IT5005 Artificial Intelligence

Sirigina Rajendra Prasad
AY2025/2026: Semester 1

Transformers

Self-supervised Learning

- Model is trained on unlabeled data by having the model generate its labels from data itself
- **NLP Example**: Next-token prediction
  - Unlabeled data: "*Singapore is known for its efficient public transport system*"

"*Singapore*" ---> Target token: "*is*"
"*Singapore is*" ---> Target token: "*known*"
"*Singapore is known*" ---> Target token: "*for*"
"*Singapore is known for*" ---> Target token: "*its*"
"*Singapore is known for its*" ---> Target token: "*efficient*"
"*Singapore is known for its efficient*" ---> Target token: "*public*"
"*Singapore is known for its efficient public*" ---> Target token: "*transport*"
"*Singapore is known for its efficient public transport*" ---> Target token: "*system*"

Self-supervised Learning

- Model is trained on unlabeled data by having the model generate its labels from data itself
- **NLP Example**: Masked Language Model
  - Unlabeled data: "*Singapore is known for its efficient public transport system*"

"*Singapore is _____ for its efficient public transport system*"

Target token: "*known*"

"*Singapore is known for its ____ public transport system*"

Target token: "*efficient*"

Transformers: Motivation

- Language modeling and masked language modelling
  - Can be modelled using Markov chains
  - Transition probabilities capture the relation between words

- **Issue**: Scaling
  - Context window size is equivalent to order of Markov chains
  - Memory and computational complexity is exponential in context window size

Other alternatives for sequential data

- Convolutional Neural Networks (1-D kernels)
- Recurrent Neural Networks
- Long Short Term Memory (LSTM) and its variants (SOTA till 2017)

CNNs for Sequential Data

- 1-D kernels to capture temporal relations

Only local context and position-agnostic

Recurrent Neural Networks

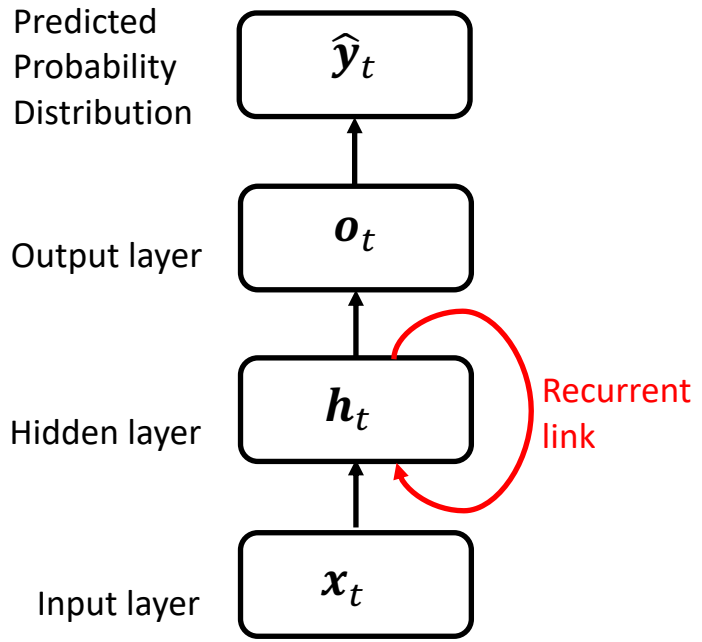
- Accept dynamic or sequence data of unequal length

$\boldsymbol{x}_t$: Input at time t

$\boldsymbol{h}_t$: Hidden layer at time t

$\boldsymbol{o}_t$: Output vector at time t

$\hat{\boldsymbol{y}}_t$: Output of SoftMax layer at time t

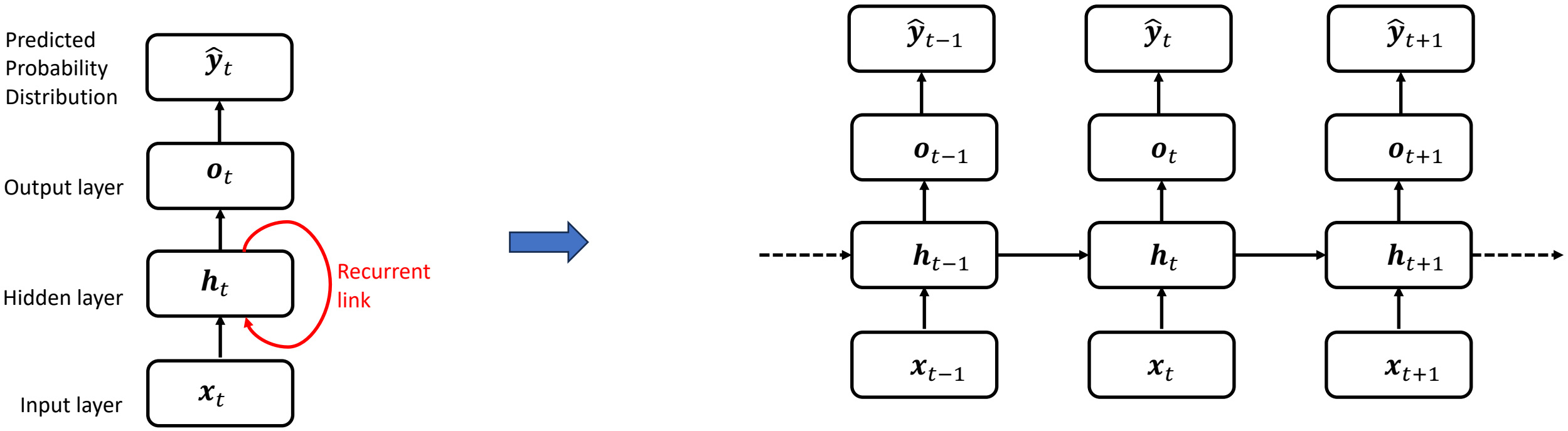


- Recurrent link
  - Output is feedback as input

- Recurrent link can be unrolled to handle variable length data

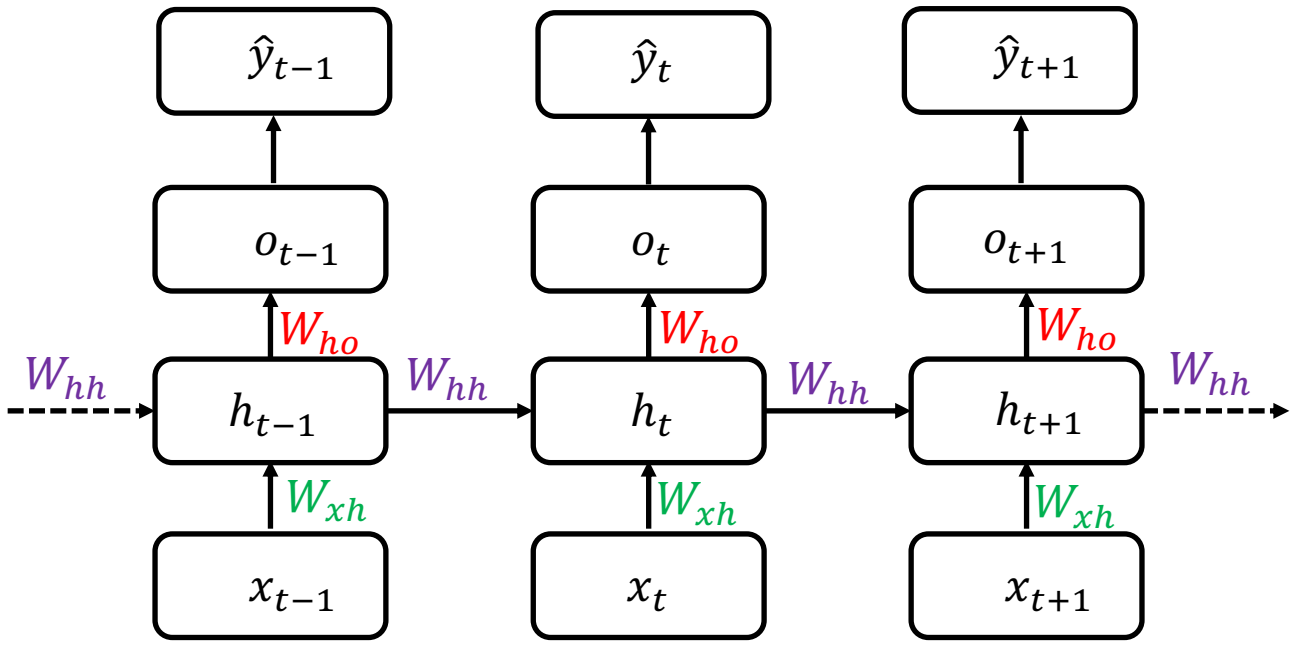
RNN: Unrolled

Only for your reading



- Input sequences can be of different lengths
- Hidden layers from previous time instant provides context or memory

RNN: Unrolled



Reuse the same weight matrix W_{xx} at each time step

$$\boldsymbol{h}_t = f(\boldsymbol{a}_t)$$

where $\boldsymbol{a}_t = W_{xh}\boldsymbol{x}_t + W_{hh}\boldsymbol{h}_{t-1} + \boldsymbol{b}$

$W_{xh} \in \mathbb{R}^{k \times d}$

$W_{hh} \in \mathbb{R}^{k \times k}$

$\boldsymbol{b} \in \mathbb{R}^{k}$

f is Tanh or ReLU

$$\hat{y}_t = g(\boldsymbol{o}_t)$$

where $\boldsymbol{o}_t = W_{ho}\boldsymbol{h}_t + \boldsymbol{c}$

$W_{ho} \in \mathbb{R}^{|V| \times k}$

$\boldsymbol{c} \in \mathbb{R}^{|V|}$

g is softmax

RNNs for Language Modelling

- RNNs model the joint probability $P(x_1, x_2, \dots, x_n)$, where $x_1, x_2, \dots, x_n$ represent sequence of words.

$$P(x_1, x_2, \dots, x_n) = \prod_{t=1}^{n} P(x_t | x_{t-1}, x_{t-2}, \dots, x_1)$$

- Useful for many applications involving text/sentence generation
  - Prediction of next token, machine translation, speech recognition, question answering , etc.

Prediction of Next Token

During Training

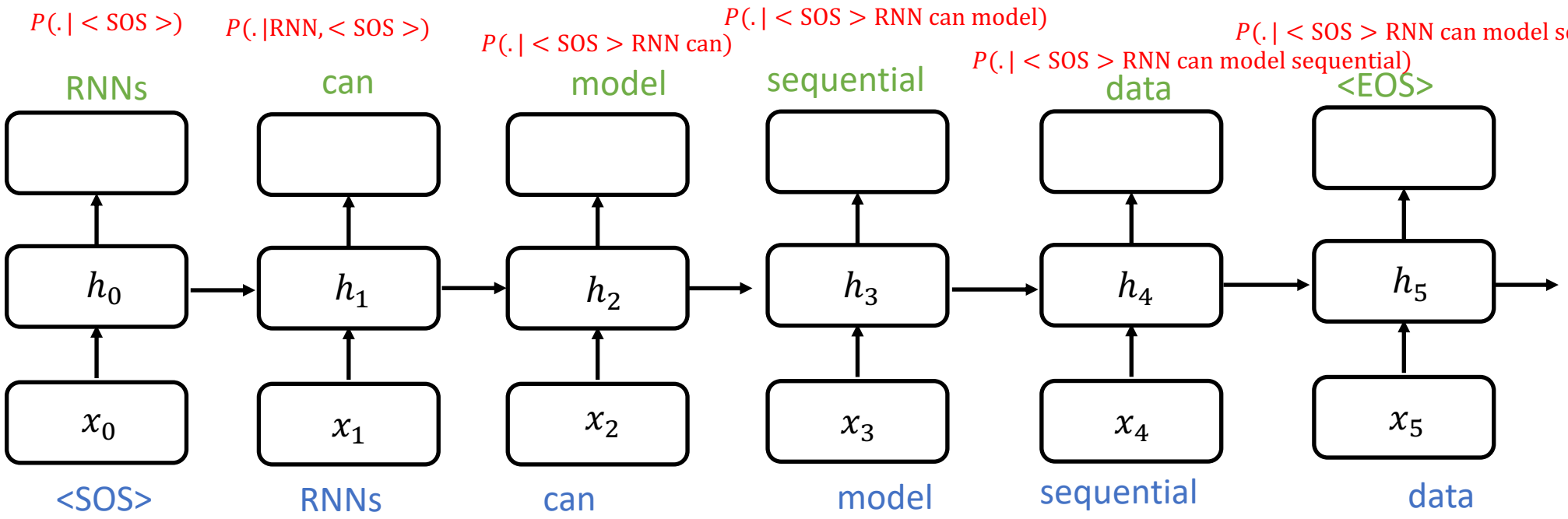
"RNNs can model the sequential data"

Input	Ground Truth	Output Distribution
x_o = "<BOS>"	y_o: "RNNs"	$P(.\|x_0)$
$x_{o:1}$ = "<BOS> RNNs"	y_1: "can"	$P(.\|x_{0:1})$
$x_{o:2}$ = "<BOS> RNN can"	y_2: "model"	$P(.\|x_{0:2})$
$x_{o:3}$ = "<BOS> RNN can model"	y_3: "sequential"	$P(.\|x_{0:3})$
$x_{o:4}$ = "<BOS> RNN can model sequential"	y_4: "data"	$P(.\|x_{0:4})$
$x_{o:5}$ = "<BOS> RNN can model sequential data"	y_5: "<EOS>"	$P(.\|x_{0:5})$

Prediction of Next Token

During Training

"RNNs can model the sequential data"



Input
Ground truth
Output distribution

Prediction of Next Token

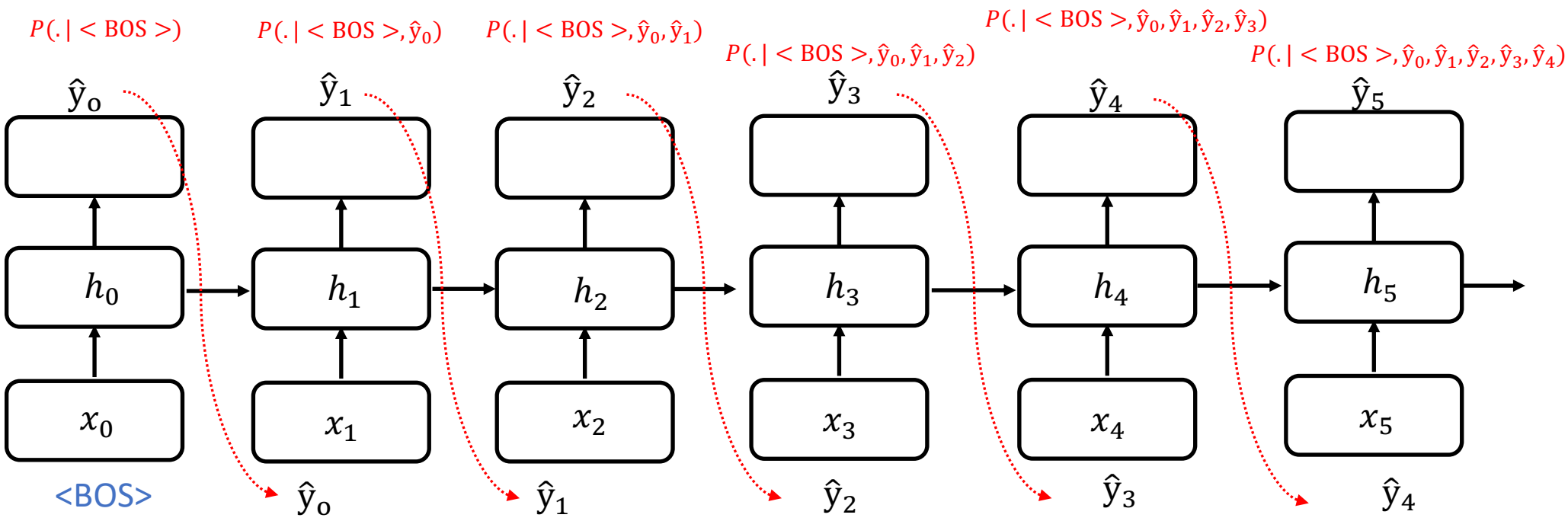
During Inference

"RNNs can model the sequential data"

Input	Predicted	Output Distribution
x_o = <BOS>	$\hat{y}_o$	$P(.\|x_0)$
$x_{o:1}$ = <BOS> $\hat{y}_o$	$\hat{y}_1$	$P(.\|x_{0:1})$
$x_{o:2}$ = <BOS> $\hat{y}_o, \hat{y}_1$	$\hat{y}_2$	$P(.\|x_{0:2})$
$x_{o:3}$ = <BOS> $\hat{y}_o, \hat{y}_1, \hat{y}_2$	$\hat{y}_3$	$P(.\|x_{0:3})$
$x_{o:4}$ = <BOS> $\hat{y}_o, \hat{y}_1, \hat{y}_2, \hat{y}_3$	$\hat{y}_4$	$P(.\|x_{0:4})$
$x_{o:5}$ = <BOS> $\hat{y}_o, \hat{y}_1, \hat{y}_2, \hat{y}_3, \hat{y}_4$	$\hat{y}_5$	$P(.\|x_{0:5})$

Prediction of Next Token

During Inference

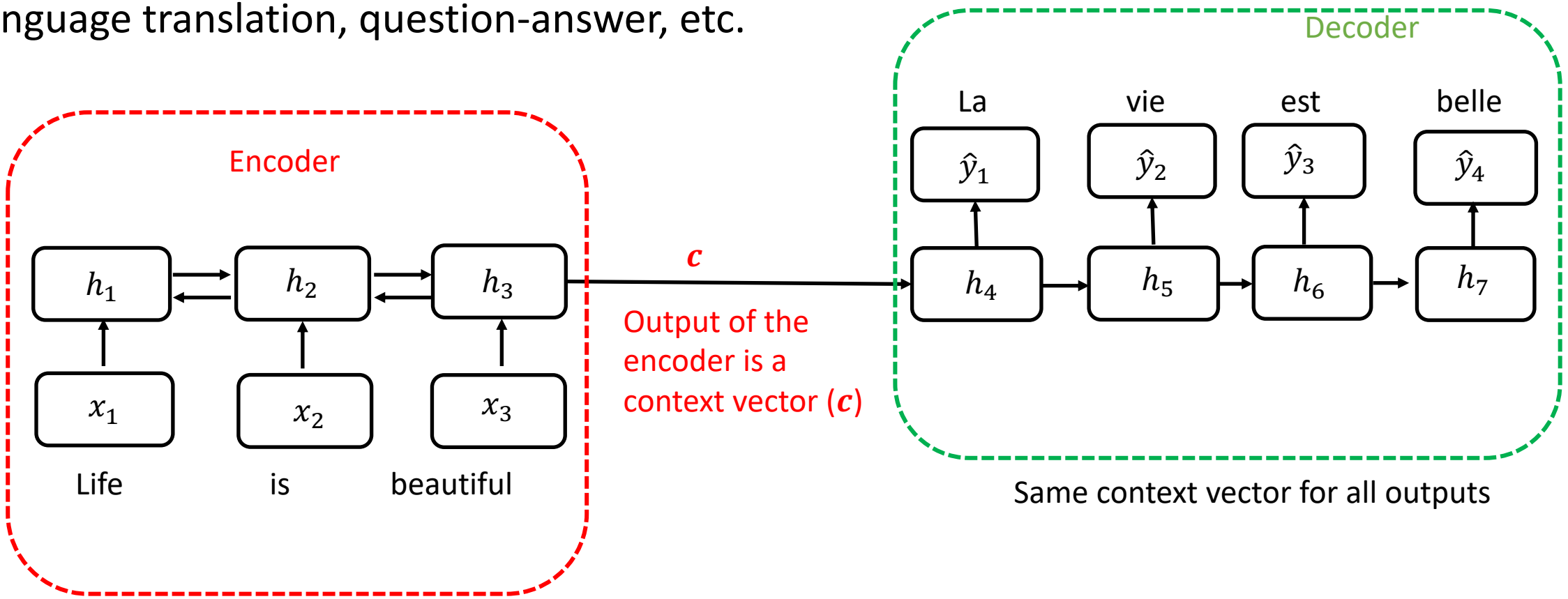


Input

Output distribution

Sequence-to-Sequence Model or Encoder-Decoder Model

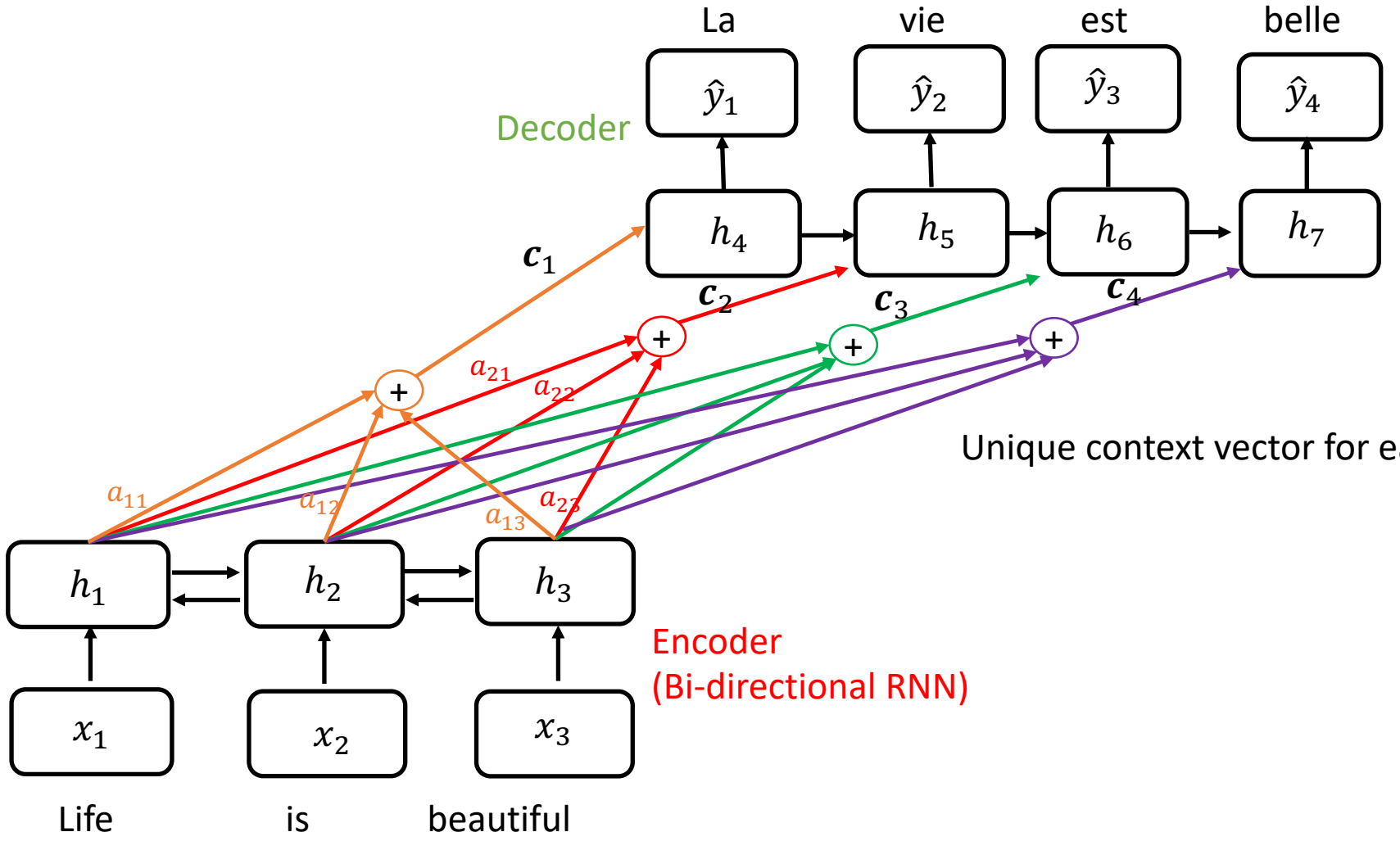
- RNNs process the entire input *sequentially* and then produce the output
- **Applications**:
  - language translation, question-answer, etc.



Context vector (c) is called “bottleneck”

Variants of RNNs: RNNs with Attention

Only for your reading



Unique context vector for each output

$$c_i = \sum_{j=1}^{T} a_{ij} h_j$$

$$a_{ij} = \frac{exp(e_{ij})}{\sum_{k=1}^{T} exp(e_{ik})}$$

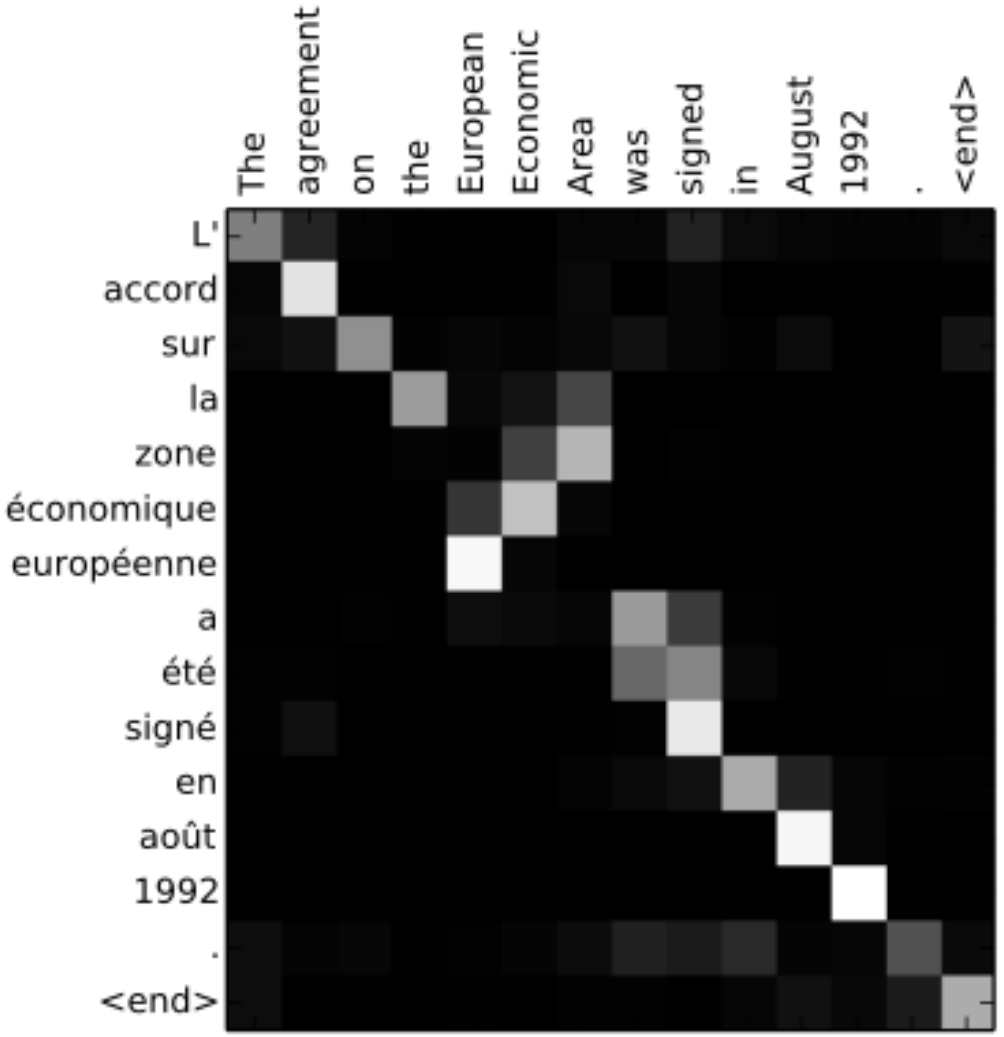
$$e_{ij} = a(y_i, h_j)$$

Neural Machine Translation by Jointly Learning to Align and Translate

Code: https://docs.pytorch.org/tutorials/intermediate/seq2seq_translation_tutorial.html

Variants of RNN: RNN with Attention

Only for your reading



$$\textbf{Attention Matrix} = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix}$$

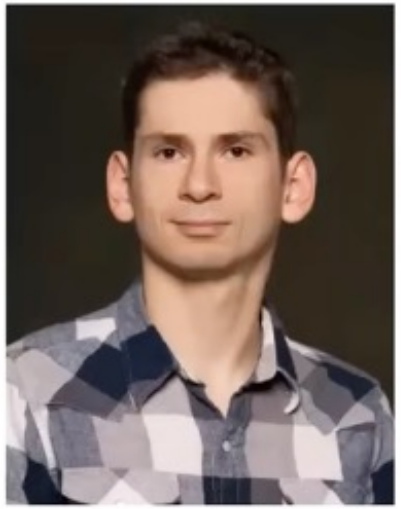
$$c_i = \sum_{j=1}^{T} a_{ij} h_j$$

$$a_{ij} = \frac{exp(e_{ij})}{\sum_{k=1}^{T} exp(e_{ik})}$$

$$e_{ij} = a(y_i, h_j)$$

Neural Machine Translation by Jointly Learning to Align and Translate

Code: https://docs.pytorch.org/tutorials/intermediate/seq2seq_translation_tutorial.html



Dzmitry Bahdanau

RE: history of “attention”
(from email correspondence)

So I started thinking about how to avoid the bottleneck between encoder and decoder RNN. My first idea was to have a model with two "cursors", one moving through the source sequence (encoded by a BiRNN) and another one moving through the target sequence. The cursor trajectories would be marginalized out using dynamic programming. KyungHyun Cho recognized this as an equivalent to Alex Graves' RNN Transducer model. Following that, I may have also read Graves' hand-writing recognition paper. The approach looked inappropriate for machine translation though.

The above approach with cursors would be too hard to implement in the remaining 5 weeks of my internship. So I tried instead something simpler - two cursors moving at the same time synchronously (effectively hard-coded diagonal attention). That sort of worked, but the approach lacked elegance.

So one day I had this thought that it would be nice to enable the decoder RNN to learn to search where to put the cursor in the source sequence. This was sort of inspired by translation exercises that learning English in my middle school involved. Your gaze shifts back and forth between source and target sequence as you translate. I expressed the soft search as softmax and then weighted averaging of BiRNN states. It worked great from the very first try to my great excitement. I called the architecture RNNSearch, and we rushed to publish an ArXiV paper as we knew that Ilya and co at Google are somewhat ahead of us with their giant 8 GPU LSTM model (RNN Search still ran on 1 GPU).

As it later turned out, the name was not great. The better name (attention) was only added by Yoshua to the conclusion in one of the final passes.

Stanford

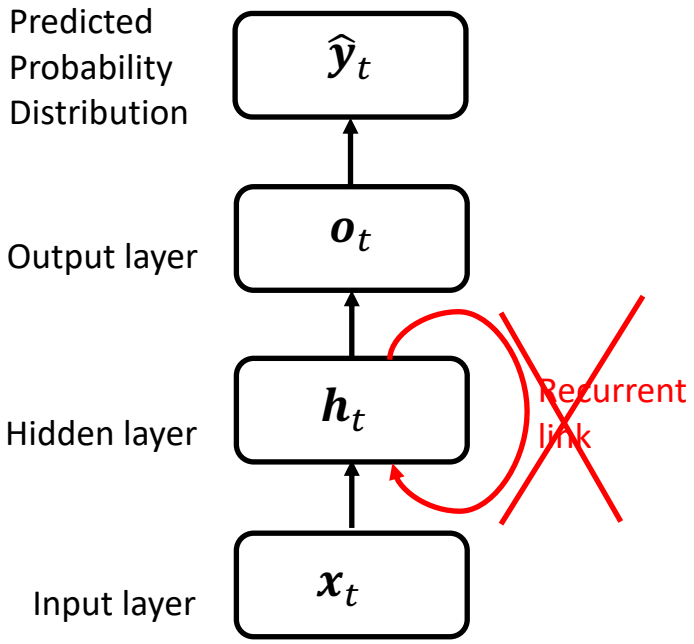
Screenshot from: https://www.youtube.com/watch?v=XfpMkf4rD6E

Issues with Sequential Models

- CNNs
  - Smaller context windows
  - Lack of positional information
- RNNs and LSTMs
  - Difficult to achieve parallelism due to sequential processing
  - Vanishing/exploding gradients for larger sequences

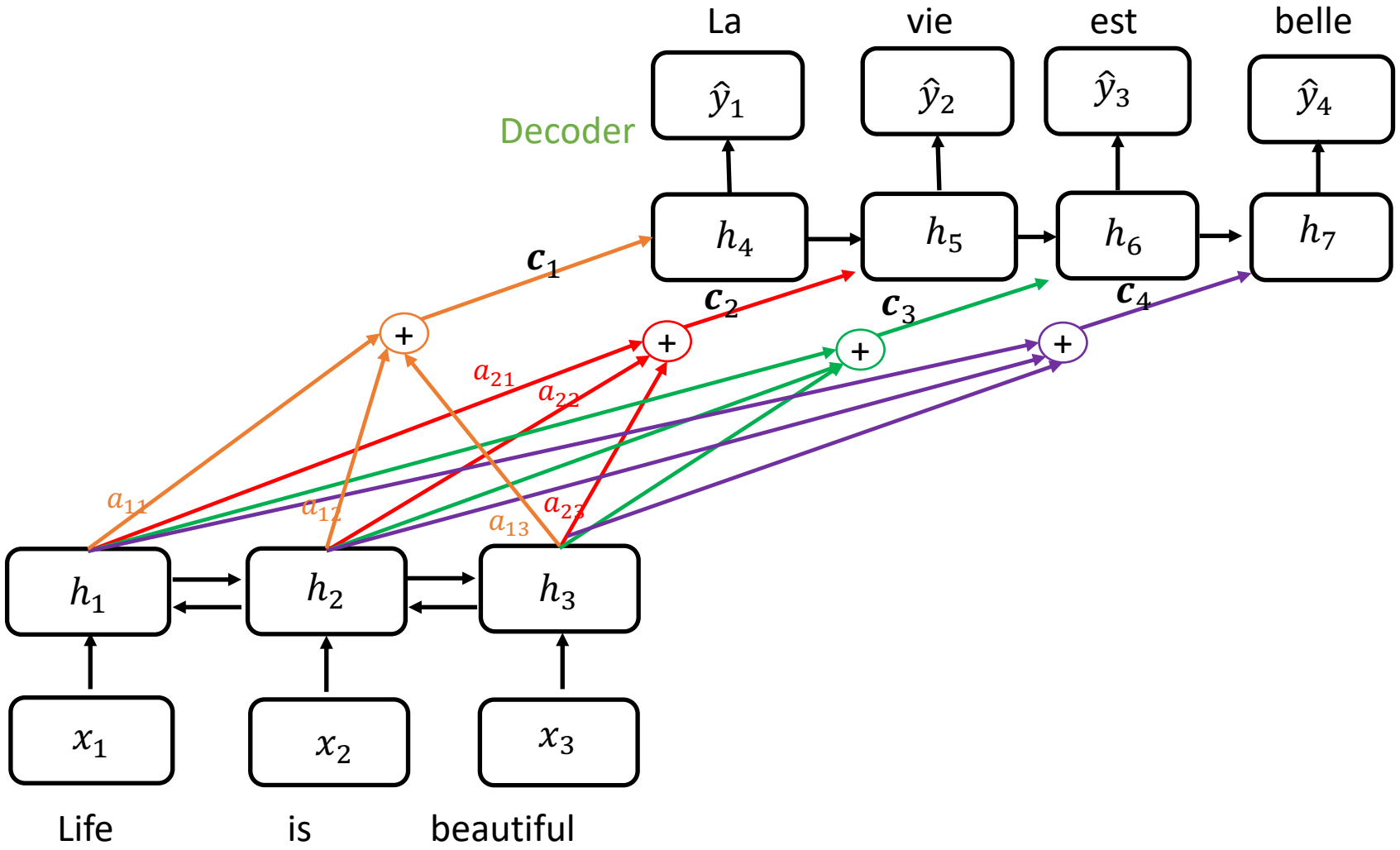
How transformers overcome this issue?

- Recursion Elimination



How transformers overcome this issue?

- Parallel processing all inputs simultaneously

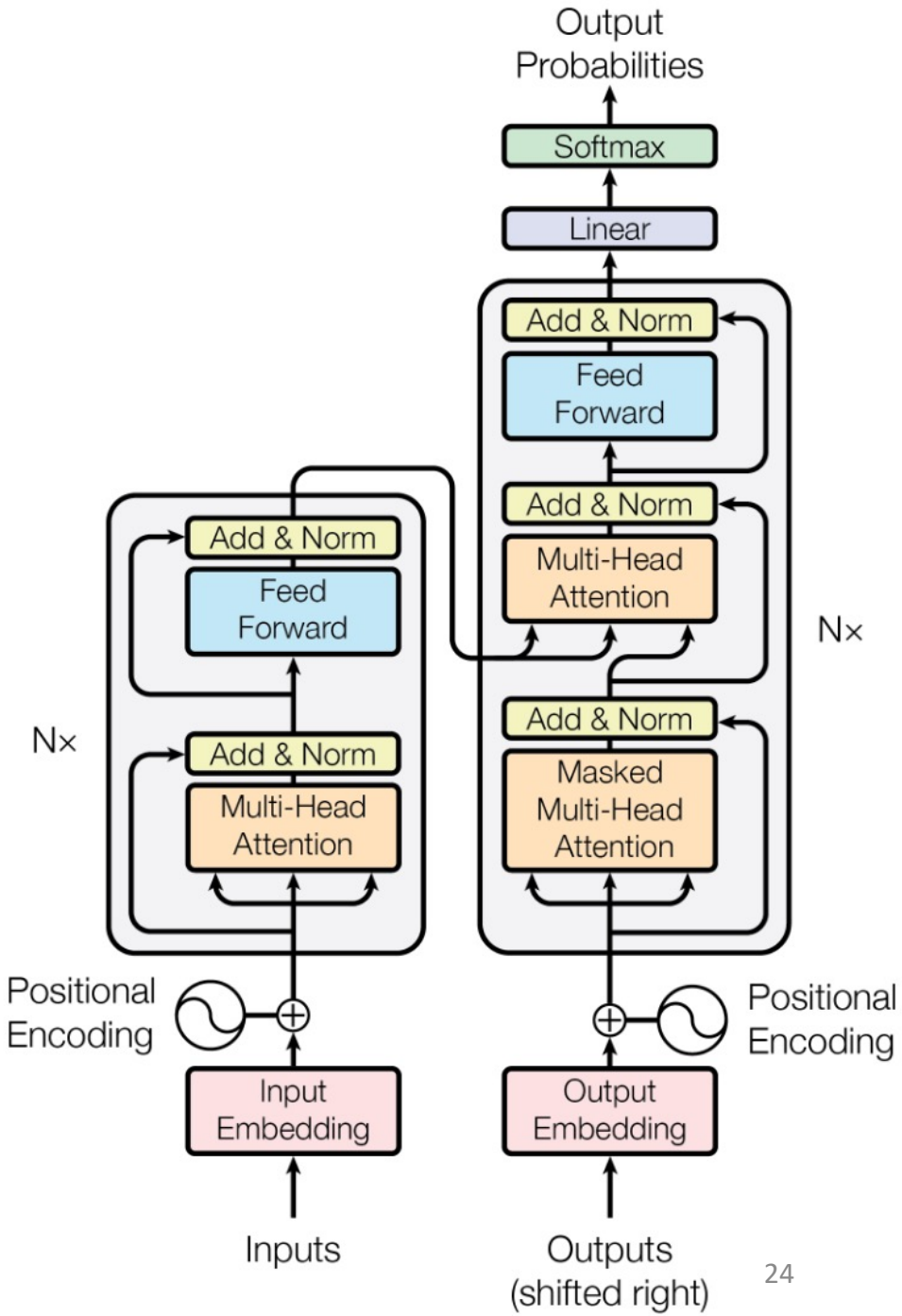


Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com
Noam Shazeer* Google Brain noam@google.com
Niki Parmar* Google Research nikip@google.com
Jakob Uszkoreit* Google Research usz@google.com
Llion Jones* Google Research llion@google.com
Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu
Łukasz Kaiser* Google Brain lukaszkaiser@google.com
Illia Polosukhin* ‡ illia.polosukhin@gmail.com

Abstract

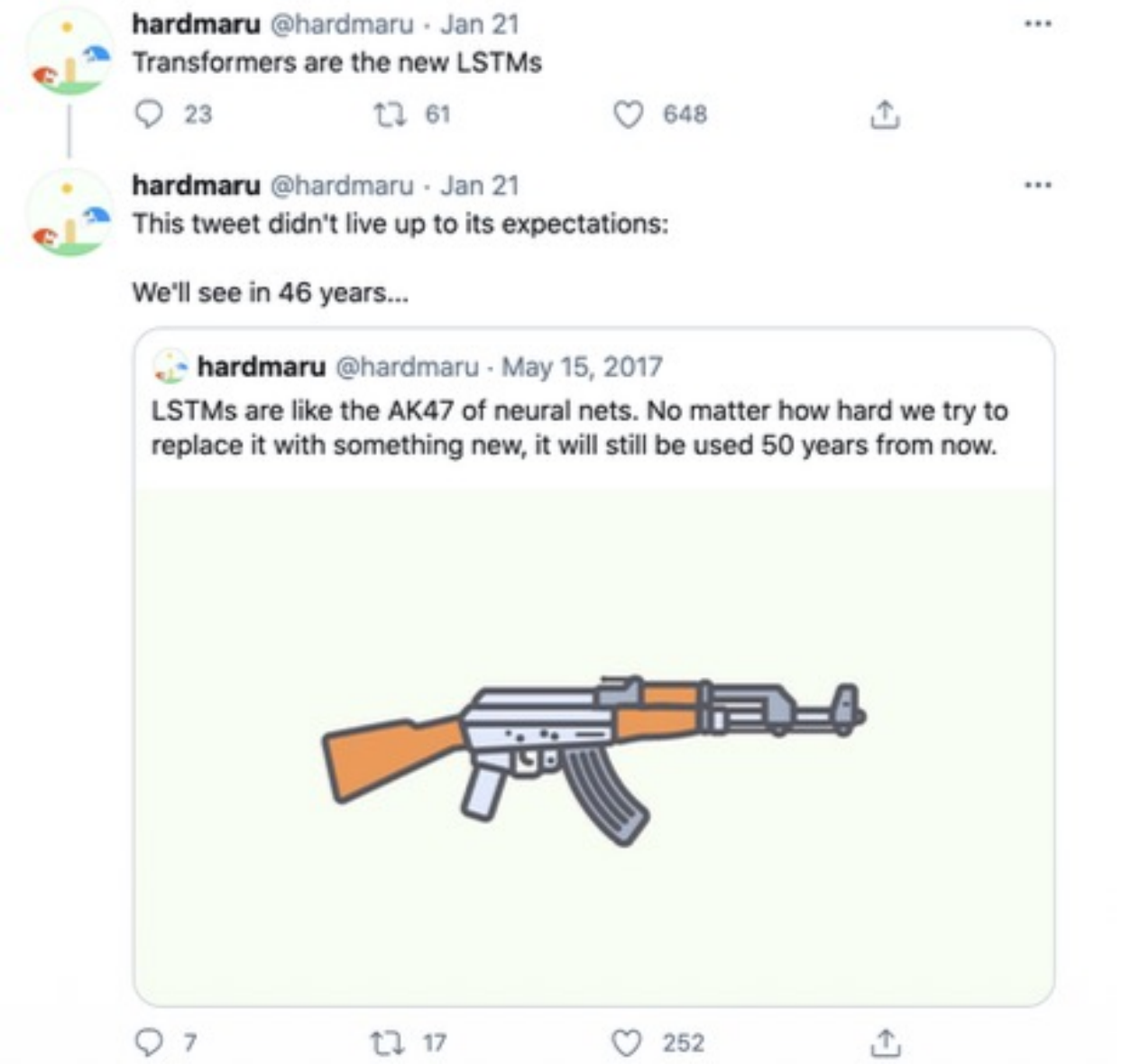
The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.8 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature. We show that the Transformer generalizes well to other tasks by applying it successfully to English constituency parsing both with large and limited training data.



Of course, we also need

1. Skip connections for deeper networks
2. Layer normalization
3. Positional encoding for explicit capture of sequence information

Transformers



Slide credit: Prof. You Yang (CS5242)

Transformers

- Transformers are friendly to modern hardware like TPUs and GPUs
  - We need a high parallelism to make full use of the computing units in TPUs and GPUs (e.g. thousands of threads)

Slide credit: Prof. You Yang (CS5242)

Transformer is transforming everything in deep learning

- All the attention of deep learning is now on Attention!

Slide credit: Prof. You Yang (CS5242)

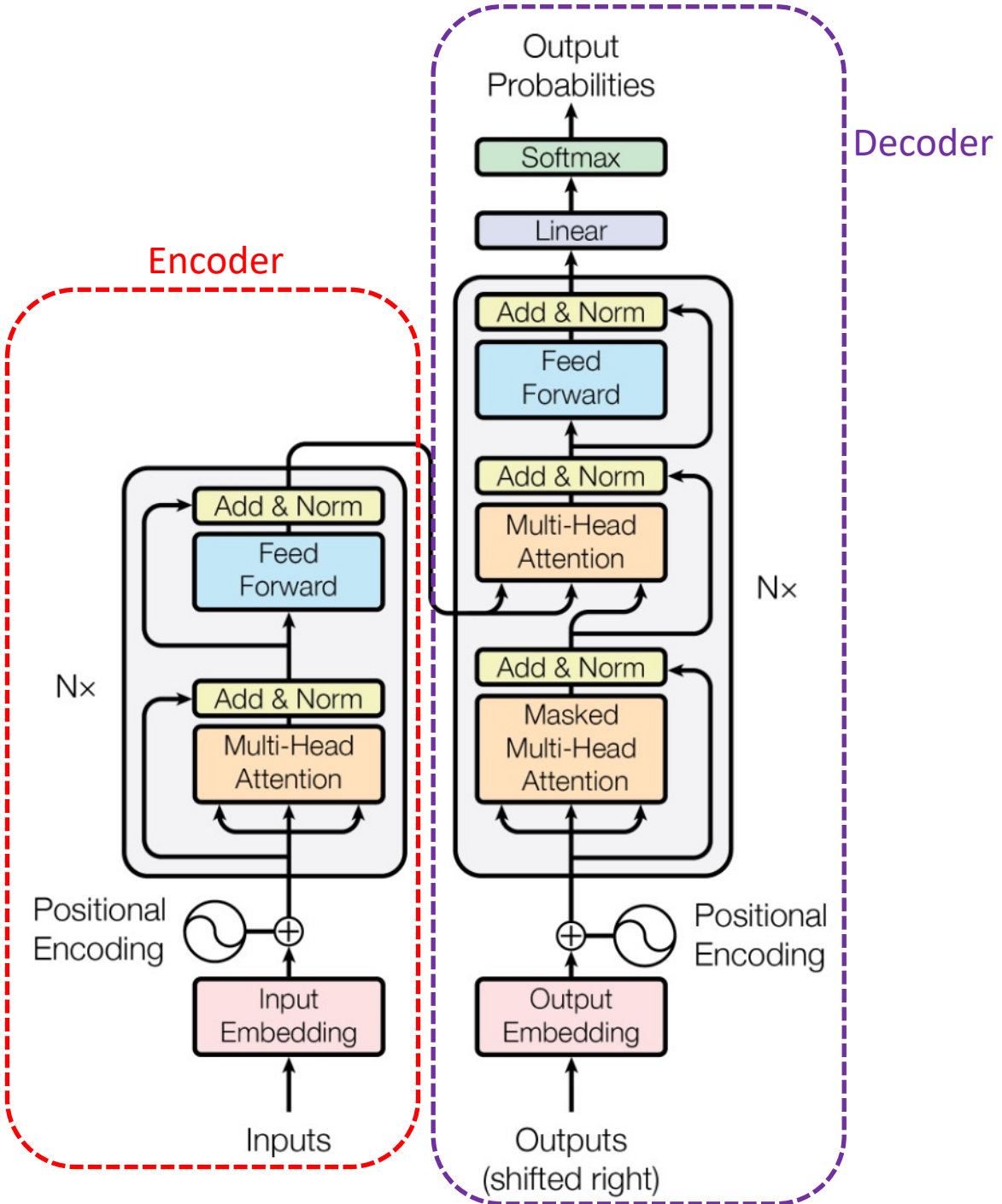
Transformers

- Prior to the transformers, research is fragmented across domains

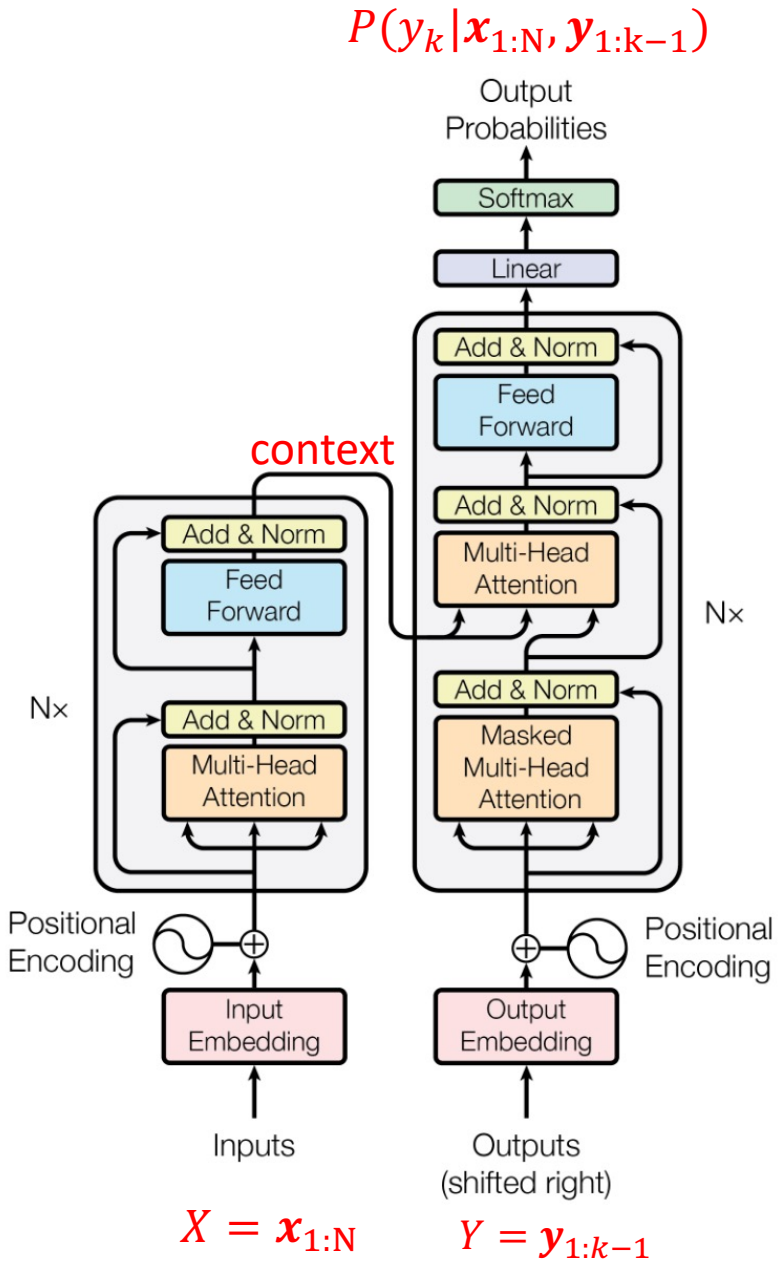
- Transformer is the unifying force
  - Single architecture for all tasks

generated using OpenAI's GPT Image 1 model.

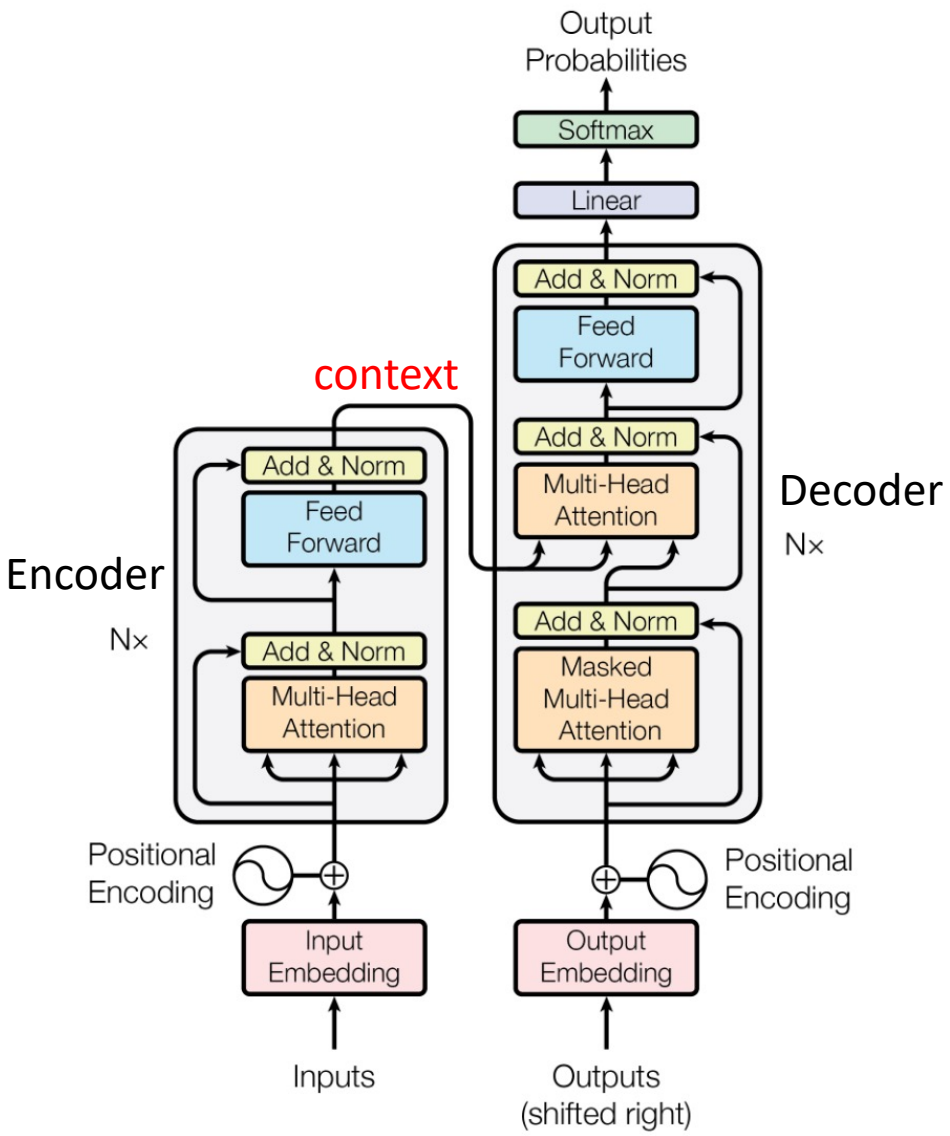
Transformer Architecture

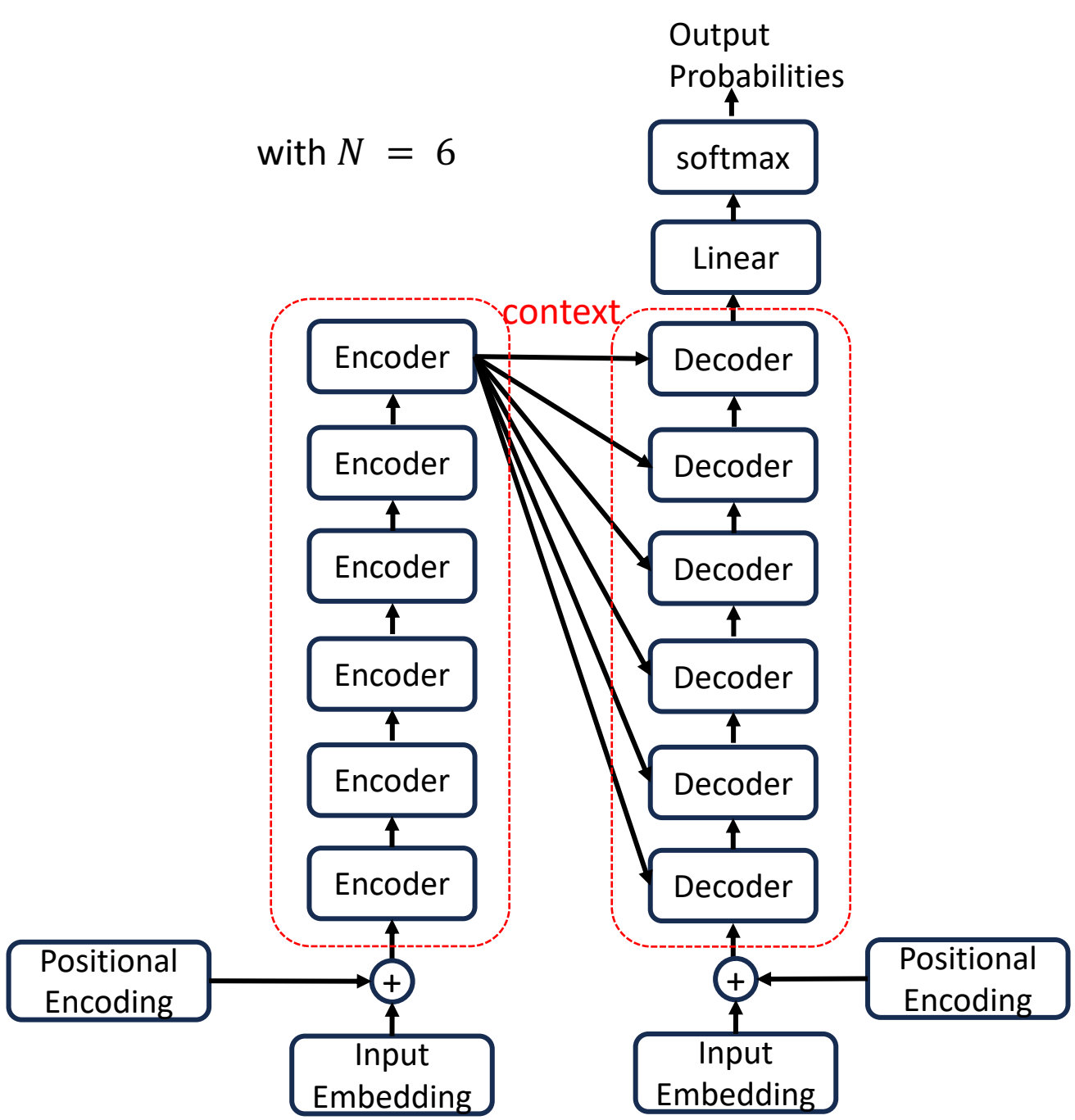


Transformer Architecture



Transformer Architecture





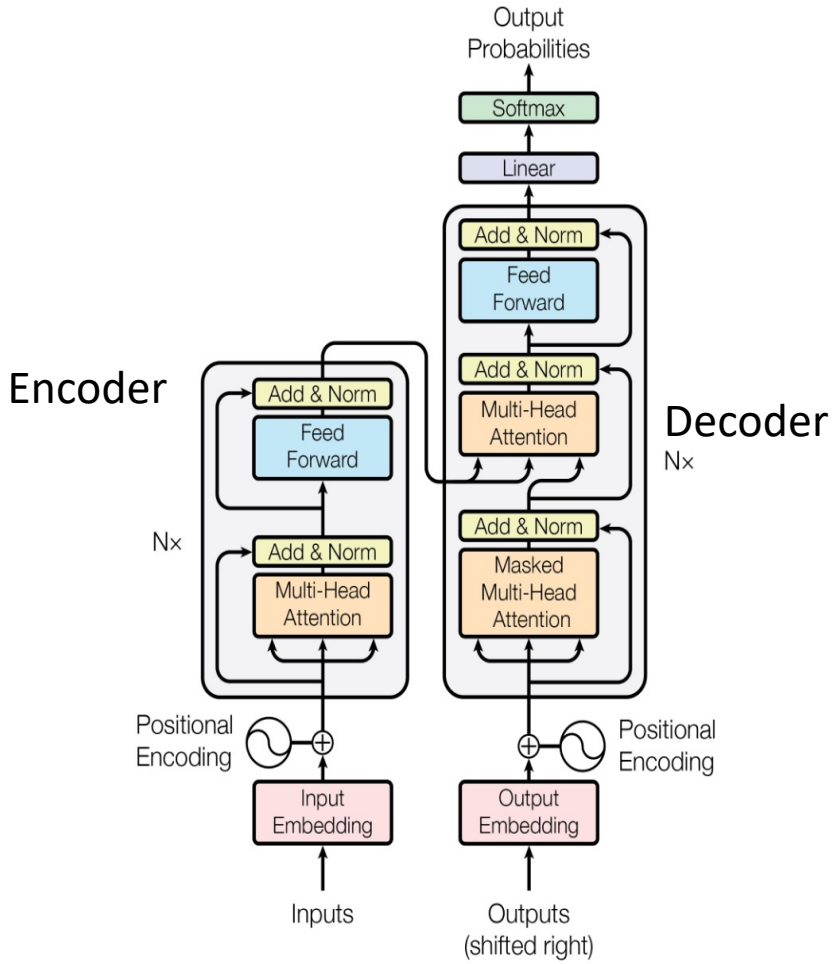
Variants of Transformer Architectures

- Both Encoder and Decoder
  - Language Translation

- Encoder-only
  - Eg: BERT

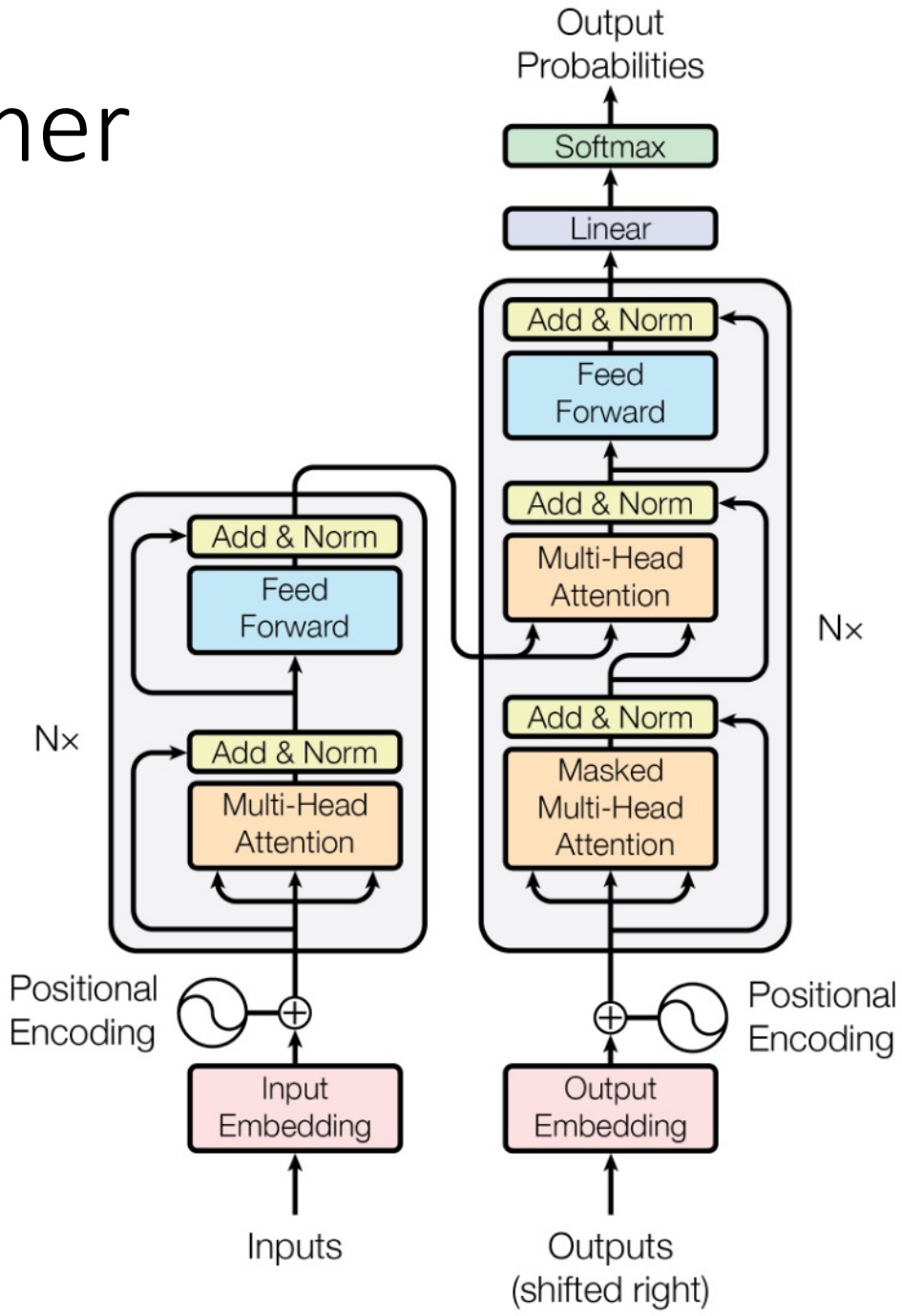
  (Bidirectional encoder representations from transformers)

- Decoder-only
  - Eg: GPT (Generative Pre-trained Transformer)



Building Blocks of Transformer

1. Tokenizer
2. Token Embedding
3. Positional Encoder
4. Multi-head Attention
   a. Self Attention
   b. Cross Attention
   c. Masked Attention
5. Skip Connection
6. Normalization
7. Feed Forward Network
8. Linear Layer
9. Softmax

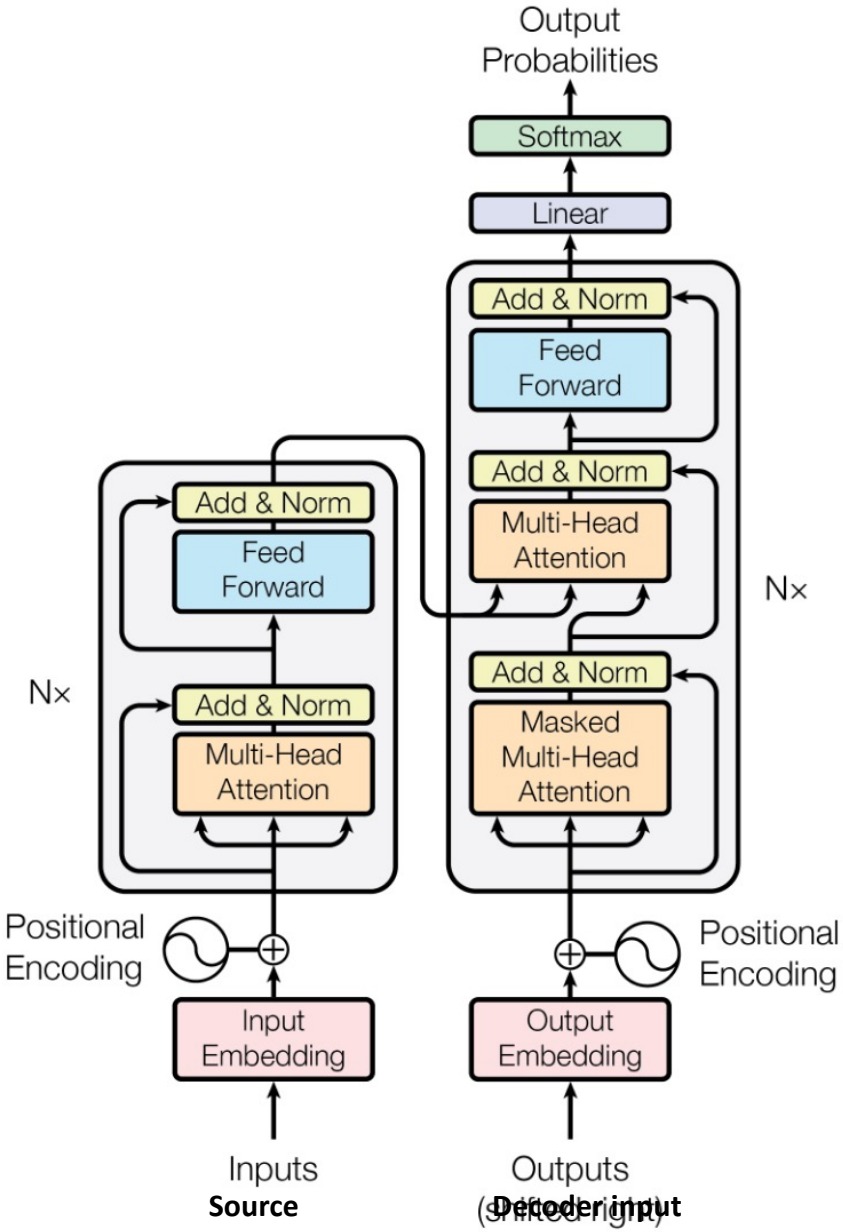


Language Translation: Example

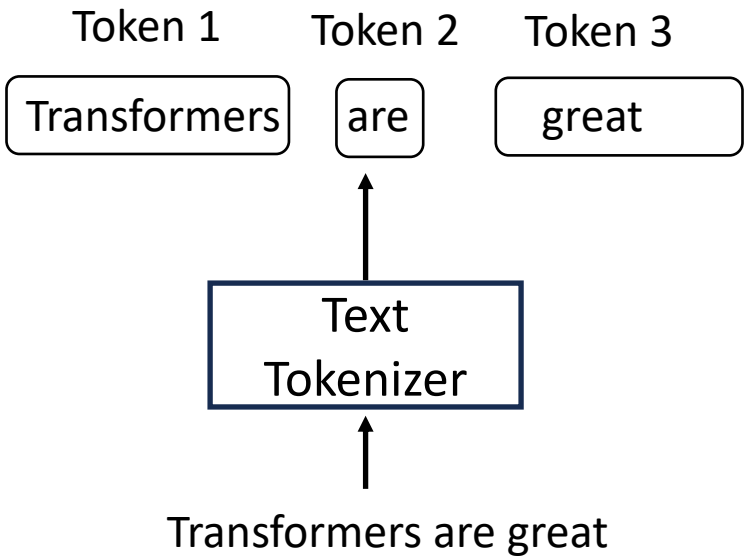
- **Source**: “Transformers are great”

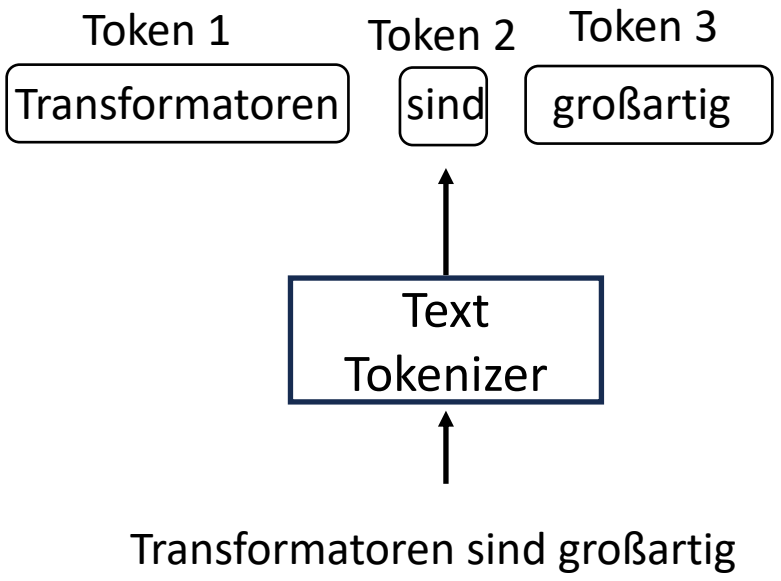
- **Target**: “Transformatoren sind großartig“

- During Training:
  - **Source (English):**
    - ["Transformers", "are", "great"]
  - **Target (German):**
    - ["Transformatoren", "sind", "großartig", "</eos>"]
  - **Decoder input (shifted-right targets):**
    - ["<bos>", "Transformatoren", "sind", "großartig"]



Tokenizer





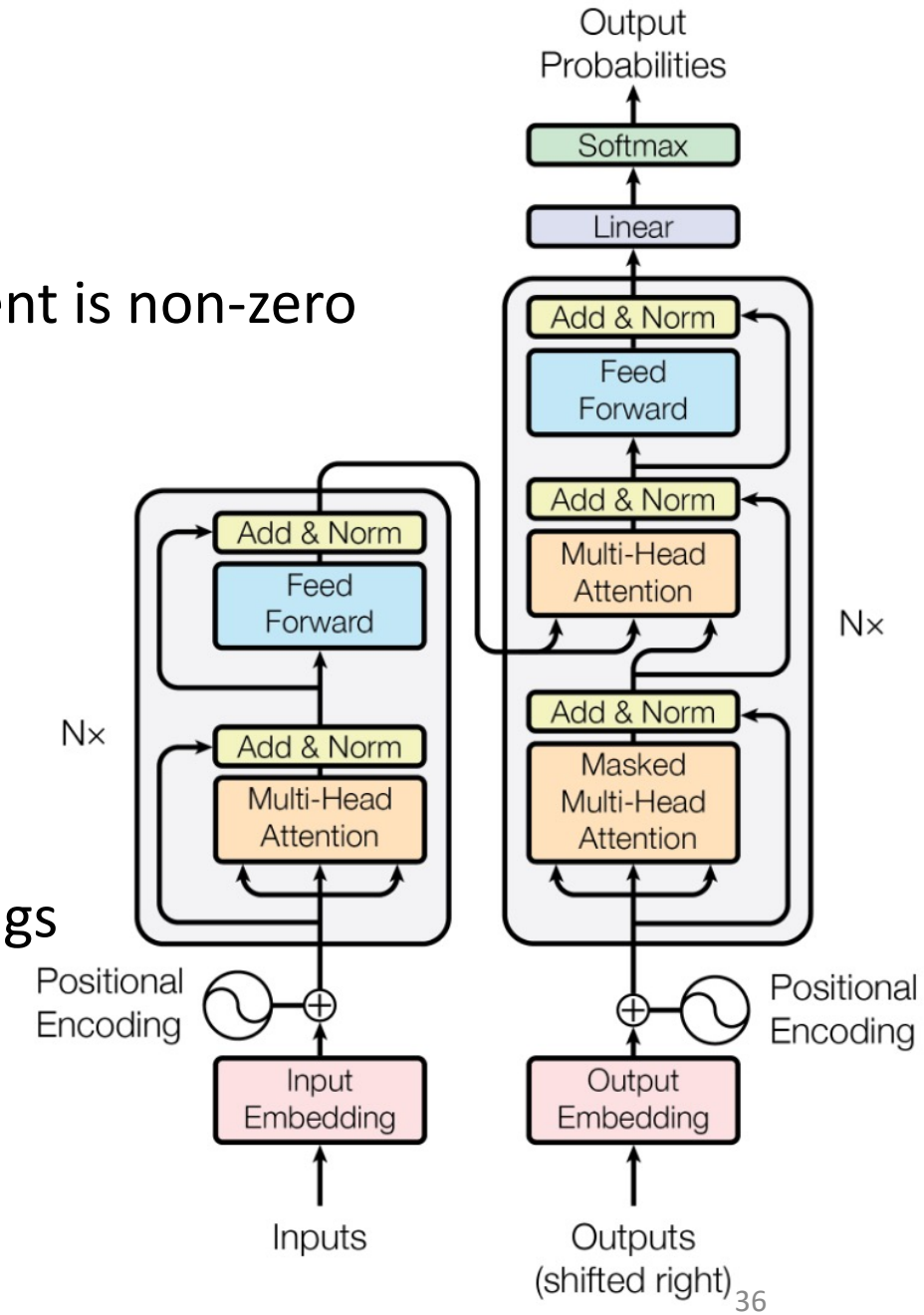
- Natural language
  - $tiktoken$ (for use in OpenAI's models), etc.

https://github.com/openai/tiktoken

Token Representation

- One-hot encoding
  - Represent each token as a vector with only one element is non-zero
    - Index of non-zero element (one) represent the token
  - Size of the vector is same as size of vocabulary
  - Leads to sparse representation

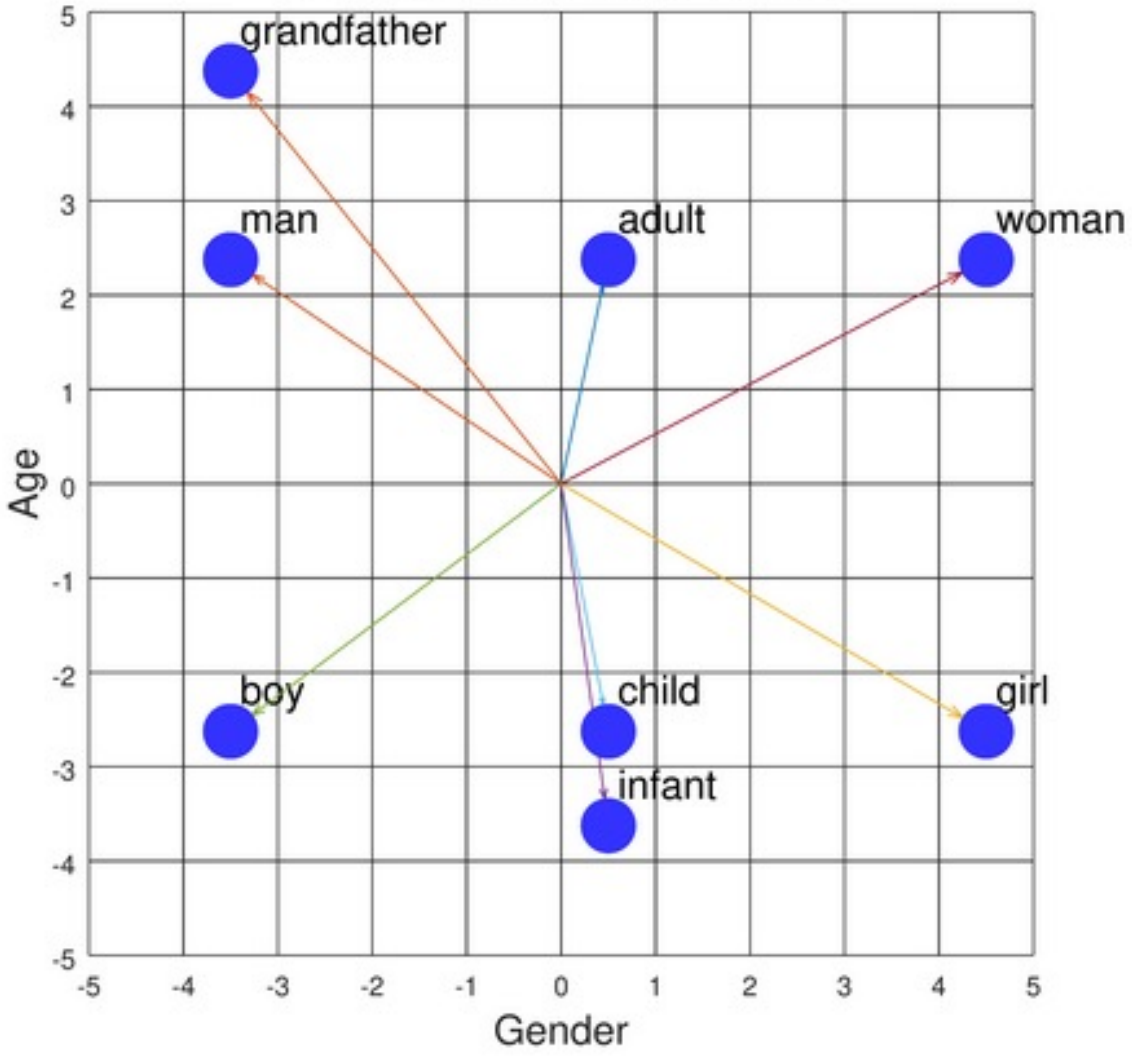
- Token Embedding
  - Maps each token to dense real vector
  - Can be learned or can use pretrained token embeddings
    - Example: GloVe and Word2Vec



https://www.cs.cmu.edu/~dst/WordEmbeddingDemo/tutorial.html
https://www.geeksforgeeks.org/nlp/word-embeddings-in-nlp/

Token Embedding

- Word embeddings capture semantic relations
  - Clustering based on semantics
- Word embeddings are static
  - Lack contextual information
- Self-Attention captures contextual information



https://www.cs.cmu.edu/~dst/WordEmbeddingDemo/tutorial.html

Token Embedding: Example

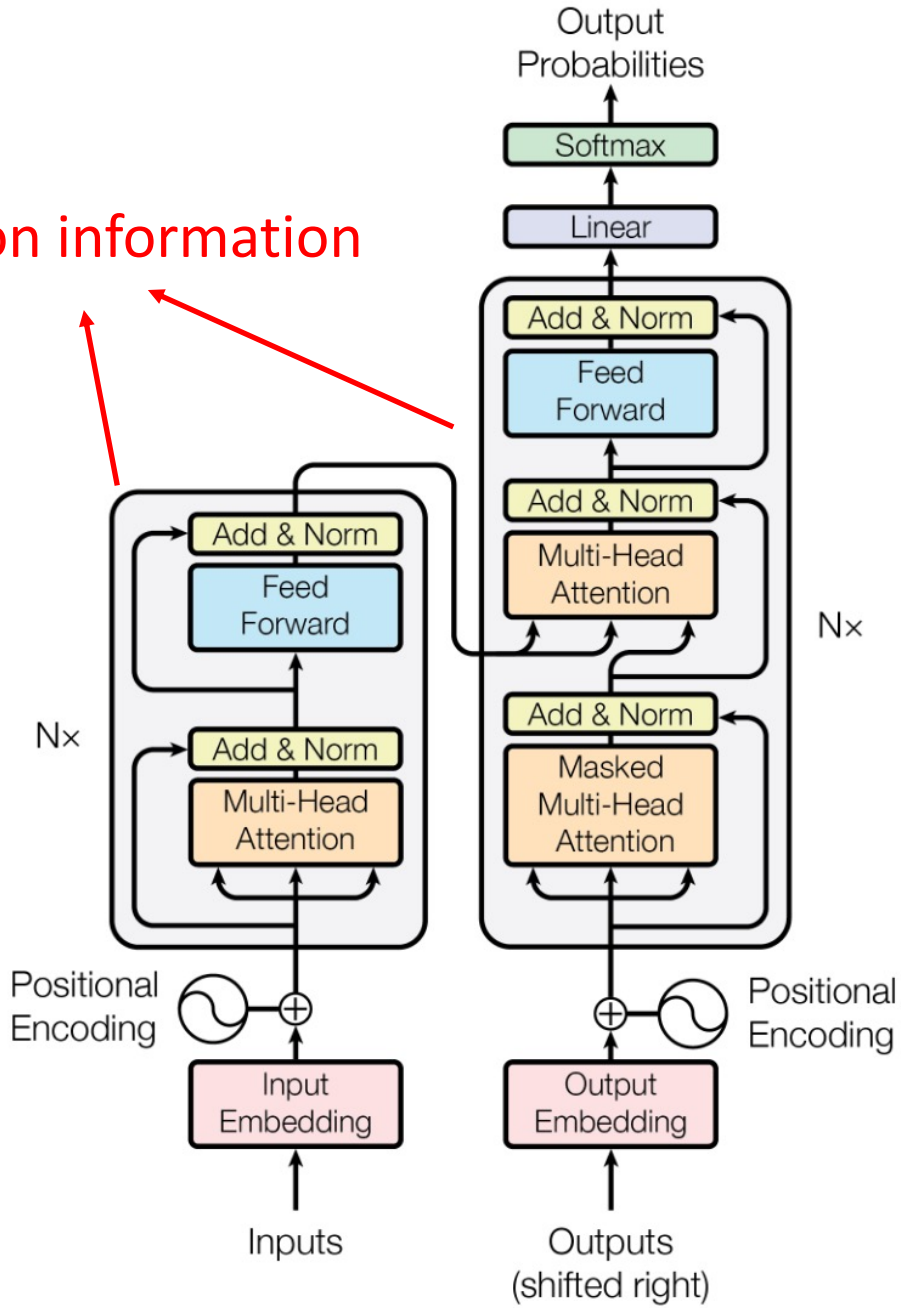
- *"The phone didn't connect to the network because it was down."*
- *"The phone didn't connect to the network because it was broken."*
- Semantically both "it"s are same
  - Token embeddings would be the same for both
- But given the context they refer to different tokens
  - Self-attention layer fixes this issue

Why Positional Encoding?

- $Feed\ Forward$ and $Attention$ layers process set of vectors
  - Set has no inherent order
  - Ignores position information or order of data

- Need explicit injection of positional information

- Adding positional encoding to data helps capture relative positional information of words or tokens

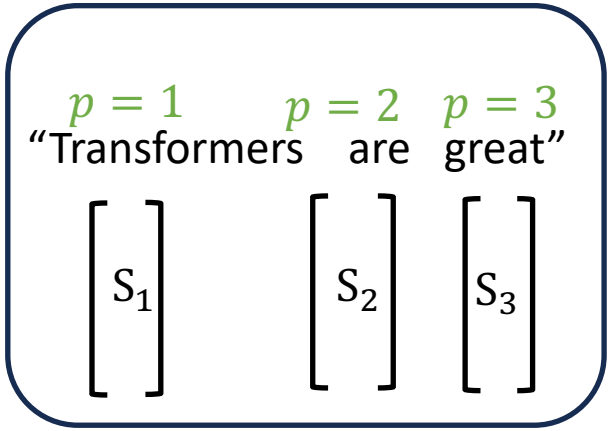


Positional Encoding

- Positional Encoding (PE) vector is unique for a given position in a sequence

$$PE(p, 2i) = \sin(w_i p)$$

$$PE(p, 2i+1) = \cos(w_i p)$$



where

$$w_i = \frac{1}{10000^{\frac{2i}{d}}}$$

p is the position of the embedding vector in the sequence of embedding vectors

i is the position within the embedding vector

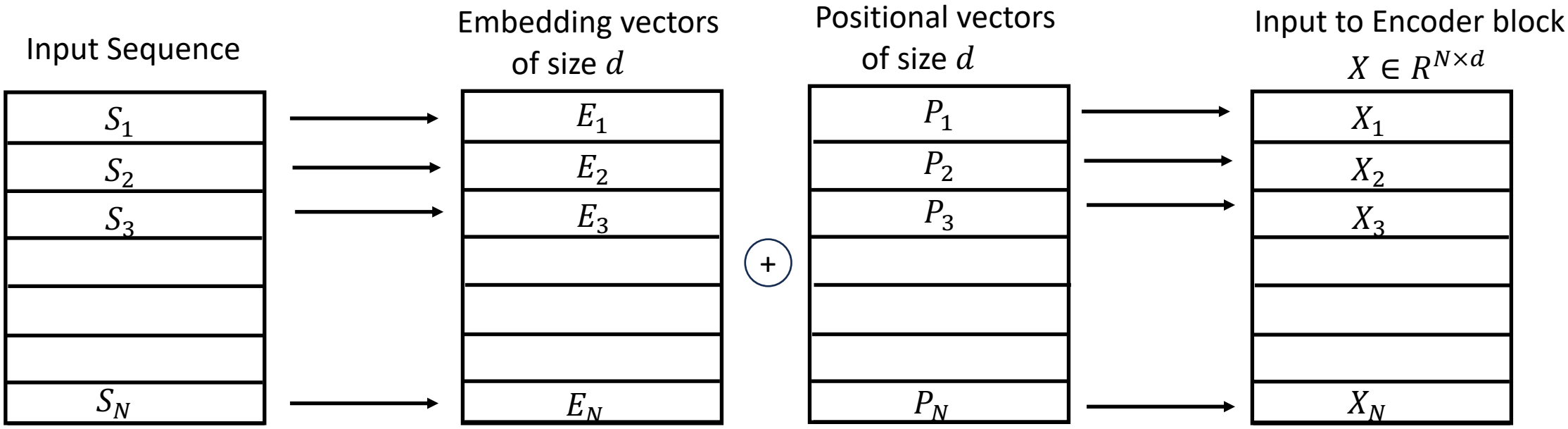
Positional Encoding Vector

$$\mathrm{PE}(p) = \begin{bmatrix} \sin(\omega_1 p) \\ \cos(\omega_1 p) \\ \sin(\omega_2 p) \\ \cos(\omega_2 p) \\ \vdots \\ \sin(\omega_{d/2} p) \\ \cos(\omega_{d/2} p) \end{bmatrix}$$

where $p = pos$, $d = d_{model}$, and $\omega_k = \frac{1}{10000^{2(k-1)/d}}$, for $k = 1, 2, \ldots, d/2$.

Number of learnable parameters in positional embedding?

Positional Encoding



How PE helps?

- Dot product of positional encoding vectors reveals the relative distance between two positions
- Given two positions p_1 and p_2, PE vectors for these two positions:

$$\mathrm{PE}(p_1) = \begin{bmatrix} \sin(\omega_1 p_1) \\ \cos(\omega_1 p_1) \\ \sin(\omega_2 p_1) \\ \cos(\omega_2 p_1) \\ \vdots \\ \sin(\omega_{d/2} p_1) \\ \cos(\omega_{d/2} p_1) \end{bmatrix} \qquad \mathrm{PE}(p_2) = \begin{bmatrix} \sin(\omega_1 p_2) \\ \cos(\omega_1 p_2) \\ \sin(\omega_2 p_2) \\ \cos(\omega_2 p_2) \\ \vdots \\ \sin(\omega_{d/2} p_2) \\ \cos(\omega_{d/2} p_2) \end{bmatrix}$$

- Inner product of these two PE vectors:

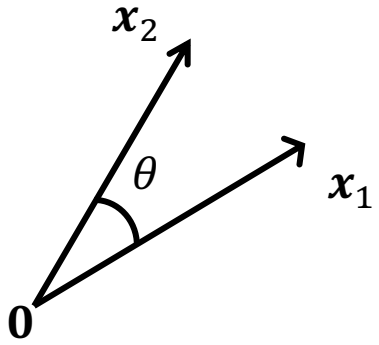
$$\begin{aligned} \mathrm{PE}(p_1) \cdot \mathrm{PE}(p_2) &= \sum_{k=1}^{d/2} \Big(\sin(\omega_k p_1) \sin(\omega_k p_2) + \cos(\omega_k p_1) \cos(\omega_k p_2) \Big) \\ &= \sum_{k=1}^{d/2} \cos\left(\omega_k (p_1 - p_2)\right) \end{aligned}$$

→ function of distance between two vectors

Attention: Intuition

- Measures similarity or relation between two vectors

- Inner or Dot product provides a measure of similarity
  - $\boldsymbol{x}_1 \cdot \boldsymbol{x}_2 = \boldsymbol{x}_1{}^T \boldsymbol{x}_2 = \boldsymbol{x}_2{}^T \boldsymbol{x}_1 = < \boldsymbol{x}_1, \boldsymbol{x}_2 >$
  - $< \boldsymbol{x}_1, \boldsymbol{x}_2 > = \|\boldsymbol{x}_1\|_2 \|\boldsymbol{x}_2\|_2 \cos\theta$



- Inner or dot product between the vectors represent relation between them

What does $Attention$ focus on?

“The book that John gave to Mary was interesting.”

“Le livre que John a donné à Mary était intéressant.”

What does *Attention* focus on?

"The book that John gave to Mary was interesting."
"Le livre que John a donné à Mary était intéressant."

Self-Attention (Encoder)

- Determiner-noun binding
  "The" → "book"
- Subject–verb agreement
  "book" ↔ "was"
- Long-range dependency
  "book" ↔ "that John gave to Mary"
- Entity tracking
  Tracks "John", "Mary" throughout
- Word order mapping
  Helps rearrange for French syntax

Cross-Attention (Decoder)

- Article choice:
  "The" ↔ "Le"
- Noun alignment
  "book" ↔ "livre"
- Subject–verb agreement:
  "book" ↔ "était"
- Verb decomposition
  "gave to" ↔ "a donné à"

Self-attention block at *Decoder* captures similar relations within the sentence in French

Why multiple heads?

- To simultaneously capture multiple types of relationships

"The book that John gave to Mary was interesting."
"Le livre que John a donné à Mary était intéressant."

- **Head 1:** Subject–verb agreement
  Focuses on "book" ↔ "was/était"
- **Head 2:** Long-range dependency
  Links "book" ↔ "that John gave to Mary"
- **Head 3:** Entity tracking
  Tracks "John", "Mary" throughout
- **Head 4:** Local context
  Focuses on "gave to" ↔ "a donné à"
- **Head 5:** Word order mapping
  Helps rearrange for French syntax

Query ($\mathbf{W^Q}$), Key ($\mathbf{W^K}$) and Value ($\mathbf{W^V}$) Parameters

- Model uses W^Q, W^K and W^V parameters to capture these relationships
  - $W^Q \in R^{d \times d}$
  - $W^K \in R^{d \times d}$
  - $W^V \in R^{d \times d}$

Differentiable parameters and learnable
Hence the name *differentiable* dictionary

- Projection of the input embedding onto W^Q converts the input to a query
- Projection of the input embedding onto W^K converts the input to a key
- Projection of the input embedding onto W^V converts the input to a value

How to think of Query (Q), Key (K) and Value (V)?

- **Query**:

  "What do I need right now?"

- **Key**:

  "How should others find me?"

- **Value**:

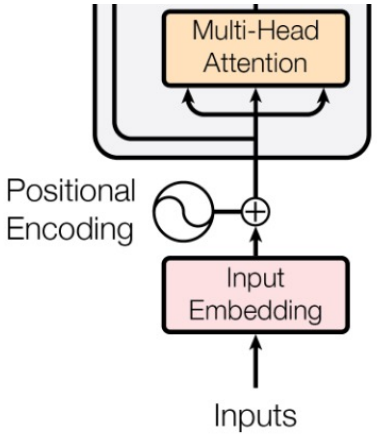
  "What information do I provide if they pick me?"

Example for Self-Attention

"The book that John gave to Mary was interesting."
"Le livre que John a donné à Mary était intéressant."

- Example: Resolving subject-verb
  - **Query** (W^{Q}) while processing the token "*was*" learns to ask "Find the subject?"
  - **Key** (W^{K}) of "book" learns to say "*I am a singular noun and may be a subject*"
  - **Value** (W^{V}) of the "book" provides the semantic content

Multi-Head Attention (MHA)



- $MHA(Q,K,V) = \mathrm{HW_O}$

where $\mathrm{H} = \mathrm{concat}(\mathrm{head_1}, \ldots, \mathrm{head_h})$

$$head_i = softmax\left(\frac{(QW_i^Q)(KW_i^K)^T}{\sqrt{d_K}} \circ \mathrm{M}\right)(VW_i^V)$$

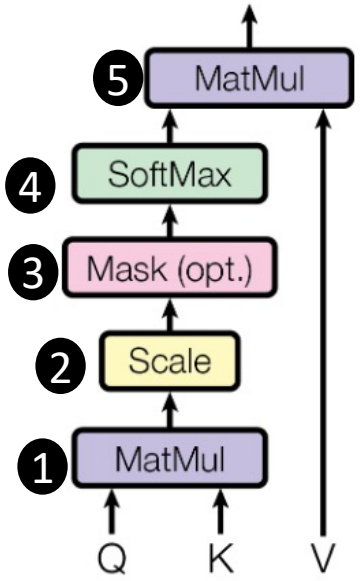
1 MatMul

2 Scale

3 Mask (optional)

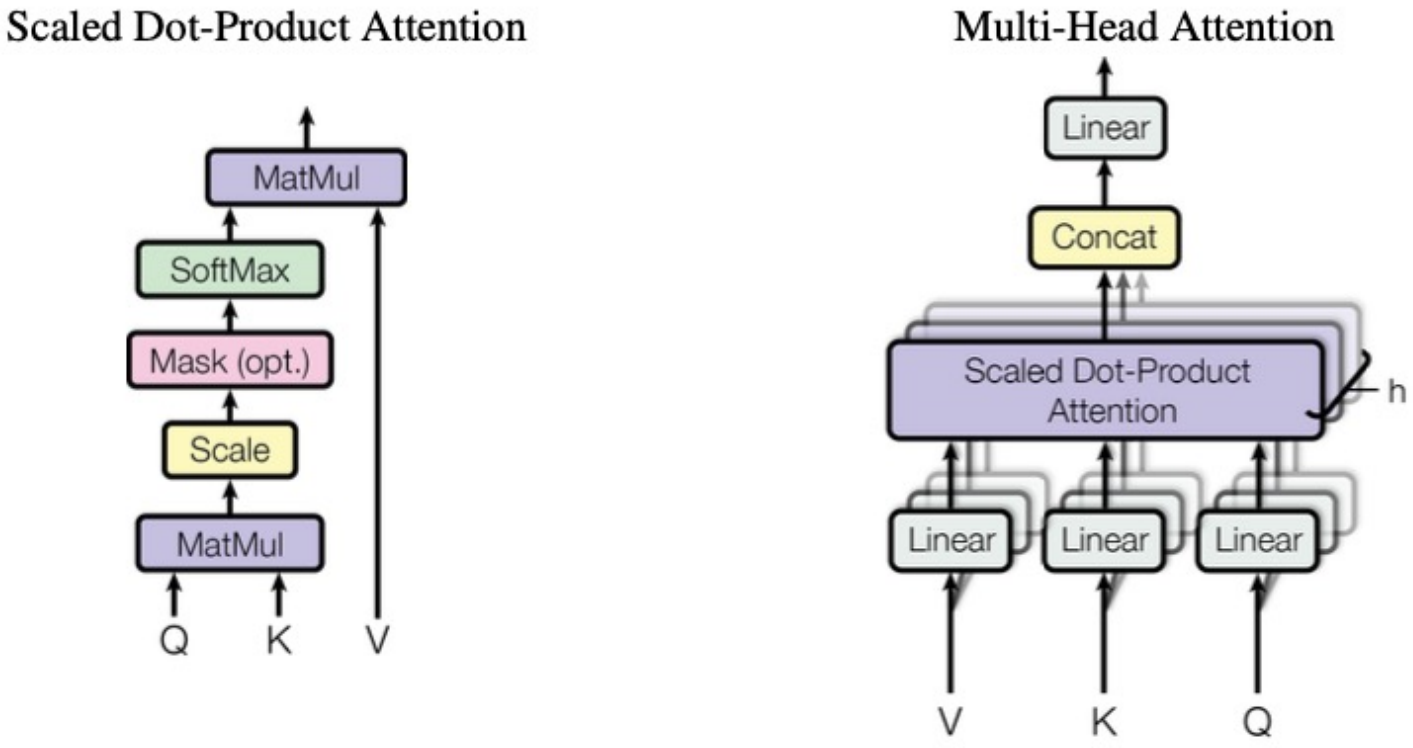
4 softmax

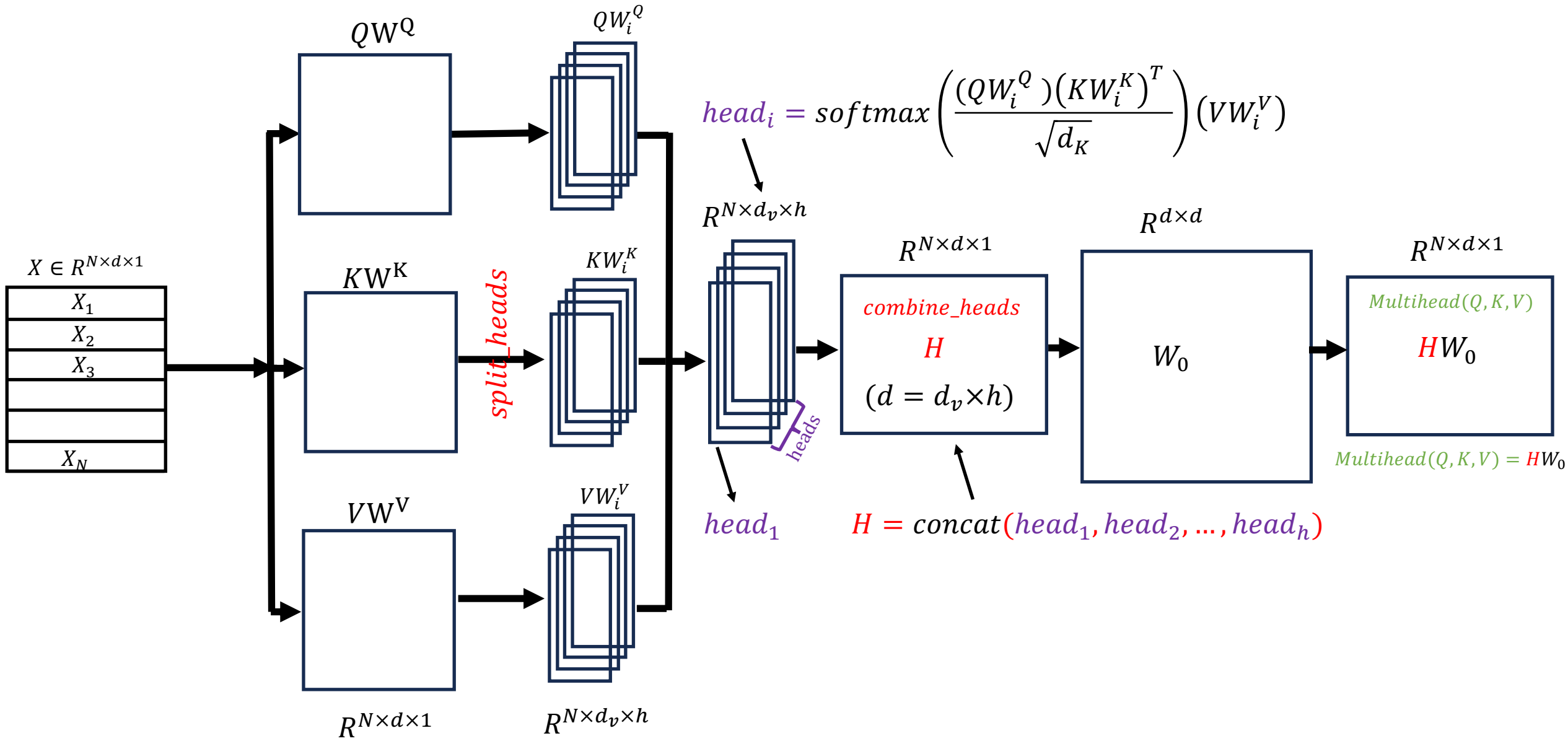
5 MatMul



Ws are learnable parameters

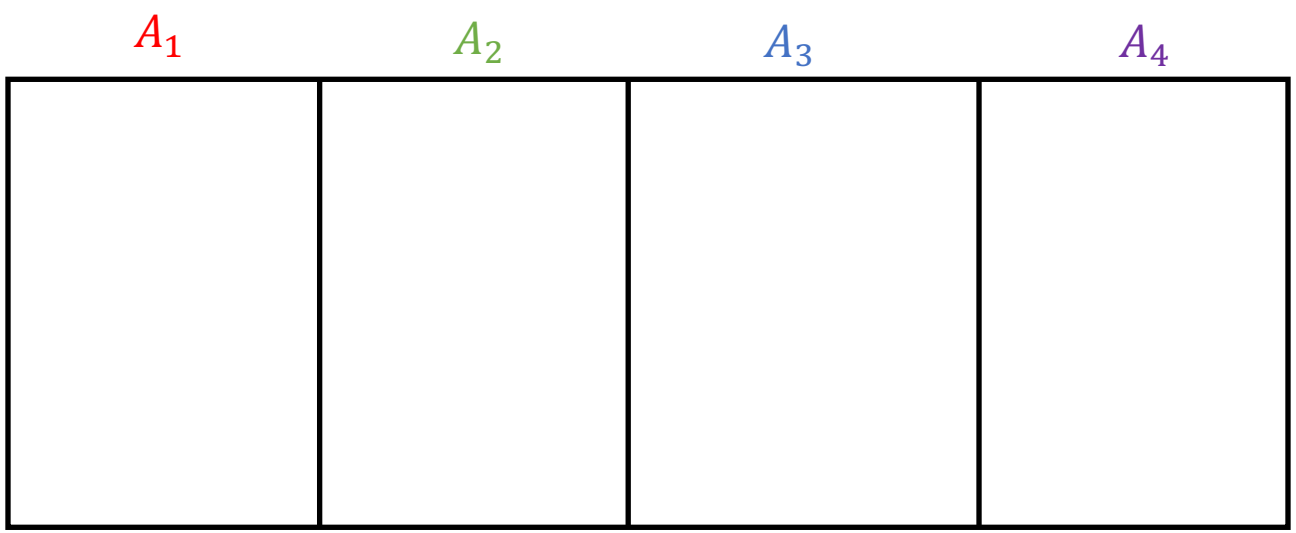
Multi-Head Attention (MHA)





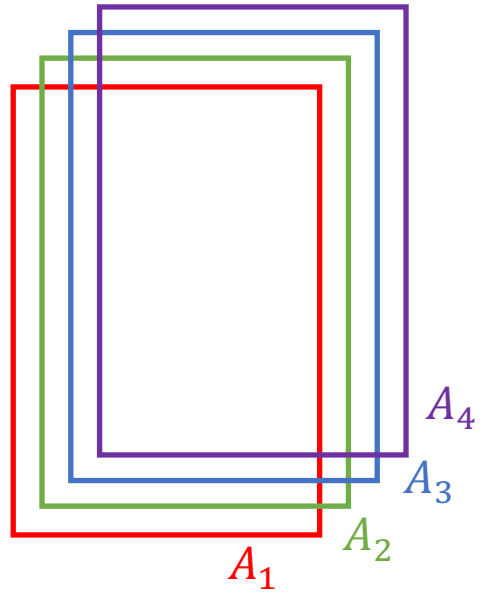
$$d_v = \frac{d}{h}$$

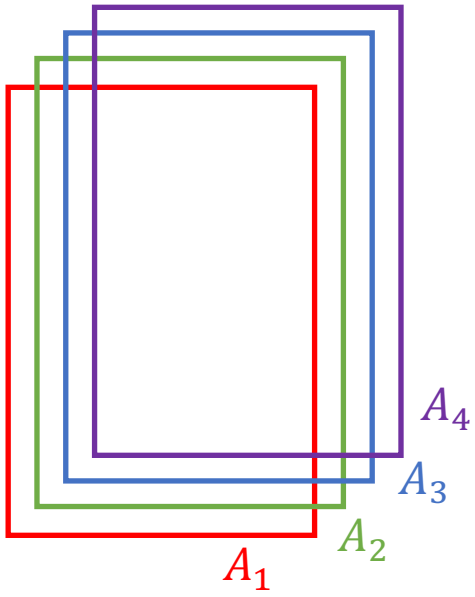
Refer next slide for *split_heads* and *combine_heads*



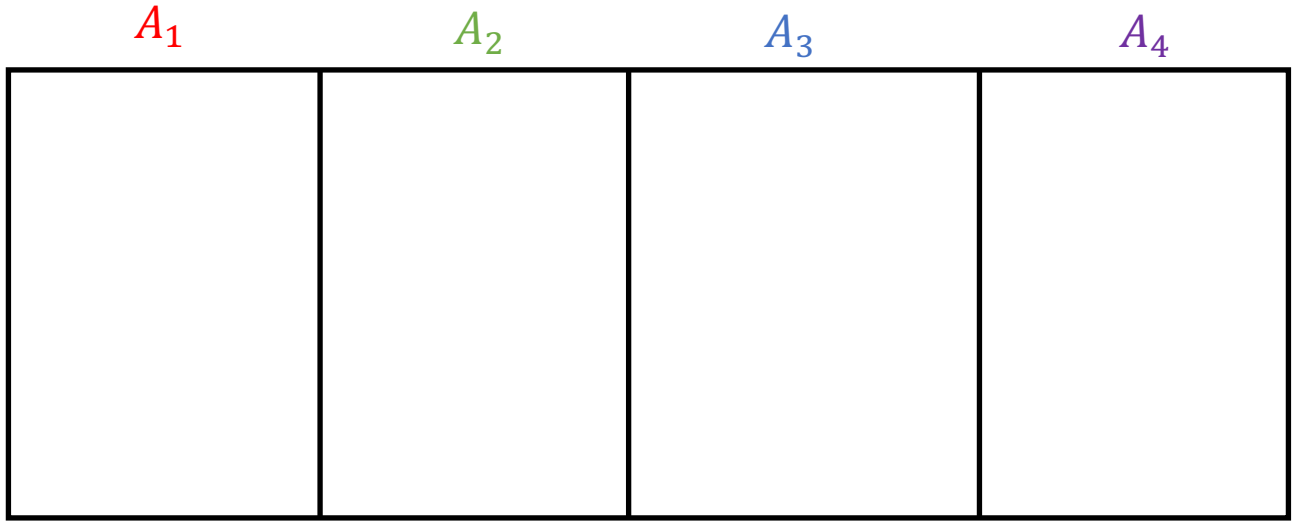
$split\_heads$

- Split along embedding dimension
- Each row corresponds to a token





$combine\_heads$



Scaled Dot-Product Attention: Example with Single Head

$\boldsymbol{w}_m^j \in R^{2\times1}$: Column Vector

$(\boldsymbol{x}_i\ \boldsymbol{w}_m^j) \in R$: Scalar

$\boldsymbol{x}_i \in R^{1\times2}$: Row Vector

$q_i, k_i, v_i \in R^{1\times2}$: Row Vectors (one for each token)

Transformers
are
great

Raw Data

$X \in R^{3\times2\times1}$

$\boldsymbol{x}_1$
$\boldsymbol{x}_2$
$\boldsymbol{x}_3$

SE+PE

$d = 2$

$Q \in R^{3\times2\times1}$

$\boldsymbol{x}_1$
$\boldsymbol{x}_2$
$\boldsymbol{x}_3$

$W^Q \in R^{2\times2}$

$[\boldsymbol{w}_1^q \ \ \boldsymbol{w}_2^q]$

$QW_i^Q = Q \in R^{3\times2\times1}$

$[\boldsymbol{x}_1\boldsymbol{w}_1^q \ \ \boldsymbol{x}_1\boldsymbol{w}_2^q] = \boldsymbol{q}_1$
$[\boldsymbol{x}_2\boldsymbol{w}_1^q \ \ \boldsymbol{x}_2\boldsymbol{w}_2^q] = \boldsymbol{q}_2$
$[\boldsymbol{x}_3\boldsymbol{w}_1^q \ \ \boldsymbol{x}_3\boldsymbol{w}_2^q] = \boldsymbol{q}_3$

$K \in R^{3\times2\times1}$

$\boldsymbol{x}_1$
$\boldsymbol{x}_2$
$\boldsymbol{x}_3$

$W^K \in R^{2\times2}$

$[\boldsymbol{w}_1^k \ \ \boldsymbol{w}_2^k]$

$KW_i^K = K \in R^{3\times2\times1}$

$[\boldsymbol{x}_1\boldsymbol{w}_1^k \ \ \boldsymbol{x}_1\boldsymbol{w}_2^k] = \boldsymbol{k}_1$
$[\boldsymbol{x}_2\boldsymbol{w}_1^k \ \ \boldsymbol{x}_2\boldsymbol{w}_2^k] = \boldsymbol{k}_2$
$[\boldsymbol{x}_3\boldsymbol{w}_1^k \ \ \boldsymbol{x}_3\boldsymbol{w}_2^k] = \boldsymbol{k}_3$

$V \in R^{3\times2\times1}$

$\boldsymbol{x}_1$
$\boldsymbol{x}_2$
$\boldsymbol{x}_3$

$W^V \in R^{2\times2}$

$[\boldsymbol{w}_1^v \ \ \boldsymbol{w}_2^v]$

$VW_i^V = V \in R^{3\times2\times1}$

$[\boldsymbol{x}_1\boldsymbol{w}_1^v \ \ \boldsymbol{x}_1\boldsymbol{w}_2^v] = \boldsymbol{v}_1$
$[\boldsymbol{x}_2\boldsymbol{w}_1^v \ \ \boldsymbol{x}_2\boldsymbol{w}_2^v] = \boldsymbol{v}_2$
$[\boldsymbol{x}_3\boldsymbol{w}_1^v \ \ \boldsymbol{x}_3\boldsymbol{w}_2^v] = \boldsymbol{v}_3$

$$head_i = softmax\left(\frac{(QW_i^Q)(KW_i^K)^T}{\sqrt{d_K}}\right)(VW_i^V)$$

$$\frac{(QW_i^Q)(KW_i^K)^T}{\sqrt{d_k}} = \begin{bmatrix} \frac{\boldsymbol{q}_1\boldsymbol{k}_1^T}{\sqrt{d_1}} & \frac{\boldsymbol{q}_1\boldsymbol{k}_2^T}{\sqrt{d_1}} & \frac{\boldsymbol{q}_1\boldsymbol{k}_3^T}{\sqrt{d_1}} \\ \frac{\boldsymbol{q}_2\boldsymbol{k}_1^T}{\sqrt{d_2}} & \frac{\boldsymbol{q}_2\boldsymbol{k}_2^T}{\sqrt{d_2}} & \frac{\boldsymbol{q}_2\boldsymbol{k}_3^T}{\sqrt{d_2}} \\ \frac{\boldsymbol{q}_3\boldsymbol{k}_1^T}{\sqrt{d_3}} & \frac{\boldsymbol{q}_3\boldsymbol{k}_2^T}{\sqrt{d_3}} & \frac{\boldsymbol{q}_3\boldsymbol{k}_3^T}{\sqrt{d_3}} \end{bmatrix}$$

Attention Scores

	Transformers	are	great
Transformers	a_{11}	a_{12}	a_{13}
are	a_{21}	a_{22}	a_{23}
great	a_{31}	a_{32}	a_{33}

$$softmax\left(\frac{(QW_i^Q)(KW_i^K)^T}{\sqrt{d_k}}\right) = \begin{bmatrix} softmax\left(\begin{bmatrix} \frac{\boldsymbol{q}_1\boldsymbol{k}_1^T}{\sqrt{d_1}} & \frac{\boldsymbol{q}_1\boldsymbol{k}_2^T}{\sqrt{d_1}} & \frac{\boldsymbol{q}_1\boldsymbol{k}_3^T}{\sqrt{d_1}} \end{bmatrix}\right) \\ softmax\left(\begin{bmatrix} \frac{\boldsymbol{q}_2\boldsymbol{k}_1^T}{\sqrt{d_2}} & \frac{\boldsymbol{q}_2\boldsymbol{k}_2^T}{\sqrt{d_2}} & \frac{\boldsymbol{q}_2\boldsymbol{k}_3^T}{\sqrt{d_2}} \end{bmatrix}\right) \\ softmax\left(\begin{bmatrix} \frac{\boldsymbol{q}_3\boldsymbol{k}_1^T}{\sqrt{d_3}} & \frac{\boldsymbol{q}_3\boldsymbol{k}_2^T}{\sqrt{d_3}} & \frac{\boldsymbol{q}_3\boldsymbol{k}_3^T}{\sqrt{d_3}} \end{bmatrix}\right) \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

Visualization of
Attention Scores
From “Attention is all you need”

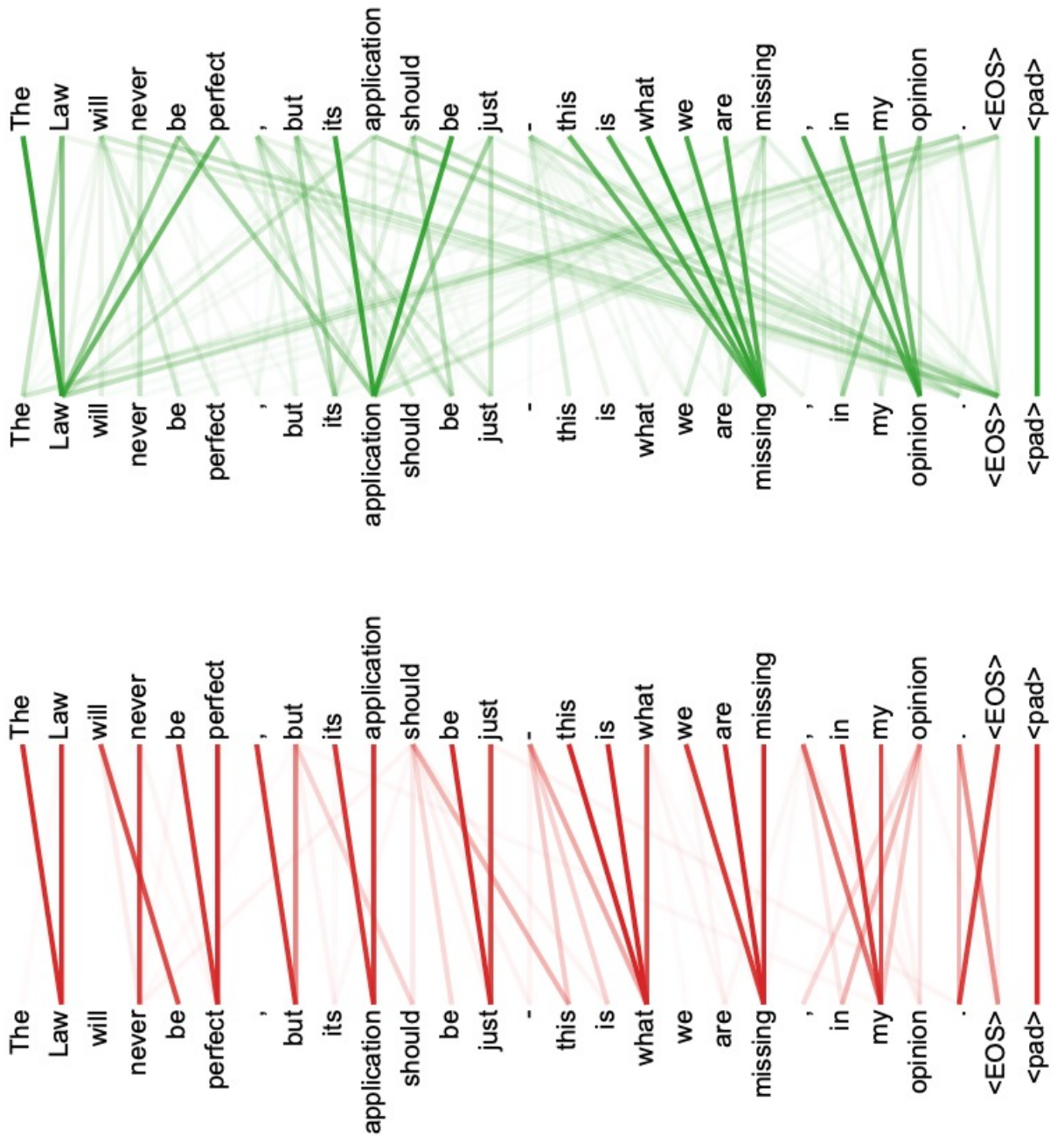


Figure 5: Many of the attention heads exhibit behaviour that seems related to the structure of the sentence. We give two such examples above, from two different heads from the encoder self-attention at layer 5 of 6. The heads clearly learned to perform different tasks.

Softmax - Recap

$$\boldsymbol{z} = [z_1, z_2, z_3, \dots, z_n]$$

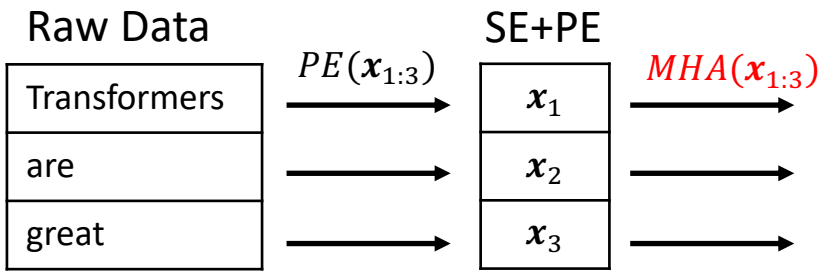
$$softmax(\boldsymbol{z}) = \left[\frac{\exp(z_1)}{\sum_i \exp(z_i)}, \frac{\exp(z_2)}{\sum_i \exp(z_i)}, \frac{\exp(z_3)}{\sum_i \exp(z_i)}, \dots, \frac{\exp(z_n)}{\sum_i \exp(z_i)}\right]$$

$$head_i = softmax\left(\frac{(QW_i^Q)(KW_i^K)^T}{\sqrt{d_K}}\right)(VW_i^V)$$

Attention Scores

$$softmax\left(\frac{(QW_i^Q)(KW_i^K)^T}{\sqrt{d_k}}\right)\begin{bmatrix} \boldsymbol{x}_1\boldsymbol{w}_1^v & \boldsymbol{x}_1\boldsymbol{w}_2^v \\ \boldsymbol{x}_2\boldsymbol{w}_1^v & \boldsymbol{x}_2\boldsymbol{w}_2^v \\ \boldsymbol{x}_3\boldsymbol{w}_1^v & \boldsymbol{x}_3\boldsymbol{w}_2^v \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}\begin{bmatrix} \boldsymbol{x}_1\boldsymbol{w}_1^v & \boldsymbol{x}_1\boldsymbol{w}_2^v \\ \boldsymbol{x}_2\boldsymbol{w}_1^v & \boldsymbol{x}_2\boldsymbol{w}_2^v \\ \boldsymbol{x}_3\boldsymbol{w}_1^v & \boldsymbol{x}_3\boldsymbol{w}_2^v \end{bmatrix}$$

Each row vector includes **semantic**, **positional** and **contextual** information of a word

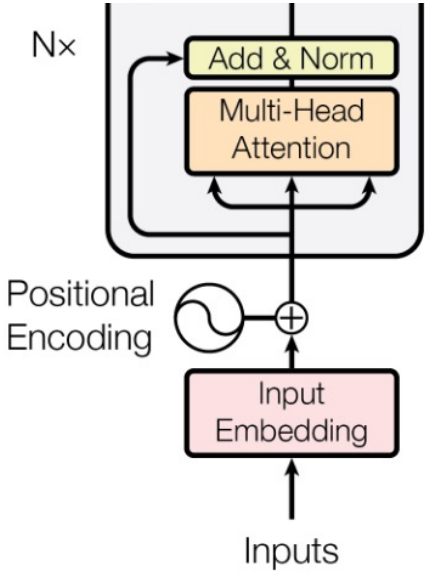


$$\begin{bmatrix} a_{11}(\boldsymbol{x}_1\boldsymbol{w}_1^v) + a_{12}(\boldsymbol{x}_2\boldsymbol{w}_1^v) + a_{13}(\boldsymbol{x}_3\boldsymbol{w}_1^v) & a_{11}(\boldsymbol{x}_1\boldsymbol{w}_2^v) + a_{12}(\boldsymbol{x}_2\boldsymbol{w}_2^v) + a_{13}(\boldsymbol{x}_3\boldsymbol{w}_3^v) \\ a_{21}(\boldsymbol{x}_1\boldsymbol{w}_1^v) + a_{22}(\boldsymbol{x}_2\boldsymbol{w}_1^v) + a_{23}(\boldsymbol{x}_3\boldsymbol{w}_1^v) & a_{21}(\boldsymbol{x}_1\boldsymbol{w}_2^v) + a_{22}(\boldsymbol{x}_2\boldsymbol{w}_2^v) + a_{23}(\boldsymbol{x}_3\boldsymbol{w}_2^v) \\ a_{31}(\boldsymbol{x}_1\boldsymbol{w}_1^v) + a_{32}(\boldsymbol{x}_2\boldsymbol{w}_1^v) + a_{33}(\boldsymbol{x}_3\boldsymbol{w}_1^v) & a_{31}(\boldsymbol{x}_1\boldsymbol{w}_2^v) + a_{32}(\boldsymbol{x}_2\boldsymbol{w}_2^v) + a_{33}(\boldsymbol{x}_3\boldsymbol{w}_2^v) \end{bmatrix}$$

Embedding of “great” is a linear combination of embeddings of “Transformers”, “are”, and “great”

$$a_{ij} = \frac{\exp(q_i k_j^T)}{\sum_m \exp(q_i k_m^T)}$$

Add and Norm after MHA

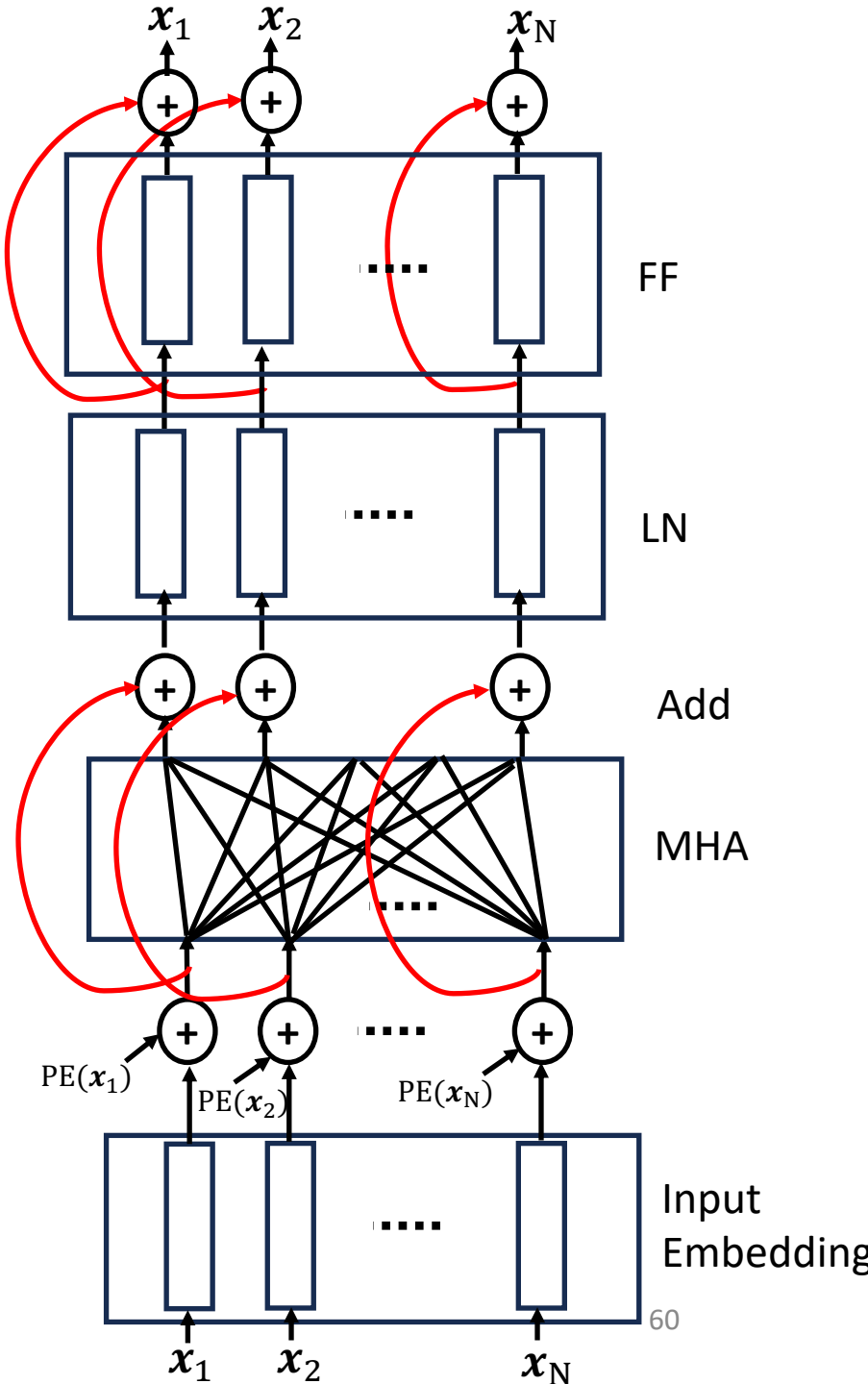


Layer Normalization

Residual connection

$$Output = LN(MHA(\boldsymbol{x}_{1:N}) + \boldsymbol{x}_{1:N})$$

LN is applied for each embedding vector independently (refer next slide)



Layer Normalization (LN)

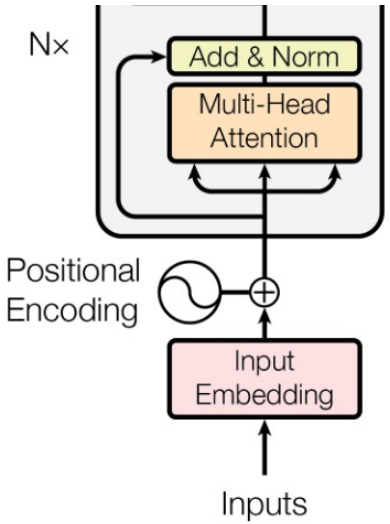
- $LN(\boldsymbol{x}_\mathrm{i}) = \boldsymbol{\gamma} \circ \left(\frac{\boldsymbol{x}_\mathrm{i} - \mu_\mathrm{i}}{\sigma_\mathrm{i}}\right) + \boldsymbol{\beta}$

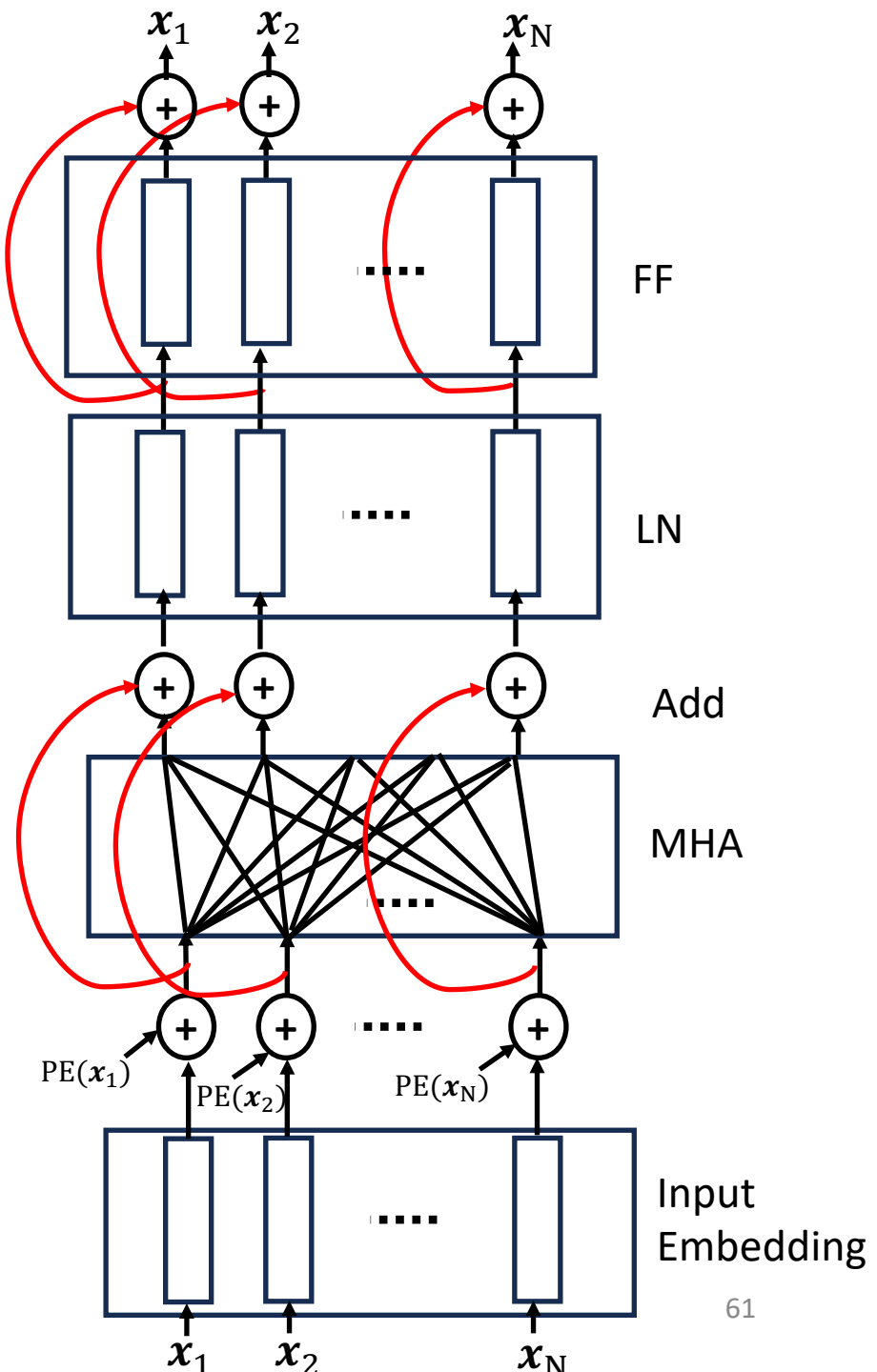
where

$$\mu_\mathrm{i} = mean(\boldsymbol{x}_\mathrm{i})$$

$$\sigma_\mathrm{i} = \mathrm{std}(\boldsymbol{x}_\mathrm{i})$$

$$\boldsymbol{x}_\mathrm{i}, \boldsymbol{\gamma}, \boldsymbol{\beta} \in \mathrm{R}^\mathrm{d}$$





$\boldsymbol{\gamma}$ and $\boldsymbol{\beta}$ are learnable parameters shared across the tokens

Add and Norm after MHA

Post-Layer Norm:

Layer norm is applied after MHA block

Layer Normalization

$$\text{Output} = LN(MHA(\boldsymbol{x}_{1:N}) + \boldsymbol{x}_{1:N})$$

Residual connection

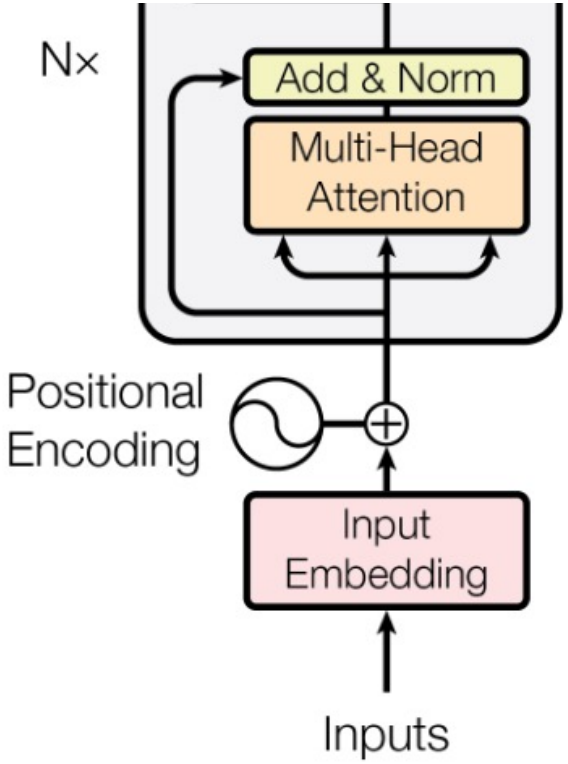
Pre-Layer Norm:

Layer norm is applied before MHA block

Layer Normalization

$$\text{Output} = MHA(LN(\boldsymbol{x}_{1:N})) + \boldsymbol{x}_{1:N}$$

Residual connection

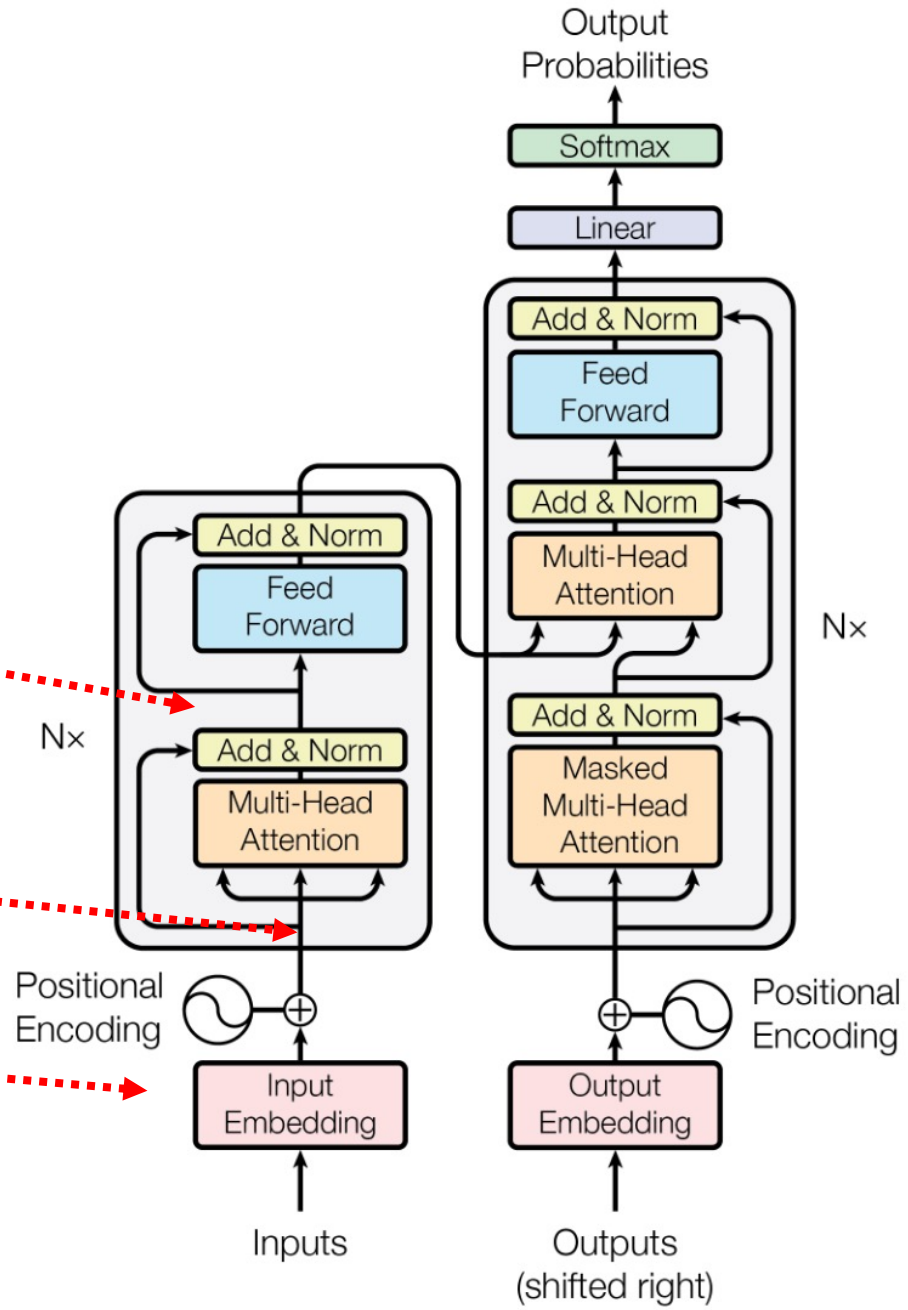


Semantic Embedding to Contextual Embedding

Embeddings with Semantic, Positional and Contextual Information

Embeddings with Semantic and Positional Information

Embeddings with Semantic Information



Positionwise Feedforward Layer

- A two-layer MLP with ReLU activation, skip connection and layer normalization

Layer Normalization

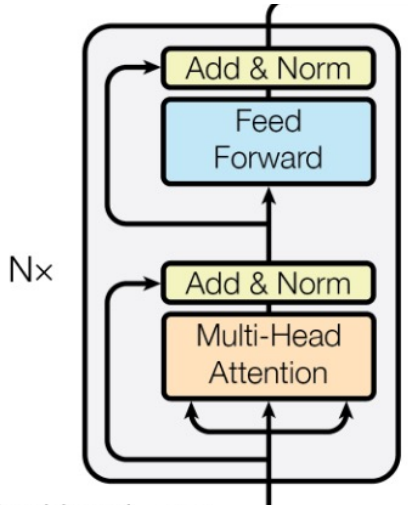
Linear function

ReLU activation

Residual Connection

$$MLP(\boldsymbol{x}_i) = LN\left(\boldsymbol{f}^{[2]}\left(\boldsymbol{g}^{[1]}\left(\boldsymbol{f}^{[1]}(\boldsymbol{x}_i)\right)\right) + \boldsymbol{x}_i\right)$$

$$\boldsymbol{f}^{[j]}(\boldsymbol{x}_i) = \left(\mathrm{W}^{[j]}\right)^{\mathrm{T}}\boldsymbol{x}_i + \boldsymbol{b}^{[j]}$$



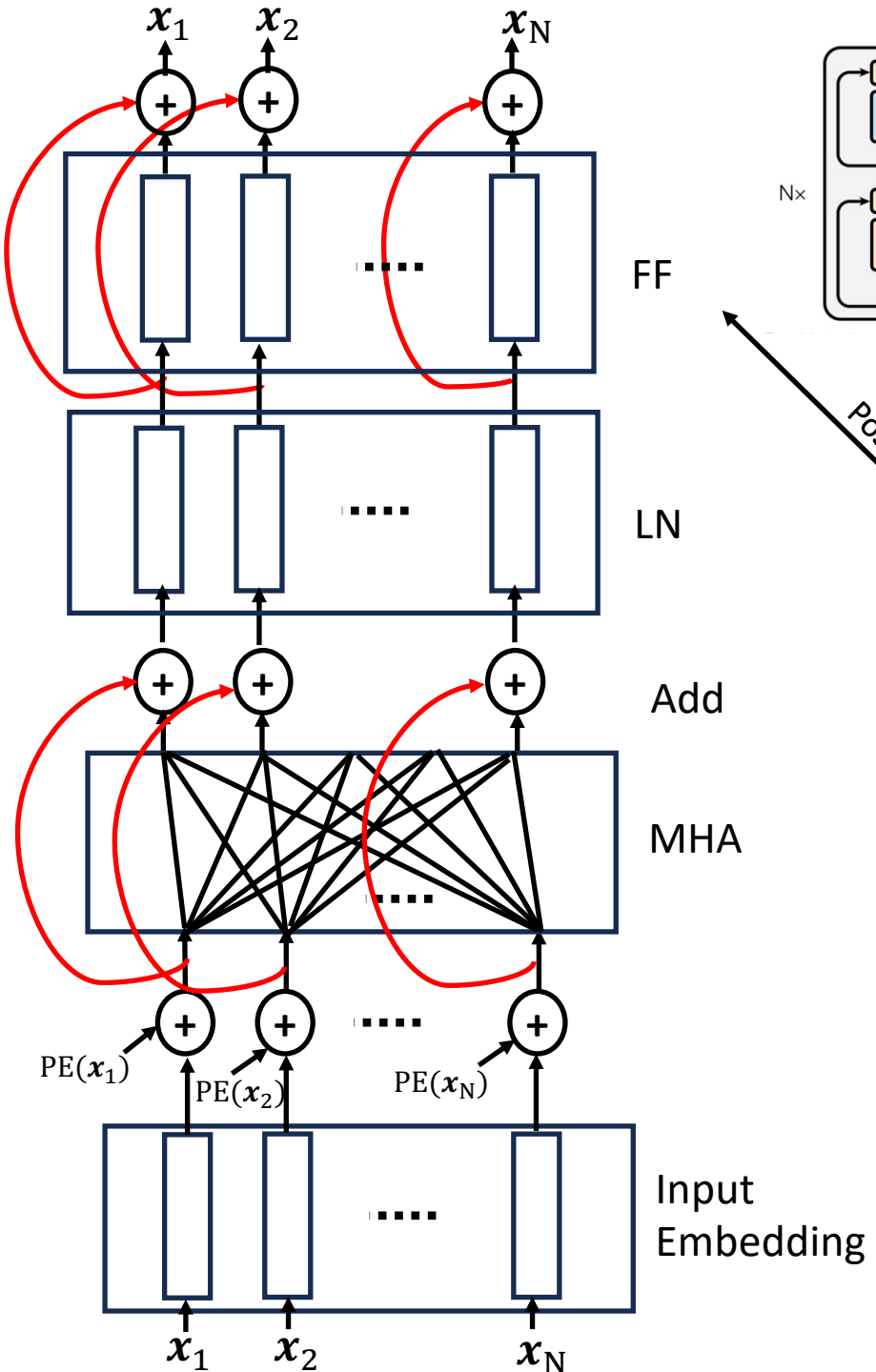
- Each embedding vector is transformed independently
- Example with ReLU activation:

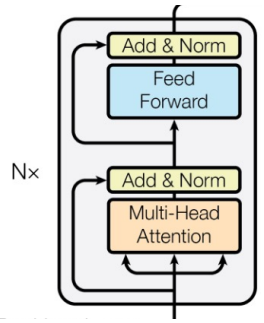
$$MLP(\boldsymbol{x}_i) = LN\left(\left(\mathrm{W}^{[2]}\right)^{\mathrm{T}} \max\left(0, \left(\mathrm{W}^{[1]}\right)^{\mathrm{T}}\boldsymbol{x}_i + \boldsymbol{b_1}\right) + \boldsymbol{b_2} + \boldsymbol{x}_i\right)$$

$\mathrm{W}^{[1]} \in \mathrm{R}^{\mathrm{d}\times\mathrm{d_f}}$

$\mathrm{W}^{[2]} \in \mathrm{R}^{\mathrm{d_f}\times\mathrm{d}}$

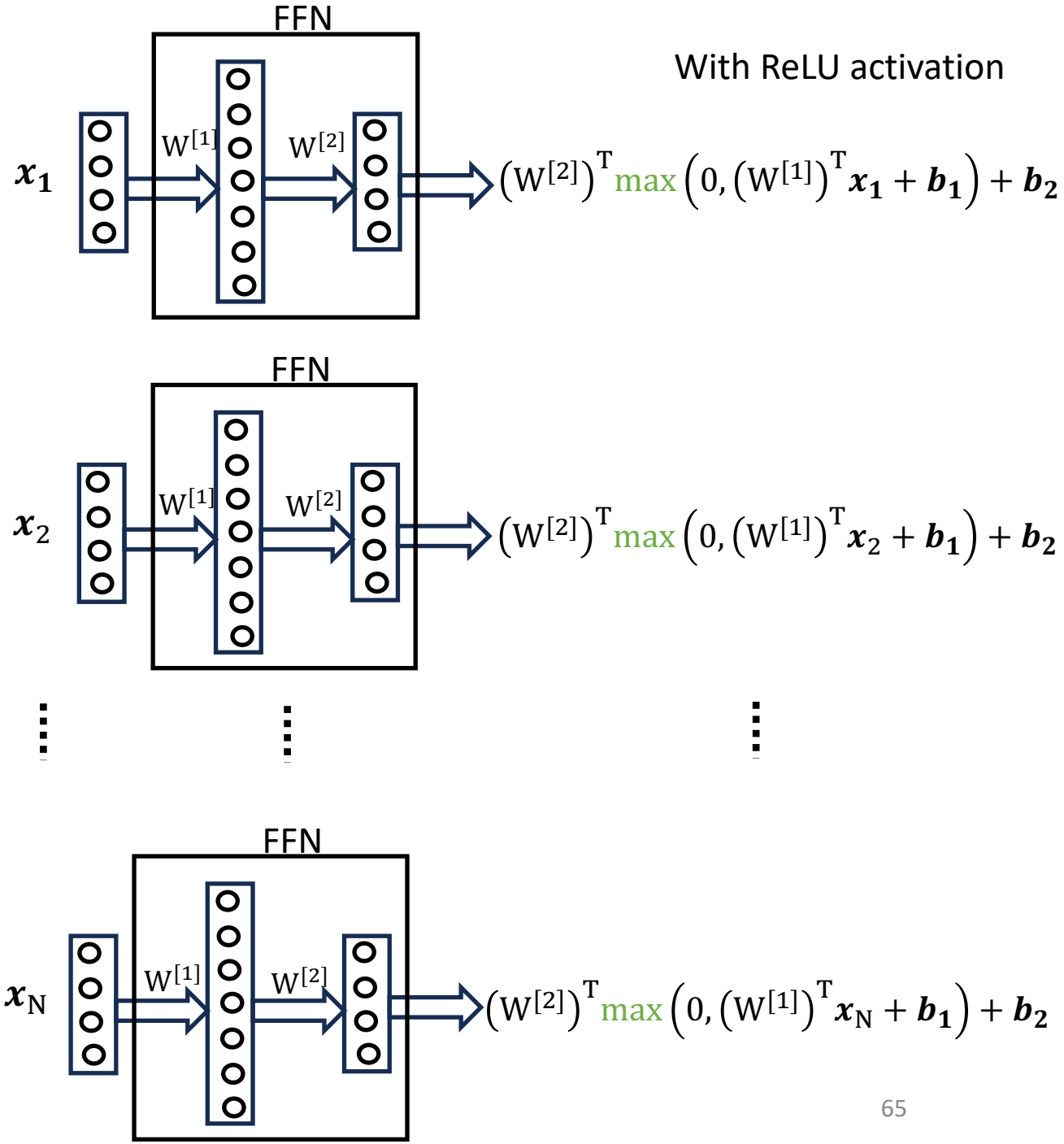
$\mathrm{W}^{[j]}$ are shared across the tokens





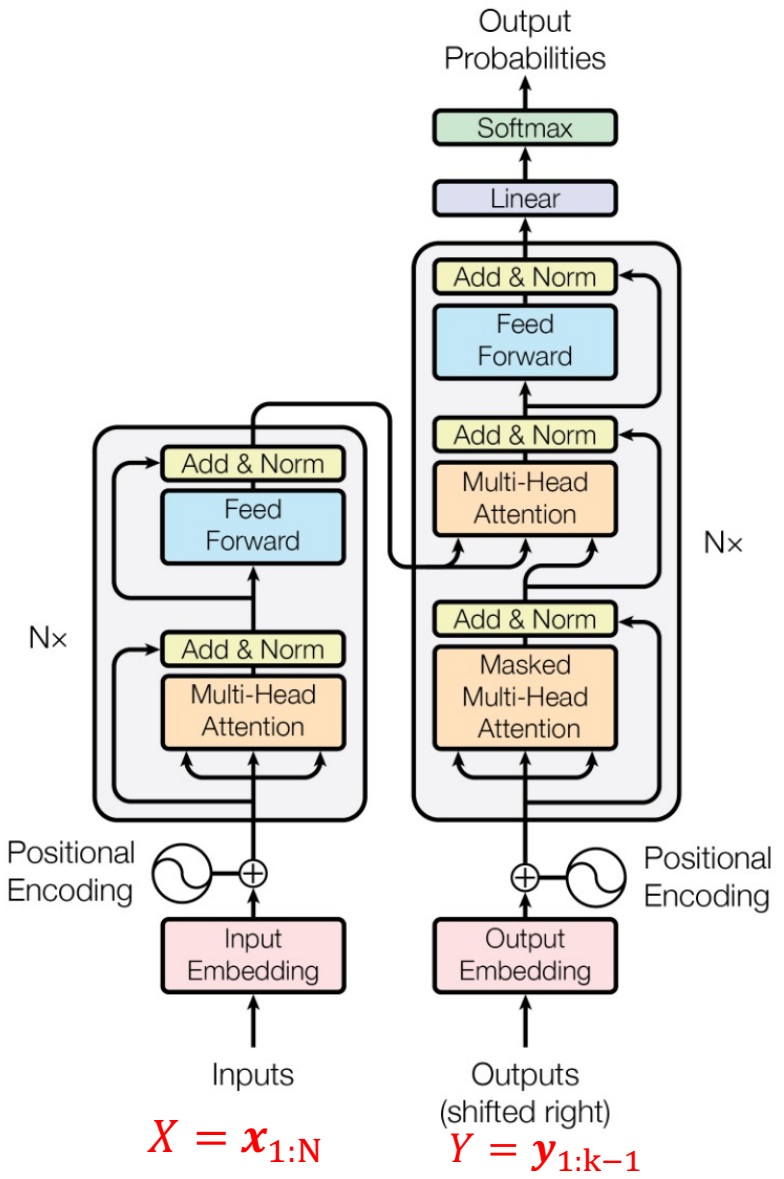
Positionwise FF Layer

With ReLU activation



Why Masked Multi-head Attention at Decoder?

$P(y_k|\boldsymbol{x}_{1:\mathrm{N}}, \boldsymbol{y}_{1:k-1})$



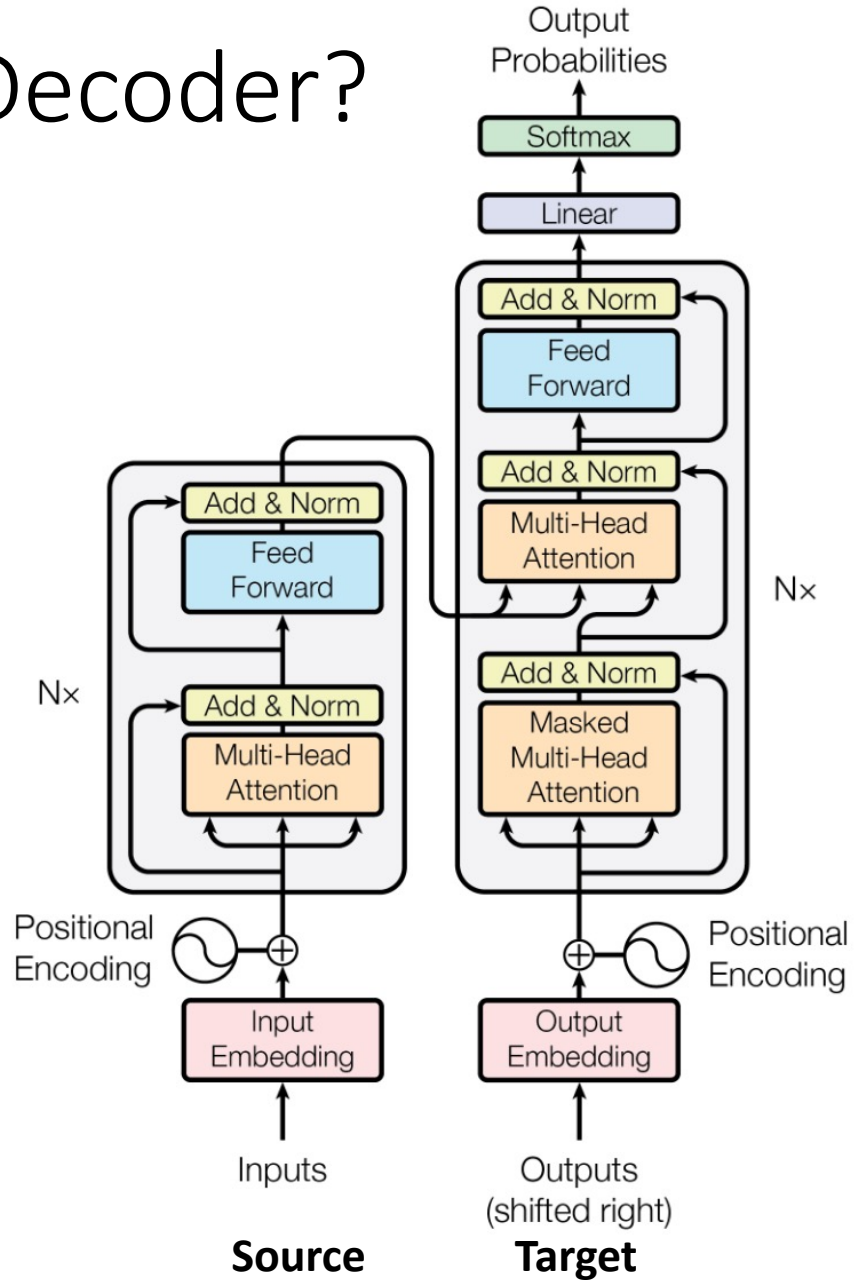
$X = \boldsymbol{x}_{1:\mathrm{N}}$

$Y = \boldsymbol{y}_{1:\mathrm{k}-1}$

Why Masked Multi-head Attention at Decoder?

- During Training:
  - **Source (English):**
    - ["Transformers", "are", "great"]
  - **Target (German):**
    - ["Transformatoren", "sind", "großartig", "</eos>"]
  - **Decoder input (shifted-right targets):**
    - ["<bos>", "Transformatoren", "sind", "großartig"]

- Encoder is bi-directional (no masking)
  - Each word looks at the past and future words to build context
  - The encoder processes the entire source sequence at once

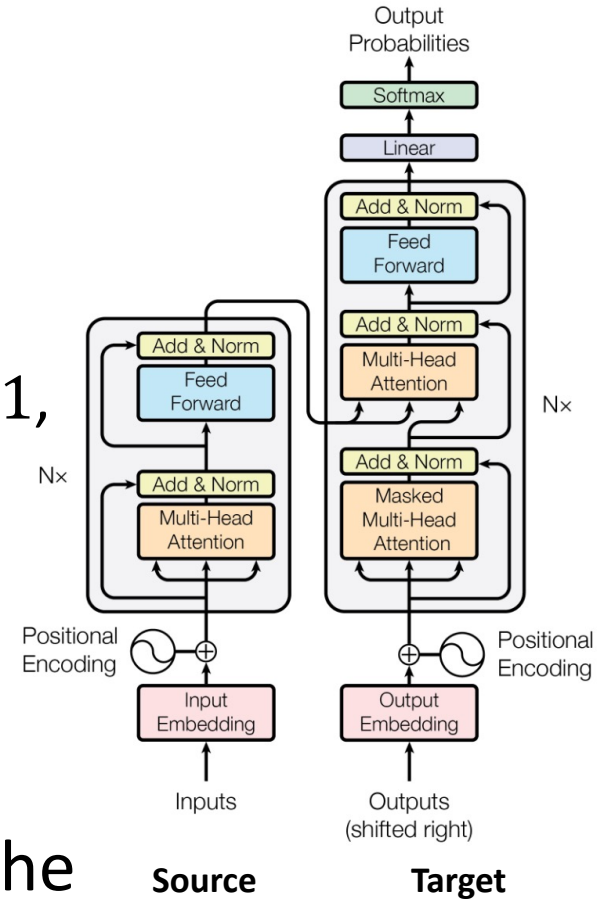


Why Masked Multi-head Attention at Decoder?

- Decoder generates tokens sequentially (one per step)

- What the decoder sees and predicts?
  - At each target position t, the decoder input is everything up to $t-1$,
  - Decoder predicts the token at t.
    - $t = 1$: input "<bos>" → predict "Transformatoren"
    - $t = 2$: input "<bos> Transformatoren" → predict "sind"
    - $t = 3$: input "<bos> Transformatoren sind" → predict "großartig"

- We don't want decoder to look at the future words to build the context



*notes

Masked Multi-head Attention

- $head_i = softmax\left(\frac{(QW_i^Q)(KW_i^K)^T}{\sqrt{d_K}} \circ \mathrm{M}\right)(VW_i^V)$

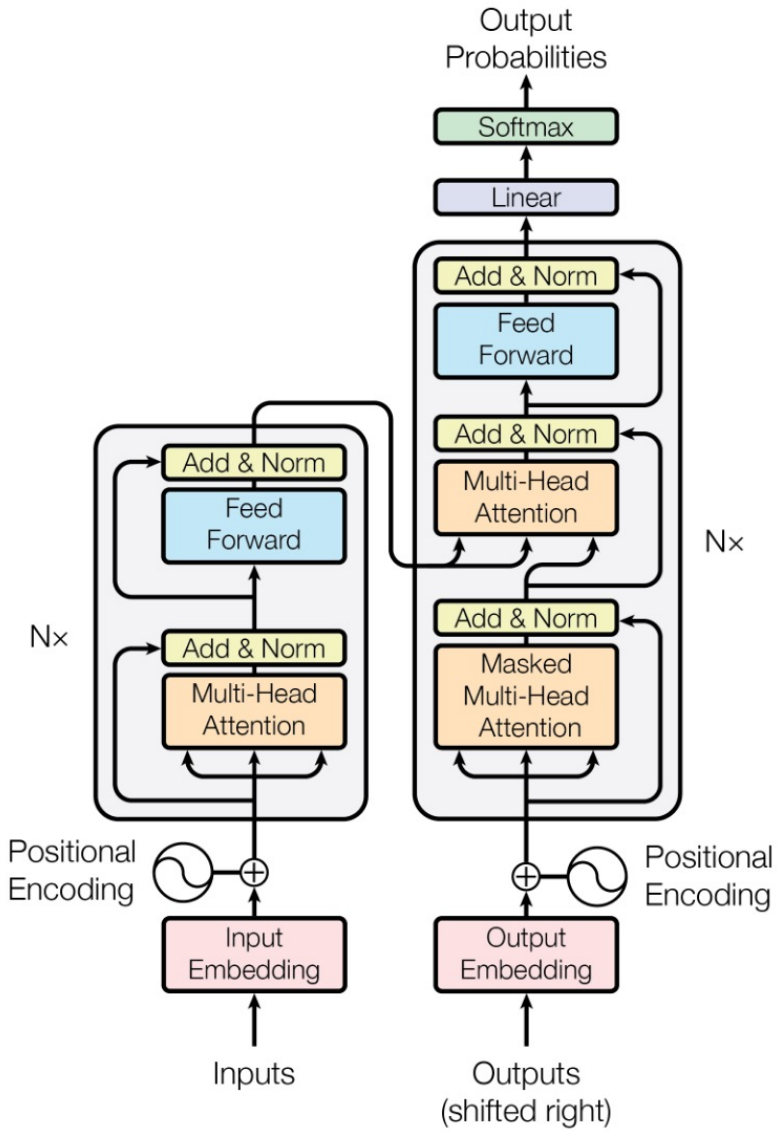
$$\text{where } \mathrm{M_{ij}} = \begin{cases} 1, & \text{if attention between i and j} \\ -\infty, & \text{if no attention} \end{cases}$$

  - Example Mask (M):

$$\mathrm{M} = \begin{bmatrix} 1 & -\infty & -\infty & -\infty & -\infty \\ 1 & 1 & -\infty & -\infty & -\infty \\ 1 & 1 & 1 & -\infty & -\infty \\ 1 & 1 & 1 & 1 & -\infty \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

- Masking ensures causal property is not violated
  - Don't predict current output based on future outputs

$P(y_k|\boldsymbol{x}_{1:\mathrm{N}}, \boldsymbol{y}_{1:k-1})$



$X = \boldsymbol{x}_{1:\mathrm{N}}$ $Y = \boldsymbol{y}_{1:\mathrm{k}-1}$

$$head_i = softmax\left(\frac{(QW_i^Q)(KW_i^K)^T}{\sqrt{d_K}} \circ \mathrm{M}\right)(VW_i^V)$$

$$\frac{(QW_i^Q)(KW_i^K)^T}{\sqrt{d_k}} = \begin{bmatrix} \frac{\boldsymbol{q}_1\boldsymbol{k}_1^T}{\sqrt{d_1}} & -\infty & -\infty \\ \frac{\boldsymbol{q}_2\boldsymbol{k}_1^T}{\sqrt{d_2}} & \frac{\boldsymbol{q}_2\boldsymbol{k}_2^T}{\sqrt{d_2}} & -\infty \\ \frac{\boldsymbol{q}_3\boldsymbol{k}_1^T}{\sqrt{d_3}} & \frac{\boldsymbol{q}_3\boldsymbol{k}_2^T}{\sqrt{d_3}} & \frac{\boldsymbol{q}_3\boldsymbol{k}_3^T}{\sqrt{d_3}} \end{bmatrix}$$

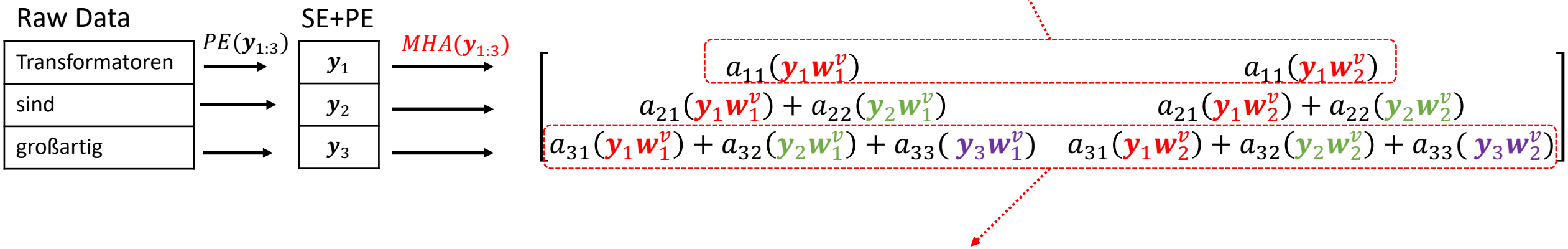
	Transformatoren	sind	großartig
Transformatoren	a_{11}	0	0
sind	a_{21}	a_{22}	0
großartig	a_{31}	a_{32}	a_{33}

$$softmax\left(\frac{(QW_i^Q)(KW_i^K)^T}{\sqrt{d_k}}\right) = \begin{bmatrix} softmax\left(\left[\frac{\boldsymbol{q}_1\boldsymbol{k}_1^T}{\sqrt{d_1}} & -\infty & -\infty\right]\right) \\ softmax\left(\left[\frac{\boldsymbol{q}_2\boldsymbol{k}_1^T}{\sqrt{d_2}} & \frac{\boldsymbol{q}_2\boldsymbol{k}_2^T}{\sqrt{d_2}} & -\infty\right]\right) \\ softmax\left(\left[\frac{\boldsymbol{q}_3\boldsymbol{k}_1^T}{\sqrt{d_3}} & \frac{\boldsymbol{q}_3\boldsymbol{k}_2^T}{\sqrt{d_3}} & \frac{\boldsymbol{q}_3\boldsymbol{k}_3^T}{\sqrt{d_3}}\right]\right) \end{bmatrix} = \begin{bmatrix} a_{11} & 0 & 0 \\ a_{21} & a_{22} & 0 \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

Attention Scores

$$softmax\left(\frac{(QW_i^Q)(KW_i^K)^T}{\sqrt{d_k}}\right)\begin{bmatrix}\boldsymbol{y}_1\boldsymbol{w}_1^v & \boldsymbol{y}_1\boldsymbol{w}_2^v\\ \boldsymbol{y}_2\boldsymbol{w}_1^v & \boldsymbol{y}_2\boldsymbol{w}_2^v\\ \boldsymbol{y}_3\boldsymbol{w}_1^v & \boldsymbol{y}_3\boldsymbol{w}_2^v\end{bmatrix} = \begin{bmatrix}a_{11} & 0 & 0\\ a_{21} & a_{22} & 0\\ a_{31} & a_{32} & a_{33}\end{bmatrix}\begin{bmatrix}\boldsymbol{y}_1\boldsymbol{w}_1^v & \boldsymbol{y}_1\boldsymbol{w}_2^v\\ \boldsymbol{y}_2\boldsymbol{w}_1^v & \boldsymbol{y}_2\boldsymbol{w}_2^v\\ \boldsymbol{y}_3\boldsymbol{w}_1^v & \boldsymbol{y}_3\boldsymbol{w}_2^v\end{bmatrix}$$

Embedding of “Transformatoren” does not depend on “sind”, and “großartig”



Embedding of “großartig” is a linear combination of embeddings of “Transformatoren”, “sind”, and “großartig”

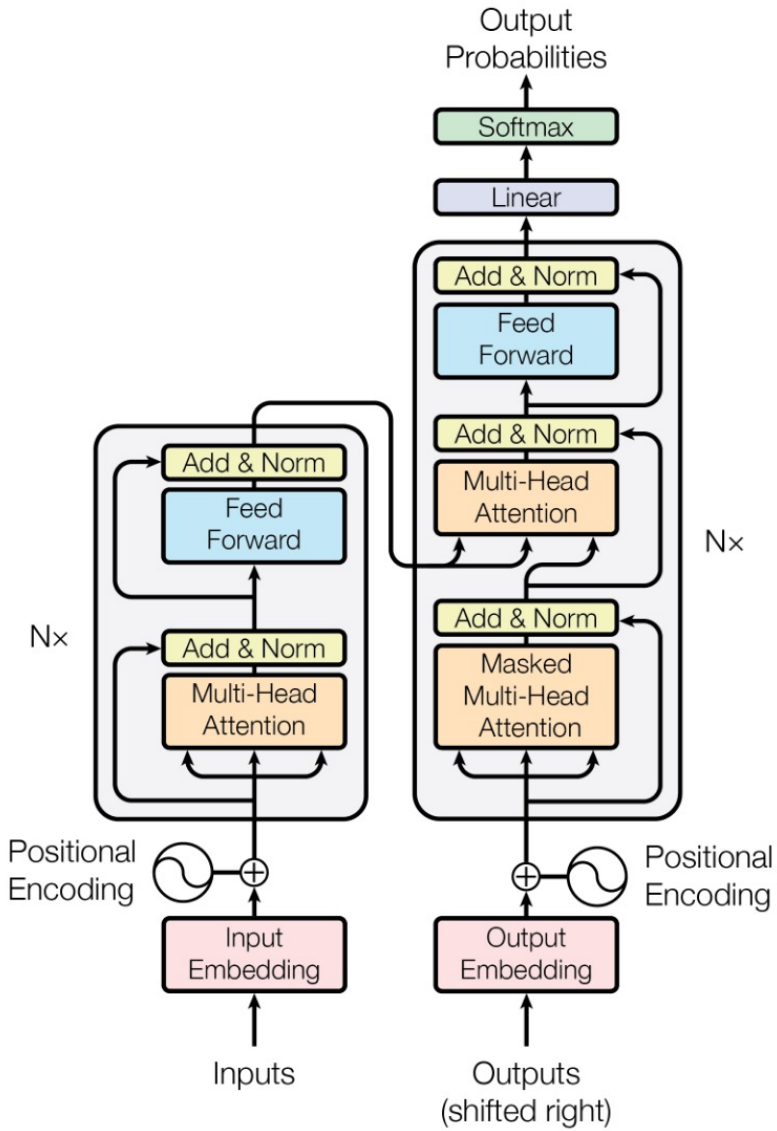
Cross-Attention Layer

- Same as self-attention layer, except that
  - K and V are from encoder
  - Q is from output sequence

- Why no masking?
  - Decoding starts only after entire encoder input is processed
  - Decoder's queries attend to the encoder's keys/values
  - Attending to any source position does not leak future target information
  - Therefore, no causal mask is needed in cross-attention.

- Streaming or online tasks
  - Eg: Live translation, etc.
  - We may need masking

$P(y_k|\boldsymbol{x}_{1:\mathrm{N}}, \boldsymbol{y}_{1:k-1})$

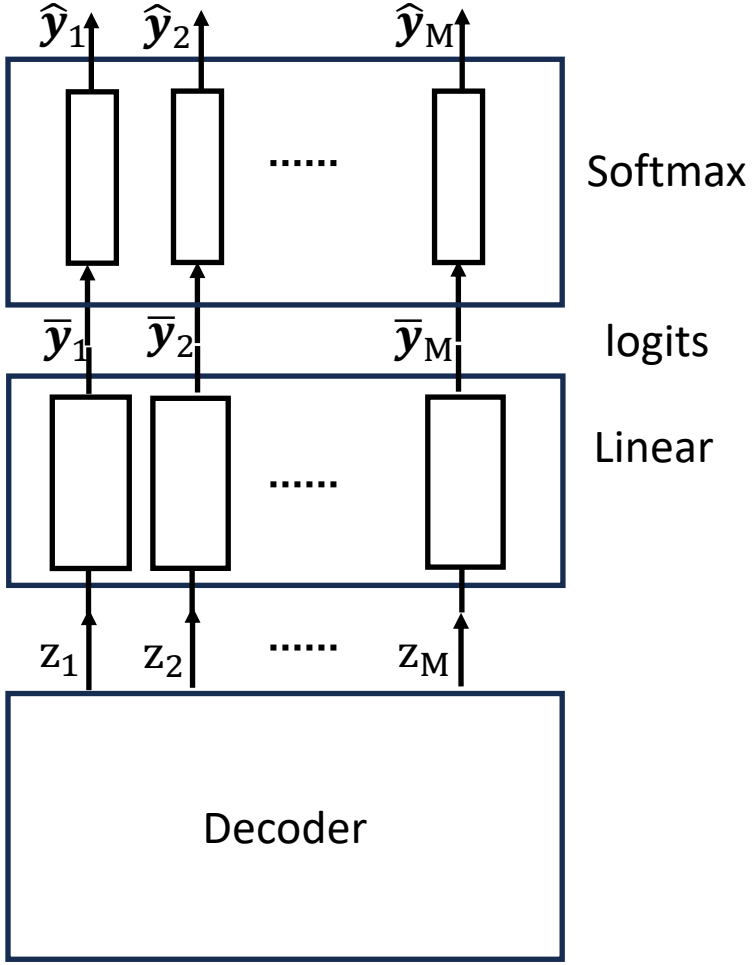


$X = \boldsymbol{x}_{1:\mathrm{N}}$

$Y = \boldsymbol{y}_{1:\mathrm{k}-1}$

Output Layer

- Let $Z \in R^{M \times d}$ be the output of decoder, where M is the output sequence length and d is the embedding dimension
  - $Z = [z_1 \quad z_2 \quad \cdots \quad z_M]$, where $z \in R^{d \times 1}$



$$\hat{y}_i = \text{softmax}(\bar{y}_i)$$

$$\hat{y}_i \in R^{|V| \times 1}$$

$$\bar{y}_i \in R^{|V| \times 1}$$

$$\bar{y}_i = (W^O)^T z_i$$

$$W^O \in R^{d \times |V|}$$