Transformer Translation Models

Jonathan May

October 19, 2022(Prepared for Fall 2022)

This will look very similar to a previous note set, 'Transformer Language Models.' That's because I wrote this one first and then adapted it to language models. I revisit the architecture here in the context of machine translation.

In 2017 some researchers at Google considered whether the recurrent part of RNNs/LSTMs was really that important at all in neural MT. In the paper 'Attention is all you need', they described their model, Transformer, which outperformed the state of the art at the time, at a variety of data points and at lower cost to train.

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**	$2.3 \cdot 10^{19}$	

Within a year or so Transformer models took over most of NLP as they were shown to be useful as language models and as feature sets for classification and structured prediction models. As I write this it's unclear if yet another model will prove even more compelling but these models seem quite good for now. All the images in these notes come from others' papers, lectures, blog posts, etc. Apart from the original transformer paper I recommend the illustrated transformer [1] or the annotated transformer[2].

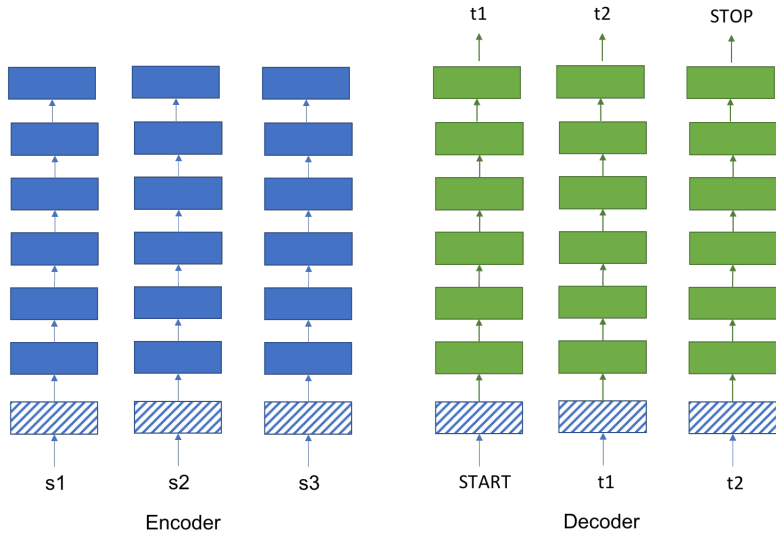
1 Base Model

We're going to cover the details in Transformer in various order, sort of from the outside in. To begin with, the overall shape is stacks of representations, conventionally of size 6, with one representation stack per word in the input and in the output. To begin with let's imagine each block is just a feed-forward network. Each word is embedded, then at each stage, it's passed through nonlinear transformation via ReLU. In fact there are two linear

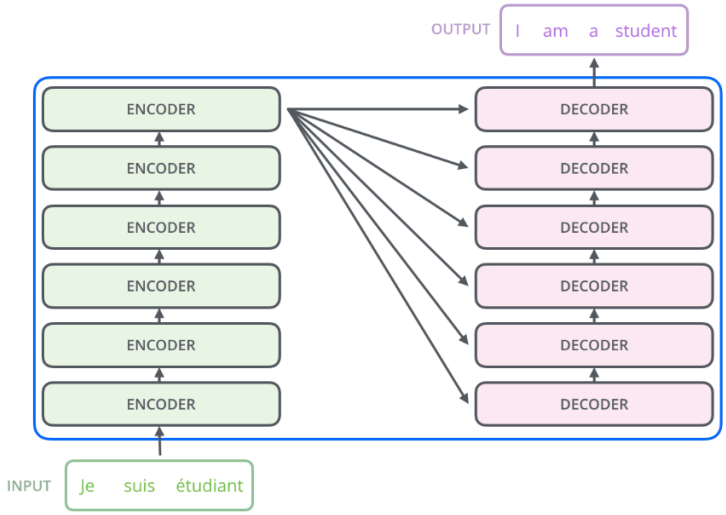
[1] `http://jalammar.github.io/illustrated-transformer/`

[2] `https://nlp.seas.harvard.edu/2018/04/03/attention.html`

transformations and one ReLU at each level. So if x is the embedding (or input from last layer), the output is $\max(\max(0, xW_1 + b_1)W_2 + b_2)$.[3]



And as you might imagine, attention is heavily involved. There are multiple kinds of attention but to begin with let's consider the 'normal' attention we've already discussed, i.e. from source to target words (from Illustrated transformer):

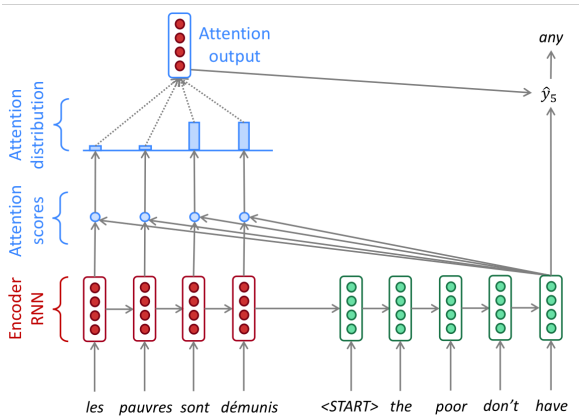


Notice that this attention is performed at *every layer* of the decoder.

1.1 Key, Query, Value Attention

Here's how we did attention before (fig from abi see):

[3] W_1 is (512×2048) and W_2 is (2048×512).



let $h_1, \ldots, h_N$ be hidden states of the encoder and s_t be the hidden state of the decoder. Then score vector $e^{(t)} = [s_t^T h_1, s_t^T h_2, \ldots, s_t^T h_N]$. Then distribution vector $\alpha^{(t)} = \text{softmax}(e^{(t)})$. Then make a linear combination of h; $a_t = \sum_{i=1}^{N} \alpha_i^{(t)} h_i$. That is then concatenated to s_t and used to predict.

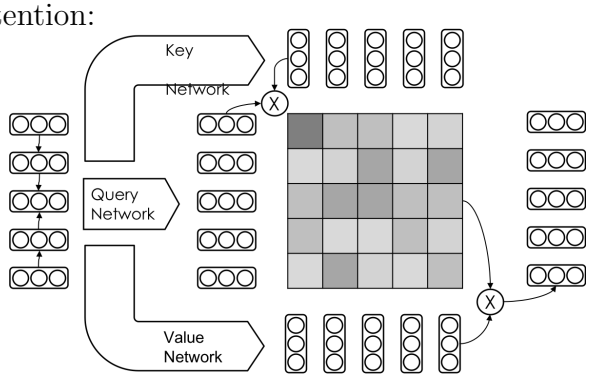
Transformer does it a bit differently. Instead of directly taking a dot product of s_t and each h_i, each of these is transformed linearly; s_t by a "query" matrix Q and h_i by a "key" matrix K. Then $s_t Q (h_i K)^T$ is the score; this is done for every h_i and turned into a distribution α_t by softmax.[4]

Now, instead of using α_t to linearly combine each h_i, the h_i are transformed again by a "value" matrix V. These are then linearly combined. That is then fed to the feed-forward unit. Attention, followed by feed forward, is one layer, and there are six.

1.2 self-attention

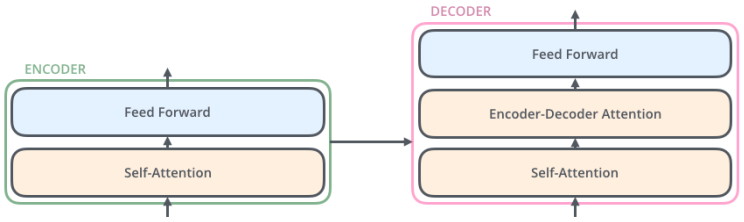
Why should attention be limited to target words looking at related source words? For h_t we can calculate $\alpha_{\text{self}}^{(t)} = h_t Q (h_i K)^T$ on the source side. On the target side we can almost do the same thing; we calculate $s_t Q (s_i K)^T$ but only for $i < t$; otherwise we'd be training on the future, which is not helpful at inference time! In practice a mask is used to prevent 'peeking' on the decoder during training.

I think the figure below by Alireza zareian nicely expresses the calculation for self-attention:



[4]Not quite. Actually it's $\frac{s_t Q (h_i K)^T}{\sqrt{|K|}}$, i.e. divide by the dimension of K. This keeps gradients from getting too small, per notes in the paper.

So each layer constitutes a number of *sublayers*. Jay Alammar of Illustrated Transformer has a nice figure:

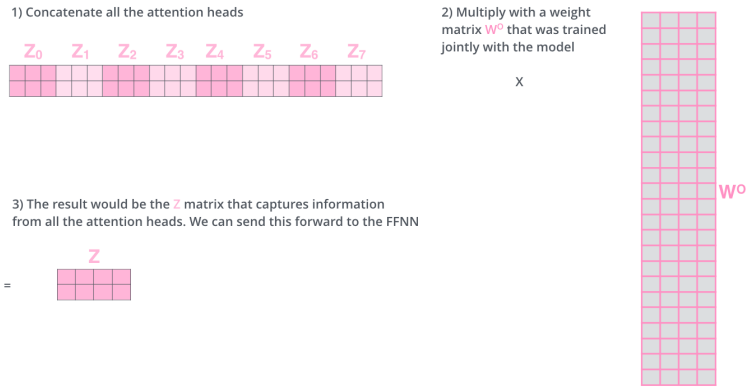


1.3 multi-head

Self attention can be viewed as a generalization of convolving kernels used in convolutional neural networks (CNNs). CNN filters, however, have dimension tied to the relative offset of adjacent inputs (words, pixels, hidden units) while the same Q, K, and V are applied to each input on a layer (different set for source, self-encoder, and self-decoder). Also, CNN filters do fixed combination, not a distributional interpolation. But the information sharing paradigm is very similar.

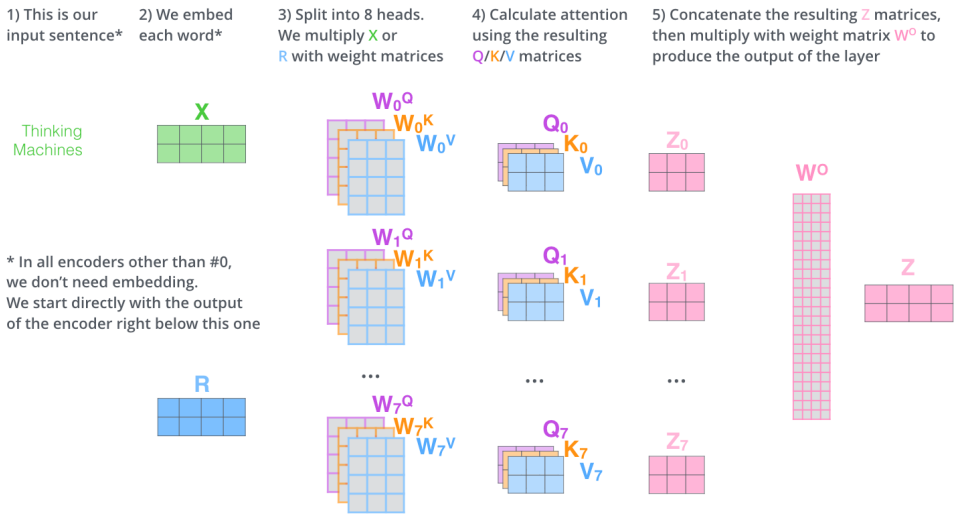
What are we actually doing when we do self-attention? In source-attention the semantics seemed clear; we're looking at corresponding words to be translated. But in self attention that's not the case. We are probably combining some semantic and syntactic coordination.

But there are different aspects of information we might want to attend. It seems odd to distill them down into a single (Q, K, V) triple. And since we noticed the similarity to CNNs we can use a technique used in CNNs: multiple filters! Indeed, we actually do attention in one place many[5] times with a different learned (Q, K, V) set for each time; each attention that is learned is called a 'head'. Rather than use e.g. max-pooling or mean-pooling as is often done in CNN, Transformer instead does a linear projection of the heads (Alammar):



Here is attention all together (Alammar):

[5] eight



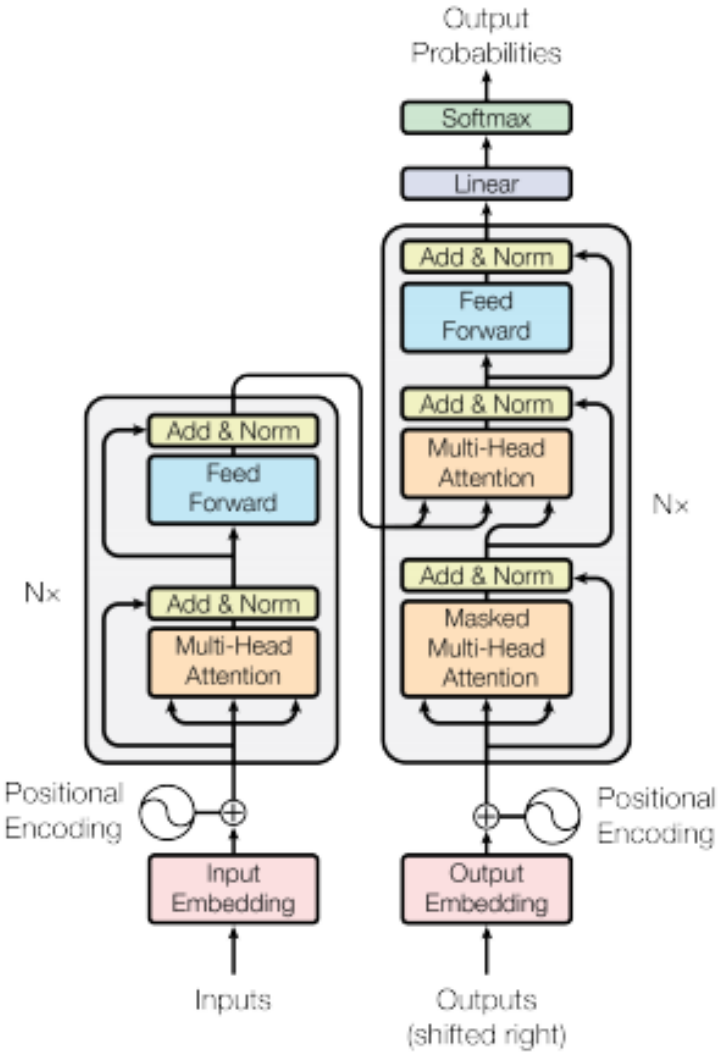
1.4 Residual Connections and Layer Norm

In the story we've told so far, data enters a layer, is combined with information from all the other words in the sentence (so far, for decoder) with self attention, if the decoder, is then combined with all the words in the source, then is projected through a feed-forward layer. So if we call the input to a layer x and the self-attention, source-attention, and feed-forward sublayers functions $self$, $source$, and ff, the output on the encoder is $ff(self(x))$ and on the decoder is $ff(source(self(x)))$. This seems like a good opportunity for the information at that position to get lost; self-attention could decide not to attend to the self! A well-known technique called *residual connections* is used; in each case we simply add the input back again after each sublayer. This is only done per-sublayer.

We introduce some sub-results: on the encoder we calculate $x' = self(x) + x$. Then the output is $ff(x') + x'$. Similarly on the decoder we calculate x' as before, then $x'' = source(x') + x'$ and the output is $ff(x'') + x''$.

OK but we don't actually even use the original x or the other intermediates without modification either! Instead we use a technique called 'layer norm' [**?**] which essentially modifies each item by subtracting the mean and divides by the standard deviation over the vector.[6] So in fact $x' = self(norm(x)) + x$, and for the decoder, $x'' = source(norm(x')) + x'$; For the encoder, the output is $ff(norm(x')) + x'$, and for the decoder it is $ff(norm(x'')) + x''$. This is not well-described in the original paper but is what has been uncovered (by TG and others). We now know almost everything in this handy diagram from the original paper:

[6] it's a little more complicated than that but this is already rather in the weeds.


Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Nx
Nx
Positional Encoding
Positional Encoding
Input Embedding
Output Embedding
Inputs
Outputs (shifted right)