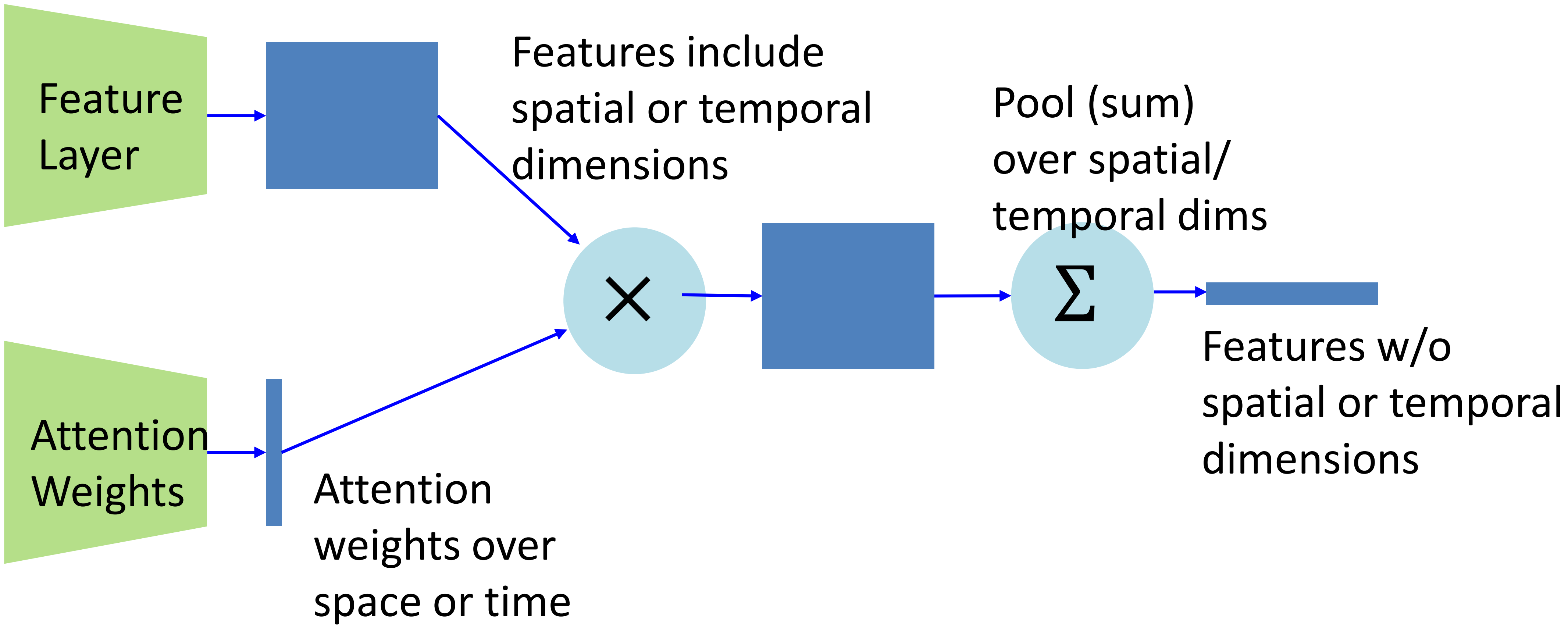
Natural Language Processing
Attention and Transformer

Sudeshna Sarkar

26 Sep 2019

Attention Mechanics

- Typically, soft attention involves a feature layer, a weight predictor, and (optionally) pooling:

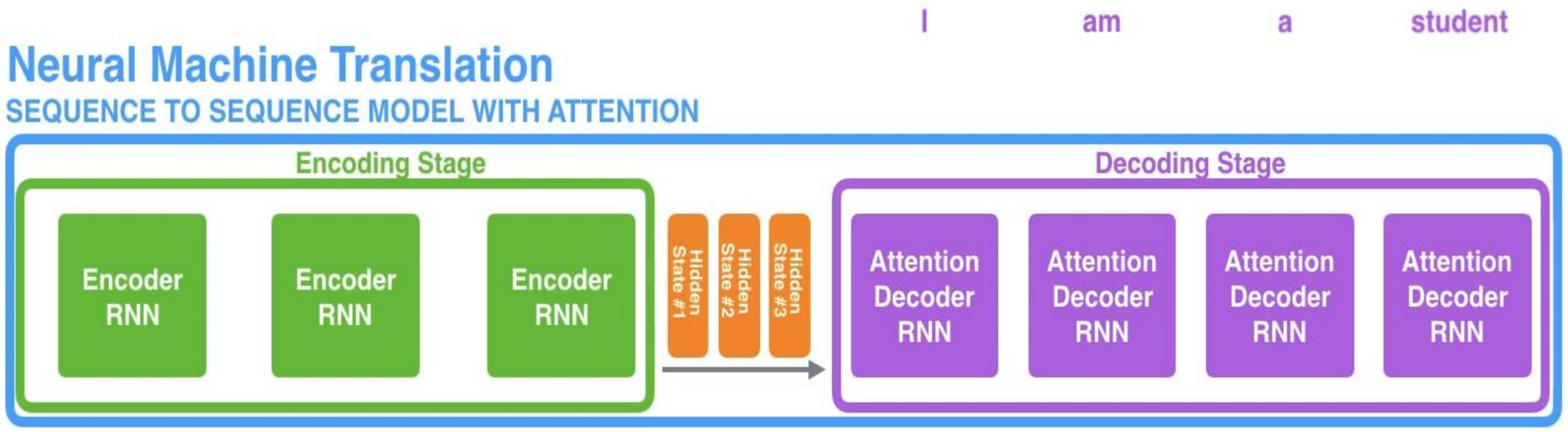

Feature Layer
Features include spatial or temporal dimensions
Pool (sum) over spatial/ temporal dims
×
Σ
Features w/o spatial or temporal dimensions
Attention Weights
Attention weights over space or time

Attention as Explanation

- Deep Network behavior is generally inscrutable.
- Deep Networks do not model data like classical ML models.
- Activations don't have obvious meaning (mostly).

- Attention maps are explanations of net behavior because they identify the influential parts of the input stream.

Sequence to sequence with attention



Attention

This is the encoder-decoder attention.

Attention between encoder and decoder is crucial in NMT

Why not use (self-)attention for the representations?

Score

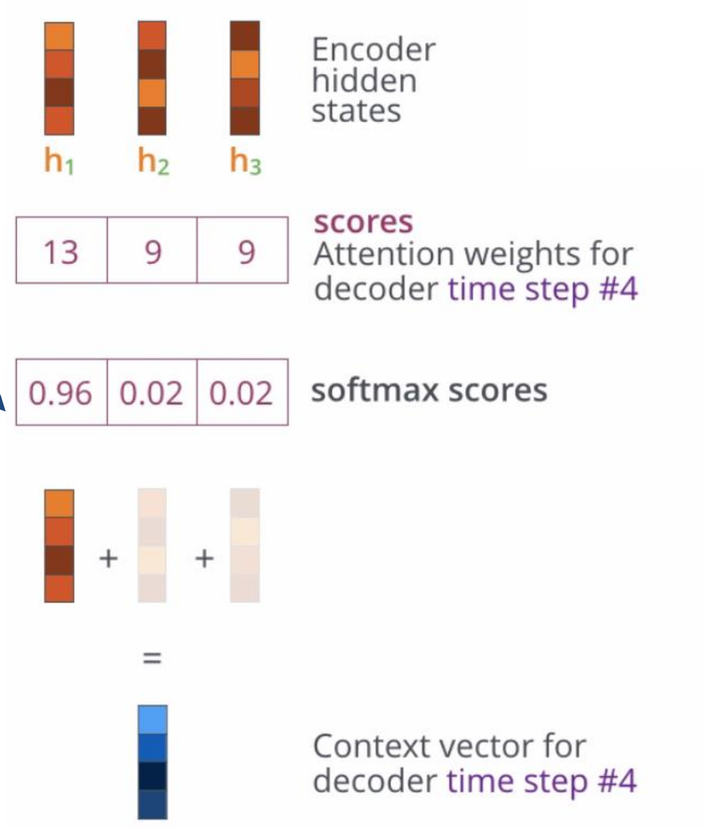
$$\alpha^i(t) = f(h_{dec}(t-1); h_{enc}(i-1))$$

Query (Q)

Key (K)

Value (V)

$$c(t) = \sum_{i=1}^{T} \alpha^i(t) h_{enc}(i-1)$$

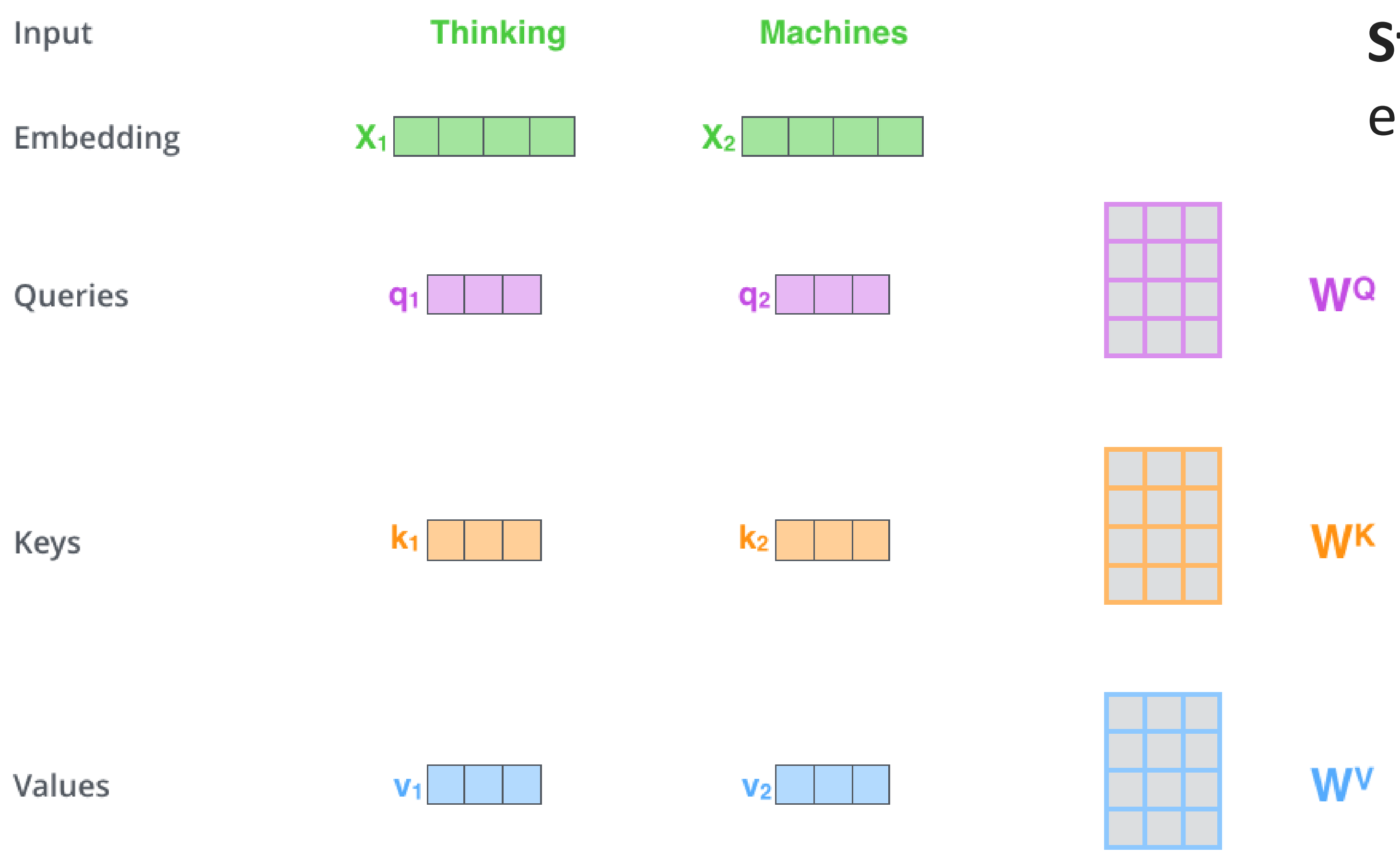


h1
h2
h3
Encoder hidden states
13
9
9
scores
Attention weights for decoder time step #4
0.96
0.02
0.02
softmax scores
+
+
=
Context vector for decoder time step #4

What functions can be used to calculate the score?

1. Additive attention
   - Computes the compatibility function using a feed-forward network with a single hidden layer
2. Dot-product (multiplicative) attention
   - Dot-product attention is faster and more space-efficient in practice, since it can be implemented using highly optimized matrix multiplication code

Dot-product (multiplicative) attention

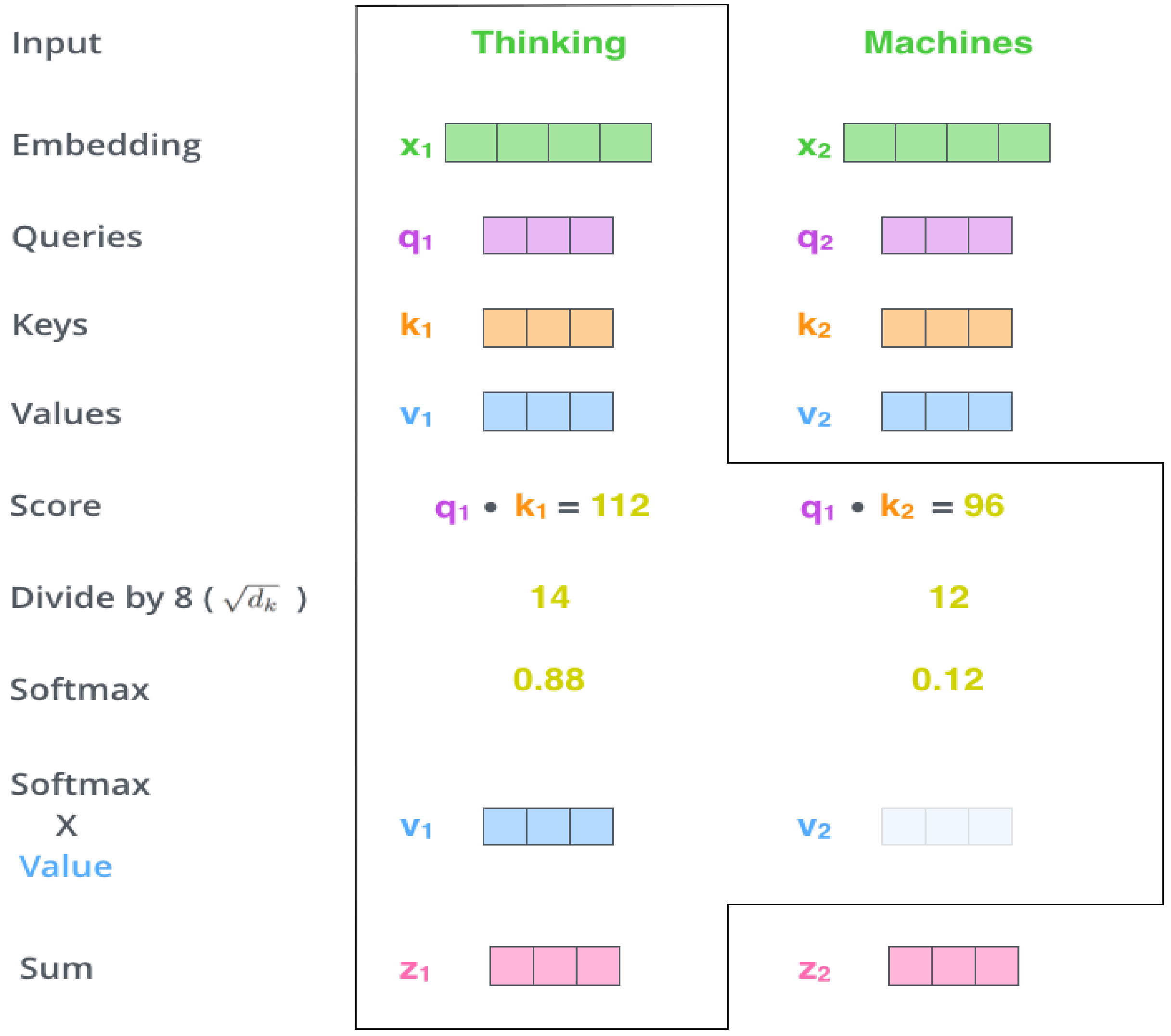


Step 1: Create three vectors from each of the encoder's input vectors.

"query", "key", and "value" vectors

They're abstractions that are useful for calculating and thinking about attention

Dot-product (multiplicative) attention

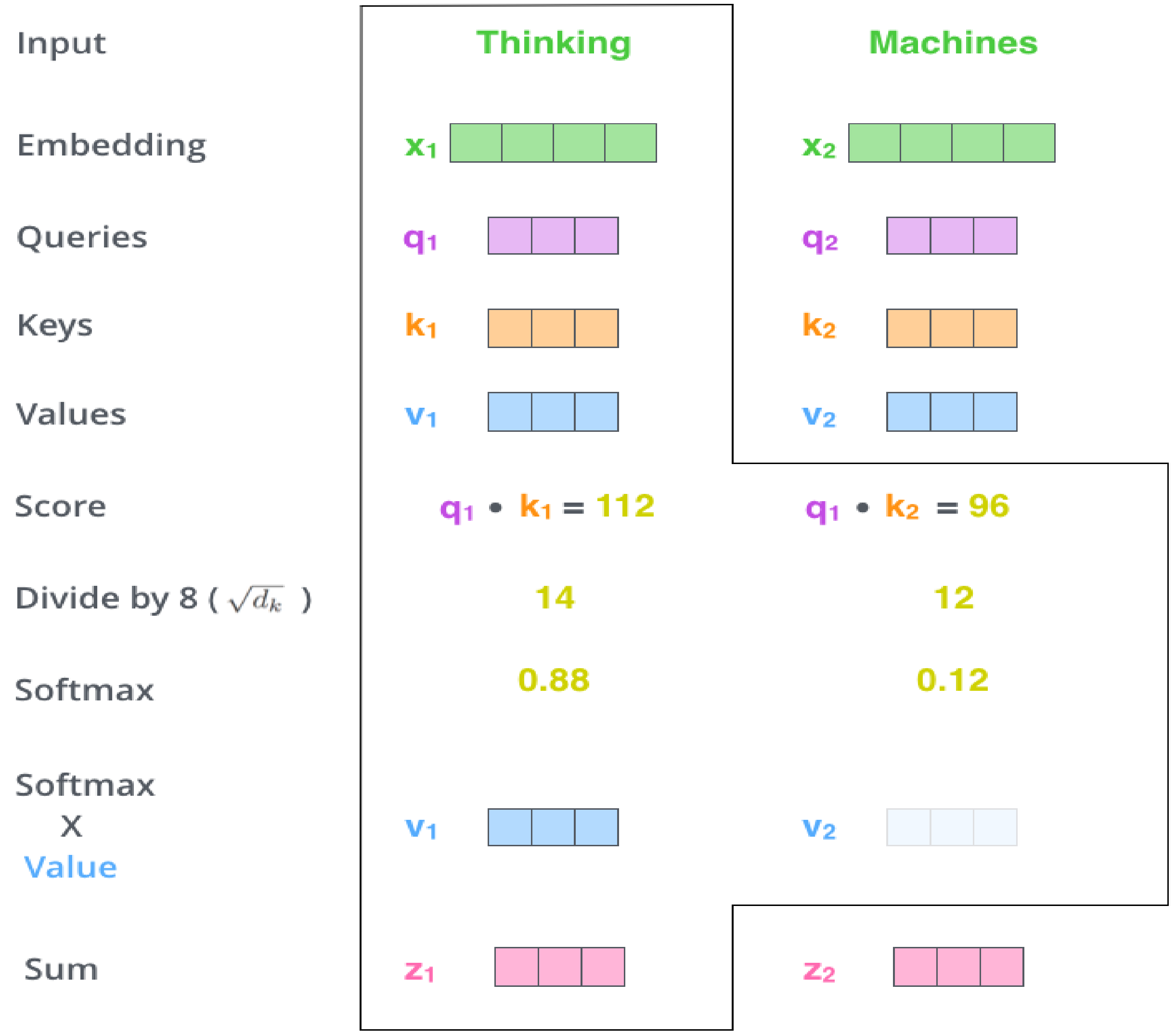


Step 2: Calculate score

Say we're calculating the self-attention for the first word "Thinking".
We need to score each word of the input sentence against this word. The score determines how much focus to place on other parts of the input sentence as we encode a word at a certain position.

The score is calculated by taking the dot product of the query vector with the key vector of the respective word we're scoring.
So if we're processing the self-attention for the word in position #1, the first score would be the dot product of q1 and k1. The second score would be the dot product of q1 and k2.

Dot-product (multiplicative) attention



Step 2: Calculate score

Step 3: divide the scores by 8 (the square root of the dimension of the key vectors used in the paper – 64. This leads to having more stable gradients.

Step 4: pass the result through a softmax operation.

This softmax score determines how much how much each word will be expressed at this position. Clearly the word at this position will have the highest softmax score, but sometimes it's useful to attend to another word that is relevant to the current word.

Scaling Factor

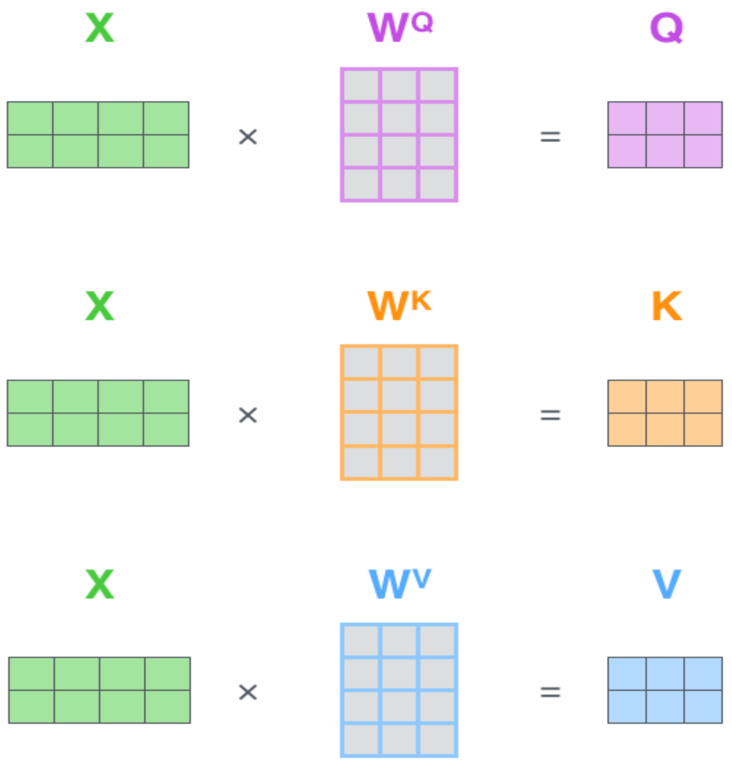
Why divide by square root of dk ?

Problem: Additive attention outperforms dot product attention without scaling for larger values of d_k.

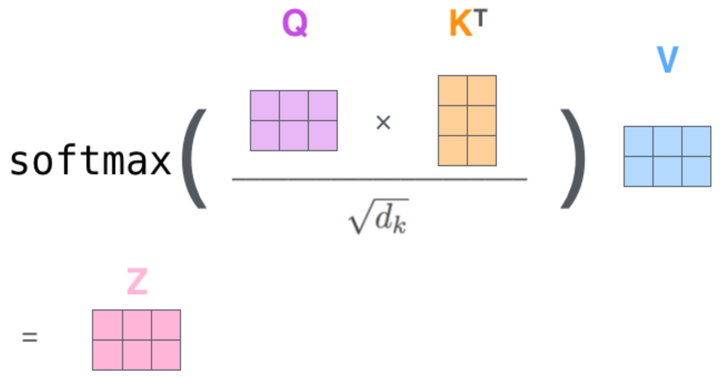
Cause: Dot products grow large in magnitude, pushing the softmax function into regions where it has extremely small gradients.

Solution: To counteract this effect, scale the dot products by $1/\sqrt{d_k}$

Dot-product (multiplicative) attention



Step 5: multiply each value vector by the softmax score

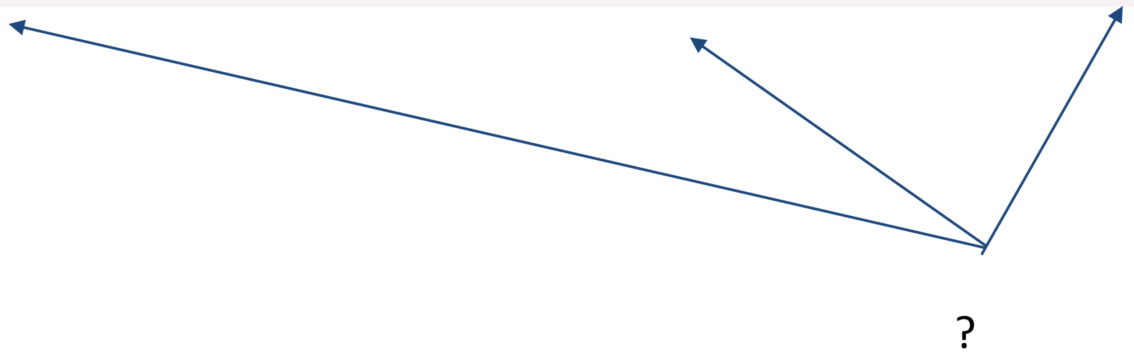


$$\text{softmax}\left(\frac{Q \times K^T}{\sqrt{d_k}}\right) V = Z$$

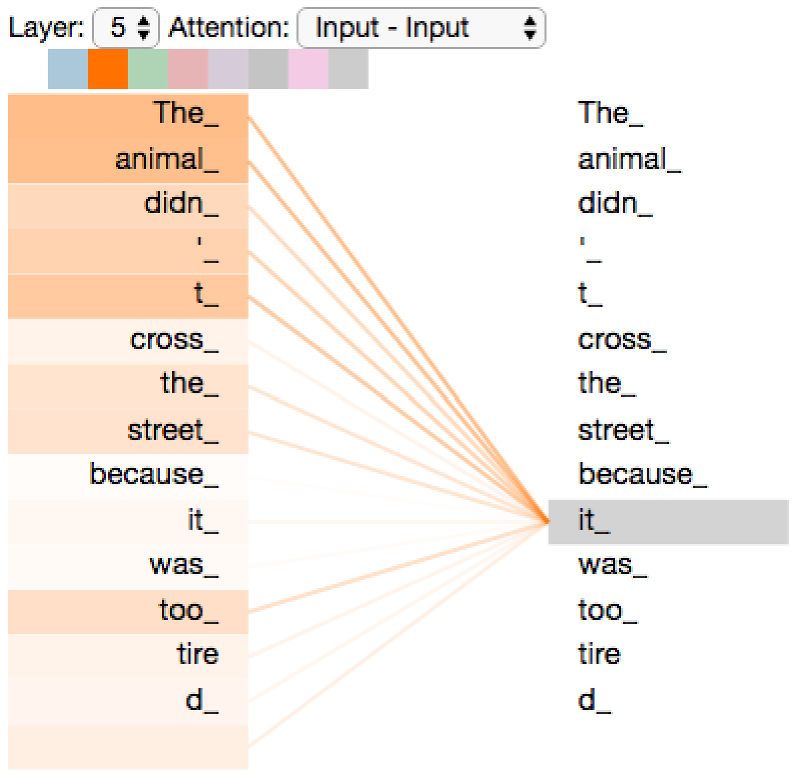
Step 6: Sum up the weighted value vectors.
This produces the output of the self-attention layer at this position

Self Attention

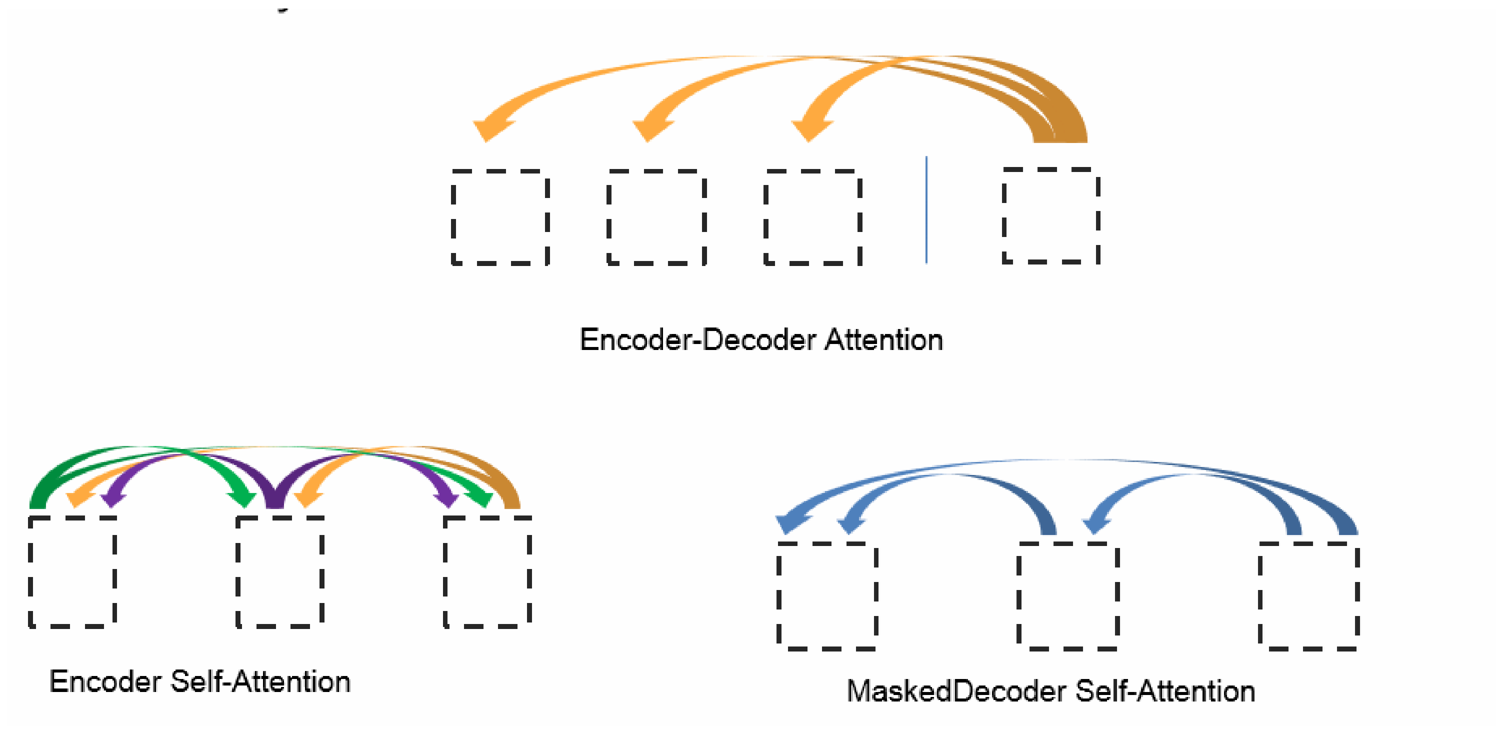
"`The animal didn't cross the street because it was too tired`"



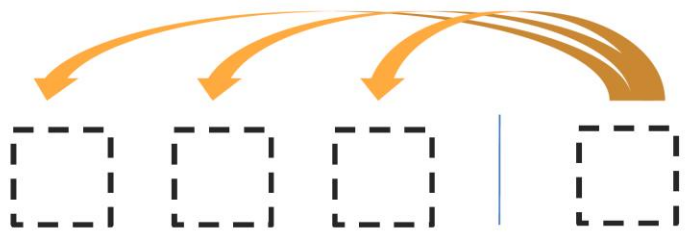
Self Attention



Three ways of Attention

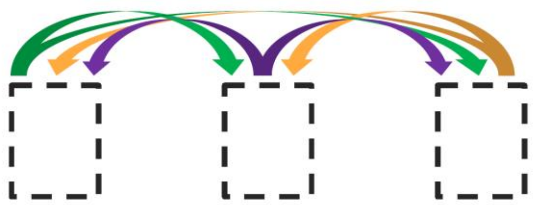


Attention in Transformer Networks

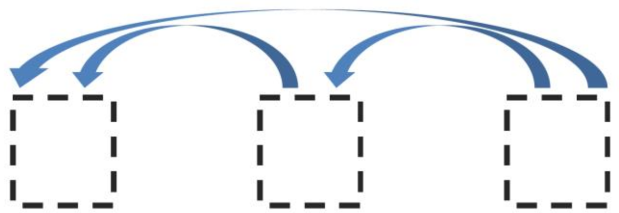


Encoder-Decoder Attention

We saw this in Bahdanau and Luong models



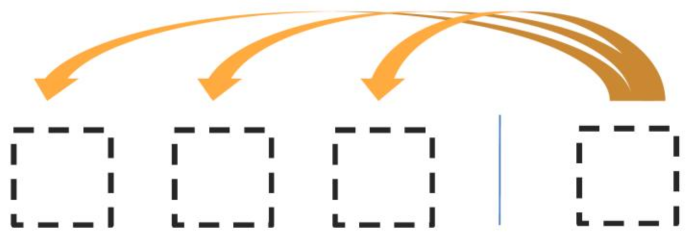
Encoder Self-Attention



MaskedDecoder Self-Attention

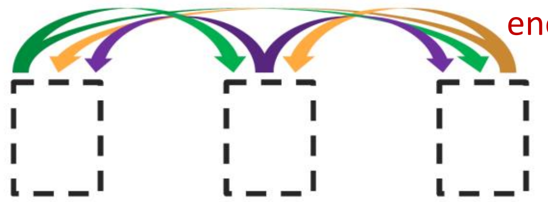
image from Lukas Kaiser, Stanford NLP seminar

Attention in Transformer Networks

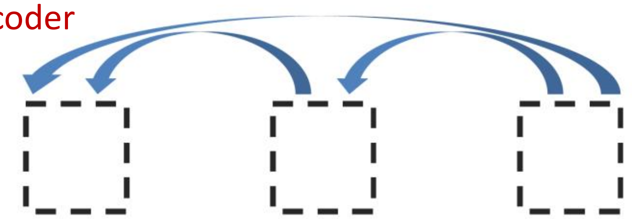


Encoder-Decoder Attention

Replaces word recurrence in encoder and decoder



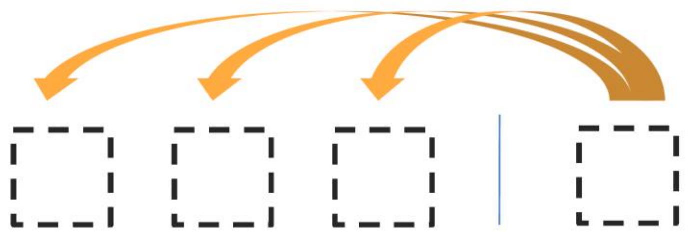
Encoder Self-Attention



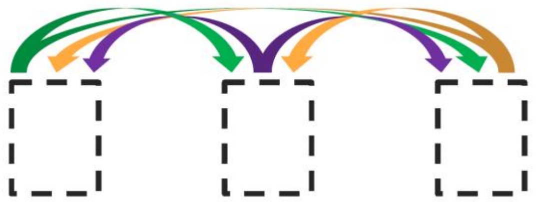
MaskedDecoder Self-Attention

image from Lukas Kaiser, Stanford NLP seminar

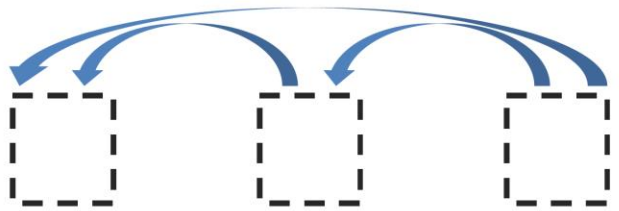
Attention in Transformer Networks



Encoder-Decoder Attention



Encoder Self-Attention



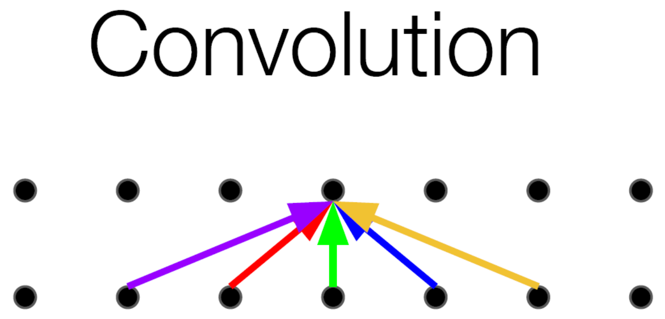
MaskedDecoder Self-Attention

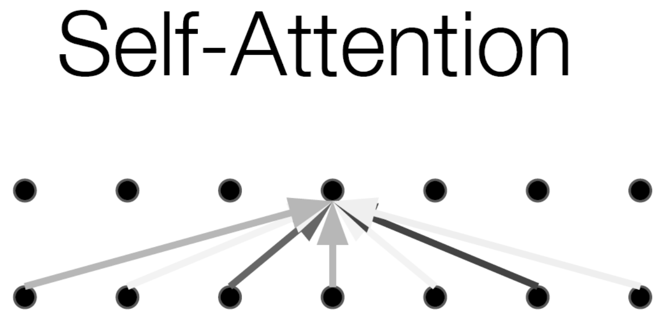
Masking limits attention to earlier units:
y_i depends only on y_j for $j < i$.

image from Lukas Kaiser, Stanford NLP seminar

Self-Attention

Information flows from within the same sub-network (either encoder or decoder). Convolution applies fixed transform weights. Self-attention applies variable weights (but typically not transformations):





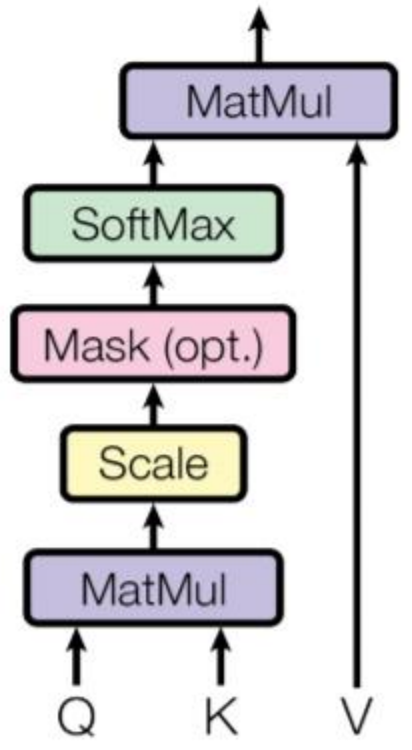
Encoder Self Attention

- Each position in the encoder can attend to all positions in the previous layer of the encoder.
- All of the keys, values, and queries come from the same place, in this case, the output of the previous layer in the encoder.

Decoder Self Attention

- Similarly, self-attention layers in the decoder allow each position in the decoder to attend to all positions in the decoder up to and including that position.
- To preserve the auto-regressive property, mask out (setting to $-\infty$) all values in the input of the softmax which correspond to illegal connections.

Scaled Dot-Product Attention



Attention-only Translation Models

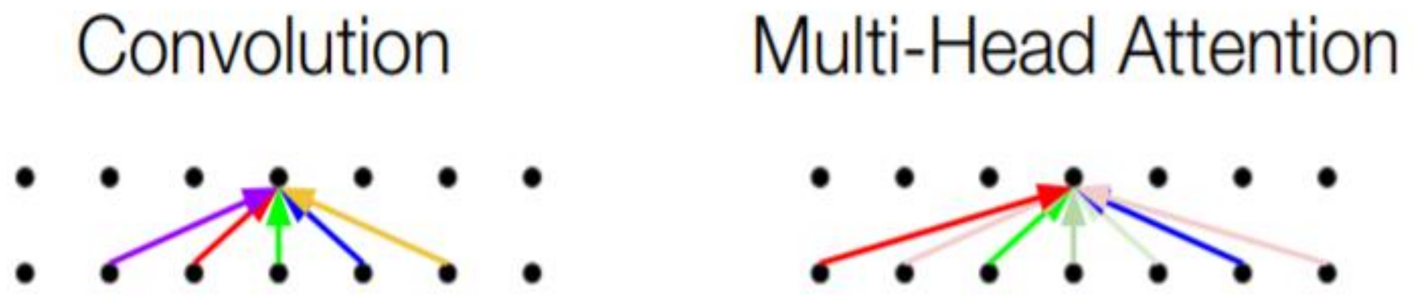
- Problems with recurrent networks:
- Sequential training and inference: time grows in proportion to sentence length. Hard to parallelize.
- Long-range dependencies have to be remembered across many single time steps.
- Tricky to learn hierarchical structures (“car”, “blue car”, “into the blue car”…)

Alternative:

- Convolution – but has other limitations.

Multi-head Attention

- Multiple attention layers (heads) in parallel (shown by different colors)
- Each head uses different linear transformations.
- Different heads can learn different relationships.



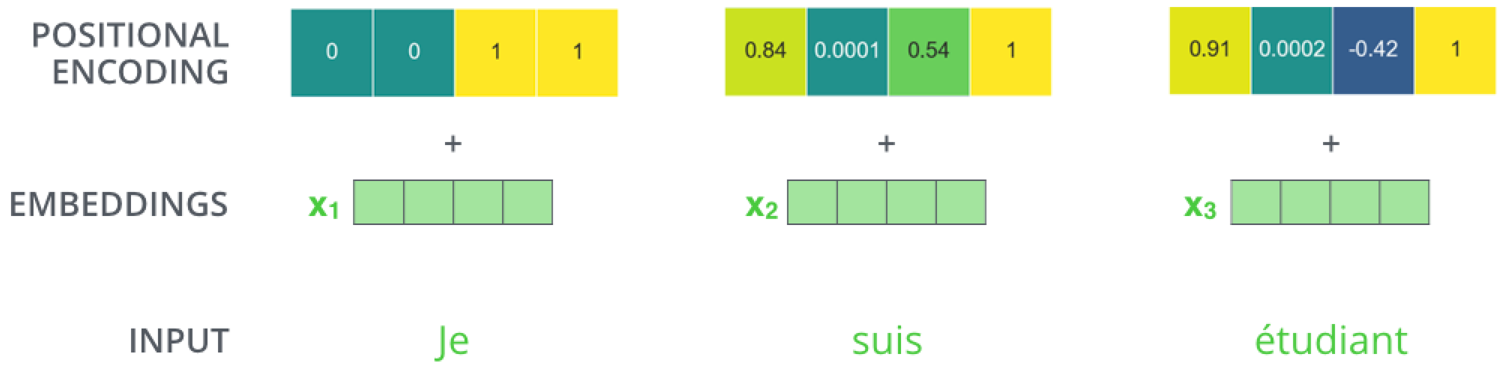
Self-Attention “Transformers”

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

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

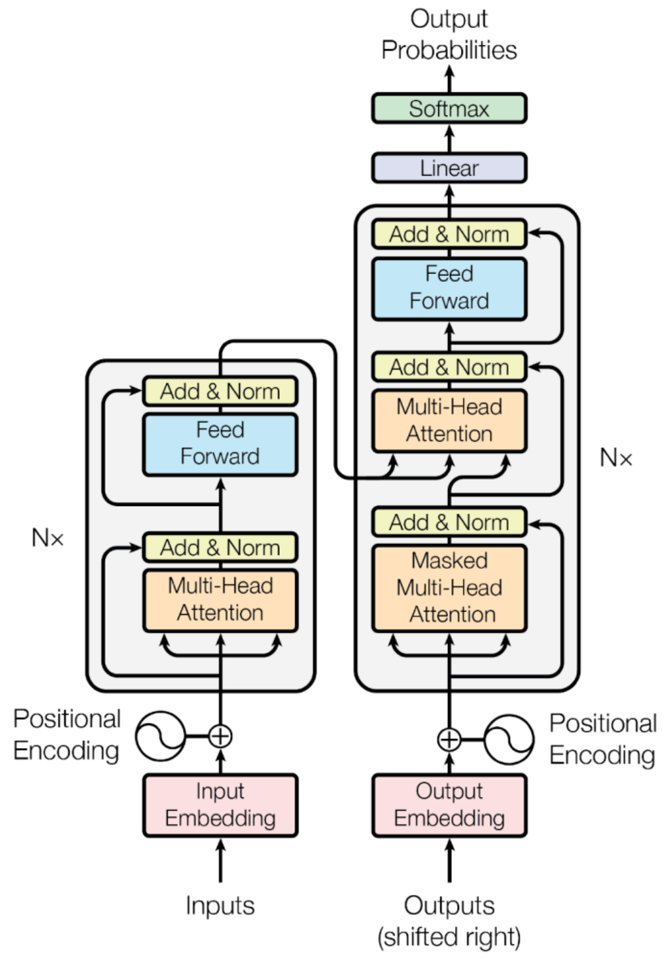
Positional Encoding

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$



The Transformer

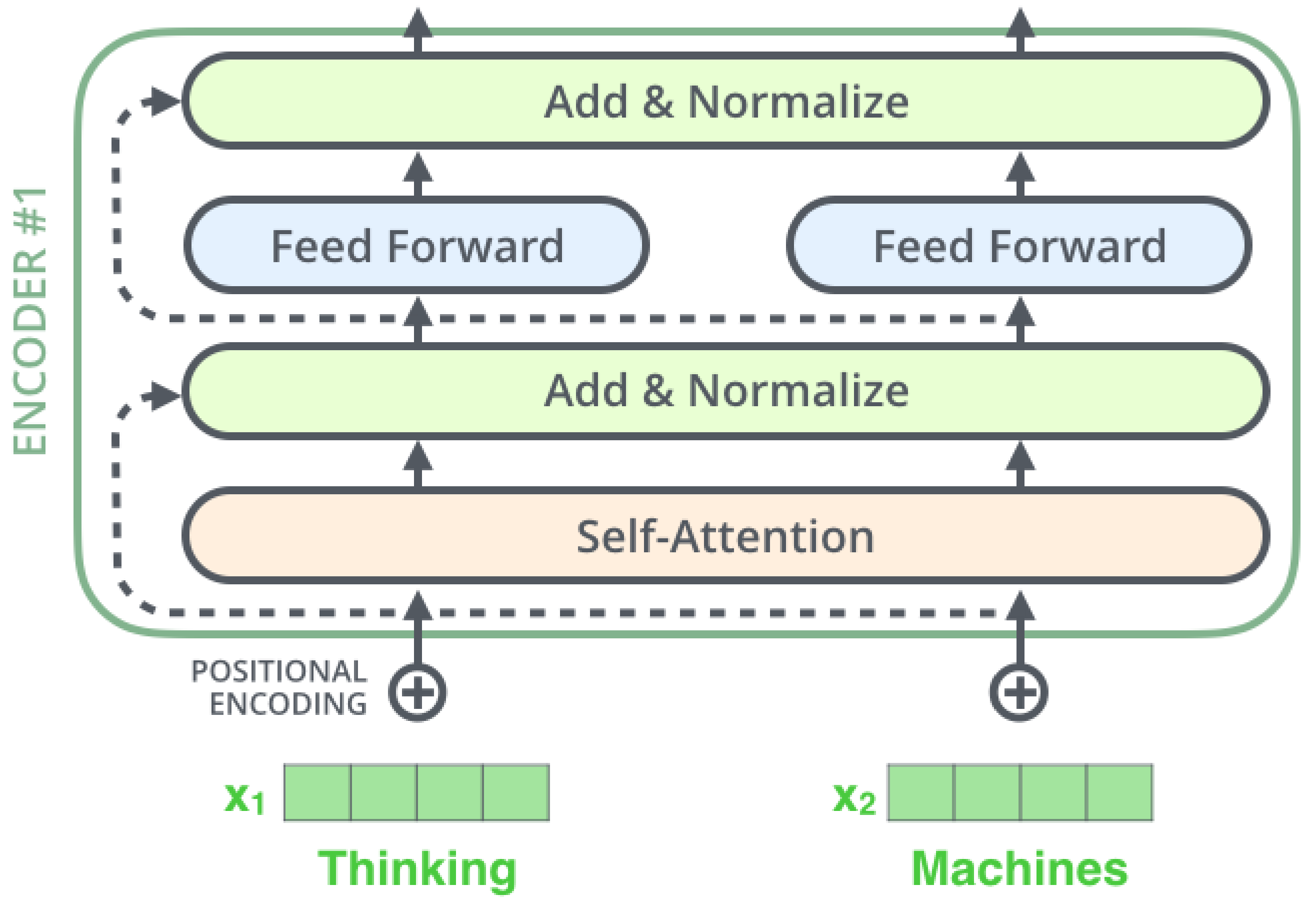
- Basic unit shown at right.
- In experiments, stacked with N=6.
- Output words fed back as input, shifted right. Can use beam search as before.
- Inputs and outputs are embedded in vector spaces of fixed dimension.
- Positional encoding: when words are combined through attention, their location is lost. Positional encoding adds it back.

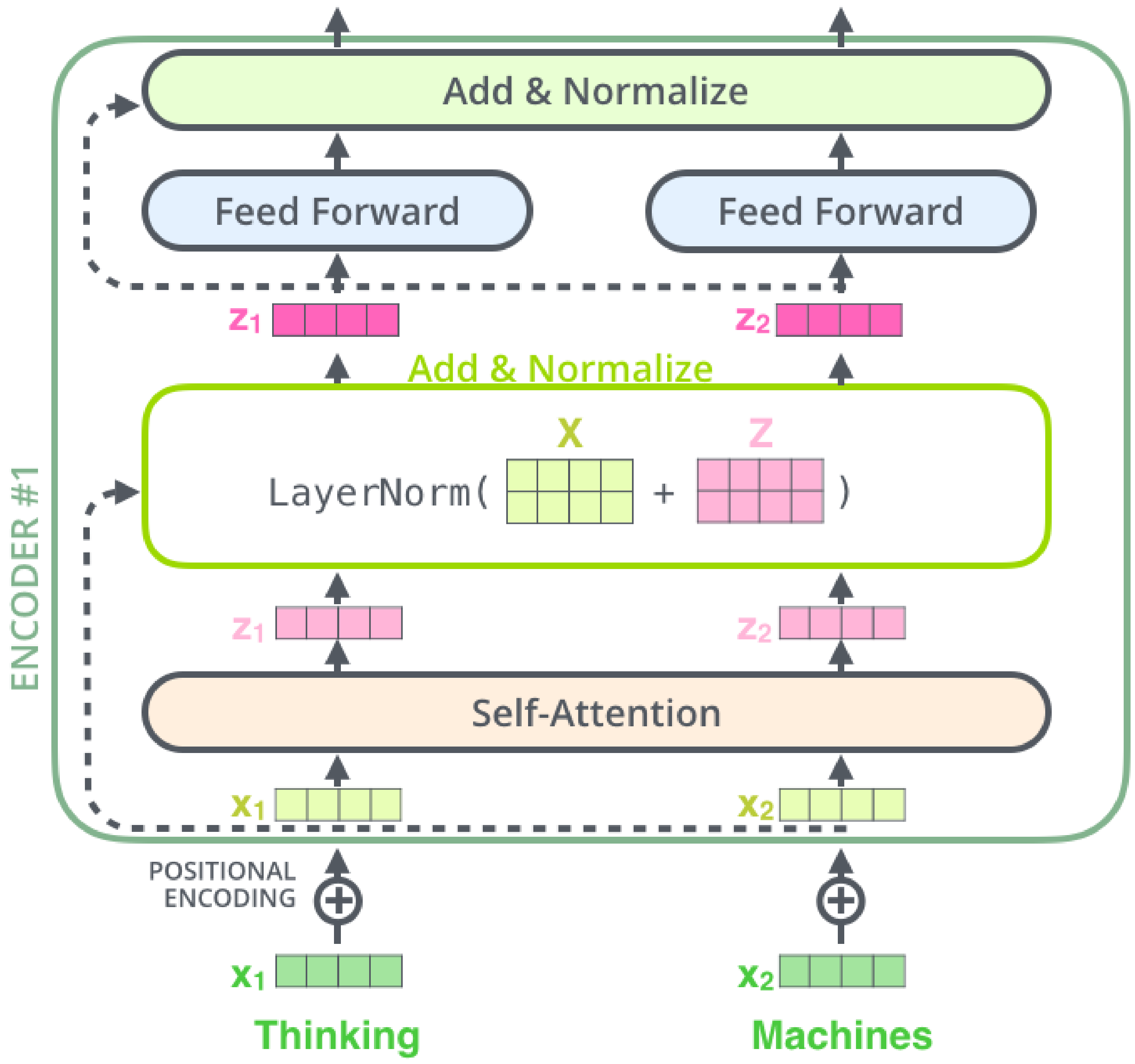


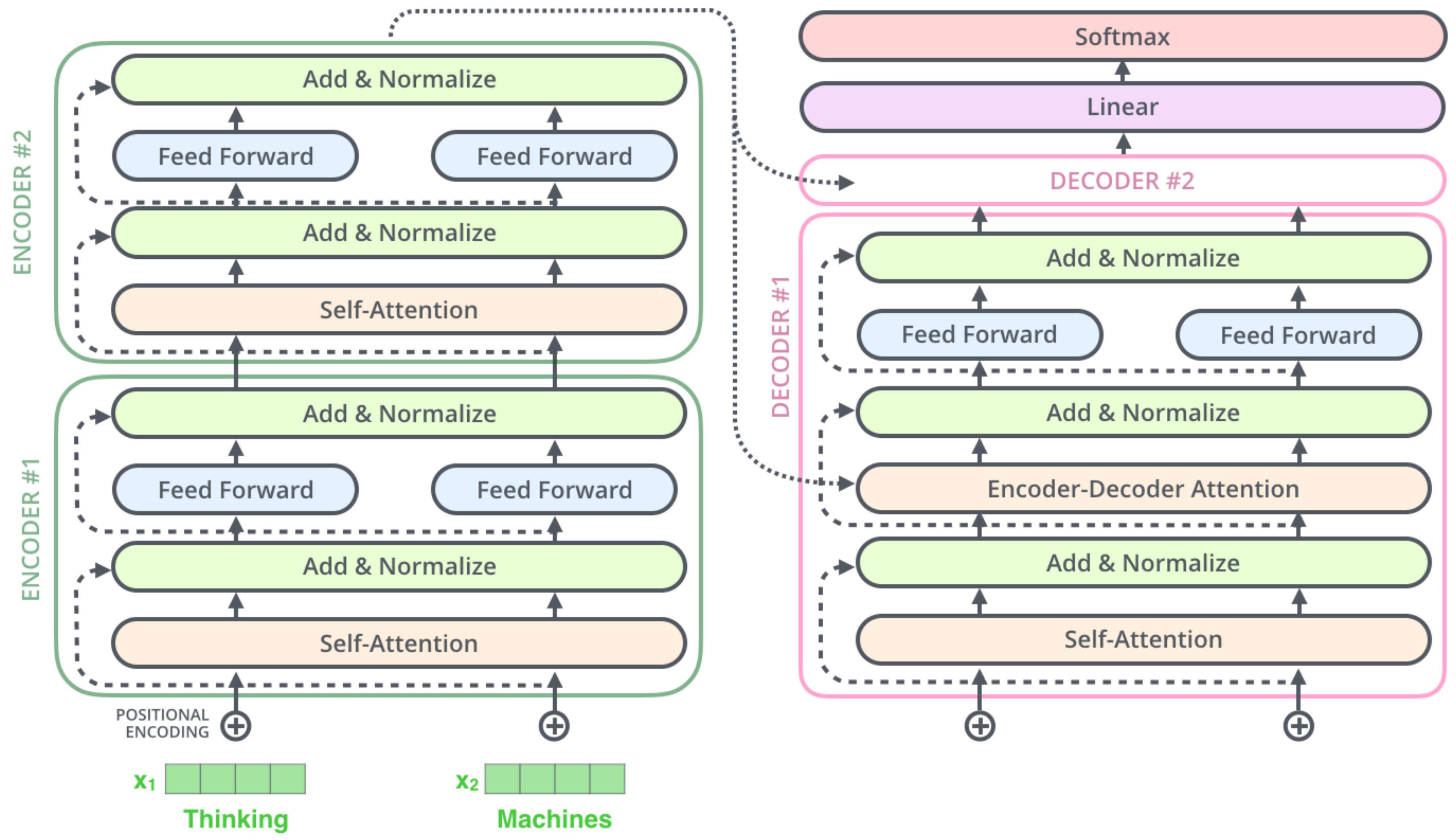
- http://jalammar.github.io/illustrated-transformer/

The Residuals

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




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


Softmax
Linear
DECODER #2
DECODER #1
Add & Normalize
Feed Forward
Feed Forward
Add & Normalize
Encoder-Decoder Attention
Add & Normalize
Self-Attention
ENCODER #2
Add & Normalize
Feed Forward
Feed Forward
Add & Normalize
Self-Attention
ENCODER #1
Add & Normalize
Feed Forward
Feed Forward
Add & Normalize
Self-Attention
POSITIONAL ENCODING
x1
Thinking
x2
Machines

Attention Implementation

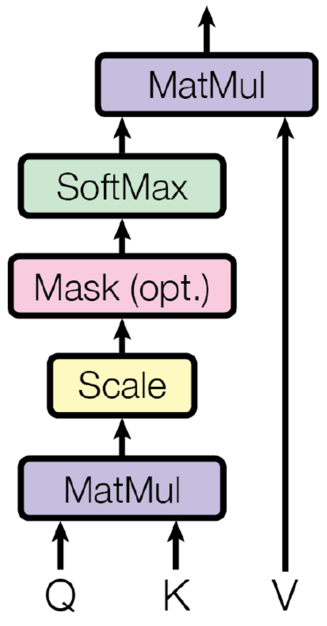
- Attention is modeled as a key-value store:
  Q = query vector
  K = key
  V = value

Encoder-decoder layer: the queries come from the previous decoder layer, and the memory keys and values come from the output of the encoder. (Similar to Bahdanau).

Self-attention layer: all of the keys, values and queries come from the output of the previous layer in the encoder.

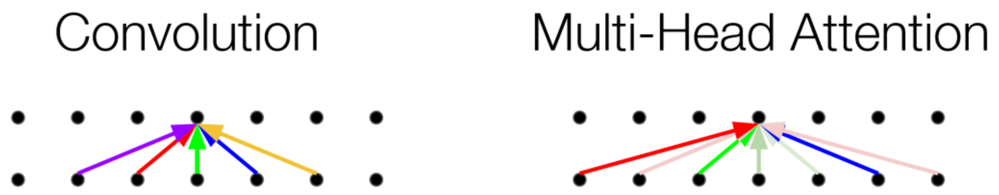
$$\mathrm{Attention}(Q, K, V) = \mathrm{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

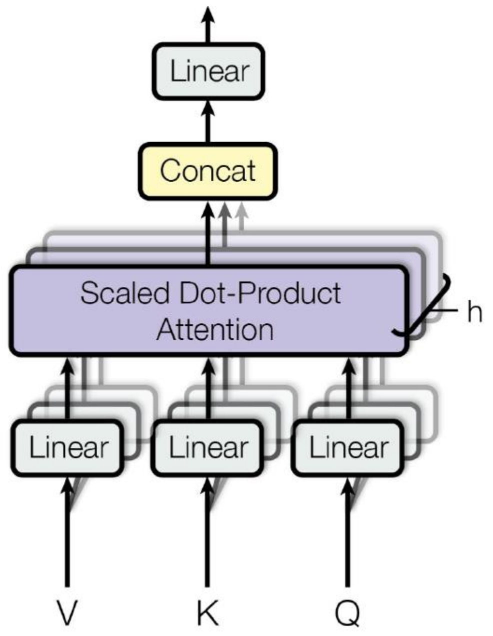
Scaled Dot-Product Attention



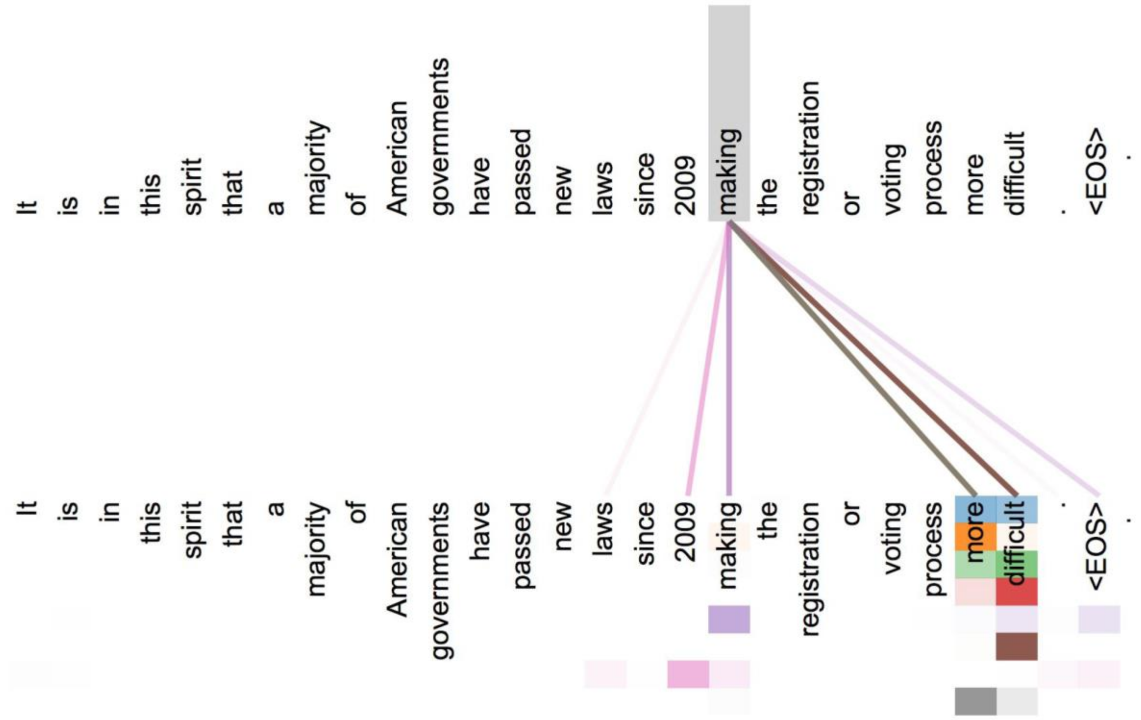
Multi-Headed Attention

- Simple attention blends the results of all the attended-to inputs. It doesn't allow a per-input transformation, as convolution does.
- The solution is to use "multi-headed attention":



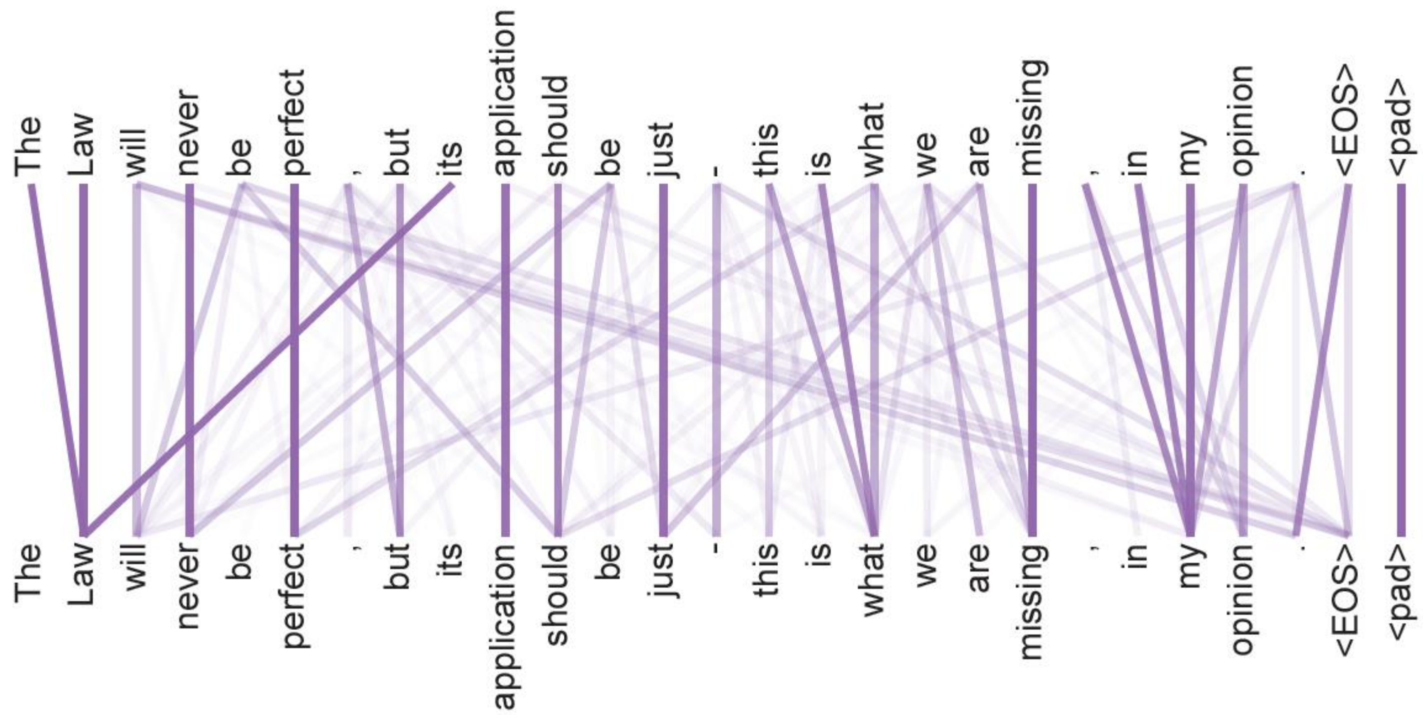


Multi-Headed Attention



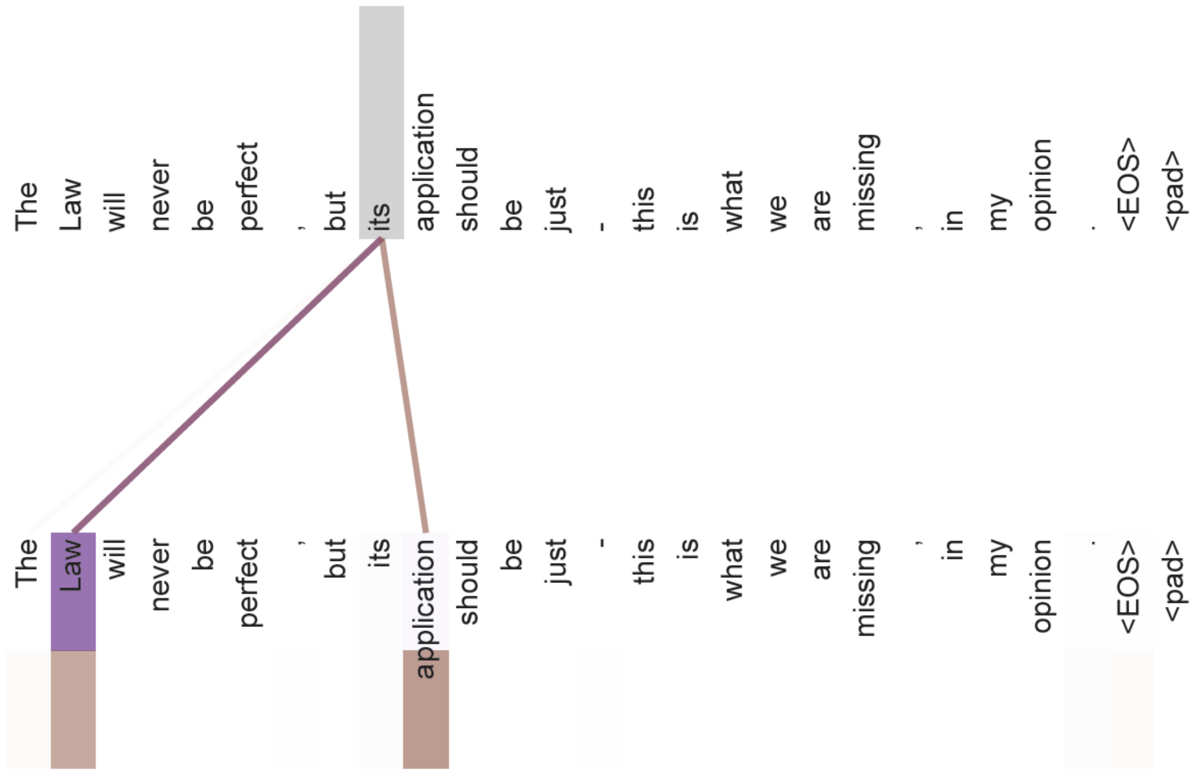
It is in this spirit that a majority of American governments have passed new laws since 2009 making the registration or voting process more difficult . <EOS>
It is in this spirit that a majority of American governments have passed new laws since 2009 making the registration or voting process more difficult . <EOS>

Multi-Headed Attention



Anaphora (pronoun or article) resolution

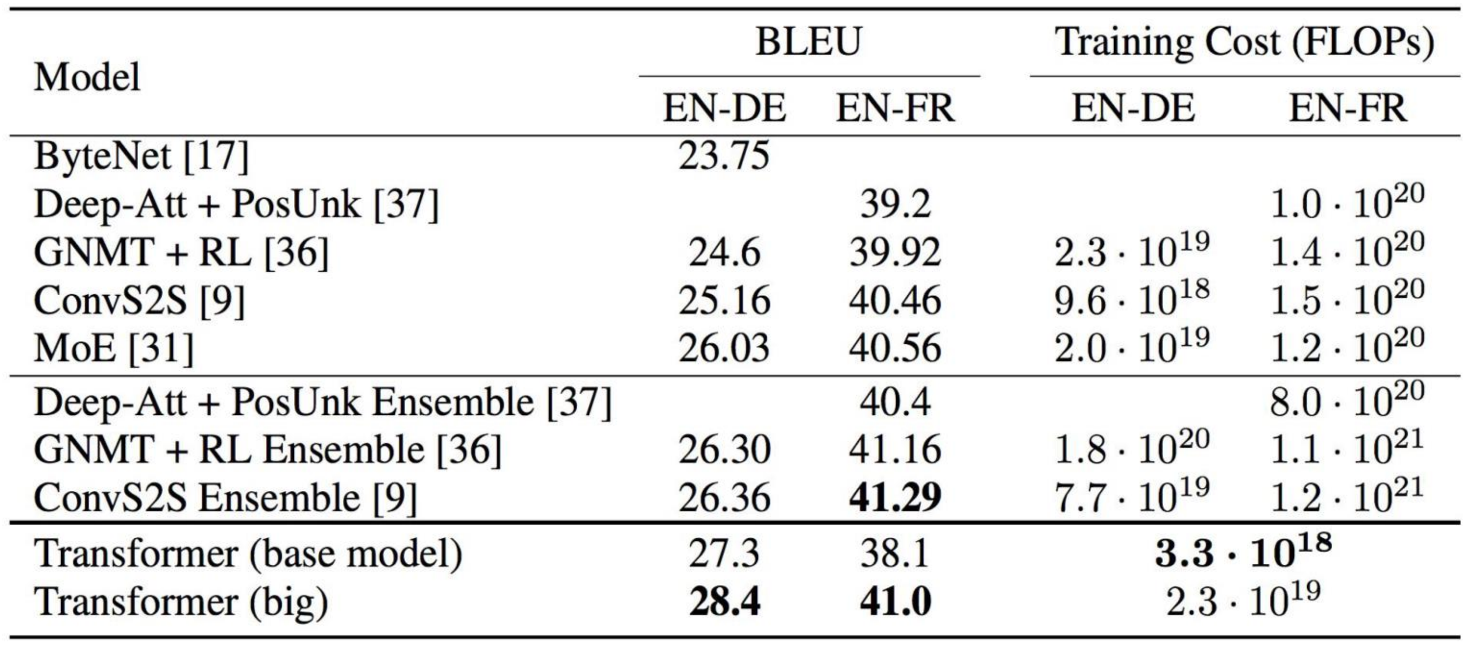
Multi-Headed Attention



Anaphora (pronoun or article) resolution

Transformer Results

Machine Translation Results: WMT-14



Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [17]	23.75			
Deep-Att + PosUnk [37]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [36]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [31]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [37]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [36]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	$\mathbf{3.3 \cdot 10^{18}}$	
Transformer (big)	**28.4**	**41.0**	$2.3 \cdot 10^{19}$	

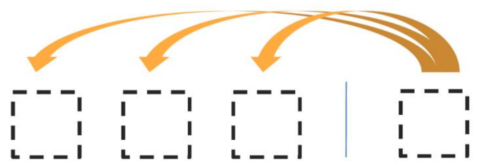
English-to-English Translation ?!

Yes, it does make sense. a.k.a. summarization.

Liu et al, “GENERATING WIKIPEDIA BY SUMMARIZING LONG SEQUENCES” arXiv 2018

M = input length, N = output length

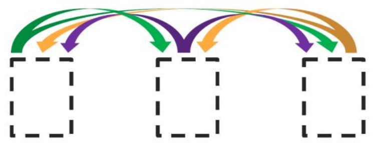
Summarization: M >> N



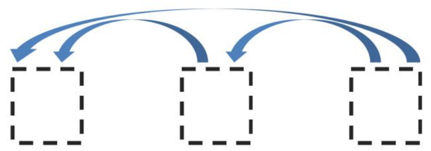
$O(MN)$

Encoder-Decoder Attention

$O(M^2)$



Encoder Self-Attention



$O(N^2)$

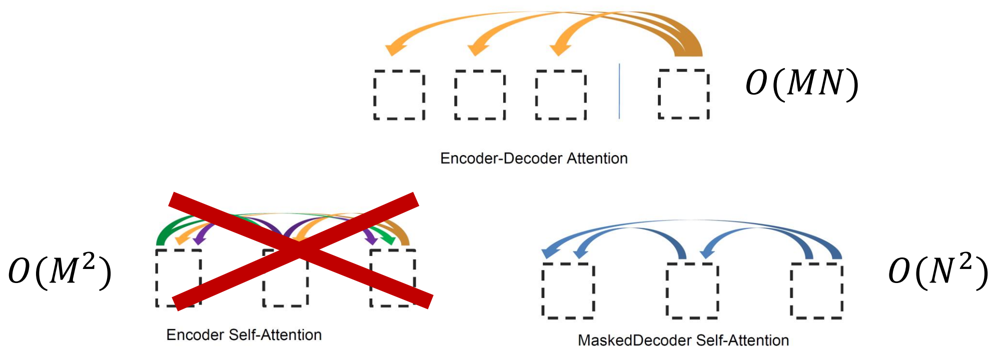
MaskedDecoder Self-Attention

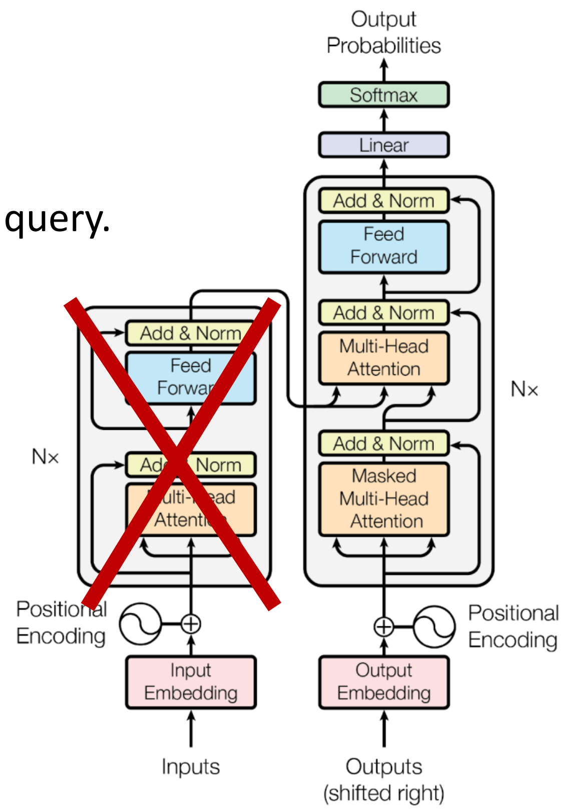
Large-scale Summarization (Wikipedia)

Like translation, but we completely remove the encoder.

Source data (large!):

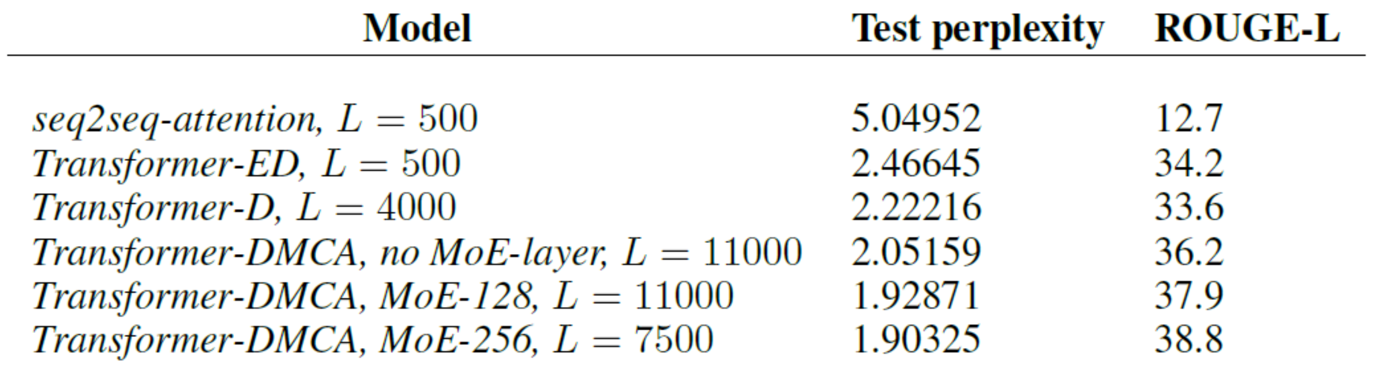
- The references for a Wikipedia article.
- Web search using article section titles, ~ 10 web pages per query.





Large-scale Summarization

Results:



Model	Test perplexity	ROUGE-L
seq2seq-attention, $L = 500$	5.04952	12.7
Transformer-ED, $L = 500$	2.46645	34.2
Transformer-D, $L = 4000$	2.22216	33.6
Transformer-DMCA, no MoE-layer, $L = 11000$	2.05159	36.2
Transformer-DMCA, MoE-128, $L = 11000$	1.92871	37.9
Transformer-DMCA, MoE-256, $L = 7500$	1.90325	38.8

L = input window length.
ED = encoder-decoder.
D = decoder only.
DMCA = a memory compression technique (strided convolution).
MoE = mixture of experts layer.

Liu et al, “GENERATING WIKIPEDIA BY SUMMARIZING LONG SEQUENCES” arXiv 2018

Translation Takeaways

- Sequence-to-sequence translation

  - Input reversal
  - Narrow beam search

- Adding Attention

  - Compare latent states of encoder/decoder (Bahdanau).
  - Simplify and avoid more recurrence (Luong).

- Attention only models:

  - Self-attention replaces recurrence, improves performance.
  - Use depth to model hierarchical structure.
  - Multi-headed attention allows interpretation of inputs.

Translation Takeaways

- Attention only models:
  - Self-attention replaces recurrence, improves performance.
  - Use depth to model hierarchical structure.
  - Multi-headed attention allows interpretation of inputs.