2024-Spring AI Application Thesis Research

Attention is All You Need

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones,
Aidan N. Gomez, Łukasz Kaiser, Illia Polosukhin
Google Brain, Google Research

Accepted to NIPS 2017

Yejin Yoon, Kangmin Lee

HYU 한양대학교 HANYANG UNIVERSITY

Natural Language Processing Lab.,
Hanyang University.

Attention is All You Need

Contents

PRE-REQUISITE

Brief Overview of Sequence-to-sequence models

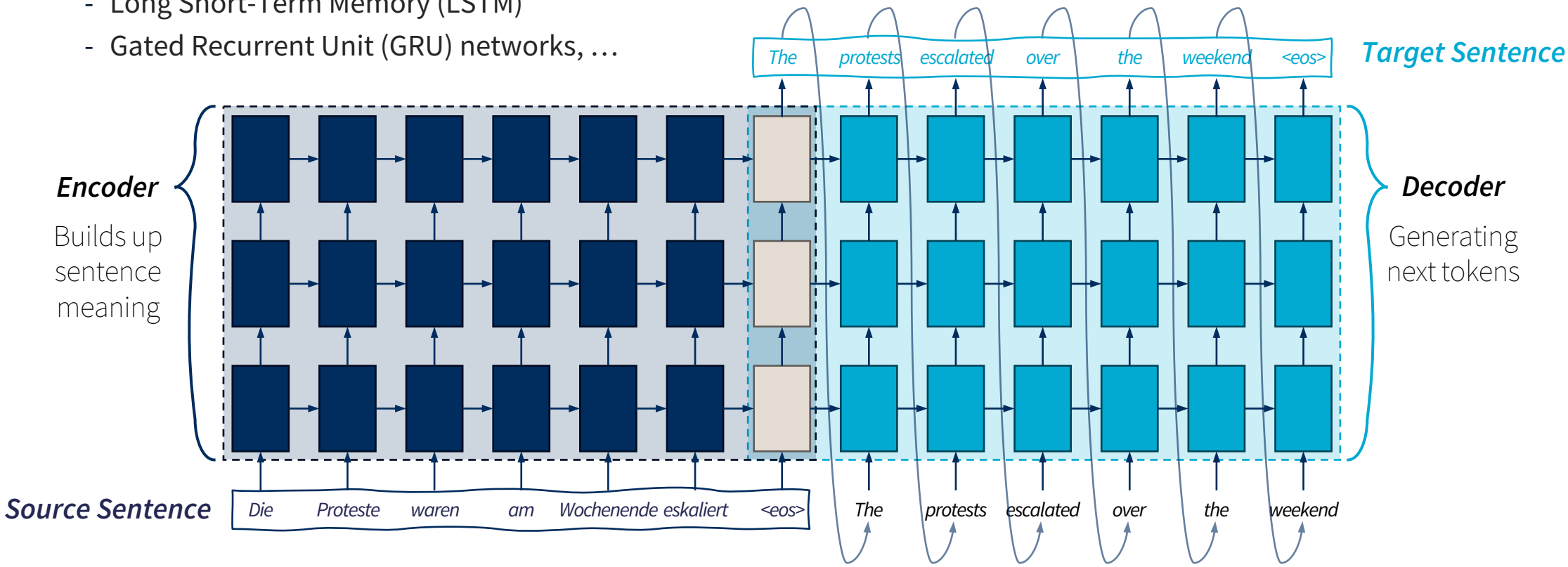
Limitations of Recurrent Models

Sequence-to-sequence with Attention

Brief Overview of Sequence-to-sequence models

Sutskever et al. 2024; Luong et al. 2015

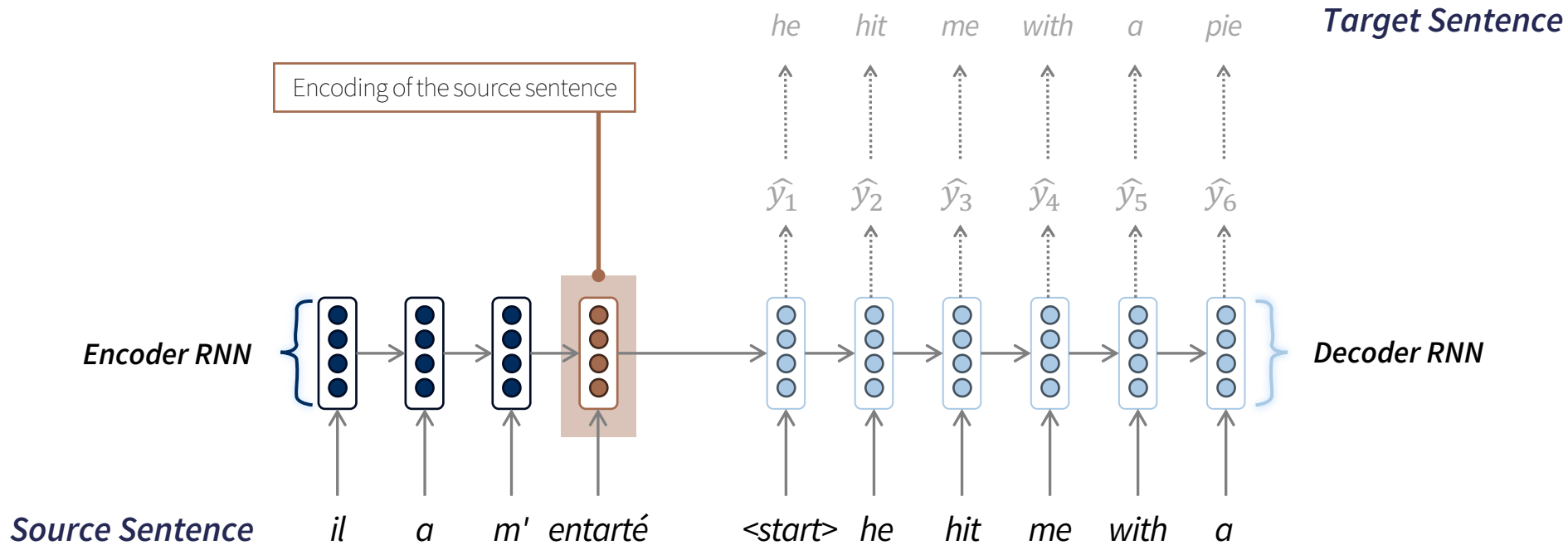
- **Recurrent Neural Networks (RNNs)**
  - RNNs and their variations : SOTA for for Sequence Modeling tasks (e.g., Machine Translation)
    - Long Short-Term Memory (LSTM)
    - Gated Recurrent Unit (GRU) networks, …



The hidden states from RNN layer i are the inputs to RNN layer $i+1$

Limitations of Recurrent Models #1

- **Information bottleneck**
  - Encoding of the source sentence needs to capture all information about the source sentence.

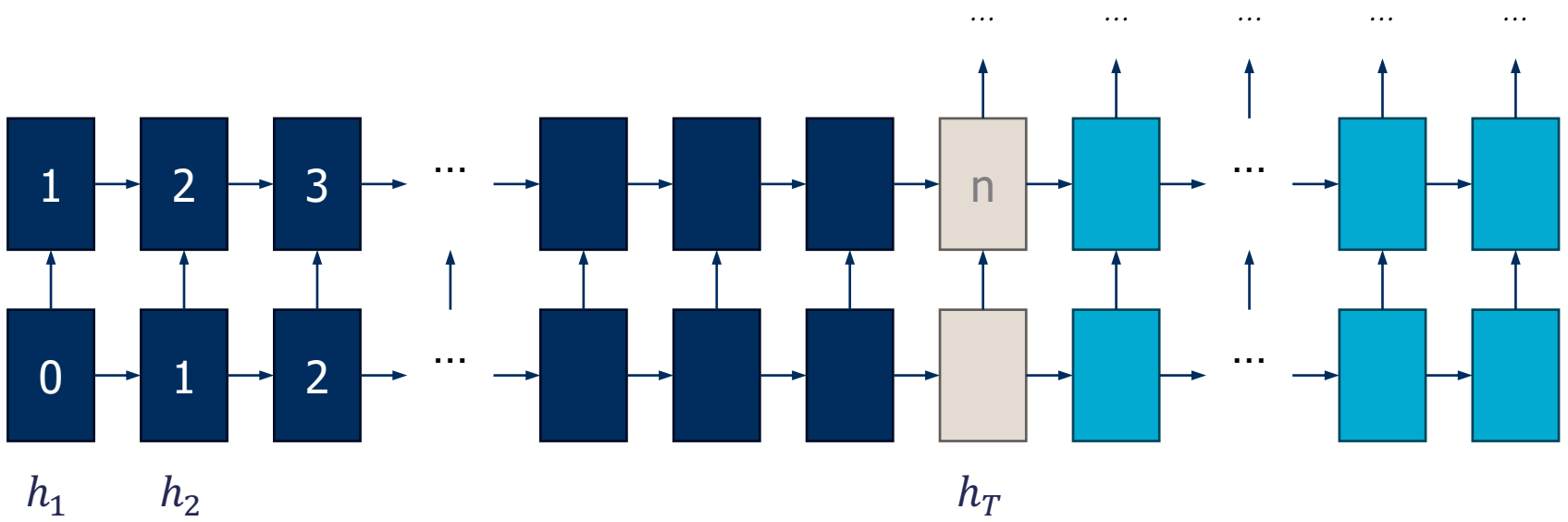


Limitations of Recurrent Models #2

- **Linear interaction distance**
  - O(sequence length) steps for distant word pairs to interact means hard to learn long-distance dependencies (because of gradient problems)

Limitations of Recurrent Models #3

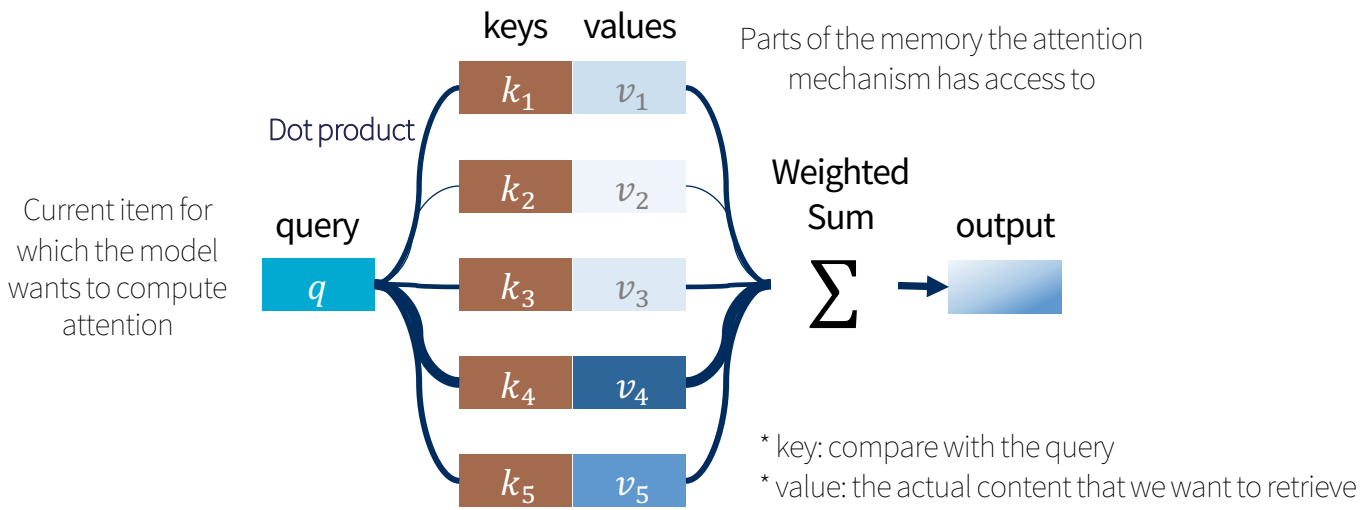
- **Lack of parallelizability**
  - GPUs can perform a bunch of independent computations at once
  - Future RNN hidden states can't be computed in full before past RNN hidden states have been computed



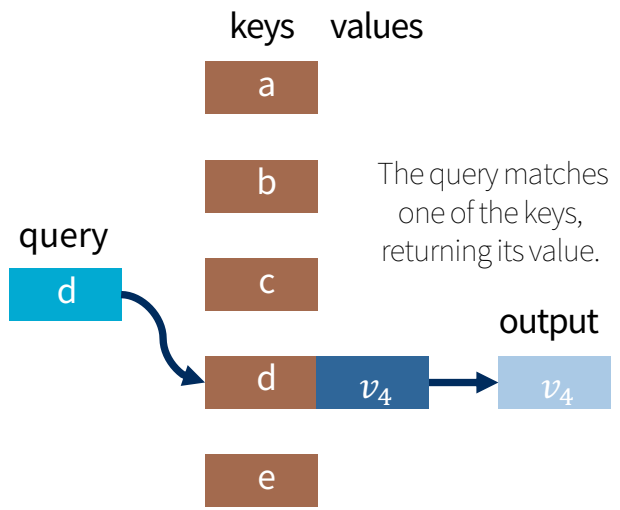
Numbers indicate min ***# of steps*** before a state can be computed

The Concepet of Attention Mechanisms

- **Attention is just a weighted average**
  - Weighted Average
    - Query Matching: The query matches all keys ‘softly’, to a weight between 0 and 1.
    - Weighted Sum: The keys’ values are multiplied by the weights and summed.



[Weighted Average]

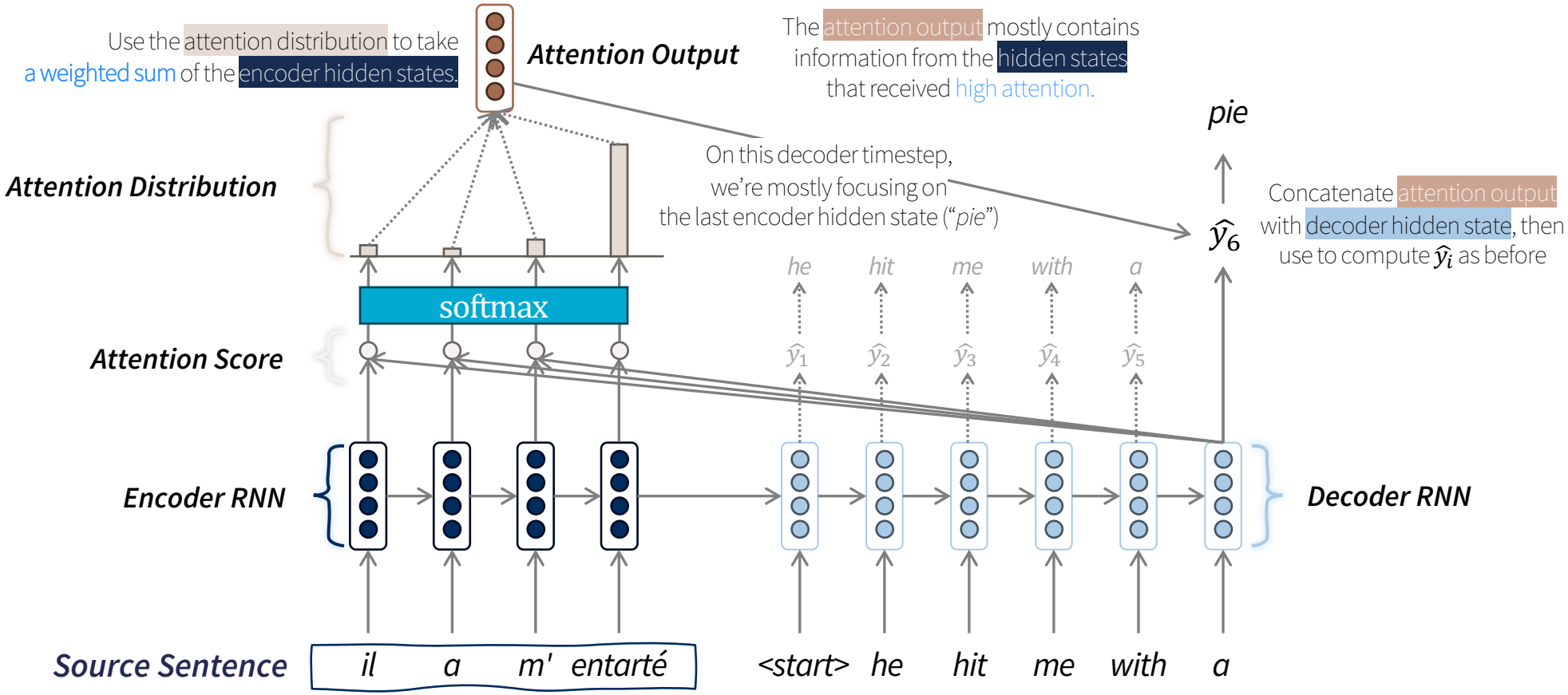


[Lookup Table]

On each step of the decoder, use *direct connection to the encoder* to focus on a particular part of the source sequence

Sequence-to-sequence with Attention

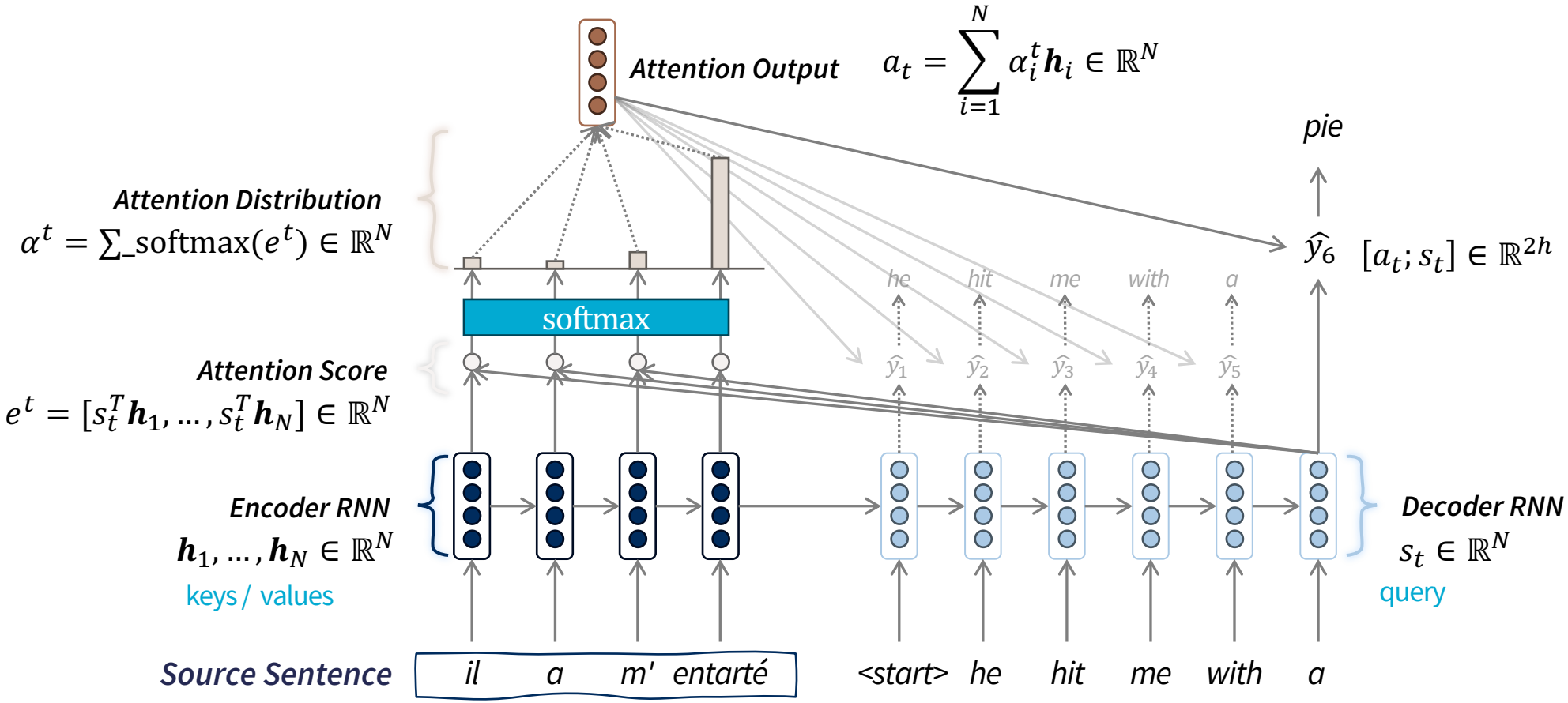
- **Example of Neural Machine Translation**



On each step of the decoder, use *direct connection to the encoder* to focus on a particular part of the source sequence

Sequence-to-sequence with Attention

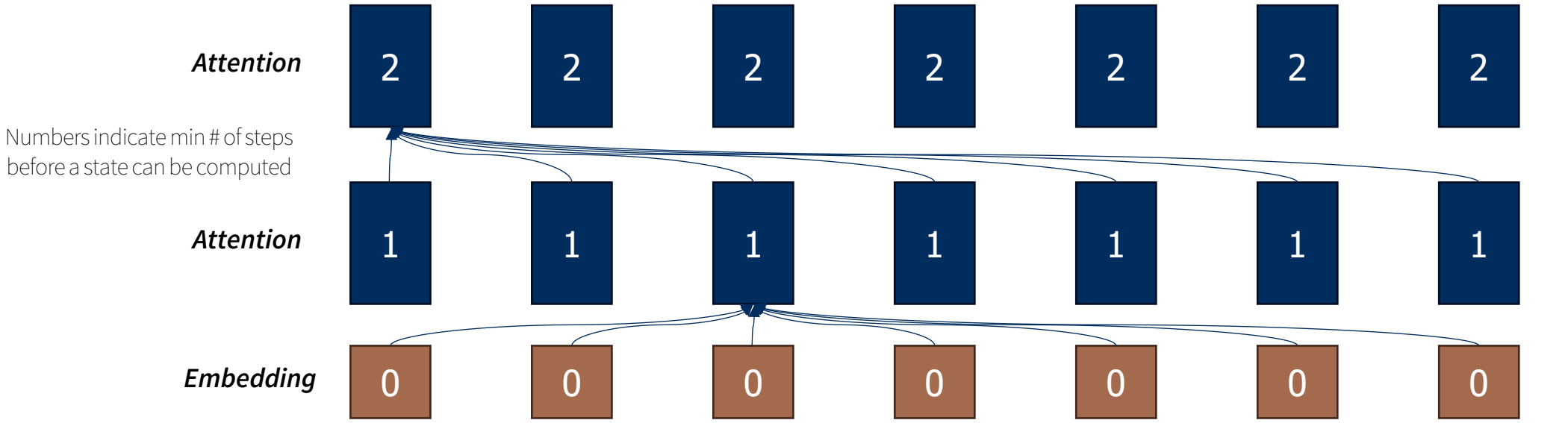
- **Example of Neural Machine Translation**



On each step of the decoder, use *direct connection to the encoder* to focus on a particular part of the source sequence

How attention mechanisms improve sequence modeling

- **Attention is parallelixable, and solves bottleneck issues.**
  - Attention treats each word's representation as a query to acess and incorporate information from a set of values.
  - Number of unparallelizable operations does not increase with sequence length
  - Maximum interaction distance: $O(1)$, since all words interact at every layer.



Transformer Architecture

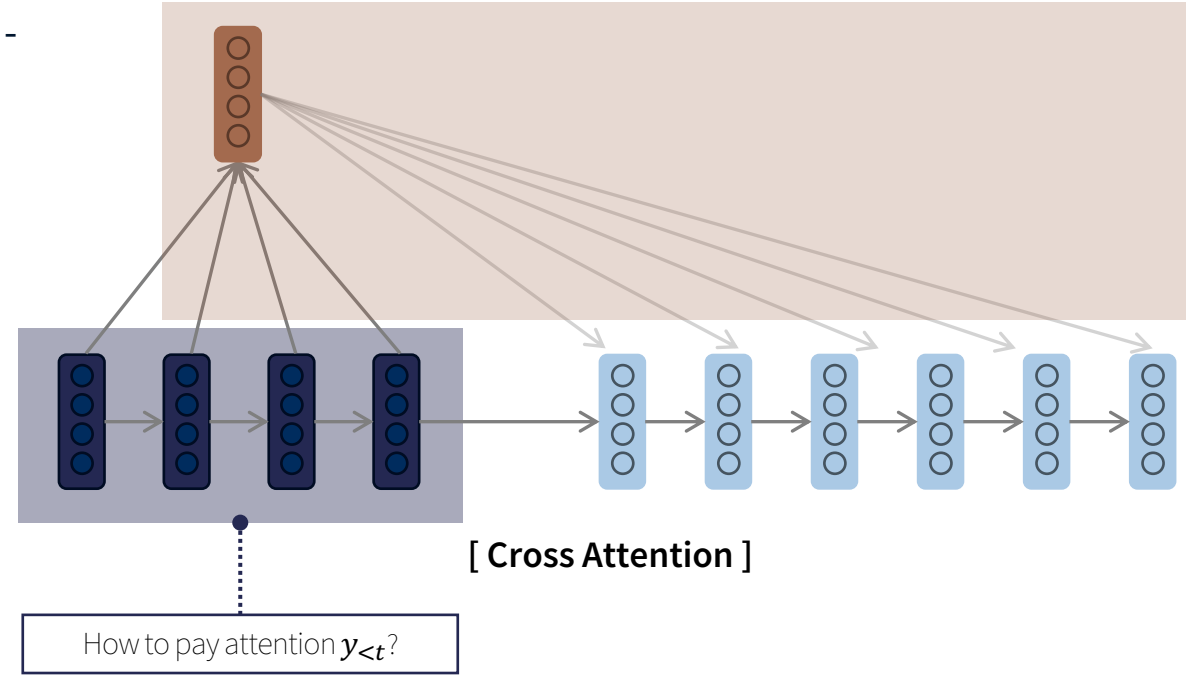
Self-Attention

Positional Embedding

Adding Nonlinearities by FFN

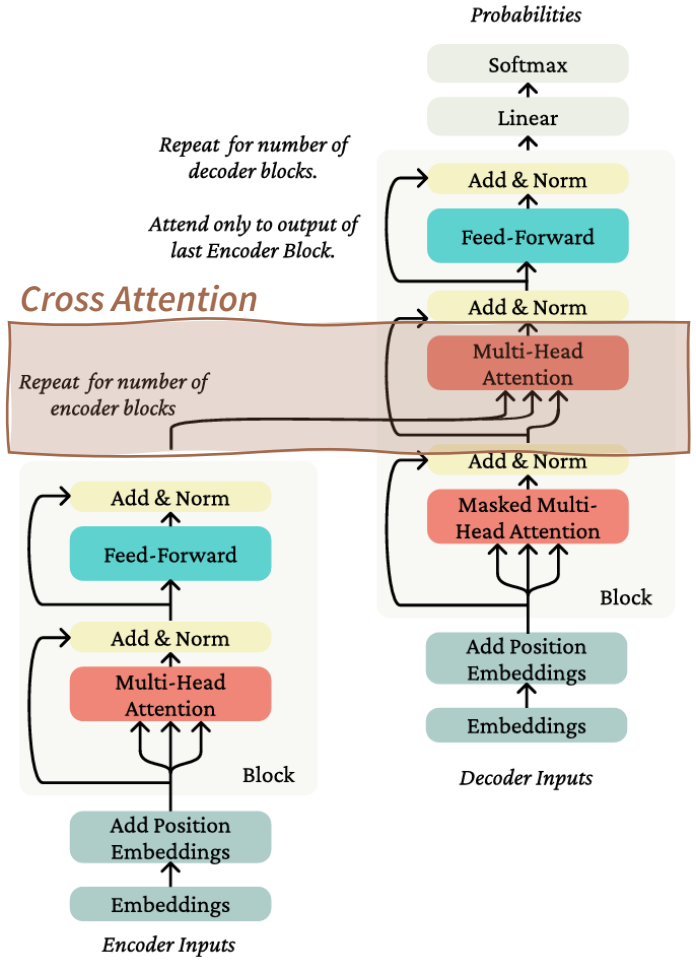
RQ #1 Can We Just Get Rid of the RNN Entirely?

- **Attention is a way to pass information from a sequence (x) to a neural network input. (h_t)**
  - What we talked about **Cross-Attention**
    : paying attention to the input x to generated y_t
  -



[Cross Attention]

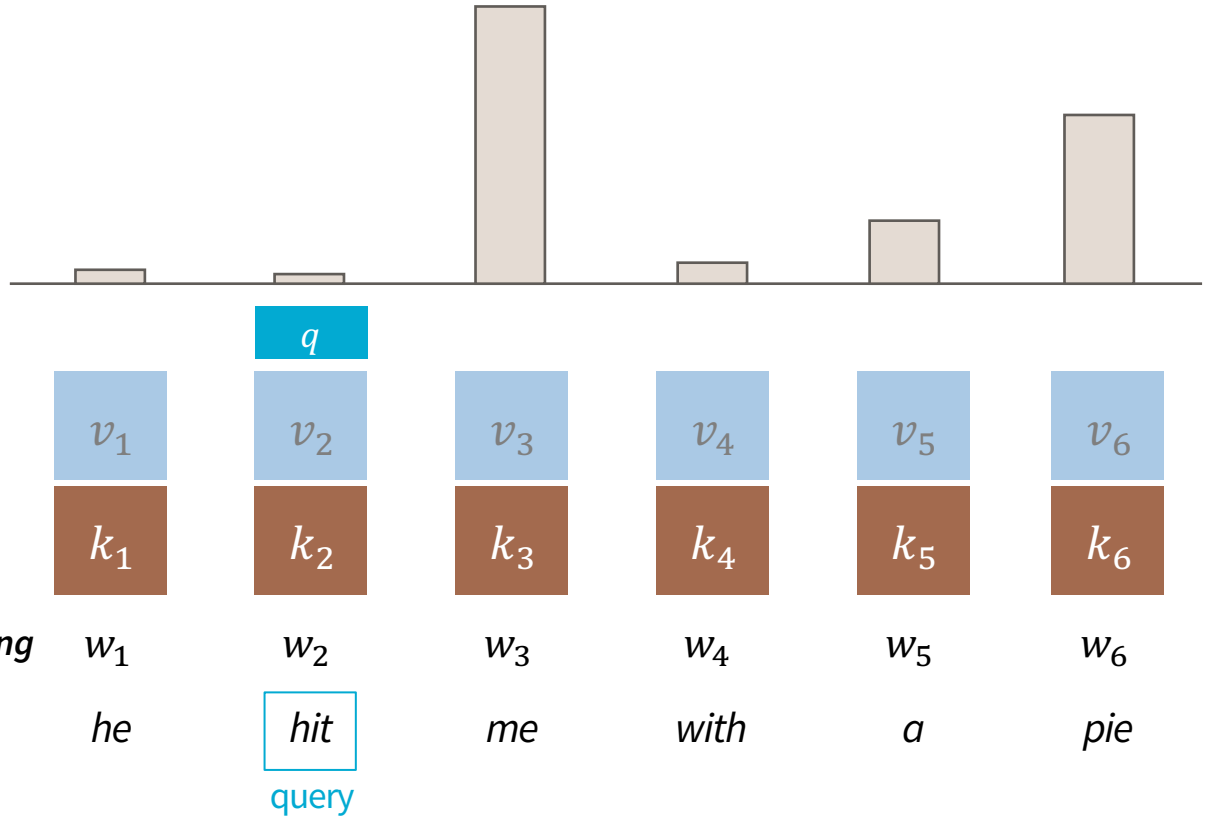
  - What we need is Self-Attention
    : to generate y_t, we need to pay attention to $y_{<t}$



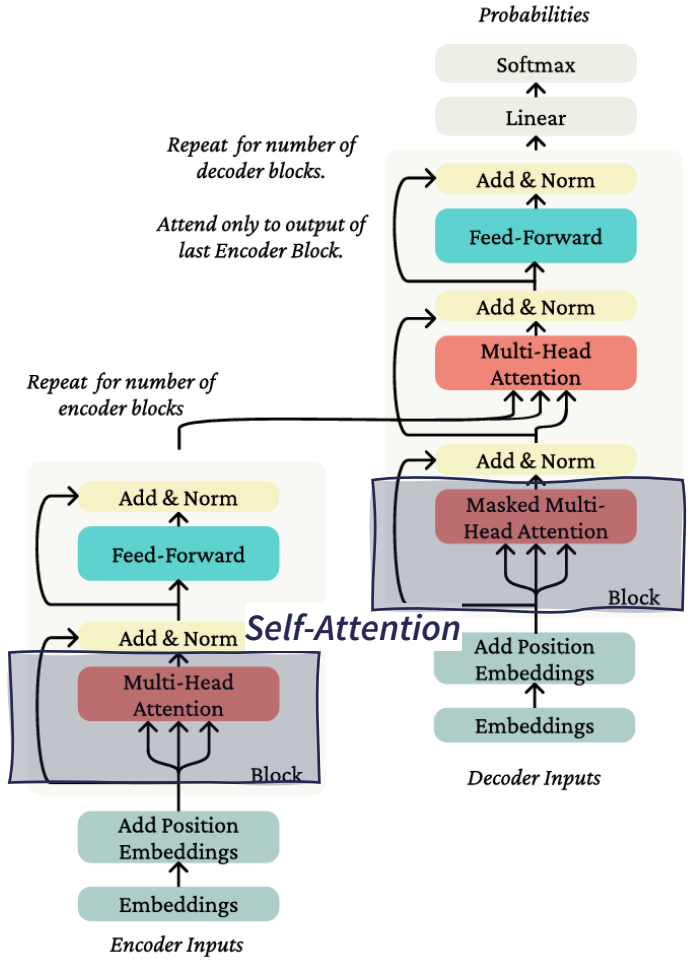
[Transformers]

RQ #1 Can We Just Get Rid of the RNN Entirely?

- **Hypothetical Example of Self-Attention**



Self-attention doesn't build in order information.



[Transformers]

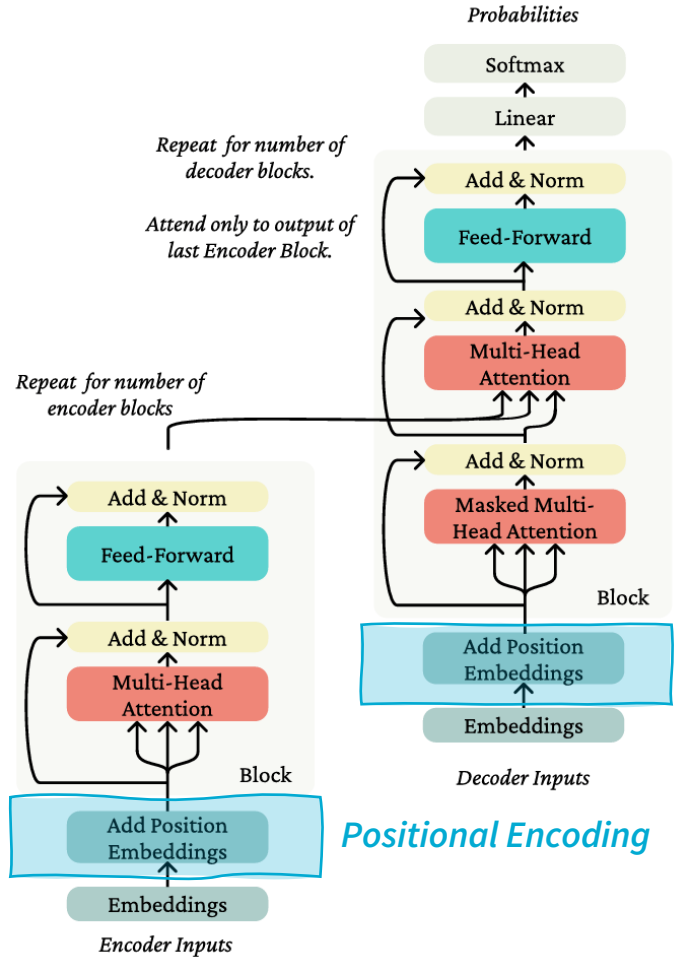
RQ #2 How to Incorporate Order Information?

- **Consider representing each sequence index as a vector**
  - Easy to incorporate this information into the self-attention : just **add** the p_i to the inputs
  - position vectors: $p_i \in \mathbb{R}^d$, for $i \in \{1, 2, 3, \ldots, n\}$
  - $\widetilde{x_i} = x_i + p_i$ or $\widetilde{x_i} = [x_i; p_i]$

- **Sinusoidal position representations**
  - Concatenate sinusoidal functions of varing periods
  - i is the position, d is total number of demension int the embedding

$$p_i = \begin{pmatrix} \sin i/10000^{(2*1)/d} \\ \cos i/10000^{(2*1)/d} \\ \cdots \\ \sin i/10000^{(2*d/2)/d} \\ \cos i/10000^{(2*d/2)/d} \end{pmatrix}$$

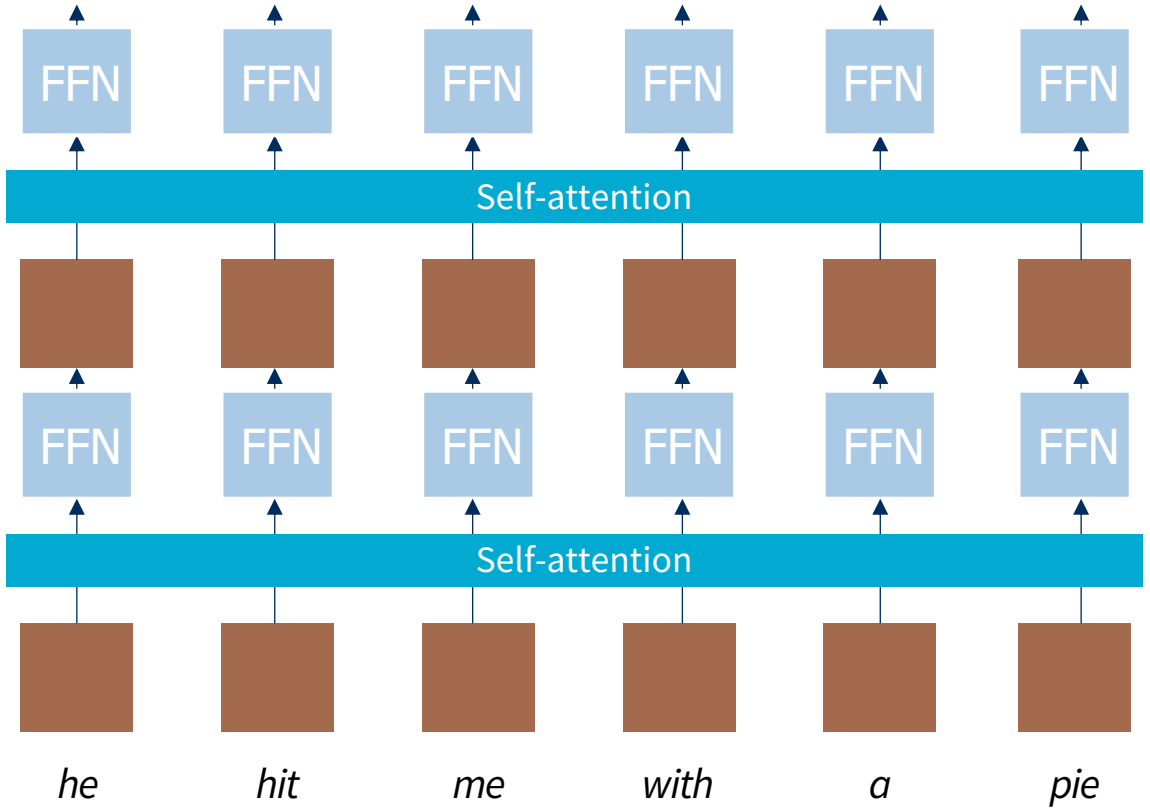
  - Maybe can extrapolate to long sequences as periods restart
  - Not learnable

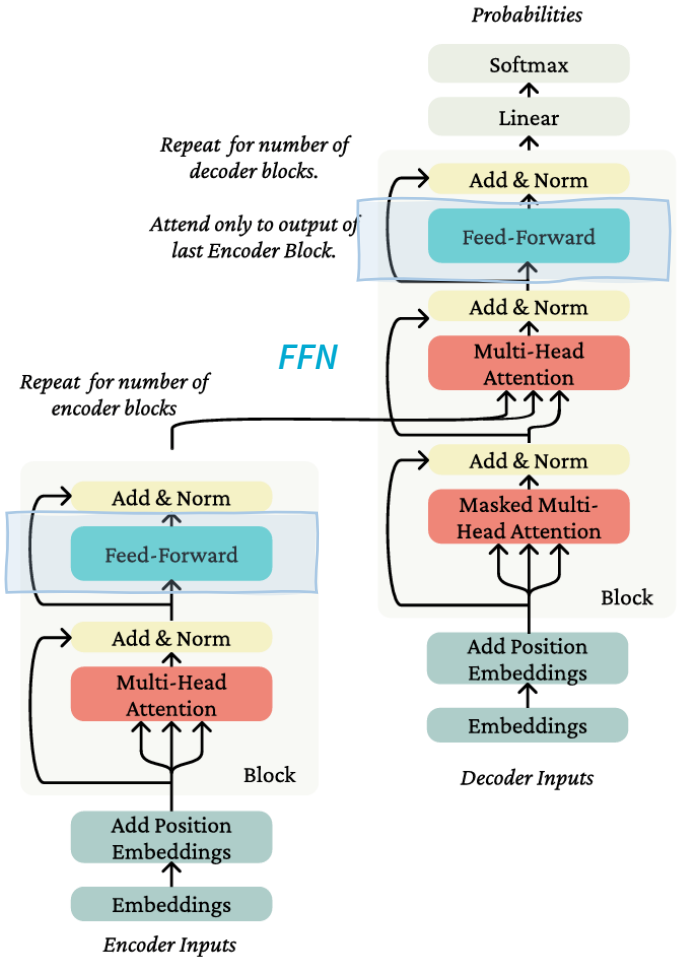


[Transformers]

RQ #3 There are No Nonlinearities for Deep Learning

- **There are no elementwise nonlinearities in self-attention.**
  - Easy to fix: add a feed-forward network to post-process each output.
  - $\mathrm{FFN}(x) = \mathrm{ReLU}(xW_1 + b_1)W_2 + b_2$; x is output of self-attention.



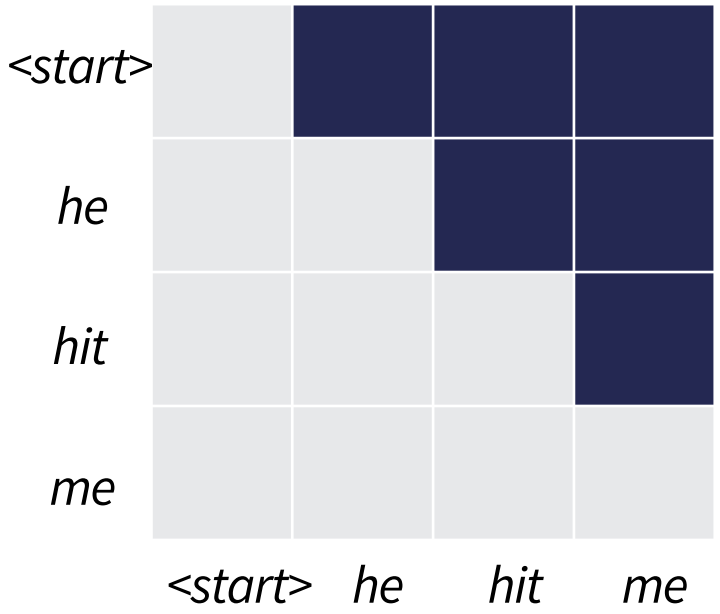


[Transformers]

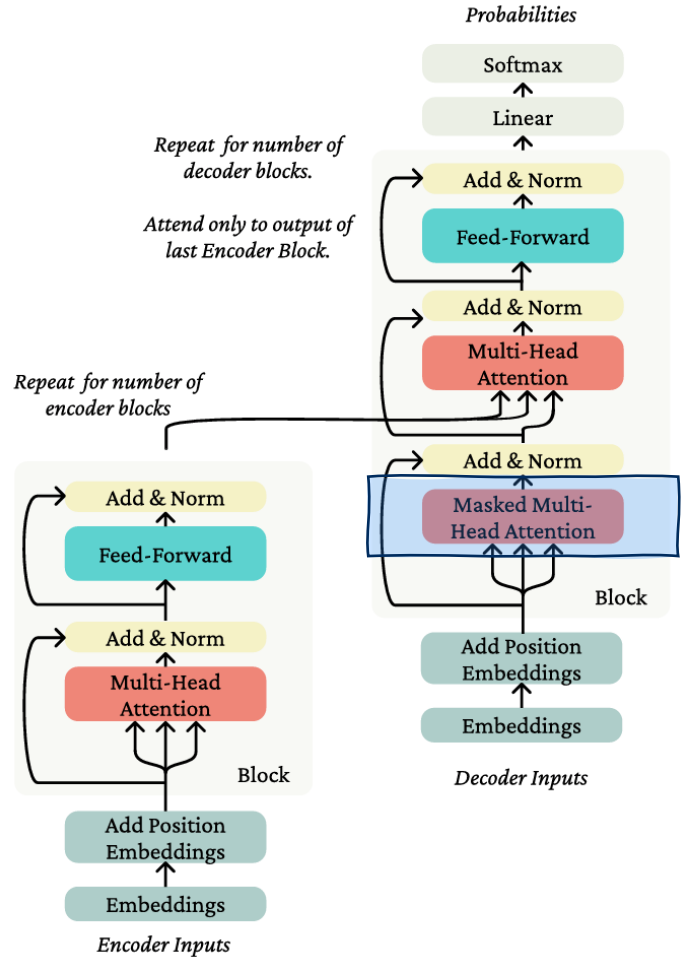
RQ #4 How to Ensure to Avoid Looking at Future

- **Masking the future in self-attention**
  - To use self-attention in decoders, **mask out attention** to future words by setting attention scores to $-\infty$.

$$e_{ij} = \begin{cases} q_i^\intercal k_j, j \leq i \\ -\infty, j > i \end{cases}$$



Don't look these black-out words
=future words

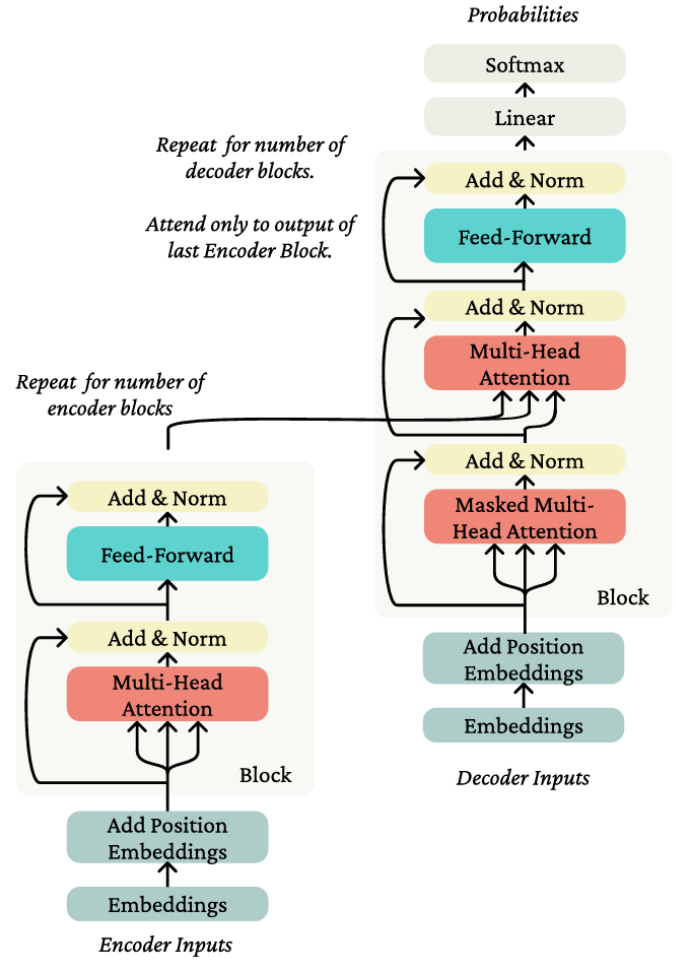


[Transformers]

Wrap-up: the Block of Transformers

- **Self-attention**: the basis of the method
- **Position representations**: Specify the sequence order, since self-attention is an unordered function of its inputs
- **Nonlinearities**: at the output of the self-attention block, frequently implemented as a simple feed-forward network
- **Masking**: to parallelize operations while not looking at the future

Let's just talk about multi-head attention.

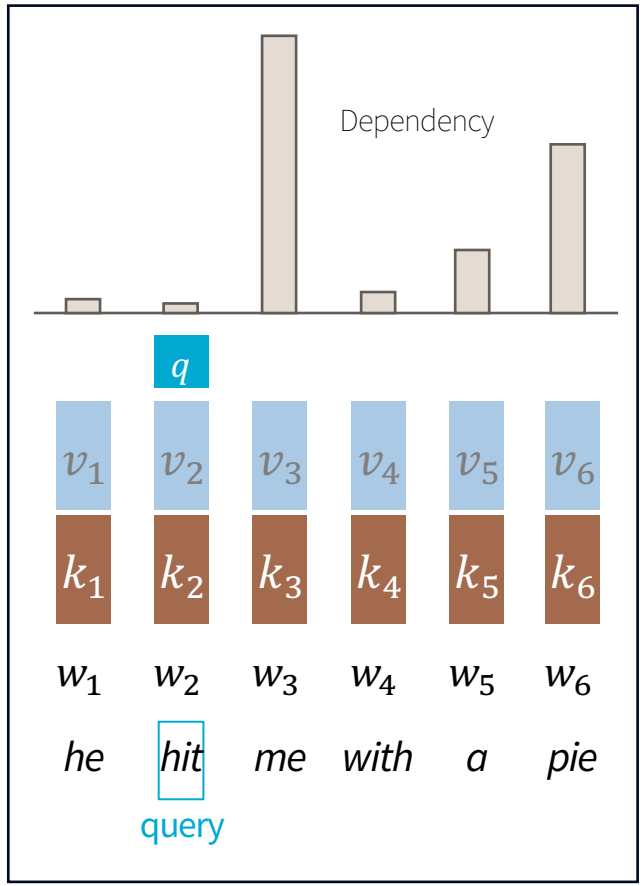


[Transformers]

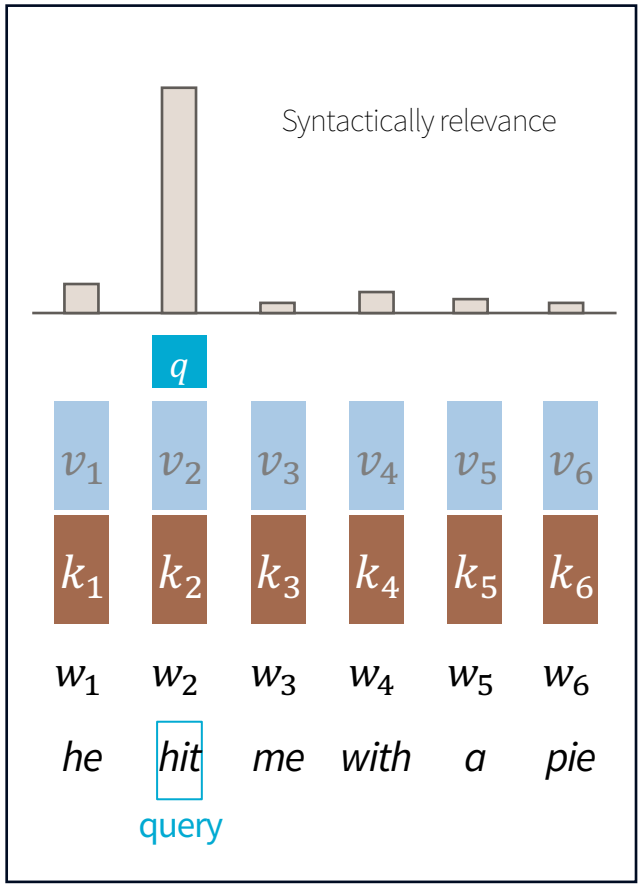
Multi-head Attention

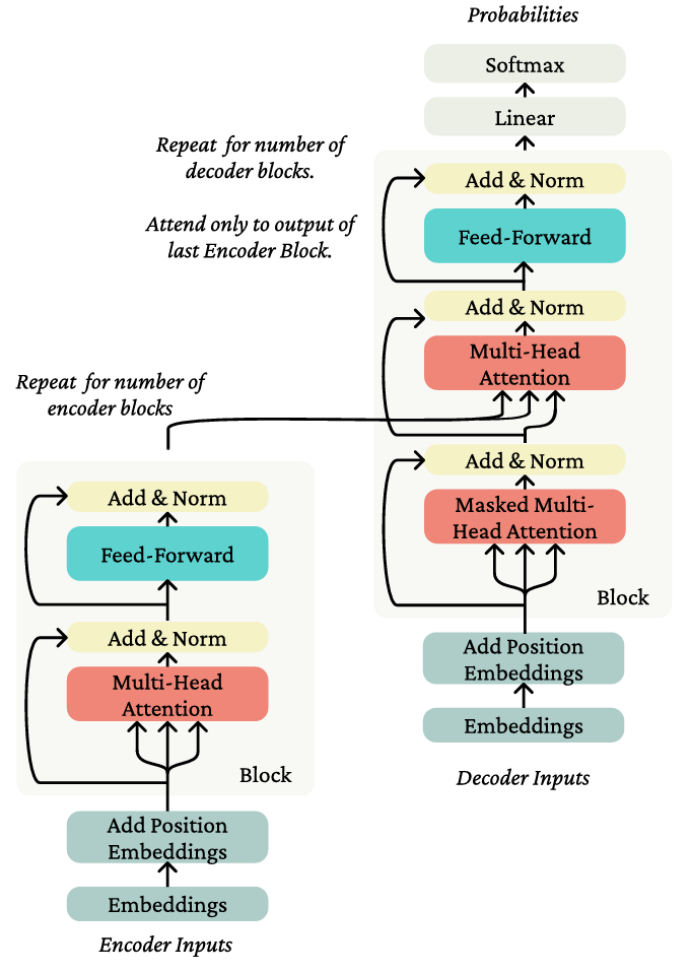
- **Each head gets to "look" at different things, and construct value vectors differently.**

Attention head 1



Attention head 8



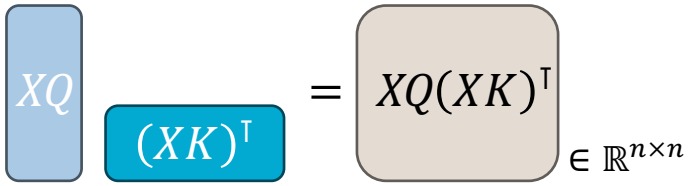


[Transformers]

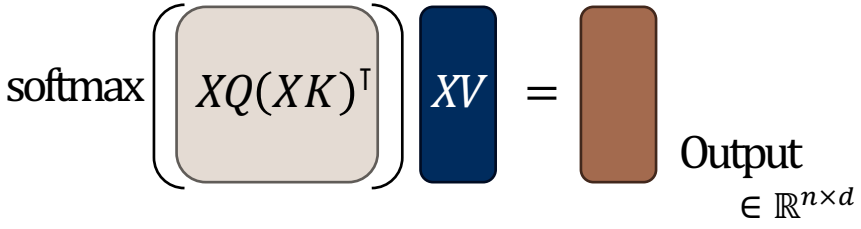
Multi-head Attention

- **Computationally Efficiency**
  - Even though computing h-head, it's not much costly.
  - Let $X = [x_1; \ldots; x_n] \in \mathbb{R}^{n \times d}$ be the concatenation of input vector.
  - $XQ \in \mathbb{R}^{n \times d}$ and likewise for XK and XV.
  - Output is defined as $\text{output} = \text{softmax}(XQ(XK)^\intercal)XV \in \mathbb{R}^{n \times d}$

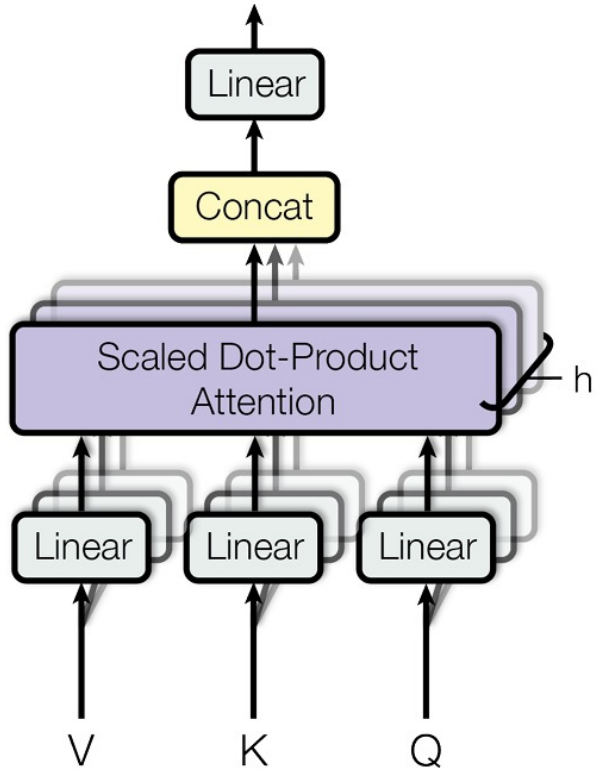
[Attention]



Take the query-key dot products in one matrix multiplication



Take Softmax and compute the weighted average with another matrix multiplication

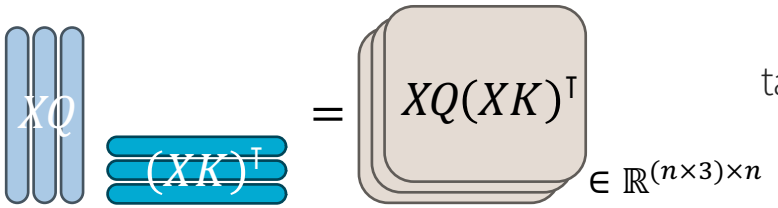


[Multi-Head Attention]

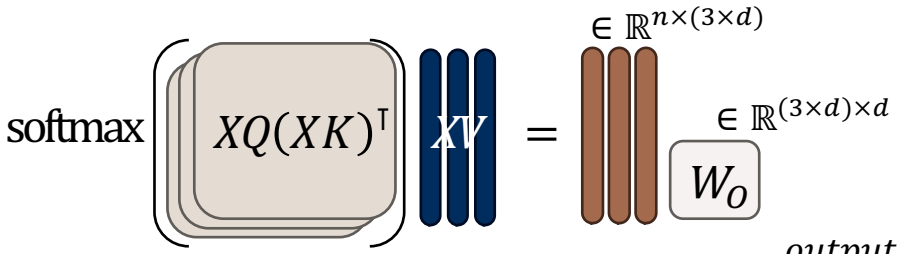
Multi-head Attention

- **Computationally Efficiency**
  - Even though computing h-head, it's not much costly.
  - Let $X = [x_1; \ldots; x_n] \in \mathbb{R}^{n \times d}$ be the concatenation of input vector.
  - $XQ \in \mathbb{R}^{n \times d}$ and likewise for XK and XV.
  - Output is defined as output $= \text{softmax}(XQ(XK)^\intercal)XV \in \mathbb{R}^{n \times d}$
  - And then reshape XQ , XK and XV to $\mathbb{R}^{n \times h \times d/h}$.

[Multi-head Attention]

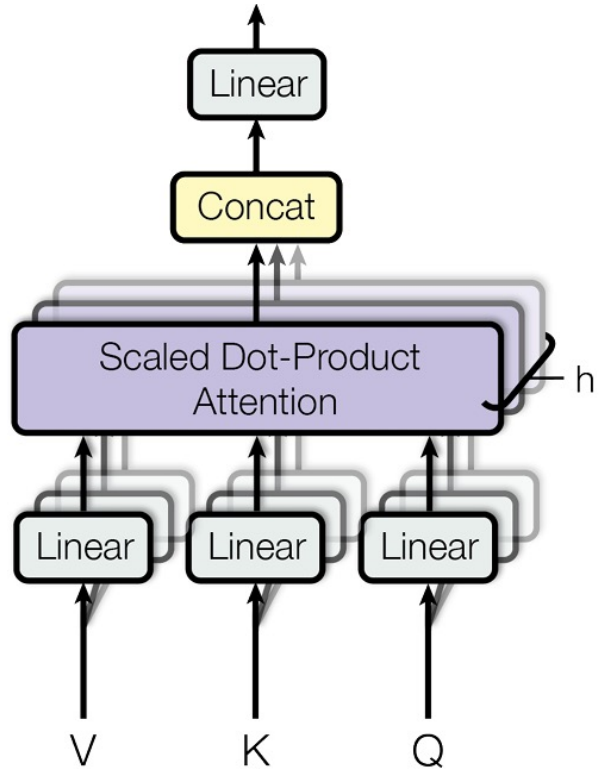


Let $h = 3$,
take the query-key dot products
in one matrix multiplication



Take Softmax and compute
the weighted average
with another matrix multiplication

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_\text{h})W^O$$
$$\text{where head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$



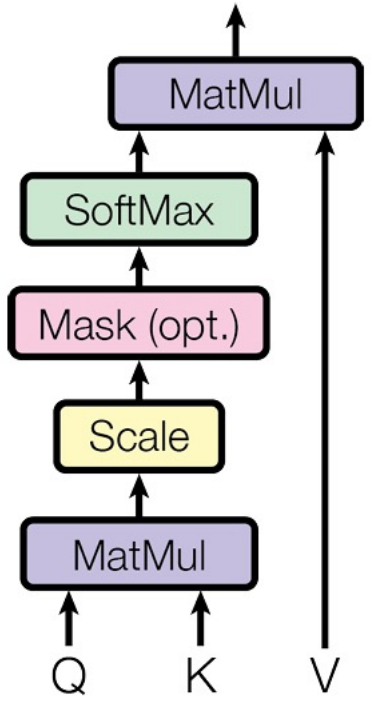
[Multi-Head Attention]

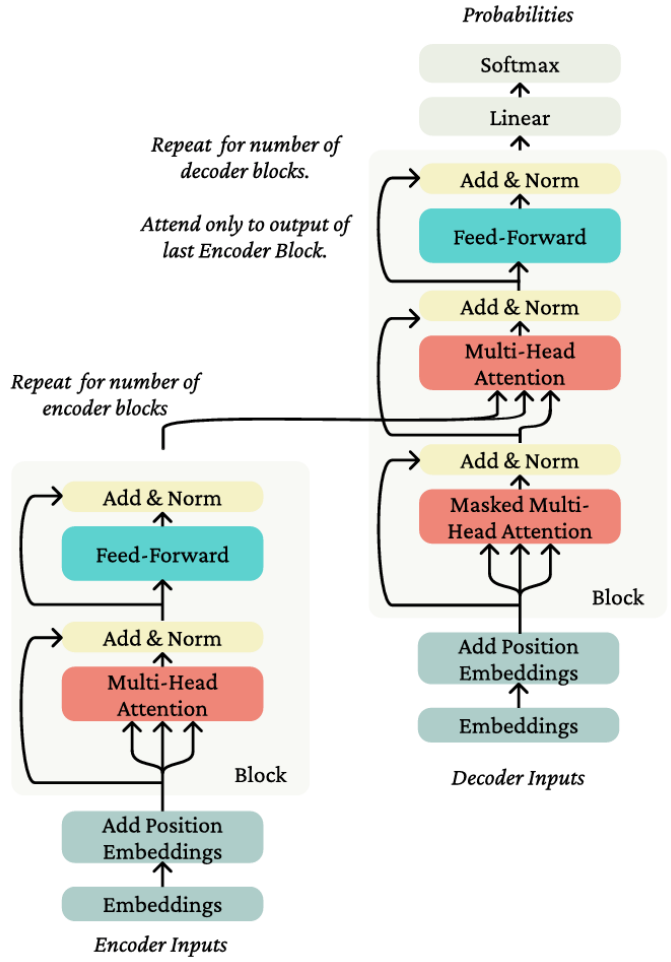
Scaled Dot Product

- **When dimensionality d becomes large, dot products between vectors tend to become large**
  - Because of this, inputs to the softmax fucntion can be large, making the gradients small.
  - Just divide the attenscores by $\sqrt{\mathrm{d}_k}$ to stop the scores from becoming large just as a function of d.

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_\mathrm{K}}})\mathbf{V}$$

  - For multi-head attention , use $\sqrt{\mathrm{d}_k/h}$.



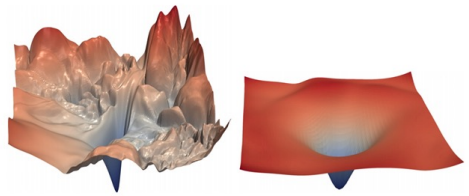


[Transformers]

Optimization Tricks

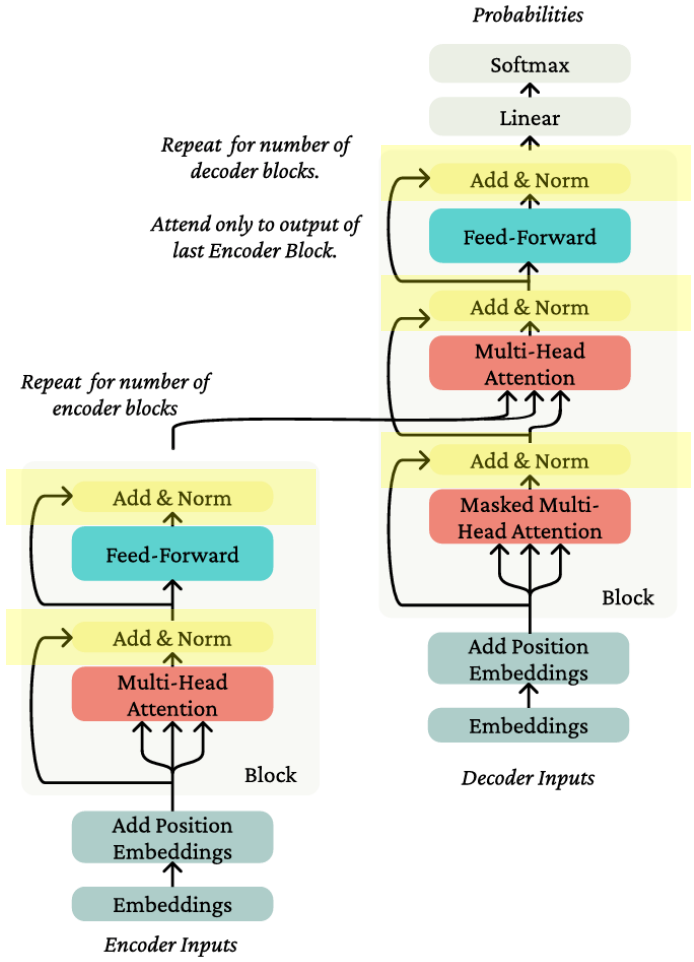
- **Residual Connections** help the model train better
  - Instead of $X^{(i)} = \text{Layer}(X^{(i-1)})$,

  - let $X^{(i)} = X^{(i-1)} + \text{Layer}(X^{(i-1)})$



- **Layer Normalization** help the model train faster
  - Cut down on uninformative variation in hidden vector values by normalizing to unit mean and standard deviation within each layer

$$\text{output} = \frac{x - \mu}{\sqrt{\sigma + \epsilon}} * \gamma + \beta$$



[Transformers]

Experiment Results

Machine Translation

Machine Translation

- **English German, English Franch**

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.

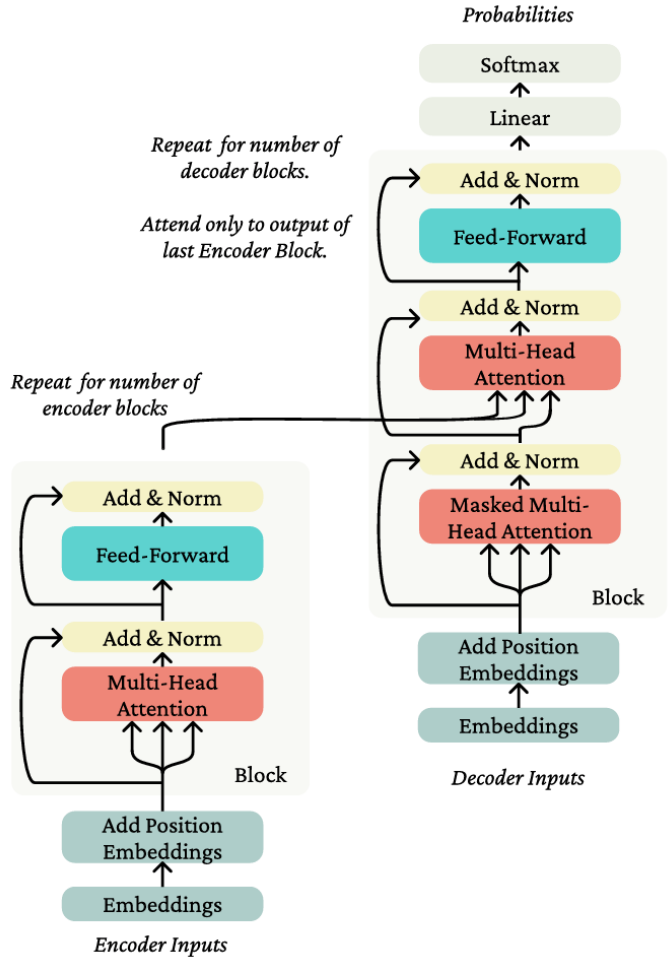
Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
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	**$3.3 \cdot 10^{18}$**	
Transformer (big)	**28.4**	**41.8**	$2.3 \cdot 10^{19}$	

Conclusion

Summary

- **Main Contribution**
  - Introduction of Self-Attention
  - Elimination of Recurrence and Convolution
  - Scalability and Performance

- **And then, …**
  - A foundational architecture in the field of NLP
  - BERT, GPT, and arious other models



[Transformers]

Future Work?

- **Do Transformer Modifications Transfer Across Implementations and Applications?**
  - EMNLP 2021
  - “Surprisingly, we find that most modifications **do not** meaningfully improve performance.”

Model	Params	Ops	Step/s	Early loss	Final loss	SGLUE	XSum	WebQ
Vanilla Transformer	$223M$	$11.1T$	3.90	2.245 ± 0.005	1.865	69.72	16.94	24.60
GeLU	$223M$	$11.1T$	3.88	2.220 ± 0.005	**1.863**	**70.36**	**17.10**	23.28
Swish	$223M$	$11.1T$	3.93	2.234 ± 0.005	1.865	**69.60**	**17.07**	24.34
ELU	$223M$	$11.1T$	3.86	2.333 ± 0.013	1.942	64.30	16.21	24.07
GLU	$223M$	$11.1T$	3.88	2.212 ± 0.005	**1.834**	**70.43**	**17.42**	24.34
GeGLU	$223M$	$11.1T$	3.85	2.172 ± 0.010	**1.807**	**72.36**	**17.69**	**24.87**
ReGLU	$223M$	$11.1T$	3.87	2.190 ± 0.008	**1.832**	**70.63**	**17.38**	21.96
SeLU	$223M$	$11.1T$	3.84	2.372 ± 0.016	1.967	64.68	16.00	23.28
SwiGLU	$223M$	$11.1T$	3.82	2.168 ± 0.006	**1.806**	**70.90**	**17.51**	**25.13**
LiGLU	$223M$	$11.1T$	3.88	2.180 ± 0.002	**1.816**	**71.23**	**17.55**	**24.60**
Sigmoid	$223M$	$11.1T$	3.94	2.947 ± 1.152	1.908	69.36	16.64	23.02
Softplus	$223M$	$11.1T$	3.77	2.324 ± 0.032	1.885	68.99	16.92	21.96
RMS Norm	$223M$	$11.1T$	3.99	2.209 ± 0.008	**1.856**	69.11	16.90	23.55
Rezero	$223M$	$11.1T$	4.14	3.180 ± 0.719	2.506	54.01	6.44	20.90
Rezero + LayerNorm	$223M$	$11.1T$	3.78	2.229 ± 0.006	1.902	64.75	16.40	23.02
Rezero + RMS Norm	$223M$	$11.1T$	3.90	2.306 ± 0.016	1.948	59.86	15.66	23.02
Fixup	$223M$	$11.1T$	3.32	2.473 ± 0.014	2.236	57.98	12.51	23.28
24 layers, $d_{\text{ff}} = 1536, H = 6$	$224M$	$11.1T$	3.12	2.260 ± 0.014	1.874	**70.59**	**17.11**	23.02
18 layers, $d_{\text{ff}} = 2048, H = 8$	$223M$	$11.1T$	3.27	2.268 ± 0.037	1.878	**70.40**	16.87	23.02
8 layers, $d_{\text{ff}} = 4608, H = 18$	$223M$	$11.1T$	3.61	2.243 ± 0.003	1.871	68.67	**17.03**	23.55
6 layers, $d_{\text{ff}} = 6144, H = 24$	$223M$	$11.1T$	3.59	2.250 ± 0.004	1.882	68.08	16.93	23.81
Block sharing	$65M$	$11.1T$	4.03	2.777 ± 0.019	2.237	63.06	13.89	21.96
+ Factorized embeddings	$45M$	$9.4T$	4.35	2.670 ± 0.178	2.205	57.17	12.13	20.11
+ Factorized & Shared embeddings	$20M$	$9.1T$	4.49	2.874 ± 0.059	2.362	57.46	11.78	19.58
Encoder only block sharing	$170M$	$11.1T$	3.80	2.399 ± 0.008	2.016	64.08	14.74	21.69
Decoder only block sharing	$144M$	$11.1T$	3.92	2.542 ± 0.067	2.048	**69.95**	16.01	21.96
Factorized Embedding	$227M$	$9.4T$	3.97	2.273 ± 0.019	1.886	68.91	16.41	21.43
Factorized & shared embeddings	$202M$	$9.1T$	4.08	2.387 ± 0.006	2.018	**69.93**	16.07	21.96
Tied encoder/decoder input embeddings	$248M$	$11.1T$	3.86	2.254 ± 0.008	1.872	68.34	16.60	22.75
Tied decoder input and output embeddings	$248M$	$11.1T$	3.86	2.262 ± 0.006	1.871	69.48	16.85	23.28
Untied embeddings	$273M$	$11.1T$	3.83	2.265 ± 0.013	1.872	67.99	16.66	23.02
Adaptive input embeddings	$204M$	$9.2T$	4.15	2.321 ± 0.006	1.934	69.20	16.69	21.96
Adaptive softmax	$204M$	$9.2T$	4.21	2.425 ± 0.005	2.009	67.71	15.74	20.11
Adaptive softmax without projection	$223M$	$10.8T$	3.97	2.357 ± 0.009	1.937	68.68	16.45	22.75
Mixture of softmaxes	$232M$	$16.3T$	2.50	3.112 ± 1.169	**1.843**	**70.70**	16.78	22.75
Relative attention with bias	$223M$	$11.3T$	3.49	2.197 ± 0.005	**1.832**	**74.06**	**17.63**	**24.87**
Relative attention with shared bias	$223M$	$11.3T$	3.57	2.194 ± 0.006	**1.840**	**74.14**	**17.62**	24.34
Relative position representation	$223M$	$11.1T$	3.10	2.189 ± 0.008	**1.838**	**74.26**	**17.67**	24.07
Sinusoidal positional encoding	$223M$	$11.1T$	3.91	2.278 ± 0.032	1.906	**69.76**	16.25	22.75
Transparent attention	$223M$	$11.1T$	3.61	2.244 ± 0.013	1.949	53.77	6.39	15.08
Dynamic convolution	$257M$	$11.8T$	2.65	2.405 ± 0.007	2.038	55.16	10.25	4.50
Lightweight convolution	$224M$	$10.4T$	4.05	2.356 ± 0.006	1.990	61.32	14.08	24.08
Evolved Transformer	$217M$	$9.7T$	3.11	2.233 ± 0.004	1.890	67.88	16.40	24.08
Synthesizer (dense)	$224M$	$11.4T$	3.61	2.339 ± 0.019	1.965	61.02	14.48	18.25
Synthesizer (dense plus)	$243M$	$12.6T$	3.34	2.200 ± 0.008	**1.832**	**74.16**	**16.96**	**24.87**
Synthesizer (dense plus alpha)	$243M$	$12.6T$	3.11	2.204 ± 0.005	**1.846**	**75.18**	16.94	**24.60**
Synthesizer (factorized)	$207M$	$10.1T$	4.10	2.629 ± 0.573	1.964	61.76	15.44	22.49
Synthesizer (random)	$254M$	$10.1T$	4.26	2.458 ± 0.167	1.972	64.61	15.39	23.02
Synthesizer (random plus)	$292M$	$12.0T$	3.79	2.202 ± 0.010	**1.849**	**76.84**	**17.04**	23.02
Synthesizer (random plus alpha)	$292M$	$12.0T$	3.55	2.212 ± 0.013	**1.856**	**75.02**	**17.08**	**24.87**
Universal Transformer	$84M$	$40.0T$	0.88	2.443 ± 0.022	2.111	60.54	12.02	17.73
Mixture of experts	$648M$	$11.7T$	3.20	2.194 ± 0.008	**1.846**	68.82	**17.12**	**24.87**
Switch Transformer	$1100M$	$11.8T$	3.41	2.175 ± 0.005	**1.775**	**72.21**	**17.78**	**24.87**
Funnel Transformer	$223M$	$1.9T$	4.83	2.291 ± 0.008	1.925	67.11	16.33	21.64

Thank You

Yejin Yoon

HYU NLP Lab.
Hanyang University, South Korea

stillwithyou@hanyang.ac.kr