CSL446
Neural Network and Deep Learning

Module 4

Introduction to Transformers:

Exploring attention mechanisms

Dr. Snehal B Shinde

Computer Science and Engineering

Indian Institute of Information Technology, Nagpur.

Sequence Models

- Sequence Models
- RNN
- BPTT
- LSTM & GRU
- Word Embeddings
- Attention Mechanism

Sequence-to-Sequence (Seq2Seq) Overview

- A sequence-to-sequence model is a model that takes a sequence of items (words, letters, features of an images…etc) and outputs another sequence of items. A trained model would work like this:

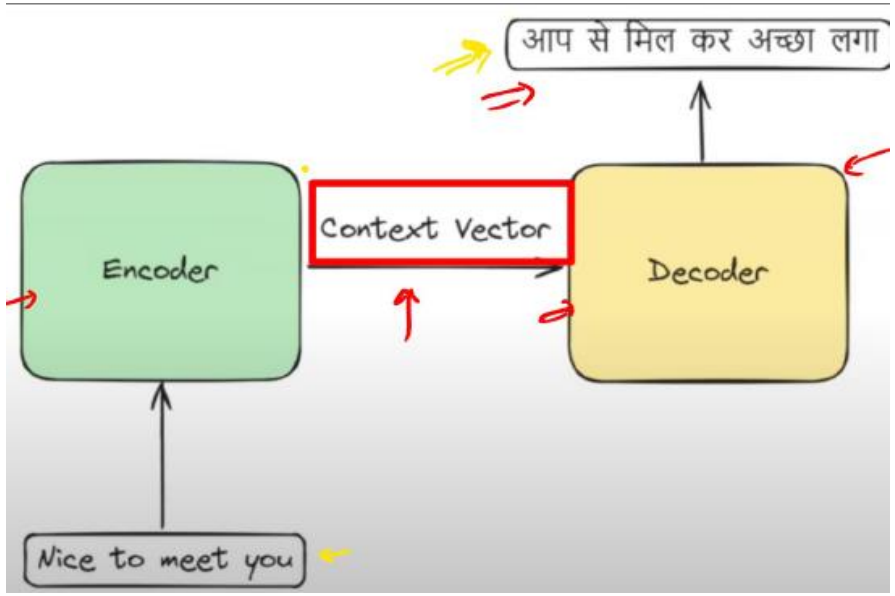
- Sequence-to-sequence (seq2seq) models in NLP are used to convert sequences of Type A to sequences of Type B

- For example, translation of English sentences to Hindi sentences is a sequence-to-sequence task.

- Recurrent Neural Network (RNN) based sequence-to sequence models have garnered a lot of traction ever since they were introduced in 2014.

- https://arxiv.org/abs/1409.3215

1. The Encoder–Decoder Architecture (2014)



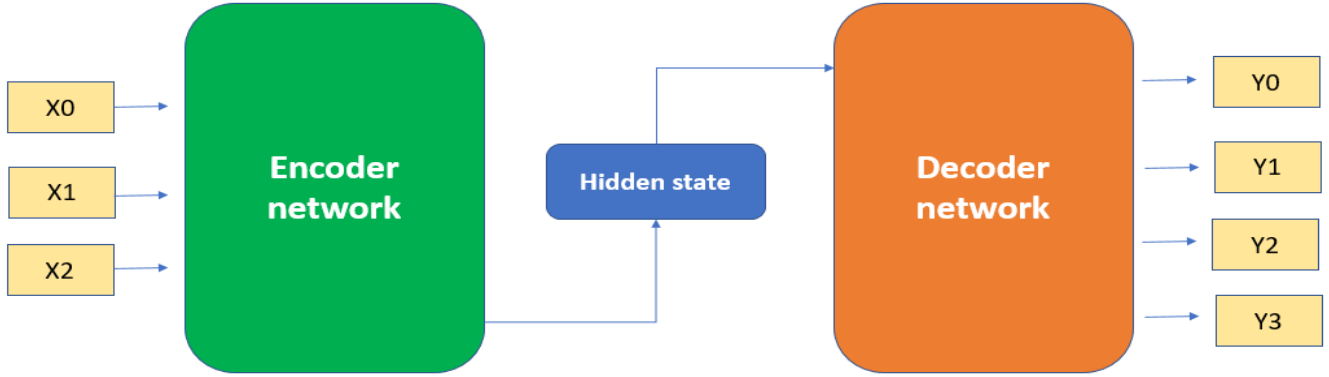
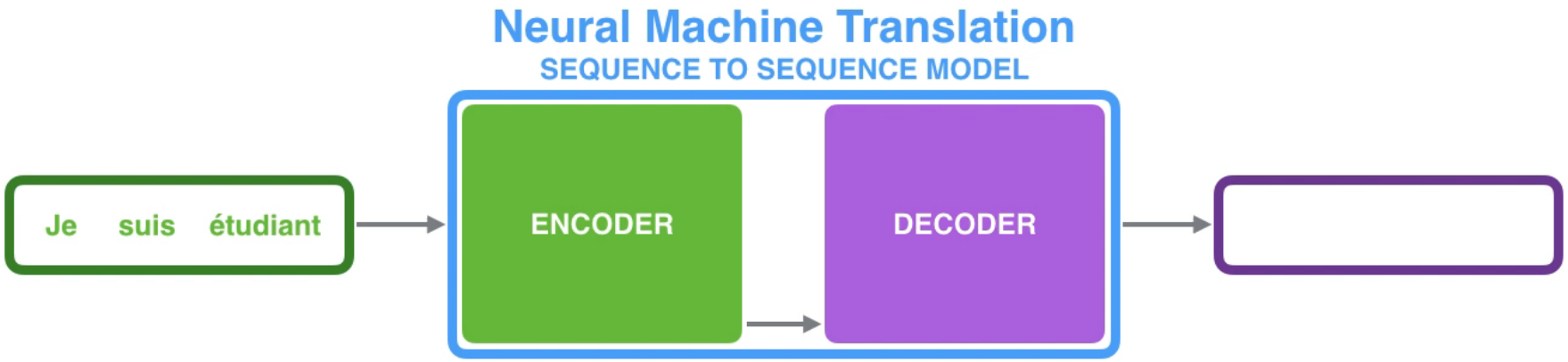


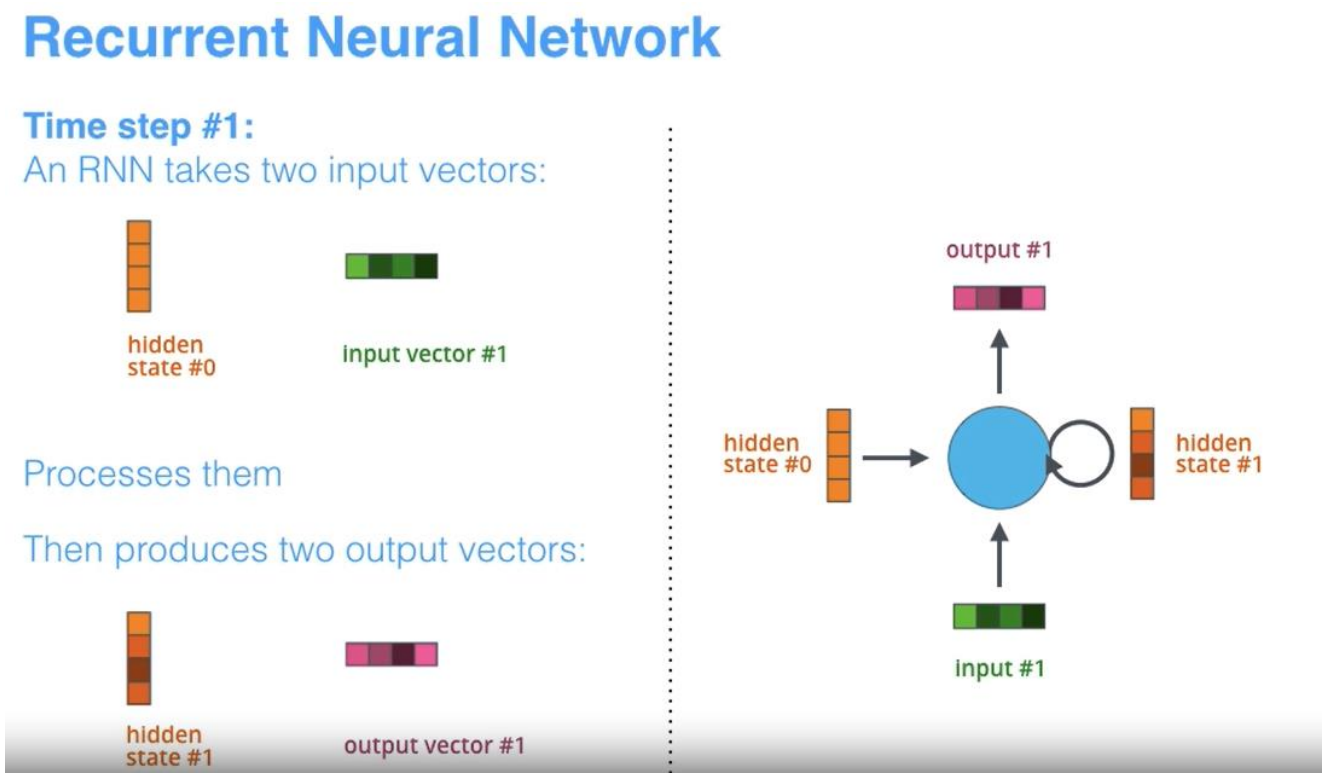
Fig Source- Internet

Sequence-to-Sequence (Seq2Seq) Overview



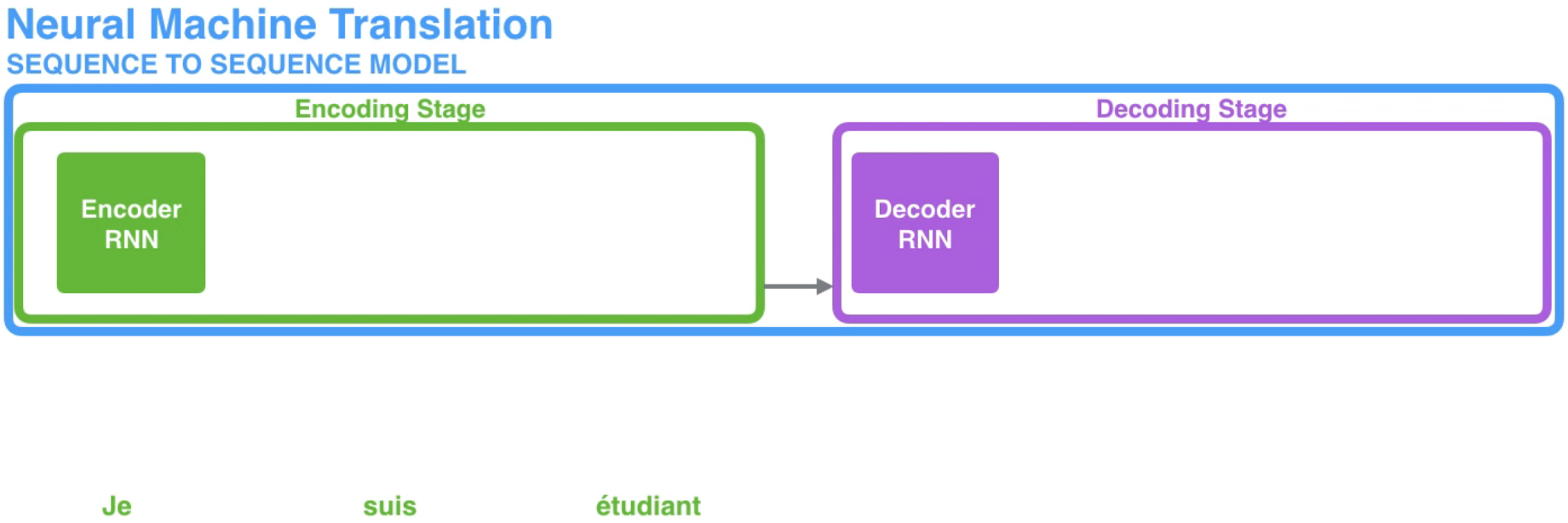
The context is a vector (an array of numbers, basically) in the case of machine translation.
The encoder and decoder tend to both be recurrent neural networks

Sequence-to-Sequence (Seq2Seq) Overview



The context is a vector (an array of numbers, basically) in the case of machine translation.
The encoder and decoder tend to both be recurrent neural networks

Sequence-to-Sequence (Seq2Seq) Overview



- Each pulse for the encoder or decoder is that RNN processing its inputs and generating an output for that time step.
- The last hidden state is actually the context we pass along to the decoder.

The Encoder–Decoder Architecture . . .

Encoder:

- Processes the input sequence and converts it into a **fixed-size context vector**.
- Example: Sentence in English → Vector representation.

- **Decoder**:
  - Generates the output sequence from the context vector, step by step.
  - Example: Vector representation → Sentence in expected language.

- **Input and Output Sequence Lengths**:
  - Input and output sequences can have **different lengths**, making Seq2Seq suitable for translation and summarization.

Sequence Models

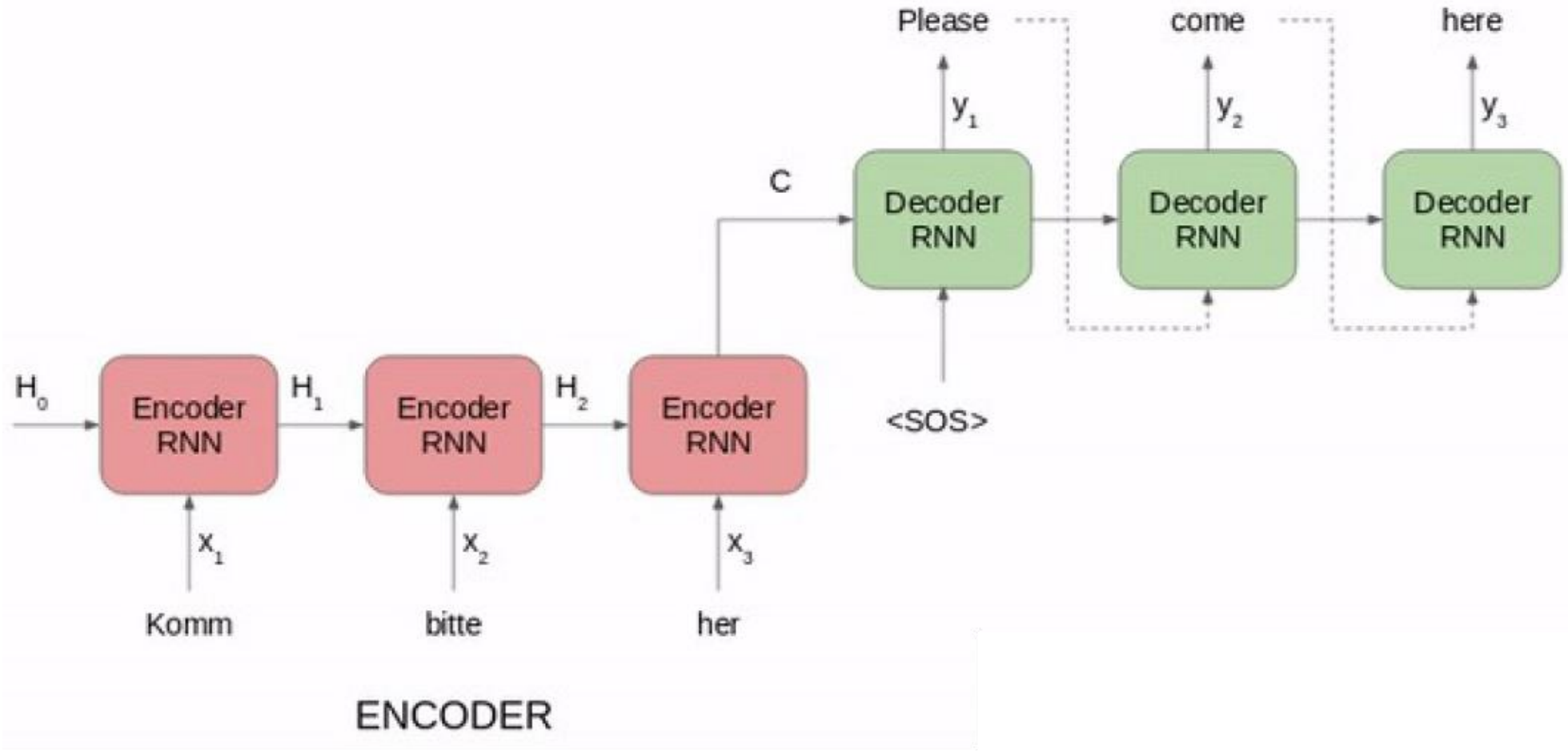


Fig Source- Internet

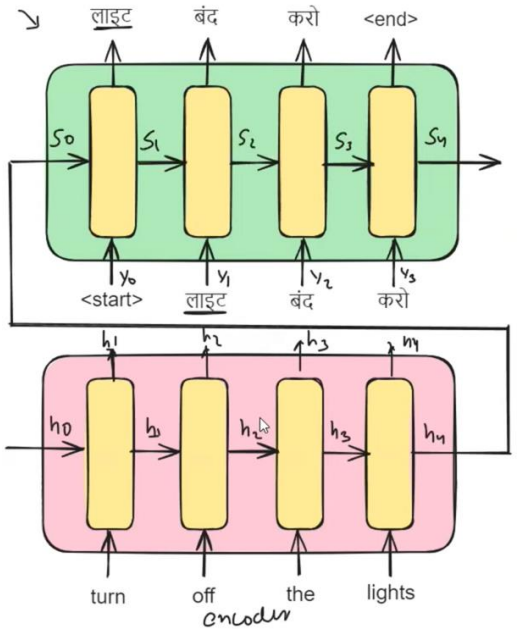
Challenges with The Encoder–Decoder Architecture

- Fixed-Size Context Vector:
  - Entire input sequence is compressed into a single vector.
  - Causes information loss for long sequences.

- Long-Term Dependencies:
  - Difficulty in retaining information from earlier parts of the sequence.
  - Example: Translating "The dog ran quickly because it saw a rabbit." may fail to link "it" to "the dog.“

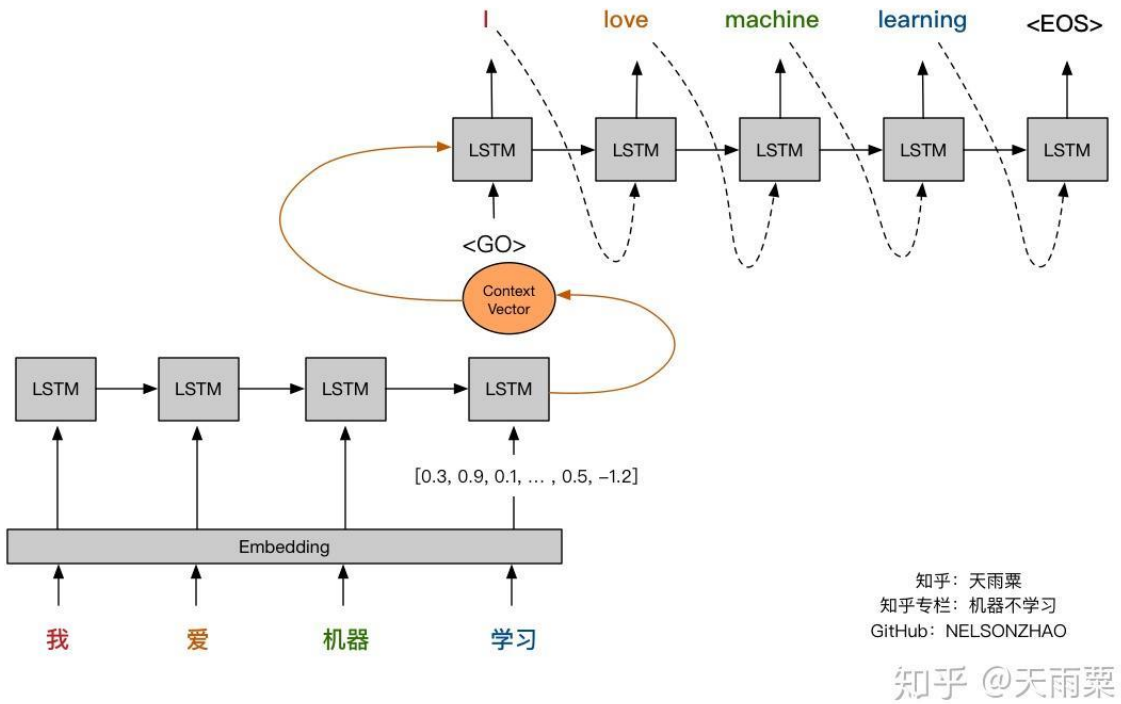
- Sequential Processing: Cannot utilize parallel computation efficiently.

2. Encoder-Decoder with Attention (2015)



- **Core Idea**:
  - Attention allows the model to focus on different parts of the input sentence when producing each word in the output.
  - Instead of using a single fixed-size context vector, attention dynamically computes a **weighted context vector** for each decoding step.
  - This allows the decoder to focus on the most relevant parts of the input sequence.
  - The model learns which words in the input sentence are important for generating each word in the output.
- Example: English to French Translation Input: "I am reading a book“
- Output: "Je lis un livre“
- When predicting "livre", attention gives higher weight to "book" than other words.

Sequence-to-Sequence (Seq2Seq) Model with LSTM (Long Short-Term Memory) Networks



The entire sentence is compressed into a single context vector, making it difficult for long sentences to retain information.

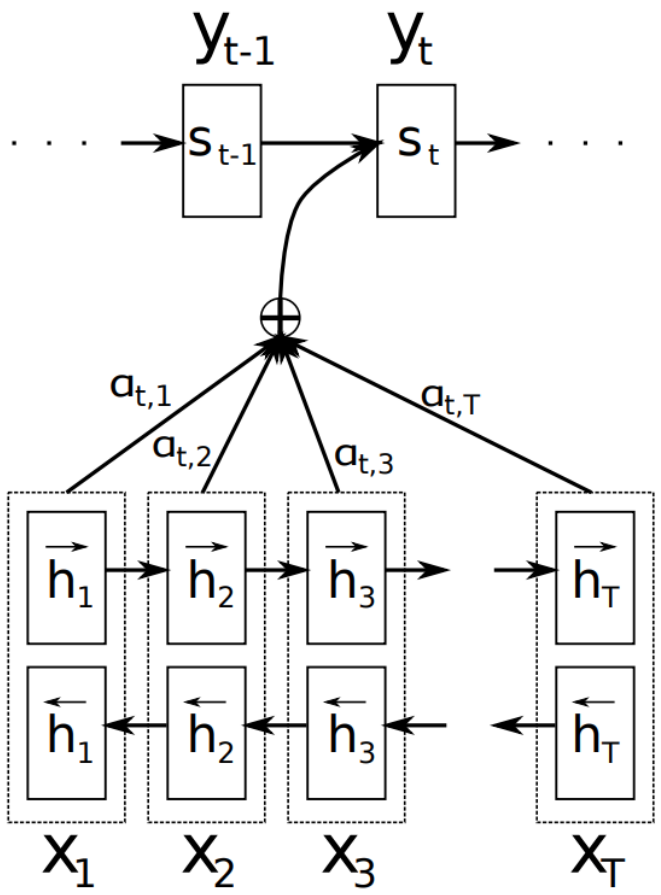
Decoder

- The Decoder is another LSTM network that generates the output sentence.
- It takes the Context Vector as the initial hidden state.
- The decoder starts with a <GO> token and predicts words one by one.
- Each predicted word (e.g., "I") is fed into the next LSTM unit to generate the next word ("love"), continuing until the <EOS> (End of Sentence) token is produced.

Encoder:

- The input sentence in Chinese ("我 爱 机器 学习") is first passed through an Embedding Layer, converting words into vector representations.
- These embeddings are then processed sequentially by LSTM layers.
- The final hidden state of the last LSTM unit is called the Context Vector, which contains the entire meaning of the input sentence.

Attention-based Sequence-to-Sequence (Seq2Seq) model



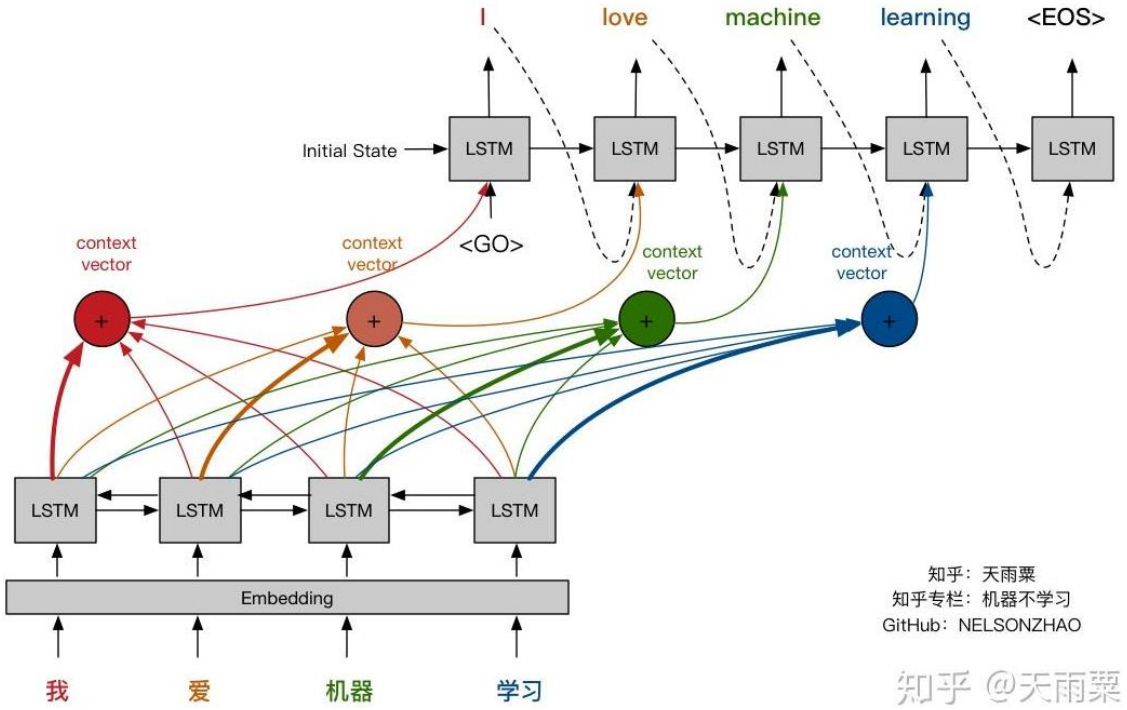
https://arxiv.org/pdf/1409.0473v7

To produce the output word at time yt the decoder uses the

- last hidden state from the decoder - one can think about this as some sort of representation of the already produced words
- and a dynamically computed context vector based on the input sequence.

Attention at time step 4

Attention-based Sequence-to-Sequence (Seq2Seq) model



1. Each word is converted into a **vector representation** using an **embedding layer**.

2. The decoder starts **generating the output sequence** (English translation).

3. Instead of using only the **final hidden state** from the encoder (as in a basic Seq2Seq model), the decoder now **computes attention weights** for each encoder output at every decoding step.

4. **Looks at all encoder hidden states** (instead of just the last one).

5. **Computes attention weights** based on how important each encoder word is for the current output word and Generates a weighted sum (context vector)

where si **is the hidden state for time** i**, computed by:**

$$s_i = f(s_{i-1}, y_{i-1}, c_i)$$

Advantages of Attention in Encoder-Decoder

- **Improved Translation Quality**:
  - Handles long sequences by dynamically focusing on relevant input tokens.
- **Alignment Visualization**:
  - Attention provides interpretable alignment between input and output tokens.
- **Scalability**:
  - Foundational concept for self-attention in Transformers.

Limitation of Attention in Encoder-Decoder

- Although attention improved the Seq2Seq model, it still had limitations:

- **Sequential Processing** – LSTMs process words one at a time, making training slow.

- **Long Dependency Issues** – Even with attention, LSTMs still struggle with **very long sentences**.

- **Hard to Parallelize** – Since LSTMs process tokens sequentially, they **cannot take advantage of GPUs efficiently**.

3. Transformers (2017)- Attention Is All You Need.

The key idea: **Self-Attention + Multi-Head Attention** for better context understanding.

Transformers' Solution:

- Fully attention-based architecture for faster, parallelizable sequence modelling.
- **Better at capturing global dependencies.**
- Instead of processing one word at a time like an LSTM, Self-Attention allows every word to directly interact with every other word in a sentence.
- Example: Sentence: "The cat sat on the mat.
- "With Self-Attention, when encoding "**mat**", the model can focus on "**sat**" and "**on**", learning their relationships directly.

Sequence Models

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

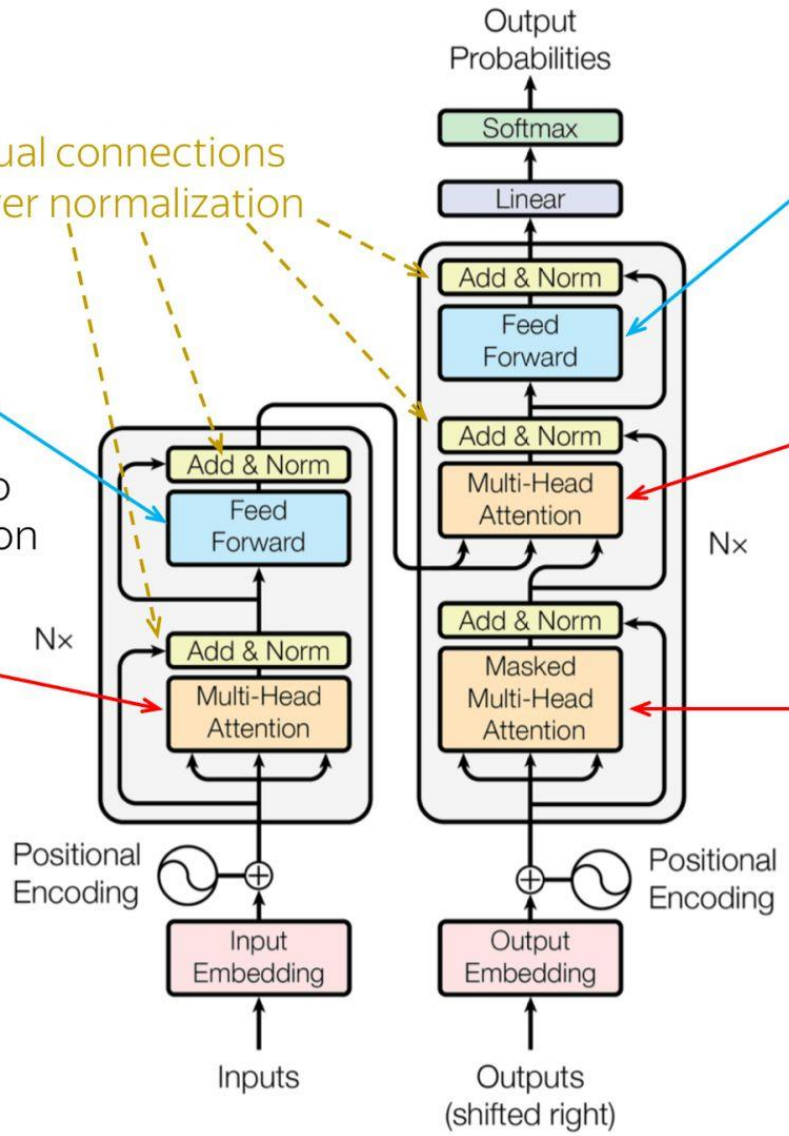
Llion Jones* Google Research llion@google.com

Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu

Łukasz Kaiser* Google Brain lukaszkaiser@google.com

Illia Polosukhin* ‡ illia.polosukhin@gmail.com

Transformer



Residual connections and layer normalization

Feed-forward network: after taking information from other tokens, take a moment to think and process this information

Encoder self-attention: tokens look at each other

queries, keys, values are computed from encoder states

Feed-forward network: after taking information from other tokens, take a moment to think and process this information

Decoder-encoder attention: target token looks at the source

queries – from decoder states; keys and values from encoder states

Decoder self-attention (masked): tokens look at the previous tokens

queries, keys, values are computed from decoder states

Sequence Models

1. **1. Sentence**: *"I am at river bank"*

2. **Tokens:** ["I", "am", "at", "river", "bank"]

3. **Word Embeddings**: token to vector

"I" → [0.2, 0.5, -0.1]
"am" → [0.1, -0.3, 0.4]
"at" → [-0.2, 0.1, 0.3]
"river" → [0.7, -0.4, 0.2]
"bank" → [0.5, 0.6, -0.1]

Instead of creating embeddings from scratch, we can use pre-trained word vectors. Some popular pre-trained embeddings include:

- **Word2Vec (Google News, Wikipedia)**
- **GloVe (Common Crawl, Wikipedia)**
- **FastText (Facebook Research)**

Sequence Models

1. Sentence: *"I am at HDFC bank"*

2. Tokens: ["I", "am", "at", “HDFC", "bank"]

2. Word Embeddings: token to vector

"I" → [0.2, 0.5, -0.1]
"am" → [0.1, -0.3, 0.4]
"at" → [-0.2, 0.1, 0.3]
”HDFC" → [0.5, -0.3, 0.7]
"bank" → [0.5, 0.6, -0.1]

Sequence Models

1. Sentence: *"I am at HDFC bank"*

2. Tokens: ["I", "am", "at", “HDFC", "bank"]

2. Word Embeddings: token to vector

"I" → [0.2, 0.5, -0.1]
"am" → [0.1, -0.3, 0.4]
"at" → [-0.2, 0.1, 0.3]
”HDFC" → [0.5, -0.3, 0.7]
"bank" → [0.5, 0.6, -0.1]

1. Sentence: *"I am at river bank"*

2. Tokens: ["I", "am", "at", "river", "bank"]

2. Word Embeddings: token to vector

"I" → [0.2, 0.5, -0.1]
"am" → [0.1, -0.3, 0.4]
"at" → [-0.2, 0.1, 0.3]
"river" → [0.7, -0.4, 0.2]
"bank" → [0.5, 0.6, -0.1]

Sequence Models

1. Sentence: *"I am at HDFC bank"*

2. Tokens: ["I", "am", "at", “HDFC", "bank"]

2. Word Embeddings: token to vector

"I" → [0.2, 0.5, -0.1]
"am" → [0.1, -0.3, 0.4]
"at" → [-0.2, 0.1, 0.3]
”HDFC" → [0.5, -0.3, 0.7]
"bank" → [0.5, 0.6, -0.1]

1. Sentence: *"I am at river bank"*

2. Tokens: ["I", "am", "at", "river", "bank"]

2. Word Embeddings: token to vector

"I" → [0.2, 0.5, -0.1]
"am" → [0.1, -0.3, 0.4]
"at" → [-0.2, 0.1, 0.3]
"river" → [0.7, -0.4, 0.2]
"bank" → [0.5, 0.6, -0.1]

Static Word Embeddings

Self Attention

- **Contextual embedding in self-attention** refers to the process of creating token representations that incorporate information from the entire input sequence, allowing each token's embedding to reflect not only its own meaning but also its contextual relationships with other tokens.

Sequence Models

1. Sentence: *"I am at river bank"*

2. Tokens: ["I", "am", "at", "river", "bank"]

2. Word Embeddings:

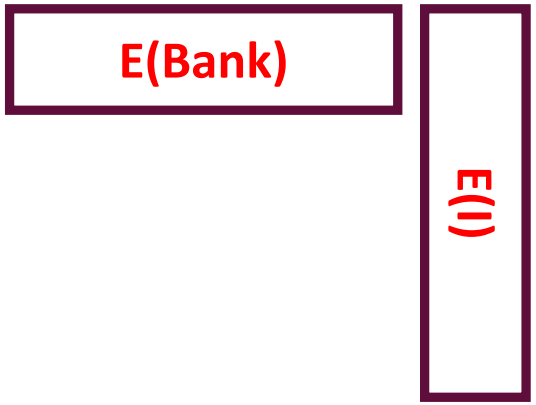
E(bank)= 0.2*E(I)+ 0.1* E(am)+ 0.1* E(at)+ 0.3* E(river)+ 0.3* E(bank)

Similarity score (scalar)

Sequence Models

E(bank)= 0.2*E(I)+ 0.1* E(am)+ 0.1* E(at)+ 0.3* E(river)+ 0.3* E(bank)

Similarity score (scalar)?



= Similarity score between “Bank ” and “I”

Sequence Models

E(bank)= 0.2*E(I)+ 0.1* E(am)+ 0.1* E(at)+ 0.3* E(river)+ 0.3* E(bank)

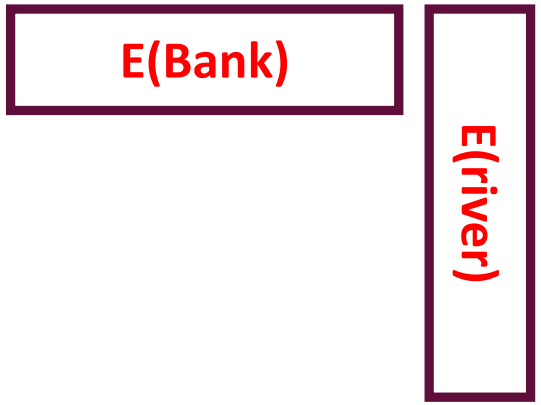
Similarity score (scalar)?

E(Bank) **E(am)** = Similarity score between “Bank ” and “am”

Sequence Models

E(bank)= 0.2*E(I)+ 0.1* E(am)+ 0.1* E(at)+ 0.3* E(river)+ 0.3* E(bank)

Similarity score (scalar)?



= Similarity score between “Bank ” and “river”

Sequence Models

E(bank)= 0.2*E(I)+ 0.1* E(am)+ 0.1* E(at)+ 0.3* E(river)+ 0.3* E(bank)

E(bank)= [**E(bank).** E(I)T]*E(I)+

[**E(bank).** E(am)T]* E(am)+

[**E(bank).** E(at)T]* E(at)+

[**E(bank).** E(river)T]* E(river)+

[**E(bank).** E(bank)T]* E(bank)

Sequence Models

E(bank)= 0.2*E(I)+ 0.1* E(am)+ 0.1* E(at)+ 0.3* E(river)+ 0.3* E(bank)

E(bank)= [**E(bank).** E(I)T]*E(I)+

[**E(bank).** E(am)T]* E(am)+

[**E(bank).** E(at)T]* E(at)+

[**E(bank).** E(river)T]* E(river)+

[**E(bank).** E(bank)T]* E(bank)

Sequence Models

E(bank)= 0.2*E(I)+ 0.1* E(am)+ 0.1* E(at)+ 0.3* E(river)+ 0.3* E(bank)

E(bank)= [**E(bank).** E(I)T]*E(I)+

[**E(bank).** E(am)T]* E(am)+

[**E(bank).** E(at)T]* E(at)+

[**E(bank).** E(river)T]* E(river)+

[**E(bank).** E(bank)T]* E(bank)

E(bank) --→ query

E(bank) **--→ key**

E(bank) **--→ value**

Sequence Models

E(bank)= 0.2*E(I)+ 0.1* E(am)+ 0.1* E(at)+ 0.3* E(river)+ 0.3* E(bank)

E(bank)= [**E(bank).** E(I)T]*E(I)+

[**E(bank).** E(am)T]* E(am)+

[**E(bank).** E(at)T]* E(at)+

[**E(bank).** E(river)T]* E(river)+

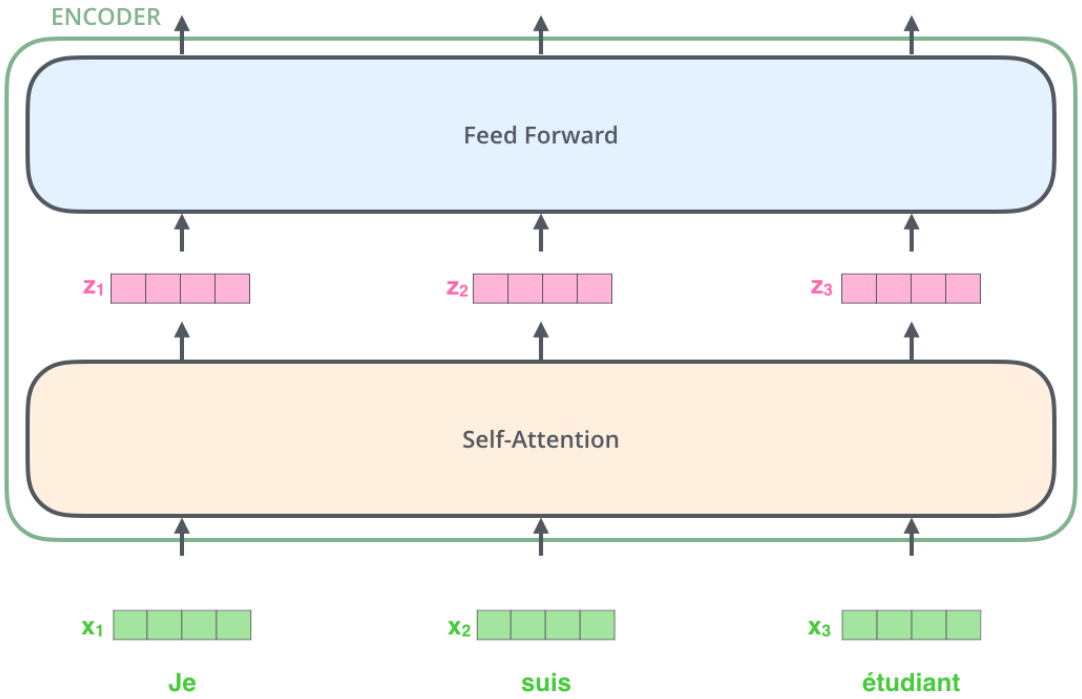
[**E(bank).** E(bank)T]* E(bank)

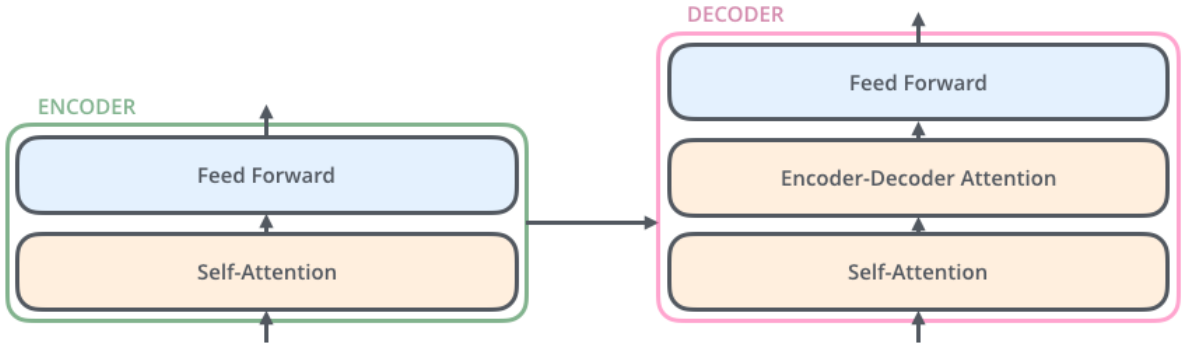
One embedding playing 3 roles

There is no learning

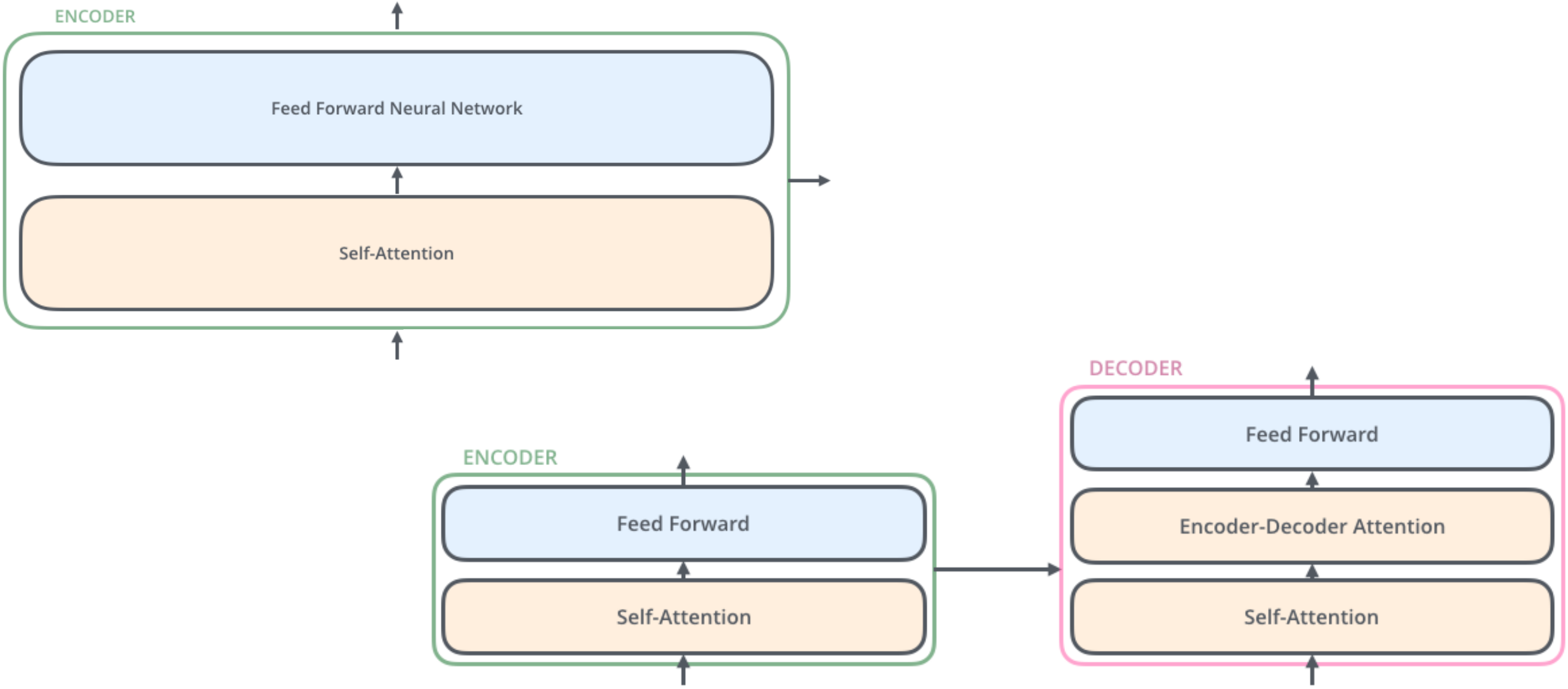
Only calculations

Transformer

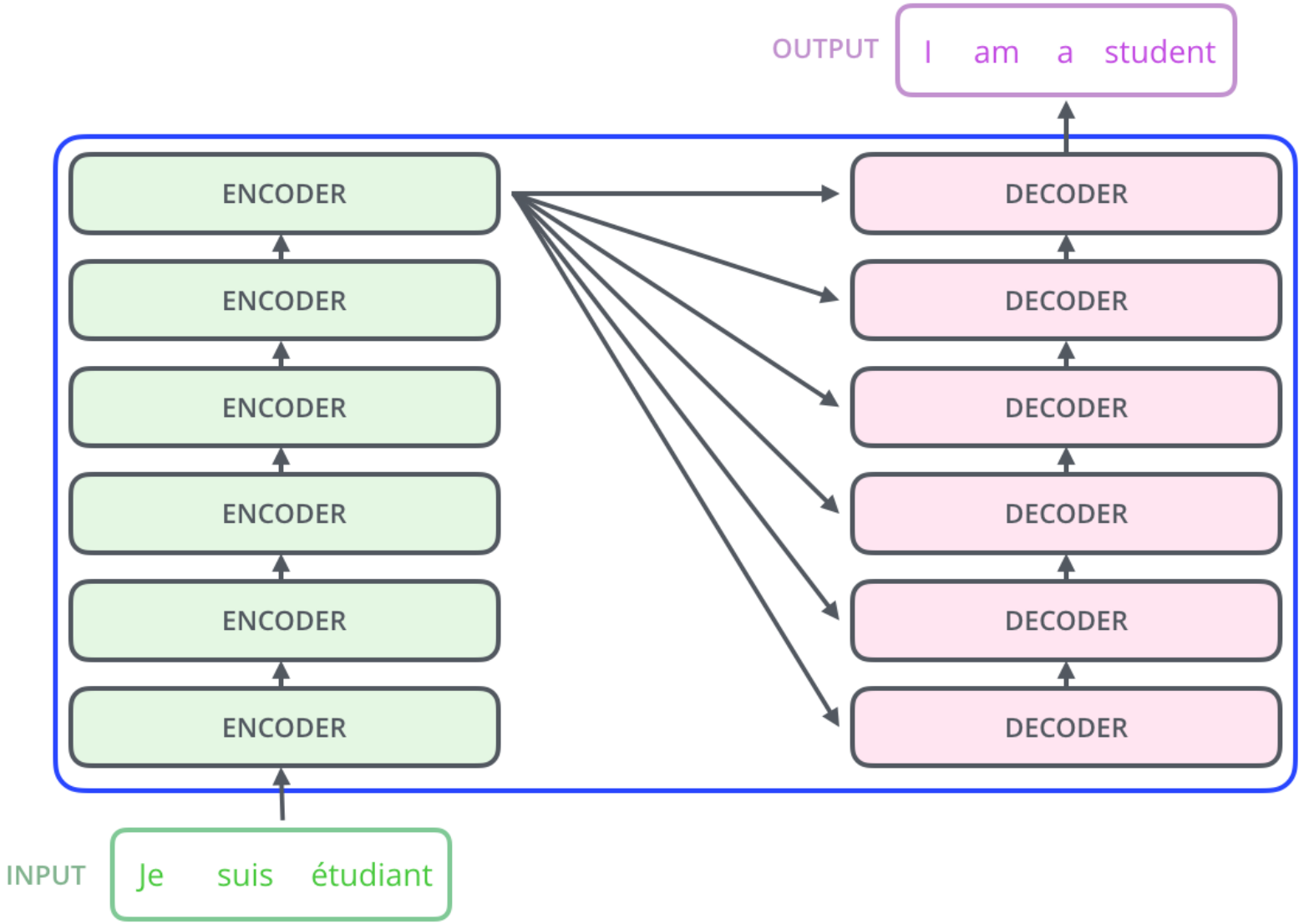




Transformer



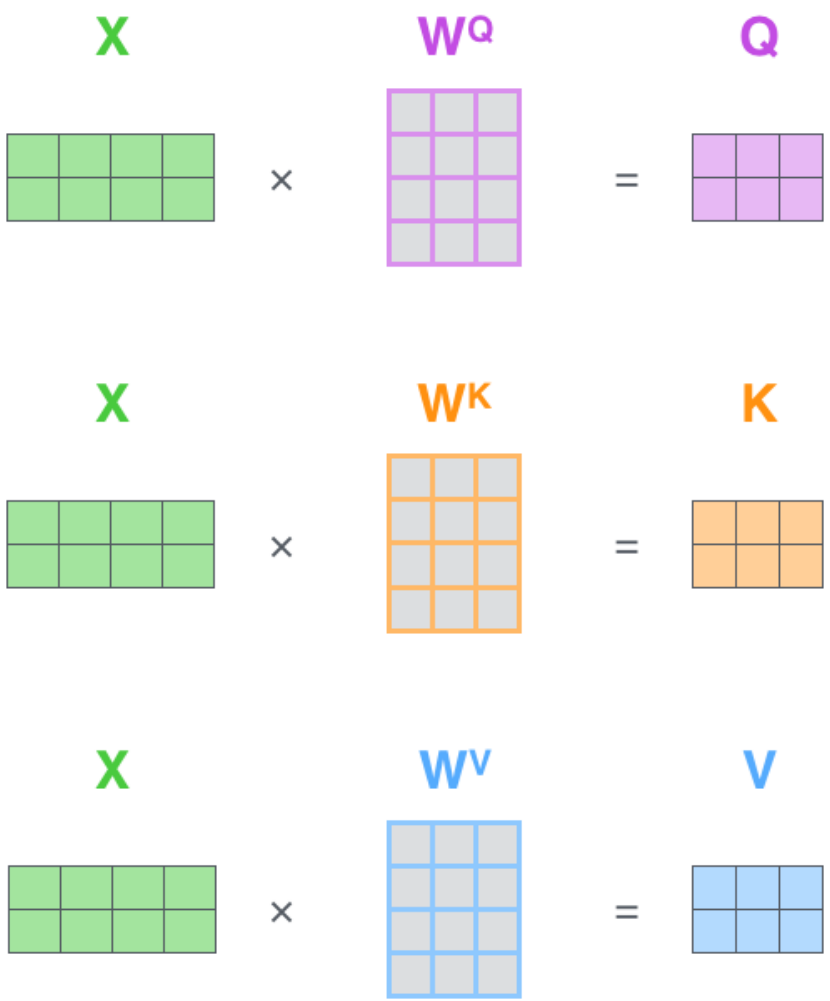
Transformer



Self-Attention at a High Level

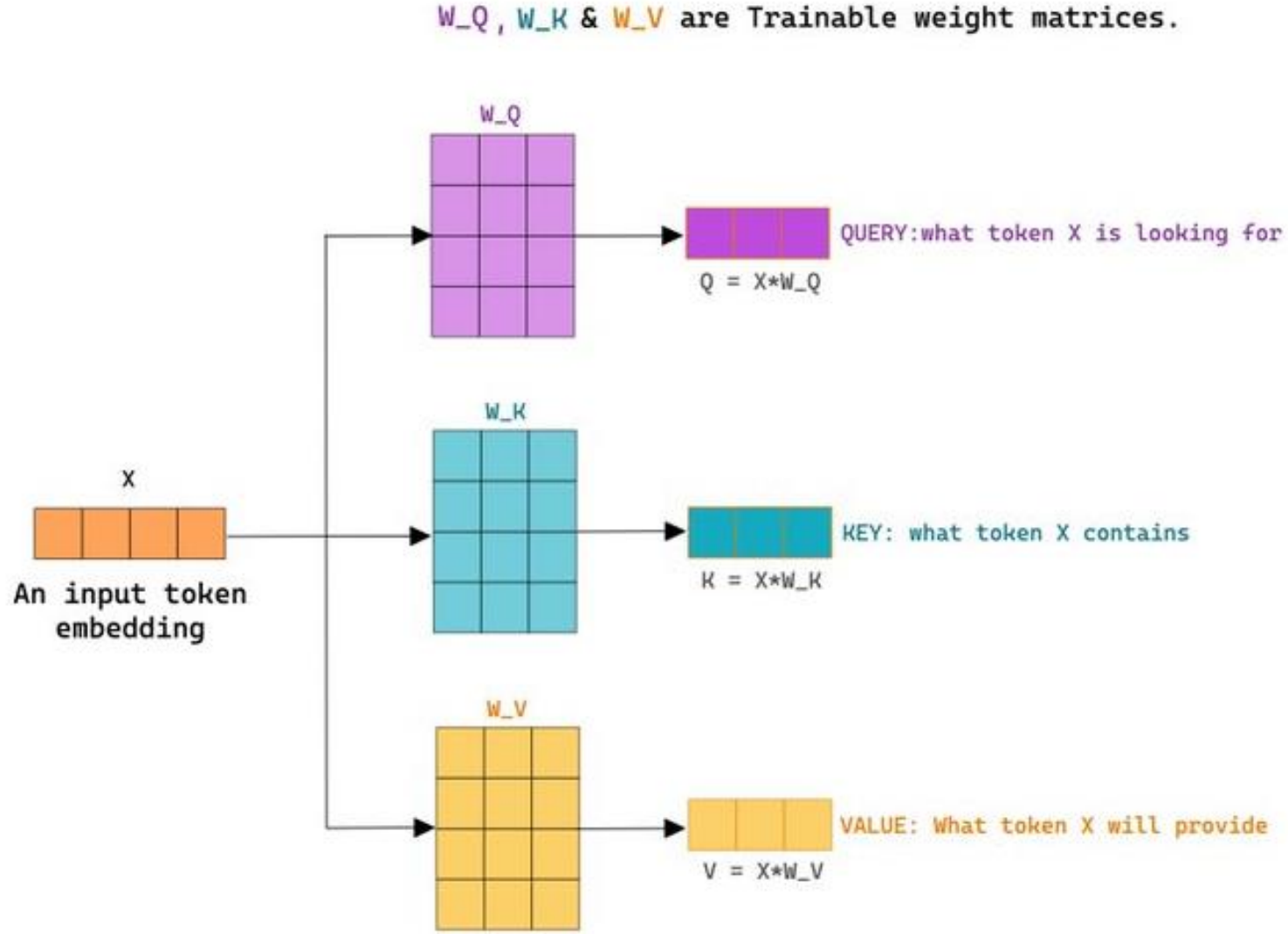
- ”The animal didn't cross the street because it was too tired”
- What does “it” in this sentence refer to? Is it referring to the street or to the animal? It’s a simple question to a human, but not as simple to an algorithm.
- When the model is processing the word “it”, self-attention allows it to associate “it” with “animal”.

Matrix Calculation of Self-Attention

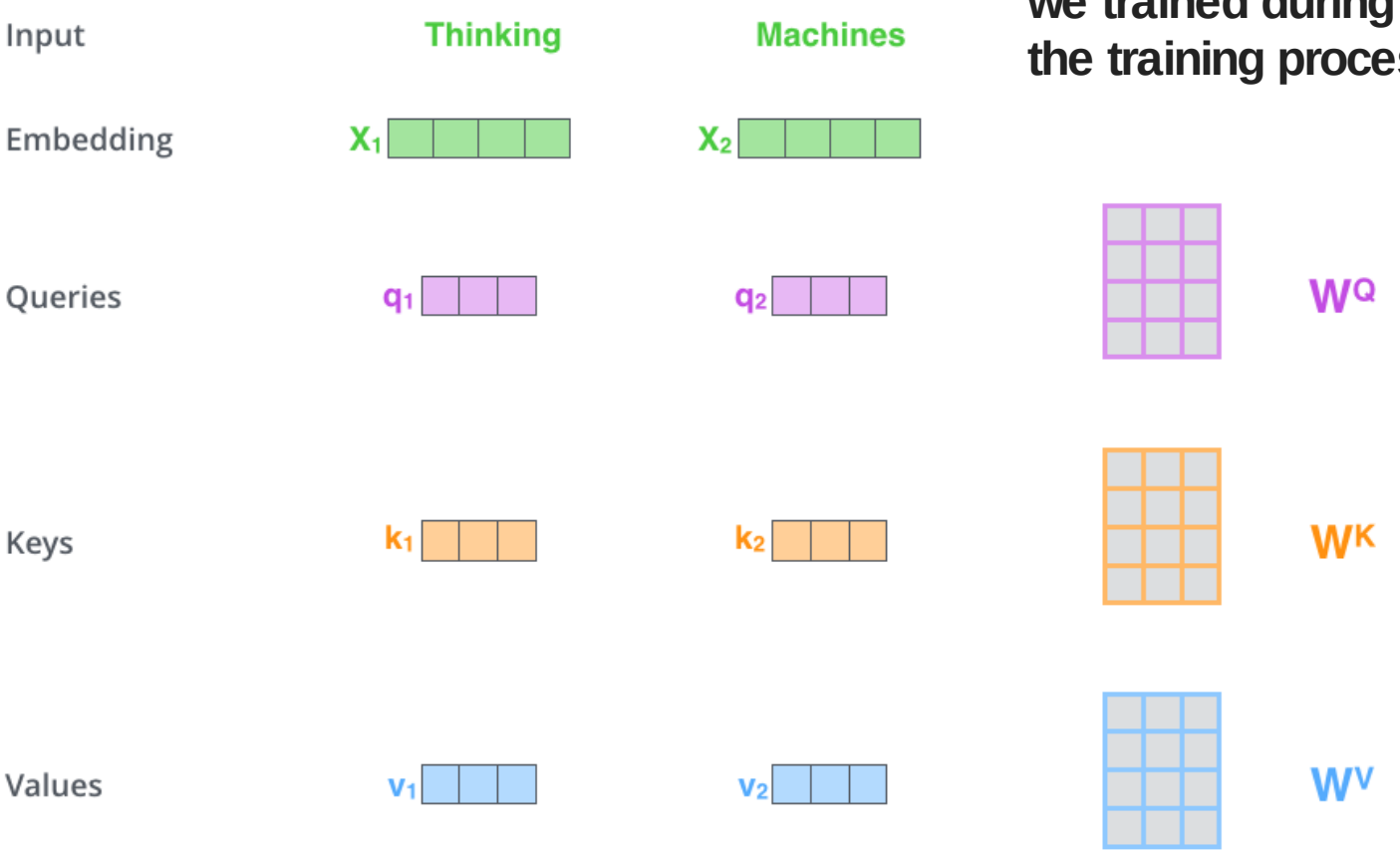


The first step is to calculate the Query, Key, and Value matrices. We do that by packing our embeddings into a matrix X, and multiplying it by the weight matrices we've trained (WQ, WK, WV).

Matrix Calculation of Self-Attention



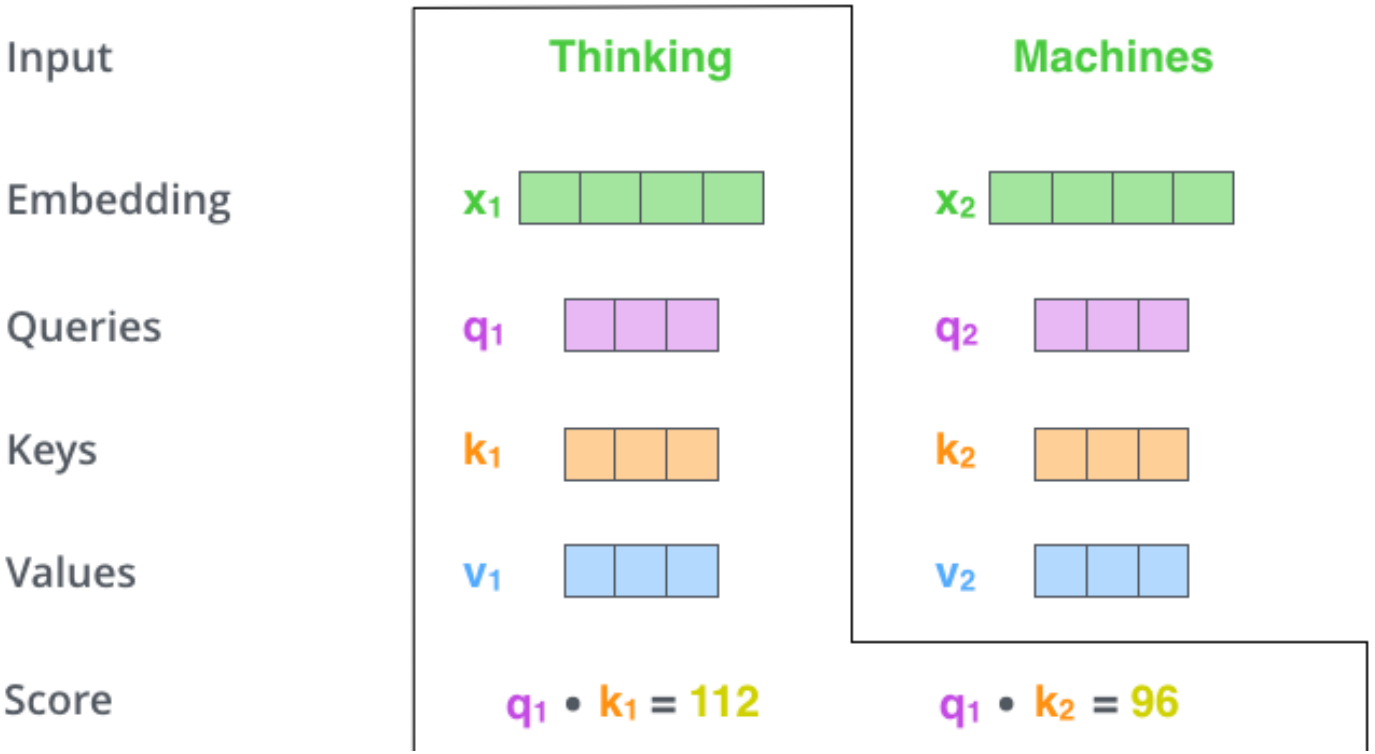
Self-Attention at a High Level



three matrices that we trained during the training process.

- The first step to create three vectors from each of the encoder's input vectors (in this case, the embedding of each word).
- So for each word, we create a **Query vector, a Key vector, and a Value vector.**

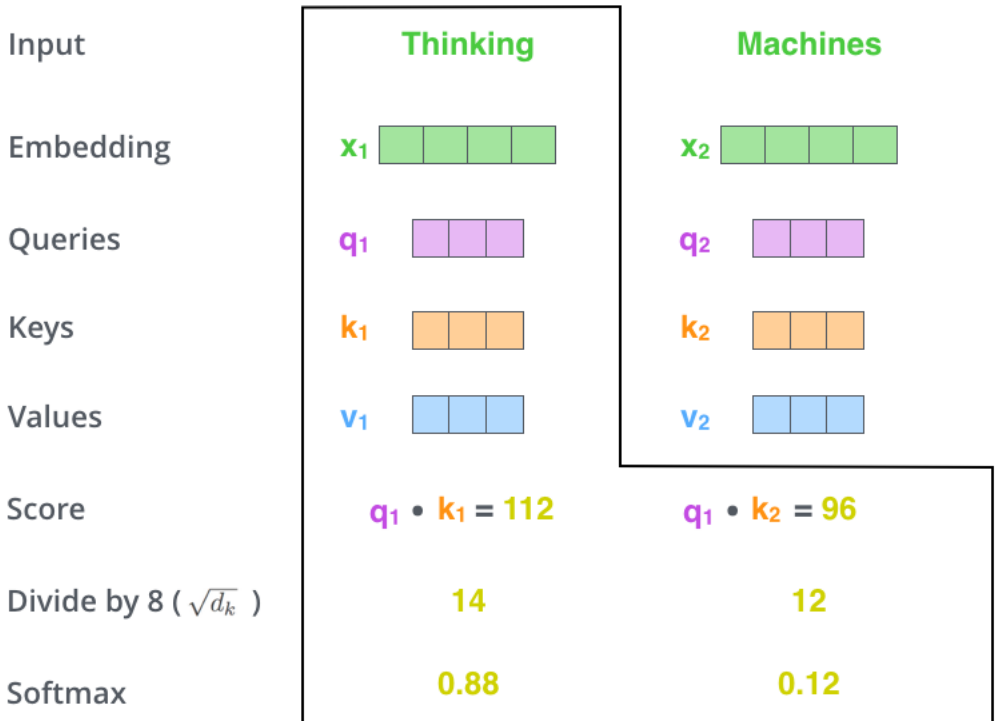
Self-Attention at a High Level



The **second step** in calculating self-attention is to calculate a score.

- The score is calculated by taking the dot product of the query vector with the key vector of the respective word we're scoring.
- So if we're processing the self-attention for the word in position #1, the first score would be the dot product of q1 and k1. The second score would be the dot product of q1 and k2.

Self-Attention at a High Level

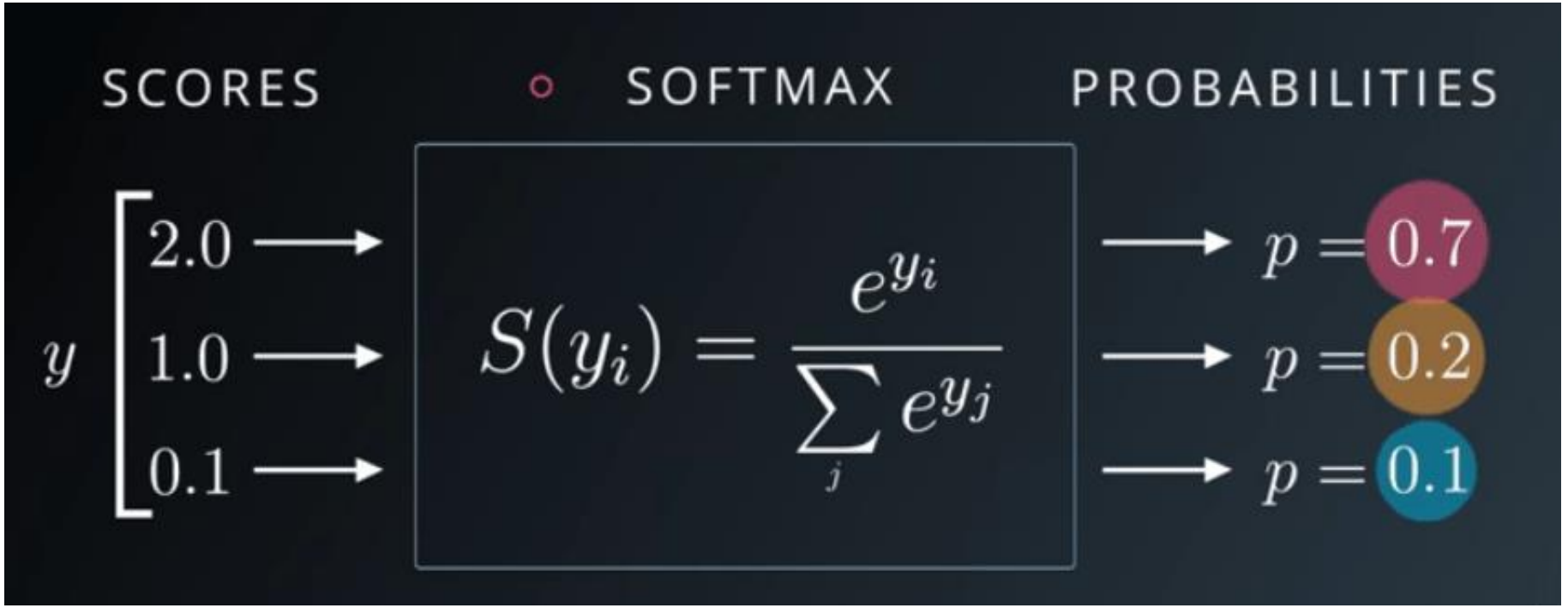


- The **third and fourth steps** are to divide the scores by 8 (the square root of the dimension of the key vectors used in the paper – 64.

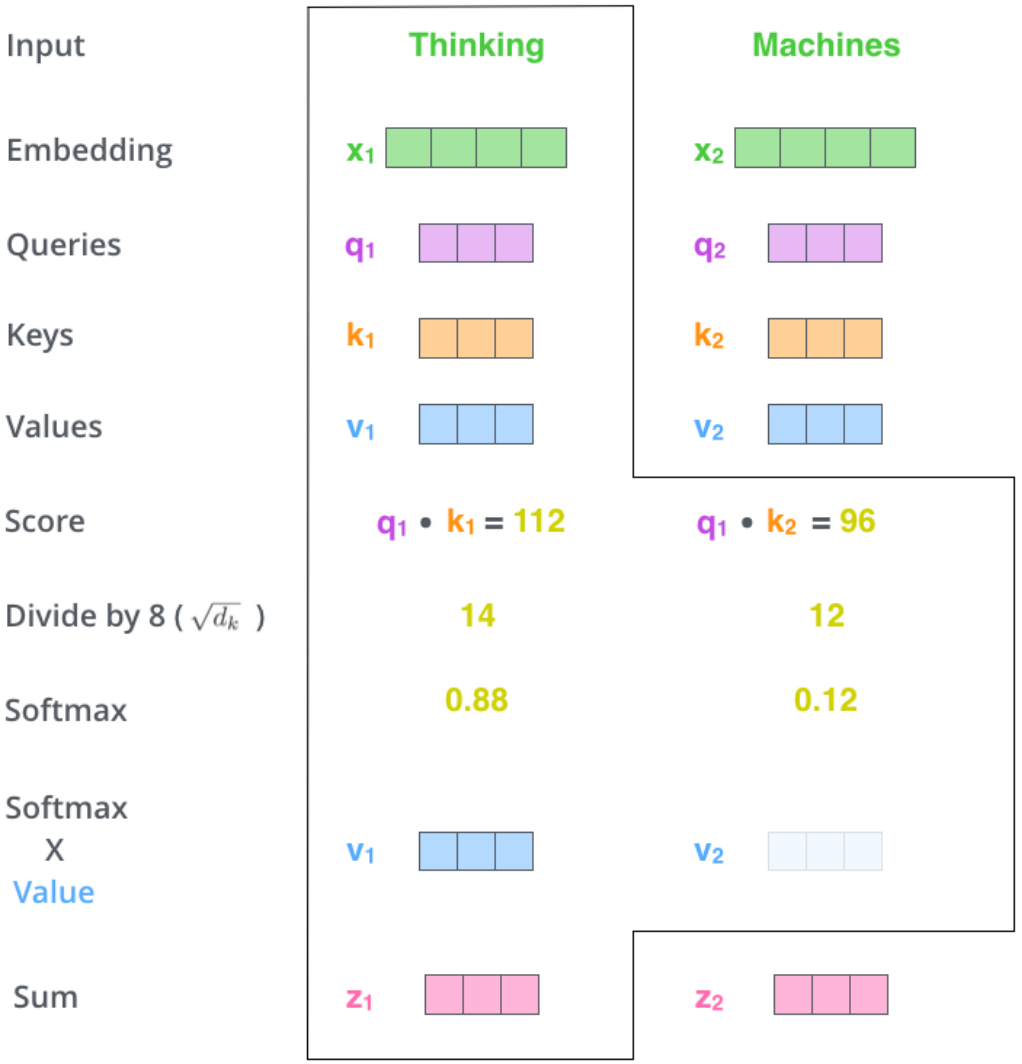
- This softmax score determines how much each word will be expressed at this position.

- Clearly the word at this position will have the highest softmax score, but sometimes it's useful to attend to another word that is relevant to the current word.

Sequence Models



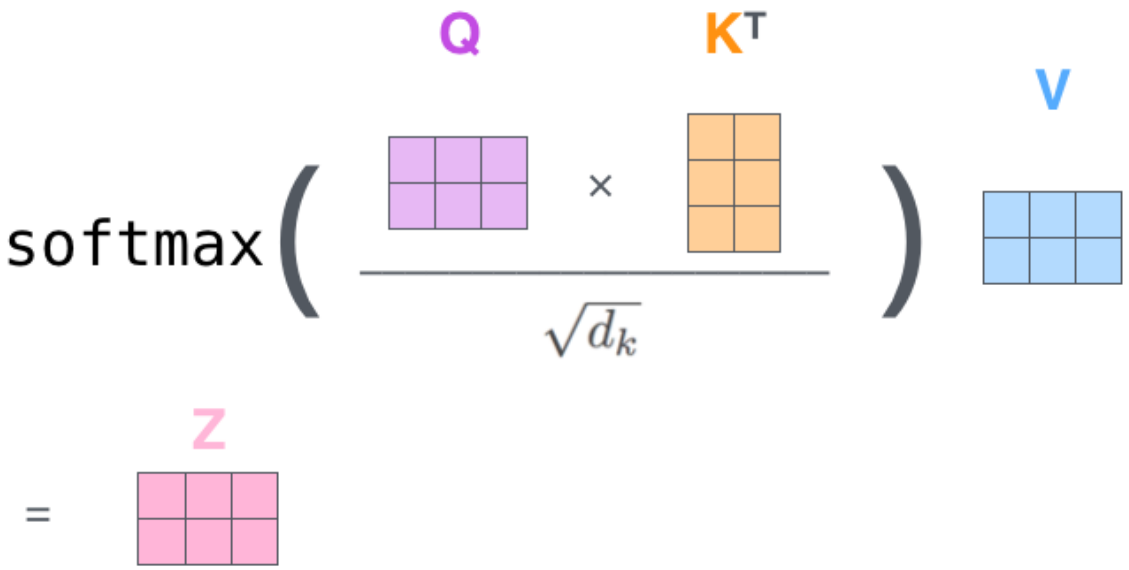
Self-Attention at a High Level



- The **fifth step** is to multiply each value vector by the softmax score (in preparation to sum them up). The intuition here is to keep intact the values of the word(s) we want to focus on, and drown-out irrelevant words (by multiplying them by tiny numbers like 0.001, for example).

- The **sixth step** is to sum up the weighted value vectors. This produces the output of the self-attention layer at this position (for the first word).

Self-Attention at a High Level



Attention Scores

- The attention mechanism calculates how much focus each token should have on the others.
- This is done by taking the dot product of the Query and Key matrices, then scaling it by the square root of the dimension dk (to stabilize gradients).

$$\text{Attention Scores} = \frac{QK^T}{\sqrt{d_k}}$$

self-attention

The self-attention mechanism is a fundamental component of Transformer models, enabling them to weigh the importance of different input elements relative to one another.

Self-Attention Equation:

The self-attention mechanism can be expressed mathematically as:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

1.Q (Query): A matrix derived from the input, representing the queries for attention.

2.K (Key): A matrix derived from the input, representing the keys for attention.

3.V (Value): A matrix derived from the input, representing the values to be attended to.

4.dk: Dimensionality of the key vector.

5. $\frac{QK^T}{\sqrt{d_k}}$ Scaled dot-product of the query and key matrices. Scaling by prevents the dot product from growing too large and destabilizing the softmax function.

6.softmax: A function applied row-wise to compute attention weights, ensuring they sum to 1 for each query.

Softmax for Weights

The scores are normalized using a softmax function to produce attention weights, which represent the importance of each token relative to others.

$$\text{Attention Weights} = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)$$

The attention weights are multiplied with the Value matrix V, effectively aggregating the information from the relevant parts of the input sequence.

$$\text{Output} = \text{Attention Weights} \cdot V$$

Step 1: Convert Words to Vectors

Word	Vector Representation (Embedding)
He	[2, 3, 1]
reads	[1, 1, 2]
books	[3, 2, 1]

$$X = \begin{bmatrix} 2 & 3 & 1 \\ 1 & 1 & 2 \\ 3 & 2 & 1 \end{bmatrix}$$

Step 2: Compute Query (Q), Key (K), and Value (V) Matrices

$$W_Q = \begin{bmatrix} 1 & 0 & 2 \\ 0 & 1 & 1 \\ 1 & 1 & 0 \end{bmatrix}$$

$$W_K = \begin{bmatrix} 2 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 2 & 1 \end{bmatrix}$$

$$W_V = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 1 & 2 \\ 2 & 1 & 0 \end{bmatrix}$$

$$Q = X \cdot W_Q$$

$$K = X \cdot W_K$$

$$V = X \cdot W_V$$

$$Q = \begin{bmatrix} 3 & 4 & 7 \\ 3 & 4 & 3 \\ 4 & 3 & 7 \end{bmatrix} \qquad K = \begin{bmatrix} 7 & 4 & 4 \\ 3 & 5 & 3 \\ 8 & 5 & 3 \end{bmatrix} \qquad V = \begin{bmatrix} 4 & 8 & 8 \\ 5 & 7 & 5 \\ 5 & 9 & 7 \end{bmatrix}$$

Step 3: Compute Attention Scores QK^T

$$\text{Score} = Q \cdot K^T$$

$$\text{Score} = \begin{bmatrix} 3 & 4 & 7 \\ 3 & 4 & 3 \\ 4 & 3 & 7 \end{bmatrix} \cdot \begin{bmatrix} 7 & 4 & 8 \\ 4 & 5 & 5 \\ 4 & 3 & 3 \end{bmatrix}$$

$$\text{Score} = \begin{bmatrix} 49 & 52 & 62 \\ 41 & 44 & 47 \\ 48 & 47 & 61 \end{bmatrix}$$

Step 4: Apply Scaling and Softmax

Scaling by $\sqrt{d_k} = \sqrt{3} \approx 1.73$:

$$\text{Scaled Score} = \frac{\text{Score}}{1.73} \qquad A = \begin{bmatrix} 0.25 & 0.35 & 0.40 \\ 0.30 & 0.33 & 0.37 \\ 0.28 & 0.32 & 0.40 \end{bmatrix}$$

Step 5: Compute Final Attention Output

$$\text{Output} = A \cdot V$$

$$\text{Output} = \begin{bmatrix} 0.25 & 0.35 & 0.40 \\ 0.30 & 0.33 & 0.37 \\ 0.28 & 0.32 & 0.40 \end{bmatrix} \cdot \begin{bmatrix} 4 & 8 & 8 \\ 5 & 7 & 5 \\ 5 & 9 & 7 \end{bmatrix}$$

$$\text{Output} = \begin{bmatrix} 4.90 & 8.20 & 6.90 \\ 4.85 & 7.80 & 6.55 \\ 4.88 & 8.05 & 6.80 \end{bmatrix}$$

The attention mechanism assigns different importance to words.
The final output is context-aware word representations.

Multi-Head Attention

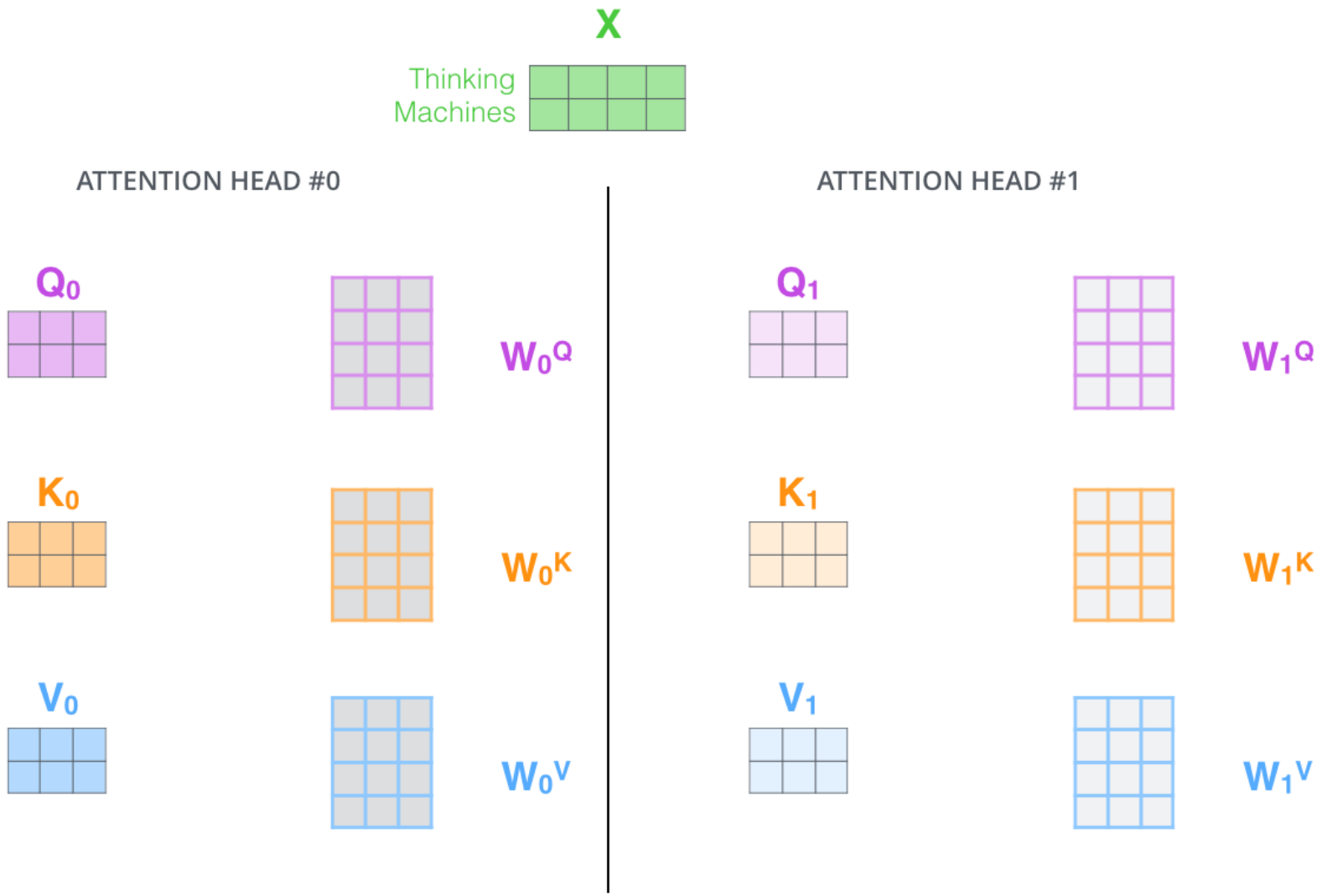
- Allows the model to focus on different parts of the input sequence simultaneously.
- Captures diverse relationships: Each head learns distinct aspects of attention.
- Improves learning efficiency and representation power.
- The Attention module **repeats its computations multiple times in parallel**. Each of these is called an Attention Head.
- The Attention module splits its Query, Key, and Value parameters N-ways and passes each split independently through a separate Head.
- All of these similar Attention calculations are then combined together to produce a final Attention score. This is called Multi-head attention and gives the Transformer greater power to encode multiple relationships

Multi-Head Attention

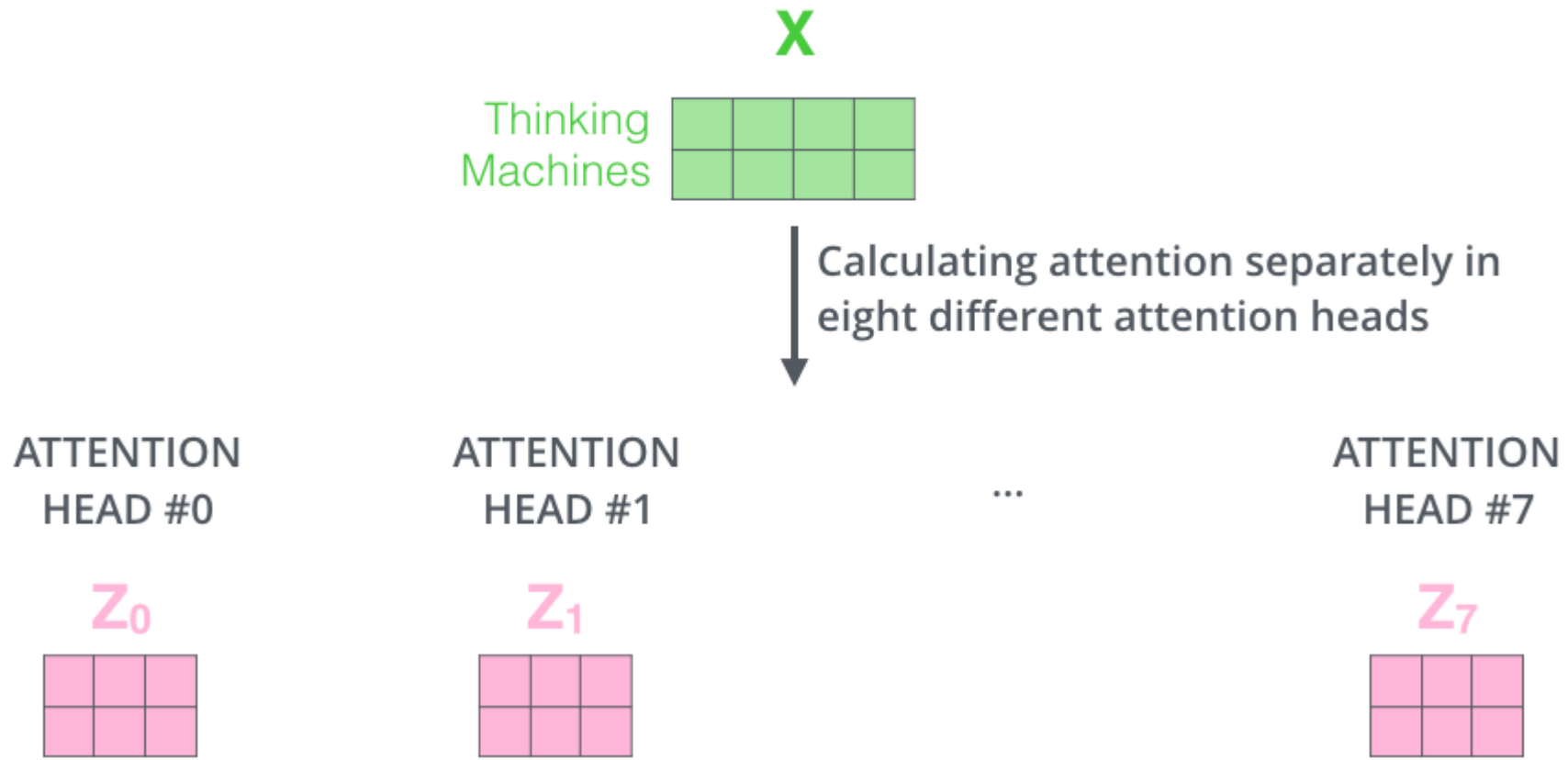
- Let's say we have a sentence:
  "The cat sat on the mat.“

- A standard self-attention mechanism might focus on **"sat"** when processing **"cat"**.

- However, in **multi-head attention**, different heads might focus on different words:
  - **Head 1:** Focuses on **"sat"** (verb association)
  - **Head 2:** Focuses on **"mat"** (object of action)
  - **Head 3:** Focuses on **"the"** (determiner)



with multi-headed attention we have not only one, but multiple sets of Query/Key/Value weight matrices



If we do the same self-attention calculation we outlined above, just eight different times with different weight matrices, we end up with eight different Z matrices

Multi-Head Attention

Equation:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \text{head}_2, \ldots, \text{head}_h)W^O$$

Where each head is:

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

1) This is our input sentence*

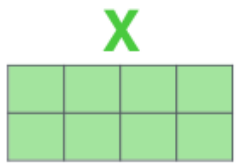
2) We embed each word*

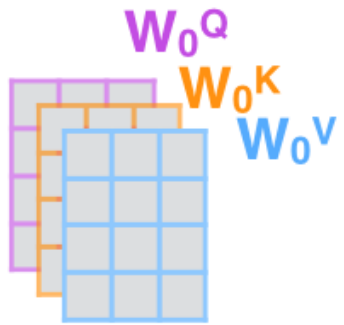
3) Split into 8 heads. We multiply X or R with weight matrices

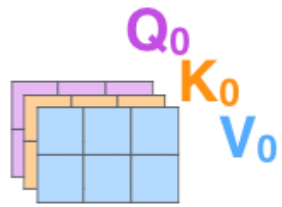
4) Calculate attention using the resulting Q/K/V matrices

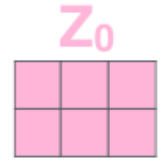
5) Concatenate the resulting Z matrices, then multiply with weight matrix W^O to produce the output of the layer

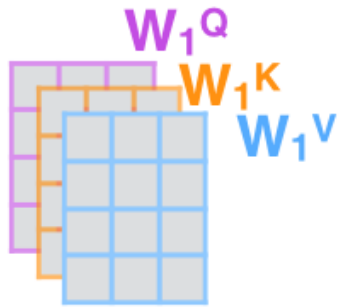
Thinking Machines

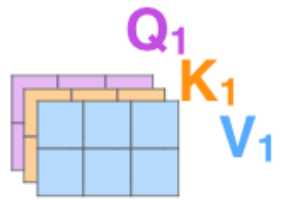


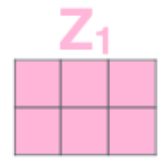


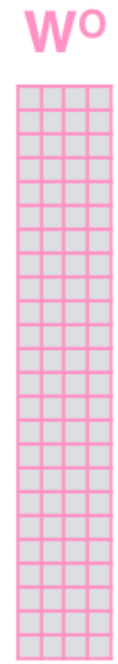


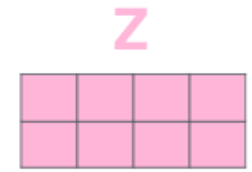




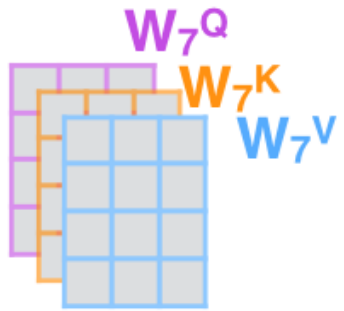


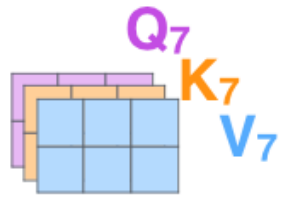


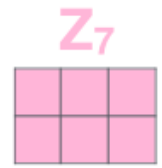




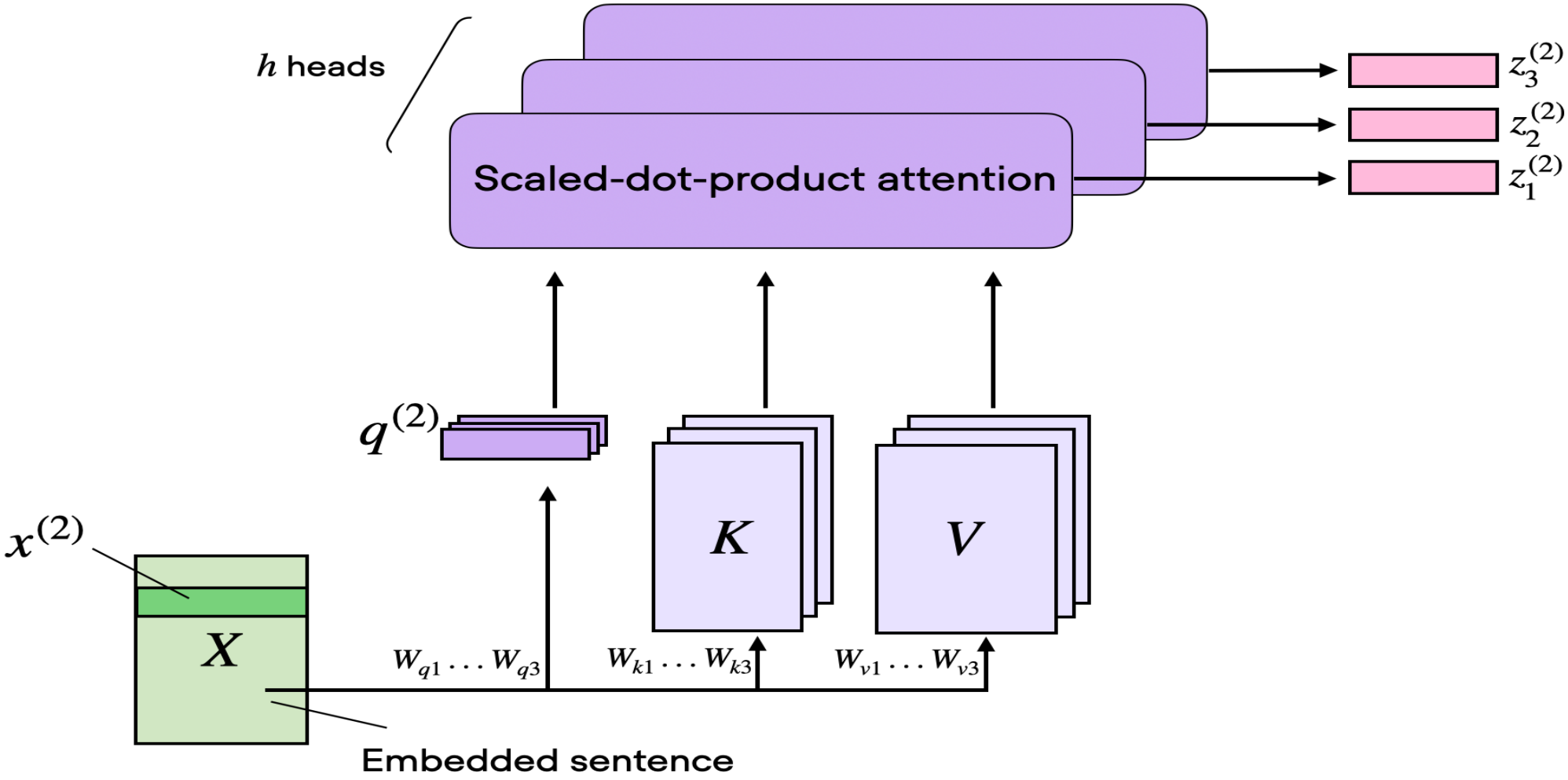
* In all encoders other than #0, we don't need embedding. We start directly with the output of the encoder right below this one





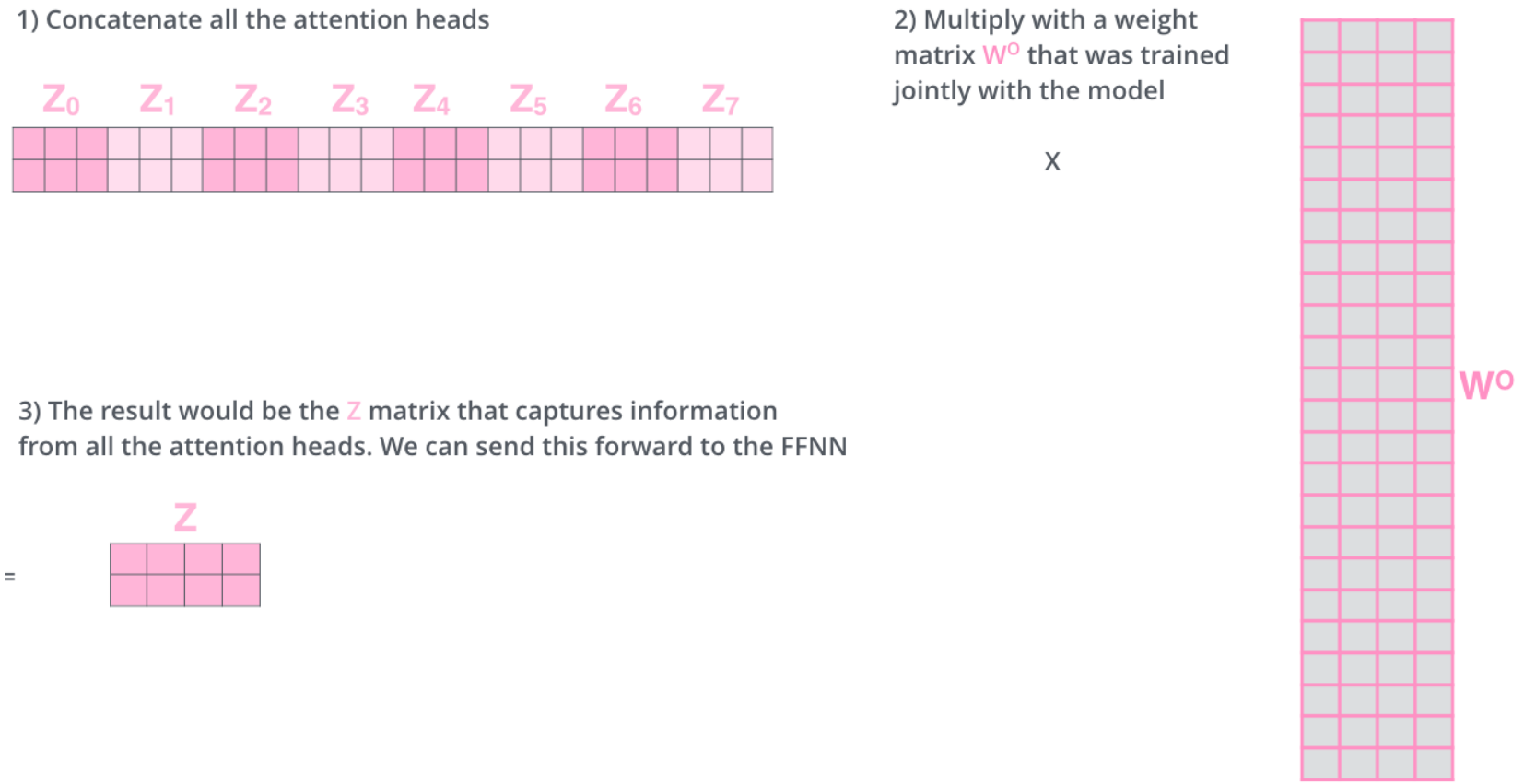


Multi-Head Attention



- The feed-forward layer is not expecting eight matrices – it's expecting a single matrix (a vector for each word). So we need a way to condense these eight down into a single matrix.

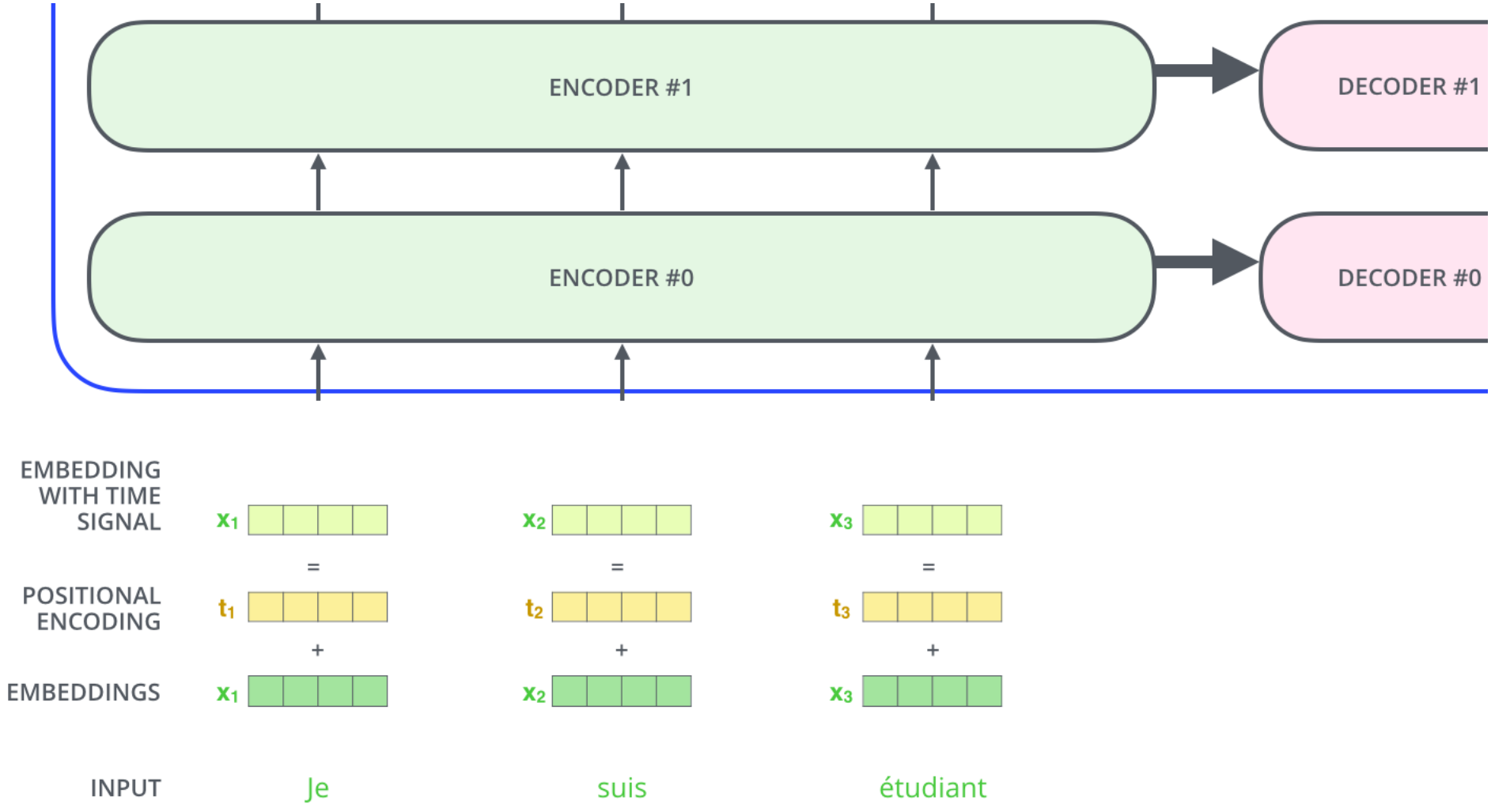
- How do we do that? We concat the matrices then multiply them by an additional weights matrix WO.



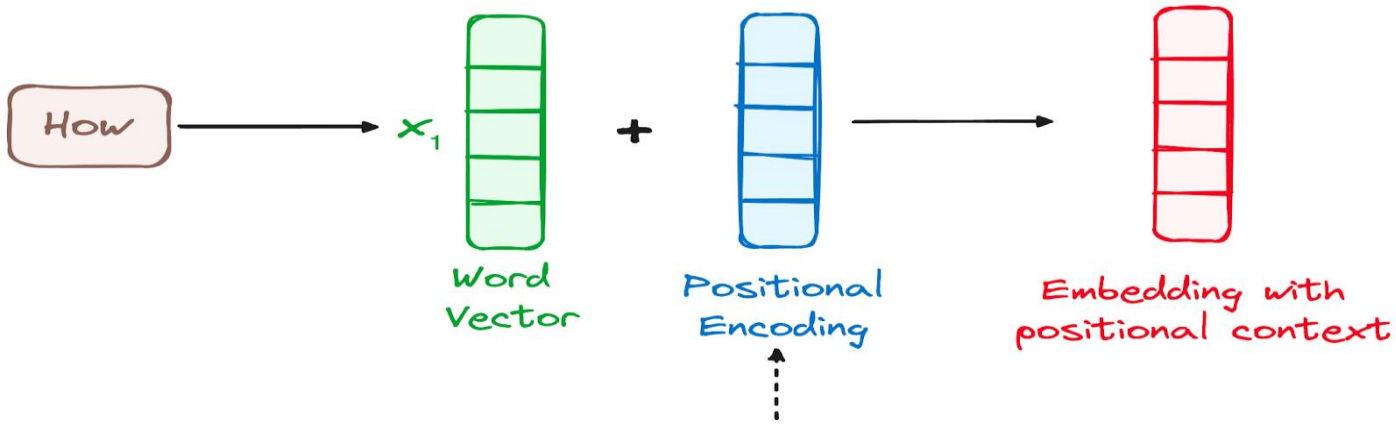
Positional Encoding (Representing The Order of The Sequence)

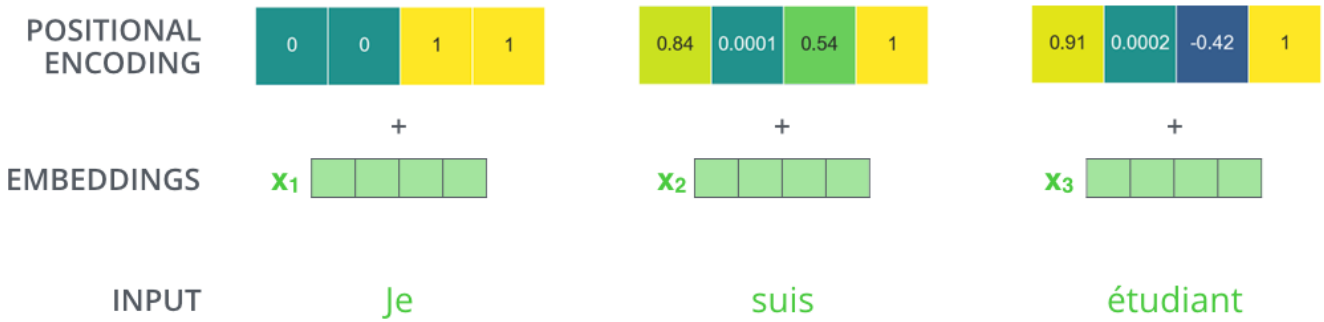
- Transformers lack an inherent sense of word order because they process input sequences in parallel.
- Positional Encoding assigns a unique vector to each position in the input sequence.
- These vectors are added to word embeddings to ensure that position information is incorporated into the model.

Positional Encoding (Representing The Order of The Sequence)



Sequence Models





Sequence Models

Mathematical Formula (Sinusoidal Encoding):

For position pos and dimension i in the embedding:

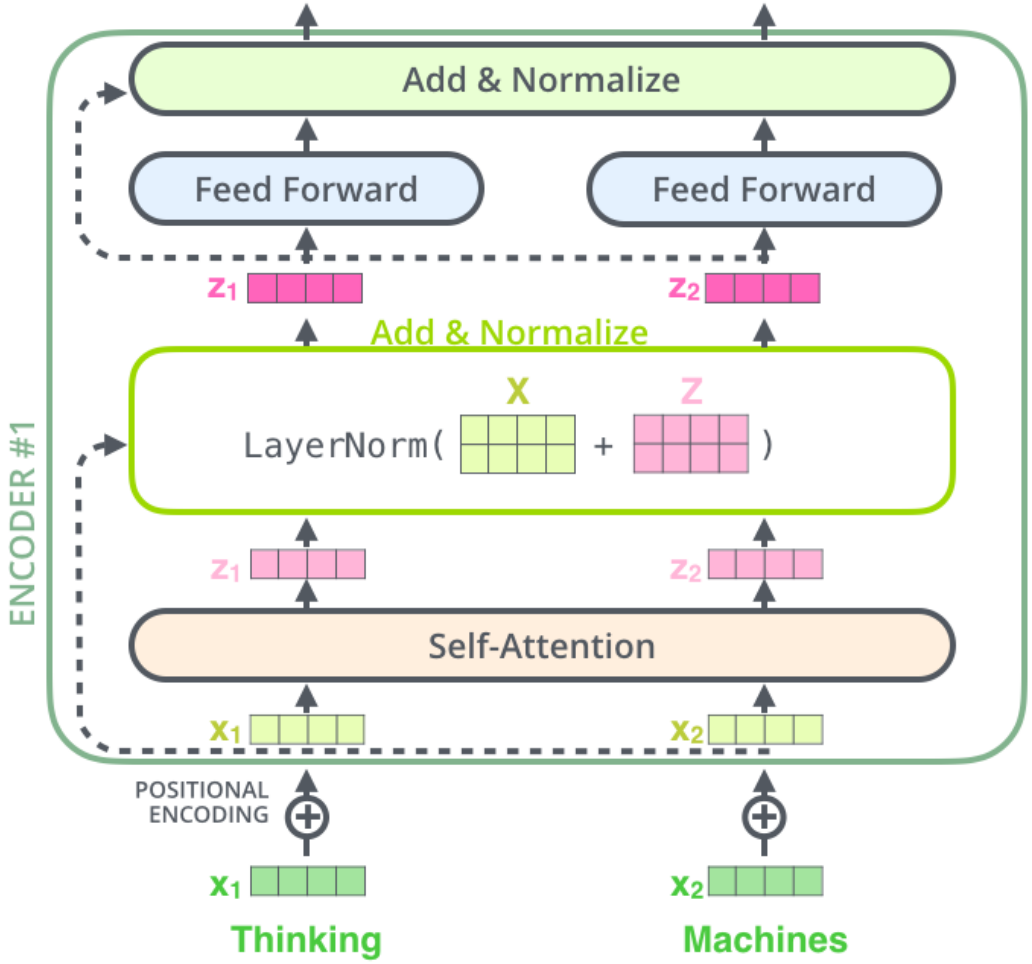
$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right)$$

$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right)$$

Where:

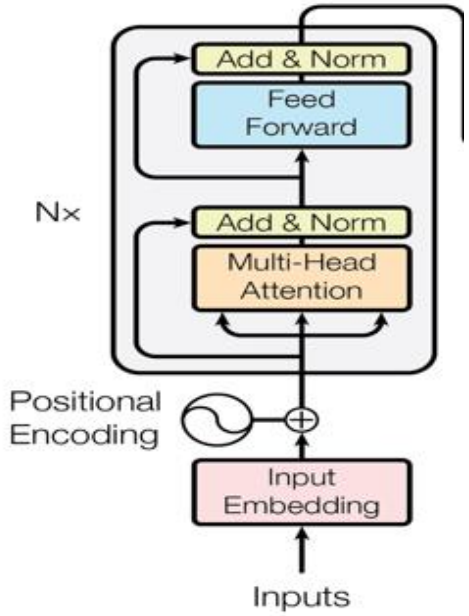
- pos: The position of the word in the sequence.
- i: The dimension index of the embedding vector.
- d_{model}: The total embedding dimension.

The Residuals

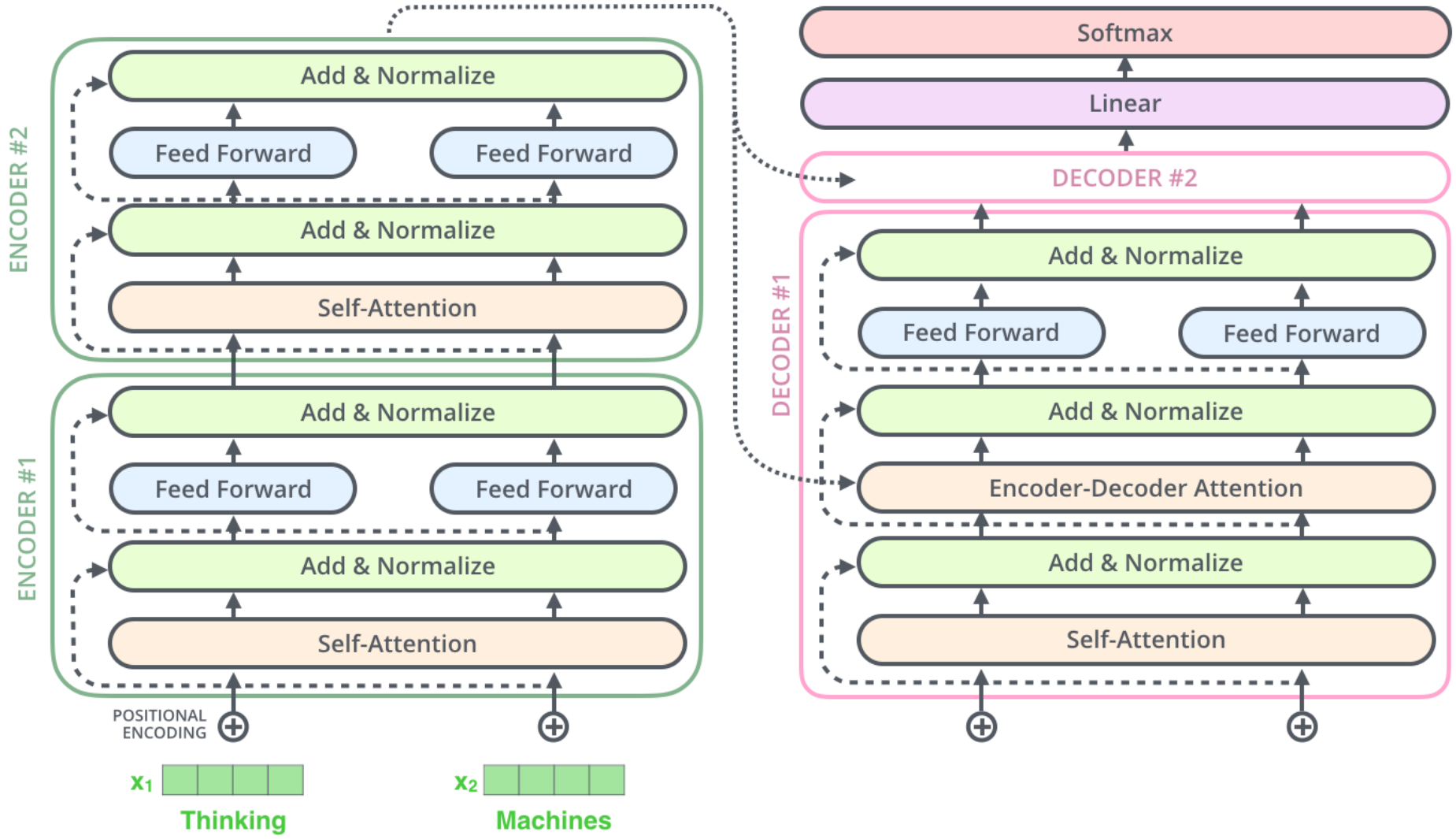


One detail in the architecture of the encoder that we need to mention before moving on, is that each sub-layer (self-attention, ffnn) in each encoder has a residual connection around it, and is followed by a layer-normalization step.

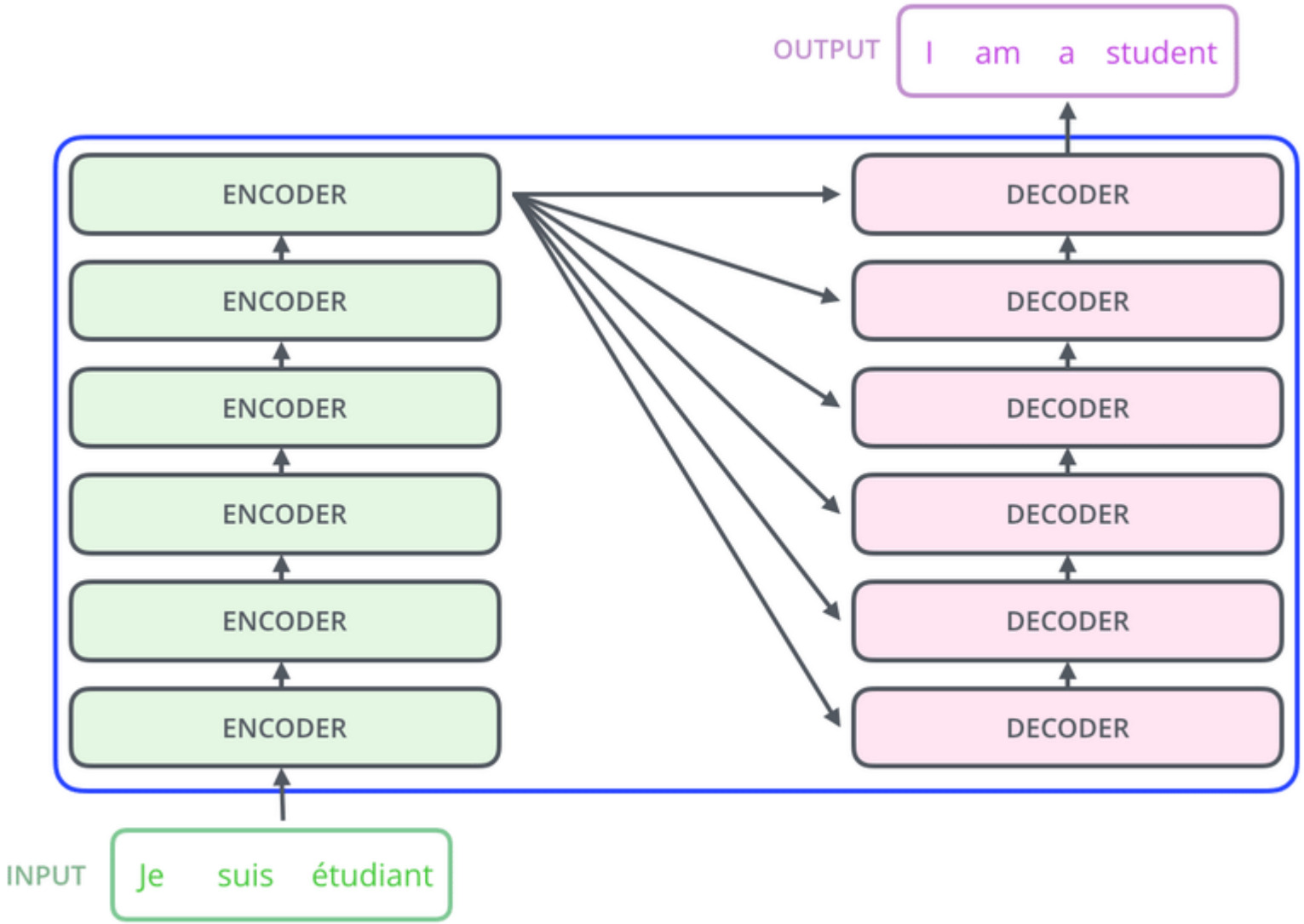
LN **computes mean and variance per training case (across all neurons in a layer at a single time step)**.



The Residuals



Decoder Side



Decoder of Transformer

- The Transformer decoder is **autoregressive at inference time and non-autoregressive at training time.**

- At inference time, the decoder works in an auto-regressive manner, meaning **it generates each tok in a sequence by using the tokens generated so fa**

- In contrast, during training, the decoder doesn't re on its own previous outputs but **uses the actual target sequence,** which allows it to be faster and more efficient.

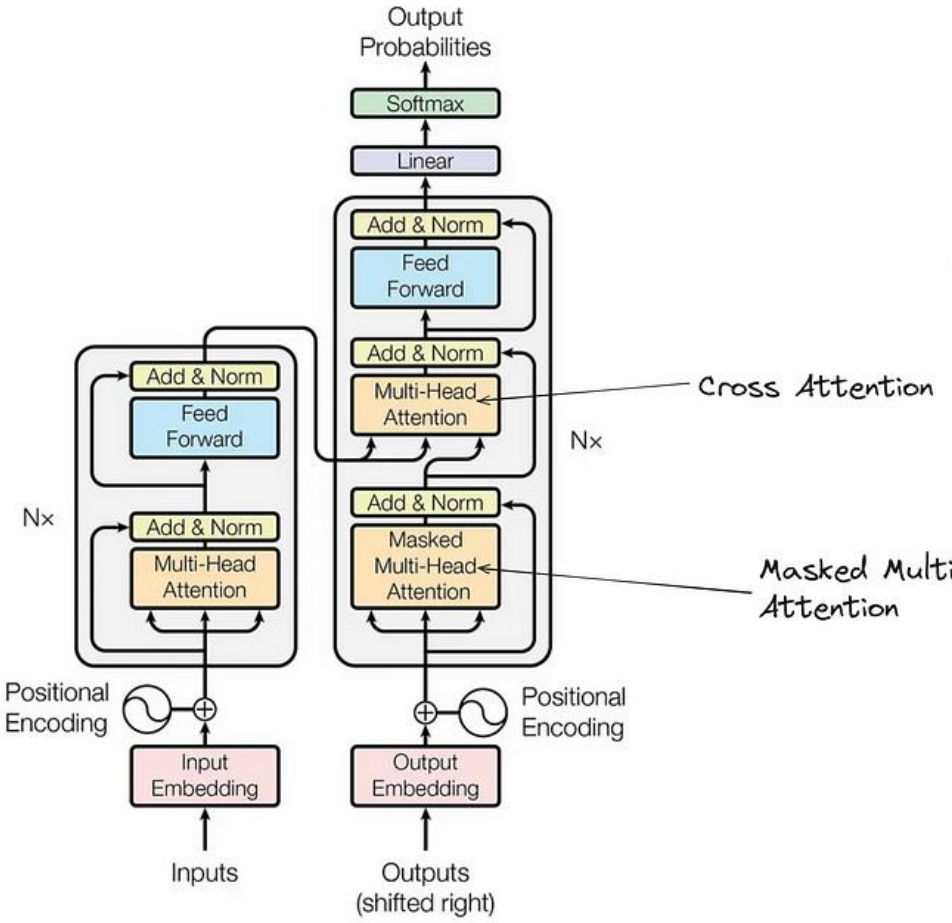
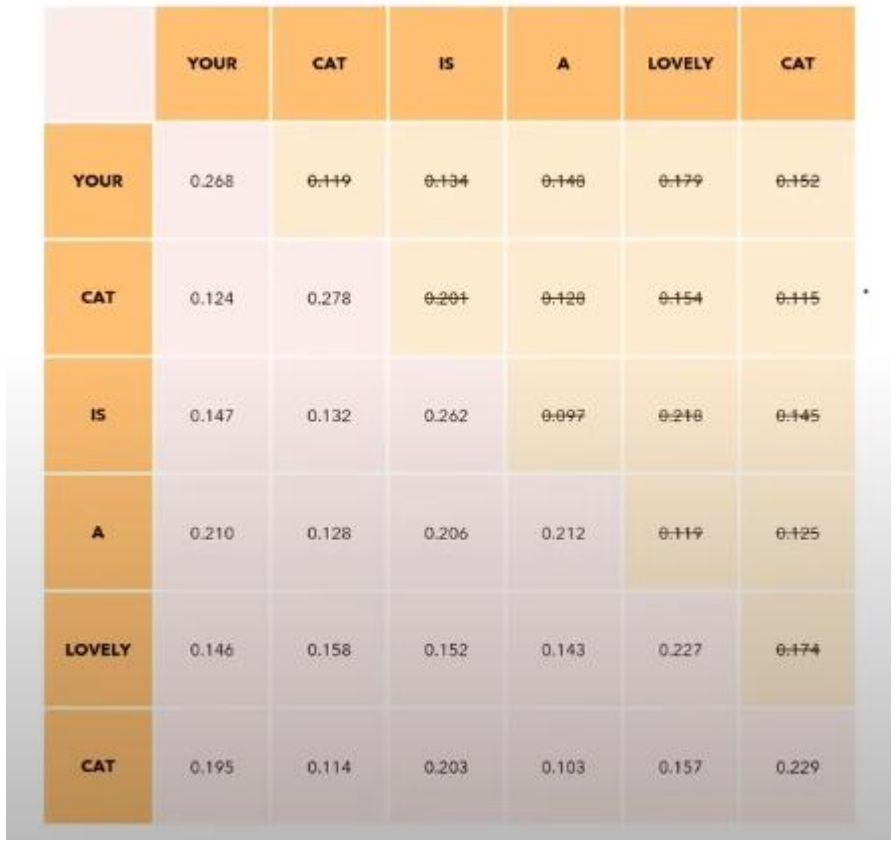


Figure 1: The Transformer - model architecture.

Decoder of Transformer



	YOUR	CAT	IS	A	LOVELY	CAT
YOUR	0.268	~~0.119~~	~~0.134~~	~~0.148~~	~~0.179~~	~~0.152~~
CAT	0.124	0.278	~~0.201~~	~~0.128~~	~~0.154~~	~~0.115~~
IS	0.147	0.132	0.262	~~0.097~~	~~0.218~~	~~0.145~~
A	0.210	0.128	0.206	0.212	~~0.119~~	~~0.125~~
LOVELY	0.146	0.158	0.152	0.143	0.227	~~0.174~~
CAT	0.195	0.114	0.203	0.103	0.157	0.229

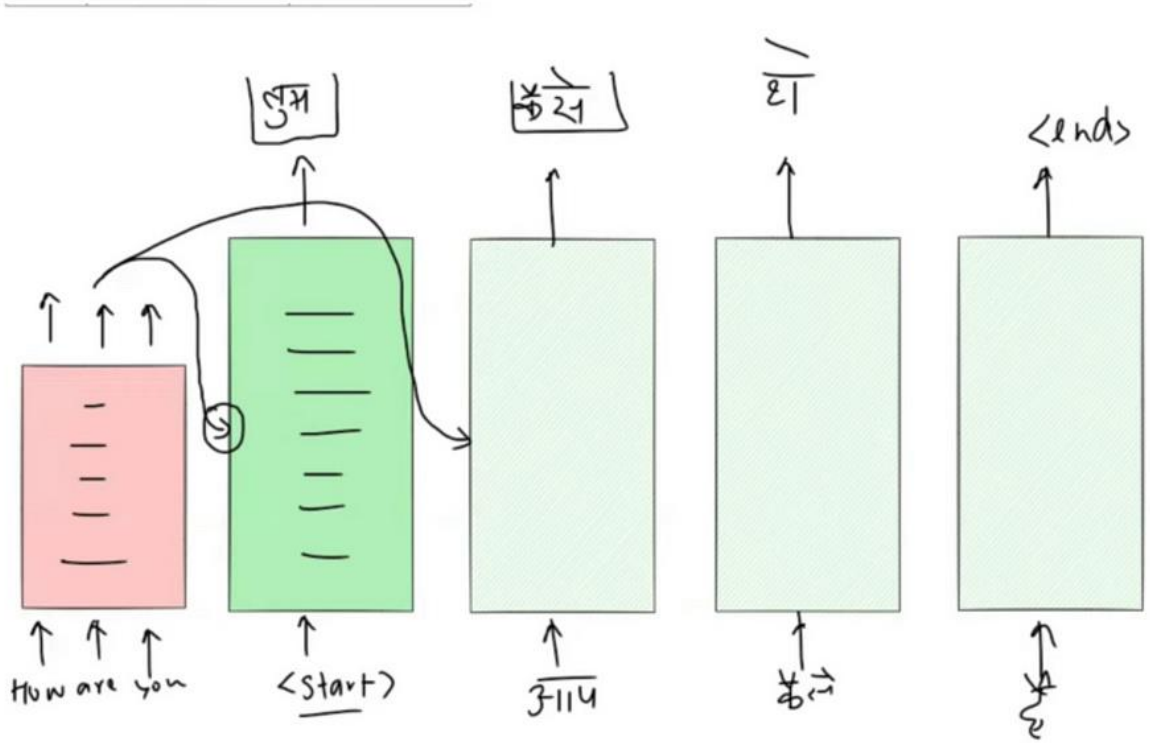
Why Masking?

In tasks like text generation or language modeling, predicting a word must only depend on previous words.

Masking ensures that when a word t is being predicted, the model only attends to tokens 1 through t−1. Future tokens (like t+1 or t+2) are hidden.

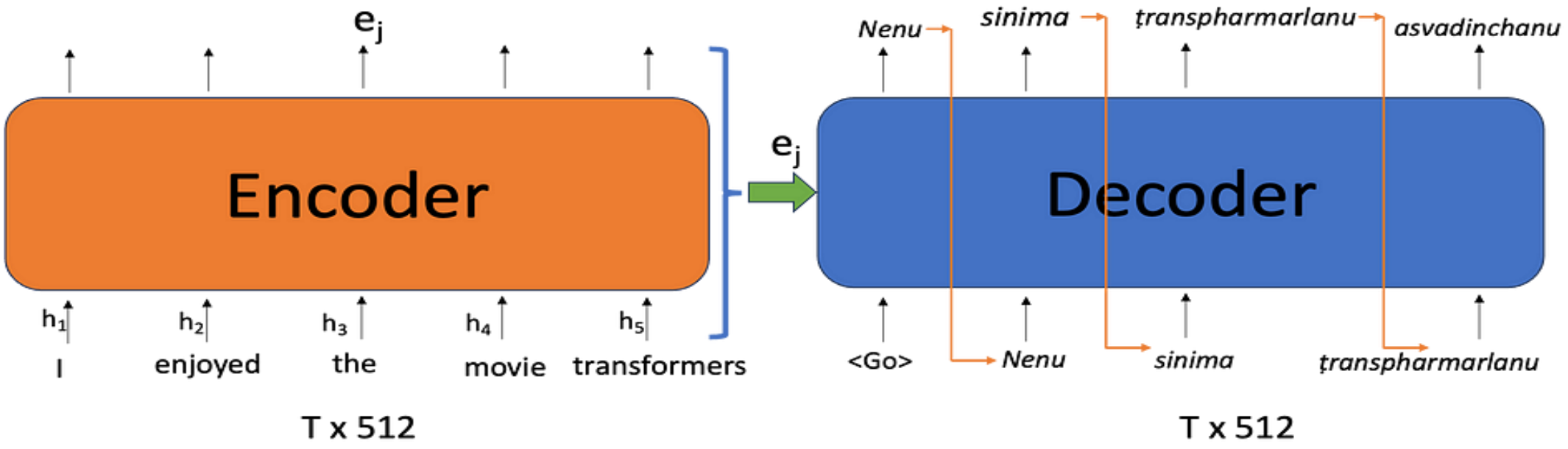
It ensures that each word (or token) in a sequence only attends to previous words and does not "see" future words, preventing information leakage.

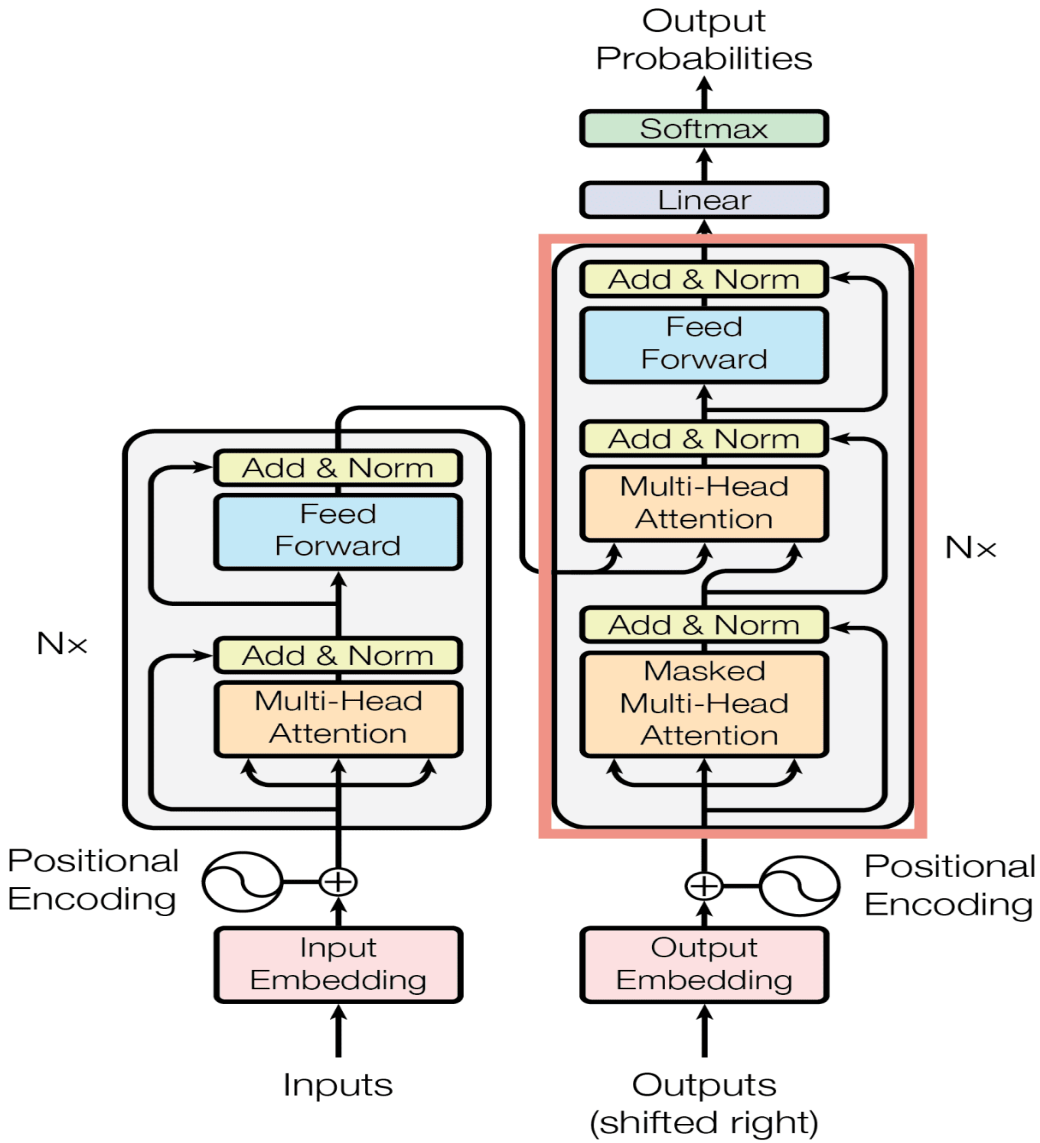
Teacher Forcing



- Teacher forcing involves feeding the **true output** from the previous time step as the next input to the model, instead of feeding the model's own prediction.
- The decoder continues, and we keep using the correct outputs at each time step. This ensures that the training process is smoother and doesn't get derailed by incorrect predictions early on.

Teacher Forcing in Sequence Models




Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
Nx
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Nx
Positional Encoding
Positional Encoding
Input Embedding
Output Embedding
Inputs
Outputs (shifted right)

Sequence Models

1. Cross-Entropy Loss Function

Cross-entropy loss is commonly used for **classification problems** like machine translation, where the model's output at each time step is a probability distribution over all possible words in the vocabulary.

Mathematical Formula:

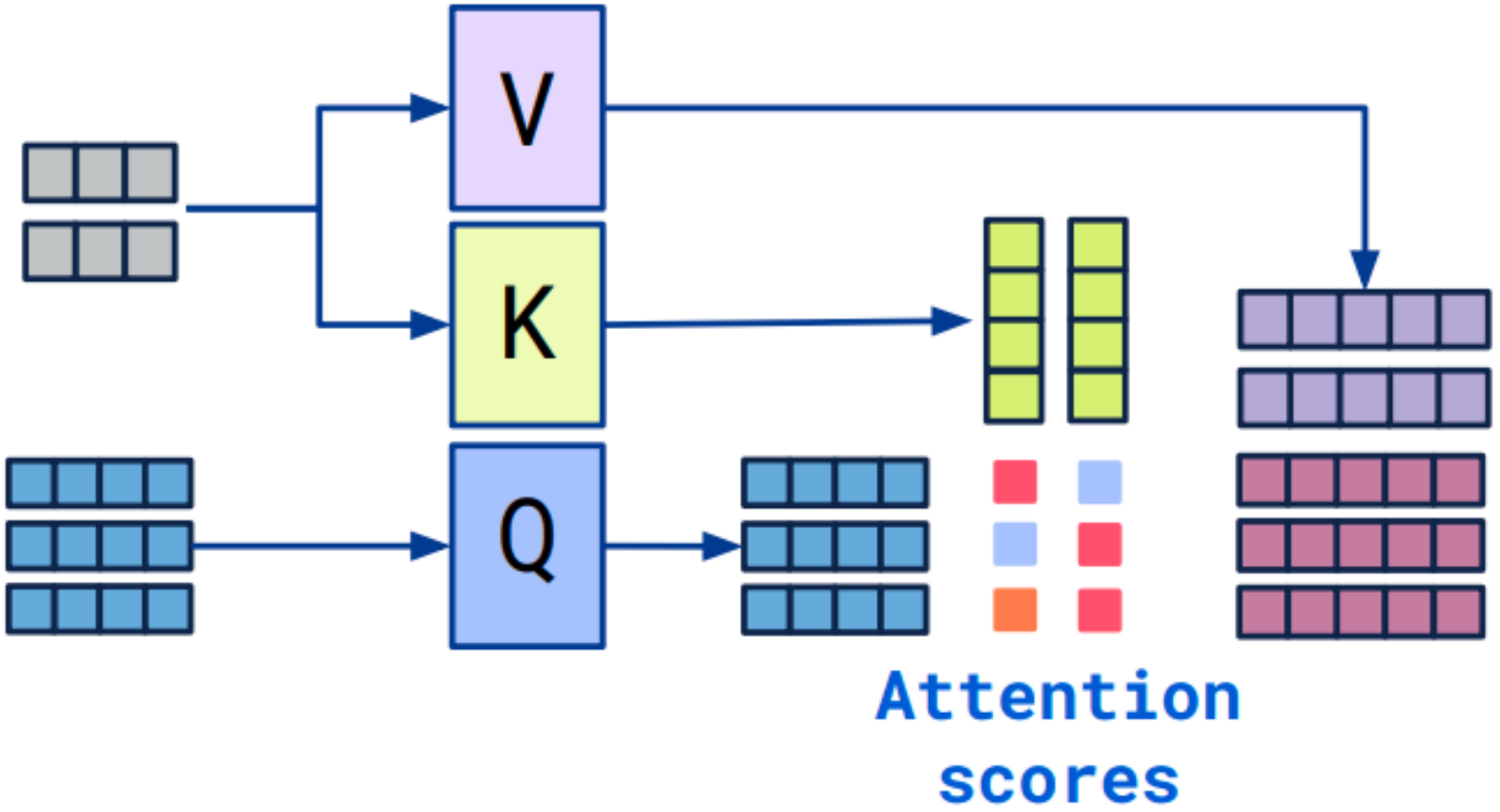
For a sequence prediction task, the cross-entropy loss is calculated as:

$$L = -\sum_{t=1}^{T}\sum_{v=1}^{V} y_{t,v}\log(p_{t,v})$$

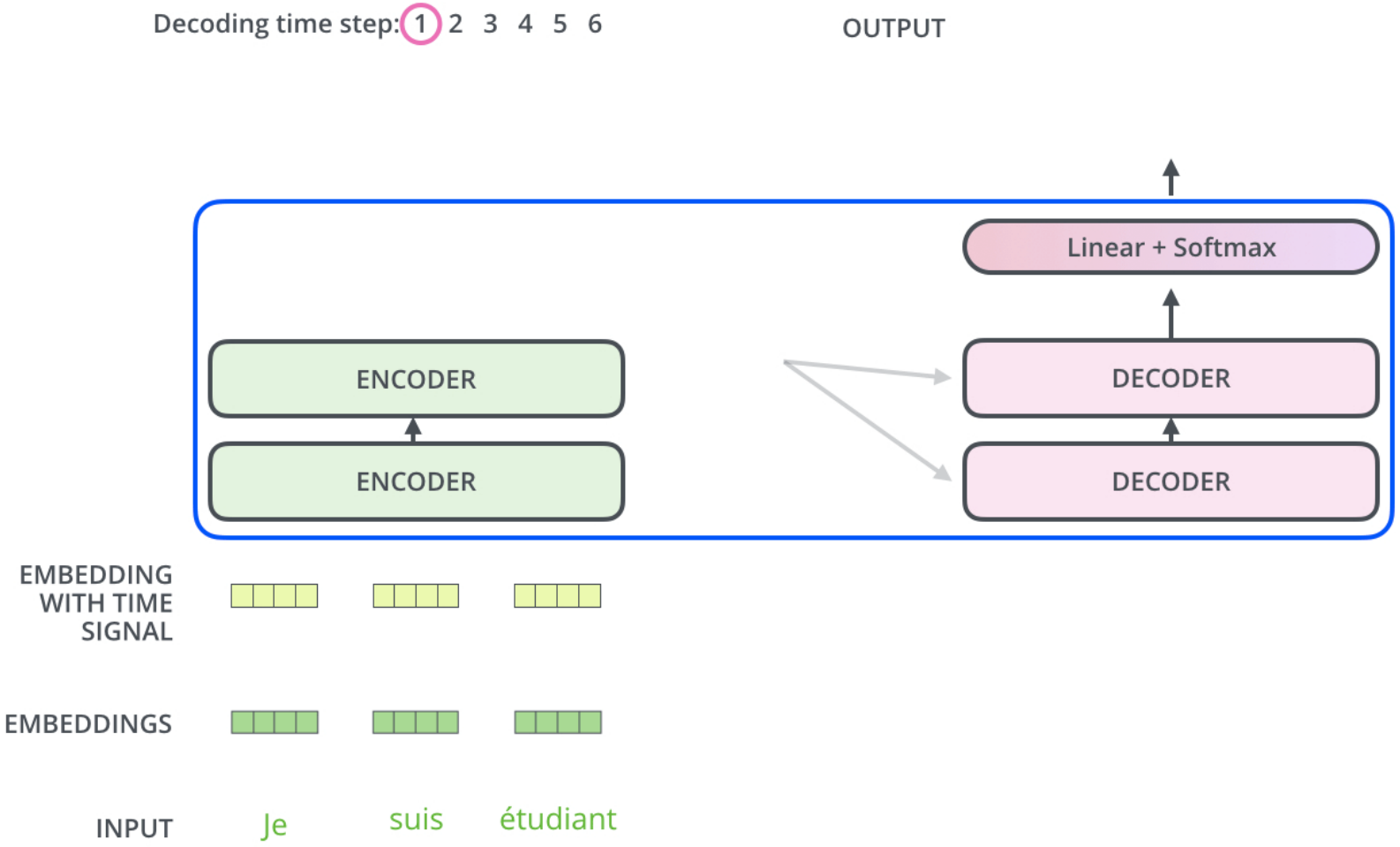
Where:

- T is the length of the target sequence.
- V is the size of the vocabulary.
- $y_{t,v}$ is the true probability distribution (one-hot encoded) for the word at time step t.
- $p_{t,v}$ is the predicted probability distribution for the word at time step t for vocabulary item v.

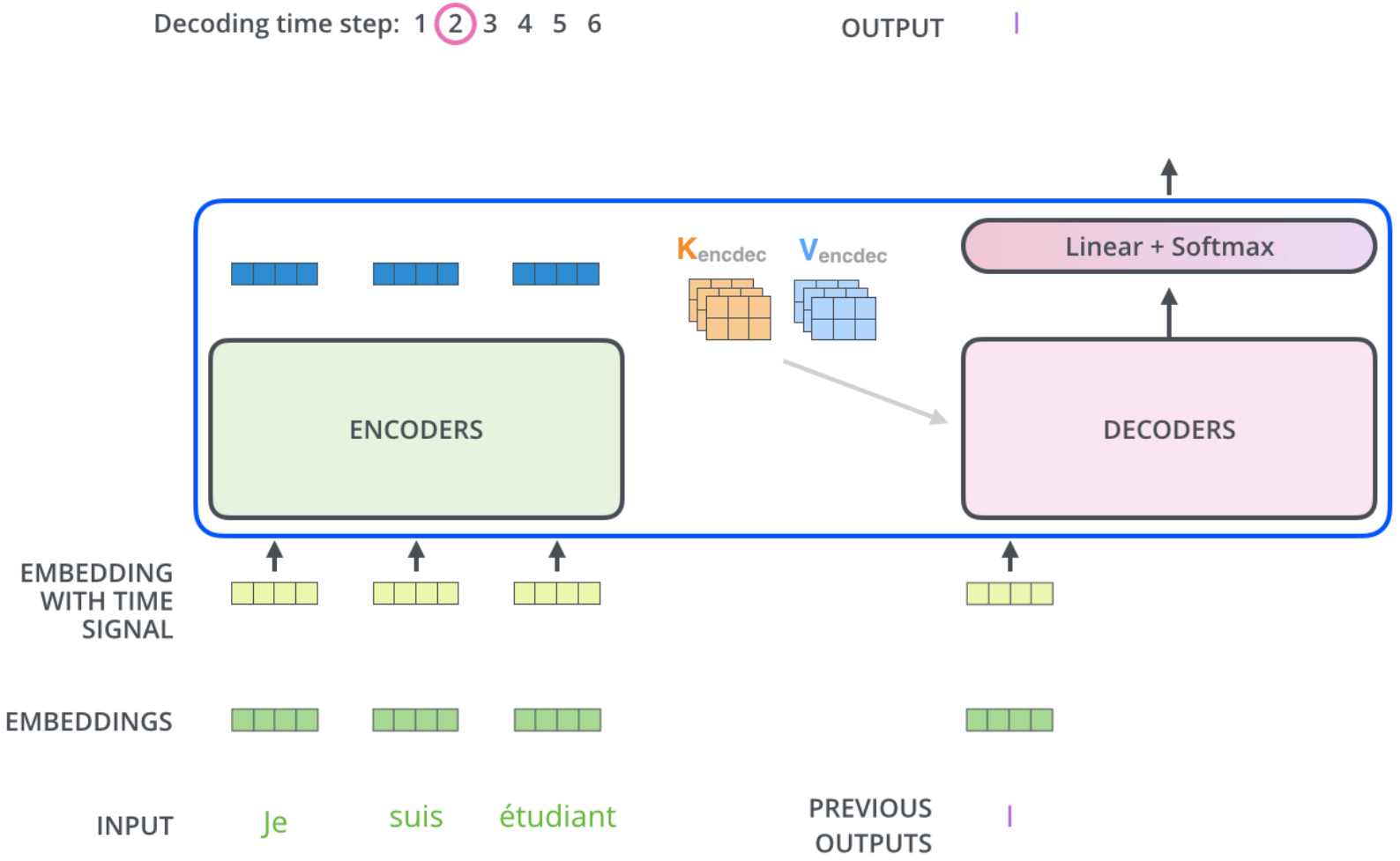
Cross Attention



The Decoder Side

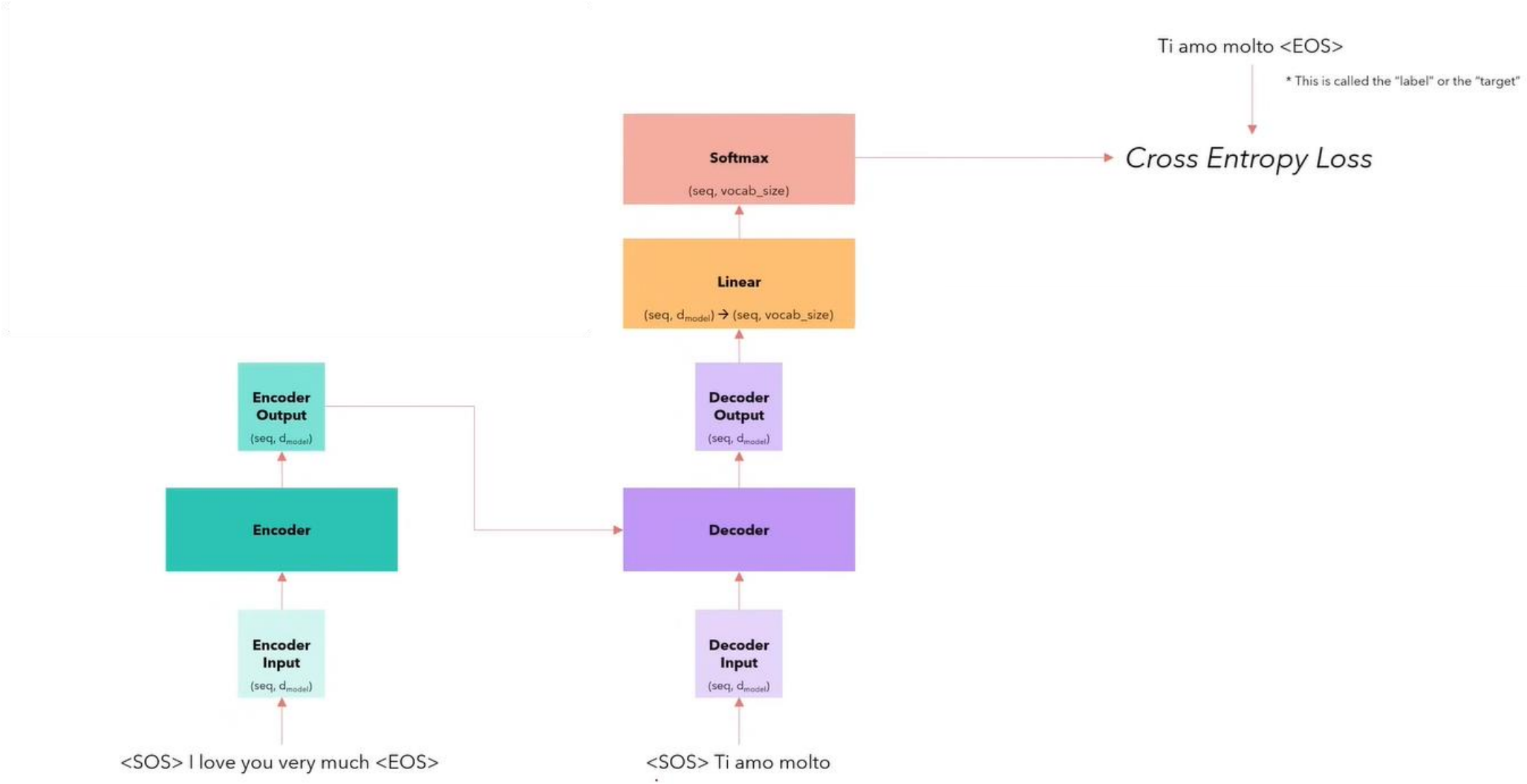


The Decoder Side



The output of each step is fed to the bottom decoder in the next time step, and the decoders bubble up their decoding results just like the encoders did.

Training


Ti amo molto <EOS>
* This is called the "label" or the "target"
Softmax
(seq, vocab_size)
Cross Entropy Loss
Linear
(seq, d_model) → (seq, vocab_size)
Encoder Output
(seq, d_model)
Decoder Output
(seq, d_model)
Encoder
Decoder
Encoder Input
(seq, d_model)
Decoder Input
(seq, d_model)
<SOS> I love you very much <EOS>
<SOS> Ti amo molto

Lab- Session

https://colab.research.google.com/drive/12Q0Mle1BOv0J2Zj8Hof6gTcEijwfWBtY?usp=sharing

Some practical approaches for engaging with transformer models:

1. Use pre-trained models
2. Fine-Tuning a Pre-trained Transformer Model
3. Building Custom Transformer Models:

   You can build a transformer model from scratch using PyTorch.

Major open-source Pre-trained Transformer models

- BERT (Bidirectional Encoder Representations from Transformers)
- GPT (Generative Pre-trained Transformer)
- Vision Transformer (ViT)
- DALL·E
- LLaMA (Large Language Model Meta AI)
- OpenLLaMA

Hugging Face

- These open-source pre-trained Transformer models are hosted on platforms like Hugging Face, making them accessible for research, experimentation, and deployment.

References

- What are the query, key, and value vectors? | by RAHULRAJ P V | Medium
- How Transformers work in deep learning and NLP: an intuitive introduction | AI Summer
- Explain Cross-Attention and how is it different from Self-Attention? - AIML.com


Thank You!