Transformers

CS 5624: Natural Language Processing

Fall 2026

https://tuvllms.github.io/nlp-fall-2026/

Tu Vu

VIRGINIA TECH

Logistics

- Office hours starting tomorrow
  - both in-person and via Zoom (locations and links available on course website)

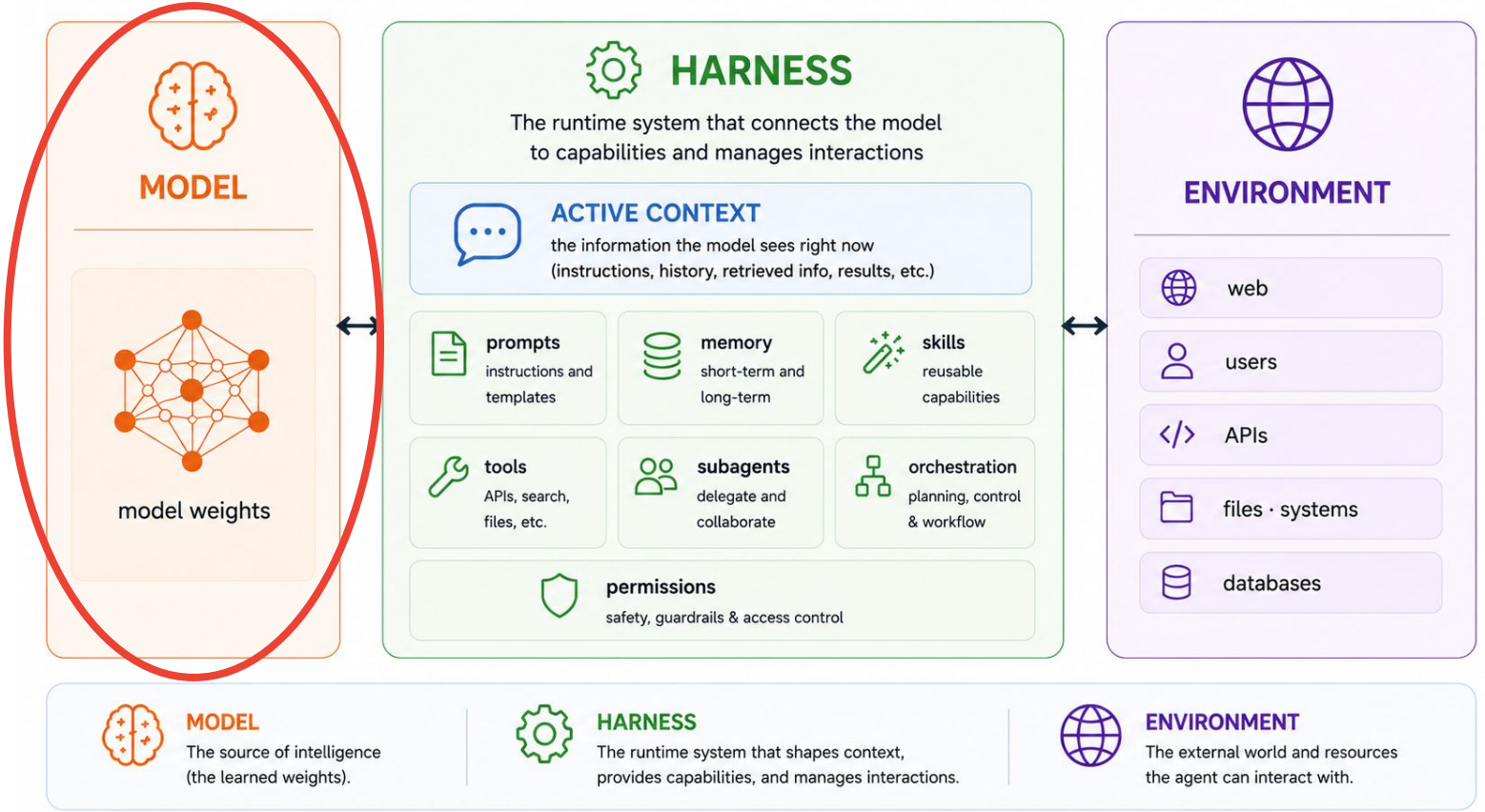
- Student presentations
  - Search for teammates on Piazza/Discord or reach out to us at cs5624instructors@gmail.com
  - Google form for submitting group information available on Piazza/Discord (due 9/8)

- HW0 / Quiz0 released later in the week

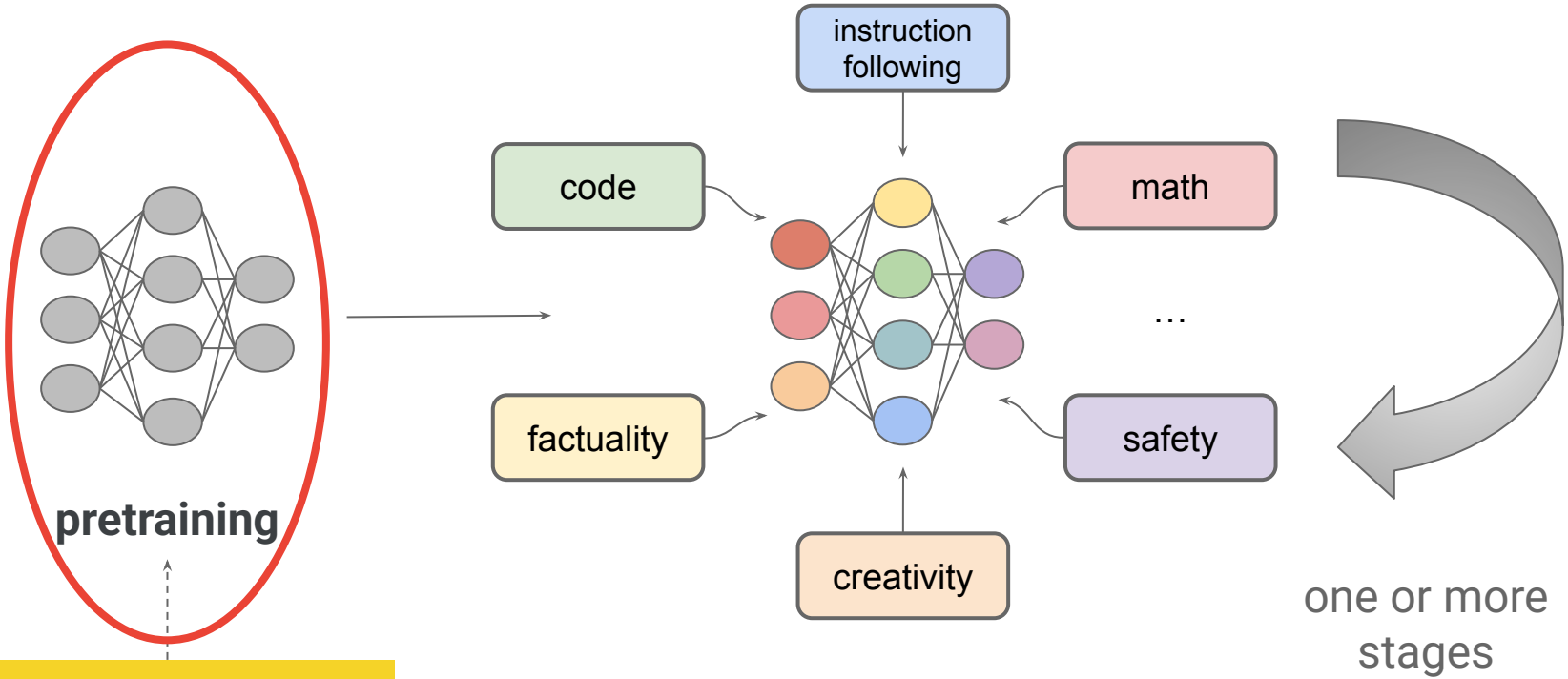
Where are we?

AI → Language understanding → Language modeling

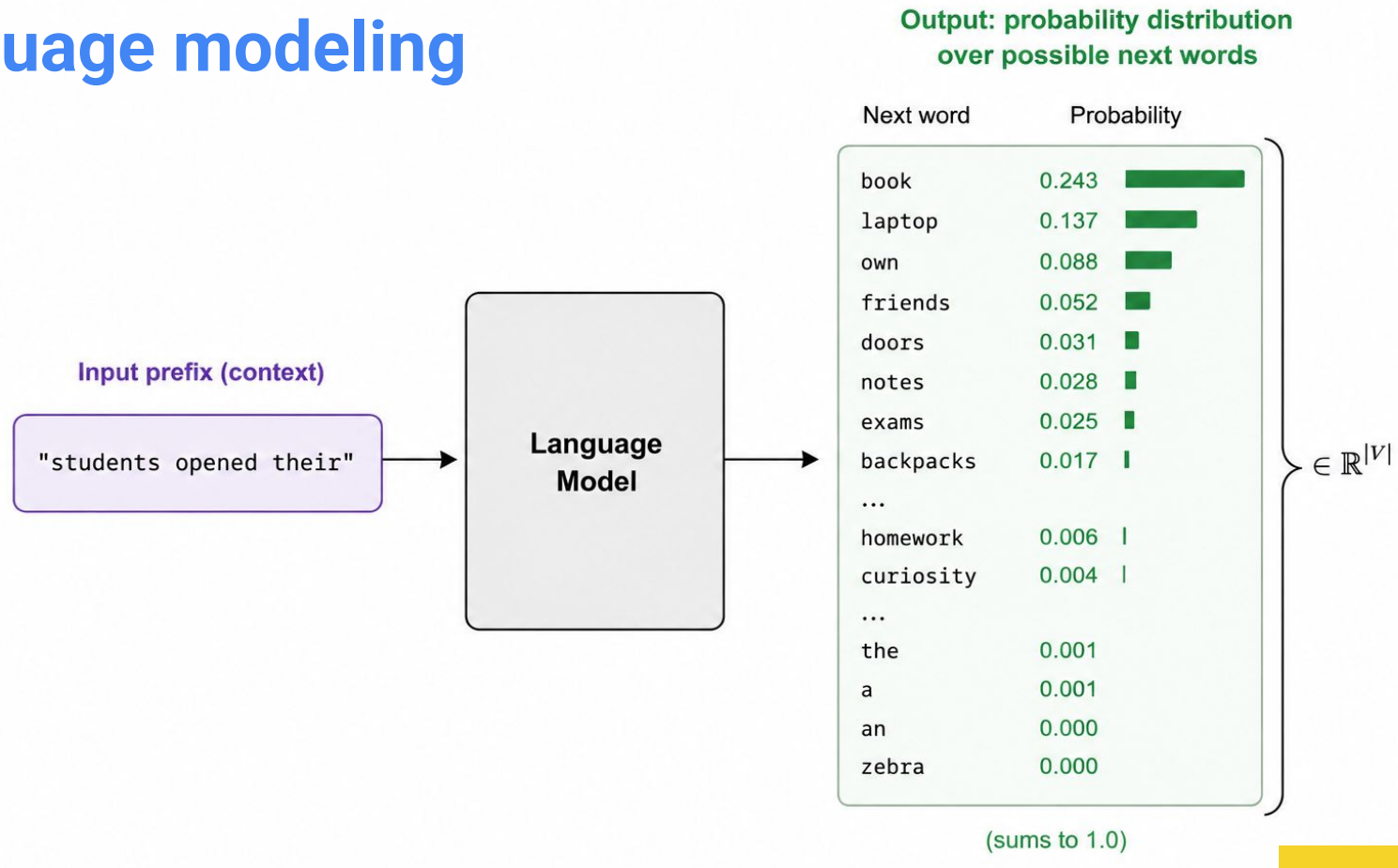
Where are we? (cont'd)



The development of modern LLMs



Language modeling



Autoregressive decoding

- **What is your favorite thing about fall?** → LM

Language modeling (cont'd)

$P(w \mid \text{“students opened their”})$?

N-gram language model

- Approximate the prefix by just the last ***n-1*** words
- bigram (n=2) model

$$P(w \mid \text{“students opened their”}) = P(w \mid \text{“their”})$$

- trigram (n=3) model

$$P(w \mid \text{“students opened their”}) = P(w \mid \text{“open their”})$$

N-gram language model (cont'd)

unigram counts

i	want	to	eat	chinese	food	lunch	spend
2533	927	2417	746	158	1093	341	278

target

prefix

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

want followed **i** 827 times

source: Jurafsky and Martin

Problems with n-gram language models

- As n increases, the number of possible n-grams grows exponentially (many n-grams have insufficient or no data)
- Storing and processing large n-grams requires more memory and computational power
- Beyond a certain point, increasing n may not yield significant performance improvements, especially if the dataset does not contain sufficient examples of longer n-grams

N-gram language model (cont'd)

Model	Probability	Naive storage
Unigram	$P(w_1)$	V
Bigram	$P(w_2 \mid w_1)$	V^2
Trigram	$P(w_3 \mid w_1, w_2)$	V^3
n-gram	$P(w_n \mid w_1, \ldots, w_{n-1})$	V^n

Problems with n-gram language models (cont'd)

- Treat semantically similar prefixes independently of each other

  "students opened their ___"

  "pupils opened their ___"

  "scholars opened their ___"

  "students began reading their ___"

Shouldn't we share information across these prefixes?

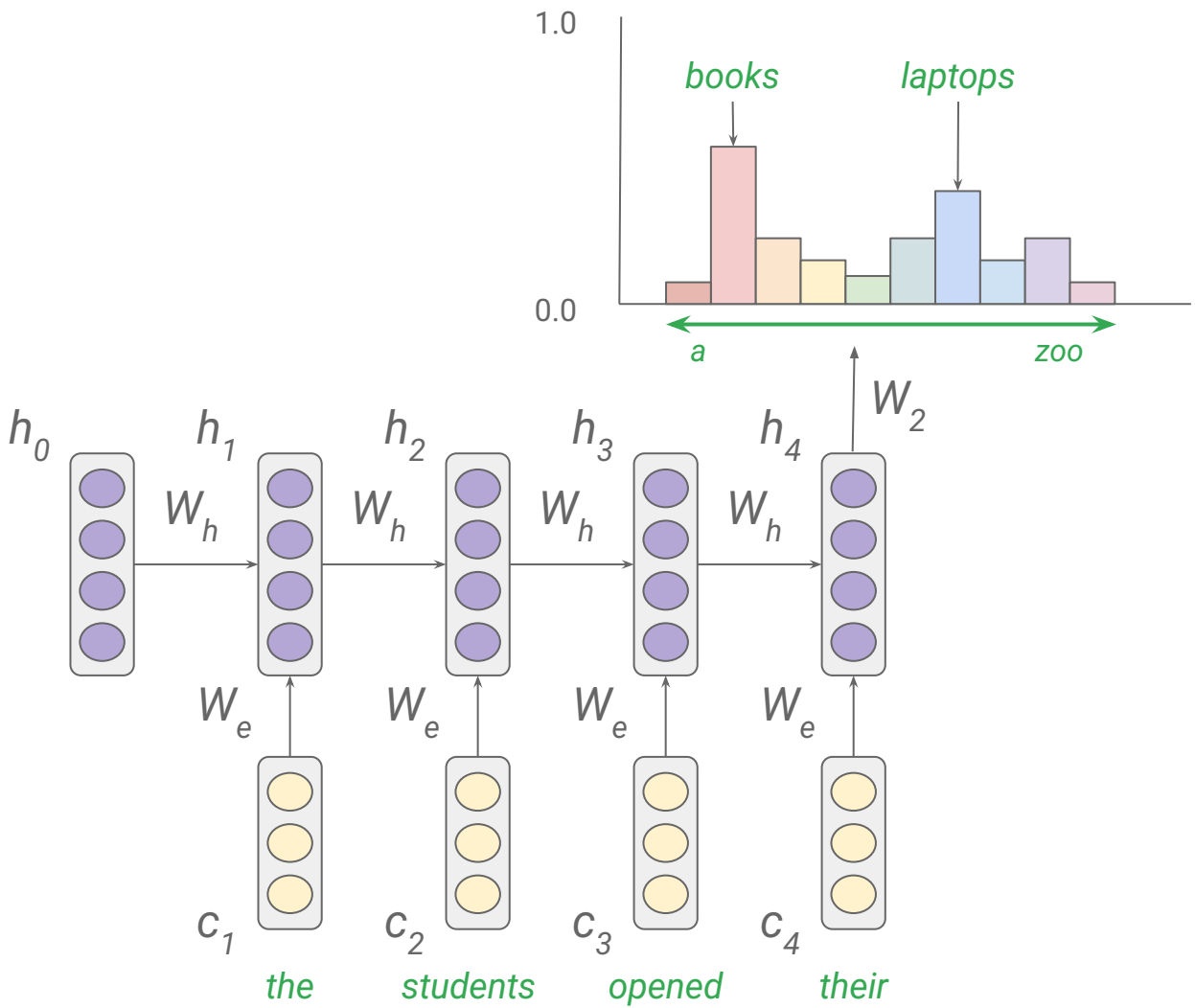
Recurrent neural network (RNN) language models

hidden states

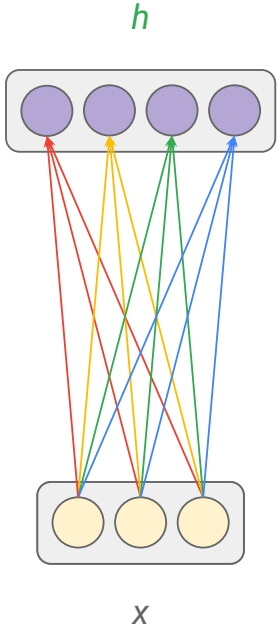
$$h^{(t)} = f(W_h h^{(t-1)} + W_e c^t)$$

output distribution

$$\hat{y} = softmax(W_2 h^{(n-1)})$$



Matrix-vector multiplication



linear projection

Matrix A (dimensions 4×3):

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \\ a_{41} & a_{42} & a_{43} \end{bmatrix}$$

Vector x (dimensions 3×1):

$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

Resulting vector b (dimensions 4×1):

$$b = A \cdot x = \begin{bmatrix} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 \\ a_{41}x_1 + a_{42}x_2 + a_{43}x_3 \end{bmatrix}$$

Recurrent neural networks (RNNs)

- RNNs advantages
  - can handle much longer histories
  - can generalize better over contexts of similar words
  - are more accurate at word-prediction

- RNNs disadvantages
  - are much more complex
  - are slower and need more energy to train
  - and are less interpretable than n-gram models

Problems with RNNs

- Bottleneck representation issue
- Lack of parallelism

Transformers

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com

Noam Shazeer* Google Brain noam@google.com

Niki Parmar* Google Research nikip@google.com

Jakob Uszkoreit* Google Research usz@google.com

Llion Jones* Google Research llion@google.com

Aidan N. Gomez*† University of Toronto aidan@cs.toronto.edu

Łukasz Kaiser* Google Brain lukaszkaiser@google.com

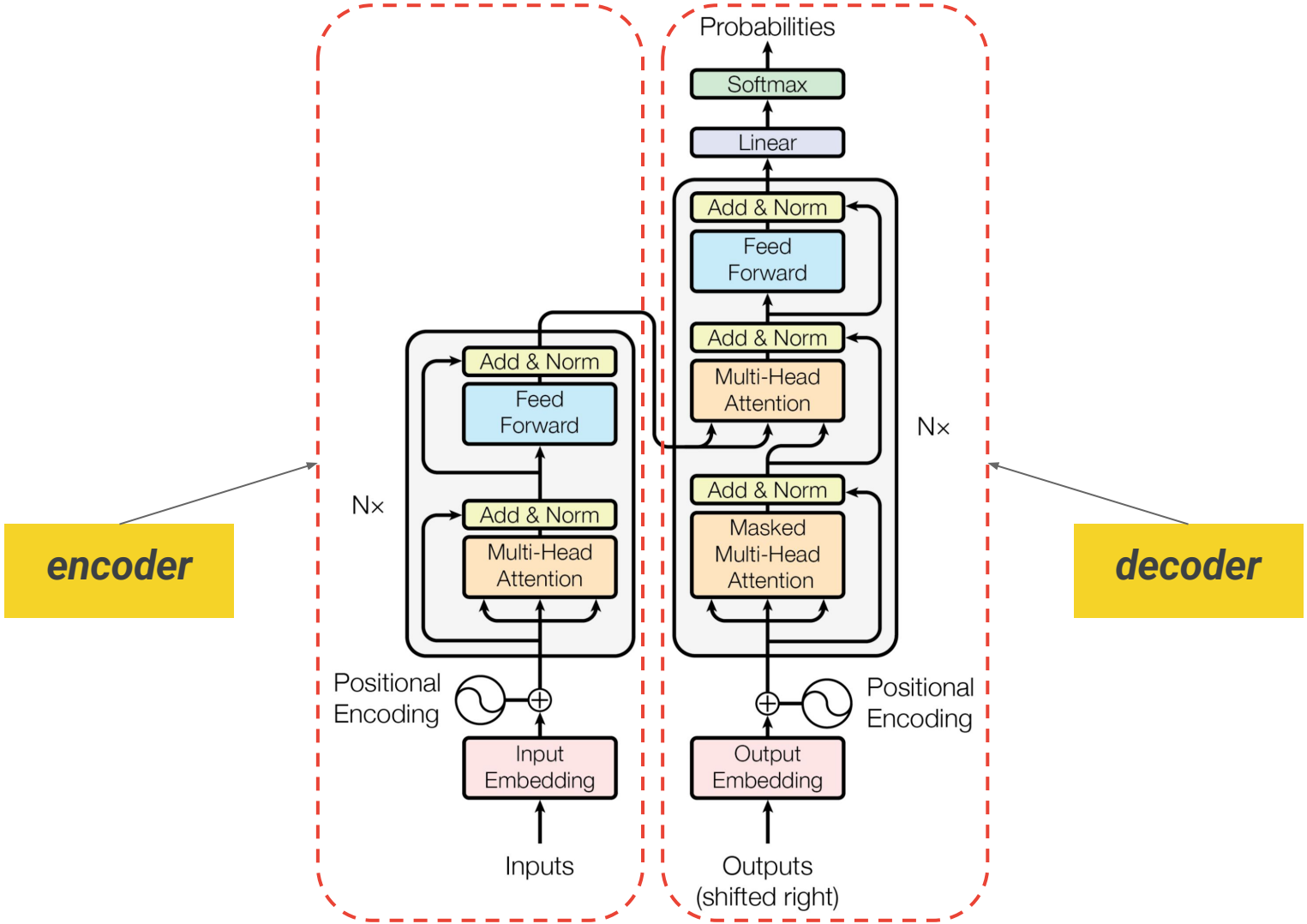
Illia Polosukhin* illia.polosukhin@gmail.com

Abstract

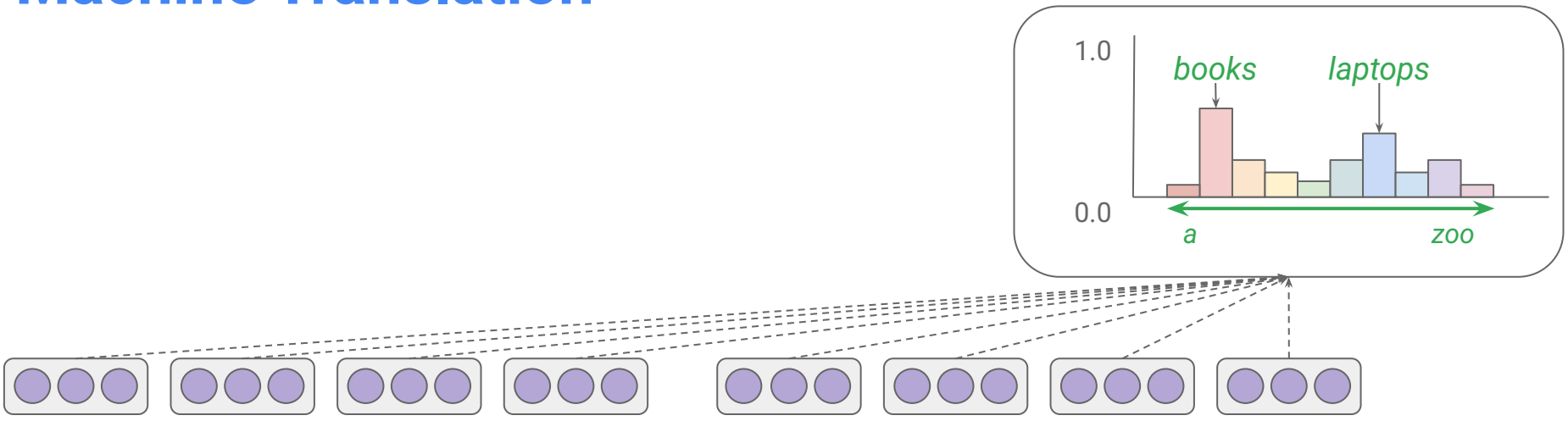
The dominant sequence transduction models are based on complex recurrent or convolutional neural networks in an encoder-decoder configuration. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.0 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature. We show that the Transformer generalizes well to other tasks by applying it successfully to English constituency parsing both with large and limited training data.

Different Transformers architectures

- Encoder-only
  - BERT
- Encoder-decoder
  - T5
- Decoder-only
  - GPT


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

Machine Translation

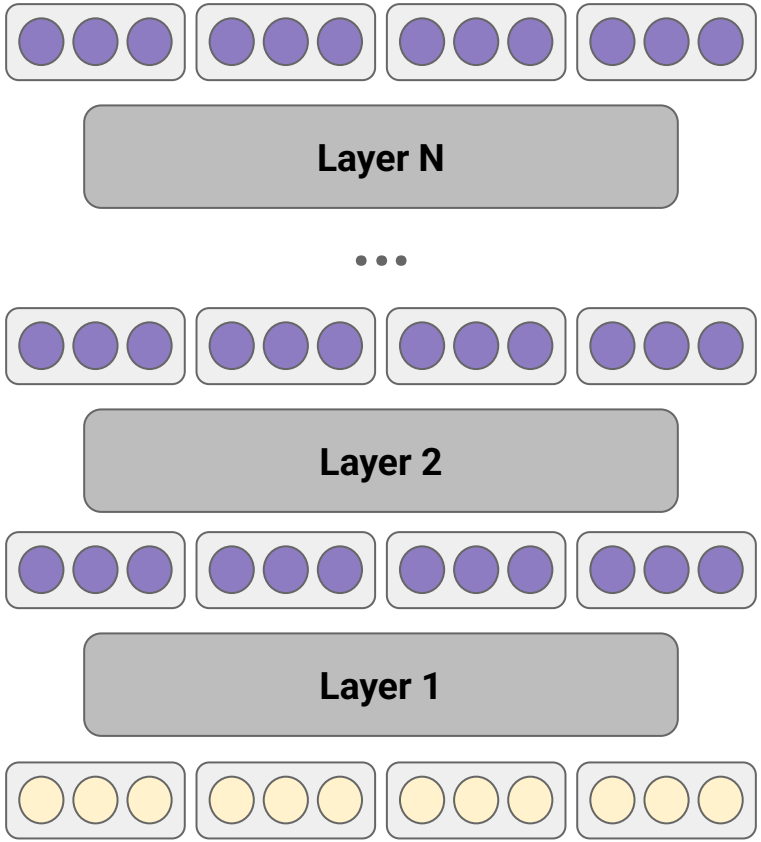


les étudiants ont ouvert leurs livres

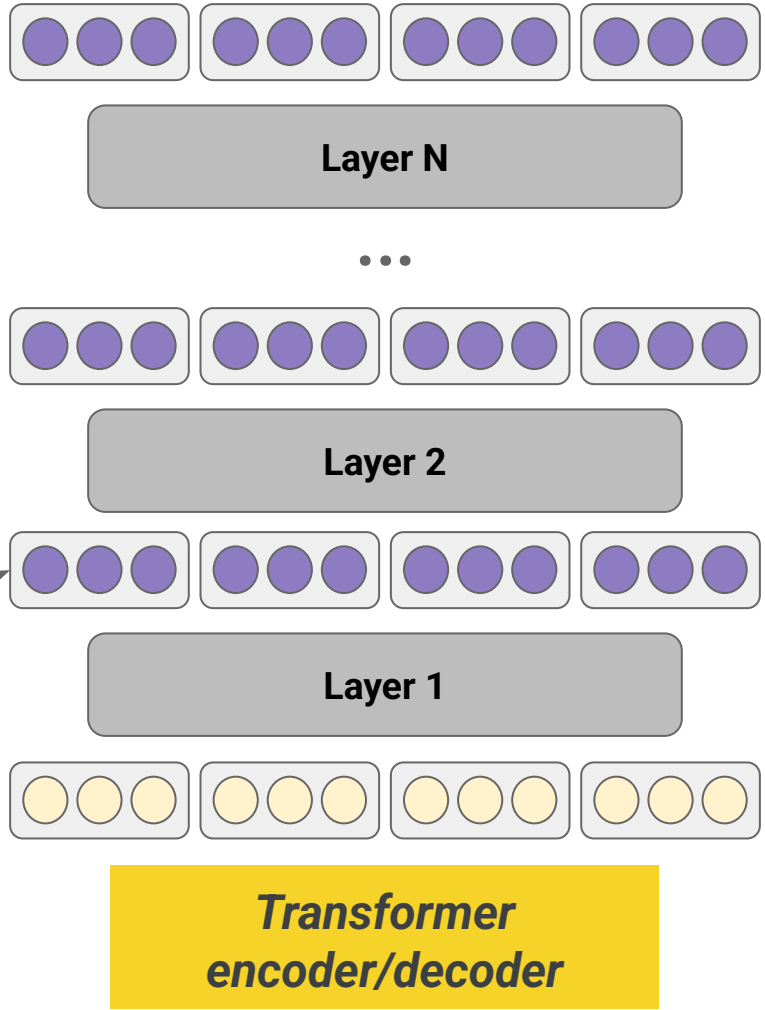
the *students* *opened* *their*

encoder ***decoder***

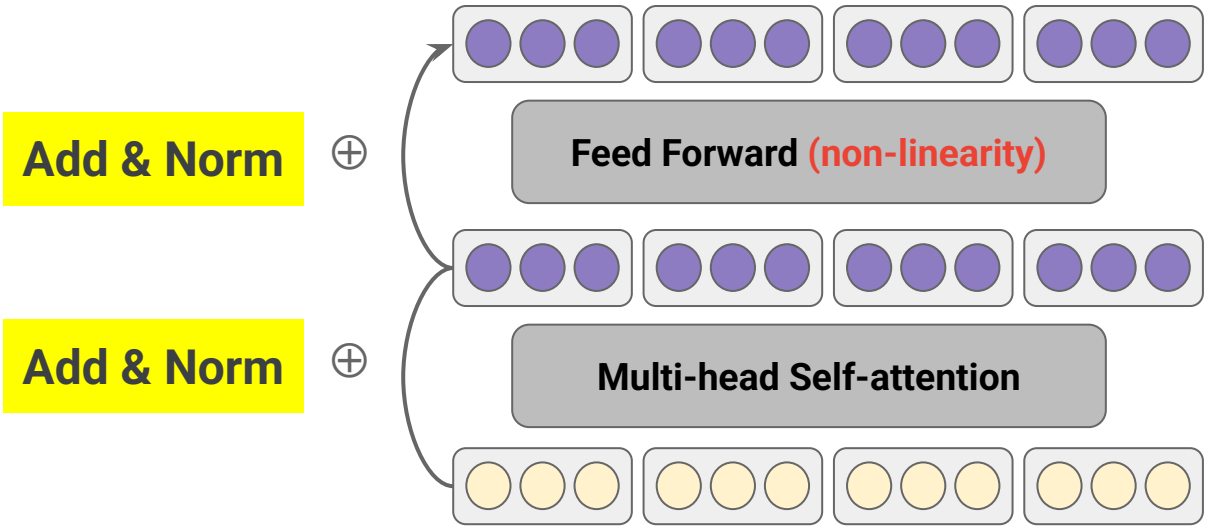
N layers



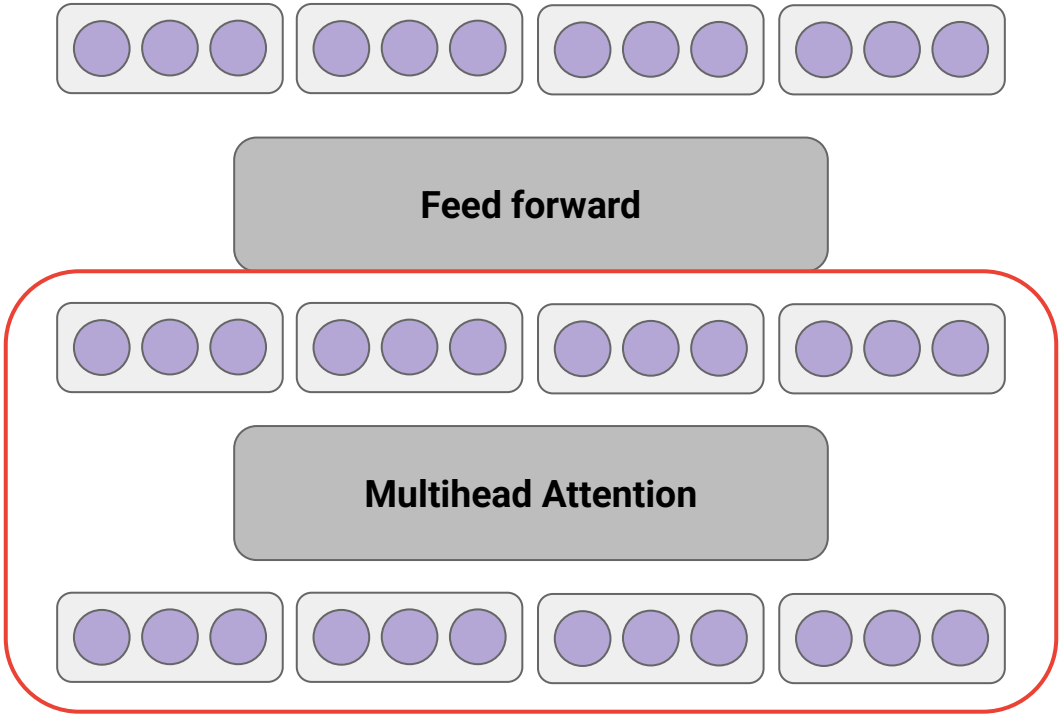
N layers


Layer N
Layer 2
Layer 1
hidden state/ activations
Transformer encoder/decoder

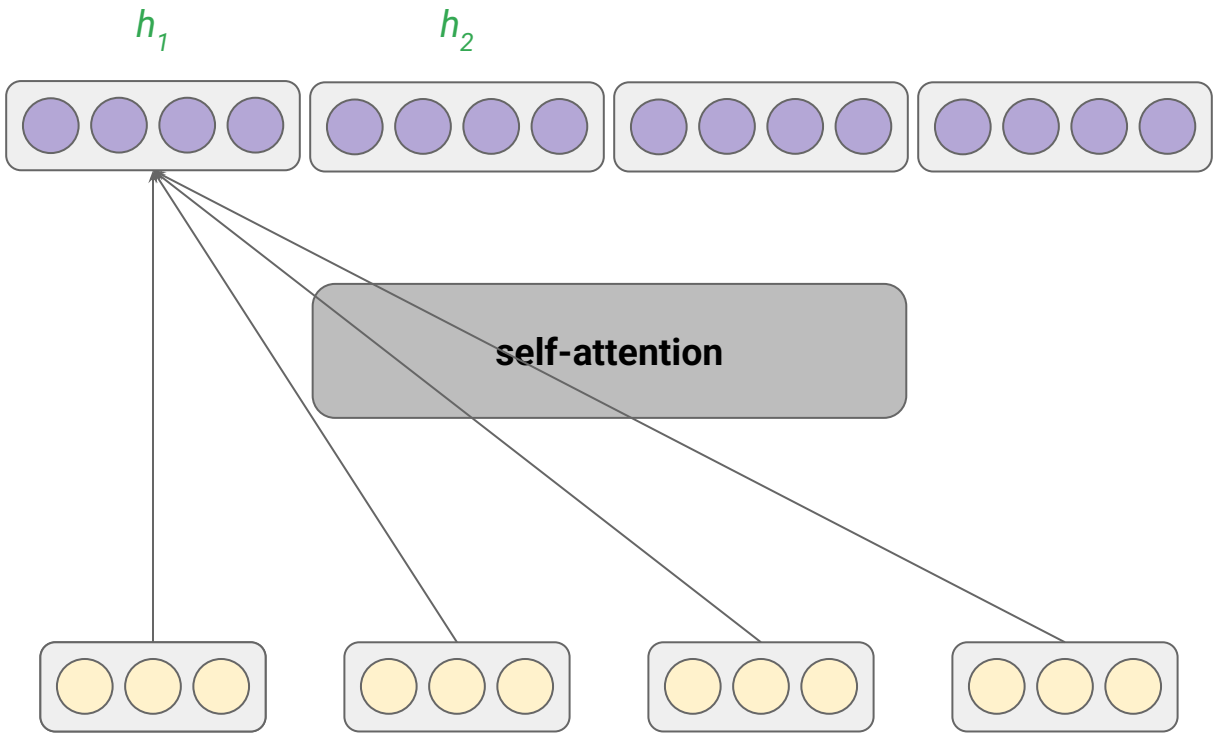
Transformer block (one layer)



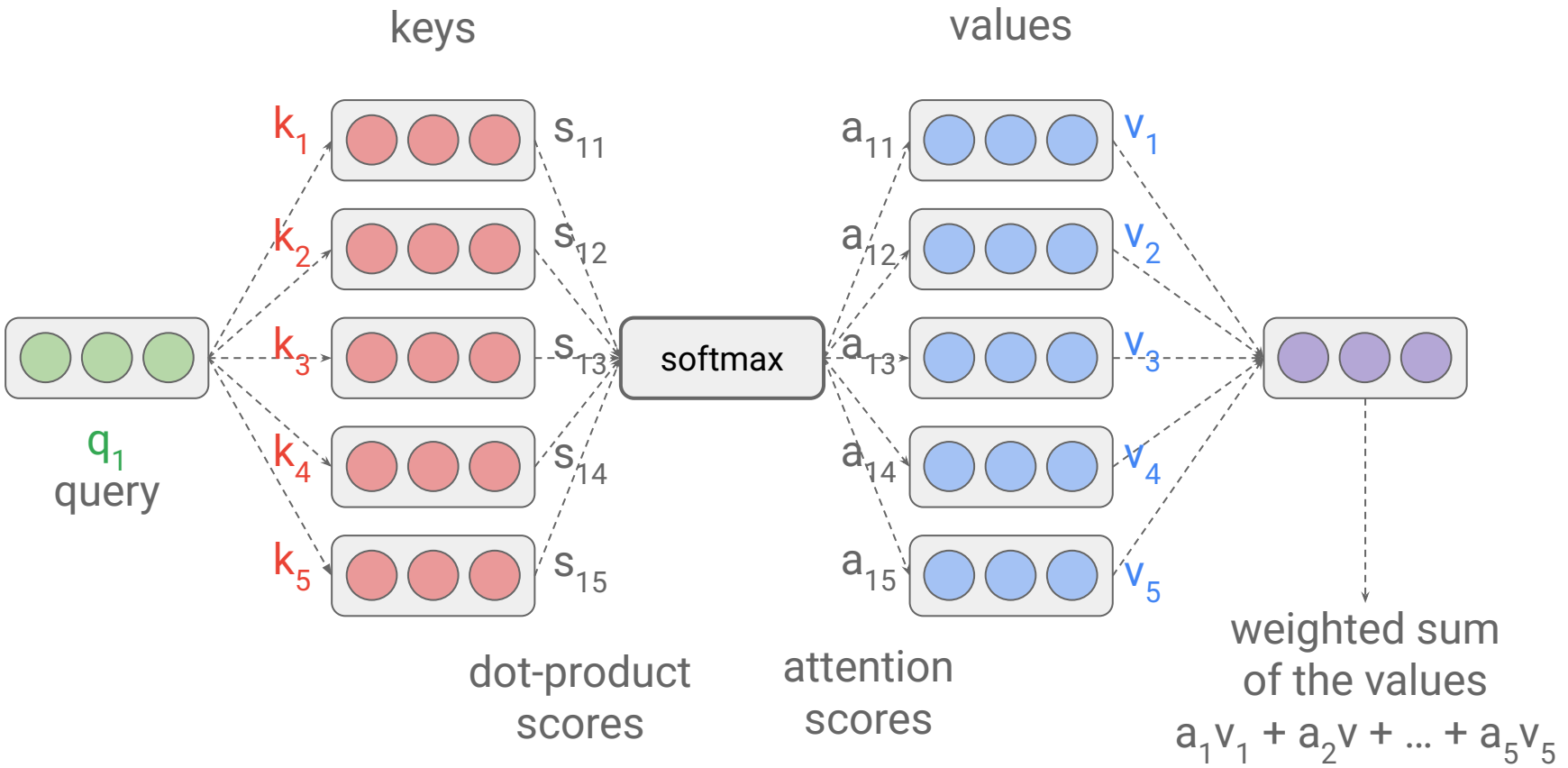
Transformer block



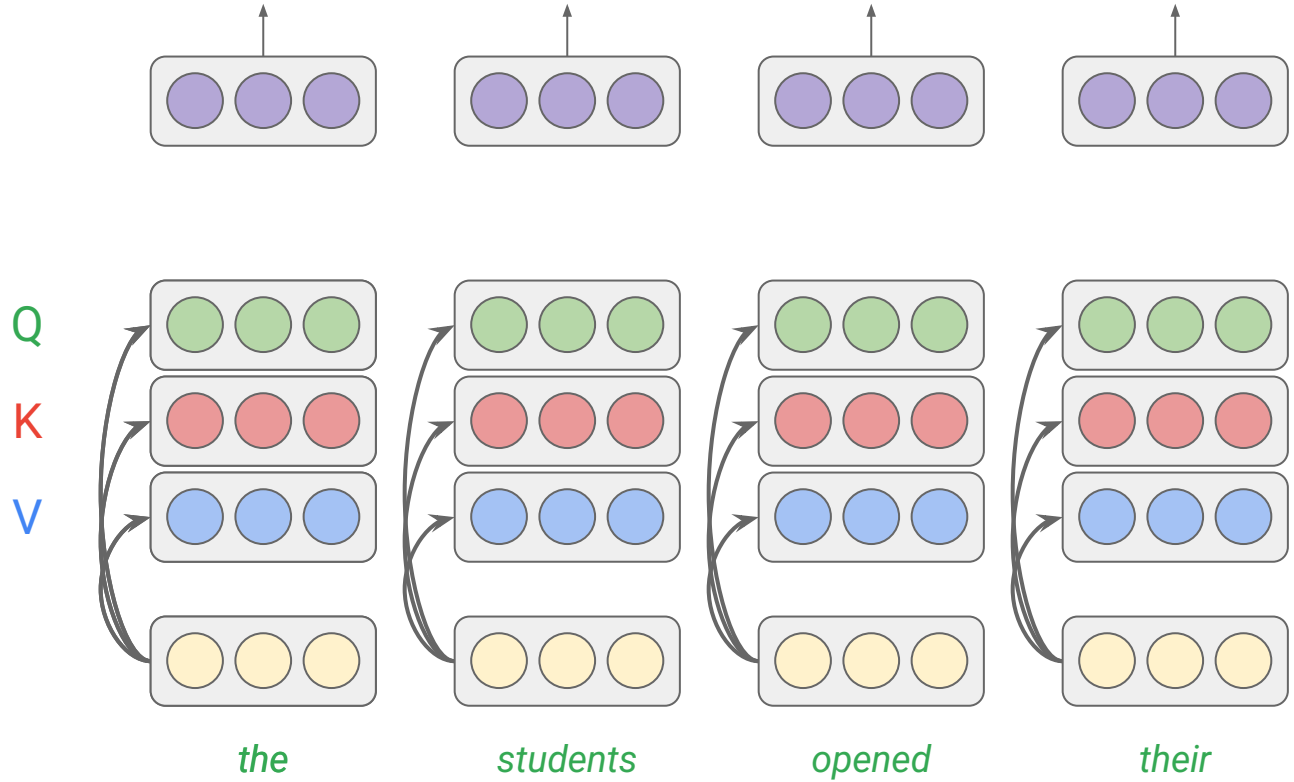
Self-attention



Attention


keys
values
k_1
k_2
k_3
k_4
k_5
s_{11}
s_{12}
s_{13}
s_{14}
s_{15}
softmax
a_{11}
a_{12}
a_{13}
a_{14}
a_{15}
v_1
v_2
v_3
v_4
v_5
q_1
query
dot-product scores
attention scores
weighted sum of the values
$a_1v_1 + a_2v + \ldots + a_5v_5$

Attention (cont'd)



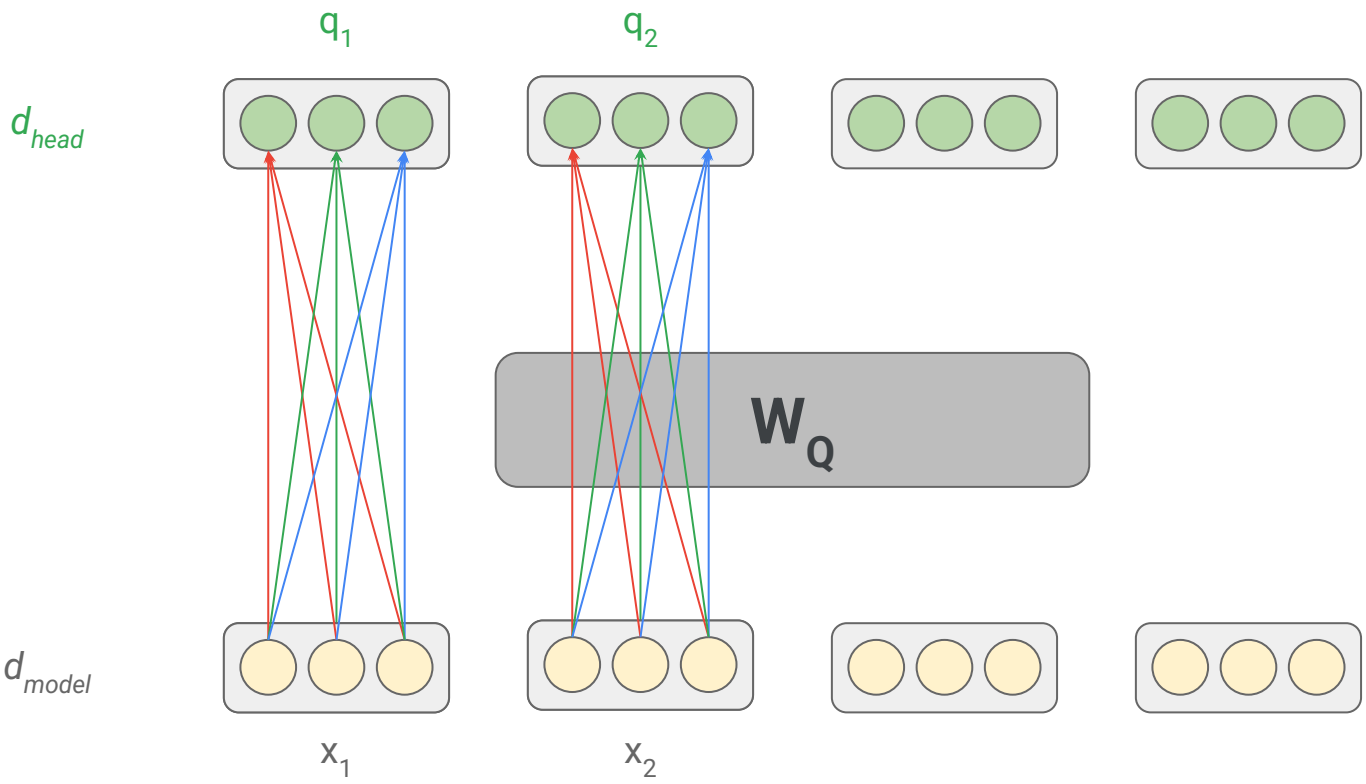
$$Q = X \cdot W_Q$$

$$K = X \cdot W_K$$

$$V = X \cdot W_V$$

linear projections

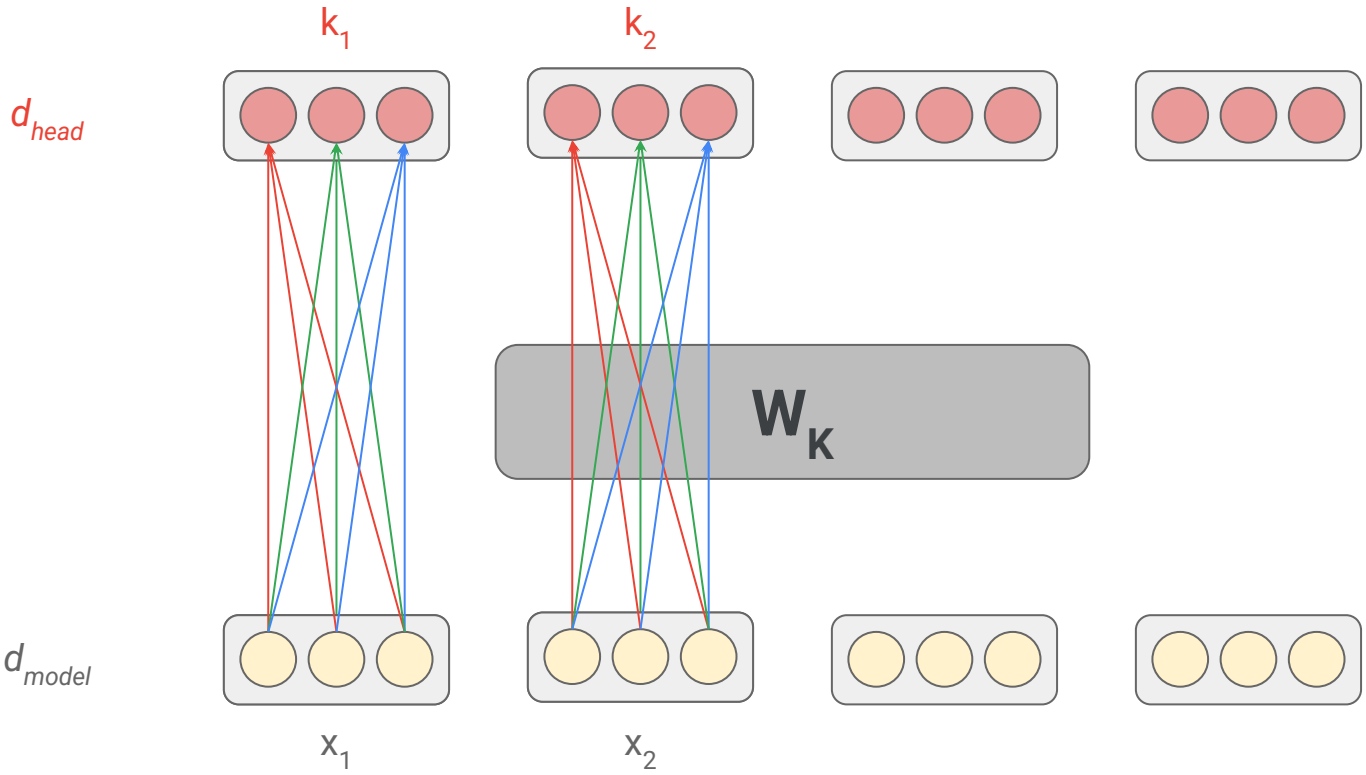
Query vectors



$$Q = X \cdot W_Q$$

linear projections

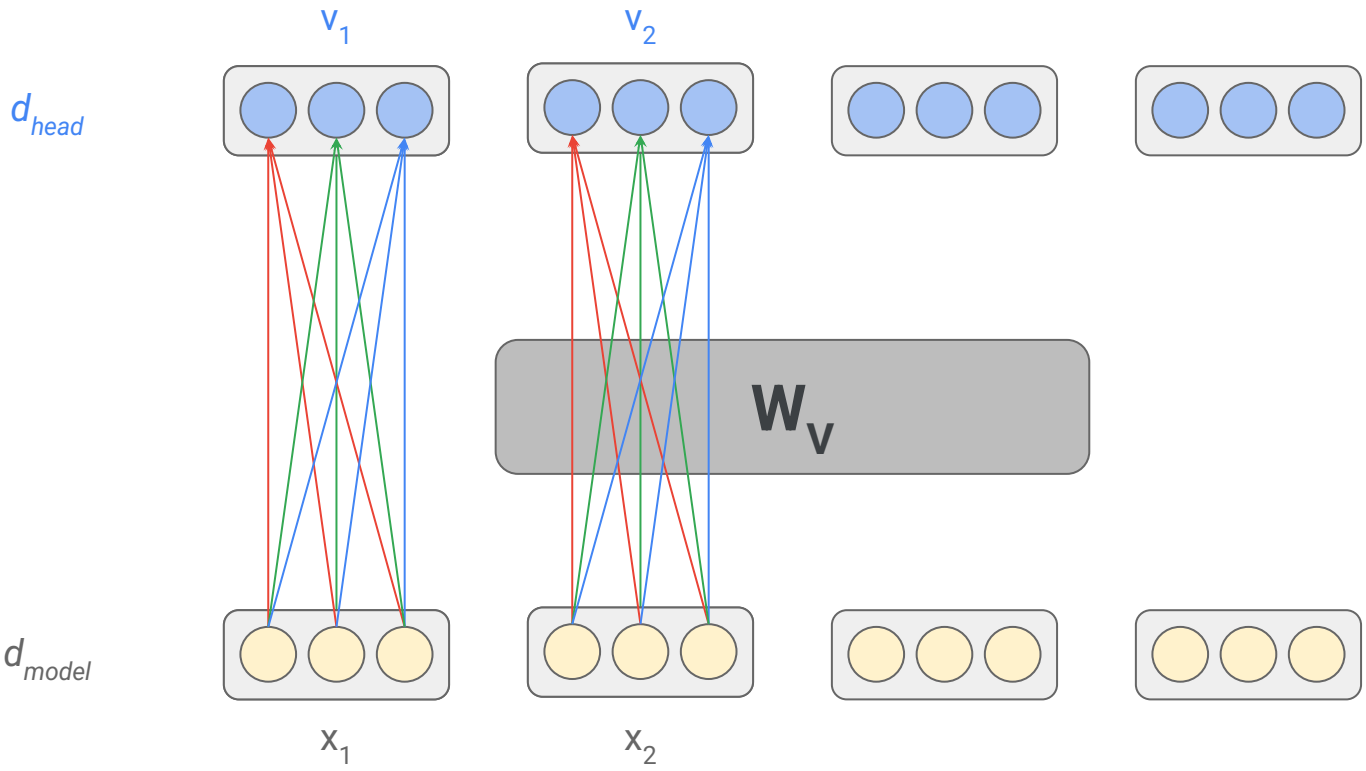
Key vectors



$$K = X \cdot W_K$$

linear projections

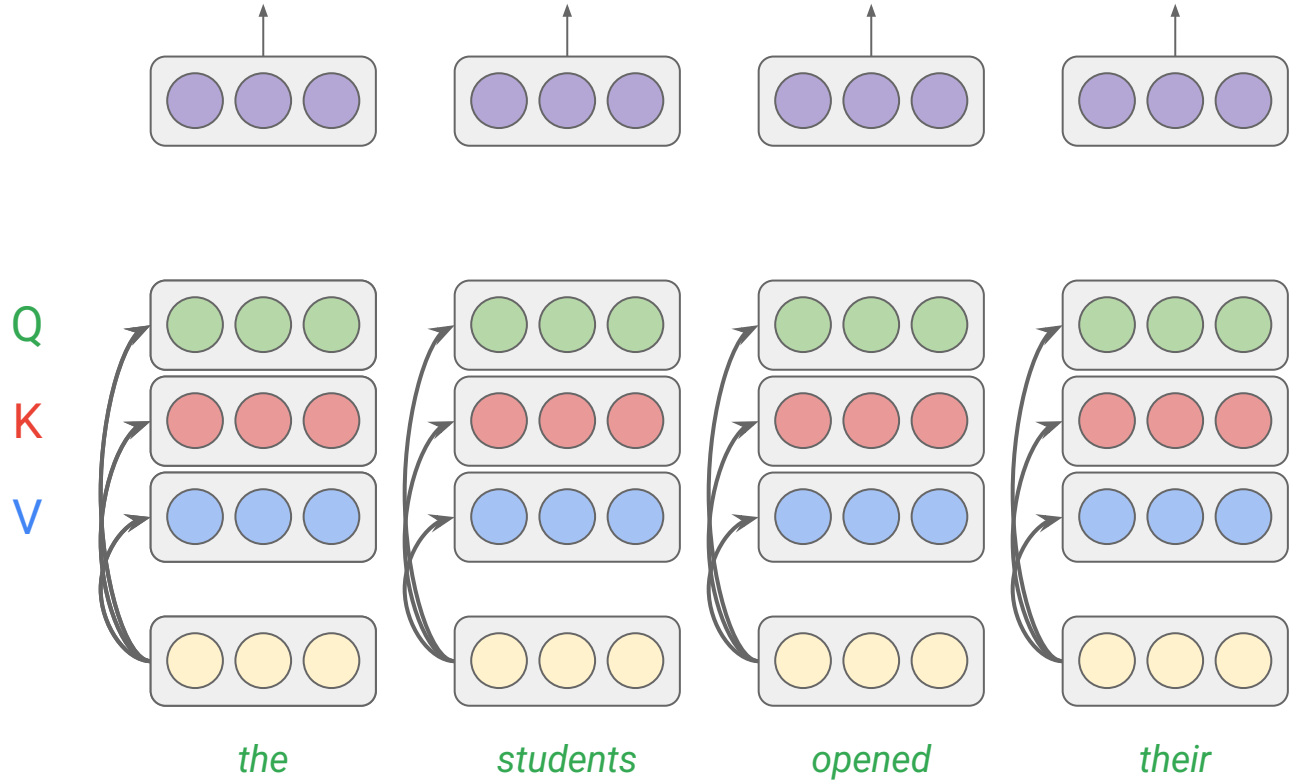
Value vectors



$$V = X \cdot W_V$$

linear projections

Attention (cont'd)



$$Q = X \cdot W_Q$$

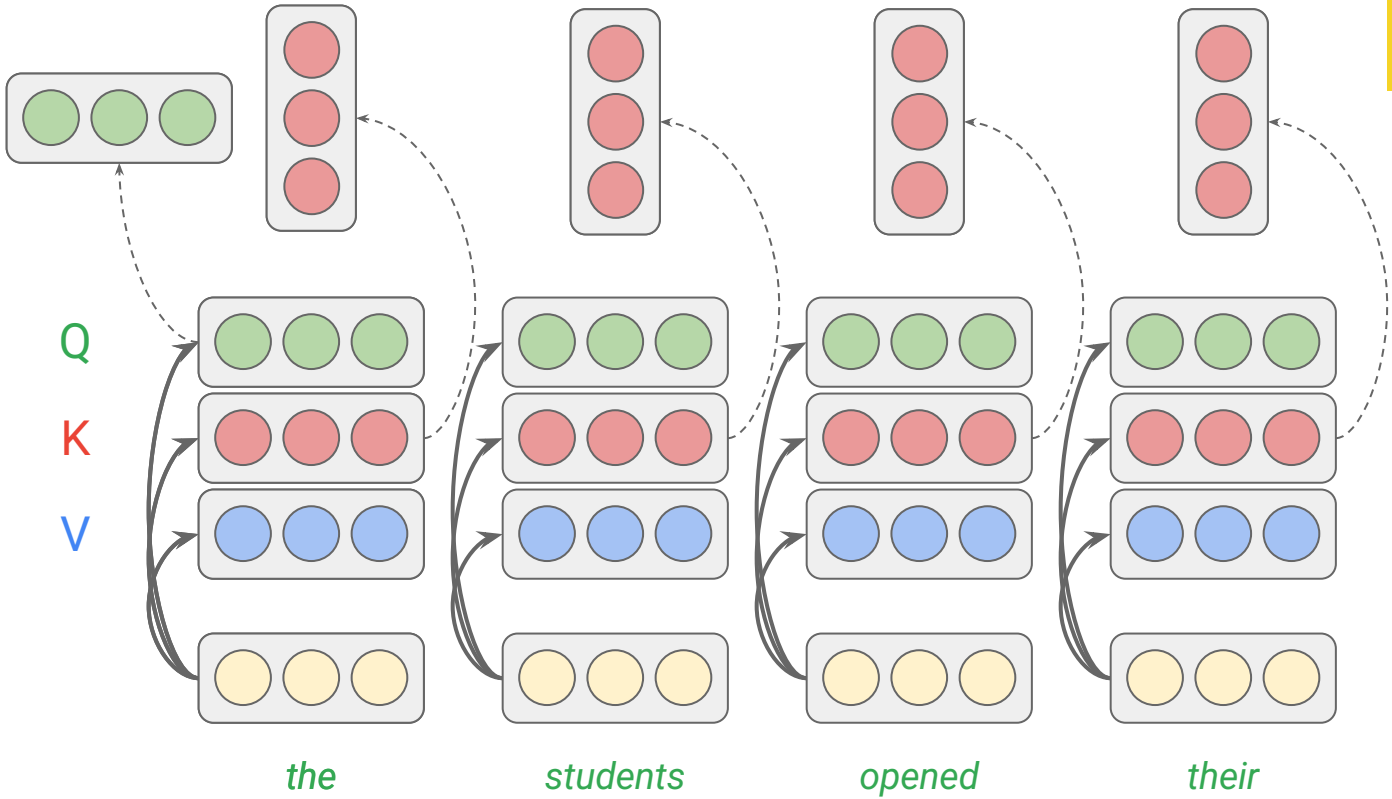
$$K = X \cdot W_K$$

$$V = X \cdot W_V$$

linear projections

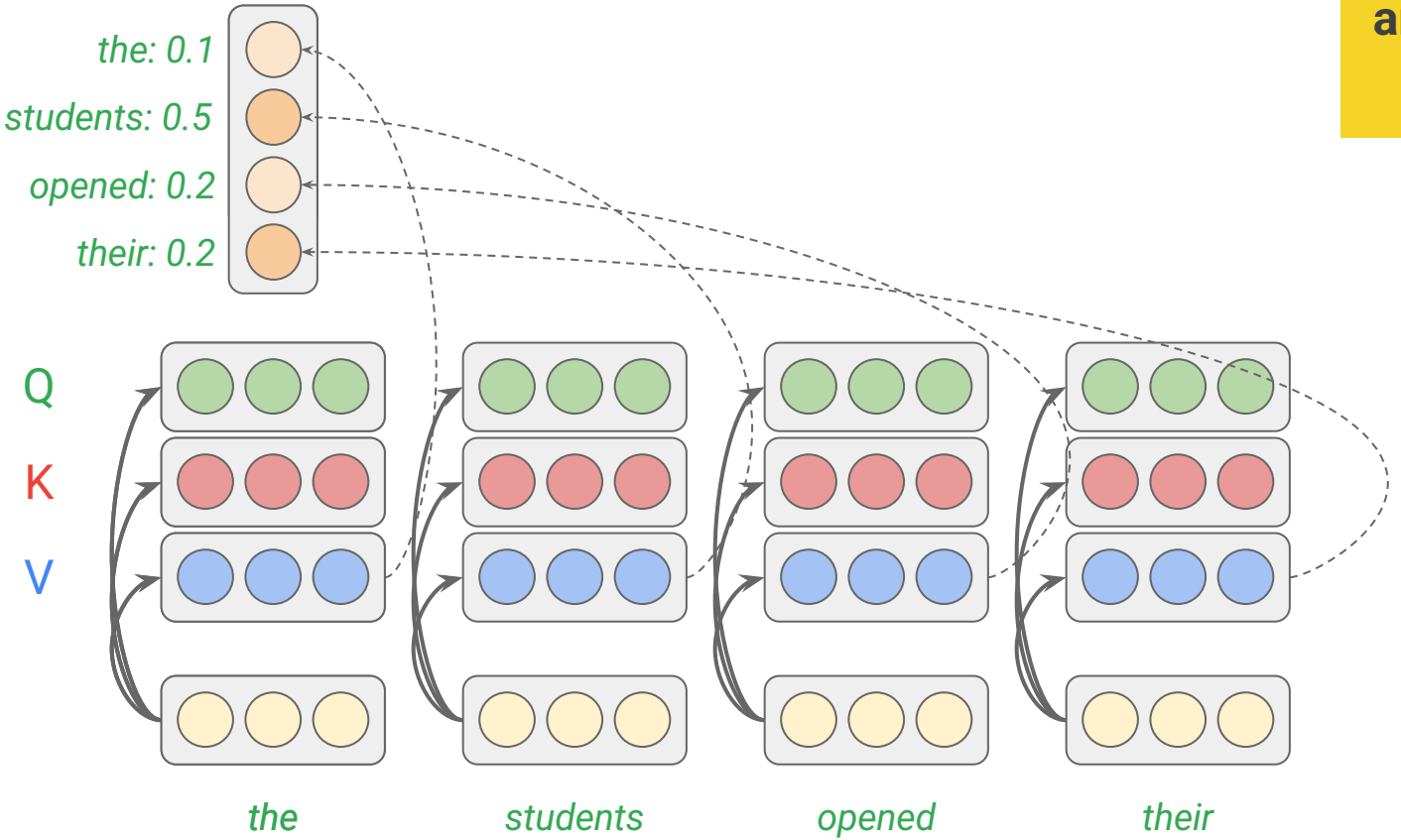
Self-attention (cont'd)

***all* computations are parallelized**



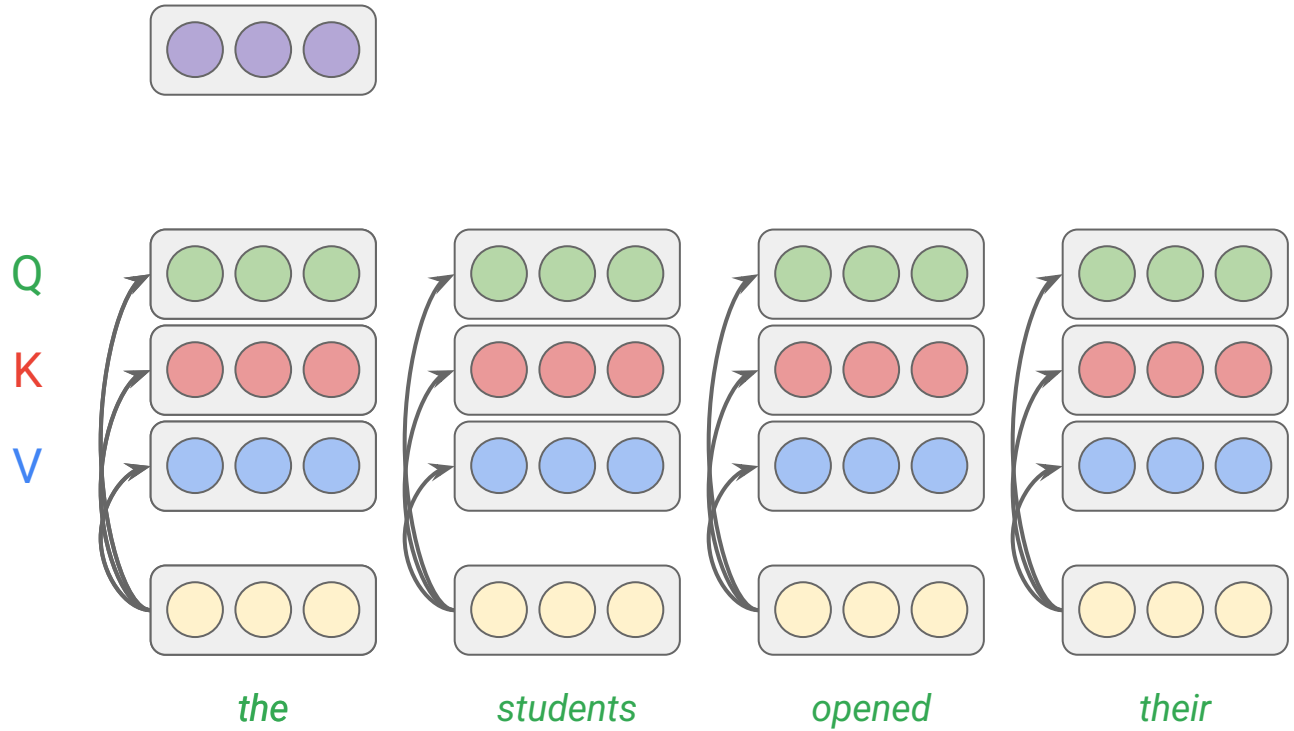
Self-attention (cont'd)

all computations are parallelized

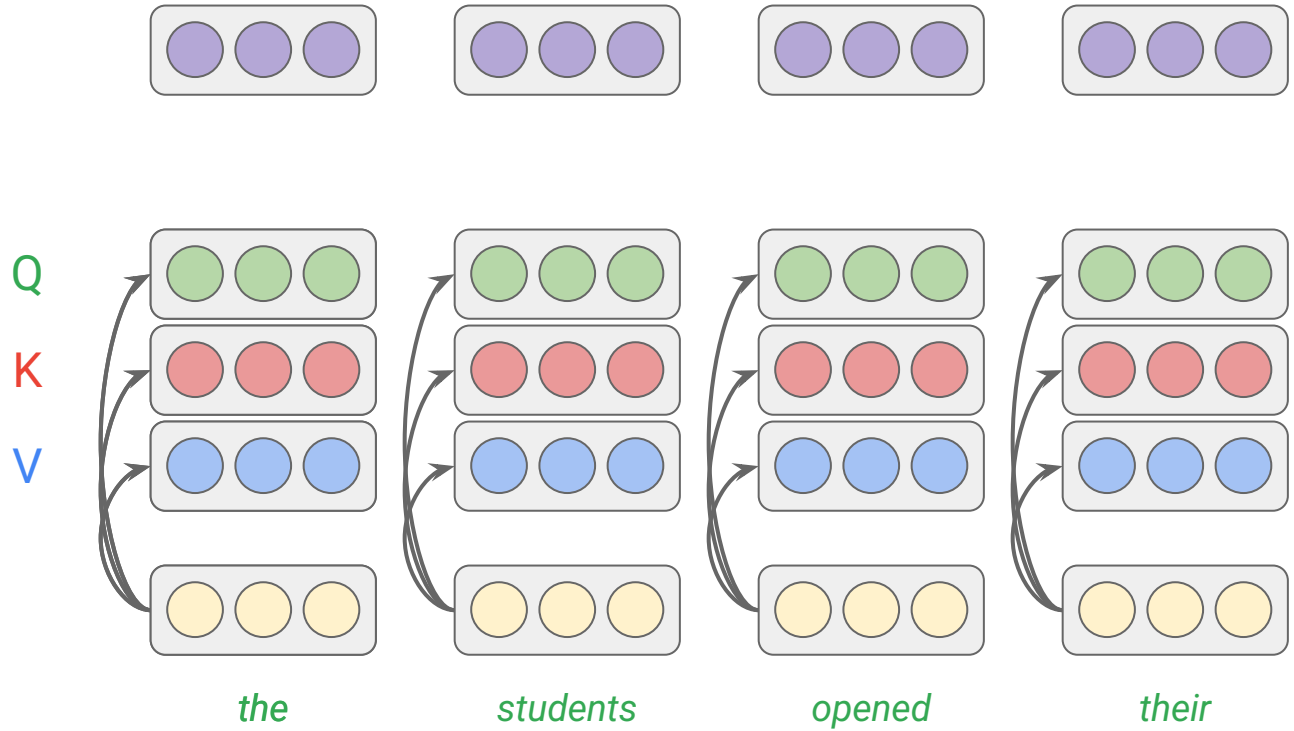


Self-attention (cont'd)

***all* computations are parallelized**

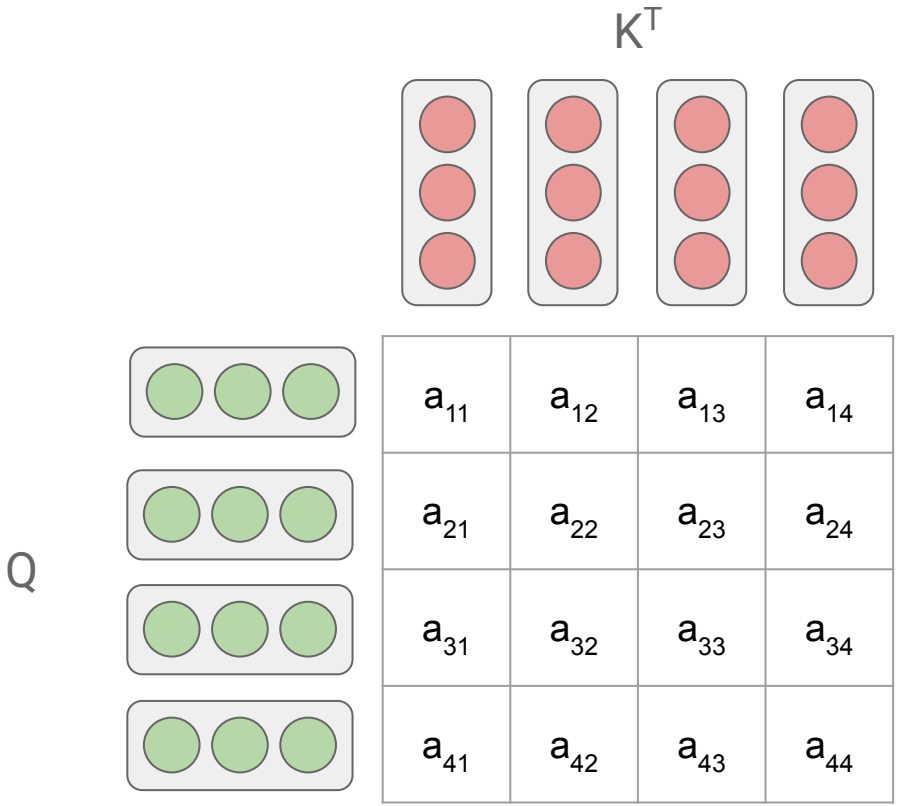


Self-attention (cont'd)



all **computations are parallelized during training and sequential during inference**

Quadratic complexity



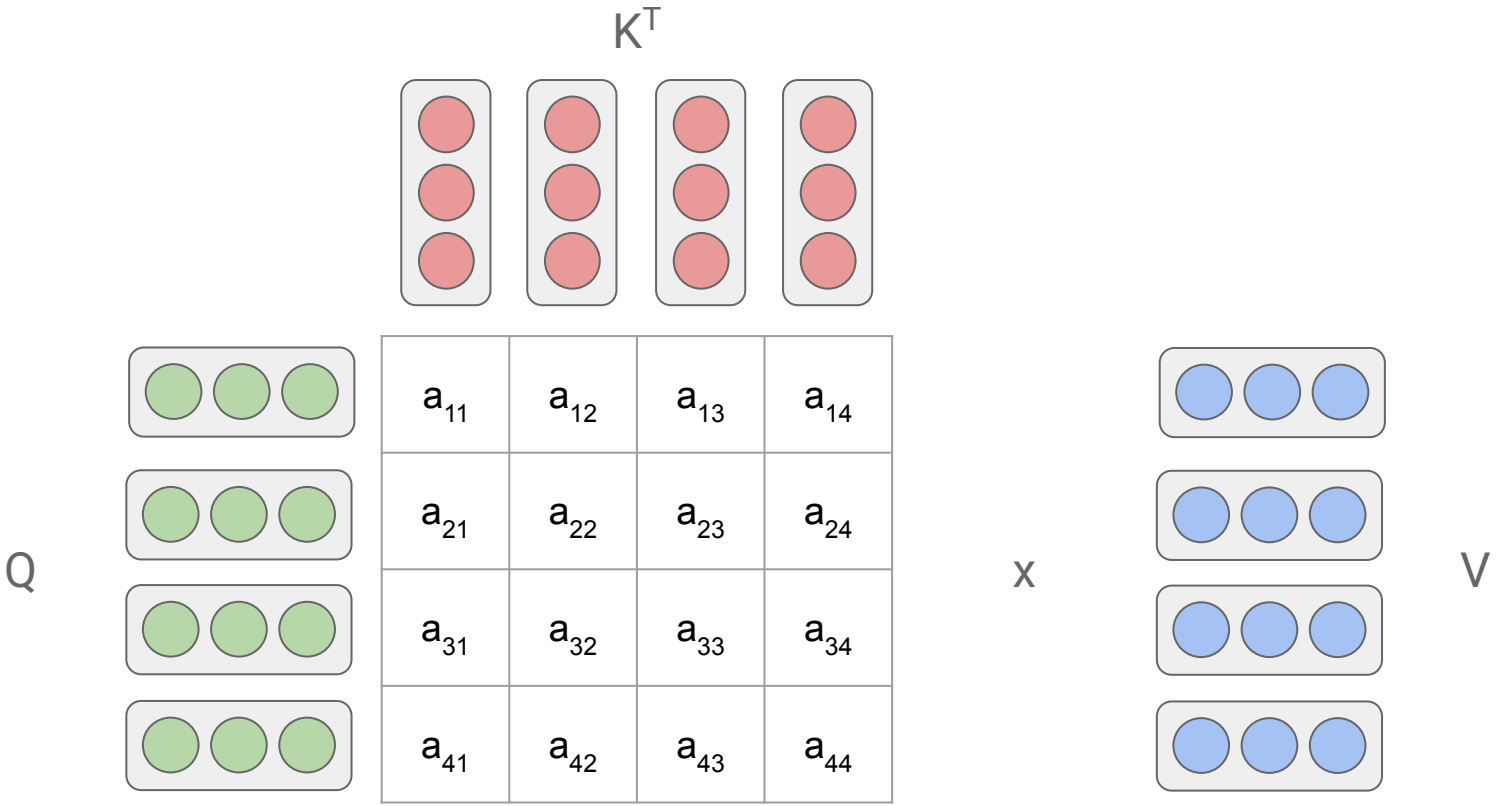
The time complexity of self-attention is quadratic in the input length $O(n^2)$

$$Q = \begin{pmatrix} q_{11} & q_{12} \\ q_{21} & q_{22} \\ q_{31} & q_{32} \end{pmatrix} = \begin{pmatrix} - & q_1 & - \\ - & q_2 & - \\ - & q_3 & - \end{pmatrix},$$

$$K = \begin{pmatrix} k_{11} & k_{12} \\ k_{21} & k_{22} \\ k_{31} & k_{32} \end{pmatrix} = \begin{pmatrix} - & k_1 & - \\ - & k_2 & - \\ - & k_3 & - \end{pmatrix}.$$

$$QK^\top = \begin{pmatrix} q_{11}k_{11} + q_{12}k_{12} & q_{11}k_{21} + q_{12}k_{22} & q_{11}k_{31} + q_{12}k_{32} \\ q_{21}k_{11} + q_{22}k_{12} & q_{21}k_{21} + q_{22}k_{22} & q_{21}k_{31} + q_{22}k_{32} \\ q_{31}k_{11} + q_{32}k_{12} & q_{31}k_{21} + q_{32}k_{22} & q_{31}k_{31} + q_{32}k_{32} \end{pmatrix}.$$

Quadratic complexity (cont'd)



Let

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix}, \quad V = \begin{bmatrix} v_{11} & v_{12} & v_{13} \\ v_{21} & v_{22} & v_{23} \\ v_{31} & v_{32} & v_{33} \\ v_{41} & v_{42} & v_{43} \end{bmatrix},$$

where the hidden dimension is 3.

Then $AV \in \mathbb{R}^{4\times 3}$ and expands to

$$AV = \begin{bmatrix} a_{11}v_{11} + a_{12}v_{21} + a_{13}v_{31} + a_{14}v_{41} & a_{11}v_{12} + a_{12}v_{22} + a_{13}v_{32} + a_{14}v_{42} & a_{11}v_{13} + a_{12}v_{23} + a_{13}v_{33} + a_{14}v_{43} \\ a_{21}v_{11} + a_{22}v_{21} + a_{23}v_{31} + a_{24}v_{41} & a_{21}v_{12} + a_{22}v_{22} + a_{23}v_{32} + a_{24}v_{42} & a_{21}v_{13} + a_{22}v_{23} + a_{23}v_{33} + a_{24}v_{43} \\ a_{31}v_{11} + a_{32}v_{21} + a_{33}v_{31} + a_{34}v_{41} & a_{31}v_{12} + a_{32}v_{22} + a_{33}v_{32} + a_{34}v_{42} & a_{31}v_{13} + a_{32}v_{23} + a_{33}v_{33} + a_{34}v_{43} \\ a_{41}v_{11} + a_{42}v_{21} + a_{43}v_{31} + a_{44}v_{41} & a_{41}v_{12} + a_{42}v_{22} + a_{43}v_{32} + a_{44}v_{42} & a_{41}v_{13} + a_{42}v_{23} + a_{43}v_{33} + a_{44}v_{43} \end{bmatrix}$$

Expanding the first row of AV,

$$(AV)_{1:} = \begin{bmatrix} a_{11}v_{11} + a_{12}v_{21} + a_{13}v_{31} + a_{14}v_{41},\ a_{11}v_{12} + a_{12}v_{22} + a_{13}v_{32} + a_{14}v_{42},\ a_{11}v_{13} + a_{12}v_{23} + a_{13}v_{33} + a_{14}v_{43} \end{bmatrix}$$

Rewrite this as a column vector,

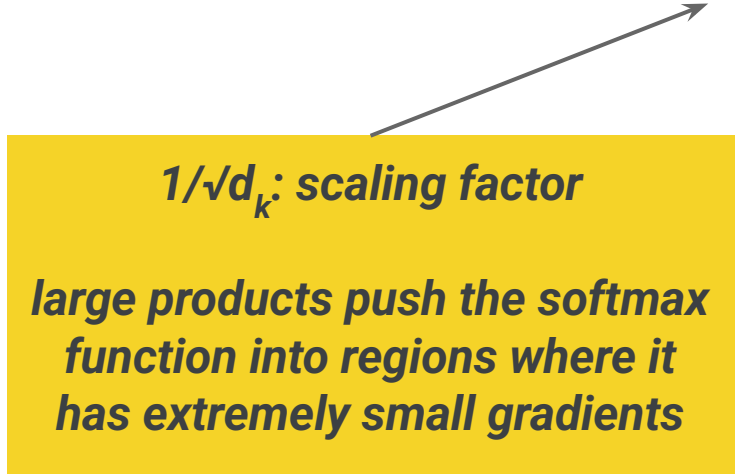
$$(AV)_{1:}^{\top} = \begin{bmatrix} a_{11}v_{11} + a_{12}v_{21} + a_{13}v_{31} + a_{14}v_{41} \\ a_{11}v_{12} + a_{12}v_{22} + a_{13}v_{32} + a_{14}v_{42} \\ a_{11}v_{13} + a_{12}v_{23} + a_{13}v_{33} + a_{14}v_{43} \end{bmatrix}.$$

Factor by coefficients,

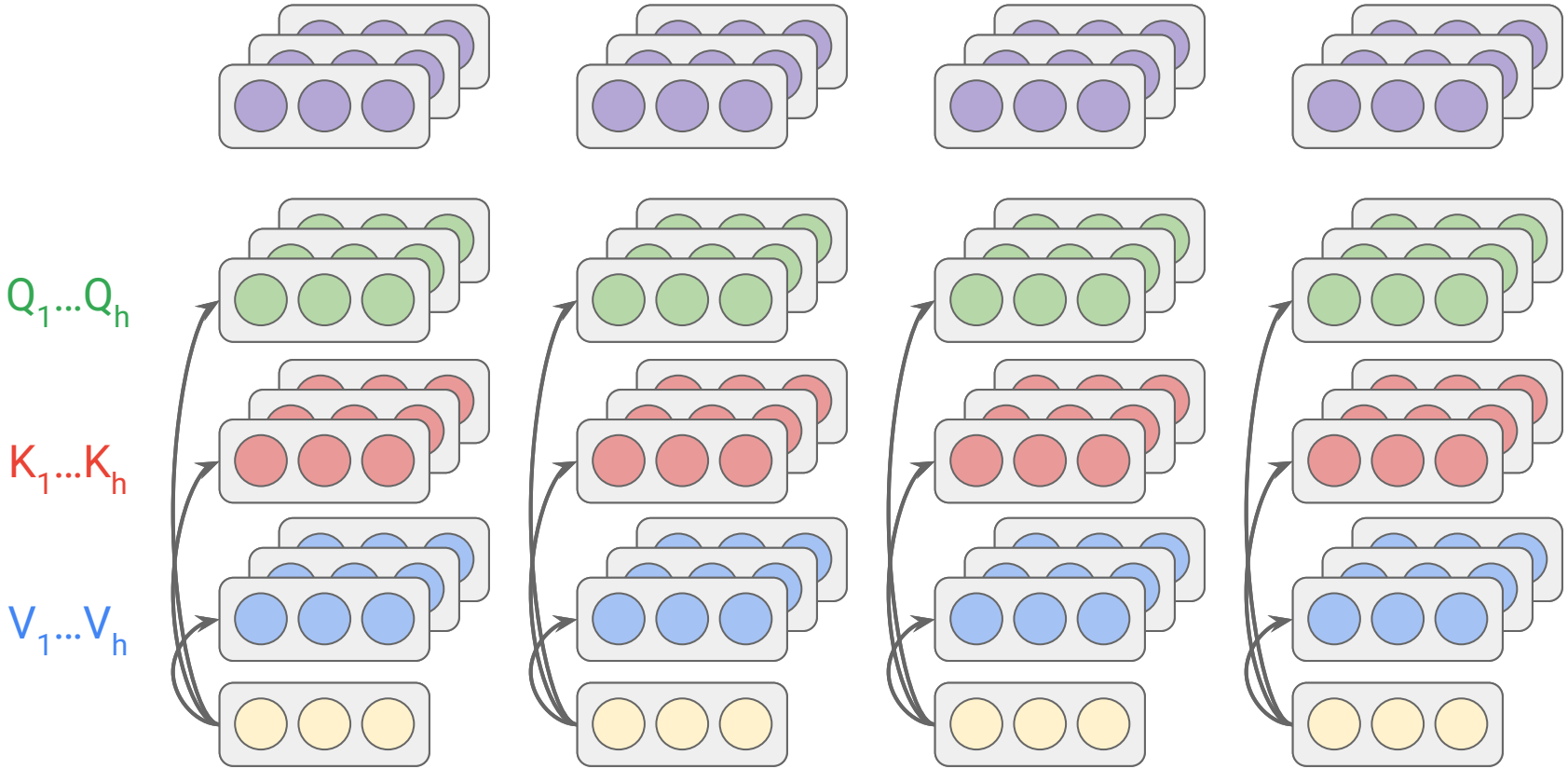
$$(AV)_{1:}^{\top} = a_{11} \begin{bmatrix} v_{11} \\ v_{12} \\ v_{13} \end{bmatrix} + a_{12} \begin{bmatrix} v_{21} \\ v_{22} \\ v_{23} \end{bmatrix} + a_{13} \begin{bmatrix} v_{31} \\ v_{32} \\ v_{33} \end{bmatrix} + a_{14} \begin{bmatrix} v_{41} \\ v_{42} \\ v_{43} \end{bmatrix}.$$

All computations are parallelized

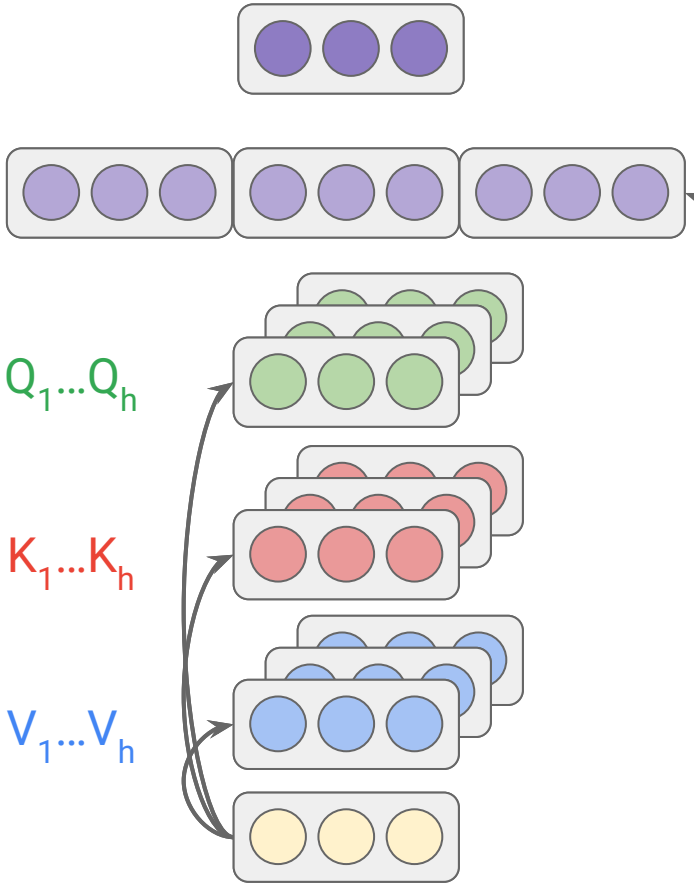
$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$



Multi-head attention

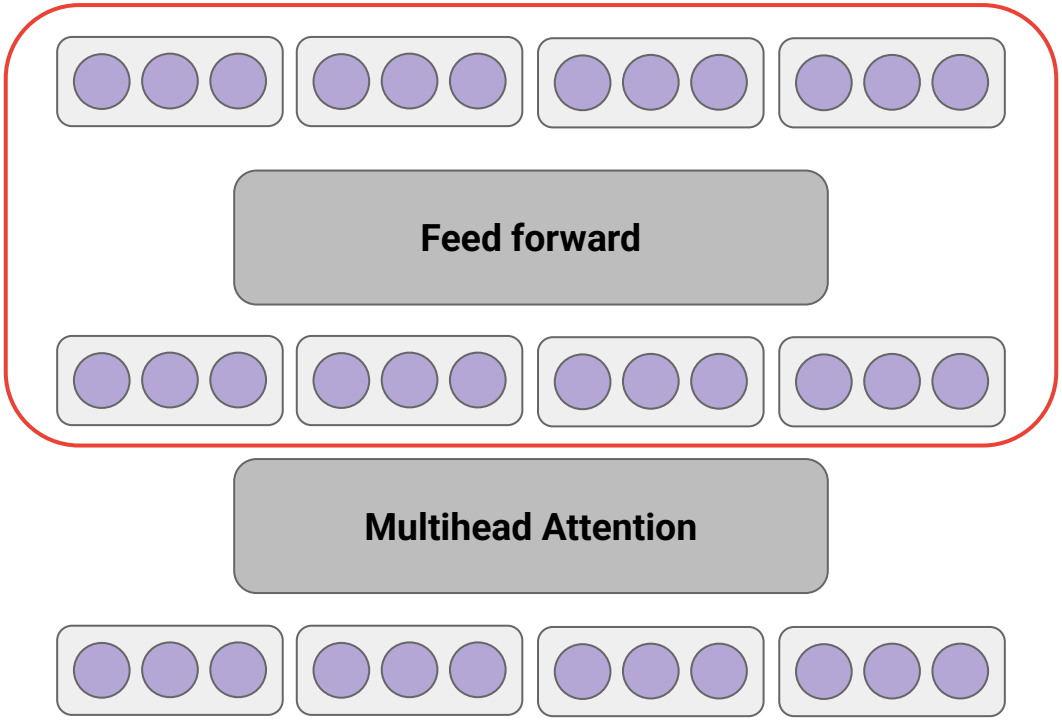


Multi-head attention (cont'd)

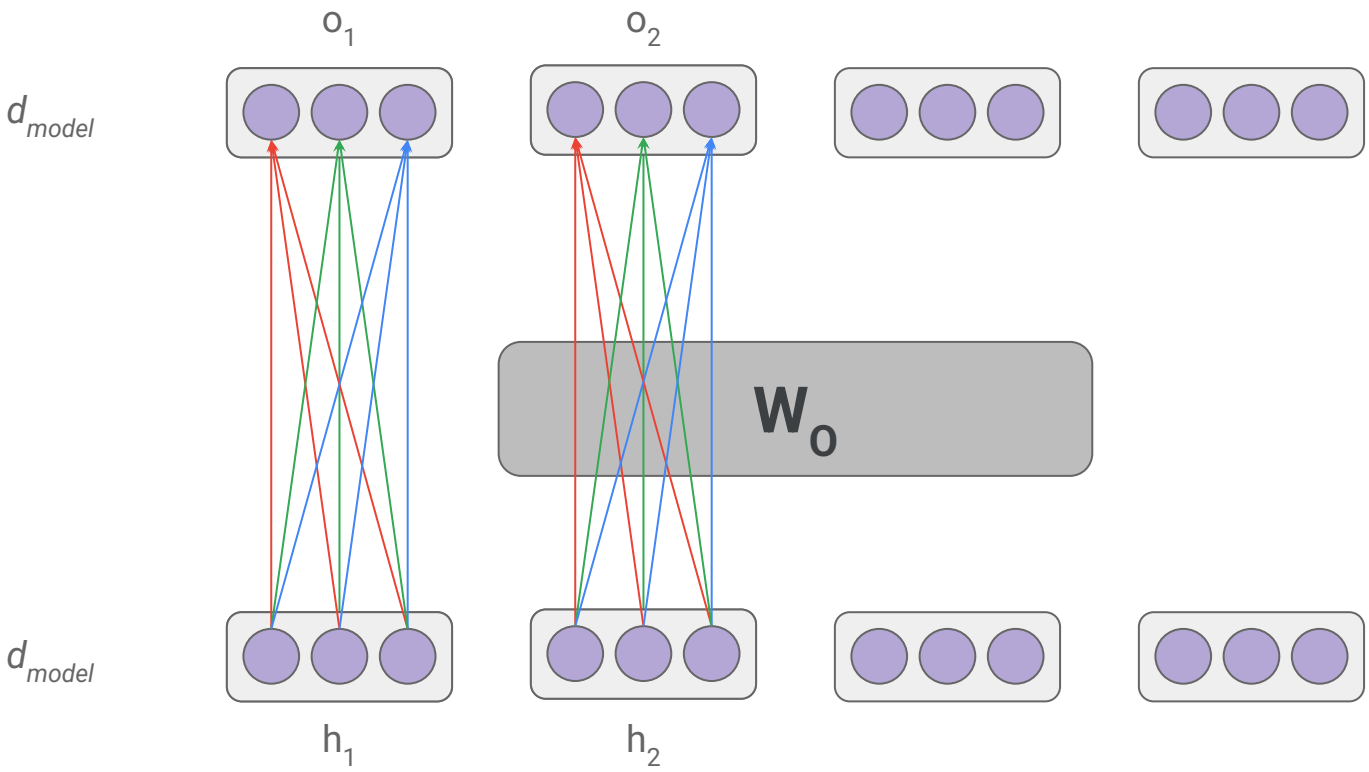


These output values are concatenated and once again projected

Transformer block (cont'd)



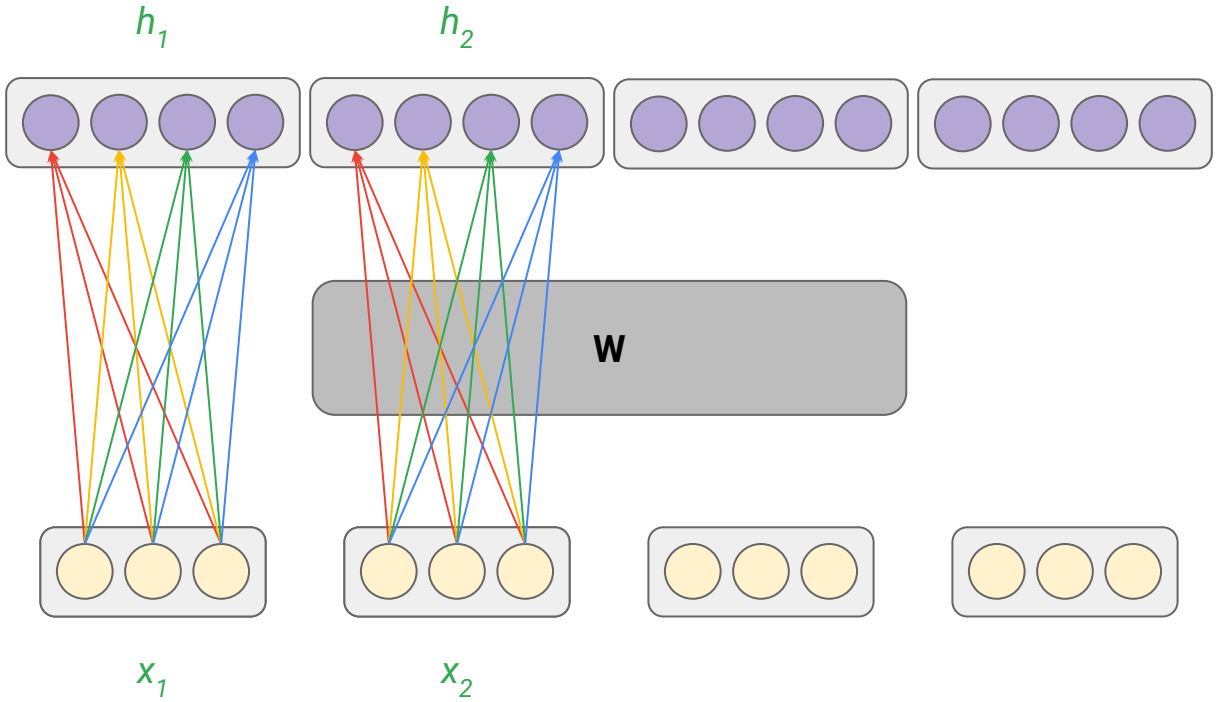
output vectors



$$O = H \cdot W_O$$

linear projections

Position-wise feedforward networks



We multiply the weight matrix W (size 4×3) with the embeddings matrix X (size 3×2):

$$H = WX$$

Performing the multiplication:

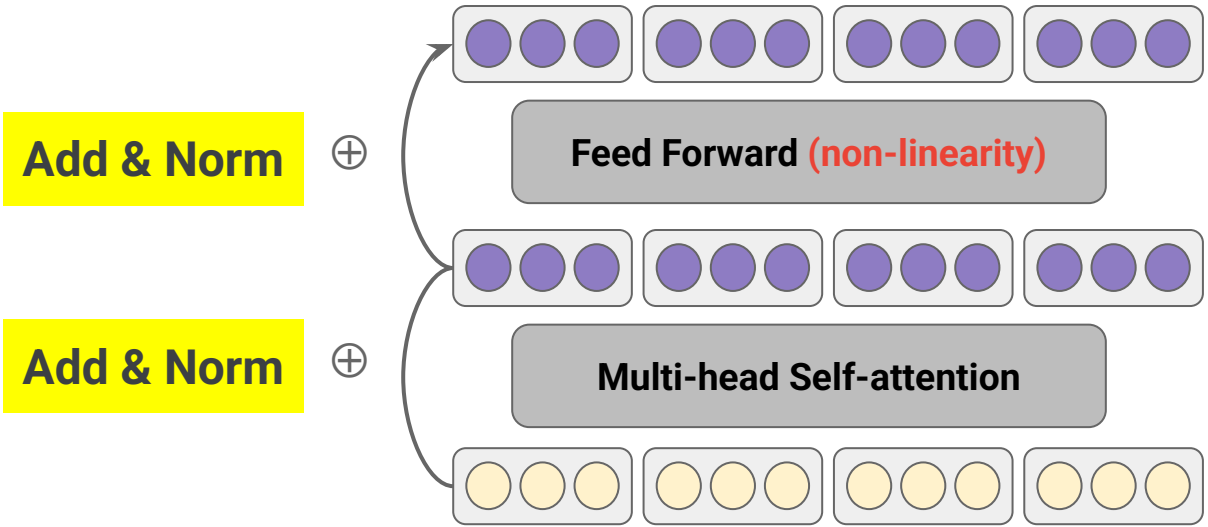
$$H = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \\ w_{41} & w_{42} & w_{43} \end{bmatrix} \begin{bmatrix} x_{11} & x_{21} \\ x_{12} & x_{22} \\ x_{13} & x_{23} \end{bmatrix}$$

This results in:

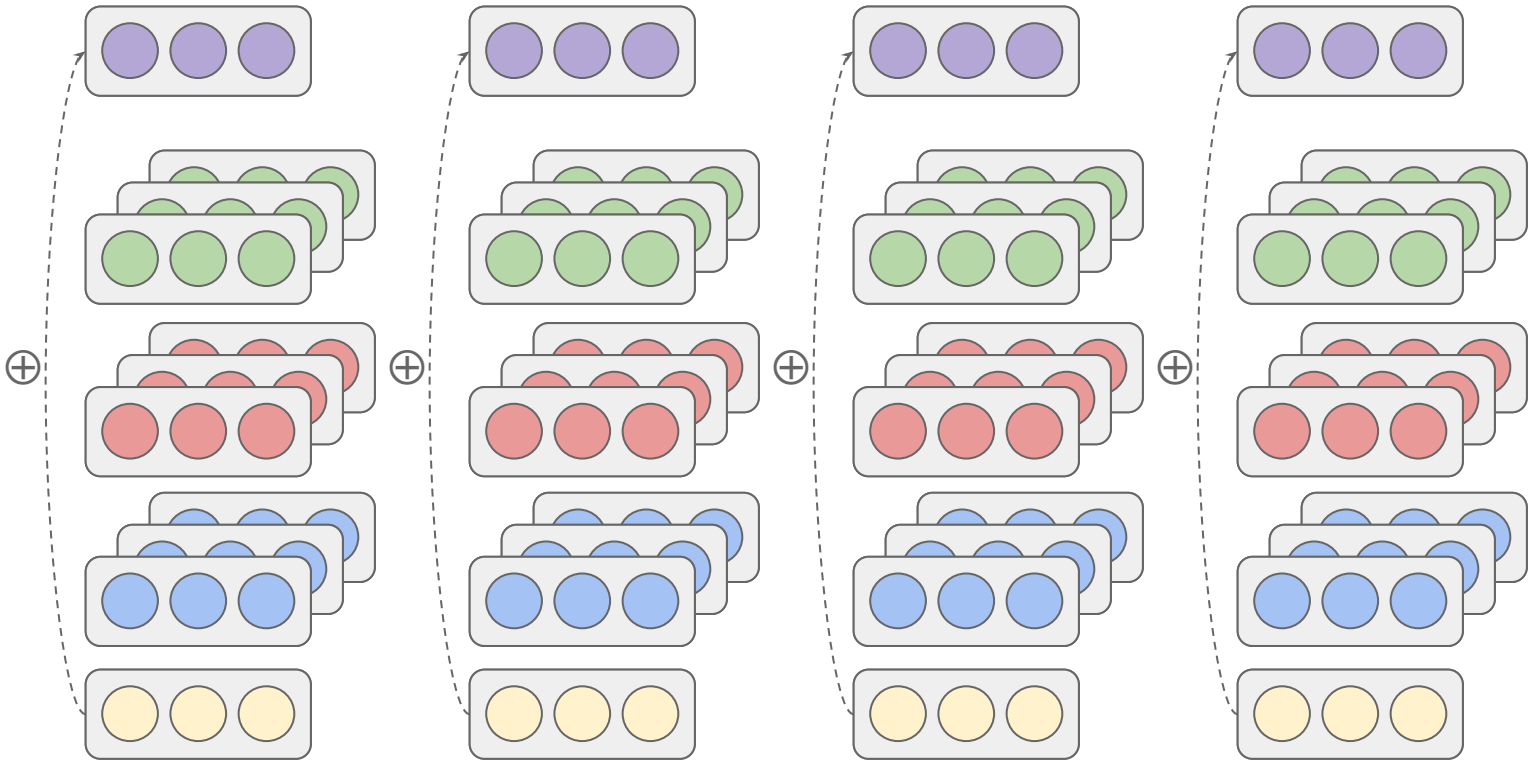
h_1 h_2

$$H = \begin{bmatrix} w_{11}x_{11} + w_{12}x_{12} + w_{13}x_{13} & w_{11}x_{21} + w_{12}x_{22} + w_{13}x_{23} \\ w_{21}x_{11} + w_{22}x_{12} + w_{23}x_{13} & w_{21}x_{21} + w_{22}x_{22} + w_{23}x_{23} \\ w_{31}x_{11} + w_{32}x_{12} + w_{33}x_{13} & w_{31}x_{21} + w_{32}x_{22} + w_{33}x_{23} \\ w_{41}x_{11} + w_{42}x_{12} + w_{43}x_{13} & w_{41}x_{21} + w_{42}x_{22} + w_{43}x_{23} \end{bmatrix}$$

Transformer block (one layer)



Residual connection



Residual connection

$$\text{output} = \text{sublayer}(x) + x$$

Layer normalization

$$\text{Norm}(z) = \frac{z - \mu}{\sigma} \cdot \gamma + \beta$$

where μ and σ are the mean and standard deviation of the activations, and γ, β are learnable parameters.

Each activation vector is normalized so that its components have mean 0 and variance 1. This prevents activations from becoming too large or too small as they propagate through the network.

Residual connection and layer normalization

$$\text{LayerNorm}(x + \text{Sublayer}(x))$$

Position-wise Feed-Forward Networks

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

ReLU (Rectified Linear Unit)

Sinusoidal positional encoding

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

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

- Sequence: **“I like cats”**
- Positions: $0, 1, 2$
- Embedding dimension: $d_{\text{model}} = 4$

For each position, we need **4 positional-encoding numbers**, corresponding to dimensions $0, 1, 2, 3$.

For $d_{\text{model}} = 4$, your equations become:

$$PE(pos, 0) = \sin(pos)$$

$$PE(pos, 1) = \cos(pos)$$

$$PE(pos, 2) = \sin(pos/100)$$

$$PE(pos, 3) = \cos(pos/100)$$

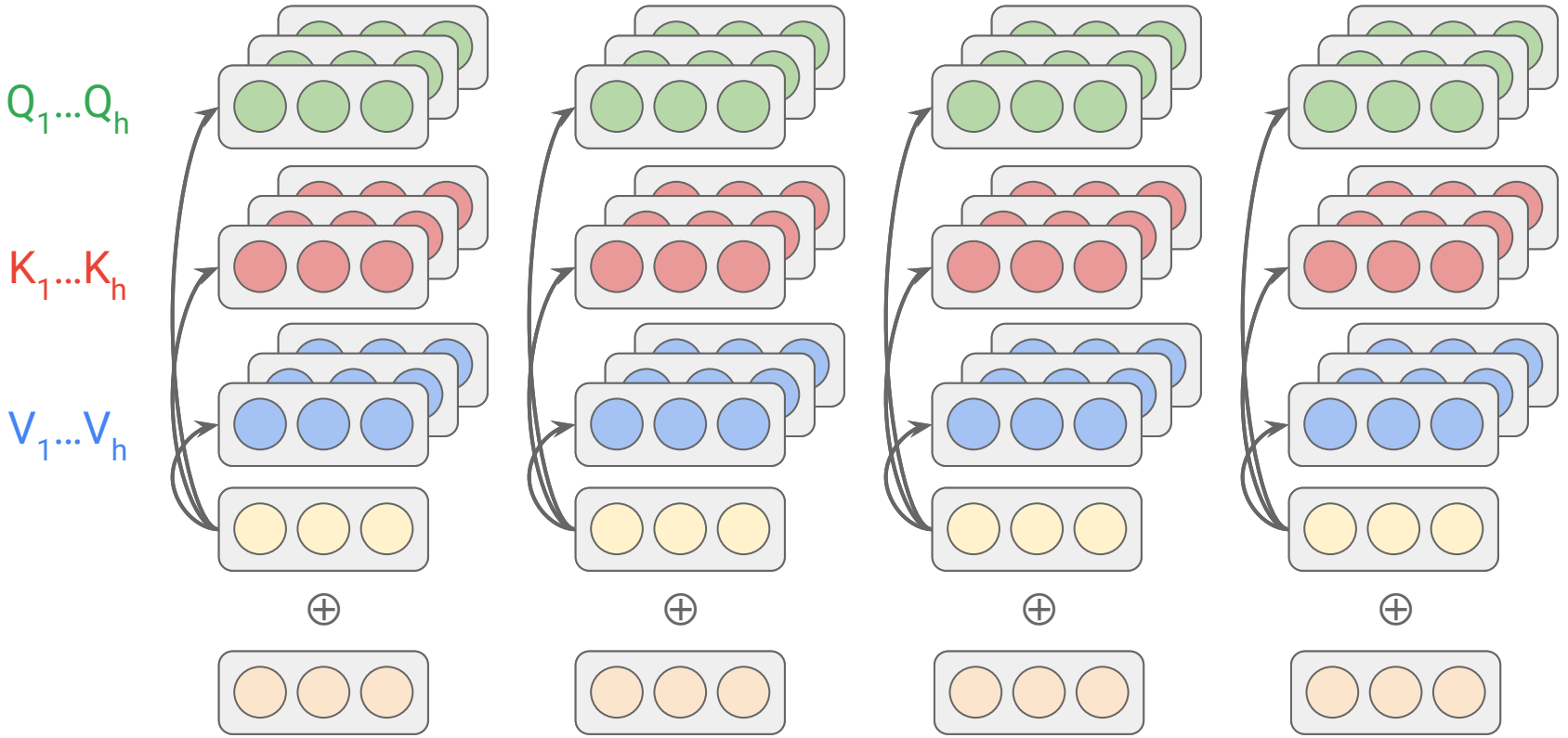
Token	pos	dim 0	dim 1	dim 2	dim 3
I	0	0.000	1.000	0.000	1.000
like	1	0.841	0.540	0.010	1.000
cats	2	0.909	-0.416	0.020	1.000

For example, for **"like"**, which is at $pos = 1$:

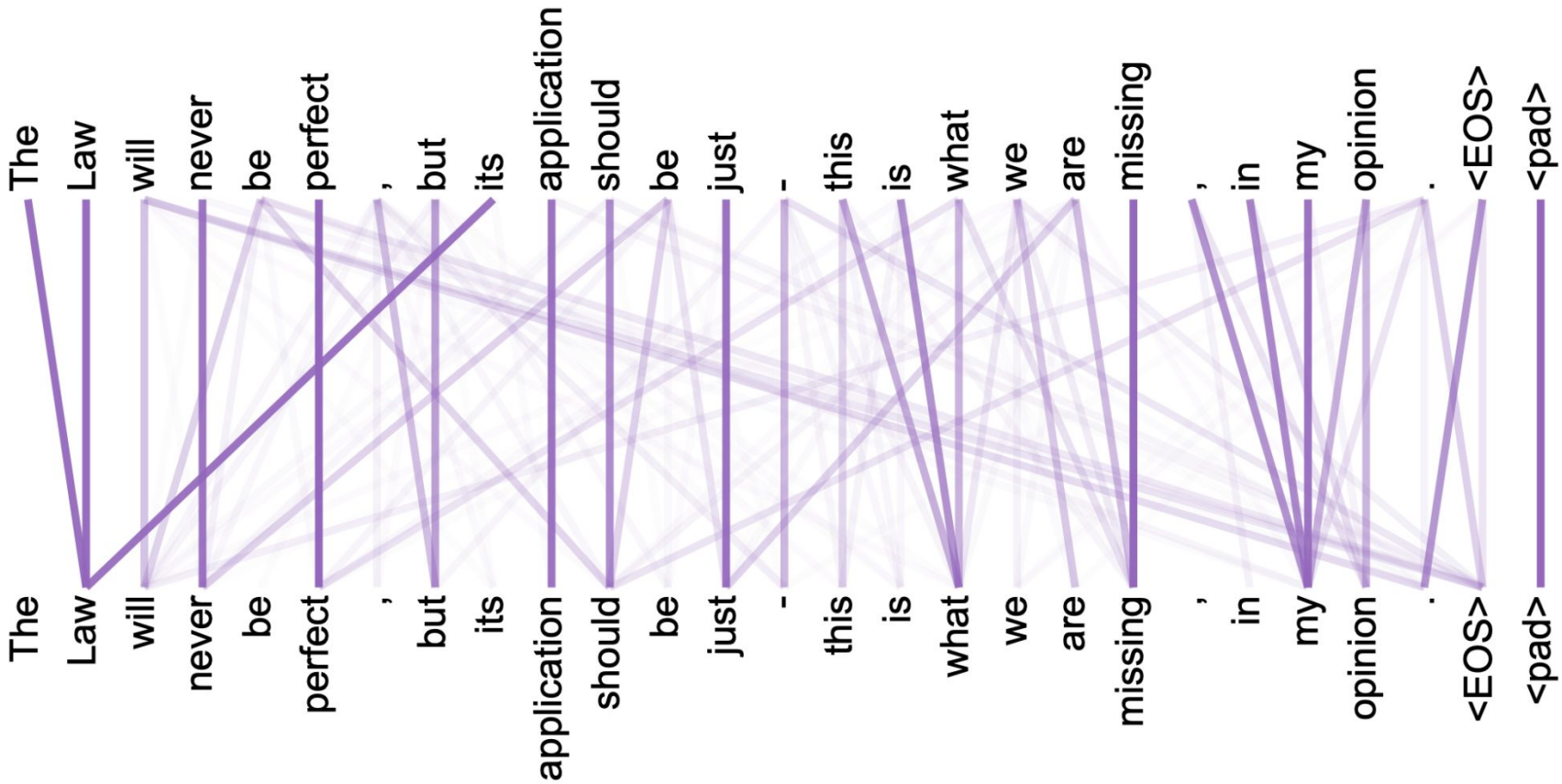
$$PE(1) = [\sin(1), \cos(1), \sin(1/100), \cos(1/100)]$$

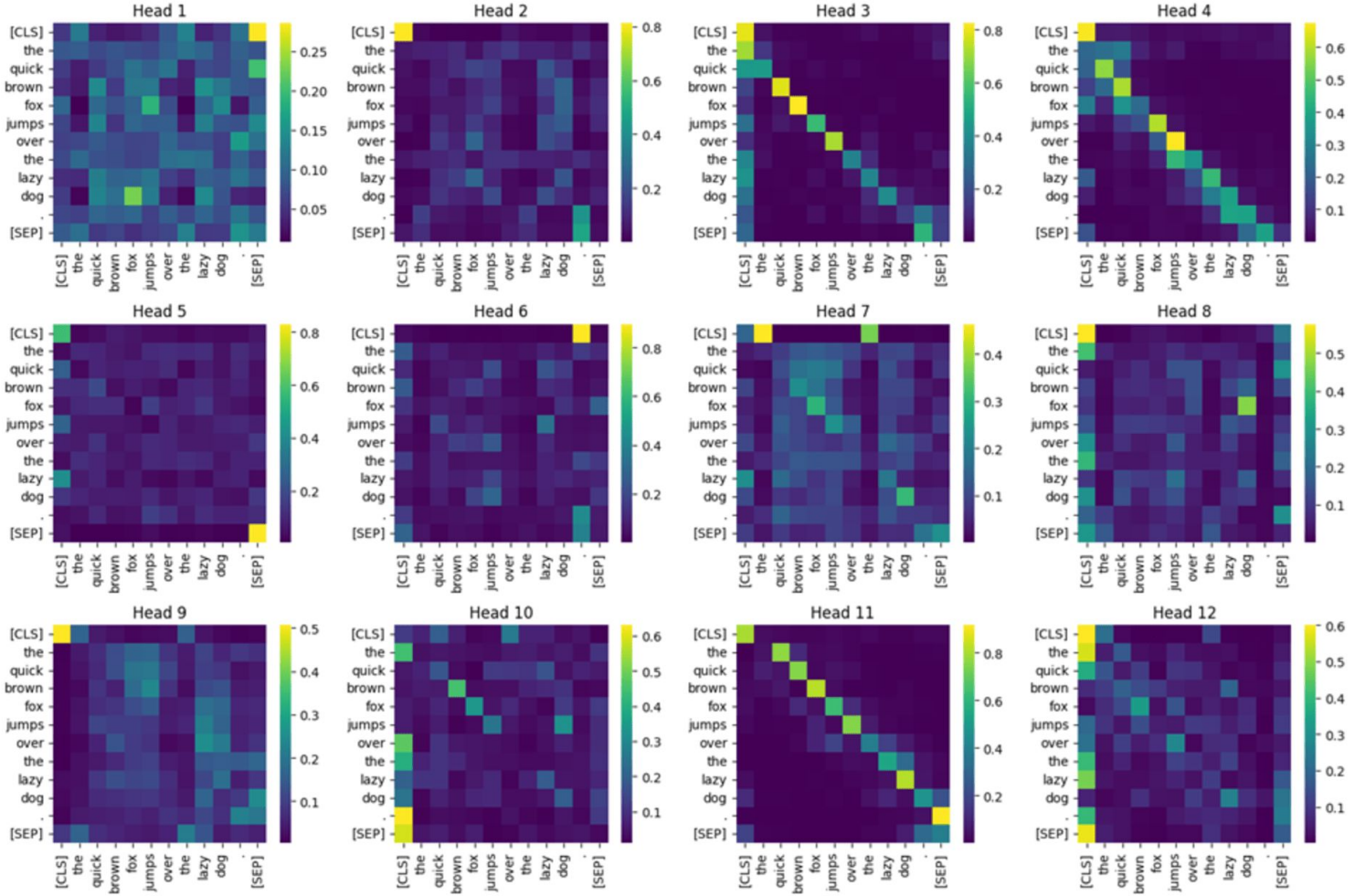
$$\approx [0.841,\ 0.540,\ 0.010,\ 1.000].$$

Positional Encoding (cont'd)



Attention visualizations



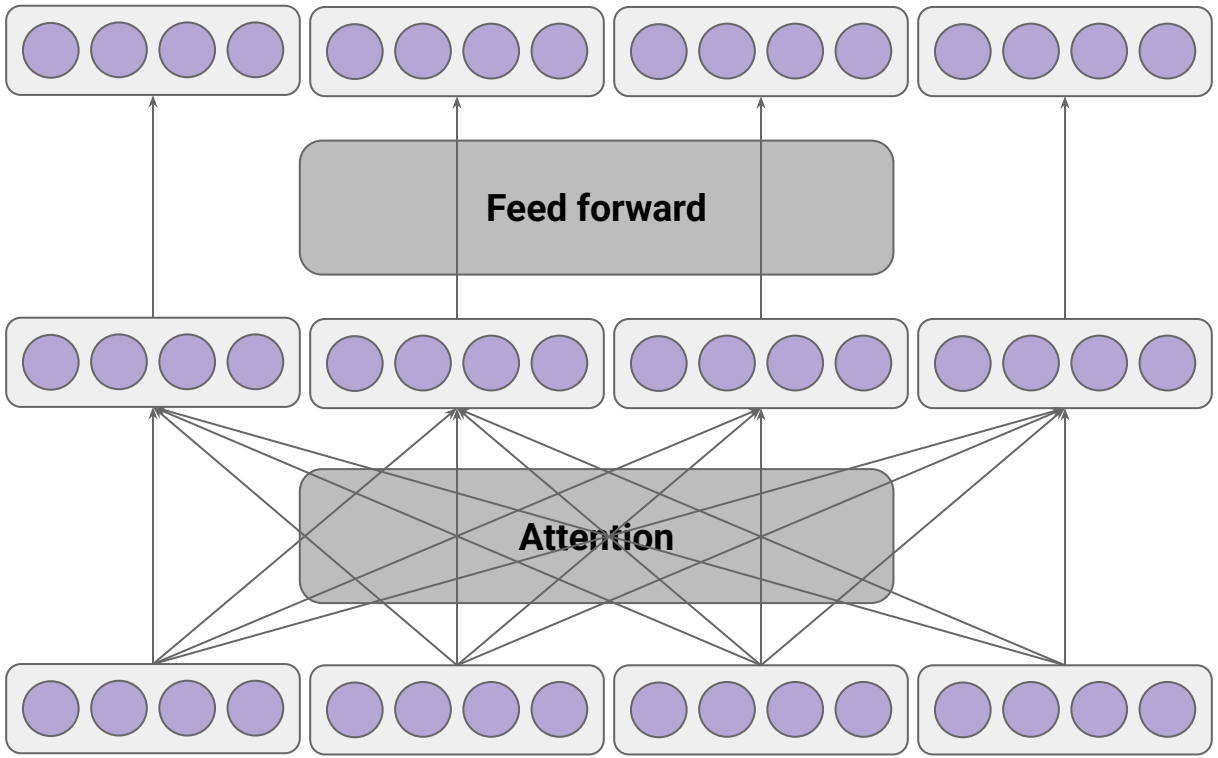


https://www.kdnuggets.com/how-to-visualize-model-internals-and-attention-in-hugging-face-transformers

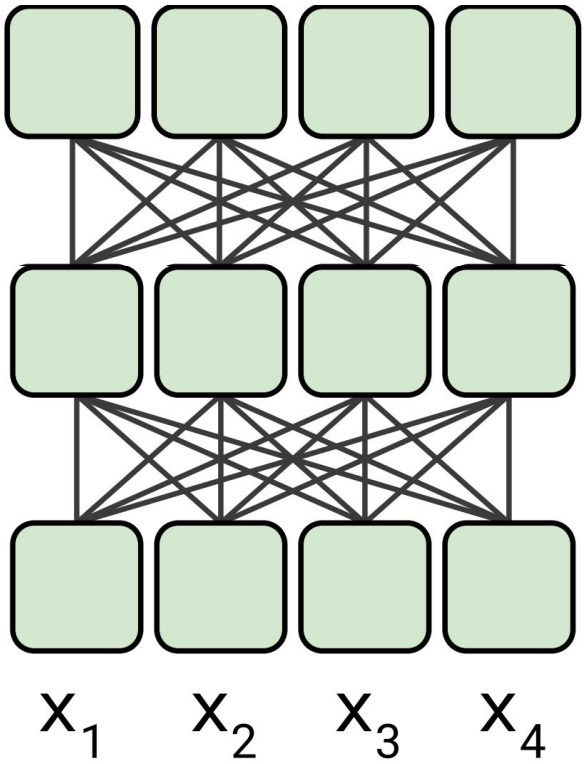
Attention visualizations (cont’d)

https://github.com/jessevig/bertviz

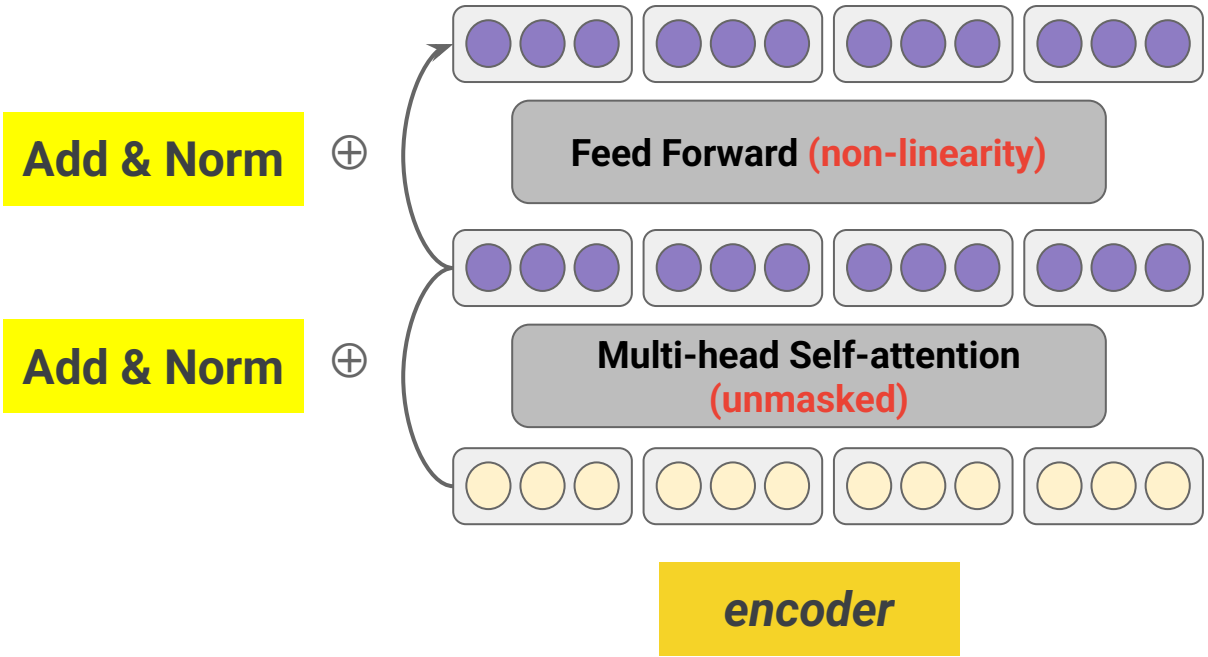
Putting it all together



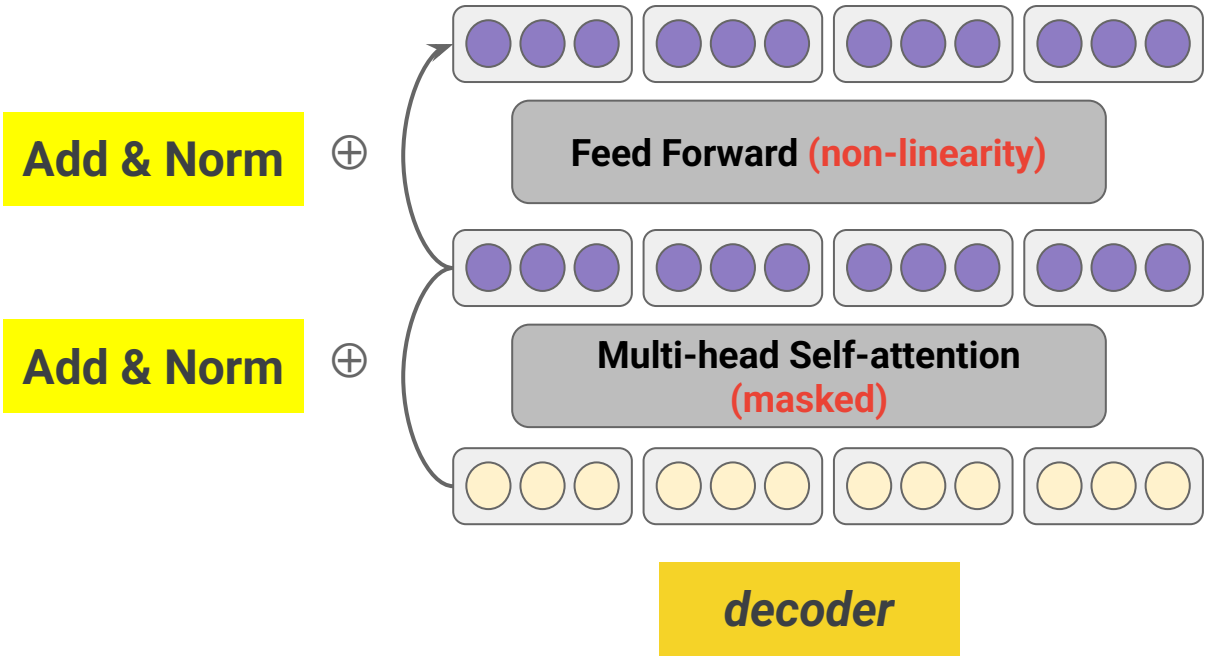
Encoder only



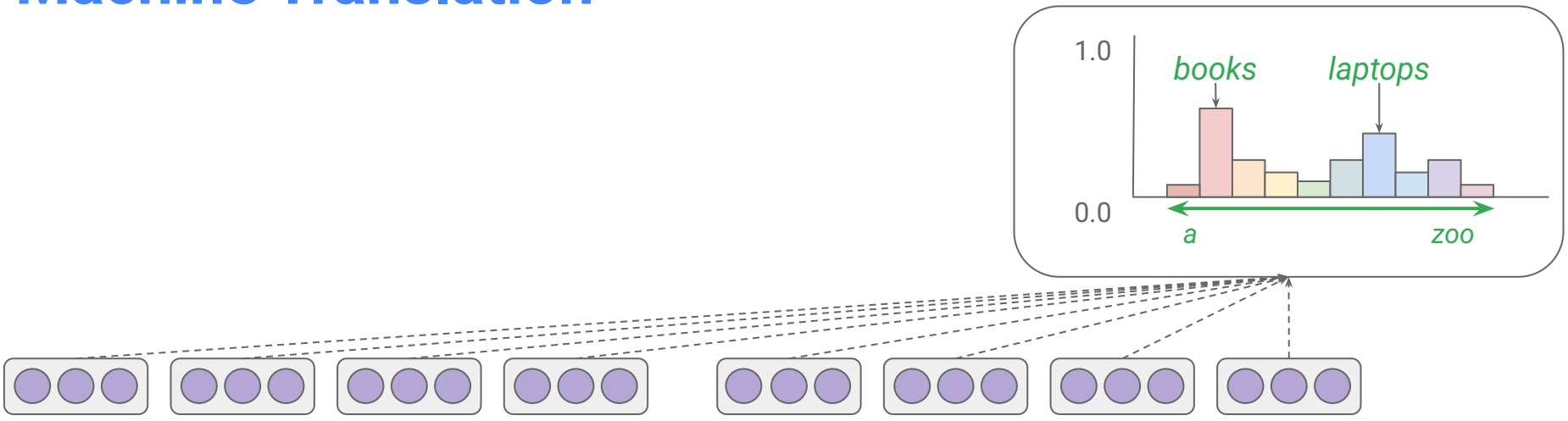
Encoder (one layer)



Decoder (one layer)



Machine Translation

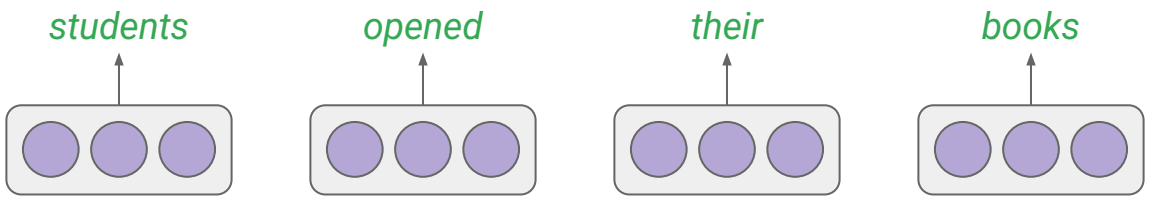


les étudiants ont ouvert leurs livres

the *students* *opened* *their*

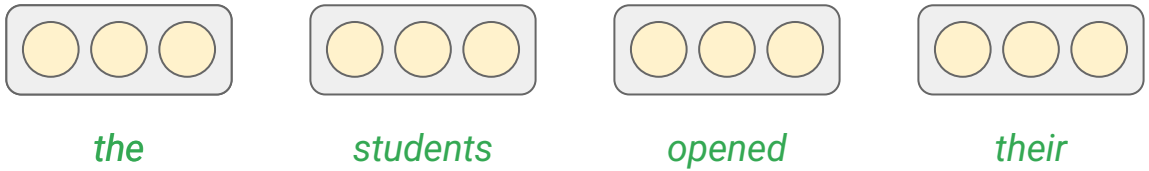
encoder ***decoder***

Transformer decoder

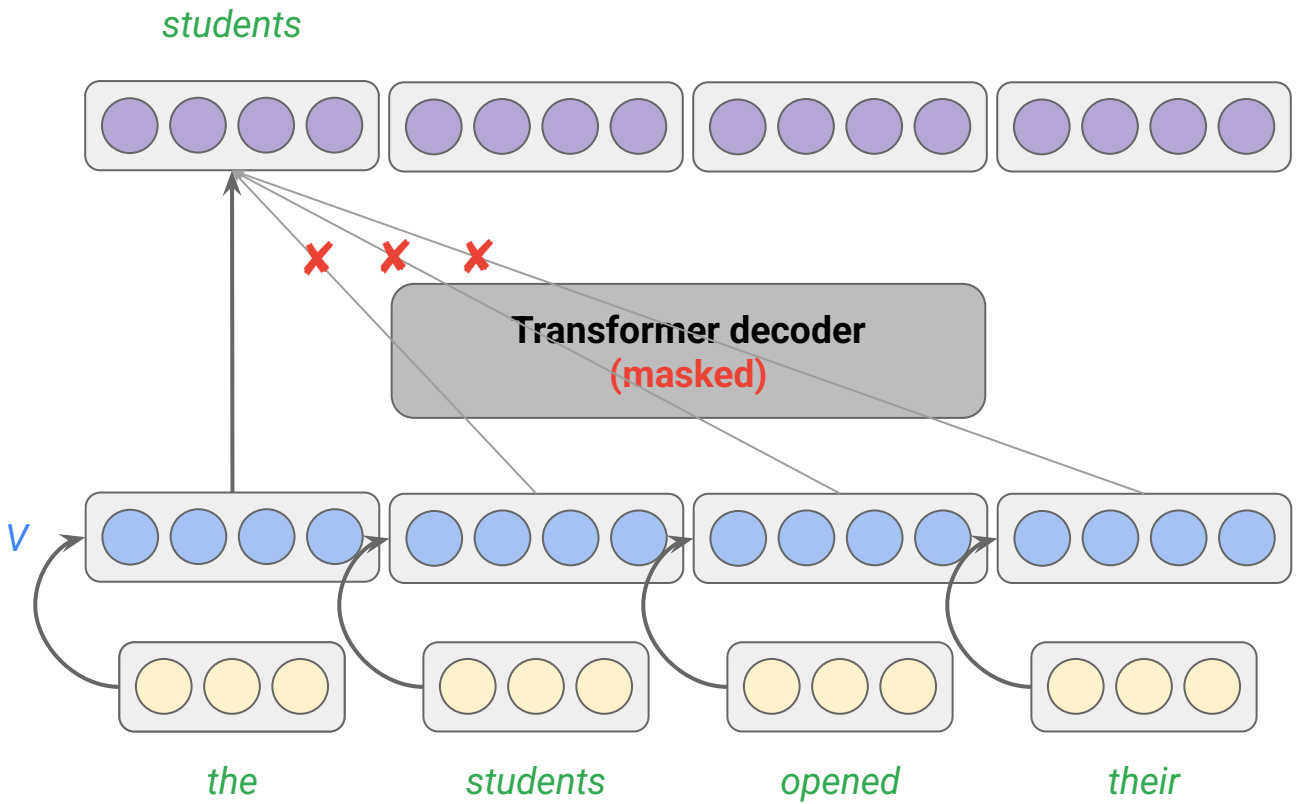


the architecture used in frontier LLMs

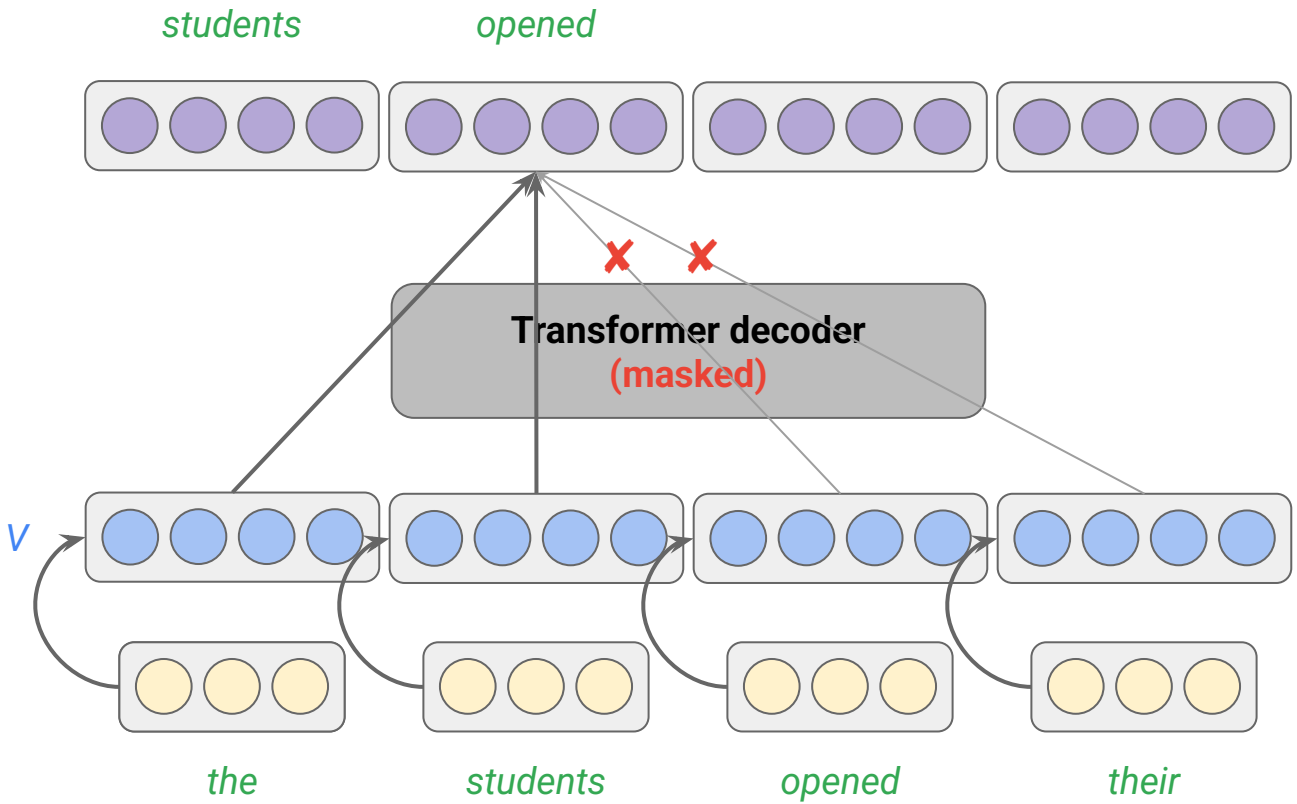
Transformer decoder
(masked)



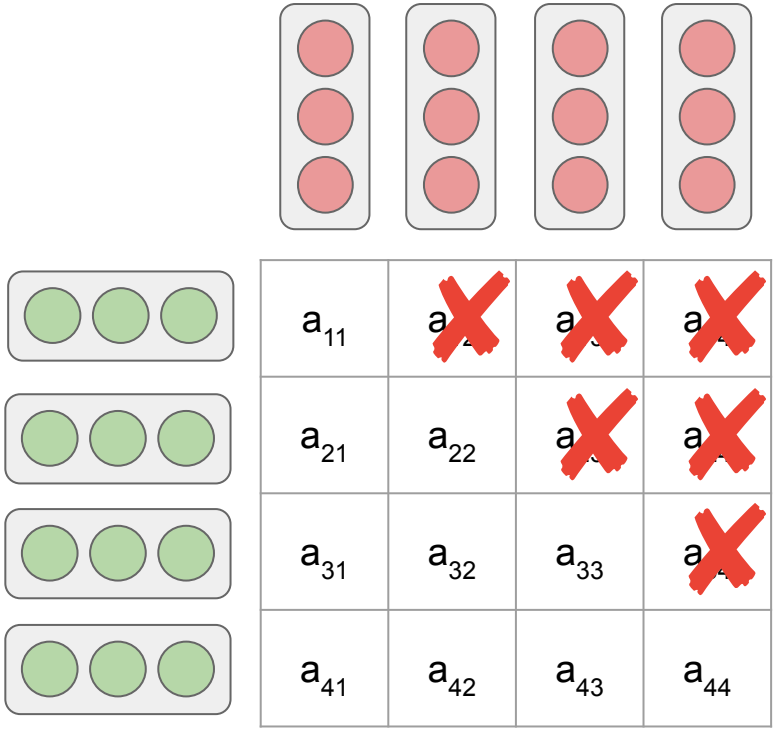
Transformer decoder (cont'd)



Transformer decoder (cont'd)

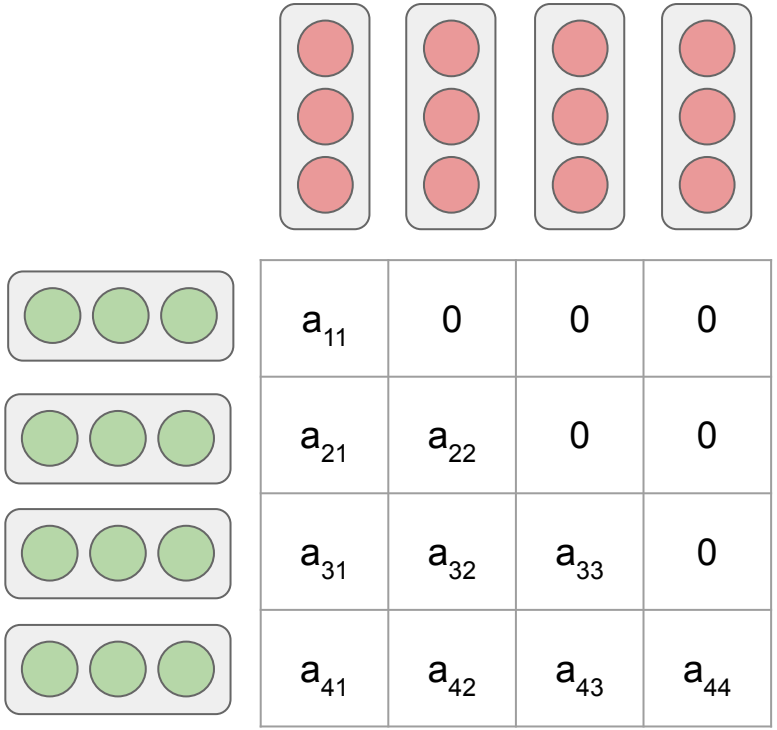


Self-attention in the decoder



masking out all values in the input of the softmax which correspond to illegal connections

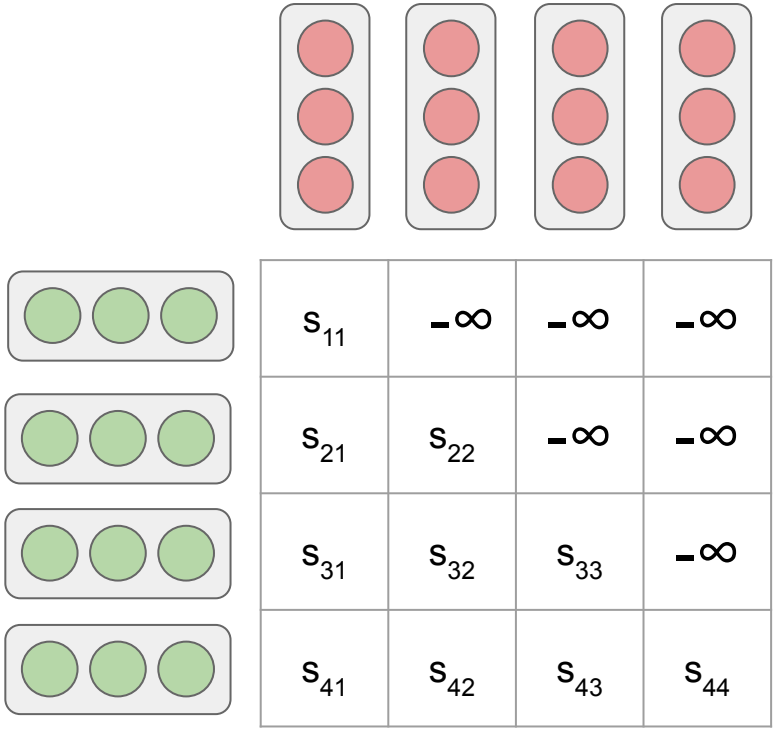
Self-attention in the decoder (cont'd)



a_{11}	0	0	0
a_{21}	a_{22}	0	0
a_{31}	a_{32}	a_{33}	0
a_{41}	a_{42}	a_{43}	a_{44}

masking out all values in the input of the softmax which correspond to illegal connections

Self-attention in the decoder (cont'd)



masking out (setting to $-\infty$) all values in the input of the softmax which correspond to illegal connections

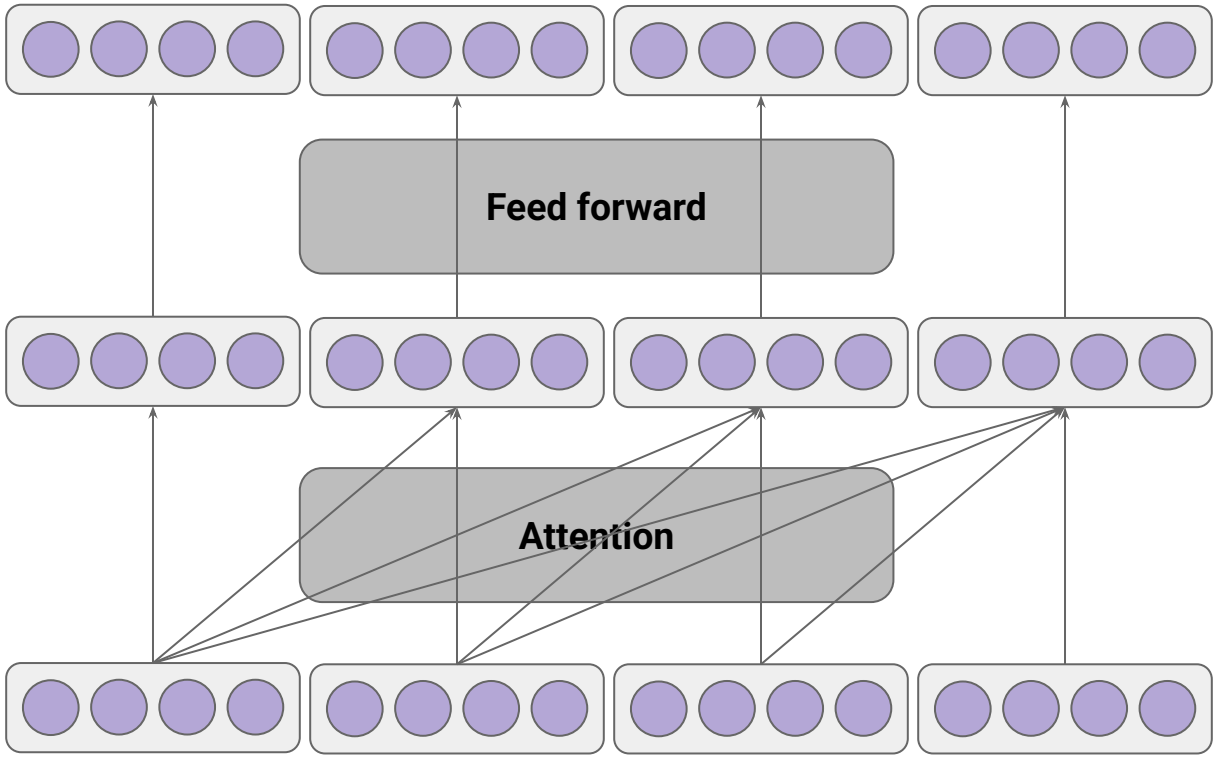
Softmax function

For a vector $y = [y_1, y_2, \ldots, y_V]$ of dimension V, the softmax transformation is calculated as:

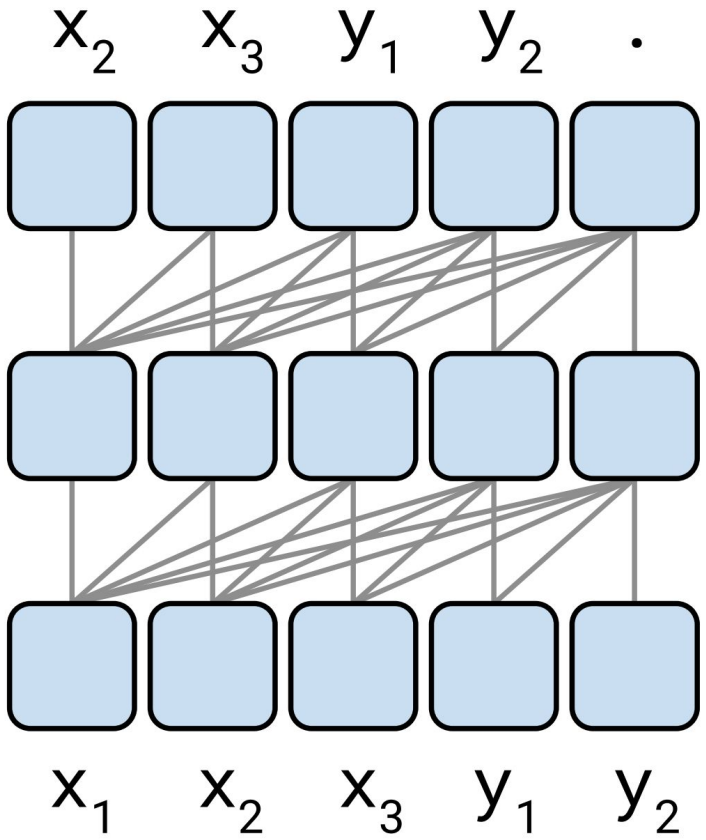
$$\text{softmax}(y) = \left[\frac{e^{y_1}}{\sum e^y}, \frac{e^{y_2}}{\sum e^y}, \ldots, \frac{e^{y_V}}{\sum e^y} \right]$$

where $\sum e^y = e^{y_1} + e^{y_2} + \cdots + e^{y_V}$.

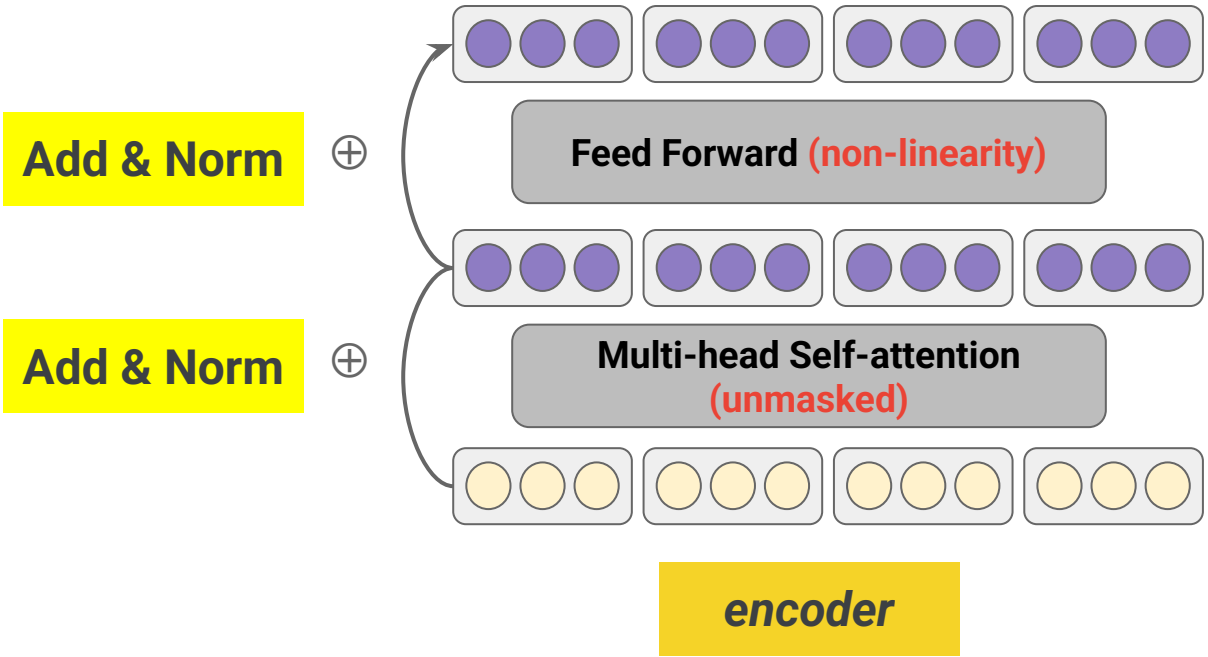
Decoder (cont'd)



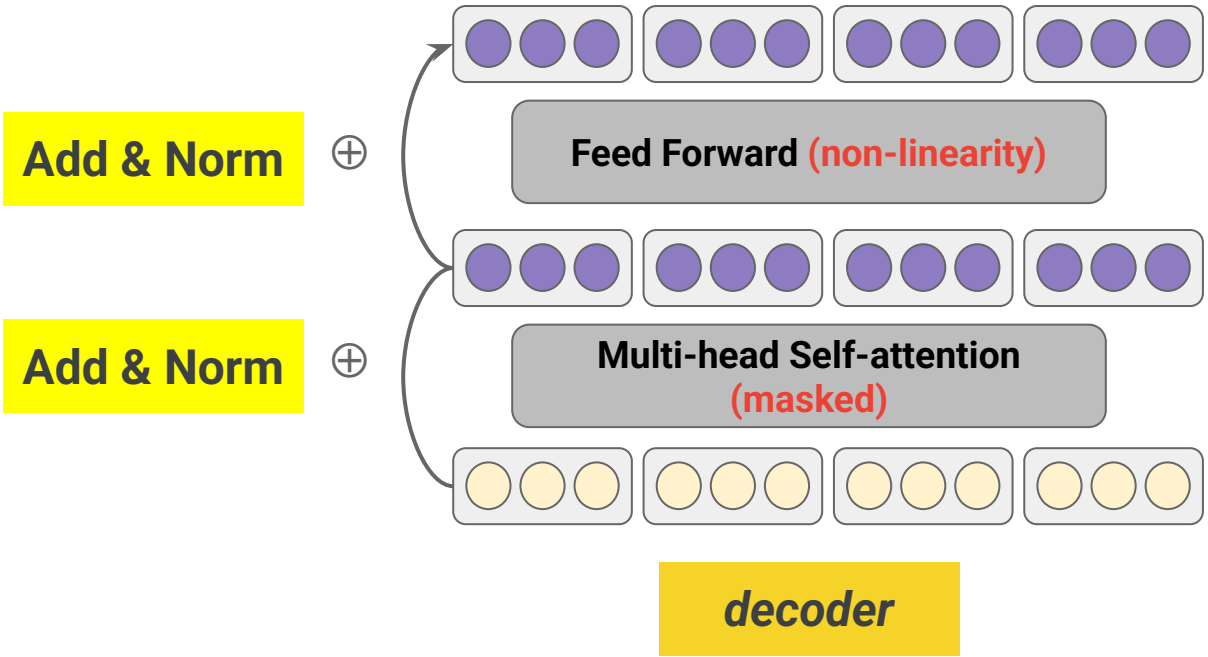
Decoder (cont'd)



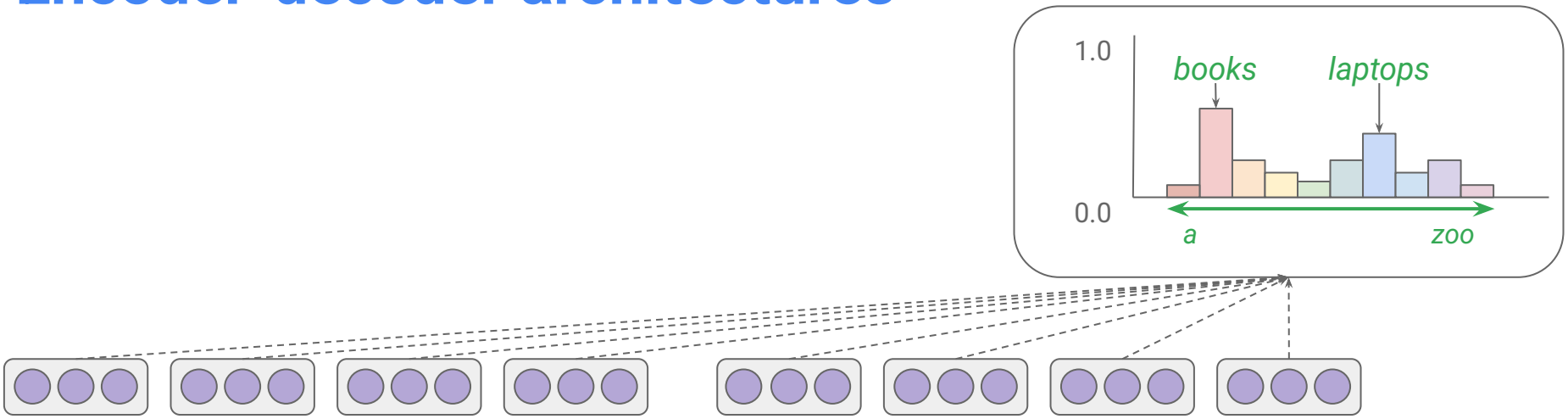
Encoder (one layer)



Decoder (one layer)



Encoder-decoder architectures

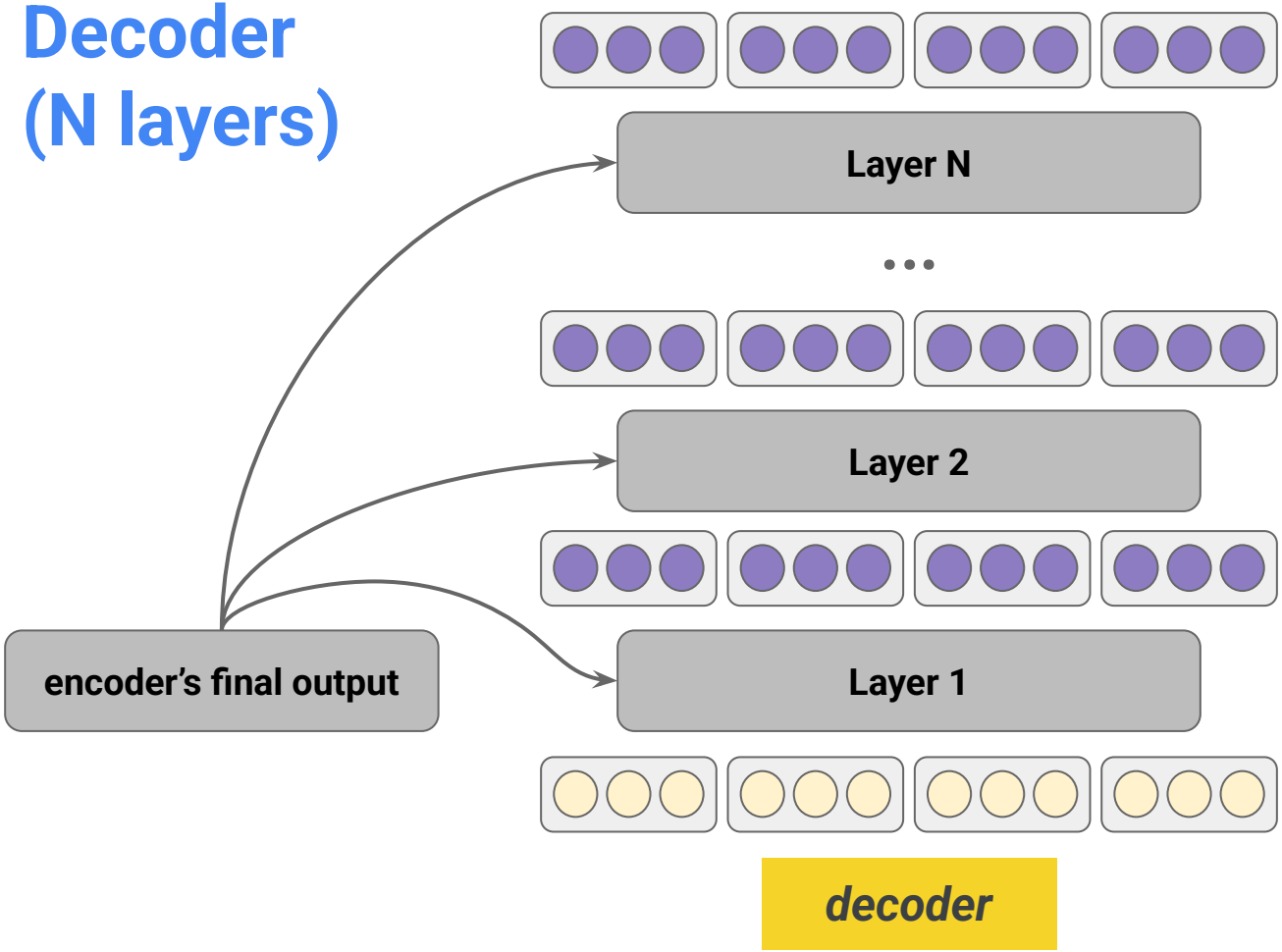


les étudiants ont ouvert leurs livres

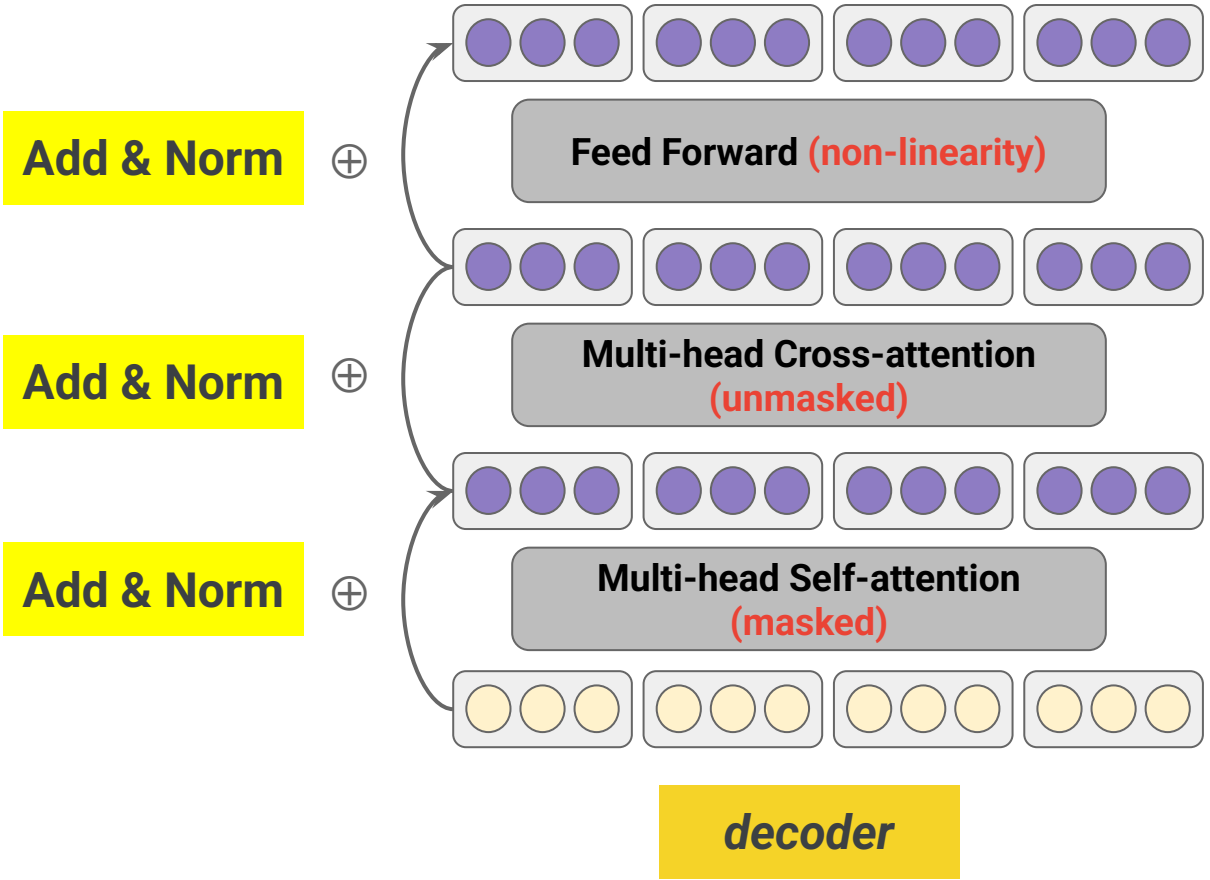
the *students* *opened* *their*

encoder ***decoder***

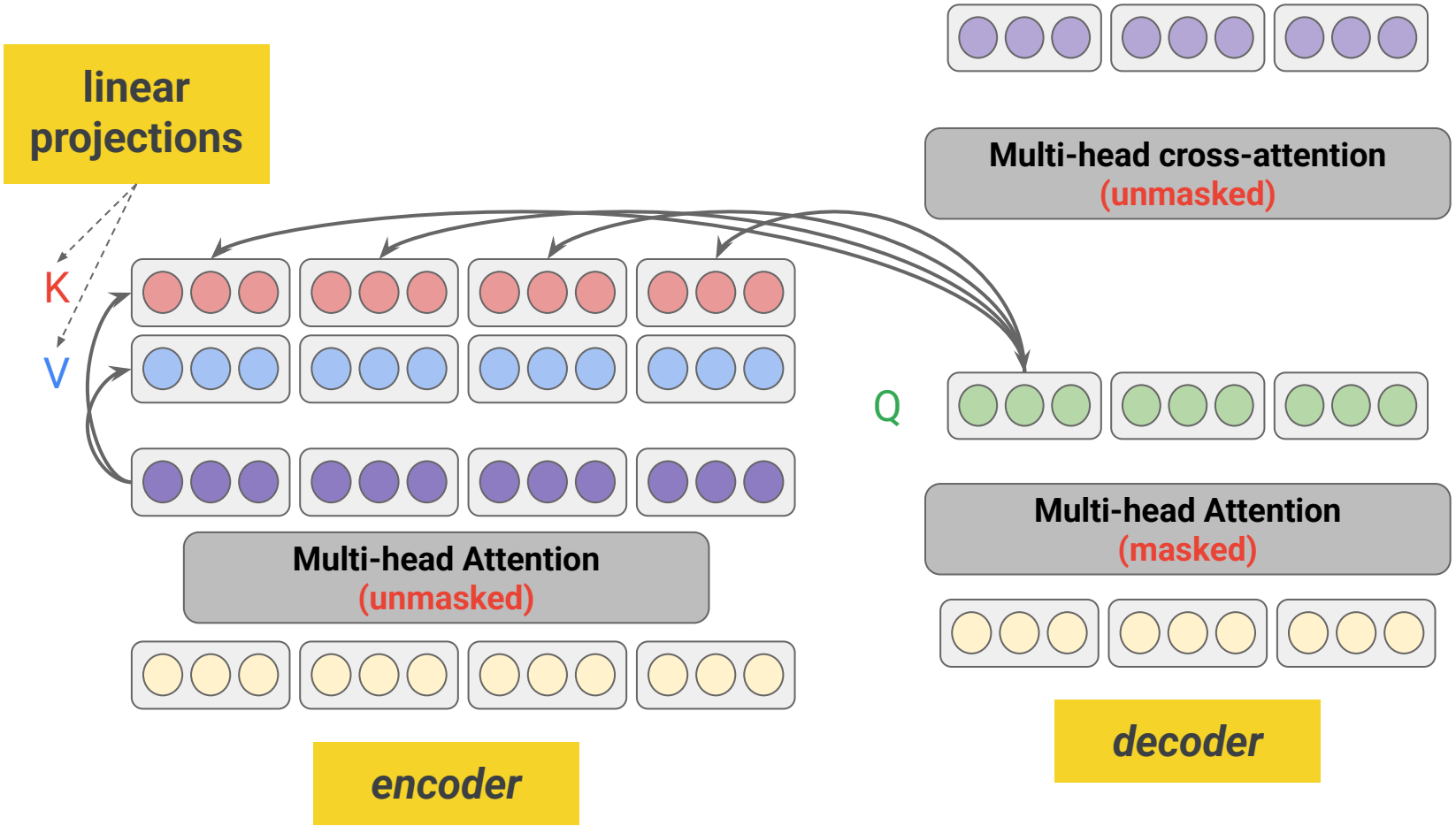
Decoder (N layers)



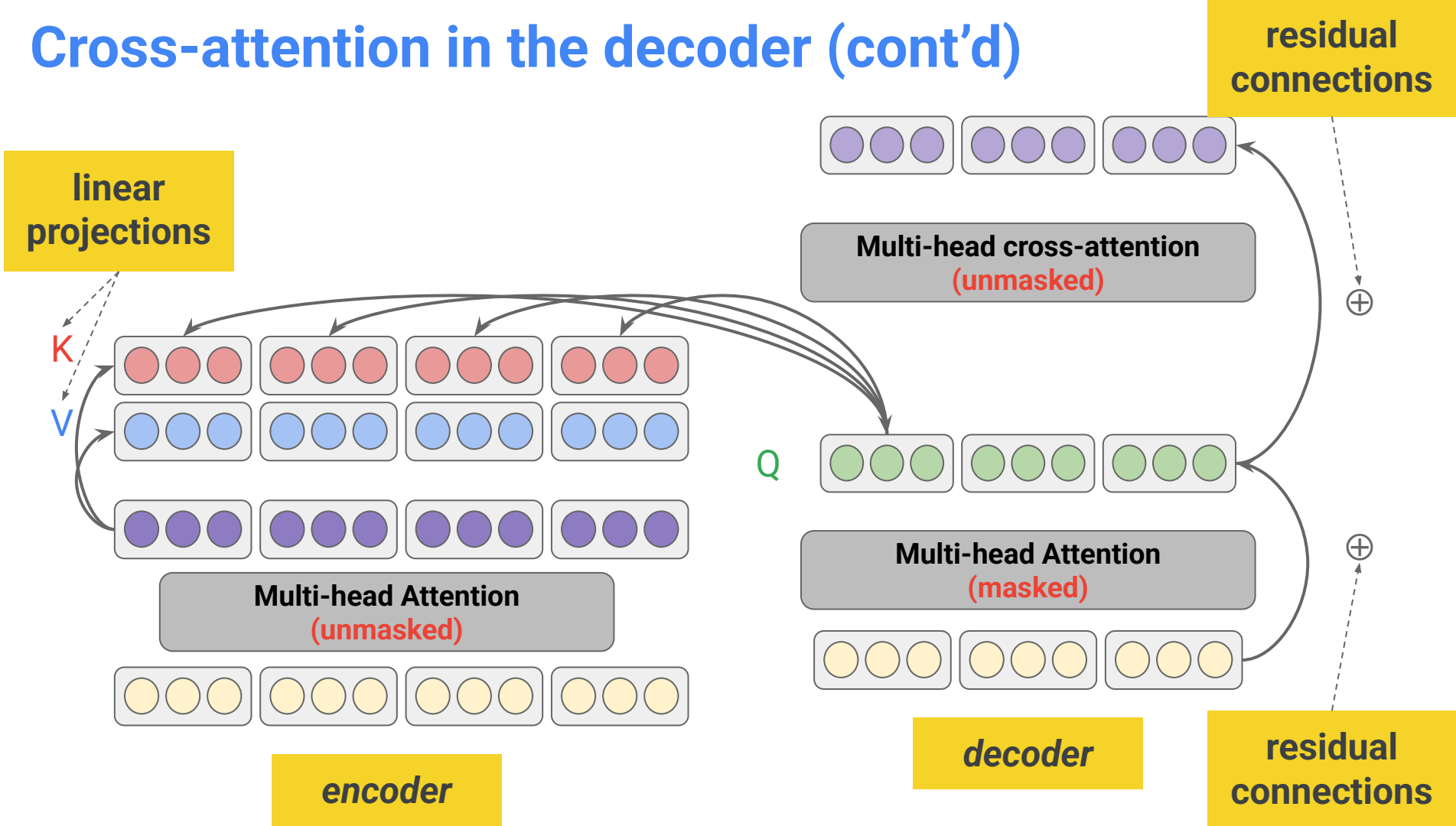
Decoder (one layer)

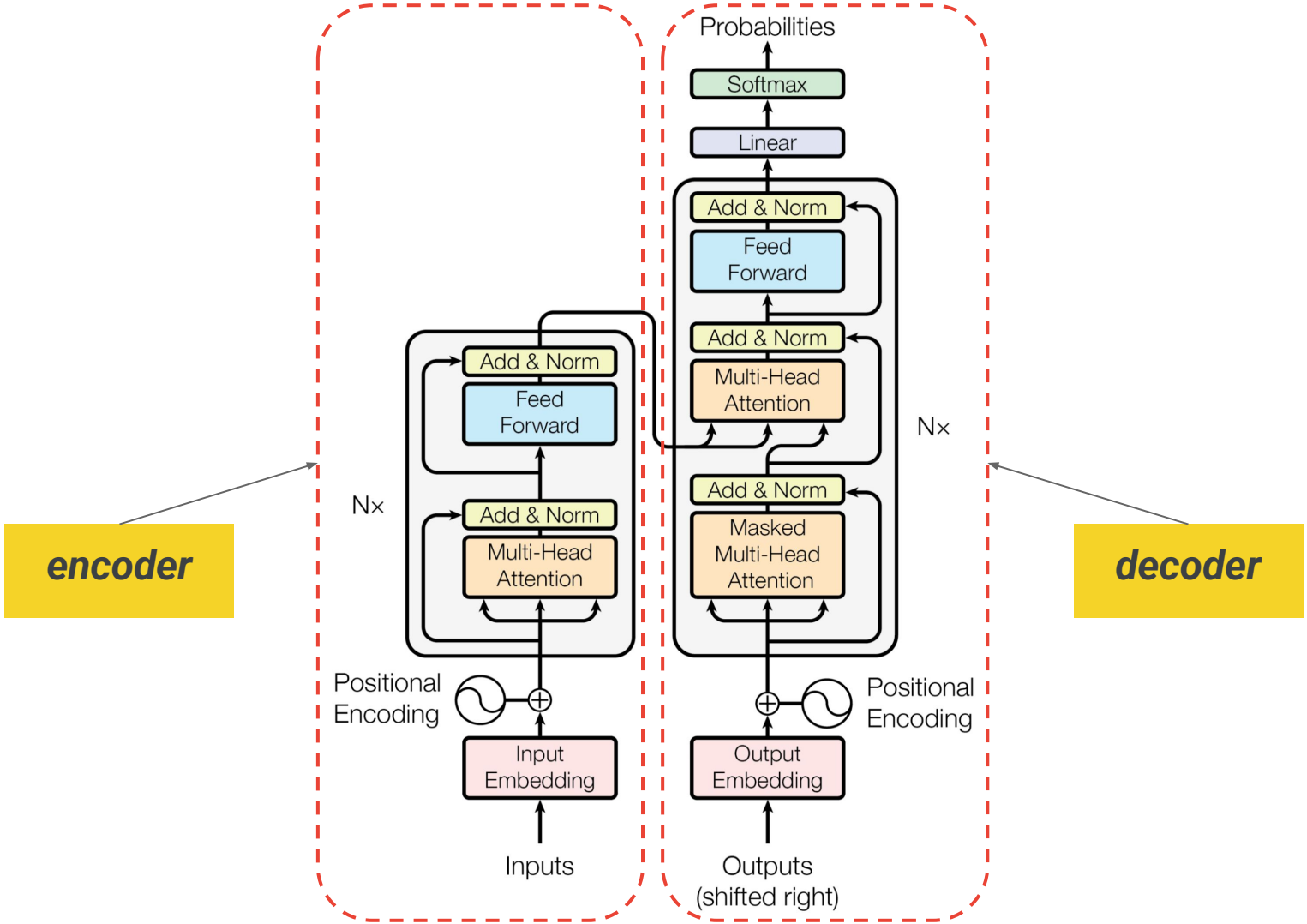


Cross-attention in the decoder



Cross-attention in the decoder (cont'd)




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

Thank you!