

Paper Review: Attention Is All You Need

Revisiting the Architecture via Matrix Operations and Complexity Analysis

김미림, 김상엽, 김진욱, 유지아, 임재범, 정필상

Jan. 24th, 2026

목차

- 1 Transformer의 등장배경과 기존모델과의 비교
- 2 Transformer Architecture
 - Encoder
 - Decoder
- 3 Why Self-Attention? 계산복잡도를 중심으로.
- 4 Transformer 훈련과정 및 성능

Abstract

이 논문이 나오기 전까지 시퀀스 모델링의 지배적인 흐름은 RNN이나 CNN이 메인이고, Attention은 이를 보조하는 역할에 불과했다.

- **Main Architecture:** RNN or CNN.
- **Role of Attention:** 인코더와 디코더를 연결하는 보조적인 역할.

그러나 이 논문은 말합니다.

→ "Attention Is All You Need"

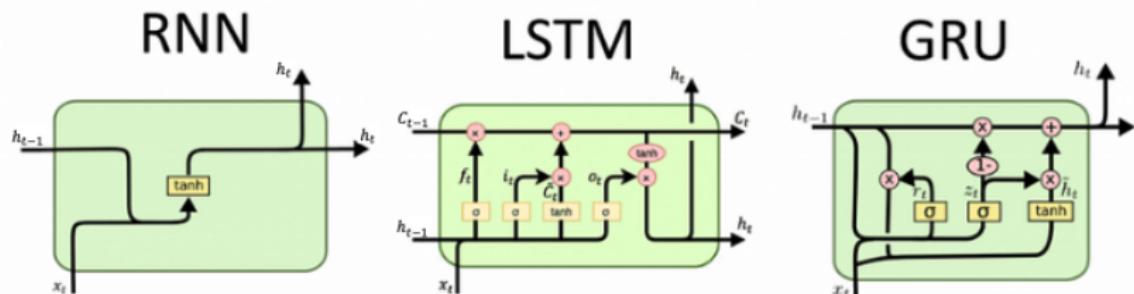
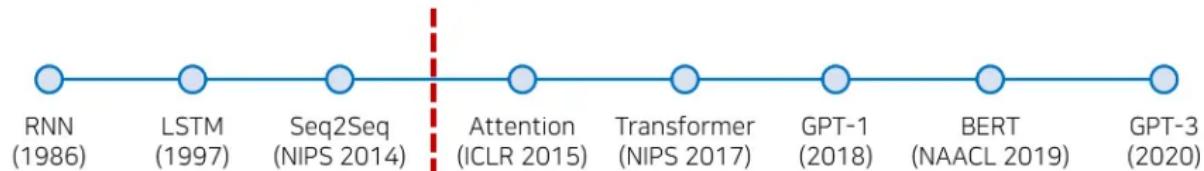
Abstract

본 논문은 복잡한 **순환**이나 **합성곱**을 모두 제거하고, 오직 **Self-Attention**만으로 인코더와 디코더를 구성한 **Transformer**라는 모델을 제안한다.

- 기존 RNN이 가진 **시간 의존성**을 제거함
- 완전한 **병렬 처리**를 가능하게 함
- 번역 품질과 학습 속도 면에서 **SOTA**를 달성

“그렇다면 왜 잘 쓴 RNN을 대신할
새로운 아키텍처가 필요했을까요?”

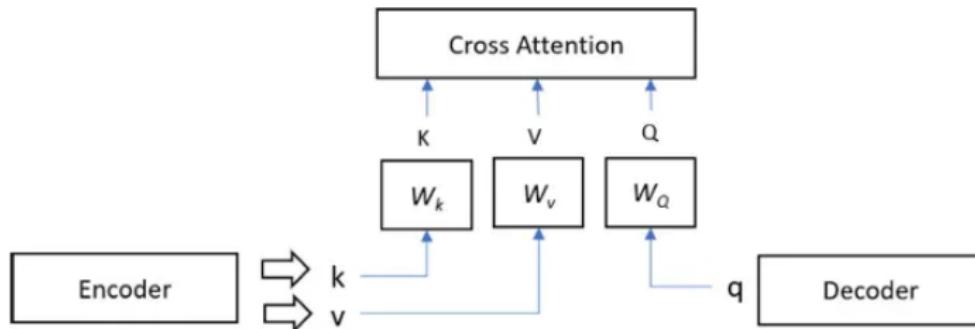
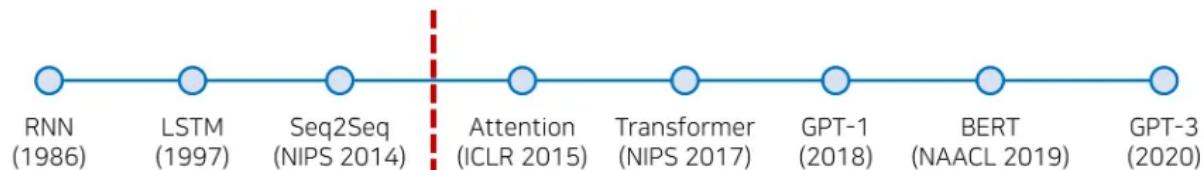
Limitation 1: Standard RNN



- **Structural Limit:** 입력과 출력의 길이가 같아야 하는 동기식 구조.
- **No Parallelization:**

$$h_t = f(h_{t-1}, x_t)$$

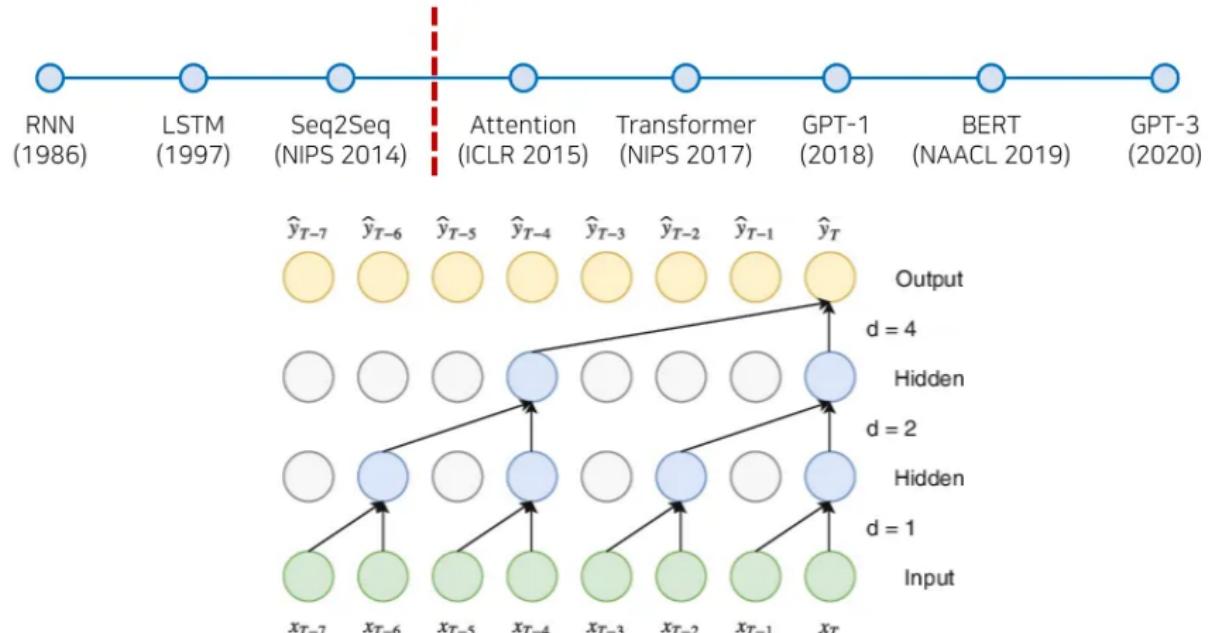
Limitation 2: Seq2Seq with Attention



- **Solution:** Encoder-Decoder로 구조를 분리하여 가변 길이 문제 해결.
- **New Problem:** 문장이 길어지면 정보 손실 발생.

→ 연관성이 높은 정보에 집중하자!

Limitation 3: Convolutional Models



- RNN의 속도 문제를 해결하기 위해, 병렬 처리가 가능한 CNN 도입.
- 한 번에 볼 수 있는 시야가 고정되어, 레이어를 아주 깊게 쌓아야 함.

Why Transformer?

Transformer는 RNN의 순차성과 CNN의 좁은 시야 문제를 모두 해결한다.

- ① **RNN**: 전체 문맥은 보지만 **병렬화 불가로 느림**.
- ② **CNN**: 병렬화는 되지만 **멀리 있는 정보 보기 어려움**.
- ③ **Transformer**: **병렬화 가능**하고, 거리 제약 없이 **모든 정보를 한 번에 참조**.

→ Recurrence나 Convolution 없이, 오직 Self-Attention만으로 입출력을 계산하는 **최초의 시퀀스 변환 모델**

The Overview of the Model Architecture

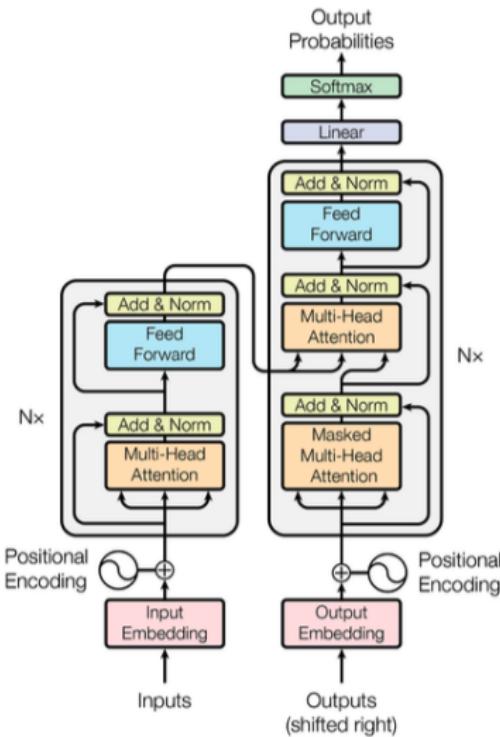


Figure 1: The Transformer - model architecture.

Input Embedding

“I want to go home.”

$$\begin{bmatrix} 0.24 & -1.05 & 0.82 & \dots & -0.41 \\ -0.91 & 0.33 & -0.14 & \dots & 0.76 \\ 1.56 & -0.88 & 0.09 & \dots & -1.23 \\ -0.32 & 0.67 & -1.90 & \dots & 0.44 \\ 0.05 & -0.21 & 0.55 & \dots & -0.98 \end{bmatrix}$$

$d_{model}=512$

Positional Encoding

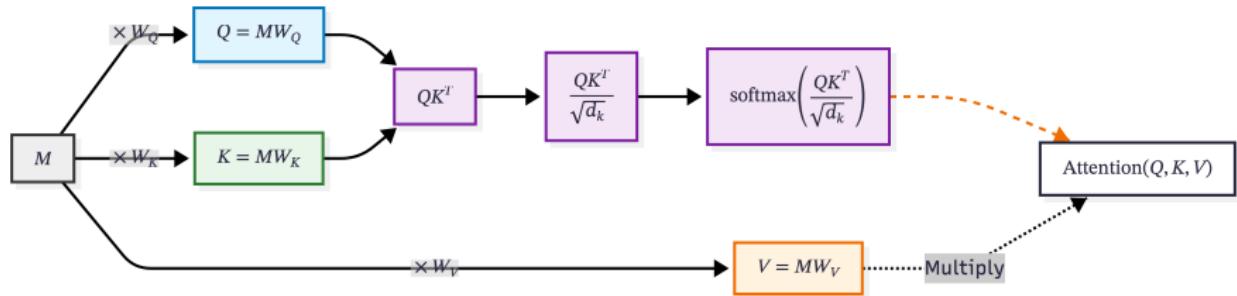
$$M_{ij} \leftarrow M_{ij} + PE_{ij}$$

, where

$$PE_{ij} = \begin{cases} \sin(i/10000^{j/d_{\text{model}}}) & \text{if } j \text{ is even} \\ \cos(i/10000^{(j-1)/d_{\text{model}}}) & \text{if } j \text{ is odd} \end{cases}$$

$1 \leq i \leq n$ (number of tokens), $1 \leq j \leq d_{\text{model}} = 512$.

Single-Head Self Attention (SSA)



Here, $W_Q, W_K \in \mathbb{R}^{d_{model} \times d_k}$, and $W_V \in \mathbb{R}^{d_{model} \times d_v}$. The softmax is implemented row-wise.

How do attention heads learn various contexts of sentences?

"The **animal** didn't cross the **street** because **it** was too tired."



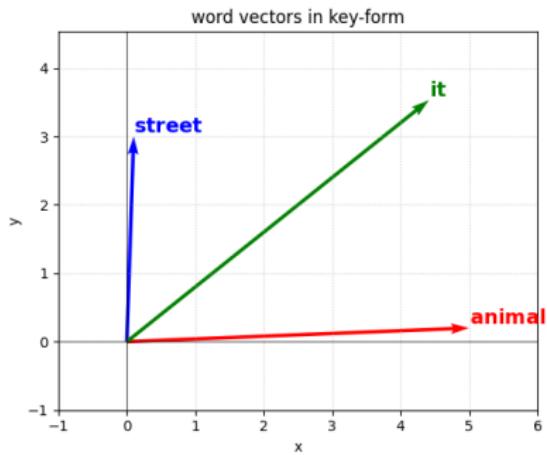
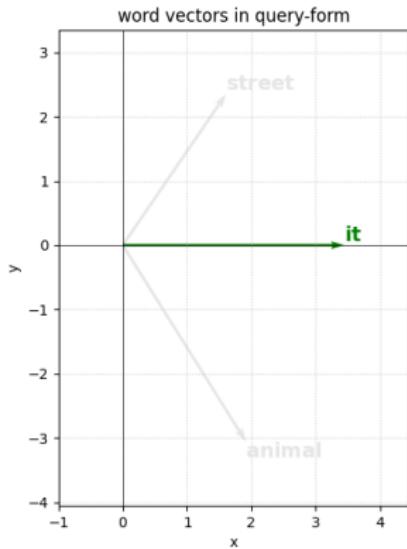
For simplicity, suppose $d_{\text{model}} = 2$.

$$\mathbf{it} = \begin{pmatrix} 3.14 \\ 1.45 \end{pmatrix} \quad (1)$$

$$\mathbf{street} = \begin{pmatrix} 0.37 \\ 2.81 \end{pmatrix} \quad (2)$$

$$\mathbf{animal} = \begin{pmatrix} 2.99 \\ -1.99 \end{pmatrix} \quad (3)$$

The role of Q, K, and V



$$W^Q = \begin{pmatrix} 0.91 & -0.41 \\ 0.41 & 0.91 \end{pmatrix}$$

$$W^K = \begin{pmatrix} 1.51 & 0.70 \\ -0.23 & 0.95 \end{pmatrix}$$

Attention Score Computation

$$\frac{QK^T}{\sqrt{d_k}} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1.91 & -3.06 \\ 1.61 & 2.35 \\ 3.45 & 0.00 \end{pmatrix} \begin{pmatrix} 5.00 & 0.10 & 4.41 \\ 0.20 & 3.00 & 3.53 \end{pmatrix} \quad (4)$$

$$= \begin{pmatrix} 6.34 & -6.35 & -1.65 \\ 6.01 & 5.09 & 10.87 \\ 12.18 & 0.24 & 10.76 \end{pmatrix} \quad (5)$$

$$\text{softmax} \left(\frac{QK^T}{\sqrt{d_k}} \right) = \begin{pmatrix} \dots & \dots & \dots \\ \dots & \dots & \dots \\ 0.81 & 0.00 & 0.19 \end{pmatrix}$$

How words attend each other

For simplicity, assume $V = M$, i.e., $W^V = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$. Then,

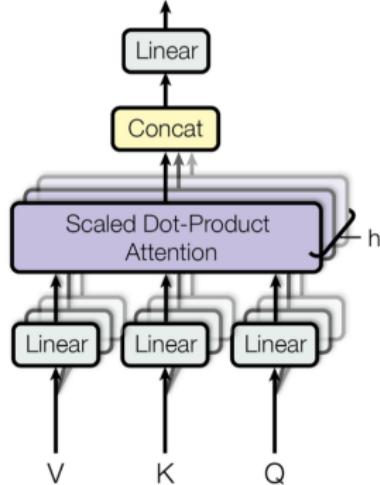
$$\begin{pmatrix} \textcolor{red}{animal}' \\ \textcolor{blue}{street}' \\ \textcolor{green}{it}' \end{pmatrix} = \text{Attention}(Q, K, V) \quad (6)$$

$$= \begin{pmatrix} \dots & \dots & \dots \\ \dots & \dots & \dots \\ 0.81 & 0.00 & 0.19 \end{pmatrix} \begin{pmatrix} \textcolor{red}{2.99} & \textcolor{red}{-1.99} \\ \textcolor{blue}{0.37} & \textcolor{blue}{2.81} \\ \textcolor{green}{3.14} & \textcolor{green}{1.45} \end{pmatrix} \quad (7)$$

$$= \begin{pmatrix} \dots \\ \dots \\ \dots \\ 0.81\textcolor{red}{animal} + 0.00\textcolor{blue}{street} + 0.19\textcolor{green}{it} \end{pmatrix} \quad (8)$$

Multi-Head Attention

Multi-Head Attention



For $1 \leq i \leq h$, $W_i^Q, W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$,
 $W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$, where $Q, K, V \in \mathbb{R}^{n \times d_{\text{model}}}$

$$Q_i = QW_i^Q$$

$$K_i = KW_i^K$$

$$V_i = VW_i^V$$

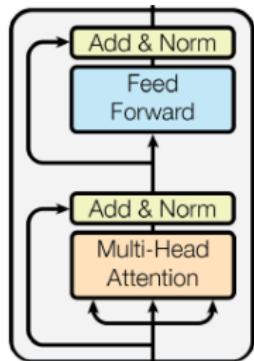
Put $A_i = \text{Attention}(Q_i, K_i, V_i)$.

$$\text{Multihead}(Q, K, V) := \text{Concat}(A_1, \dots, A_h)W^O$$

, where $W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$

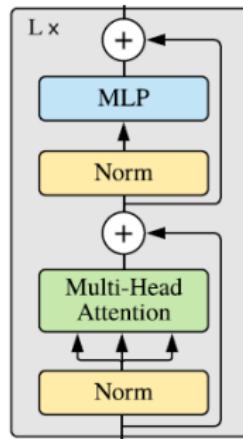
Normalization: Post-LN vs. Pre-LN

Post-LN (Original)



$$x_{l+1} = \text{LN}(x_l + f(x_l))$$

Pre-LN (ViT, Modern)



$$x_{l+1} = x_l + f(\text{LN}(x_l))$$

Feed-Forward Networks

$$\text{FFN}(M_i) = \max(0, M_i W_1 + b_1) W_2 + b_2 \quad (9)$$

. Suppose

$$\text{apple} = (1 \ 1 \ 0)$$

, where

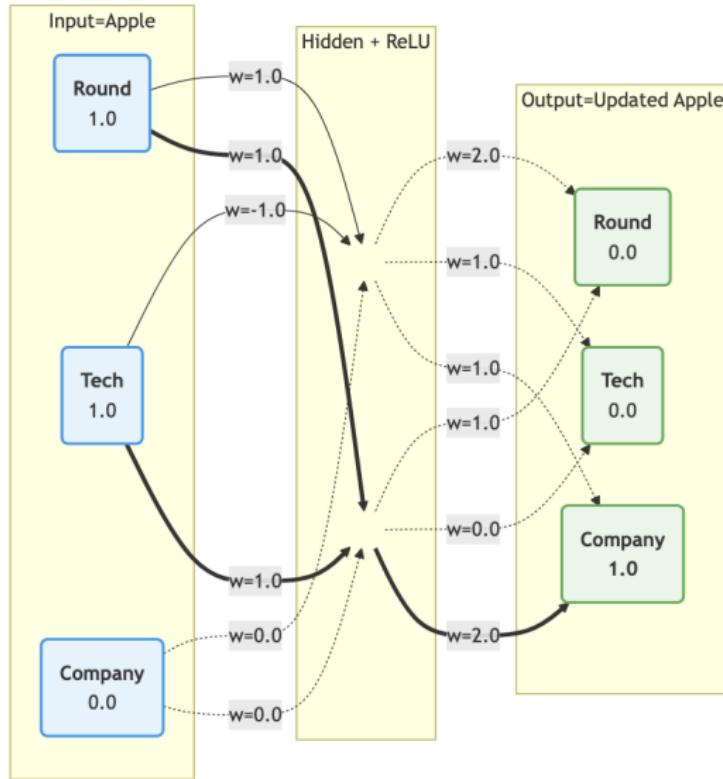
- dimension 1 represents the “roundness”
- dimension 2 represents “relation to tech”
- dimension 3 represents “whether it’s a company”

Can we make

$$\text{FFN}(\text{apple} = (1, 1, 0)) = (1, 1, 1)$$

?

How words obtain specific meanings after FFN's



Decoder: Two Types of Attention

디코더에서도 인코더와 유사하게 Attention과 FFN을 거치지만, Attention의 목적과 방식에서 결정적인 차이가 있다. 디코더 블록은 다음과 같이 두 종류의 Multi-Head Attention을 포함한다.

- ① **Masked Multi-Head Attention**: 자기 자신의 과거 토큰들만 참조
- ② **Multi-Head (Cross) Attention**: 인코더의 정보를 가져와 참조

이 구조를 통해 디코더는 인코더가 추출한 문맥 정보를 바탕으로 적절한 다음 단어를 하나씩 생성해 나간다.

1. Masked Multi-Head Attention

왜 마스킹(Masking)이 필요한가? 정답 유출(Cheating) 방지를 위해!

- **문제점:** 학습 시에는 타겟 문장 전체가 입력으로 들어와 마스킹이 없다면 모델은 현재 단어를 예측할 때 미래에 올 단어를 미리 보고 베끼는 법을 배우게 됨
- **해결책:** i 번째 단어를 생성할 때, $i + 1$ 번째 이후의 위치를 $-\infty$ 로 처리하여 Attention Score를 0으로 만듦

결과적으로 모델은 오직 과거의 단어들에만 의존하여 다음 단어를 추론하는 법을 배웁니다.

1. Masked Attention: Masking & Softmax

예문을 통해 직접 마스킹(masking) 과정을 살펴보자.

“I really like python”

Raw Scores	Look-ahead Mask	Masked Scores
$\begin{bmatrix} 15.2 & 10.1 & 8.5 & 9.4 \\ 8.4 & 12.1 & 7.3 & 11.2 \\ 4.7 & 9.2 & 14.5 & 8.1 \\ 3.1 & 5.8 & 10.2 & 13.8 \end{bmatrix}$	$+ \begin{bmatrix} 0 & -\infty & -\infty & -\infty \\ 0 & 0 & -\infty & -\infty \\ 0 & 0 & 0 & -\infty \\ 0 & 0 & 0 & 0 \end{bmatrix}$	$= \begin{bmatrix} 15.2 & -\infty & -\infty & -\infty \\ 8.4 & 12.1 & -\infty & -\infty \\ 4.7 & 9.2 & 14.5 & -\infty \\ 3.1 & 5.8 & 10.2 & 13.8 \end{bmatrix}$

Softmax (Masked Scores) \longrightarrow

$$\begin{bmatrix} 1.0 & 0 & 0 & 0 \\ 0.02 & 0.98 & 0 & 0 \\ 0.01 & 0.08 & 0.91 & 0 \\ 0.00 & 0.02 & 0.15 & 0.83 \end{bmatrix}$$

2. Cross Multi-Head Attention

Cross Attention은 인코더와 디코더를 잇는 다리 역할을 한다. 이를 위해 Q , K , V 의 출처가 달라진다.

- **Query (Q)**: 디코더의 이전 레이어 출력 (번역할 위치의 정보)
- **Key (K) & Value (V)**: **인코더의 최종 출력 행렬** (원문의 전체 문맥)

디코더의 Q 가 인코더의 K 와 비교되어, "현재 번역하려는 단어가 원문의 어떤 부분과 가장 연관이 깊은가?"를 계산하여 해당 부분의 V 를 집중적으로 가져온다. **치팅하는 것 아님!**

2. Cross Multi-Head Attention: 정보의 결합

한글을 영어로 번역할 때, 인코더의 **한글 원문 전체(K, V)**와 디코더의 **현재 영어 단어(Q)**가 어떻게 결합되는지 살펴보자.

Encoder (Source)

"나는 파이썬을 정말 좋아한다"



Decoder (Target)

"I really"

\times



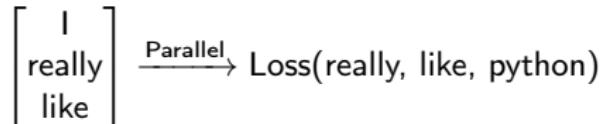
$$\text{Attention}(Q, K, V) = \text{Softmax} \left(\frac{Q \times K^T}{\sqrt{d_k}} \right) \times V$$

- **Query (Q)**: 디코더에서 발생. "지금 내가 번역할 단어에 필요한 정보는?"
- **Key (K), Value (V)**: 인코더에서 제공. "원문 문장 전체의 정보는 이거야."
- **효과**: 원문(Source)의 전체 맥락 중 현재 출력에 가장 필요한 부분을 찾아 집중함.

Attention in Application: Training vs. Inference

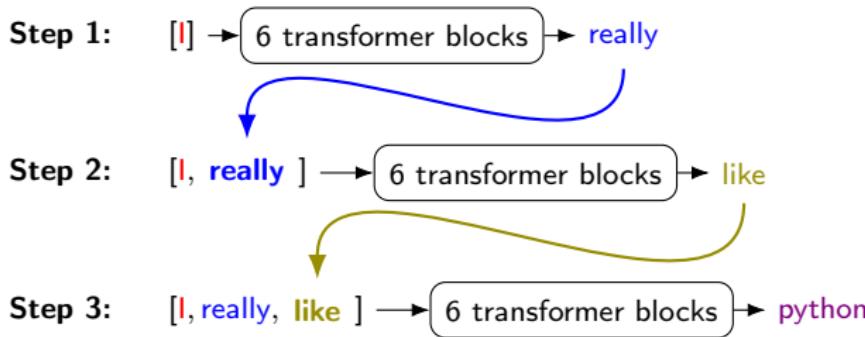
1. 학습 시 (Training): Parallel Processing

마스크를 통해 미래 단어를 가린 채 모든 단어를 한꺼번에 처리한다.



2. 추론 시 (Inference): Auto-regressive Generation

생성된 단어가 다시 입력이 되는 **순차적(Step-by-step)** 과정이다.



특징: 추론 단계에서는 자신이 생성한 이전 단어들에 의존하여 EOS까지 반복한다.

Final Layers: Linear & Softmax

디코더 블록의 최종 출력인 은닉상태($H, n \times d_{model}$)를 Linear, Softmax Layer를 거쳐 단어 확률 분포($P, n \times V$)로 변환한다.

Probability Distribution ($P \in \mathbb{R}^{n \times V}$)

Input Token	really	like	python	is	EOS
I	0.82	0.05	0.01	0.10	0.02
really	0.01	0.78	0.03	0.15	0.03
like	0.00	0.02	0.91	0.02	0.05

연산 과정:

- ① **Linear Layer:** $Z = H \times W_{linear} + b$
 - 각 값은 해당 위치에 i번째 단어가 올 점수(Logit)임
- ② **Softmax Layer:** $P = \text{Softmax}(Z)$
 - 0 1 사이의 단어 발생 확률로 변환

결과 및 활용:

학습 시: 예측 확률과 정답(Label) 간의 Loss 계산 후 **Backpropagation** 수행
추론 시: 각 행에서 가장 확률이 높은 단어 하나를 최종 출력

Summary: Learnable Parameters

Transformer 모델이 학습을 통해 업데이트하는 모든 파라미터는 아래와 같다.

구분	학습 가능한 파라미터
Embedding	Token Embedding Table, Positional Embedding
Attention	W_i^Q, W_i^K, W_i^V (헤드별), W^O (결합용)
FFN	W_1, b_1 (팽창용), W_2, b_2 (압축용)
Norm	Layer Normalization의 Scale(γ) 및 Shift(β)
Output	Final Linear Layer's W and b

이 수많은 가중치들이 수만 장의 데이터를 통해 최적화되어 문맥을 이해하게 된다.

Why Self-Attention?

① complexity per layer

한 레이어에서 필요한 연산량을 비교하여, self-attention이 문장 길이에 따라 어떻게 계산 비용이 증가하는지를 보여줌

② sequential operations

연산이 순차적으로 이루어지는지, 아니면 모든 토큰을 병렬로 동시에 처리할 수 있는지를 비교

③ maximum path length

두 단어 사이 정보가 전달되기 위해 거쳐야 하는 최대 단계 수를 비교

연산 특성 비교를 위한 기호 정의

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(\log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

- n : sequence length
- d : representation dimension
- k : kernel size of convolutions
- r : size of the neighborhood in restricted self-attention

Complexity per Layer 부|교

- **Self-Attention**

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \rightarrow O(n^2d)$$

- **Recurrent (RNN)**

$$h_t = \phi(W_x x_t + W_h h_{t-1}) \rightarrow O(nd^2)$$

- **Convolution (CNN)**

$$h_t = \phi\left(\sum_{i=0}^{k-1} W_i x_{t+i} + b\right) \rightarrow O(knd^2)$$

- **Restricted Self-Attention**

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

Sequential Operations 비교

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(\log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

- Self-Attention: 모든 토큰을 병렬적으로 처리 $\Rightarrow O(1)$
- Recurrent: 이전 시점 출력 필요 $\Rightarrow O(n)$
- Convolution: 위치별 병렬 연산 가능 $\Rightarrow O(1)$
- Restricted Self-Attention: 병렬 연산 가능 $\Rightarrow O(1)$

Maximum Path Length 비교

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(\log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

- Self-Attention: 모든 토큰이 직접 연결 $\Rightarrow O(1)$
- Recurrent: 순차적 정보 전달 $\Rightarrow O(n)$
- Convolution: k 의 크기에 따라 제한, layer가 쌓이면 감소 $\Rightarrow O(\log_k n)$
- Restricted Self-Attention: r 범위 연결 $\Rightarrow O(n/r)$

Self-Attention, RNN, CNN의 연산 특성 비교

- 예시 문장: I want to go home
- 이를 활용하여 각 레이어 구조의 계산 복잡도, 순차 연산 수, 최대 경로 길이를 수치 예시로 비교

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$\mathcal{O}(n^2 \cdot d) = 5^2 \cdot 512 = 12,800$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
Recurrent (RNN)	$\mathcal{O}(n \cdot d^2) = 5 \cdot 512^2 = 1,310,720$	$\mathcal{O}(n)$	$\mathcal{O}(n)$
Convolution (CNN)	$\mathcal{O}(k \cdot n \cdot d^2) = 2 \cdot 5 \cdot 512^2 = 2,621,440$	$\mathcal{O}(1)$	$\mathcal{O}(\log_k n)$
Self-Attention (restricted)	$\mathcal{O}(r \cdot n \cdot d) = 1 \cdot 5 \cdot 512 = 2,560$	$\mathcal{O}(1)$	$\mathcal{O}(n/r)$

본 예시에서는 문장이 매우 짧기 때문에 국소적 문맥만을 고려하는 설정을 가정하여 CNN의 커널 크기와 **restricted self-attention**의 이웃 크기를 각각 $k = 2$, $r = 1$ 로 설정하였다.

왜 Self-Attention을 사용해야 하는가

Self-Attention은 순차적으로 계산하지 않고 병렬적으로 모든 토큰 간의 관계를 계산할 수 있어, 연산량(복잡도)은 크지만 순차 연산 단계와 정보 전달 경로가 짧아 학습과 장거리 의존성 모델링에 유리하다.

Training Details

DataSet:

WMT 2014 English-German dataset : encoded using byte-pair encoding, shared source target vocabulary of about 37000 tokens

WMT 2014 English-French dataset: 36M sentences and split tokens into a 32000 word-piece vocabulary

Hardware:

8 NVIDIA P100 GPU

Optimizer:

Adam with $\beta_1 = 0.9$, $\beta_2 = 0.98$ and $\epsilon = 10^{-9}$

Regularization:

Dropout($p_dropout = 0.1$ & Residual)

Model Variations

	N	d_{model}	d_{ff}	h	d_k	d_v	P_{drop}	ϵ_{ls}	train steps	PPL (dev)	BLEU (dev)	params $\times 10^6$			
base	6	512	2048	8	64	64	0.1	0.1	100K	4.92	25.8	65			
(A)				1	512	512				5.29	24.9				
				4	128	128				5.00	25.5				
				16	32	32				4.91	25.8				
				32	16	16				5.01	25.4				
(B)					16					5.16	25.1	58			
					32					5.01	25.4	60			
(C)				2						6.11	23.7	36			
				4		32				5.19	25.3	50			
				8						4.88	25.5	80			
				256						5.75	24.5	28			
				1024		128				4.66	26.0	168			
				1024						5.12	25.4	53			
				4096						4.75	26.2	90			
(D)							0.0			5.77	24.6				
							0.2			4.95	25.5				
							0.0			4.67	25.3				
							0.2			5.47	25.7				
(E)	positional embedding instead of sinusoids									4.92	25.7	213			
big	6	1024	4096	16			0.3	300K	4.33	26.4					

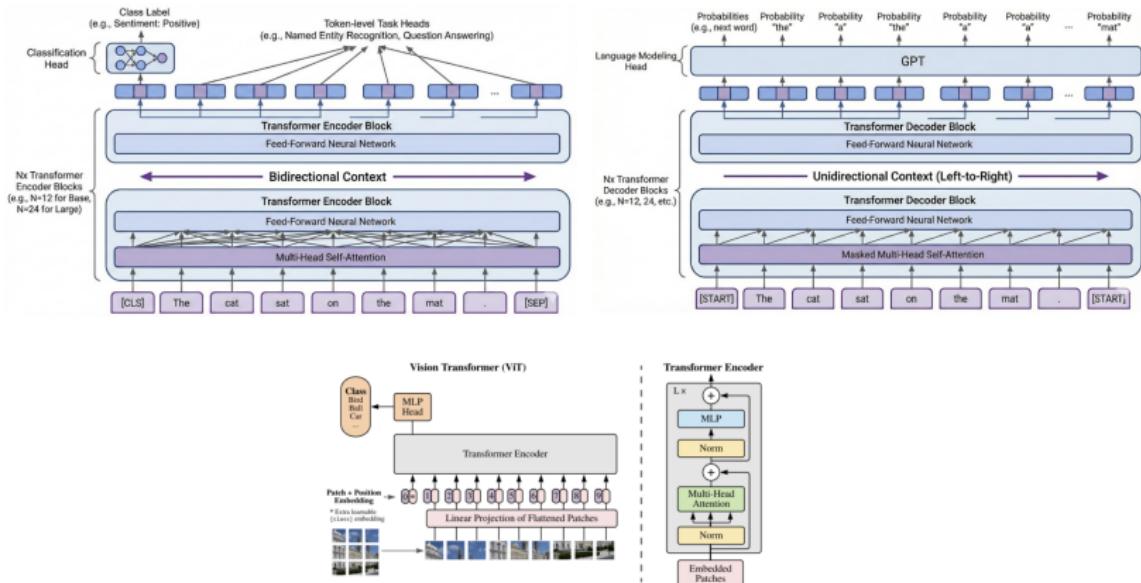
Results

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [38]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	26.36	41.29	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1		$3.3 \cdot 10^{18}$
Transformer (big)	28.4	41.8		$2.3 \cdot 10^{19}$

We are excited about the future of attention-based models and plan to apply them to other tasks. We plan to extend the Transformer to problems involving input and output modalities other than text and to investigate local, restricted attention mechanisms to efficiently handle large inputs and outputs such as images, audio and video. Making generation less sequential is another research goals of ours.



future of attention-based models



Bibliography I

Kyunghyun Cho et al. "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 2014.

Jacob Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. 2019, pp. 4171–4186.

Alexey Dosovitskiy et al. "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale". In: *International Conference on Learning Representations*. 2021.

Dan Hendrycks and Kevin Gimpel. "Gaussian error linear units (gelus)". In: *arXiv preprint arXiv:1606.08415* (2016).

Bibliography II

Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780.

Dongbin Na. *Transformer: Attention Is All You Need Paper Review*. YouTube. Source for Timeline & Diagram Styles. 2021.

Alec Radford et al. *Improving language understanding by generative pre-training*. 2018.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. "Learning representations by back-propagating errors". In: *Nature* 323.6088 (1986), pp. 533–536.

Ashish Vaswani et al. "Attention is all you need". In: *Advances in neural information processing systems*. 2017.

Ruixin Xiong et al. "On layer normalization in the transformer architecture". In: *International Conference on Machine Learning*. 2020, pp. 10524–10533.

Bibliography III

Yifan Zhang and Peter Fitch. "Multi-task temporal convolutional network for predicting water quality sensor data". In: *Communications in Computer and Information Science*. Springer, 2019.