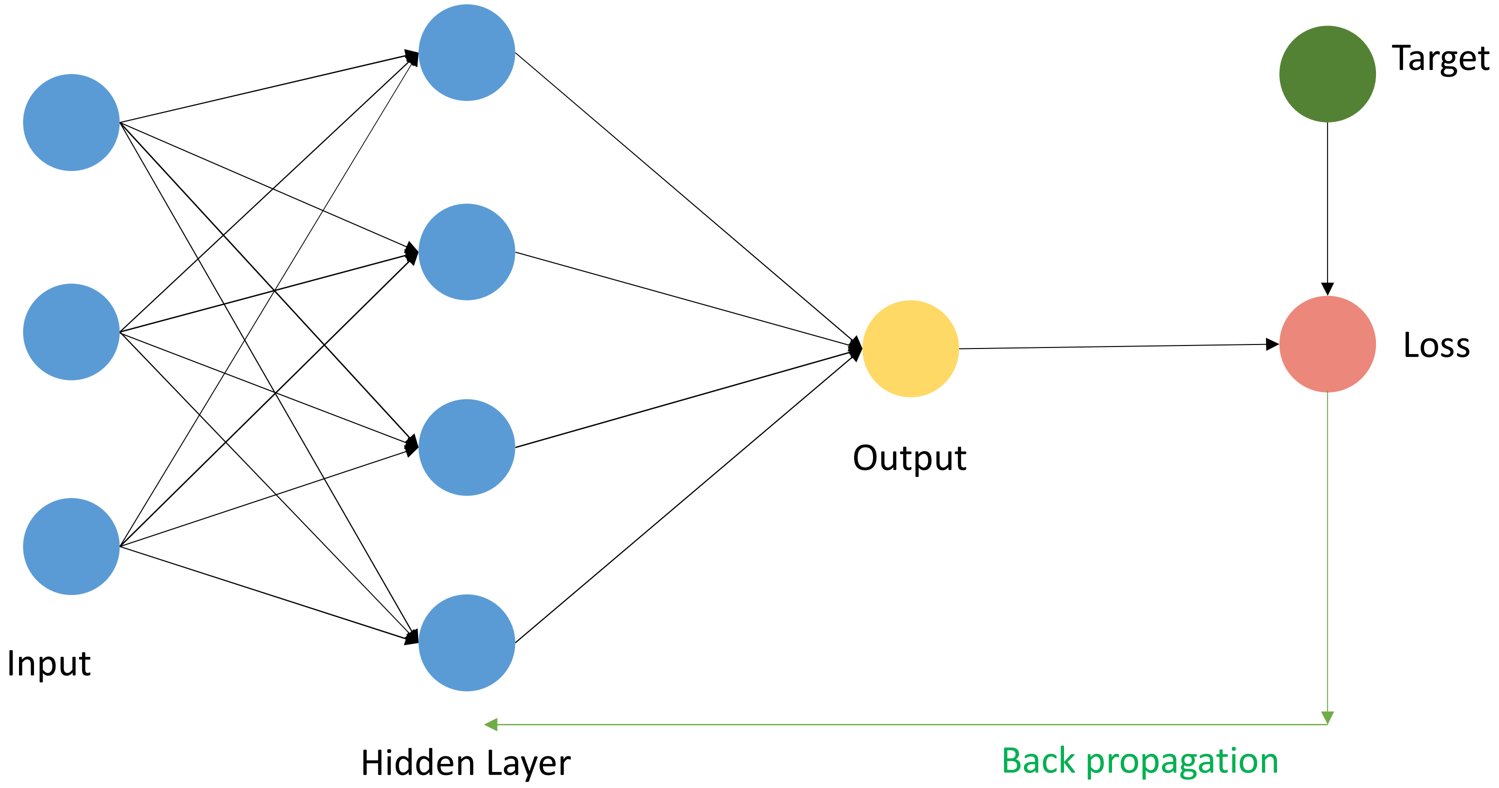
Inside ChatGPT: How Large Language Models Really Work

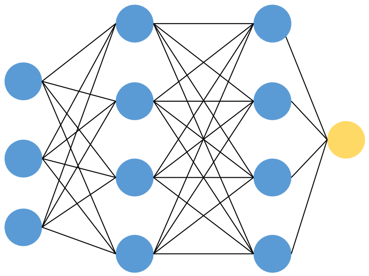
By:

Aman Khokhar

Neural Networks



Neural Networks!

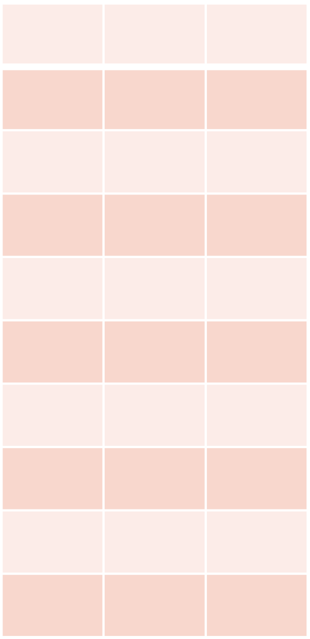


$$O = a(XW^T + b)$$

a is the activation function to introduce non-linearity

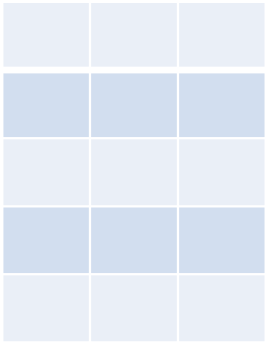
X =

(10,3)



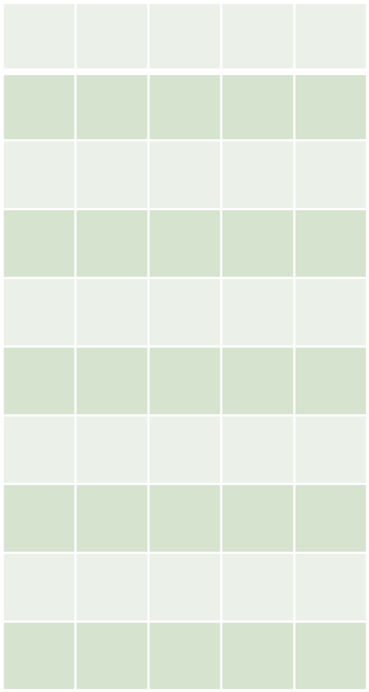
W =

(5,3)

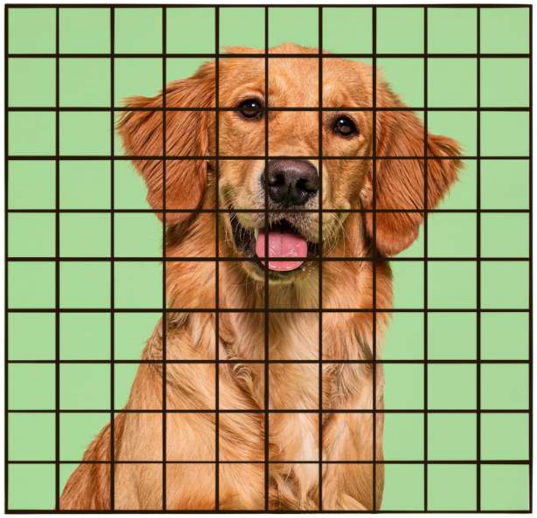


O =

(10,5)

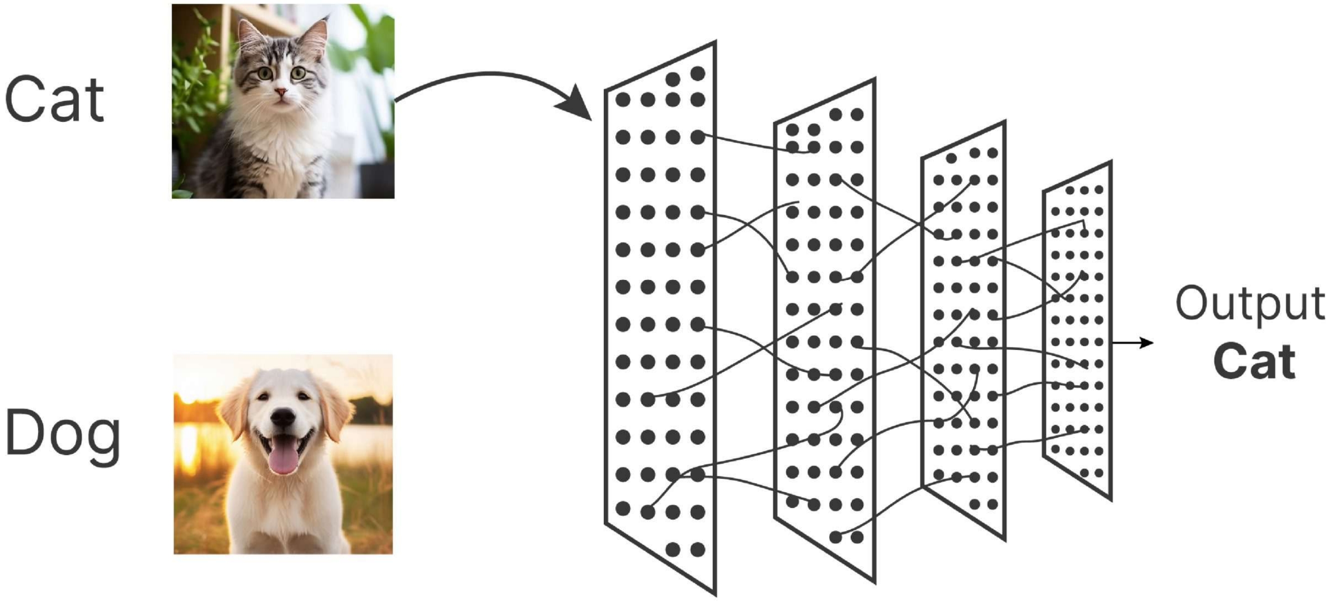


Matrix representation of Image



2	2	2	2	2	2	2	2	2	2
2	2	30	65	202	211	223	62	2	2
2	2	15	31	254	200	211	54	7	2
2	2	60	198	211	224	250	218	60	2
2	2	74	203	217	180	192	200	2	2
2	2	2	201	195	150	231	216	2	2
2	2	50	199	194	234	215	75	2	2
2	2	102	190	189	178	175	45	2	2
2	60	147	180	174	186	195	4	2	2
2	132	201	195	214	211	185	32	2	2

Why do we need transformers?

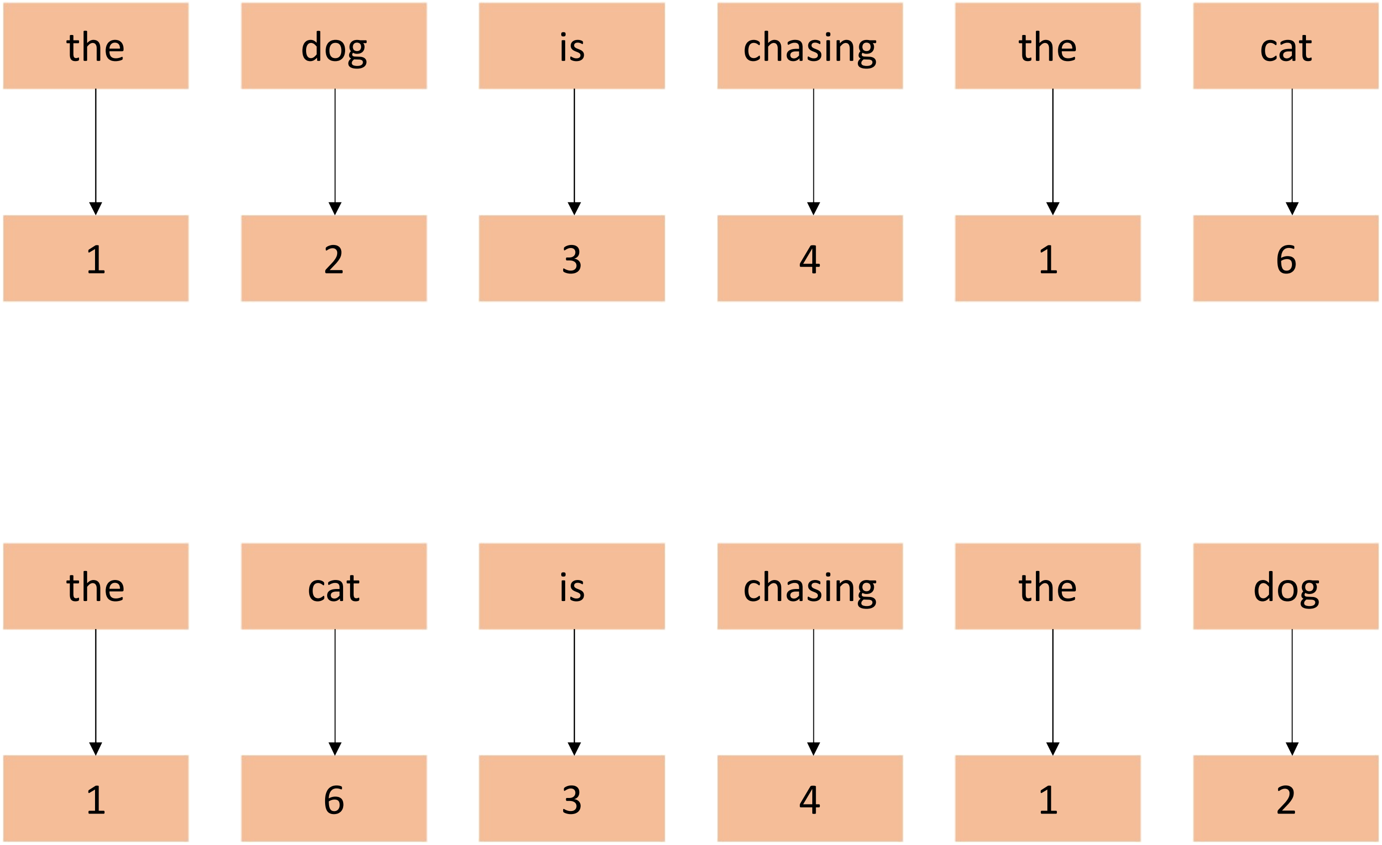


Cat

Dog

Output

Cat



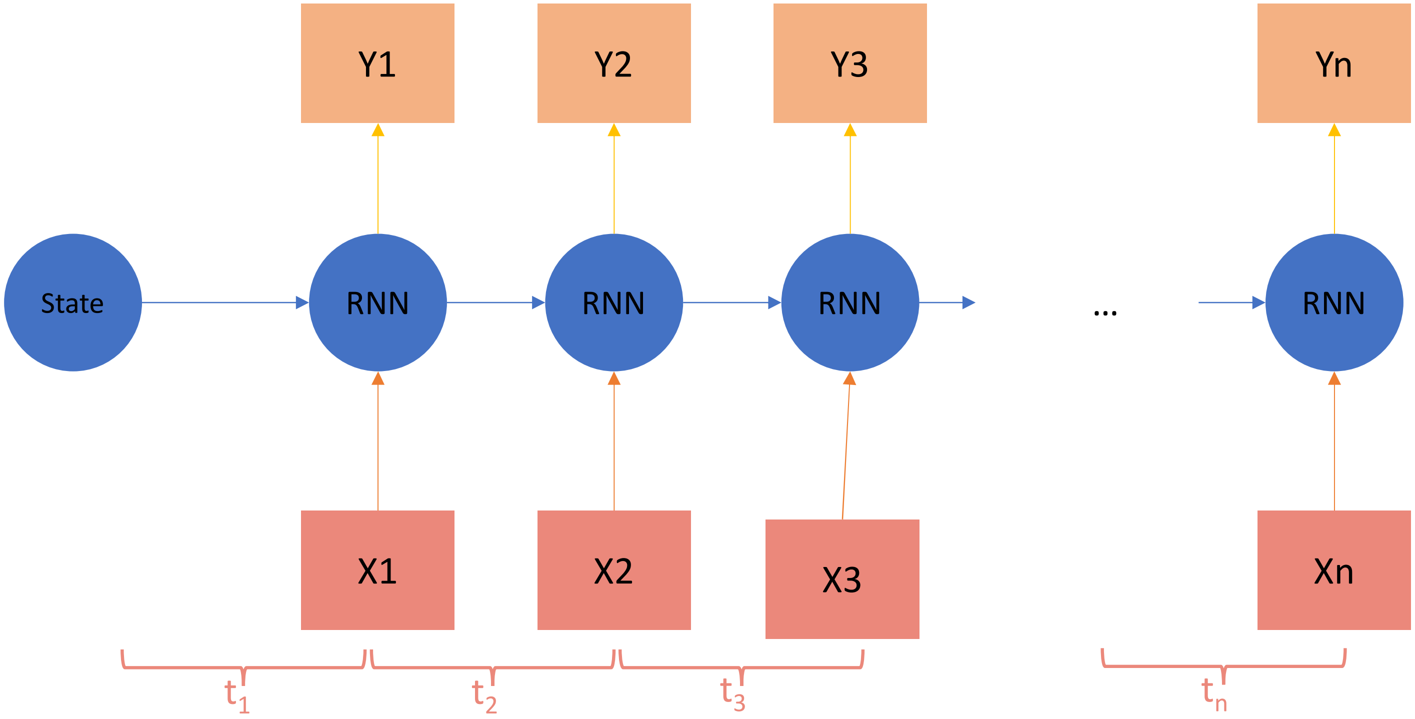
the dog is chasing the cat

1 2 3 4 1 6

the cat is chasing the dog

1 6 3 4 1 2

Recurrent Neural Networks



Problems with RNNs

- Slow computation for very long sequences
- Problems of vanishing and exploding gradients
- Assume we're computing the gradient of a loss L at time step 3 with respect to the hidden state at time step 0:

$$\frac{\partial L}{\partial h_0} = \frac{\partial h_3}{\partial h_2} . \frac{\partial h_2}{\partial h_1} . \frac{\partial h_1}{\partial h_0} \cdot \frac{\partial L}{\partial h_3}$$

- Difficulty accessing information from long time ago

Attention Is All You Need

Ashish Vaswani[*]
Google Brain
avaswani@google.com

Noam Shazeer[*]
Google Brain
noam@google.com

Niki Parmar[*]
Google Research
nikip@google.com

Jakob Uszkoreit[*]
Google Research
usz@google.com

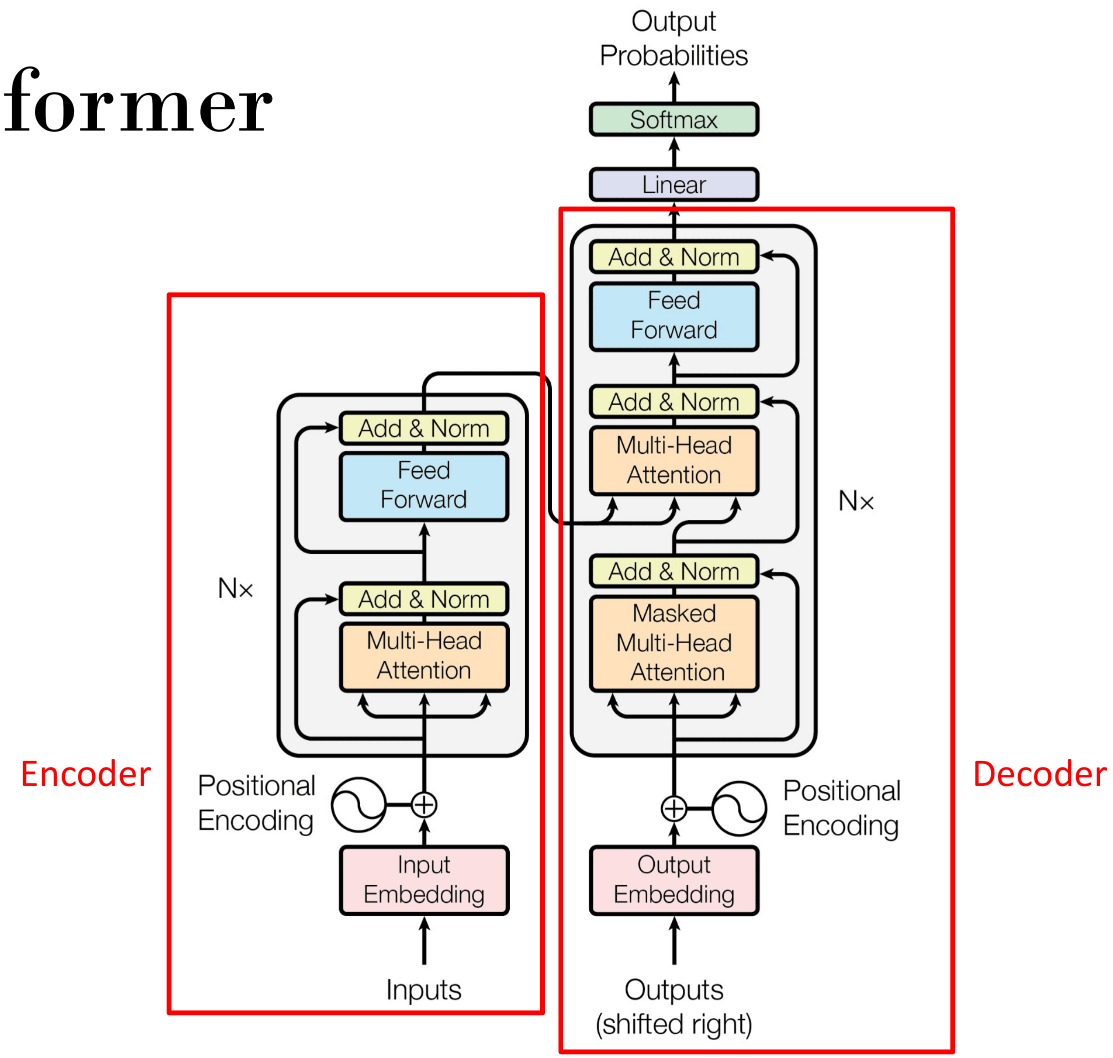
Llion Jones[*]
Google Research
llion@google.com

Aidan N. Gomez[*] [†]
University of Toronto
aidan@cs.toronto.edu

Łukasz Kaiser[*]
Google Brain
lukaszkaiser@google.com

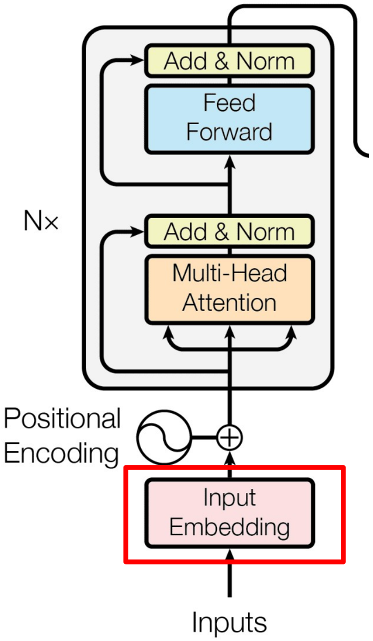
Illia Polosukhin[*] [‡]
illia.polosukhin@gmail.com

Transformer

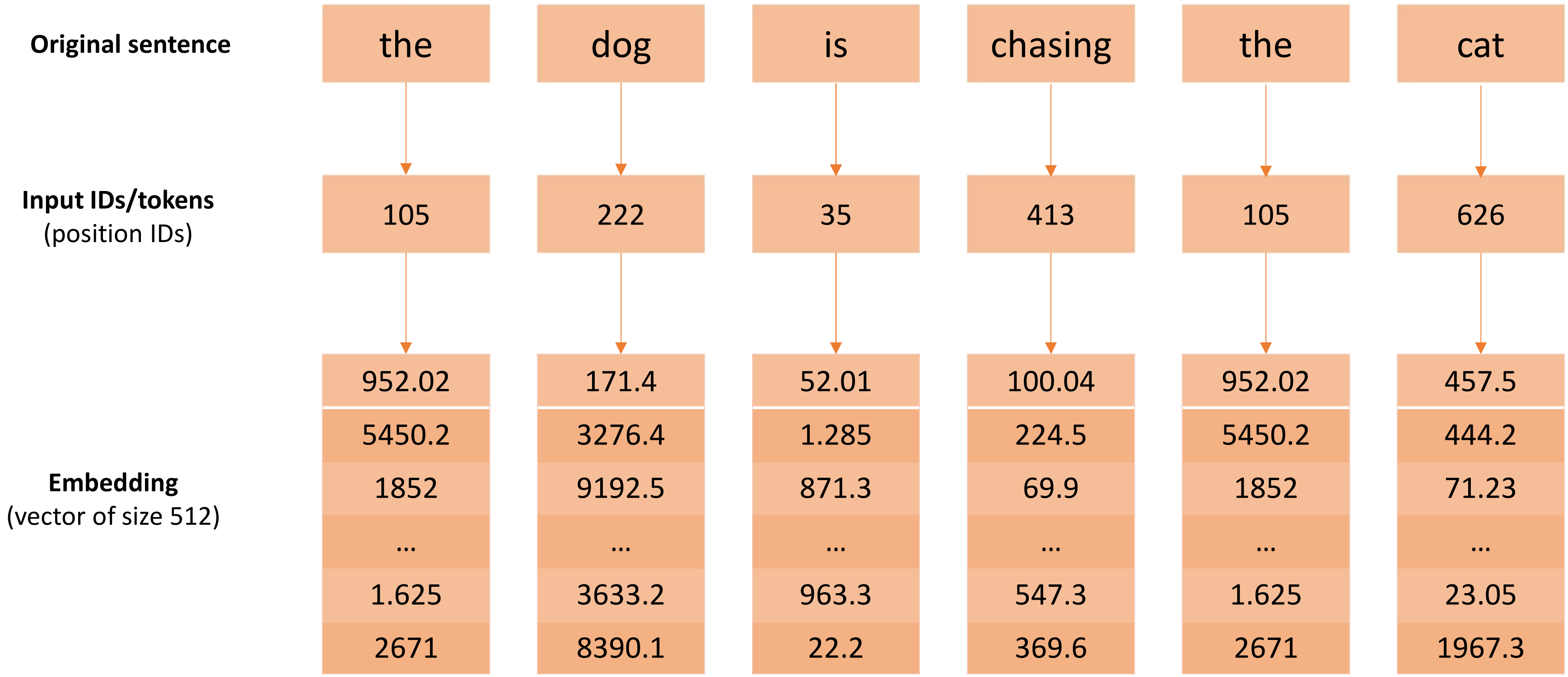


“Attention is all you need”

Input Embedding



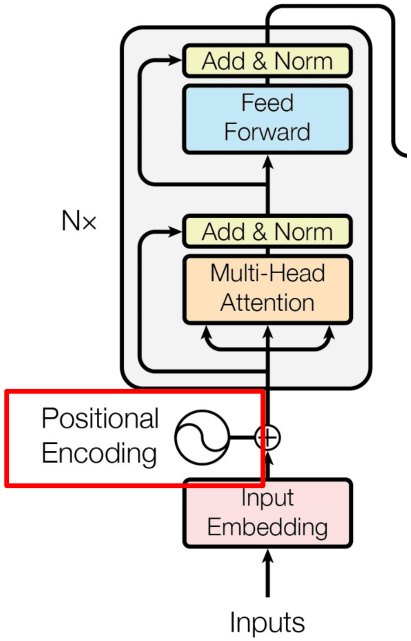
Input Embeddings



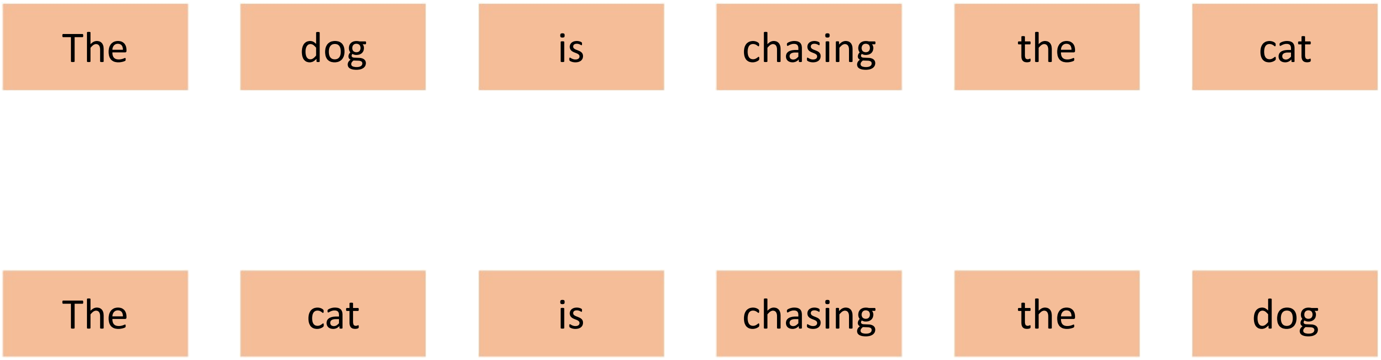
Original sentence	the	dog	is	chasing	the	cat
Input IDs/tokens (position IDs)	105	222	35	413	105	626
Embedding (vector of size 512)	952.02 5450.2 1852 … 1.625 2671	171.4 3276.4 9192.5 … 3633.2 8390.1	52.01 1.285 871.3 … 963.3 22.2	100.04 224.5 69.9 … 547.3 369.6	952.02 5450.2 1852 … 1.625 2671	457.5 444.2 71.23 … 23.05 1967.3

$\mathbf{d}_{\mathbf{model}}$ = 512, which represents size of the embedding vector

Positional Encoding



Positional Encoding



The dog is chasing the cat

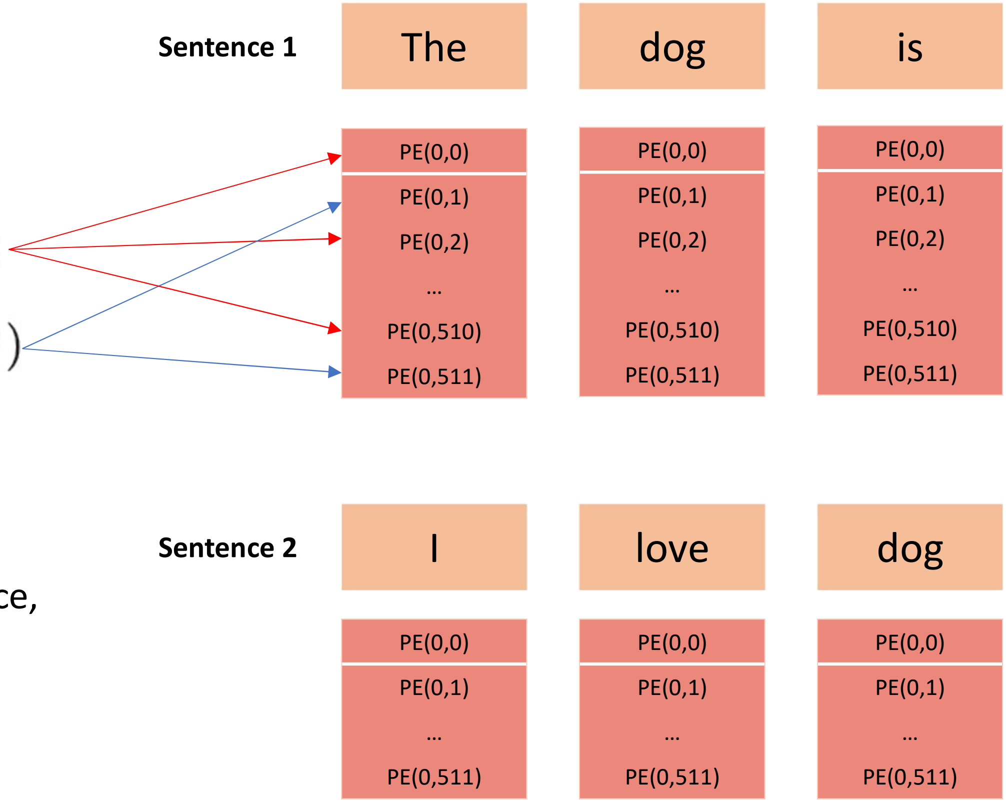
The cat is chasing the dog

Positional encoding represent the spatial pattern between each word/token that model can learn

Positional Encoding

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{model}})$$

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



Sentence 1

The	dog	is

PE(0,0)
PE(0,1)
PE(0,2)
…
PE(0,510)
PE(0,511)
	PE(0,0)
	PE(0,1)
	PE(0,2)
	…
	PE(0,510)
	PE(0,511)
		PE(0,0)
		PE(0,1)
		PE(0,2)
		…
		PE(0,510)
		PE(0,511)

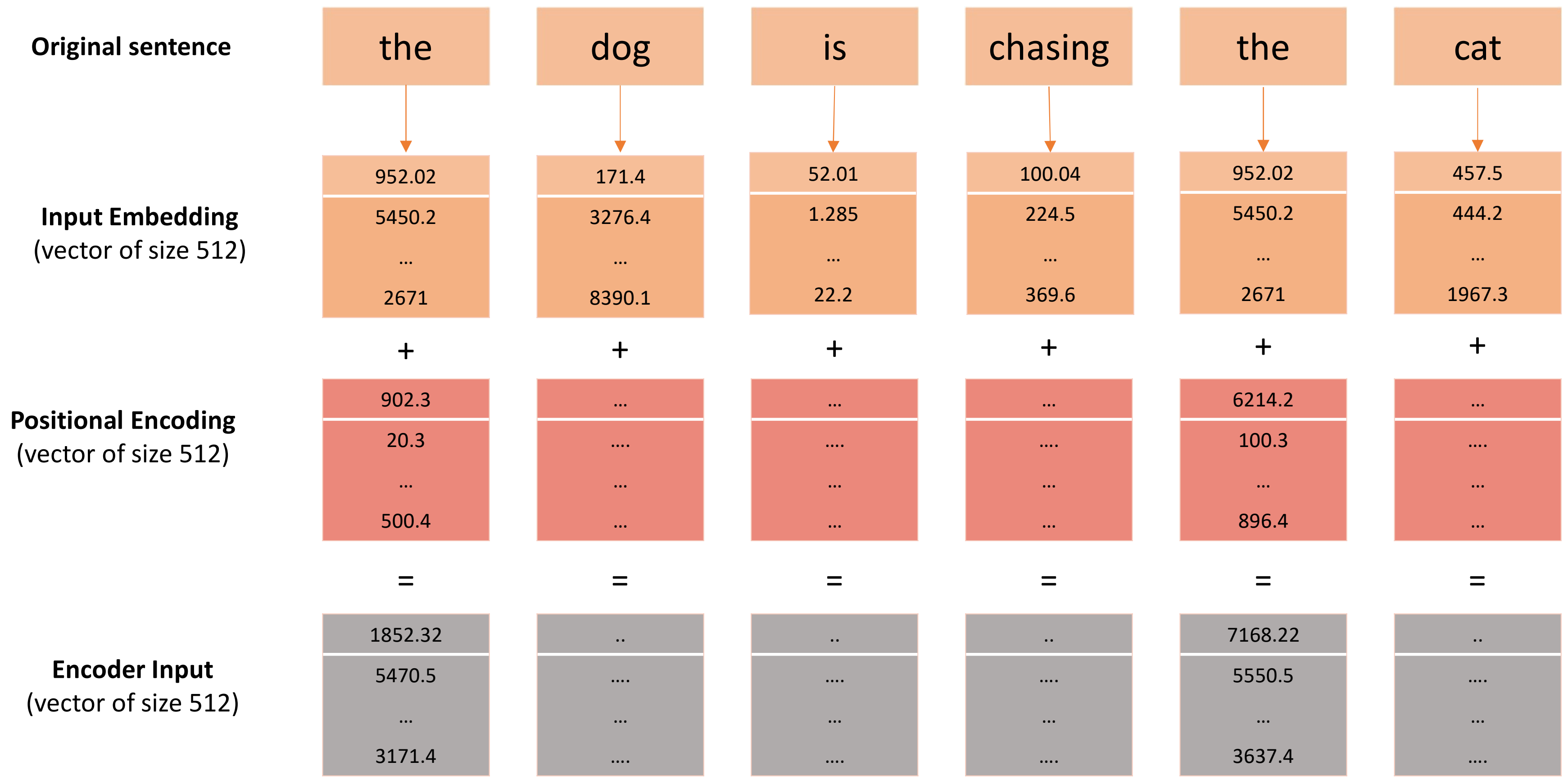
Sentence 2

I	love	dog

PE(0,0)
PE(0,1)
…
PE(0,511)
	PE(0,0)
	PE(0,1)
	…
	PE(0,511)
		PE(0,0)
		PE(0,1)
		…
		PE(0,511)

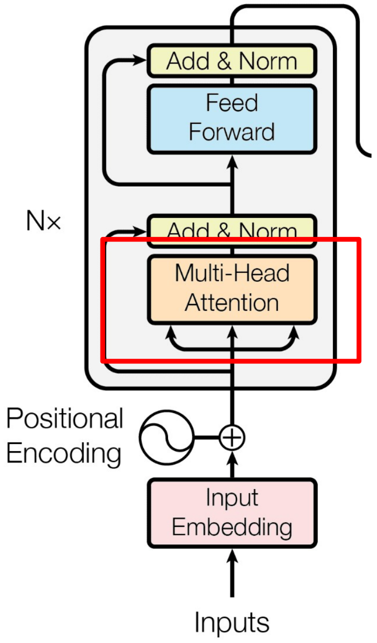
We only compute the positional encoding once, and reuse it for every sentence

Positional Encoding



Original sentence	the	dog	is	chasing	the	cat
Input Embedding (vector of size 512)	952.02 5450.2 ... 2671	171.4 3276.4 ... 8390.1	52.01 1.285 ... 22.2	100.04 224.5 ... 369.6	952.02 5450.2 ... 2671	457.5 444.2 ... 1967.3
	+	+	+	+	+	+
Positional Encoding (vector of size 512)	902.3 20.3 ... 500.4	... .	... .	... .	6214.2 100.3 ... 896.4	... .
	=	=	=	=	=	=
Encoder Input (vector of size 512)	1852.32 5470.5 ... 3171.4	.. ..	.. ..	.. ..	7168.22 5550.5 ... 3637.4	.. ..

Attention


Add & Norm
Feed
Forward
Nx
Add & Norm
Multi-Head
Attention
Positional
Encoding
Input
Embedding
Inputs

Self-Attention

- Self-Attention allows model to learn how words are related to each other
- For example, in sentence “The dog is chasing the cat”, self-attention mechanism will give higher score to the word “dog” and “chasing”, because in this context dog is doing the action -> chasing.
- $Attention\ (Q, K, V) = Softmax\left(\frac{QK^T}{\sqrt{d}}\right)V$
- So for the sentence, “The dog is chasing the cat”, the attention matrix might look like:

The	0.03
dog	0.4
is	0.05
chasing	0.3
the	0.02
cat	0.2

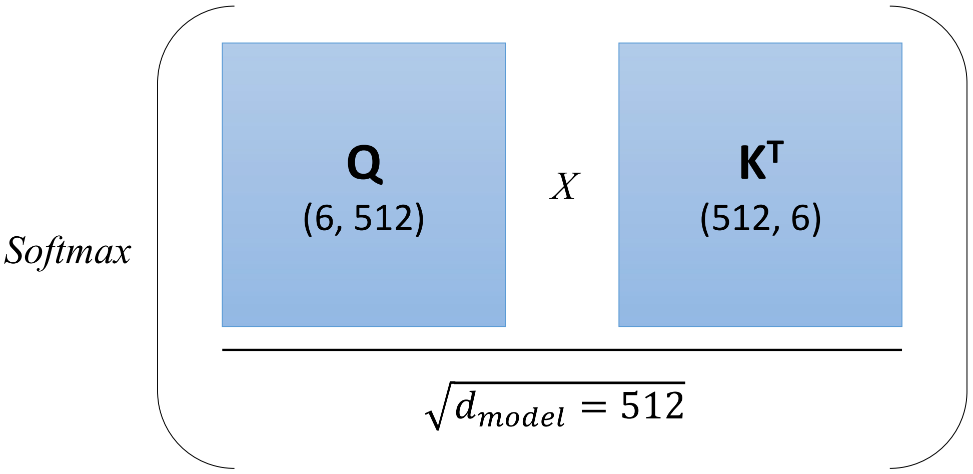
Self-Attention

The dog is chasing the cat

Suppose, **seq** = 6 and **d$_{model}$** = 512

Q, **K** and **V** are just input sentence embeddings

$$Attention\,(Q,K,V) = Softmax\left(\frac{QK^T}{\sqrt{d_{model}}}\right)V$$



$$Softmax\left(\frac{\underset{(6,\ 512)}{\mathbf{Q}} \; X \; \underset{(512,\ 6)}{\mathbf{K^T}}}{\sqrt{d_{model}} = 512}\right) =$$

	The	dog	is	chasing	the	cat
The	0.268	0.119	0.134	0.148	0.179	0.152
dog	0.124	0.278	0.128	0.201	0.115	0.154
is	0.147	0.132	0.262	0.097	0.218	0.145
chasing	0.128	0.201	0.206	0.212	0.119	0.125
the	0.146	0.158	0.152	0.143	0.227	0.174
cat	0.195	0.114	0.203	0.103	0.157	0.229

(6,6)

Self-Attention

$$Attention\ (Q,K,V) = Softmax\left(\frac{QK^T}{\sqrt{d_{model}}}\right)V$$

	The	dog	is	chasing	the	cat
The	0.268	0.119	0.134	0.148	0.179	0.152
dog	0.124	0.278	0.128	0.201	0.115	0.154
is	0.147	0.132	0.262	0.097	0.218	0.145
chasing	0.128	0.201	0.206	0.212	0.119	0.125
the	0.146	0.158	0.152	0.143	0.227	0.174
cat	0.195	0.114	0.203	0.103	0.157	0.229

X

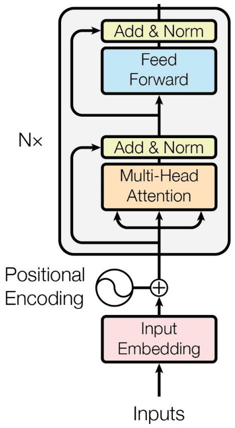
V
(6, 512)

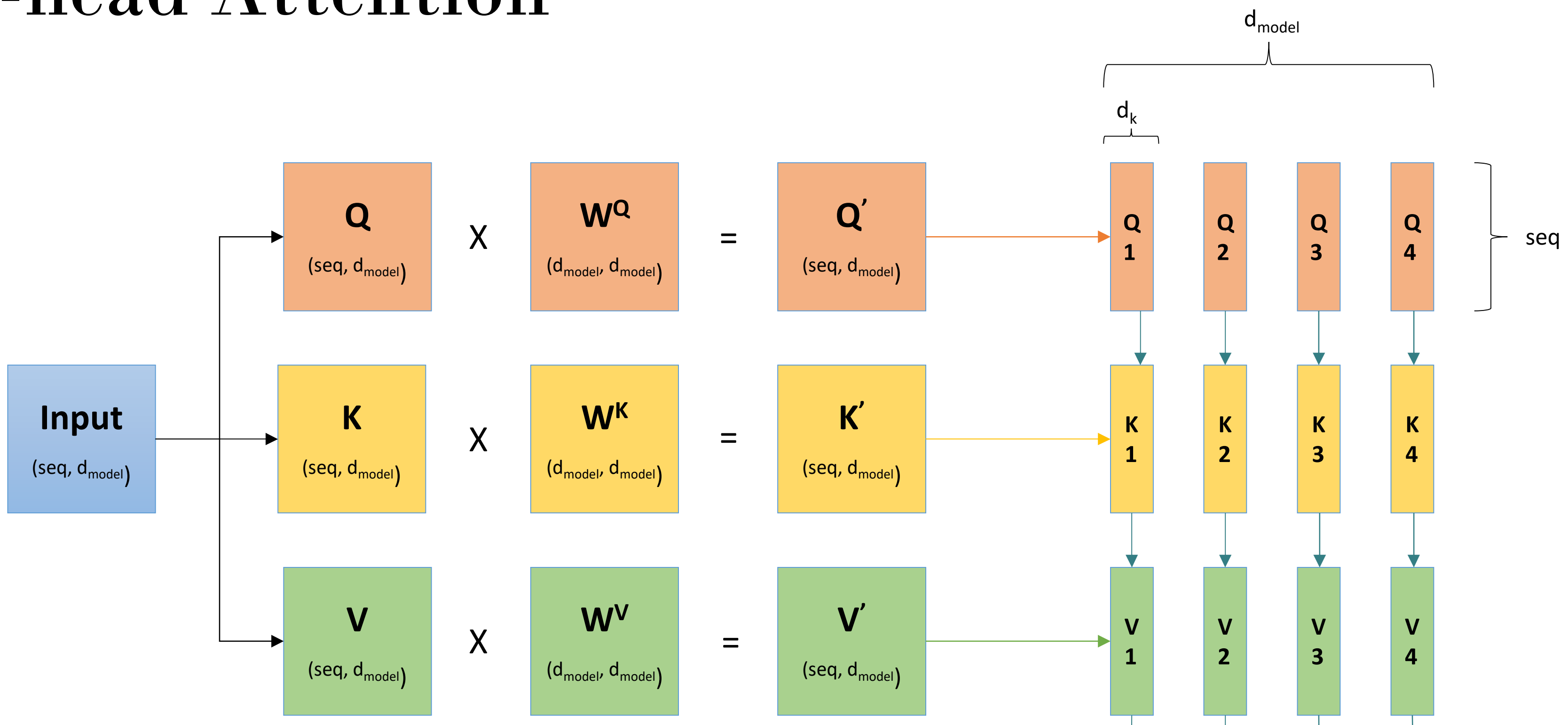
=

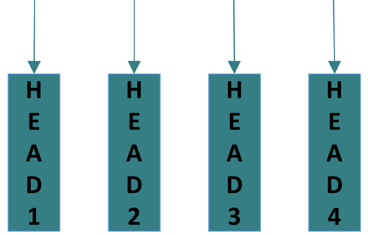
Attention
(6, 512)

Each row in this matrix not only captures
the embeddings or the position in the sentence
but also, each word's interactions with each other

Multi-head Attention



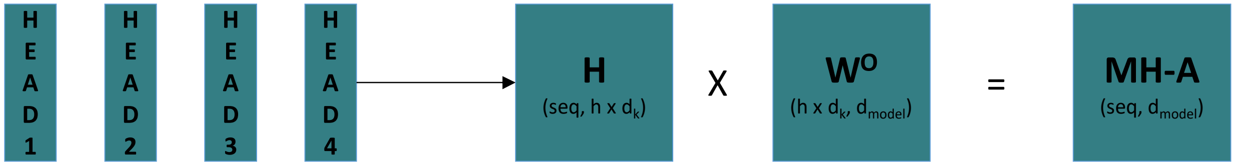




$$Attention\ (Q,K,V) = Softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

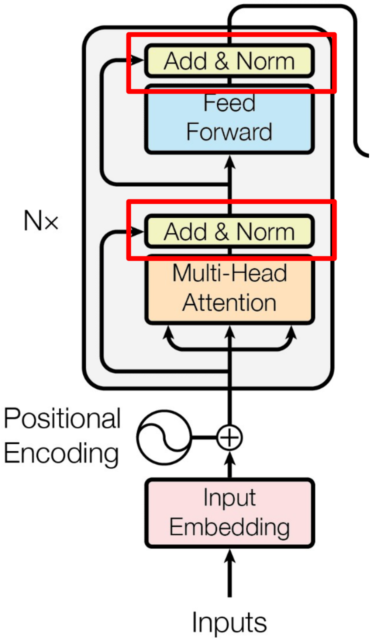
$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$$

Multi-head Attention

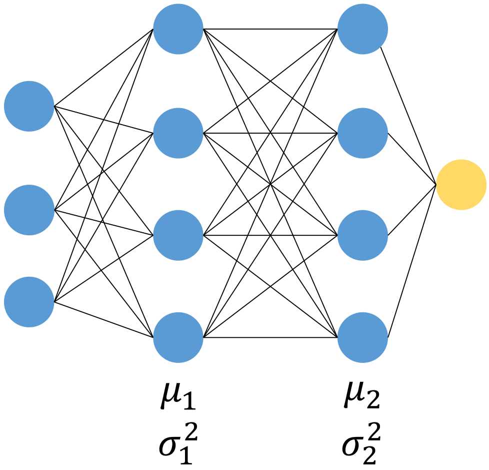


$$MultiHead\ (Q, K, V) = Concat(head_1, \ldots, head_h)W^O$$

Normalization

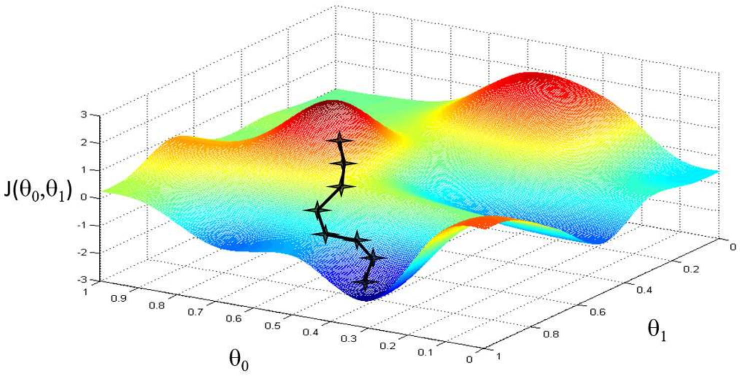


Layer Normalization



Gradient Descend

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



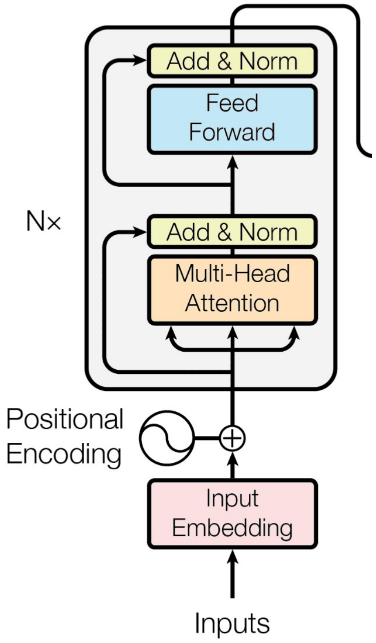
Without normalization, the input to each layer can vary a lot, causing erratic gradients.

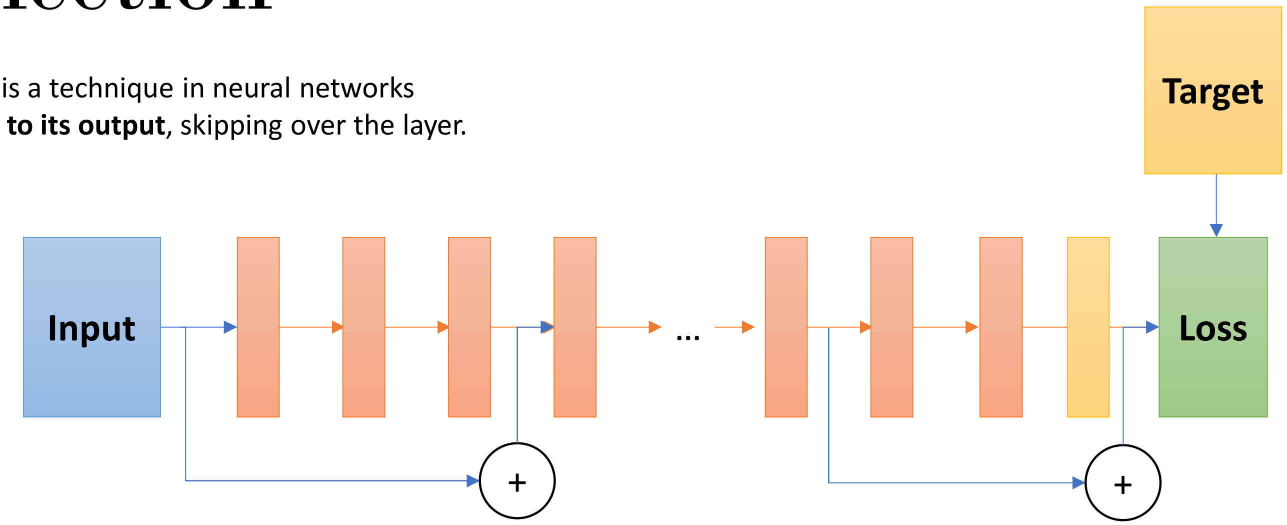
$$\widehat{x_j} = \frac{x_j - \mu_j}{\sqrt{{\sigma_j}^2 + \in}} * \gamma + \beta$$

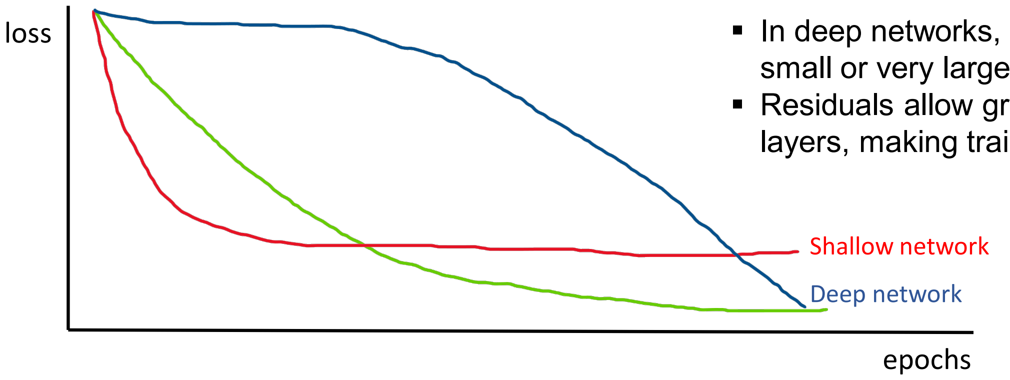
We also introduce two learnable parameters, usually called **gamma** (multiplicative) and **beta** (additive) that introduce some fluctuations in data. Model learns to tune these two parameters to introduce fluctuations when necessary.

Residual Connection

A **residual connection** (or **skip connection**) is a technique in neural networks where the input to a layer is **added directly to its output**, skipping over the layer.



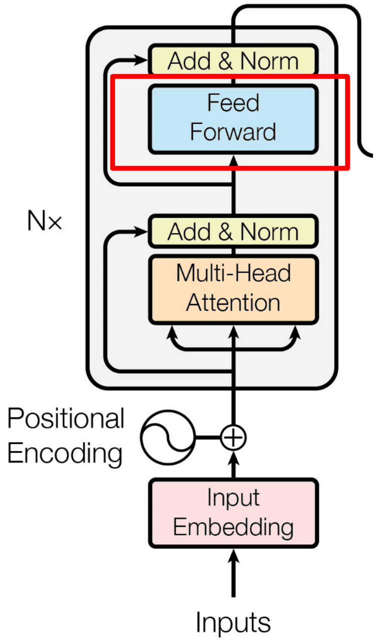




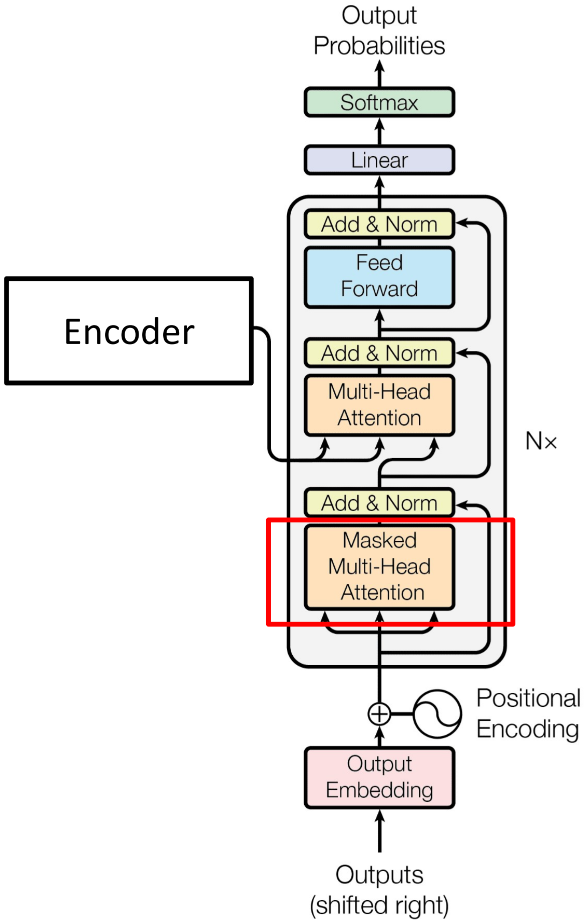
- In deep networks, gradients can become very small or very large as they are backpropagated.
- Residuals allow gradients to flow directly through layers, making training deep networks feasible.

Feed Forward Layer

- Refines token representations by applying non-linear transformations independently to each position after attention.
- Increases model capacity through dimensional expansion and compression (e.g., 512 → 2048 → 512).
- Adds non-linearity via activation functions (e.g., ReLU or GELU), enabling the model to learn complex patterns.



Decoder


Output
Probabilities
Softmax
Linear
Add & Norm
Feed
Forward
Add & Norm
Multi-Head
Attention
Encoder
Nx
Add & Norm
Masked
Multi-Head
Attention
Positional
Encoding
Output
Embedding
Outputs
(shifted right)

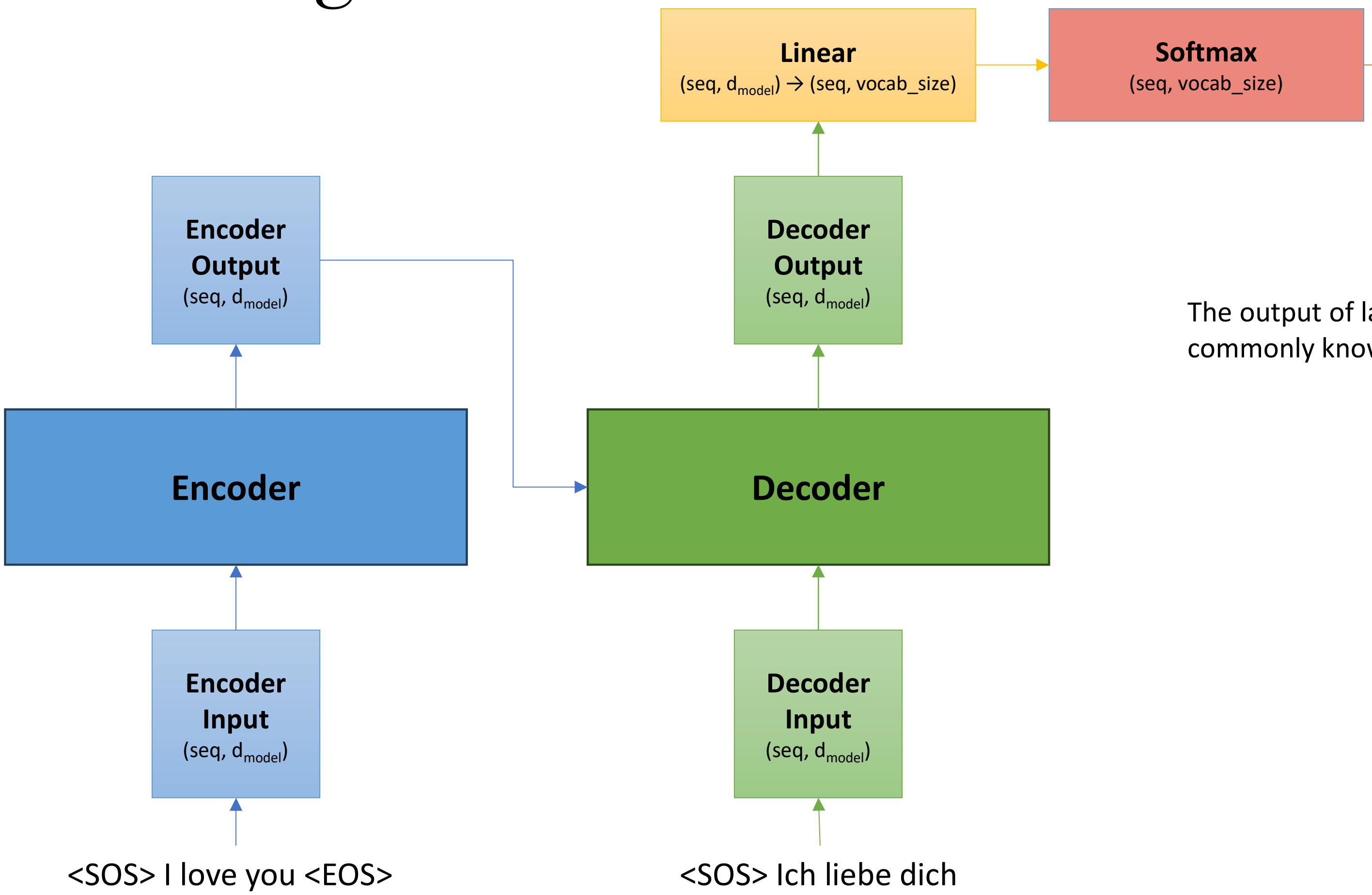
Masked Multi-Head Attention

Our goal is to make the model **causal**, ensuring that during masked multi-head attention, each position can only attend to previous or current positions and not future tokens. This prevents the model from accessing information from future words during training.

	The	dog	is	chasing	the	cat
The	0.268	0.119	0.134	0.148	0.179	0.152
dog	0.124	0.278	0.128	0.201	0.115	0.154
is	0.147	0.132	0.262	0.097	0.218	0.145
chasing	0.128	0.201	0.206	0.212	0.119	0.125
the	0.146	0.158	0.152	0.143	0.227	0.174
cat	0.195	0.114	0.203	0.103	0.157	0.229

To enforce causality in masked multi-head attention, elements above the main diagonal in the attention score matrix are replaced with negative infinity ($-\infty$). After applying the softmax function, these $-\infty$ values become zeros, effectively masking out future tokens and ensuring that each position can only attend to itself and preceding positions.

Training



<SOS> I love you <EOS>

Encoder Input
(seq, d_{model})

Encoder

Encoder Output
(seq, d_{model})

<SOS> Ich liebe dich

Decoder Input
(seq, d_{model})

Decoder

Decoder Output
(seq, d_{model})

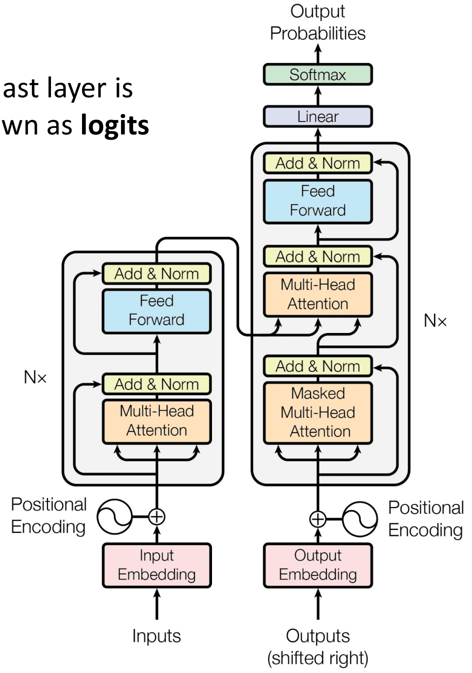
Linear
(seq, d_{model}) → (seq, vocab_size)

Softmax
(seq, vocab_size)

Cross Entropy Loss

Ich liebe dich <EOS>

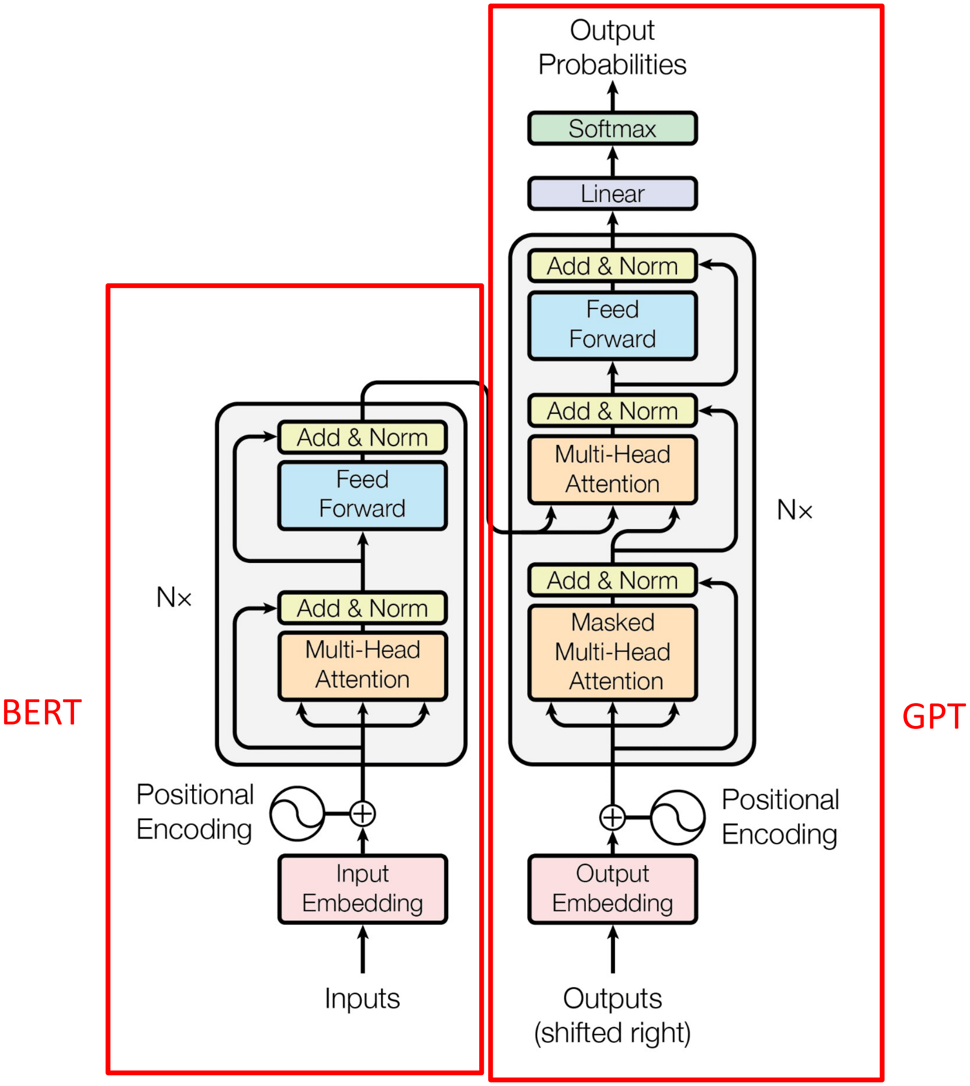
The output of last layer is commonly known as **logits**



Large Language Models

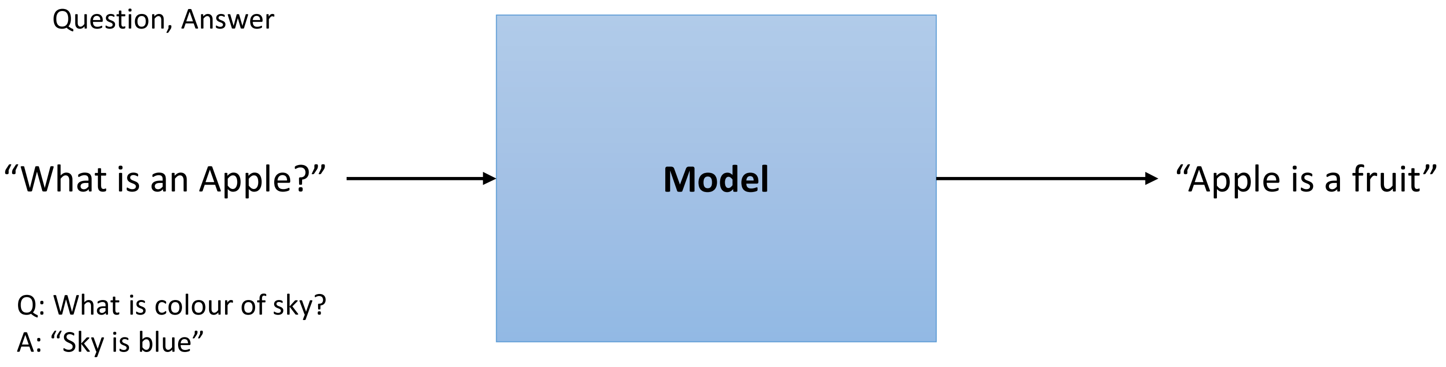
- **BERT (Bidirectional Encoder Representations from Transformers)** is a transformer-based model that uses only the encoder component of the original Transformer architecture. It consists of multiple encoder layers stacked on top of each other, allowing it to process input text in a bidirectional (context-aware) manner. This means that BERT considers both the left and right context of a word simultaneously, making it highly effective for understanding the meaning of words in context.

- **GPT (Generative Pre-trained Transformer),** on the other hand, is a decoder-only model built on the Transformer architecture. It stacks multiple transformer decoder blocks and processes text in a left-to-right (autoregressive) manner, meaning it only looks at past tokens to predict the next token. GPT is pre-trained using **causal language modeling (CLM)**, where it learns to predict the next word in a sequence. Its architecture and training make it especially suited for natural language generation (NLG) tasks, such as text completion, story generation, and dialogue systems.



GPT-Model

- **Pretraining:** The model is trained using **self-supervised learning** to develop a general understanding of the language. It learns patterns, grammar, and semantics from large amounts of unlabeled text.
- **Fine tuning:** The pretrained model is further trained on a specific task using **supervised learning** with labeled data. This adapts the model to perform well on tasks like classification, translation, or question answering.
- **Meta-learning**: The model is trained across multiple tasks to learn how to adapt quickly to new tasks with minimal data, effectively "learning how to learn."



Question, Answer

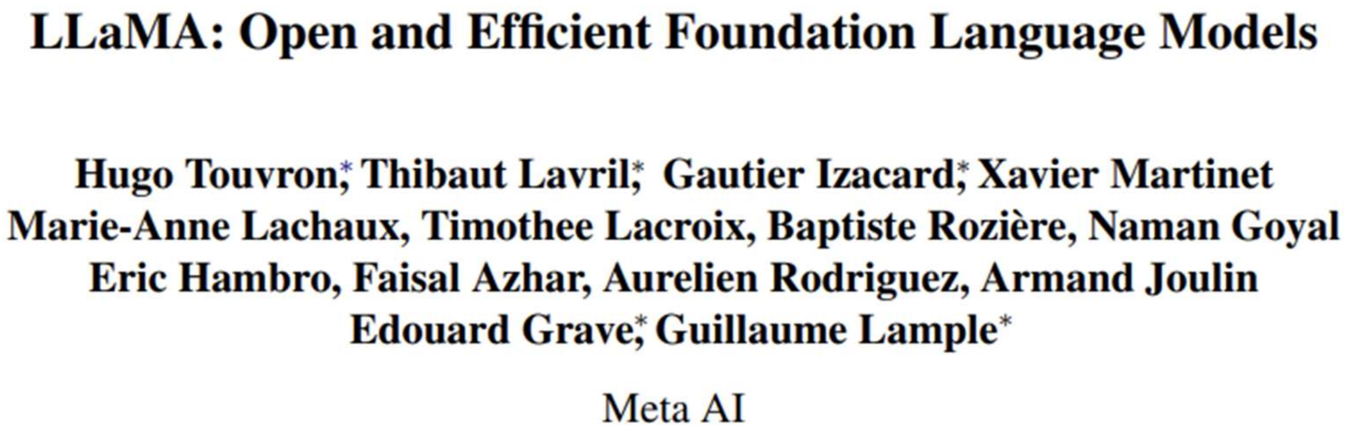
“What is an Apple?”

Model

“Apple is a fruit”

Q: What is colour of sky?
A: “Sky is blue”

Llama Model

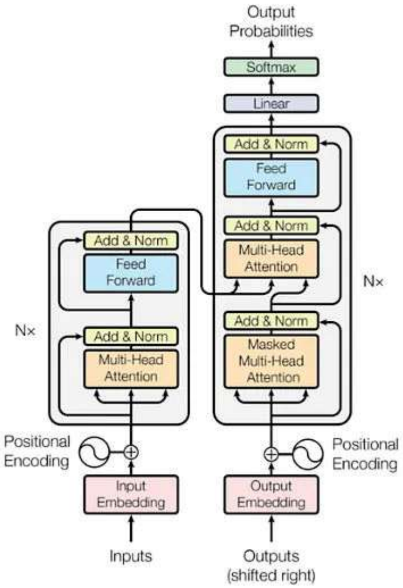


LLaMA: Open and Efficient Foundation Language Models

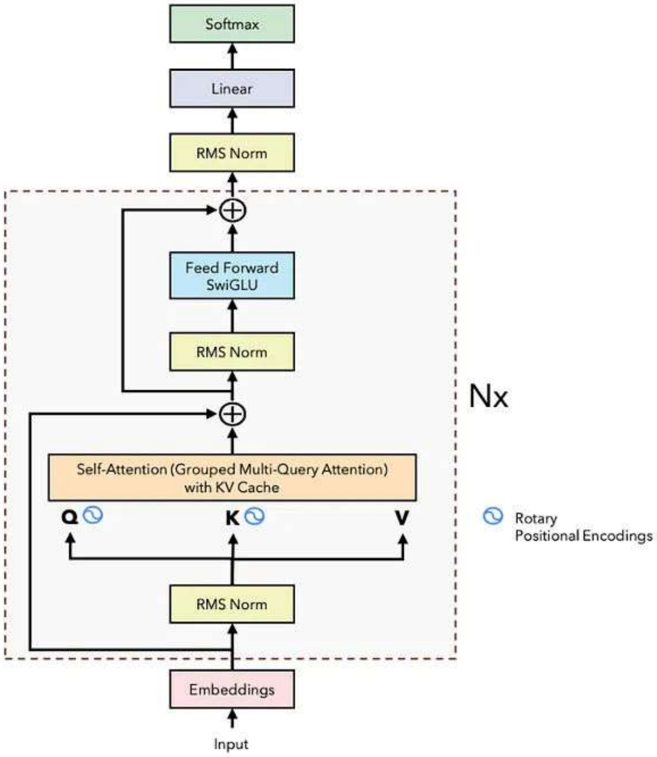
Hugo Touvron[*], Thibaut Lavril[*], Gautier Izacard[*], Xavier Martinet
Marie-Anne Lachaux, Timothee Lacroix, Baptiste Rozière, Naman Goyal
Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin
Edouard Grave[*], Guillaume Lample[*]

Meta AI

LLaMA Model

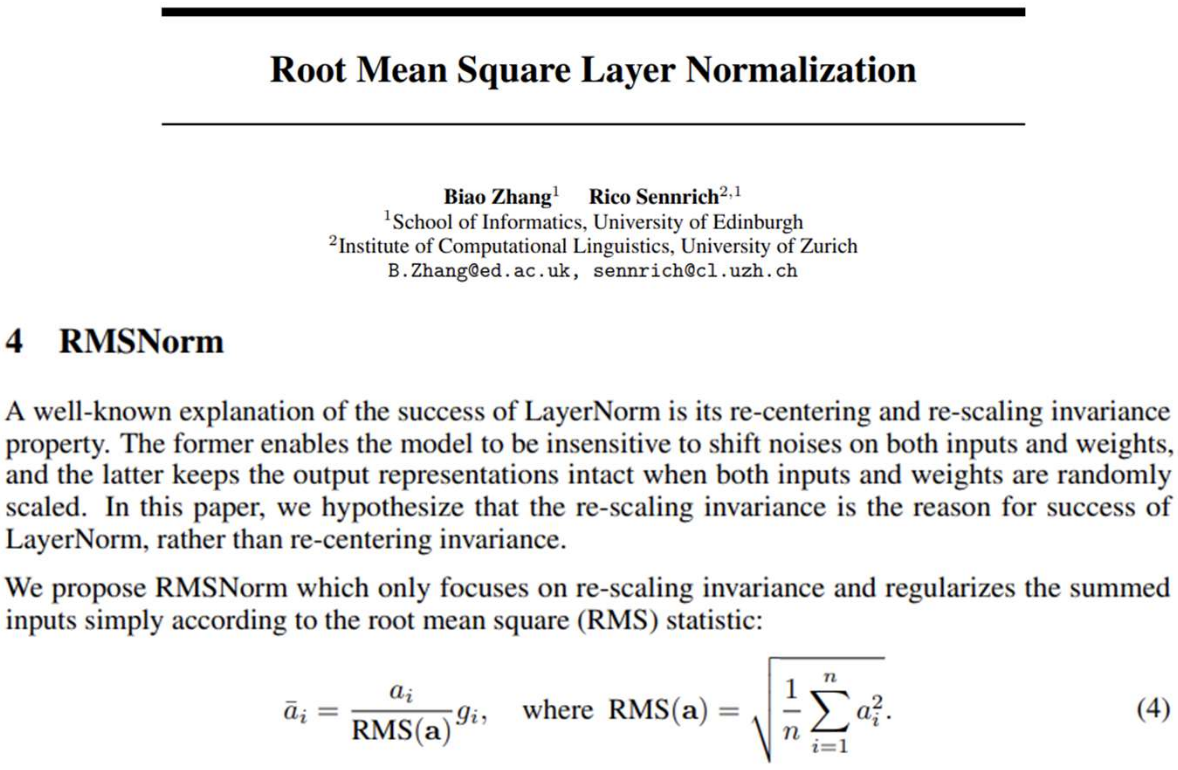


Transformer
("Attention is all you need")


Softmax
Linear
RMS Norm
Feed Forward SwiGLU
RMS Norm
Nx
Self-Attention (Grouped Multi-Query Attention) with KV Cache
Q
K
V
Rotary Positional Encodings
RMS Norm
Embeddings
Input

LLaMA

RMS Normalization



Root Mean Square Layer Normalization

Biao Zhang[1] **Rico Sennrich**[2,1]
[1]School of Informatics, University of Edinburgh
[2]Institute of Computational Linguistics, University of Zurich
B.Zhang@ed.ac.uk, sennrich@cl.uzh.ch

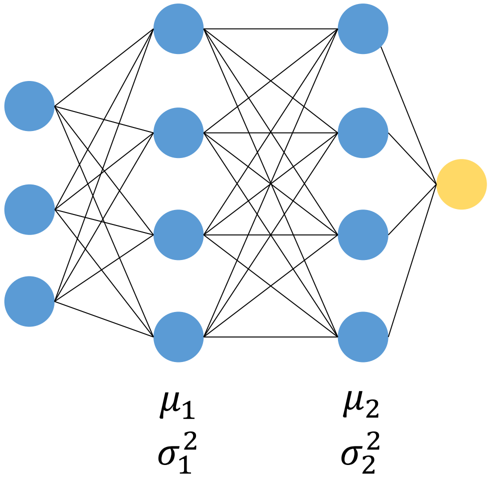
4 RMSNorm

A well-known explanation of the success of LayerNorm is its re-centering and re-scaling invariance property. The former enables the model to be insensitive to shift noises on both inputs and weights, and the latter keeps the output representations intact when both inputs and weights are randomly scaled. In this paper, we hypothesize that the re-scaling invariance is the reason for success of LayerNorm, rather than re-centering invariance.

We propose RMSNorm which only focuses on re-scaling invariance and regularizes the summed inputs simply according to the root mean square (RMS) statistic:

$$\bar{a}_i = \frac{a_i}{\text{RMS}(\mathbf{a})} g_i, \quad \text{where } \text{RMS}(\mathbf{a}) = \sqrt{\frac{1}{n} \sum_{i=1}^{n} a_i^2}. \tag{4}$$

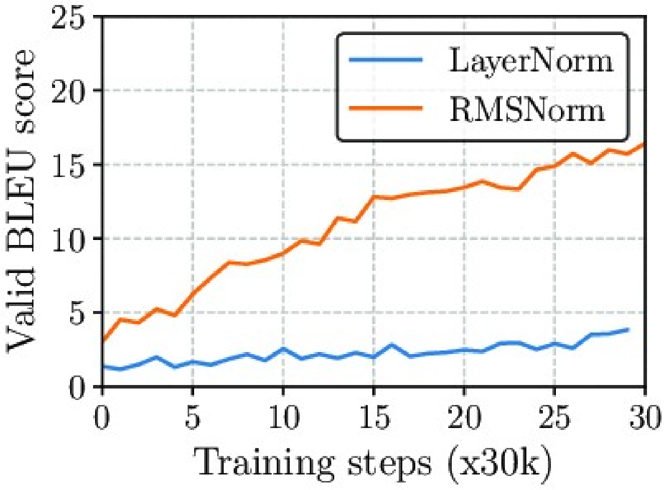
RMS Normalization

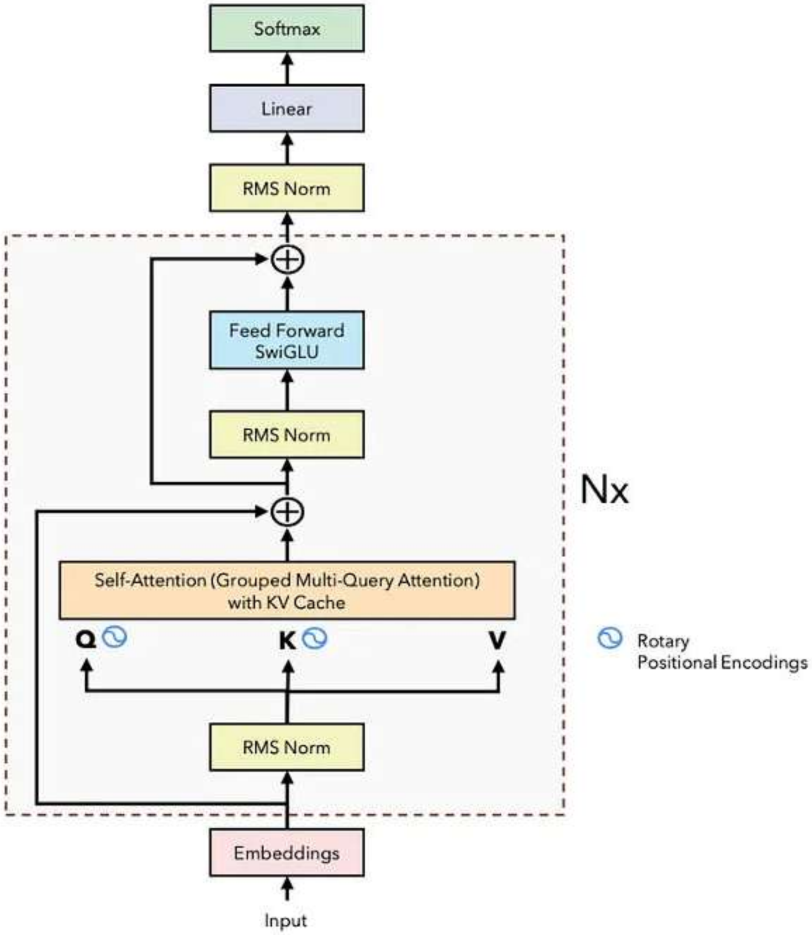


$$\widehat{y_j} = \frac{x_j - \mu_j}{\sqrt{{\sigma_j}^2 + \in}} * \gamma + \beta$$

μ: Mean
σ²: Variance

- Less computation time than layer normalization
- faster convergence, better generalization, or more stable training dynamics when using RMSNorm for language generation

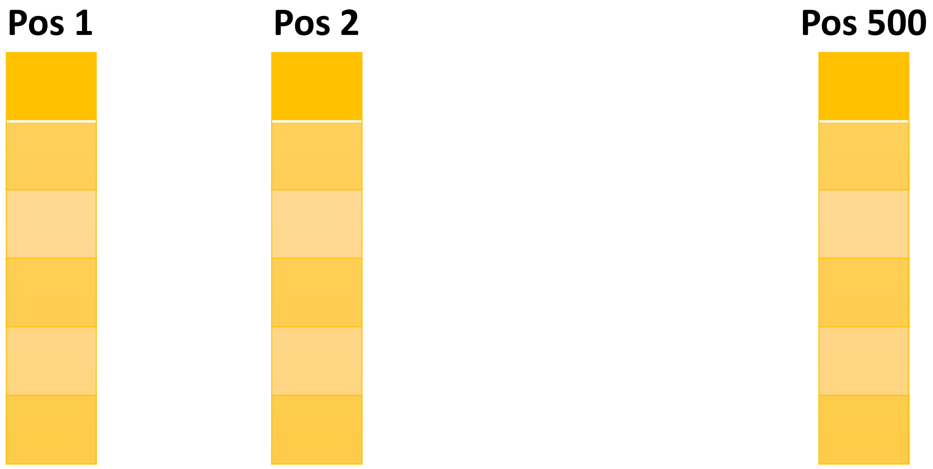



Softmax
Linear
RMS Norm
Feed Forward
SwiGLU
RMS Norm
Nx
Self-Attention (Grouped Multi-Query Attention)
with KV Cache
Q
K
V
Rotary
Positional Encodings
RMS Norm
Embeddings
Input

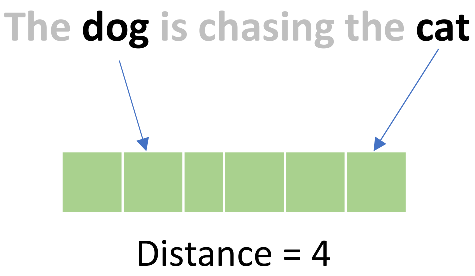
LLaMA

Problem with Positional Embeddings

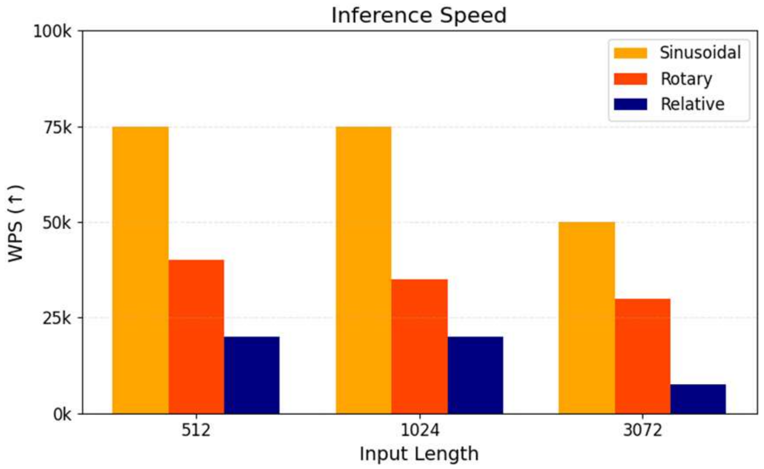
Positional encodings represent absolute positions, so they don't directly capture relative distances between tokens — which are often more important for understanding context. Intuitively, positions 1 and 2 are close together, but the model doesn't inherently capture that relationship.



Relative Positional Embedding


The **dog** is chasing the **cat**

Distance = 4



The dog is chasing the cat

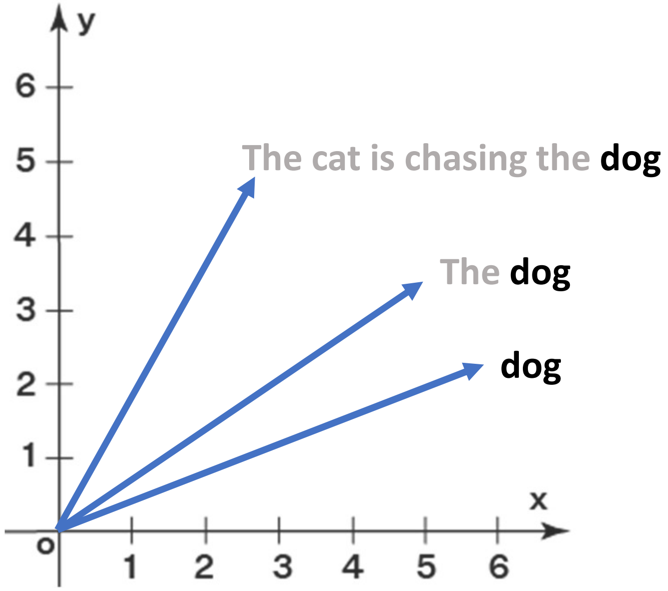
b_0	b_1	b_2	b_3	b_4	b_5
b_{-1}	b_0	b_1	b_2	b_3	b_4
b_{-2}	b_{-1}	b_0	b_1	b_2	b_3
b_{-3}	b_{-2}	b_{-1}	b_0	b_1	b_2
b_{-4}	b_{-3}	b_{-2}	b_{-1}	b_0	b_1
b_{-5}	b_{-4}	b_{-3}	b_{-2}	b_{-1}	b_0

$$Softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)$$

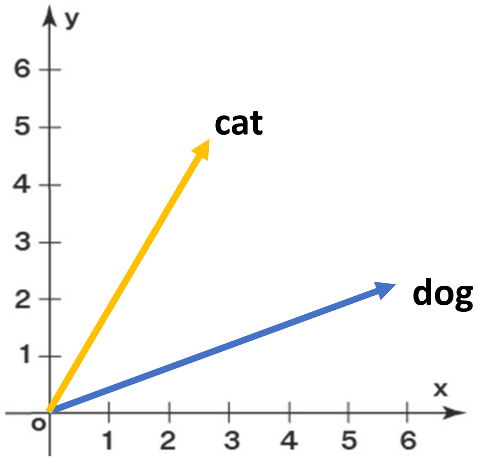
In the relative positional embedding matrix, b_1 represents a word one position ahead, b_{-1} a word one position behind, and b_4 a word four positions ahead of the current word.

Rotary Positional Embedding

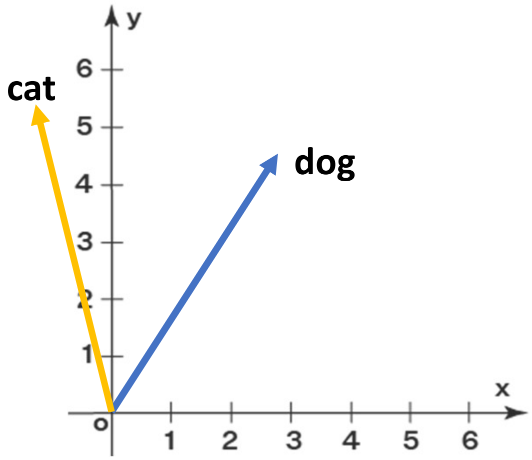
- Instead of adding a positional vector to the token embedding, the embedding is rotated by an angle θ based on its position.
- This approach has a key advantage: more tokens can be added after a given word without affecting its encoded position.



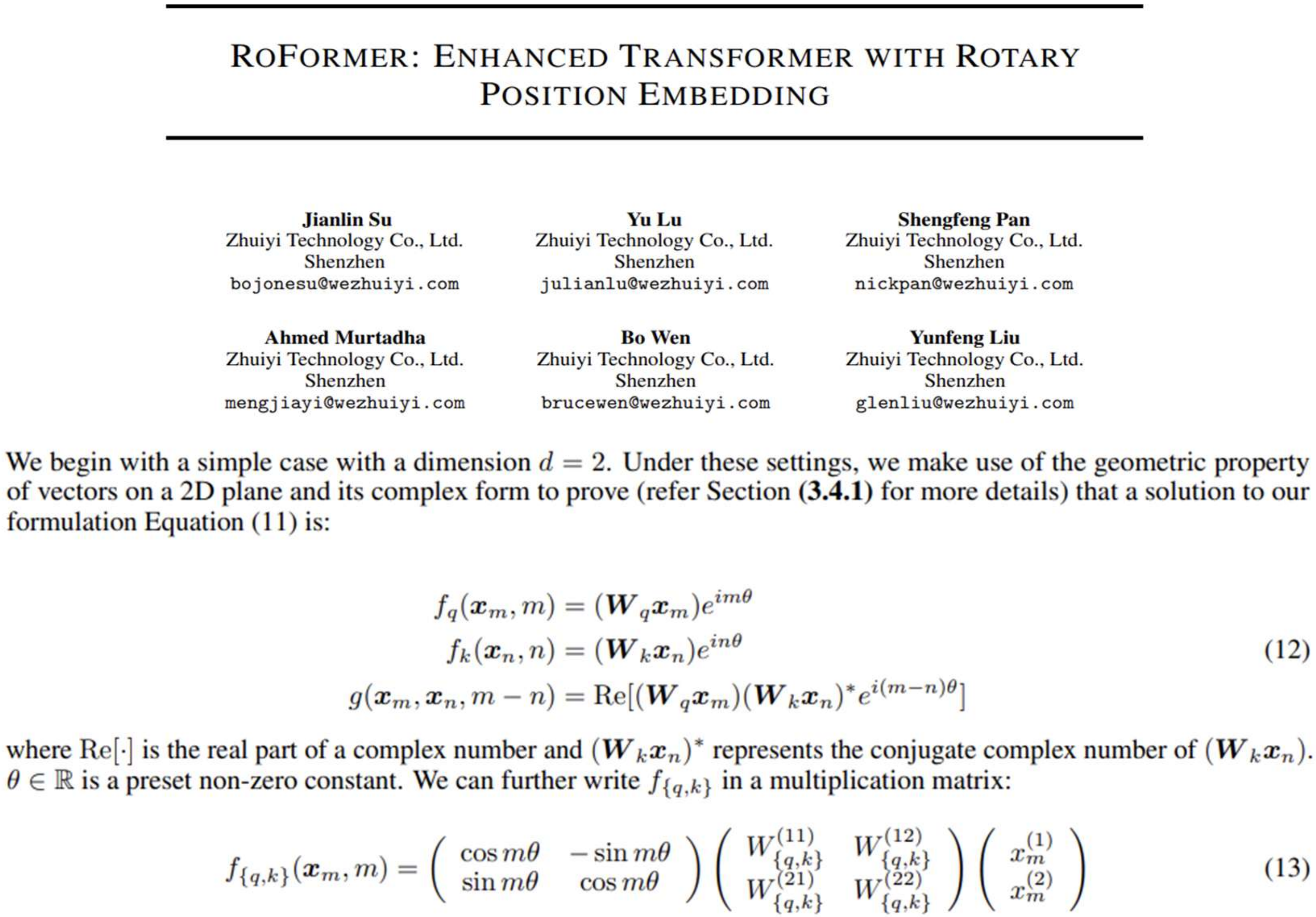
Rotary Positional Embedding



The **dog** chased the **cat**



Once upon a time, the **dog** chased the **cat**



RoFormer: Enhanced Transformer with Rotary Position Embedding

Jianlin Su
Zhuiyi Technology Co., Ltd.
Shenzhen
bojonesu@wezhuiyi.com

Yu Lu
Zhuiyi Technology Co., Ltd.
Shenzhen
julianlu@wezhuiyi.com

Shengfeng Pan
Zhuiyi Technology Co., Ltd.
Shenzhen
nickpan@wezhuiyi.com

Ahmed Murtadha
Zhuiyi Technology Co., Ltd.
Shenzhen
mengjiayi@wezhuiyi.com

Bo Wen
Zhuiyi Technology Co., Ltd.
Shenzhen
brucewen@wezhuiyi.com

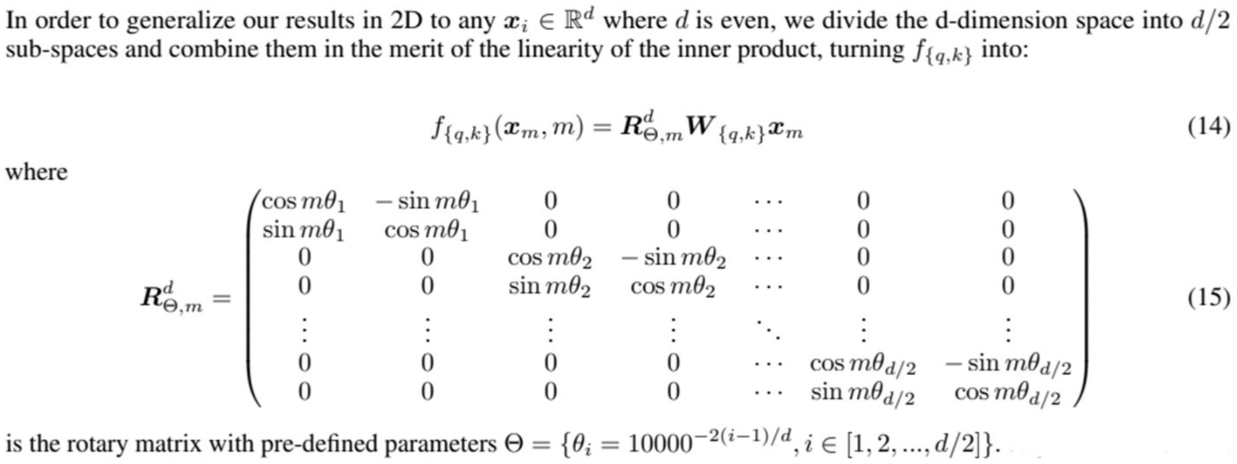
Yunfeng Liu
Zhuiyi Technology Co., Ltd.
Shenzhen
glenliu@wezhuiyi.com

We begin with a simple case with a dimension $d = 2$. Under these settings, we make use of the geometric property of vectors on a 2D plane and its complex form to prove (refer Section **(3.4.1)** for more details) that a solution to our formulation Equation (11) is:

$$\begin{aligned} f_q(\boldsymbol{x}_m, m) &= (\boldsymbol{W}_q \boldsymbol{x}_m) e^{im\theta} \\ f_k(\boldsymbol{x}_n, n) &= (\boldsymbol{W}_k \boldsymbol{x}_n) e^{in\theta} \\ g(\boldsymbol{x}_m, \boldsymbol{x}_n, m-n) &= \mathrm{Re}[(\boldsymbol{W}_q \boldsymbol{x}_m)(\boldsymbol{W}_k \boldsymbol{x}_n)^* e^{i(m-n)\theta}] \end{aligned} \tag{12}$$

where $\mathrm{Re}[\cdot]$ is the real part of a complex number and $(\boldsymbol{W}_k \boldsymbol{x}_n)^*$ represents the conjugate complex number of $(\boldsymbol{W}_k \boldsymbol{x}_n)$. $\theta \in \mathbb{R}$ is a preset non-zero constant. We can further write $f_{\{q,k\}}$ in a multiplication matrix:

$$f_{\{q,k\}}(\boldsymbol{x}_m, m) = \begin{pmatrix} \cos m\theta & -\sin m\theta \\ \sin m\theta & \cos m\theta \end{pmatrix} \begin{pmatrix} W^{(11)}_{\{q,k\}} & W^{(12)}_{\{q,k\}} \\ W^{(21)}_{\{q,k\}} & W^{(22)}_{\{q,k\}} \end{pmatrix} \begin{pmatrix} x^{(1)}_m \\ x^{(2)}_m \end{pmatrix} \tag{13}$$



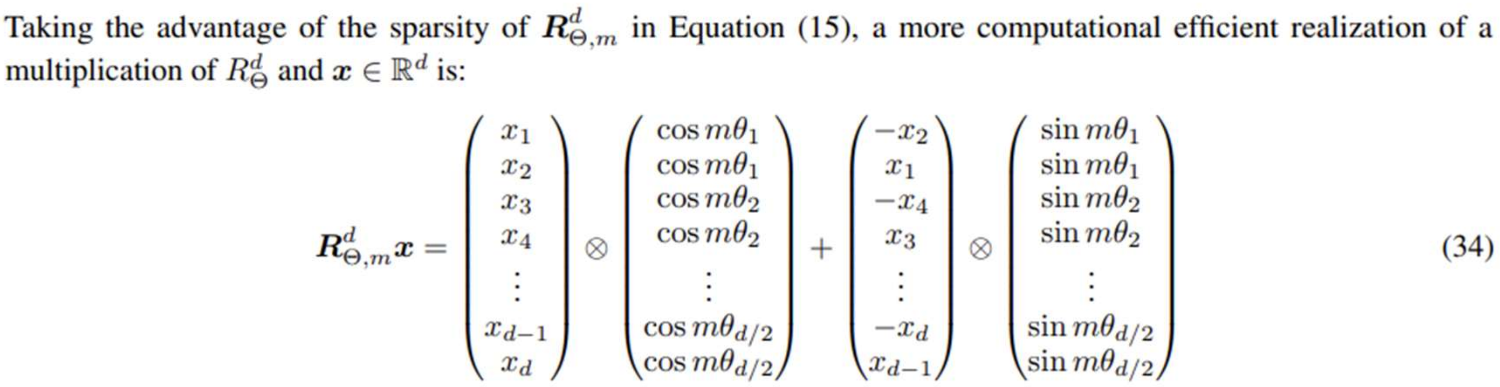
In order to generalize our results in 2D to any $\boldsymbol{x}_i \in \mathbb{R}^d$ where d is even, we divide the d-dimension space into $d/2$ sub-spaces and combine them in the merit of the linearity of the inner product, turning $f_{\{q,k\}}$ into:

$$f_{\{q,k\}}(\boldsymbol{x}_m, m) = \boldsymbol{R}^d_{\Theta,m} \boldsymbol{W}_{\{q,k\}} \boldsymbol{x}_m \tag{14}$$

where

$$\boldsymbol{R}^d_{\Theta,m} = \begin{pmatrix} \cos m\theta_1 & -\sin m\theta_1 & 0 & 0 & \cdots & 0 & 0 \\ \sin m\theta_1 & \cos m\theta_1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \cos m\theta_2 & -\sin m\theta_2 & \cdots & 0 & 0 \\ 0 & 0 & \sin m\theta_2 & \cos m\theta_2 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & \cos m\theta_{d/2} & -\sin m\theta_{d/2} \\ 0 & 0 & 0 & 0 & \cdots & \sin m\theta_{d/2} & \cos m\theta_{d/2} \end{pmatrix} \tag{15}$$

is the rotary matrix with pre-defined parameters $\Theta = \{\theta_i = 10000^{-2(i-1)/d}, i \in [1, 2, ..., d/2]\}$.



Taking the advantage of the sparsity of $\boldsymbol{R}^d_{\Theta,m}$ in Equation (15), a more computational efficient realization of a multiplication of R^d_{Θ} and $\boldsymbol{x} \in \mathbb{R}^d$ is:

$$\boldsymbol{R}^d_{\Theta,m}\boldsymbol{x} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ \vdots \\ x_{d-1} \\ x_d \end{pmatrix} \otimes \begin{pmatrix} \cos m\theta_1 \\ \cos m\theta_1 \\ \cos m\theta_2 \\ \cos m\theta_2 \\ \vdots \\ \cos m\theta_{d/2} \\ \cos m\theta_{d/2} \end{pmatrix} + \begin{pmatrix} -x_2 \\ x_1 \\ -x_4 \\ x_3 \\ \vdots \\ -x_d \\ x_{d-1} \end{pmatrix} \otimes \begin{pmatrix} \sin m\theta_1 \\ \sin m\theta_1 \\ \sin m\theta_2 \\ \sin m\theta_2 \\ \vdots \\ \sin m\theta_{d/2} \\ \sin m\theta_{d/2} \end{pmatrix} \tag{34}$$

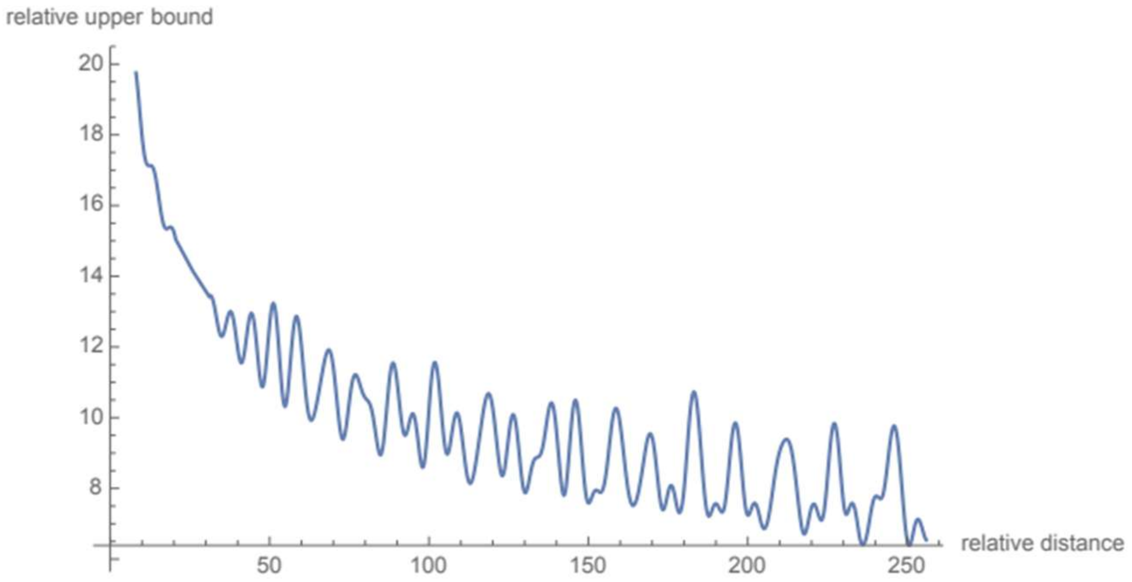
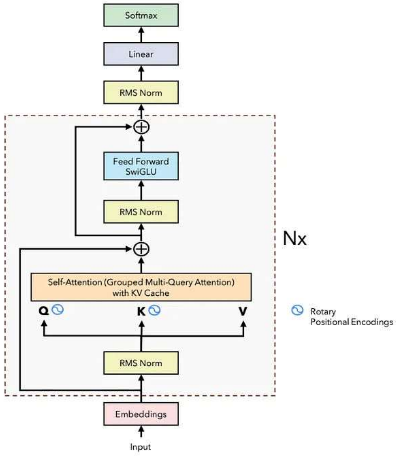


Figure 2: Long-term decay of RoPE.

Once upon a time, the **dog** chased the **cat**, but just as the cat leapt over a fallen flowerpot, the sky above them shimmered and a peculiar star blinked far too close to the **earth**, spinning like a dizzy firefly. The chase paused as both animals stared, their feud forgotten, watching the star burp out a trail of glittering stardust that made the garden hum like a sleepy lullaby.

Next token prediction task

Output: The dog is chasing the cat [EOS]



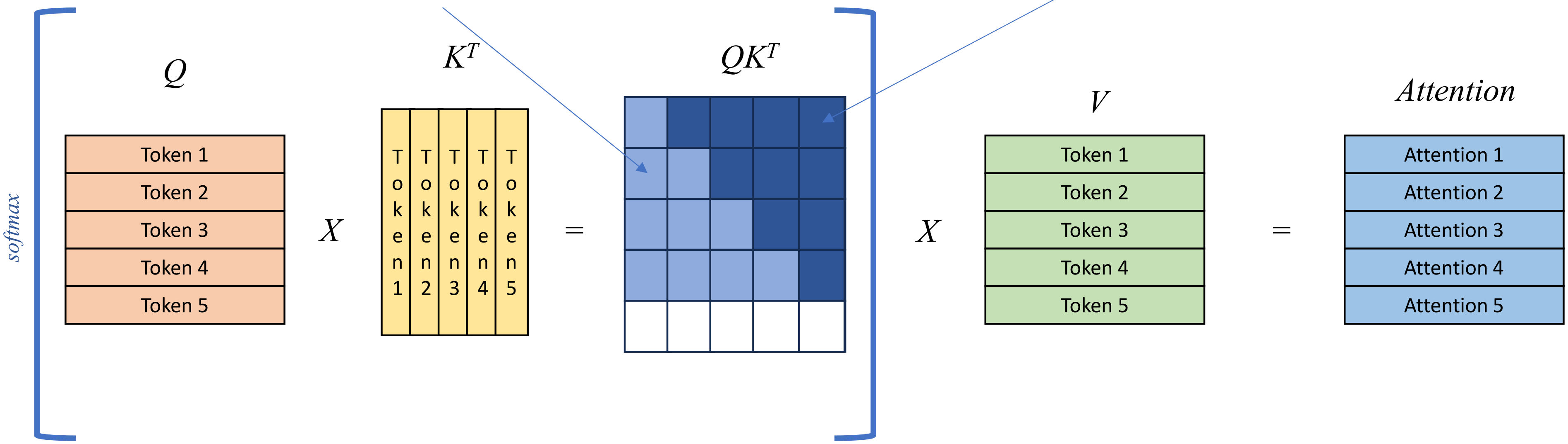
LLaMA

Input: [SOS] The dog is chasing the cat

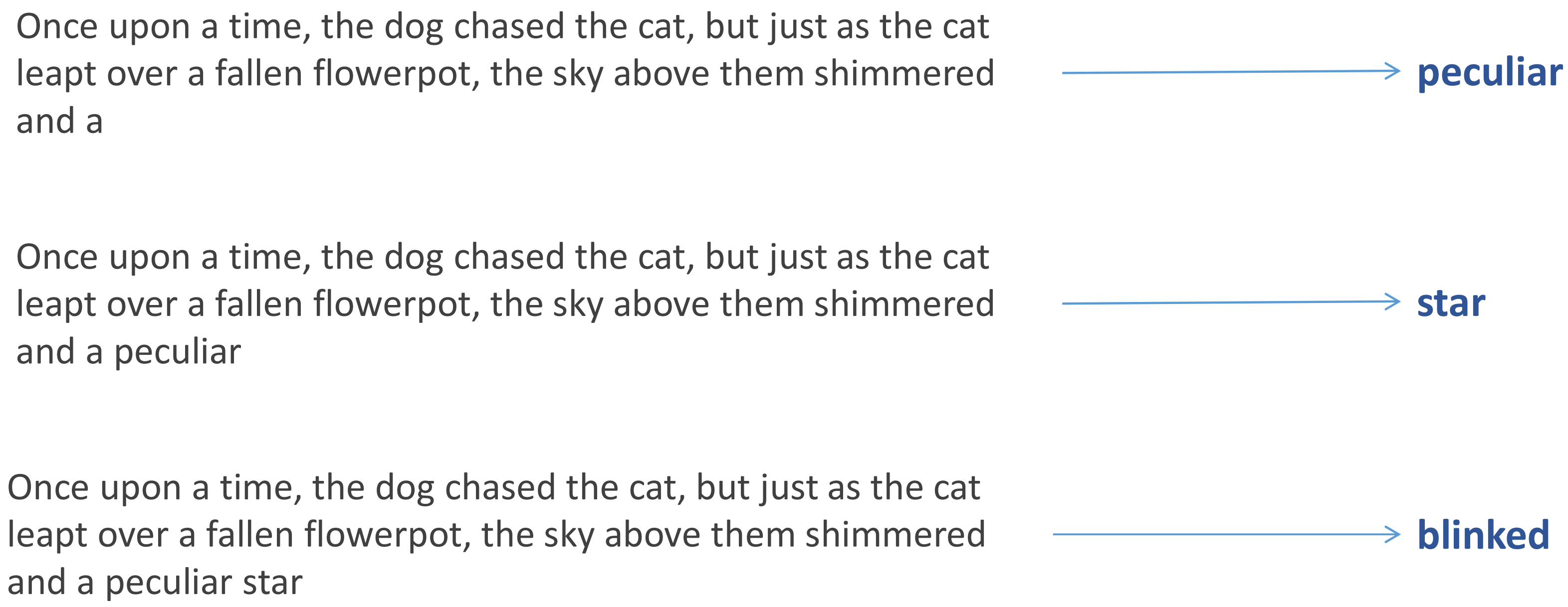
Attention with Next token prediction

We already computed the dot product for these values in the previous step

Since model is **causal**, we don't care about the attention of a token with the next token



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

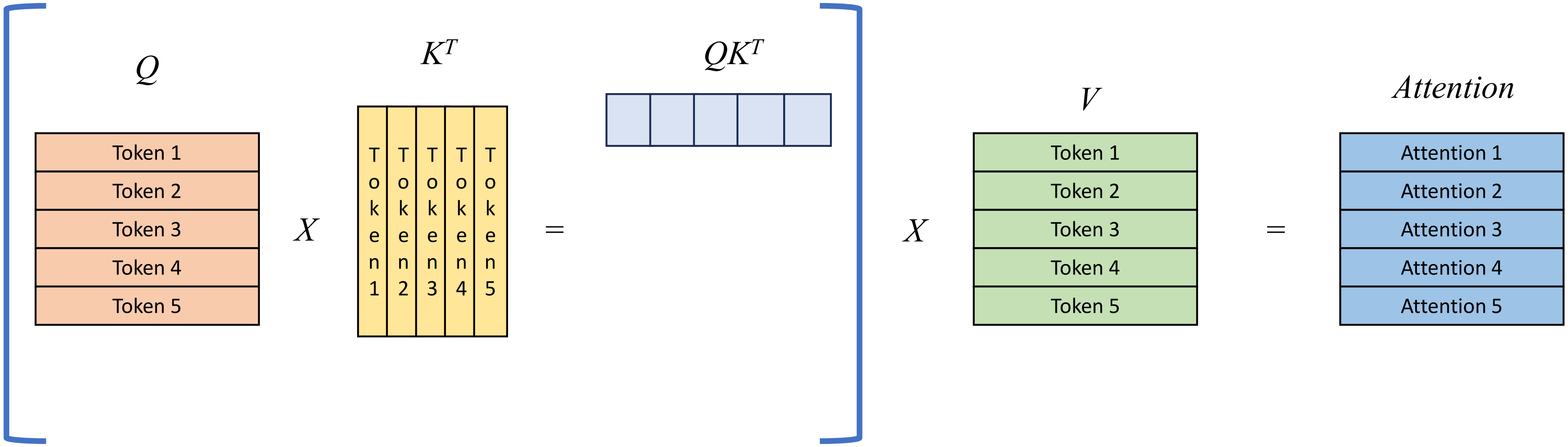


Once upon a time, the dog chased the cat, but just as the cat leapt over a fallen flowerpot, the sky above them shimmered and a → **peculiar**

Once upon a time, the dog chased the cat, but just as the cat leapt over a fallen flowerpot, the sky above them shimmered and a peculiar → **star**

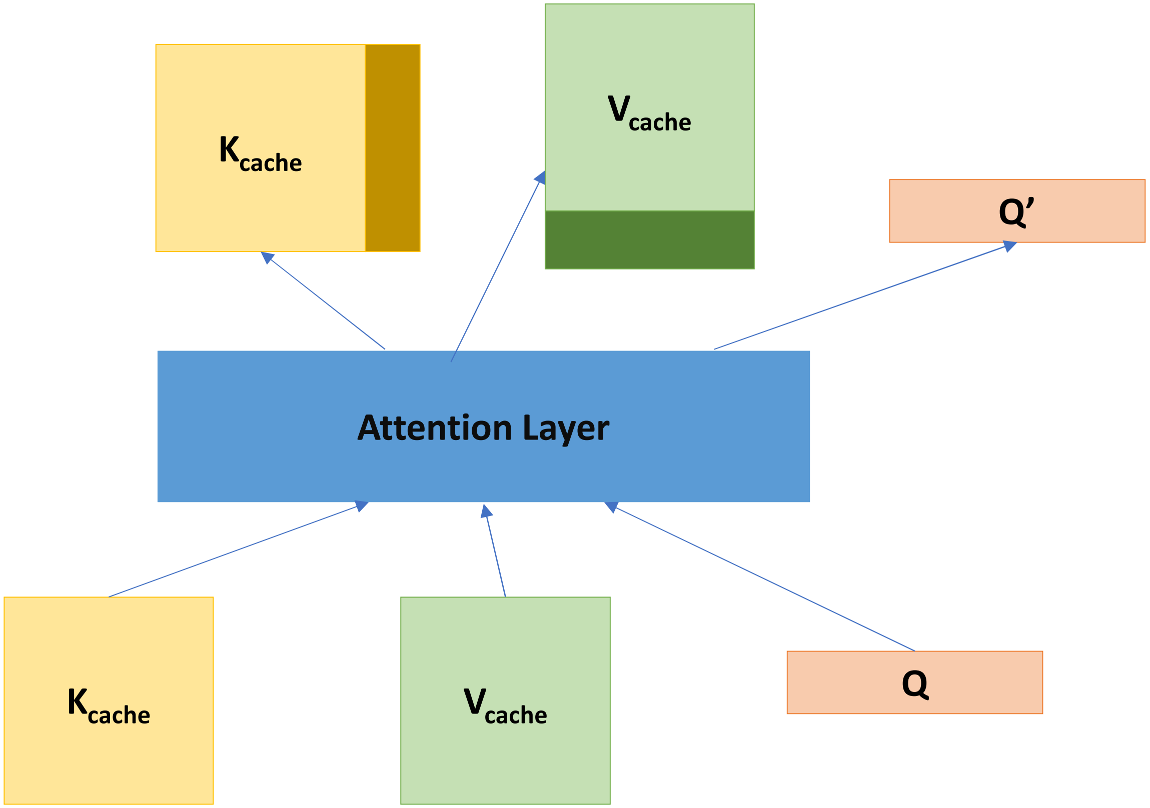
Once upon a time, the dog chased the cat, but just as the cat leapt over a fallen flowerpot, the sky above them shimmered and a peculiar star → **blinked**

KV - Cache

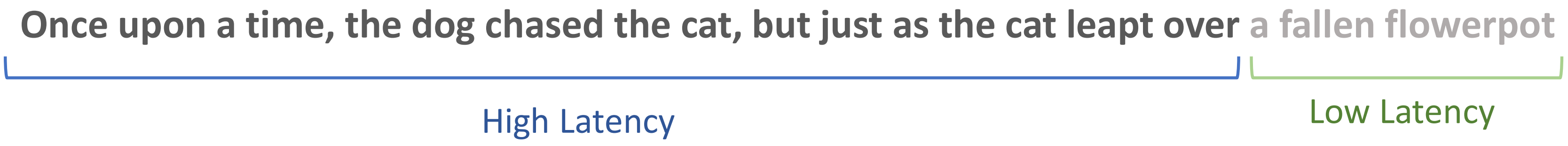


$$Attention\,(Q, K, V) = Softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

KV - Cache



KV - Cache

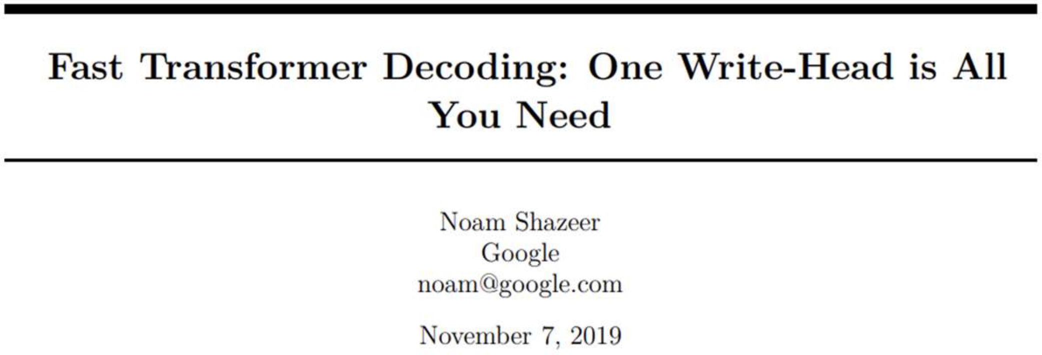


Once upon a time, the dog chased the cat, but just as the cat leapt over a fallen flowerpot

High Latency

Low Latency

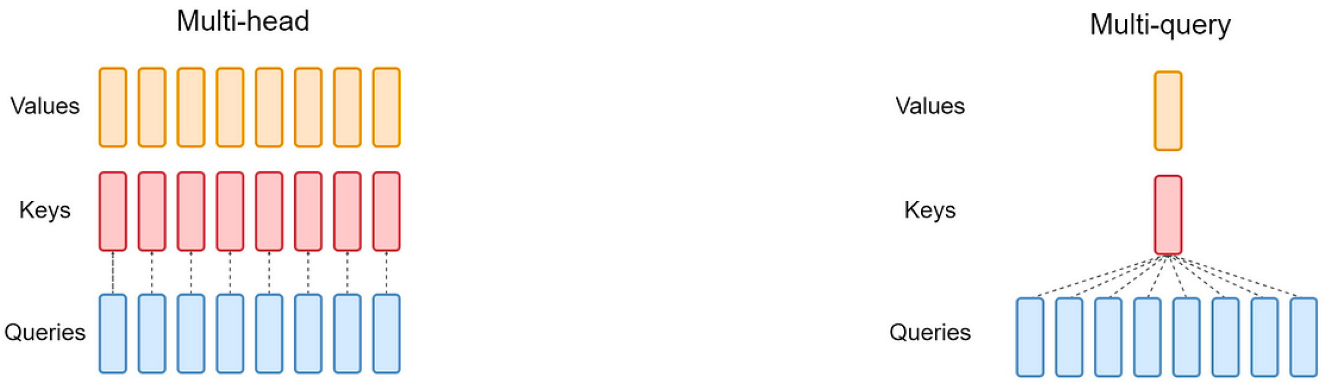
Multi-Query Attention



Fast Transformer Decoding: One Write-Head is All You Need

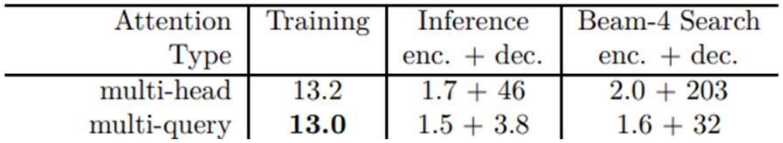
Noam Shazeer
Google
noam@google.com

November 7, 2019

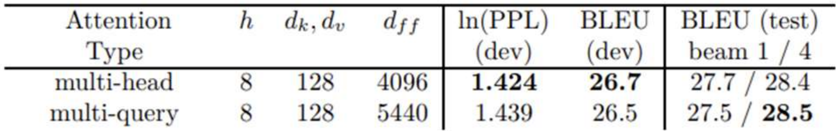


Multi-Query Attention

- **Multi-headed attention (MHA)** gives each “head” its own key- and value-projection, which maximizes representational capacity but incurs a heavy memory-bandwidth cost during decoding.
- **Multi-query attention (MQA)** addresses this by sharing a *single* key and value projection across all H query heads. This reduces the size of the key-value cache and hence the per-step memory bandwidth, by a factor of H, yielding much faster decoding with only minor quality degradation in practice

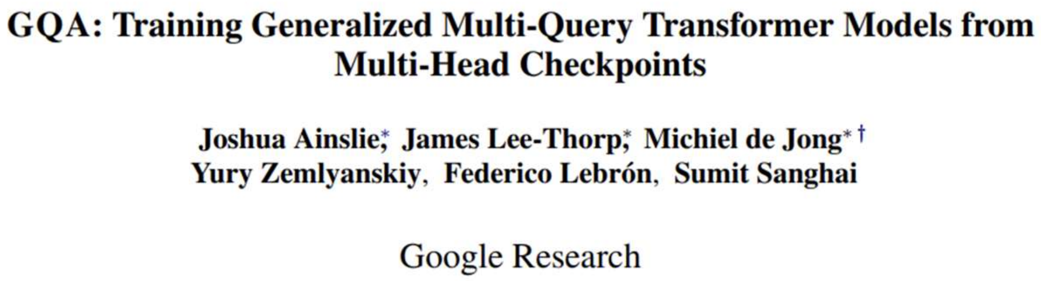


Attention	Training	Inference	Beam-4 Search
Type		enc. + dec.	enc. + dec.
multi-head	13.2	1.7 + 46	2.0 + 203
multi-query	**13.0**	1.5 + 3.8	1.6 + 32



Attention	h	d_k, d_v	d_{ff}	ln(PPL)	BLEU	BLEU (test)
Type				(dev)	(dev)	beam 1 / 4
multi-head	8	128	4096	**1.424**	**26.7**	27.7 / 28.4
multi-query	8	128	5440	1.439	26.5	27.5 / **28.5**

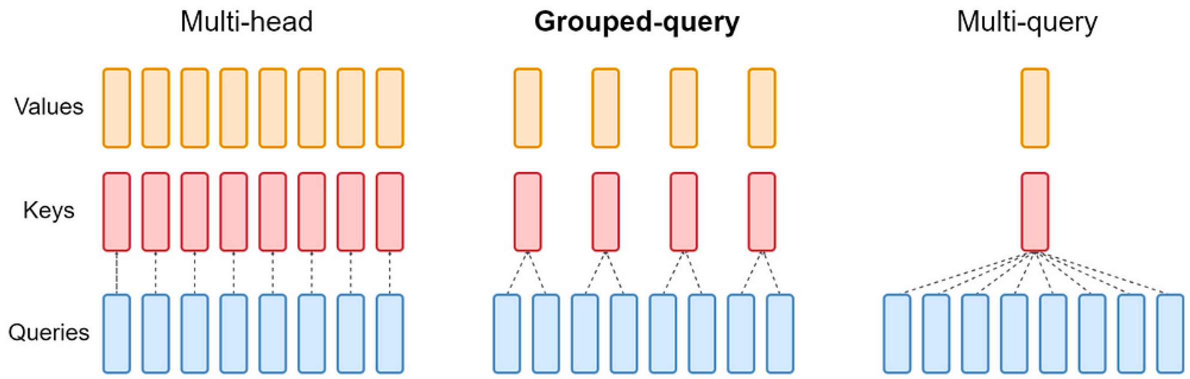
Grouped Multi-Query Attention



GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints

Joshua Ainslie[*], James Lee-Thorp[*], Michiel de Jong[*] [†]
Yury Zemlyanskiy, Federico Lebrón, Sumit Sanghai

Google Research



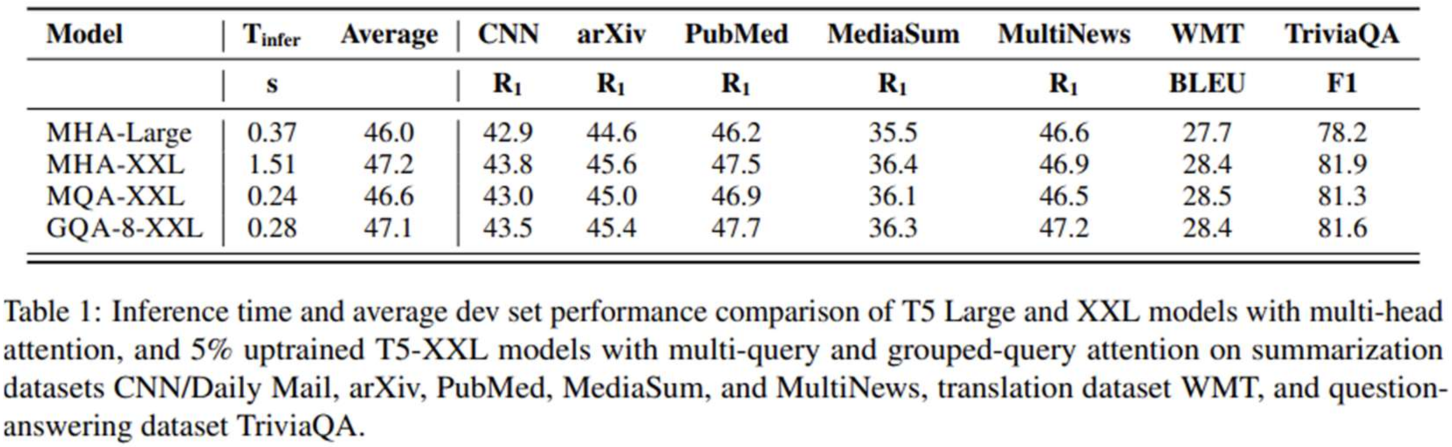
Grouped Multi-Query Attention

- **Grouped-query attention (GQA)** generalizes these two extremes by partitioning the H query heads into G groups (1 ≤ G ≤ H), each with its own shared key/value head. Thus:
- G = H ⇒ equivalent to full **MHA**,
- G = 1 ⇒ equivalent to **MQA**,
- intermediate G gives a **bandwidth–capacity trade-off**: you reduce KV cache size by a factor of G (versus H), while retaining more distinct key/value projections than MQA.

Matrix	MHA	MQA	GQA
W^Q	$d \times d$	$d \times d$	$d \times d$
W^K	$d \times d$	$d \times (d/H)$	$G\,[d \times (d/G)] = d^2$
W^V	$d \times d$	$d \times (d/H)$	$G\,[d \times (d/G)] = d^2$
W^O	$d \times d$	$d \times d$	$d \times d$
Total	$4d^2$	$2d^2 + 2\frac{d^2}{H}$	$4d^2$

Scheme	Cache size	Numeric example
MHA	$2L\,d$	$\approx 16MB$
MQA	$2L\,(d/H)$	$\approx 1MB$
GQA	$2L\,d(G/H)$	$\approx 4MB$

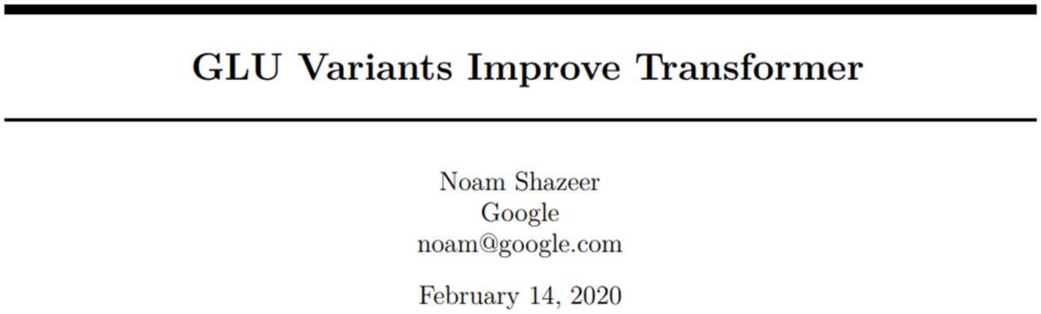
Grouped Multi-Query Attention



Model	T_{infer}	Average	CNN	arXiv	PubMed	MediaSum	MultiNews	WMT	TriviaQA
	s		R_1	R_1	R_1	R_1	R_1	BLEU	F1
MHA-Large	0.37	46.0	42.9	44.6	46.2	35.5	46.6	27.7	78.2
MHA-XXL	1.51	47.2	43.8	45.6	47.5	36.4	46.9	28.4	81.9
MQA-XXL	0.24	46.6	43.0	45.0	46.9	36.1	46.5	28.5	81.3
GQA-8-XXL	0.28	47.1	43.5	45.4	47.7	36.3	47.2	28.4	81.6

Table 1: Inference time and average dev set performance comparison of T5 Large and XXL models with multi-head attention, and 5% uptrained T5-XXL models with multi-query and grouped-query attention on summarization datasets CNN/Daily Mail, arXiv, PubMed, MediaSum, and MultiNews, translation dataset WMT, and question-answering dataset TriviaQA.

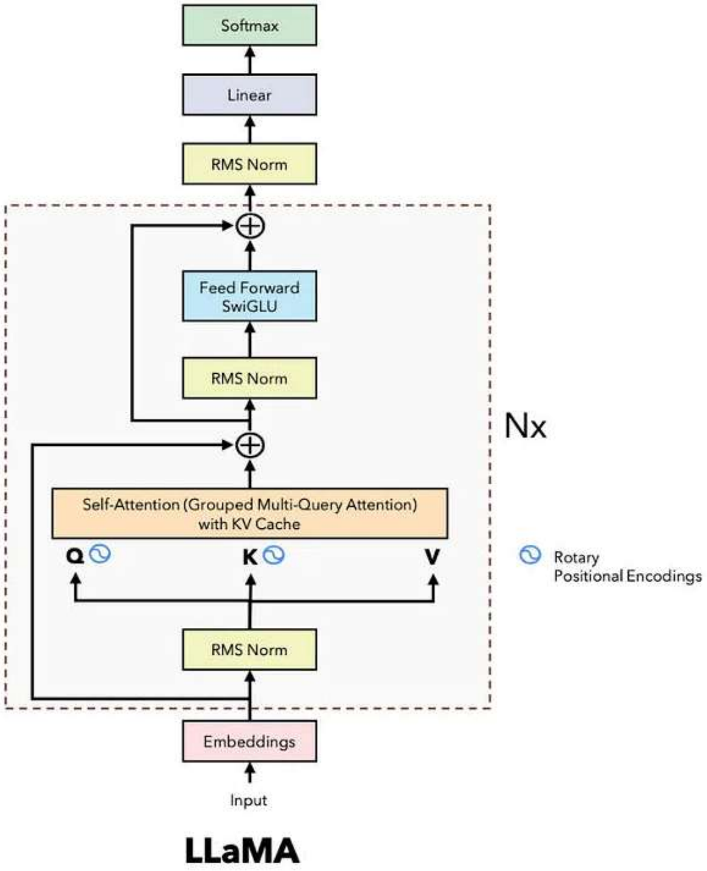
SwiGLU activation



GLU Variants Improve Transformer

Noam Shazeer
Google
noam@google.com

February 14, 2020


Softmax
Linear
RMS Norm
Feed Forward SwiGLU
RMS Norm
Nx
Self-Attention (Grouped Multi-Query Attention) with KV Cache
Q
K
V
Rotary Positional Encodings
RMS Norm
Embeddings
Input
LLaMA

SwiGLU activation

- **Swish** is a smooth, non-monotonic activation defined by:

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

where σ is the sigmoid function and βis a learnable scalar. For negative inputs it suppresses activations toward zero, and for positive inputs it behaves almost linearly, much like ReLU.

- The **Gated Linear Unit (GLU)** splits its input into two parallel affine transforms, one of which passes through a sigmoid gate:

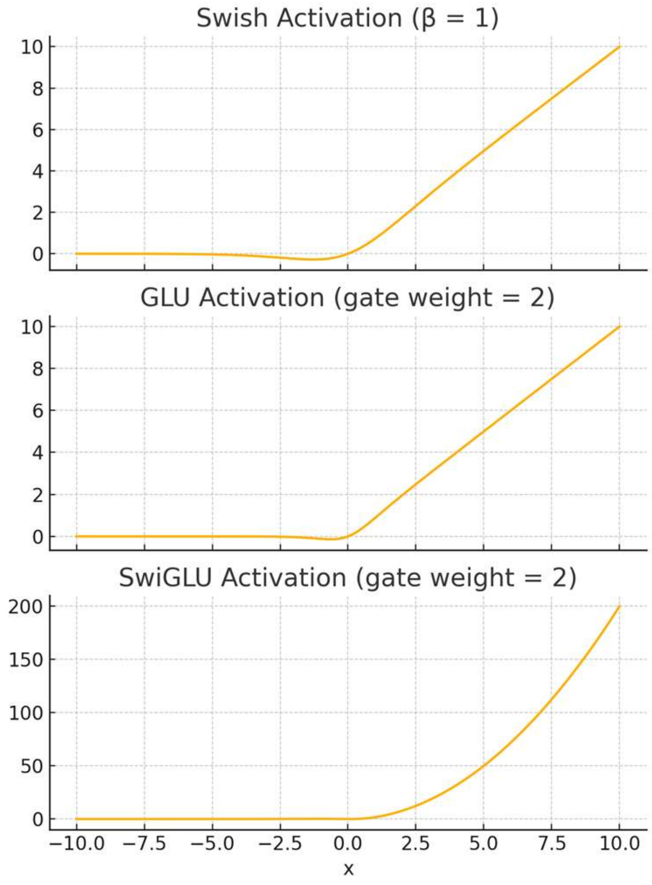
$$GLU(x) = (Wx + b) \otimes \sigma(Vx + c)$$

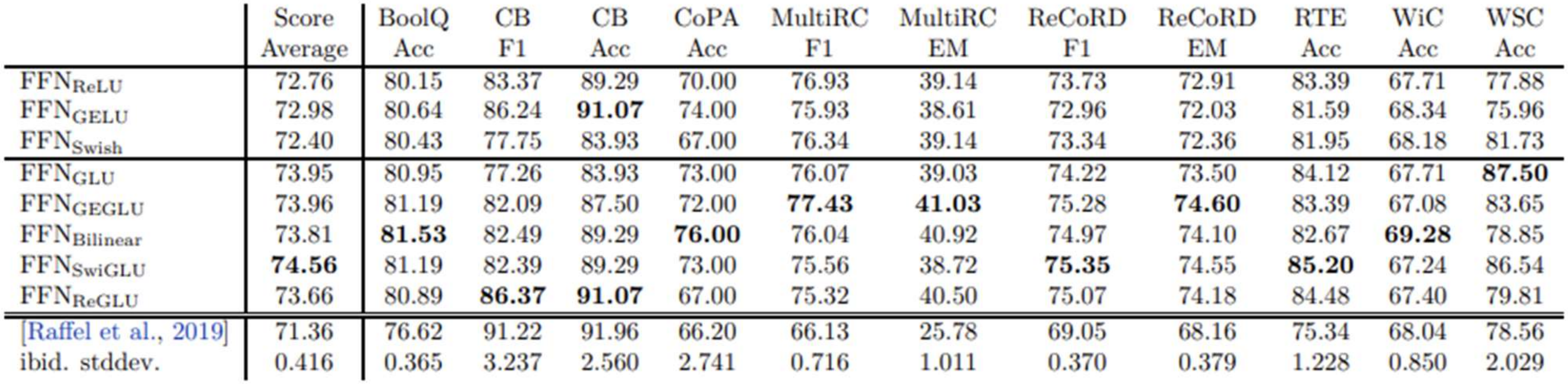
where "⊗" denotes element-wise multiplication.

- **SwiGLU** simply replaces the sigmoid gate in GLU with the Swish function. Concretely, if you split your input and apply two linear projections *U* and *V*, with biases *d* and *e*, then

$$SwiGLU(x) = (Ux + d) \otimes Swish(Vx + c)$$

This combines the smooth gating of Swish with the feature-selective power of GLU.





	Score Average	BoolQ Acc	CB F1	CB Acc	CoPA Acc	MultiRC F1	MultiRC EM	ReCoRD F1	ReCoRD EM	RTE Acc	WiC Acc	WSC Acc
FFN$_{\text{ReLU}}$	72.76	80.15	83.37	89.29	70.00	76.93	39.14	73.73	72.91	83.39	67.71	77.88
FFN$_{\text{GELU}}$	72.98	80.64	86.24	**91.07**	74.00	75.93	38.61	72.96	72.03	81.59	68.34	75.96
FFN$_{\text{Swish}}$	72.40	80.43	77.75	83.93	67.00	76.34	39.14	73.34	72.36	81.95	68.18	81.73
FFN$_{\text{GLU}}$	73.95	80.95	77.26	83.93	73.00	76.07	39.03	74.22	73.50	84.12	67.71	**87.50**
FFN$_{\text{GEGLU}}$	73.96	81.19	82.09	87.50	72.00	**77.43**	**41.03**	75.28	**74.60**	83.39	67.08	83.65
FFN$_{\text{Bilinear}}$	73.81	**81.53**	82.49	89.29	**76.00**	76.04	40.92	74.97	74.10	82.67	**69.28**	78.85
FFN$_{\text{SwiGLU}}$	**74.56**	81.19	82.39	89.29	73.00	75.56	38.72	**75.35**	74.55	**85.20**	67.24	86.54
FFN$_{\text{ReGLU}}$	73.66	80.89	**86.37**	**91.07**	67.00	75.32	40.50	75.07	74.18	84.48	67.40	79.81
[Raffel et al., 2019]	71.36	76.62	91.22	91.96	66.20	66.13	25.78	69.05	68.16	75.34	68.04	78.56
ibid. stddev.	0.416	0.365	3.237	2.560	2.741	0.716	1.011	0.370	0.379	1.228	0.850	2.029

Thank you so much for watching!