Seq2Seq and Attention Model

how we pay visual attention to different regions of an image?

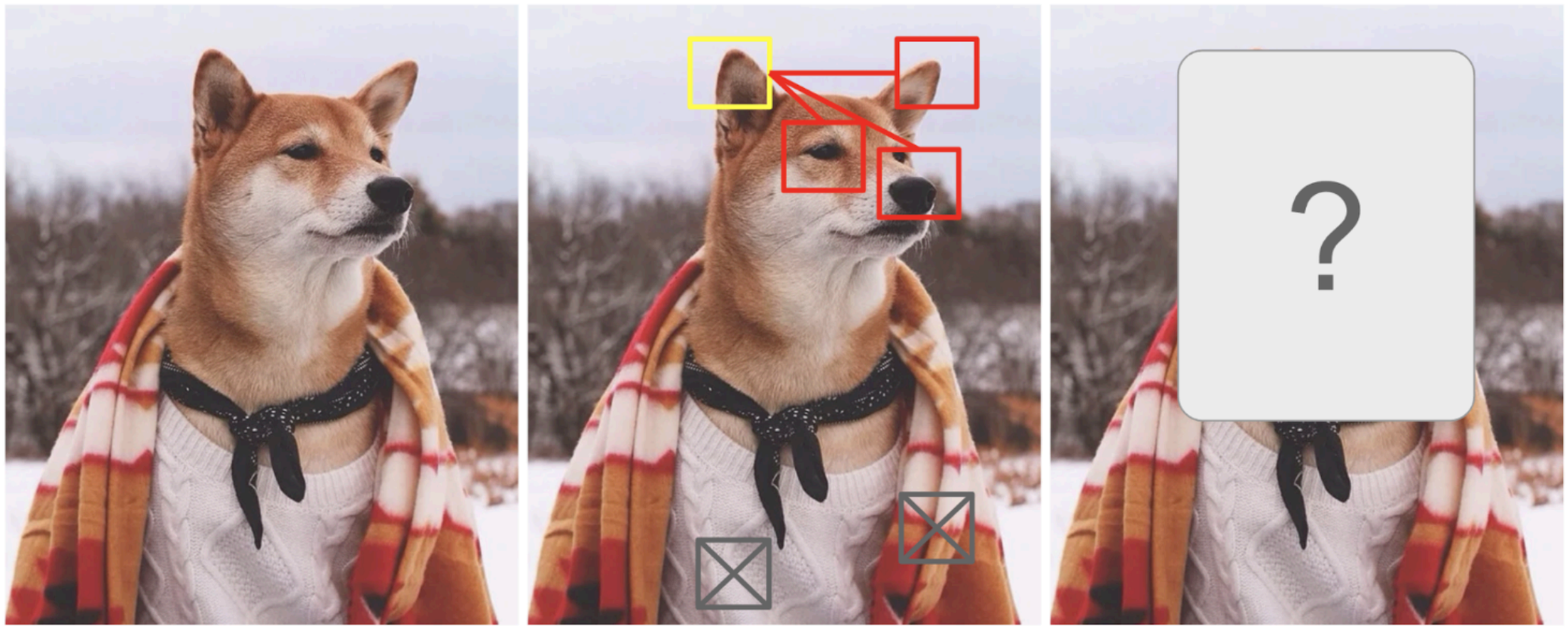


Fig. 1. A Shiba Inu in a men's outfit. The credit of the original photo goes to Instagram @mensweardog.

or correlate words in one sentence?

What is the time?

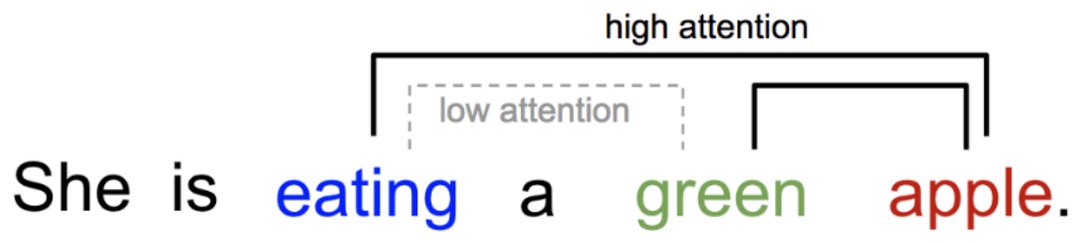
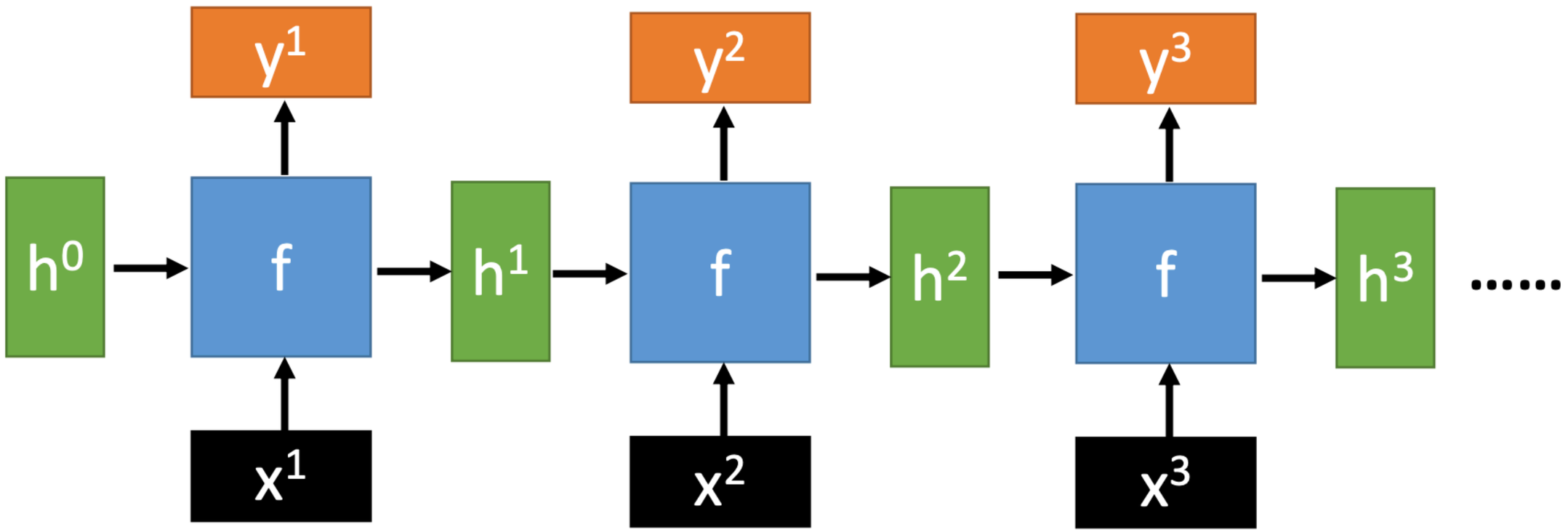


Fig. 2. One word "attends" to other words in the same sentence differently.

Recurrent Neural Network

- Given function f: $h', y = f(h, x)$

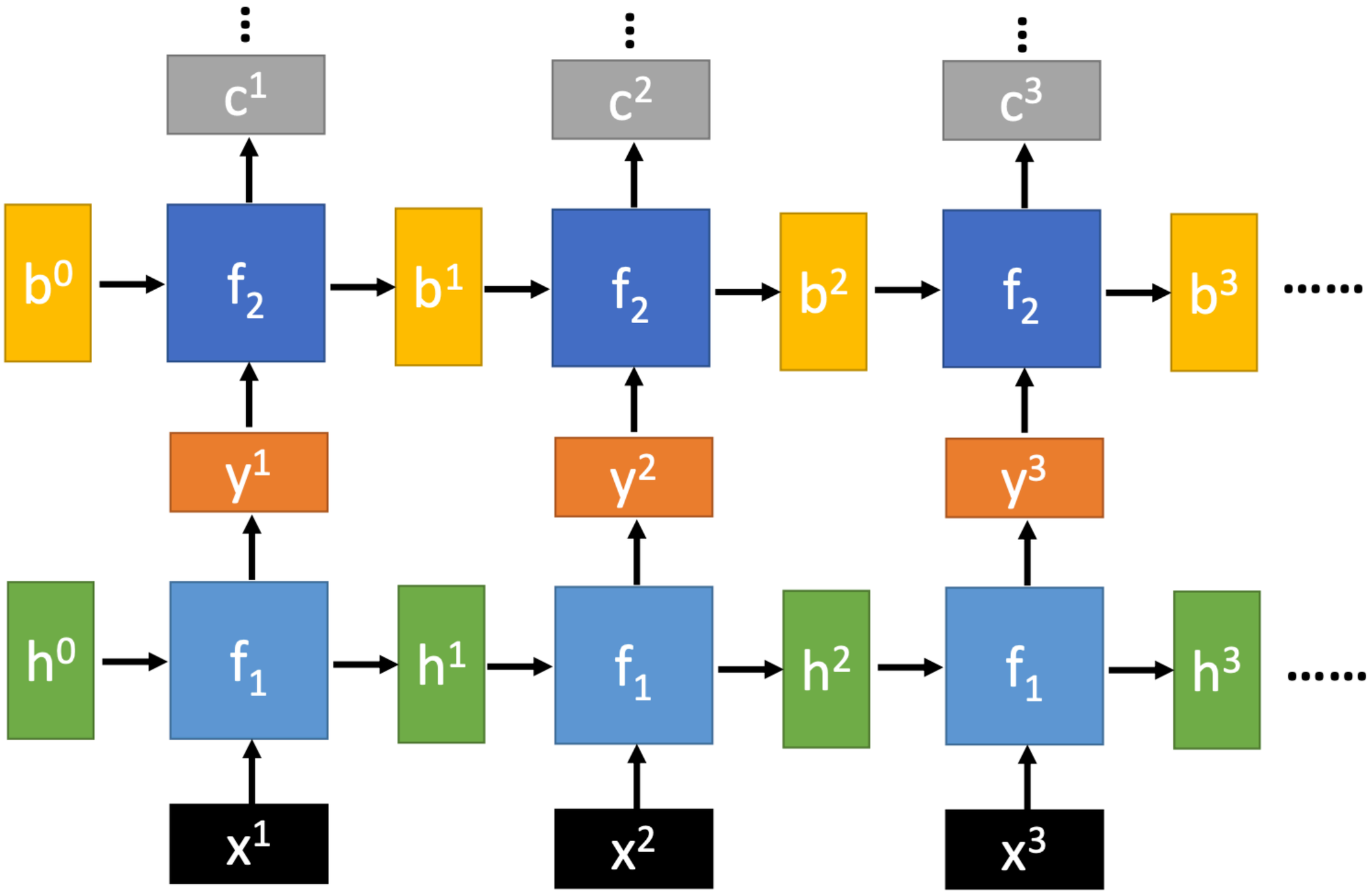
h and h' are vectors with the same dimension



No matter how long the input/output sequence is, we only need one function f

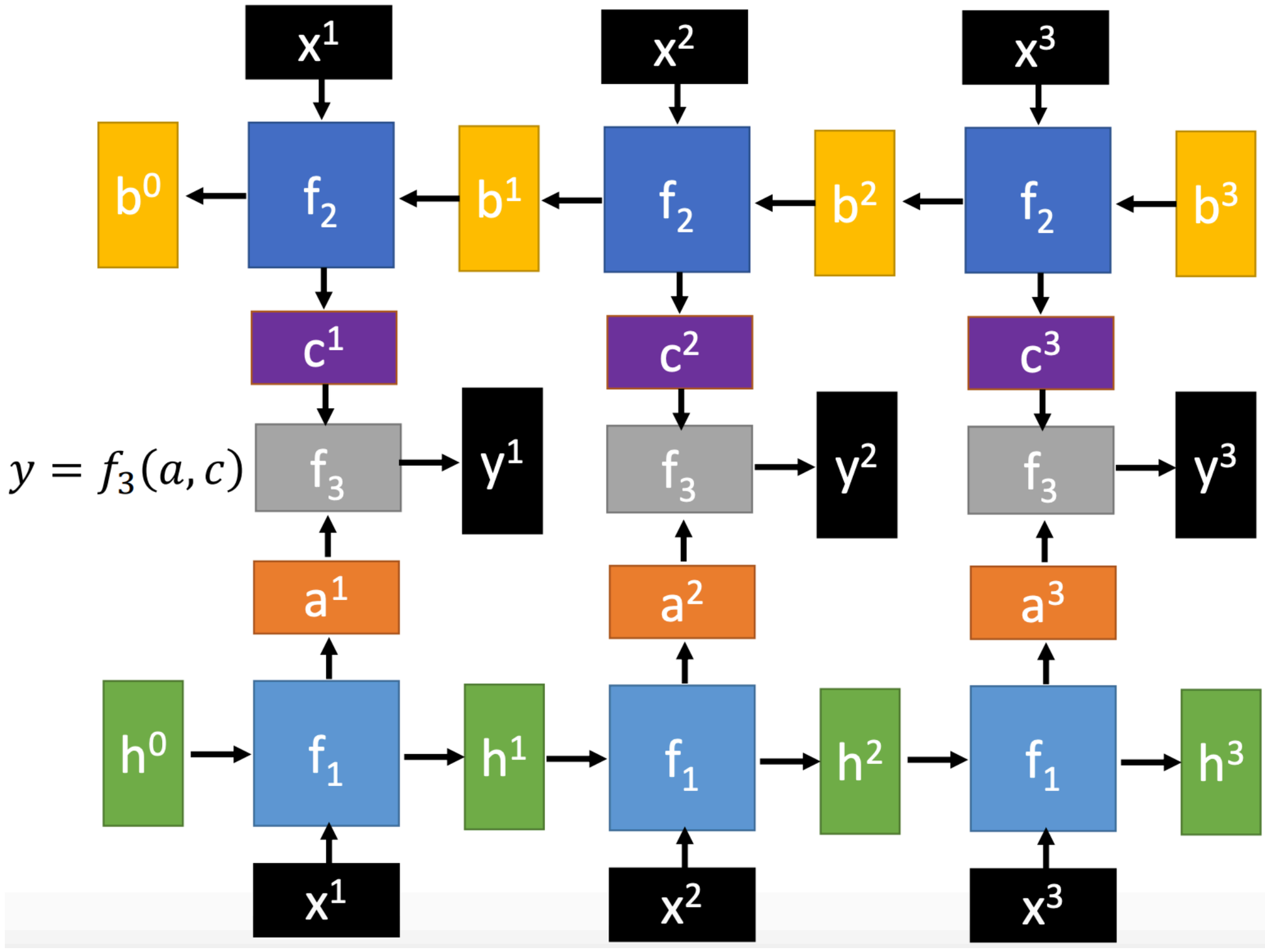
Deep RNN

$$h', y = f_1(h, x) \quad b', c = f_2(b, y) \quad \cdots$$



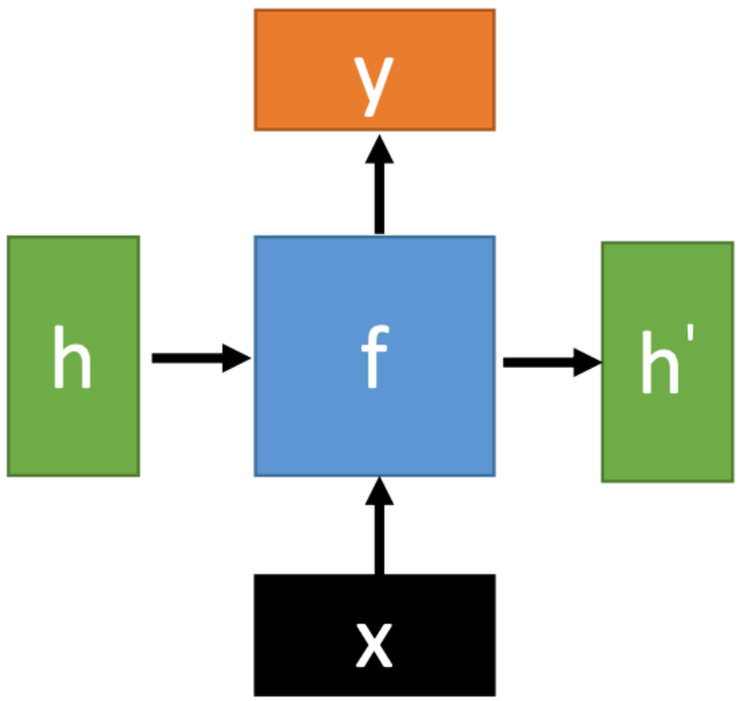
Bidirectional RNN

$$h', a = f_1(h, x) \quad b', c = f_2(b, x)$$



Basic RNN Cell

- Given function f: $h', y = f(h, x)$



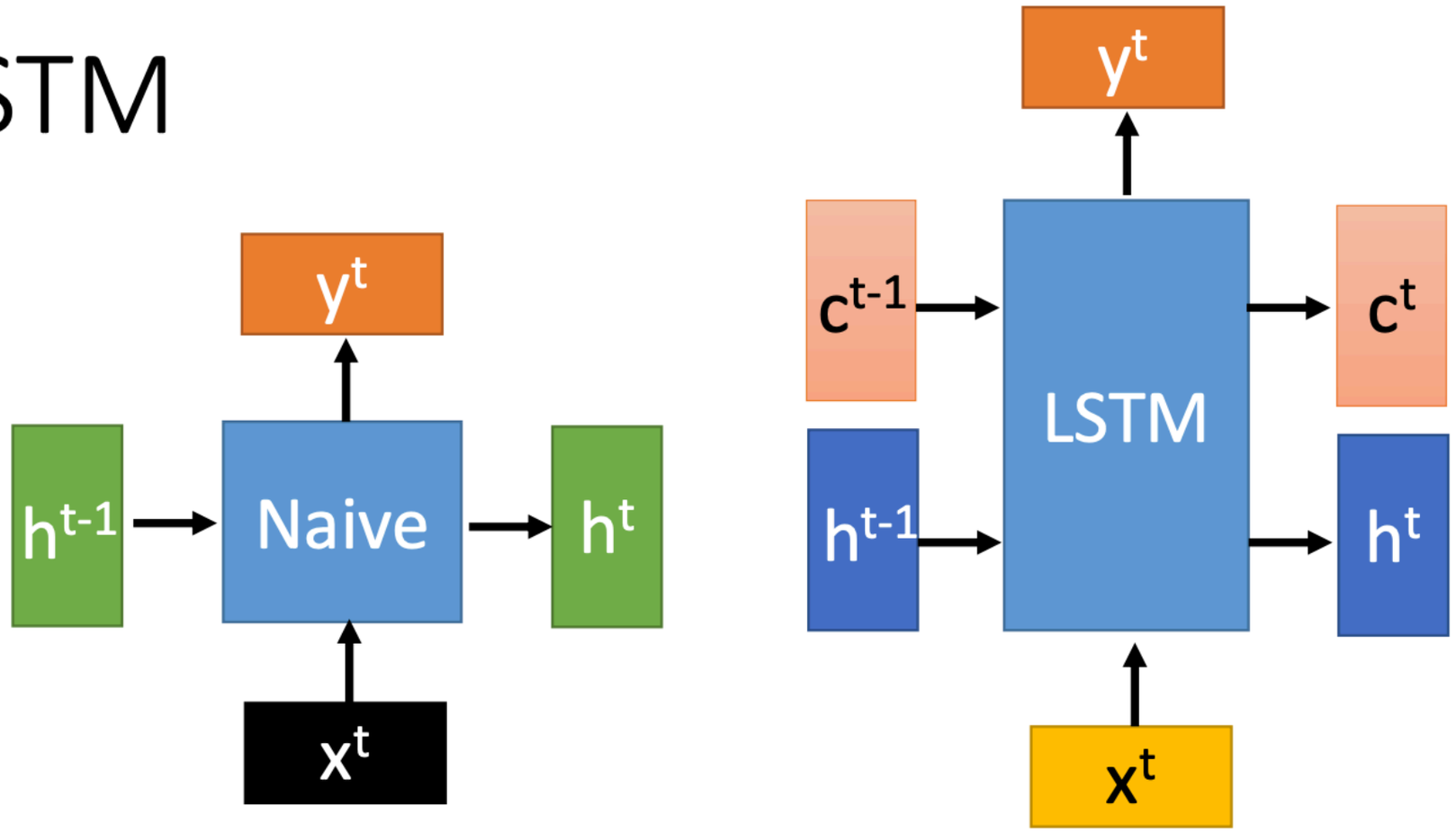
$$h' = \sigma(\ W^h\ h\ +\ W^i\ x\ )$$

$$y = \sigma(\ W^o\ h'\ )$$

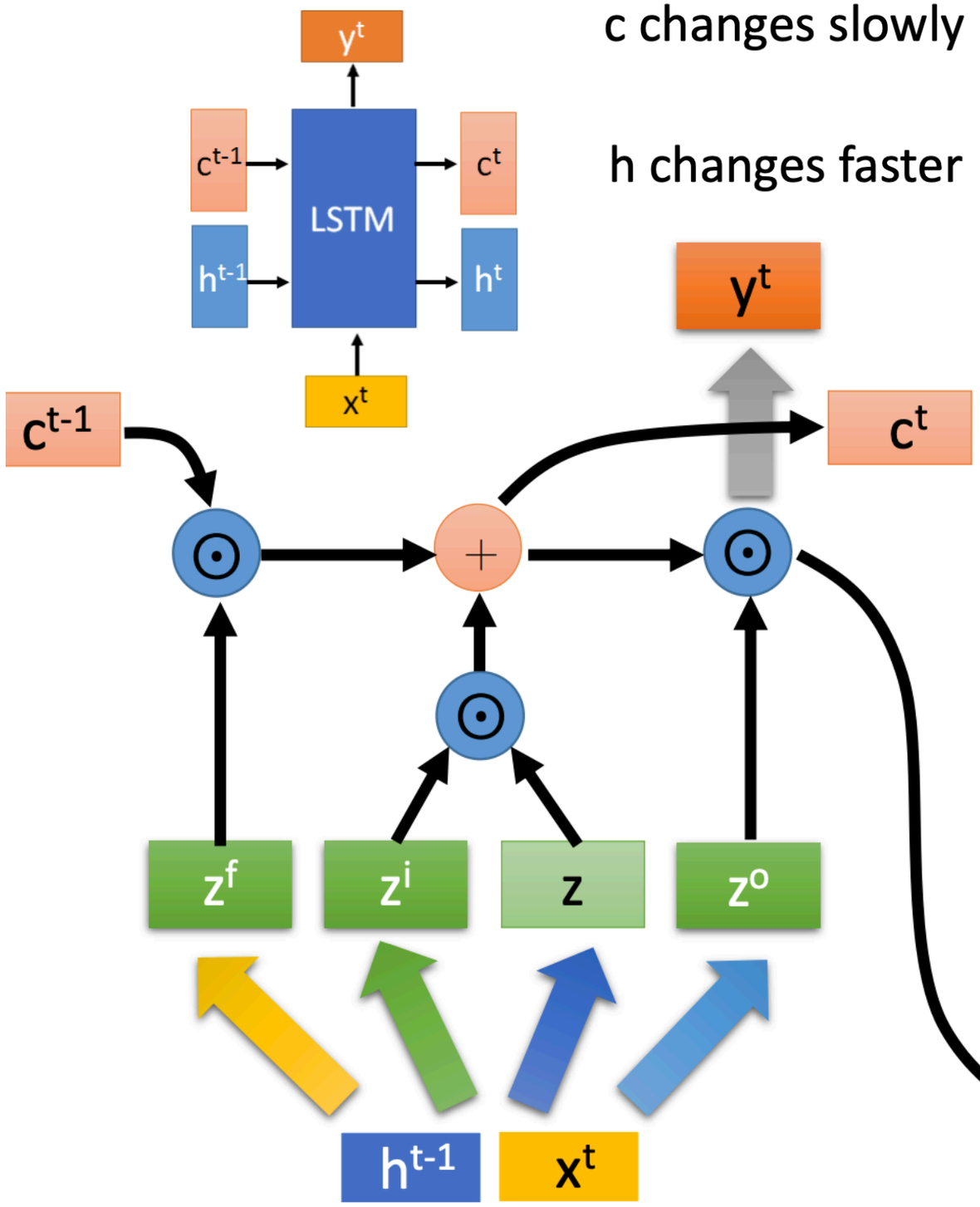
softmax

Ignore bias here

LSTM



So why LSTM works better than RNN?



c changes slowly ➡ c^t is c^{t-1} added by something

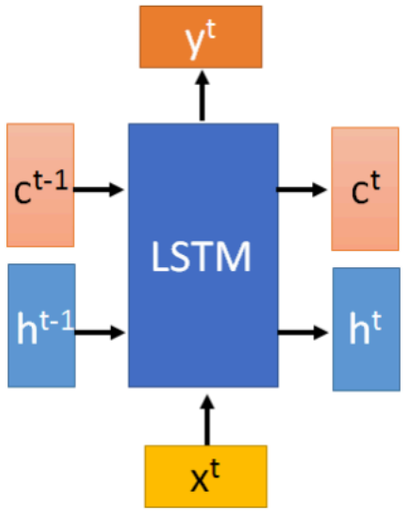
h changes faster ➡ h^t and h^{t-1} can be very different

$$c^t = z^f \odot c^{t-1} + z^i \odot z$$

$$h^t = z^o \odot tanh(c^t)$$

$$y^t = \sigma(W' h^t)$$

C can keep a long memory ! Better performance in long sequence!
vanishing gradient problem !



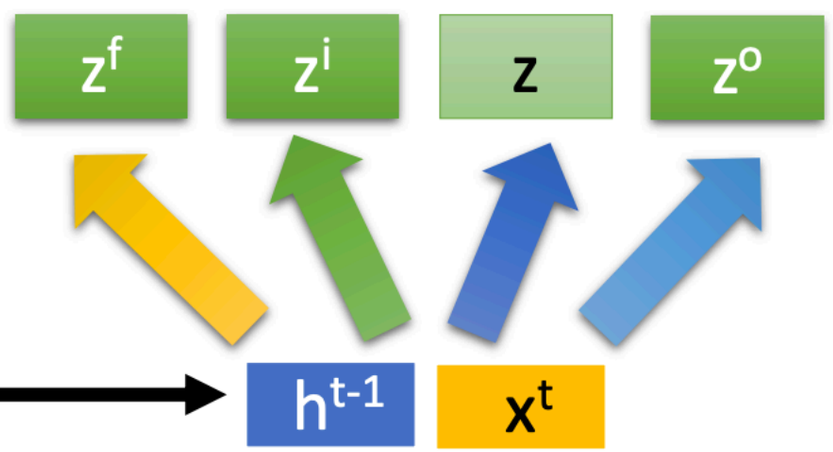
$$z = tanh\left(\begin{bmatrix} W & \\ & \end{bmatrix} \begin{bmatrix} x^t \\ h^{t-1} \\ c^{t-1} \end{bmatrix} \right)$$

diagonal

c^{t-1}

"peephole"

z^o z^f z^i obtained by the same way

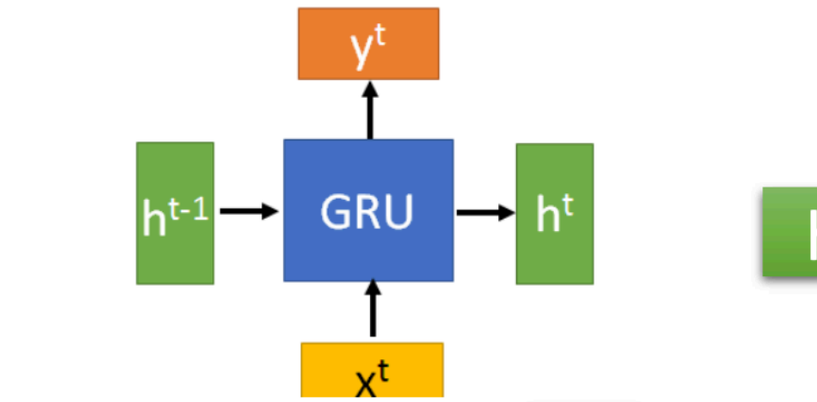


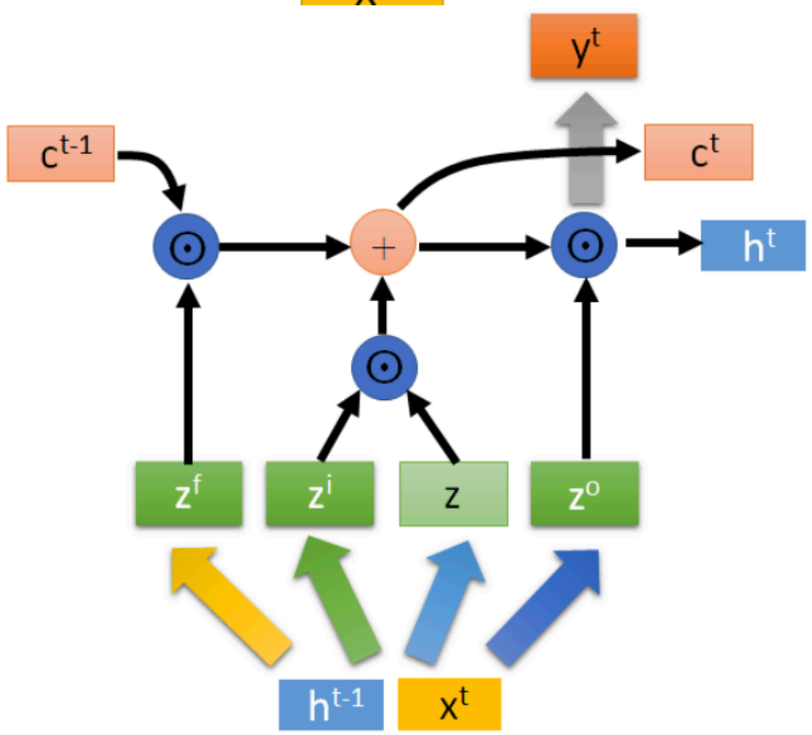
We more familiar with

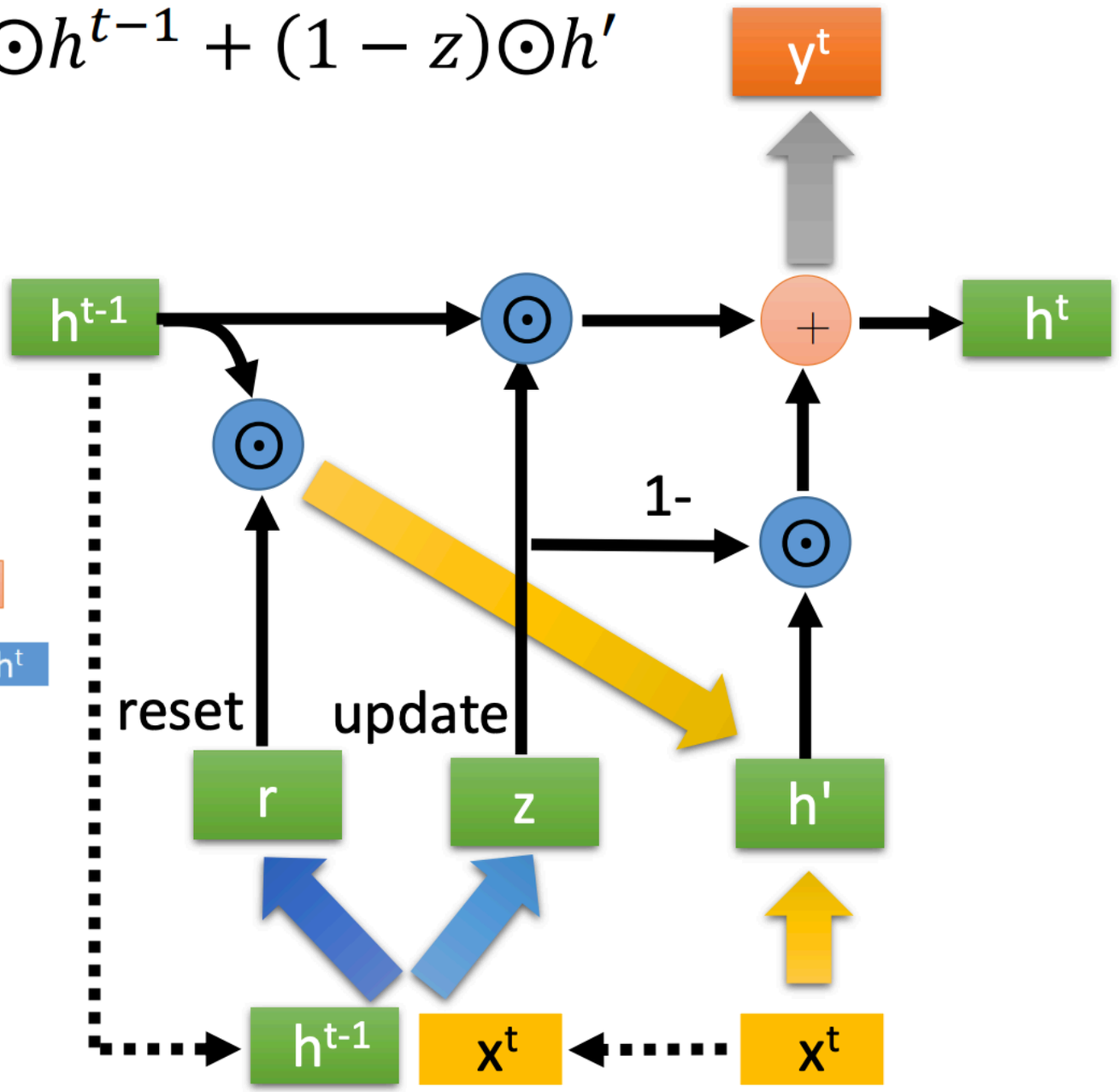
$$z = tanh\left(W \begin{bmatrix} x^t \\ h^{t-1} \end{bmatrix} \right)$$

GRU

$$h^t = z \odot h^{t-1} + (1 - z) \odot h'$$

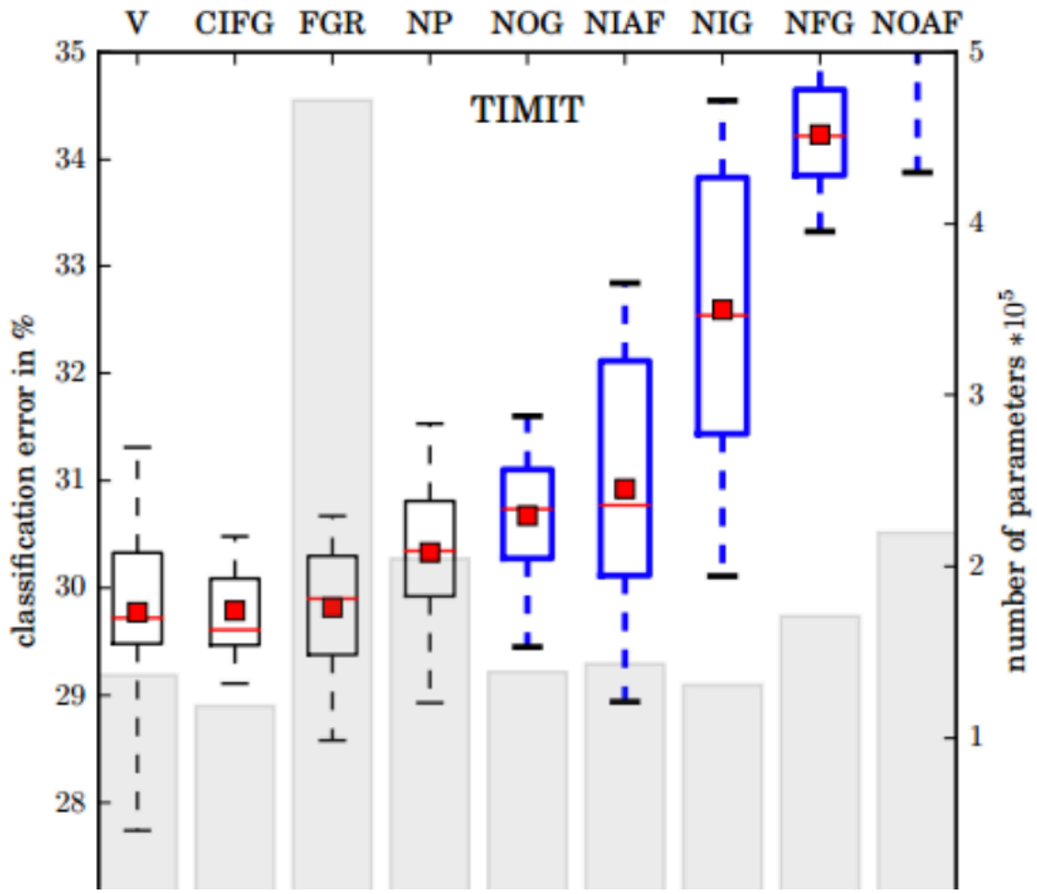






LSTM: A Search Space Odyssey

1. No Input Gate (NIG)
2. No Forget Gate (NFG)
3. No Output Gate (NOG)
4. No Input Activation Function (NIAF)
5. No Output Activation Function (NOAF)
6. No Peepholes (NP)
7. Coupled Input and Forget Gate (CIFG)
8. Full Gate Recurrence (FGR)



Standard LSTM works well

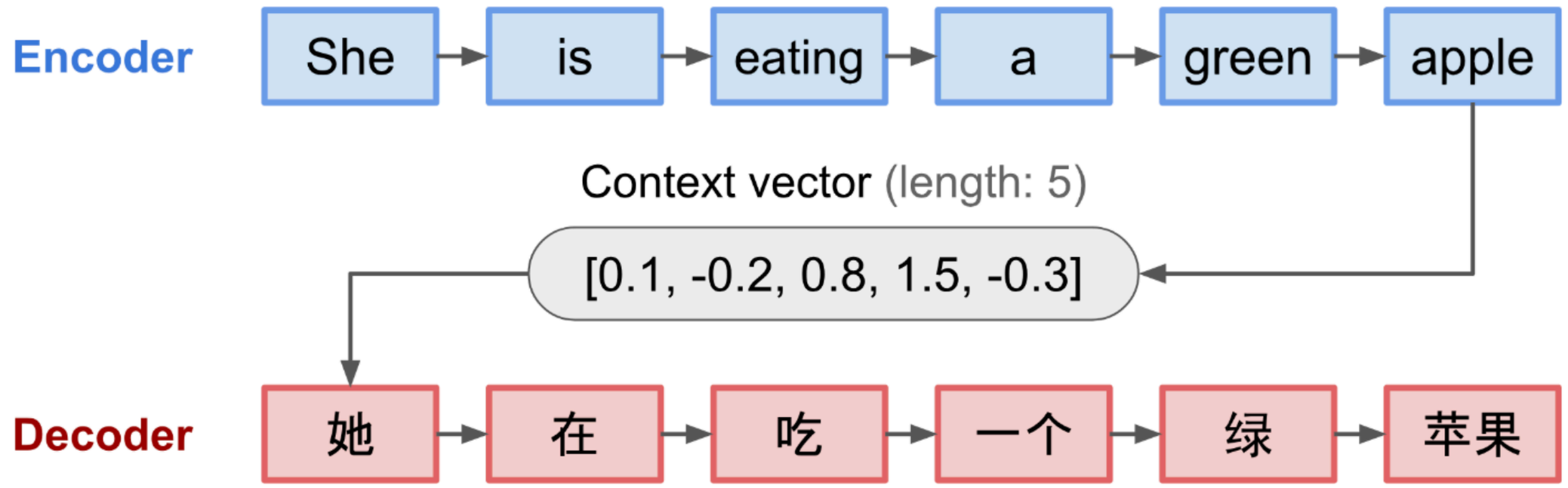
Simply LSTM: coupling input and forget gate, removing peephole

Forget gate is critical for performance

Output gate activation function is critical

seq2seq model

Both the encoder and decoder are recurrent neural networks, i.e. using LSTM or GRU units.



A critical and apparent disadvantage of this fixed-length context vector design is incapability of remembering long sentences.

- encoder hidden states;
- decoder hidden states;
- alignment between source and target.

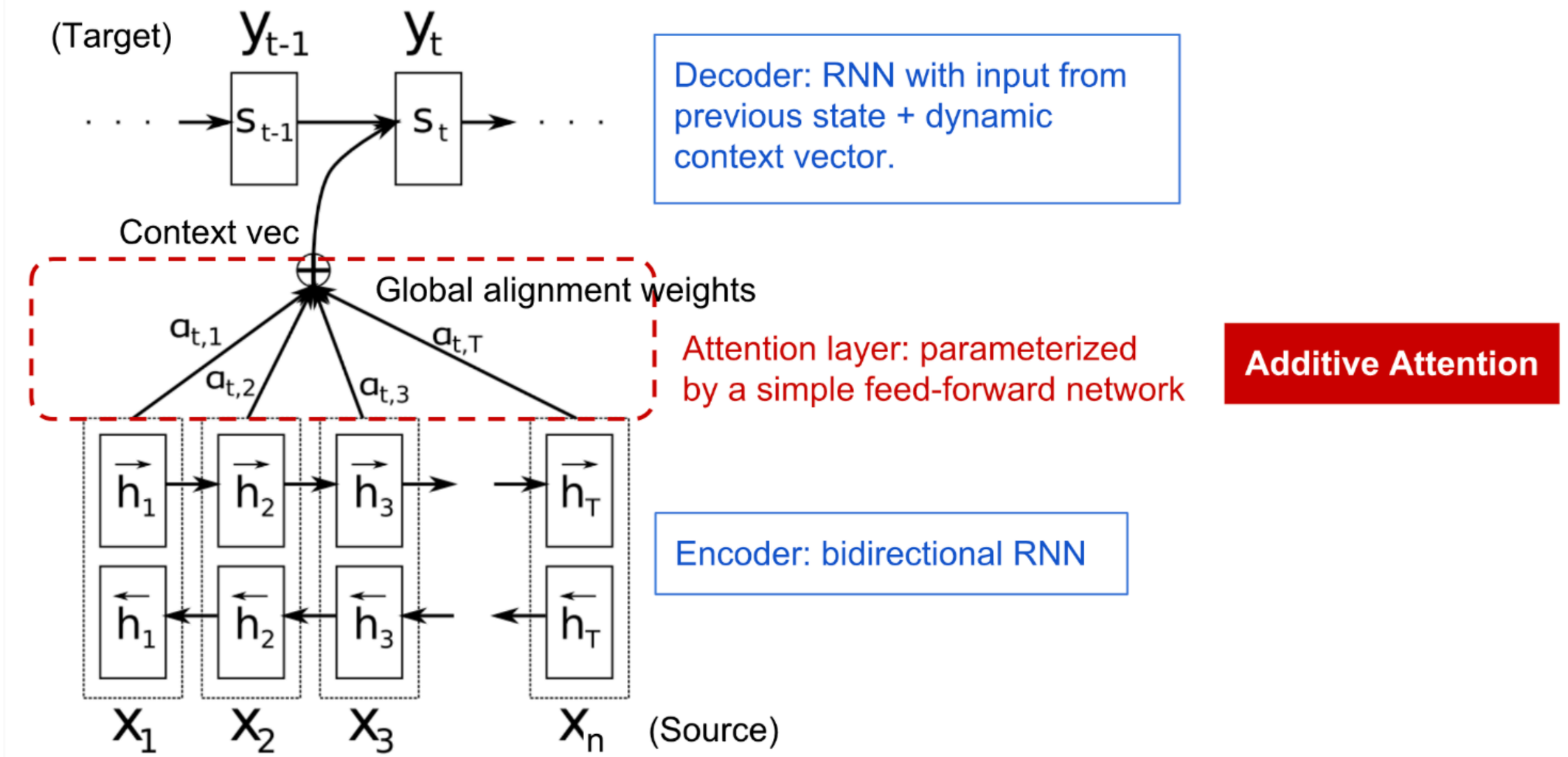
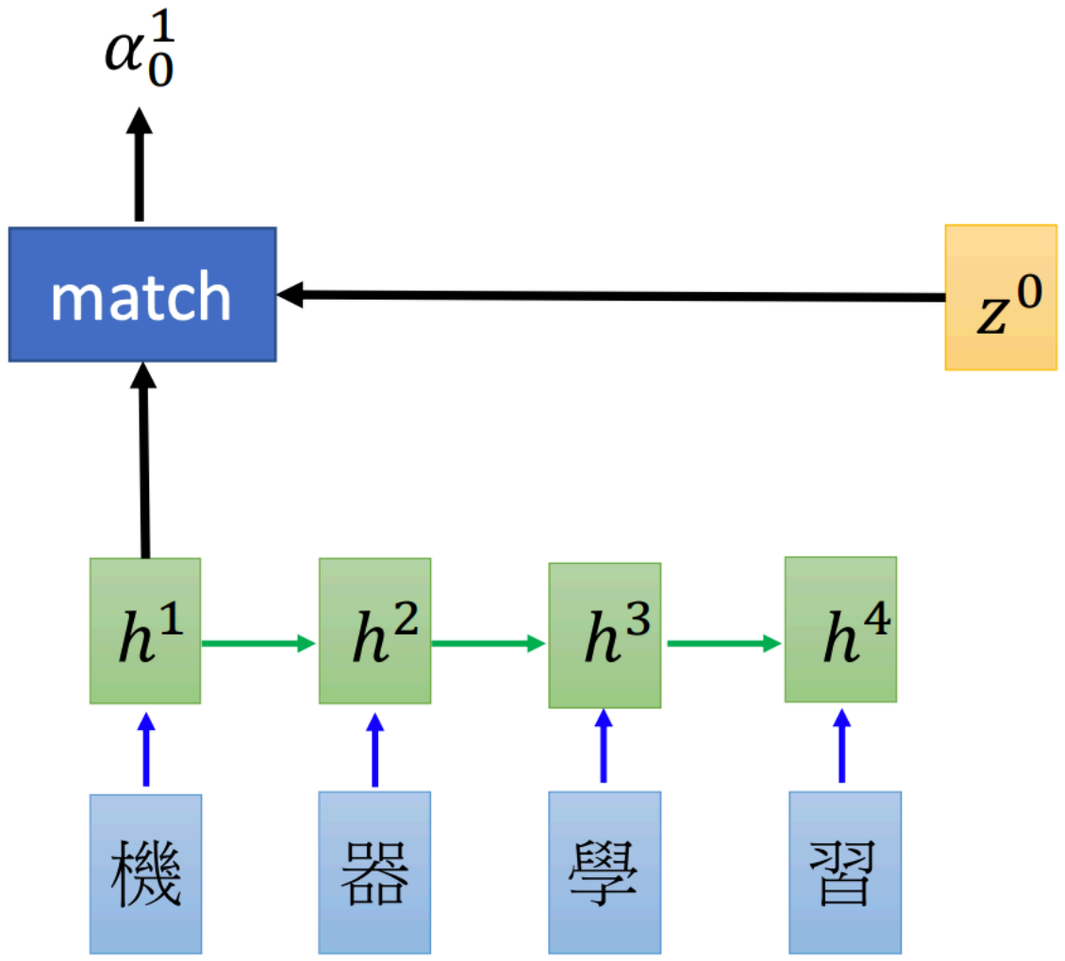


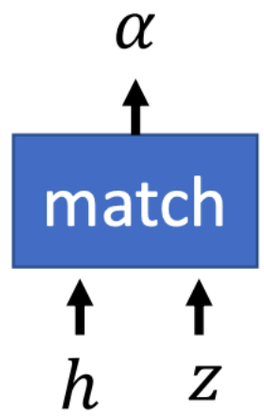
Fig. 4. The encoder-decoder model with additive attention mechanism in Bahdanau et al., 2015.

Machine Translation

- Attention-based model



Jointly learned with other part of the network

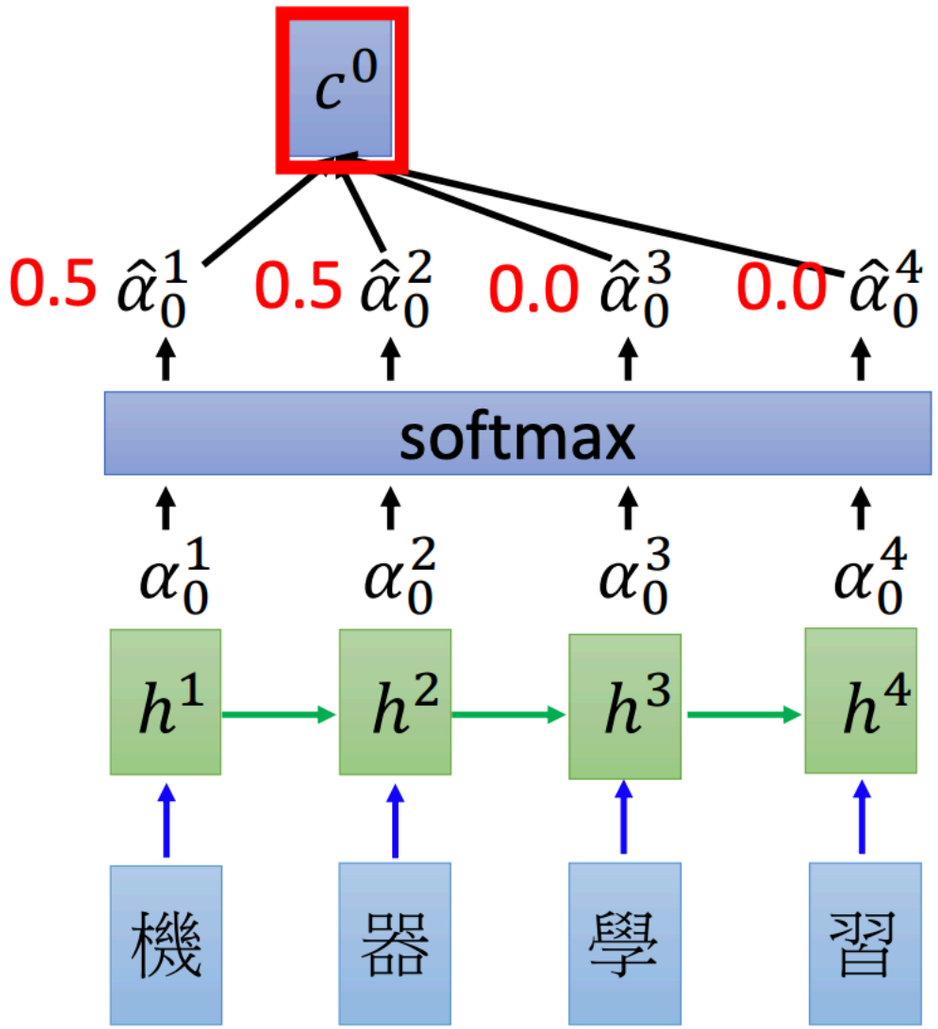


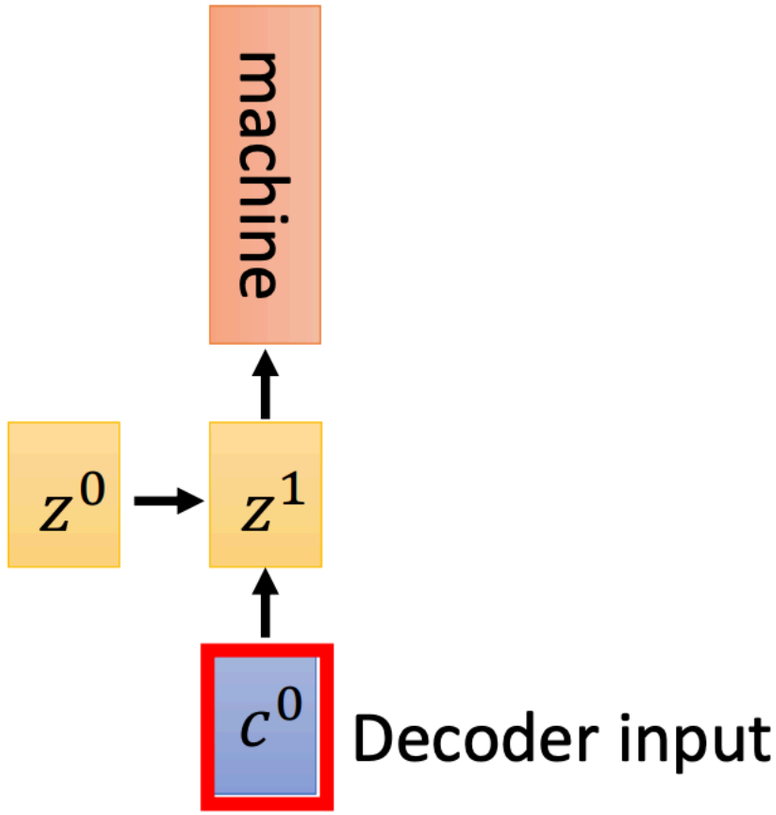
What is match ?

Design by yourself

- Cosine similarity of z and h
- Small NN whose input is z and h, output a scalar
- $\alpha = h^T W z$

- Attention-based model

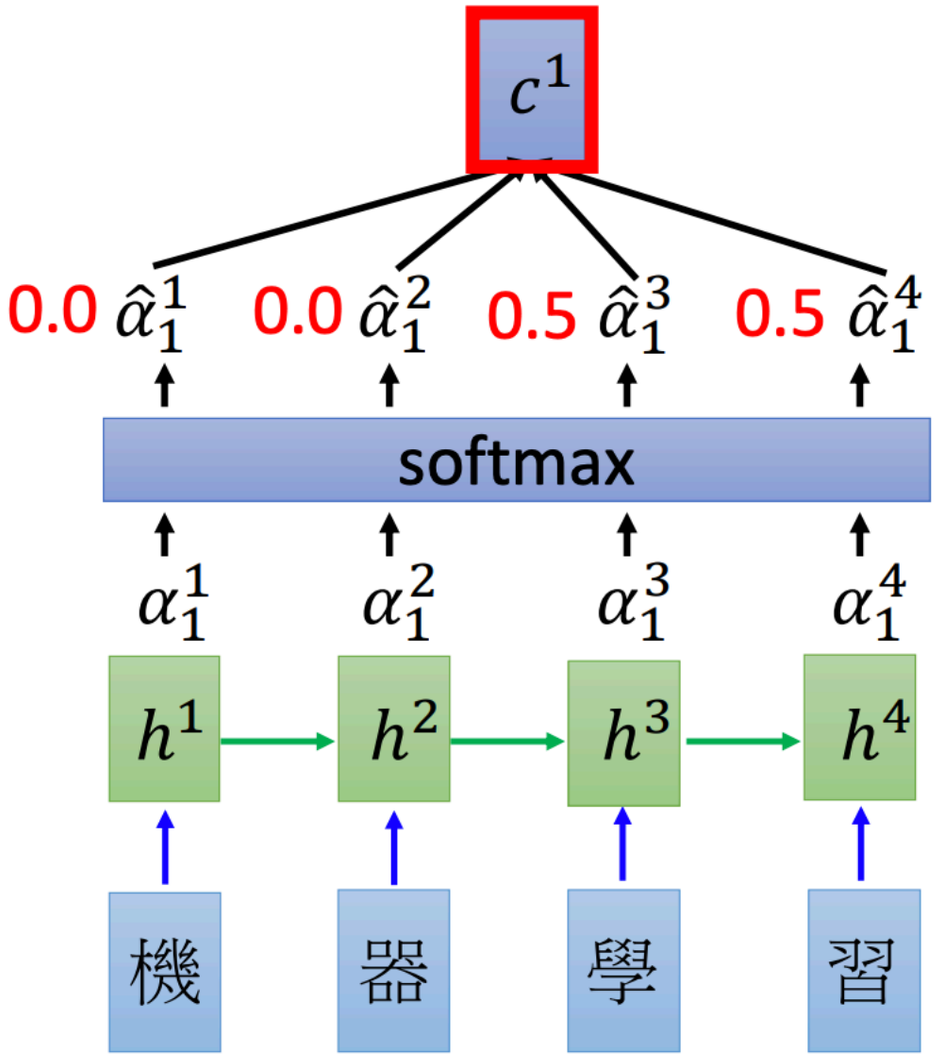


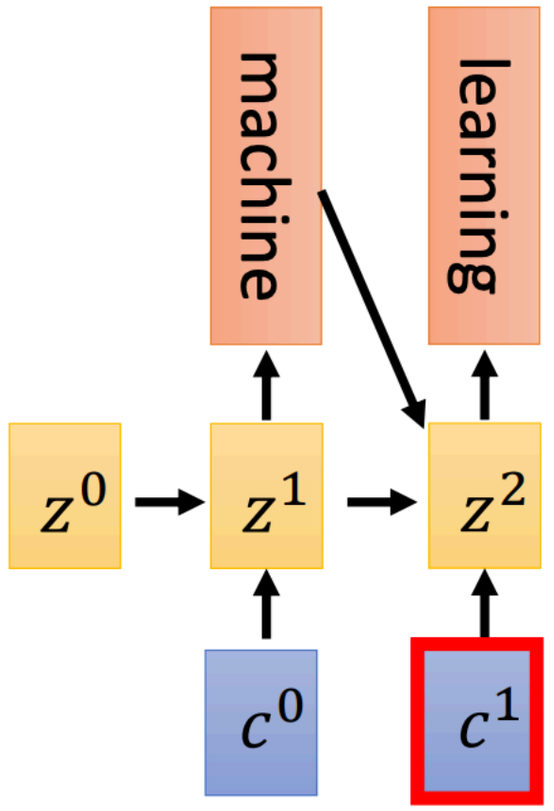


$$c^0 = \sum \hat{\alpha}_0^i h^i$$

$$= 0.5h^1 + 0.5h^2$$

- Attention-based model





$$c^1 = \sum \hat{\alpha}_1^i h^i$$

$$= 0.5h^3 + 0.5h^4$$

The same process repeat until generating

<EOS>

Summary

Below is a summary table of several popular attention mechanisms (or broader categories of attention mechanisms).

Name	Alignment score function	Citation
Content-base attention		Graves2014
Additive(*)		Bahdanau2015
Location-Base	Note: This simplifies the softmax alignment to only depend on the target position.	Luong2015
General	where is a trainable weight matrix in the attention layer.	Luong2015
Dot-Product		Luong2015
Scaled Dot-Product(^)	Note: very similar to the dot-product attention except for a scaling factor; where n is the dimension of the source hidden state.	Vaswani2017
Self-Attention(&)	Relating different positions of the same input sequence. Theoretically the self-attention can adopt any score functions above, but just replace the target sequence with the same input sequence.	Cheng2016
Global/Soft	Attending to the entire input state space.	Xu2015
Local/Hard	Attending to the part of input state space; i.e. a patch of the input image.	Xu2015; Luong2015

Self-Attention

Self-attention, also known as **intra-attention**, is an attention mechanism relating different positions of a single sequence in order to compute a representation of the same sequence. It has been shown to be very useful in machine reading, abstractive summarization, or image description generation.

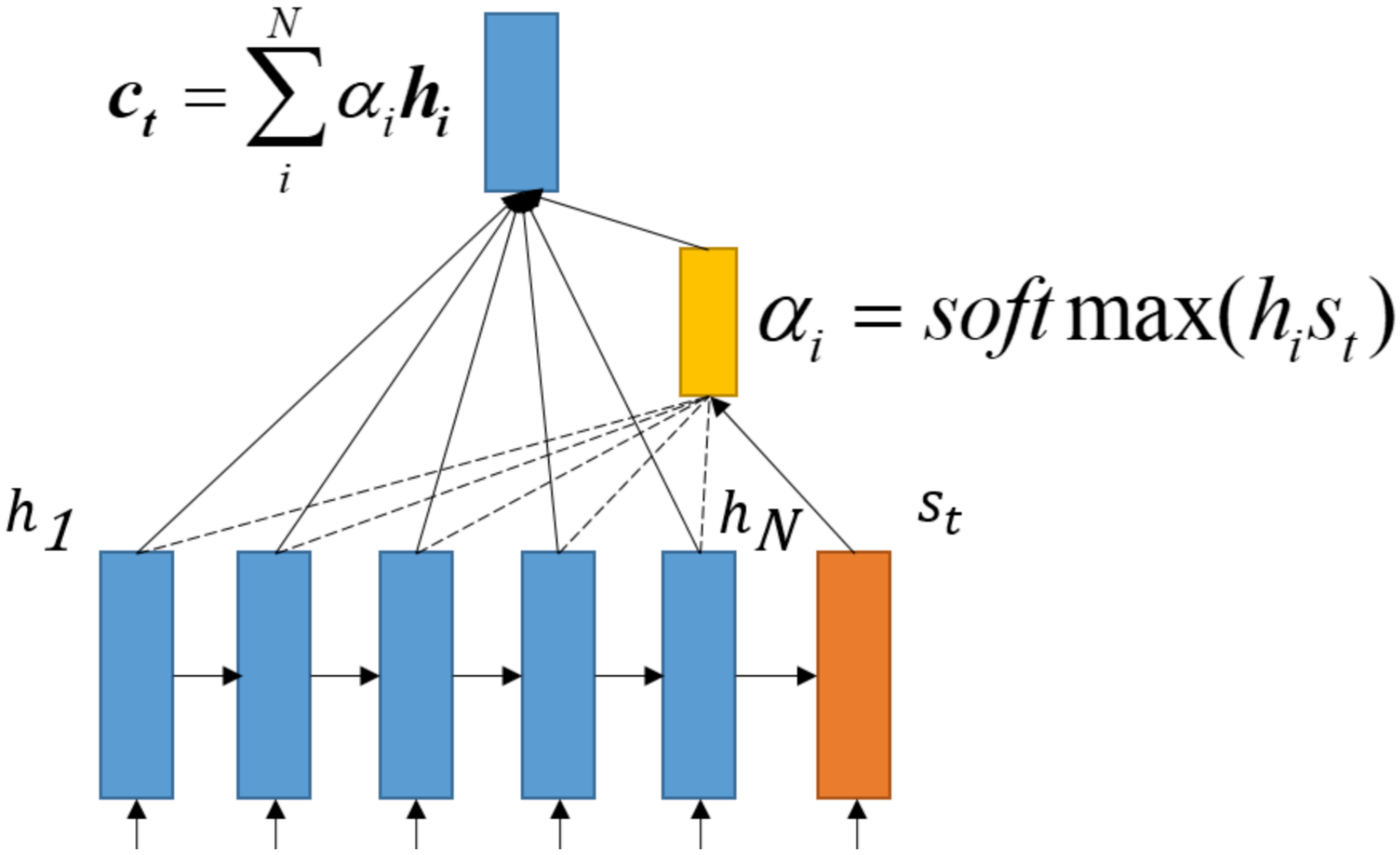
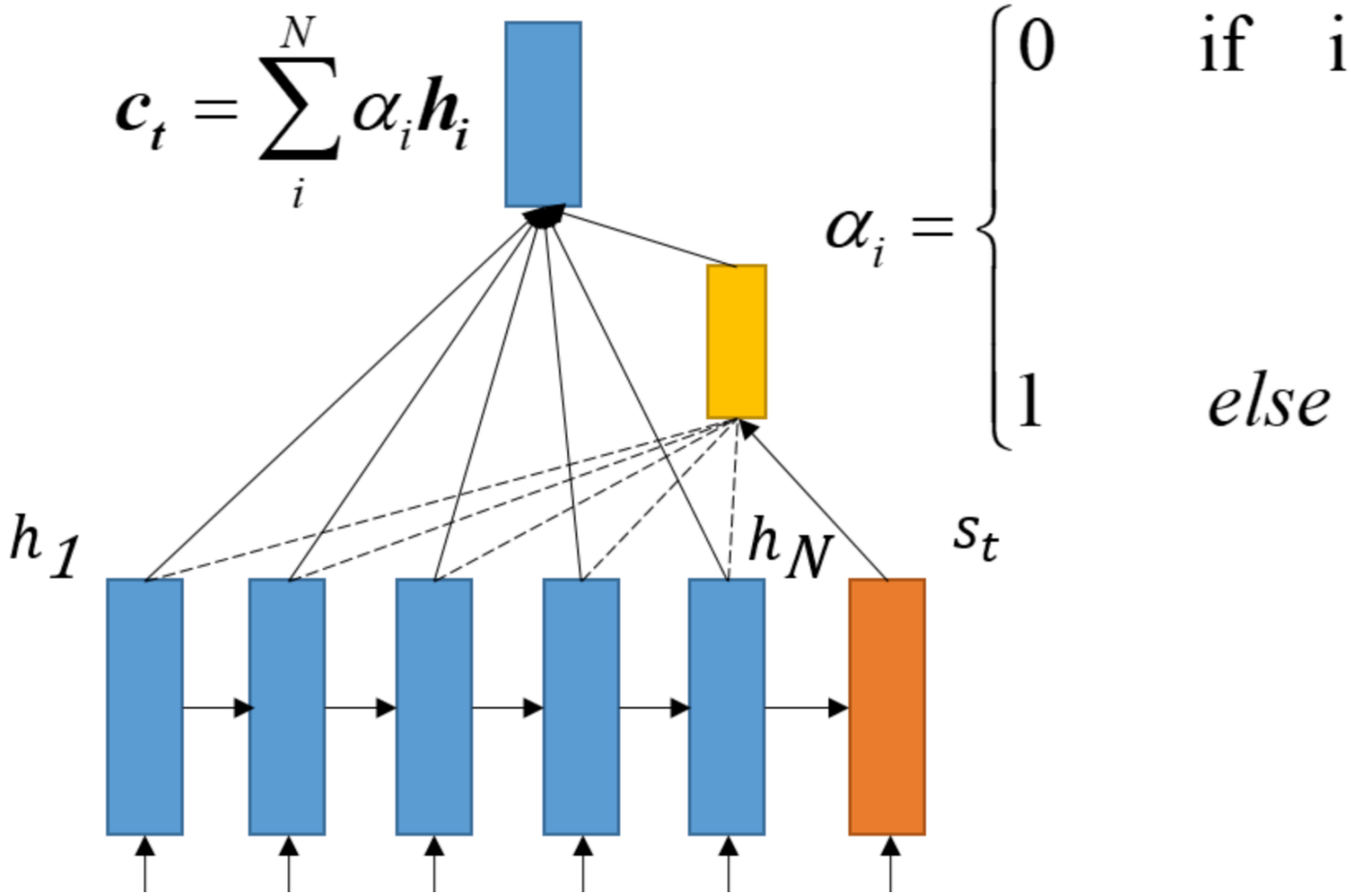


Fig. 7. "A woman is throwing a frisbee in a park." (Image source: Fig. 6(b) in Xu et al. 2015)

http://proceedings.mlr.press/v37/xuc15.pdf

Soft vs Hard Attention

- **Soft** Attention: the alignment weights are learned and placed "softly" over all patches in the source image; essentially the same idea as in Bahdanau et al., 2015.
  - Pro: the model is smooth and differentiable.
  - Con: expensive when the source input is large.
- **Hard** Attention: only selects one patch of the image to attend to at a time.
  - Pro: less calculation at the inference time.
  - Con: the model is non-differentiable and requires more complicated techniques such as variance reduction or reinforcement learning to train. (Luong, et al., 2015)



http://proceedings.mlr.press/v37/xuc15.pdf

Global vs Local Attention

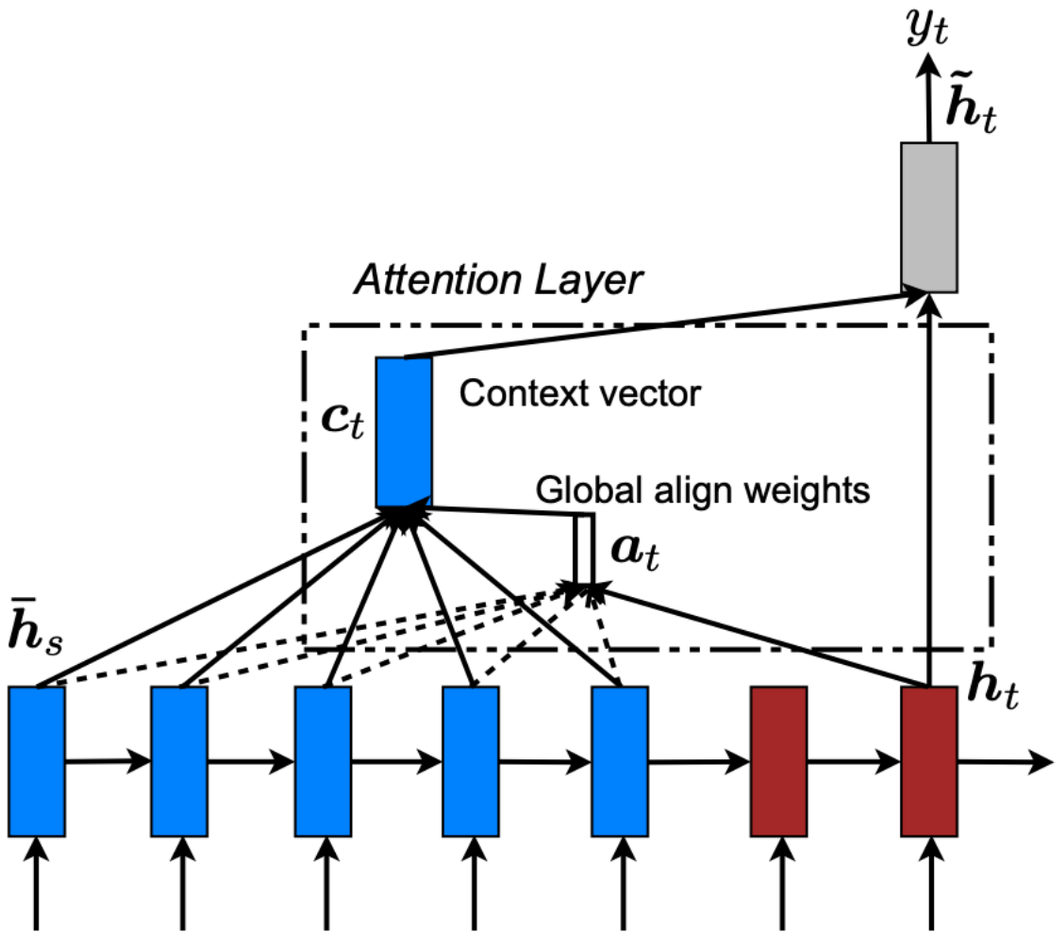


Figure 2: **Global attentional model** – at each time step t, the model infers a *variable-length* alignment weight vector $\boldsymbol{a}_t$ based on the current target state $\boldsymbol{h}_t$ and all source states $\bar{\boldsymbol{h}}_s$. A global context vector $\boldsymbol{c}_t$ is then computed as the weighted average, according to $\boldsymbol{a}_t$, over all the source states.

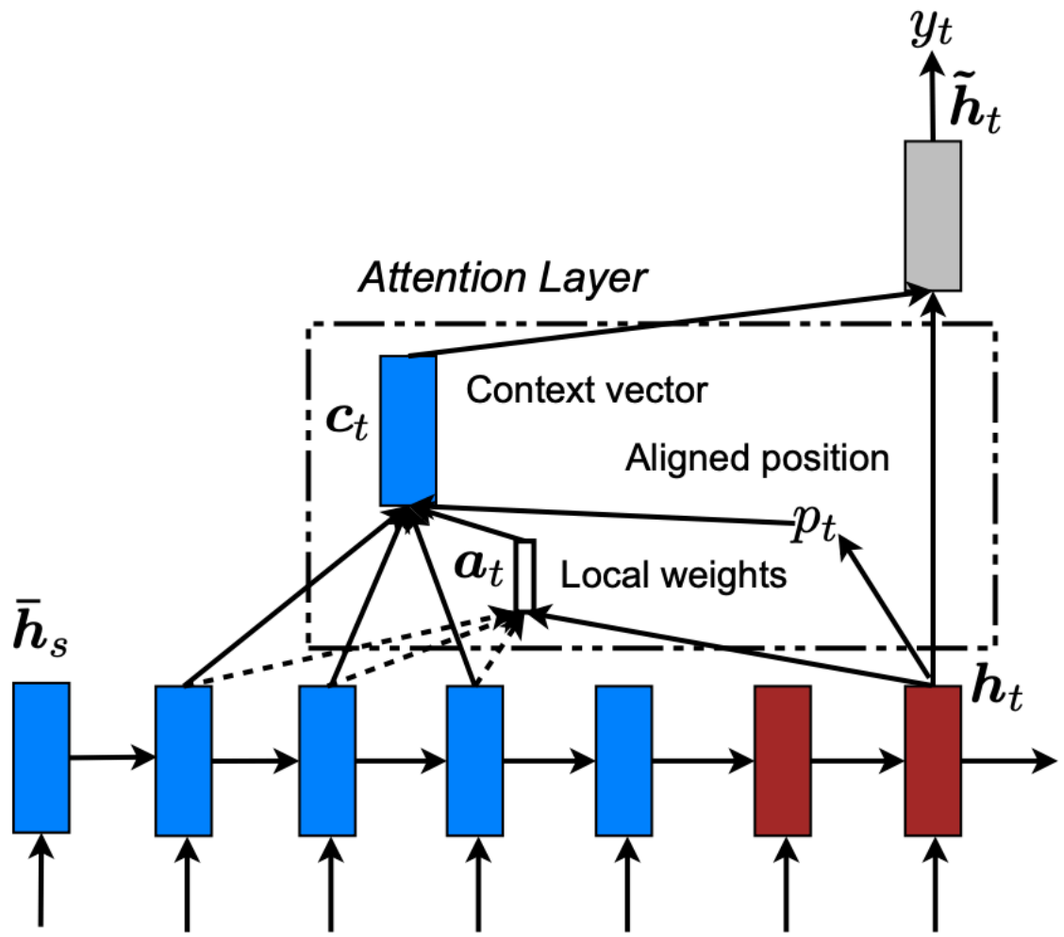


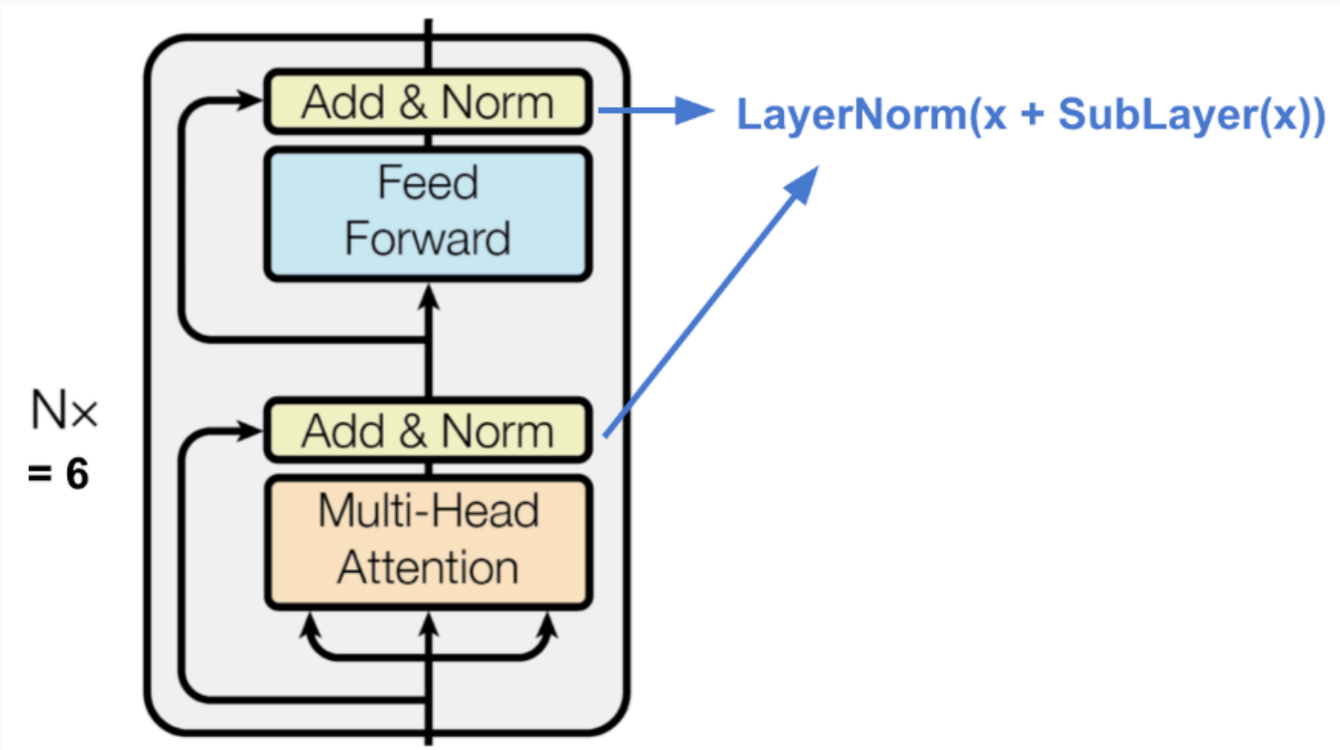
Figure 3: **Local attention model** – the model first predicts a single aligned position p_t for the current target word. A window centered around the source position p_t is then used to compute a context vector $\boldsymbol{c}_t$, a weighted average of the source hidden states in the window. The weights $\boldsymbol{a}_t$ are inferred from the current target state $\boldsymbol{h}_t$ and those source states $\bar{\boldsymbol{h}}_s$ in the window.

https://arxiv.org/pdf/1508.04025.pdf

Thanks && QA

Attention is all you need.

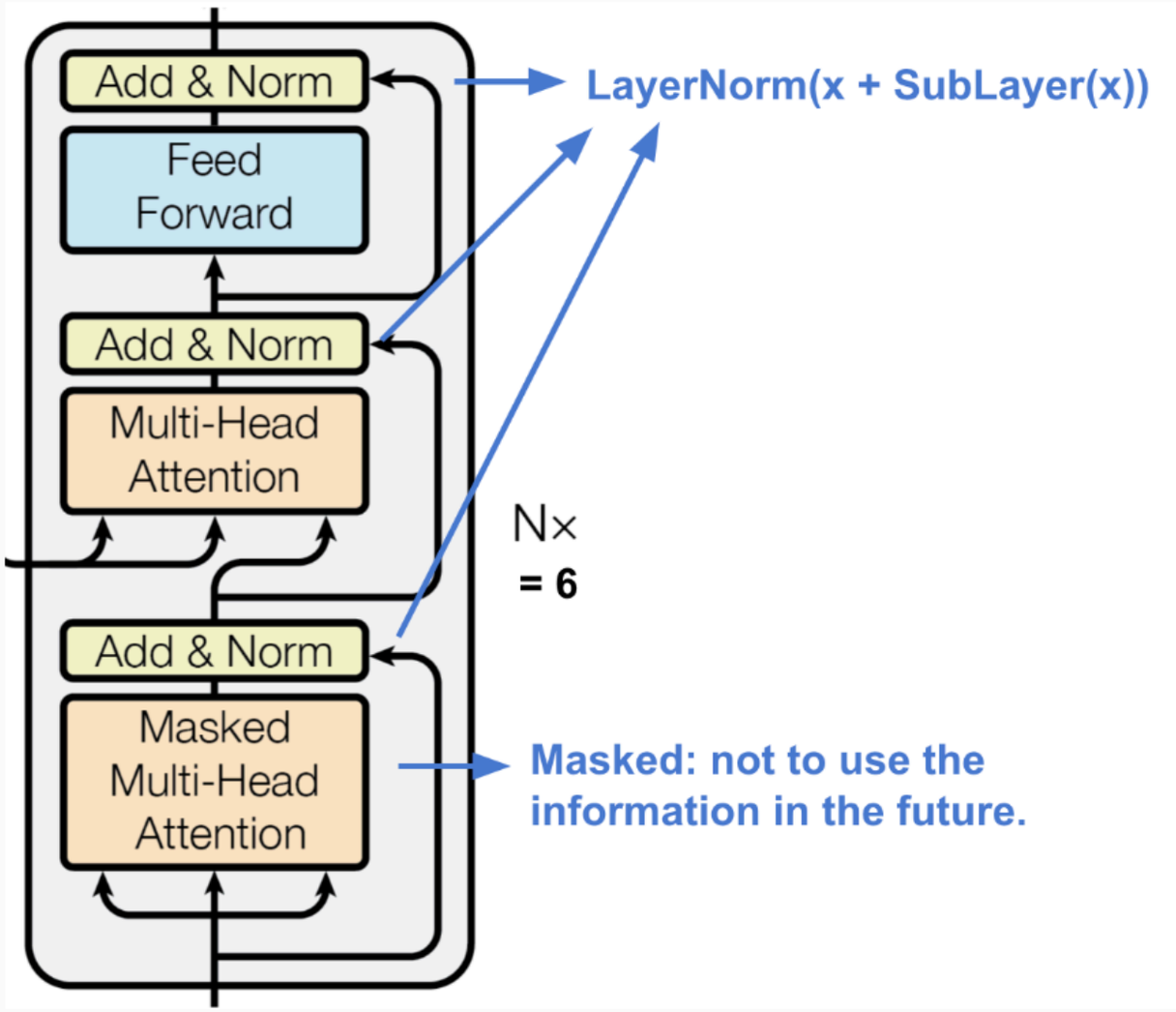
Encoder



The encoder generates an attention-based representation with capability to locate a specific piece of information from a potentially infinitely-large context.

- A stack of N=6 identical layers.
- Each layer has a **multi-head self-attention layer** and a simple position-wise **fully connected feed-forward network**.
- Each sub-layer adopts a **residual** connection and a layer **normalization**. All the sub-layers output data of the same dimension .

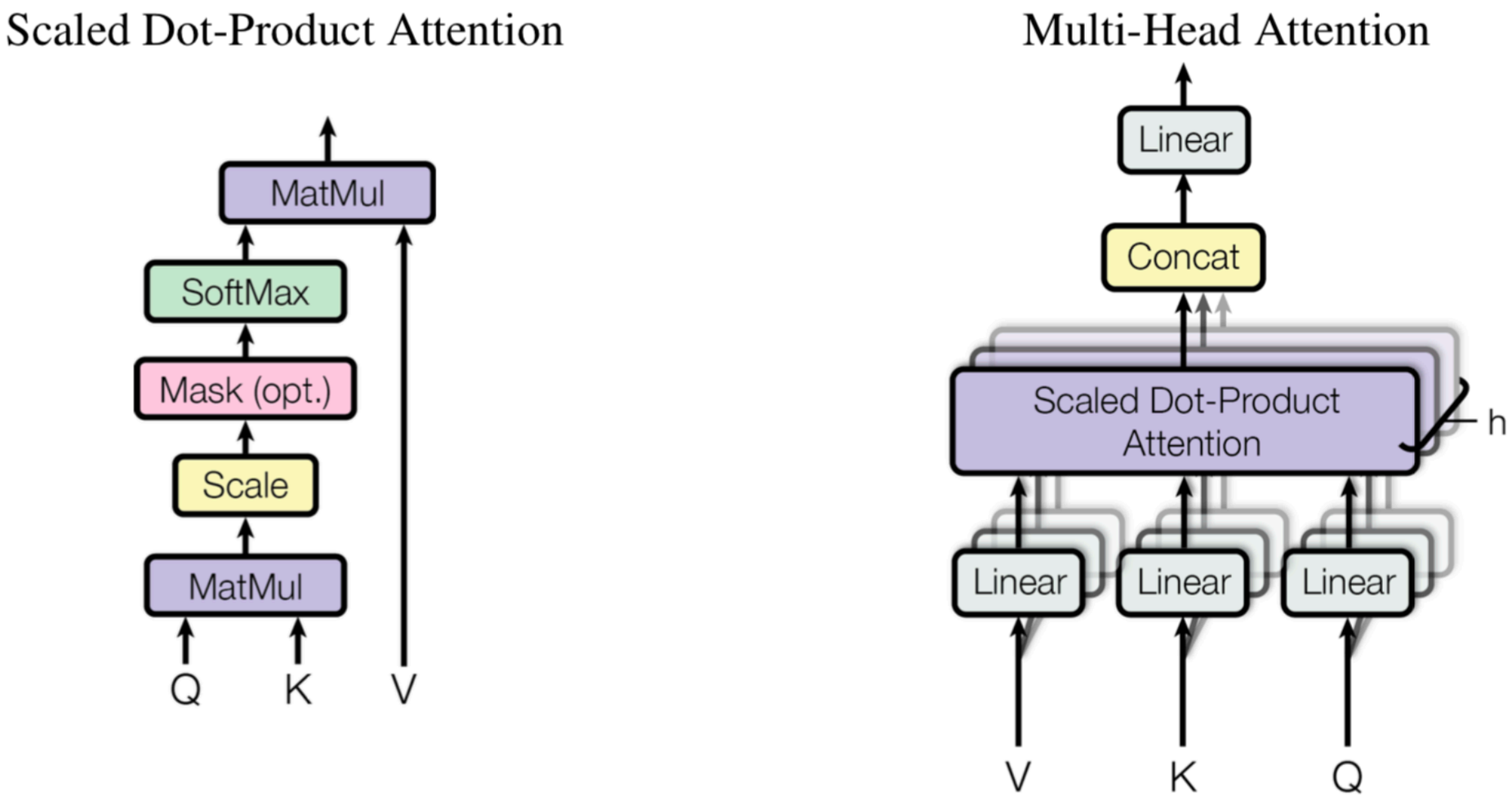
Decoder



The decoder is able to retrieval from the encoded representation.

- A stack of N = 6 identical layers
- Each layer has two sub-layers of multi-head attention mechanisms and one sub-layer of fully-connected feed-forward network.
- Similar to the encoder, each sub-layer adopts a residual connection and a layer normalization.
- The first multi-head attention sub-layer is **modified** to prevent positions from attending to subsequent positions, as we don't want to look into the future of the target sequence when predicting the current position.

Multi-Head Self-Attention

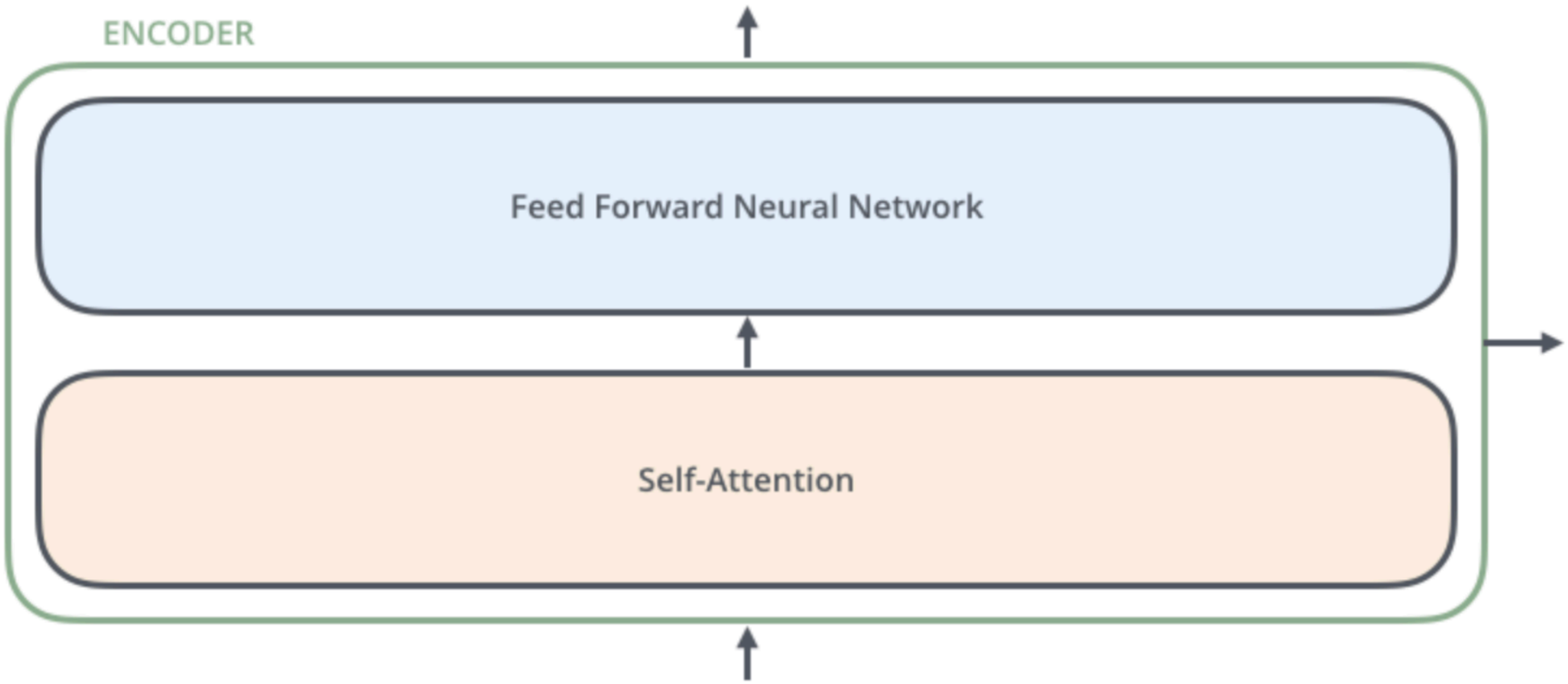


Rather than only computing the attention once, the multi-head mechanism runs through the scaled dot-product attention multiple times in parallel. The independent attention outputs are simply concatenated and linearly transformed into the expected dimensions. I assume the motivation is because ensembling always helps? ;) According to the paper, *"multi-head attention allows the model to jointly attend to information from different representation* ***subspaces*** *at different positions. With a single attention head, averaging inhibits this."*

http://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf

Encoder

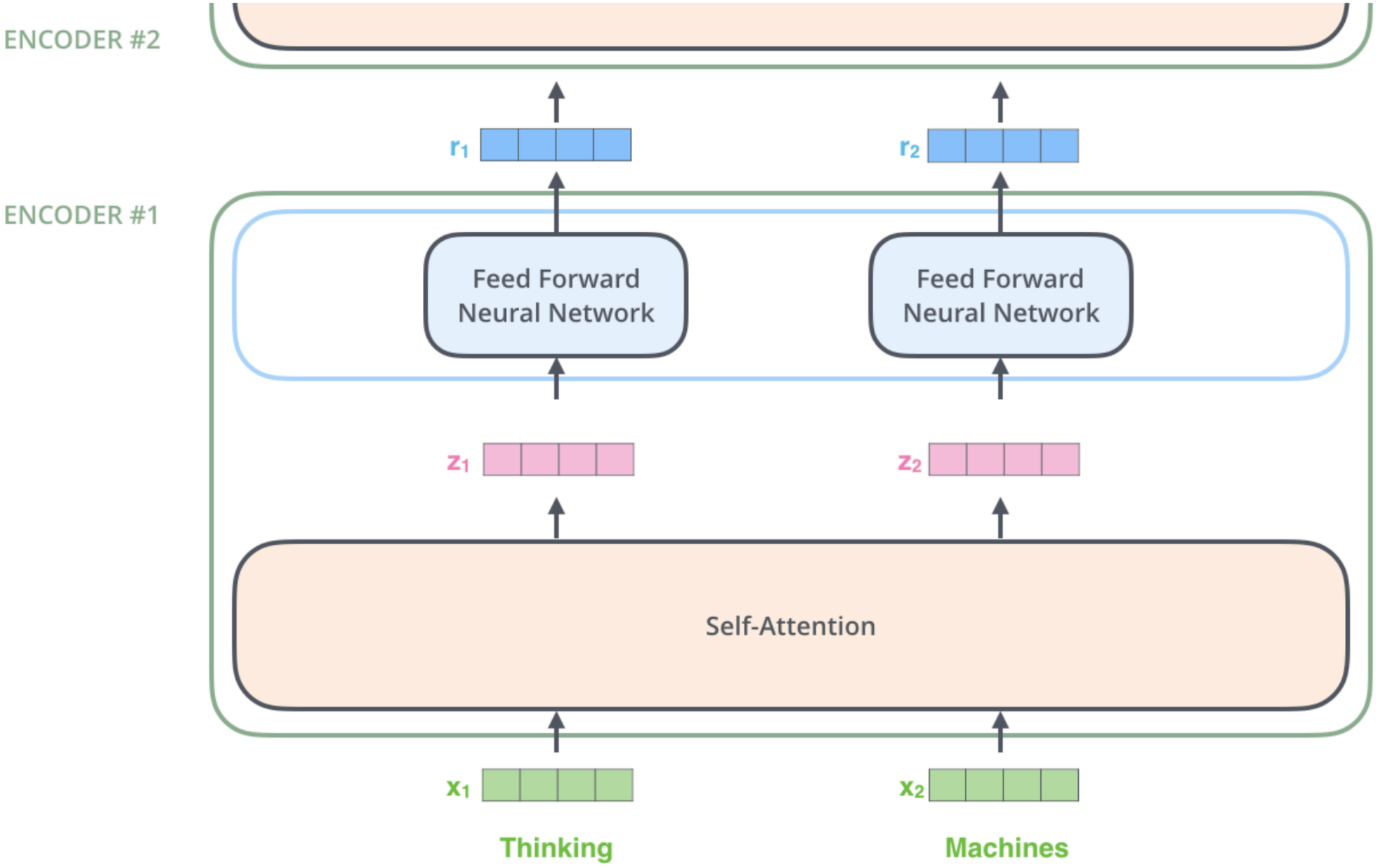
The encoders are all identical in structure (yet they do not share weights). Each one is broken down into two sub-layers:



As is the case in NLP applications in general, we begin by turning each input word into a vector using an embedding algorithm.

Each word is embedded into a vector of size 512. We'll represent those vectors with these simple boxes.

Now We're Encoding!



The word at each position passes through a self-encoding process. Then, they each pass through a feed-forward neural network -- the exact same network with each vector flowing through it separately.

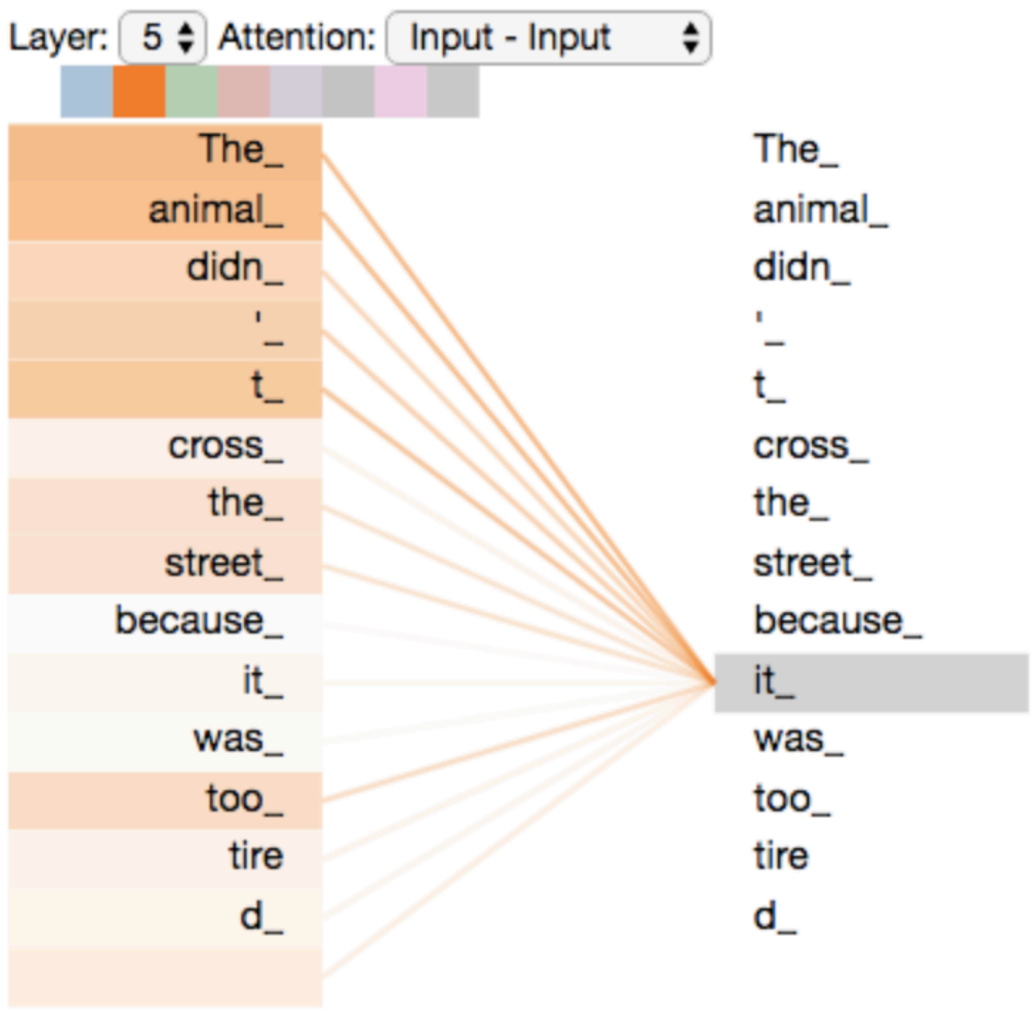
Self-Attention at a High Level

Say the following sentence is an input sentence we want to translate:

"`The animal didn't cross the street because it was too tired`"

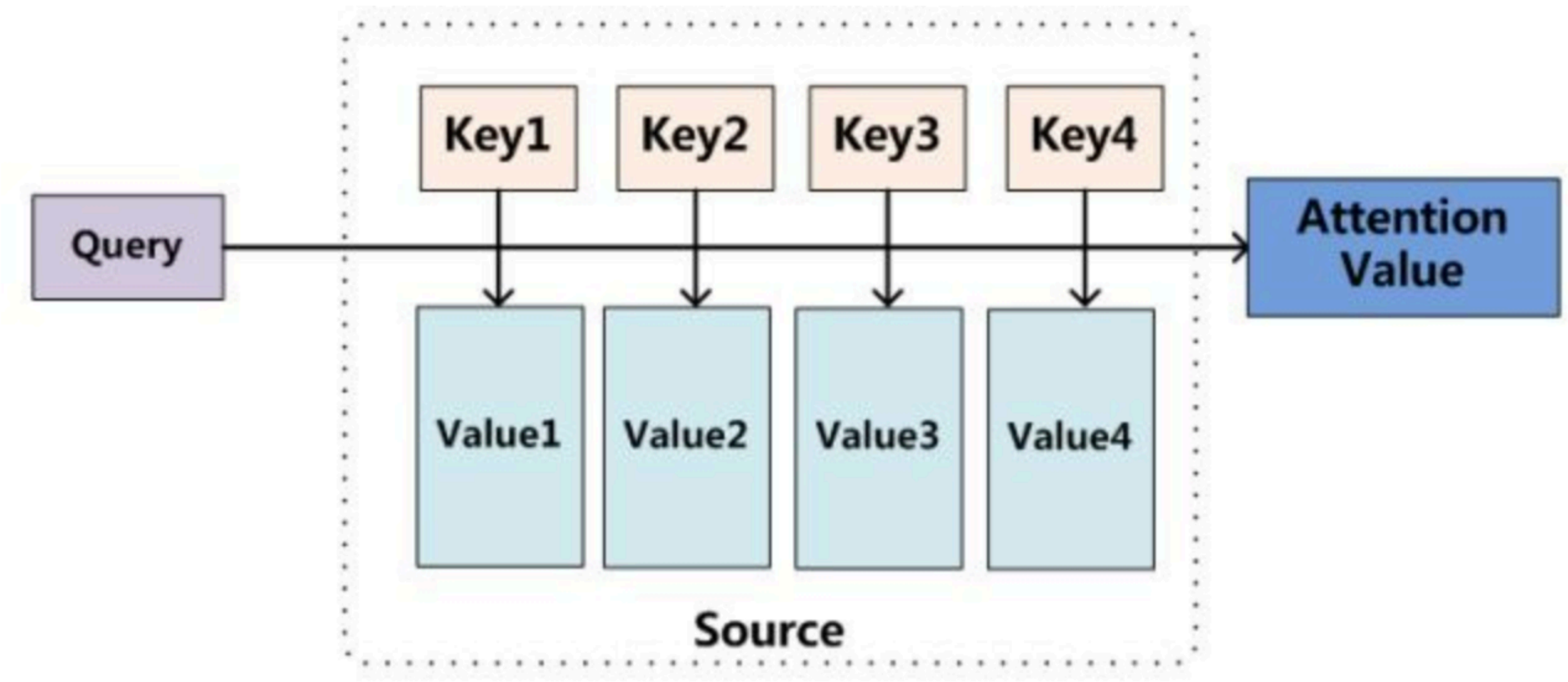
What does "it" in this sentence refer to? Animal or Street???

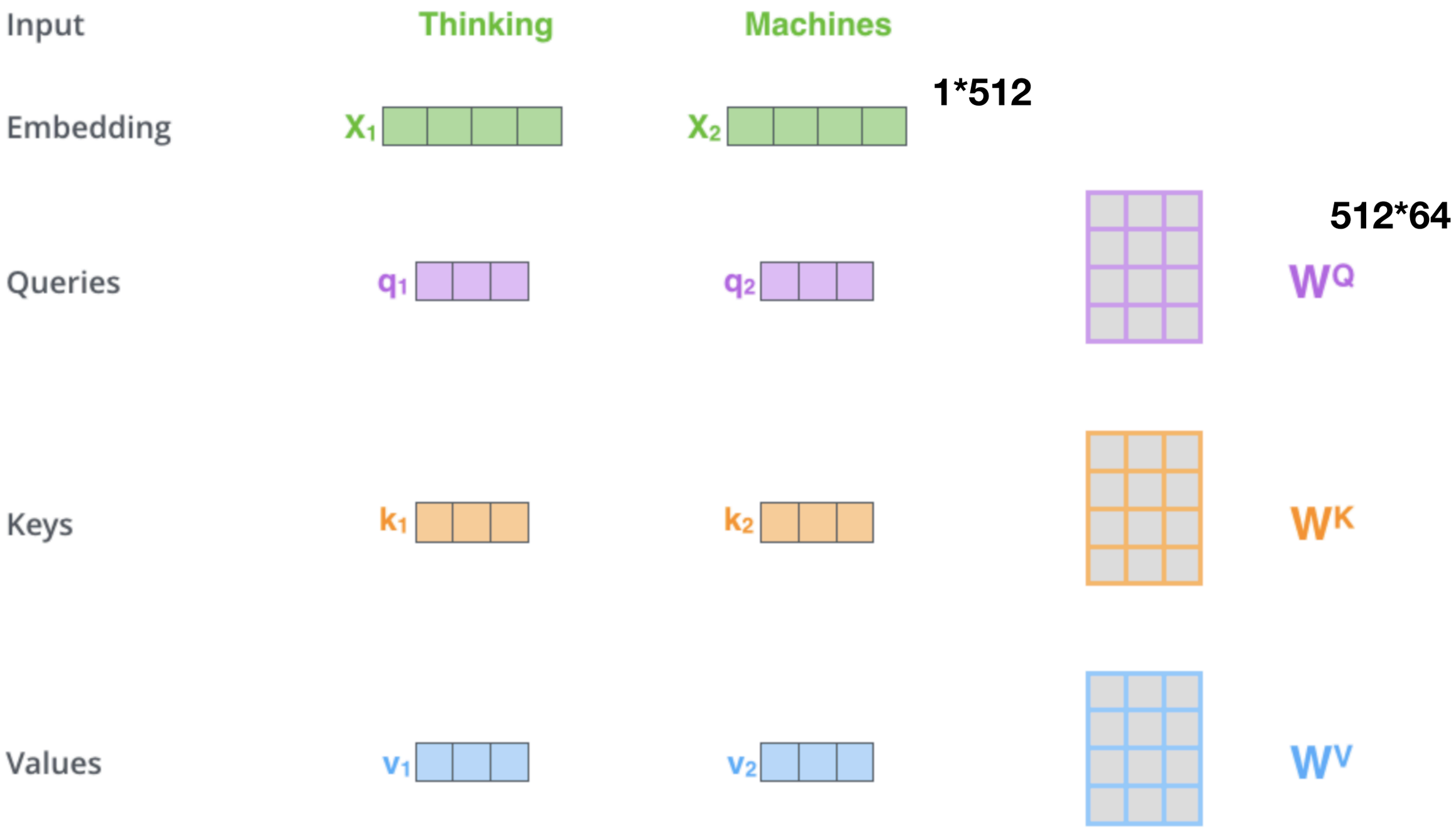
As the model processes each word (each position in the input sequence), self attention allows it to look at other positions in the input sequence for clues that can help lead to a better encoding for this word.



Self-Attention in Detail

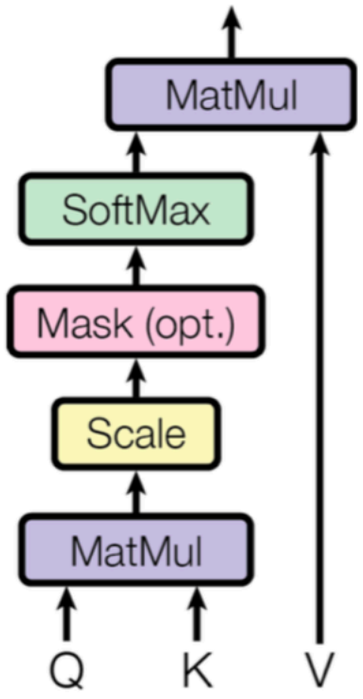
An attention function can be described as mapping a query and a set of key-value pairs to an output, where the query, keys, values, and output are all vectors. The output is computed as a weighted sum of the values, where the weight assigned to each value is computed by a compatibility function of the query with the corresponding key.





Multiplying x1 by the WQ weight matrix produces q1, the "query" vector associated with that word. We end up creating a "query", a "key", and a "value" projection of each word in the input sentence.

Scaled Dot-Product Attention



Input

Embedding

Queries

Keys

Values

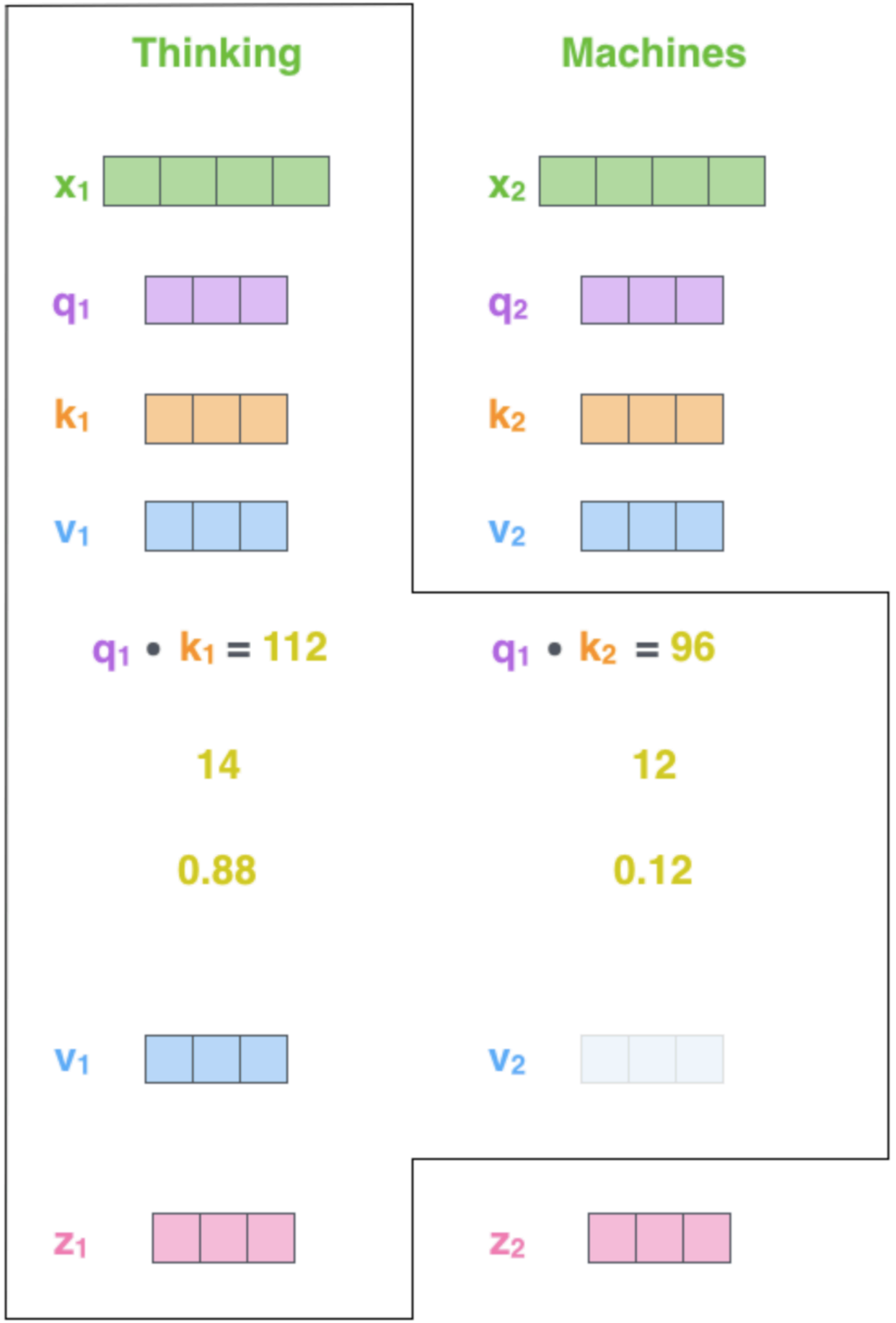
Score

Divide by 8 ($\sqrt{d_k}$)

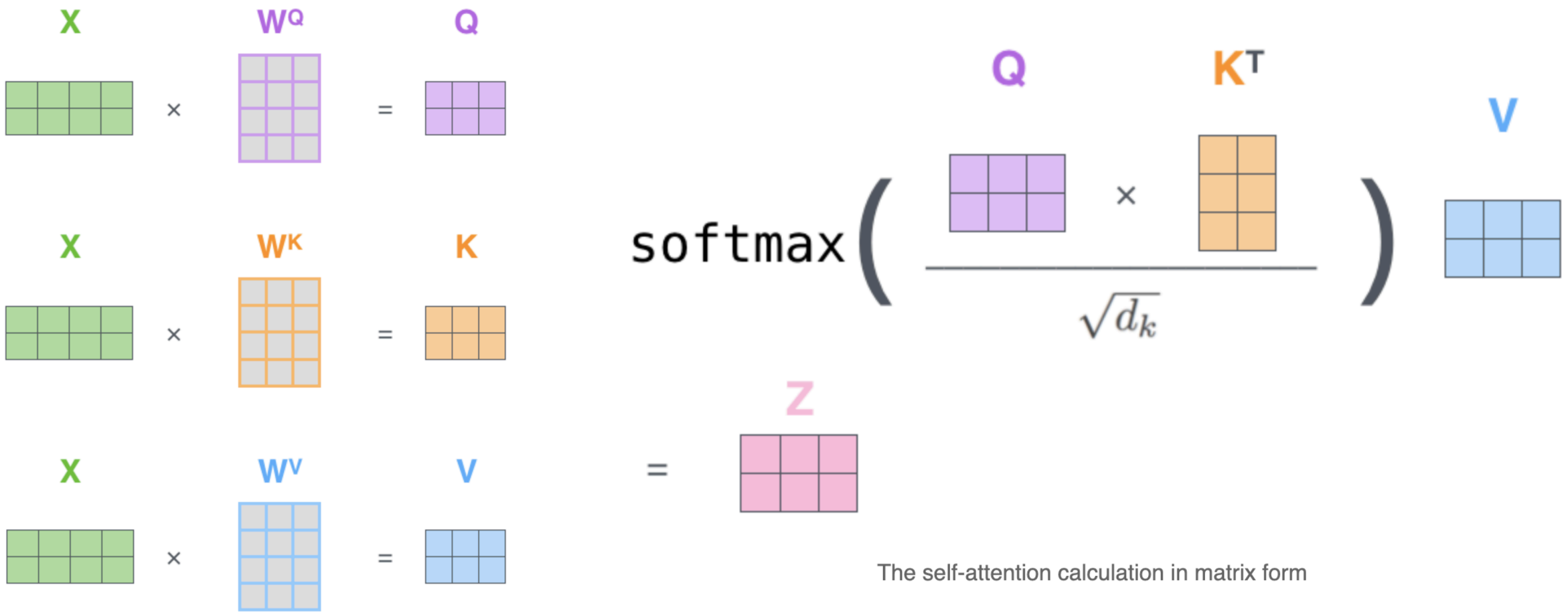
Softmax

Softmax
X
Value

Sum



Matrix Calculation of Self-Attention

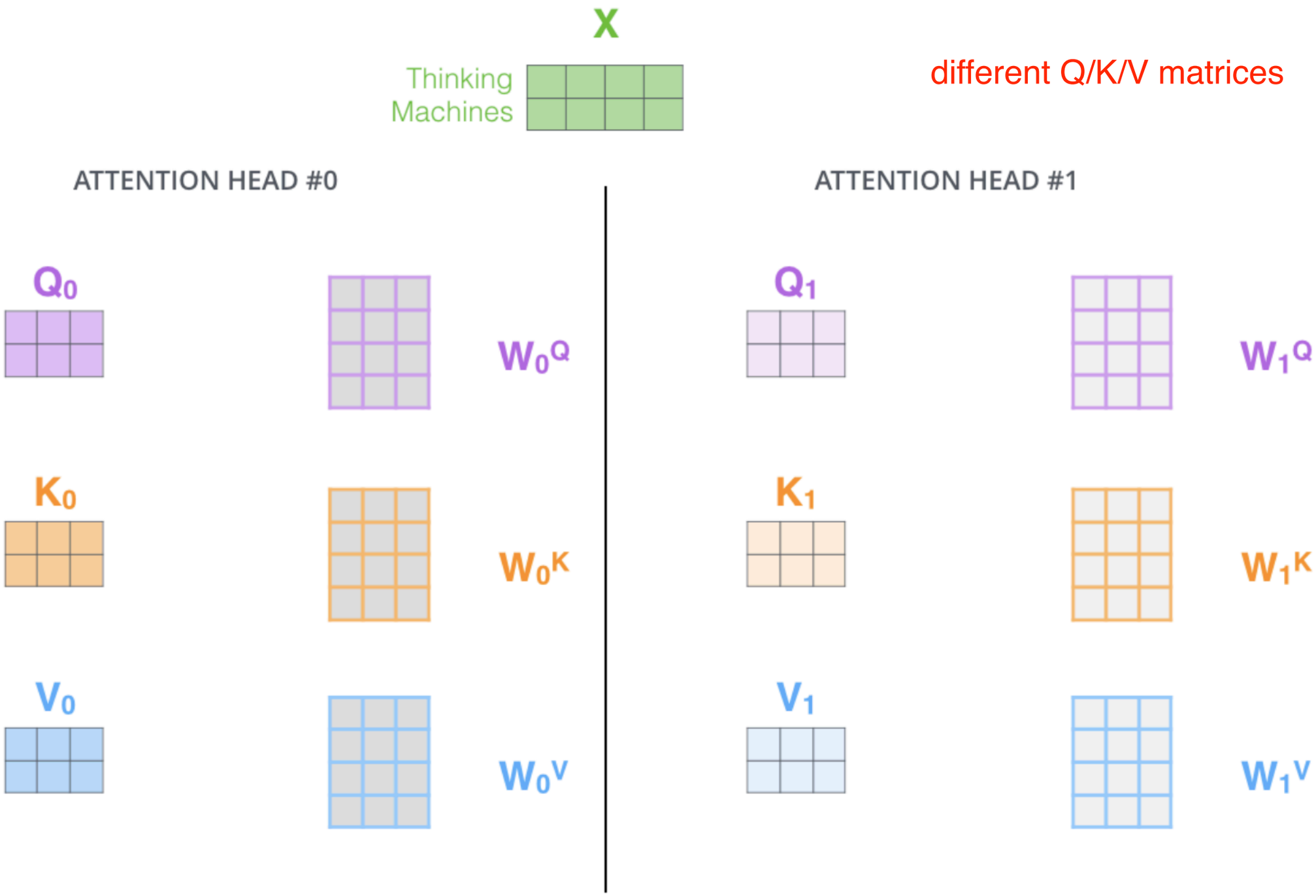


The self-attention calculation in matrix form

In practice, we compute the attention function on a set of queries simultaneously, packed together into a matrix Q. The keys and values are also packed together into matrices K and V. We compute the matrix of outputs as:

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V \tag{1}$$

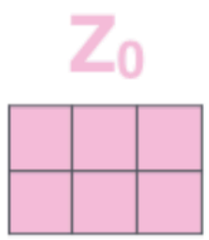
“multi-headed” attention



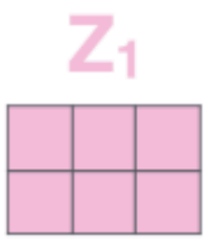
It expands the model’s ability to focus on different positions.

It gives the attention layer multiple “representation subspaces”.

ATTENTION HEAD #0

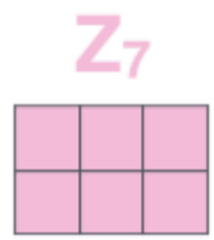


ATTENTION HEAD #1

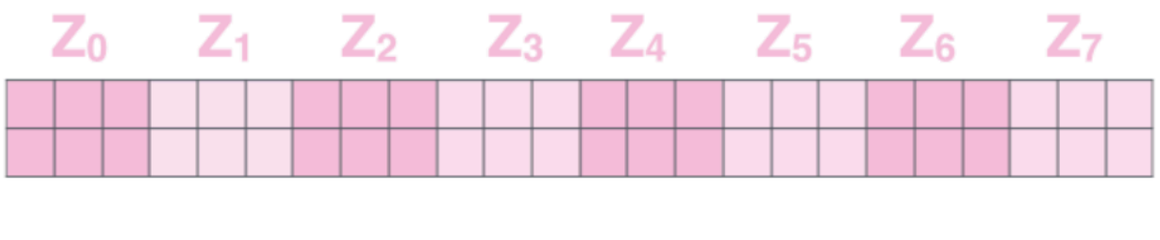


...

ATTENTION HEAD #7

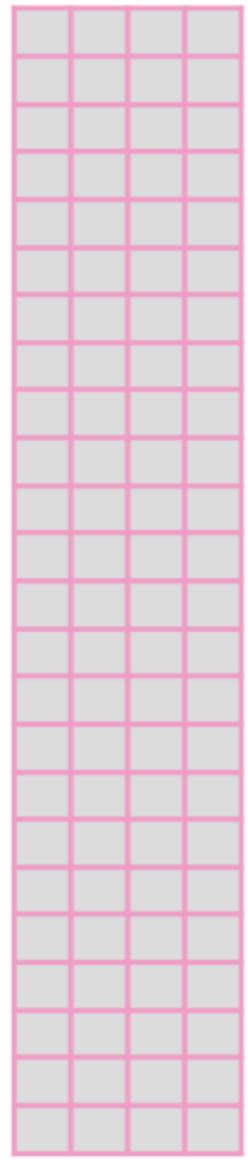


1) Concatenate all the attention heads



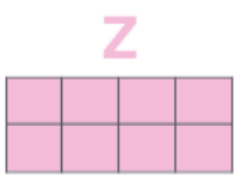
2) Multiply with a weight matrix W^O that was trained jointly with the model

X



3) The result would be the Z matrix that captures information from all the attention heads. We can send this forward to the FFNN

=

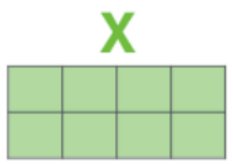


multi-headed self-attention

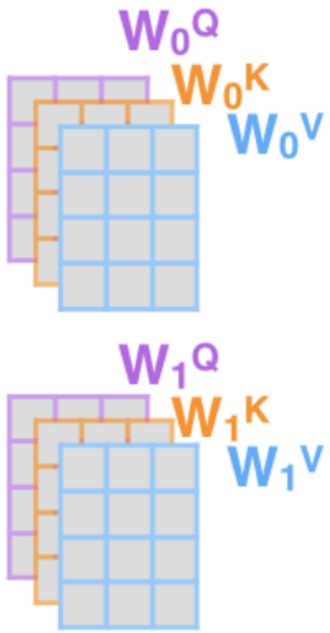
1) This is our input sentence*

Thinking
Machines

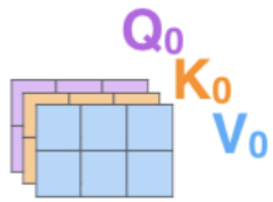
2) We embed each word*



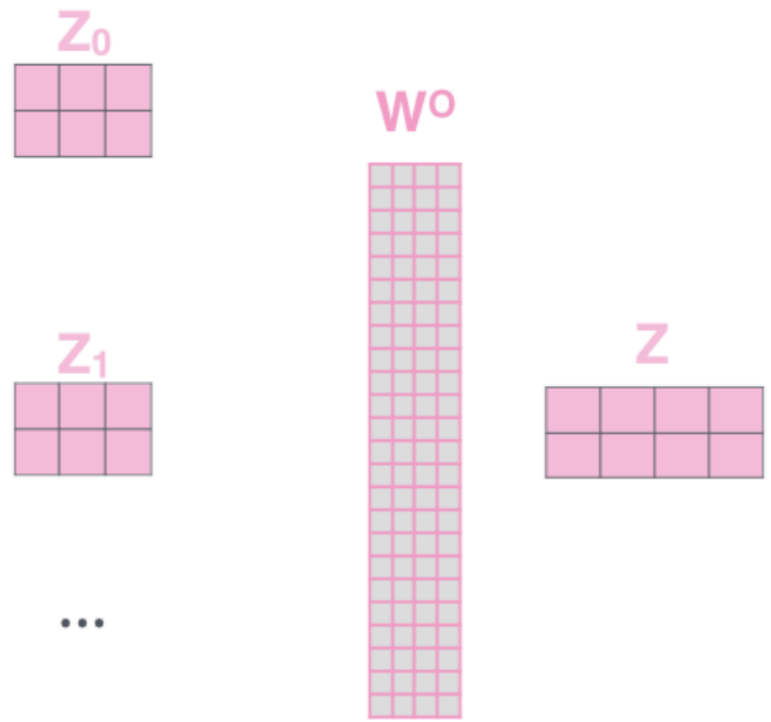
3) Split into 8 heads. We multiply X or R with weight matrices



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

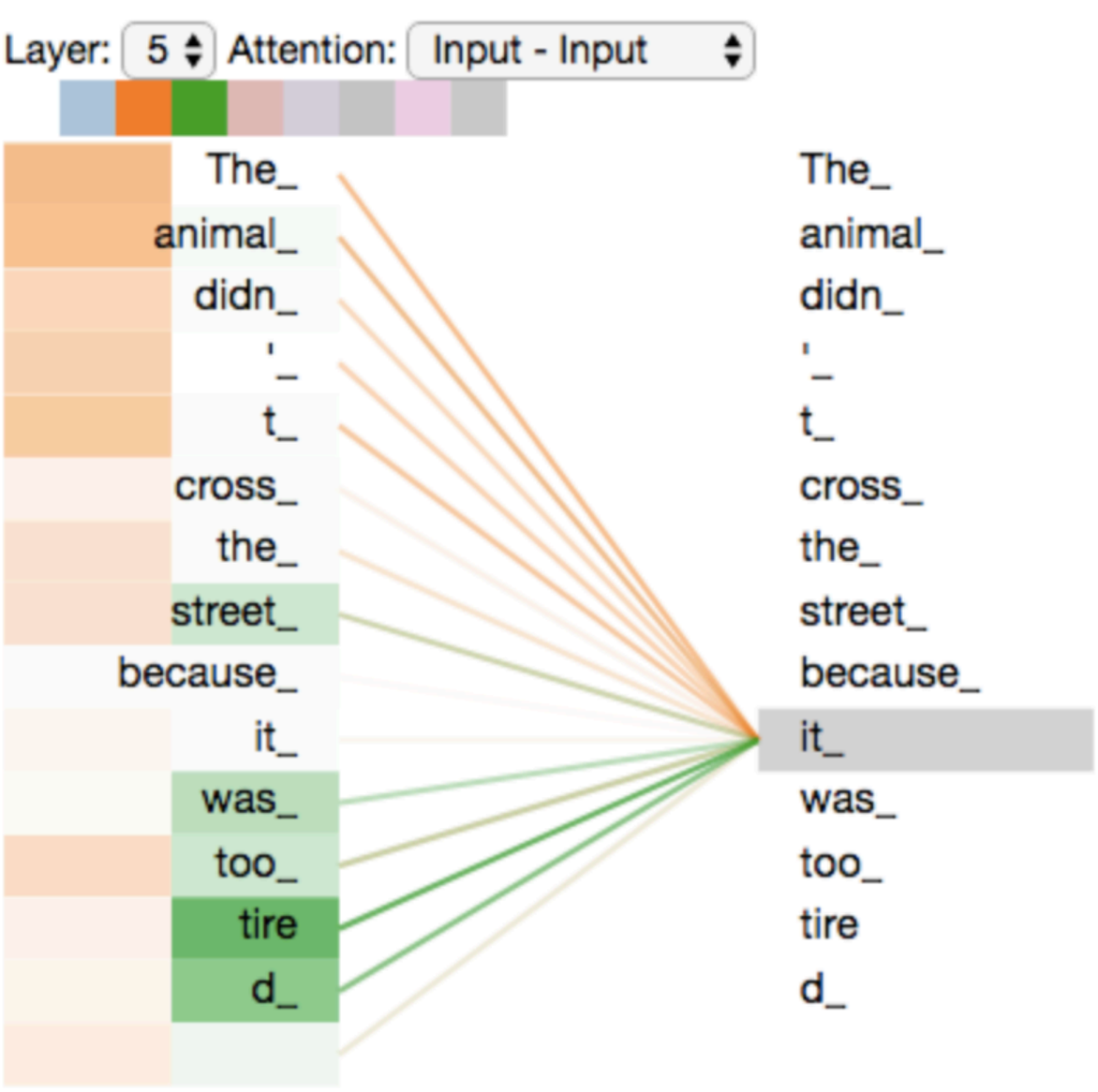


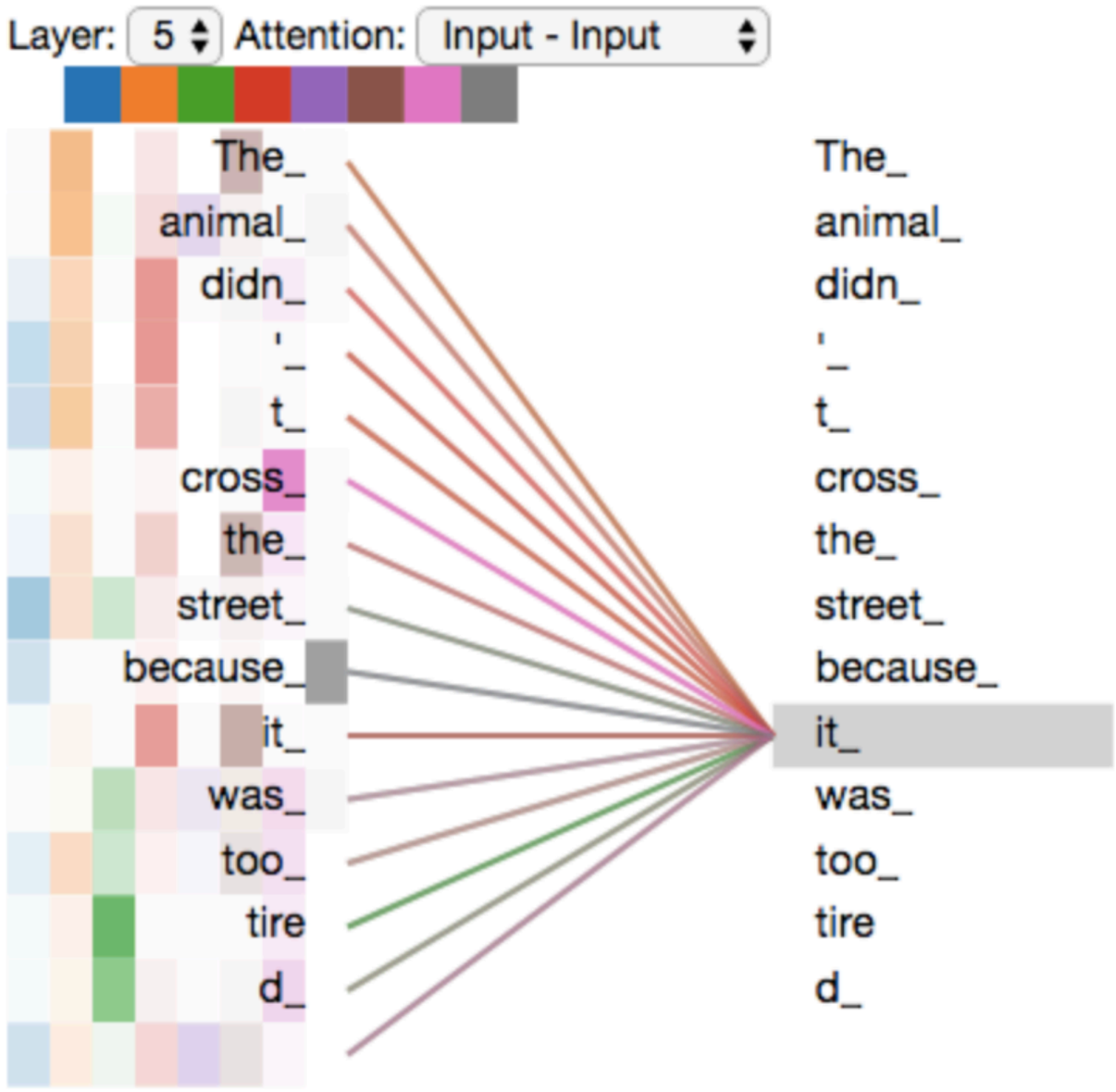
5) Concatenate the resulting Z matrices, then multiply with weight matrix W^O to produce the output of the layer



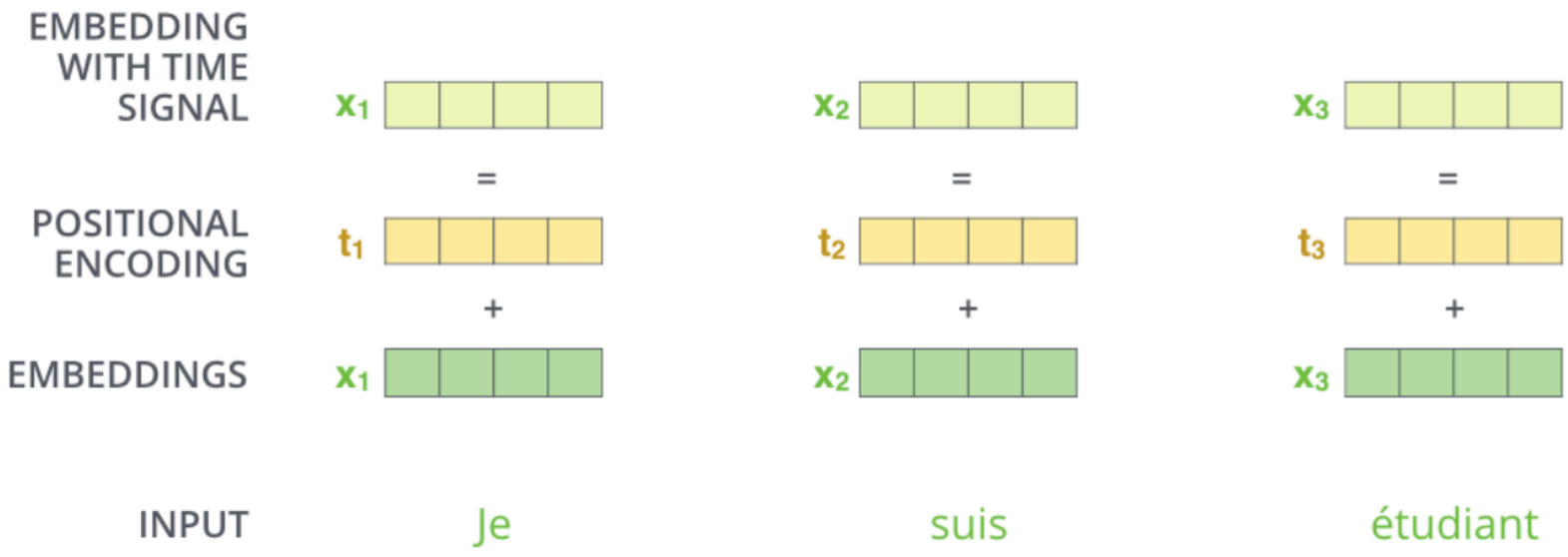
* In all encoders other than #0, we don't need embedding. We start directly with the output of the encoder right below this one

Visualize Attention Score





Representing The Order of The Sequence Using Positional Encoding



To give the model a sense of the order of the words, we add positional encoding vectors -- the values of which follow a specific pattern.

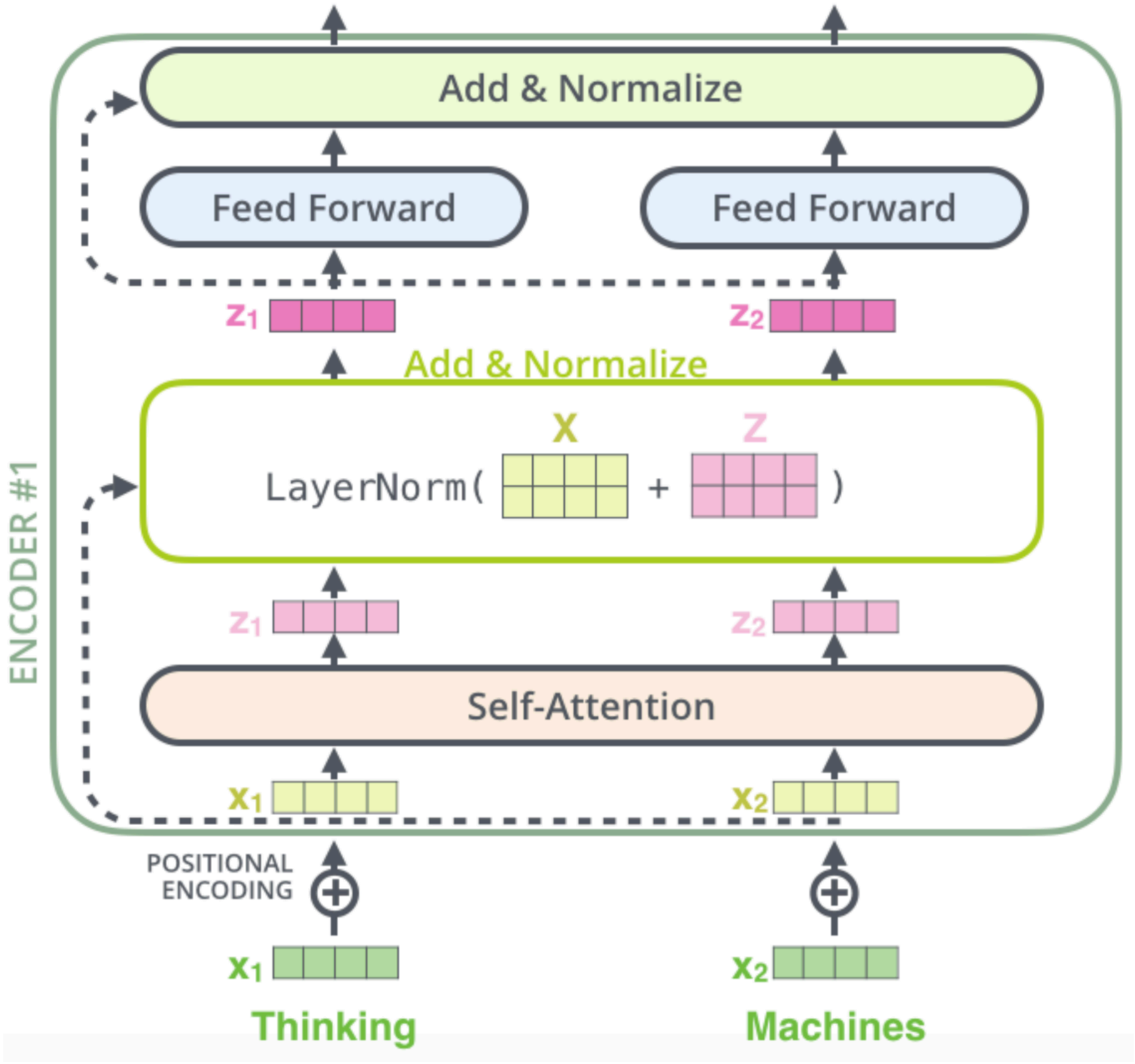
$$PE(pos, 2i) = sin(\frac{pos}{10000^{\frac{2i}{d_{model}}}}) \qquad (3)$$

$$PE(pos, 2i+1) = cos(\frac{pos}{10000^{\frac{2i}{d_{model}}}}) \qquad (4)$$

$$sin(\alpha+\beta) = sin\alpha cos\beta + cos\alpha sin\beta \qquad cos(\alpha+\beta) = cos\alpha cos\beta - sin\alpha sin\beta$$

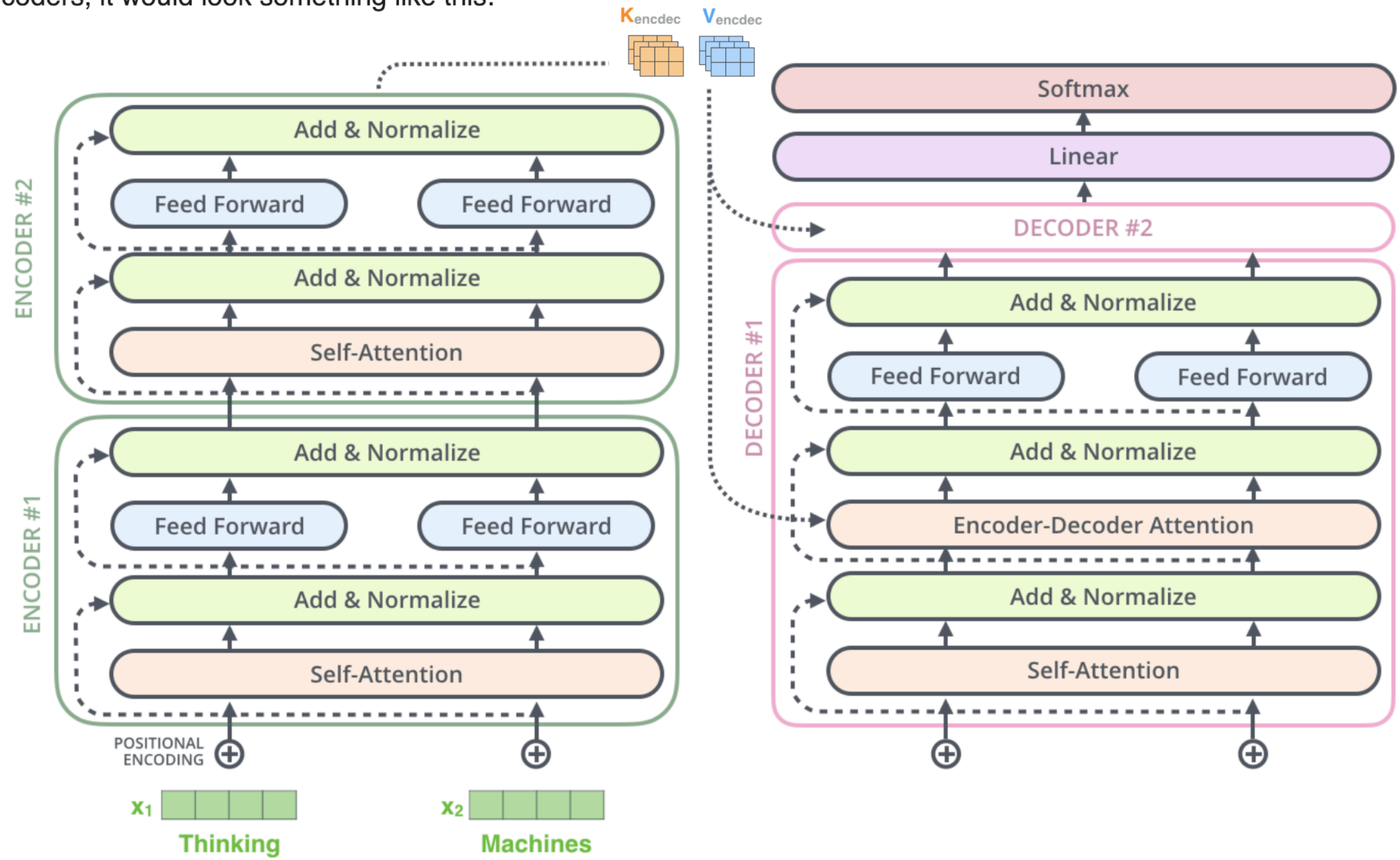
It is shown that the position vector of position k+p can be expressed as the linear change of the feature vector of position K.

The Residuals

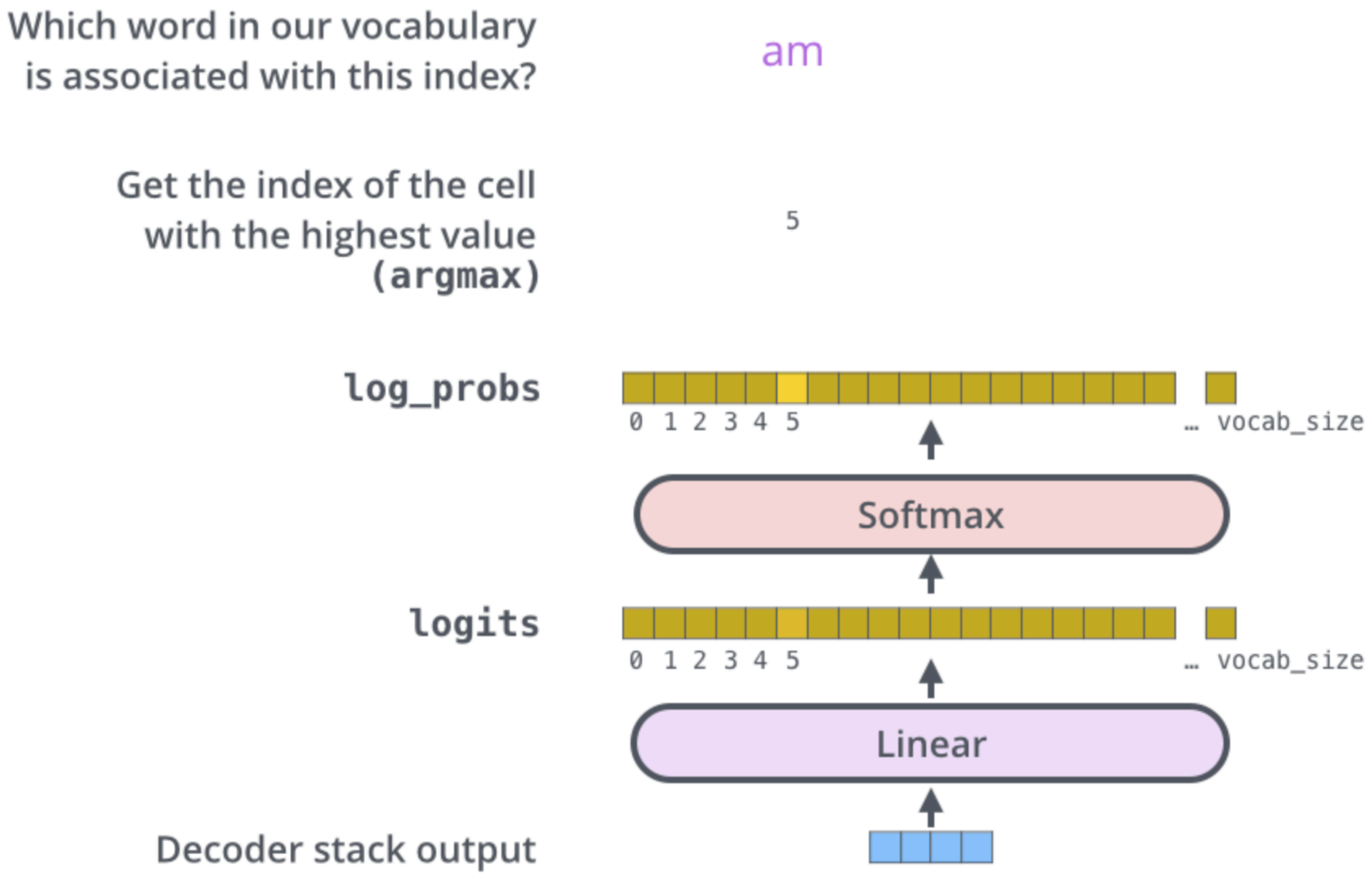


The Decoder Side

This goes for the sub-layers of the decoder as well. If we're to think of a Transformer of 2 stacked encoders and decoders, it would look something like this:



The Final Linear and Softmax Layer

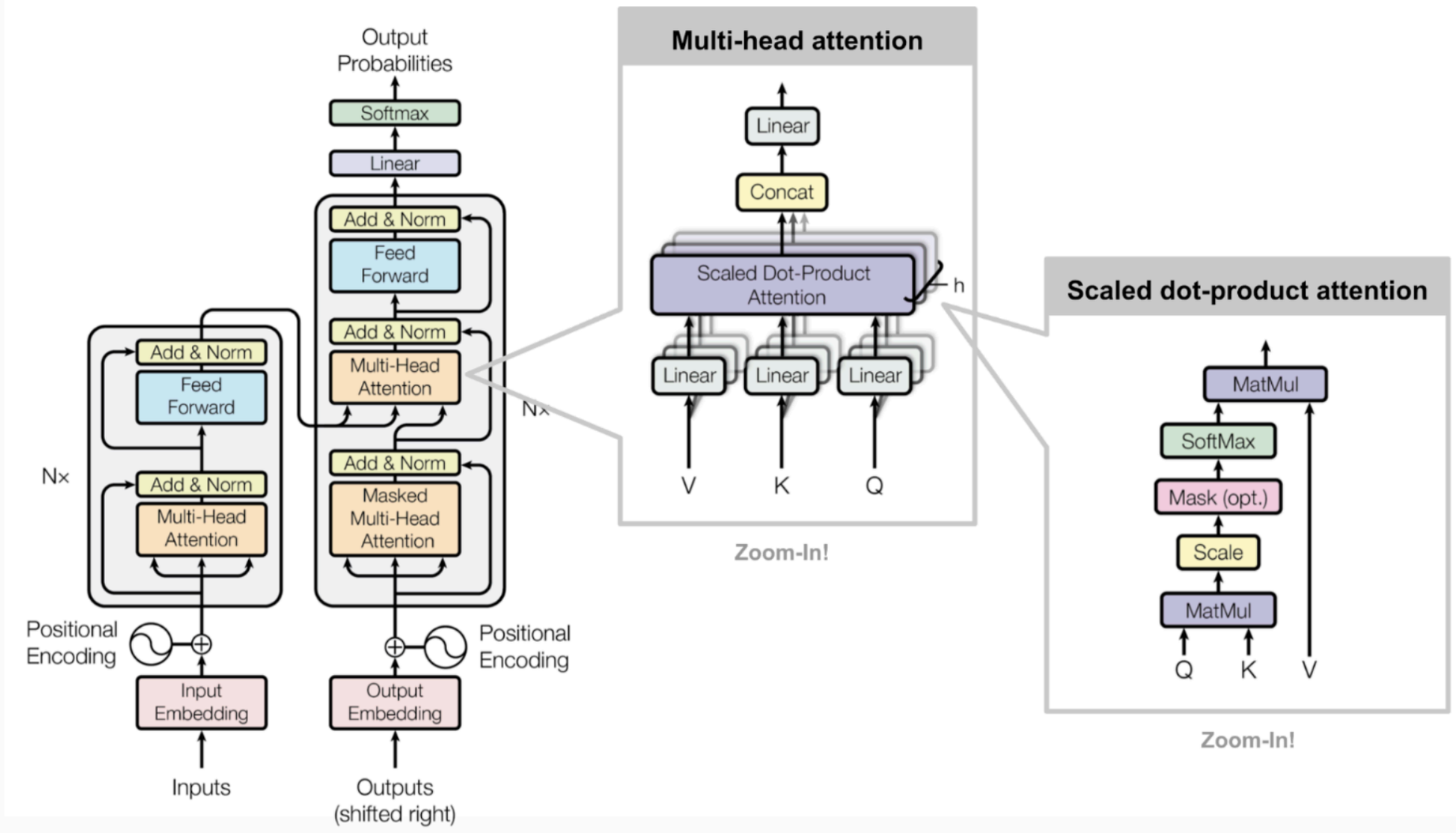


This figure starts from the bottom with the vector produced as the output of the decoder stack. It is then turned into an output word.

Full Architecture

Finally here is the complete view of the transformer's architecture:

- Both the source and target sequences first go through embedding layers to produce data of the same dimension .
- To preserve the position information, a sinusoid-wave-based positional encoding is applied and summed with the embedding output.
- A softmax and linear layer are added to the final decoder output.



Thanks && QA