# Transformers

Encoder maps an input sequence of symbol representations $(x_1 \ldots\ldots x_n)$ to a sequence of continuous representations $z = (z_1 \ldots\ldots z_n)$

Decoder: Given Z the decoder then generates an output sequence $(y_1, \ldots\ldots y_n)$ of symbols one at a time.

the model is auto-regressive and consumes the previously generated symbols as additional input.

Encoder:

Six identical layers.

Each layer has two sub layers.

multi-head self-attention mechanism and position-wise fully connected layer.

- residual connection
- Normalization around each layer.

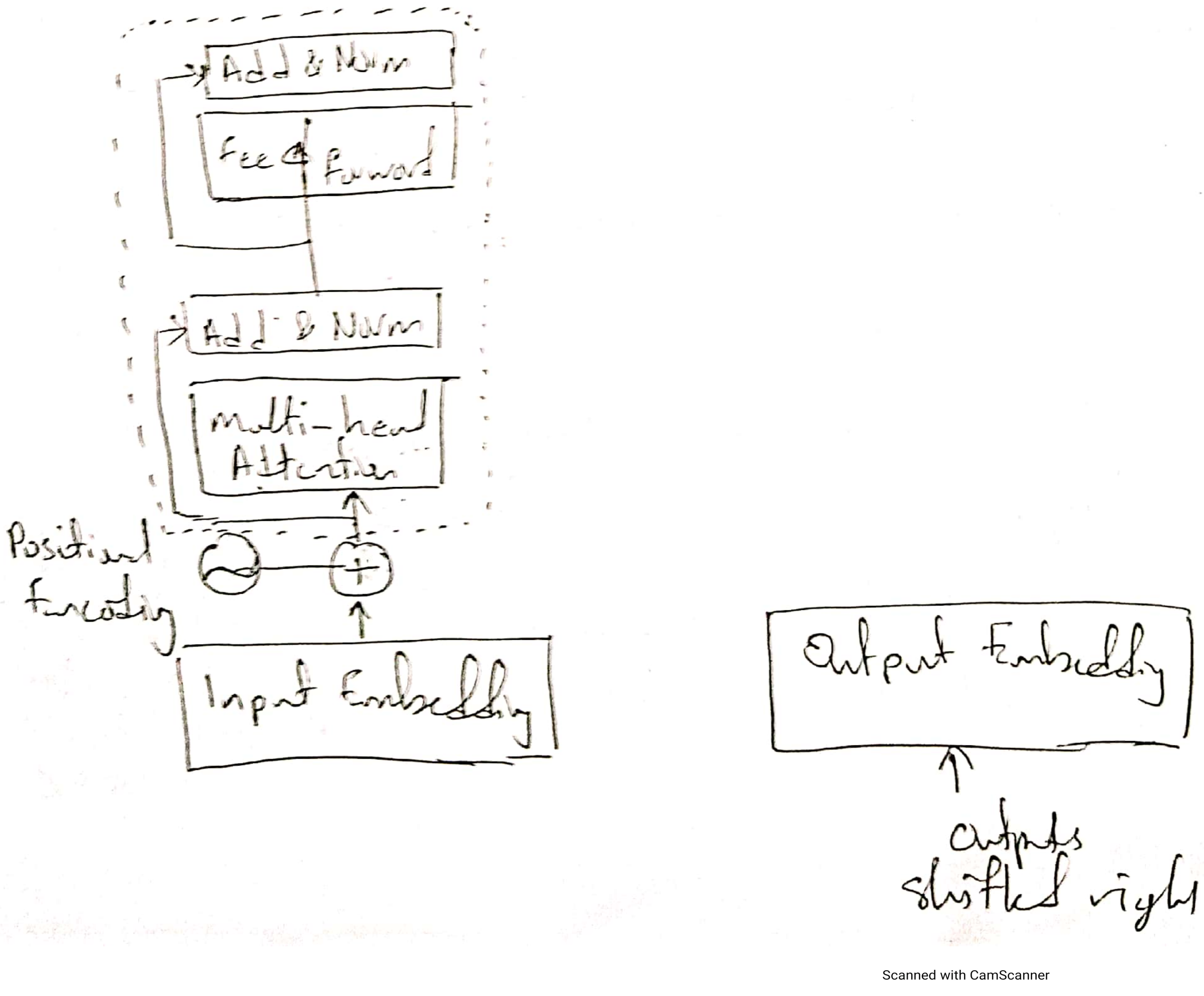
in each ~~two~~ sub-layer of the ENCODER

Decoder: N = 6 identical layers.

$\boxed{\text{two sub-layer from encoder}}$ a third sub-layer performs multi-head attention ~~performs~~ over the output of the encoder stack.

- residual connection
- Normalization

} each sublayer



Attention:
A function which maps a query and a set of key-value pairs to an output where the query, keys, values and output are all vectors.

The o/p is computed as weighted sum of the values
weight is computed by a compatibility function of query with the key.

Scaled dot Product Attention.
input is queries and keys of dimension $d_k$ and values of dimension $d_v$
dot product of query with all keys, divide each by $\sqrt{d_k}$ and apply a softmax function to obtain value

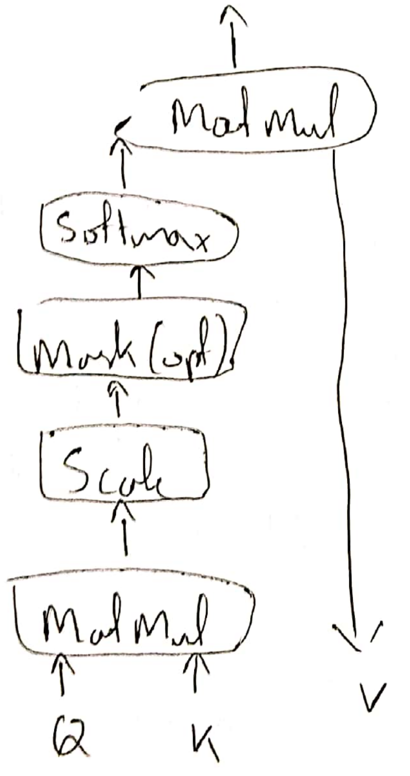
Query → vector representation of one word in the sequence
K → keys vector representation of all the words in the sequence
For multi-head V consists of the same word sequence than Q.
For other V is different

Attention is computed on a set of queries simultaneously packed together into a matrix. Q

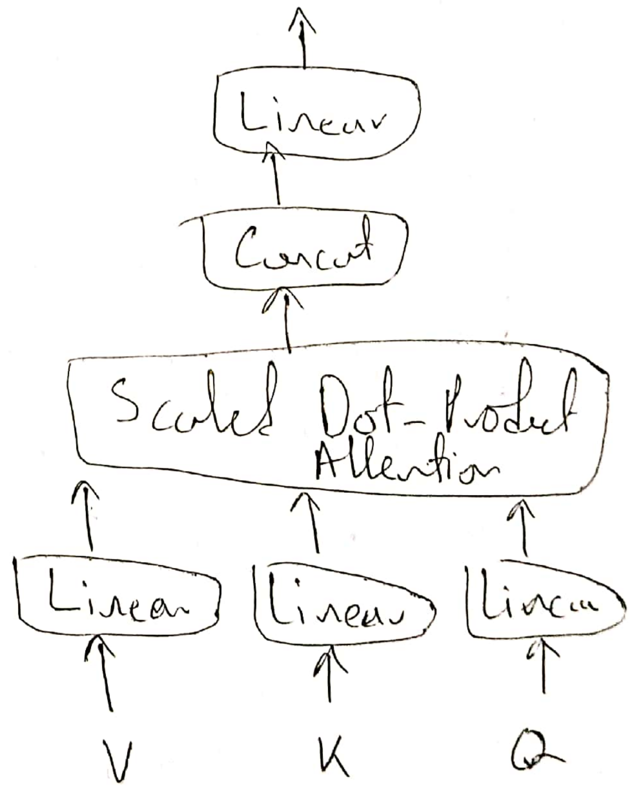
K and V are also matrices.

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

- additive attention
- dot-product (multiplicative attention)




Scaled dot product Attention




Multi-head Attention

Multi-head Attention

. linearly project queries keys and values h times with different learned linear projections.

- ~~Fully Connected~~ . Feed Forward Network

$$FFN(x) = max(0, xW_1 + b_1)W_2 + b_2$$

linear transformations are same across different positions.

- Embeddings

- Positional Encoding

sine and cosine

<u>Self Attention:</u>

— total computational complexity per layer.

— Amount of computation that can be parallized.

— Path length for long range dependencies.