香港中文大學（深圳）
The Chinese University of Hong Kong, Shenzhen

CSC6052/5051/4100/DDA6307/MDS5110 Natural Language Processing

Lecture 4-2: Transformer and beyond.

Spring 2025

Benyou Wang
School of Data Science

To recap…

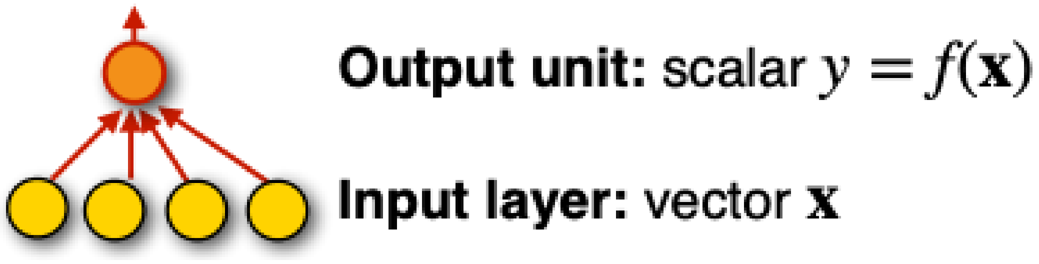
Last lecture

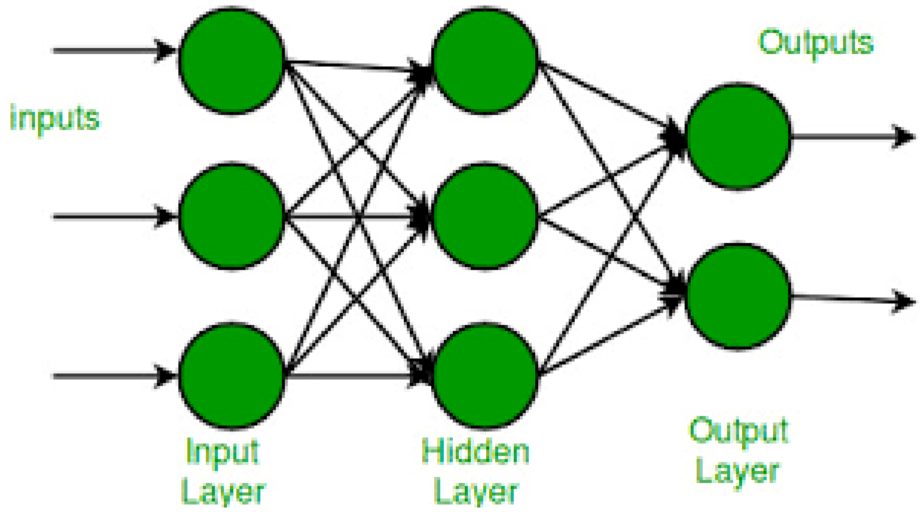
- **MLP**
  +: Strongest inductive bias: if all words are concatenated
  +: Weakest inductive bias: if all words are averaged
  - : The interaction at the token-level is too weak
- CNN & RNN
  +: The interaction at the token-level is slightly better.
    CNN: Bringing the global token-level interaction to the window-level
      - : Make simplifications, its global dependencies are limited
    RNN: An ideal method for processing token sequences
      - : Its recursive nature has the problem of disaster forgetting.
- Transformer
  +: Achieve **global dependence** at the **token-level** by **decoupling** token-level interaction and feature-level abstraction into two components, in **SAN** and **FNN**.

Multilayer Perceptron (MLP)

Definition: The Multilayer Perceptron (MLP) is a type of artificial neural network (ANN) that consists of multiple layers of interconnected artificial neurons or perceptrons.

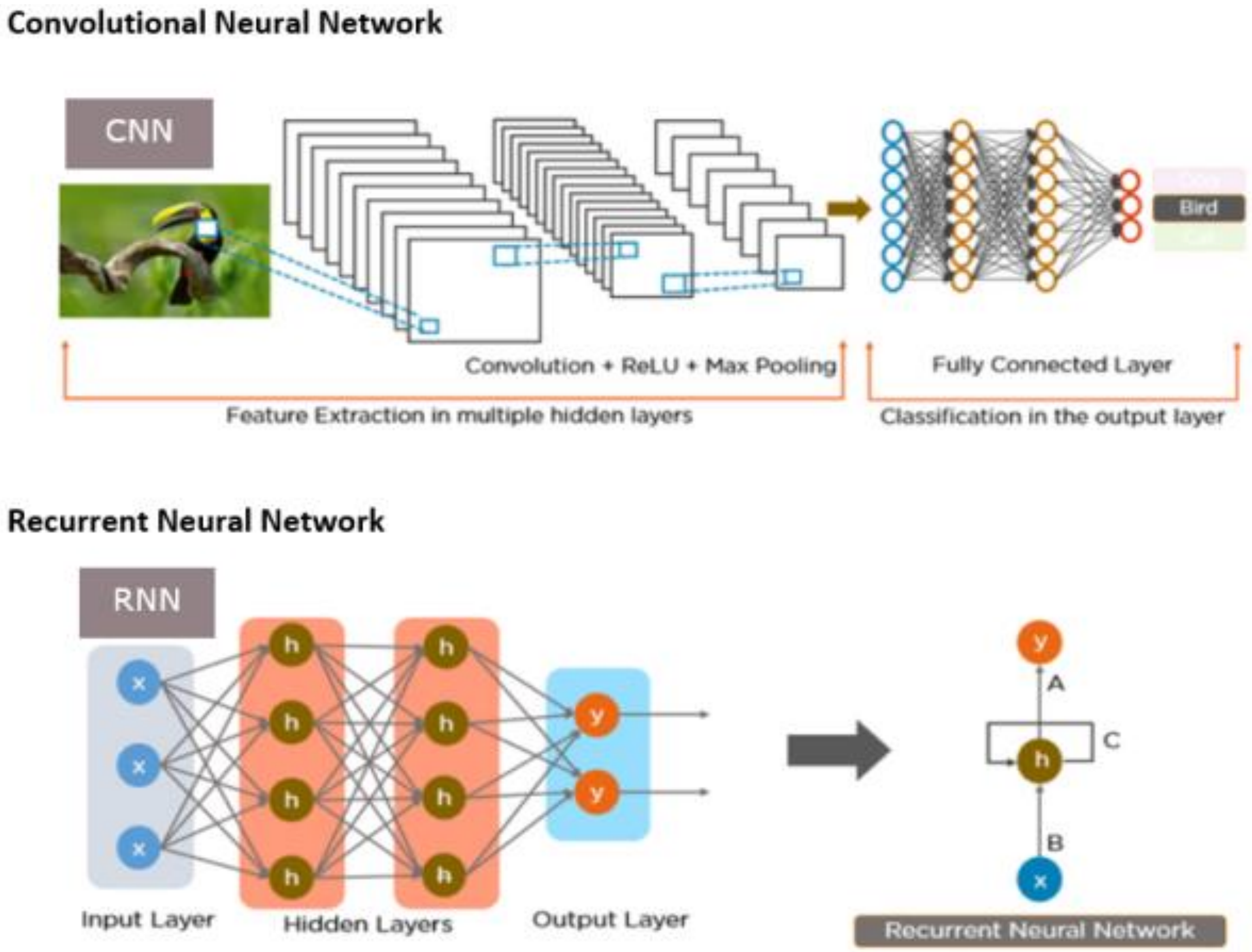
A **perceptron** can be seen as a single neuron (one output unit with a vector or **layer** of input units):





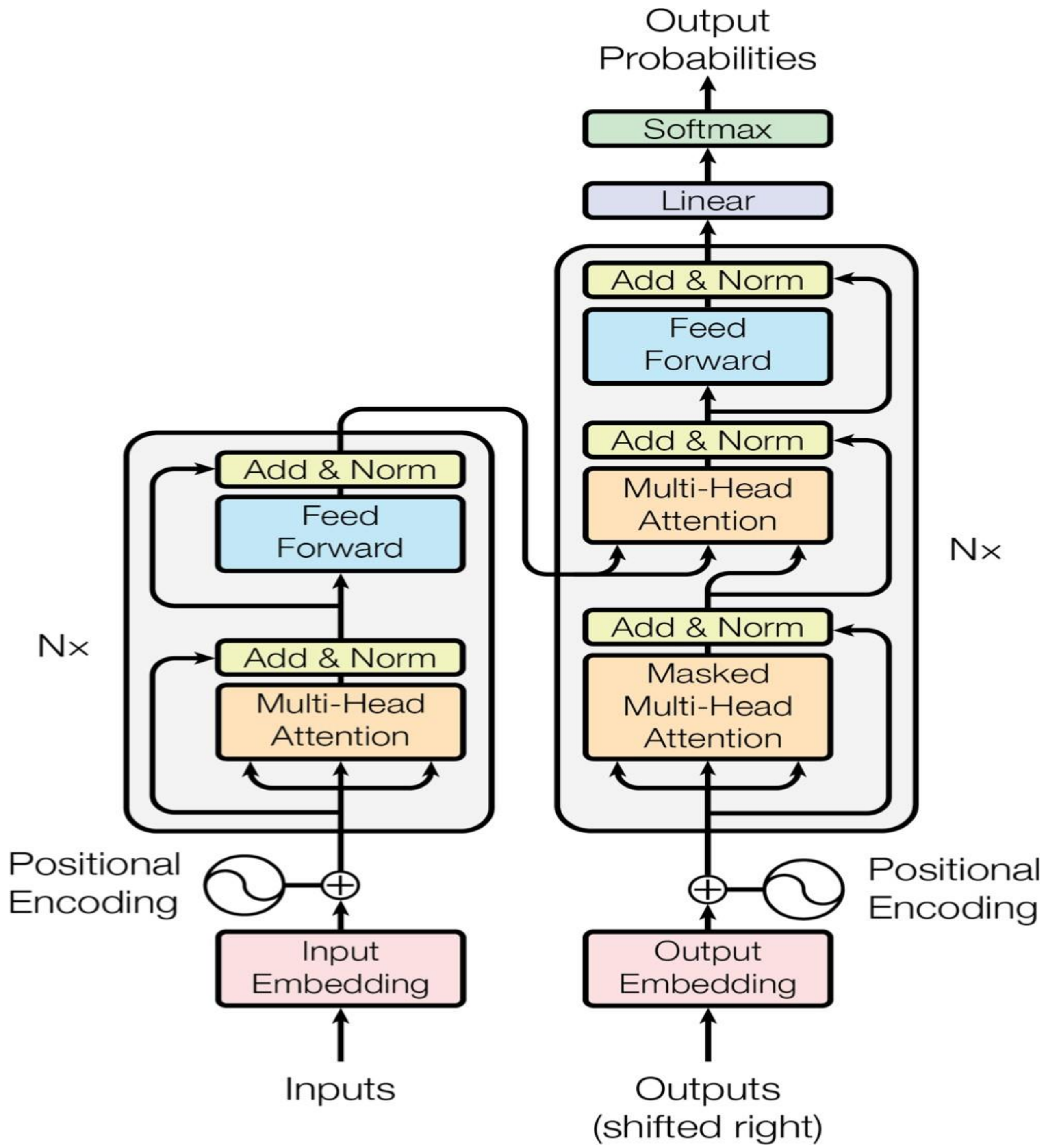
CNN&RNN

- Convolutional Neural Network (CNN)
- Recurrent Neural Network (RNN)

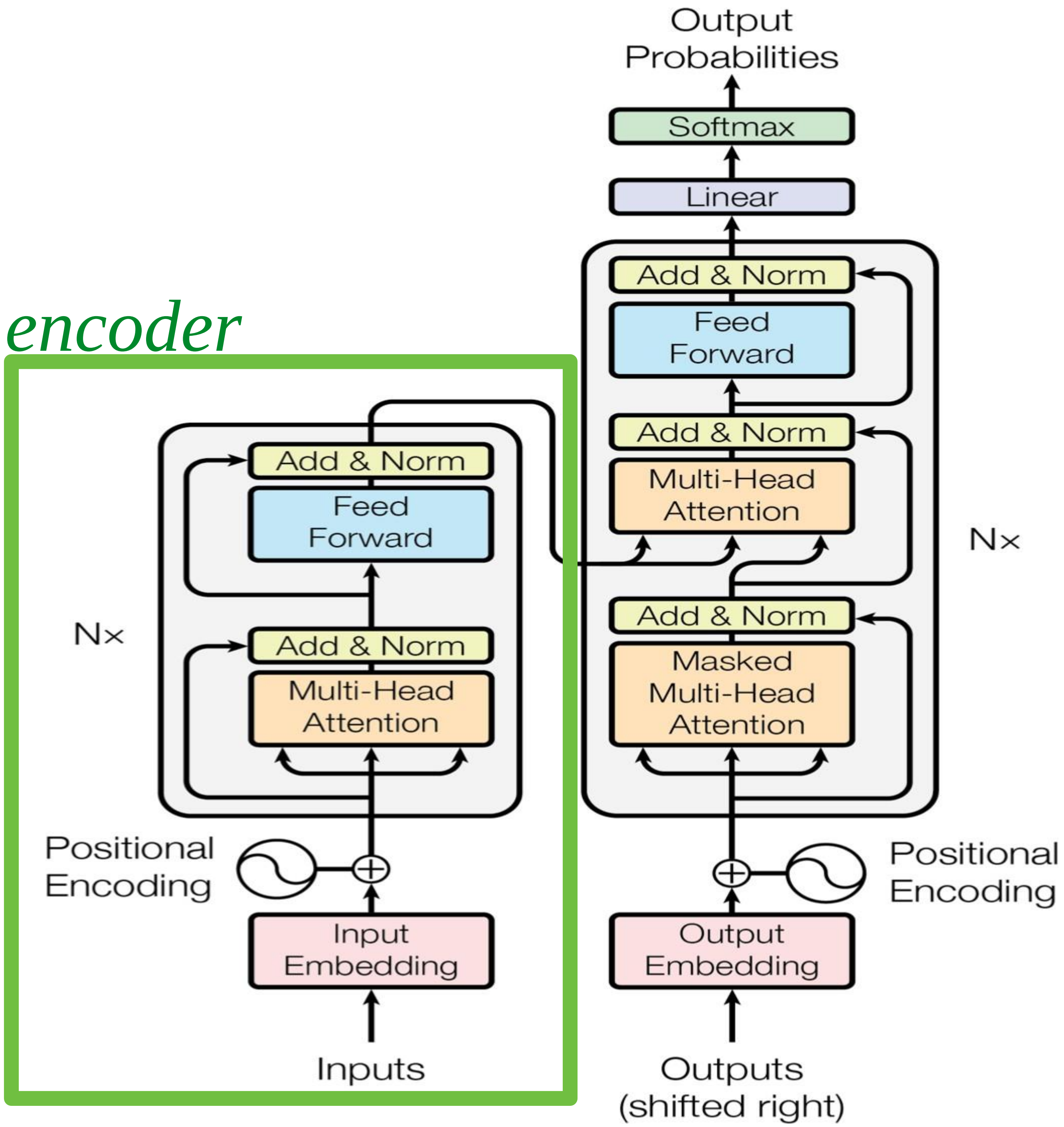


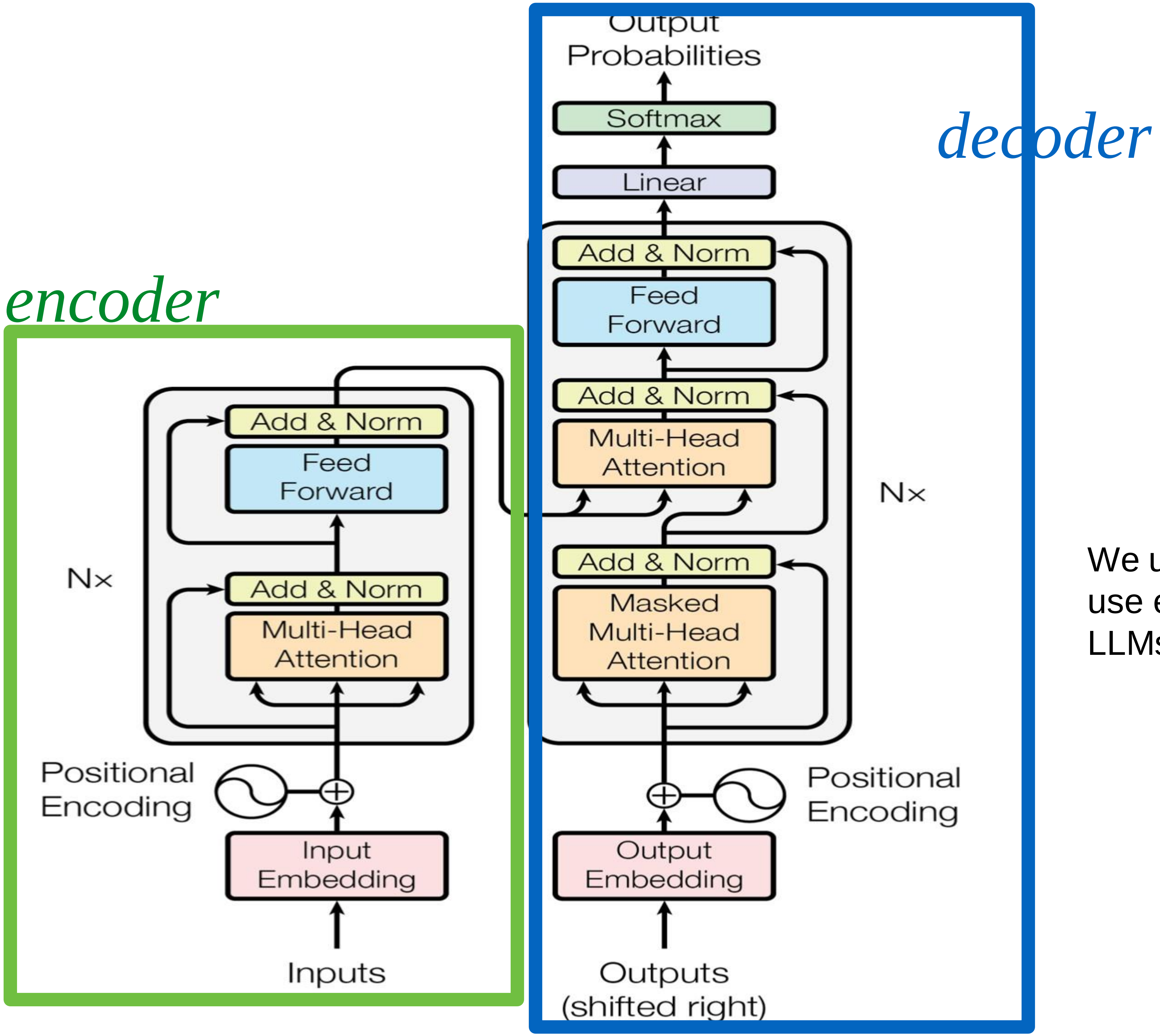
Today’s Lecture： Transformer

- **Encoder**
- **Decoder**
- **Self-attention**
- **Multi-head self-attention**
- **Positional Encoding**

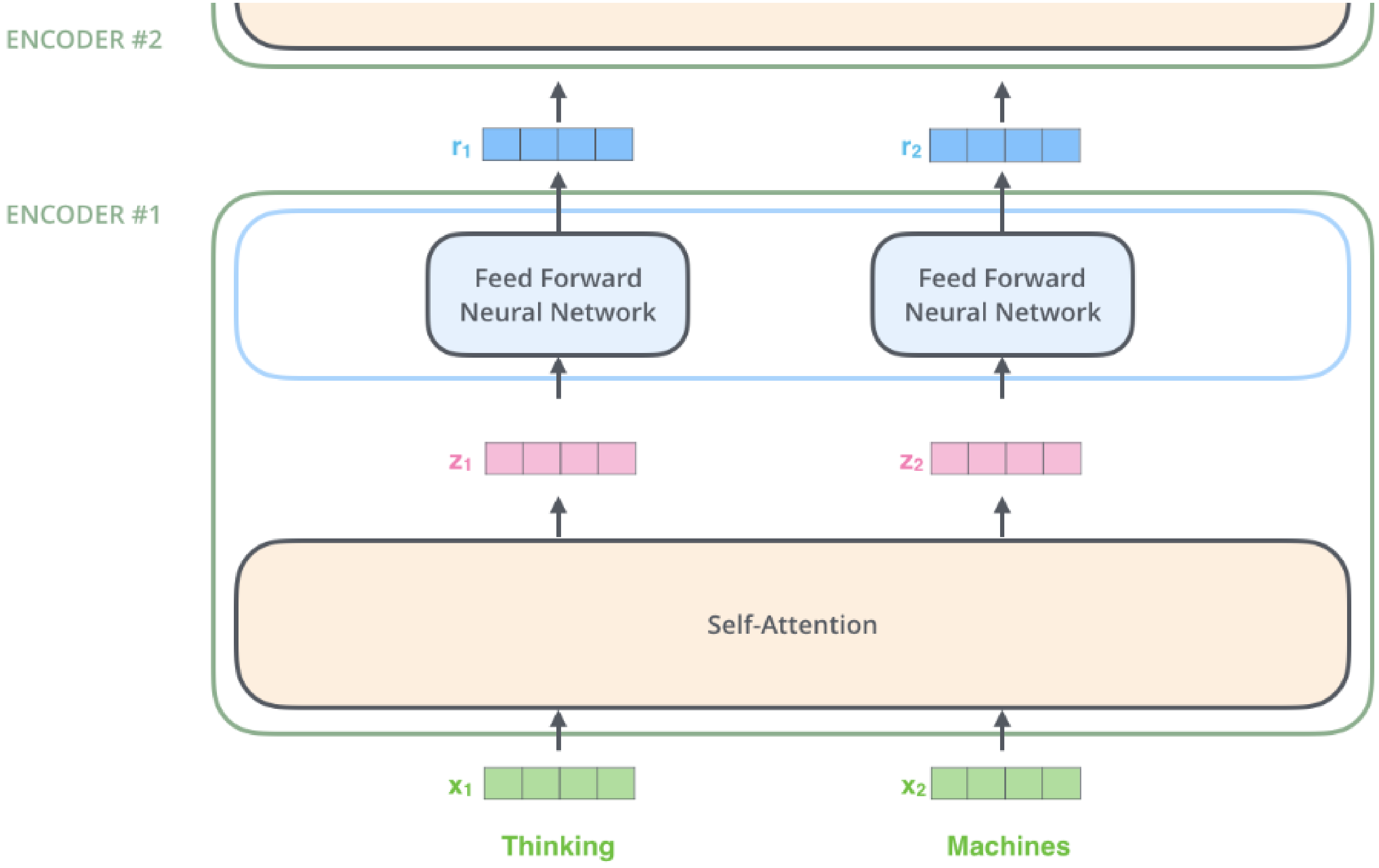


Output
Probabilities
Softmax
Linear
Add & Norm
Feed
Forward
Add & Norm
Multi-Head
Attention
Add & Norm
Masked
Multi-Head
Attention
Nx
Add & Norm
Feed
Forward
Add & Norm
Multi-Head
Attention
Nx
Positional
Encoding
Positional
Encoding
Input
Embedding
Output
Embedding
Inputs
Outputs
(shifted right)


encoder
Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
Nx
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Nx
Positional Encoding
Input Embedding
Inputs
Positional Encoding
Output Embedding
Outputs (shifted right)



We usually do not use encoder in LLMs (decoder-only)

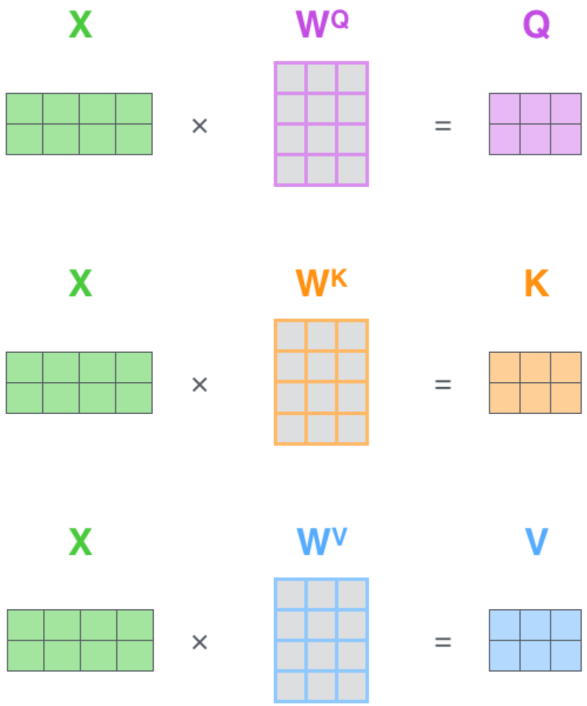


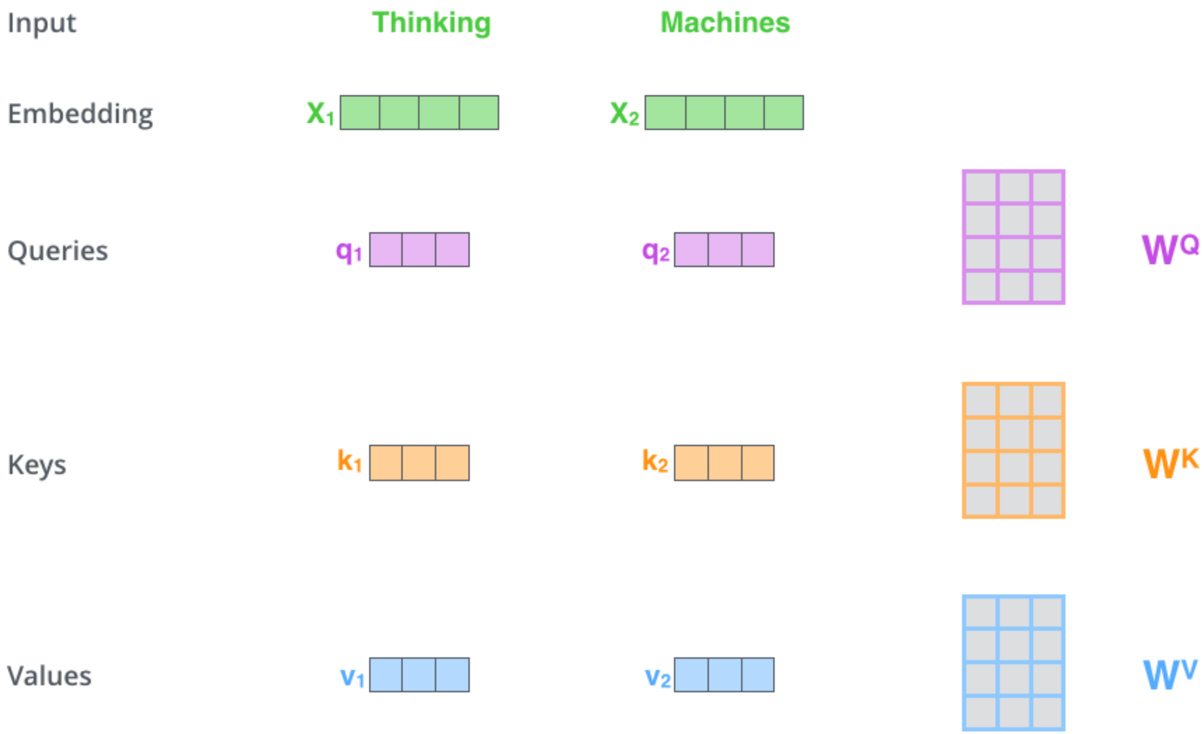
The word at each position passes through a self-attention process. Then, they each pass through a feed-forward neural network -- the exact same network with each vector flowing through it separately.

Token abstraction

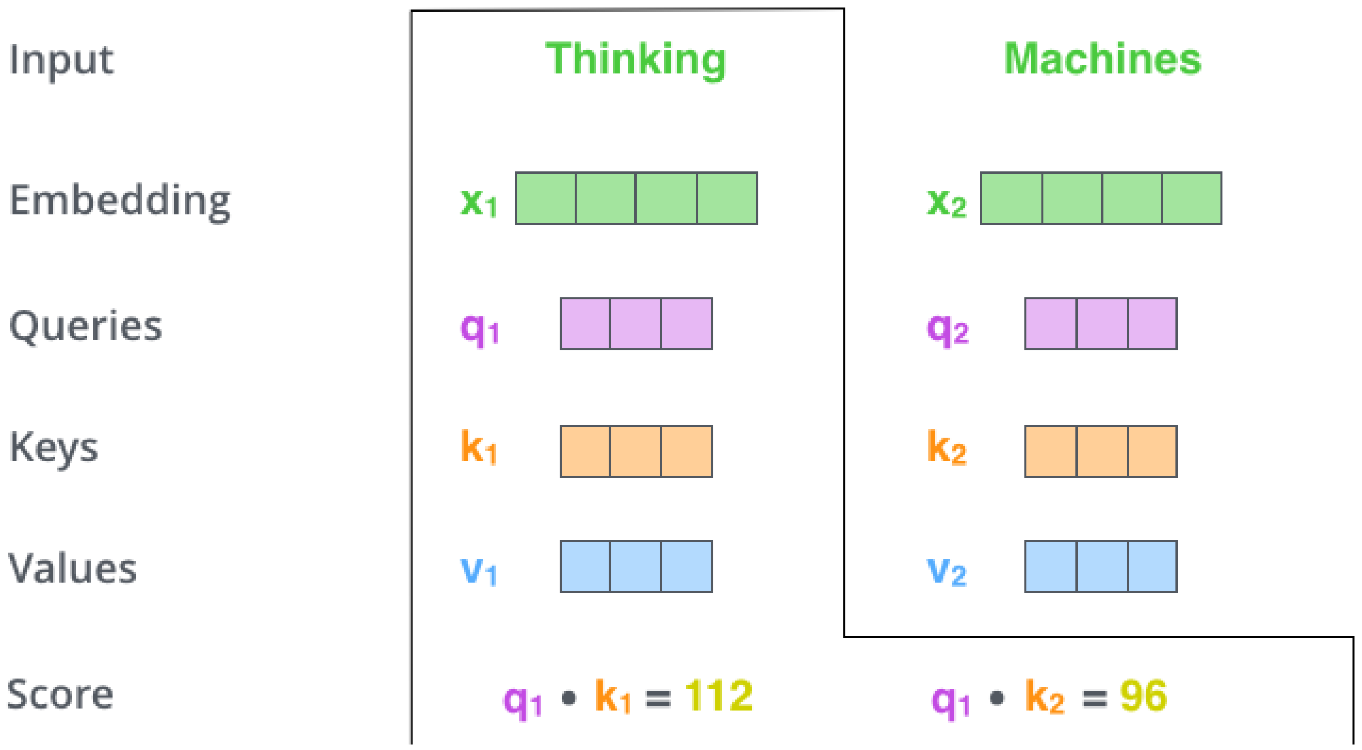
Between-token interaction

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

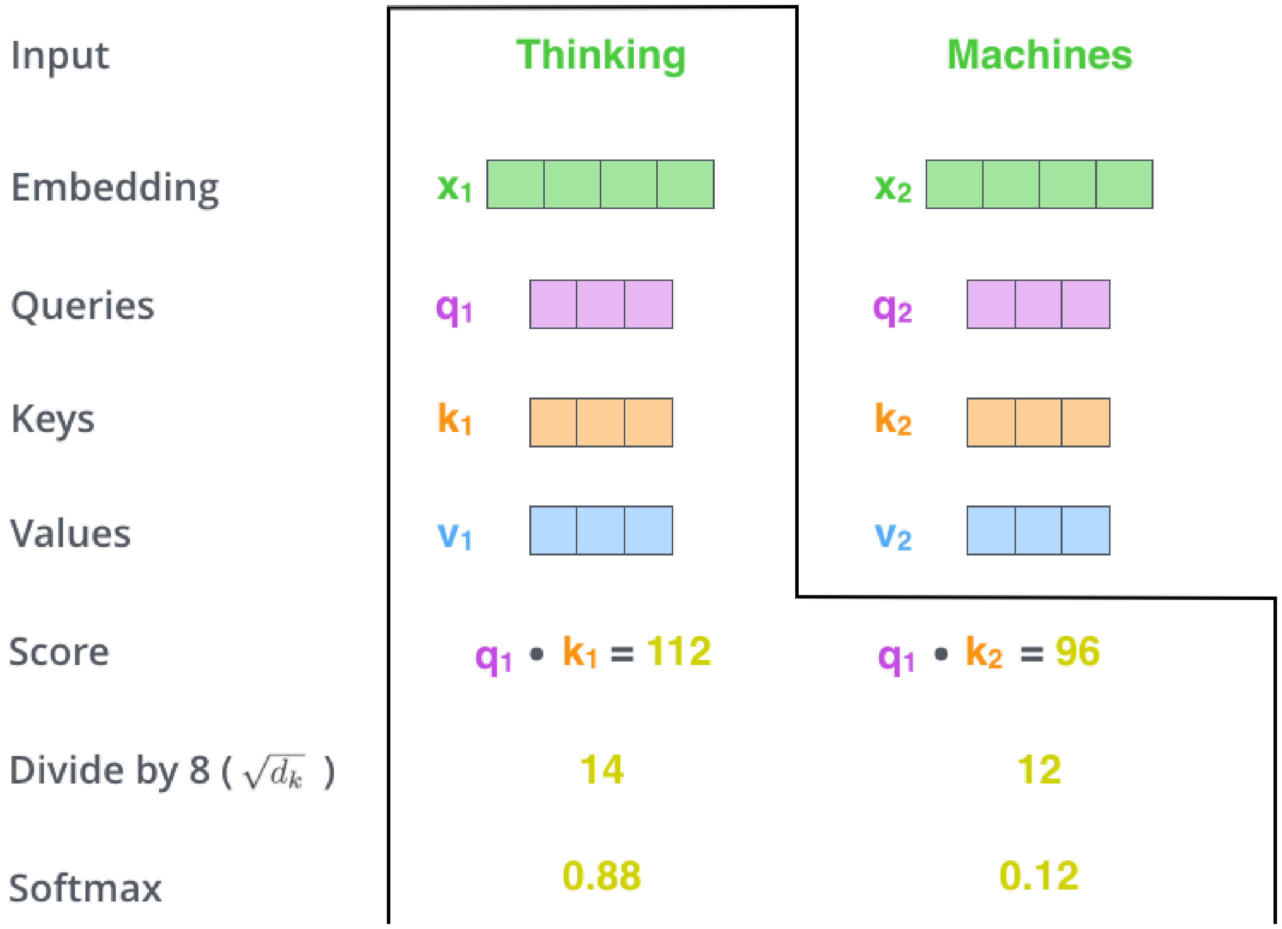




Multi-faced token representation (QKV)

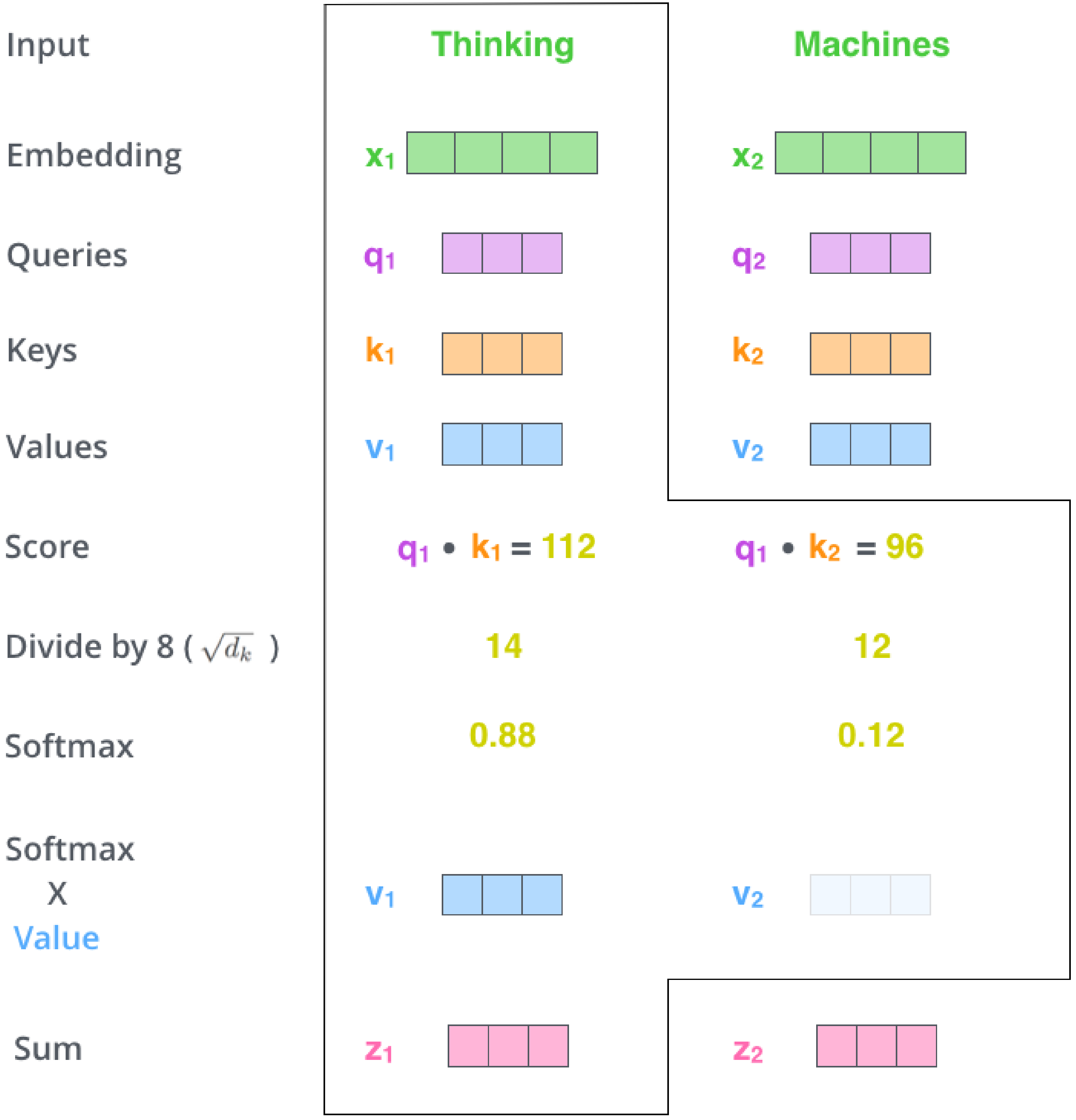


Key-value interaction between tokens



Normalize the **Key-value** interaction as attention (a probability distribution)

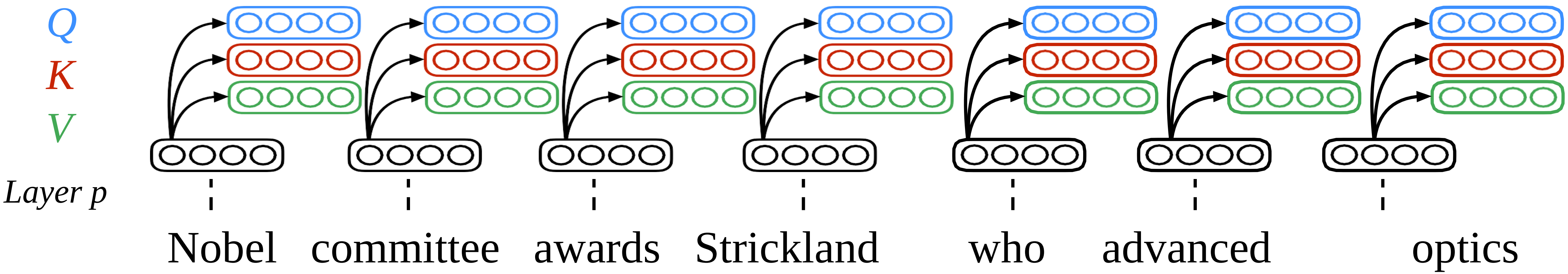
p.s. why softmax makes a probability distribution?



Input
Thinking
Machines
Embedding
x1
x2
Queries
q1
q2
Keys
k1
k2
Values
v1
v2
Score
q1 • k1 = 112
q1 • k2 = 96
Divide by 8 ($\sqrt{d_k}$)
14
12
Softmax
0.88
0.12
Softmax
X
Value
v1
v2
Sum
z1
z2

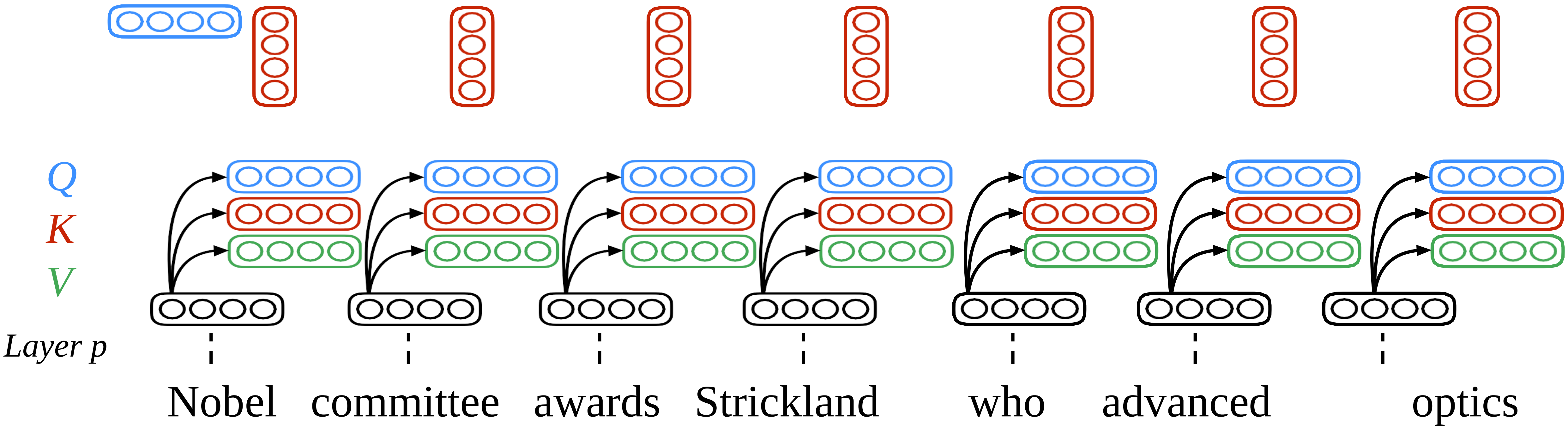
[Vaswani et al. 2017]

Self-attention (in encoder)



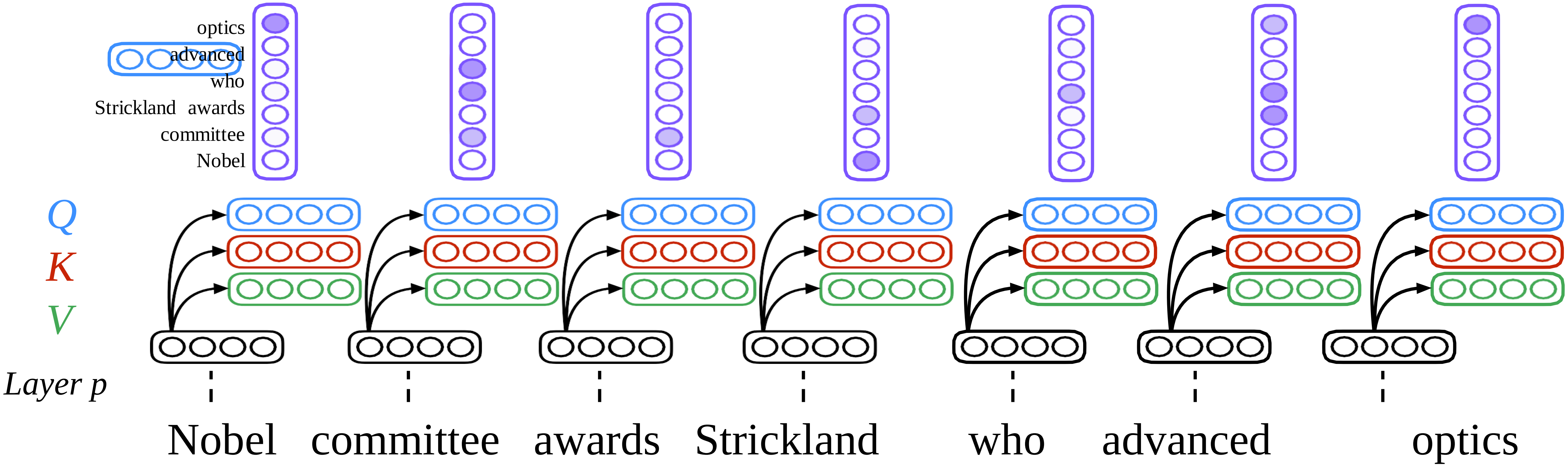
[Vaswani et al. 2017]

Self-attention (in encoder)



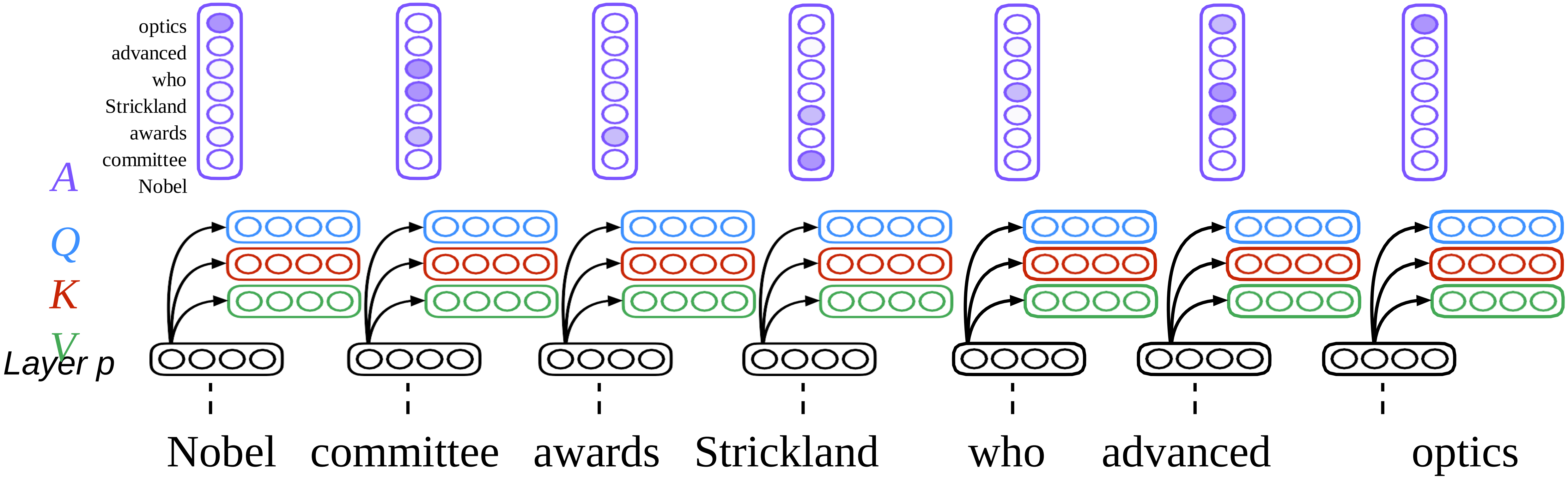
[Vaswani et al. 2017]

Self-attention (in encoder)



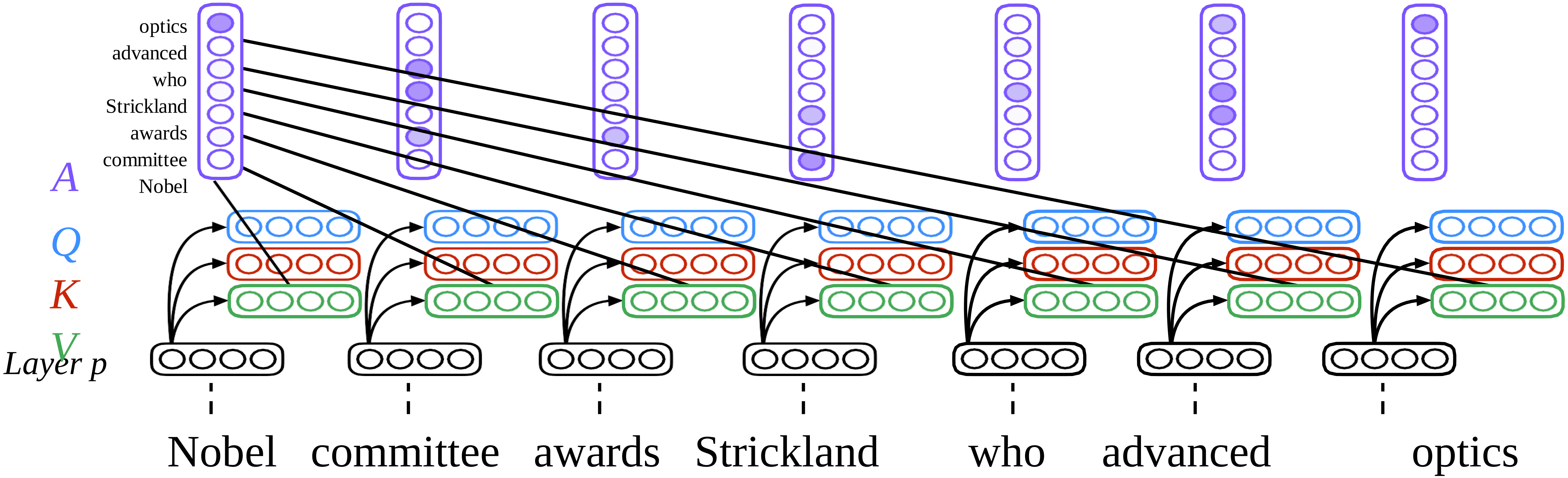
[Vaswani et al. 2017]

Self-attention (in encoder)



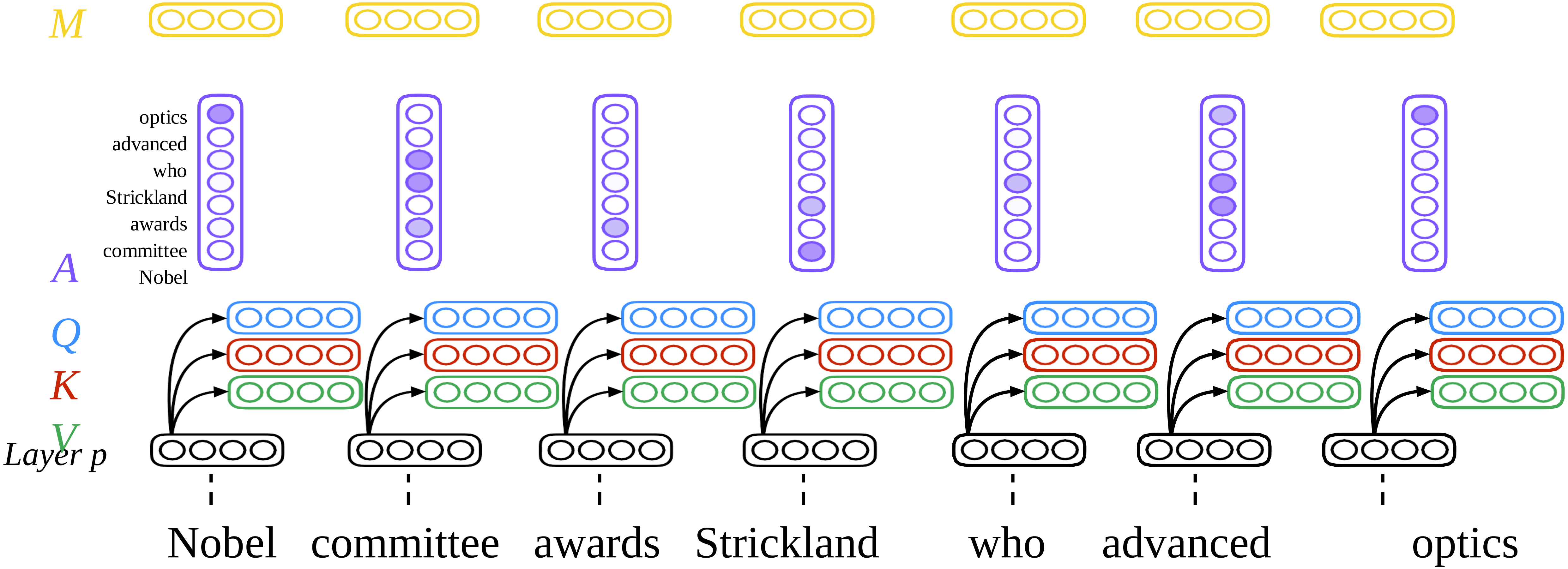
[Vaswani et al. 2017]

Self-attention (in encoder)



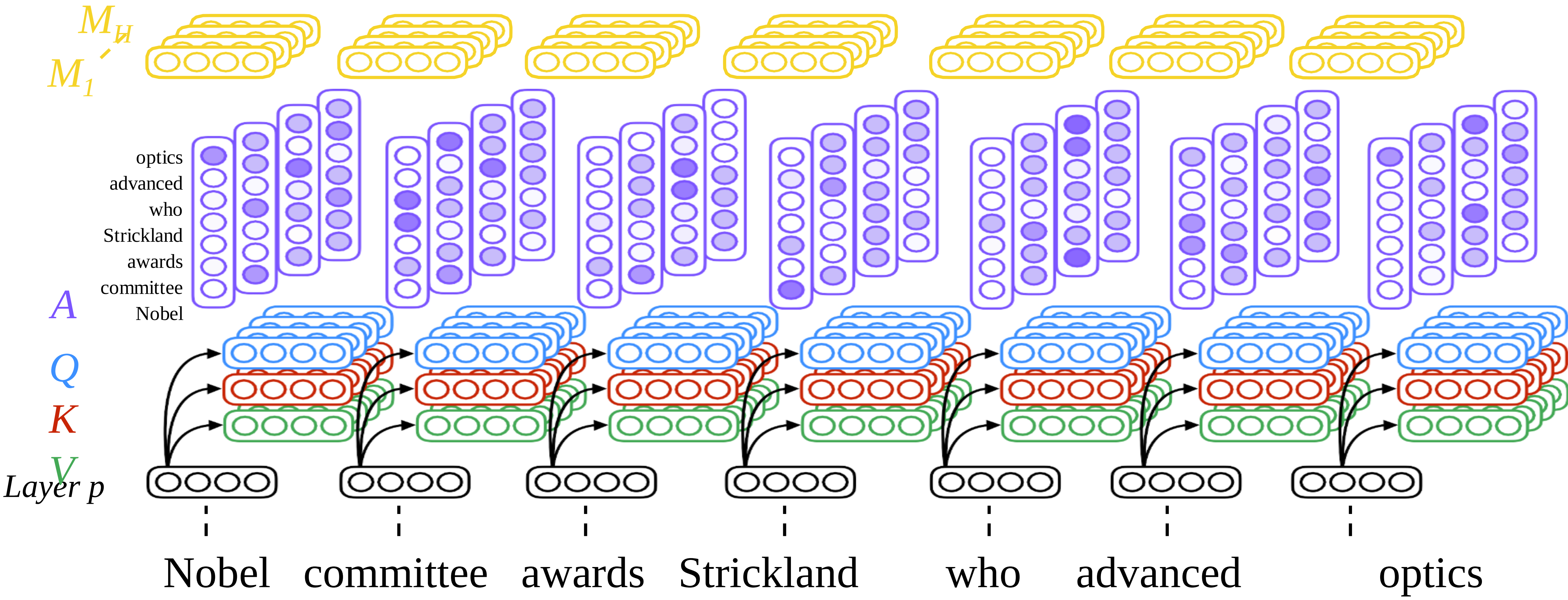
[Vaswani et al. 2017]

Self-attention (in encoder)

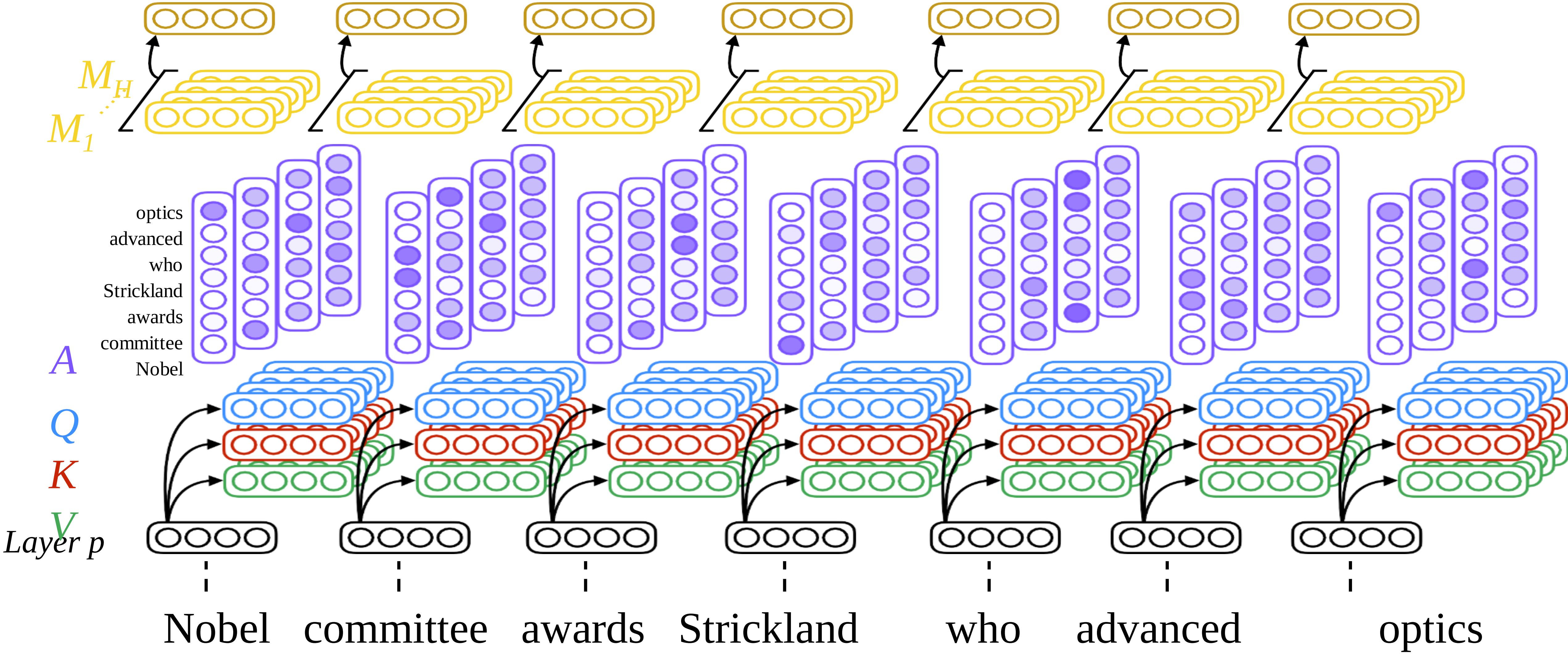


Multi-head self-attention

[Vaswani et al. 2017]

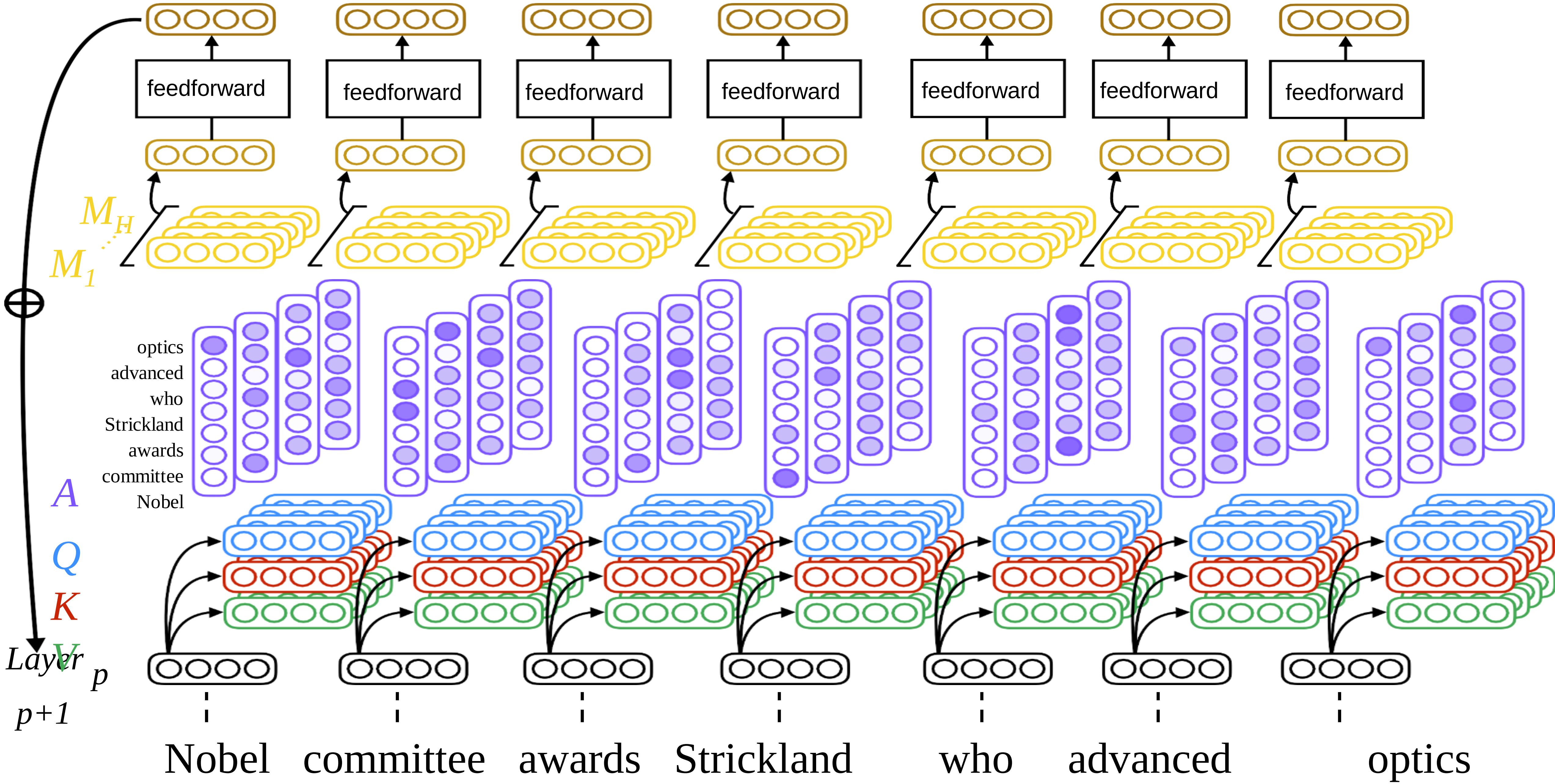


Multi-head self-attention

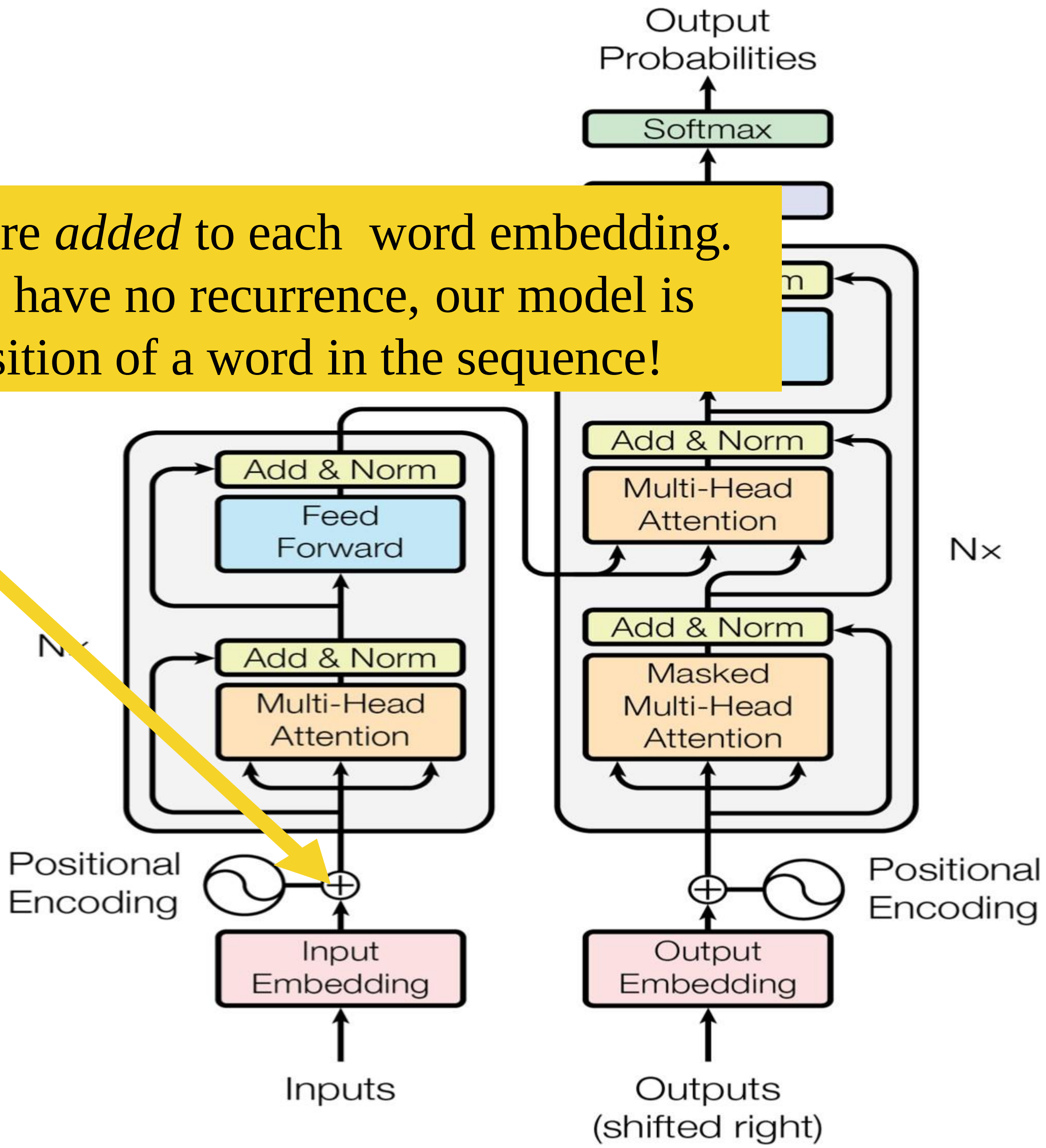


Multi-head self-attention

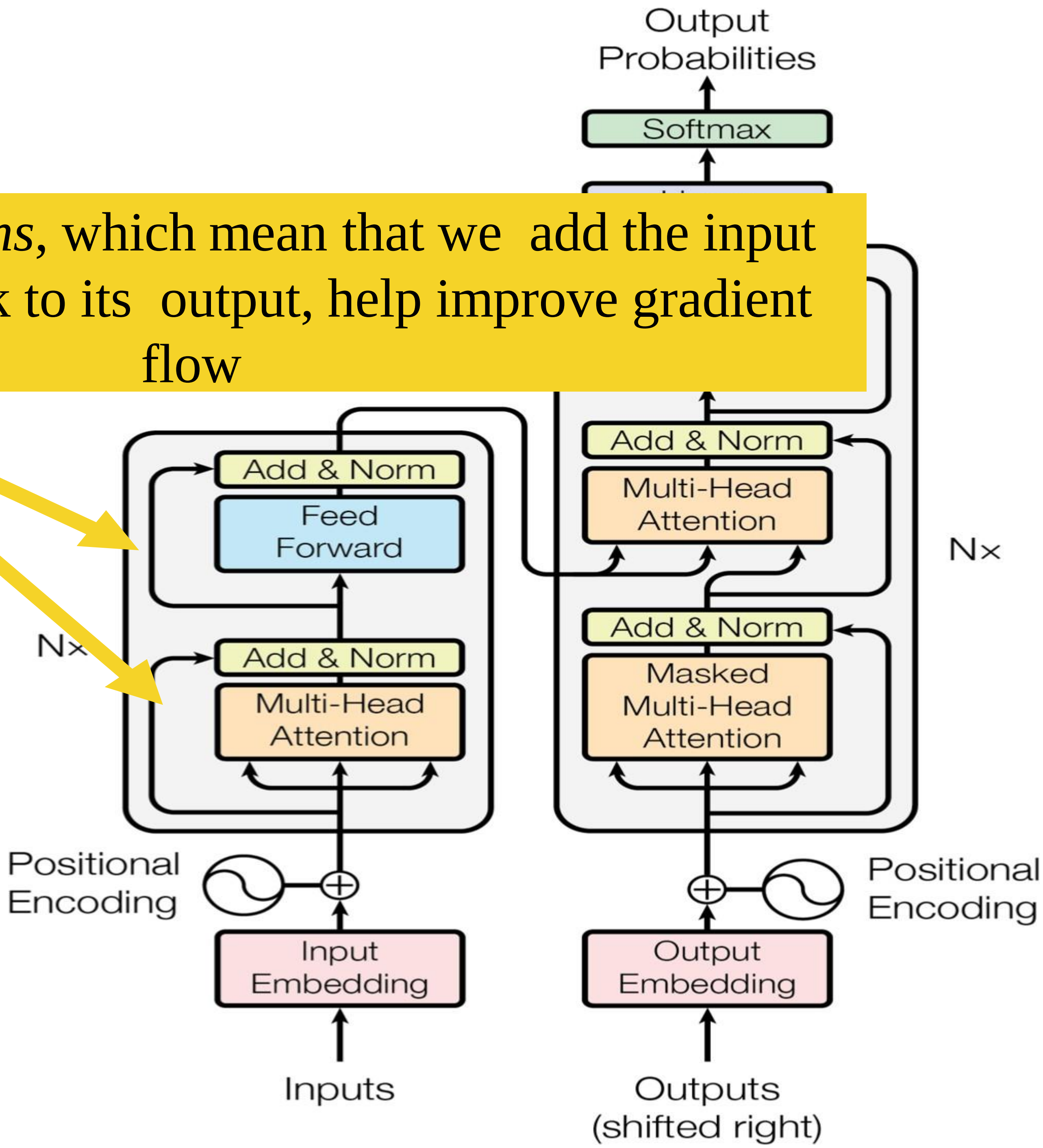
[Vaswani et al. 2017]



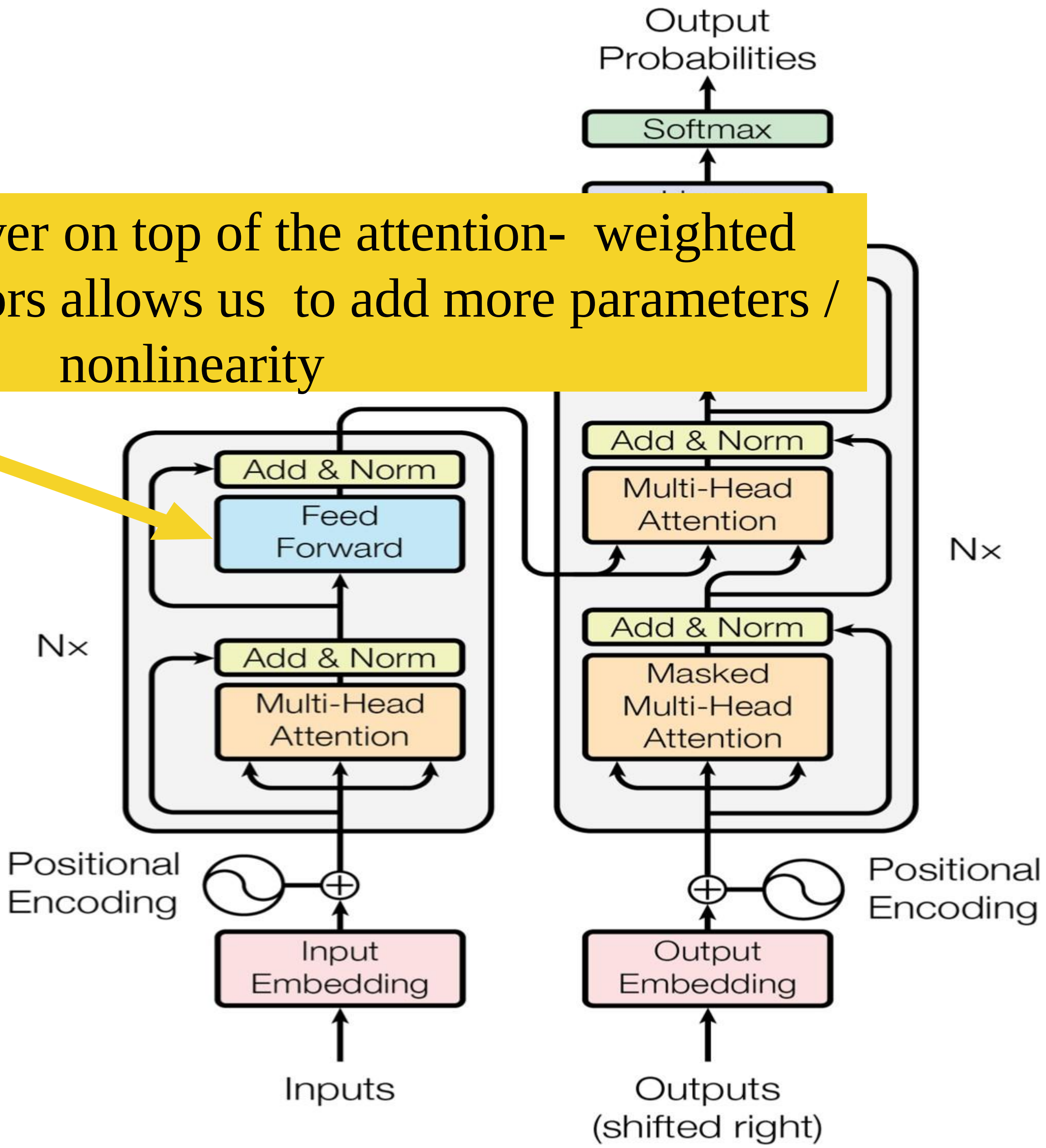
Position embeddings are *added* to each word embedding. Otherwise, since we have no recurrence, our model is unaware of the position of a word in the sequence!



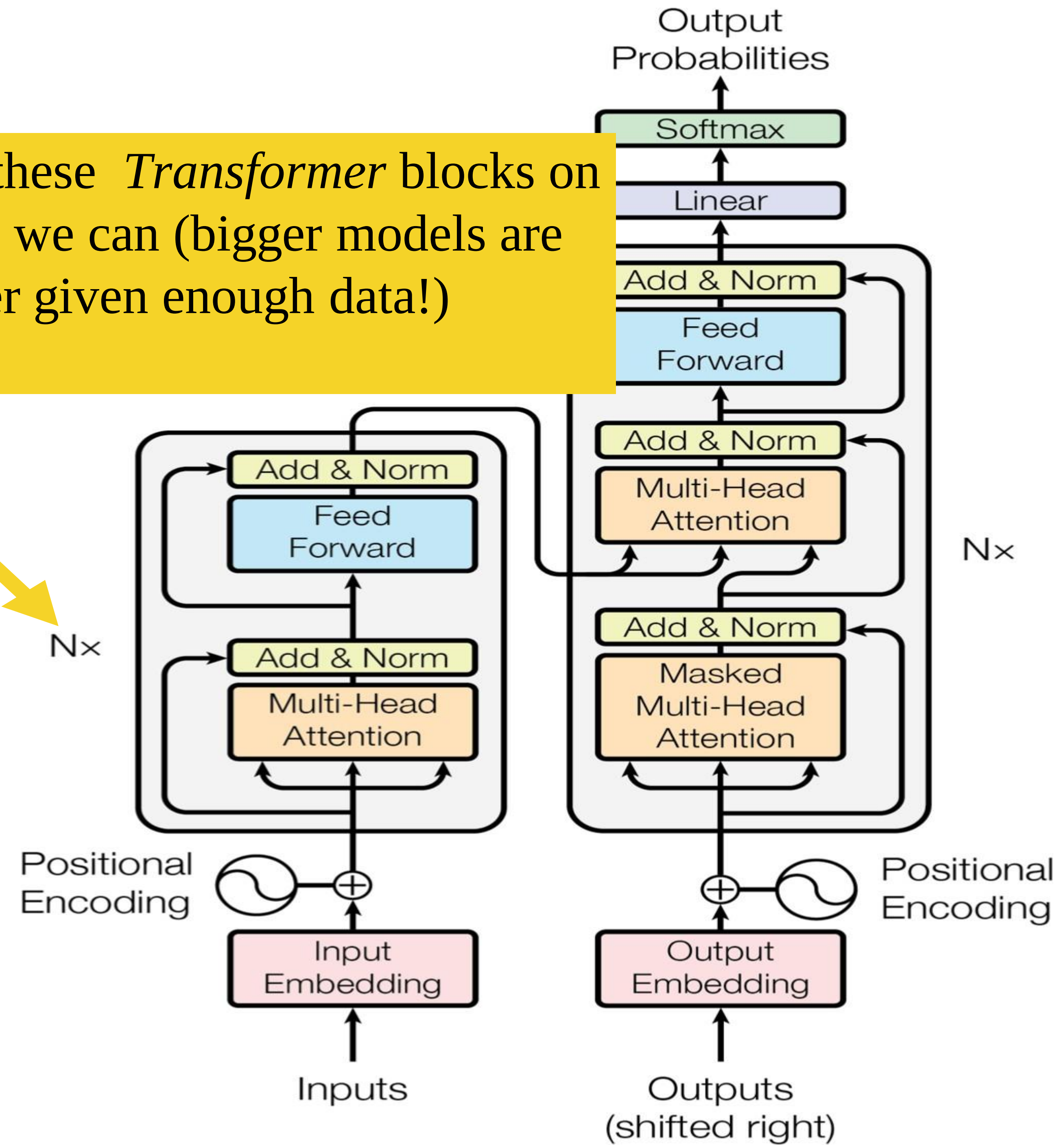
Residual connections, which mean that we add the input to a particular block to its output, help improve gradient flow



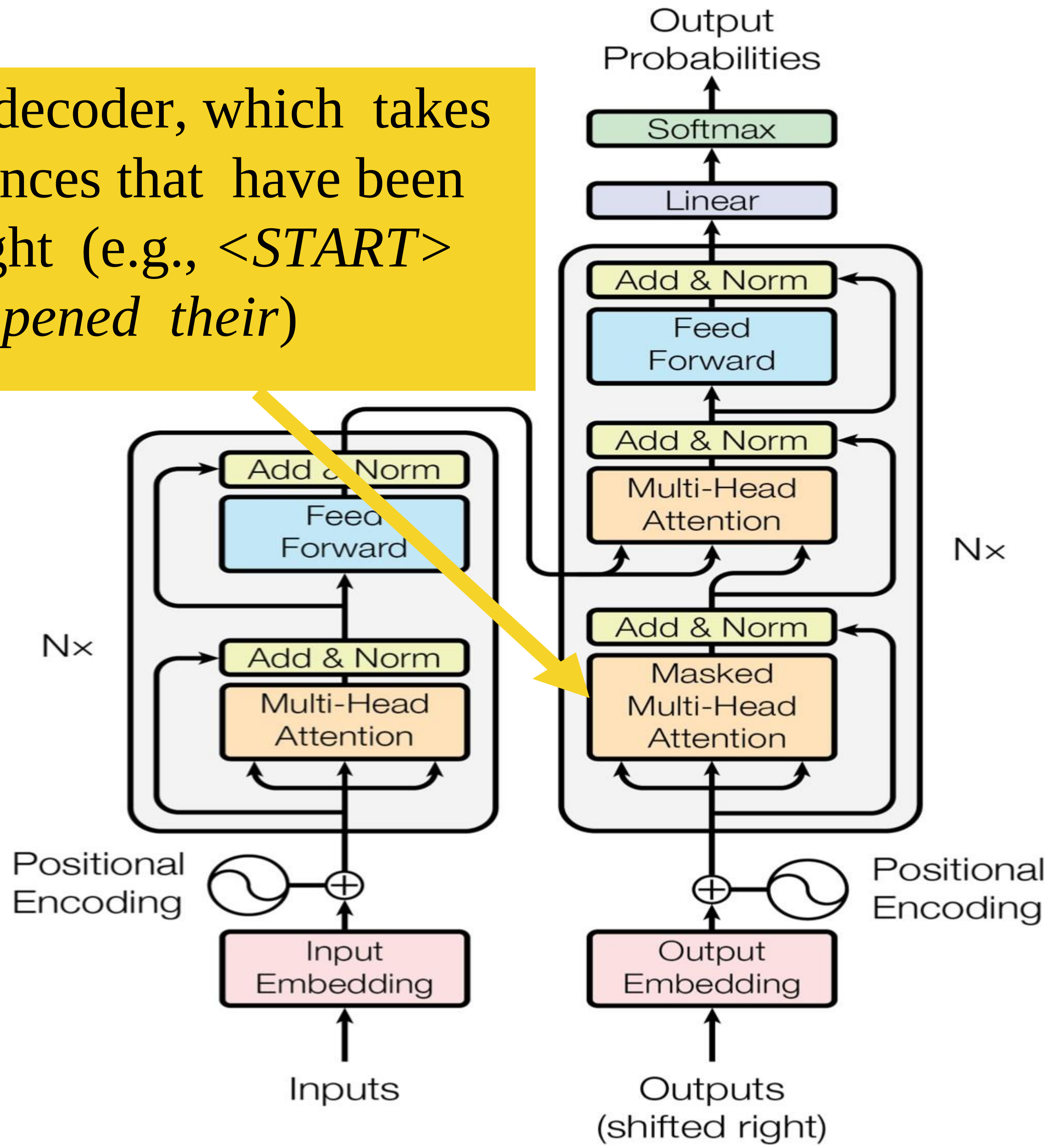
A feed-forward layer on top of the attention- weighted averaged value vectors allows us to add more parameters / nonlinearity



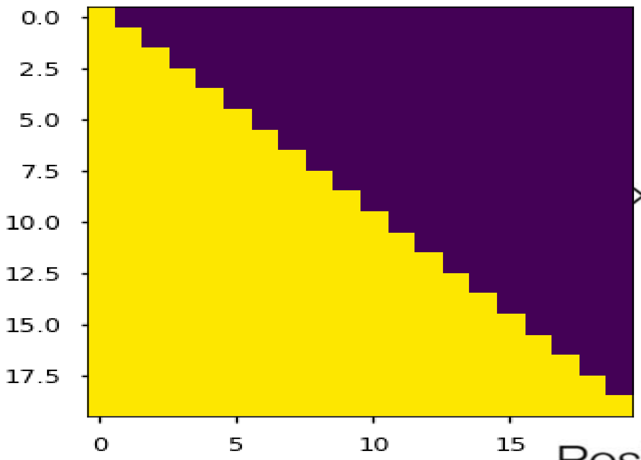
We stack as many of these *Transformer* blocks on top of each other as we can (bigger models are generally better given enough data!)

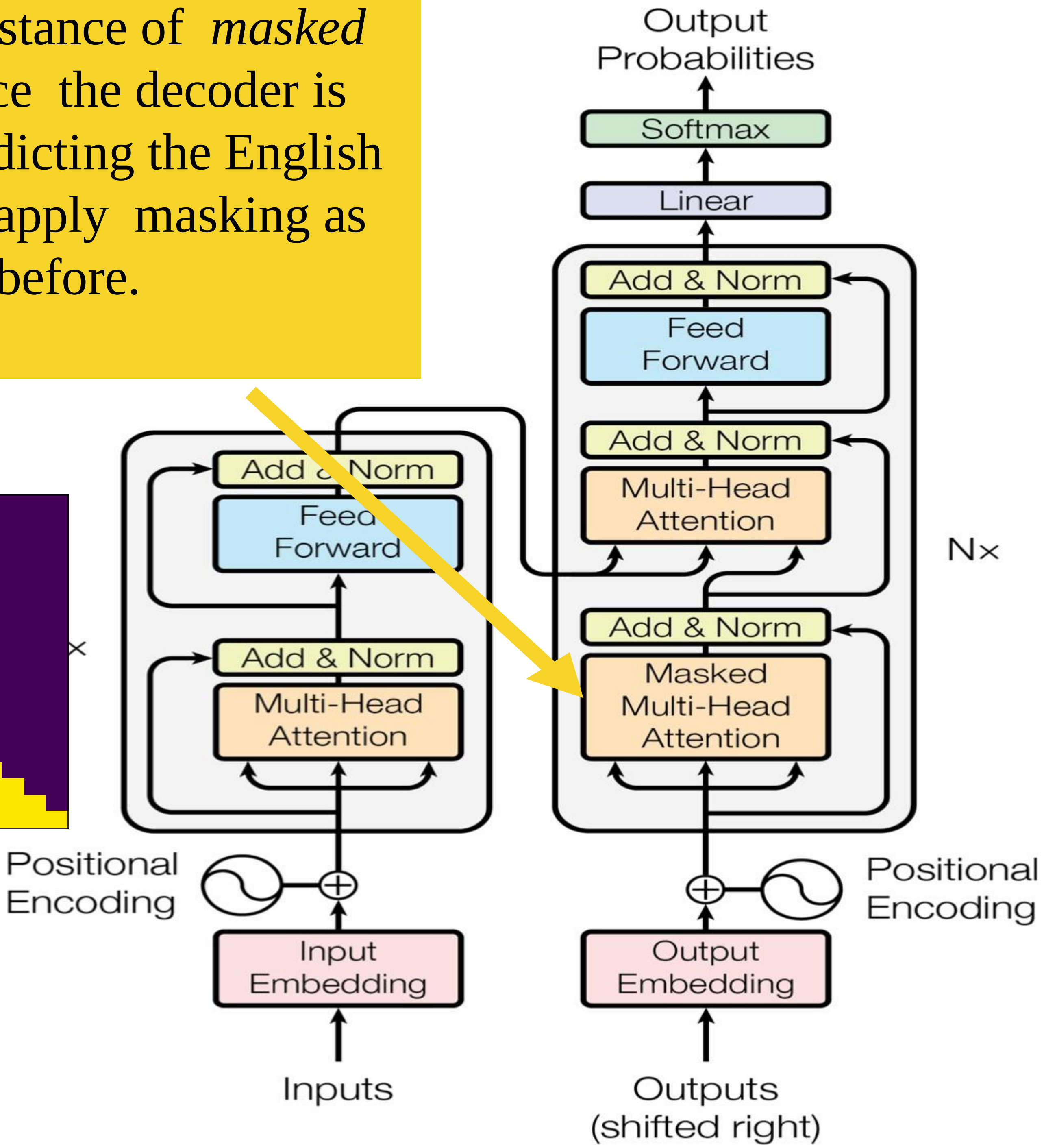


Moving onto the decoder, which takes in English sequences that have been shifted to the right (e.g., *<START> schools opened their*)



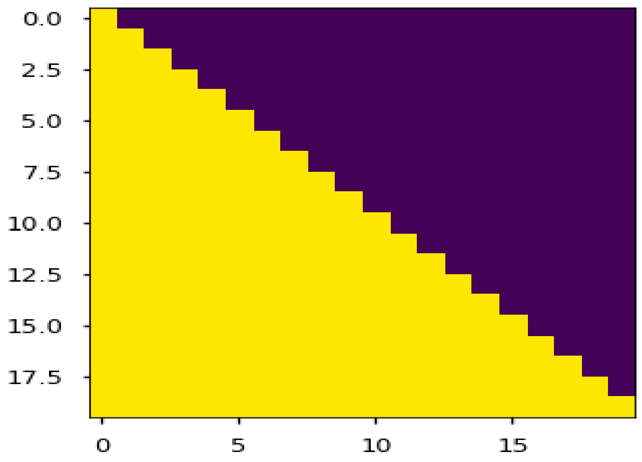
We first have an instance of *masked self attention*. Since the decoder is responsible for predicting the English words, we need to apply masking as we saw before.

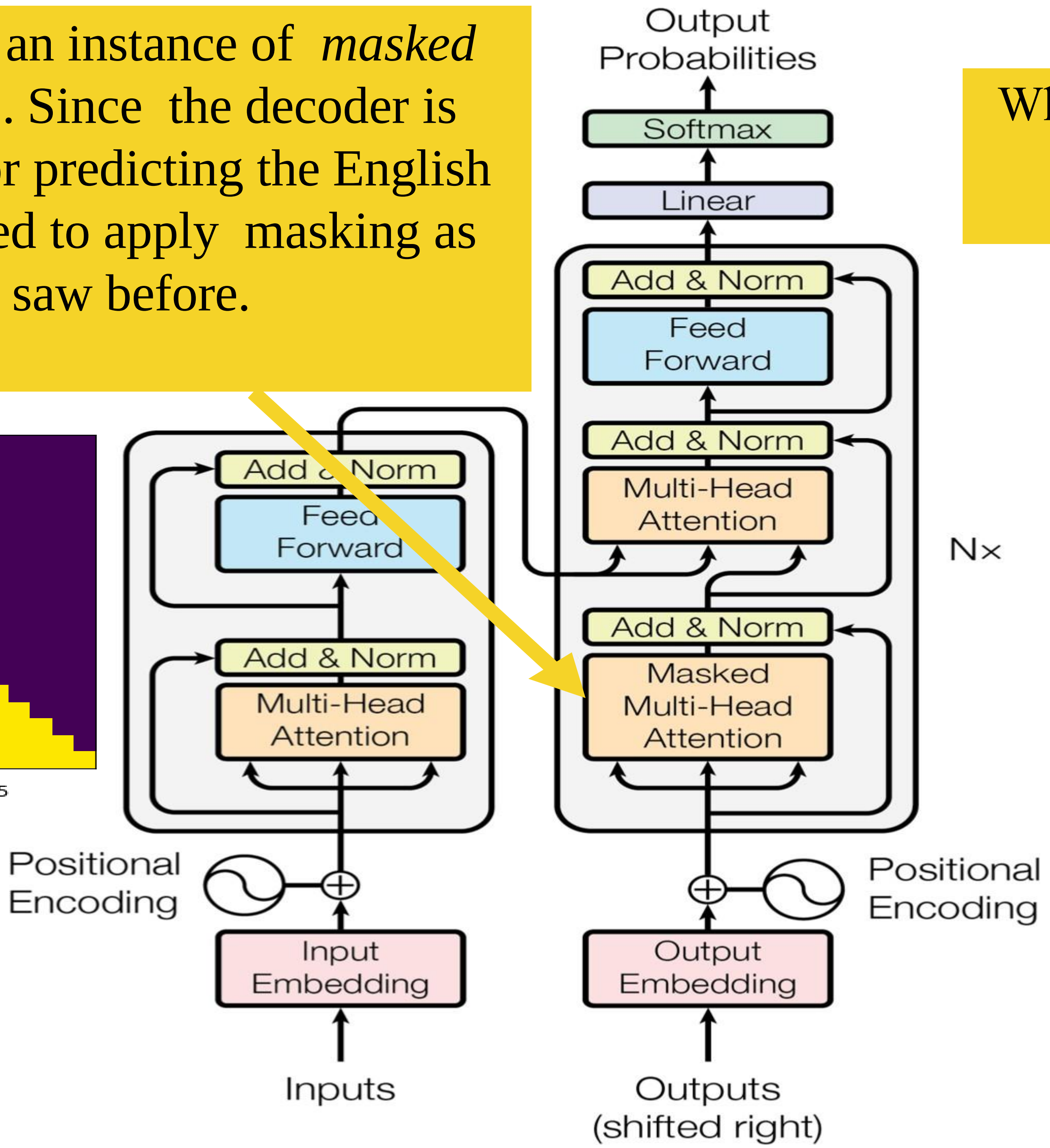




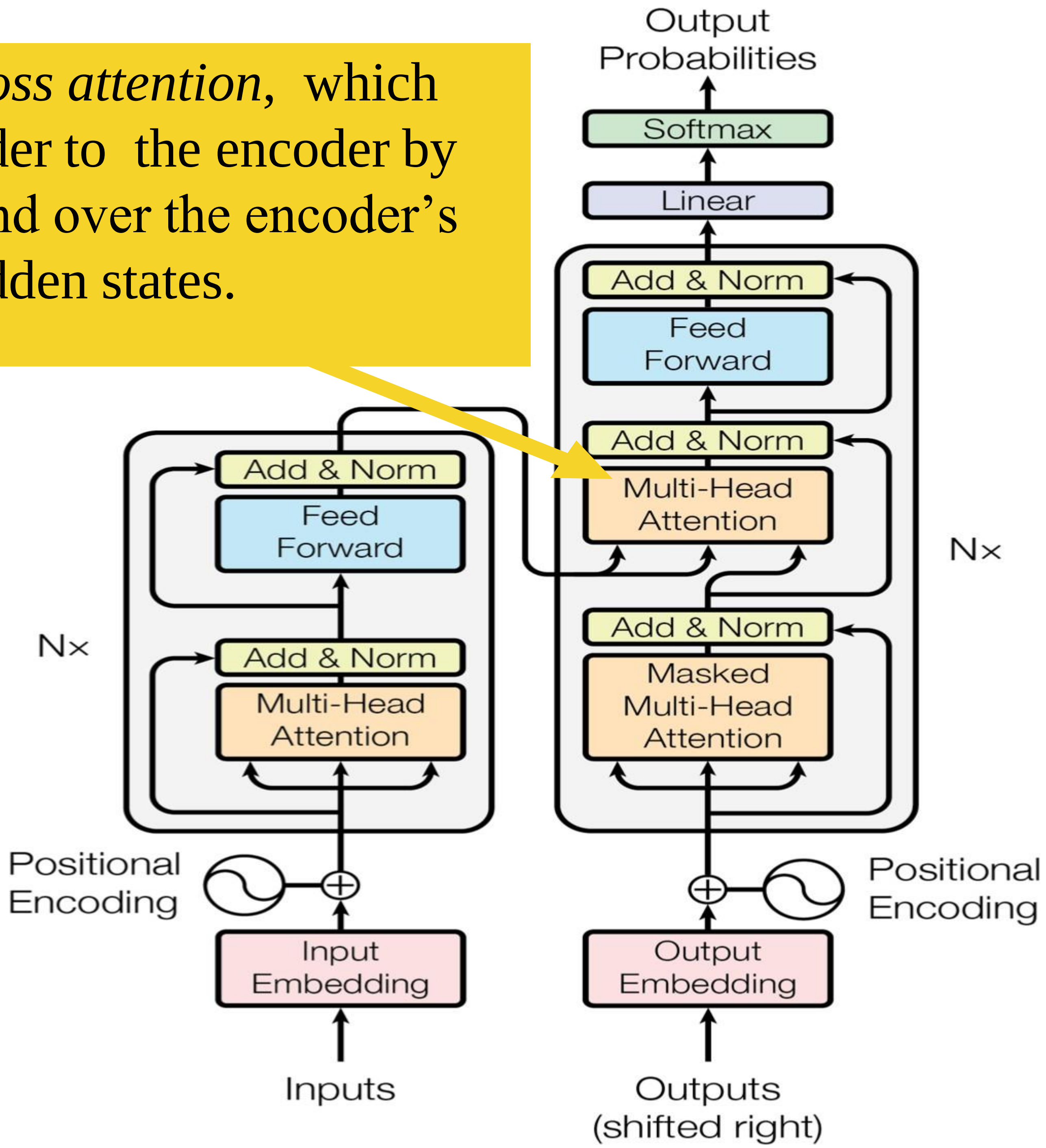
We first have an instance of *masked self attention*. Since the decoder is responsible for predicting the English words, we need to apply masking as we saw before.

Why don't we do masked self-attention in the encoder?

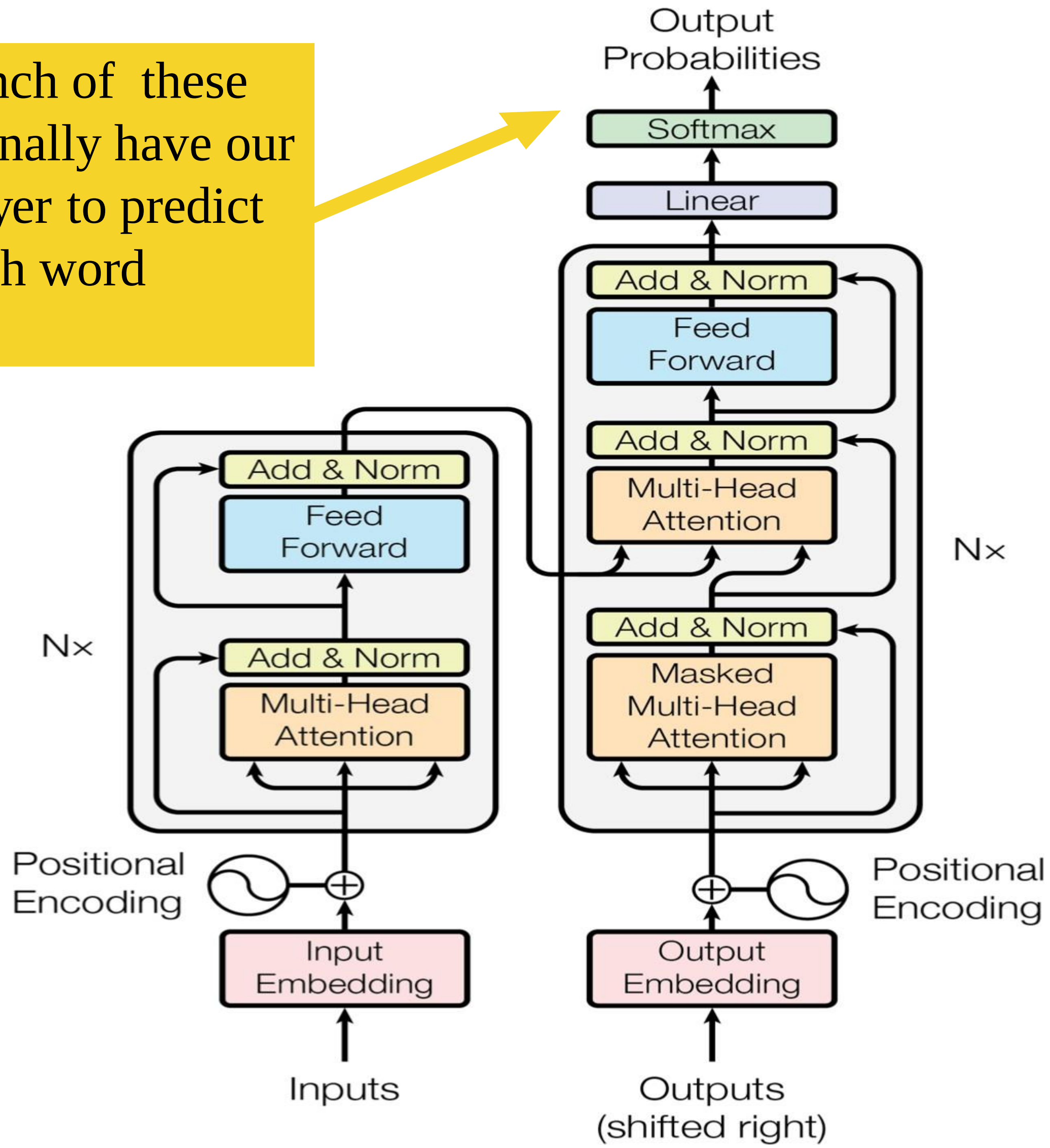




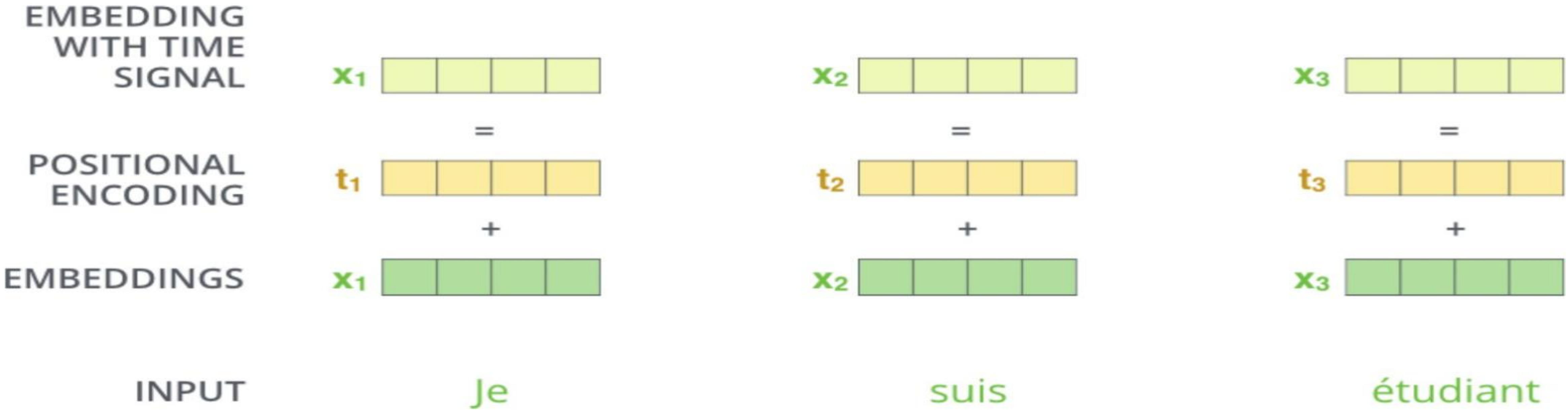
Now, we have *cross attention*, which connects the decoder to the encoder by enabling it to attend over the encoder's final hidden states.



After stacking a bunch of these decoder blocks, we finally have our familiar Softmax layer to predict the next English word



Positional encoding



Transformer positional encoding (pre-defined)

without need of training

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{model}}}\right)$$

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

Positional encoding is a 512d vector

i = a particular dimension of this vector

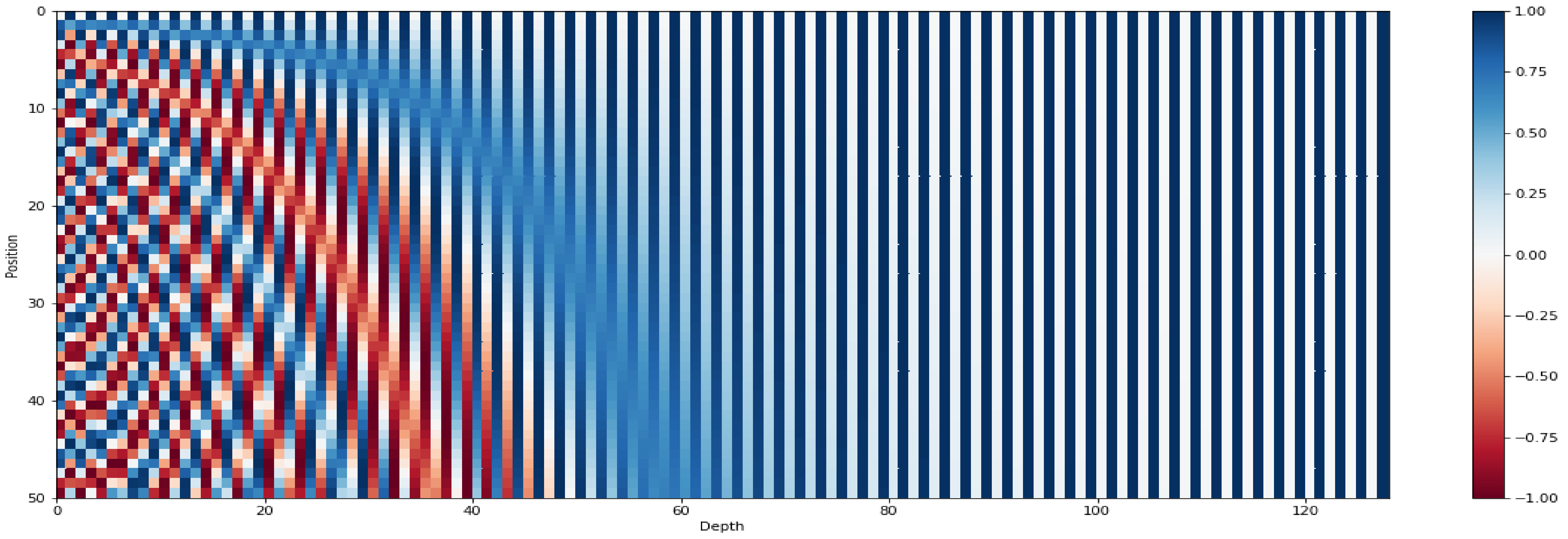
pos = dimension of the word

d_model = 512

One could also user absolute/relative position embedding that is trainable

What does this look like?

(each row is the pos. emb. of a 50-word sentence)



https://kazemnejad.com/blog/transformer_architecture_positional_encoding/

Intuitive explanation

0 :	0	0	0	0	8 :	1	0	0	0
1 :	0	0	0	1	9 :	1	0	0	1
2 :	0	0	1	0	10 :	1	0	1	0
3 :	0	0	1	1	11 :	1	0	1	1
4 :	0	1	0	0	12 :	1	1	0	0
5 :	0	1	0	1	13 :	1	1	0	1
6 :	0	1	1	0	14 :	1	1	1	0
7 :	0	1	1	1	15 :	1	1	1	1

https://kazemnejad.com/blog/transformer_architecture_positional_encoding/

On position embedding in BERT

Table 3: Experiments on GLUE. The evaluation metrics are following the official GLUE benchmark (Wang et al., 2018). The best performance of each task is bold.

PEs	single sentence CoLA acc	SST-2 acc	sentence pair MNLI acc	MRPC F1	QNLI acc	QQP F1	RTE acc	STS-B spear. cor.	WNLI acc	mean ± std
BERT without PE	39.0	86.5	80.1	86.2	83.7	86.5	63.0	87.4	33.8	76.6 ± 0.41
fully learnable (BERT-style) APE	60.2	**93.0**	84.8	89.4	88.7	87.8	**65.1**	**88.6**	37.5	82.2 ± 0.30
fixed sin. APE	57.1	92.6	84.3	89.0	88.1	87.5	58.4	86.9	45.1	80.5 ± 0.71
learnable sin. APE	56.0	92.8	84.8	88.7	88.5	87.7	59.1	87.0	40.8	80.6 ± 0.29
fully-learnable RPE	58.9	92.6	84.9	**90.5**	88.9	**88.1**	60.8	88.6	50.4	81.7 ± 0.31
fixed sin. RPE	60.4	92.2	84.8	89.5	88.8	88.0	62.9	88.1	45.1	81.8 ± 0.53
learnable sin. RPE	60.3	92.6	**85.2**	90.3	**89.1**	**88.1**	63.5	88.3	49.9	82.2 ± 0.40
fully learnable APE + fully-learnable RPE	59.8	92.8	85.1	89.6	88.6	87.8	62.5	88.3	**51.5**	81.8 ± 0.17
fully learnable APE + fixed sin. RPE	59.2	92.4	84.8	89.9	88.8	87.9	61.0	88.3	48.2	81.5 ± 0.20
fully learnable APE+ learnable sin. RPE	**61.1**	92.8	**85.2**	**90.5**	89.5	87.9	**65.1**	88.2	49.6	**82.5** ± 0.44
learnable sin. APE + fully-learnable RPE	57.2	92.7	84.8	88.9	88.5	87.8	58.6	88.0	51.3	80.8 ± 0.44
learnable sin. APE + fixed sin. RPE	57.6	92.6	84.5	88.8	88.6	87.6	63.1	87.4	48.7	81.3 ± 0.43
learnable sin. APE + learnable sin. RPE	57.7	92.7	85.0	89.6	88.7	87.8	62.3	87.5	50.1	81.4 ± 0.33

Benyou Wang, Lifeng Shang, Christina Lioma, Xin Jiang, Hao Yang, Qun Liu, Jakob Grue Simonsen. On Position Embeddings in BERT. ICLR 2021.

Modern embedding

Rotary embedding

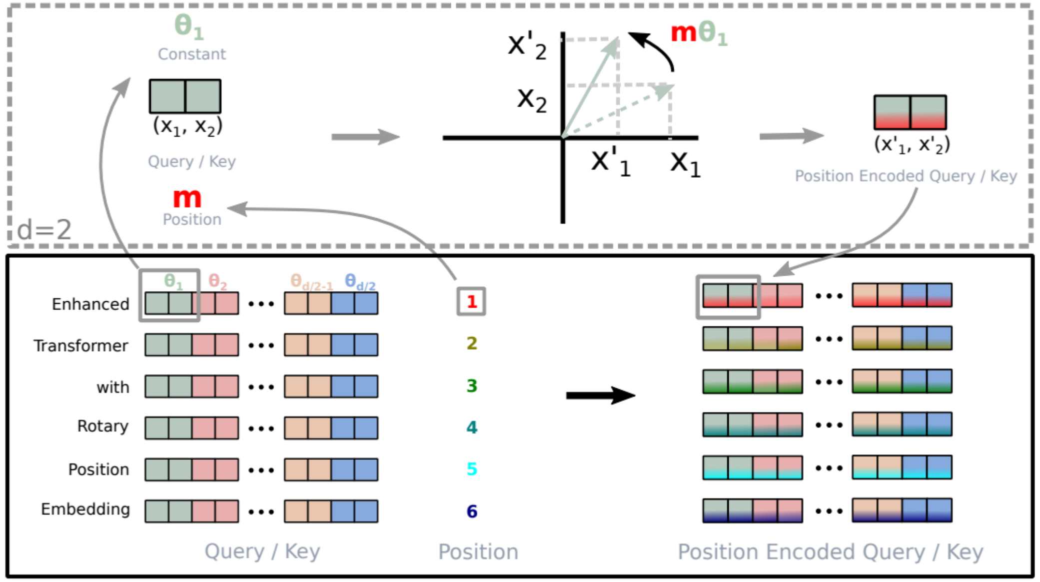


Figure 1: Implementation of Rotary Position Embedding(RoPE).

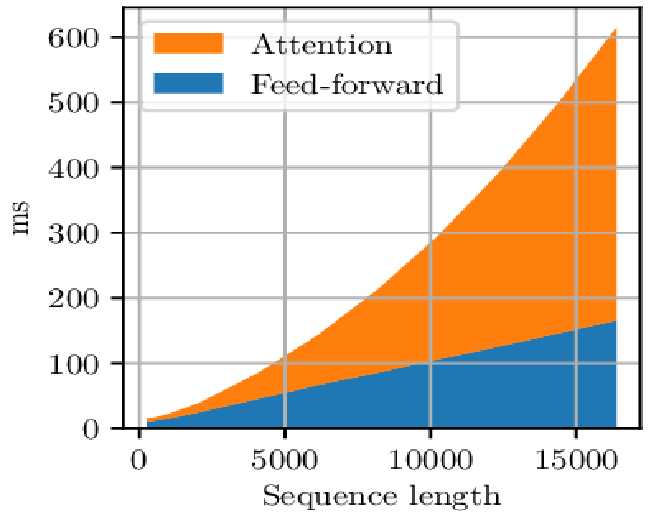
Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, Yunfeng Liu. RoFormer: Enhanced Transformer with Rotary Position Embedding. https://arxiv.org/abs/2104.09864
Benyou Wang, Donghao Zhao, Christina Lioma, Qiuchi Li, Peng Zhang, Jakob Grue Simonsen. Encoding word order in complex embeddings. https://arxiv.org/abs/1912.12333

More on new-Transformer

What would we like to fix about the Transformer?

Quadratic compute in self-attention (today):

- Computing all pairs of interactions means our computation grows quadratically with the sequence length!
- For recurrent models, it only grew linearly!



Quadratic computation as a function of sequence length

- One of the benefits of self-attention over recurrence was that it's highly parallelizable.
- However, its total number of operations grows as $O(n^2d)$, where n is the sequence length, and d is the dimensionality.

$$XQ \quad K^\top X^\top = XQK^\top X^\top \in \mathbb{R}^{n\times n}$$

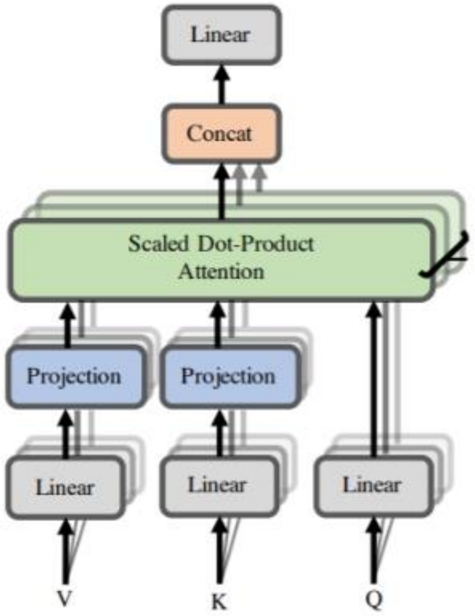
Need to compute all pairs of interactions!
$O(n^2d)$

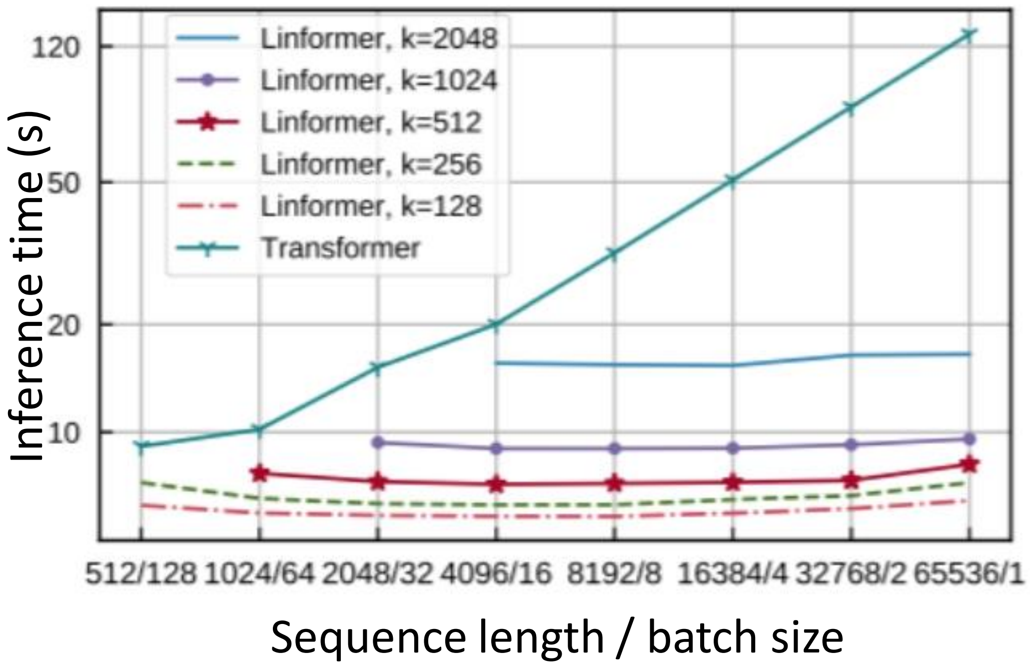
- Think of d as around $\mathbf{1,000}$ (though for large language models it's much larger!).
  - So, for a single (shortish) sentence, $n \leq 30; n^2 \leq \mathbf{900}$.
  - In practice, we set a bound like $n = 512$.
  - **But what if we'd like $\boldsymbol{n} \geq \mathbf{50,000}$?** For example, to work on long documents?

Work on improving on quadratic self-attention cost

Considerable recent work has gone into the question, *Can we build models like Transformers without paying the all-pairs self-attention cost?*
For example, **Linformer** [Wang et al., 2020]

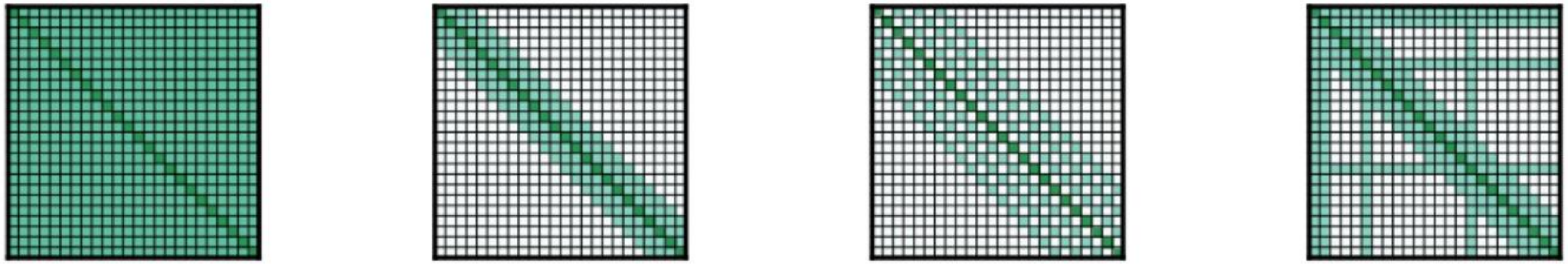
Key Idea:
- Linformer introduces a novel concept called "compressed" or "linearized" self-attention.
- Instead of computing attention scores for all pairs of input elements, it employs linear projections to reduce the complexity.





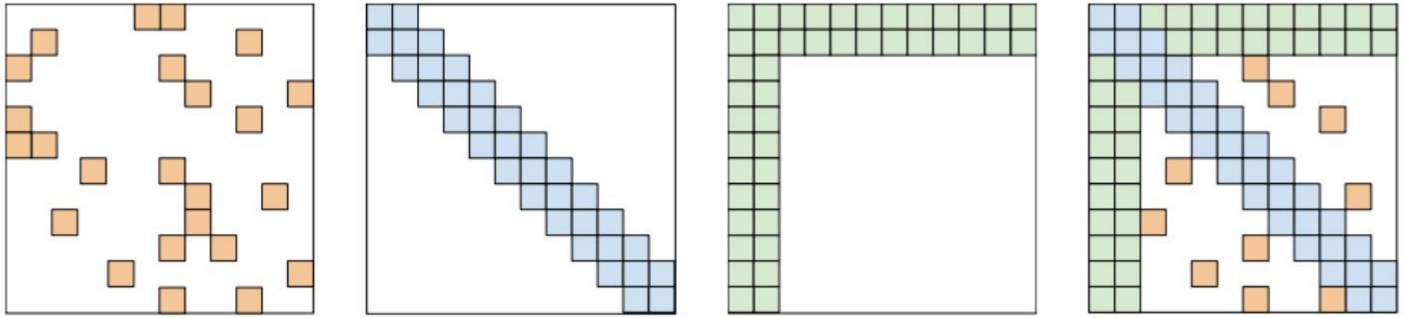
Example: Longformer / Big Bird

Key idea: use sparse attention patterns!



(a) Full n^2 attention (b) Sliding window attention (c) Dilated sliding window (d) Global+sliding window

(Beltagy et al., 2020): Longformer: The Long-Document Transformer



(a) Random attention (b) Window attention (c) Global Attention (d) BIGBIRD

(Zaheer et al., 2021): Big Bird: Transformers for Longer

Do we even need to remove the quadratic cost of attention?

- **As Transformers Scale Up:** When Transformers are scaled to larger sizes, an increasingly significant portion of computational resources is allocated to tasks outside of the self-attention mechanism, despite its quadratic computational cost.
- **Current Practice:** In practice, nearly all large Transformer-based language models continue to rely on the traditional quadratic-cost attention mechanism that has been presented.
- **Challenges with Cost-Efficiency:** Alternative, more computationally efficient methods often do not perform as effectively when applied at a large scale.
- **Exploring Cheaper Alternatives:** Is there value in exploring cost-efficient alternatives to self-attention, or could we unlock the potential for significantly improved models with much longer contextual information (e.g., >100k tokens) if we find the right approach?

Do Transformer Modifications Transfer?

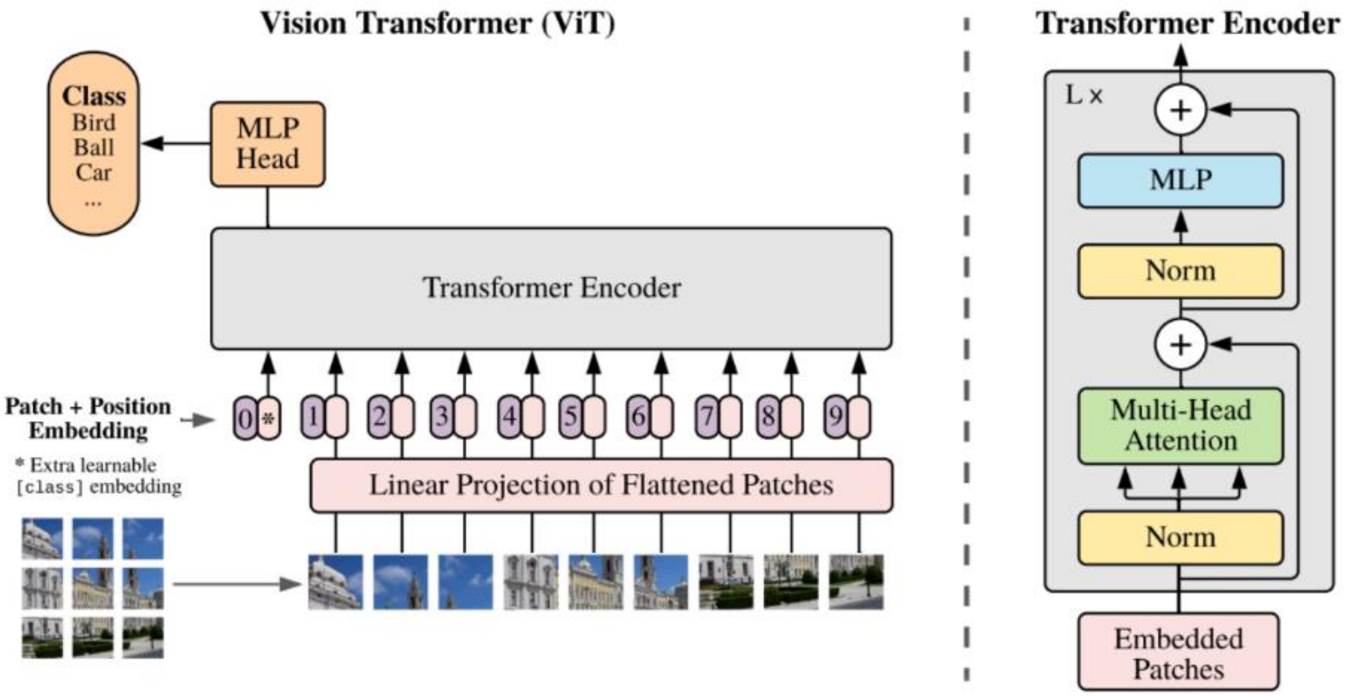
- "Surprisingly, we find that most modifications do not meaningfully improve performance."

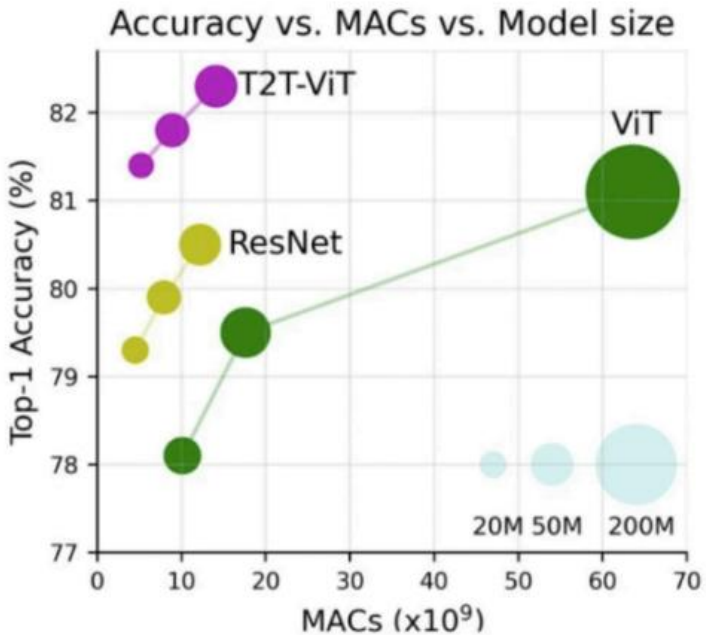
Model	Params	Ops	Step/s	Early loss	Final loss	SGLUE	XSum	WebQ	WMT EnDe
Vanilla Transformer	223M	11.1T	3.50	2.182 ± 0.005	1.838	71.66	17.78	23.02	26.62
GeLU	223M	11.1T	3.58	2.179 ± 0.003	1.838	**75.79**	**17.86**	**25.13**	26.47
Swish	223M	11.1T	3.62	2.186 ± 0.003	1.847	**73.77**	17.74	**24.34**	**26.75**
ELU	223M	11.1T	3.56	2.270 ± 0.007	1.932	67.83	16.73	23.02	26.08
GLU	223M	11.1T	3.59	2.174 ± 0.003	**1.814**	**74.20**	**17.42**	24.34	**27.12**
GeGLU	223M	11.1T	3.55	2.130 ± 0.006	**1.792**	**75.96**	**18.27**	**24.87**	**26.87**
ReGLU	223M	11.1T	3.57	2.145 ± 0.004	**1.803**	**76.17**	**18.36**	**24.87**	**27.02**
SeLU	223M	11.1T	3.55	2.315 ± 0.004	1.948	68.76	16.76	22.75	25.99
SwiGLU	223M	11.1T	3.53	2.127 ± 0.003	**1.789**	**76.00**	**18.20**	**24.34**	**27.02**
LiGLU	223M	11.1T	3.59	2.149 ± 0.005	**1.798**	**75.34**	**17.97**	**24.34**	26.53
Sigmoid	223M	11.1T	3.63	2.291 ± 0.019	1.867	**74.31**	17.51	23.02	26.30
Softplus	223M	11.1T	3.47	2.207 ± 0.011	1.850	**72.45**	17.65	**24.34**	**26.89**
RMS Norm	223M	11.1T	3.68	2.167 ± 0.008	**1.821**	**75.45**	**17.94**	**24.07**	**27.14**
Rezero	223M	11.1T	3.51	2.262 ± 0.003	1.939	61.69	15.64	20.90	26.37
Rezero + LayerNorm	223M	11.1T	3.26	2.223 ± 0.006	1.858	70.42	17.58	23.02	26.29
Rezero + RMS Norm	223M	11.1T	3.34	2.221 ± 0.009	1.875	70.33	17.32	23.02	26.19
Fixup	223M	11.1T	2.95	2.382 ± 0.012	2.067	58.56	14.42	23.02	26.31
24 layers, $d_{ff} = 1536$, $H = 6$	224M	11.1T	3.33	2.200 ± 0.007	1.843	**74.89**	17.75	**25.13**	**26.89**
18 layers, $d_{ff} = 2048$, $H = 8$	223M	11.1T	3.38	2.185 ± 0.005	**1.831**	**76.45**	16.83	**24.34**	**27.10**
8 layers, $d_{ff} = 4608$, $H = 18$	223M	11.1T	3.69	2.190 ± 0.005	1.847	**74.58**	17.69	**23.28**	**26.85**
6 layers, $d_{ff} = 6144$, $H = 24$	223M	11.1T	3.70	2.201 ± 0.010	1.857	**73.55**	17.59	**24.60**	**26.66**
Block sharing	65M	11.1T	3.91	2.497 ± 0.037	2.164	64.50	14.53	21.96	25.48
+ Factorized embeddings	45M	9.4T	4.21	2.631 ± 0.305	2.183	60.84	14.00	19.84	25.27
+ Factorized & shared embeddings	20M	9.1T	4.37	2.907 ± 0.313	2.385	53.95	11.37	19.84	25.19
Encoder only block sharing	170M	11.1T	3.68	2.298 ± 0.023	1.929	69.60	16.23	23.02	26.23
Decoder only block sharing	144M	11.1T	3.70	2.352 ± 0.029	2.082	67.93	16.13	**23.81**	26.08
Factorized Embedding	227M	9.4T	3.80	2.208 ± 0.006	1.855	70.41	15.92	22.75	26.50
Factorized & shared embeddings	202M	9.1T	3.92	2.320 ± 0.010	1.952	68.69	16.33	22.22	26.44
Tied encoder/decoder input embeddings	248M	11.1T	3.55	2.192 ± 0.002	1.840	**71.70**	17.72	**24.34**	26.49
Tied decoder input and output embeddings	248M	11.1T	3.57	2.187 ± 0.007	**1.827**	**74.86**	17.74	**24.87**	**26.67**
Untied embeddings	273M	11.1T	3.53	2.195 ± 0.005	**1.834**	**72.99**	17.58	**23.28**	26.48
Adaptive input embeddings	204M	9.2T	3.55	2.250 ± 0.002	1.899	66.57	16.21	**24.07**	**26.66**
Adaptive softmax	204M	9.2T	3.60	2.364 ± 0.005	1.982	**72.91**	16.67	21.16	25.56
Adaptive softmax without projection	223M	10.8T	3.43	2.229 ± 0.009	1.914	**71.82**	17.10	23.02	25.72
Mixture of softmaxes	232M	16.3T	2.24	2.227 ± 0.017	**1.821**	**76.77**	17.62	22.75	**26.82**
Transparent attention	223M	11.1T	3.33	2.181 ± 0.014	1.874	54.31	10.40	21.16	**26.80**
Dynamic convolution	257M	11.8T	2.65	2.403 ± 0.009	2.047	58.30	12.67	21.16	17.03
Lightweight convolution	224M	10.4T	4.07	2.370 ± 0.010	1.989	63.07	14.86	23.02	24.73
Evolved Transformer	217M	9.9T	3.09	2.220 ± 0.003	1.863	**73.67**	10.76	**24.07**	26.58
Synthesizer (dense)	224M	11.4T	3.47	2.334 ± 0.021	1.962	61.03	14.27	16.14	**26.63**
Synthesizer (dense plus)	243M	12.6T	3.22	2.191 ± 0.010	1.840	**73.98**	16.96	**23.81**	**26.71**
Synthesizer (dense plus alpha)	243M	12.6T	3.01	2.180 ± 0.007	**1.828**	**74.25**	17.02	**23.28**	26.61
Synthesizer (factorized)	207M	10.1T	3.94	2.341 ± 0.017	1.968	62.78	15.39	**23.55**	26.42
Synthesizer (random)	254M	10.1T	4.08	2.326 ± 0.012	2.009	54.27	10.35	19.56	26.44
Synthesizer (random plus)	292M	12.0T	3.63	2.189 ± 0.004	1.842	**73.32**	17.04	**24.87**	26.43
Synthesizer (random plus alpha)	292M	12.0T	3.42	2.186 ± 0.007	**1.828**	**75.24**	17.08	**24.08**	26.39
Universal Transformer	84M	40.0T	0.88	2.406 ± 0.036	2.053	70.13	14.09	19.05	23.91
Mixture of experts	648M	11.7T	3.20	2.148 ± 0.006	**1.785**	**74.55**	**18.13**	**24.08**	**26.94**
Switch Transformer	1100M	11.7T	3.18	2.135 ± 0.007	**1.758**	**75.38**	**18.02**	**26.19**	**26.81**
Funnel Transformer	223M	1.9T	4.30	2.288 ± 0.008	1.918	67.34	16.26	22.75	23.20
Weighted Transformer	280M	71.0T	0.59	2.378 ± 0.021	1.989	69.04	16.98	23.02	26.30
Product key memory	421M	386.6T	0.25	2.155 ± 0.003	**1.798**	**75.16**	17.04	**23.55**	**26.73**

Do Transformer Modifications Transfer Across Implementations and Applications?

Sharan Narang* Hyung Won Chung Yi Tay William Fedus
Thibault Fevry† Michael Matena† Karishma Malkan† Noah Fiedel
Noam Shazeer Zhenzhong Lan† Yanqi Zhou Wei Li
Nan Ding Jake Marcus Adam Roberts Colin Raffel†

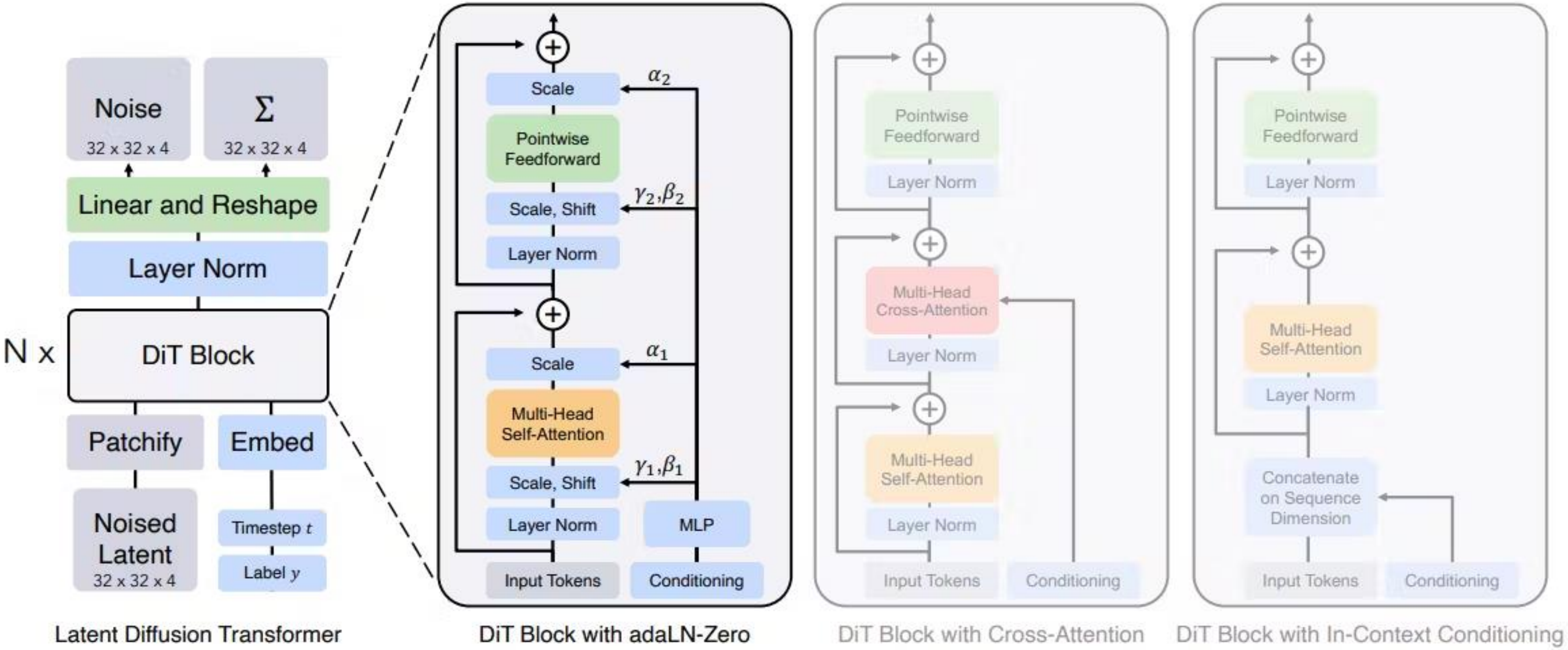
Vision Transformer (ViT)





(Dosovitskiy et al., 2021): An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale

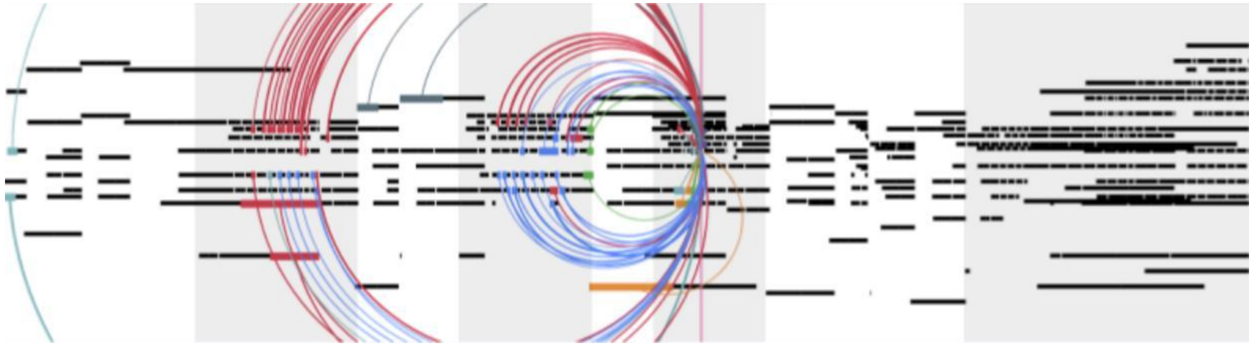
Diffusion Transformer (DiT)



(William Peebles et al., 2022): Scalable Diffusion Models with Transformers.

DiT aims to improve the performance of diffusion models by replacing the commonly used U-Net backbone with a transformer.

Music Transformer



https://magenta.tensorflow.org/music-transformer

(Huang et al., 2018): Music Transformer: Generating Music with Long-Term Structure

Why transformer

Why Pretraining + Transformers

- 1.Because transformers are more efficient?

Transformers are shower comparing to LSTM with same amount parameters

Credits from 杨植麟,
Recurrent AI

Why Pretraining + Transformers

- 1.Because transformers are more efficient?

Transformers are shower comparing to LSTM with same amount parameters

- 2. Because transformers are better on machine translation?

RNNs and CNNs are equally good in machine translations

Why Pretraining + Transformers

- 1.Because transformers are more efficient?

Transformers are shower comparing to LSTM with same amount parameters

- 2. Because transformers are better on machine translation?

RNNs and CNNs are equally good in machine translations

- 3. Because transformers use nothing but attention?

So what?

Why Pretraining + Transformers

- 1.Because transformers are more efficient?

Transformers are shower comparing to LSTM with same amount parameters

- 2. Because transformers are better on machine translation?

RNNs and CNNs are equally good in machine translations

- 3. Because transformers use nothing but attention?

So what?

- 4. Because transformers learns contextualised word embeddings?

<u>RNN also can learn contextualised word embeddings</u>

Why Pretraining + Transformers

- **Capacity**: The model has sufficient expressive capabilities
- **Optimization**: Can optimize and obtain better solutions in a huge expression space
- **Generalization**: Better solutions can generalize on test data

"Exploring the Limits of Language Modeling
Jozefowicz et al 2016

LSTM-8192-1024, 1.8 billion params, ppl 30.6
LSTM-8192-2048, 3.3 billion params, ppl 32.2

Dai, Yang et al 2016
Transformer-XL Base, 0.46 billion params, ppl 23.5
Transformer-XL Large, 0.8 billion params, ppl 21.8

ppl=perplexity, the lower the better

Scalability: Transformers scale much better with more parameters

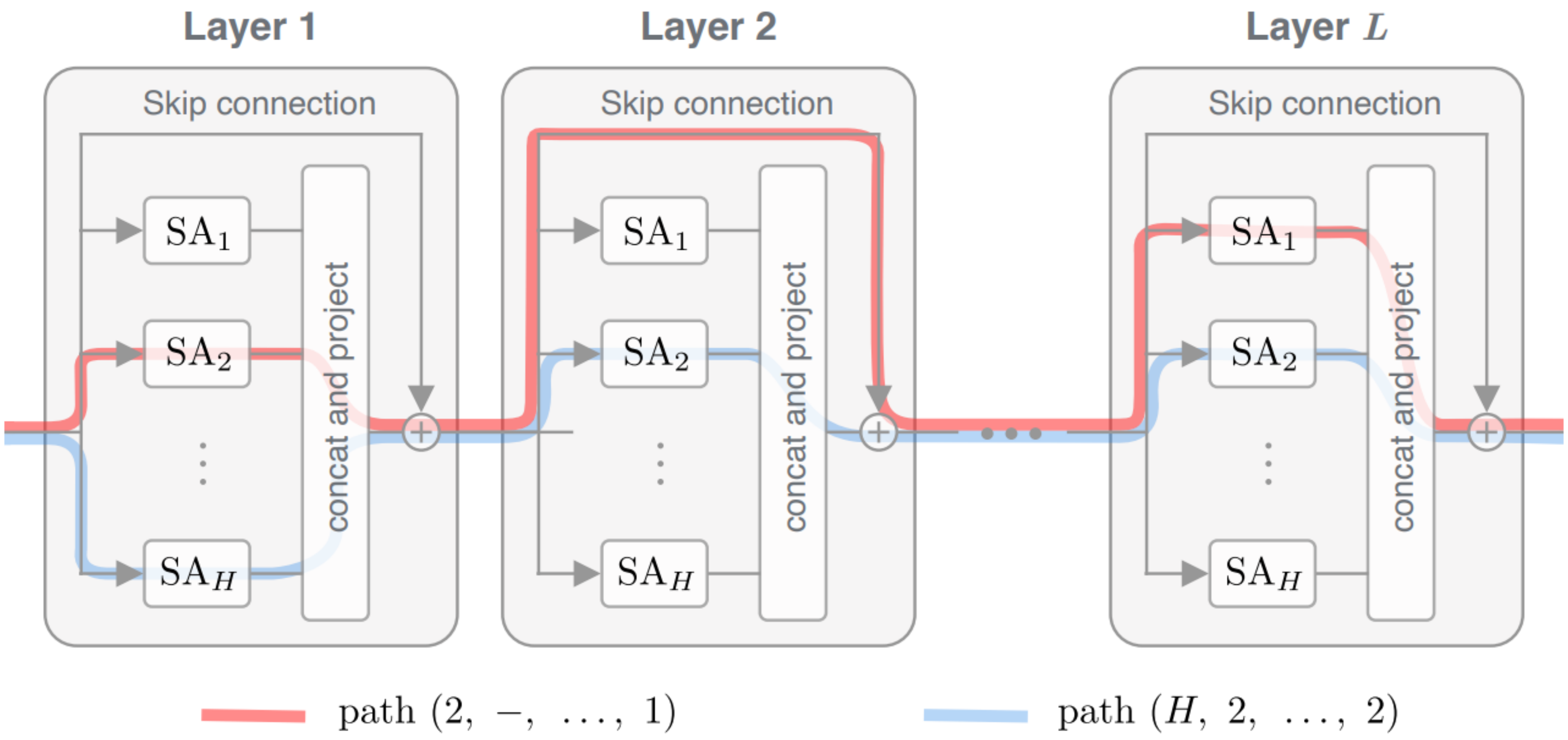
Deep understanding of transformer

An Ablation Study

What if

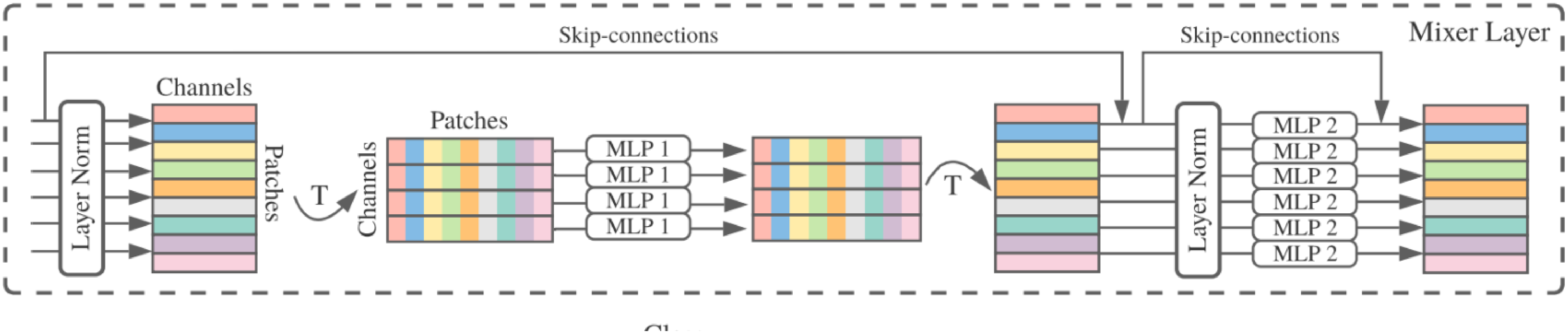
- ✓ removing SAN
- ✓ removing FFN
- ✓ removing PE
- ✓ and many others?

Without FFN, pure SAN



Y Dong, JB Cordonnier, A Loukas. Attention is not all you need: Pure attention loses rank doubly exponentially with depth. https://browse.arxiv.org/pdf/2103.03404.pdf

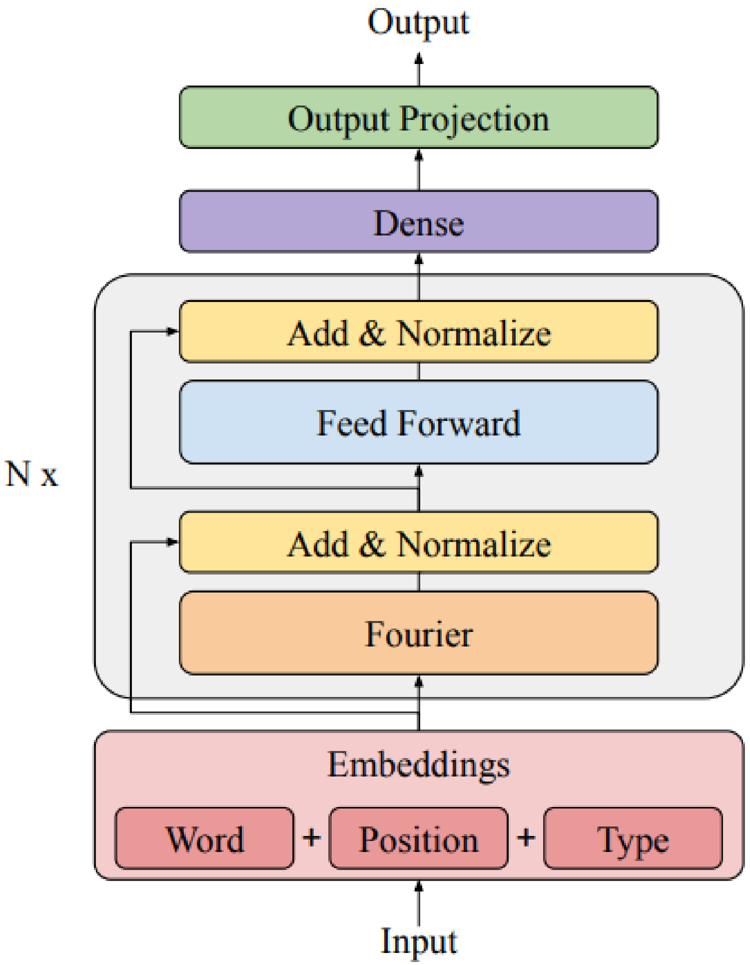
Without SAN, pure FNN



At least it works for computer vision.

Ilya Tolstikhin et.al MLP-Mixer: An all-MLP Architecture for Vision https://browse.arxiv.org/pdf/2105.01601.pdf

Replace SAN with fourier



- ❖ Highlight the potential of linear units as a drop-in replacement for the attention mechanism in text classification tasks.
- ❖ FNet will be effective as a lightweight

James Lee-Thorp, Joshua Ainslie, Ilya Eckstein, Santiago Ontanon .
FNet: Mixing Tokens with Fourier Transforms. NAACL 2022

How to place FFN and SAN?

sfsfsfsfsfsfsfsfsfsfsfsfsfsf

(a) Interleaved Transformer

sssssssfsfsfsfsfsfsfsfffffff

(b) Sandwich Transformer

Figure 1: A transformer model (a) is composed of interleaved self-attention (green) and feedforward (purple) sublayers. Our sandwich transformer (b), a reordering of the transformer sublayers, performs better on language modeling. Input flows from left to right.

Model	PPL
fsfsfffsffsfsssffsfssfssssffsffs	20.74
sfssffsffffssssfsfffsfsffsfssssf	20.64
fsffssffssssffsssssffsfssfsfffff	20.33
fsffffffsssfssffsfssffsfsssffsss	20.27
fssffffffsfsssfffssssfffssssffss	19.98
sssfssfsffffssfsfsfsssffsfsfffsf	19.92
fffsfsssfsffsfsffsffsssssffssffs	19.69
fffsffssffsssfssfsssfffffsfsssfs	19.54
sfsfsfsfsfsfsfsfsfsfsfsfsfsfsfsf	**19.13**
fsffssfssfffssssfffsssffffsfssfs	19.08
sfsffssssffssffffsssffsssfsffsff	18.90
sfsfsfsfsfsfsfsfsfsfsfsfsfsfsfsf	**18.83**
sssssssffsffsfsfsffffsfffsfssffs	18.83
sffsfsffsfsssffssfssssssfffffffs	18.77
sssfssffsfssfsffsfffssffsfsffssf	18.68
fffsssssfffsfssssffsfsfsfssffsff	18.64
sfffsssfsfssfsssssfssfffffsfffsf	18.61
ssffssfssssffffffssffsssfsffssff	18.60
fsfsssssfsfsfffffsfffsffssffssss	18.55
sfsfsfsfsfsfsfsfsfsfsfsfsfsfsfsf	**18.54**
sfsfsfsfsfsfsfsfsfsfsfsfsfsfsfsf	**18.49**
fsfsssssfsfffssfsffsfsfsfsffffss	18.38
sfssffsfsfsffsssssfffsssfffsffsf	18.28
sfsfsfsfsfsfsfsfsfsfsfsfsfsfsfsf	**18.25**
sfsfssfsssffsfsfsfsffffssffsfssf	18.19

Ofir Press, Noah A. Smith, Omer Levy. Improving Transformer Models by Reordering their Sublayers.
https://browse.arxiv.org/pdf/1911.03864.pdf

What if position embedding is removed?

Table 3: Experiments on GLUE. The evaluation metrics are following the official GLUE benchmark (Wang et al., 2018). The best performance of each task is bold.

PEs	single sentence		sentence pair							
	CoLA acc	SST-2 acc	MNLI acc	MRPC F1	QNLI acc	QQP F1	RTE acc	STS-B spear. cor.	WNLI acc	mean $\pm$ std
BERT without PE	39.0	86.5	80.1	86.2	83.7	86.5	63.0	87.4	33.8	76.6 ± 0.41
fully learnable (BERT-style) APE	60.2	**93.0**	84.8	89.4	88.7	87.8	**65.1**	**88.6**	37.5	82.2 ± 0.30
fixed sin. APE	57.1	92.6	84.3	89.0	88.1	87.5	58.4	86.9	45.1	80.5 ± 0.71
learnable sin. APE	56.0	92.8	84.8	88.7	88.5	87.7	59.1	87.0	40.8	80.6 ± 0.29
fully-learnable RPE	58.9	92.6	84.9	**90.5**	88.9	**88.1**	60.8	88.6	50.4	81.7 ± 0.31
fixed sin. RPE	60.4	92.2	84.8	89.5	88.8	88.0	62.9	88.1	45.1	81.8 ± 0.53
learnable sin. RPE	60.3	92.6	**85.2**	90.3	**89.1**	**88.1**	63.5	88.3	49.9	82.2 ± 0.40
fully learnable APE + fully-learnable RPE	59.8	92.8	85.1	89.6	88.6	87.8	62.5	88.3	**51.5**	81.8 ± 0.17
fully learnable APE + fixed sin. RPE	59.2	92.4	84.8	89.9	88.8	87.9	61.0	88.3	48.2	81.5 ± 0.20
fully learnable APE+ learnable sin. RPE	**61.1**	92.8	**85.2**	**90.5**	89.5	87.9	**65.1**	88.2	49.6	$\mathbf{82.5} \pm 0.44$
learnable sin. APE + fully-learnable RPE	57.2	92.7	84.8	88.9	88.5	87.8	58.6	88.0	51.3	80.8 ± 0.44
learnable sin. APE + fixed sin. RPE	57.6	92.6	84.5	88.8	88.6	87.6	63.1	87.4	48.7	81.3 ± 0.43
learnable sin. APE + learnable sin. RPE	57.7	92.7	85.0	89.6	88.7	87.8	62.3	87.5	50.1	81.4 ± 0.33

Benyou Wang, Lifeng Shang, Christina Lioma, Xin Jiang, Hao Yang, Qun Liu, Jakob Grue Simonsen. On Position Embeddings in BERT. https://openreview.net/pdf?id=onxoVA9FxMw ICLR 2021.

Improvements for Norm

DeepNet - 1000 layer Transformers

A new normalization function (DEEPNORM) is introduced
[replacing it is not Layer Norm! Instead, modify it similarly to:

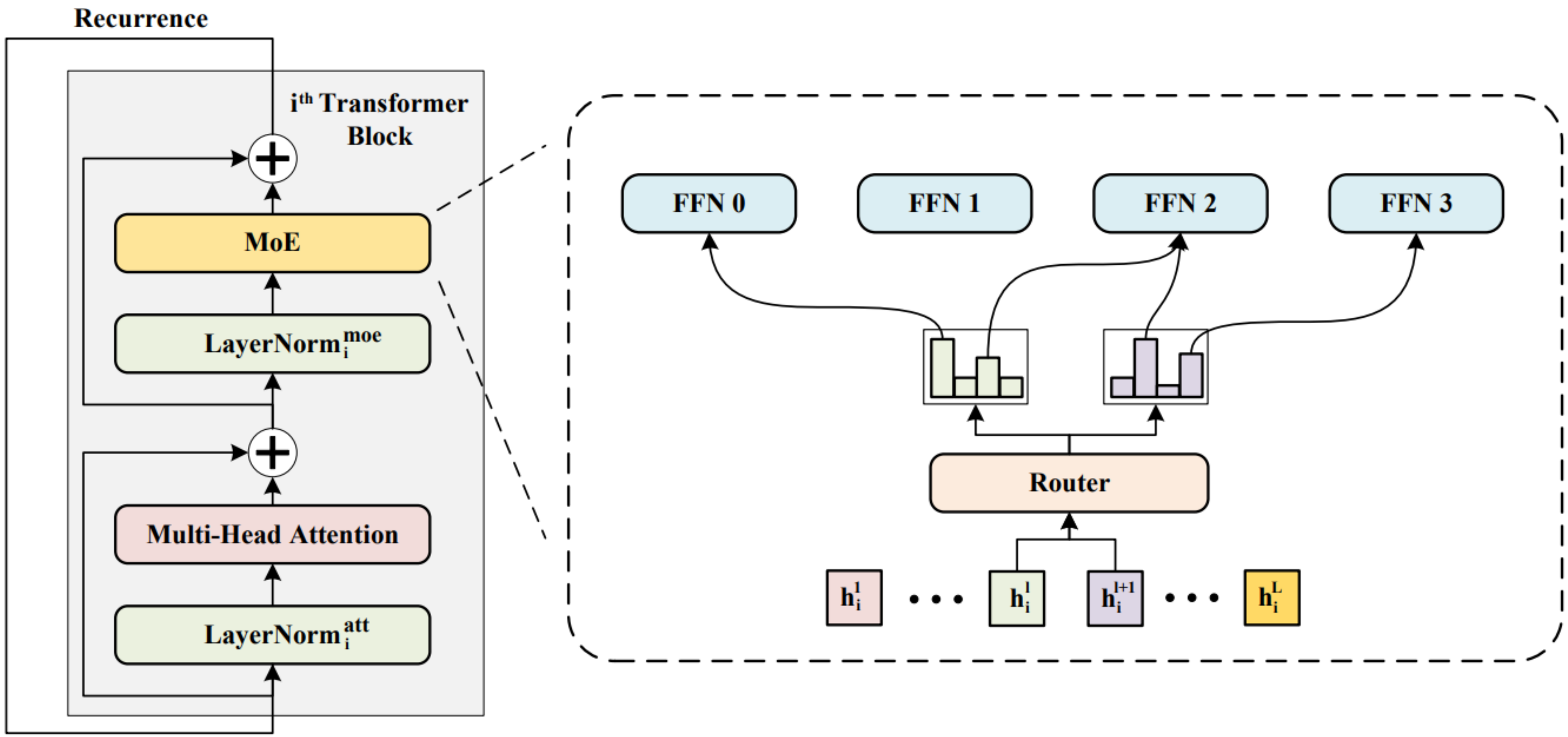
layernorm (x + f(x)) ---> layernorm(x*alpha + f(x)).

The proposed method combines the advantages of both schools, namely the good performance of Post-LN and the stable training of Pre-LN, making DEEPNORM the preferred alternative.

Hongyu Wang, Shuming Ma, Li Dong, Shaohan Huang, Dongdong Zhang, Furu Wei. DeepNet: Scaling Transformers to 1,000 Layers. https://browse.arxiv.org/pdf/2203.00555.pdf

Is the model deeper or wider?

Go Wider Instead of Deeper



- WideNet first compresses trainable parameters along with depth by parameter-sharing across transformer blocks.
- Each expert requires enough tokens to train.

Fuzhao Xue, Ziji Shi, Futao Wei, Yuxuan Lou, Yong Liu, Yang You. **Go Wider Instead of Deeper. https://arxiv.org/abs/2107.11817**

Scaling law?

Scaling Law for Neural Language Models

Performance depends strongly on scale! We keep getting better performance as we scale the model, data, and compute up!

Scaling Laws for Neural Language Models

Jared Kaplan *
Johns Hopkins University, OpenAI
jaredk@jhu.edu

Sam McCandlish*
OpenAI

Tom Henighan
OpenAI
henighan@openai.com

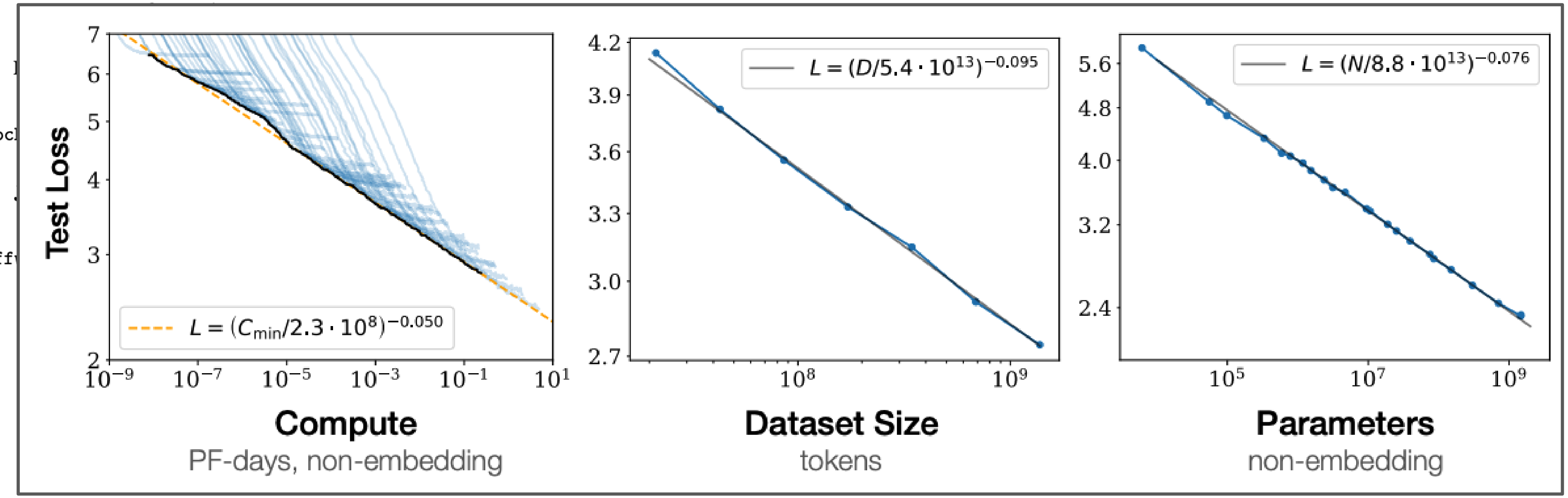
Tom B. Brown
OpenAI
tom@openai.com

bc

Scott Gray
OpenAI
scott@openai.com

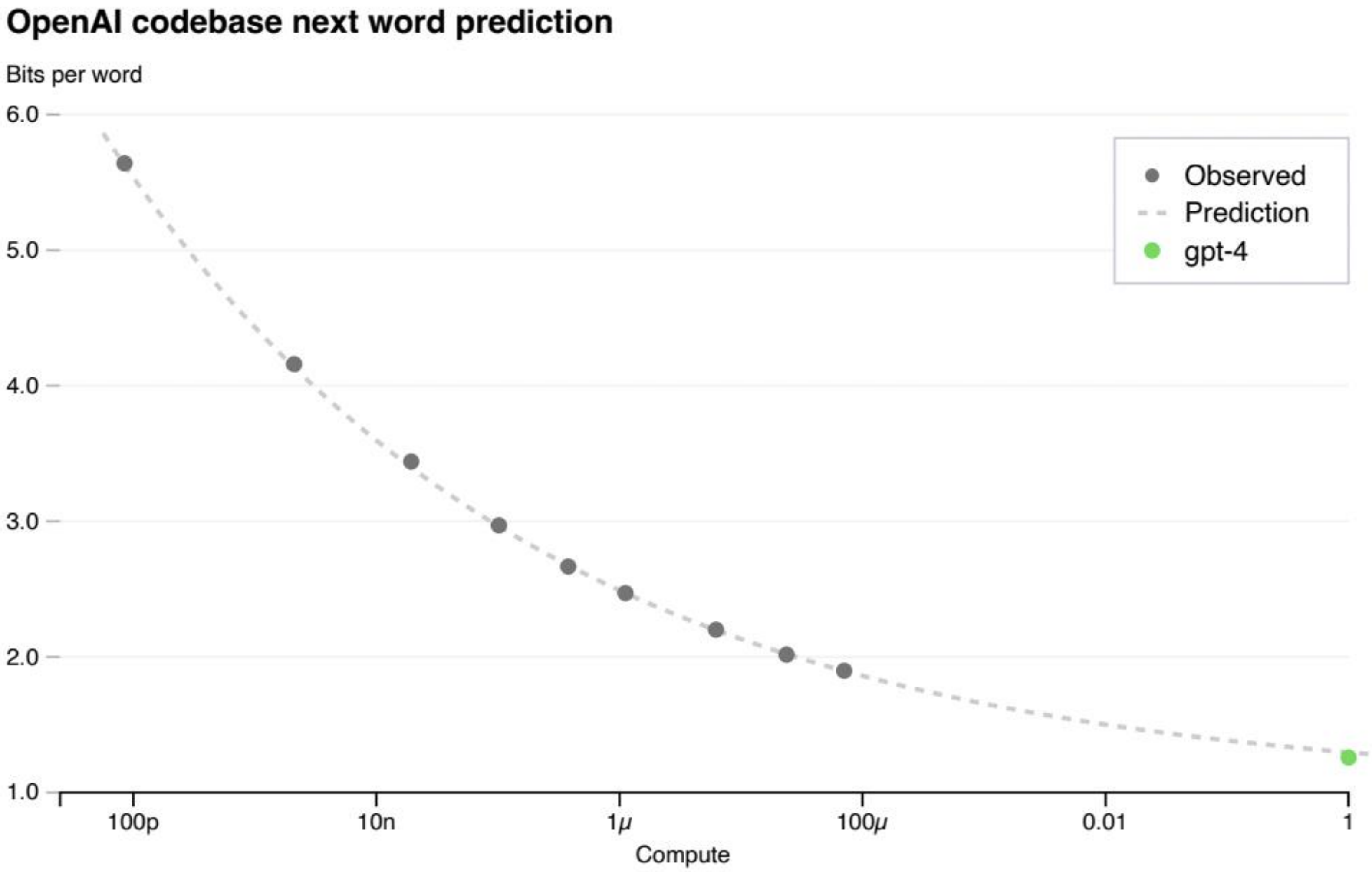
Alec Radford
OpenAI
alec@openai.com

jeff



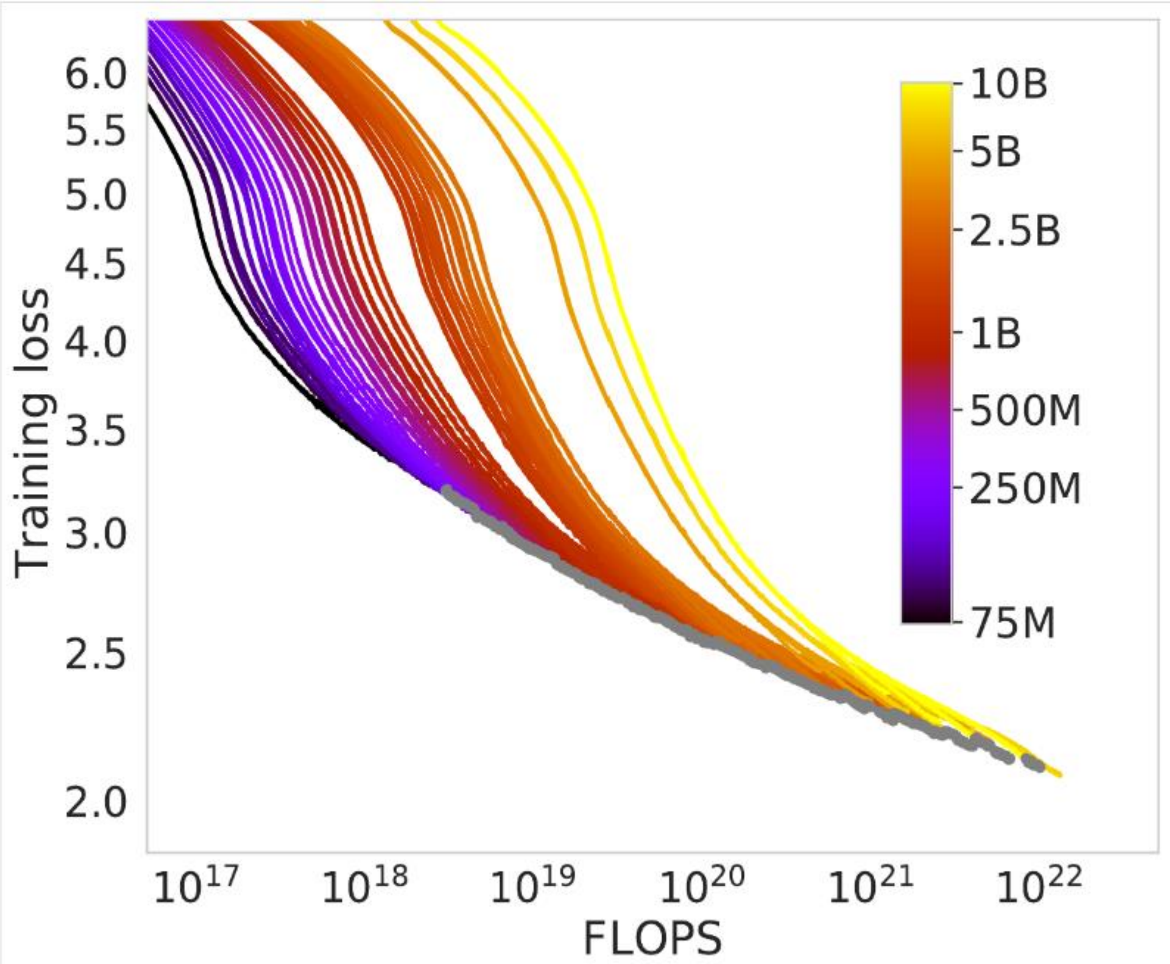
Emergent abilities of large language models (TMLR ‘22).
J. Wei, Y. Tay, R. Bommasani, C. Raffel, B. Zoph, S. Borgeaud, D. Yogatama, M. Bosma, D. Zhou, D. Metzler, E. Chi, T. Hashimoto, O. Vinyals, P. Liang, J. Dean, & W. Fedus.

Scaling laws



GPT-4 Technical Report, OpenAI (2023)

Challenge to scaling law: Chinchilla's Death



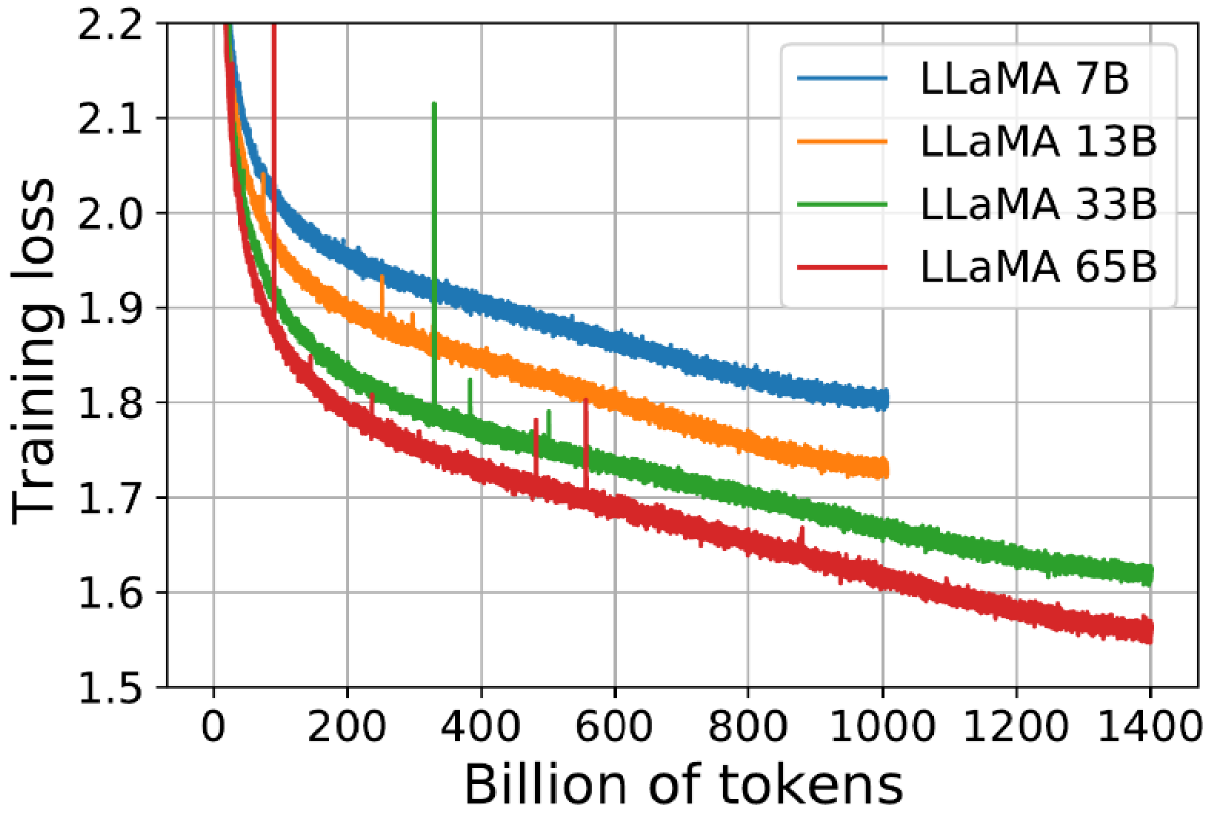
Smaller models eventually reach the limit of their capacity for knowledge, and their learning slows, while that of a **larger model, with a larger capacity, will overtake them** and reach better performance past a given amount of training time.

While estimating how to get the best bang during training, OpenAI & DeepMind attempted to draw the Pareto frontier.

https://espadrine.github.io/blog/posts/chinchilla-s-death.html

Challenge to scaling law: Chinchilla's Death

Can Chinchillas picture a Llama's sights?

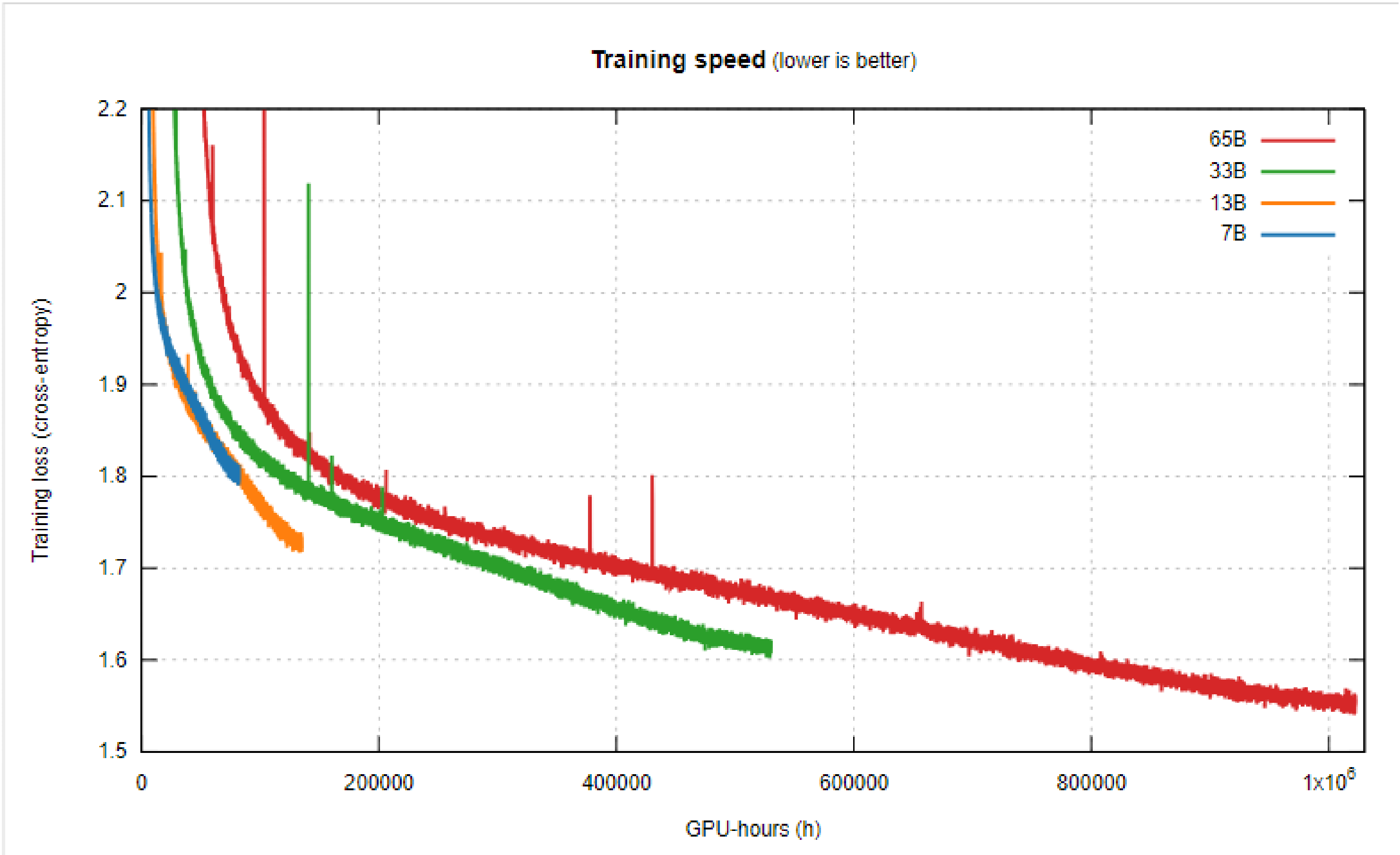


- ❖ Each curve first plummets in a **power law**,
- ❖ and then seemingly enters a **nearly-linear** decrease in loss (corresponding to a fairly constant rate of knowledge acquisition).
- ❖ At the very tip of the curve, they all break this line by **flattening** slightly.

- ❖ This should consider the cosine LR schedule.

https://espadrine.github.io/blog/posts/chinchilla-s-death.html

Challenge to scaling law: Chinchilla's Death



Let's picture instead a race: All those models start at the same time, and we want to know which one crosses the finish line first.

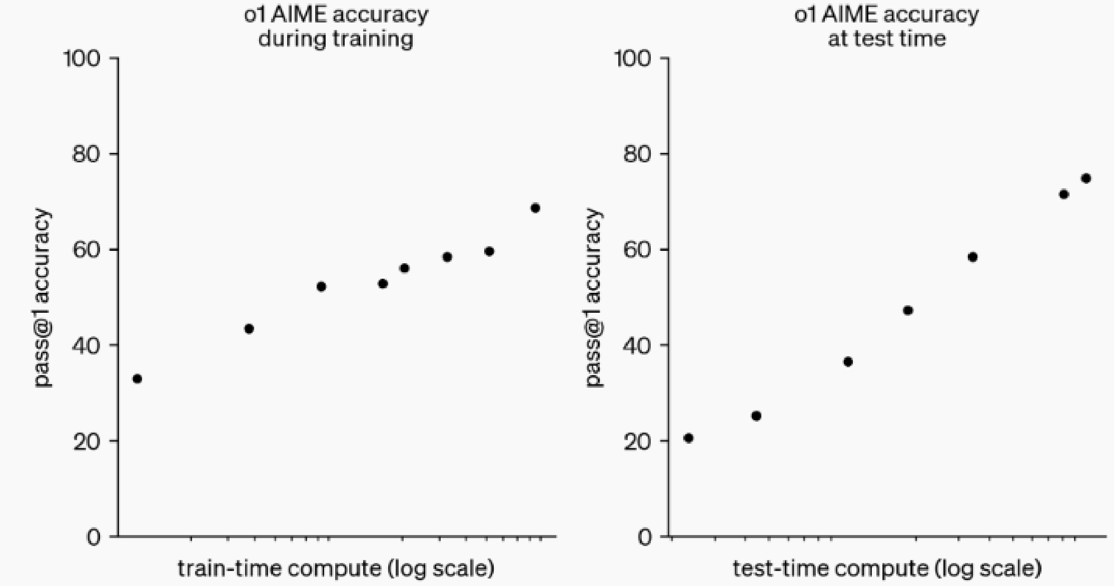
In other words, when throwing a fixed amount of compute at the training, who learns the most in that time?

the 7B enters a near-linear regime, with a steep downward trend, and seems on its way to maybe overpass the 13B again?

https://espadrine.github.io/blog/posts/chinchilla-s-death.html

New Scaling law from OpenAI o1

Our **large-scale reinforcement learning** algorithm teaches the model how to think productively using its chain of thought in a highly data-efficient training process. We have found that the performance of o1 consistently improves with **more reinforcement learning** (train-time compute) and with **more time spent thinking** (test-time compute). The constraints on scaling this approach differ substantially from those of LLM pretraining.



o1 performance smoothly improves with both train-time and test-time compute

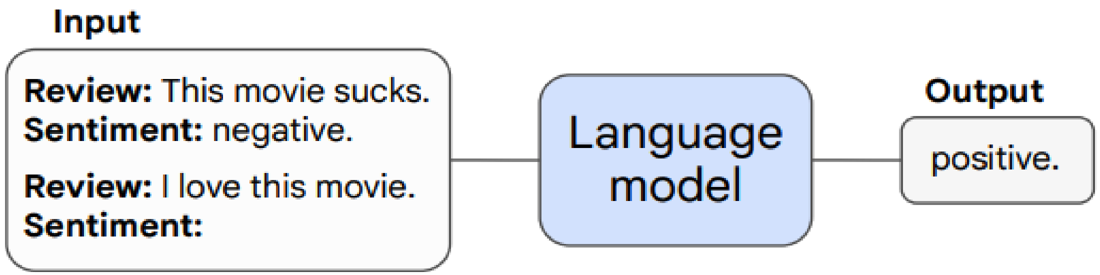
https://openai.com/index/learning-to-reason-with-llms/

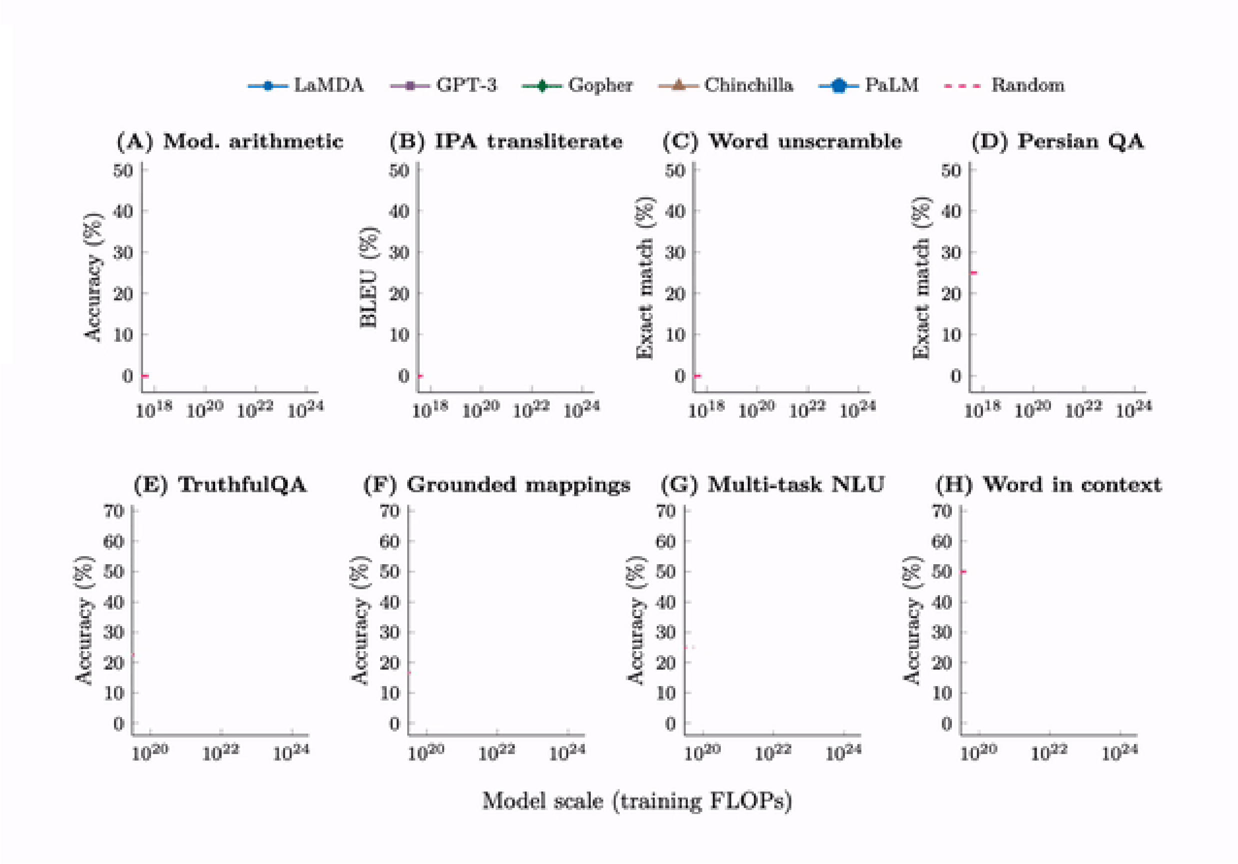
Emergent ability?

Emergent properties in LLMs:

Some ability of LM is not present in smaller models but is present in larger models

Emergent Capability: Few-shot prompting

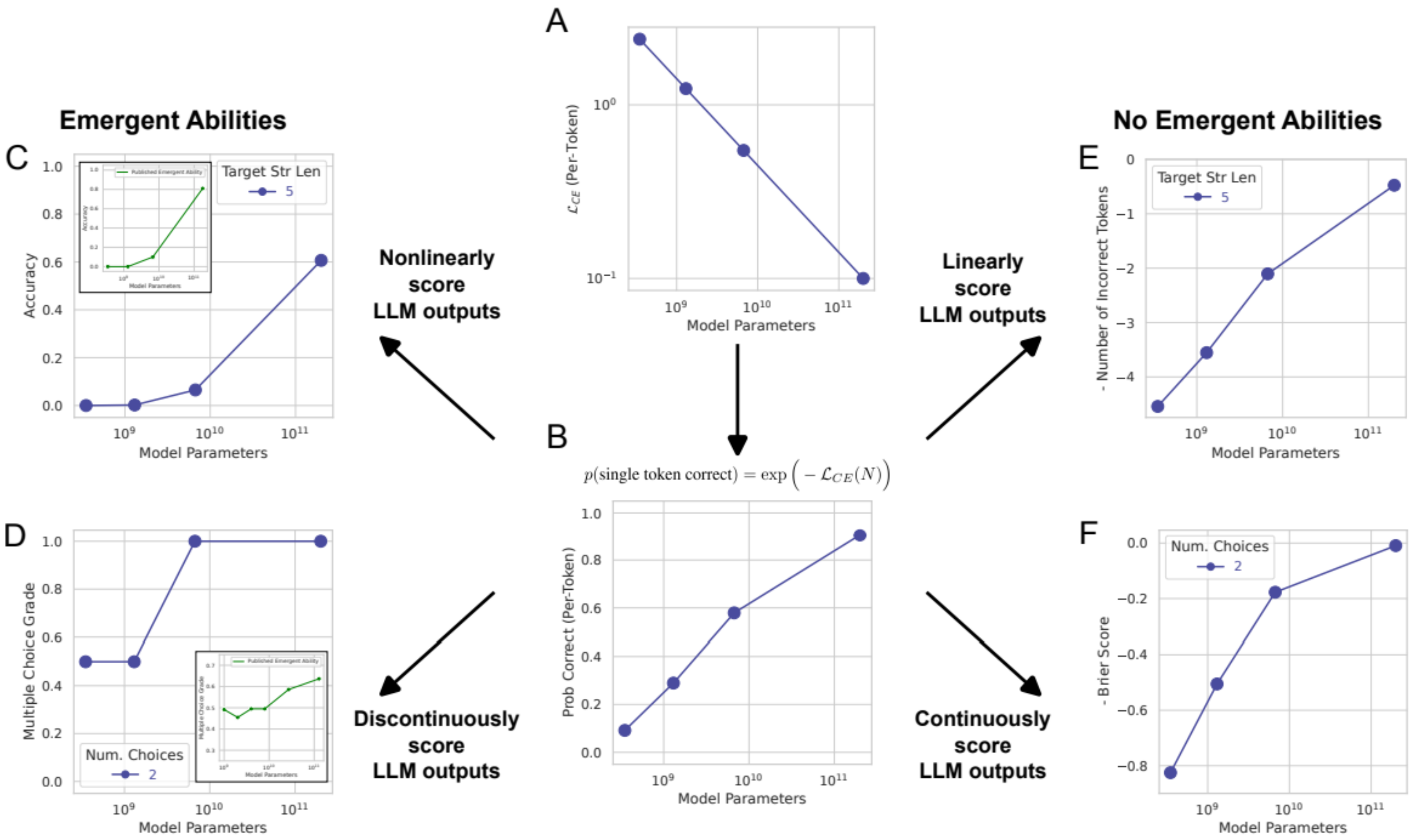




> A few-shot prompted task is emergent if it achieves random accuracy for small models and above-random accuracy for large models.

https://docs.google.com/presentation/d/1yzbmYB5E7G8lY2-KzhmArmPYwwl7o7CUST1xRZDUu1Y/edit?resourcekey=0-6_TnUMoK WCk_FN2BiPxmbw#slide=id.g1fc34b3ac18_0_27

Emergent capabilities may be a consequence of metric choice



It seems that emergent ability of a model only occurs if the measure of per-token error rate of any model is scaled **non-linearly or discontinuously**.

Rylan Schaeffer, Brando Miranda, and Sanmi Koyejo. Are Emergent Abilities of Large Language Models a Mirage? https://browse.arxiv.org/pdf/2304.15004.pdf

A Quick Reminder

Assignment 1:

Our first assignment has been posted for a while. Please be aware if you still haven’t started yet
The deadline is Feb. 27th, by the end of the day.

A study

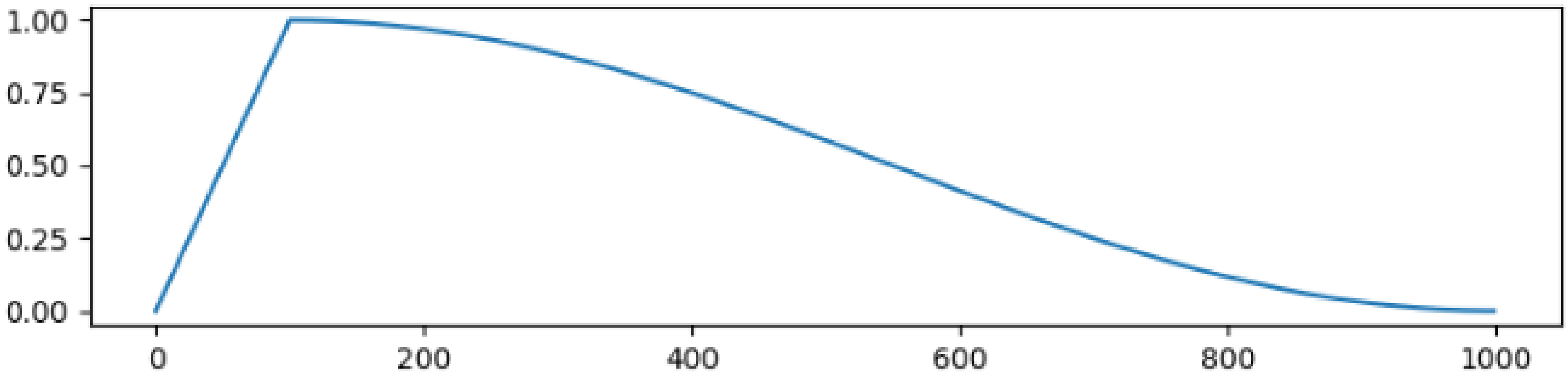
What is new in Qwen 2.5 and DeepSeek V3?

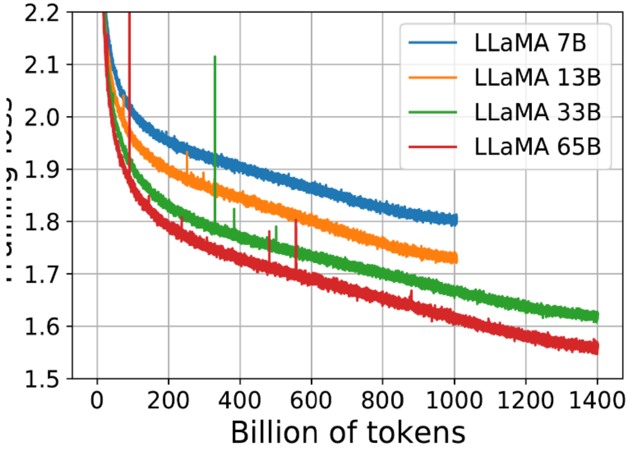
Acknowledgement

- Princeton COS 484: Natural Language Processing. Contextualized Word Embeddings. Fall 2019
- CS447: Natural Language Processing. Language Models. *http://courses.engr.illinois.edu/cs447*
- *http://cs231n.stanford.edu/*
- *https://medium.com/@gautam.karmakar/summary-seq2seq-model-using-convolutional-neural-network-b1eb100fb4c4*
- Transformers and sequence- to-sequence learning. CS 685, Fall 2021. Mohit Iyyer. College of Information and Computer Sciences. University of Massachusetts Amherst. https://people.cs.umass.edu/~miyyer/cs685_f21/slides/05-transformers.pdf
- https://www.digitalocean.com/community/tutorials/deepseek-r1-large-language-model-capabilities

Challenge to scaling law: Chinchilla's Death

Can Chinchillas picture a Llama's sights?





The slowdown in learning is an artefact of cosine schedule. The model does not necessarily cease to have the capacity to learn at the same near-linear rate!

https://espadrine.github.io/blog/posts/chinchilla-s-death.html