Attention is All You Need

Andrew Drozdov

September 19, 2017

What?

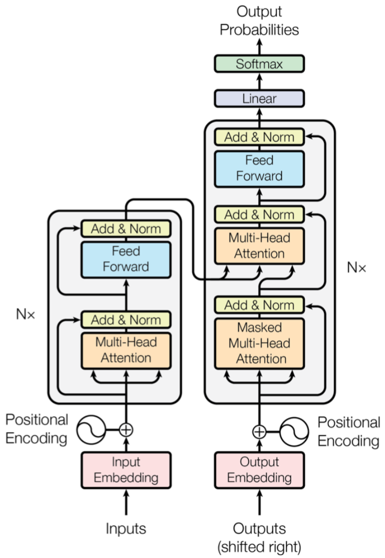


Figure 1: The Transformer - model architecture.

Why? (Architecture Perspective)

- Many NLP tasks have SotA set by LSTM or GRU, models with sequential dependencies making them difficult to paralellize.
- There are other popular architectures lately:
  - QRNN [1] / SRU [2]
  - CNN [3]

  But they still usually have some sequential dependencies.
- The model in AIAYN has no sequential dependencies.
- Attention-only model does exist, but not for decoding.

[1] `https://www.salesforce.com/products/einstein/ai-research/neural-network-building-block-accurate-understanding/`

[2] `https://github.com/taolei87/sru`

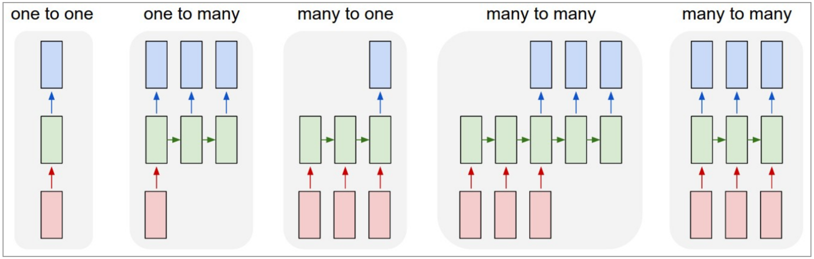
[3] `https://github.com/facebookresearch/fairseq`

Why? (Performance Perspective)

- Transformer Network can be used for many tasks:
  - WMT 14 (En to Ge). 28.4 BLEU (+2)
  - WMT 14 (En to Fr). 41.0 BLEU (single model, fast/easy to train)
  - Constituency Parsing

  Some of these numbers are more impressive than seen in some similar papers.

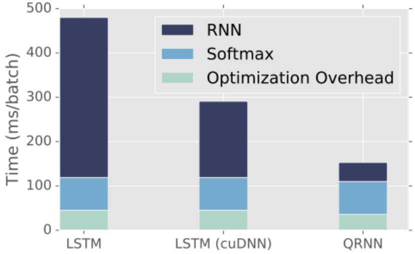
Background (RNNs)



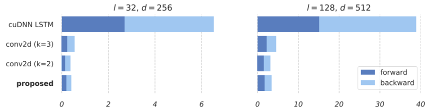
http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Background (LSTM/GRU and motivation for QRNN)

- Hochreiter and Schmidhuber. 1997. LSTM.
- Cho et al. 2014. GRU.



QRNN



SRU

Model: Transform Network (Encoder and Decoder)

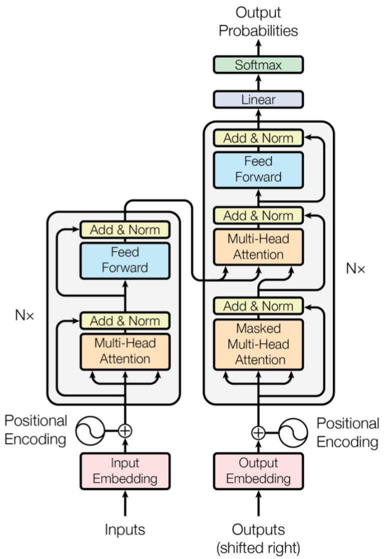
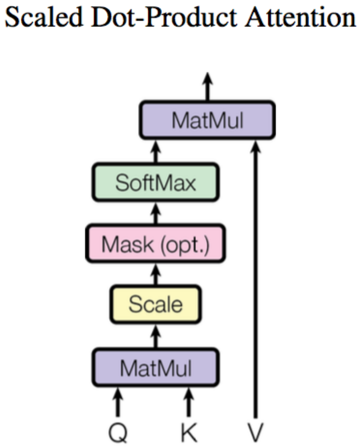


Figure 1: The Transformer - model architecture.

Scaled Dot-Product Attention



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

- Scaling is the same as Softmax plus Temperature. Higher Temperature brings probabilities closer to uniform.

Scaled Dot-Product Attention (Pytorch)

Serial.

```
for query in queries:
  attn = query.view(1, D) * keys.view(K, D)
  attn = attn.sum(1) / scale
  attn = softmax(attn)
```

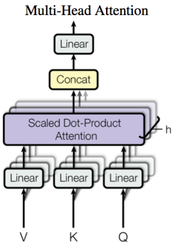
Parallel.

```
attn = queries.repeat(1, K).view(Q * K, D) * keys.repeat(Q, 1)
attn = attn.sum(1) / scale
attn = softmax(attn)
```

Parallel with broadcasting.

```
attn = querys.view(Q, 1, D) * keys.view(1, K, D)
attn = attn.view(Q * K, D).sum(1) / scale
attn = softmax(attn)
```

Multi-Head Attention



$$\mathrm{MultiHead}(Q,K,V) = \mathrm{Concat}(\mathrm{head_1},...,\mathrm{head_h})W^O$$
$$\text{where } \mathrm{head_i} = \mathrm{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

Where the projections are parameter matrices $W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$ and $W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$.

In this work we employ $h = 8$ parallel attention layers, or heads. For each of these we use $d_k = d_v = d_{\text{model}}/h = 64$. Due to the reduced dimension of each head, the total computational cost is similar to that of single-head attention with full dimensionality.

Perform attention multiple times (almost like a dynamic attention “kernel”).

Other Tricks

1. Masking in Softmax is done with $-\infty$.
2. The parameters are shared between the embedding layers *and the pre-softmax layer*.
3. Positional encodings give a weak sense of ordering (similar was done in Parikh et al.).

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

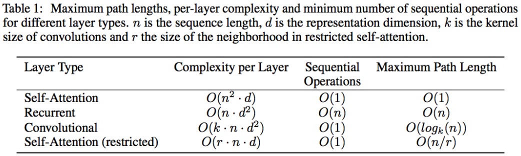


Table 1: Maximum path lengths, per-layer complexity and minimum number of sequential operations for different layer types. n is the sequence length, d is the representation dimension, k is the kernel size of convolutions and r the size of the neighborhood in restricted self-attention.

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

Training

- WMT 14 (En to Ge). 4.5M pairs. BPE w. 37k vocab.
- WMT 14 (En to Fr). 36M pairs(?). Wordpiece w. 32k vocab.
- Batch Size: 25k.
- 8 GPUs (x 3584 cores; 16GB; 720GBps)
- Small = 0.4s / step. 100k steps (12 hours). Big = 1s / step. 300k steps (3.5 days).
- Adam with learning rate warmup and decay.
- Regularization...

Training (Regularization)

- Residual Dropout
- Attention Dropout
- Label Smoothing
- Layer Normalization [4][5]

[4] https://github.com/pytorch/pytorch/issues/1959

[5] Ba, Kiros, Hinton. 2016. https://arxiv.org/abs/1607.06450

Results: WMT

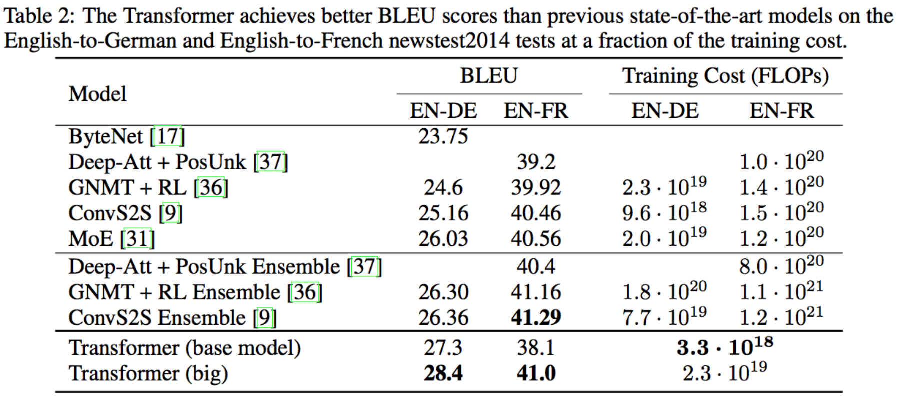


Table 2: The Transformer achieves better BLEU scores than previous state-of-the-art models on the English-to-German and English-to-French newstest2014 tests at a fraction of the training cost.

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [17]	23.75			
Deep-Att + PosUnk [37]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [36]	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 [31]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [37]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [36]	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**	**41.0**	$2.3 \cdot 10^{19}$	

Results: Variations

Table 3: Variations on the Transformer architecture. Unlisted values are identical to those of the base model. All metrics are on the English-to-German translation development set, newstest2013. Listed perplexities are per-wordpiece, according to our byte-pair encoding, and should not be compared to per-word perplexities.

	N	d_{model}	d_{ff}	h	d_k	d_v	P_{drop}	ϵ_{ls}	train steps	PPL (dev)	BLEU (dev)	params $\times 10^6$
base	6	512	2048	8	64	64	0.1	0.1	100K	4.92	25.8	65
(A)				1	512	512				5.29	24.9	
				4	128	128				5.00	25.5	
				16	32	32				4.91	25.8	
				32	16	16				5.01	25.4	
(B)					16					5.16	25.1	58
					32					5.01	25.4	60
(C)	2									6.11	23.7	36
	4									5.19	25.3	50
	8									4.88	25.5	80
		256			32	32				5.75	24.5	28
		1024			128	128				4.66	26.0	168
			1024							5.12	25.4	53
			4096							4.75	26.2	90
(D)							0.0			5.77	24.6	
							0.2			4.95	25.5	
								0.0		4.67	25.3	
								0.2		5.47	25.7	
(E)	positional embedding instead of sinusoids									4.92	25.7	
big	6	1024	4096	16			0.3		300K	**4.33**	**26.4**	213

Results: PTB

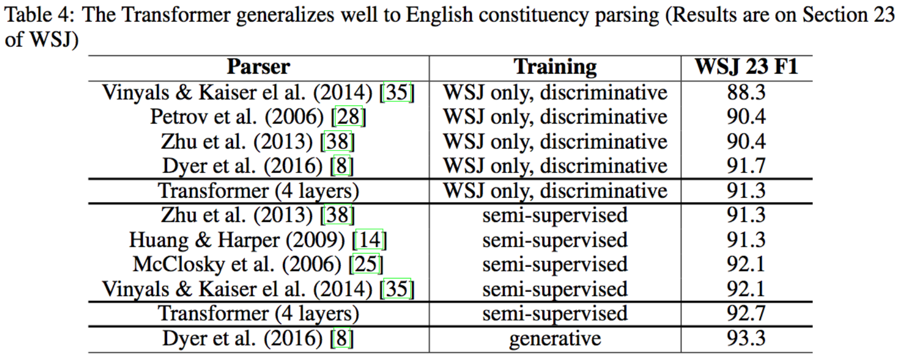


Table 4: The Transformer generalizes well to English constituency parsing (Results are on Section 23 of WSJ)

Parser	Training	WSJ 23 F1
Vinyals & Kaiser el al. (2014) [35]	WSJ only, discriminative	88.3
Petrov et al. (2006) [28]	WSJ only, discriminative	90.4
Zhu et al. (2013) [38]	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) [38]	semi-supervised	91.3
Huang & Harper (2009) [14]	semi-supervised	91.3
McClosky et al. (2006) [25]	semi-supervised	92.1
Vinyals & Kaiser el al. (2014) [35]	semi-supervised	92.1
Transformer (4 layers)	semi-supervised	92.7
Dyer et al. (2016) [8]	generative	93.3

Conclusion

If you use the right tricks, you can get SotA and decrease your computational cost.