CSDS 451: Designing High Performant Systems for AI

Lecture 15

10/16/2024

Sanmukh Kuppannagari

sanmukh.kuppannagari@case.edu

https://sanmukh.research.st/

Case Western Reserve University

Outline

- Transformer Models
- Transformer Models – Computational Challenges

Announcements

- Midterm Graded

Outline

- Transformer Models
- Transformer Models – Computational Challenges

Transformer Based Models

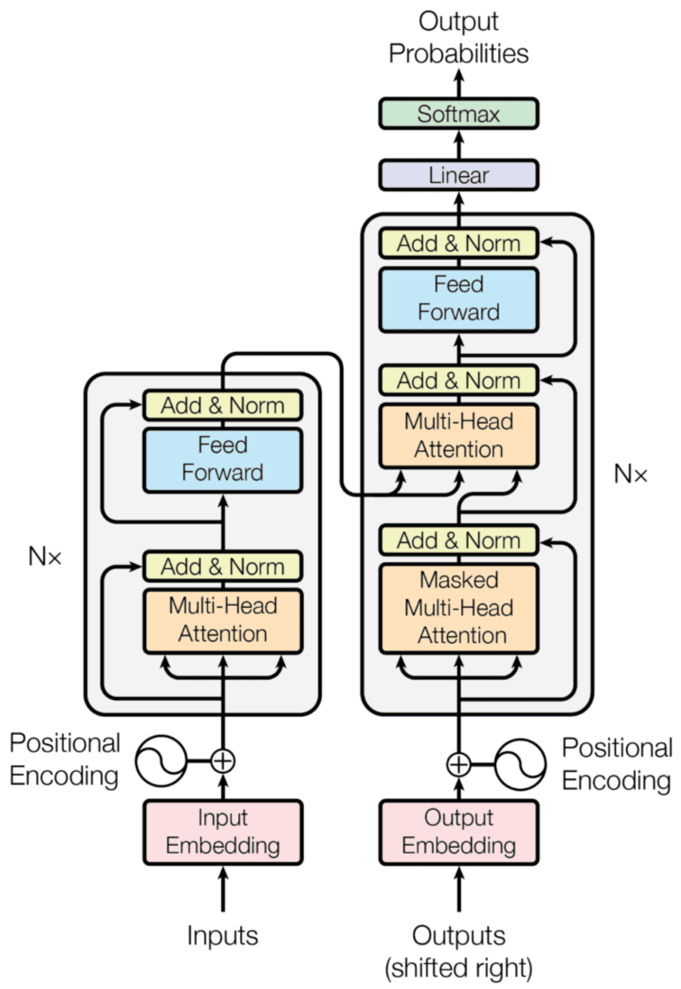
- Neural network for language modeling that use Transformer Architecture as a key building block

- State of the art architectures for most of the Natural Language Processing Tasks

- ChatGPT, Llama, …

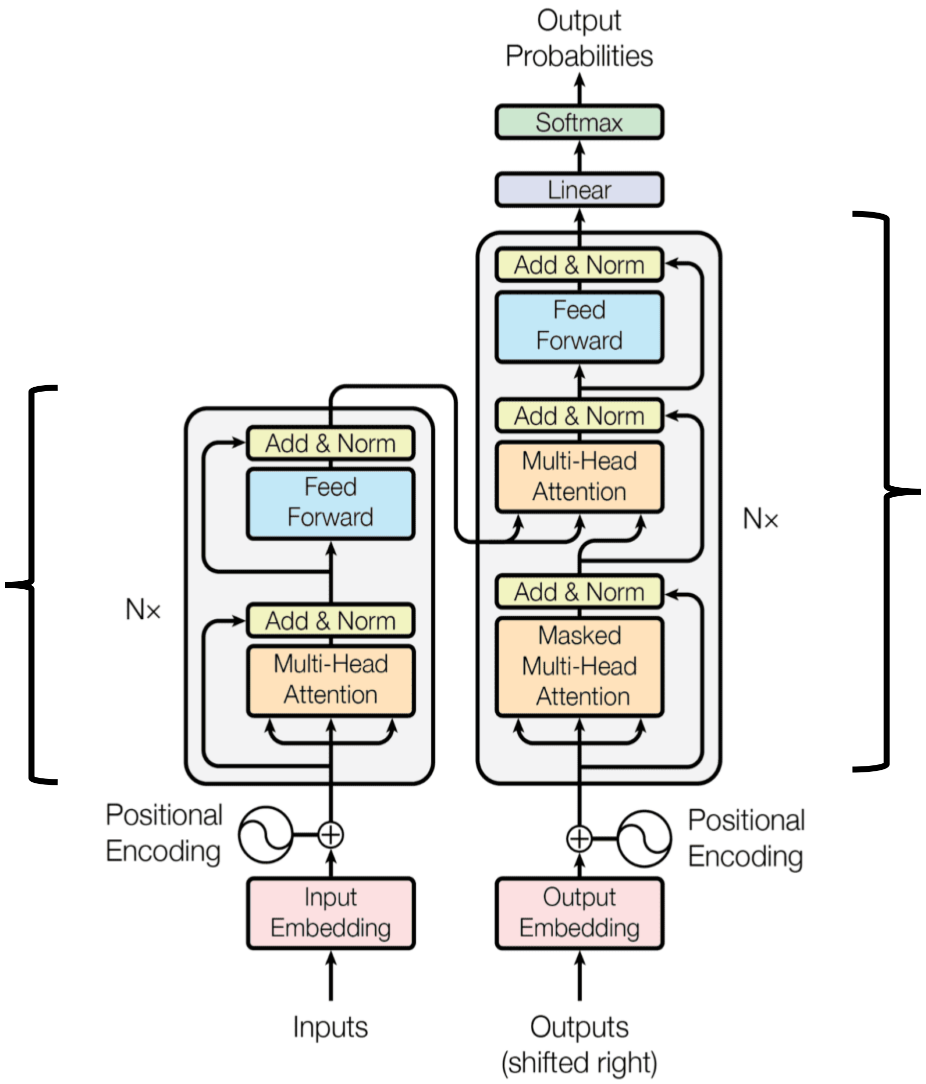
Transformer Models

- Key Idea: Attention Mechanism
- Original paper: Attention Is All You Need: https://arxiv.org/abs/1706.03762



Transformer Models

N Layers of Encoder Blocks



N Layers of Decoder Blocks

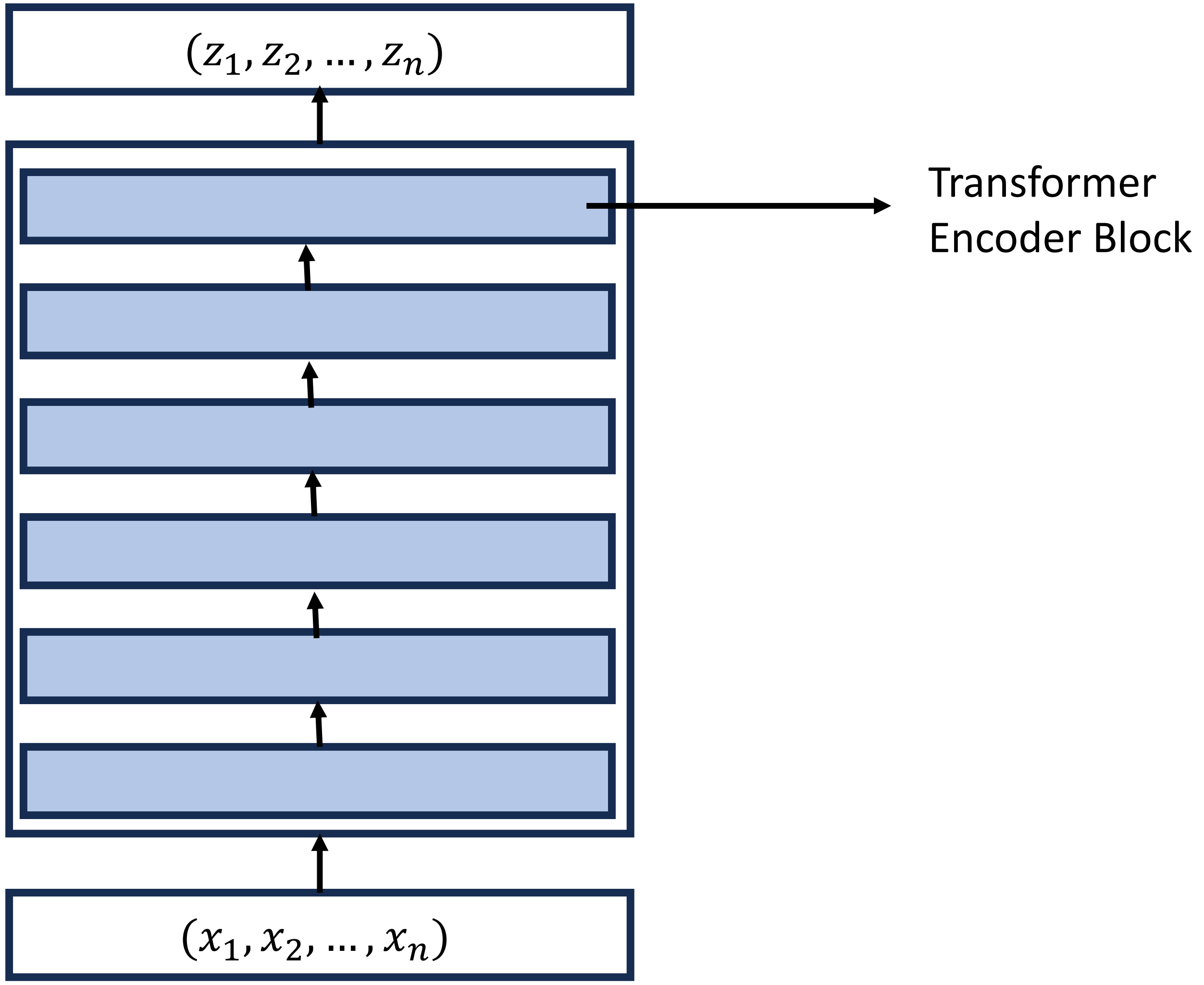
Transformer Models

- Encoder-Decoder Architecture

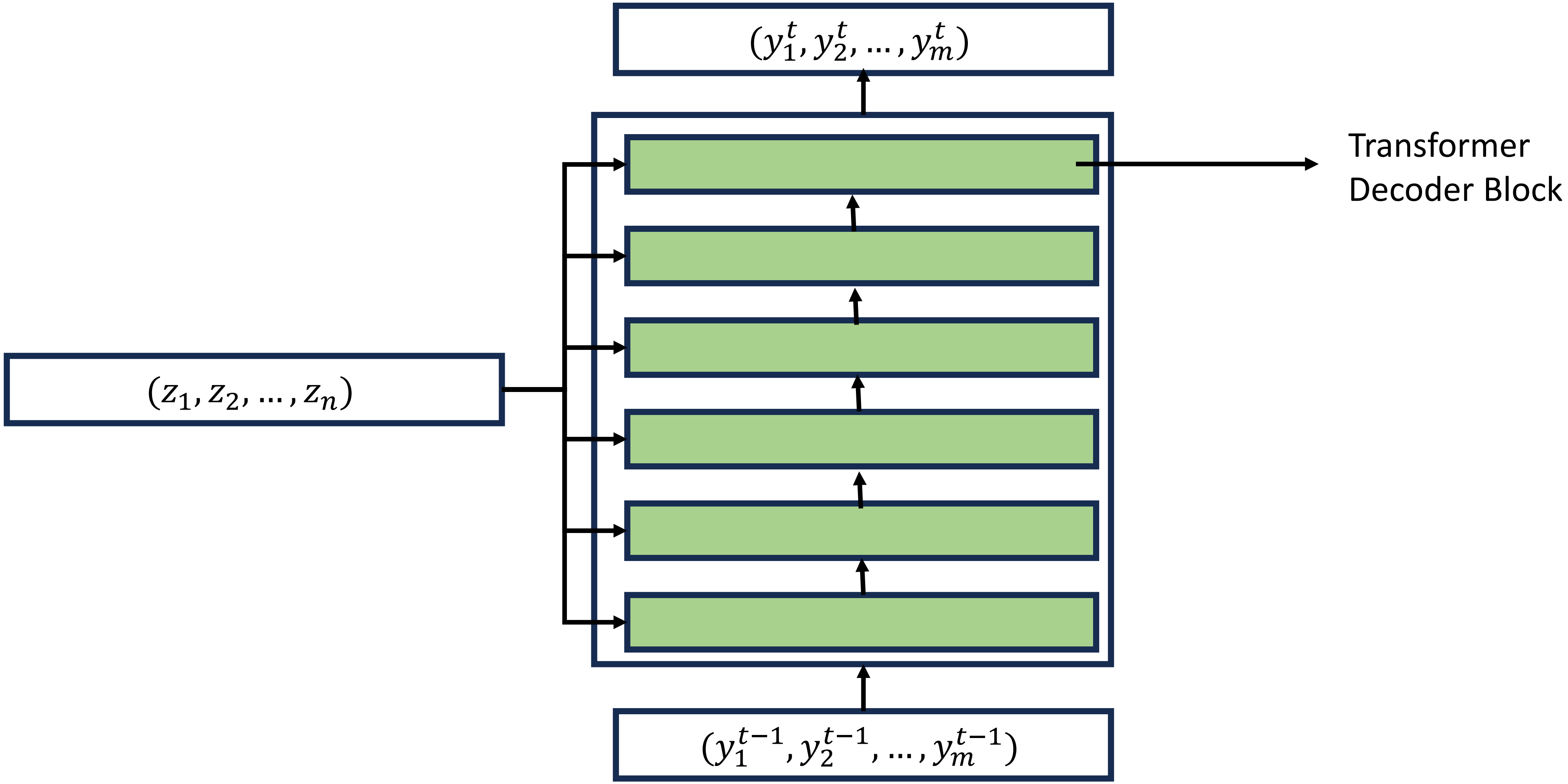
- Encoder:
  - Map a sequence of input symbols $(x_1, x_2, \dots, x_n)$, to
  - A sequence of continuous representation $(z_1, z_2, \dots, z_n)$

- Decoder:
  - Given $(z_1, z_2, \dots, z_n)$ and output from previous iteration $(y_1^{t-1}, y_2^{t-1}, \dots, y_m^{t-1})$
  - Output $(y_1^t, y_2^t, \dots, y_m^t)$

Transformer Models - Encoder



Transformer Models - Decoder



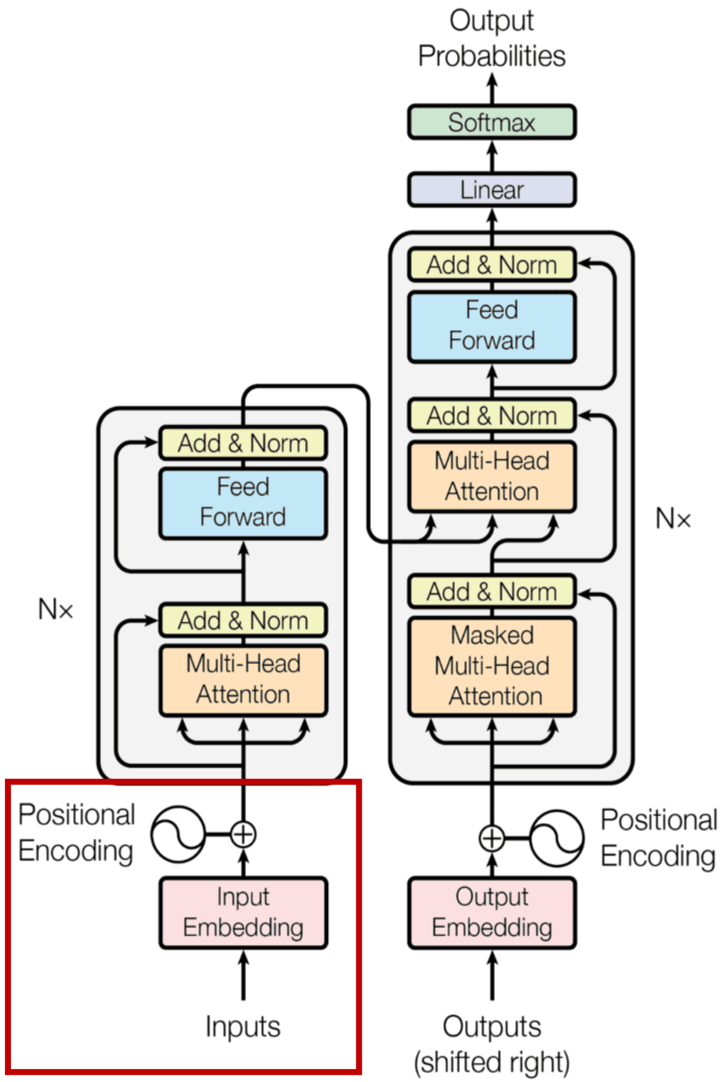
Transformer Models - Inference

- The actual process depends upon the task
- Here we will consider a simple sentence prediction
- Input: AI stands for
- Output: Artificial Intelligence

Transformer Models - Inference

N Layers of Encoder Blocks

Generate the input embeddings



N Layers of Decoder Blocks

Transformer Models - Inference

- Convert the input into embeddings
  - Basically, each word (token) is transformed into a high dimensional vector
  - Example embedding scheme: Word2Vec

- Input: [AI stands for]

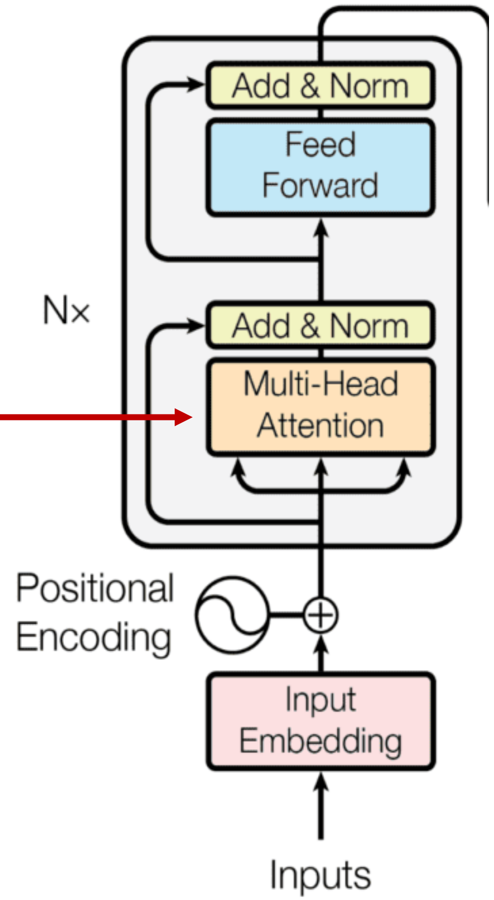
- Embeddings: $[x_1, x_2, x_3]$

- Add positional embeddings – Used to incorporate information about the location of each word. We will not discuss this.

Transformer Models - Inference

Input Encoding Phase

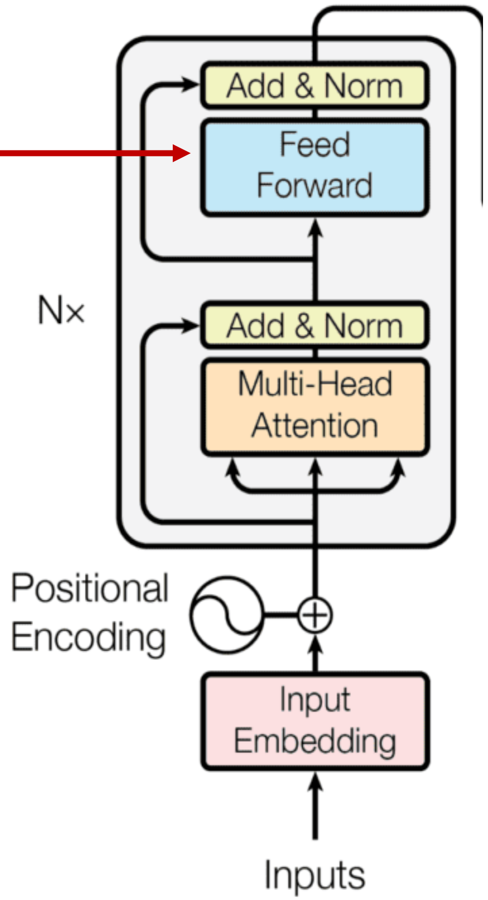
Perform Multi-headed Attention



Transformer Models - Inference

Input Encoding Phase

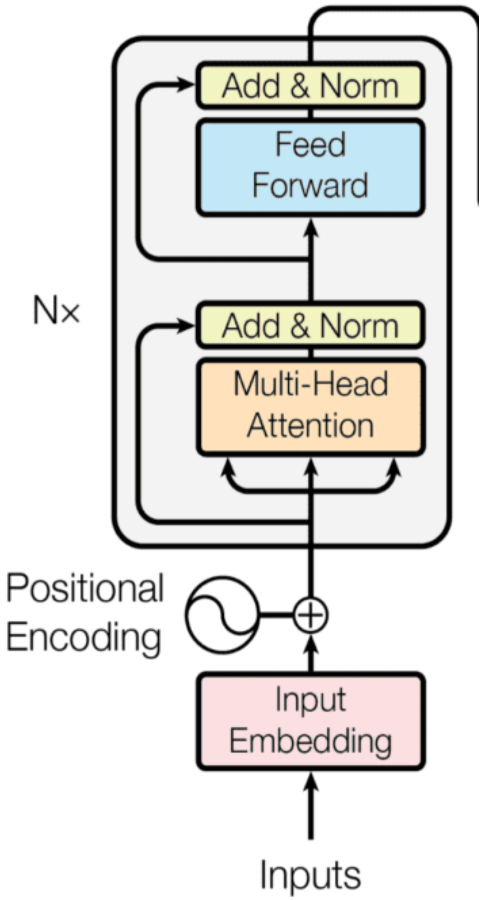
Feed-forward
Network



Transformer Models - Inference

Input Encoding Phase

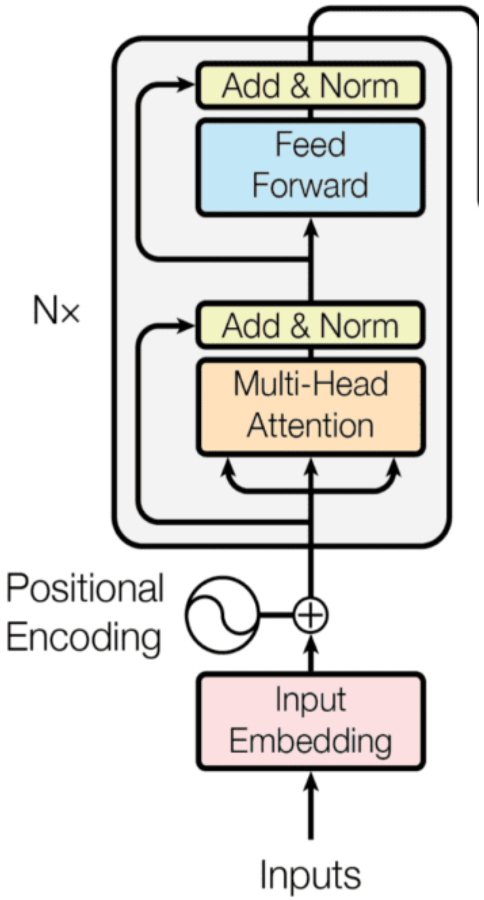
Perform N times using N different layers



Transformer Models - Inference

Input Encoding Phase

Produce Input Encoding:
$[z_1, z_2, z_3]$
Note, same length



Transformer Models - Inference

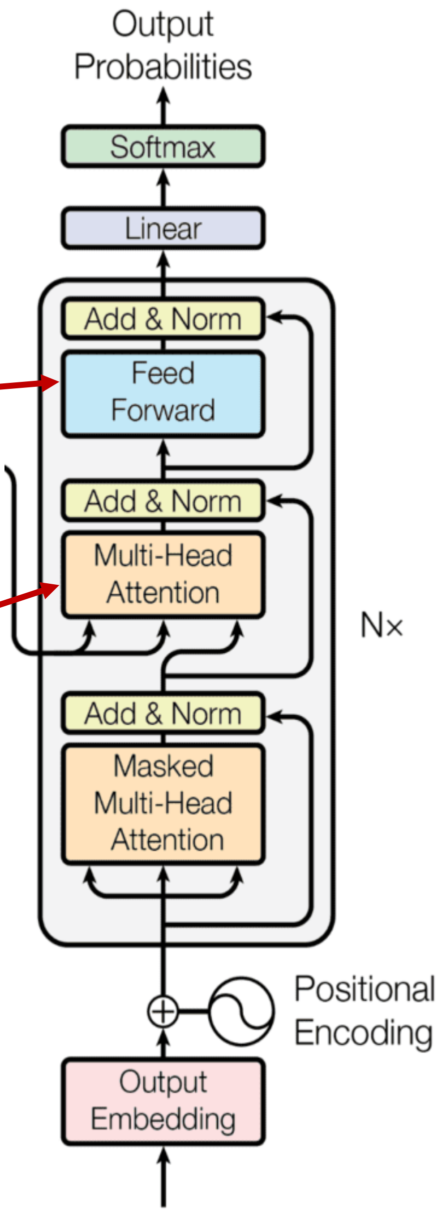
- Once the input encodings are produced, the decoding phase starts

- Proceeds in an iterative manner, in each iteration:
  - Using the output produced till now, produce the next token

Transformer Models - Inference

Feed forward Network

$[z_1, z_2, z_3]$



Perform Multi-headed Attentions

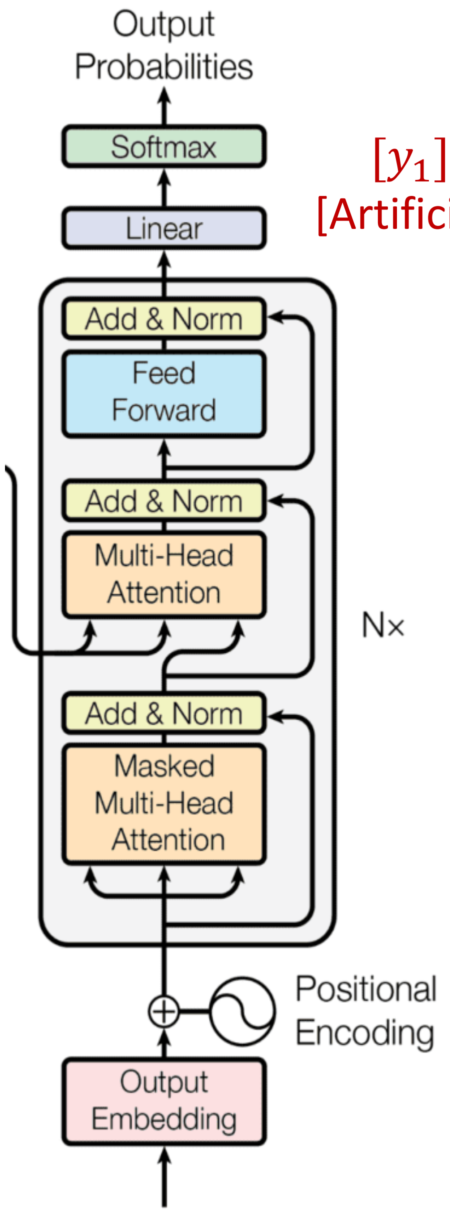
Start with a special start token

<s>

Transformer Models - Inference

$[y_1]$
[Artificial]

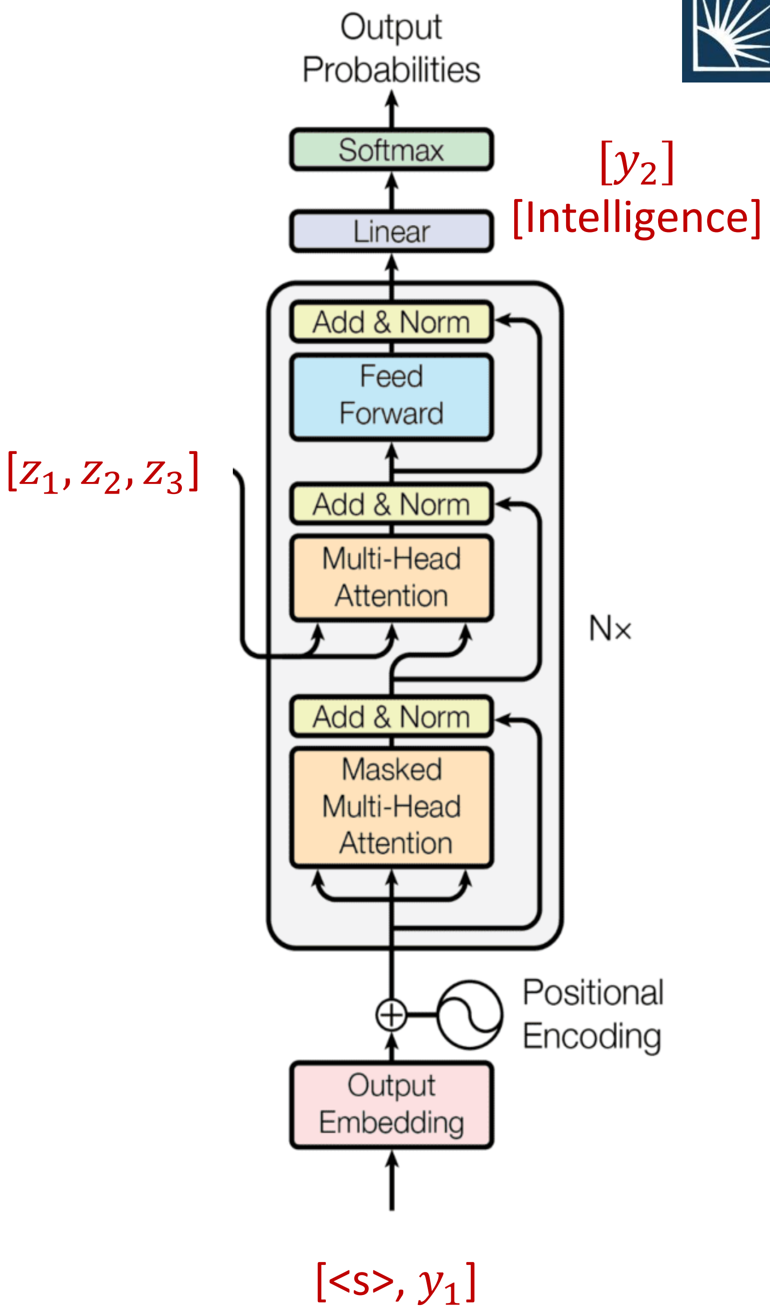
$[z_1, z_2, z_3]$



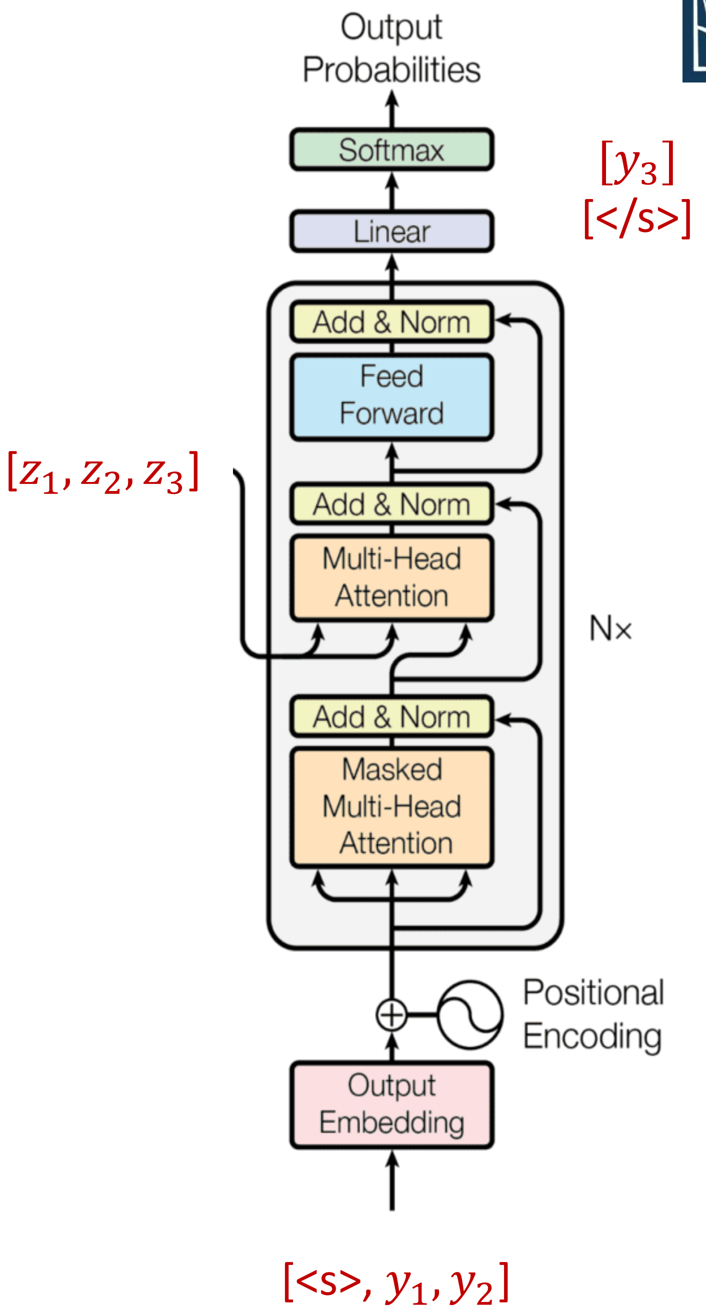
<s>

CASE SCHOOL OF ENGINEERING
CASE WESTERN RESERVE UNIVERSITY

Transformer Models - Inference



Transformer Models - Inference

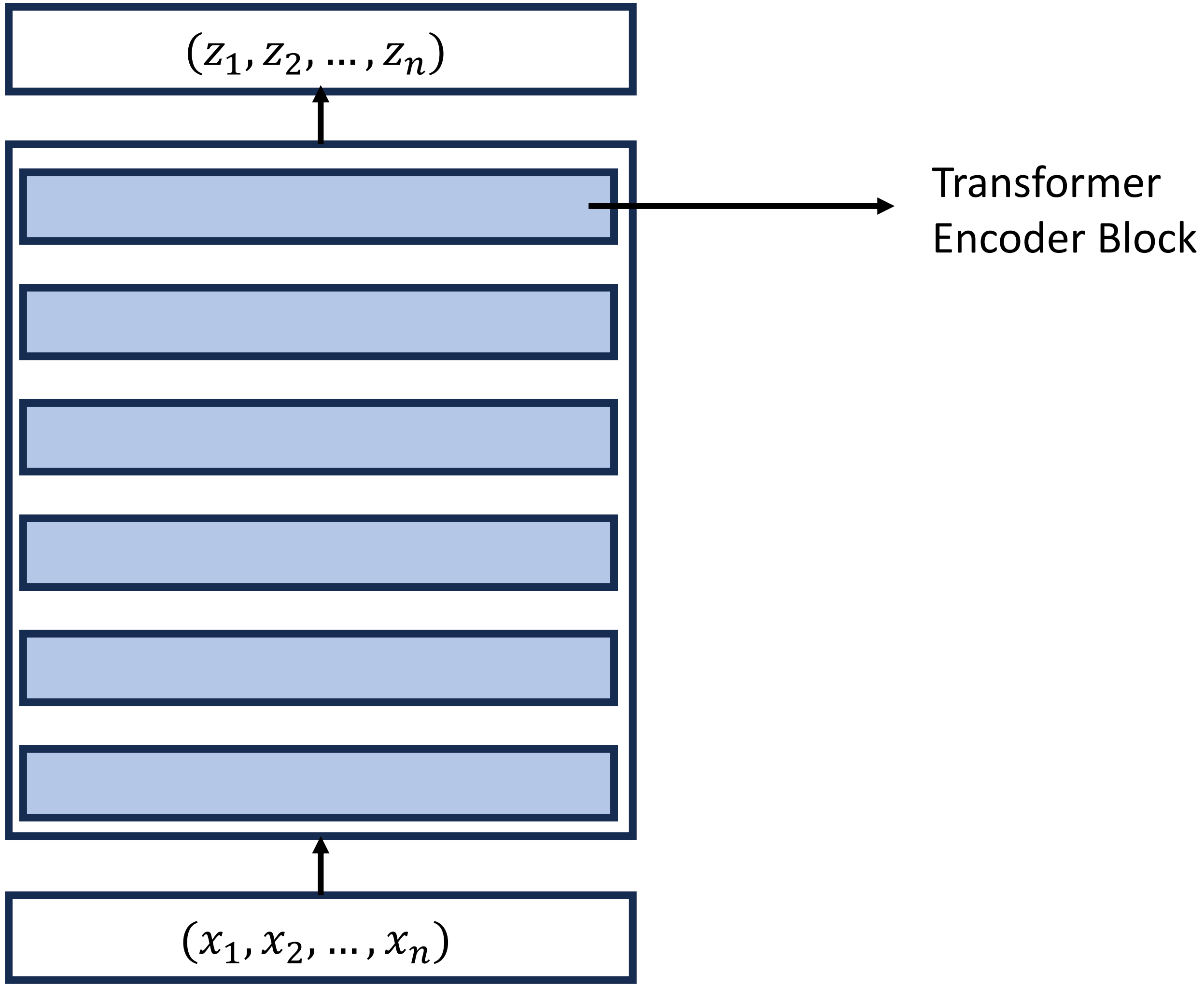


Stop when a special stop signal is generated

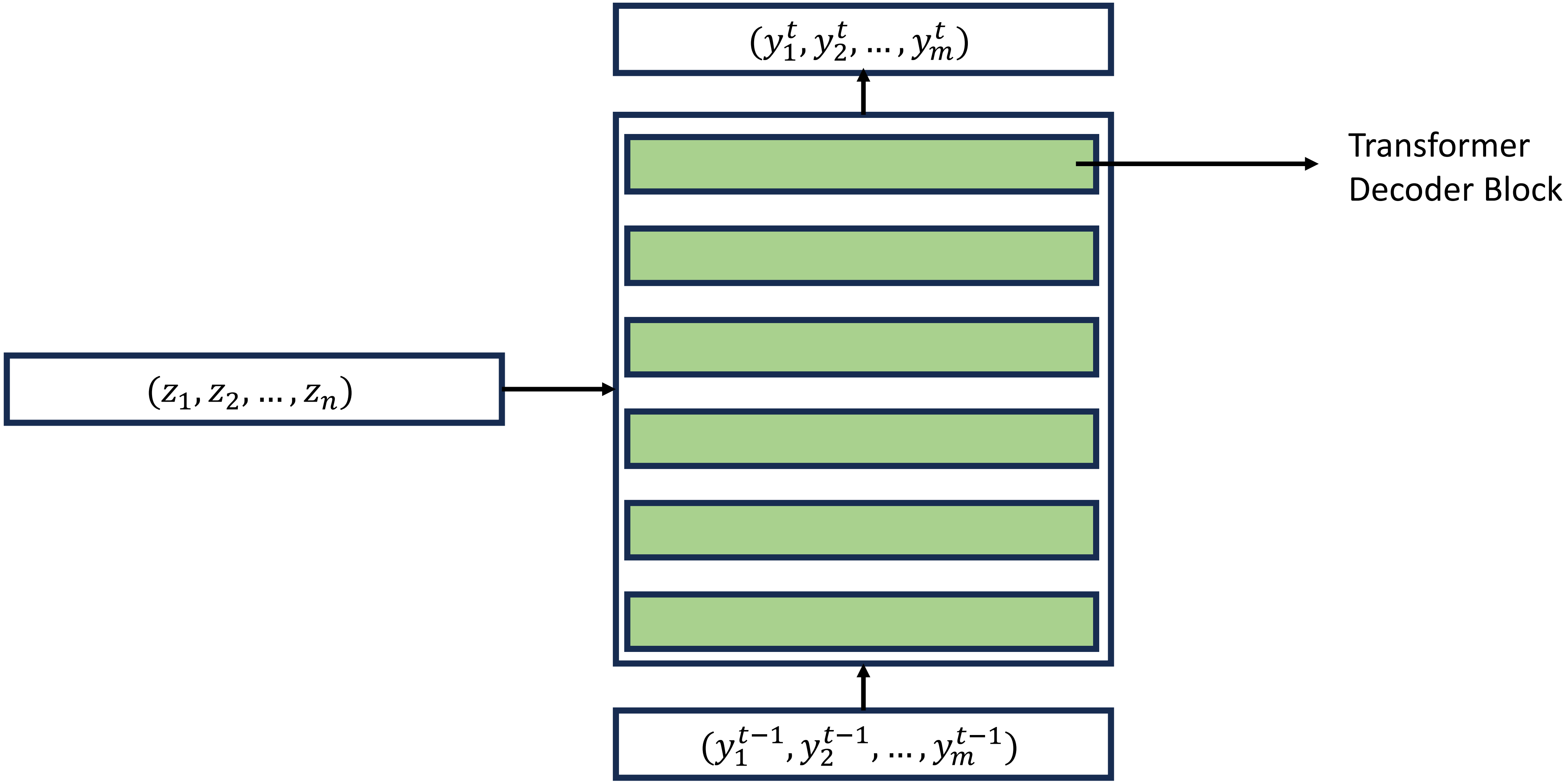
Transformer Model Types

- Encoder Only Models: Use only the Encoder portion for tasks
  - BERT
  - Good for tasks that require understanding the entire text
  - Example: Classification, similarity,
- Decoder Only Models: Use only the Decoder portion for tasks
  - GPT
  - Good for auto-regressive tasks
  - Example: Text completion, Chatbots, code/text generation
- Encoder-Decoder Models: Have both Encoder-Decoder portions
  - Original Transformer that we discussed, T5
  - Good for tasks that require input to output mappings
  - Example: Translation, summarization

Transformer Models - Encoder

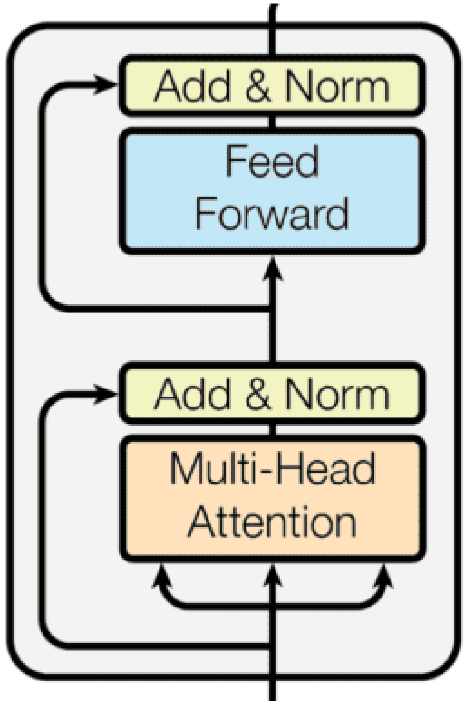


Transformer Models - Decoder



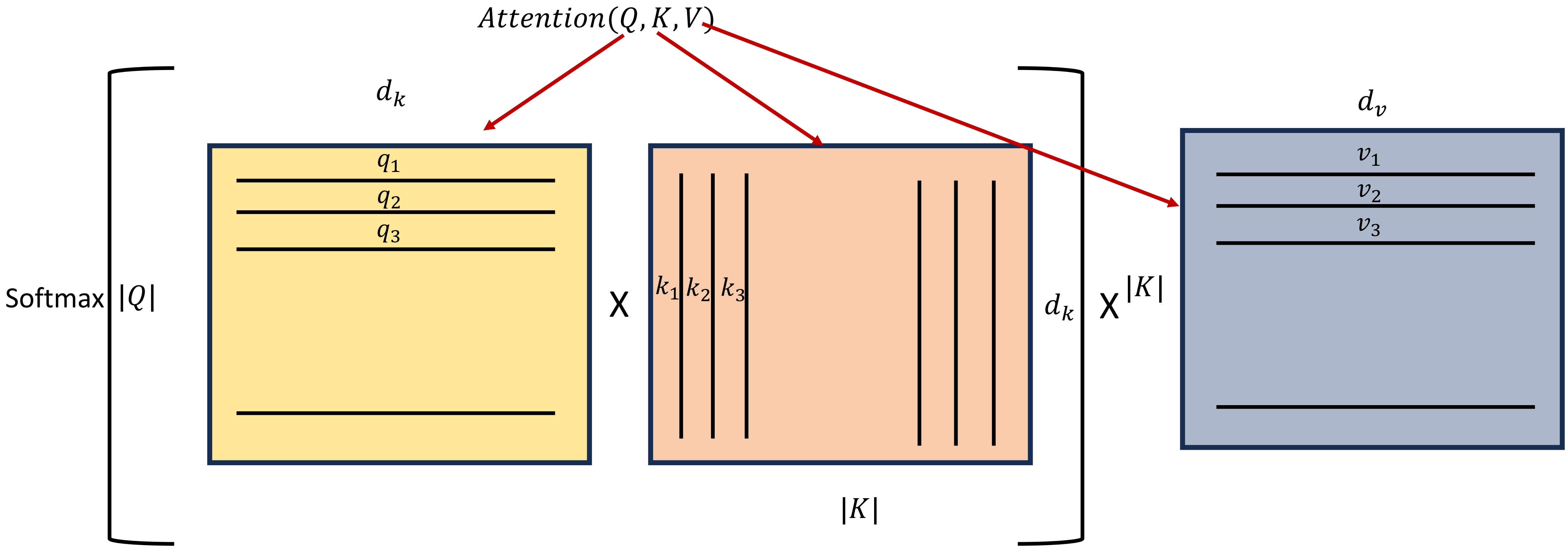
Transformer Encoder Block

- Contains two sub-layers
  - Attention Mechanism
  - Feed Forward Network



CASE SCHOOL OF ENGINEERING
CASE WESTERN RESERVE UNIVERSITY

Attention Mechanism



Self-Attention: $|Q| = |K|$
Number of queries = number of keys = number of values

Q Matrix: Each row is a query
K Matrix: Each column is a key
V Matrix: Each row is a value

X: Matrix MM

Attention Mechanism

$\boldsymbol{q_1}$

$\boldsymbol{k_1}$ $\boldsymbol{v_1}$

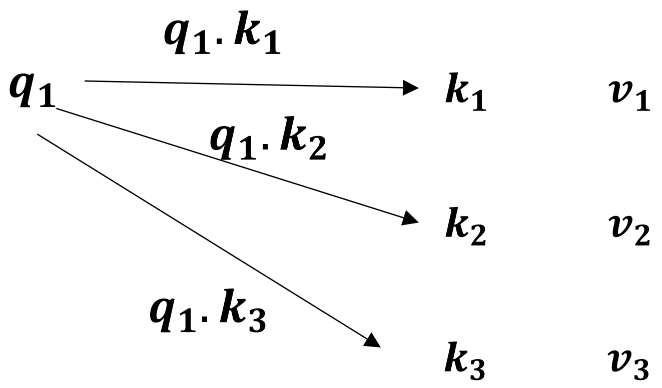
$\boldsymbol{k_2}$ $\boldsymbol{v_2}$

$\boldsymbol{k_3}$ $\boldsymbol{v_3}$

Attention Mechanism: Think of it as a lookup table

Given a **query**, output a **value** that is corresponding to the key that is ***most similar*** to query

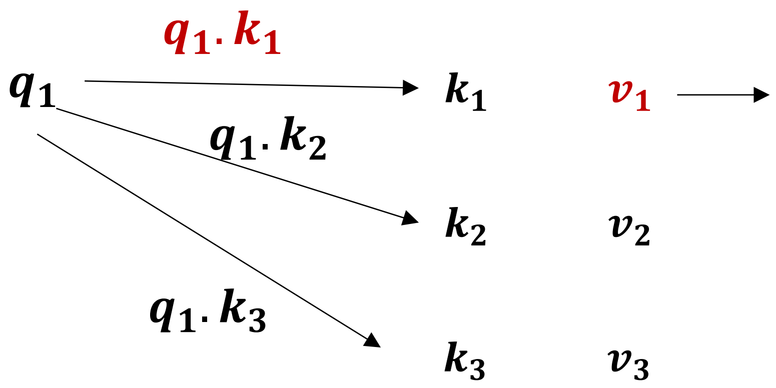
Attention Mechanism



Attention Mechanism: Think of it as a general version of a lookup table

Lookup Table: Given a **query**, output a **value** that is corresponding to the key that is ***most similar*** to query

Attention Mechanism

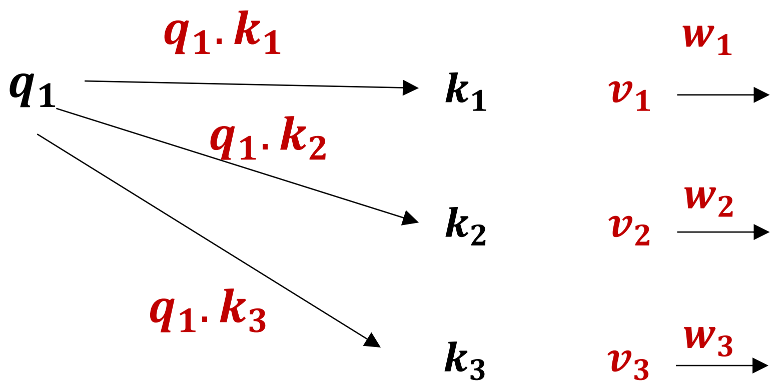


Attention Mechanism: Think of it as a general version of a lookup table

Lookup Table: Given a **query**, output a **value** that is corresponding to the key that is ***most similar*** to query

Attention Mechanism

$$w_i = \frac{e^{q_1 . k_i}}{\sum_j e^{q_1 . k_j}}$$

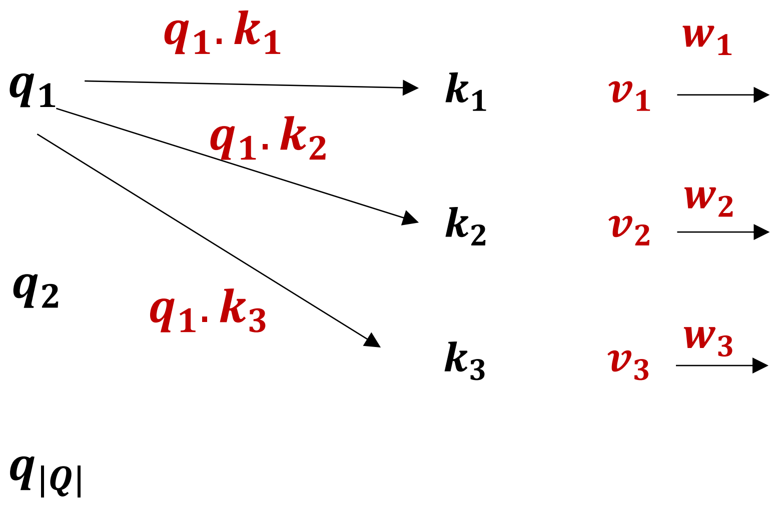


$$\sum w_i . v_i$$

Lookup Table: Given a **query**, output a **value** that is corresponding to the key that is ***most similar*** to query

Attention Mechanism: weighted combination of values with similarity score as weights.

Attention Mechanism

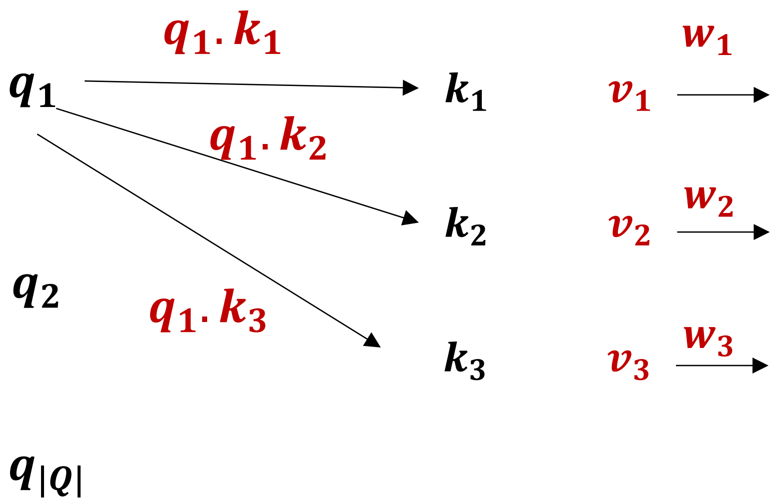


$$w_i = \frac{e^{q_1.k_i}}{\sum_j e^{q_1.k_j}}$$

Attention Mechanism: weighted combination of values with similarity score as weights.

Do that for all queries

Attention Mechanism



$$w_i = \frac{e^{q_1.k_i}}{\sum_j e^{q_1.k_j}}$$

What are queries, keys, and values?

All three are simply the projection of the input sequence $(x_1, x_2, \ldots, x_n)$

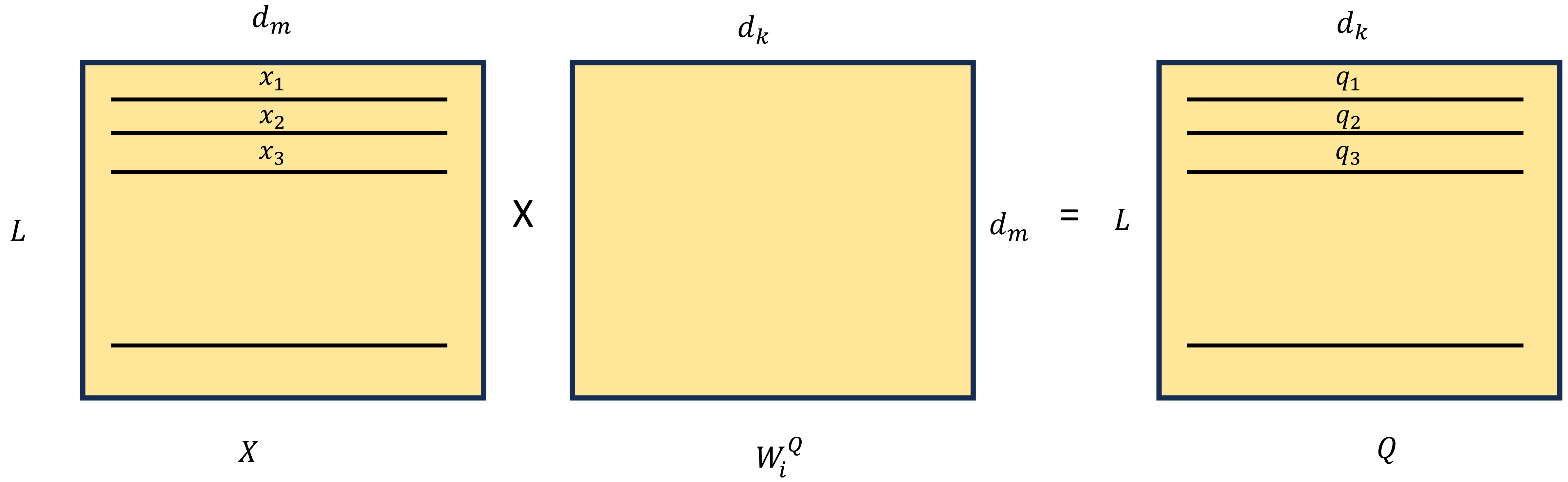
Multi-Headed Attention

- Instead of single attention, perform h attention computations
- Assuming dimension of embeddings is d_m (also called **model dimension**)
- Project Q, K, V using $W_i^Q \in R^{d_m \times d_k}, W_i^K \in R^{d_m \times d_k}, W_i^V \in R^{d_m \times d_v}\, \forall k \in \{1,2,\dots,h\}$
- $head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$
- $Concat(head_1, head_2, \dots, head_h)W^O$ where $W^O \in R^{hd_v \times d_m}$

Multi-Headed Attention

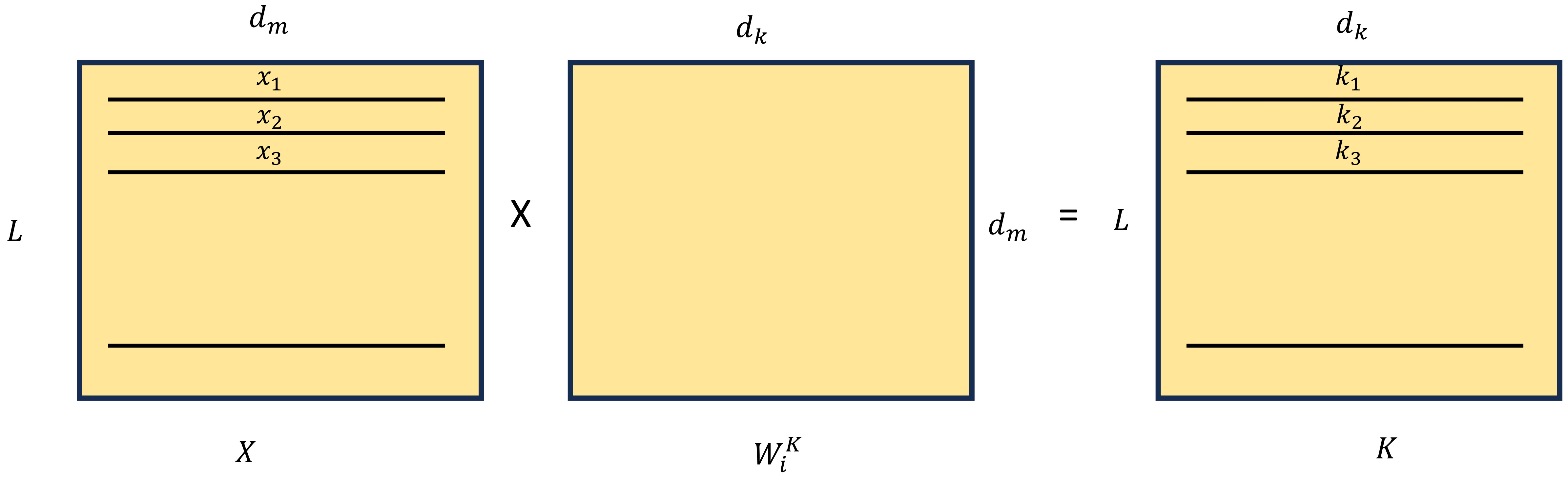
- Instead of single attention, perform h attention computations
- Assuming dimension of embeddings is d_m (also called **model dimension**)
- Project Q, K, V using $W_i^Q \in R^{d_m \times d_k}, W_i^K \in R^{d_m \times d_k}, W_i^V \in R^{d_m \times d_v} \forall k \in \{1,2, \dots, h\}$
- $head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$
- $Concat(head_1, head_2, \dots, head_h)W^O$ where $W^O \in R^{hd_v \times d_m}$

Multi-Headed Attention



$$i = \{1,2, \dots, h\}$$

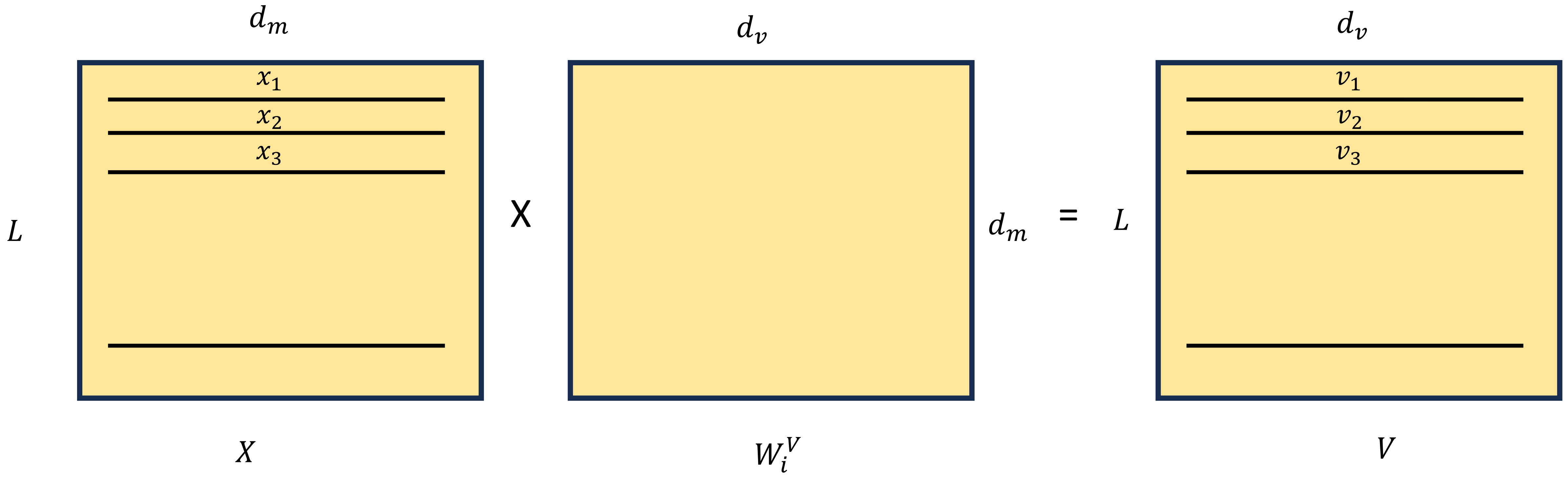
Multi-Headed Attention



$i = \{1,2,\dots,h\}$

Transpose K to use in attention mechanism

Multi-Headed Attention

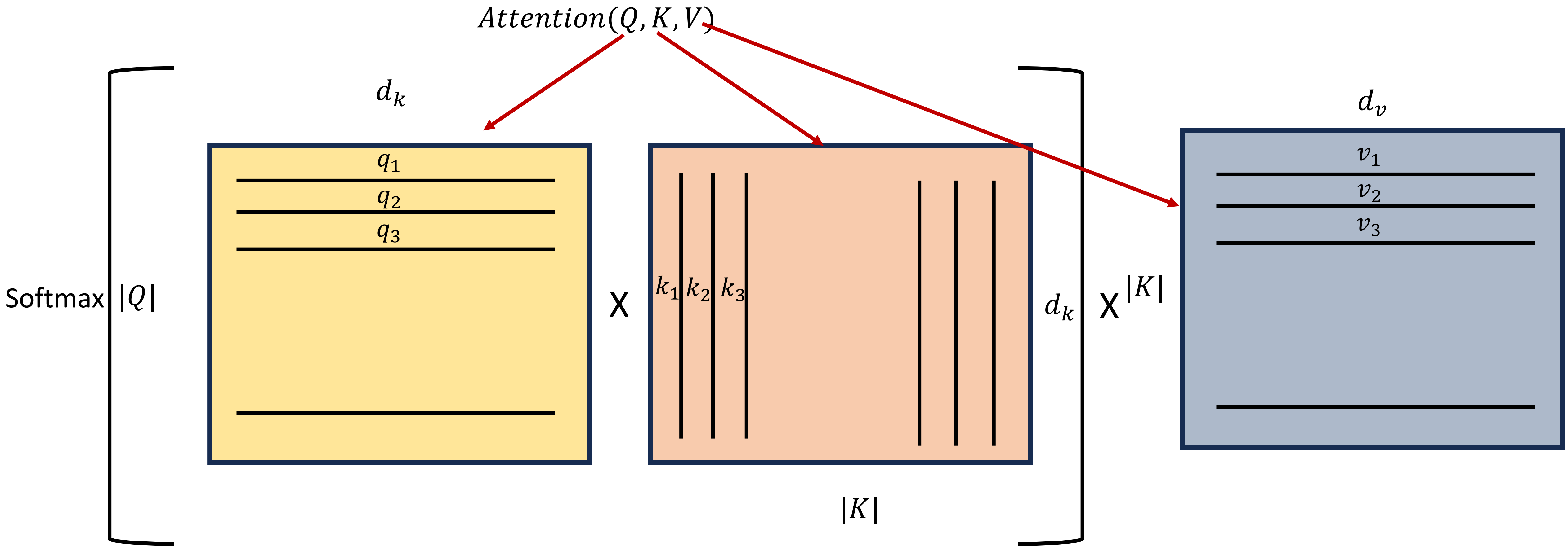


$$i = \{1,2,\dots,h\}$$

Multi-Headed Attention

- Instead of single attention, perform h attention computations
- Assuming dimension of embeddings is d_m (also called **model dimension**)
- Project Q, K, V using $W_i^Q \in R^{d_m \times d_k}, W_i^K \in R^{d_m \times d_k}, W_i^V \in R^{d_v \times d_v} \forall k \in \{1,2,\ldots,h\}$
- $head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$
- $Concat(head_1, head_2, \ldots, head_h)W^O$ where $W^O \in R^{hd_v \times d_m}$

Multi-Headed Attention

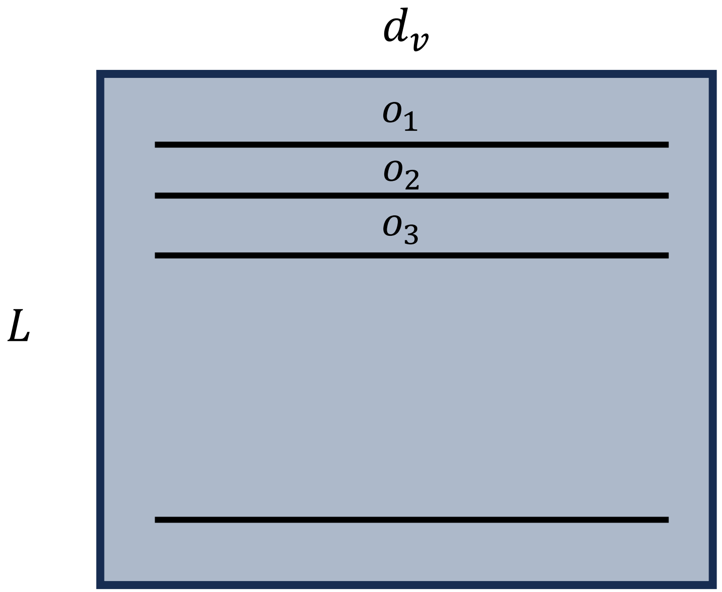


Perform attention for each head i

X: Matrix MM

Multi-Headed Attention

$$O = \ Attention(Q, K, V)$$



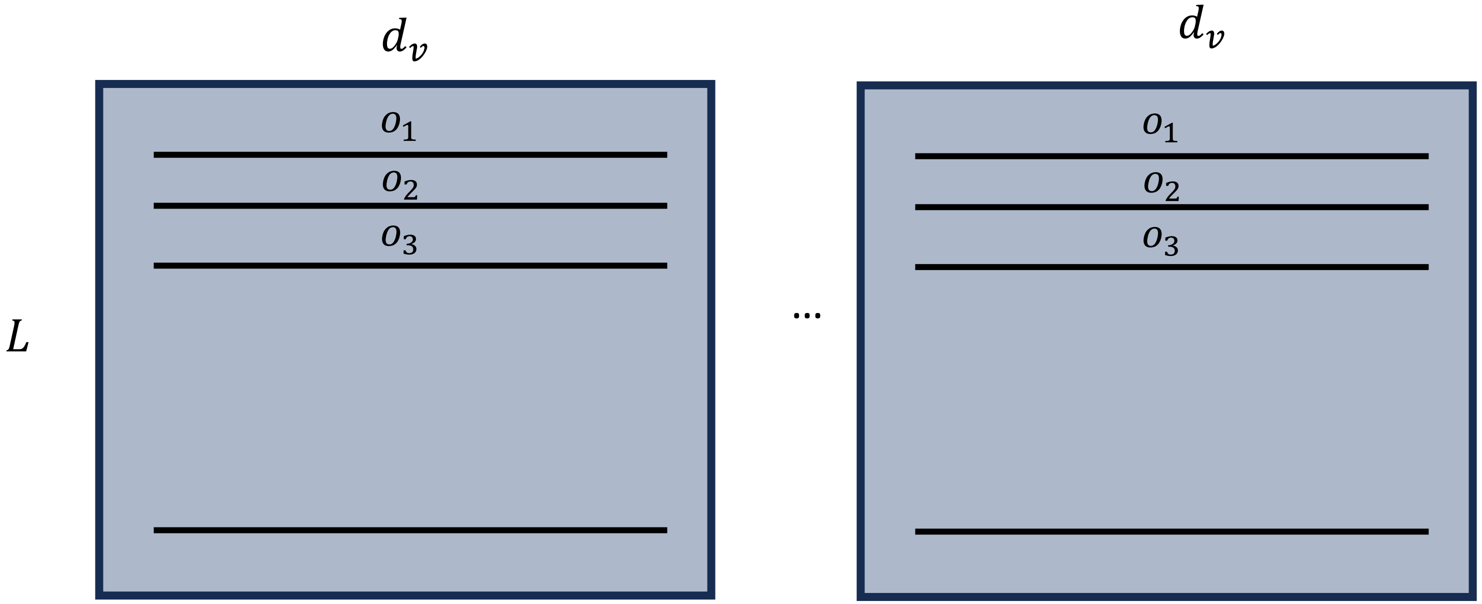
Perform attention for each head i

X: Matrix MM

CASE SCHOOL OF ENGINEERING
CASE WESTERN RESERVE UNIVERSITY

Multi-Headed Attention

$$O = Attention(Q, K, V)$$



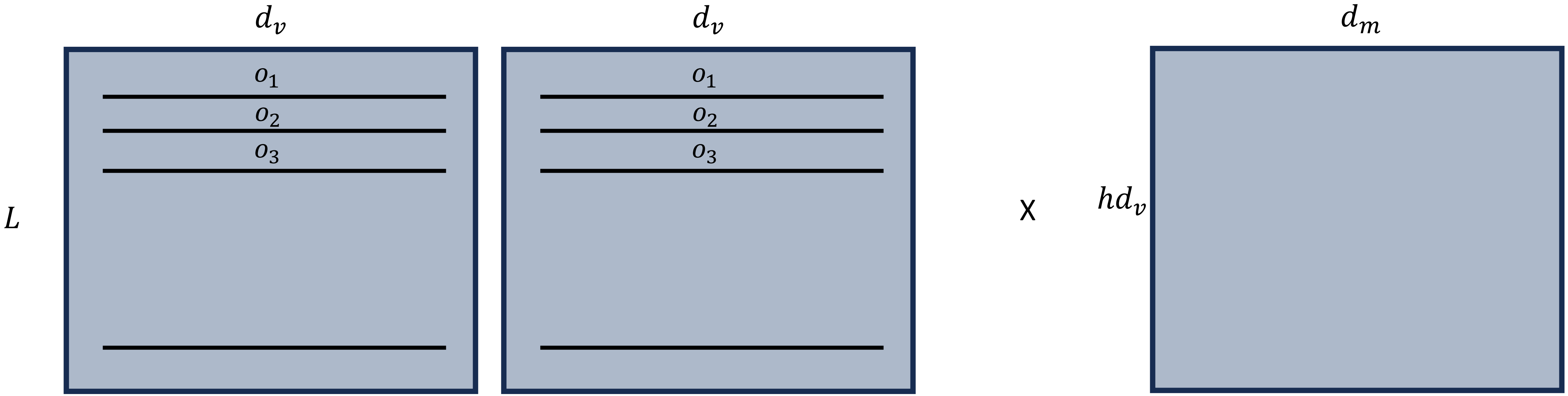
Concatenate all heads to produce hd_k dimensional output

Multi-Headed Attention

- Instead of single attention, perform h attention computations
- Assuming dimension of embeddings is d_m (also called **model dimension**)
- Project Q, K, V using $W_i^Q \in R^{d_m \times d_k}, W_i^K \in R^{d_m \times d_k}, W_i^V \in R^{d_m \times d_v} \forall k \in \{1,2, \dots, h\}$
- $head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$
- $Concat(head_1, head_2, \dots, head_h)W^O$ where $W^O \in R^{hd_v \times d_m}$

Multi-Headed Attention

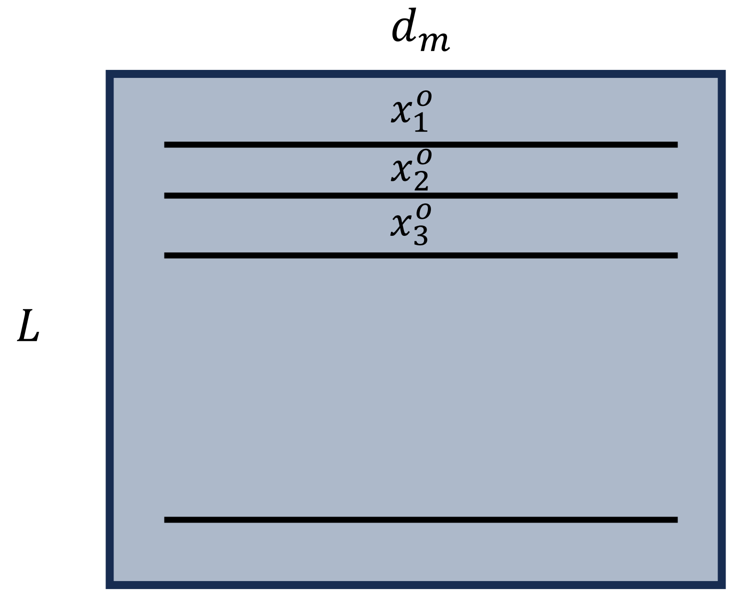
$$O = \; Attention(Q, K, V)$$



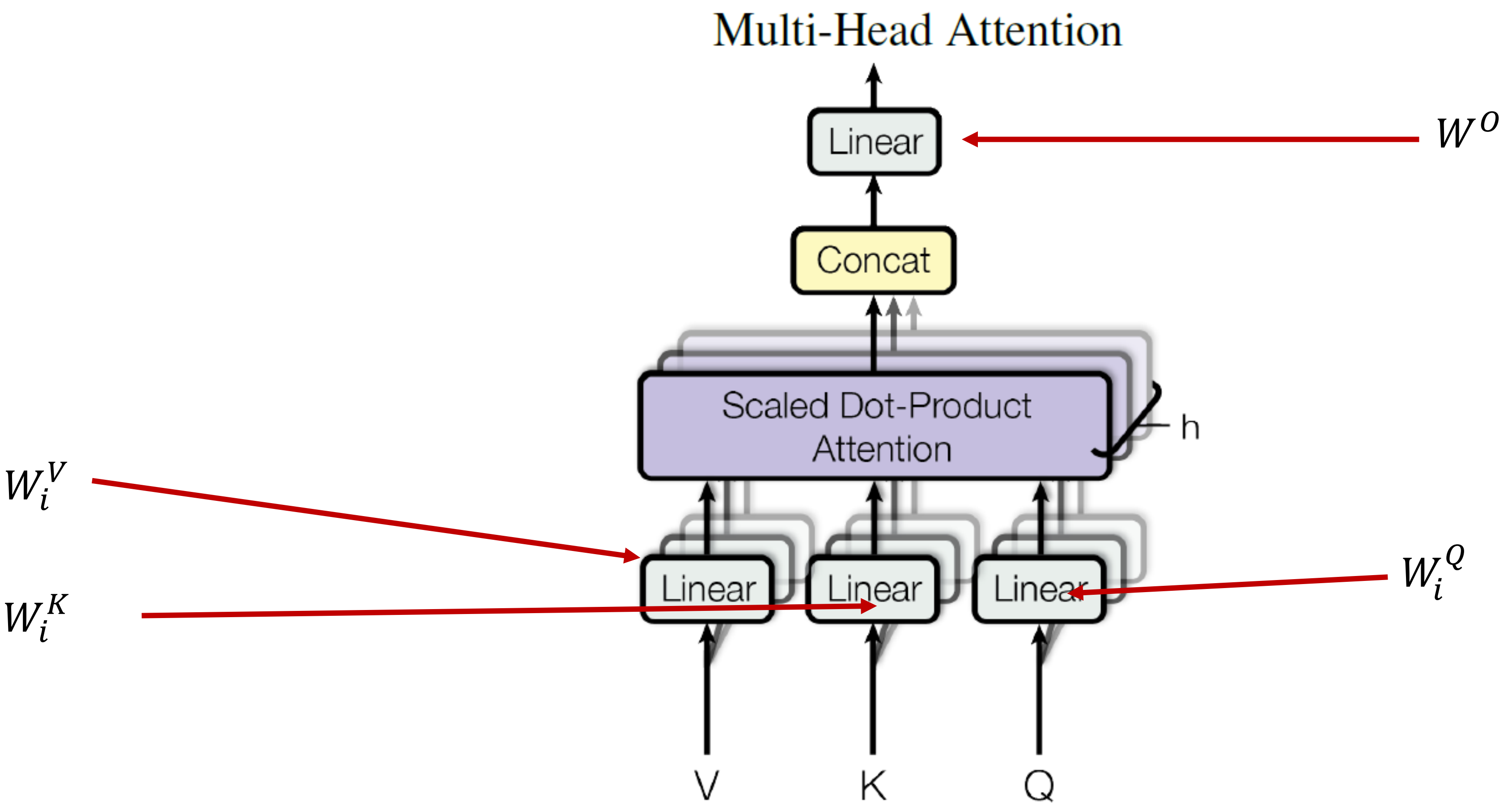
Multiply with W^O to produce d_m dimensional input for the next layer

Multi-Headed Attention

Input to the next layer is again a sequence of dimension d_m - model size

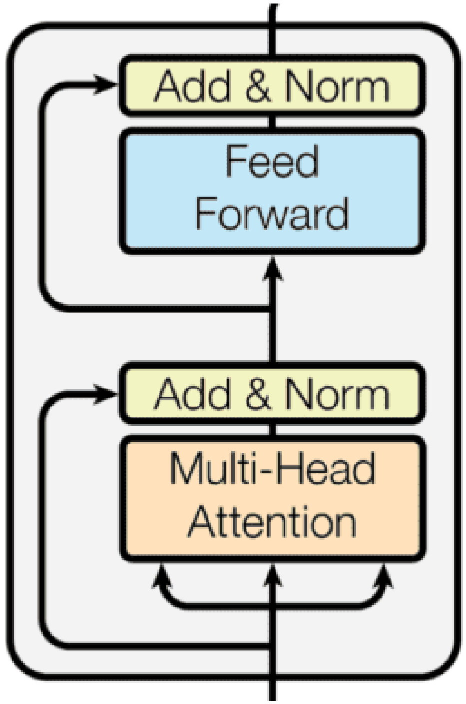


Multi-head Attention



Transformer Block

- Contains two sub-layers
  - Attention Mechanism
  - Feed Forward Network



Feed-Forward Network

- Simply a 2 layer fully connected network
- Input dimension = d_m
- Hidden dimension = d_{ff}
- Output dimension = d_m

Computationally

- Operations in each Transformer Block

- QW_i^Q, KW_i^K, VW_i^V
  - 3 Matrix Multiplications
  - $3k$ Matrix Multiplications for k heads

- Attention(QW_i^Q, KW_i^K, VW_i^V)
  - 2 Matrix Multiplications per attention
  - $2k$ Matrix Multiplications for k heads
  - 1 Matrix Multiplication after concatenation

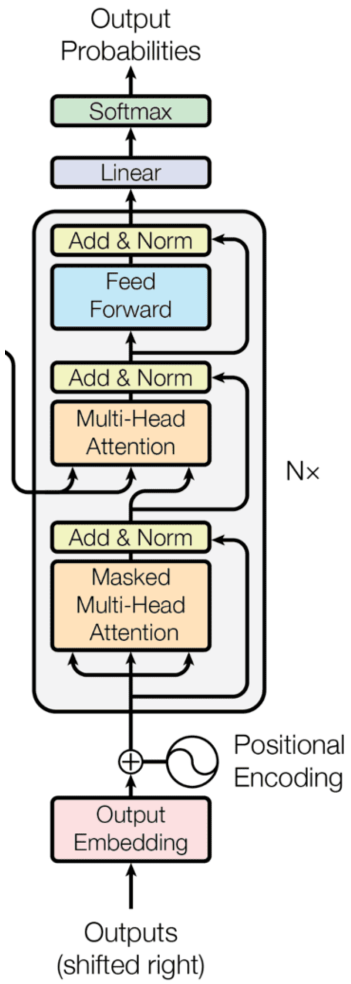
- Feedforward Network
  - 2 Matrix Multiplications

Transformer Decoder Block

- Contains three sub-layers
  - Attention Mechanisms
  - Feed Forward Network

$(z_1, z_2, \dots, z_n)$

- Total Matrix Multiplications:
  - $10k$ for the attention mechanisms
  - 1 for feed forward network



Outline

- Project Information
- Transformer Models
- Challenges in Transformer Models

Learnable Parameters

- For each Transformer Block
- Projection Matrices: $W_i^Q, W_i^K, W_i^V \in R^{d_m \times d_k/d_v}$
- Output of Concatenate Matrix: $W^O \in R^{hd_v \times d_m}$
- Feed Forward Matrices: $W_{1ff} \in R^{d_m \times d_{ff}}, W_{2ff} \in R^{d_{ff} \times d_m}$

Matrix Multiplication Operations

- Assuming a sequence length of l
- Projection Matrices: $\mathrm{O}(l \times d^m \times d^{k/v})$
  - Multiply l K,Q, V of d^m dimensions with the projection matrices
- Multi-head Attention: $\mathrm{O}(l \times d^k \times l), \mathrm{O}(l \times l \times d^v)$
  - QK product: First matrix multiplication calculates similarities between all pairs of the sequence $l \times l$
  - Softmax-V product: Second matrix multiplication computes weighted sum for all elements of sequence using all values (again elements of the sequence)

Matrix Multiplication Operations

- Output of Concatenate Matrix: $O(l \times hd^v \times d_m) = O(l \times d_m \times d_m)$

- Feedforward Matrix Multiplications:
  - $O(l \times d_m \times d_{ff})$
  - $O(l \times d_{ff} \times d_m)$

- Main Bottleneck: $O(l^2)$ dependency on the sequence length
  - Limits the number of tokens that models can take

Memory Requirements

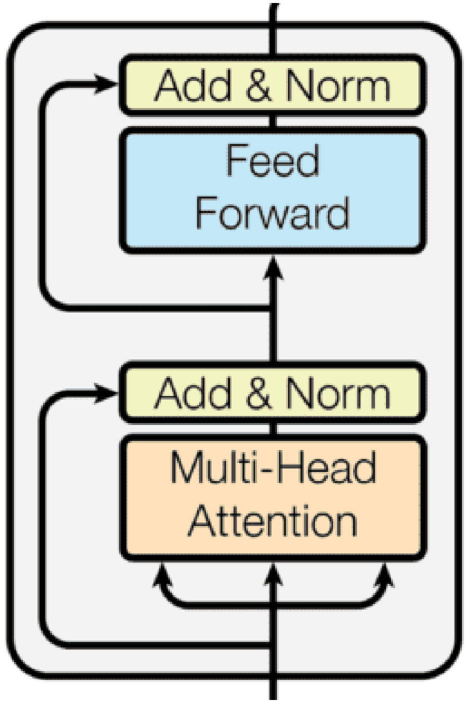
- Training/Fine-tuning: Learn the LLM model using a large corpus of data

- Inference: Query the model to obtain output

Memory Requirements

- Inference

- Need to store
  - Projection matrices for each Transformer block
  - Concatenate Output matrices for each Transformer block
  - Feedforward weights for each Transformer block
  - A few copies of Inputs/Activations need to be stored at any given time (to create K, Q, output of layers, etc.)

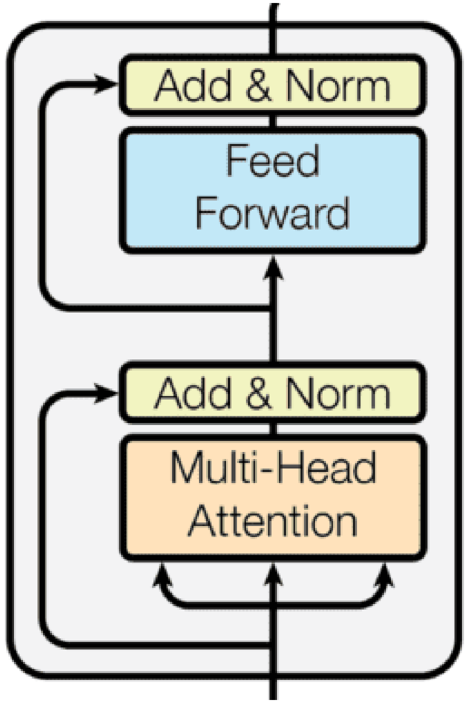


Memory Requirements

- Inference

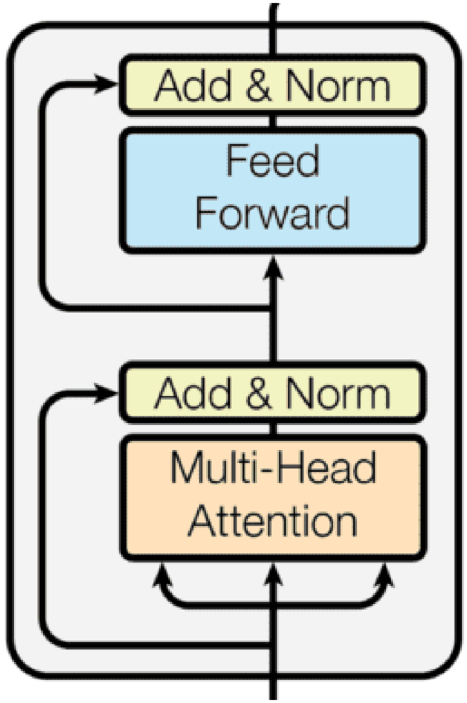
- N number of Encoder/Decoder Layers

- Need to store
  - Projection matrices for each Transformer block - $O(d_m \times d_{k/v} \times N \times h)$
  - Concatenate Output matrices for each Transformer block - $O(N \times d_m \times d_m)$
  - Feedforward weights for each Transformer block - $O(N \times d_m \times d_{ff})$
  - A few copies of Inputs/Activations need to be stored at any given time (to create K, Q, output of layers, etc.) - $O(l \times d_m)$



Memory Requirements

- Training/Fine-tuning
- N number of Encoder/Decoder Layers
- Need to store
  - Projection matrices for each Transformer block and their gradients - $O(d_m \times d_{k/v} \times N \times h)$
  - Concatenate Output matrices for each Transformer block and their gradients - $O(N \times d_m \times d_m)$
  - Feedforward weights for each Transformer block and their gradients - $O(N \times d_m \times d_{ff})$
  - **All** Inputs/Activations need to be stored at any given time (to create K, Q, output of layers, etc.) - $O(N \times l \times d_m)$

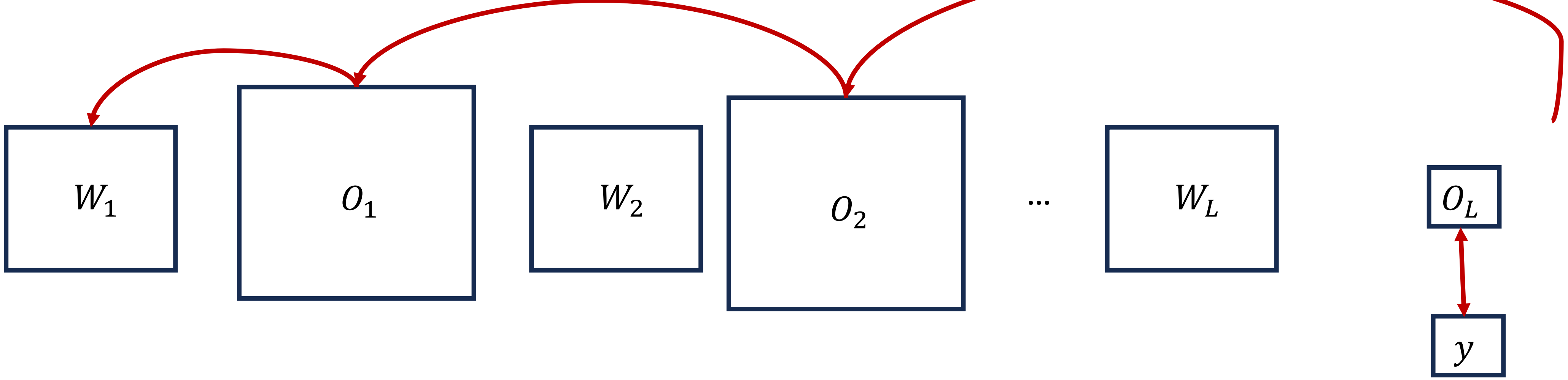


Memory Requirements

- Why do we need to store ALL Inputs/Activations?

Memory Requirements

- Why do we need to store ALL Inputs/Activations? Recall from CNN.
- Here, we need matrix multiplications instead of Conv



$$\frac{\delta E(W)}{\delta W_1} = \frac{\delta O_1}{\delta W_1} \times \frac{\delta O_2}{\delta O_1} \times \frac{\delta O_3}{\delta O_2} \times \cdots \times \frac{\delta O_L}{\delta O_{L-1}} \times \frac{\delta E(W)}{\delta O_L}$$

Memory Requirements – Going Deeper

- Excellent paper that explains the memory requirements by Microsoft Research
  - Rajbhandari, S., Rasley, J., Ruwase, O., & He, Y. (2020, November). ***Zero: Memory optimizations toward training trillion parameter models.*** In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis* (pp. 1-16). IEEE.

Memory Requirements – Going Deeper

- Analysis from the paper
- Consider 1.5 Billion parameter GPT-2 model.
  - Each parameter requires 2 bytes
- Total memory needed for the parameters???

Memory Requirements – Going Deeper

- Analysis from the paper

- Consider 1.5 Billion parameter GPT-2 model.
  - Each parameter requires 2 bytes

- Total memory needed for the parameters – 3 GB

Memory Requirements – Going Deeper

- Consider a GPU with 32 GB memory
- Consider the GPT model that requires 3 GB memory.
- Can we train it on the GPU???

Memory Requirements – Going Deeper

- Consider a GPU with 32 GB memory
- Consider the GPT model that requires 3 GB memory.
- Can we train it on the GPU – No
- Optimizer states and residual memory consume a significant portion of memory

Memory Requirements – Going Deeper

- Consider a model with P parameters, with ADAM optimizer
- Memory for parameters: $2P$ bytes
- Memory for gradients: $2P$ bytes
- For our GPT model this (gradients, parameters) is 6 GB.

Memory Requirements – Going Deeper

- Consider a model with P parameters, with ADAM optimizer

- Memory to store parameters for ADAM optimizer state : $4P$ bytes (floating points stored for improved precision)

- Memory to store momentum for ADAM: $4P$ bytes

- Memory to store variance for ADAM optimizer state : $4P$ bytes

- For our GPT model this (optimizer states) is 12 x 1.5 GB = 18 GB.

Memory Requirements – Going Deeper

- So, total memory requirement - $4P + KP$ bytes, where K is the memory requirement factor due to the optimizer

- In our example, $K = 12$, so total memory needed = 24 GB

- Other memory requirements
  - Activations
  - Temporary buffers
  - Memory fragmentation

Memory Requirements – Going Deeper

- Memory for Activations

- Our GPT model with 1K sequence length and batch size of 32 will require 60 GB (equation provided in the paper)

Accelerating Transformer Models

- To summarize
- Learnable Parameters – Projection matrices, feedforward networks consume memory
- Inputs and activations consume memory
- However, in practice the temporary matrix that we create by the product of K and Q has the biggest memory footprint. $O(l^2)$.
  - This is what we will focus in this class

Accelerating Transformer Models

- Ungraded HW assignment
- Read the Microsoft Paper and understand as much as you can
- We did not have time to go into it detail now. We will revisit this paper and related optimizations when we do distributed training.

Accelerating Transformer Models

- Approaches for Accelerating LLMs

- Tiled attention calculation to improve cache reuse: Flashattention
- Sparsity to reduce the number of attentions or quantization
- Brief Overview of Other techniques: Kernel methods, alternates to attention for sequence modeling, KV Caching
- Distributed Training

CASE SCHOOL OF ENGINEERING
CASE WESTERN RESERVE UNIVERSITY

Next Class

- 10/23 Lecture 16
  - Accelerating Transformer Model: Sparsity

Thank You

- Questions?

- Email: sanmukh.kuppannagari@case.edu