Some Reviews :

How do we deal with time dependent data ?

① time as input = ??

(Time 2 Vec ? , one-hot encoding ?)

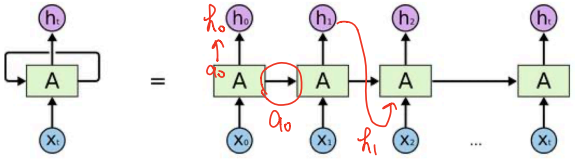
② measure dependency between data

⇒ to have memory property.

Recurrent Network:

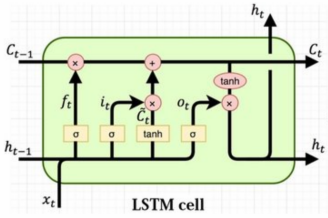
※ diff definition, same concept

Vanilla RNN :



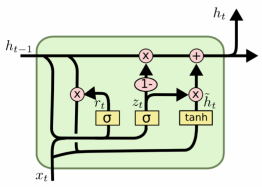
A: is the same network

LSTM :



$$
\begin{aligned}
i_t &= \sigma\left(x_t U^i + h_{t-1} W^i\right) \\
f_t &= \sigma\left(x_t U^f + h_{t-1} W^f\right) \\
o_t &= \sigma\left(x_t U^o + h_{t-1} W^o\right) \\
\tilde{C}_t &= \tanh\left(x_t U^g + h_{t-1} W^g\right) \\
C_t &= \sigma\left(f_t * C_{t-1} + i_t * \tilde{C}_t\right) \\
h_t &= \tanh(C_t) * o_t
\end{aligned}
$$

GRU :



$$
\begin{aligned}
z_t &= \sigma\left(W_z \cdot [h_{t-1}, x_t]\right) \\
r_t &= \sigma\left(W_r \cdot [h_{t-1}, x_t]\right) \\
\tilde{h}_t &= \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right) \\
h_t &= (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t
\end{aligned}
$$

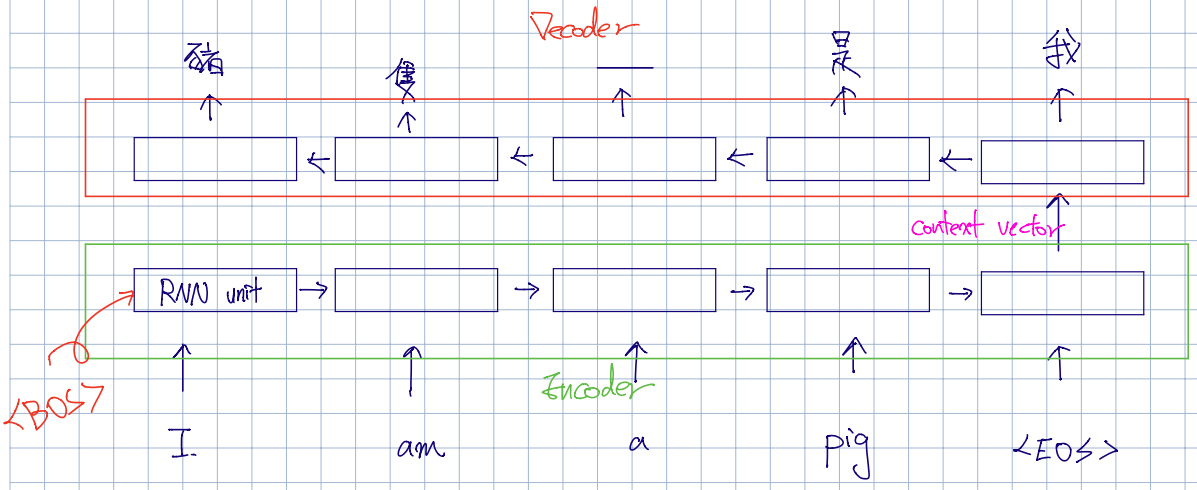
5mins

How about translation tasks ?

Input : seq , Output : seq.

(length of seq cannot be pre-determined)

=> Encoder - Decoder model.



※ Encoder actions are trivial, but

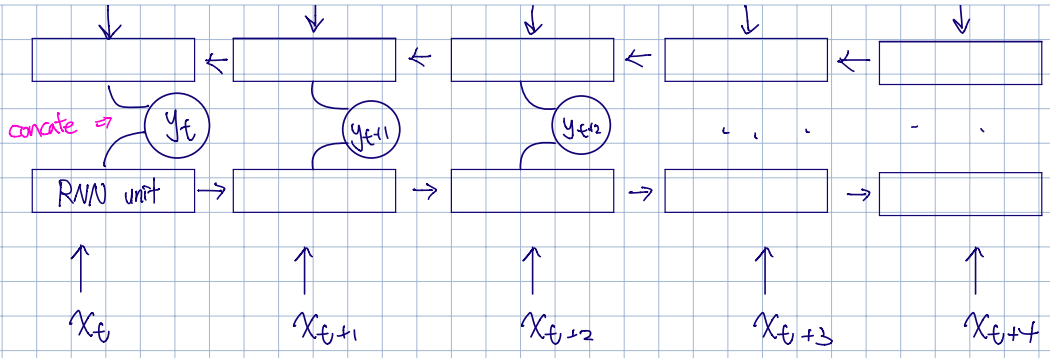
Decoder: In training stage:

h_{t-1} & h_t => Using truth answer as input of h_t

y_{t-1}

Bidirectional LSTM instead of just RNN units, i

x_t x_{t+1} x_{t+2} x_{t+3} x_{t+4}



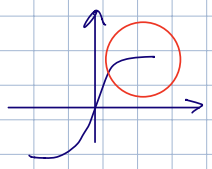
7min

But still, we will have some problem.

① context vector is a fixed dim vector.

⇒ no matter how long is the sequence

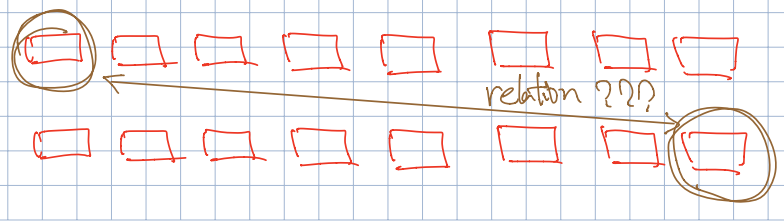
② RNN has the gradient vanishing, exploding problem. ⇒ while use ReLU.

⇒  sigmoid. ,  tanh

CNN ⇒ receptive field problem.

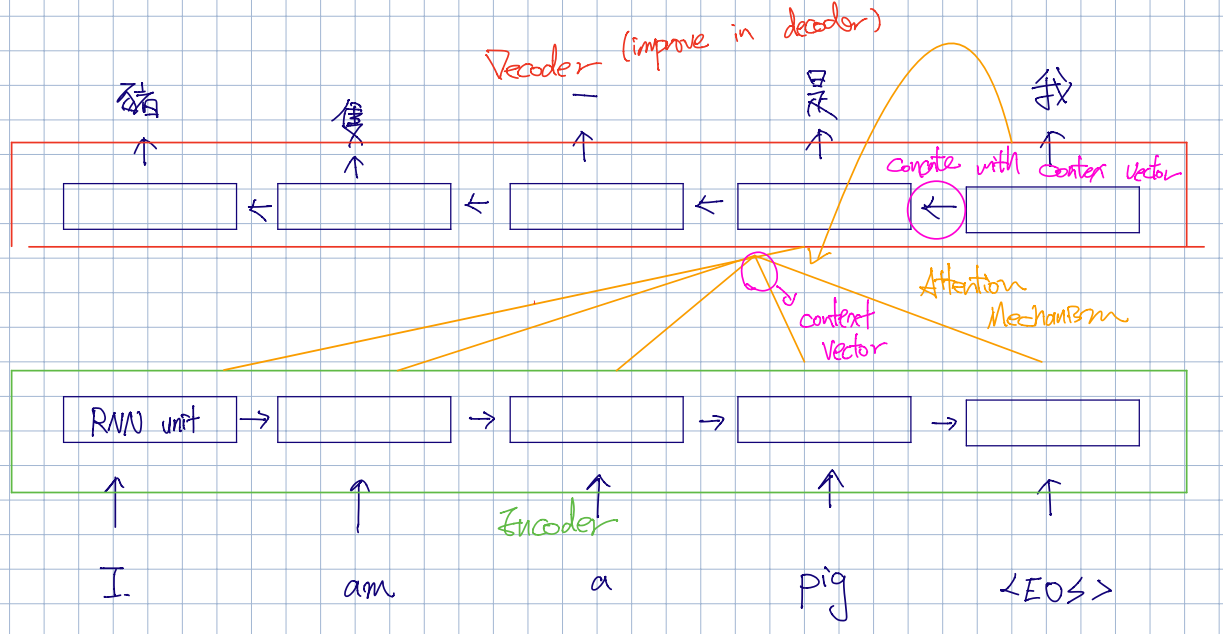
Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

③ As the seq body longer, the previous token has no effectiess. Same as in Decoder.



7min

Attention !



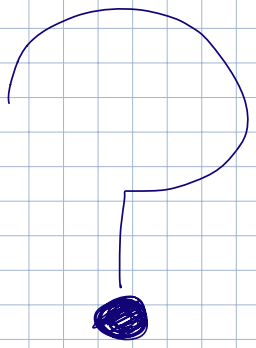
Cacluate each units' their own context vector

There are lots of ways to do "attention":

Ex: dot-product attention, multiplicative attention, additive attention

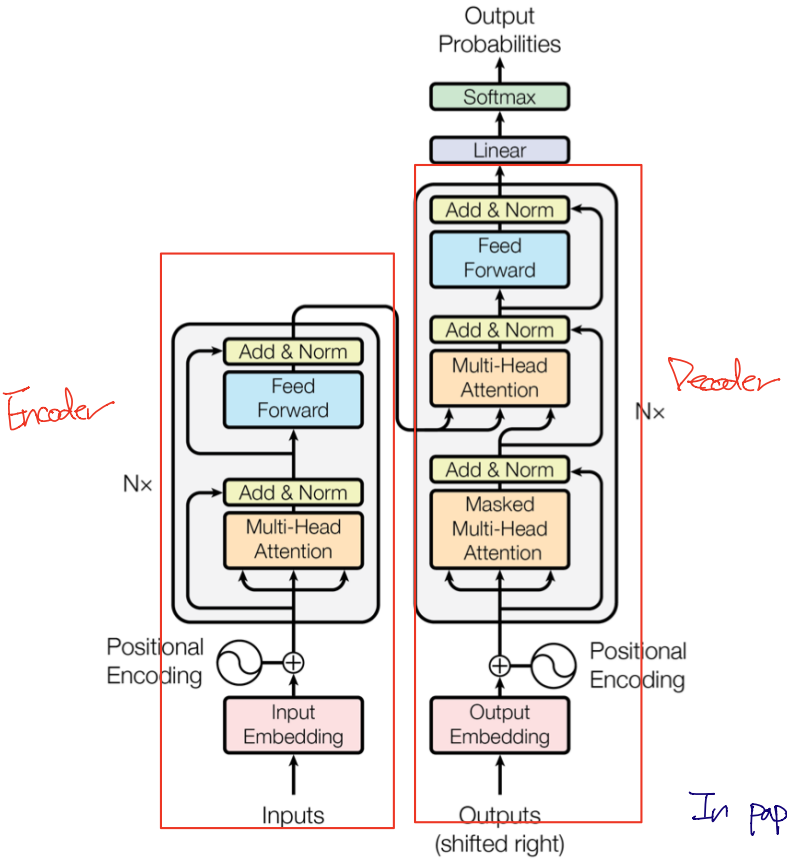
We now solved problem ①. ②.

=> Is there anything else we can improve ?



7min

Transformer : self-attention (Bye^2 RNN, CNN)

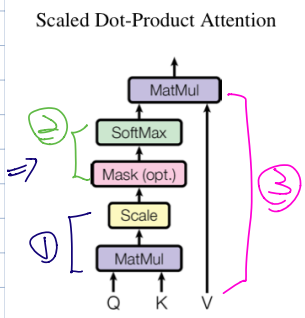


In paper, N=6.

Don't worry! we will go through step by step.

1. Attention

(i) Scaled dot-product attention



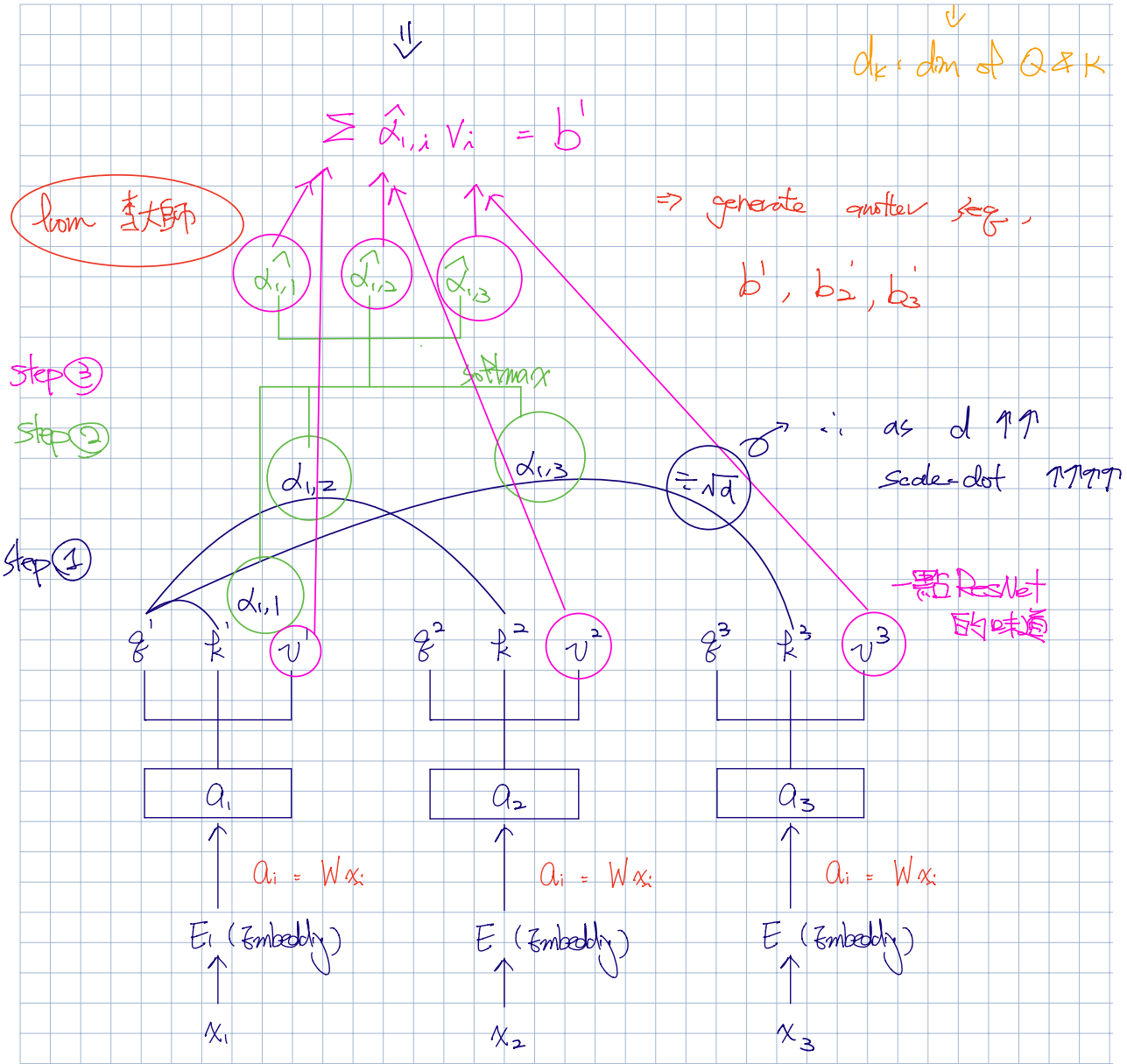
From paper ⇒

grey key value

↓ ↓ ↓

Attention (Q, K, V)

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



Generalization form:

① $W^q a_i = q^i$

$W^k a_i = k^i$

$W^v a_i = v^i$

$\Rightarrow$

$W^q \; a^1 a^2 a^3 = q^1 q^2 q^3$

$W^k \; a = k$

$W^v \; a = v$

Similarly,

② $\begin{bmatrix} \alpha_{1,1} & \alpha_{2,1} & \alpha_{3,1} \\ \alpha_{1,2} & \alpha_{2,2} & \alpha_{3,2} \\ \alpha_{1,3} & \alpha_{2,3} & \alpha_{3,3} \end{bmatrix} = \begin{bmatrix} k^1 \\ k^2 \\ k^3 \end{bmatrix} \begin{bmatrix} q^1 q^2 q^3 \end{bmatrix}$

$A \Downarrow \text{softmax}$

$\hat{A}$

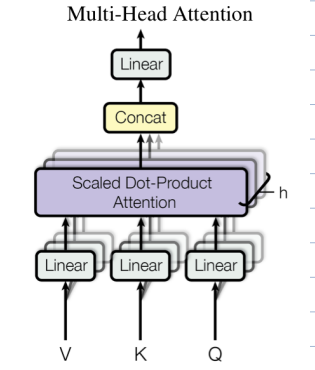
Similarly

③ $\begin{bmatrix} b_1 & b_2 & b_3 \end{bmatrix} = \begin{bmatrix} v_1 & v_2 & v_3 \end{bmatrix} \hat{A}$

Eventually, we are just caculating matrix multiplication !!

↳ speed up, paralle computation in GPU.

(ii) Multi-head Attention



MultiHead (Q, K, V)

= Concat (head$_1$, head$_2$, .. head$_h$) W^O

$\Downarrow$

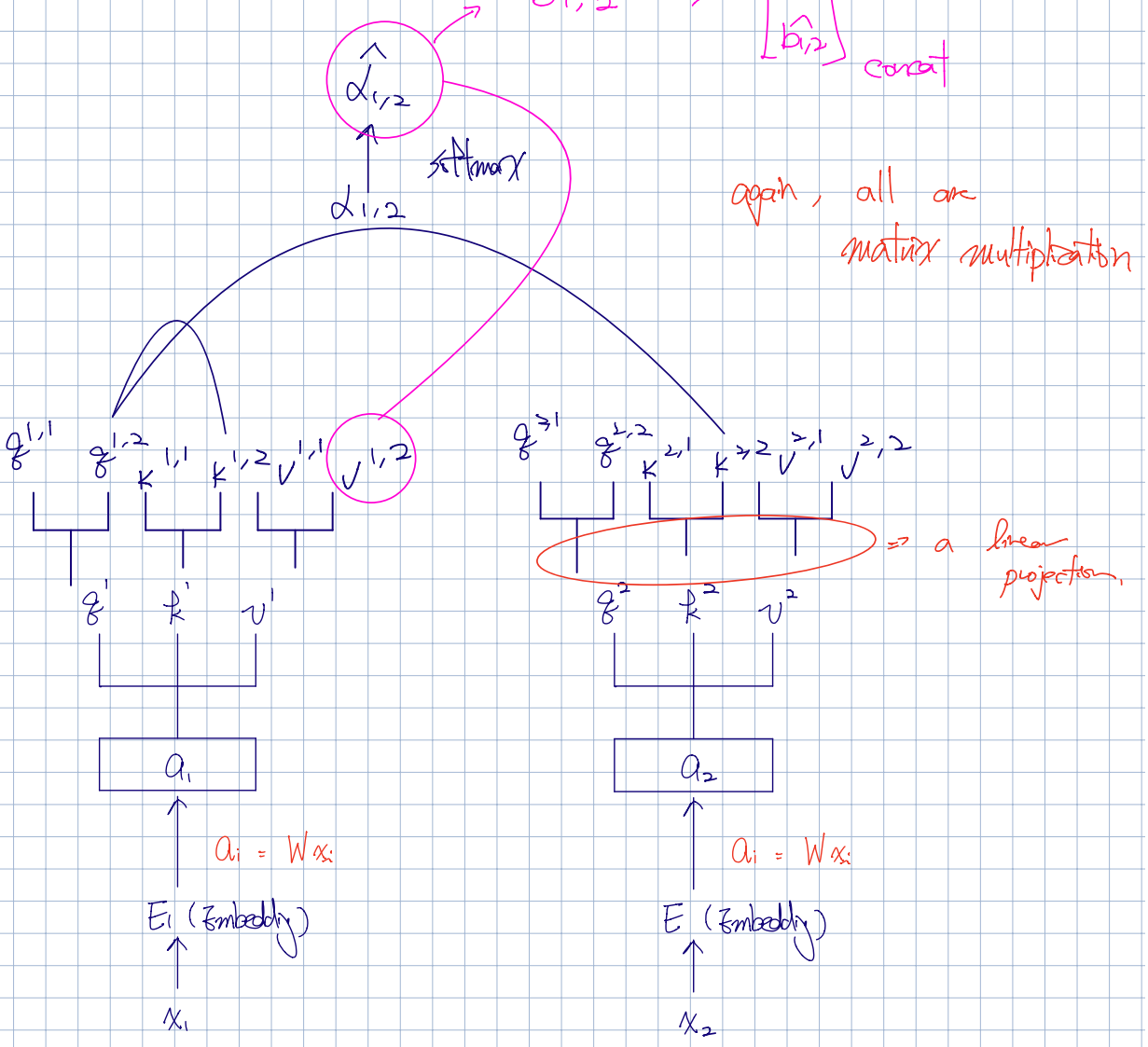
head$_i$ = Attention (QW_i^Q, KW_i^K, VW_i^V)

$head_i = Attention(QW_i, KW_i, VW_i)$

$\downarrow \in \mathbb{R}^{d_{model} \times d_k}$

Similar as above, but heads ?

$\hat{b}_{1,2} \Rightarrow \begin{bmatrix} \hat{b}_{1,1} \\ \hat{b}_{1,2} \end{bmatrix}$ concat



again, all are matrix multiplication !

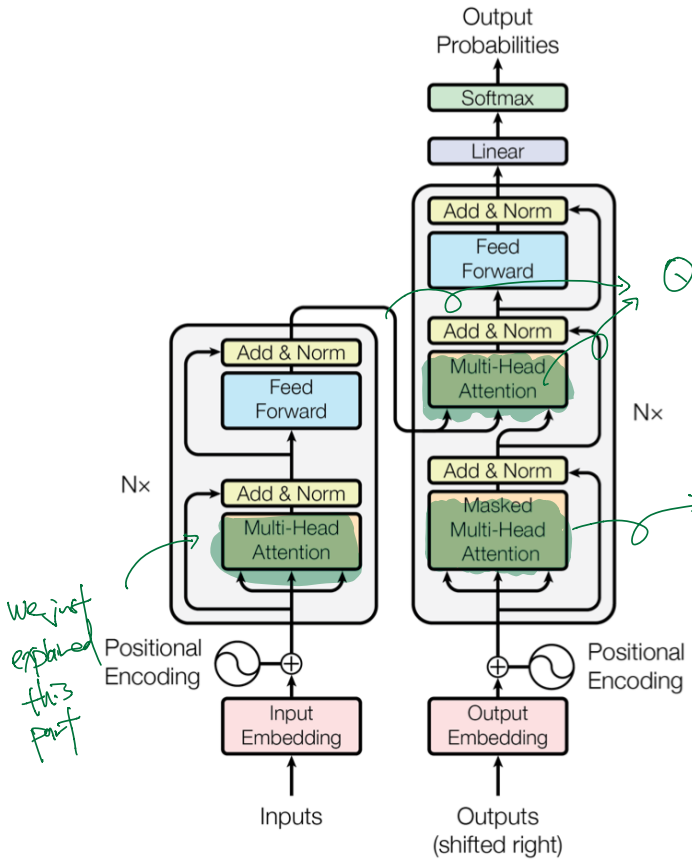
⇒ a linear projection.

In paper, # heads = 8, $d_{k,q,v} = 64$, $d_{model} = 64 \times 8 = 512$

↳ we can regard heads as different characteristic of words, ex: grammar, syntax, ...

Back to Graph:

20min



Q, V comes from Encoder part

we just explained this part

In order to prevent future information to be accounted while predicting,

⇓

We create a "mask", which sets the value of weight → $-\infty$

(actually, $-10{,}000$)

Ex: In Decoder,

我是一隻豬.

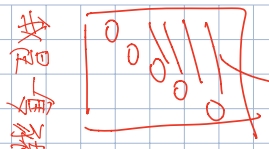
(set to $-\infty$)

this word should not have attention with

(while doing this, it's not hard to imagine we should set the QK^T's W. to -10000.

(mask upper triangle)

我是一隻豬 ⇓



Should be masked!

Q . K .

Note: there is another mask, we will talk later.

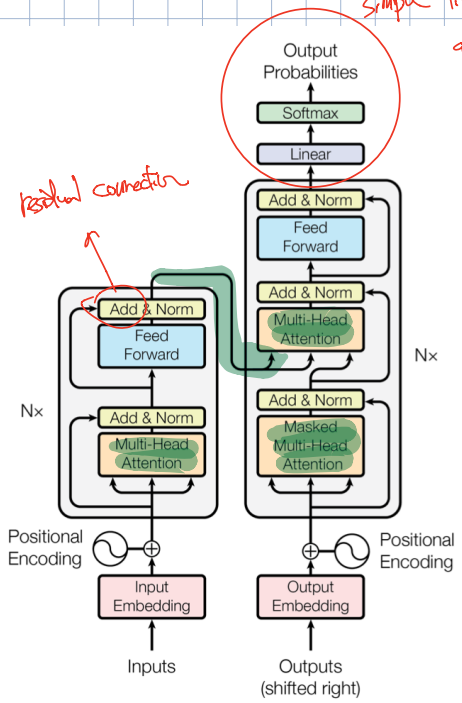
FFN & Norm:

Smin

Normalize before → softmax

(weights × $\sqrt{d_{model}}$)

simple linear transformation and softmax before output ⇒ is another NN



residual connection

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

↓

ReLU activation

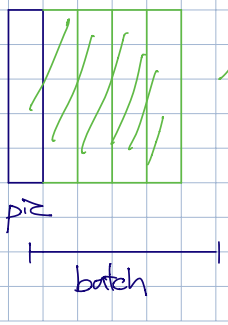
actually, it is just 2 x NN.

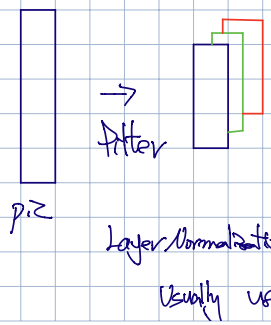
※ in paper, d output = 512

d inner = 2014

d ff.

Norm:





Layer Normalization:

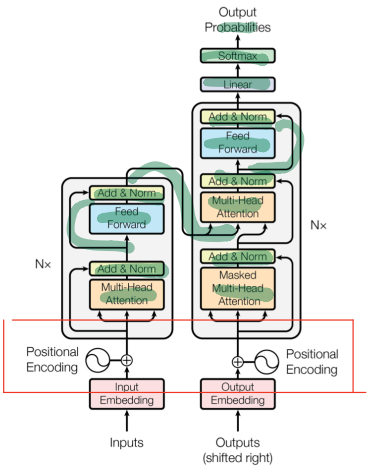
Usually use in RNN

The Last and most important idea!!

(In my view)

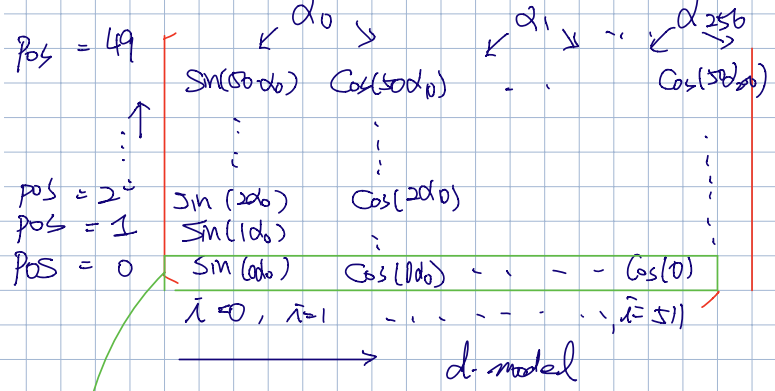
Positional Encoding:

our architecture cannot tell the model the order about the seq !!!



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

A matrix, Let $\alpha_i = \frac{1}{10000^{2i/d\text{-model}}}$



little bit similar to base.

surprisingly it's a fixed value too.

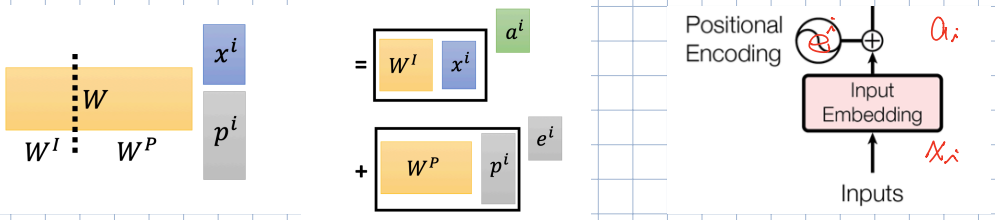
⇒ and ∀k, PE(pos+k) can be rewritten as the linear combination of PE(pos+k).

※ In practice, we just create sin, cos np and concat

Why added instead of concate it ? 讚嘆李大師

=> Suppose p_i represent one-hot position encoding.

x_i is the input



↳ and the residual connection prevents this information lose !!!

※ Optimizer & Dropout :

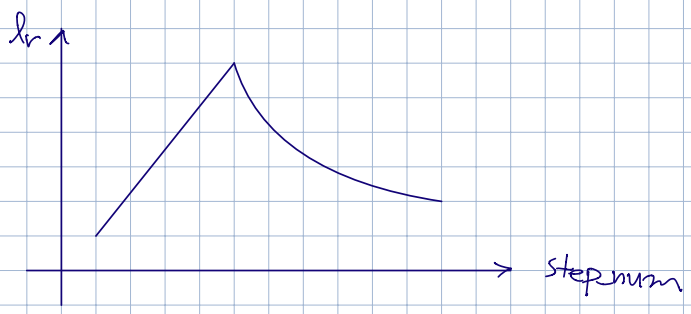
We used the Adam optimizer [20] with $\beta_1 = 0.9$, $\beta_2 = 0.98$ and $\epsilon = 10^{-9}$. We varied the learning rate over the course of training, according to the formula:

$$lrate = d_{\text{model}}^{-0.5} \cdot \min(step\_num^{-0.5}, step\_num \cdot warmup\_steps^{-1.5}) \quad (3)$$

linear ↑ while num_steps < 4000

start decay while num_steps > 4000

In paper, Step_num = 4000



Dropout rate = 0.1 in every output of sublayer.