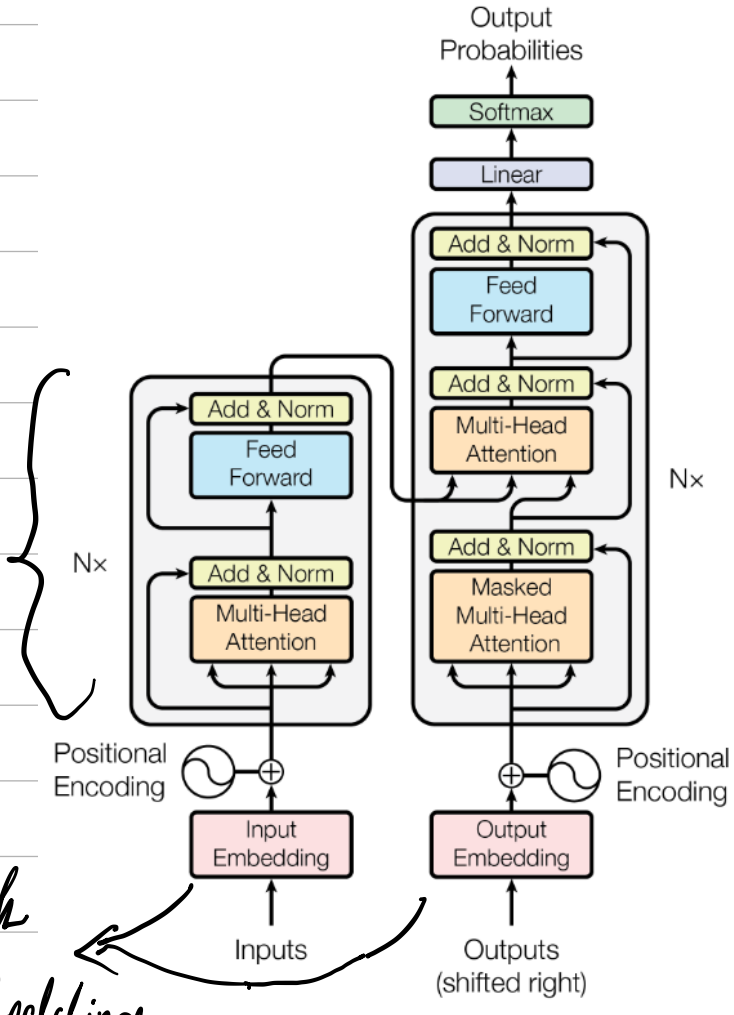
Transformer - Attention is all you need

Model architecture

↳ encoder - decoder
↳ encoder, maps input $(x_1, \dots x_n)$ to $z = (z_1 \dots z_n)$
↳ decoder maps z to $(y_1, \dots y_n)$
↳ auto-regressive, meaning it consumes previously generated symbols as additional input.



encoder block

decoder block

in practice, only q and k gets pos. embedding

Encoder

- They use 6 blocks
- Each block has 2 sub-layers
  - multi-head self-attention
  - position-wise fully connected feed forward network
- residual connections, as in the image above
- Uses layer norm
- $d_{model} = 512$

Decoder

- same way, using 6 blocks
- blocks are similar, but has an additional sub-layer
  - ⇓
  - multi-head attention over the output of the encoder stack

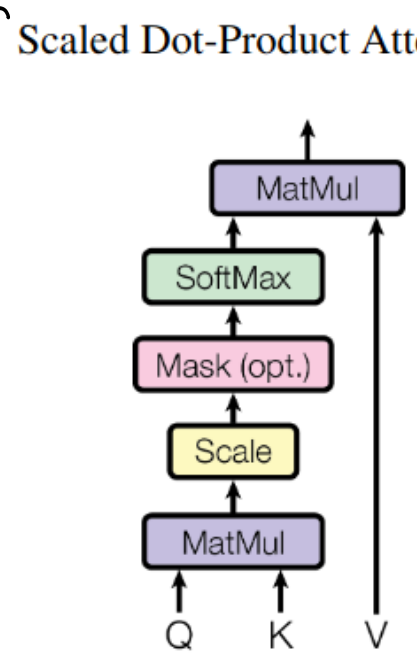
↳ also, the self-attention is modified (masked) to prevent from looking at future positions

Attention

↳ Scaled dot product attention

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

Scaling
⇓
to counter softmax creating small gradients when the values are large

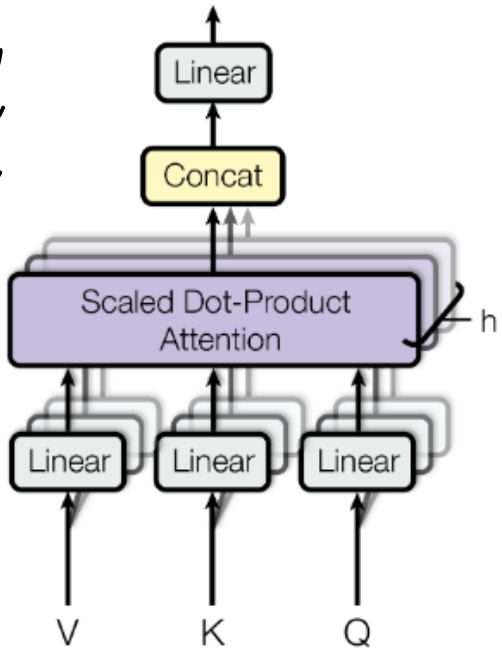


It's the same as above, just visualized

↳ Multi-head attention

↳ meaning, we linearly project Q, K, V h times and do this parallel, then concat

allows the model to jointly attend to information from different representation subspaces at different positions.



Position-wise Feed Forward Networks

↳ Each encoder, decoder block contains a fully-connected FFN

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

RELU

↳ Inner dim is → $d_{ff} = 2048$, while $d_{model} = 512$

Embedding and Softmax

↳ Learned embeddings to convert input and output tokens to dim d_{model}

↳ Learned linear transform + softmax is used to convert decoder output to next-token prob.

↳ Shared weight matrix between the 2 embedding layers + pre-softmax linear transform

↳ W_e → used to embed input tokens

$W_{e'}$ → used to embed output tokens (during training)

W_{out} → decoder output projection to logits

$$W_e = W_{e'} = W_{out}^T = W$$

Positional encoding

↳ Order of sequence is important, but the architecture doesn't have any information on it

⇓

This is why we need positional emb.

↳ same sized vector as d_{model}, and is added to K, Q

↳ In the original implementation

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

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

where,

pos = position in the sequence

i = index inside the pos. emb. vector

range(i) => 0 to $\underbrace{\frac{d_{model}}{2}}_{\text{/2 because of sin+cos}} - 1$ -> 0. starting index

This way, the wavelengths range from 2π to $10000 \cdot 2\pi$

⇓

The positional encoding at pos+k can be expressed as a linear func. of the pos. encoding at position pos when k is a fixed offset

⇓

- Linearity makes it easy for the model to learn the relationship between tokens.

↘ „Proof"

$$\omega_i = \frac{1}{10000^{\frac{2i}{d_{model}}}}$$

$$\sin(A+B) = \sin A \cos B + \cos A \sin B$$
$$\cos(A+B) = \cos A \cos B - \sin A \sin B$$

$$PE(pos, 2i) = \sin(\omega_i \cdot pos)$$
$$PE(pos, 2i+1) = \cos(\omega_i \cdot pos)$$

$$PE(pos+k, 2i) = \sin(\omega_i (pos+k)) = \sin(\omega_i pos + \omega_i k)$$
$$PE(pos+k, 2i+1) = \cos(\omega_i (pos+k)) = \cos(\omega_i pos + \omega_i k)$$

⇓⇓

$$PE(pos+k, 2i) = \sin(\omega_i pos + \omega_i k) =$$

$$= \sin(\omega_i pos)\cos(\omega_i k) + \cos(\omega_i pos)\sin(\omega_i k)$$

$$= PE(pos, 2i)\cos(\omega_i k) + PE(pos, 2i+1)\sin(\omega_i k)$$

$$PE(pos+k, 2i+1) = \cos(\omega_i pos + \omega_i k) =$$

$$\cos(\omega_i pos)\cos(\omega_i k) - \sin(\omega_i pos)\sin(\omega_i k)$$

$$= PE(pos, 2i+1)\cos(\omega_i k) - PE(pos, 2i)\sin(\omega_i k)$$

As we can see, $pos+k$ is indeed a lin. comb of pos

Why self-attention?

↳ Consdand number of sequentially executed operations, regardless of sequence length
↳ because all positions run in parallel

↳ complexity per layer → $O(n^2 \cdot d)$
- n: sequence length
- d: vector size

n^2, because we check everything for everything (all vectors)

This is faster than RNNs $O(n \cdot d^2)$, if $n < d$, which is often the case

↳ Proposed problem
↳ dealing with long sequences (n^2)

↓

Restricted self-attention: could only see r positions around itself.

⇓

reduced complexity: $O(n \cdot r \cdot d)$

explains $n \cdot r$ as well