



CS60010: Deep Learning

Spring 2023

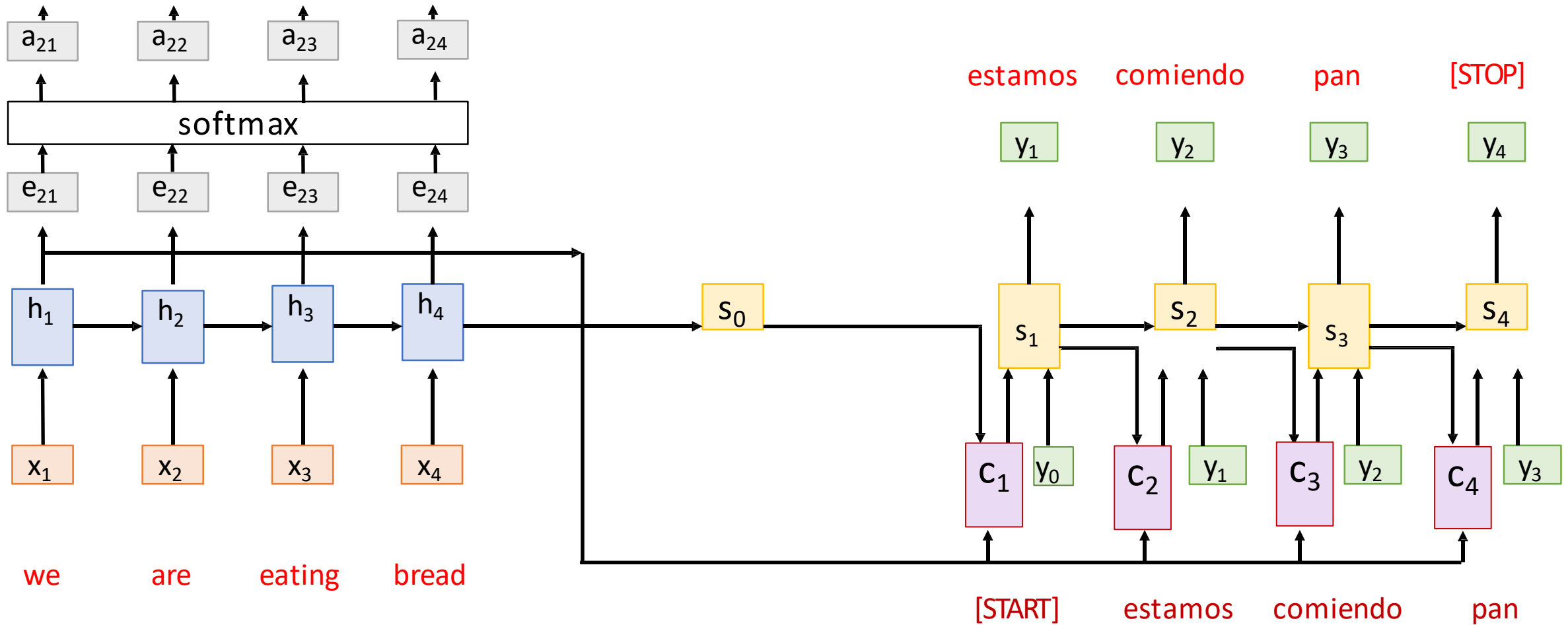
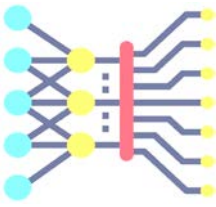
Sudeshna Sarkar

Transformer- Part 1

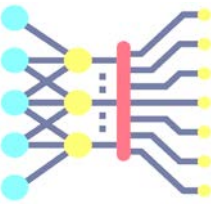
Sudeshna Sarkar

10 Mar 2023

Encoder Decoder Attention

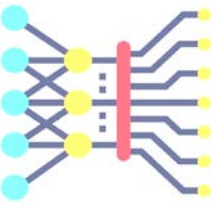


Attention-only Translation Models



- 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



- Consider an input (or intermediate) sequence
- Construct the next level representation:
which can choose “where to look”, by assigning a weight to each input position.
- Create a weighted sum.

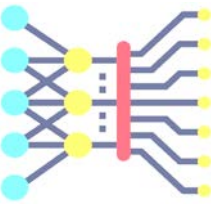
● Level k+1

Softmax over lower
locations conditioned
on context at lower and
higher locations

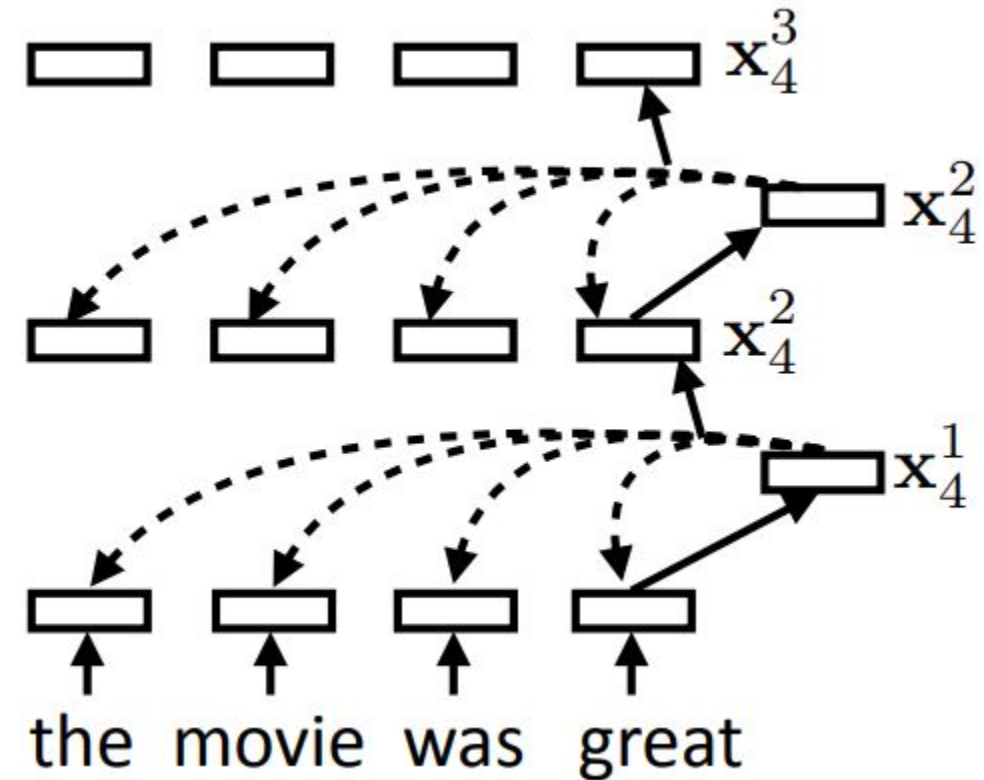


Level k

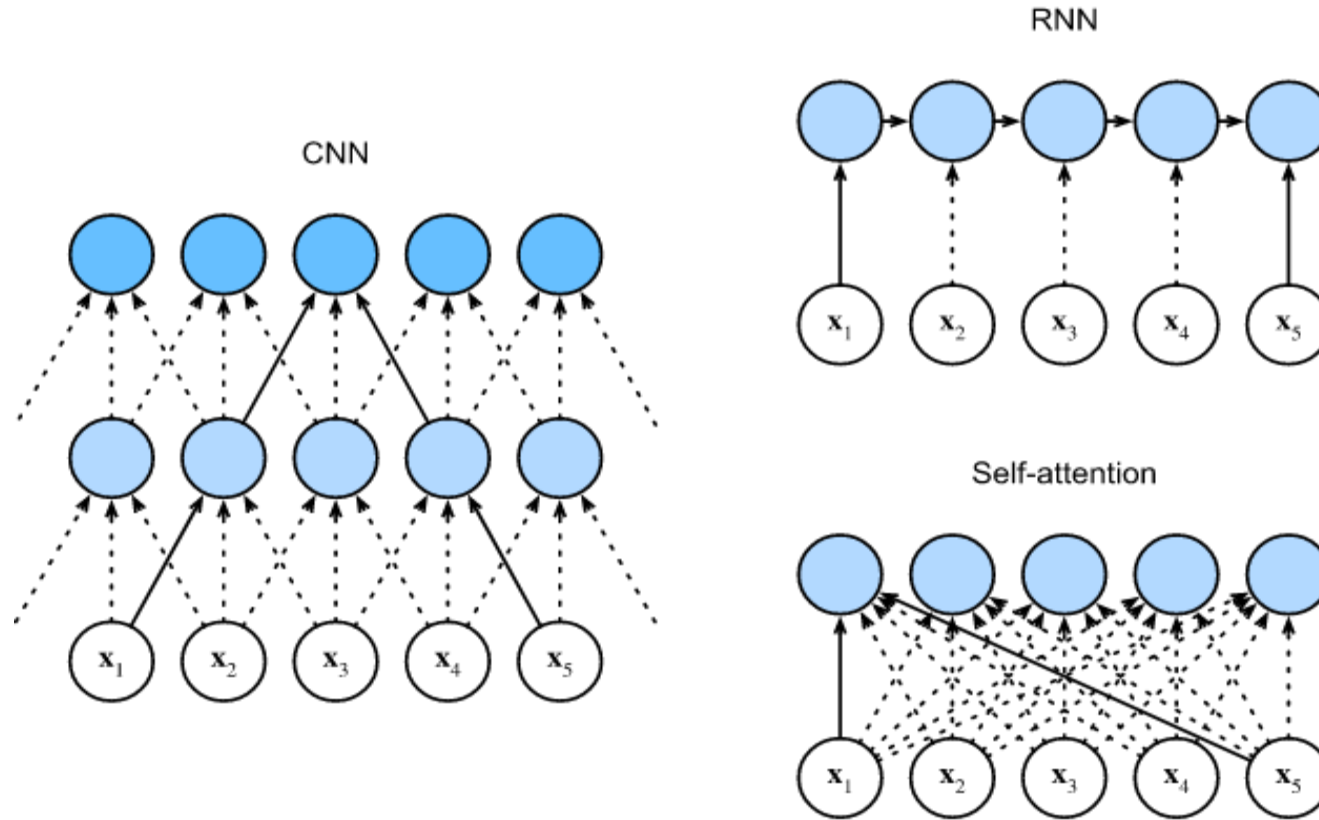
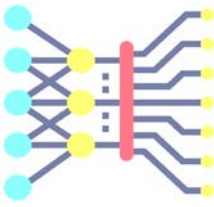
Self-attention

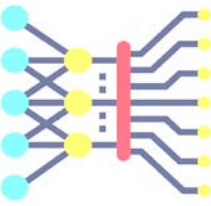


- Each word is a query to form attention over all tokens
- This generates a context-dependent representation of each token: a weighted sum of all tokens
- The attention weights dynamically mix how much is taken from each token



Comparing CNNs, RNNs, and 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.
- If we have attention, do we even need recurrent connections?
- Can we transform our RNN into a purely attention-based model?

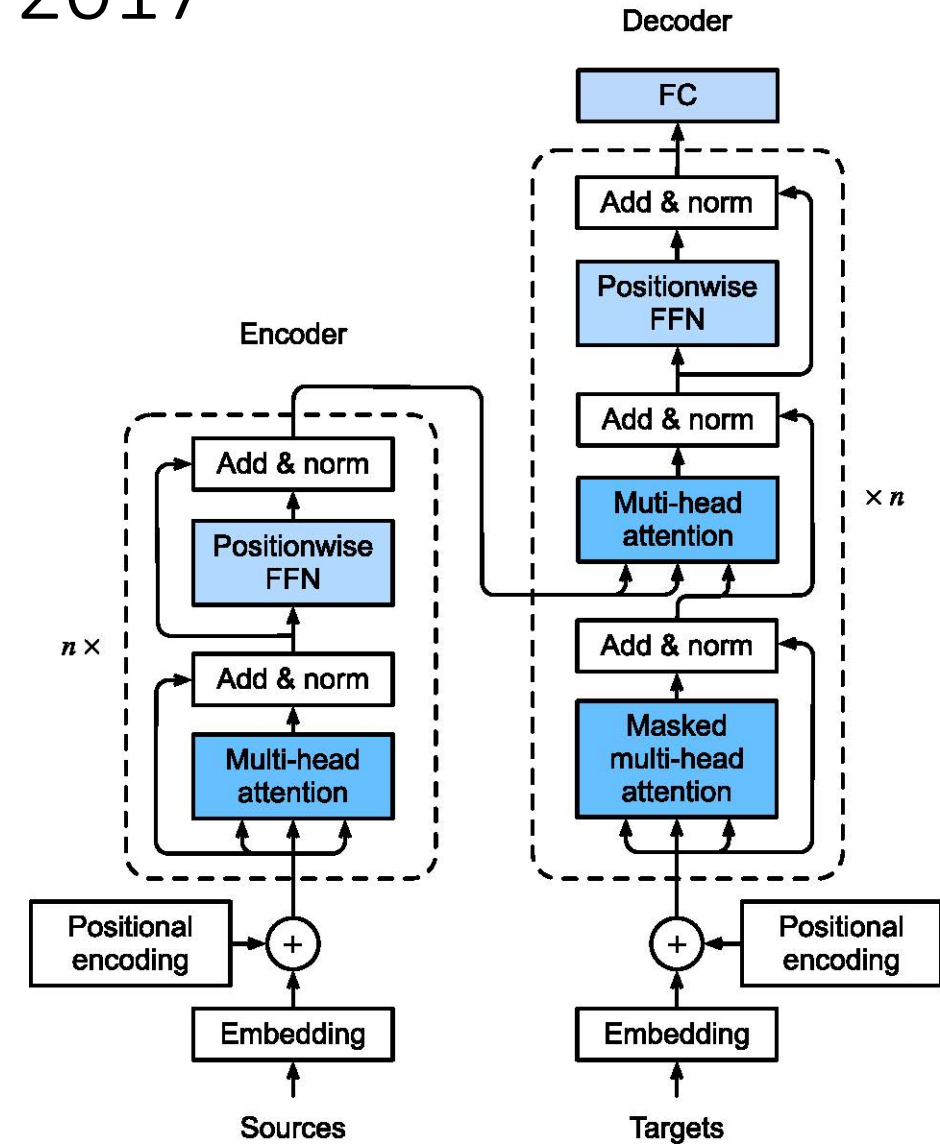
Vaswani et al. “Attention is all you need”, arXiv 2017

<https://arxiv.org/abs/1706.03762>



Transformer Model Vaswani et al. 2017

- Transformers map sequences of input vectors (x_1, x_2, \dots, x_n) to sequences of output vectors (y_1, y_2, \dots, y_m) .
- Made up of stacks of Transformer blocks.
 - combine linear layers, feedforward networks, and self-attention layers.
- **Self-attention** allows a network to directly extract and use information from arbitrarily large contexts



Transformer Model

Queries, Keys, and Values

- For each element in the sequence S (length m) define key, value and query

$$Attention(q, S) \stackrel{def}{=} \sum_{i=1}^m \alpha(q, k_i) v_i$$

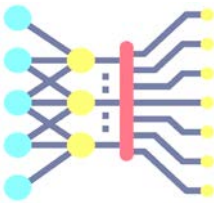
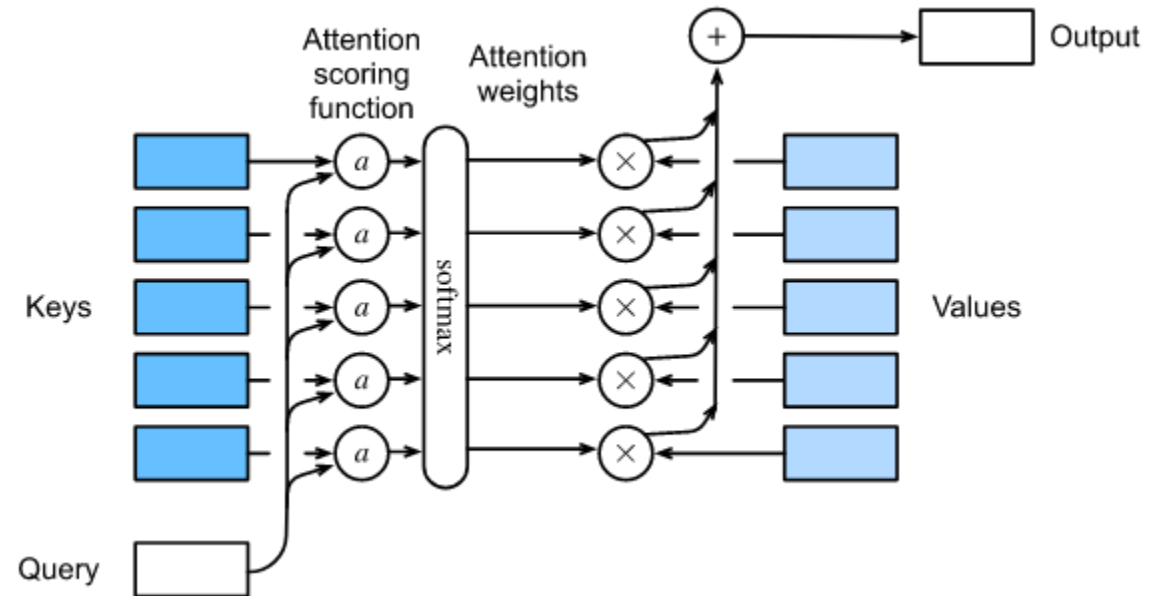
- Normalize attention weights to sum to 1 and be non-negative.

$$\alpha(q, k_i) = \frac{\exp(\alpha(q, k_i))}{\sum_j \exp(\alpha(q, k_j))}$$

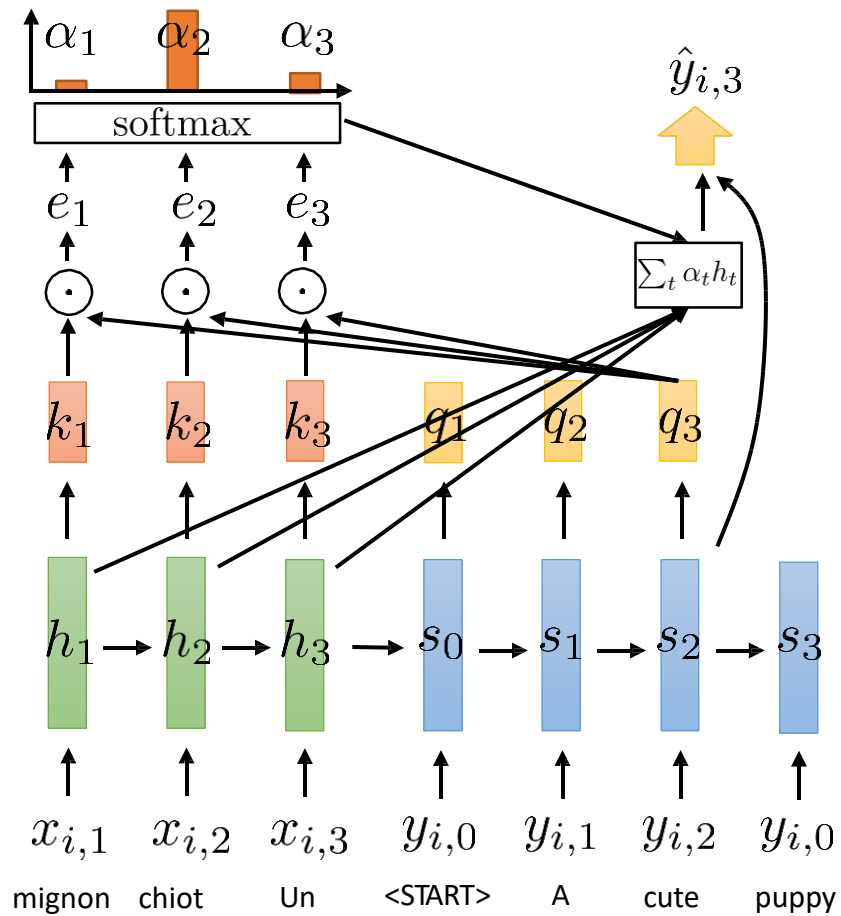
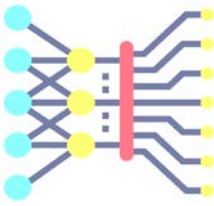
Dot Product Attention

$$\alpha(q, k_i) = \frac{q^T k_i}{\sqrt{d}}$$

$$\alpha(q, k_i) = \text{softmax}(\alpha(q, k_i))$$
$$= \frac{\exp(q^T k_i / \sqrt{d})}{\sum_j \exp(q^T k_j / \sqrt{d})}$$



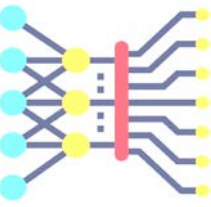
Is Attention all we need?



Attention can in principle do **everything** that recurrence can, and more!

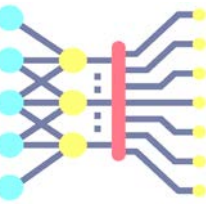
This has a few issues we must overcome:

1. The encoder has no temporal dependencies.



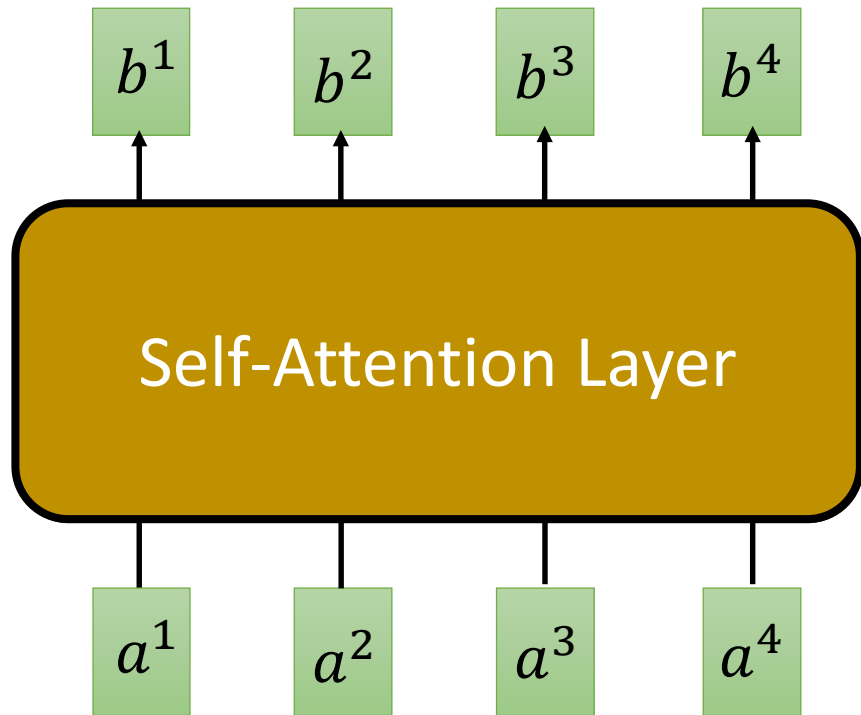
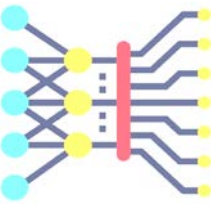
From Self-Attention to Transformers

1. **Positional encoding** addresses lack of sequence information
2. **Multi-headed attention** allows querying multiple positions at each layer
3. **Adding nonlinearities**
4. **Masked decoding** how to prevent attention lookups into the future?



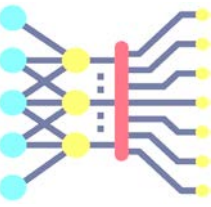
Self-Attention explained

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.



q : query (to match others)

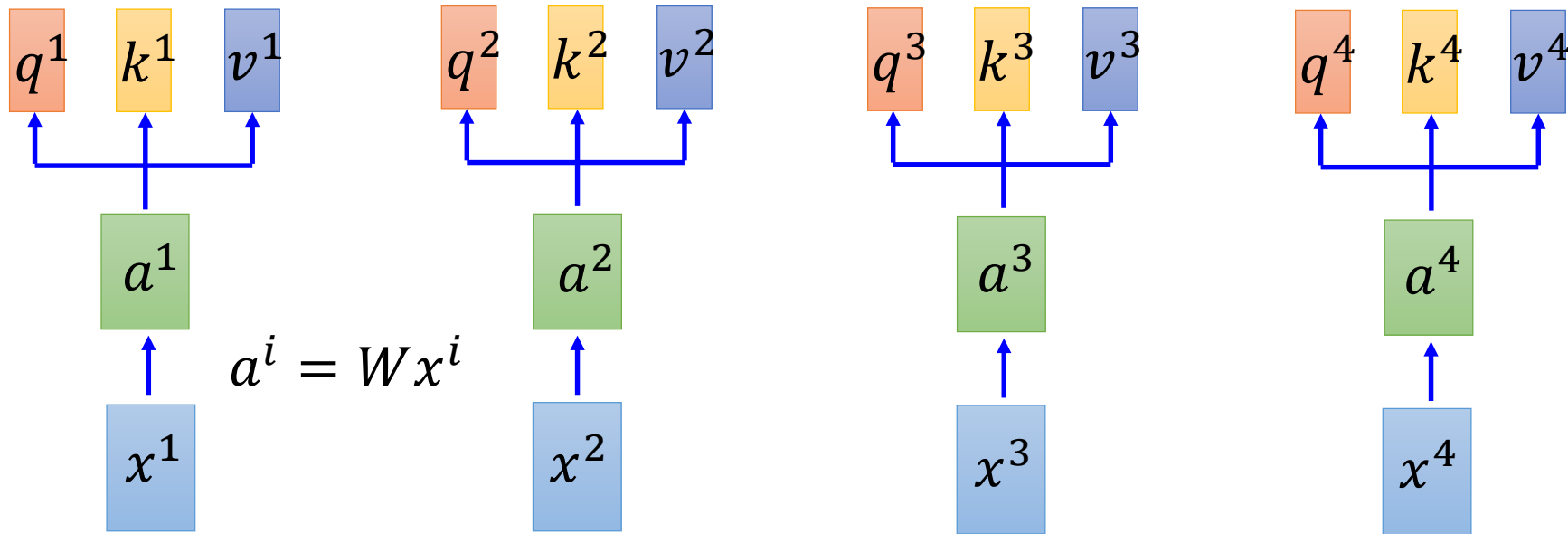
$$q^i = W^q a^i$$

k : key (to be matched)

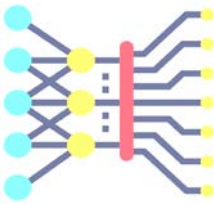
$$k^i = W^k a^i$$

v : information to be extracted

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



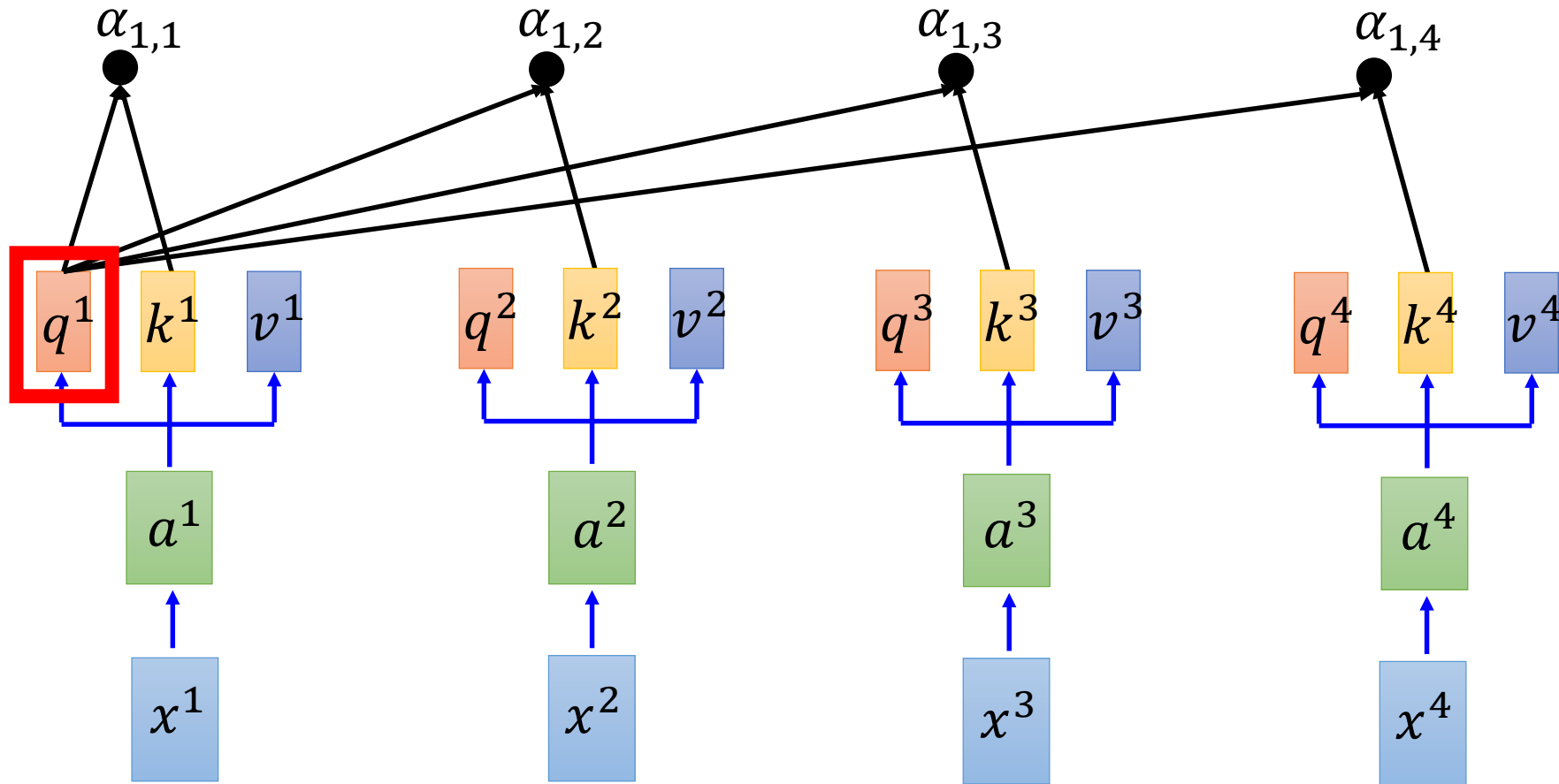
Scaled Dot-Product Attention



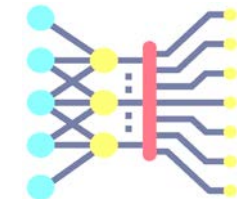
d is the dim of q and k

$$\alpha_{1,i} = \underbrace{q^1 \cdot k^i}_{\text{dot product}} / \sqrt{d}$$

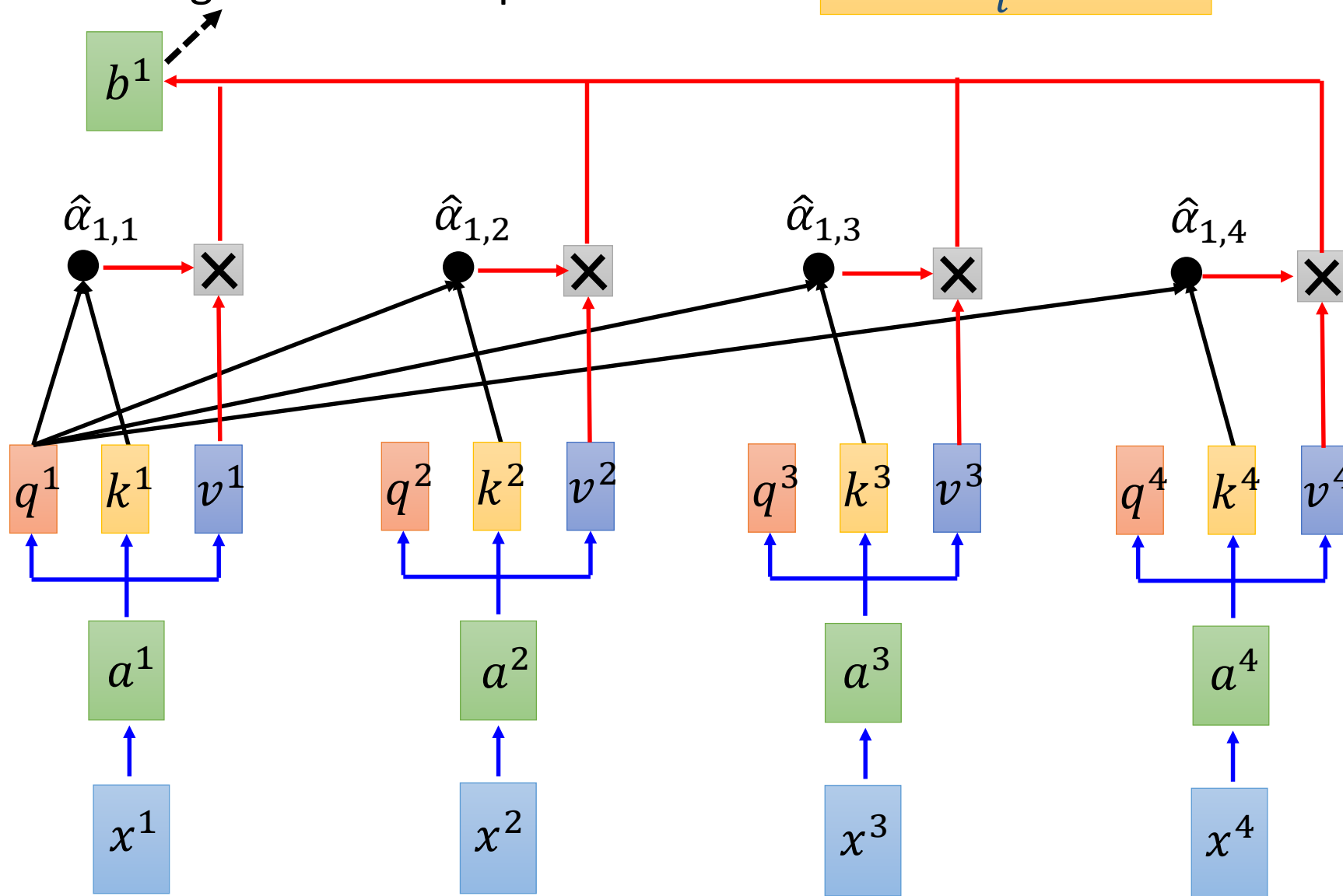
dot product



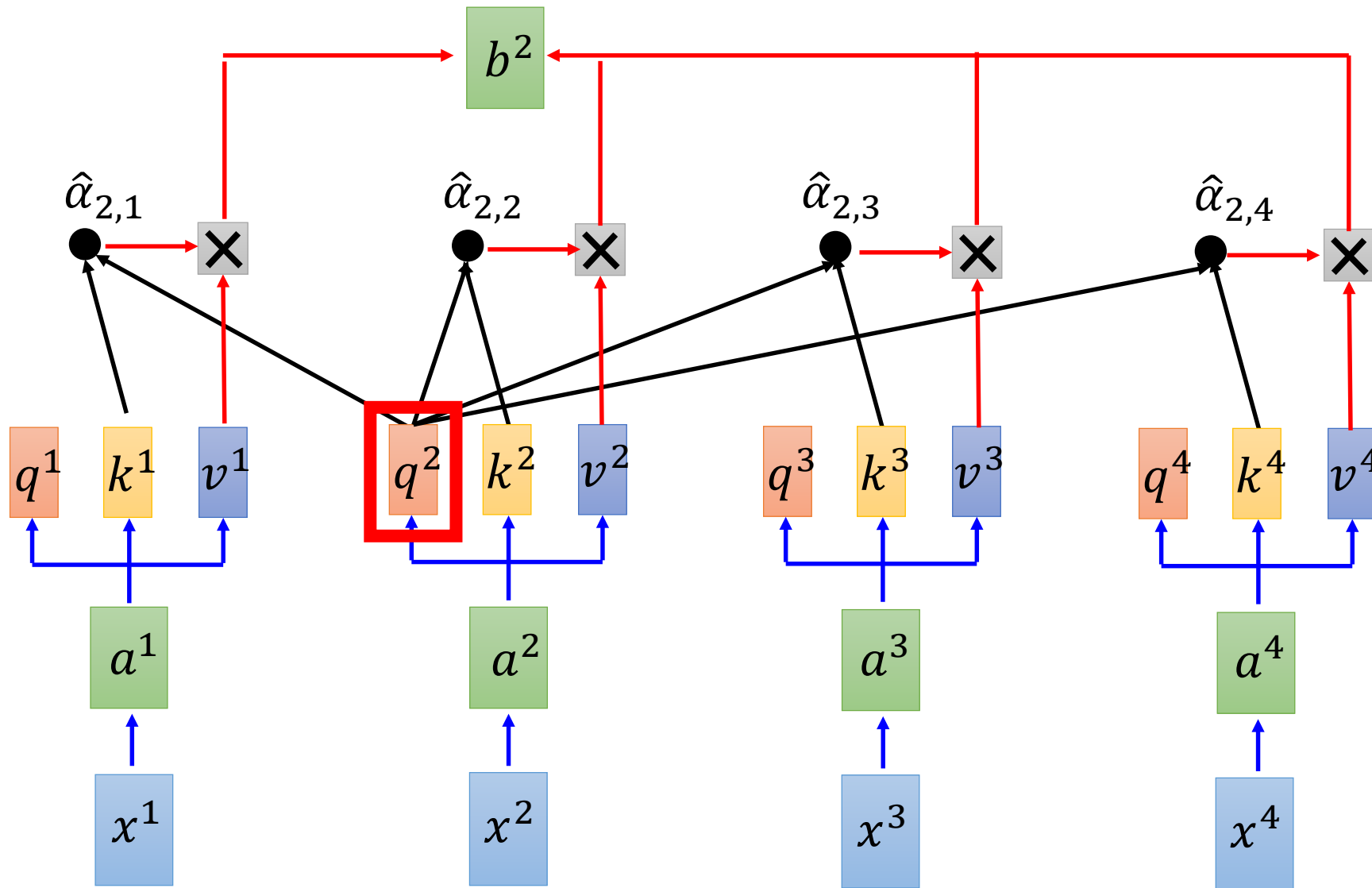
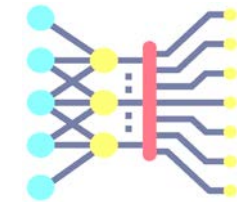
$$b^1 = \sum_i \hat{\alpha}_{1,i} v^i$$

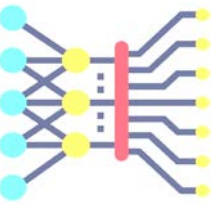


Considering the whole sequence

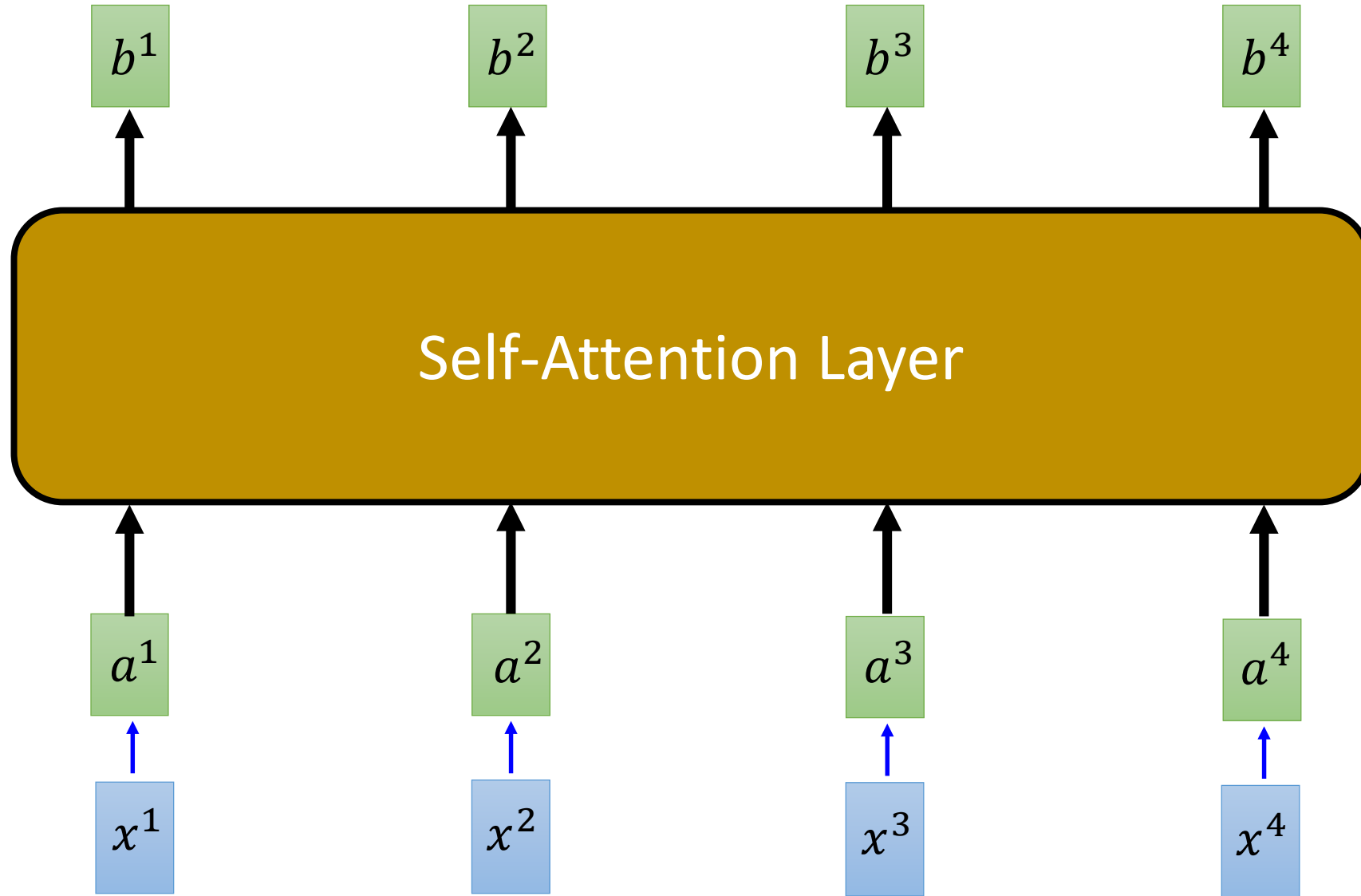


$$b^2 = \sum_i \hat{\alpha}_{2,i} v^i$$





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



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

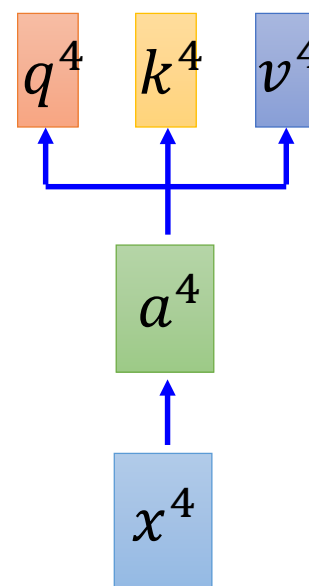
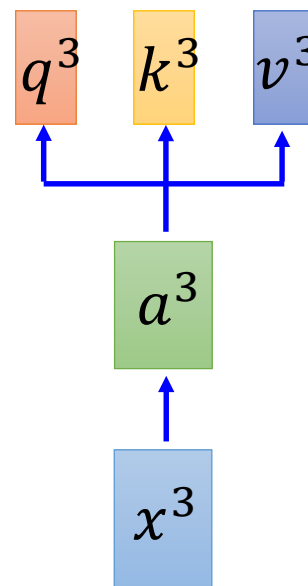
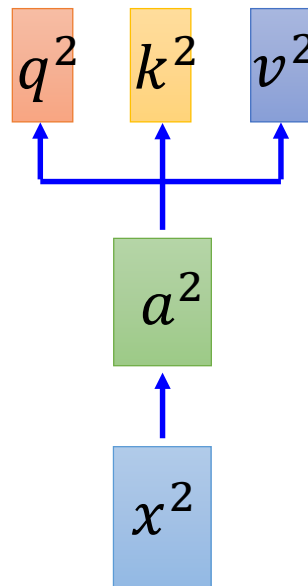
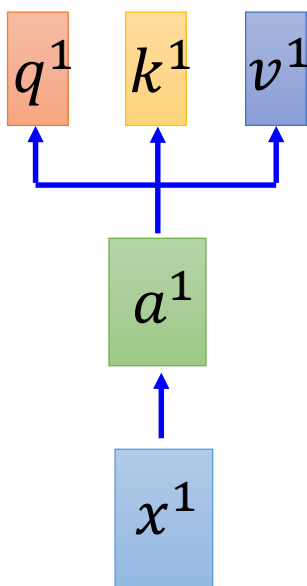
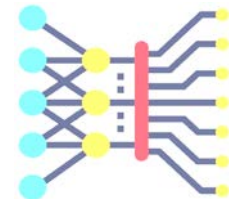
$$k^i = W^k a^i$$

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

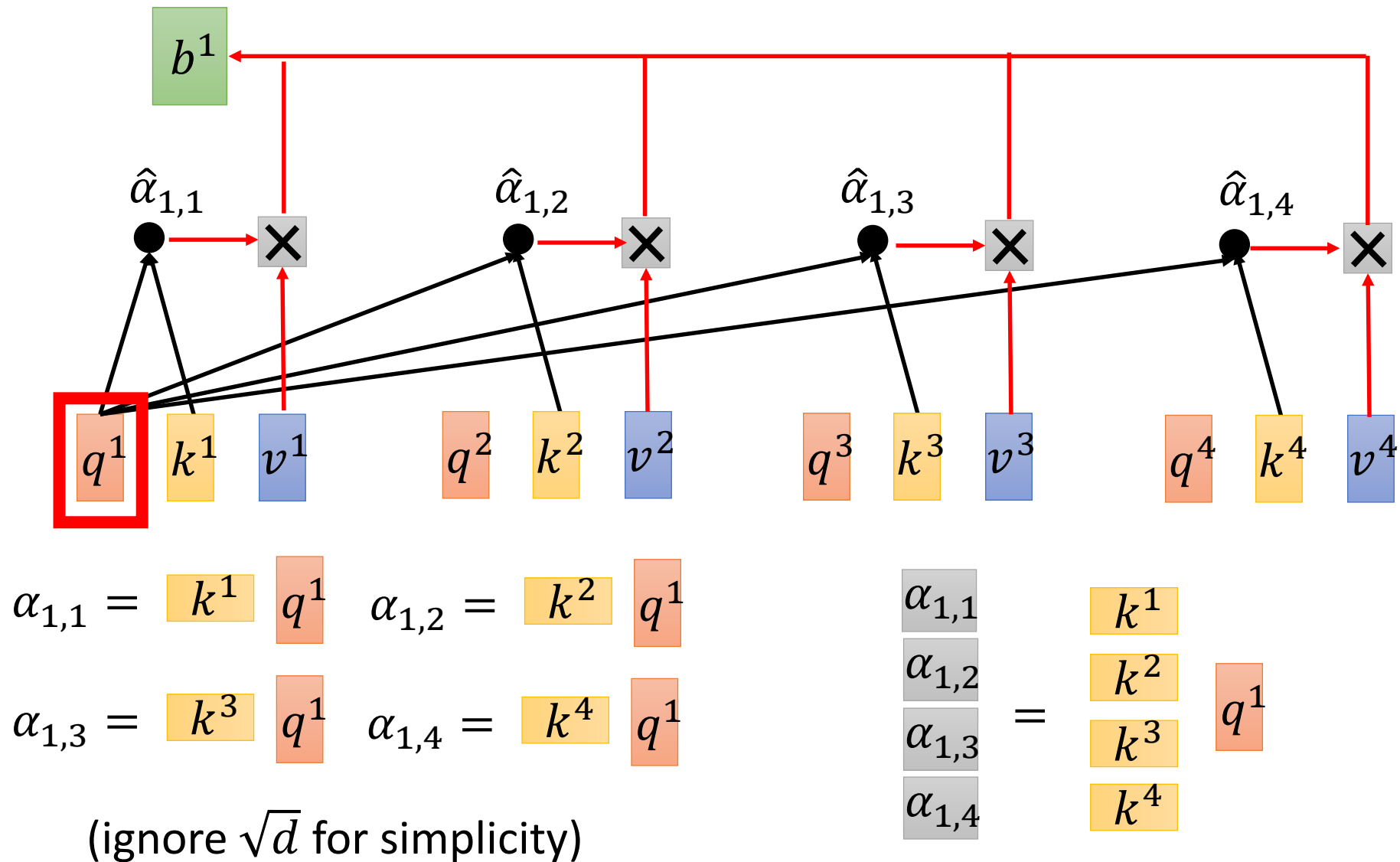
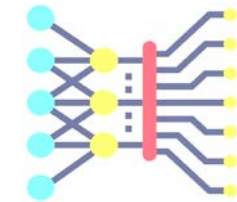
$$\begin{matrix} q^1 & q^2 & q^3 & q^4 \\ \hline Q \end{matrix} = \begin{matrix} W^q & \begin{matrix} a^1 & a^2 & a^3 & a^4 \\ \hline I \end{matrix} \end{matrix}$$

$$\begin{matrix} k^1 & k^2 & k^3 & k^4 \\ \hline K \end{matrix} = \begin{matrix} W^k & \begin{matrix} a^1 & a^2 & a^3 & a^4 \\ \hline I \end{matrix} \end{matrix}$$

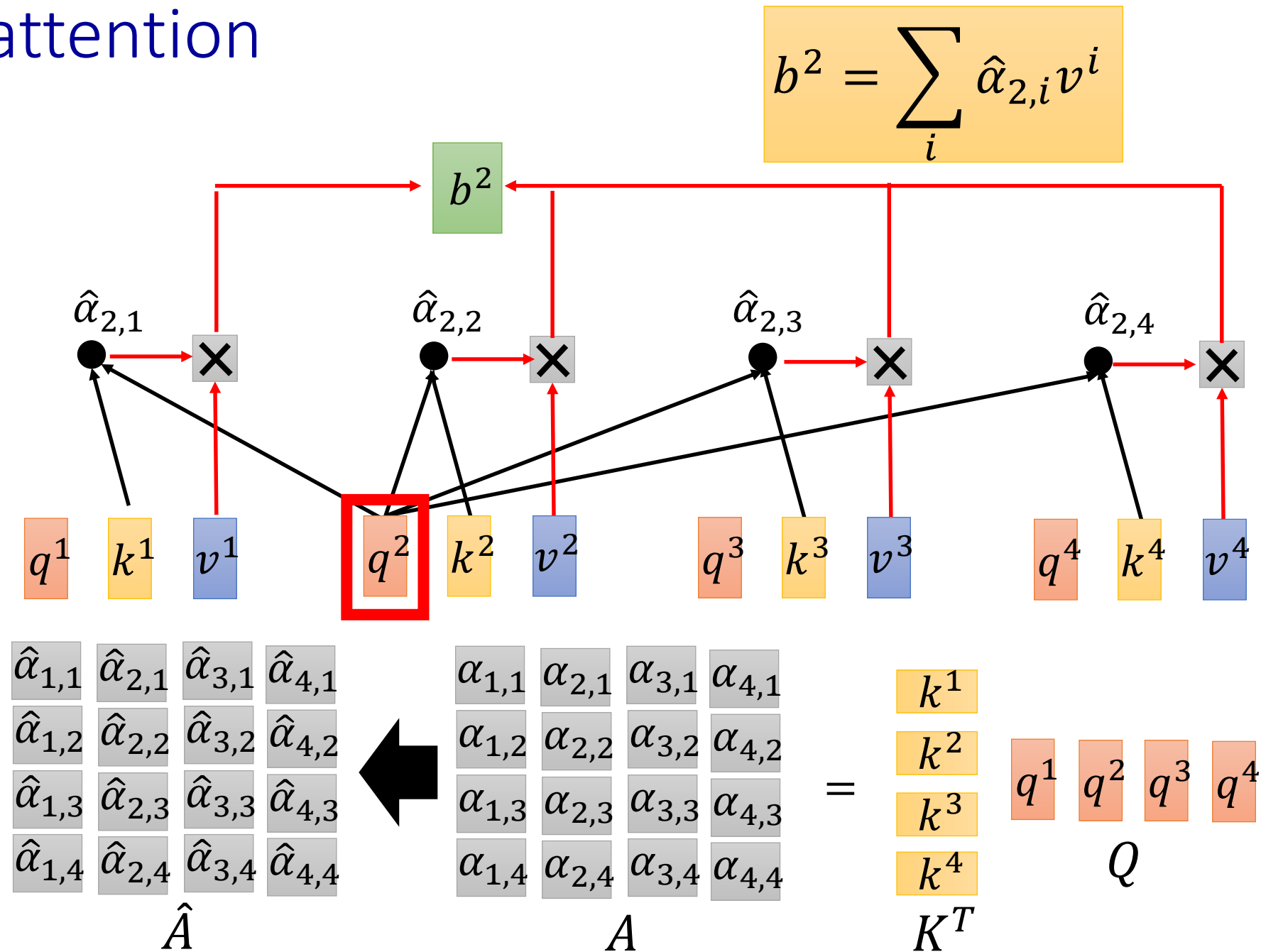
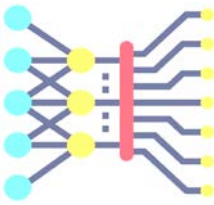
$$\begin{matrix} v^1 & v^2 & v^3 & v^4 \\ \hline V \end{matrix} = \begin{matrix} W^v & \begin{matrix} a^1 & a^2 & a^3 & a^4 \\ \hline I \end{matrix} \end{matrix}$$



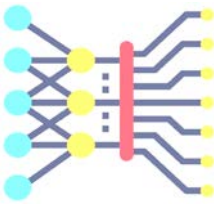
Self-attention



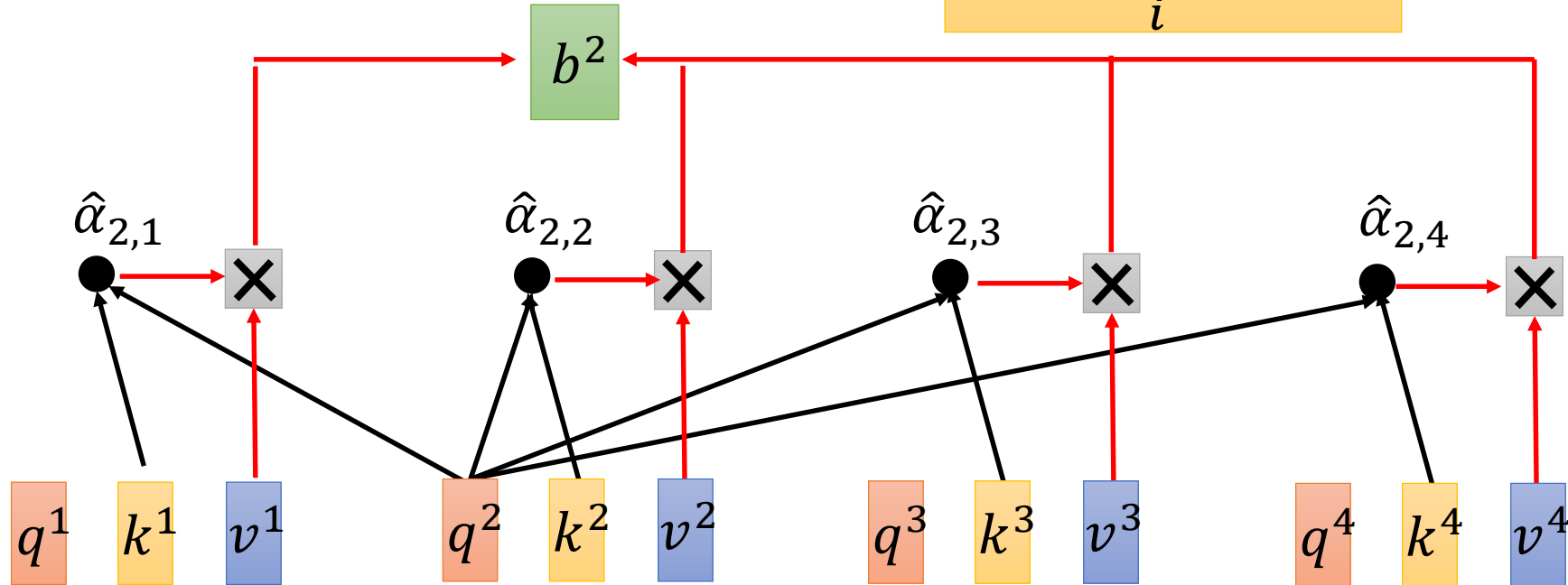
Self-attention



Self-attention

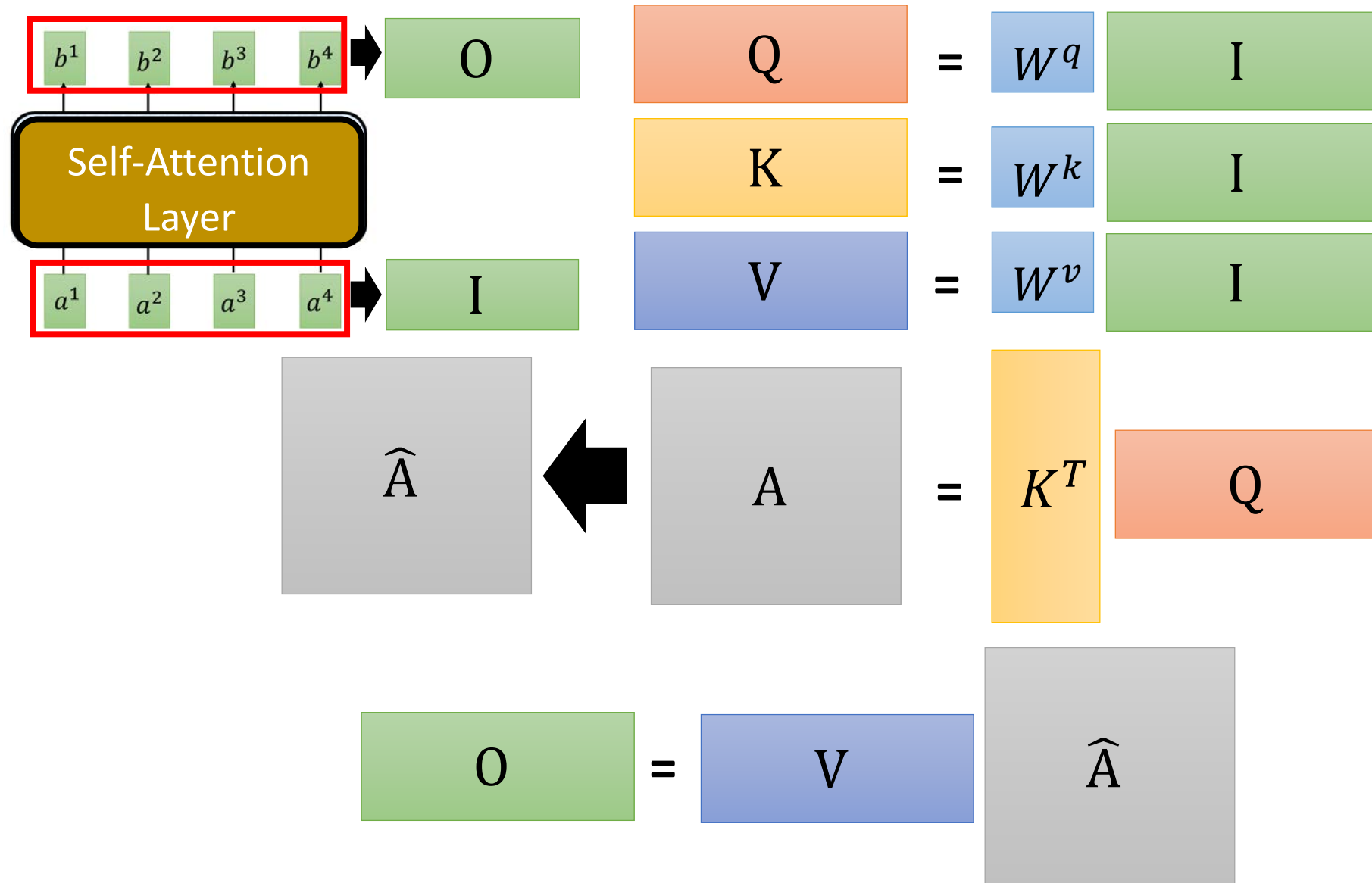
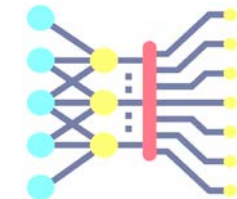


$$b^2 = \sum_i \hat{\alpha}_{2,i} v^i$$

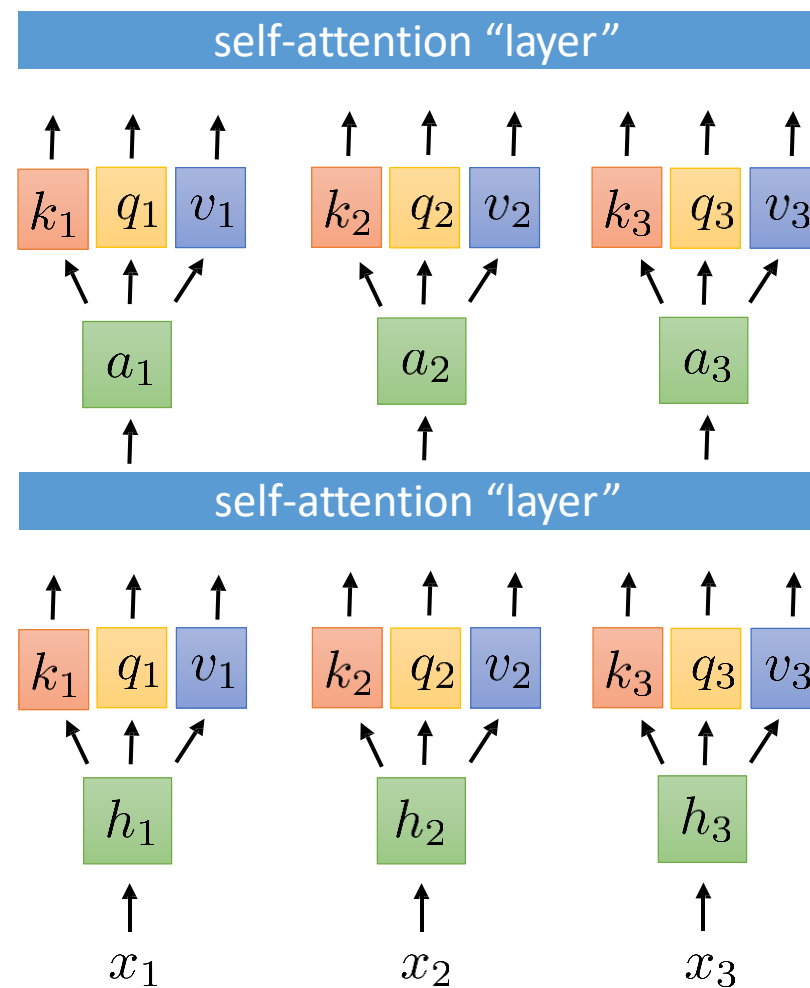
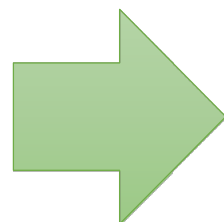
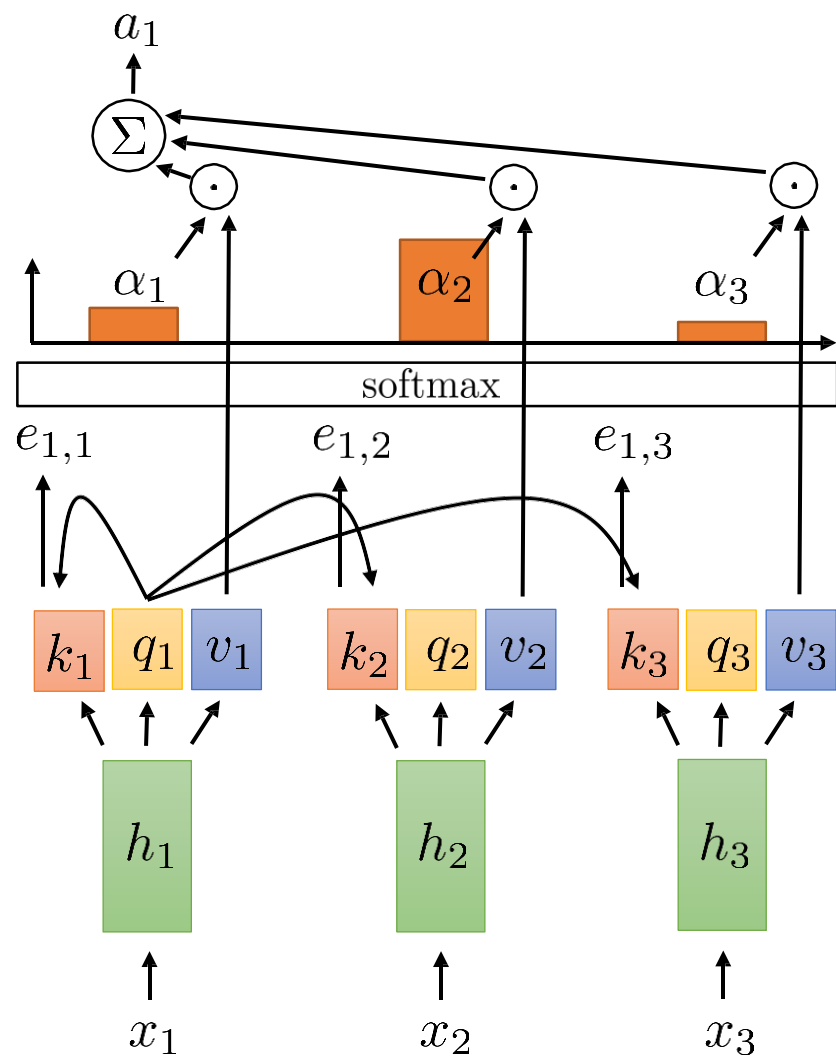


$$\begin{array}{c}
 \boxed{b^1} \boxed{b^2} \boxed{b^3} \boxed{b^4} \\
 O
 \end{array}
 =
 \begin{array}{c}
 \boxed{v^1} \boxed{v^2} \boxed{v^3} \boxed{v^4} \\
 V
 \end{array}
 \begin{array}{|c|c|c|c|}
 \hline
 \hat{\alpha}_{1,1} & \hat{\alpha}_{2,1} & \hat{\alpha}_{3,1} & \hat{\alpha}_{4,1} \\
 \hline
 \hat{\alpha}_{1,2} & \hat{\alpha}_{2,2} & \hat{\alpha}_{3,2} & \hat{\alpha}_{4,2} \\
 \hline
 \hat{\alpha}_{1,3} & \hat{\alpha}_{2,3} & \hat{\alpha}_{3,3} & \hat{\alpha}_{4,3} \\
 \hline
 \hat{\alpha}_{1,4} & \hat{\alpha}_{2,4} & \hat{\alpha}_{3,4} & \hat{\alpha}_{4,4} \\
 \hline
 \end{array}
 \hat{A}$$

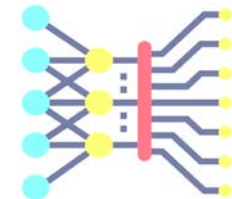
Self-attention

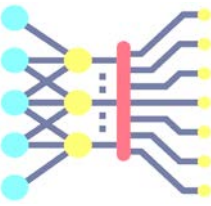


Self-Attention



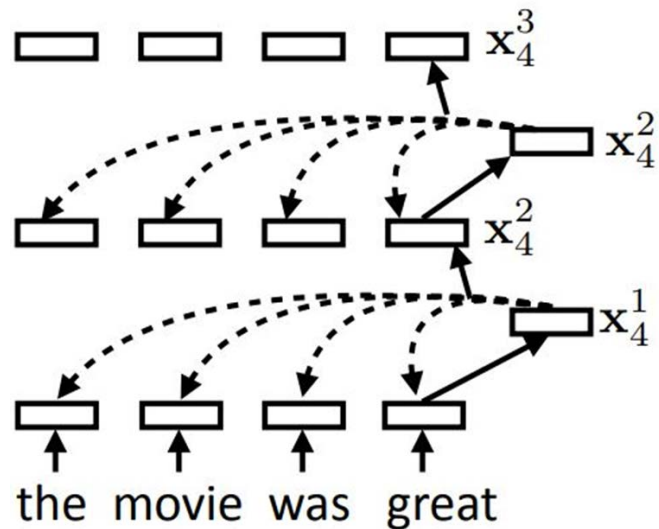
keep repeating until we've processed this enough
at the end, decode it into an answer





Multiple Attention Heads

- Multiple attention heads can learn to attend in different ways
- Requires additional parameters to compute different attention values and transform vectors



k : level number

L : number of heads

$$X = \mathbf{x}_1, \dots, \mathbf{x}_n$$

$$\mathbf{x}_i^1 = \mathbf{x}_i$$

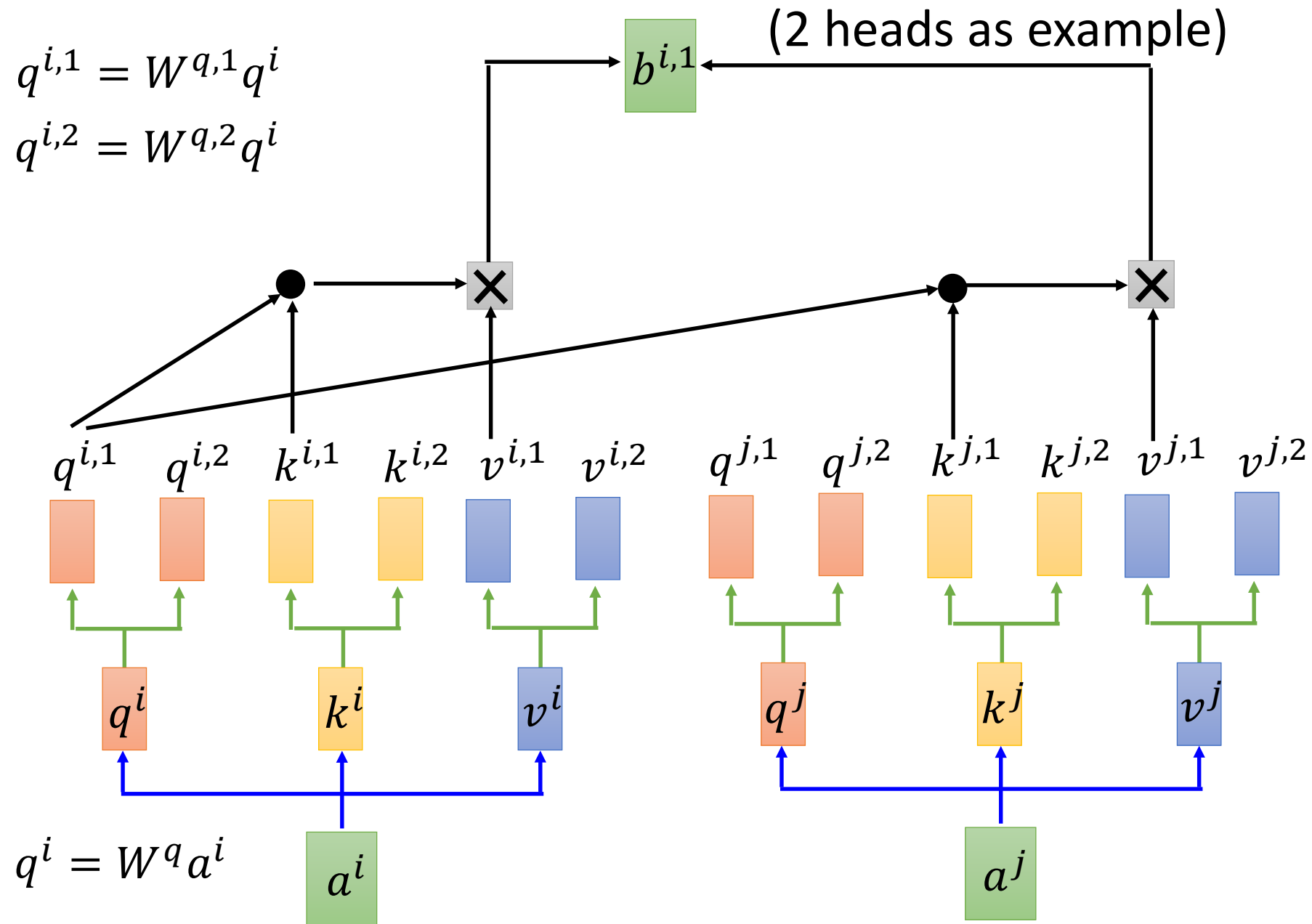
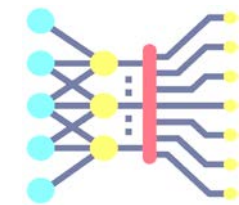
$$\bar{\alpha}_{i,j}^{k,l} = \mathbf{x}_i^{k-1} \mathbf{W}^{k,l} \mathbf{x}_j^{k-1}$$

$$\alpha_i^{k,l} = \text{softmax}(\bar{\alpha}_{i,1}^{k,l}, \dots, \bar{\alpha}_{i,n}^{k,l})$$

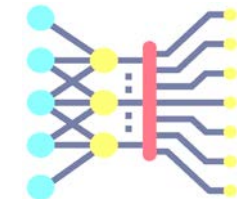
$$\mathbf{x}_i^{k,l} = \sum_{j=1}^n \alpha_{i,j}^{k,l} \mathbf{x}_j^{k-1}$$

$$\mathbf{x}_i^k = \mathbf{V}^k [\mathbf{x}_i^{k,1}; \dots; \mathbf{x}_i^{k,L}]$$

Multi-head Self-attention



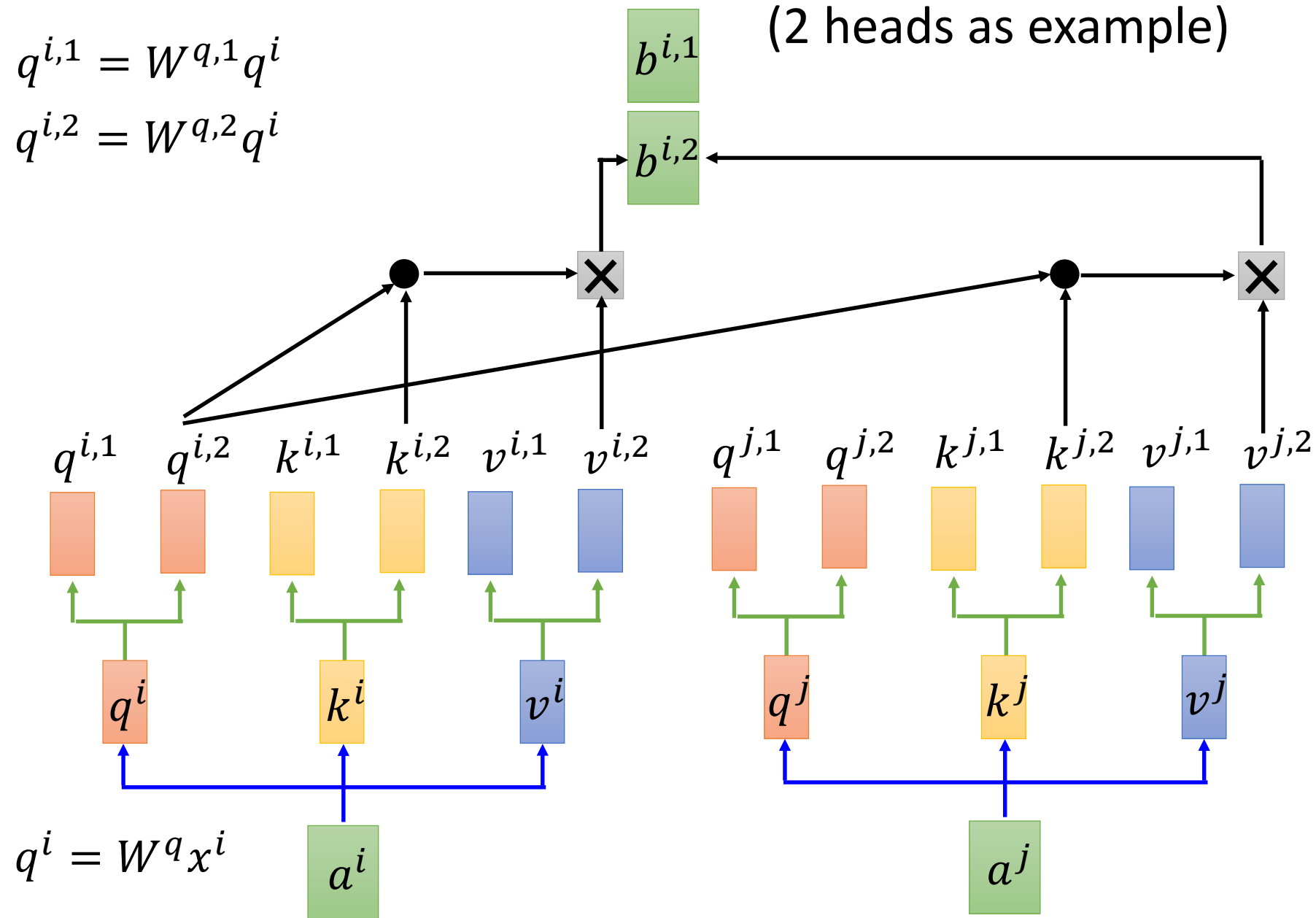
Multi-head Self-attention



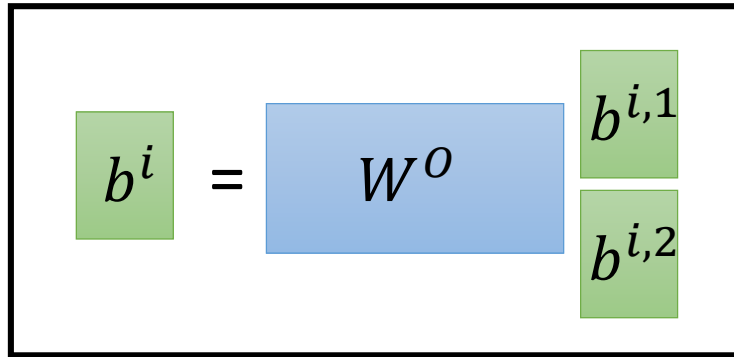
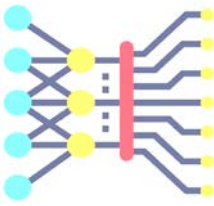
$$q^{i,1} = W^{q,1} q^i$$

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

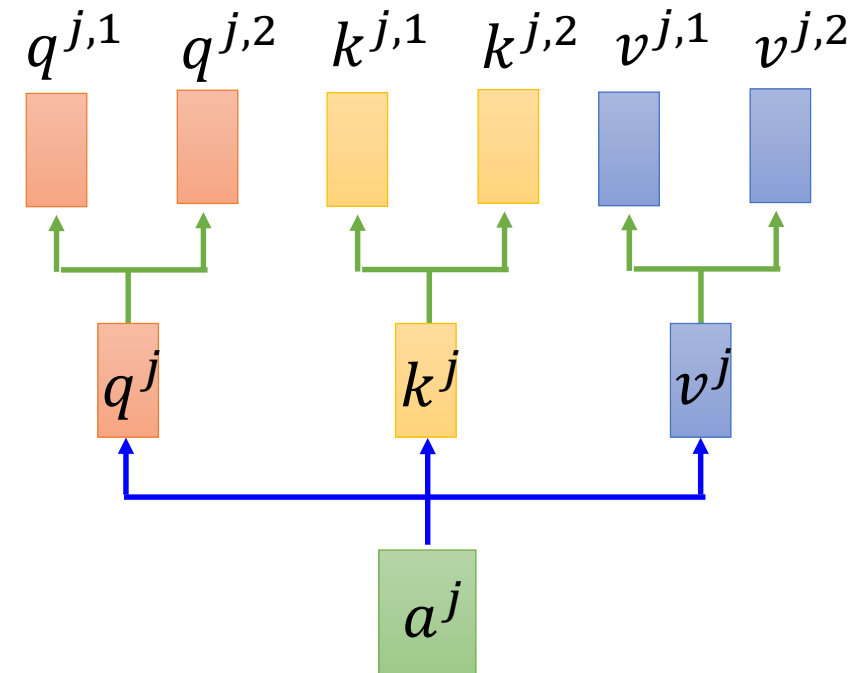
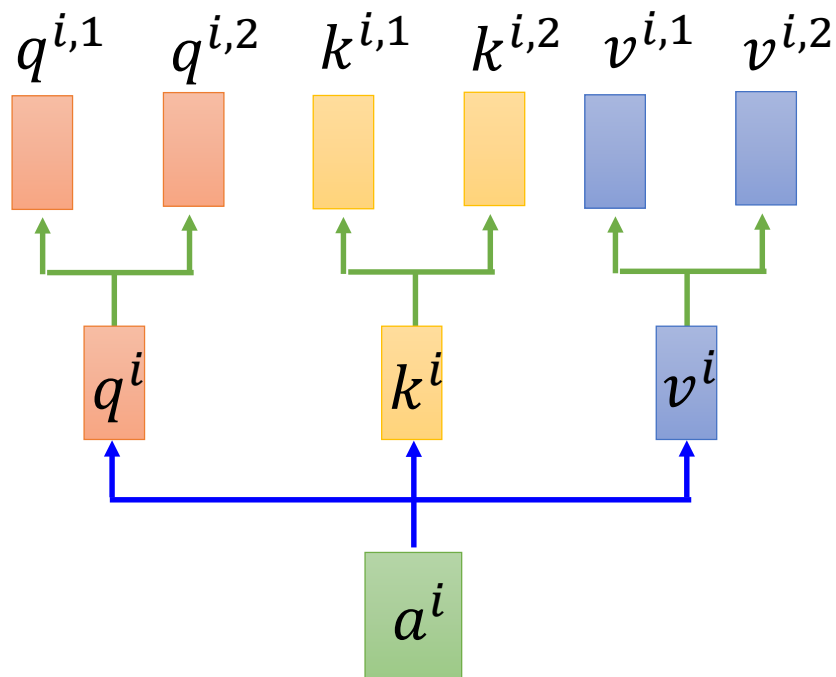
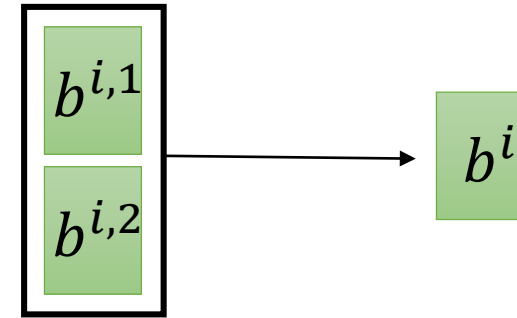
(2 heads as example)



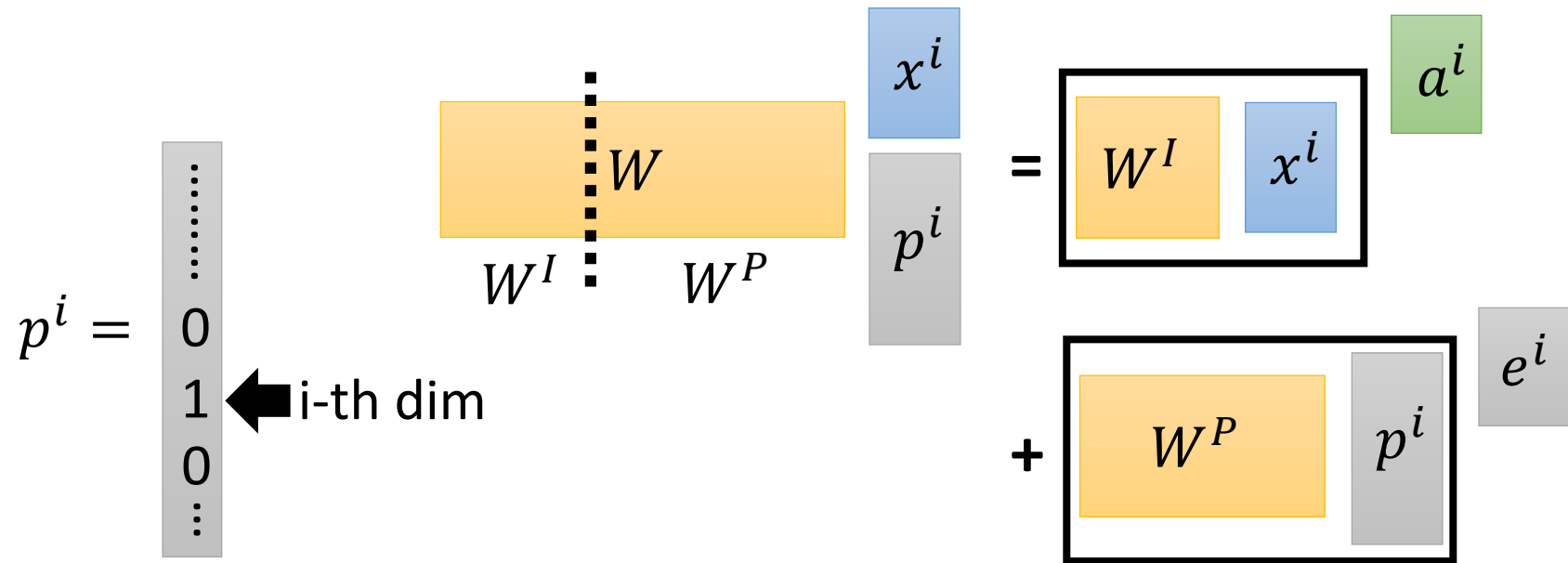
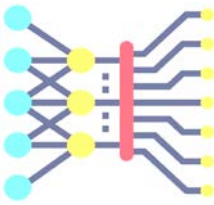
Multi-head Self-attention



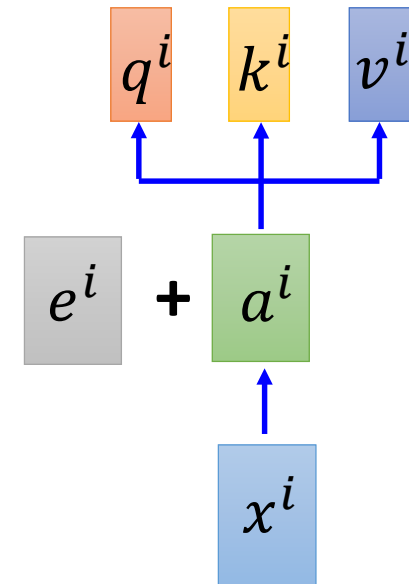
(2 heads as example)



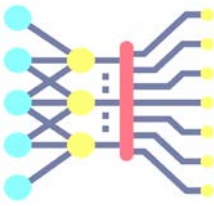
Positional Encoding



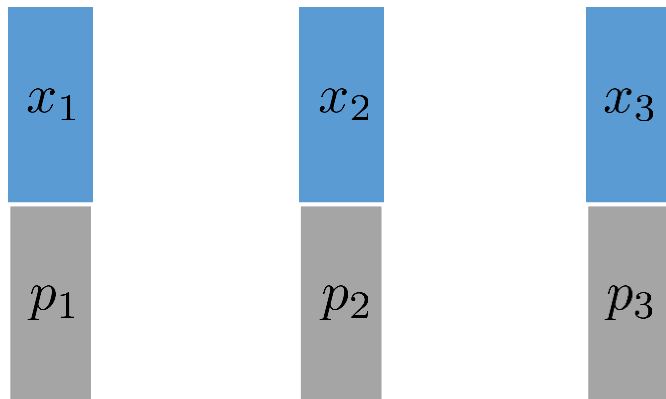
- Each position has a unique positional vector e^i (not learned from data)
- each x^i appends a one-hot vector p^i or add them



Positional encoding: learned



Another idea: just learn a positional encoding



Different for every input sequence

The same learned values for every sequence

but different for different time steps

How many values do we need to learn?

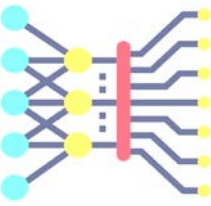
dimensionality max sequence length

$$P = [p_1, p_2, \dots, p_T] \in \mathbb{R}^{d \times T}$$

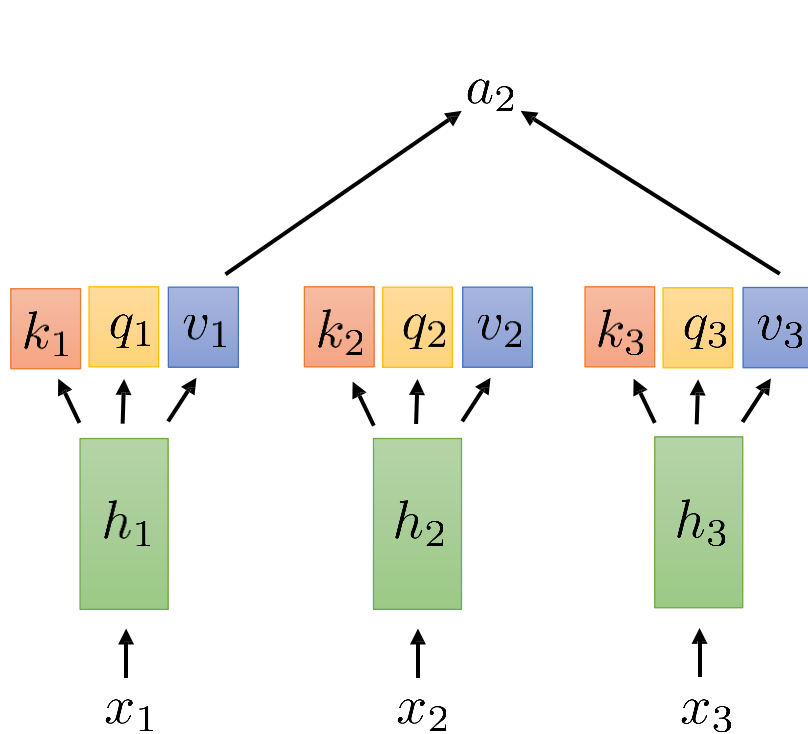
+ more flexible (and perhaps more optimal) than sin/cos encoding

+ a bit more complex, need to pick a max sequence length (and can't generalize beyond it)

Multi-head attention



Since we are relying **entirely** on attention now, we might want to incorporate **more than one** time step



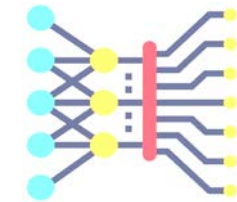
$$a_l = \sum_t \alpha_{l,t} v_t$$

because of softmax, this will
be dominated by one value

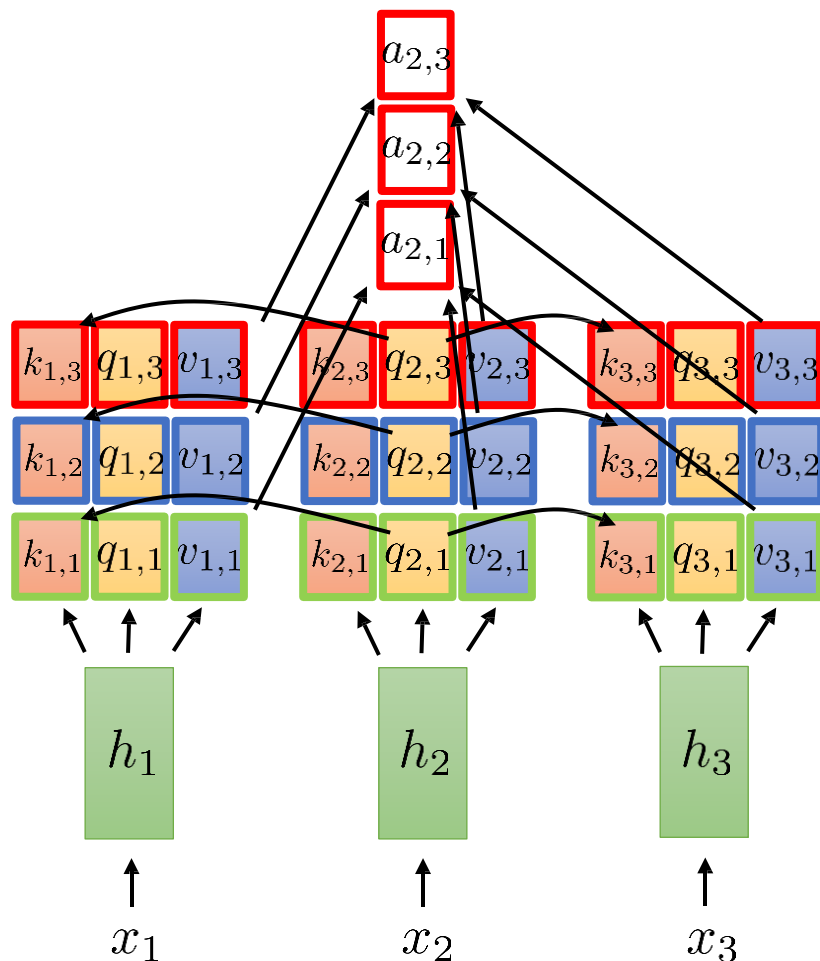
$$e_{l,t} = q_l \cdot k_t$$

hard to specify that you want two
different things (e.g., the subject
and the object in a sentence)

Multi-head attention



Idea: have multiple keys, queries, and values for every time step!



full attention vector formed by concatenation:

$$a_2 = \begin{bmatrix} a_{2,1} \\ a_{2,2} \\ a_{2,3} \end{bmatrix}$$

compute weights **independently** for each head

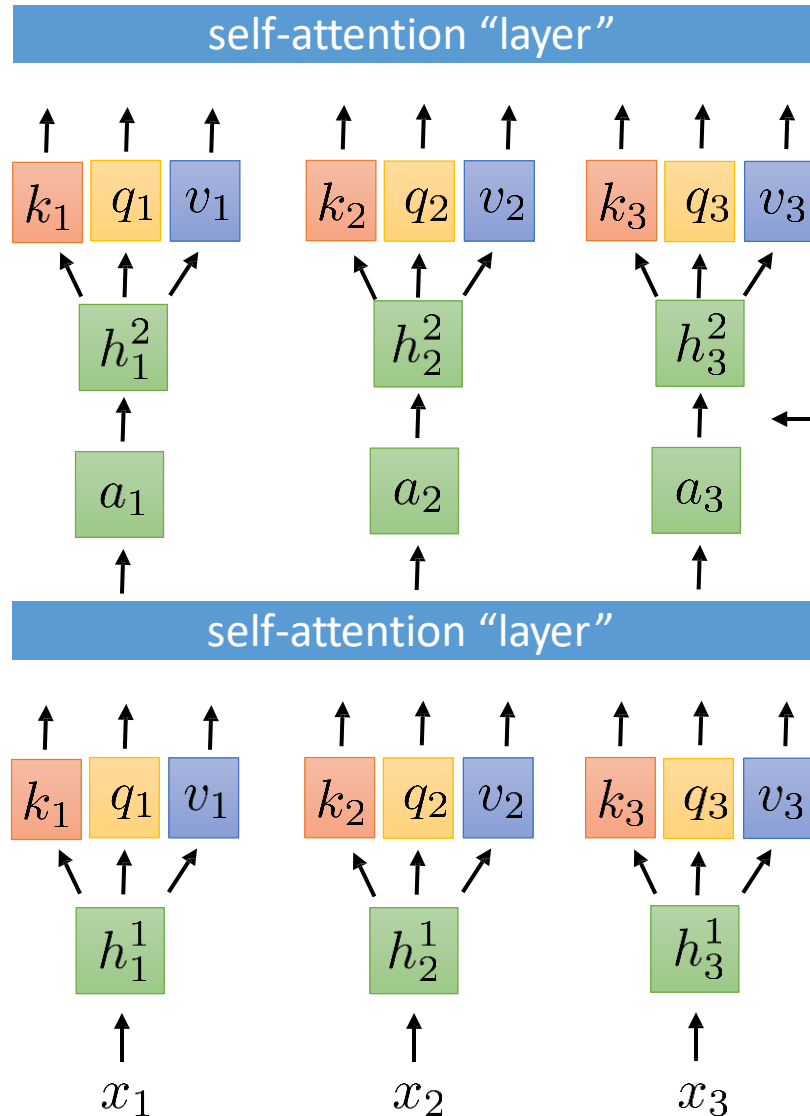
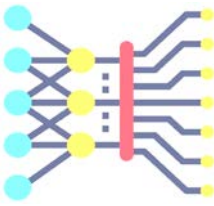
$$e_{l,t,i} = q_{l,i} \cdot k_{l,i}$$

$$\alpha_{l,t,i} = \exp(e_{l,t,i}) / \sum_{t'} \exp(e_{l,t',i})$$

$$a_{l,i} = \sum_t \alpha_{l,t,i} v_{t,i}$$

around **8** heads seems to work
pretty well for big models

Alternating self-attention & nonlinearity



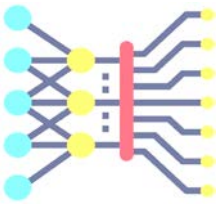
some non-linear (learned) function
e.g., $h_t^\ell = \sigma(W^\ell a_t^\ell + b^\ell)$

just a neural net applied at every position
after every self-attention layer!

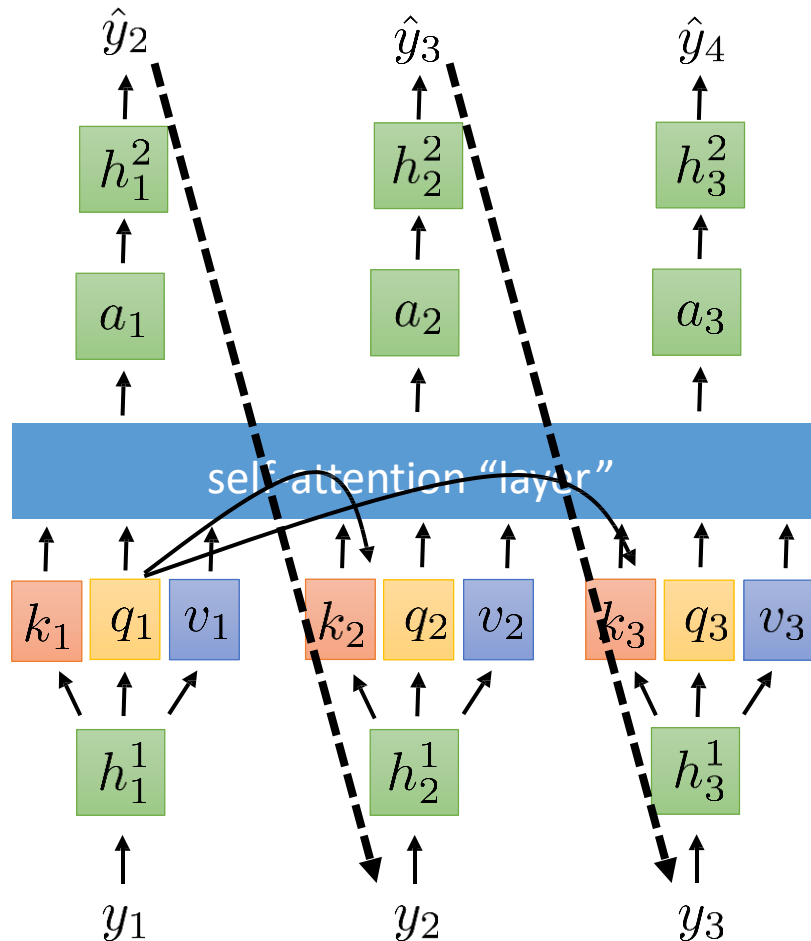
Sometimes referred to as “position-
wise feedforward network”

We’ll describe some specific
commonly used choices shortly

Self-attention can see the future!



A **crude** self-attention “language model”:



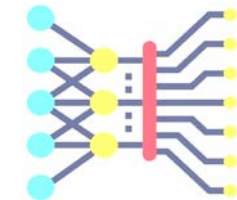
(in reality, we would have many alternating self-attention layers and position-wise feedforward networks, not just one)

Big problem: self-attention at step 1 can look at the value at steps 2 & 3, which is based on the **inputs** at steps 2 & 3

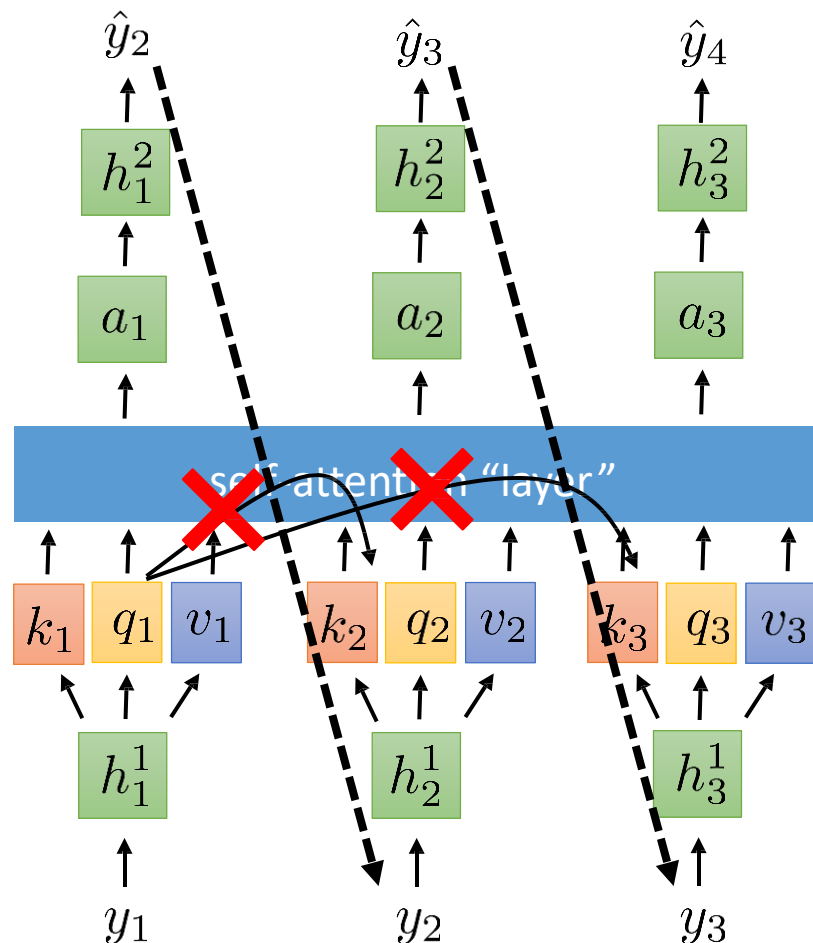
At test time (when decoding), the **inputs** at steps 2 & 3 will be based on the output at step 1...

...which requires knowing the **input** at steps 2 & 3

Masked attention



A **crude** self-attention “language model”:



At test time (when decoding), the **inputs** at steps 2 & 3 will be based on the output at step 1...

...which requires knowing the **input** at steps 2 & 3

Must allow self-attention into the **past**...

...but not into the **future**

Easy solution:

~~$$e_{l,t} = q_l \cdot k_t$$~~

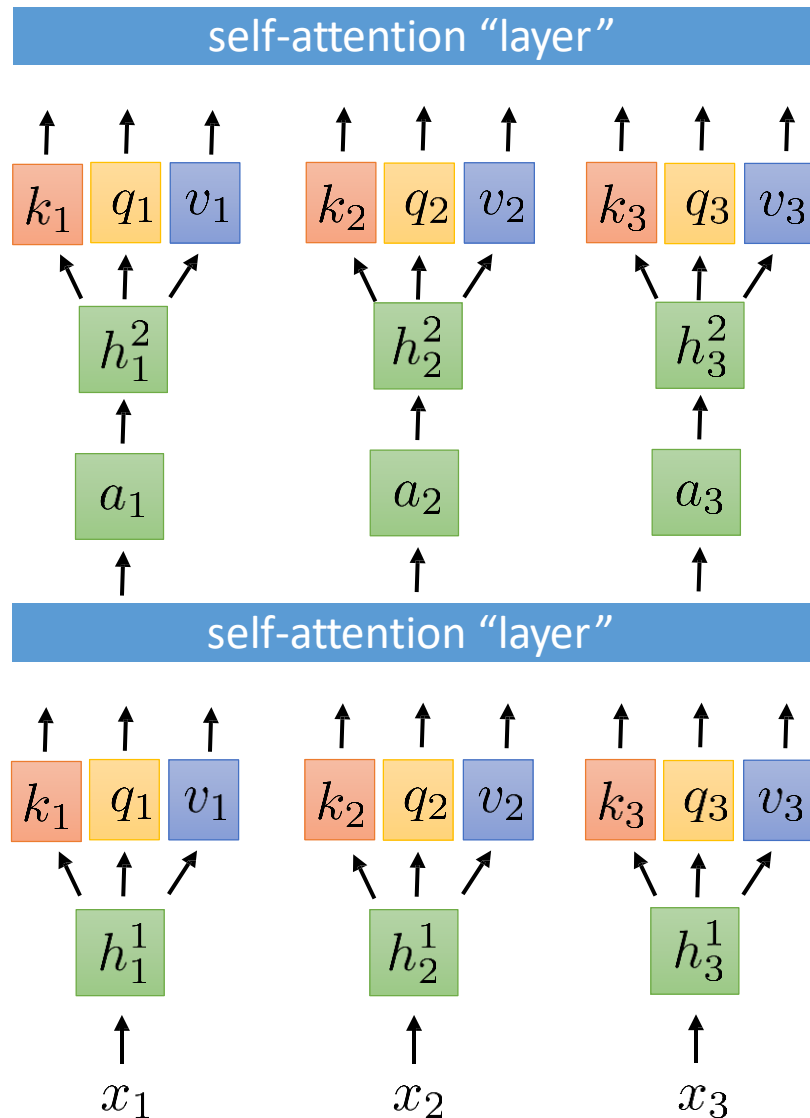
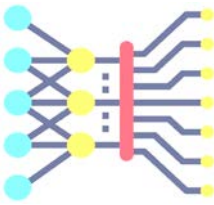
$$e_{l,t} = \begin{cases} q_l \cdot k_t & \text{if } l \geq t \\ -\infty & \text{otherwise} \end{cases}$$

in practice:

just replace $\exp(e_{l,t})$ with 0 if $l < t$

inside the softmax

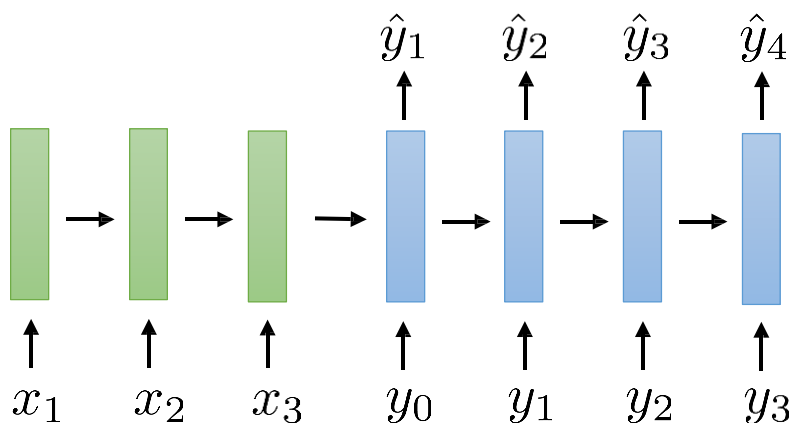
Transformer summary



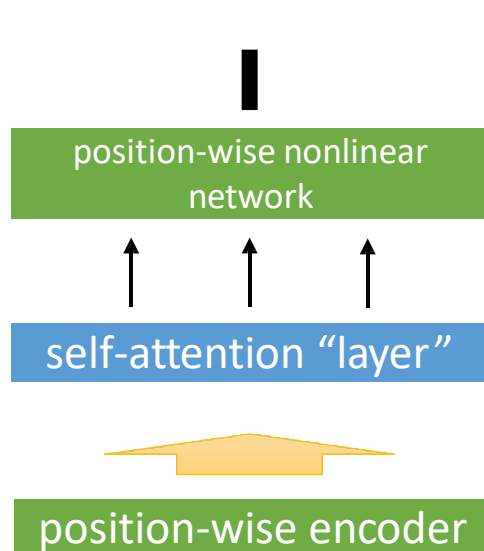
- These are generally called “Transformers” because they transform one sequence into another at each layer.
- Alternate self-attention “layers” with nonlinear position-wise feedforward networks (to get nonlinear transformations)
- Use positional encoding (on the input or input embedding) to make the model aware of relative positions of tokens
- Use multi-head attention
- Use masked attention if you want to use the model for decoding.

The “classic” transformer

As compared to a sequence
to sequence RNN model

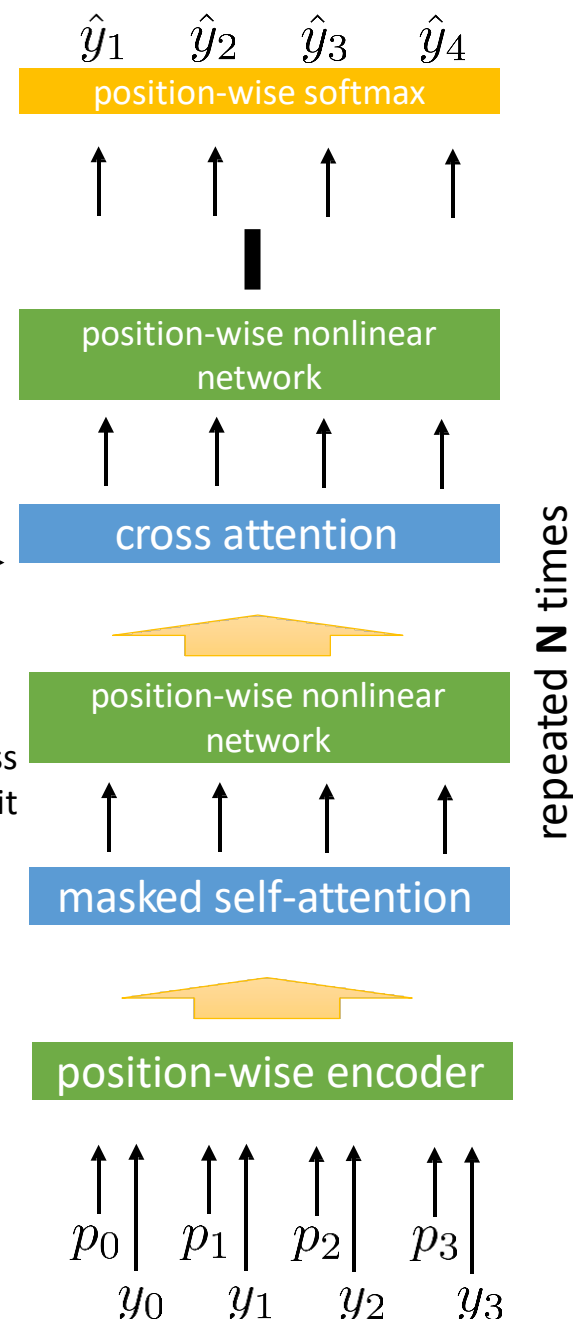


repeated **N** times

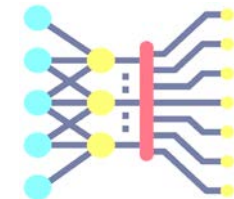


p_1 p_2 p_3
 x_1 x_2 x_3

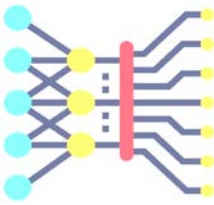
we’ll discuss
how this bit
works soon



p_0 p_1 p_2 p_3
 y_0 y_1 y_2 y_3

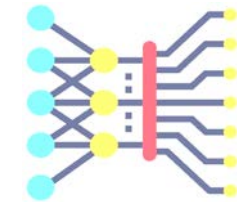


The Final Linear and Softmax Layer



- 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.

Combining encoder and decoder values



“Cross-attention”

Much more like the **standard** attention from the previous lecture

$$\text{query: } q_l^\ell = W_q^\ell s_l^\ell$$

output of position-wise nonlinear network at (decoder) layer ℓ , step l

$$\text{key: } k_t^\ell = W_k^\ell h_t^\ell$$

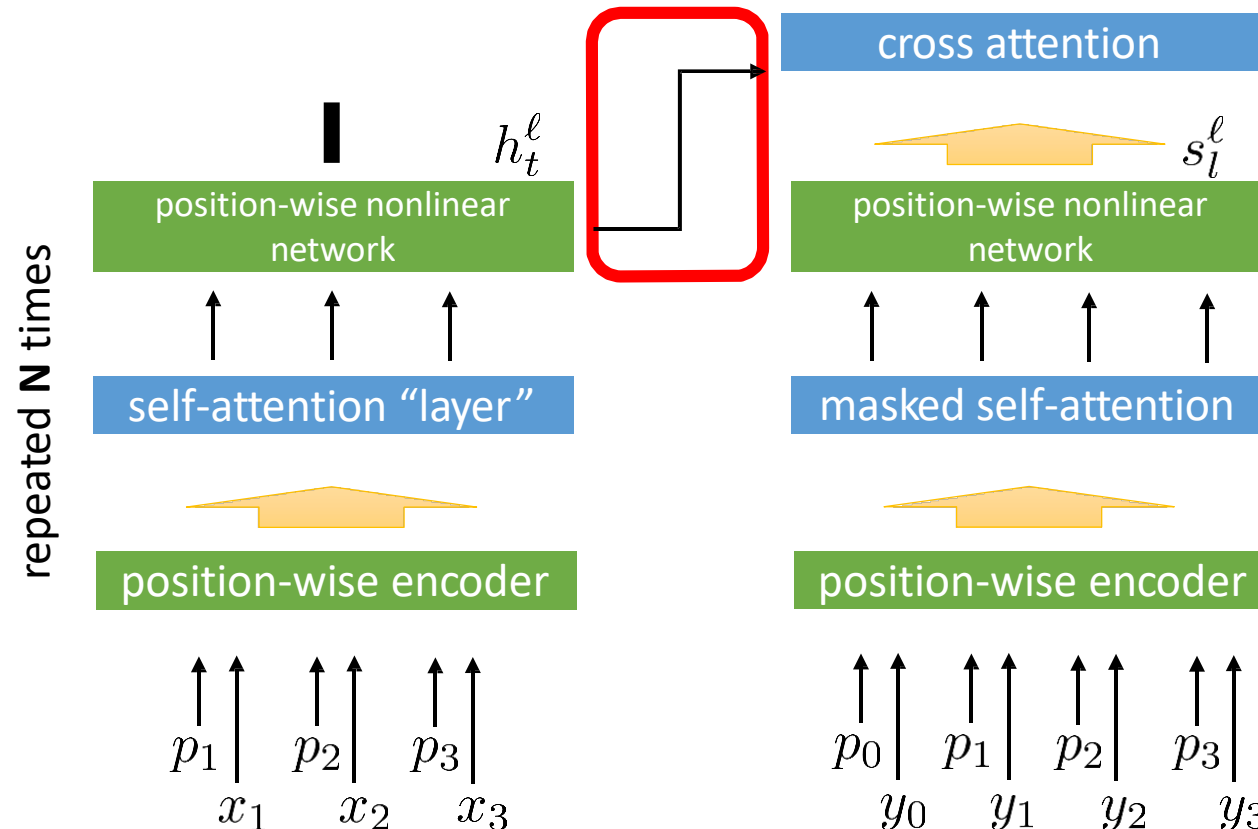
output of position-wise nonlinear network at (encoder) layer ℓ , step t

$$\text{value: } v_t^\ell = W_v^\ell h_t^\ell$$

$$e_{l,t}^\ell = q_l^\ell \cdot k_t^\ell$$

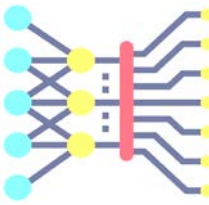
$$\alpha_{l,t}^\ell = \frac{\exp(e_{l,t}^\ell)}{\sum_{t'} \exp(e_{l,t'}^\ell)}$$

$$c_l^\ell = \sum_t \alpha_{l,t}^\ell v_t^\ell \quad \text{cross attention output}$$

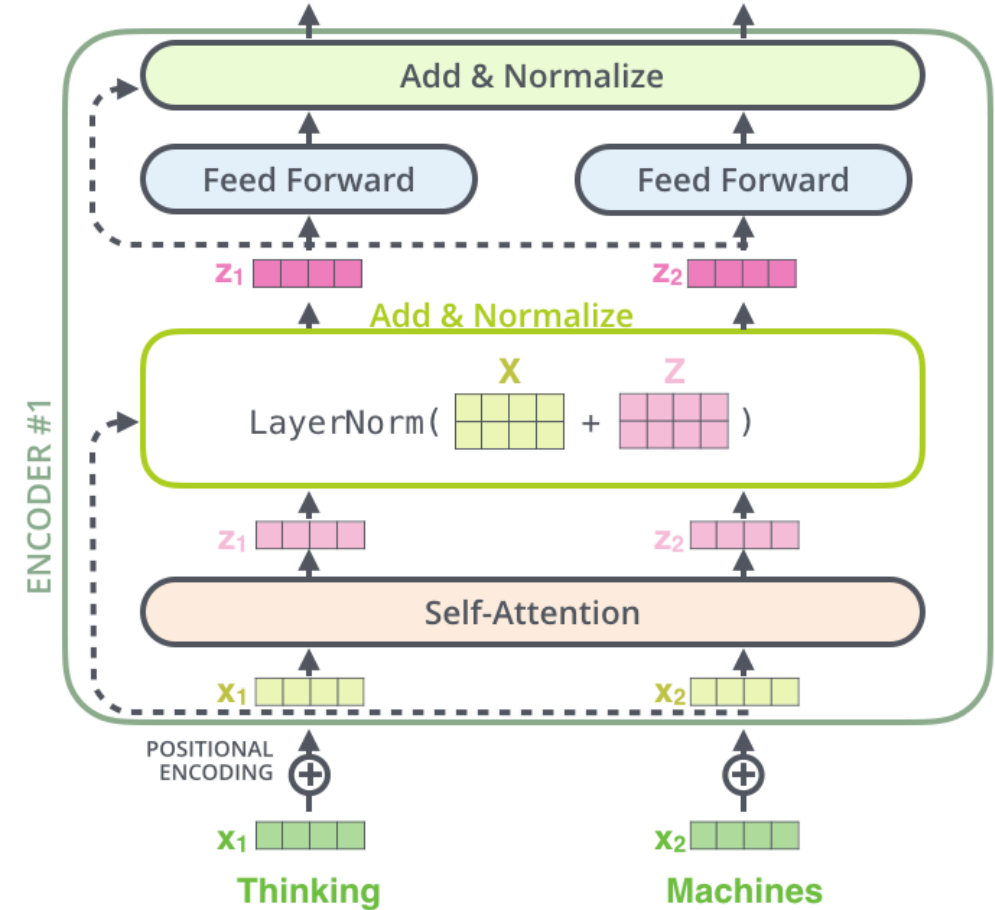
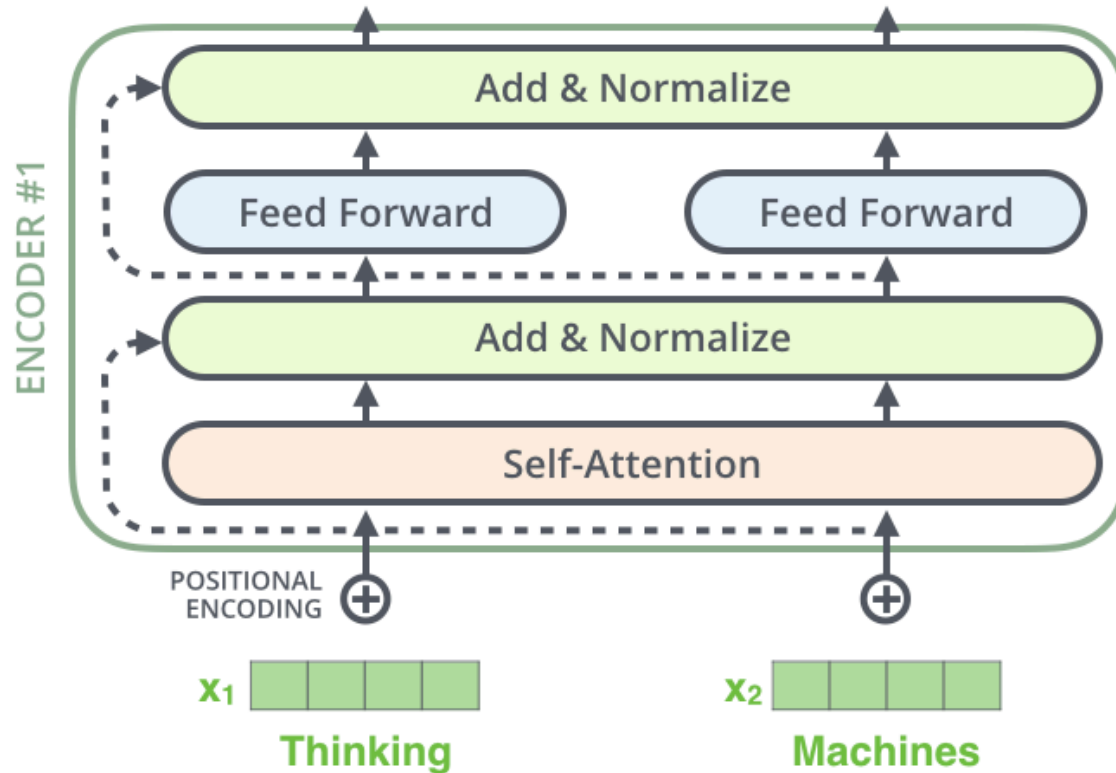


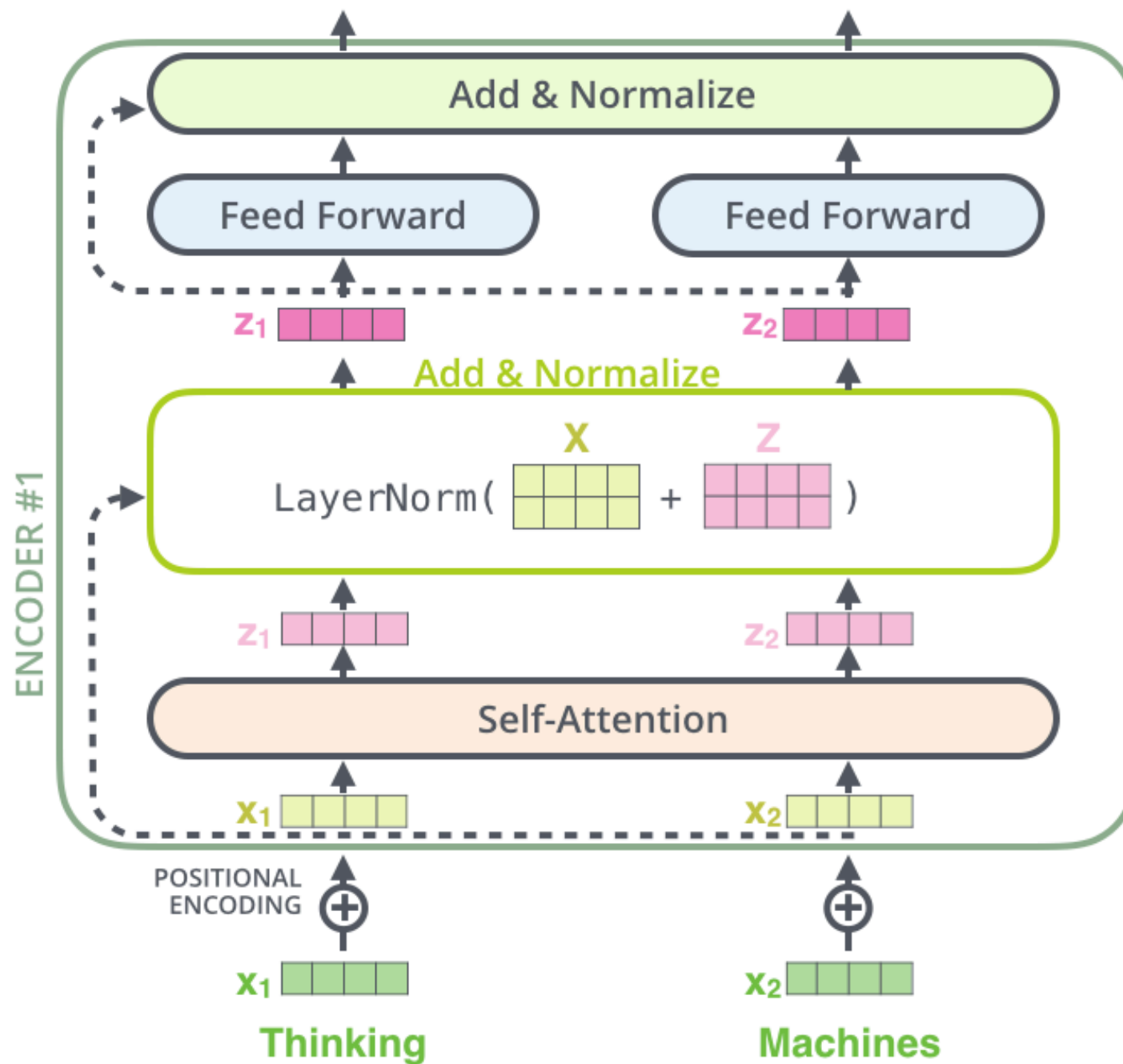
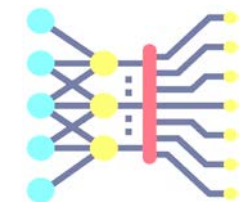
in reality, cross-attention is **also** multi-headed!

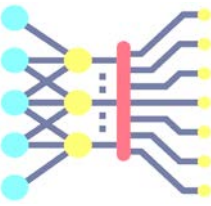
The Residuals



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







Layer normalization

Main idea: batch normalization is very helpful, but hard to use with sequence models. Sequences are different lengths, makes normalizing across the batch hard. Sequences can be very long, so we sometimes have small batches.

Simple solution: “layer normalization” – like batch norm, but not across the batch.

Batch norm

a_1, a_2, \dots, a_B ← d -dimensional vectors for each sample in batch

d -dim → $\mu = \frac{1}{B} \sum_{i=1}^B a_i$ $\sigma = \sqrt{\frac{1}{B} \sum_{i=1}^B (a_i - \mu)^2}$

$$\bar{a}_i = \frac{a_i - \mu}{\sigma} \gamma + \beta$$

Layer norm

Different dimensions of a

a

$\mu = \frac{1}{d} \sum_{j=1}^d a_j$ $\sigma = \sqrt{\frac{1}{d} \sum_{j=1}^d (a_j - \mu)^2}$

1-dim

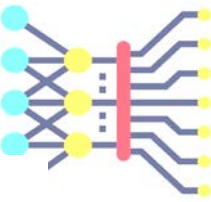
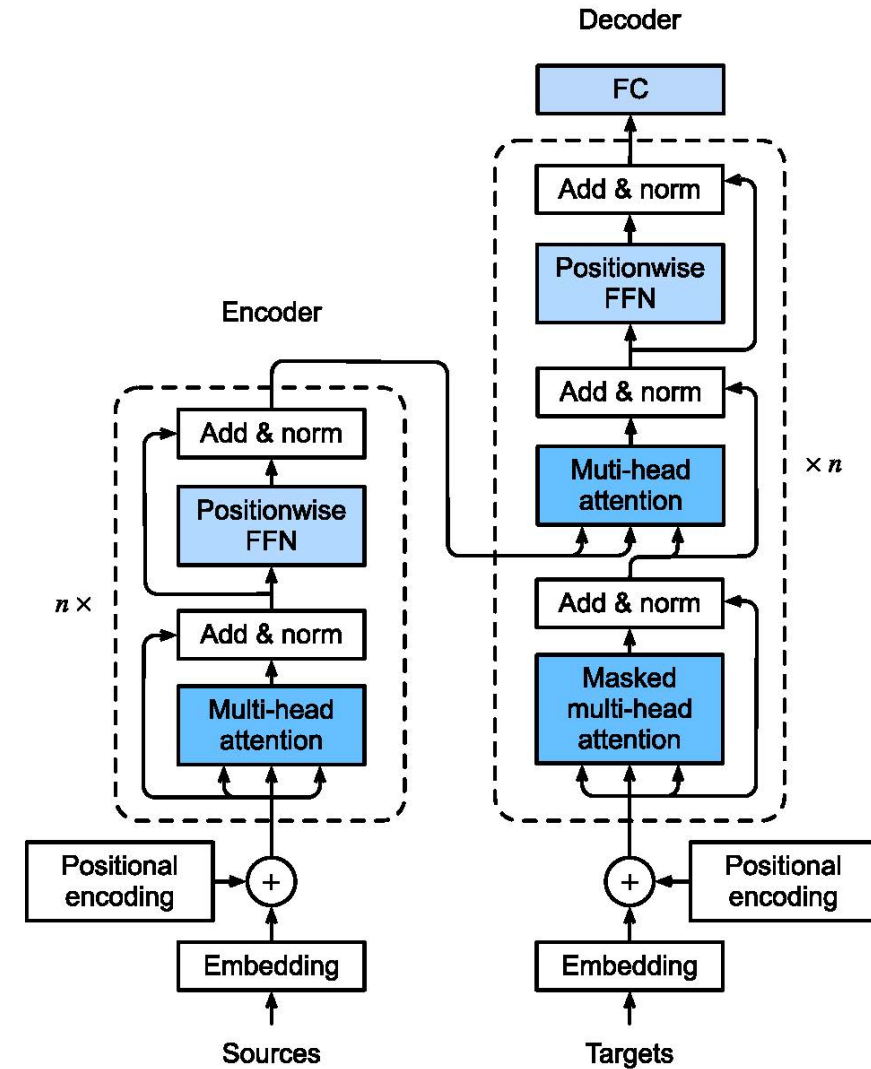
$$\bar{a} = \frac{a - \mu}{\sigma} \gamma + \beta$$

The Transformer Architecture

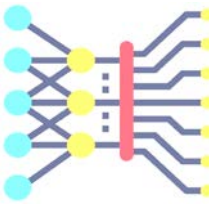
Composed of an encoder and a decoder.

- **Encoder:**

- a stack of multiple identical blocks
- each block has two sublayers
 1. a multi-head self-attention pooling (queries, keys, and values are all from the outputs of the previous encoder layer)
 2. a positionwise feed-forward network
 3. A residual connection is employed around both sublayers followed by layer normalization

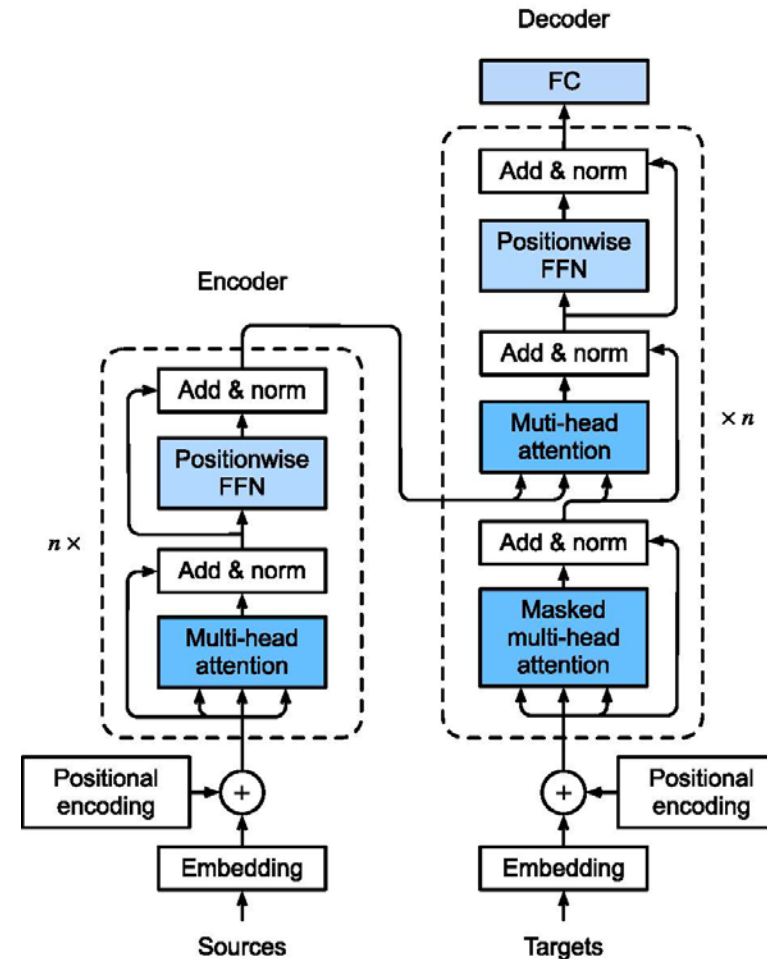


The Transformer Architecture

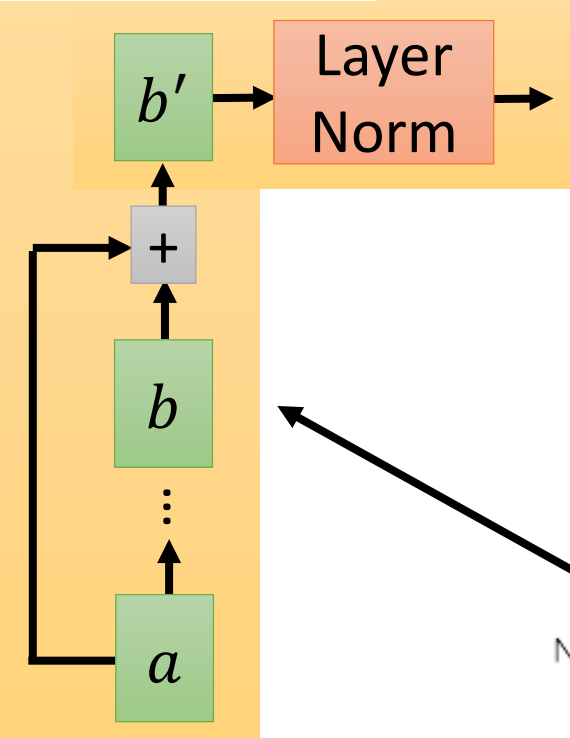
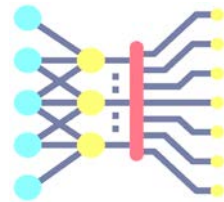


Decoder:

- a stack of multiple identical blocks
- each block has three sublayers.
 1. a multi-head self-attention pooling -- each position in the decoder is allowed to only attend to all positions in the decoder up to that position
 2. Encoder-decoder attention: queries are from the outputs of the previous decoder layer, and the keys and values are from the Transformer encoder outputs
 3. A positionwise feed-forward network
- A residual connection is employed around both sublayers followed by layer normalization

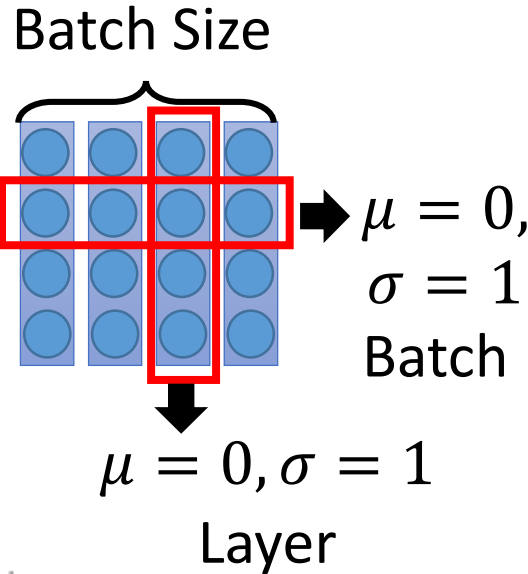
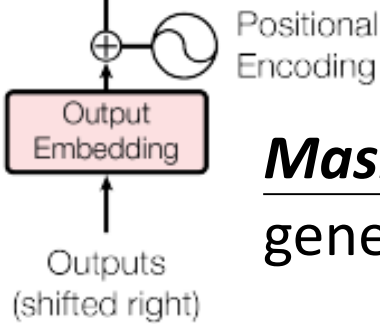
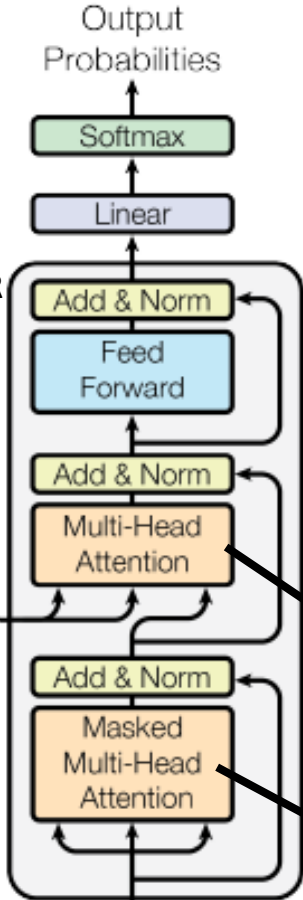
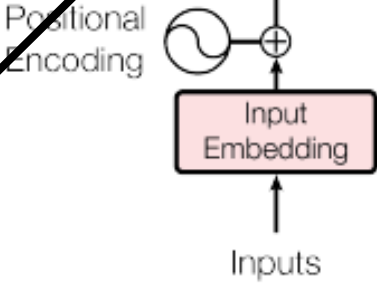
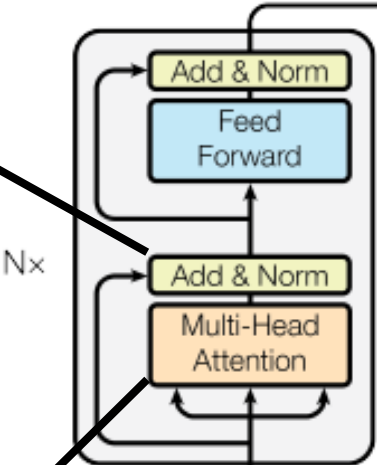
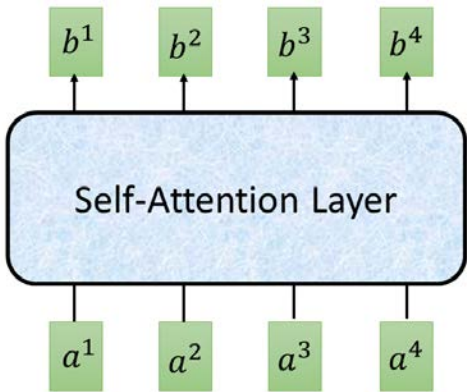


Transformer



Layer Norm:
<https://arxiv.org/abs/1607.06450>

Batch Norm:
<https://www.youtube.com/watch?v=BZh1ltr5Rkg>

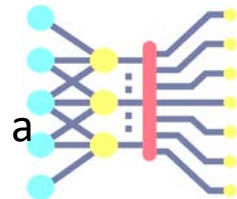


attend on the input sequence

Masked: attend on the generated sequence

Putting it all together

Decoder decodes one position at a time with masked attention



The Transformer

6 layers, each with $d = 512$

$\bar{h}_t^\ell = \text{LayerNorm}(\bar{a}_t^\ell + h_t^\ell)$
passed to next layer $\ell + 1$

multi-head attention keys and values
 $k_{t,1}^\ell, \dots, k_{t,m}^\ell$ and $v_{t,1}^\ell, \dots, v_{t,m}^\ell$

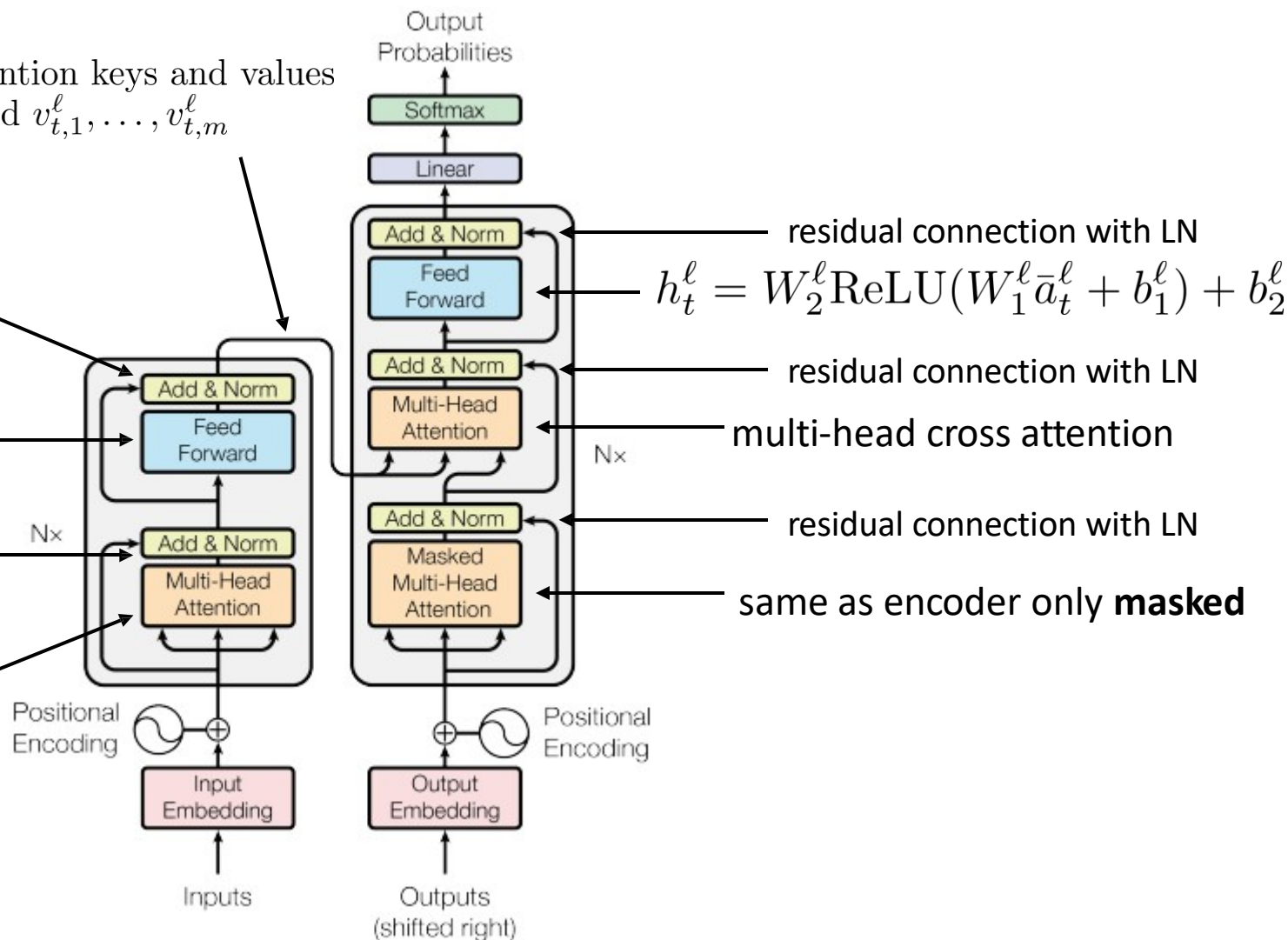
$$h_t^\ell = W_2^\ell \text{ReLU}(W_1^\ell \bar{a}_t^\ell + b_1^\ell) + b_2^\ell$$

2-layer neural net at each position

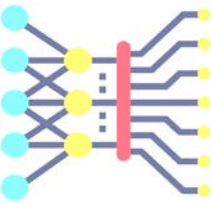
$\bar{a}_t^\ell = \text{LayerNorm}(\bar{h}_t^{\ell-1} + a_t^\ell)$
essentially a residual connection with LN

input: $\bar{h}_t^{\ell-1}$
output: a_t^ℓ

concatenates attention from all heads



Why transformers?



Downsides:

- **Attention computations are technically $O(n^2)$**
- **Somewhat more complex to implement (positional encodings, etc.)**

Benefits:

- + **Much better long-range connections**
- + **Much easier to parallelize**
- + **In practice, can make it much deeper (more layers) than RNN**

The benefits seem to **vastly** outweigh the downsides, and transformers work **much** better than RNNs (and LSTMs) in many cases

Arguably one of the most important sequence modeling improvements of the past decade