Seq2seq, NMT, Transformer

Milan Straka

Apr 17, 2023

EUROPEAN UNION
European Structural and Investment Fund
Operational Programme Research,
Development and Education

Charles University in Prague
Faculty of Mathematics and Physics
Institute of Formal and Applied Linguistics

unless otherwise stated

Sequence-to-Sequence Architecture

Sequence-to-Sequence Architecture

Sequence-to-Sequence is a name for an architecture allowing to produce an arbitrary output sequence $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_M$ from an input sequence $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$.

Unlike CRF/CTC, no assumptions are necessary and we condition each output sequence element on all input sequence elements and all already generated output sequence elements:

$$P(\boldsymbol{y}_i | \boldsymbol{x}_1, \ldots, \boldsymbol{x}_N, \boldsymbol{y}_1, \ldots, \boldsymbol{y}_{i-1}).$$

Sequence-to-Sequence Architecture

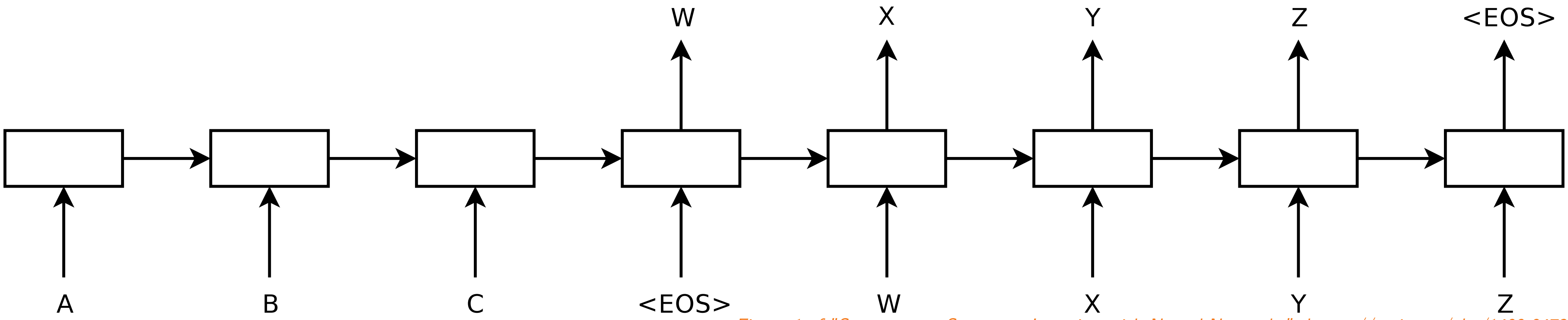


Figure 1 of "Sequence to Sequence Learning with Neural Networks", https://arxiv.org/abs/1409.0473

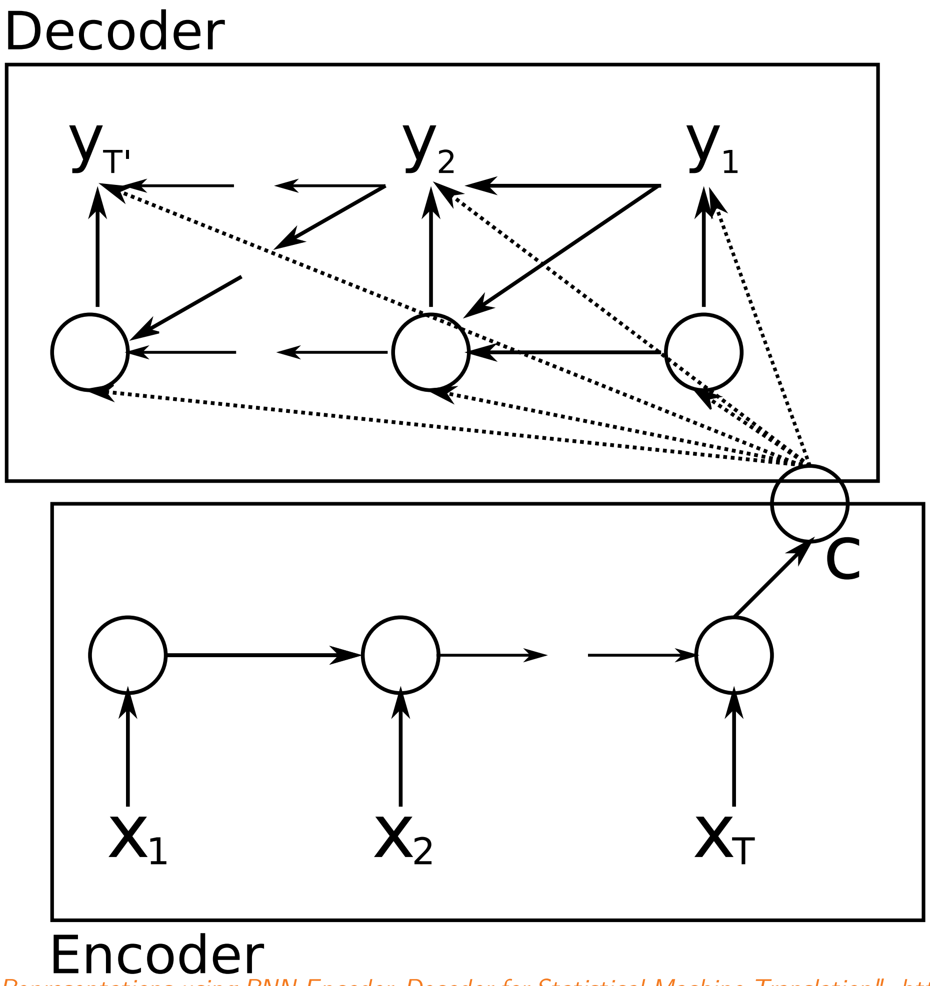
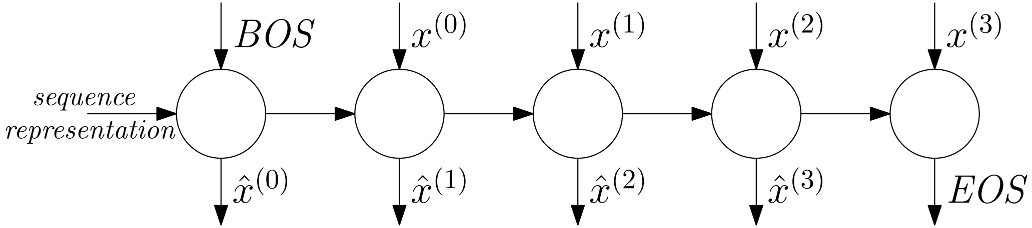


Figure 1 of "Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation", https://arxiv.org/abs/1406.1078

Training

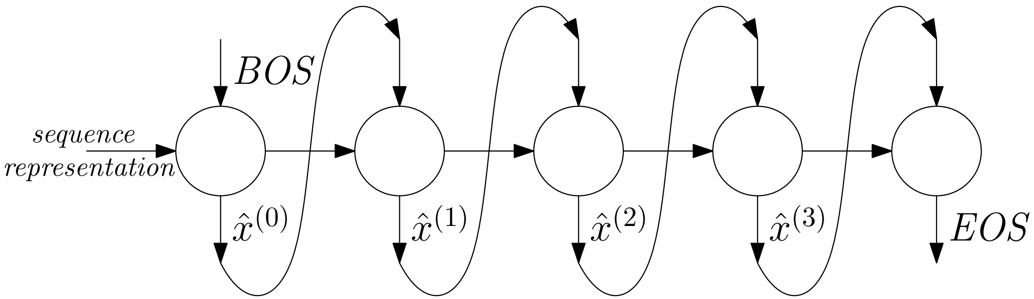
The so-called **teacher forcing** is used during training – the gold outputs are used as inputs during training.



Inference

During inference, the network processes its own predictions – such an approach is called **autoregressive decoding**.

Usually, the generated logits are processed by an $\arg\max$, the chosen word embedded and used as next input.



Tying Word Embeddings

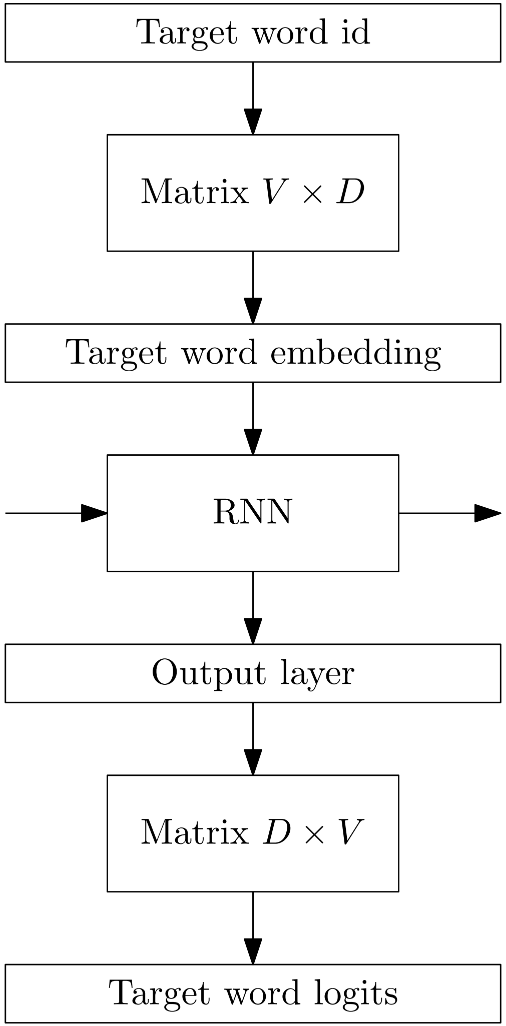
In the decoder, we both:

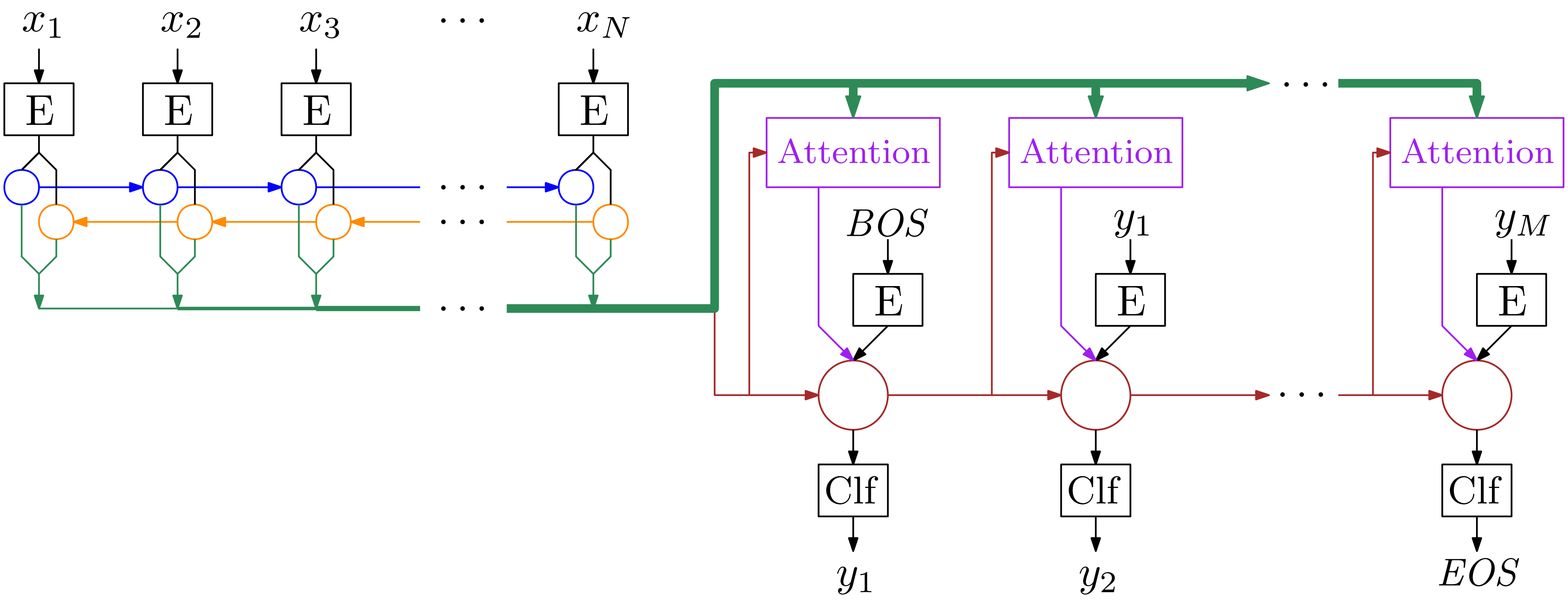
- embed the previous prediction, using a matrix of size $\mathbb{R}^{V \times D}$, where V is the vocabulary size and D is the embedding size;
- classify the hidden state into current prediction, using a matrix of size $\mathbb{R}^{D \times V}$.

Both these matrices have similar meaning – they represent words in the embedding space (the first explicitly represents words by the embeddings, the second produces logits by computing weighted cosine similarity of the inputs and columns of the weight matrix).

Therefore, it makes sense to **tie** these matrices, i.e., to represent one of them as a transposition of the other.

- However, while the embedding matrix should usually have constant variance per dimension, the output layer should keep the variance of the RNN output; therefore, the output layer matrix is usually the embedding matrix divided by $\sqrt{D}$.




x_1
x_2
x_3
$\cdots$
x_N
E
Attention
BOS
y_1
y_M
Clf
y_1
y_2
EOS

As another input during decoding, we add *context vector* c_i:

$$\boldsymbol{s}_i = f(\boldsymbol{s}_{i-1}, \boldsymbol{y}_{i-1}, \boldsymbol{c}_i).$$

We compute the context vector as a weighted combination of source sentence encoded outputs:

$$\boldsymbol{c}_i = \sum_j \alpha_{ij} \boldsymbol{h}_j$$

The weights α_{ij} are softmax of e_{ij} over j,

$$\boldsymbol{\alpha}_i = \operatorname{softmax}(\boldsymbol{e}_i),$$

with e_{ij} being

$$e_{ij} = \boldsymbol{v}^\top \tanh(\boldsymbol{V}\boldsymbol{h}_j + \boldsymbol{W}\boldsymbol{s}_{i-1} + \boldsymbol{b}).$$

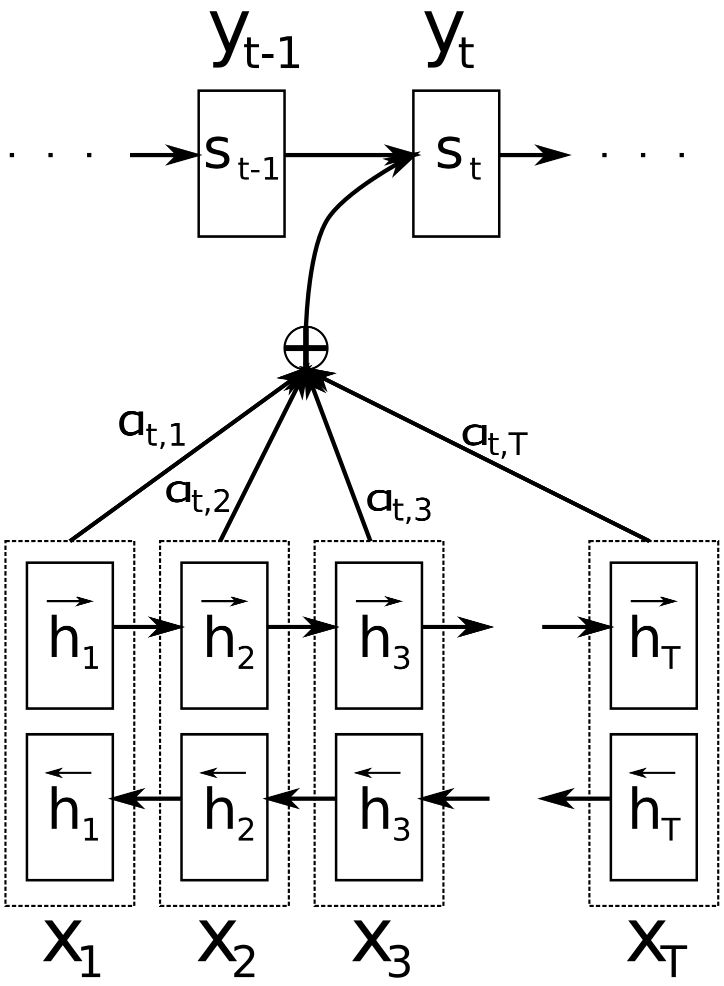
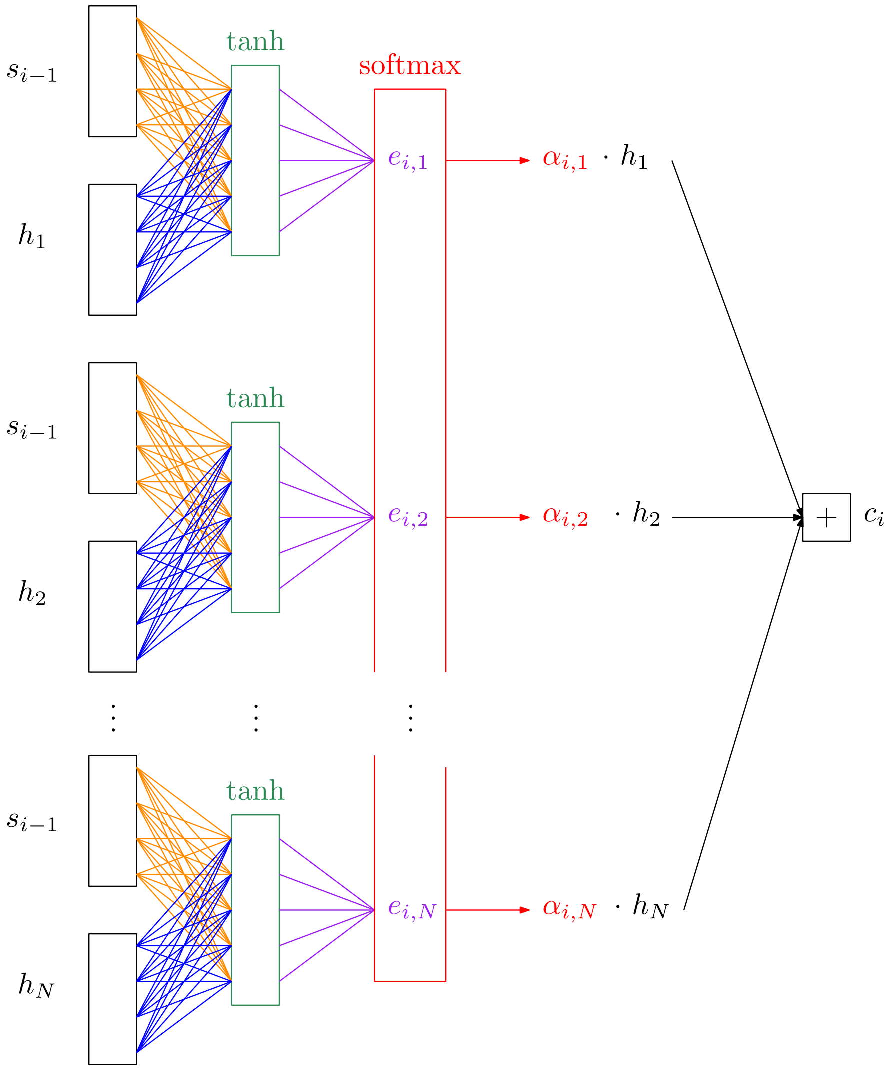


Figure 1 of "Neural Machine Translation by Jointly Learning to Align and Translate", https://arxiv.org/abs/1409.0473

Bahdanau Attention Implementation



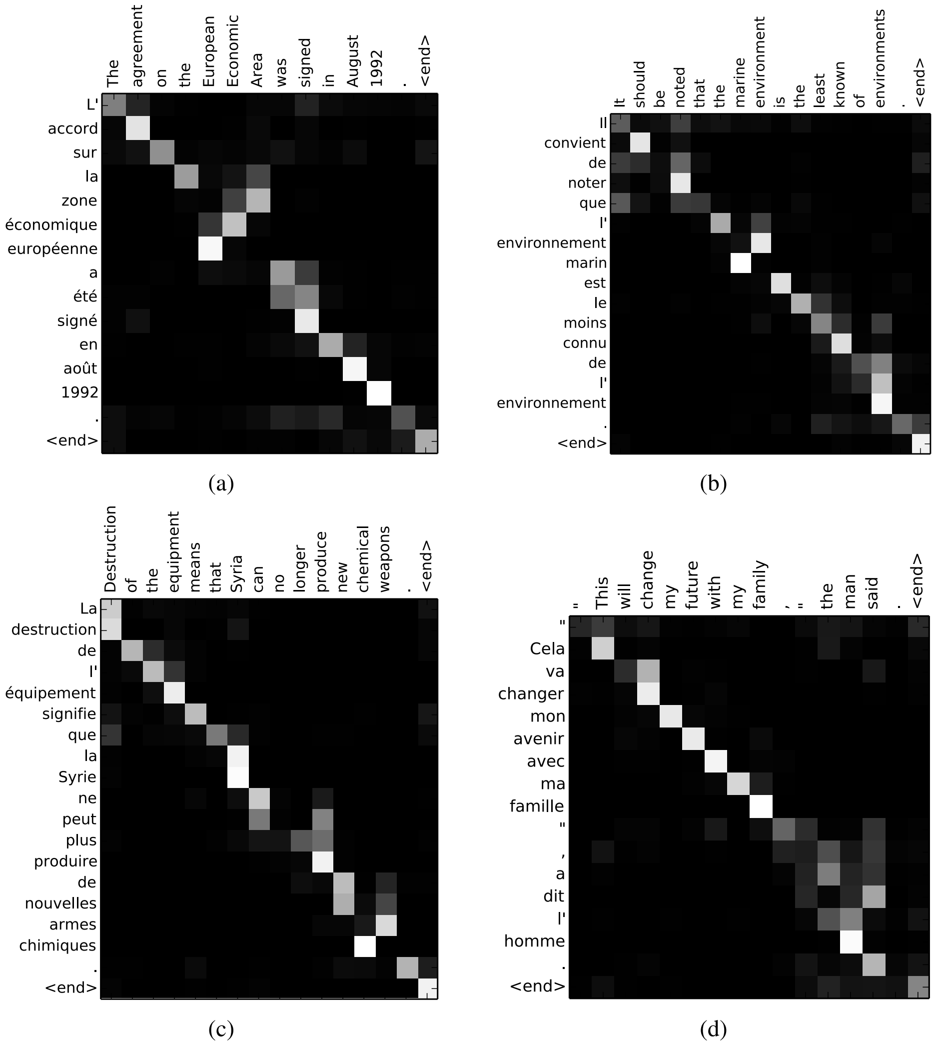


Figure 3 of "Neural Machine Translation by Jointly Learning to Align and Translate", https://arxiv.org/abs/1409.0473

In the described *Bahdanau* (or *additive*) attention, we performed

$$e_{ij} = \boldsymbol{v}^\top \tanh(\boldsymbol{V}\boldsymbol{h}_j + \boldsymbol{W}\boldsymbol{s}_{i-1} + \boldsymbol{b}).$$

There are however other methods how $\boldsymbol{V}\boldsymbol{h}_j$ and $\boldsymbol{W}\boldsymbol{s}_{i-1}$ can be combined, most notably the *Luong* (or *dot-product*) attention, which uses just a dot product:

$$e_{ij} = \big(\boldsymbol{V}\boldsymbol{h}_j\big)^T\big(\boldsymbol{W}\boldsymbol{s}_{i-1}\big).$$

The latter is easier to implement, but may sometimes be more difficult to train (scaling helps a bit, wait for the Transformer self-attention description); both approaches are used in quite a few papers.

Subword Units

Translate **subword units** instead of words. The subword units can be generated in several ways, the most commonly used are:

- **BPE**: Using the *byte pair encoding* algorithm. Start with individual characters plus a special end-of-word symbol •. Then, merge the most occurring symbol pair A, B by a new symbol AB, with the symbol pair never crossing word boundary (so that the end-of-word symbol cannot be inside a subword).

  Considering text with words *low, lowest, newer, wider*, a possible sequence of merges:

$$r \;\; \bullet \rightarrow r\bullet$$
$$l \;\; o \rightarrow lo$$
$$lo \;\; w \rightarrow low$$
$$e \;\; r\bullet \rightarrow er\bullet$$

  The BPE algorithm is executed on the training data, and it generates the resulting dictionary, merging rules, and training data encoded using this dictionary.

- **Wordpieces**: Given a text divided into subwords, we can compute unigram probability of every subword, and then get the likelihood of the text under a unigram language model by multiplying the probabilities of the subwords in the text.

  When we have only a text and a subword dictionary, we divide the text in a greedy fashion, iteratively choosing the longest existing subword.

  When constructing the subwords, we again start with individual characters (compared to BPE, we have a *start-of-word* character instead of an *end-of-word* character), and then repeatedly join such a pair of subwords that increases the unigram language model likelihood the most.
  - In the original implementation, the input data were once in a while "reparsed" (retokenized) in a greedy fashion with the up-to-date dictionary. However, the recent implementations do not seem to do it – but they retokenize the training data with the final dictionary, contrary to the BPE approach.

For both approaches, usually quite little subword units are used (32k-64k), often generated on the union of the two vocabularies of the source and target languages (the so-called *joint BPE* or *shared wordpieces*).

BPE and WordPieces Comparison

Both the BPE and the WordPieces give very similar results; the biggest difference is that during the inference:

- for BPE, the sequence of merges must be performed in the same order as during the construction of the BPE (because we use the output of BPE as training data),
- for Wordpieces, it is enough to find longest matches from the subword dictionary (because we reprocessed the training data with the final dictionary);
- note that the above difference is mostly artificial – if we reparsed the training data in the BPE approach, we could also perform "greedy tokenization".

Of course, the two algorithms also differ in the way how they choose the pair of subwords to merge.

Both algorithms are implemented in quite a few libraries, most notably the `sentencepiece` library and the Hugging Face `tokenizers` package.

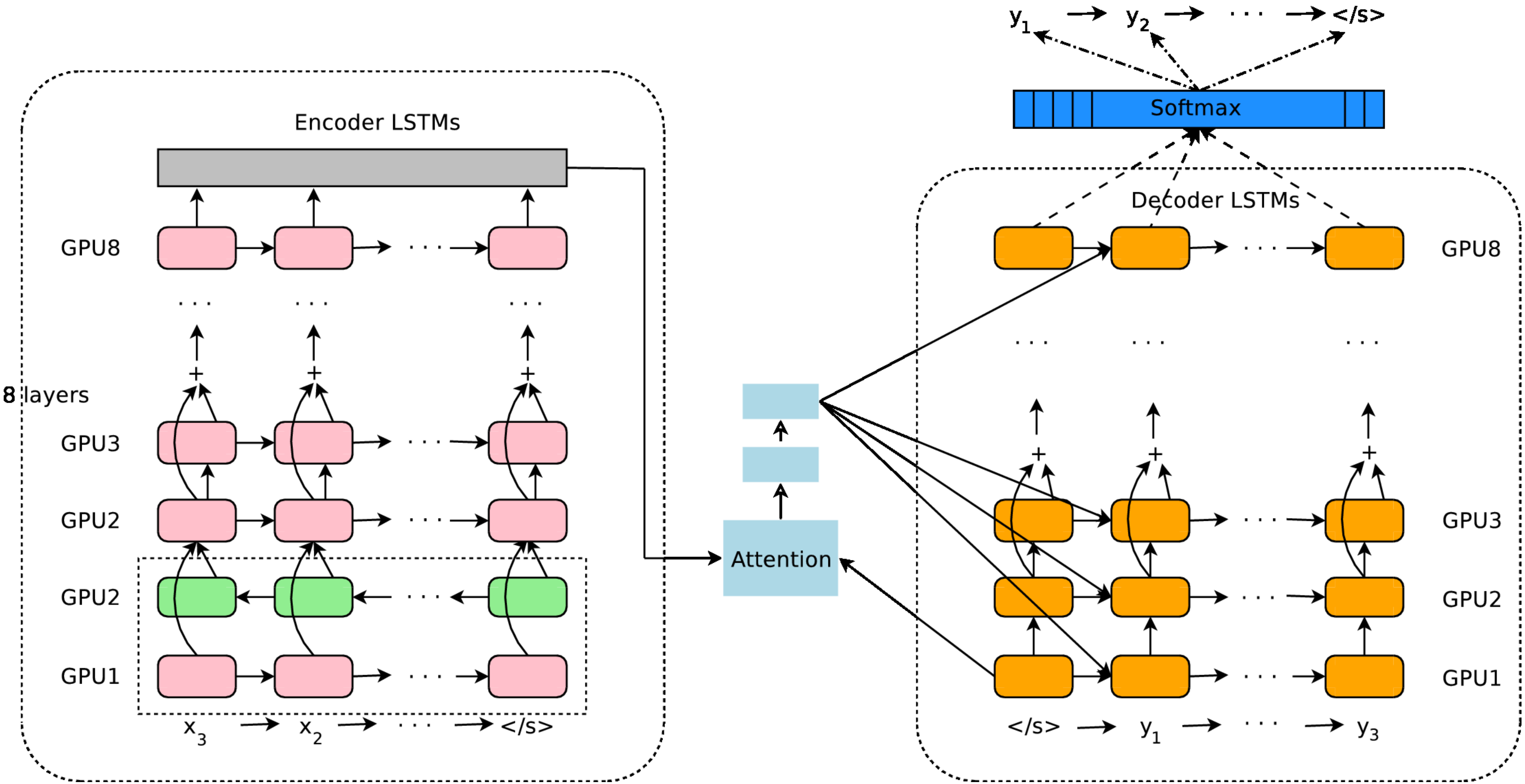


Figure 1 of "Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation", https://arxiv.org/abs/1609.08144

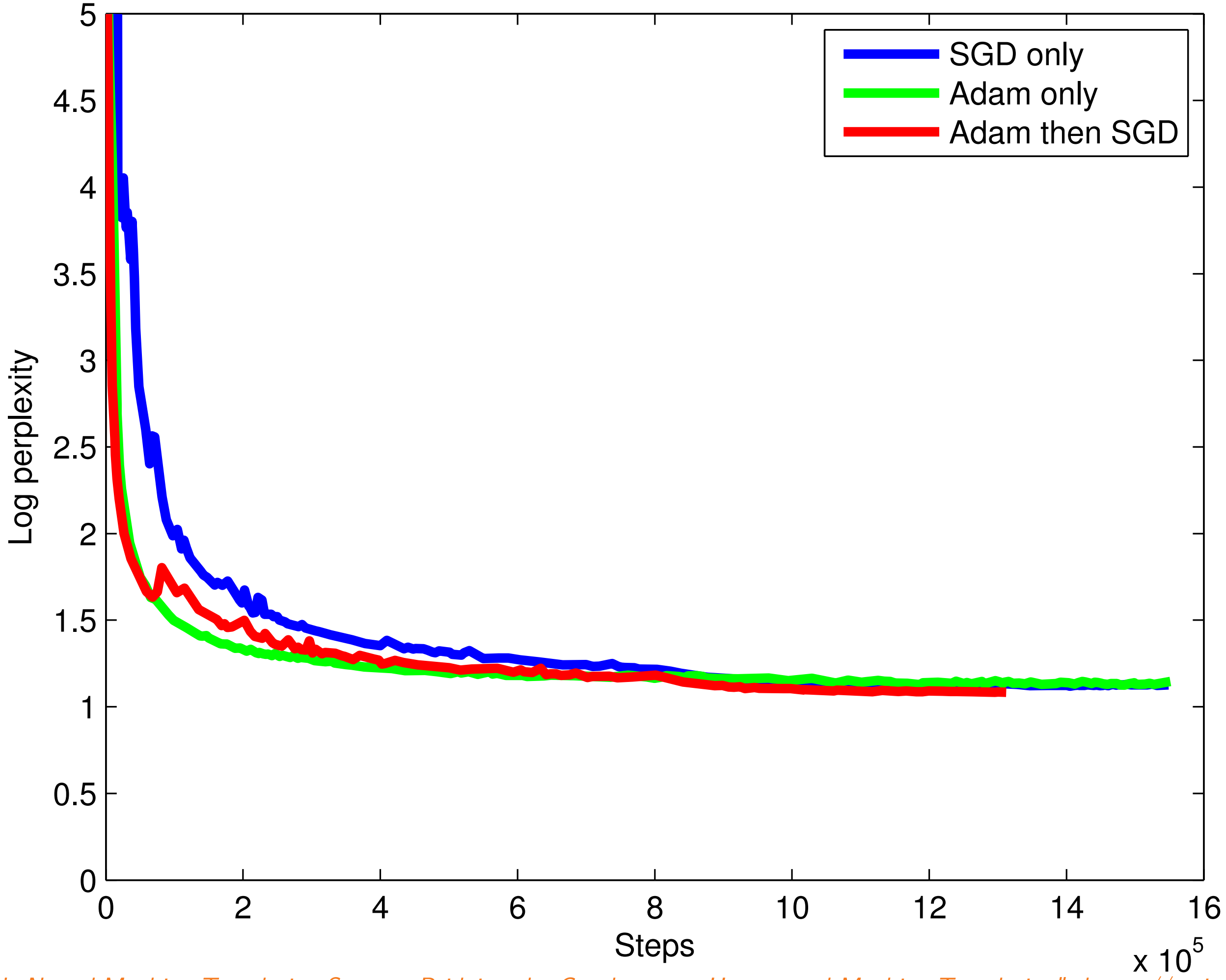


Figure 5 of "Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation", https://arxiv.org/abs/1609.08144

Google NMT

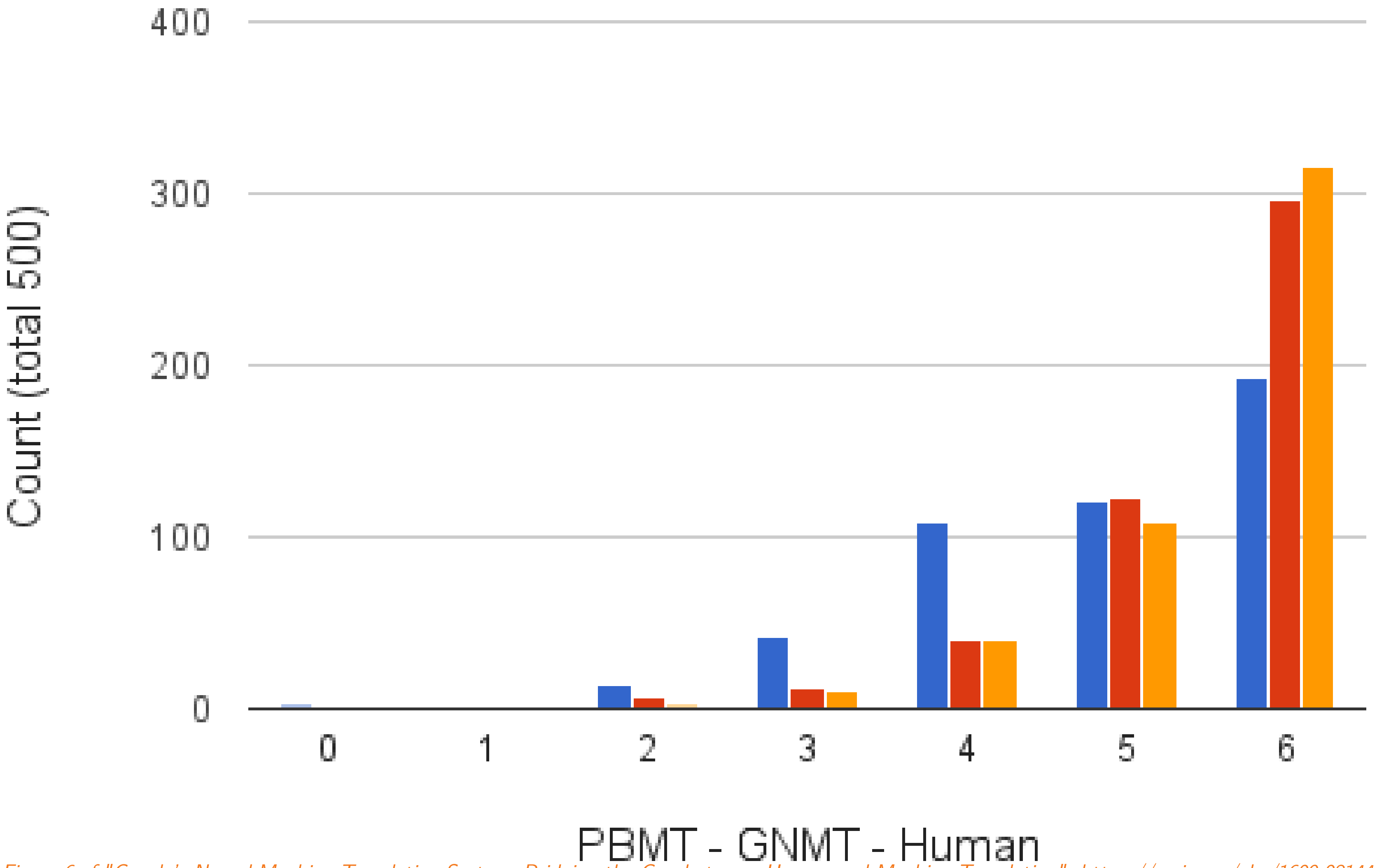


Figure 6 of "Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation", https://arxiv.org/abs/1609.08144

Beyond one Language Pair

Fig. 5. A selection of evaluation results, grouped by human rating.

Figure 5 of "Show and Tell: Lessons learned from the 2015 MSCOCO...", https://arxiv.org/abs/1609.06647

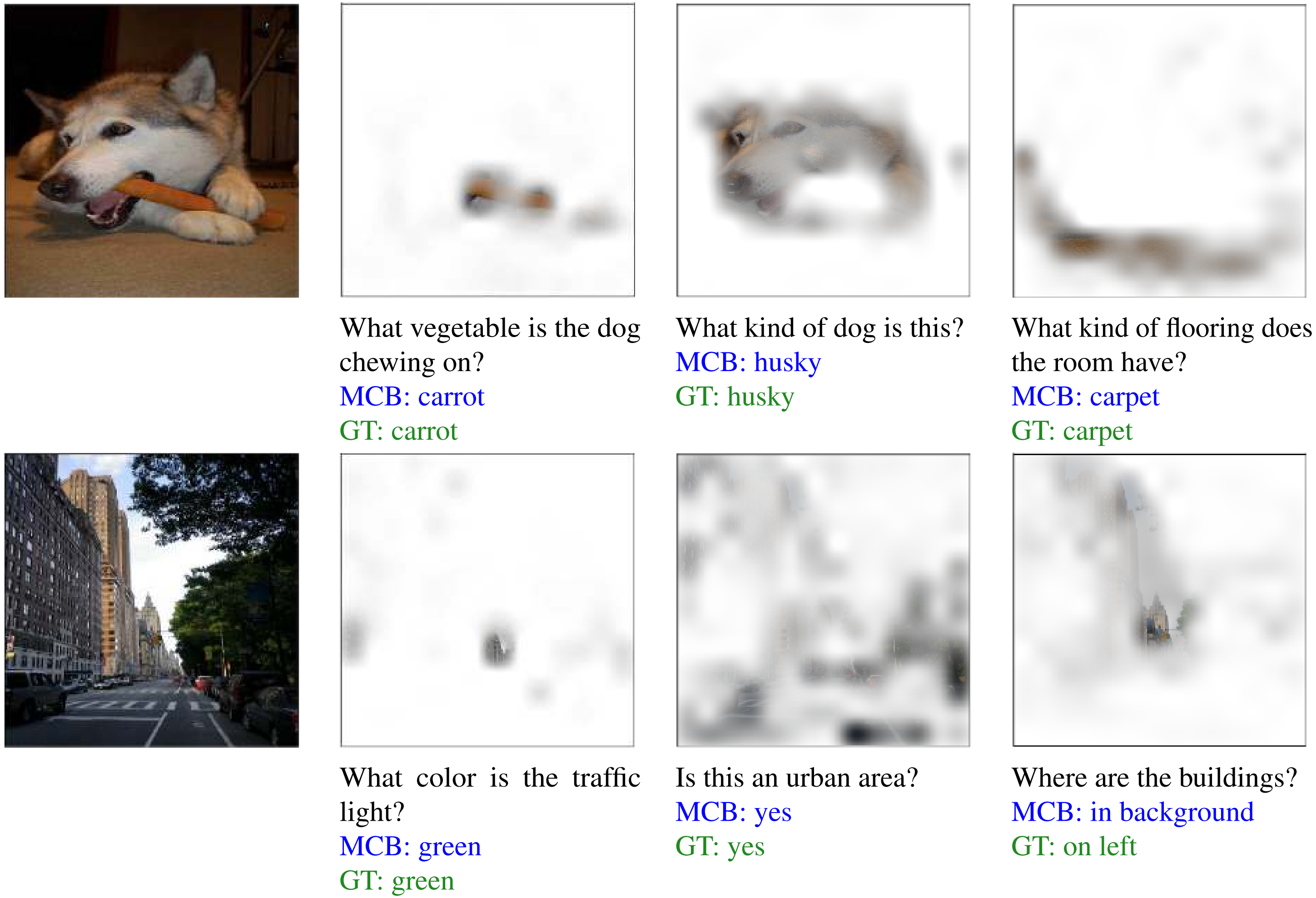


Figure 6 of "Multimodal Compact Bilinear Pooling for VQA and Visual Grounding", https://arxiv.org/abs/1606.01847

Multilingual and Unsupervised Translation

Many attempts at multilingual translation.

- Individual encoders and decoders, shared attention.
- Shared encoders and decoders.

Surprisingly, even unsupervised translation is attempted lately. By unsupervised we understand settings where we have access to large monolingual corpora, but no parallel data.

In 2019, the best unsupervised systems were on par with the best 2014 supervised systems.

		WMT-14			
		fr-en	en-fr	de-en	en-de
Unsupervised	Proposed system	33.5	36.2	27.0	22.5
	*detok. SacreBLEU**	33.2	33.6	26.4	21.2
Supervised	WMT best*	35.0	35.8	29.0	20.6†
	Vaswani et al. (2017)	-	41.0	-	28.4
	Edunov et al. (2018)	-	45.6	-	35.0

Table 3: Results of the proposed method in comparison to different supervised systems (BLEU).

Table 3 of "An Effective Approach to Unsupervised Machine Translation", https://arxiv.org/abs/1902.01313

Attention is All You Need

For some sequence processing tasks, *sequential* processing (as performed by recurrent neural networks) of its elements might be too restrictive.

Instead, we may want to be able to combine sequence elements independently on their distance.

Such processing is allowed in the **Transformer** architecture, originally proposed for neural machine translation in 2017 in *Attention is All You Need* paper.

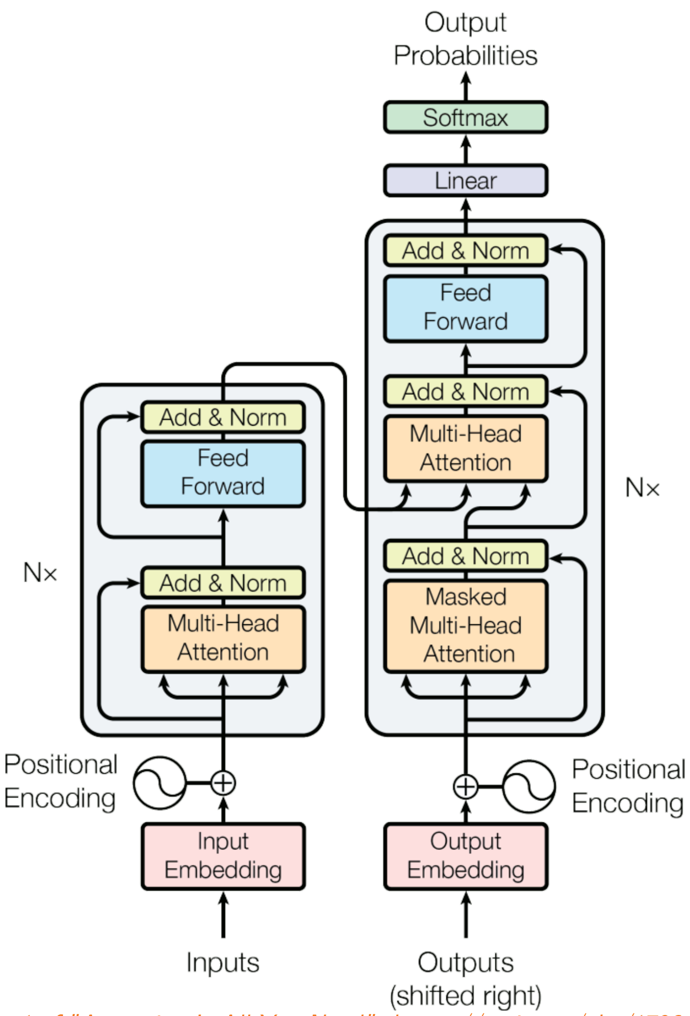
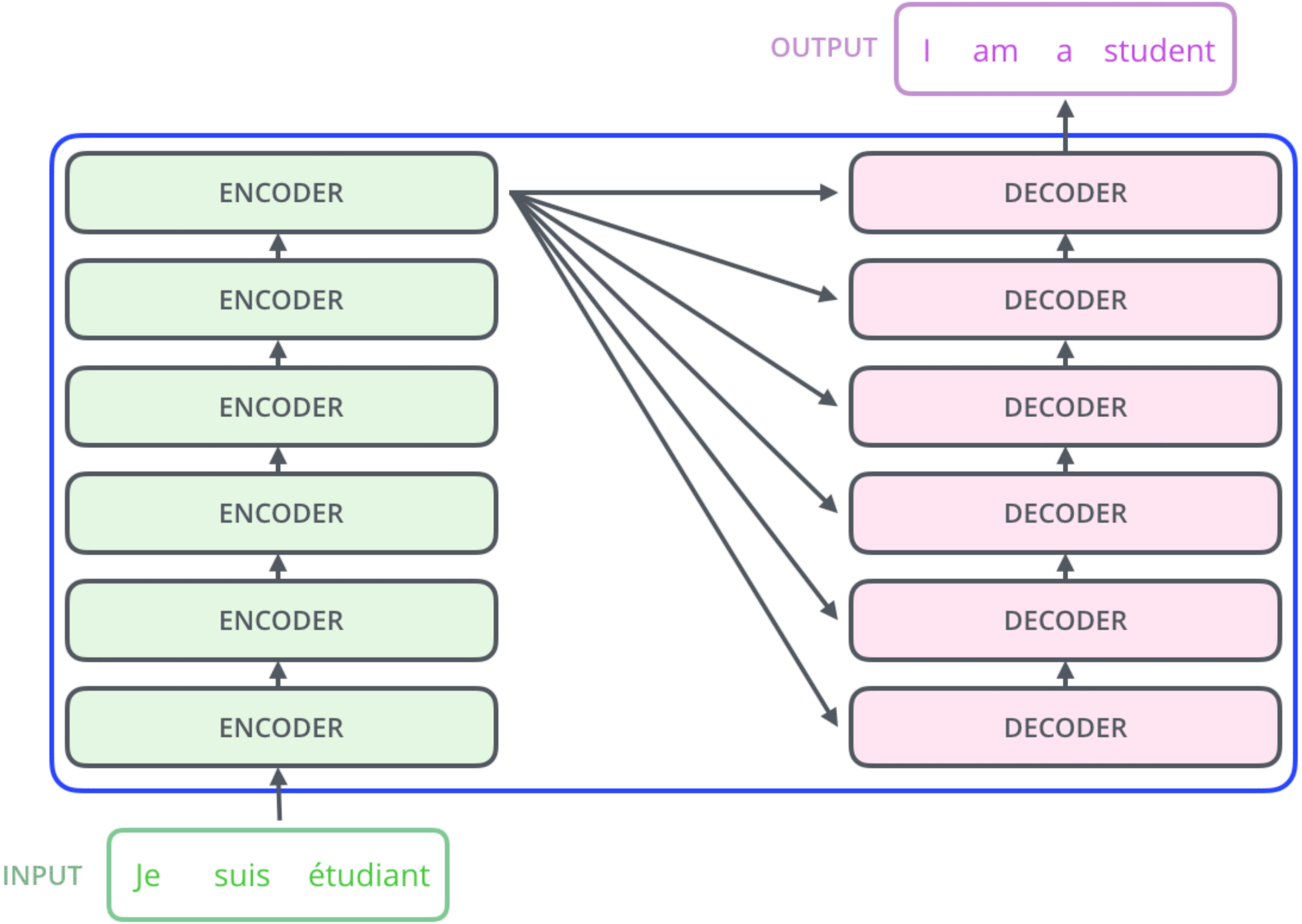
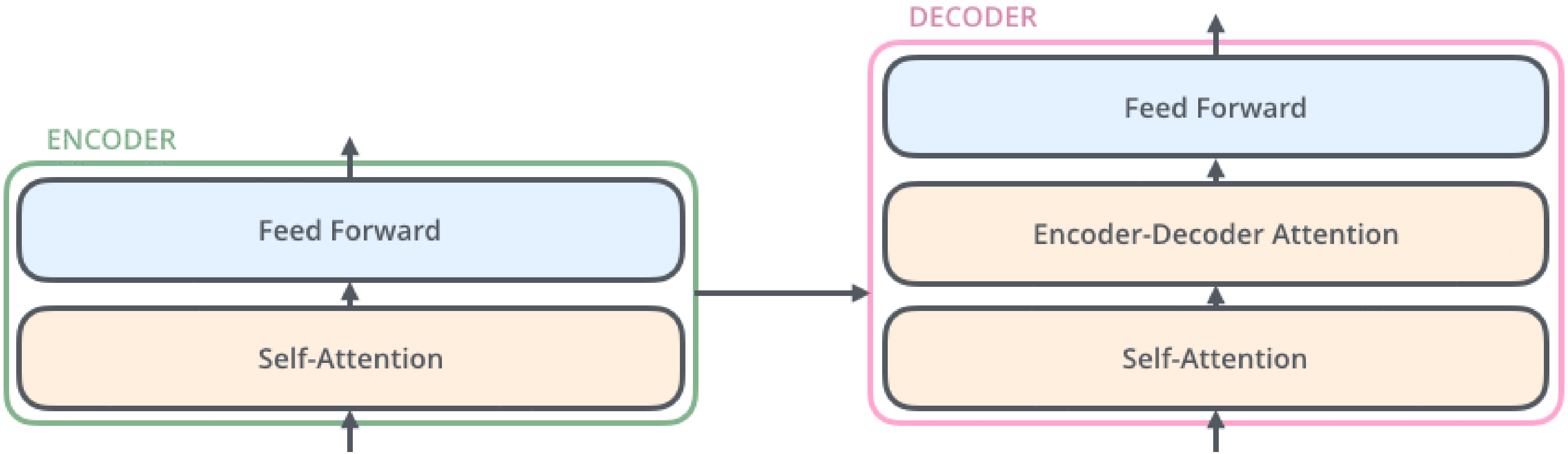


Figure 1 of "Attention Is All You Need", https://arxiv.org/abs/1706.03762

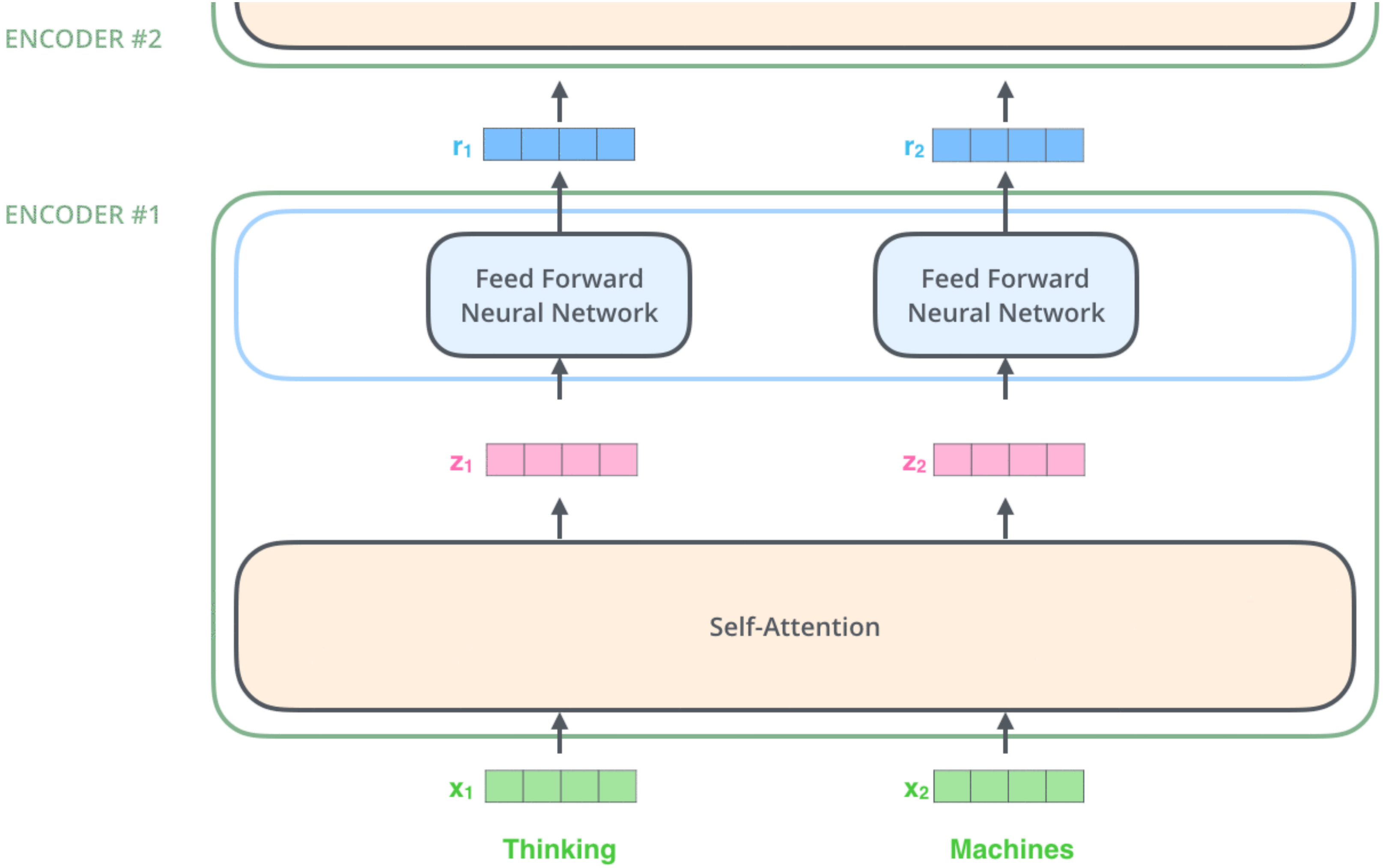
Transformer



http://jalammar.github.io/images/t/The_transformer_encoder_decoder_stack.png



http://jalammar.github.io/images/t/Transformer_decoder.png



http://jalammar.github.io/images/t/encoder_with_tensors_2.png

Transformer – Self-Attention

Assume that we have a sequence of n words represented using a matrix $\boldsymbol{X} \in \mathbb{R}^{n \times d}$.

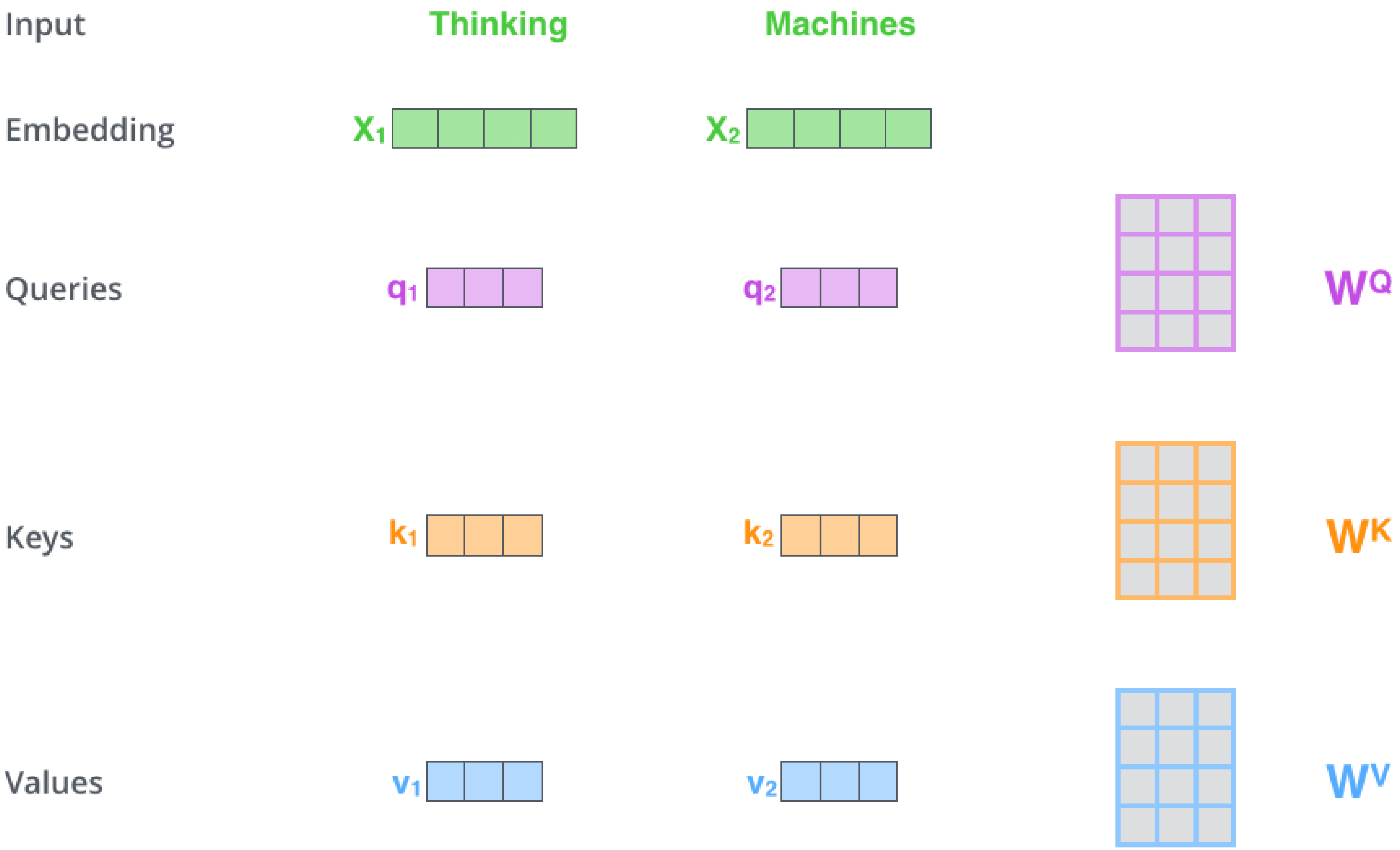
The attention module for queries $\boldsymbol{Q} \in \mathbb{R}^{n \times d_k}$, keys $\boldsymbol{K} \in \mathbb{R}^{n \times d_k}$ and values $\boldsymbol{V} \in \mathbb{R}^{n \times d_v}$ is defined as:

$$\operatorname{Attention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \operatorname{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{d_k}}\right)\boldsymbol{V}.$$

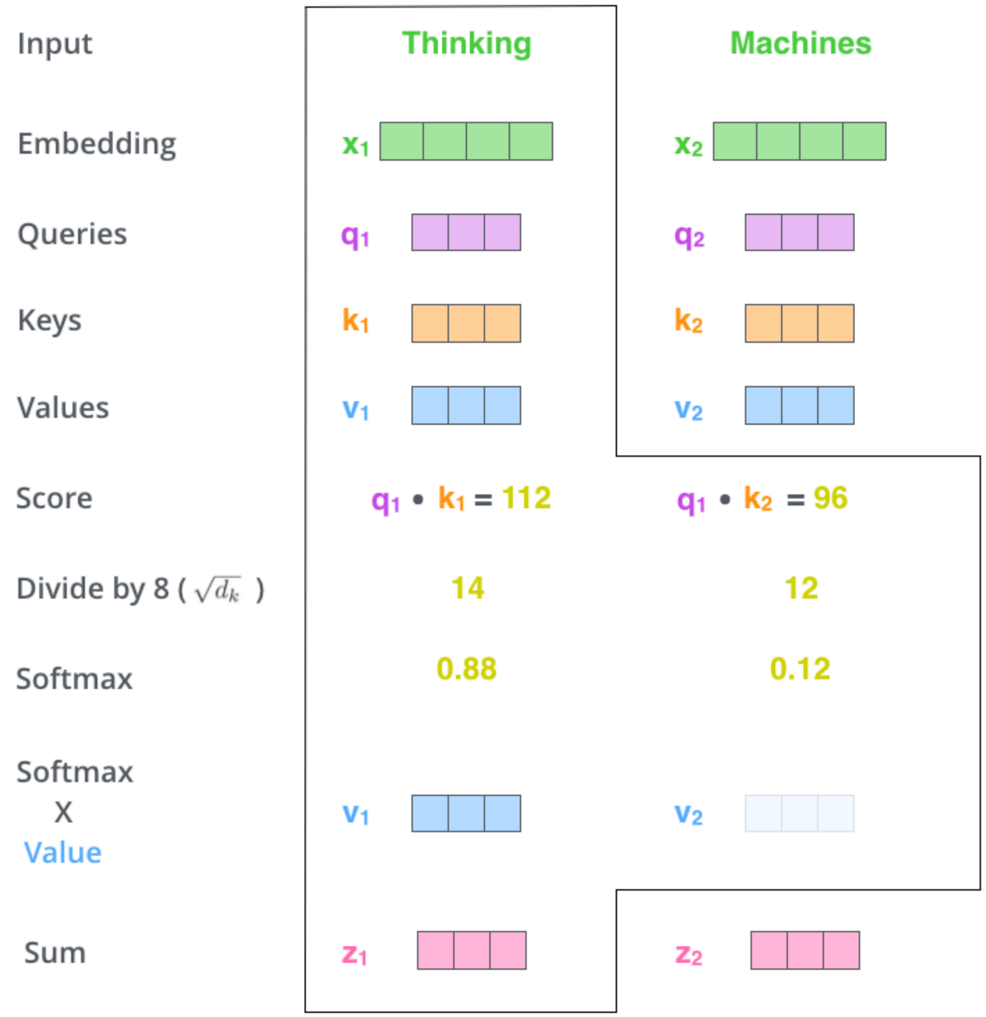
The queries, keys and values are computed from the input word representations $\boldsymbol{X}$ using a linear transformation as

$$\begin{aligned}\boldsymbol{Q} &= \boldsymbol{X}\boldsymbol{W}^Q\\ \boldsymbol{K} &= \boldsymbol{X}\boldsymbol{W}^K\\ \boldsymbol{V} &= \boldsymbol{X}\boldsymbol{W}^V\end{aligned}$$

for trainable weight matrices $\boldsymbol{W}^Q, \boldsymbol{W}^K \in \mathbb{R}^{d \times d_k}$ and $\boldsymbol{W}^V \in \mathbb{R}^{d \times d_v}$.

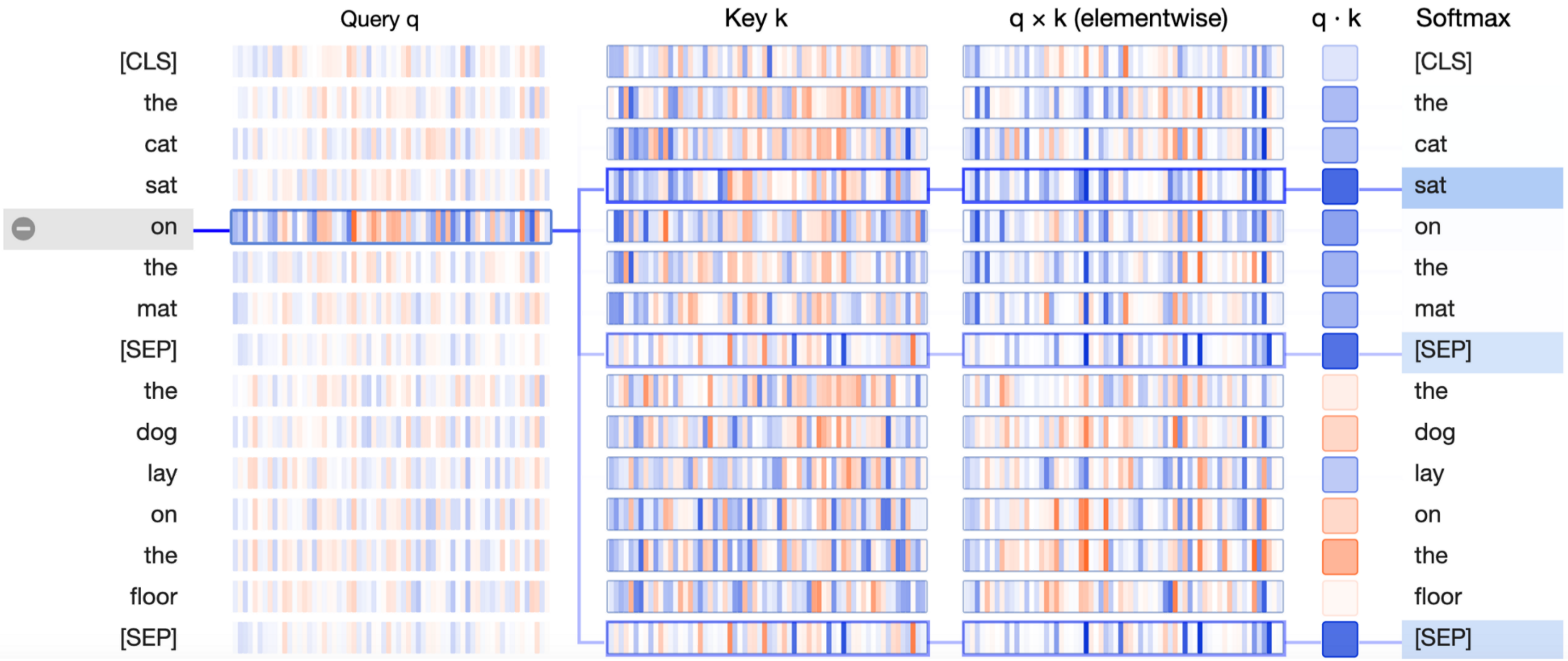


http://jalammar.github.io/images/t/transformer_self_attention_vectors.png

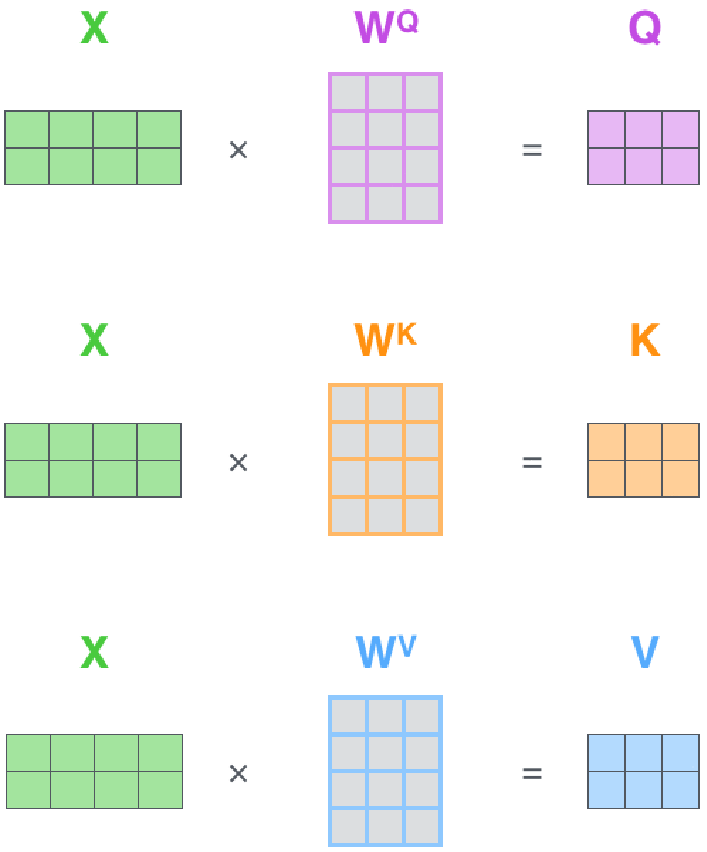


http://jalammar.github.io/images/t/self-attention-output.png

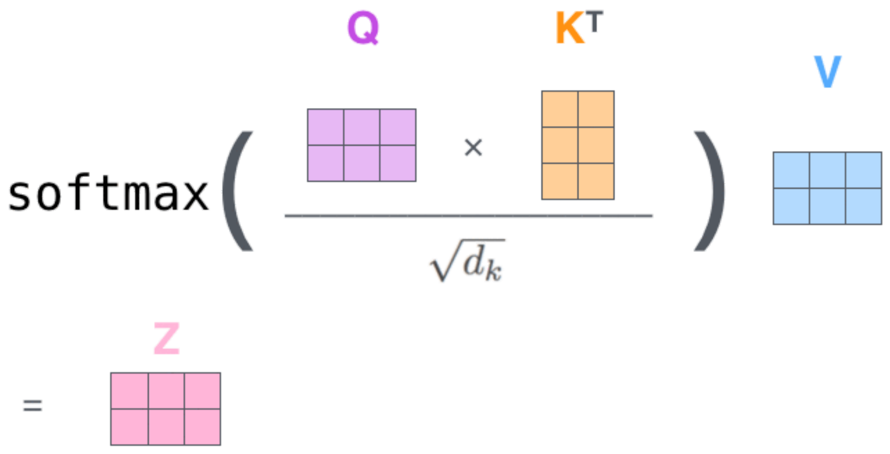
Transformer – Self-Attention



https://miro.medium.com/max/2000/1*jBsfVNOOcJ-I3tsLVgni_w.png



http://jalammar.github.io/images/t/self-attention-matrix-calculation.png



http://jalammar.github.io/images/t/self-attention-matrix-calculation-2.png

Multihead attention is used in practice. Instead of using one huge attention, we split queries, keys and values to several groups (similar to how ResNeXt works), compute the attention in each of the groups separately, concatenate the results and multiply them by a matrix $\boldsymbol{W}^O$.

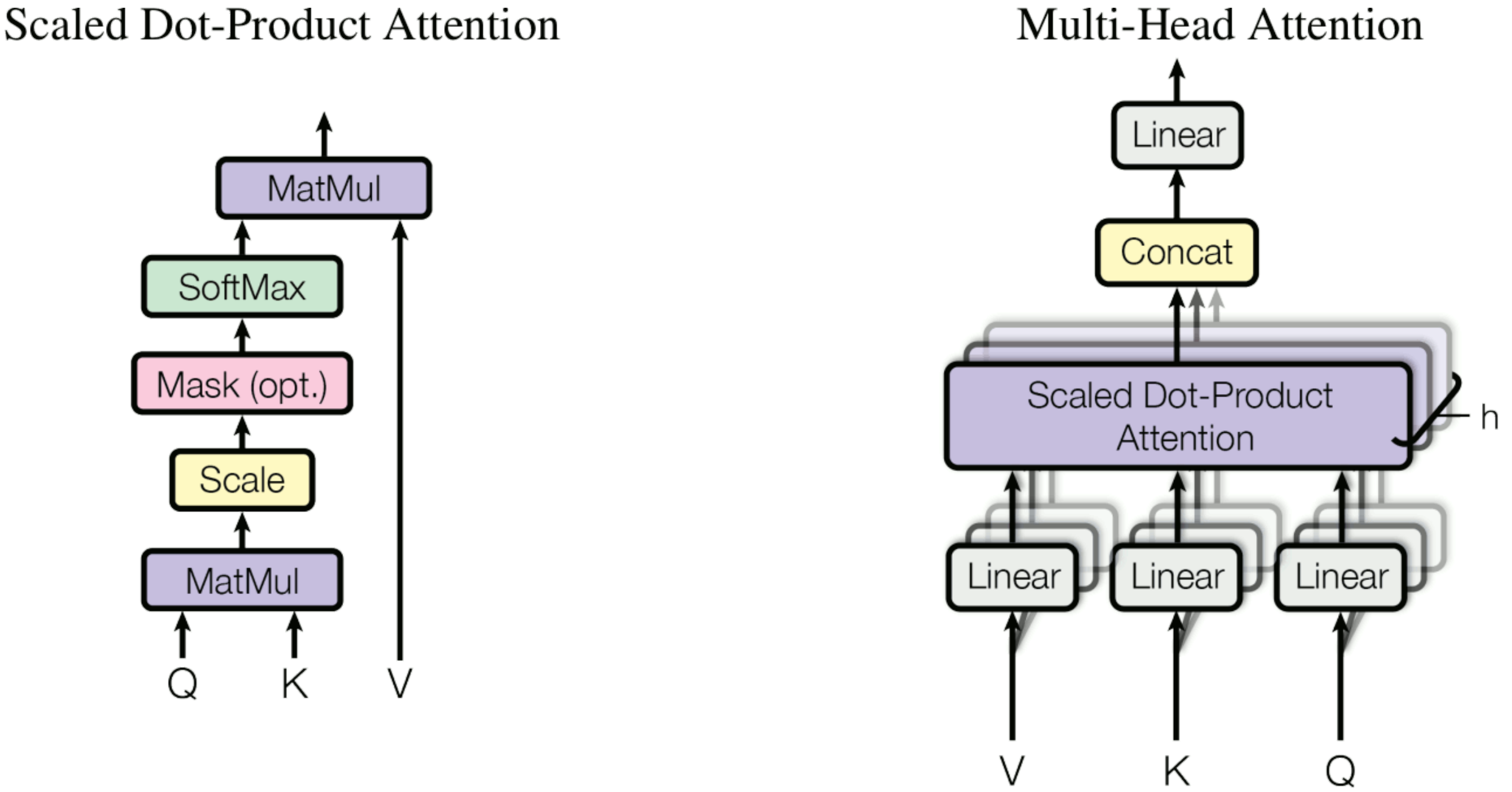
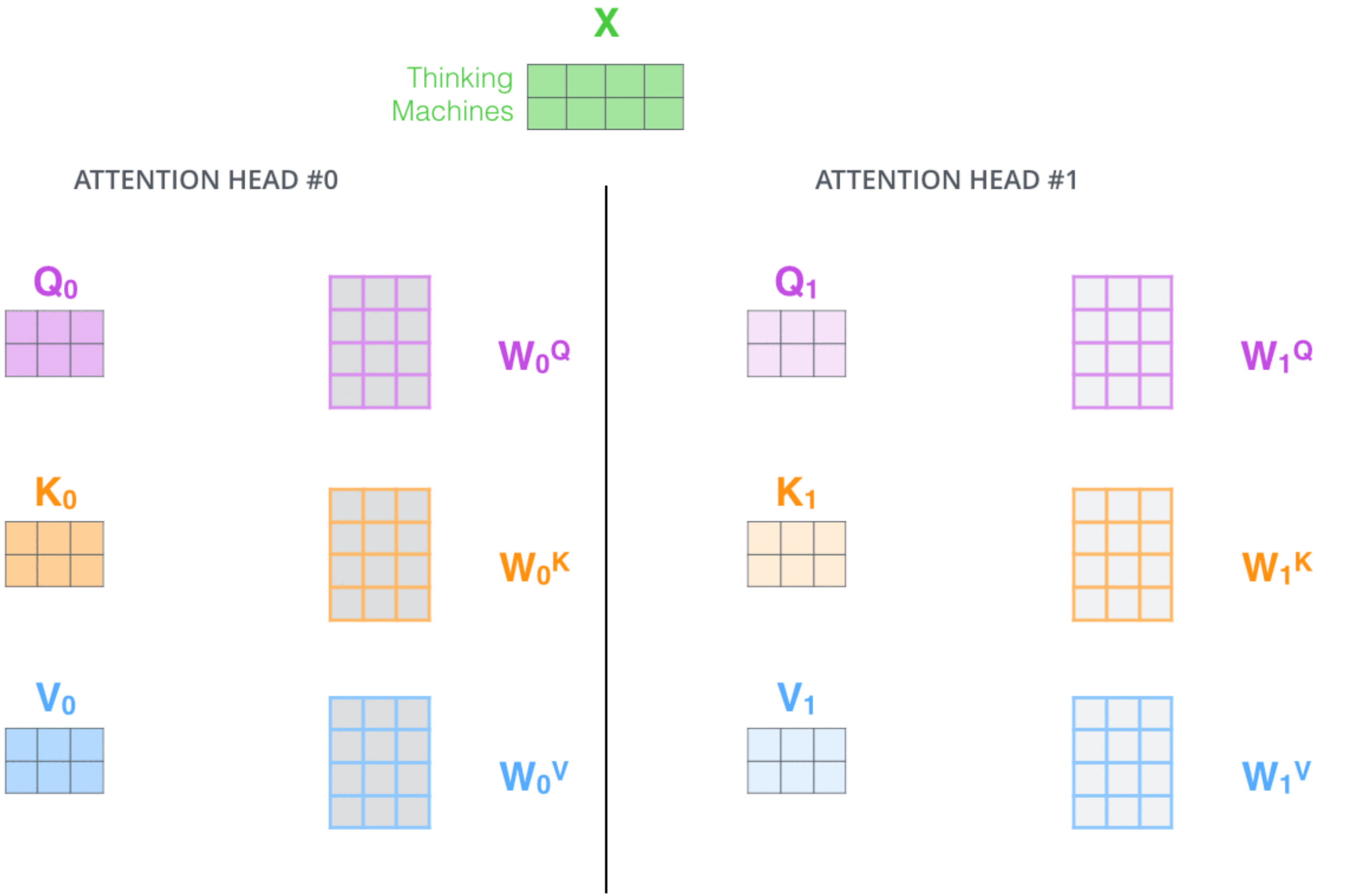
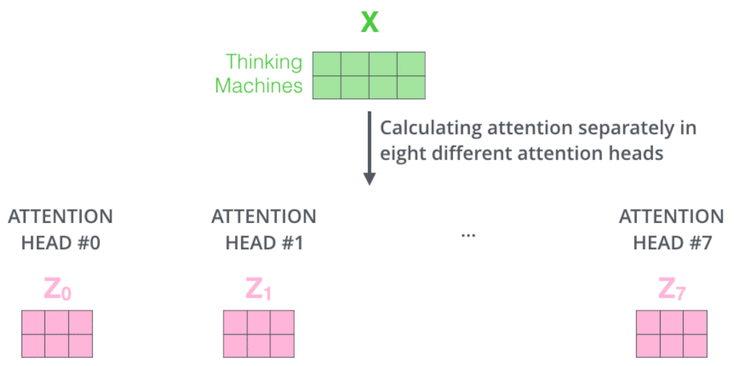


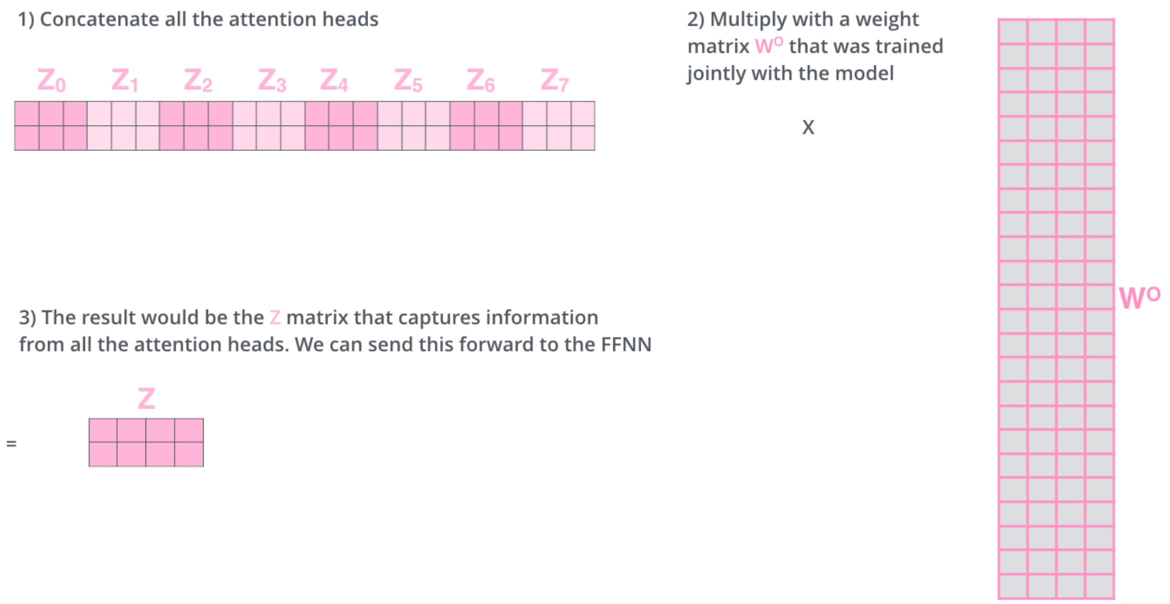
Figure 2 of "Attention Is All You Need", https://arxiv.org/abs/1706.03762



http://jalammar.github.io/images/t/transformer_attention_heads_qkv.png

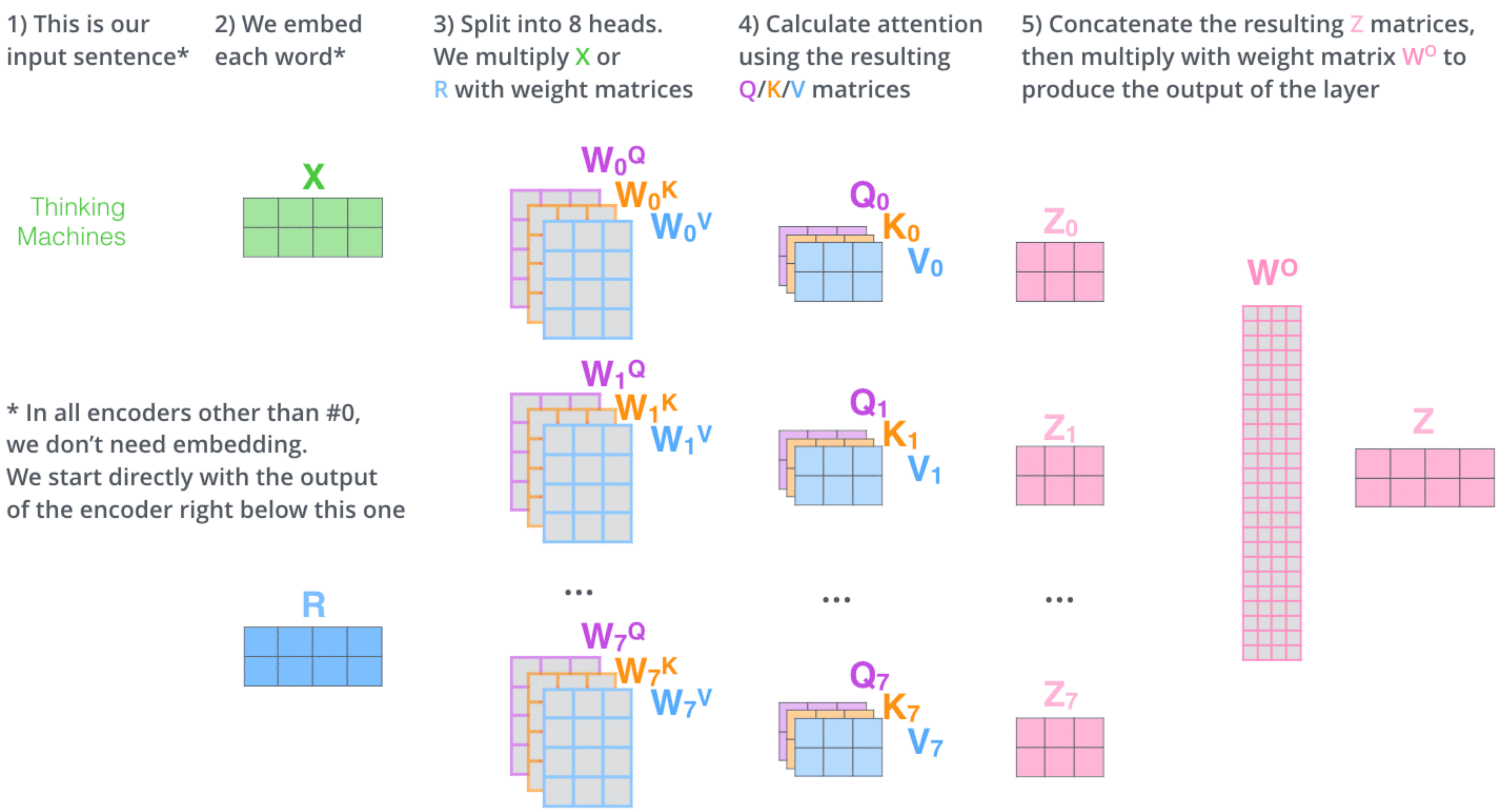


http://jalammar.github.io/images/t/transformer_attention_heads_z.png



http://jalammar.github.io/images/t/transformer_attention_heads_weight_matrix_o.png

Transformer – Multihead Attention



http://jalammar.github.io/images/t/transformer_multi-headed_self-attention-recap.png

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)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

Table 1 of "Attention Is All You Need", https://arxiv.org/abs/1706.03762