Attention is all you need

- ARUN

Motivation

- RNN's => Sequential computation
- Difficult to parallelize => Memory constraints
- Increased Computational efficiency using
  - Factorization techniques
  - Conditional computation
- Reduced computation => Bytenet & ConvS2S. But operations to relate signals from distant position grows linearly and logarithmically
- Transformer => Constant number of operations
- Reduced effective resolution => Multi head attention

Transformer Model

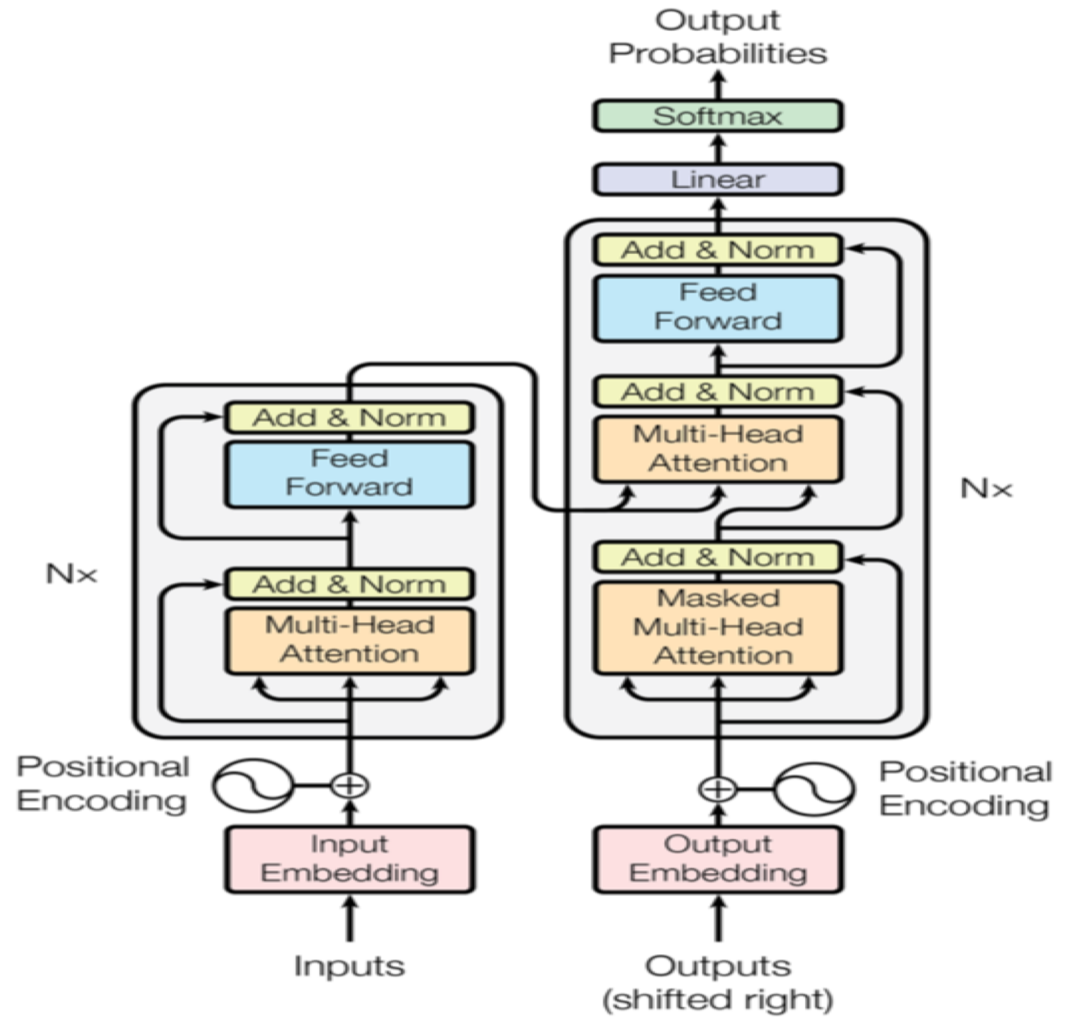


Figure 1: The Transformer - model architecture.

Attention

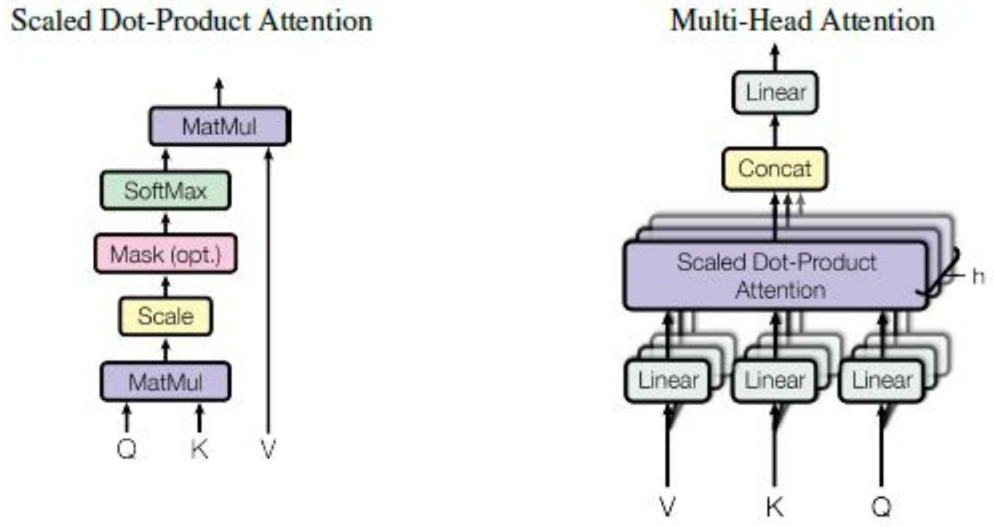


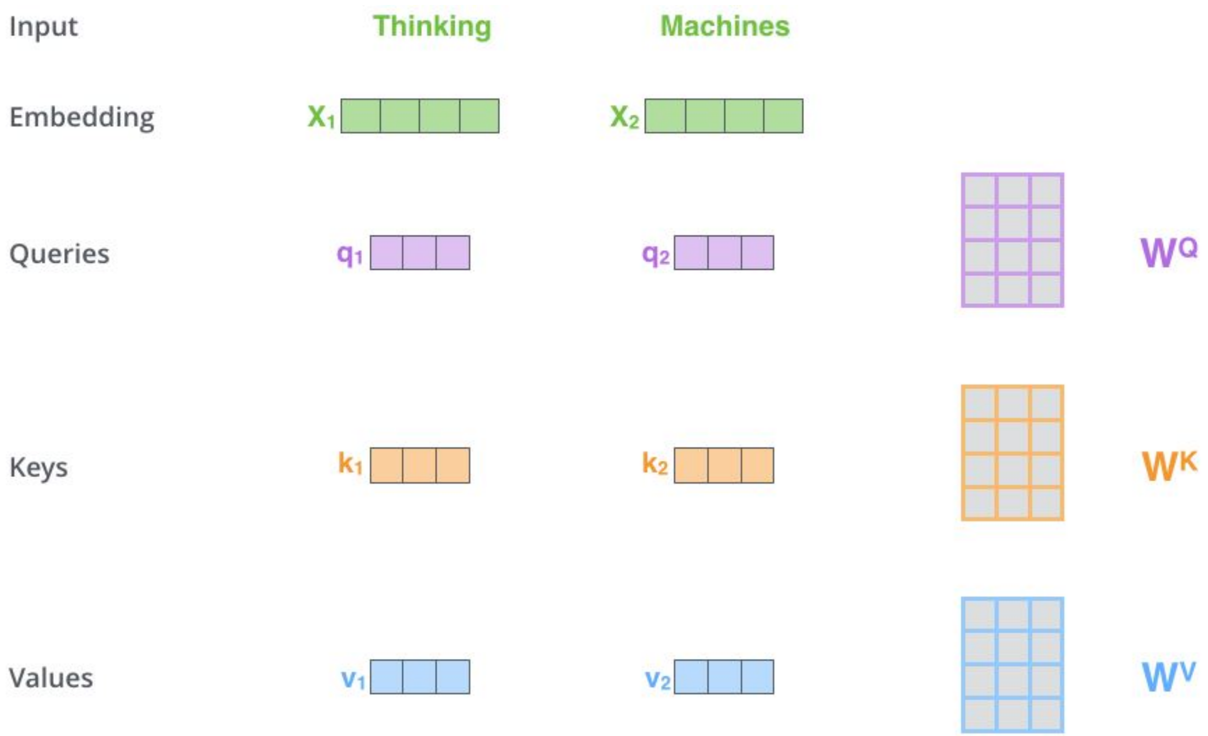
Figure 2: (left) Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several attention layers running in parallel.

Types of Attention

- Additive Attention vs Dot product attention
- Scaled Dot product attention
    - Faster and space efficient (optimized matrix multiplication code)
- Scaling to increase performance for large values

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

Key,Value,Query



/lultiplying x1 by the WQ weight matrix produces q1, the "query" vector associated with that word. We end up creating a "query", a "key", and a "value" projection of each word in the input sentence.

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 $W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$.

- dmodel = 512 ; dk = dv = dmodel/h
- Jointly attend to information from different representation subspaces at different positions
- Reduced dimension of each head, the total computational cost is similar to single-head attention with full dimensionality

Applications

1) Encoder – Decoder Attention

=> Key, Value from Input & Query from Output

2) Encoder - Self attention

=> Key,Value,Query all from Input

3) Decoder - Masked Self attention

=> Key, Value, Query all from Output

=> Prevent Leftward information flow , autoregressive property

=> Masking out all values in the input of the Softmax which correspond to illegal connections

Positional Encoding

- sine and cosine functions of different frequencies
- Can also use learned positional embeddings
- sinusoidal over learned positional embeddings as it allows model to extrapolate

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

Model

- Embeddings + Positional Encoding
- Encoder

    - Multi head attention + Fully Positionwise FFN

    - Implements residual connections with batch normalization
- Decoder

    - Masked Multi Head Attention : to prevent positions from attending to subsequent positions

    - Encoder Decoder Multi Head Attention + Fully Positionwise FFN
- Two Linear Layers & Softmax

Why Self Attention

1) Total computational complexity per layer.

2) Amount of computation that can be parallelized, as measured by the minimum number of sequential operations required.

3) The third is the path length between long-range dependencies in the network.

4) Computational Complexity

- Self attention better than RNN for sequences smaller than representation dimensionality

- Convolution expensive than RNN. Separable conv decrease complexity but still equal to Self Attention + Positionwise FFN

Advantages of Transformer

- More parallelizable
- Path length is less
- Eradicates model forgetting problems
- Less time to train
- Output of one step is for one sample
- No multi step backpropagation.
- No recurrence

Path Length

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

Translation

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

Model 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.

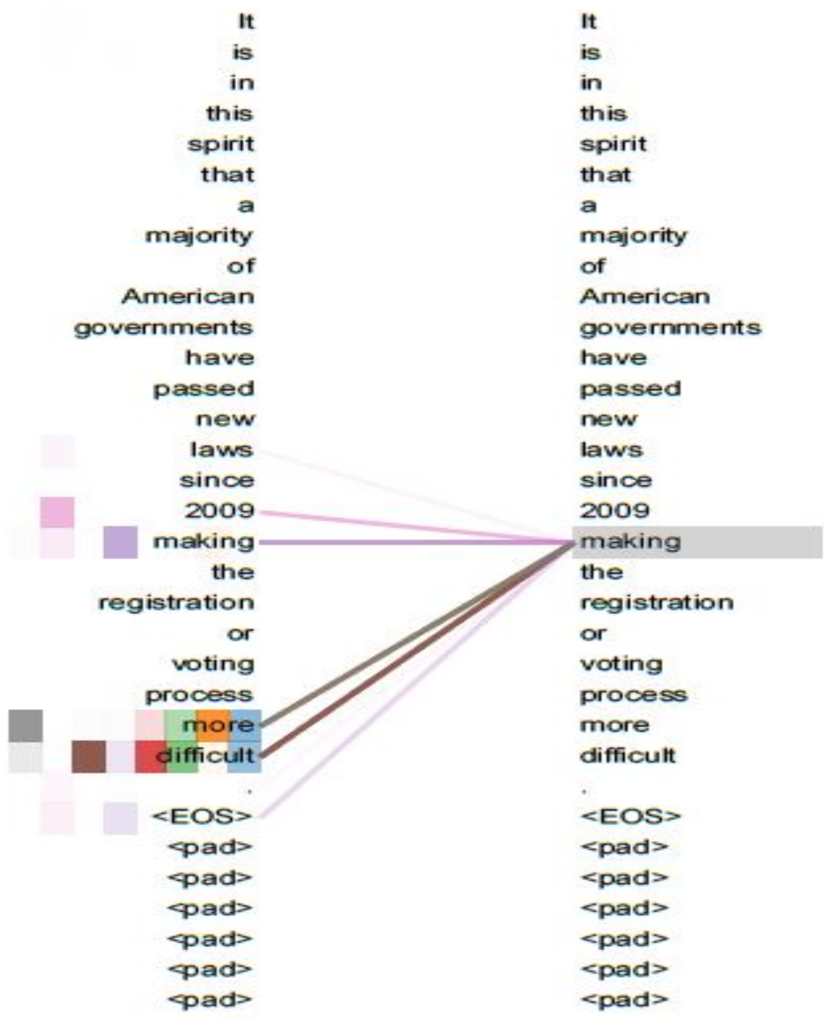
	N	d_{model}	d_{ff}	h	d_k	d_v	P_{drop}	ϵ_{ls}	train steps	PPL (dev)	BLEU (dev)	params $\times 10^6$
base	6	512	2048	8	64	64	0.1	0.1	100K	4.92	25.8	65
(A)				1	512	512				5.29	24.9	
				4	128	128				5.00	25.5	
				16	32	32				4.91	25.8	
				32	16	16				5.01	25.4	
(B)					16					5.16	25.1	58
					32					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

Constituency Parsing

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) [37]	WSJ only, discriminative	88.3
Petrov et al. (2006) [29]	WSJ only, discriminative	90.4
Zhu et al. (2013) [40]	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) [40]	semi-supervised	91.3
Huang & Harper (2009) [14]	semi-supervised	91.3
McClosky et al. (2006) [26]	semi-supervised	92.1
Vinyals & Kaiser el al. (2014) [37]	semi-supervised	92.1
Transformer (4 layers)	semi-supervised	92.7
Luong et al. (2015) [23]	multi-task	93.0
Dyer et al. (2016) [8]	generative	93.3

Attention Visualisation



Practical Techniques

Regularization

- Residual dropout
- Label smoothing

Practical Techniques

1) Choosing a good number of attention heads (both too little & too many heads hurt performance)
2) Applying dropout to the output of each sub-layer as well as the attention outputs
3) Using a sufficiently large key size d_k for computing attention

References

- https://www.youtube.com/watch?v=iDulhoQ2pro&t=5s
- https://hub.packtpub.com/paper-in-two-minutes-attention-is-all-you-need/
- https://nlp.stanford.edu/seminar/details/lkaiser.pdf
- http://nlp.seas.harvard.edu/2018/04/03/attention.html
- https://medium.com/@sharaf/a-paper-a-day-24-attention-is-all-you-need-26eb2da90a91
- https://www.reddit.com/r/MachineLearning/comments/89uy75/p_the_annotated_transformer_linebyline_pytorch/
- https://github.com/guillaume-chevalier/Linear-Attention-Recurrent-Neural-Network/blob/master/AnnotatedMultiHeadAttention.ipynb
- http://mlexplained.com/2017/12/29/attention-is-all-you-need-explained/
- https://ai.googleblog.com/2017/08/transformer-novel-neural-network.html
- http://jalammar.github.io/illustrated-transformer/