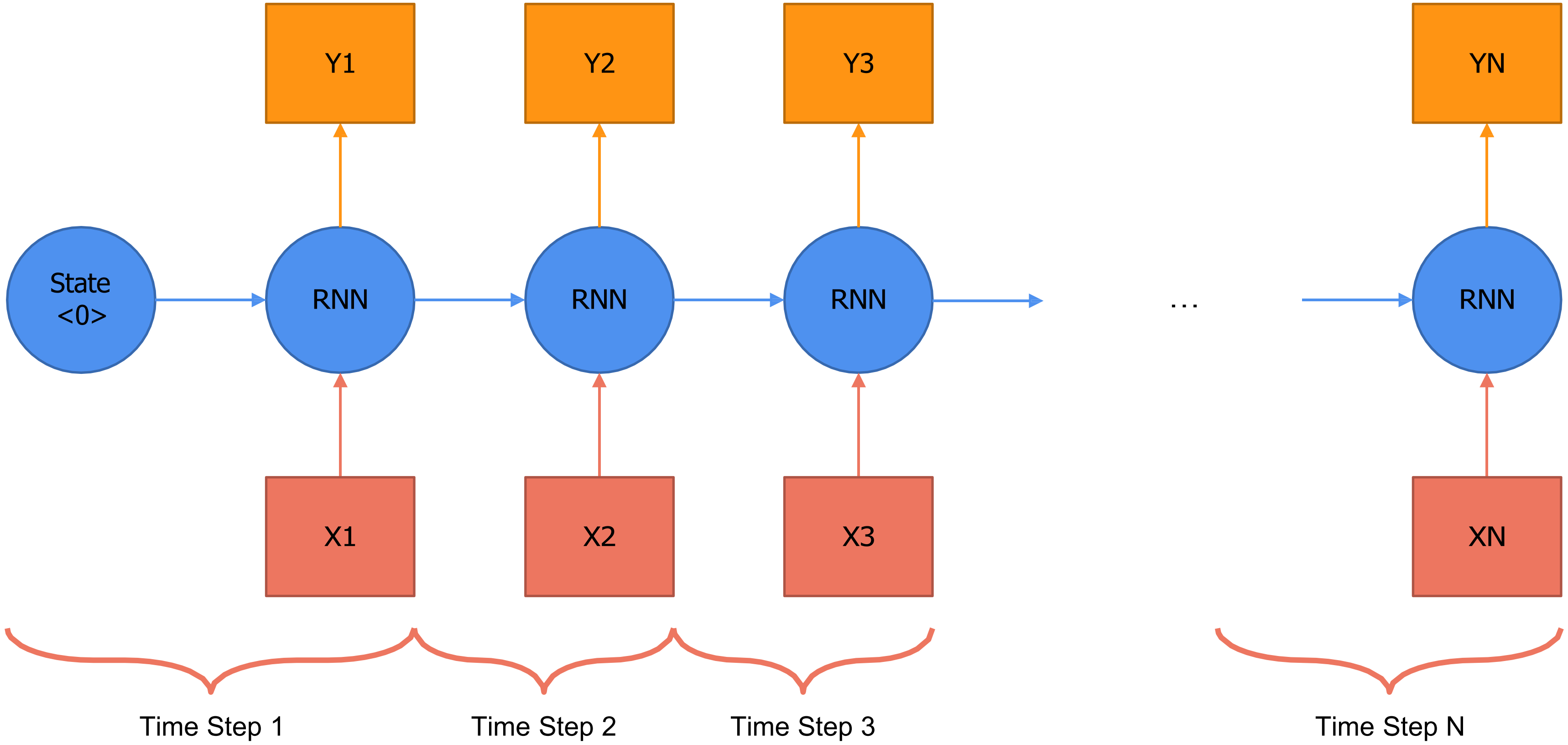
Recurrent Neural Networks (RNN)

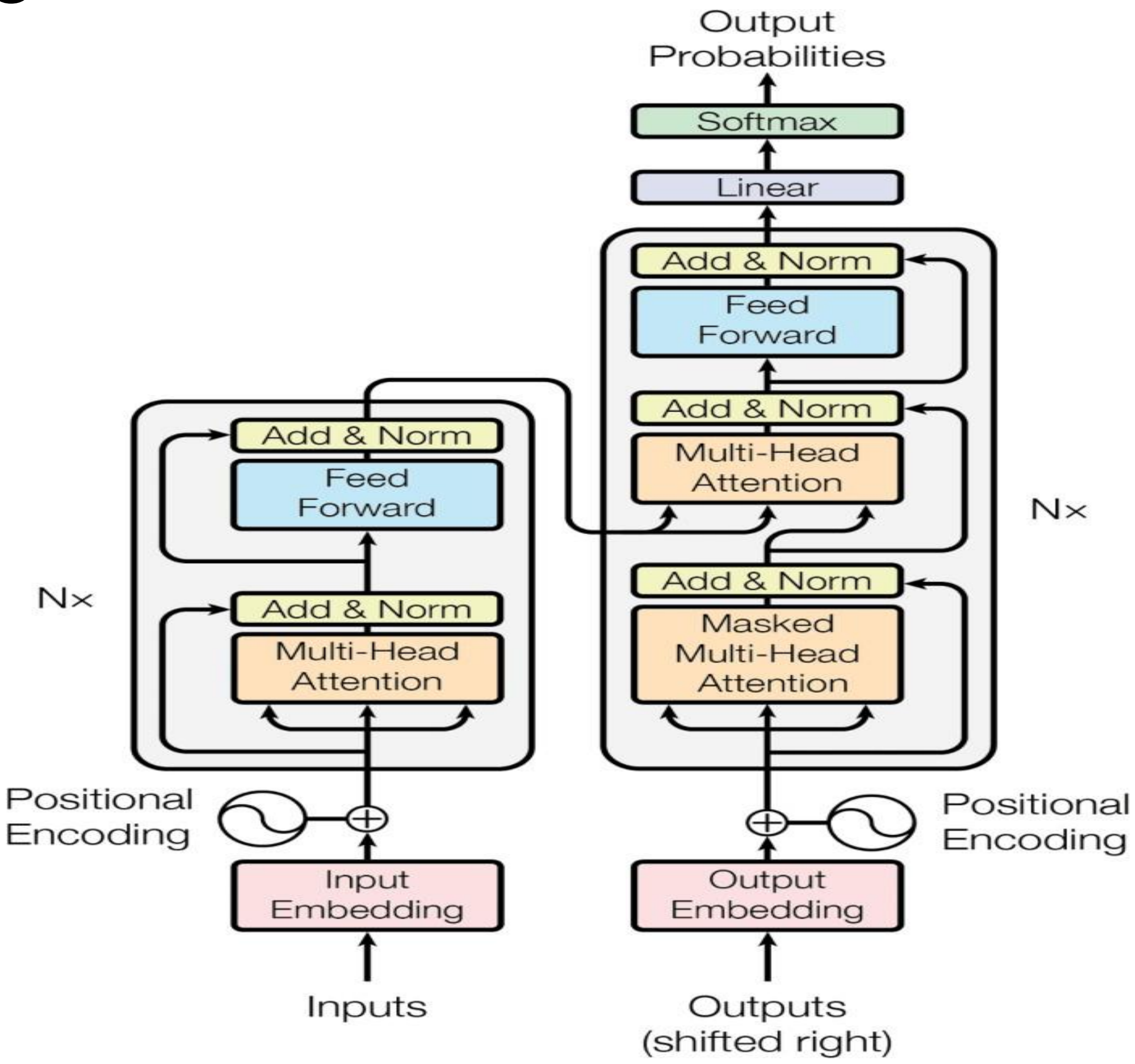


https://github.com/VijayKanagaraj7/Transformers

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

Introducing the Transformer



https://github.com/VijayKanagaraj7/Transformers

Notations

Input matrix (sequence, d_{model})

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

https://github.com/VijayKanagaraj7/Transformers

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.

Positional encoding:

- Positional encoding in Transformers is a technique used to inject information about the order of tokens in a sequence.Transformers have no inherent sense of position due to their parallel processing nature.

- It works by adding a unique vector to each input token's embedding based on its position in the sequence, allowing the model to distinguish between different positions. Typically, this is done using sinusoidal functions that generate a fixed pattern of values for each position and dimension, enabling the model to learn relative positions and generalize to sequences of different lengths during training and inference.

What is positional encoding?

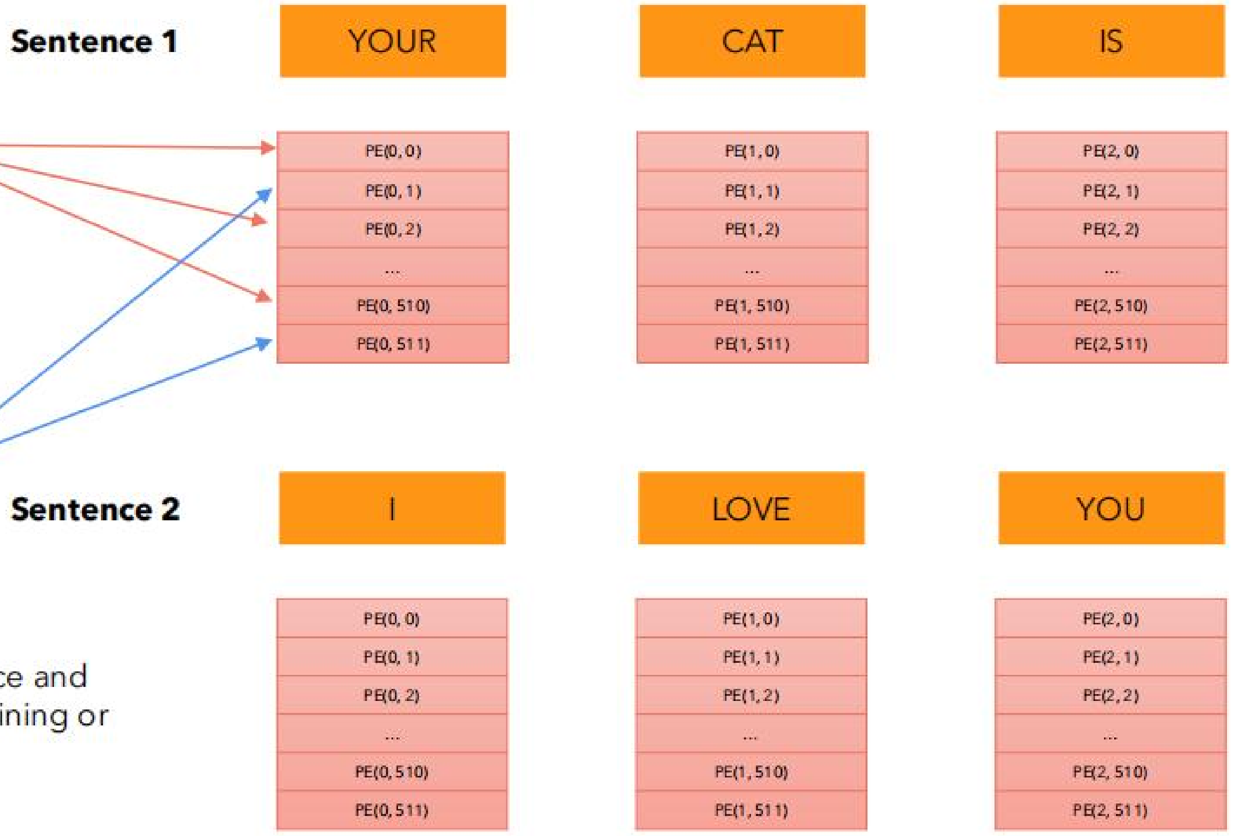
Original sentence	YOUR	CAT	IS	A	LOVELY	CAT
Embedding (vector of size 512)	952.207	171.411	621.659	776.562	6422.693	171.411
	5450.840	3276.350	1304.051	5567.288	6315.080	3276.350
	1853.448	9192.819	0.565	58.942	9358.778	9192.819
	…	…	…	…	…	…
	1.658	3633.421	7679.805	2716.194	2141.081	3633.421
	2671.529	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

https://github.com/VijayKanagaraj7/Transformers

What is positional encoding?

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

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

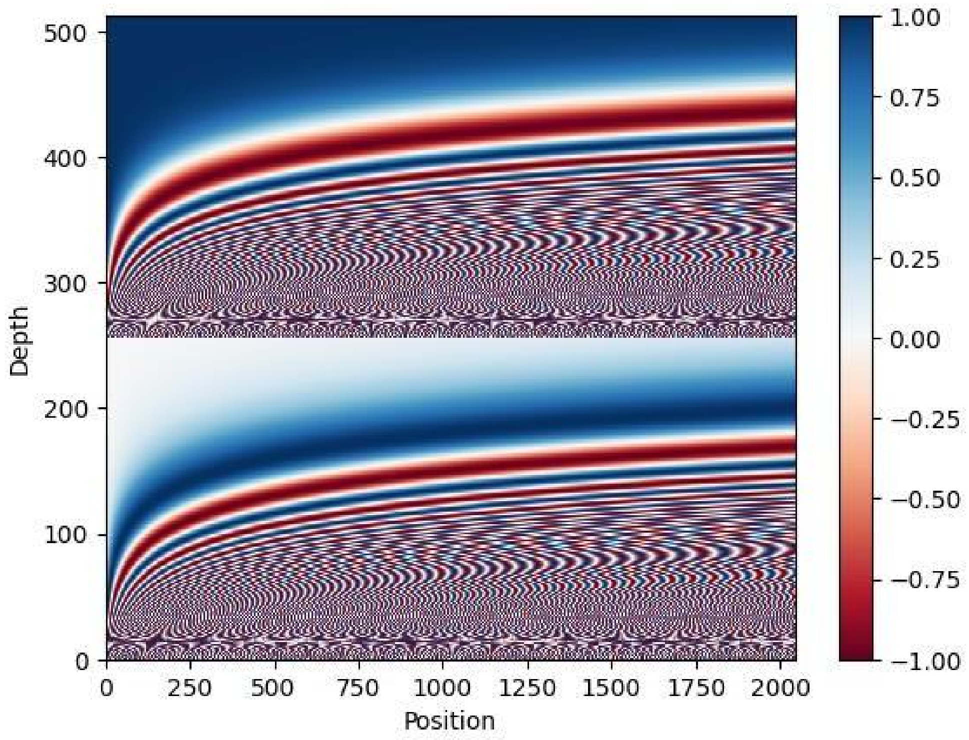


We only need to compute the positional encodings once and then reuse them for every sentence, no matter if it is training or inference.

https://github.com/VijayKanagaraj7/Transformers

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.



https://github.com/VijayKanagaraj7/Transformers

What is Self-Attention?

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

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.

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

	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	
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 consider d only one head, which makes $d_{model} = d_k$.

https://github.com/VijayKanagaraj7/Transformers

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.

https://github.com/VijayKanagaraj7/Transformers

Self-Attention in detail

- Self-Attention is permutation invariant.
- 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

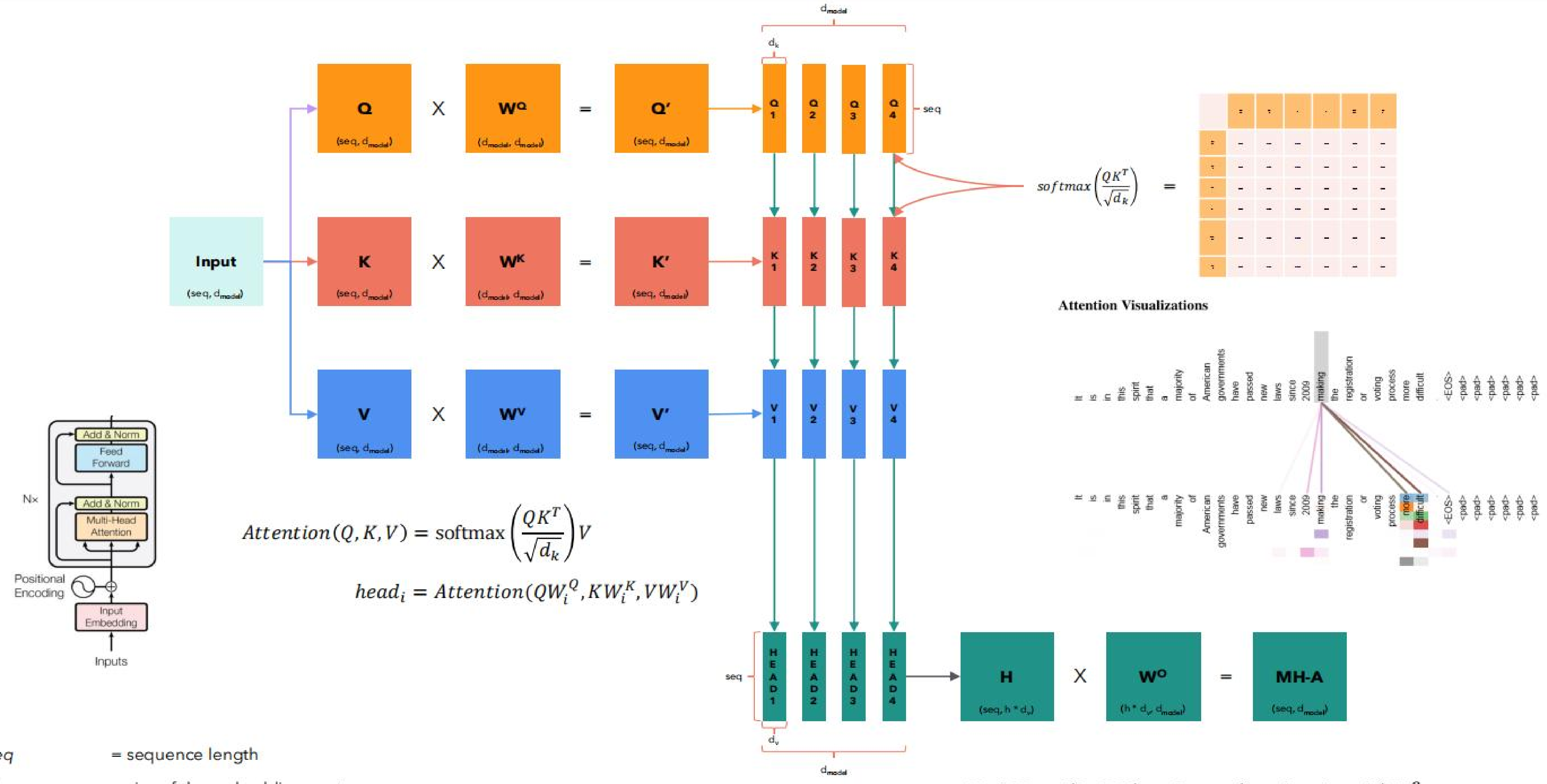
https://github.com/VijayKanagaraj7/Transformers

Multi-head Attention

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

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

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



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

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

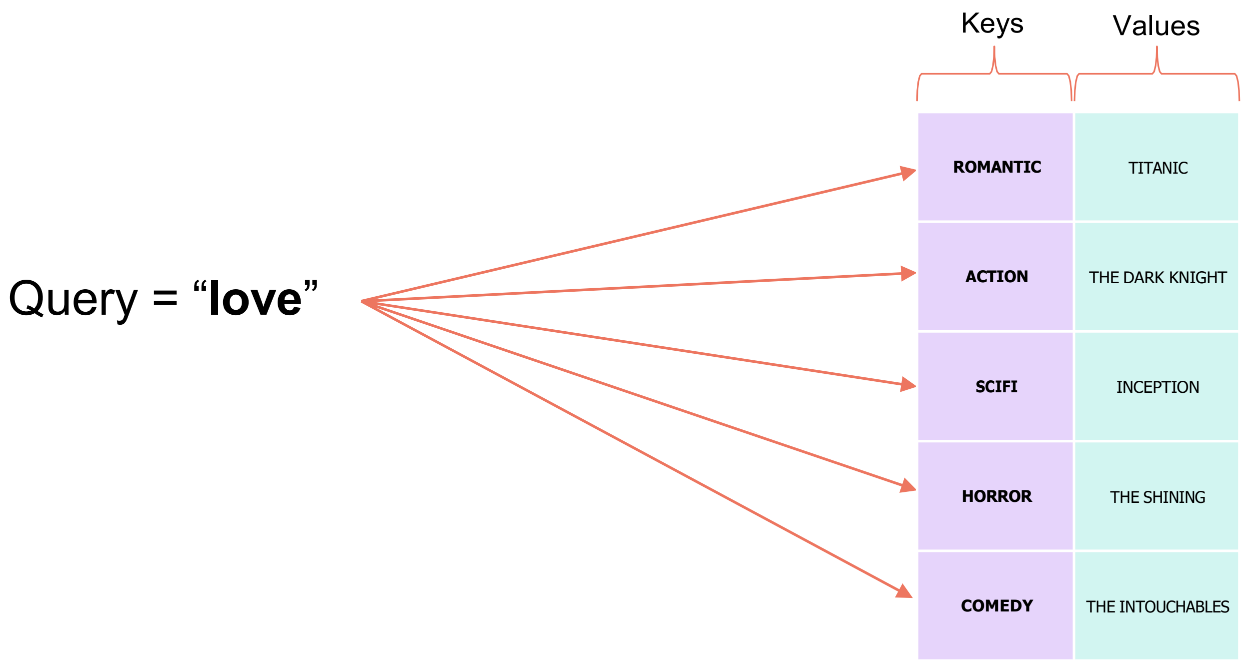
$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$$

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

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

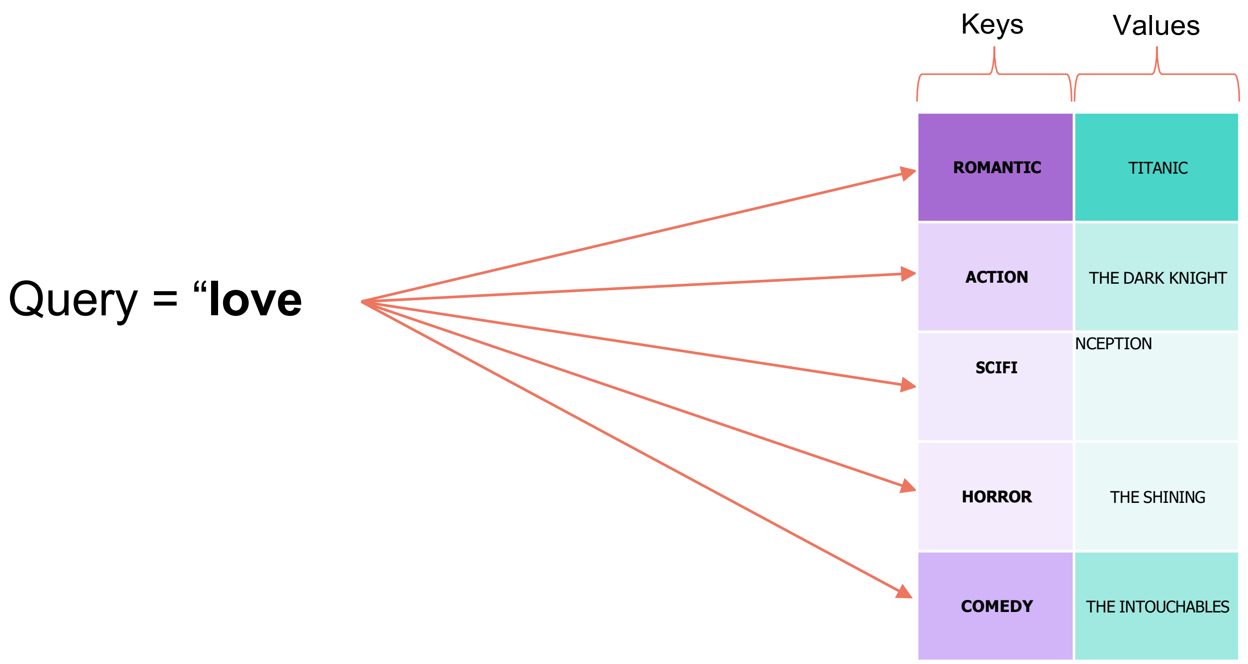
Why query, keys and values?

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



Why query, keys and values?

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

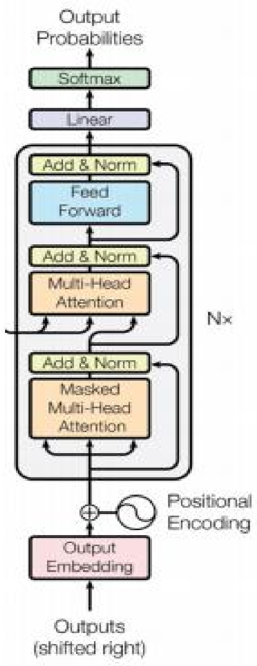


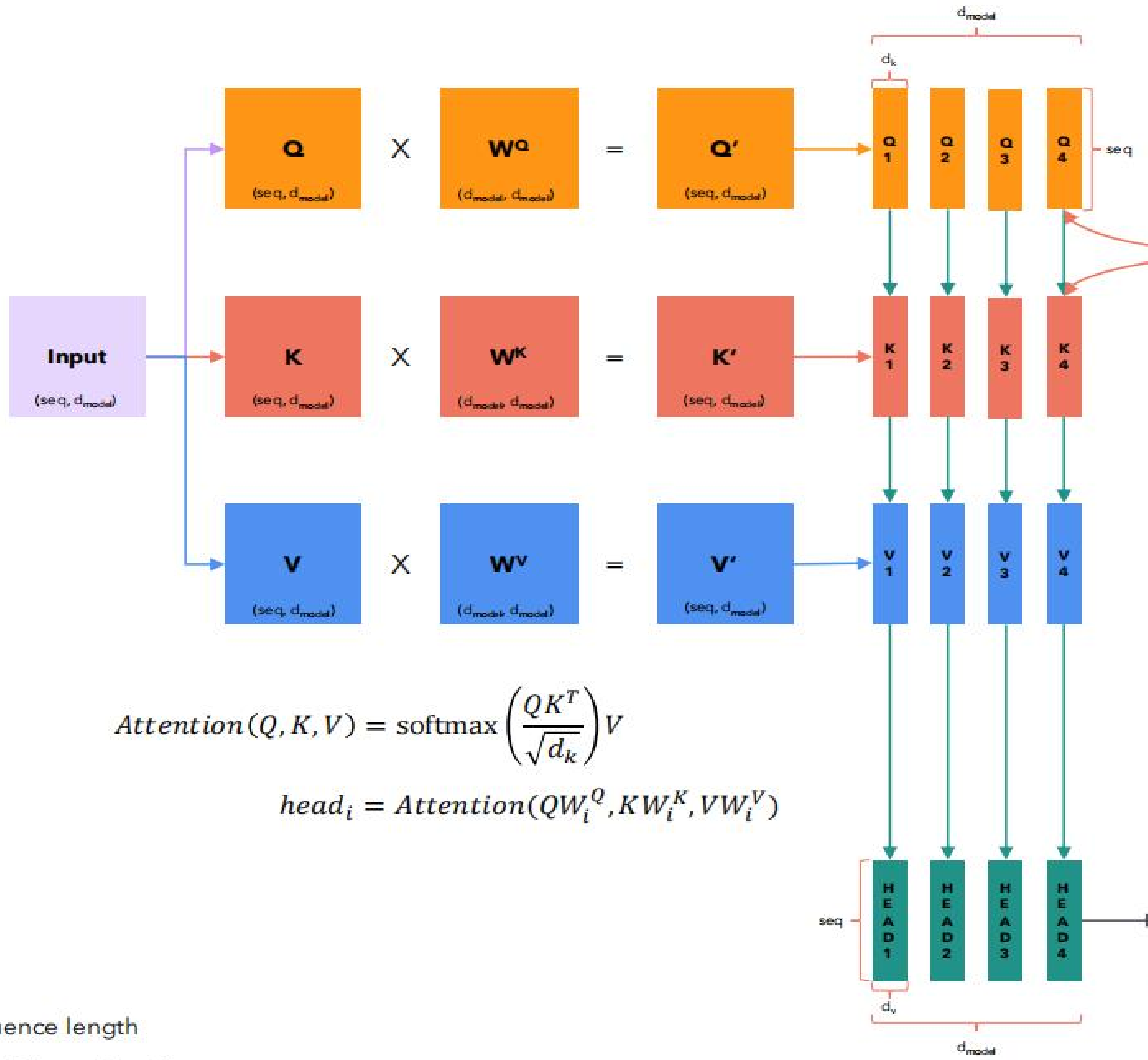
What is Masked Multi-Head Attention?

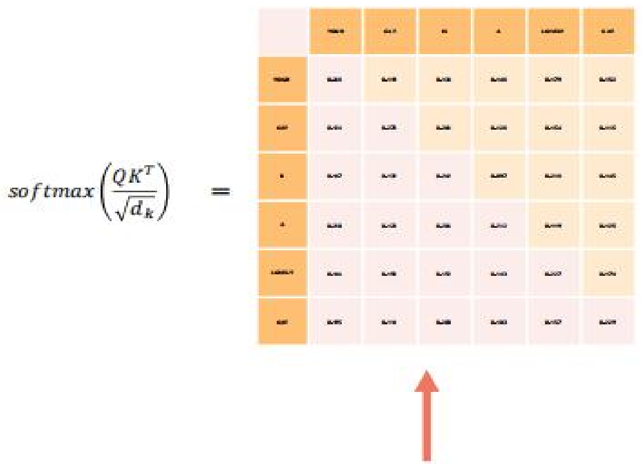
Our goal is to make the model causal: it means the output at a certain position can only depend on the words on the previous positions. The model **must not** be able to see future words.

	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

https://github.com/VijayKanagaraj7/Transformers







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

Training

Time Step = 1

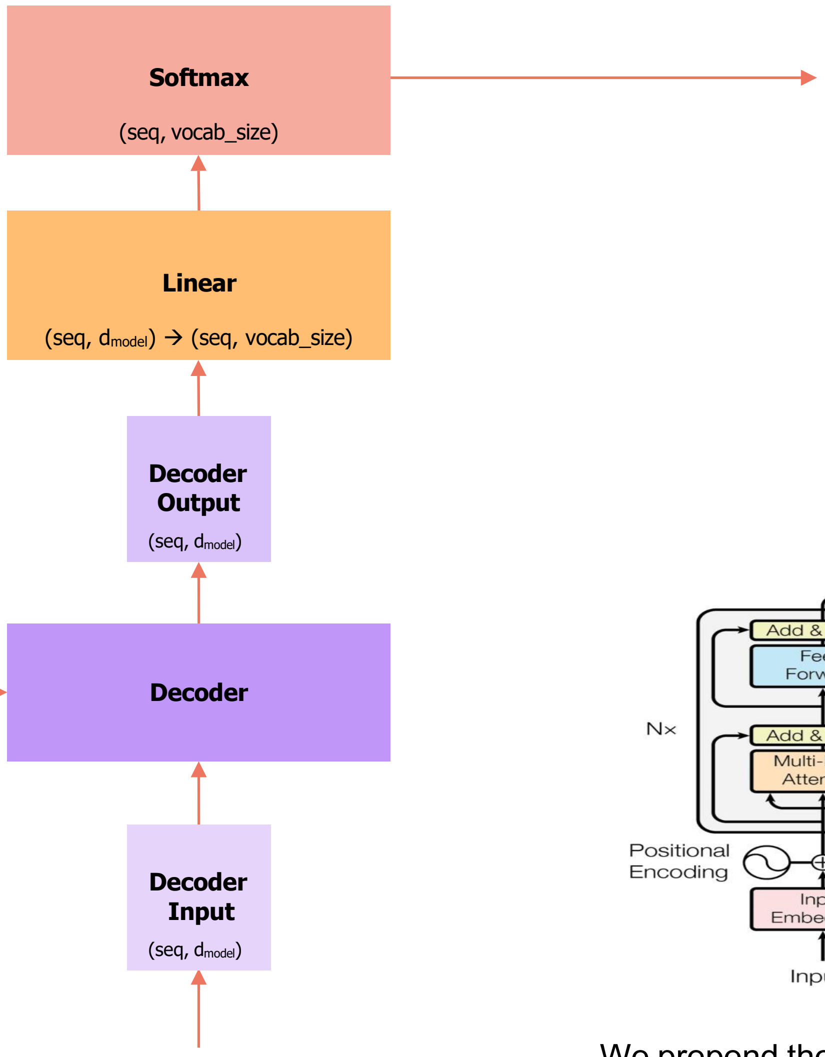
It all happens in one time step!

ti amo molto il cibo <EOS>

*** This is called the “label” or the “target”**

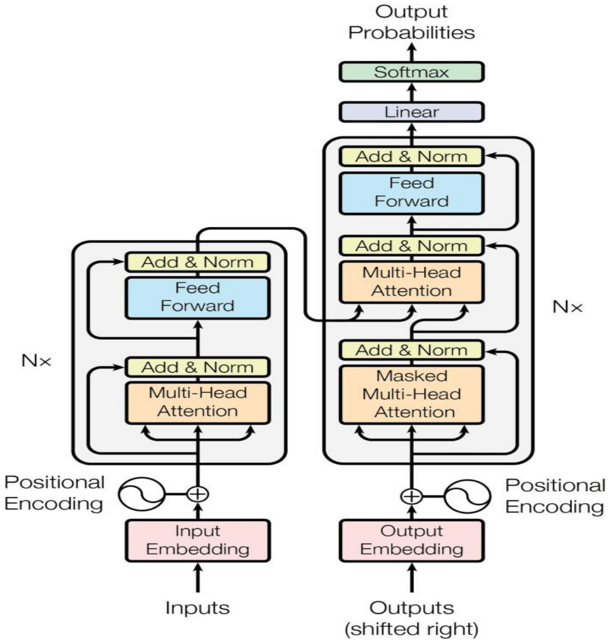
Cross Entropy Loss

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.



<SOS> **I love food very much** <EOS>

<SOS> **ti amo molto il cibo**



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

https://github.com/VijayKanagaraj7/Transformers

Inference

I love food very much

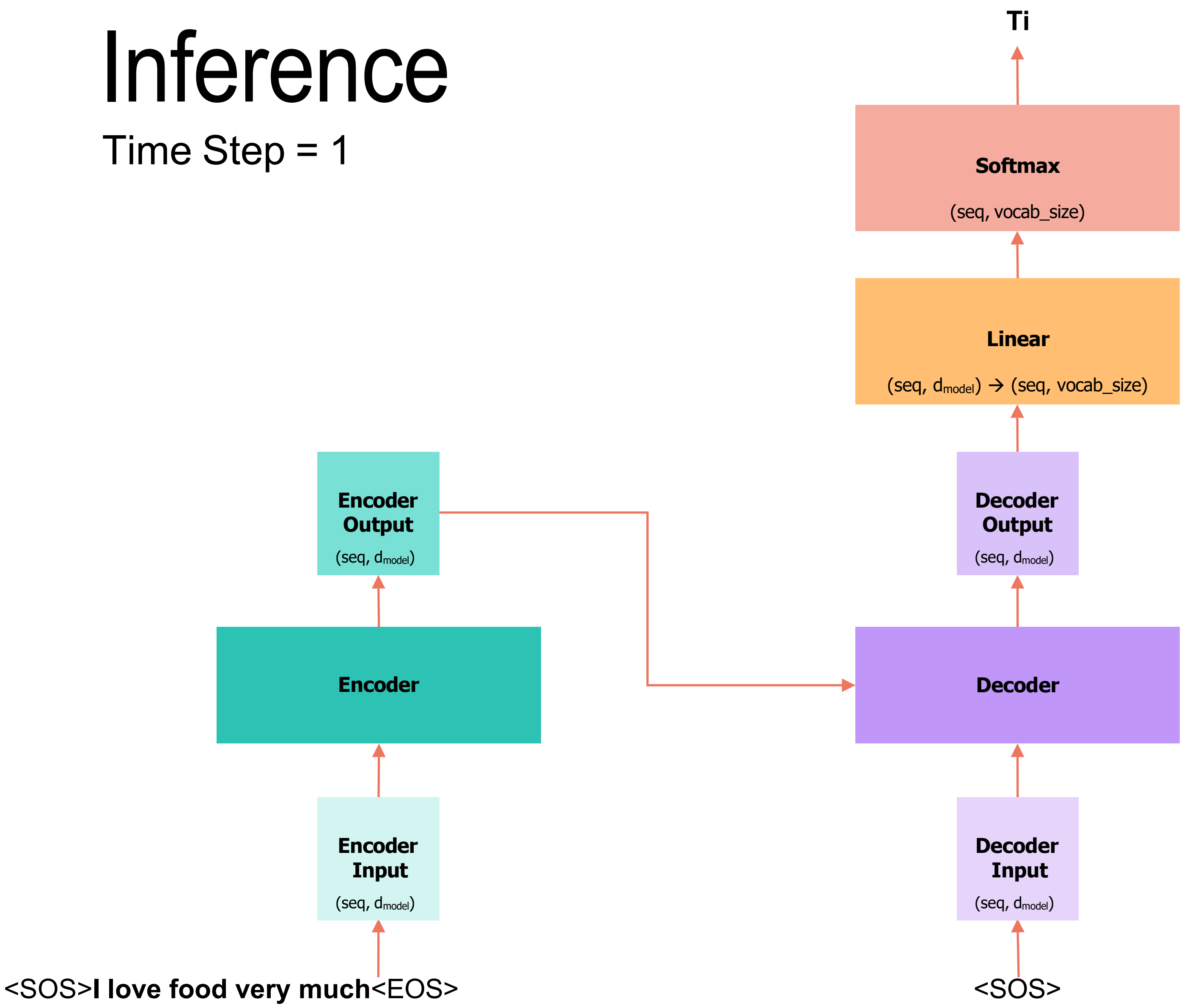
ti amo molto il cibo

<

https://github.com/VijayKanagaraj7/Transformers

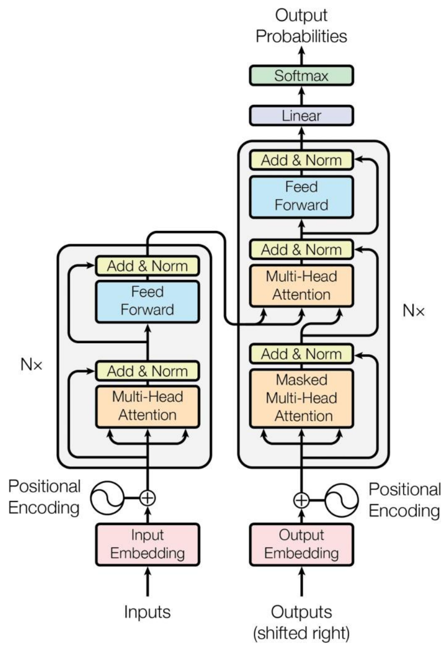
Inference

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**

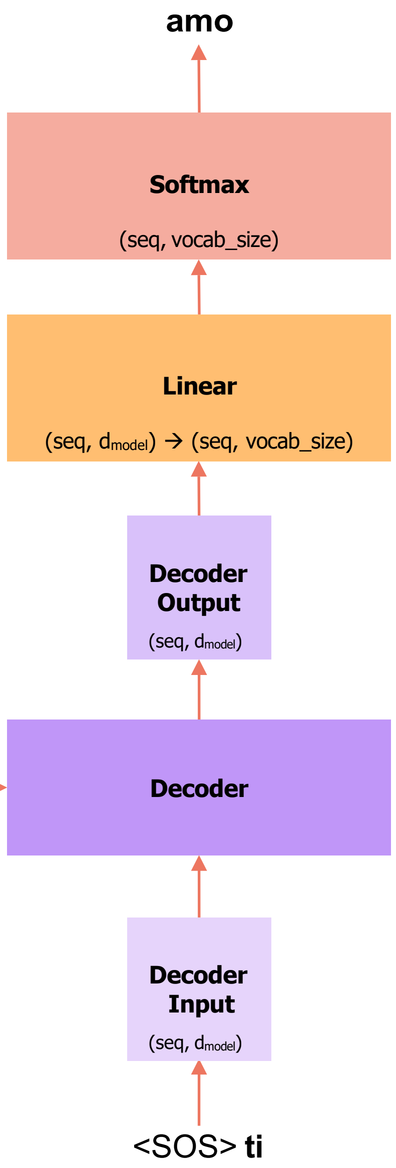


* Both sequences will have same length thanks to padding

https://github.com/VijayKanagaraj7/Transformers

Inference

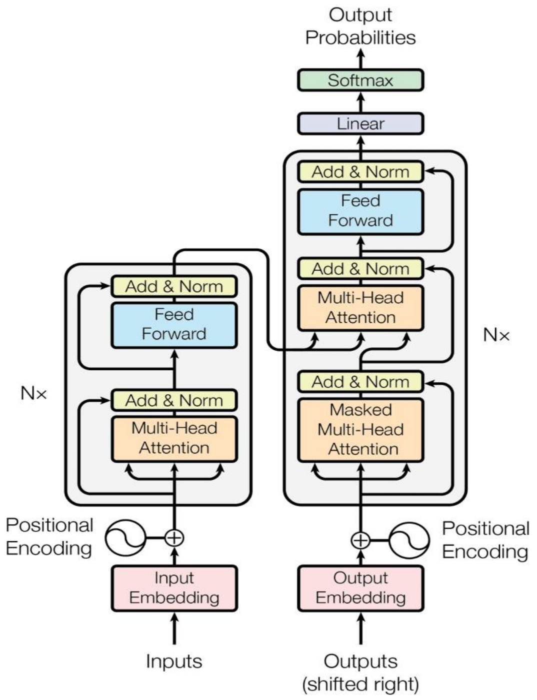
Time Step = 2



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 food very much**<EOS>

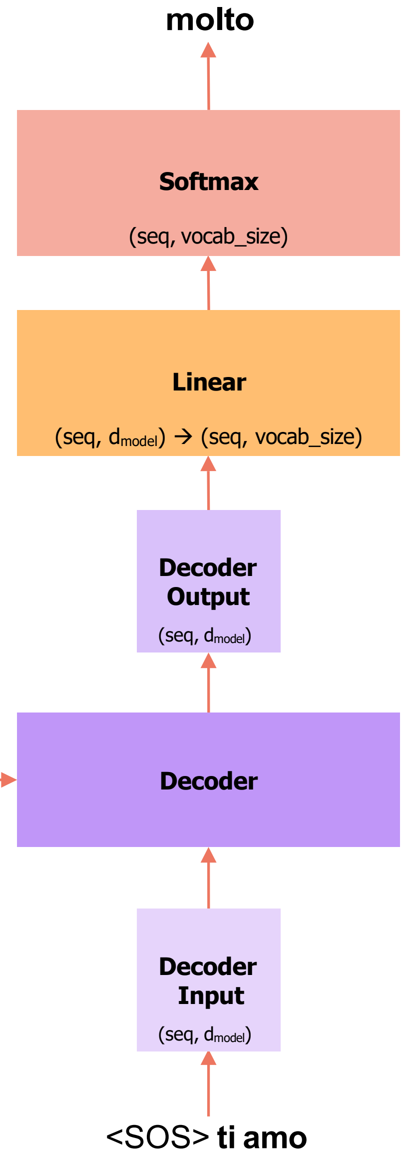


Append the previously output word to the decoder input

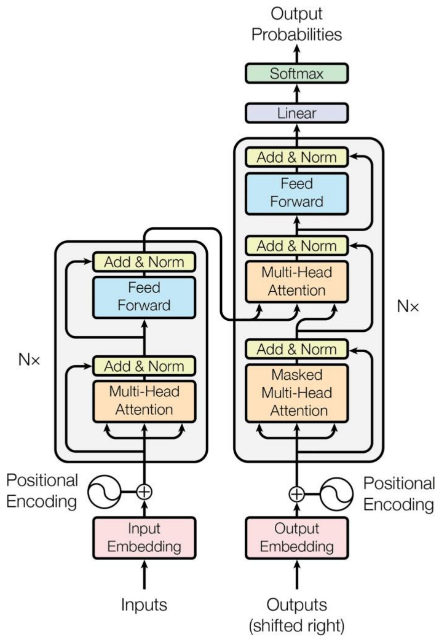
https://github.com/VijayKanagaraj7/Transformers

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

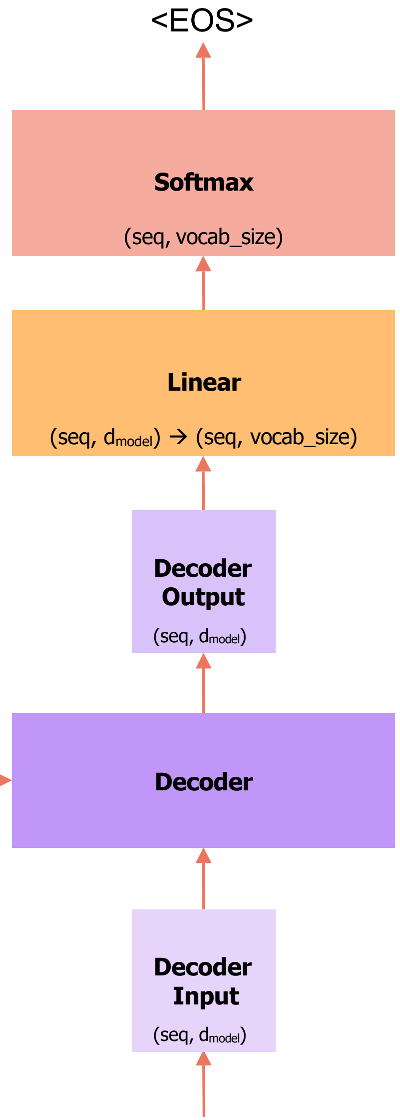
<SOS>**I love food very much**<EOS>

Append the previously output word to the decoder input

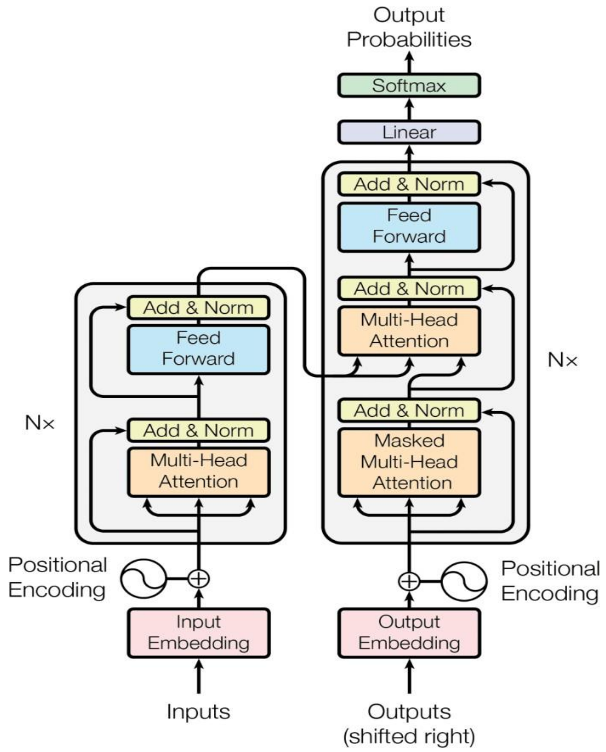
https://github.com/VijayKanagaraj7/Transformers

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 food very much**<EOS>

<SOS> **ti amo molto il cibo**

Append the previously output word to the decoder input

https://github.com/VijayKanagaraj7/Transformers

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.
- Unlike greedy decoding (which picks the highest probability token at each step), beam search keeps track of the **top-k most probable sequences (called beams)** at each decoding step, where k is the **beam width.**
- **Beam Search** is a **heuristic search algorithm** used for decoding sequences in models like **Transformers**

Why Use Beam Search in Transformers?

Transformers (like GPT, BERT for generation, T5, etc.) output probabilities over a vocabulary at each step. Beam search improves the quality of the final output by:

✓ Leads to more **context accurate** and **natural-sounding outputs**.
✓ **Explores multiple options** instead of just one path.
Increase fluency and relevance of generated text.
Generates more coherent and grammatically correct outputs