DSE 3121 DEEP LEARNING

Transformers

B.Tech Data Science & Engineering
Rohini R Rao & Abhilash Pai
Department of Data Science and Computer Applications
MIT Manipal

Transformers

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

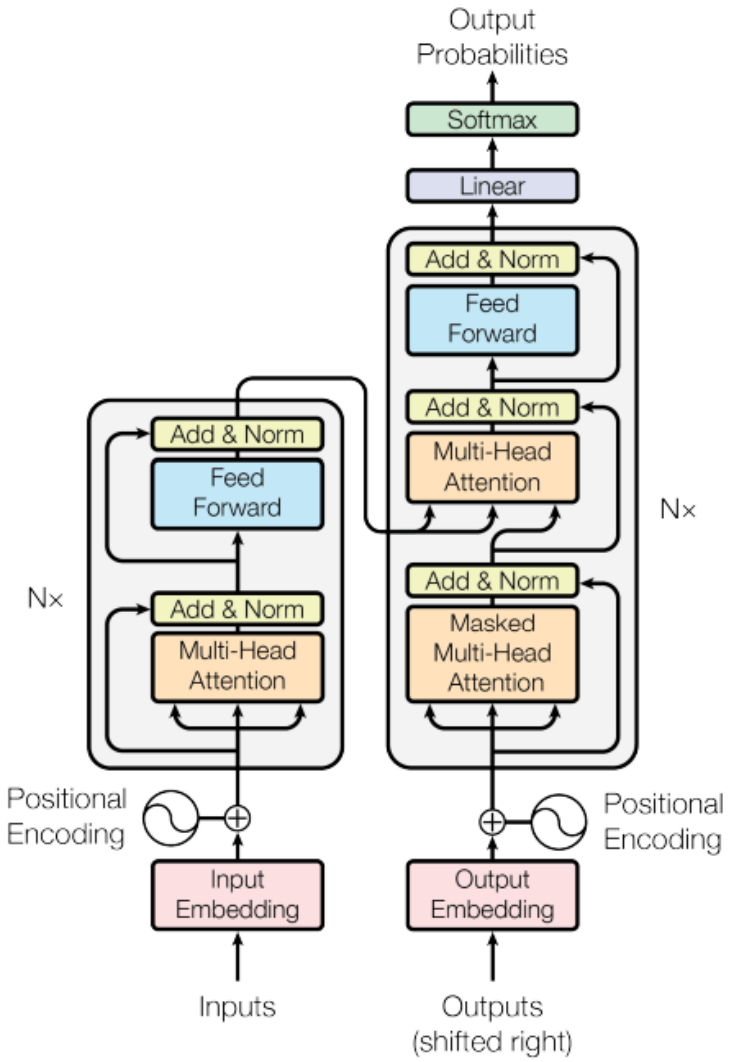
Illia Polosukhin* ‡ illia.polosukhin@gmail.com

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 41.0 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature.

Vaswani, Ashish, et al. "Attention is all you need." Advances in neural information processing systems 30 (2017).

Transformers

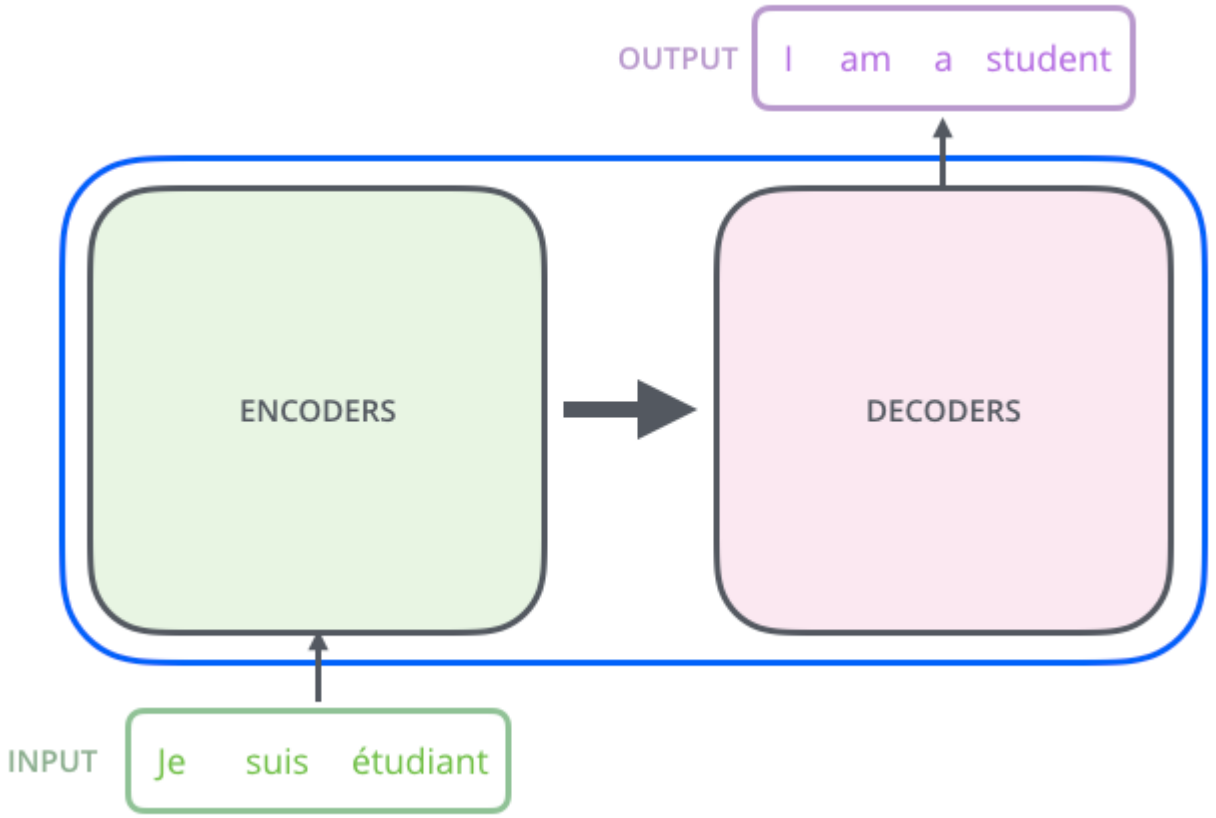


- Vaswani et al. 2017 proposed the transformer model, entirely built on **self attention mechanism without using sequence aligned recurrent architectures.**

- Key components:
  - Self attention
  - Multi-head attention
  - Positional encoding
  - Encoder-Decoder architecture

Vaswani, Ashish, et al. "Attention is all you need." Advances in neural information processing systems 30 (2017).

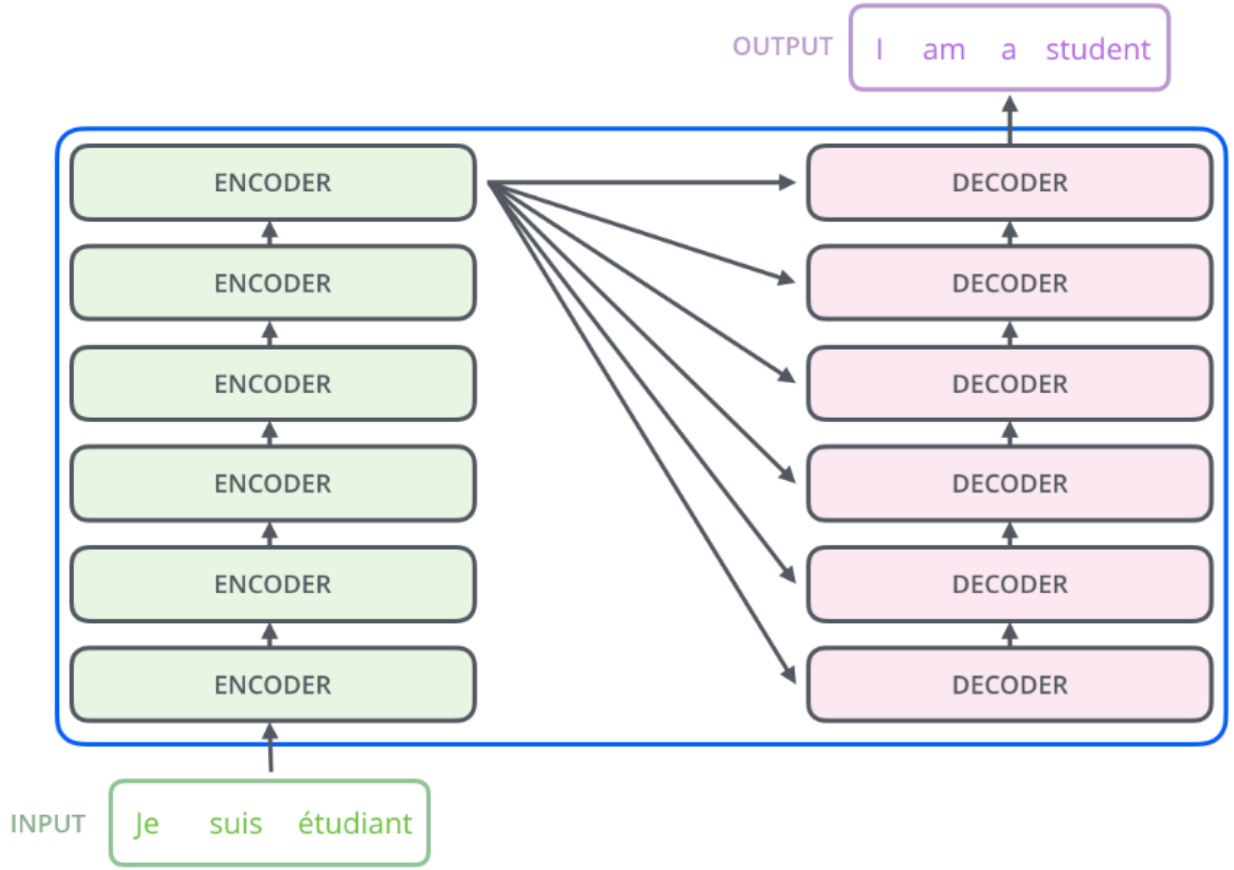
Architecture



The entire network can be viewed as an encoder decoder architecture

The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time. (jalammar.github.io)

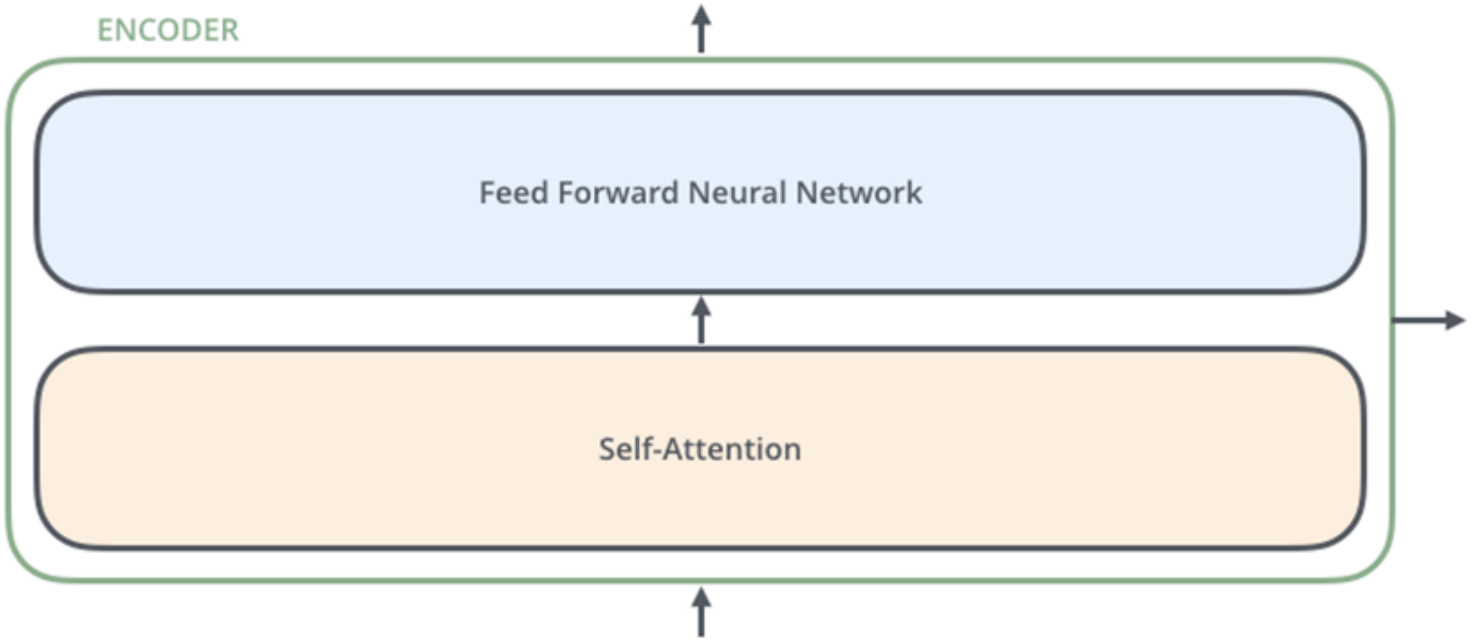
Architecture



The encoding component is a stack of encoders and the decoding component is also a stack of encoders of the same number (in the paper, this number = 6)

The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time. (jalammar.github.io)

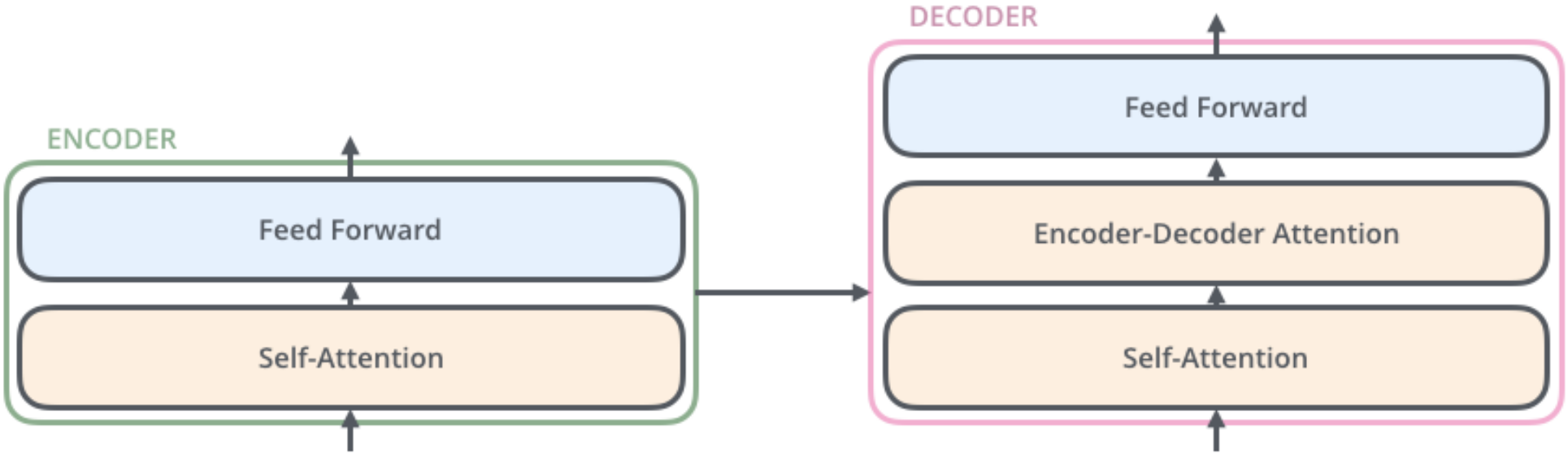
Architecture



- The encoders are all identical in structure (yet they do not share weights).
- Each one is broken down into two sub-layers
- The encoder's inputs first flow through a self-attention layer – a layer that helps the encoder look at other words in the input sentence as it encodes a specific word – to the feed forward neural network.

The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time. (jalammar.github.io)

Architecture

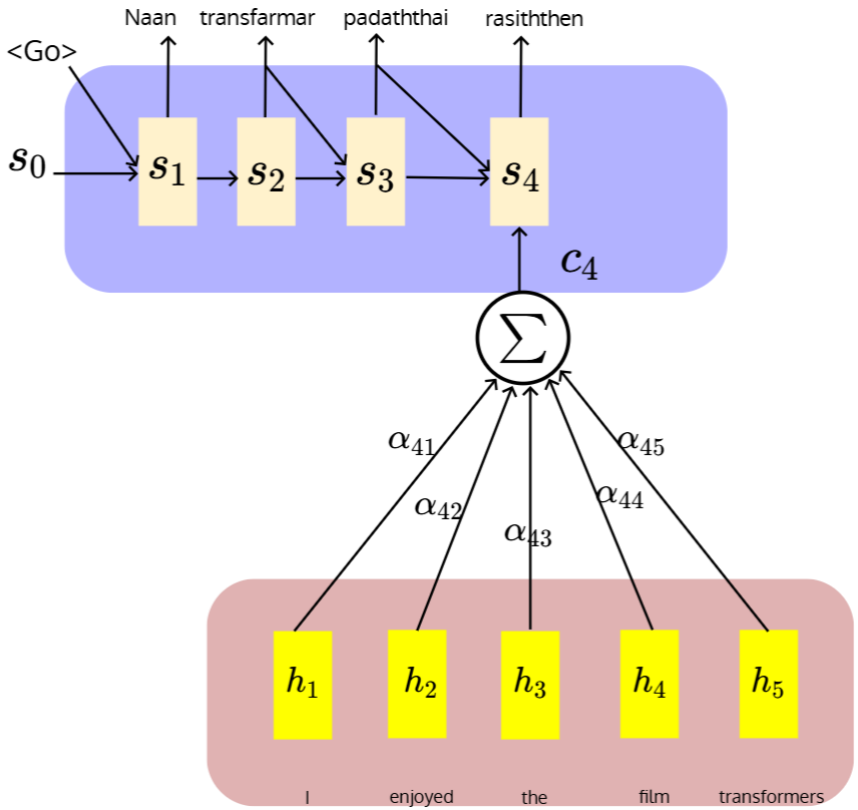


The decoder has both those layers, but between them is an attention layer that helps the decoder focus on relevant parts of the input sentence (similar what attention does in seq2seq models).

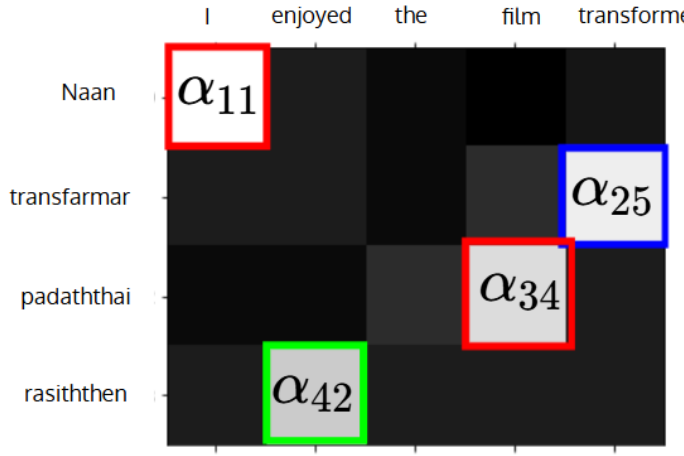
the encoder decoder attn similar to of seq2seq is called cross attention

The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time. (jalammar.github.io)

Recap of Attention Mechanism



Alignment of words:



$y_t \rightarrow$ Naan | transfarmar | padaththai | rasiththen | I | enjoyed | the film | transformers

$$\mathbf{c}_t = \sum_{i=1} \alpha_{ti} \boldsymbol{h}_i \quad ; \text{Context vector for output } y_t$$

$$\begin{bmatrix} | & | & | & | & | \\ h_1 & h_2 & h_3 & h_4 & h_5 \\ | & | & | & | & | \end{bmatrix} \begin{bmatrix} \alpha_{41} \\ \alpha_{42} \\ \alpha_{43} \\ \alpha_{44} \\ \alpha_{45} \end{bmatrix}$$

n, number of words

t= time step for decoder

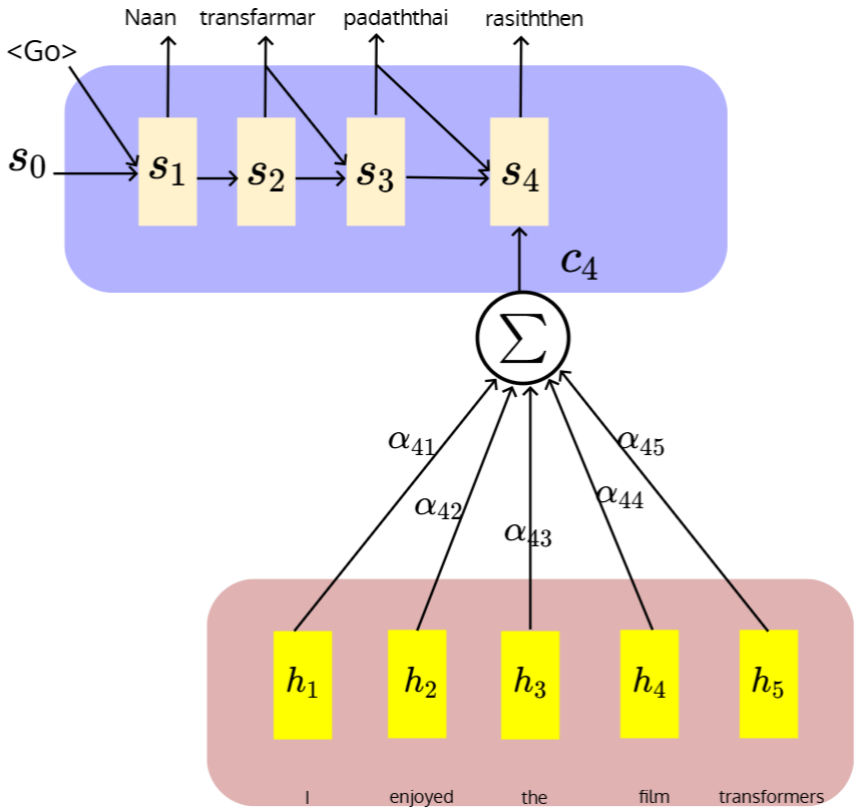
$$\alpha_{ti} = align(y_t, h_i)$$

$$= \frac{\exp(\text{score}(\boldsymbol{s}_{t-1}, \boldsymbol{h}_i))}{\sum_{i'=1}^{n} \exp(\text{score}(\boldsymbol{s}_{t-1}, \boldsymbol{h}_{i'}))}$$

e_{ti}

Source: CS7015 Deep Learning, Dept. of CSE, IIT Madras

Recap of Attention Mechanism



$$\alpha_{ti} = align(y_t, h_i)$$

$$= \frac{\exp(\text{score}(\boldsymbol{s}_{t-1}, \boldsymbol{h}_i))}{\sum_{i'=1}^{n} \exp(\text{score}(\boldsymbol{s}_{t-1}, \boldsymbol{h}_{i'}))}$$

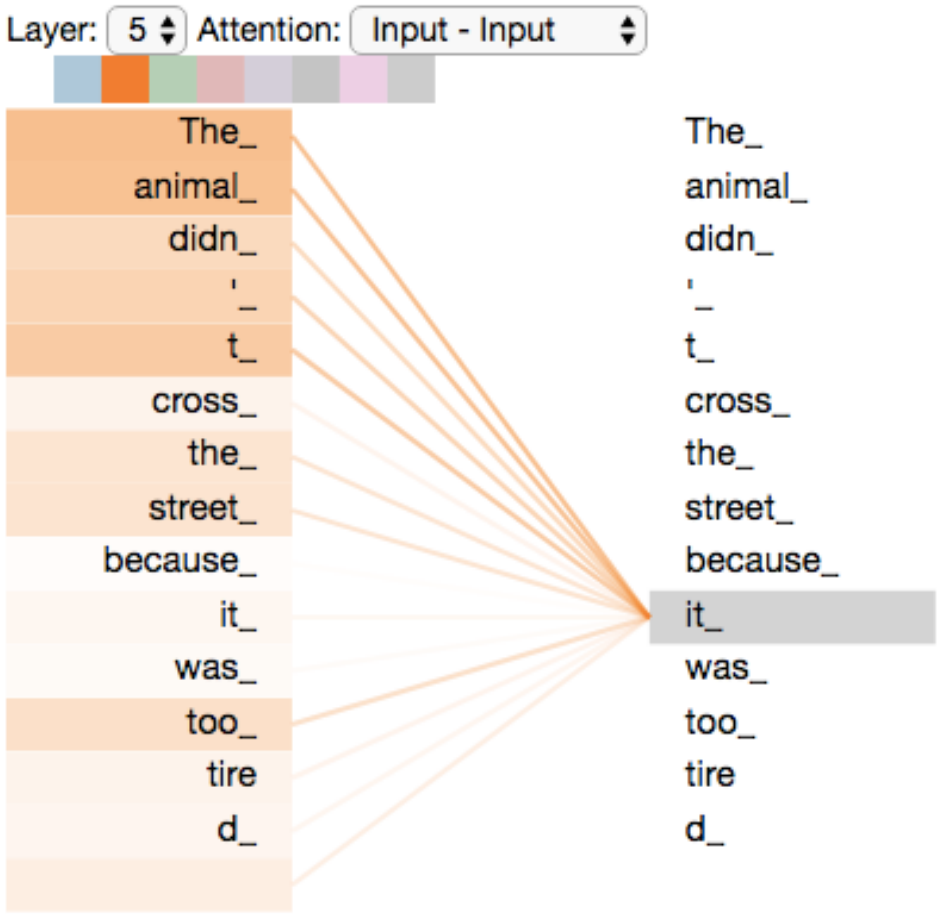
- Can α_{ti} be computed in parallel for all i at time step t ?
  - Yes! h_i is available for all i and s_{t-1} is also available at time step t.
  - Take away: **Attention can be parallelized.**

- **Can c_i for all i, be computed in parallel?**
  - No! As each c_i is dependent on s_{i-1}

Limitation: Given a training example, we can't **parallelize the sequence of computations across time steps.**

Source: CS7015 Deep Learning, Dept. of CSE, IIT Madras

Self-attention in Transformers



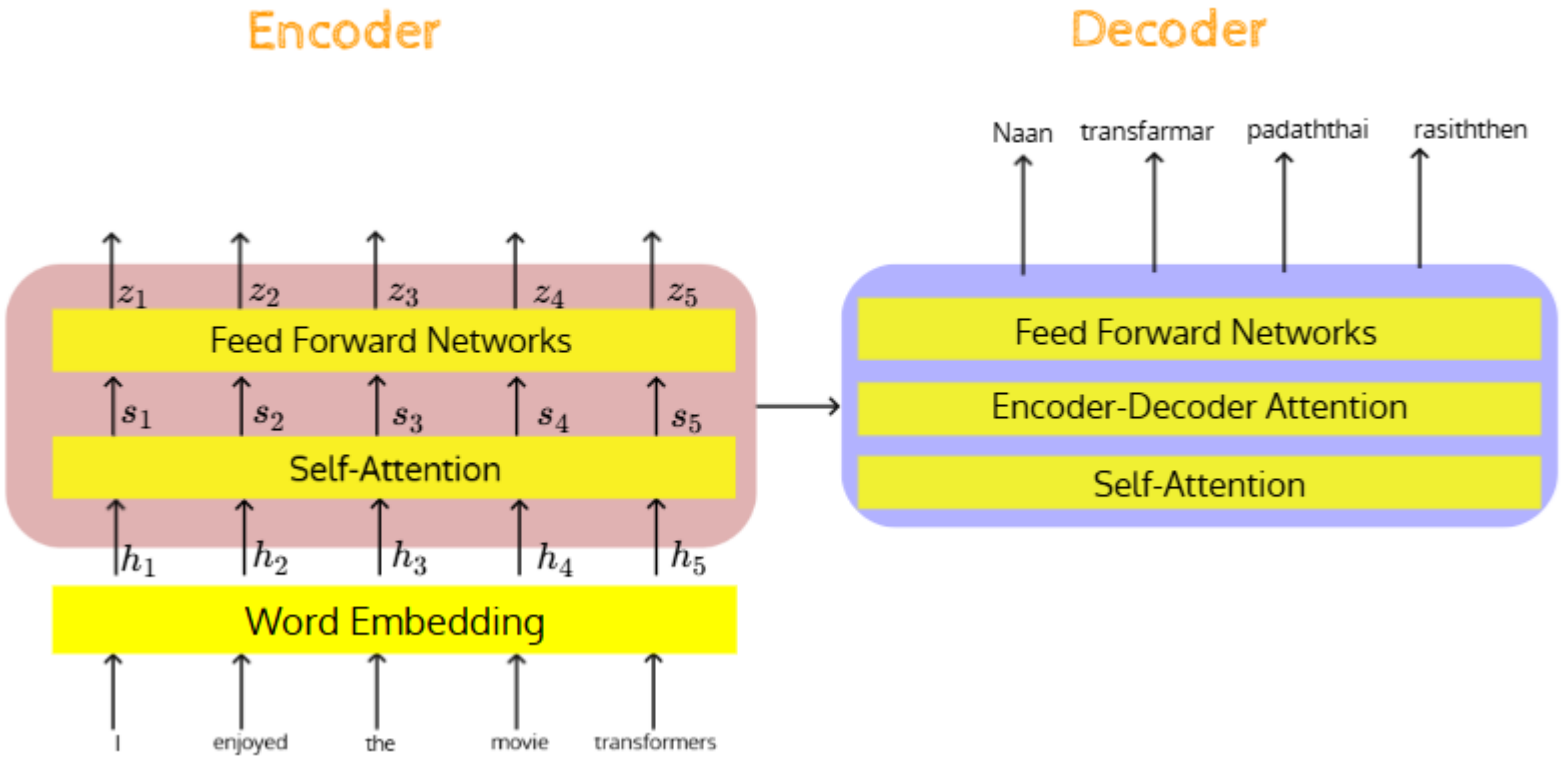
S1: Animal didn't cross the street because it was too tired

S2: Animal didn't cross the street because it was too wide

- In sentence S1, "it" refers to "animal" and in sentence S2 "it" refers to "street".
- Such deductions are hard for traditional sequence to sequence models.
- While processing each word in a sequence, **self-attention mechanism allows the model to decide as to which other parts of the same sequence it needs to focus on**, which makes such deductions easier and allows better encoding.
- RNNs which maintain a hidden state to incorporate the representation of previous vectors are no longer needed!

The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time. (jalammar.github.io)

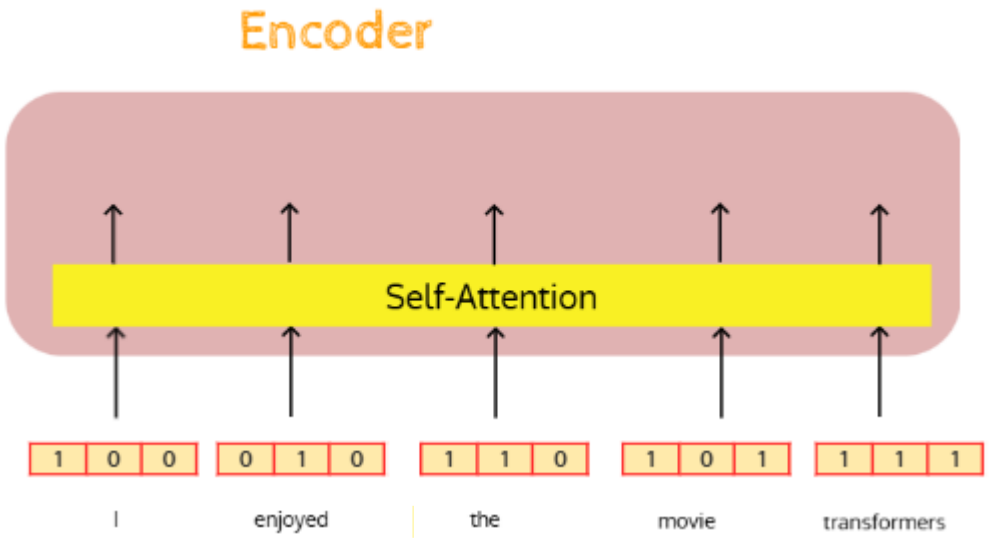
Computation of Self-attention



Consider the above notations for the encoder

Source: CS7015 Deep Learning, Dept. of CSE, IIT Madras

Computation of Self-attention



- We are interested in the self attention mechanism.
- Note that the inputs are vectors (word embeddings) and the outputs are also vectors.
- We know what attention is. All it requires is a pair of vectors as input.

Source: CS7015 Deep Learning, Dept. of CSE, IIT Madras

Computation of Self-attention

Goal: Given a word in a sentence, we want to compute the relational score between the **word** and the **rest of the words** in the **sentence** such that the score is higher if they are related **contextually**

- Now both the vectors (hi) and (hj), for all i,j are available for all the time
- Does it allow us to compute the values for the rows parallelly?

h_i

h_j	The	animal	didn't	cross	the	street	bacause	it
The	0.6	0.1	0.05	0.05	0.02	0.02	0.02	0.1
animal	0.02	0.5	0.06	0.15	0.02	0.05	0.01	0.12
didn't	0.01	0.35	0.45	0.1	0.01	0.02	0.01	0.03
cross	.							
the	.							
street	.							
because	.							
it	0.01	0.6	0.02	0.1	0.01	0.2	0.01	0.01

For eg: more weight should be given to "animal" when we are calculating the contextual representation for the word "it"

Source: CS7015 Deep Learning, Dept. of CSE, IIT Madras

Computation of Self-attention

- Recall that in the earlier Encoder-Decoder Attention model, we used the score function $e_{jt,}$ defined as:

One dot product

$$score(s_{t-1}, h_j) = V_a^T \tanh(U_{att} s_{t-1} + W_{att} h_j)$$

One Non-linearity

Two linear transformations

- There are three vectors (s,h,v) involved in computing the score for each time step
- However, the input to the self-attention module is only the word embeddings h_j for all j
- So, we need to get three vectors for each word embedding!

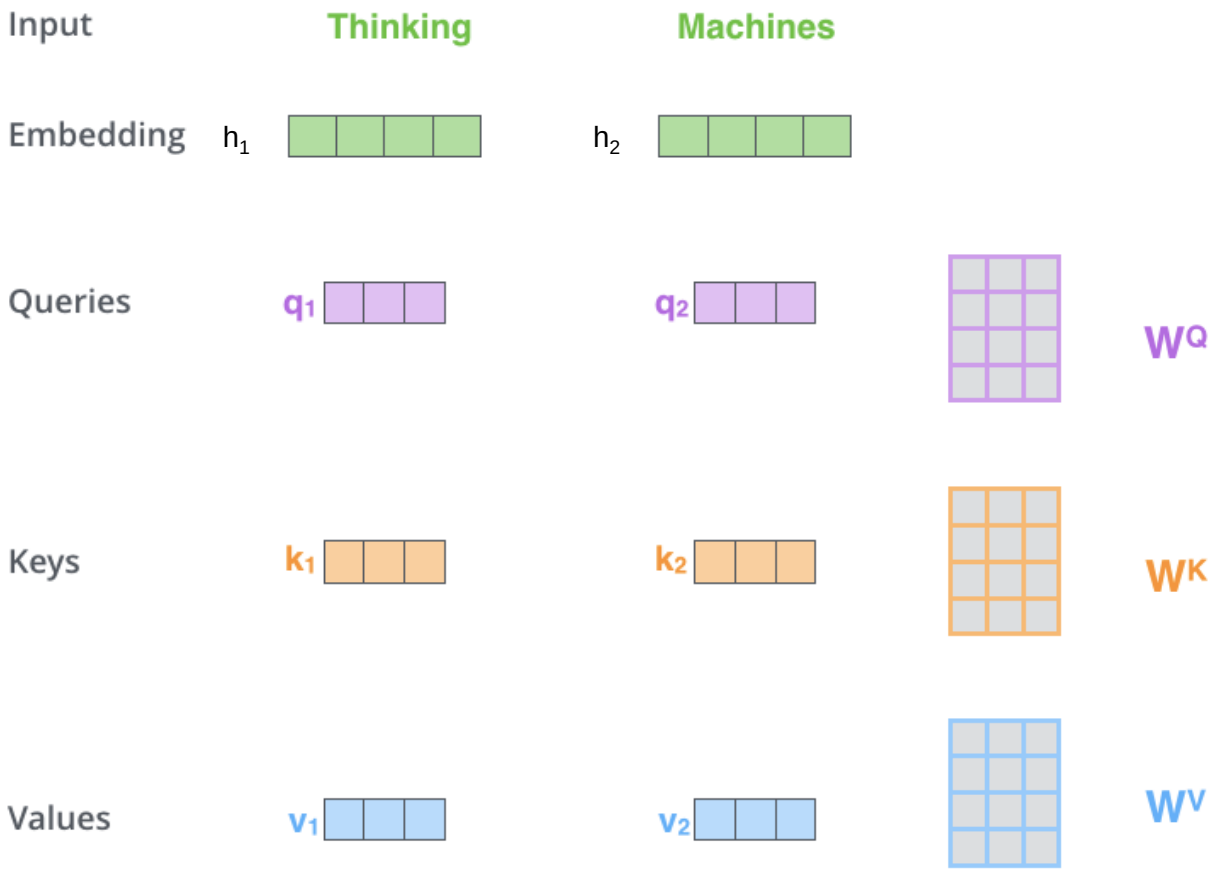
Source: CS7015 Deep Learning, Dept. of CSE, IIT Madras

Computation of Self attention

Step 1: Create three vectors from encoder's input embedding vector (h_i):

- Query vector (q_i)
- Key vector (k_i)
- Value vector (v_i)

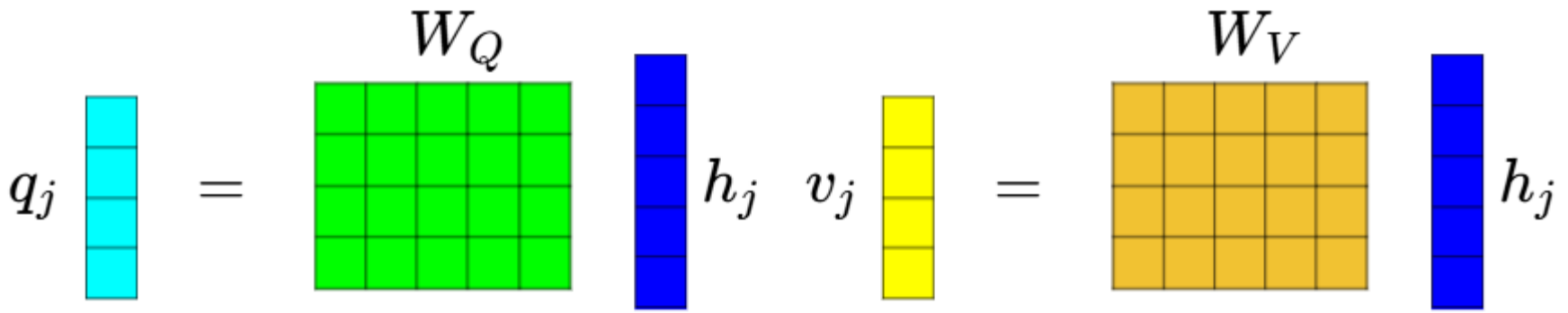
These are created by multiplying the input with weight matrices W^q, W^k, W^v, learned during training.

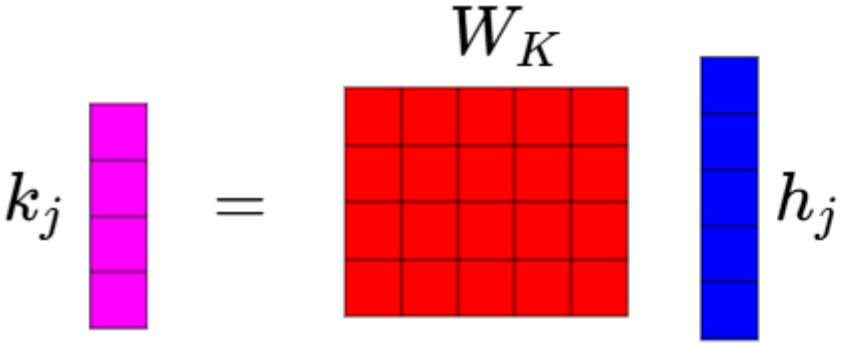


- Note: In the paper by Vaswani et al, the **q, k and w** vectors has dimension of **64** and the input vector h_i has a dimension of **512**,

Source: CS7015 Deep Learning, Dept. of CSE, IIT Madras

Computation of Self attention





q_j is called query vector for the word embedding h_j

k_j is called key vector for the word embedding h_j

v_j is called value vector for the word embedding h_j

W_Q, W_K and W_V are called respective linear transformation (parameter) matrices.

Source: CS7015 Deep Learning, Dept. of CSE, IIT Madras

Computation of Self attention

For each word embedding calculate query, key and value vectors

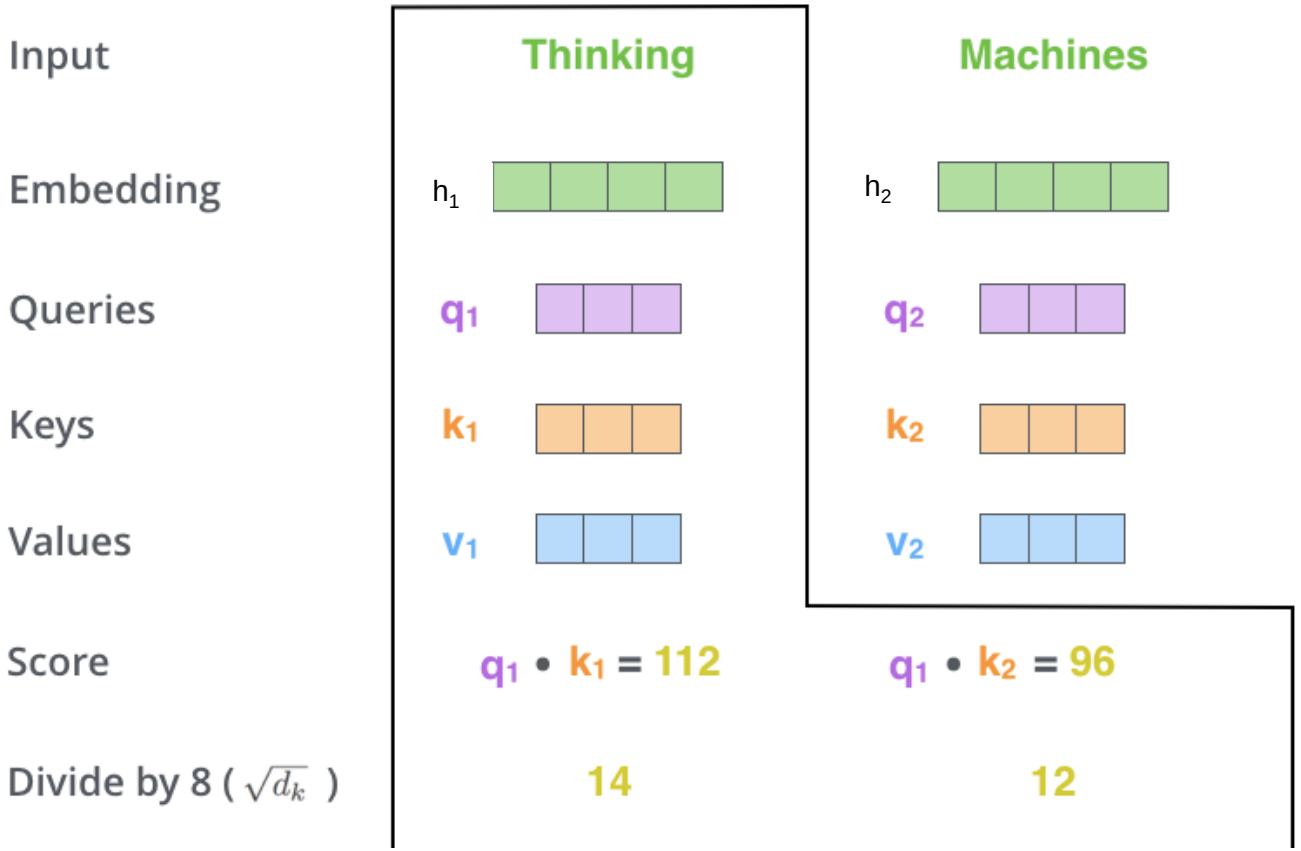
Source: CS7015 Deep Learning, Dept. of CSE, IIT Madras

Transformers : Self attention

Step 2: Calculate self attention scores of each word against the other words in the same sentence.

- This is done by taking the dot product of query vector with the key vector of respective words.
- The scores are the divided by square root of the key length/dimension.

- This is called **Scaled Dot-Product attention** - it leads to more stable gradients.



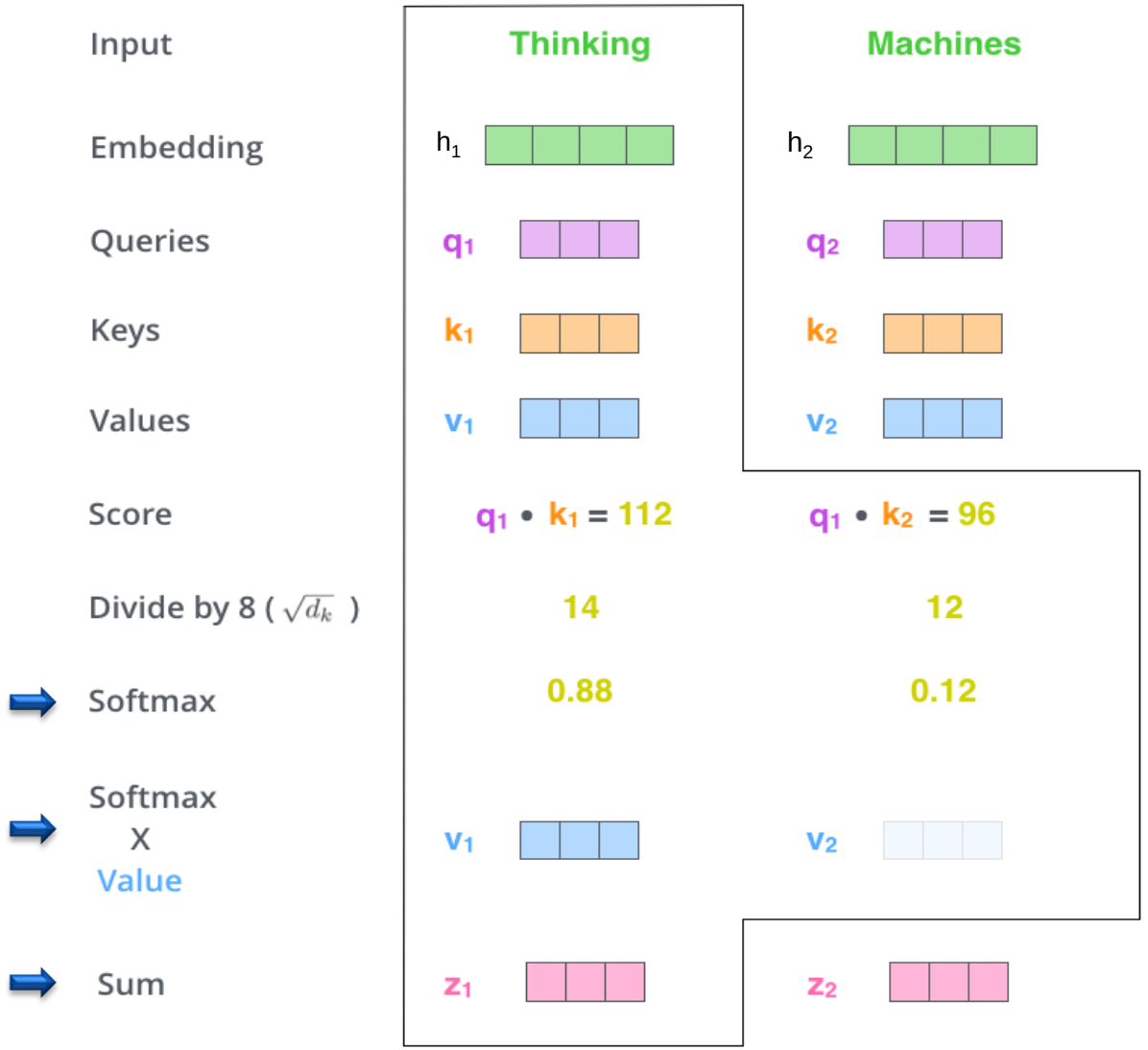
The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time. (jalammar.github.io)

Transformers : Self attention

Step 3: Softmax is used to obtain normalized probability scores; determines how much each word will be expressed at this position.

Step 4: Multiply each value vector by the Softmax score; to keep values of words we want to focus on and drown out irrelevant words.

Step 5: Sum up the weighted value vectors ; produces the output of self-attention layer at this position (for first word)



The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time. (jalammar.github.io)

Computation of Self attention

We are interested in calculating:

$score\ (q_i, k_j)$

Score function essentially computes the dot product of q and k

Source: CS7015 Deep Learning, Dept. of CSE, IIT Madras

Computation of Self attention

$$e_1 = score\,(q_1\,,\,k_j)$$

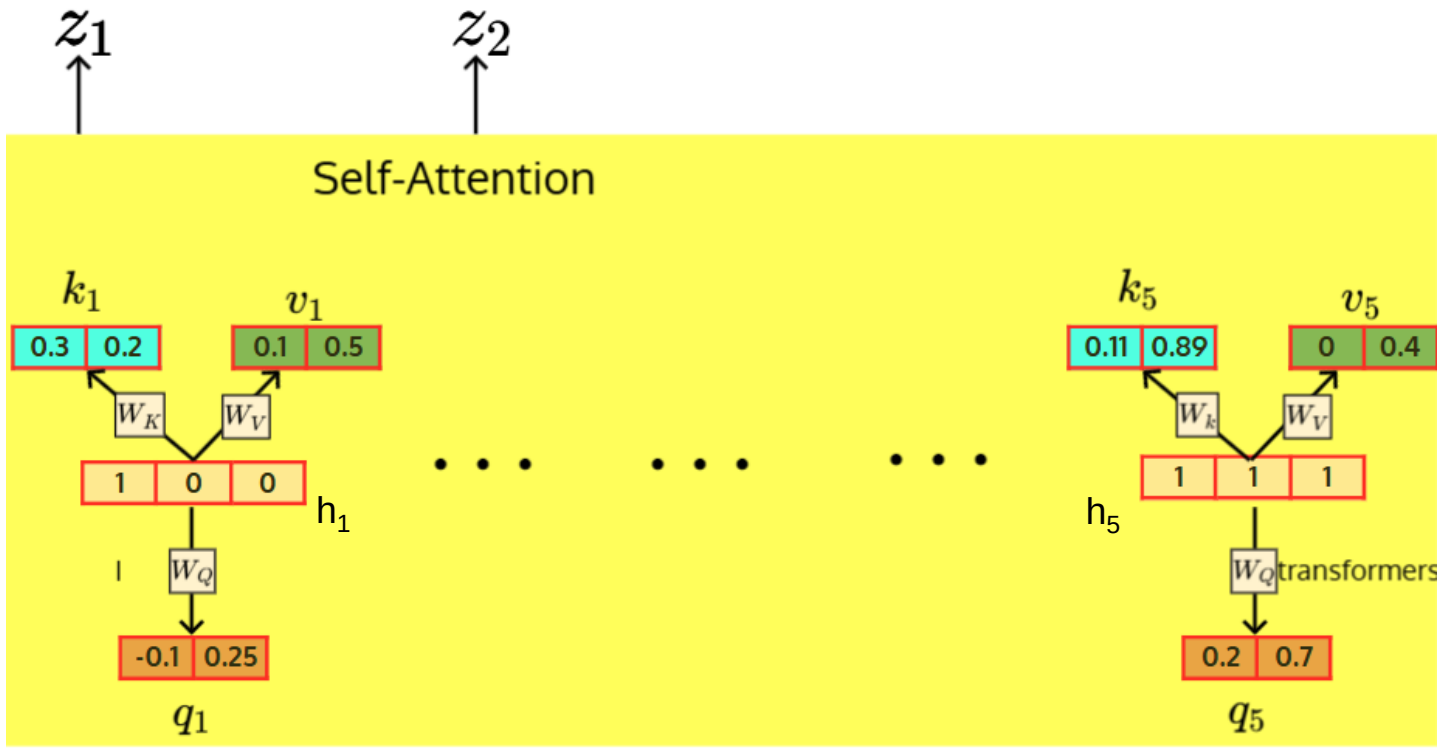
$$e_1 = [q_1 \cdot k_1, \quad q_1 \cdot k_2, \quad \cdots, \quad \cdots \quad q_1 \cdot k_5]$$

$$\alpha_{1j} = softmax(\frac{e_{1j}}{\sqrt{d_k}})$$

$$z_1 = \sum_{j=1}^{5} \alpha_{1j} v_j$$

Source: CS7015 Deep Learning, Dept. of CSE, IIT Madras

Computation of Self attention



$$e_1 = score\,(q_2\,,\,k_j)$$

$$e_{2j} = [q_2 \cdot k_1, \quad q_2 \cdot k_2, \quad \cdots, \quad \cdots \quad q_2 \cdot k_5]$$

$$\alpha_{2j} = softmax\left(\frac{e_{2j}}{\sqrt{d_k}}\right)$$

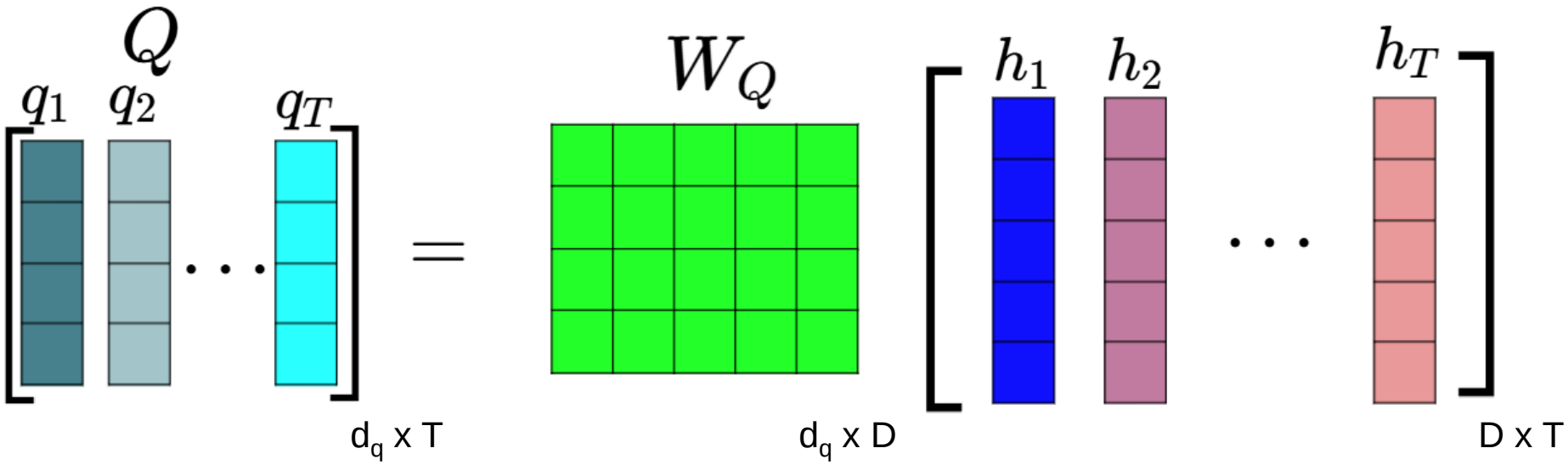
$$z_2 = \sum_{j=1}^{5} \alpha_{2j} v_j$$

Repeat the procedure to obtain all other z

Source: CS7015 Deep Learning, Dept. of CSE, IIT Madras

Computation of Self attention

- In the actual implementation, however, Step 1 to Step 4 is done in vectorized form to compute all the z's in parallel.



All the Q's can be calculated in parallel.

D: Embedding dimension (most often called as d_{model})

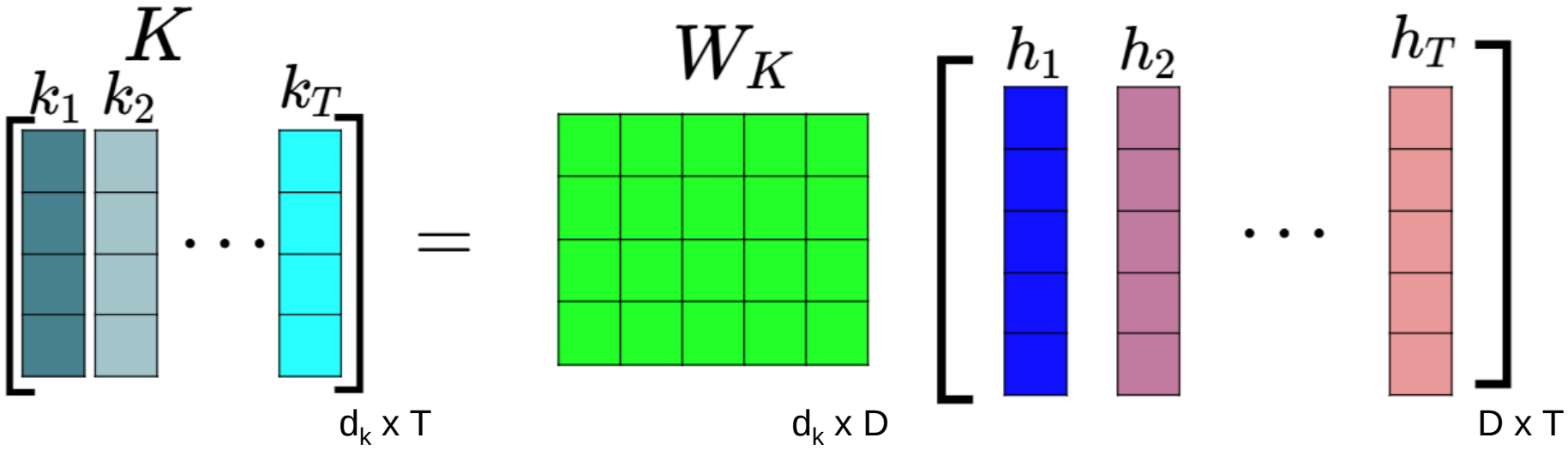
T: size of the input

d_q : Dimension of query vector

Source: CS7015 Deep Learning, Dept. of CSE, IIT Madras

Computation of Self attention

- In the actual implementation, however, Step 1 to Step 4 is done in vectorized form to compute all the z's in parallel.



All the K's can be calculated in parallel.

Note: $d_k = d_q = (D/H)$
Where, H is the no. of attention heads (discussed later)

D: Embedding dimension

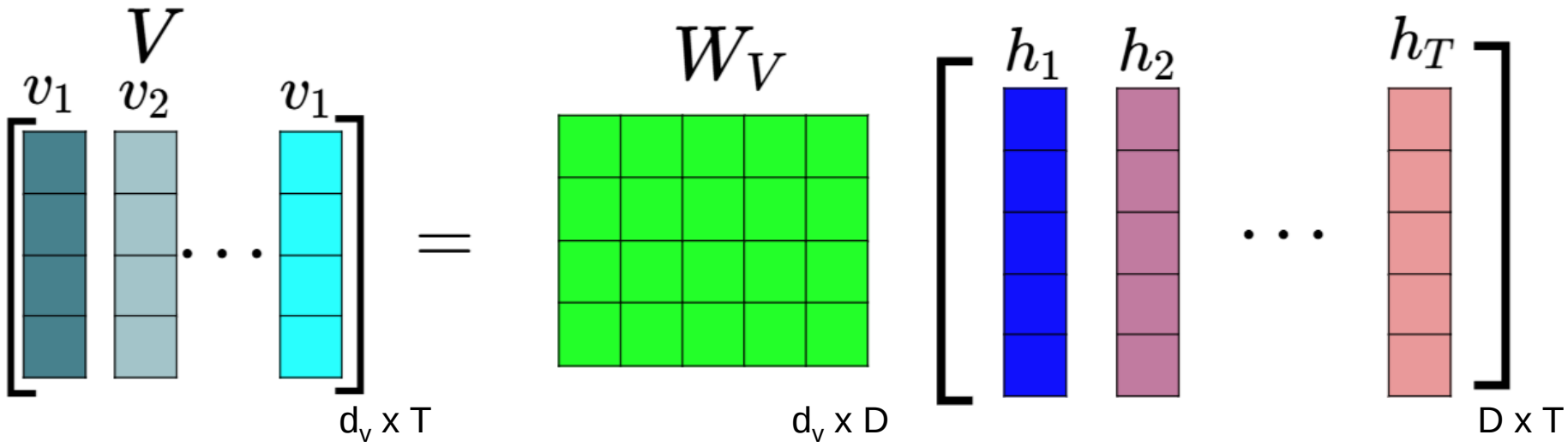
T: size of the input

d_k : Dimension of query vector

Source: CS7015 Deep Learning, Dept. of CSE, IIT Madras

Computation of Self attention

- In the actual implementation, however, Step 1 to Step 4 is done in vectorized form to compute all the z's in parallel.



All the V's can be calculated in parallel.

D: Embedding dimension

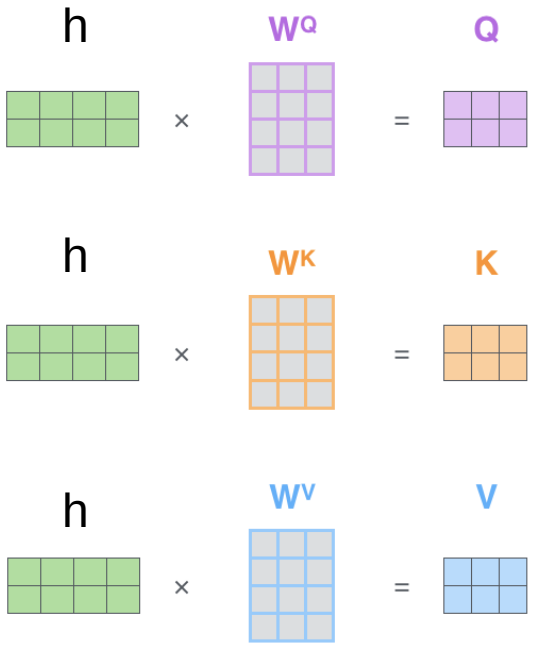
T: size of the input

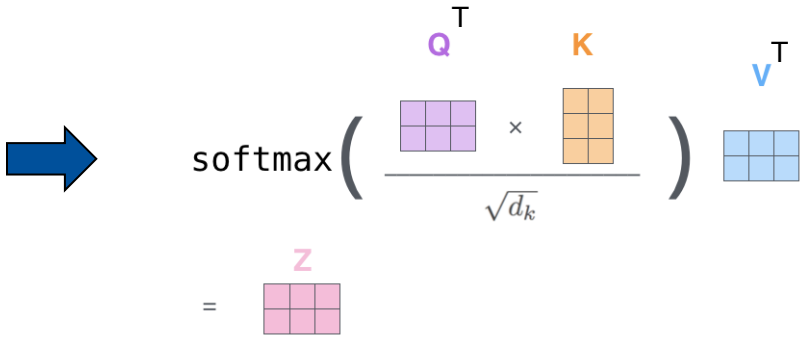
d_v : Dimension of value vector

Note: d_v need not be same a d_k and d_q
However, in the paper by Vaswani et al., $d_v = dk = dq$

Source: CS7015 Deep Learning, Dept. of CSE, IIT Madras

Computation of Self attention: Vectorized Output





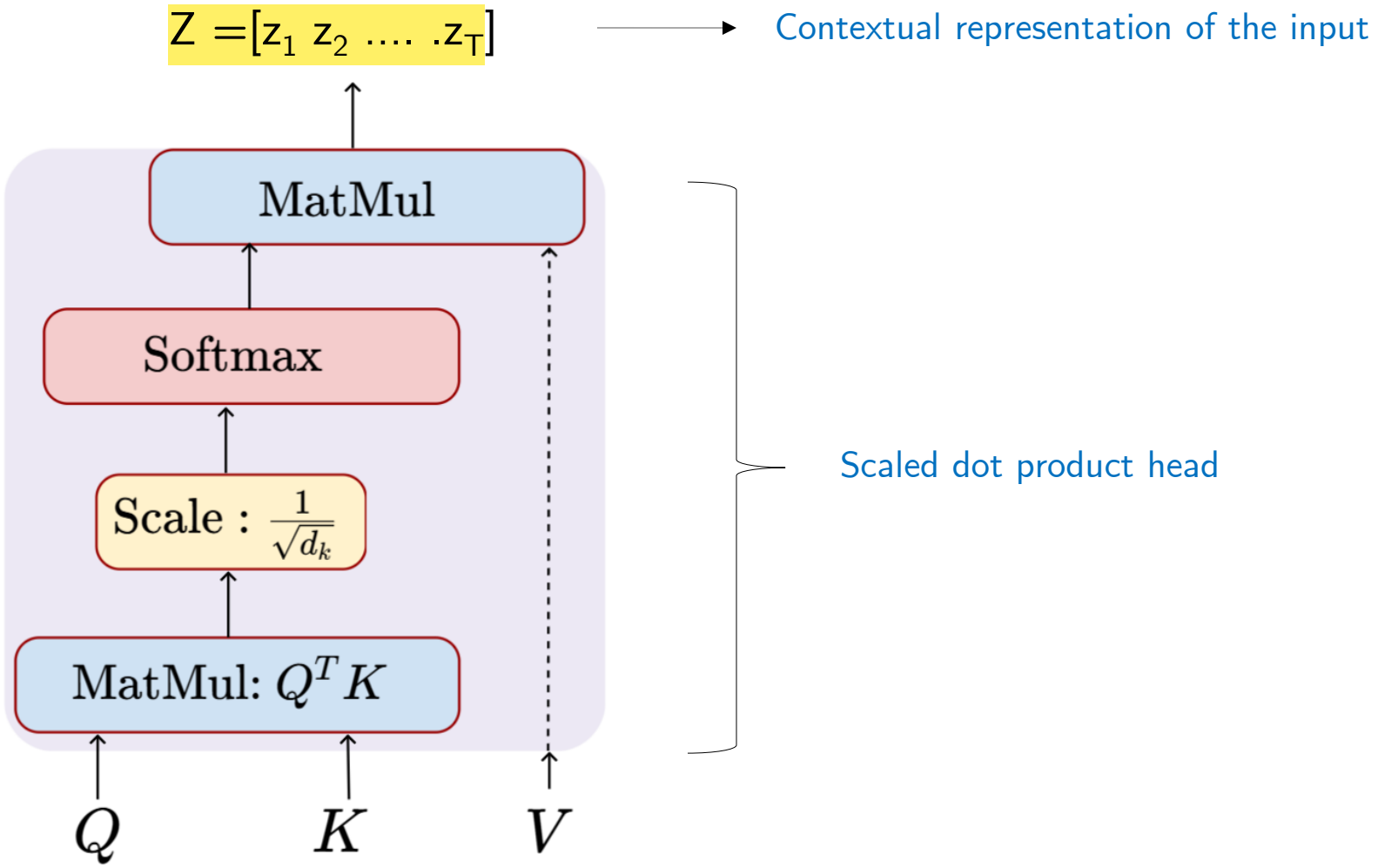
$$Z = [z_1, z_2, \cdots, z_T]$$
$$= softmax(\frac{Q^T K}{\sqrt{d_k}})V^T$$

[where d_k is the dimension of key/query vector]

Since d_k scales the values of $Q^T K$, it is called a scaled dot-product

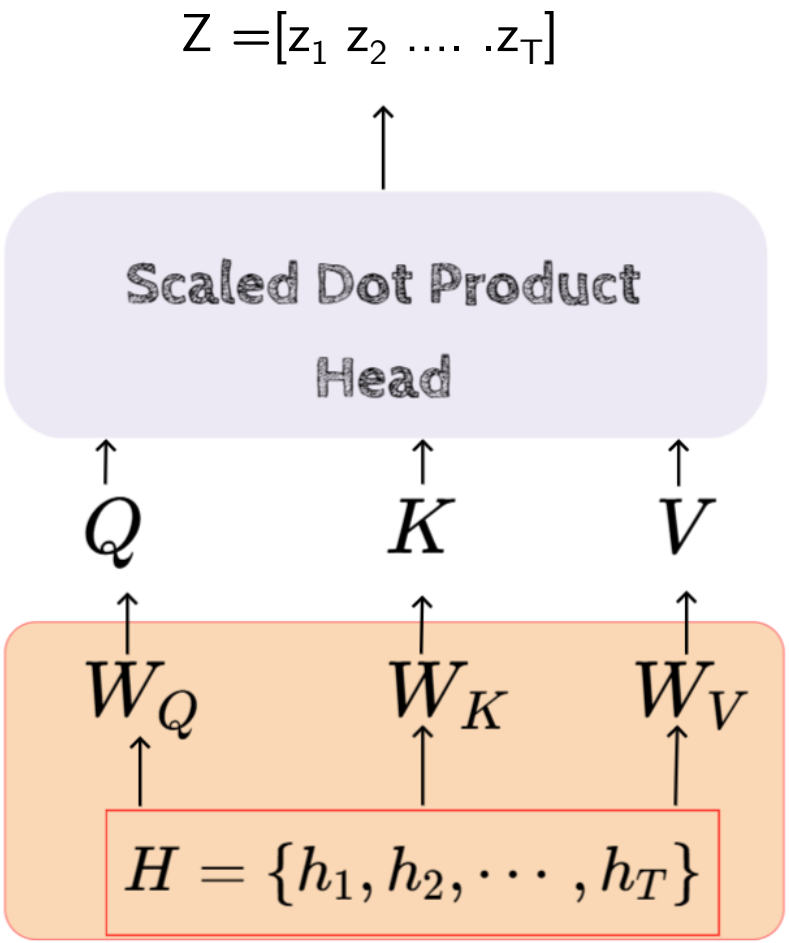
$$dim(Q^T K) : T \times 64 \times 64 \times T = T \times T$$
$$dim(Z) : T \times T \times T \times 64 = T \times 64$$

Self attention layer



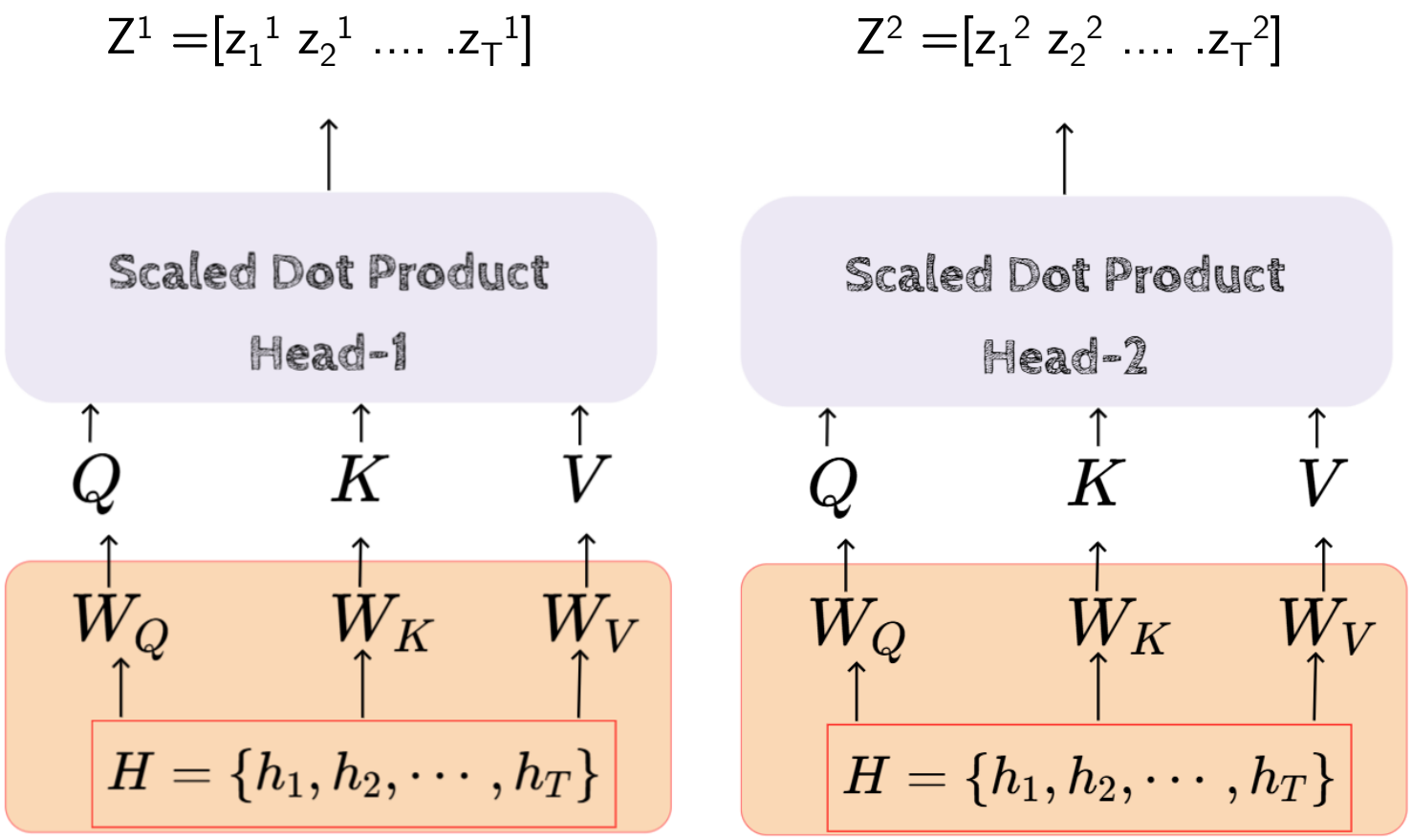
Source: CS7015 Deep Learning, Dept. of CSE, IIT Madras

Input + Self attention layer



Source: CS7015 Deep Learning, Dept. of CSE, IIT Madras

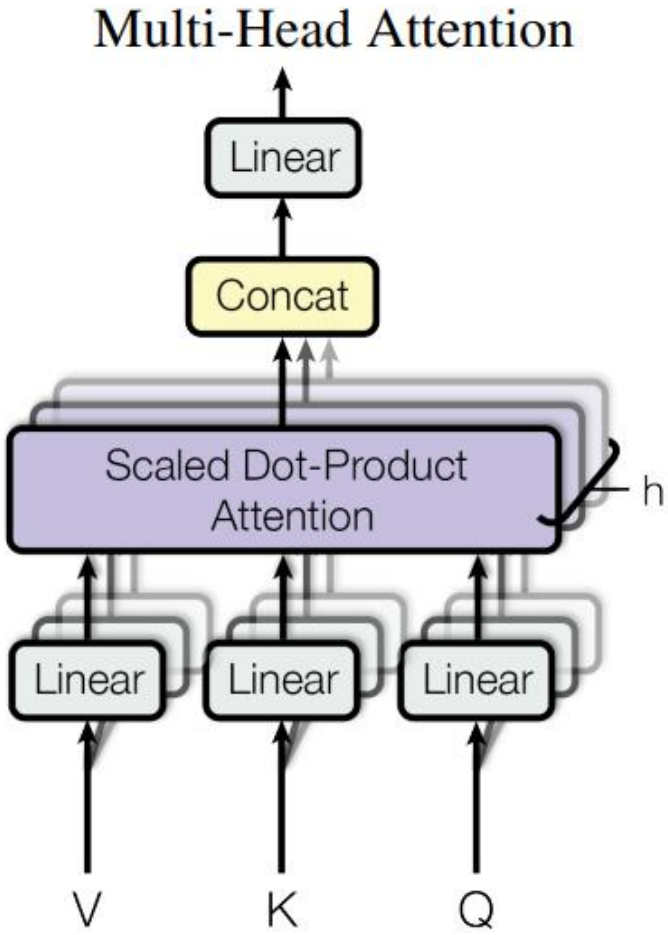
Two-head attention



Source: CS7015 Deep Learning, Dept. of CSE, IIT Madras

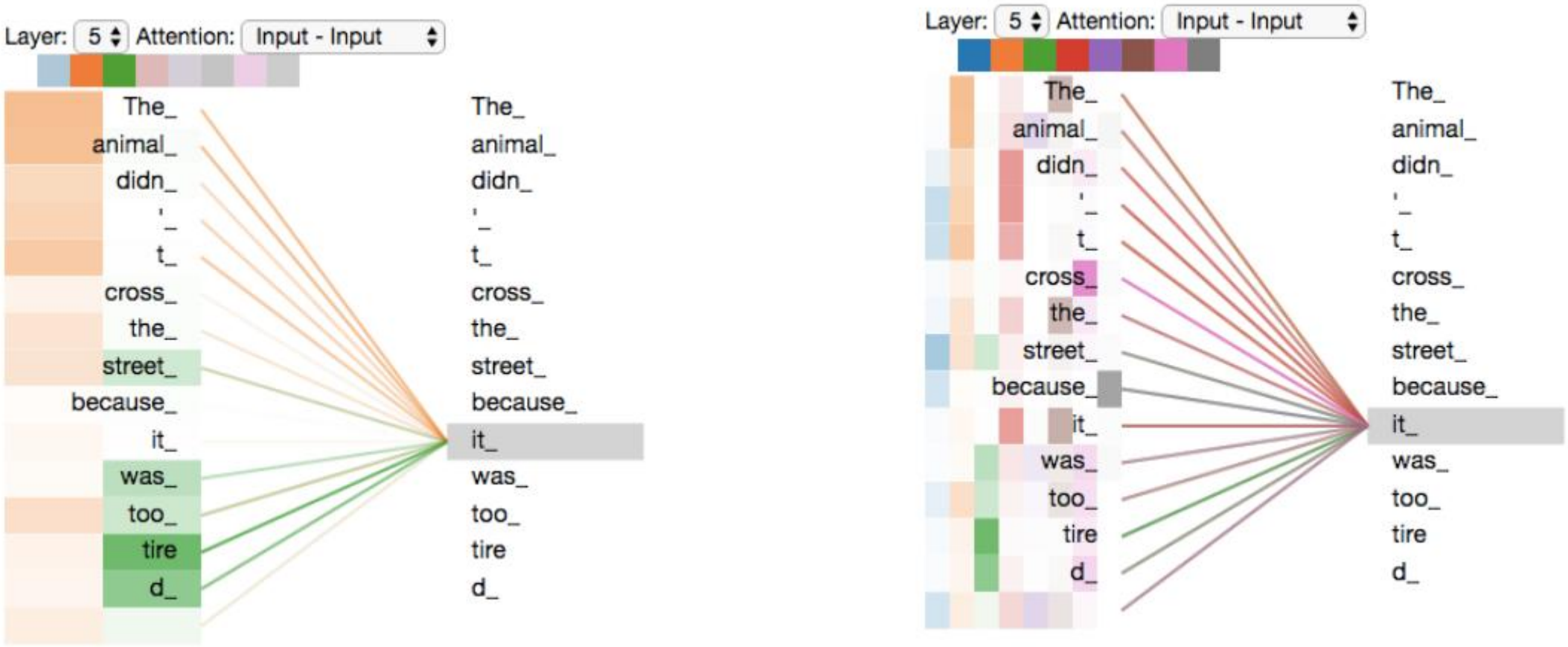
Multi-head Attention

- Multi-head attention improves the performance of attention layer in two ways:
  - It expands the model's ability to focus on different positions.
  - It gives the attention layer multiple "representation subspaces" i.e., we have not only one, but multiple sets of Query/Key/Value weight matrices.
  - Just like CNNs use multiple filters to learn more abstract representations and capture meaningful interactions between inputs, having multiple self attention heads with different parameter matrices (W_Q^i, W_K^i, W_v^i) enables to capture subtle contextual information.



The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time. (jalammar.github.io)

Multi-head Attention



- As we encode the word "it", one attention head is focusing most on "the animal", while another is focusing on "tired" – in a sense, the model's representation of the word "it" bakes in some of the representation of both "animal" and "tired".

The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time. (jalammar.github.io)

Multi-head Attention

Yes, let's break down the multi-head attention mechanism in the encoder of the "Attention is All You Need" model, particularly focusing on the steps you mentioned.

1. Multi-Head Attention Mechanism

The multi-head attention layer in the encoder has multiple (8 in this case) attention heads that allow the model to focus on different parts of the input sequence simultaneously. Here's the step-by-step process:

Step 1.1: Split into Multiple Heads

Each attention head operates independently and has its own set of learned projection matrices for queries Q, keys K, and values V. Given an input sequence of length T (with T being the number of tokens) and hidden dimension D (e.g., 512), each head projects the input to a lower dimension (64 in this case) using these projection matrices.

This means each head outputs an attention map of shape $T \times 64$.

Step 1.2: Compute Attention

Each attention head computes attention by performing a scaled dot-product between the queries and keys, applying a softmax to get attention weights, and then multiplying by the values. This results in each head producing an output of shape $T \times 64$.

Step 1.3: Concatenate Heads

The outputs from each of the 8 attention heads are concatenated along the last dimension, resulting in a matrix of shape $T \times 512$ (since $8 \times 64 = 512$).

2. Linear Transformation

After concatenating the outputs of the attention heads, this $T \times 512$ matrix is passed through a linear transformation using a weight matrix of shape 512×512. This projection effectively combines the information from the different heads and ensures that the output remains in the same dimension as the original input (i.e., $T \times 512$).

3. Feed-Forward Network

The output from the multi-head attention layer (which is $T \times 512$) is then fed into a position-wise feed-forward neural network. This feed-forward network consists of two linear transformations with a ReLU activation in between:

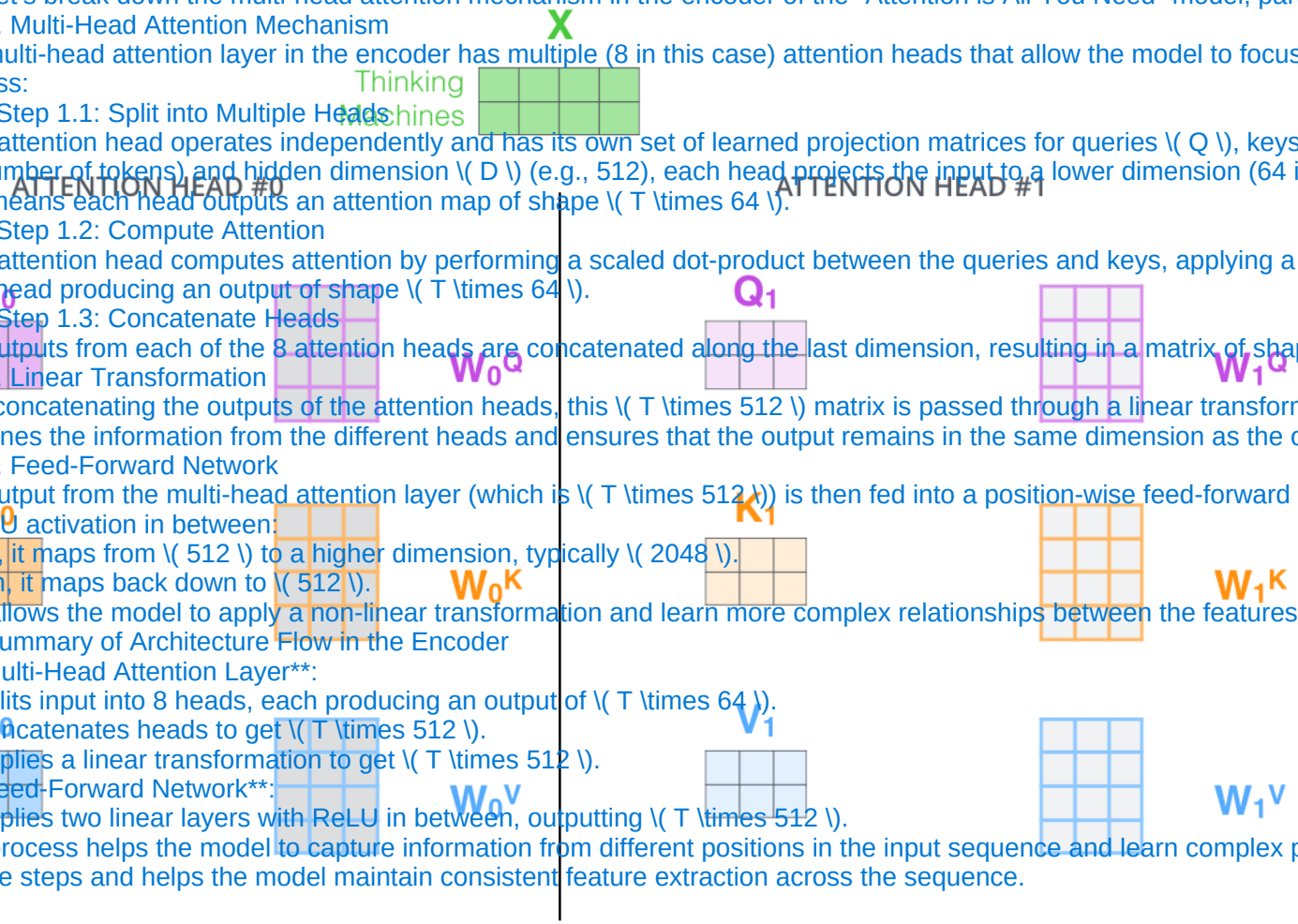
- First, it maps from 512 to a higher dimension, typically 2048.
- Then, it maps back down to 512.

This allows the model to apply a non-linear transformation and learn more complex relationships between the features while preserving the shape $T \times 512$.

Summary of Architecture Flow in the Encoder

1. **Multi-Head Attention Layer**:
   - Splits input into 8 heads, each producing an output of $T \times 64$.
   - Concatenates heads to get $T \times 512$.
   - Applies a linear transformation to get $T \times 512$.
2. **Feed-Forward Network**:
   - Applies two linear layers with ReLU in between, outputting $T \times 512$.

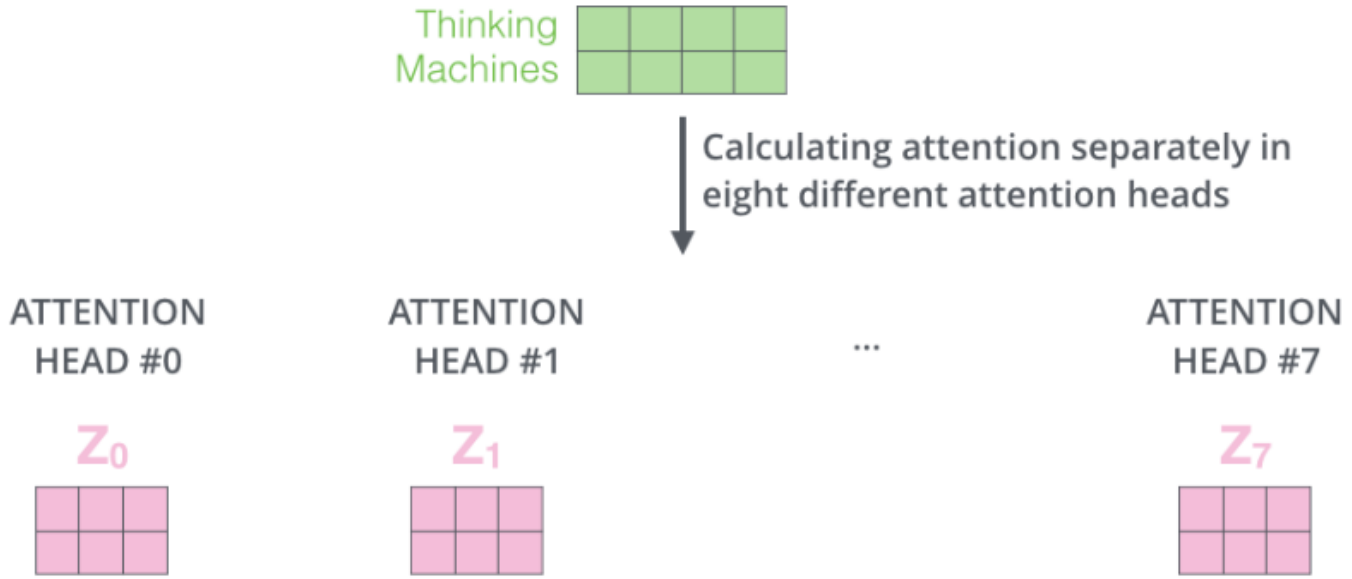
This process helps the model to capture information from different positions in the input sequence and learn complex patterns effectively. The single feed-forward network (not multiple) is shared across all time steps and helps the model maintain consistent feature extraction across the sequence.



With multi-headed attention, we maintain separate Q, K, V weight matrices for each head resulting in different Q, K, V matrices.

The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time. (jalammar.github.io)

Multi-head Attention



- Eight different weight matrices are obtained as an o/p.
- However, the feed-forward layer is not expecting eight matrices – it's expecting a single matrix (a vector for each word).

The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time. (jalammar.github.io)

Multi-head Attention

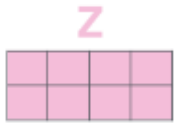
1) Concatenate all the attention heads

2) Multiply with a weight matrix W^O that was trained jointly with the model

X

3) The result would be the Z matrix that captures information from all the attention heads. We can send this forward to the FFNN

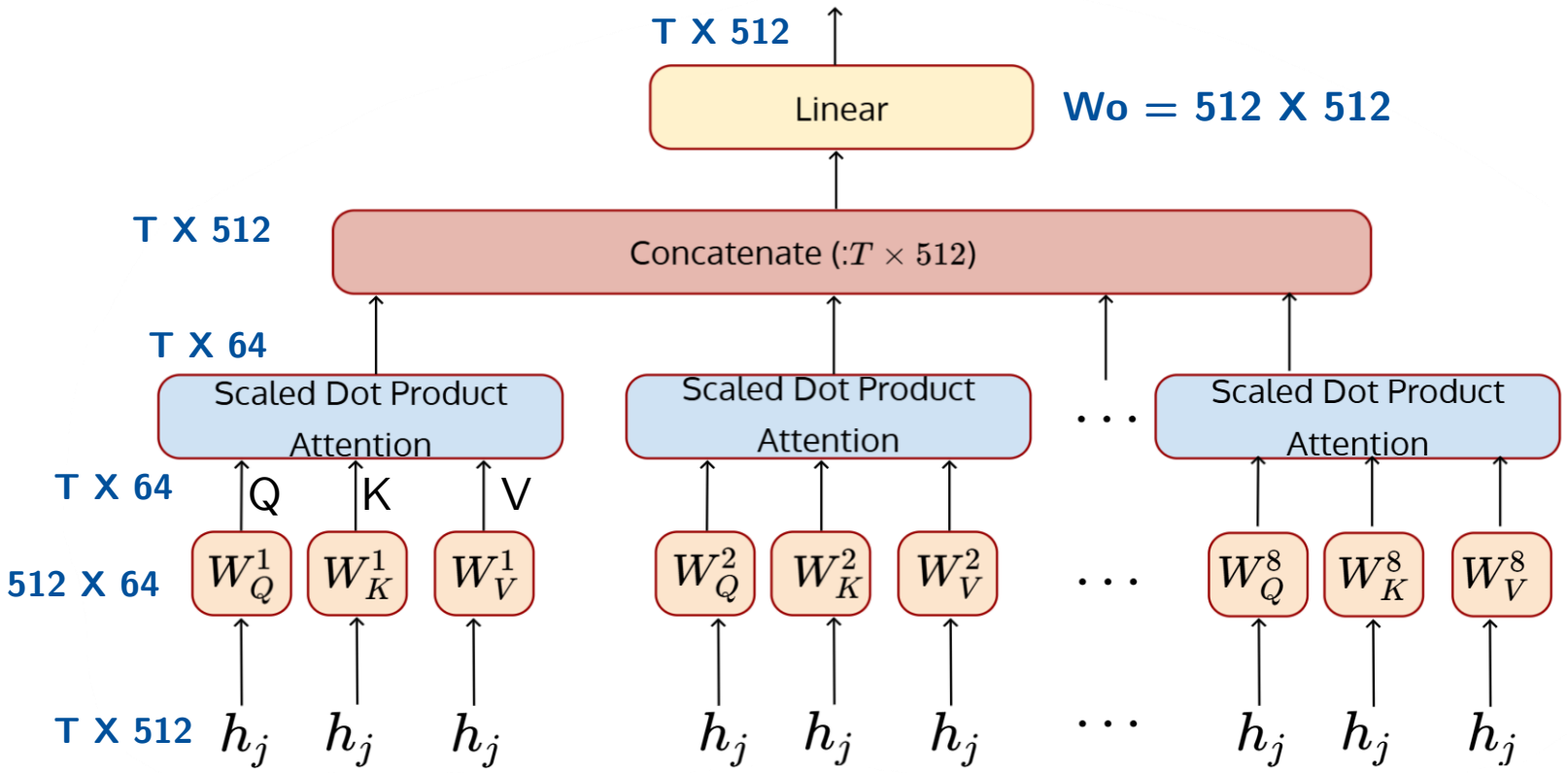
=



To tackle the issue, the eight matrices are concatenated and multiplied with additional weight matrix W^O

The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time. (jalammar.github.io)

Multi-head Attention



- The input is projected into h=8 different representation subspaces.
- So, the multi-head attention allows the model to jointly attend to information from **different** representation **subspaces** at different positions.

Source: CS7015 Deep Learning, Dept. of CSE, IIT Madras

Multi-head Attention

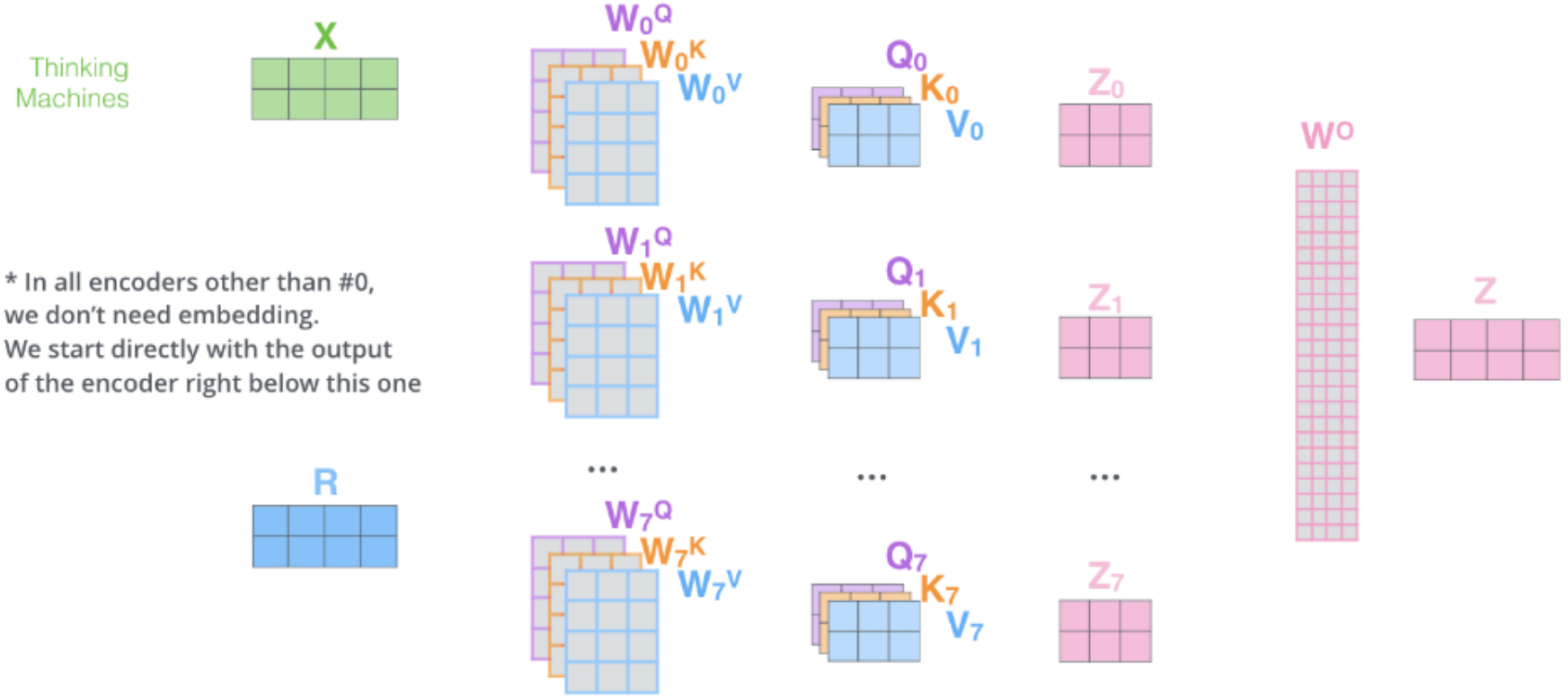
1) This is our input sentence*

2) We embed each word*

3) Split into 8 heads. We multiply X or R with weight matrices

4) Calculate attention using the resulting Q/K/V matrices

5) Concatenate the resulting Z matrices, then multiply with weight matrix W^O to produce the output of the layer



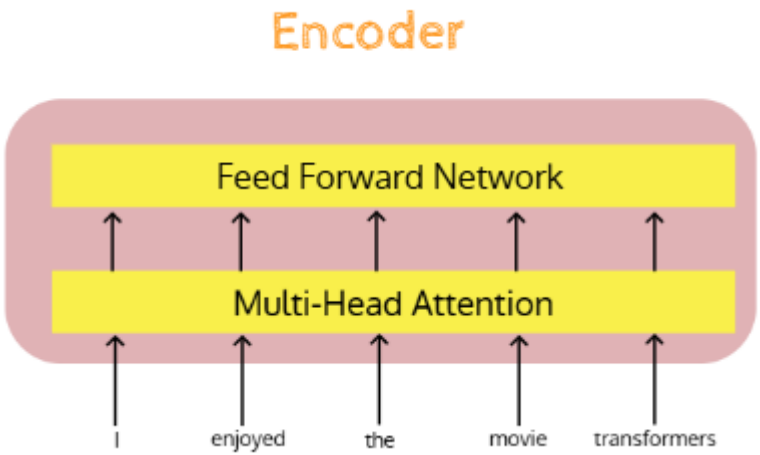
* In all encoders other than #0, we don't need embedding. We start directly with the output of the encoder right below this one

The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time. (jalammar.github.io)

Feed Forward Network

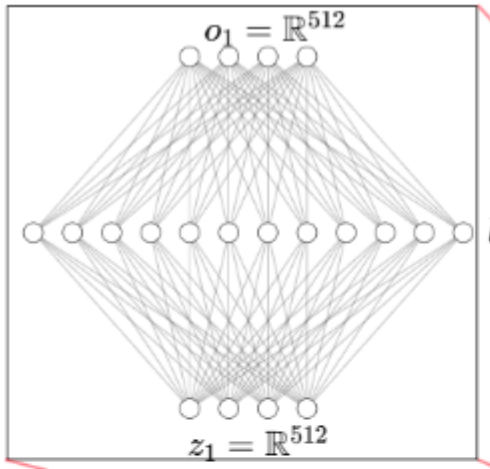
- The output of the feed forward neural network is fed to a feed forward neural network which non-linearity through the use of activation functions (typically ReLU) between two linear transformations.

- This non-linear transformation allows the model to capture complex relationships and features that cannot be represented solely through linear operations in the self-attention mechanism.



Source: CS7015 Deep Learning, Dept. of CSE, IIT Madras

Feed Forward Network



Exactly! The term **position-wise** in this context means that each position (or token) in the sequence is processed independently by the same feed-forward neural network (FCNN).

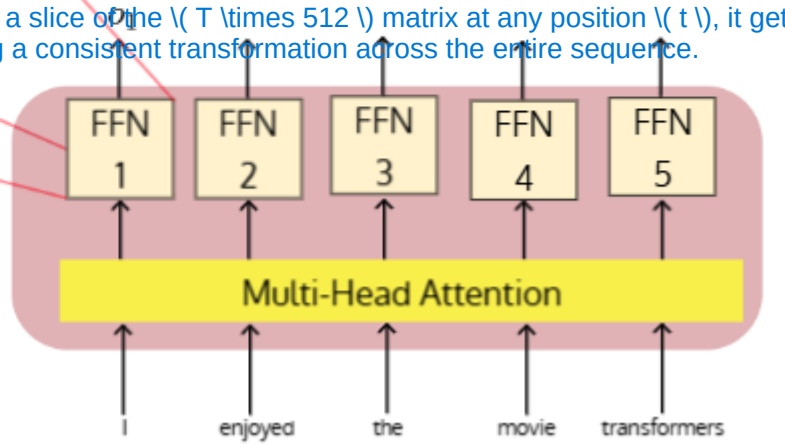
To clarify:

1. **Input Shape**: After the multi-head attention layer, you have an output matrix of shape \(T \times 512 \), where \(T \) is the sequence length (number of tokens), and 512 is the hidden dimension.

2. **Position-Wise Feed-Forward Network (FCNN)**: This network takes each \(512 \)-dimensional vector (corresponding to each position \(t \)) and applies the same two-layer feed-forward network to it.
   - The feed-forward network first projects the 512 dimensions up to 2048, applies a ReLU activation, and then projects back down to 512 dimensions.
   - This is applied to each \(T \) position individually, but **with the same network parameters** (weights and biases).

So, the feed-forward network operates **independently on each position**, but it uses the **same weights** for each token in the sequence. This shared network across all positions allows the model to handle sequences of varying lengths while applying the same learned transformation at each position.

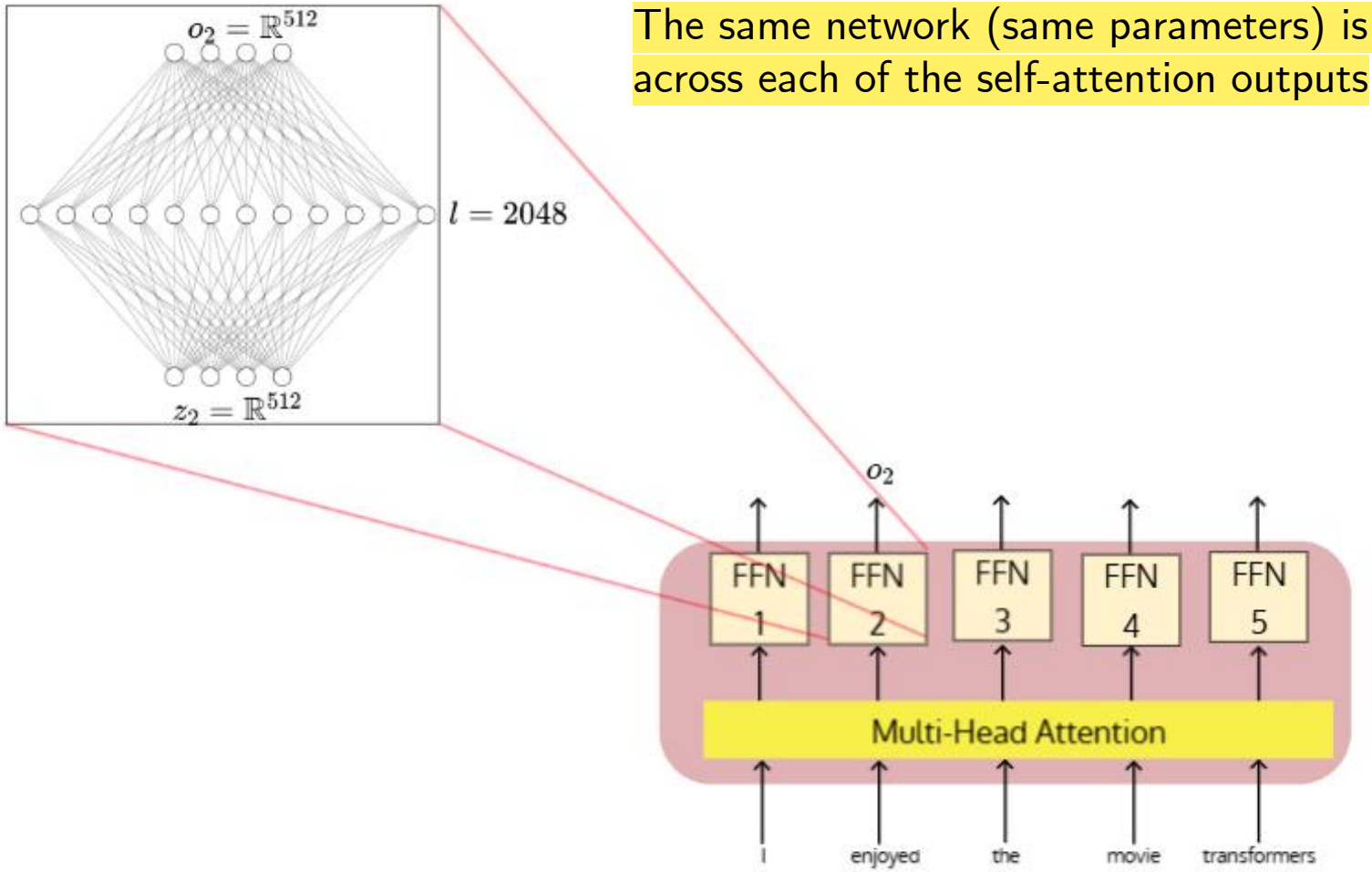
In other words, if you take a slice of the \(T \times 512 \) matrix at any position \(t \), it gets processed by the **same FCNN** as every other position, maintaining a consistent transformation across the entire sequence.



Source: CS7015 Deep Learning, Dept. of CSE, IIT Madras

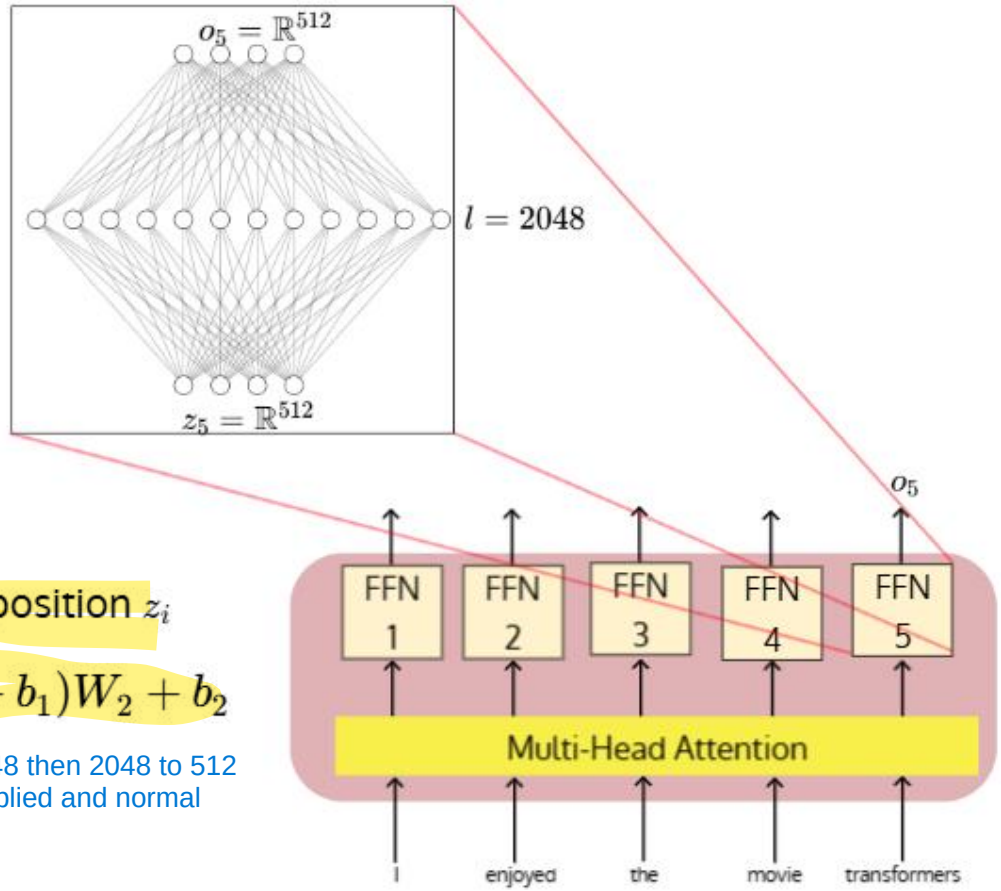
Feed Forward Network

The same network (same parameters) is used across each of the self-attention outputs



Source: CS7015 Deep Learning, Dept. of CSE, IIT Madras

Feed Forward Network



Identical network for each position z_i

$$\text{FFN}(z) = max(0, W_1 z + b_1) W_2 + b_2$$

since two way one is 512 to 2048 then 2048 to 512 and max(0,) wala is just relu applied and normal feed forward it is

Source: CS7015 Deep Learning, Dept. of CSE, IIT Madras

Encoder : No. of learnable parameters

$T = 40 \quad h = 8$

$d_k, d_q, d_v = 64$

$W_Q : 64 \times 512 =$ **32,768**

$W_K : 64 \times 512 =$ **32,768**

$W_V : 64 \times 512 =$ **32,768**

For 8 heads:

$8 * 64 \times 512 = 2,62144$

$8 * 64 \times 512 = 2,62144$

$8 * 64 \times 512 = 2,62144$

7,86,432

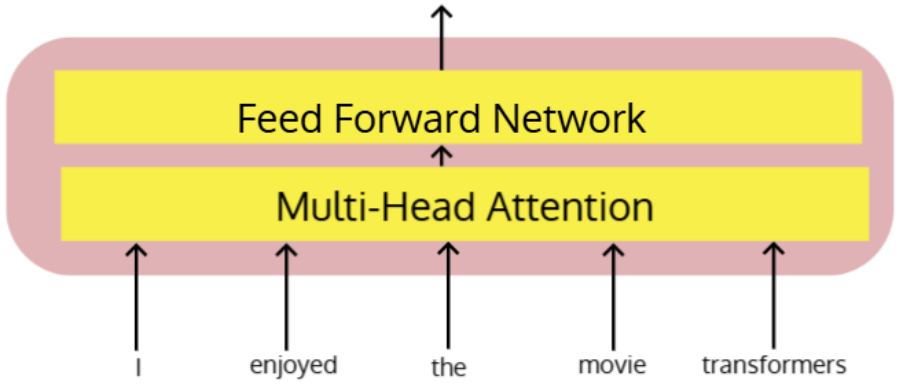
$W_O : 512 \times 512 = 262,144$

1,048,576

$FFN = (512 * 2048) + 2048 + (2048 * 512) + 512 =$ **2,099,712**

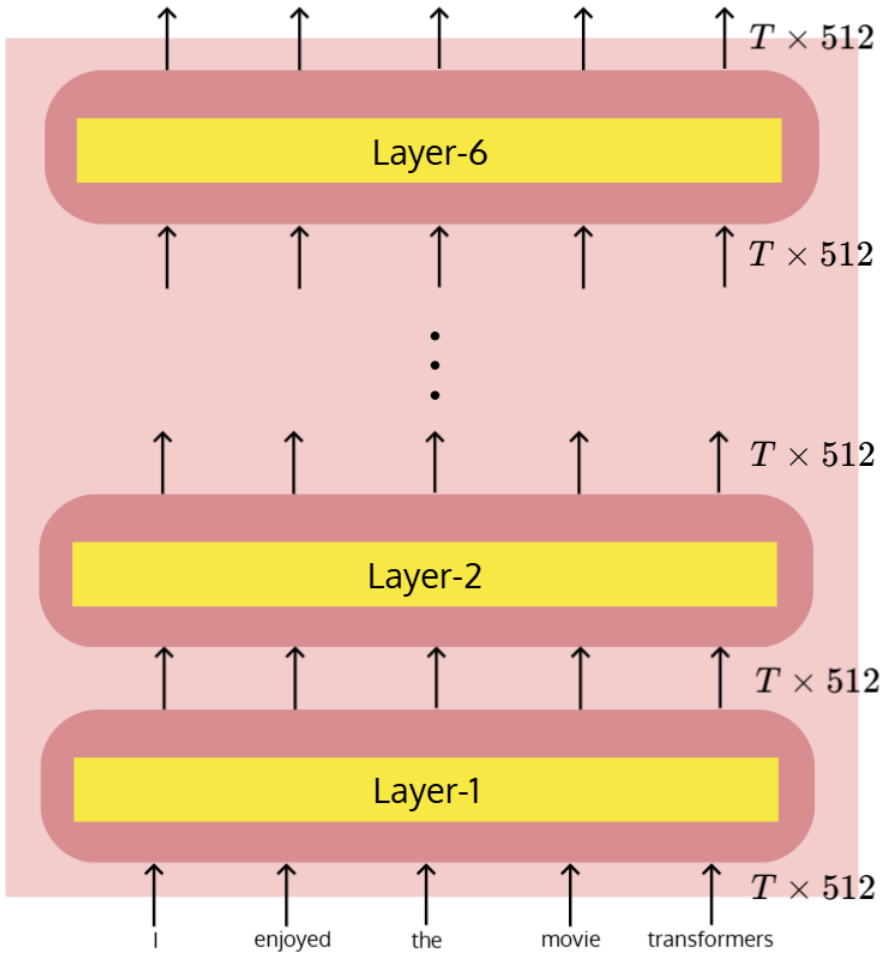
- Multi-Head Attention: **1,048,576**
- Feed-Forward Network: **2,099,7122**
- **Total: 1,048,576+2,099,712 = 3,148,288**

In total, approx. 3.1M parameters per encoder layer!!



Source: CS7015 Deep Learning, Dept. of CSE, IIT Madras

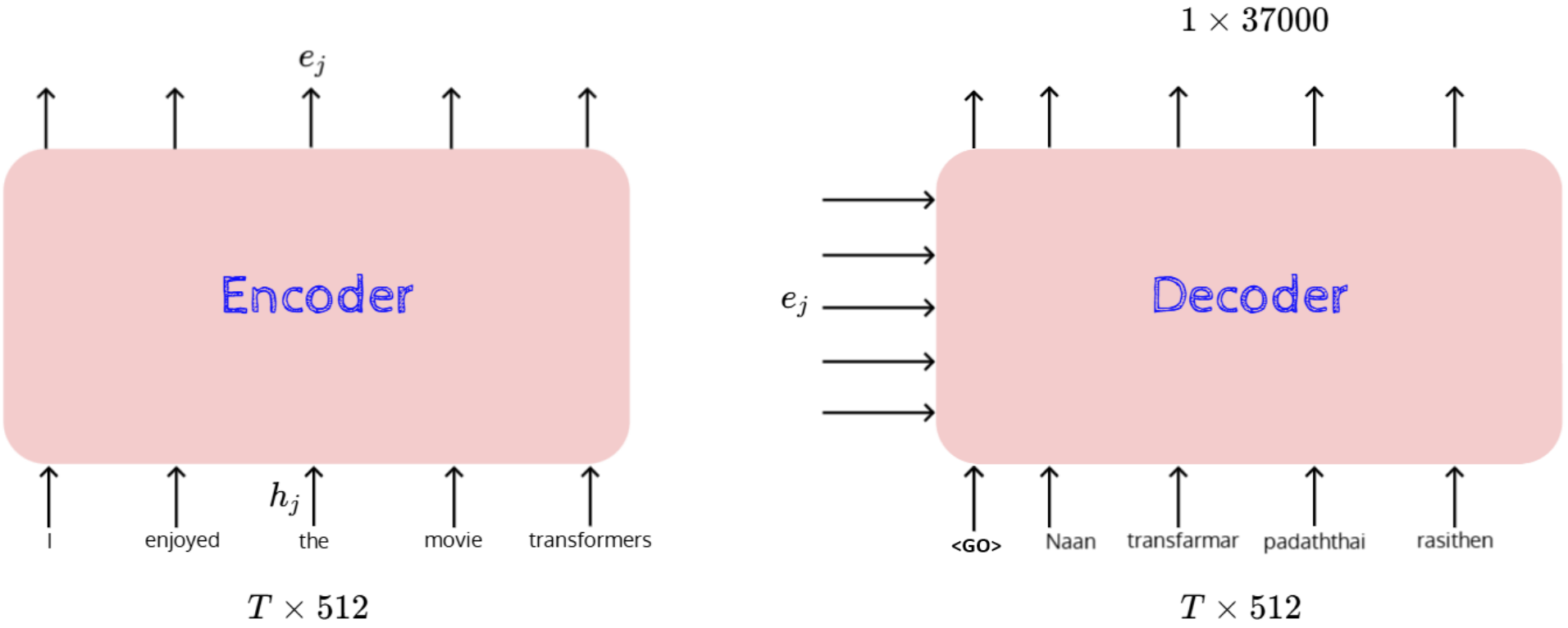
Encoder Stack



- The encoder is composed of N identical layers and each layer is composed of 2 sub-layers.
- **Number of Parameters: 6 x 3.1 M = 18.6 M**
- The computation is parallelized in the horizontal direction (i.e., within a training sample) of the encoder stack, NOT along the vertical direction (as the encoder in the higher-layer is dependent on the lower-layer encoder)

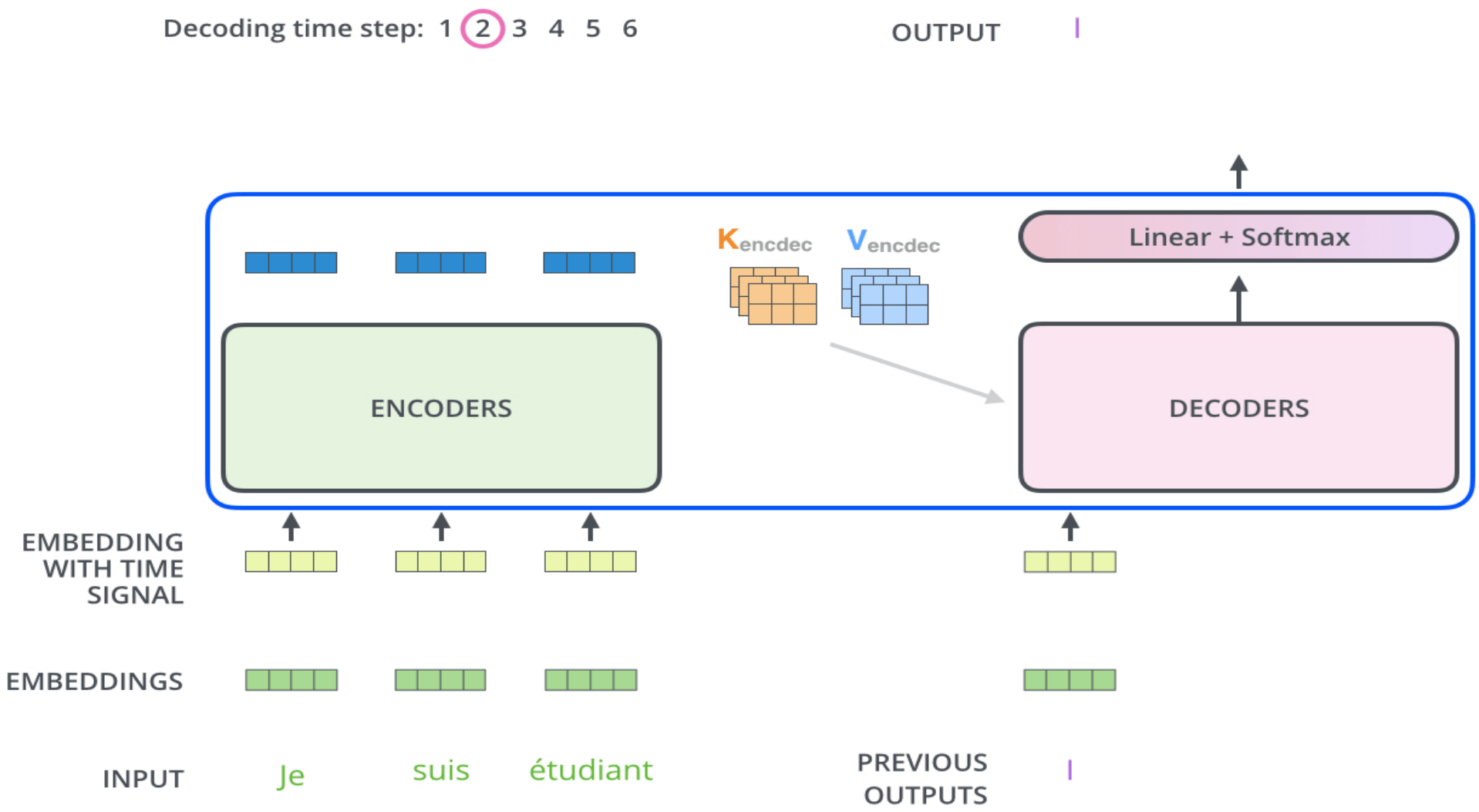
Source: CS7015 Deep Learning, Dept. of CSE, IIT Madras

Decoder



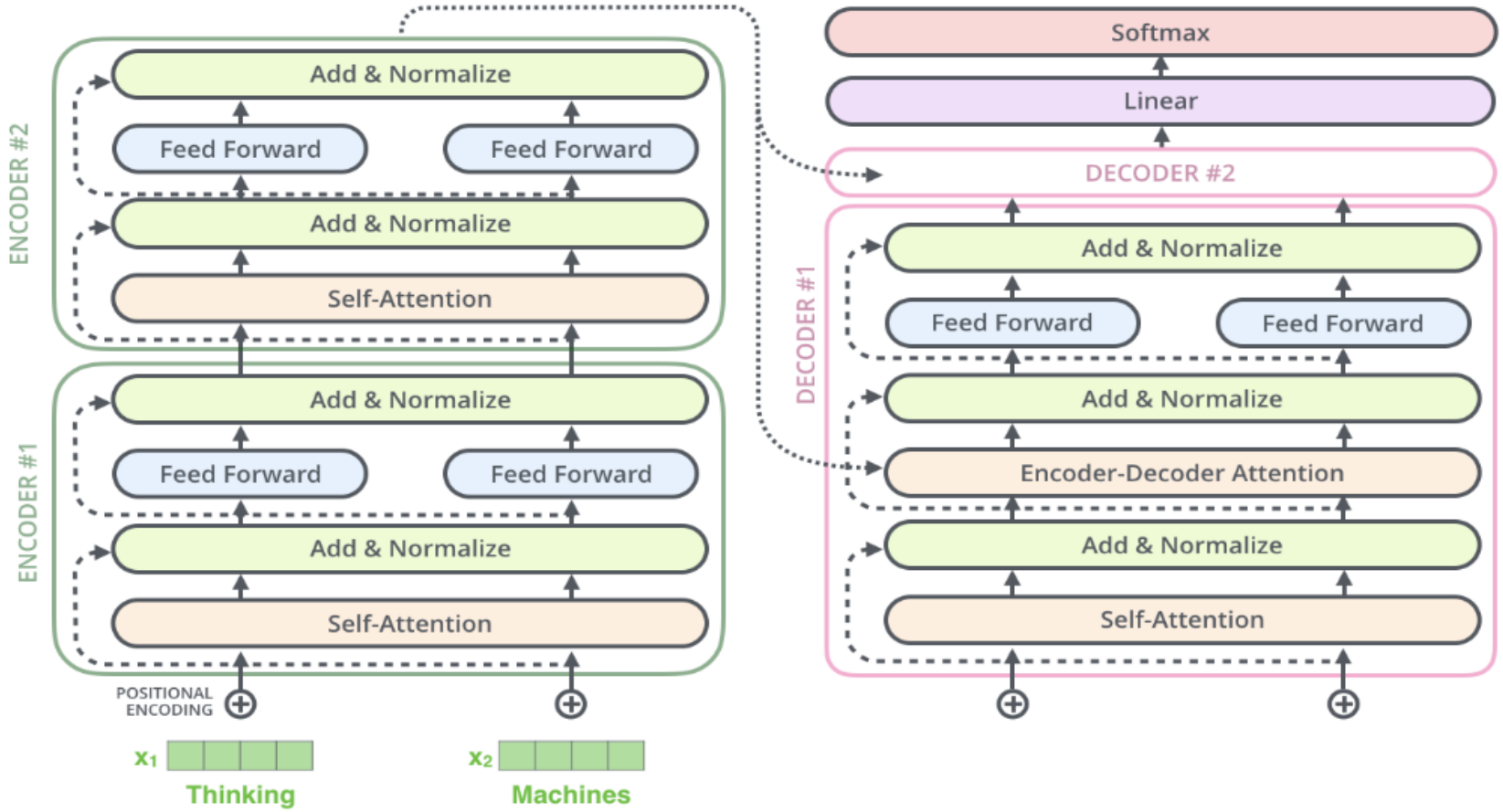
Source: CS7015 Deep Learning, Dept. of CSE, IIT Madras

Decoder


Decoding time step: 1 2 3 4 5 6
OUTPUT
I
Linear + Softmax
Kencdec
Vencdec
ENCODERS
DECODERS
EMBEDDING WITH TIME SIGNAL
EMBEDDINGS
INPUT
Je
suis
étudiant
PREVIOUS OUTPUTS
I

Decoder

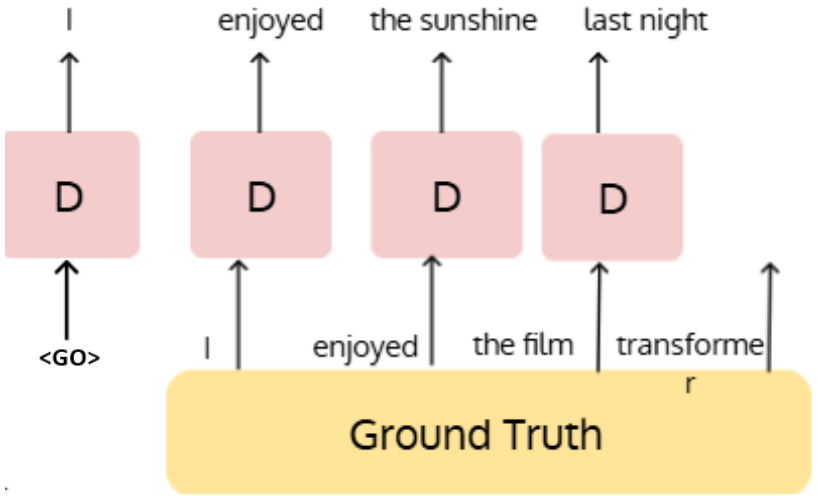
Example of how a 2-layer encoder stack is connected to a 2-layer decoder stack



The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time. (jalammar.github.io)

Decoder –Teacher Forcing

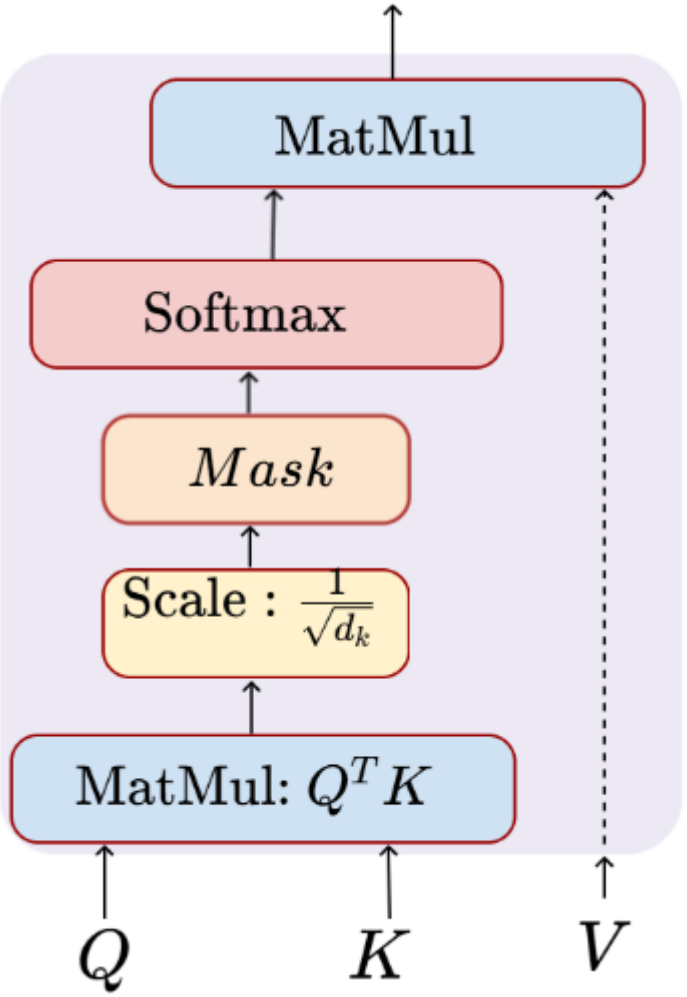
- Why the target sentence is being fed as one of the inputs to the decoder?
  - Usually, we use only the decoder's previous prediction as input to make the next prediction in the sequence.
  - However, the drawback of this approach is that if the first prediction goes wrong then there is a high chance the rest of the predictions will go wrong (because of conditional probability), leading to accumulation of errors.
  - Of course, the algorithm has to fix this as training progresses. But, it takes a long time to training the model



Consider the target language as English. The decoder "D" predicts the target word one-by-one. Notice anything peculiar here?

Source: CS7015 Deep Learning, Dept. of CSE, IIT Madras

Masked (Self) Attention



- In the encoder self-attention, we computed the query, key, and value vectors by multiplying the word embeddings (h_1,h_2,.....,h_T) with the transformation matrices.
- The same is repeated in the decoder layers. This time the h_1,h_2,.....,h_T are the word embeddings of target sentence.
- But, in the decoder masking is implemented to enforce the teacher-forcing approach during training.
- Note:
  - We can't use teacher-forcing during inference. Instead, the decoder act as a auto-regressor.
  - Encoder block also uses masking in attention sublayer in practice to mask the padded tokens in sequences having length < T

Source: CS7015 Deep Learning, Dept. of CSE, IIT Madras

Masked (Self) Attention

QK^T = T

q1•k1	$-\infty$	$-\infty$	$-\infty$	$-\infty$
q2•k1	q2•k2	$-\infty$	$-\infty$	$-\infty$
q3•k1	q3•k2	q3•k3	$-\infty$	$-\infty$
q4•k1	q4•k2	q4•k3	q4•k4	$-\infty$
q5•k1	q5•k2	q5•k3	q5•k4	q5•k5

T

- Masking is done by inserting negative infinite at the respective positions.

Example:

$$QK^T = \begin{bmatrix} 0.9 & 0.2 & 0.1 \\ 0.3 & 0.8 & 0.4 \\ 0.2 & 0.5 & 0.7 \end{bmatrix}$$

$$\begin{bmatrix} 0.9 & 0.2 & 0.1 \\ 0.3 & 0.8 & 0.4 \\ 0.2 & 0.5 & 0.7 \end{bmatrix} + \begin{bmatrix} 0 & -\infty & -\infty \\ 0 & 0 & -\infty \\ 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0.9 & -\infty & -\infty \\ 0.3 & 0.8 & -\infty \\ 0.2 & 0.5 & 0.7 \end{bmatrix}$$

After Softmax:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0.31 & 0.69 & 0 \\ 0.22 & 0.34 & 0.44 \end{bmatrix}$$

Source: CS7015 Deep Learning, Dept. of CSE, IIT Madras

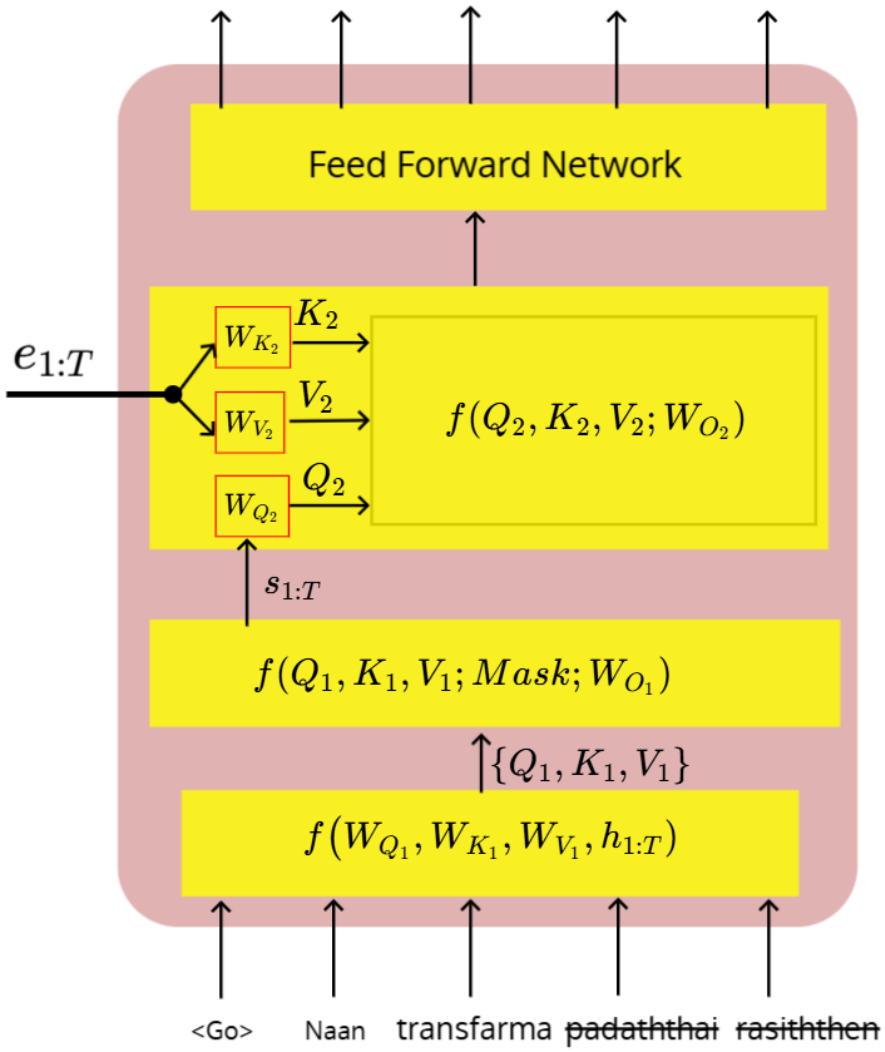
Multi-head Cross Attention

- Now we have the vectors S $=\{s_1,s_2,\cdots,s_T\}$ coming from the self-attention layer of **decoder**.
- We have also a set of vector E $=\{e_1,e_2,\cdots,e_T\}$ coming from top layer of **encoder stack** that is shared with all layers of **decoder stack**.
- Again, we need to create query, key and value vectors by applying the linear transformation matrices W_{Q2},W_{K2}, &W_{V2} on these vectors s_i and e_j.

imp here it is multi head cross attention so q is from decoder layer and k and v from the encoder layer

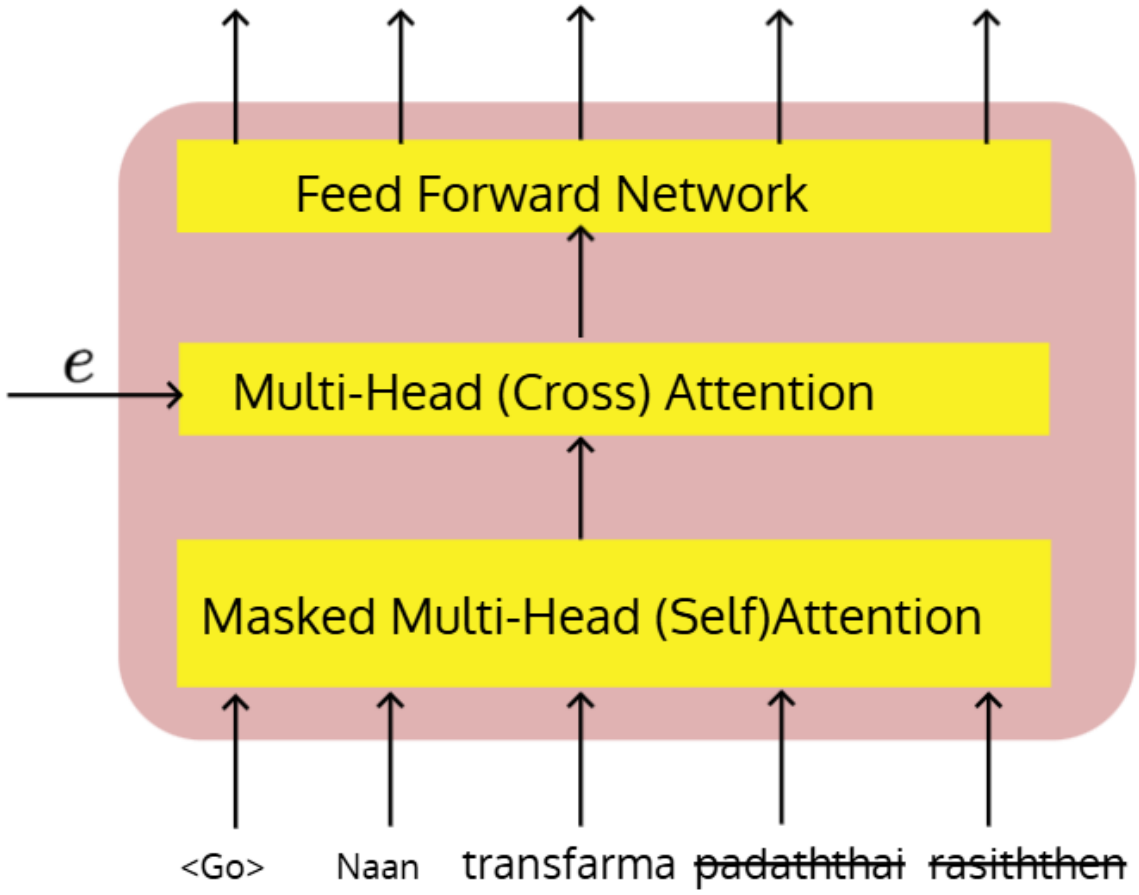
$$Q_2 = W_{Q_2} S \quad K_2 = W_{K_2} E \quad V_2 = W_{V_2} E$$

$$Z = softmax(Q_2^T K_2) V_2^T$$



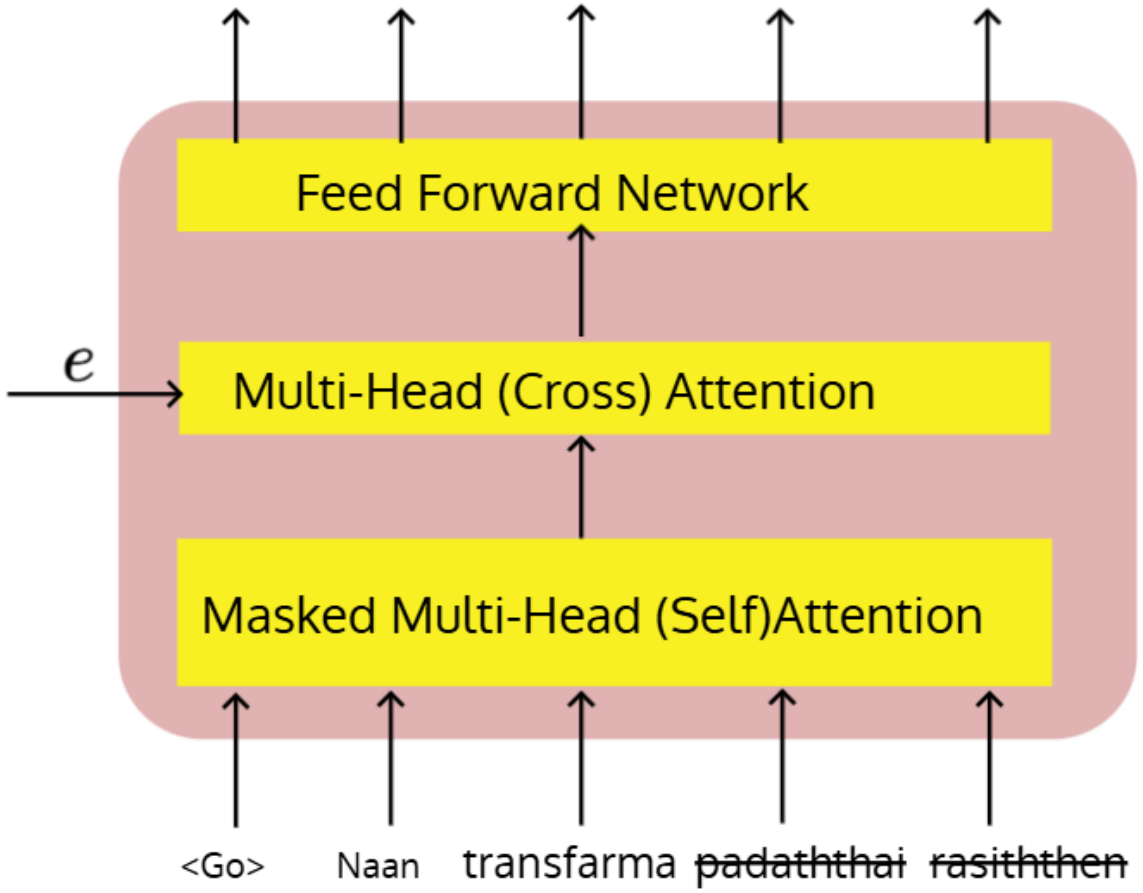
Source: CS7015 Deep Learning, Dept. of CSE, IIT Madras

Bottom most Decoder Layer



Source: CS7015 Deep Learning, Dept. of CSE, IIT Madras

Decoder Layer: Number of Parameters



- The computation of parameters for self attention and cross attention layers in the decoder are similar to the computation of parameters of the self attention layers in the encoder.

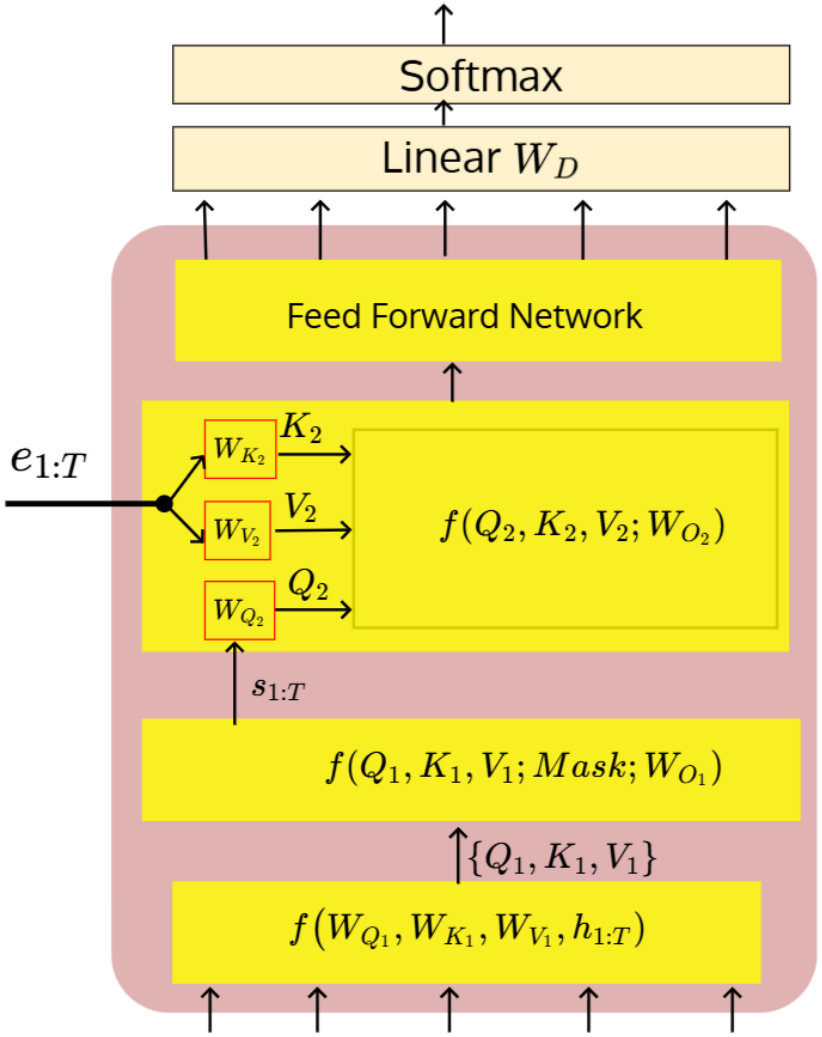
- Masked Multi-Head (Self) Attention: **1,048,576**
- Multi-Head (Cross) Attention: **1,048,576**
- Feed-Forward Network: **2,099,7122**
- **Total: (1,048,576)*2 +2,099,712 = 4,196,864**

In total, approx. 4.1M parameters per decoder layer!!

Approx., 4.1*6 = 24.6M parameters for decoder stack!!

Source: CS7015 Deep Learning, Dept. of CSE, IIT Madras

Decoder Output



- The output from the top most decoder layer is linearly transformed by the matrix W_D of size 512×|V| where |V| is the size of the vocabulary.
- The probability distribution for the predicted words is obtained by applying softmax function.
- This alone contributes about 19 million parameters of the total 65 million parameters of the architecture.

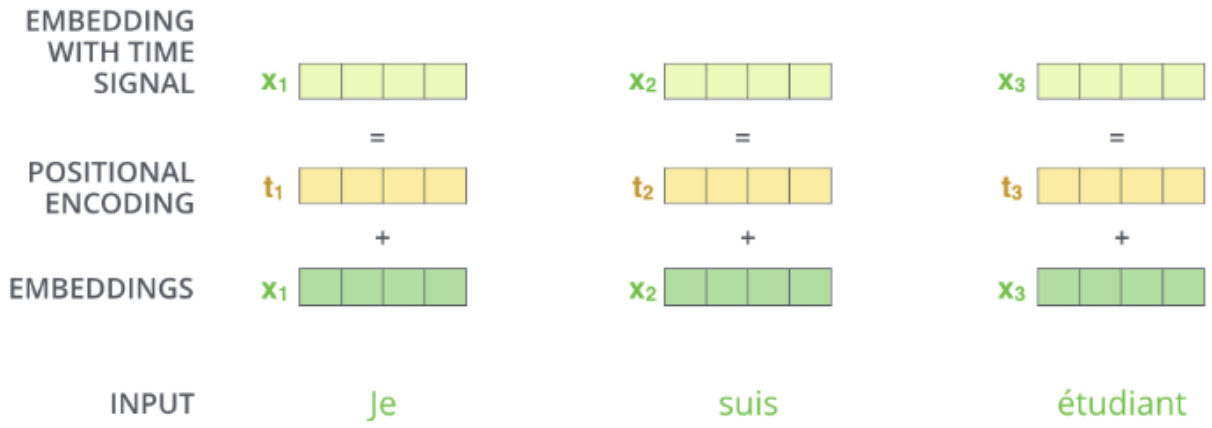
Source: CS7015 Deep Learning, Dept. of CSE, IIT Madras

Positional Encoding

- Order of the sequence conveys important information for machine translation tasks and language modeling.
- Position encoding is a way to account for the order of words in the input sequence.
- The positional information of the input token in the sequence is added to the input embedding vectors.

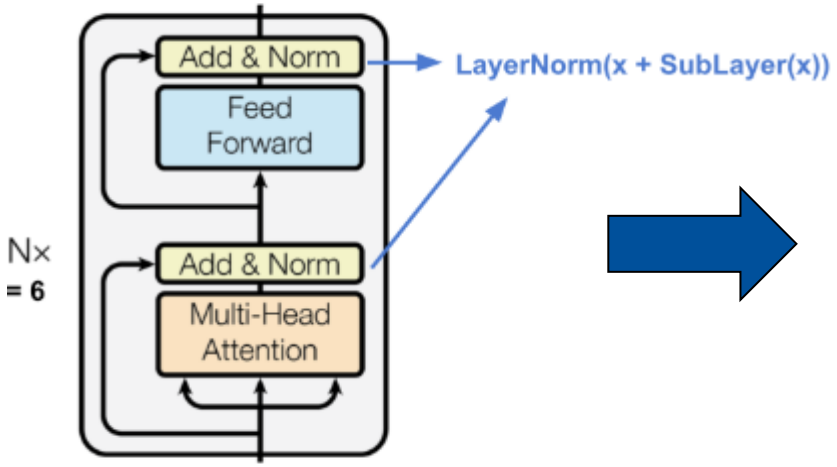
512

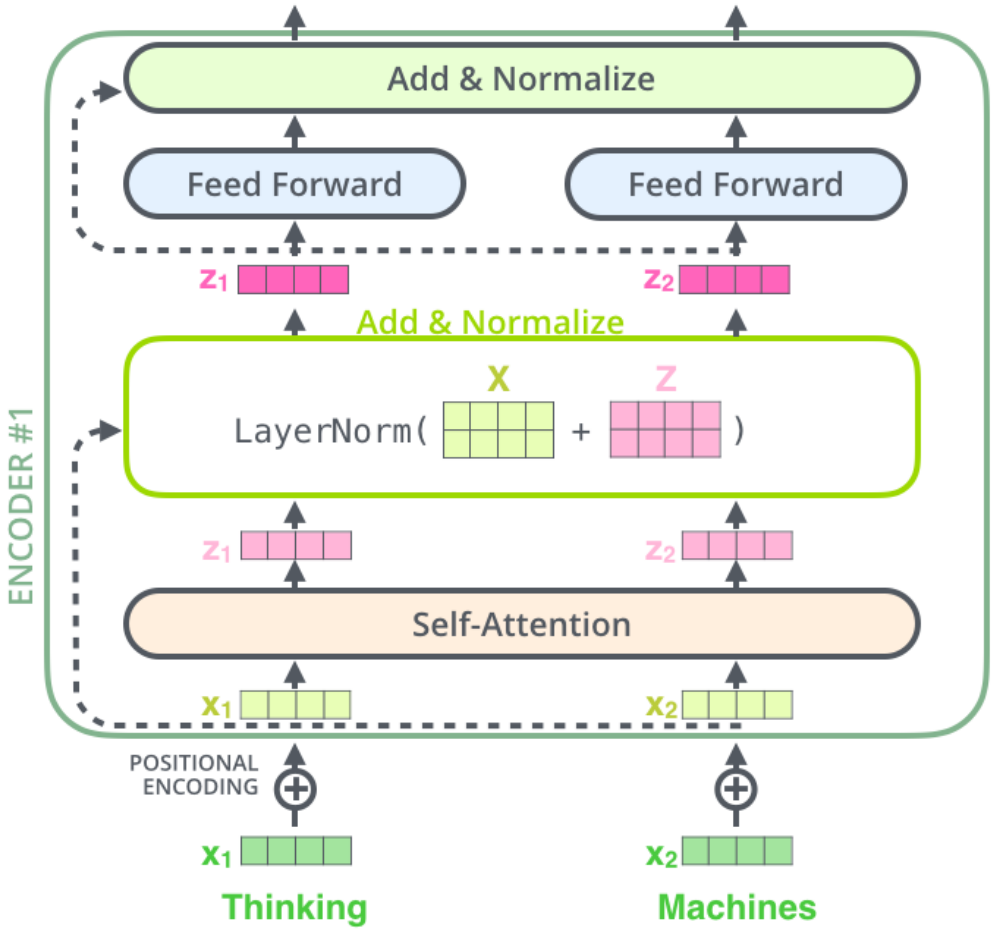
$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$



The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time. (jalammar.github.io)

Layer Normalization





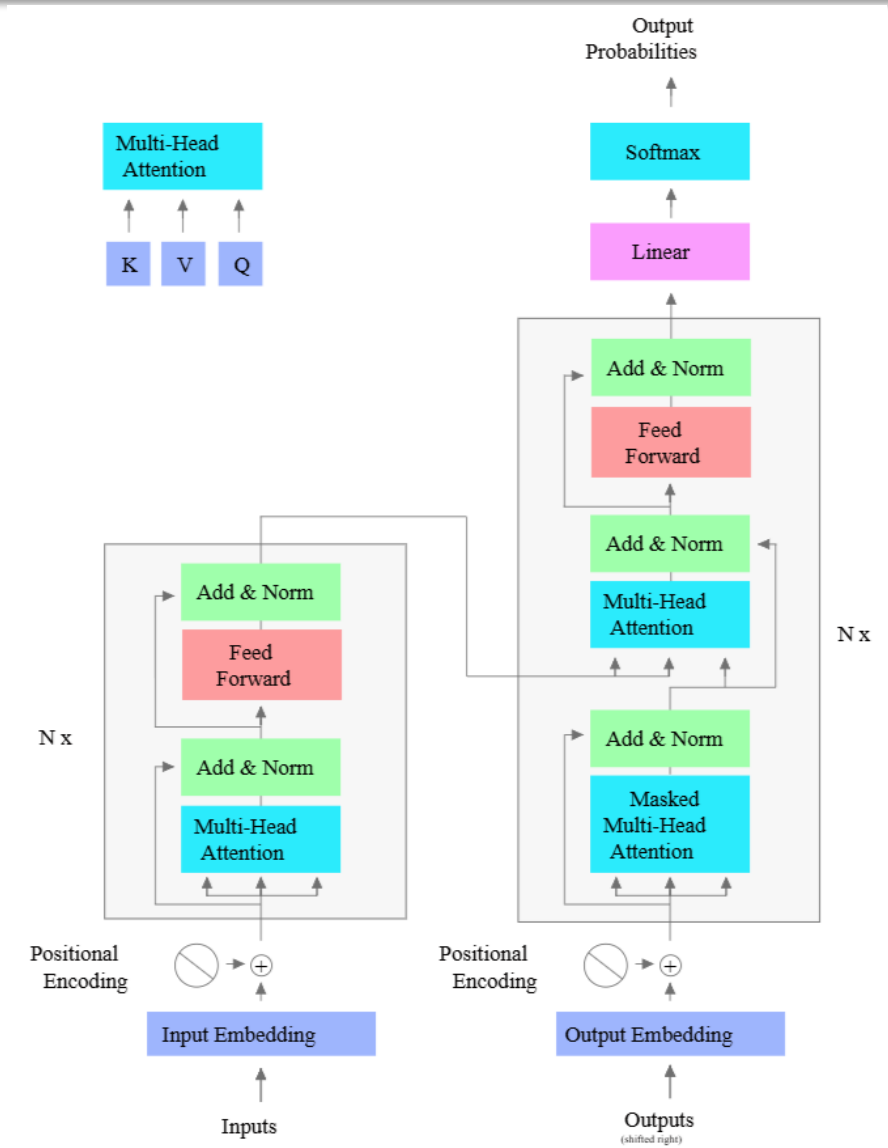
Layer Norm

statistics are calculated across all features and all elements (words), for each instance(sentence) independently.

Attention? Attention! | Lil'Log (lilianweng.github.io)

The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time. (jalammar.github.io)

The Transformer Architecture



Massive Deep Learning Language models

- **Language models**
  - estimate the probability of words appearing in a sentence, or of the sentence itself existing.
  - building blocks in a lot of NLP applications
- **Massive deep learning language models**
  - pretrained using an enormous amount of unannotated data to provide a general-purpose deep learning model.
  - Downstream users can create task-specific models with smaller annotated training datasets (transfer learning)
- **Tasks executed with BERT and GPT models:**
  - **Natural language inference**
    - enables models to determine whether a statement is true, false or undetermined based on a premise.
    - For example, if the premise is “tomatoes are sweet” and the statement is “tomatoes are fruit” it might be labelled as undetermined.
  - **Question answering**
    - model receives a question regarding text content and returns the answer in text, specifically marking the beginning and end of each answer.
  - **Text classification**
    - is used for sentiment analysis, spam filtering, news categorization

GPT (Generative Pre-Training) by Open AI Pretraining

- Unsupervised learning served as **pre-training** objective for supervised fine-tuned models
  - generative language model using unlabeled data
  - then fine-tuning the model by providing examples of specific downstream tasks like classification, sentiment analysis, textual entailment etc.
- **Semi supervised learning using 3 components**
  1. **Unsupervised Language Modelling** (Pre-training)

$$L_1(T) = \sum_i log\, P(t_i | t_{i-k}, \ldots, t_{i-1}; \theta)$$

where

k is the size of the context window, and ii) conditional probability P is modeled with the help of a neural network (NN) with parameters Θ

GPT (Generative Pre-Training) by Open AI

2. **Supervised Fine-Tuning**: maximising the likelihood of observing label y, given features or tokens x_1,...,x_n.

$$L_2(C) = \sum_{x,y} log\ P(y|x_1, \ldots, x_n)$$

where C was the labeled dataset made up of training examples.

3. **Auxiliary learning objective for supervised fine-tuning to get better generalisation and faster convergence.**

$$L_3(C) = L_2(C) + \lambda L_1(C)$$

where $L_1(C)$ was the auxiliary objective of learning language model

λ was the weight given to this secondary learning objective. λ was set to 0.5.

- Supervised fine-tuning is achieved by adding a linear and a softmax layer to the transformer model to get the task labels for downstream tasks.
- "zero-shot" framework
  - measure a model's performance having *never* been trained on the task.

GPT-n series created by OpenAI (2018 onwards)

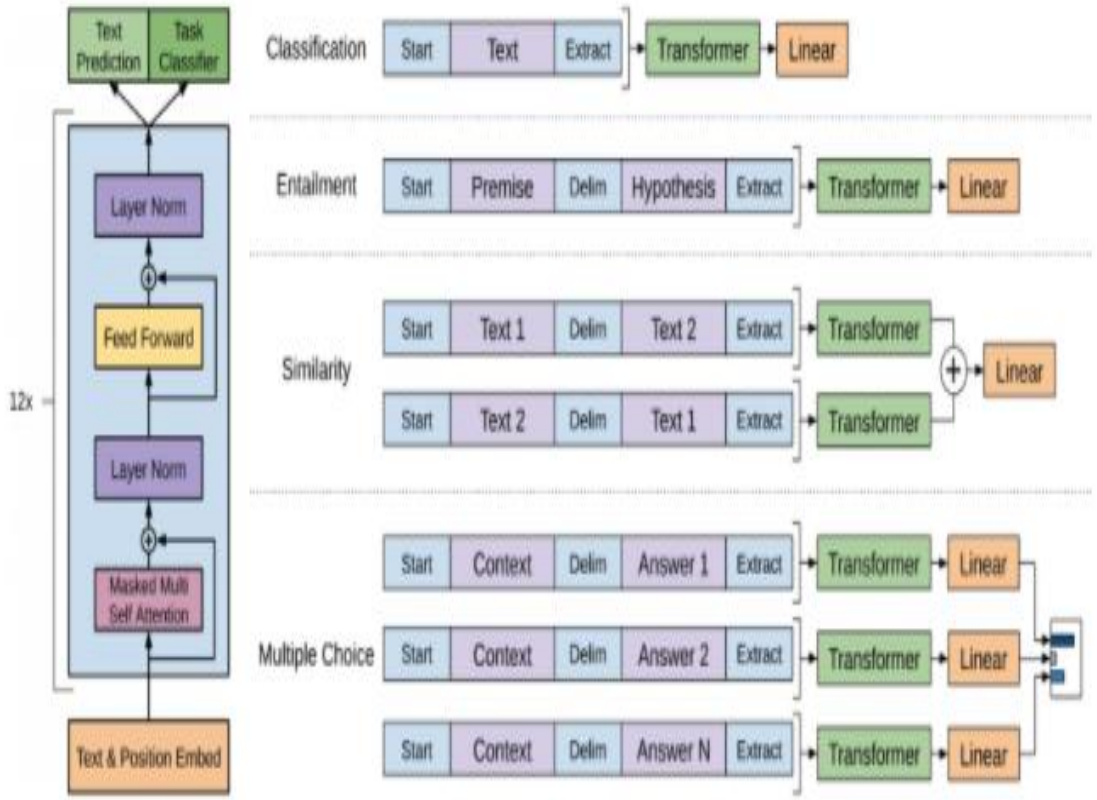


Figure 1: (**left**) Transformer architecture and training objectives used in this work. (**right**) Input transformations for fine-tuning on different tasks. We convert all structured inputs into token sequences to be processed by our pre-trained model, followed by a linear+softmax layer.

- **Generative** models are a type of statistical model that are used to generate new data points.
  - learn the underlying relationships between variables in a dataset in order to generate new data points similar to those in the dataset.
- Trained on BooksCorpus dataset contains over 7,000 unique unpublished books from a variety of genres
- 12-layer decoder-only transformer with masked self-attention heads
- GPT-2
  - Application - generate long passages of coherent text
  - https://transformer.huggingface.co/doc/gpt2-large

GPT-3 (2020)

- step to **A**rtificial **G**eneral **I**ntelligence(AGI)

"I am open to the idea that a worm with 302 neurons is conscious, so I am open to the idea that GPT-3 with 175 billion parameters is conscious too." — David Chalmers

- 3rd-gen language prediction model in capacity of **175 billion** parameters.
- trained with 499 Billion tokens
- trained using next word prediction
- Context window size was increased from 1024 for GPT-2 to 2048 tokens for GPT-3.
- Size of word embeddings was increased to 12888 for GPT-3 from 1600 for GPT-2.
- To train models of different sizes, the batch size is increased according to number of parameters, while the learning rate is decreased accordingly.

Model Name	n_{params}	n_{layers}	d_{model}	n_{heads}	d_{head}	Batch Size	Learning Rate
GPT-3 Small	125M	12	768	12	64	0.5M	6.0×10^{-4}
GPT-3 Medium	350M	24	1024	16	64	0.5M	3.0×10^{-4}
GPT-3 Large	760M	24	1536	16	96	0.5M	2.5×10^{-4}
GPT-3 XL	1.3B	24	2048	24	128	1M	2.0×10^{-4}
GPT-3 2.7B	2.7B	32	2560	32	80	1M	1.6×10^{-4}
GPT-3 6.7B	6.7B	32	4096	32	128	2M	1.2×10^{-4}
GPT-3 13B	13.0B	40	5140	40	128	2M	1.0×10^{-4}
GPT-3 175B or "GPT-3"	175.0B	96	12288	96	128	3.2M	0.6×10^{-4}

GPT-3

zero-shot task transfer OR meta learning

- The model is supposed to understand the task based on the examples and instruction.
- For English to French translation task, the model was given an English sentence followed by the word French and a prompt (:)

Zero-shot

The model predicts the answer given only a natural language discription of the task. No gradient updates are performed.

```
Translate English to French:        ← task description
cheese =>                           ← prompt
```

One-shot

In addition to the task description, the model sees a single example of the task. No gradient updates are performed.

```
Translate English to French:        ← task description
sea otter => loutre de mer          ← example
cheese =>                           ← prompt
```

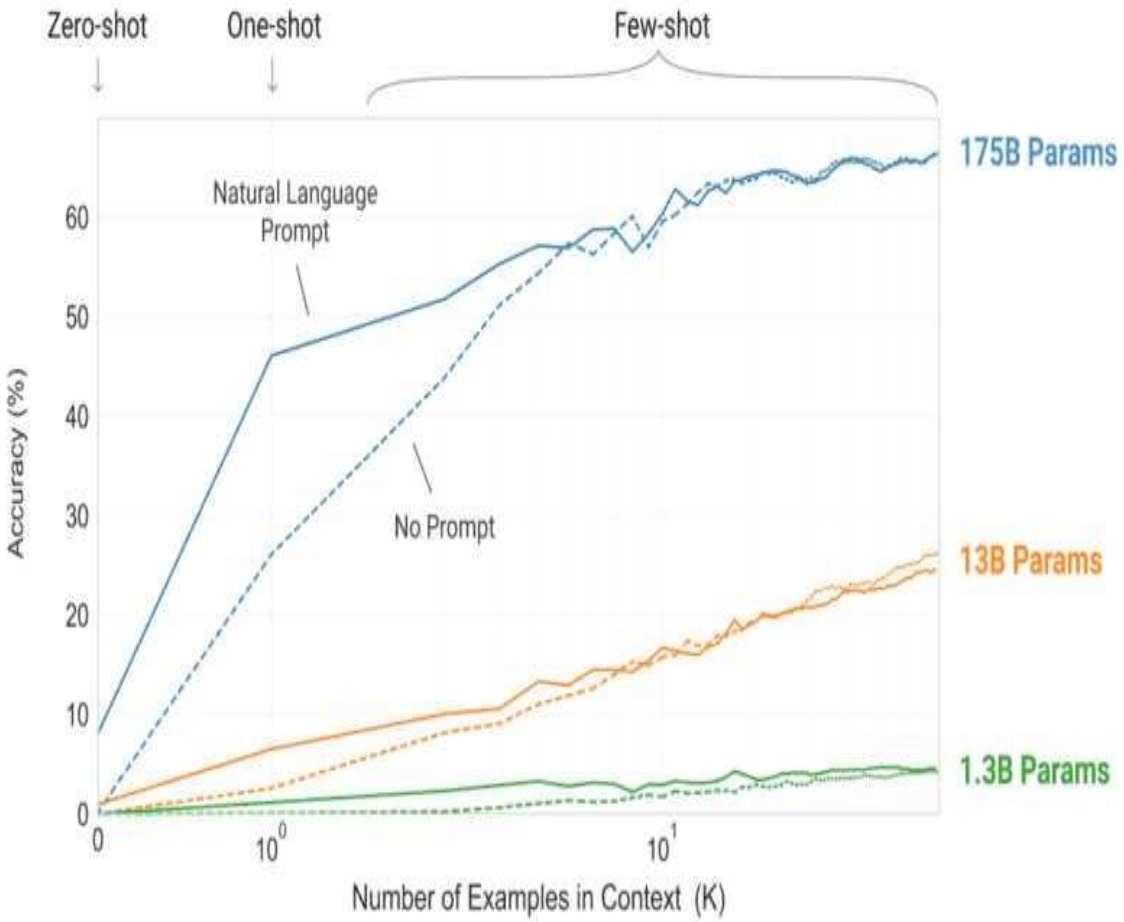
Few-shot

In addition to the task description, the model sees a few examples of the task. No gradient updates are performed.

```
Translate English to French:        ← task description
sea otter => loutre de mer          ← examples
peppermint => menthe poivrée
plush girafe => girafe peluche
cheese =>                           ← prompt
```

GPT-3 Task Agnostic Model

- **LIMITATIONS OF GPT-3**
- GPT-3 can perform a wide variety of operations such as compose prose, write code and fiction, business articles
- Does not have any internal representation of what these words even mean. misses the *semantically grounded model* of the topics on which it discusses.
- If the model is faced with data that is not in a similar form or is unavailable from the Internet's corpus of existing text that was used initially in the training phase, then the language generated is a loss.
- Expensive and complex inferencing due to hefty architecture, less interpretability of the language, and uncertainty around what helps the model achieve its few-shot learning behavior.
- The text generated carries bias of the language it is initially trained on.
- The articles, blogs, memos generated by GPT-3 may face gender, ethnicity, race, or religious bias.
- model is capable of producing high-quality text, sometimes looses coherence with data while generating long sentences and thus may repeat sequences of text again and again in a paragraph.

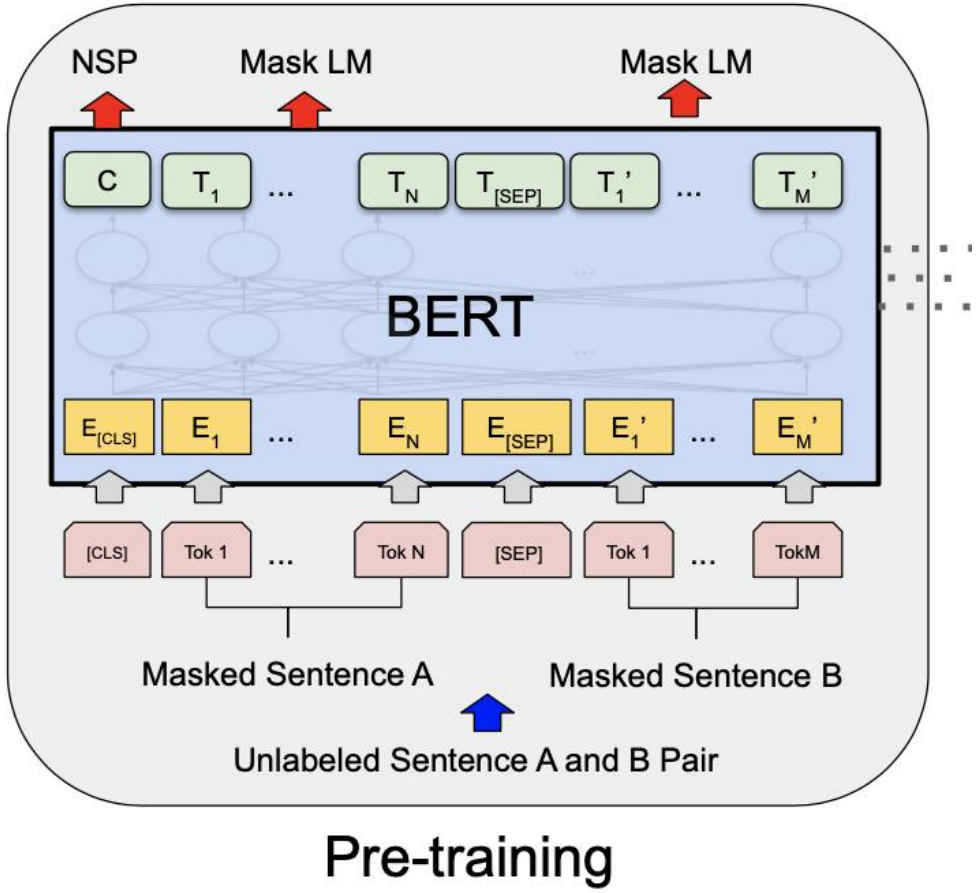


BERT (Bidirectional Encoder Representations from Transformers) by google

- "BERT stands for Bidirectional Encoder Representations from Transformers. It is designed to pre-train deep bidirectional representations from unlabeled text by jointly conditioning on both left and right context. As a result, the pre-trained BERT model can be fine-tuned with just one additional output layer to create state-of-the-art models for a wide range of NLP tasks."
- Two variants
  - BERT Base: 12 layers (transformer blocks), 12 attention heads, and 110 million parameters
  - BERT Large: 24 layers (transformer blocks), 16 attention heads and, 340 million parameters
- **BERT is pre-trained on two NLP tasks**:
  - Masked Language Modeling
    - replace 15% of the input sequence with [MASK] and model learns to detect the masked word
  - Next Sentence Prediction
    - two sentences A and B are separated with the special token [SEP] and are formed in such a way that 50% of the time B is the actual next sentence and 50% of the time is a random sentence.

BERT (Bidirectional Encoder Representations from Transformers) by google

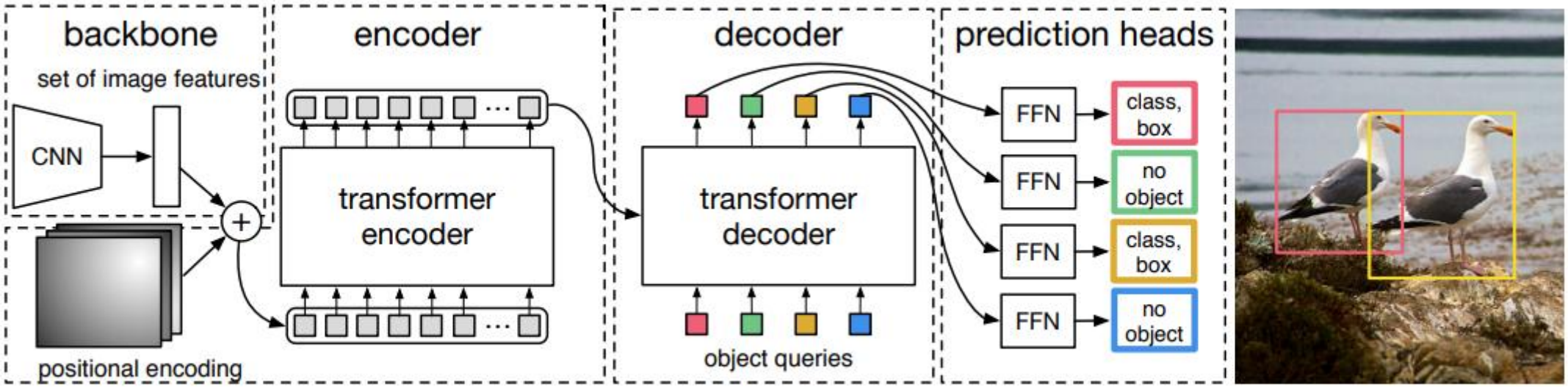
- every input embedding is a combination of 3 embeddings:
  - Position Embeddings: captures "sequence" or "order" information
  - Segment Embeddings: can also take sentence pairs as inputs for tasks (Question-Answering)
  - Token Embeddings: learned for the specific token from the WordPiece token vocabulary
- Output is an embedding since it has only encoder and no decoder



Pre-training

Transformers in Computer Vision

DEtection TRansformer (DETR)

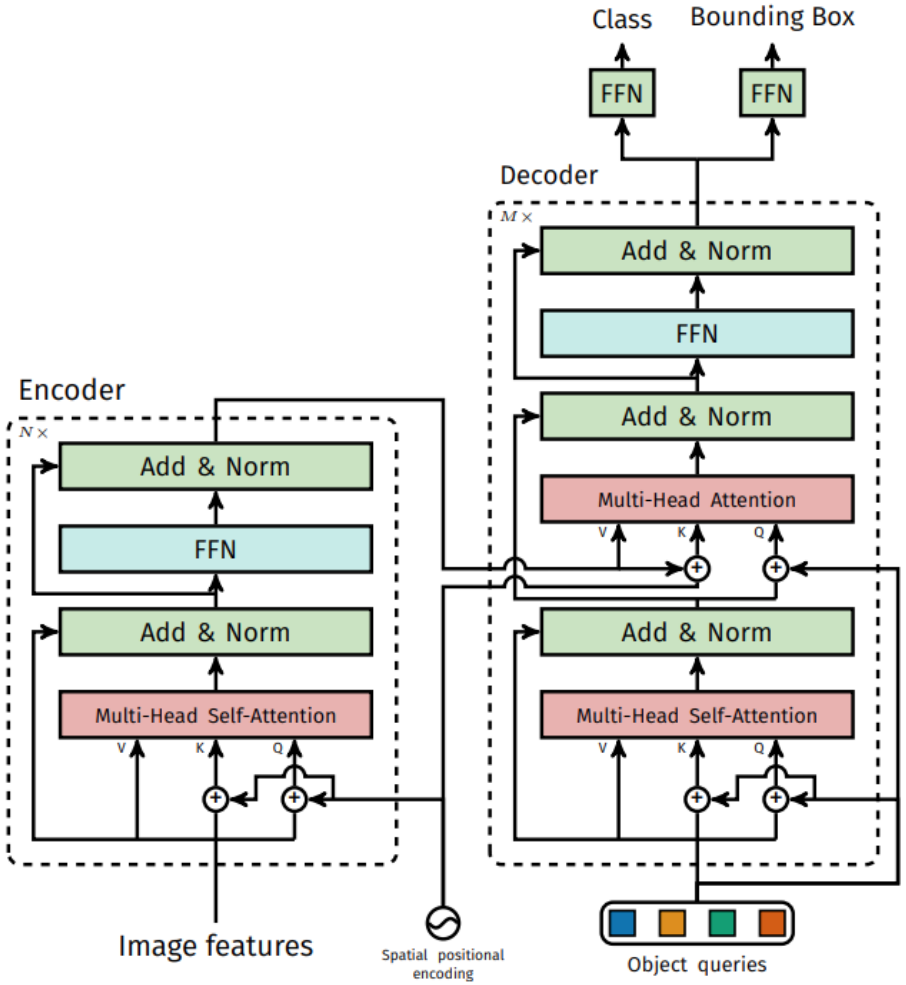


https://github.com/facebookresearch/detr

Carion, Nicolas, et al. "End-to-end object detection with transformers." *European conference on computer vision*. Springer, Cham, 2020.

Transformers in Computer Vision

DEtection TRansformer (DETR)



https://github.com/facebookresearch/detr

Carion, Nicolas, et al. "End-to-end object detection with transformers." *European conference on computer vision*. Springer, Cham, 2020.

Transformers in Computer Vision

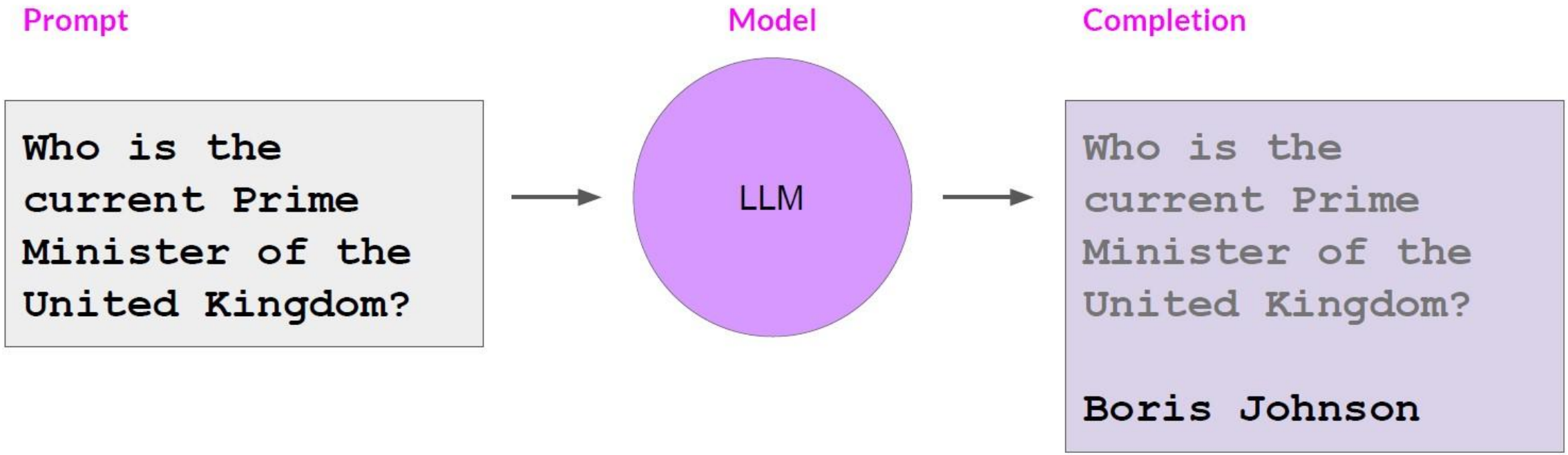
DEtection TRansformer (DETR): Results on COCO 2017 dataset (AP = Average Precision)

Model	GFLOPS/FPS	#params	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
Faster RCNN-DC5	320/16	166M	39.0	60.5	42.3	21.4	43.5	52.5
Faster RCNN-FPN	180/26	42M	40.2	61.0	43.8	24.2	43.5	52.0
Faster RCNN-R101-FPN	246/20	60M	42.0	62.5	45.9	25.2	45.6	54.6
Faster RCNN-DC5+	320/16	166M	41.1	61.4	44.3	22.9	45.9	55.0
Faster RCNN-FPN+	180/26	42M	42.0	62.1	45.5	26.6	45.4	53.4
Faster RCNN-R101-FPN+	246/20	60M	44.0	63.9	**47.8**	**27.2**	48.1	56.0
DETR	86/28	41M	42.0	62.4	44.2	20.5	45.8	61.1
DETR-DC5	187/12	41M	43.3	63.1	45.9	22.5	47.3	61.1
DETR-R101	152/20	60M	43.5	63.8	46.4	21.9	48.0	61.8
DETR-DC5-R101	253/10	60M	**44.9**	**64.7**	47.7	23.7	**49.5**	**62.3**

https://github.com/facebookresearch/detr

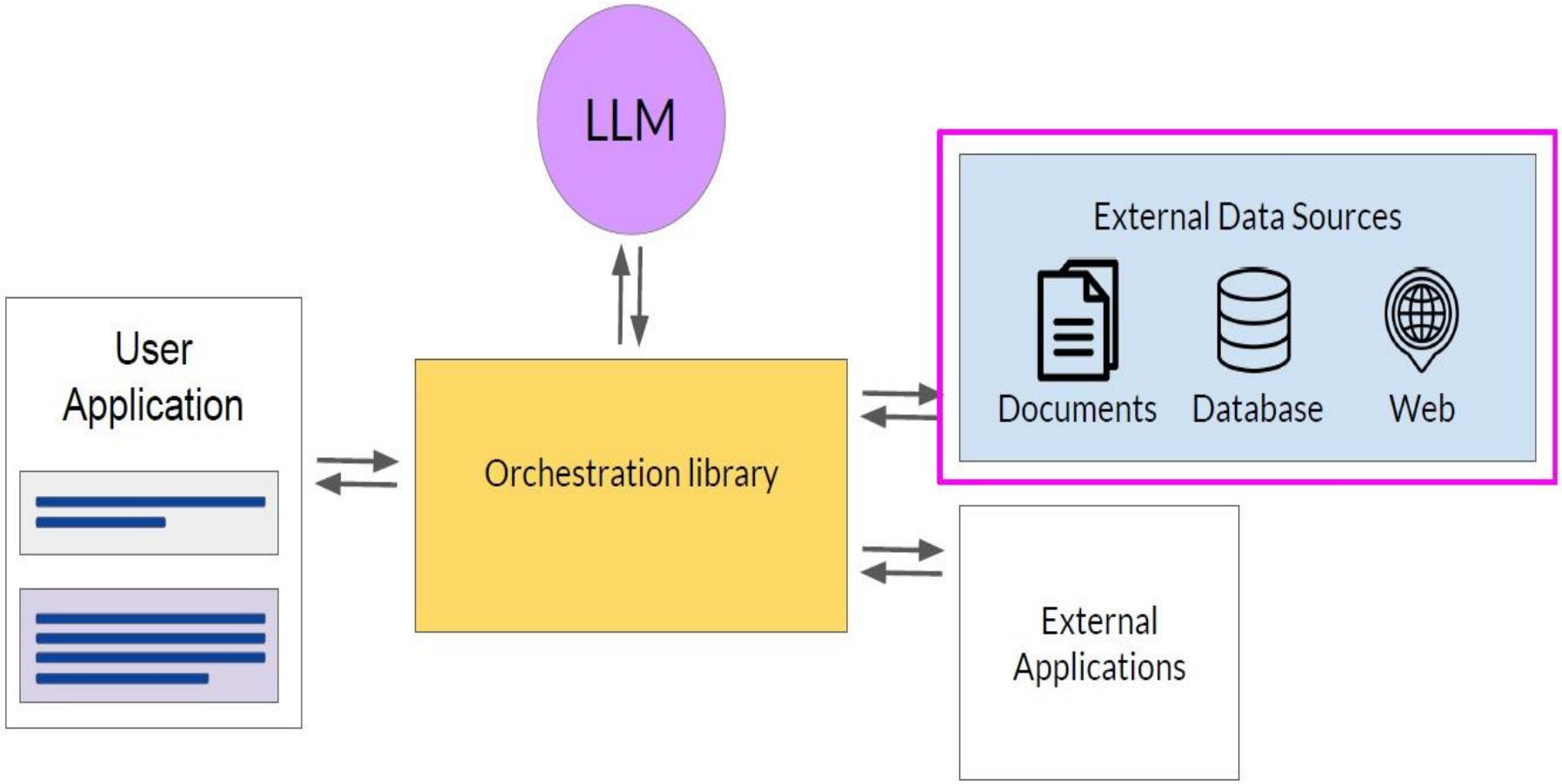
Carion, Nicolas, et al. "End-to-end object detection with transformers." *European conference on computer vision*. Springer, Cham, 2020.

Knowledge cut-offs in LLMs



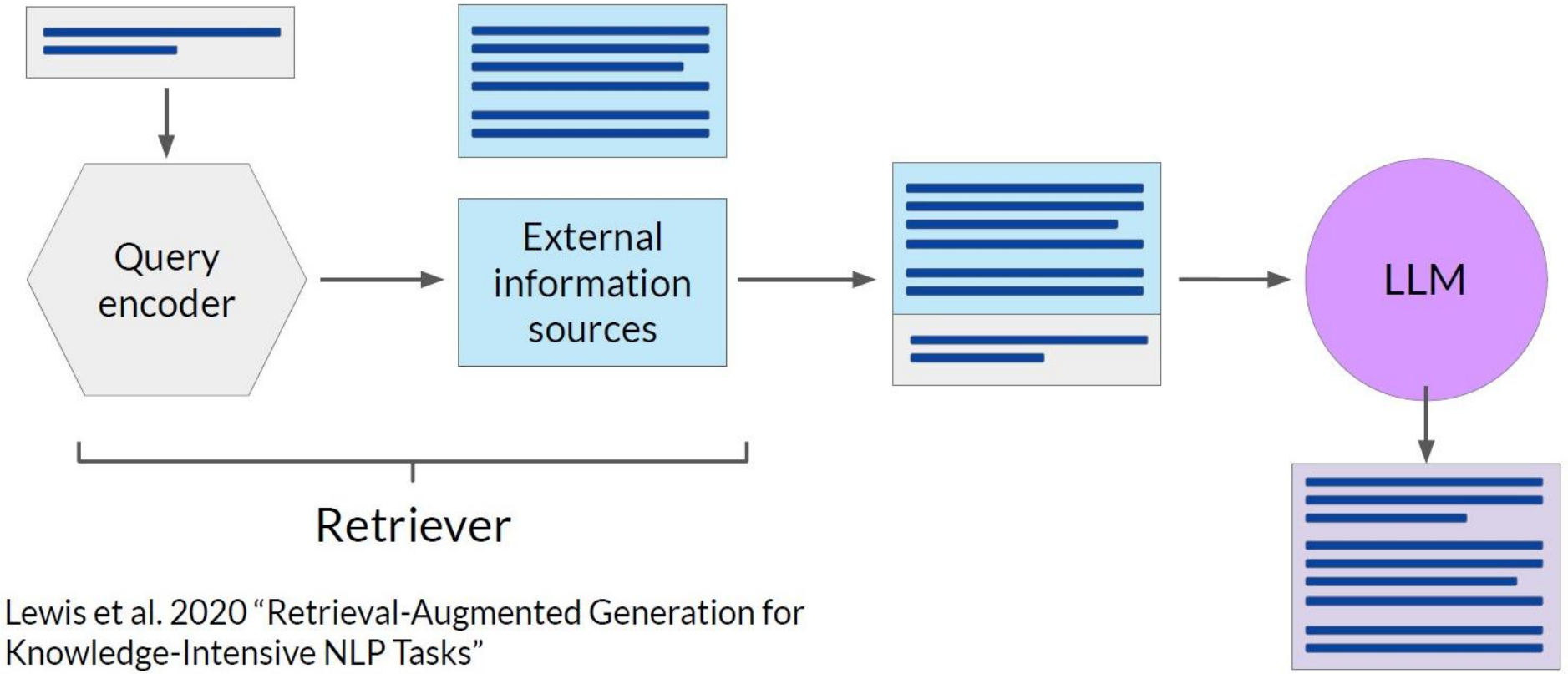
- Coursera Lecture Notes : **Generative AI with Large Language Models**

LLM-powered applications



- Coursera Lecture Notes : **Generative AI with Large Language Models**

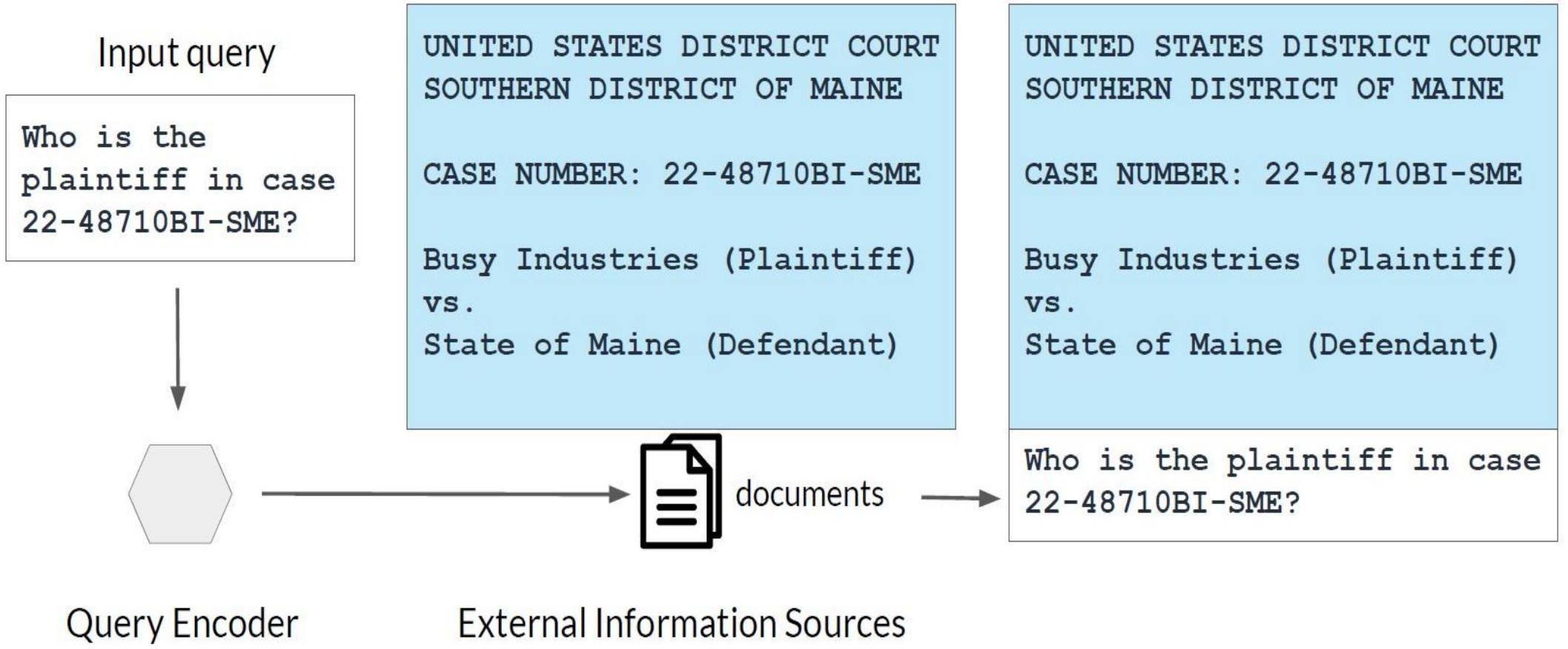
Retrieval Augmented Generation (RAG)



Lewis et al. 2020 "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks"

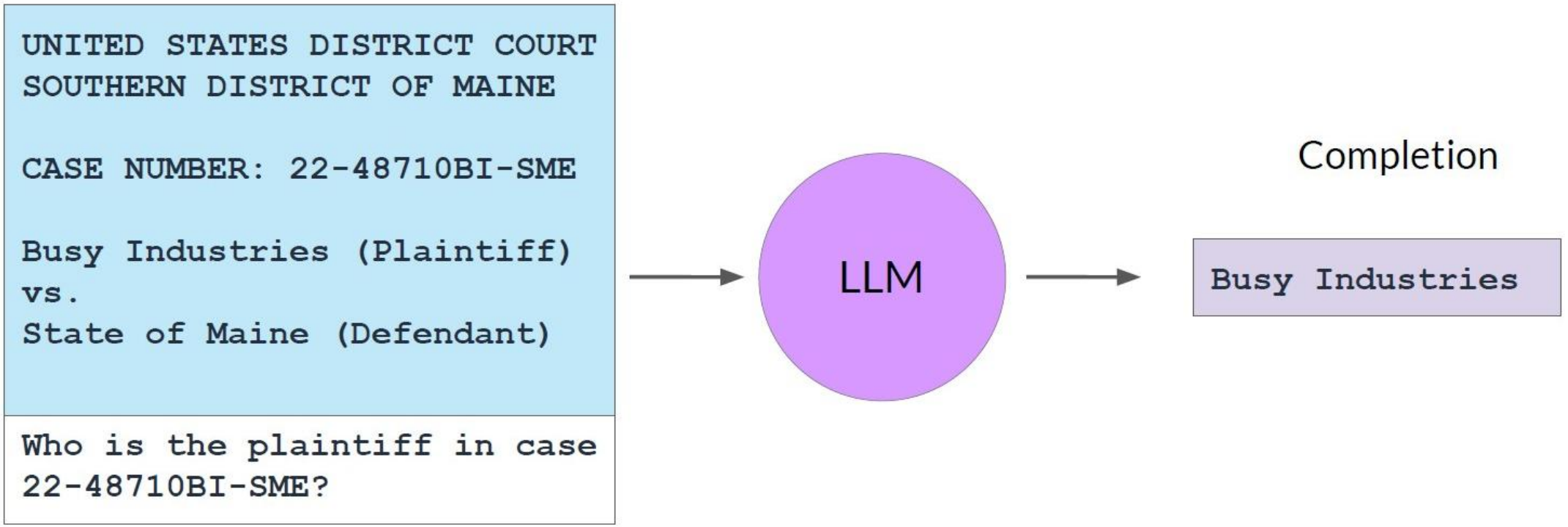
- Coursera Lecture Notes : **Generative AI with Large Language Models**

Example: Searching Legal Documents



- Coursera Lecture Notes : **Generative AI with Large Language Models**

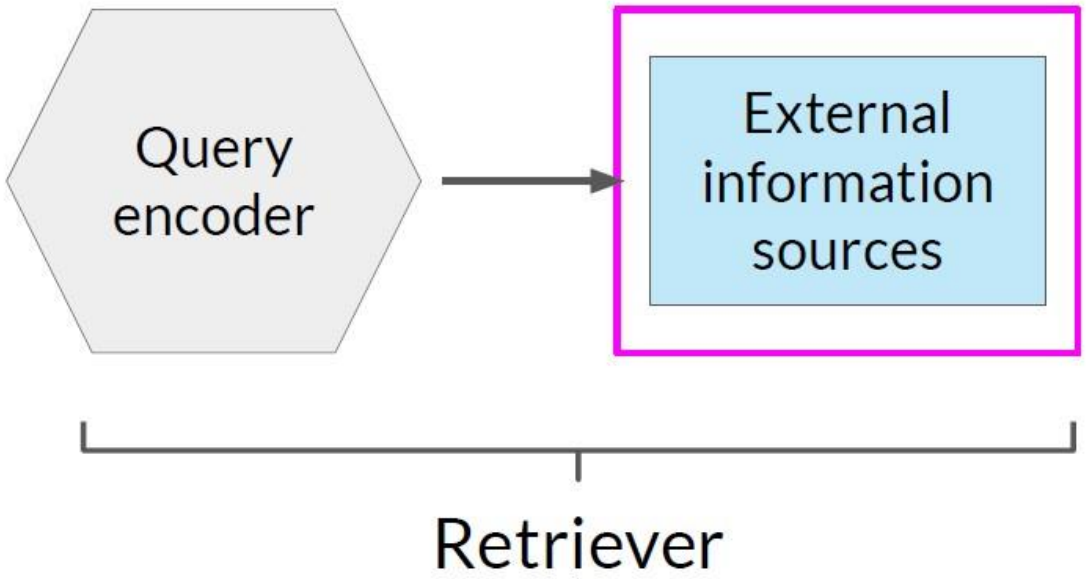
Example: Searching Legal Documents



- Coursera Lecture Notes : **Generative AI with Large Language Models**

RAG- Data Sources

RAG integrates with many types of data sources



External Information Sources

- Documents
- Wikis
- Expert Systems
- Web pages
- Databases
- Vector Store

Data preparation for vector store for RAG

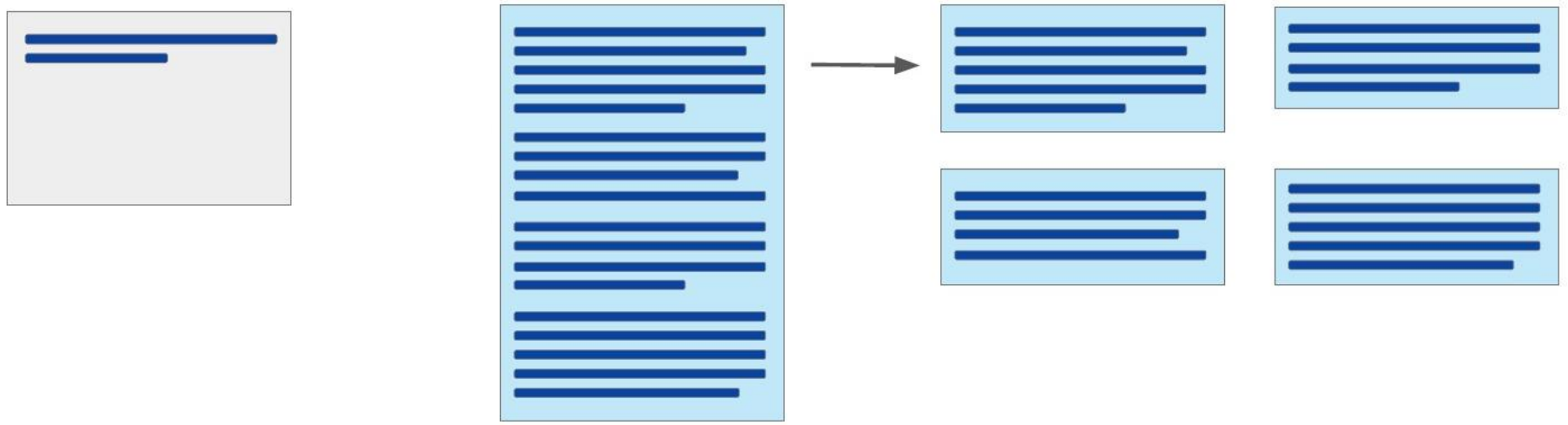
Two considerations for using external data in RAG:

1. Data must fit inside context window

Prompt context limit few 1000 tokens

Single document too large to fit in window

Split long sources into short chunks



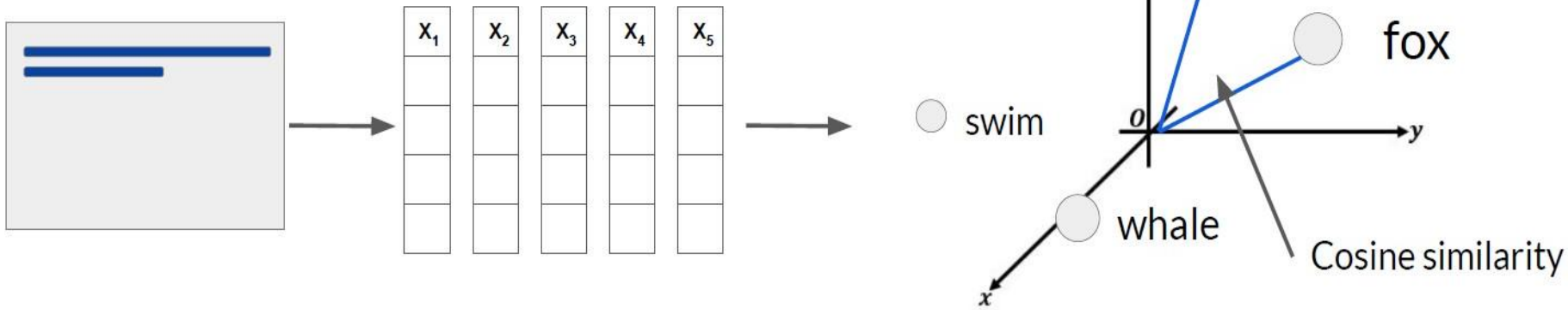
- Coursera Lecture Notes : **Generative AI with Large Language Models**

Data preparation for RAG

Two considerations for using external data in RAG:

1. Data must fit inside context window
2. Data must be in format that allows its relevance to be assessed at inference time: **Embedding vectors**

Prompt text converted to embedding vectors



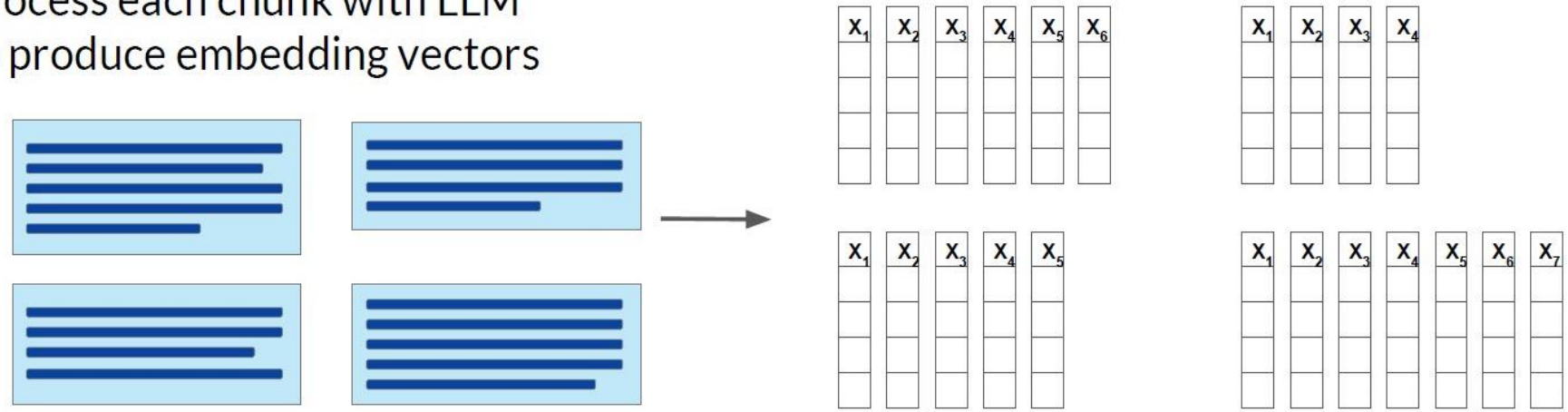
- Coursera Lecture Notes : **Generative AI with Large Language Models**

Data preparation for RAG

Two considerations for using external data in RAG:

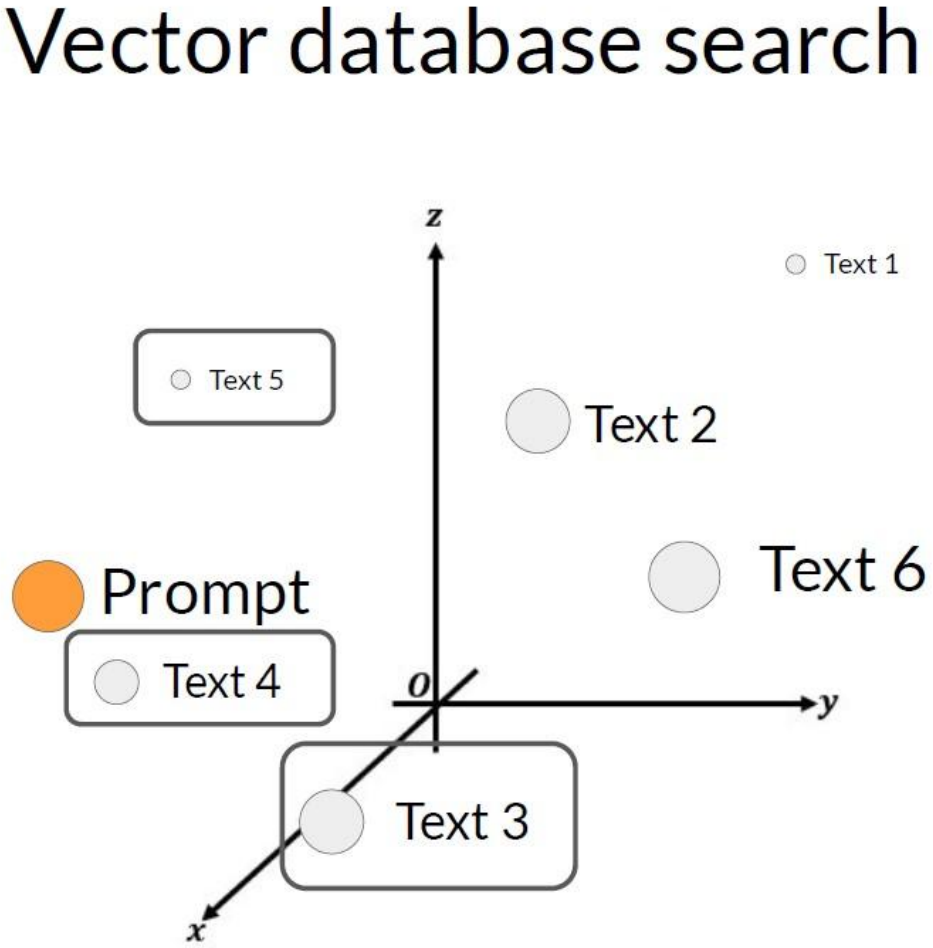
1. Data must fit inside context window
2. Data must be in format that allows its relevance to be assessed at inference time: **Embedding vectors**

Process each chunk with LLM
to produce embedding vectors



- Coursera Lecture Notes : **Generative AI with Large Language Models**

Vector Database Search



- Each text in vector store is identified by a key
- Enables a **citation** to be included in completion

- Coursera Lecture Notes : **Generative AI with Large Language Models**