Lecture 9

Attention, Transformer

Ekaterina Lobacheva (actual talk by Ildus Sadrtdinov)

Outline

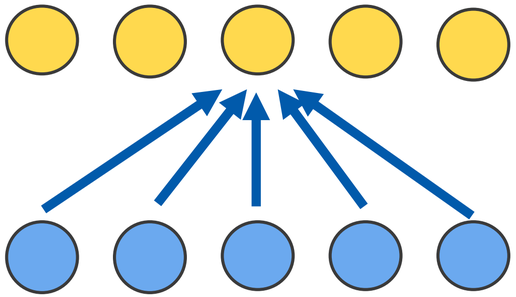
- Attention Layers
- Transformer Model
- Metrics and Inference for sequence generation
- Transformer-based LMs and contextualized word embeddings:

  ELMO, BERT, GPT
- Transformer for Computer Vision

Attention Layers

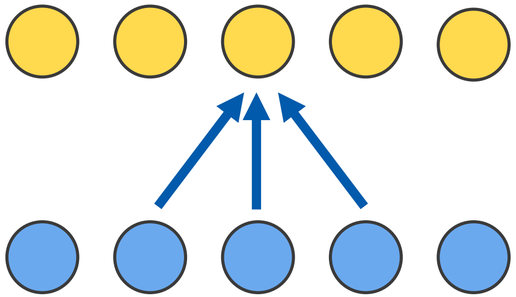
Previous architectures

Fully-connected



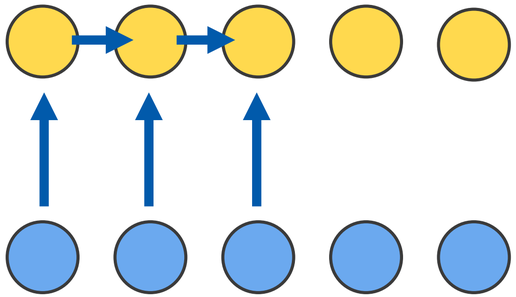
- Fixed window
- A lot of weights

Convolutional



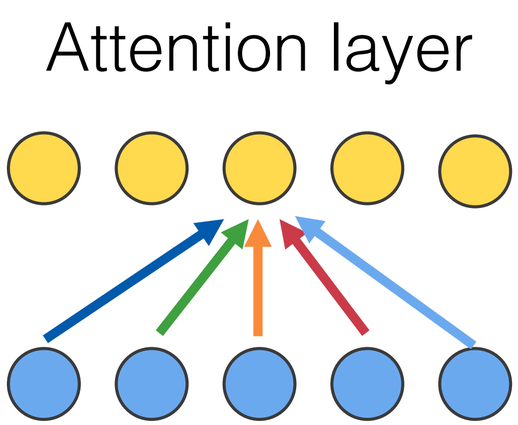
- Local dependencies

Recurrent



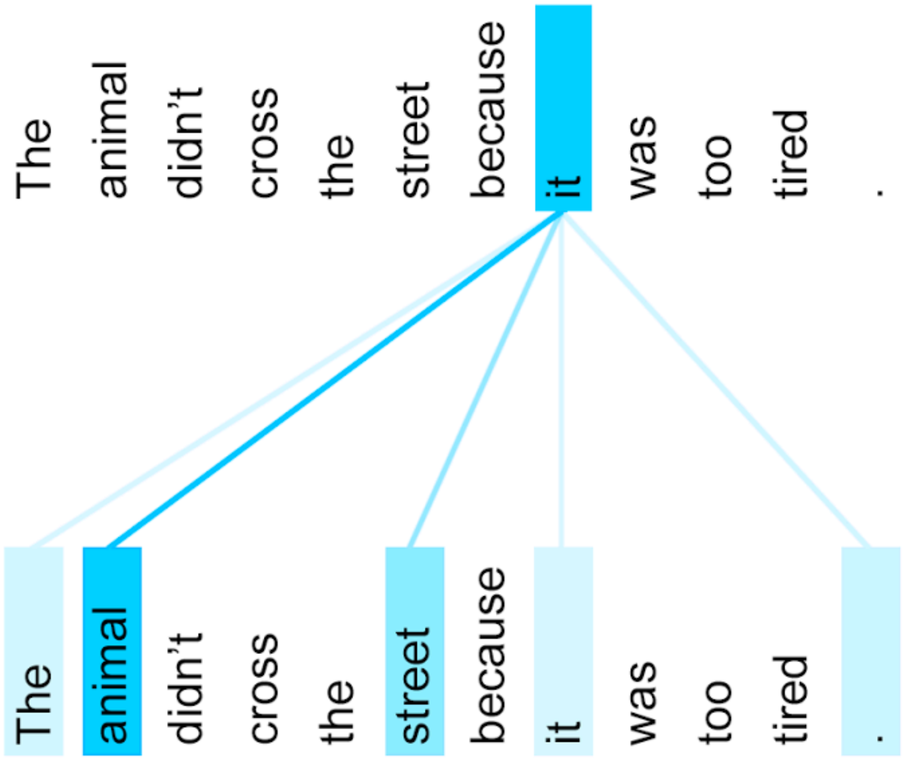
- Slow training (autoregressive)
- Troubles with gradients

Attention layer



- Look at all inputs to learn more global dependencies
- Share weights to work with variable length window
- Do not use autoregressive structure to make training parallelizable

Self-attention - idea



We update an embedding of each token using all input tokens:

- Find tokens relevant to the current one (compute relevance scores)
- Compute the weighted combination of their embeddings

https://ai.googleblog.com/2017/08/transformer-novel-neural-network.html

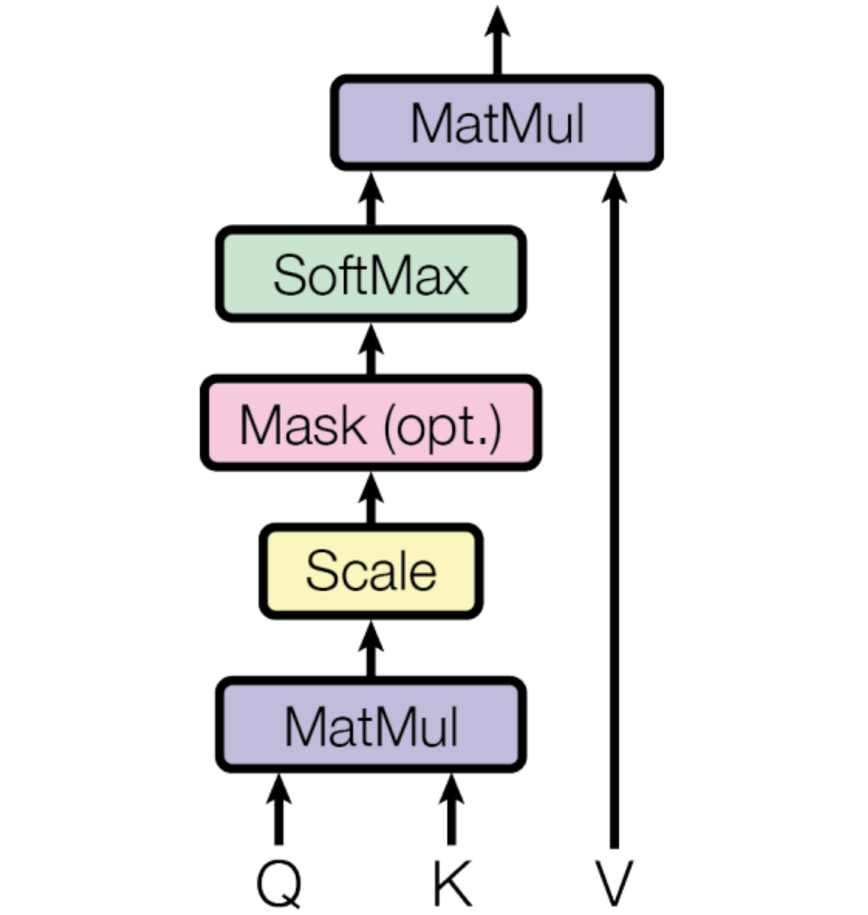
Self-attention

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

To compute relevance scores we use two linear transformations of the input:

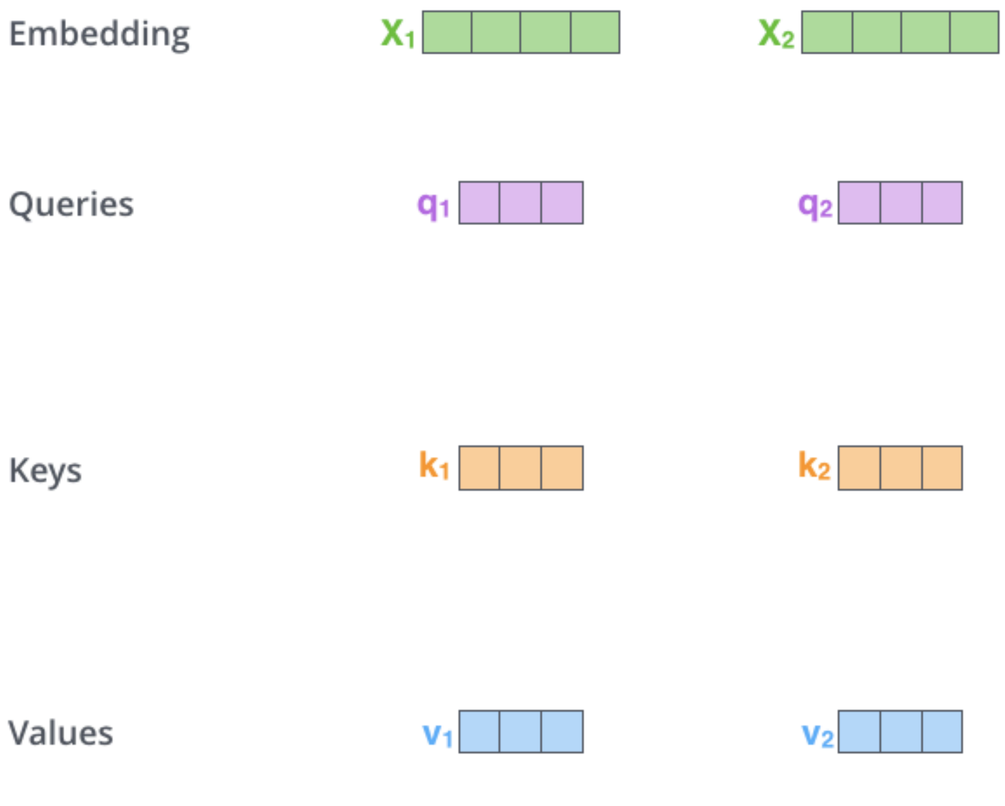
- Q - queries
- K - keys

And to compute final embeddings we use the third one - V - values



https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

Self-attention step by step



W^Q, W^K, W^V – trainable parameters

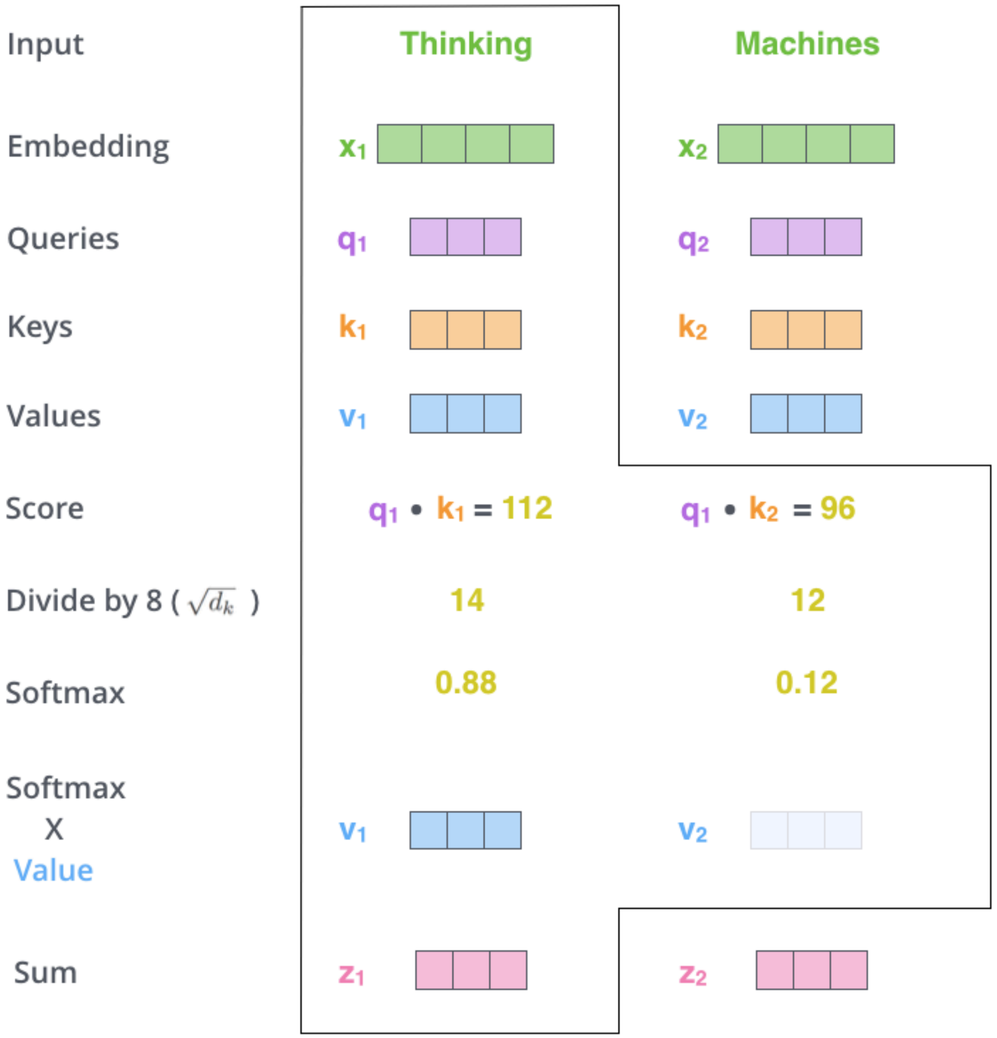
$$q_i = x_i W^Q$$

$$k_i = x_i W^K$$

$$v_i = x_i W^V$$

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

Self-attention step by step



- Compute queries, keys, values for all inputs using the same linear transformations

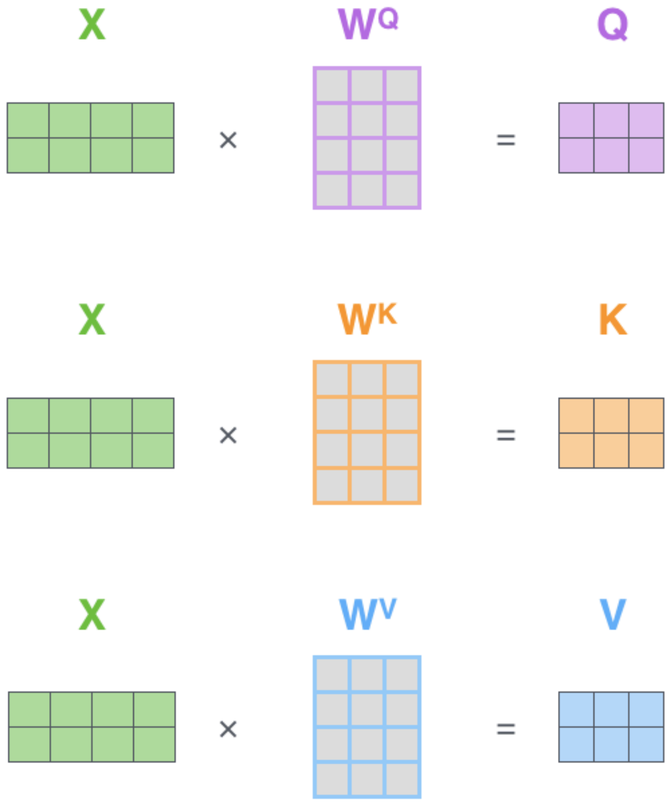
For token i:

- Compute relevance scores using query i and all the keys
- Compute weighted combination of values, weights from relevance scores

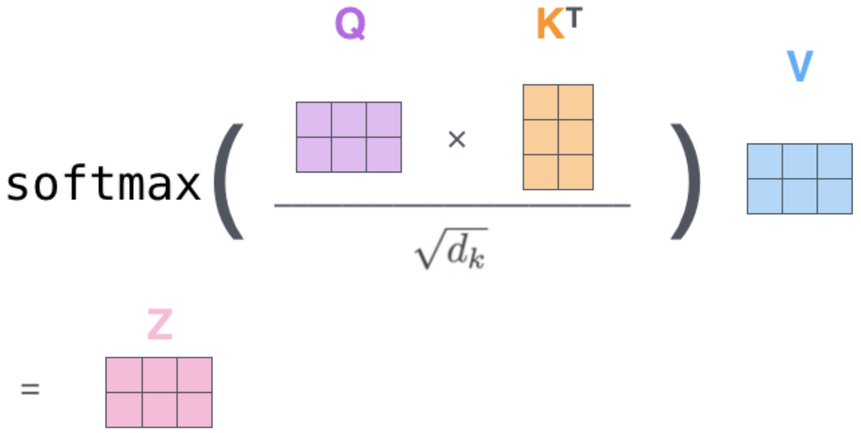
https://jalammar.github.io/illustrated-transformer/

Self-attention in matrix form

Step 1



Step 2



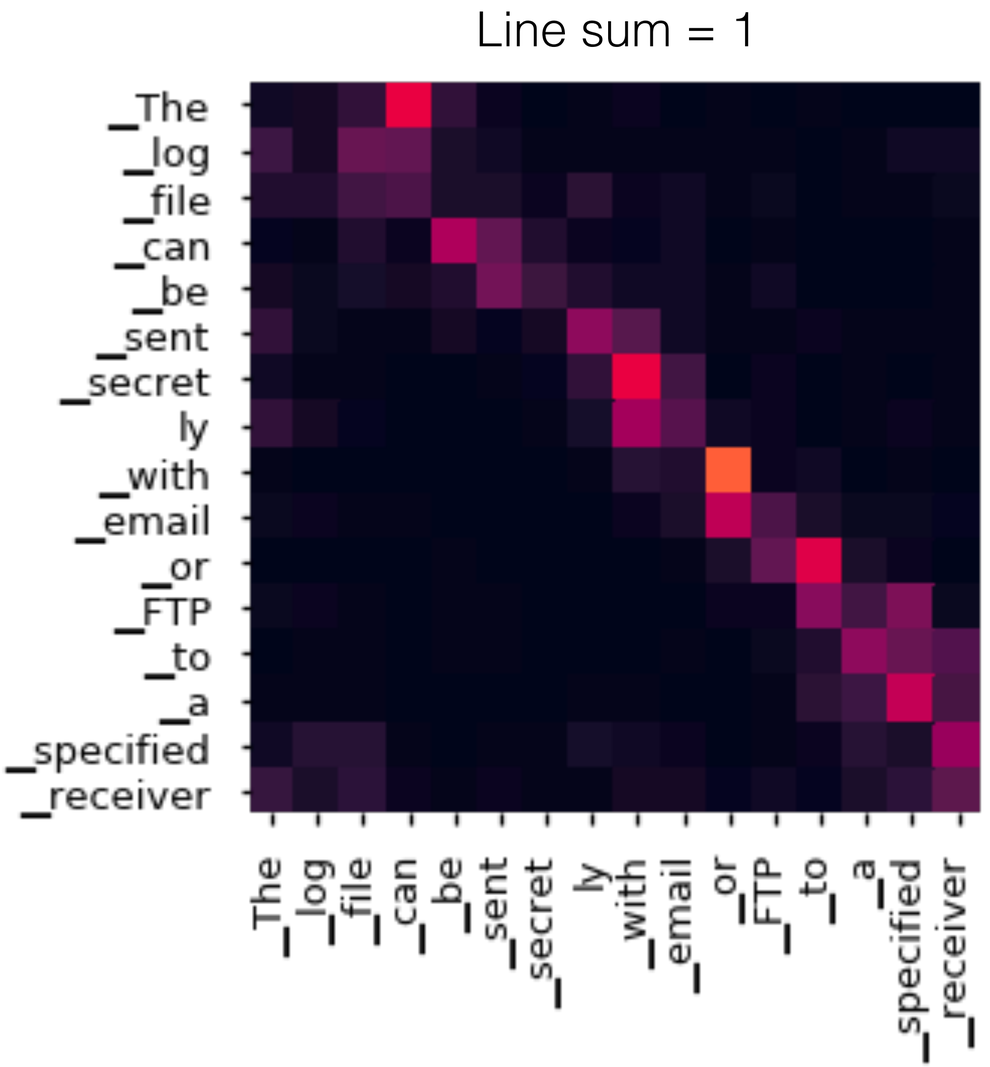
$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

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

Attention matrix

Scaled dot-product attention

$$\text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)$$



https://nlp.seas.harvard.edu/2018/04/03/attention.html

Other score functions

Name	Alignment score function	Citation
Content-base attention	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \text{cosine}[\boldsymbol{s}_t, \boldsymbol{h}_i]$	Graves2014
Additive(*)	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \mathbf{v}_a^\top \tanh(\mathbf{W}_a[\boldsymbol{s}_t; \boldsymbol{h}_i])$	Bahdanau2015
Location-Base	$\alpha_{t,i} = \text{softmax}(\mathbf{W}_a \boldsymbol{s}_t)$ Note: This simplifies the softmax alignment to only depend on the target position.	Luong2015
General	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \boldsymbol{s}_t^\top \mathbf{W}_a \boldsymbol{h}_i$ where $\mathbf{W}_a$ is a trainable weight matrix in the attention layer.	Luong2015
Dot-Product	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \boldsymbol{s}_t^\top \boldsymbol{h}_i$	Luong2015
Scaled Dot-Product(^)	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \frac{\boldsymbol{s}_t^\top \boldsymbol{h}_i}{\sqrt{n}}$ Note: very similar to the dot-product attention except for a scaling factor; where n is the dimension of the source hidden state.	Vaswani2017

https://lilianweng.github.io/lil-log/2018/06/24/attention-attention.html#whats-wrong-with-seq2seq-model

Complexity

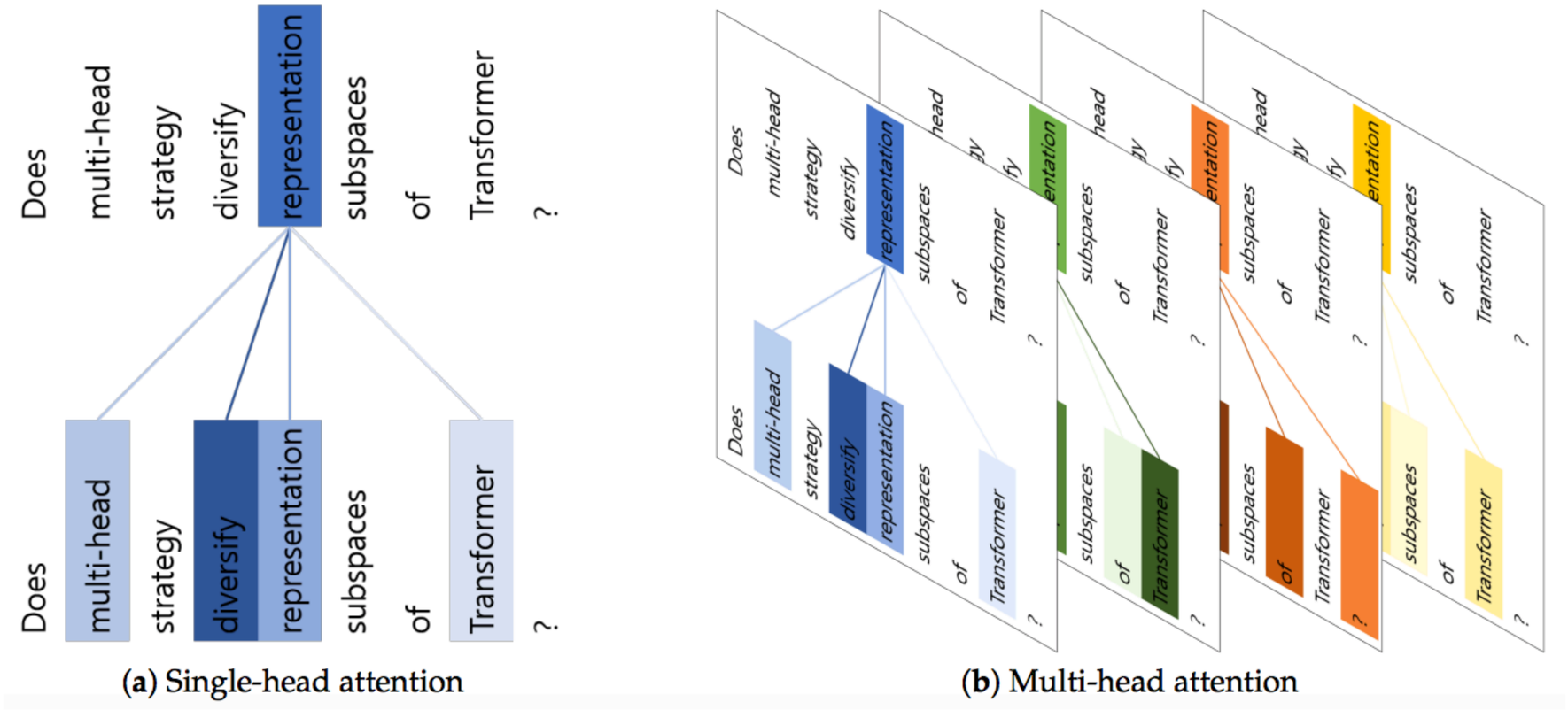
Table 1: Maximum path lengths, per-layer complexity and minimum number of sequential operations for different layer types. n is the sequence length, d is the representation dimension, k is the kernel size of convolutions and r the size of the neighborhood in restricted self-attention.

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

https://nlp.seas.harvard.edu/2018/04/03/attention.html

Multi-Head Attention - idea

Want to consider different types of relevance

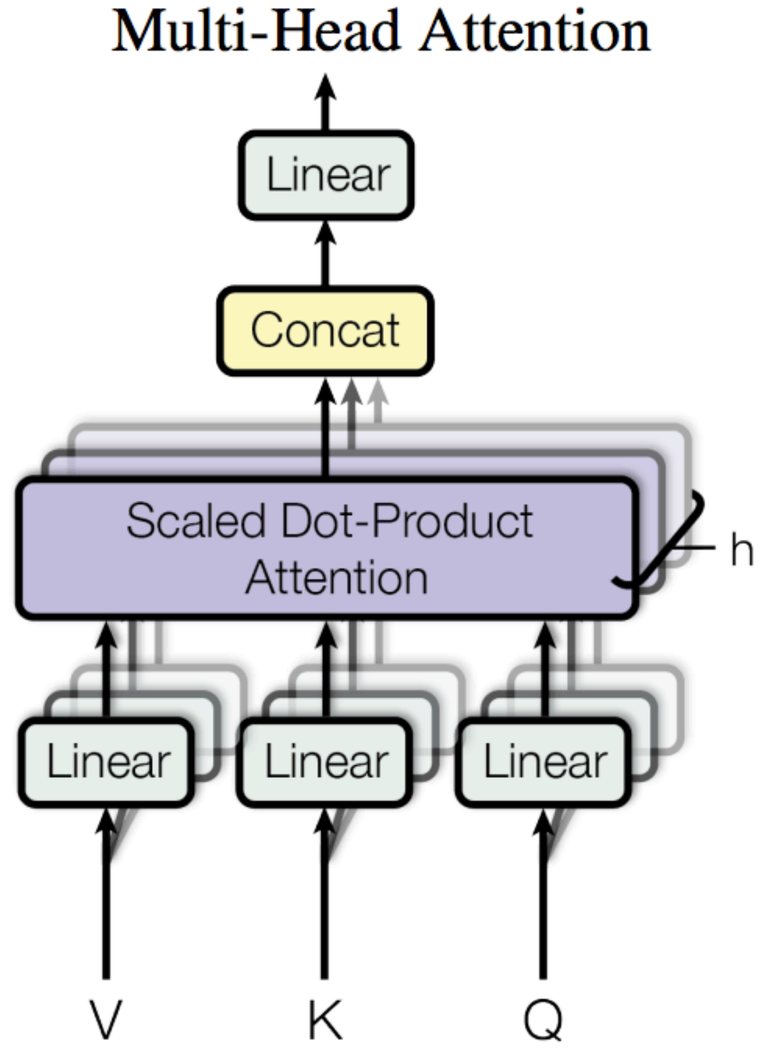


https://www.mdpi.com/2076-3417/11/4/1548

Multi-Head Attention

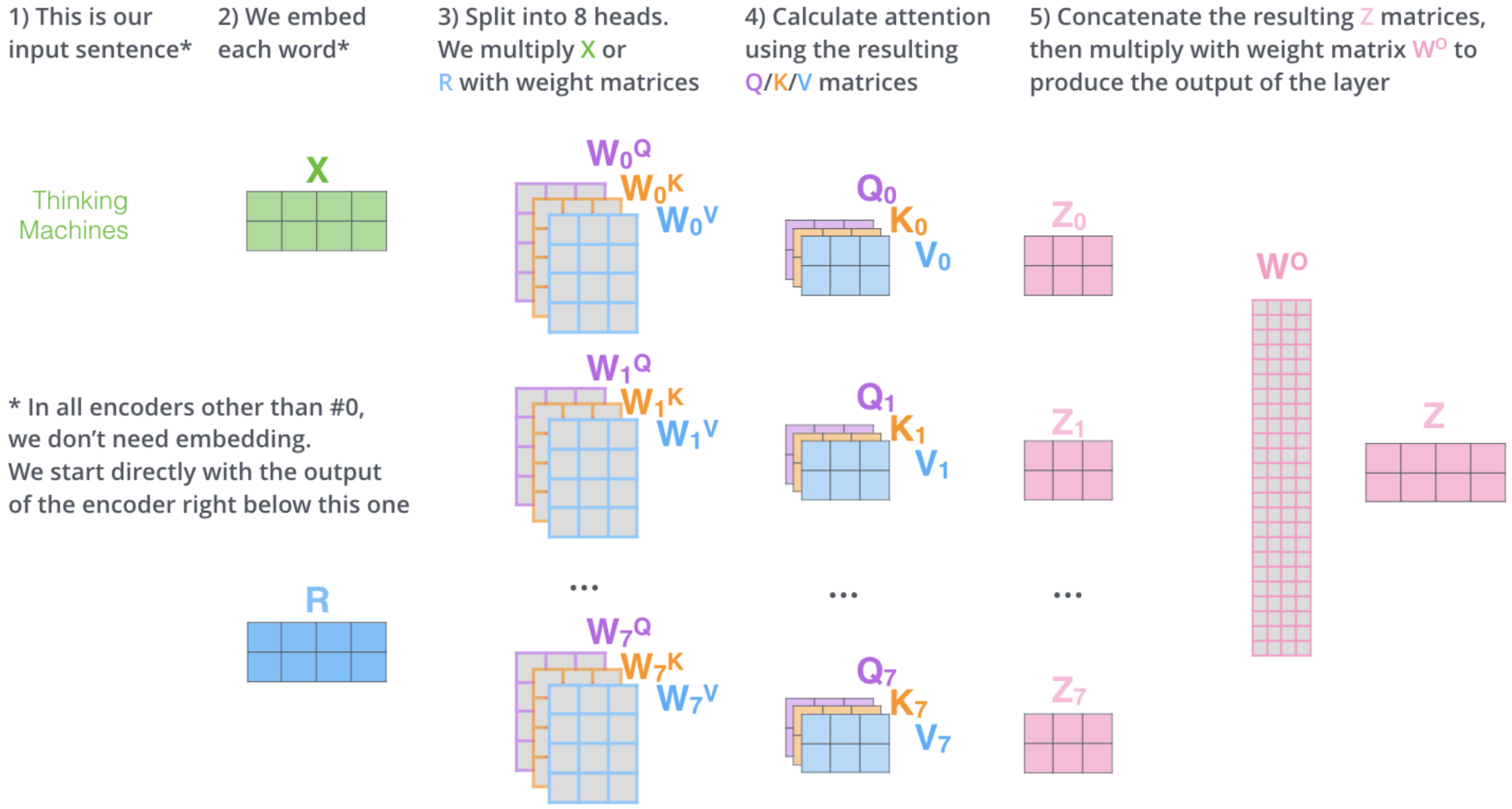
$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O$$
$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

Use several attention heads - each will learn its own type of relevance



https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

Multi-Head Attention - step by step



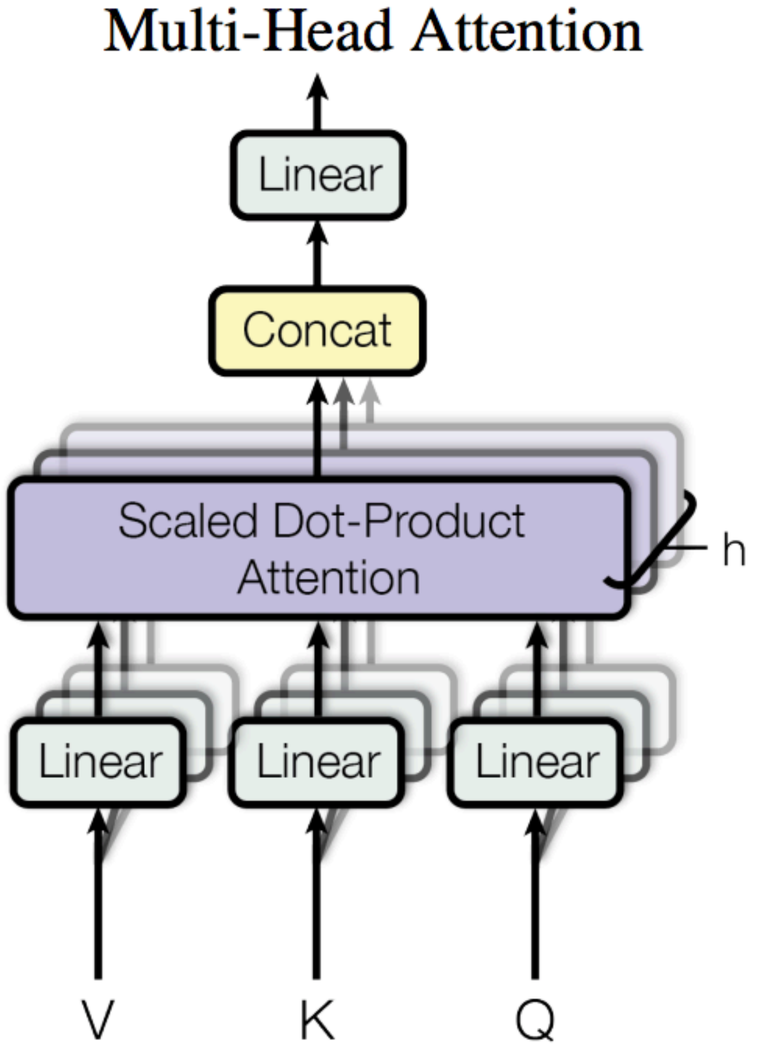
https://jalammar.github.io/illustrated-transformer/

Multi-Head Attention

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O$$
$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

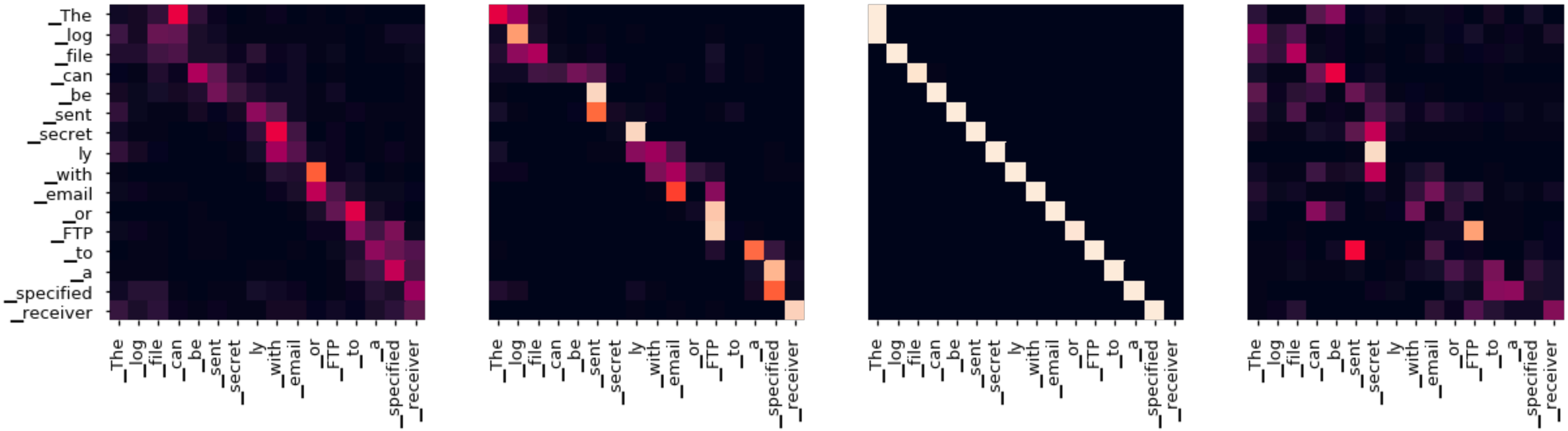
Trainable parameters:

$$W_i^Q \in \mathbb{R}^{d_{model} \times d_k}$$
$$W_i^K \in \mathbb{R}^{d_{model} \times d_k}$$
$$W_i^V \in \mathbb{R}^{d_{model} \times d_v}$$
$$W^O \in \mathbb{R}^{hd_v \times d_{model}}$$
$$d_k = d_v = d_{model}/h$$



https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

Attention matrices



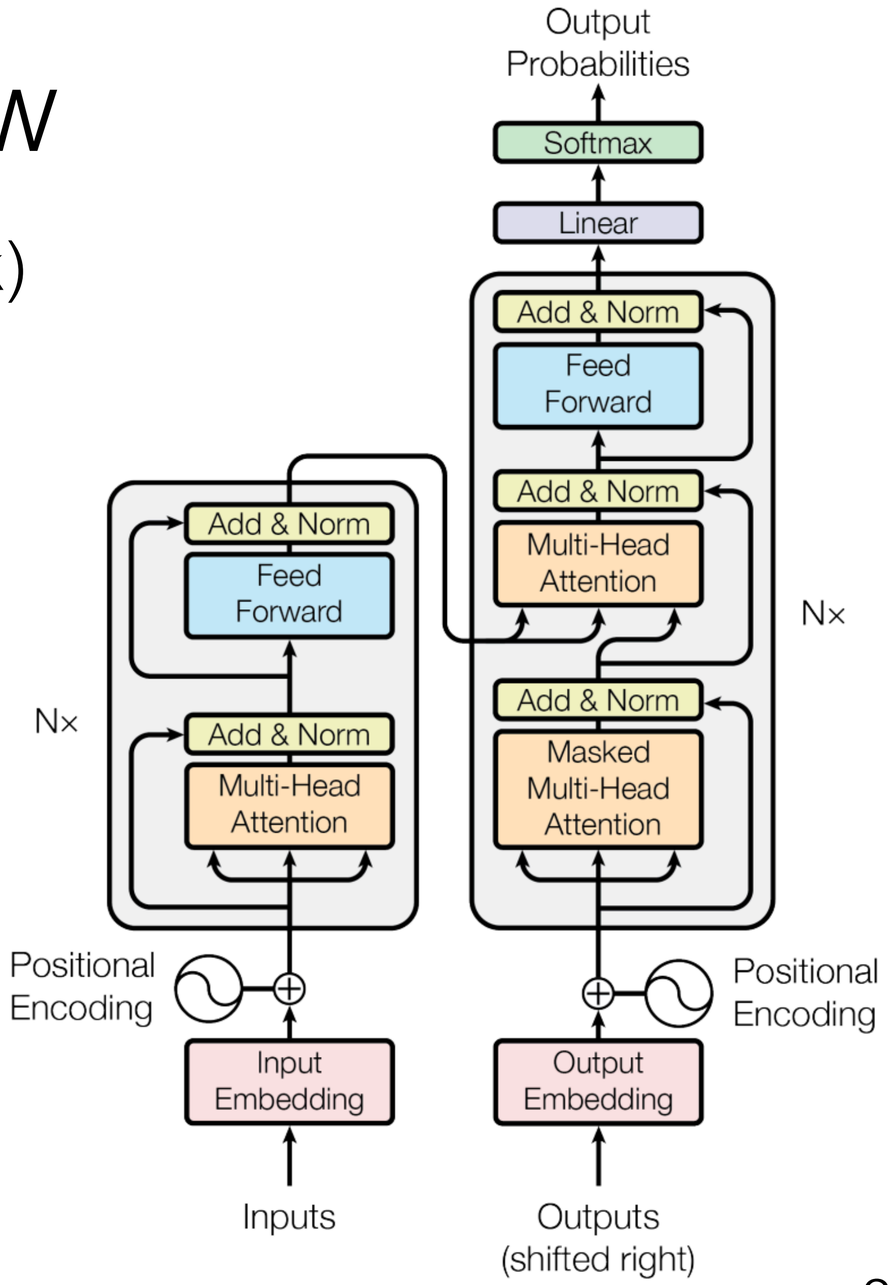
https://nlp.seas.harvard.edu/2018/04/03/attention.html

Transformer Model

Transformer model overview

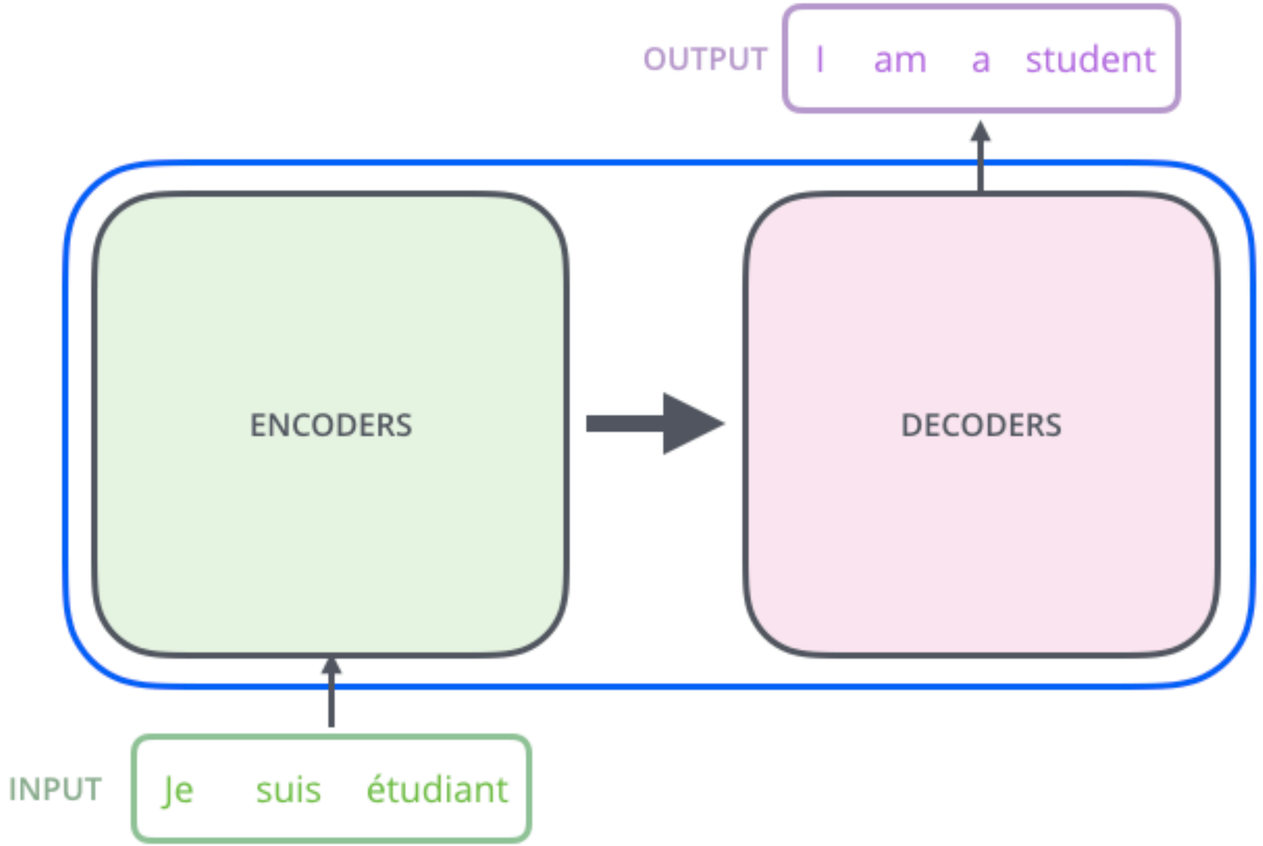
Task - neural machine translation (Seq2seq task)

- Encoder-decoder architecture
- 3 types of attention operations:
  - Encoder-encoder (self-attention)
  - Decoder-decoder (self-attention)
  - Encoder-decoder
- Other operations (position-wise):
  - Feed forward layers
  - Residual connections
  - Layer norm
- Positional encoding



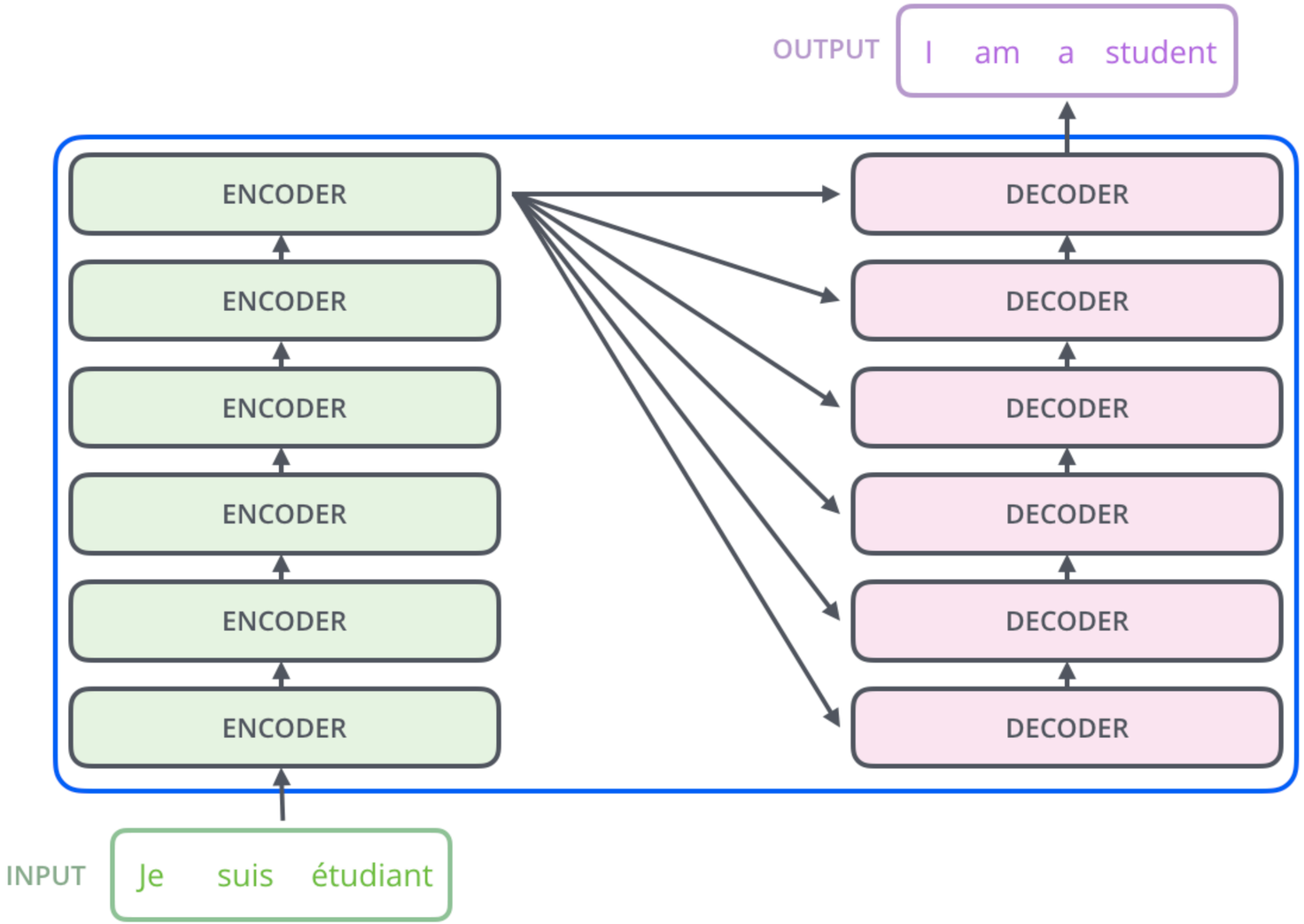
https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

Transformer: encoder-decoder



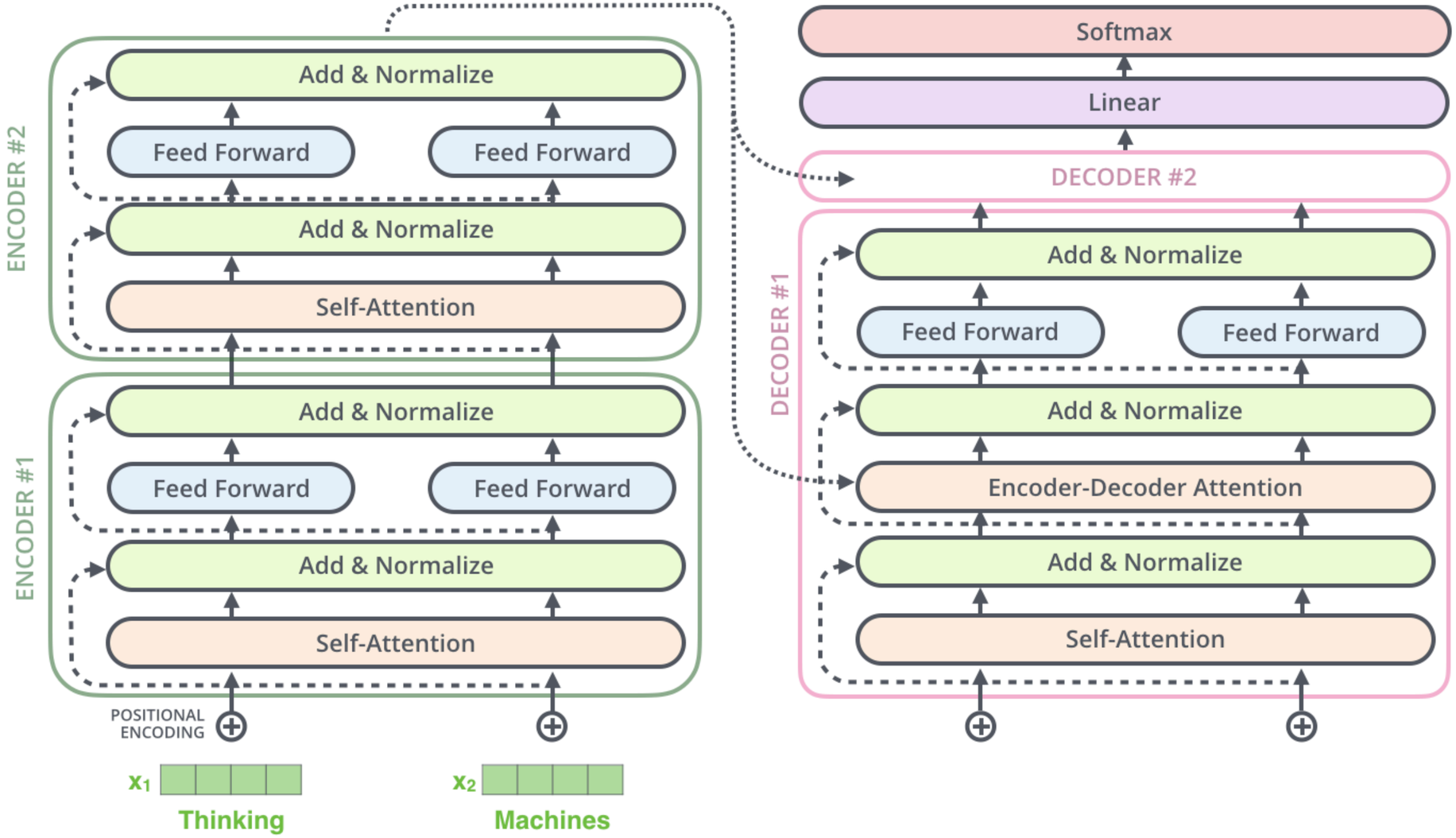
https://jalammar.github.io/illustrated-transformer/

Transformer: encoder-decoder



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

Transformer: encoder-decoder



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

Encoder-encoder attention

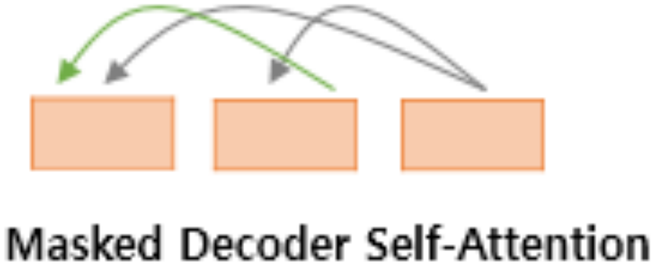
The one we have discussed earlier:

- Multi-head
- Scaled dot-product
- Self-attention: Q, K, V are computed from the same input matrix X

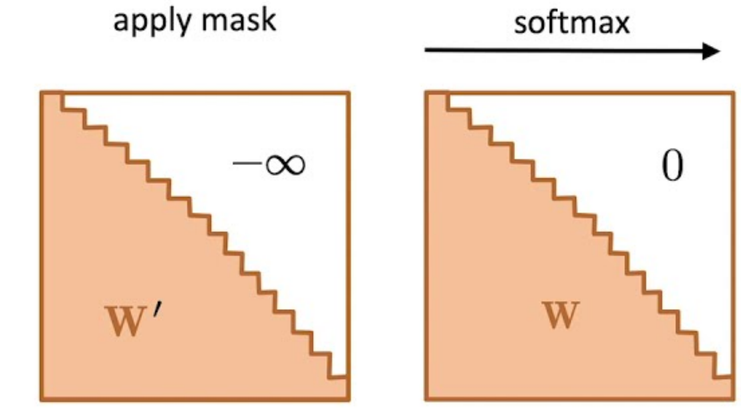
Update embeddings of tokens from the input sequence

https://wikidocs.net/31379

Decoder-decoder attention



We need mask attention matrix to not look at future tokens during training:

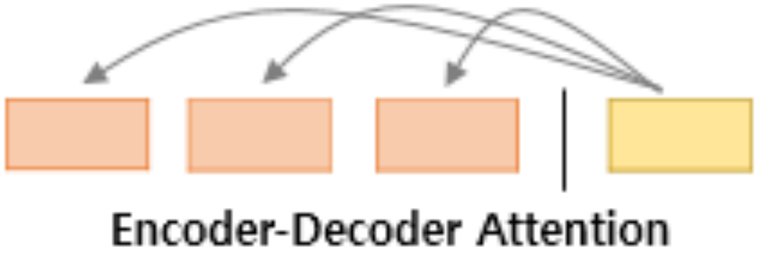


$$\mathbf{W}'_{ij} \leftarrow -\infty \text{ if } j > i \qquad \mathbf{W} = \text{softmax}(\mathbf{W}')$$

Attend to the previous words in the generated output sequence

https://wikidocs.net/31379
https://www.youtube.com/watch?v=oUhGZMCTHtI

Encoder-decoder attention



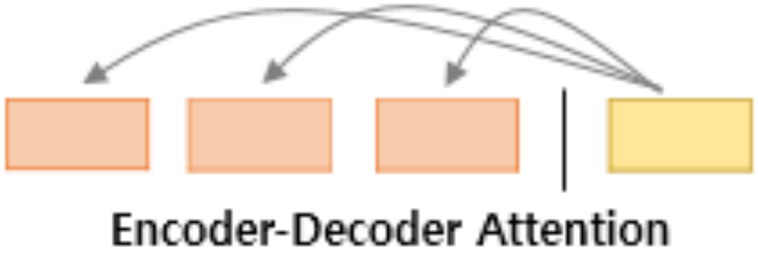
This is not self-attention:

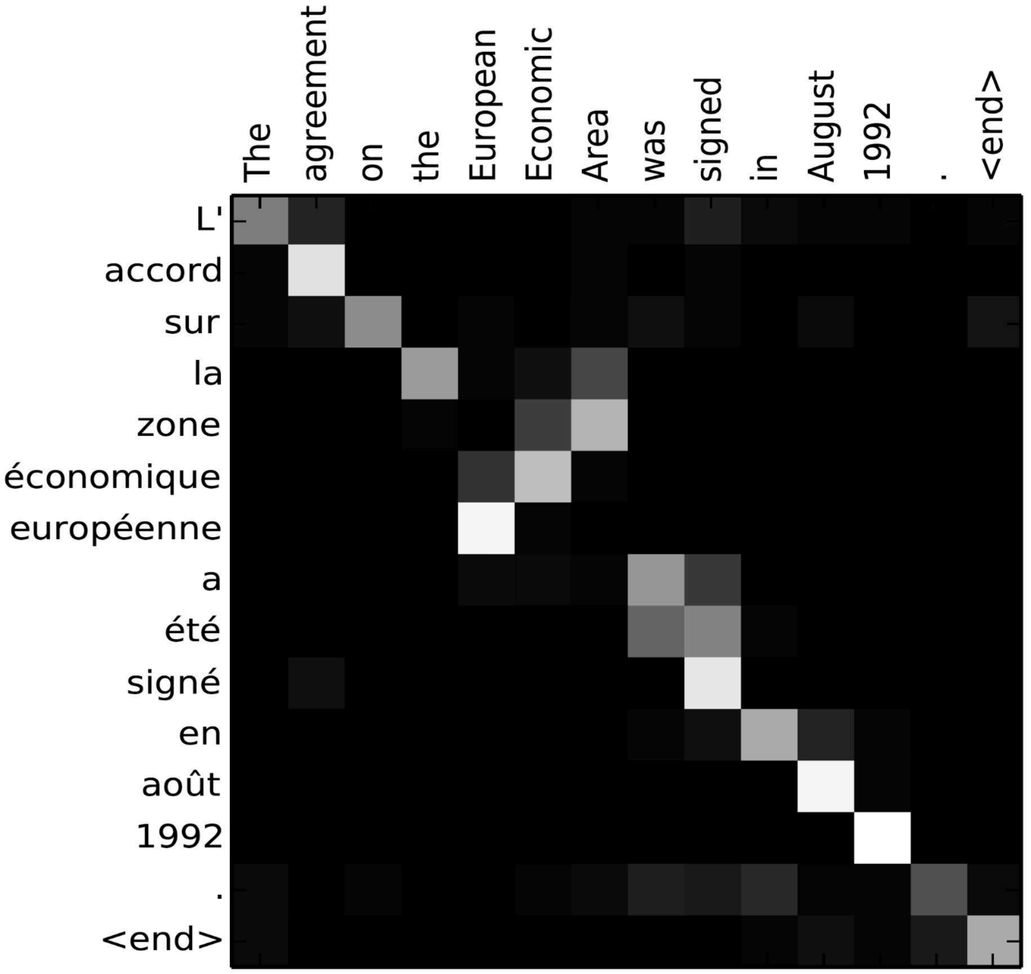
- Q from decoder
- K, V from encoder

Attend to tokens from the input sequence relevant for the generation of the next output token

https://wikidocs.net/31379

Encoder-decoder attention

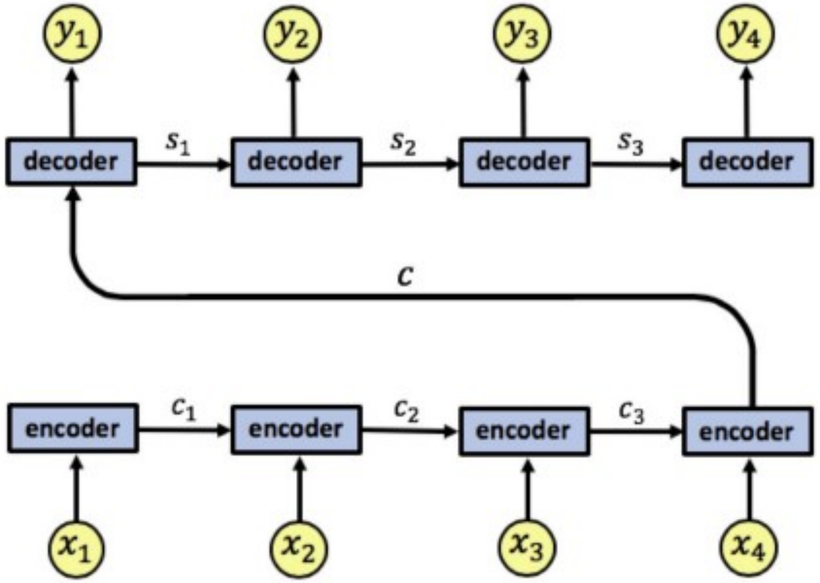




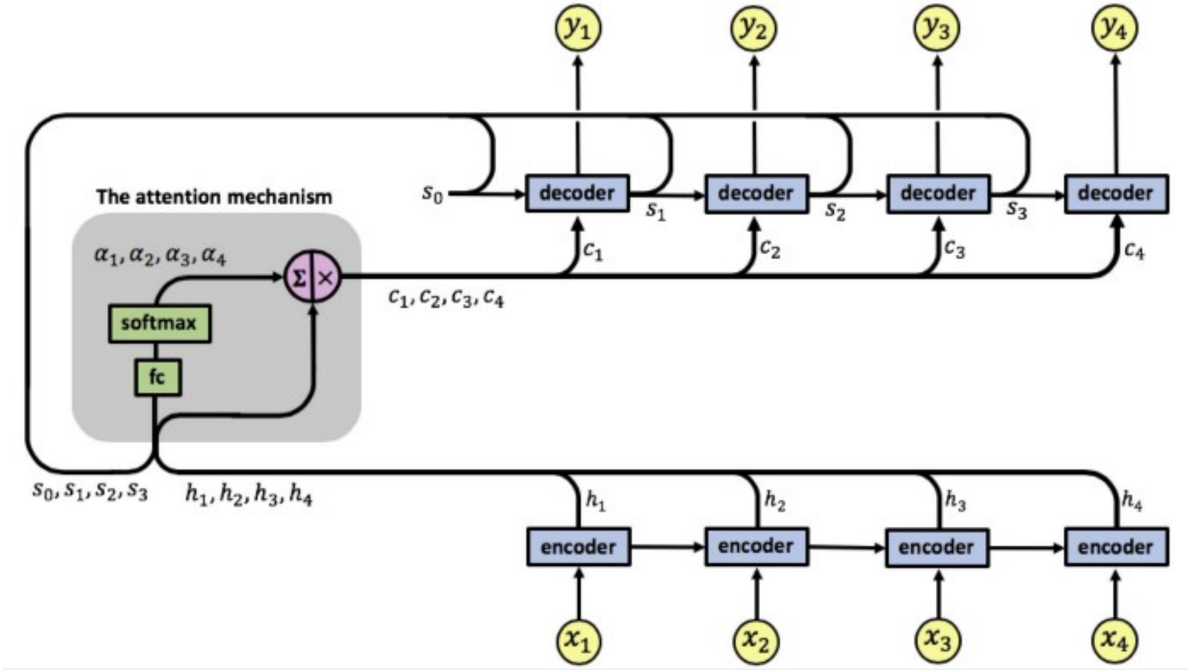
https://wikidocs.net/31379

Encoder-decoder attention in RNNs

Seq2seq without attention

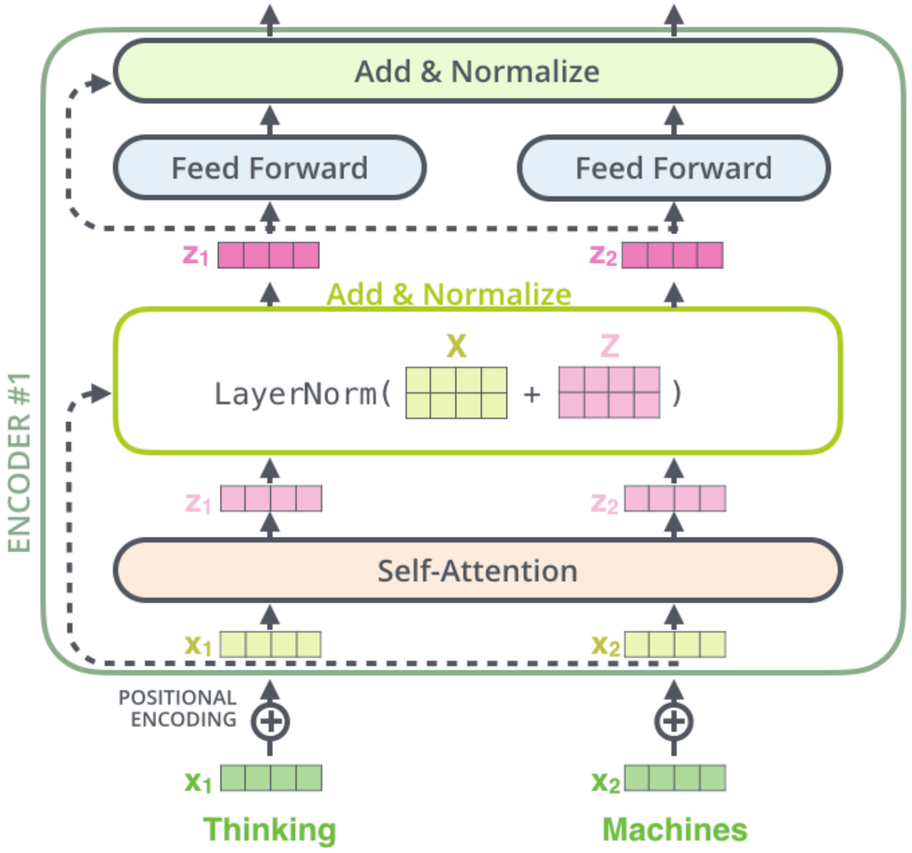


Seq2seq with attention



https://medium.datadriveninvestor.com/attention-in-rnns-321fbcd64f05

Other operations

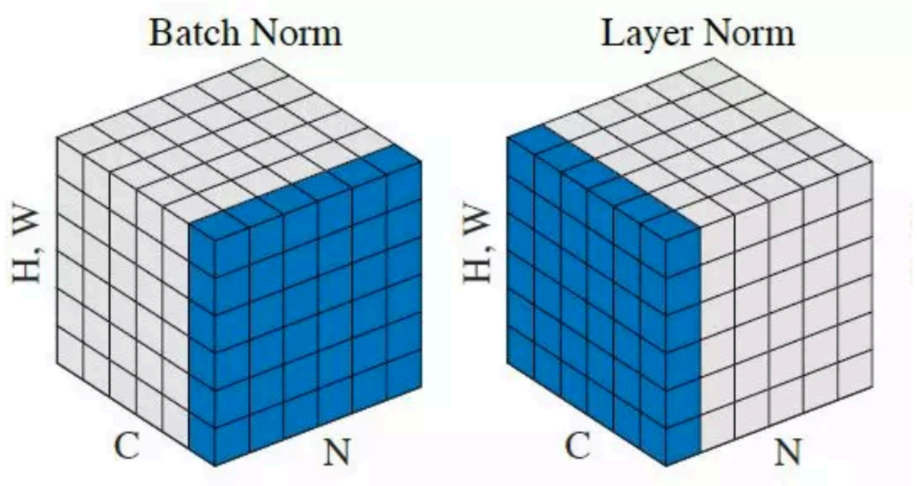


ALL other operations are position-wise:

- Feed forward layers

$$\mathrm{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

- Residual connections
- Layer norm

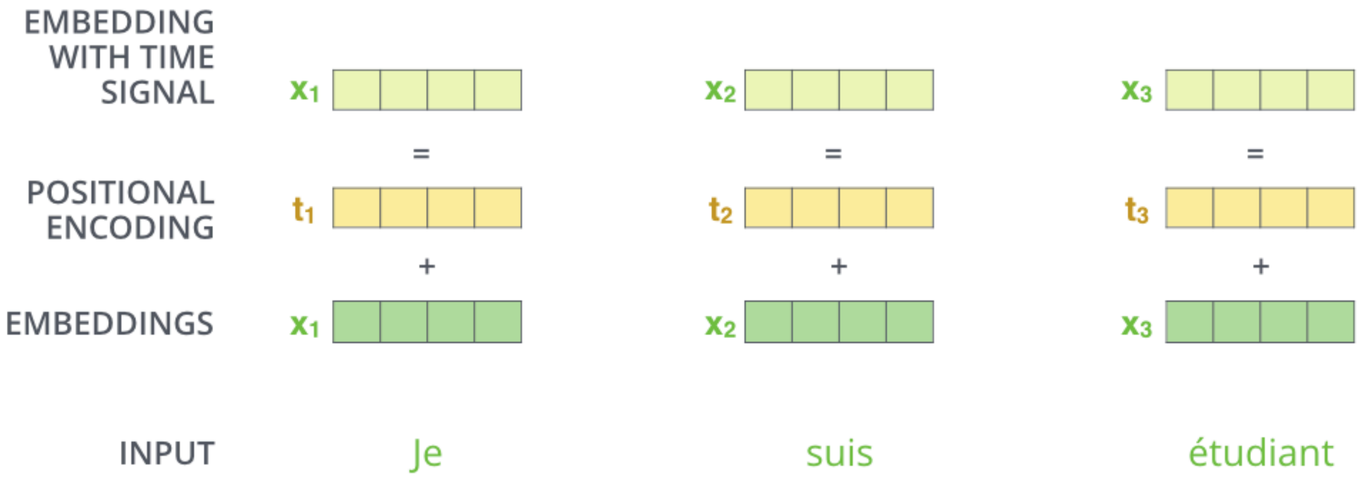


http://proceedings.mlr.press/v119/shen20e/shen20e.pdf
https://jalammar.github.io/illustrated-transformer/

Positional encoding

Attention and position-wise feed forward layers look at the input sequence as a bag of tokens

We need to encode positional information into the input embeddings



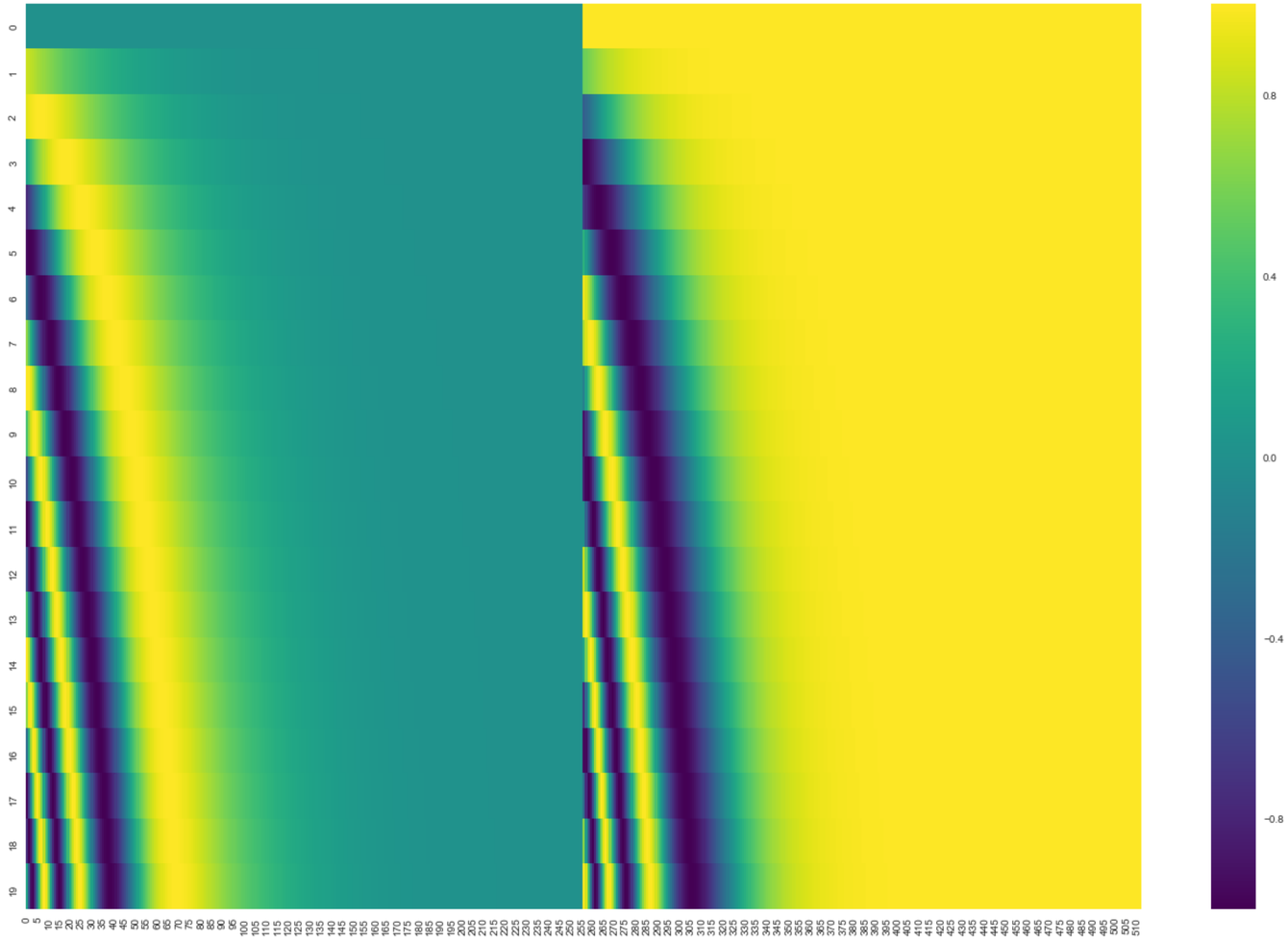
https://jalammar.github.io/illustrated-transformer/

Positional encoding

Variant from the initial paper:

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

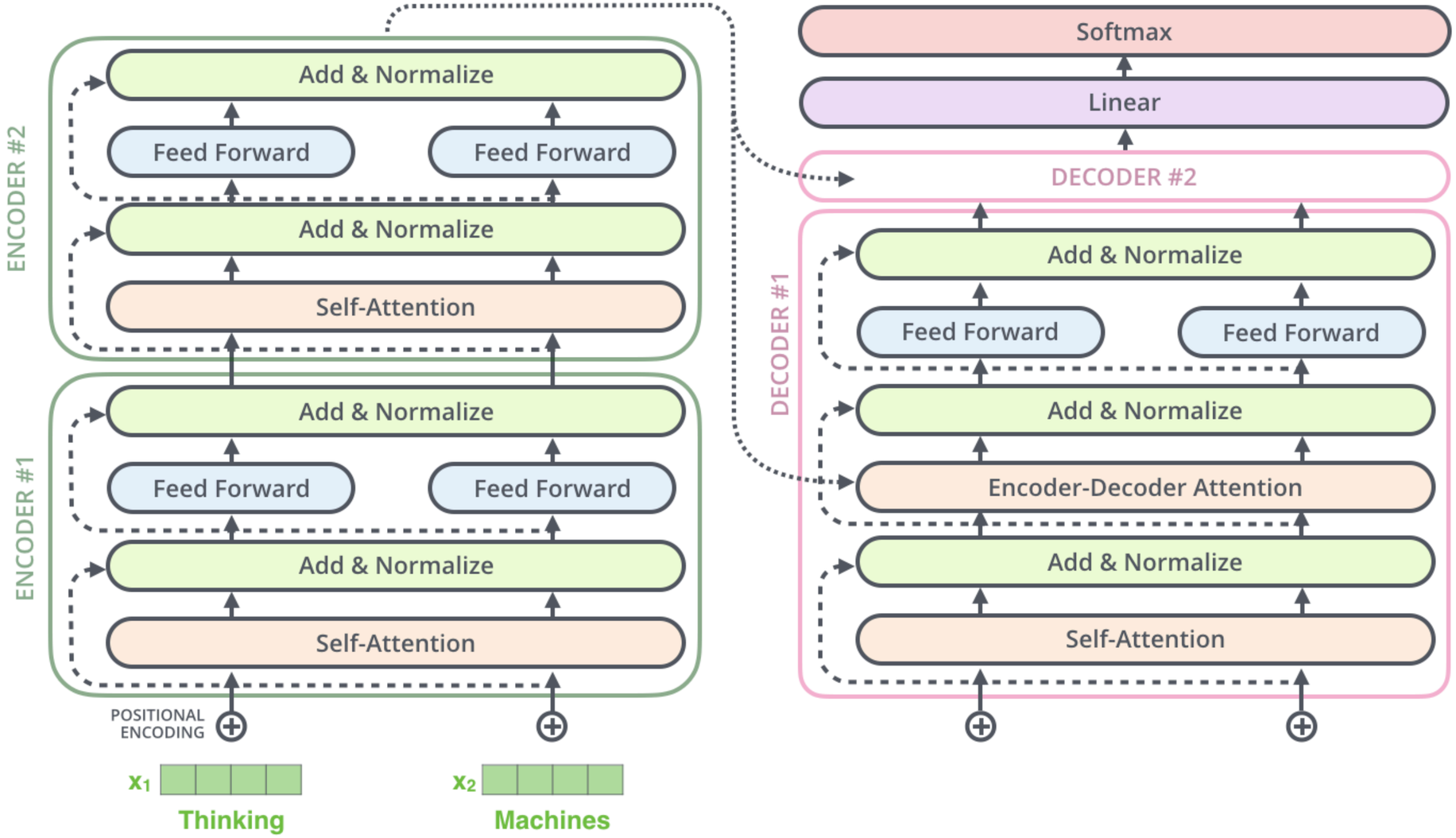
Other common variant - trainable vectors



A real example of positional encoding for 20 words (rows) with an embedding size of 512 (columns). You can see that it appears split in half down the center. That's because the values of the left half are generated by one function (which uses sine), and the right half is generated by another function (which uses cosine). They're then concatenated to form each of the positional encoding vectors.

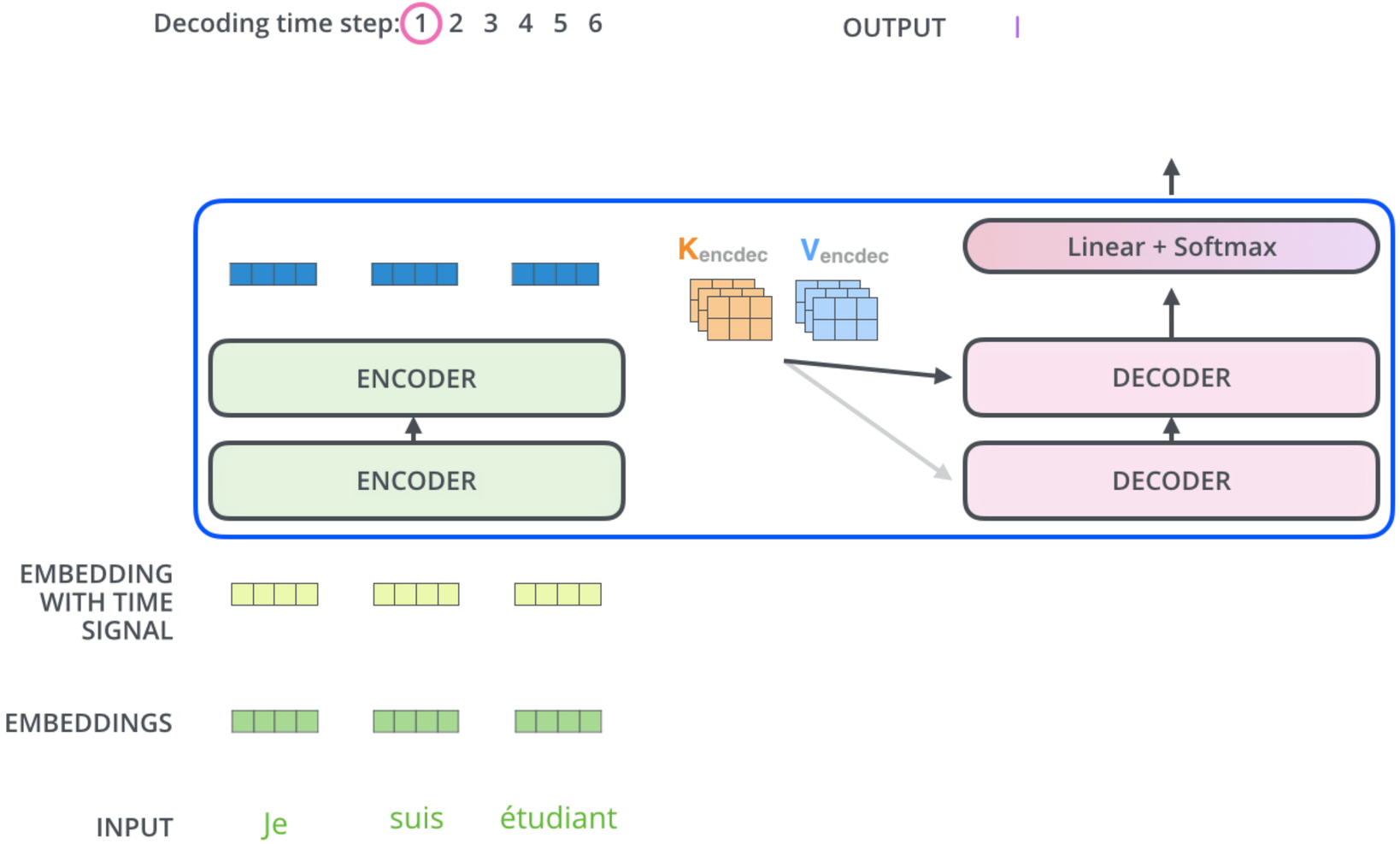
https://jalammar.github.io/illustrated-transformer/
https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

Transformer: encoder-decoder



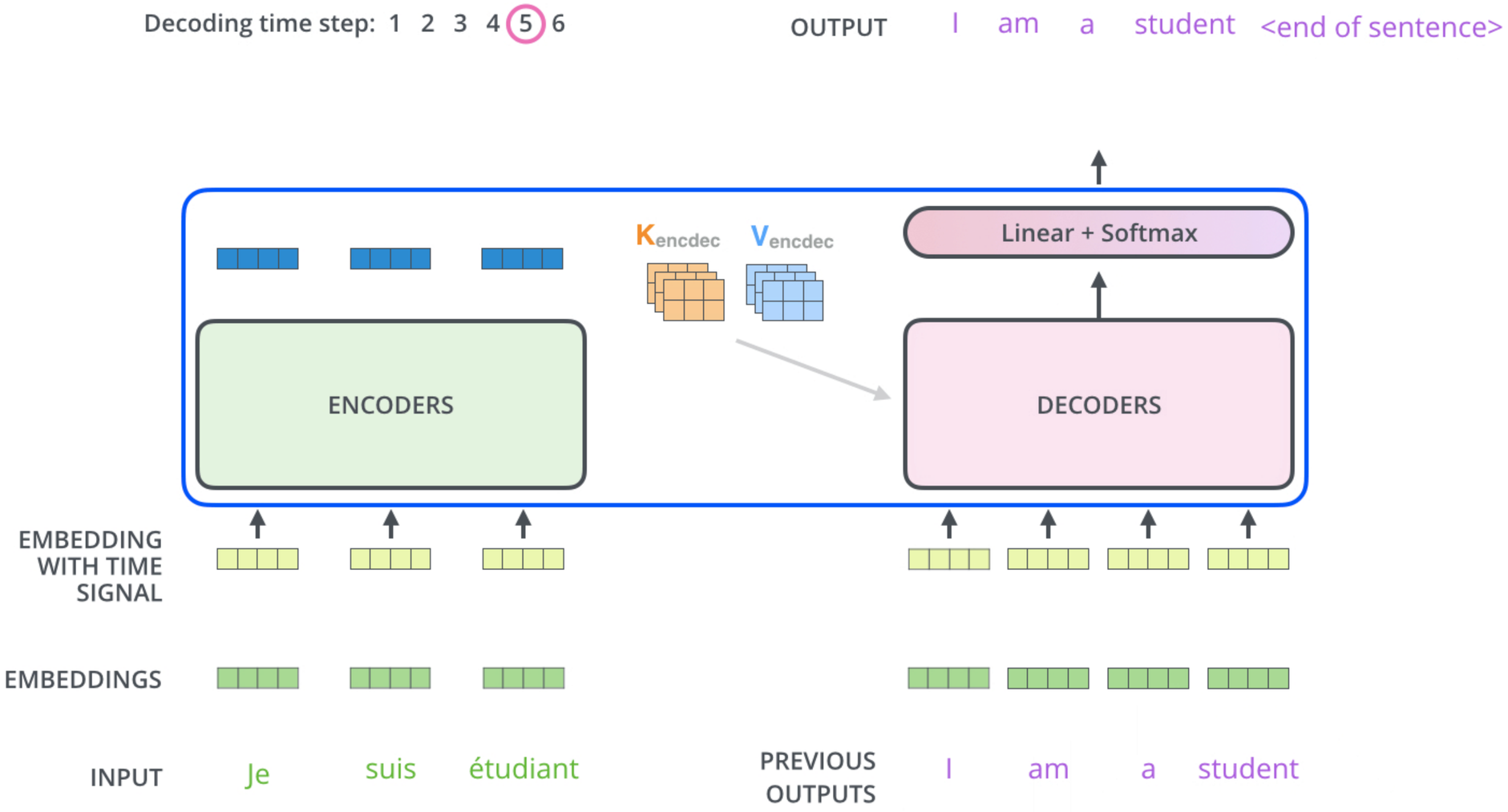
https://jalammar.github.io/illustrated-transformer/

Transformer: encoder-decoder



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

Transformer: encoder-decoder



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

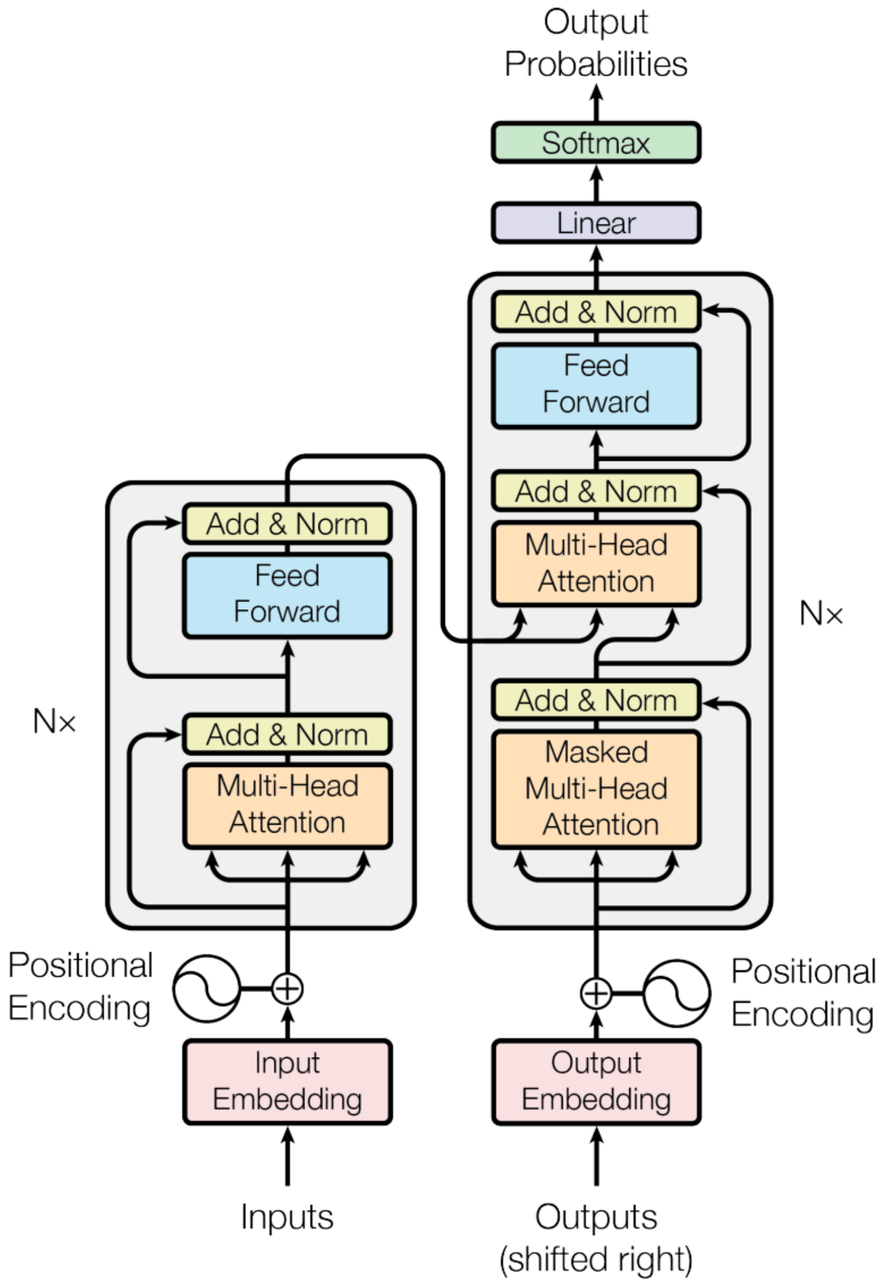
Transformer size

- Number of layers (blocks) in encoder/ decoder
- Hidden dimensionality (d_model)
- Number of heads
- Dimensionality of inner feed-forward layers (usually 4 x hidden dimensionality)

Standard model:

6+6 layers, H = 512, 8 heads

65M params



https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

Training details

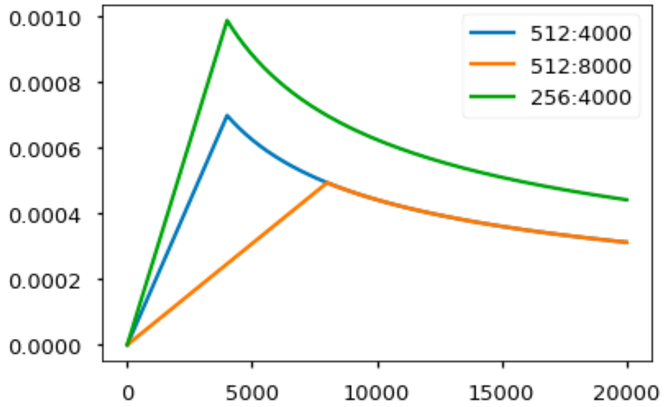
- Loss - standard NLL (cross-entropy) - lower is better:

$$NLL(y_{1:M}) = -\sum_{t=1}^{M} \log p(y_t|t_{t-1})$$

Instead perplexity is usually reported - higher is better:

$$Perplexity(y_{1:M}) = 2^{\frac{1}{M} NLL(y_{1:M})}$$

lr schedule



- Teacher forcing for decoder
- Adam optimizer
- Learning rate schedule with warm-up:

$$lr = d_{model}^{-0.5} \cdot \min(step\_num^{-0.5}, step\_num \cdot warmup^{-1.5})$$

https://nlp.seas.harvard.edu/2018/04/03/attention.html

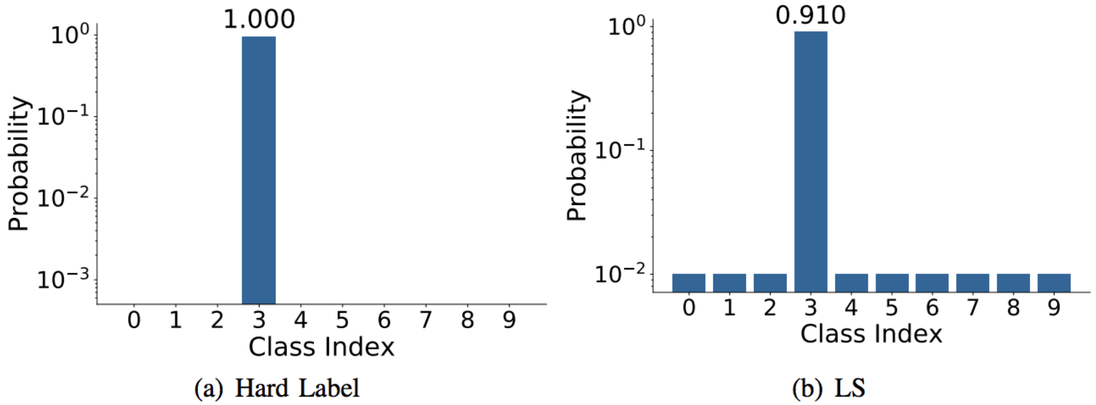
Training details

- Label smoothing

$$y_{ls} = (1 - \alpha) \cdot y_{hot} + \alpha / K$$

- Residual dropout - to the output of each sub-layer (before add+norm) + to the sums of the embeddings and the positional encodings
- BPE/Word-piece (shared embeddings for input/output)
- Model averaging (average last k checkpoints - SWA)

label smoothing



https://arxiv.org/pdf/2011.12562.pdf SWA: https://arxiv.org/abs/1803.05407

Output generation: greedy search

How to find the most probable output sequence $y_1, \ldots, y_T$?

Greedy search: $y_t = \text{argmax}_{y \in \mathcal{Y}} P(y|y_1, \ldots, y_{t-1}, C)$

Result of greedy search

Time step	1	2	3	4
A	0.5	0.1	0.2	0.0
B	0.2	0.4	0.2	0.2
C	0.2	0.3	0.4	0.2
<eos>	0.1	0.2	0.2	0.6

$P = 0.5 \times 0.4 \times 0.4 \times 0.6 = 0.048$

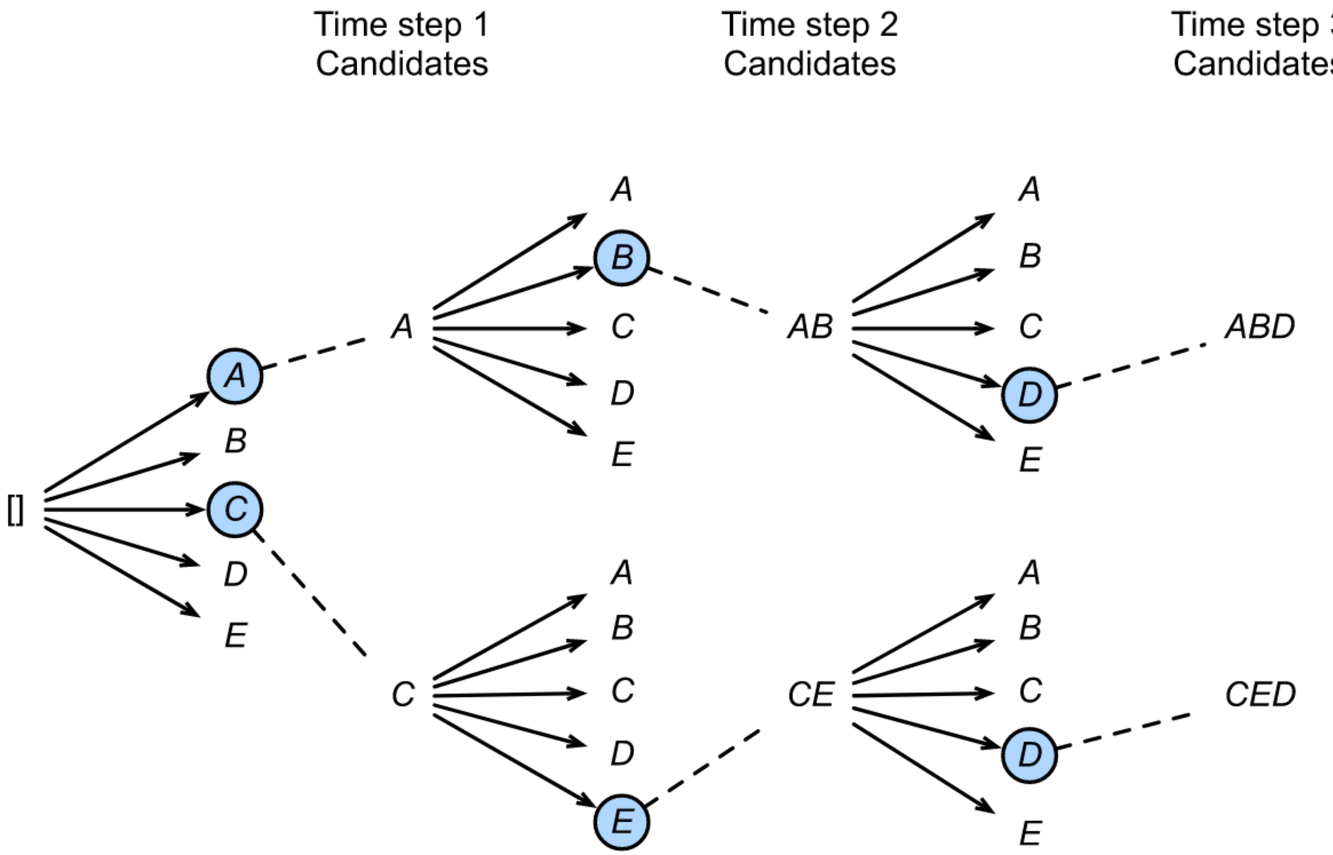
Better output sequence

Time step	1	2	3	4
A	0.5	0.1	0.1	0.1
B	0.2	0.4	0.6	0.2
C	0.2	0.3	0.2	0.1
<eos>	0.1	0.2	0.1	0.6

$P = 0.5 \times 0.3 \times 0.6 \times 0.6 = 0.054$

https://d2l.ai/chapter_recurrent-modern/beam-search.html

Output generation: beam search



- At each step choose k best candidates (k=~5)
- Stop when k candidates with <END> token have been generated
- Choose the best one:

$$\frac{1}{L^{\alpha}} \log P(y_1, \ldots, y_L | C)$$

length penalty, alpha = 0.75

https://d2l.ai/chapter_recurrent-modern/beam-search.html

BLEU metric for NMT

Compare NMT output and reference translation based of the n-gram overlap:

N-gram precision: $$p_n = \frac{\sum_{n\text{-}gram \in hyp} count_{clip}(n\text{-}gram)}{\sum_{n\text{-}gram \in hyp} count(n\text{-}gram)}$$

n=1-4

$$count_{clip} = min(count, max\_ref\_count)$$

Brevity penalty: $$BP = \begin{cases} e^{(1-|ref|/|hyp|)} & if\ |ref| > |hyp| \\ 1 & otherwise \end{cases}$$

Result (higher is better): $$BLEU = BP \cdot \exp\left[\frac{1}{N}\sum_{n=1}^{N} \log p_n\right]$$

https://aclanthology.org/P02-1040.pdf

BLEU metric for NMT

Compare NMT output and reference translation based of the n-gram overlap:

N-gram precision: $$p_n = \frac{\sum_{n\text{-}gram \in hyp} count_{clip}(n\text{-}gram)}{\sum_{n\text{-}gram \in hyp} count(n\text{-}gram)}$$

n=1-4

$$count_{clip} = min(count, max\_ref\_count)$$

Brevity penalty: $$BP = \begin{cases} e^{(1-|ref|/|hyp|)} & if\ |ref| > |hyp| \\ 1 & otherwise \end{cases}$$

Result (higher is better): $$\log BLEU = min\left(1 - \frac{|ref|}{|hyp|}, 0\right) + \frac{1}{N}\sum_{n=1}^{N} \log p_n$$

https://aclanthology.org/P02-1040.pdf

Results

Table 2: The Transformer achieves better BLEU scores than previous state-of-the-art models on the English-to-German and English-to-French newstest2014 tests at a fraction of the training cost.

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [15]	23.75			
Deep-Att + PosUnk [32]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [31]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [8]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [26]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [32]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [31]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [8]	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**	**41.0**	$2.3 \cdot 10^{19}$	

https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

What else?

- Larger receptive field, memory
- More efficient w.r.t. the sequence length
- Other tasks in NLP + CV, speech, graphs…

Contextualized Word Embeddings: ELMO, BERT, GPT

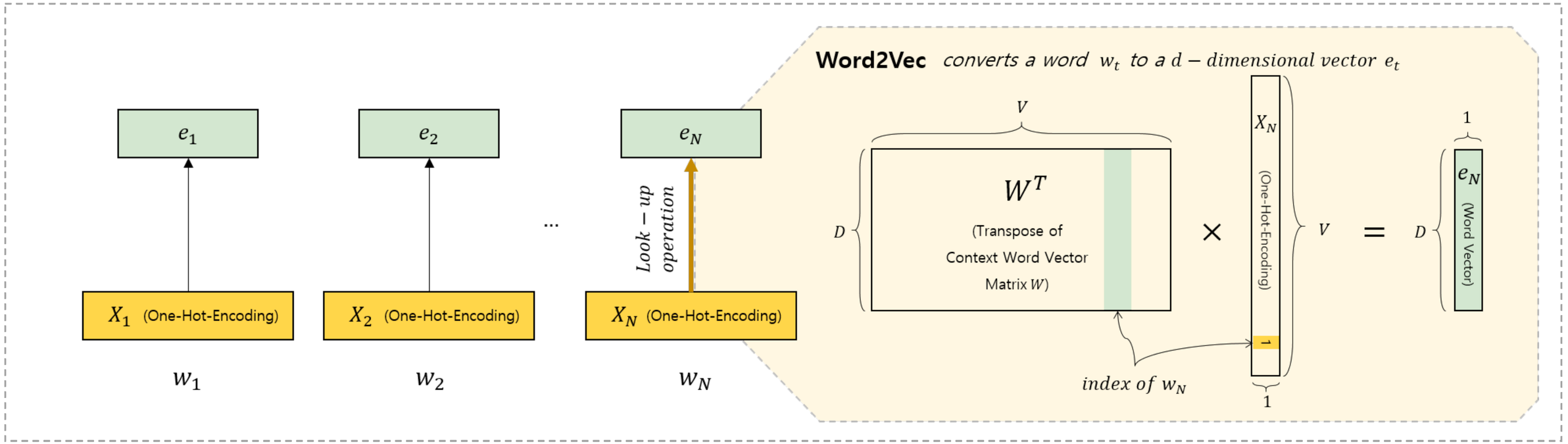
Embeddings in NLP

- Pretraining of token embeddings (unsupervised as LM or supervised on the task with a lot of data)
- Incorporating embeddings into a model for a target task
- Training of the target task model:
  - feature-based: use embeddings as additional features
  - fine-tuning: fine-tune embeddings

Important for fine-tuning:

- enough data for the target task
- architecture need to be applicable both to LM and to the target task

Individual embeddings



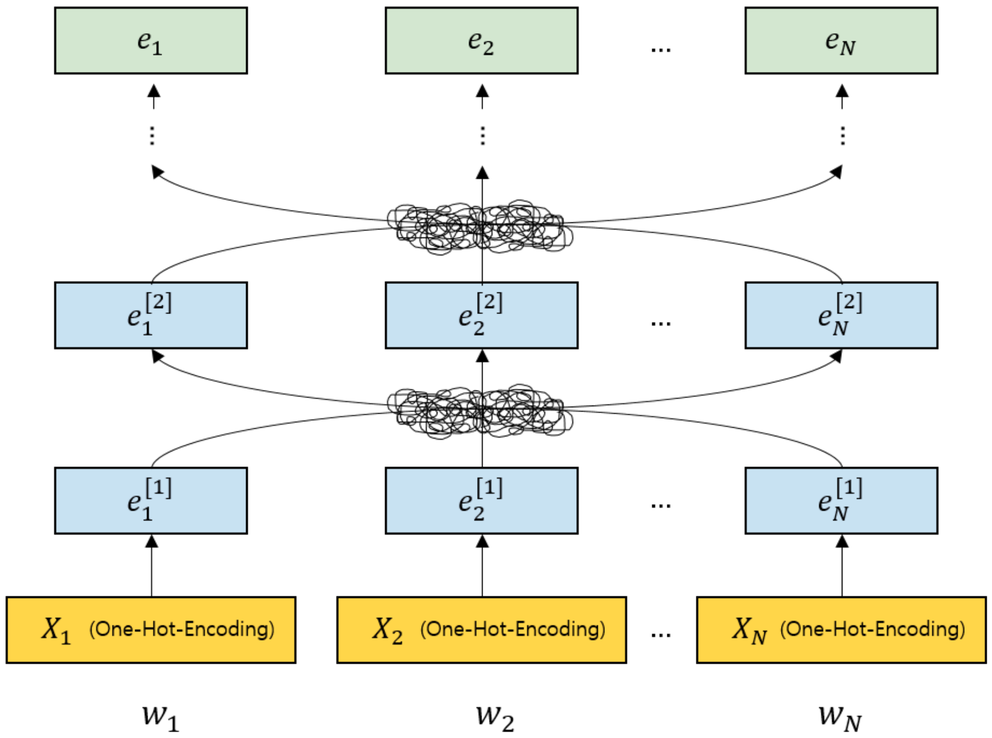
Wor2Vec, Glove, fastText …

http://intelligence.korea.ac.kr/members/wschoi/nlp/deeplearning/paperreview/Contextualized-Word-Embedding/

Contextualized embeddings

Contextualized Word Representations

converts a sequence of words (more precisely, tokens) $w_1 w_2 \dots w_N$ *to a series of* $d-$ *dimensional vectors* $e_1 e_2, \dots e_N$



Output:
series of $d-$ *dimensional vectors* $e_1 e_2, \dots e_N$

deep neural network
(e.g. deep bi-LSTM or deep-self-attention Nets)
for extracting features

simple word (or token) embedding
+ (optional) positional embedding

CoVe, ELMo, BERT, GPT…

http://intelligence.korea.ac.kr/members/wschoi/nlp/deeplearning/paperreview/Contextualized-Word-Embedding/

Contextualised embeddings

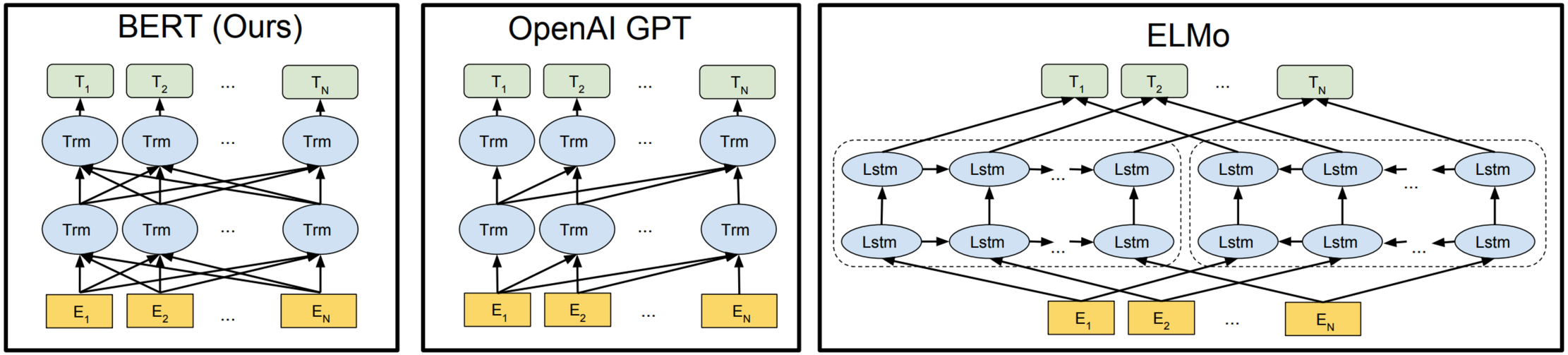
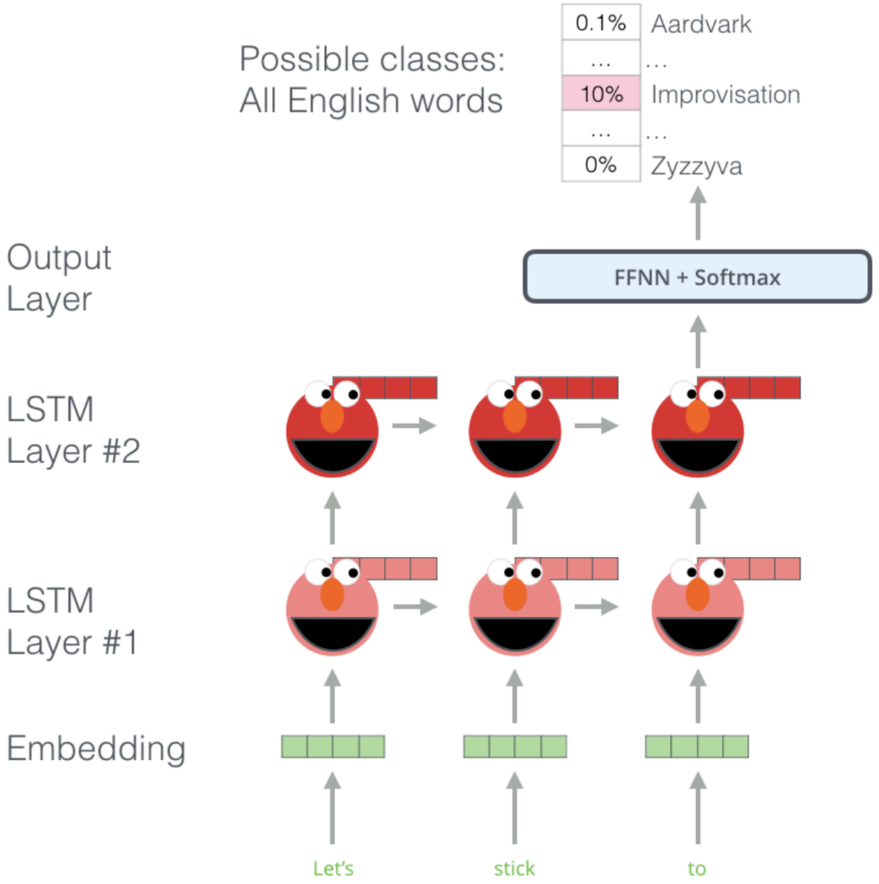


Figure 3: Differences in pre-training model architectures. BERT uses a bidirectional Transformer. OpenAI GPT uses a left-to-right Transformer. ELMo uses the concatenation of independently trained left-to-right and right-to-left LSTMs to generate features for downstream tasks. Among the three, only BERT representations are jointly conditioned on both left and right context in all layers. In addition to the architecture differences, BERT and OpenAI GPT are fine-tuning approaches, while ELMo is a feature-based approach.

ELMO: pre-training



- Use 2 LSTMs: forward and backward
- Params of embeddings and softmax are shared
- Optimize log likelihood:

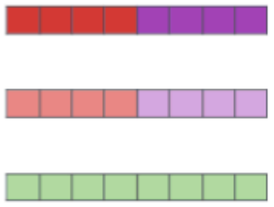
$$\sum_{k=1}^{N} (\log p(t_k \mid t_1, \ldots, t_{k-1}; \Theta_x, \overrightarrow{\Theta}_{LSTM}, \Theta_s) + \log p(t_k \mid t_{k+1}, \ldots, t_N; \Theta_x, \overleftarrow{\Theta}_{LSTM}, \Theta_s))$$

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

ELMO: embeddings

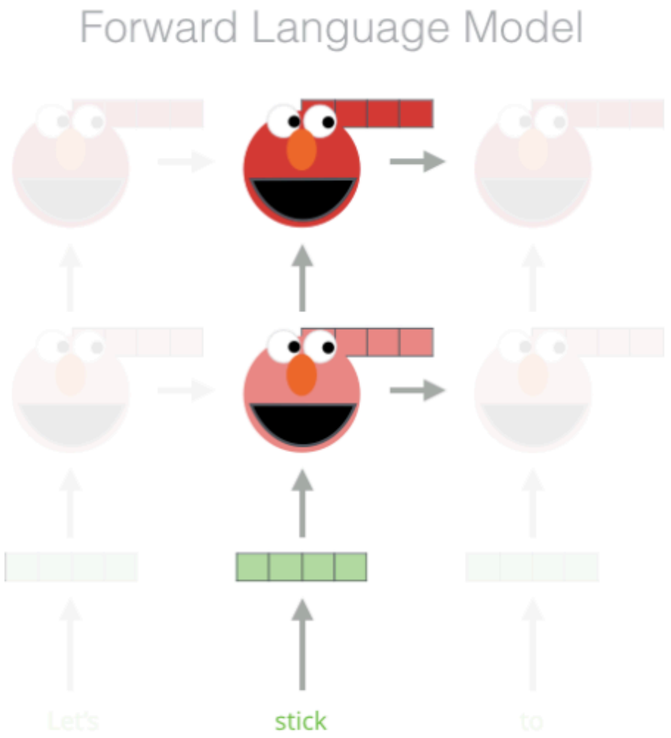
1- Concatenate hidden layers

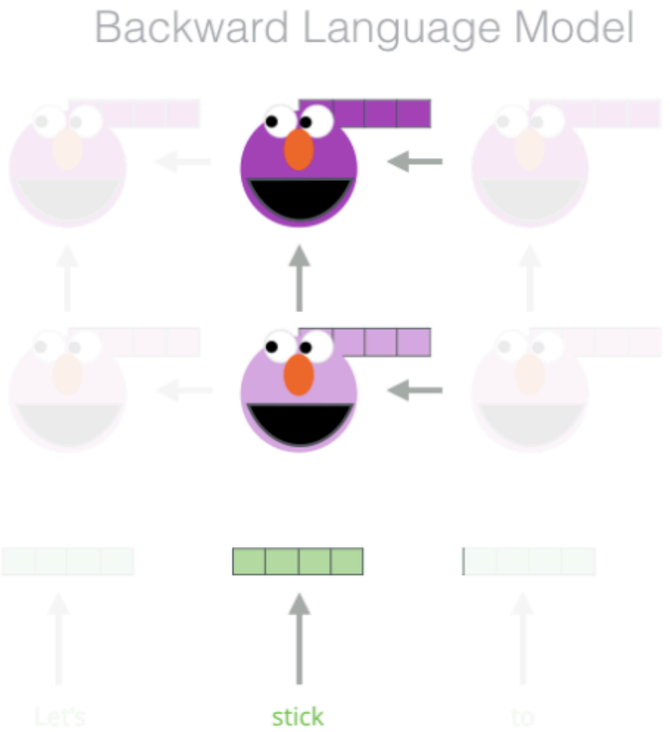
2- Multiply each vector by a weight based on the task



3- Sum the (now weighted) vectors

ELMo embedding of "stick" for this task in this context





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

ELMO

Pros:

- new SOTA at the time
- left and right context

Cons:

- RNN
- left and right context act almost independently
- mostly feature-based

https://arxiv.org/pdf/1802.05365.pdf