i번째 단어 x_i로 질문하는 법을 배운다)
$k_i = x_i \cdot W_K$(내 i번째 단어 x_i로 확인하는 법을 배운다)
$v_i = x_i \cdot W_V$(내 i번째 단어 x_i로 본질을 만드는 법을 배운다)

이 W 행렬들이야말로 트랜스포머가 **훈련 과정에서 학습하는 핵심이**다.
모델은 어떤 질문(Q)을 던져야 문맥 파악에 유리한지, 어떤 간판(K)을 내걸어야 하는지를 배운다.

X를 **Attention 연산에 필요한 서로 다른 역할의 벡터로 변환**해 주는 가중치

Self Attention

식은 다음과 같다.

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

1. **QK^T (유사도 계산):**

   Query 행렬과 Key 행렬을 내적한다.
   `bank`의 Query(q_{bank})가 `money`의 Key(k_{money})와 내적된다.
   이는 문장 내 모든 단어(Q)가 다른 모든 단어(K)와 얼마나 유사한지를 나타내는 **유사도 점수 행렬**이다.

Self Attention

2. $\frac{QK^T}{\sqrt{d_k}}$ **스케일링:**

d_k는 Key 벡터의 차원(논문에서는 64)이다.

d_k가 커질수록, QK^T의 내적값(유사도 점수)이 너무 커지는 경향이 있다.
이 값이 너무 크면, 다음 단계인 Softmax 함수에서 그래디언트가 0에 가까워져 소멸하게 된다

그래디언트가 0이 되면 모델이 학습을 멈추게 된다.
따라서 d_k의 제곱근($\sqrt{64} = 8$)으로 나눠줌으로써 값을 안정화시켜 **학습이 원활하게** 되도록 한다.

3. Transformer 4. Self Attention

Self Attention

3. $\text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)$ **가중치 정규화:**

안정화된 유사도 점수에 Softmax 함수를 적용한다.
각 행의 합이 1이 되는 어텐션 가중치가 출력된다.
(예: "bank" → I: 0.05, ... money: 0.7, ...)

Self Attention

4. $\text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$ **가중합:**

앞서 구한 확률에 실제 정보**Value**들을 섞는다.

$$z_{\text{bank}} = (0.05 \times v_I) + \cdots + \left(0.7 \times v_{\text{money}}\right) + \cdots$$

단순히 'bank'라는 단어가 아니라, '돈(money)'의 의미가 70% 섞인 **문맥이 반영된 새로운 벡터**가 탄생한다.

Self Attention

입력 시퀀스의 각 요소가 **같은 시퀀스 내의 다른 모든 요소**와 얼마나 관련되어 있는지 계산

1. 하나의 입력 시퀀스(예: 문장)가 들어온다.
2. 시퀀스의 각 단어(위치)는 **Q**, **K**, **V** 세 벡터를 **모두 자기 자신**으로부터 생성한다.
3. 특정 단어의 **Q**를 시퀀스 내 다른 모든 단어의 **K**와 비교하여 관계 점수를 얻는다.
4. 이 점수를 이용해 모든 **V**를 가중합하여, 문맥 정보가 풍부하게 담긴 **새로운 벡터**를 만들어 낸다.

문장 내의 단어 "**그것**"이 문장 내의 다른 단어(예: **"강아지"**) 중 누구를 가리키는지 스스로 파악하여 의미를 명확히 하는 과정과 같다.

하나의 시퀀스 내부의 단어들 간의 관계를 파악하여 문맥을 이해한다.

Attention VS Self Attention

	Attention	Self-Attention
목적	두 시퀀스 간의 관계 파악	하나의 시퀀스 내부의 관계 파악
사용 위치	주로 디코더 부분	주로 인코더, 디코더 부분
Q, K, V 출처	Q: 타겟 시퀀스(디코더 출력)	Q, K, V: 동일한 시퀀스(자신)
	K, V: 소스 시퀀스(인코더 출력)	
대표 모델	RNN 기반 Seq2Seq	Transformer, GPT 등
역할	소스 문장에서 타겟 단어 생성에 가장 관련 높은 부분을 찾아 집중	문장 내에서 단어가 다른 단어들과 얼마나 관련 있는지 파악하여 풍부한 문맥 백터 생성

Multi Head Attention

한 번의 Self-Attention은 한 가지 관점(예: 'bank'와 'river'의 관계)만 볼 수 있다

한 명의 전문가가 문장을 보는 것보다, 8명의 서로 다른 전문가(Head)가 각자 다른 관점으로 동시에 보면 어떨까?

헤드 1: "bank"와 "river"의 **의미적 관계**에 주목할 수 있다.
헤드 2: "I"와 "went"의 **문법적 관계**(주어-동사)에 주목할 수 있다.
헤드 3: "to"와 "the"의 **위치 관계**에 주목할 수 있다.

Multi Head Attention

$$\mathrm{MultiHead}(Q,K,V) = \mathrm{Concat}(\mathrm{head}_1, \ldots, \mathrm{head}_h)W^O$$

$$\text{where } \mathrm{head}_i = \mathrm{Attention}\left(QW_i^Q, KW_i^K, VW_i^V\right)$$

1. **프로젝션 (Projection):**
   h개(예: 8개)의 헤드 각각에 대해 서로 다른 가중치 행렬(W_i^Q, W_i^K, W_i^V)을 준비한다. 원본 Q, K, V를 이 행렬들에 곱해 각 헤드를 위한 저차원(예: 64차원)의 Q_i, K_i, V_i를 만든다.
2. **병렬 어텐션:**
   8개의 헤드가 각각 독립적으로, 동시에(병렬로) Scaled Dot-Product Attention을 수행한다. (head_1 ~ head_8 계산)
3. **연결 (Concatenation):**
   8개의 헤드에서 나온 64차원 출력 벡터들을 모두 **연결**하여 다시 512차원(8×64) 행렬을 만든다.
4. **최종 프로젝션:**
   이 연결된 행렬에 또 다른 가중치 행렬 W^O를 곱하여 최종 MHA 출력을 얻는다.

Multi Head Attention 더 자세히 알아보자

1단계: 선형 변환 및 병렬 어텐션 수행

1. **입력 분할 (Projection):** 입력으로 들어온 **쿼리 (Query), 키 (Key), 값 (Value)** 벡터를 여러 개의 **헤드 (Head)** 수만큼 분할한다

   - 각 헤드 (i)는 독립적인 **선형 변환** 가중치 행렬 (W_i^Q, W_i^K, W_i^V)을 사용하여 Q, K, V를 더 작은 차원으로 투영한다

   - 예를 들어, 모델 차원 d_{model}을 사용하고 헤드 수 H를 사용한다면, 각 헤드의 차원은 $d_k = d_{\text{model}}/H$가 된다

$$Q_i = QW_i^Q, K_i = KW_i^K, V_i = VW_i^V$$

2. **독립적인 어텐션 계산:** 각 헤드 (i)는 투영된 Q_i, K_i, V_i를 사용하여 일반적인 **스케일드 닷 프로덕트 어텐션 (Scaled Dot-Product Attention)**을 독립적으로 수행한다.

$$Head_i = \text{Attention}(Q_i, K_i, V_i) = \text{softmax}\left(\frac{Q_i K_i^T}{\sqrt{d_k}}\right) V_i$$

Multi Head Attention 더 자세히 알아보자

2단계: 결과 결합 (Concatenation)

- 각 헤드에서 독립적으로 계산된 H개의 출력 값 $(Head_1, Head_2, \dots, Head_H)$**을 원래의 $\boldsymbol{d}_{\mathbf{model}}$ 차원으로 복원하기 위해 하나로 이어 붙인다**

$$\mathrm{Concat}(Head_1, \dots, Head_H)$$

3단계: 최종 선형 변환

- 이어 붙인 최종 결과에 또 하나의 **선형 변환** 가중치 행렬 (W^O)을 곱하여 최종 출력을 만들어 낸다.

$$\mathrm{MultiHead}(Q, K, V) = \mathrm{Concat}(Head_1, \dots, Head_H)W^O$$

Multi Head Attention

하나의 어텐션 : 단 하나의 관점으로만 문맥을 파악한다.

멀티 헤드 어텐션 : 여러 개의 독립적인 어텐션(헤드)을 통해 다양한 관점과 관계를 동시에 파악하고, 그 결과를 결합하여 **더 풍부하고 강력한 문맥 표현 벡터**를 생성

하나의 벡터 안에 문법, 의미, 위치 등 **다양한 관점의 문맥 정보가 풍부하게** 압축된다.

Feed Forward Network

Multi-Head Attention을 통과하며 얻은 문맥 정보를 한 번 더 **비선형적으로 변환하고 정제**하여 표현력을 높이는 역할이다.
각 토큰의 위치마다 **독립적으로** 적용되는 일반적인 신경망(MLP)이다.

그냥 Dense Layer 아닌가요?

Feed Forward Network

FFN은 두 개의 Dense Layer 로 구성되어 있다

1단계: 확장 (Expand)

입력 벡터 $\mathbf{x}$는 첫 번째 Dense Layer(W_1)를 통과

이때 입력 차원 d_{model}은 보통 4배 더 큰 차원 d_{ff}로 확장(Expand) (예: $512 \rightarrow 2048$)

여기에 **ReLU**와 같은 비선형 활성화 함수가 적용된다.

$$\mathbf{z} = \mathrm{ReLU}(\mathbf{x}W_1 + \mathbf{b_1})$$

3. Transformer 6. FFN

Feed Forward Network

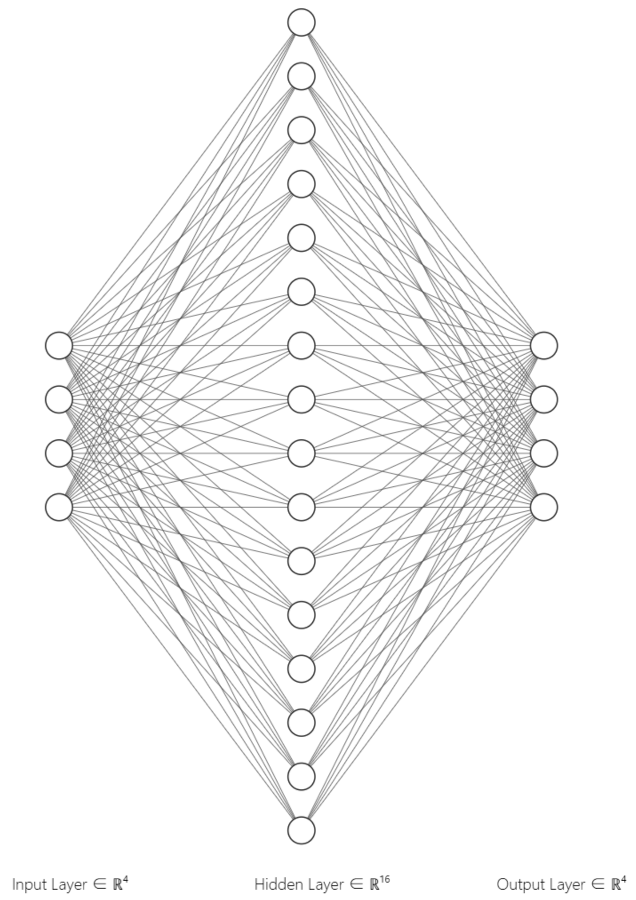
FFN은 두 개의 Dense Layer 로 구성되어 있다

2단계: 축소 (Contract)

확장된 벡터 $\mathbf{z}$는 두 번째 Dense Layer(W_2)를 통과

이 층은 차원을 원래의 d_{model}로 **축소(Contract)**시켜 출력

$$\mathrm{FFN}(\mathbf{x}) = \mathbf{z}W_2 + \mathbf{b_2}$$



하나의 층(Layer)이 아니라 두 개의 Dense Layer로 이루어진 작은 네트워크(Sub-network)

Add & Norm

인코더와 디코더의 모든 하위 층(MHA, FFN) 주위에는 `Add & Norm`이라는 두 가지 장치가 필수로 들어간다

Input → Sublayer(MHA or FFN) → Add&Norm → Output

Add (잔차 연결, Residual Connection):

$$\mathrm{Output} = x + \mathrm{Sublayer}(x)$$

하위 층의 입력 x를 하위 층의 출력 $\mathrm{Sublayer}(x)$에 **그대로 더해준다.**

x라는 원천 정보가 그대로 다음 층으로 전달되는 고속도로 역할을 한다. 층이 6개, 12개, 100개로 깊어지더라도 정보가 소실되는 것을 막아 **매우 깊은 모델의 학습을 가능하게** 하는 핵심 장치이다.

Add & Norm

Norm (계층 정규화, Layer Normalization):

$$\text{Output} = \text{LayerNorm}\big(x + \text{Sublayer}(x)\big)$$

이유:
'Add'로 더해진 값들의 스케일이 널뛰는 것을 방지하고 학습을 안정화시킨다.
+ 배치 정규화(Batch Norm)는 문장 길이가 가변적인 NLP에서 불안정할 수 있어, 배치 크기와 무관하게 각 토큰 벡터의 특성을 기준으로 정규화하는 계층 정규화(Layer Norm)를 사용한다

Encoder Layer

인코더는 6개의 동일한 레이어로 구성되며, 각 레이어는 2개의 하위 층을 가진다

1. **Multi-Head Self-Attention** (Q, K, V가 모두 인코더의 이전 층 출력)
2. **Add & Norm**
3. **Position-wise Feed-Forward Network (FFN)**
4. **Add & Norm**

1층의 출력이 2층의 입력으로 들어간다.
6층을 거친 최종 출력은 "**입력 문장의 모든 문맥을 완벽하게 이해한**" 벡터들의 리스트(메모리)가 된다

4. Transformer PINN

Transformer vs PINN

Transformer	PINN
Input Data token	Input Data coordinate

단순하게 좌표값만 Transformer에 넣는다 해 버리면 고주파 성분에 대해 전혀 학습하지 못하게 된다.

Transformer-PINN에서는 입력단에서부터 다른 접근이 필요하다.

4. Transformer PINN 1. Embedding

Fourier Feature Mapping

좌표를 고차원 특징 공간으로 매핑하여 Transformer가 이해할 수 있는 형태(토큰화)로 만든다

$$gamma(x) = [\cos(2\pi \mathbf{B}x)\,, \sin(2\pi \mathbf{B}x)]$$

입력 좌표에 위치 인코딩Positional Encoding을 적용할 때, 다양한 주파수의 $\sin, \cos$ 함수를 통과시킨다.

이때, 강제로 고주파 진동 성분을 입력으로 주입하면 네트워크가 고주파 패턴을 특징으로 인식하여 학습 속도가 비약적으로 빨라진다

그런데, 학습 입력 데이터에 시간이 포함된다??

4. Transformer PINN 1. Embedding

Fourier Feature Mapping

왜 시간 t 가 입력에 포함 될 수 있을까?

1. 전통적 방식 like FEKO

$t = 0$일 때의 값으로 $t = \Delta t$를 계산하고, 그걸로 다시 $t = 2\Delta t$를 계산하는 **순차적 반복(Loop)** 방식이다.
그래서 입력은 공간 (x, z)만 들어가고, 시간은 반복문(for loop)으로 처리한다.

2. PINN 방식 (Spatio-Temporal Domain)

PINN은 시공간(Space-Time) 전체를 한 번에 학습하는 방식.
네트워크는 $u = f(x, z, t)$라는 함수 자체를 근사한다.
즉, 네트워크에게 좌표 (x, z)에서 시간 t일 때의 물리량(전자기장 등)이 얼마야?
라고 물어봐야 하기 때문에, **t도 x, z와 동등한 하나의 좌표축으로 취급**하여 입력에 넣는다.

시퀀스가 아닌 어텐션만을 사용하기 때문에 시공간을 입력으로 하여 편미
분 방정식의 해를 최적화 시킬 수 도 있다.

4. Transformer PINN 1. Embedding

Fourier Feature Mapping

1. 4차원 데이터 정의

가상의 파라볼릭 안테나 해석 공간에서, 임의의 한 점을 선택해보자.
안테나 표면 근처($x = 0.5, y = -0.2, z = 1.0$)에서의 전자기장을 알고 싶은 상황.

입력 벡터 $\mathbf{x}$ (Shape: 3×1)

$$\mathbf{x} = [0.5, -0.2, 1.0]^T$$

Fourier Feature Mapping

2. 가우시안 랜덤 행렬 B 생성

입력 좌표 (x, y, z)를 무작위로 섞어서 다양한 주파수 성분을 만들어낸다.

학습되지 않는 고정된 상수로 특징차원이다.

보통 여기서 3 * 64 차원으로 올린다 하지만, 여기서는 2차원으로 설명하겠다

$$MatrixB = \begin{bmatrix} [1.5 & -0.5], \\ [0.2 & 1.0], \\ [-1.0 & 0.5] \end{bmatrix}$$

1행: x 좌표에 곱해질 가중치
2행: y 좌표에 곱해질 가중치
3행: z 좌표에 곱해질 가중치

위치 임베딩과 같은 과정이다

4. Transformer PINN 1. Embedding

Fourier Feature Mapping

3. 연산

입력 벡터와 행렬 $\mathbf{B}$를 곱한다.　　$\mathbf{x}^{\mathbf{T}}\mathbf{B}$ 또는 $\mathbf{B}^{\mathbf{T}}\mathbf{x}$ 형태로 연산

$$\mathbf{v} = 2\pi \cdot \left(\mathbf{x}^{\mathbf{T}} \cdot \mathbf{B}\right)$$

$$\mathbf{x} = [0.5, -0.2, 1.0]^{T}$$

$$MatrixB = \begin{bmatrix} [1.5 & -0.5], \\ [0.2 & 1.0], \\ [-1.0 & 0.5] \end{bmatrix}$$

내적 계산

첫 번째 특징값: $(0.5 \times 1.5) + (-0.2 \times 0.2) + (1.0 \times -1.0) = 0.75 - 0.04 - 1.0 = -0.29$

두 번째 특징값: $(0.5 \times -0.5) + (-0.2 \times 1.0) + (1.0 \times 0.5) = -0.25 - 0.2 + 0.5 = 0.05$

2π 스케일링

$\mathbf{v_1} = 2\pi \times (-0.29) \approx -1.82$　　$\mathbf{v_2} = 2\pi \times 0.05 \approx 0.31$　　$\longrightarrow$　　$\mathbf{v} = \begin{bmatrix} -1.82, & 0.31 \end{bmatrix}$

4. Transformer PINN 1. Embedding

ONECLICK AI

Fourier Feature Mapping

푸리에 특징 매핑

구해진 $\mathbf{v}$값에 $\sin$ 과 $\cos$ 을 각각 적용하여 벡터를 2배로 늘린다

$$\gamma(\mathbf{x}) = [\sin(\mathbf{v}), \cos(\mathbf{v})]$$
$$\sin(-1.82) \approx -0.97$$
$$\cos(-1.82) \approx -0.25$$
$$\sin(0.31) \approx 0.30$$
$$\cos(0.31) \approx 0.95$$

$$\gamma(\mathbf{x}) = \begin{bmatrix} -0.97, & -0.25, & 0.30, & 0.95 \end{bmatrix}$$

좌표값 입력이 비선형 변환을 거쳐 **주파수 성분을 포함한 고차원 벡터**로 변환되었다.
이제 신경망은 위치를 값이 아닌 진동하는 패턴으로 인식한다

B를 재대로 설정하였다면, 여기서 128차원이 되었을 것이다.

4. Transformer PINN 1. Embedding

ONECLICK AI

Fourier Feature Mapping

5. 트랜스포머 임베딩

자 예시는 4 차원이였지만, 128차원이라고 치고 이어나가 보자.

이제 이 128차원 벡터는 물리적 위치 정보를 담은 하나의 토큰으로 취급된다.
단어가 벡터로 변한 것과 똑같은 상태.

좌표 하나 = 단어 하나

$H^{(0)} \in R^{N \times 128}$

$\boldsymbol{N}$ **:** 전체 점(Point)의 개수 (예: 10,000개)

128: 모델의 은닉 차원 (d_{model})

이제 이 좌표는 물리적 위치 정보를 담은 하나의 토큰(Token)이 되어 인코더로 들어간다.

4. Transformer PINN 2. Encoder

Multi Head Attention

각 좌표점들이 **서로의 물리적 상관관계(Correlation)**를 계산.

이 과정 덕분에 MLP 보다 압도적인 성능을 낼 수 있다.

1. Query, Key, Value

입력 $\mathbf{H}$를 h개의 헤드(Head)로 나누어 각각 다른 관점의 특징을 추출한다.

i는 헤드 인덱스

$$\mathbf{Q_i} = \mathbf{HW_i^Q}, \quad \mathbf{K_i} = \mathbf{HW_i^K}, \quad \mathbf{V_i} = \mathbf{HW_i^V}$$

$\mathbf{W_i^Q}, \mathbf{W_i^K}, \mathbf{W_i^V} \in R^{d_{model} \times d_k}$ (학습 파라미터)

4. Transformer PINN 2. Encoder

Multi Head Attention

$$\text{Head}_i = \text{Softmax}\left(\frac{\mathbf{Q_i K_i^T}}{\sqrt{d_k}}\right)\mathbf{V_i}$$

$\mathbf{Q_i K_i^T}$ 행렬($N \times N$)은 **모든 점들 사이의 관계**를 나타낸다.

이 값이 크다는 것은 점 A의 파동 변화가 점 B의 전자기장에 큰 영향을 미친다
인과관계가 있다 그런 뜻이다.

Softmax는 이 영향력을 확률적으로 정규화한다.

Multi Head Attention

여러 헤드에서 나온 결과를 합쳐서, 다양한 주파수 대역과 방향성을 동시에 고려한다.

$$\mathrm{MHA}(\mathbf{H}) = \mathrm{Concat}(\mathrm{Head}_1, \dots, \mathrm{Head}_h)\mathbf{W}^{\mathbf{O}}$$

$\mathbf{W}^{\mathbf{O}}$: 결과를 다시 d_{model} 차원으로 섞어주는 선형 변환.

4. Transformer PINN 2. Encoder

ONECLICK AI

Add & Norm

모델이 깊어져도 학습이 잘 되게

$$\mathbf{H}' = \mathrm{LayerNorm}\left(\mathbf{H}^{(\mathbf{0})} + \mathrm{MHA}\left(\mathbf{H}^{(\mathbf{0})}\right)\right)$$

Add : $\mathbf{H}^{(\mathbf{0})} + \cdots$

원래의 위치 정보(입력)를 잊어버리지 않도록 더해준다.
네가 어디에 있는지(x, y, z)를 잊지 말고, 주변 관계Attention를 추가하라는 의미이다.

LayerNorm : 데이터의 분포를 평균 0, 분산 1로 맞춰 학습 안정성을 확보.

4. Transformer PINN 2. Encoder

Feed Forword Network

Attention => 각 점 끼리의 관계
FFN => 점 자체의 **물리 함수를 근사한다.**

$$\mathrm{FFN}(\mathbf{x}) = \phi(\mathbf{x}\mathbf{W_1} + \mathbf{b_1})\mathbf{W_2} + \mathbf{b_2}$$

Linear → Activation → Linear

$\boldsymbol{\phi}(\cdot)$: 보통 ReLU나 GELU를 쓰지만, PINN에서는 미분을 두번 해야 하므로 **Tanh**나 **GELU**를 많이 사용한다.

지배 방정식의 비선형적인 해를 맞추기 위해 데이터를 변형하고 조작한다.

4. Transformer PINN 2. Encoder

Add & Norm 2

인코더 레이어의 마지막

$$\mathbf{H}^{(1)} = \mathrm{LayerNorm}\big(\mathbf{H}' + \mathrm{FFN}(\mathbf{H}')\big)$$

이 $\mathbf{H}^{(1)}$은 다음 인코더 레이어의 입력으로 들어가며, 이 과정을 L번 반복한다 (예: $L = 6$).

최종적으로 나온 $\mathbf{H}^{(\mathbf{L})}$은 시공간 상의 물리적 상태를 완벽하게 함축하고 있는 잠재 벡터가 된다.

Output Projection

마지막으로, 추상적인 128차원(d_{model}) 벡터를 우리가 원하는 물리량인 **전기장 벡터**로 변환한다.
디코딩이라 생각하면 된다.

$\hat{Y} = \mathbf{H}^{(\mathbf{L})} W_{out} + b_{out}$ 여기서 W_{out}은 128×3 크기의 행렬이다.

$$\widehat{Y_i} = \left[E_x(x_i), \quad E_y(x_i), \quad E_z(x_i)\right]$$

E_x: x축 방향 전기장 성분
E_y: y축 방향 전기장 성분
E_z: z축 방향 전기장 성분

이 값들은 특정 위치 (x, y, z)에서의 **전기장 세기와 방향**을 나타낸다.

4. Transformer PINN 4. Loss Function

Loss Function

$\mathcal{L} = \mathcal{L}_{\mathcal{PDE}} + \mathcal{L}_{\mathcal{BC}}$

$\boldsymbol{\mathcal{L}}_{\mathrm{BC}}$: 경계 조건이나 초기 조건 예: 금속 표면에서는 전기장이 0이어야 함 $E_{tan} = 0.$

$\boldsymbol{\mathcal{L}}_{\mathcal{PDE}}$: 모델의 출력 E를 다시 x, y, z, t로 자동 미분하여 지배 방정식을 만족하는지 확인.

이 Loss가 0이 되도록 역전파를 수행하여, 앞선 모든 가중치들을 업데이트한다.

이 과정을 통해서 트랜스포머 PINN이 완료된다.

5. Writing a PINN Paper

기본 논리? flow

실험은 돈 많이 들어가고 시간도 오래걸린다.

그래서 실험하기 전에 시뮬레이션으로 근사값을 얻는다

그런데, FEKO 나 HFSS오래걸리고 실물 안테나 크기가 너무 커 지면 실험이 안된다.

하지만, 이를 PINN을 통해서 해석할 수 있다.

실험 환경을 데이터가 충분하지 않고 복잡한 비선형 문제가 있는 환경으로 할 수록 논문 난이도가 낮아진다.

5. Writing a PINN Paper

모델에 대한 증명

PINN의 지배방정식을 같은 구조의 아키텍쳐가 풀 수 있다 로 가면 안된다.

예를 들어, 지배 방정식을 PINN에 넣고, 올바른 답이 나오는 것은 근거로 사용할 수 없다

때문에, 이어지는 내용에 대해서 증명해야 한다.

1. 먼저, 해당 아키텍쳐가 그 방정식을 풀 수 있음을 모델을 한 번 사용하는 걸 통해서 증명한다.

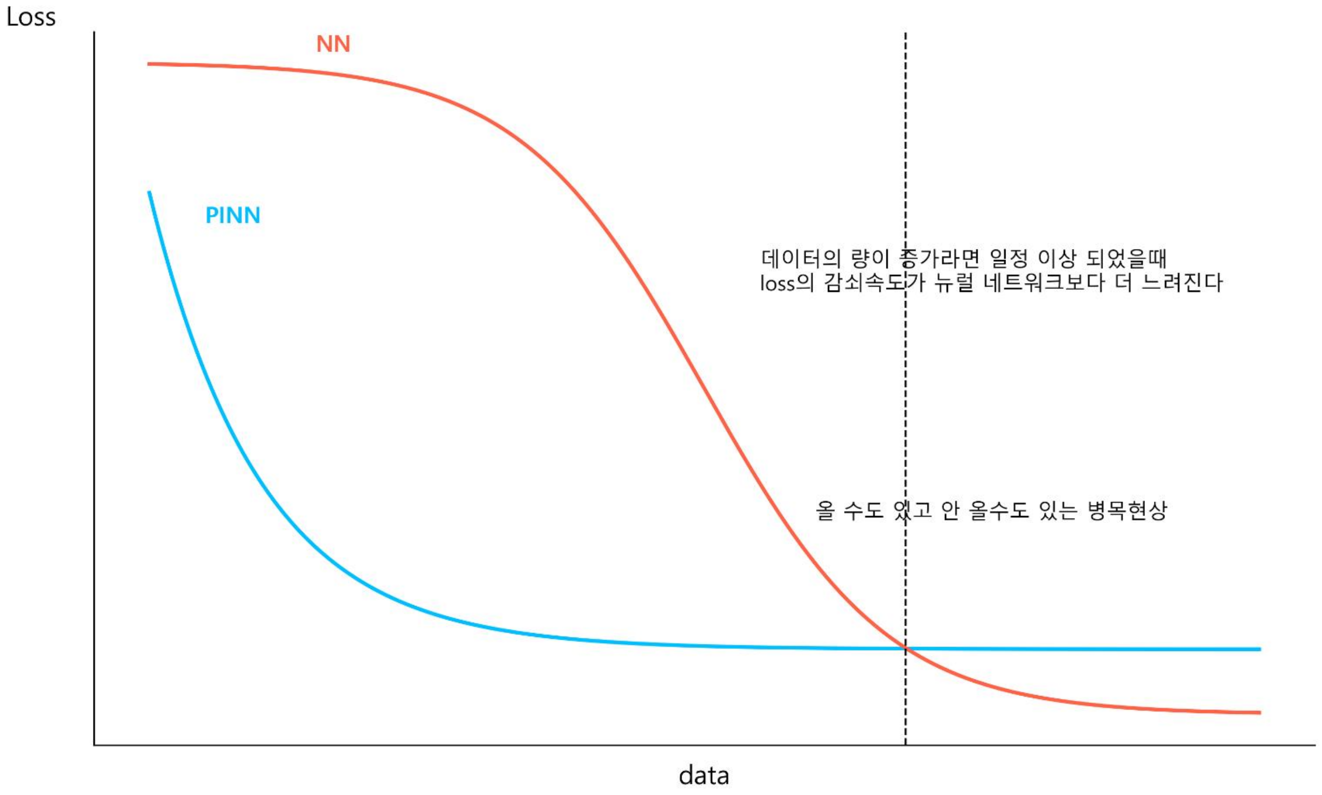
2. 이어서, 다음에 대한 비교를 진행한다.
    1. 실험 결과
    2. 시뮬레이션 결과(FEKO)
    3. PINN 결과
    4. MLP 결과

3, 4 는 무조건 비교에 + 1 아니면 2 를 최소한으로 가야 한다

5. Writing a PINN Paper

PINN 단점

PINN과 뉴럴 네트워크를 비교했을 때,



PINN으로 풀 수 있는거는
무조건 NN으로도 풀 수 있다

5. Writing a PINN Paper

문제점

예시로, 1000m 리플렉터 안테나 등은 FEKO나 HFSS로는 해석이 불가능하다.

그러면 앞선 시뮬레이션과 비교가 불가능한게 아닌가요??

이럴 때에는 실제 실험 데이터를 연구소에 수소문 해서 얻은 다음,
그 데이터와 비교해야 리젝 당하지 않는다.

5. Writing a PINN Paper

문제점

먼저, FEKO로 실험 가능한 사이즈로 실험을 한다. 예를들어, 1m , 2m, 이런식으로
실험할 수 있는 최대 크기까지 한다

이어서, 같은 조건 크기로 PINN으로 한 실험결과를 가지고 온다

이 둘을 비교하며 오차 점점 줄어드는 것을 그래프로 제일 먼저 보여주어야 한다

이후 FEKO 로 실험 못하는 크기의 안테나 해석한 다음, 그걸 결론으로 제출해야 한다.

5. Writing a PINN Paper

독이든 성배와 같은 PINN

논문 수가 적어 기회의 장 같아 보이지만, 뛰어들면 쉽지 않다.

감사합니다