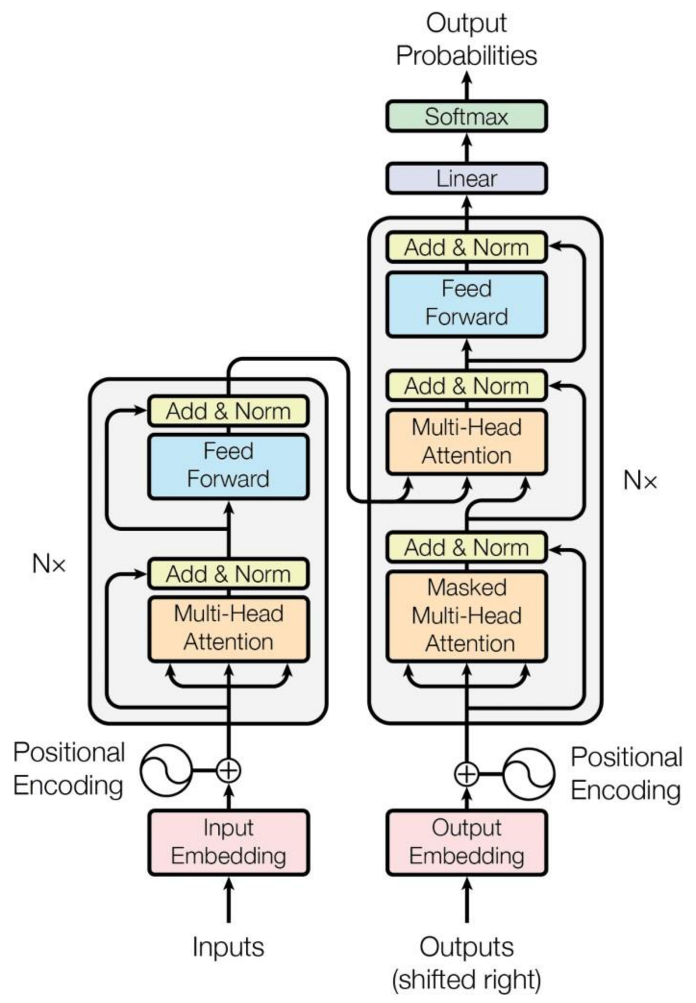
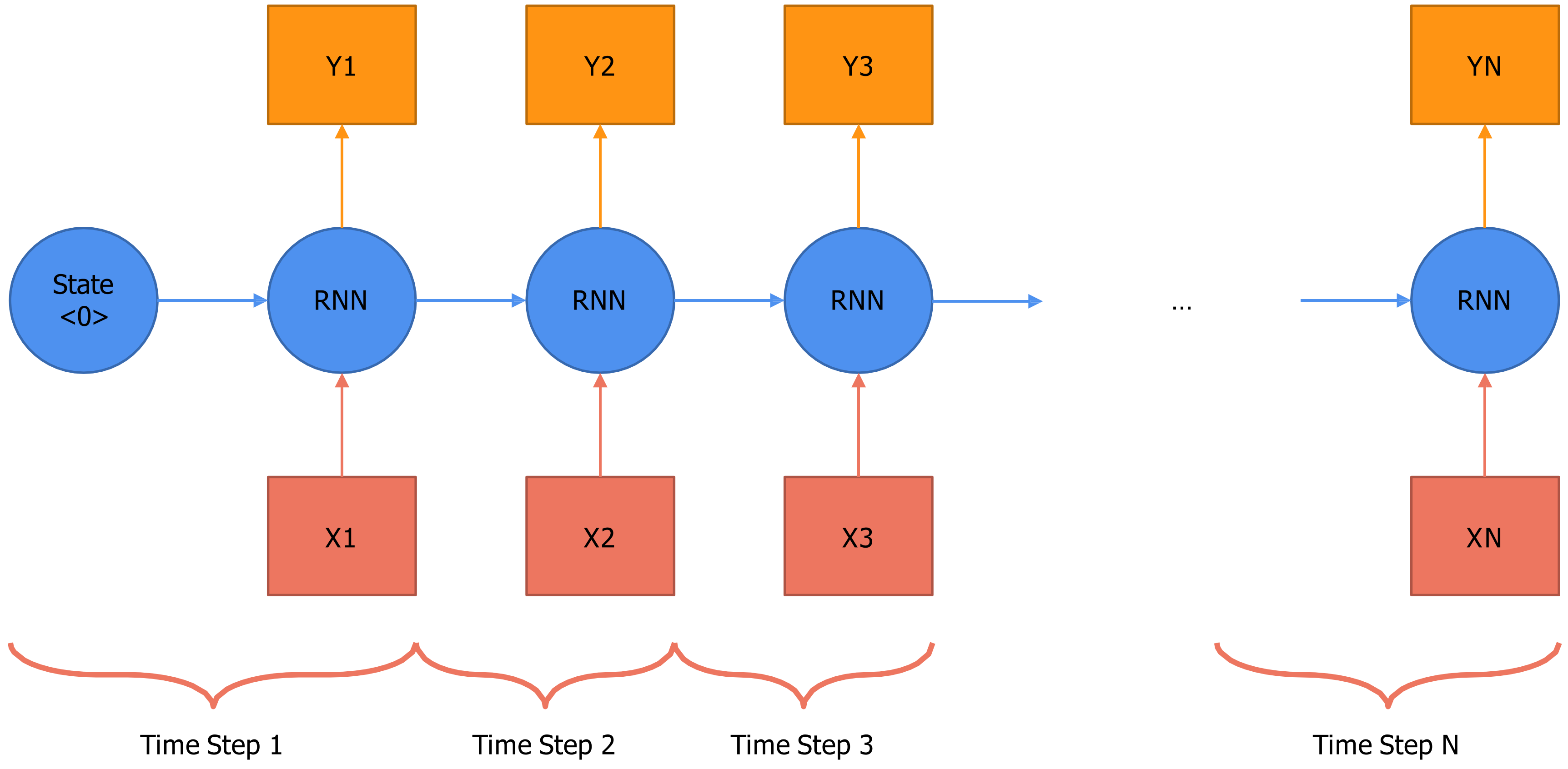


Transformers

Paper: Attention is all you need

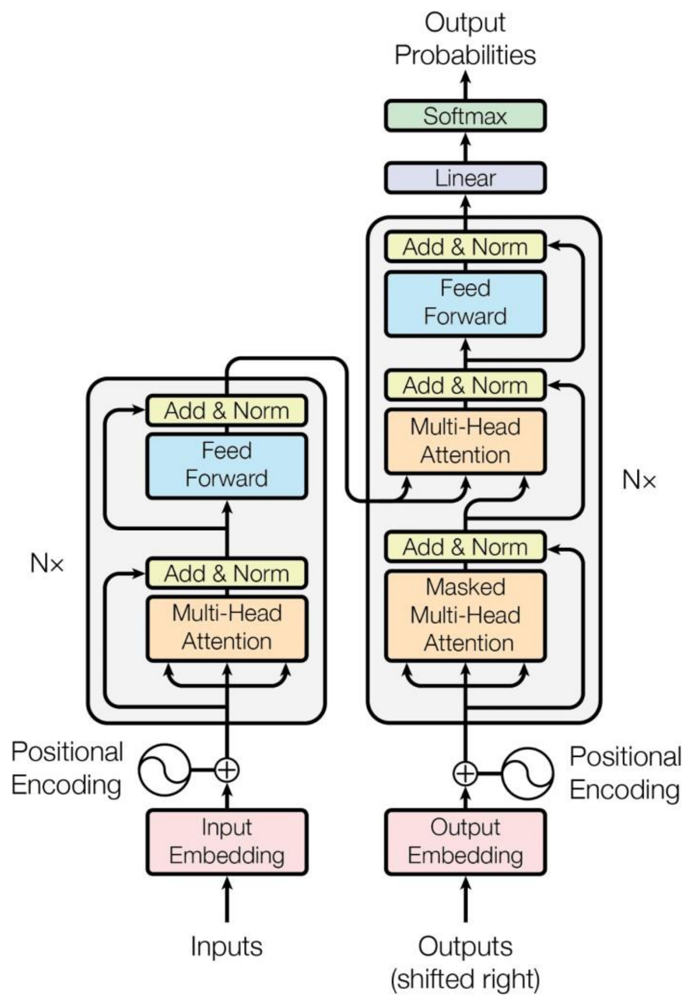
Recurrent Neural Networks (RNN)



Problems with RNN (among others)

1. Slow computation for long sequences
2. Vanishing or exploding gradients
3. Difficulty in accessing information from long time ago

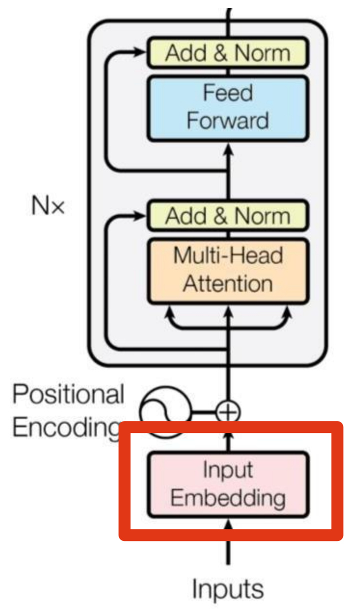
Transformers



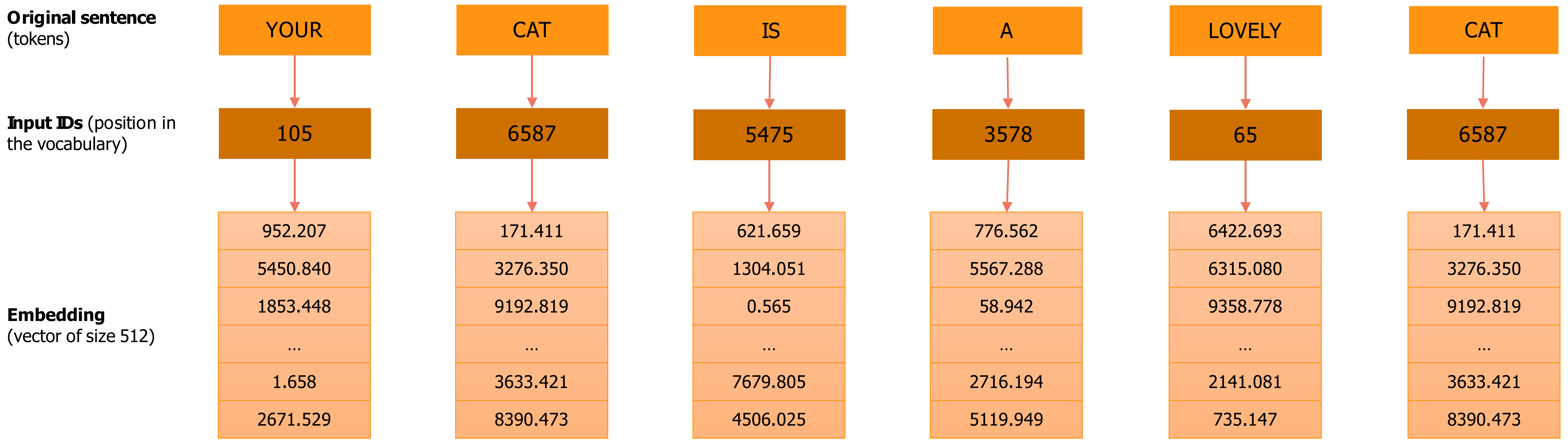
Notations

Input matrix (sequence, d_{model})

Encoder


Add & Norm
Feed Forward
Nx
Add & Norm
Multi-Head Attention
Positional Encoding
Input Embedding
Inputs

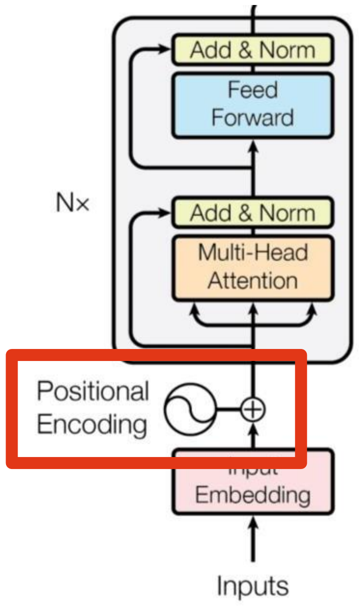
What is an input embedding?



Original sentence (tokens)	YOUR	CAT	IS	A	LOVELY	CAT
Input IDs (position in the vocabulary)	105	6587	5475	3578	65	6587
Embedding (vector of size 512)	952.207	171.411	621.659	776.562	6422.693	171.411
	5450.840	3276.350	1304.051	5567.288	6315.080	3276.350
	1853.448	9192.819	0.565	58.942	9358.778	9192.819
	…	…	…	…	…	…
	1.658	3633.421	7679.805	2716.194	2141.081	3633.421
	2671.529	8390.473	4506.025	5119.949	735.147	8390.473

We define d_{model} = 512, which represents the size of the embedding vector of each word

Encoder


Add & Norm
Feed Forward
Nx
Add & Norm
Multi-Head Attention
Positional Encoding
Input Embedding
Inputs

What is positional encoding?

- We want each word to carry some information about its position in the sentence.
- To inject information about the position or order of elements in a sequence
- We want the model to treat words that appear close to each other as “close” and words that are distant as “distant”.
- We want the positional encoding to represent a pattern that can be learned by the model.

What is positional encoding?

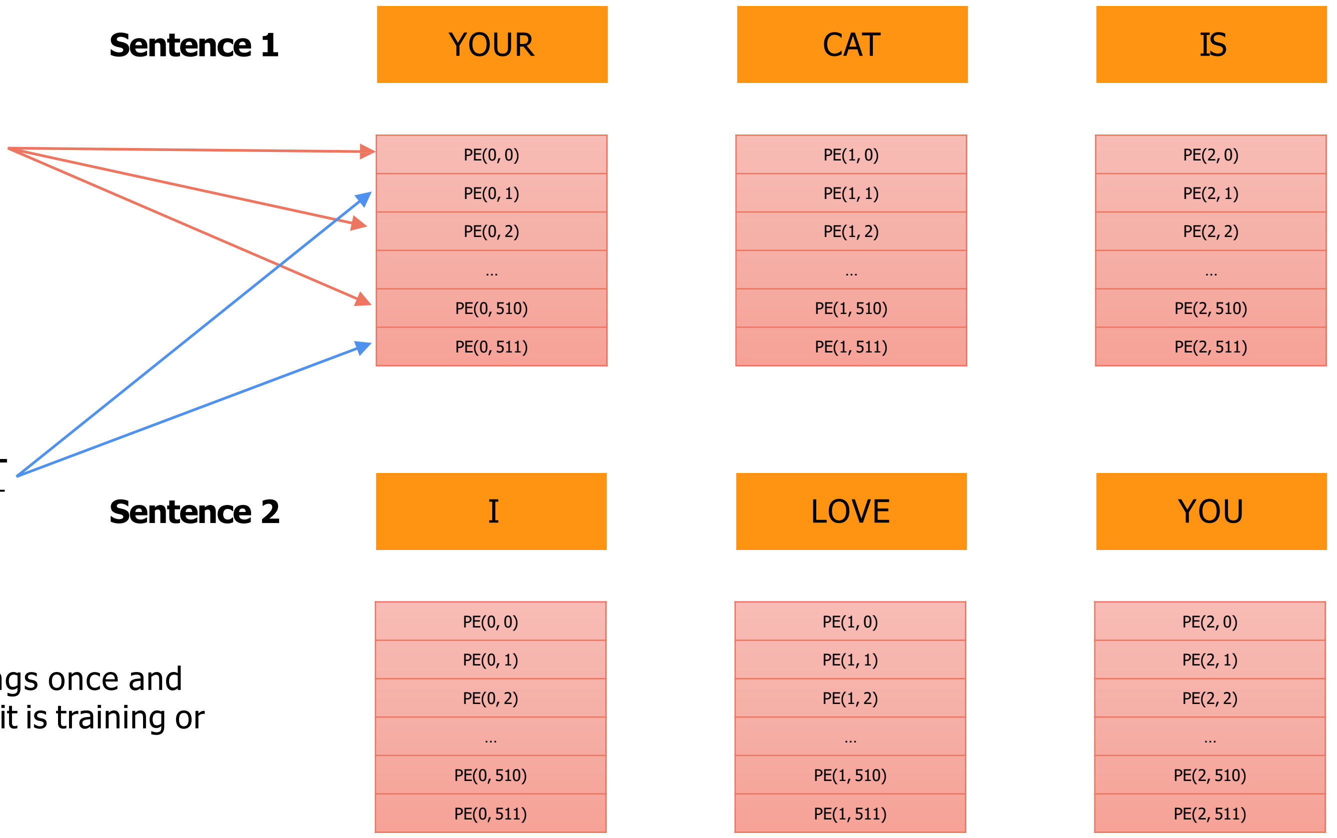
Original sentence	YOUR	CAT	IS	A	LOVELY	CAT
Embedding (vector of size 512)	952.207	171.411	621.659	776.562	6422.693	171.411
	5450.840	3276.350	1304.051	5567.288	6315.080	3276.350
	1853.448	9192.819	0.565	58.942	9358.778	9192.819
	…	…	…	…	…	…
	1.658	3633.421	7679.805	2716.194	2141.081	3633.421
	2671.529	8390.473	4506.025	5119.949	735.147	8390.473
	+	+	+	+	+	+
Position Embedding (vector of size 512). Only computed once and reused for every sentence during training and inference.	…	1664.068	…	…	…	1281.458
	…	8080.133	…	…	…	7902.890
	…	2620.399	…	…	…	912.970
	…	…	…	…	…	3821.102
	…	9386.405	…	…	…	1659.217
	…	3120.159	…	…	…	7018.620
	=	=	=	=	=	=
Encoder Input (vector of size 512)	…	1835.479	…	…	…	1452.869
	…	11356.483	…	…	…	11179.24
	…	11813.218	…	…	…	10105.789
	…	…	…	…	…	…
	…	13019.826	…	…	…	5292.638
	…	11510.632	…	…	…	15409.093

What is positional encoding?

These sinusoidal functions are designed to create unique and distinct positional embeddings for different positions in a sequence, allowing the model to distinguish between tokens based on their position. The formula is fixed and does not adapt to the specific content of the tokens.

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

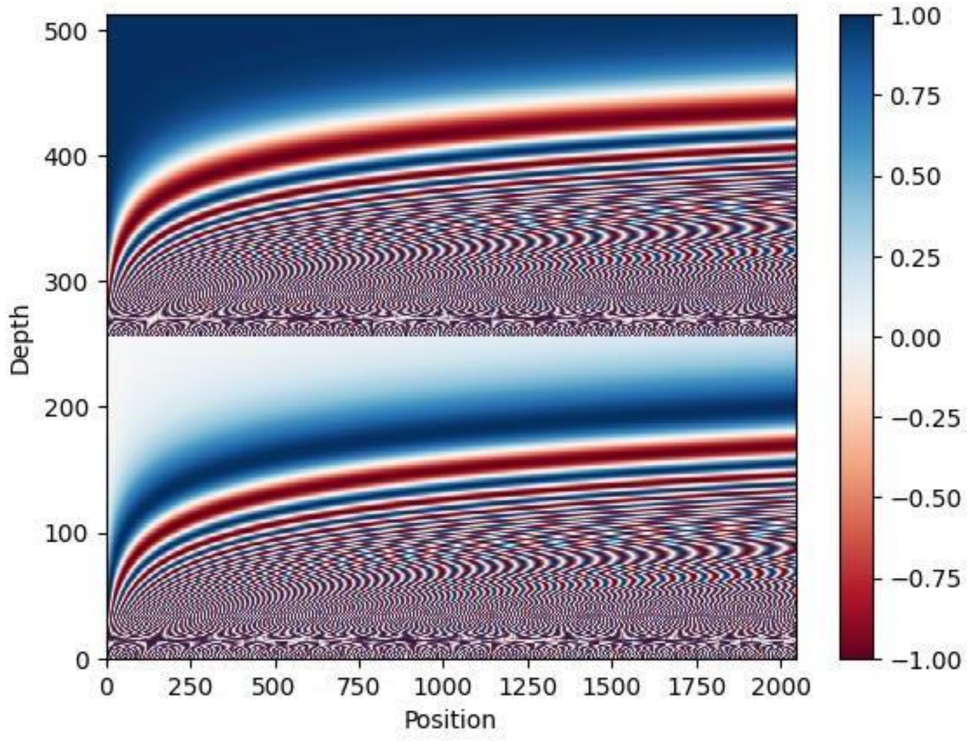
$$PE(pos, 2i+1) = \cos \frac{pos}{10000^{\frac{2i}{d_{model}}}}$$



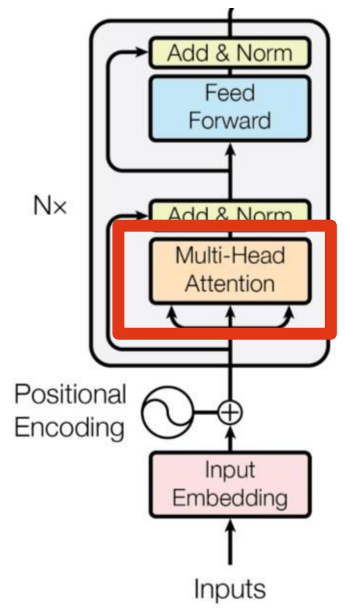
We only need to compute the positional encodings once and then reuse them for every sentence, no matter if it is training or inference.

Why trigonometric functions?

Trigonometric functions like **cos** and **sin** naturally represent a pattern that the model can recognize as continuous, so relative positions are easier to see for the model. By watching the plot of these functions, we can also see a regular pattern, so we can hypothesize that the model will see it too.



Encoder


Add & Norm
Feed Forward
Nx
Add & Norm
Multi-Head Attention
Positional Encoding
Input Embedding
Inputs

What is Self-Attention?

Self-Attention allows the model to relate words to each other.

In this simple case we consider the sequence length **seq** = 6 and $\mathbf{d}_{\mathbf{model}} = \mathbf{d}_{\mathbf{k}} = 512$.

The matrices **Q**, **K** and **V** are just the input sentence.

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

$$\text{softmax}\left(\frac{\underset{(6,\,512)}{\mathbf{Q}} \times \underset{(512,\,6)}{\mathbf{K^T}}}{\sqrt{512}}\right) =$$

	YOUR	CAT	IS	A	LOVELY	CAT	Σ
YOUR	0.268	0.119	0.134	0.148	0.179	0.152	1
CAT	0.124	0.278	0.201	0.128	0.154	0.115	1
IS	0.147	0.132	0.262	0.097	0.218	0.145	1
A	0.210	0.128	0.206	0.212	0.119	0.125	1
LOVELY	0.146	0.158	0.152	0.143	0.227	0.174	1
CAT	0.195	0.114	0.203	0.103	0.157	0.229	1

(6, 6)

* all values are random.

* for simplicity I considered only one head, which makes $d_{model} = d_k$.

How to compute Self-Attention?

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

	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

(6, 6)

X

V

(6, 512)

=

Attention

(6, 512)

Each row in this matrix captures not only the meaning (given by the embedding) or the position in the sentence (represented by the positional encodings) but also each word's interaction with other words.

Self-Attention in detail

- Self-Attention is permutation invariant since it does not cater the order of words, and treats each word independently
  (if you do not consider the contribution of the positional encoding)
- Self-Attention requires no parameters. Up to now the interaction between words has been driven by their embedding and the positional encodings. This will change later.
- We expect values along the diagonal to be the highest.
- If we don't want some positions to interact, we can always set their values to $-\infty$ before applying the *softmax* in this matrix and the model will not learn those interactions. We will use this in the decoder.

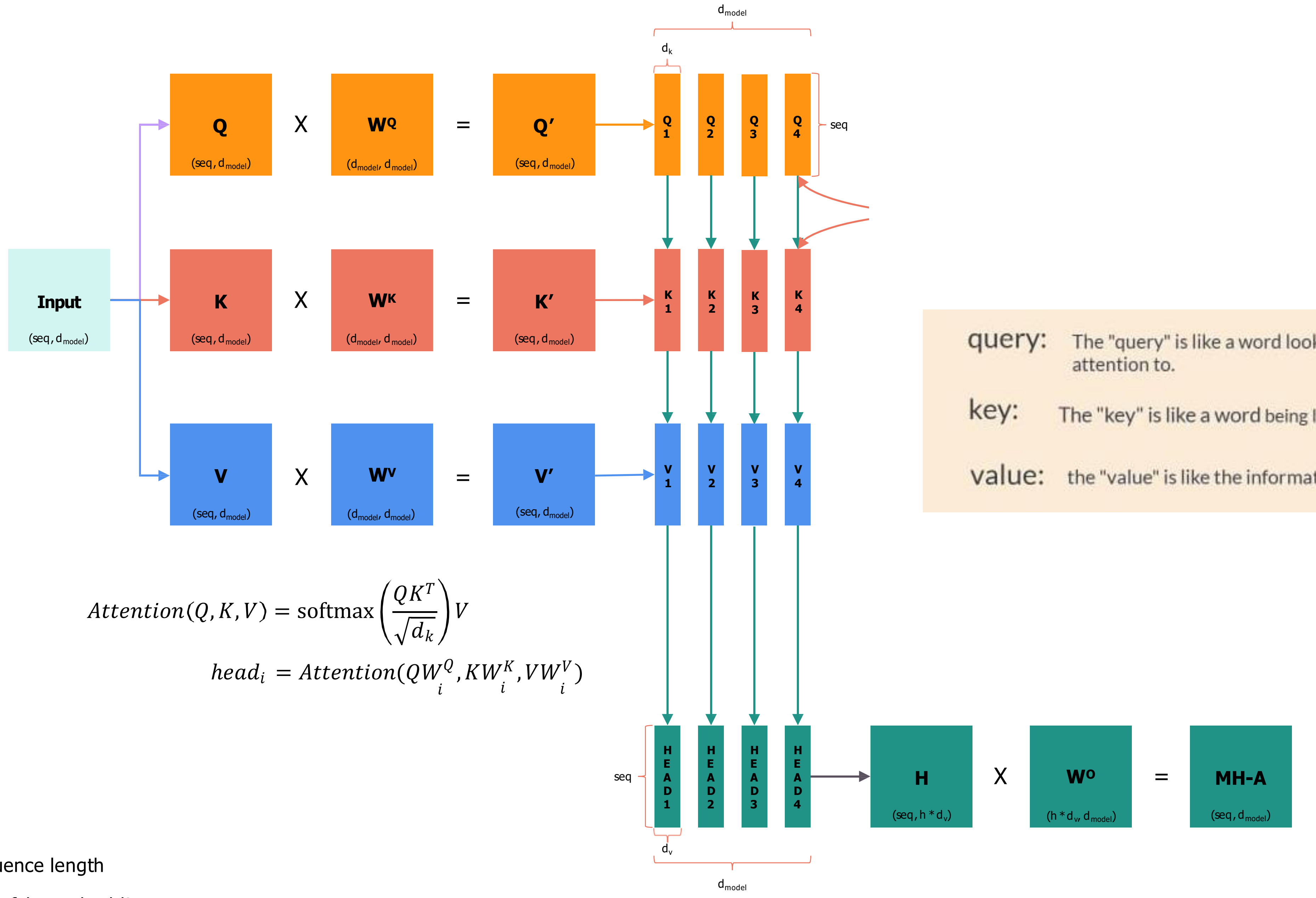
	YOUR	CAT	IS	A	LOVELY	CAT
YOUR	0.268	0.119	0.134	0.148	0.179	0.152
CAT	0.124	0.278	0.201	0.128	0.154	0.115
IS	0.147	0.132	0.262	0.097	0.218	0.145
A	0.210	0.128	0.206	0.212	0.119	0.125
LOVELY	0.146	0.158	0.152	0.143	0.227	0.174
CAT	0.195	0.114	0.203	0.103	0.157	0.229

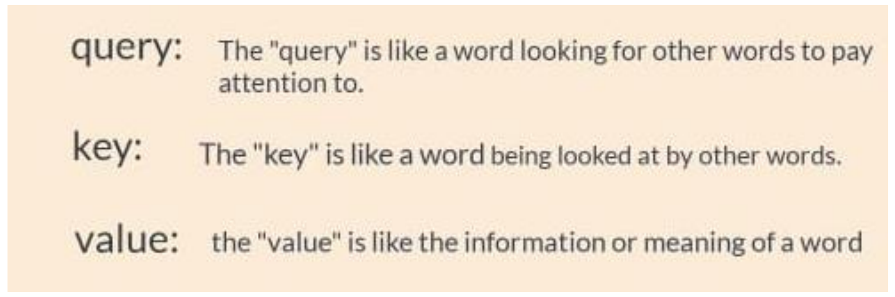
Multi-head Attention

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

$$MultiHead(Q, K, V) = Concat(head_1 \ \ldots head_h)W^O$$

$$head_i = Attention(QW^Q, KW_i^K, VW^V)$$

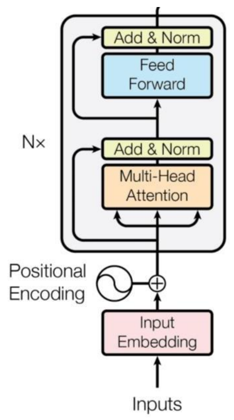




query: The "query" is like a word looking for other words to pay attention to.

key: The "key" is like a word being looked at by other words.

value: the "value" is like the information or meaning of a word

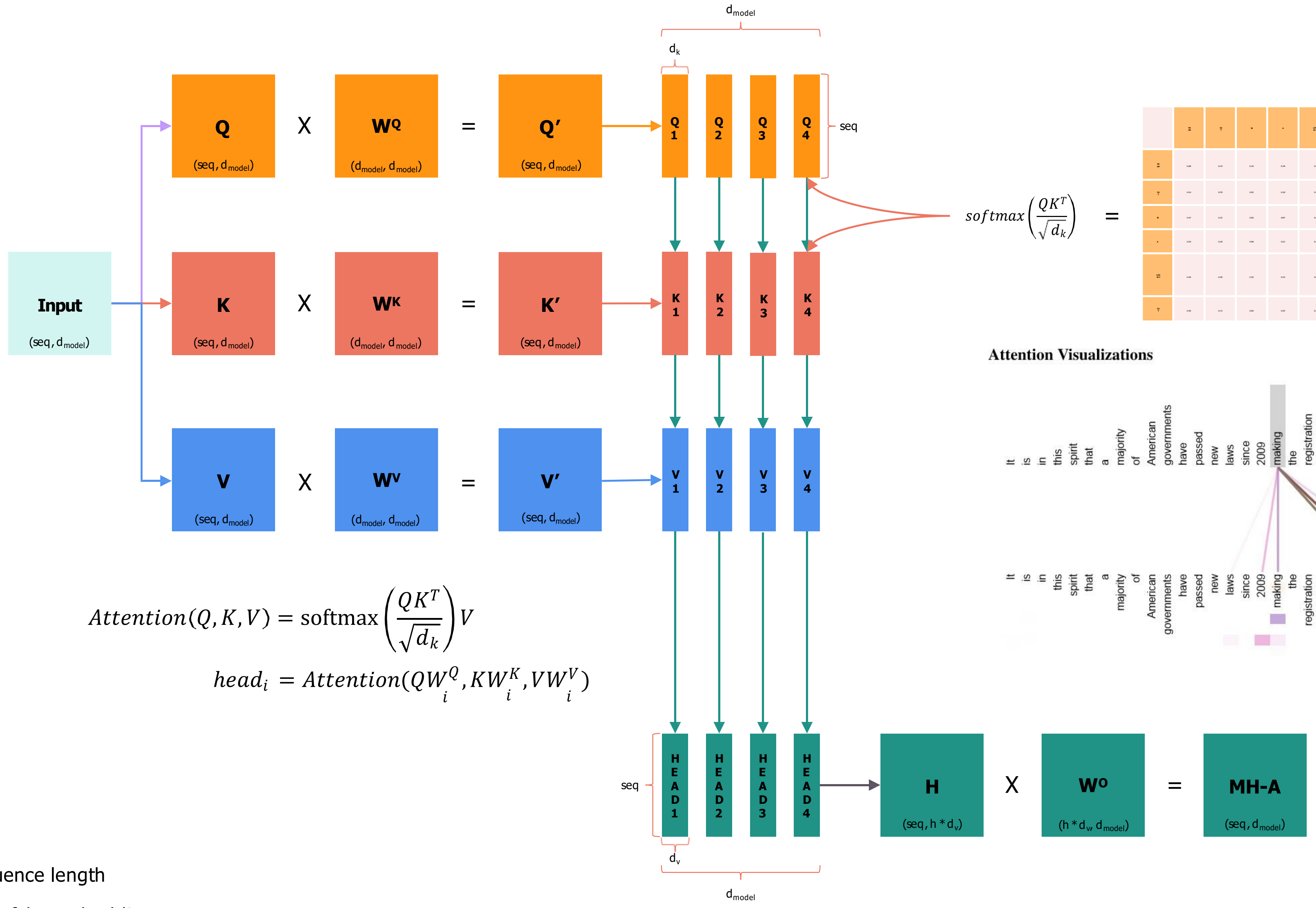


$$Attention(Q,K,V) = \mathrm{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

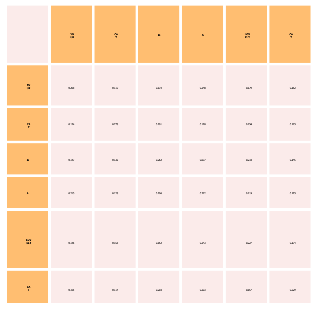
$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$$

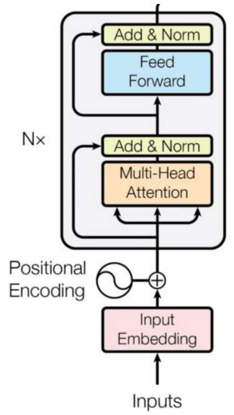
$$MultiHead(Q,K,V) = Concat(head_1 \ \dots head_h)W^O$$

seq	= sequence length
d_{model}	= size of the embedding vector
h	= number of heads
$d_k = d_v$	= d_{model} / h



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

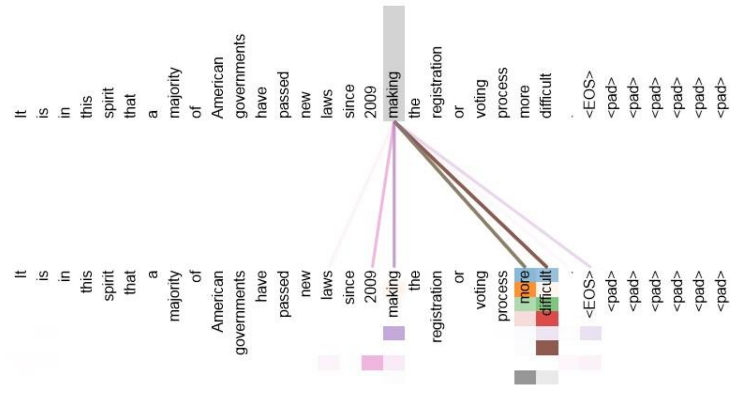




$$Attention(Q,K,V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$$

Attention Visualizations



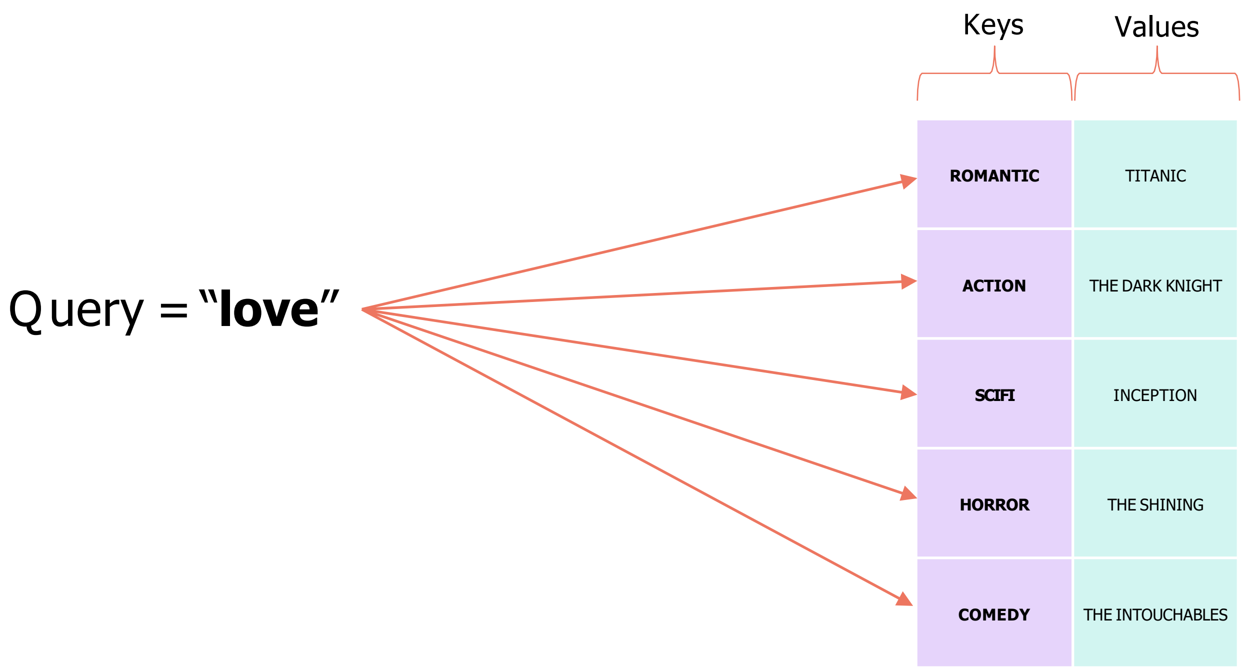
$$MultiHead(Q,K,V) = Concat(head_1 \ \dots head_h)W^O$$

seq	= sequence length
d_{model}	= size of the embedding vector
h	= number of heads
$d_k = d_v$	= d_{model} / h

The purpose of using multiple heads is to allow the model to focus on different aspects or patterns in the input sequence in parallel.

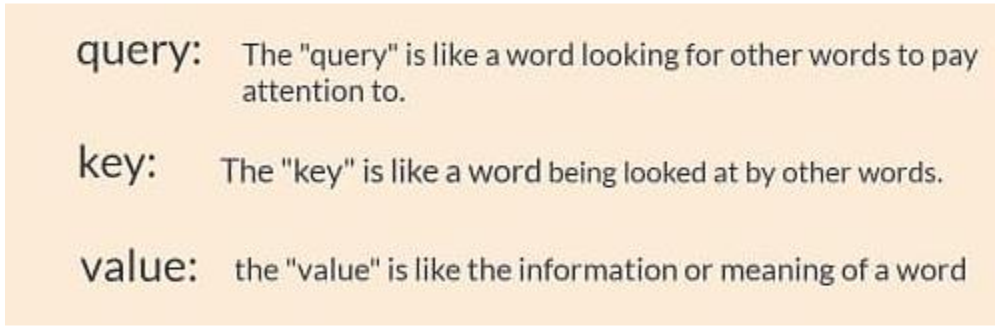
Why query, keys and values?

The Internet says that these terms come from the database terminology or the Python-like dictionaries.



Keys	Values
ROMANTIC	TITANIC
ACTION	THE DARK KNIGHT
SCIFI	INCEPTION
HORROR	THE SHINING
COMEDY	THE INTOUCHABLES

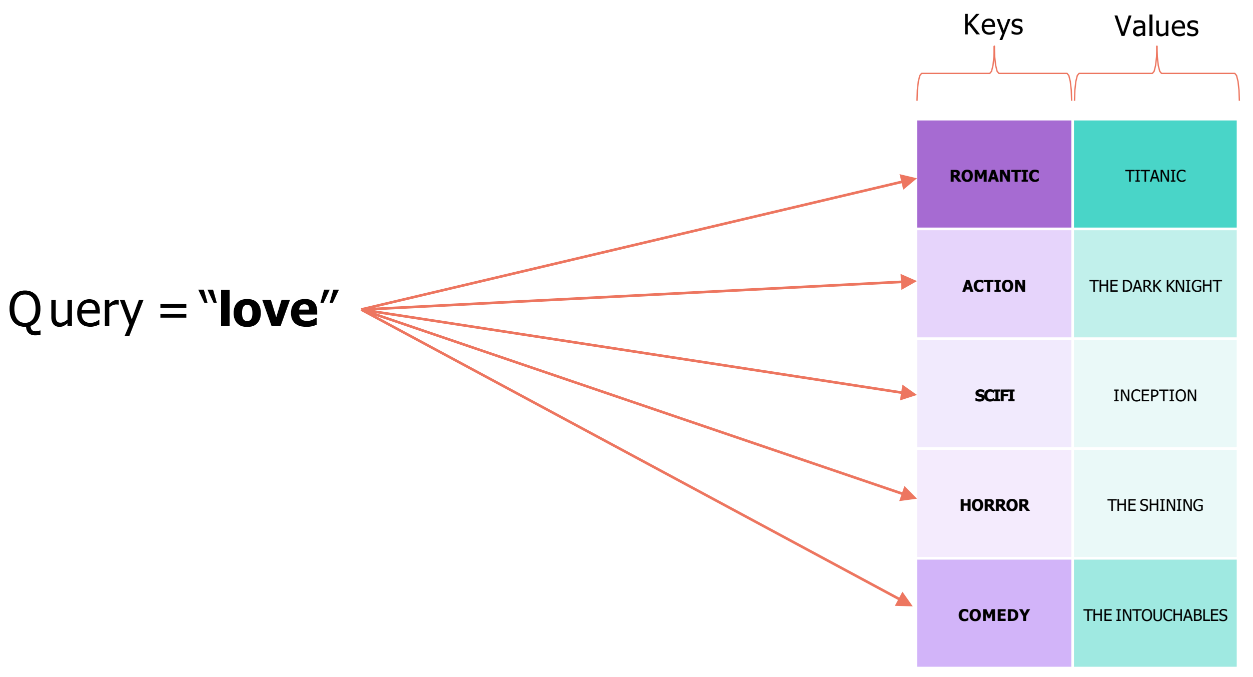
* this could be a Python dictionary or a database table.



query:	The "query" is like a word looking for other words to pay attention to.
key:	The "key" is like a word being looked at by other words.
value:	the "value" is like the information or meaning of a word

Why query, keys and values?

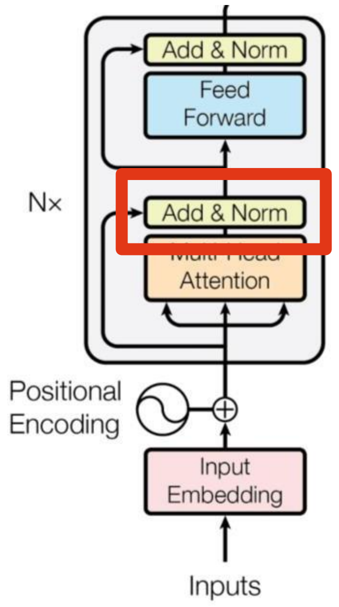
The Internet says that these terms come from the database terminology or the Python-like dictionaries.



Keys	Values
ROMANTIC	TITANIC
ACTION	THE DARK KNIGHT
SCIFI	INCEPTION
HORROR	THE SHINING
COMEDY	THE INTOUCHABLES

* this could be a Python dictionary or a database table.

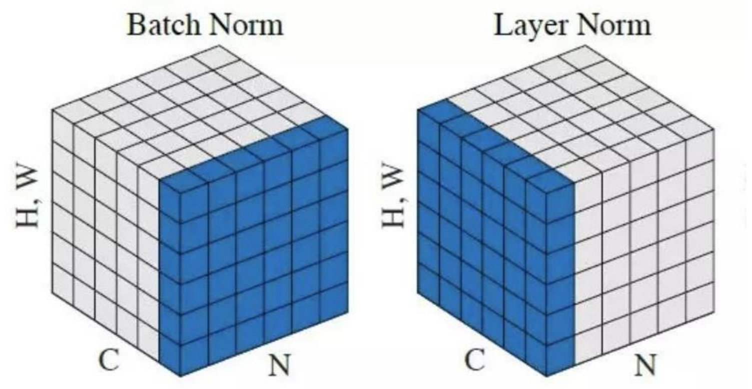
Encoder


Add & Norm
Feed Forward
Nx
Add & Norm
Multi-Head Attention
Positional Encoding
Input Embedding
Inputs

What is layer normalization?

Batch of 3 items

ITEM 1	ITEM 2	ITEM 3
50.147	1242.223	9.370
3314.825	688.123	4606.674
…	…	…
…	…	…
8463.361	434.944	944.705
8.021	149.442	21189.444
μ_1	μ_2	μ_3
σ_1^2	σ_2^2	σ_3^2

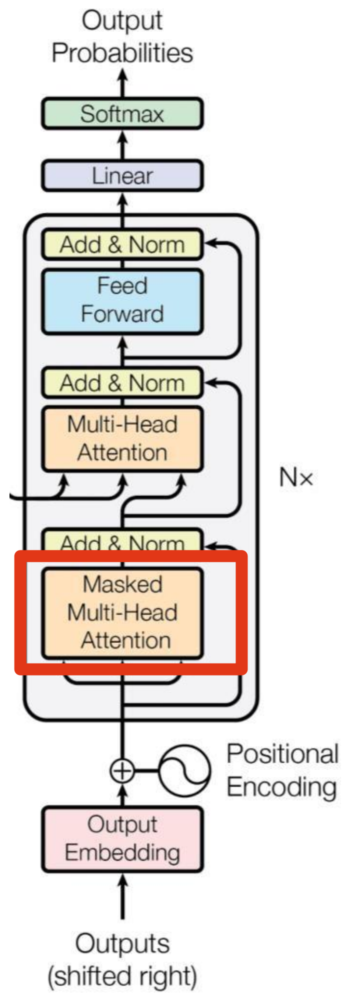


$$\hat{x}_j = \frac{x_j - \mu_j}{\sqrt{\sigma^2 + \epsilon}}$$

We also introduce two parameters, usually called **gamma** (multiplicative) and **beta** (additive) that introduce some fluctuations in the data, because maybe having all values between 0 and 1 may be too restrictive for the network. The network will learn to tune these two parameters to introduce fluctuations when necessary.

Decoder

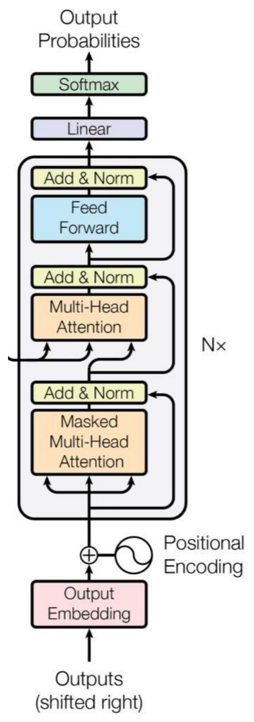
Cross attention: keys, values from encoder Queries from decoder

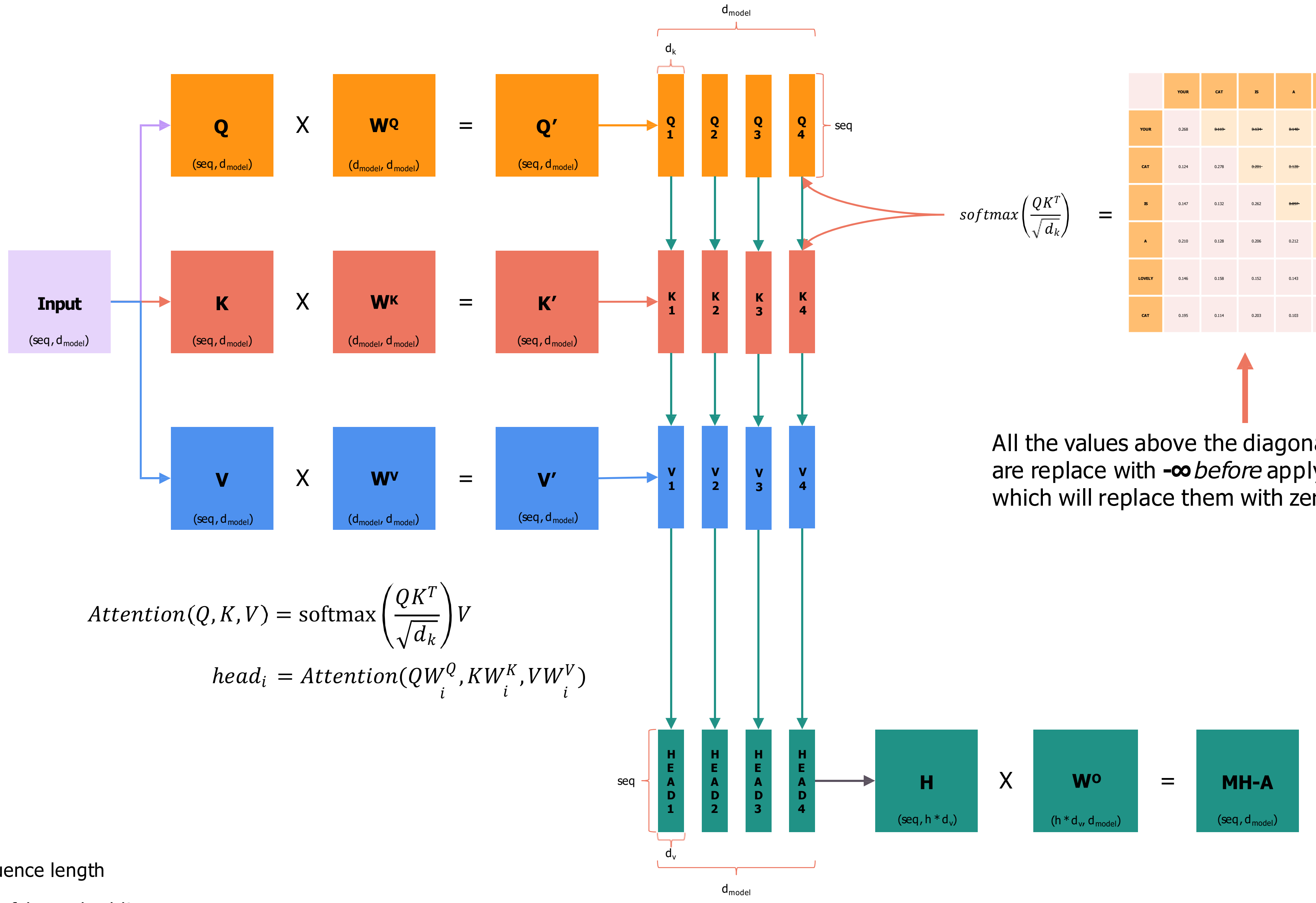


What is Masked Multi-Head Attention?

Our goal is to make the model causal: it means the output at a certain position can only depend on the
words on the previous positions. The model **must not** be able to see future words.

	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





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

	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

All the values above the diagonal are replace with **-∞** *before* applying the softmax, which will replace them with zero.

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

$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$$

$$MultiHead(Q,K,V) = Concat(head_1 \ldots head_h)W^O$$

seq	= sequence length
d_{model}	= size of the embedding vector
h	= number of heads
$d_k = d_v$	$= d_{model} / h$

Inference and training of a Transformer model

Training

I love you very much

$\downarrow$

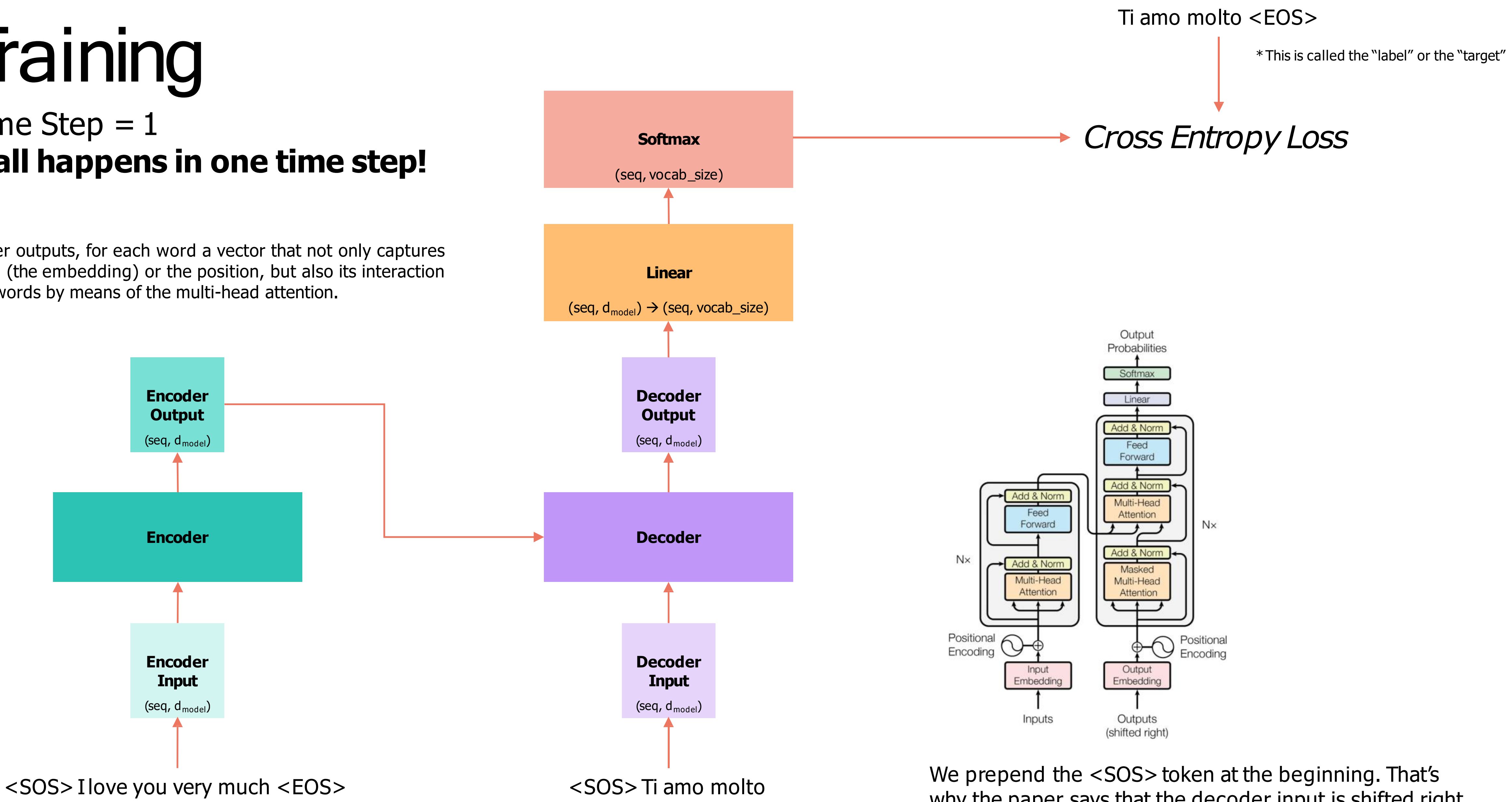
Ti amo molto

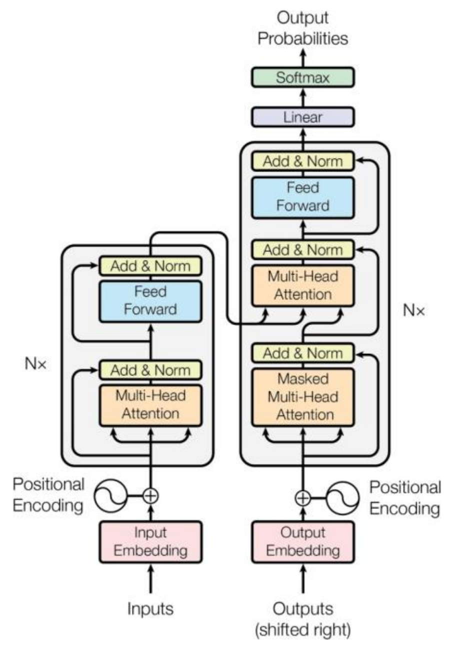
Training

Time Step = 1

It all happens in one time step!

The encoder outputs, for each word a vector that not only captures its meaning (the embedding) or the position, but also its interaction with other words by means of the multi-head attention.





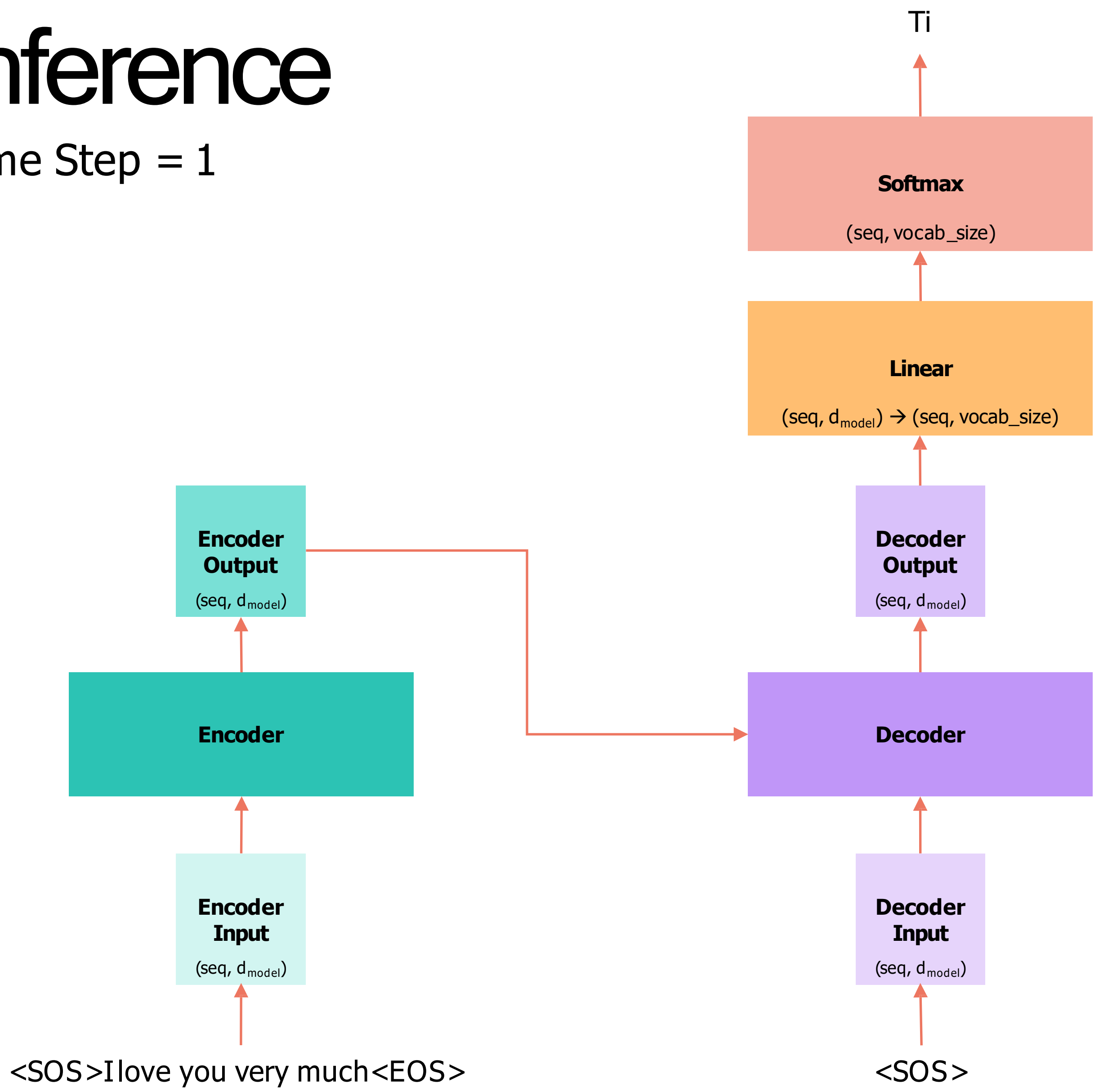
We prepend the <SOS> token at the beginning. That's why the paper says that the decoder input is shifted right.

Inference

$$\begin{tikzcd}
\text{I love you very much} \arrow[d] \\
\text{Ti amo molto}
\end{tikzcd}$$

Inference

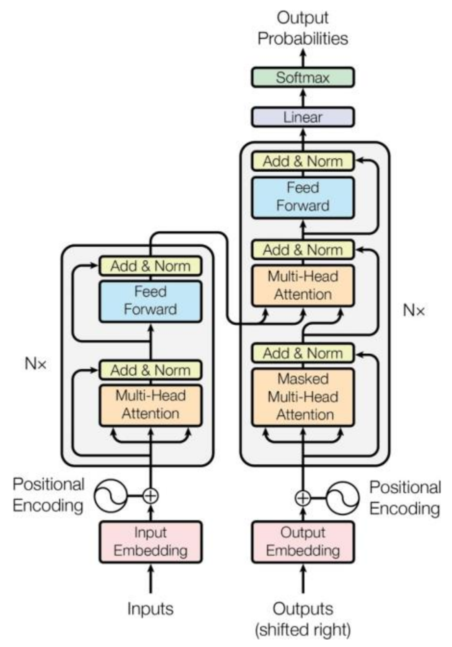
Time Step = 1



- The decoder is initialized with the encoder's output and a special start-of-sequence token.
- In a loop, the decoder generates one token at a time, and the generated token is used as input for the next step.
- The process continues until an end-of-sequence token is generated or a maximum sequence length is reached.

We select a token from the vocabulary corresponding to the position of the token with the maximum value.

The output of the last layer is commonly known as **logits**

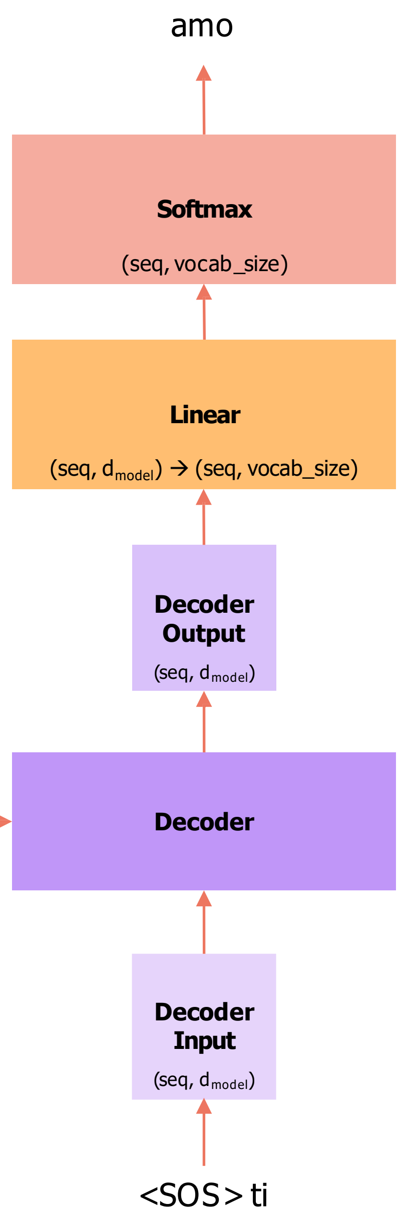


* Both sequences will have same length thanks to padding

Inference

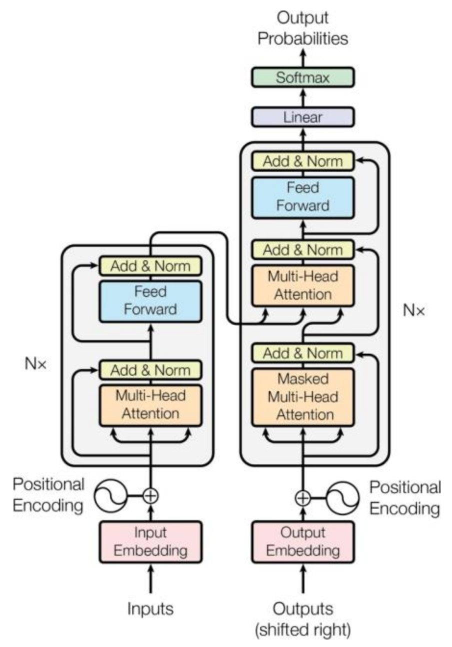
Time Step = 2

Since decoder input now contains **two** tokens, we select the softmax corresponding to the second token.



Use the encoder output from the first time step

<SOS>I love you very much<EOS>

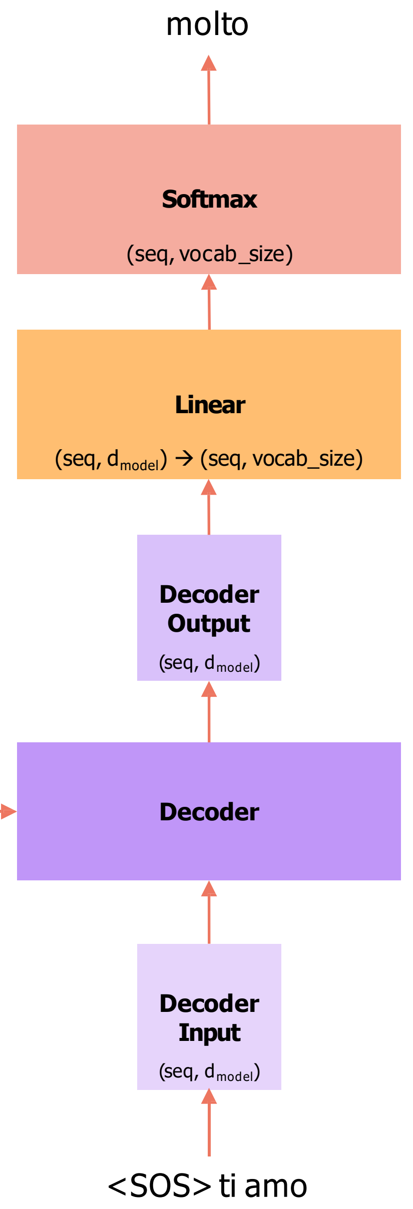


Append the previously output word to the decoder input

Inference

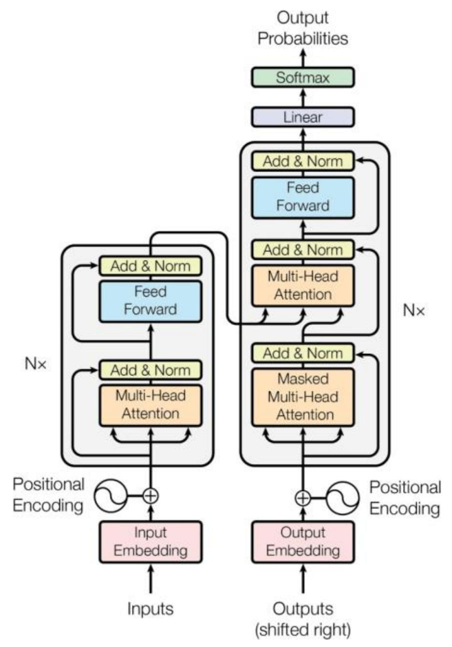
Time Step = 3

Since decoder input now contains **three** tokens, we select the softmax corresponding to the third token.



Use the encoder output from the first time step

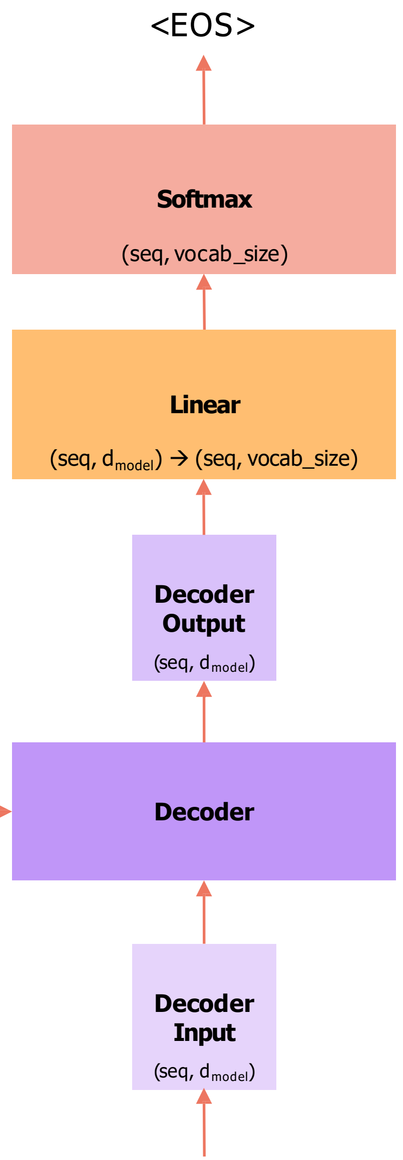
<SOS>I love you very much<EOS>



Append the previously output word to the decoder input

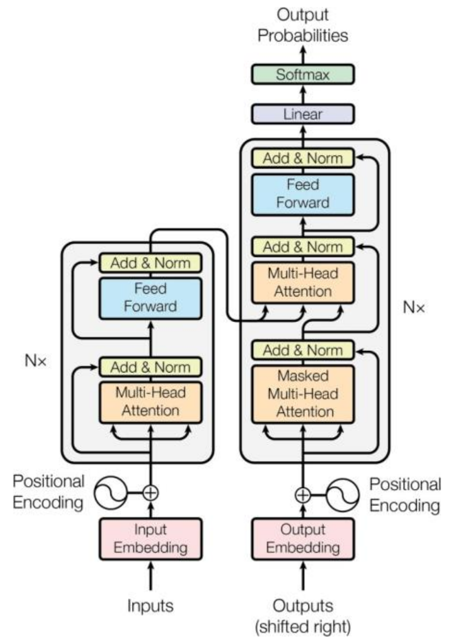
Inference

Time Step = 4



Since decoder input now contains **four** tokens, we select the softmax corresponding to the fourth token.

Use the encoder output from the first time step



<SOS>I love you very much<EOS>

<SOS> ti amo molto

Append the previously output word to the decoder input

Inference strategy

- We selected, at every step, the word with the maximum softmax value. This strategy is called **greedy** and usually does not perform very well.
- A better strategy is to select at each step the top *B* words and evaluate all the possible next words for each of them and at each step, keeping the top *B* most probable sequences. This is the **Beam Search** strategy and generally performs better.