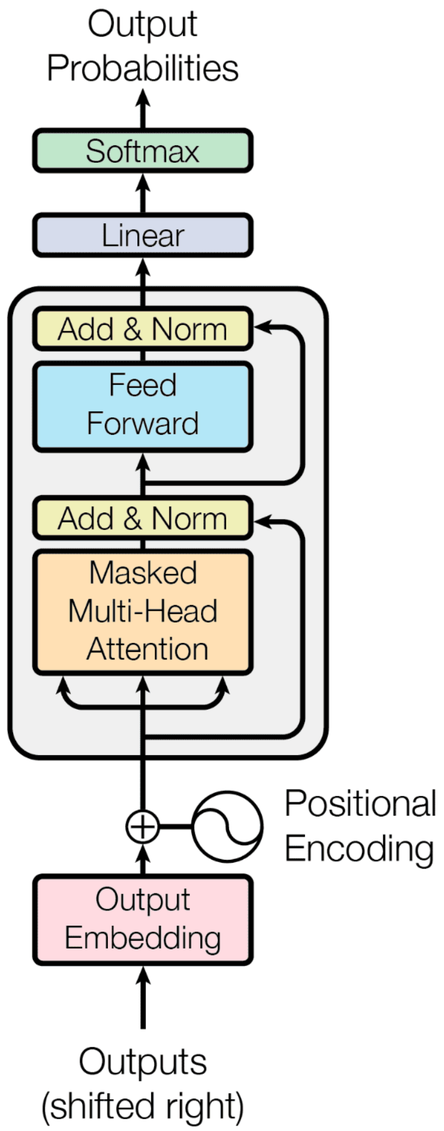
Modern LLMs

Plan

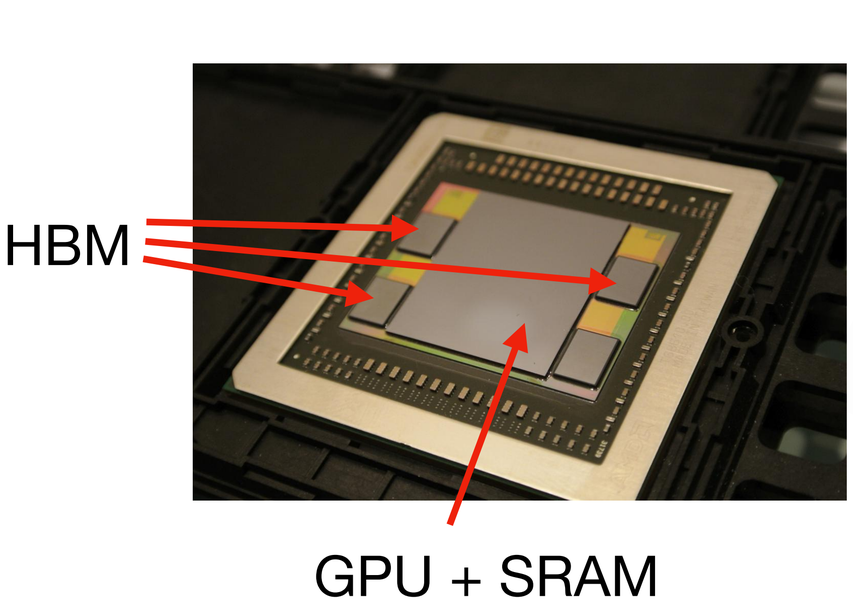
- Attention
- Positional Encoding
- Feed Forward Network
- Normalization

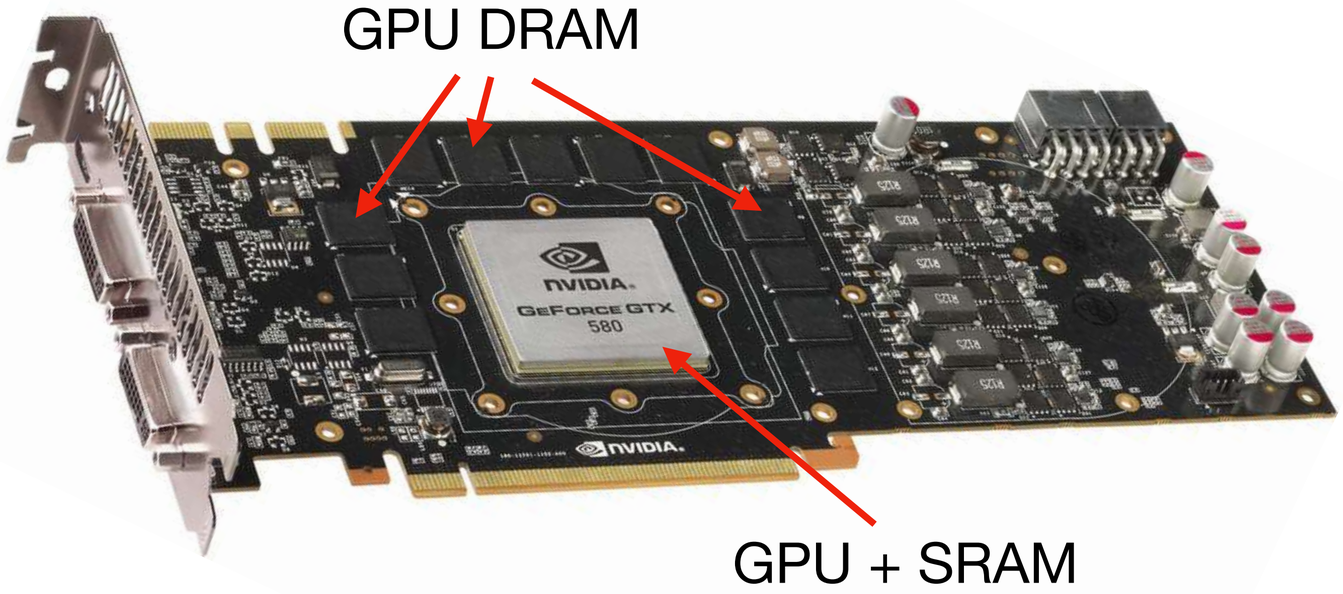


Attention

Memory bottleneck

- GPUs are too fast, speed is limited by memory handling
- There are three types of memory:
  - GPU SRAM (Static Random-Access Memory) - the fastest but smallest memory, embedded in the processor (19 TB/s, 20 MB)
  - GPU HBM (High Bandwidth Memory) - GPU's main memory (1.5 TB/s, 40 GB)
  - CPU DRAM (Dynamic Random-Access Memory) - the slowest (12.8 GB/s, >1 TB)





Vanila attention

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

Algorithm 0 Standard Attention Implementation

Require: Matrices $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{N \times d}$ in HBM.

1: Load $\mathbf{Q}, \mathbf{K}$ by blocks from HBM, compute $\mathbf{S} = \mathbf{Q}\mathbf{K}^\top$, write $\mathbf{S}$ to HBM.
2: Read $\mathbf{S}$ from HBM, compute $\mathbf{P} = \text{softmax}(\mathbf{S})$, write $\mathbf{P}$ to HBM.
3: Load $\mathbf{P}$ and $\mathbf{V}$ by blocks from HBM, compute $\mathbf{O} = \mathbf{P}\mathbf{V}$, write $\mathbf{O}$ to HBM.
4: Return $\mathbf{O}$.

Flash attention

- Flash attention reduces the number of interactions with HBM by performing almost all computations on SRAM
- Attention is counted by blocks that fit in SRAM

$$m(x) := \max_i \quad x_i, \quad f(x) := \begin{bmatrix} e^{x_1 - m(x)} & \dots & e^{x_B - m(x)} \end{bmatrix}, \quad \ell(x) := \sum_i f(x)_i, \quad \text{softmax}(x) := \frac{f(x)}{\ell(x)}.$$

For vectors $x^{(1)}, x^{(2)} \in \mathbb{R}^B$, we can decompose the softmax of the concatenated $x = \begin{bmatrix} x^{(1)} & x^{(2)} \end{bmatrix} \in \mathbb{R}^{2B}$ as:

$$m(x) = m(\begin{bmatrix} x^{(1)} & x^{(2)} \end{bmatrix}) = \max(m(x^{(1)}), m(x^{(2)})), \quad f(x) = \begin{bmatrix} e^{m(x^{(1)}) - m(x)} f(x^{(1)}) & e^{m(x^{(2)}) - m(x)} f(x^{(2)}) \end{bmatrix},$$

$$\ell(x) = \ell(\begin{bmatrix} x^{(1)} & x^{(2)} \end{bmatrix}) = e^{m(x^{(1)}) - m(x)} \ell(x^{(1)}) + e^{m(x^{(2)}) - m(x)} \ell(x^{(2)}), \quad \text{softmax}(x) = \frac{f(x)}{\ell(x)}.$$

https://arxiv.org/abs/2205.14135

Flash attention

Algorithm 1 FlashAttention

Require: Matrices $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{N \times d}$ in HBM, on-chip SRAM of size M.

Set block sizes $B_c = \lceil \frac{M}{4d} \rceil$, $B_r = \min\left(\lceil \frac{M}{4d} \rceil, d\right)$.

Initialize $\mathbf{O} = (0)_{N \times d} \in \mathbb{R}^{N \times d}$, $\ell = (0)_N \in \mathbb{R}^N$, $m = (-\infty)_N \in \mathbb{R}^N$ in HBM.

Divide $\mathbf{Q}$ into $T_r = \left\lceil \frac{N}{B_r} \right\rceil$ blocks $\mathbf{Q}_1, \ldots, \mathbf{Q}_{T_r}$ of size $B_r \times d$ each, and divide $\mathbf{K}, \mathbf{V}$ in to $T_c = \left\lceil \frac{N}{B_c} \right\rceil$ blocks $\mathbf{K}_1, \ldots, \mathbf{K}_{T_c}$ and $\mathbf{V}_1, \ldots, \mathbf{V}_{T_c}$, of size $B_c \times d$ each.

Divide $\mathbf{O}$ into T_r blocks $\mathbf{O}_i, \ldots, \mathbf{O}_{T_r}$ of size $B_r \times d$ each, divide ℓ into T_r blocks $\ell_i, \ldots, \ell_{T_r}$ of size B_r each, divide m into T_r blocks $m_1, \ldots, m_{T_r}$ of size B_r each.

for $1 \le j \le T_c$ **do**

  Load $\mathbf{K}_j, \mathbf{V}_j$ from HBM to on-chip SRAM.

  for $1 \le i \le T_r$ **do**

    Load $\mathbf{Q}_i, \mathbf{O}_i, \ell_i, m_i$ from HBM to on-chip SRAM.

    On chip, compute $\mathbf{S}_{ij} = \mathbf{Q}_i \mathbf{K}_j^T \in \mathbb{R}^{B_r \times B_c}$.

    On chip, compute $\tilde{m}_{ij} = \text{rowmax}(\mathbf{S}_{ij}) \in \mathbb{R}^{B_r}$, $\tilde{\mathbf{P}}_{ij} = \exp(\mathbf{S}_{ij} - \tilde{m}_{ij}) \in \mathbb{R}^{B_r \times B_c}$ (pointwise), $\tilde{\ell}_{ij} = \text{rowsum}(\tilde{\mathbf{P}}_{ij}) \in \mathbb{R}^{B_r}$.

    On chip, compute $m_i^{\text{new}} = \max(m_i, \tilde{m}_{ij}) \in \mathbb{R}^{B_r}$, $\ell_i^{\text{new}} = e^{m_i - m_i^{\text{new}}} \ell_i + e^{\tilde{m}_{ij} - m_i^{\text{new}}} \tilde{\ell}_{ij} \in \mathbb{R}^{B_r}$.

    Write $\mathbf{O}_i \leftarrow \text{diag}(\ell_i^{\text{new}})^{-1}(\text{diag}(\ell_i) e^{m_i - m_i^{\text{new}}} \mathbf{O}_i + e^{\tilde{m}_{ij} - m_i^{\text{new}}} \tilde{\mathbf{P}}_{ij} \mathbf{V}_j)$ to HBM.

    Write $\ell_i \leftarrow \ell_i^{\text{new}}$, $m_i \leftarrow m_i^{\text{new}}$ to HBM.

  end for

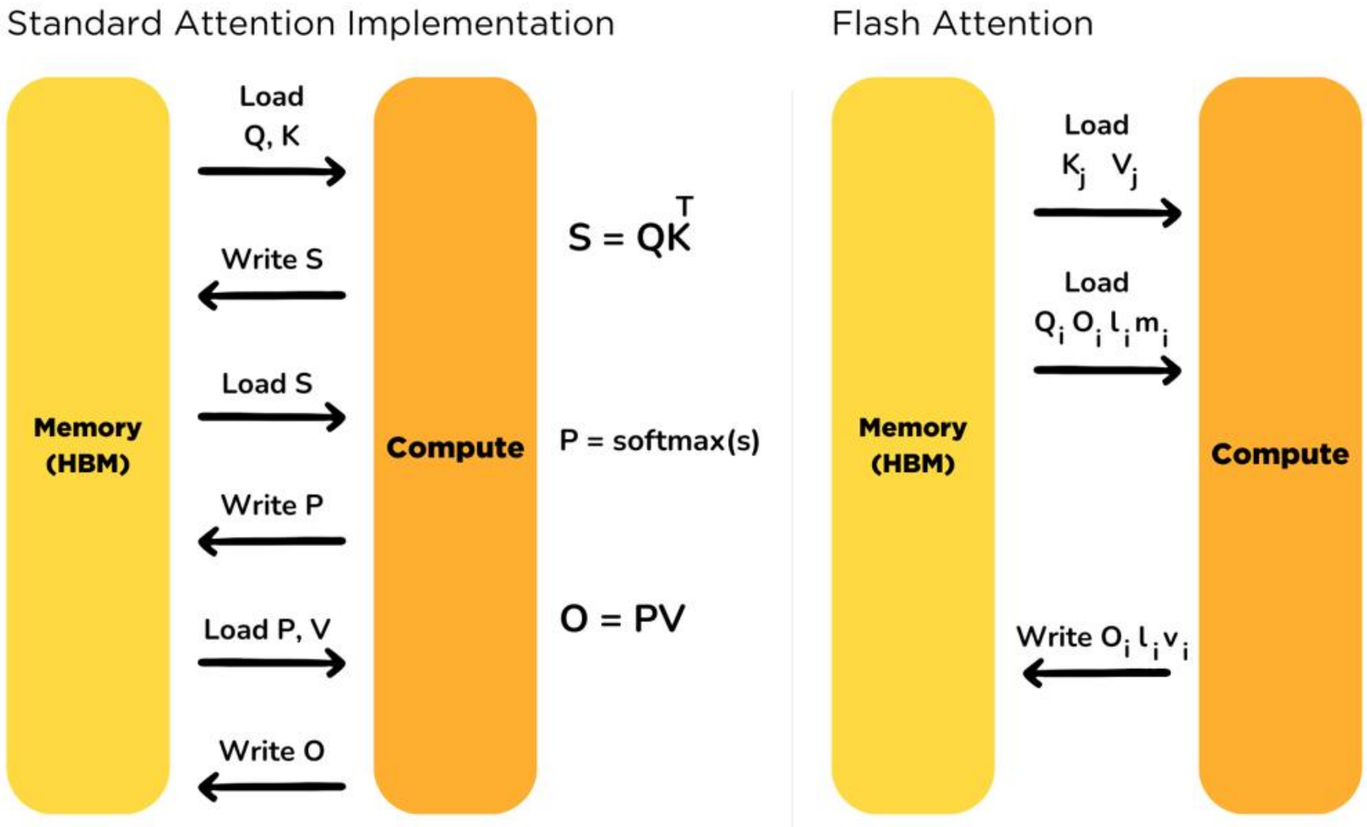
end for

Return $\mathbf{O}$.

https://arxiv.org/abs/2205.14135

Flash attention

- Flash attention reduces the number of interactions with HBM by performing almost all computations on SRAM
- Attention is counted by blocks that fit in SRAM
- O - output matrix, l - softmax normalization constant, m - maximum value of attention rate



https://huggingface.co/docs/text-generation-inference/conceptual/flash_attention

Flash attention

Flash attention increases the number of operations, but significantly (6-9 times) reduces the computation time due to reduced accesses to HBM

Attention	Standard	FlashAttention
GFLOPs	66.6	75.2
HBM R/W (GB)	40.3	4.4
Runtime (ms)	41.7	7.3

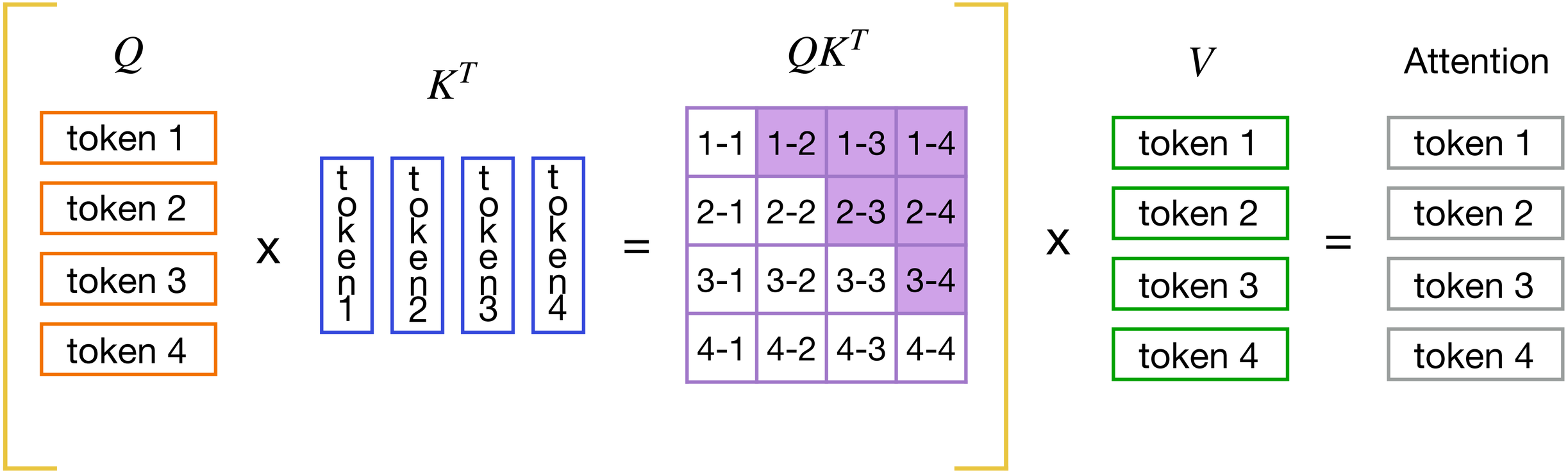
Text generation

- All autoregressive models generate text one token at a time
- At each iteration, you must run the entire sequence through the model
- One attention count takes $O(l^2hd + lh^2d^2)$ operations

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

Text generation

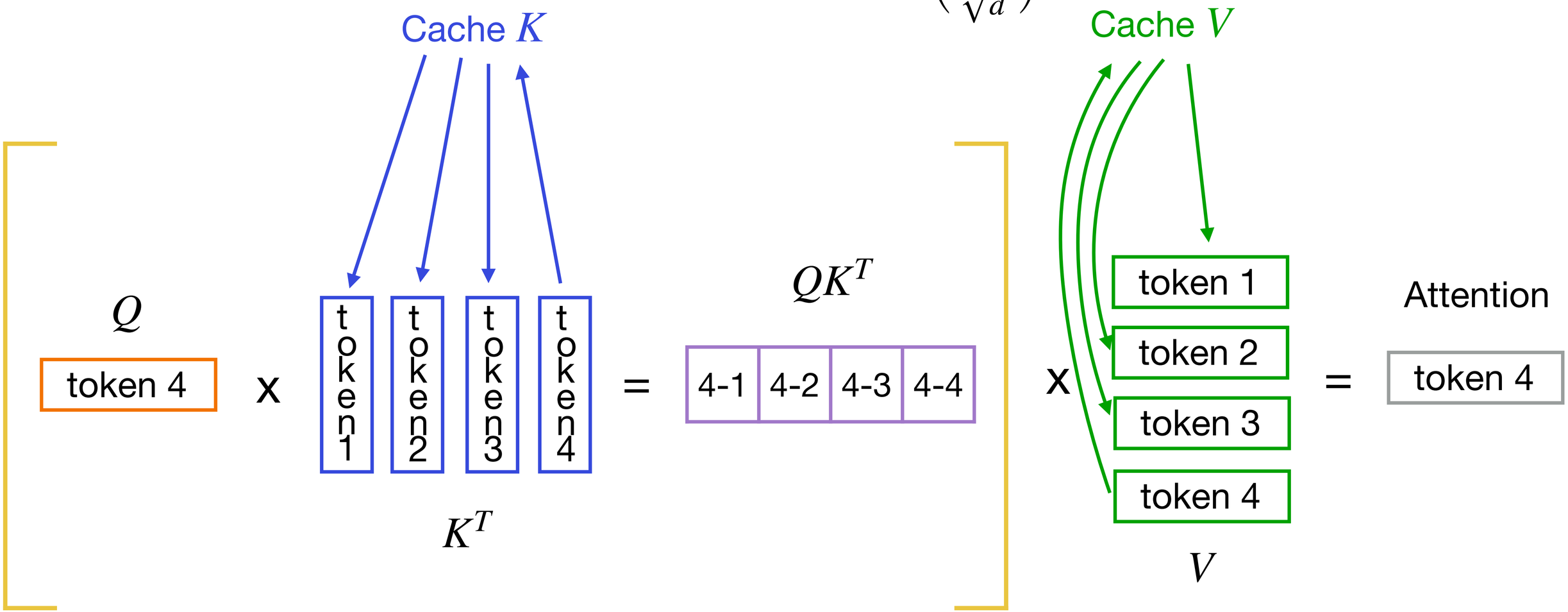
$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right)V$$



We recalculate the values of Q, K and V, even though they do not change!
Each token depends only on the ones before it

KV caching

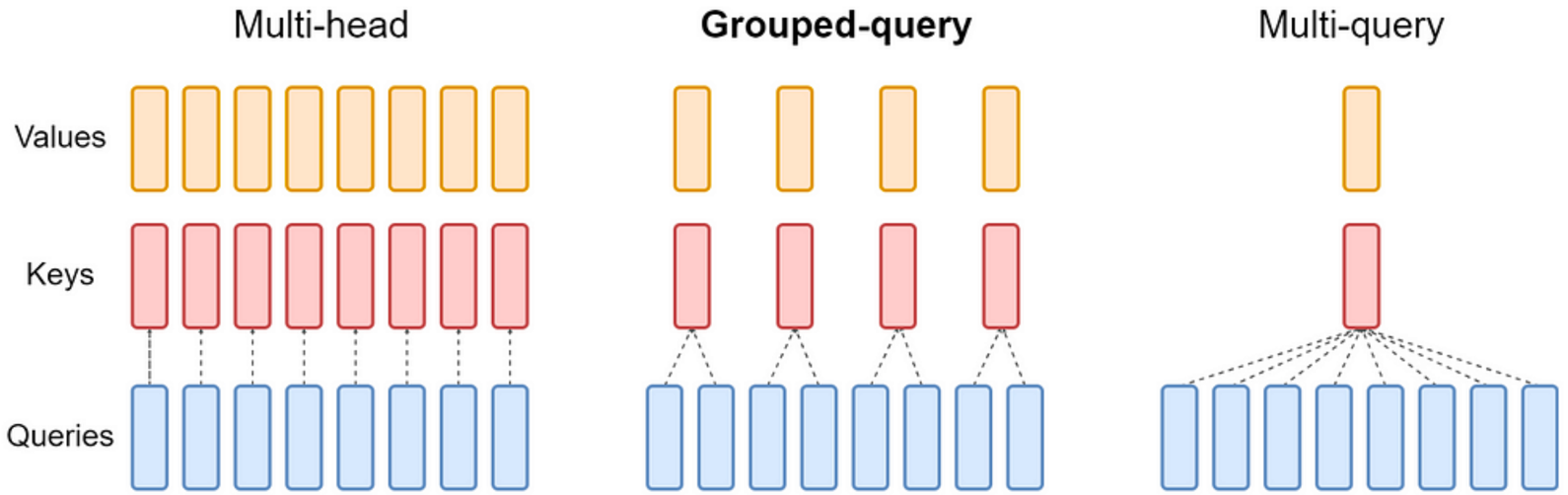
$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right)V$$



Each iteration costs $O(lhd + hd^2)$ instead of $O(l^2hd + lh^2d^2)$!

Multi-Query Attention

- KV caching requires storing the outputs of all layers
- Generation speed drops due to memory handling
- You can leave one head for K and V
- Usually (Llama2, Falcon, Mistral, PaLM) choose the middle option and keep about 8 heads



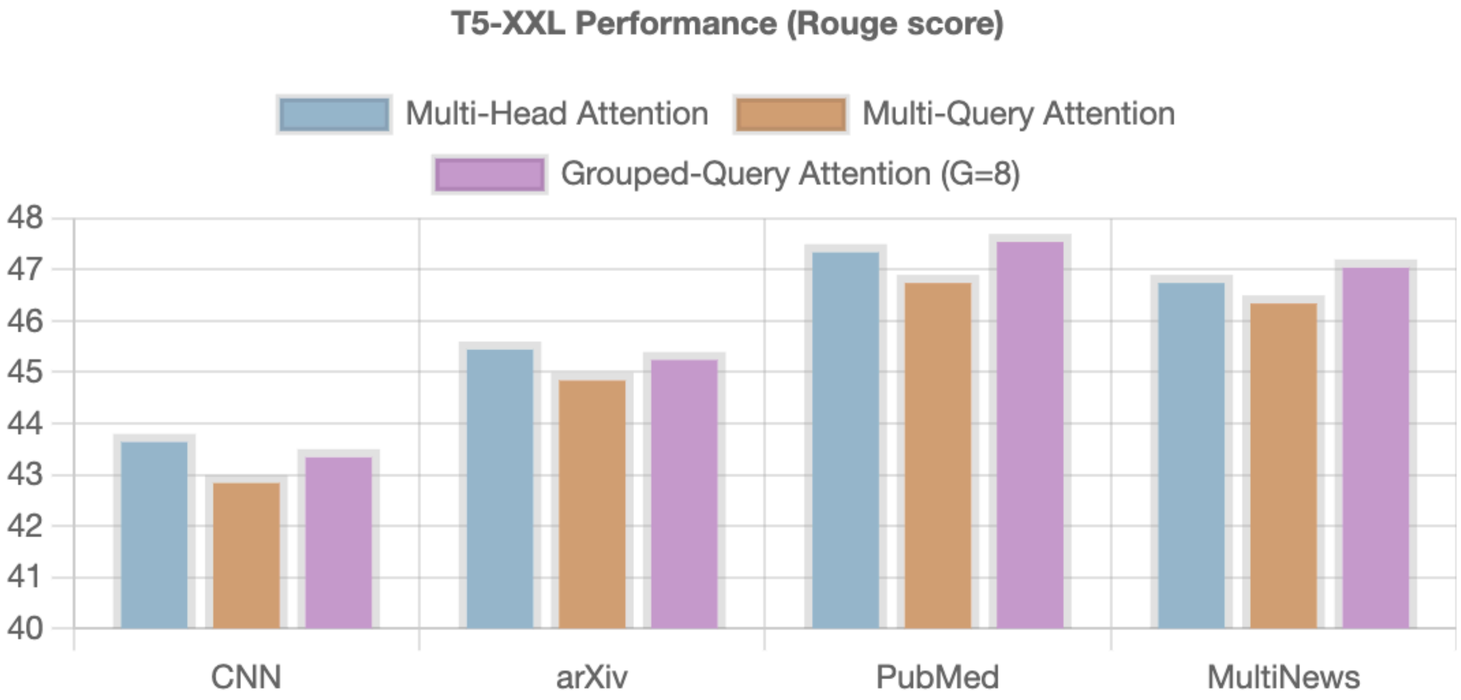
https://arxiv.org/pdf/1911.02150v1

Multi-Query Attention

Attention Type	h	d_k, d_v	d_{ff}	ln(PPL) (dev)	BLEU (dev)	BLEU (test) 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**
multi-head local	8	128	4096	1.427	26.6	27.5 / 28.3
multi-query local	8	128	5440	1.437	26.5	27.6 / 28.2

Attention Type	Training	Inference enc. + dec.	Beam-4 Search enc. + dec.
multi-head	13.2	1.7 + 46	2.0 + 203
multi-query	**13.0**	1.5 + 3.8	1.6 + 32
multi-head local	13.2	1.7 + 23	1.9 + 47
multi-query local	**13.0**	**1.5 + 3.3**	**1.6 + 16**

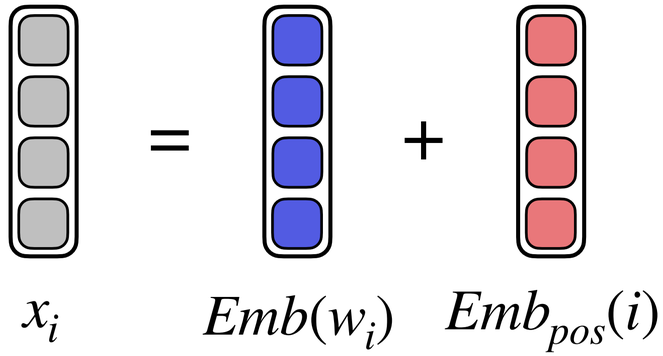
Multi-Query Attention



Positional Encodings

Absolute positional encodings

- Relative distance is not taken into account
- The model can't be applied to longer texts
- It's essential to save position information in all intermediate layers

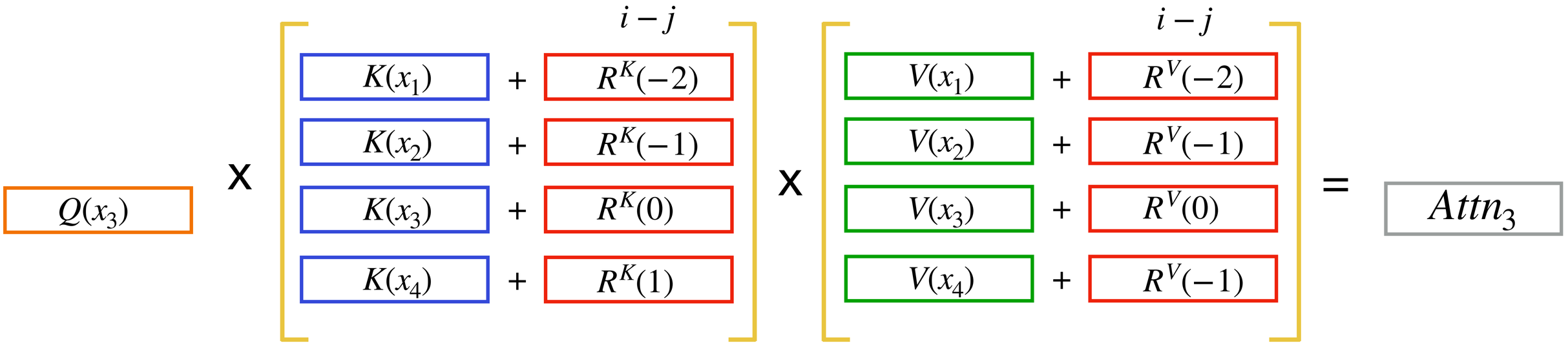


Relative Position Encodings (RPE)

- Discard absolute positional encodings
- Explicitly introduce information about the relative position of tokens to the attention mechanism
- RPE allows to use the model with longer texts
- Used in T5, ALBERT, DeBERTa, Longformer, etc.

Relative Position Encodings (RPE)

$$Attn_i = softmax\left(\frac{Q_i^T(K^T + R_i^K)}{\sqrt{d}}\right)(V + R_i^V)$$



Relative Position Encodings (RPE)

- During implementation, a matrix of relative positions is generated and embeddings are found from it
- To be able to adapt the model to longer texts one can limit the maximum distance between tokens

$$Attn_i = softmax\left(\frac{Q_i^T(K^T + R_i^K)}{\sqrt{d}}\right)(V + R_i^V)$$

$$P = \begin{pmatrix} 0 & 1 & 2 & 2 & 2 \\ -1 & 0 & 1 & 2 & 2 \\ -2 & -1 & 0 & 1 & 2 \\ -2 & -2 & -1 & 0 & 1 \\ -2 & -2 & -2 & -1 & 0 \end{pmatrix}$$

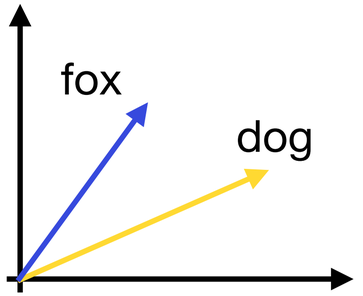
$$R^K = Emb_K(P) \in \mathbb{R}^{[n \times n \times d]}$$

$$R^V = Emb_V(P) \in \mathbb{R}^{[n \times n \times d]}$$

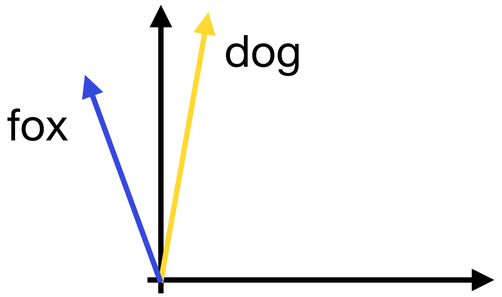
Rotary Position Embeddings (RoPE)

- The vectors q and k are rotated by the angle $i\theta$, where i is the position in the text
- Thus, the relative distance does not change when the position changes
- Vectors for similar positions will be rotated by a similar angle, distant vectors will be rotated by different angles

fox jumps over the **dog**



quick brown **fox** jumps over the **dog**



Formal definition of RoPE

The vector rotation operation is performed by multiplying a vector by the rotation matrix

$$Q_i^r = \begin{pmatrix} \cos i\theta & -\sin i\theta \\ \sin i\theta & \cos i\theta \end{pmatrix} Q_i \qquad K_j^r = \begin{pmatrix} \cos j\theta & -\sin j\theta \\ \sin j\theta & \cos j\theta \end{pmatrix} K_j$$

Attention is calculated in the same way as before, but with rotated matrices Q and K

$$Attn = softmax\left(\frac{Q^r K^{rT}}{\sqrt{d}}\right) V$$

Multidimensional RoPE

- Rotation matrix is replaced by a block-diagonal matrix
- Each block is a rotation matrix with a particular angle

$$R_j^d = \begin{pmatrix} \cos j\theta_1 & -\sin j\theta_1 & 0 & 0 & \cdots & 0 & 0 \\ \sin j\theta_1 & \cos j\theta_1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \cos j\theta_2 & -\sin j\theta_2 & \cdots & 0 & 0 \\ 0 & 0 & \sin j\theta_2 & \cos j\theta_2 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & \cos j\theta_{d/2} & -\sin j\theta_{d/2} \\ 0 & 0 & 0 & 0 & \cdots & \sin j\theta_{d/2} & \cos j\theta_{d/2} \end{pmatrix},$$

$$Q_i^{rT} K_j^r = (R_i^d Q_i)^T (R_j^d K_j) = Q_i^T R_i^{dT} R_j^d K_j = Q_i^T R_{j-i}^d K_j$$

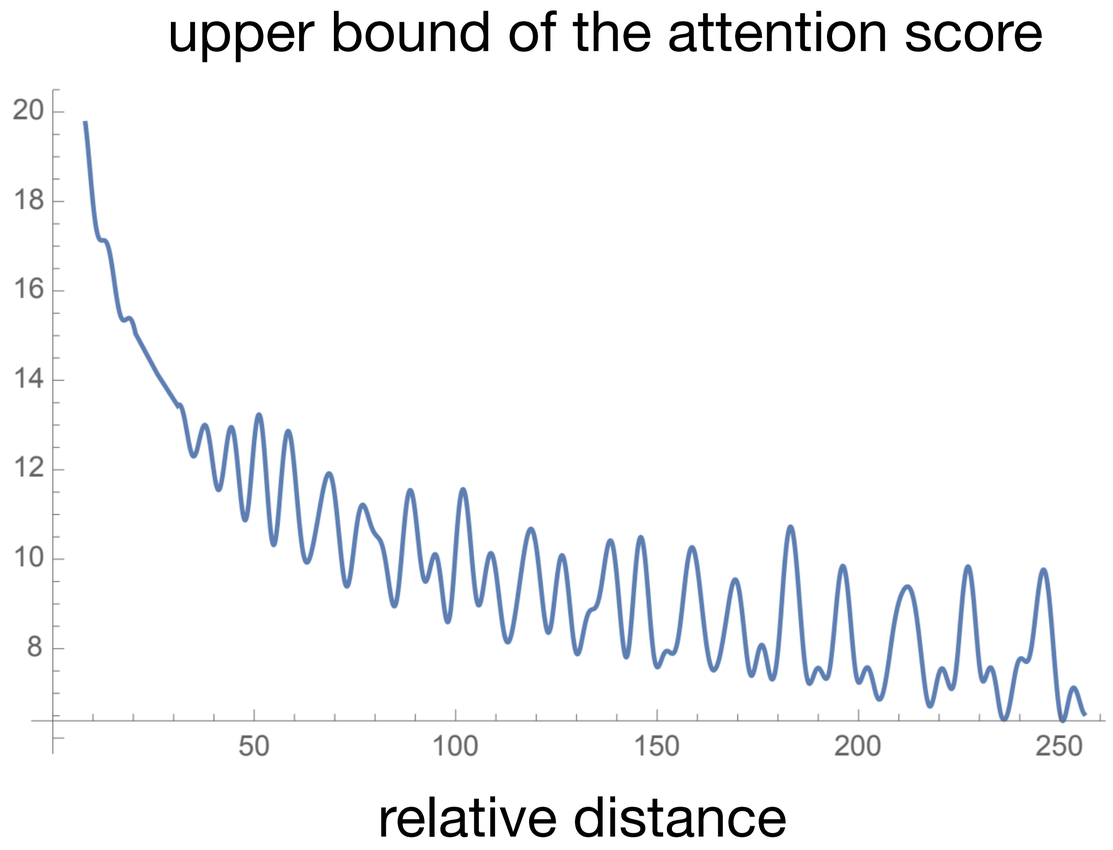
where $\theta_i = 10000^{-2(i-1)/d}$. The value 10000 is chosen empirically.

Each angle specifies a different wave frequency

- Coordinates with **higher** frequency store **local** position information
- Coordinates with **lower** frequency store **global** position information

RoPE justification

We can theoretically find the upper bound of the attention between distant tokens
The greater the distance between tokens, the smaller the attention value between them



RoPE efficiency

Taking in account the sparsity of the matrix R_j^d, we can improve the efficiency of multiplication

$$\begin{pmatrix} \cos j\theta & -\sin j\theta \\ \sin j\theta & \cos j\theta \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} x_1 \cos j\theta - x_2 \sin j\theta \\ x_2 \cos j\theta + x_1 \sin j\theta \end{pmatrix}$$

$$R_j^d x = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ \vdots \\ x_{d-1} \\ x_d \end{pmatrix} \otimes \begin{pmatrix} \cos j\theta_1 \\ \cos j\theta_1 \\ \cos j\theta_2 \\ \cos j\theta_2 \\ \vdots \\ \cos j\theta_{d/2} \\ \cos j\theta_{d/2} \end{pmatrix} + \begin{pmatrix} -x_2 \\ x_1 \\ -x_4 \\ x_3 \\ \vdots \\ -x_d \\ x_{d-1} \end{pmatrix} \otimes \begin{pmatrix} \sin j\theta_1 \\ \sin j\theta_1 \\ \sin j\theta_2 \\ \sin j\theta_2 \\ \vdots \\ \sin j\theta_{d/2} \\ \sin j\theta_{d/2} \end{pmatrix}$$

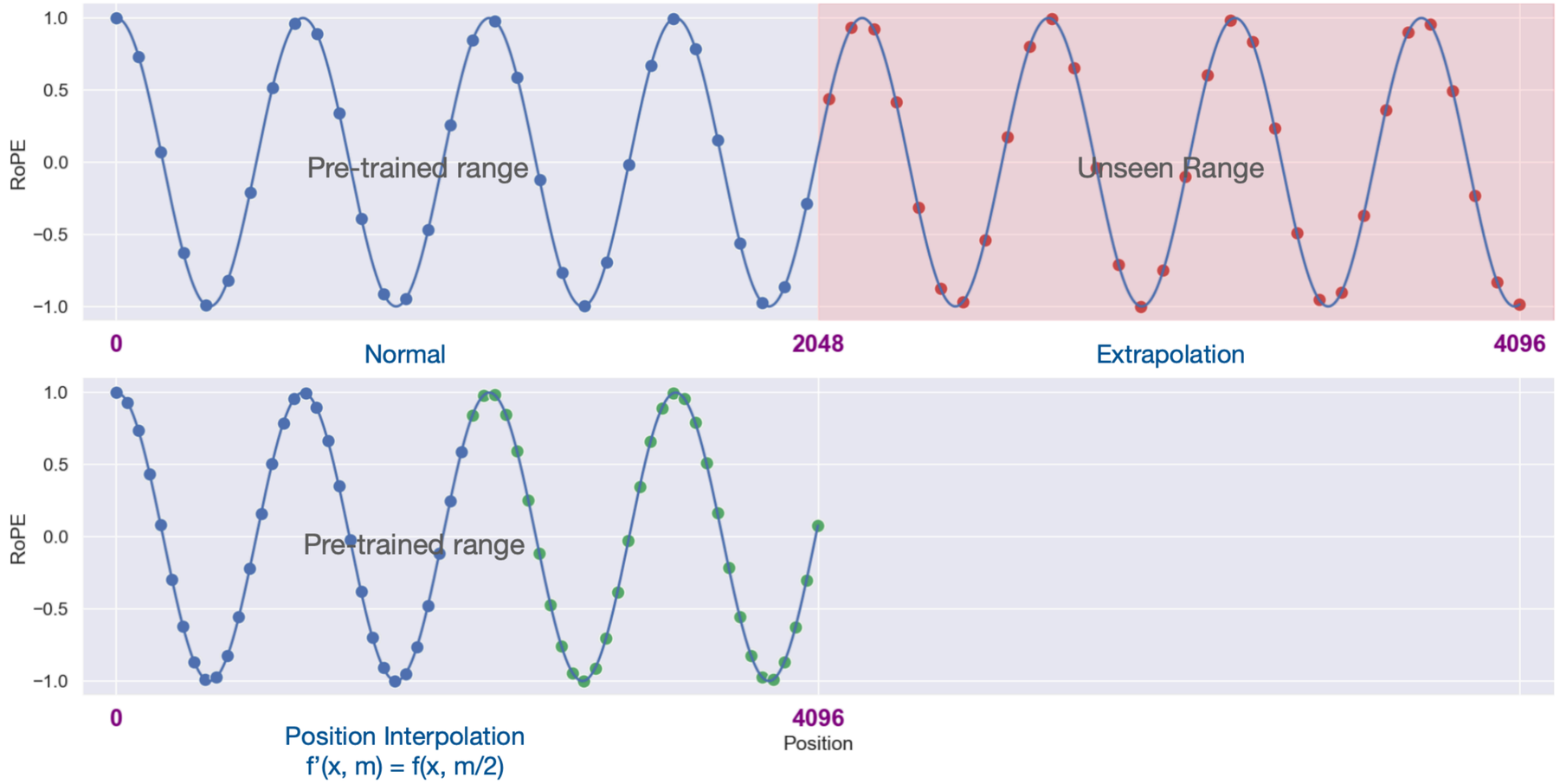
Now multiplication costs $O(d)$ instead of $O(d^2)$.

RoPE: Outcomes

- RoPE allows you to take the relative location of tokens into account
- RoPE **does not** add trainable parameters
- RoPE is very computationally efficient
- RoPE is very popular, used in Llama, Mistral, PaLM, GPT-4, Falcon, Qwen2, etc.
- RoPE does not work well with increasing text length

RoPE position interpolation

Extrapolation doesn't work for RoPE, but interpolation works great



RoPE position interpolation

FT – fine-tuning with extrapolation
PI – fine-tuning with interpolation

Model			Evaluation Context Window Size				
Size	Context Window	Method	2048	4096	8192	16384	32768
7B	2048	None	7.20	$> 10^3$	$> 10^3$	$> 10^3$	$> 10^3$
7B	8192	FT	7.21	7.34	7.69	-	-
7B	8192	PI	7.13	6.96	6.95	-	-
7B	16384	PI	7.11	6.93	6.82	6.83	-
7B	32768	PI	7.23	7.04	6.91	6.80	6.77
13B	2048	None	6.59	-	-	-	-
13B	8192	FT	6.56	6.57	6.69	-	-
13B	8192	PI	6.55	6.42	6.42	-	-
13B	16384	PI	6.56	6.42	6.31	6.32	-
13B	32768	PI	6.54	6.40	6.28	6.18	6.09
33B	2048	None	5.82	-	-	-	-
33B	8192	FT	5.88	5.99	6.21	-	-
33B	8192	PI	5.82	5.69	5.71	-	-
33B	16384	PI	5.87	5.74	5.67	5.68	-
65B	2048	None	5.49	-	-	-	-
65B	8192	PI	5.42	5.32	5.37	-	-

Attention with Linear Biases (ALiBi)

- Add a negative shift to each attention value
- The greater the difference between the positions, the less attention remains

$$Attn = softmax\left(\frac{QK^T + M}{\sqrt{d}}\right)V$$

Vanila Attention

$$M_{ij} = \begin{cases} 0, & i \leq j \\ -\infty, & i > j \end{cases}$$

ALiBi

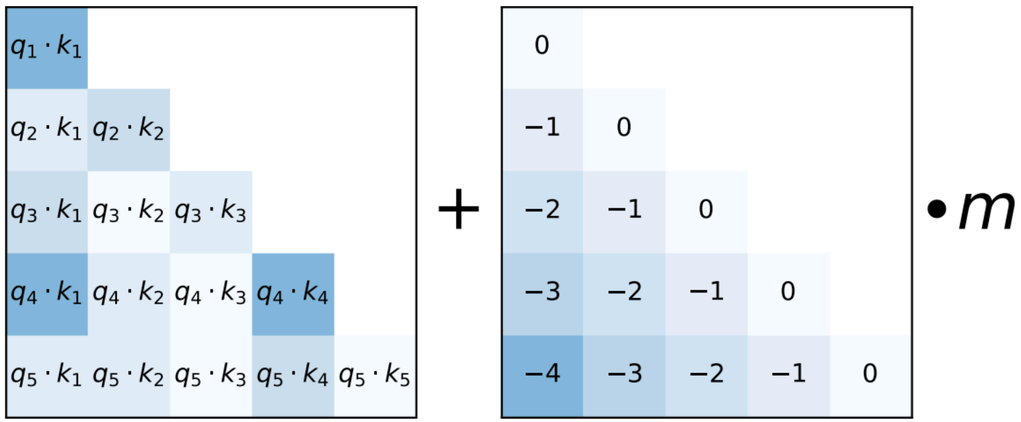
$$M_{ij}^h = \begin{cases} m_h(i-j), & i \leq j \\ -\infty, & i > j \end{cases}$$

Each head has its own $m_h = 2^{-\frac{h}{2}}$

Attention with Linear Biases (ALiBi)

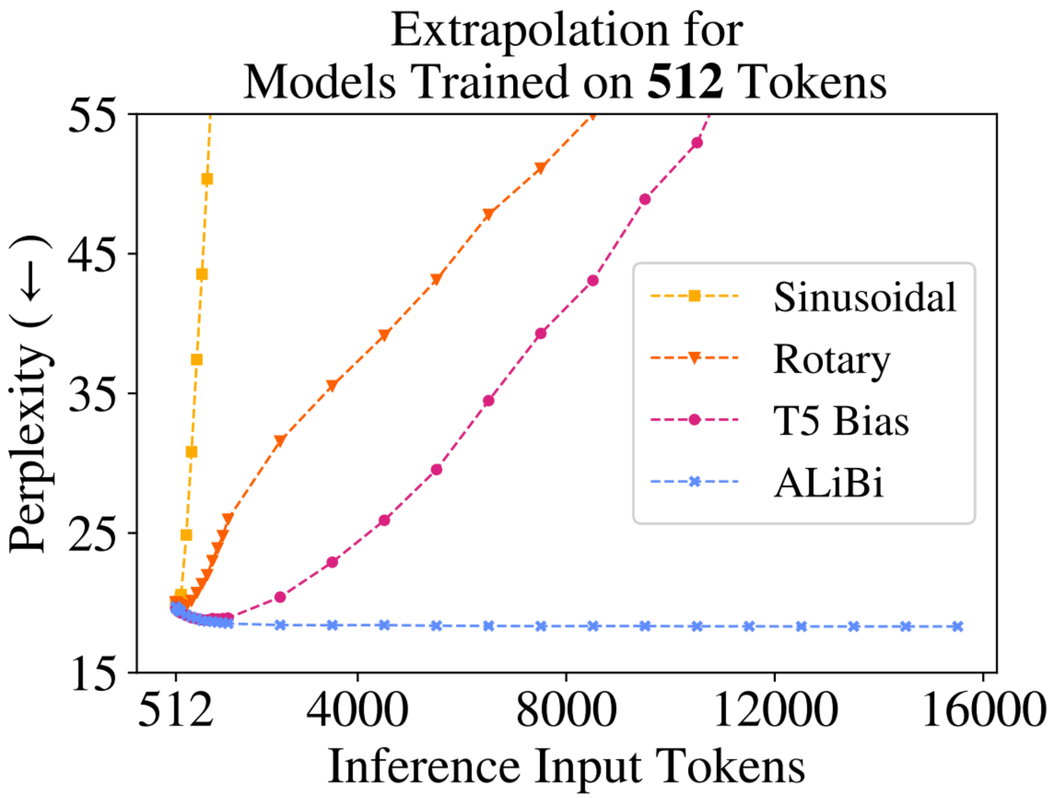
- The coefficient m is needed so that some heads look at local information and others look at global information

- ALiBi does not complicate the model because it does not add training parameters or additional operations



ALiBi: Outcomes

- ALiBi does not complicate the model as it does not add training parameters nor additional operations
- ALiBi adapts very well to increasing text lengths
- ALiBi is used in BLOOM, Claude



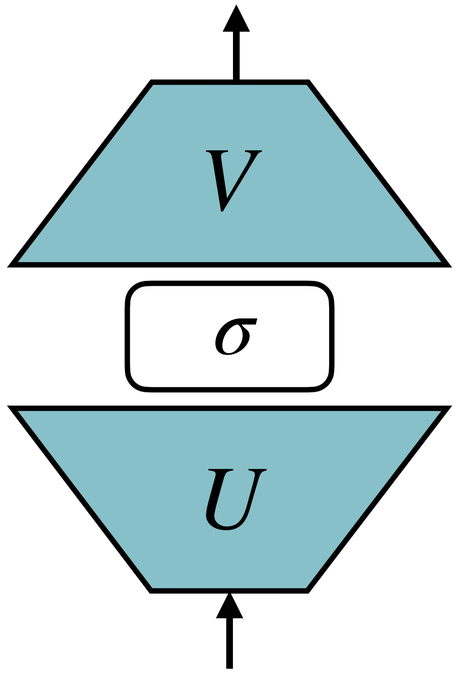
https://arxiv.org/pdf/2108.12409

Feed forward network

Feed forward network

- FFN consists of two linear layers with nonlinearity between them
- The first layer increases the dimensionality and the second layer decreases it back

$$FFN(x) = \sigma(xU + b_u)V + b_v$$



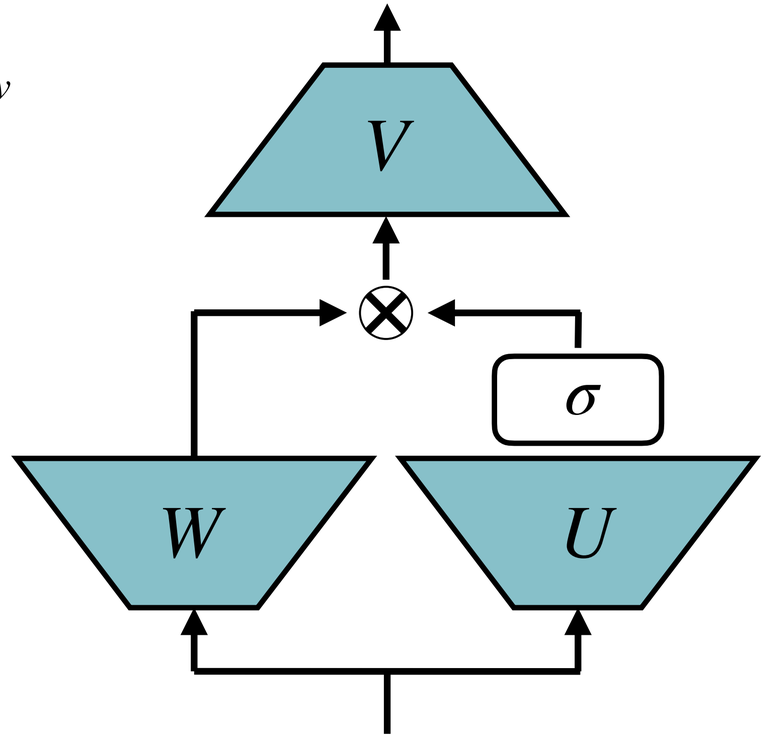
$$V \in \mathbb{R}^{D \times d}$$

$$U \in \mathbb{R}^{d \times D}$$

Gated Linear Unit (GLU)

- GLU adds an additional linear layer W whose outputs are multiplied by the outputs of U after activation
- W acts as a filter to filter out unnecessary components

$$FFN_{GLU}(x) = \left(\sigma(xU + b_u) \otimes (xW + b_w)\right)V + b_v$$



$$V \in \mathbb{R}^{D \times d}$$

$$U \in \mathbb{R}^{d \times D}$$

$$W \in \mathbb{R}^{d \times D}$$

Gated Linear Unit (GLU)

- GLU can be used with different activations
- GLU is most often used with Swish activation (very similar to ReLU)
- GLU is used instead of FFN in Llama, Mistral, Gemma, Qwen2

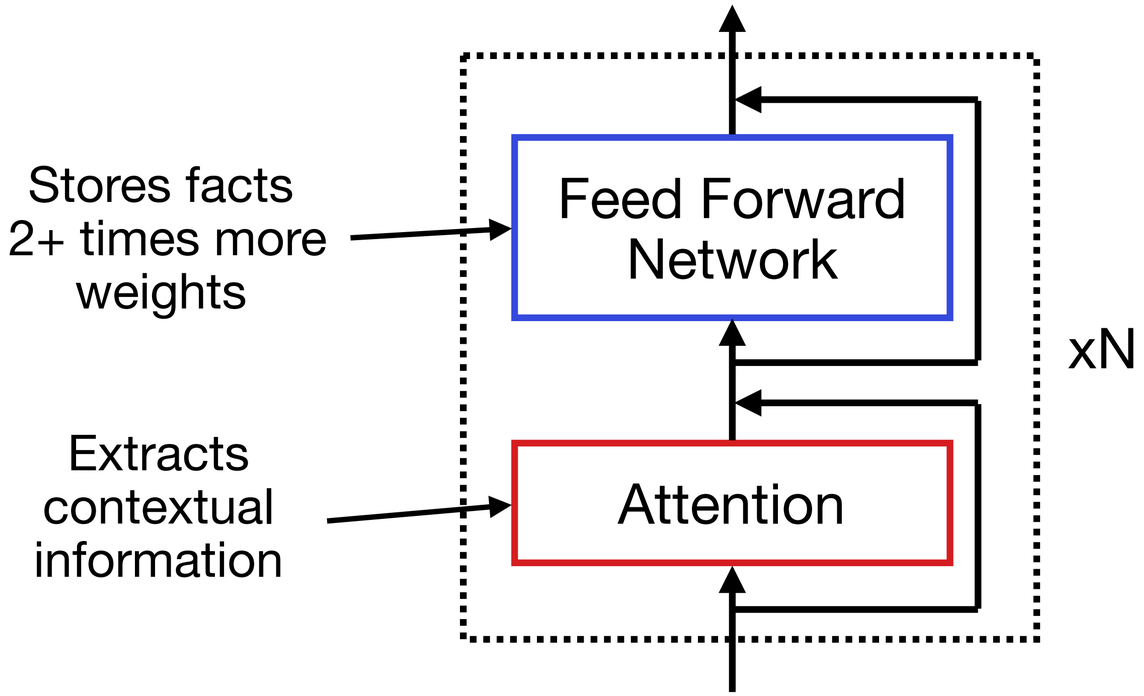
$$FFN_{GLU}(x) = \left(\sigma(xU + b_u) \otimes (xW + b_w)\right)V + b_v$$

	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

https://arxiv.org/pdf/2002.05202v1

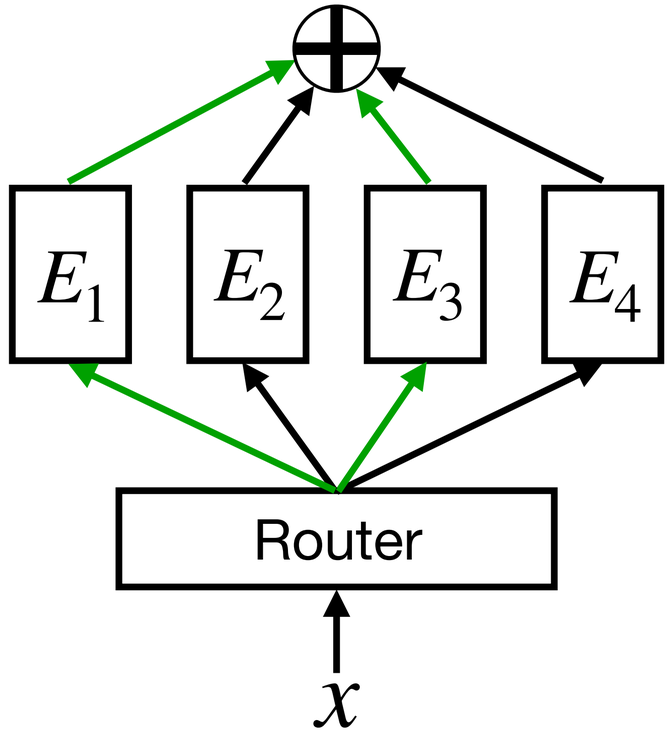
Mixture of Experts (MoE)

- Instead of only using one large FFN, a set of small specialized FFNs can be trained
- This way, it is possible to speed up the model application and reduce the memory cost



Mixture of Experts (MoE)

- MoE replaces the FFN layer
- MoE consists of a router and experts
- The router determines which token to give to which expert
- Experts are FFNs. They process tokens independently
- Usually, each token is processed by several experts at the same time



Router

- The router solves a standard classification problem
- It returns a probability for each expert
- The router is very simple and often consists of a single matrix

$$p(E \mid x) = softmax(R(x)) = softmax(xW)$$

Router

- The router solves a standard classification problem
- It returns a probability for each expert
- The router is very simple and often consists of a single matrix

$$p(E \mid x) = softmax(R(x)) = softmax(xW)$$

- In **Sparse MoE**, the probabilities for most experts are zeroed out
- This allows the model to be applied more quickly

$$p(E \mid x) = softmax(TopK(xW)) \qquad K = 2$$

Expert aggregation

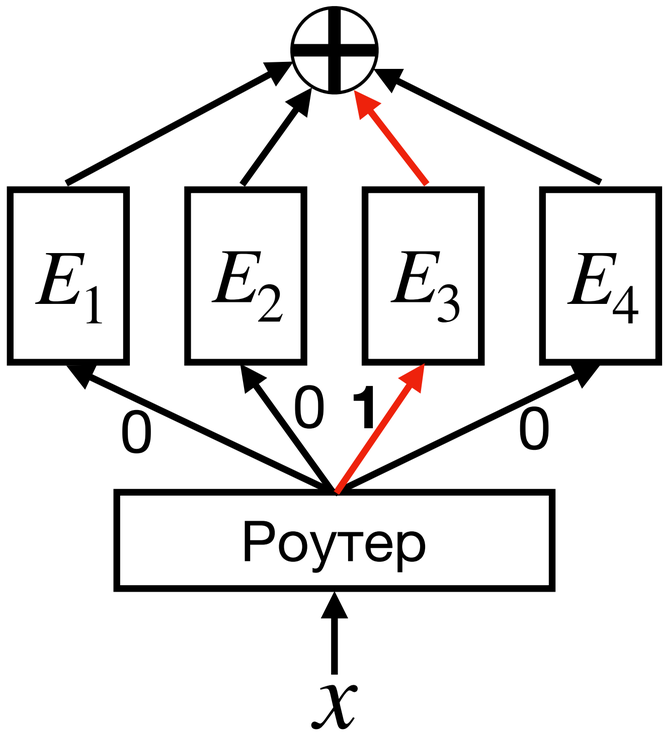
Experts' outputs are weighted by the probabilities and summed up

$$x' = \sum_{i=1}^{n} p(E_i \mid x) E_i(x),$$

where n – number of experts.

MoE training

- During training, the router learns along with the experts
- Often the router learns to give all probability to one expert
- To avoid this, you can
  - Add noise to the probabilities
  - Add regularization



Adding noise to the probabilities

- By adding noise to the probabilities, we reduce the probability of one expert being selected
- The remaining experts will receive gradients and start learning

$$p(E \mid x) = (1 - \alpha) \cdot softmax(R(x)) + \alpha \cdot \varepsilon,$$

where $\varepsilon \in [0,1]^n$ and $\sum_{i=0}^{n-1} \varepsilon_i = 1.$

- If α is too small, the problem will not disappear
- If α is too large, experts will learn the same thing

Regularization

To balance probabilities by experts, let's add the following regularization to the loss

$$L = \alpha \cdot n \cdot \sum_{i=1}^{n} f_i \cdot P_i,$$

where f_i – the proportion of tokens, given to i-th expert

$$f_i = \frac{1}{l} \sum_{k=1}^{l} [argmax\{p(E \mid x_k)\} = i]$$

and P_i – average probability for i-th expert

$$P_i = \frac{1}{l} \sum_{k=1}^{l} p(E_i \mid x_k)$$

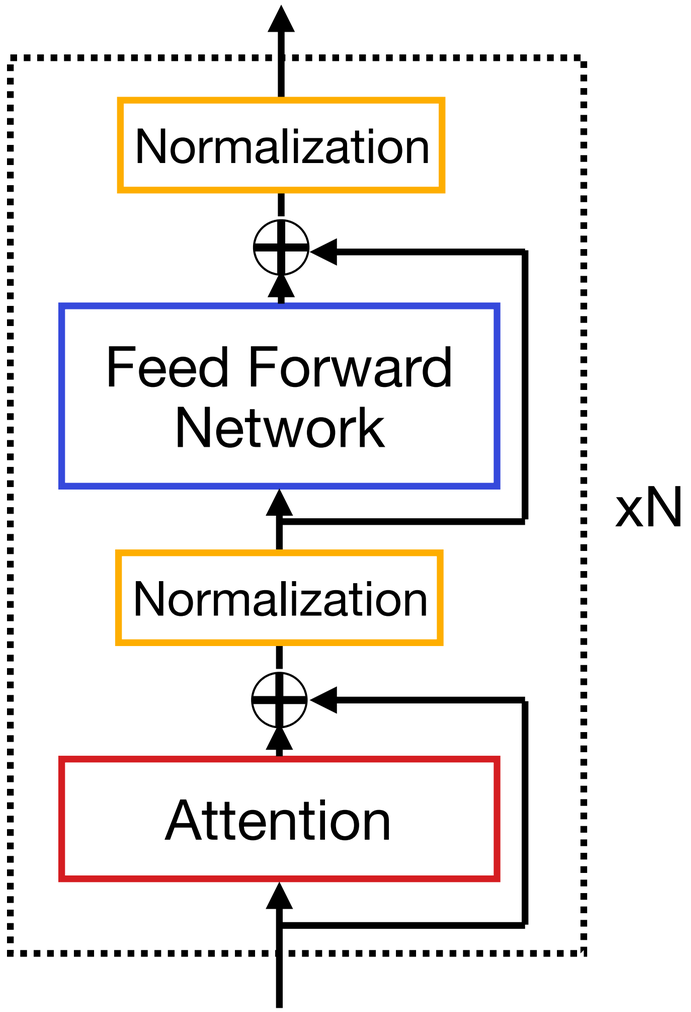
The minimum of the function is reached when $f_i = P_i = \frac{1}{n}$. That is exactly what we need.

Normalization

Normalization

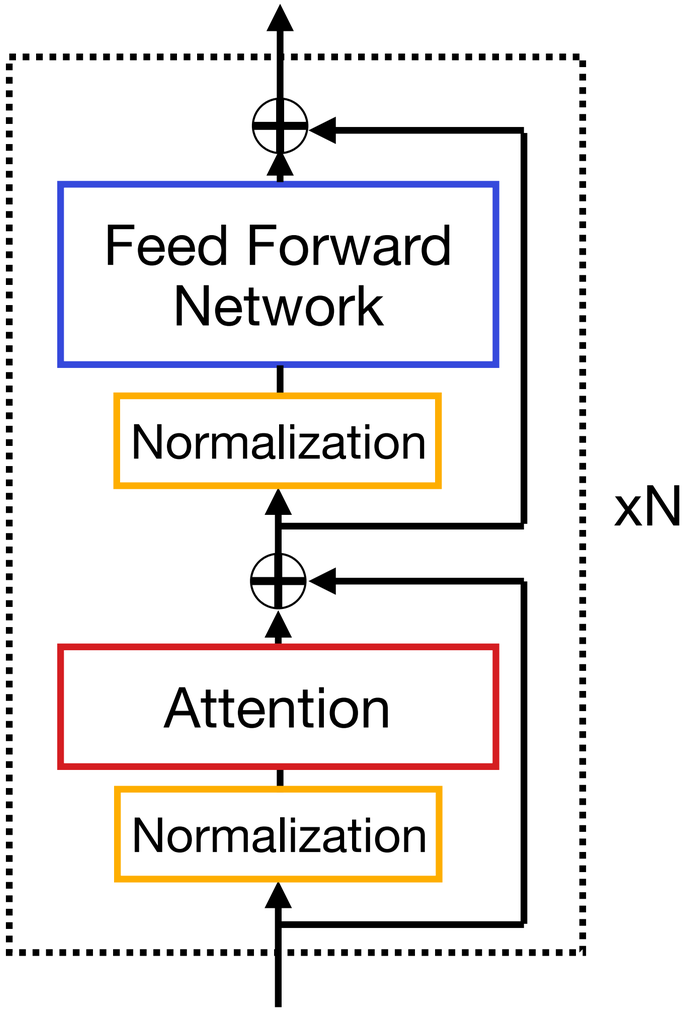
- Normalization stabilizes training by not allowing the outputs of the layers to diverge

- However, due to normalization, gradients start to vanish at the beginning of training
- This is why **warm-up** is used



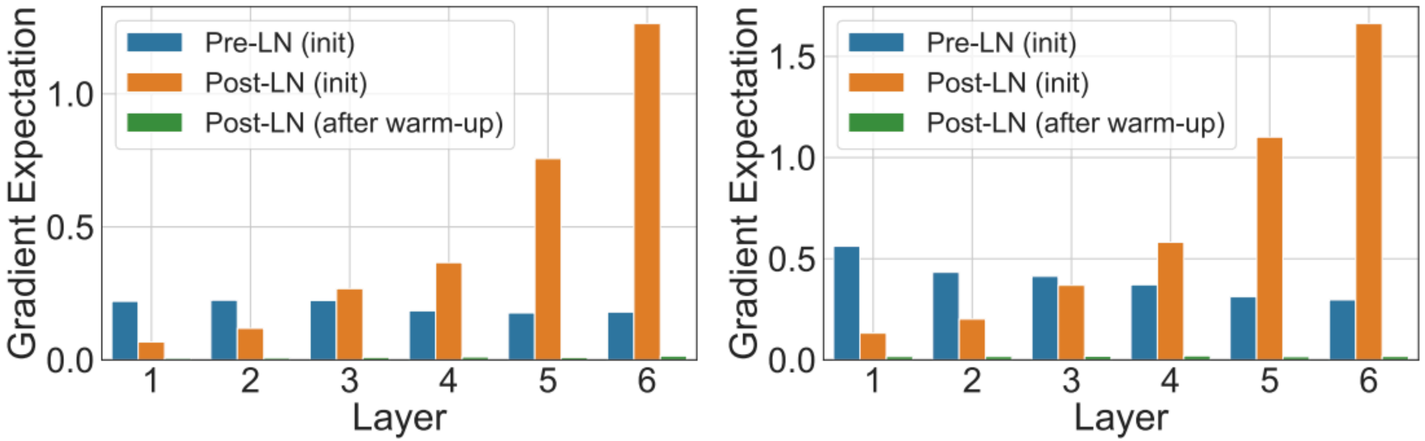
Pre-normalization

- It is possible to normalize not all tensors, but only the inputs for the attention and FFN layers
- This approach is called **Pre-LN** Transformer
- The standard approach is **Post-LN**
- Gradients now behave more stably and Transformer can be taught without warm-up



Gradient behaviour

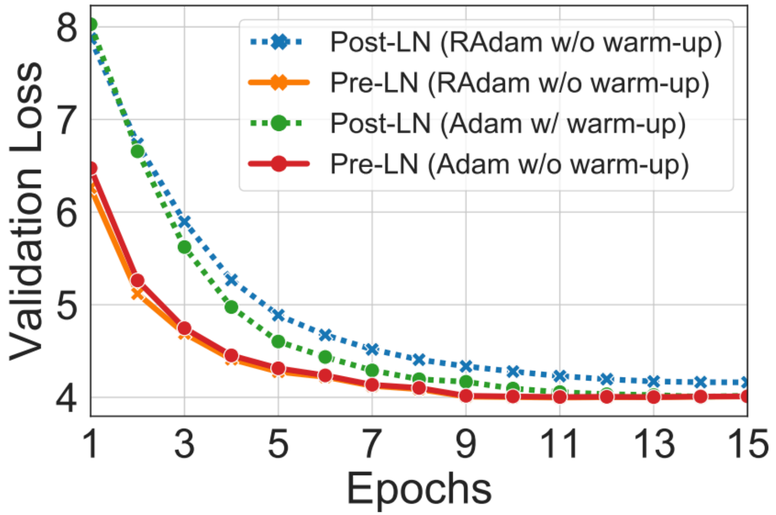
- The **Post-LN** gradients increase for deeper layers without warm-up
- This does not happen with **Pre-LN**
- **Pre-LN** is used in: Mistral, BLOOM, Falcon, Qwen2, etc.
- For **smaller** Transformers, the quality of Pre-LN is usually worse



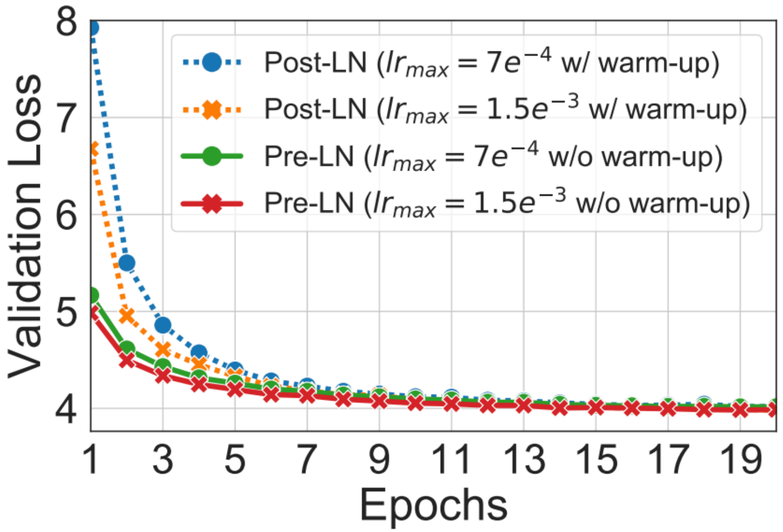
(a) W^1 in the FFN sub-layers (b) W^2 in the FFN sub-layers

https://arxiv.org/pdf/2002.04745

Pre-normalization in practice



(a) Validation Loss (IWSLT)



(c) Validation Loss (WMT)

https://arxiv.org/pdf/2002.04745

Layer Normalization

By default Layer Normalization is used in Transformers

$$y = \frac{x - \mathbb{E}(x)}{\sqrt{Var(x) + \epsilon}} \odot \gamma + \beta$$

To implement it we need to calculate mean and variance

$$\mathbb{E}(x) = \frac{1}{n}\sum_{i=1}^{n} x \qquad Var(x) = \frac{1}{n-1}\sum_{i=1}^{n} (x_i - \mathbb{E}(x))^2$$

Root Mean Square (RMS) Norm

By default Layer Normalization is used in Transformers

$$y = \frac{x - \mathbb{E}(x)}{\sqrt{Var(x) + \epsilon}} \odot \gamma + \beta$$

To implement it we need to calculate mean and variance

$$\mathbb{E}(x) = \frac{1}{n}\sum_{i=1}^{n} x \qquad Var(x) = \frac{1}{n-1}\sum_{i=1}^{n} (x_i - \mathbb{E}(x))^2$$

It turns out that you can consider the mean to be zero and discard it.

$$y = \frac{x}{\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2 + \epsilon}} \odot \gamma \qquad \gamma \in \mathbb{R}^d$$

Root Mean Square (RMS) Norm

- RMSNorm often performs better than Layer Norm
- RMSNorm is ~25% faster, since we don't need to estimate the tensor average
- Moreover, only a **part** of tensor elements can be used for variance estimation
- RMSNorm is used in modern models: Mistral, Llama, Qwen2, etc.

Model	Test14	Test17	Time
Baseline	21.7	23.4	399±3.40s
LayerNorm	22.6	23.6	665±32.5s
L2-Norm	20.7	22.0	482±19.7s
RMSNorm	22.4	**23.7**	501±11.8s (24.7%)
*p*RMSNorm	**22.6**	23.1	493±10.7s (25.9%)

Table 2: SacreBLEU score on newstest2014 (Test14) and newstest2017 (Test17) for RNNSearch using Tensorflow-version Nematus. "*Time*": the time in second per 1k training steps. We set p to 6.25%. We highlight the best results in bold, and show the speedup of RMSNorm against LayerNorm in bracket.

Baseline – with out normalization