CS60010: Deep Learning

Spring 2021

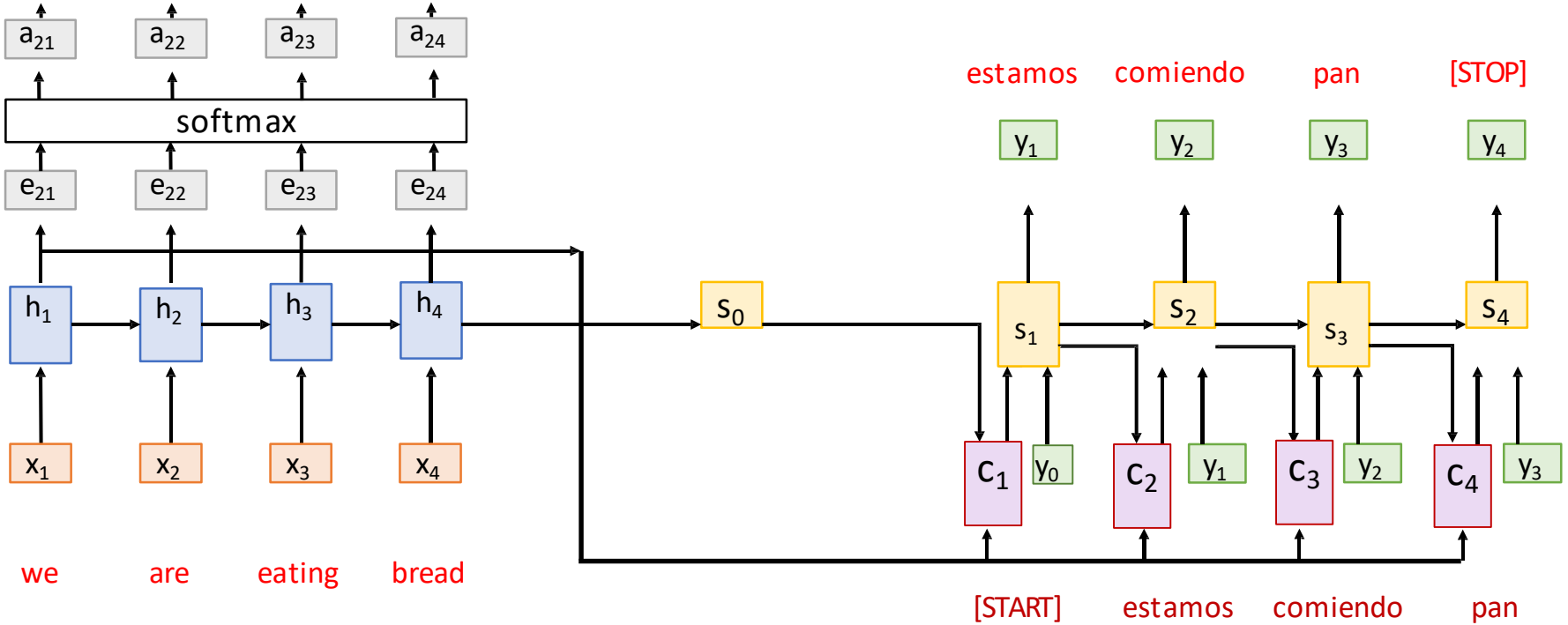
Sudeshna Sarkar

Module 5

Self-Attention and
Transformer Architecture

16 Mar 2021

Attention


a_{21} a_{22} a_{23} a_{24}
softmax
e_{21} e_{22} e_{23} e_{24}
h_1 h_2 h_3 h_4
x_1 x_2 x_3 x_4
we are eating bread
s_0
estamos comiendo pan [STOP]
y_1 y_2 y_3 y_4
s_1 s_2 s_3 s_4
c_1 y_0 c_2 y_1 c_3 y_2 c_4 y_3
[START] estamos comiendo pan

Attention-only Translation Models

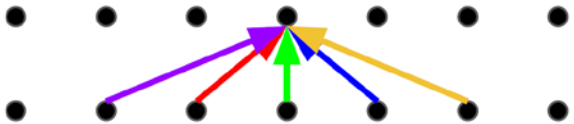
- Problems with recurrent networks:
- Sequential training and inference: Time grows in proportion to sentence length. Hard to parallelize.
- Long-range dependencies have to be remembered across many single time steps.
- Tricky to learn hierarchical structures ("car", "blue car", "into the blue car"…)

- Alternative:
- Convolution – but has other limitations.

Self-Attention

- Information flows from within the same subnetwork (encoder or decoder).
- Convolution applies fixed transform weights. Self-attention applies variable weights.

Convolution



Self-Attention

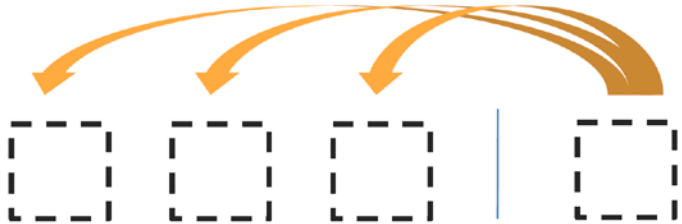
Self-Attention “Transformers”

- Constant path length between any two positions.
- Variable receptive field (or the whole input sequence).
- Supports hierarchical information flow by stacking self-attention layers.
- Trivial to parallelize.
- Attention weighting controls information propagation.
- Can replace word-based recurrence entirely.

Vaswani et al. “Attention is all you need”, arXiv 2017

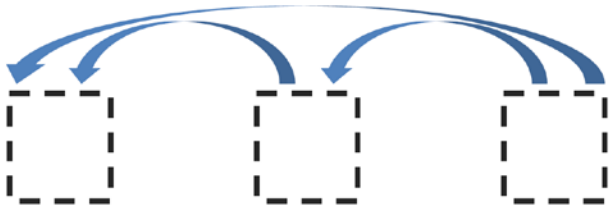
https://arxiv.org/abs/1706.03762

Attention in Transformer Networks



Encoder-Decoder Attention

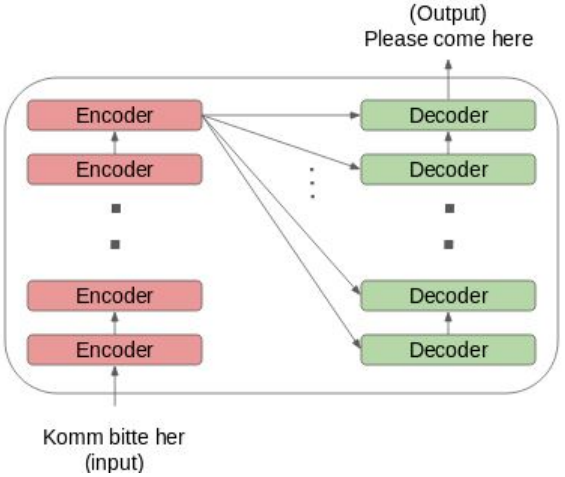
Encoder Self-Attention

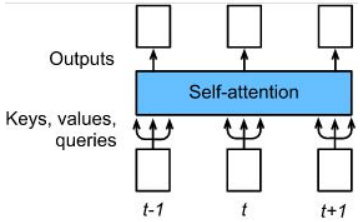


MaskedDecoder Self-Attention

Transformers

- The Transformer starts by generating initial representations for each word.
- Using self-attention, it aggregates information from all of the other words, generating a new representation per word
- This step is repeated multiple times in parallel for all words, successively generating new representations.

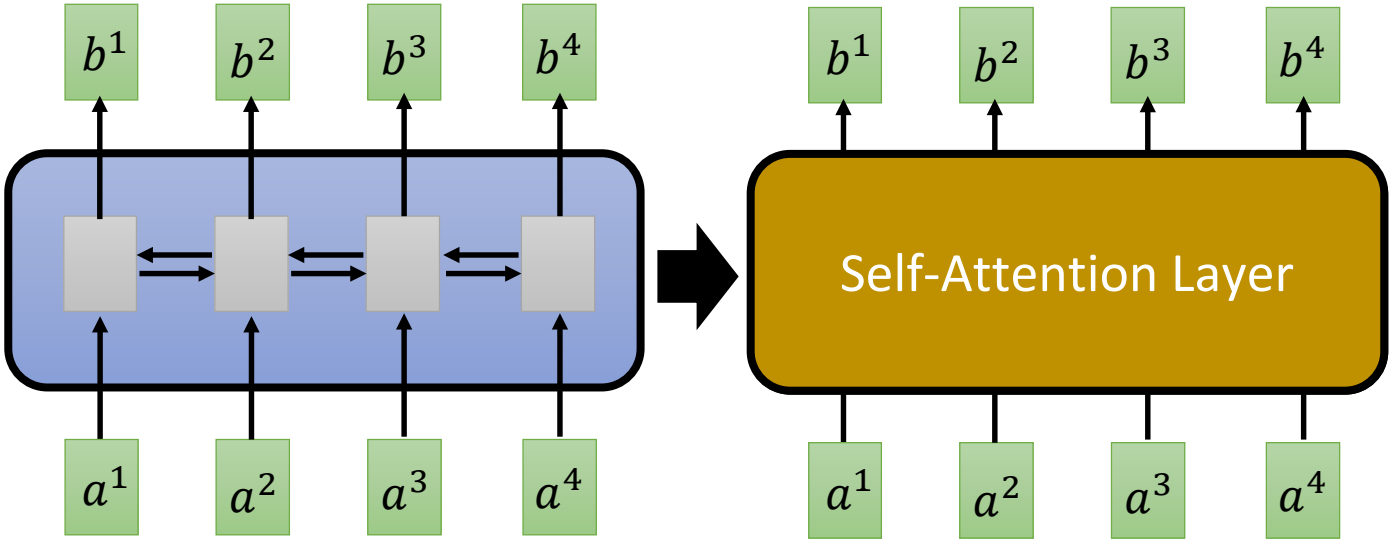




Self-Attention

b^i is obtained based on the whole input sequence.

b^1, b^2, b^3, b^4 can be computed in parallel.



You can try to replace any thing that has been done by RNN with self-attention.

q: query (to match others)

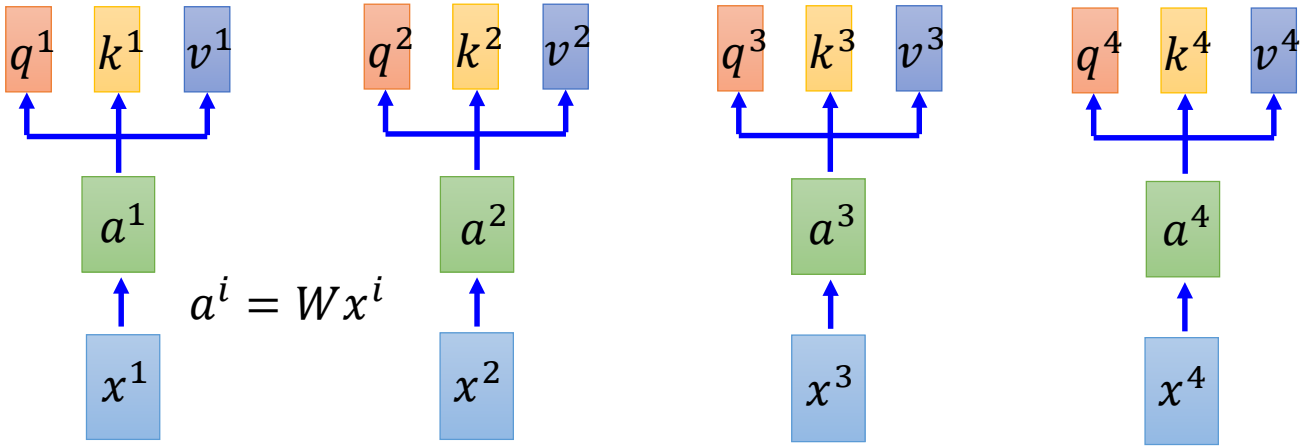
$$q^i = W^q a^i$$

k: key (to be matched)

$$k^i = W^k a^i$$

v: information to be extracted

$$v^i = W^v a^i$$

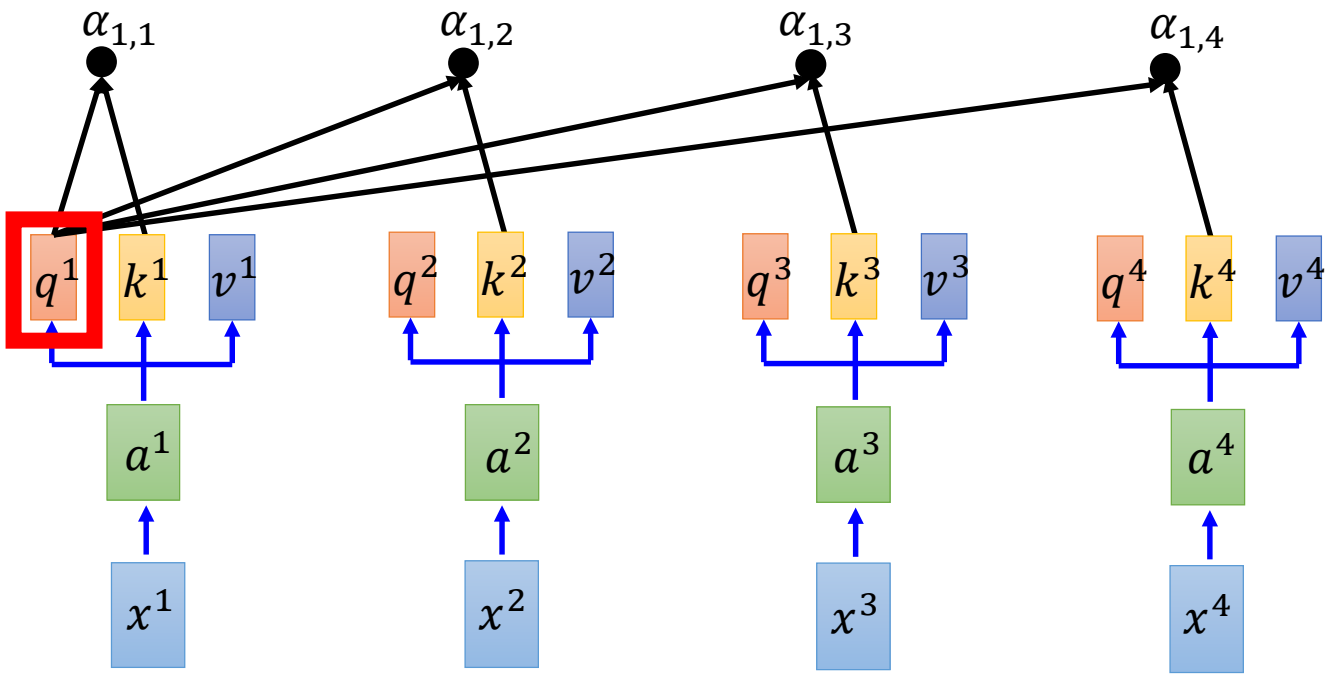


Scaled Dot-Product Attention

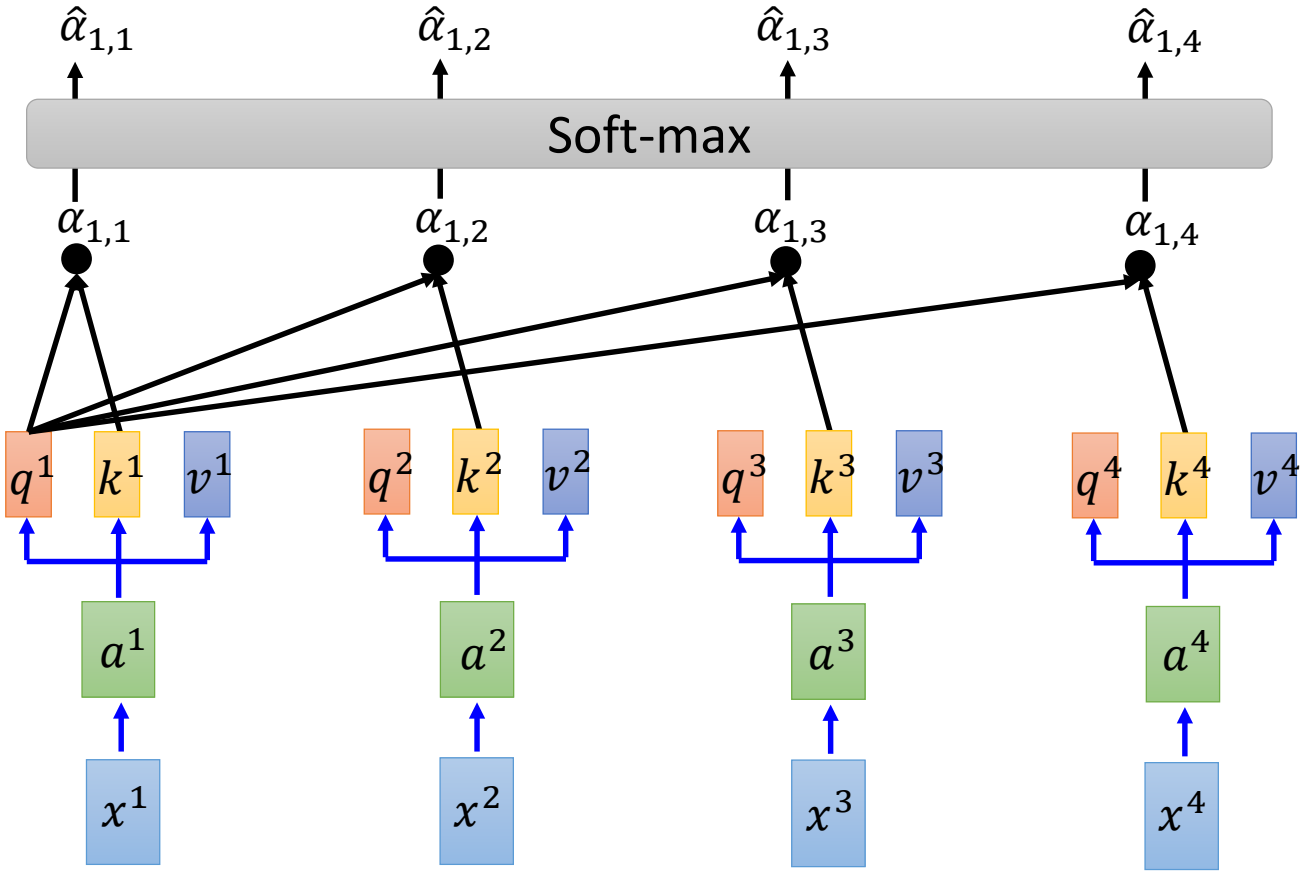
d is the dim of q and k

$$\alpha_{1,i} = q^1 \cdot k^i / \sqrt{d}$$

dot product

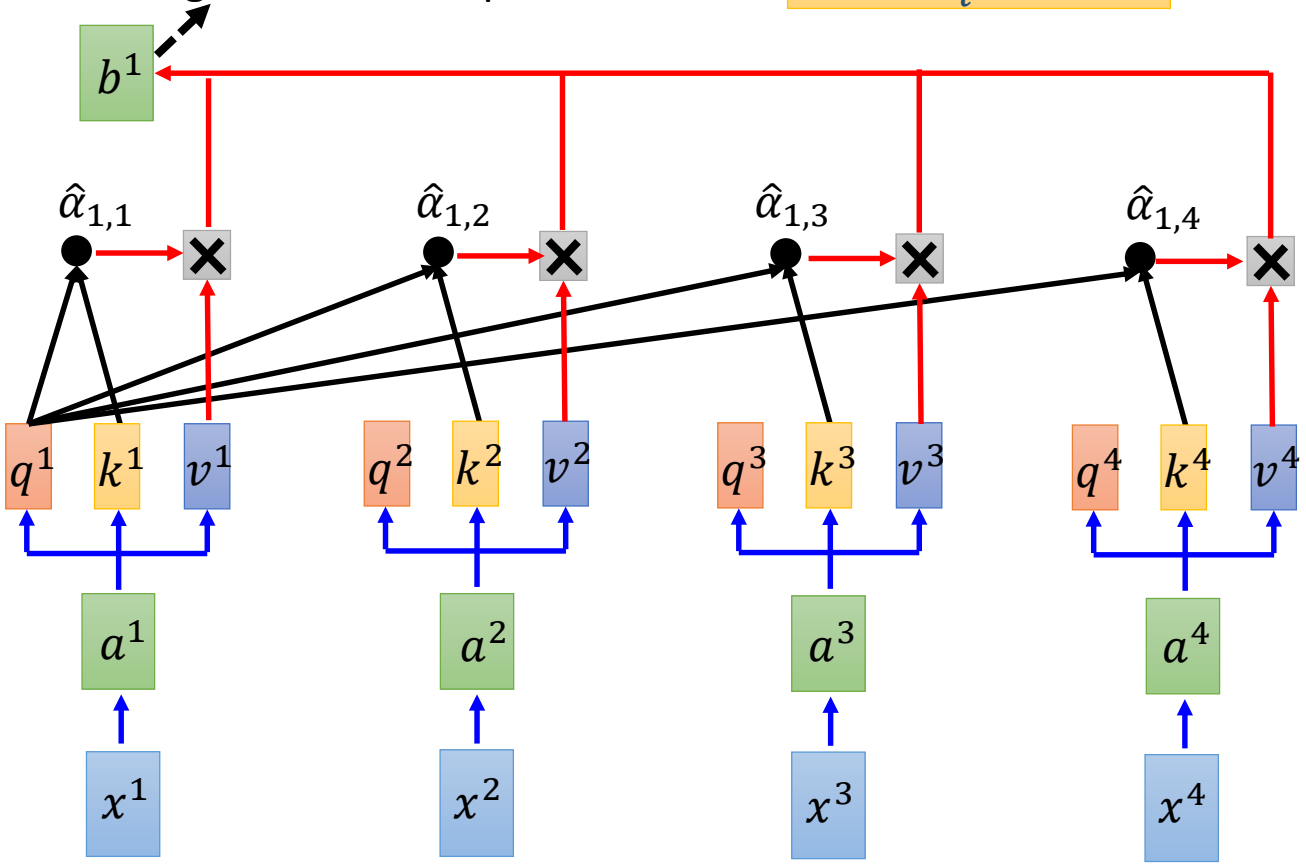


$$\hat{\alpha}_{1,i} = exp(\alpha_{1,i}) / \sum_j exp(\alpha_{1,j})$$

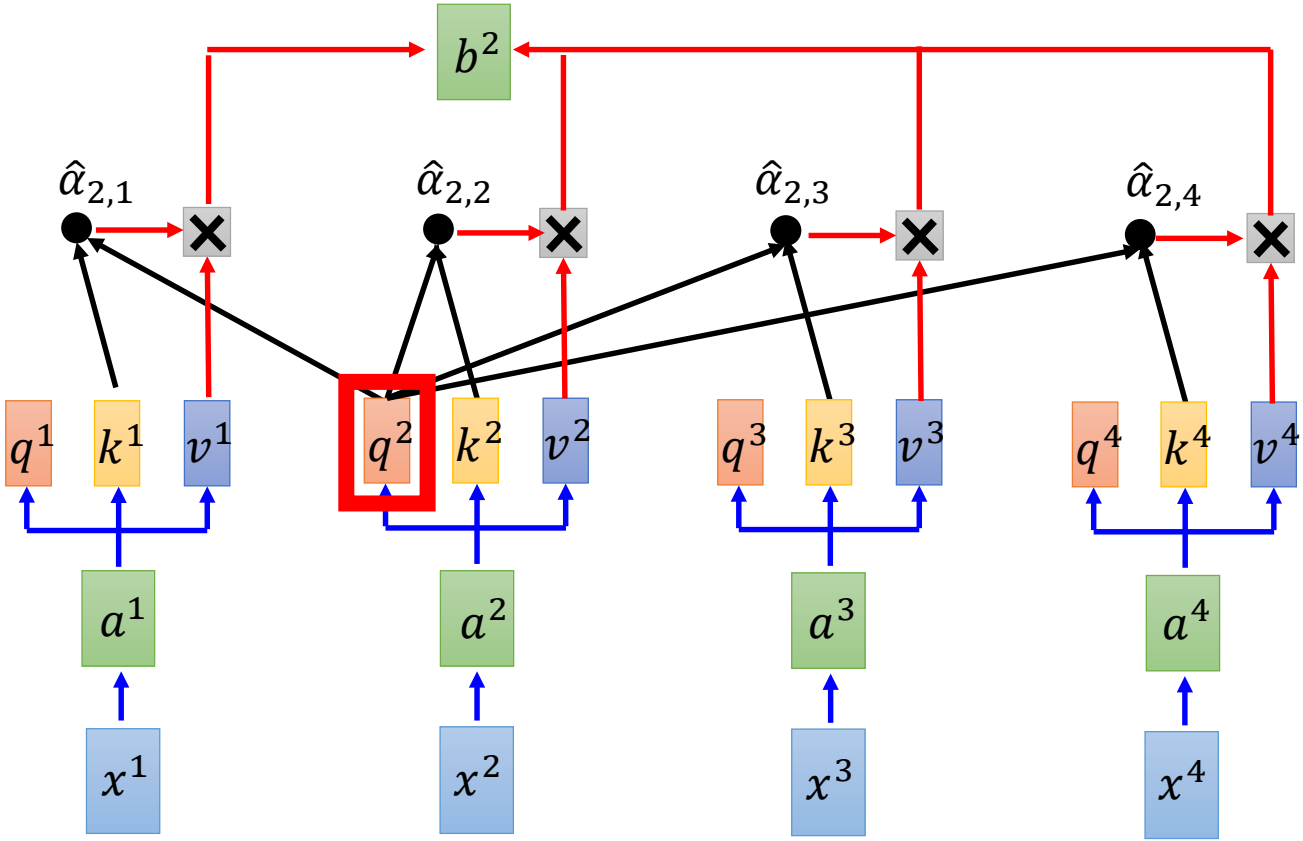


$$b^1 = \sum_i \hat{\alpha}_{1,i} v^i$$

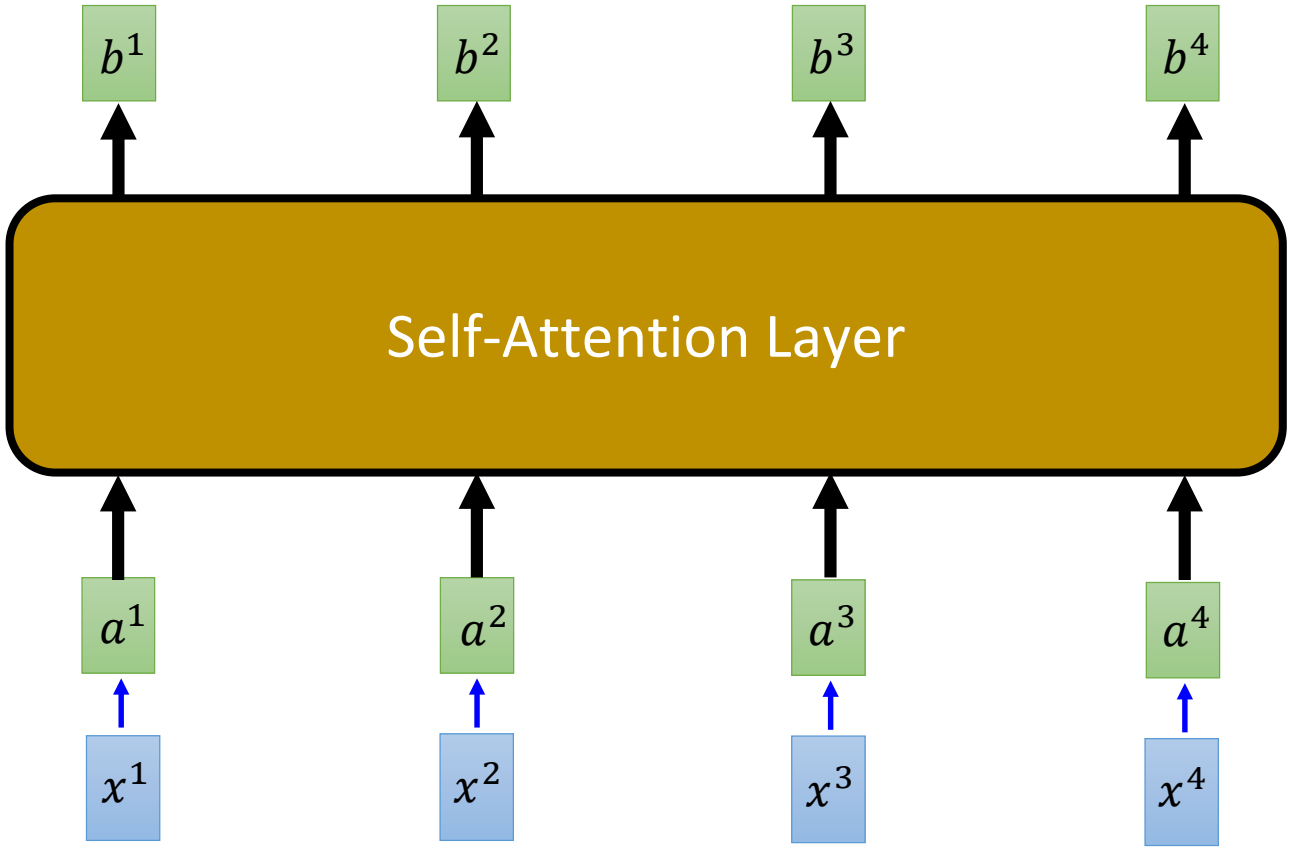
Considering the whole sequence



$$b^2 = \sum_i \hat{\alpha}_{2,i} v^i$$



b^1, b^2, b^3, b^4 can be computed in parallel.



$$q^i = W^q a^i$$

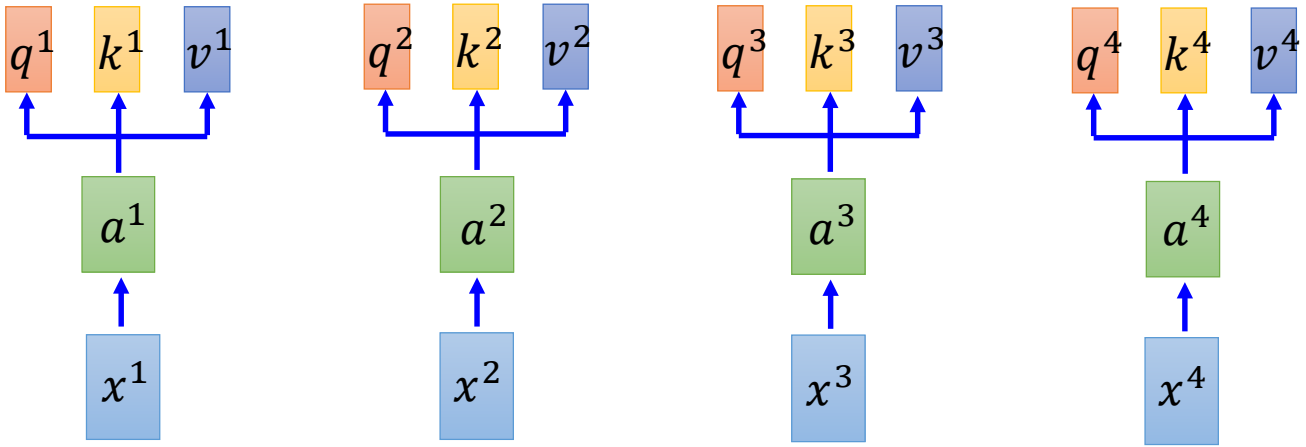
$$k^i = W^k a^i$$

$$v^i = W^v a^i$$

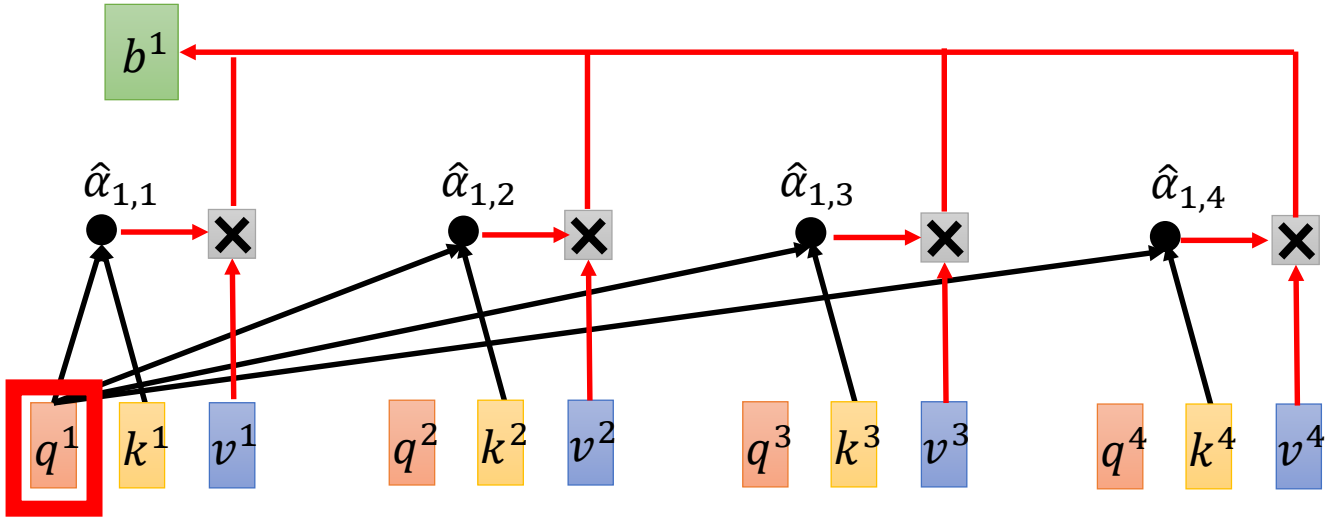
$$\underbrace{q^1\, q^2\, q^3\, q^4}_{Q} = W^q \underbrace{a^1\, a^2\, a^3\, a^4}_{I}$$

$$\underbrace{k^1\, k^2\, k^3\, k^4}_{K} = W^k \underbrace{a^1\, a^2\, a^3\, a^4}_{I}$$

$$\underbrace{v^1\, v^2\, v^3\, v^4}_{V} = W^v \underbrace{a^1\, a^2\, a^3\, a^4}_{I}$$



Self-attention



$\alpha_{1,1} = k^1 q^1$ $\alpha_{1,2} = k^2 q^1$

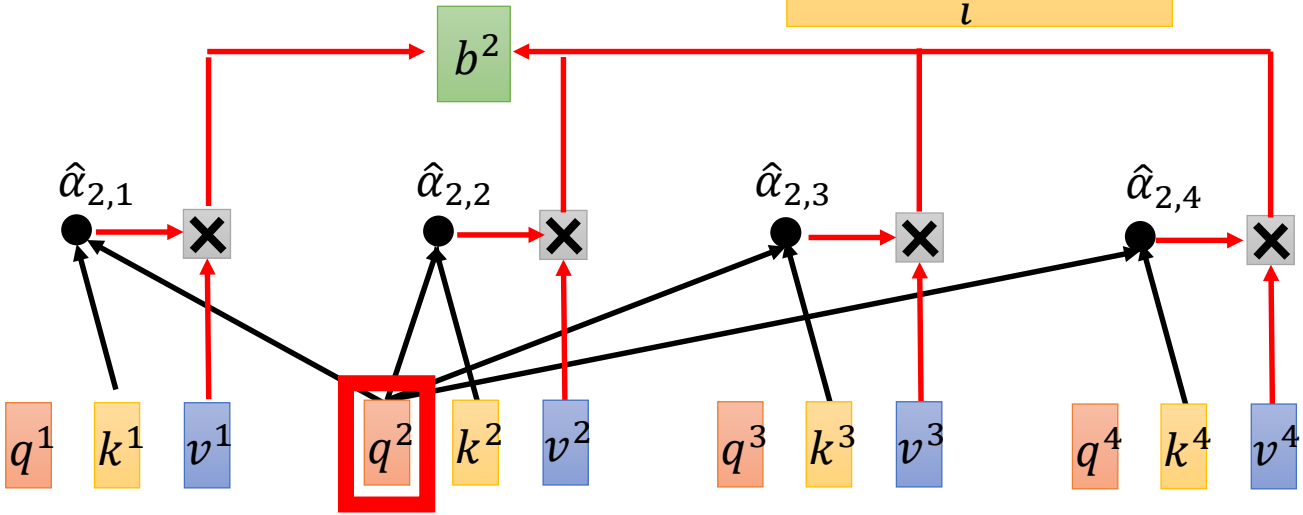
$\alpha_{1,3} = k^3 q^1$ $\alpha_{1,4} = k^4 q^1$

(ignore $\sqrt{d}$ for simplicity)

$$\begin{bmatrix}\alpha_{1,1}\\ \alpha_{1,2}\\ \alpha_{1,3}\\ \alpha_{1,4}\end{bmatrix} = \begin{bmatrix}k^1\\ k^2\\ k^3\\ k^4\end{bmatrix} q^1$$

Self-attention

$$b^2 = \sum_i \hat{\alpha}_{2,i} v^i$$

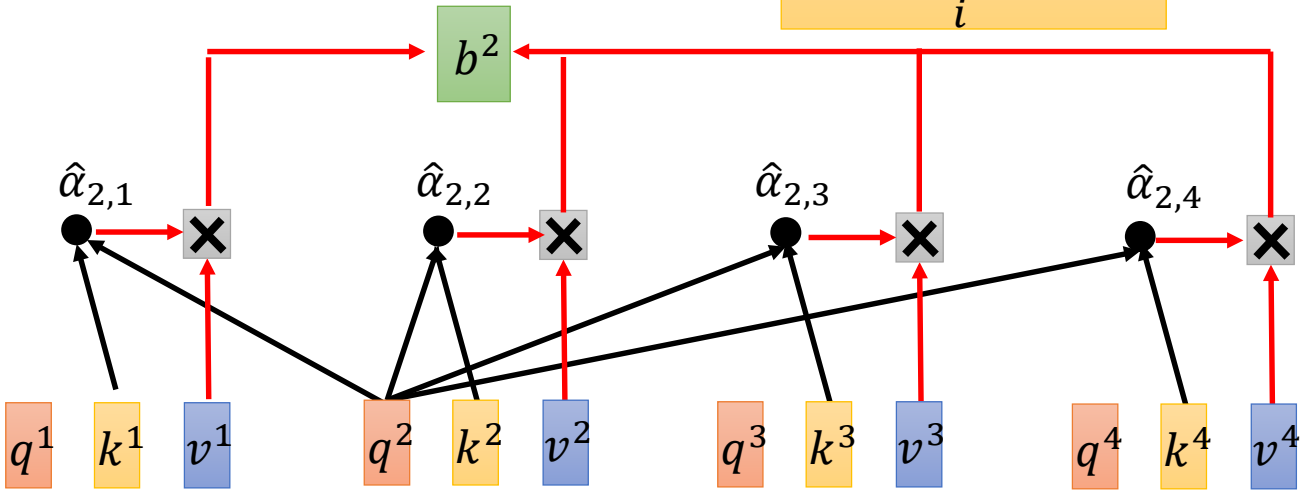


$$\begin{bmatrix} \hat{\alpha}_{1,1} & \hat{\alpha}_{2,1} & \hat{\alpha}_{3,1} & \hat{\alpha}_{4,1} \\ \hat{\alpha}_{1,2} & \hat{\alpha}_{2,2} & \hat{\alpha}_{3,2} & \hat{\alpha}_{4,2} \\ \hat{\alpha}_{1,3} & \hat{\alpha}_{2,3} & \hat{\alpha}_{3,3} & \hat{\alpha}_{4,3} \\ \hat{\alpha}_{1,4} & \hat{\alpha}_{2,4} & \hat{\alpha}_{3,4} & \hat{\alpha}_{4,4} \end{bmatrix} \leftarrow \begin{bmatrix} \alpha_{1,1} & \alpha_{2,1} & \alpha_{3,1} & \alpha_{4,1} \\ \alpha_{1,2} & \alpha_{2,2} & \alpha_{3,2} & \alpha_{4,2} \\ \alpha_{1,3} & \alpha_{2,3} & \alpha_{3,3} & \alpha_{4,3} \\ \alpha_{1,4} & \alpha_{2,4} & \alpha_{3,4} & \alpha_{4,4} \end{bmatrix} = \begin{bmatrix} k^1 \\ k^2 \\ k^3 \\ k^4 \end{bmatrix} \begin{bmatrix} q^1 & q^2 & q^3 & q^4 \end{bmatrix}$$

$\hat{A}$ A K^T Q

Self-attention

$$b^2 = \sum_i \hat{\alpha}_{2,i} v^i$$

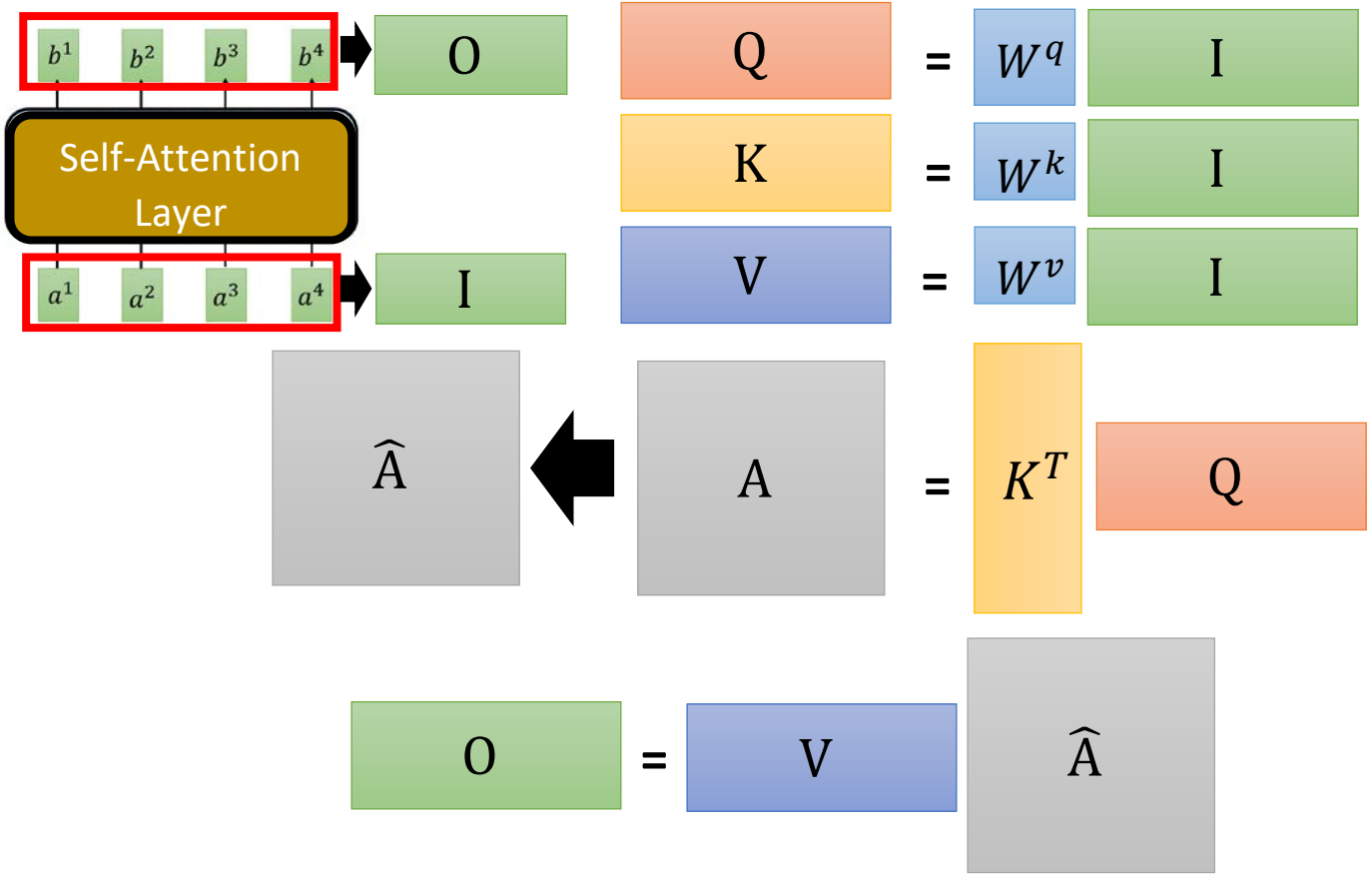


$b^1 b^2 b^3 b^4$ = $v^1 v^2 v^3 v^4$

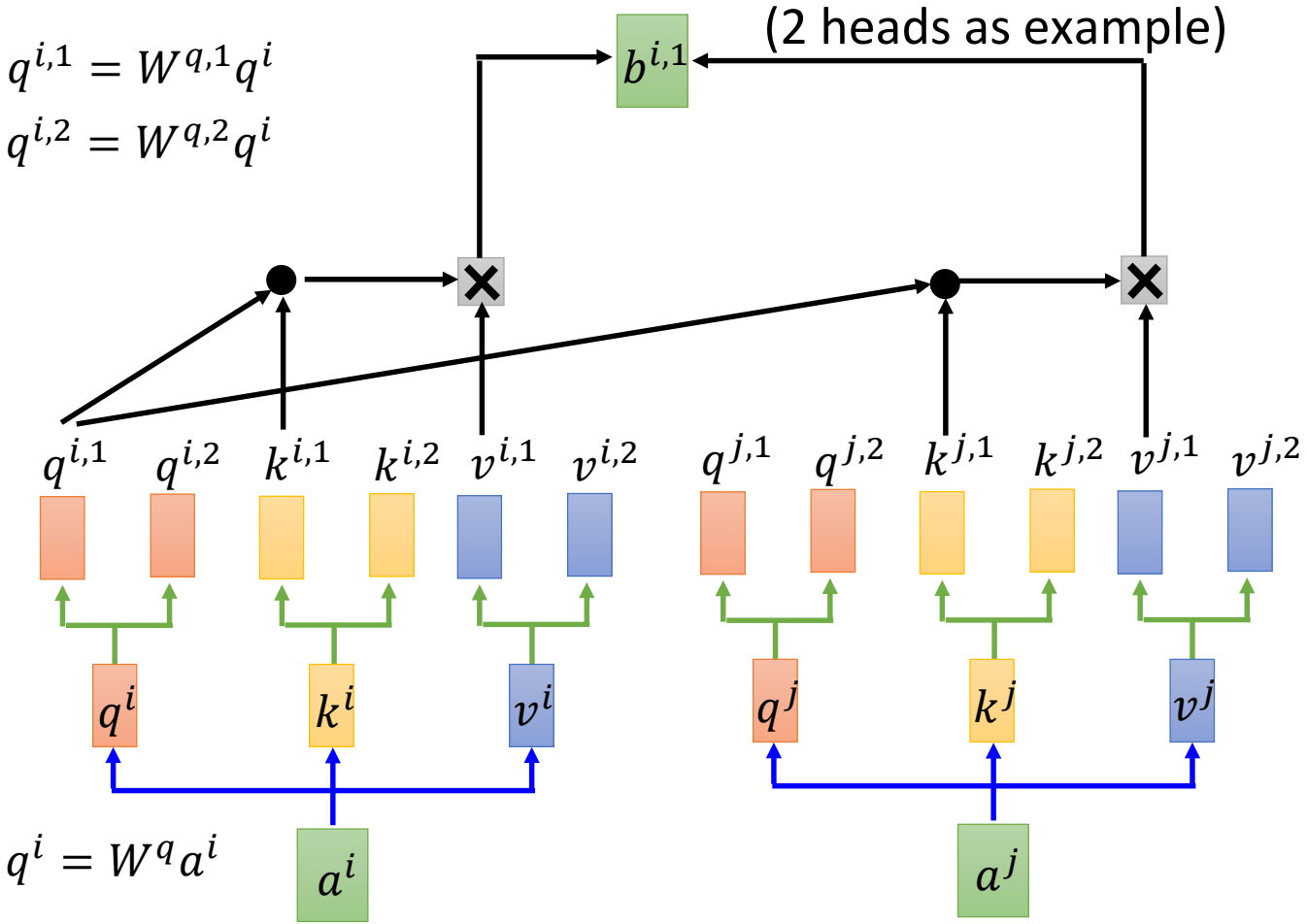
O V

$$\hat{A} = \begin{bmatrix} \hat{\alpha}_{1,1} & \hat{\alpha}_{2,1} & \hat{\alpha}_{3,1} & \hat{\alpha}_{4,1} \\ \hat{\alpha}_{1,2} & \hat{\alpha}_{2,2} & \hat{\alpha}_{3,2} & \hat{\alpha}_{4,2} \\ \hat{\alpha}_{1,3} & \hat{\alpha}_{2,3} & \hat{\alpha}_{3,3} & \hat{\alpha}_{4,3} \\ \hat{\alpha}_{1,4} & \hat{\alpha}_{2,4} & \hat{\alpha}_{3,4} & \hat{\alpha}_{4,4} \end{bmatrix}$$

Self-attention



Multi-head Self-attention

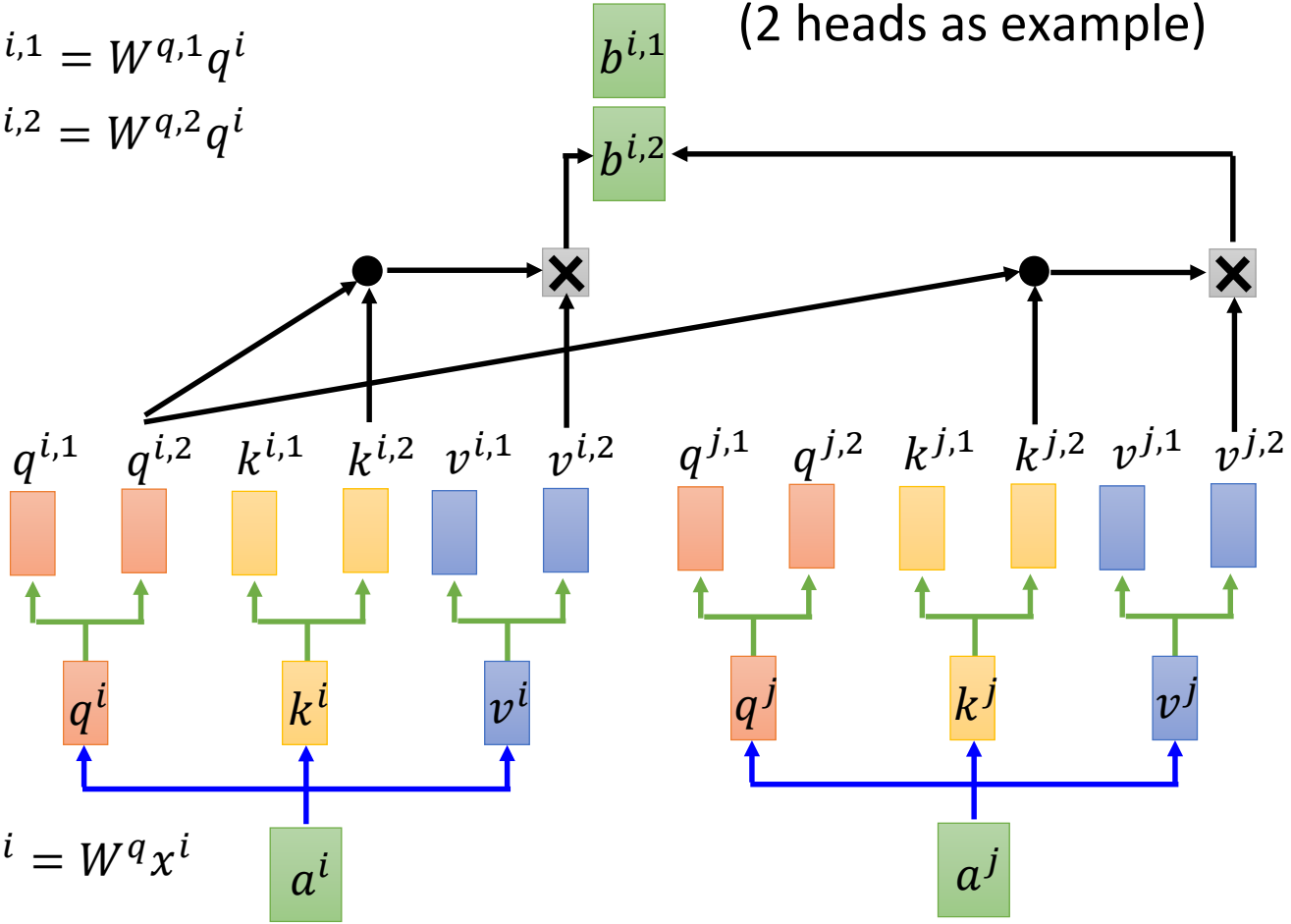


Multi-head Self-attention

$q^{i,1} = W^{q,1} q^i$

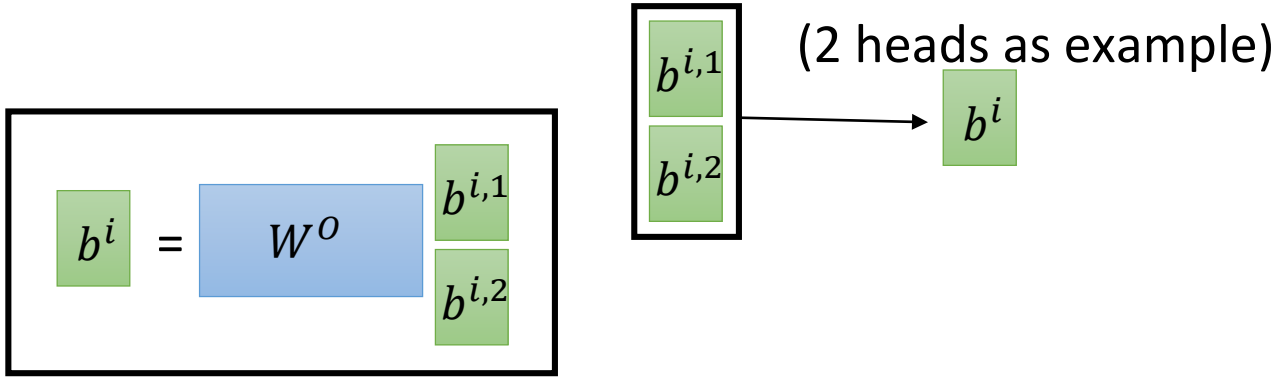
$q^{i,2} = W^{q,2} q^i$

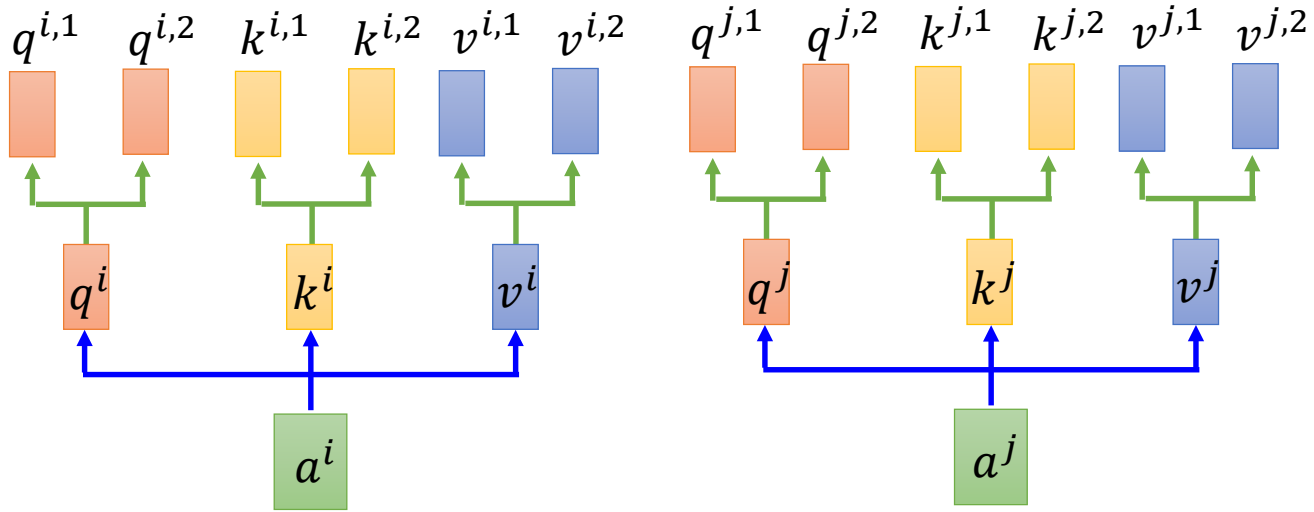
(2 heads as example)



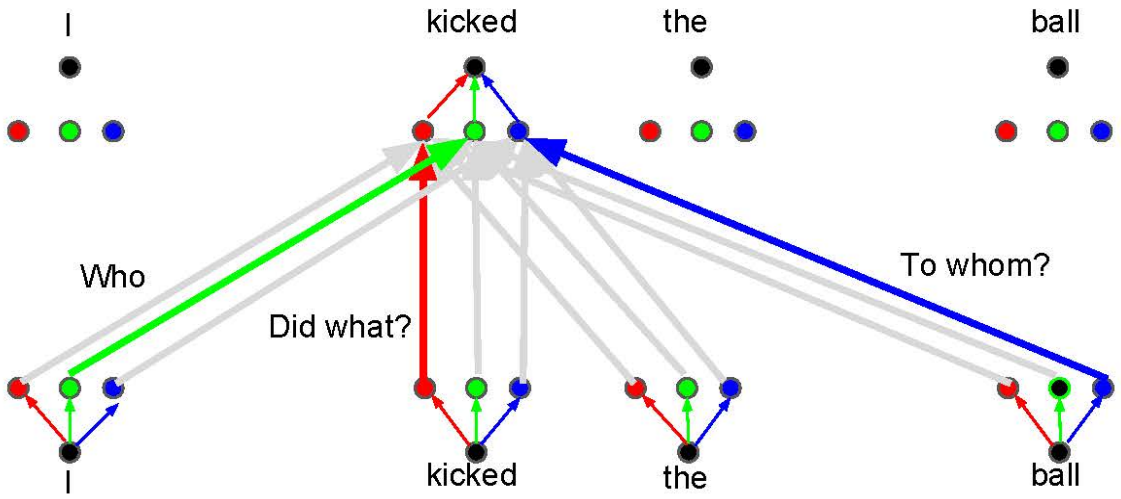
$q^i = W^q x^i$

Multi-head Self-attention

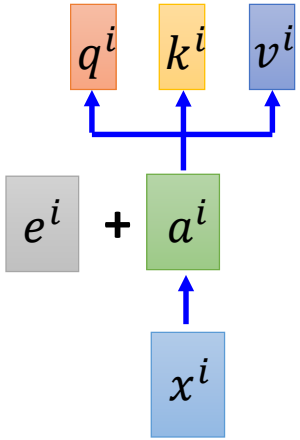




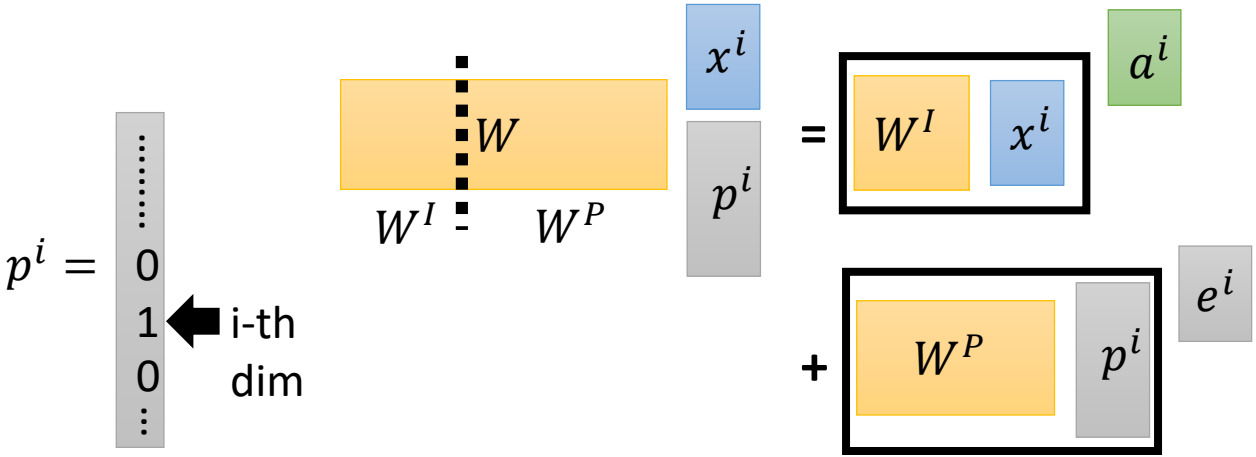
Parallel Attention Heads



Positional Encoding



- No position information in self-attention.
- Original paper: each position has a unique positional vector e^i (not learned from data)
- In other words: each x^i appends a one-hot vector p^i



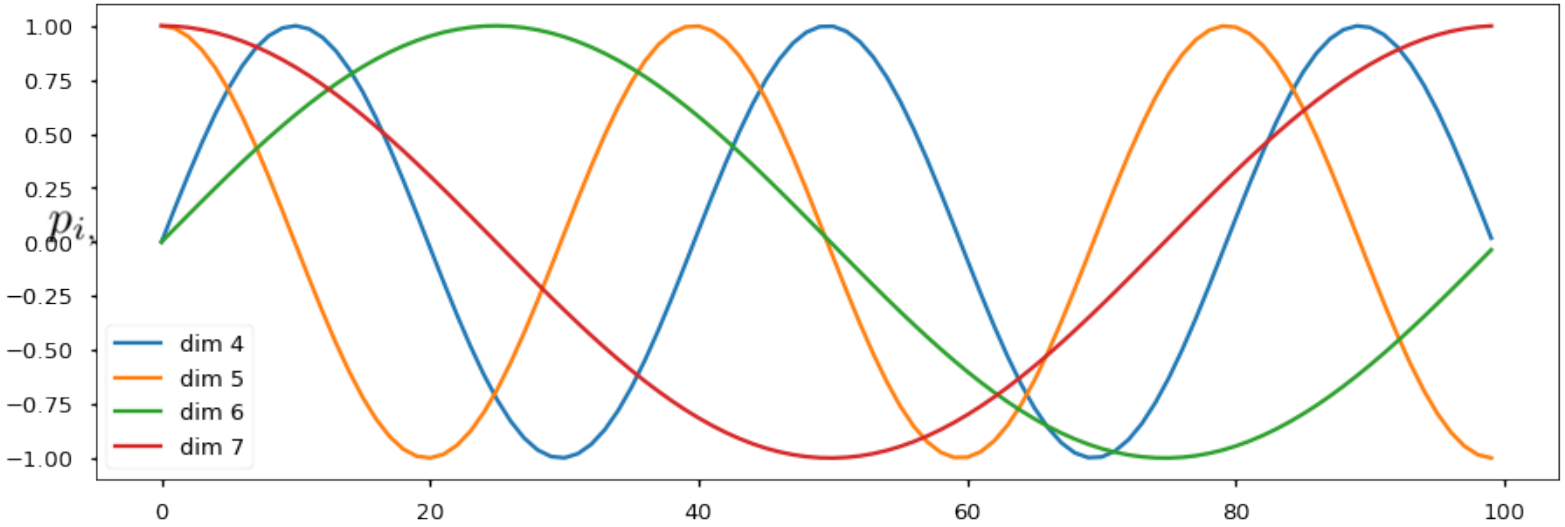
Positional Encoding

- BERT used learned positional embeddings
- Vaswani: add numbers between [-1,1] using predetermined (non-learned) sinusoidal functions to the token embeddings.
- Mathematically, using i for the position of the token in the sequence and *j* for the position of the embedding feature:

$$p_{i,j} = \begin{cases} \sin\left(\frac{i}{10000^{\frac{j}{d_{emb\_dim}}}}\right) & \text{if } j \text{ is even} \\ \cos\left(\frac{i}{10000^{\frac{j-1}{d_{emb\_dim}}}}\right) & \text{if } j \text{ is odd} \end{cases}$$

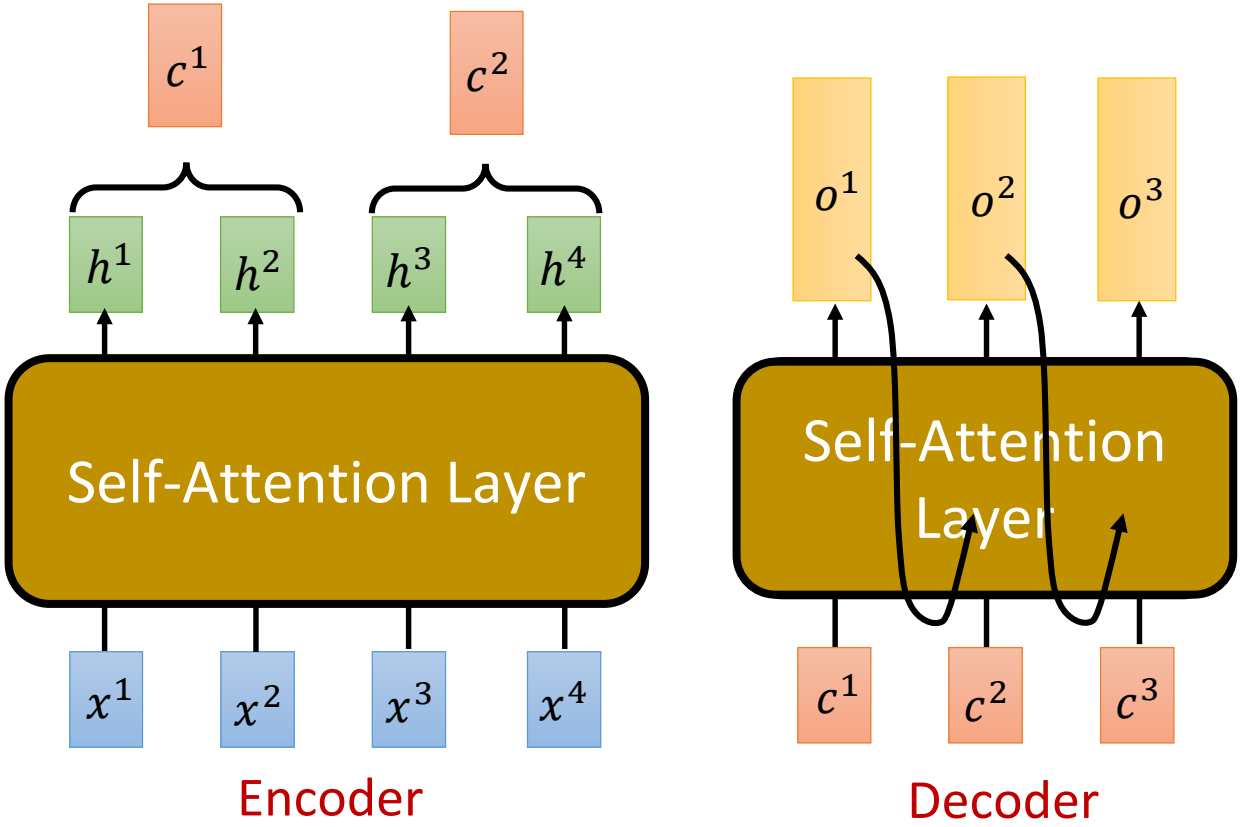
Positional Encoding

$$p_{i,j} = \begin{cases} \sin\left(\frac{i}{10000^{\frac{j}{d_{emb\_dim}}}}\right) & \text{if } j \text{ is even} \\ \cos\left(\frac{i}{10000^{\frac{j-1}{d_{emb\_dim}}}}\right) & \text{if } j \text{ is odd} \end{cases}$$



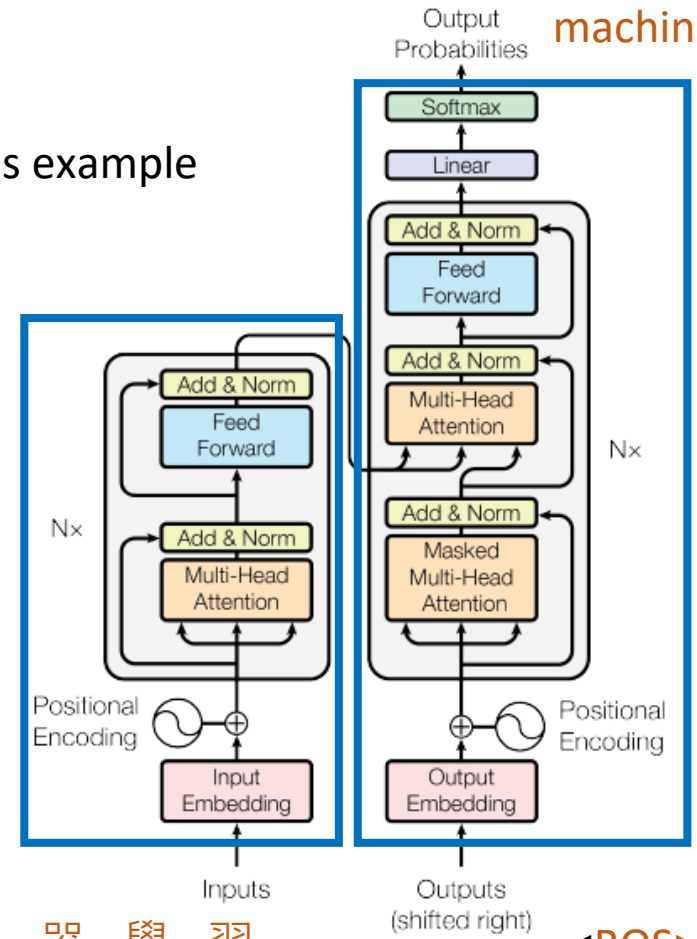
Review: https://www.youtube.com/watch?v=ZjfjPzXw6og&feature=youtu.be

Seq2seq with Attention



Transformer

Using machine translation as example



machine learning

Encoder

Decoder

機 器 學 習

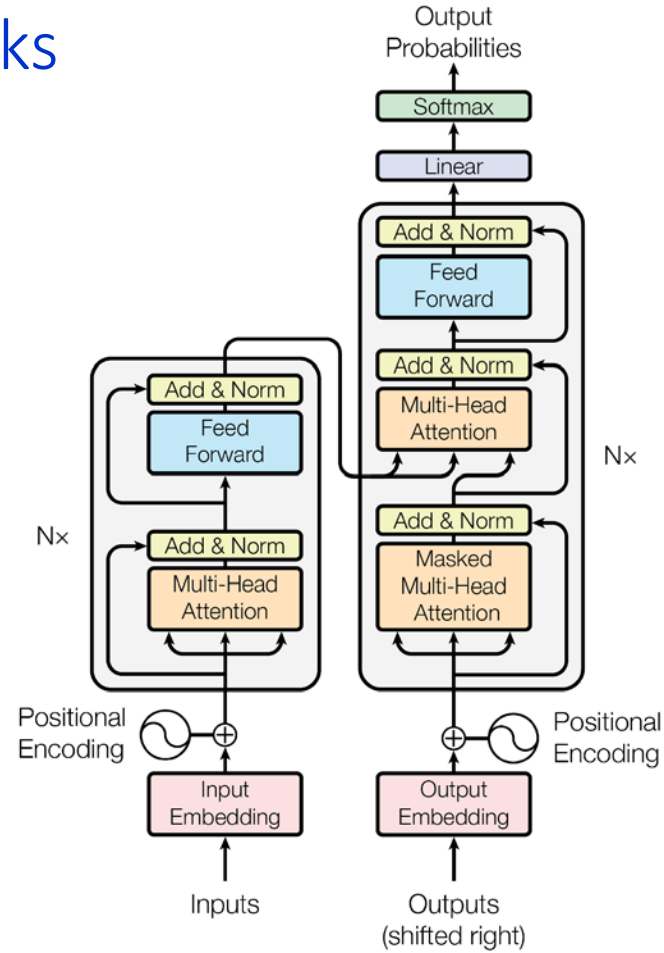
<BOS> machine

The Transformer Attention Tricks

- Self Attention
- Multi-headed Attention
- Normalized Dot-product Attention
- Positional Encodings

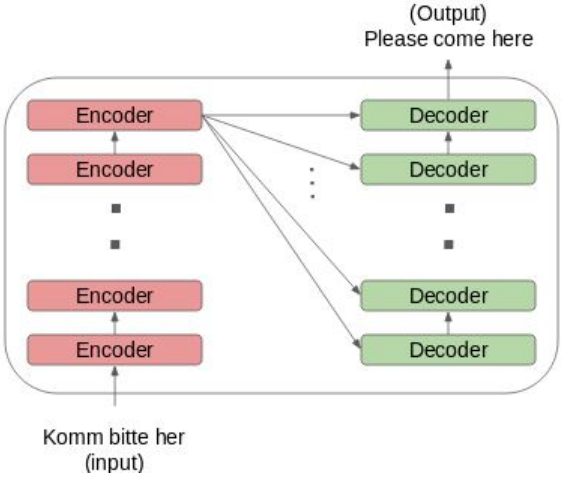
The Transformer Training Tricks

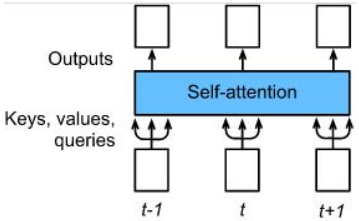
- Layer Normalization
- Label Smoothing
- Masking for Efficient Training



Transformers

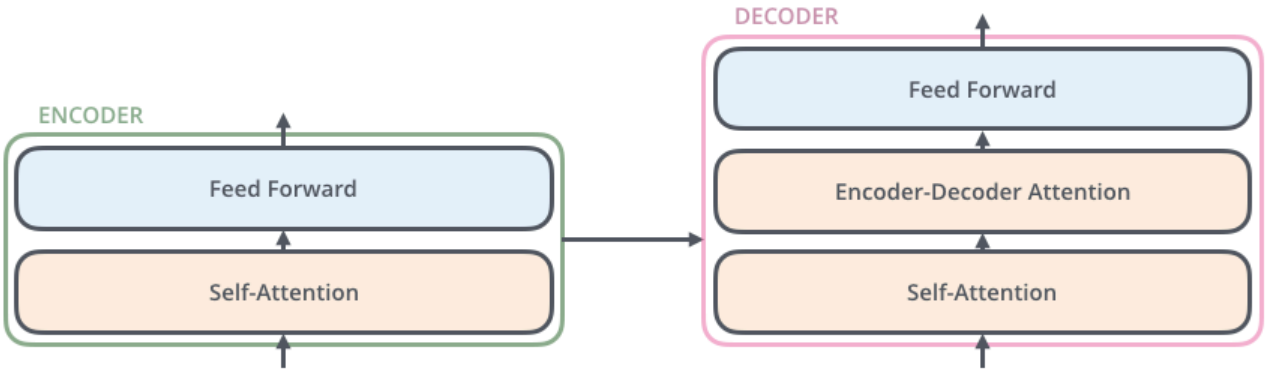
- The Transformer starts by generating initial representations for each word.
- Using self-attention, it aggregates information from all of the other words, generating a new representation per word
- This step is repeated multiple times in parallel for all words, successively generating new representations.





Attention in Transformer

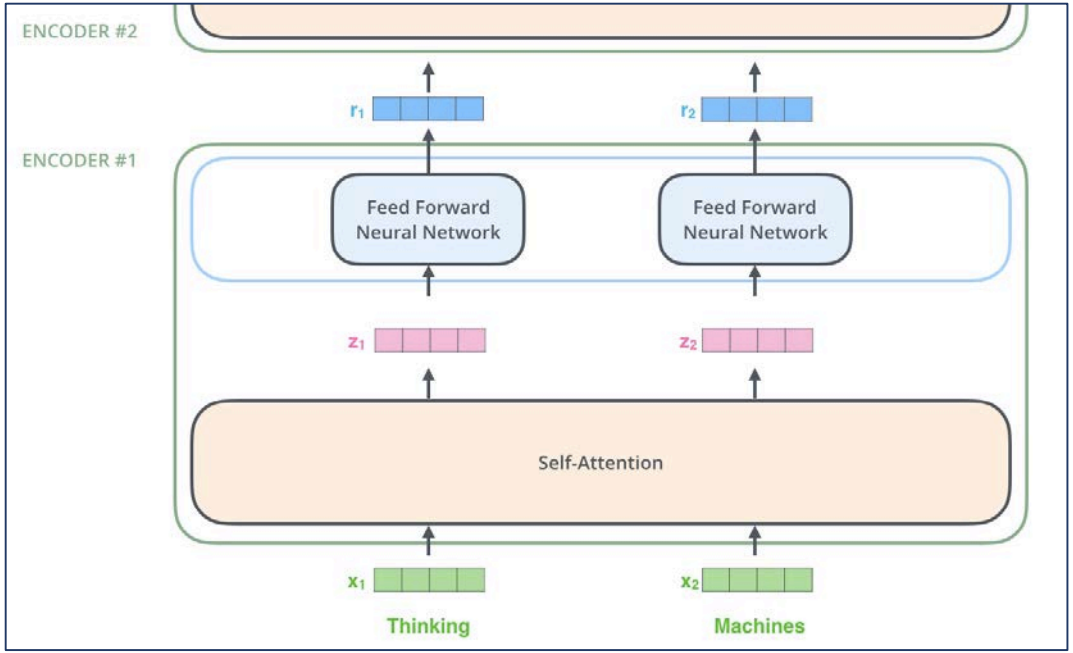
- The encoder's inputs first flow through a self-attention layer
- The outputs are fed to a feed-forward neural network. The exact same feed-forward network is independently applied to each position.
- The decoder has both those layers, but between them is an attention layer that helps the decoder focus on relevant parts of the input sentence



Encoder

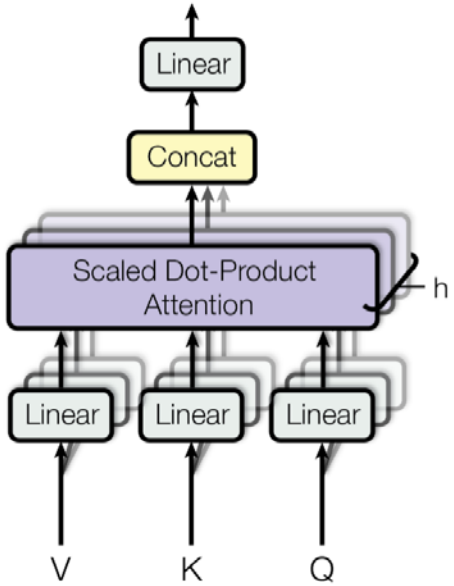
The word at each position passes through a self-attention process.

Then, they each pass through a feed-forward neural network -- the exact same network with each vector flowing through it separately.



Multi-headed attention

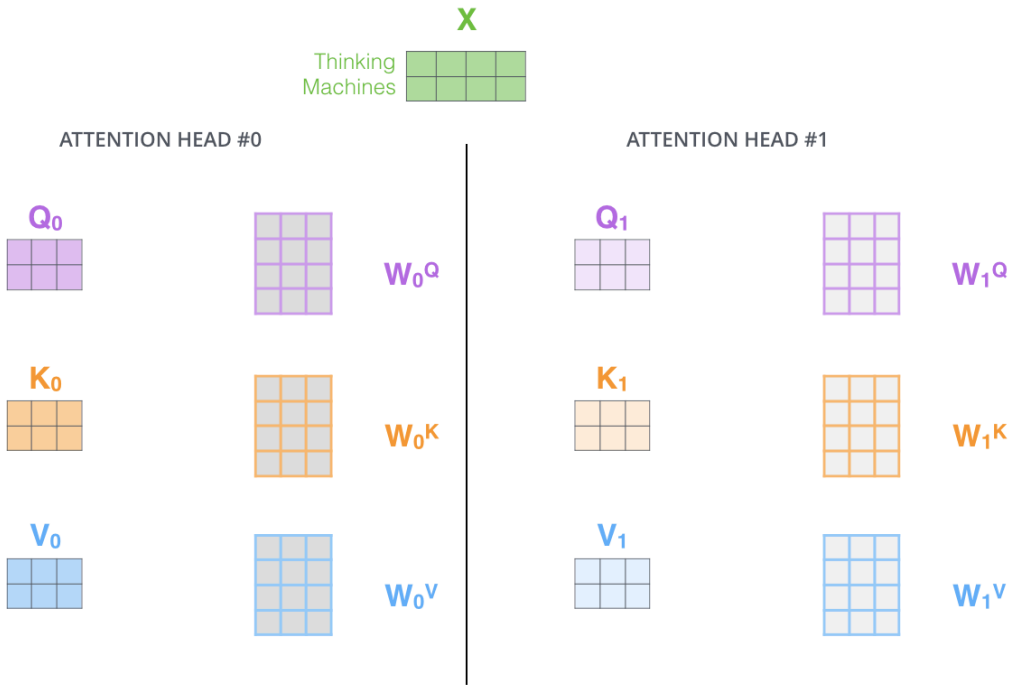
- Use multiple sets of Query/Key/Value weight matrices
- The Transformer uses eight attention heads
  - so there are eight sets for each encoder/decoder
  - Each of these sets is randomly initialized.
- After training, each set is used to project the input embeddings (or vectors from lower encoders/decoders) into a different representation subspace.

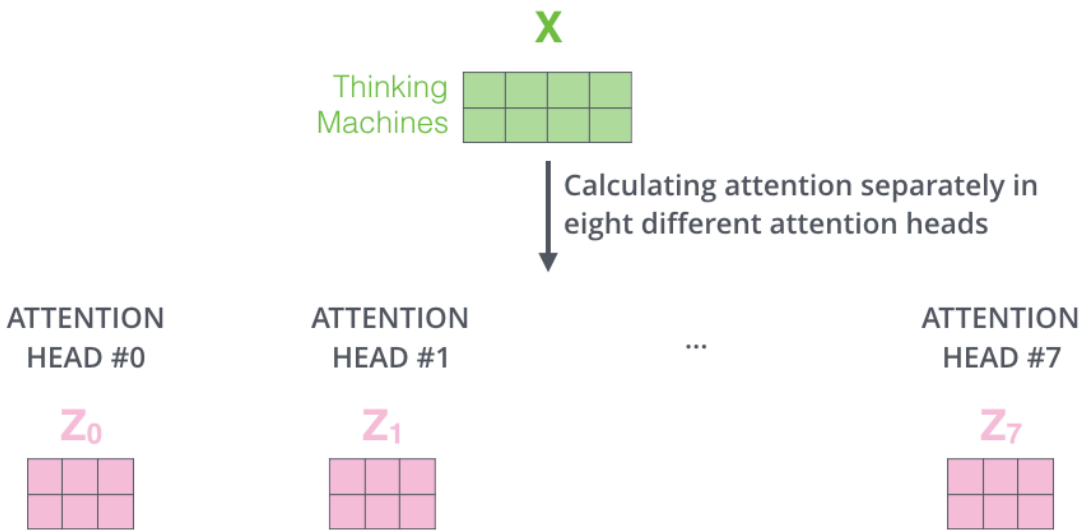


$$\text{MultiHead}\,(Q,K,V) = [head_1;\ldots;head_h]W^O$$

where $\text{head}_i = \text{Attention}(QW^Q{}_i, KW^K{}_i, VW^V{}_i)$

With multi-headed attention, maintain separate Q/K/V weight matrices for each head resulting in different Q/K/V matrices.





we need a way to condense these eight down into a single matrix as input to the feedforward network.

Putting it all together

1) This is our input sentence*

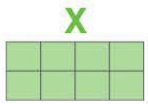
2) We embed each word*

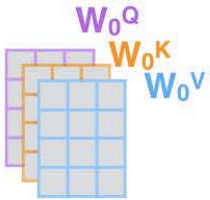
3) Split into 8 heads. We multiply X or R with weight matrices

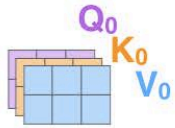
4) Calculate attention using the resulting Q/K/V matrices

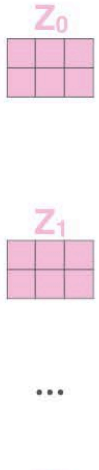
5) Concatenate the resulting Z matrices, then multiply with weight matrix W^O to produce the output of the layer

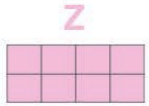
Thinking Machines







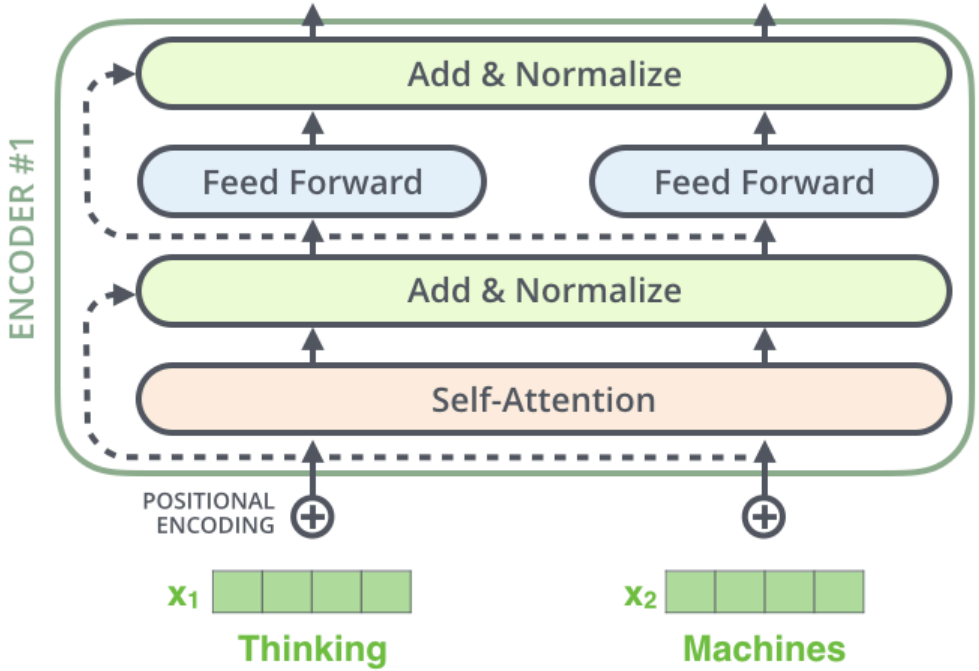


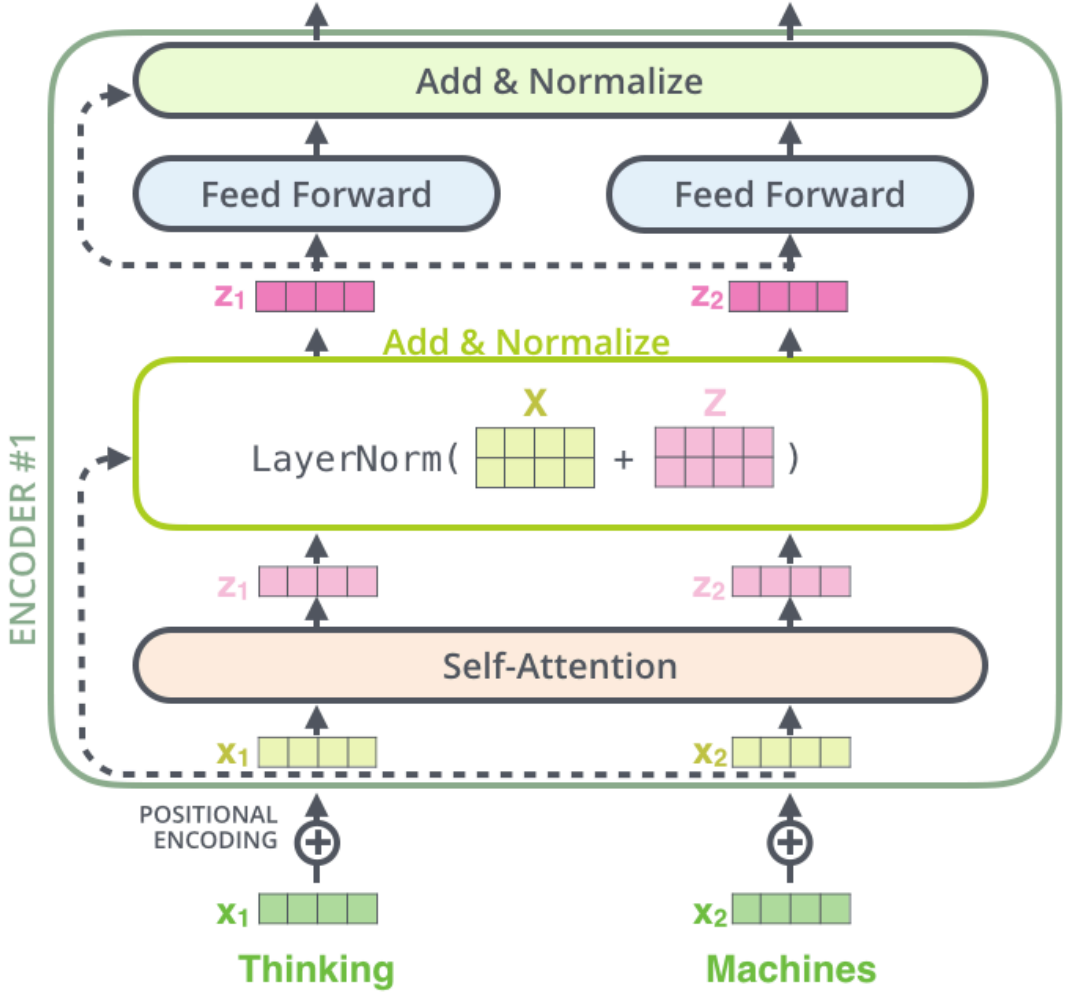


* In all encoders other than #0, we don't need embedding. We start directly with the output of the encoder right below this one

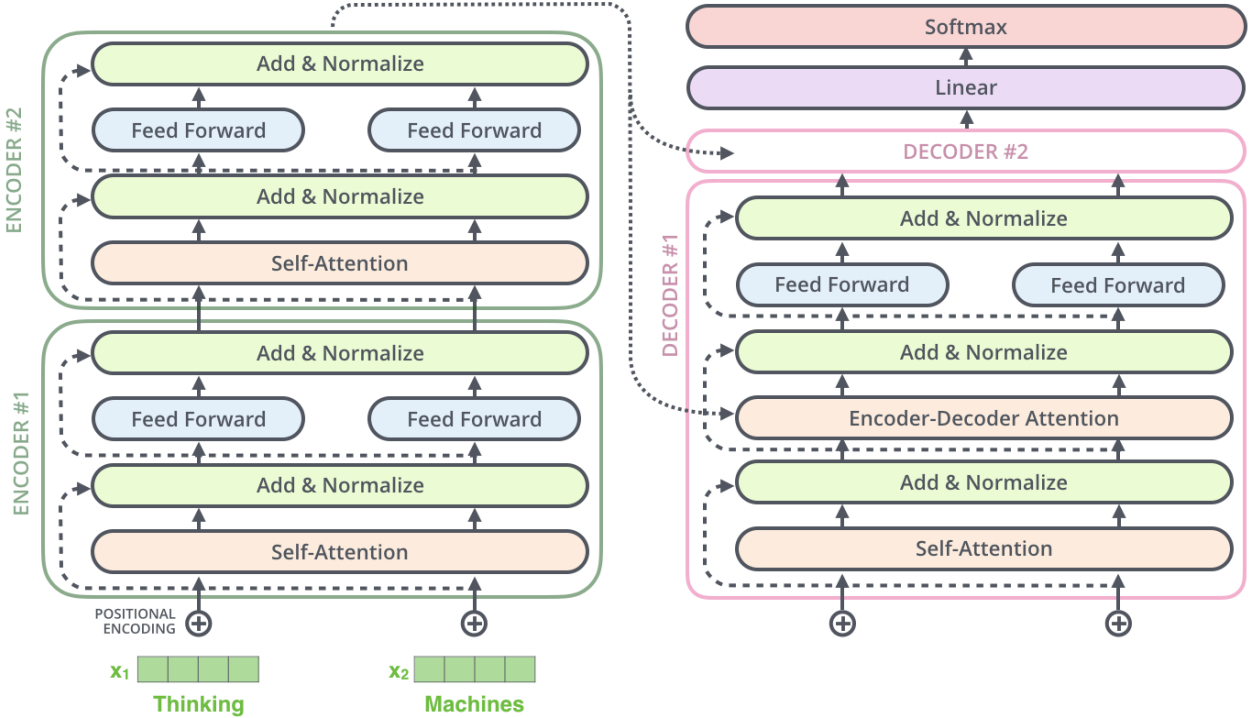
The Residuals

each sub-layer (self-attention, ffnn) in each encoder has a residual connection around it, and is followed by a layer-normalization step.




Add & Normalize
Feed Forward
Feed Forward
z1
z2
Add & Normalize
X
Z
LayerNorm(+)
z1
z2
Self-Attention
x1
x2
ENCODER #1
POSITIONAL ENCODING
x1
x2
Thinking
Machines

Stacked Encoder

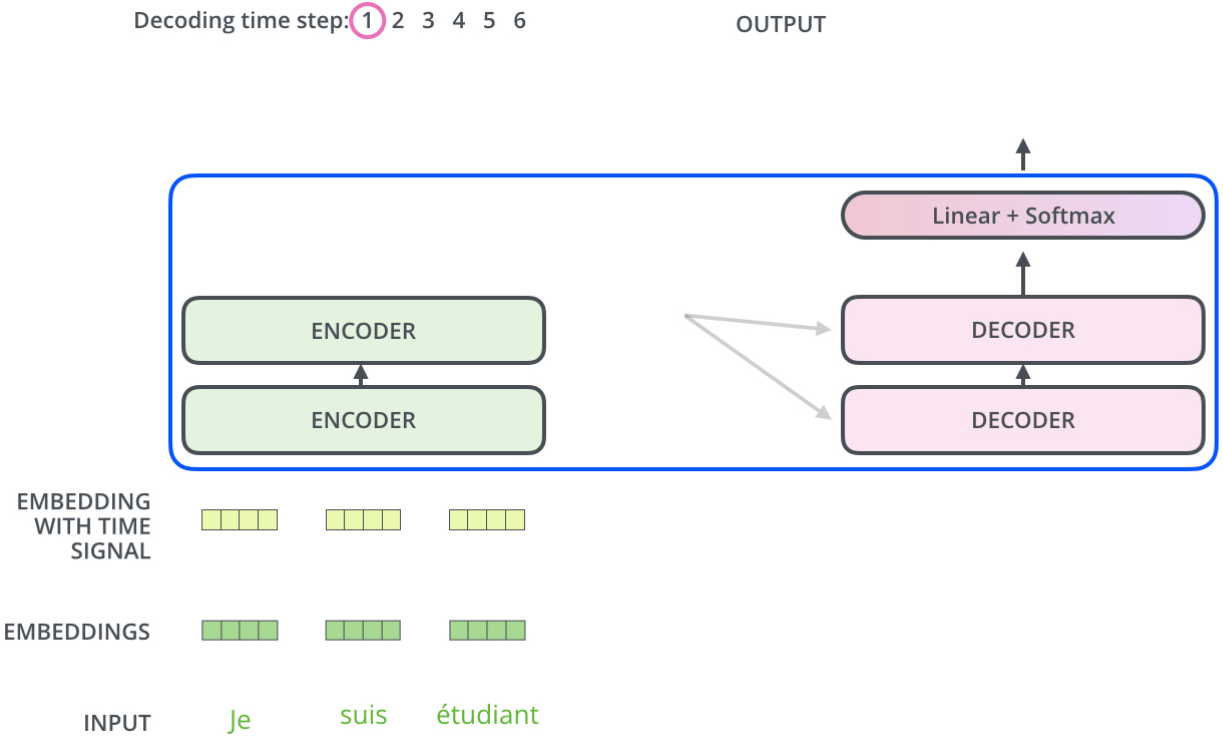


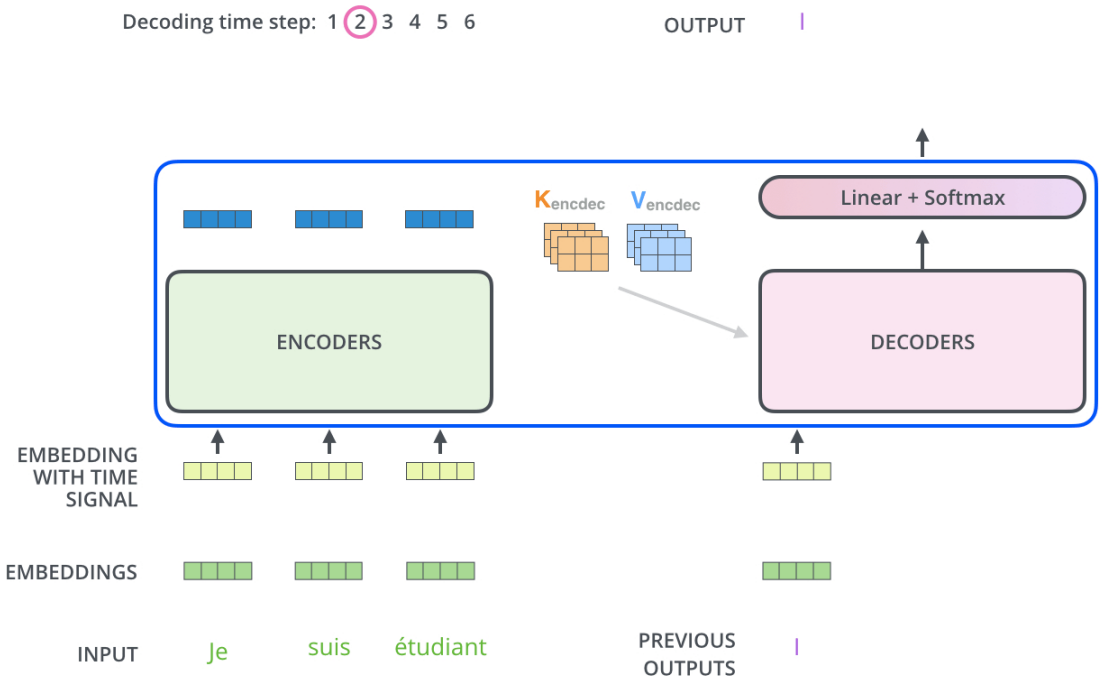
Decoder Side

The encoder start by processing the input sequence.

The output of the top encoder is then transformed into a set of attention vectors K and V.

These are to be used by each decoder in its “encoder-decoder attention” layer which helps the decoder focus on appropriate places in the input sequence:




Decoding time step: 1 2 3 4 5 6
OUTPUT
I
Linear + Softmax
Kencdec
Vencdec
ENCODERS
DECODERS
EMBEDDING WITH TIME SIGNAL
EMBEDDINGS
INPUT
Je
suis
étudiant
PREVIOUS OUTPUTS
I

Decoder Attention

- In the decoder, the self-attention layer is only allowed to attend to earlier positions in the output sequence. This is done by masking future positions (setting them to -inf) before the softmax step in the self-attention calculation.

- The “Encoder-Decoder Attention” layer works just like multiheaded self-attention, except it creates its Queries matrix from the layer below it, and takes the Keys and Values matrix from the output of the encoder stack.

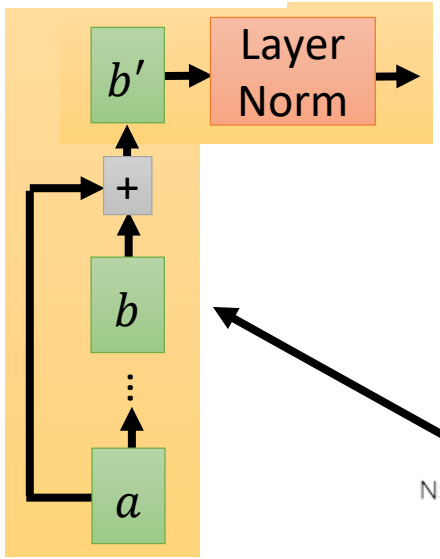
The Final Linear and Softmax Layer

- A Softmax Layer to output word

- Let’s assume that our model knows 10,000 unique English words (our model’s “output vocabulary”) that it’s learned from its training dataset.

- The softmax layer then turns those scores into probabilities (all positive, all add up to 1.0). The cell with the highest probability is chosen, and the word associated with it is produced as the output for this time step.

Transformer

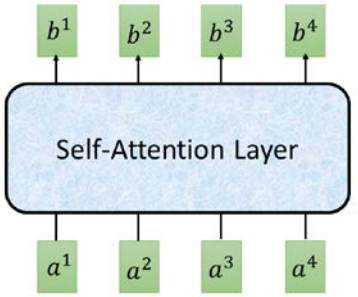


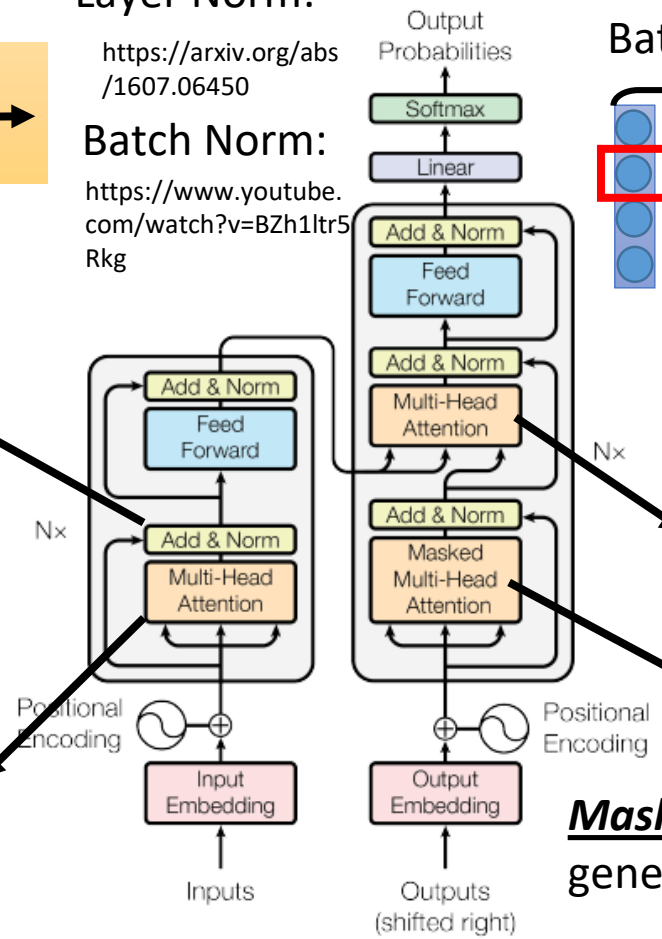
Layer Norm:

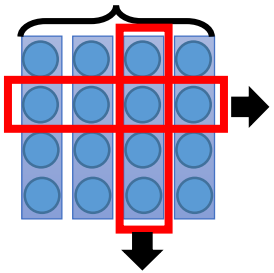
https://arxiv.org/abs/1607.06450

Batch Norm:

https://www.youtube.com/watch?v=BZh1ltr5Rkg



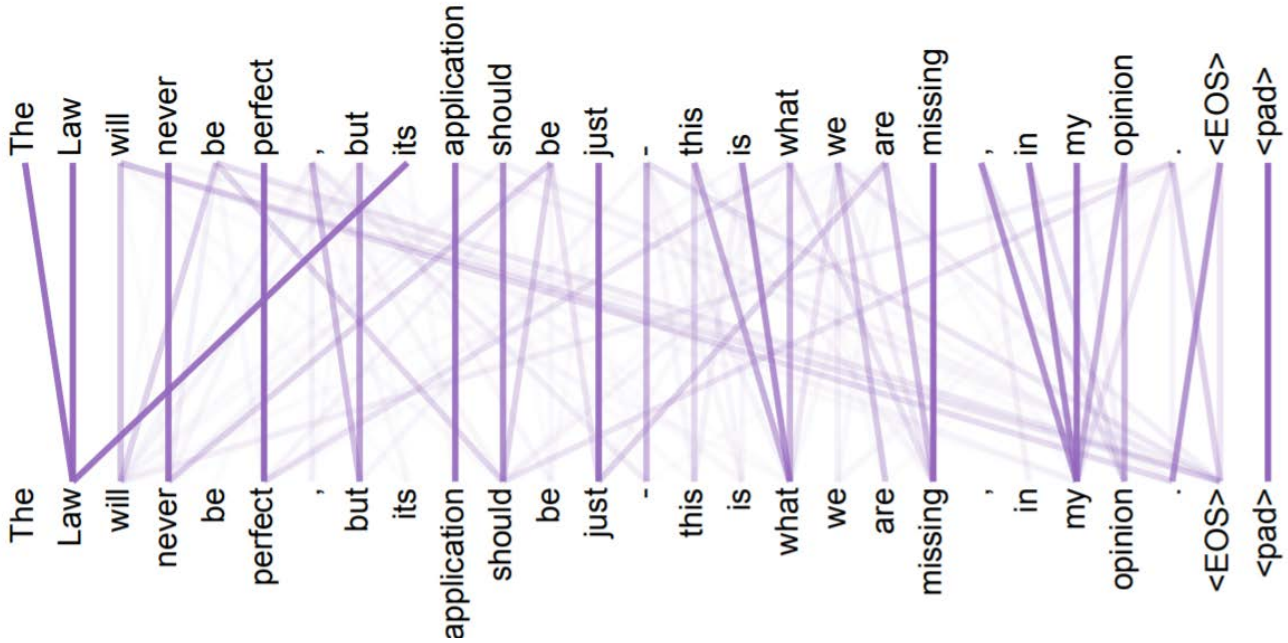




attend on the input sequence

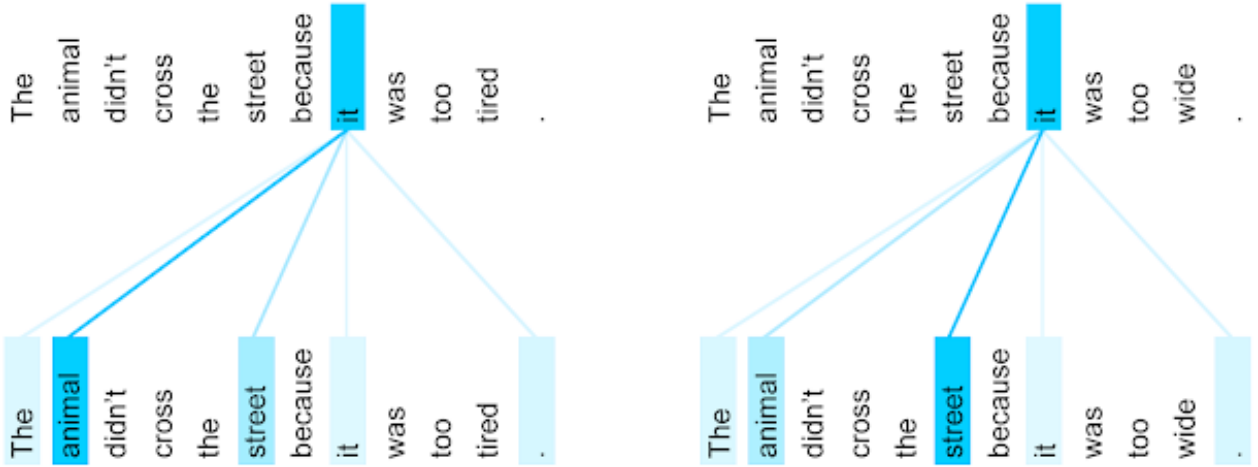
Masked: attend on the generated sequence

Attention Visualization



https://arxiv.org/abs/1706.03762

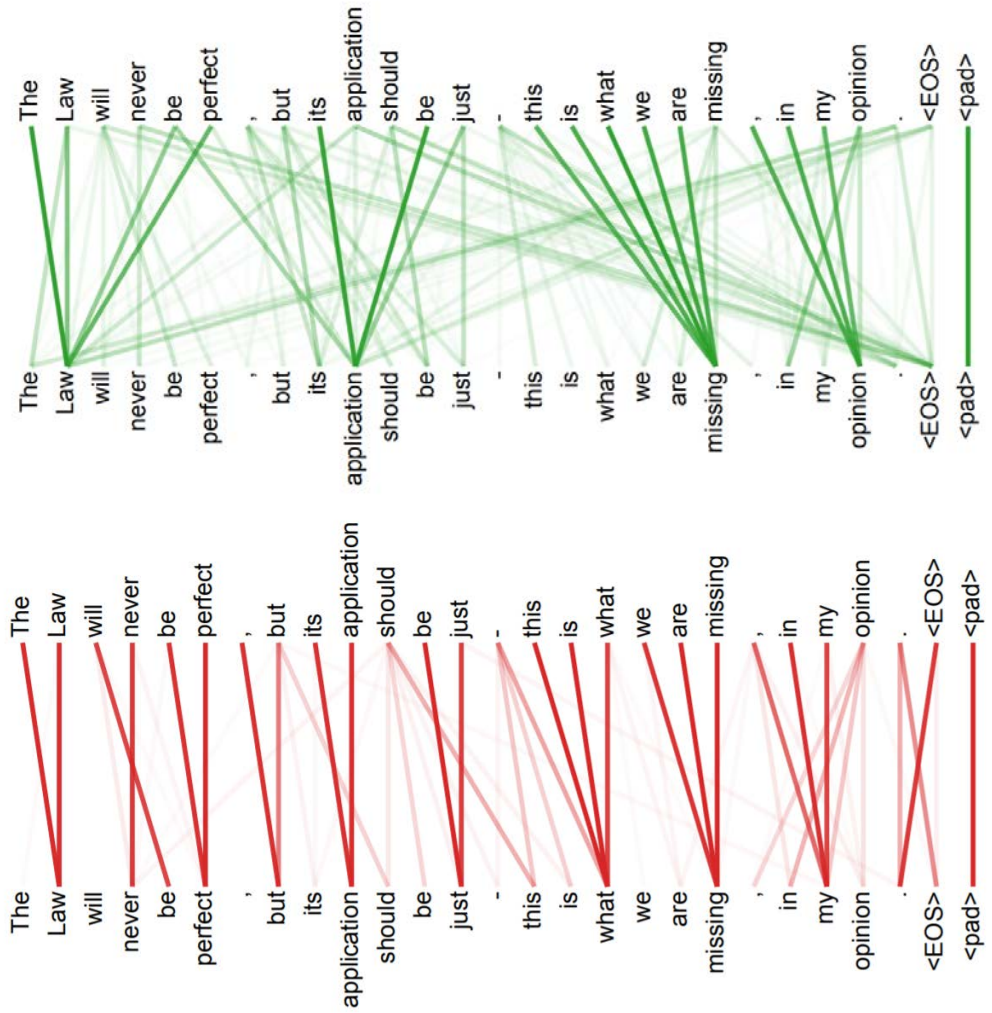
Attention Visualization



The encoder self-attention distribution for the word “it” from the 5th to the 6th layer of a Transformer trained on English to French translation (one of eight attention heads).

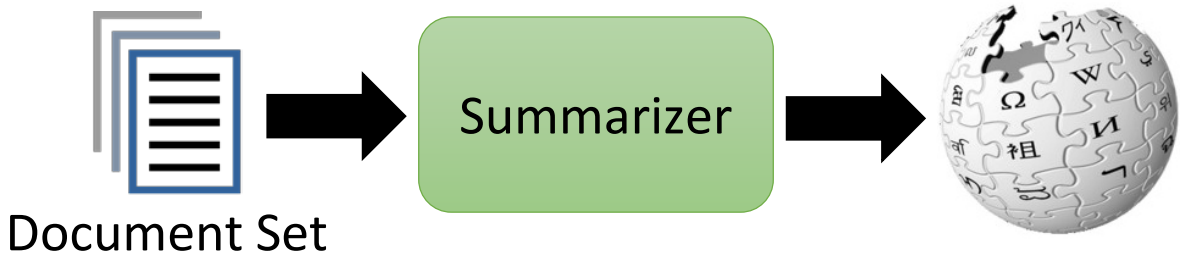
https://ai.googleblog.com/2017/08/transformer-novel-neural-network.html

Multi-head Attention



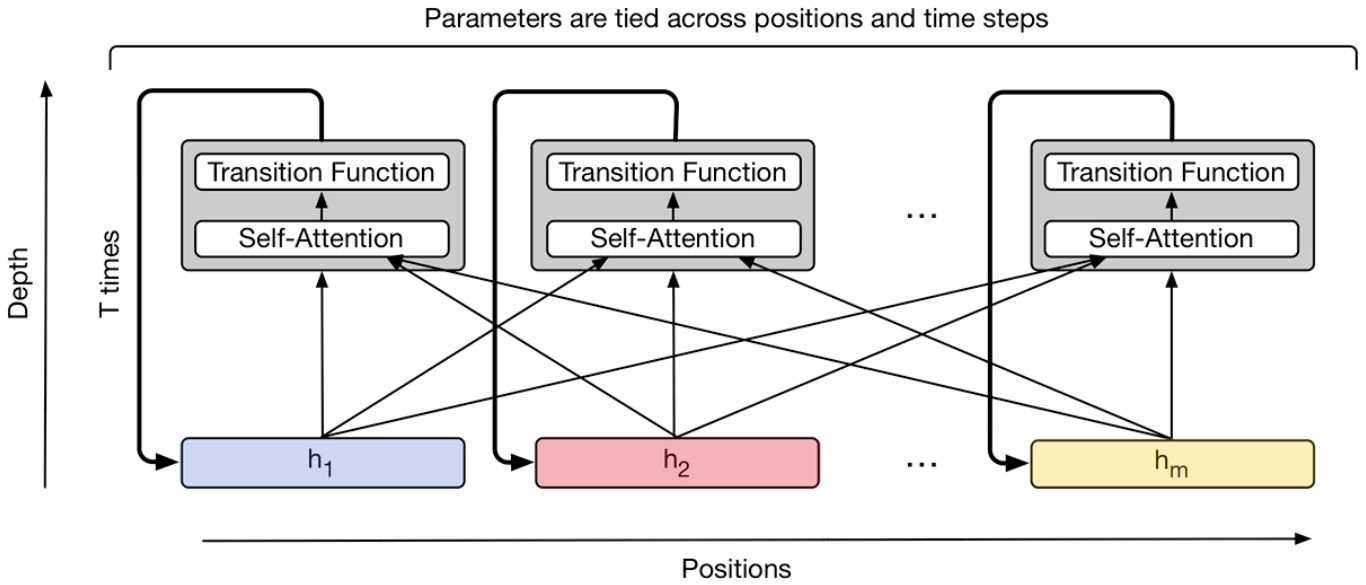
Example Application

- If you can use seq2seq, you can use transformer.



https://arxiv.org/abs/1801.10198

Universal Transformer



https://ai.googleblog.com/2018/08/moving-beyond-translation-with.html