Quick Announcements

- Gradescope is now live for HW1. Find the link for it from Canvas.
- Upload a requirements.txt file to Gradescope if your code needs non-standard packages.
- Office hours on the course website have been updated

Architecture Advancement on Transformers

Large Language Models: Methods and Applications

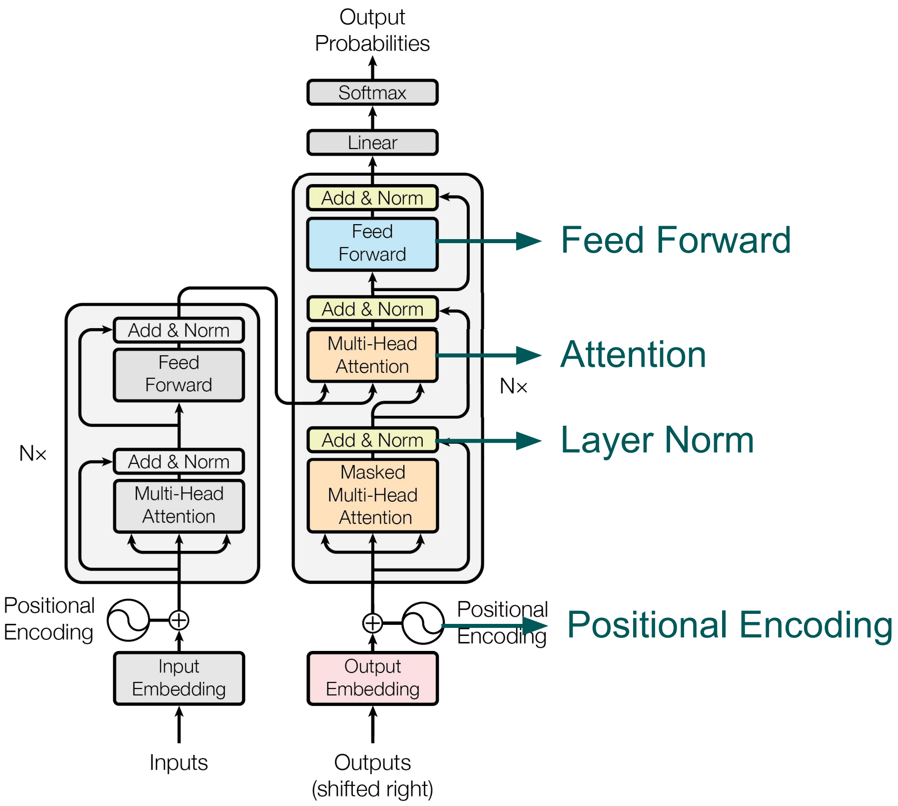
Daphne Ippolito and Chenyan Xiong

Learning Objectives

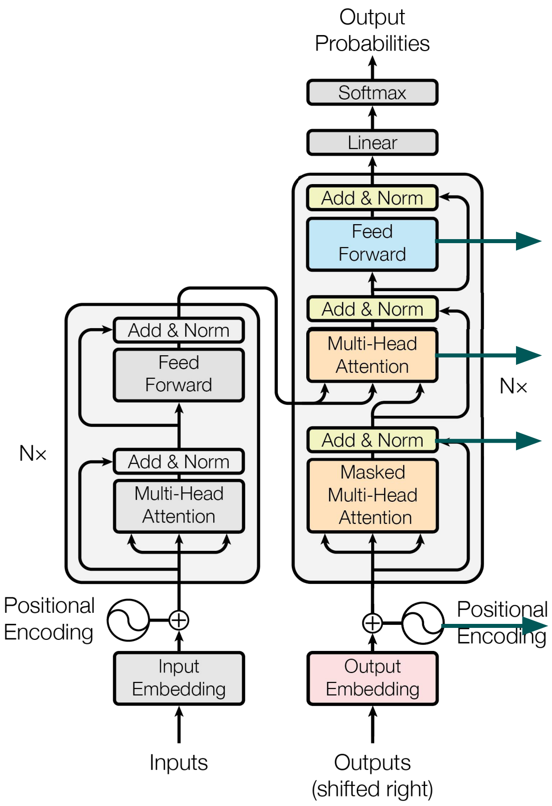
An overview of recent architecture advancements on top of Transformers

- A clear grasp on the details of new architectures
- Understand the motivation and benefits of each architecture upgrades
- Apply the right architecture specifications for target scenarios
- [Optional] Explore new architecture designs in your research

Places for Improvements



Places for Improvements



Feed Forward: Activations and Bilinear Layers

Attention

Layer Norm

Positional Encoding

Feed Forward: Activations and Bilinear Layers

Variants of linear FFN Layers (omitting bias):

$$\text{FFN}_{\text{RELU}}(x) = \text{RELU}(\,x\mathbf{W}_1)\mathbf{W}_2;\ \text{RELU}(\,x\mathbf{W}_1) = \max(0, x\mathbf{W}_1)\mathbf{W}_2$$

[1] Hendrycks and Gimpel. 2016. Gaussian Error Linear Units.
[2] Ramachandran et al. 2017. Searching for Activation Functions.

Feed Forward: Activations and Bilinear Layers

Variants of linear FFN Layers (omitting bias):

$$\mathrm{FFN}_{\mathrm{RELU}}(x) = \mathrm{RELU}(x\mathbf{W}_1)\mathbf{W}_2;\ \mathrm{RELU}(x\mathbf{W}_1) = \max(0, x\mathbf{W}_1)\mathbf{W}_2$$

$$\mathrm{FFN}_{\mathrm{GELU}}(x) = \mathrm{GELU}(x\mathbf{W}_1)\mathbf{W}_2; \mathrm{GELU}(x\mathbf{W}_1) = xP(X < x) = x\Phi(x)$$

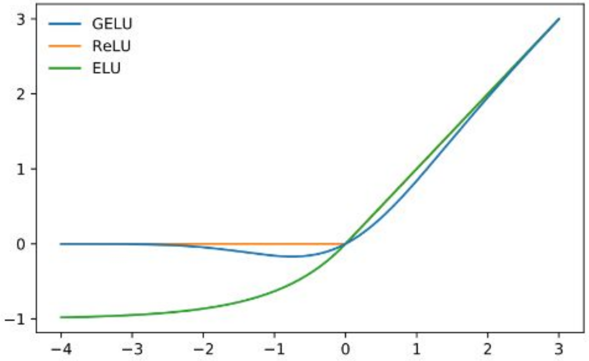
[1] Hendrycks and Gimpel. 2016. Gaussian Error Linear Units.
[2] Ramachandran et al. 2017. Searching for Activation Functions.

Feed Forward: Activations and Bilinear Layers

Variants of linear FFN Layers (omitting bias):

$$\mathrm{FFN}_{\mathrm{RELU}}(x) = \mathrm{RELU}(x\mathbf{W}_1)\mathbf{W}_2;\ \mathrm{RELU}(x\mathbf{W}_1) = \max(0, x\mathbf{W}_1)\mathbf{W}_2$$

$$\mathrm{FFN}_{\mathrm{GELU}}(x) = \mathrm{GELU}(x\mathbf{W}_1)\mathbf{W}_2; \mathrm{GELU}(x\mathbf{W}_1) = xP(X < x) = x\Phi(x)$$



GELU Activation [1]

[1] Hendrycks and Gimpel. 2016. Gaussian Error Linear Units.
[2] Ramachandran et al. 2017. Searching for Activation Functions.

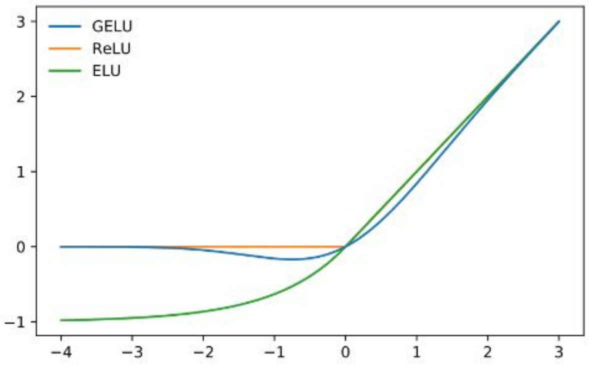
Feed Forward: Activations and Bilinear Layers

Variants of linear FFN Layers (omitting bias):

$$\mathrm{FFN}_{\mathrm{RELU}}(x) = \mathrm{RELU}(\,x\mathbf{W}_1)\mathbf{W}_2;\ \mathrm{RELU}(\,x\mathbf{W}_1) = \max(0, x\mathbf{W}_1)\mathbf{W}_2$$

$$\mathrm{FFN}_{\mathrm{GELU}}(x) = \mathrm{GELU}(\,x\mathbf{W}_1)\mathbf{W}_2; \mathrm{GELU}(\,x\mathbf{W}_1) = xP(X < x) = x\Phi(x)$$

$$\mathrm{FFN}_{\mathrm{Switch}}(x) = \mathrm{Swish}_1(\,x\mathbf{W}_1)\mathbf{W}_2; \mathrm{Swish}_\beta(\,x\mathbf{W}_1) = x\mathrm{Sigmod}(\beta x)$$



GELU Activation [1]

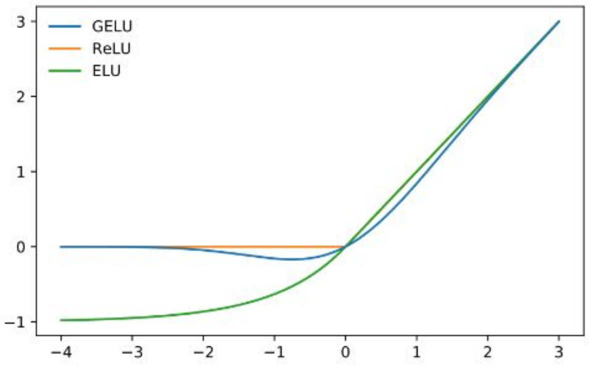
Feed Forward: Activations and Bilinear Layers

Variants of linear FFN Layers (omitting bias):

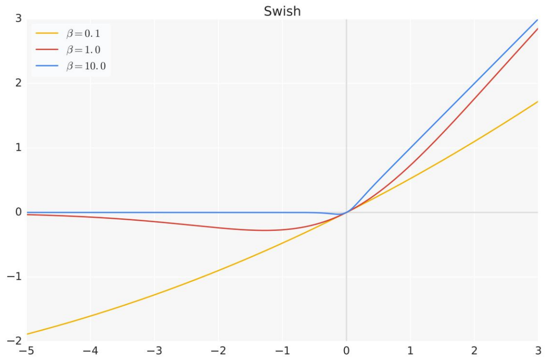
$$\mathrm{FFN}_{\mathrm{RELU}}(x) = \mathrm{RELU}(x\mathbf{W}_1)\mathbf{W}_2;\ \mathrm{RELU}(x\mathbf{W}_1) = \max(0, x\mathbf{W}_1)\mathbf{W}_2$$

$$\mathrm{FFN}_{\mathrm{GELU}}(x) = \mathrm{GELU}(x\mathbf{W}_1)\mathbf{W}_2; \mathrm{GELU}(x\mathbf{W}_1) = xP(X < x) = x\Phi(x)$$

$$\mathrm{FFN}_{\mathrm{Switch}}(x) = \mathrm{Swish}_1(x\mathbf{W}_1)\mathbf{W}_2; \mathrm{Swish}_\beta(x\mathbf{W}_1) = x\mathrm{Sigmod}(\beta x)$$



GELU Activation [1]



Switch Activation [2]

[1] Hendrycks and Gimpel. 2016. Gaussian Error Linear Units.
[2] Ramachandran et al. 2017. Searching for Activation Functions.

Feed Forward: Activations and Bilinear Layers

Bilinear FFNs (Omitting Bias):

$\mathrm{FFN}_{\mathrm{Bilinear}}(x) = (x\mathbf{W} \cdot x\mathbf{V})\mathbf{W}_2$. Two FFN with componentwise product

[3]] Shazeer 2017. GLU Variants Improve Transformer

Feed Forward: Activations and Bilinear Layers

Bilinear FFNs (Omitting Bias):

$\mathrm{FFN}_{\mathrm{Bilinear}}(x) = (x\mathbf{W} \cdot x\mathbf{V})\mathbf{W}_2$. Two FFN with componentwise product

$\mathrm{FFN}_{\mathrm{ReGLU}}(x) = (\mathrm{RELU}(x\mathbf{W}) \cdot x\mathbf{V})\mathbf{W}_2$. Adding RELU activation on one FFN

$\mathrm{FFN}_{\mathrm{GEGLU}}(x) = (\mathrm{GELU}(x\mathbf{W}) \cdot x\mathbf{V})\mathbf{W}_2$. Adding GELU activation on one FFN

$\mathrm{FFN}_{\mathrm{SwiGLU}}(x) = (\mathrm{Swish}_1(x\mathbf{W}) \cdot x\mathbf{V})\mathbf{W}_2$. Adding Swish activation on one FFN

[3] Shazeer 2017. GLU Variants Improve Transformer

Feed Forward: Activations and Bilinear Layers

Bilinear FFNs (Omitting Bias):

$\mathrm{FFN}_{\mathrm{Bilinear}}(x) = (x\mathbf{W} \cdot x\mathbf{V})\mathbf{W}_2$. Two FFN with componentwise product

$\mathrm{FFN}_{\mathrm{ReGLU}}(x) = (\mathrm{RELU}(x\mathbf{W}) \cdot x\mathbf{V})\mathbf{W}_2$. Adding RELU activation on one FFN

$\mathrm{FFN}_{\mathrm{GEGLU}}(x) = (\mathrm{GELU}(x\mathbf{W}) \cdot x\mathbf{V})\mathbf{W}_2$. Adding GELU activation on one FFN

$\mathrm{FFN}_{\mathrm{SwiGLU}}(x) = (\mathrm{Swish}_1(x\mathbf{W}) \cdot x\mathbf{V})\mathbf{W}_2$. Adding Swish activation on one FFN

With reduced hidden dimension of the projections to keep parameter count the same

Training Steps	65,536	524,288
$\mathrm{FFN}_{\mathrm{ReLU}}(baseline)$	1.997 (0.005)	1.677
$\mathrm{FFN}_{\mathrm{GELU}}$	1.983 (0.005)	1.679
$\mathrm{FFN}_{\mathrm{Swish}}$	1.994 (0.003)	1.683
$\mathrm{FFN}_{\mathrm{GLU}}$	1.982 (0.006)	1.663
$\mathrm{FFN}_{\mathrm{Bilinear}}$	1.960 (0.005)	1.648
$\mathrm{FFN}_{\mathrm{GEGLU}}$	**1.942** (0.004)	**1.633**
$\mathrm{FFN}_{\mathrm{SwiGLU}}$	**1.944** (0.010)	**1.636**
$\mathrm{FFN}_{\mathrm{ReGLU}}$	1.953 (0.003)	1.645

Improved speed-quality

T5 base Perplexity at Pretraining Steps [3]

[3] Shazeer 2017. GLU Variants Improve Transformer

Feed Forward: Activations and Bilinear Layers

Bilinear FFNs (Omitting Bias):

$\text{FFN}_{\text{Bilinear}}(x) = (x\mathbf{W} \cdot x\mathbf{V})\mathbf{W}_2$. Two FFN with componentwise product

$\text{FFN}_{\text{ReGLU}}(x) = (\text{RELU}(x\mathbf{W}) \cdot x\mathbf{V})\mathbf{W}_2$. Adding RELU activation on one FFN

$\text{FFN}_{\text{GEGLU}}(x) = (\text{GELU}(x\mathbf{W}) \cdot x\mathbf{V})\mathbf{W}_2$. Adding GELU activation on one FFN

$\text{FFN}_{\text{SwiGLU}}(x) = (\text{Swish}_1(x\mathbf{W}) \cdot x\mathbf{V})\mathbf{W}_2$. Adding Swish activation on one FFN

With reduced hidden dimension of the projections to keep parameter count the same

Training Steps	65,536	524,288
$\text{FFN}_{\text{ReLU}}(\textit{baseline})$	1.997 (0.005)	1.677
FFN_{GELU}	1.983 (0.005)	1.679
$\text{FFN}_{\text{Swish}}$	1.994 (0.003)	1.683
FFN_{GLU}	1.982 (0.006)	1.663
$\text{FFN}_{\text{Bilinear}}$	1.960 (0.005)	1.648
$\text{FFN}_{\text{GEGLU}}$	**1.942** (0.004)	**1.633**
$\text{FFN}_{\text{SwiGLU}}$	**1.944** (0.010)	**1.636**
$\text{FFN}_{\text{ReGLU}}$	1.953 (0.003)	1.645

Improved speed-quality

T5 base Perplexity at Pretraining Steps [3]

"We offer no explanation as to why these architectures seem to work; we attribute their success, as all else, to divine benevolence"---Noam Shazeer. 2017

[3] Shazeer 2017. GLU Variants Improve Transformer

Feed Forward: Activations and Bilinear Layers

Bilinear FFNs (Omitting Bias):

$\text{FFN}_{\text{Bilinear}}(x) = (x\mathbf{W} \cdot x\mathbf{V})\mathbf{W}_2$. Two FFN with componentwise product

$\text{FFN}_{\text{ReGLU}}(x) = (\text{RELU}(x\mathbf{W}) \cdot x\mathbf{V})\mathbf{W}_2$. Adding RELU activation on one FFN

$\text{FFN}_{\text{GEGLU}}(x) = (\text{GELU}(x\mathbf{W}) \cdot x\mathbf{V})\mathbf{W}_2$. Adding GELU activation on one FFN

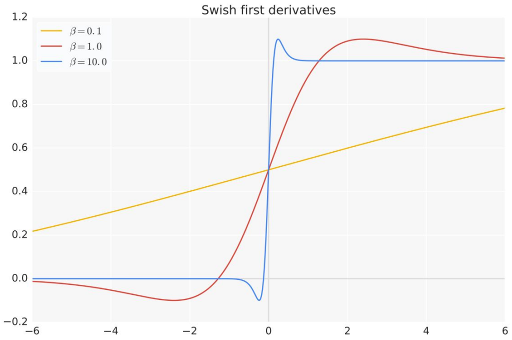
$\text{FFN}_{\text{SwiGLU}}(x) = (\text{Swish}_1(x\mathbf{W}) \cdot x\mathbf{V})\mathbf{W}_2$. Adding Swish activation on one FFN

With reduced hidden dimension of the projections to keep parameter count the same

Training Steps	65,536	524,288
$\text{FFN}_{\text{ReLU}}(baseline)$	1.997 (0.005)	1.677
FFN_{GELU}	1.983 (0.005)	1.679
$\text{FFN}_{\text{Swish}}$	1.994 (0.003)	1.683
FFN_{GLU}	1.982 (0.006)	1.663
$\text{FFN}_{\text{Bilinear}}$	1.960 (0.005)	1.648
$\text{FFN}_{\text{GEGLU}}$	**1.942** (0.004)	**1.633**
$\text{FFN}_{\text{SwiGLU}}$	**1.944** (0.010)	**1.636**
$\text{FFN}_{\text{ReGLU}}$	1.953 (0.003)	1.645

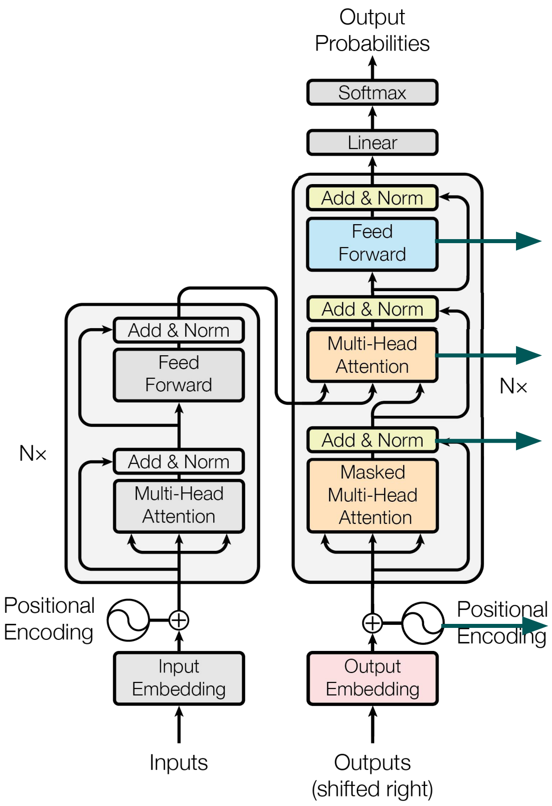
Improved speed-quality

T5 base Perplexity at Pretraining Steps [3]



Switch Gradients [2]

Places for Improvements



Feed Forward: Activations and Bilinear Layers

Attention: Efficient Attention Mechanisms

Layer Norm

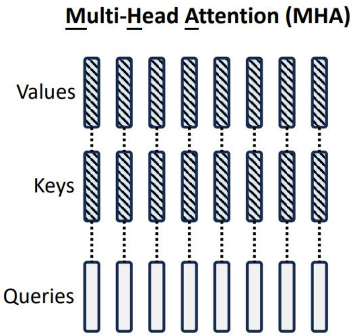
Positional Encoding

Attention: Efficient Attention Mechanisms

Standard Multi-Head Attention

$$\mathrm{head}_1 = \mathrm{Attention}\left(\mathbf{Q}\mathbf{W}_1^Q, \mathbf{K}\mathbf{W}_1^K, \mathbf{V}\mathbf{W}_1^V\right)$$
$$\vdots$$
$$\mathrm{head}_H = \mathrm{Attention}\left(\mathbf{Q}\mathbf{W}_H^Q, \mathbf{K}\mathbf{W}_H^K, \mathbf{V}\mathbf{W}_H^V\right)$$

$$\mathrm{MultiHeadAtt}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \mathrm{Concat}(\mathrm{head}_1, \dots, \mathrm{head}_H)$$



Attention: Efficient Attention Mechanisms

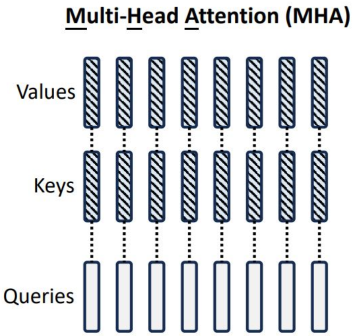
Standard Multi-Head Attention

$$\text{head}_1 = \text{Attention}(\mathbf{QW}_1^Q, \mathbf{KW}_1^K, \mathbf{VW}_1^V)$$
$$\vdots$$
$$\text{head}_H = \text{Attention}(\mathbf{QW}_H^Q, \mathbf{KW}_H^K, \mathbf{VW}_H^V)$$

$$\text{MultiHeadAtt}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Concat}(\text{head}_1, \ldots, \text{head}_H)$$

Inputs and outputs of each layer are the same dimensions:

$$\mathbf{Q} \in \mathbb{R}^{T \times d_{\text{model}}}$$
$$\mathbf{K} \in \mathbb{R}^{T \times d_{\text{model}}}$$
$$\mathbf{V} \in \mathbb{R}^{T \times d_{\text{model}}}$$



Attention: Efficient Attention Mechanisms

Standard Multi-Head Attention

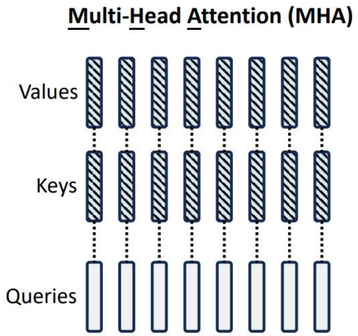
$$\text{head}_1 = \text{Attention}(\mathbf{Q}\mathbf{W}_1^Q, \mathbf{K}\mathbf{W}_1^K, \mathbf{V}\mathbf{W}_1^V)$$
$$\vdots$$
$$\text{head}_H = \text{Attention}(\mathbf{Q}\mathbf{W}_H^Q, \mathbf{K}\mathbf{W}_H^K, \mathbf{V}\mathbf{W}_H^V)$$

$$\text{MultiHeadAtt}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Concat}(\text{head}_1, \ldots, \text{head}_H)$$

Inputs and outputs of each layer are the same dimensions:

$$\mathbf{Q} \in \mathbb{R}^{T \times d_{\text{model}}}$$
$$\boxed{\begin{array}{l}\mathbf{K} \in \mathbb{R}^{T \times d_{\text{model}}} \\ \mathbf{V} \in \mathbb{R}^{T \times d_{\text{model}}}\end{array}}$$

Huge memory consumption during inference. Needs to keep one K, V for each layer and each position



Attention: Efficient Attention Mechanisms

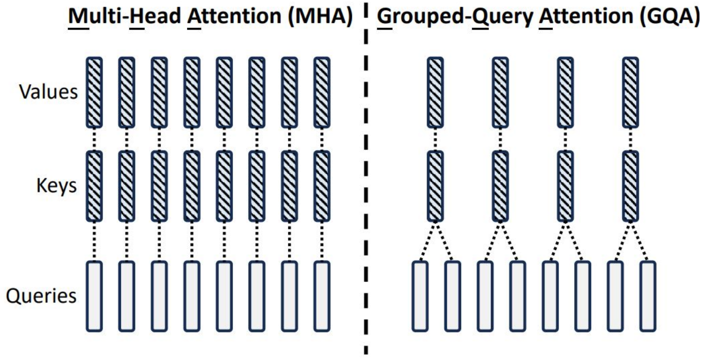
Grouped-Query Attention: Divide Q in G groups, and share K, V in the same group [4]

$$\text{head}_1 = \text{Attention}\left(\mathbf{Q}\mathbf{W}_1^Q, \mathbf{K}\mathbf{W}_1^K, \mathbf{V}\mathbf{W}_1^V\right)$$

$$\vdots$$

$$\text{head}_H = \text{Attention}\left(\mathbf{Q}\mathbf{W}_H^Q, \mathbf{K}\mathbf{W}_G^K, \mathbf{V}\mathbf{W}_G^V\right)$$

$$\text{MultiHeadAtt}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Concat}(\text{head}_1, \ldots, \text{head}_H)$$



[4] Ainslie et al. 2023. GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints

Attention: Efficient Attention Mechanisms

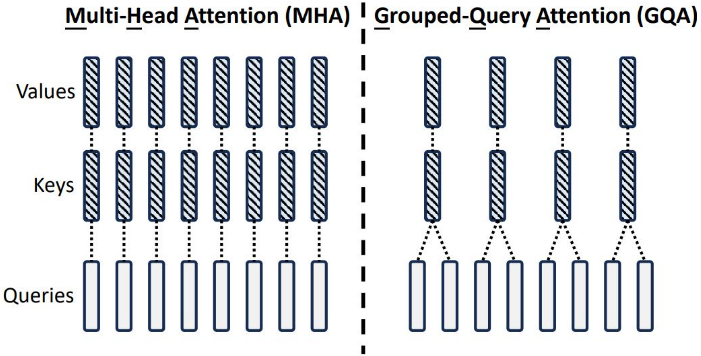
Grouped-Query Attention: Divide Q in G groups, and share K, V in the same group [4]

$$\text{head}_1 = \text{Attention}\left(\mathbf{Q}\mathbf{W}_1^Q, \mathbf{K}\mathbf{W}_1^K, \mathbf{V}\mathbf{W}_1^V\right)$$
$$\vdots$$
$$\text{head}_H = \text{Attention}\left(\mathbf{Q}\mathbf{W}_H^Q, \mathbf{K}\mathbf{W}_G^K, \mathbf{V}\mathbf{W}_G^V\right)$$

$$\text{MultiHeadAtt}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Concat}(\text{head}_1, \ldots, \text{head}_H)$$

Inputs and outputs of each layer are the same dimensions:

$$\mathbf{Q} \in \mathbb{R}^{T \times d_{\text{model}}}$$
$$\mathbf{K} \in \mathbb{R}^{T/(\frac{H}{G}) \times d_{\text{model}}}$$
$$\mathbf{V} \in \mathbb{R}^{T/(\frac{H}{G}) \times d_{\text{model}}}$$



Attention: Efficient Attention Mechanisms

Grouped-Query Attention: Divide Q in G groups, and share K, V in the same group [4]

$$\text{head}_1 = \text{Attention}(\mathbf{Q}\mathbf{W}_1^Q, \mathbf{K}\mathbf{W}_1^K, \mathbf{V}\mathbf{W}_1^V)$$

$$\vdots$$

$$\text{head}_H = \text{Attention}(\mathbf{Q}\mathbf{W}_H^Q, \mathbf{K}\mathbf{W}_G^K, \mathbf{V}\mathbf{W}_G^V)$$

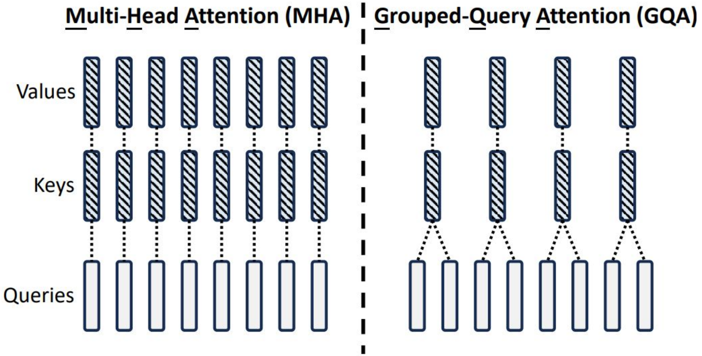
$$\text{MultiHeadAtt}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Concat}(\text{head}_1, \dots, \text{head}_H)$$

Inputs and outputs of each layer are the same dimensions:

$$\mathbf{Q} \in \mathbb{R}^{T \times d_{\text{model}}}$$

$$\boxed{\begin{array}{l}\mathbf{K} \in \mathbb{R}^{T/(\frac{H}{G}) \times d_{\text{model}}} \\ \mathbf{V} \in \mathbb{R}^{T/(\frac{H}{G}) \times d_{\text{model}}}\end{array}} \rightarrow$$

Reduce K, V cache storage to group sizes



Attention: Efficient Attention Mechanisms

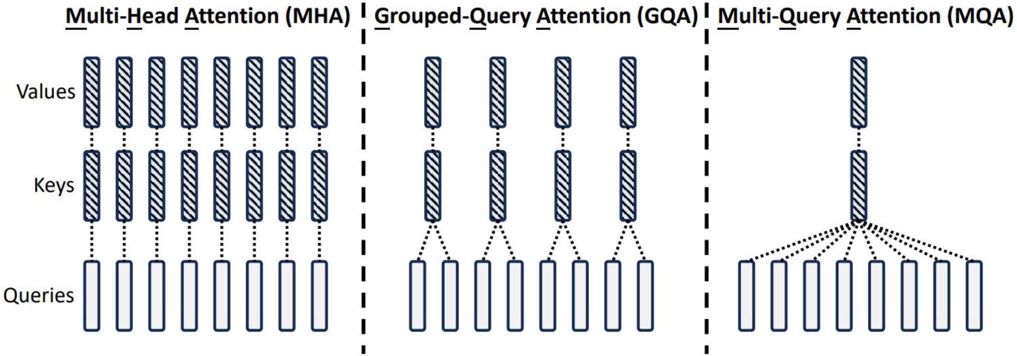
Multi-Query Attention: Single K, V for all Q heads [5]

$$\text{head}_1 = \text{Attention}\left(\mathbf{QW}_1^Q, \mathbf{KW}_1^K, \mathbf{VW}_1^V\right)$$
$$\vdots$$
$$\text{head}_H = \text{Attention}\left(\mathbf{QW}_H^Q, \mathbf{KW}_1^K, \mathbf{VW}_1^V\right)$$

$$\text{MultiHeadAtt}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Concat}(\text{head}_1, \dots, \text{head}_H)$$

Inputs and outputs of each layer are the same dimensions:

$$\mathbf{Q} \in \mathbb{R}^{T \times d_{\text{model}}}$$
$$\mathbf{K} \in \mathbb{R}^{T/H \times d_{\text{model}}}$$
$$\mathbf{V} \in \mathbb{R}^{T/H \times d_{\text{model}}}$$



[5] Noam Shazeer. 2019. Fast transformer decoding: One write-head is all you need.

Attention: Efficient Attention Mechanisms

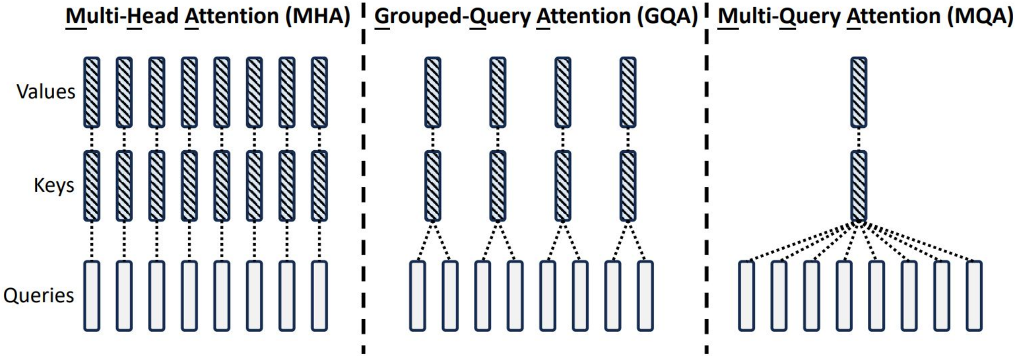
Multi-Query Attention: Single K, V for all Q heads [5]

$$\text{head}_1 = \text{Attention}(\mathbf{Q}\mathbf{W}_1^Q, \mathbf{K}\mathbf{W}_1^K, \mathbf{V}\mathbf{W}_1^V)$$
$$\vdots$$
$$\text{head}_H = \text{Attention}(\mathbf{Q}\mathbf{W}_H^Q, \mathbf{K}\mathbf{W}_1^K, \mathbf{V}\mathbf{W}_1^V)$$

$$\text{MultiHeadAtt}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Concat}(\text{head}_1, \dots, \text{head}_H)$$

Inputs and outputs of each layer are the same dimensions:

$$\mathbf{Q} \in \mathbb{R}^{T \times d_{\text{model}}}$$
$$\boxed{\mathbf{K} \in \mathbb{R}^{T/H \times d_{\text{model}}} \quad \mathbf{V} \in \mathbb{R}^{T/H \times d_{\text{model}}}} \longrightarrow \text{Further Reduce K, V Cache Size}$$



[5] Noam Shazeer. 2019. Fast transformer decoding: One write-head is all you need.

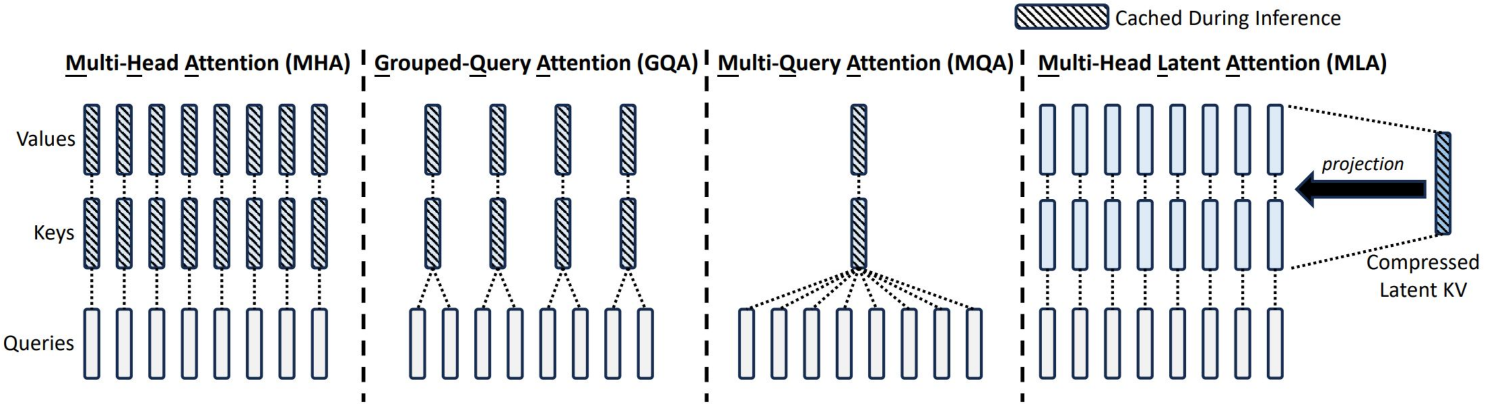
Attention: Efficient Attention Mechanisms

Multi-Head Latent Attention: Project K, V into a lower dimension latent vector [6]

$$\boldsymbol{k}_t = \mathbf{W}^{UK} \mathbf{c}_t^{KV}$$
$$\boldsymbol{v}_t = \mathbf{W}^{UV} \mathbf{c}_t^{KV}$$

Recovery of k, v can be merged with q in attention layer operations

$$\mathbf{c}_t^{KV} = \mathbf{W}^{DKV} \mathbf{h}_t$$ Only latent vector to store

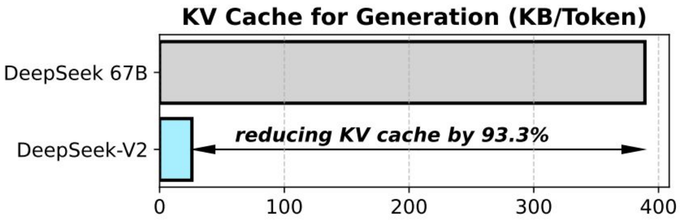


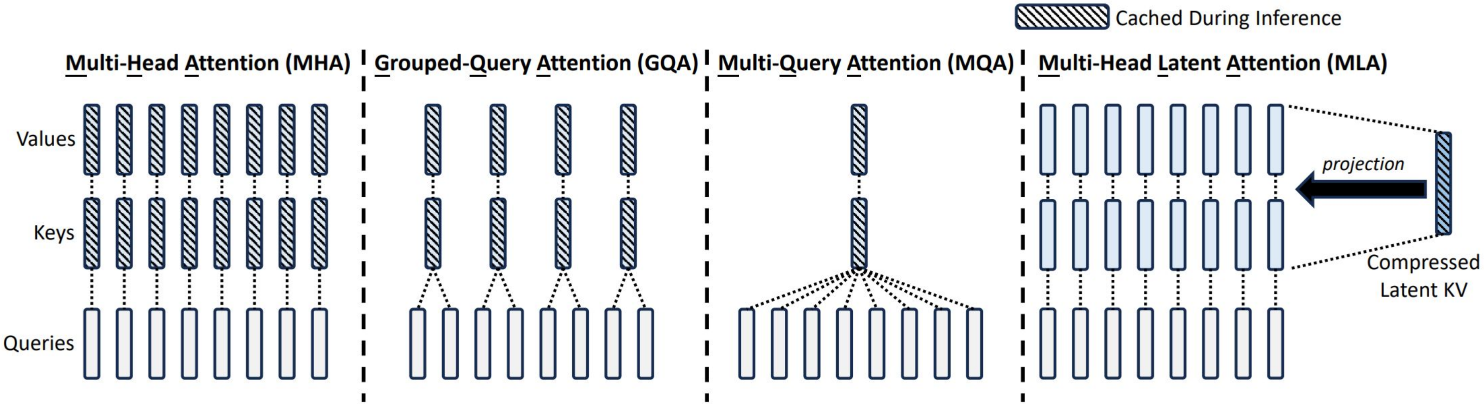
Attention: Efficient Attention Mechanisms

Multi-Head Latent Attention: Project K, V into a lower dimension latent vector [6]

$$\boldsymbol{k}_t = \mathbf{W}^{UK} \boldsymbol{c}_t^{KV}$$
$$\boldsymbol{v}_t = \mathbf{W}^{UV} \boldsymbol{c}_t^{KV}$$

$$\boldsymbol{c}_t^{KV} = \mathbf{W}^{DKV} \mathbf{h}_t$$ Only latent vector to store





Attention: Efficient Attention Mechanisms

How to use efficient attention mechanisms:

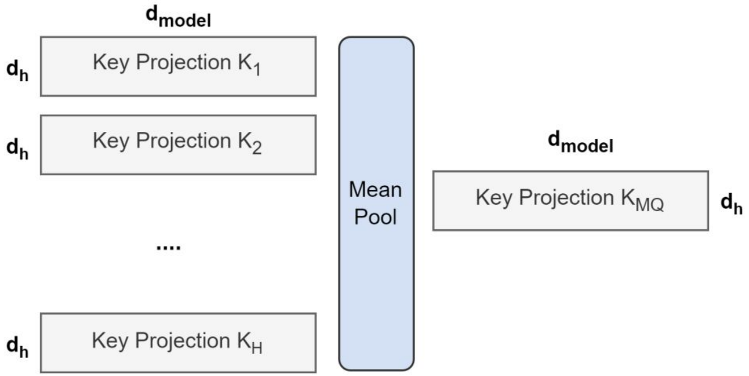
- Pretraining directly with updated architecture
- Or use it as a compression method to speed up a rich multi-head attention model

[4] Ainslie et al. 2023. GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints

Attention: Efficient Attention Mechanisms

How to use efficient attention mechanisms:

- Pretraining directly with updated architecture
- Or use it as a compression method to speed up a rich multi-head attention model



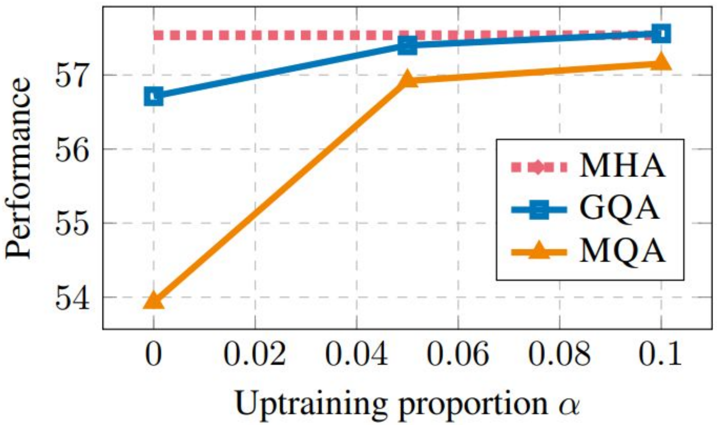
Mean Pooling Multi-Head Attention to Grouped-Query Attention [4]

[4] Ainslie et al. 2023. GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints

Attention: Efficient Attention Mechanisms

Performance:

- Recovering similar effectiveness as multi-head attention
- Significantly improve generation speed

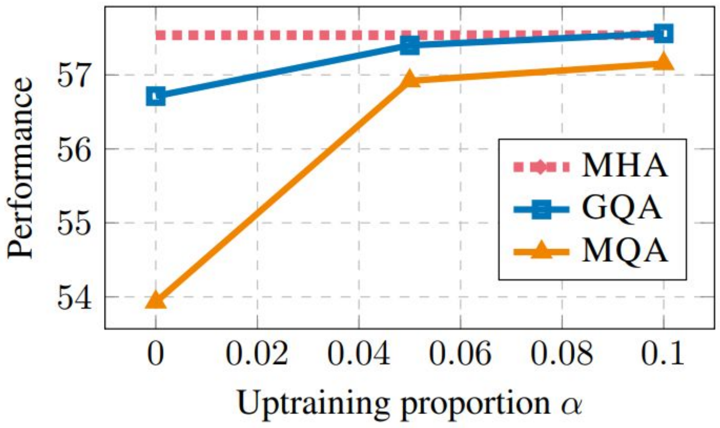


Performance of Grouped-Query Attention Adapted from Multi-Head Attention [4]

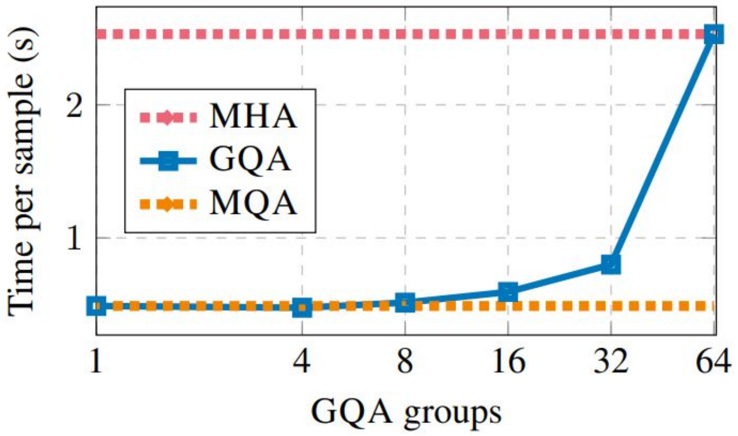
[4] Ainslie et al. 2023. GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints

Attention: Efficient Attention Mechanisms

Performance:
- Recovering similar effectiveness as multi-head attention
- Significantly improve generation speed

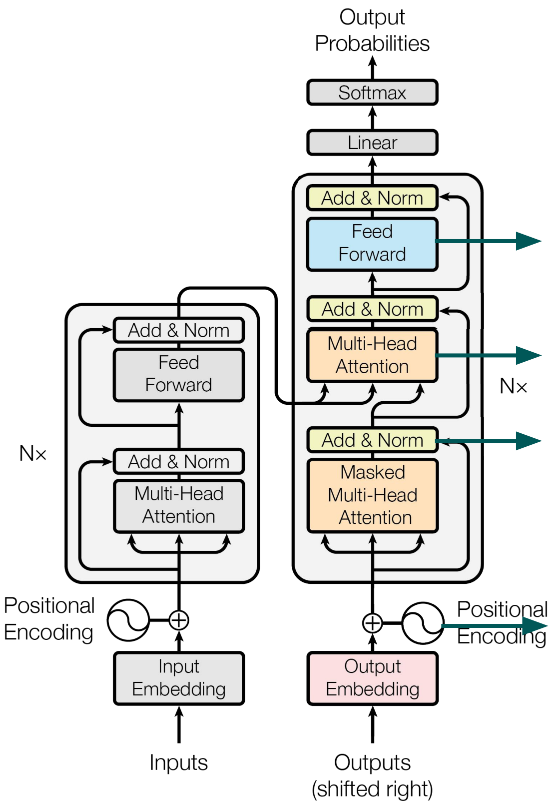


Performance of Grouped-Query Attention Adapted from Multi-Head Attention [4]



Generation Efficiency with Grouped-Query Attention [4]

Places for Improvements



Feed Forward: New Activations and Bilinear Layers

Attention: Efficient Attention Mechanisms

Layer Norm: Simpler and Stabler Layer Norms

Positional Encoding

Layernorm: Simpler and Stabler Layernorms

The Layer Normalization Layer [5]:

$$\mathrm{LN}(x_i) = \frac{(x_i - \mu)}{\sigma} g_i; \quad \mu = \frac{1}{d}\sum_i x_i, \sigma = \sqrt{\frac{1}{d}\sum_i (x_i - \mu)^2}$$

Learnable parameter started from 1

Align the outputs to standard distributions to improve training convergence and stability

[5] Ba et al. 2016. Layer Normalization
[6] Xiong et al. 2020. On Layer Normalization in the Transformer Architecture

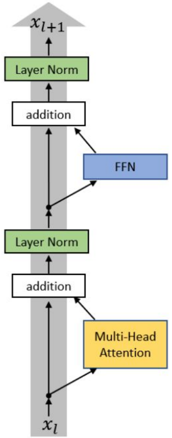
Layernorm: Simpler and Stabler Layernorms

The Layer Normalization Layer [5]:

$$\mathrm{LN}(x_i) = \frac{(x_i - \mu)}{\sigma} g_i; \quad \mu = \frac{1}{d}\sum_i x_i \,, \sigma = \sqrt{\frac{1}{d}\sum_i (x_i - \mu)^2}$$

Learnable parameter started from 1

Align the outputs to standard distributions to improve training convergence and stability



Post-LN: After Residual Sum

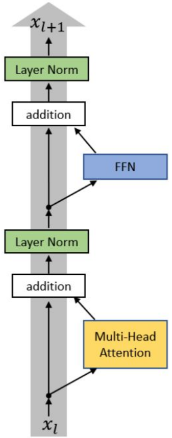
Layernorm: Simpler and Stabler Layernorms

The Layer Normalization Layer [5]:

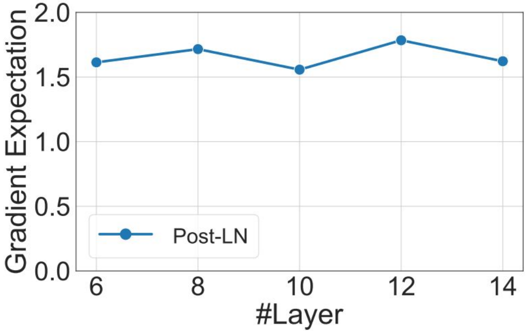
$$\mathrm{LN}(x_i) = \frac{(x_i - \mu)}{\sigma} g_i; \quad \mu = \frac{1}{d}\sum_i x_i \,, \sigma = \sqrt{\frac{1}{d}\sum_i (x_i - \mu)^2}$$

Learnable parameter started from 1

Align the outputs to standard distributions to improve training convergence and stability



Post-LN: After Residual Sum



Large gradient L2 norm, leading to unstable training at large scale [6]

[5] Ba et al. 2016. Layer Normalization
[6] Xiong et al. 2020. On Layer Normalization in the Transformer Architecture

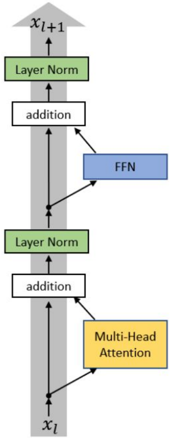
Layernorm: Simpler and Stabler Layernorms

The Layer Normalization Layer [5]:

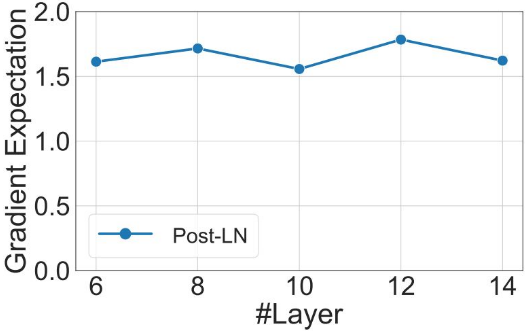
$$\mathrm{LN}(x_i) = \frac{(x_i - \mu)}{\sigma} g_i; \quad \mu = \frac{1}{d}\sum_i x_i \,, \sigma = \sqrt{\frac{1}{d}\sum_i (x_i - \mu)^2}$$

Learnable parameter started from 1

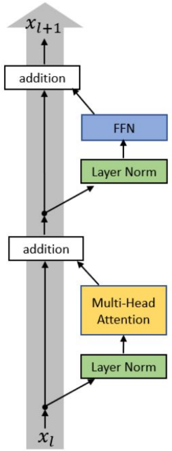
Align the outputs to standard distributions to improve training convergence and stability



Post-LN: After Residual Sum



Large gradient L2 norm, leading to unstable training at large scale [6]



Pre-LN: Before Residual Sum

[5] Ba et al. 2016. Layer Normalization
[6] Xiong et al. 2020. On Layer Normalization in the Transformer Architecture

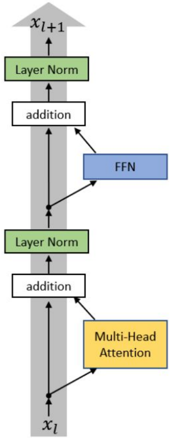
Layernorm: Simpler and Stabler Layernorms

The Layer Normalization Layer [5]:

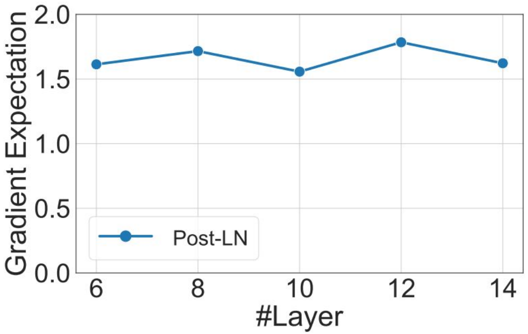
$$\mathrm{LN}(x_i) = \frac{(x_i - \mu)}{\sigma} g_i; \quad \mu = \frac{1}{d}\sum_i x_i, \sigma = \sqrt{\frac{1}{d}\sum_i (x_i - \mu)^2}$$

Learnable parameter started from 1

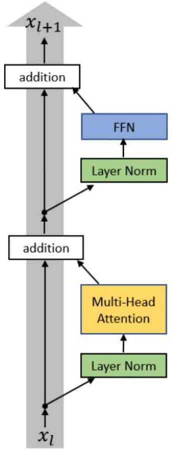
Align the outputs to standard distributions to improve training convergence and stability



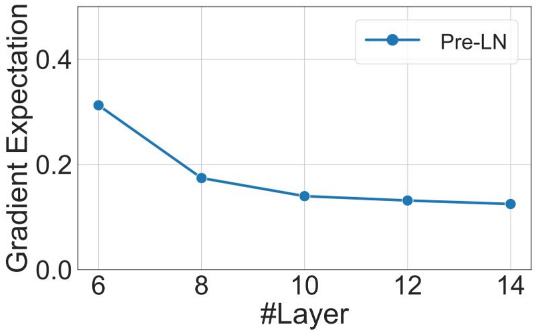
Post-LN: After Residual Sum



Large gradient L2 norm, leading to unstable training at large scale [6]



Pre-LN: Before Residual Sum



Lower gradient L2 norm, improved training stability [6]

[5] Ba et al. 2016. Layer Normalization
[6] Xiong et al. 2020. On Layer Normalization in the Transformer Architecture

Layernorm: Simpler and Stabler Layernorms

The Layer Normalization Layer [5]:

$$\mathrm{LN}(x_i) = \frac{(x_i - \mu)}{\sigma} g_i; \quad \mu = \frac{1}{d}\sum_i x_i \,, \sigma = \sqrt{\frac{1}{d}\sum_i (x_i - \mu)^2}$$

[5] Ba et al. 2016. Layer Normalization
[7] Zhang et al. 2019. Root Mean Square Layer Normalization

Layernorm: Simpler and Stabler Layernorms

The Layer Normalization Layer [5]:

$$\mathrm{LN}(x_i) = \frac{(x_i - \mu)}{\sigma} g_i; \quad \mu = \frac{1}{d}\sum_i x_i, \sigma = \sqrt{\frac{1}{d}\sum_i (x_i - \mu)^2}$$

RMSNorm: Only rescaling, no recentering [7]

$$\mathrm{LN}(x_i) = \frac{x_i}{\mathrm{RMS}(\mathbf{x})} g_i; \ \mathrm{RMS}(\mathbf{x}) = \sqrt{\frac{1}{d}(x_i)^2}$$

[5] Ba et al. 2016. Layer Normalization
[7] Zhang et al. 2019. Root Mean Square Layer Normalization

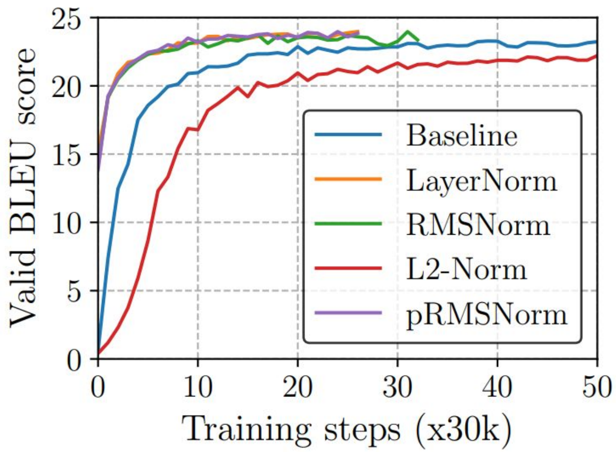
Layernorm: Simpler and Stabler Layernorms

The Layer Normalization Layer [5]:

$$\mathrm{LN}(x_i) = \frac{(x_i - \mu)}{\sigma} g_i; \quad \mu = \frac{1}{d}\sum_i x_i, \sigma = \sqrt{\frac{1}{d}\sum_i (x_i - \mu)^2}$$

RMSNorm: Only rescaling, no recentering [7]

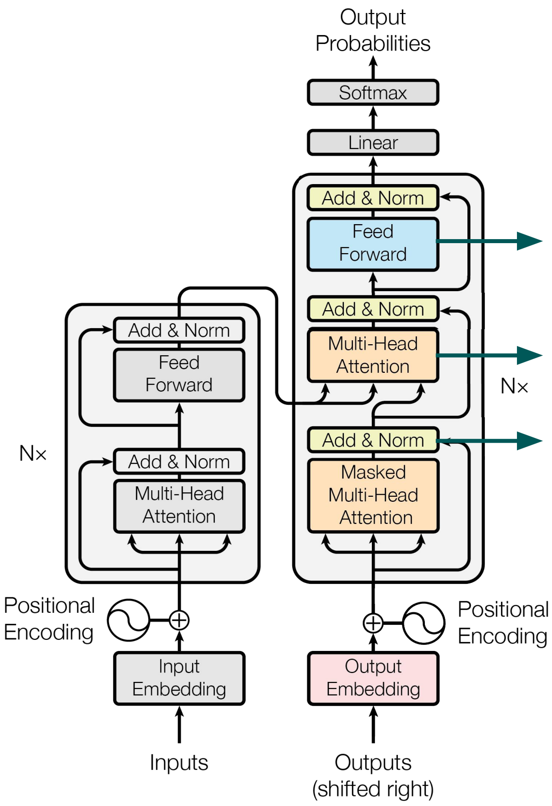
$$\mathrm{LN}(x_i) = \frac{x_i}{\mathrm{RMS}(\mathbf{x})} g_i; \ \mathrm{RMS}(\mathbf{x}) = \sqrt{\frac{1}{d}(x_i)^2}$$



Better Convergence Rate on Machine Translation and Many NLP Tasks [7]

[5] Ba et al. 2016. Layer Normalization
[7] Zhang et al. 2019. Root Mean Square Layer Normalization

Places for Improvements: Recap



Feed Forward: New Activations and Bilinear Layers

Improving Speed-Quality with Different Shapes

Attention: Efficient Attention Mechanisms

Trade Off Some Quality for (Inference) Speed

Layer Norm: Simpler and Stabler Layer Norms

Improving Training Speed and Robustness with Simplicity

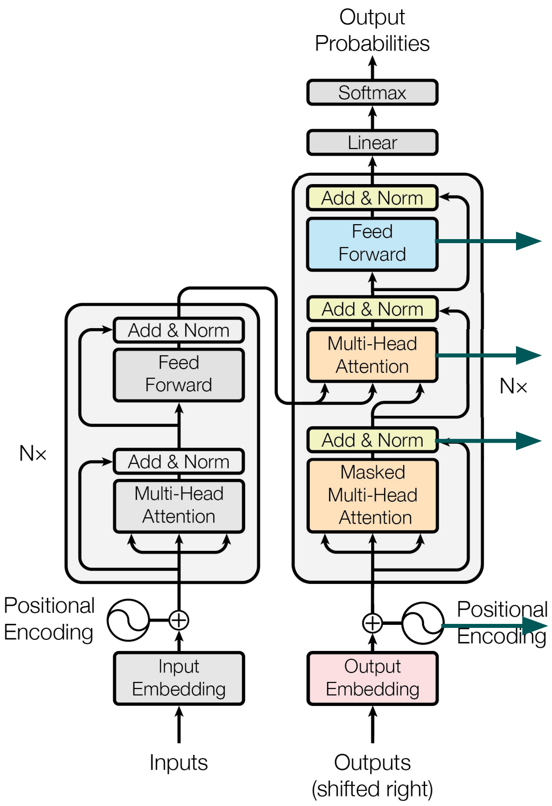
These architecture upgrades are adapted in most recent public production LLMs

Most Transformer architecture upgrades are for efficiency and large-scale learning stability. Simplicity is often the winner.

Most Transformer architecture upgrades are for efficiency and large-scale learning stability. Simplicity is often the winner.

Why?

Places for Improvements



Feed Forward: New Activations and Bilinear Layers

Attention: Efficient Attention Mechanisms

Layer Norm: Simpler and Stabler Layer Norms

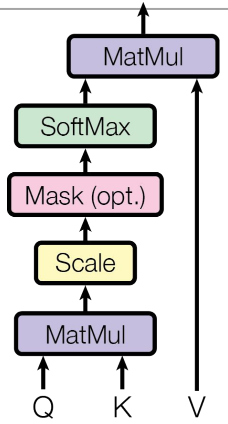
Positional Encoding: Better Capturing Position Information

Position Encoding: Capturing Position Information

Transformer itself is position agnostic:

$$\text{attention output at position } j = \sum_{i=1}^{T} \text{score}(\mathbf{q}_j, \mathbf{k}_i) \cdot \mathbf{v}_i$$

$$\text{score}(\mathbf{q}_j, \mathbf{k}_i) = \frac{\mathbf{q}_j \cdot \mathbf{k}_i}{\sqrt{d_k}}$$

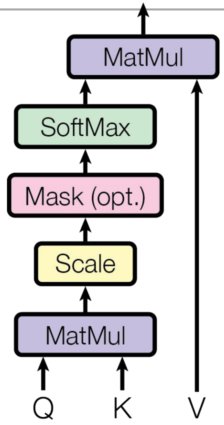


Position Encoding: Capturing Position Information

Transformer itself is position agnostic:

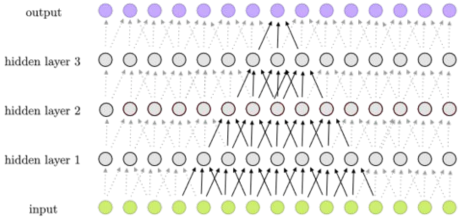
$$\text{attention output at position } j = \sum_{i=1}^{T} \text{score}(\mathbf{q}_j, \mathbf{k}_i) \cdot \mathbf{v}_i$$

$$\text{score}(\mathbf{q}_j, \mathbf{k}_i) = \frac{\mathbf{q}_j \cdot \mathbf{k}_i}{\sqrt{d_k}}$$



In comparison

CNN has locality prior:

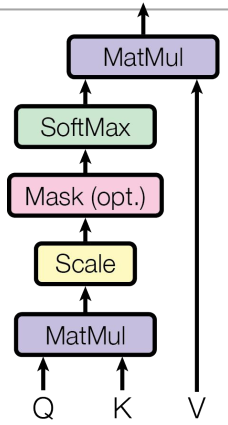


Position Encoding: Capturing Position Information

Transformer itself is position agnostic:

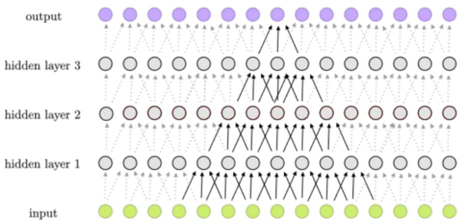
$$\text{attention output at position } j = \sum_{i=1}^{T} \text{score}(\mathbf{q}_j, \mathbf{k}_i) \cdot \mathbf{v}_i$$

$$\text{score}(\mathbf{q}_j, \mathbf{k}_i) = \frac{\mathbf{q}_j \cdot \mathbf{k}_i}{\sqrt{d_k}}$$

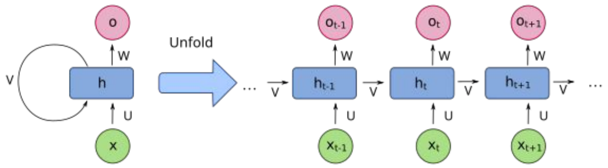


In comparison

CNN has locality prior:



RNN has sequential prior:

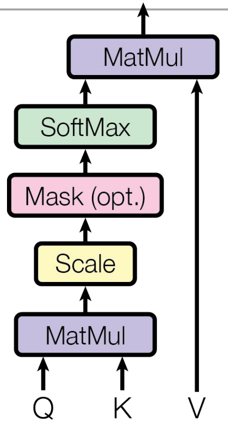


Position Encoding: Capturing Position Information

Transformer itself is position agnostic:

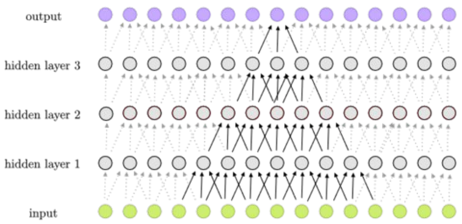
$$\text{attention output at position } j = \sum_{i=1}^{T} \text{score}(\mathbf{q}_j, \mathbf{k}_i) \cdot \mathbf{v}_i$$

$$\text{score}(\mathbf{q}_j, \mathbf{k}_i) = \frac{\mathbf{q}_j \cdot \mathbf{k}_i}{\sqrt{d_k}}$$

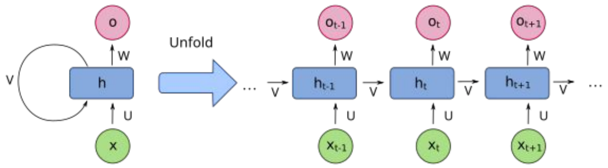


In comparison

CNN has locality prior:



RNN has sequential prior:



Position encoding adds positional information which is useful for language

Position Encoding: Absolute Position Information

Additive Position Encoding: add the positional information in the token embedding layer

$$\boldsymbol{x}' = \boldsymbol{x} + \underline{\boldsymbol{p}}_{pos}$$

Position Embedding at position pos

Position Encoding: Absolute Position Information

Additive Position Encoding: add the positional information in the token embedding layer

$$\boldsymbol{x}' = \boldsymbol{x} + \underline{\boldsymbol{p}_{pos}}$$

Position Embedding at position pos

Sinusoid position embedding

$$p_{pos,2i} = sin(pos/10000^{2i/d})$$
$$p_{pos,2i+1} = cos(pos/10000^{2i/d})$$

Position Encoding: Absolute Position Information

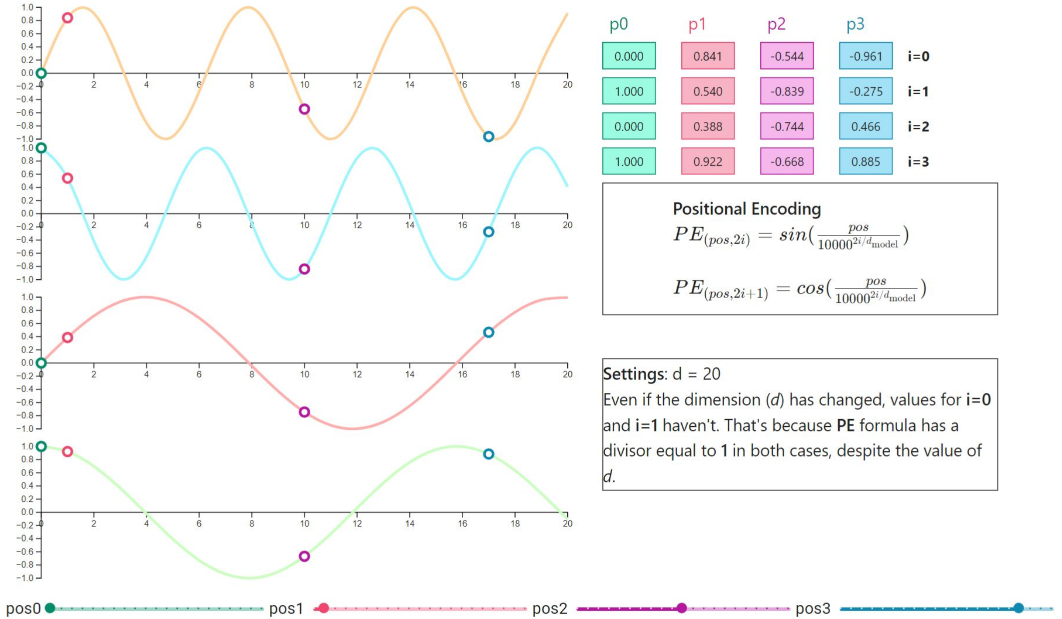
Additive Position Encoding: add the positional information in the token embedding layer

$$\boldsymbol{x}' = \boldsymbol{x} + \underline{\boldsymbol{p}_{pos}}$$

Position Embedding at position pos

Sinusoid position embedding

$$p_{pos,2i} = sin(pos/10000^{2i/d})$$
$$p_{pos,2i+1} = cos(pos/10000^{2i/d})$$



https://erdem.pl/2021/05/understanding-positional-encoding-in-transformers

Position Encoding: Absolute Position Information

Additive Position Encoding: add the positional information in the token embedding layer

$$\boldsymbol{x}' = \boldsymbol{x} + \underline{\boldsymbol{p}_{pos}}$$

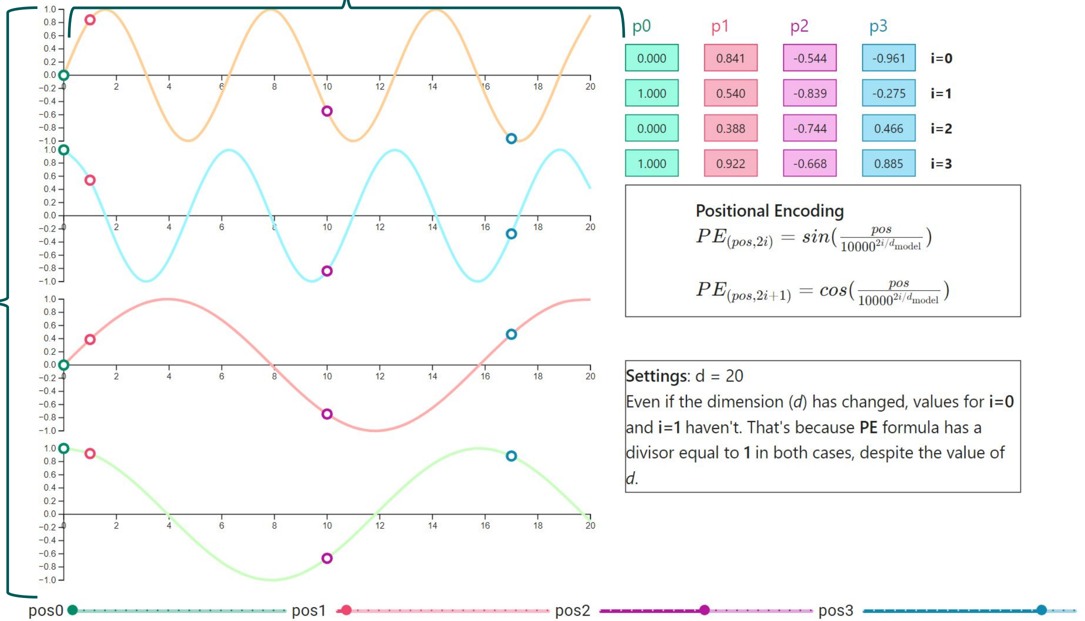
Position Embedding at position pos

Sinusoid position embedding

$$p_{pos,2i} = sin(pos/10000^{2i/d})$$
$$p_{pos,2i+1} = cos(pos/10000^{2i/d})$$

Different wavelength at different embedding dimension to capture different relative positions

Adding different values based on positions



https://erdem.pl/2021/05/understanding-positional-encoding-in-transformers

Position Encoding: Absolute Position Information

Additive Position Encoding: add the positional information in the token embedding layer

$$\boldsymbol{x}' = \boldsymbol{x} + \underline{\boldsymbol{p}_{pos}}$$

Position Embedding at position pos

Fully Learned Embeddings (e.g., in BERT)

$$\boldsymbol{p}_{pos} = Embedding(pos)$$

One embedding vector for each pos

Position Encoding: Absolute Position Information

Additive Position Encoding: add the positional information in the token embedding layer

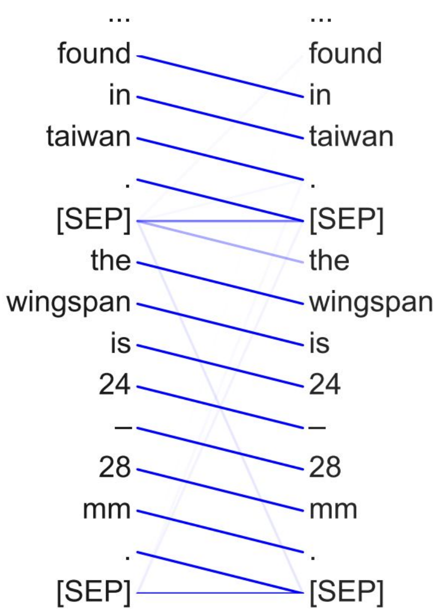
$$\boldsymbol{x}' = \boldsymbol{x} + \underline{\boldsymbol{p}_{pos}}$$

Position Embedding at position pos

Fully Learned Embeddings (e.g., in BERT)

$$\boldsymbol{p}_{pos} = Embedding(pos)$$

One embedding vector for each pos



Learned some strong
position-based attention patterns [8]

[8] Clark et al. 2019. What Does BERT Look At? An Analysis of BERT's Attention.

Position Encoding: Relative Position Information

Language Prior: Only relative positions matters in language

I took my dog, Fido, to the park for his walk….

Distance= 5

Distance= 5

Blablablabla…I took my dog, Fido, to the park for his walk….

Position Encoding: Relative Position Information

Language Prior: Only relative positions matters in language

I took my dog, Fido, to the park for his walk….

Distance= 5

Distance= 5

Blablablabla…I took my dog, Fido, to the park for his walk….

Encode relative position information in attention mechanism:

$$\text{Attention score}(\mathbf{q}_j, \mathbf{k}_i) = g(\mathbf{q}_j, \mathbf{k}_i, i - j)$$

[9] Raffel et al. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer
[10] Shaw et al. 2018. Self-Attention with Relative Position Representations

Position Encoding: Relative Position Information

Language Prior: Only relative positions matters in language

I took my dog, Fido, to the park for his walk….

Distance= 5

Distance= 5

Blablablabla…I took my dog, Fido, to the park for his walk….

Encode relative position information in attention mechanism:

$$\text{Attention score}(\mathbf{q}_j, \mathbf{k}_i) = g(\mathbf{q}_j, \mathbf{k}_i, i - j)$$

Using relative position embeddings [9]:

$$\text{Attention score}(\mathbf{q}_j, \mathbf{k}_i) = \frac{\mathbf{q}_j \cdot \mathbf{k}_i}{\sqrt{d_k}} + \underline{b_{i-j}}$$

Trainable Relative Position Bias

[9] Raffel et al. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer
[10] Shaw et al. 2018. Self-Attention with Relative Position Representations

Position Encoding: Relative Position Information

Language Prior: Only relative positions matters in language

I took my dog, Fido, to the park for his walk….

Distance= 5

≠

Distance= 5

Blablablabla…I took my dog, Fido, to the park for his walk….

Encode relative position information in attention mechanism:

$$\text{Attention score}(\mathbf{q}_j, \mathbf{k}_i) = g(\mathbf{q}_j, \mathbf{k}_i, i - j)$$

Using relative position embeddings [9]:

$$\text{Attention score}(\mathbf{q}_j, \mathbf{k}_i) = \frac{\mathbf{q}_j \cdot \mathbf{k}_i}{\sqrt{d_k}} + \underline{b_{i-j}}$$

Trainable Relative Position Bias

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**

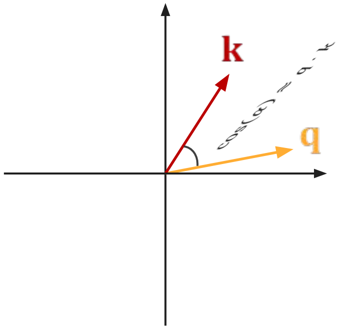
Performance of Relative Position Embedding on Machine Translation [10]

[9] Raffel et al. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer
[10] Shaw et al. 2018. Self-Attention with Relative Position Representations

Position Encoding: Rotational Position Embedding

Geometry of dot product in attention mechanism

$$\text{Attention score}(\mathbf{q}_j, \mathbf{k}_i) = \frac{\mathbf{q}_j \cdot \mathbf{k}_i}{\sqrt{d_k}}$$

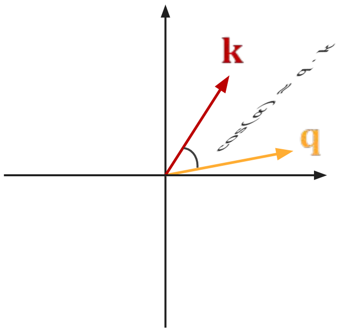


Attention score roughly as cosine of the vectors, assuming unit-lengths.

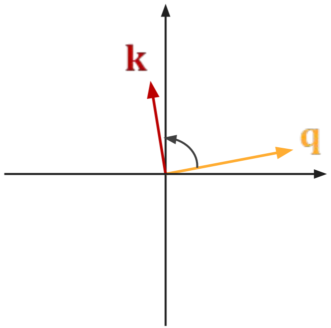
Position Encoding: Rotational Position Embedding

Geometry of dot product in attention mechanism

$$\text{Attention score}(\mathbf{q}_j, \mathbf{k}_i) = \frac{\mathbf{q}_j \cdot \mathbf{k}_i}{\sqrt{d_k}}$$



Attention score roughly as cosine of the vectors, assuming unit-lengths.

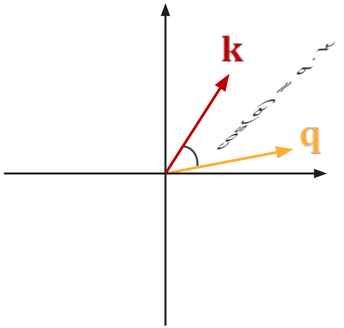


Lower attention importance if positions are far: rotating away

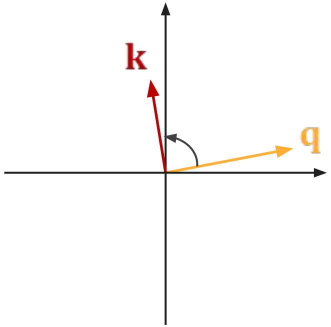
Position Encoding: Rotational Position Embedding

Geometry of dot product in attention mechanism

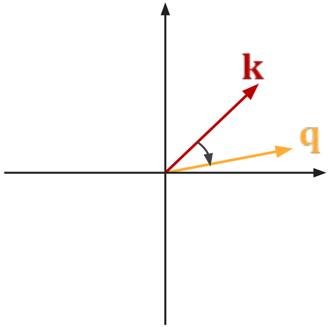
$$\text{Attention score}(\mathbf{q}_j, \mathbf{k}_i) = \frac{\mathbf{q}_j \cdot \mathbf{k}_i}{\sqrt{d_k}}$$



Attention score roughly as cosine of the vectors, assuming unit-lengths.



Lower attention importance if positions are far: rotating away

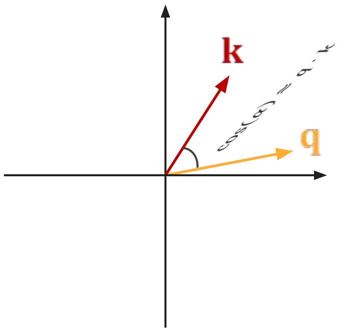


Higher attention importance if positions are close: rotating close

Position Encoding: Rotational Position Embedding

How to make the rotation only depend on relative positions?

$$\text{Attention score}(\mathbf{q}_j, \mathbf{k}_i) = g(\mathbf{q}_j, \mathbf{k}_i, i - j)$$

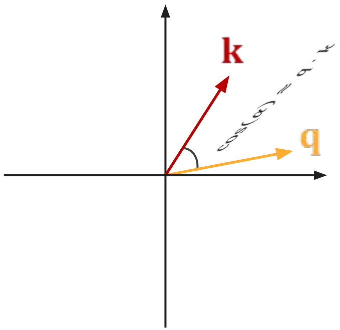


Attention score roughly as cosine of the vectors, assuming unit-lengths.

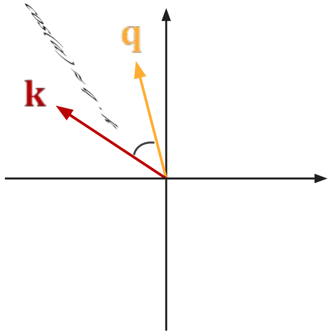
Position Encoding: Rotational Position Embedding

How to make the rotation only depend on relative positions?

$$\text{Attention score}(\mathbf{q}_j, \mathbf{k}_i) = g(\mathbf{q}_j, \mathbf{k}_i, i - j)$$



Attention score roughly as cosine of the vectors, assuming unit-lengths.

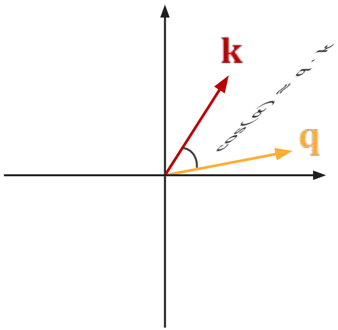


Rotating vectors together for the same disagrees based on positions changes

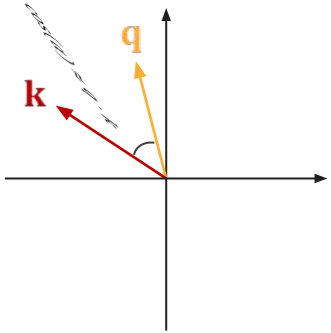
Position Encoding: Rotational Position Embedding

How to make the rotation only depend on relative positions?

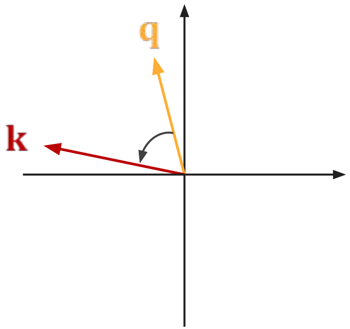
$$\text{Attention score}(\mathbf{q}_j, \mathbf{k}_i) = g(\mathbf{q}_j, \mathbf{k}_i, i - j)$$



Attention score roughly as cosine of the vectors, assuming unit-lengths.



Rotating vectors together for the same disagrees based on positions changes

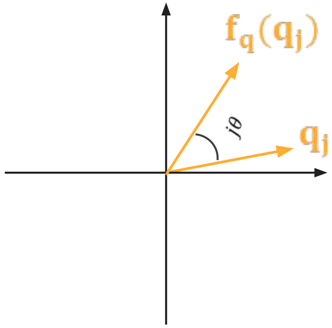


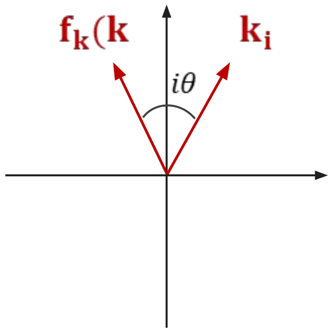
Position based prior holds the same at new absolute positions

Position Encoding: Rotational Position Embedding

Incorporate the vector rotation in the attention mechanism (2d space) [11]:

$$f_q(q_j) = \begin{pmatrix} \cos j\theta & -\sin j\theta \\ \sin j\theta & \cos j\theta \end{pmatrix} \begin{pmatrix} q_j^1 \\ q_j^2 \end{pmatrix} \qquad f_k(k_i) = \begin{pmatrix} \cos i\theta & -\sin i\theta \\ \sin i\theta & \cos i\theta \end{pmatrix} \begin{pmatrix} k_i^1 \\ k_i^2 \end{pmatrix} \qquad \theta_k = (1/10000^{2(i-1)/d})$$

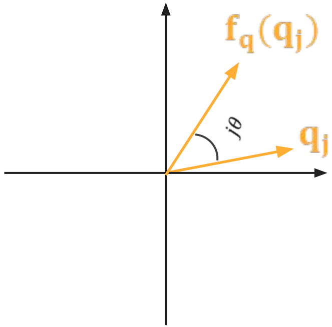




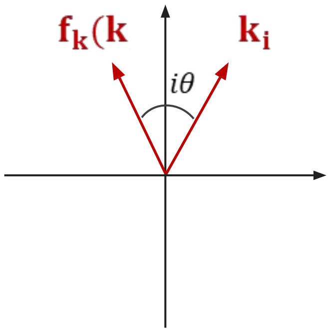
Position Encoding: Rotational Position Embedding

Incorporate the vector rotation in the attention mechanism (2d space) [11]:

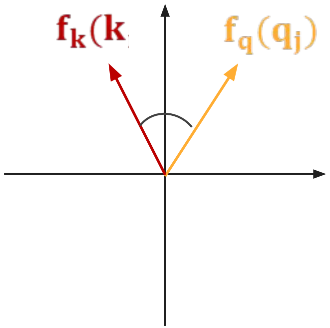
$$f_q(q_j) = \begin{pmatrix} \cos j\theta & -\sin j\theta \\ \sin j\theta & \cos j\theta \end{pmatrix} \begin{pmatrix} q_j^1 \\ q_j^2 \end{pmatrix} \qquad f_k(k_i) = \begin{pmatrix} \cos i\theta & -\sin i\theta \\ \sin i\theta & \cos i\theta \end{pmatrix} \begin{pmatrix} k_i^1 \\ k_i^2 \end{pmatrix} \qquad \theta_k = (1/10000^{2(i-1)/d})$$



Rotate q by $j\theta$



Rotate k by $i\theta$

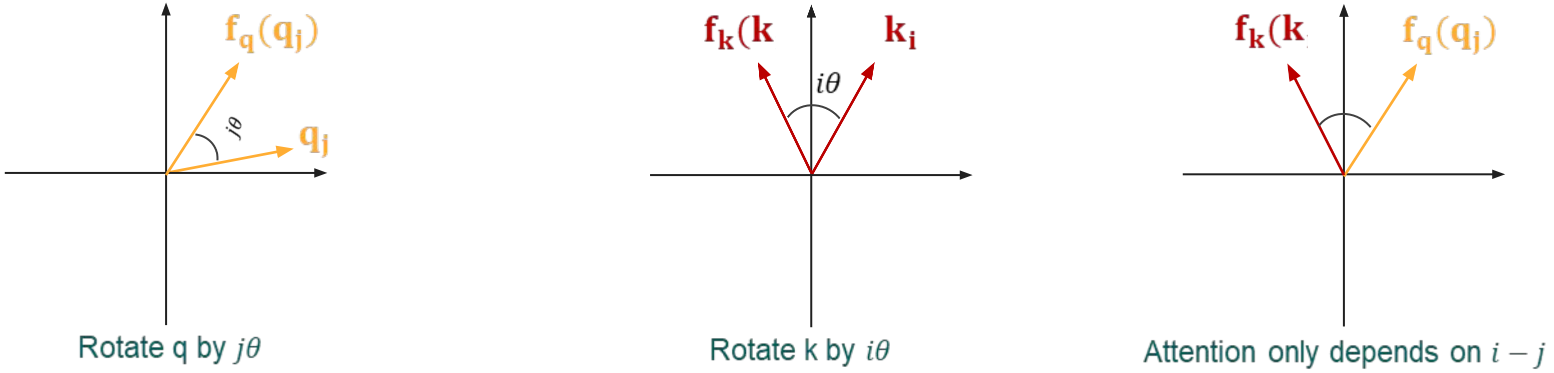


Attention only depends on $i - j$

Position Encoding: Rotational Position Embedding

Incorporate the vector rotation in the attention mechanism (2d space) [11]:

$$f_q(q_j) = \begin{pmatrix} \cos j\theta & -\sin j\theta \\ \sin j\theta & \cos j\theta \end{pmatrix} \begin{pmatrix} q_j^1 \\ q_j^2 \end{pmatrix} \qquad f_k(k_i) = \begin{pmatrix} \cos i\theta & -\sin i\theta \\ \sin i\theta & \cos i\theta \end{pmatrix} \begin{pmatrix} k_i^1 \\ k_i^2 \end{pmatrix} \qquad \theta_k = (1/10000^{2(i-1)/d})$$



Attention score by the dot prod of rotated vectors:

$$\text{Attention score}(\mathbf{q}_j, \mathbf{k}_i) = f_q(q_j) \cdot f_k(k_i) = \begin{pmatrix} q_j^1 \\ q_j^2 \end{pmatrix}^T \begin{pmatrix} \cos j\theta & -\sin j\theta \\ \sin j\theta & \cos j\theta \end{pmatrix}^T \begin{pmatrix} \cos i\theta & -\sin i\theta \\ \sin i\theta & \cos i\theta \end{pmatrix} \begin{pmatrix} k_i^1 \\ k_i^2 \end{pmatrix}$$

Position Encoding: Rotational Position Embedding

Full form in the high dimensional space [11]:

$$f_{q,k}(x_i) = \begin{pmatrix} \begin{pmatrix} \cos i\theta_1 & -\sin i\theta_1 \\ \sin i\theta_1 & \cos i\theta_1 \end{pmatrix} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \begin{pmatrix} \cos i\theta_{d/2} & -\sin i\theta_{d/2} \\ \sin i\theta_{d/2} & \cos i\theta_{d/2} \end{pmatrix} \end{pmatrix} \begin{matrix} \boxed{\begin{pmatrix} x_i^1 \\ x_i^2 \end{pmatrix}} \\ \vdots \\ \begin{pmatrix} x_i^{d/2} \\ x_i^{d/2} \end{pmatrix} \end{matrix}$$

Partition dimensions into pairs and do the 2d rotation on each pair

With different wavelet lengths: $\theta_k = (1/10000^{2(k-1)/d})$

Position Encoding: Rotational Position Embedding

Full form in the high dimensional space [11]:

$$f_{q,k}(x_i) = \begin{pmatrix} \begin{pmatrix} \cos i\theta_1 & -\sin i\theta_1 \\ \sin i\theta_1 & \cos i\theta_1 \end{pmatrix} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \begin{pmatrix} \cos i\theta_{d/2} & -\sin i\theta_{d/2} \\ \sin i\theta_{d/2} & \cos i\theta_{d/2} \end{pmatrix} \end{pmatrix} \begin{matrix} \boxed{\begin{pmatrix} x_i^1 \\ x_i^2 \end{pmatrix}} \\ \vdots \\ \begin{pmatrix} x_i^{d/2} \\ x_i^{d/2} \end{pmatrix} \end{matrix}$$

Partition dimensions into pairs and do the 2d rotation on each pair

With different wavelet lengths: $\theta_k = (1/10000^{2(k-1)/d})$

Rooted in the sinusoid absolute position embedding, but does multiplication (rotation):

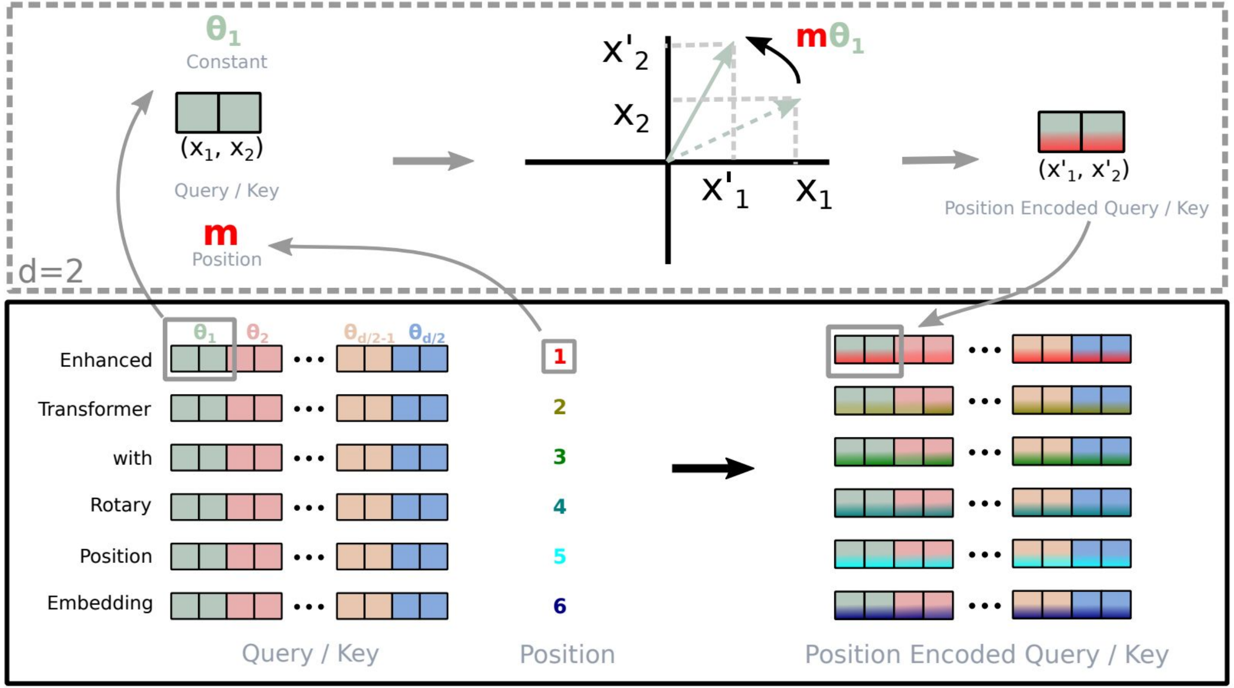
$$\boldsymbol{x}' = \boldsymbol{x} + \boldsymbol{p}_{pos} \qquad p_{pos,2i} = sin(pos/10000^{2i/d})$$
$$p_{pos,2i+1} = cos(pos/10000^{2i/d})$$

"We chose this function because we hypothesized it would allow the model to easily learn to attend by relative positions"---Transformer Paper

[11] Sun et al. 2023. RoFormer: Enhanced transformer with Rotary Position Embedding

Position Encoding: Rotational Position Embedding

Putting it all together:

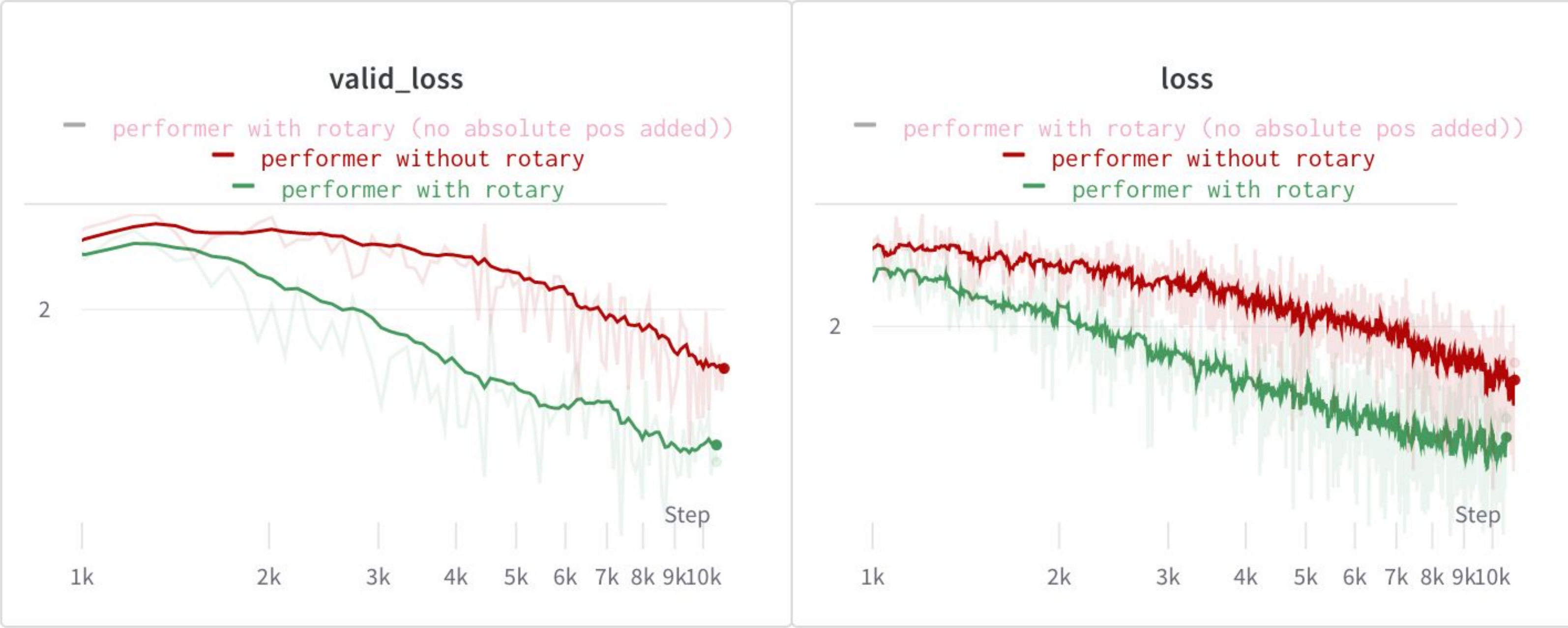


RoPE Embedding [11]

[11] Sun et al. 2023. RoFormer: Enhanced transformer with Rotary Position Embedding

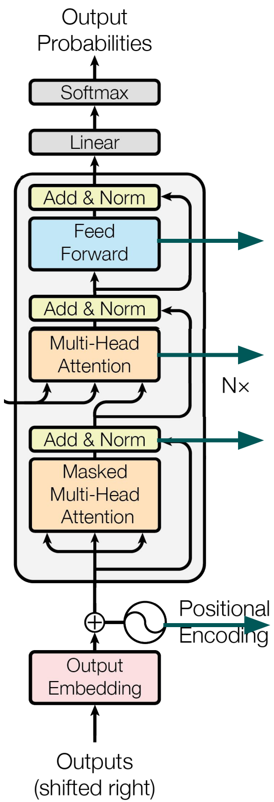
Position Encoding: Rotational Position Embedding

Performance with RoPE



https://blog.eleuther.ai/rotary-embeddings/

LLaMA3's Choice



Feed Forward: SwiGLU

Attention: Grouped-Query Attention

Layer Norm: Pre-Layernorm with RMSNorm

Positional Encoding: RoPE

Questions?