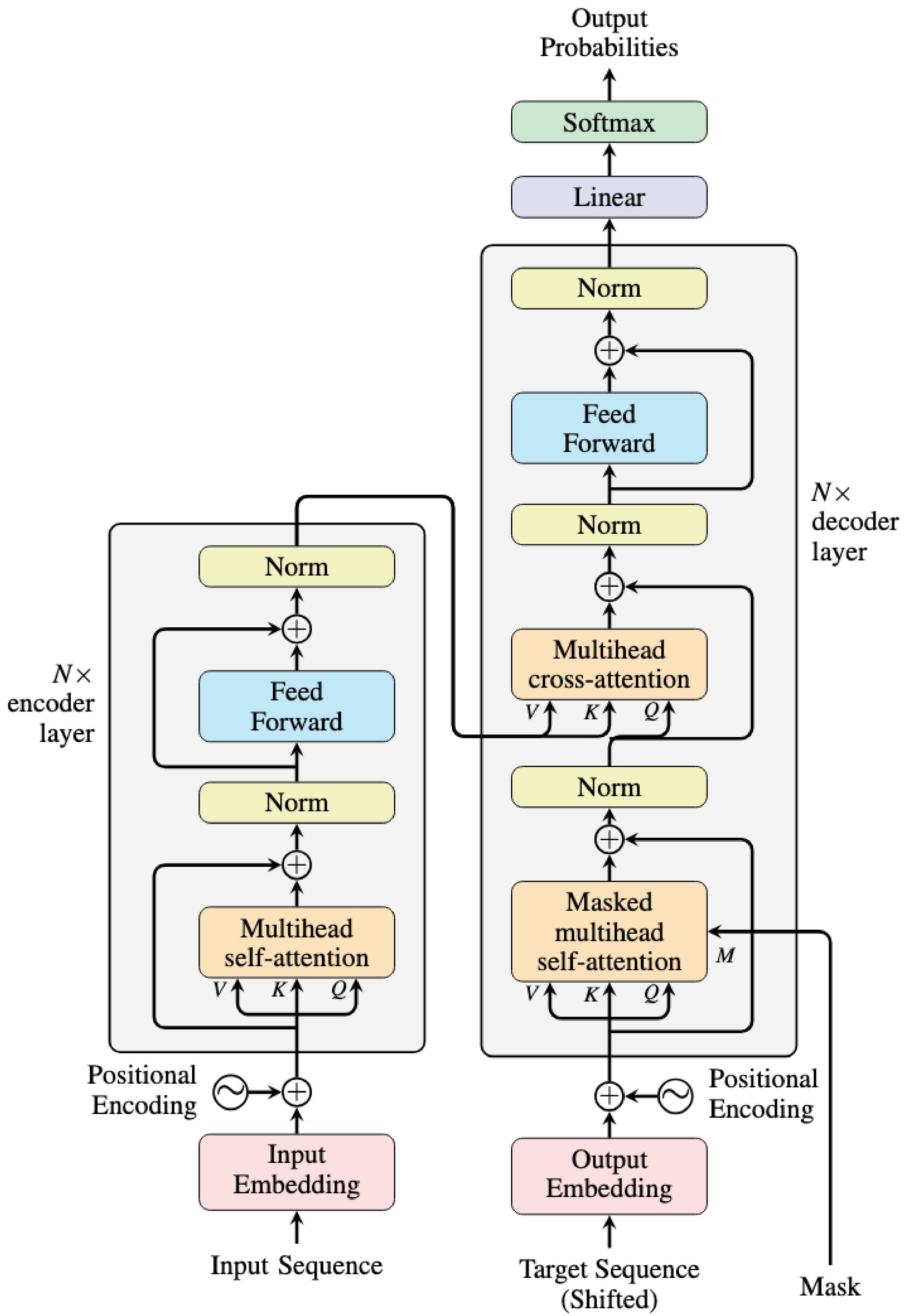

Output Probabilities
Softmax
Linear
Norm
Feed Forward
Norm
Multihead cross-attention
V
K
Q
Norm
Masked multihead self-attention
M
V
K
Q
N× decoder layer
Norm
Feed Forward
Norm
Multihead self-attention
V
K
Q
N× encoder layer
Positional Encoding
Input Embedding
Input Sequence
Positional Encoding
Output Embedding
Target Sequence (Shifted)
Mask

What is an input embedding?

Original sentence (tokens)	YOUR	CAT	IS	A	LOVELY	CAT
Input IDs (position in the vocabulary)	105	6587	5478	3578	65	6587
Embedding (vector of size 512)	952.207	171.411	621.059	776.652	7726.083	171.411
	5450.800	3279.350	1306.051	1515.055	9512.385	3514.080
	1853.448	9128.816	0.545	50.942	9228.798	9128.888
	...	...	...	...	...	...
	165.338	5633.822	7639.885	2337.192	3679.086	3615.998
	2671.531	8090.473	8090.405	5090.879	755.047	757.142

What is positional encoding?

- We want each word to carry some information about its position in the sentence.
- 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.
- With sinusoidal positional encoding, we build one fixed table of vectors for positions 0...max sequence length, and every token simply uses the vector of its position, so all sentences share the same position table independent of vocabulary size.

What is positional encoding?

What is positional encoding?

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

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

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

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 $\mathbf{d_{model}} = \mathbf{d_k} = 512$.

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	0.266	0.119	0.134	0.148	0.179	0.152	1
CAT	0.124	0.278	0.201	0.128	0.145	0.115	1
IS	0.147	0.132	0.392	0.281	0.218	0.145	1
A	0.210	0.106	0.206	0.193	0.119	0.125	1
LOVELY	0.146	0.146	0.142	0.143	0.227	0.219	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 $d_{model} = d_v$.

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	0.266	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	0.193	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

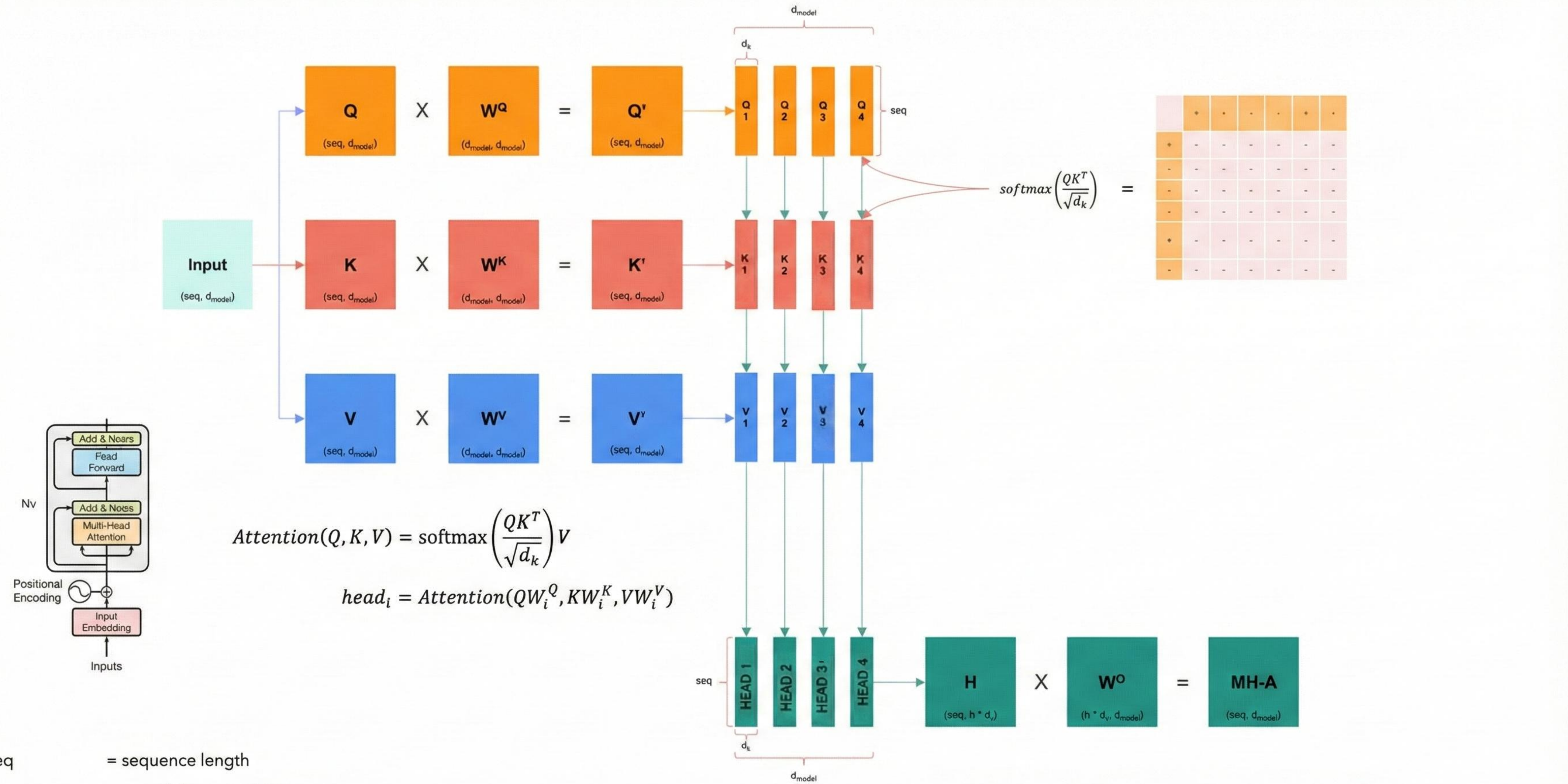
(6, 512)

=

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.



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

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

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

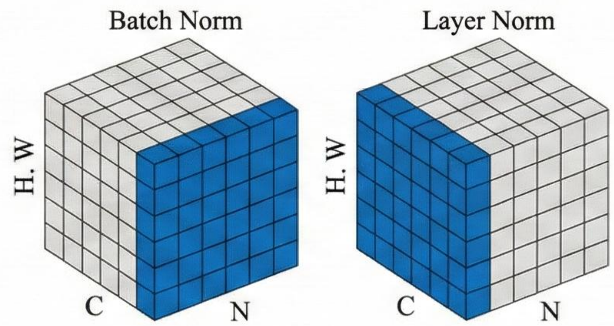
$$MultiHead(Q,K,V) = Concat(head_1 \dots head_h)W^O$$

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

What is layer normalization?

Batch of 3 items

ITEM 1	ITEM 2	ITEM 3
58.147	1242.229	9.310
3014.823	688.123	4664.873
...	...	...
...	...	...
9403.361	423.099	944.705
8.021	102.442	21199.444
μ_1	μ_2	μ_3
σ_1^2	σ_2^2	σ_3^2



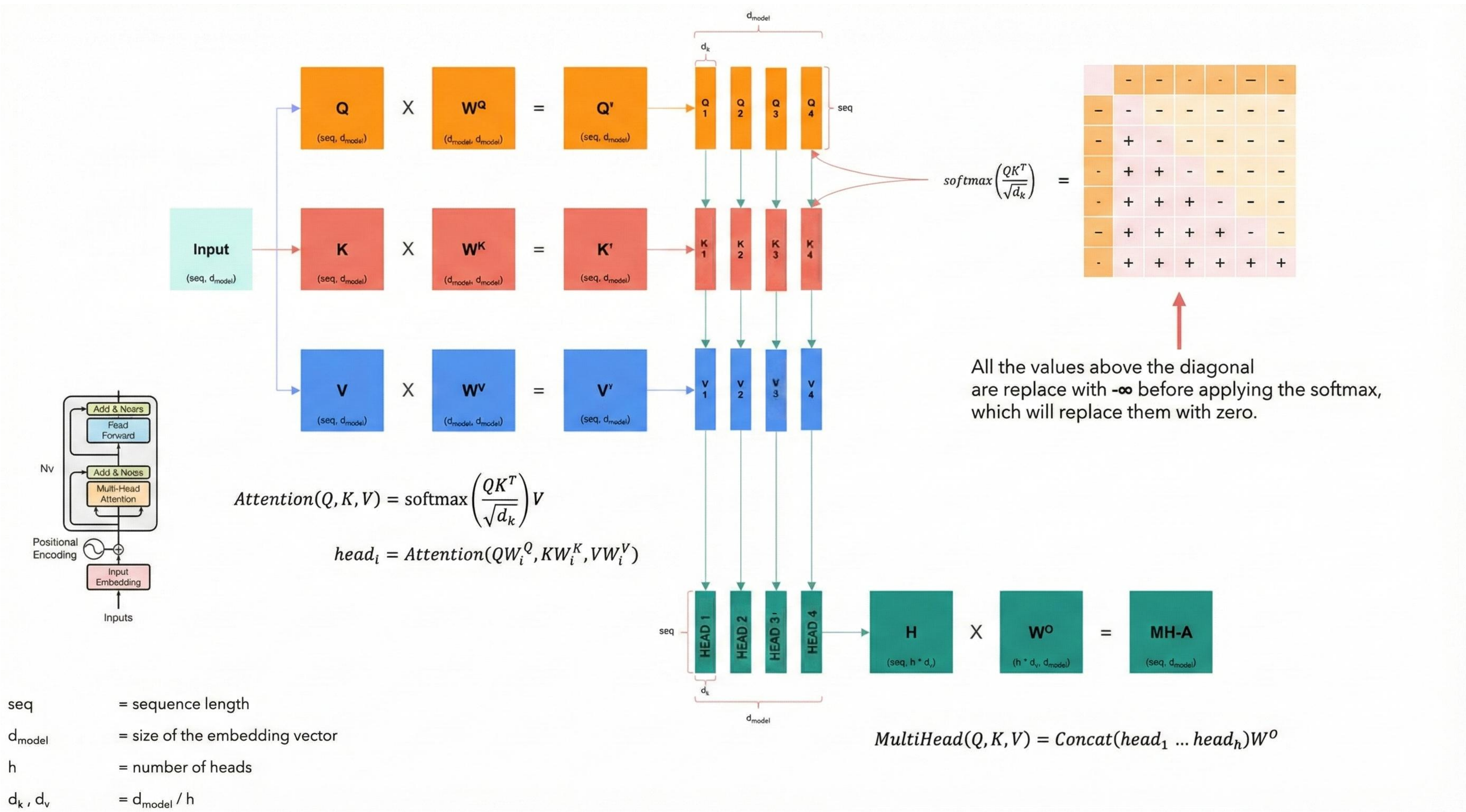
$$\hat{x}_j = \frac{x_j - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

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.

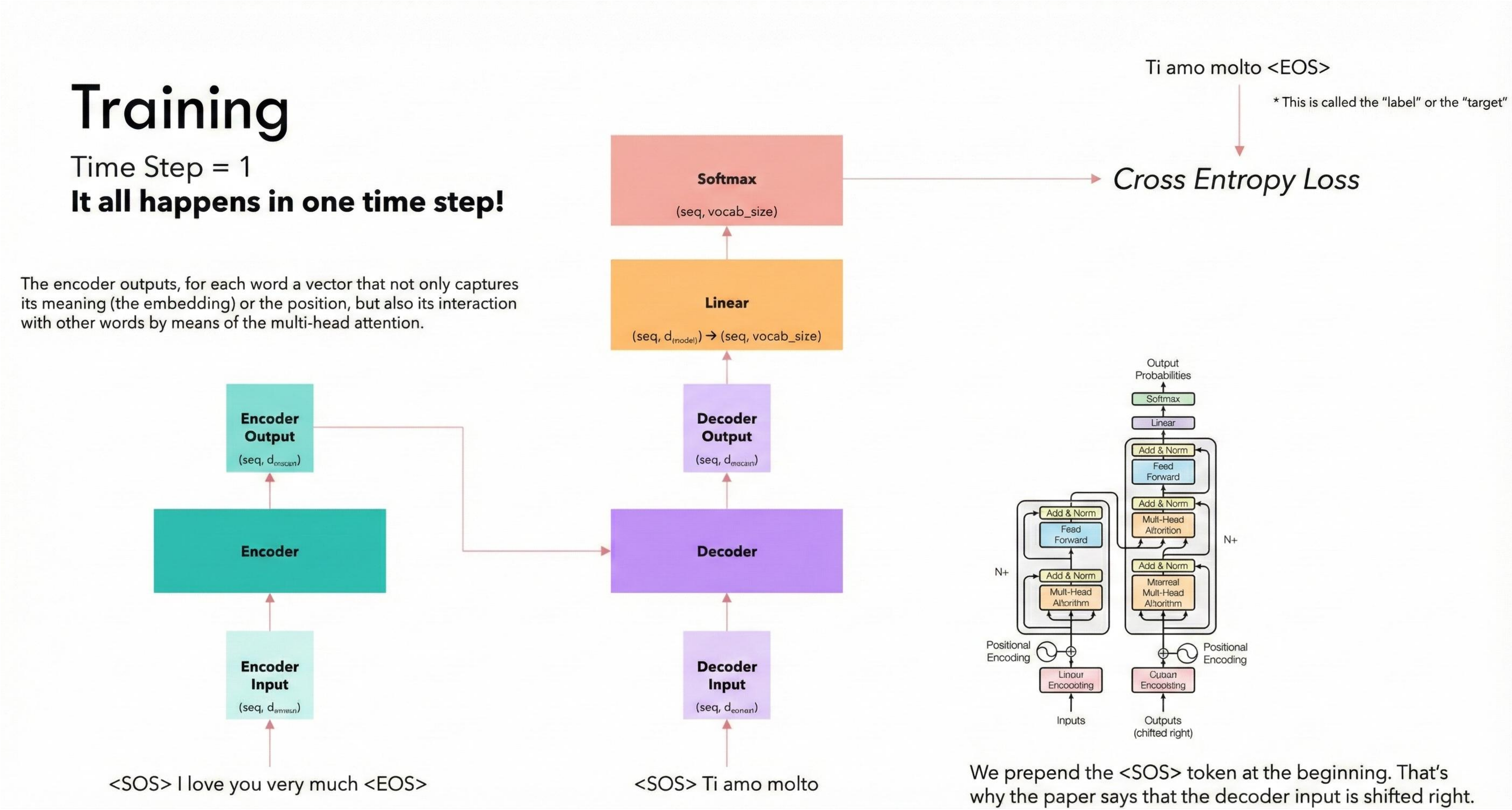
What is Masked Multi-Head Attention?

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	$-\infty$ [illegible]	[illegible]	[illegible]	[illegible]	[illegible]
CAT	0.134	0.279	[illegible]	[illegible]	[illegible]	[illegible]
IS	0.147	0.132	0.132	~~0.282~~	[illegible]	[illegible]
A	0.210	0.128	0.206	0.212	[illegible]	[illegible]
LOVELY	0.146	0.152	0.132	0.140	0.227	[illegible]
CAT	0.195	0.114	0.203	0.103	0.157	0.229

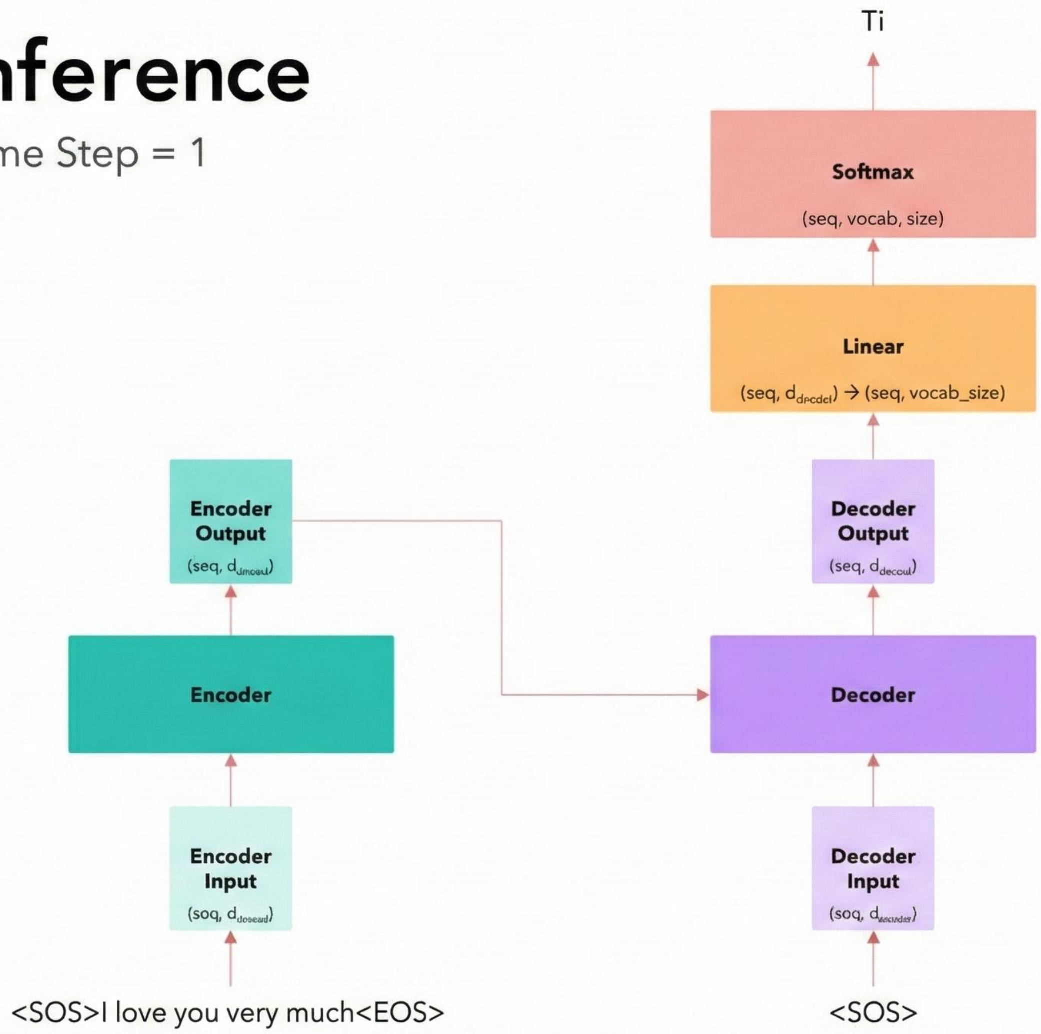


d_{model}
d_k
Q
(seq, d_{model})
X
W^Q
(d_{model}, d_{model})
=
Q'
(seq, d_{model})
Q 1
Q 2
Q 3
Q 4
seq
Input
(seq, d_{model})
K
(seq, d_{model})
X
W^K
(d_{model}, d_{model})
=
K'
(seq, d_{model})
K 1
K 2
K 3
K 4
V
(seq, d_{model})
X
W^V
(d_{model}, d_{model})
=
V'
(seq, d_{model})
V 1
V 2
V 3
V 4
$softmax\left(\frac{QK^T}{\sqrt{d_k}}\right) =$
All the values above the diagonal
are replace with -∞ before applying the softmax,
which will replace them with zero.
Add & Noars
Fead Forward
Nv
Add & Noas
Multi-Head Attention
Positional Encoding
Input Embedding
Inputs
$Attention(Q,K,V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$
$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$
seq
HEAD 1
HEAD 2
HEAD 3
HEAD 4
d_k
d_{model}
H
(seq, h * d_v)
X
W^O
(h * d_v, d_{model})
=
MH-A
(seq, d_{model})
$MultiHead(Q,K,V) = Concat(head_1 \ldots head_h)W^O$
seq = sequence length
d_{model} = size of the embedding vector
h = number of heads
d_k, d_v = d_{model} / h


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.
Ti amo molto <EOS>
* This is called the "label" or the "target"
Cross Entropy Loss
Softmax
(seq, vocab_size)
Linear
(seq, d_{model}) → (seq, vocab_size)
Encoder Output
(seq, d_{onscan})
Decoder Output
(seq, d_{macain})
Encoder
Decoder
Encoder Input
(seq, d_{amusn})
Decoder Input
(seq, d_{eonan})
<SOS> I love you very much <EOS>
<SOS> Ti amo molto
Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Mult-Head Altorition
Add & Norm
Mrerreal Mult-Head Alhorithm
Add & Norm
Feed Forward
Add & Norm
Mult-Head Alhorithm
N+
N+
Positional Encoding
Positional Encoding
Linour Encoooting
Cuban Encoosting
Inputs
Outputs (chifted right)
We prepend the <SOS> token at the beginning. That's why the paper says that the decoder input is shifted right.

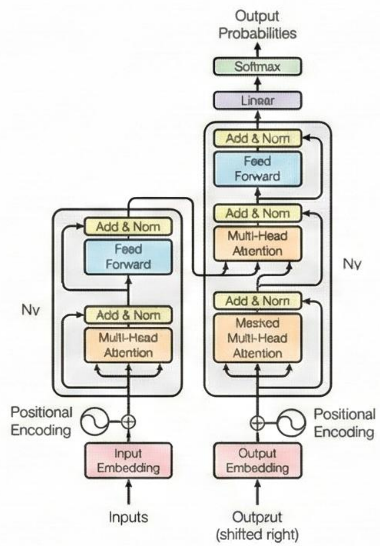
Inference

Time Step = 1



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**

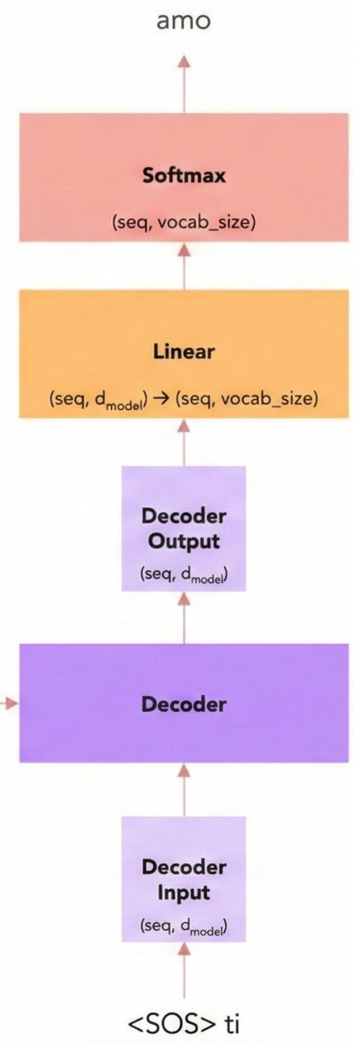


* Both sequences will have same length thanks to padding

Inference

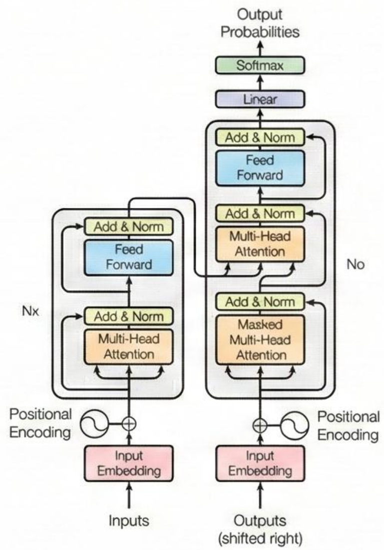
Time Step = 2

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>

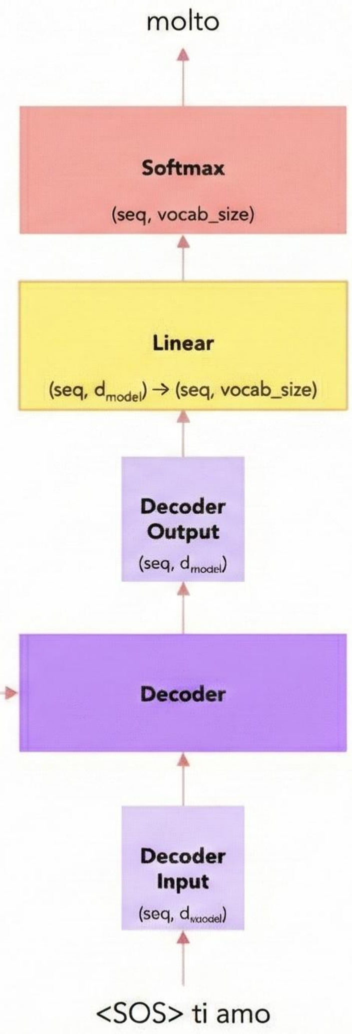


Append the previously output word to the decoder input

Inference

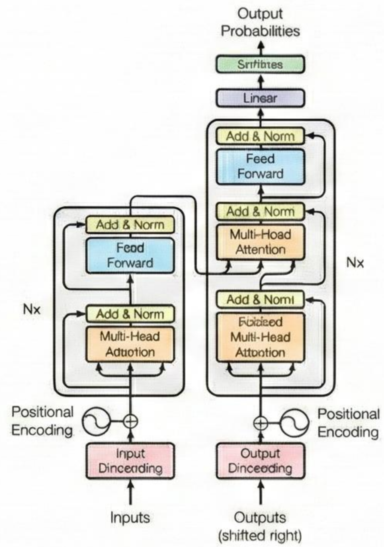
Time Step = 3

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

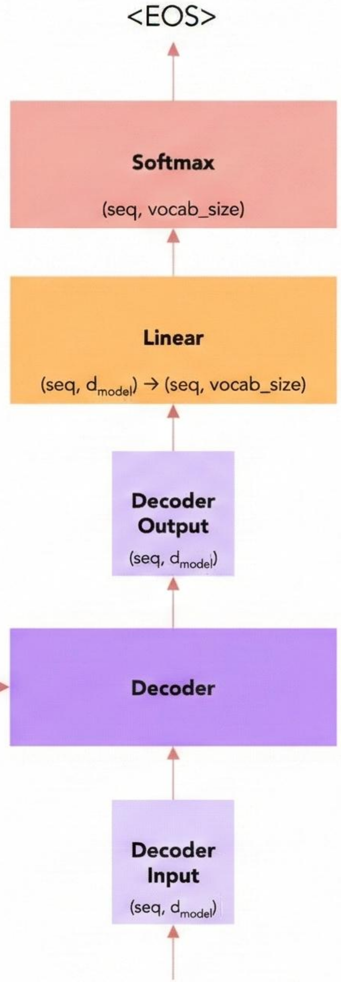
<SOS>I love you very much<EOS>



Append the previously output word to the decoder input

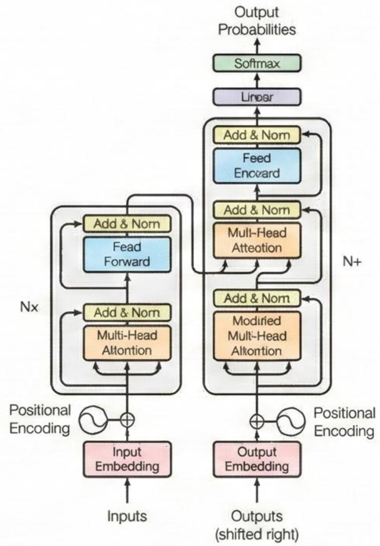
Inference

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.