ATTENTION IS ALL YOU NEED

Vaswani et al

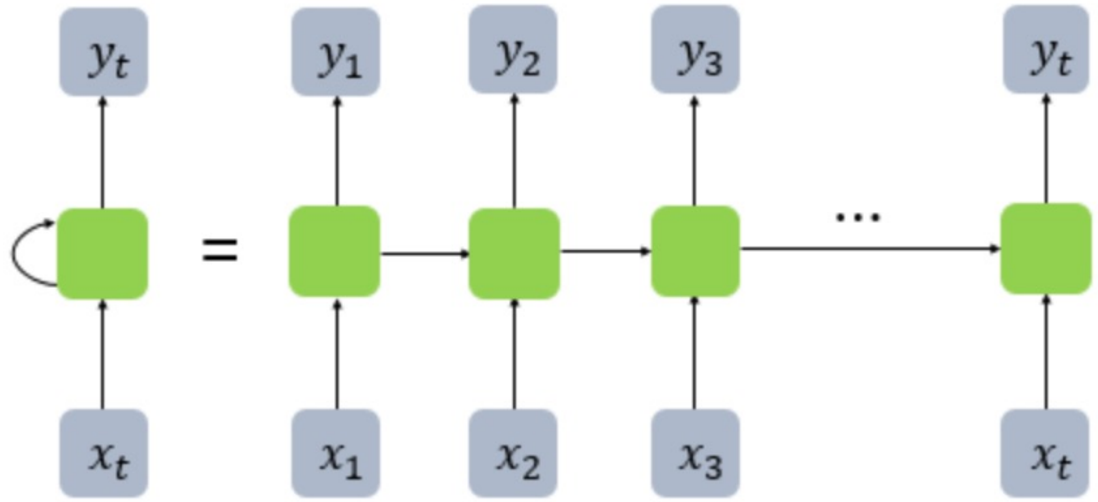
31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

<Transformer>

based on solely an attention mechanism(no recurrence, no convolution)

more parallelizable, requiring less time than recurrent model and convolutional model

<Recurrent models>



Memory cell(RNN cell): memorize the previous value

Hidden state: the values that the memory cells send to next memory cell or output layer

<Recurrent models>

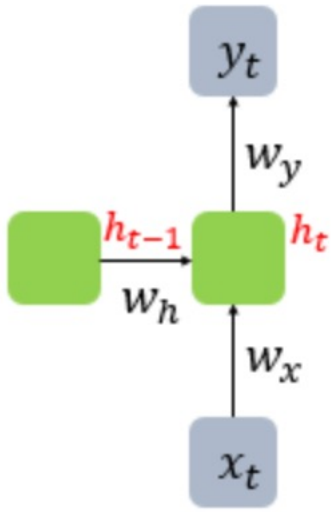
The unit of input, output of cells: word vector

One to many: image captioning

Many to one: sentiment classification

Many to many: translation, tagging

<Recurrent models>



$$h_t = tanh(W_x x_t + W_h h_{t-1} + b)$$
$$y_t = f(W_y h_t + b)$$

Two weights are used to calculate h_t

f is nonlinear activation function

Deep RNN, Bidirectional RNN

Hard to parallelization: memory constraints limit batching, Gradient vanishing and gradient exploding

02. Model Architecture

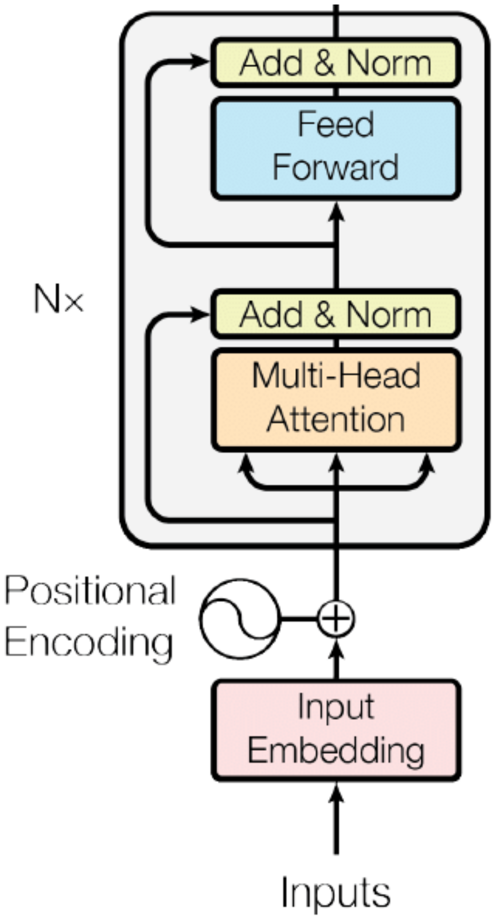
<Basic architecture>

Symbol architecture (X1 ,X2, …, Xn) | ENCODER | Sequence of continuous representation (Z1, Z2, …, Zn) | DECODER | Output sequence (Y1, Y2, …, Yn)

At each step, the model is auto regressive

※ auto regressive: consuming the previous generated symbol as additional input of the text

<Encoder>



- Positional Encoding
- Multi-Head Attention
- Add & Norm
- Feed Forward
- Residual Connection

<Encoder: positional encoding>

To represent the order of the sequence, we must inject information that contains the position of the tokens in the sequence

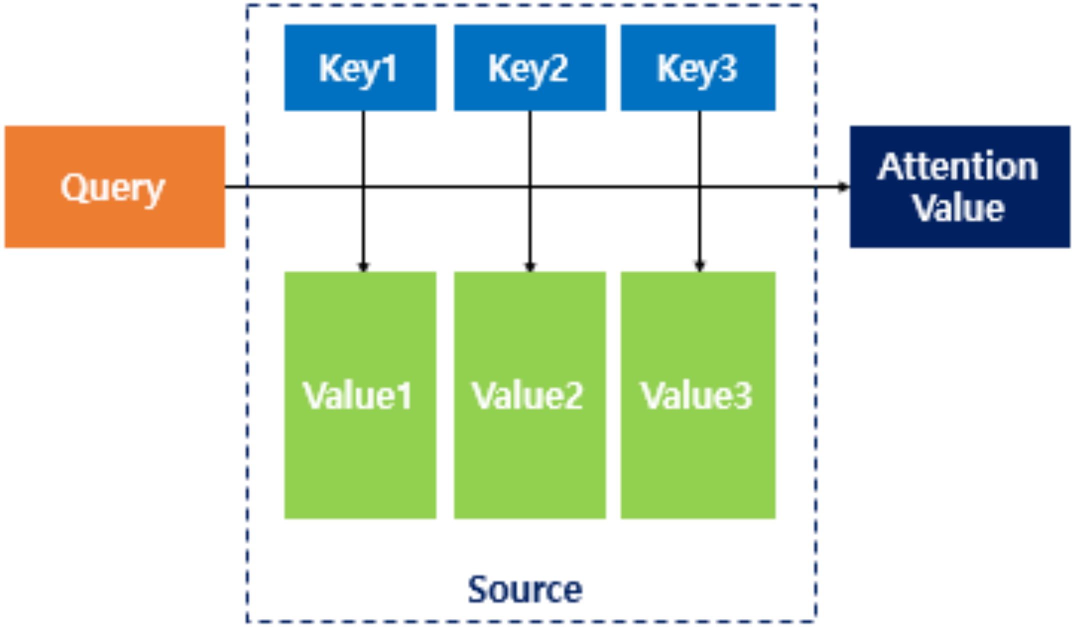
- Use sinusoidal functions to make different values per position and dimension

- Same dimension with input embeddings so that we can use the summation of those

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

pos = position
i = dimension

<Encoder: attention>

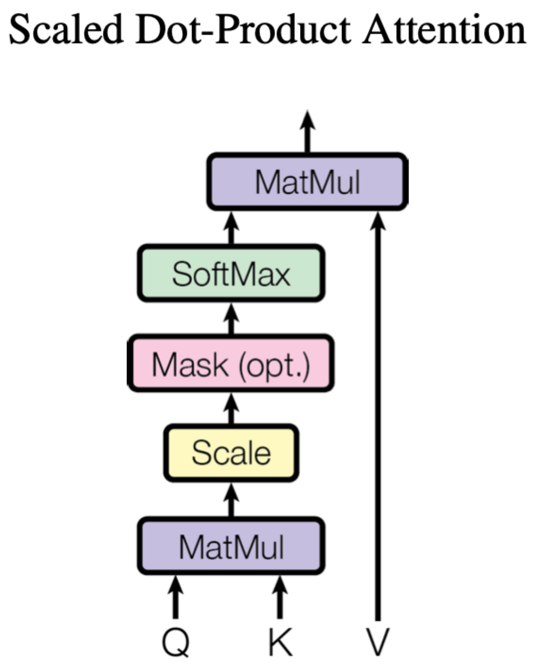


Mapping a query and a set of key-value pairs to an output

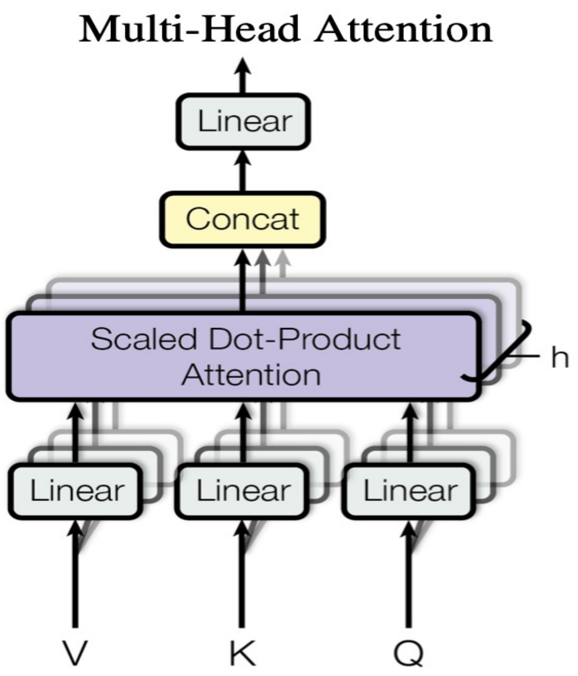
1. Get similarities between Query and Key
2. Use normalized similarities as weights
3. The weights are multiplied with value vectors
4. Add all Values that contain weights to get attention value

Attention function is the process that calculating the weights of each word in the input sequence
-> we can represent the importance of each word in output

<Encoder: attention>

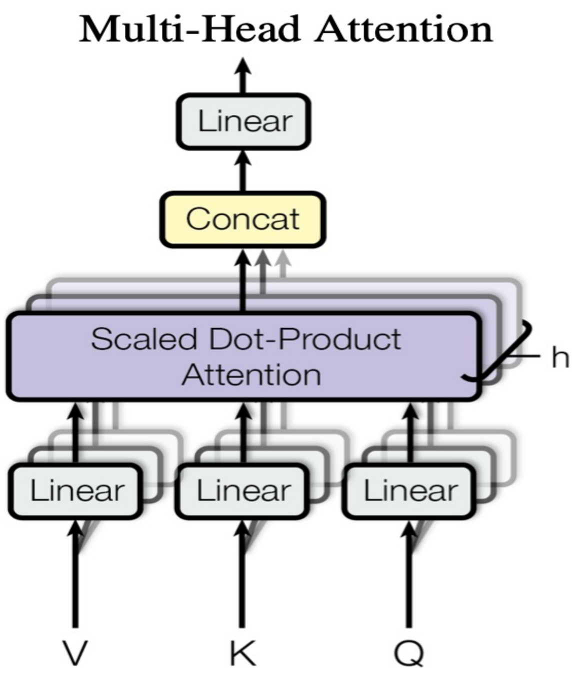


$$\mathrm{Attention}(Q, K, V) = \mathrm{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$



$$\mathrm{MultiHead}(Q, K, V) = \mathrm{Concat}(\mathrm{head}_1, ..., \mathrm{head}_h)W^O$$
$$\text{where } \mathrm{head}_i = \mathrm{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

<Encoder: multi-Head attention>

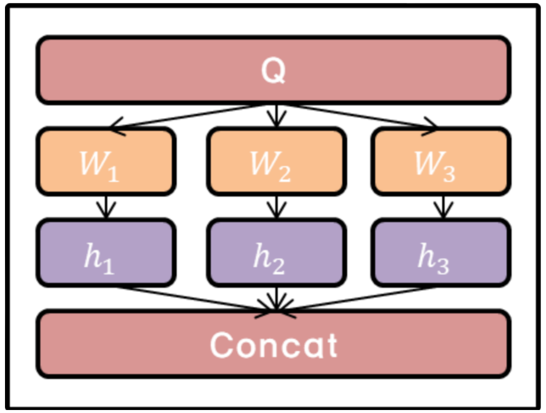


$$\mathrm{MultiHead}(Q,K,V) = \mathrm{Concat}(\mathrm{head_1},...,\mathrm{head_h})W^O$$
$$\text{where } \mathrm{head_i} = \mathrm{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

Linearly project the queries, keys and values h times with differently learned linear projection to dq, dv, dm respectively

- Give attention to many parts simultaneously

- we can get many dependencies(type of sentence, relation, nouns..)



<Multi-Head attention>

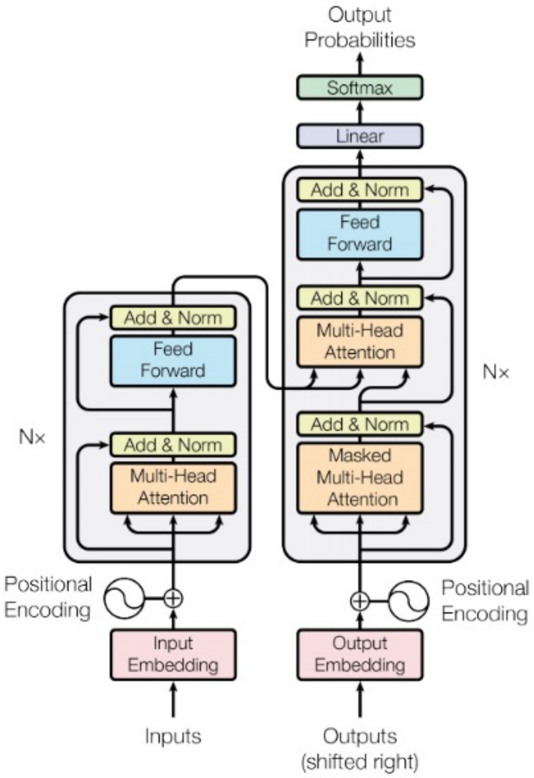


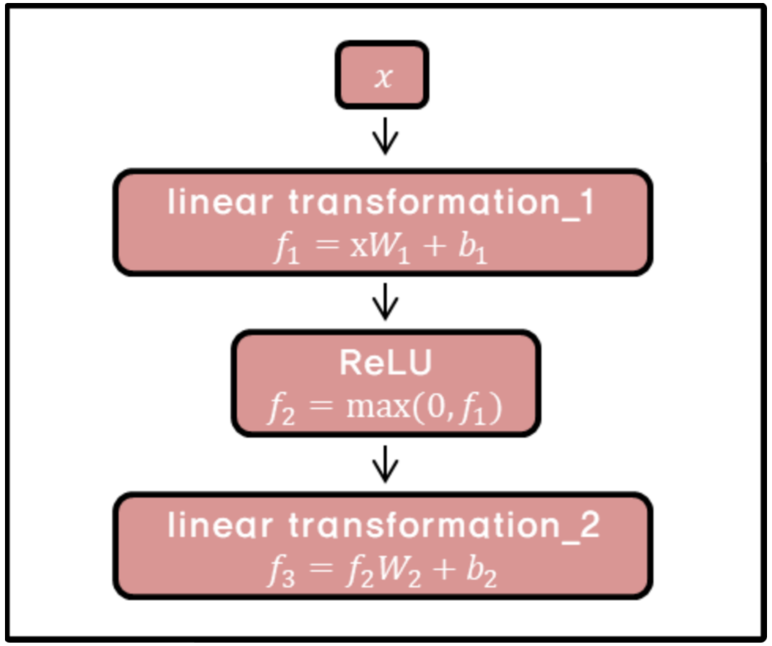
Figure 1: The Transformer - model architecture.

1. Encoder's self attention layer
   - all keys, values and queries come from the output of previous layer in encoder
   - every position in the encoder attends overall positions in the previous layer

2. Decoder's self attention layer
   - similar to encoder's self attention layer

3. Encoder, Decoder attention layer
   - queries from previous decoder layer, key and value from output of the encoder

<Encoder: position-wise feedforward network>

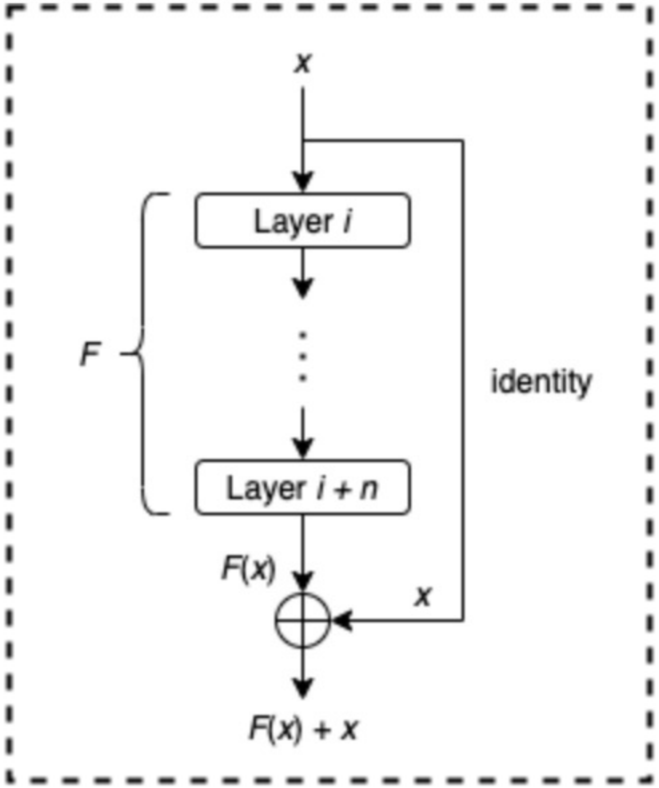


Position-wise: feedforward layer effects to each position of input sequence

Feed forward: No backpropagation

$$\mathrm{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

<Encoder: residual connection>

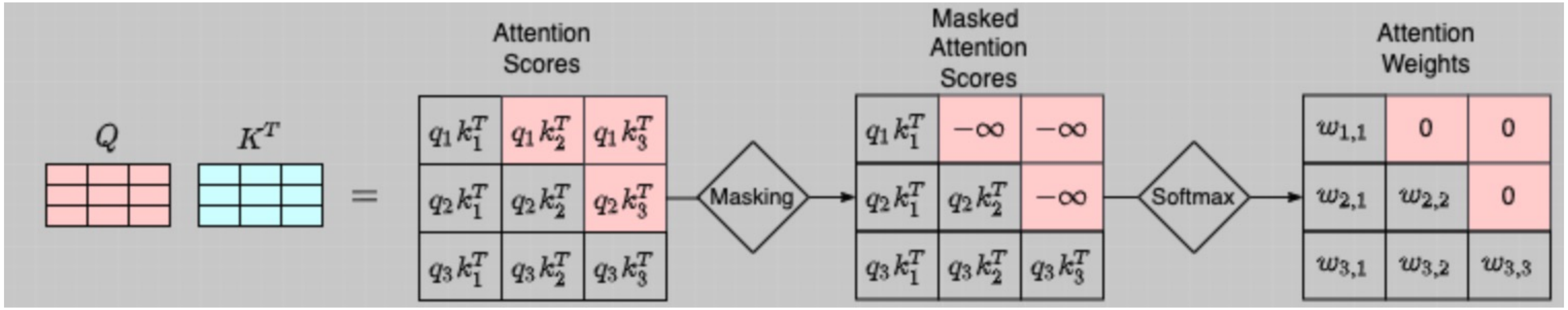


The deeper the model becomes, the harder it is to learn

With residual connection, we can provide other ways to reach the latter part of the model

We can prevent gradient vanishing and gradient exploding

<Decoder: masked multi-head attention>



We mask the future inputs not to use them to make prediction

<Transformer vs recurrent, convolutional model>

1. total computational complexity per layer (n < d)

2. the amount of computation that can be parallelized
   - measured by the minimum number of sequential operations required

3. path length between long-range dependencies
   - the length of the paths forward and backward signals have to traverse in the network

4. interpretable

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

04. Training

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [15]	23.75			
Deep-Att + PosUnk [32]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [31]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [8]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [26]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [32]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [31]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [8]	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**	$2.3 \cdot 10^{19}$	

WMT English-German datasets, Dropout(P=0.1), Adam optimizer are used

05. Results

	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