

# TRANSFORMER

Slides by Vaswani, Ashish, et al. "Attention is all you need." *Advances in neural information processing systems*. 2017.

# OUTLINES



- INTRODUCTION
- MODEL ARCHITECTURE

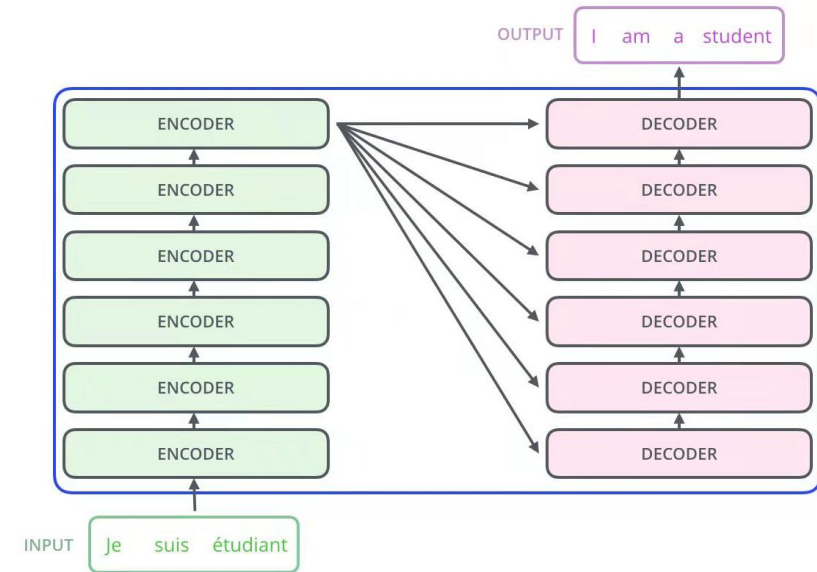
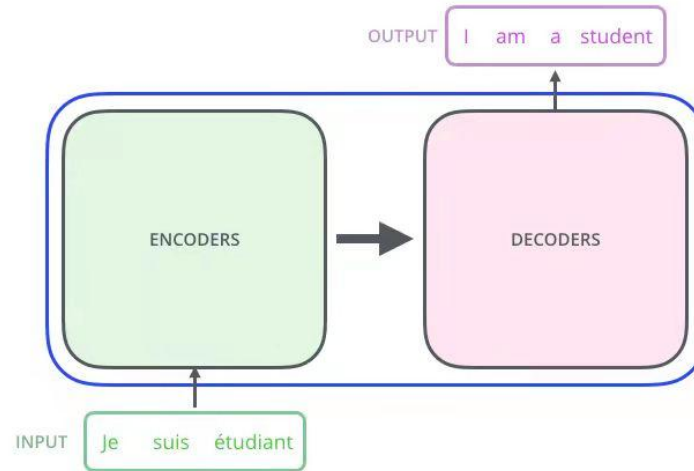
# INTRODUCTION



**The Transformer** – a model that uses attention to boost the speed with which these models can be trained. It was proposed in the paper [Attention is All You Need](#).

- The Transformers outperforms the Google Neural Machine Translation model in specific tasks. The biggest benefit is how The Transformer lends itself to parallelization.
- It is in fact Google Cloud's recommendation to use The Transformer as a reference model to use their [Cloud TPU](#) offering.

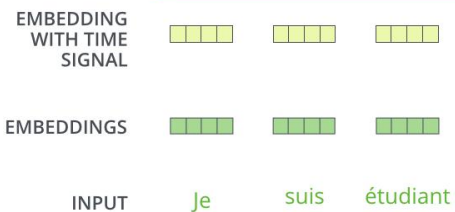
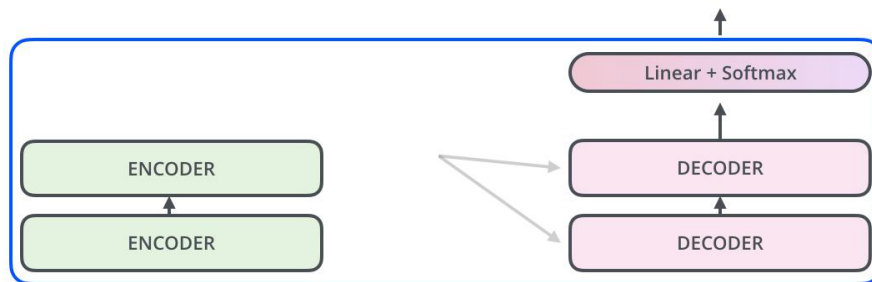
# MODEL ARCHITECTURE



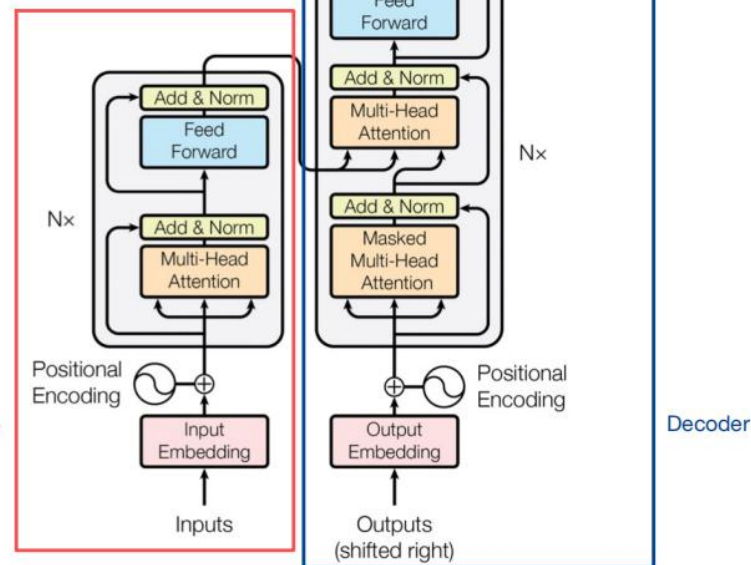


Decoding time step: 1 2 3 4 5 6

OUTPUT

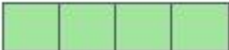


Encoder

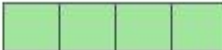


# Embedding Algorithm

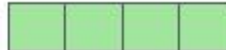
- The embedding only happens in the bottom-most encoder.
- The abstraction that is common to all the encoders is that they receive a list of vectors each of the size 512.
- The size of this list is hyperparameter we can set – basically it would be the length of the longest sentence in our training dataset.

$x_1$  

Je

$x_2$  

suis

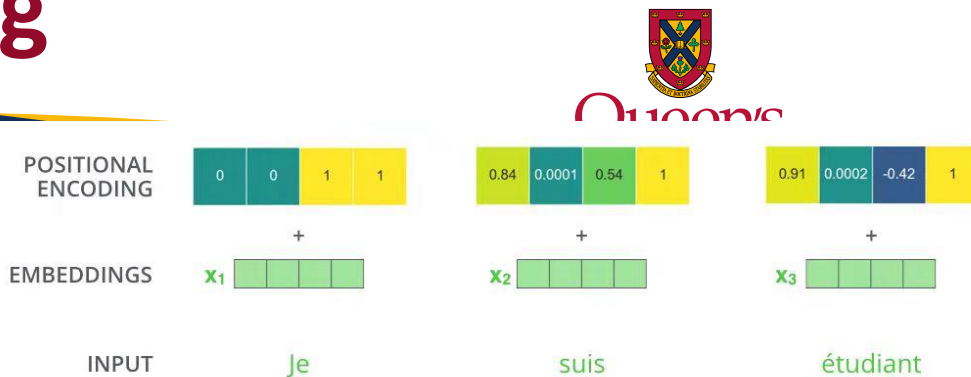
$x_3$  

étudiant

# Positional Encoding

Since our model contains no recurrence and no convolution, in order for the model to make use of the order of the sequence, we must inject some information about the relative or absolute position of the tokens in the sequence.

To this end, we add "positional encodings" to the input embeddings at the bottoms of the encoder and decoder stacks.



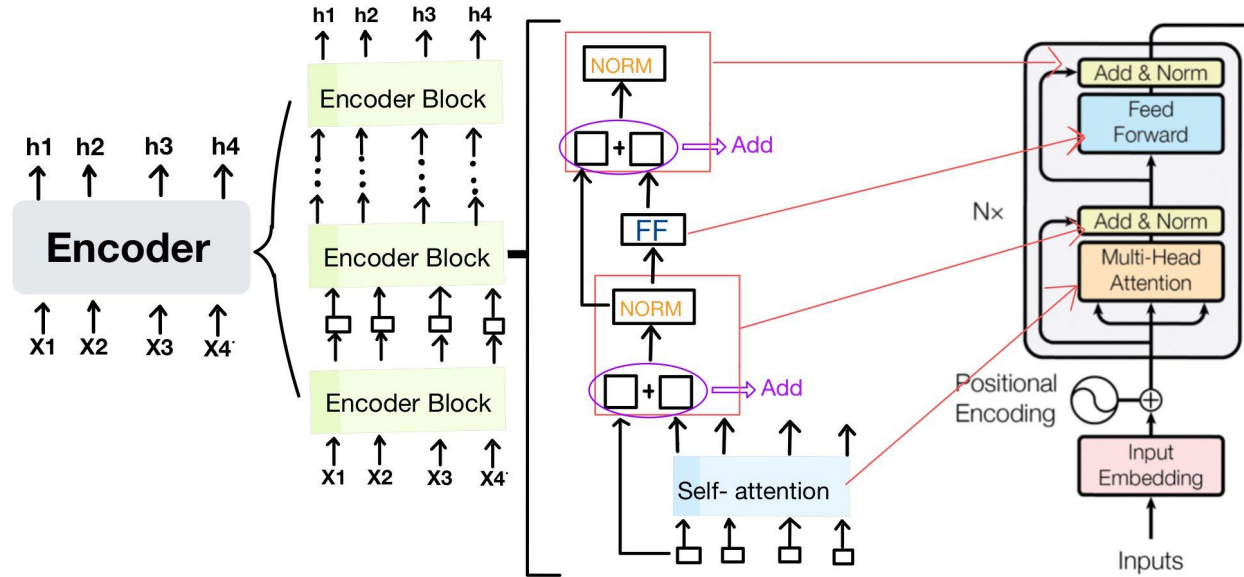
$$PE_{(pos, 2i)} = \sin(pos/10000^{2i/d_{\text{model}}})$$

$$PE_{(pos, 2i+1)} = \cos(pos/10000^{2i/d_{\text{model}}})$$

pos is the position and i is the dimension

# Encoder

- The encoder is composed of a stack of  $N = 6$  identical layers.
- Each layer has two sub-layers. The first is a multi-head self-attention mechanism, and the second is a simple, positionwise fully connected feed-forward network.
- The output of each sub-layer is  $\text{LayerNorm}(x + \text{Sublayer}(x))$ , where  $\text{Sublayer}(x)$  is the function implemented by the sub-layer itself.

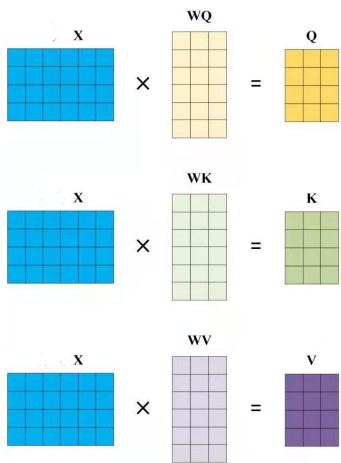




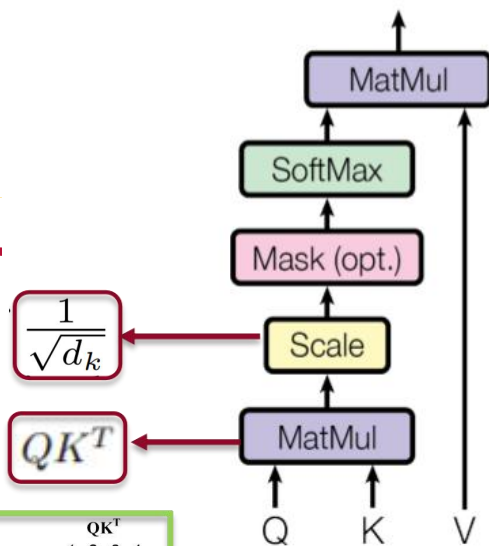
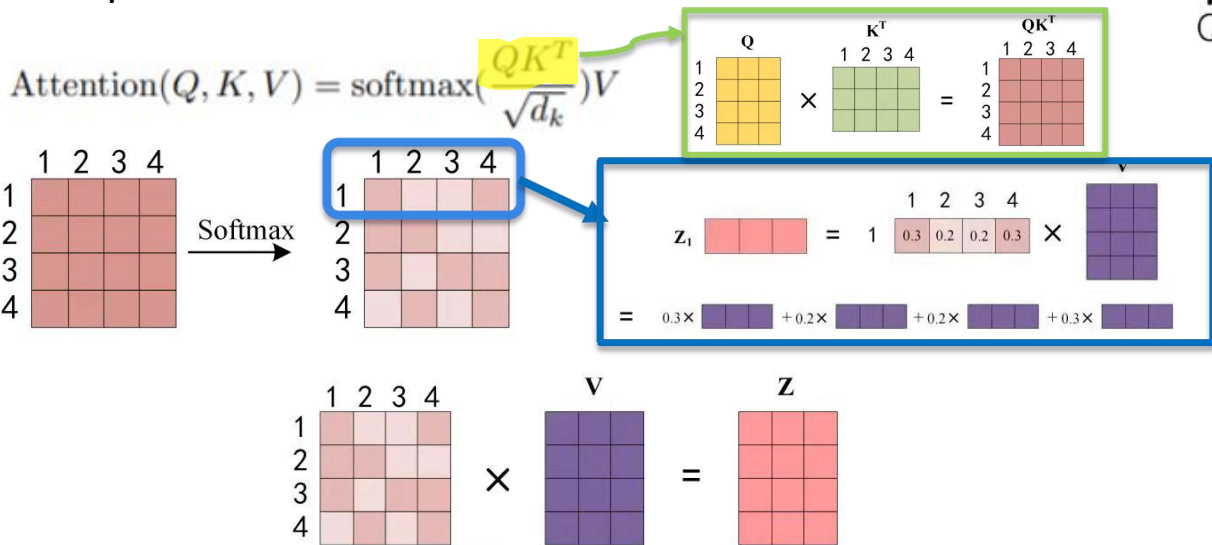
# Self- Attention

- We call our particular attention "Scaled Dot-Product Attention"
- The input consists of queries and keys of dimension  $d_k$ , and values of dimension  $d_v$ .

Step 1:



Step 2:



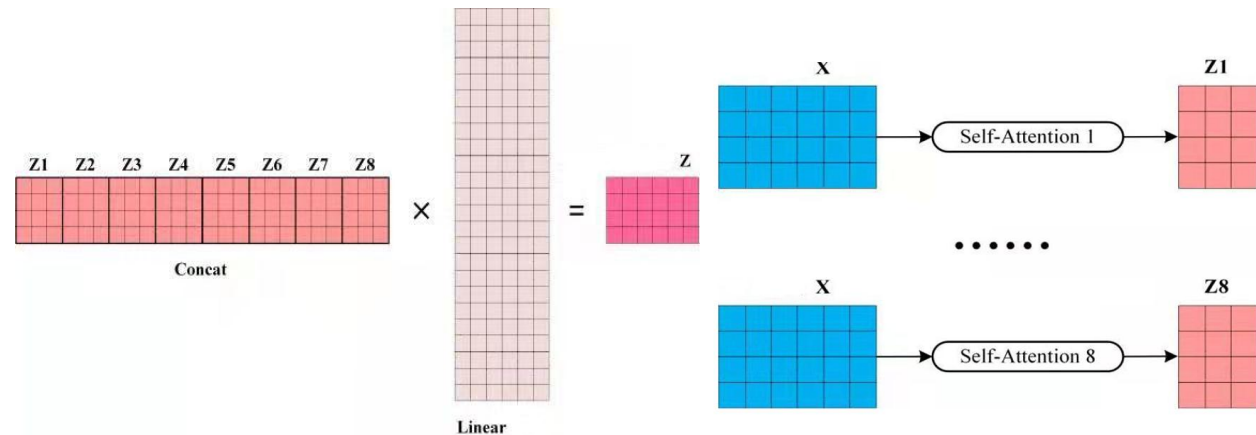
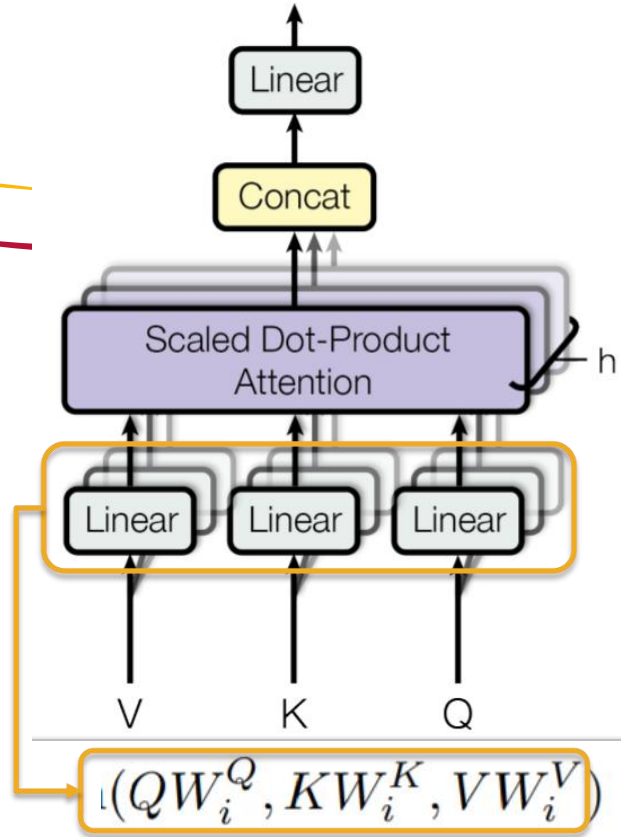
# Multi-Head Attention

Step 1: input V, K, Q and Linear

Step 2:  $\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$

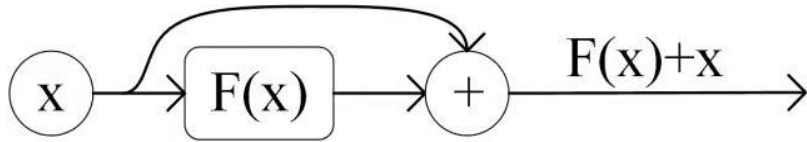
Step 3:  $\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_h)W^O$

Where the projections are parameter matrices  $W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}$ ,  $W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$ ,  $W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$  and  $W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$ .



# ADD & NORM

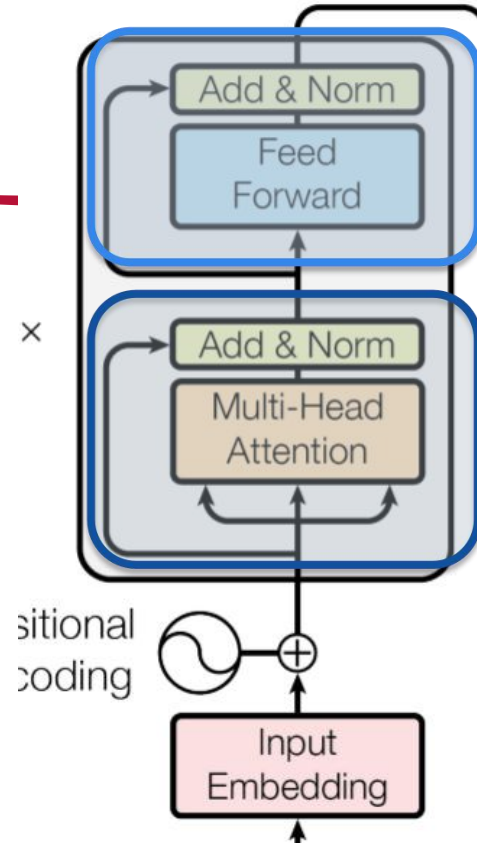
ADD:



NORM:

$\text{LayerNorm}(X + \text{MultiHeadAttention}(X))$

$\text{LayerNorm}(X + \text{FeedForward}(X))$



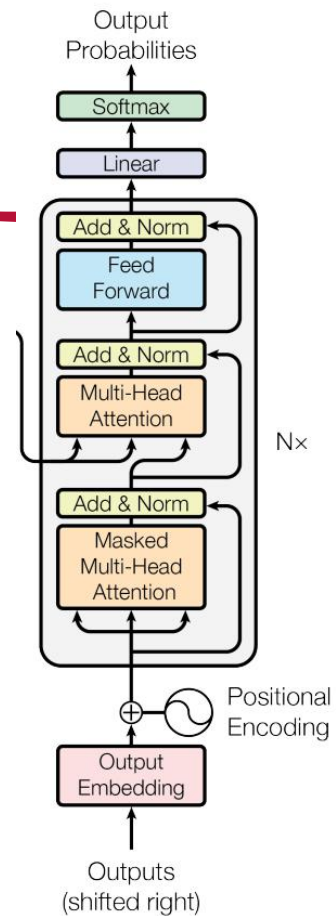
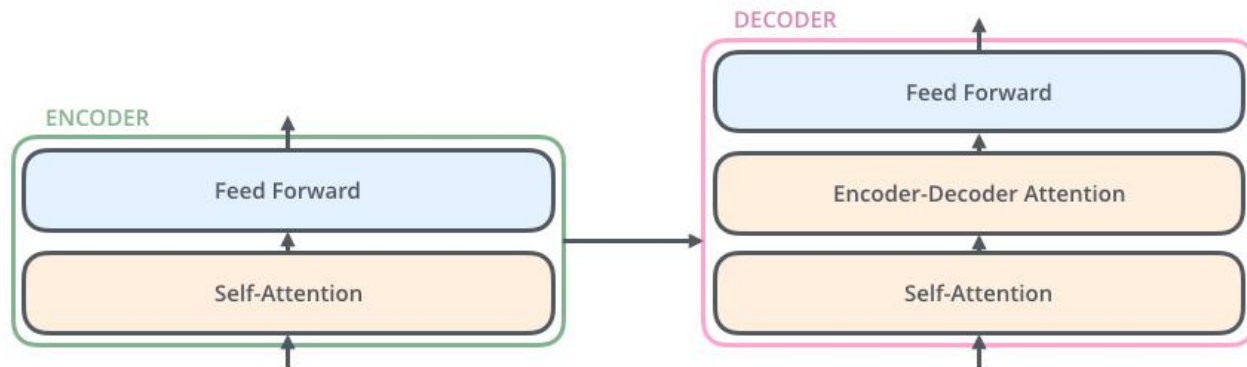
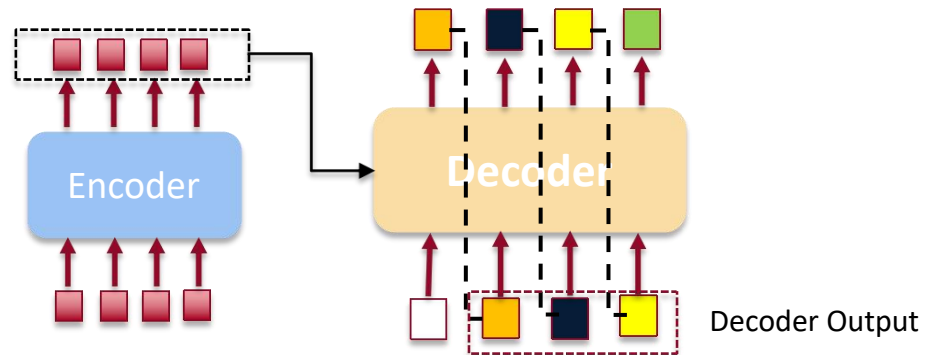
# Fully connected Feed-Forward network



$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

Each of the layers in our encoder and decoder contains a fully connected feed-forward network, which is applied to each position separately and identically. This consists of two linear transformations with a ReLU activation in between.

# Decoder

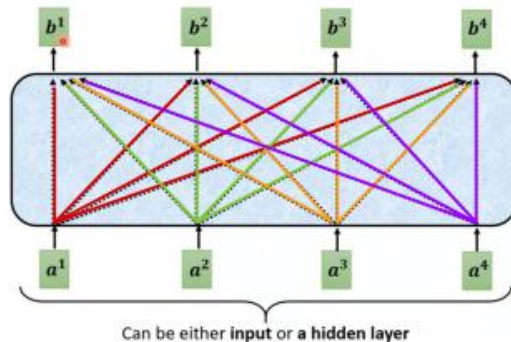


# Mask Multi-Head Attention

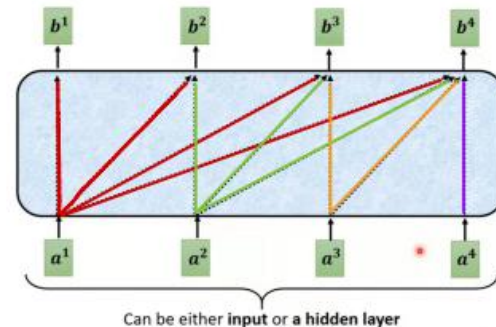
The reason for using Masked is the process of translation is in sequence.

Which means the  $i+1$ th vocabulary will be translated after the  $n$ th vocabulary translated.

Self-attention

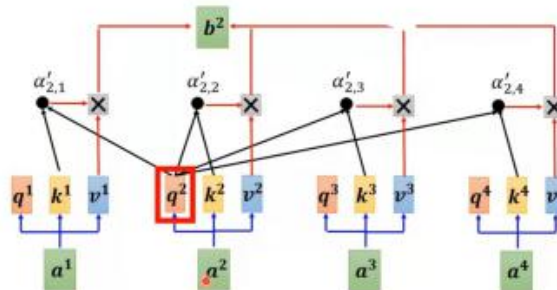


Self-attention → Masked Self-attention

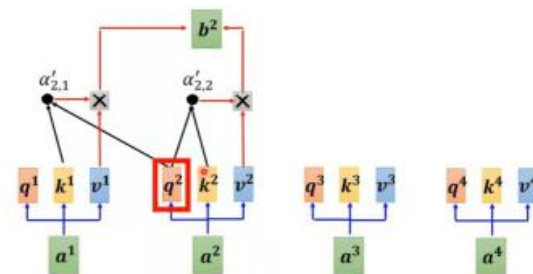


VS.

Self-attention



Self-attention → Masked Self-attention



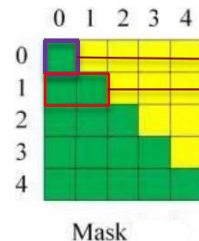
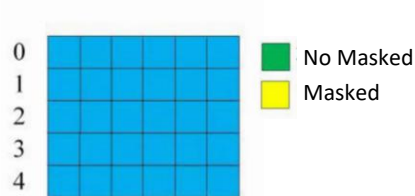
VS.



## Step1: Get Input matrix and Mask matrix

I Have A Pretty Cat

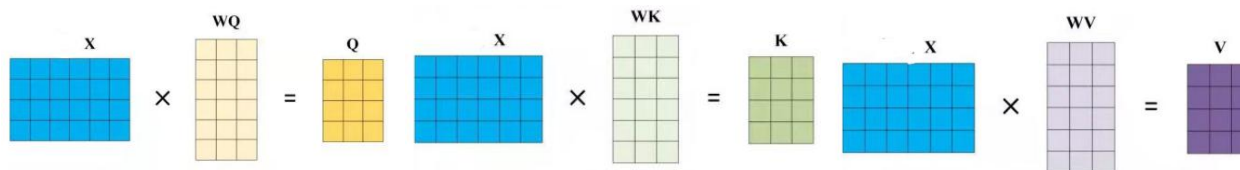
0 1 2 3 4



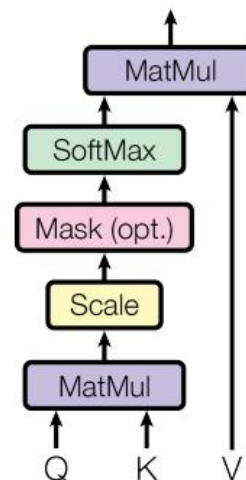
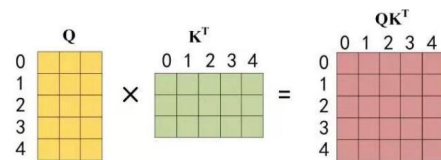
Vocabulary 0 can just use the information for itself.

Vocabulary 1 can use information from 1 and 0.

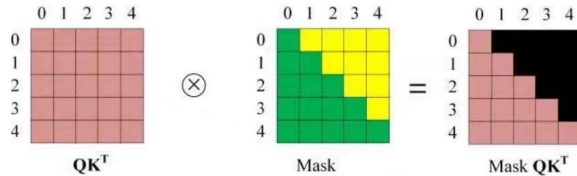
## Step2: Get Q, K, V



## Step3: MatMul

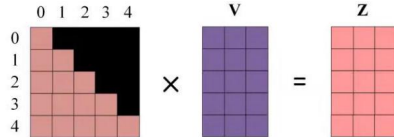


### Step4: Use Mask Matrix to Mask $QK^T$



### Step5: SoftMax

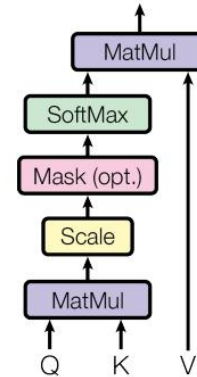
### Step6: Get $(\text{Mask } QK^T)V$



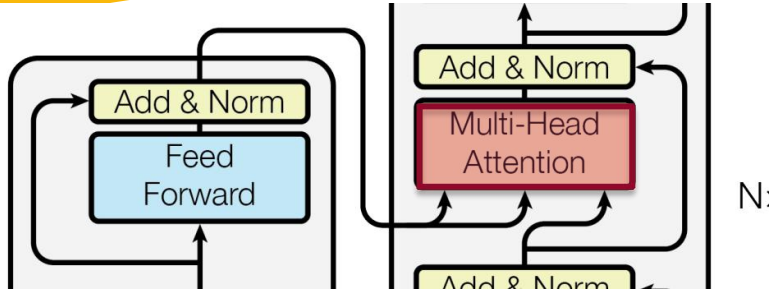
The output vector  $Z$  for Vocabulary 1 will just include information from vocabulary 1

### Step7: Same step with Multi-attention, concat, Linear

### Scaled Dot-Product Attention

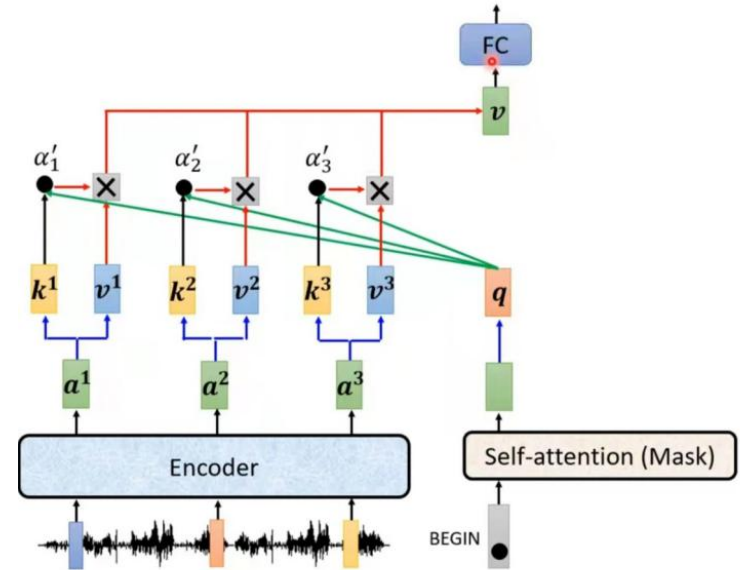




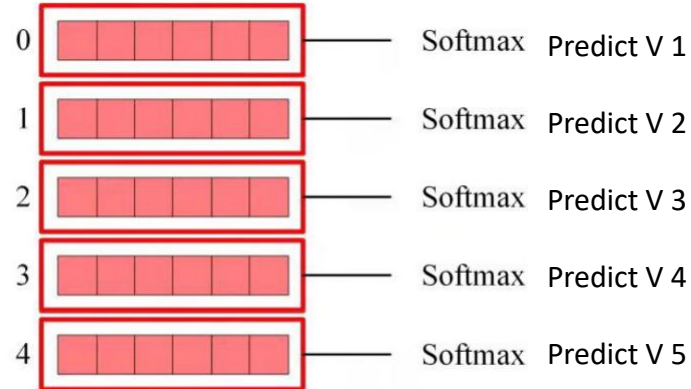
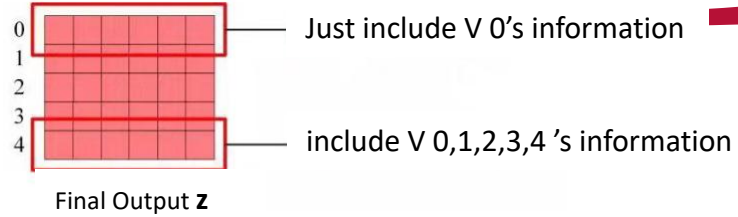
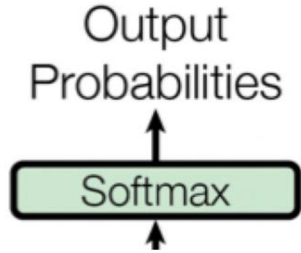


The difference between self-attention in Decoder and self-attention in encoder is: Self-attention in Decoder will get K and V from encoder.

The reason is every vocabulary will use all vocabulary's information from Encoder



# Softmax



## Reference

- [1] Vaswani A, Shazeer N, Parmar N, et al. Attention is all you need [C]//Advances in Neural Information Processing Systems. 2017: 5998-6008.
- [2] Bahdanau D, Cho K, Bengio Y. Neural machine translation by jointly learning to align and translate[J]. arXiv preprint arXiv:1409.0473, 2014.
- [3] Devlin J, Chang M W, Lee K, et al. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding[J]. arXiv preprint arXiv:1810.04805, 2018.
- [4] <http://jalammar.github.io/illustrated-transformer>
- [5] He K, Zhang X, Ren S, et al. Deep residual learning for image recognition[C]//Proceedings of the IEEE conference on computer vision and pattern recognition.