Transformer: Part II (Encoder, Decoder)

CS7150, Spring 2025

Prof. Huaizu Jiang

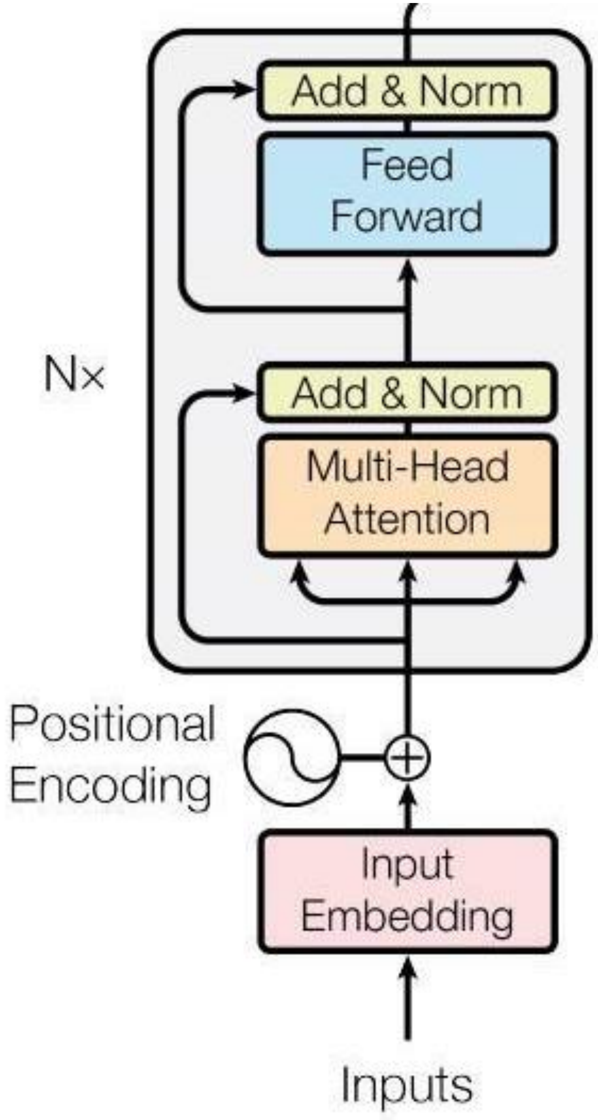
Northeastern University

Reminder: In-class midterm

- On Friday, February 28
- Cover all topics by Friday, February 21
- Work on paper with pens, no coding! Similar to the in-class quizzes.
- A practice exam will be released
- Everyone is expected to show up in the classroom
  - Unless you have valid reasons
  - Contact the instructor if you do by Tuesday, February 21
- If you need accommodation, send your request to DRC@northeastern.edu
  - Someone will work the instructor together to figure out a solution

Recap

Overview of the Transformer Encoder (Cell)

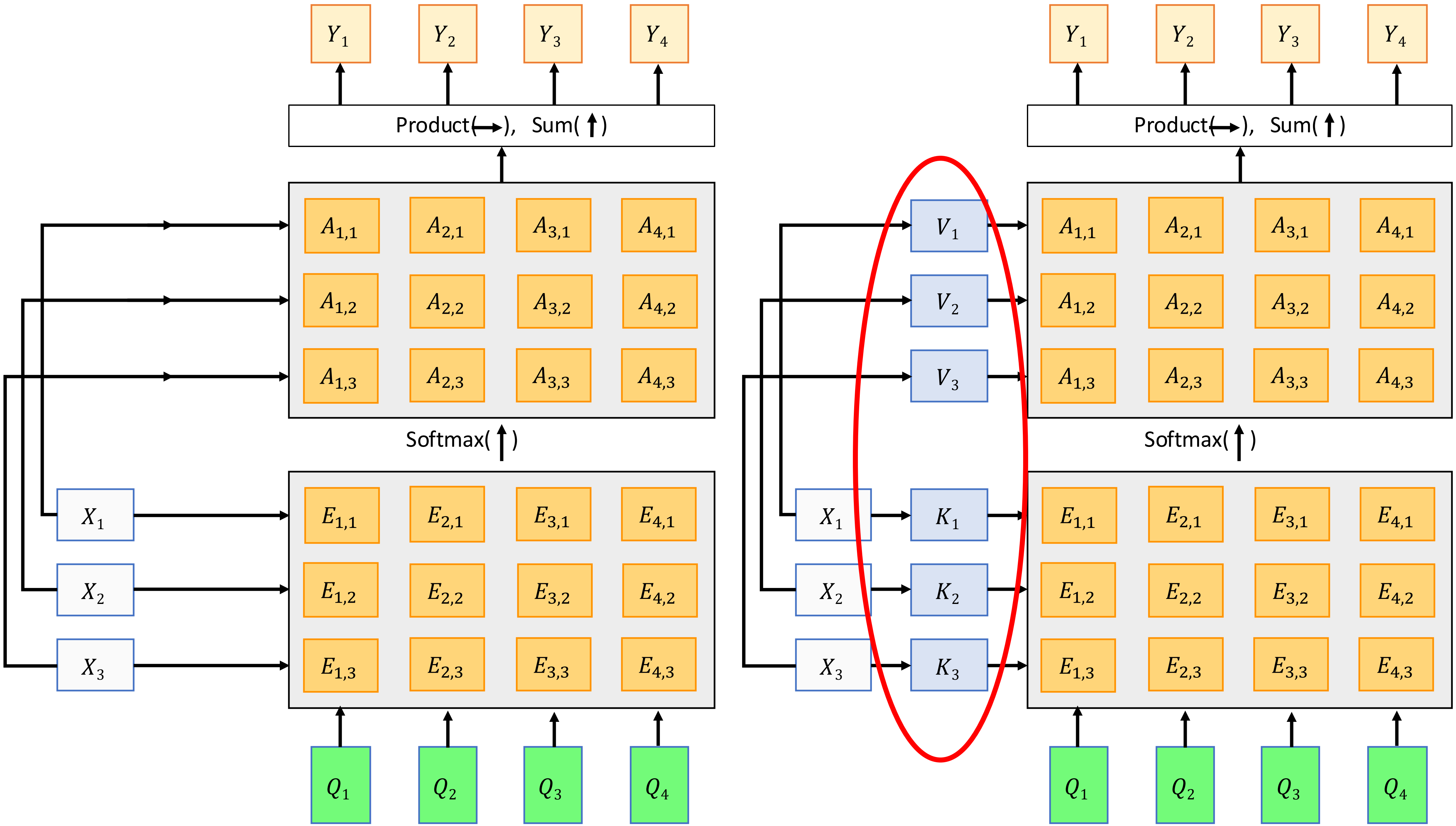


Key components:

- Multi-head (self-)attention
- Positional encoding
- Feedforward network
- Residual connections
- Regularization tricks
  - Dropout
  - LayerNorm

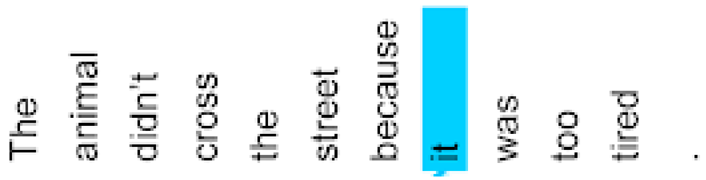
[Vaswani et al., Attention is all you need. NeurIPS 2017]

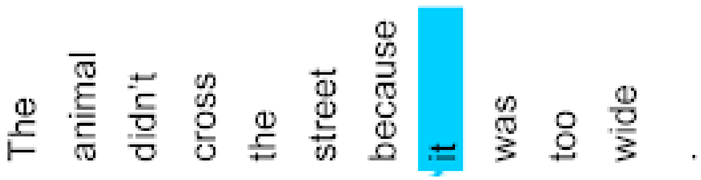
Different Attention Mechanisms



Self-attention

- Used to capture context *within the sequence*





As we are encoding "it", we should focus on "the animal"

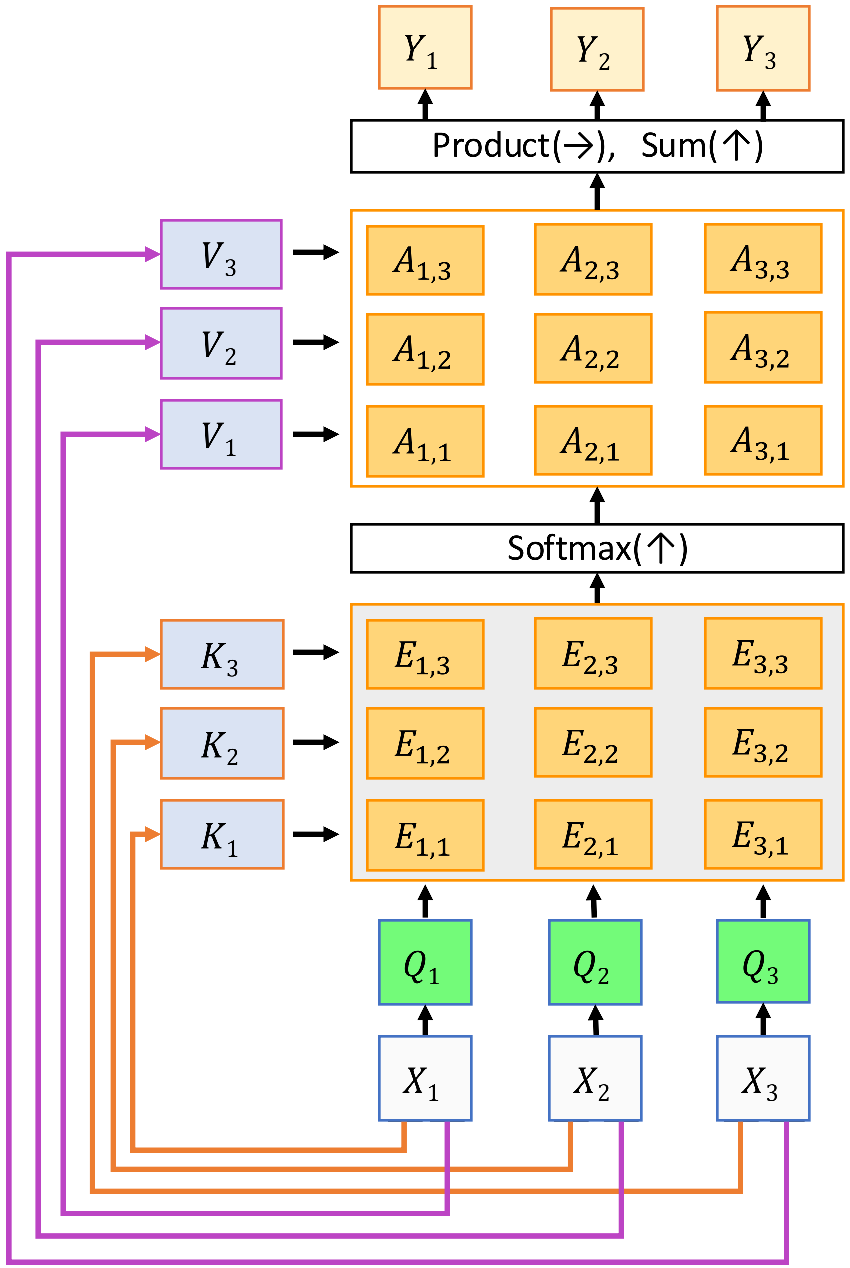
As we are encoding "it", we should focus on "the street"

Slide credit: S. Lazebnik

Image source

Self-attention layer

- Query vectors: $Q = XW_Q$
- Key vectors: $K = XW_K$
- Value vectors: $V = XW_V$
- Similarities: *scaled dot-product attention*
  - $E_{i,j} = \frac{(Q_i \cdot K_j)}{\sqrt{D}}$ or $E = QK^T / \sqrt{D}$
  - (D is the dimensionality of the keys)
- Attn. weights: $A = \mathrm{softmax}(E, \mathrm{dim} = 1)$
- Output vectors:
  - $Y_i = \sum_j A_{i,j} V_j$ or $Y = AV$

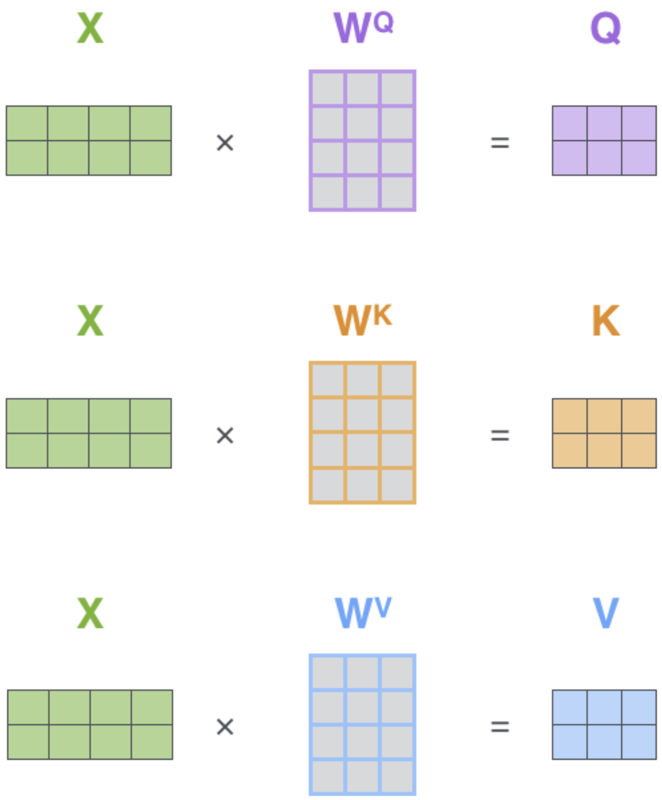


One query per input vector

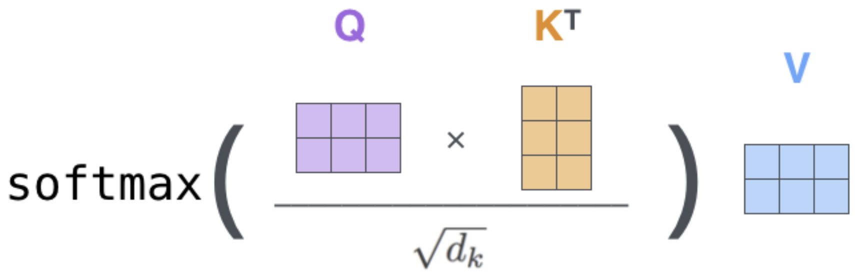
Slide credit: S. Lazebnik

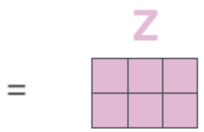
Matrix Calculation of Self-Attention

$X \in R^{L \times C}, W^Q \in R^{C \times D}, \;\; Q \in R^{L \times D}$



Dimension: $R^{L \times L}$



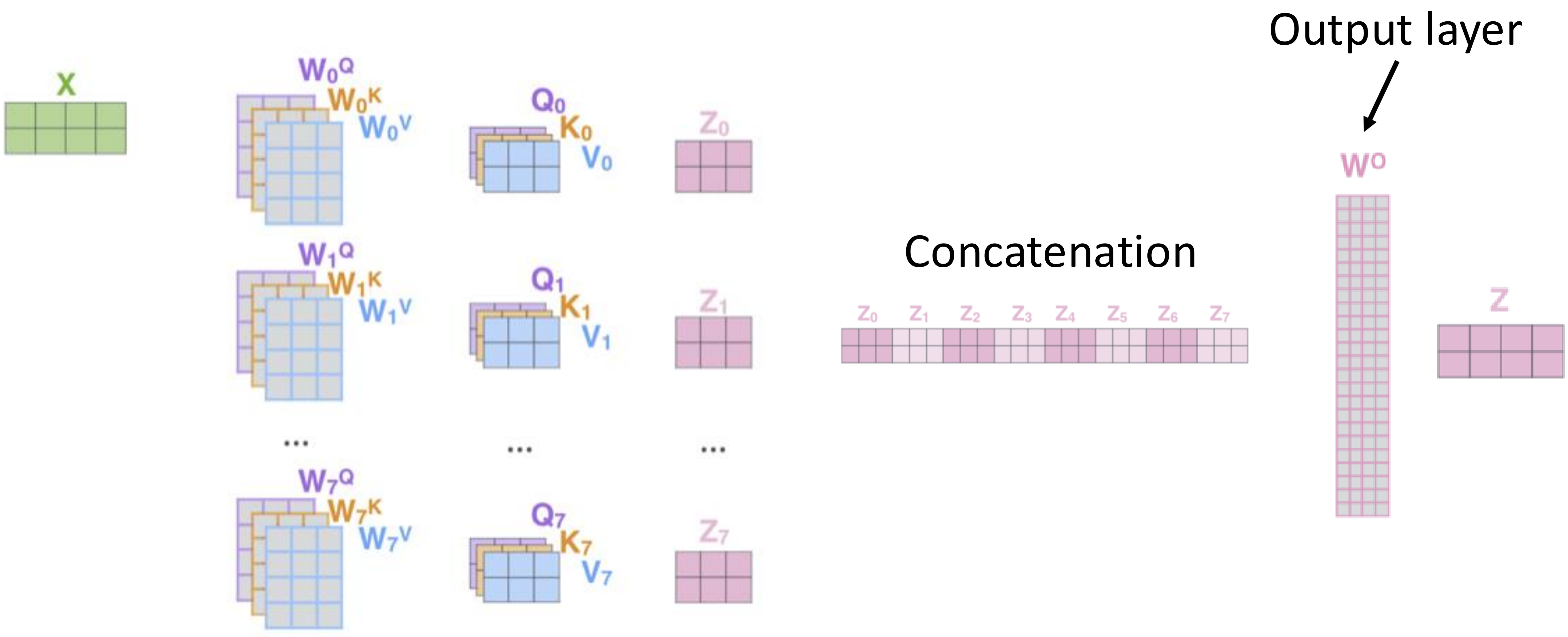


$d_k = D$

In the implementation, we need to process batched data, where $X \in R^{B \times L \times C}$, check torch.matmul for batched matrix multiplication.

https://jalammar.github.io/illustrated-transformer/

Multi-Head Self-Attention

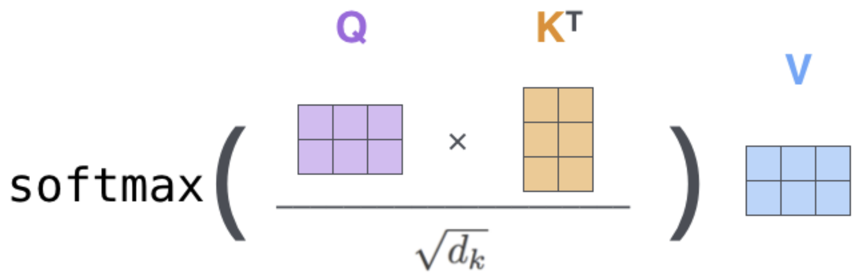


Can we finish the entire operation without any for loops?

https://jalammar.github.io/illustrated-transformer/

Finish Multi-head Attention without For Loops

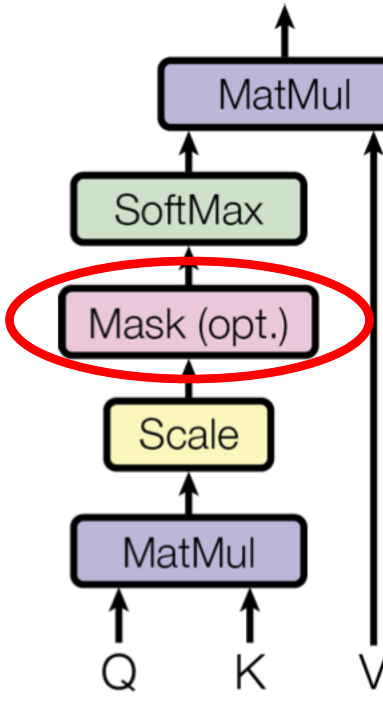
- Linear transformations of Query, Key, and Value
  - Use a big (concatenated) transformation matrix for each of them, respectively
- How about the attention?
  - What are the shapes of Q, K, and V?
    - With and without the multi-head attention



  - The matrix multiplication is defined over a single attention
  - Solution:
    - Reshape the tensors so that the channel dimension is only about a single head
    - Merge the number of heads to the batch dimension

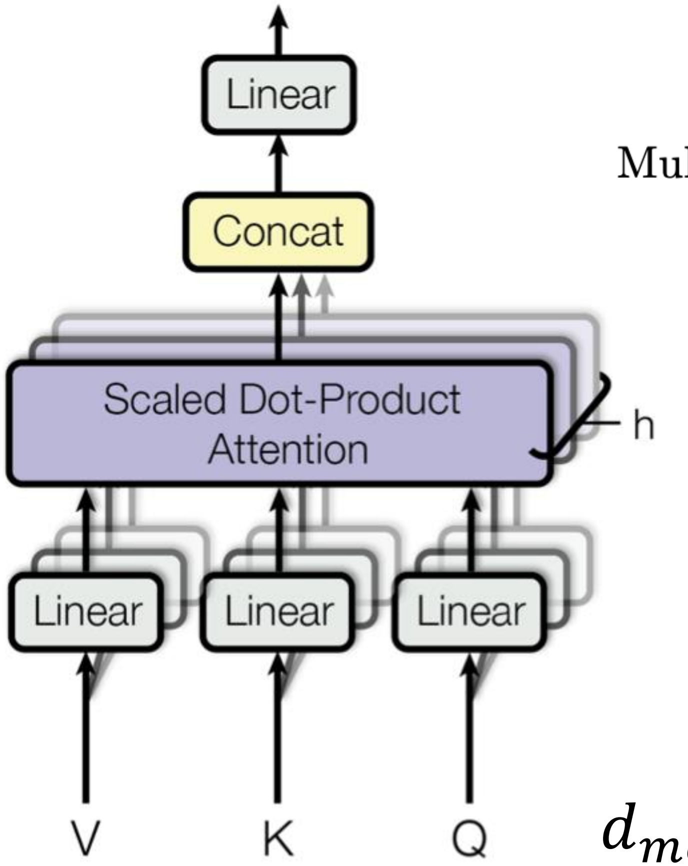
Multi-Head Self-Attention Summary

Scaled Dot-Product Attention



Will be clear when we talk about the decoder

Multi-Head Attention



$$\mathrm{head_i} = \mathrm{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

$$\mathrm{MultiHead}(Q, K, V) = \mathrm{Concat}(\mathrm{head_1}, ..., \mathrm{head_h})W^O$$

$$W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}, W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$$

$$W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$$

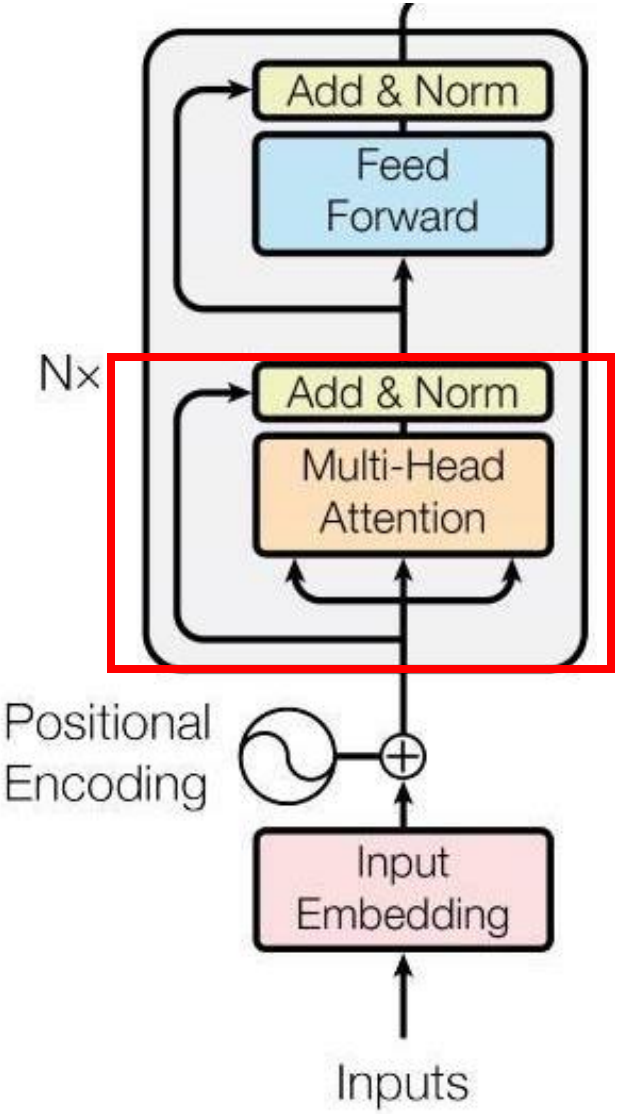
$$W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$$

$$d_k = d_v = d_{\text{model}}/h$$

d_{model}: dimension of input embeddings
h: number of attention heads

[Vaswani et al., Attention is all you need. NeurIPS 2017]

Residual Connections, Dropout, and Normalization



$$X_{att} = \mathrm{MHA}(X)$$
$$X_{att} = \mathrm{Dropout}(X_{att}, p = 0.1)$$
$$X = \mathrm{LayerNorm}(X + X_{att})$$

Recall: Layer Normalization

1. Variable lengths in the input.
2. The recurrent nature of RNN.

Computing of μ, σ is independent of the batch and length dimension.

But the learnable parameters γ, β are still for each channel.

Largely adopted in Transformer.

Layer Normalization for **recurrent** networks

$$x : N \times L \times C$$

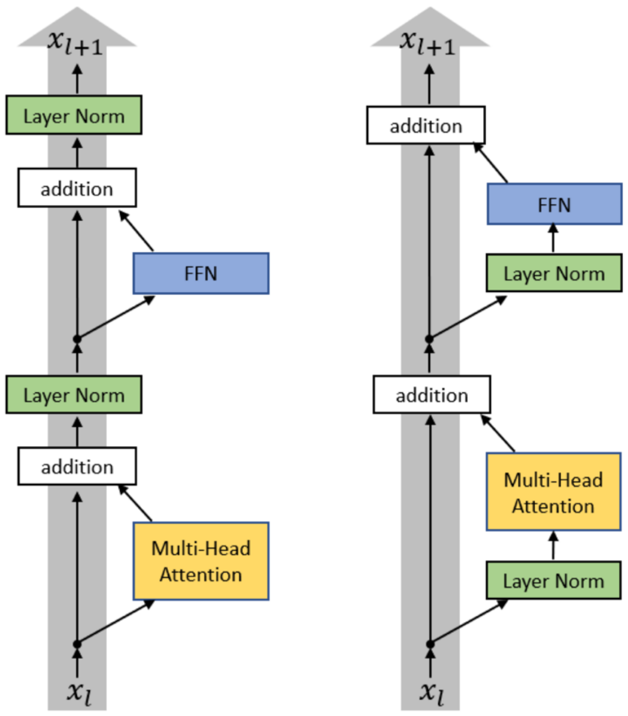
$$\mu, \sigma : N \times L \times 1$$

$$\gamma, \beta : 1 \times 1 \times C$$

$$y = \frac{(x-\mu)}{\sigma}\gamma + \beta$$

On the LayerNorm in Transformer

https://arxiv.org/pdf/2002.04745



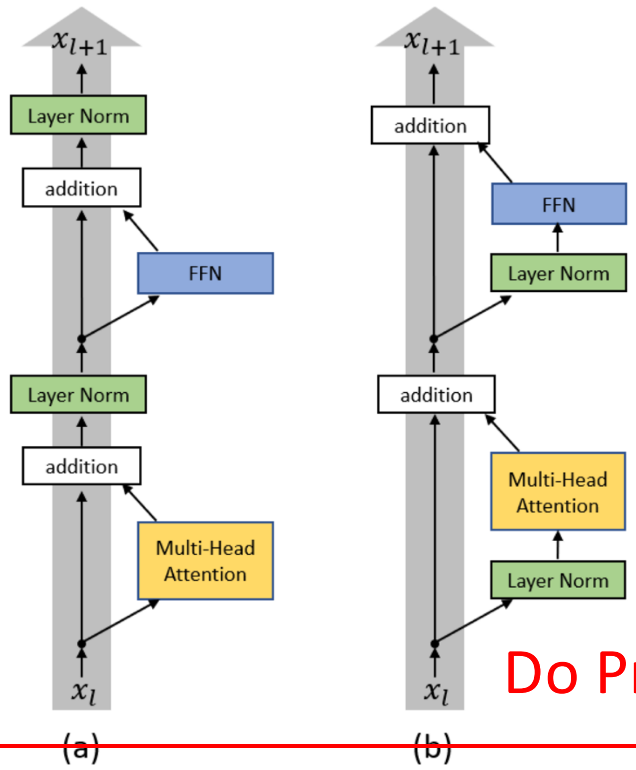
(a) (b)

Implement Post-Norm in PA3

Post-LN Transformer	Pre-LN Transformer
$x_{l,i}^{post,1} = \text{MultiHeadAtt}(x_{l,i}^{post}, [x_{l,1}^{post}, \cdots, x_{l,n}^{post}])$	$x_{l,i}^{pre,1} = \text{LayerNorm}(x_{l,i}^{pre})$
$x_{l,i}^{post,2} = x_{l,i}^{post} + x_{l,i}^{post,1}$	$x_{l,i}^{pre,2} = \text{MultiHeadAtt}(x_{l,i}^{pre,1}, [x_{l,1}^{pre,1}, \cdots, x_{l,n}^{pre,1}])$
$x_{l,i}^{post,3} = \text{LayerNorm}(x_{l,i}^{post,2})$	$x_{l,i}^{pre,3} = x_{l,i}^{pre} + x_{l,i}^{pre,2}$
$x_{l,i}^{post,4} = \text{ReLU}(x_{l,i}^{post,3} W^{1,l} + b^{1,l}) W^{2,l} + b^{2,l}$	$x_{l,i}^{pre,4} = \text{LayerNorm}(x_{l,i}^{pre,3})$
$x_{l,i}^{post,5} = x_{l,i}^{post,3} + x_{l,i}^{post,4}$	$x_{l,i}^{pre,5} = \text{ReLU}(x_{l,i}^{pre,4} W^{1,l} + b^{1,l}) W^{2,l} + b^{2,l}$
$x_{l+1,i}^{post} = \text{LayerNorm}(x_{l,i}^{post,5})$	$x_{l+1,i}^{pre} = x_{l,i}^{pre,5} + x_{l,i}^{pre,3}$
	Final LayerNorm: $x_{Final,i}^{pre} \leftarrow \text{LayerNorm}(x_{L+1,i}^{pre})$

On the LayerNorm in Transformer

https://arxiv.org/pdf/2002.04745



Do Pre-Norm in practice

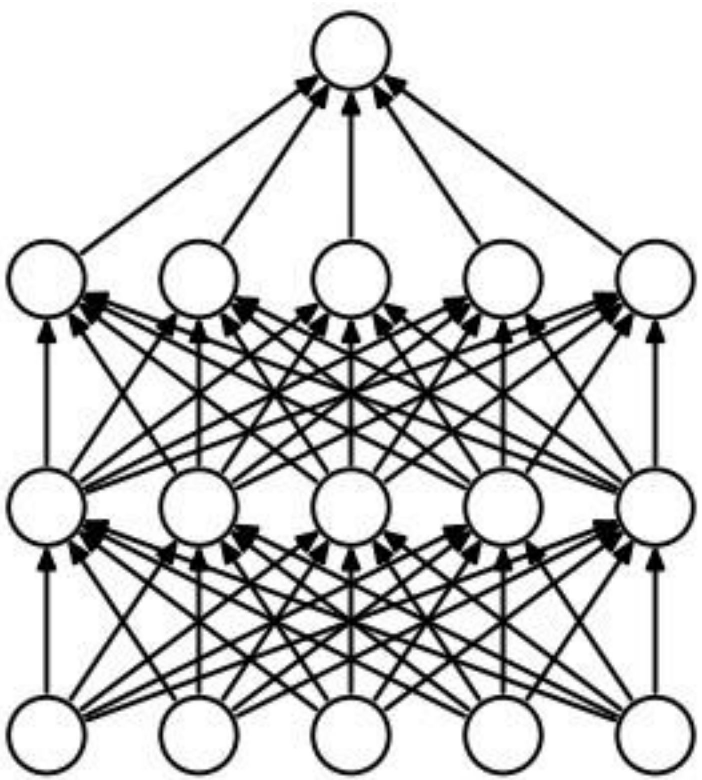
Post-LN Transformer	Pre-LN Transformer
$x^{post,1}_{l,i} = \text{MultiHeadAtt}(x^{post}_{l,i}, [x^{post}_{l,1}, \cdots, x^{post}_{l,n}])$	$x^{pre,1}_{l,i} = \text{LayerNorm}(x^{pre}_{l,i})$
$x^{post,2}_{l,i} = x^{post}_{l,i} + x^{post,1}_{l,i}$	$x^{pre,2}_{l,i} = \text{MultiHeadAtt}(x^{pre,1}_{l,i}, [x^{pre,1}_{l,1}, \cdots, x^{pre,1}_{l,n}])$
$x^{post,3}_{l,i} = \text{LayerNorm}(x^{post,2}_{l,i})$	$x^{pre,3}_{l,i} = x^{pre}_{l,i} + x^{pre,2}_{l,i}$
$x^{post,4}_{l,i} = \text{ReLU}(x^{post,3}_{l,i}W^{1,l} + b^{1,l})W^{2,l} + b^{2,l}$	$x^{pre,4}_{l,i} = \text{LayerNorm}(x^{pre,3}_{l,i})$
$x^{post,5}_{l,i} = x^{post,3}_{l,i} + x^{post,4}_{l,i}$	$x^{pre,5}_{l,i} = \text{ReLU}(x^{pre,4}_{l,i}W^{1,l} + b^{1,l})W^{2,l} + b^{2,l}$
$x^{post}_{l+1,i} = \text{LayerNorm}(x^{post,5}_{l,i})$	$x^{pre}_{l+1,i} = x^{pre,5}_{l,i} + x^{pre,3}_{l,i}$
	Final LayerNorm: $x^{pre}_{Final,i} \leftarrow \text{LayerNorm}(x^{pre}_{L+1,i})$

Today’s Class

- Transformer Encoder
- Transformer Decoder

Regularization: **Dropout**

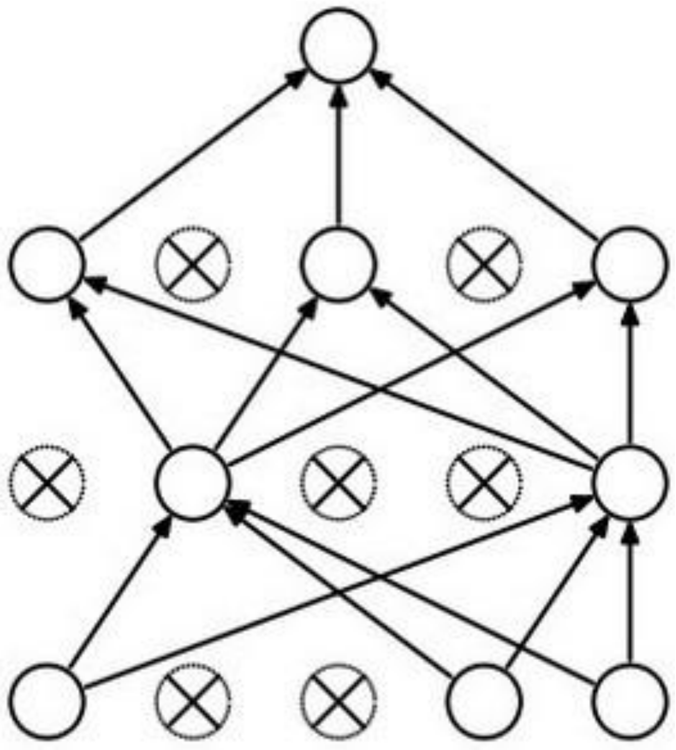
“randomly set some neurons to zero in the forward pass”



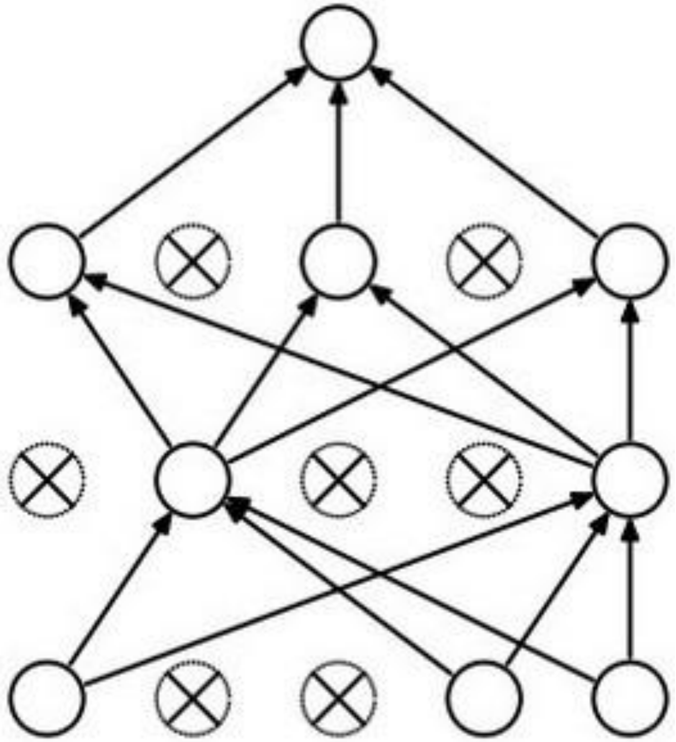
(a) Standard Neural Net

Randomly set the output value of some neurons to be 0

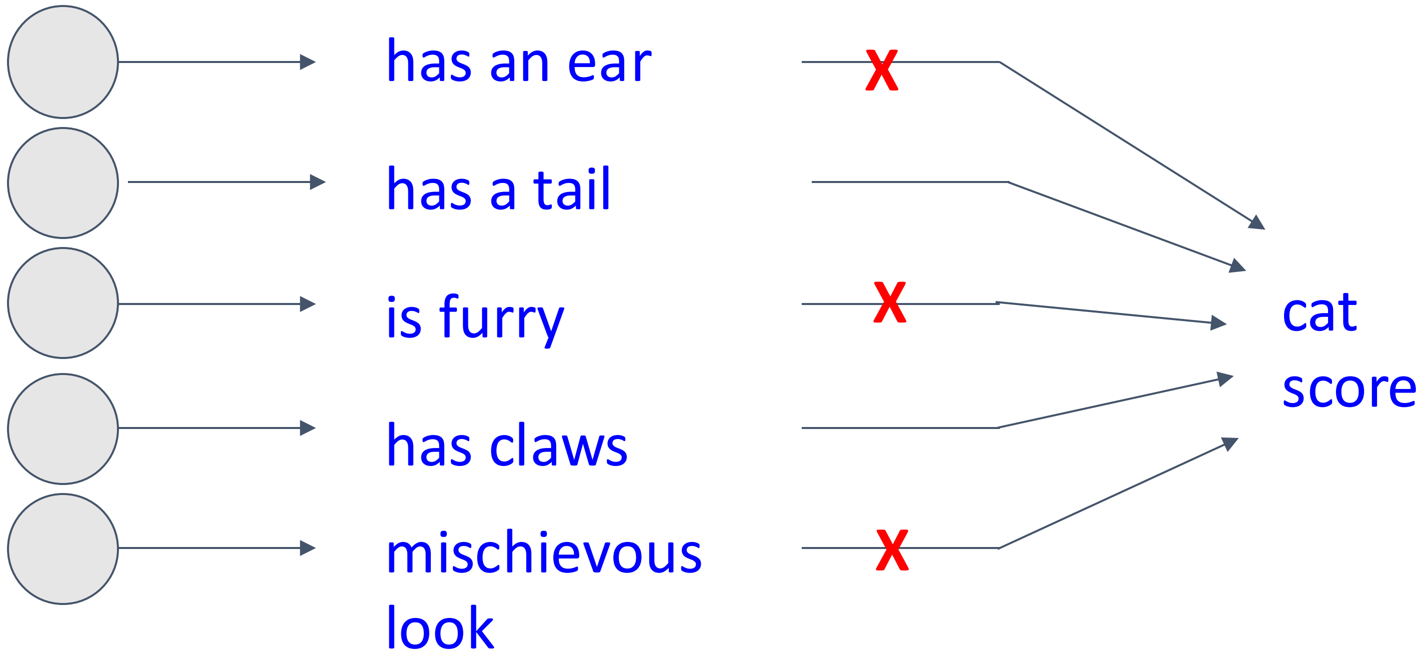
Waaaait a second…
How could this possibly be a good idea?



Waaaait a second…
How could this possibly be a good idea?



Forces the network to have a redundant representation.



Slide credit: E. Learned-Miller, originally developed by Andrej Karpathy, Justin Johnson, Fei-Fei Li

Training with occlusions?

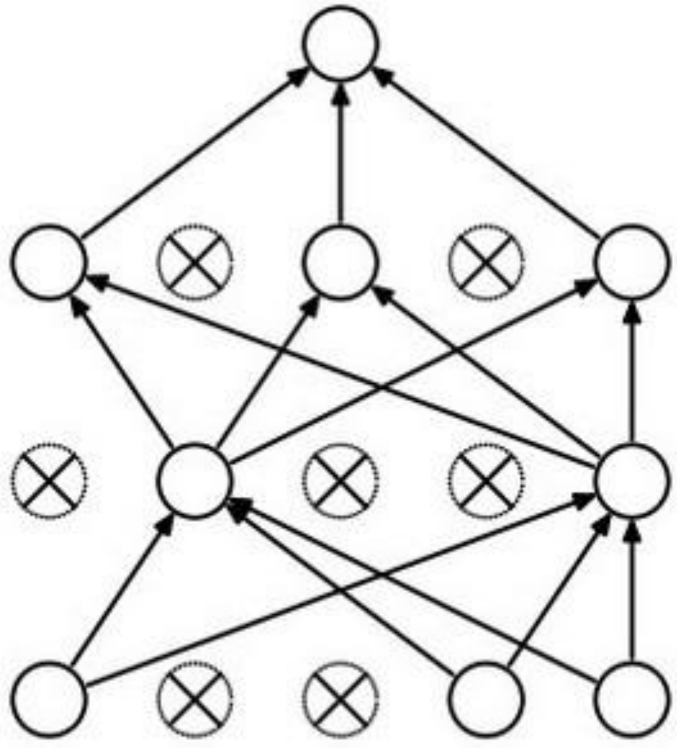







Slide credit: E. Learned-Miller, originally developed by Andrej Karpathy, Justin Johnson, Fei-Fei Li

Waaaait a second…
How could this possibly be a good idea?



Another interpretation:

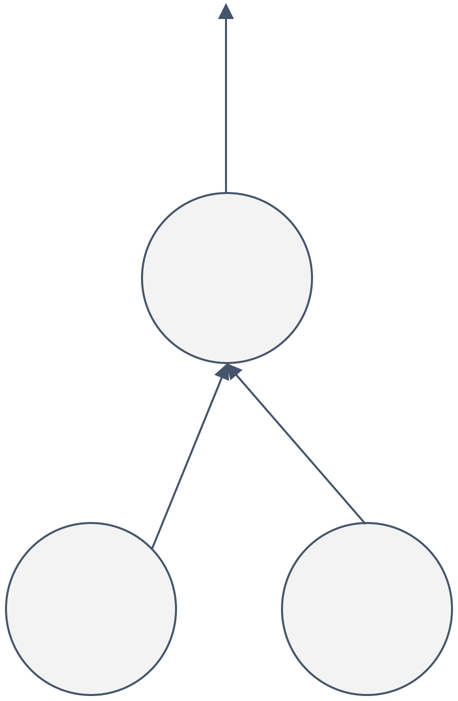
Dropout is training a large ensemble of models (that share parameters).

Each binary mask is one model, gets trained on only ~one datapoint.

Dropout: Test Time

Can in fact do this with a single forward pass! (approximately)

Leave all input neurons turned on (no dropout).



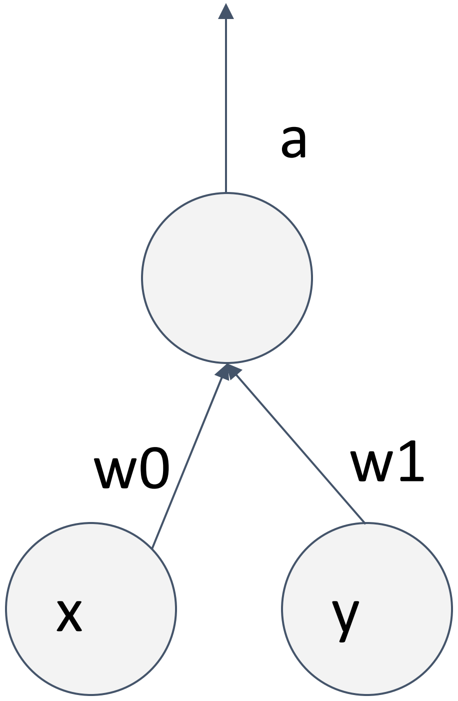
Q: Suppose that with all inputs present at test time the output of this neuron is *a*.

What would its output be during training time, in expectation? (e.g. if p = 0.5)

Dropout: Test Time

Can in fact do this with a single forward pass! (approximately)

Leave all input neurons turned on (no dropout).



during test: $\boldsymbol{a = w_0 * x + w_1 * y}$

during train:

$$E[a] = \tfrac{1}{4} * (w_0{*}0 + w_1{*}0 + w_0{*}0 + w_1{*}y + w_0{*}x + w_1{*}0 + w_0{*}x + w_1{*}y)$$

$$= \tfrac{1}{4} * (2\, w_0{*}x + 2\, w_1{*}y)$$

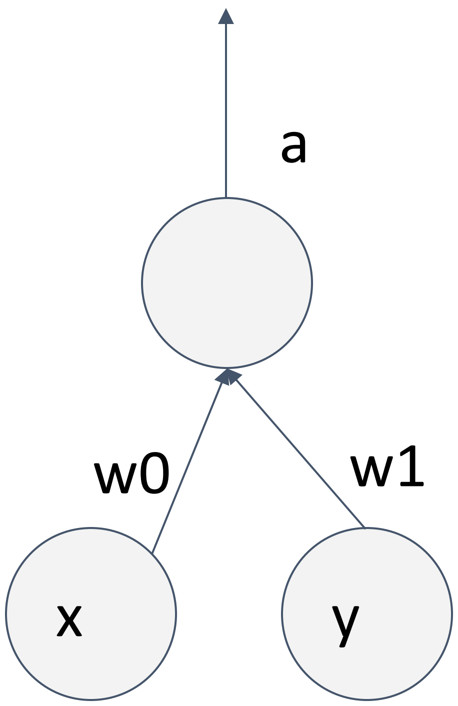
$$\boldsymbol{= \tfrac{1}{2} * (w_0{*}x + w_1{*}y)}$$

Slide credit: E. Learned-Miller, originally developed by Andrej Karpathy, Justin Johnson, Fei-Fei Li

At test time....

Can in fact do this with a single forward pass! (approximately)

Leave all input neurons turned on (no dropout).



during test: $\boldsymbol{a = w_0 * x + w_1 * y}$

during train:

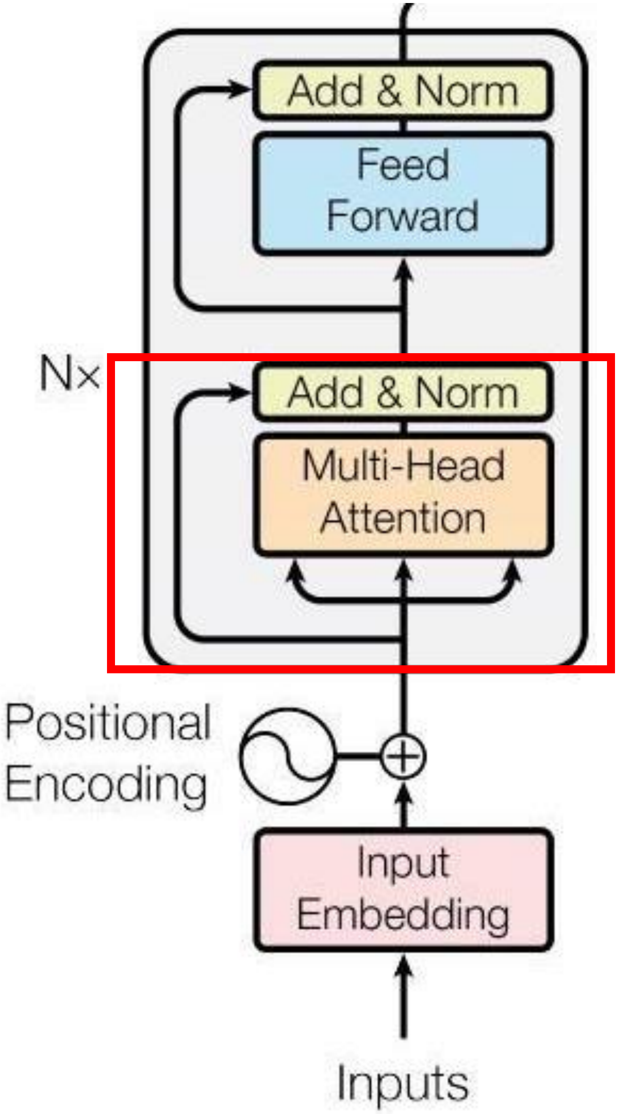
$$E[a] = \tfrac{1}{4} * (w_0*0 + w_1*0 + w_0*0 + w_1*y + w_0*x + w_1*0 + w_0*x + w_1*y)$$

$$= \tfrac{1}{4} * (2\, w_0*x + 2\, w_1*y)$$

$$= \boldsymbol{\tfrac{1}{2} * (w_0*x + w_1*y)}$$

With p=0.5, using all inputs in the forward pass would inflate the activations by 2x from what the network was "used to" during training!
=> Have to compensate by scaling the activations back down by ½

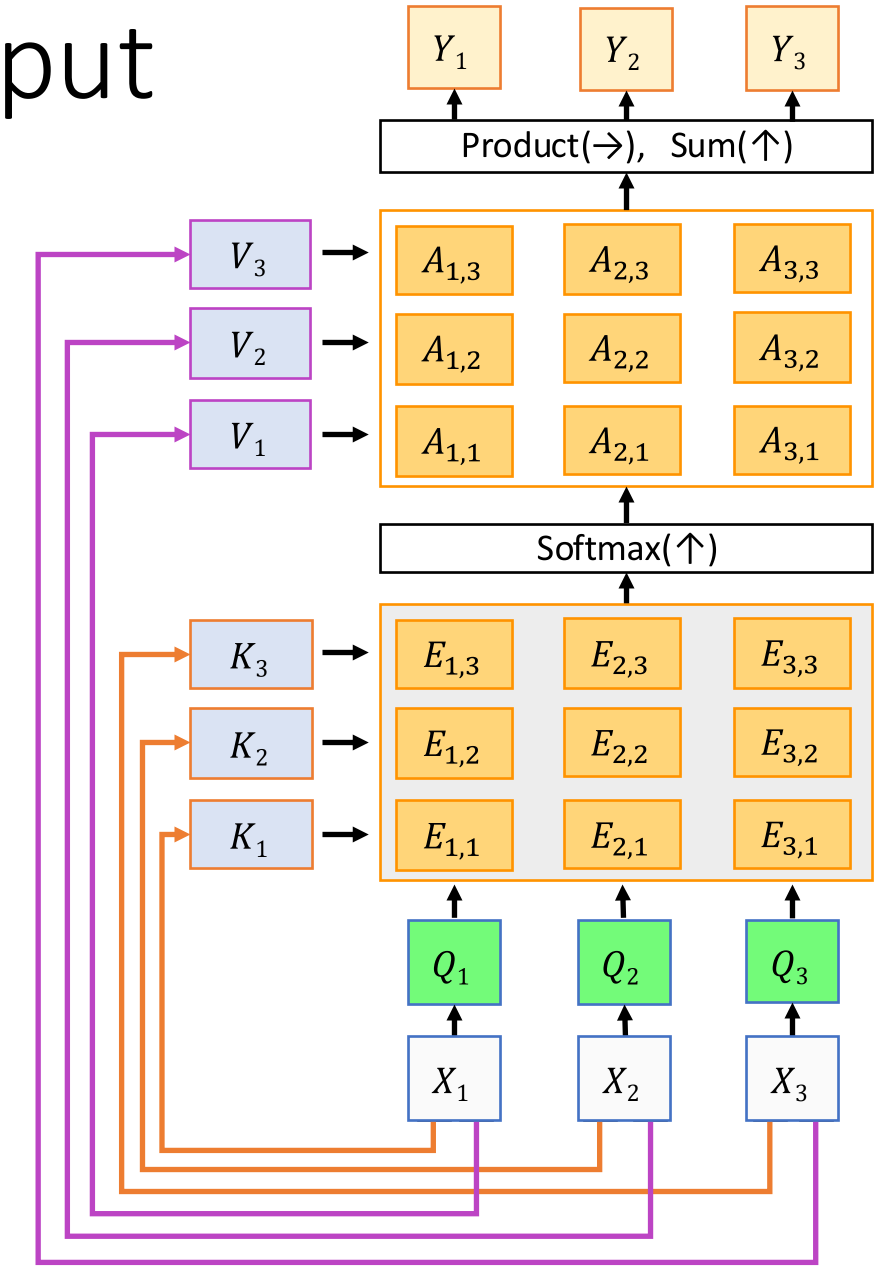
Residual Connections, Dropout, and Normalization



$$X_{att} = \mathrm{MHA}(X)$$
$$X_{att} = \mathrm{Dropout}(X_{att}, p = 0.1)$$
$$X = \mathrm{LayerNorm}(X + X_{att})$$

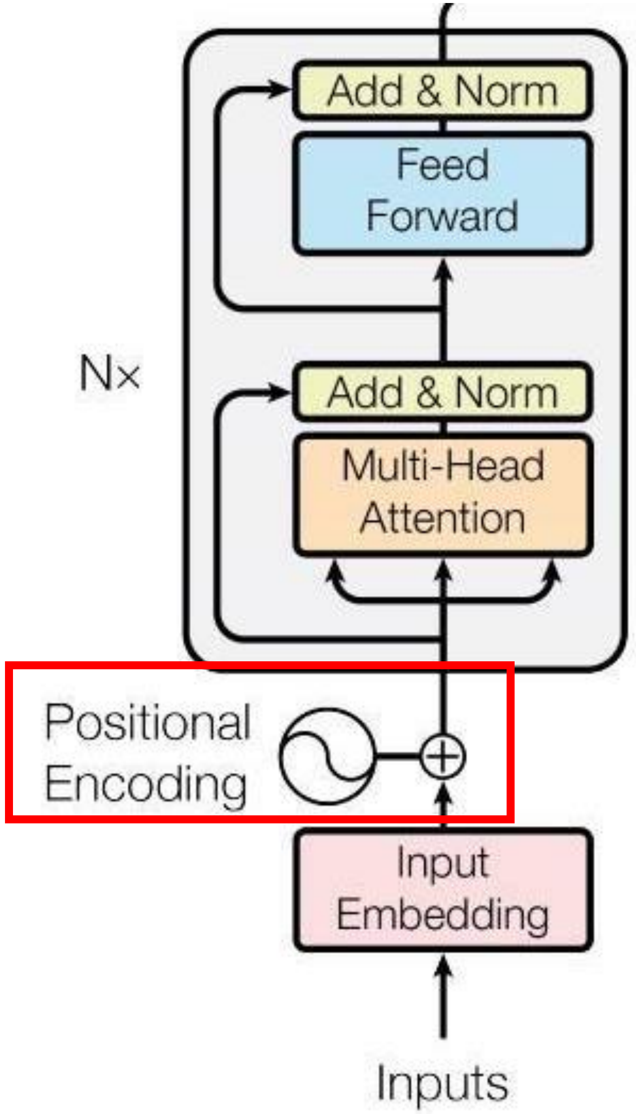
Capturing the Order of the Input

- Do the embeddings of each token/input change if we randomly permute the input?
- Is it good or bad?



One query per input vector

Augmenting the MHA with Positional Encoding



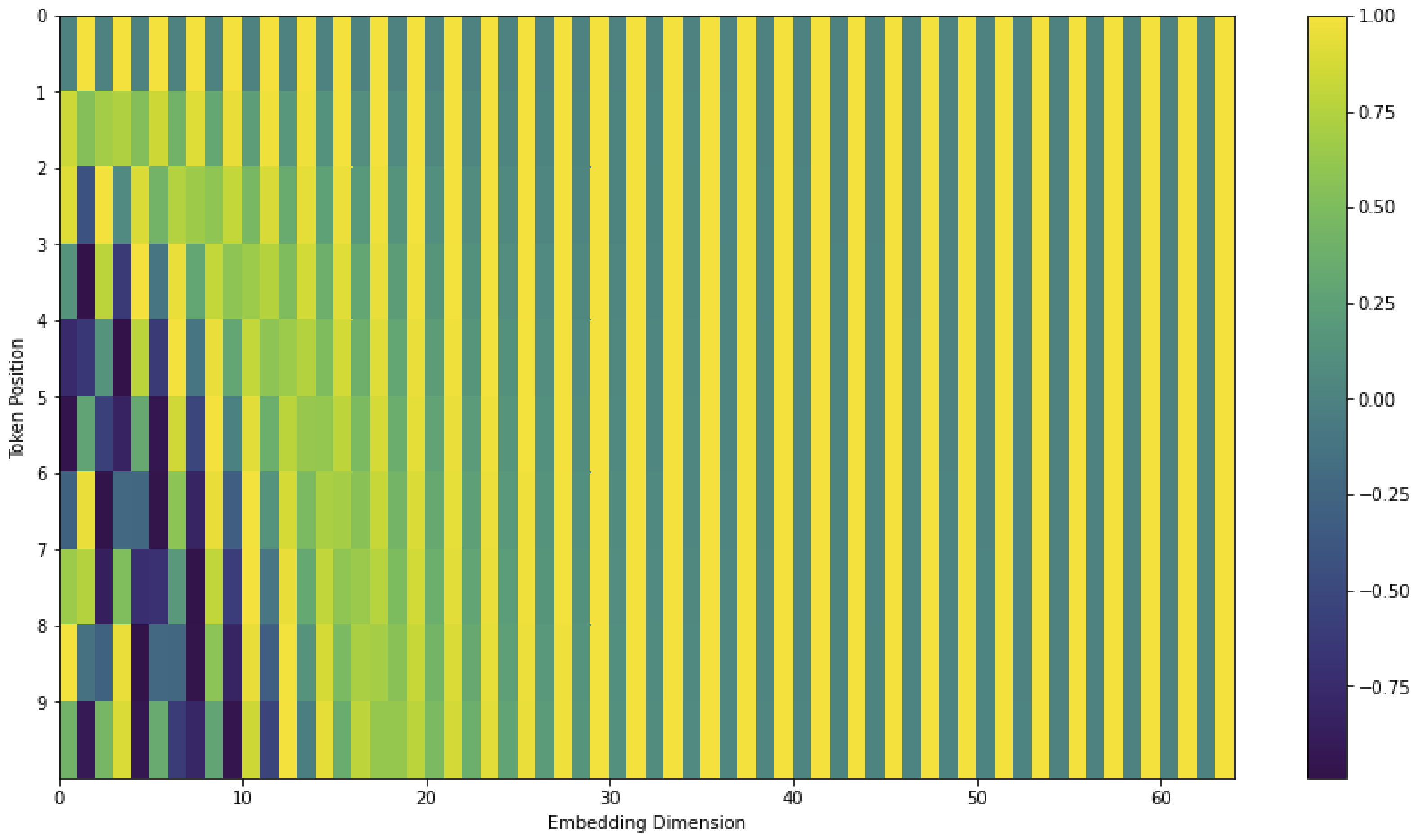
$$X = X + PE(X)$$

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

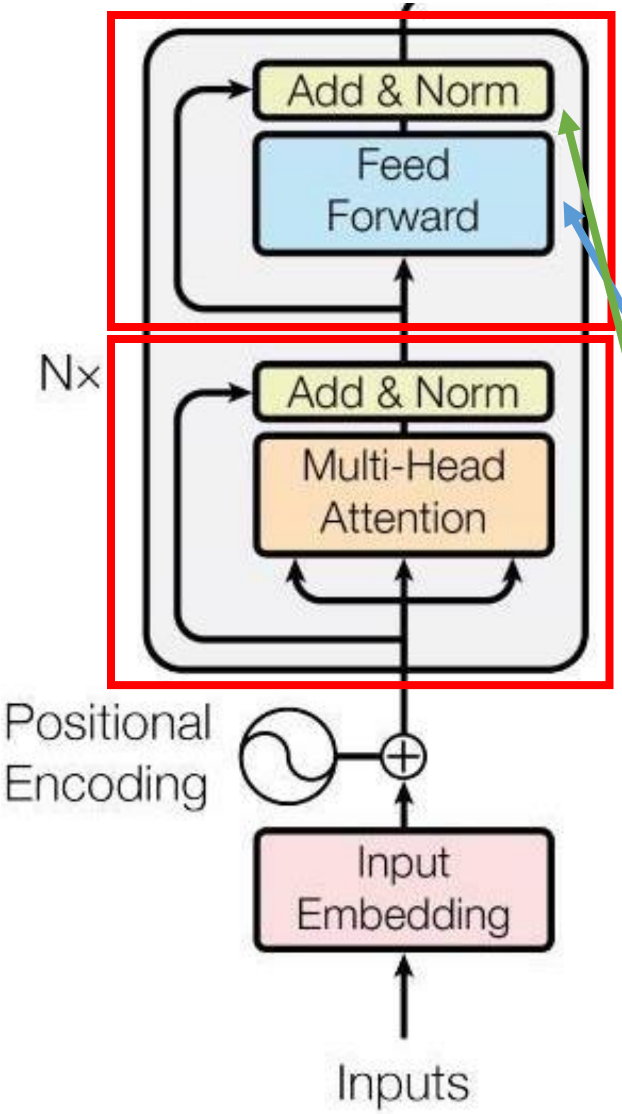
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

- *PE* has the same dimension with the input embeddings of each token.
- *pos*: Position of each token in the input, [*0, L-1*]
- *i*: index of the embeddings, [*0*, d_{model} *-1*]
- What does 10000 mean here?
  - The maximum input length
  - The PE will repeat after the 10000th token

What Does Positional Encoding Look Like?



Feedforward Network

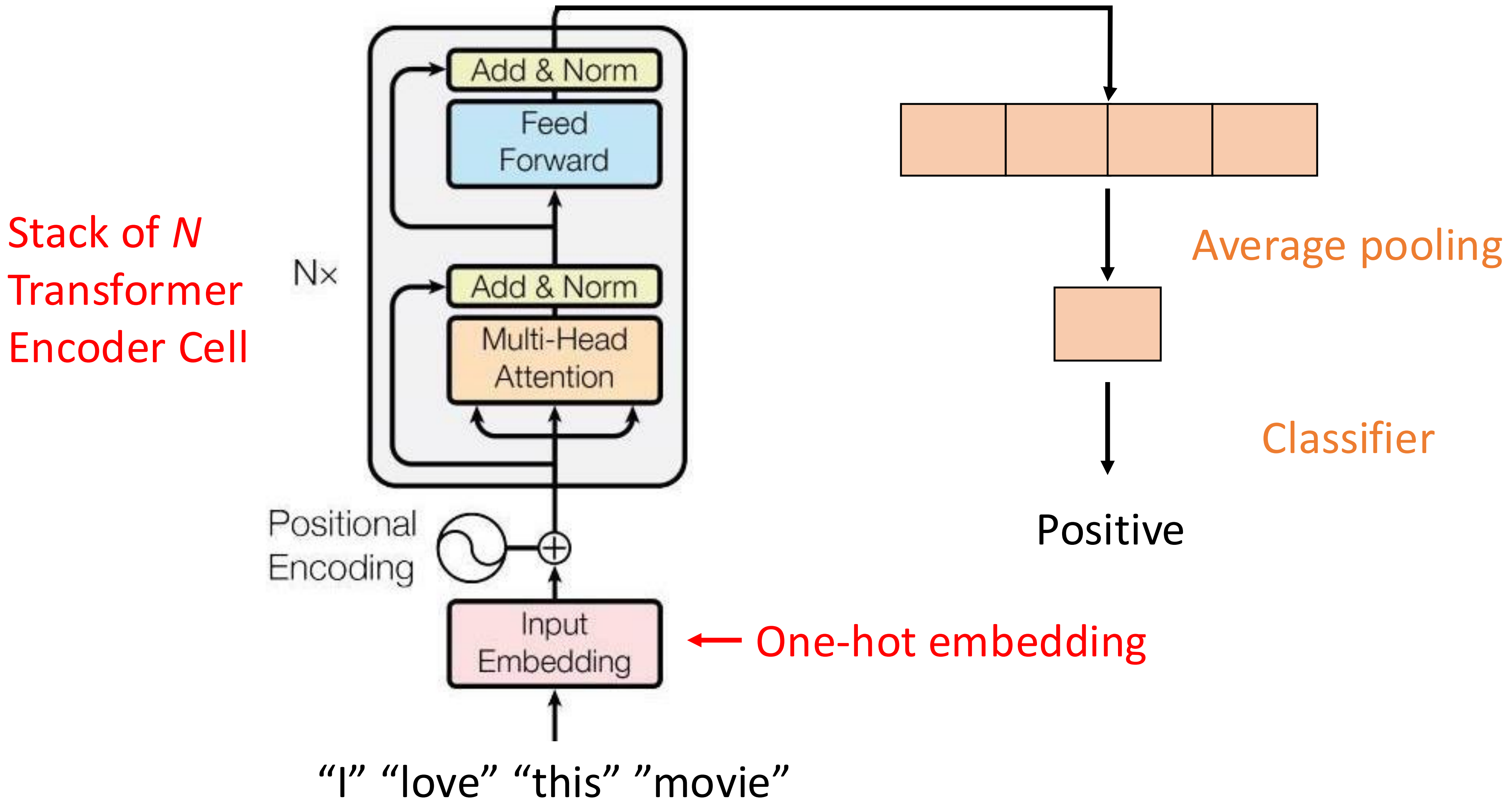


Are there any non-linearities in the lower part?

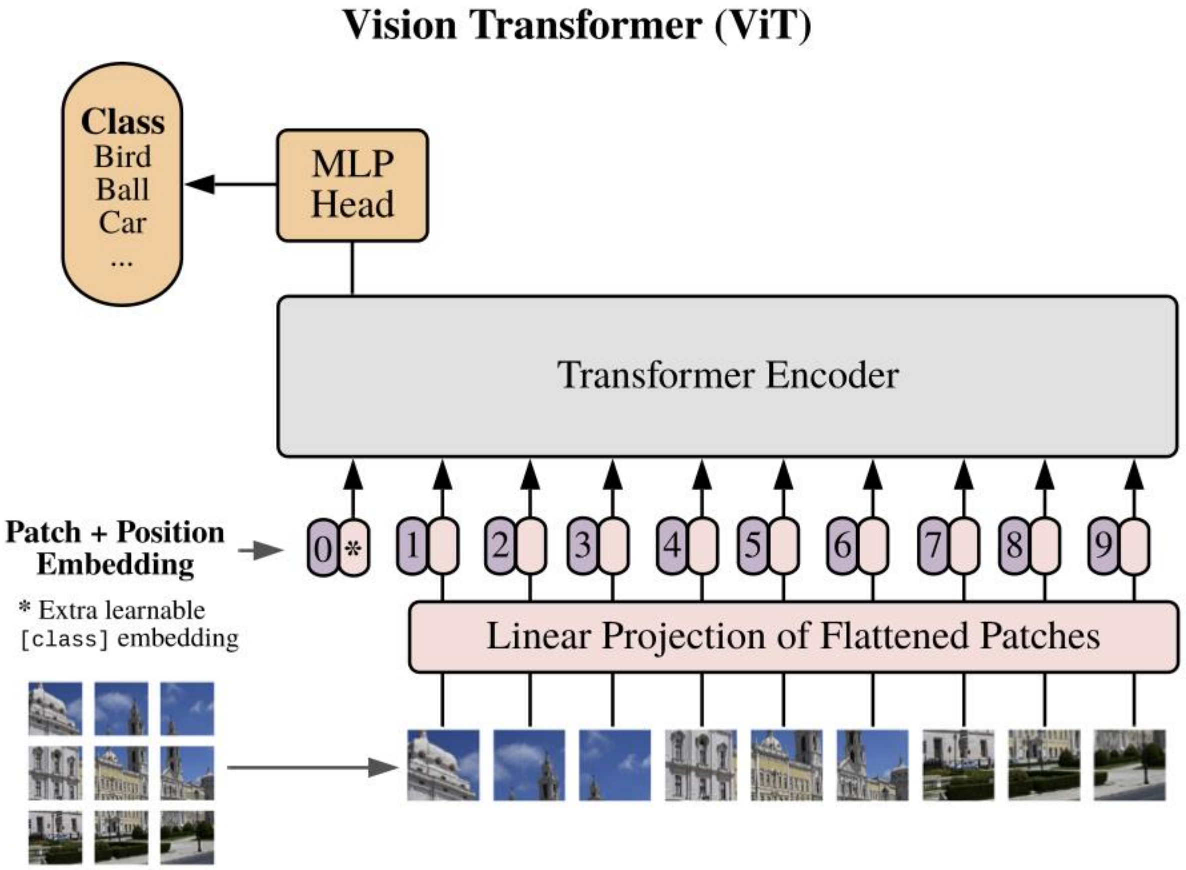
A two-layer fully-connected network (one hidden layer).

Residual connections, dropout, and normalization work in a similar way to the multi-head self-attention module

Transformer Encoder for Text Classification



Transformer Encoder for Image Classification



How to represent each patch?

Flattened RGB pixels

Extra credits in PA3:

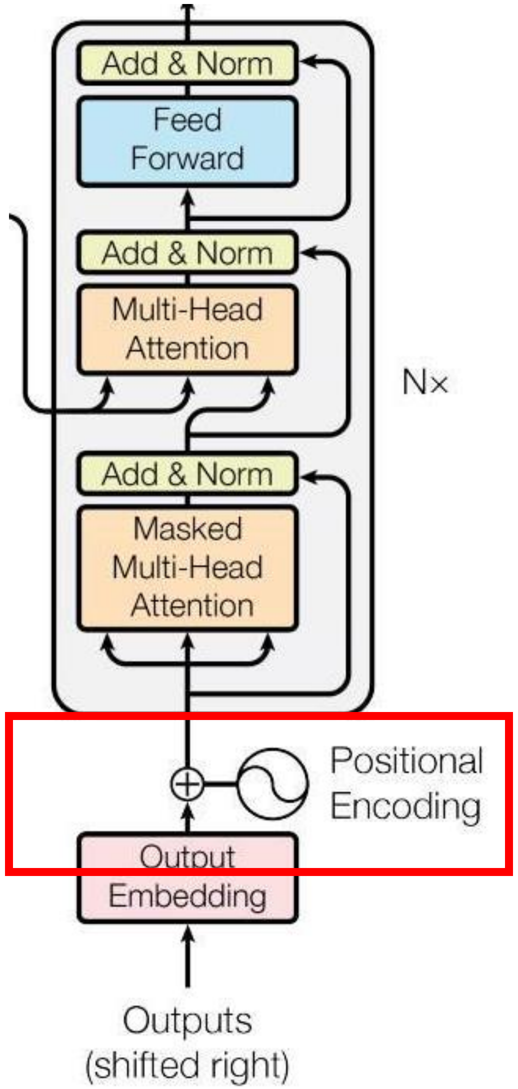
- Use the special [CLASS] token to aggregate the image's information
- Alternatively, we can get embeddings of each patch and do average pooling as in the previous example
- Positional encoding: 1D is sufficient.
- Free of inductive bias
- Can be easily integrated with tokens from other modalities (e.g., text, audio)

[Dosovitskiy et al., An image is worth 16x16 words. ICLR 2021]

Transformer Decoder: Autoregressive Target Sequence Generation

Output (encodings) from the encoder

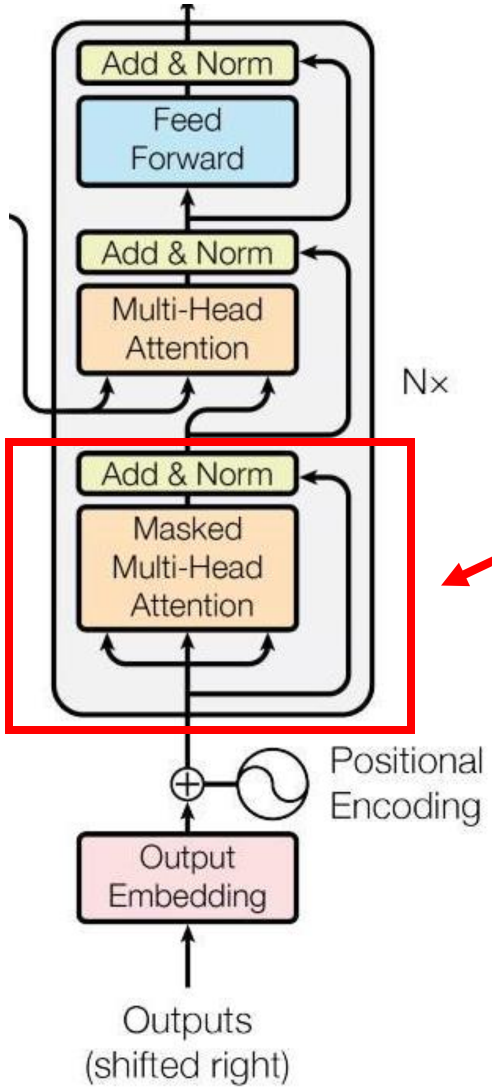
What is the shape?



Transformer Decoder: Autoregressive Target Sequence Generation

Output (encodings) from the encoder

What is the shape?



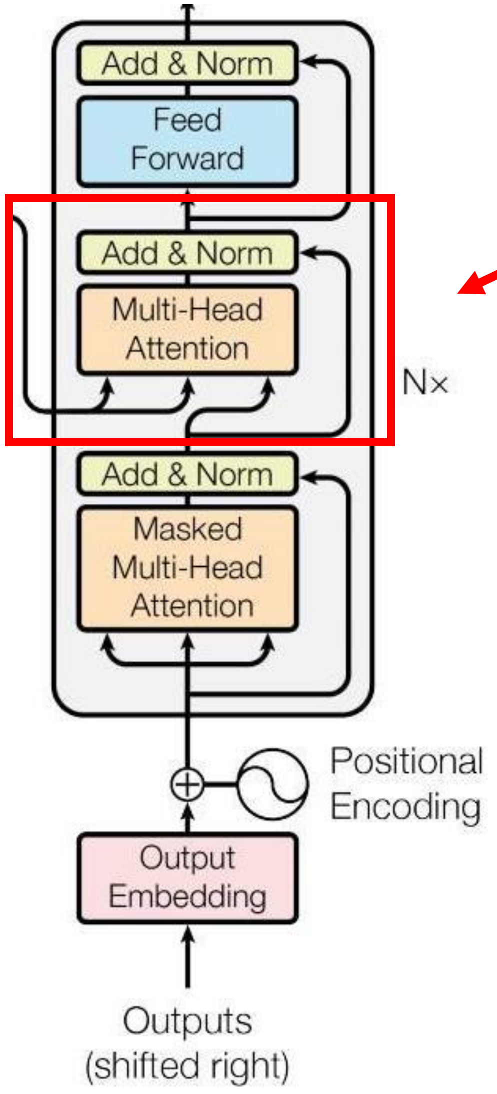
What is this module doing?

Self-attention: get embeddings of the output.

Transformer Decoder: Autoregressive Target Sequence Generation

Output (encodings) from the encoder

What is the shape?

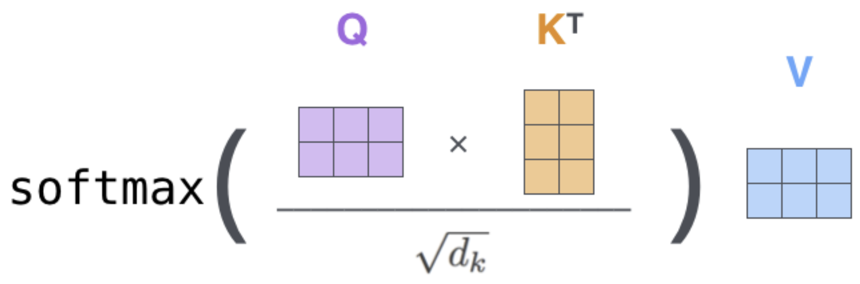


What is this module doing?

Query: from the decoder

Key, Value: from the encoder

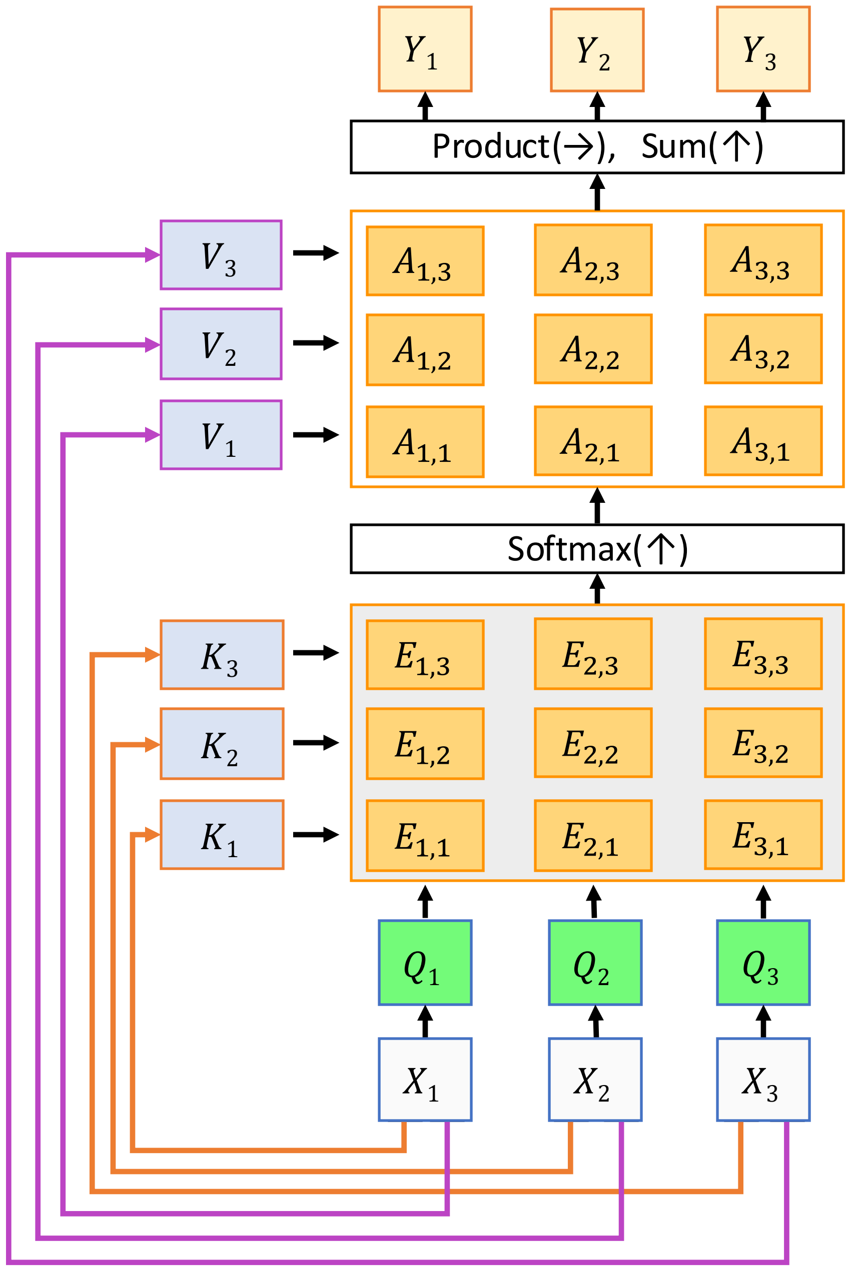
Attention:



Cross-attention: borrow the embeddings from the input.

Masked self-attention layer

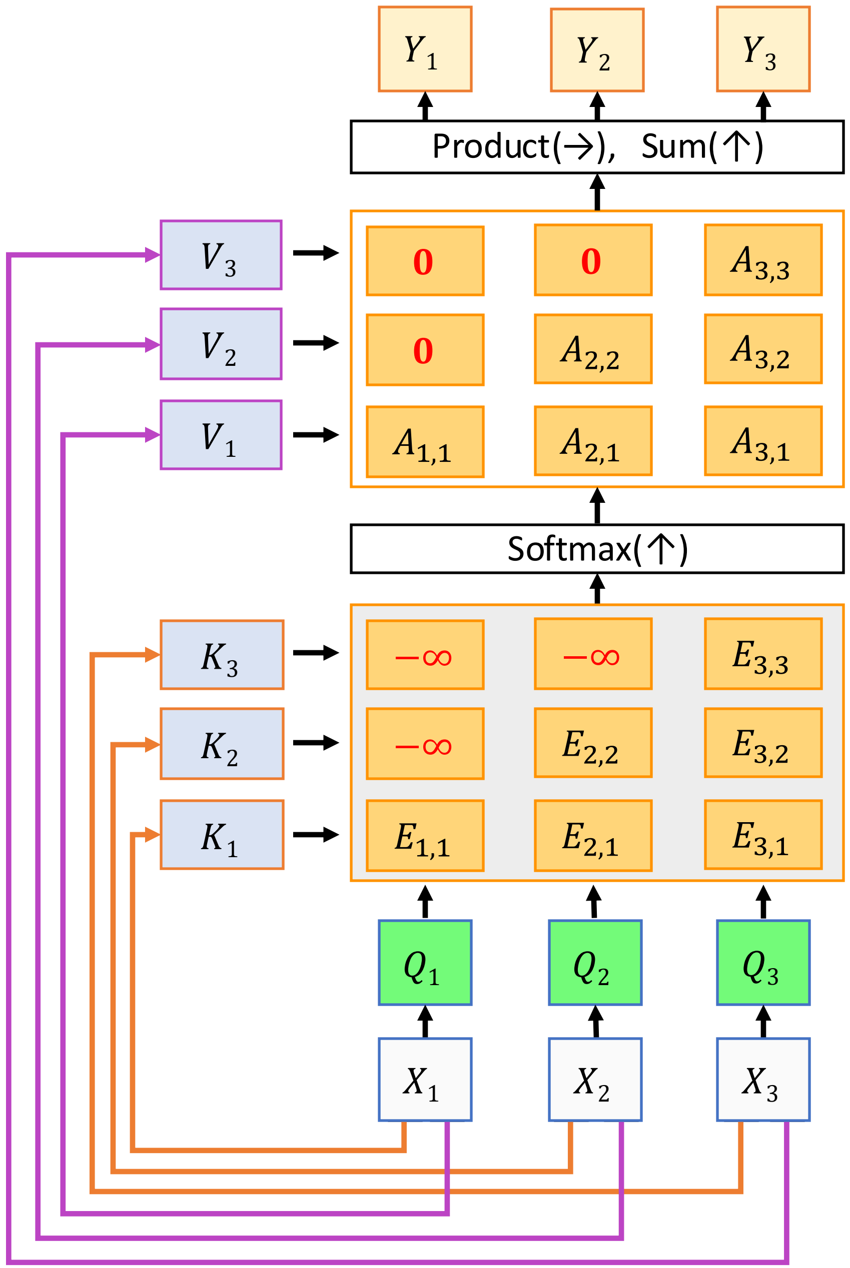
- We usually put complete sentences in a batch to train the model
- The decoder should not “look ahead” in the output sequence during training
- Otherwise, the model can simply cheat



Slide credit: S. Lazebnik

Masked self-attention layer

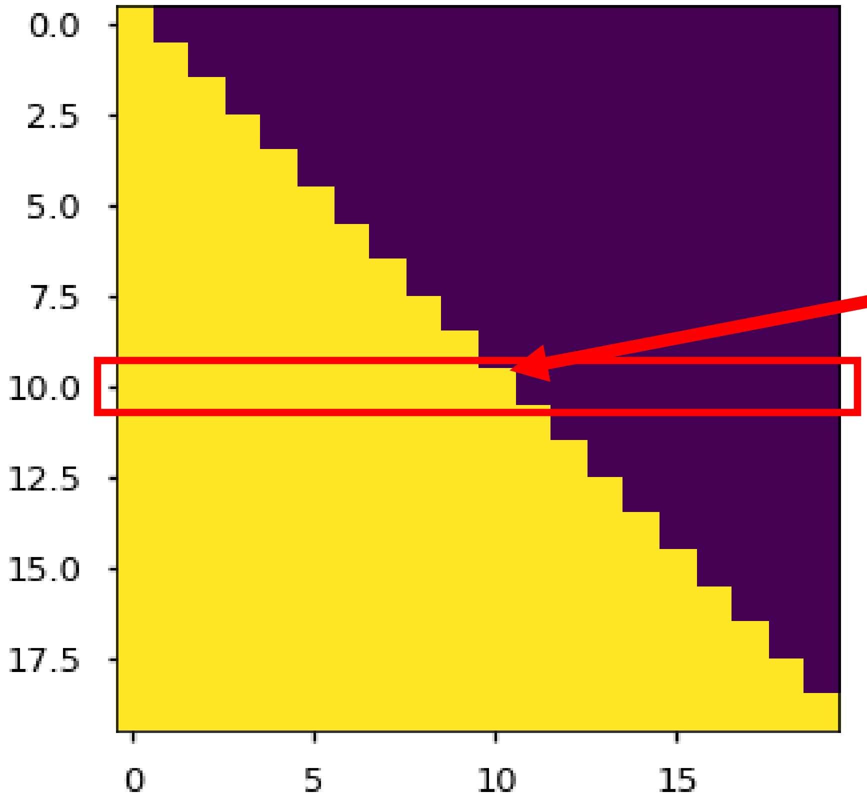
- Solution: Modify the attention weights to remove the contribution from the future tokens



Slide credit: S. Lazebnik

Masked self-attention layer

Mask for self attention



Sequence length in the decoder

(yellow: 1, dark blue: 0)

Simply set the dot product scores to be 0 if the mask value is 0. Why?

At each location, it can only attend to previous tokens, including itself.

Why?

Masked self-attention layer

- Mainly used in the decoder
  - The decoding process works in a sequential/autoregressive manner
- Do we need masked self-attention in the encoder?
  - In the training, we usually need to pad the input sequences
  - Shall we attend to the extra <PAD> tokens?
  - No. So the attention mask is about whether a token is <PAD> or not.

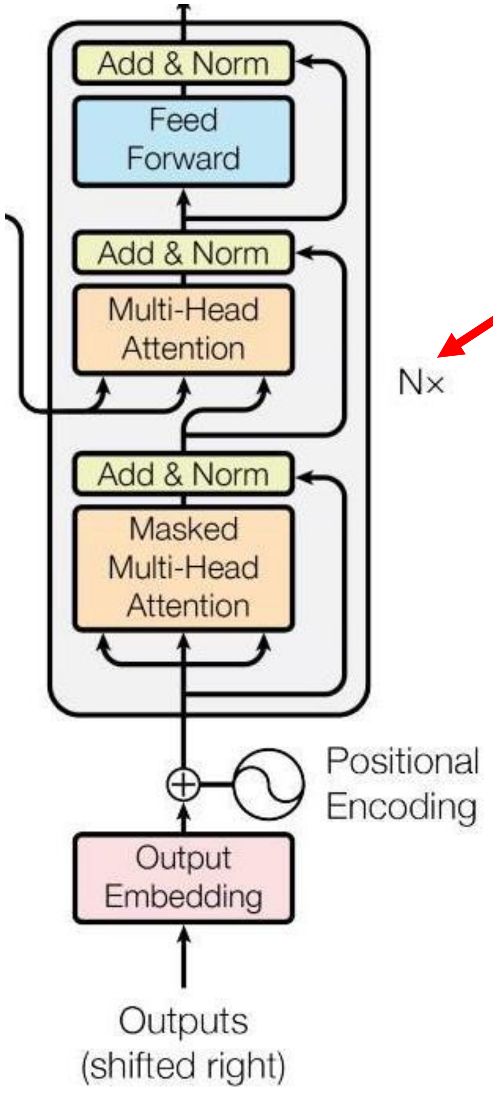
Decoder: Training vs Inference

- Training:
  - Batched data that needs to be processed in parallel
  - The goal is to predict the next token (that’s why the masked self-attention is useful)
- Inference:
  - Generating words one by one, so called autoregressive
  - Inherently slow

Transformer Decoder

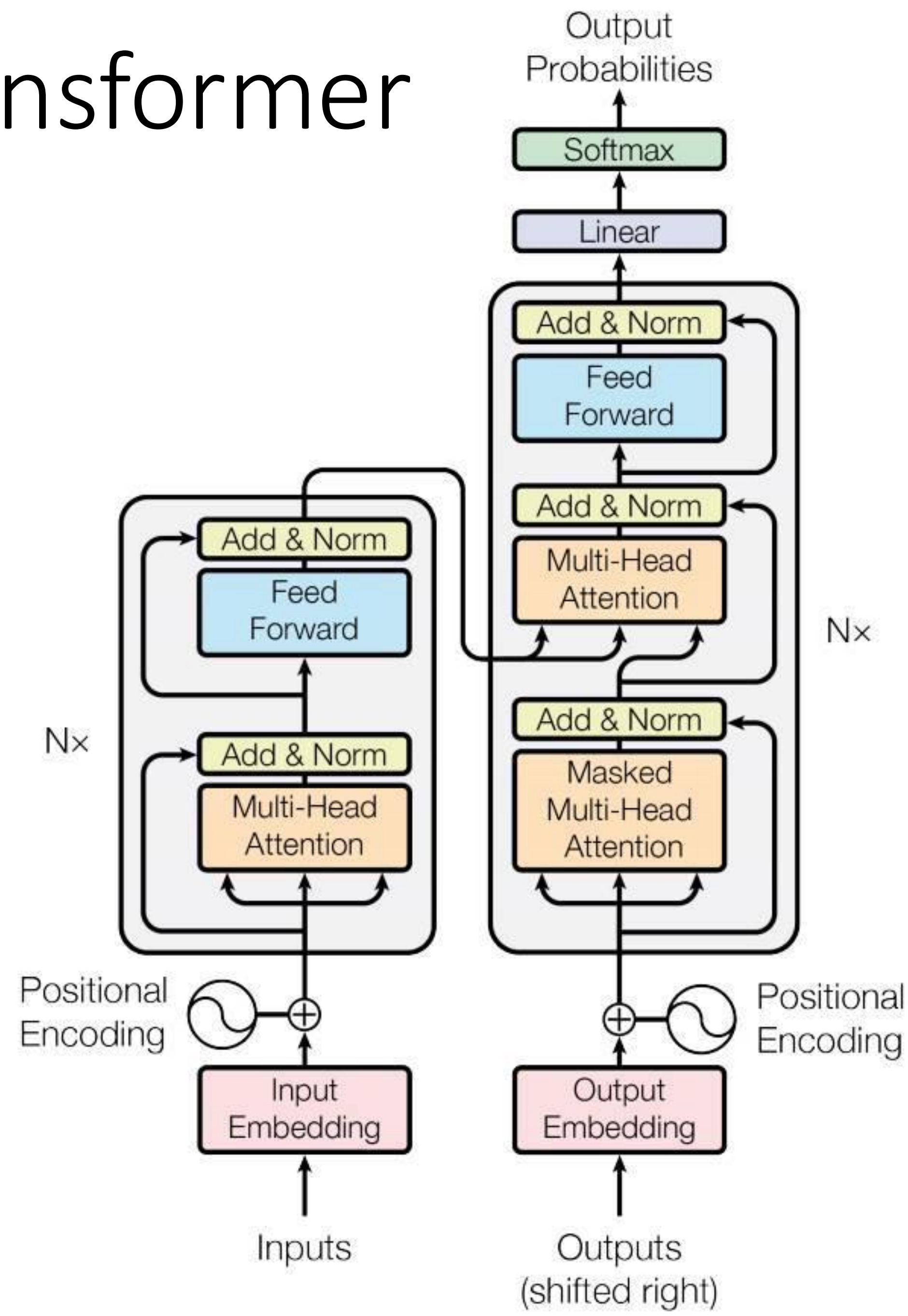
Output (encodings) from the encoder

What is the shape?



Multiple stacks of the decoder cell (hyper parameter)

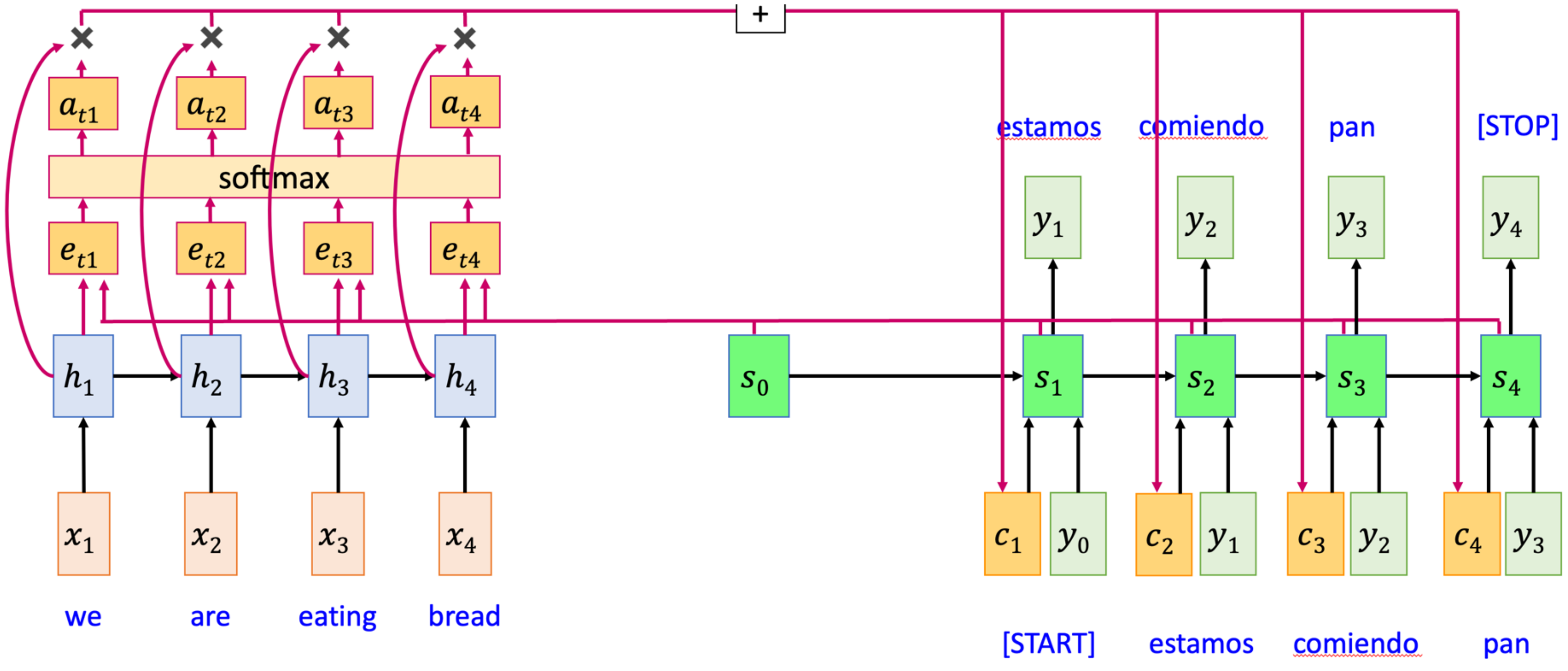
Transformer

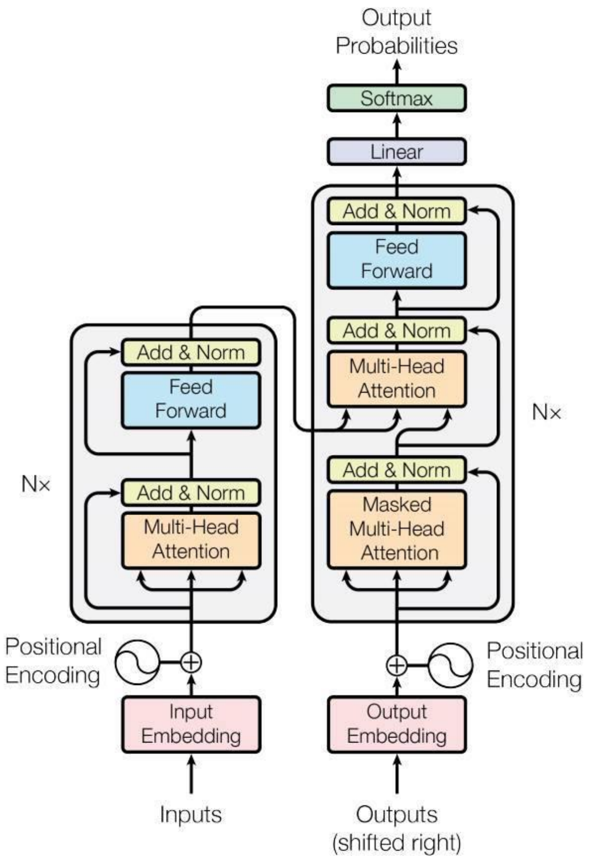


Key components:

- Multi-head (self-)attention
  - With a mask (optional)
- Positional encoding
- Feedforward network
- Residual connections
- Regularization tricks
  - Dropout
  - LayerNorm

RNN vs Transformer





Encoder:

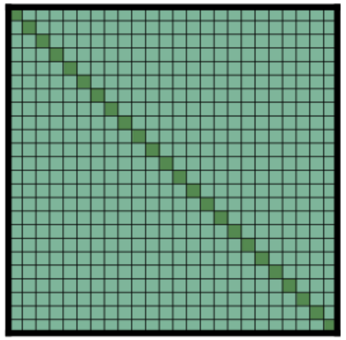
- RNN: sequential vs Transformer: parallel
- RNN: hard to be deep vs Transformer: could be deep

Decoder:

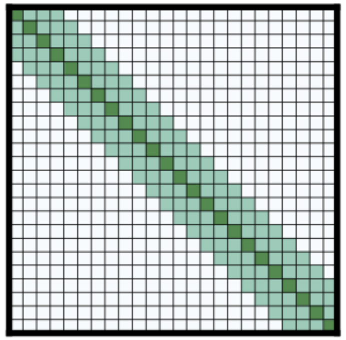
- RNN & Transformer: sequential
- RNN: hard to be deep vs Transformer: could be deep

Sparse Attention in Transformer

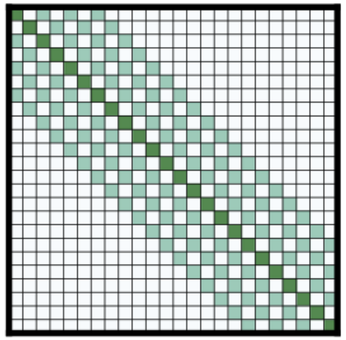
What is the main bottleneck of the Transformer?



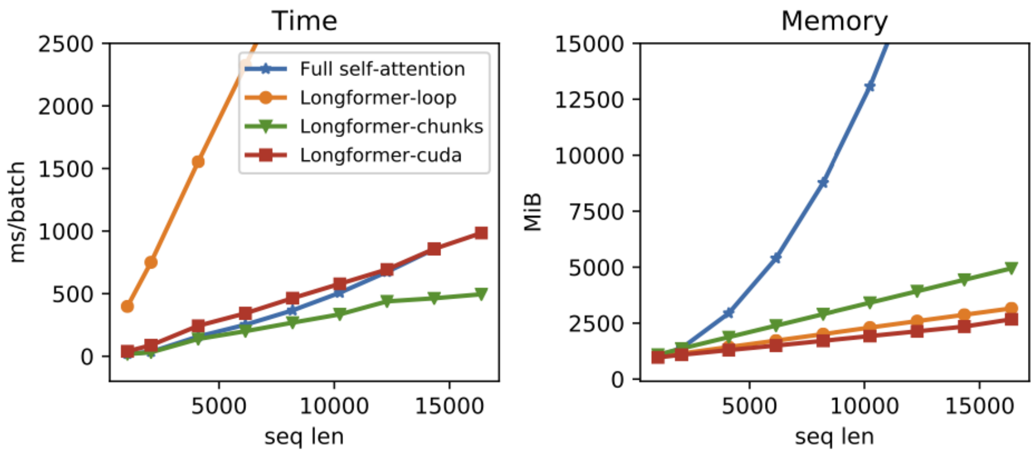
(a) Full n^2 attention



(b) Sliding window attention



(c) Dilated sliding window



[Beltagy et al., Longformer: The Long-Document Transformer. arXiv 2020]

Next Class

	Recurrent Neural Networks and Transformer		
Week 6	02/12/2025	Recurrent Neural Networks	
	02/14/2025	Seq2seq with RNN, Attention	
Week 7	02/19/2025	Transformer 1	
	02/21/2025	Transformer 2	pa2 due, pa3 out
Week 8	02/26/2025	no class: self review, work on pa3	
	02/28/2025	In-class midterm	
	03/05/2025	Spring break, no classes	
	03/07/2025		
	Applications of Deep Neural Networks		
Week 9	03/12/2025	Visualization of neural networks	
	03/14/2025	Object Detection	
Week 10	03/19/2025	Image Segmentation	pa3 due, pa4 out
	Generative Models (GenAI)		
	03/21/2025	NeRF & 3D Gaussian Splatting	
Week 11	03/26/2025	GANs	
	03/28/2025	VAE	
Week 12	04/02/2025	Diffusion Models	
	04/04/2025	Building ChatGPT	pa4 due, pa5 out (optional)
	04/09/2025	From ChatGPT to GPT4: self-supervised learning in vision	