Neural Language Models & Transformers

CS 5624: Natural Language Processing

Fall 2026

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

Tu Vu

VIRGINIA TECH

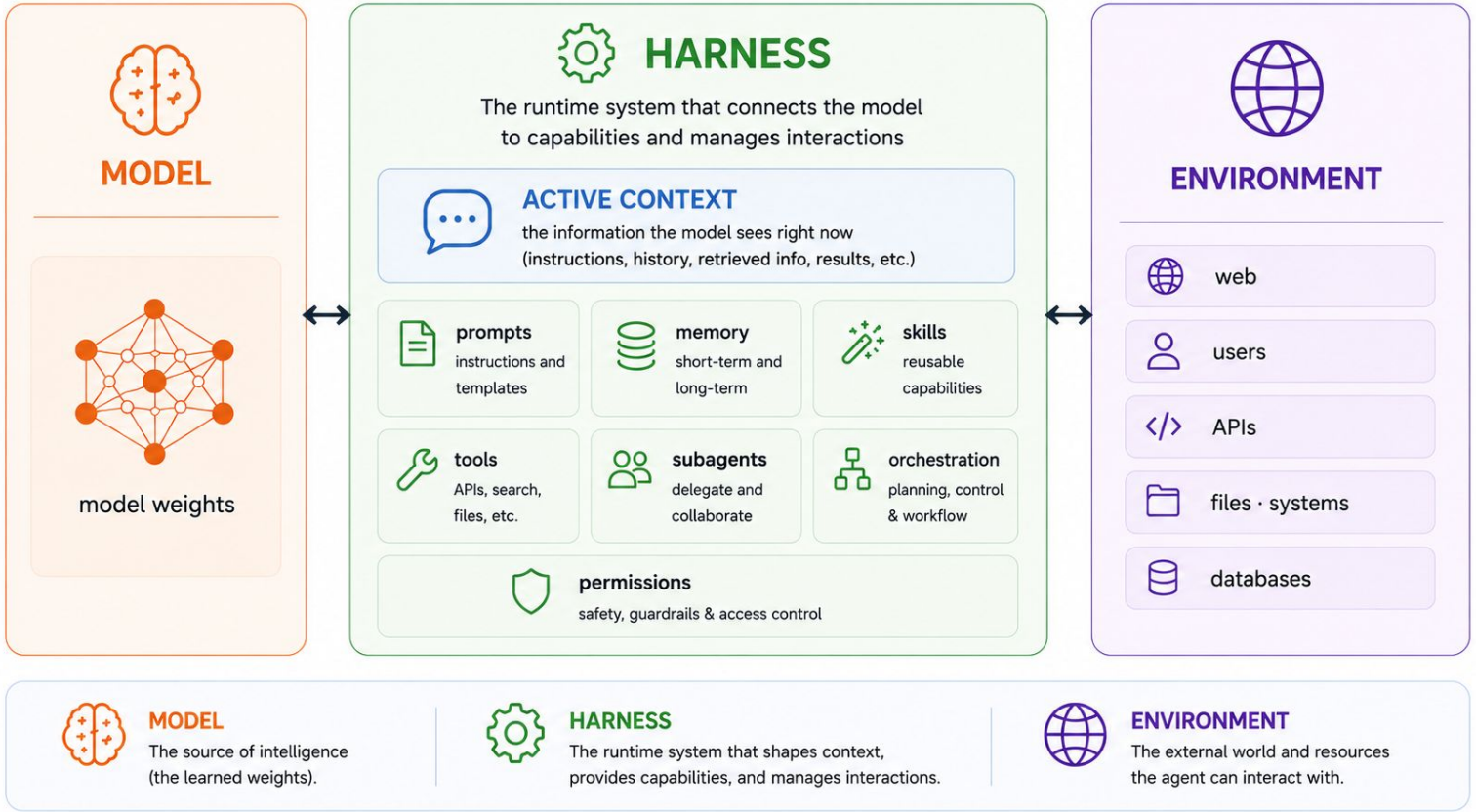
Logistics

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

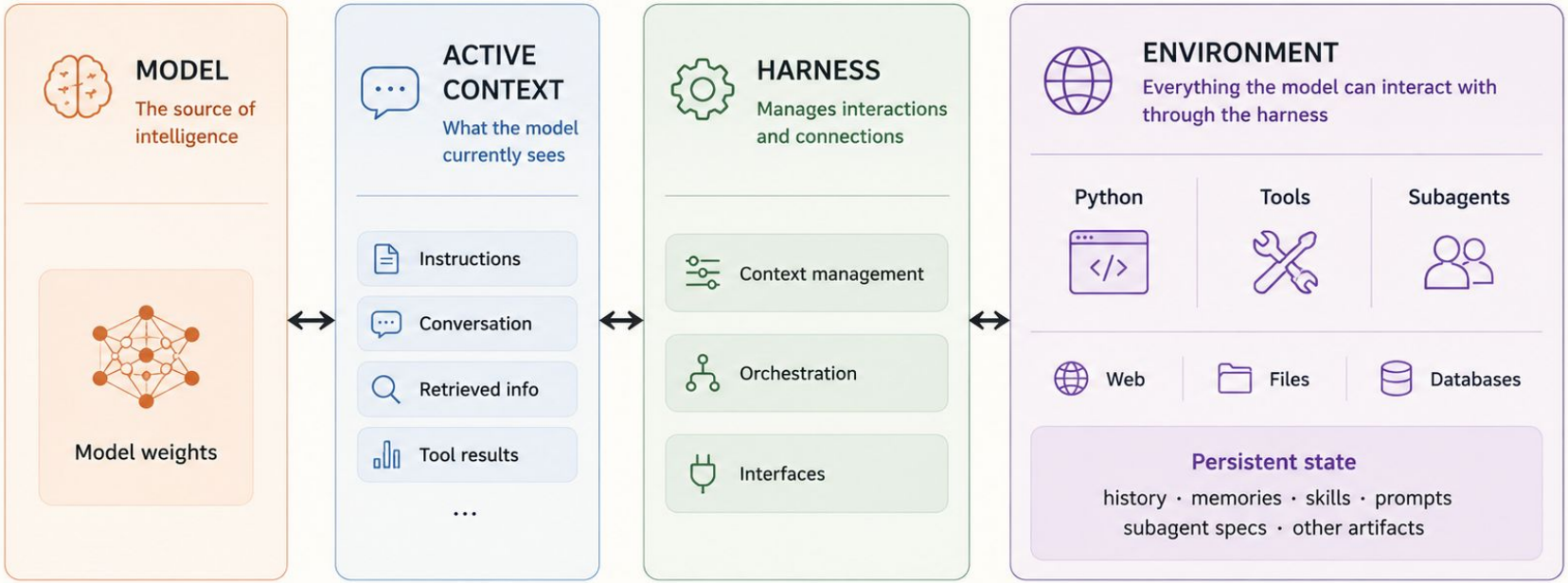
- Student presentations
  - Search for teammates on Piazza/Discord or reach out to us at cs5624instructors@gmail.com
  - Google form for submitting group information available later today (due 9/8)

AI news

Conceptual architecture of an LLM agent



A narrower definition of harness



The model generates decisions based on its context.

The harness builds that context and manages how the model interacts with the environment.

Agent frameworks

- Claude Code
- OpenCode
- Codex Gemini CLI / Antigravity CLI
- OpenHands
- Pi
- Prime Agent
- …

Prime Agent: A Self-Improving RLM Harness

Seth Karten[♠♡◇] Alex L. Zhang[♡♣◇] Kevin Thomas[♡] Sebastian Müller[♡]
Elie Bakouch[♡] Daniel Auras[♡] Mika Senghaas[♡] Fares Obeid[♡]
Konstantin Dunas[♡] Johannes Hagemann[♡] Sami Jaghouar[♡]

[♠] Princeton University [♡] Prime Intellect [♣] MIT
[◇] Correspondence: seth@primeintellect.ai, altzhang@mit.edu

First published: August 5, 2026 • Current version: August 24, 2026

Abstract

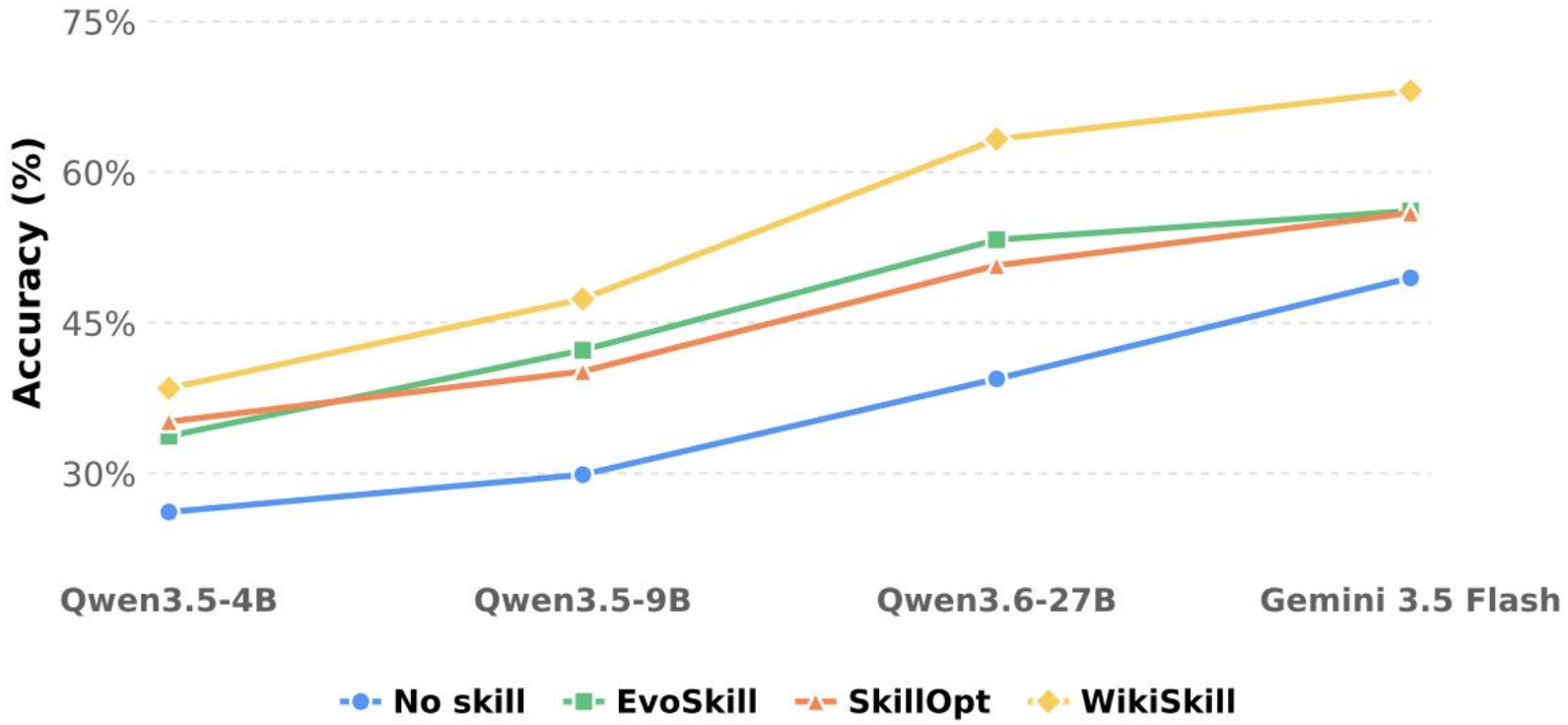
Language models are sequential processors, but long-horizon agency requires external information and computation beyond model weights and active context. Prime Agent is an open-source harness for long-horizon evaluation and coding-agent workflows. A persistent IPython REPL follows the Recursive Language Model abstraction for programmatic context processing and test-time compute, while Continual Harness preserves histories, memories, skills, prompts, and subagent specifications across trajectories. Recursive subagents coordinate through direct agent-to-agent communication, and the Agents View lets humans inspect and manage daemon-backed sessions. Prime Agent standardizes execution, recovery, verification, and resource accounting while leaving strategy construction to the model. This low-friction, expressive membrane prevents harness failures from becoming model failures and pushes measurement toward the model's true maximal underlying capability. Prime Agent raises ARC-AGI-3 RHAE Best@1 from 30% to 95.5% and matches or exceeds native and popular harnesses across long-context coding, GPU-kernel generation, emulator construction, and autonomous nanoGPT speedruns. On Factorio, we find refinement allows for continuous technology progression and dedicated subagents enable parallelized work. Code is available at https://github.com/PrimeIntellect-ai/prime-agent.

Google Research 2026-08-28

WikiSkill: Compiling Agent Experience into Persistent Knowledge for Skill Evolution

Liyan Tang[1], Cyrus Rashtchian[1], Chun-Sung Ferng[1], Andrew Tomkins[1], Da-Cheng Juan[1] and Tu Vu[1,2]
[1]Google Research, [2]Virginia Tech

Agent skills package specialized knowledge and workflows into reusable resources that extend AI agent capabilities. Recent work automatically discovers such skills from agent experience, which enables agents to progressively adapt through interaction. However, the insights that guide skill development typically remain scattered across optimization histories, limiting their systematic reuse across iterations. We introduce WikiSkill, a framework that co-evolves agent skills with a *persistent* knowledge base (wiki). At a high level, WikiSkill separates raw execution experience, accumulated knowledge, and executable skills, while continuously consolidating experience into the wiki, which subsequent skill updates can build on. Across diverse benchmarks and models, WikiSkill consistently outperforms *state-of-the-art* skill-evolution methods and improves over no-skill baselines in most model-benchmark settings. We find that skill evolution complements model scaling: larger models generally benefit more from evolved skills, while smaller models with skills can outperform substantially larger models without them. We also find that evolved skills transfer effectively across models and model families, and skills evolved by other models can outperform self-evolved skills. Finally, our ablation studies confirm that persistent knowledge accumulation in the wiki is critical for effective skill evolution. These results demonstrate the benefits of systematically accumulating and refining agent experience for developing reusable and transferable skills.

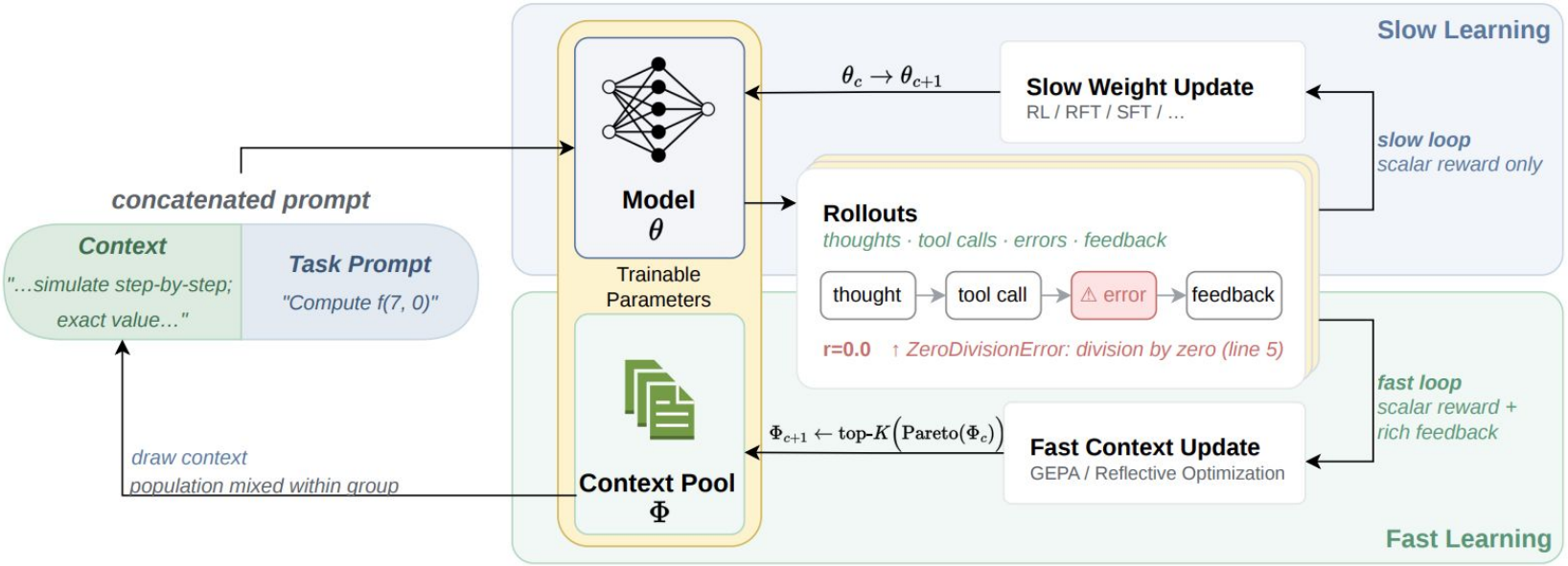

75%
60%
45%
30%
Accuracy (%)
Qwen3.5-4B
Qwen3.5-9B
Qwen3.6-27B
Gemini 3.5 Flash
No skill
EvoSkill
SkillOpt
WikiSkill

Skill & harness evolution

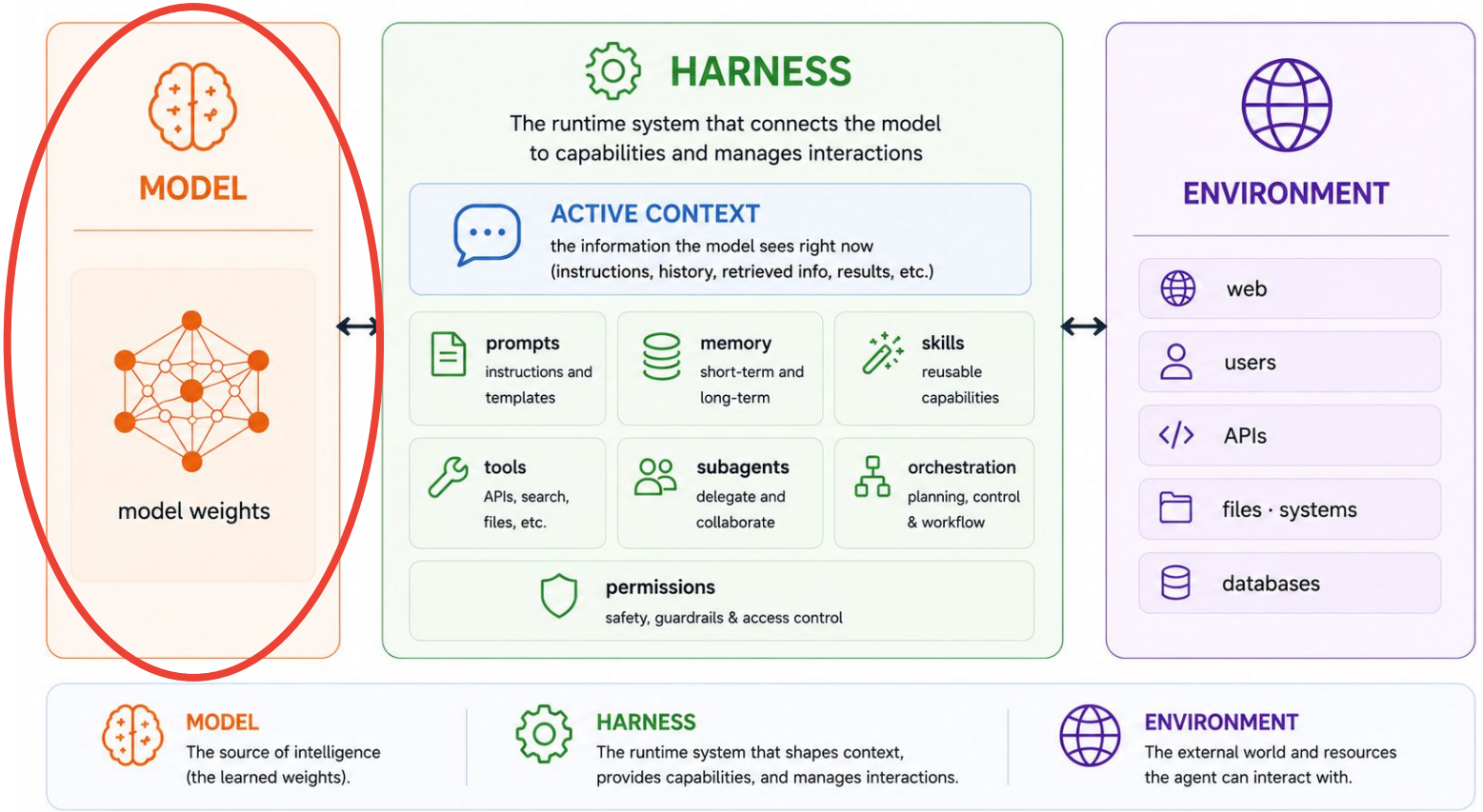
- Meta-Harness: https://arxiv.org/abs/2603.28052
- EvoSkill: https://arxiv.org/abs/2603.02766
- SkillOpt: https://arxiv.org/abs/2605.23904
- …

Model & harness co-evolution

- Learning, Fast and Slow: https://arxiv.org/abs/2605.12484
- HarnessX: https://arxiv.org/abs/2606.14249



Today


MODEL
model weights
HARNESS
The runtime system that connects the model to capabilities and manages interactions
ACTIVE CONTEXT
the information the model sees right now (instructions, history, retrieved info, results, etc.)
prompts
instructions and templates
memory
short-term and long-term
skills
reusable capabilities
tools
APIs, search, files, etc.
subagents
delegate and collaborate
orchestration
planning, control & workflow
permissions
safety, guardrails & access control
ENVIRONMENT
web
users
APIs
files · systems
databases
MODEL
The source of intelligence (the learned weights).
HARNESS
The runtime system that shapes context, provides capabilities, and manages interactions.
ENVIRONMENT
The external world and resources the agent can interact with.

Math review

Probability

- Conditional probability $P(B|A) = \frac{P(A,B)}{P(A)}$

  Rewriting

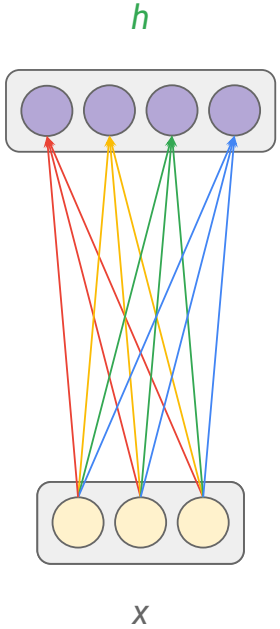
$$P(A,B) = P(A) \times P(B|A)$$

- Chain rule

$$\begin{aligned} P(X_1, X_2, \ldots, X_n) &= P(X_1, X_2, \ldots, X_{n-1}) \times P(X_n|X_1, X_2, \ldots, X_{n-1}) \\ &= P(X_1, X_2, \ldots, X_{n-2}) \times P(X_{n-1}|X_1, X_2, \ldots, X_{n-2}) \times P(X_n|X_1, X_2, \ldots, X_{n-1}) \\ &= P(X_1) \times P(X_2|X_1) \times \ldots \times P(X_n|X_1, X_2, \ldots, X_{n-1}) \end{aligned}$$

$$P(w_1, w_2, \ldots, w_n) = P(w_1) \times P(w_2|w_1) \times \ldots \times P(w_n|w_1, w_2, \ldots, w_{n-1})$$

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):

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

Softmax function

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

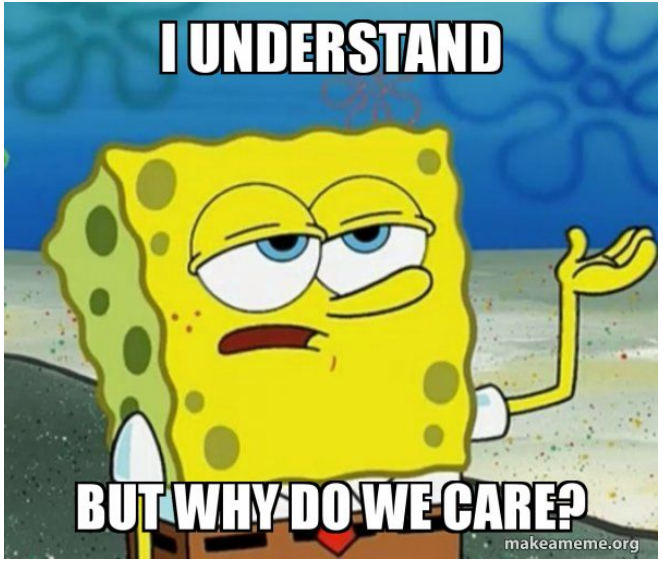
$$\text{softmax}(y) = \left[\frac{e^{y_1}}{\sum e^y}, \frac{e^{y_2}}{\sum e^y}, \ldots, \frac{e^{y_V}}{\sum e^y} \right]$$

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

Language modeling

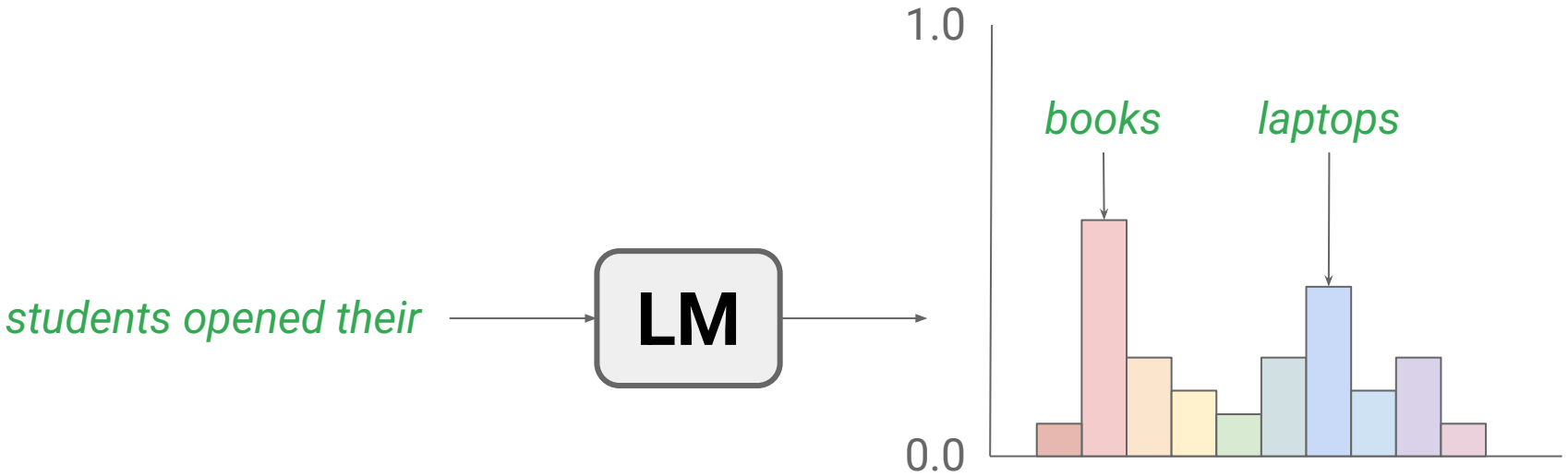
- **Predicting the next/missing word**

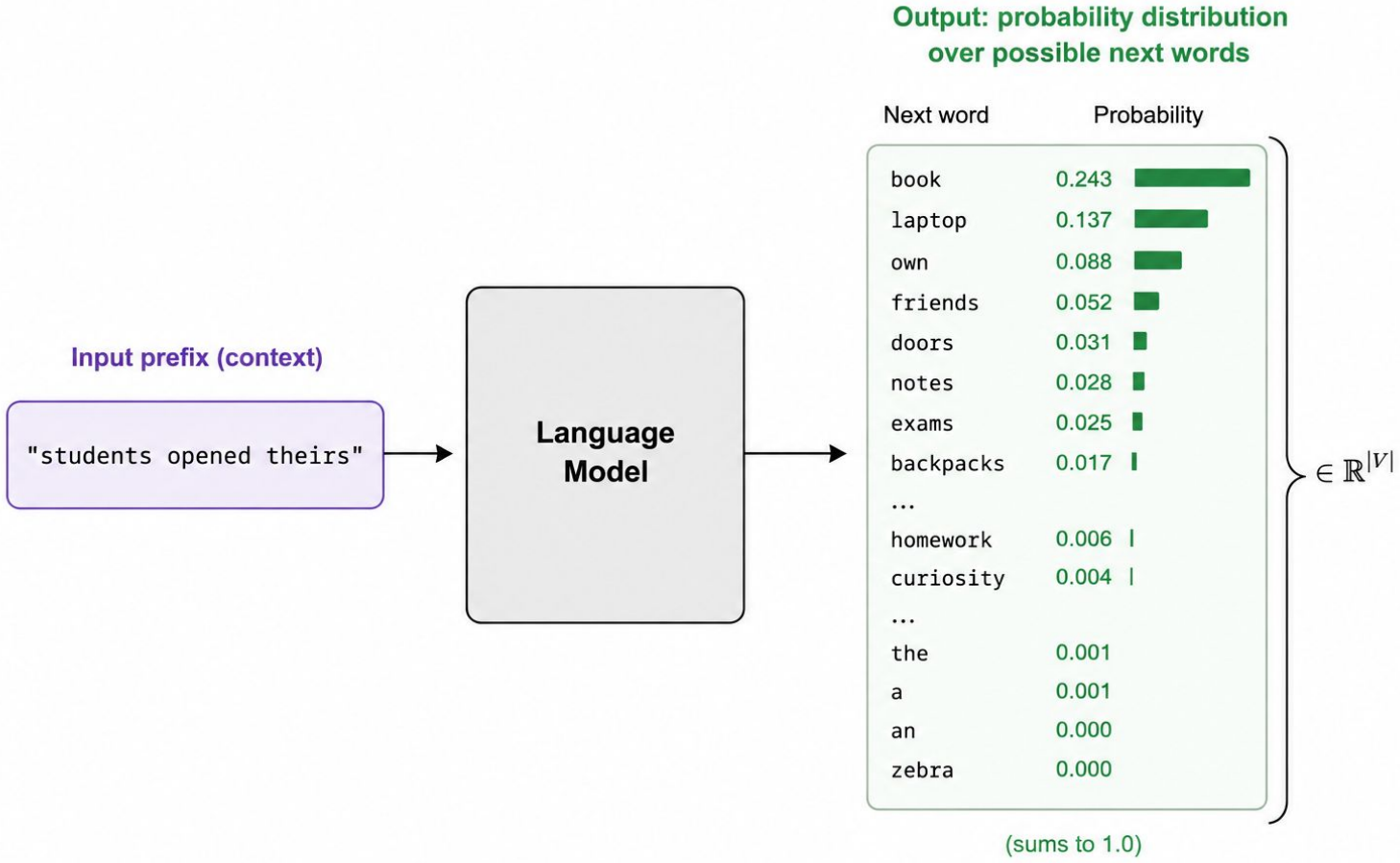
  Example: “The cat is on the ___.” → Predicted: “mat”.



What is a language model?

- A machine learning model that assigns a ***probability*** to each possible next word, or a ***probability distribution*** over possible next words




Output: probability distribution over possible next words
Next word
Probability
book 0.243
laptop 0.137
own 0.088
friends 0.052
doors 0.031
notes 0.028
exams 0.025
backpacks 0.017
...
homework 0.006
curiosity 0.004
...
the 0.001
a 0.001
an 0.000
zebra 0.000
$\in \mathbb{R}^{|V|}$
(sums to 1.0)
Input prefix (context)
"students opened theirs"
Language Model

What is a language model? (cont'd)

- A language model can also assign a probability to an entire sentence

$$P(w_1, w_2, \ldots, w_n) = P(w_1) \times P(w_2|w_1) \times \ldots \times P(w_n|w_1, w_2, \ldots, w_{n-1})$$

P(*"The cat is on the mat"*) = ***P***(*"The"*) *x* ***P***(*"cat" | "The"*) *x* ***P***(*"is" | "The cat"*) *x* ***P***(*"on" | "The cat is"*) *x* ***P***(*"the" | "The cat is on"*) *x* ***P***(*"mat" | "The cat is on the"*)

Autoregressive decoding

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

Two categories of language models

- Statistical/Probabilistic language models
  - N-gram / Count-based language models

- Neural language models (e.g., ChatGPT, Gemini)

N-grams

- An n-gram is a sequence of n words
- Unigram (n=1)
  - “The”, “water”, “of”, “Walden”, “Pond”
- Bigram (n=2)
  - “The water”, “water of”, “of Walden”, “Pond”
- Trigram (n=3)
  - “The water of”, “water of Walden”, “of Walden Pond”
- 4-gram
- …

N-grams (cont'd)

- Notation
  - **word type**: a unique word in our vocabulary
  - **token**: an individual occurrence of a word type

Example: "I am Sam. Sam am I. I do not like green eggs and ham."

→ one word type of "I", three tokens of "I"

N-grams (cont'd)

- How to compute the probabilities?

$$P(w_1, w_2, \ldots, w_n) = P(w_1) \times P(w_2|w_1) \times \ldots \times P(w_n|w_1, w_2, \ldots, w_{n-1})$$

***P**(“blue” | “The water of Walden Pond is so beautifully”)*

What is the problem with this approach?

The Markov assumption

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

$$P(\text{“blue”} \mid \text{“The water of Walden Pond is so beautifully”}) = P(\text{“blue”} \mid \text{beautifully”})$$

- trigram (n=3) model

$$P(\text{“blue”} \mid \text{“The water of Walden Pond is so beautifully”}) = P(\text{“blue”} \mid \text{so beautifully”})$$

The Markov assumption (cont'd)

- unigram model

$$P(w_1, w_2, \ldots, w_n) = P(w_1) \times P(w_2|w_1) \times \ldots \times P(w_n|w_1, w_2, \ldots, w_{n-1})$$
$$\approx P(w_1) \times P(w_2) \times \cdots \times P(w_n)$$
$$= \prod_{k=1}^{n} P(w_k)$$

- bigram model

$$P(w_1, w_2, \ldots, w_n) \approx P(w_1) \times P(w_2|w_1) \times \cdots \times P(w_n|w_{n-1})$$
$$= \prod_{k=1}^{n} P(w_k|w_{k-1})$$

Maximum likelihood estimation (MLE)

$$P(w_n|w_{n-1}) = \frac{Count(w_{n-1}w_n)}{\sum_w Count(w_{n-1}w)} = \frac{Count(w_{n-1}w_n)}{Count(w_{n-1})}$$

relative frequency

<s> I am Sam </s>
<s> Sam I am </s>
<s> I do not like green eggs and ham </s>

Here are the calculations for some of the bigram probabilities

$P(\text{I}|\text{<s>}) = \frac{2}{3} = 0.67$ $P(\text{Sam}|\text{<s>}) = \frac{1}{3} = 0.33$ $P(\text{am}|\text{I}) = \frac{2}{3} = 0.67$

$P(\text{</s>}|\text{Sam}) = \frac{1}{2} = 0.5$ $P(\text{Sam}|\text{am}) = \frac{1}{2} = 0.5$ $P(\text{do}|\text{I}) = \frac{1}{3} = 0.33$

source: Jurafsky and Martin

N-gram language models

- Use maximum likelihood estimation (MLE)

$$P(\text{“laptops”} \mid \text{“students opened their”}) = \frac{\textbf{Count}(\text{“students opened their laptops”})}{\textbf{Count}(\text{“students opened their”})}$$

Example

- From a restaurant corpus

  “can you tell me about any good cantonese restaurants close by”

  “tell me about chez panisse”

  “i’m looking for a good place to eat breakfast”

  “when is caffe venezia open during the day”

source: Jurafsky and Martin

Example (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

Example (cont'd)

827/2533

	i	want	to	eat	chinese	food	lunch	spend
i	0.002	0.33	0	0.0036	0	0	0	0.00079
want	0.0022	0	0.66	0.0011	0.0065	0.0065	0.0054	0.0011
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087
eat	0	0	0.0027	0	0.021	0.0027	0.056	0
chinese	0.0063	0	0	0	0	0.52	0.0063	0
food	0.014	0	0.014	0	0.00092	0.0037	0	0
lunch	0.0059	0	0	0	0	0.0029	0	0
spend	0.0036	0	0.0036	0	0	0	0	0

Here are a few other useful probabilities:

$P(\texttt{i}\,|\,\texttt{<s>}) = 0.25$

$P(\texttt{food}\,|\,\texttt{english}) = 0.5$

$P(\texttt{english}\,|\,\texttt{want}) = 0.0011$

$P(\texttt{</s>}\,|\,\texttt{food}) = 0.68$

$$
\begin{aligned}
P(\texttt{<s> i want english food </s>}) &= P(\texttt{i}\,|\,\texttt{<s>})P(\texttt{want}\,|\,\texttt{i})P(\texttt{english}\,|\,\texttt{want}) \\
&\quad P(\texttt{food}\,|\,\texttt{english})P(\texttt{</s>}\,|\,\texttt{food}) \\
&= 0.25 \times 0.33 \times 0.0011 \times 0.5 \times 0.68 \\
&= 0.000031
\end{aligned}
$$

source: Jurafsky and Martin

Sample generations

1 gram	–To him swallowed confess hear both. Which. Of save on trail for are ay device and rote life have
	–Hill he late speaks; or! a more to leg less first you enter
2 gram	–Why dost stand forth thy canopy, forsooth; he is this palpable hit the King Henry. Live king. Follow.
	–What means, sir. I confess she? then all sorts, he is trim, captain.
3 gram	–Fly, and will rid me these news of price. Therefore the sadness of parting, as they say, 'tis done.
	–This shall forbid it should be branded, if renown made it empty.
4 gram	–King Henry. What! I will go seek the traitor Gloucester. Exeunt some of the watch. A great banquet serv'd in;
	–It cannot be but so.

from King John

Figure 3.4 Eight sentences randomly generated from four n-grams computed from Shakespeare's works. All characters were mapped to lower-case and punctuation marks were treated as words. Output is hand-corrected for capitalization to improve readability.

source: Jurafsky and Martin

Problems with n-gram language models

$$P(\text{“laptops”} \mid \text{“students opened their”}) = \frac{\textbf{Count}(\text{“students opened their laptops”})}{\textbf{Count}(\text{“students opened their”})}$$

What if *“students opened their laptops” never occurred in training data?*

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

	i	want	to	eat	chinese	food	lunch	spend
i	0.002	0.33	0	0.0036	0	0	0	0.00079
want	0.0022	0	0.66	0.0011	0.0065	0.0065	0.0054	0.0011
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087
eat	0	0	0.0027	0	0.021	0.0027	0.056	0
chinese	0.0063	0	0	0	0	0.52	0.0063	0
food	0.014	0	0.014	0	0.00092	0.0037	0	0
lunch	0.0059	0	0	0	0	0.0029	0	0
spend	0.0036	0	0.0036	0	0	0	0	0

Need to store V^n counts for an n-gram model!

source: Jurafsky and Martin

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

sparsity issue

	i	want	to	eat	chinese	food	lunch	spend
i	0.002	0.33	0	0.0036	0	0	0	0.00079
want	0.0022	0	0.66	0.0011	0.0065	0.0065	0.0054	0.0011
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087
eat	0	0	0.0027	0	0.021	0.0027	0.056	0
chinese	0.0063	0	0	0	0	0.52	0.0063	0
food	0.014	0	0.014	0	0.00092	0.0037	0	0
lunch	0.0059	0	0	0	0	0.0029	0	0
spend	0.0036	0	0.0036	0	0	0	0	0

source: Jurafsky and Martin

Shakespeare as corpus

- T=884,647 tokens, V=29,066
- Shakespeare produced 300,000 bigram types out of V^2= 844,000,000 possible bigrams.
- **99.96%** of the possible bigrams have zero entries in the bigram table (were never seen)!

Should we increase the value of n?

- 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

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?

Word representations / embeddings

- High-dimensional / sparse / one-hot representations
- Low-dimensional / dense representations

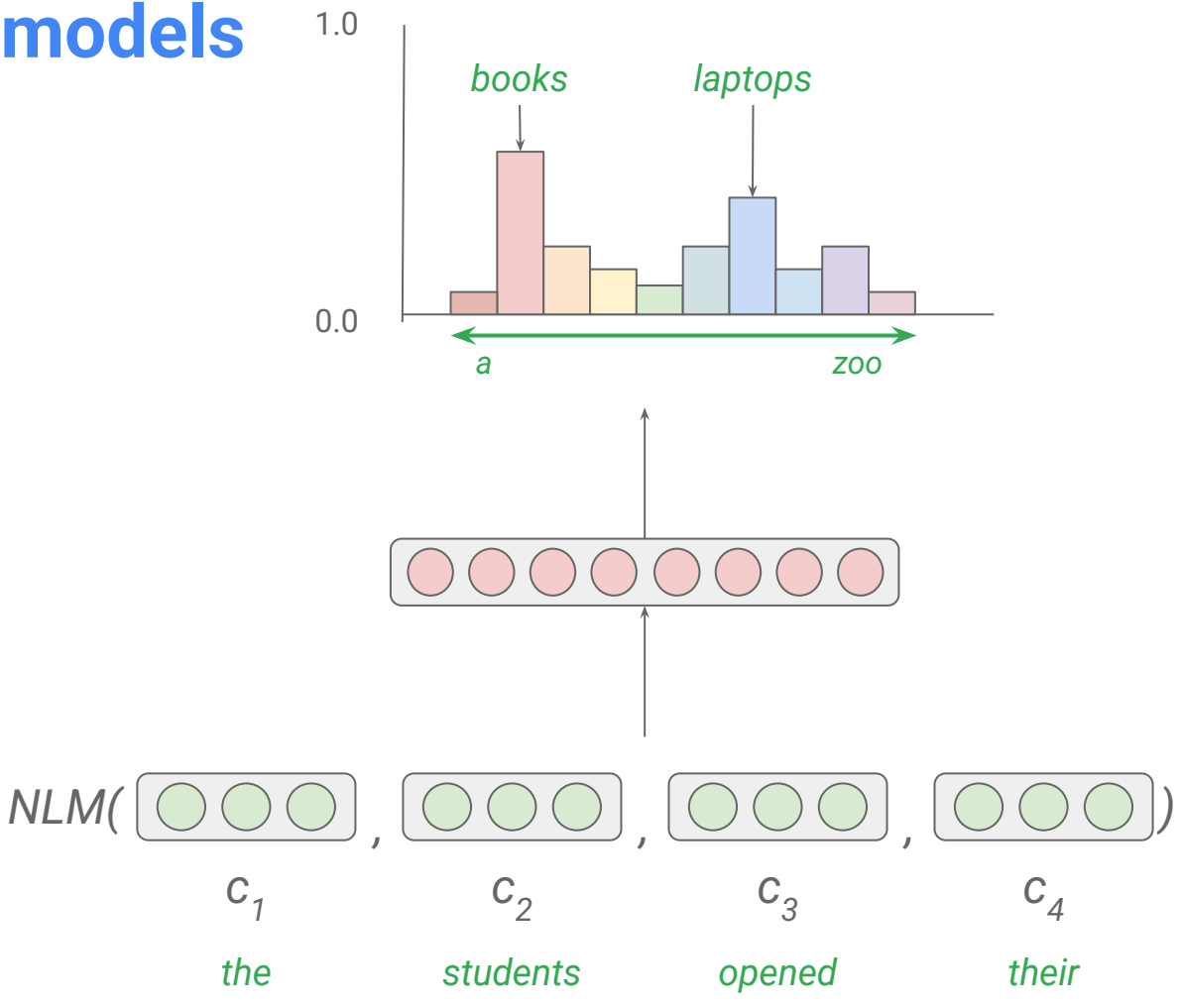
Word representations / embeddings (cont'd)

```
#What is the vector representation for a word?
w2v_model['computer']
```

```
array([ 1.07421875e-01, -2.01171875e-01,  1.23046875e-01,  2.11914062e-01,
       -9.13085938e-02,  2.16796875e-01, -1.31835938e-01,  8.30078125e-02,
        2.02148438e-01,  4.78515625e-02,  3.66210938e-02, -2.45361328e-02,
        2.39257812e-02, -1.60156250e-01, -2.61230469e-02,  9.71679688e-02,
       -6.34765625e-02,  1.84570312e-01,  1.70898438e-01, -1.63085938e-01,
       -1.09375000e-01,  1.49414062e-01, -4.65393066e-04,  9.61914062e-02,
        1.68945312e-01,  2.60925293e-03,  8.93554688e-02,  6.49414062e-02,
        3.56445312e-02, -6.93359375e-02, -1.46484375e-01, -1.21093750e-01,
       -2.27539062e-01,  2.45361328e-02, -1.24511719e-01, -3.18359375e-01,
       -2.20703125e-01,  1.30859375e-01,  3.66210938e-02, -3.63769531e-02,
       -1.13281250e-01,  1.95312500e-01,  9.76562500e-02,  1.26953125e-01,
        6.59179688e-02,  6.93359375e-02,  1.02539062e-02,  1.75781250e-01,
       -1.68945312e-01,  1.21307373e-03, -2.98828125e-01, -1.15234375e-01,
        5.66406250e-02, -1.77734375e-01, -2.08984375e-01,  1.76757812e-01,
        2.38037109e-02, -2.57812500e-01, -4.46777344e-02,  1.88476562e-01,
        5.51757812e-02,  5.02929688e-02, -1.06933594e-01,  1.89453125e-01,
       -1.16210938e-01,  8.49609375e-02, -1.71875000e-01,  2.45117188e-01,
       -1.73828125e-01, -8.30078125e-03,  4.56542969e-02, -1.61132812e-02,
        1.86523438e-01, -6.05468750e-02, -4.17480469e-02,  1.82617188e-01,
        2.20703125e-01, -1.22558594e-01, -2.55126953e-02, -3.08593750e-01,
        9.13085938e-02,  1.60156250e-01,  1.70898438e-01,  1.19628906e-01,
        7.08007812e-02, -2.64892578e-02, -3.08837891e-02,  4.06250000e-01,
       -1.01562500e-01,  5.71289062e-02, -7.26318359e-03, -9.17968750e-02,
       -1.50390625e-01, -2.55859375e-01,  2.16796875e-01, -3.63769531e-02,
        2.24609375e-01,  8.00781250e-02,  1.56250000e-01,  5.27343750e-02,
```

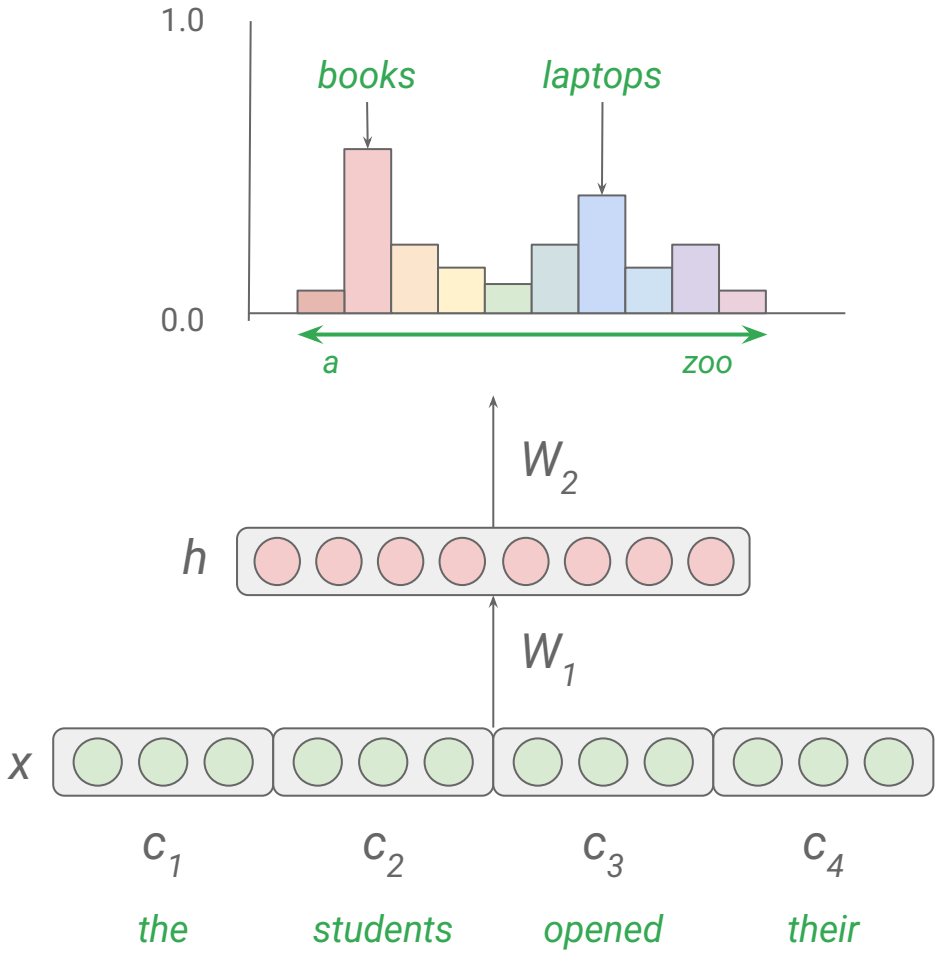
Connected to Python 3 Google Compute Engine backend

Neural language models



output distribution

$\hat{y} = softmax(W_2 h)$



Composition functions

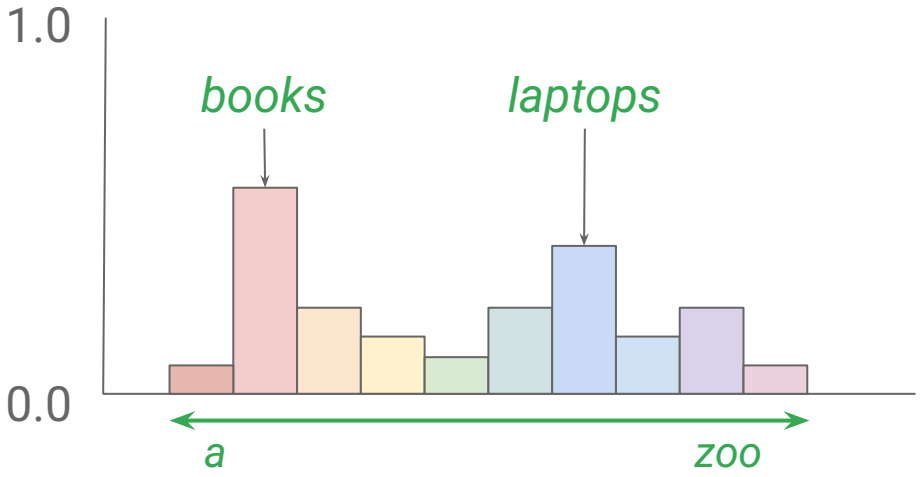
- Element-wise functions
  - e.g., just sum up all of the word embeddings
- Concatenation
- Feedforward neural networks
- Convolutional neural networks
- Recurrent neural networks
- Transformers

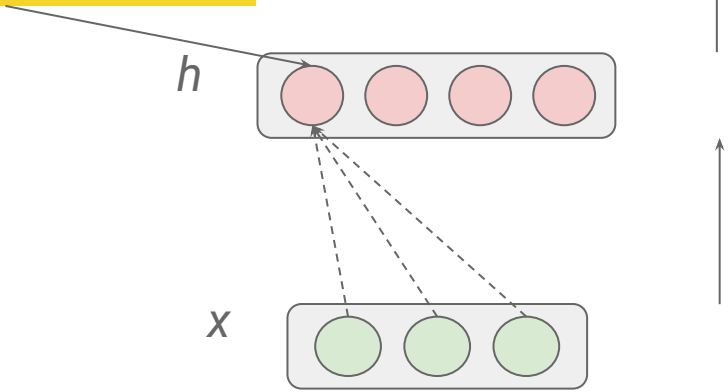
Feedforward neural language model

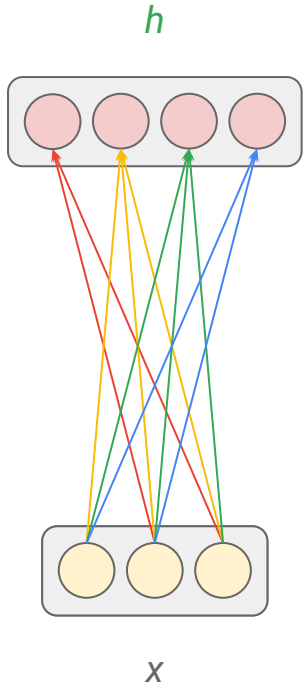
hidden layer

$h = f(W_1 x)$

hidden unit: taking a weighted sum of its inputs and then applying a non-linearity







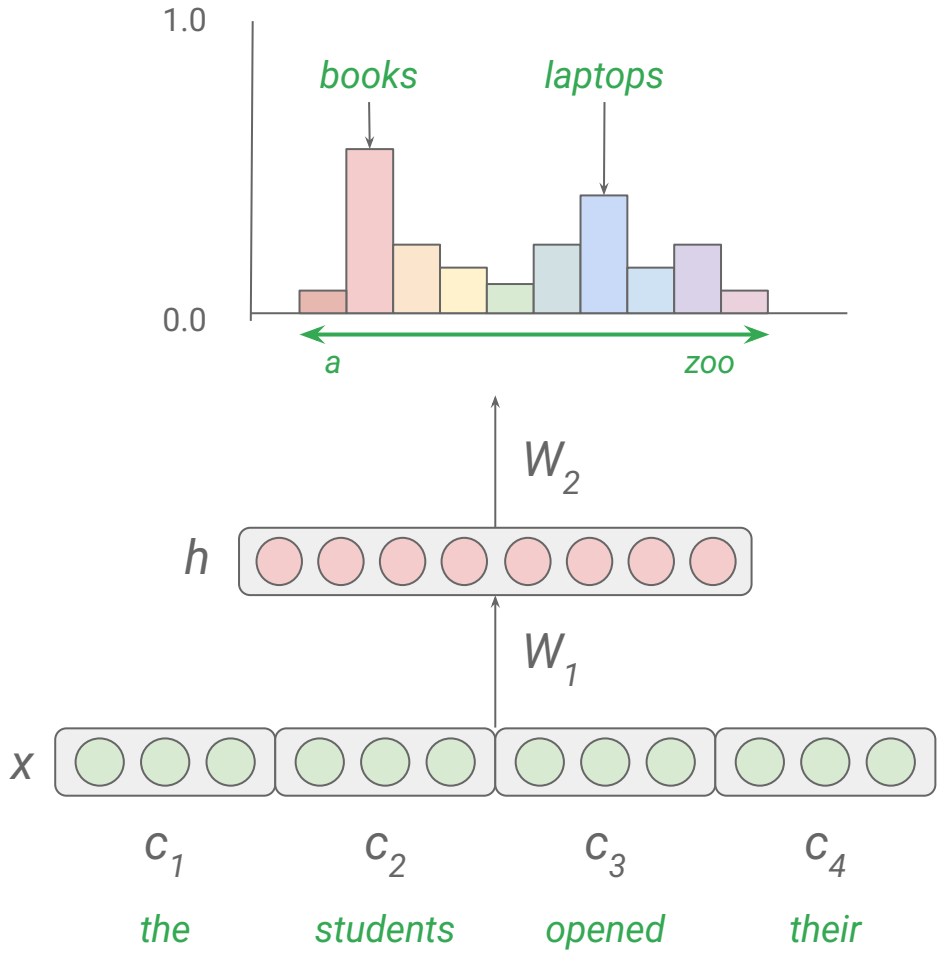
hidden layer

$h = f(W_1 x)$

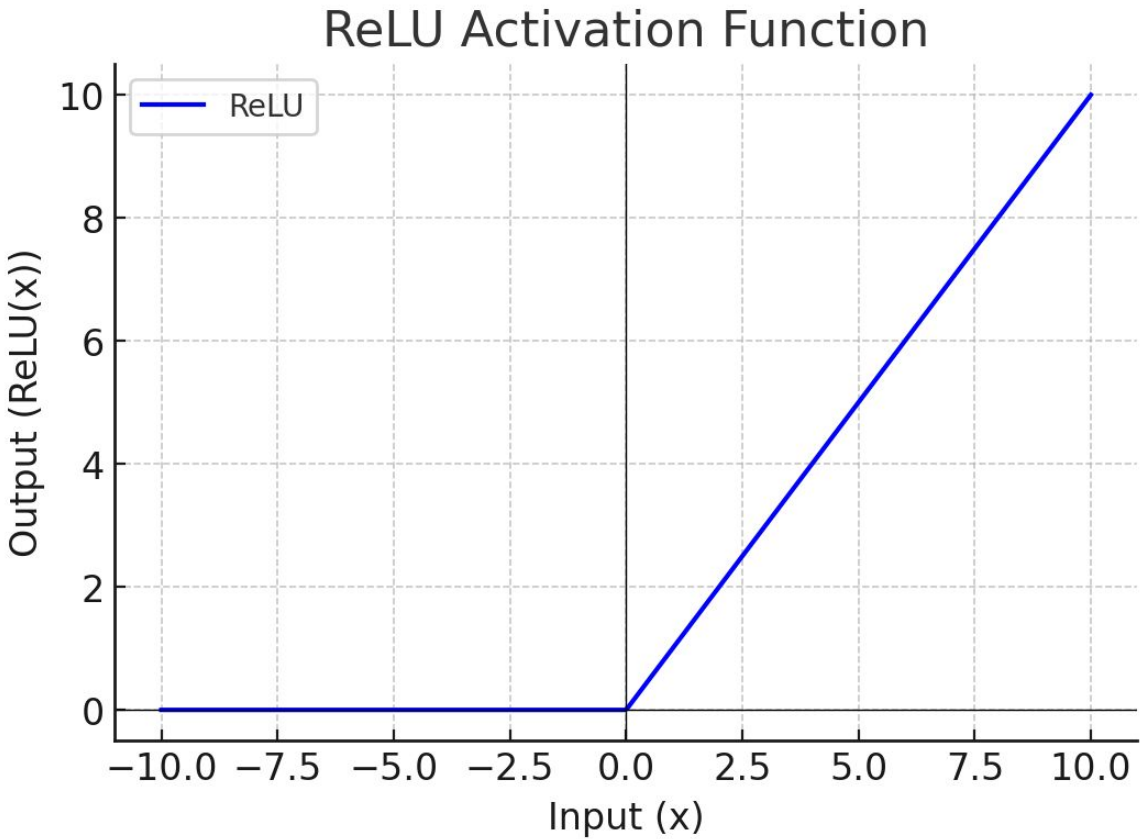
f is a non-linear activation function to model non-linear relationships between words

output distribution

$\hat{y} = softmax(W_2 h)$

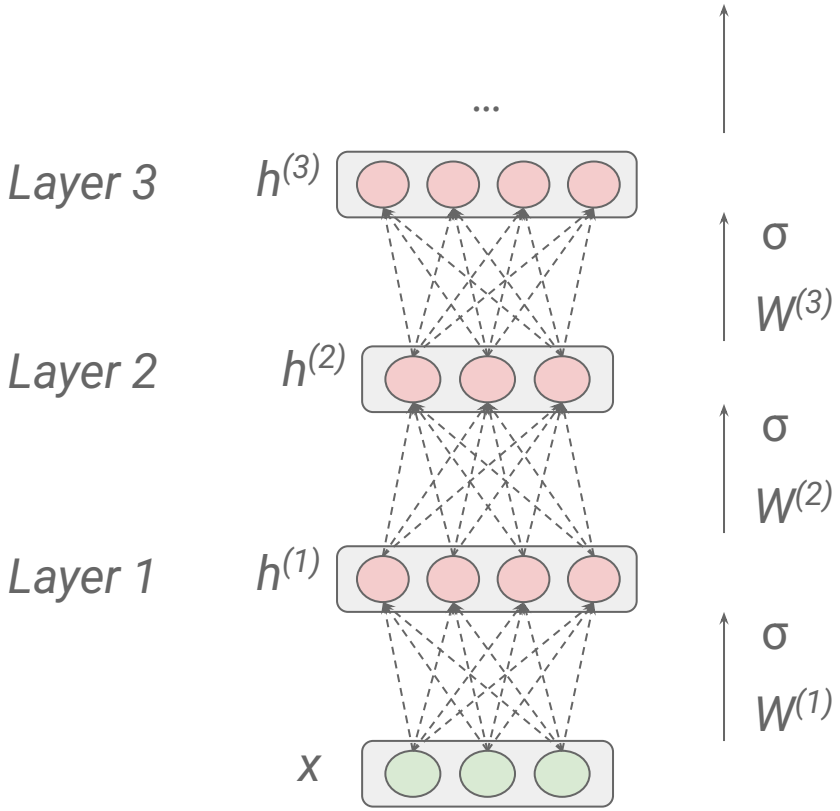


Activation functions



Rectified Linear Unit (ReLU)

Deep neural networks



hierarchical representations, where each layer builds upon the previous one

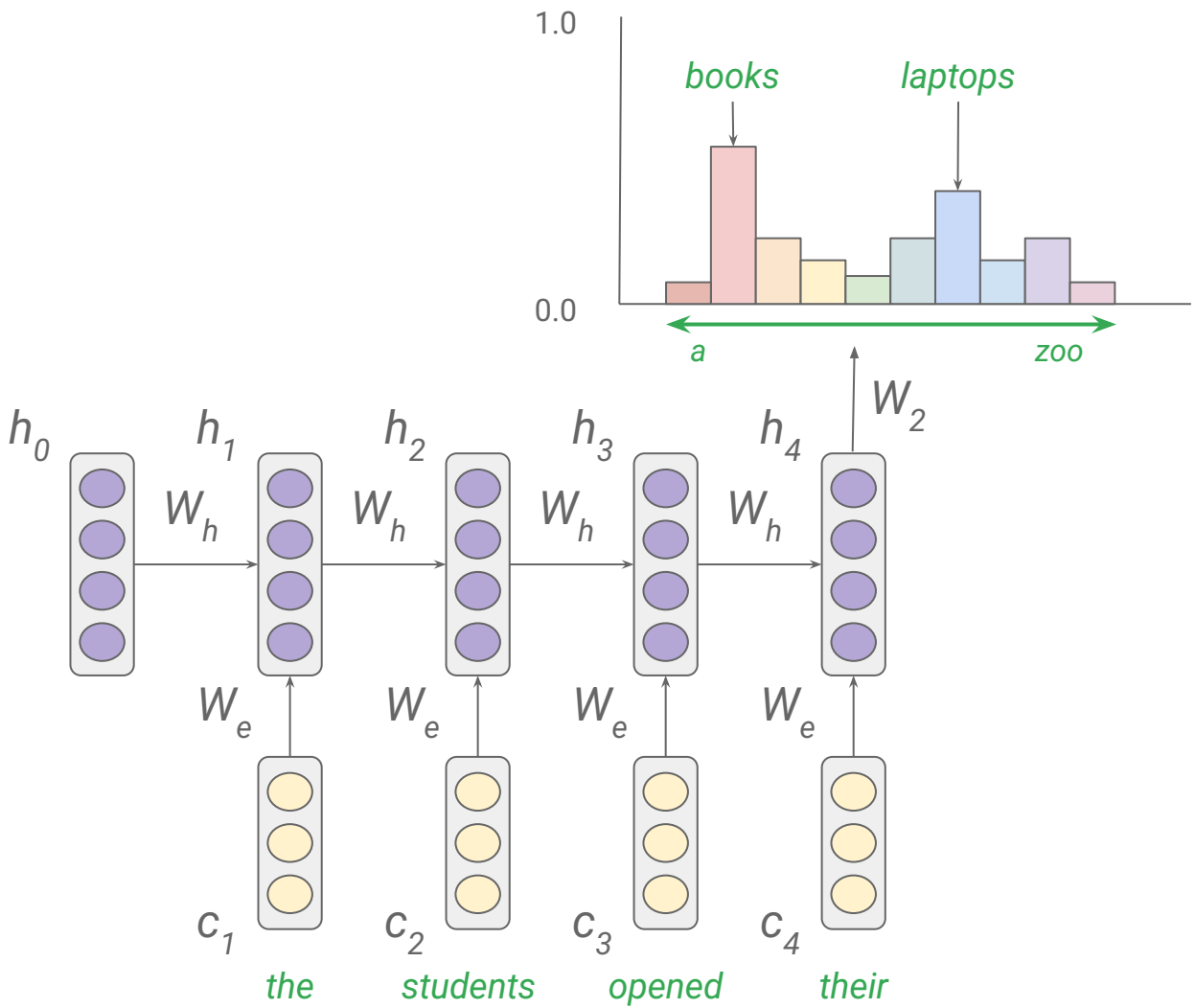
Recurrent neural networks (RNNs)

hidden states

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

output distribution

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



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

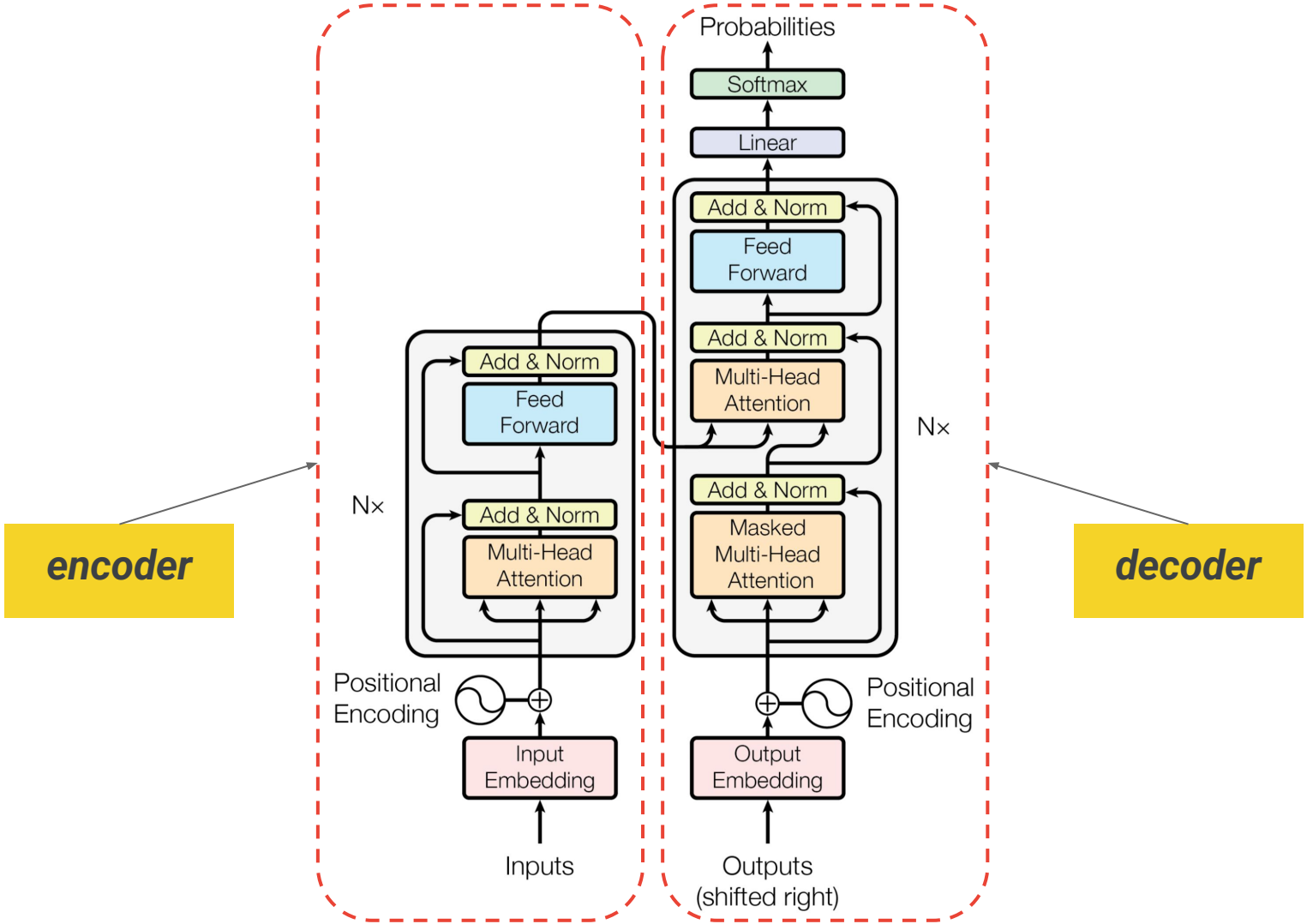
Illia Polosukhin* illia.polosukhin@gmail.com

Abstract

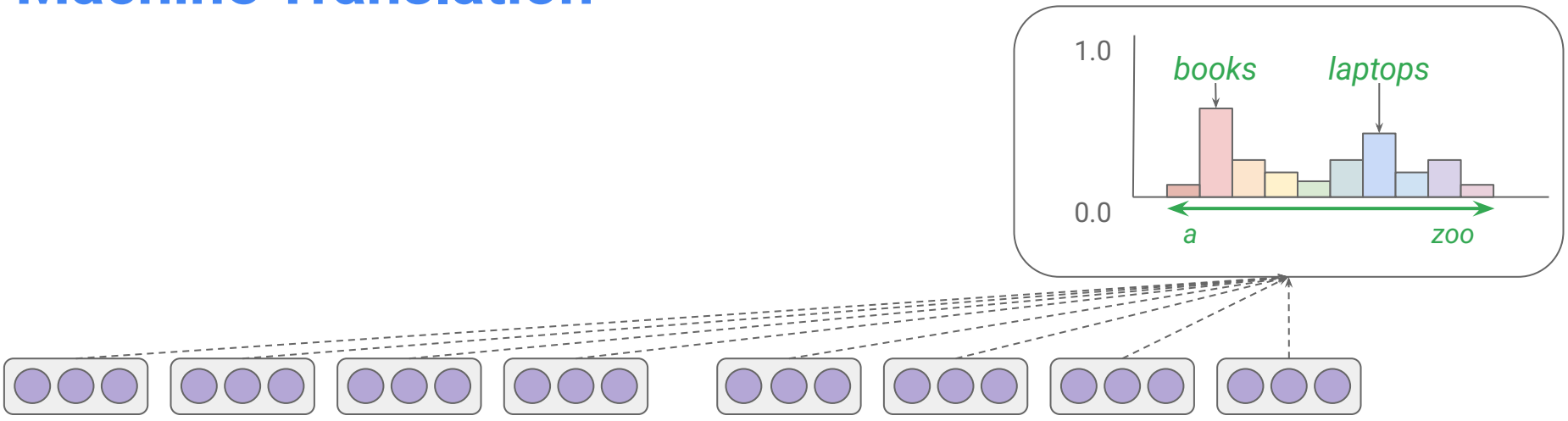
The dominant sequence transduction models are based on complex recurrent or convolutional neural networks in an encoder-decoder configuration. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, 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

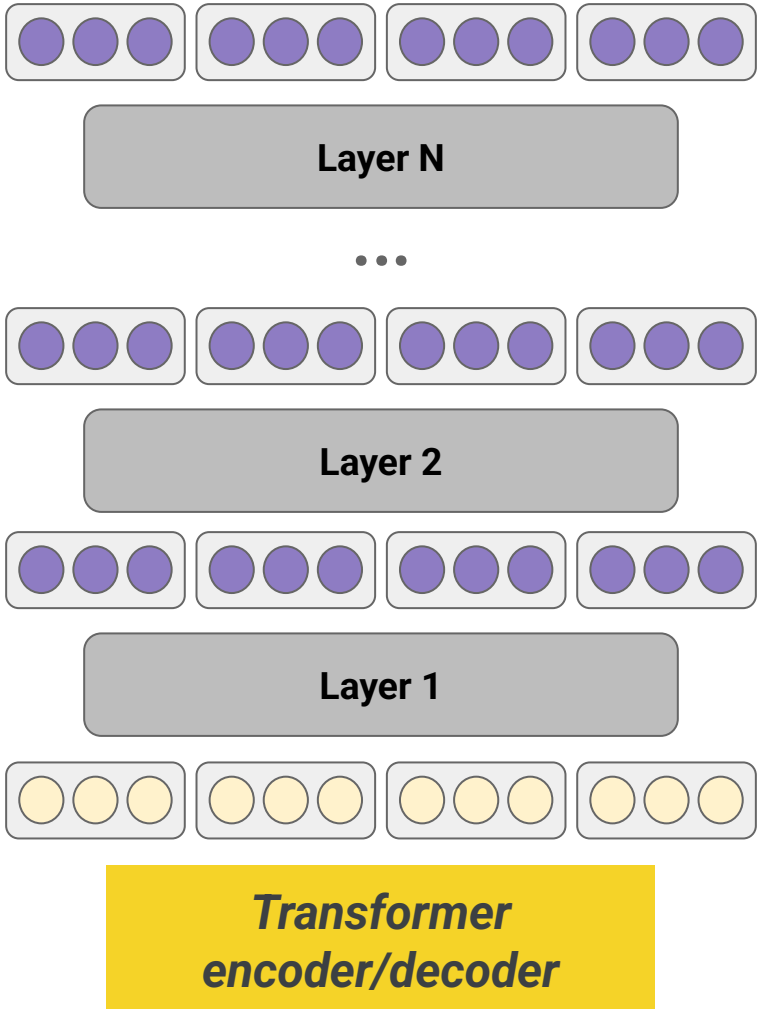


les étudiants ont ouvert leurs livres

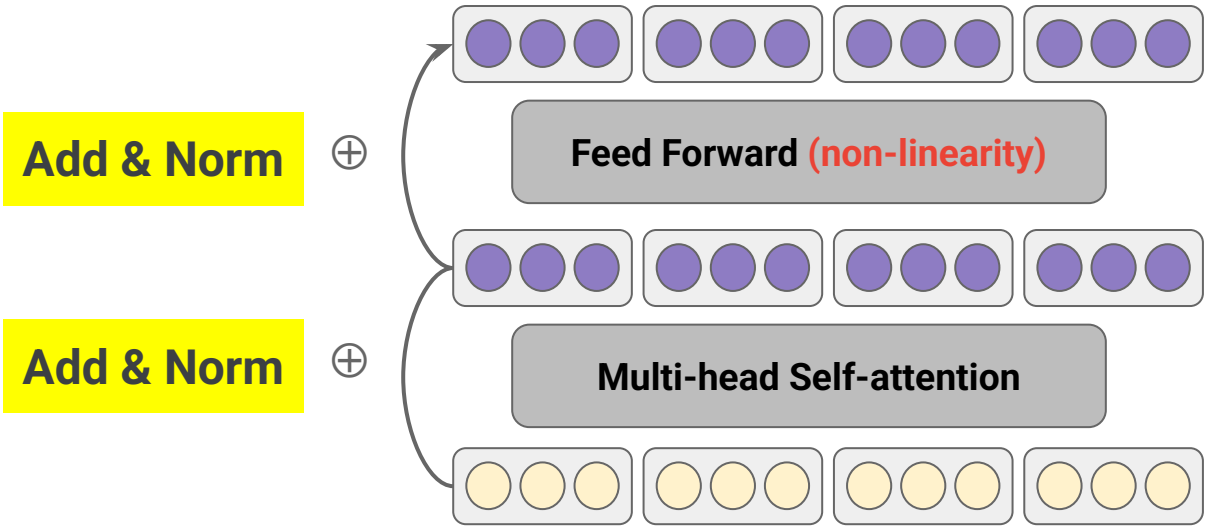
the *students* *opened* *their*

encoder ***decoder***

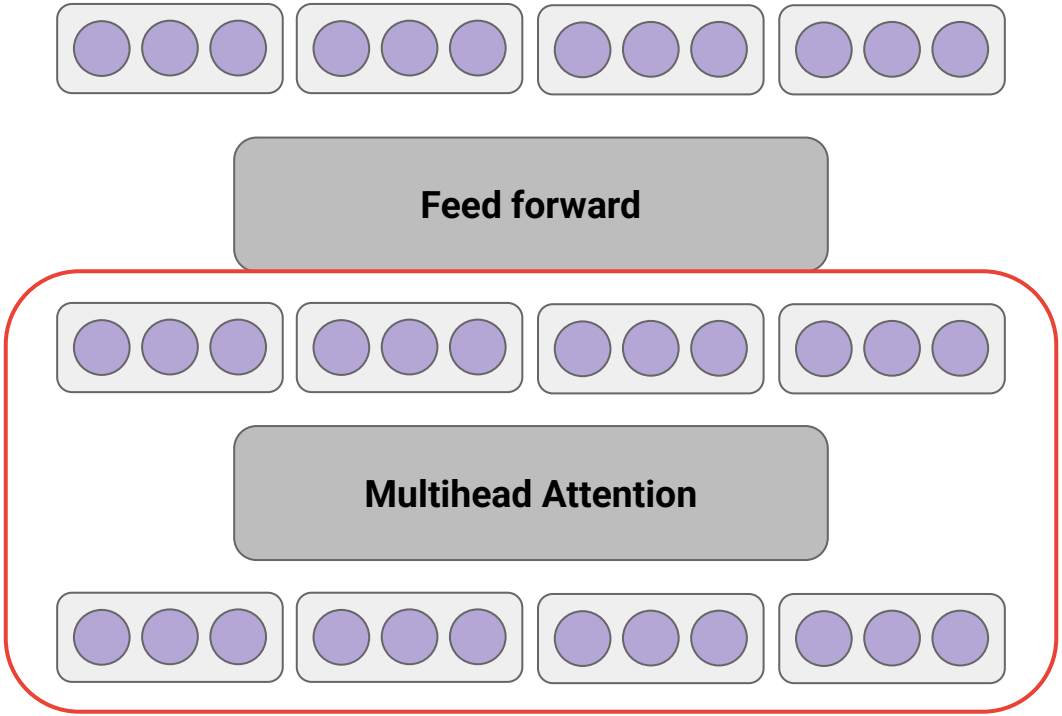
N layers



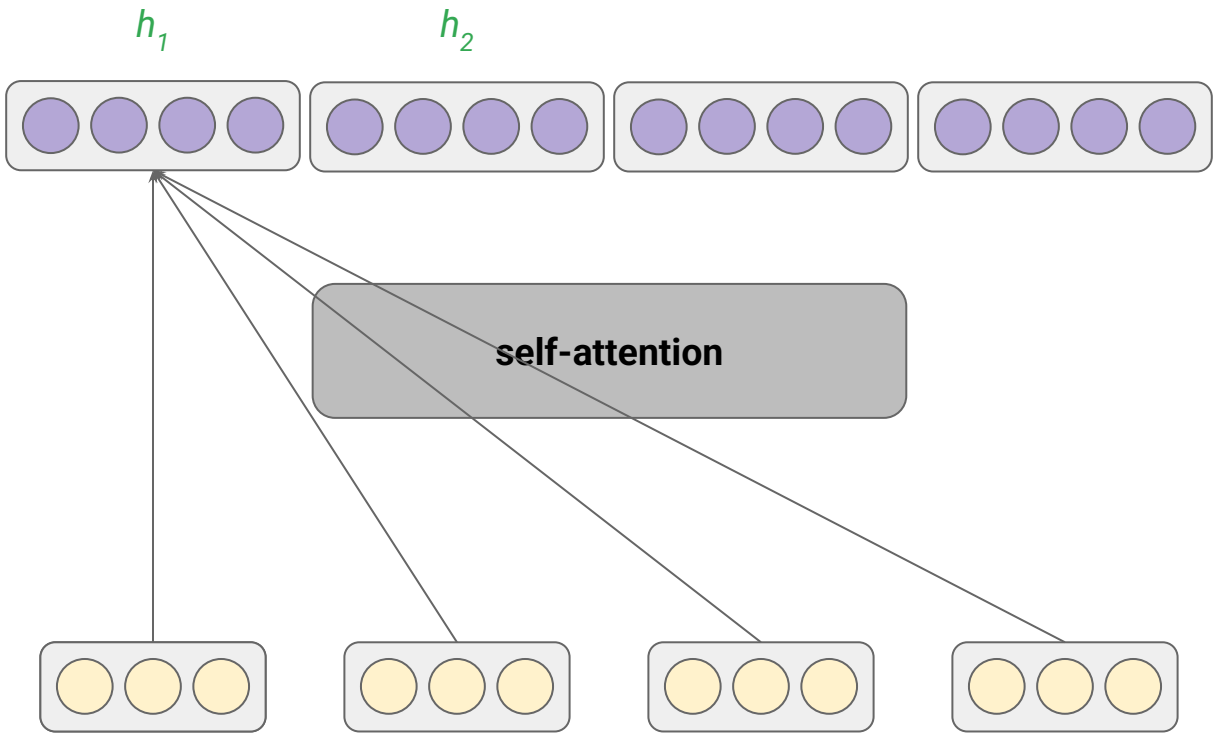
Transformer block (one layer)



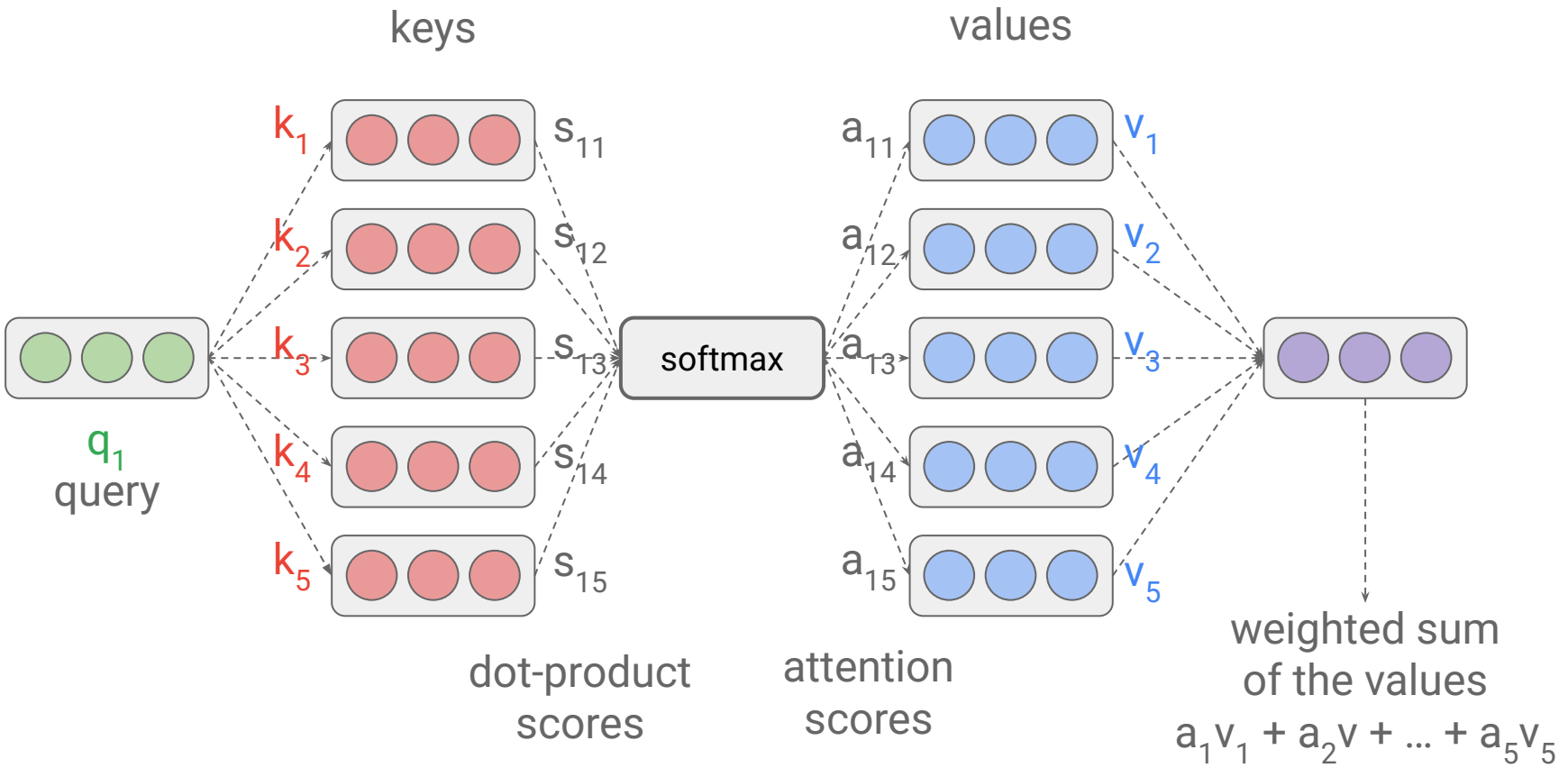
Transformer block



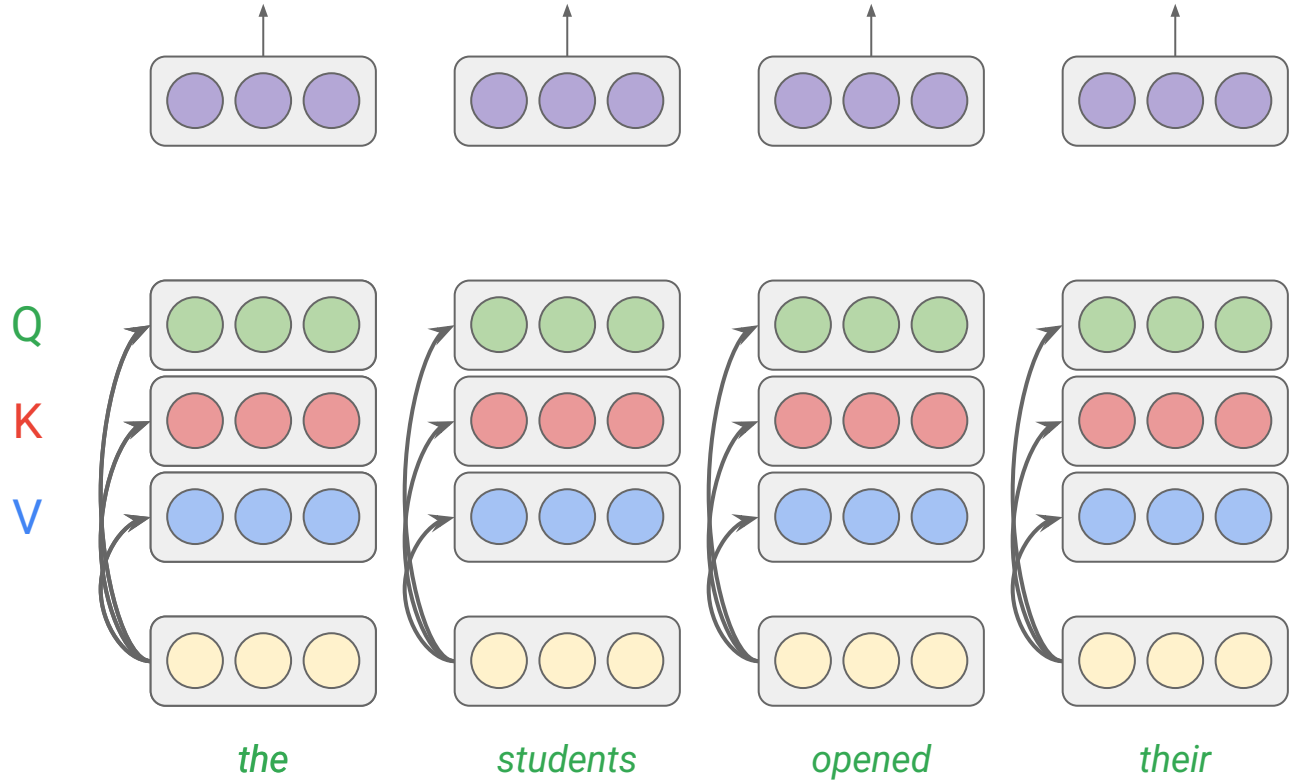
Self-attention



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)



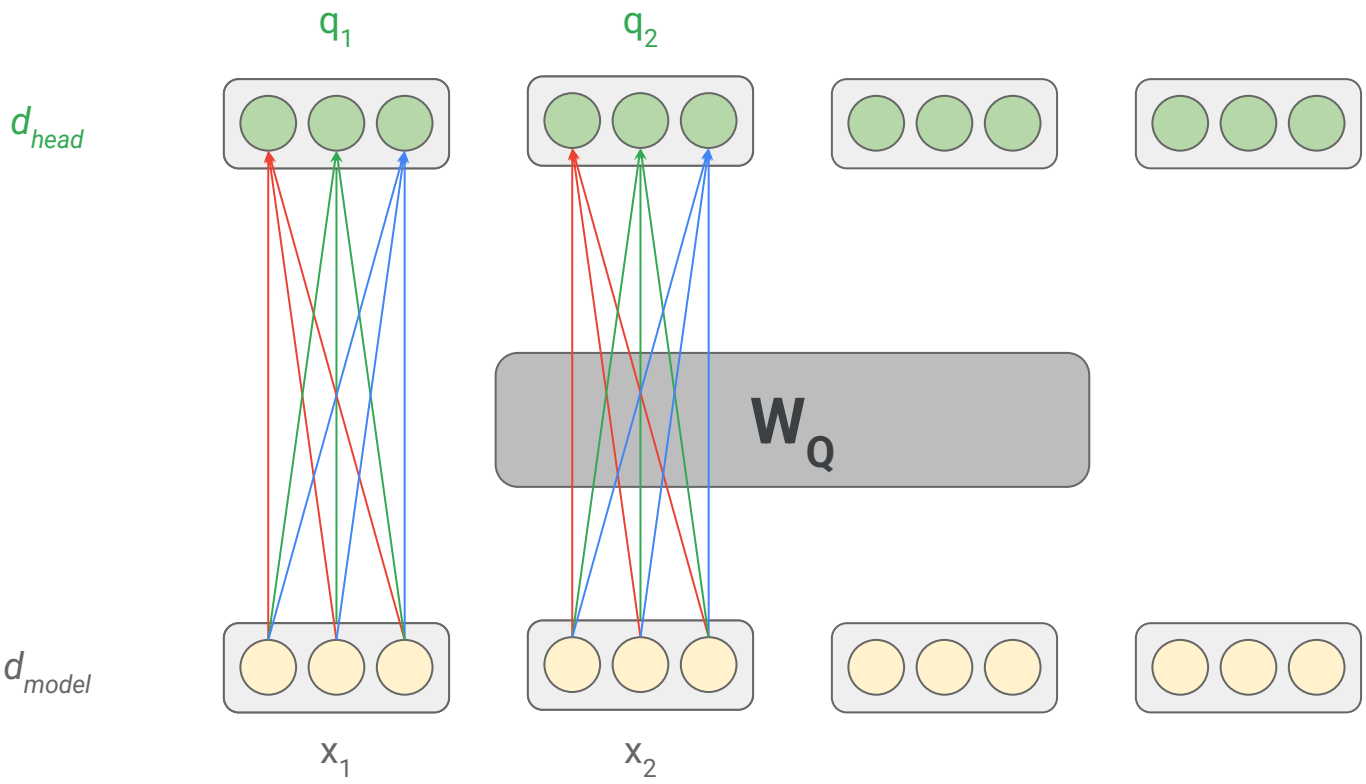
$$Q = X \cdot W_Q$$

$$K = X \cdot W_K$$

$$V = X \cdot W_V$$

linear projections

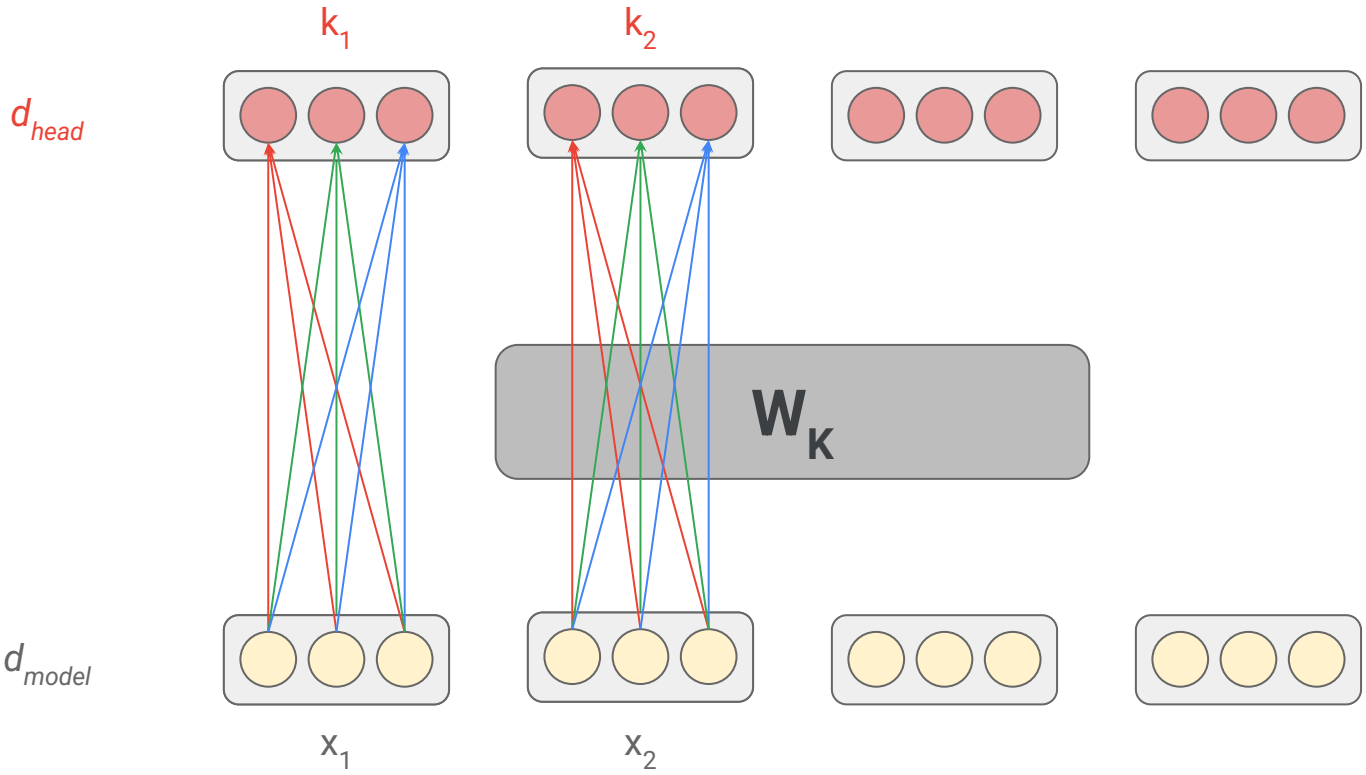
Query vectors



$$Q = X \cdot W_Q$$

linear projections

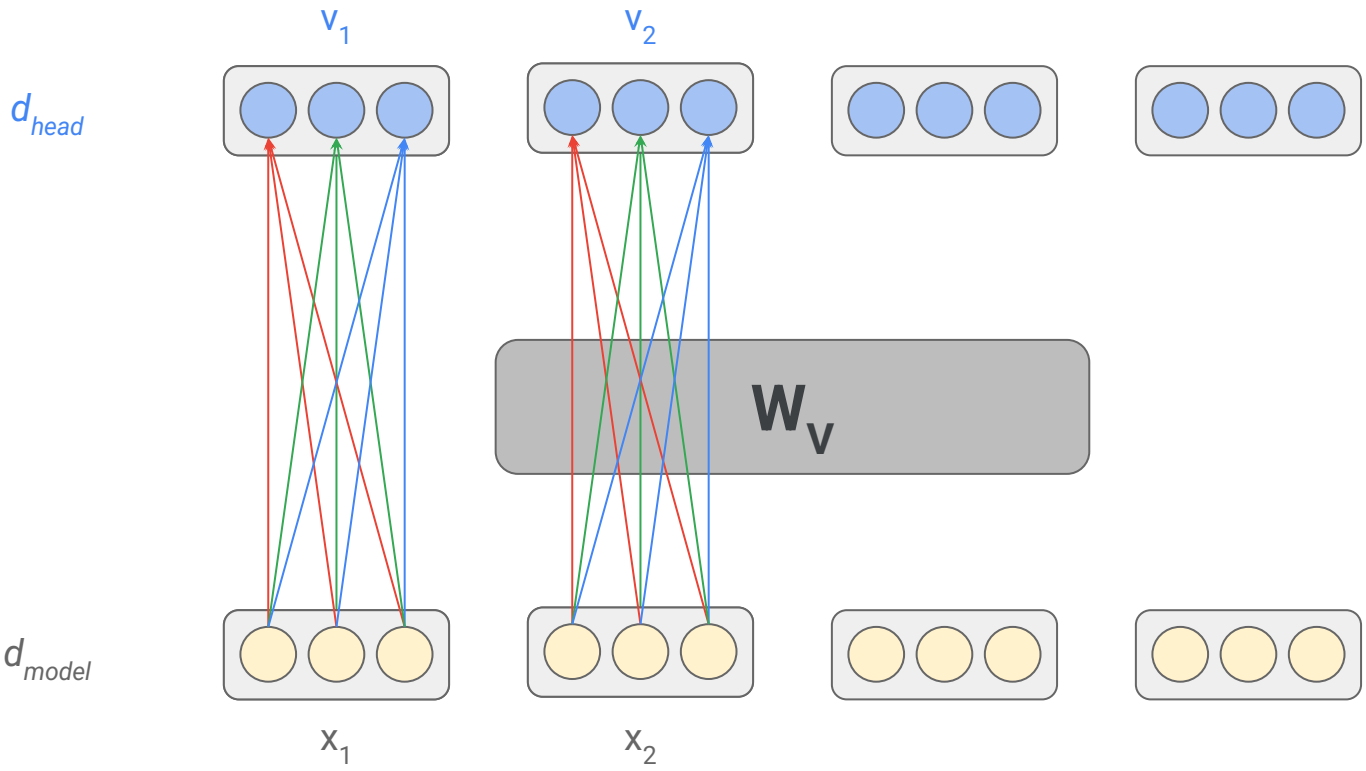
Key vectors



$$K = X \cdot W_K$$

linear projections

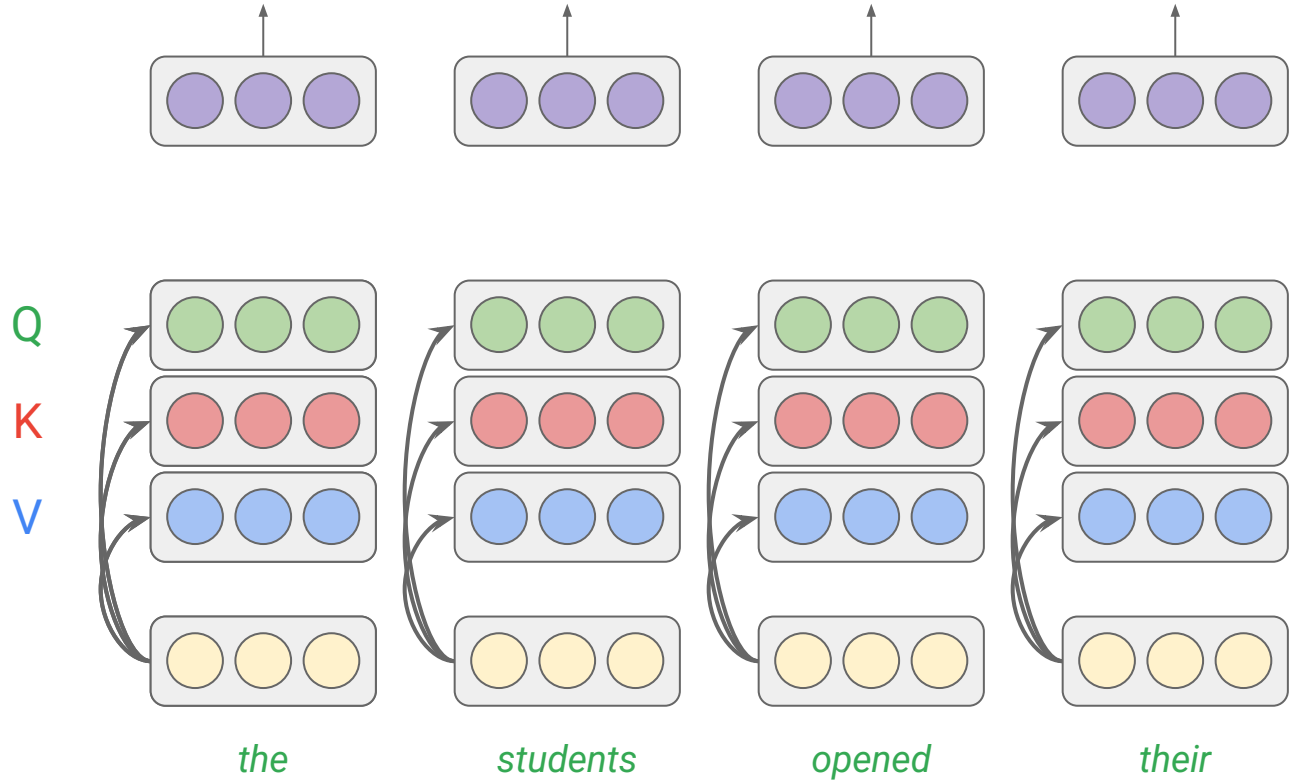
Value vectors



$$V = X \cdot W_V$$

linear projections

Attention (cont'd)



$$Q = X \cdot W_Q$$

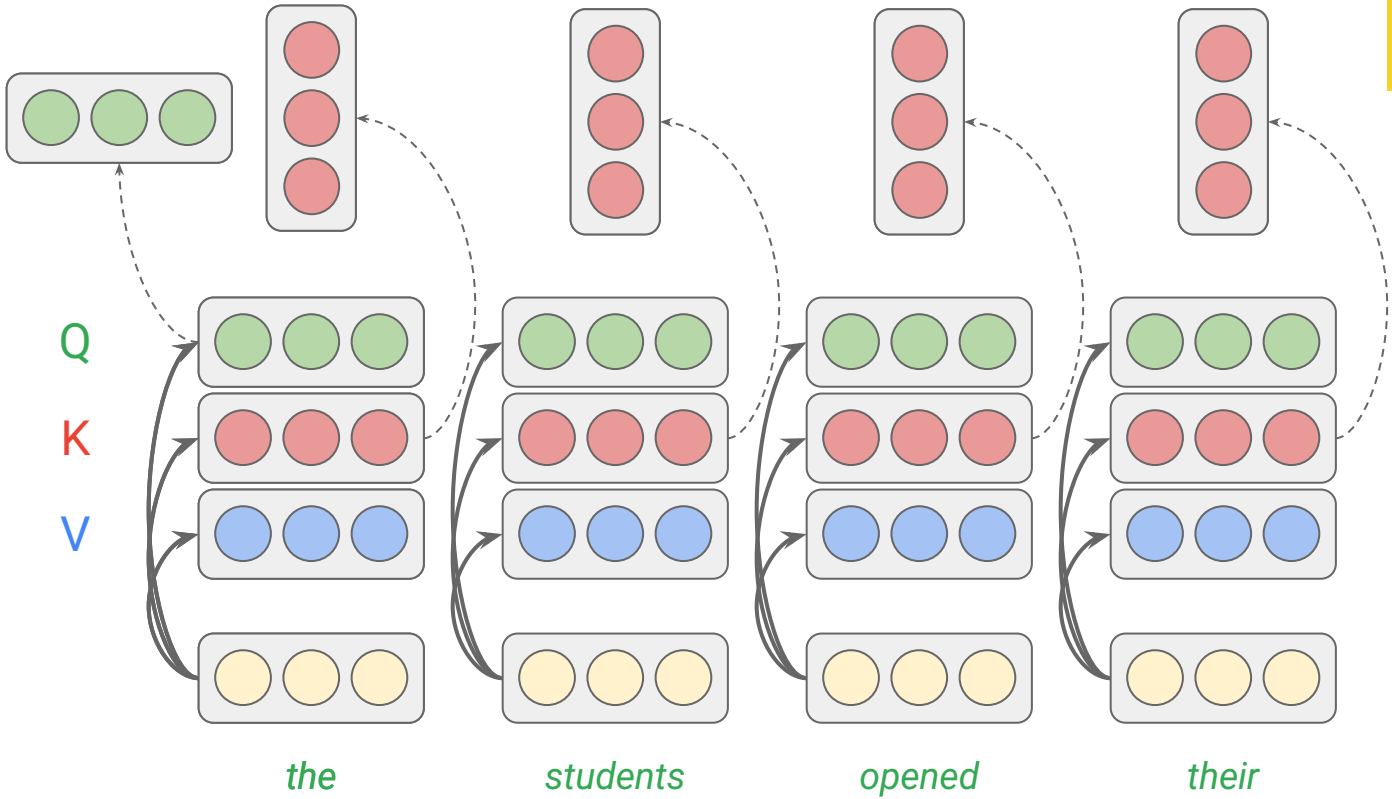
$$K = X \cdot W_K$$

$$V = X \cdot W_V$$

linear projections

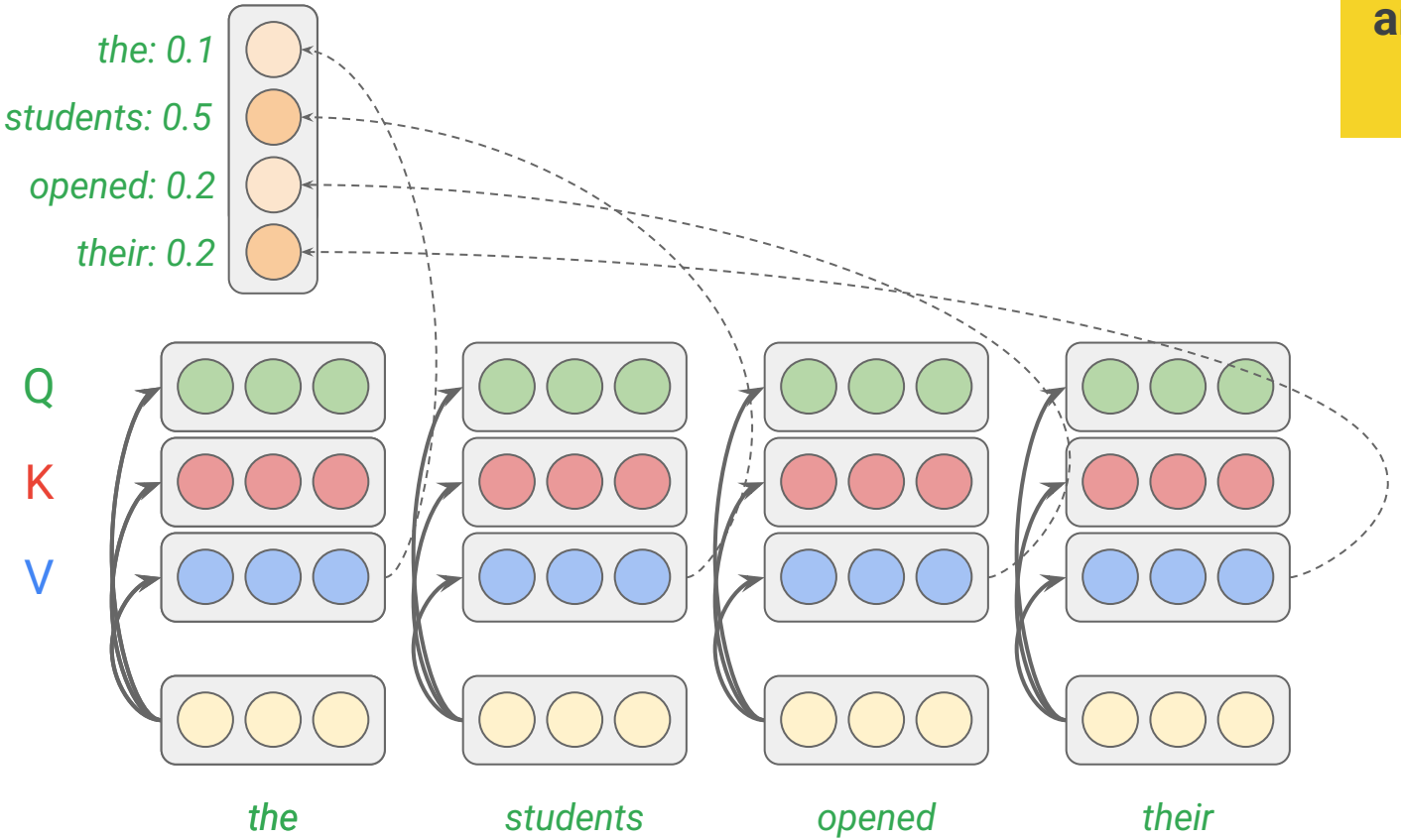
Self-attention (cont'd)

***all* computations are parallelized**



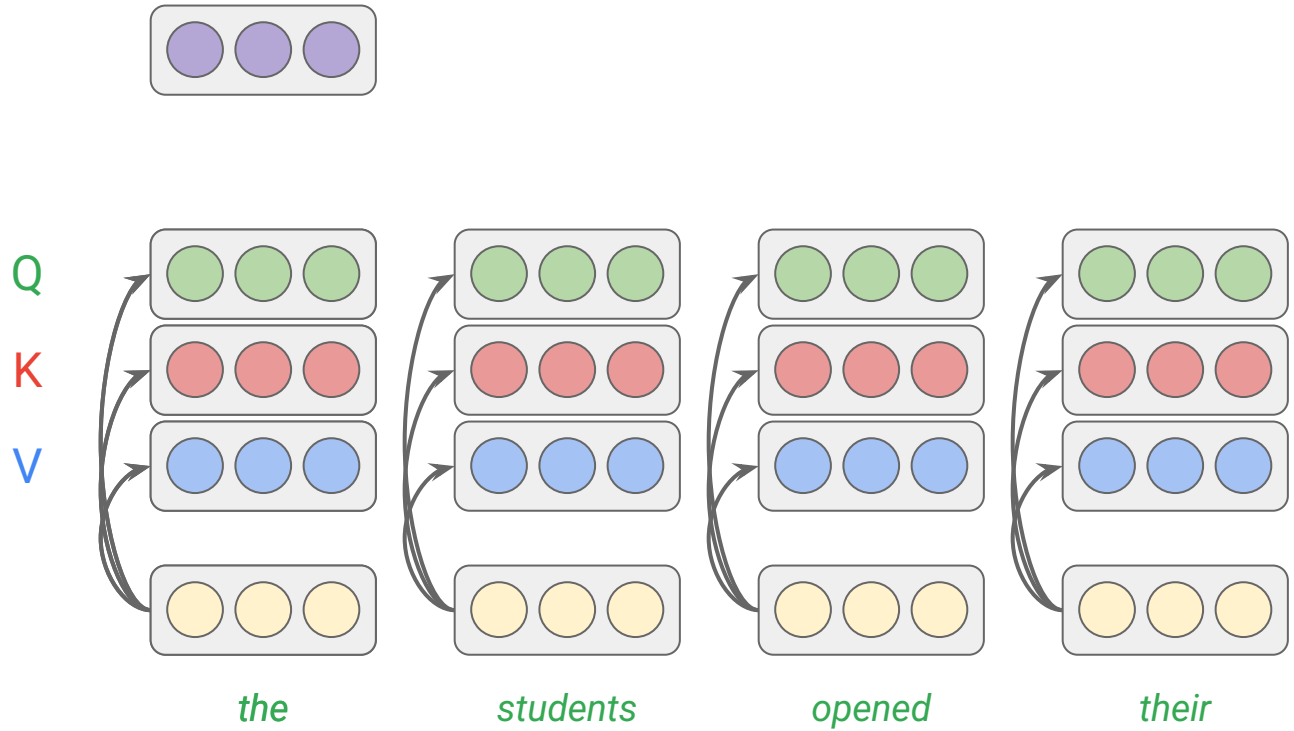
Self-attention (cont'd)

all computations are parallelized

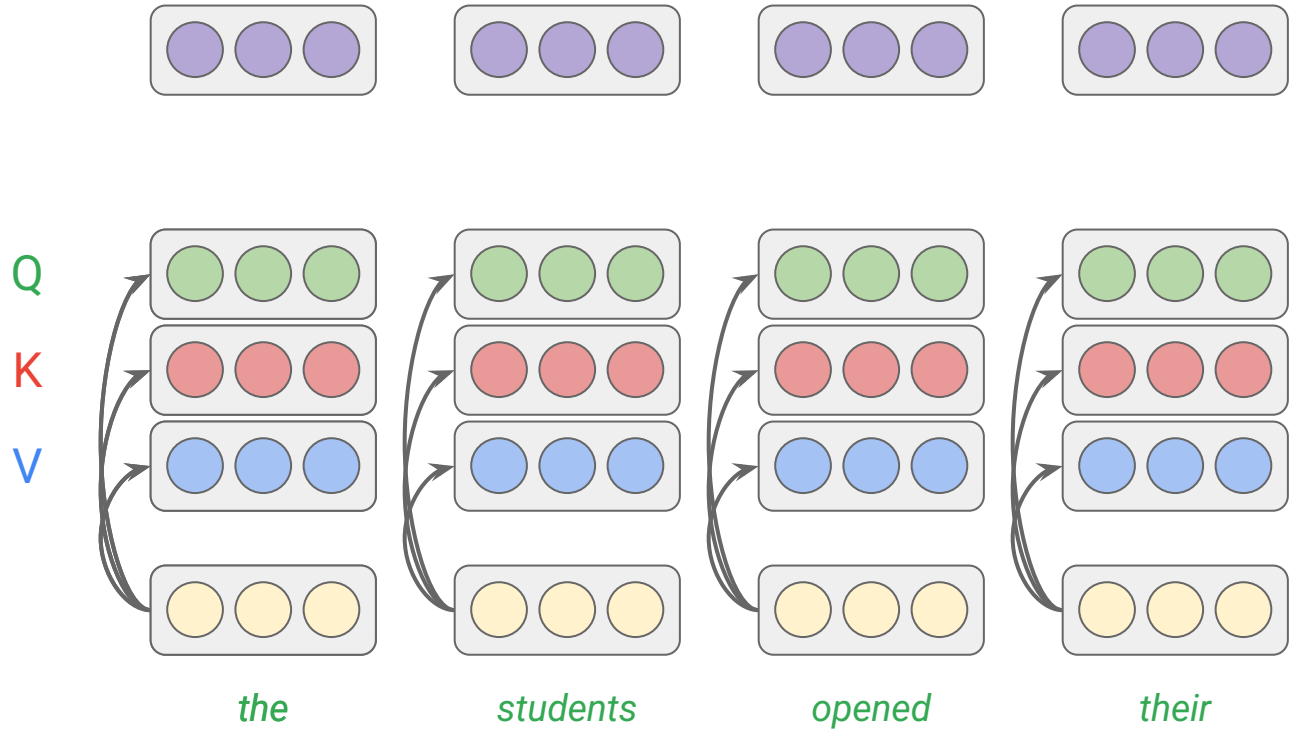


Self-attention (cont'd)

***all* computations are parallelized**

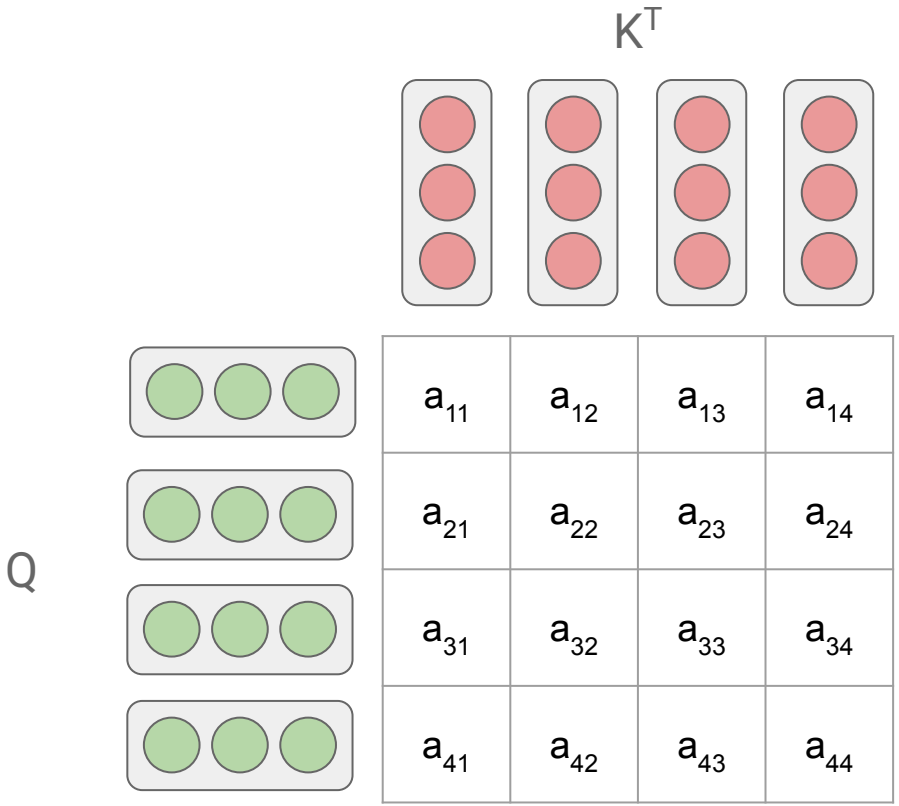


Self-attention (cont'd)



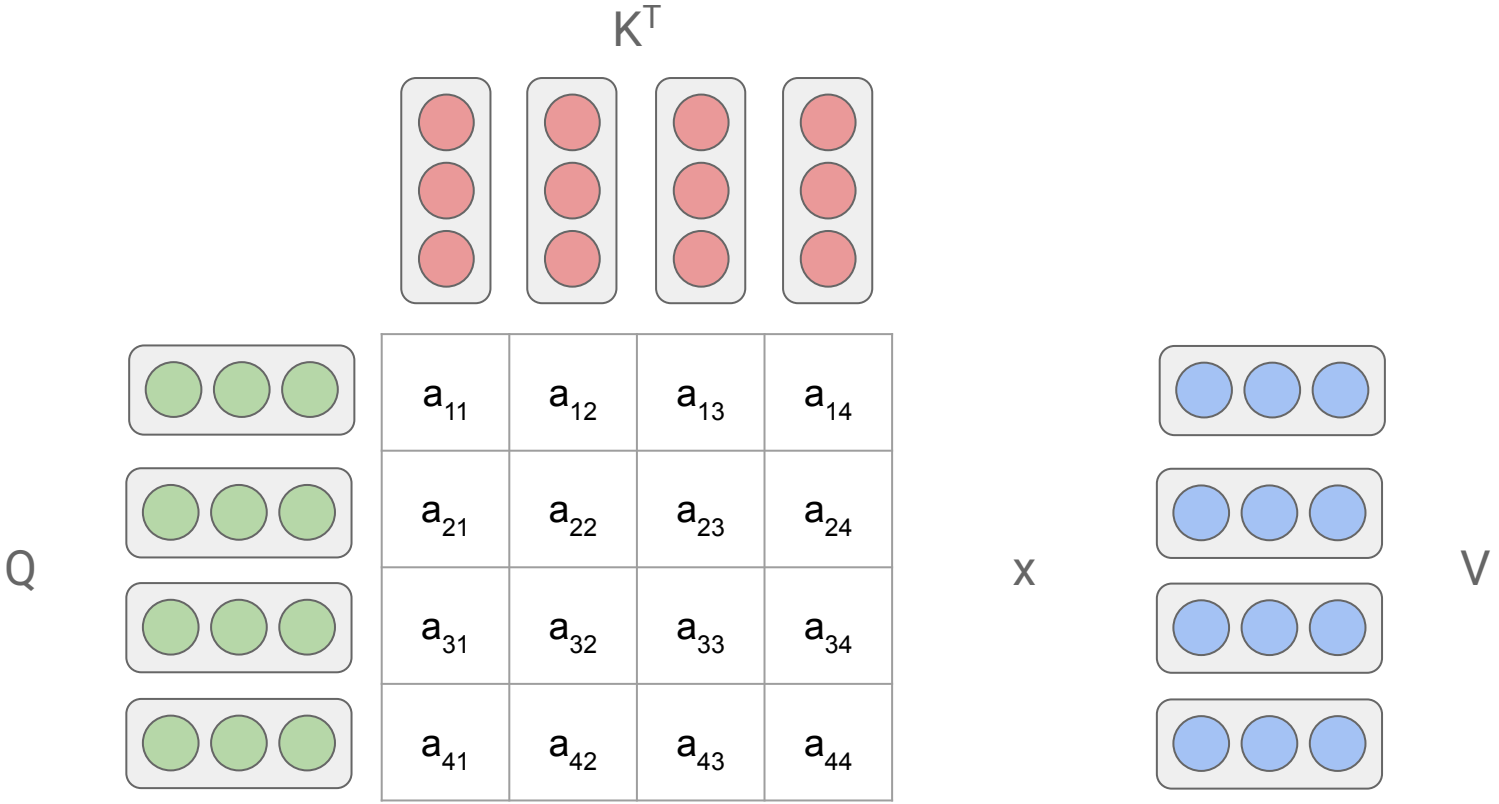
all **computations are parallelized during training and sequential during inference**

Quadratic complexity



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

Quadratic complexity



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,

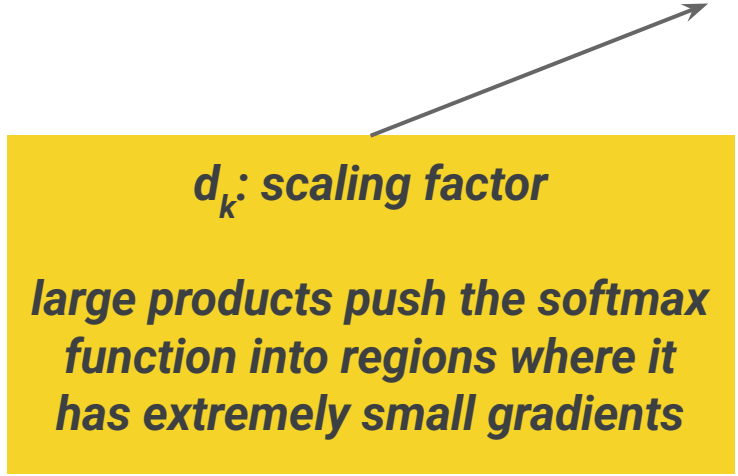
$$(AV)_{1:}^{\top} = \begin{bmatrix} a_{11}v_{11} + a_{12}v_{21} + a_{13}v_{31} + a_{14}v_{41} \\ a_{11}v_{12} + a_{12}v_{22} + a_{13}v_{32} + a_{14}v_{42} \\ a_{11}v_{13} + a_{12}v_{23} + a_{13}v_{33} + a_{14}v_{43} \end{bmatrix}.$$

Factor by coefficients,

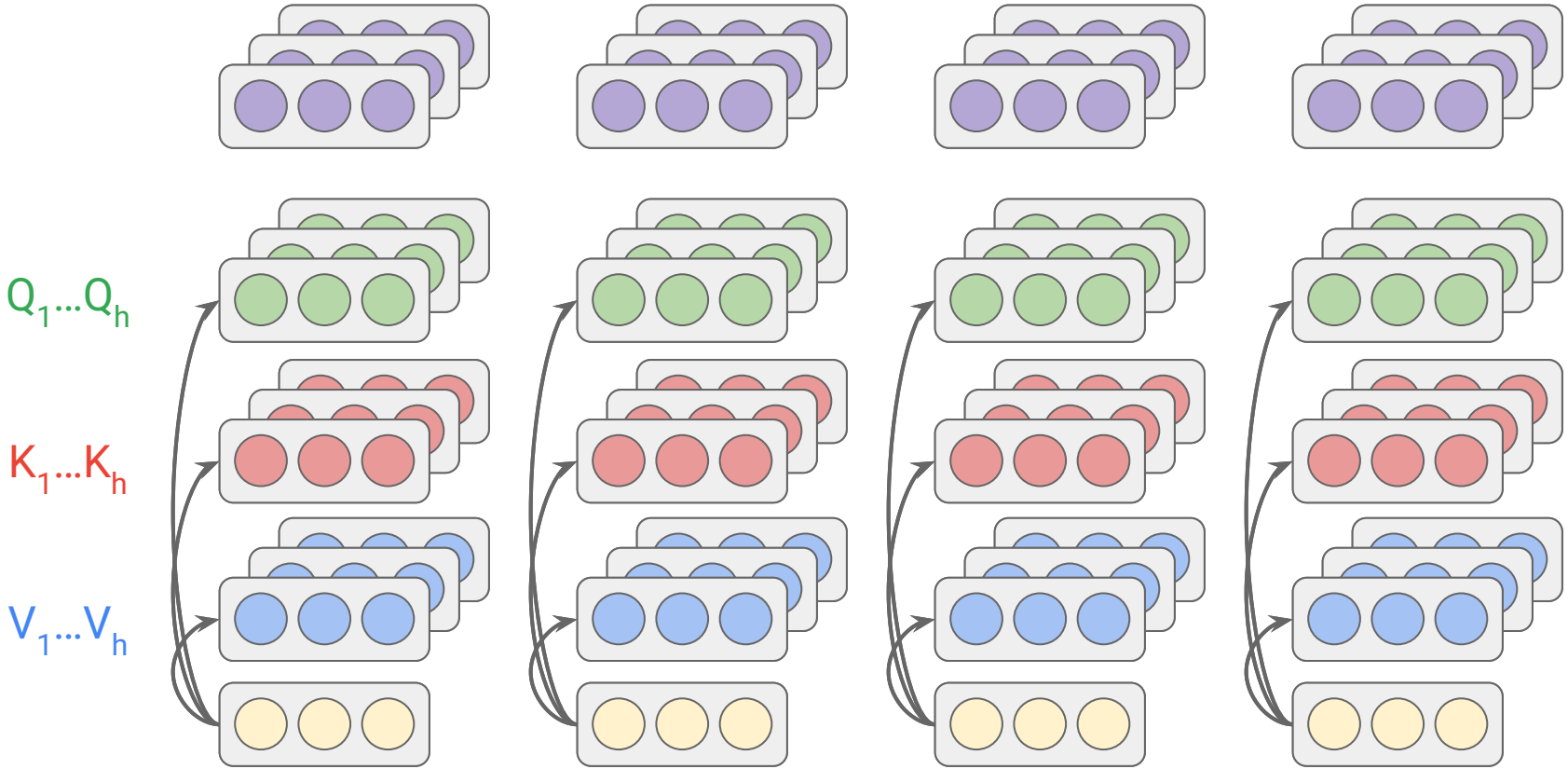
$$(AV)_{1:}^{\top} = a_{11} \begin{bmatrix} v_{11} \\ v_{12} \\ v_{13} \end{bmatrix} + a_{12} \begin{bmatrix} v_{21} \\ v_{22} \\ v_{23} \end{bmatrix} + a_{13} \begin{bmatrix} v_{31} \\ v_{32} \\ v_{33} \end{bmatrix} + a_{14} \begin{bmatrix} v_{41} \\ v_{42} \\ v_{43} \end{bmatrix}.$$

All computations are parallelized

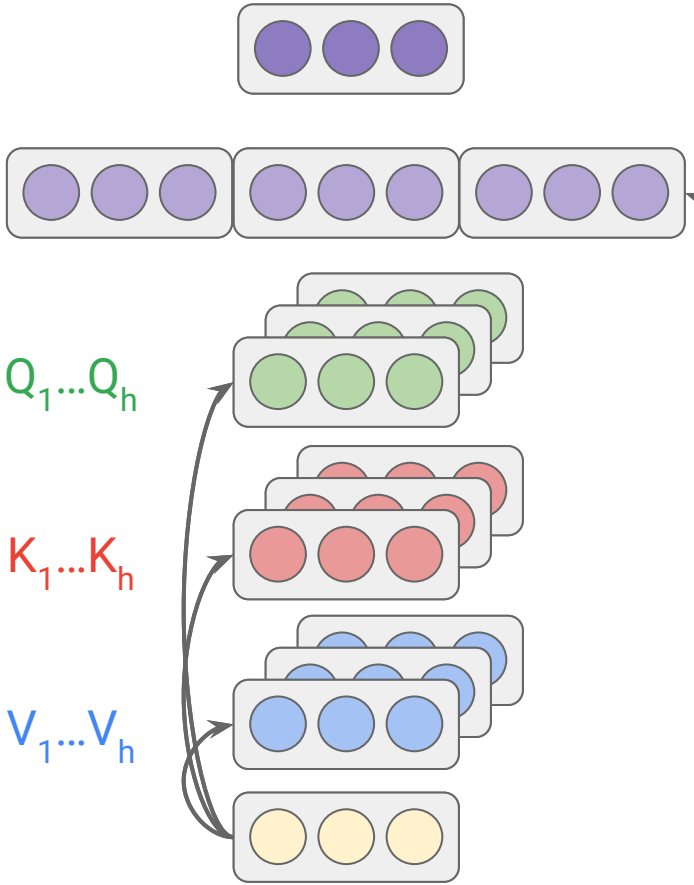
$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$



Multi-head attention

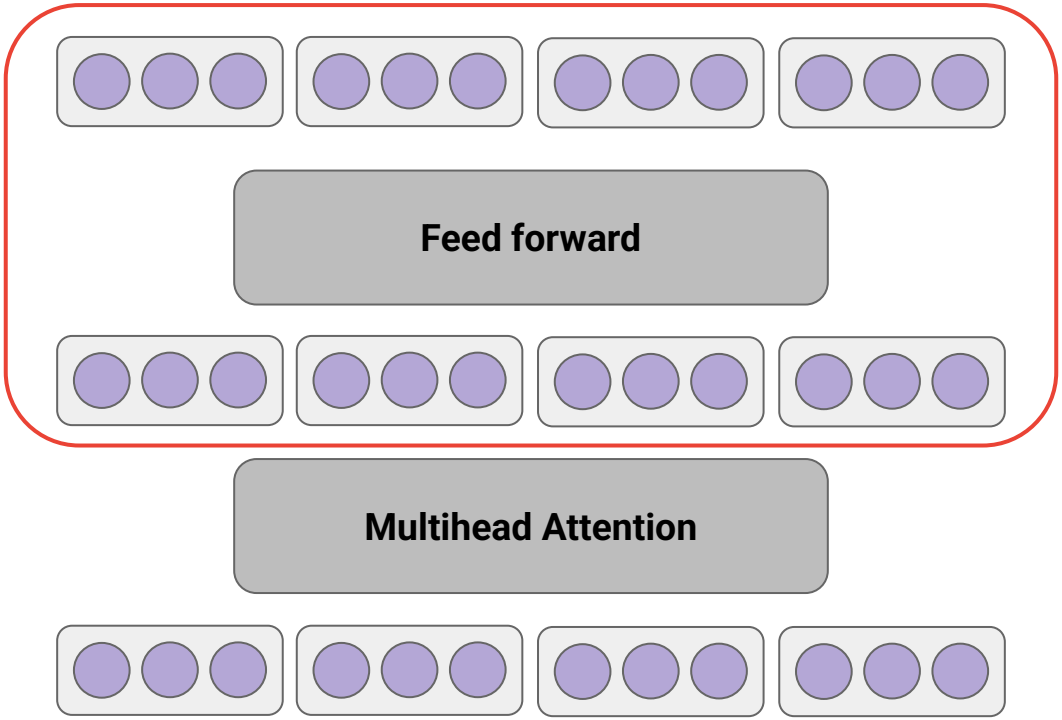


Multi-head attention (cont'd)

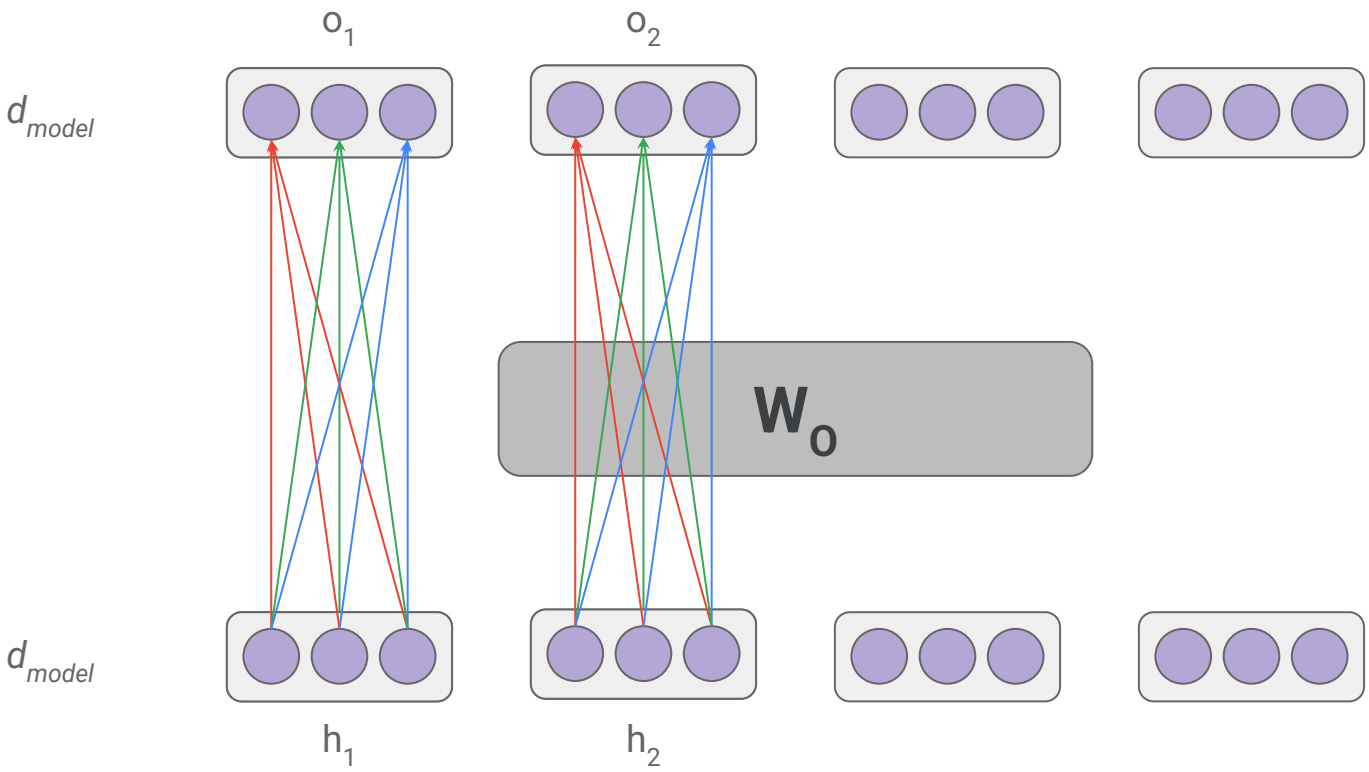


These output values are concatenated and once again projected

Transformer block (cont'd)



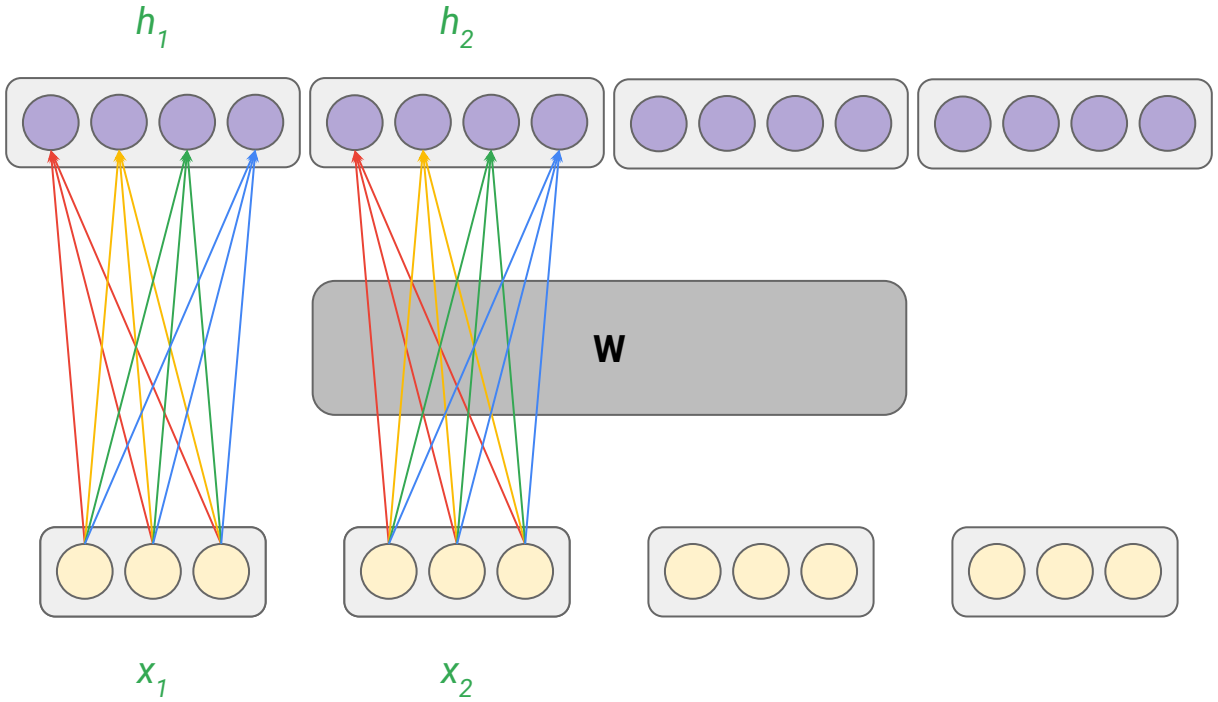
output vectors



$$O = H \cdot W_O$$

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:

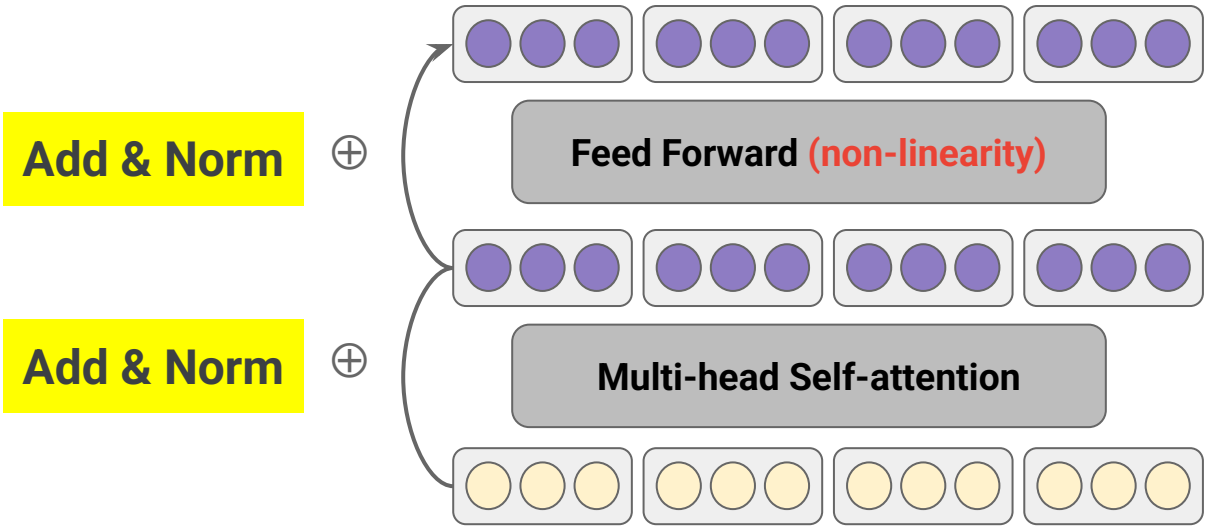
$$H = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \\ w_{41} & w_{42} & w_{43} \end{bmatrix} \begin{bmatrix} x_{11} & x_{21} \\ x_{12} & x_{22} \\ x_{13} & x_{23} \end{bmatrix}$$

This results in:

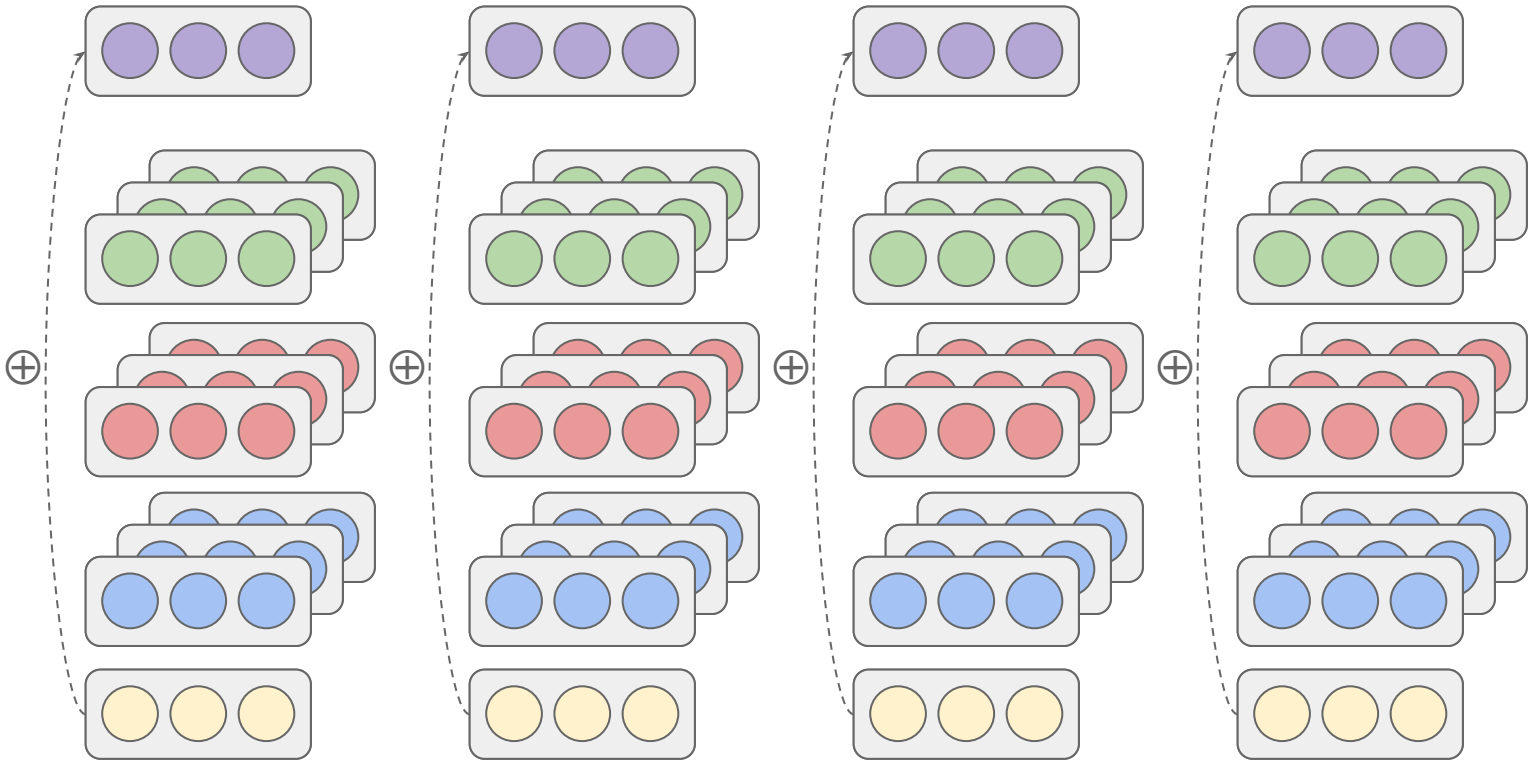
h_1 h_2

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

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

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

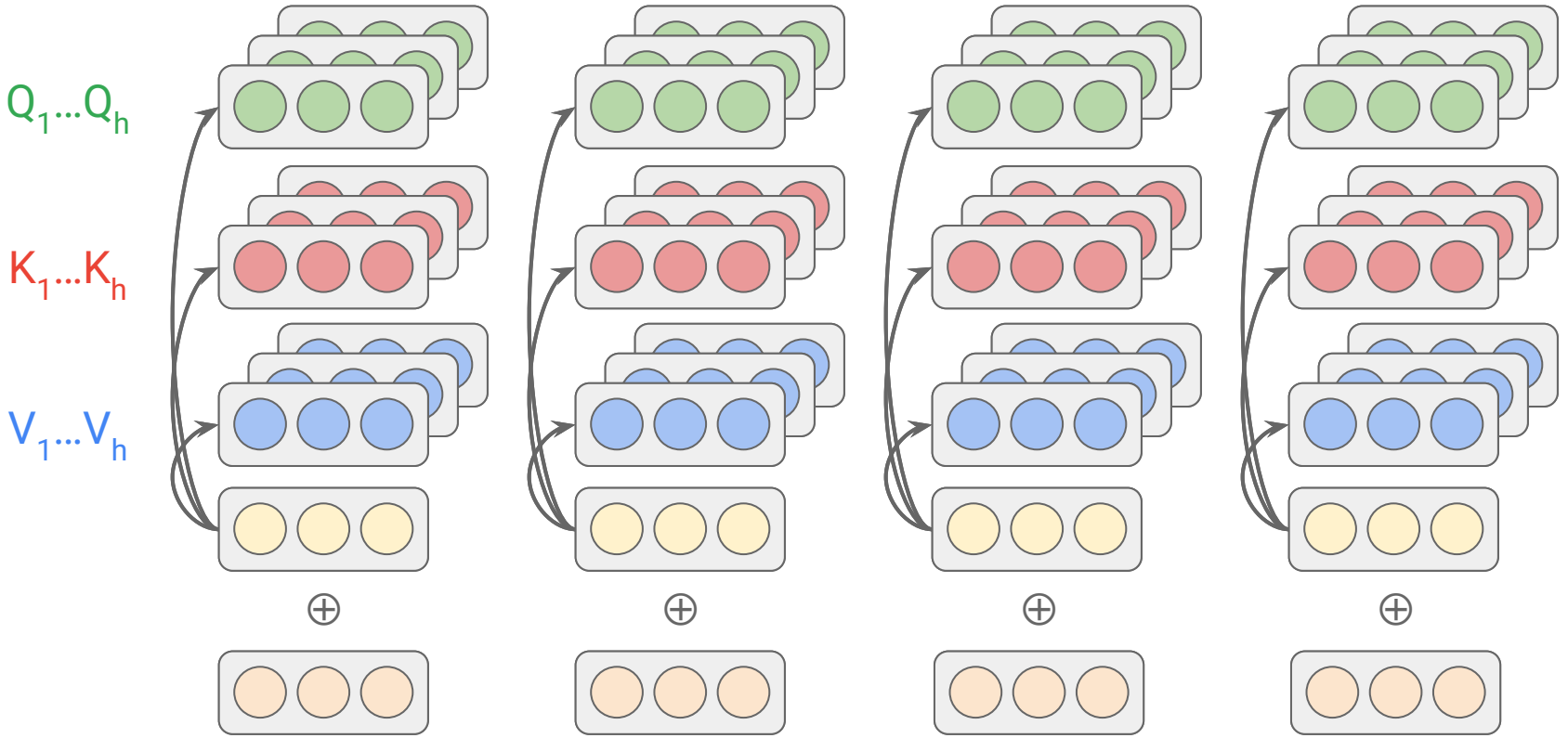
ReLU (Rectified Linear Unit)

Sinusoidal positional encoding

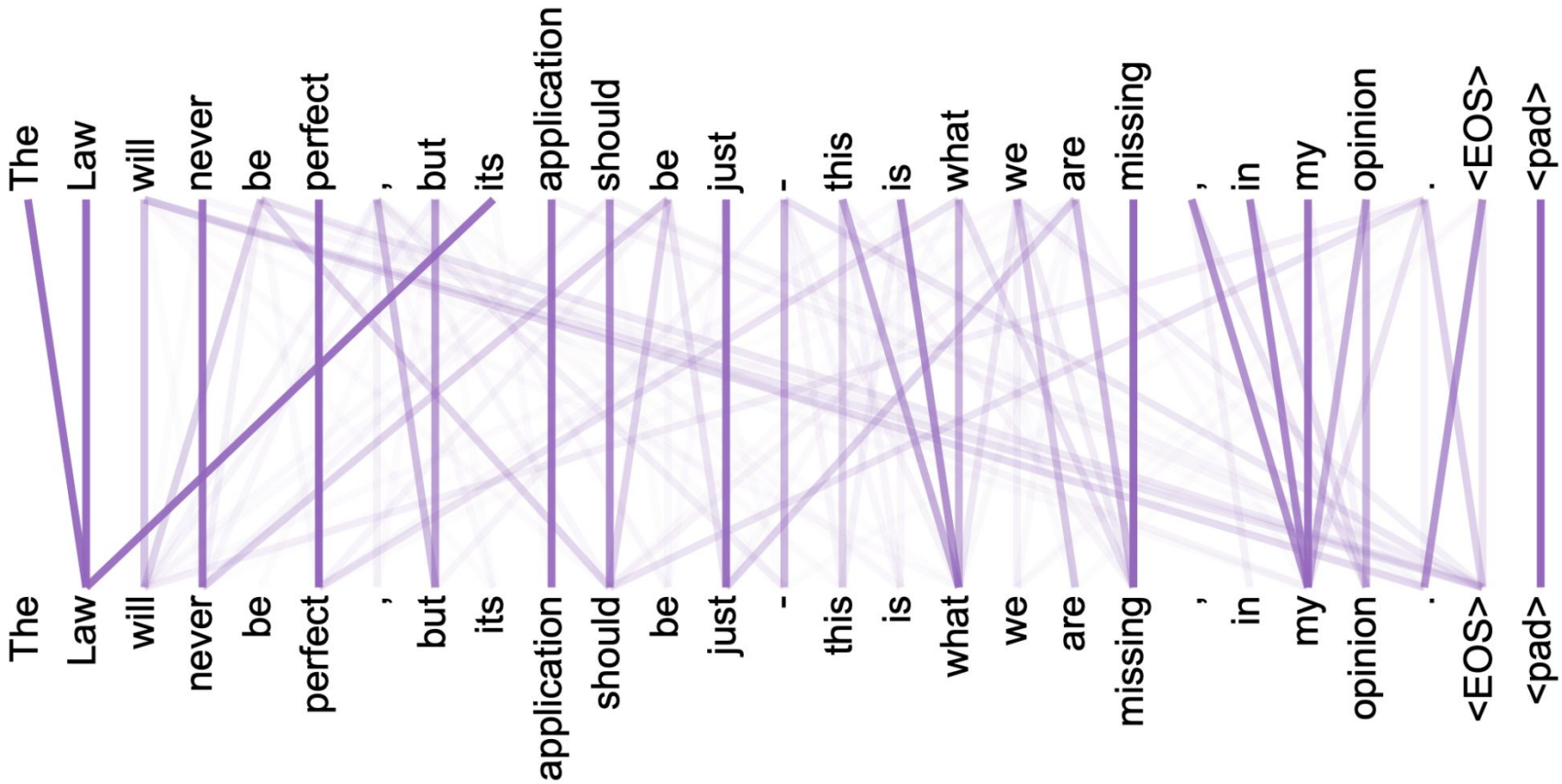
$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$

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

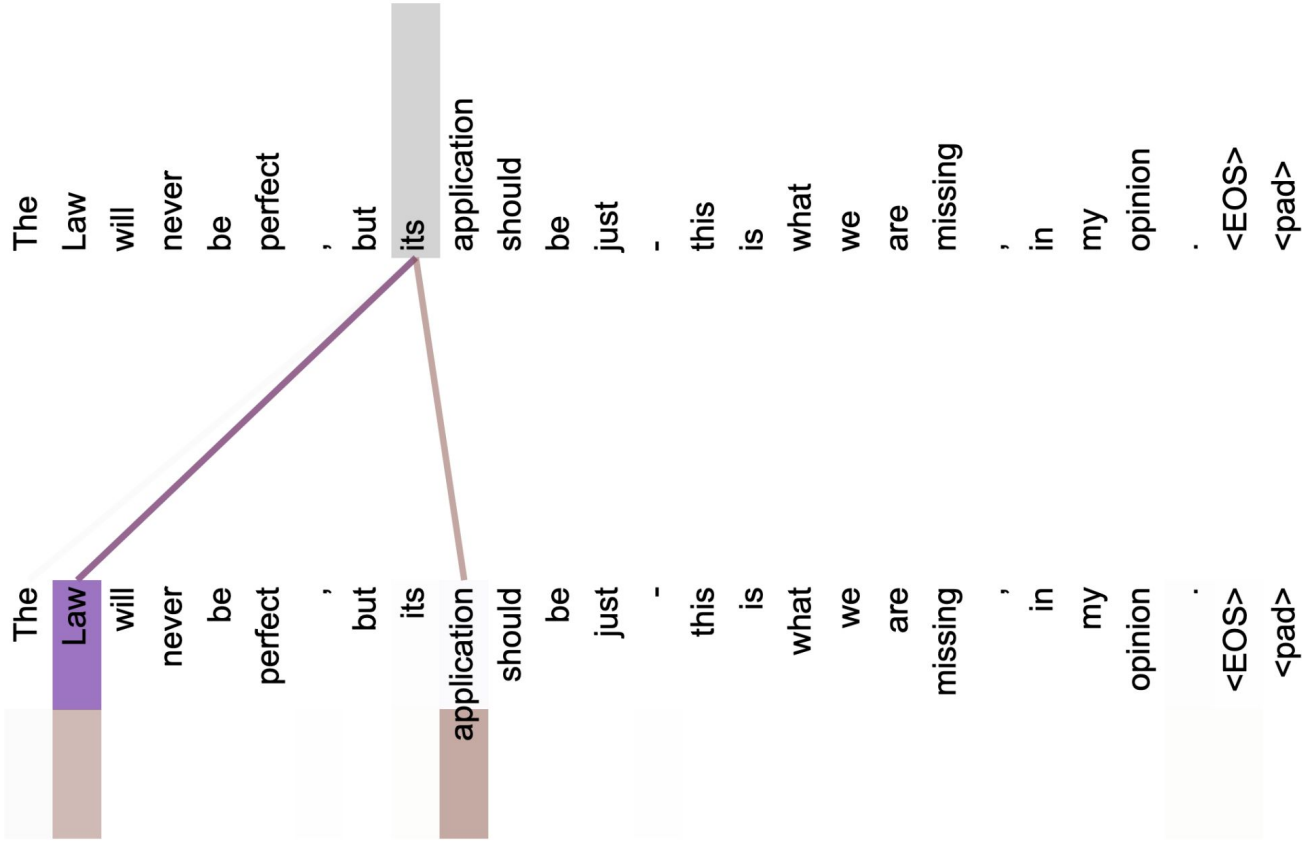
Positional Encoding (cont'd)



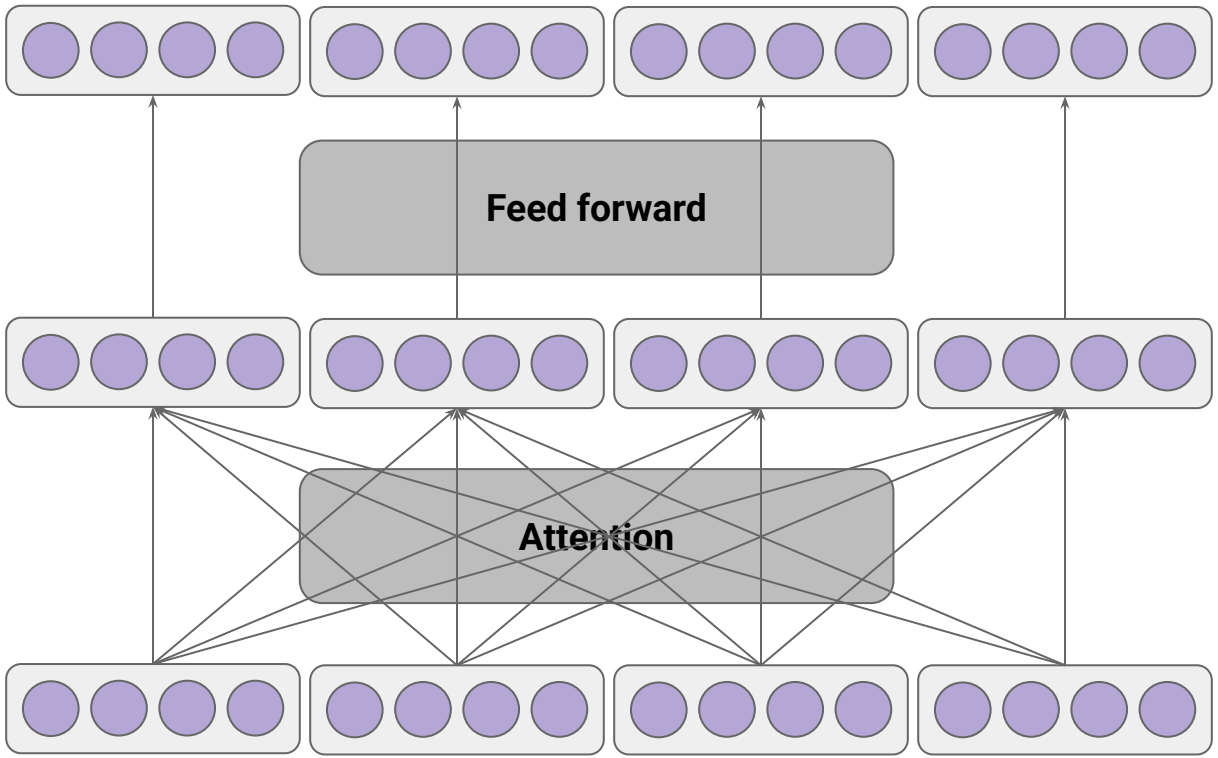
Attention visualizations



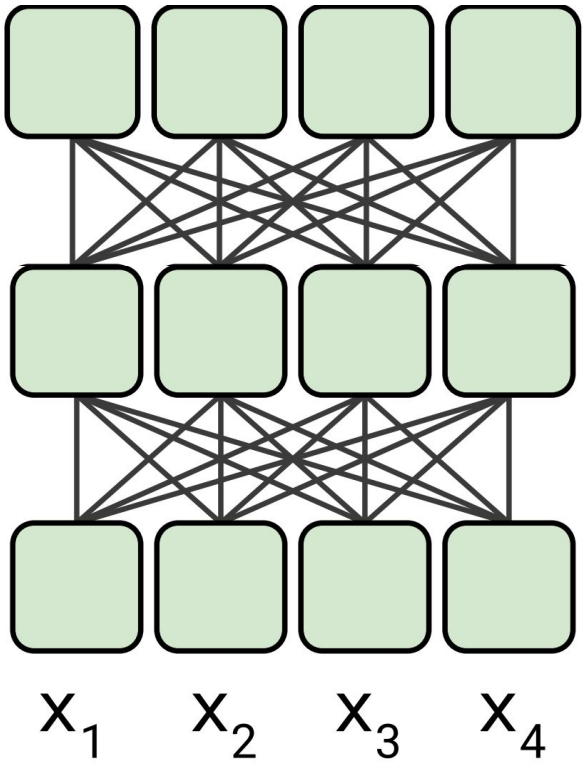
Attention visualizations (cont'd)



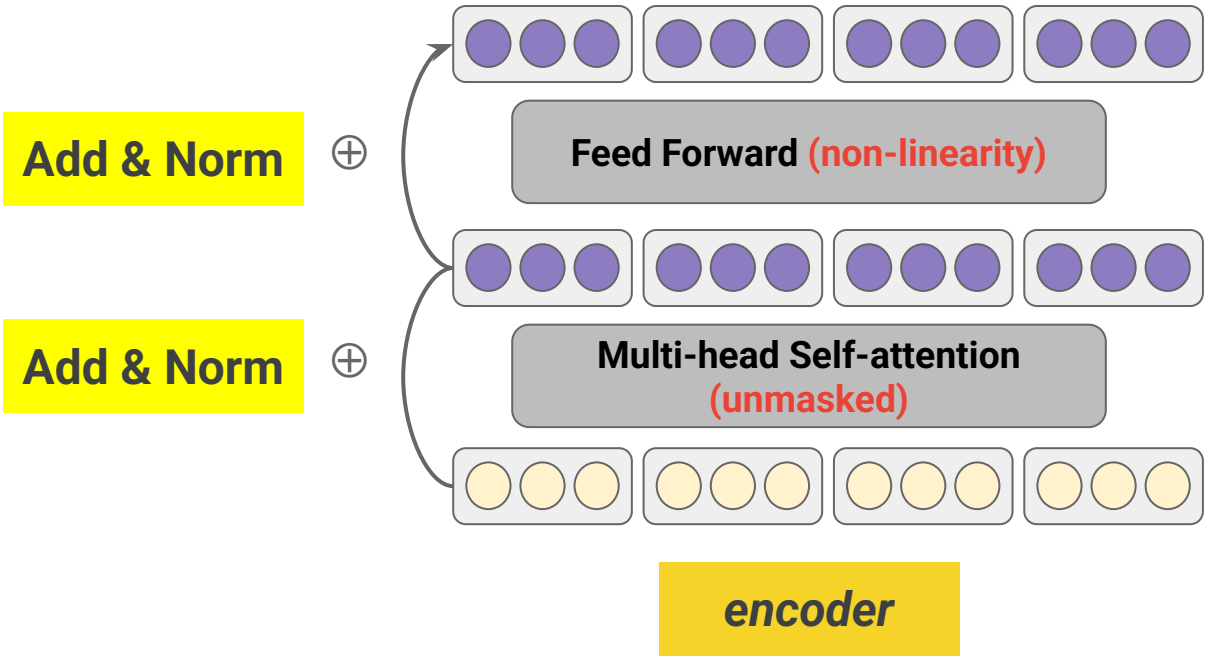
Putting it all together



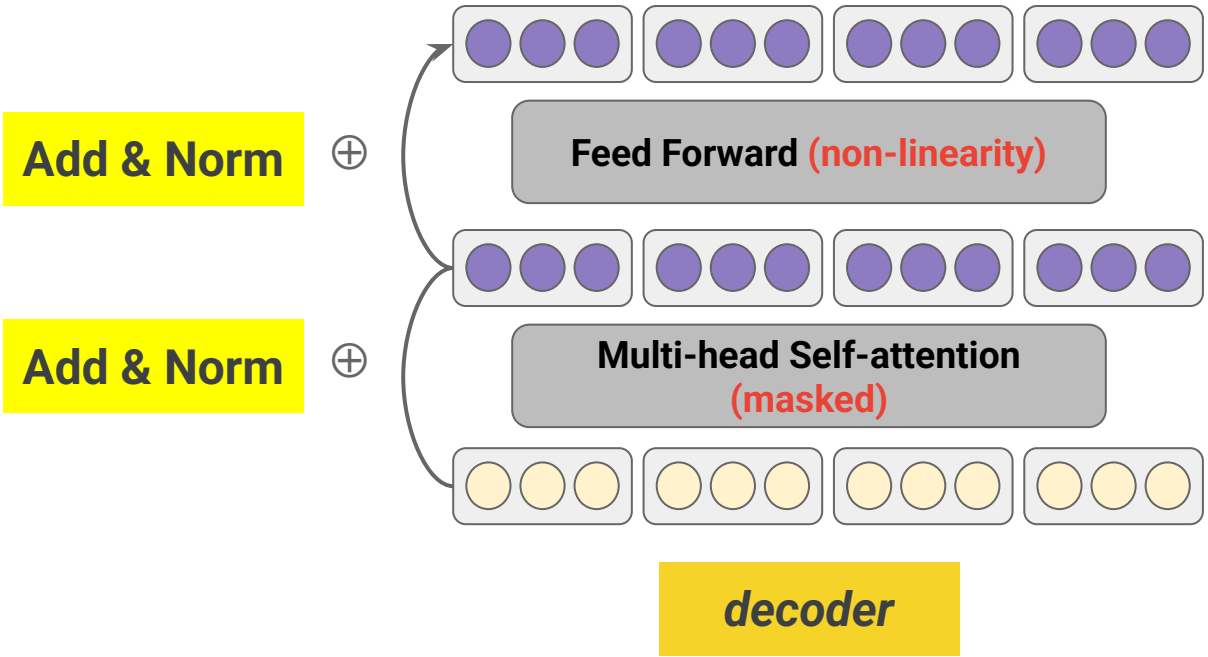
Encoder only



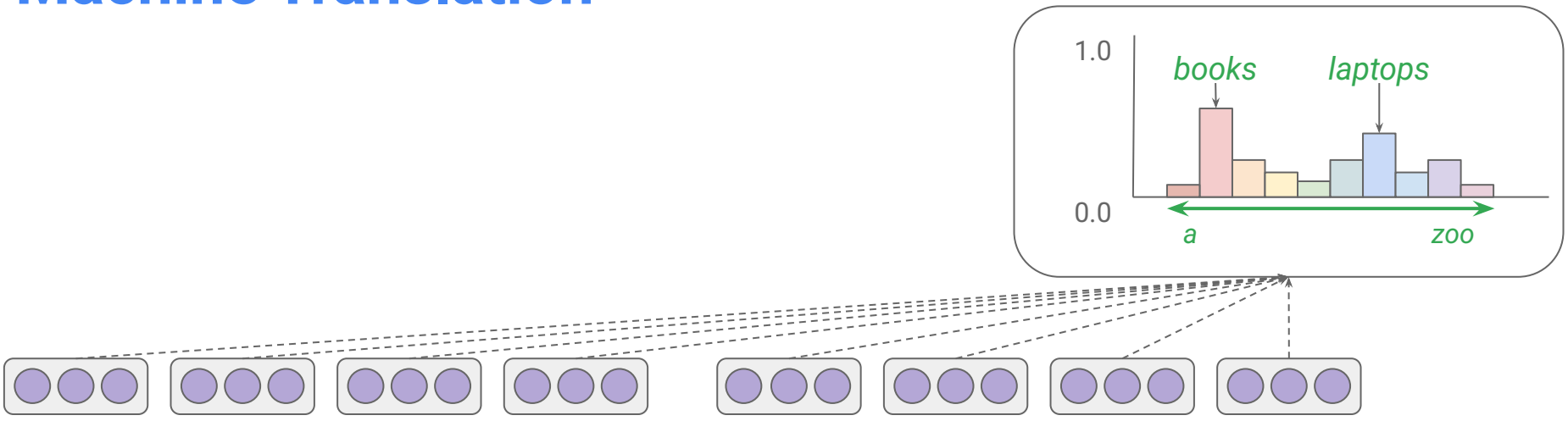
Encoder (one layer)



Decoder (one layer)



Machine Translation

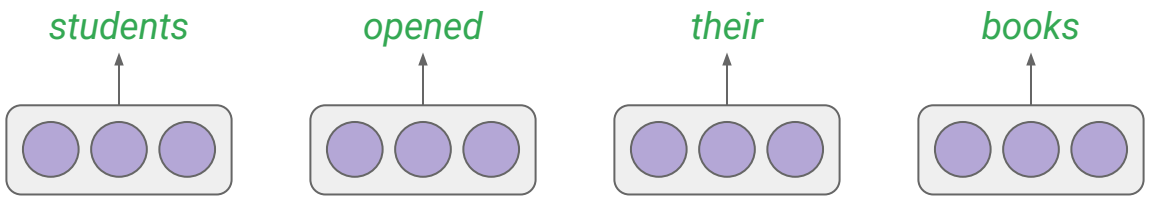


les étudiants ont ouvert leurs livres

the *students* *opened* *their*

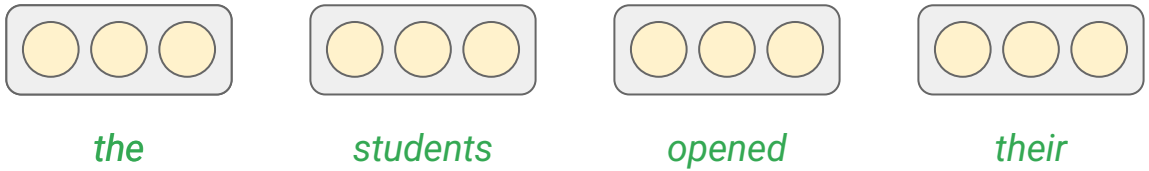
encoder ***decoder***

Transformer decoder

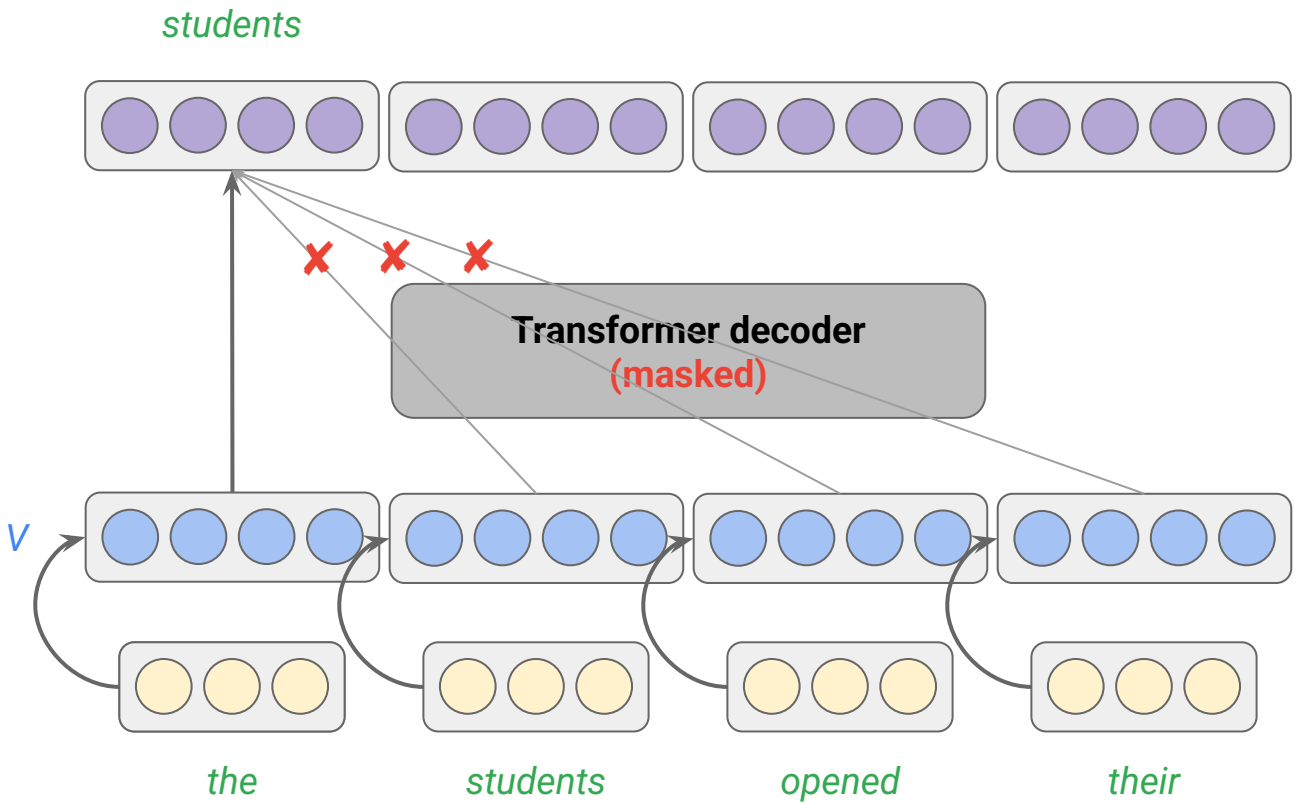


the architecture used in frontier LLMs

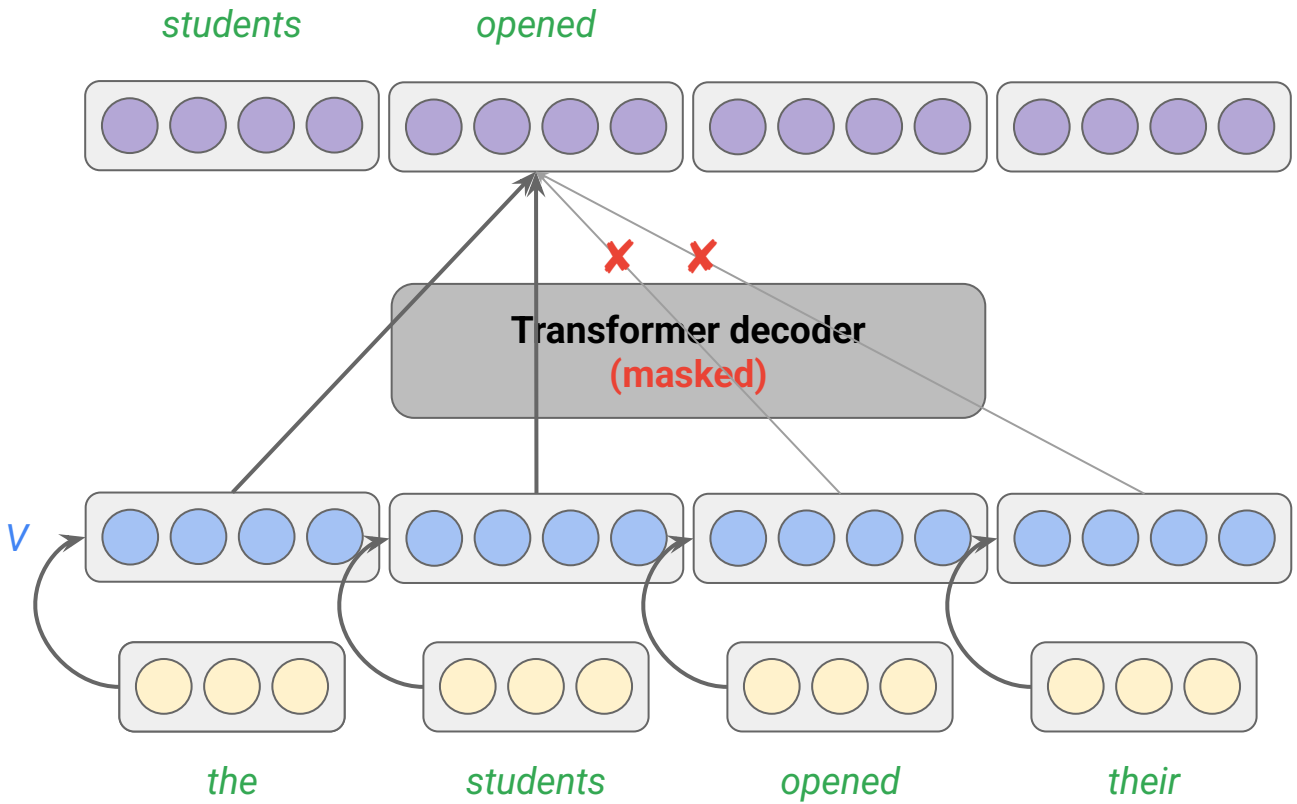
Transformer decoder
(masked)



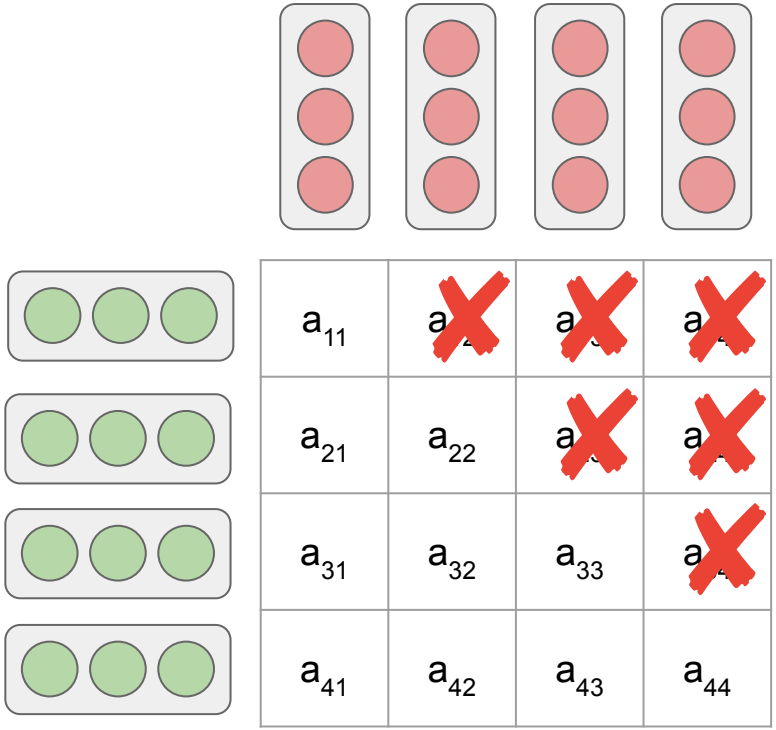
Transformer decoder (cont'd)



Transformer decoder (cont'd)

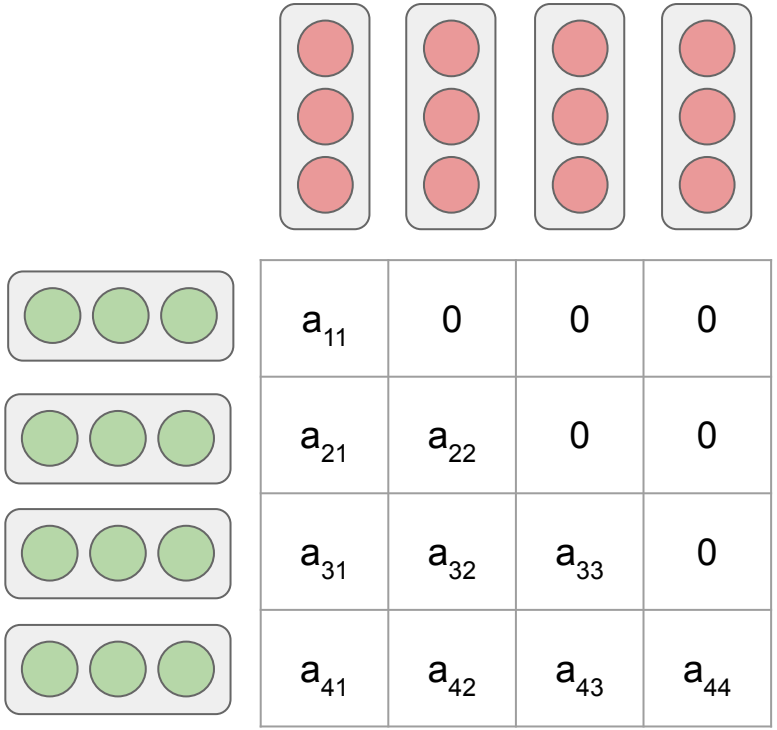


Self-attention in the decoder



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

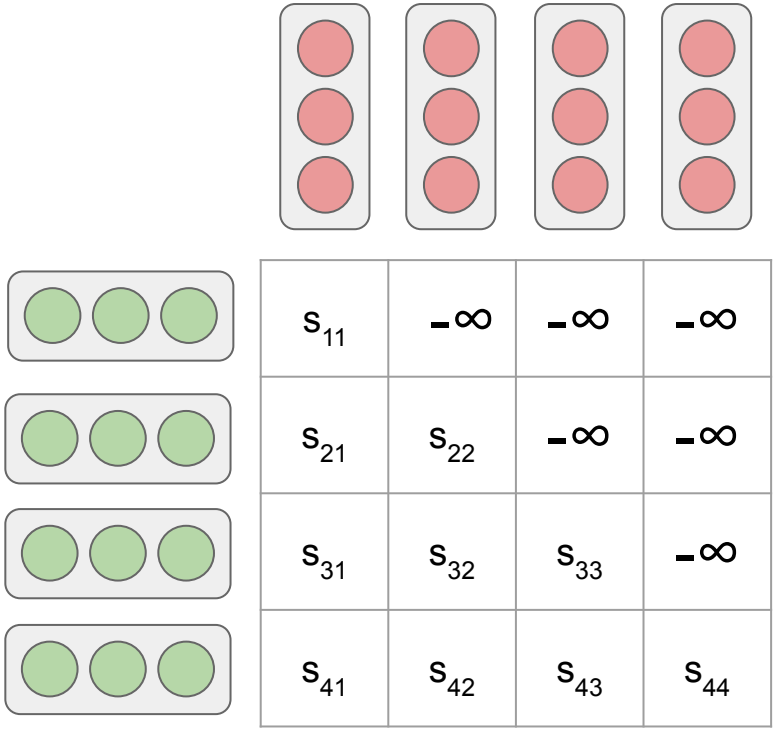
Self-attention in the decoder (cont'd)



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}

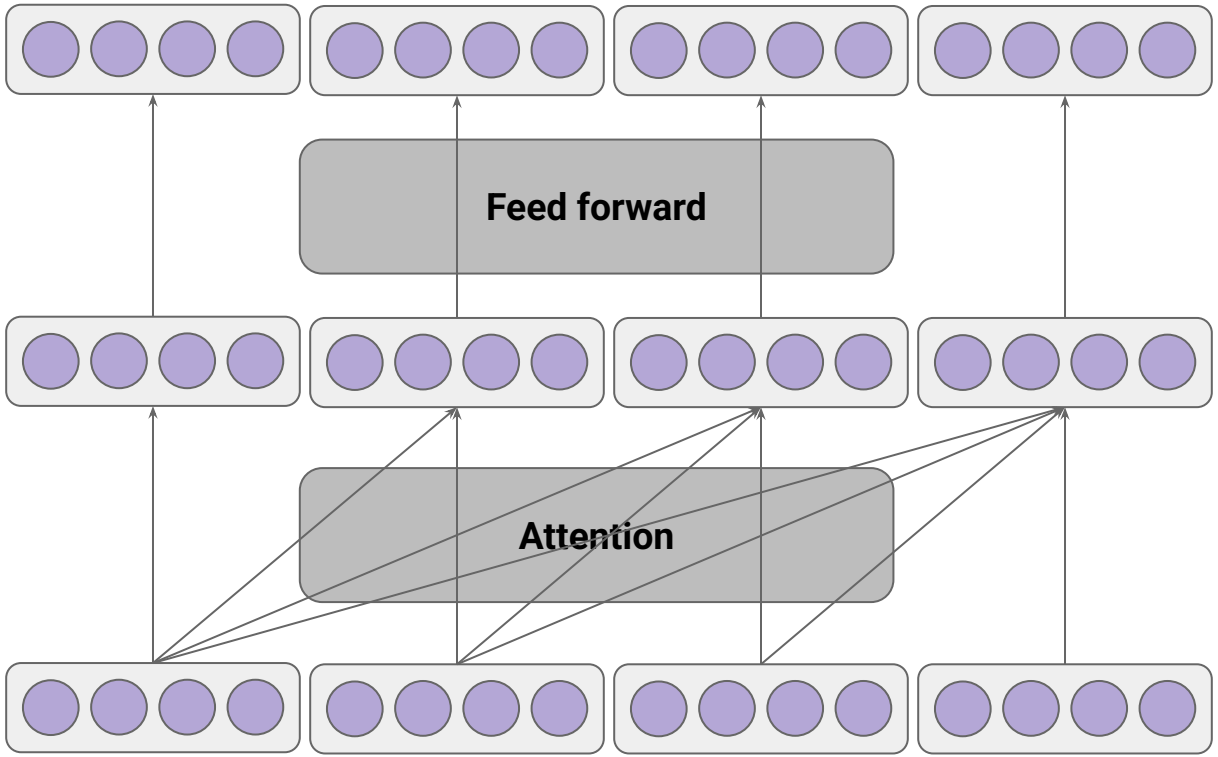
masking out all values in the input of the softmax which correspond to illegal connections

Self-attention in the decoder (cont'd)

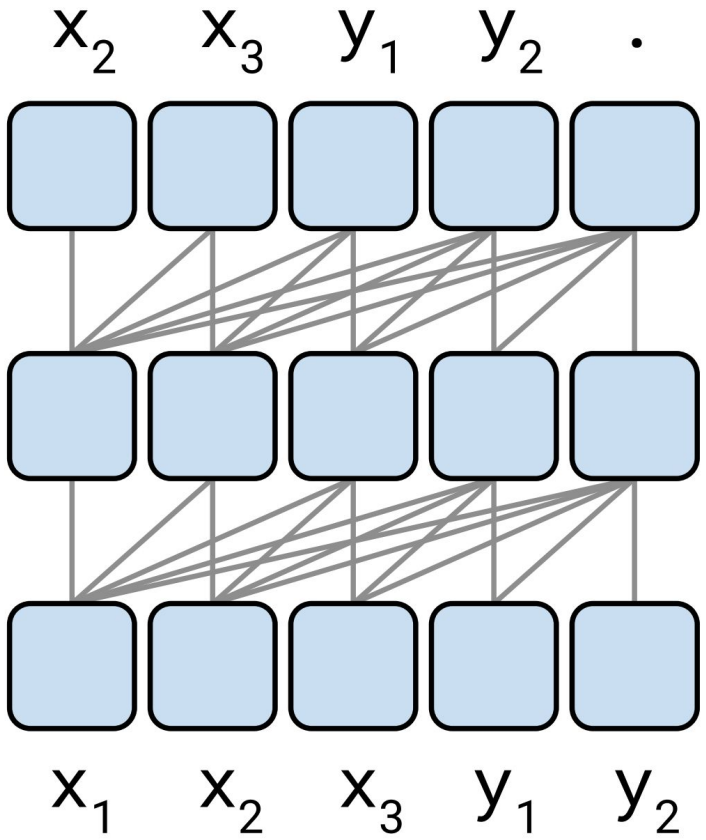


masking out (setting to $-\infty$) all values in the input of the softmax which correspond to illegal connections

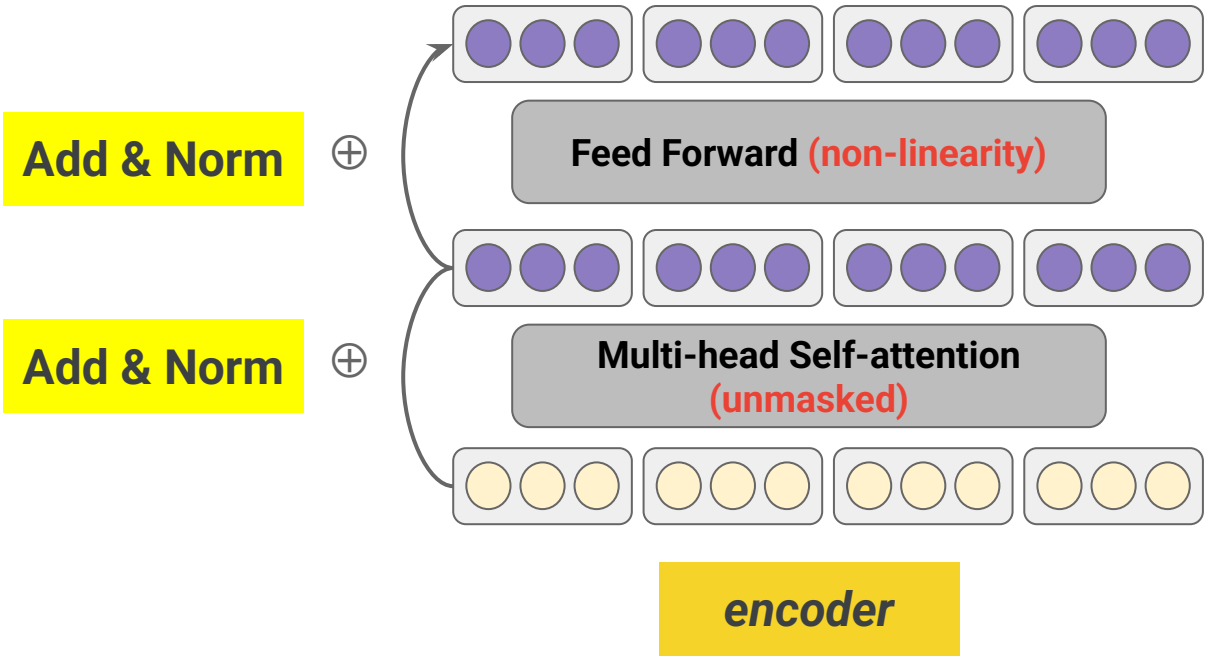
Decoder (cont'd)



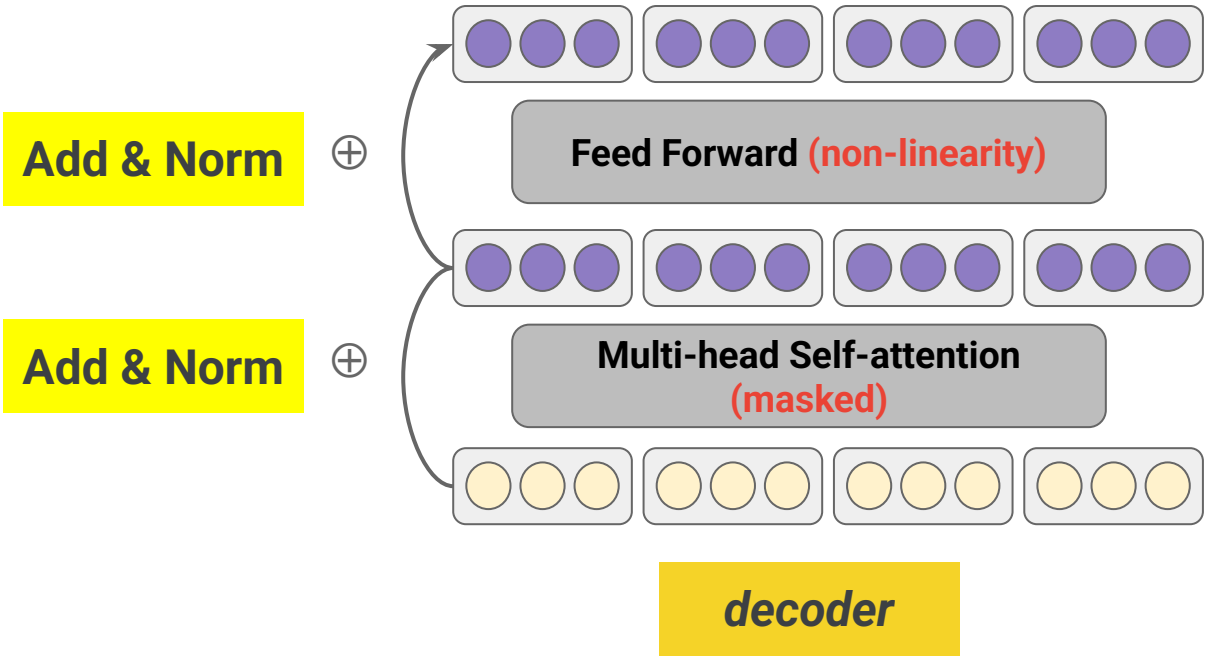
Decoder (cont'd)



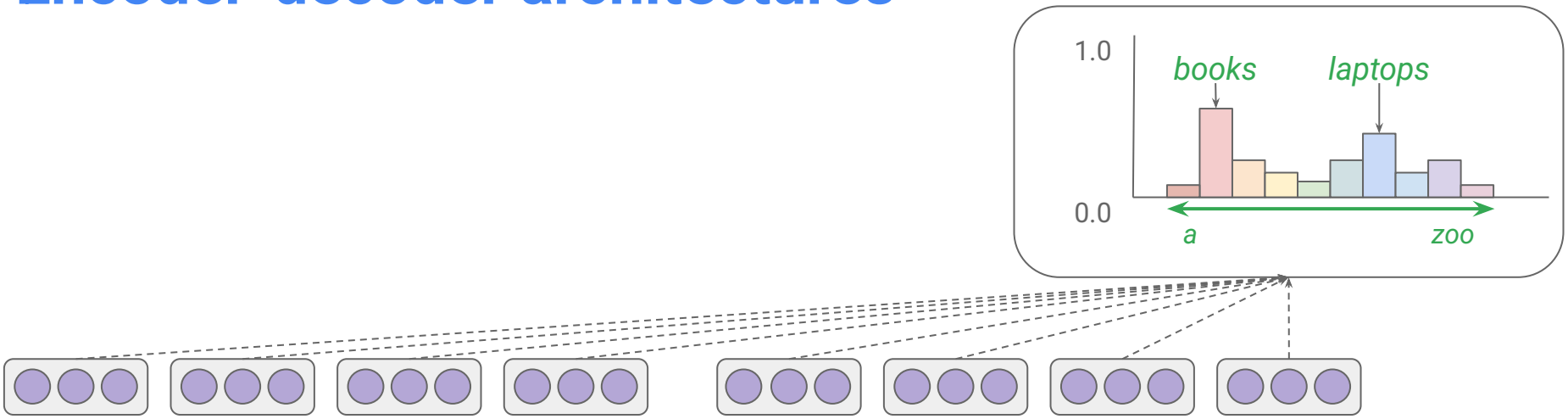
Encoder (one layer)



Decoder (one layer)



Encoder-decoder architectures

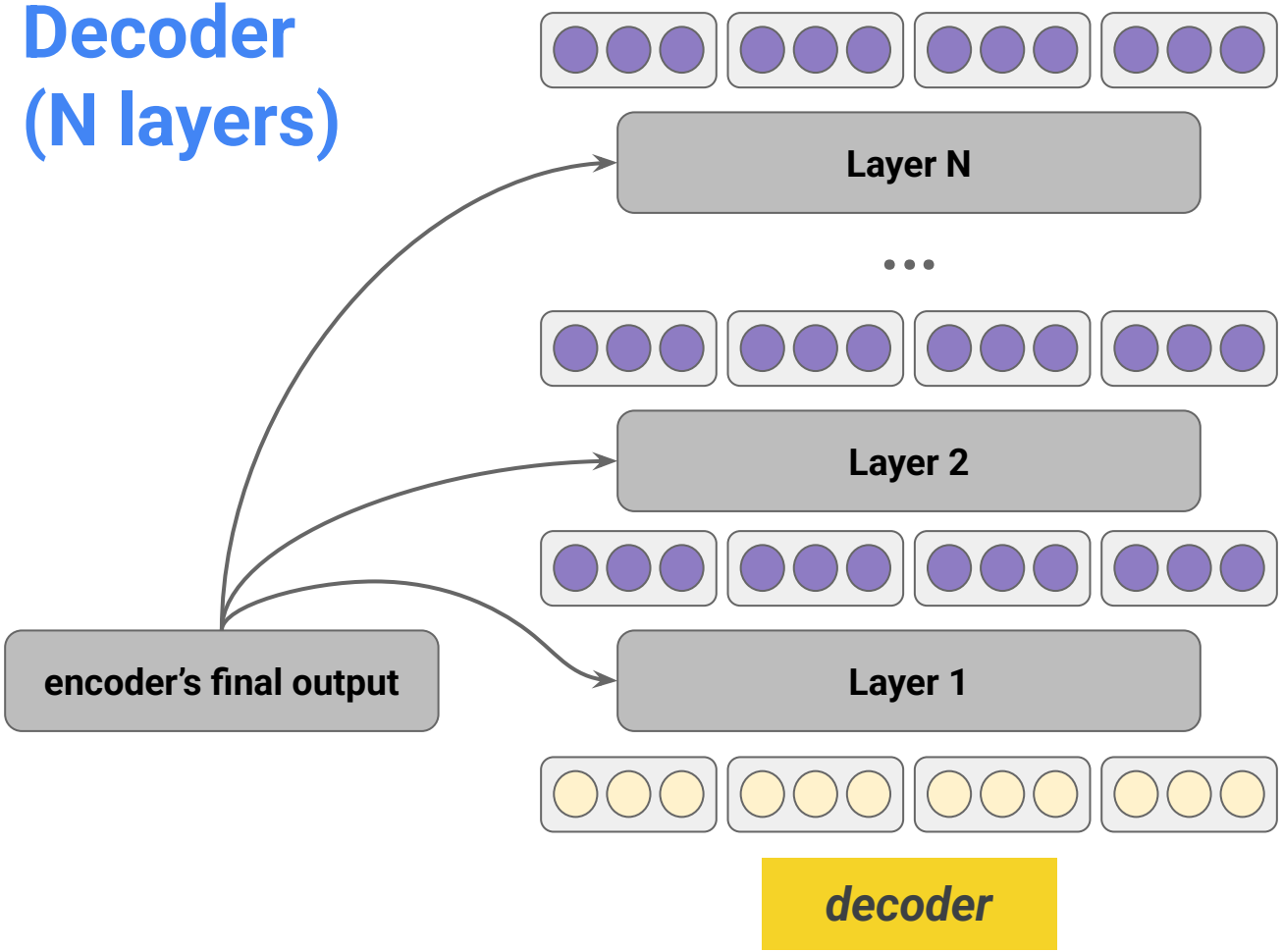


les étudiants ont ouvert leurs livres

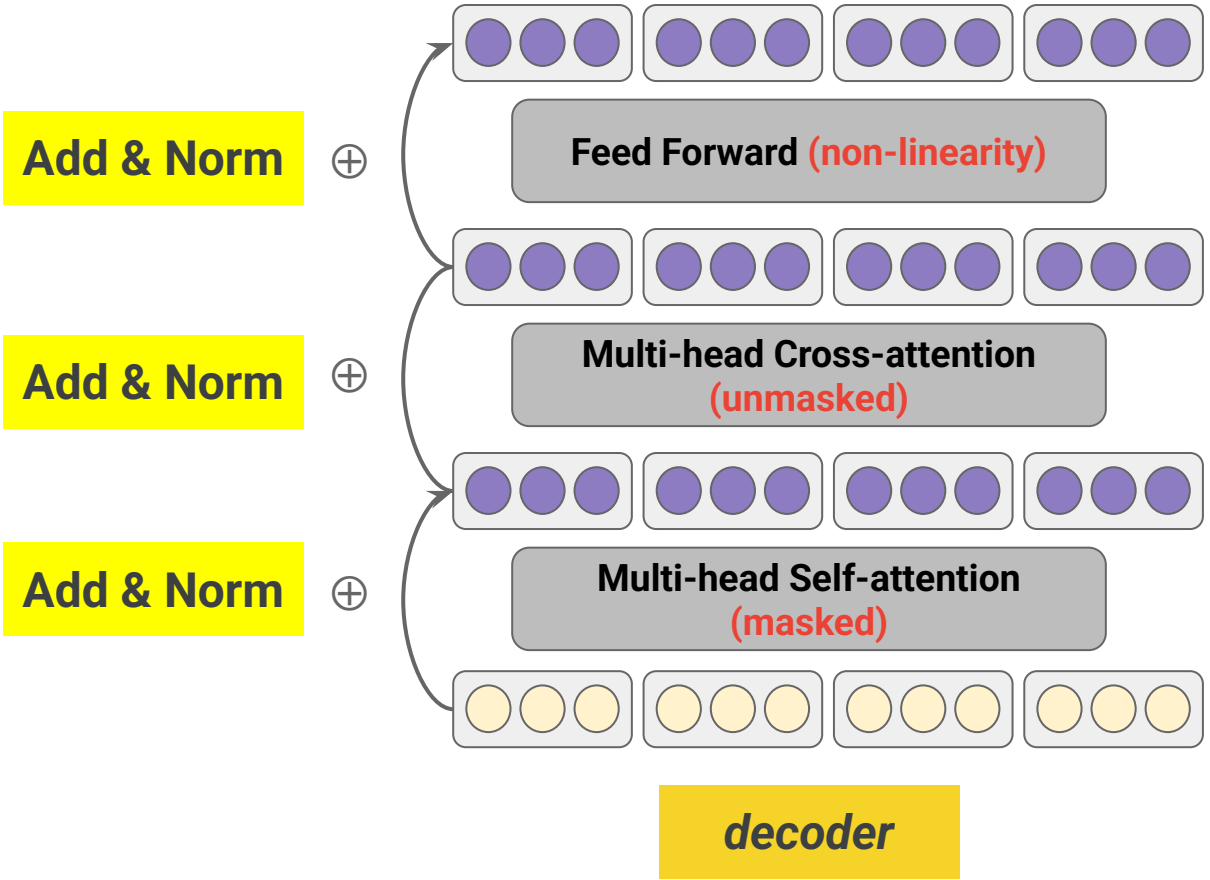
the *students* *opened* *their*

encoder ***decoder***

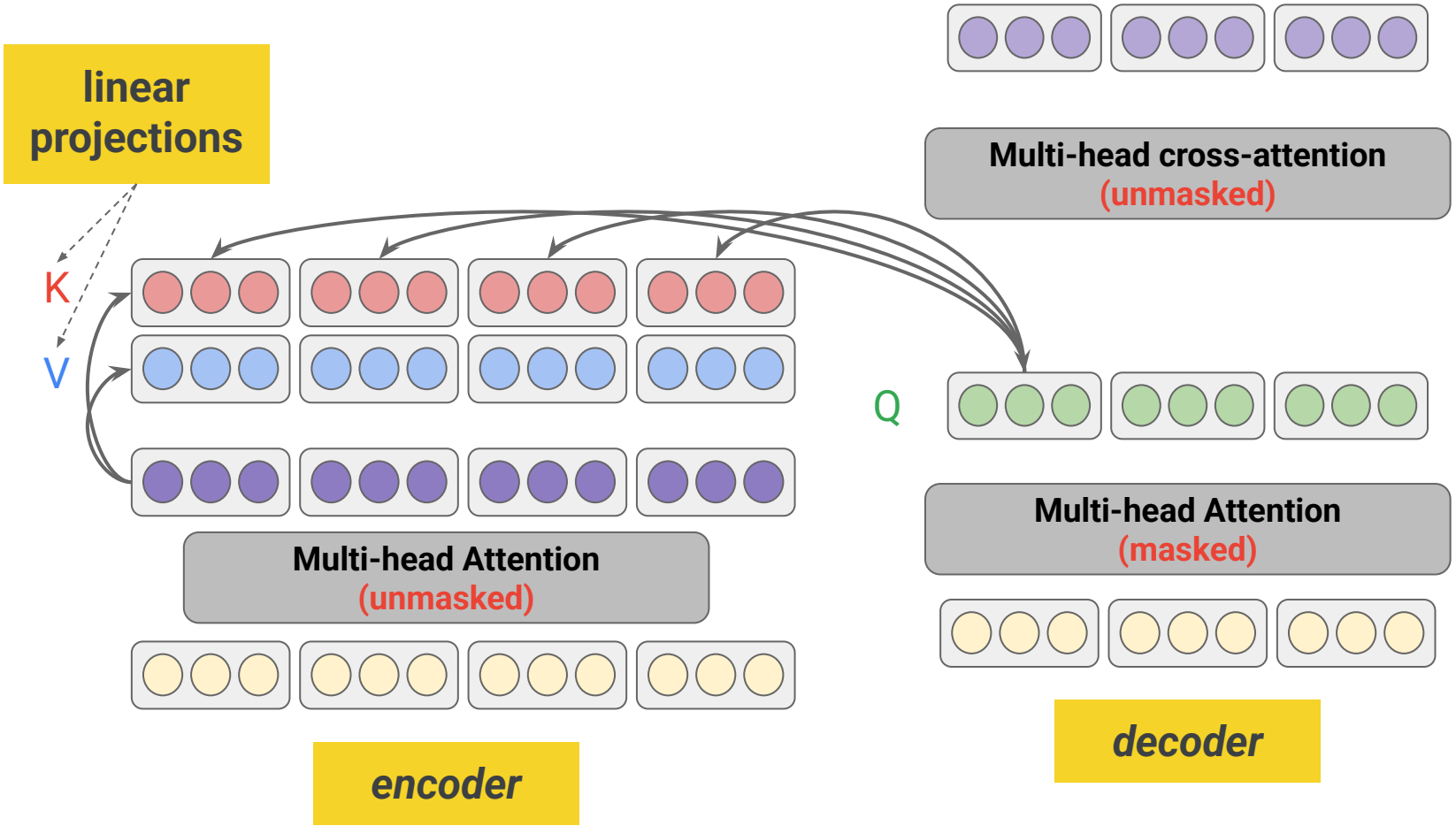
Decoder (N layers)



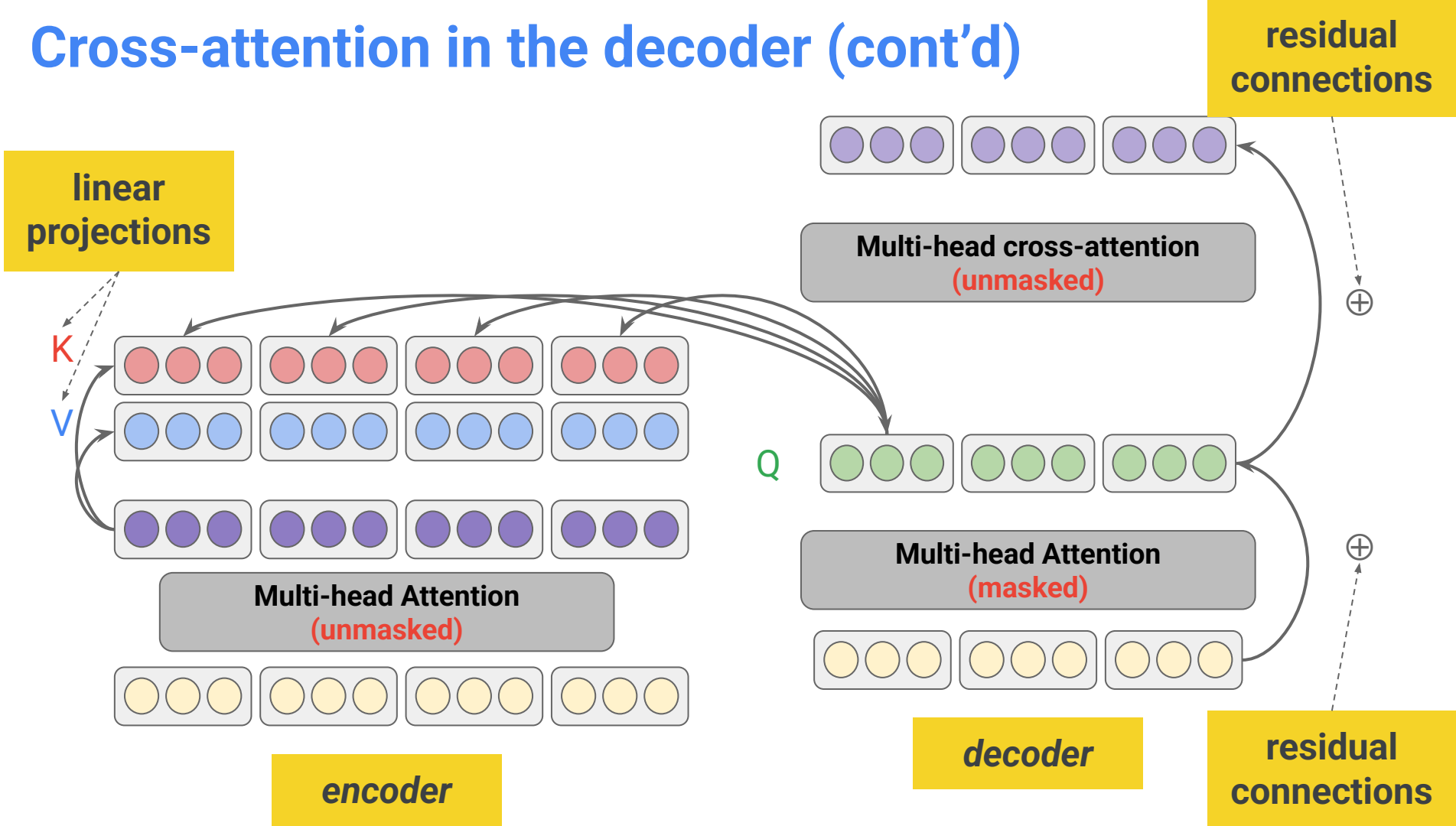
Decoder (one layer)

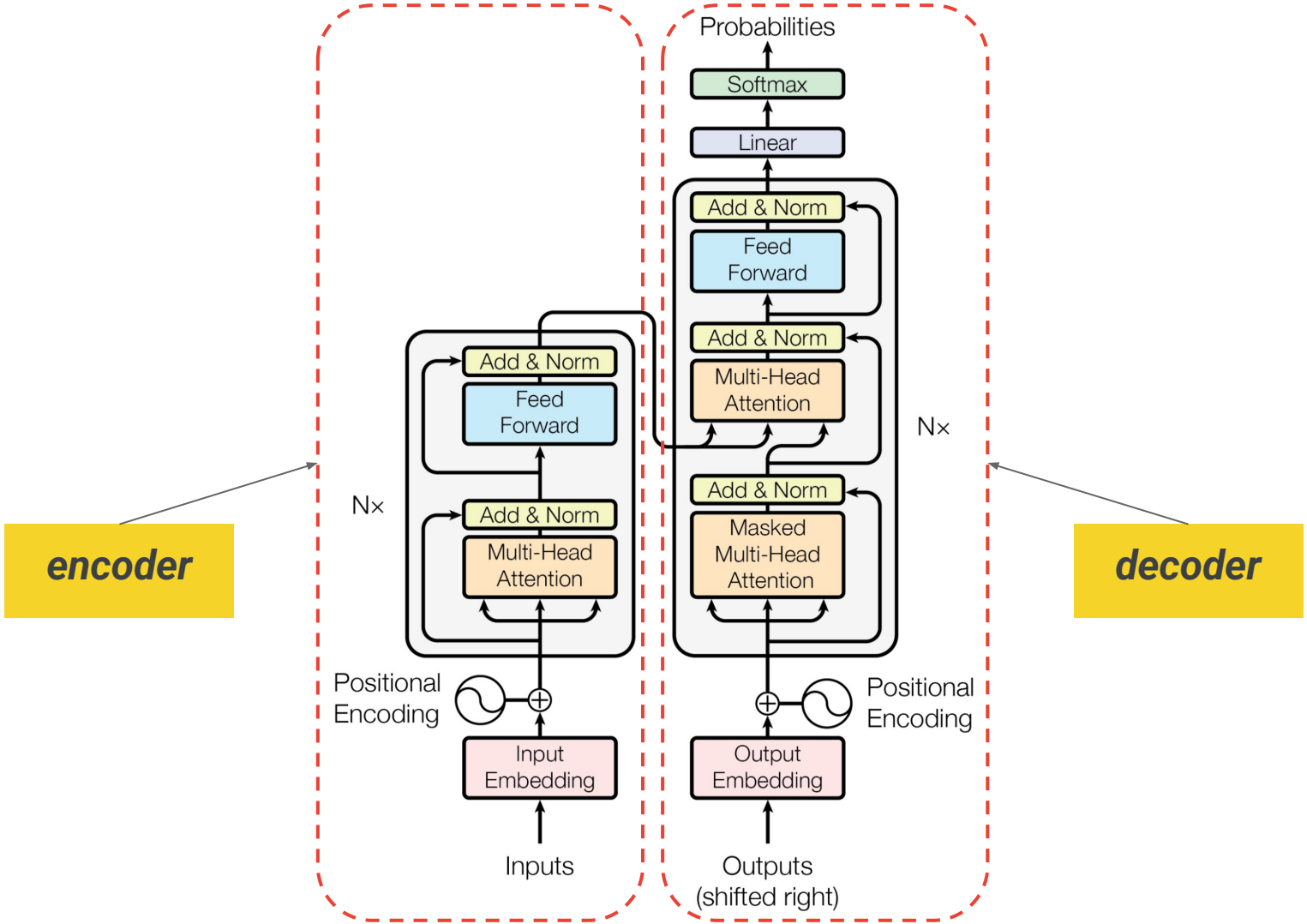


Cross-attention in the decoder



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!