LLaMA: Open and Efficient Foundation Language Models

arXiv(2023), Meta AI, 8982 citation

1 Introduction

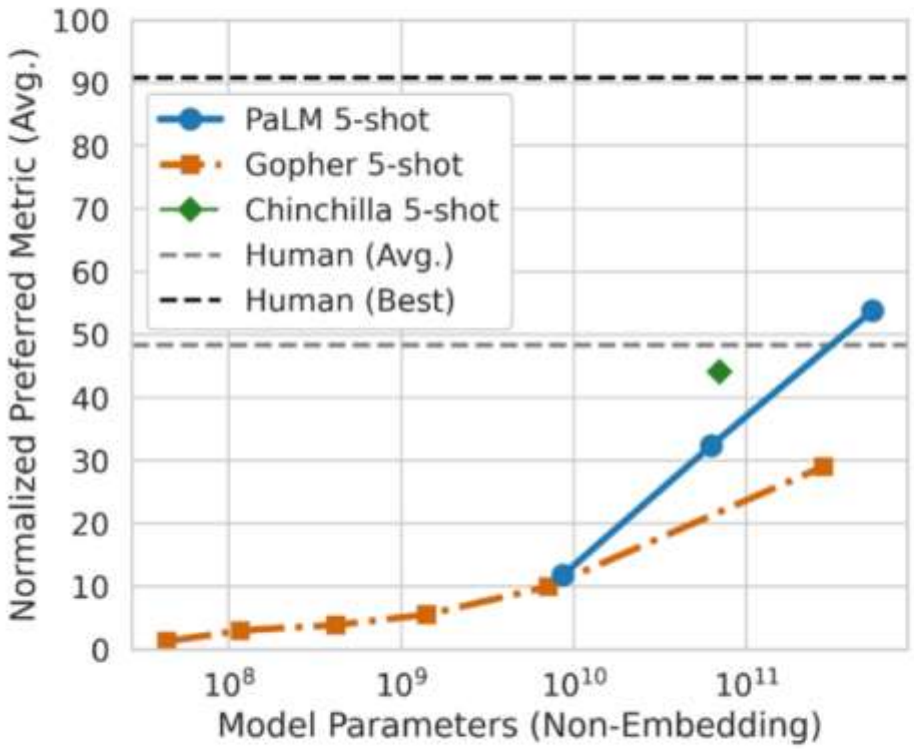
- LLaMA
  - Meta AI에서 공개한 LLM model (7B ~ 65B)
  - Public data만을 사용하여 학습
  - LLaMA-13B vs GPT-3 (175B)
    - LLaMA-13B: Single GPU로 inference가 가능하다.

B: 10억

T: 1조

1 Introduction

- LLM은 textual instruction & few example로부터 새로운 task를 수행할 수 있다.
  - 위와 같은 few-shot 능력은 모델이 충분한 크기에 도달할 때 발생한다. [OpenAI, 2020]
  - Gopher (2021), PaLM (2022)와 같이 모델의 크기를 키우는 연구들이 이뤄졌다.
  - 이러한 노력들은 더 많은 parameter가 더 나은 성능을 이끈다는 가정을 바탕으로 한다.



1 Introduction

- Chinchilla [DeepMind, 2022]
  - Compute budget이 주어졌을 때의 최고의 성능:
    모델의 크기를 키우는 게 아닌, 작은 모델을 더 많은 데이터로 학습시키는 걸 제안
  - 하지만 이는 training compute budget에 대한 관점이다

Model	Size (# Parameters)	Training Tokens
LaMDA (Thoppilan et al., 2022)	137 Billion	168 Billion
GPT-3 (Brown et al., 2020)	175 Billion	300 Billion
Jurassic (Lieber et al., 2021)	178 Billion	300 Billion
Gopher (Rae et al., 2021)	280 Billion	300 Billion
MT-NLG 530B (Smith et al., 2022)	530 Billion	270 Billion
Chinchilla	70 Billion	1.4 Trillion

	Chinchilla	*Gopher*	GPT-3	MT-NLG 530B
LAMBADA Zero-Shot	**77.4**	74.5	76.2	76.6
RACE-m Few-Shot	**86.8**	75.1	58.1	-
RACE-h Few-Shot	**82.3**	71.6	46.8	47.9

1 Introduction

- LLaMA
  - Inference budget 관점에서 language model을 학습시키겠다
  - 더 작은 모델을 더 많은 token으로 학습시키겠다
  - Chinchilla에서는 10B model을 학습시키기 위해 200B token을 권장한다
  - LLaMA는 7B에 1T token을 사용

params	dimension	n heads	n layers	learning rate	batch size	n tokens
6.7B	4096	32	32	$3.0e^{-4}$	4M	1.0T
13.0B	5120	40	40	$3.0e^{-4}$	4M	1.0T
32.5B	6656	52	60	$1.5e^{-4}$	4M	1.4T
65.2B	8192	64	80	$1.5e^{-4}$	4M	1.4T

Table 2: **Model sizes, architectures, and optimization hyper-parameters.**

2 Pre-training data

- Public dataset
  - General: CommonCrawl, C4, Wikipedia, Books, StackExchange
  - Scientific data: ArXiv
  - Code: Github
- Tokenizer: Byte-pair encoding (BPE)
  - Text를 byte단위로 쪼개서 자주 반복되는 byte쌍을 묶는다 (반복)
  - 숫자는 모두 개별 분리

Dataset	Sampling prop.	Epochs	Disk size
CommonCrawl	67.0%	1.10	3.3 TB
C4	15.0%	1.06	783 GB
Github	4.5%	0.64	328 GB
Wikipedia	4.5%	2.45	83 GB
Books	4.5%	2.23	85 GB
ArXiv	2.5%	1.06	92 GB
StackExchange	2.0%	1.03	78 GB

3 Experiments

- Model: Transformer architecture (“Attention is all you need”)

Zero-shot Textual description for task

Few-shot + few example

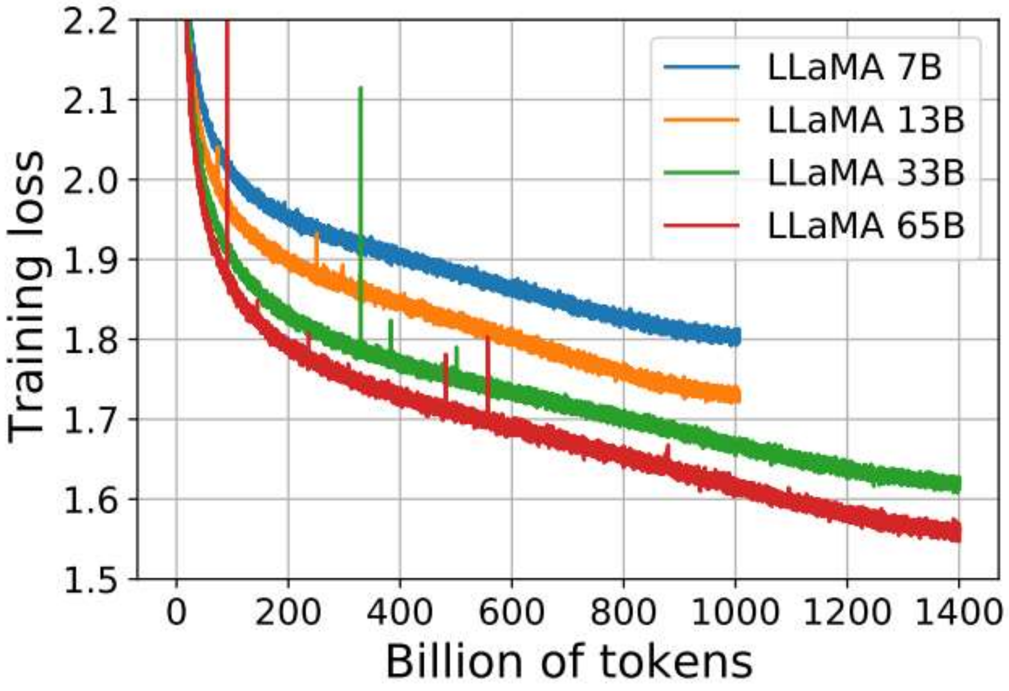


Figure 1: **Training loss over train tokens for the 7B, 13B, 33B, and 65 models.** LLaMA-33B and LLaMA-65B were trained on 1.4T tokens. The smaller models were trained on 1.0T tokens. All models are trained with a batch size of 4M tokens.

Experiments

- Zero-shot performance on Common Sense Reasoning tasks (객관식 > accuracy)
  - GPT-3가 175B vs LLaMA가 13B
  - PaLM 540B vs LLaMA 65B

		BoolQ	PIQA	SIQA	HellaSwag	WinoGrande	ARC-e	ARC-c	OBQA
GPT-3	175B	60.5	81.0	-	78.9	70.2	68.8	51.4	57.6
Gopher	280B	79.3	81.8	50.6	79.2	70.1	-	-	-
Chinchilla	70B	83.7	81.8	51.3	80.8	74.9	-	-	-
PaLM	62B	84.8	80.5	-	79.7	77.0	75.2	52.5	50.4
PaLM-cont	62B	83.9	81.4	-	80.6	77.0	-	-	-
PaLM	540B	**88.0**	82.3	-	83.4	**81.1**	76.6	53.0	53.4
LLaMA	7B	76.5	79.8	48.9	76.1	70.1	72.8	47.6	57.2
	13B	78.1	80.1	50.4	79.2	73.0	74.8	52.7	56.4
	33B	83.1	82.3	50.4	82.8	76.0	**80.0**	**57.8**	58.6
	65B	85.3	**82.8**	**52.3**	**84.2**	77.0	78.9	56.0	**60.2**

Table 3: **Zero-shot performance on Common Sense Reasoning tasks.**

3 Experiments

- Zero-shot & few-shot performance
  - Exact match
  - 모델이 더 작은데도 불구하고 성능이 좋다

		0-shot	1-shot	5-shot	64-shot
Gopher	280B	43.5	-	57.0	57.2
Chinchilla	70B	55.4	-	64.1	64.6
LLaMA	7B	50.0	53.4	56.3	57.6
	13B	56.6	60.5	63.1	64.0
	33B	65.1	67.9	69.9	70.4
	65B	**68.2**	**71.6**	**72.6**	**73.0**

Table 5: **TriviaQA.** Zero-shot and few-shot exact match performance on the filtered dev set.

		0-shot	1-shot	5-shot	64-shot
GPT-3	175B	14.6	23.0	-	29.9
Gopher	280B	10.1	-	24.5	28.2
Chinchilla	70B	16.6	-	31.5	35.5
PaLM	8B	8.4	10.6	-	14.6
	62B	18.1	26.5	-	27.6
	540B	21.2	29.3	-	39.6
LLaMA	7B	16.8	18.7	22.0	26.1
	13B	20.1	23.4	28.1	31.9
	33B	**24.9**	28.3	32.9	36.0
	65B	23.8	**31.0**	**35.0**	**39.9**

Table 4: **NaturalQuestions.** Exact match performance.

Experiments

- Reading comprehension
  - RACE: 중고등학교 영어 독해 이해력 평가

		RACE-middle	RACE-high
GPT-3	175B	58.4	45.5
PaLM	8B	57.9	42.3
	62B	64.3	47.5
	540B	**68.1**	49.1
LLaMA	7B	61.1	46.9
	13B	61.6	47.2
	33B	64.1	48.3
	65B	67.9	**51.6**

Table 6: **Reading Comprehension.** Zero-shot accuracy.

Experiments

- Code generation
  - 자연어 설명이 주어지면 이에 맞는 python code 생성 > test case를 만족해야 된다
  - @n: n개의 코드를 생성 > test case

	Params	HumanEval		MBPP	
pass@		@1	@100	@1	@80
LaMDA	137B	14.0	47.3	14.8	62.4
PaLM	8B	3.6*	18.7*	5.0*	35.7*
PaLM	62B	15.9	46.3*	21.4	63.2*
PaLM-cont	62B	23.7	-	31.2	-
PaLM	540B	**26.2**	76.2	36.8	75.0
LLaMA	7B	10.5	36.5	17.7	56.2
	13B	15.8	52.5	22.0	64.0
	33B	21.7	70.7	30.2	73.4
	65B	23.7	**79.3**	**37.7**	**76.8**

4 Model architecture

- LLaMA
  1. Pre-normalization with RMSNorm
  2. SwiGLU activation function
  3. Rotary embeddings

Pre-normalization [GPT3]. To improve the training stability, we normalize the input of each transformer sub-layer, instead of normalizing the output. We use the RMSNorm normalizing function, introduced by Zhang and Sennrich (2019).

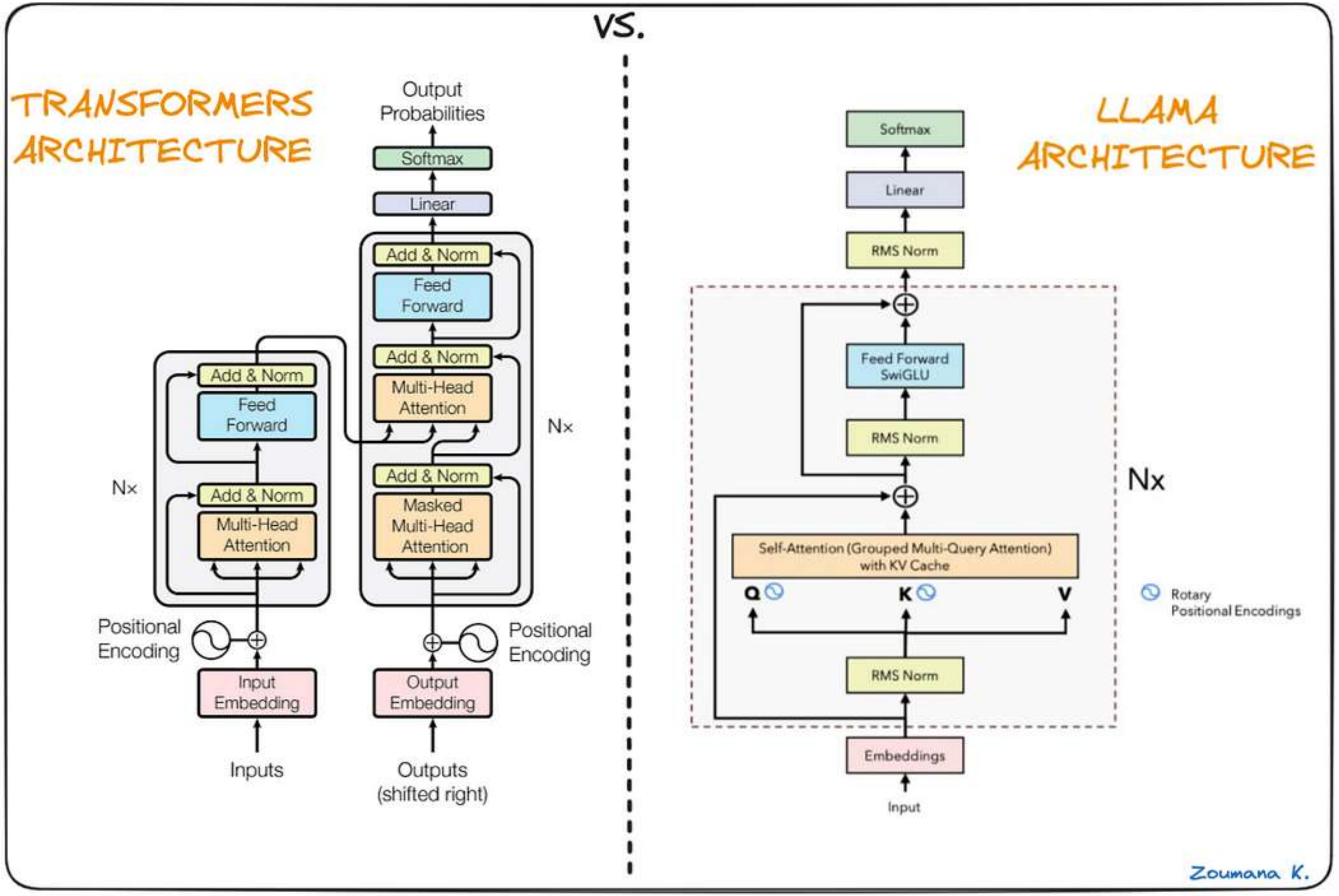
SwiGLU activation function [PaLM]. We replace the ReLU non-linearity by the SwiGLU activation function, introduced by Shazeer (2020) to improve the performance. We use a dimension of $\frac{2}{3}4d$ instead of $4d$ as in PaLM.

Rotary Embeddings [GPTNeo]. We remove the absolute positional embeddings, and instead, add rotary positional embeddings (RoPE), introduced by Su et al. (2021), at each layer of the network.

The details of the hyper-parameters for our different models are given in Table 2.

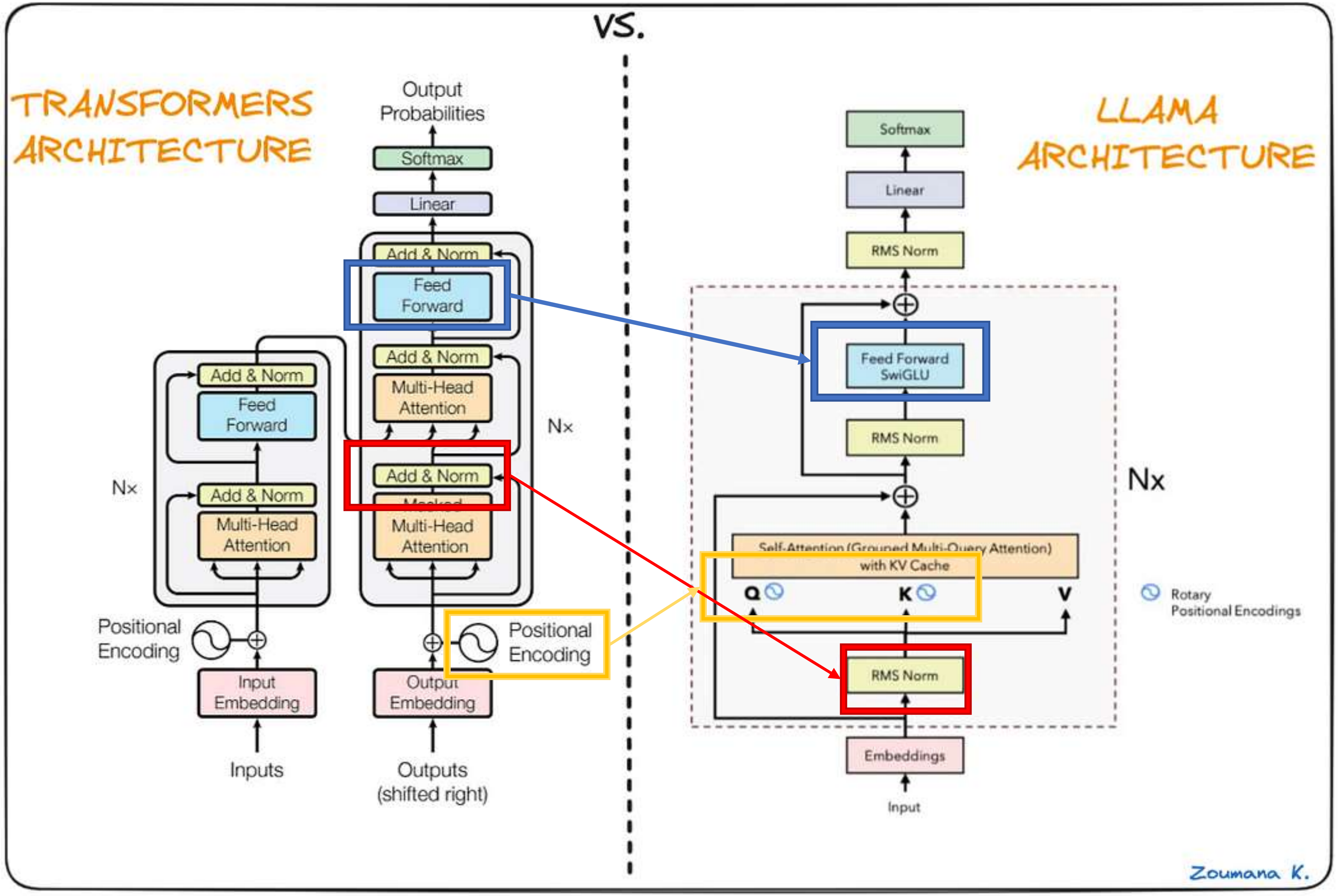
4 Model architecture

- Decoder only model



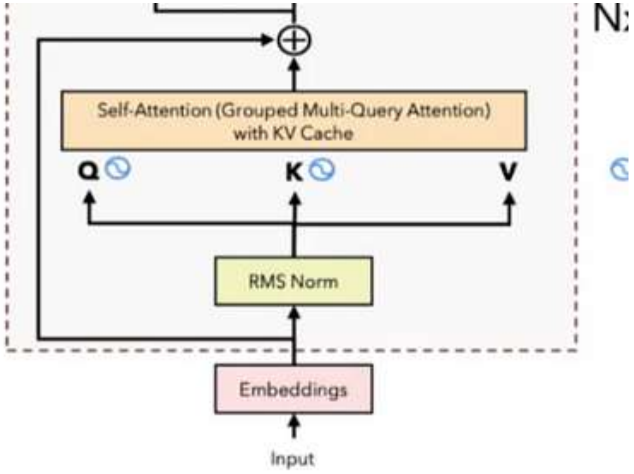
4 Model architecture

- Decoder only model



4 Model architecture

- Pre-normalization with RMSNorm



2018

Training Tips for
the Transformer Model

On Layer Normalization in the
Transformer Architecture

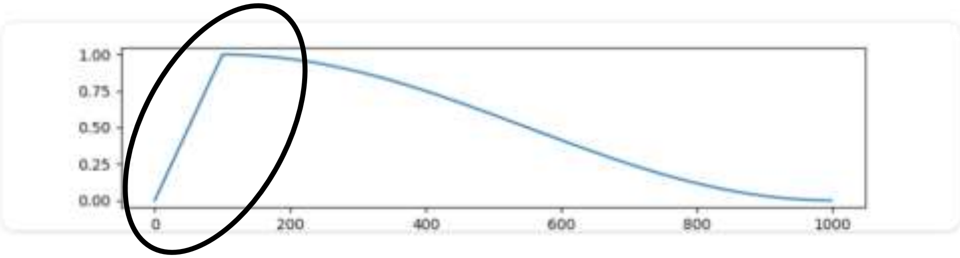
2020

2019

RMS LayerNorm

4 Model architecture

- Training Tips for the Transformer Model (2018)
  - 모델을 학습시킬 때, learning rate warmup 과정이 필요하다
  - Transformer를 학습 (learning rate, batch size 고정)
  - Warmup step이 모델의 최종 성능에 영향을 끼친다



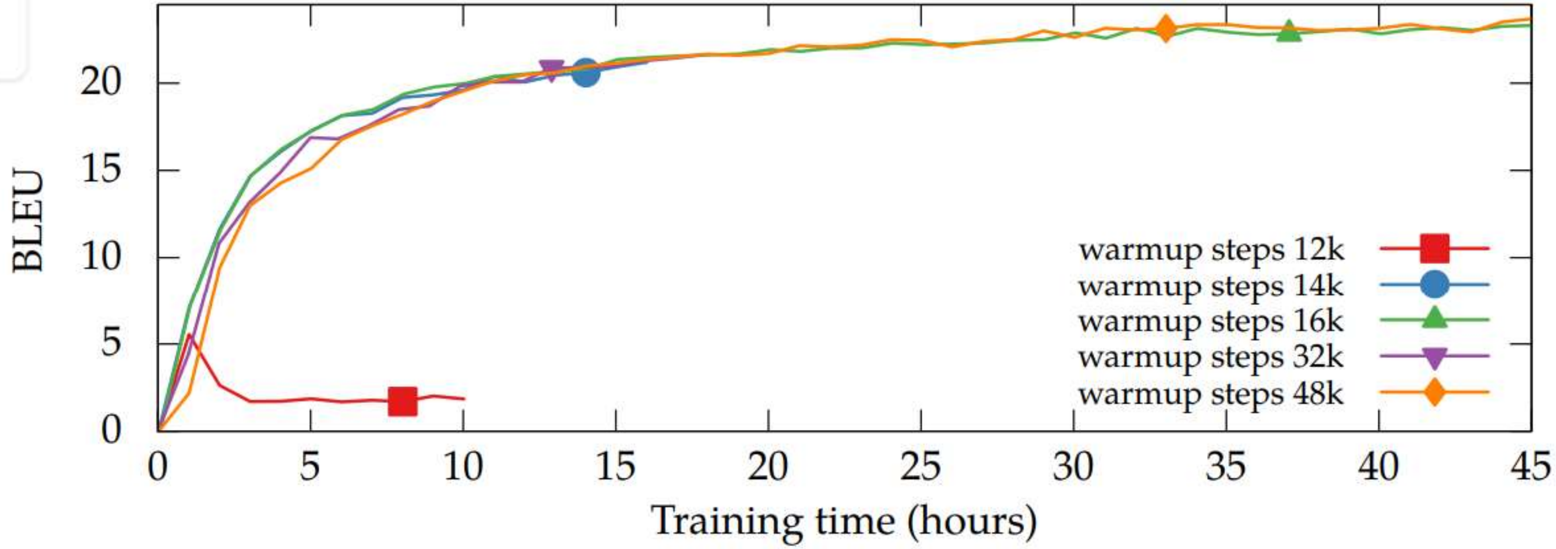
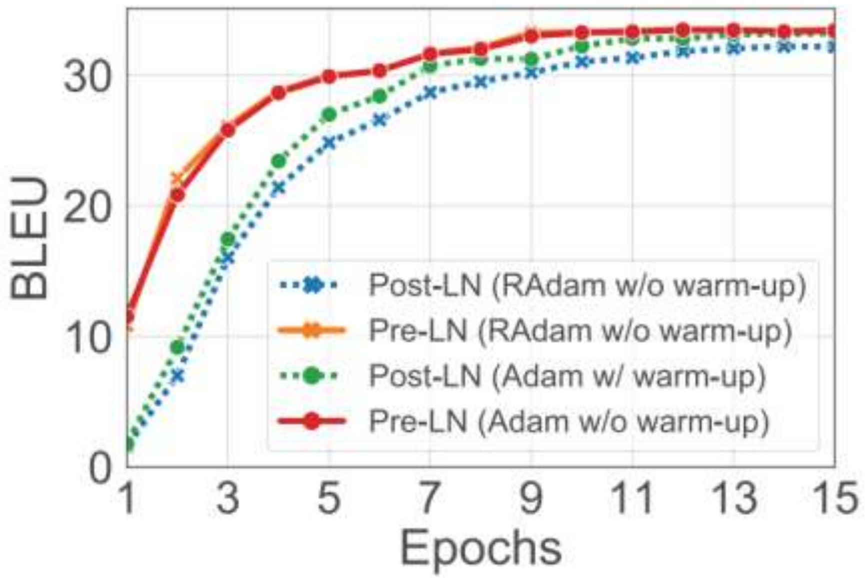


Figure 8: Effect of the warmup steps on a single GPU. All trained on CzEng 1.0 with the default batch size (1500) and learning rate (0.20).

4 Model architecture

- On Layer Normalization in the Transformer Architecture (2020)
  - Warmup step
    - ✓ 최적화 과정을 느리게 만든다
    - ✓ 학습이 불안정해질 수 있다
  - Post는 warmup이 있는 게 좋다; Pre는 상관없다



(b) BLEU (IWSLT)

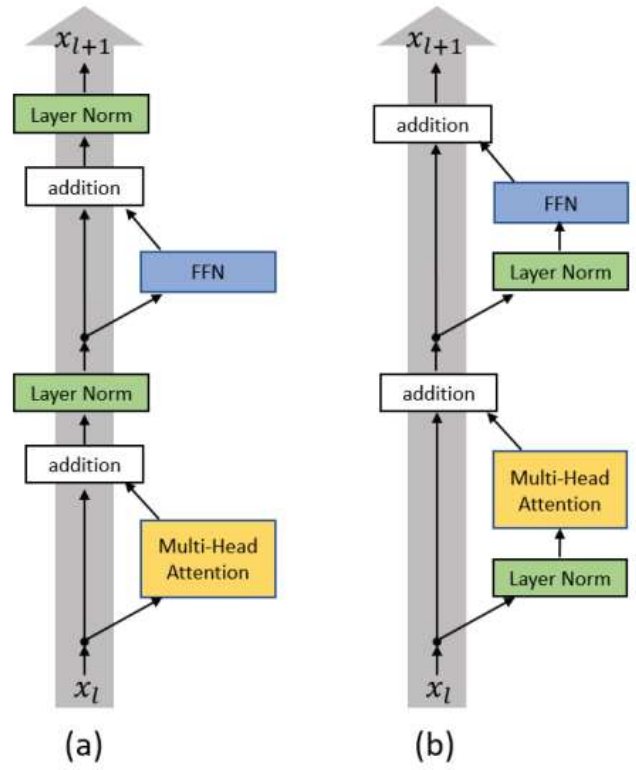
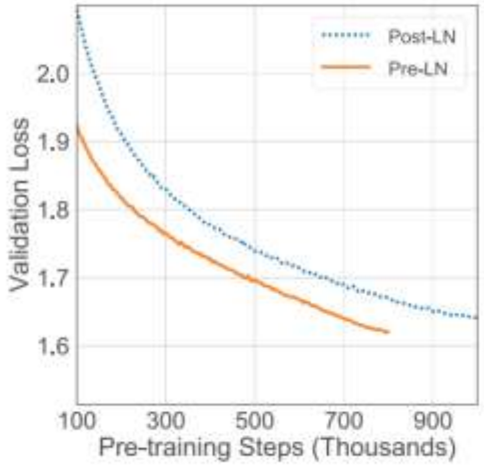


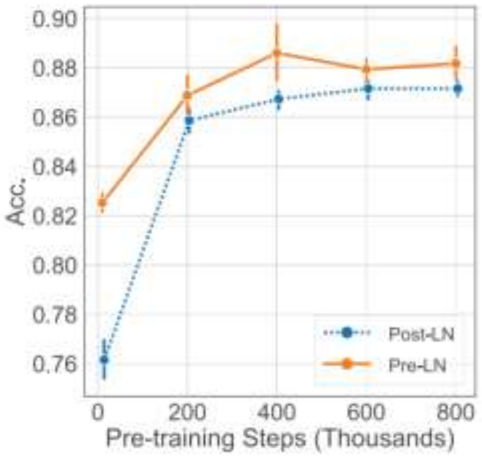
Figure 1. (a) Post-LN Transformer layer; (b) Pre-LN Transformer layer.

4 Model architecture

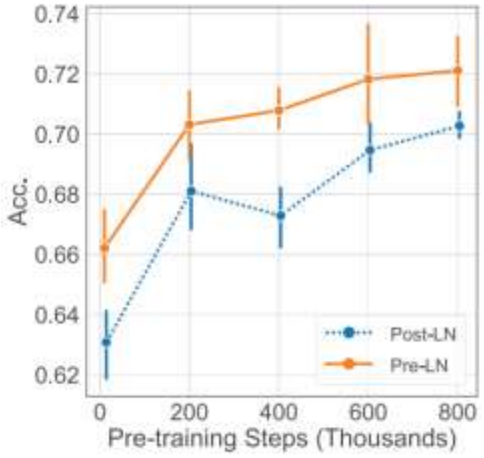
- On Layer Normalization in the Transformer Architecture (2020)
  - LayerNorm을 layer 통과 전에 배치하여 warmup step을 제거한다
  - LayerNorm: gradient 제어에 결정적인 역할을 제공 > 안정적인 학습



(a) Validation Loss on BERT



(b) Accuracy on MRPC



(c) Accuracy on RTE

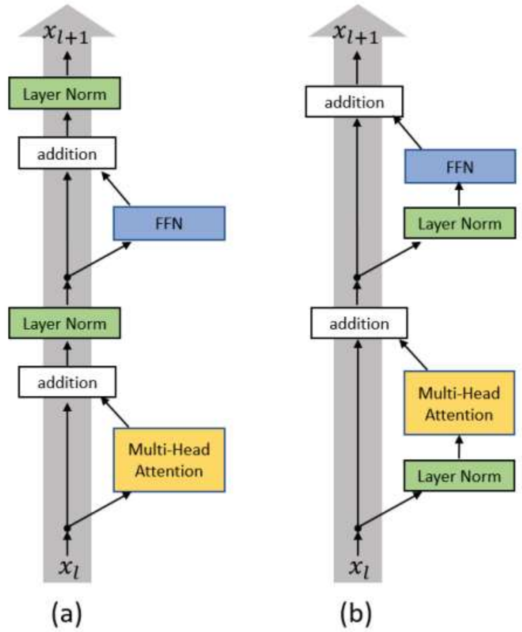
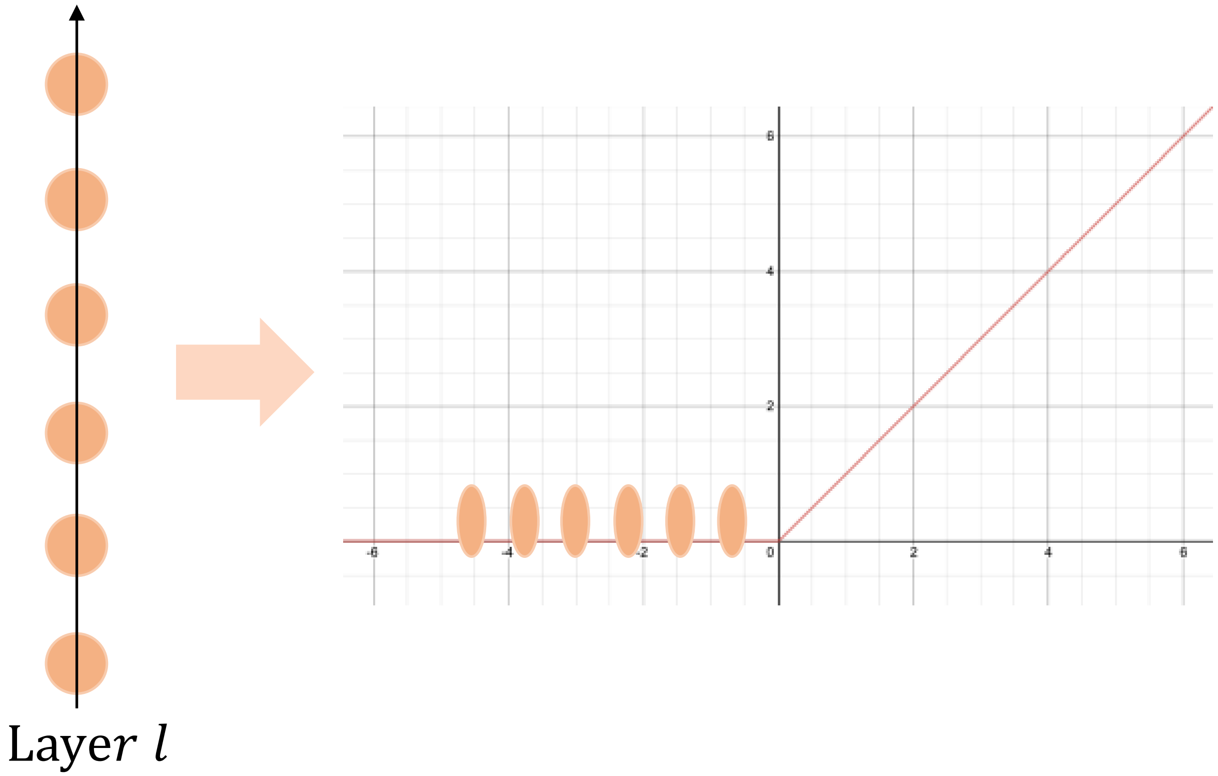


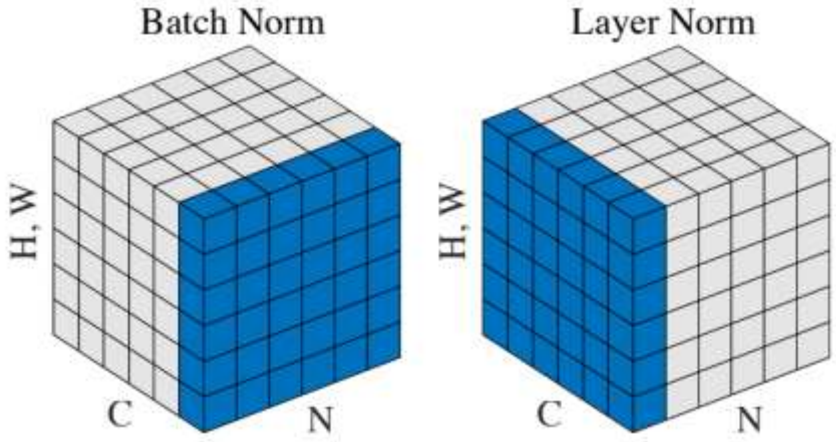
Figure 1. (a) Post-LN Transformer layer; (b) Pre-LN Transformer layer.

4 Model architecture

- LayerNorm (2016)
  - RNN이나 Transformer에 유용하게 사용된다 (vs BatchNorm)
  - 급격한 값의 변동을 줄이면서, vanishing/exploding gradient 문제를 완화한다

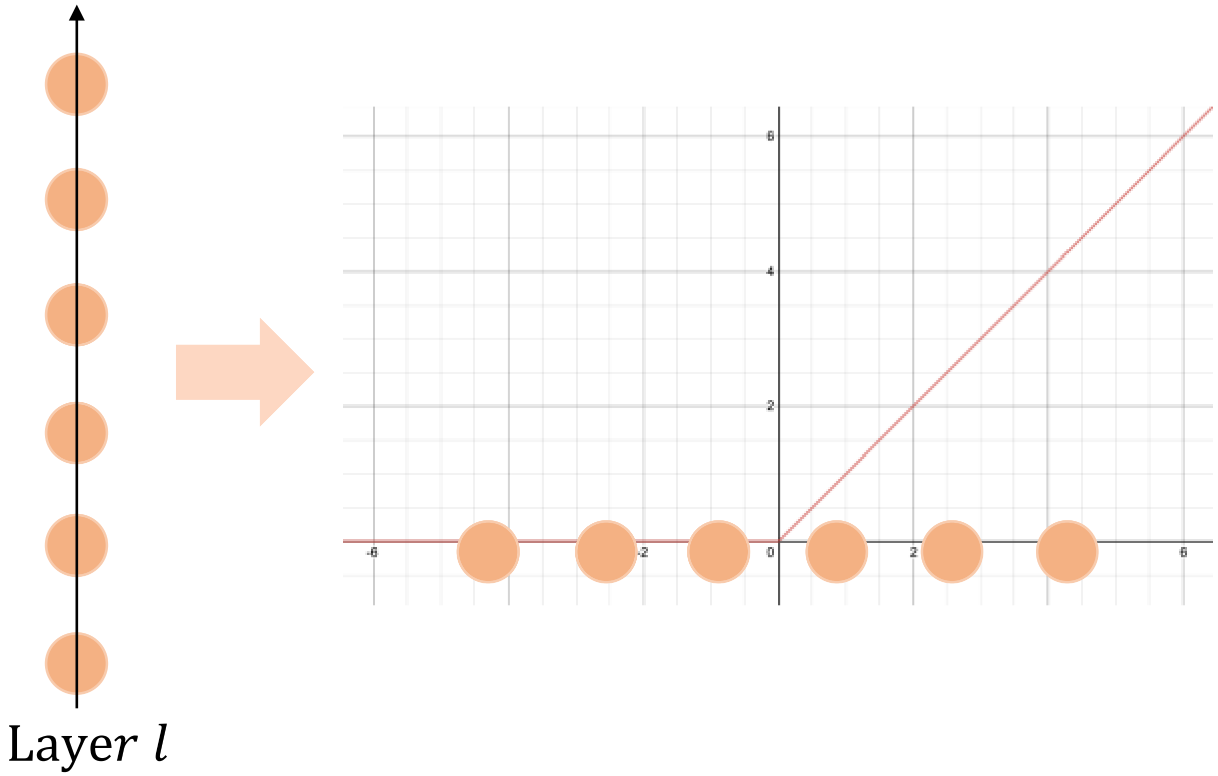


$$y = \frac{x - \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x] + \epsilon}} * \gamma + \beta$$

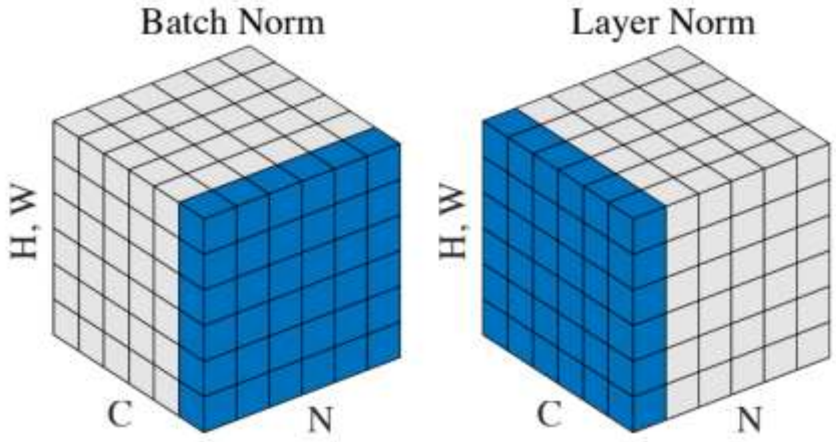


4 Model architecture

- LayerNorm (2016)
  - RNN이나 Transformer에 유용하게 사용된다 (vs BatchNorm)
  - 급격한 값의 변동을 줄이면서, vanishing/exploding gradient 문제를 완화한다



$$y = \frac{x - \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x] + \epsilon}} * \gamma + \beta$$



4 Model architecture

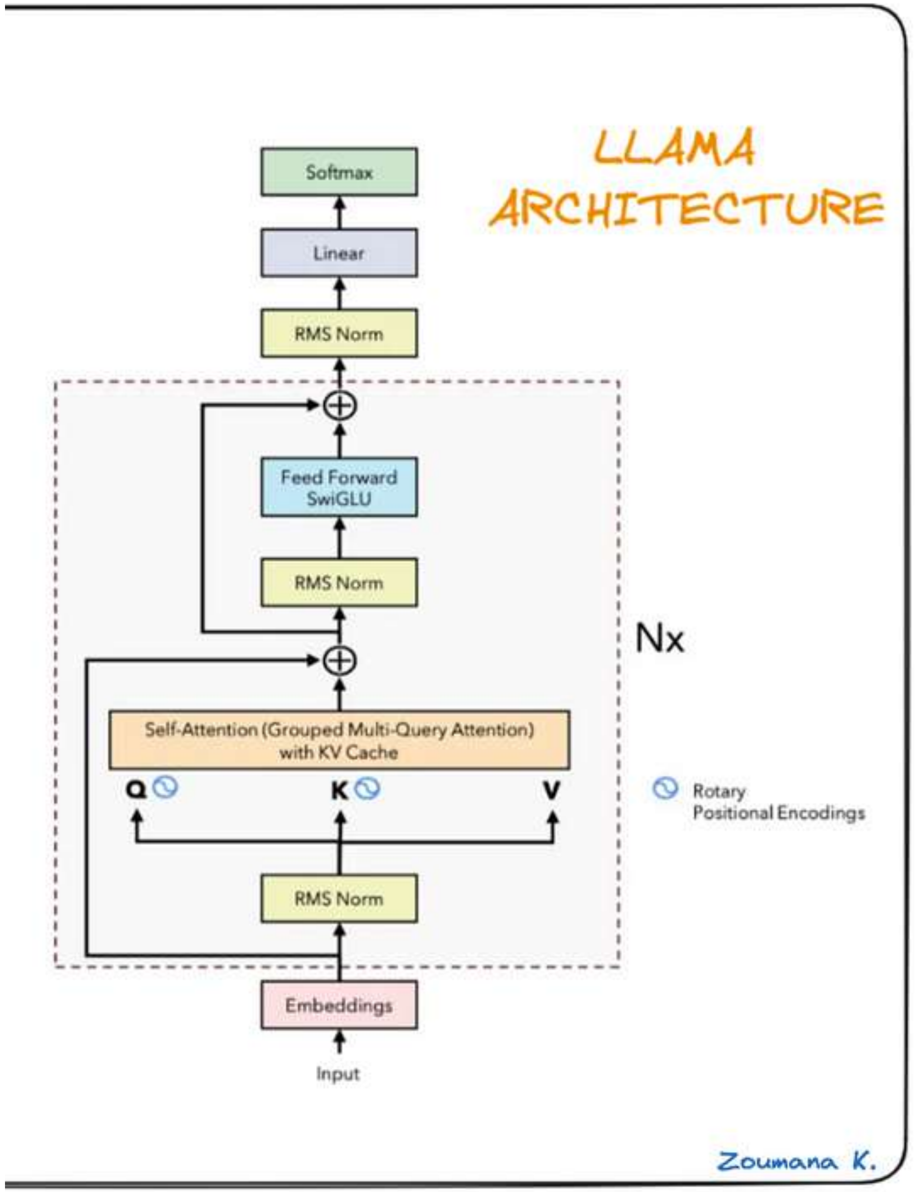
- RMSLayerNorm (2019)
  - Pre-LN에서 LayerNorm 대신 RMSLayerNorm을 사용
  - LayerNorm의 특성은 re-centering & re-scaling invariance이다
    - ✓ Re-centering: input이 shift되어도 평균을 0으로 맞춘다
    - ✓ Re-scaling: input의 크기가 변해도 표준편차로 나누기에, 크기 변화에 민감하지 않다
  - 하지만 본 논문에서는 re-scaling invariance의 효과가 더 크다고 가정하고 re-centering은 삭제

$$y = \frac{x - \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x] + \epsilon}} * \gamma + \beta$$

$$\mathbf{y} = \gamma \cdot \frac{\mathbf{x}}{\sqrt{\frac{1}{d}\sum_{i=1}^{d} x_i^2 + \epsilon}}$$

Model architecture

- Pre-normalization with RMSNorm



4 Model architecture

- SwiGLU activation function (2020)
  - Swish + GLU
  - Swish: 입력의 정보량 조절
  - GLU: 입력의 정보량 조절

$$SwiGLU(x, W, V, b, c, \beta) = Swish_\beta(xW + b) \otimes (xV + c)$$

$$Swish(x) = x\sigma(\beta x)$$

$$GLU(x, W, V, b, c) = \sigma(xW + b) \otimes (xV + c)$$

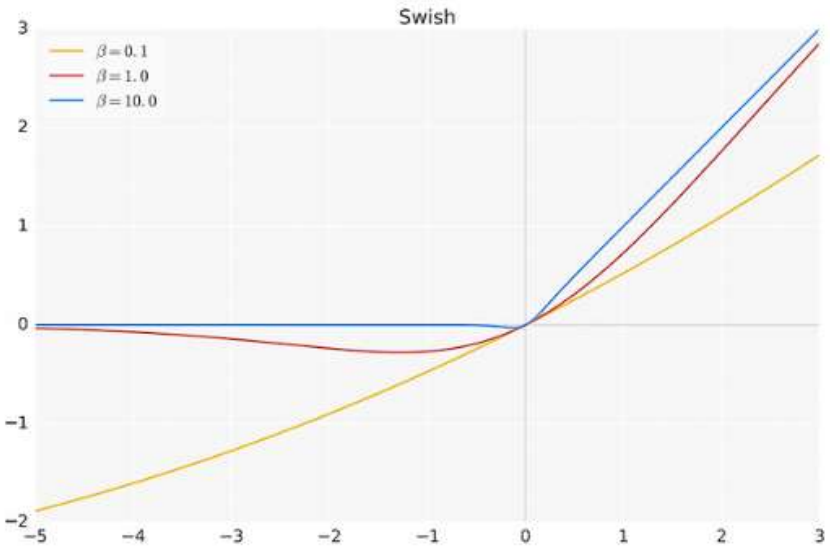


Figure 4: The Swish activation function.

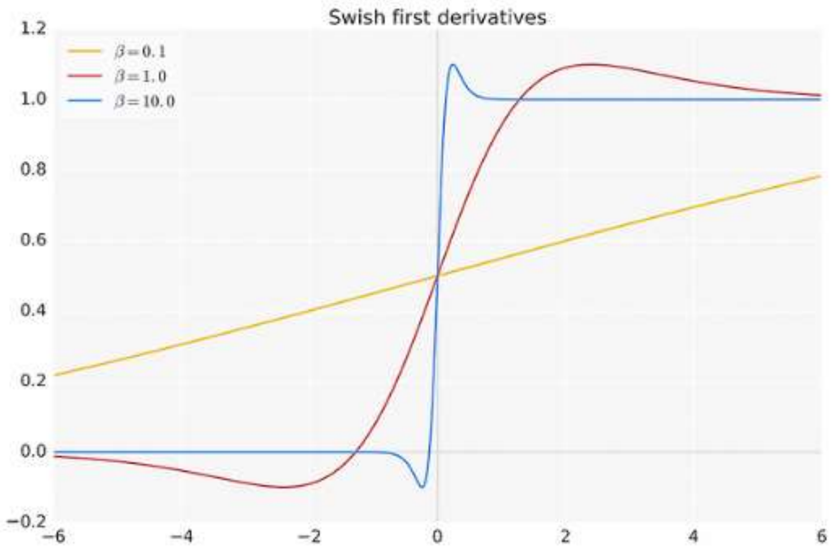


Figure 5: First derivatives of Swish.

4 Model architecture

- SwiGLU activation function (2020)
  - SwiGLU의 성공에 대한 설명을 제공하지는 않음

4 Conclusions

We have extended the GLU family of layers and proposed their use in Transformer. In a transfer-learning setup, the new variants seem to produce better perplexities for the de-noising objective used in pre-training, as well as better results on many downstream language-understanding tasks. These architectures are simple to implement, and have no apparent computational drawbacks. We offer no explanation as to why these architectures seem to work; we attribute their success, as all else, to divine benevolence.

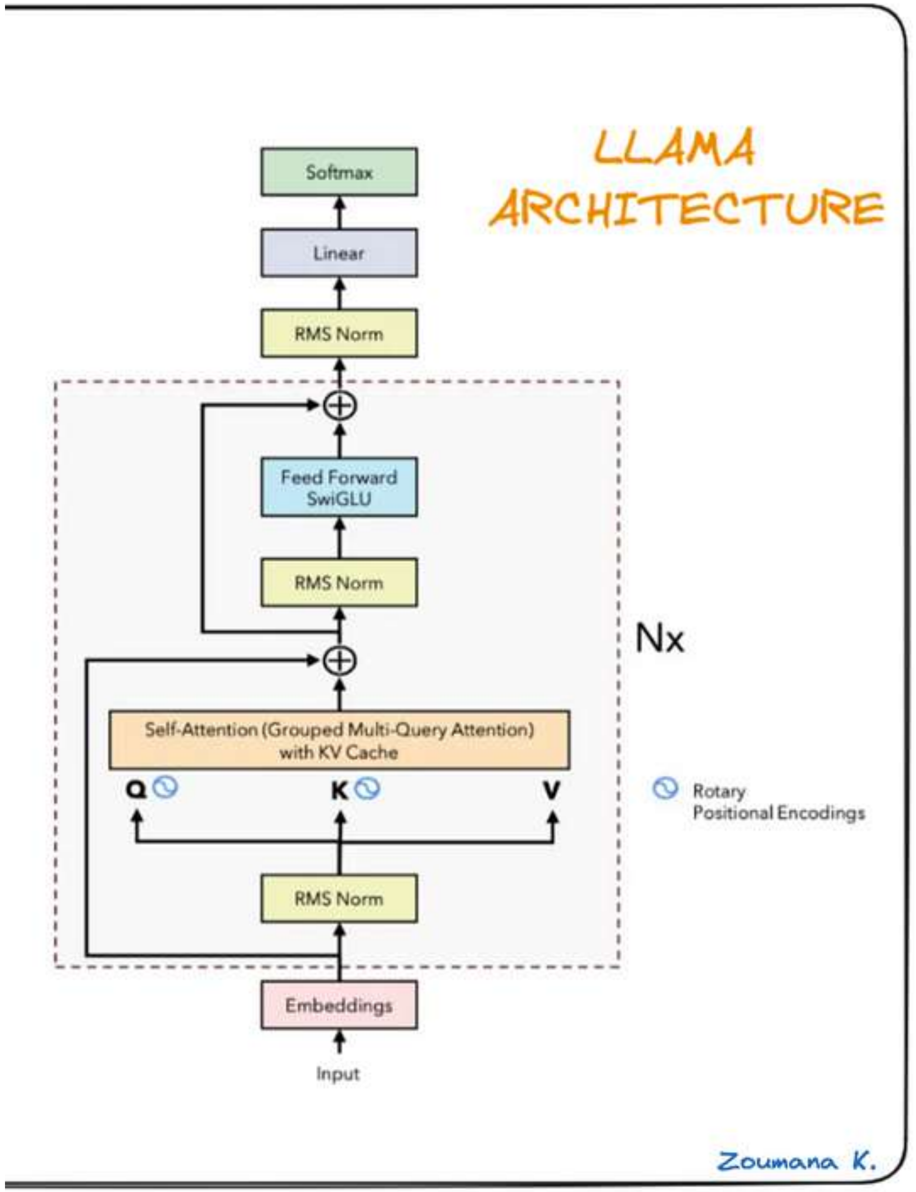
Model architecture

- SwiGLU activation function (2020)
  - FFN에 사용

	Score Average	CoLA MCC	SST-2 Acc	MRPC F1	MRPC Acc	STSB PCC	STSB SCC	QQP F1	QQP Acc	MNLIm Acc	MNLImm Acc	QNLI Acc	RTE Acc
FFN_{ReLU}	83.80	51.32	94.04	**93.08**	**90.20**	89.64	89.42	89.01	91.75	85.83	86.42	92.81	80.14
FFN_{GELU}	83.86	53.48	94.04	92.81	**90.20**	89.69	89.49	88.63	91.62	85.89	86.13	92.39	80.51
FFN_{Swish}	83.60	49.79	93.69	92.31	89.46	89.20	88.98	88.84	91.67	85.22	85.02	92.33	81.23
FFN_{GLU}	84.20	49.16	94.27	92.39	89.46	89.46	89.35	88.79	91.62	86.36	86.18	92.92	**84.12**
FFN_{GEGLU}	84.12	53.65	93.92	92.68	89.71	90.26	90.13	89.11	91.85	86.15	86.17	92.81	79.42
$FFN_{Bilinear}$	83.79	51.02	**94.38**	92.28	89.46	90.06	89.84	88.95	91.69	**86.90**	**87.08**	92.92	81.95
FFN_{SwiGLU}	84.36	51.59	93.92	92.23	88.97	**90.32**	**90.13**	**89.14**	**91.87**	86.45	86.47	**92.93**	83.39
FFN_{ReGLU}	**84.67**	**56.16**	**94.38**	92.06	89.22	89.97	89.85	88.86	91.72	86.20	86.40	92.68	81.59
[Raffel et al., 2019]	83.28	53.84	92.68	92.07	88.92	88.02	87.94	88.67	91.56	84.24	84.57	90.48	76.28
ibid. stddev.	0.235	1.111	0.569	0.729	1.019	0.374	0.418	0.108	0.070	0.291	0.231	0.361	1.393

4 Model architecture

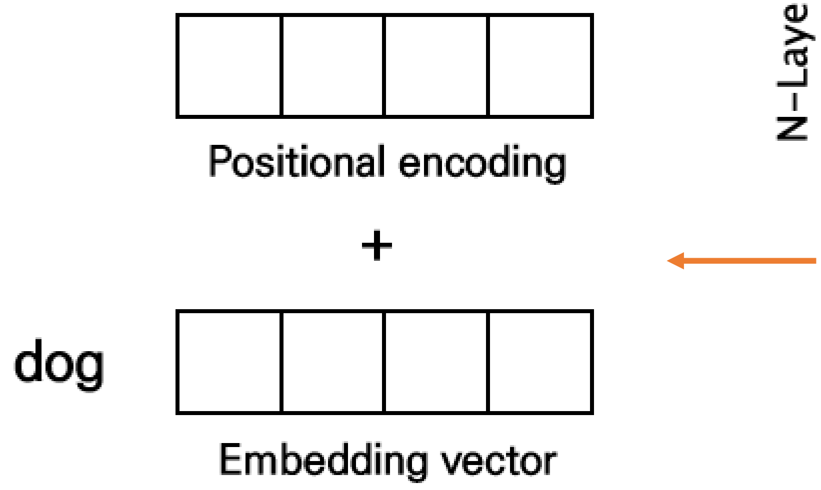
- SwiGLU activation function

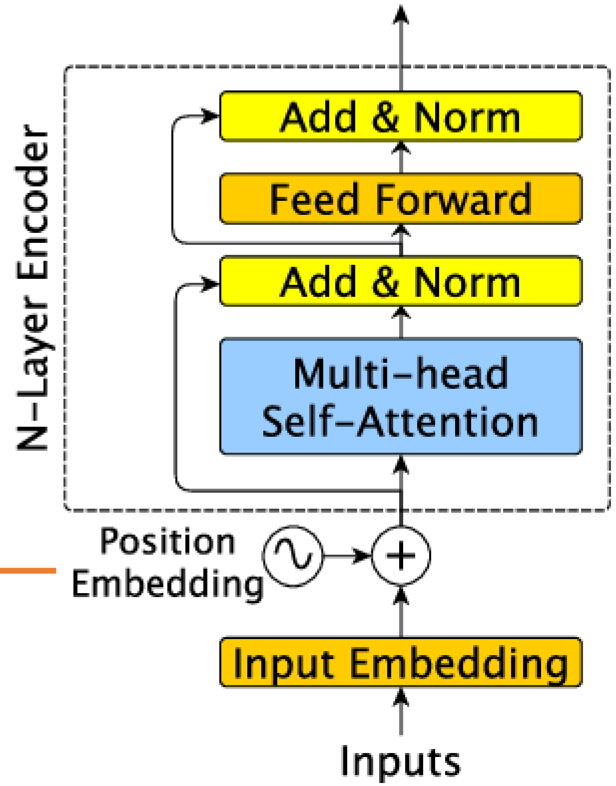


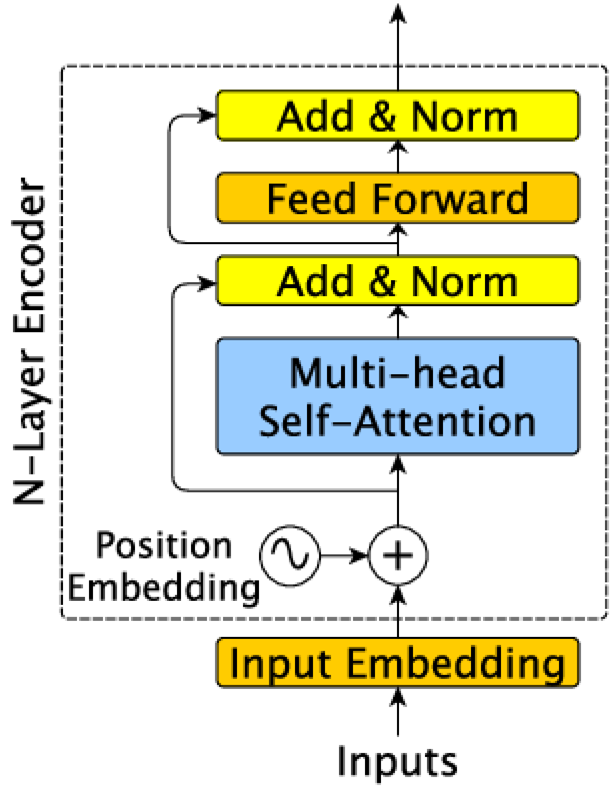
4 Model architecture

- Rotary positional embeddings
  - Absolute vs Relative positional embeddings
  - Absolute: "Attention is all you need"
  - Relative: "Self-Attention with Relative Position Representations (2018)"

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$







4 Model architecture

- Relative: “Self-Attention with Relative Position Representations”
  - Self-attention에 input element간 거리 개념 a_{ij}을 추가

QK^T:

$$e_{ij} = \frac{(x_i W^Q)(x_j W^K)^T}{\sqrt{d_z}}$$

$$e_{ij} = \frac{x_i W^Q (x_j W^K + a_{ij}^K)^T}{\sqrt{d_z}}$$

Softmax:

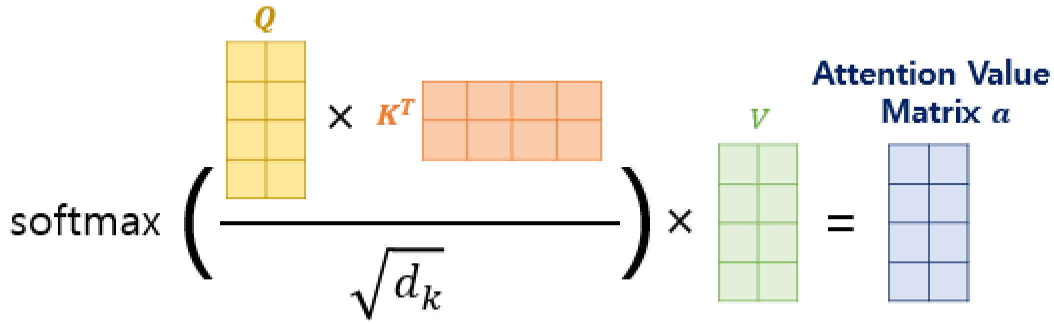
$$\alpha_{ij} = \frac{\exp e_{ij}}{\sum_{k=1}^{n} \exp e_{ik}}$$

$$\alpha_{ij} = \frac{\exp e_{ij}}{\sum_{k=1}^{n} \exp e_{ik}}$$

$Attention\ value$:

$$z_i = \sum_{j=1}^{n} \alpha_{ij}(x_j W^V)$$

$$z_i = \sum_{j=1}^{n} \alpha_{ij}(x_j W^V + a_{ij}^V)$$



4 Model architecture

- Relative: “Self-Attention with Relative Position Representations”
  - Self-attention에 input element간 거리 개념 a_{ij}을 추가

QK^T:

$$e_{ij} = \frac{(x_i W^Q)(x_j W^K)^T}{\sqrt{d_z}}$$

$$e_{ij} = \frac{x_i W^Q (x_j W^K + a_{ij}^K)^T}{\sqrt{d_z}}$$

Softmax:

$$\alpha_{ij} = \frac{\exp e_{ij}}{\sum_{k=1}^{n} \exp e_{ik}}$$

$$\alpha_{ij} = \frac{\exp e_{ij}}{\sum_{k=1}^{n} \exp e_{ik}}$$

$Attention\ value$:

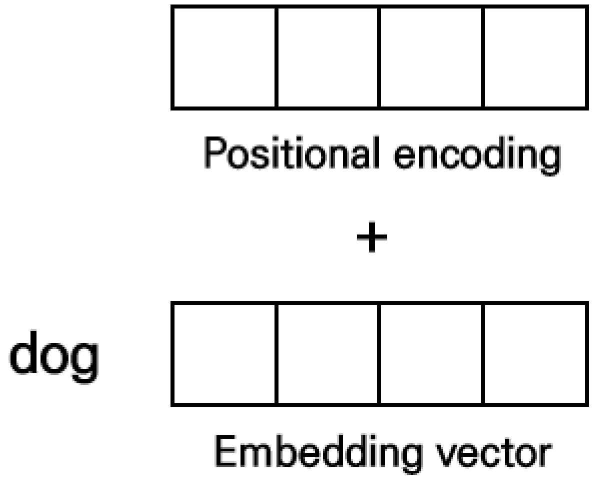
$$z_i = \sum_{j=1}^{n} \alpha_{ij}(x_j W^V)$$

$$z_i = \sum_{j=1}^{n} \alpha_{ij}(x_j W^V + a_{ij}^V)$$

Model	Position Information	EN-DE BLEU	EN-FR BLEU
Transformer (base)	Absolute Position Representations	26.5	38.2
Transformer (base)	Relative Position Representations	**26.8**	**38.7**
Transformer (big)	Absolute Position Representations	27.9	41.2
Transformer (big)	Relative Position Representations	**29.2**	**41.5**

4 Model architecture

- RoFormer: Rotary Position Embedding (2023)
  - Relative positional embeddings
  - 지금까지의 방법들은 모두 additive property 를 가진다
  - $||a|| \cdot ||b|| \cos(\theta)$: 덧셈 보단 벡터의 방향을 변화시켜서 상대적 차이에 대한 정보 제공



$$e_{ij} = \frac{x_i W^Q (x_j W^K + a_{ij}^K)^T}{\sqrt{d_z}}$$

$$\alpha_{ij} = \frac{\exp e_{ij}}{\sum_{k=1}^{n} \exp e_{ik}}$$

$$z_i = \sum_{j=1}^{n} \alpha_{ij} (x_j W^V + a_{ij}^V)$$

4 Model architecture

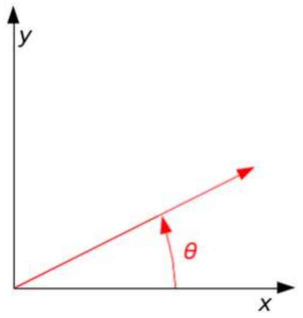
- RoFormer: Rotary Position Embedding (2023)
  - Vector의 dimension이 2라고 가정
  - Attention에서 상대적 위치가 중요한 연산은 query, key의 dot-product이다
  - Query와 key를 회전시킨다

$$x_m \quad \begin{bmatrix} x_m^{(1)} \\ x_m^{(2)} \end{bmatrix}$$

m: m번째 단어

$$\begin{pmatrix} \cos m\theta & -\sin m\theta \\ \sin m\theta & \cos m\theta \end{pmatrix} \qquad \underbrace{W_{\{q,k\}} \qquad x_m \begin{bmatrix} x_m^{(1)} \\ x_m^{(2)} \end{bmatrix}}_{Query\ or\ Key}$$

2×2 $\qquad 2 \times 2$ $\qquad 2 \times 1$



4 Model architecture

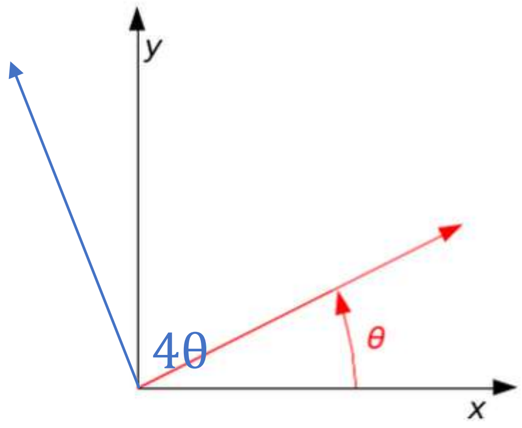
- RoFormer: Rotary Position Embedding (2023)
  - 단어간 상대적 위치가 멀수록 사이각은 멀어진다

$$\begin{pmatrix} \cos m\theta & -\sin m\theta \\ \sin m\theta & \cos m\theta \end{pmatrix}$$

2×2

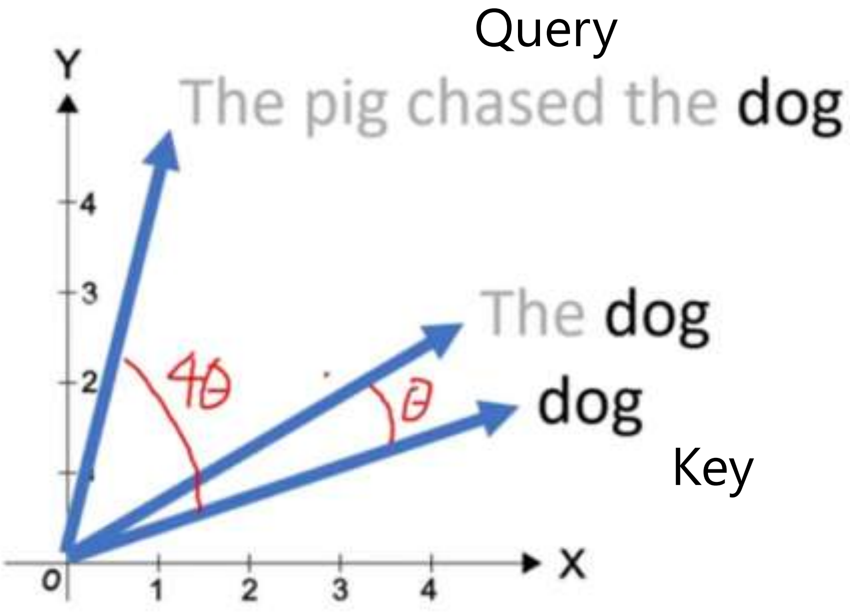
Query/Key

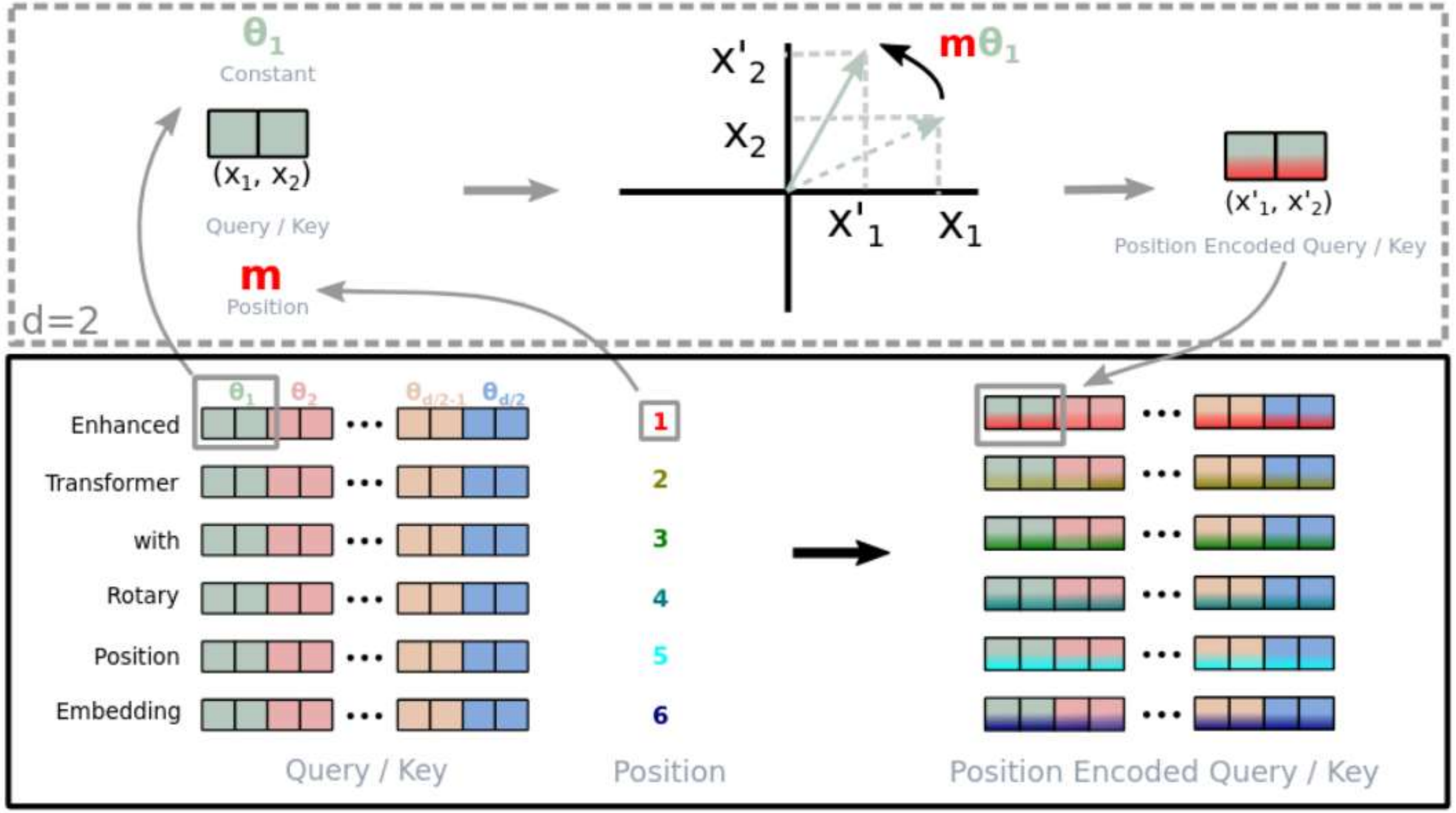
2×1



Model architecture

- RoFormer: Rotary Position Embedding (2023)
  - 같은 단어의 embedding은 위치 정보가 추가되기 전까진 동일하다
  - 상대적 위치가 멀수록 두 벡터의 사이각은 커진다





4 Model architecture

- RoFormer: Rotary Position Embedding (2023)
  - 상대적 위치가 멀수록 두 벡터의 사이각은 커진다
  - Dim=2를 d로 확장: 이전과 마찬가지로 2개씩 끊어서 계산한다

$$\{\theta_i = 10000^{-2(i-1)/d}, i \in [1, 2, ..., d/2]\}$$

For x_1, x_2

$$\begin{pmatrix} \cos m\theta_1 & -\sin m\theta_1 & 0 & 0 & \cdots & 0 & 0 \\ \sin m\theta_1 & \cos m\theta_1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \cos m\theta_2 & -\sin m\theta_2 & \cdots & 0 & 0 \\ 0 & 0 & \sin m\theta_2 & \cos m\theta_2 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & \cos m\theta_{d/2} & -\sin m\theta_{d/2} \\ 0 & 0 & 0 & 0 & \cdots & \sin m\theta_{d/2} & \cos m\theta_{d/2} \end{pmatrix}$$

For x_3, x_4

For x_{d-1}, x_d

4 Model architecture

- RoFormer: Rotary Position Embedding (2023)
  - 굳이 2차원 벡터로만 rotation시켜야 하는가?
  - 3차원씩 끊어서 계산한 논문: 3D-RPE (2024)

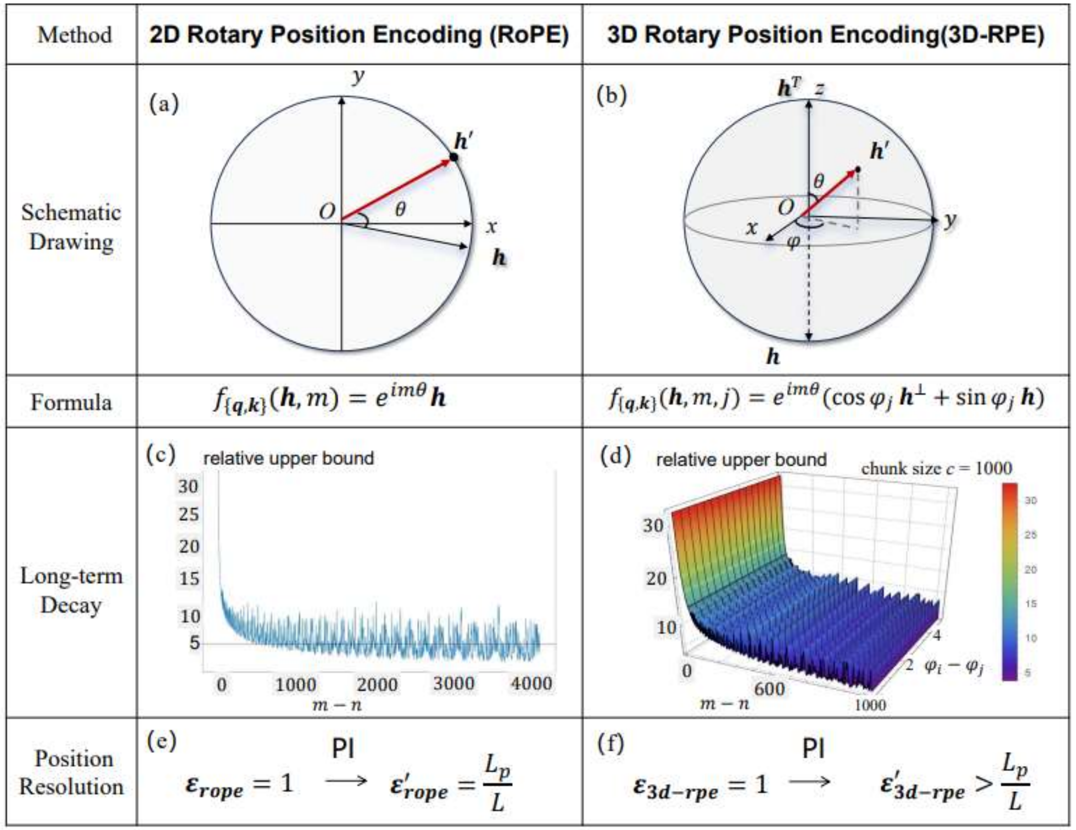


Figure 1: 2D Rotary Position Encoding (RoPE) vs. 3D Rotary Position Encoding (3D-RPE).