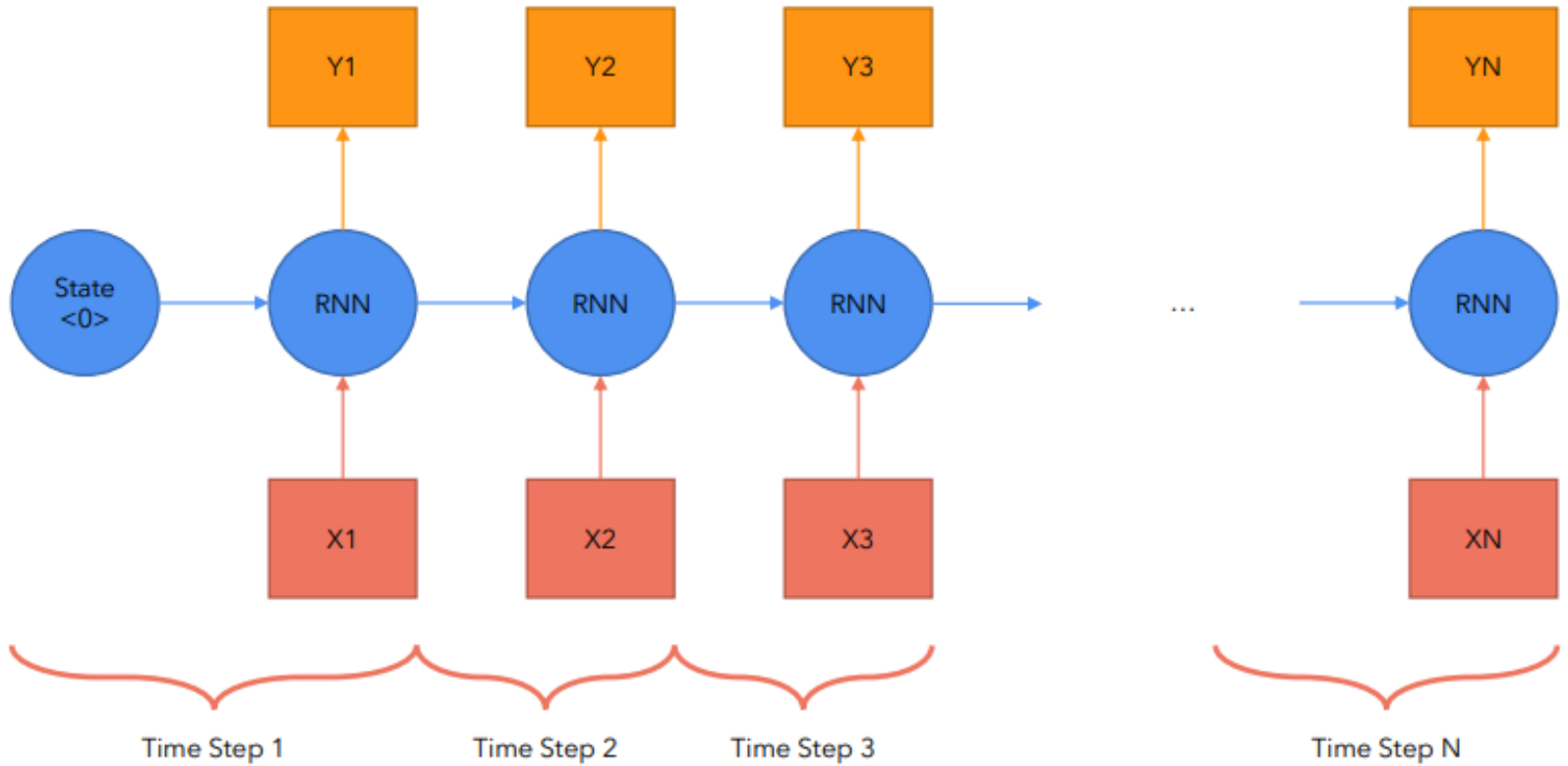

Google

Problems with RNN (among others)

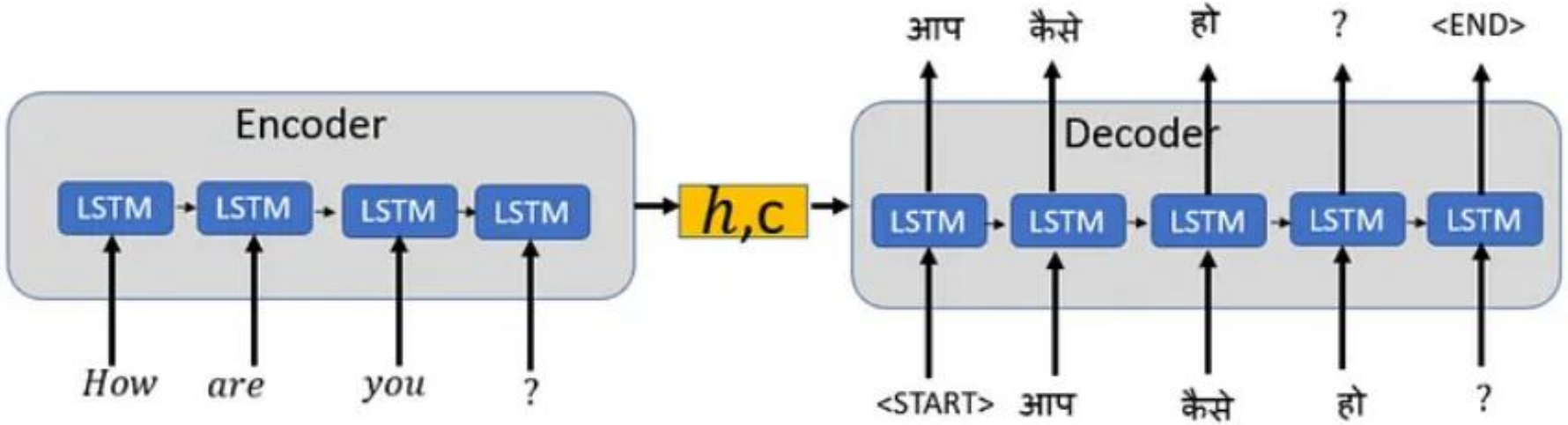
1. Slow computation for long sequences
2. Vanishing or exploding gradients
3. Difficulty in accessing information from long time ago



Bidirectional Encoder Representations from Transformers: BERT

BERT is designed to pretrain deep bidirectional representations from unlabeled text by jointly conditioning on both left and right context in all layers.

- **BERT has deep bidirectional representations meaning the model learns information from left to right and from right to left.** The bidirectional models are very powerful compared to either a left-to-right model or the shallow concatenation of a left-to-right and a right-to-left model.
- **BERT framework has two steps: pre-training and fine-tuning**
- **It is pre-trained from unlabeled data extracted from BooksCorpus** (800M words) **and English Wikipedia** (2,500M words)
- **BERT pre-trained model can be fine-tuned with just one additional output layer to solve multiple NLP tasks** like Text Summarization, Sentiment Analysis, Question-Answer chatbots, Machine Translation, etc.
- A distinctive feature of BERT is its **unified architecture across different tasks.** There is a minimal difference between the pre-trained architecture and the architecture used for various down-stream tasks.
- **BERT uses the Masked Language Model (MLM) to use the left and the right context during pre-training to create a deep bidirectional Transformers.**



Encoder-Decoder

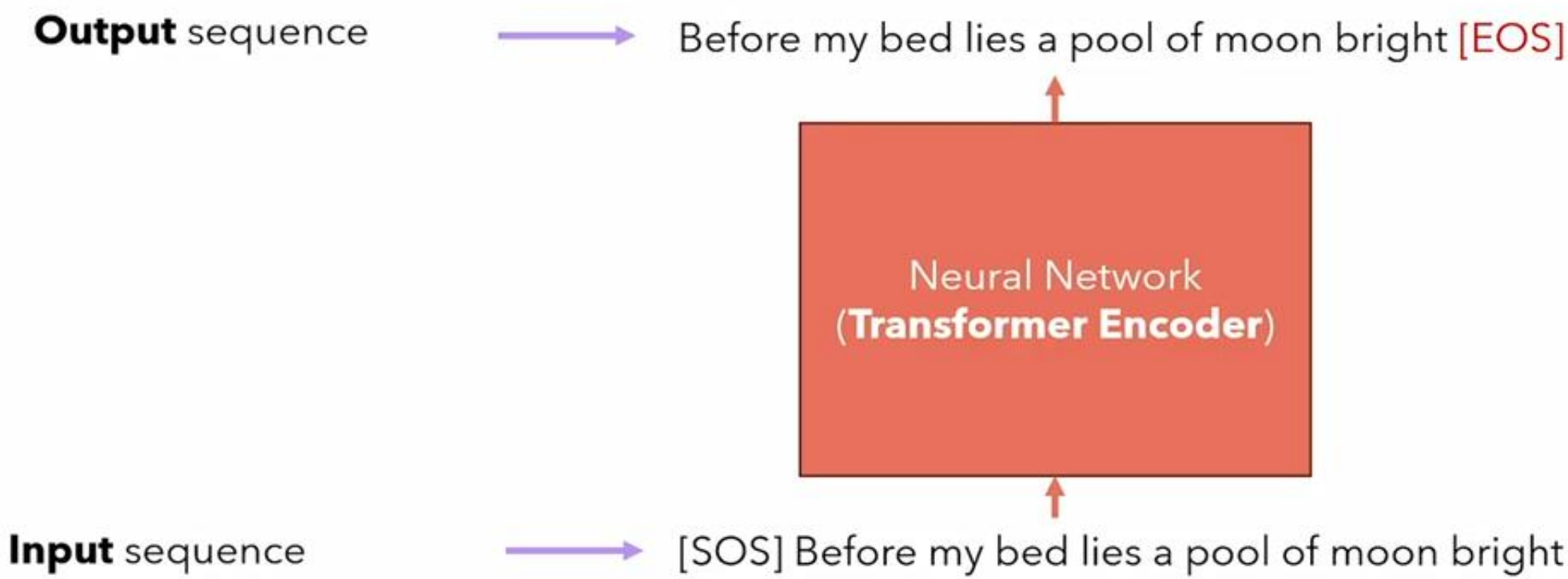
You can now translate short sentences, but to translate longer sentences, you need to pay attention to certain words in the sentence to understand the context better. This is done by adding an Attention mechanism to the Encoder-Decoder model. **The attention mechanism** allows you to pay attention to specific input words in the sentence to do a better job translating but still reading word by word in a sentence.

What is a language model?

A language model is a probabilistic model that assign probabilities to sequence of words. In practice, a language model allows us to compute the following:

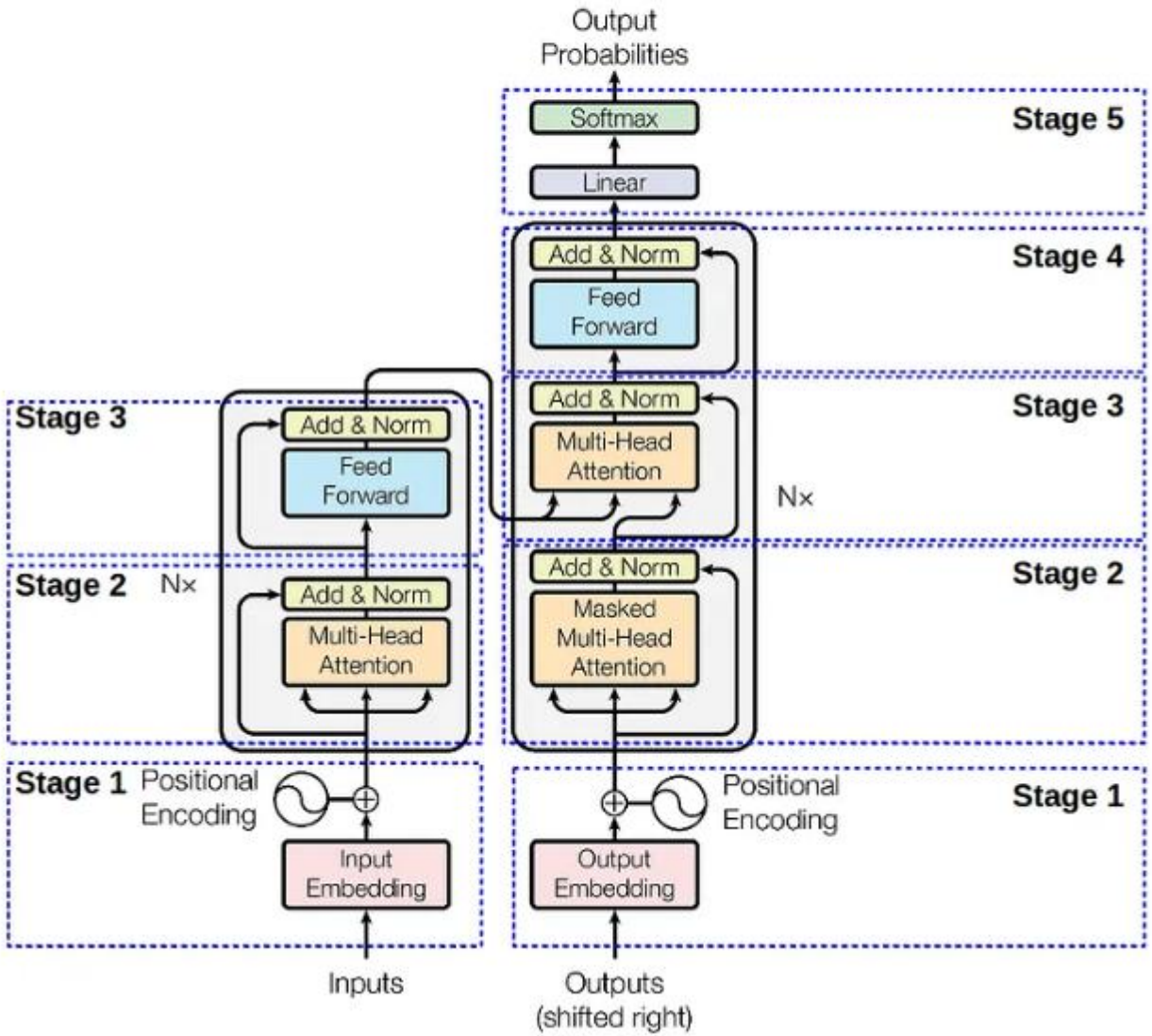
We usually train a neural network to predict these probabilities. A neural network trained on a large corpora of text is known as a Large Language Model (LLM).

How to inference a language model?



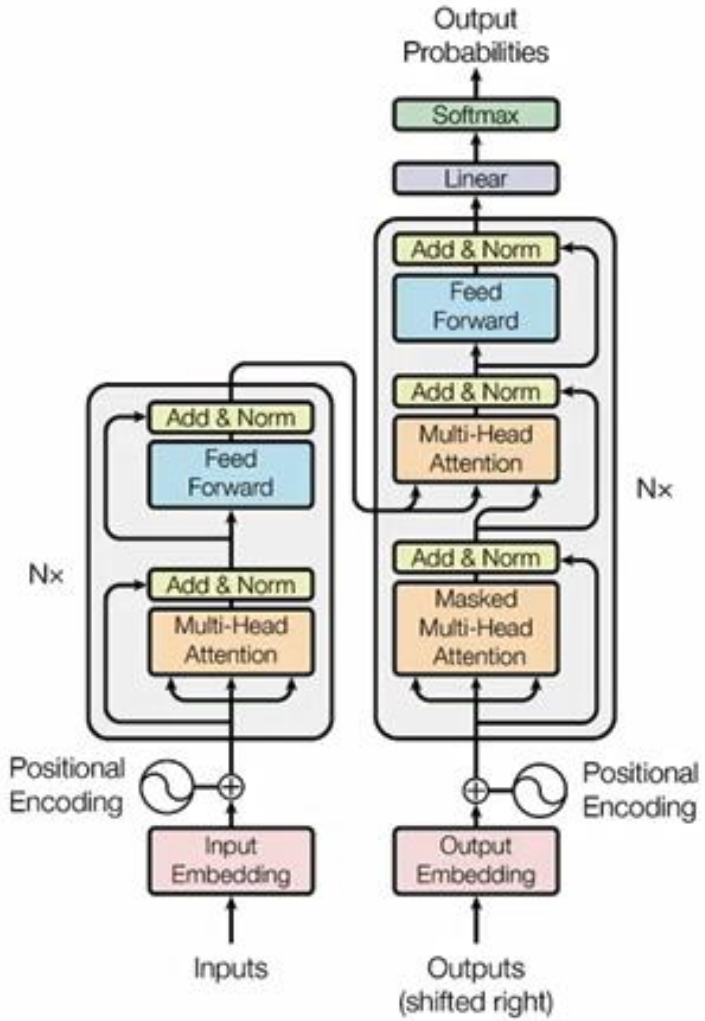
1. BERT Model Architecture:

BERT Model architecture is a multi-layer bidirectional Transformer encoder-decoder structure.

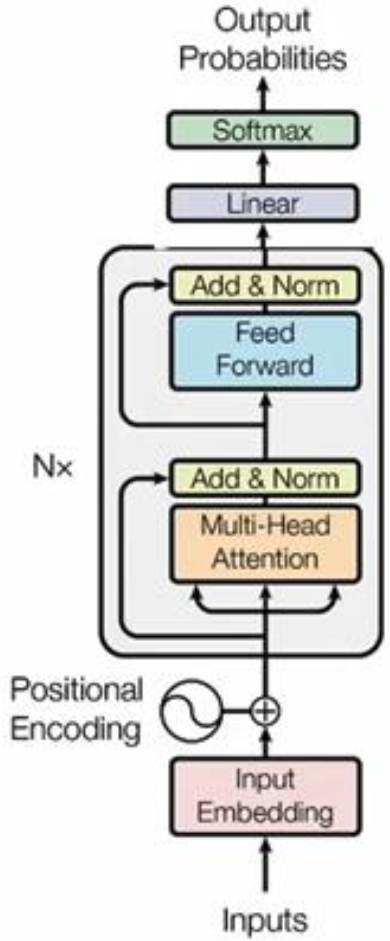


- **Encoder**: Encoder is composed of a stack of N=6 identical layers. Each layer has two sub layers. The first layer is a multi-head self-attention mechanism and the second is a position wise fully connected feed-forward network. There is a residual connection around each of the two sub layers followed by layer normalization.
- **Decoder**: Decoder is also composed of N=6 identical layers. Decoder has additional one sub-layer over two sub-layers as present in encoder, which performs multi-head attention over the output of the encoder stack. Similar to encoder we have residual connection around every sub-layers followed by layer normalization.
- **Attention**: Attention is a mechanism to know which word in the context better contributes to the current word. It is calculated using the dot product between query vector Q and key vector K. The output from attention head is the weighted sum of value vector V, where the weights assigned to each value is computed by a compatibility function of the Query with the corresponding Key.

Transformer Encoder architecture



Transformer



Transformer Encoder

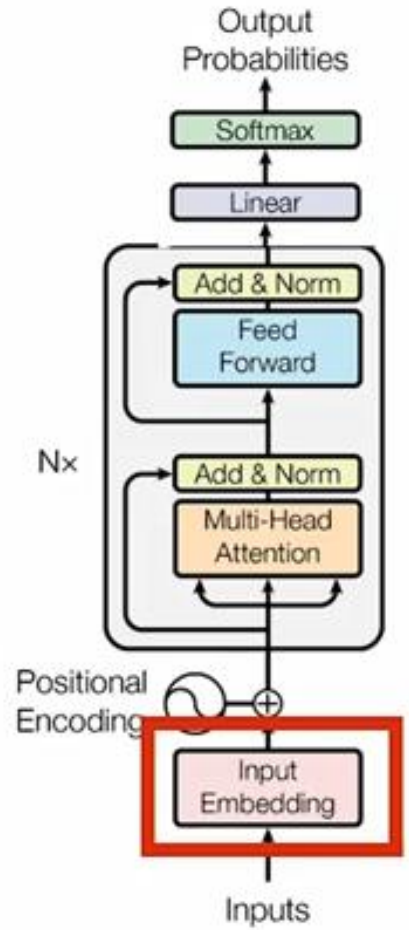
Let's convert the input into **Input Embeddings!**

Original sentence (tokens)

Input IDs (position in the vocabulary)

Embedding (vector of size 512)

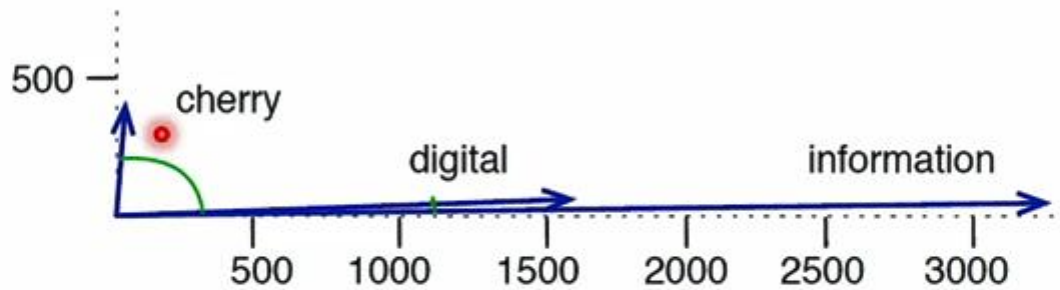
[SOS]	Before	my	bed	lies	a	pool	of	moon	bright
1	90	231	413	559	952	421	7540	62	864
3552.566	9980.851	6666.314	7512.261	5463.142	3571.487	2128.306	952.207	3065.914	5555.992
2745.925	8373.997	6239.623	8207.994	8669.221	9007.898	1685.236	5450.840	8145.629	5722.099
...	...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...	...
1070.708	8752.749	4611.106	6827.572	9521.112	9664.859	9648.558	1.658	5491.627	3623.291
1652.976	4445.452	1937.651	3222.745	9338.361	1971.318	7568.973	2671.529	1746.477	9791.989



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

Why do we use vectors to represent words?

Given the words "**cherry**", "**digital**" and "**information**", if we represent the embedding vectors using only 2 dimensions (X, Y) and we plot them, we hope to see something like this: the angle between words with similar meaning is small, while the angle between words with different meaning is big. So, the embeddings "capture" the meaning of the words they represent by projecting them into a high-dimensional space of size d_{model}.

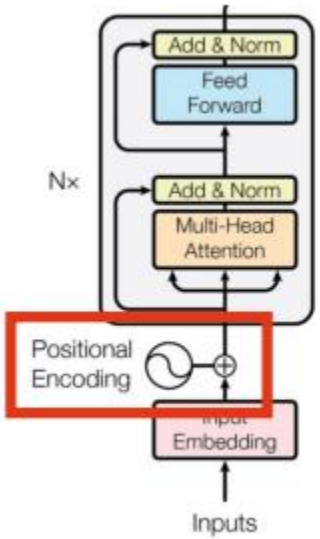


Source: Speech and Language Processing 3rd Edition Draft, Dan Jurafsky and James H. Martin

We commonly use the **cosine similarity**, which is based on the **dot product** between the two vectors.

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.



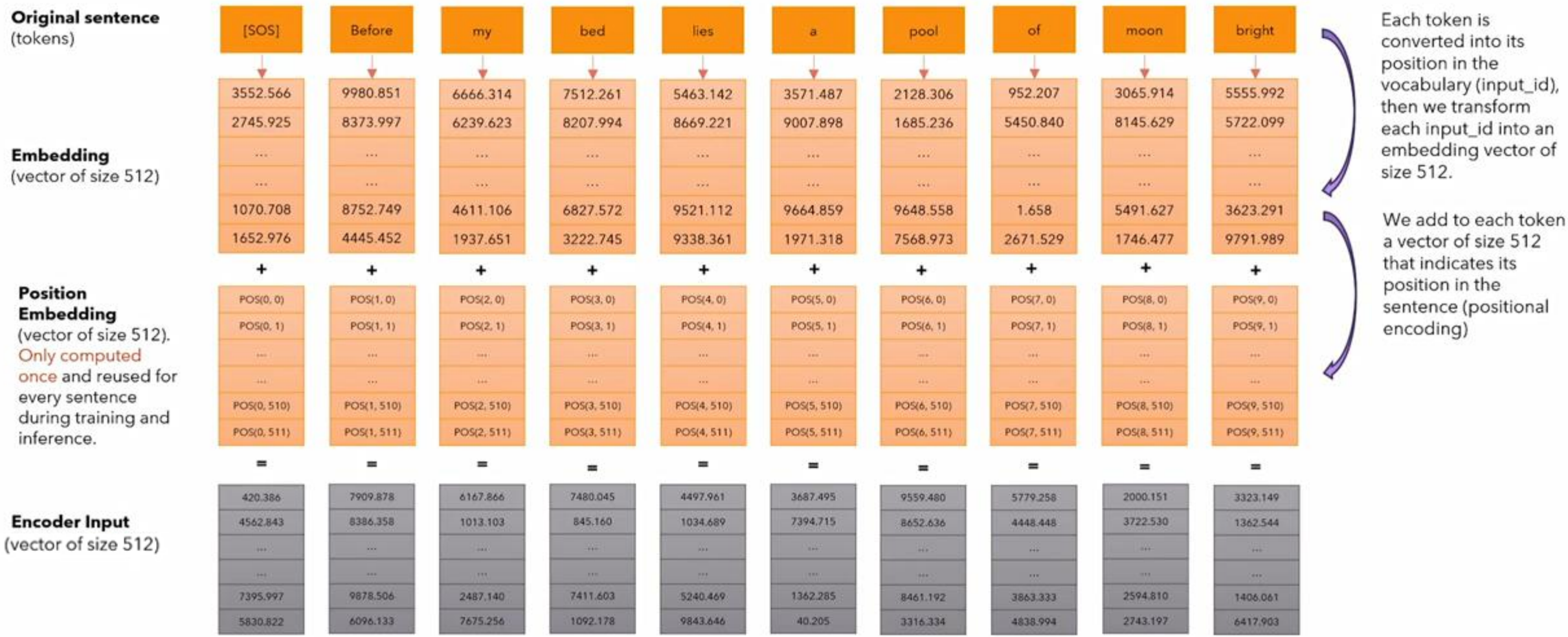
Each token is converted into its position in the vocabulary (input_id), then we transform each input_id into an embedding vector of size 512.

We add to each token a vector of size 512 that indicates its position in the sentence (positional encoding)

Position Embedding (vector of size 512). Only computed once and reused for every sentence during training and inference.

POS(0, 0)	POS(1, 0)	POS(2, 0)	POS(3, 0)	POS(4, 0)	POS(5, 0)	POS(6, 0)	POS(7, 0)	POS(8, 0)	POS(9, 0)
POS(0, 1)	POS(1, 1)	POS(2, 1)	POS(3, 1)	POS(4, 1)	POS(5, 1)	POS(6, 1)	POS(7, 1)	POS(8, 1)	POS(9, 1)
...	...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...	...
POS(0, 510)	POS(1, 510)	POS(2, 510)	POS(3, 510)	POS(4, 510)	POS(5, 510)	POS(6, 510)	POS(7, 510)	POS(8, 510)	POS(9, 510)
POS(0, 511)	POS(1, 511)	POS(2, 511)	POS(3, 511)	POS(4, 511)	POS(5, 511)	POS(6, 511)	POS(7, 511)	POS(8, 511)	POS(9, 511)

Let's add **Positional Encodings!**

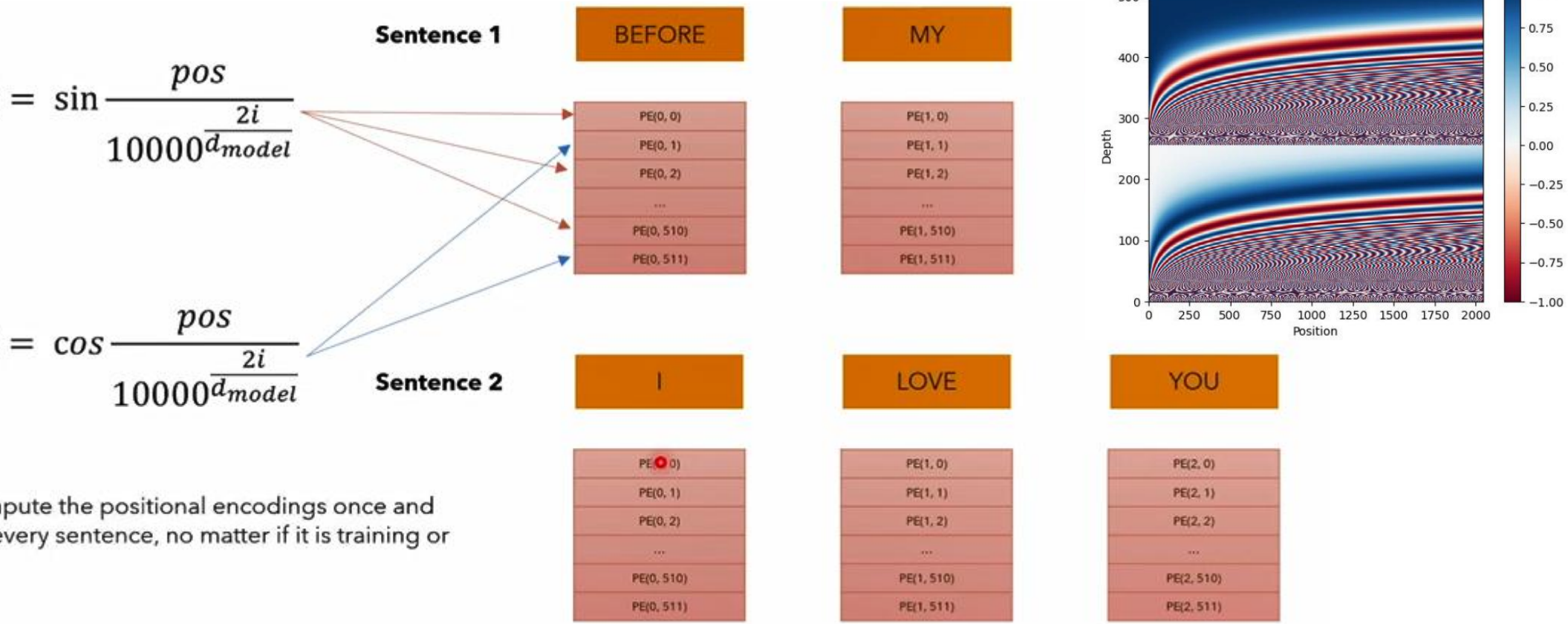


How to compute positional encodings?

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.

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

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



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

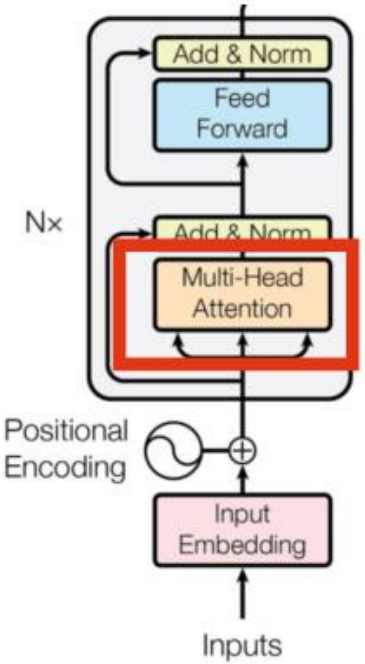
The self-attention mechanism: input

Original sentence
(tokens)

Encoder Input
(vector of size 512)

[SOS]	Before	my	bed	lies	a	pool	of	moon	bright
420.386	7909.878	6167.866	7480.045	4497.961	3687.495	9559.480	5779.258	2000.151	3323.149
4562.843	8386.358	1013.103	845.160	1034.689	7394.715	8652.636	4448.448	3722.530	1362.544
...	...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...	...
7395.997	9878.506	2487.140	7411.603	5240.469	1362.285	8461.192	3863.333	2594.810	1406.061
5830.822	6096.133	7675.256	1092.178	9843.646	40.205	3316.334	4838.994	2743.197	6417.903

Each token is converted into its position in the vocabulary (input_id), then we transform each input_id into an embedding vector of size 512 and add its position vector (positional encoding).



[SOS]	420.386	4562.843	...	...	7395.997	5830.822
Before	7909.878	8386.358	...	...	9878.506	6096.133
my	6167.866	1013.103	...	...	2487.140	7675.256
bed	7480.045	845.160	...	...	7411.603	1092.178
lies	4497.961	1034.689	...	...	5240.469	9843.646
a	3687.495	7394.715	...	...	1362.285	40.205
pool	9559.480	8652.636	...	...	8461.192	3316.334
of	5779.258	4448.448	...	...	3863.333	4838.994
moon	2000.151	3722.530	...	...	2594.810	2743.197
bright	3323.149	1362.544	...	...	1406.061	6417.903

Matrix of shape (10, 512) where each row represents a token in the input sequence.

The self-attention mechanism: Q, K and V

In a Large Language Models (LLM) we employ the Self-Attention mechanism, which means the **Query (Q)**, **Key (K)** and **Value (V)** are the **same matrix**.

	Query					
[SOS]	420.386	4562.843	...	...	7395.997	5830.822
Before	7909.878	8386.358	...	...	9878.506	6096.133
my	6167.866	1013.103	...	...	2487.140	7675.256
bed	7480.045	845.160	...	...	7411.603	1092.178
lies	4497.961	1034.689	...	...	5240.469	9843.646
a	3687.495	7394.715	...	...	1362.285	40.205
pool	9559.480	8652.636	...	...	8461.192	3316.334
of	5779.258	4448.448	...	...	3863.333	4838.994
moon	2000.151	3722.530	...	...	2594.810	2743.197
bright	3323.149	1362.544	...	...	1406.061	6417.903

(10, 512)

	Key					
[SOS]	420.386	4562.843	...	...	7395.997	5830.822
Before	7909.878	8386.358	...	...	9878.506	6096.133
my	6167.866	1013.103	...	...	2487.140	7675.256
bed	7480.045	845.160	...	...	7411.603	1092.178
lies	4497.961	1034.689	...	...	5240.469	9843.646
a	3687.495	7394.715	...	...	1362.285	40.205
pool	9559.480	8652.636	...	...	8461.192	3316.334
of	5779.258	4448.448	...	...	3863.333	4838.994
moon	2000.151	3722.530	...	...	2594.810	2743.197
bright	3323.149	1362.544	...	...	1406.061	6417.903

(10, 512)

	Value					
[SOS]	420.386	4562.843	...	...	7395.997	5830.822
Before	7909.878	8386.358	...	...	9878.506	6096.133
my	6167.866	1013.103	...	...	2487.140	7675.256
bed	7480.045	845.160	...	...	7411.603	1092.178
lies	4497.961	1034.689	...	...	5240.469	9843.646
a	3687.495	7394.715	...	...	1362.285	40.205
pool	9559.480	8652.636	...	...	8461.192	3316.334
of	5779.258	4448.448	...	...	3863.333	4838.994
moon	2000.151	3722.530	...	...	2594.810	2743.197
bright	3323.149	1362.544	...	...	1406.061	6417.903

(10, 512)

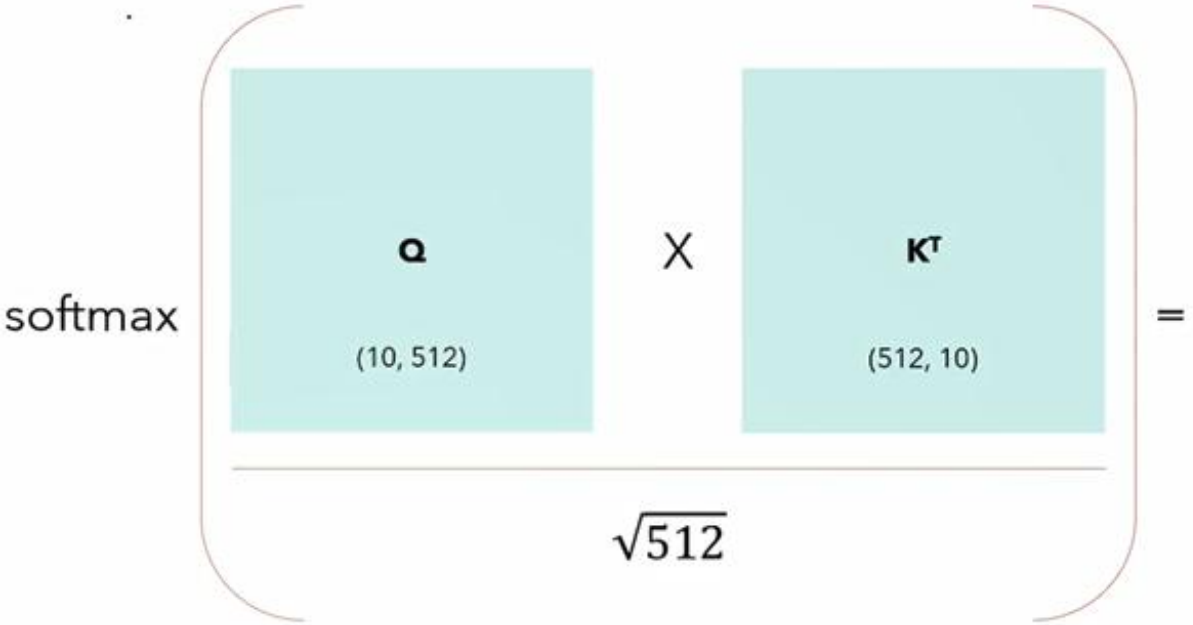
The self-attention mechanism

$$\mathbf{a} \cdot \mathbf{b} = \sum_{i=1}^{n} a_i b_i = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n$$

Softmax of the **dot product** of the word *"my"* with the word *"bed"*. Thanks to the softmax, **each row sums to 1**.

Self-Attention allows the model to relate words to each other. In our case $d_k = d_{model} = 512$.

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

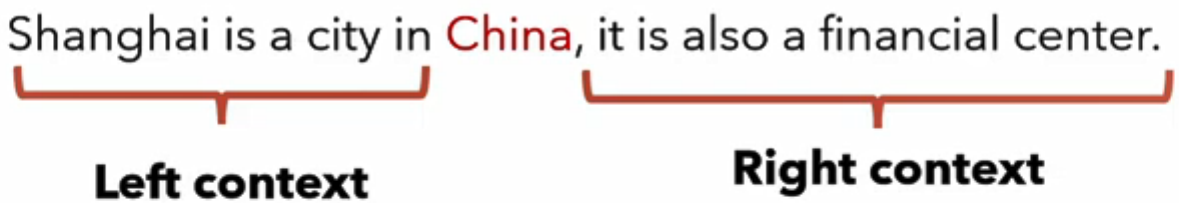


	[SOS]	Before	my	bed	lies	a	pool	of	moon	bright
[SOS]	0.62	0.19	0.02	0.02	0.04	0.01	0.00	0.09	0.00	0.02
Before	0.15	0.00	0.00	0.01	0.00	0.00	0.17	0.00	0.67	0.00
my	0.09	0.02	0.56	0.02	0.01	0.08	0.11	0.02	0.05	0.03
bed	0.10	0.06	0.03	0.00	0.53	0.12	0.01	0.11	0.00	0.04
lies	0.02	0.00	0.00	0.05	0.80	0.00	0.02	0.04	0.01	0.06
a	0.01	0.00	0.02	0.02	0.00	0.03	0.68	0.16	0.03	0.06
pool	0.00	0.16	0.02	0.00	0.03	0.56	0.00	0.00	0.22	0.01
of	0.22	0.00	0.01	0.05	0.19	0.44	0.00	0.00	0.04	0.04
moon	0.00	0.67	0.01	0.00	0.02	0.03	0.23	0.01	0.00	0.03
bright	0.06	0.00	0.03	0.03	0.43	0.21	0.03	0.06	0.13	0.03

(10, 10)

The self-attention mechanism: the reason behind the causal mask

A language model is a probabilistic model that assign probabilities to sequence of words.
In practice, a language model allows us to compute the following:



To model the probability distribution above, each word should only depend on words that come **before it** (*left context*).
We will see later that in BERT we make use of both, the left and the right context.

WE INTRODUCE THE CAUSAL MASK:

Self-Attention mechanism: causal mask

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

	[SOS]	Before	my	bed	lies	a	pool	of	moon	bright
[SOS]	**5.45**	-∞	-∞	-∞	-∞	-∞	-∞	-∞	-∞	-∞
Before	4.28	**2.46**	-∞	-∞	-∞	-∞	-∞	-∞	-∞	-∞
my	8.17	3.56	**5.54**	-∞	-∞	-∞	-∞	-∞	-∞	-∞
bed	6.71	4.13	6.76	**0.79**	-∞	-∞	-∞	-∞	-∞	-∞
lies	5.43	7.59	3.91	6.14	**9.03**	-∞	-∞	-∞	-∞	-∞
a	4.42	4.35	7.55	3.14	1.35	**7.57**	-∞	-∞	-∞	-∞
pool	8.36	6.00	4.56	0.52	3.13	6.78	**9.00**	-∞	-∞	-∞
of	2.21	3.72	4.16	6.30	0.66	6.14	7.46	**6.77**	-∞	-∞
moon	4.08	6.22	5.00	4.20	5.72	5.35	7.46	3.55	**4.70**	-∞
bright	6.43	8.88	6.17	3.65	4.54	5.22	5.51	5.55	0.64	**1.38**

$$\frac{QK^T}{\sqrt{d_k}}$$

	[SOS]	Before	my	bed	lies	a	pool	of	moon	bright
[SOS]	**1.00**	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Before	0.86	**0.14**	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
my	0.92	0.01	**0.07**	0.00	0.00	0.00	0.00	0.00	0.00	0.00
bed	0.47	0.04	0.49	**0.00**	0.00	0.00	0.00	0.00	0.00	0.00
lies	0.02	0.18	0.00	0.04	**0.75**	0.00	0.00	0.00	0.00	0.00
a	0.02	0.02	0.47	0.01	0.00	**0.48**	0.00	0.00	0.00	0.00
pool	0.31	0.03	0.01	0.00	0.00	0.06	**0.59**	0.00	0.00	0.00
of	0.00	0.01	0.02	0.15	0.00	0.12	0.47	**0.23**	0.00	0.00
moon	0.02	0.16	0.05	0.02	0.10	0.07	0.55	0.01	**0.03**	0.00
bright	0.07	0.71	0.05	0.03	0.01	0.02	0.03	0.03	0.02	**0.03**

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

Self-Attention mechanism: output sequence

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

	[SOS]	Before	my	bed	lies	a	pool	of	moon	bright
[SOS]	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Before	0.86	0.14	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
my	0.92	0.01	0.07	0.00	0.00	0.00	0.00	0.00	0.00	0.00
bed	0.47	0.04	0.49	0.00	0.00	0.00	0.00	0.00	0.00	0.00
lies	0.02	0.18	0.00	0.04	0.75	0.00	0.00	0.00	0.00	0.00
a	0.02	0.02	0.47	0.01	0.00	0.48	0.00	0.00	0.00	0.00
pool	0.31	0.03	0.01	0.00	0.00	0.06	0.59	0.00	0.00	0.00
of	0.00	0.01	0.02	0.15	0.00	0.12	0.47	0.23	0.00	0.00
moon	0.02	0.16	0.05	0.02	0.10	0.07	0.55	0.01	0.03	0.00
bright	0.07	0.71	0.05	0.03	0.01	0.02	0.03	0.03	0.02	0.03

(10, 10)

X

V

(10, 512)

=

Attention Output

(10, 512)

Each row of the "Attention Output" matrix represents the embedding of the output sequence: it captures not only the meaning of each token, not only its position, but also the interaction of each token with all the other tokens, but only the interactions for which the softmax score is not zero. All the 512 dimensions of each vector only depend on the attention scores that are non-zero.

Introducing BERT

Size & Architecture

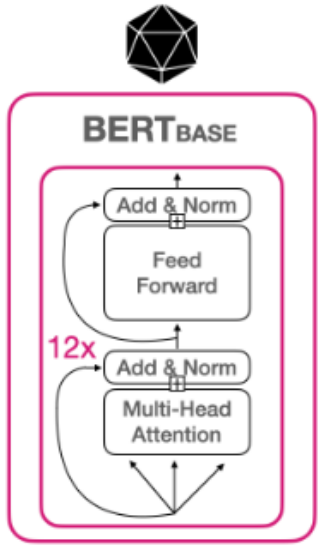
BERT's architecture is made up of layers of encoders of the Transformer mod

- $BERT_{BASE}$
  - 12 encoder layers
  - The size of the hidden size of the feedforward layer is 3072
  - 12 attention heads.
- $BERT_{LARGE}$
  - 24 encoder layers
  - The size of the hidden size of the feedforward layer is 4096
  - 16 attention heads.

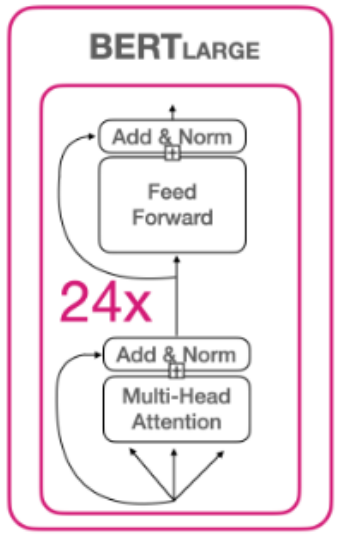
Differences with vanilla Transformer:

- The embedding vector is 768 and 1024 for the two models
- Positional embeddings are absolute and learnt during training and limited to 512 positions
- The linear layer head changes according to the application

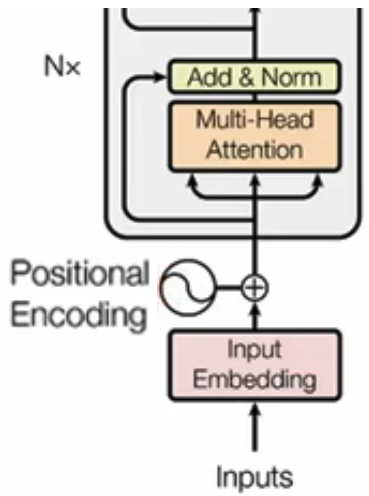
Uses the **WordPiece** tokenizer, which also allows sub-word tokens. The vocabulary size is ~ 30,000 tokens.



110M Parameters



340M Parameters



BERT vs GPT/LLaMA

BERT stands for **B**idirectional **E**ncoder **R**epresentations from **T**ransformers.

1. Unlike common language models, BERT does not handle "special tasks" with prompts, but rather, it can be specialized on a particular task by means of fine-tuning.
2. Unlike common language models, BERT has been trained using the *left context* and the *right context*.
3. Unlike common language models, BERT is not built specifically for text generation.
4. Unlike common language models, BERT has not been trained on the Next Token Prediction task, but rather, on the Masked Language Model and Next Sentence Prediction task.

*common language models = GPT, LLaMA, etc.

Tasks in GPT/LLaMA vs BERT

Question Answering in GPT/LLaMA: **Prompt Engineering**

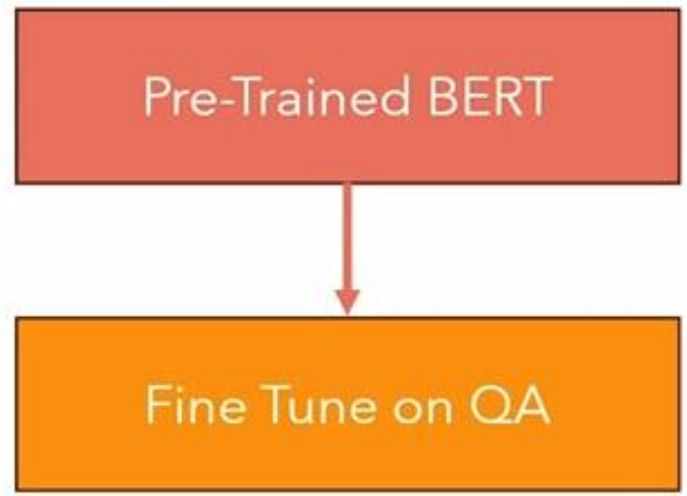
Default (GPT-3.5)

Context: "Shanghai is a City in China, it is also a financial center, its fashion capital and industrial city."
Question: "What is the fashion capital of China? Answer with one word."
Answer: Shanghai.

Context: "Shenzhen is the tech capital of China, it is also close to Hong Kong."
Question: "What city is close to Hong Kong"?
Answer:

Shenzhen.

Question Answering in BERT: **Fine Tuning**



The importance of **left** context in human conversations

Left context is used for example in phone conversations:

User: Hello! My internet line is not working, could you send a technician?

Operator: Hello! Let me check. Meanwhile, can you try restarting your WiFi router?

User: I have already restarted it but looks like the red light is not going away.

Operator: All right. I'll send someone.

The importance of **right** context in human conversations

Imagine there's a kid who just boke his mom's favorite necklace. The kid doesn't want to tell the truth to his mom, so he decides to make up a lie.

So, instead of saying directly: "Your favorite necklace has broken"

The kid may say: "Mom, I just saw the cat playing in your room and your favorite necklace has broken."

Or it may say: "Mom, aliens came through your window with laser guns and your favorite necklace has broken."

As you can see, we conditioned the lie on what we want to say next. Whatever the lie we make up, it will be always conditioned on the conclusion we want to arrive to (the necklace being broken).

Masked Language Model (MLM)

Also known as the Cloze task. It means that randomly selected words in a sentence are masked, and the model must **predict the right word given the left and right context**.

Rome is the capital of Italy, which is why it hosts many government buildings.

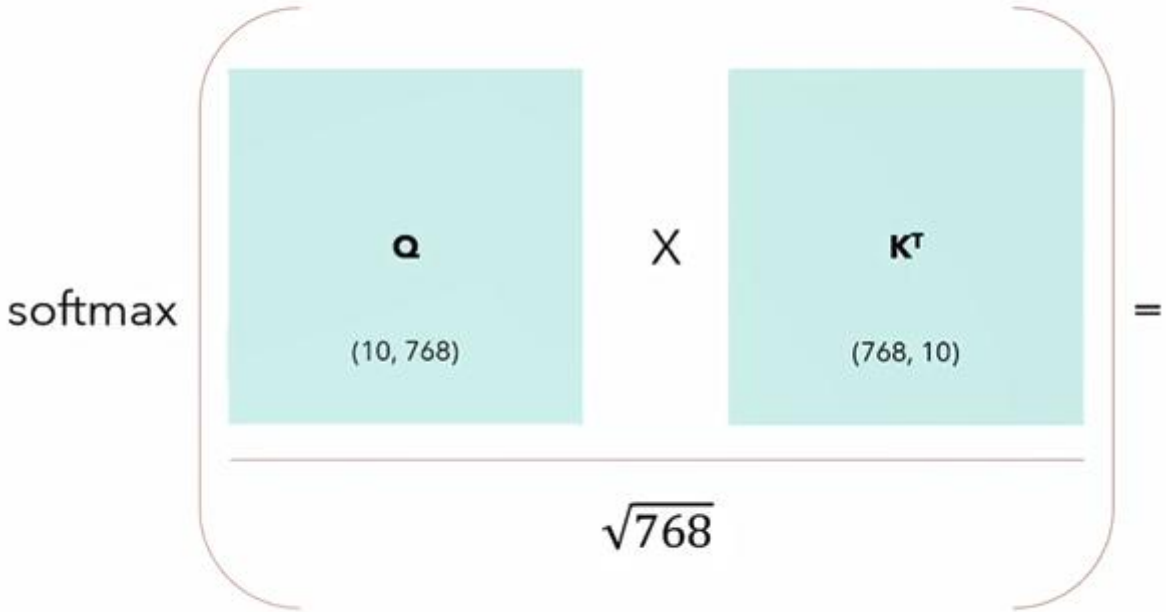
Randomly select one or more tokens and replace them with the special token **[MASK]**

Rome is the [MASK] of Italy, which is why it hosts many government buildings.

capital

Left and right context in BERT

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

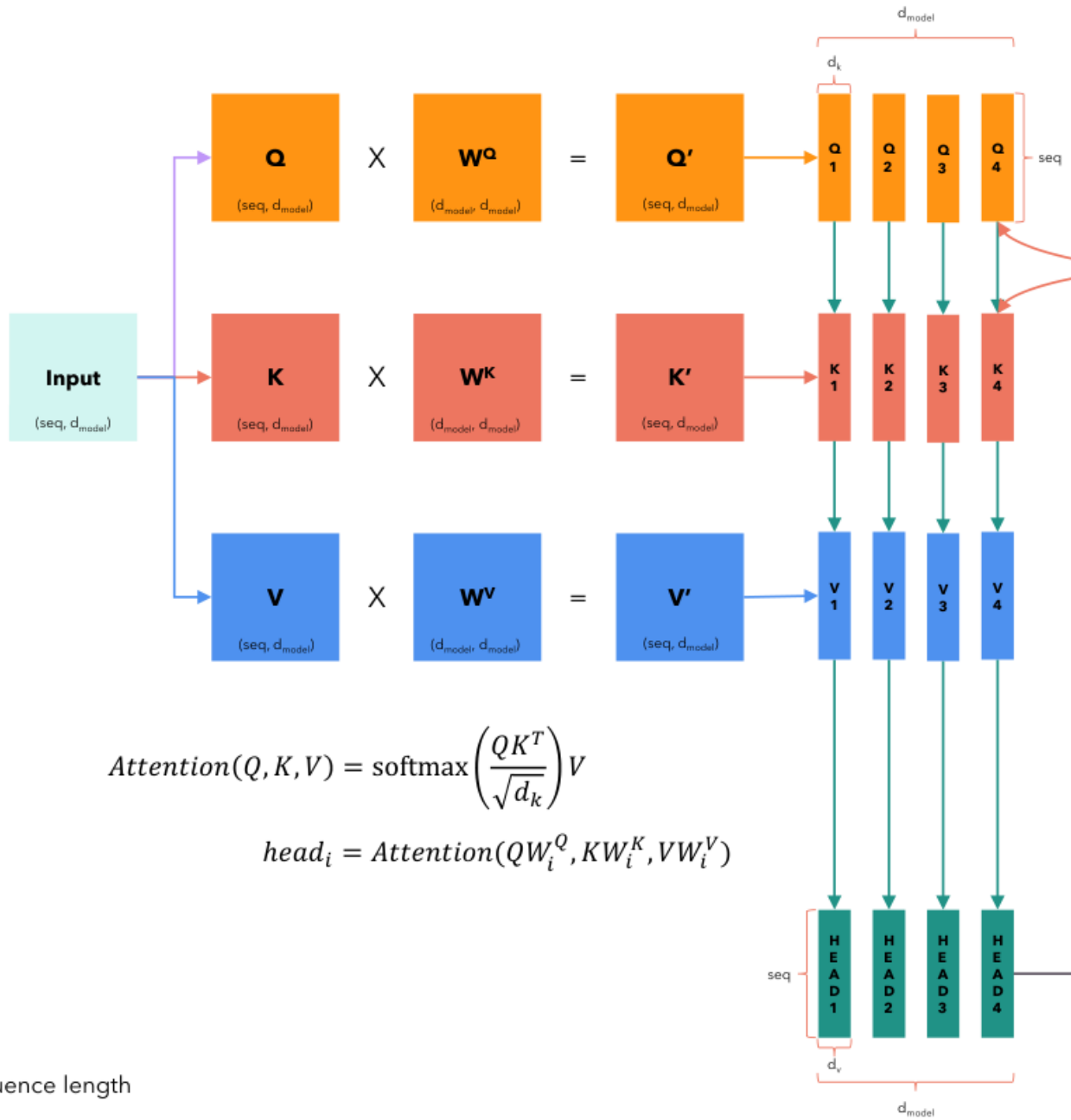


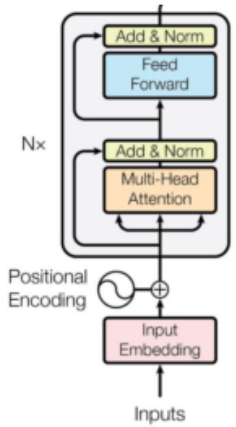
=

This is the reason it is a **Bidirectional Encoder.**
Each token "attends" token to its left and tokens to its right in a sentence.

	[SOS]	Before	my	bed	lies	a	pool	of	moon	bright
[SOS]	0.62	0.19	0.02	0.02	0.04	0.01	0.00	0.09	0.00	0.02
Before	0.15	0.00	0.00	0.01	0.00	0.00	0.17	0.00	0.67	0.00
my	0.09	0.02	0.56	0.02	0.01	0.08	0.11	0.02	0.05	0.03
bed	0.10	0.06	0.03	0.00	0.53	0.12	0.01	0.11	0.00	0.04
lies	0.02	0.00	0.00	0.05	0.80	0.00	0.02	0.04	0.01	0.06
a	0.01	0.00	0.02	0.02	0.00	0.03	0.68	0.16	0.03	0.06
pool	0.00	0.16	0.02	0.00	0.03	0.56	0.00	0.00	0.22	0.01
of	0.22	0.00	0.01	0.05	0.19	0.44	0.00	0.00	0.04	0.04
moon	0.00	0.67	0.01	0.00	0.02	0.03	0.23	0.01	0.00	0.03
bright	0.06	0.00	0.03	0.03	0.43	0.21	0.03	0.06	0.13	0.03

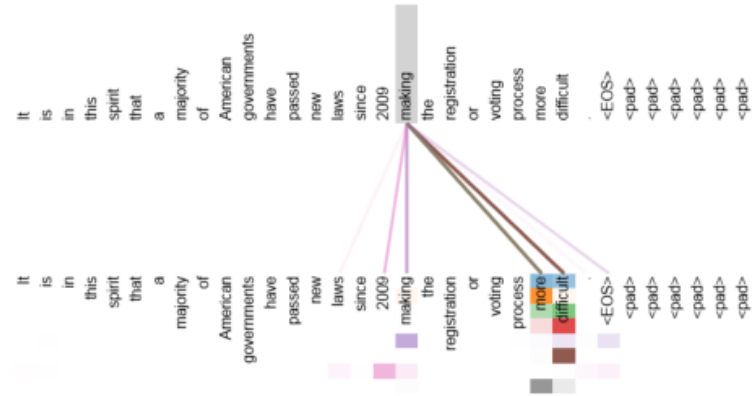
(10, 10)





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

Attention Visualizations



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

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

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

seq = sequence length

d_{model} = size of the embedding vector

h = number of heads

$d_k = d_v$ = d_{model} / h

Masked Language Model (MLM): training

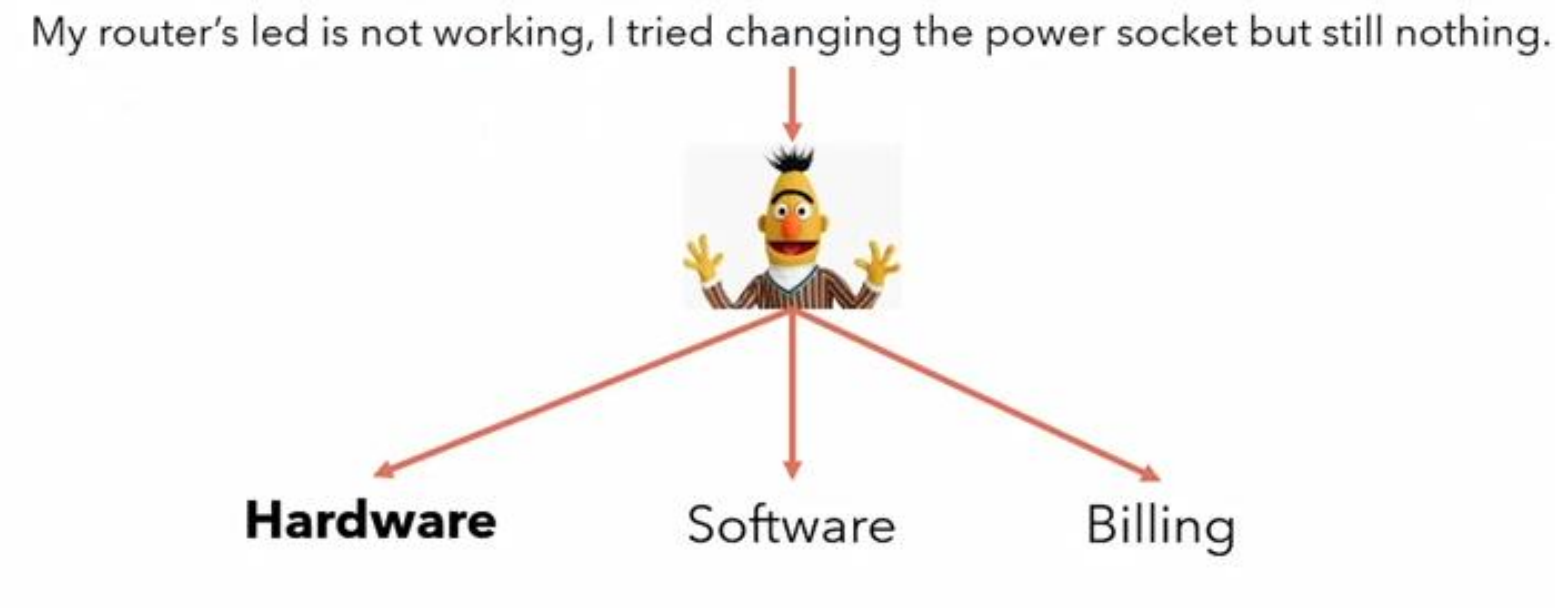
WE COMPARE CAPITAL TOKEN WITH THE TARGET, COMPUTE THE LOSS AND RUN THE BACKPROP.

Output (14 tokens):

Input (14 tokens): Rome is the [mask] of Italy, which is why it hosts many government buildings.

Text Classification

Text classification is the task of assigning a label to a piece of text. For example imagine we are running an internet provider and we receive complaints from our customers. We may want to classify requests coming from users as hardware problems, software problems or billing issues.



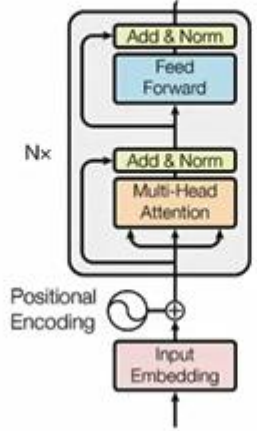
Text Classification: training

Target (1 token):

Linear Layer (3 output features) + **Softmax**

Output (16 tokens):

TK 1	TK 2	TK 3	TK 4	TK 5	TK 6	TK 7	TK 8	TK 9	TK 10	TK 11	TK 12	TK 13	TK 14



Il token [CLS] in BERT rappresenta l'intera sequenza di input ed è utilizzato per compiti di classificazione. Viene seguito da uno strato di classificazione che prende la sua rappresentazione come input, consentendo a BERT di eseguire compiti come analisi del sentiment, classificazione del testo e risposta a domande

Input (16 tokens): [CLS] My router's led is not working, I tried changing the power socket but still nothing.

Question Answering: sentence A and B

Question answering is the task of answering questions given a context.

Context: "Shanghai is a City in China, it is also a financial center, its fashion capital and industrial city."

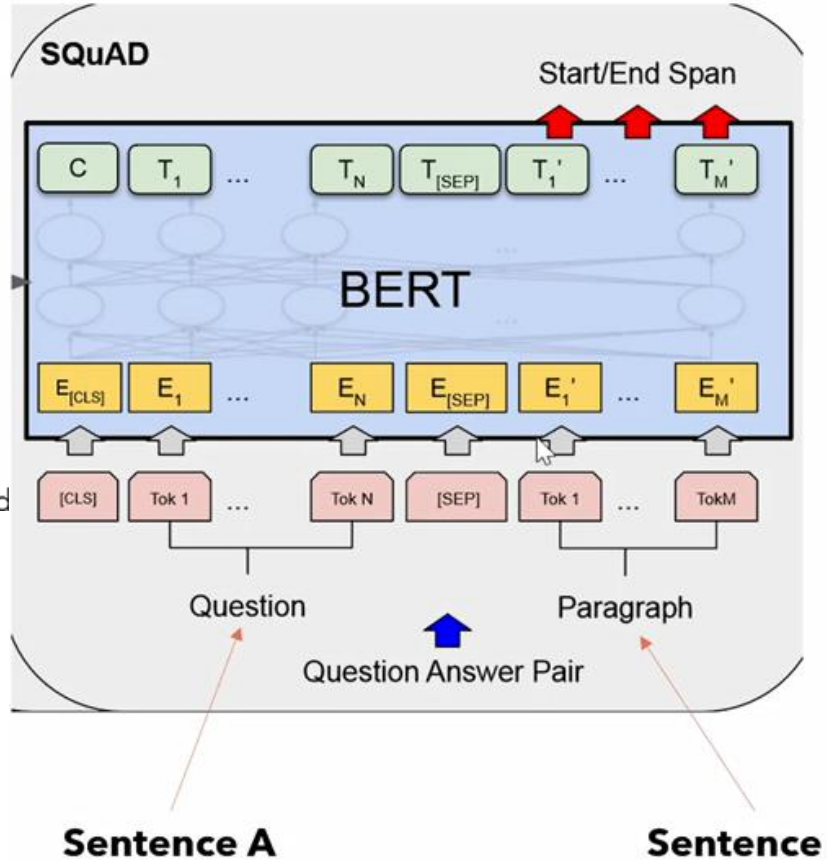
Question: "What is the fashion capital of China?"

Answer: "Shanghai is a City in China, it is also a financial center, its fashion capital and industrial city."

The model has to highlight the part of text that contains the answer.

Two problems:

1. We need to find a way for BERT to understand which part of the input is the context, which one is the question.
2. We also need to find a way for BERT to tell us where the answer starts and where it ends in the context provided



SENTENCE WILL BE ENCONDED AS SENTENCE A

THE CONTEXT WILL BE ENCODED AS SENTENCE B

Question Answering: **start** and **end** positions

Target (1 token):

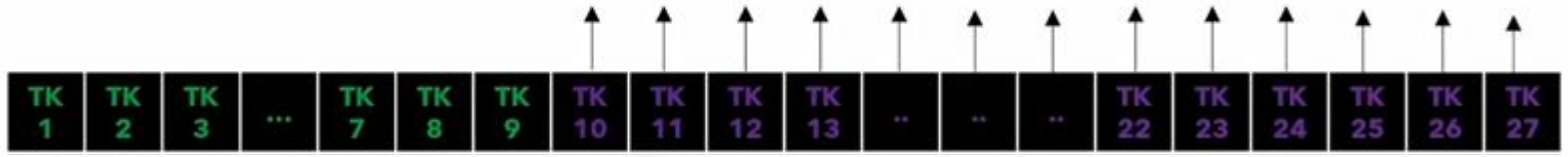
start=TK10, end=TK10

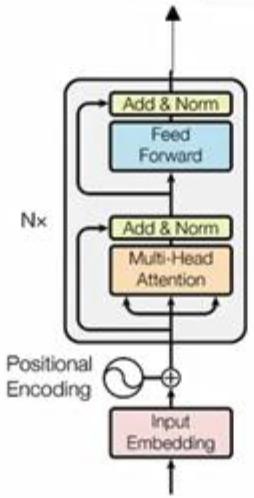
Loss → Run **backpropagation** to update the weights

2 OUTPUT FEATURES: ONE INDICATE IF PARTICULAR TOKEN IS A STARTING TOKEN AND THE OTHER ONE INDICATE IF IS AN END TOKEN

Linear Layer (2 output features) + **Softmax**

Output (27 tokens):



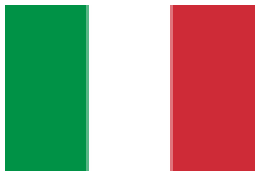


Input (27 tokens):

[CLS] What is the fashion capital of China? [SEP] Shanghai is a City in China, it is also a financial center, its fashion capital and industrial city.

TRANSLATION TASK:

I love you very much



Ti amo molto

While BERT alone can handle the encoding part, decoding requires additional components tailored for translation tasks. Complete translation systems typically incorporate both encoding and decoding steps, often utilizing architectures like the Transformer model, which combines self-attention mechanisms for capturing contextual information effectively.

Training

Time Step = 1

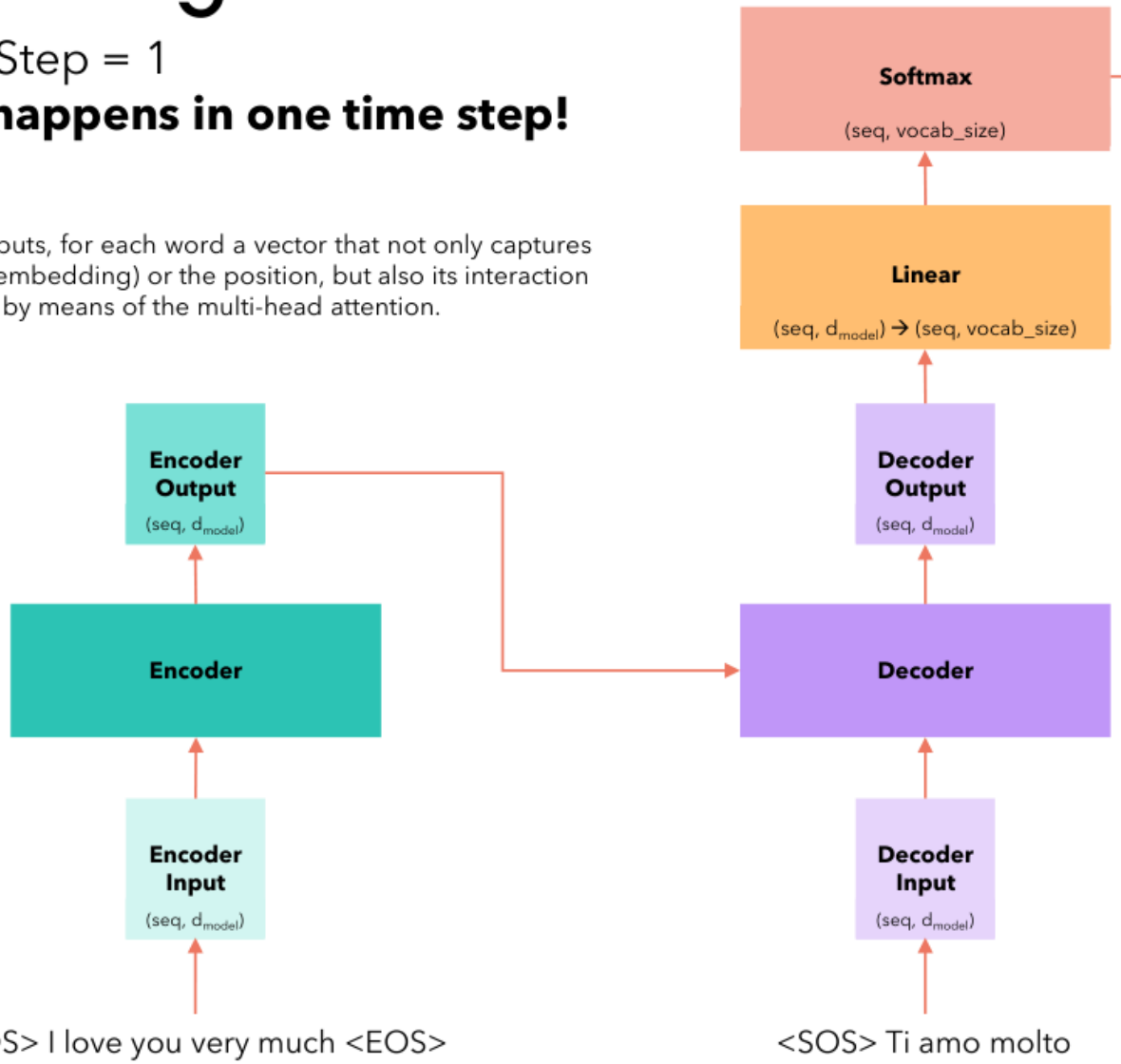
It all happens in one time step!

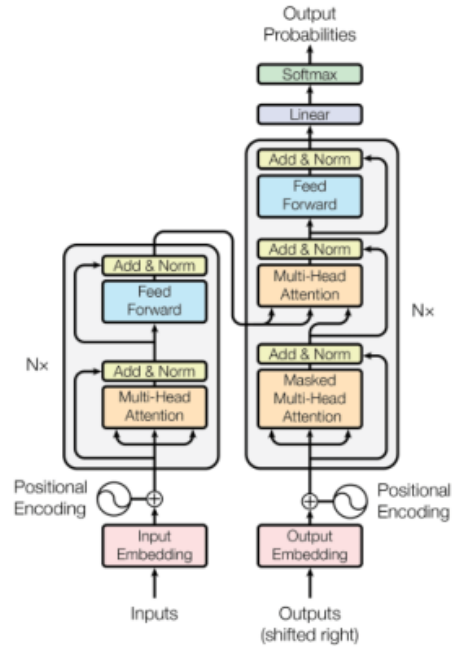
Ti amo molto <EOS>

* 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.

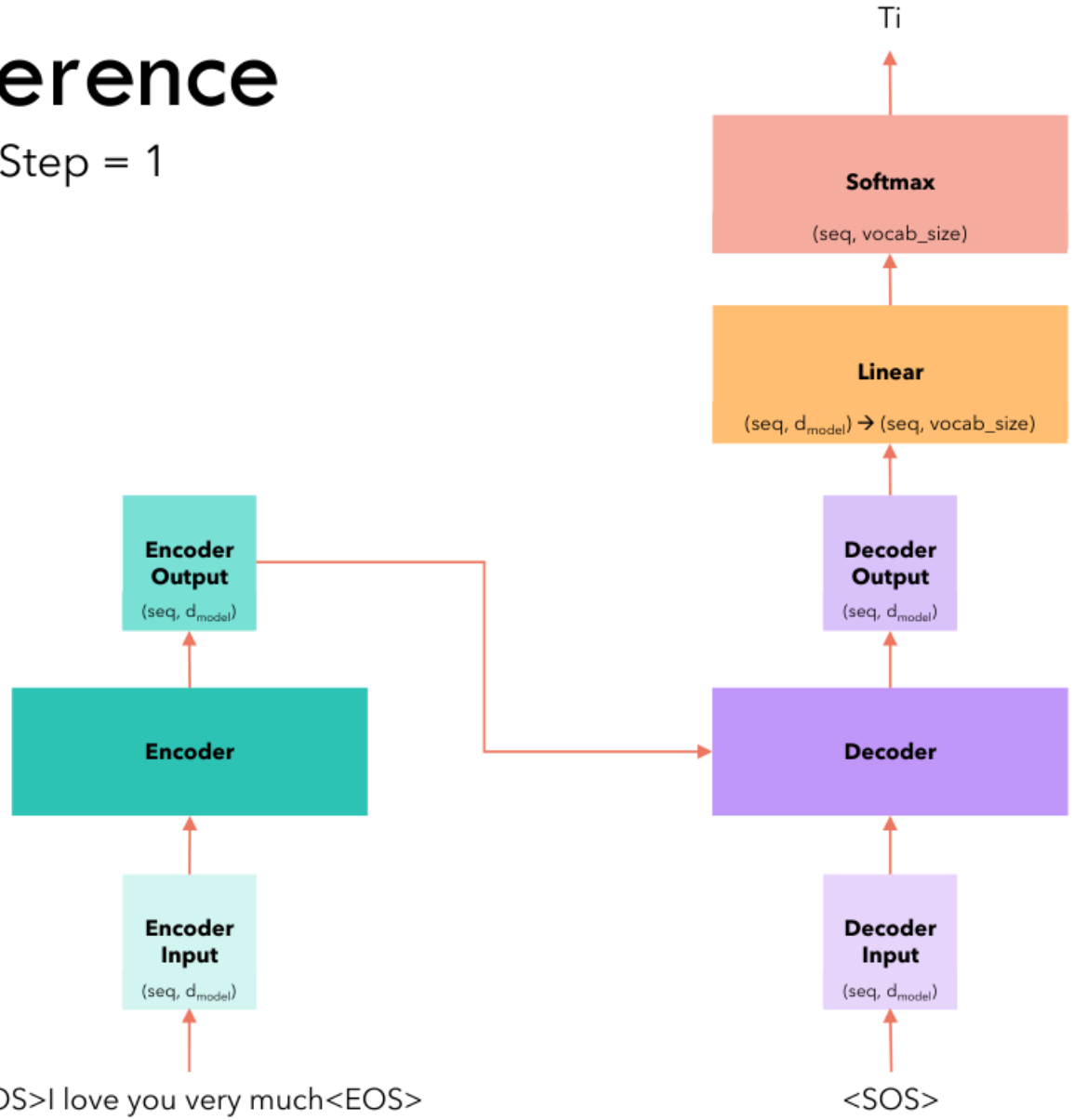




We prepend the <SOS> token at the beginning. That's why the paper says that the decoder input is shifted right.

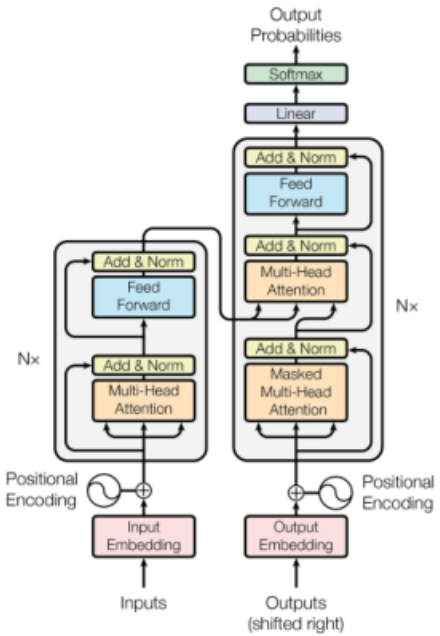
Inference

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

- 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.

Inference

Time Step = 2

amo

Softmax

(seq, vocab_size)

Linear

(seq, d_{model}) → (seq, vocab_size)

Decoder Output

(seq, d_{model})

Decoder

Use the encoder output from the first time step

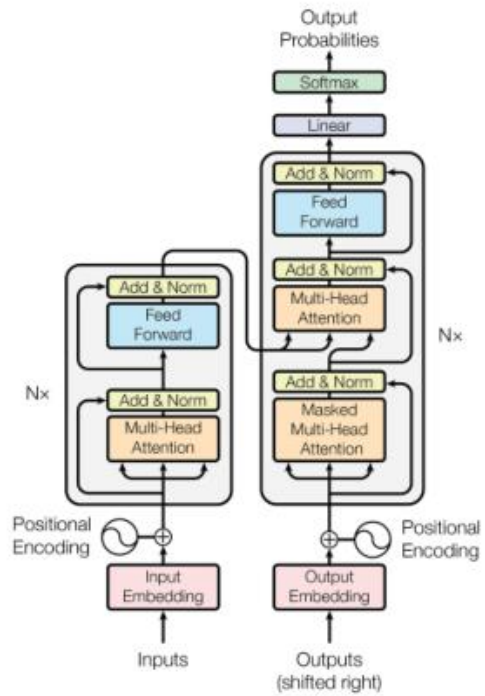
Decoder Input

(seq, d_{model})

<SOS>I love you very much<EOS>

<SOS> ti

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



Append the previously output word to the decoder input

Inference

Time Step = 3

molto

Softmax

(seq, vocab_size)

Linear

(seq, d_{model}) → (seq, vocab_size)

Decoder Output

(seq, d_{model})

Decoder

Use the encoder output from the first time step

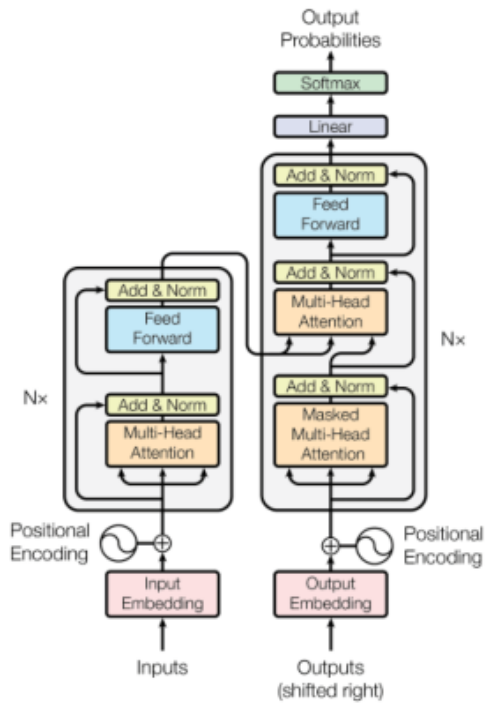
Decoder Input

(seq, d_{model})

<SOS>I love you very much<EOS>

<SOS> ti amo

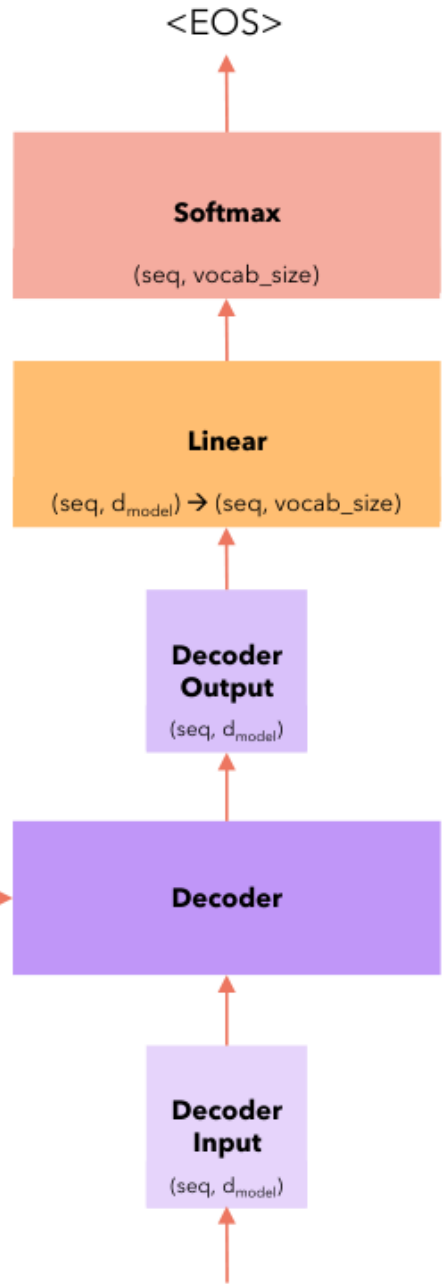
Since decoder input now contains **three** tokens, we select the softmax corresponding to the third token.



Append the previously output word to the decoder input

Inference

Time Step = 4

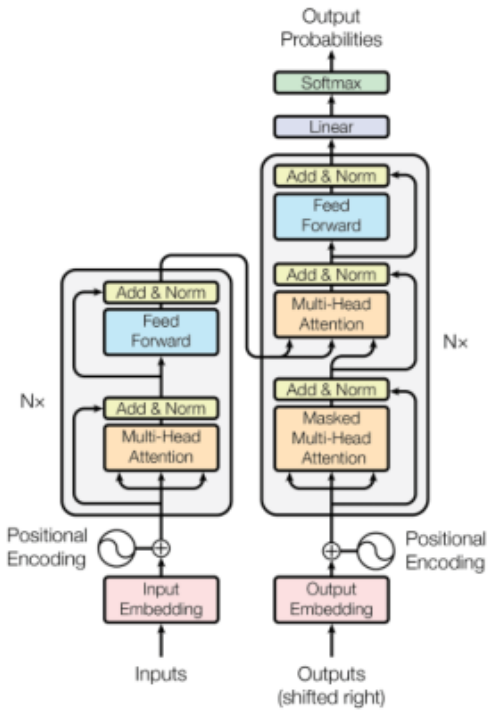


Use the encoder output from the first time step

<SOS>I love you very much<EOS>

<SOS> ti amo molto

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



Append the previously output word to the decoder input


That's all Folks!