Lecture 5

- attention history behind
- dot-product attention
- transformer architecture
- positional embeddings
- bert and gpt

Shkhanukova Milana, 2023

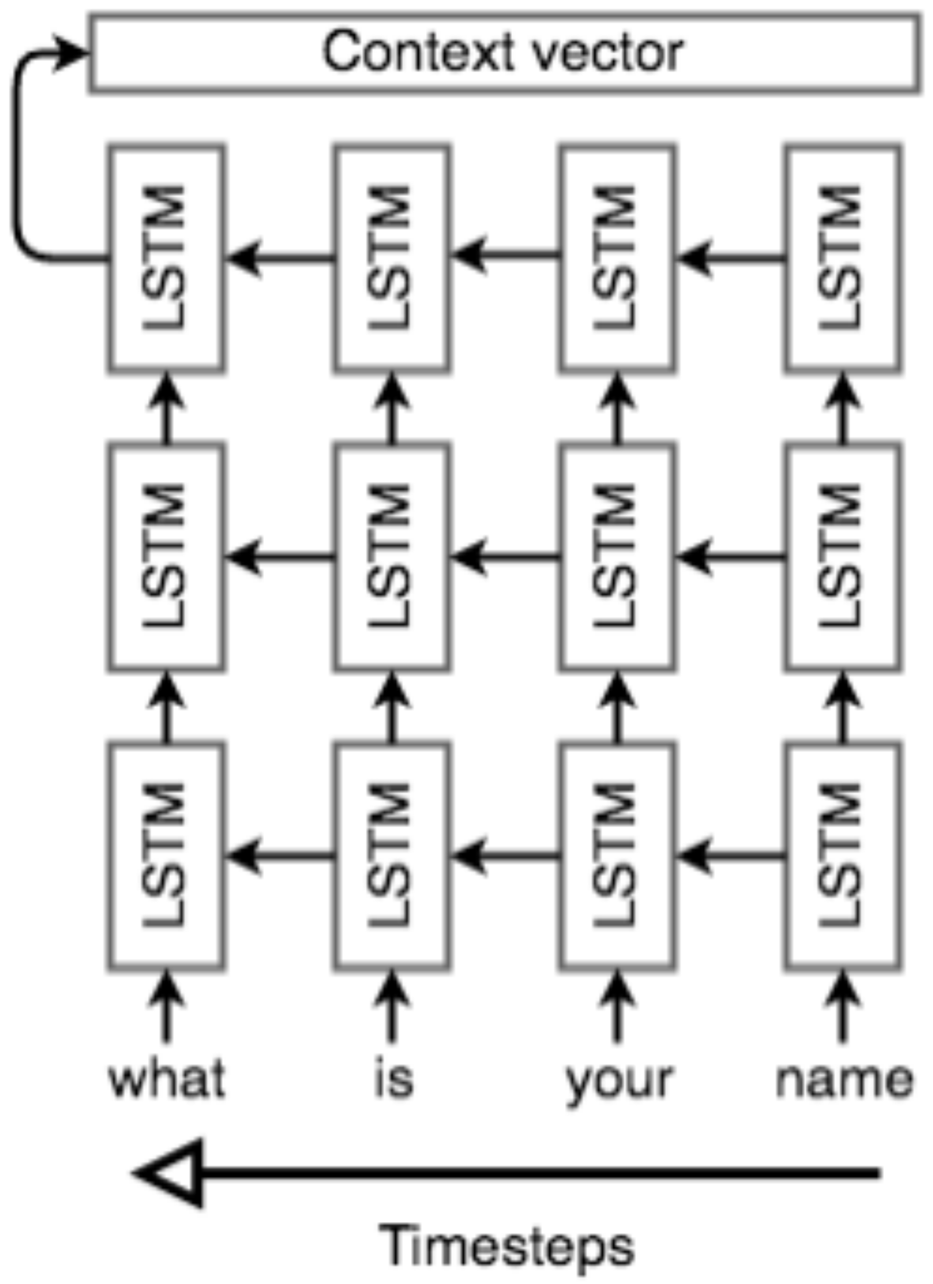
Seq2seq

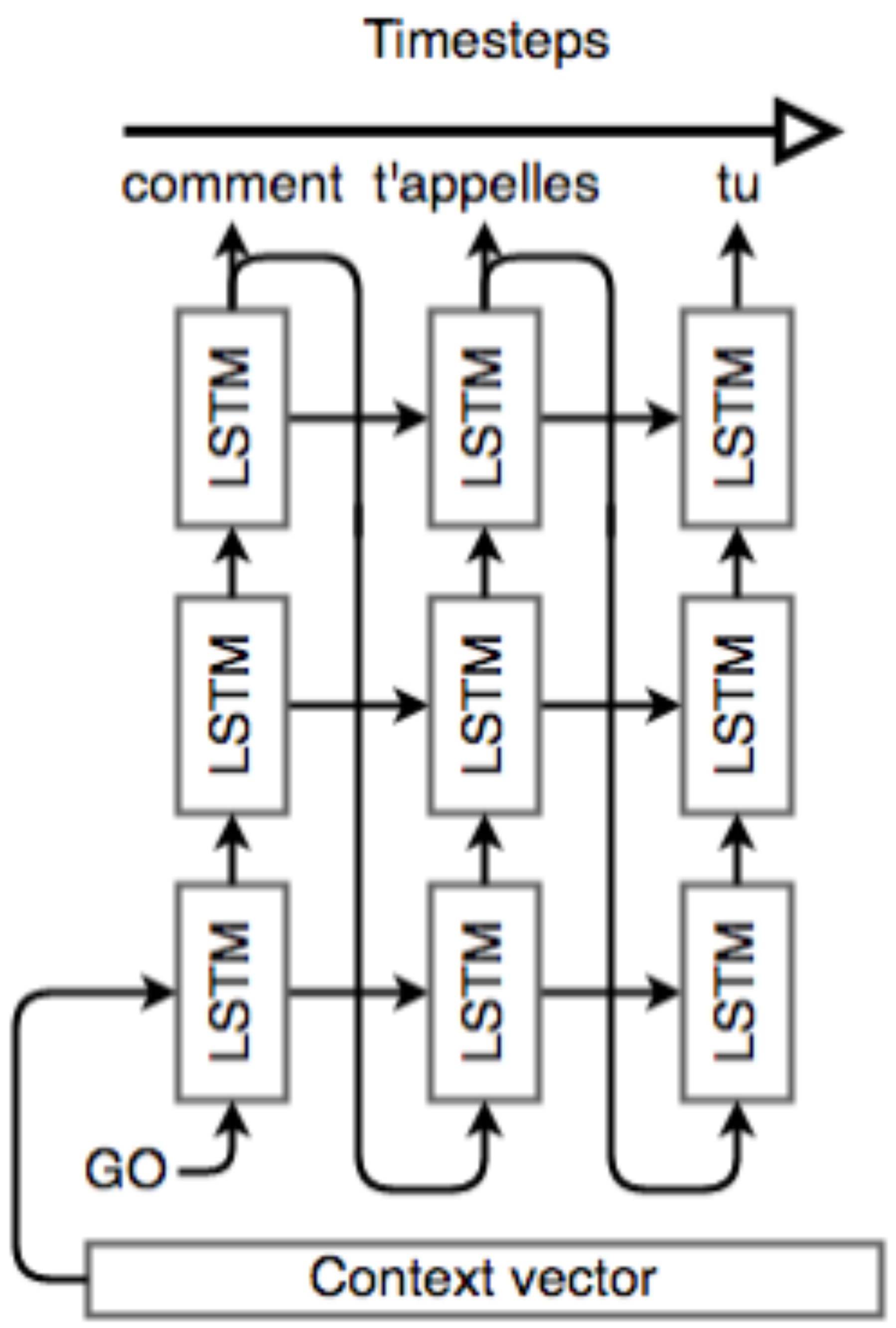
Откуда все пошло?

Sequence-to-sequence, or «Seq2Seq" (2014 for English-French translation)

At a high level, a sequence-to-sequence model is an **end-to-end model** made up of two recurrent neural networks:

- an **encoder**, which takes the model's input sequence as input and encodes it into a fixed-size "context vector", and
- a **decoder**, which uses the context vector from above as a "seed" from which to generate an output sequence





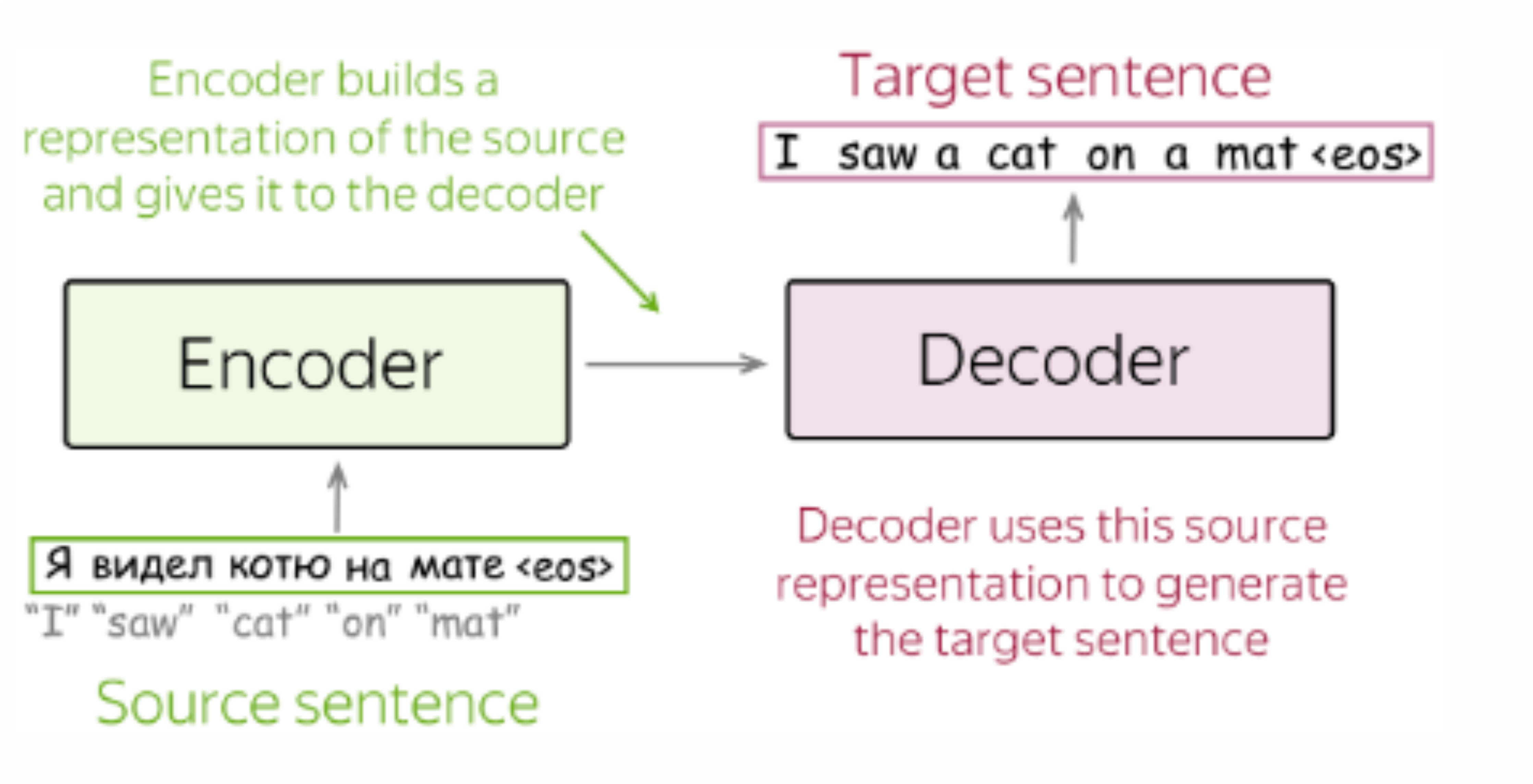
https://web.stanford.edu/class/cs224n/readings/cs224n-2019-notes06-NMT_seq2seq_attention.pdf

Откуда все пошло?

Sequence-to-sequence, or «Seq2Seq" (2014 for English-French translation)

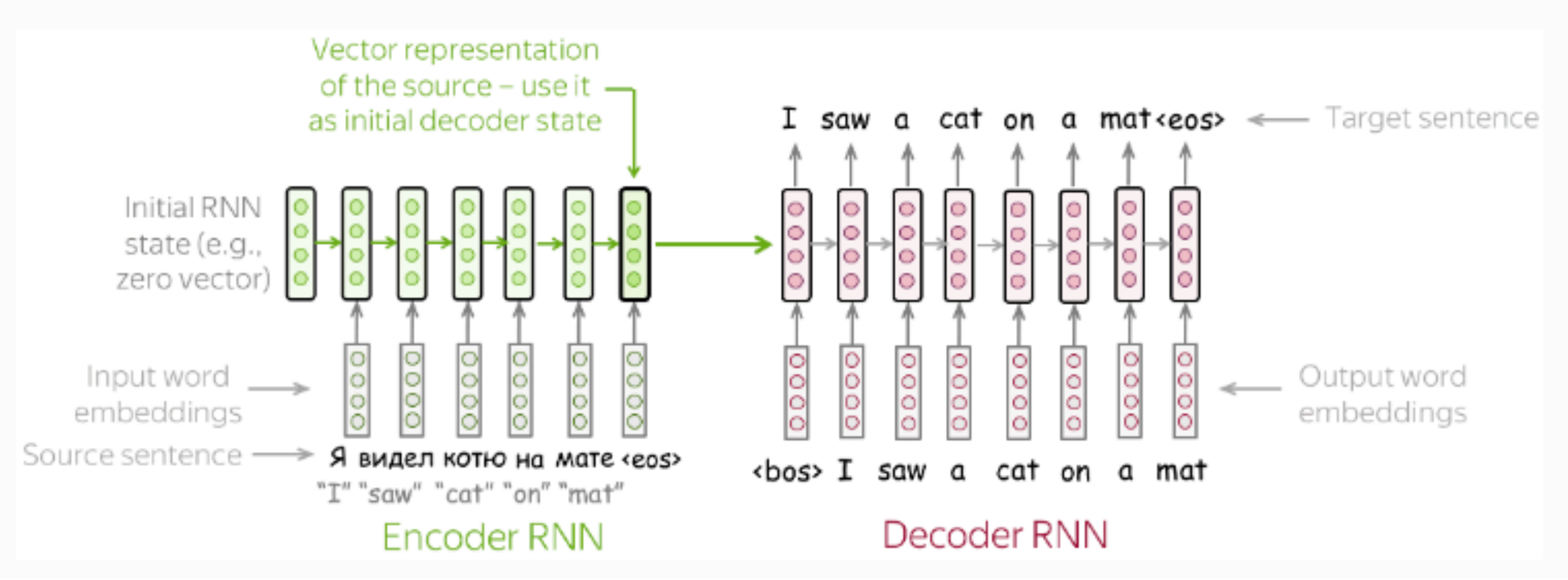
At a high level, a sequence-to-sequence model is an **end-to-end model** made up of two recurrent neural networks:

- an **encoder**, which takes the model's input sequence as input and encodes it into a fixed-size "context vector", and
- a **decoder**, which uses the context vector from above as a "seed" from which to generate an output sequence

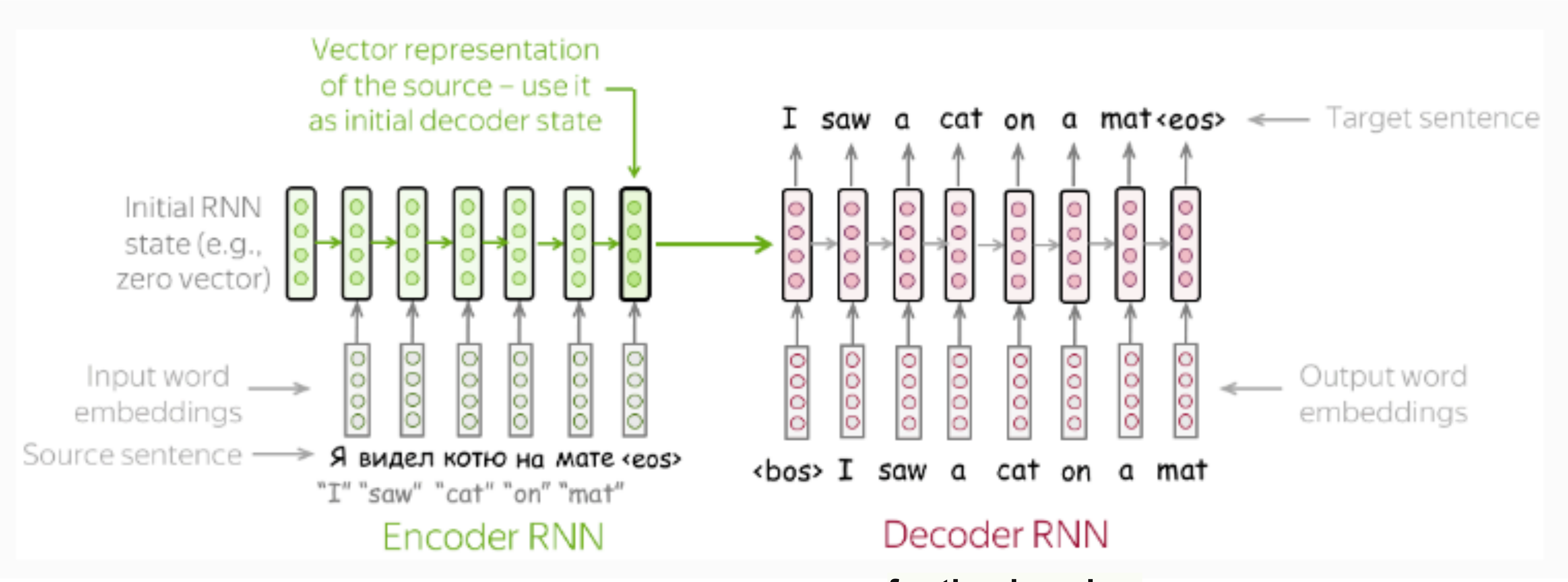


https://lena-voita.github.io/nlp_course/seq2seq_and_attention.html

Какие тут могут быть две проблемы?



Какие тут могут быть две проблемы?

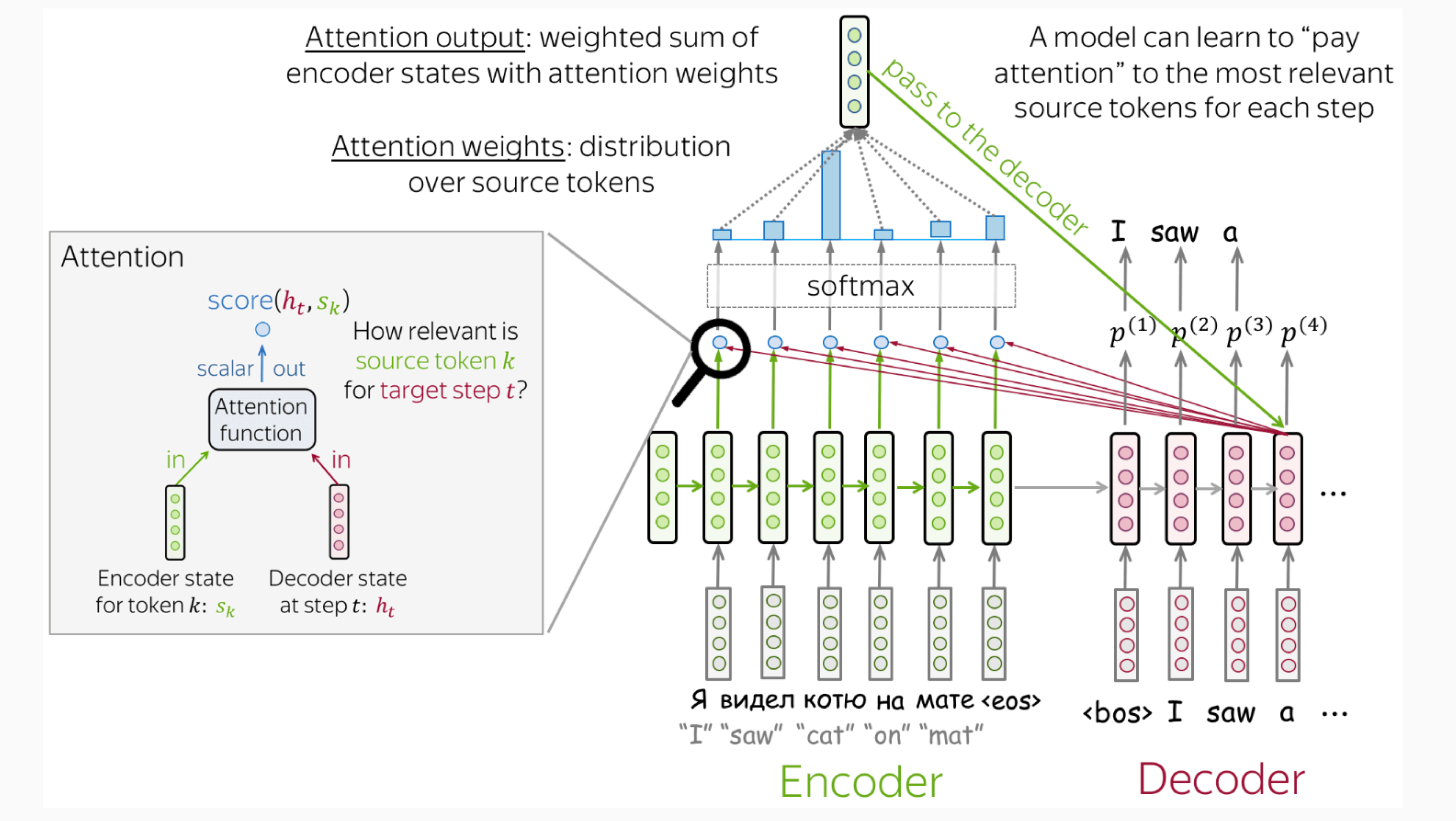


for the encoder, it is hard to compress the sentence

for the decoder, different information may be relevant at different steps.

Simple idea behind

At different steps, let a model "focus" on different parts of the input.



Simple idea behind

At different steps, let a model "focus" on different parts of the input.

Attention output

$$c^{(t)} = a_1^{(t)} s_1 + a_2^{(t)} s_2 + \dots + a_m^{(t)} s_m = \sum_{k=1}^{m} a_k^{(t)} s_k$$

"source context for decoder step t"

↑ (weighted sum)

Attention weights

$$a_k^{(t)} = \frac{\exp(\mathrm{score}(h_t, s_k))}{\sum_{i=1}^{m} \exp(\mathrm{score}(h_t, s_i))}, \mathrm{k} = 1..\mathrm{m}$$

"attention weight for source token k at decoder step t"

↑ (softmax)

Attention scores

$$\mathrm{score}(h_t, s_k), \mathrm{k} = 1..\mathrm{m}$$

"How relevant is source token k for target step t?"

↑

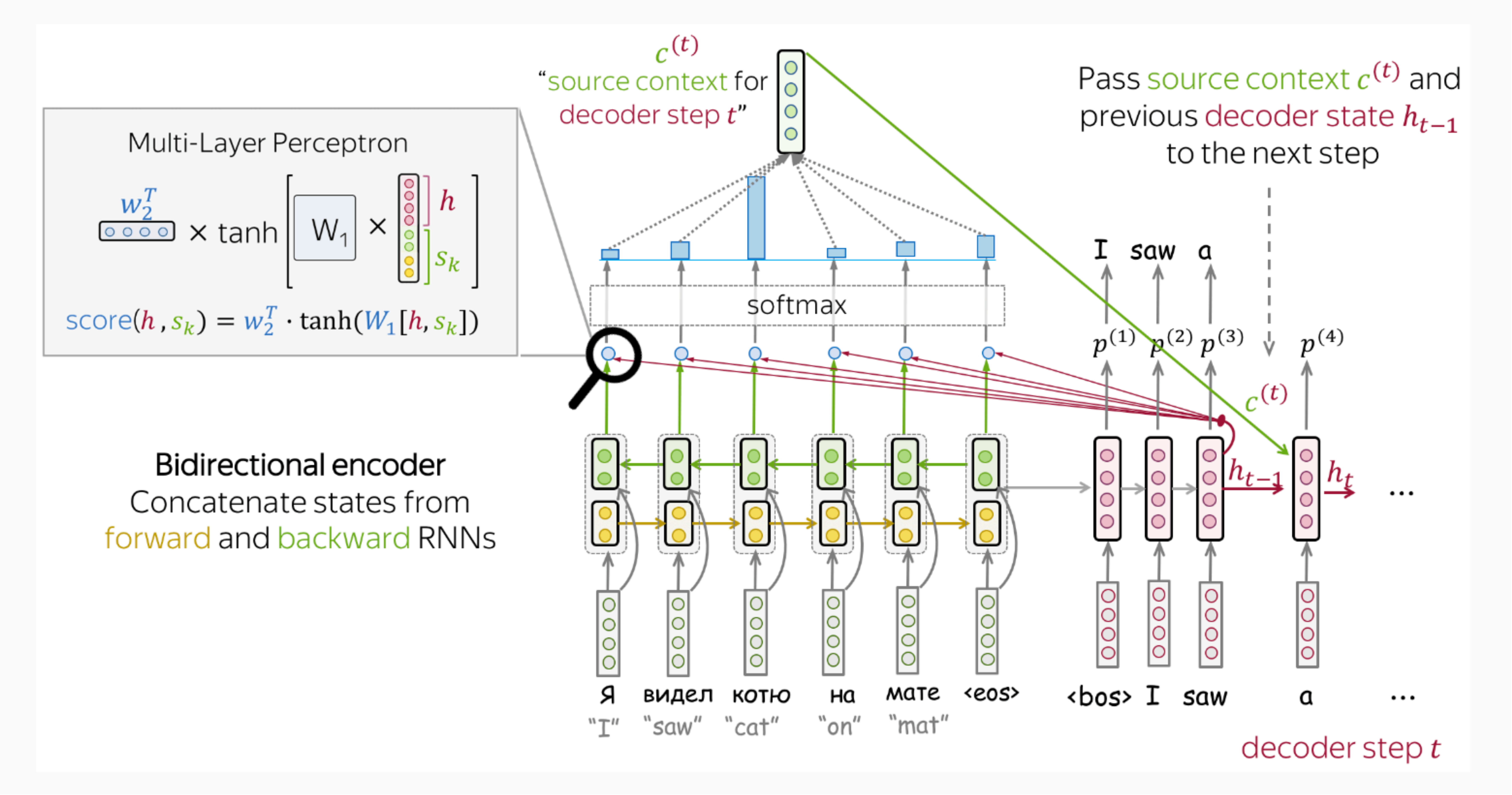
Attention input

$s_1, s_2, \dots, s_m$ — all encoder states

h_t — one decoder state

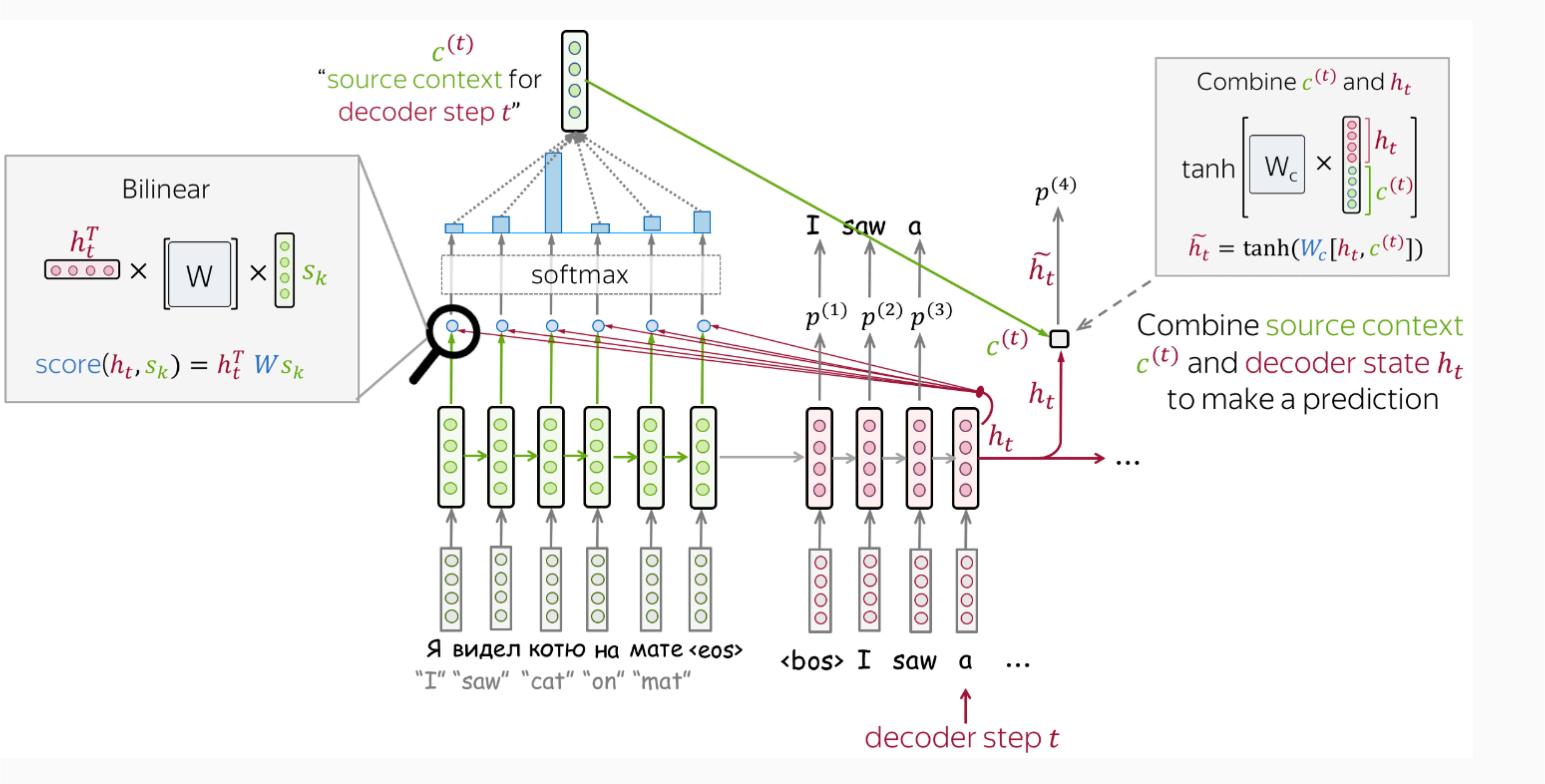
Attention before 2017

Bahdanau



https://arxiv.org/pdf/1409.0473.pdf

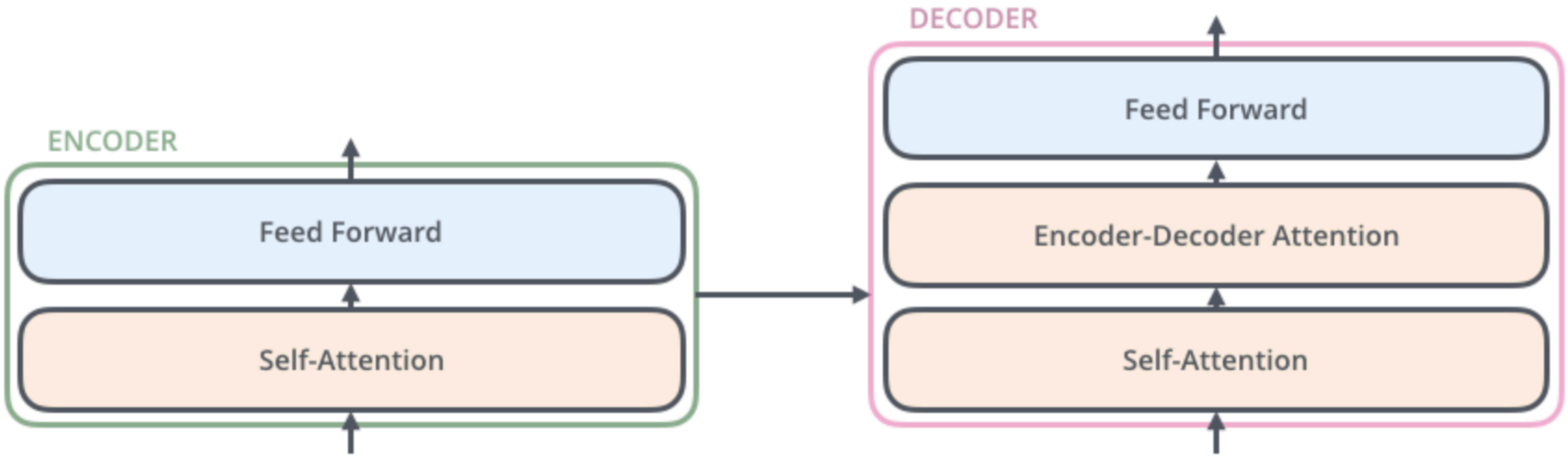
Luong

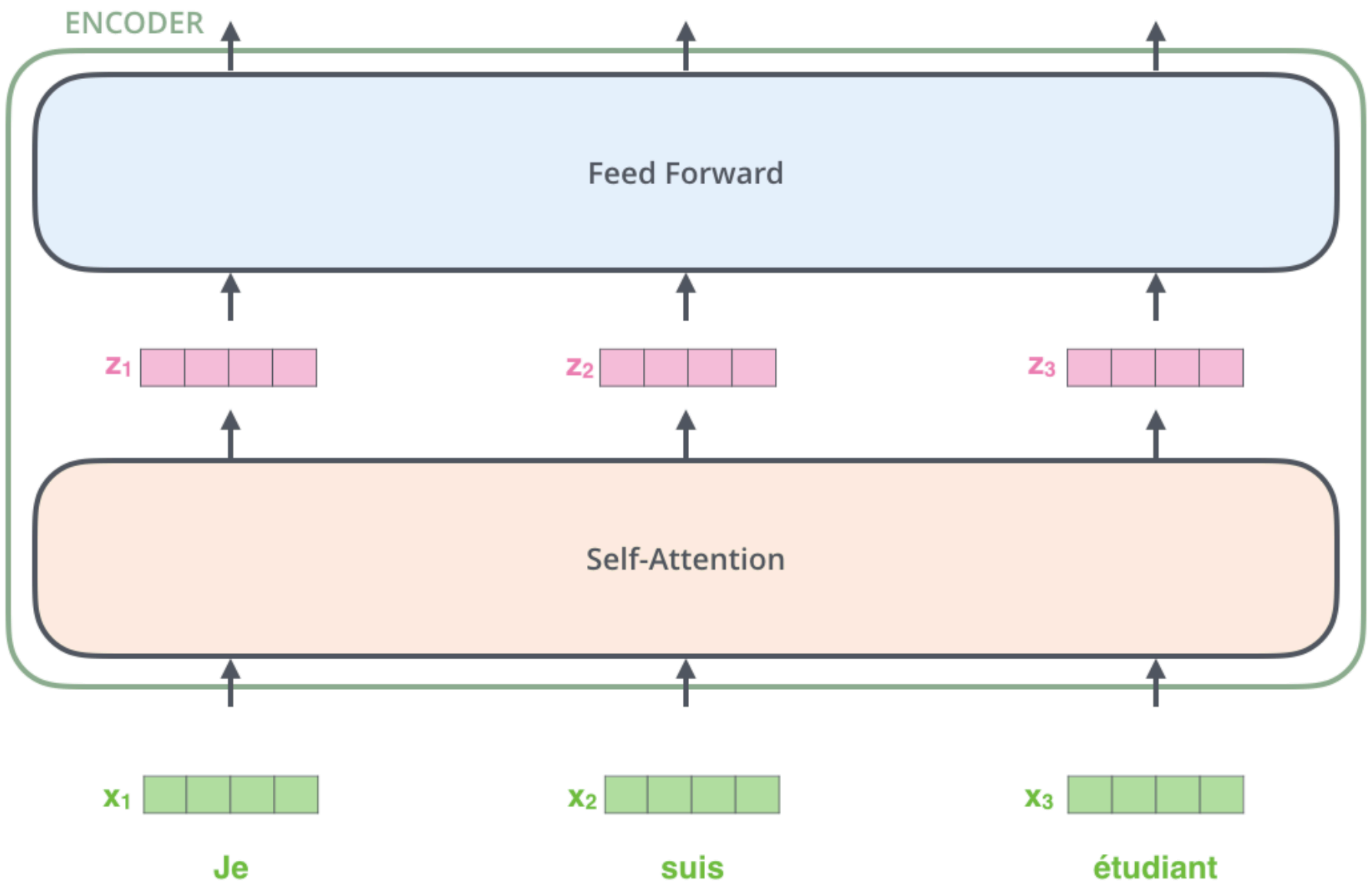


https://slds-lmu.github.io/seminar_nlp_ss20/attention-and-self-attention-for-nlp.html

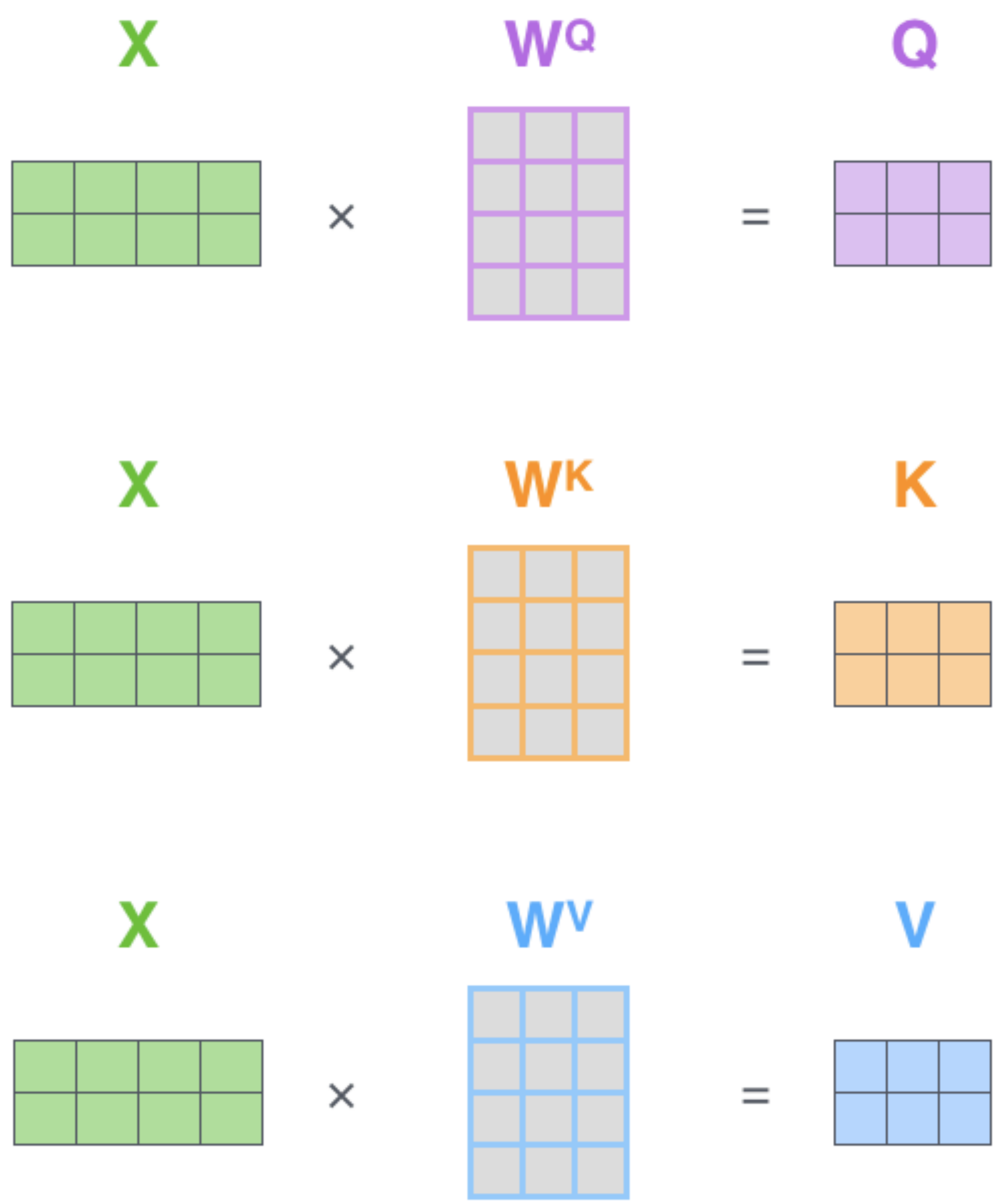
Attention
In transformer
step-by-step

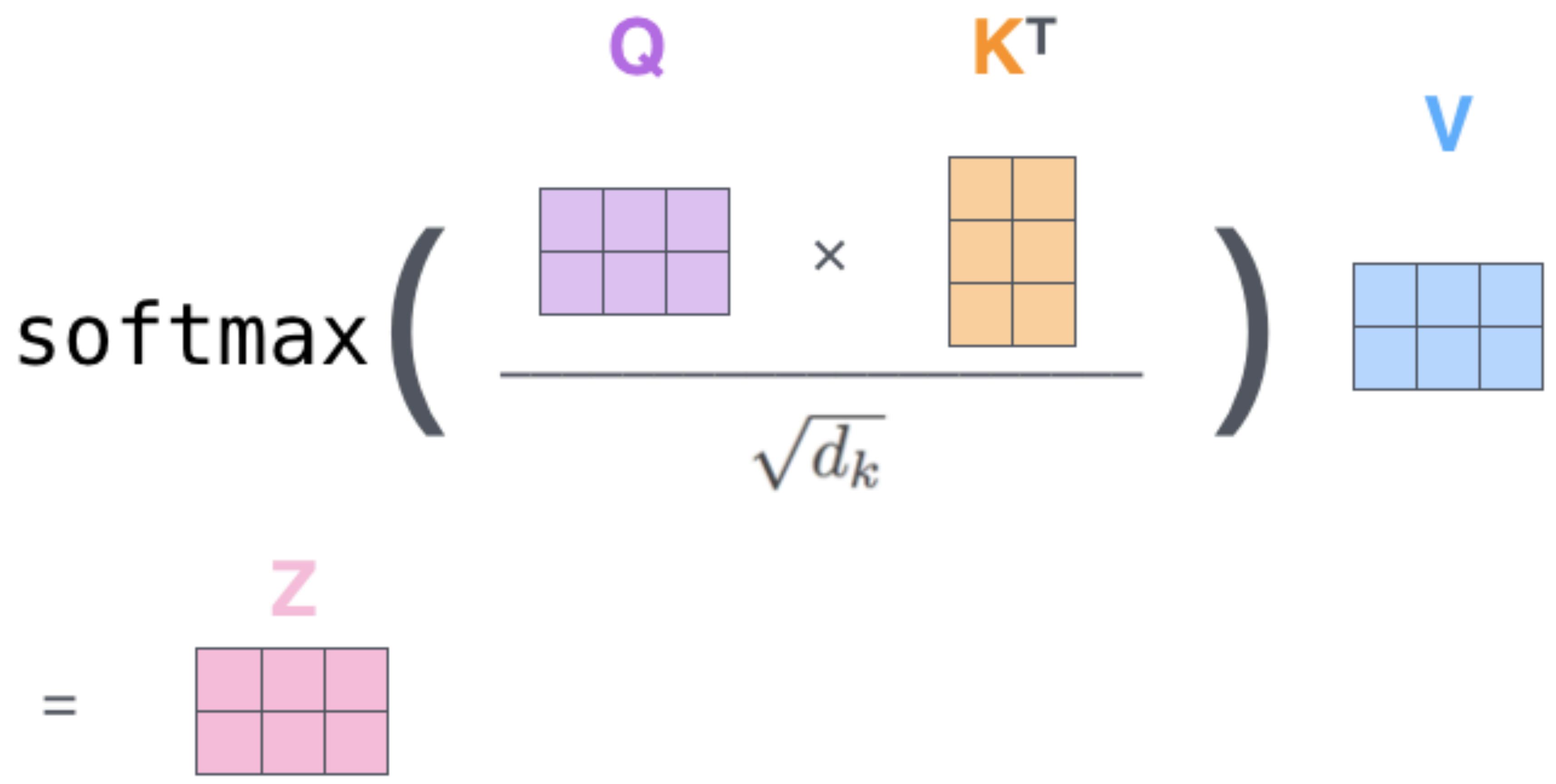
https://jalammar.github.io/illustrated-transformer/





https://slds-lmu.github.io/seminar_nlp_ss20/attention-and-self-attention-for-nlp.html





1. Создаем три вектора - queries, key, values.
2. Взвешиваем каждое слово относительно всех остальных. (Dot-product, query and key)
3. Softmax - теперь все значения будут positive и суммироваться в 1
4. Умножаем value на softmax score

The main idea behind self-attention is that instead of using a fixed embedding for each token, we can use the whole sequence to compute a *weighted average* of each embedding.

Given a sequence of token embeddings $x_1, ..., x_n$, self-attention produces a sequence of **new embeddings $x'_1, ..., x'_n$**

where each **x'_i is a linear combination of all the x_j** :

The coefficients $w_{j\,i}$ are called *attention weights* and are normalized so that $\sum_j w_{j\,i} = 1$.

$$x_i' = \sum_{j=1}^{N} w_{ji} x_j \qquad\qquad \sum w_{ji} = 1.$$

Query, keys, values

you compare the query with the keys to get scores/weights for the values
(Сравниваем query с keys, чтобы получить веса для обновления values)

- key - относительно кого считаем, исходный текст на языке оригинала
- query - для кого считаем, переведенный текст на таргет языке
- value - на чем мы считаем, снова исходный текст

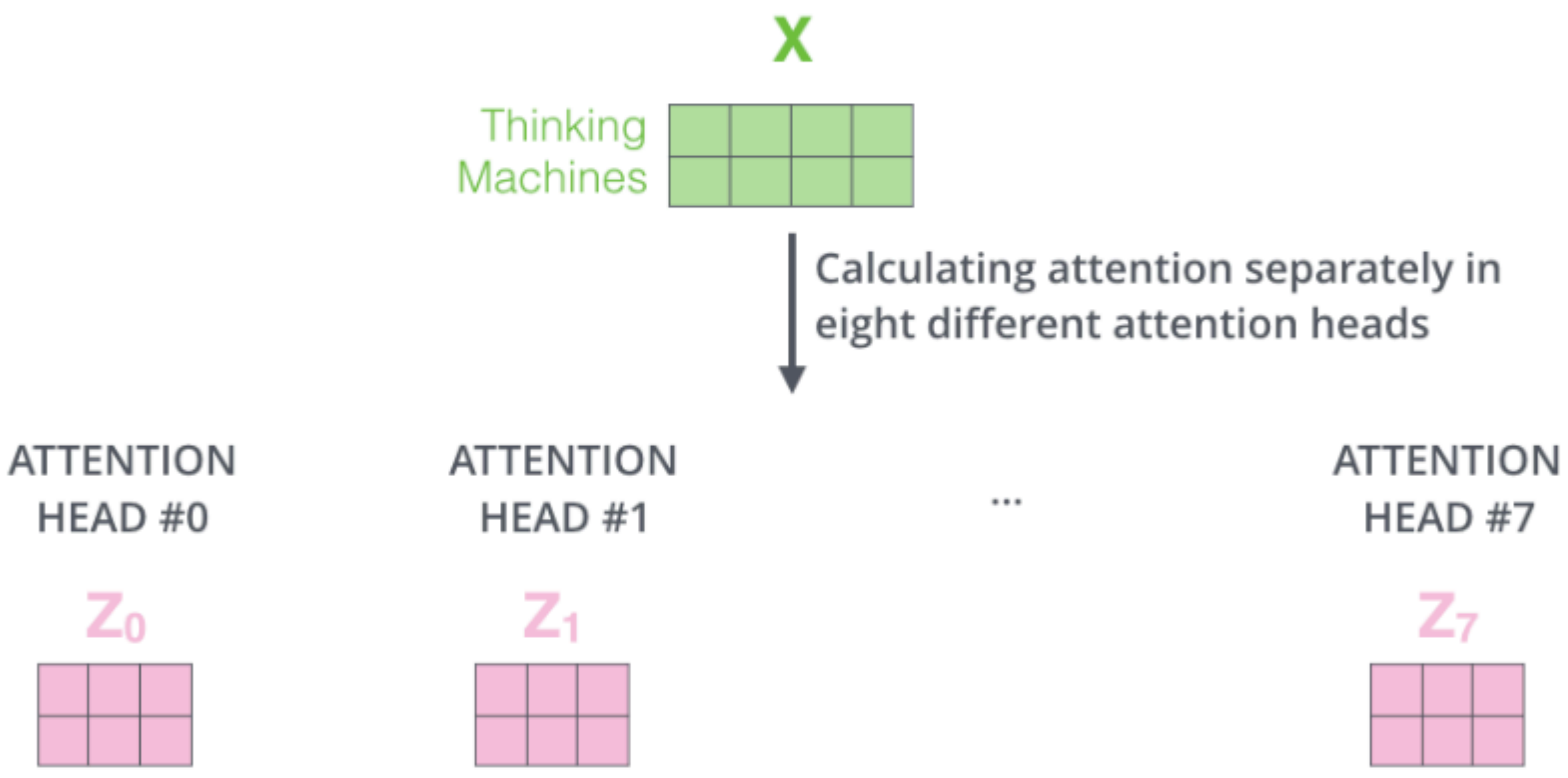
Супермаркет. Каждый ингредиент для окрошки - это **query** - то, что нужно купить. На полках много товаров.
Они могут не также называться, для нас это **key**.
Сравниваем, как сильно похоже описание в списке ингредиентов с товарами.

Для self-attention значения будут немного более smooth.

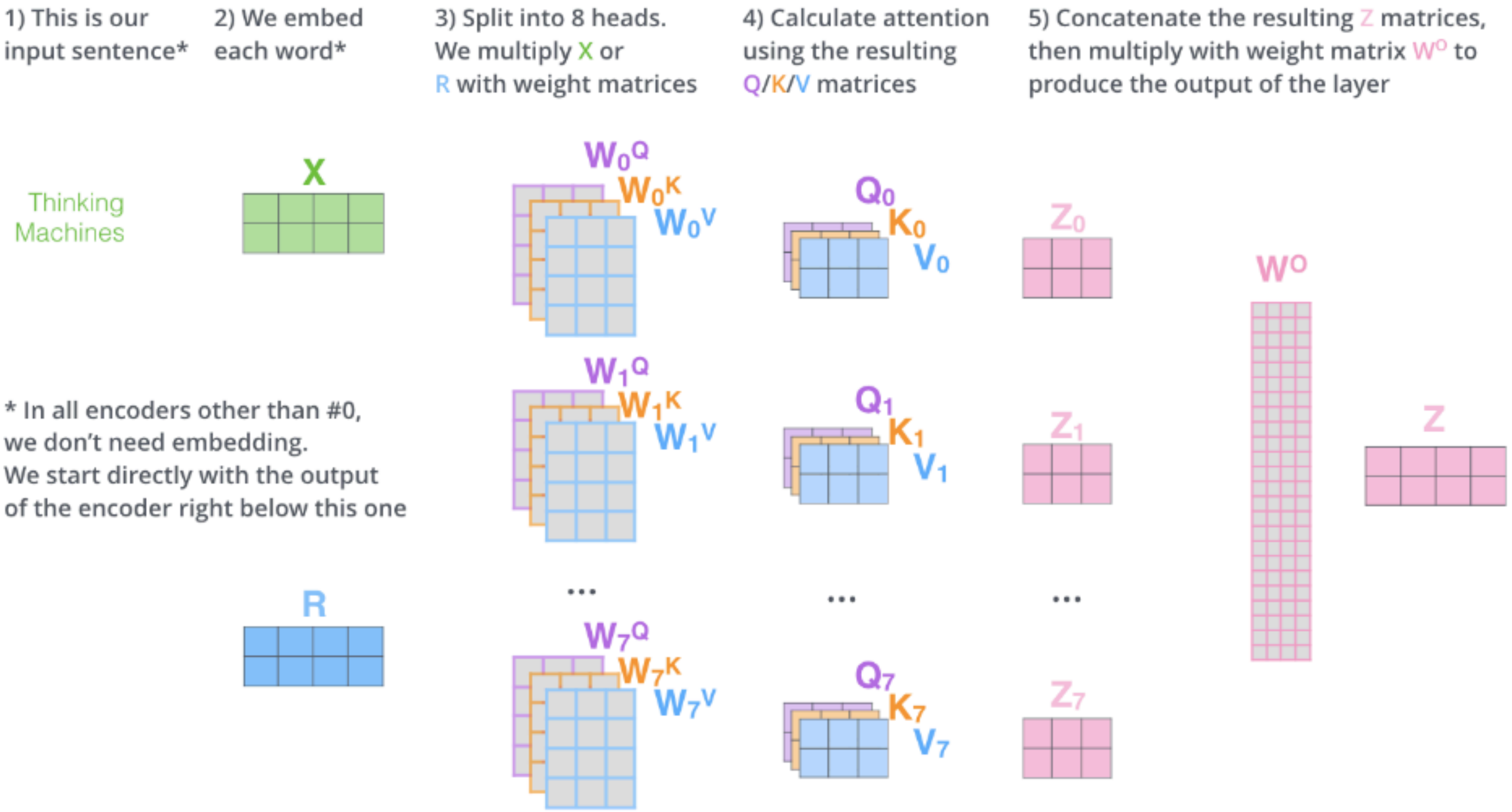
Let’s write from scratch

https://colab.research.google.com/drive/10h66zyXkEyQCLc-DCyhdhrOf3qOzRtZ7?usp=sharing

Multi-head attention

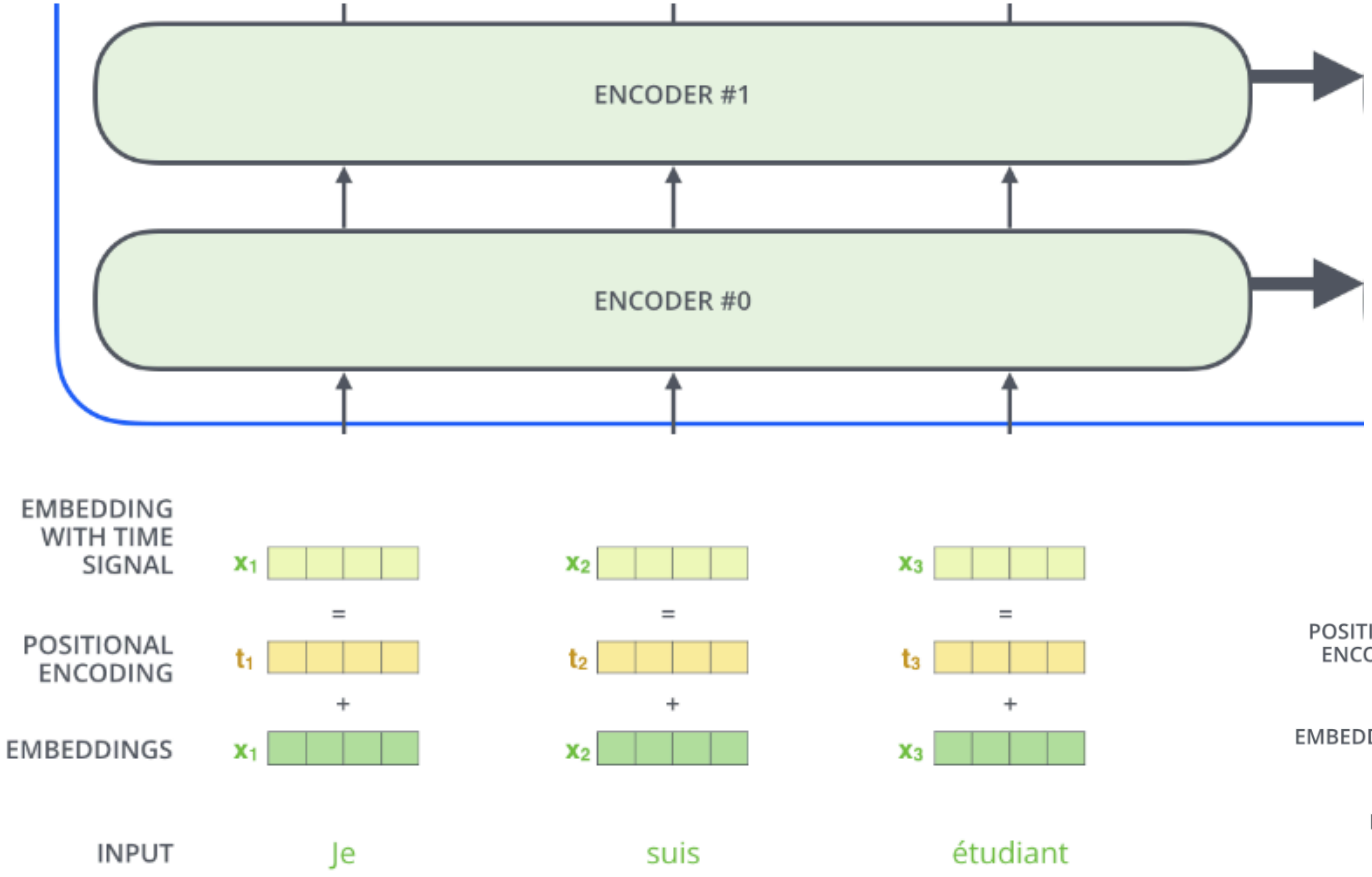


1. **Give multiple representation subspaces**
2. **Focus on different positions**

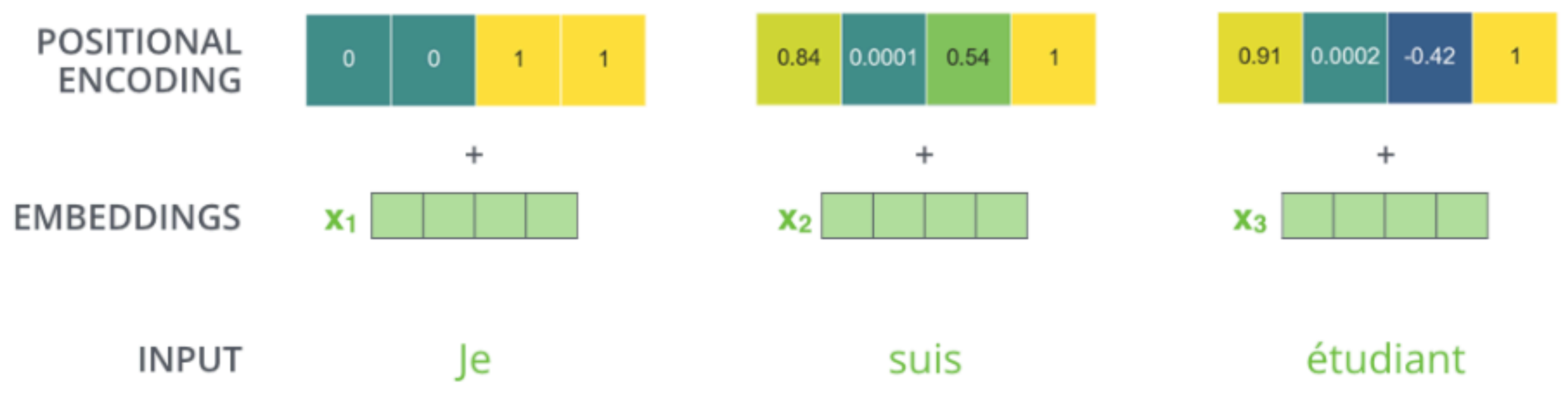


1) This is our input sentence*
2) We embed each word*
3) Split into 8 heads. We multiply X or R with weight matrices
4) Calculate attention using the resulting Q/K/V matrices
5) Concatenate the resulting Z matrices, then multiply with weight matrix W^O to produce the output of the layer
Thinking Machines
X
W_0^Q
W_0^K
W_0^V
Q_0
K_0
V_0
Z_0
W^O
* In all encoders other than #0, we don't need embedding. We start directly with the output of the encoder right below this one
W_1^Q
W_1^K
W_1^V
Q_1
K_1
V_1
Z_1
Z
R
...
W_7^Q
W_7^K
W_7^V
Q_7
K_7
V_7
Z_7

Positional encoding



1. Every position should have the same identifier, irrespectively of the sequence length or input
2. Should not be huge to push the meaning in other subspace

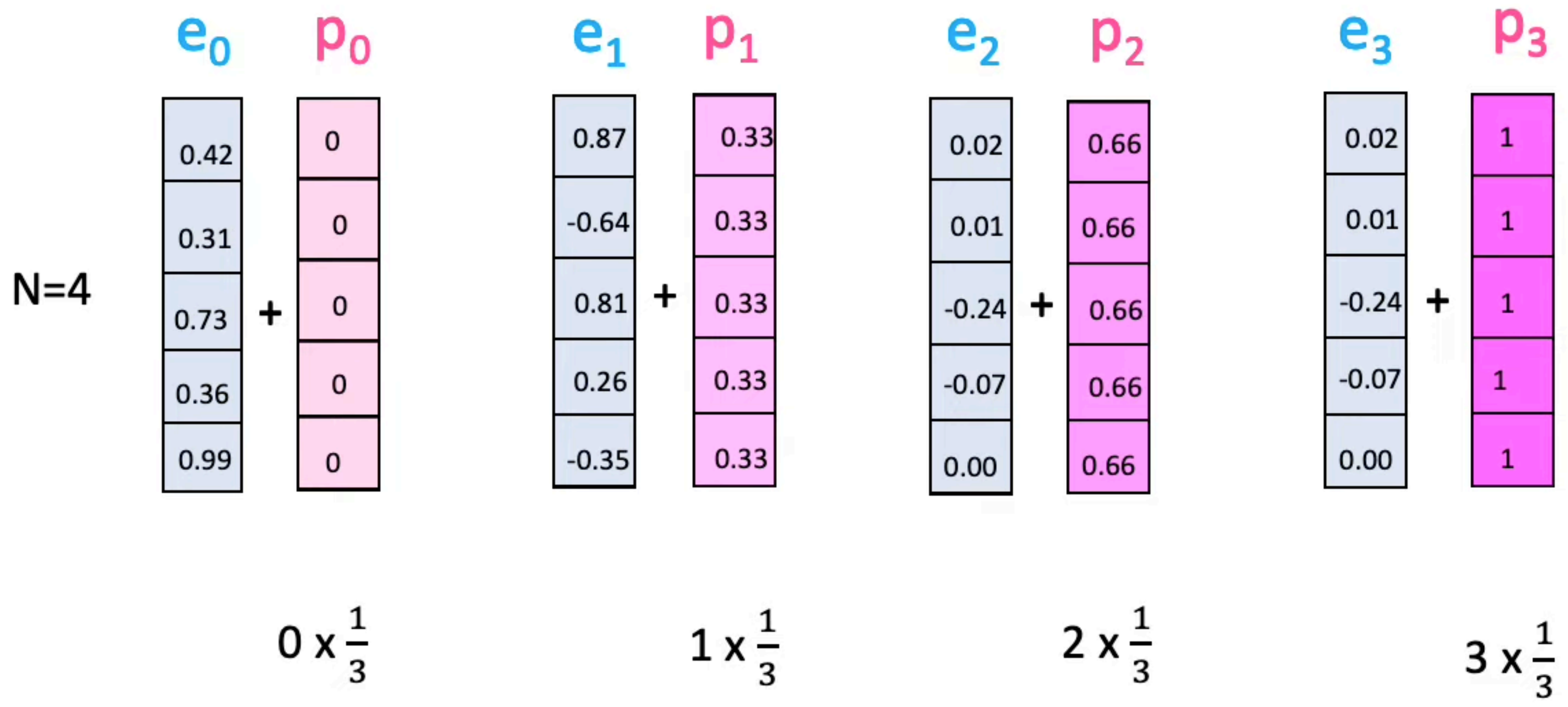


Positional encoding

Even though she did **not** win the award, she was satisfied.
Even though she did win the award, she was **not** satisfied.

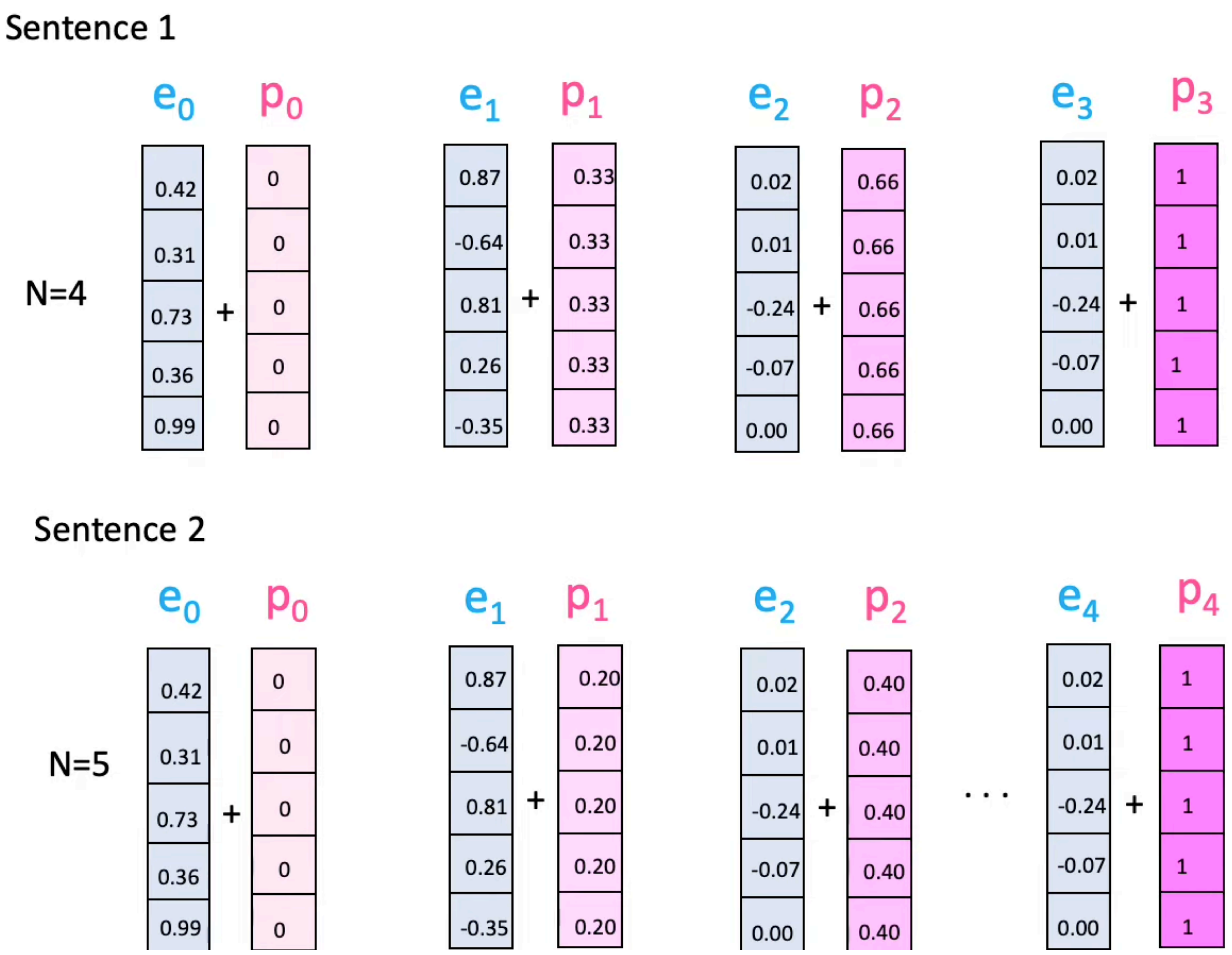
https://www.youtube.com/watch?v=dichIcUZfOw

Positional encoding



$$\frac{1}{N-1} = \frac{1}{3}$$

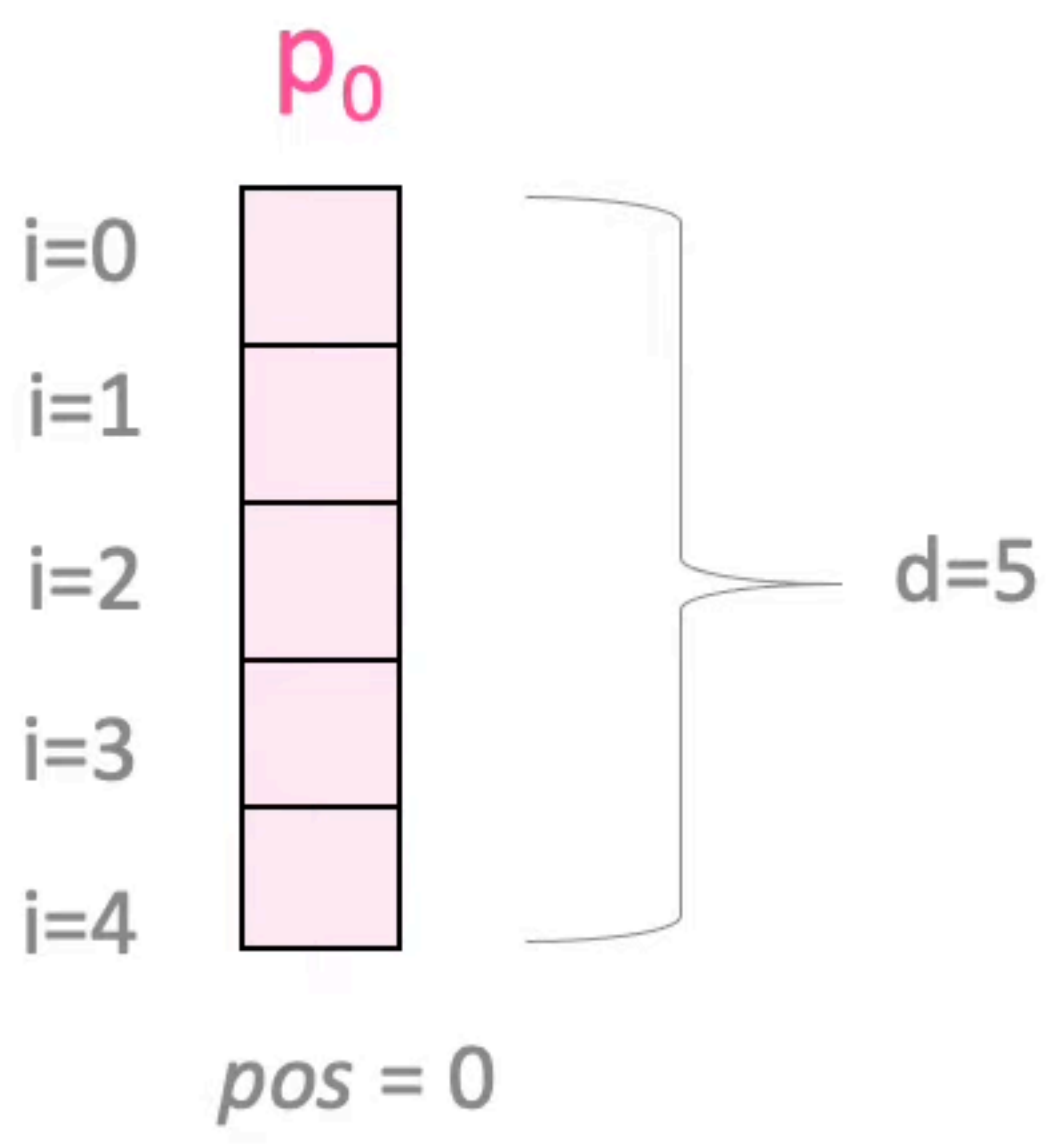
Positional encoding



Positional encoding

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{\frac{2i}{d}}}\right)$$

$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{\frac{2i}{d}}}\right)$$



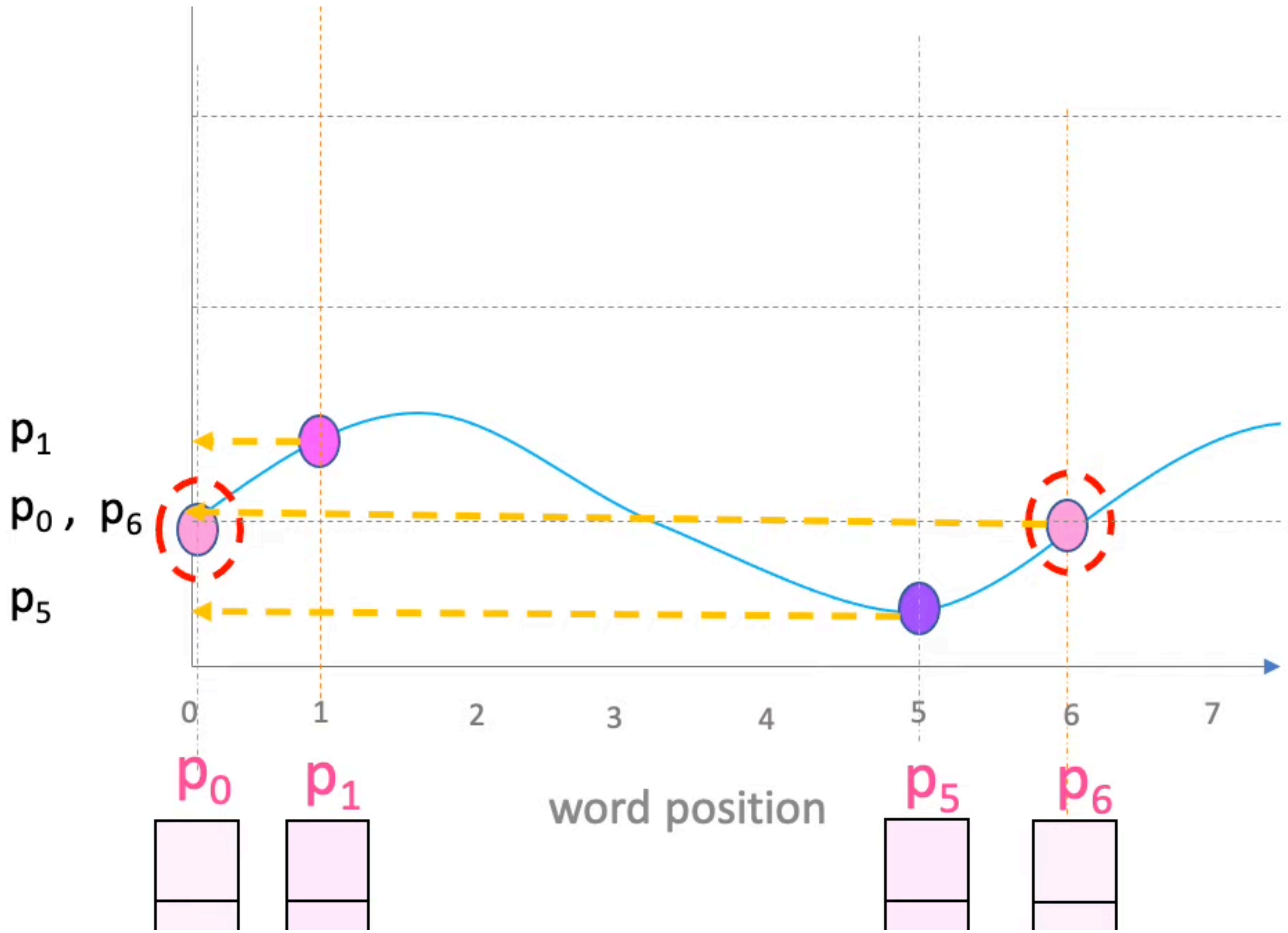
$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{\frac{2i}{d}}}\right)$$

https://www.youtube.com/watch?v=1biZfFLPRSY

Positional encoding

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{\frac{2i}{d}}}\right)$$

$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{\frac{2i}{d}}}\right)$$

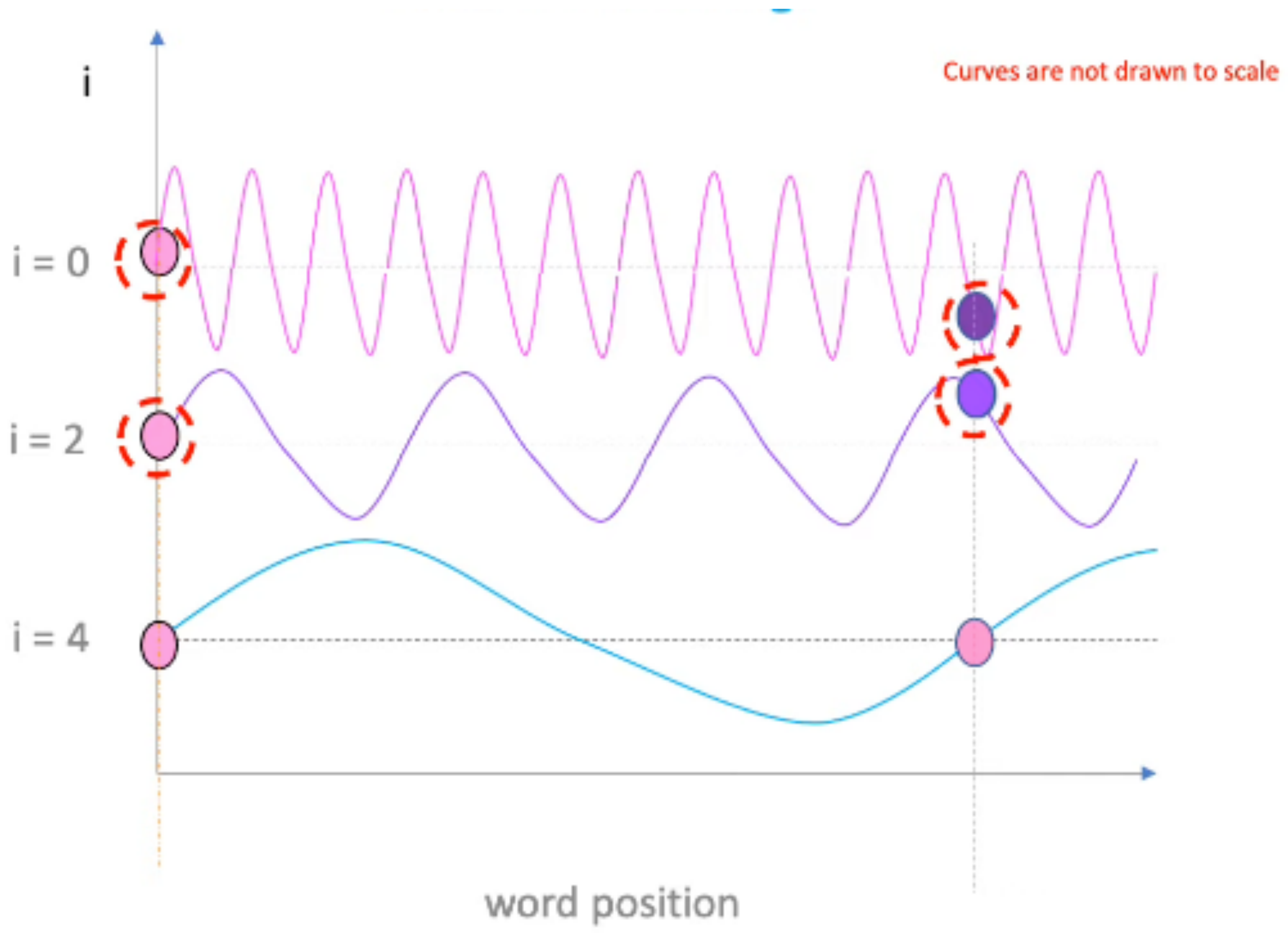


https://www.youtube.com/watch?v=1biZfFLPRSY

Positional encoding

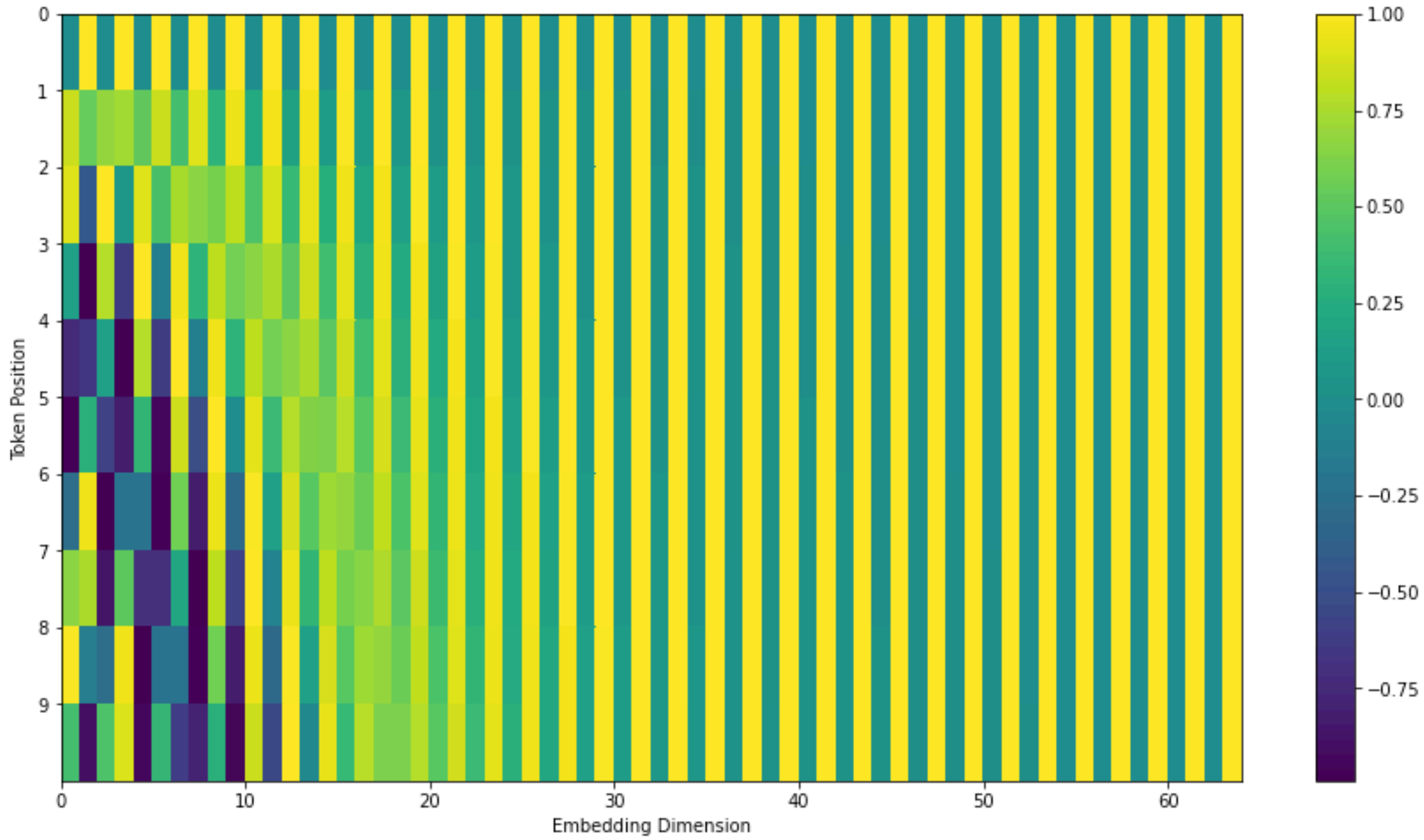
$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{\frac{2i}{d}}}\right)$$

$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{\frac{2i}{d}}}\right)$$



https://www.youtube.com/watch?v=1biZfFLPRSY

Positional encoding

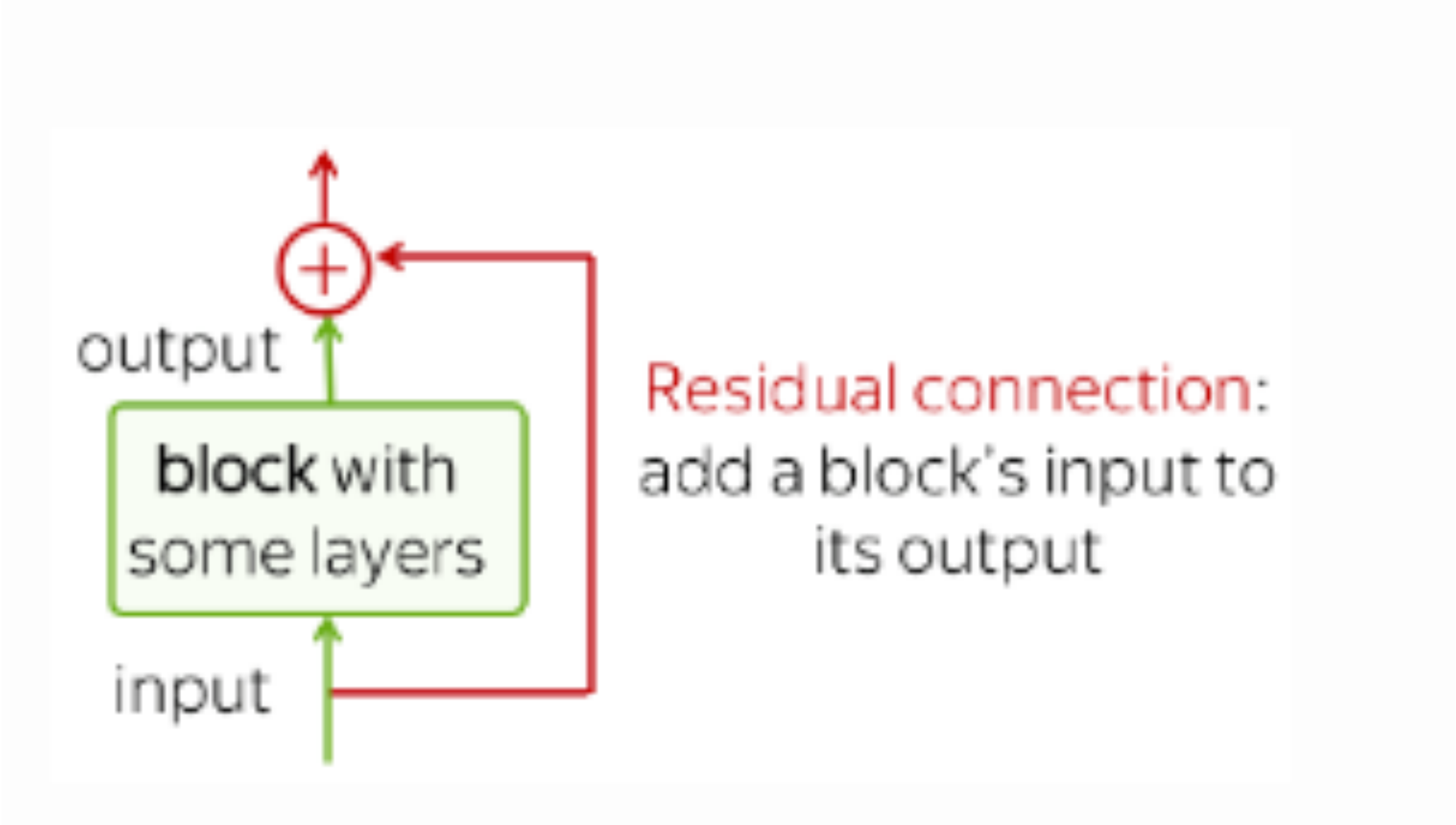


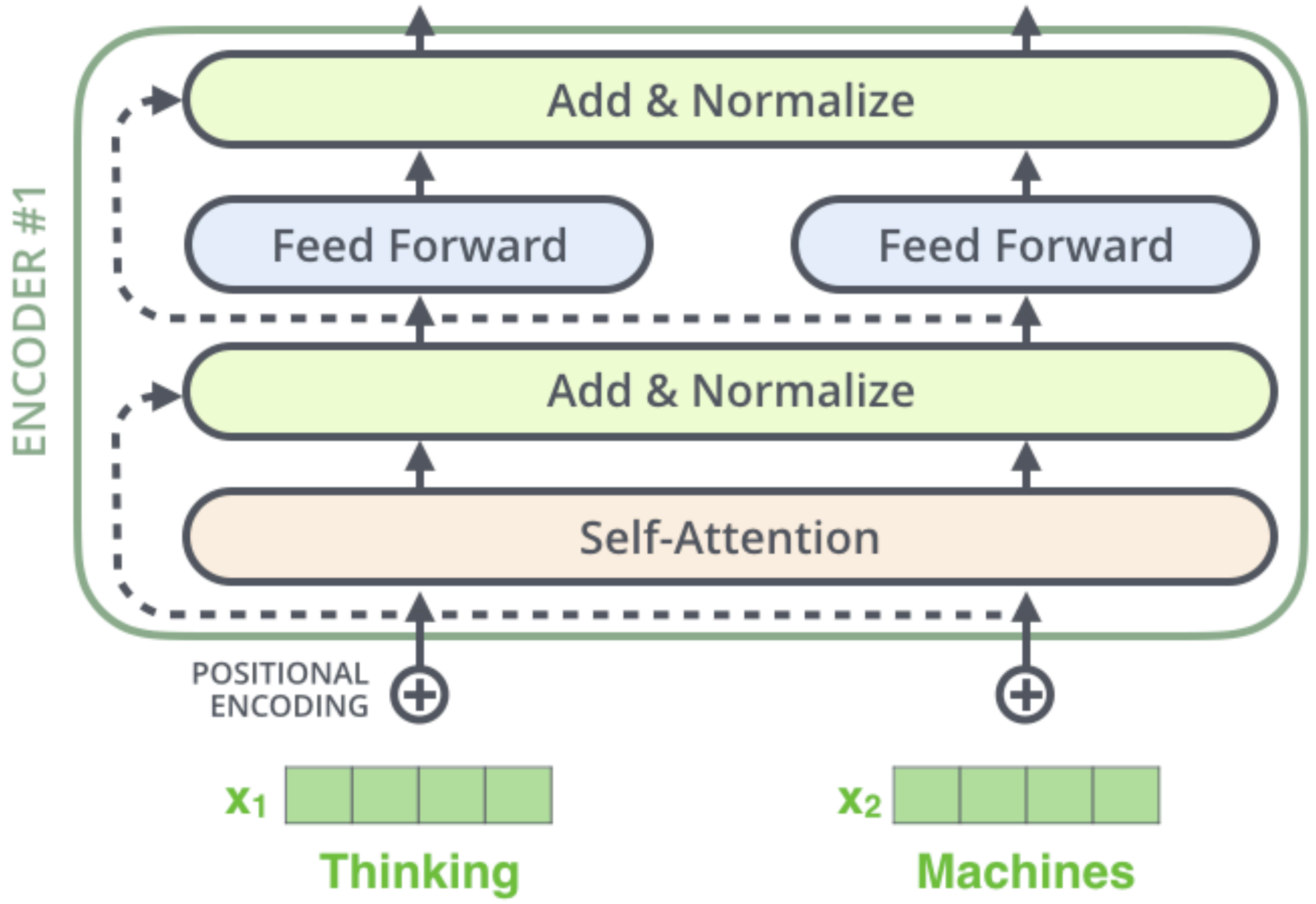
$$\vec{p_t} = \begin{bmatrix} \sin(\omega_1 . t) \\ \cos(\omega_1 . t) \\ \\ \sin(\omega_2 . t) \\ \cos(\omega_2 . t) \\ \\ \vdots \\ \\ \sin(\omega_{d/2} . t) \\ \cos(\omega_{d/2} . t) \end{bmatrix}_{d \times 1}$$

https://www.youtube.com/watch?v=1biZfFLPRSY

Residual connection

directly passing "raw" embeddings to the next layer can actually be very helpful!





https://stats.stackexchange.com/questions/565196/why-are-residual-connections-needed-in-transformer-architectures

Layer Norm and add

Normalize each input in the batch to have zero mean and unity variance.

Scale and bias = trainable parameters and used after normalization to rescale output.

The difference between the two arrangements is illustrated in Figure 3-6.

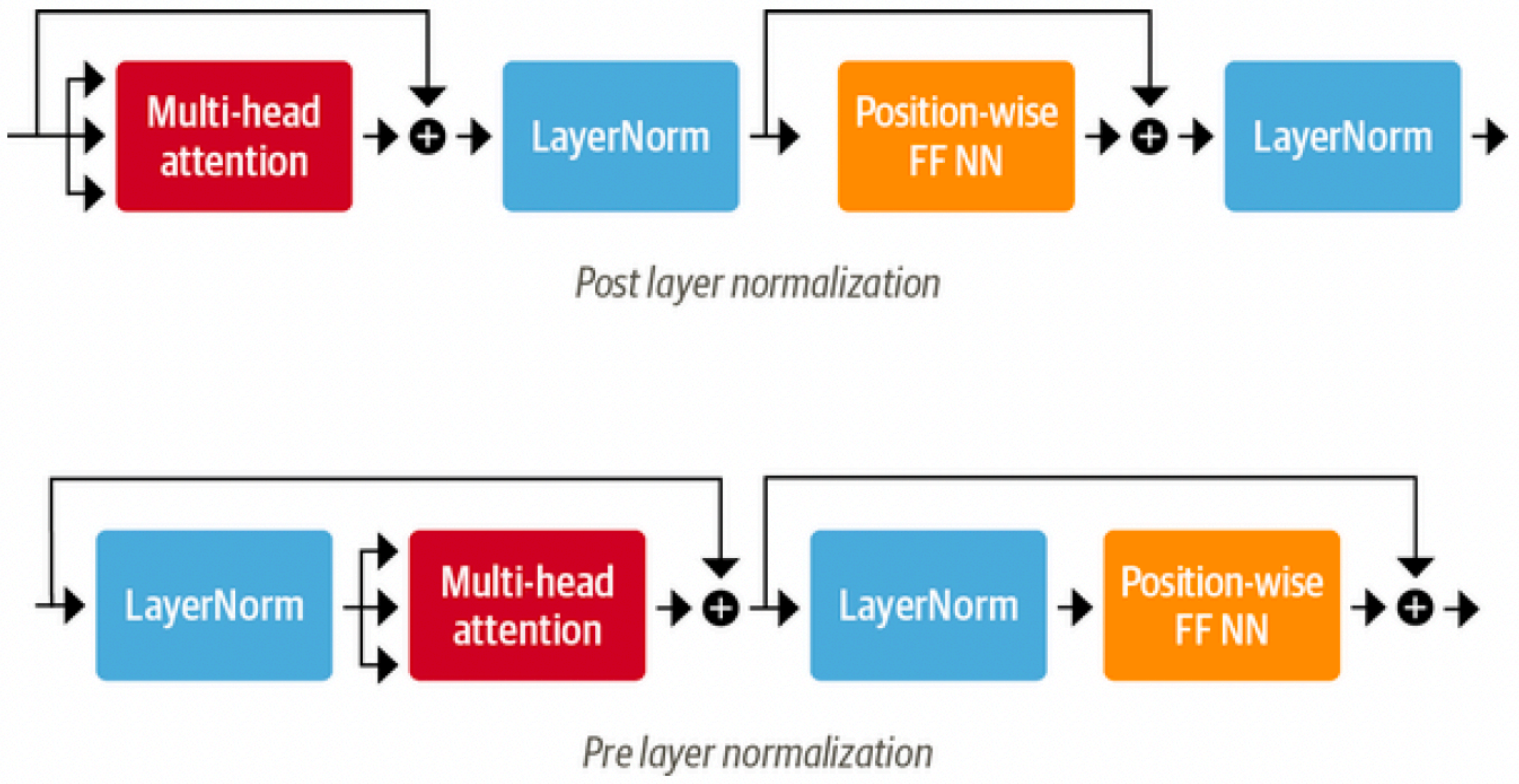
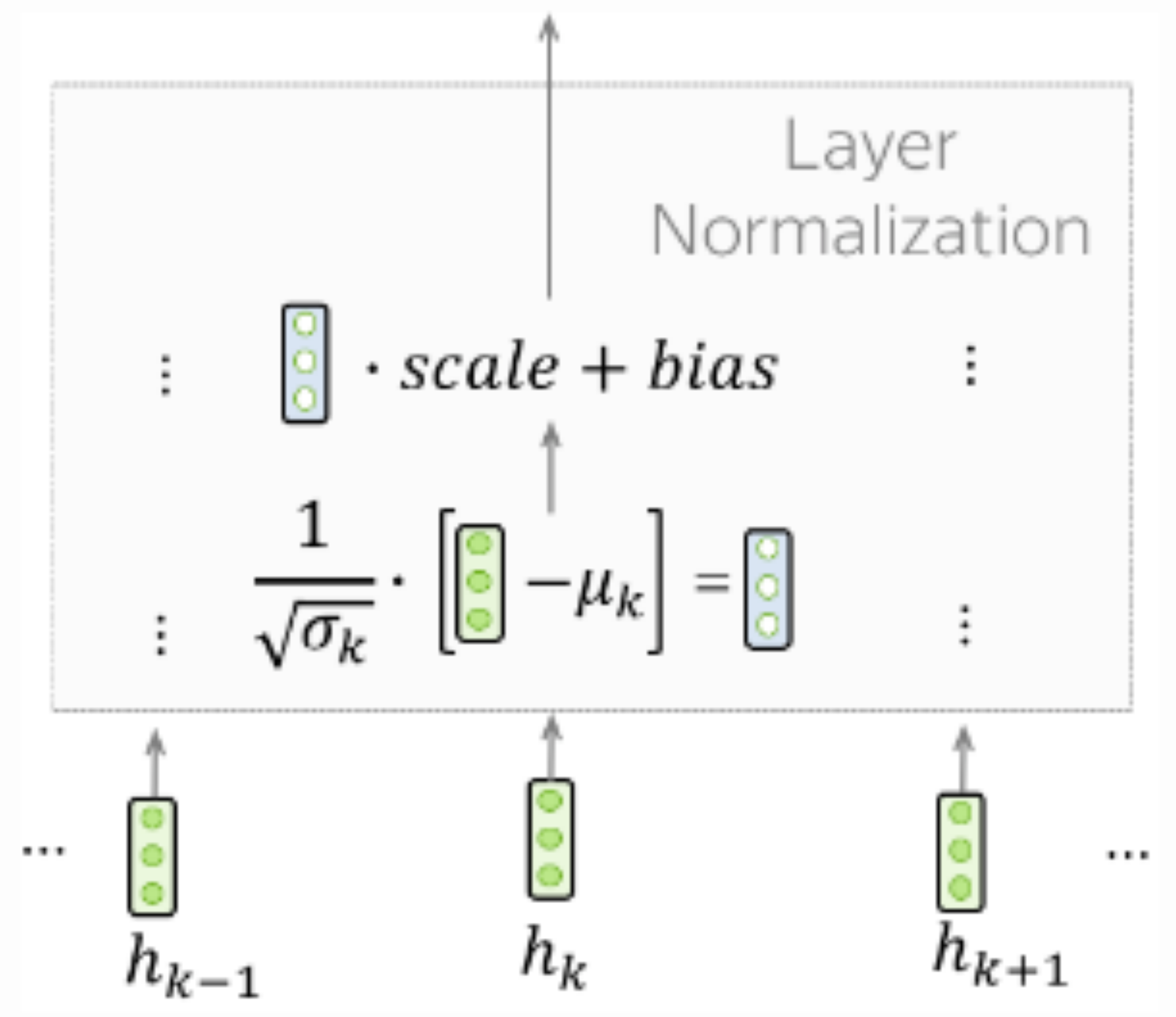


Figure 3-6. Different arrangements of layer normalization in a transformer encoder layer



https://arxiv.org/pdf/1607.06450.pdf

Local vs global

Local VS global attention

global VS local - в глобал мы берем все стейты от encoder, в local предсказываем позициии токенов, на которые будем смотреть.

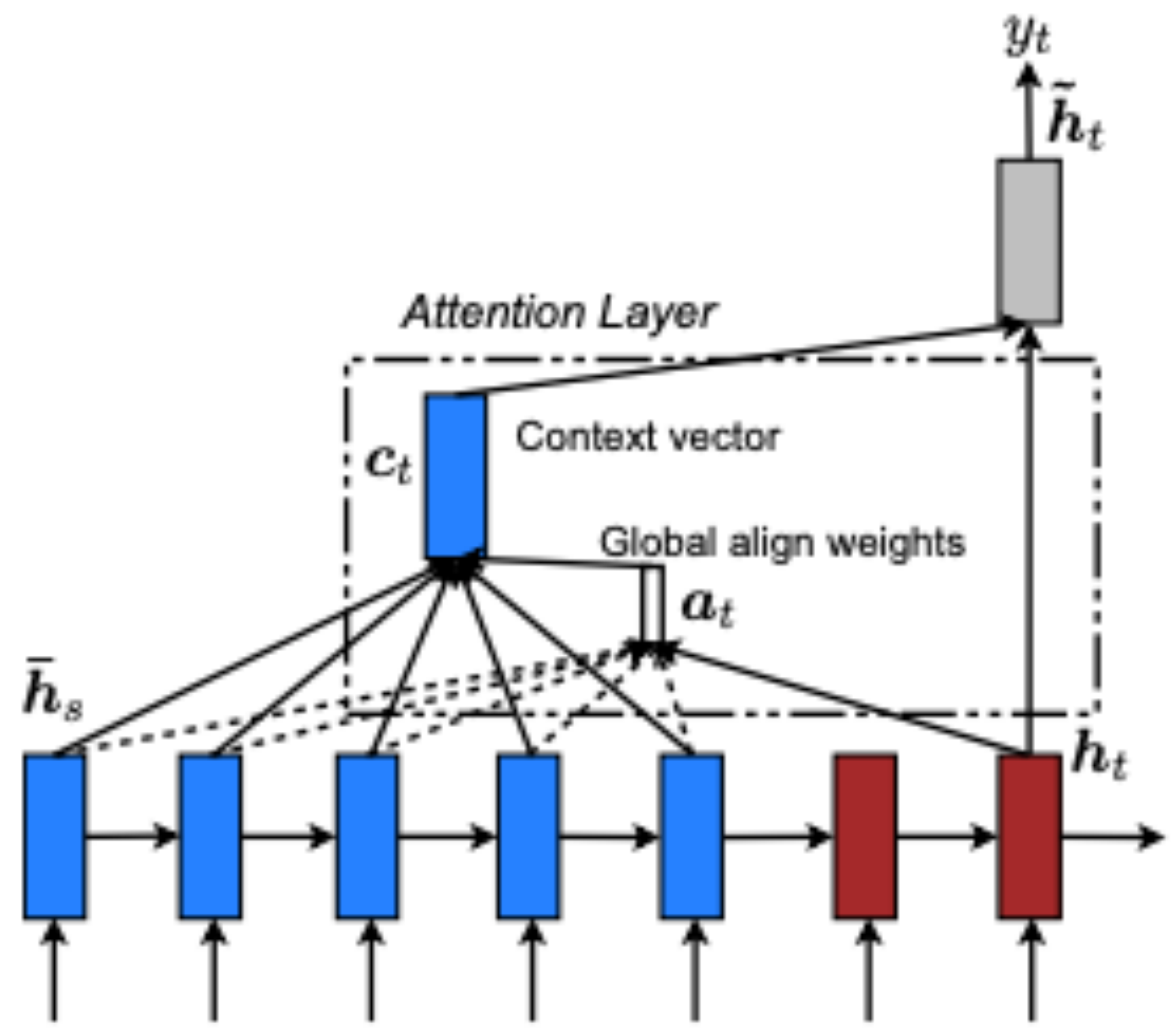


Figure 2: **Global attentional model** – at each time step t, the model infers a *variable-length* alignment weight vector $\boldsymbol{a}_t$ based on the current target state $\boldsymbol{h}_t$ and all source states $\bar{\boldsymbol{h}}_s$. A global context vector $\boldsymbol{c}_t$ is then computed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

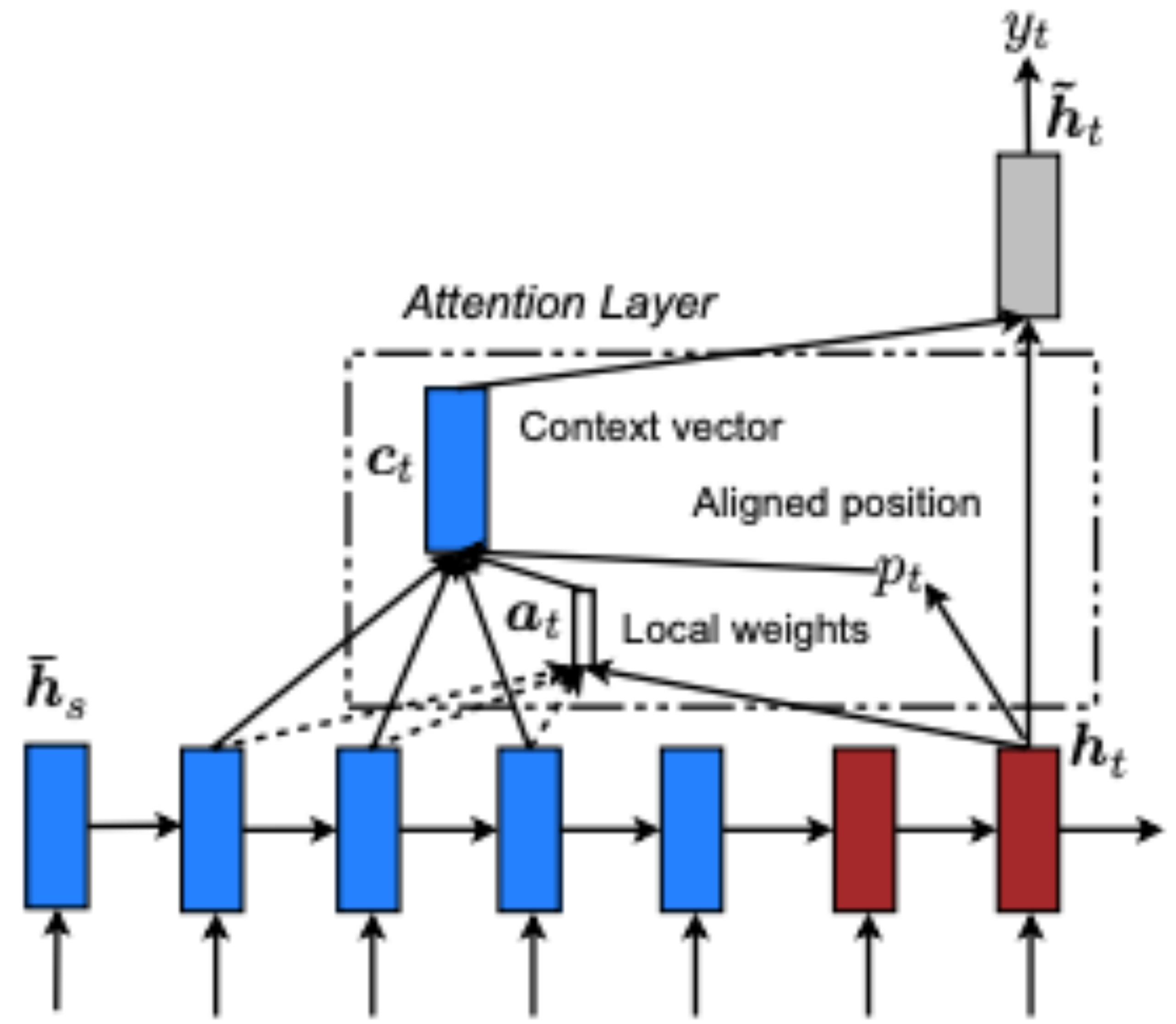


Figure 3: **Local attention model** – the model first predicts a single aligned position p_t for the current target word. A window centered around the source position p_t is then used to compute a context vector $\boldsymbol{c}_t$, a weighted average of the source hidden states in the window. The weights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

Transformers