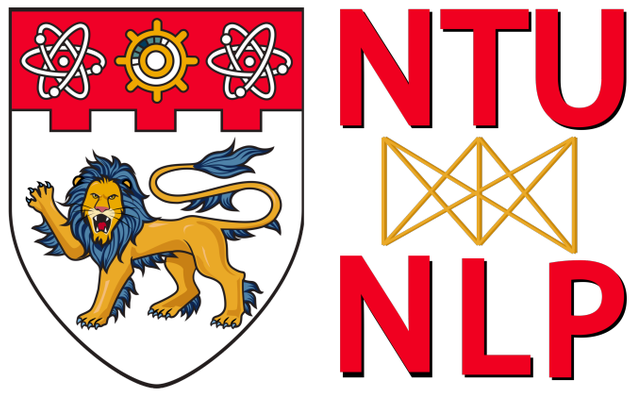
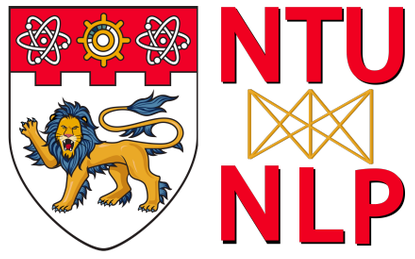
Deep Learning for Natural Language Processing

Shafiq Joty



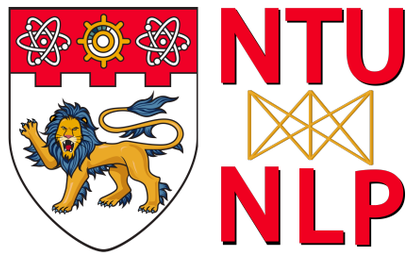
Lecture 9: Transformers

Recurrent Neural Networks



- Used to be the default model of choice for learning variable-length representations.
- Natural fit for sentences (or a sequence of objects)
- LSTMs and GRUs are the dominant recurrent models.

Recurrent Neural Networks



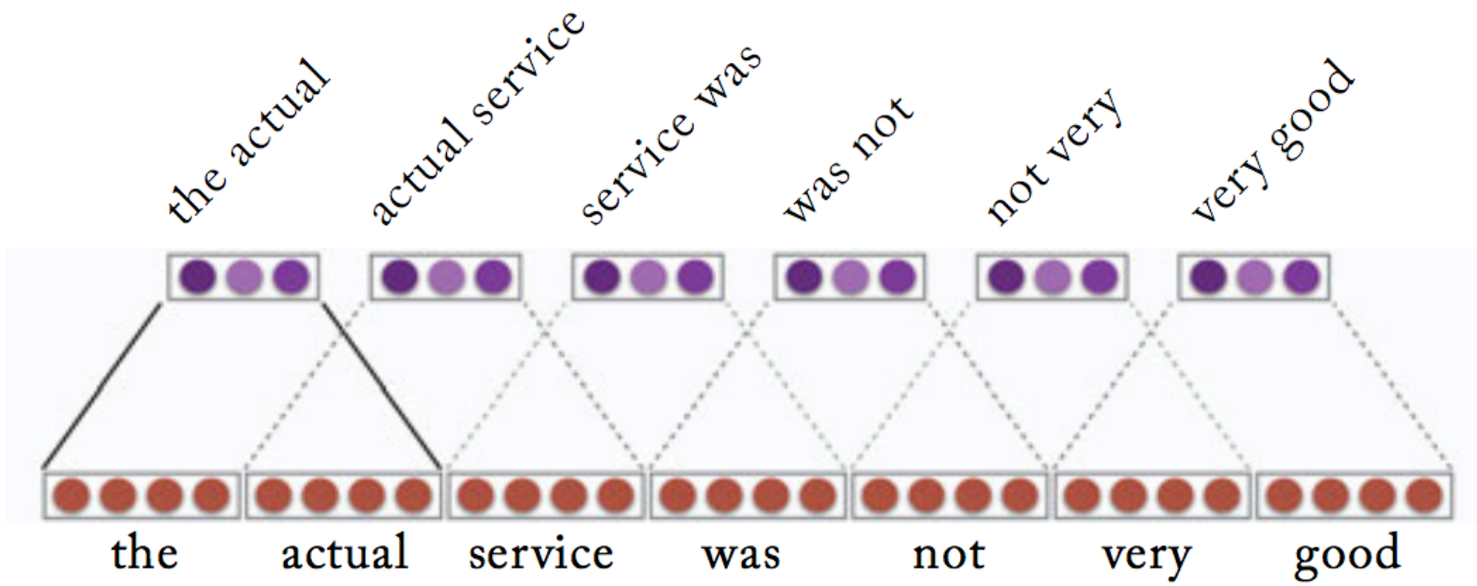
Issues with RNN Computations

- But sequential computation inhibits parallelization.
- No explicit modeling of long and short range dependencies
- Not many hierarchy of features.
- RNNs are a bit wasteful!

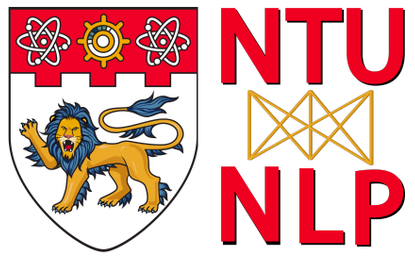
Convolutional Neural Networks?

NTU NLP

- Per-layer Parallelizeable
- Exploits local dependencies
- 'Interaction distance' between positions linear or logarithmic
- Modeling long-distance dependencies require many layers



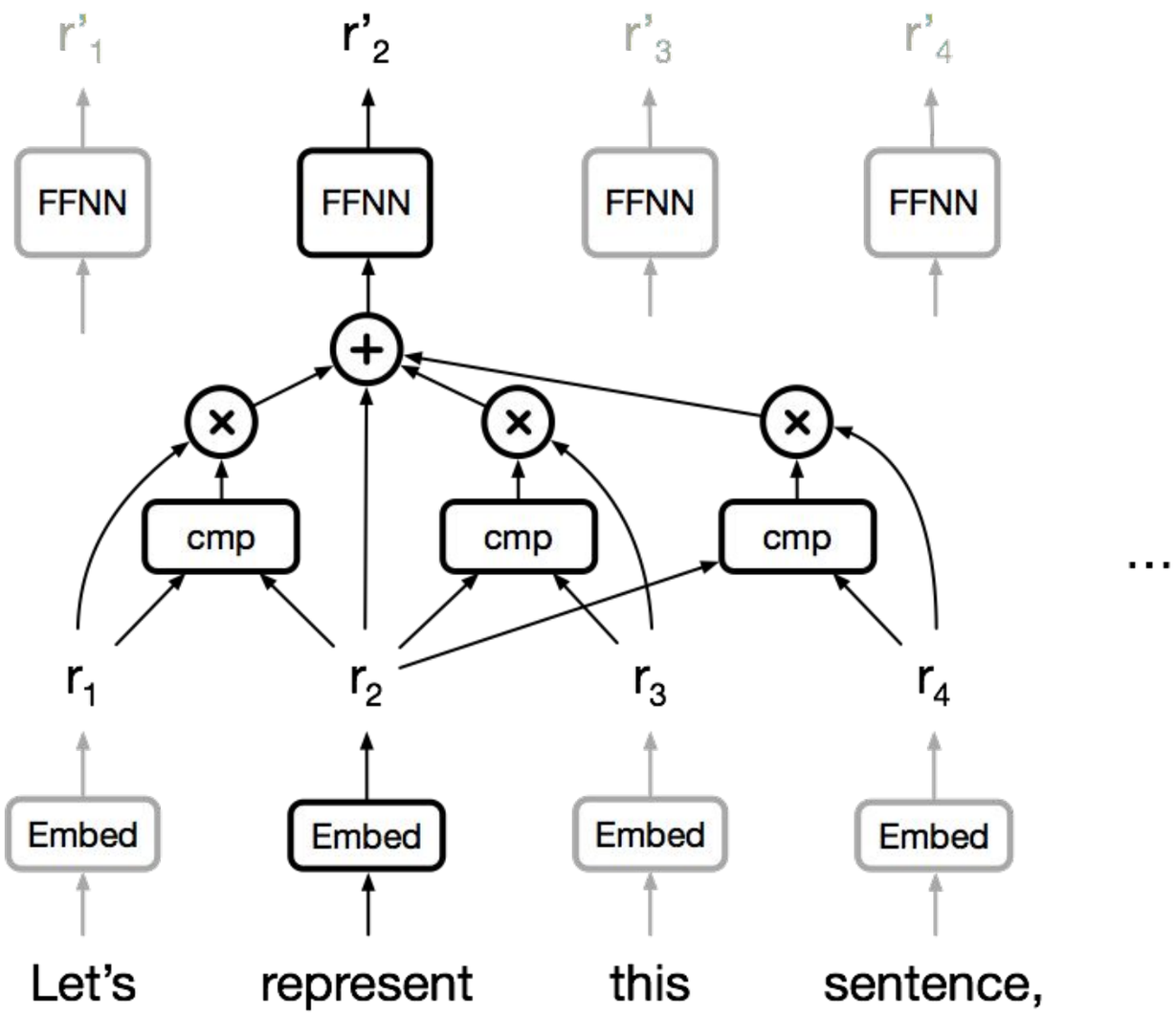
Attentions?



- Attention between encoder and decoder is crucial in NMT

Why not use attention for representations?

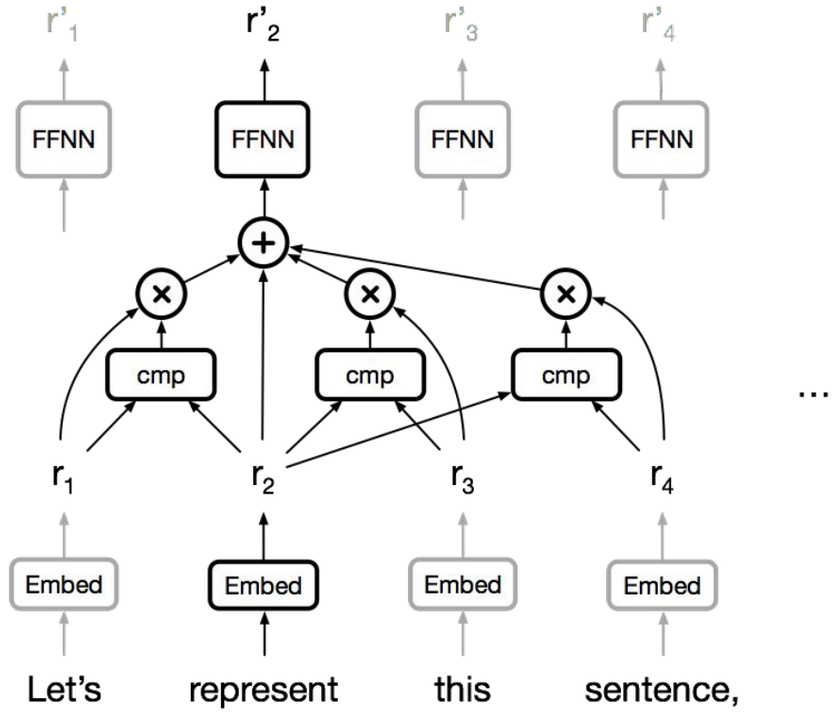
Self Attentions



source: Vaswani et al.

Self Attentions

- Constant 'path length' between any two positions.
- (Soft) Gating & multiplicative interactions.
- Easy to parallelize (per layer).
- Can replace sequential computation entirely

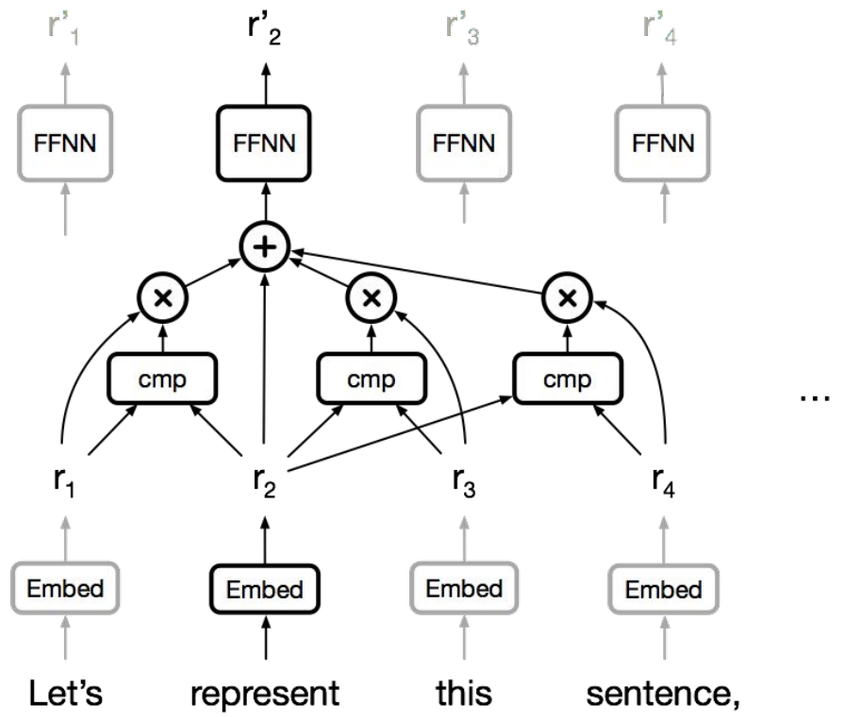


source: Vaswani et al.

Attention is Cheap

FLOPs

Self-Attention	O(length^2 · dim)
RNN (LSTM)	O(length · dim^2)
Convolution	O(length · dim^2 · kernel_width)



source: Vaswani et al.

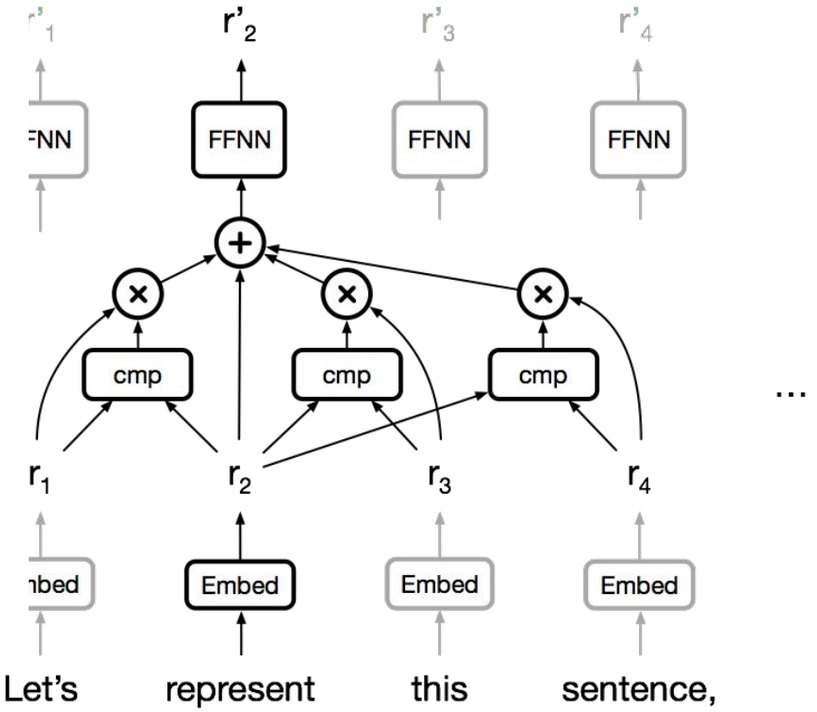
Attention is Cheap

NTU NLP

FLOPs

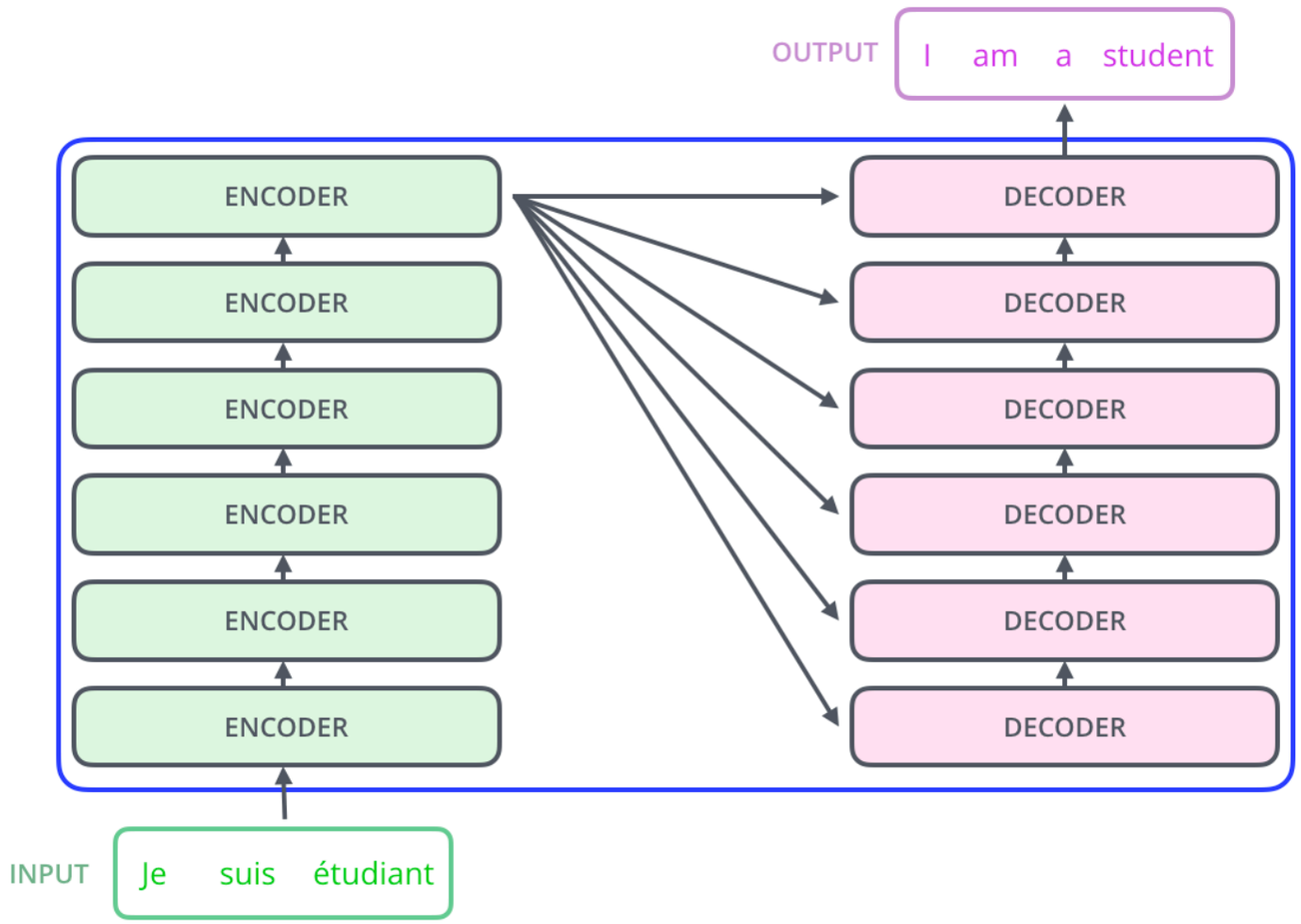
Self-Attention	$O(\text{length}^2 \cdot \text{dim})$	$= 4 \cdot 10^9$
RNN (LSTM)	$O(\text{length} \cdot \text{dim}^2)$	$= 16 \cdot 10^9$
Convolution	$O(\text{length} \cdot \text{dim}^2 \cdot \text{kernel\_width})$	$= 6 \cdot 10^9$

length=1000 dim=1000 kernel_width=3



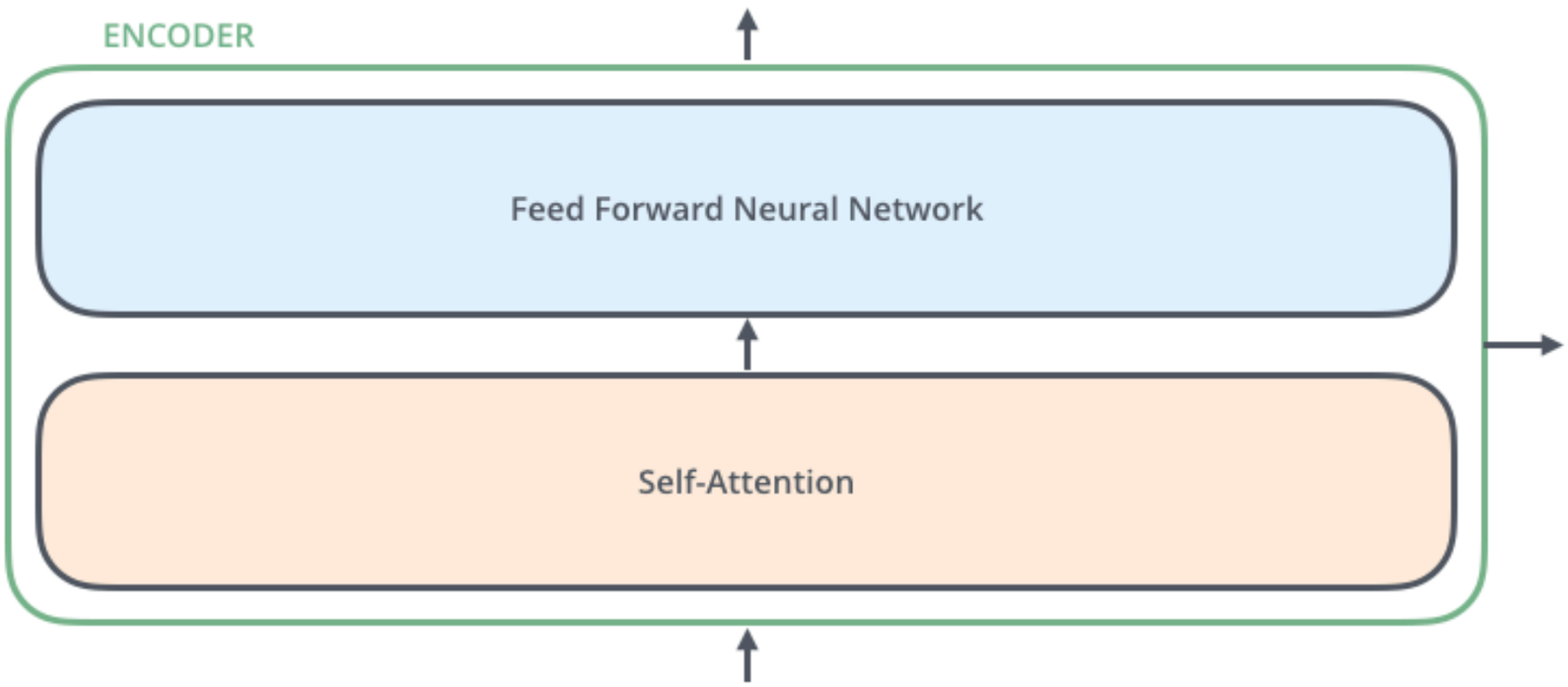
source: Vaswani et al.

Encoder-Decoder Network



source: https://jalammar.github.io/

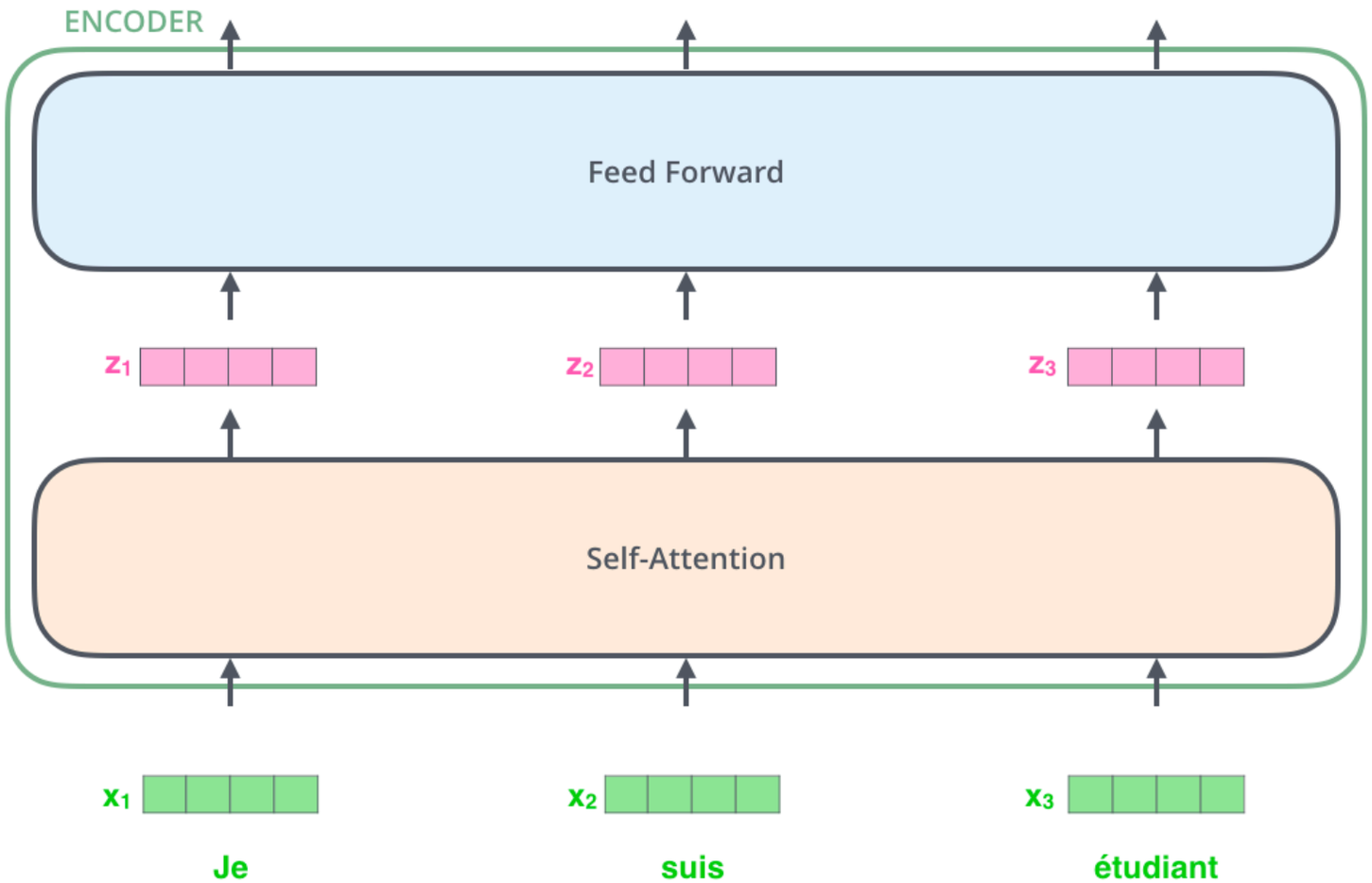
One Encoder Layer



- The encoder's inputs first flow through a self-attention layer
- The outputs of the self-attention layer are fed to a feed-forward neural network. The exact same feed-forward network is independently applied to each position.

source: https://jalammar.github.io/

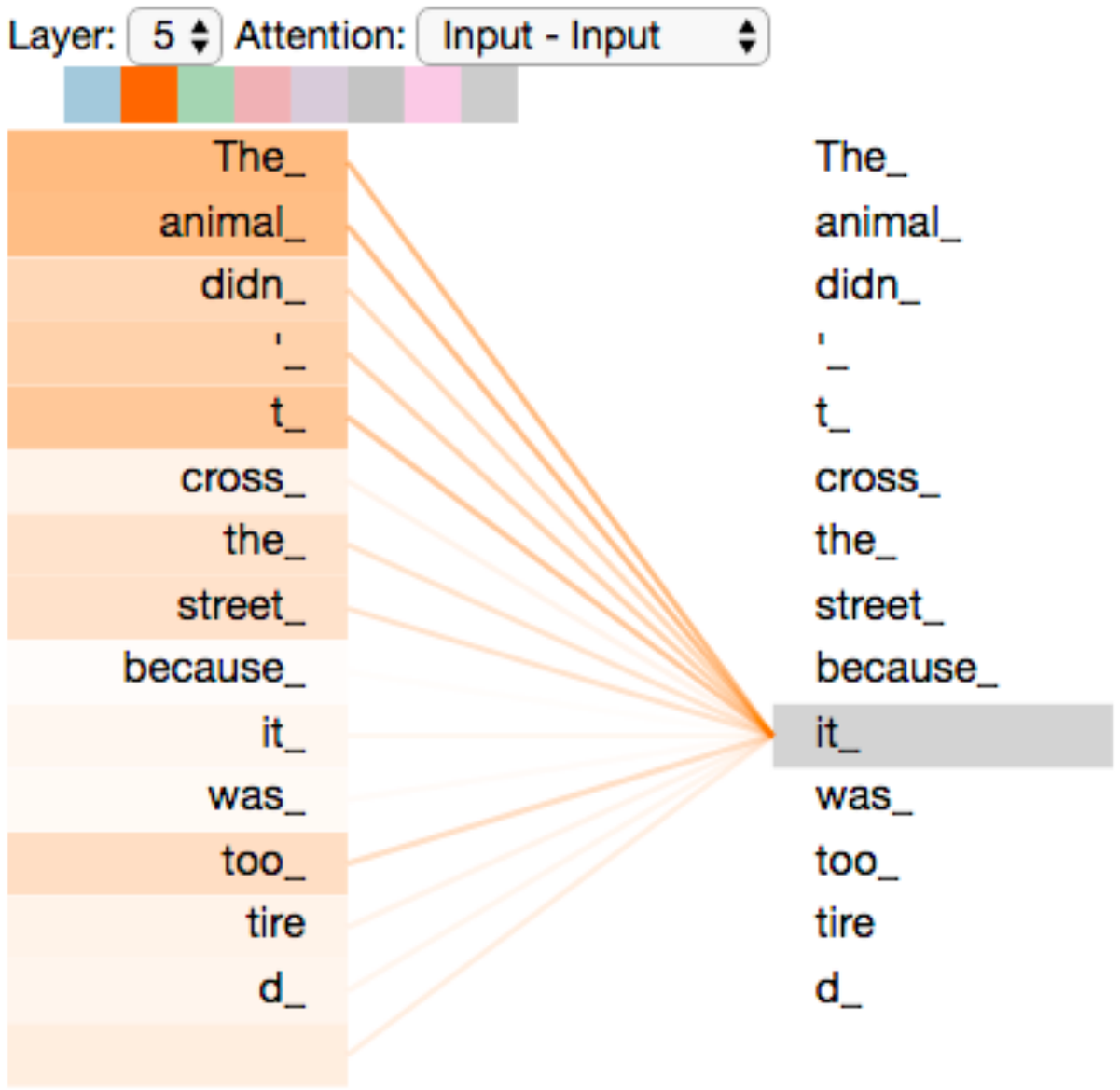
The First Encoder Layer



- The dependencies between word vectors are modelled by the self-attention layer.

source: https://jalammar.github.io/

One Self-Attention Layer



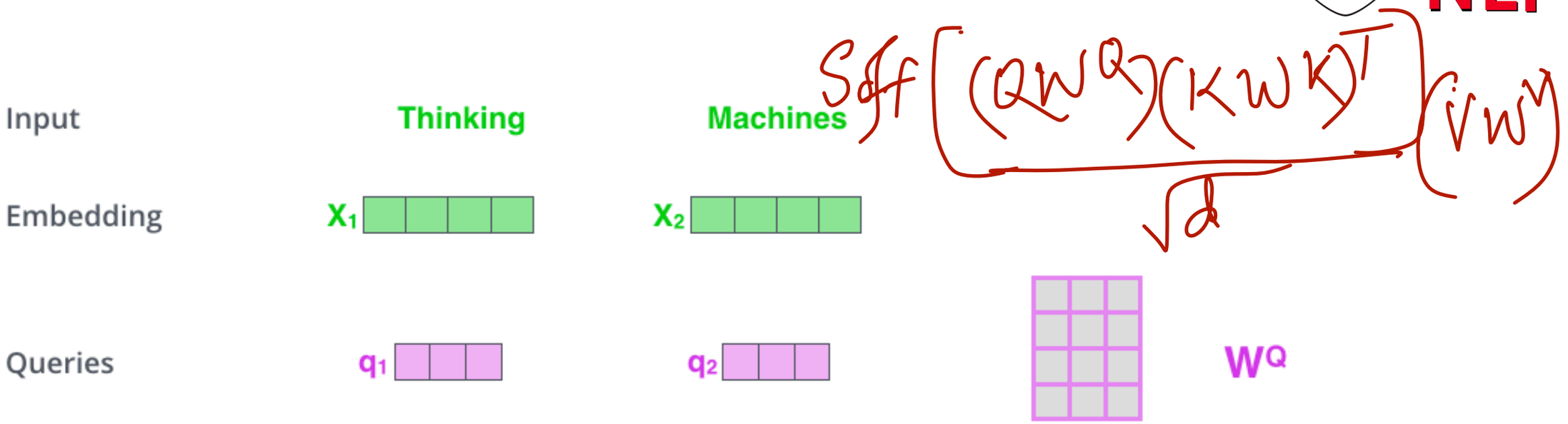
- The dependencies between word vectors are modelled by the self-attention layer.

source: https://jalammar.github.io/

One Self-Attention Layer

NTU NLP

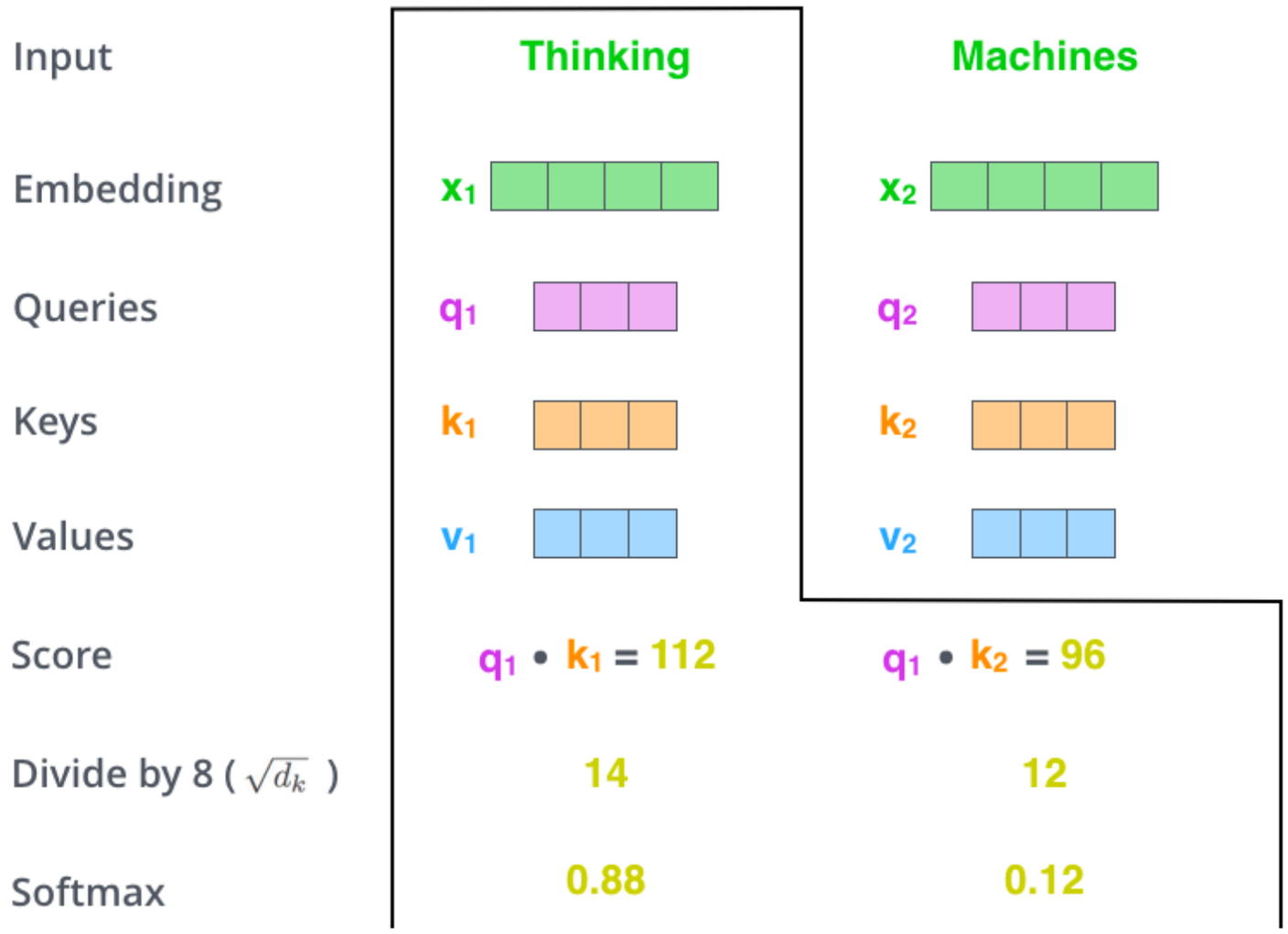
$$\text{Soft}\left[\frac{(QW^Q)(KW^K)^T}{\sqrt{d}}\right](VW^V)$$



- The dependencies between word vectors are modelled by the self-attention layer.

source: https://jalammar.github.io/

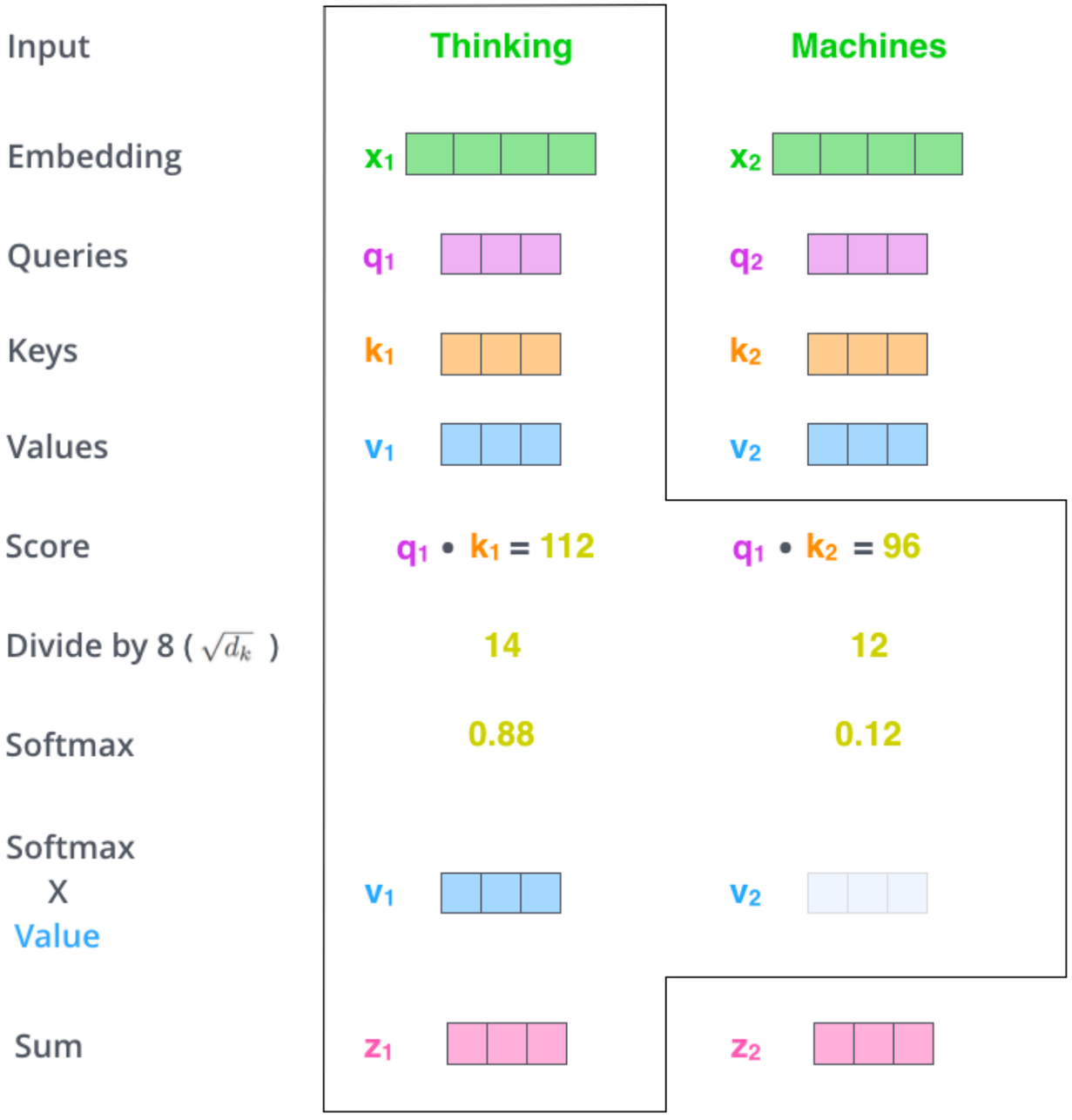
One Self-Attention Layer



- The dependencies between word vectors are modelled by the self-attention layer.

source: https://jalammar.github.io/

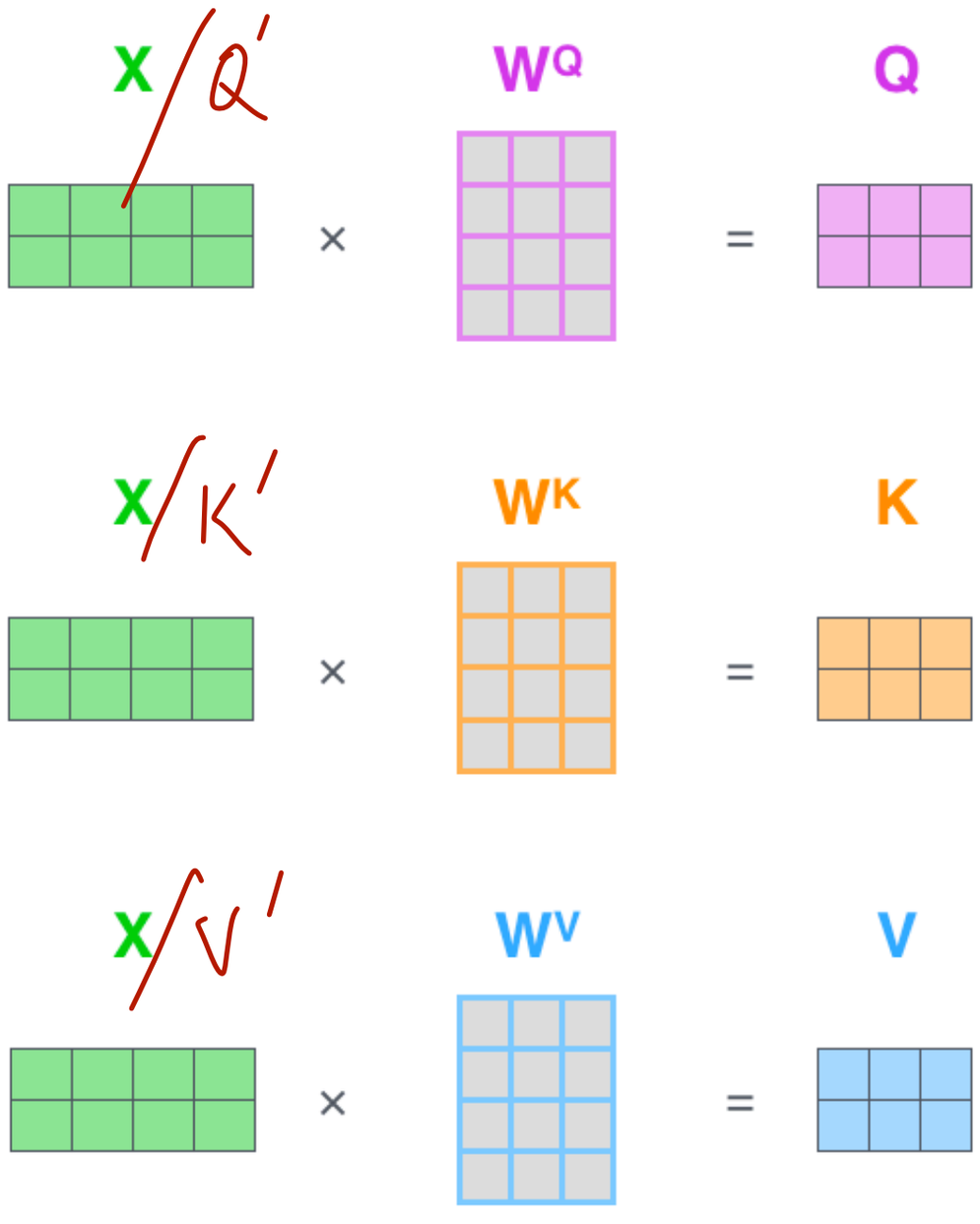
One Self-Attention Layer



- The dependencies between word vectors are modelled by the self-attention layer.

source: https://jalammar.github.io/

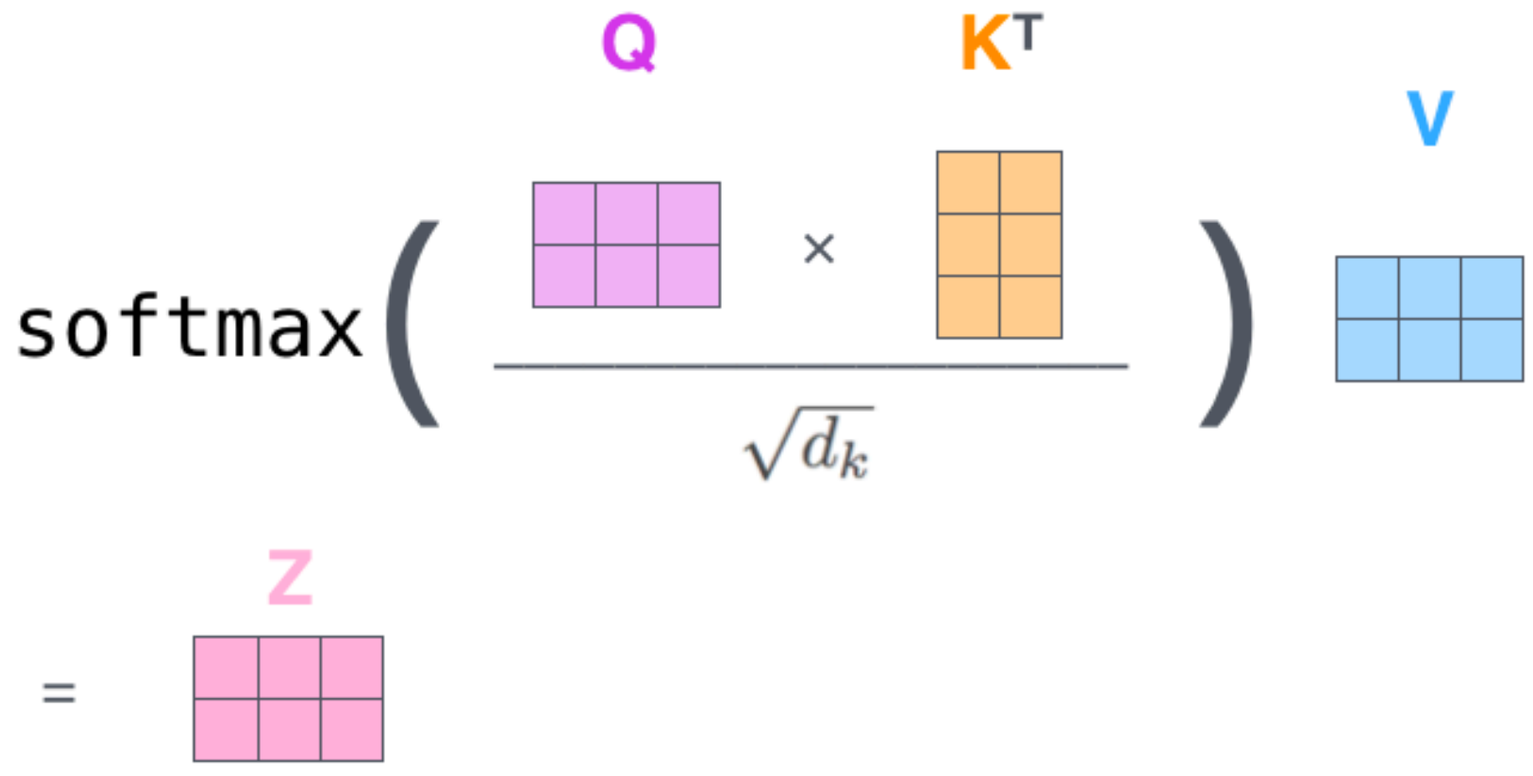
One Self-Attention Layer



- The dependencies between word vectors are modelled by the self-attention layer.

source: https://jalammar.github.io/

One Self-Attention Layer

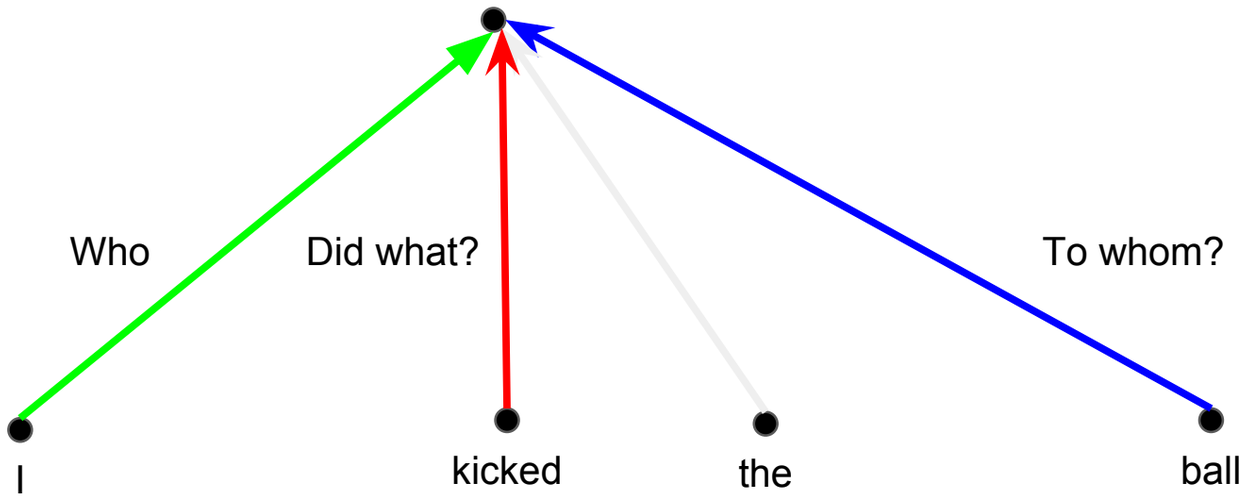


- The dependencies between word vectors are modelled by the self-attention layer.

source: https://jalammar.github.io/

Multi-head Attention

- Convolution:



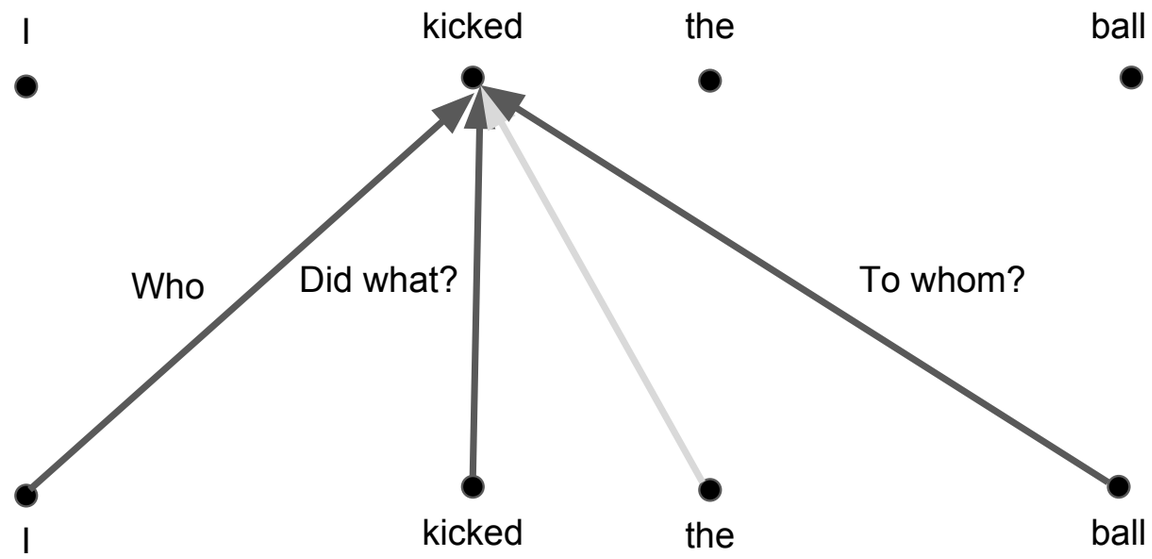
- Different kernels extract different features

source: Vaswani et al.

Multi-head Attention

- Self-attention:

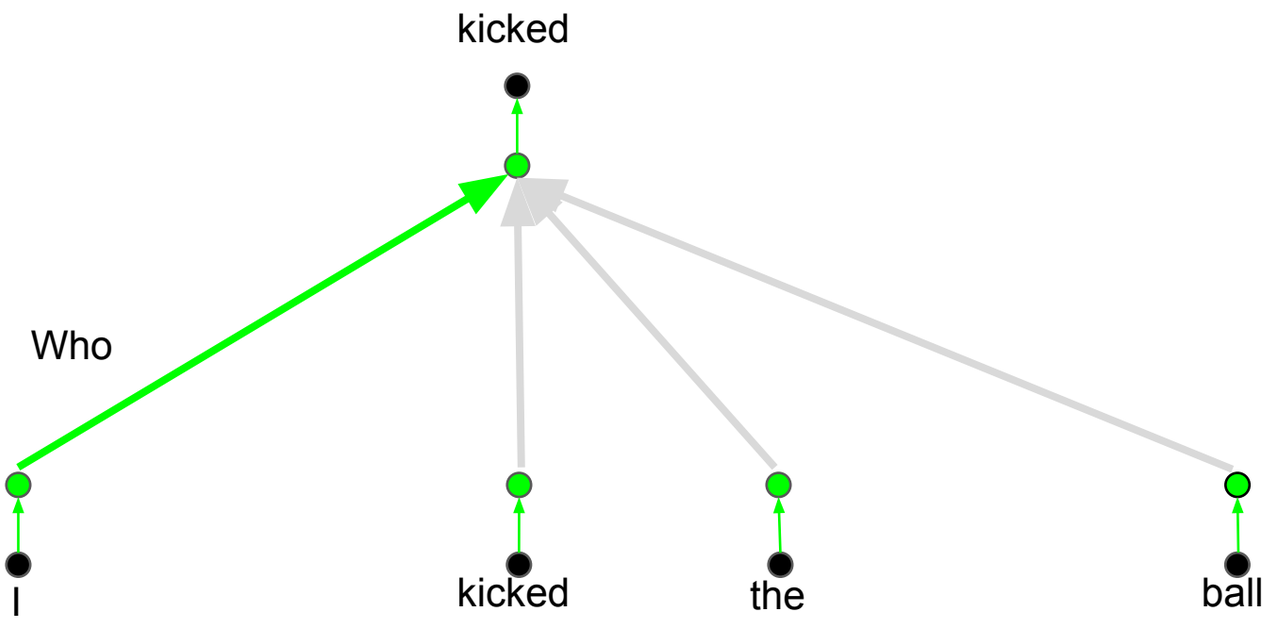
$$\text{Soft}\left[\frac{(QW^Q)(KW^K)^T}{\sqrt{d}}\right](VW^V)$$



- All these are mixed in a single vector

source: Vaswani et al.

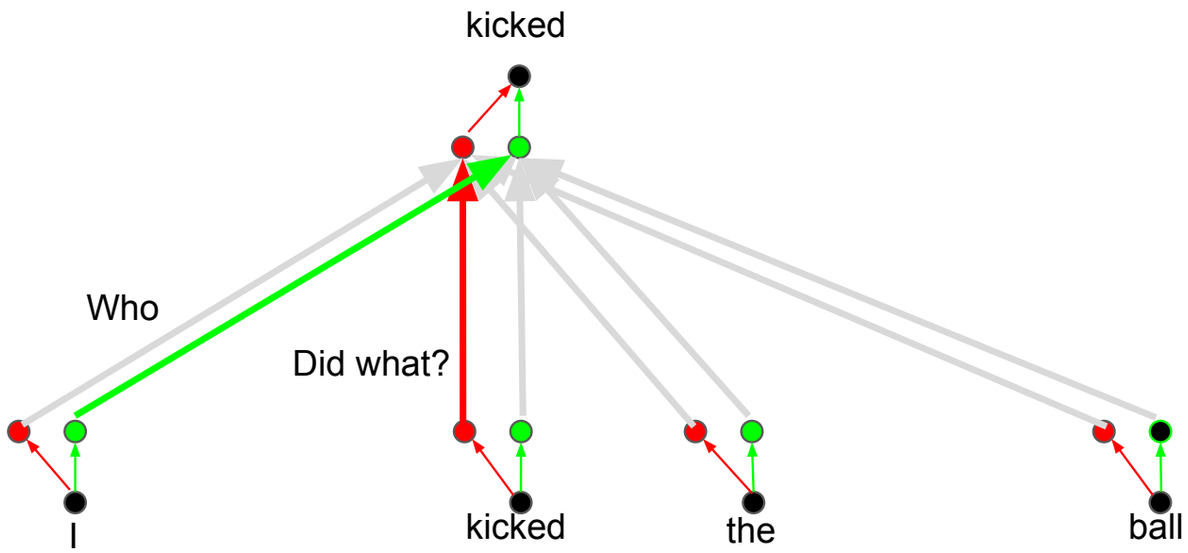
Self-attention - Who head



- Who head (kernel) extracts subj information

source: Vaswani et al.

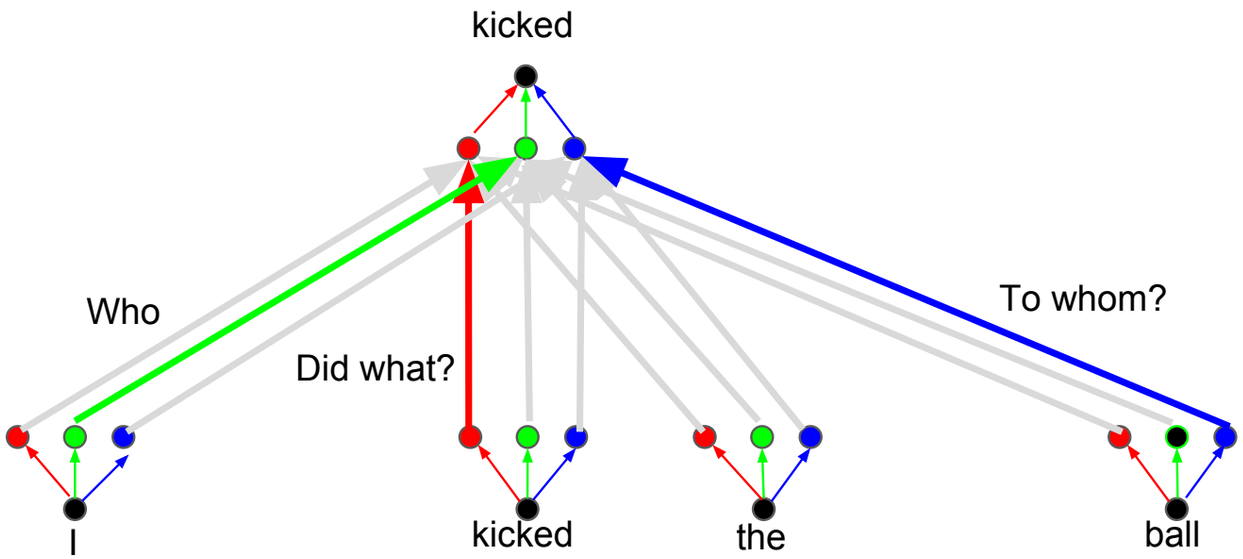
Self-attention - Did what head



- Did What head (kernel) extracts verb information

source: Vaswani et al.

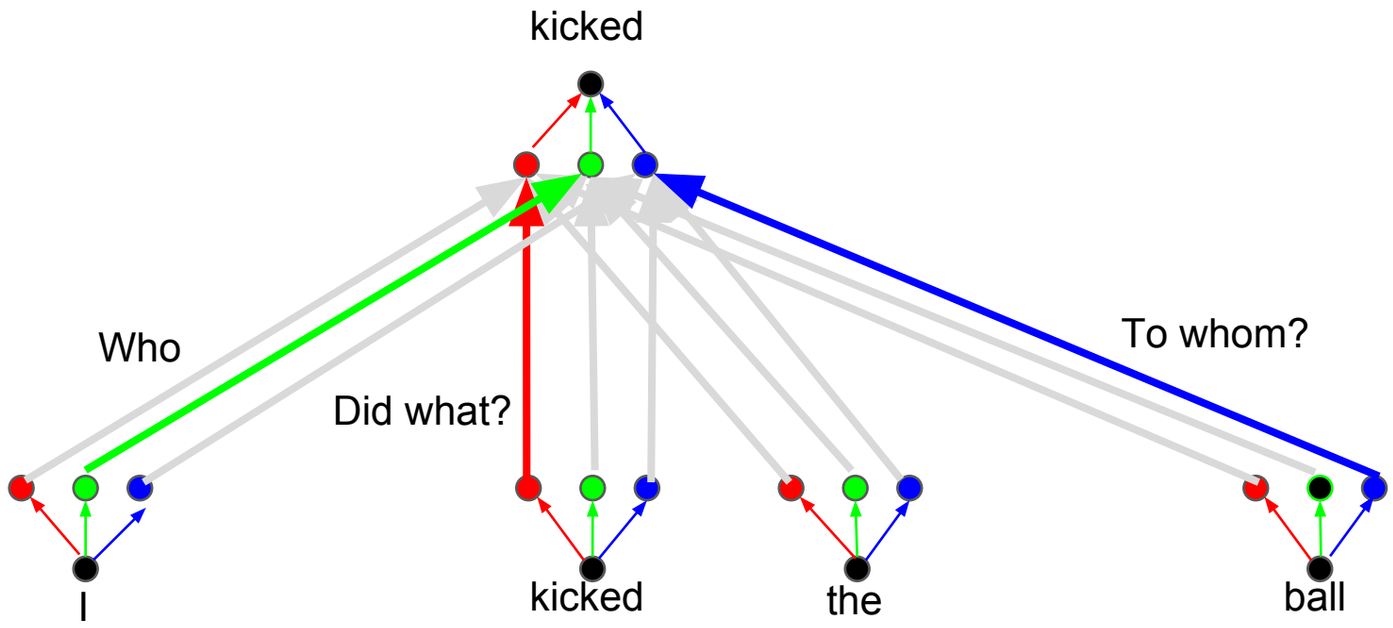
Self-attention - To whom head



- To Whom head (kernel) extracts object information

source: Vaswani et al.

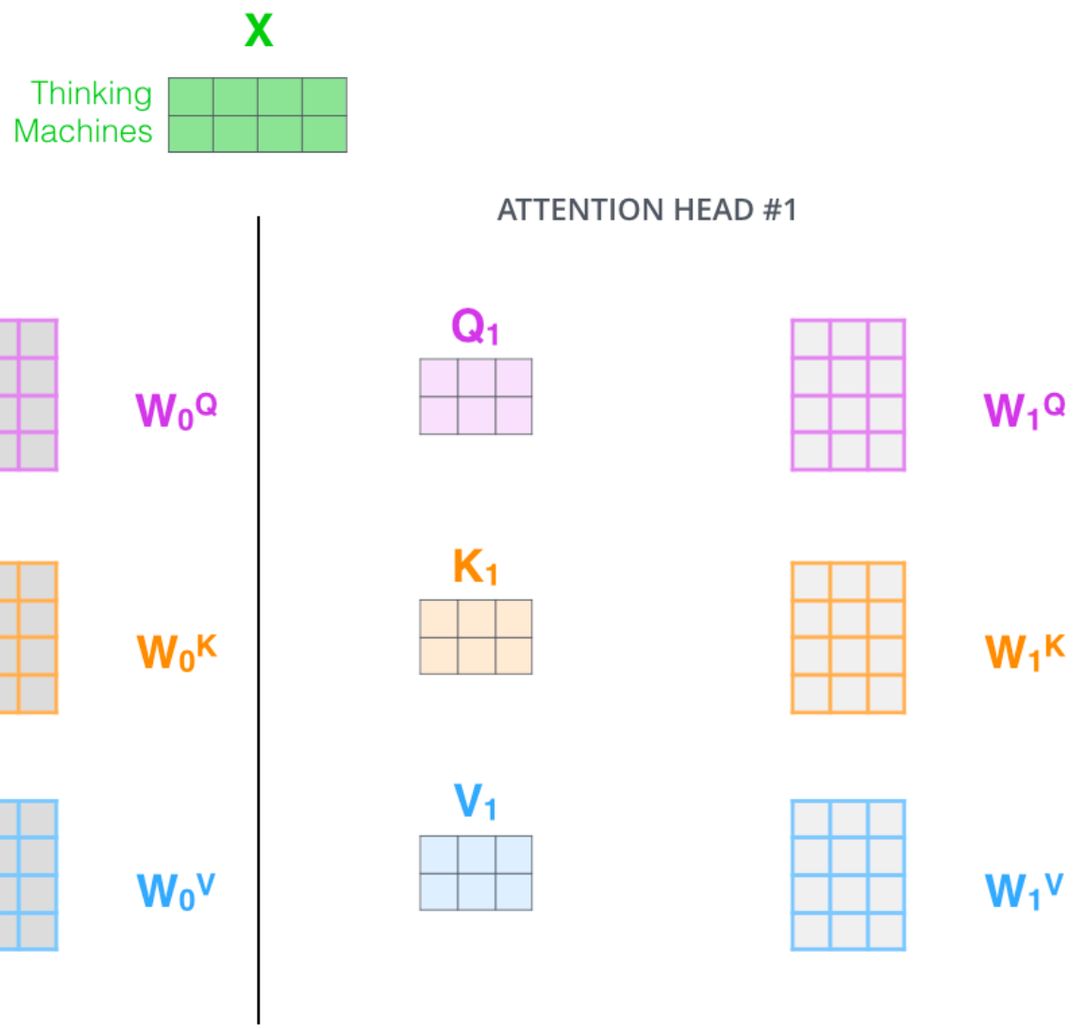
Self-attention - Multihead



- Multi-head extracts different aspects

source: Vaswani et al.

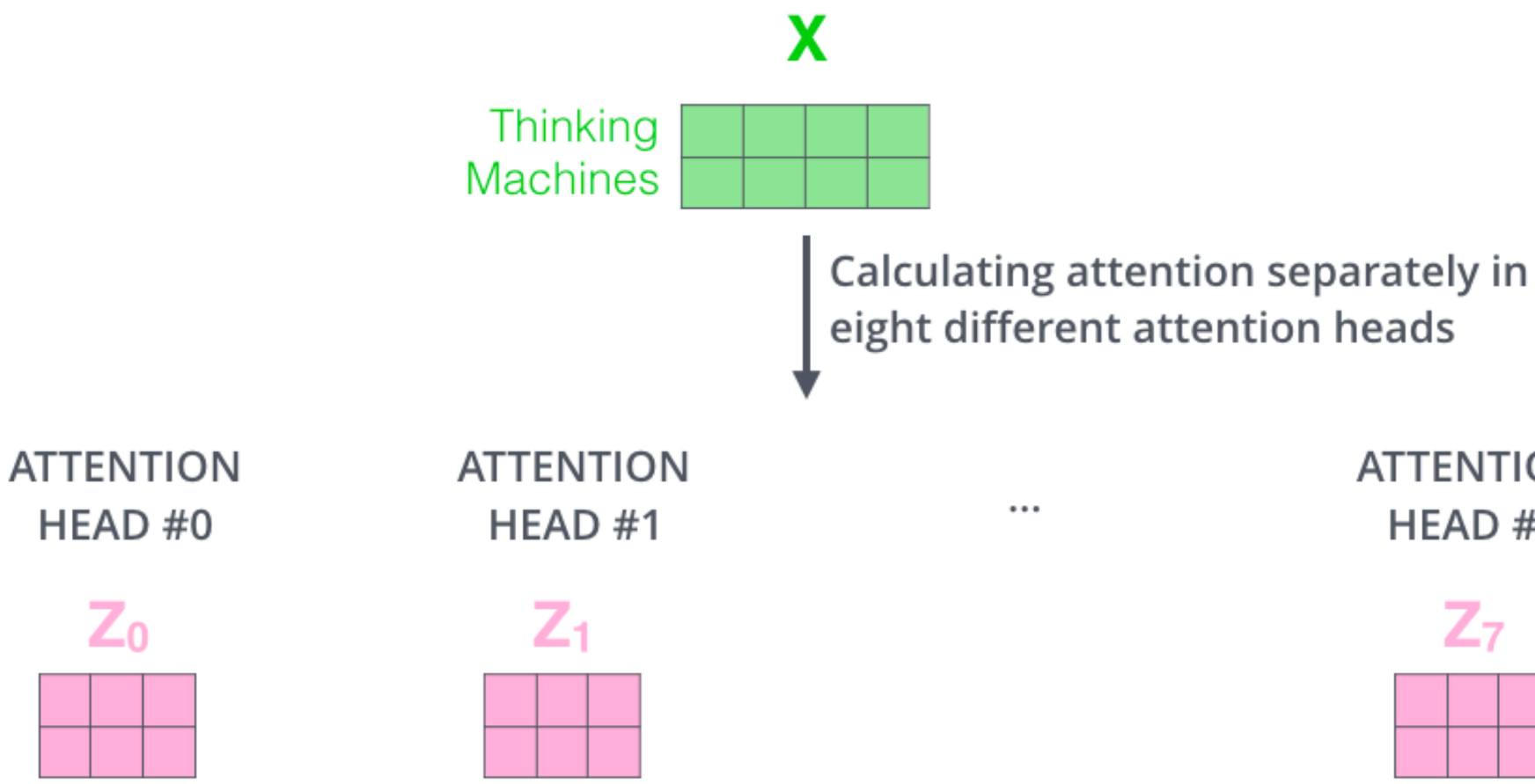
Using Multiple Heads



- Multiple heads expand the model's ability to focus on different positions.
- Each head gives the attention layer multiple representation subspaces

source: https://jalammar.github.io/

Using Multiple Heads

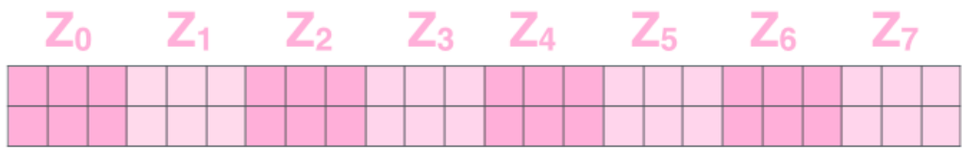


- Transformer uses 8 different heads

source: https://jalammar.github.io/

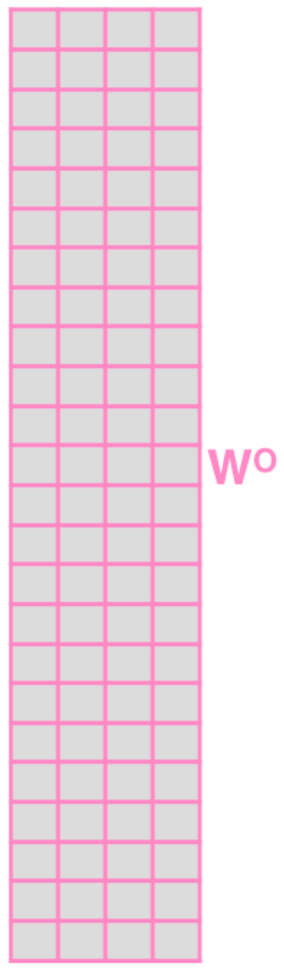
Using Multiple Heads

1) Concatenate all the attention heads



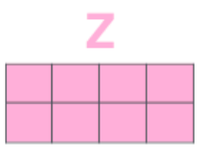
2) Multiply with a weight matrix W^O that was trained jointly with the model

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

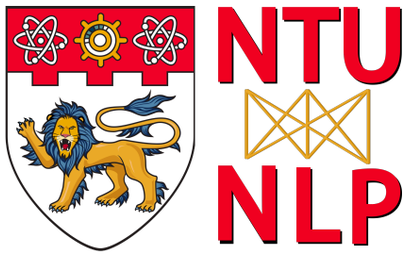
=



- Concatenate the 8-head's outputs and pass it the FFN

source: https://jalammar.github.io/

Using Multiple Heads



1) This is our input sentence*

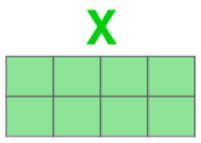
2) We embed each word*

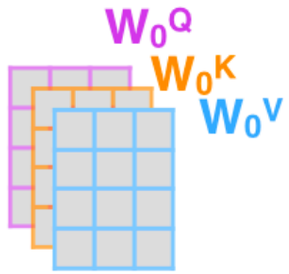
3) Split into 8 heads. We multiply X or R with weight matrices

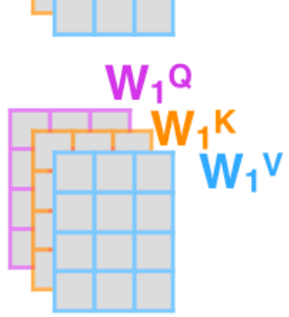
4) Calculate attention using the resulting Q/K/V matrices

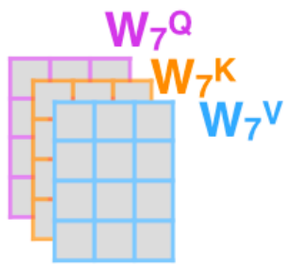
5) Concatenate the resulting Z matrices, then multiply with weight matrix W^O to produce the output of the layer

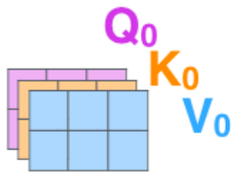
Thinking
Machines

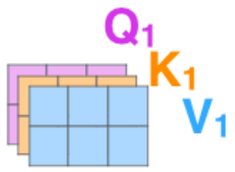


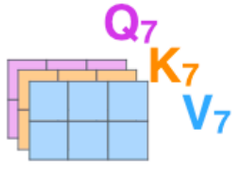


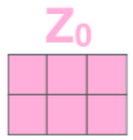


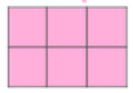


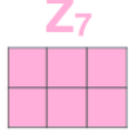


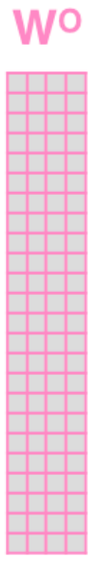




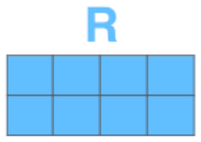






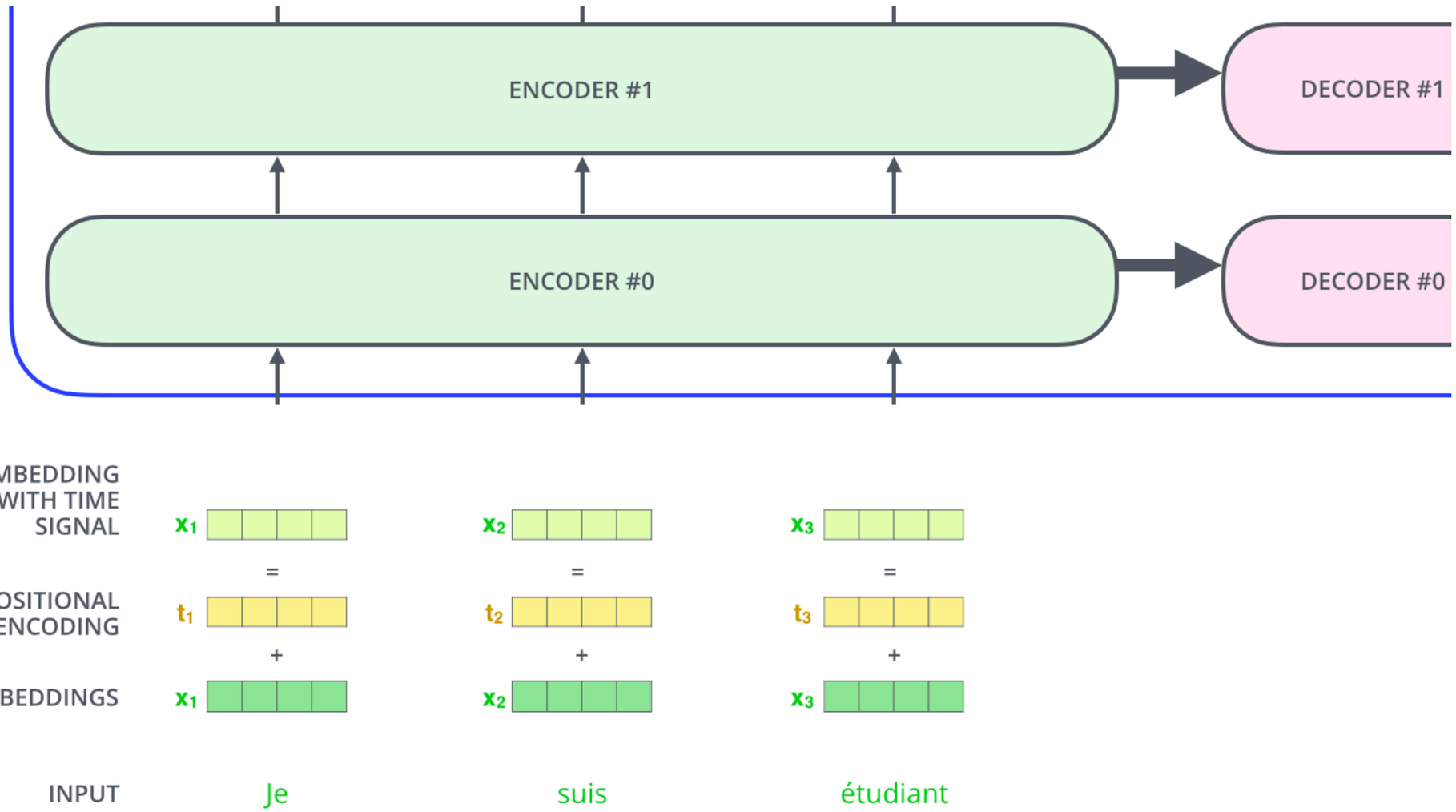


* In all encoders other than #0,
we don't need embedding.
We start directly with the output
of the encoder right below this one



source: https://jalammar.github.io/

Injecting Order into Embeddings

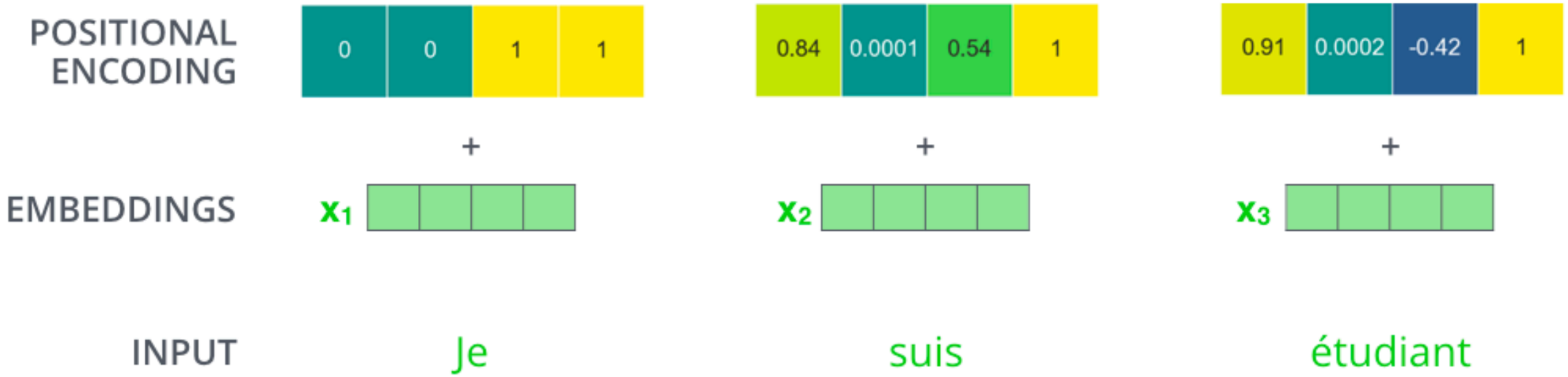


To give the model a sense of the order of the words, we add positional encoding vectors -- the values of which follow a specific pattern (sine/cosine).

source: https://jalammar.github.io/

Injecting Order into Embeddings

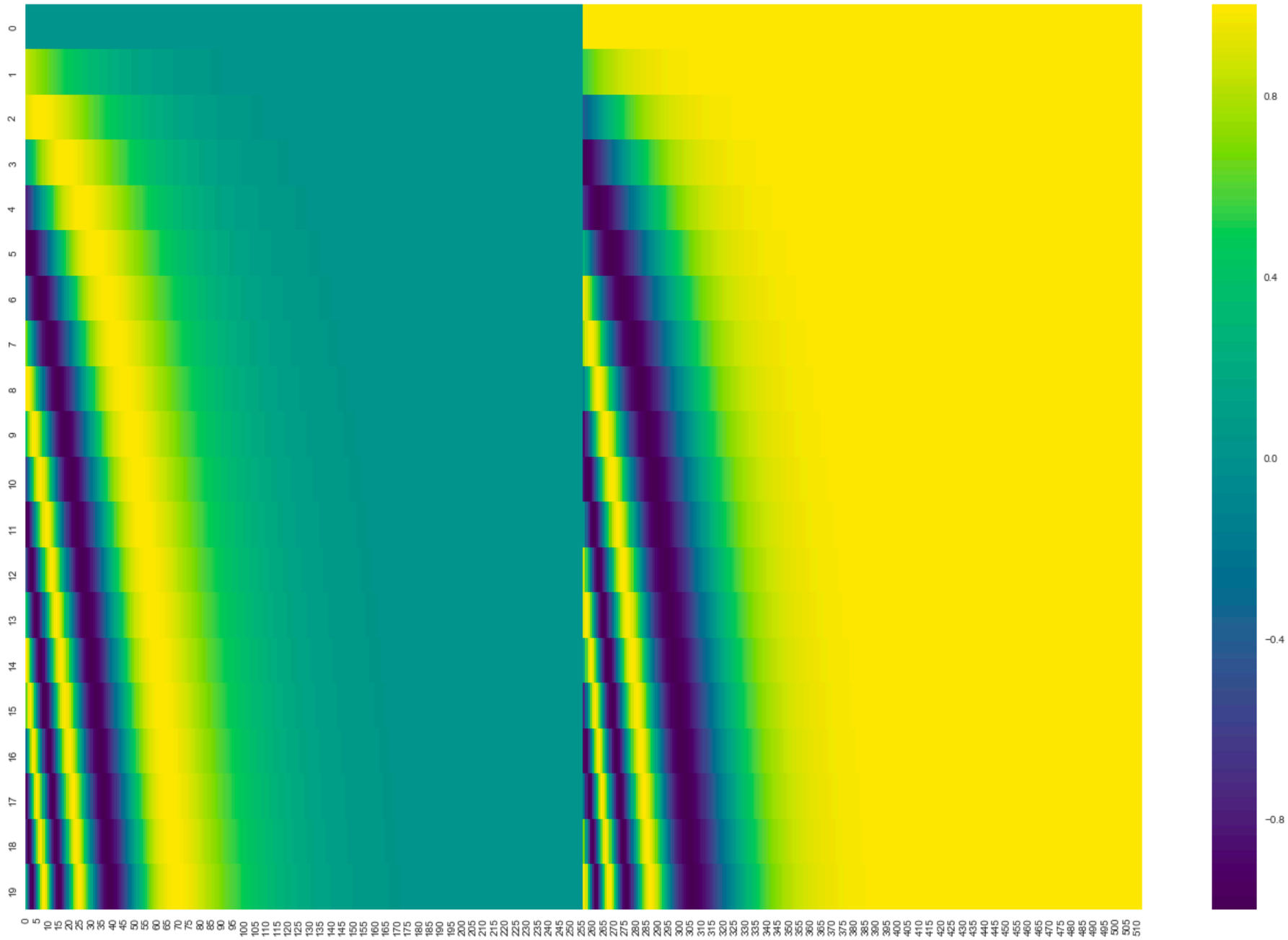
- Actual word representations are byte-pair encodings
  - As seen in last lecture



Example of positional encoding with a toy embedding size of 4

source: https://jalammar.github.io/

Injecting Order into Embeddings

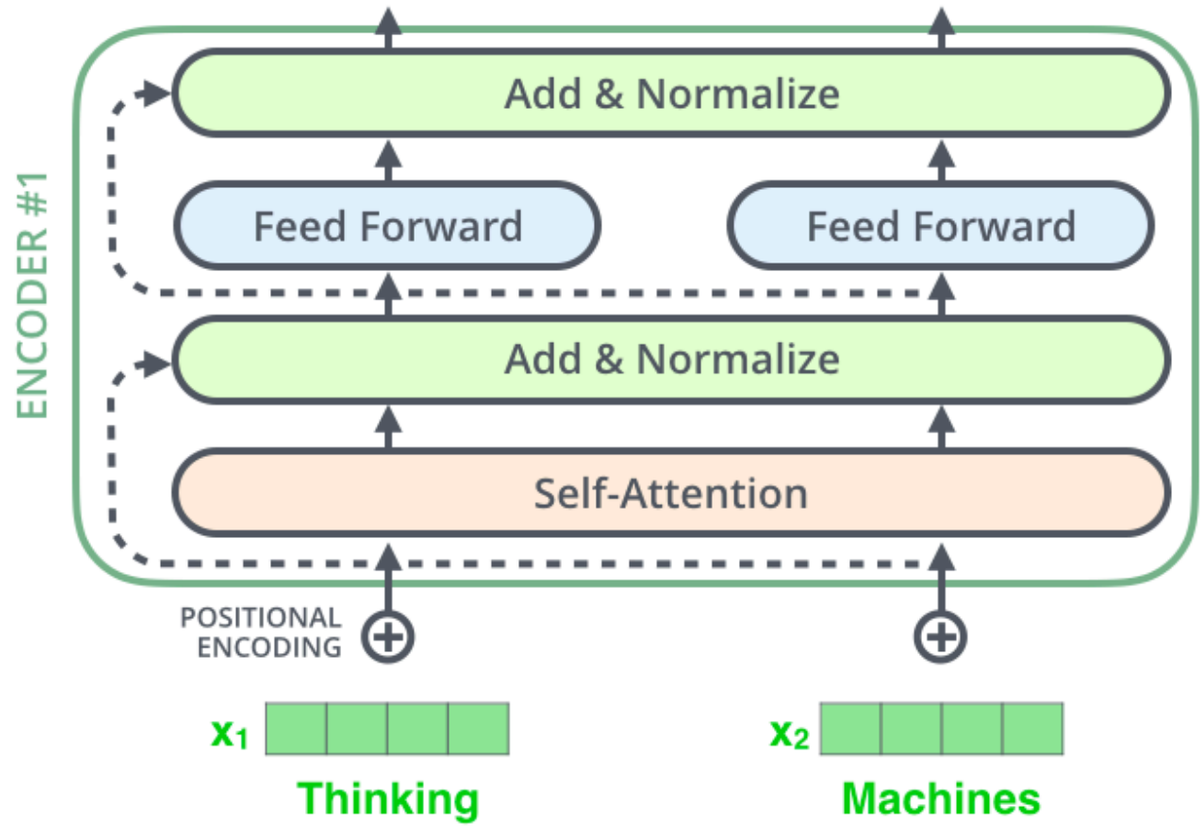


real example of positional encoding for 20 words (rows) with an embedding size of 512 (columns)

source: https://jalammar.github.io/

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
- Layernorm changes input to have mean 0 and variance 1, per layer and per training point (and adds two more parameters)

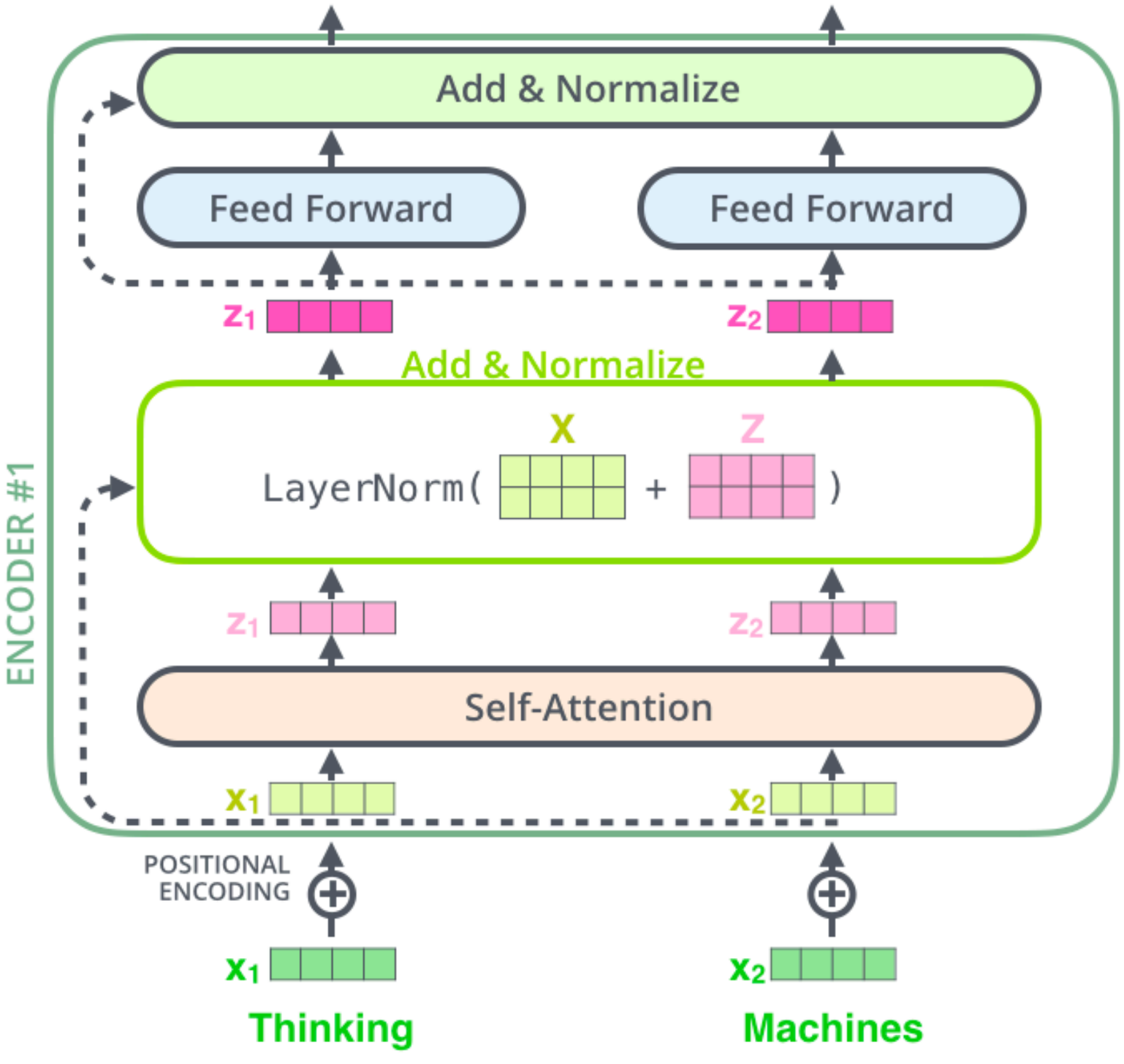


$$\mu^l = \frac{1}{H}\sum_{i=1}^{H} a_i^l \qquad \sigma^l = \sqrt{\frac{1}{H}\sum_{i=1}^{H}(a_i^l - \mu^l)^2}$$

$$h_i = f\left(\frac{g_i}{\sigma_i}(a_i - \mu_i) + b_i\right)$$

source: https://jalammar.github.io/

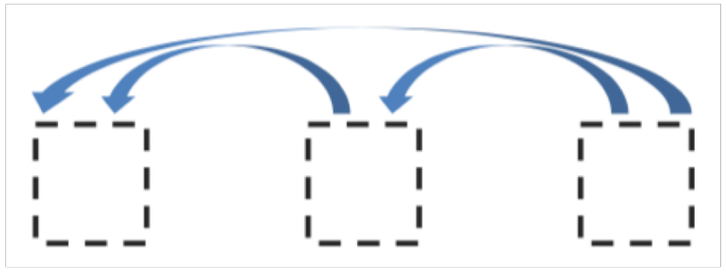
The Residuals



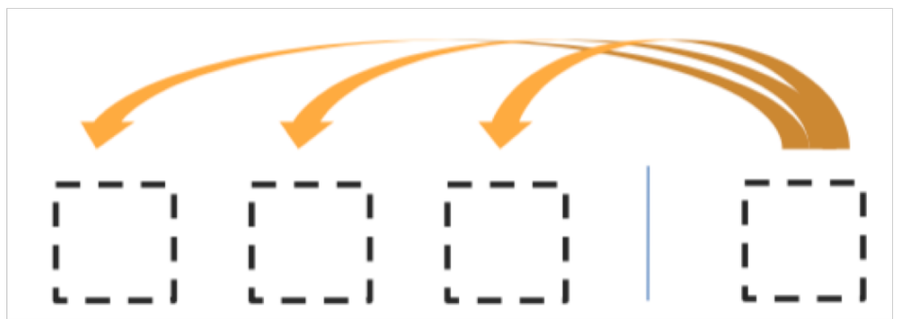
source: https://jalammar.github.io/

Decoder Layers

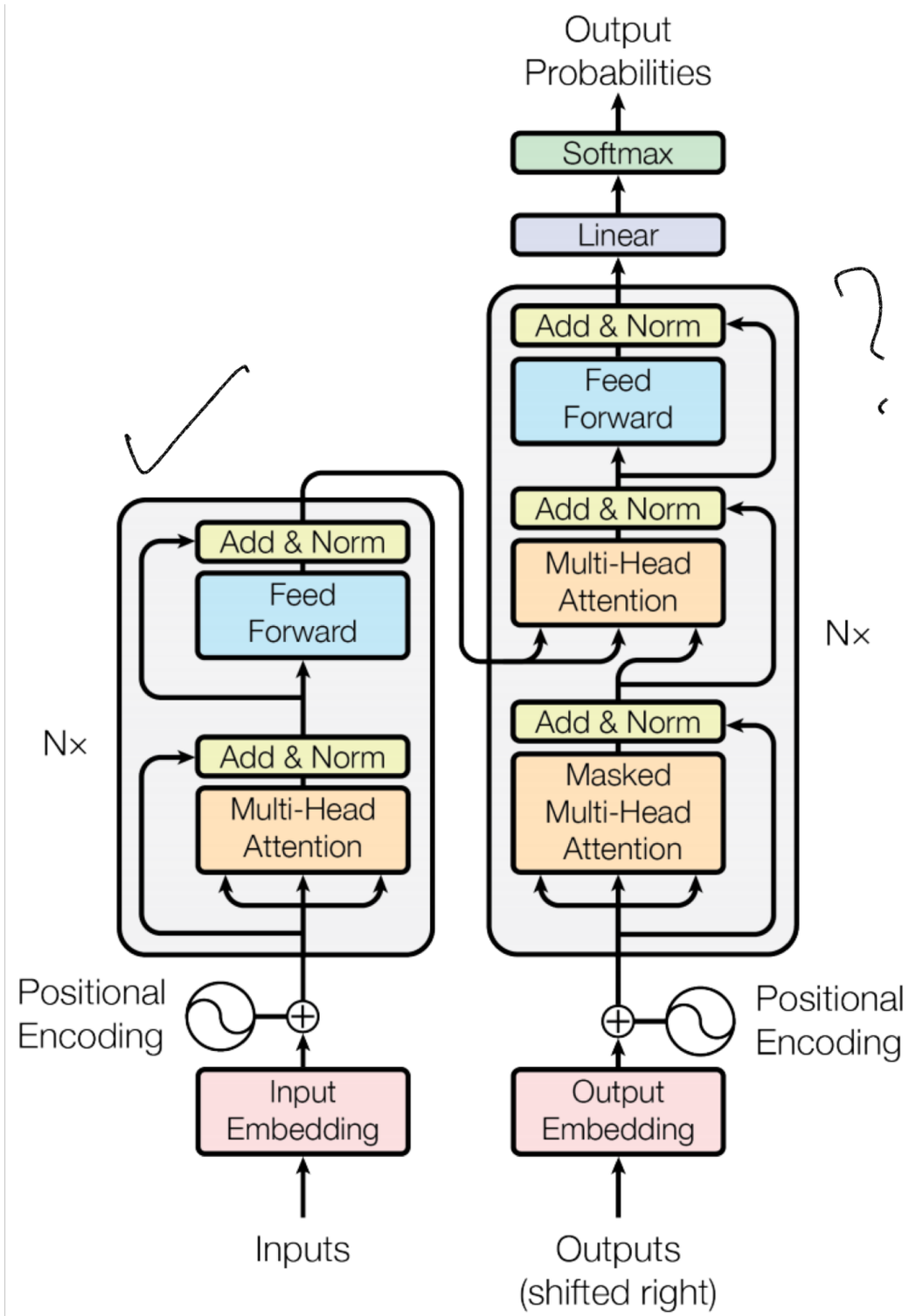
- 2 sublayer changes in decoder
- Masked decoder self-attention on previously generated outputs:



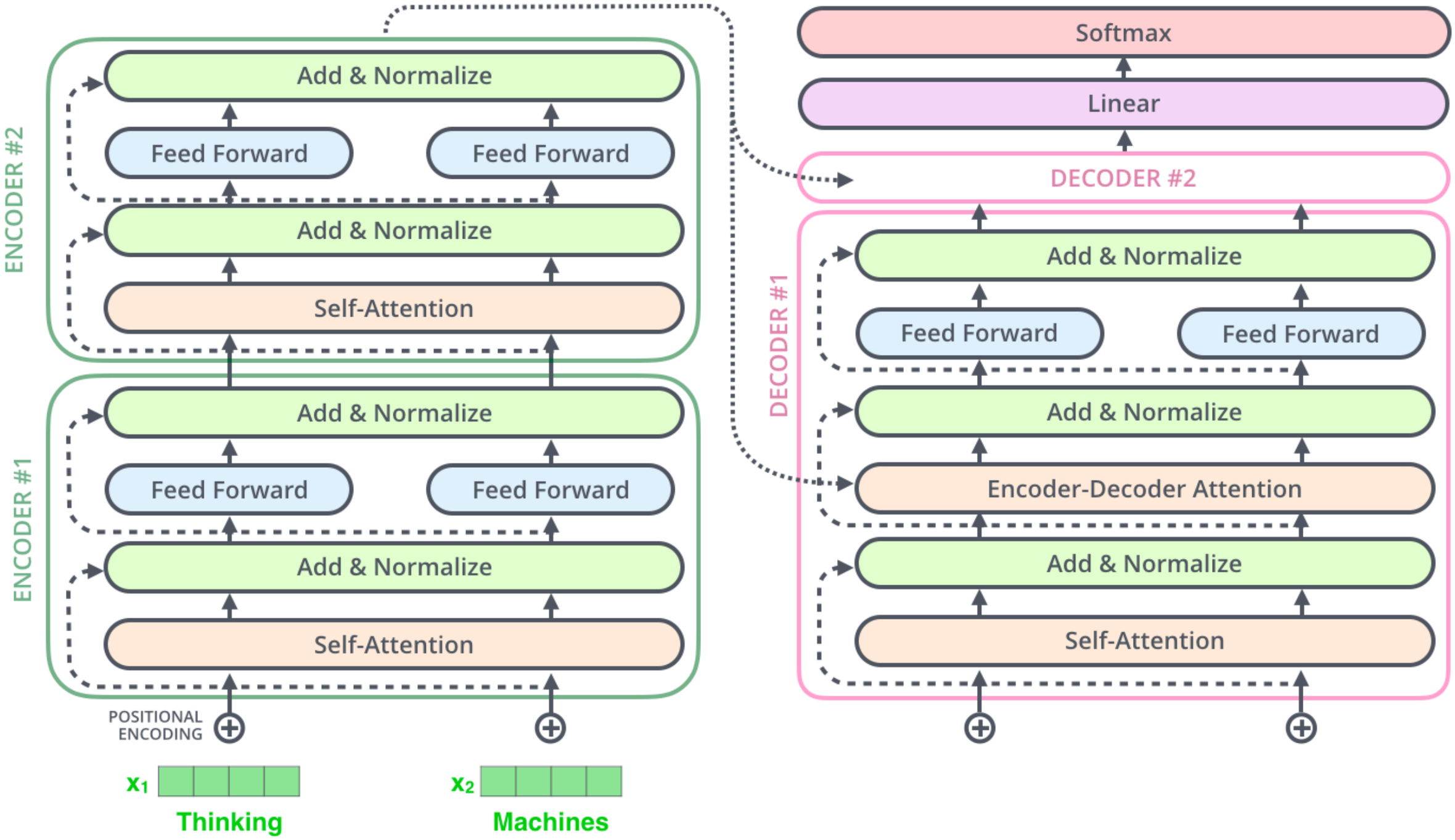
- Encoder-Decoder Attention, where queries come from previous decoder layer and keys and values come from output of encoder



- Blocks repeated 6 times also

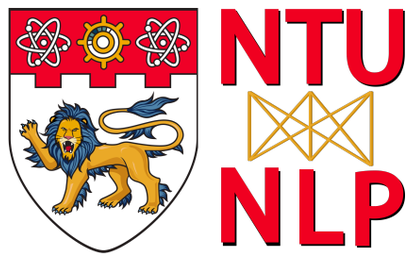


Whole Encoder-Decoder Model



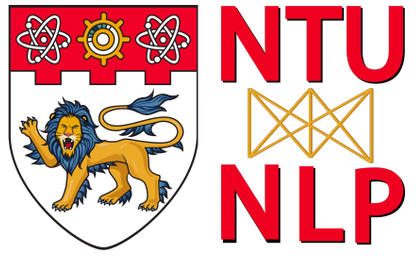
source: https://jalammar.github.io/

Transformer (Minor) Details



- Details (in paper and/or later lectures):
- Byte-pair encodings
- Checkpoint averaging
- ADAM optimizer with learning rate changes
- Dropout during training at every layer just before adding residual
- Label smoothing
- Auto-regressive decoding with beam search and length penalties
- Use of transformers is spreading but they are hard to optimize and unlike LSTMs don't usually just work out of the box and they don't play well yet with other building blocks on tasks.

Transformer Resources



Learning about transformers on your own?

- Key recommended resource:
  - http://nlp.seas.harvard.edu/2018/04/03/attention.html
- The Annotated Transformer by SashaRush

  Jupyter Notebook using PyTorch that explains everything!

Results (Translation)

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	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 [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [38]	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	**$3.3 \cdot 10^{18}$**	
Transformer (big)	**28.4**	**41.8**	$2.3 \cdot 10^{19}$	

Results (Parsing)

Parser	Training	WSJ 23 F1
Vinyals & Kaiser el al. (2014) [37]	WSJ only, discriminative	88.3
Petrov et al. (2006) [29]	WSJ only, discriminative	90.4
Zhu et al. (2013) [40]	WSJ only, discriminative	90.4
Dyer et al. (2016) [8]	WSJ only, discriminative	91.7
Transformer (4 layers)	WSJ only, discriminative	91.3
Zhu et al. (2013) [40]	semi-supervised	91.3
Huang & Harper (2009) [14]	semi-supervised	91.3
McClosky et al. (2006) [26]	semi-supervised	92.1
Vinyals & Kaiser el al. (2014) [37]	semi-supervised	92.1
Transformer (4 layers)	semi-supervised	92.7
Luong et al. (2015) [23]	multi-task	93.0
Dyer et al. (2016) [8]	generative	93.3

Summary

NTU NLP