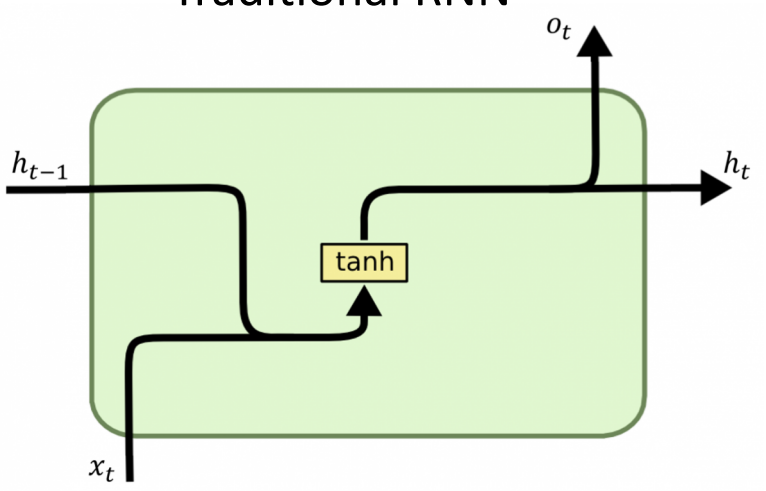
Neural Networks and Deep Learning
Transformer

Shumin Wu
Department of Computer Science
shumin.wu@colorado.edu
February 26, 2020

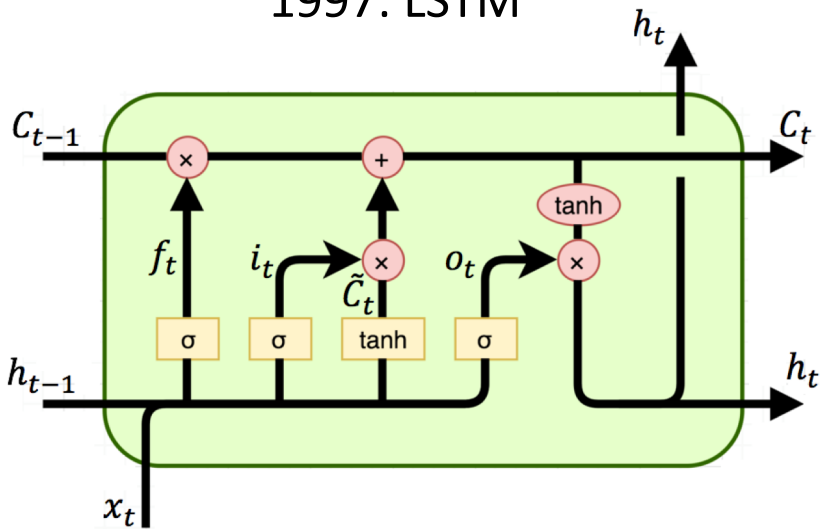
Previous Lecture: Gated RNN

1994: "…the probability of successful training of a traditional RNN via SGD rapidly reaching 0 for sequences of only length 10 or 20 ."

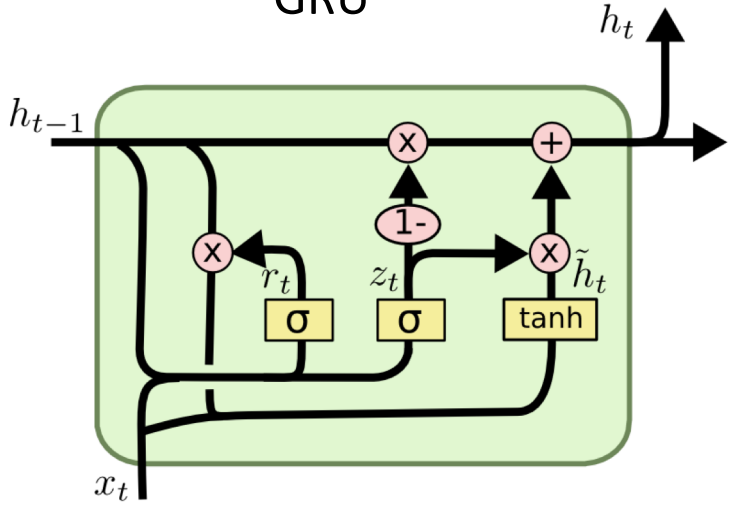
Traditional RNN



1997: LSTM



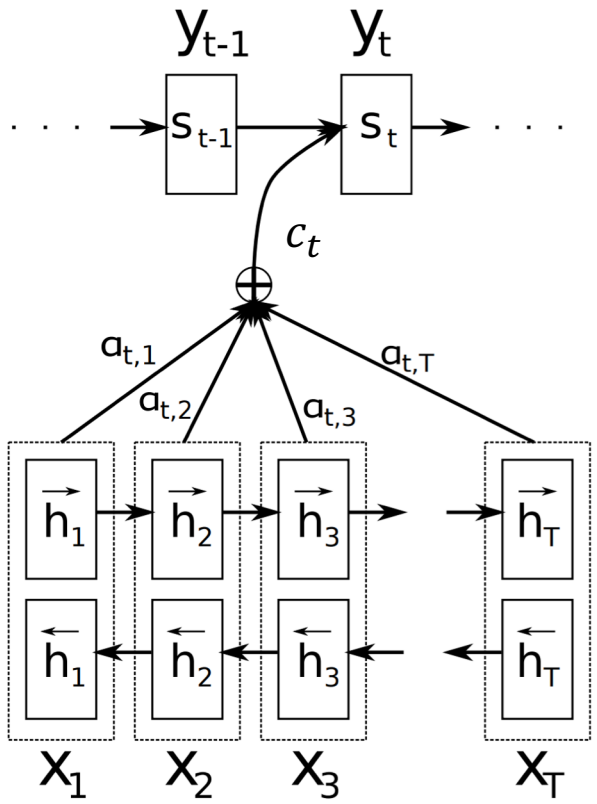
GRU



source: http://dprogrammer.org/rnn-lstm-gru

Previous Lecture: Attention (2015)

Joint alignment and translation:



source: [Bahdanau et al, 2015]

i: output index
j: source index

$$a(s_{i-1}, h_j) = v_a^T \tanh(W_a s_{i-1} + U_a h_j)$$
$$\alpha_i = \text{softmax}_j(a(s_{i-1}, h_j))$$
$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j$$

$a(s_{i-1}, h_j)$ can be interpreted as an alignment or "attention" model (what source tokens the output need to correctly emit the token).

Can be jointly trained w/ (multi-layer) feed-forward network.

Multiplicative (Dot-Product) Attention

Instead of: $$a(s_{i-1}, h_j) = v_a^T \tanh(W_a s_{i-1} + U_a h_j)$$

Use: $$a(s_{i-1}, h_j) = h_j^T W_a s_{i-1}$$

Faster and saves memory over additive attention:

$$a(s_{i-1}, H) = H^T W_a s_{i-1}$$

But output need to be scaled for larger dimension. Why?

Hint: the final attention weights are computed as:

$$\alpha_i = \mathrm{softmax}_j(a(s_{i-1}, H))$$

Self-Attention

Dot-product self-attention:

$$a(h_i, h_j) = h_j^T W_a h_i$$

Conceptually, this computes the similarity between h_j and $W_a h_i$
If we self-attend to the entire sequence, we have:

$$c_i = softmax(H^T W_a h_i) h_i$$

What does a weighted sum of hidden states similar to one's self represent? (Are we conflating the computation of attention distribution and the context output?)

Memory Network

Suppose we want to build an question/answering system of a closed database of statements and closed set of answers:

Sam walks into the kitchen. Sam picks up an apple. Sam walks into the bedroom. Sam drops the apple. Q: Where is the apple? A. Bedroom	Brian is a lion. Julius is a lion. Julius is white. Bernhard is green. Q: What color is Brian? A. White	Mary journeyed to the den. Mary went back to the kitchen. John journeyed to the bedroom. Mary discarded the milk. Q: Where was the milk before the den? A. Hallway

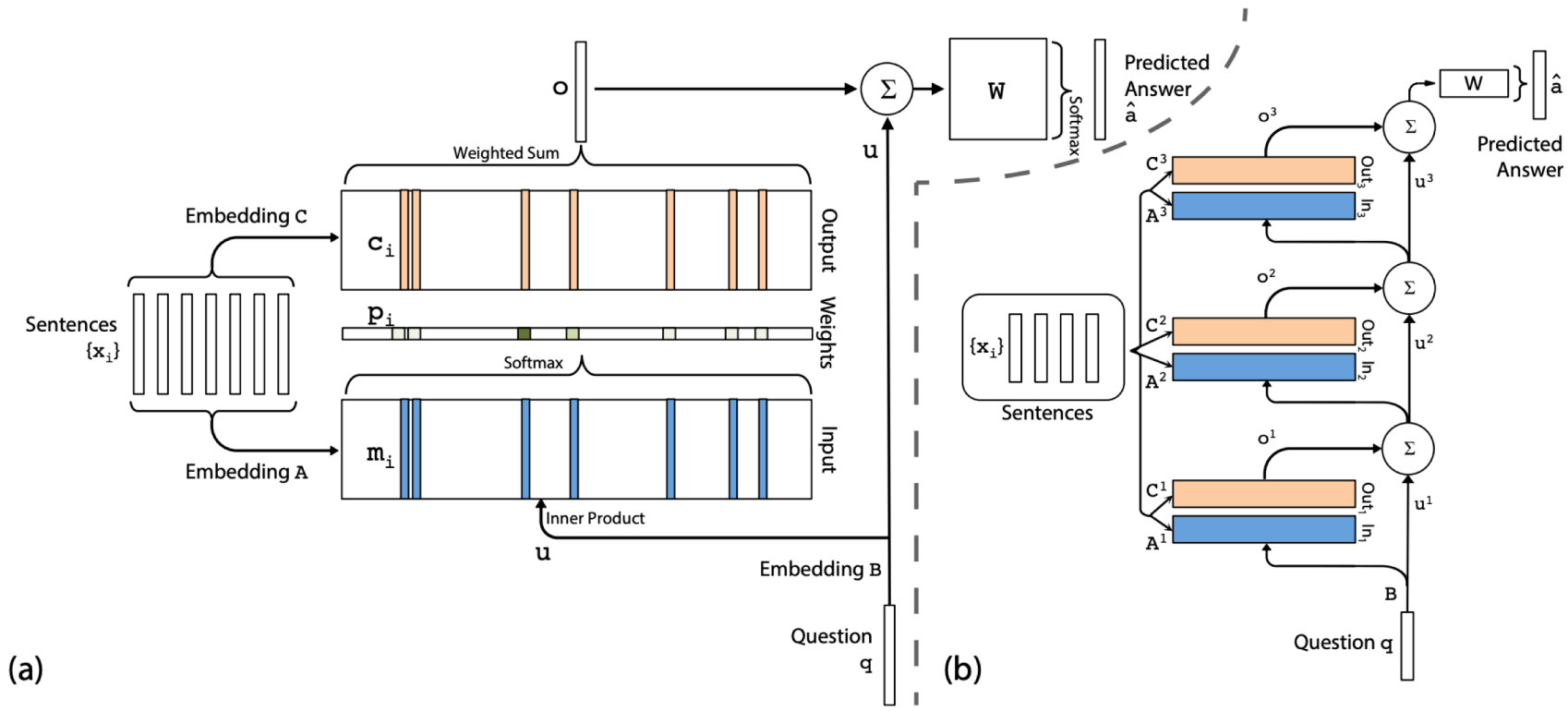
source: [Sukhbaatar et al., 2015]

We can use a general information retrieval approach adopted to deep learning:

1. Embed each statement to 2 vectors: an index/key vector k, and a value vector v
2. Embed the question to a vector q in the same space as k
3. Retrieve value vectors based on similarity between q and each k_i and merge them (weighted sum)
4. Use output function (softmax) to select the best answer.

Memory Network

$$m_i = Ax_i \quad \text{(key)}$$
$$c_i = Cx_i \quad \text{(value)}$$
$$u = Bq \quad \text{(query)}$$
$$p_i = \text{softmax}_i(u^T m_i)$$
$$o = \sum_i p_i c_i$$
$$a = \text{softmax}(W(o+u))$$



source: [Sukhbaatar et al., 2015]

Transformer: Scaled Dot-Product Attention

Maps each hidden state vector $h_i \in \mathbb{R}^{d^k}$ to 3 vectors:

Query: $q_i = W^q h_i$

Key: $k_i = W^k h_i$

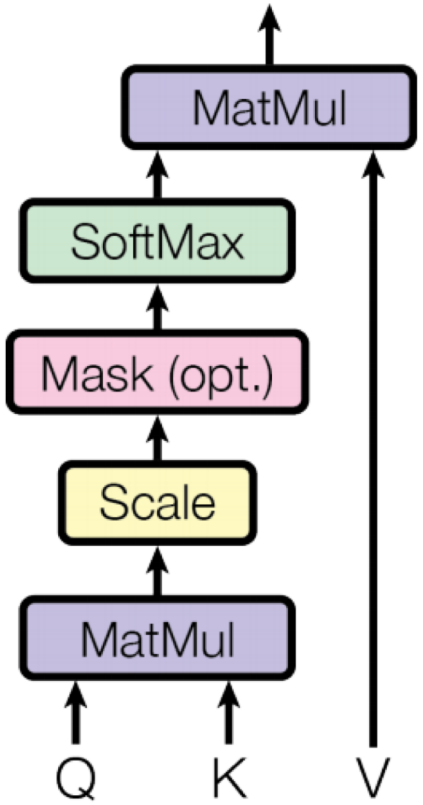
Value: $v_i = W^v h_i$

Output context:

$$C = softmax(\frac{QK^T}{\sqrt{d_k}})V$$

Works for:

- Output attention to input (q is from the output states, K, V from the input states)
- Self-attention of inputs
- Self-attention of outputs (need to mask outputs to the right during training)



Source: Vaswani et al, 2017

Scaled Dot-Product Attention Computation

Given:

$$q = \begin{bmatrix} 0.5 & 0.6 & 0.1 \end{bmatrix}$$

$$K = \begin{bmatrix} 0.8 & 0.4 & 0.2 \\ 0.4 & 0.3 & 0.7 \end{bmatrix}$$

$$V = \begin{bmatrix} 1.2 & 0.3 & 0.2 & \dots \\ 0.4 & 0.3 & 0.7 & \dots \end{bmatrix}$$

Then:

$$qK^T = \begin{bmatrix} 0.5 & 0.6 & 0.1 \end{bmatrix} \begin{bmatrix} 0.8 & 0.4 \\ 0.4 & 0.3 \\ 0.2 & 0.7 \end{bmatrix} = \begin{bmatrix} 0.66 & 0.45 \end{bmatrix}$$

$$\frac{qK^T}{\sqrt{3}} = \frac{\begin{bmatrix} 0.66 & 0.45 \end{bmatrix}}{\sqrt{3}} = \begin{bmatrix} 0.38 & 0.26 \end{bmatrix}$$

$$softmax(\frac{qK^T}{\sqrt{3}}) = \frac{\begin{bmatrix} e^{0.38} & e^{0.26} \end{bmatrix}}{e^{0.38} + e^{0.26}} = \begin{bmatrix} 0.53 & 0.47 \end{bmatrix}$$

$$c = \begin{bmatrix} 0.53 & 0.47 \end{bmatrix} \begin{bmatrix} 1.2 & 0.3 & 0.2 & \dots \\ 0.4 & 0.3 & 0.7 & \dots \end{bmatrix}$$

$$= \begin{bmatrix} 0.82 & 0.30 & 0.44 & \dots \end{bmatrix}$$

Doesn't matter how many columns V has, the attention output is just a linear combination of the rows in V. What's unsatisfying about this?

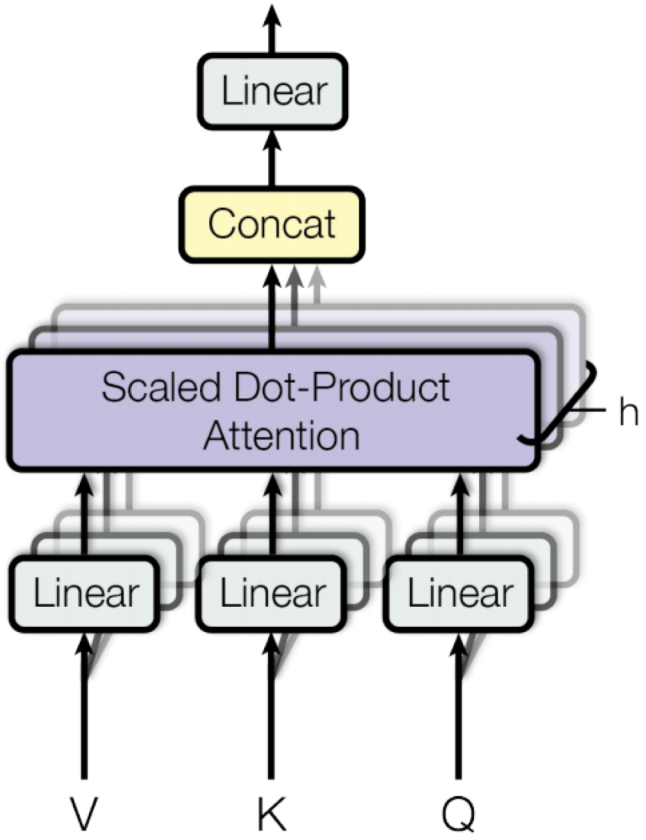
Multi-Head Attention

Issue: hidden state of a token may encode multiple pieces of information.

- One may need to pay different levels of attention to each piece of information.

Solution: split monolithic attention.

- Concatenate the output of multiple smaller & independent attentions, then add a dense layer.



source: [Vaswani et al, 2017]

Multi-Head Attention

$$\text{MultiHead}(Q, K, V) = \left[\text{head}_1 \quad \ldots \quad \text{head}_h\right] W^O$$

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

$$W_i^Q, W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$$

$$W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$$

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

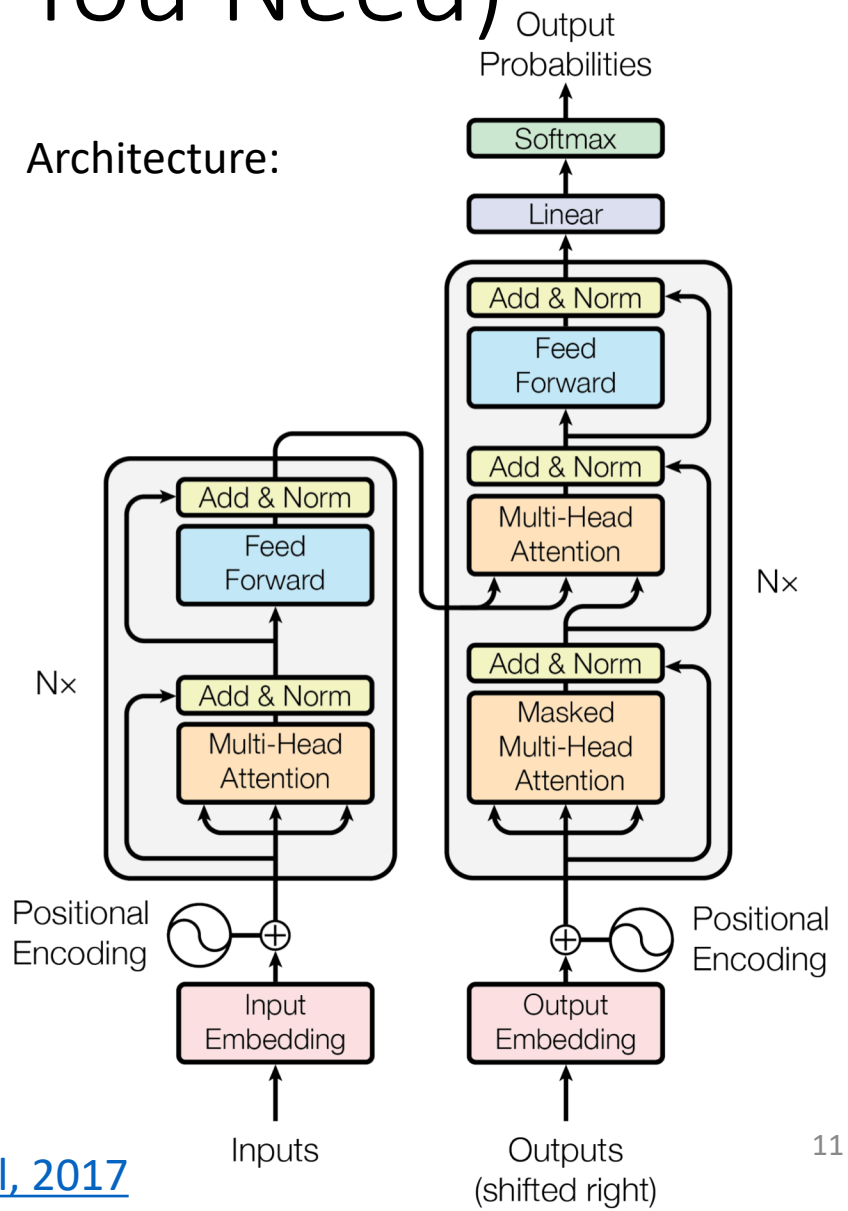
- 8 heads were used
- Each with dimension 64, 512 total
- $d_k = d_v$
- Some work found multi-head not to provide large gain for translation
- Other analysis on BERT show heads specializing on different linguistic phenomena

Transformer (Attention Is All You Need)

Replaces Bi-RNN on input sequence and RNN on output sequence with self-attention

- Since the full output sequence isn't available during decoding, tokens to the right are ignored by masking the attention weight output during training.

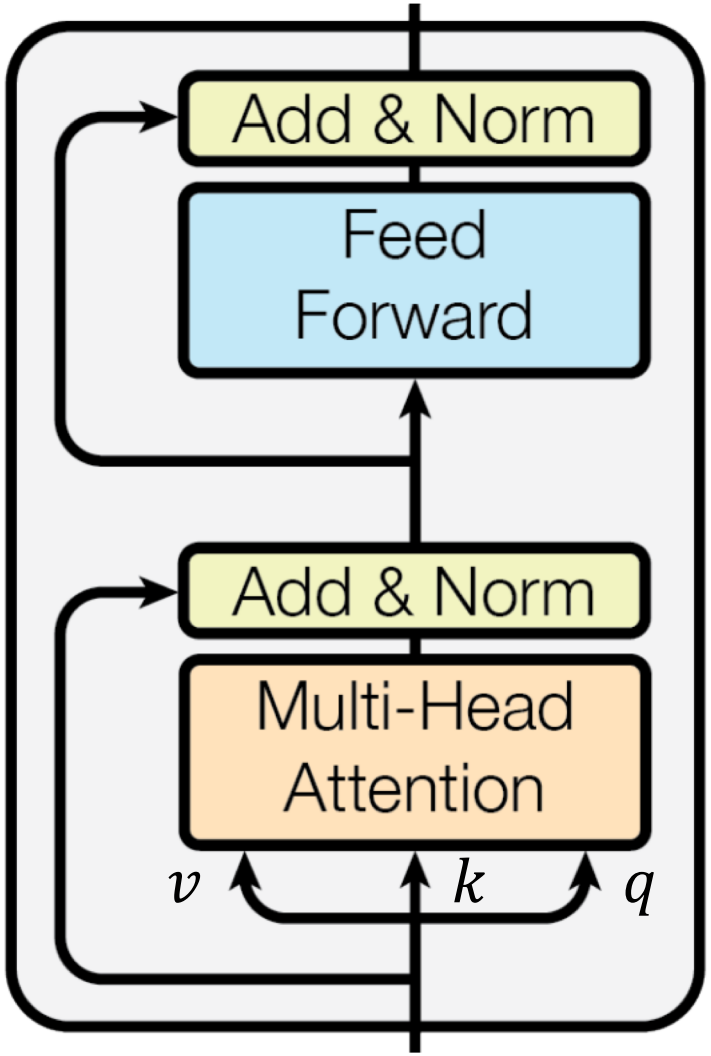
Architecture:



source: [Vaswani et al, 2017]

Encoder layers

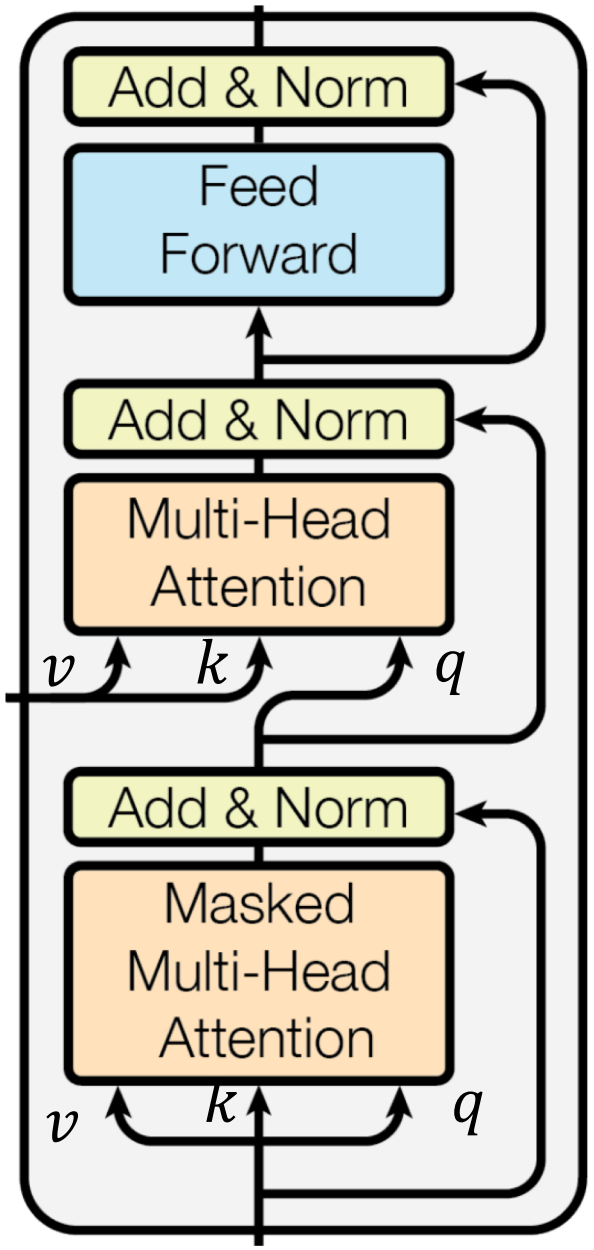
- Input embedding split into q, k, v
- Multi-head self-attention
- Residual connection w/ layer normalization (zero mean & unit variance of output vector) to FFN
- FFN w/ 1 hidden layer (4x the input dimension)
- Residual connection w/ layer normalization again.



source: Vaswani et al, 2017

Decoder layers

- Similar to encoder layers
- Has an additional input-to-output attention block between the masked self-attention and FFN
  - q generated from output self-attention
  - k, v from final input encoding



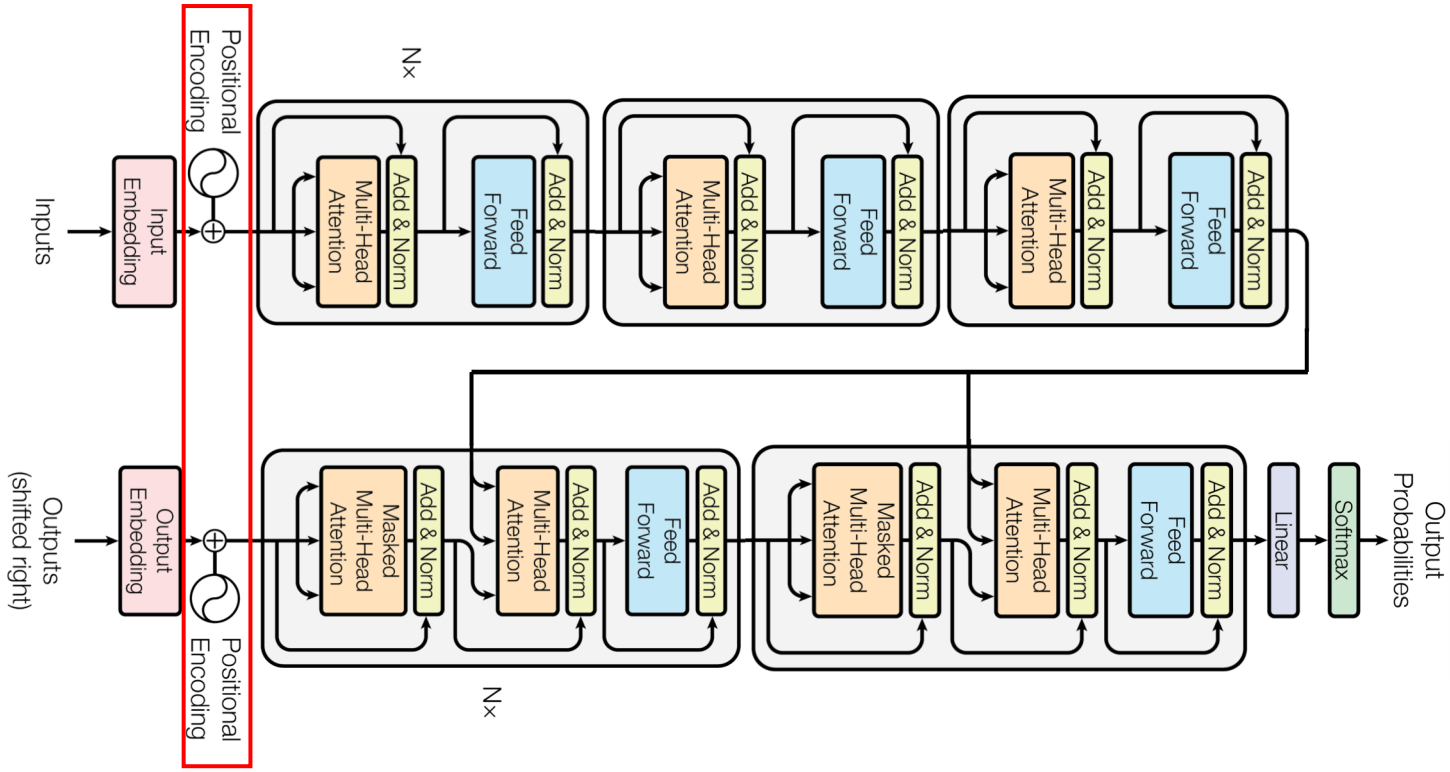
source: Vaswani et al, 2017

Multiple Blocks

Both encoder and decoder may have multiple (possibly different number of) connected blocks. Weights are not shared across blocks.

Multiple self-attention iterations may be needed to learn surrounding context (as each token gain more context, it has more to offer to its neighbors).

Residual connection and layer normalization makes learning deep layers feasible.



source: [Vaswani et al, 2017]

One more thing…

Positional Encoding

Attention computation does not consider relative/absolute token position unless explicitly encoded (unlike RNN or CNN).

Use alternating sine/cosine functions in the dimensions:

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

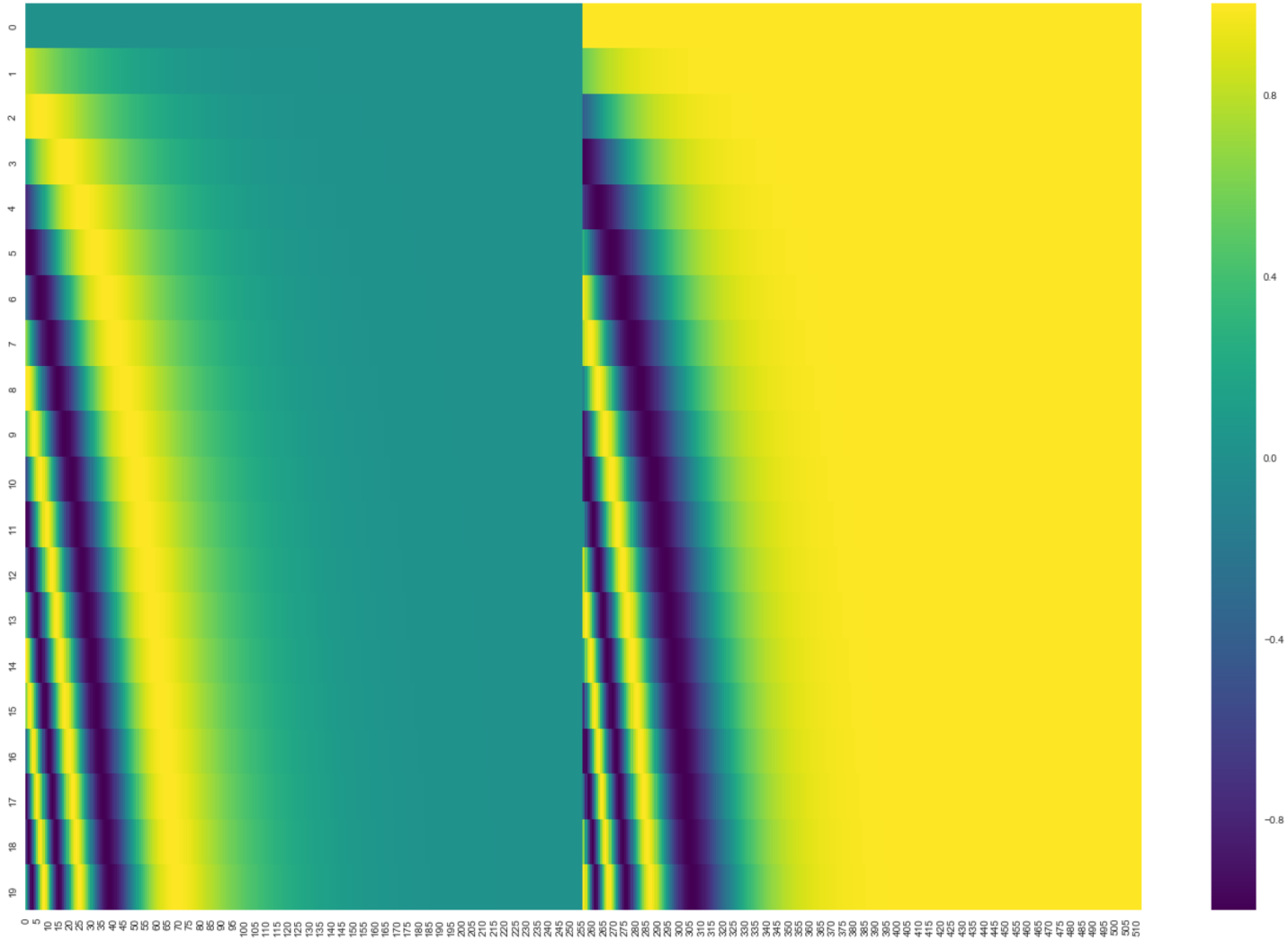
where relative distance is a linear function of PE. Or let the model learn PE (may not have long token sequence not in training data).

PE is add to the input embedding (may also be concatenated). Residual connections between layers prevents it from vanishing.

Positional Encoding Visualization

20 tokens
512 dimensions

From T2T library.



source: https://jalammar.github.io/illustrated-transformer/

Self-Attention vs RNN

- Less complex than RNN per layer when token length is smaller than the embedding dimension
  - Often the case for per-sentence processing.
  - Can limit the attention span or use approximate methods for very long sequence.
- Training and encoding parallelizable
- Attention weights offers some interpretability

n: token length
d: embedding dimension

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

source: Vaswani et al, 2017

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

GNMT + RL: RNN/LSTM + Attention + Reinforcement Learning
ConvS2S: CNN + Attention
MoE: Mixture-of-Experts (sparsely combine many FFN)

Constituent Parsing Results

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

Model Parameter Analysis

N : encoder/decoder blocks

d_{model} : model dimension

d_{ff} : FFN hidden layer size

h : attention head number

d_k : query/key vector dimension

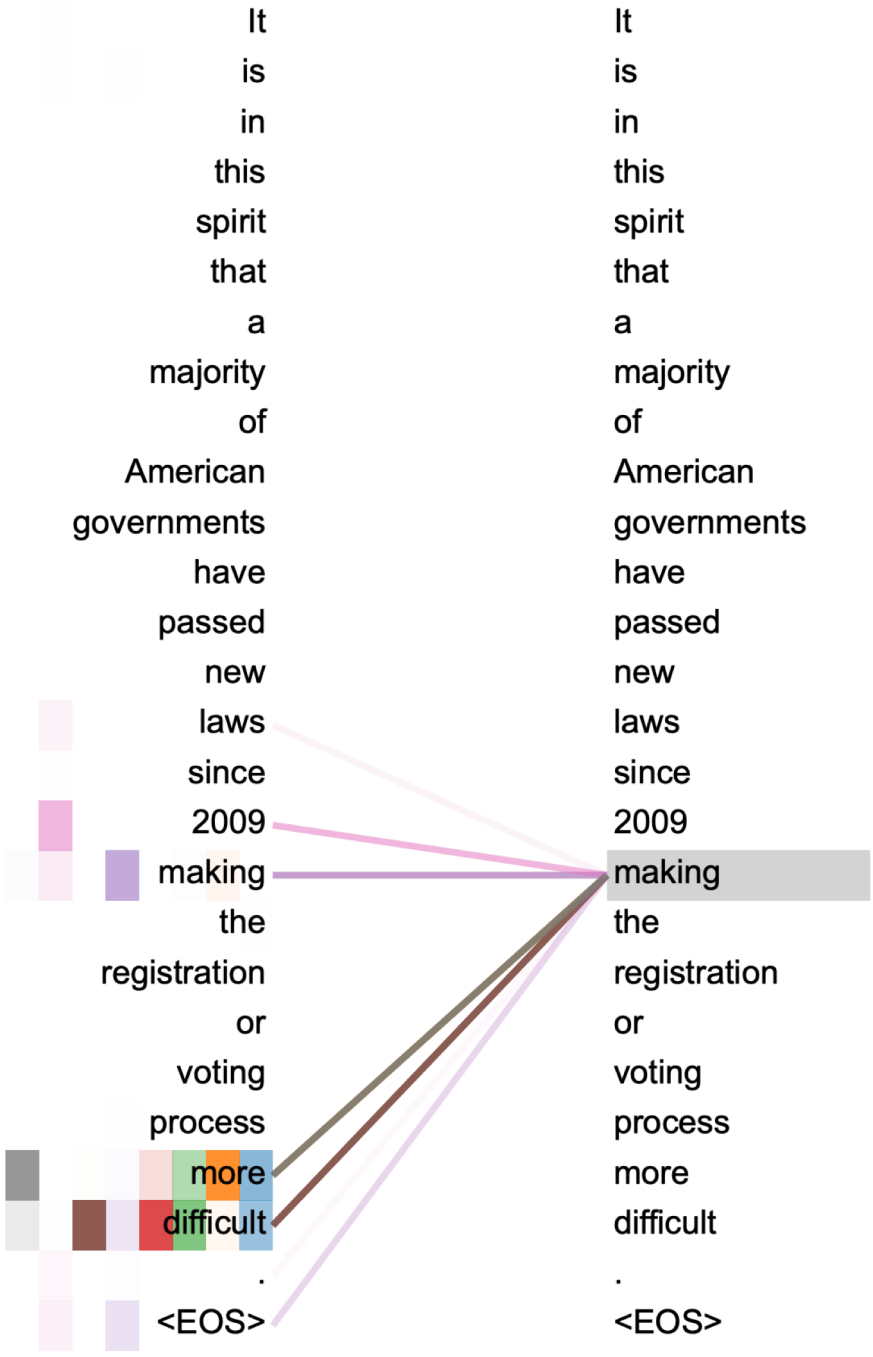
d_v : value vector dimension

P_{drop} : drop-out rate

ϵ : label smoothing factor

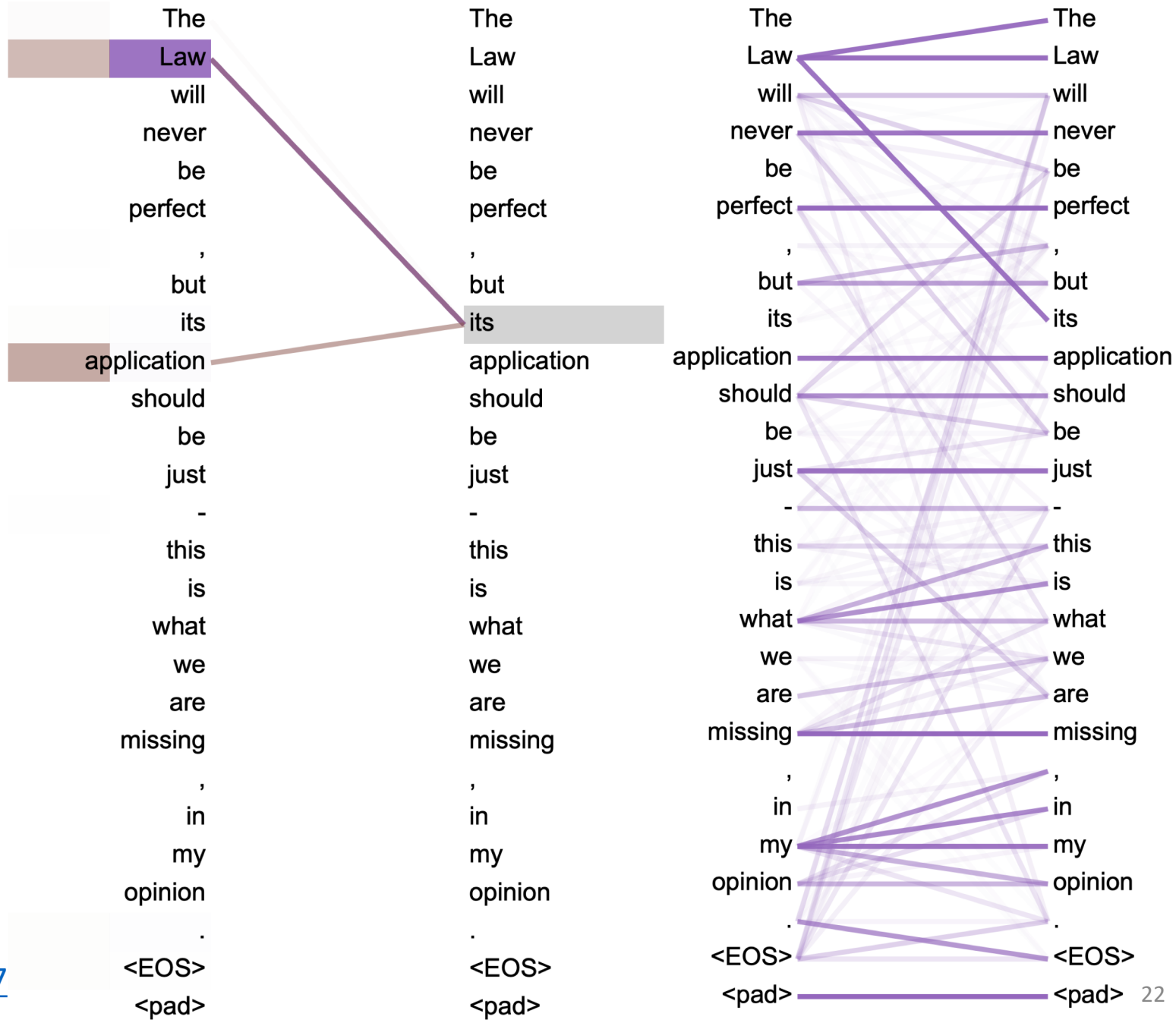
	N	d_{model}	d_{ff}	h	d_k	d_v	P_{drop}	ϵ_{ls}	train steps	PPL (dev)	BLEU (dev)	params $\times 10^6$
base	6	512	2048	8	64	64	0.1	0.1	100K	4.92	25.8	65
(A)				1	512	512				5.29	24.9	
				4	128	128				5.00	25.5	
				16	32	32				4.91	25.8	
				32	16	16				5.01	25.4	
(B)					16					5.16	25.1	58
					32			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

Visualization: Long-Term Dependency



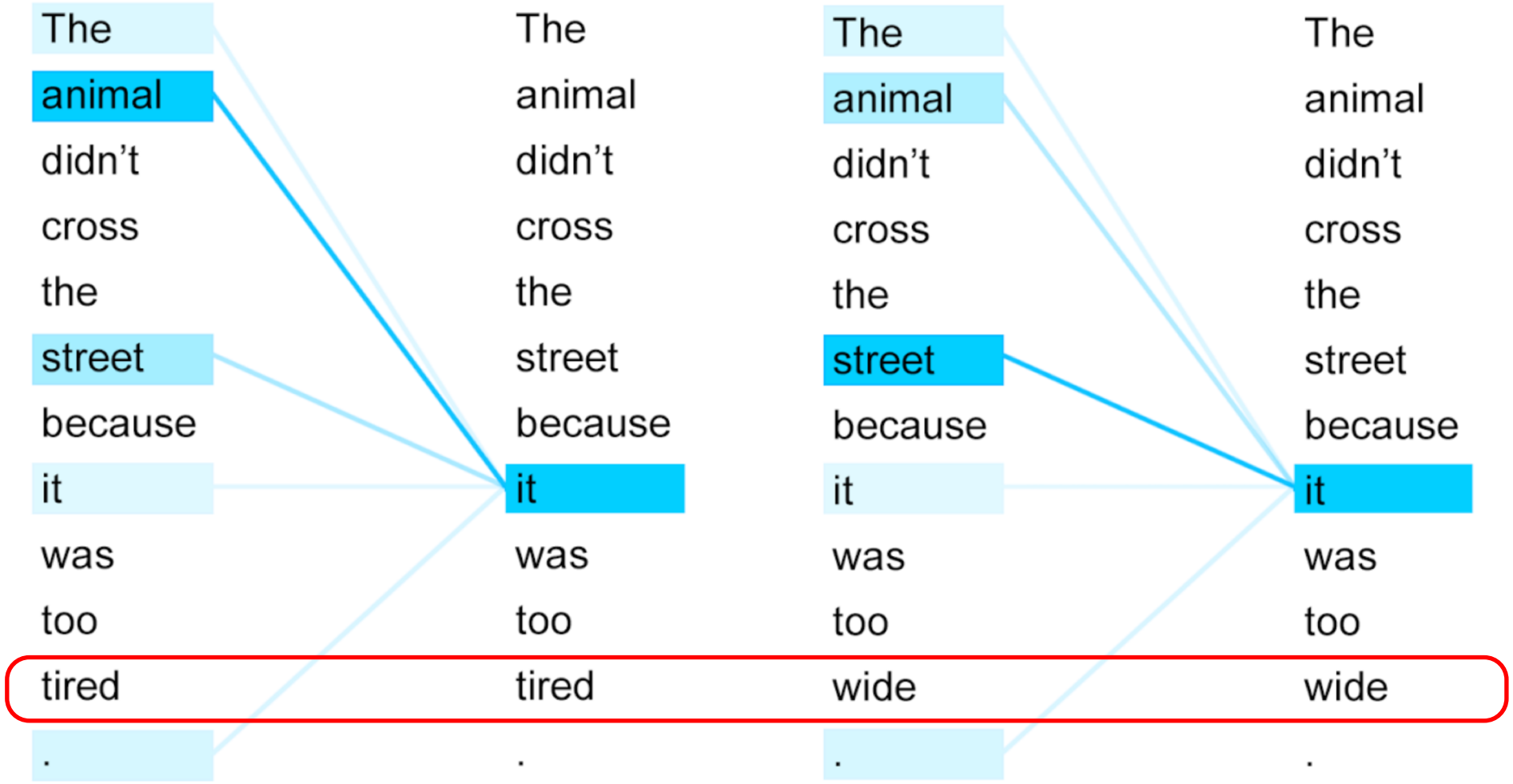
source: [Vaswani et al, 2017]

Visualization: Anaphora Resolution



source: [Vaswani et al, 2017]

Visualization: Anaphora Resolution



source: [Google Blog]