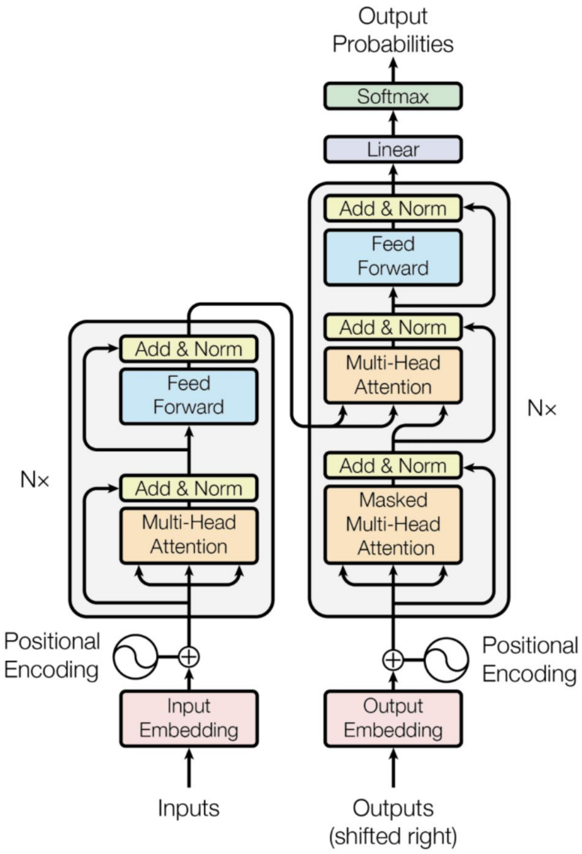
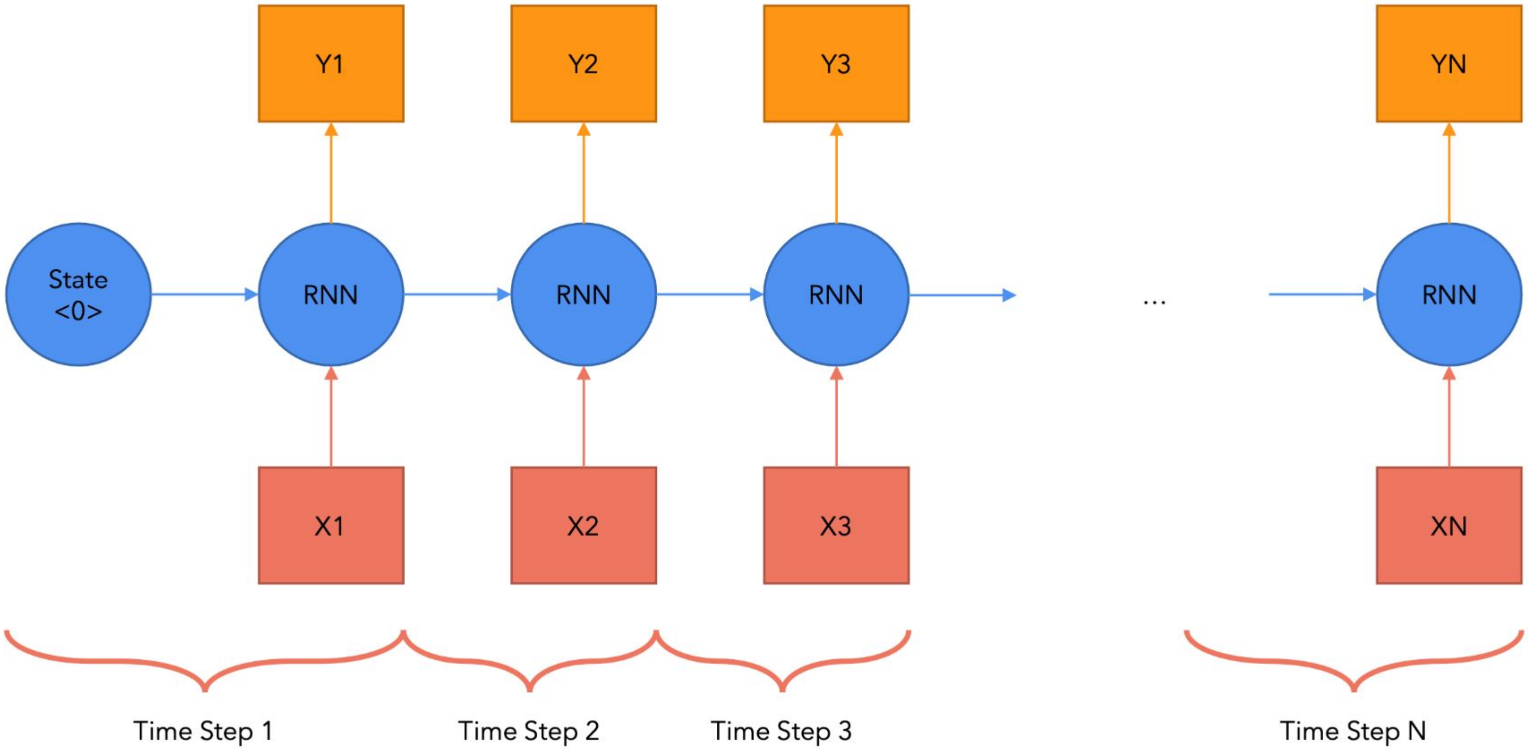


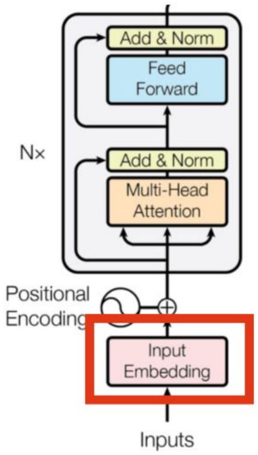

# Transformers

This is a reminder for the presenter of this presentation to not forget to start a recording of this presentation.

# Recurrent Neural Networks (RNN)



# Encoder



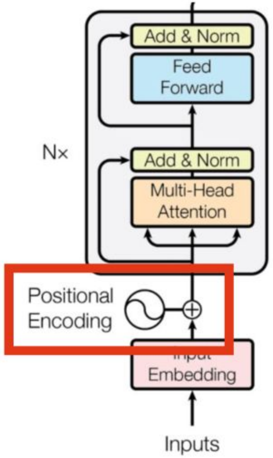
Add & Norm
Feed
Forward
N×
Add & Norm
Multi-Head
Attention
Positional
Encoding
Input
Embedding
Inputs

# 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

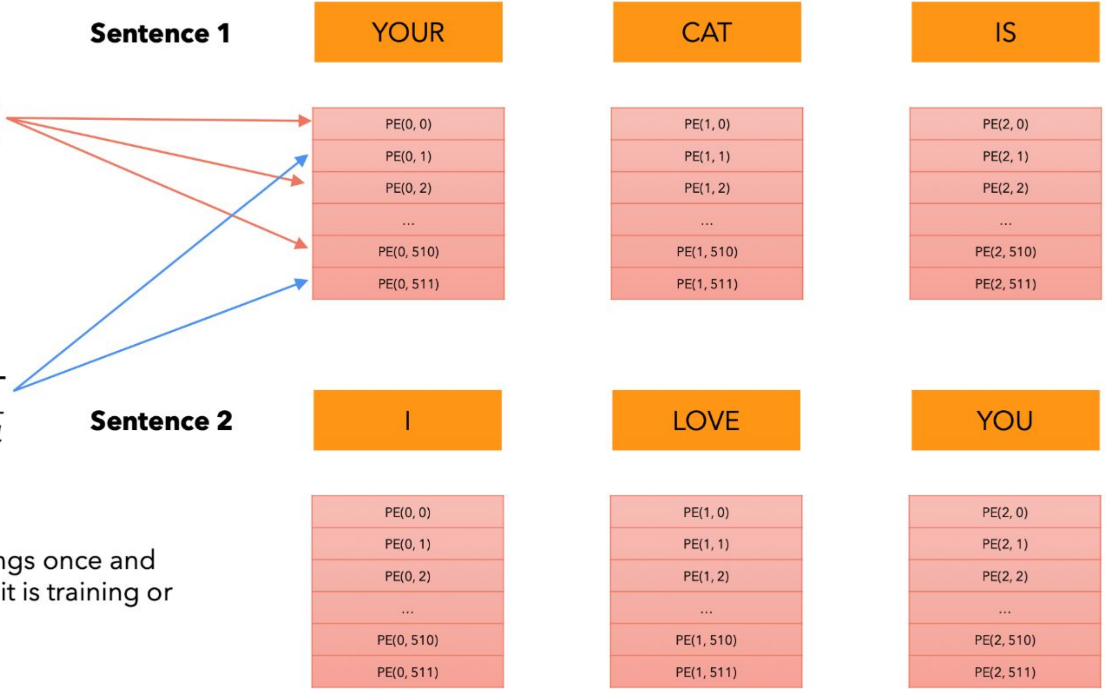
# What is positional encoding?

| | | | | | | |
|---|---|---|---|---|---|---|
| **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?

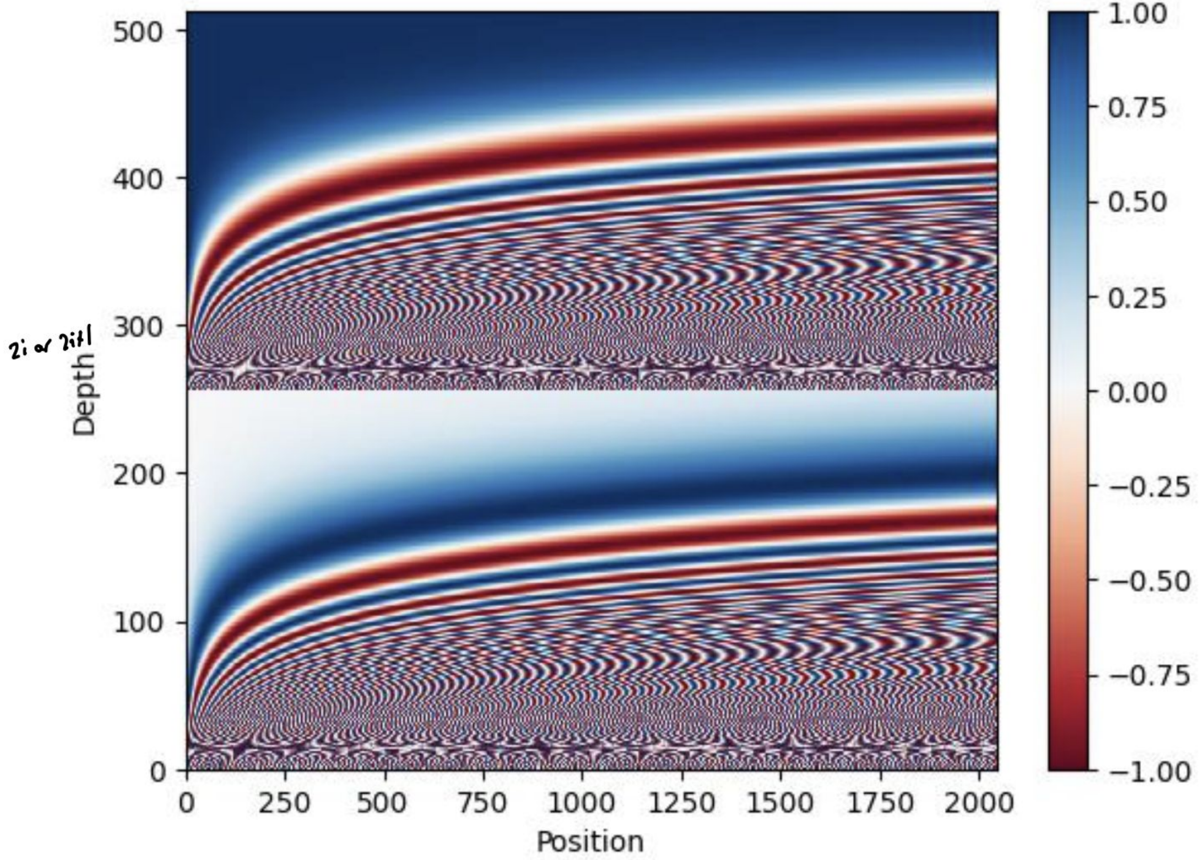
$$PE(pos, 2i) = \sin \frac{pos}{10000^{\frac{2i}{d_{model}}}}$$

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

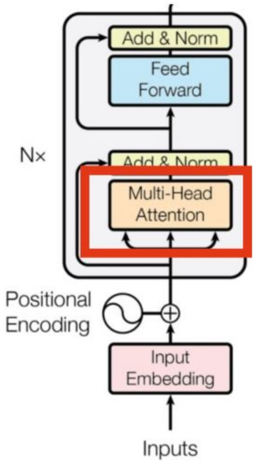



We only need to compute the positional encodings once and then reuse them for every sentence, no matter if it is training or inference.

# Why trigonometric functions?



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

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 | 0.268 | 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)

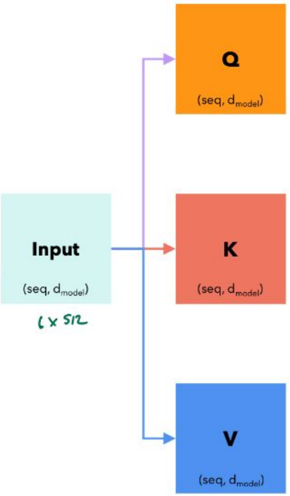
Each row in this matrix captures not only the meaning (given by the embedding) or the position in the sentence (represented by the positional encodings) but also each word's interaction with other words.

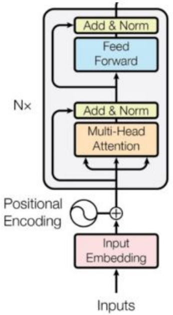
# Multi-head Attention

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

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

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

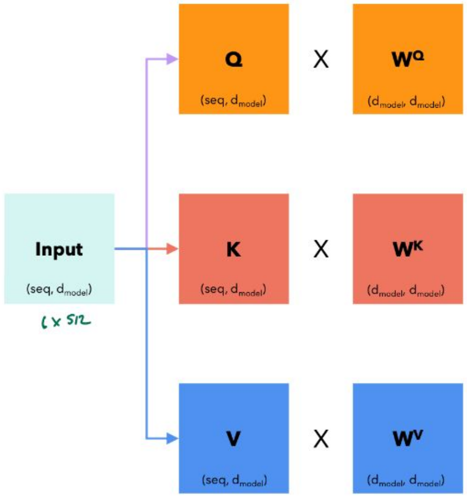


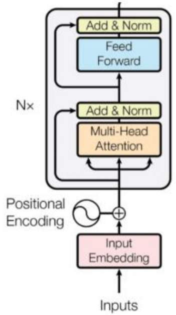




$$Attention(Q,K,V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

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

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



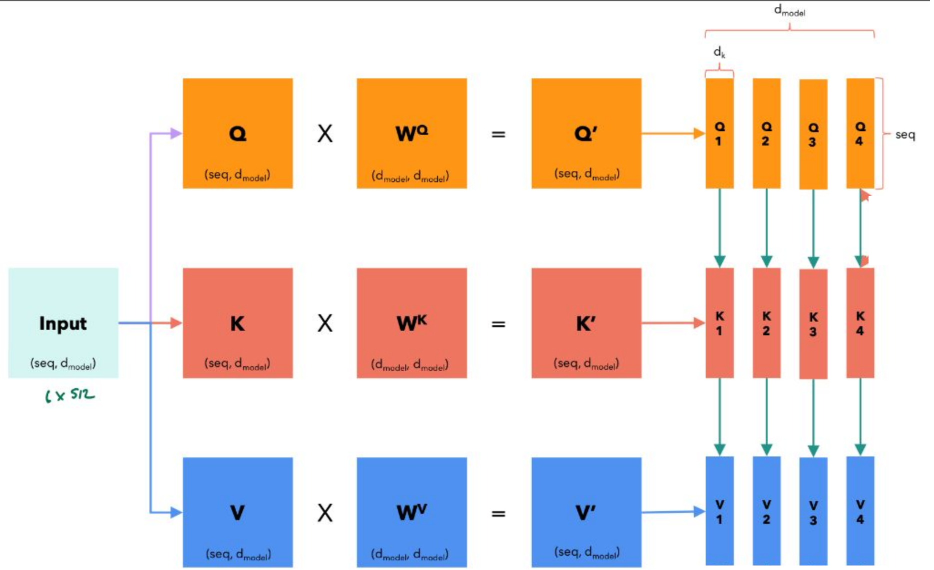



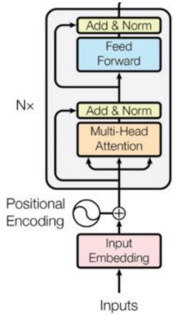

$$Attention(Q,K,V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

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

se

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



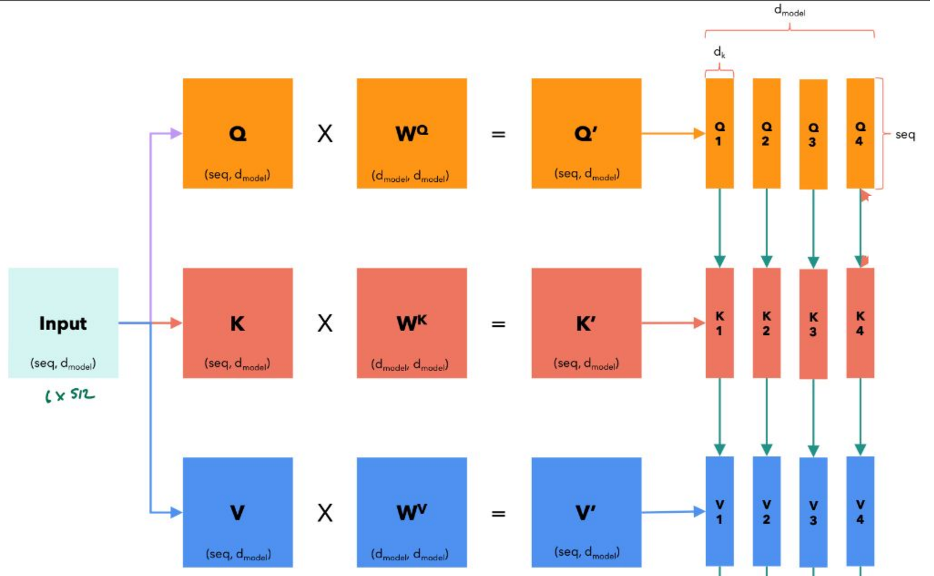



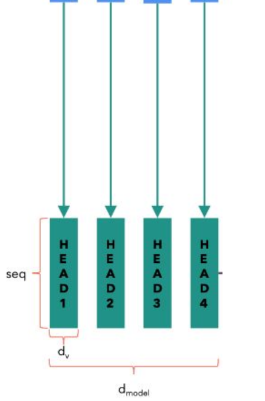

$$Attention(Q,K,V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

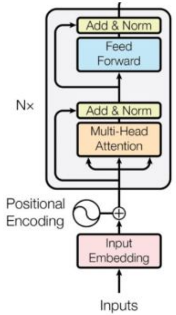
$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$$

s

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



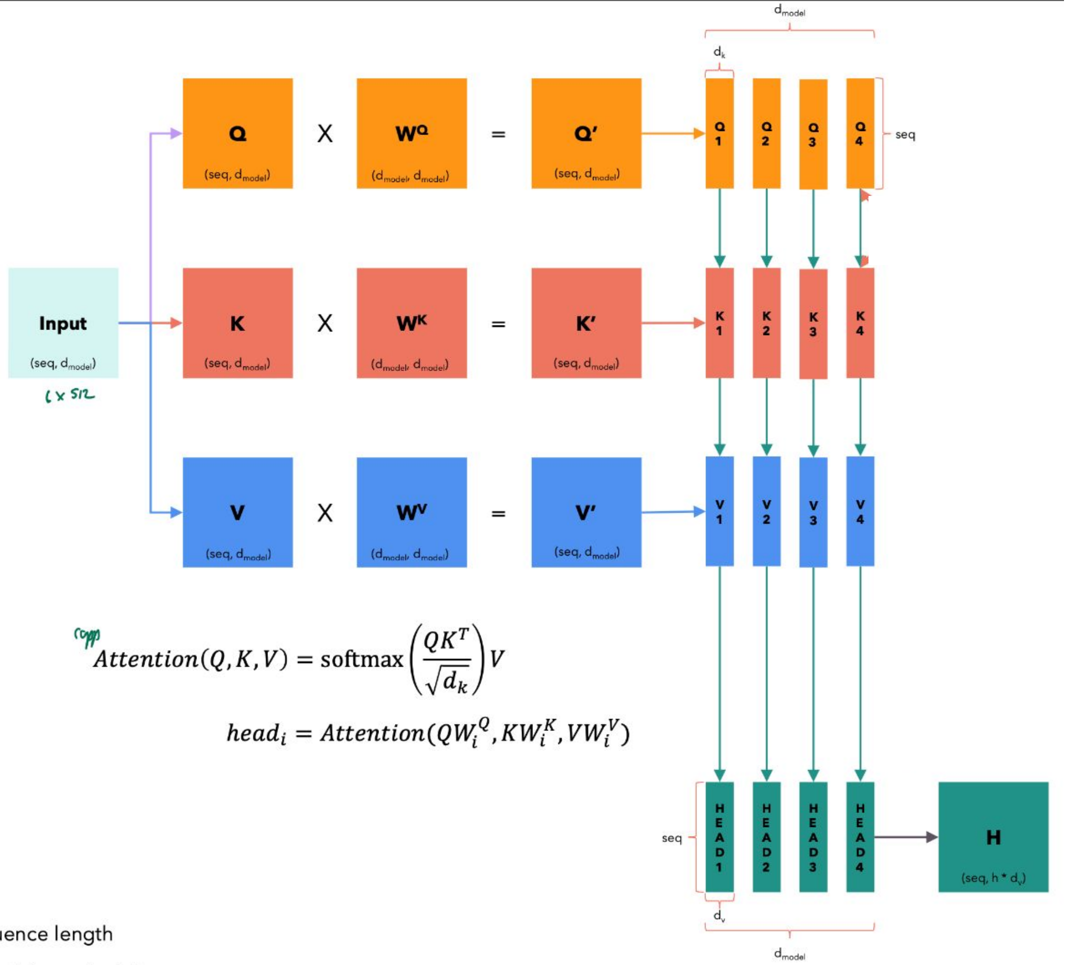



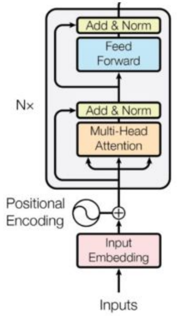

(copy)

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

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

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




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

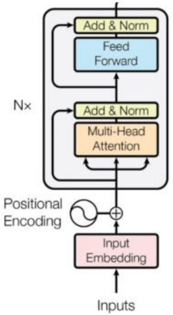
$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$$

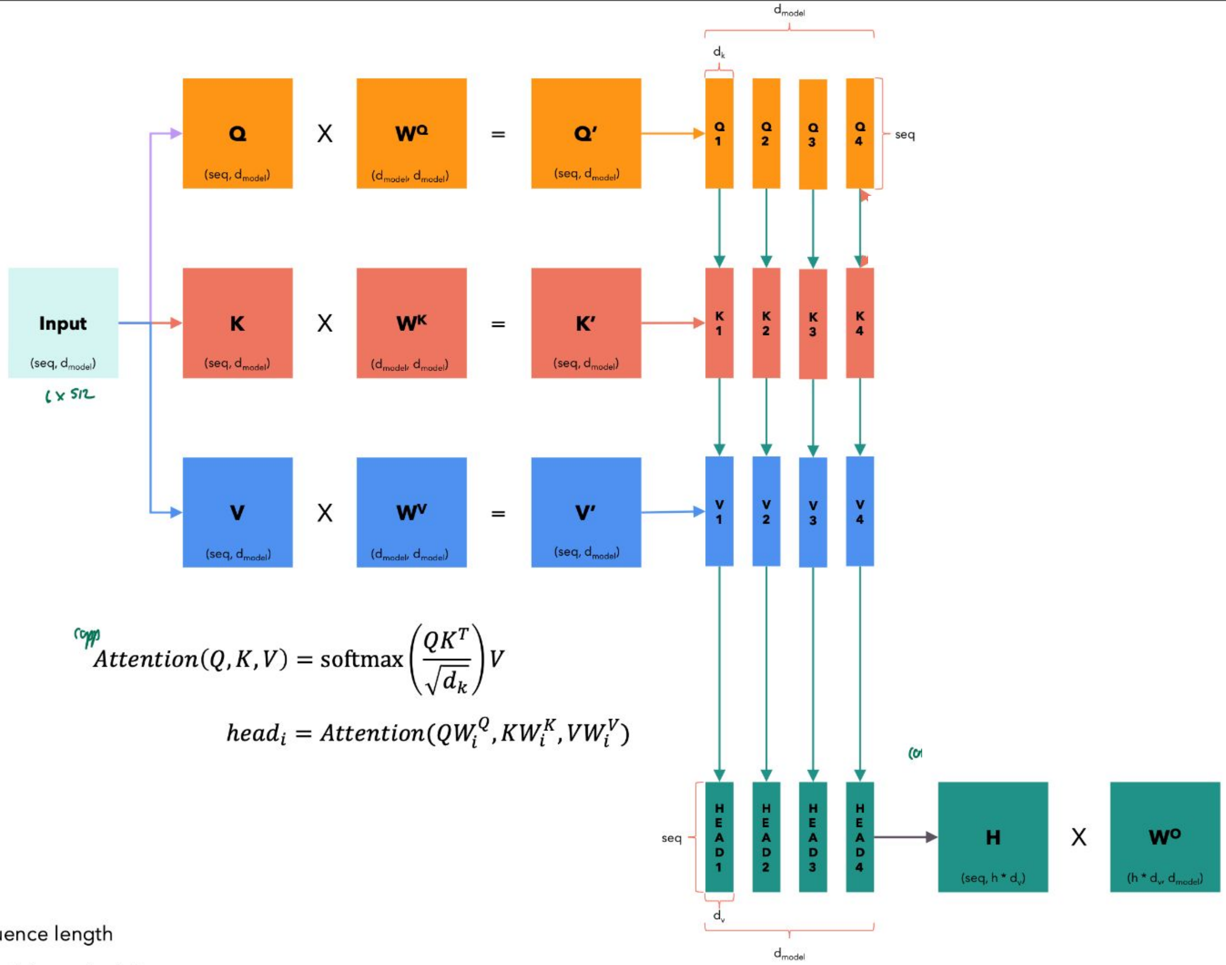
$seq$ = sequence length

$d_{model}$ = size of the embedding vector

h = number of heads

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

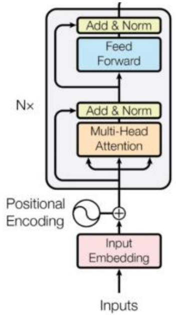


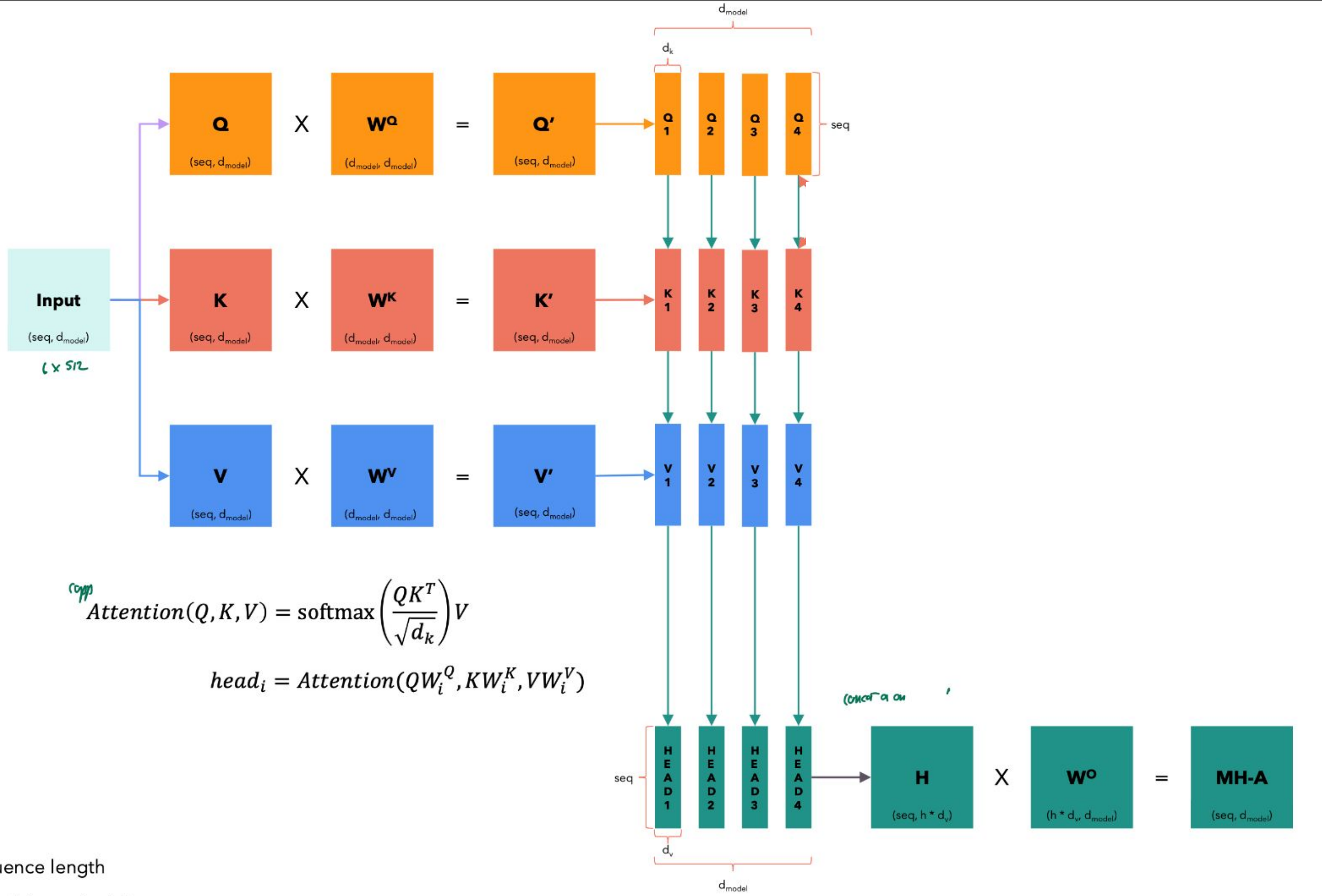




$$Attention(Q,K,V) = \mathrm{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

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

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



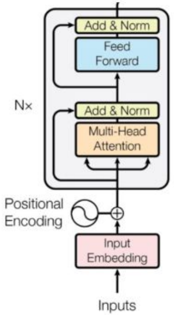



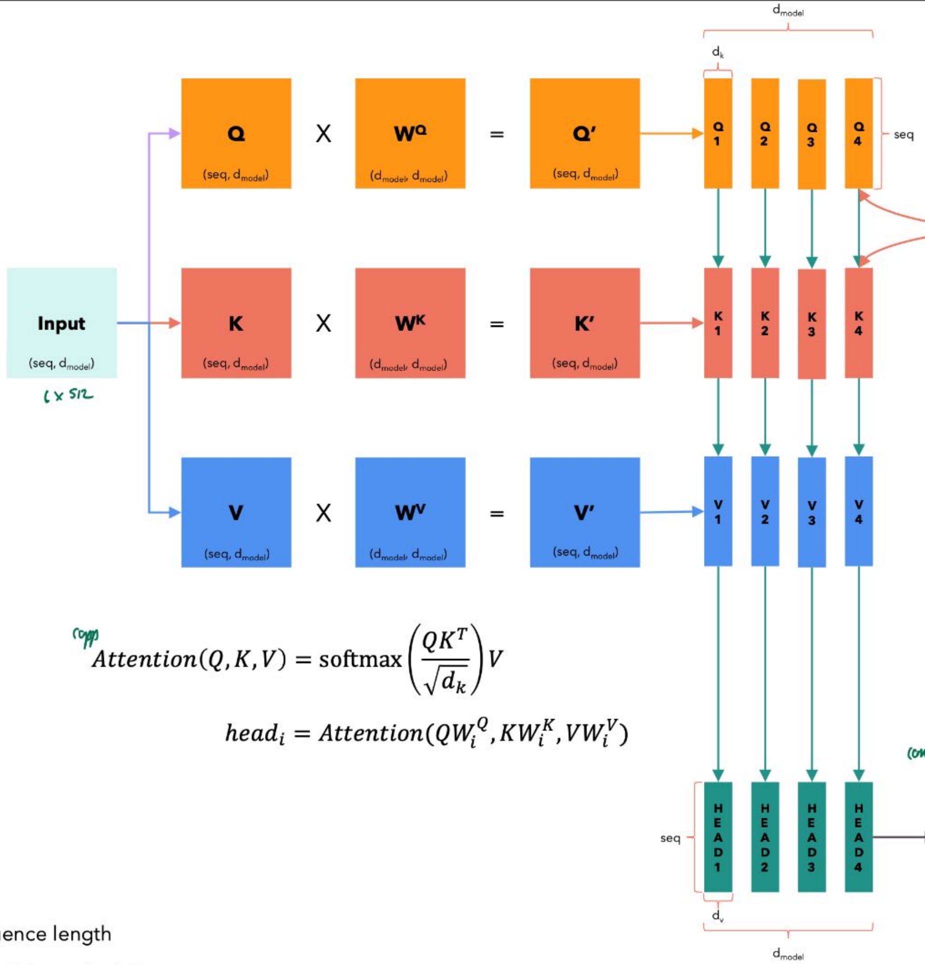

$$Attention(Q,K,V) = \mathrm{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

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

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

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

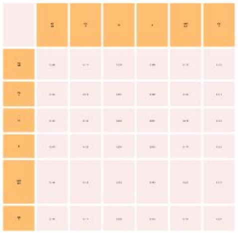


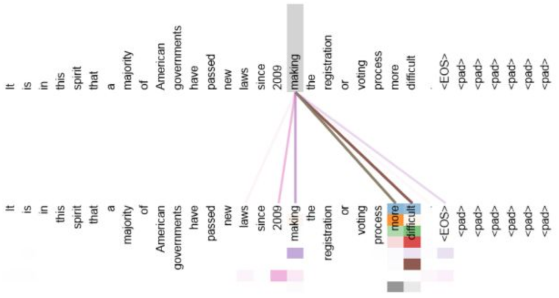

$$Attention(Q,K,V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

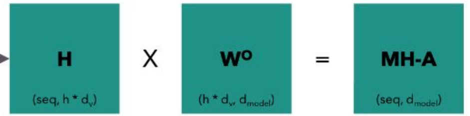
$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$$

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



**Attention Visualizations**

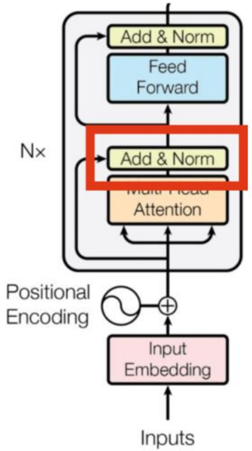



$$MultiHead(Q,K,V) = Concat(head_1 \; \ldots head_h)W^O$$

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

# Encoder


Add & Norm
Feed Forward
Nx
Add & Norm
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 |
| $\mu_1$ | $\mu_2$ | $\mu_3$ |
| $\sigma_1^2$ | $\sigma_2^2$ | $\sigma_3^2$ |

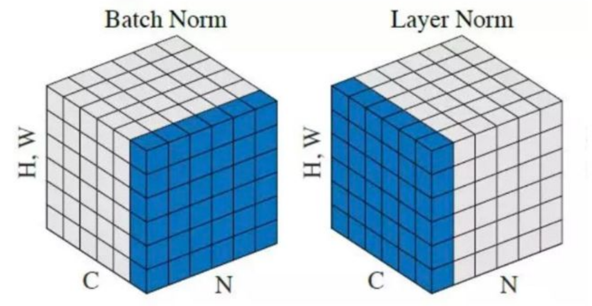
$$\hat{x}_j = \frac{x_j - \mu_j}{\sqrt{\sigma_j^2 + \epsilon}}$$

# What is layer normalization?

Batch of 3 items

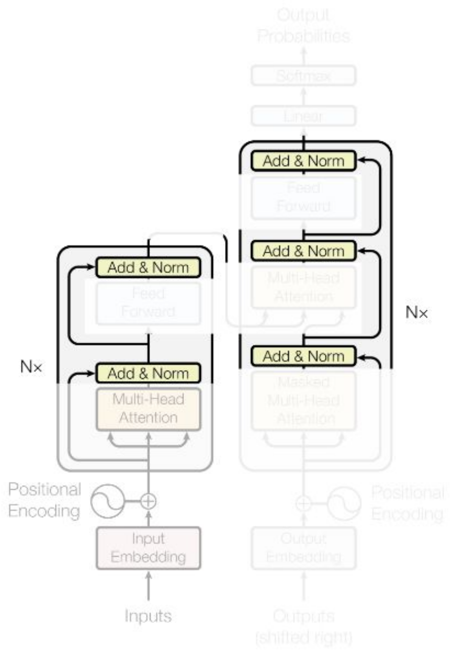
| ITEM 1 | ITEM 2 | ITEM 3 |
| --- | --- | --- |
| 50.147 | 1242.223 | 9.370 |
| 3314.825 | 688.123 | 4606.674 |
| … | … | … |
| … | … | … |
| 8463.361 | 434.944 | 944.705 |
| 8.021 | 149.442 | 21189.444 |
| $\mu_1$ | $\mu_2$ | $\mu_3$ |
| $\sigma_1^2$ | $\sigma_2^2$ | $\sigma_3^2$ |

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



# Attention Is All You Need - The Transformer architecture

**2017**, Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, Illia Polosukhin




## Residual connections

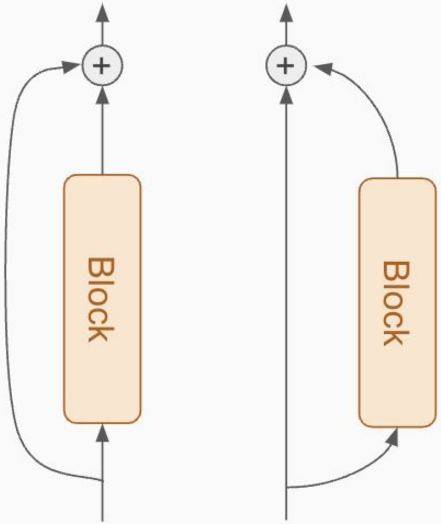
"Skip connection" == "Residual block"

Each module's output has the exact same shape as its input.

Following ResNets, the module computes a "residual" instead of a new value:

$$z_i = \mathrm{Module}(x_i) + x_i$$

This was shown to dramatically improve trainability.




## LayerNorm

Normalization also dramatically improves trainability.

There's **post-norm** (original) and **pre-norm** (modern)

$$z_i = \mathrm{LN}(\mathrm{Module}(x_i) + x_i)$$

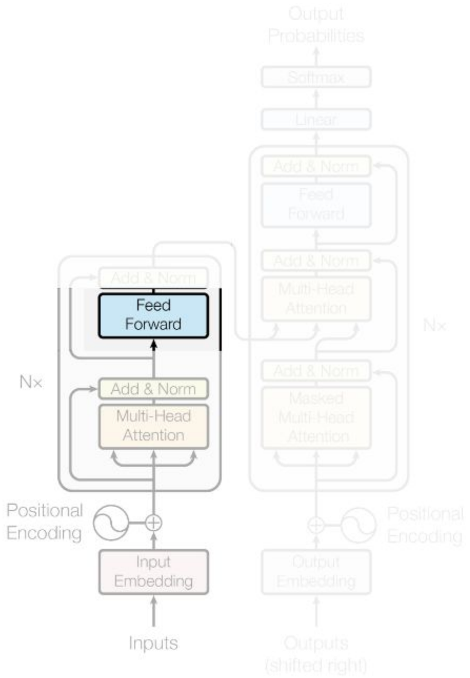
$$z_i = \mathrm{Module}(\mathrm{LN}(x_i)) + x_i$$

Thanks to Basil Mustafa for slide inspiration

Transformer image source: "Attention Is All You Need" paper

# Attention Is All You Need - The Transformer architecture

**2017**, Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, Illia Polosukhin




### Point-wise MLP

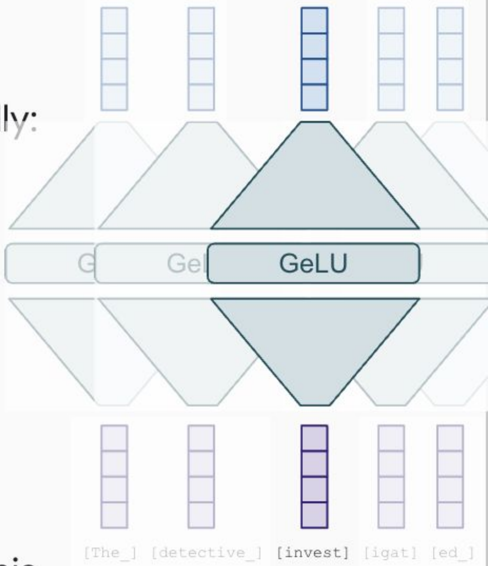
A simple MLP applied to each token individually:

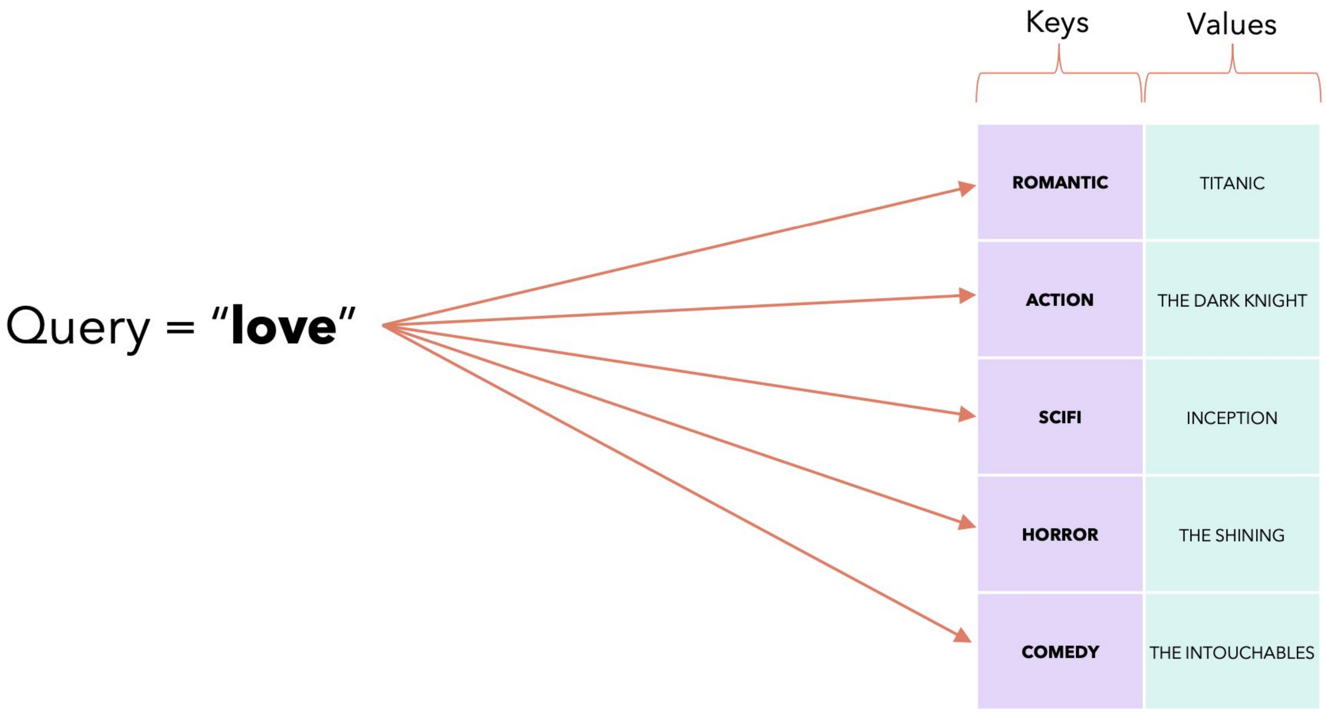
$$z_i = W_2\, \mathrm{GeLU}(W_1 x + b_1) + b_2$$

Think of it as each token pondering for itself about what it has observed previously. There's some weak evidence this is where "world knowledge" is stored, too.

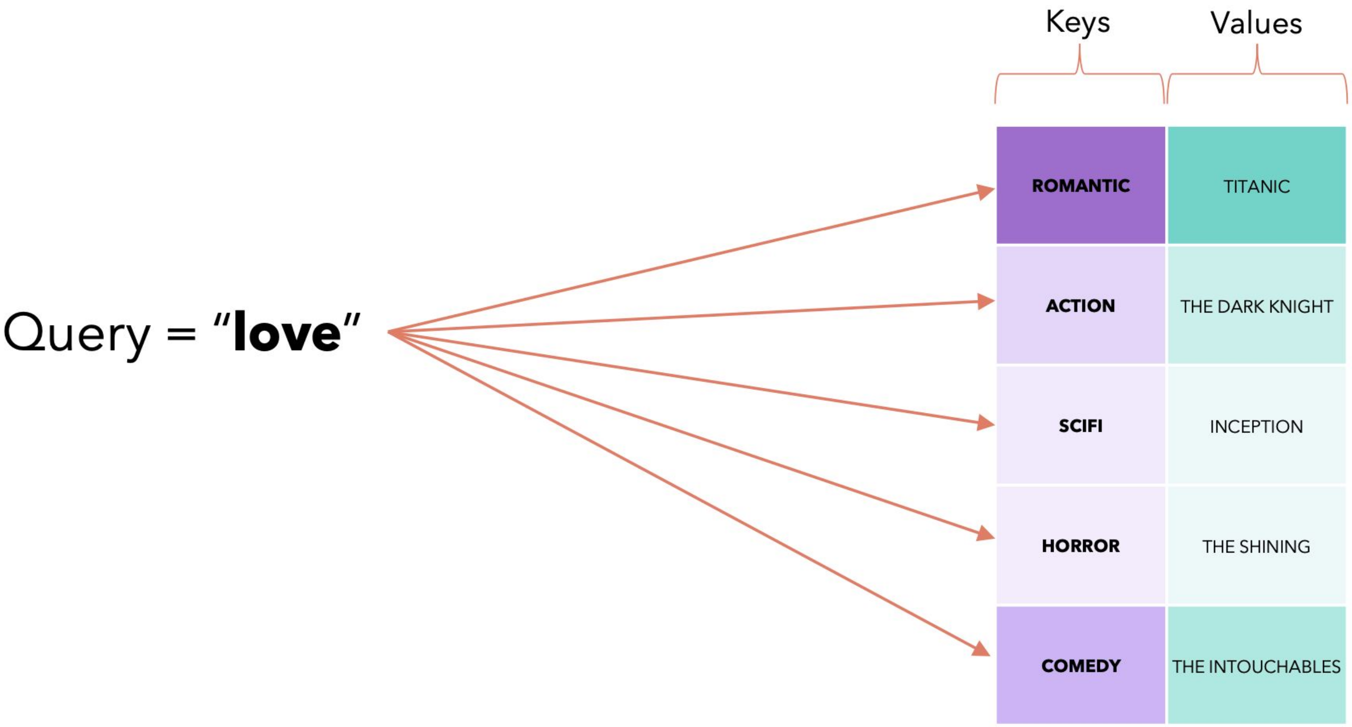
It contains the bulk of the parameters. When people make giant models and sparse/moe, this is what becomes giant.

Some people like to call it 1x1 convolution.



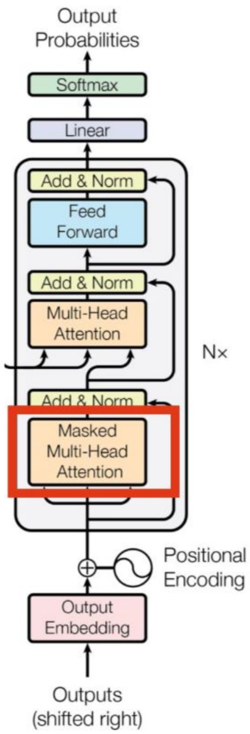



* this could be a Python dictionary
or a database table.



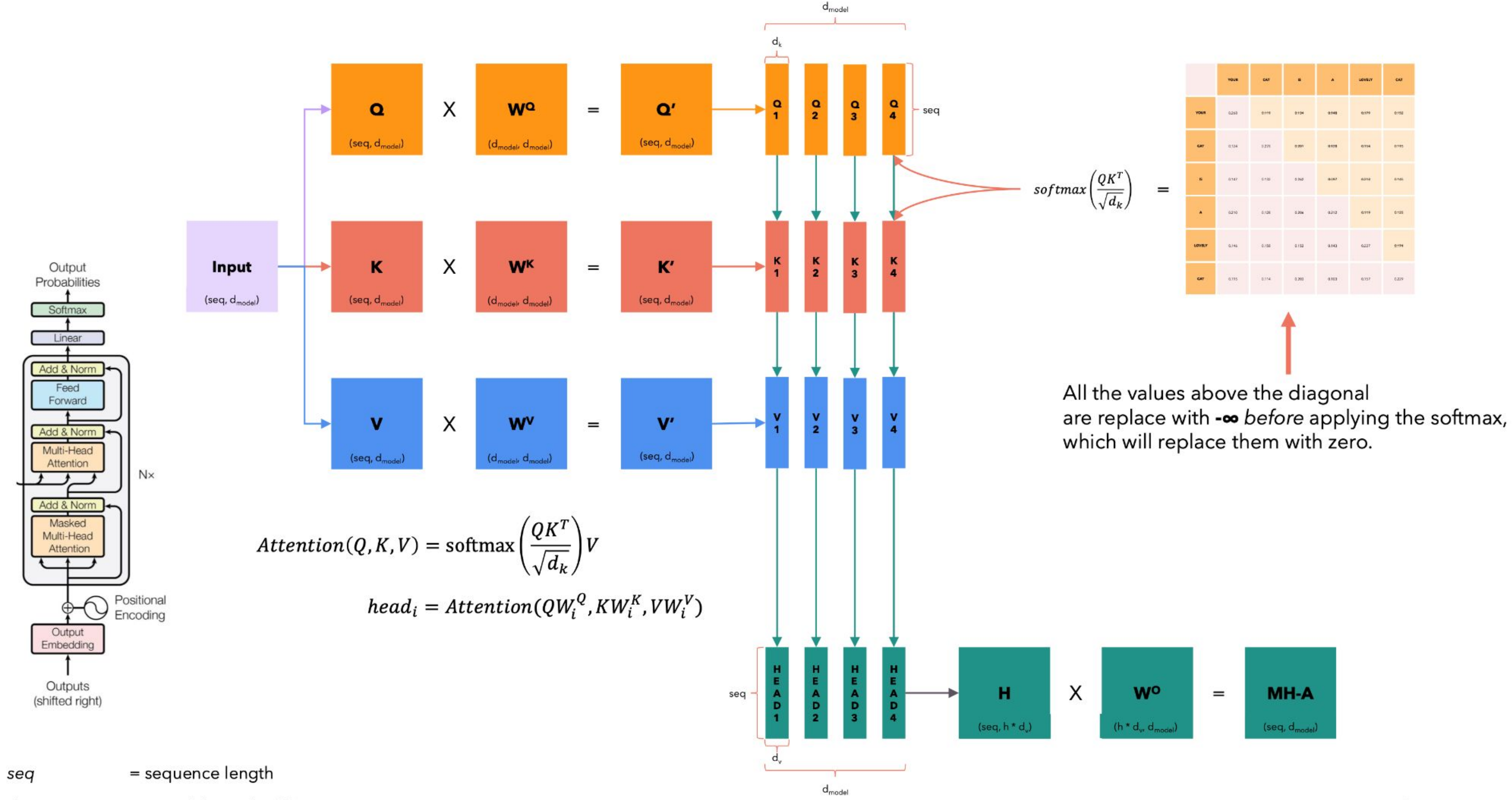

* this could be a Python dictionary
or a database table.

# Decoder


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

# What is Masked Multi-Head Attention?

| | 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 |




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

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

# Training

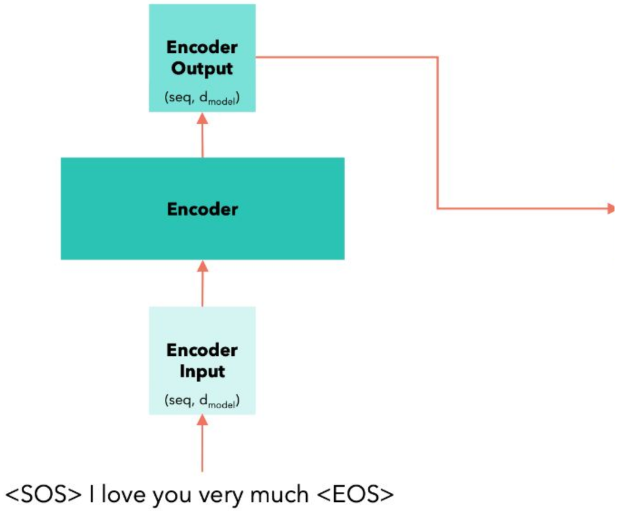
I love you very much

Ti amo molto

# Training

Time Step = 1

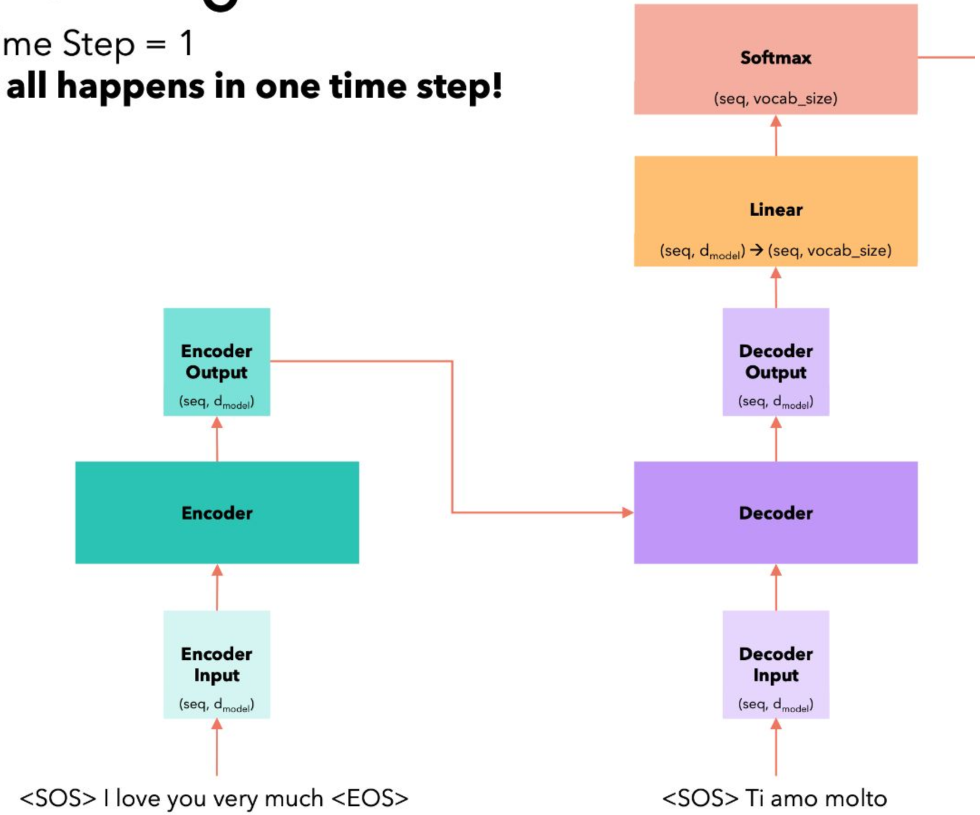
**It all happens in one time step!**



# Training

Time Step = 1

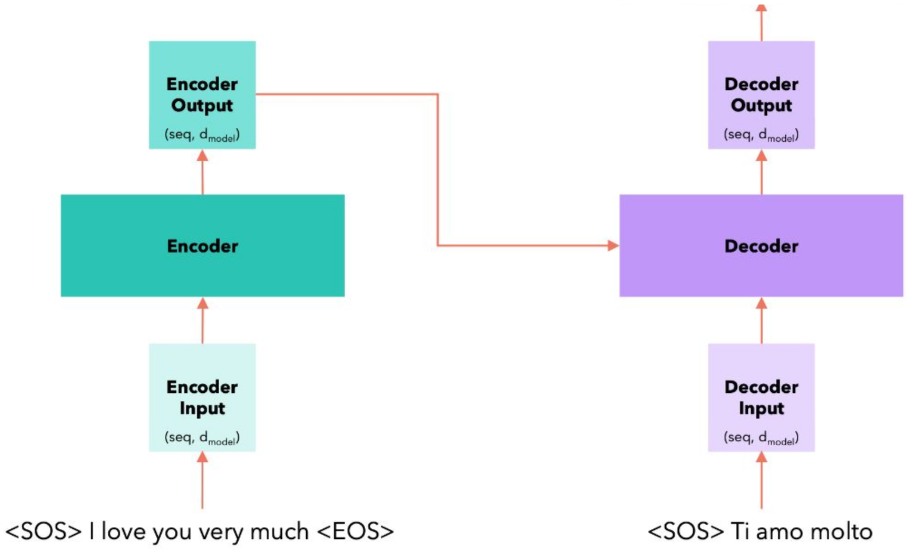
**It all happens in one time step!**

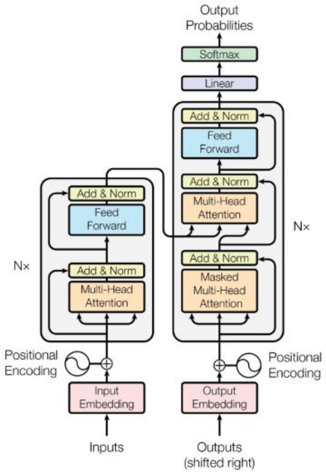


# Training

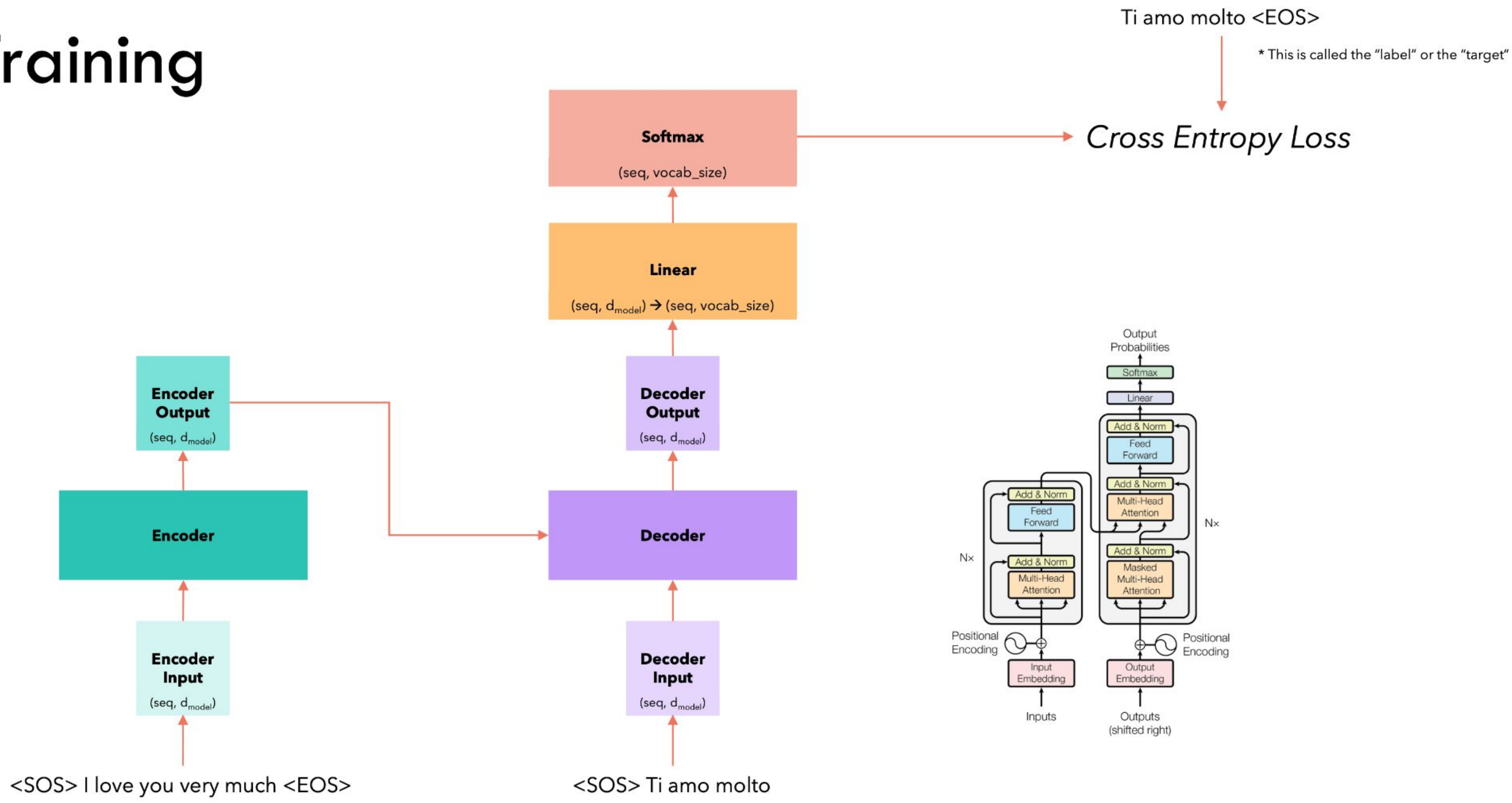
Time Step = 1

**It all happens in one time step!**



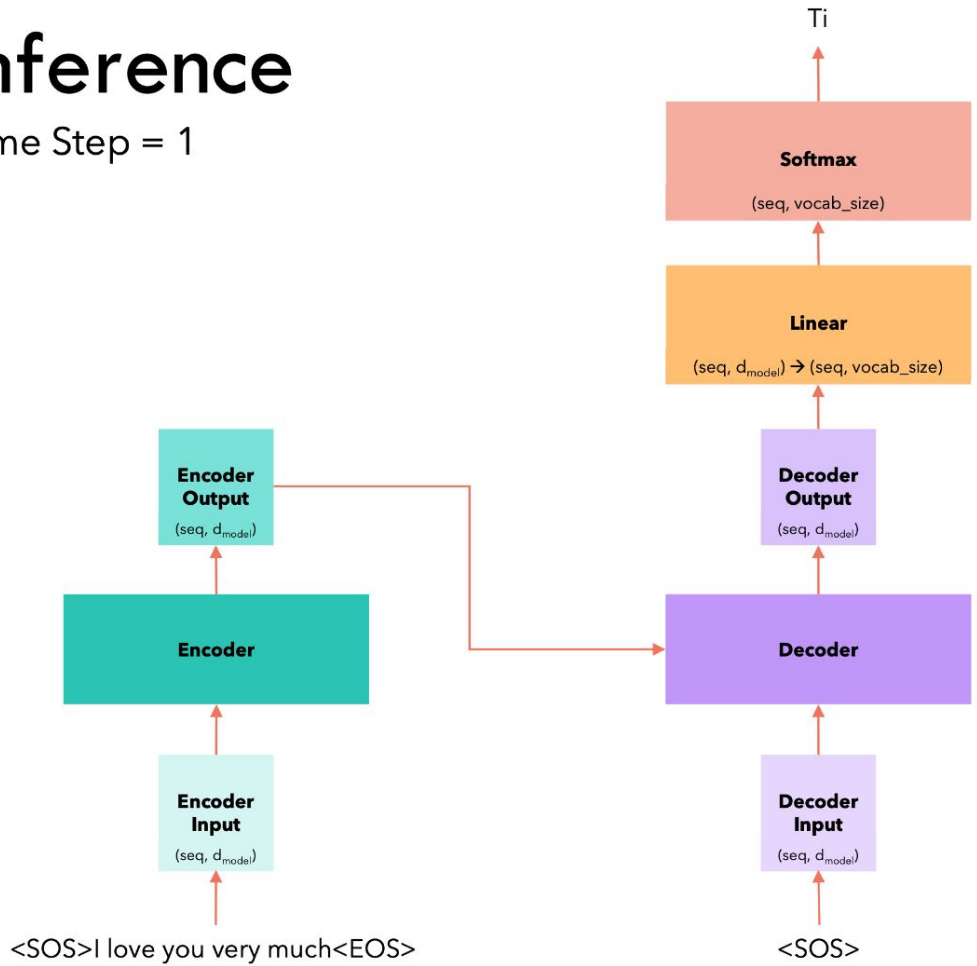



# Training


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, $d_{model}$)
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
Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
N×
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
N×
Positional Encoding
Positional Encoding
Input Embedding
Output Embedding
Inputs
Outputs (shifted right)

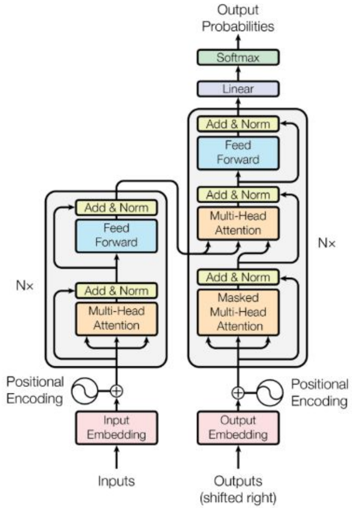
# Inference

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

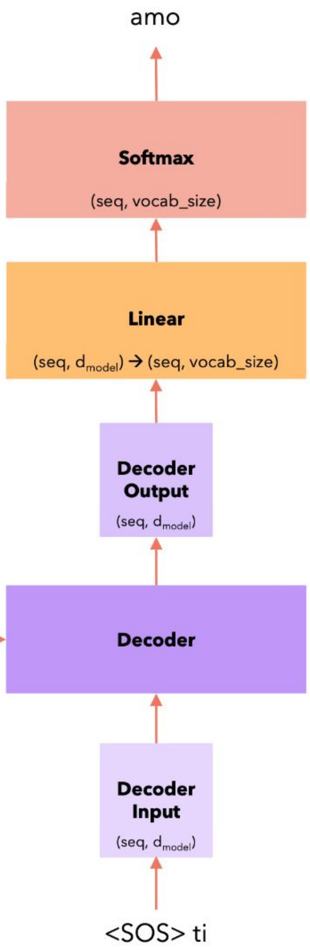
# Inference

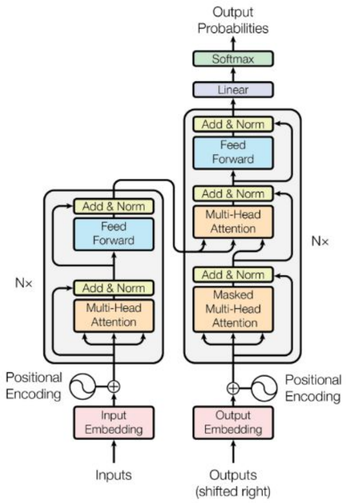
Time Step = 2

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>

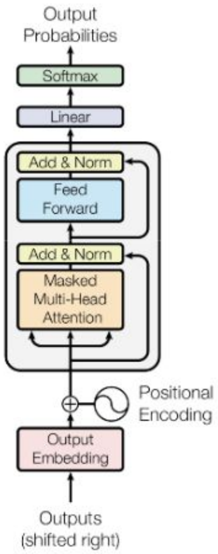



Append the previously output word to the decoder input

# Decoder-only
# GPT

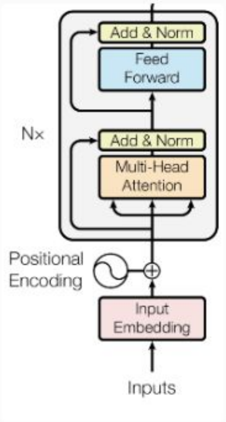
[sat_]




[START] [The_] [cat_]

# Encoder-only
# BERT

[*] [*] **[sat_]** [*] **[the_]** [*]



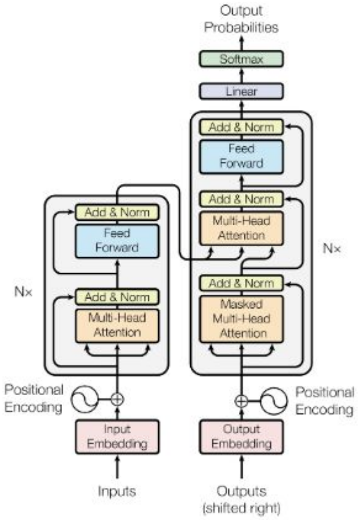

[The_] [cat_] **[MASK]** [on_] **[MASK]** [mat_]

# Enc-Dec
# T5

Das ist gut.

A storm in Attala caused 6 victims.

This is not toxic.




Translate EN-DE: This is good.

Summarize: state authorities dispatched…

Is this toxic: You look beautiful today!