Attention Is All You Need

a.k.a. «Sequence models are overrated»

N. Popov

Faculty of Computer Science
Higher School of Economics

Mar. 23, 2018

Table of Contents

Sequence processing tasks

When there's a sequence and you need to process it

We all know them, we all love them

- Machine translation:
  Translating one sequence of words into another.
- Sentiment analysis:
  Classification of sequences
- Text-to-speech and speech-to-text:
  Translation with some caveats
- And many, many more.
- We will focus on sequence-to-sequence tasks in this presentation.

Machine translation is nontrivial

- Natural language is complex and complicated;
- There is about a khgjillion ways to say the same thing differently;
- Sentences can have wildly varying lengths;
- Connected words don't have to be close;
- And yet the word order is crucial;
- The connections between words are not always straightforward
- Nor are their meanings.

Try translating this:

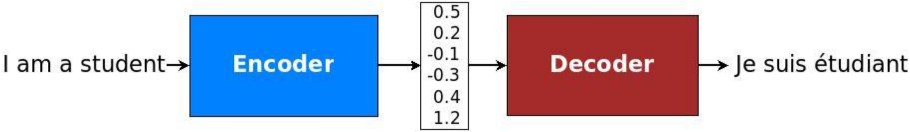
- “The horse raced past the barn fell.”
- “The old man the boat.”
- “The complex houses married and single soldiers and their families.”

How do we cope with this?

- Hard-coded rule-based translation is horrible.
- Statistical systems are better, but still quite bad.
- Deep learning to the rescue!

Generally, Deep-Learning based models consist of an encoder and a decoder:

- Encoder encodes the word sequence into some internal representation;
- Decoder decodes that representation inte a sequence of words in another language.



RNNs: the original seq2seq

By this point you should be familiar with them.

- Encoder iterates over the input sequence, building some internal vector representation of it;
- Decoder decodes that vector into a full sequence in another language.

Pros and cons:

+ Can adequately model sequences;
+ Can model (medium-)long-term dependencies;
– Not very good at very long-term dependencies;
– Is forced to compress a whole sentence into a single vector;
– Non-parallelizable; slow.

RNNs with attention

- Encoder iterates over the input sequence, generating a sequence of representations;
- Decoder selects and combines embeddings at each step to produce a full sequence in another language.

Pros and cons:

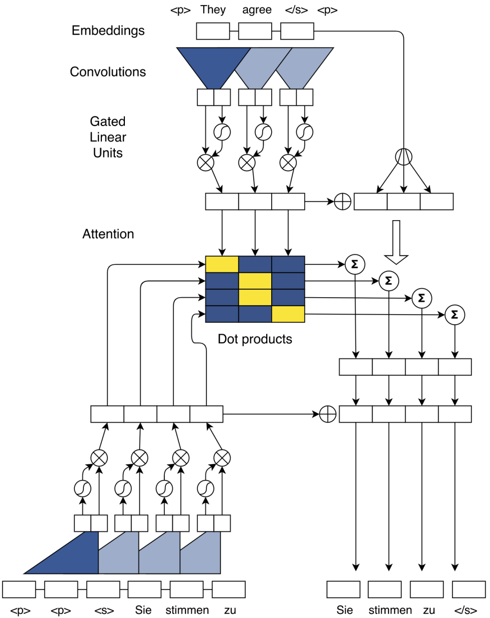
+ Are better at long-term dependencies between input and output;
+ No longer compresses; the latent space is much larger;
− Still non-parallelizable and slow.

CNNs

Encoder and decoder are CNNs instead of RNNs.
Pros and cons:

+ Are even better at long-term dependencies, since there is no bottleneck in the form of RNN's inner state;
+ Parallelizable; fast;
– Since CNN neurons have limited perception field, networks have to be very deep or use other tricks.
– For the same reason, there always is a maximum distance where model can "connect" words.

ConvS2S architecture



Main trends

- RNNs are good as an initial "baseline", but they have major drawbacks;
- Everything is better with attention;
- Additionally, we would like as much parallelization as possible;
- And no bottlenecks;
- And no limits on sequence length.

Main trends

- RNNs are good as an initial "baseline", but they have major drawbacks;
- Everything is better with attention;
- Additionally, we would like as much parallelization as possible;
- And no bottlenecks;
- And no limits on sequence length.

What can we use?

- RNNs are out of the question — they have major bottlenecks;
- CNNs are also out — they have limited perception;
- ???
- Can we make a sequence model without actually using anything sequential?

Main trends

- RNNs are good as an initial "baseline", but they have major drawbacks;
- Everything is better with attention;
- Additionally, we would like as much parallelization as possible;
- And no bottlenecks;
- And no limits on sequence length.

What can we use?

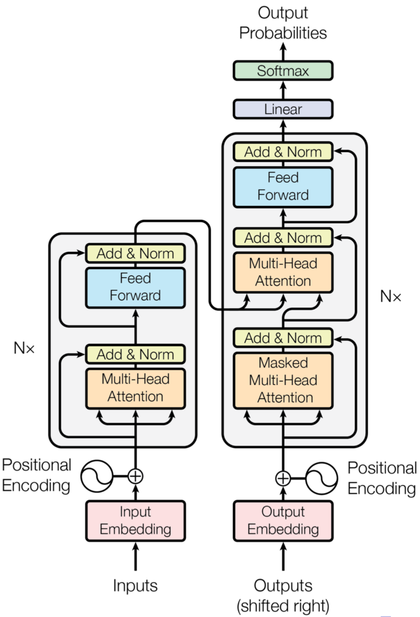
- RNNs are out of the question — they have major bottlenecks;
- CNNs are also out — they have limited perception;
- ???
- Can we make a sequence model without actually using anything sequential?
- Yes, and attention is our best friend.

Enter Transformer

Transformer is a work of several researchers, mostly from Google.
Main features:

- Consists entirely of fully-connected layers and attention;
- Attention is applied not only between encoder and decoder, but within them as well;
- Since fully-connected layers have no way of knowing the position in the sequence, *positional embeddings* are employed;
- An entirely new attention mechanism: *Multi-head attention*;
- Residual connections, layer normalization, dropouts, label smoothing...

Transformer architecture

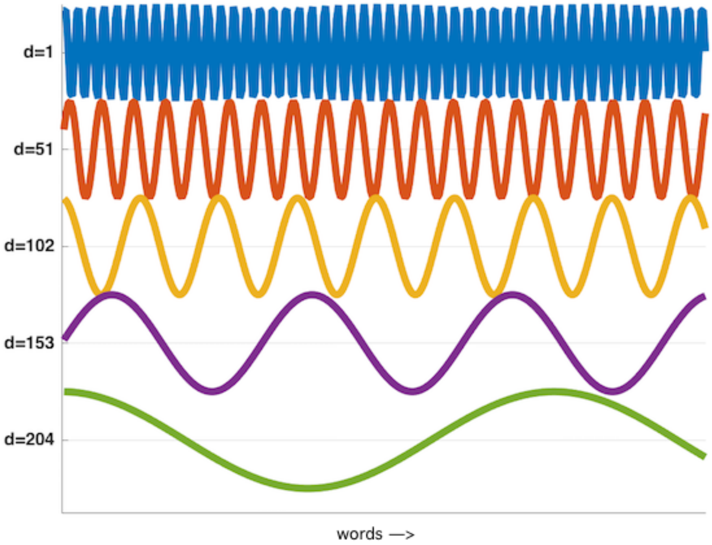


Input

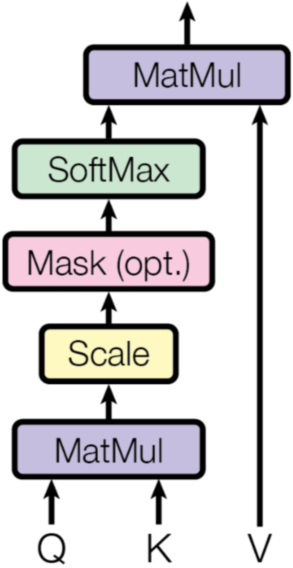
- Input and output are preprocessed using BPE (byte-pair encoding); the words are split into sub-word tokens, which are then fed into the network.
- In order to give the model a sence of location, positional embeddings are used:
  If $\mathbf{w} = (w^{(1)}, w^{(2)}, \ldots, w^{(\ldots)})$ is a sequence of word embeddings, then $\mathbf{x} = (w^{(1)} + p^{(1)}, w^{(2)} + p^{(2)}, \ldots)$ is a sequence of positionally embedded words;

$$\begin{cases} p_{2i}^{(k)} = \sin\left(\dfrac{k}{10^{\frac{8i}{d_{\text{model}}}}}\right) \\ p_{2i+1}^{(k)} = \cos\left(\dfrac{k}{10^{\frac{8i}{d_{\text{model}}}}}\right) \end{cases}$$

Positional embeddings



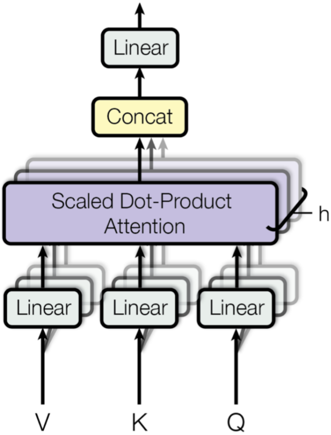
Attention



Basic building block of Multihead attention is *Scaled dot-product attention*:

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

Attention

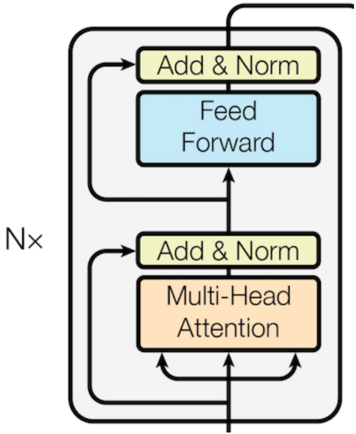


Multihead attention is simply a concatenation of several slightly different scaled dot-product attentions:

$$\text{Multihead}(Q, K, V) = (H_1, \ldots, H_h)^T W^O$$

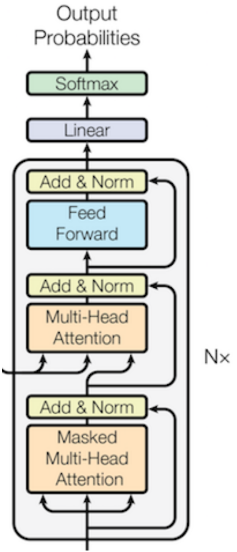
$$H_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

Encoder



- Encoder consists of N identical blocks;
- Each block consists of an attention layer and a feedforward layer with residual connections inbetween;
- The keys, values and queries all come from the previous block; this is *self-attention*;

Decoder

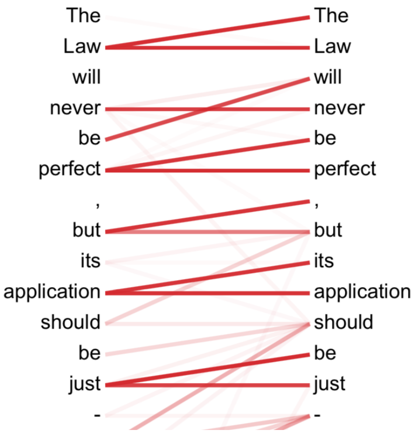


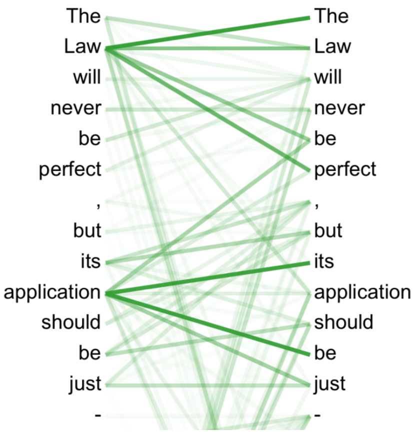
- Decoder consists of N identical blocks;
- Each block consists of:
    - Self-attention layer;
    - Attention over encoder outputs;
    - Feedforward layer;
- In the attention over encoder outputs, keys and values come from encoder, but queries are generated by the decoder.

Why does this work better?

- This architecture is easily parallelized: the only sequential part is how every word of output depends on all previous words;
- It does not employ any unusual operations; most operations are easily parallelized and heavily optimized matrix multiplications;
- The path between any two sequence items has constant length, as opposed to linear in RNNs and CNNs.
- Heavy dependence on attention makes results more interpretable, possibly giving insight into how the model "looks" at the text.

Self-attention

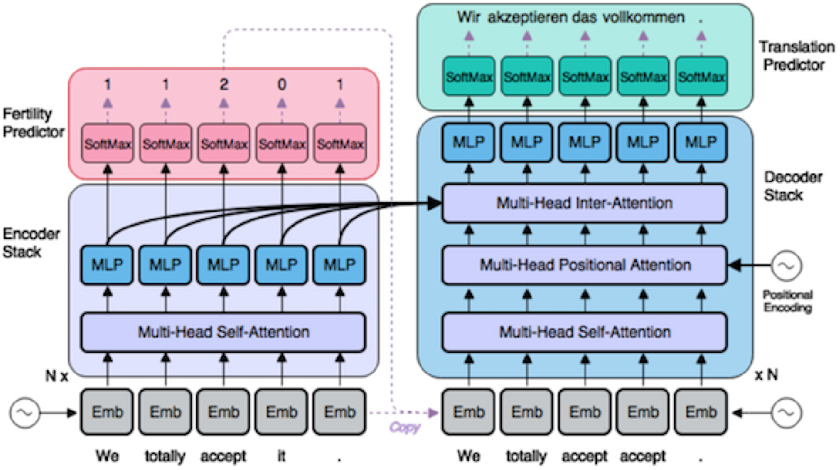




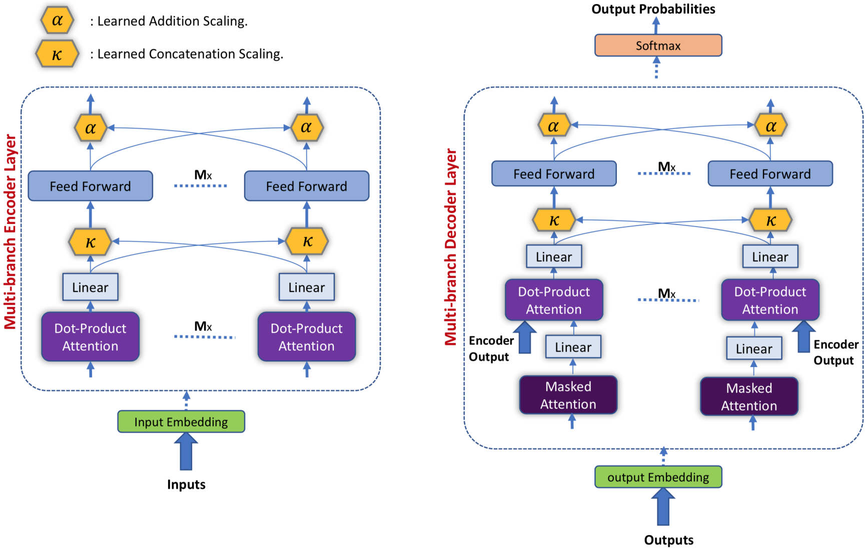
Results

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	$\mathbf{3.3 \cdot 10^{18}}$	
Transformer (big)	**28.4**	**41.8**	$2.3 \cdot 10^{19}$	

Non-autoregressive Transformer



Weighted Transformer



References I

Attention Is All You Need
Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin
arXiv:1706.03762 [cs.CL]

Weighted Transformer Network for Machine Translation
Karim Ahmed, Nitish Shirish Keskar, Richard Socher
arXiv:1711.02132 [cs.AI]

Non-Autoregressive Neural Machine Translation
Jiatao Gu, James Bradbury, Caiming Xiong, Victor O.K. Li, Richard Socher
arXiv:1711.02281 [cs.CL]

References II

Convolutional Sequence to Sequence Learning
Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, Yann N. Dauphin
arXiv:1705.03122 [cs.CL]