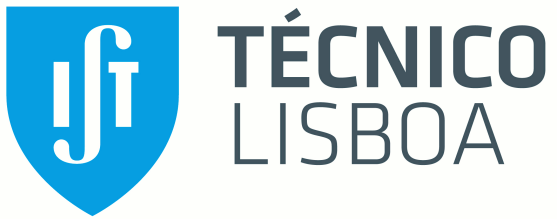
Lecture 10:
Attention Mechanisms and Transformers

André Martins, Francisco Melo, Mário Figueiredo



Deep Learning Course, Winter 2022-2023

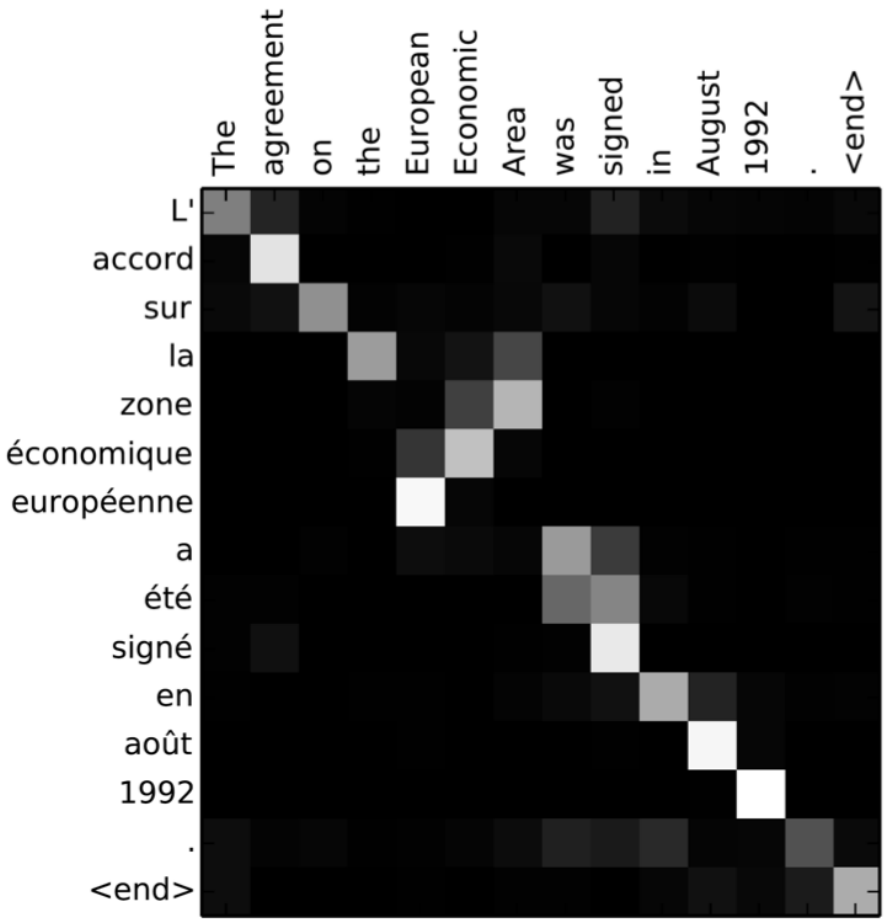
Why Attention?

We want NNs that automatically weigh input relevance

Main advantages:

- performance gain
- none or few parameters
- fast (easy to parallelize)
- tool for "interpreting" predictions

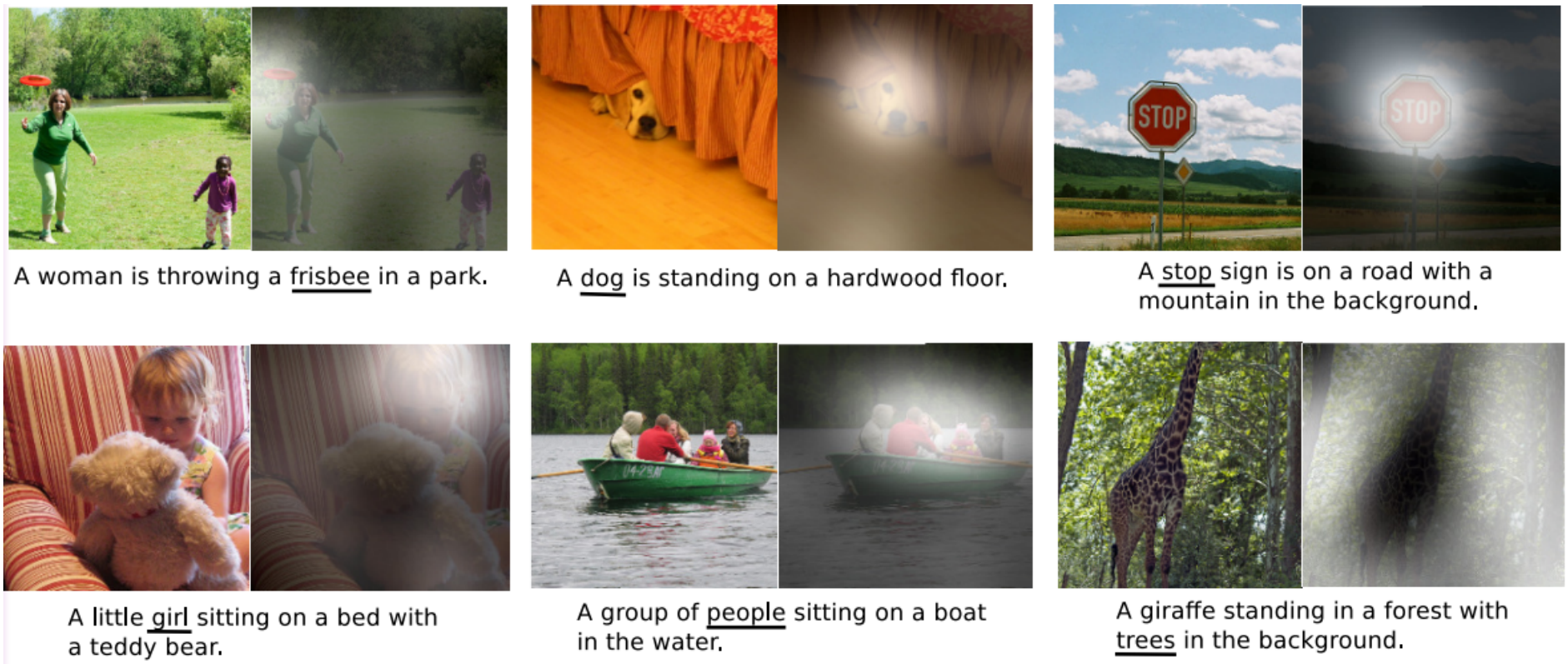
Example: Machine Translation



Dzmitry Bahdanau, KyungHuyn Cho, and Yoshua Bengio. ***Neural Machine Translation by Jointly Learning to Translate and Align****. ICLR'15.*

Example: Caption Generation

Attention over images:



(Slide credit to Yoshua Bengio)

Example: Document Classification

Task: Hotel location

you get what you pay for . not the cleanest rooms but bed was clean and so was bathroom . bring your own towels though as very thin . service was excellent , let us book in at 8:30am ! **for location and price , this ca n't be beaten** , but it is cheap for a reason . if you come expecting the hilton , then book the hilton ! for uk travellers , think of a blackpool b&b.

Task: Hotel cleanliness

you get what you pay for . **not the cleanest rooms but bed was clean and so was bathroom** . bring your own towels though as very thin . service was excellent , let us book in at 8:30am ! for location and price , this ca n't be beaten , but it is cheap for a reason . if you come expecting the hilton , then book the hilton ! for uk travellers , think of a blackpool b&b.

Task: Hotel service

you get what you pay for . not the cleanest rooms but bed was clean and so was bathroom . bring your own towels though as very thin . **service was excellent** , let us book in at 8:30am ! for location and price , this ca n't be beaten , but it is cheap for a reason . if you come expecting the hilton , then book the hilton ! for uk travellers , think of a blackpool b&b.

(Bao et al., 2018)

Attention Mechanism: Recap

Recall how attention works:

1. We have a query vector $\boldsymbol{q}$ (e.g. the decoder state)
2. We have input vectors $\boldsymbol{H} = [\boldsymbol{h}_1, \ldots, \boldsymbol{h}_L]^\top$ (e.g. one per source word)
3. We compute affinity scores $s_1, \ldots, s_L$ by "comparing" $\boldsymbol{q}$ and $\boldsymbol{H}$
4. We convert these scores to probabilities:

$$\boldsymbol{p} = \mathbf{softmax}(\boldsymbol{s})$$

5. We use this to output a representation as a weighted average:

$$\boldsymbol{c} = \boldsymbol{H}^\top \boldsymbol{p} = \sum_{i=1}^{L} p_i \boldsymbol{h}_i$$

Let's see these steps in detail!

Attention Mechanism: More General Version

1. We have a query vector $\boldsymbol{q}$ (e.g. the decoder state)

2. We have key vectors $\boldsymbol{K} = [\boldsymbol{k}_1, \ldots, \boldsymbol{k}_L]^\top \in \mathbb{R}^{L \times d_K}$
   and value vectors $\boldsymbol{V} = [\boldsymbol{v}_1, \ldots, \boldsymbol{v}_L]^\top \in \mathbb{R}^{L \times d_V}$
   (e.g. one of each per source word)

3. We compute query-key affinity scores $s_1, \ldots, s_L$ "comparing" $\boldsymbol{q}$ and $\boldsymbol{K}$

4. We convert these scores to probabilities:

$$\boldsymbol{p} = \mathbf{softmax}(\boldsymbol{s})$$

5. We output a weighted average of the values:

$$\boldsymbol{c} = \boldsymbol{V}^\top \boldsymbol{p} = \sum_{i=1}^{L} p_i \boldsymbol{v}_i \in \mathbb{R}^{d_V}$$

Self-Attention Layer

Self-attention for a sequence of length L:

1. Query vectors $\boldsymbol{Q} = [\boldsymbol{q}_1, \ldots, \boldsymbol{q}_L]^\top \in \mathbb{R}^{L \times d_Q}$
2. Key vectors $\boldsymbol{K} = [\boldsymbol{k}_1, \ldots, \boldsymbol{k}_L]^\top \in \mathbb{R}^{L \times d_K}$
3. value vectors $\boldsymbol{V} = [\boldsymbol{v}_1, \ldots, \boldsymbol{v}_L]^\top \in \mathbb{R}^{L \times d_V}$
4. Compute query-key affinity scores "comparing" $\boldsymbol{Q}$ and $\boldsymbol{K}$, e.g.,

$$\boldsymbol{S} = \boldsymbol{Q}\boldsymbol{K}^\top \in \mathbb{R}^{L \times L} \qquad \text{(dot-product affinity)}$$

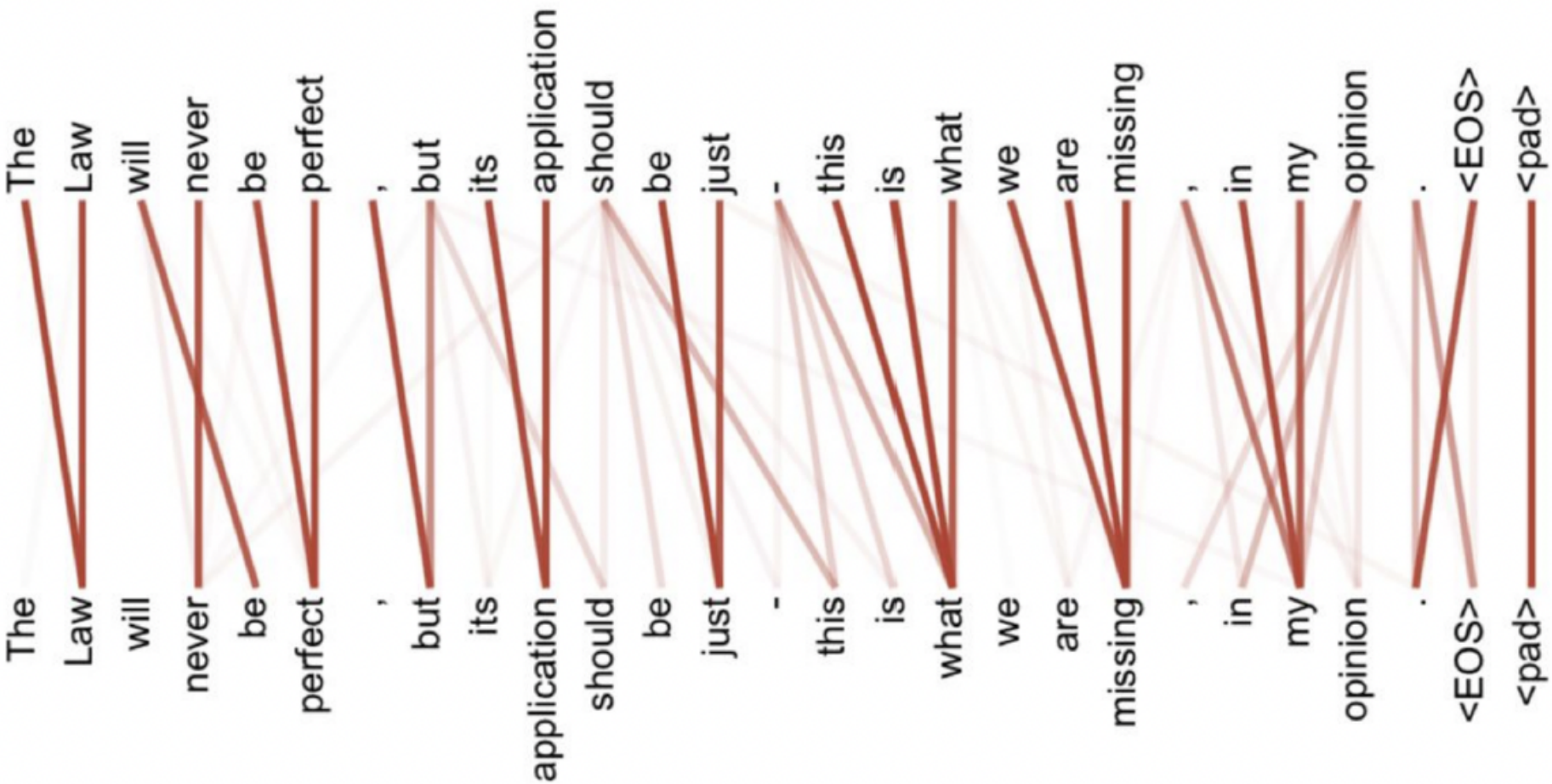
5. Convert these scores to probabilities (row-wise):

$$\boldsymbol{P} = \mathbf{softmax}(\boldsymbol{S}) \in \mathbb{R}^{L \times L}$$

6. Output the weighted average of the values:

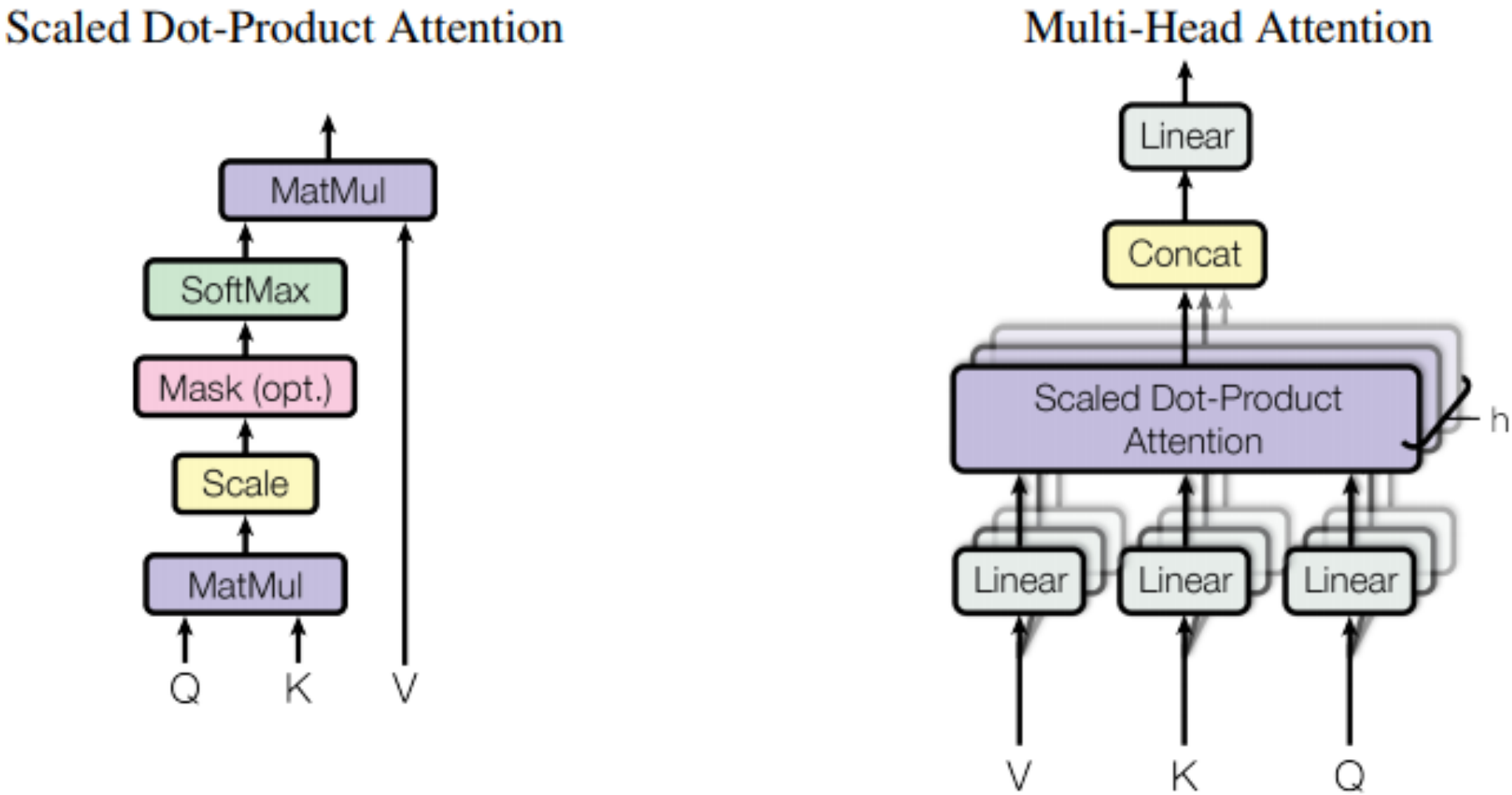
$$\boldsymbol{Z} = \boldsymbol{P}\boldsymbol{V} = \underbrace{\mathbf{softmax}(\boldsymbol{Q}\boldsymbol{K}^\top)}_{\boldsymbol{P}}\,\boldsymbol{V} \in \mathbb{R}^{L \times d_V}.$$

Self-Attention



(Vaswani et al., 2017)

Scaled Dot-Product and Multi-Head Attention



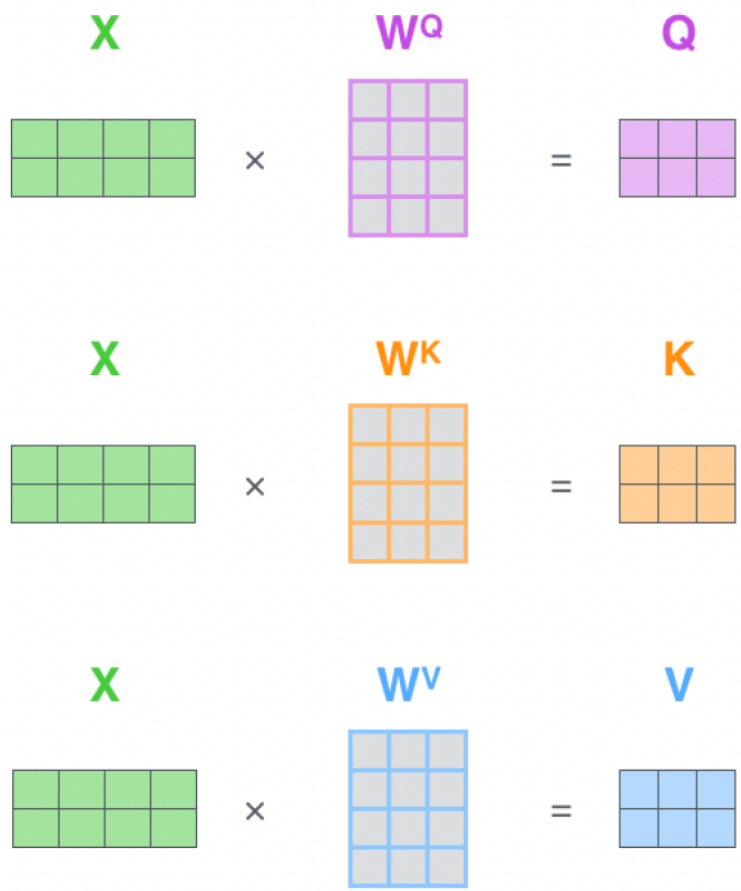
(Vaswani et al., 2017)

Transformer Self-Attention: Queries, Keys, Vectors

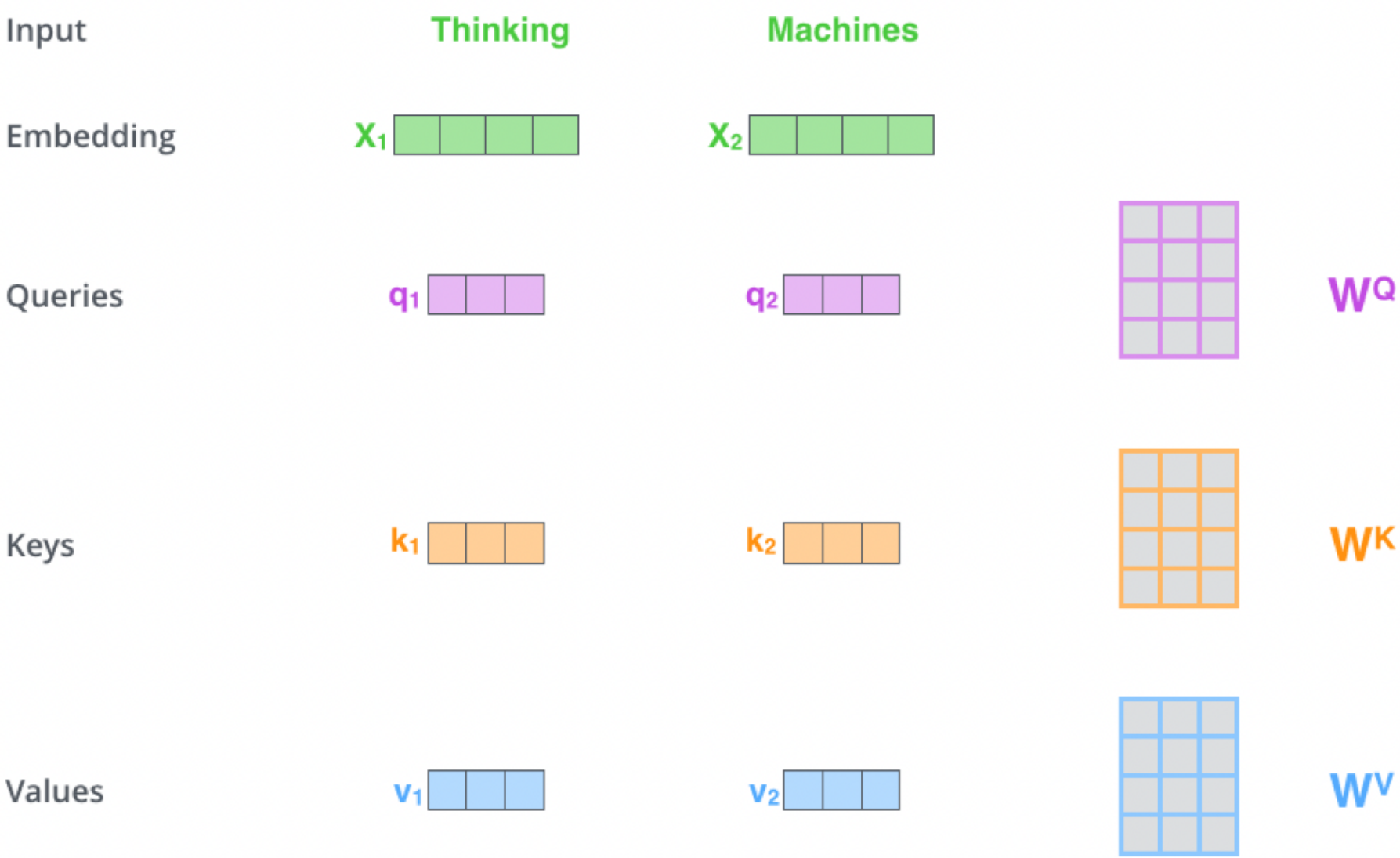
- Obtained by projecting the embedding matrix $\boldsymbol{X} \in \mathbb{R}^{L \times e}$ to a lower dimension:

$$\boldsymbol{Q} = \boldsymbol{X}\boldsymbol{W}^{Q}$$
$$\boldsymbol{K} = \boldsymbol{X}\boldsymbol{W}^{K}$$
$$\boldsymbol{V} = \boldsymbol{X}\boldsymbol{W}^{V}.$$

- The projection matrices $\boldsymbol{W}^{Q}$, $\boldsymbol{W}^{K}$, $\boldsymbol{W}^{V}$ are model parameters.



Transformer Self-Attention: Queries, Keys, Vectors



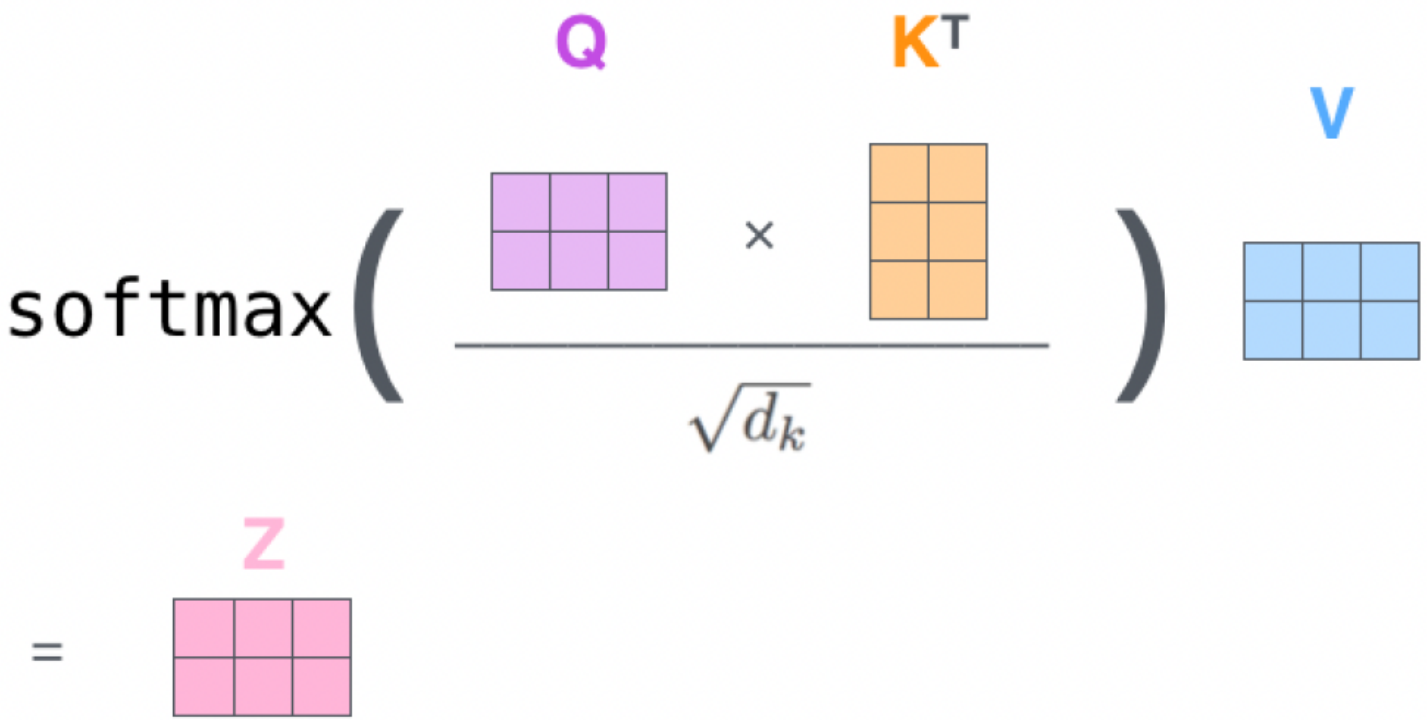
Scaled Dot-Product Attention

Problem: As d_K gets large, the variance of $\boldsymbol{q}^\top \boldsymbol{k}$ increases, the softmax gets very peaked, hence its gradient gets smaller.

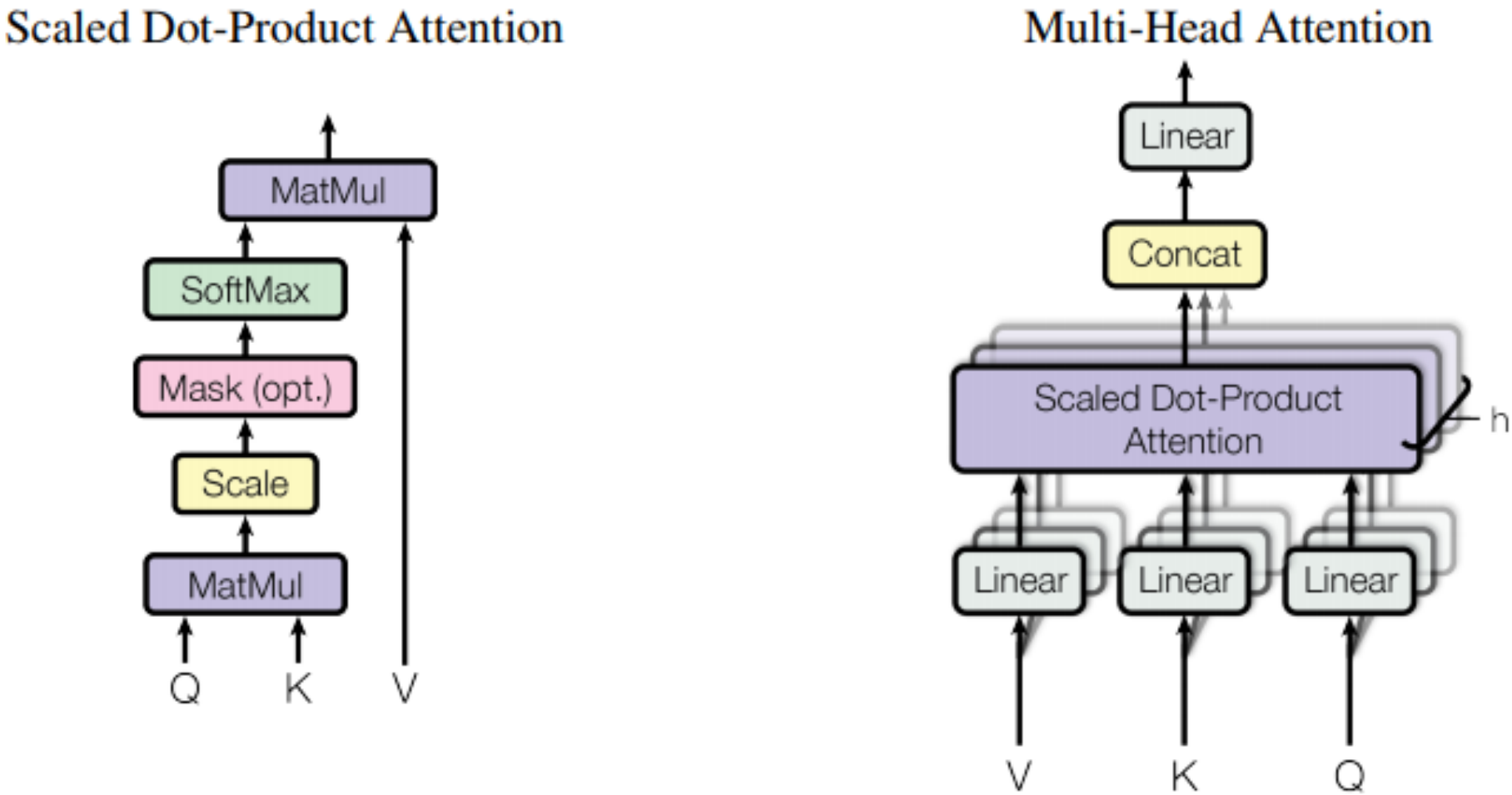
Solution: scale by length of query/key vectors:

$$\boldsymbol{Z} = \textbf{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{d_K}}\right)\boldsymbol{V}.$$

Scaled Dot-Product Attention



Scaled Dot-Product and Multi-Head Attention



(Vaswani et al., 2017)

Multi-Head Attention

Self-attention: each word forms a query vector and attends to the other words' key vectors

This is vaguely similar to a 1D convolution, but where the filter weights are "dynamic" is the window size spans the entire sentence!

Problem: only one channel for words to interact with one-another

Solution: multi-head attention!

- define h attention heads, each with their own projection matrices (e.g. $h = 8$)
- apply attention in multiple channels, concatenate the outputs and pipe through linear layer:

$$\text{MultiHead}(\boldsymbol{X}) = \text{Concat}(\boldsymbol{Z}_1, \ldots, \boldsymbol{Z}_h)\boldsymbol{W}^O,$$

where $\boldsymbol{Z}_i = \text{Attention}(\underbrace{\boldsymbol{X}\boldsymbol{W}_i^Q}_{\boldsymbol{Q}_i}, \underbrace{\boldsymbol{X}\boldsymbol{W}_i^K}_{\boldsymbol{K}_i}, \underbrace{\boldsymbol{X}\boldsymbol{W}_i^V}_{\boldsymbol{V}_i})$.

Multi-Head Attention

1) This is our input sentence*

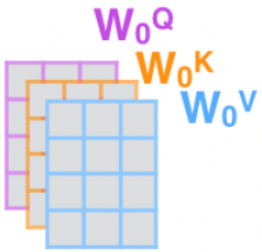
2) We embed each word*

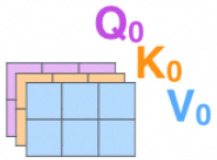
3) Split into 8 heads. We multiply X or R with weight matrices

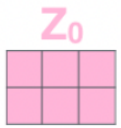
4) Calculate attention using the resulting Q/K/V matrices

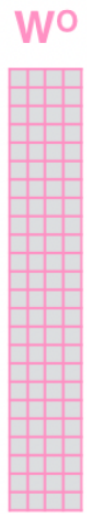
5) Concatenate the resulting Z matrices, then multiply with weight matrix W^O to produce the output of the layer

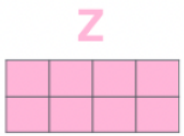
Thinking Machines

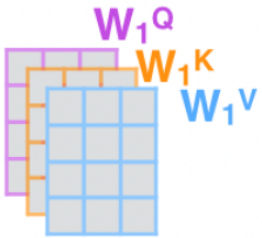


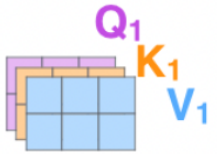


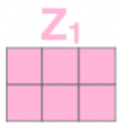




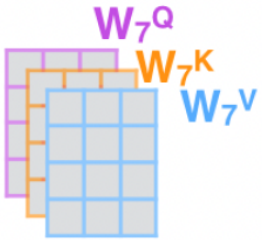


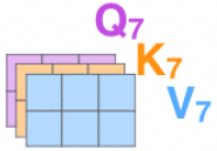


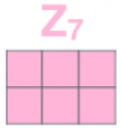




* In all encoders other than #0, we don't need embedding. We start directly with the output of the encoder right below this one







Other Tricks

- Self-attention blocks are repeated several times (e.g. 6 or 12)
- Residual connections on each attention block
- Layer normalization
- Positional encodings (to distinguish word positions)

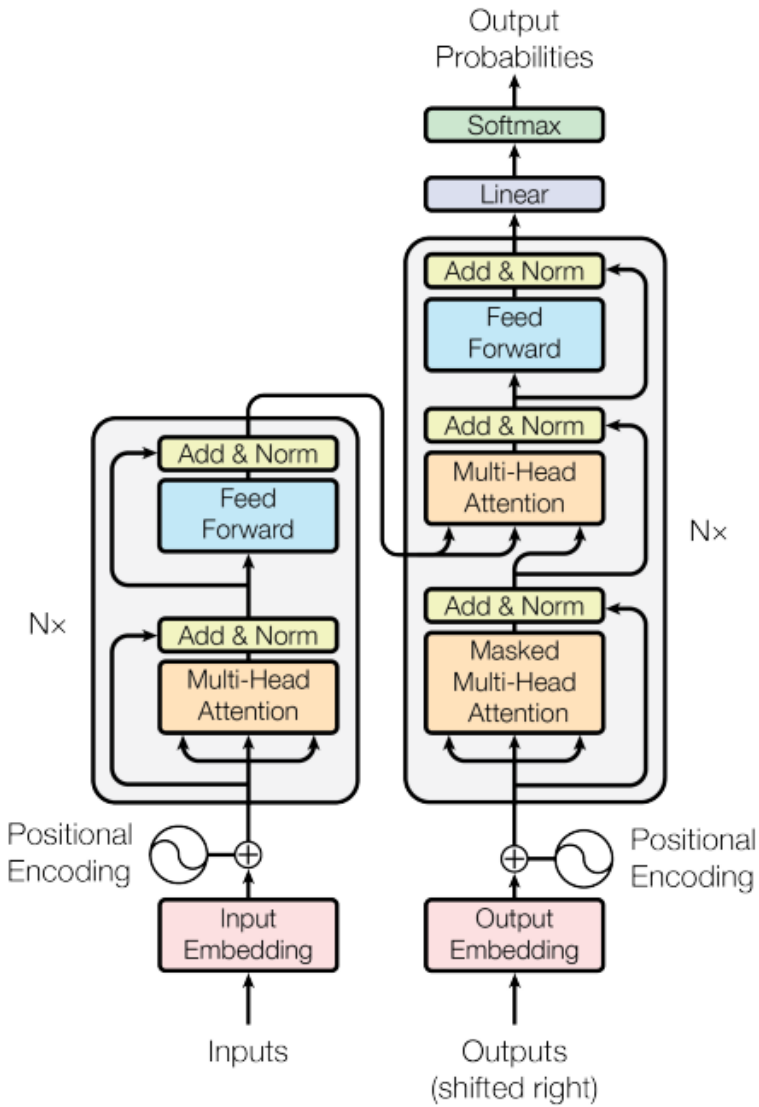
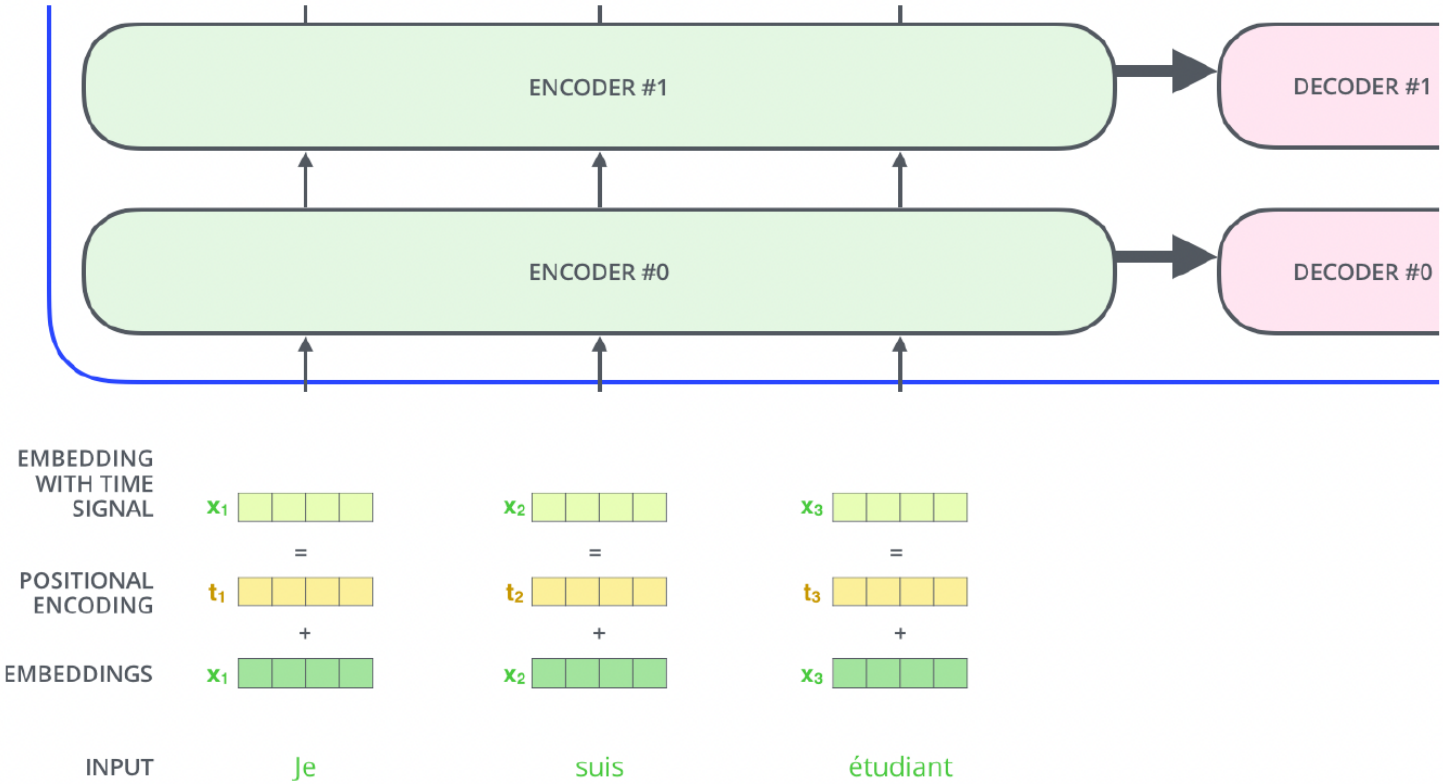


Figure 1: The Transformer - model architecture.

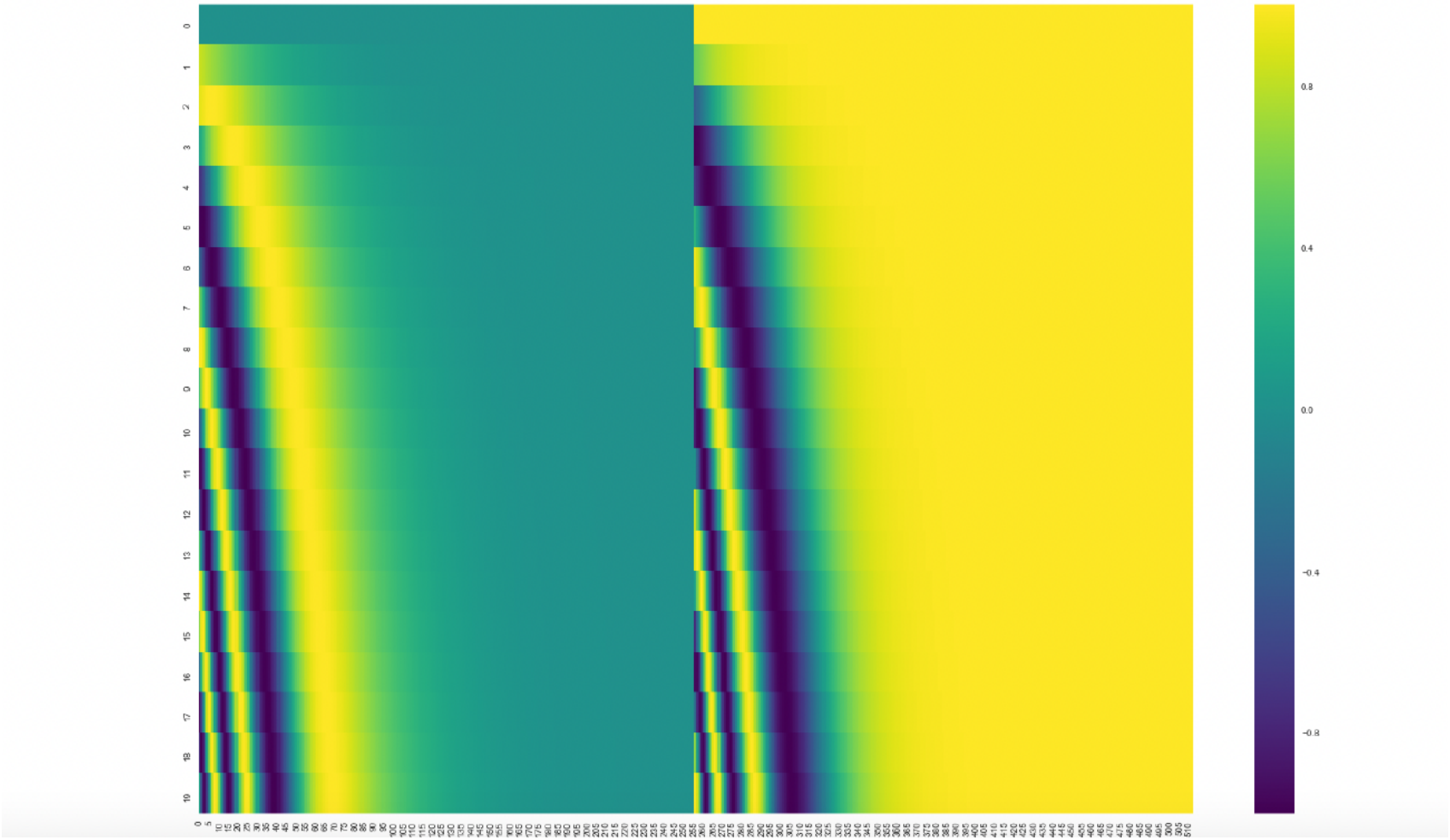
Positional Encodings

- As just described, the transformer is insensitive to word order!
  - queries attend to keys regardless of their position in the sequence
- To make it sensitive to order, we add positional encodings
- Two strategies: learn one embedding for each position (up to a maximum length) or use sinusoidal positional encodings (next)



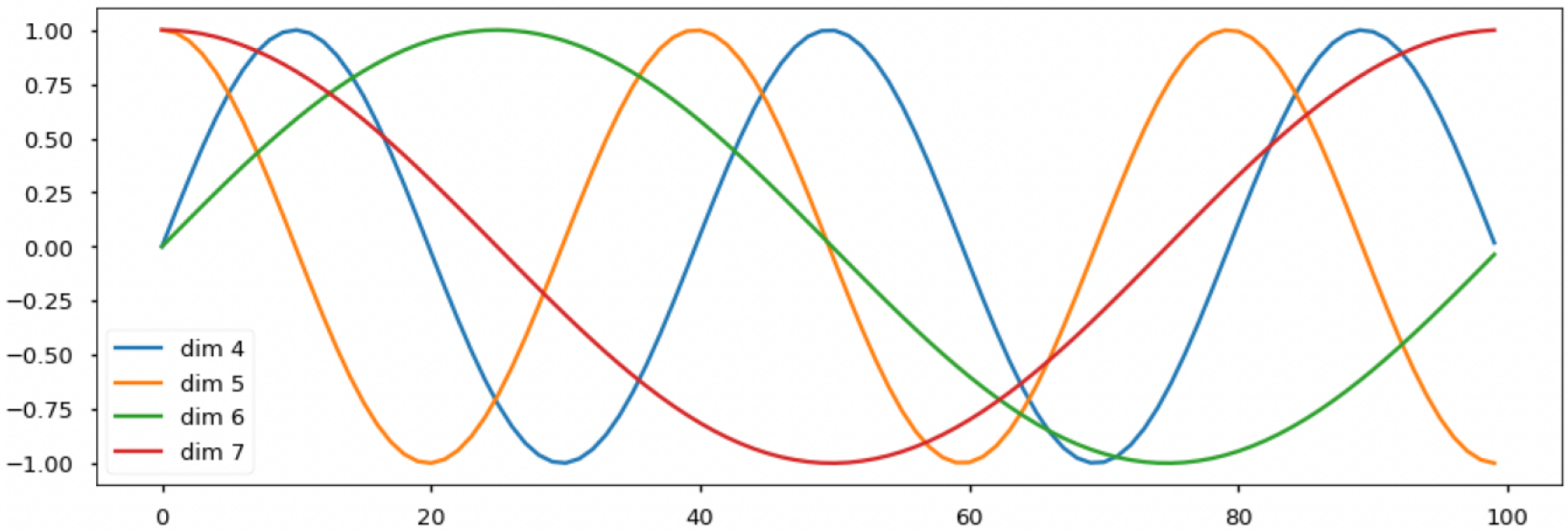
Sinusoidal Positional Encodings

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

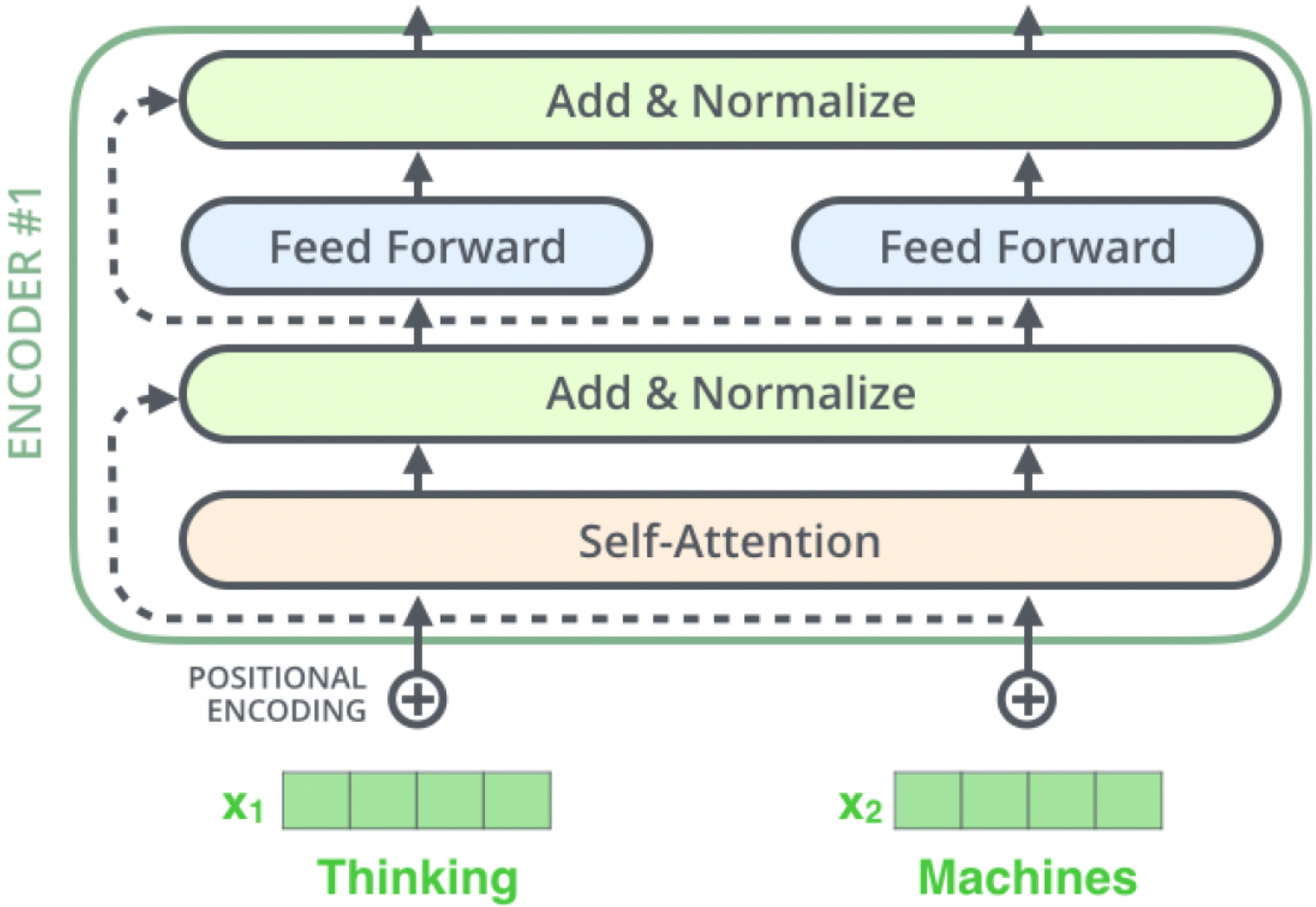


Sinusoidal Positional Encodings

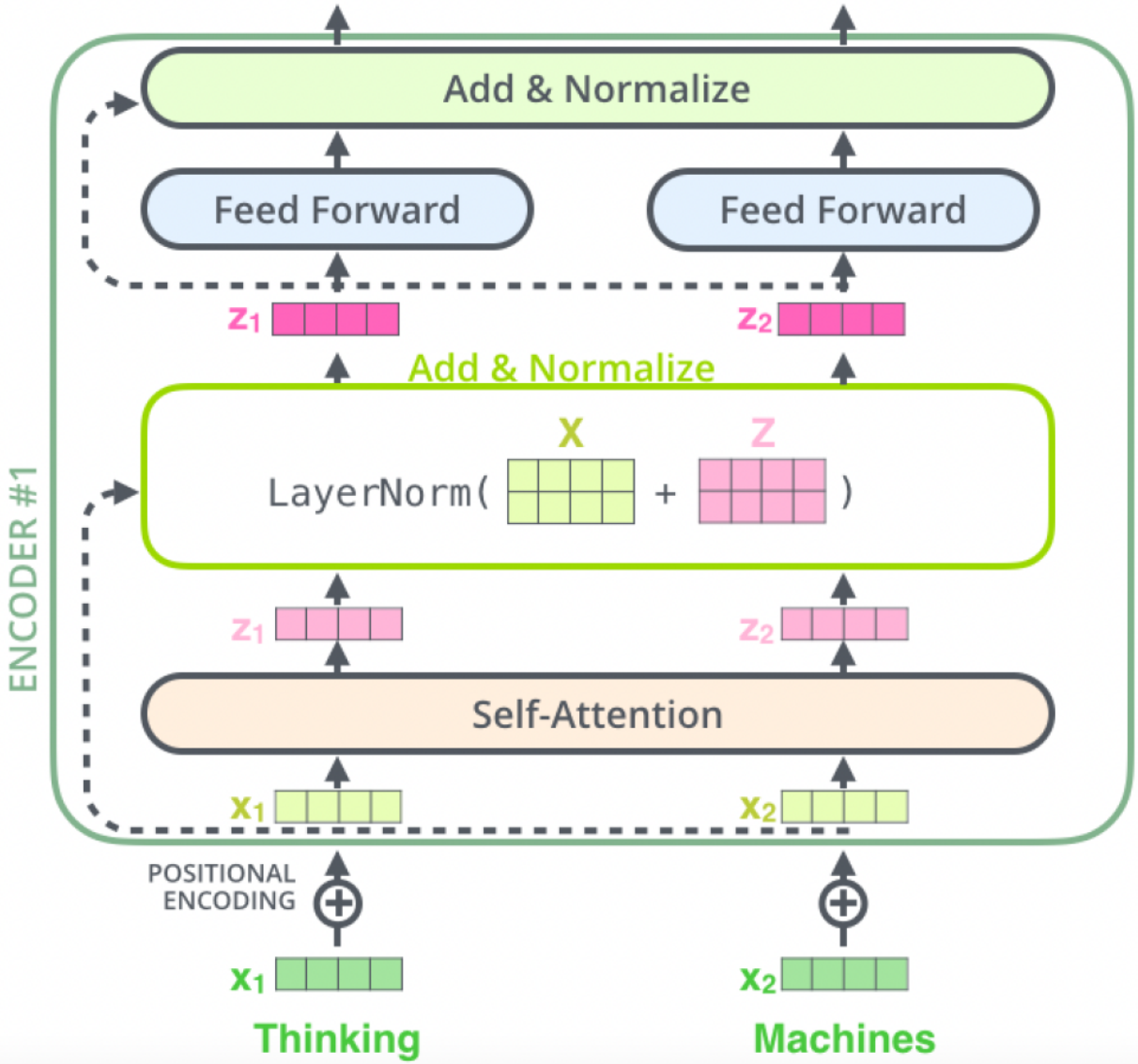
$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d}}\right) \qquad PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d}}\right)$$



Residuals and Layer Normalization



Residuals and Layer Normalization



The Decoder

What about the self-attention blocks in the decoder?

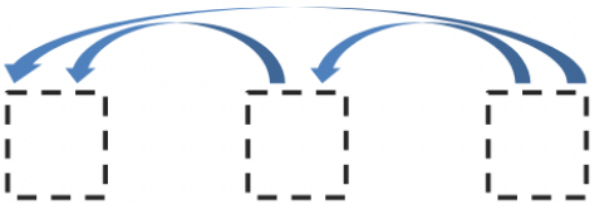
Everything is pretty much the same as in the encoder, with two twists:

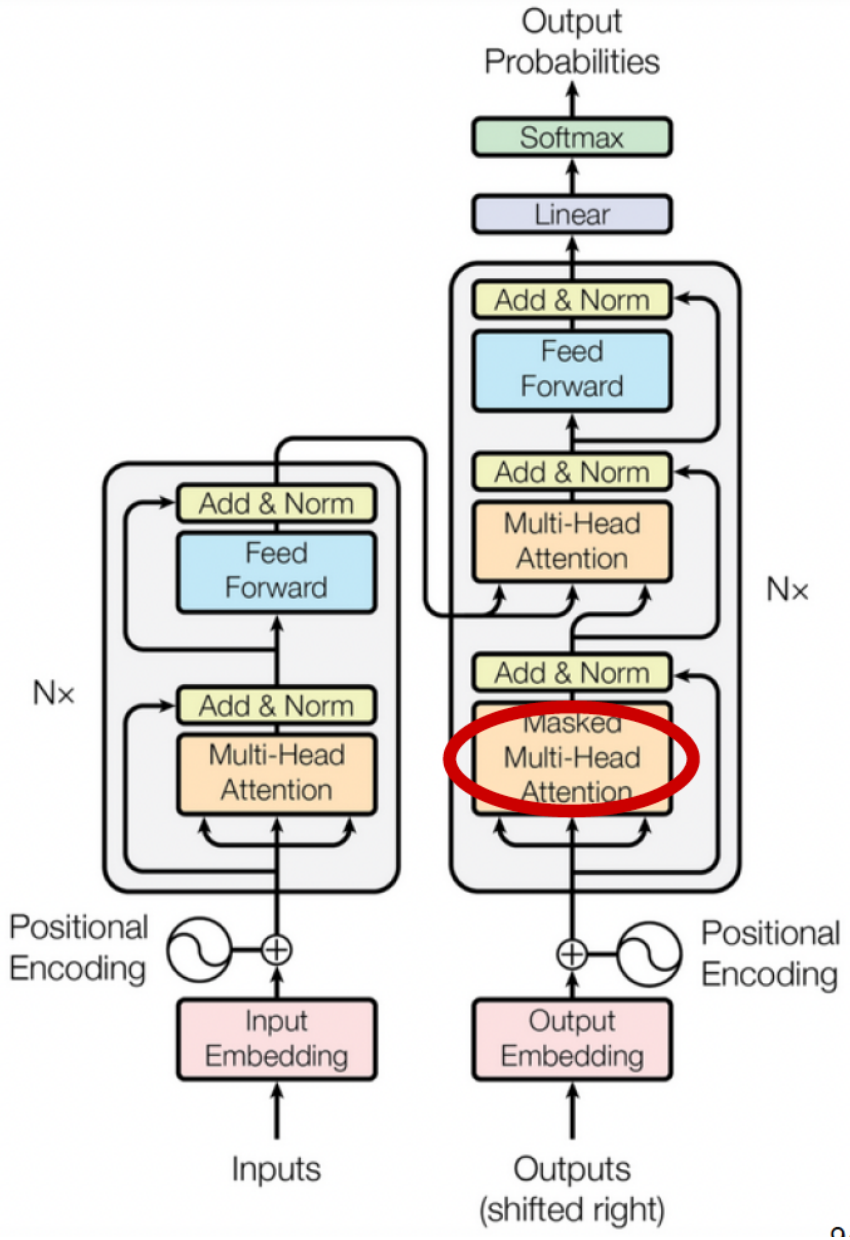
- The decoder cannot see the future! Use "causal" masking
- The decoder should attend to itself (self-attention), but also to the encoder states (contextual attention).

The Decoder

encoder self-attn

decoder self-attn (masked)

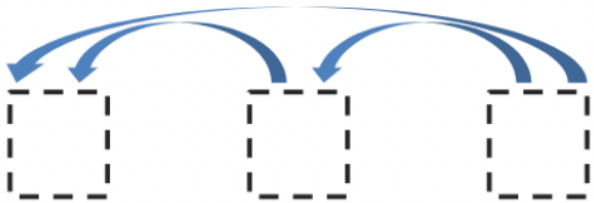




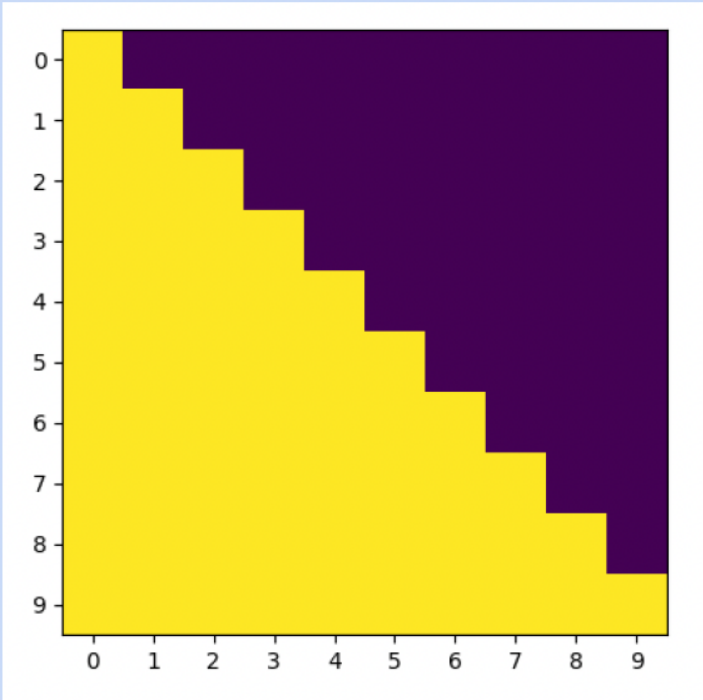
The Decoder

encoder self-attn

decoder self-attn (masked)



- Mask subsequent positions (before softmax)



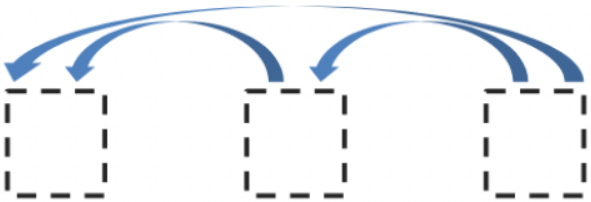
- In PyTorch

```
scores.masked_fill_(~mask, float('-inf'))
```

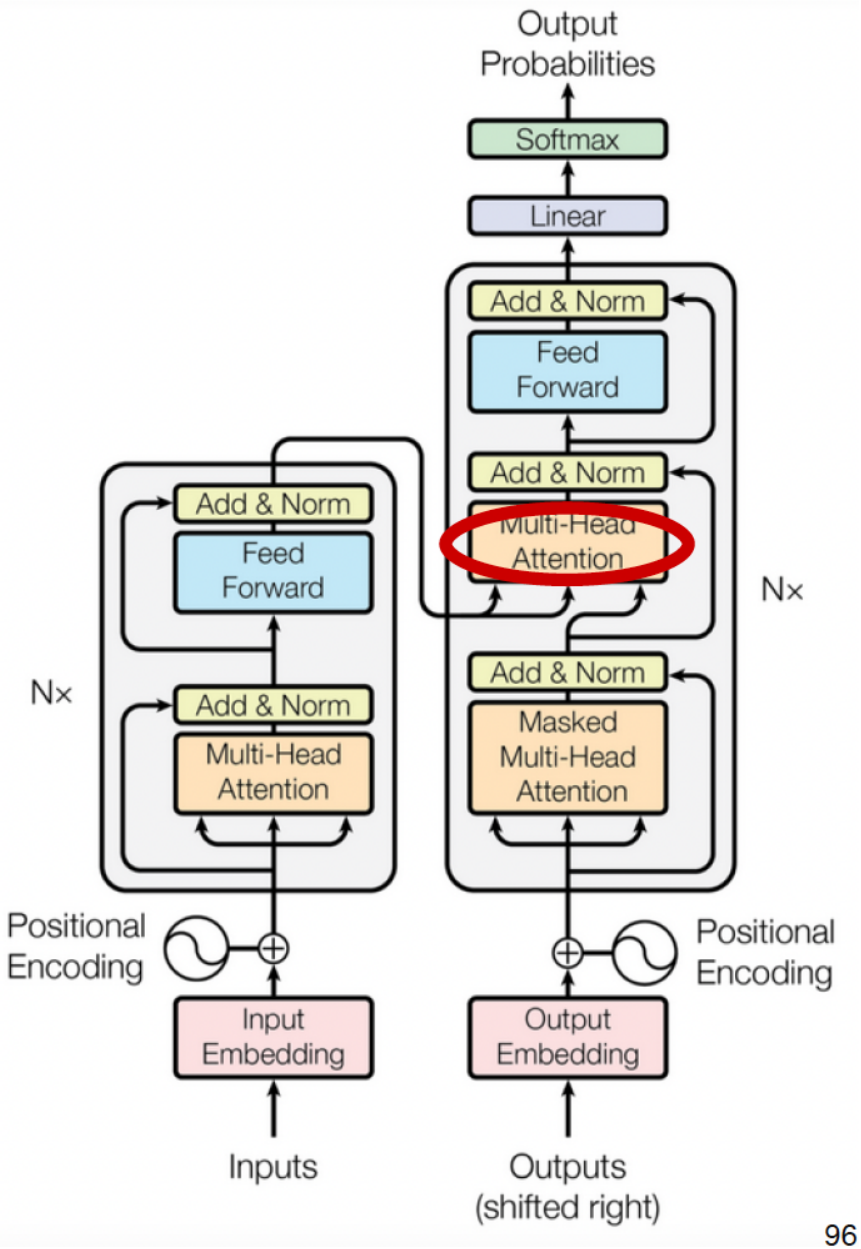
The Decoder

encoder self-attn

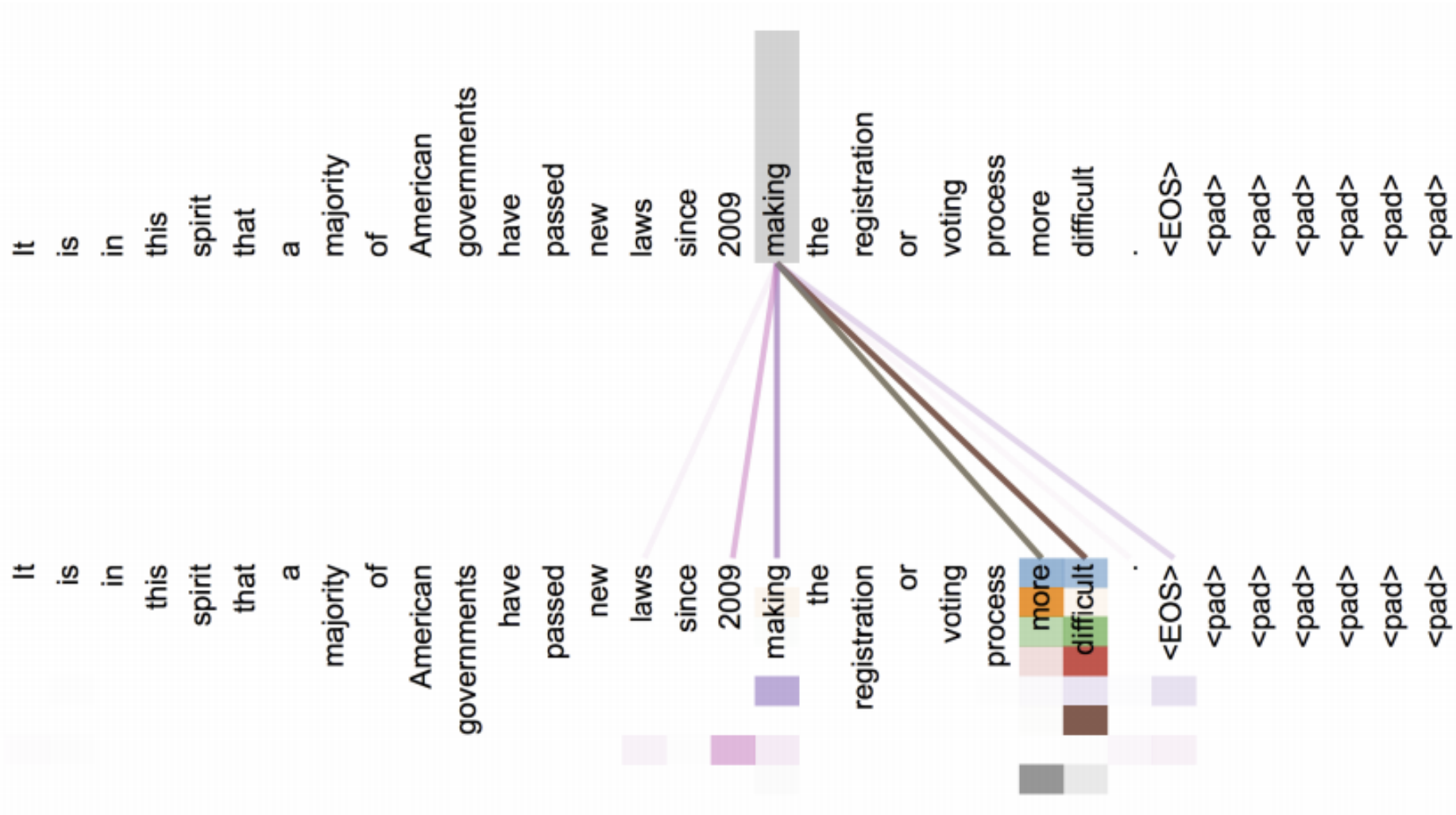
decoder self-attn (masked)



context attention



Attention Visualization Layer 5



Implicit Anaphora Resolution