CS 333: Natural Language Processing

Fall 2023

Prof. Carolyn Anderson

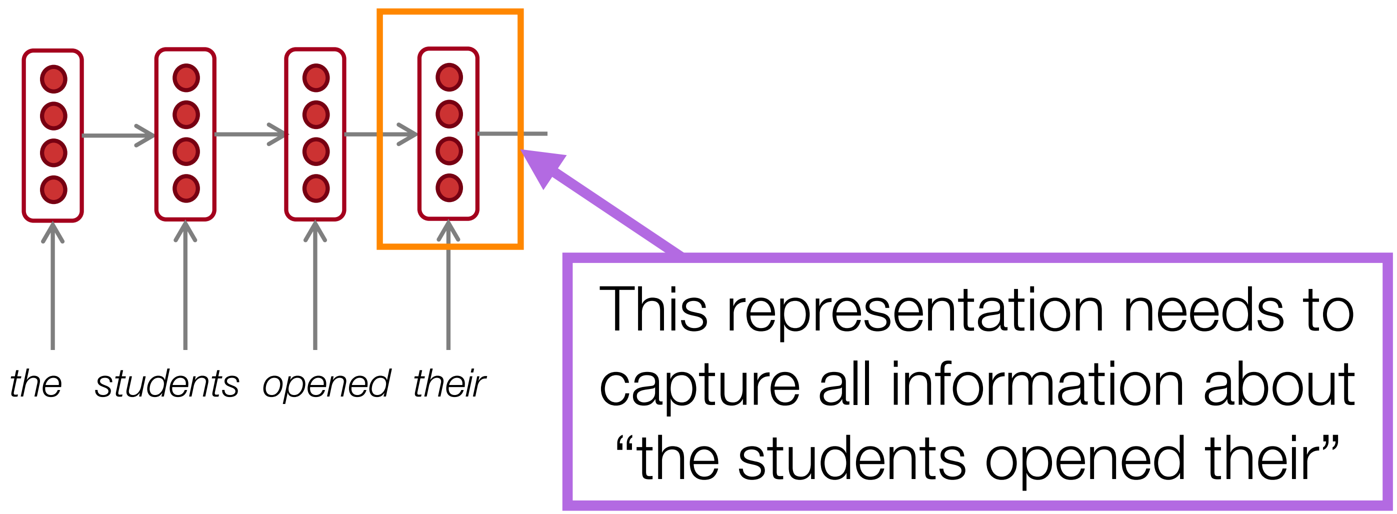
Wellesley College

Recap

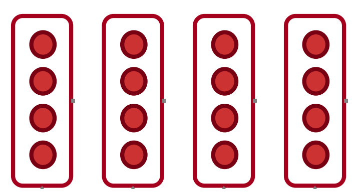
"you can't cram the meaning of a whole %&@#&ing sentence into a single $*(&@ing vector!"

— Ray Mooney (NLP professor at UT Austin)

idea: what if we use multiple vectors?



Instead of this, let's try:

the students opened their =  **(all 4 hidden states!)**

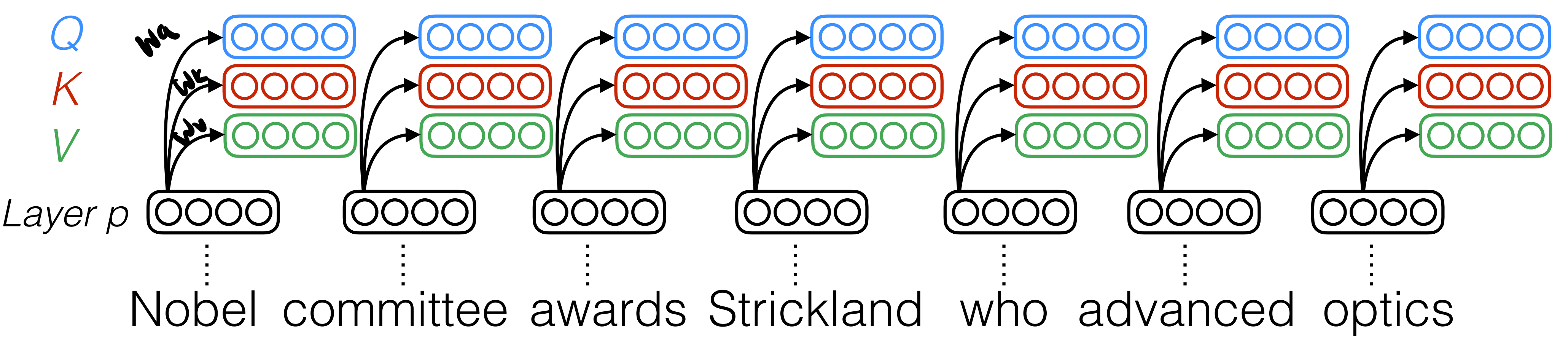
The solution: **attention**

- **Attention mechanisms** (Bahdanau et al., 2015) allow language models to focus on a particular part of the observed context at each time step
  - Originally developed for machine translation, and intuitively similar to *word alignments* between different languages

Self-Attention

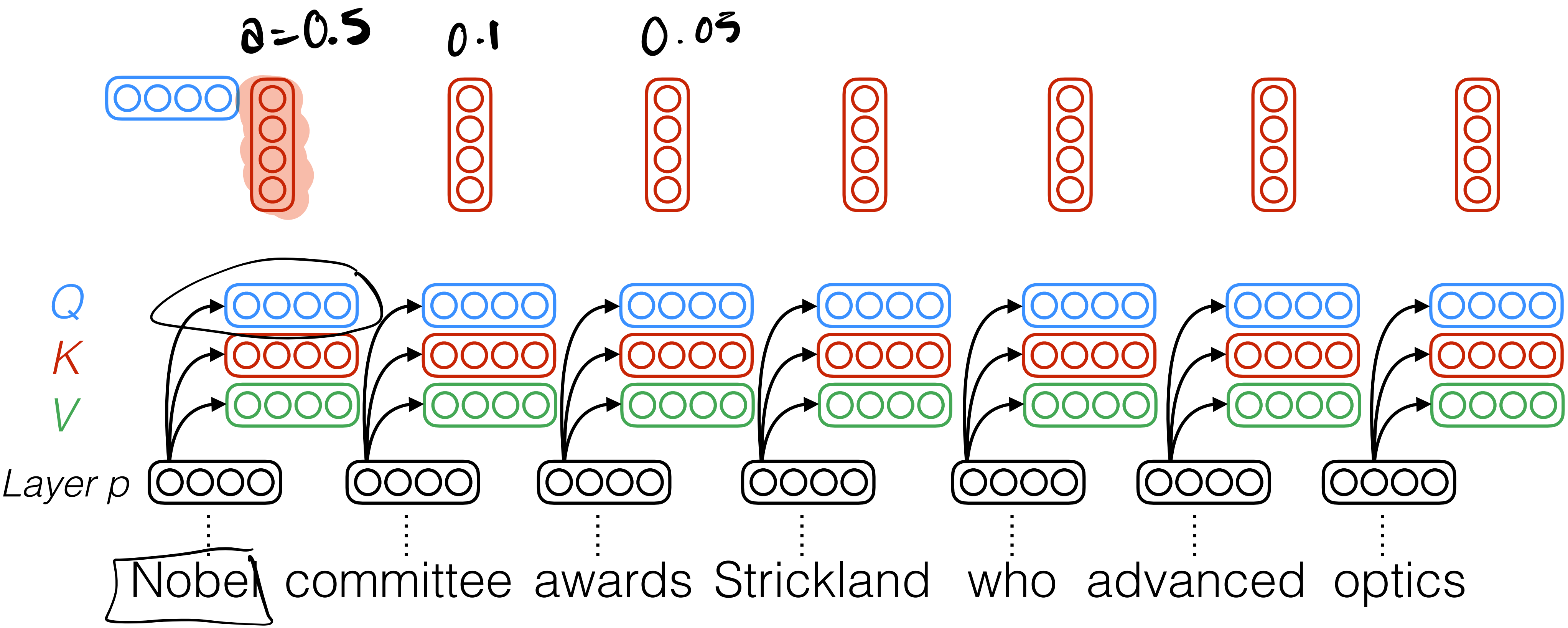
[Vaswani et al. 2017]

Self-attention



[Vaswani et al. 2017]

Self-attention



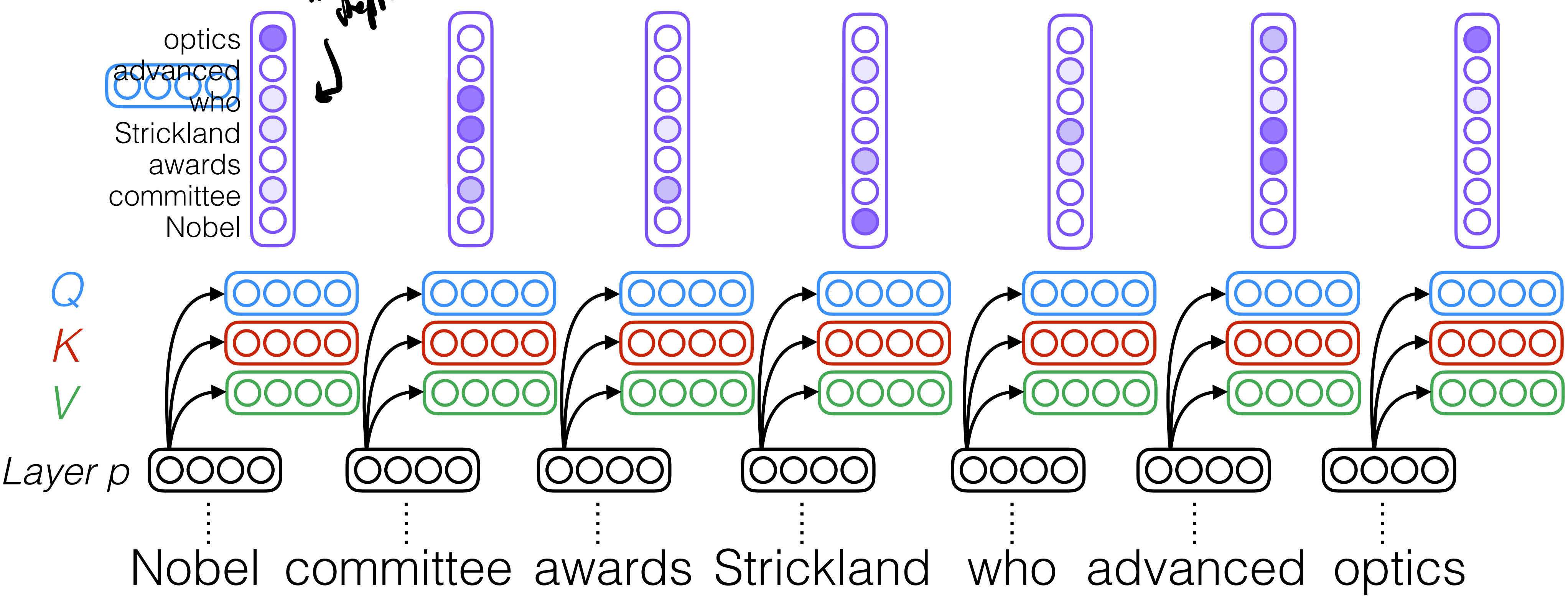
Slides by Emma Strubell!

[Vaswani et al. 2017]

Self-attention

Output of our weighted average over Value vectors

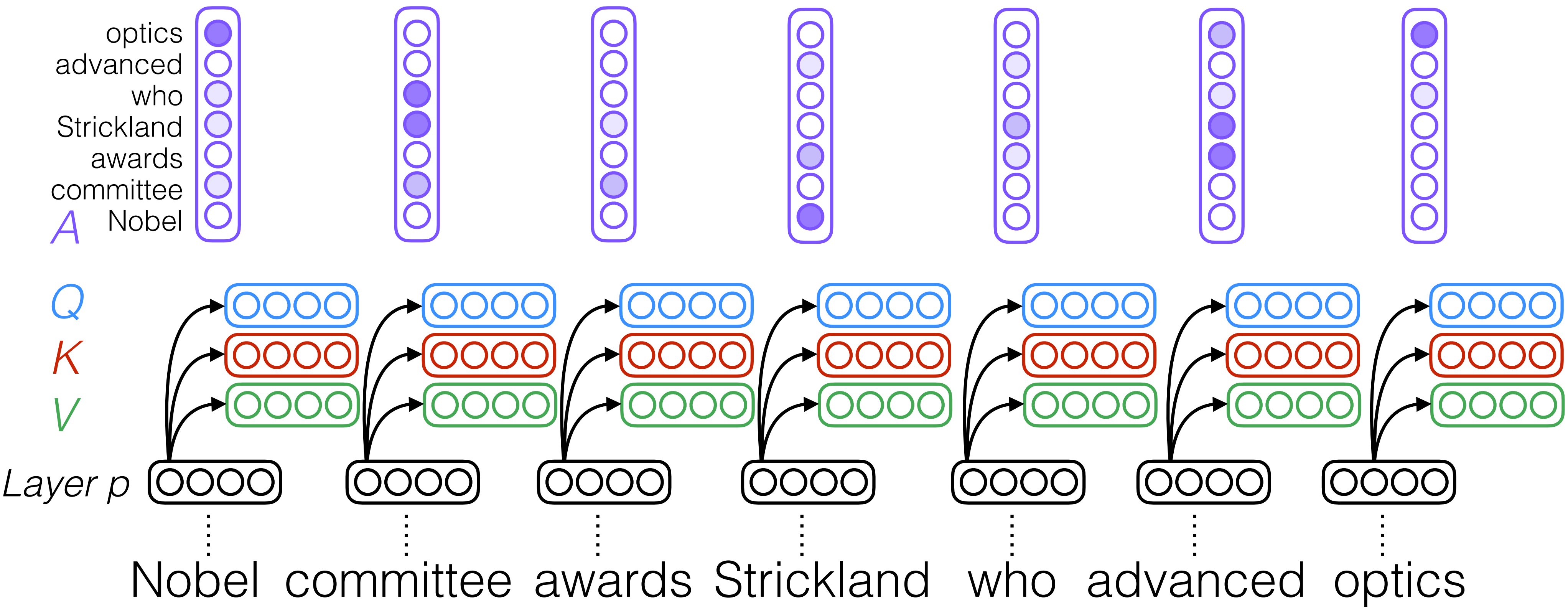
Attention representation



Slides by Emma Strubell!

[Vaswani et al. 2017]

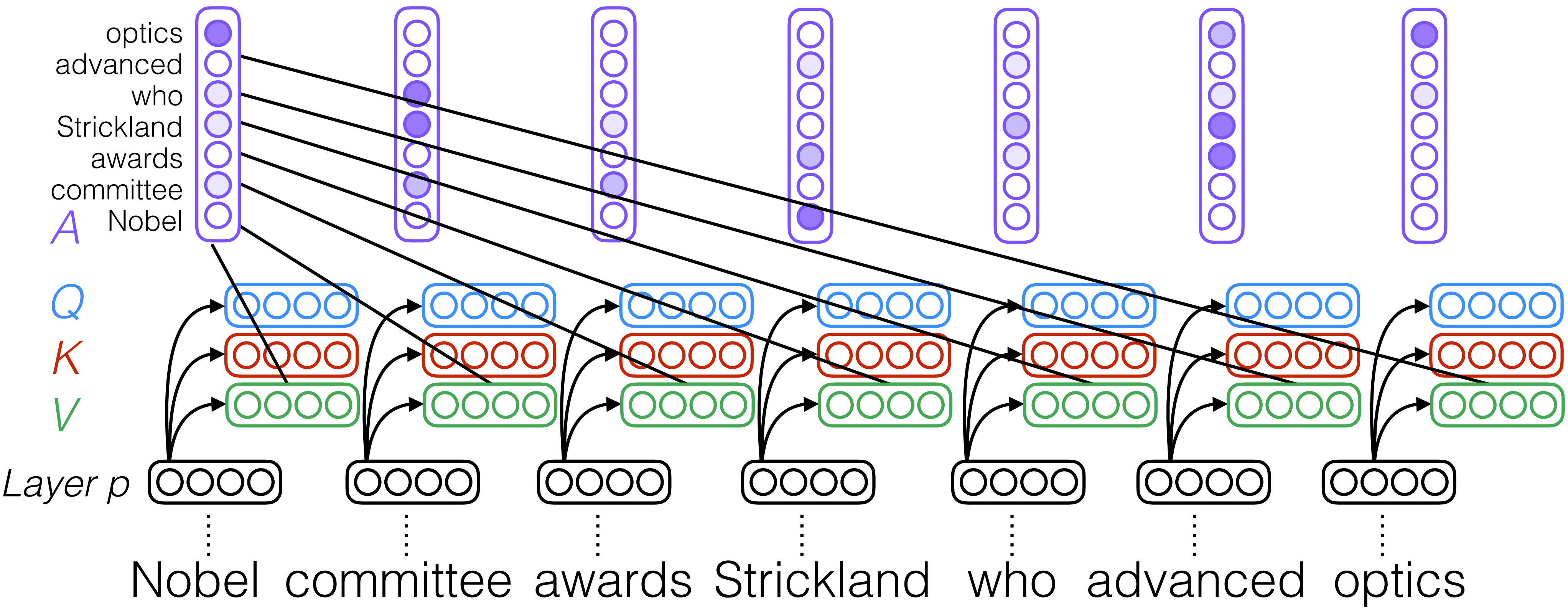
Self-attention



Slides by Emma Strubell!

[Vaswani et al. 2017]

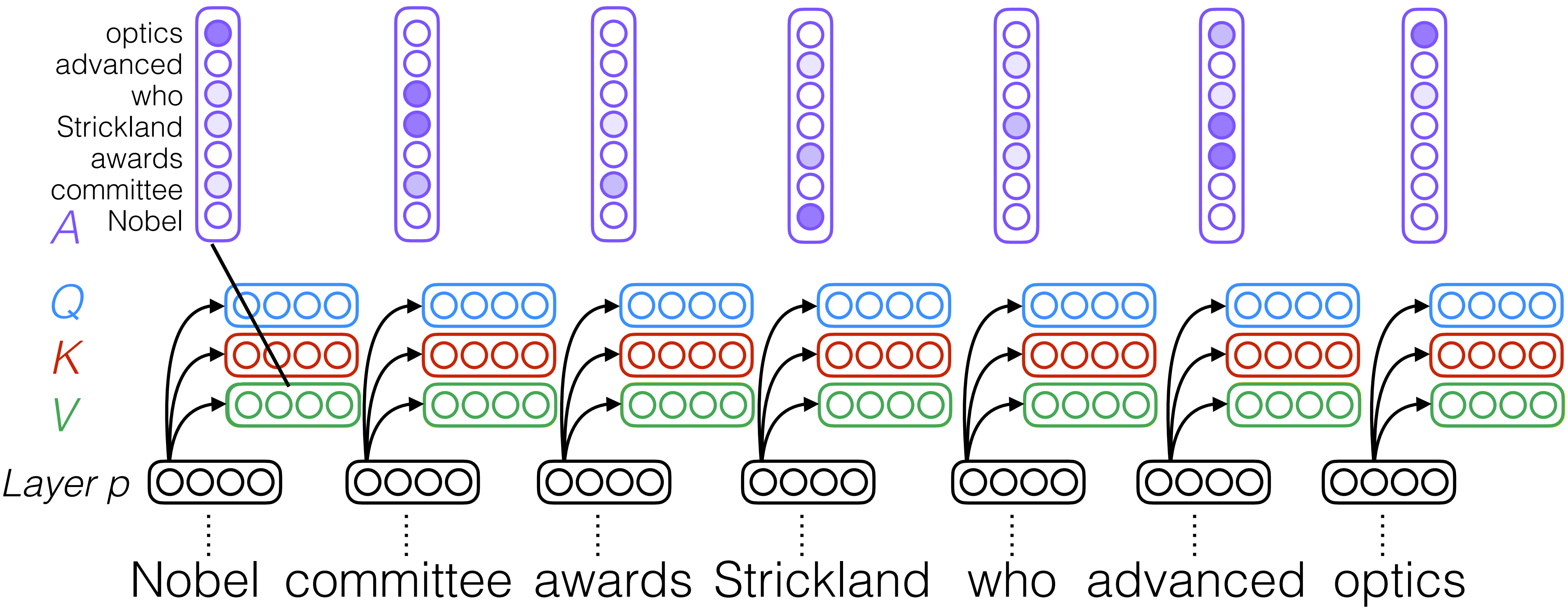
Self-attention



Slides by Emma Strubell!

[Vaswani et al. 2017]

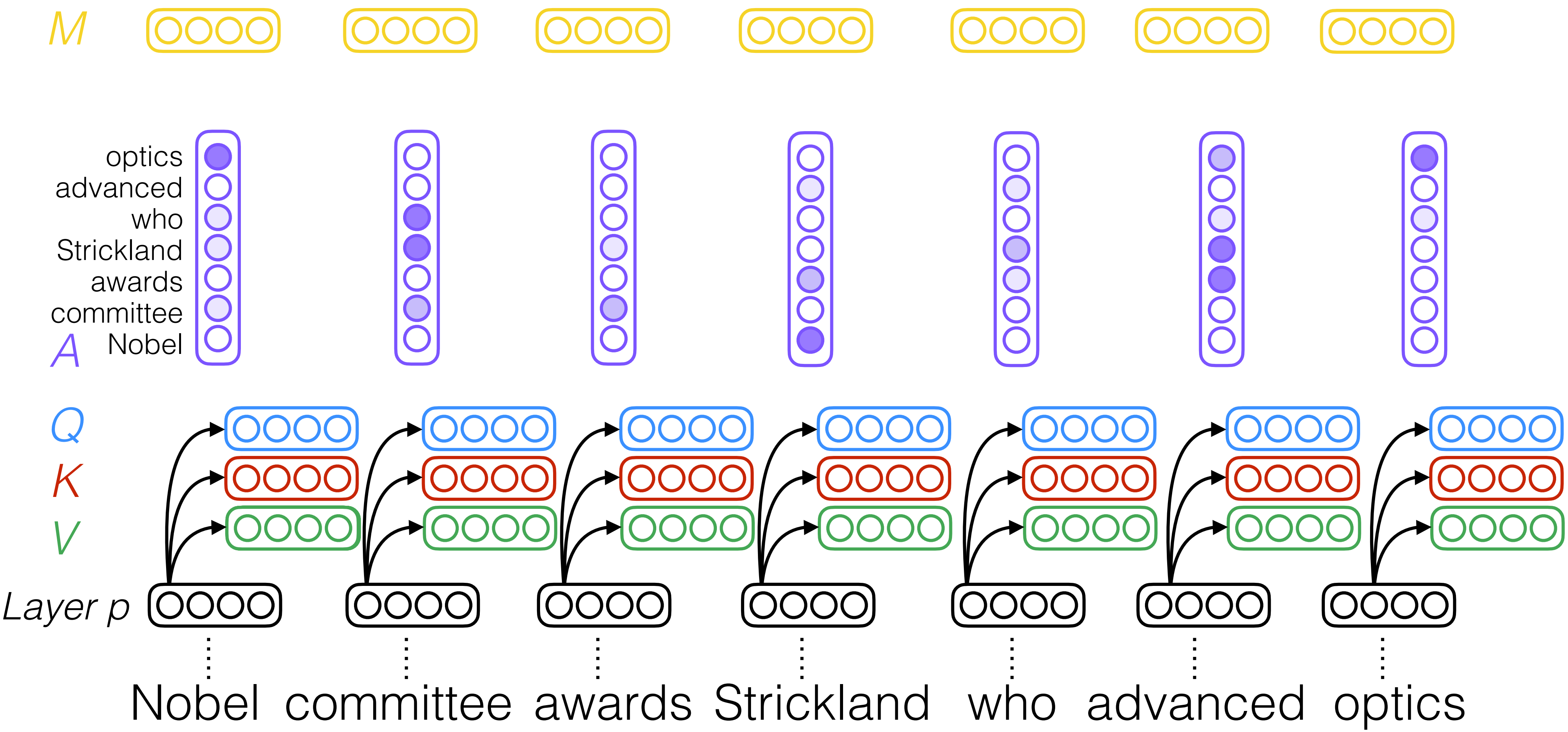
Self-attention



Slides by Emma Strubell!

[Vaswani et al. 2017]

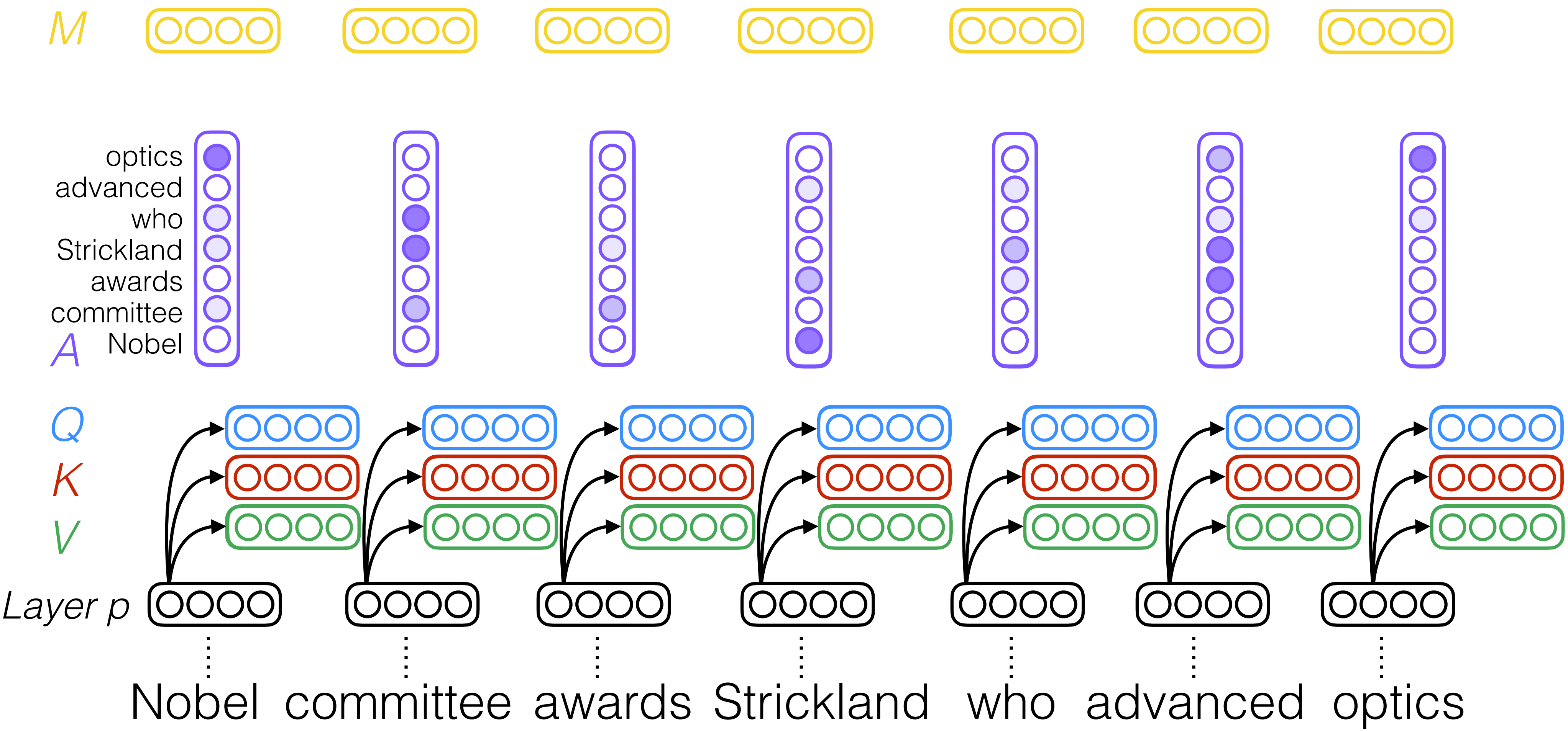
Self-attention



Slides by Emma Strubell!

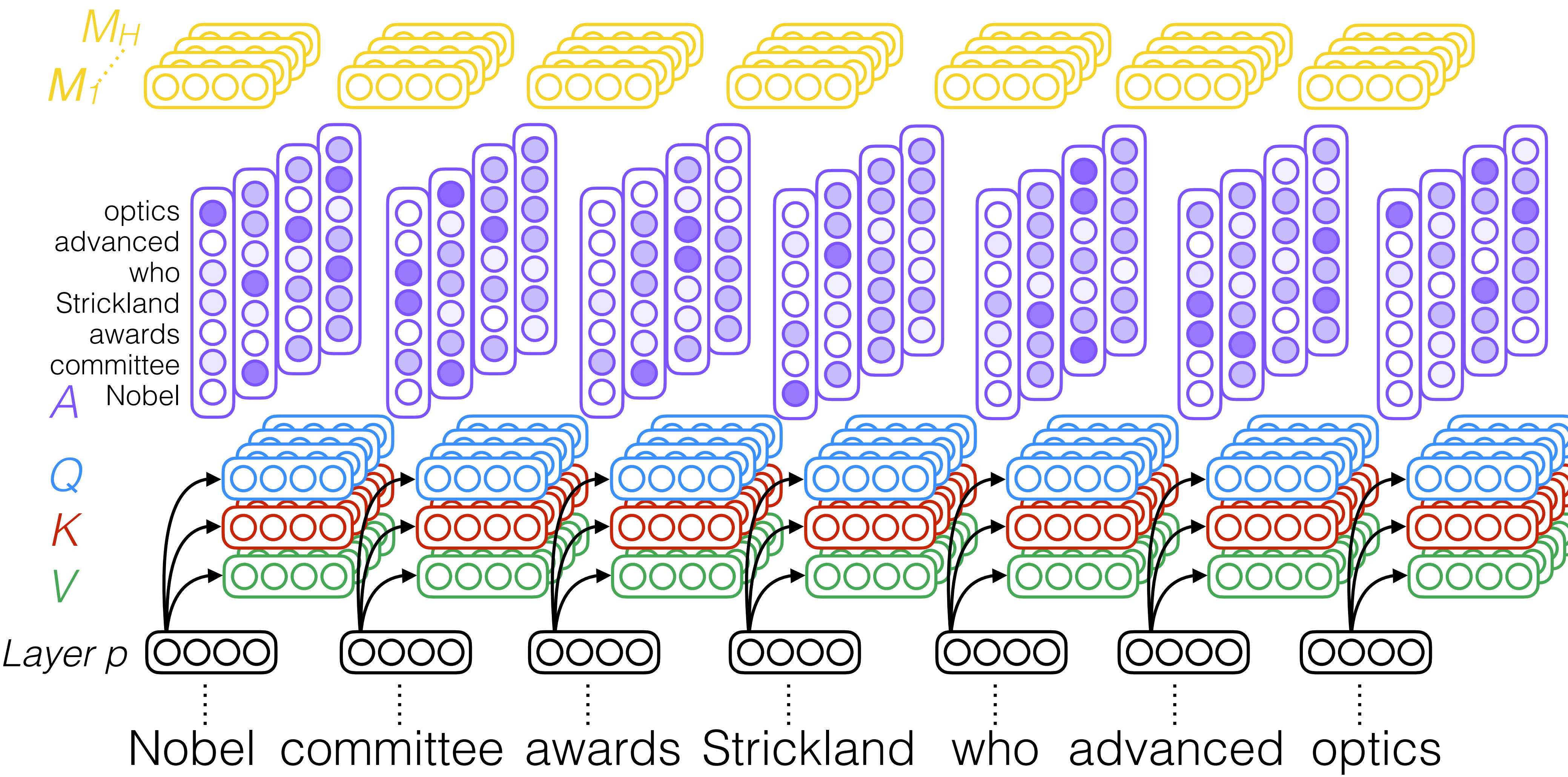
[Vaswani et al. 2017]

Self-attention



[Vaswani et al. 2017]

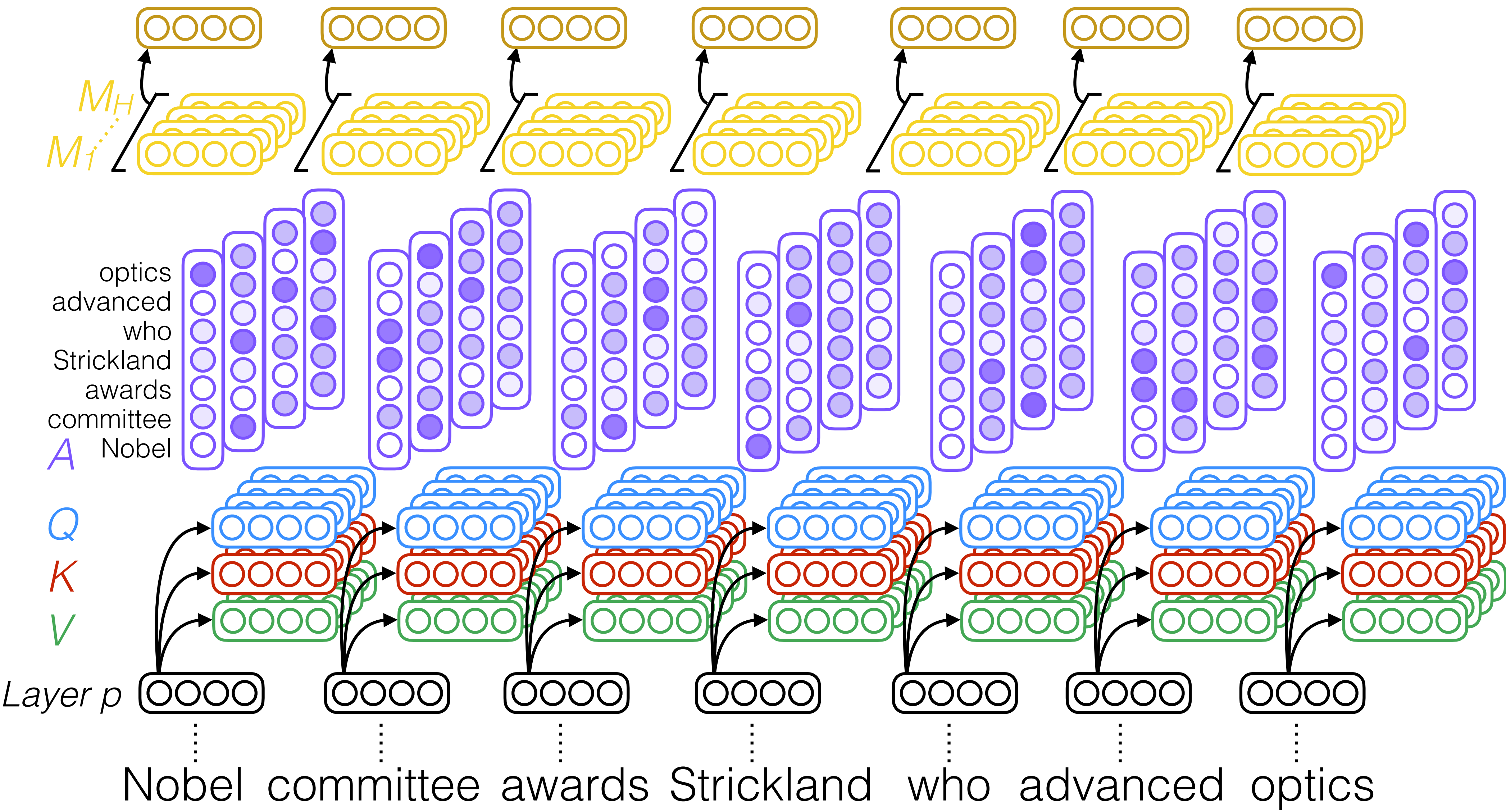
Multi-head self-attention



Slides by Emma Strubell!

[Vaswani et al. 2017]

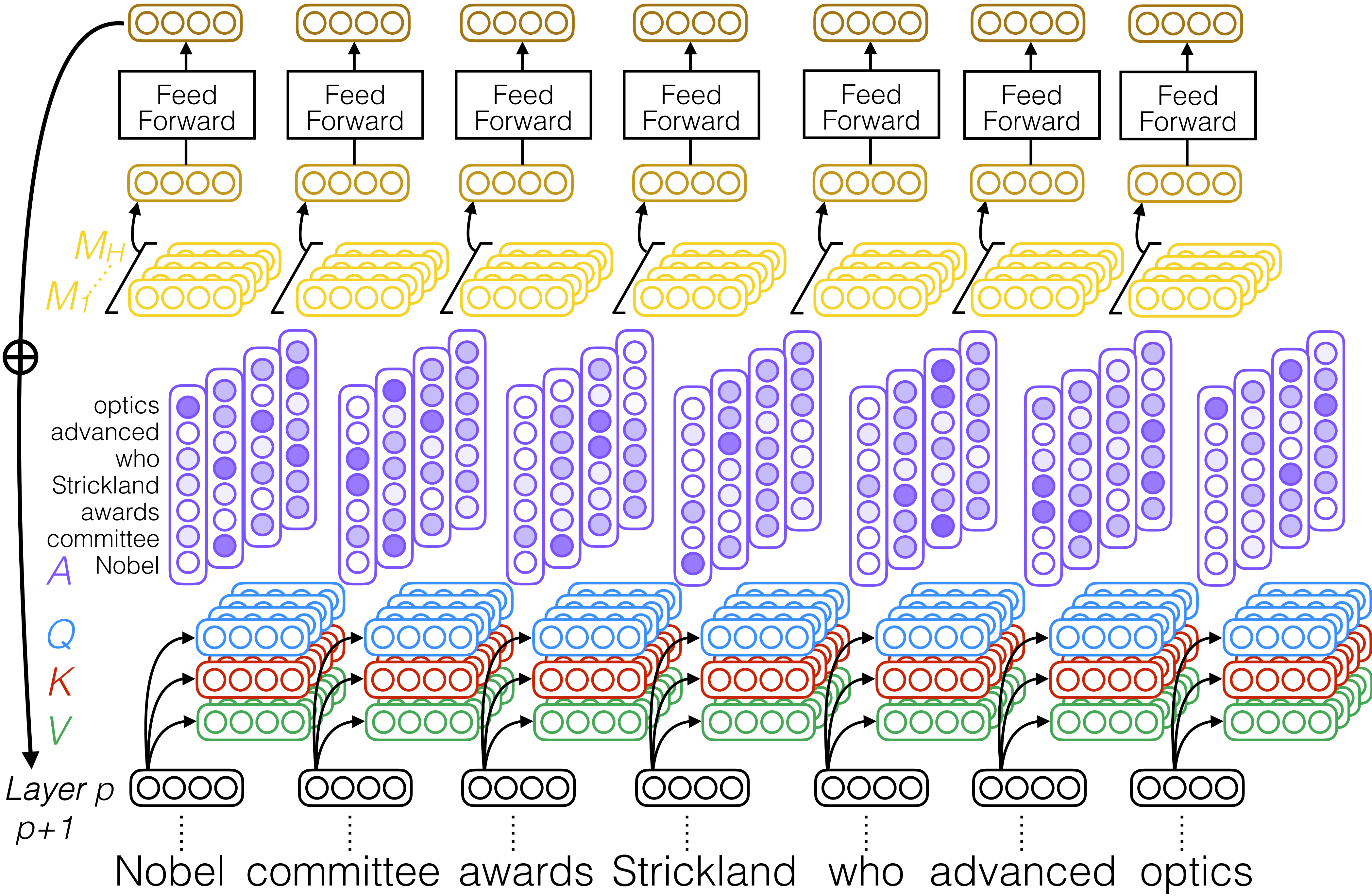
Multi-head self-attention



Slides by Emma Strubell!

[Vaswani et al. 2017]

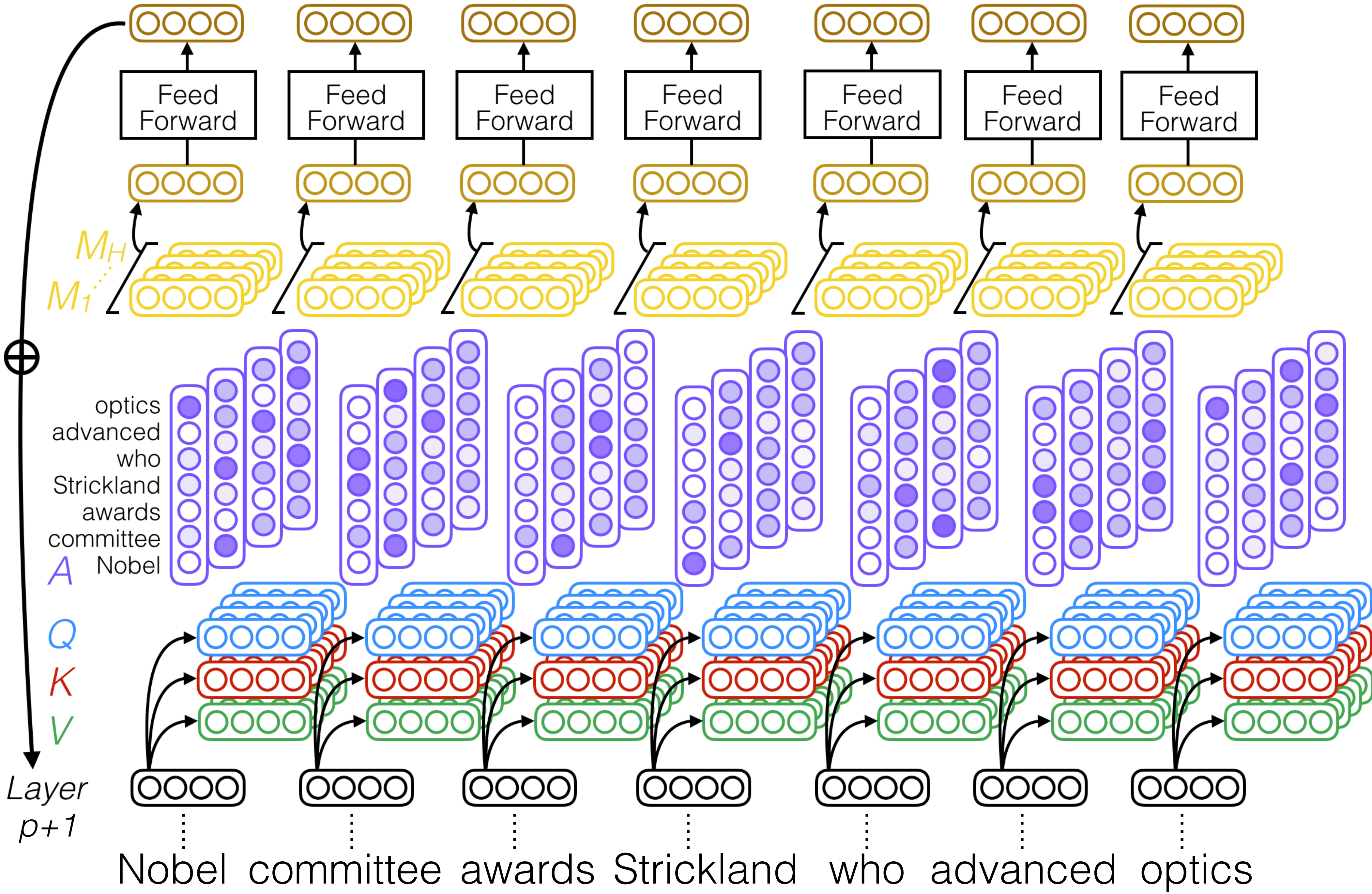
Multi-head self-attention



Slides by Emma Strubell!

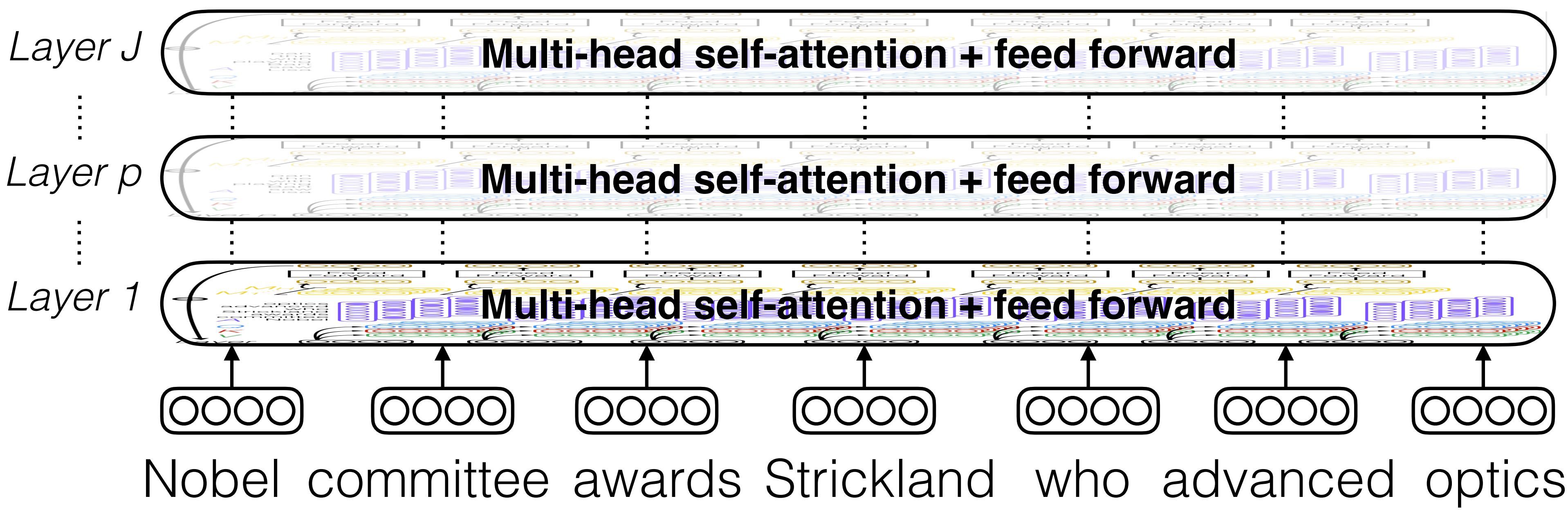
[Vaswani et al. 2017]

Multi-head self-attention



[Vaswani et al. 2017]

Multi-head self-attention



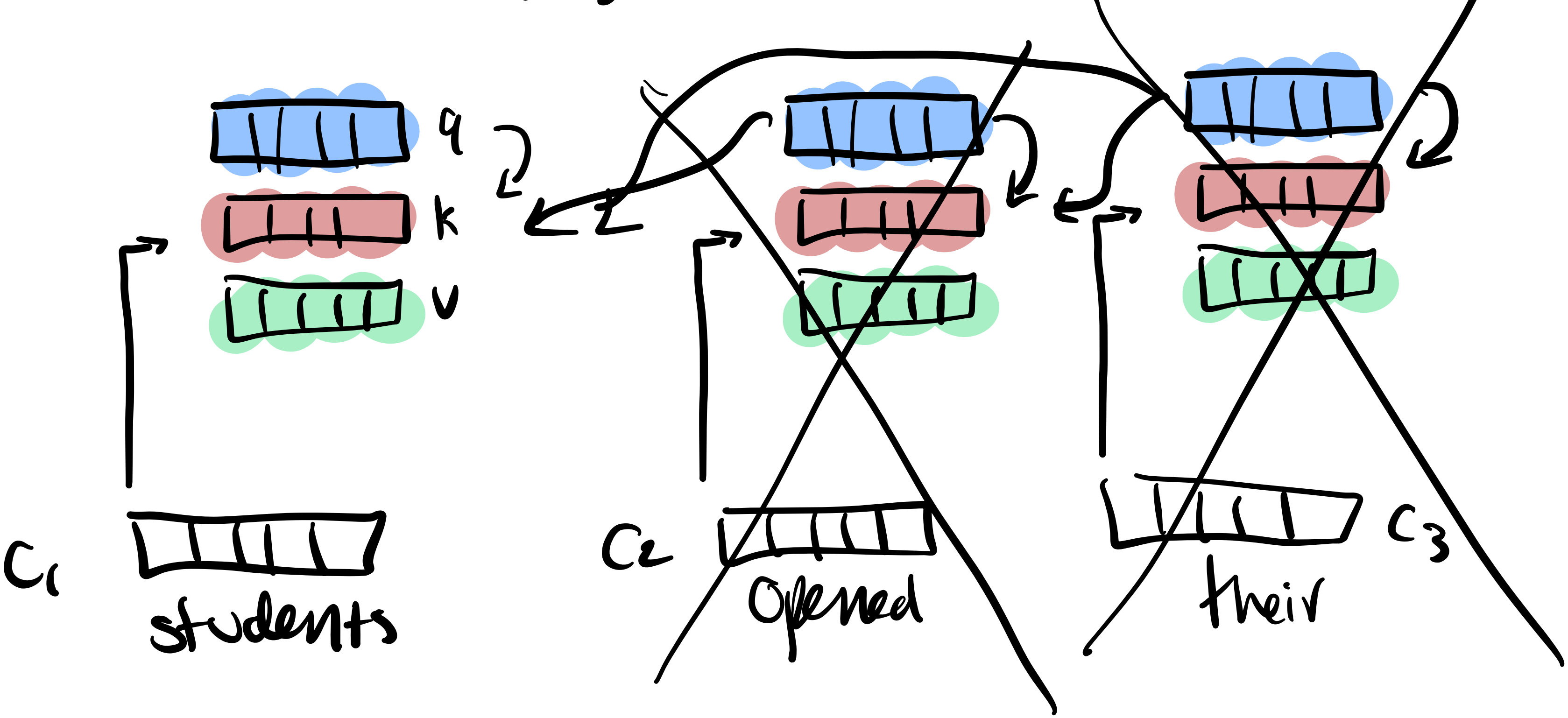
Self-Attention

1. Compute dot products

$0.25 v_1 + 0.5 v_2 + 0.25 v_3 = a_1$

$\langle 0.25 \quad 0.5 \quad 0.25 \rangle$

$\langle q_1 k_1 \quad q_1 k_2 \quad q_1 k_3 \rangle$



Self-Attention

Challenge: how to parallelize w/out cheating!

h_1: v_1

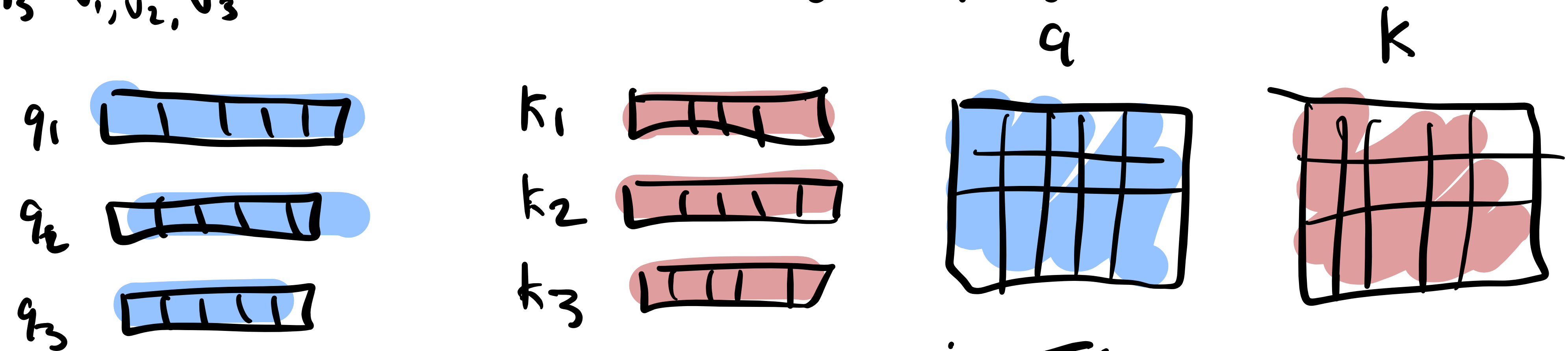
h_2: v_1, v_2

h_3: v_1, v_2, v_3

$a_1 = \langle q_1 k_1 \rangle$

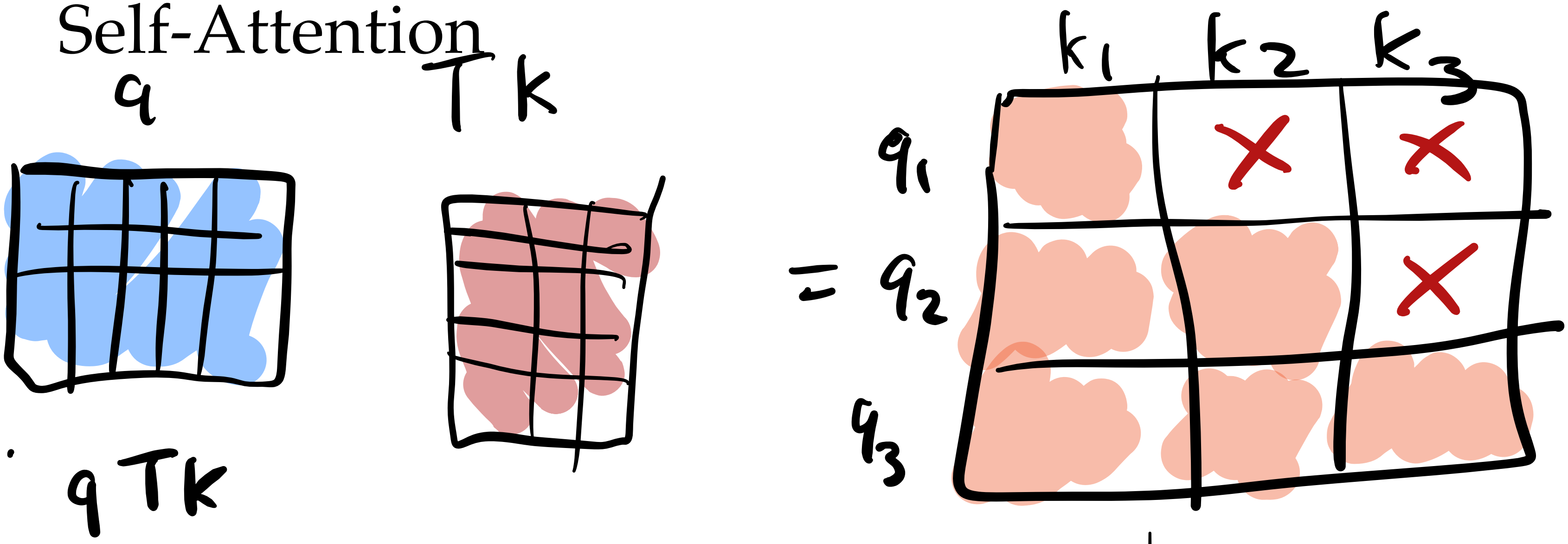
$a_2 = \langle q_2 k_1 \quad q_2 k_2 \rangle$

$a_3 = \langle q_3 k_1 \quad q_3 k_2 \quad q_3 k_3 \rangle$



$q^T K$

Self-Attention



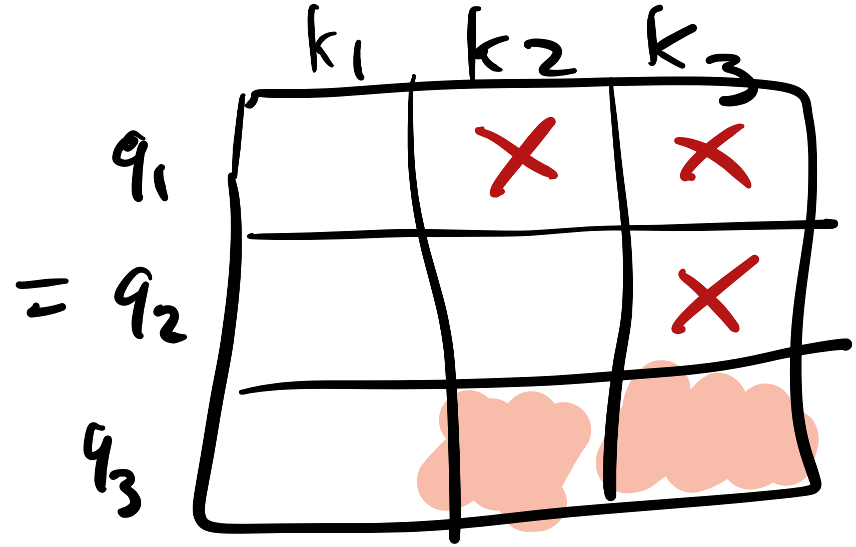
Apply a MASK to hide cheater dot products

$$
\begin{bmatrix} 1 & -\infty & -\infty \\ 1 & 1 & -\infty \\ 1 & 1 & 1 \end{bmatrix}
=
\begin{array}{c}
\begin{bmatrix} \cdot & 0 & 0 \\ \cdot & \cdot & 0 \\ \cdot & \cdot & \cdot \end{bmatrix}
\end{array}
\begin{array}{l} \leftarrow a_1 \\ \leftarrow a_2 \\ \leftarrow a_3 \end{array}
$$

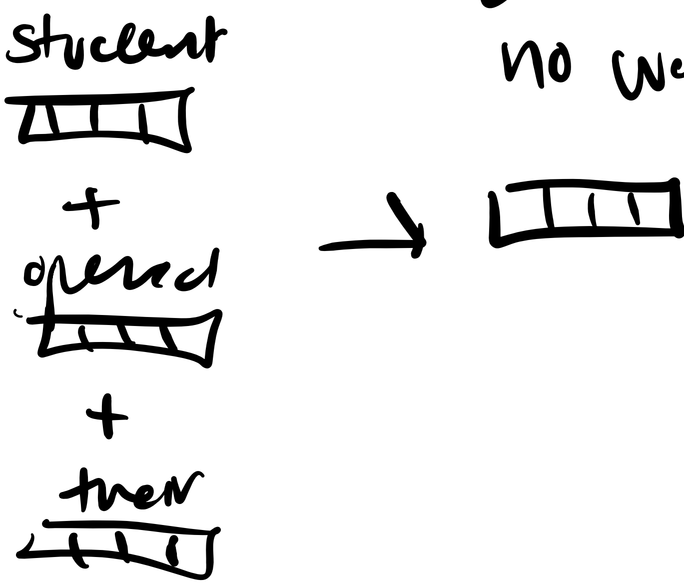
Self-Attention

Concatenation:

- Downside: fixed context window



Averaging:



- Downside: no word order!

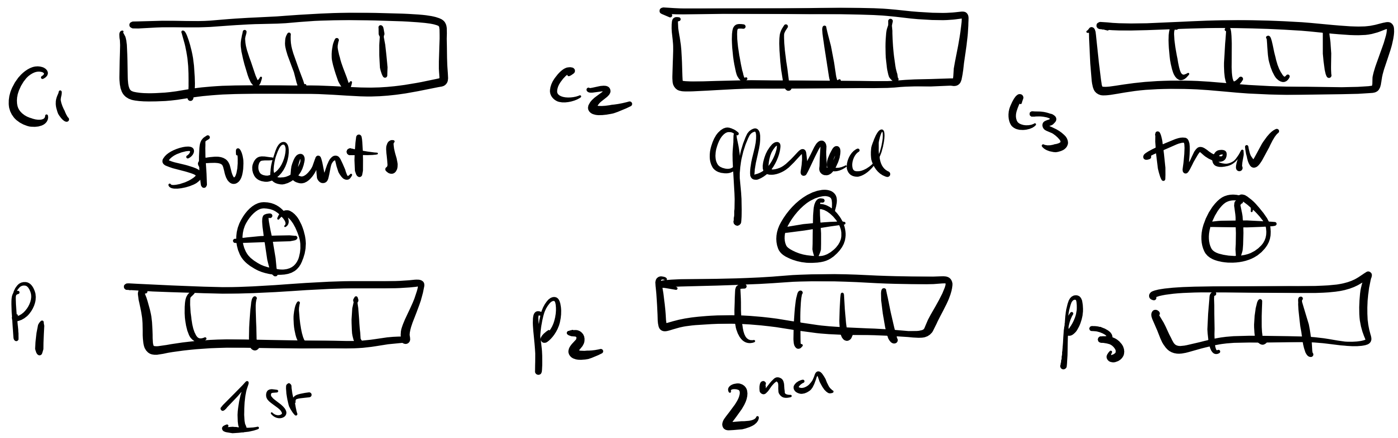
Problems

1. Fixed-context window
2. No word order

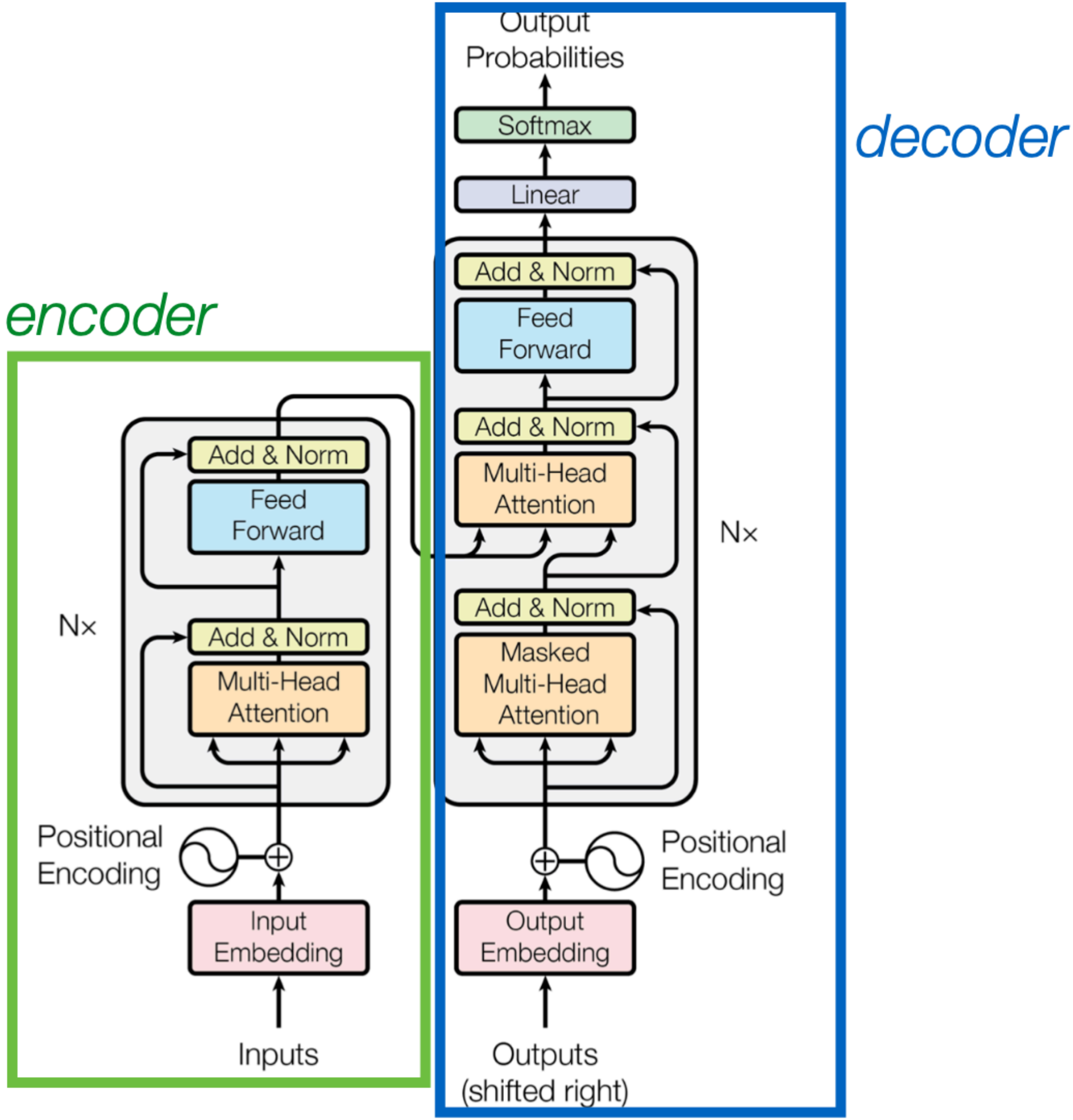
Word Order | Position Embeddings

How to make?

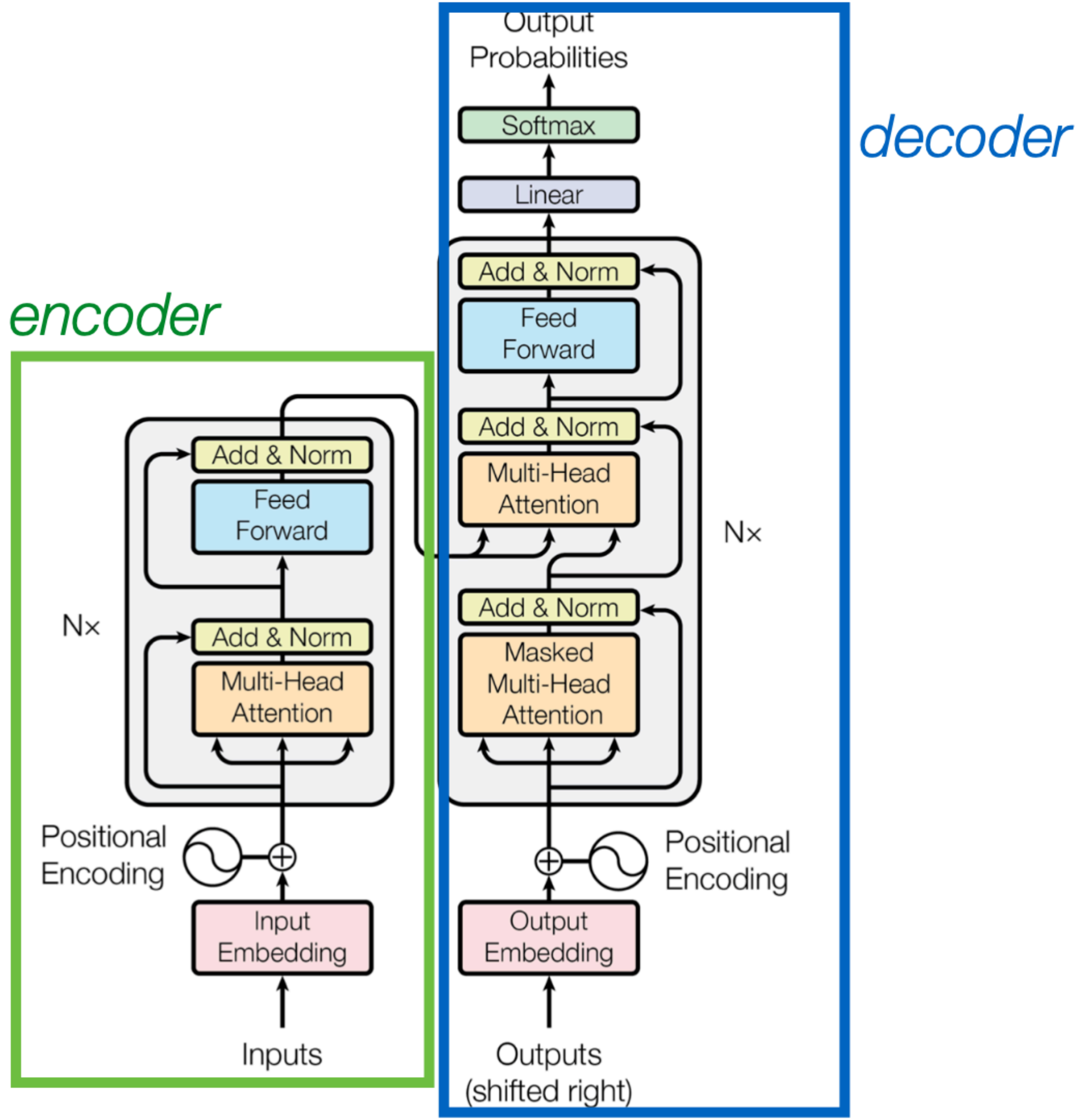
- set max length & learn an embedding for each position.



Transformers

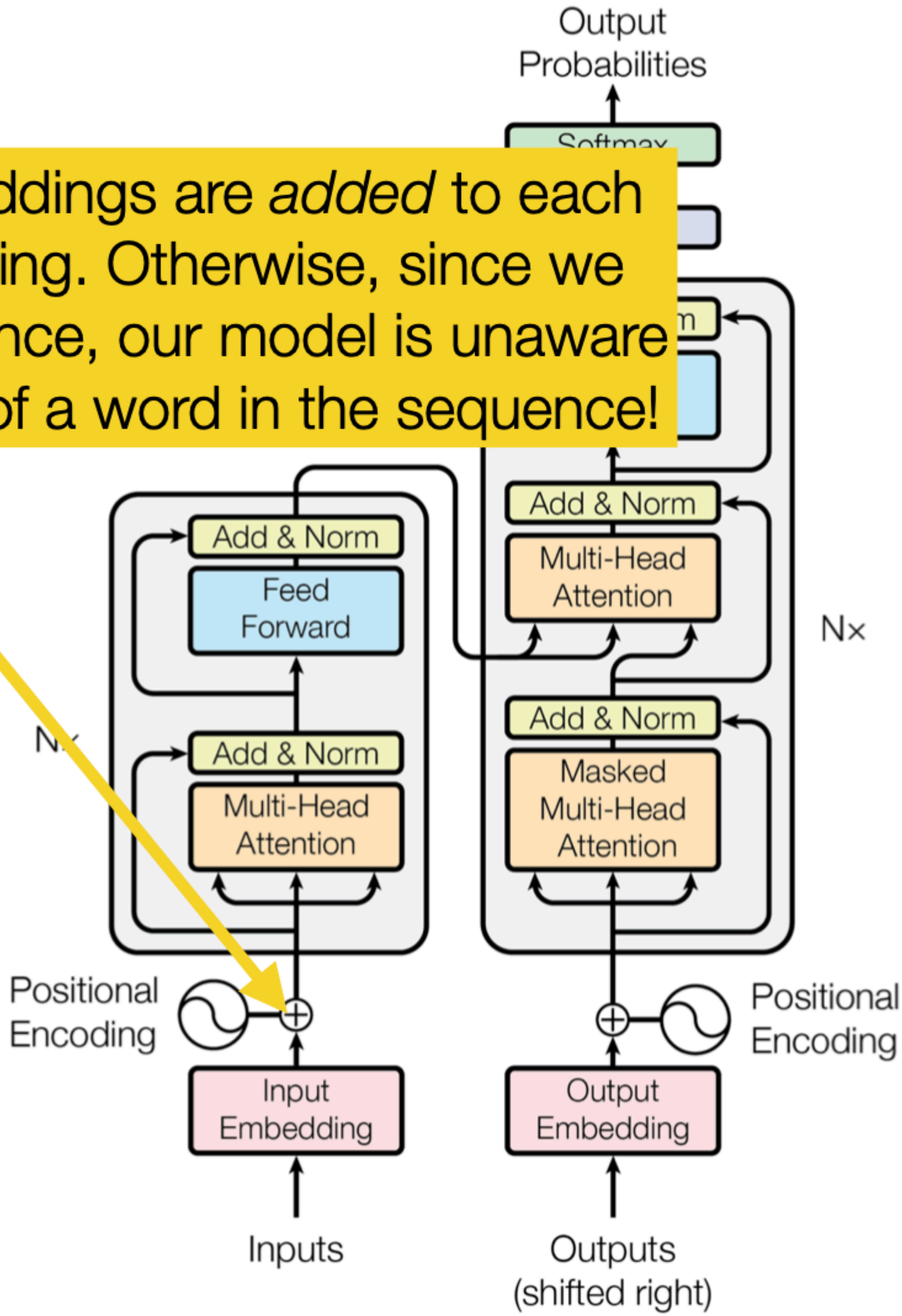


decoder
Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Nx
Add & Norm
Masked Multi-Head Attention
Positional Encoding
Output Embedding
Outputs (shifted right)
encoder
Add & Norm
Feed Forward
Nx
Add & Norm
Multi-Head Attention
Positional Encoding
Input Embedding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

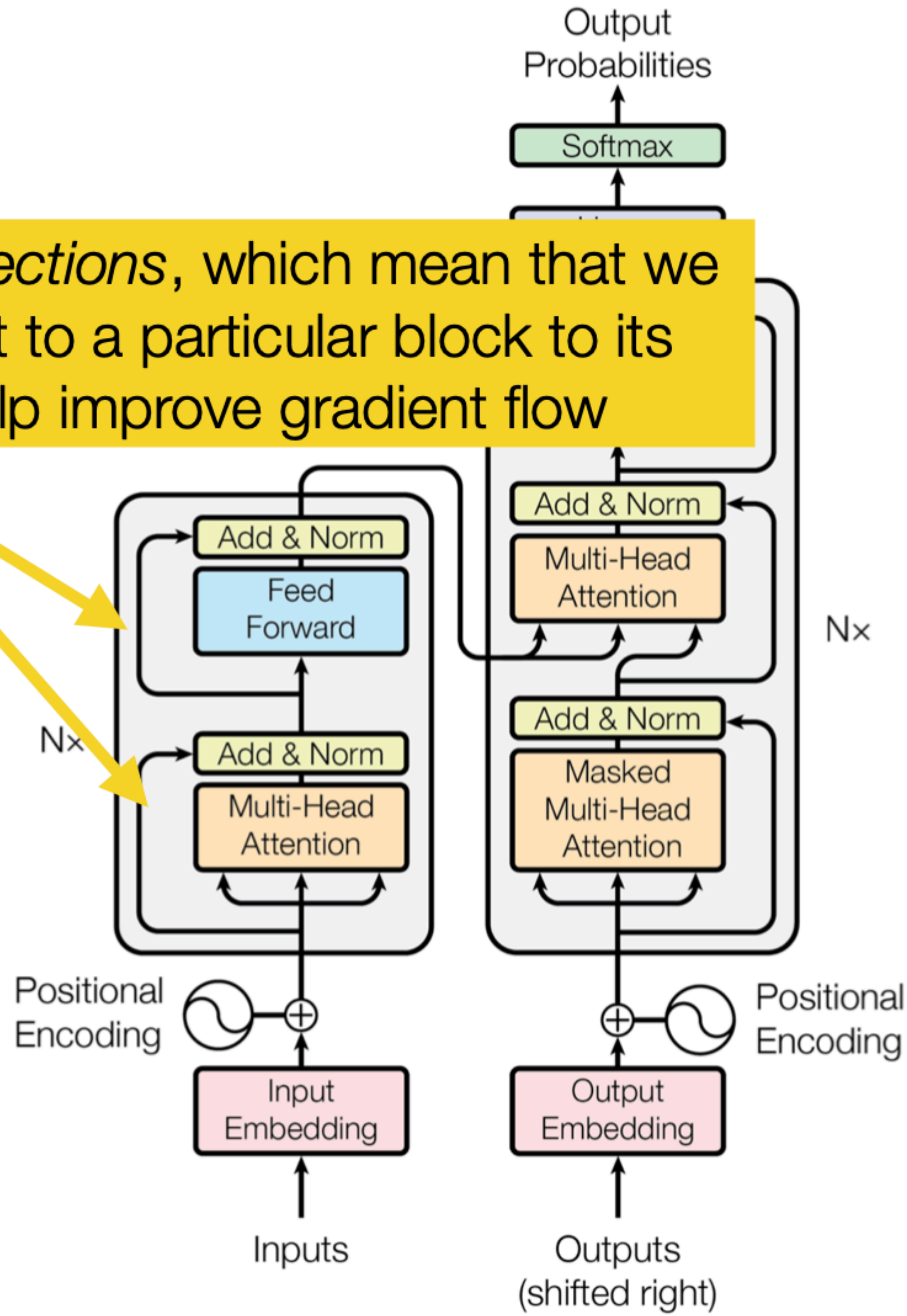


encoder
decoder
Output
Probabilities
Softmax
Linear
Add & Norm
Feed
Forward
Add & Norm
Multi-Head
Attention
Add & Norm
Masked
Multi-Head
Attention
Nx
Positional
Encoding
Output
Embedding
Outputs
(shifted right)
Add & Norm
Feed
Forward
Add & Norm
Multi-Head
Attention
Nx
Positional
Encoding
Input
Embedding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

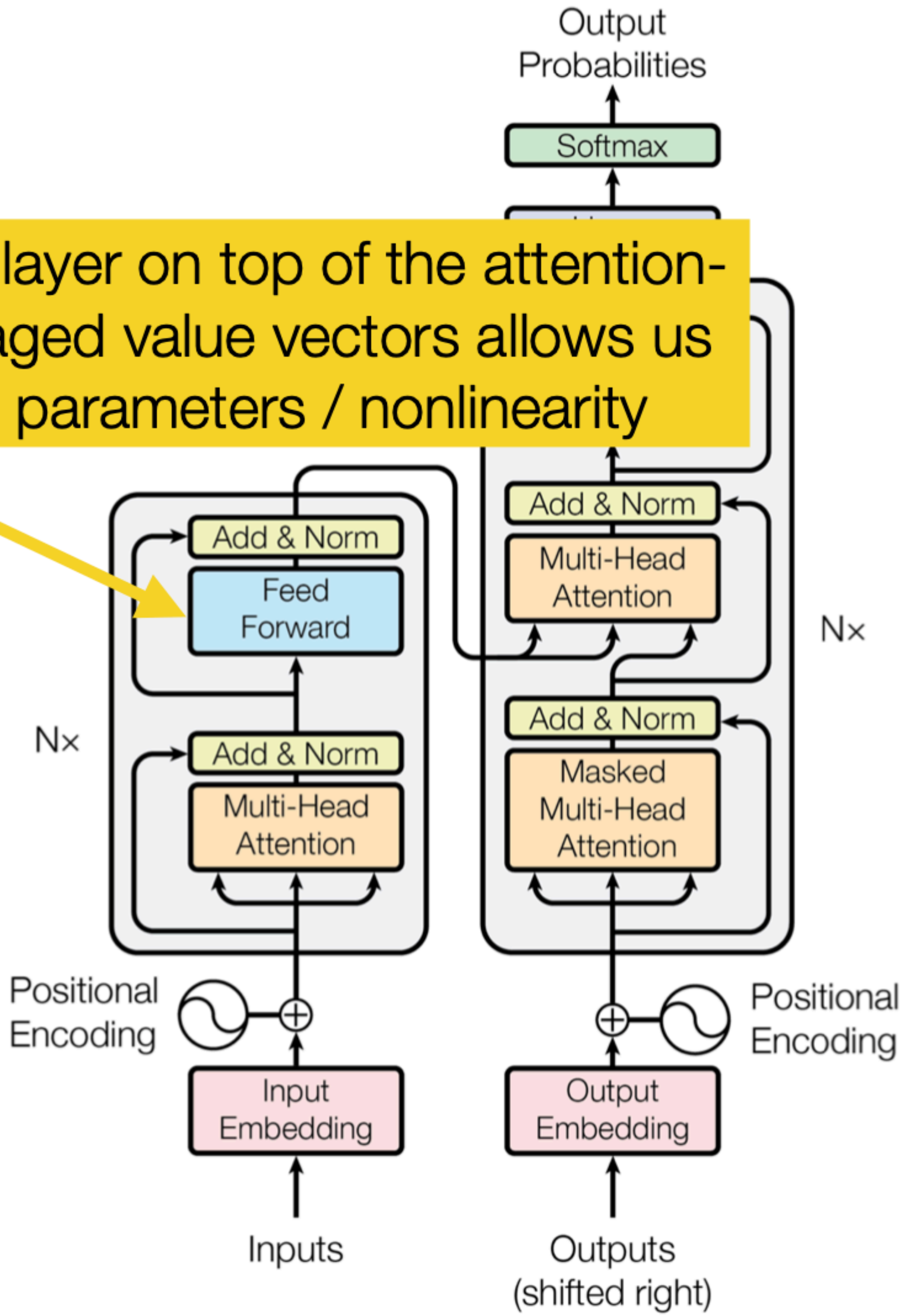
Position embeddings are *added* to each word embedding. Otherwise, since we have no recurrence, our model is unaware of the position of a word in the sequence!



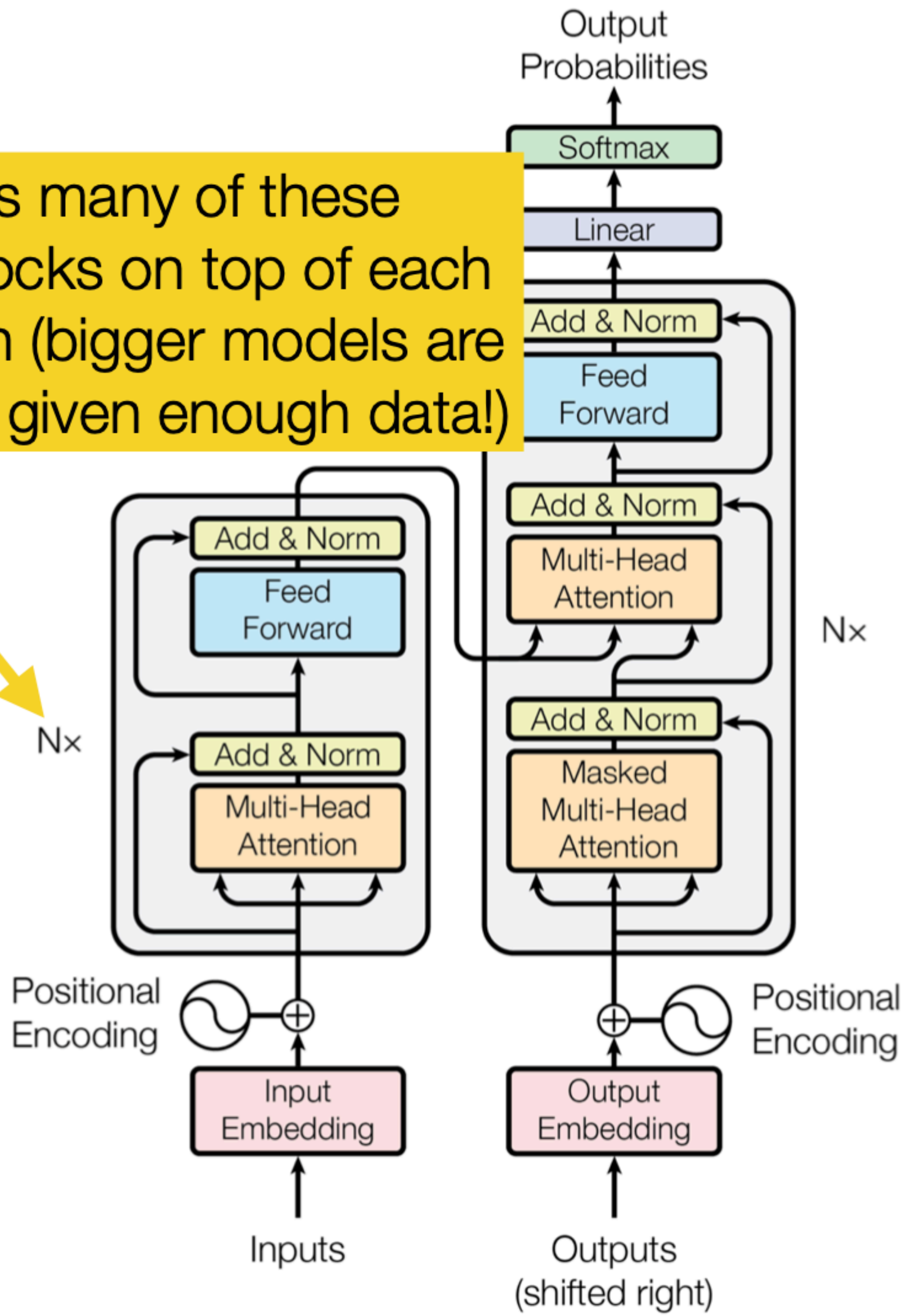
Residual connections, which mean that we add the input to a particular block to its output, help improve gradient flow



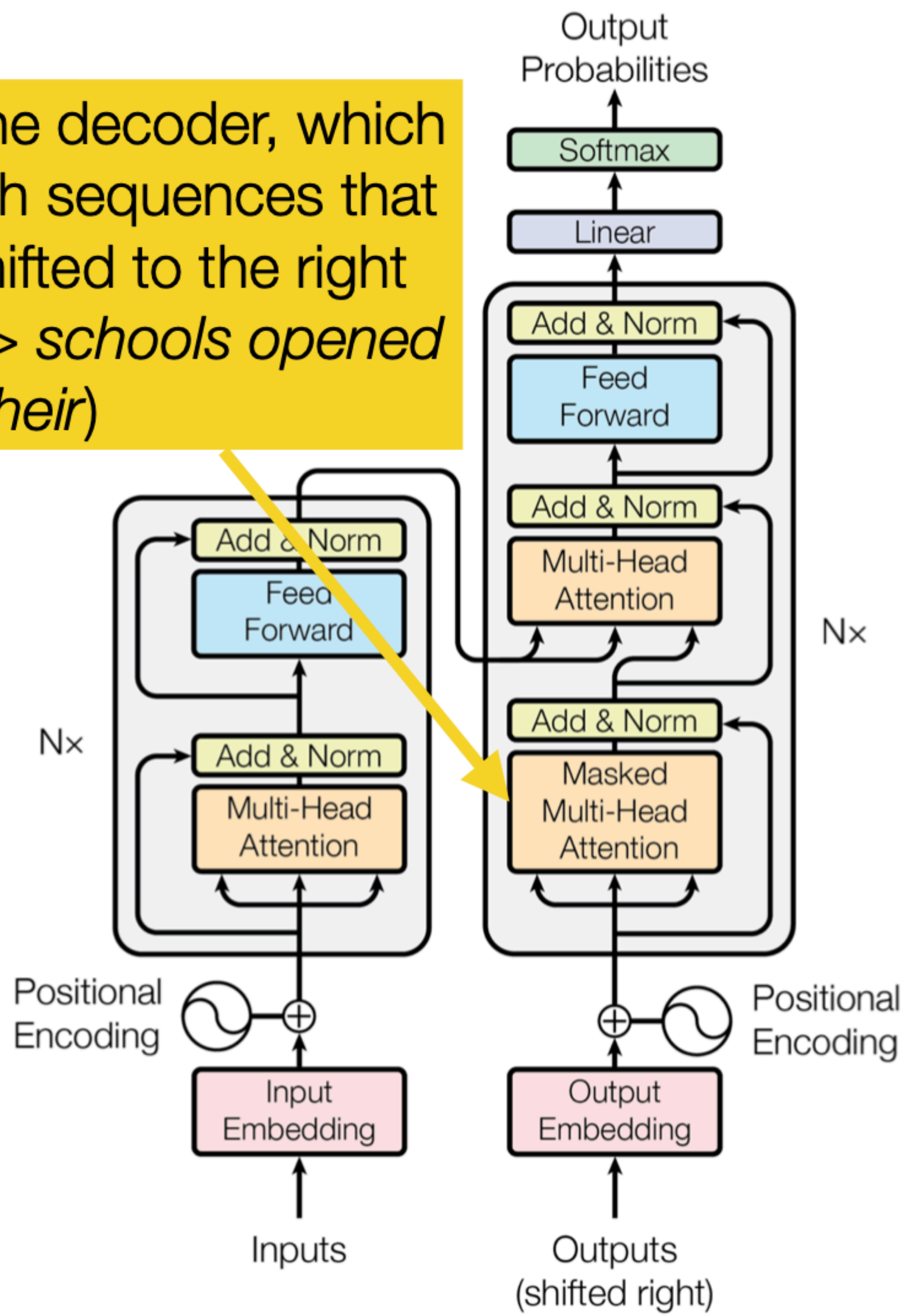
A feed-forward layer on top of the attention-weighted averaged value vectors allows us to add more parameters / nonlinearity



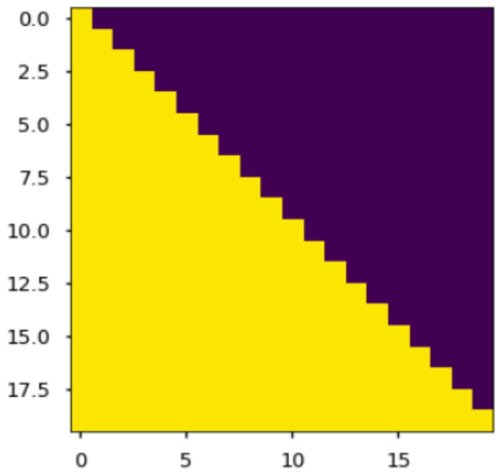
We stack as many of these *Transformer* blocks on top of each other as we can (bigger models are generally better given enough data!)

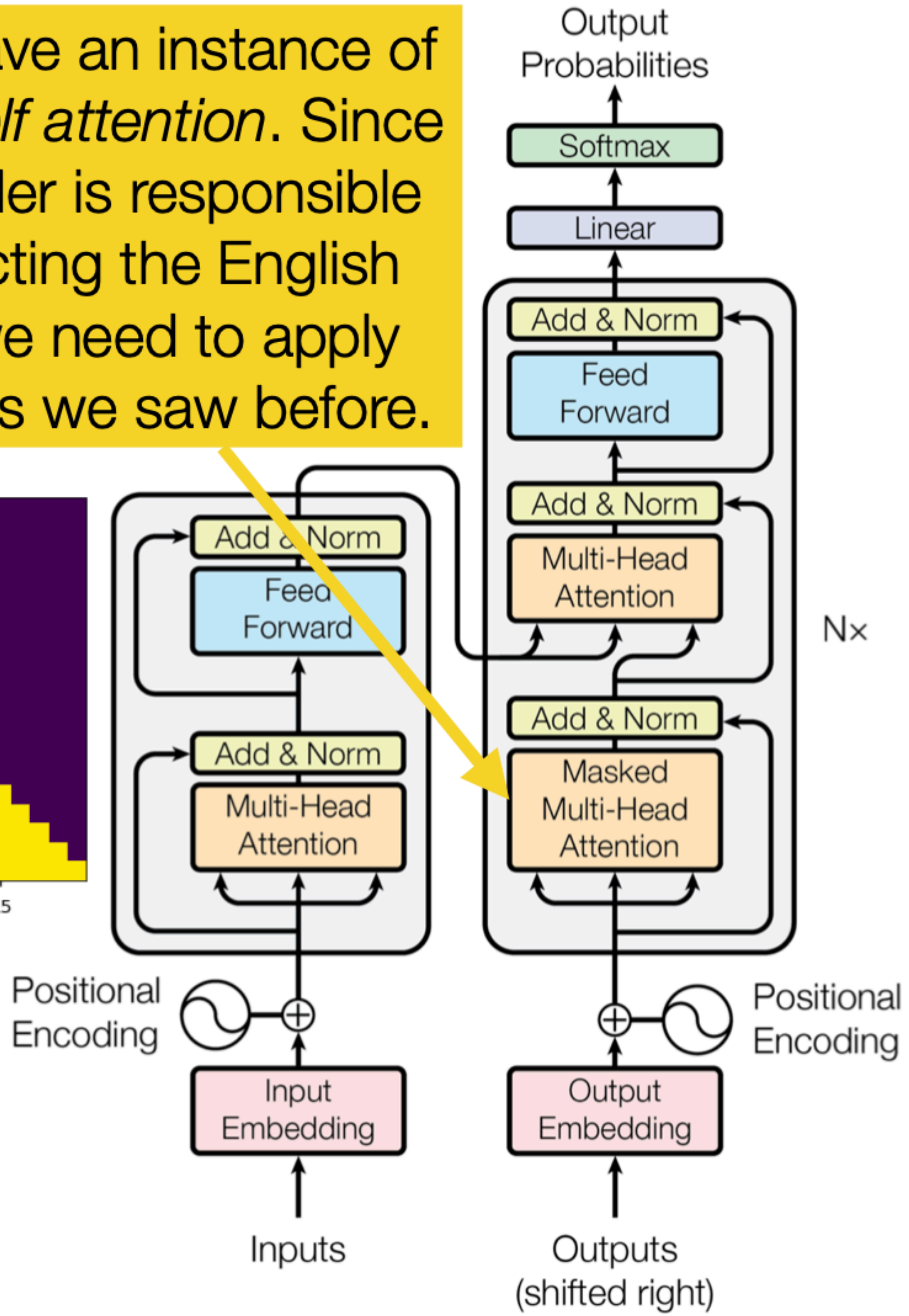


Moving onto the decoder, which takes in English sequences that have been shifted to the right (e.g., *<START> schools opened their*)



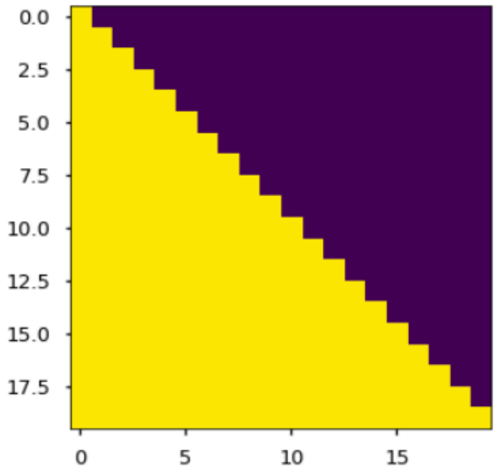
We first have an instance of *masked self attention*. Since the decoder is responsible for predicting the English words, we need to apply masking as we saw before.

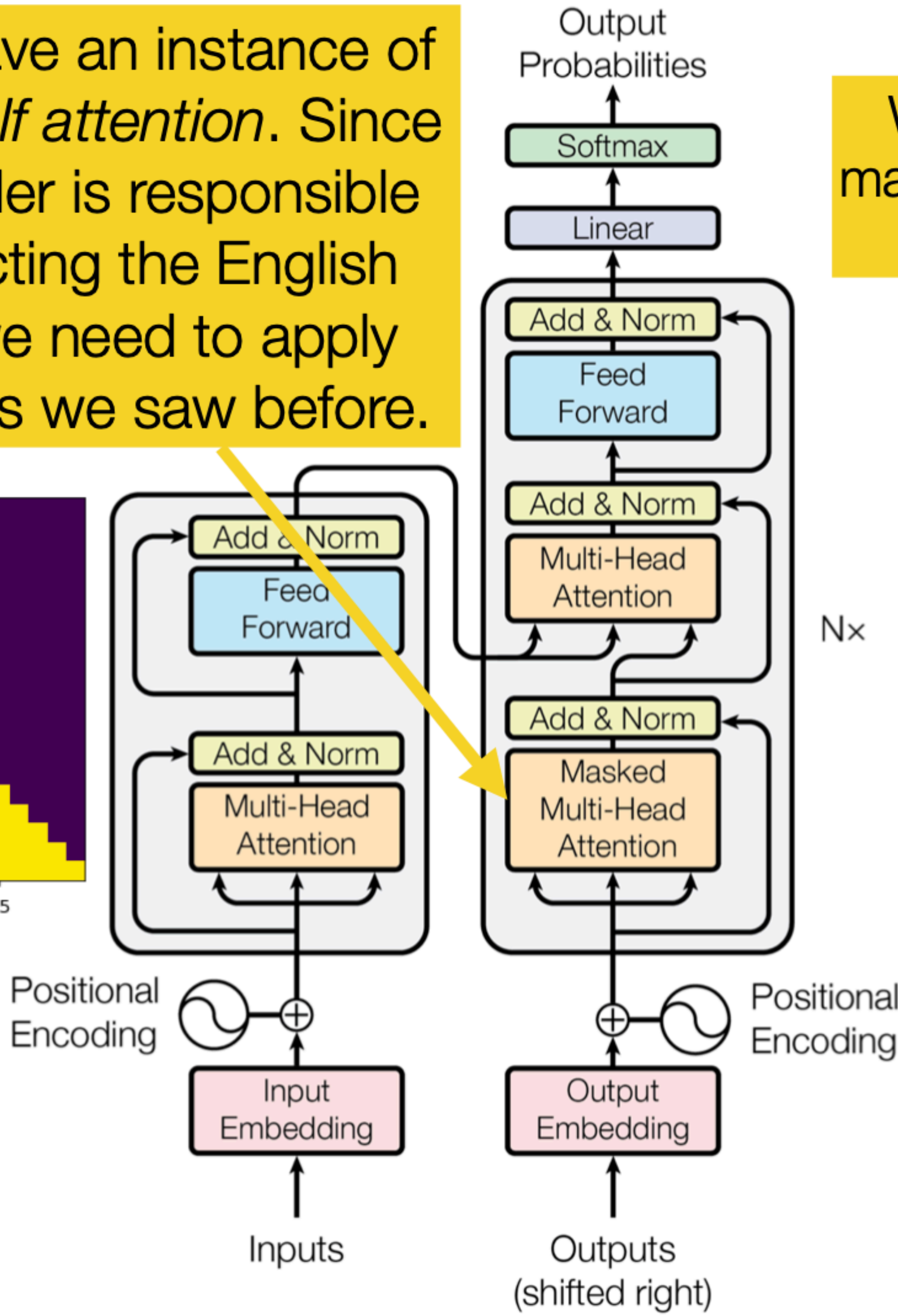




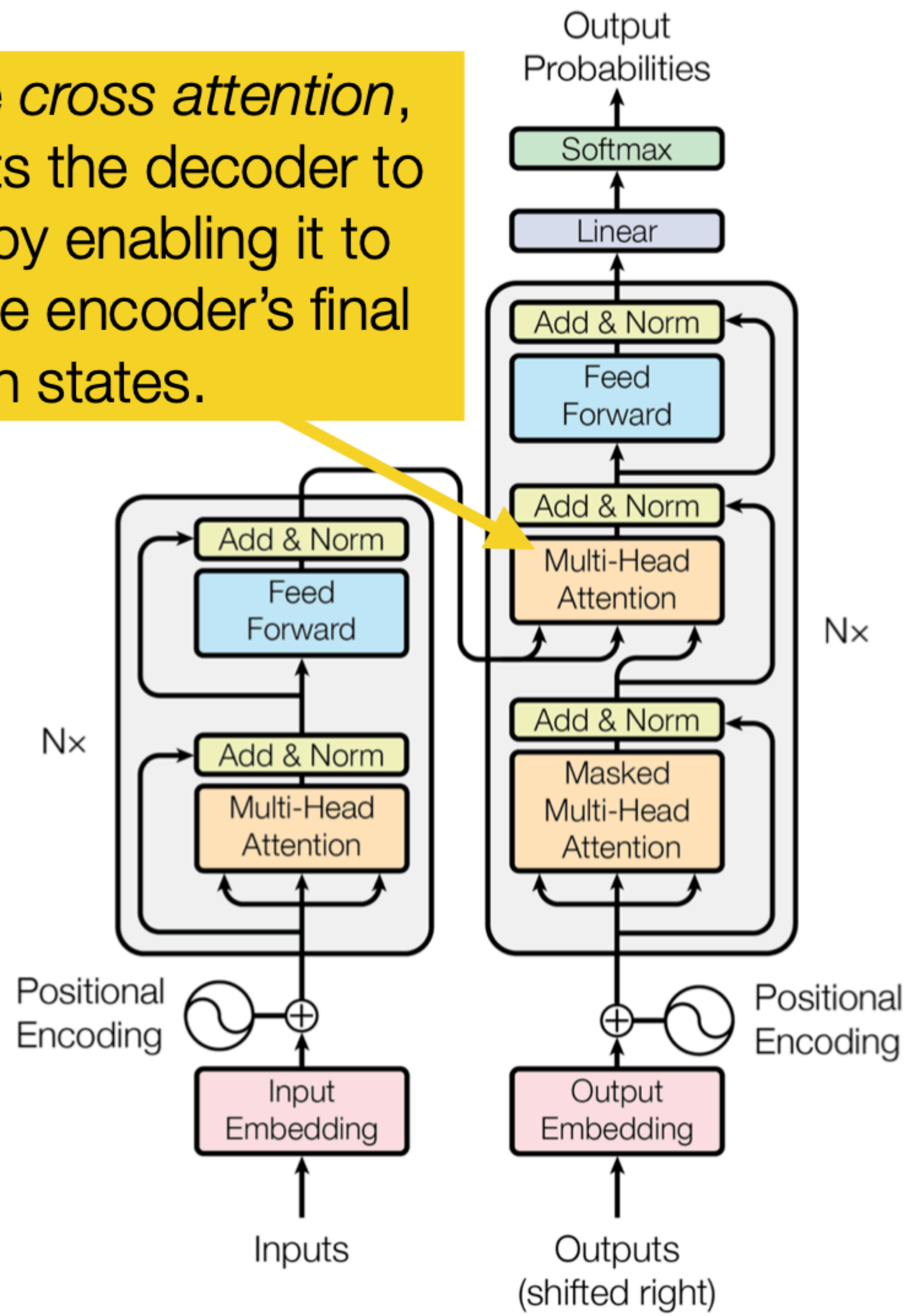
We first have an instance of *masked self attention*. Since the decoder is responsible for predicting the English words, we need to apply masking as we saw before.

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

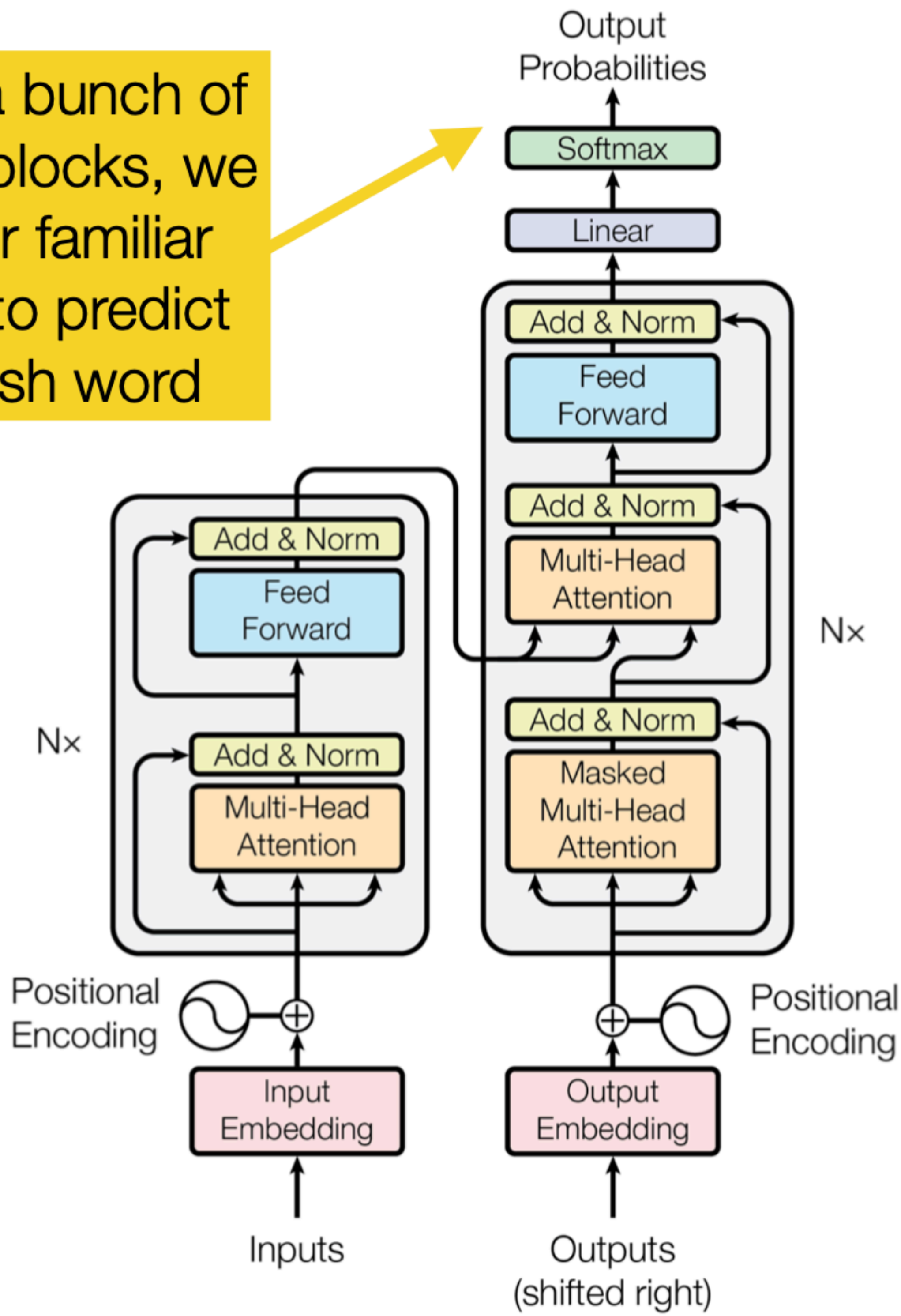




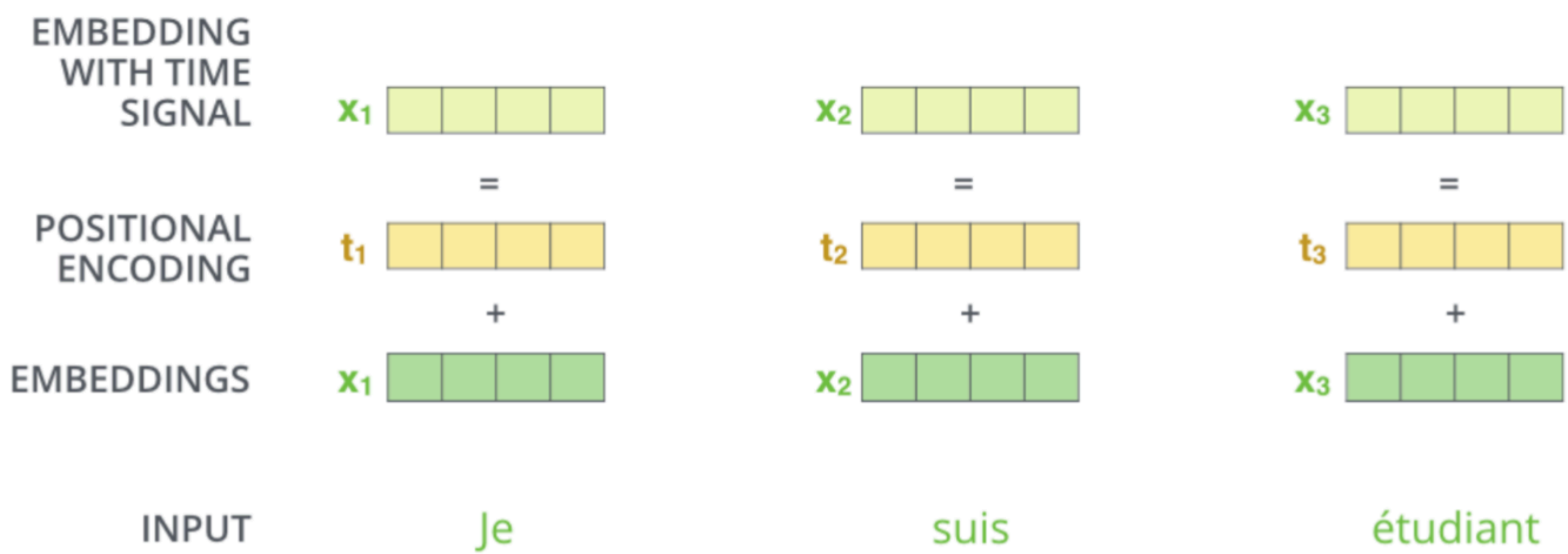
Now, we have *cross attention*, which connects the decoder to the encoder by enabling it to attend over the encoder's final hidden states.



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



Positional encoding



Creating positional encodings?

- We could just concatenate a fixed value to each time step (e.g., 1, 2, 3, … 1000) that corresponds to its position, but then what happens if we get a sequence with 5000 words at test time?

- We want something that can generalize to arbitrary sequence lengths. We also may want to make attending to *relative positions* (e.g., tokens in a local window to the current token) easier.

- Distance between two positions should be consistent with variable-length inputs

Intuitive example

$0:$	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/

Transformer positional encoding

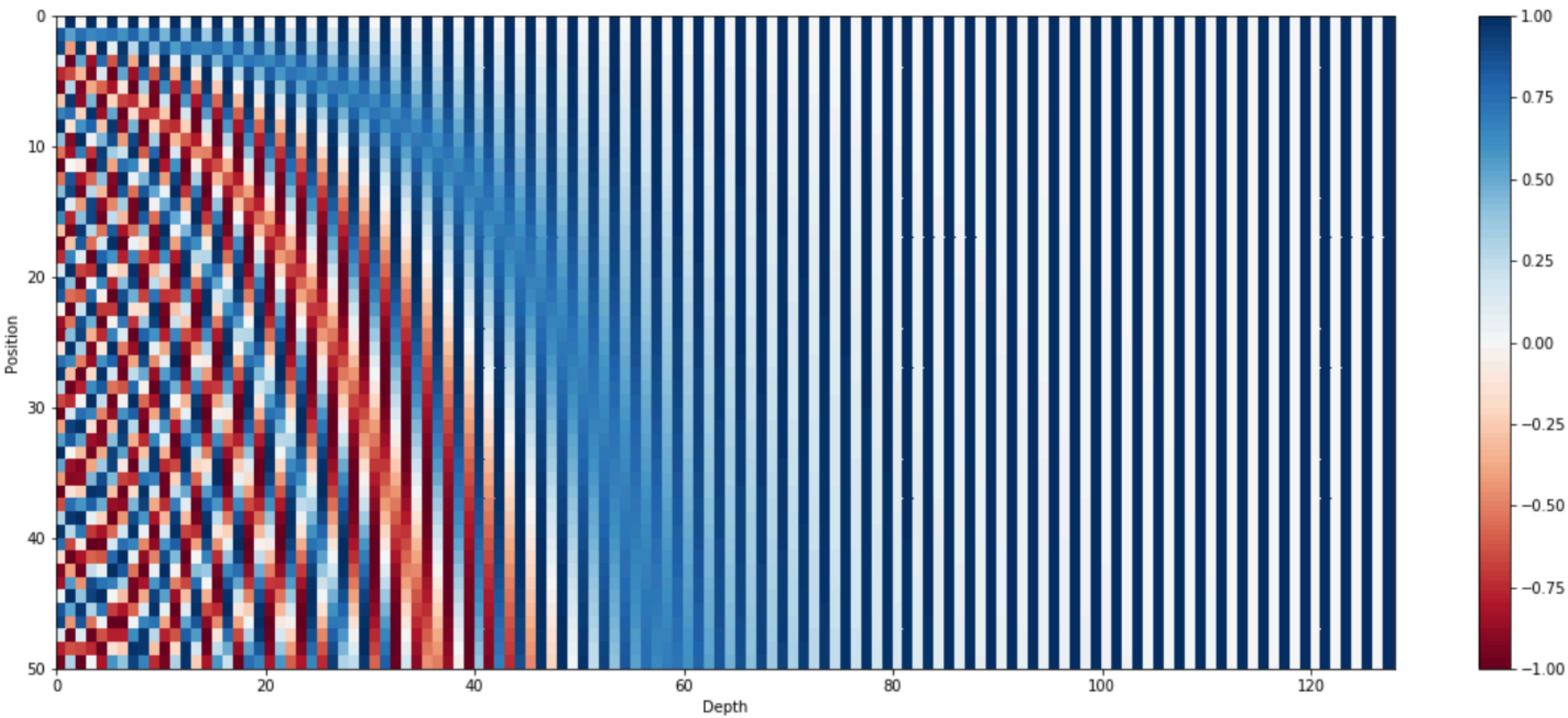
$$PE_{(pos,2i)} = \sin(\frac{pos}{10000^{2i/d_{model}}})$$

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

Positional encoding is a 512d vector
i = a particular dimension of this vector
pos = dimension of the word
d_model = 512

What does this look like?

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



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

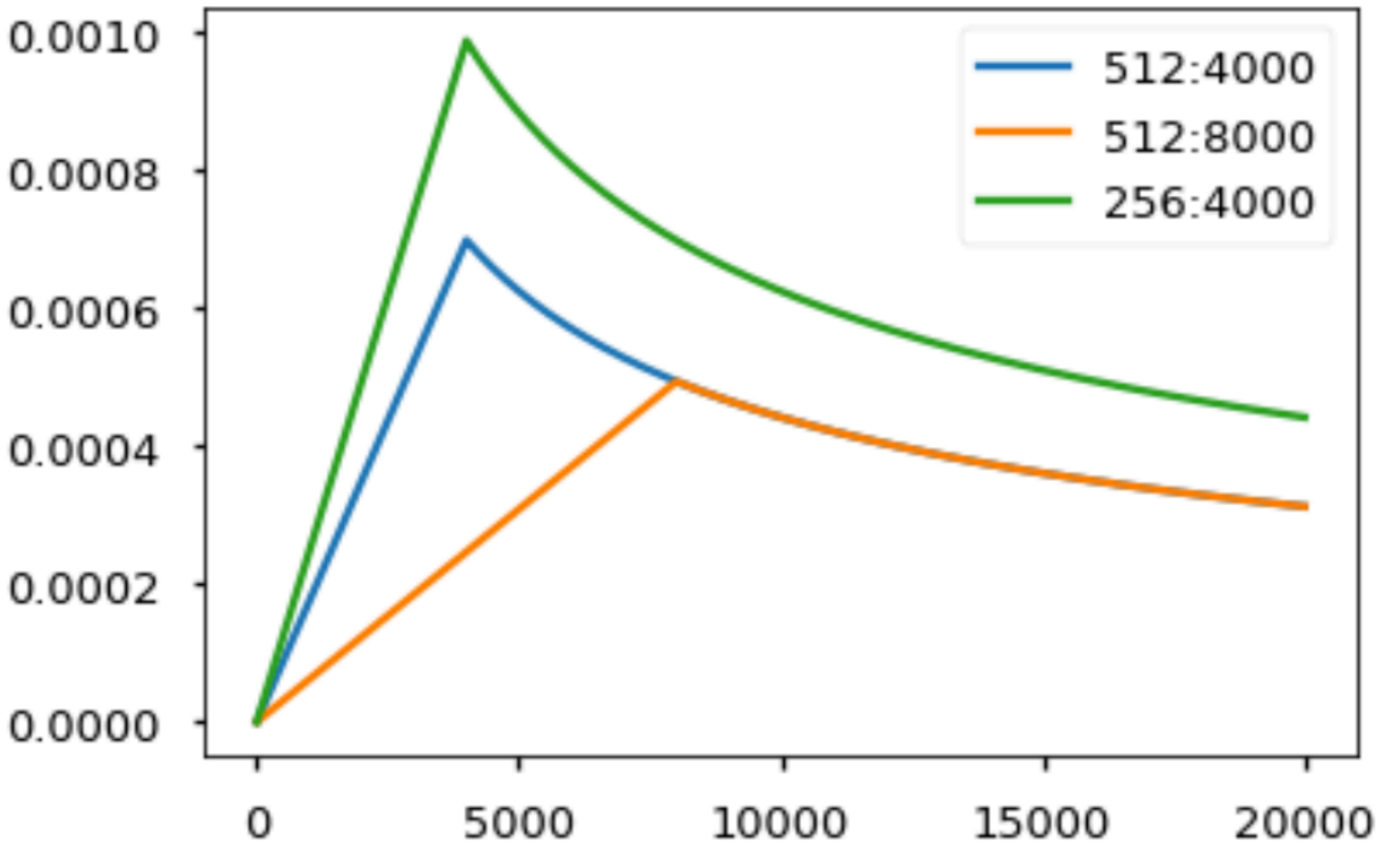
Despite the intuitive flaws, many models these days use *learned positional embeddings* (i.e., they cannot generalize to longer sequences, but this isn't a big deal for their use cases)

Hacks to make Transformers work

Optimizer

We used the Adam optimizer (cite) with $\beta_1 = 0.9$, $\beta_2 = 0.98$ and $\epsilon = 10^{-9}$. We varied the learning rate over the course of training, according to the formula: $lrate = d_{\text{model}}^{-0.5} \cdot \min(step\_num^{-0.5}, step\_num \cdot warmup\_steps^{-1.5})$ This corresponds to increasing the learning rate linearly for the first $warmup_steps$ training steps, and decreasing it thereafter proportionally to the inverse square root of the step number. We used $warmup_steps = 4000$.

> *Note: This part is very important. Need to train with this setup of the model.*



Label Smoothing

During training, we employed label smoothing of value $\epsilon_{ls} = 0.1$ (cite). This hurts perplexity, as the model learns to be more unsure, but improves accuracy and BLEU score.

> *We implement label smoothing using the KL div loss. Instead of using a one-hot target distribution, we create a distribution that has* `confidence` *of the correct word and the rest of the* `smoothing` *mass distributed throughout the vocabulary.*

I went to class and took ___

cats	TV	notes	took	sofa
0	0	1	0	0

Label Smoothing

During training, we employed label smoothing of value $\epsilon_{ls} = 0.1$ (cite). This hurts perplexity, as the model learns to be more unsure, but improves accuracy and BLEU score.

> *We implement label smoothing using the KL div loss. Instead of using a one-hot target distribution, we create a distribution that has `confidence` of the correct word and the rest of the `smoothing` mass distributed throughout the vocabulary.*

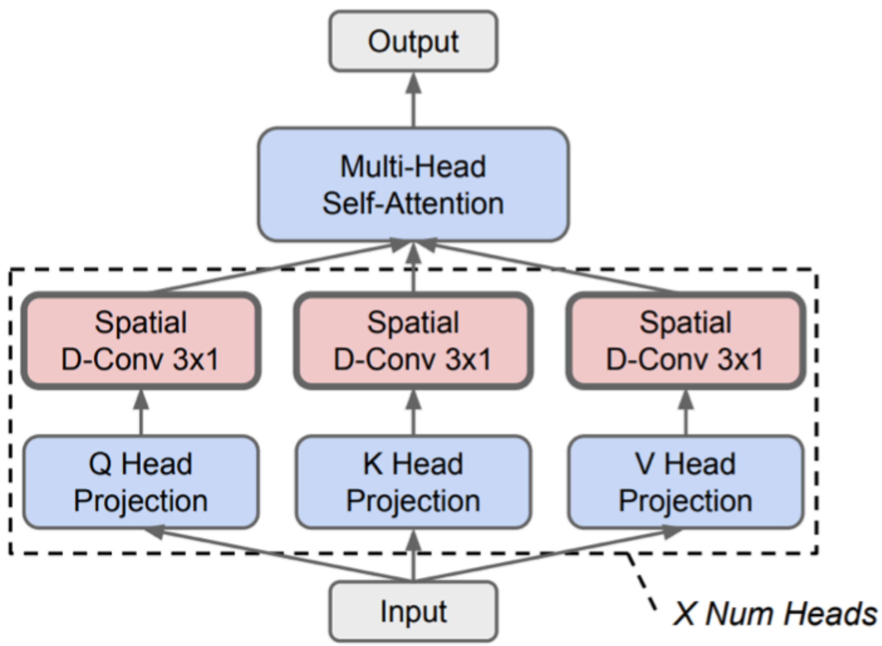
I went to class and took ___

	cats	TV	notes	took	sofa
	0	0	1	0	0
with label smoothing	0.025	0.025	0.9	0.025	0.025

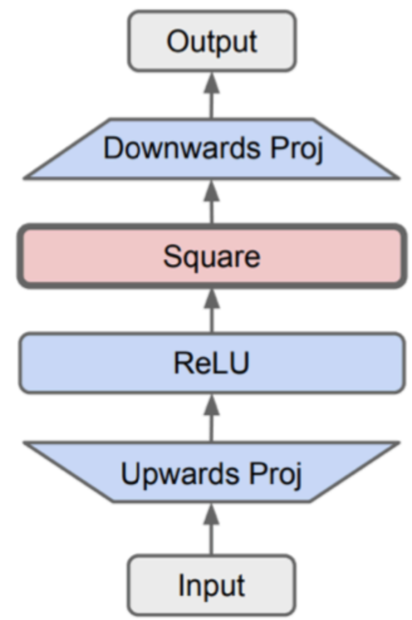
Why these decisions?

Unsatisfying answer: they empirically worked well.
Neural architecture search finds even better Transformer variants:

Multi-DConv-Head Attention (MDHA)



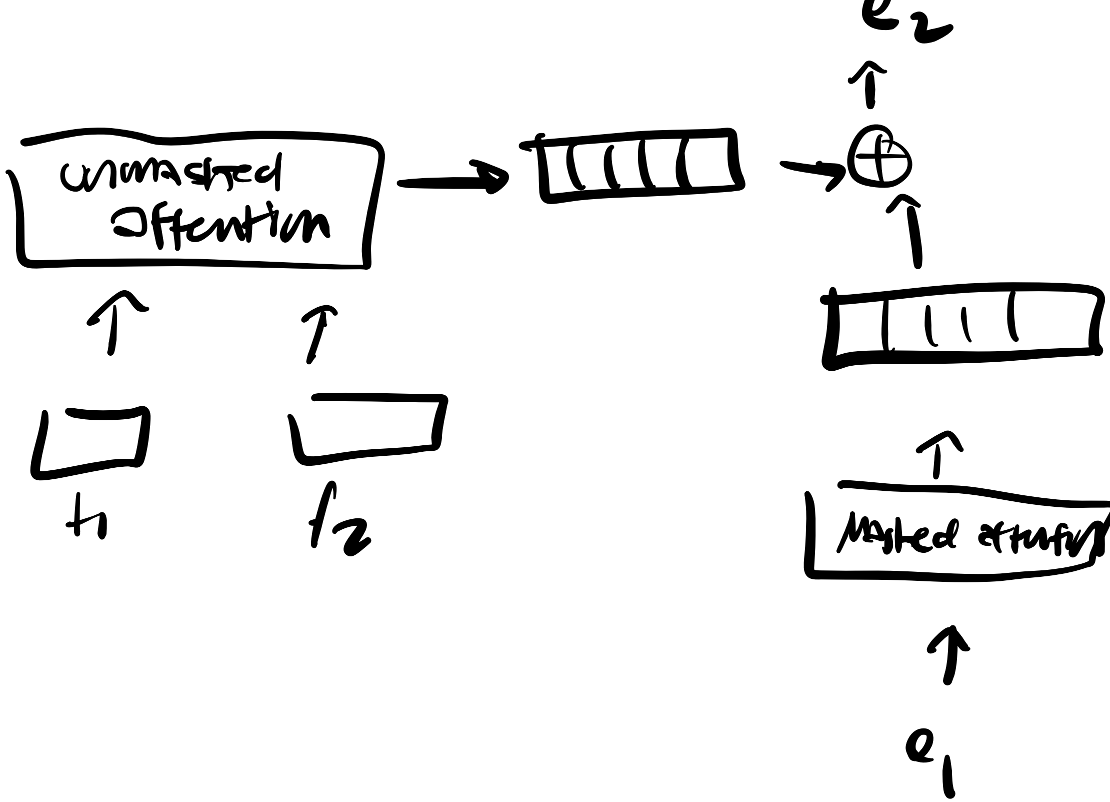
Squared ReLU in Feed Forward Block



Primer: Searching for efficient Transformer architectures… So et al., Sep. 2021

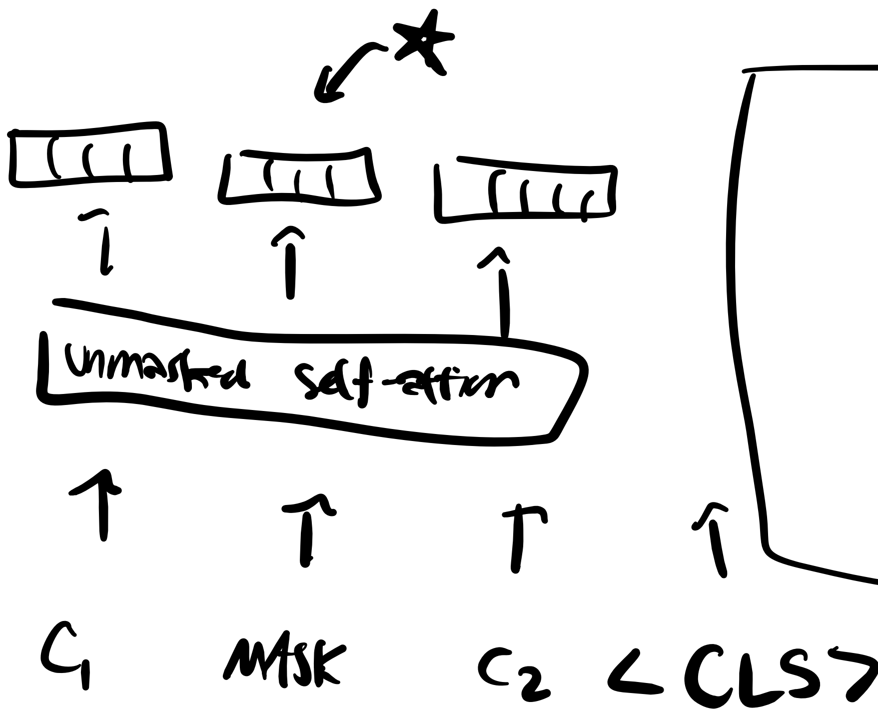
Common Transformers

* Encoder - Decoder : Useful for - machine translation
  - conditional text generation



Common Transformers

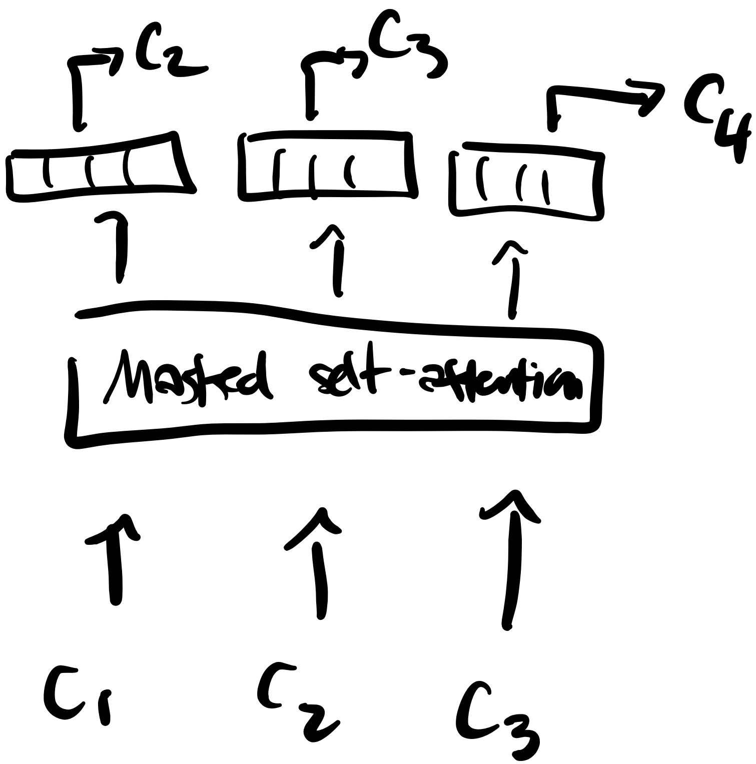
Encoder only : BERT



Useful for representation learning

Common Transformers

Decoder only: . GPT-3



Useful for text generation

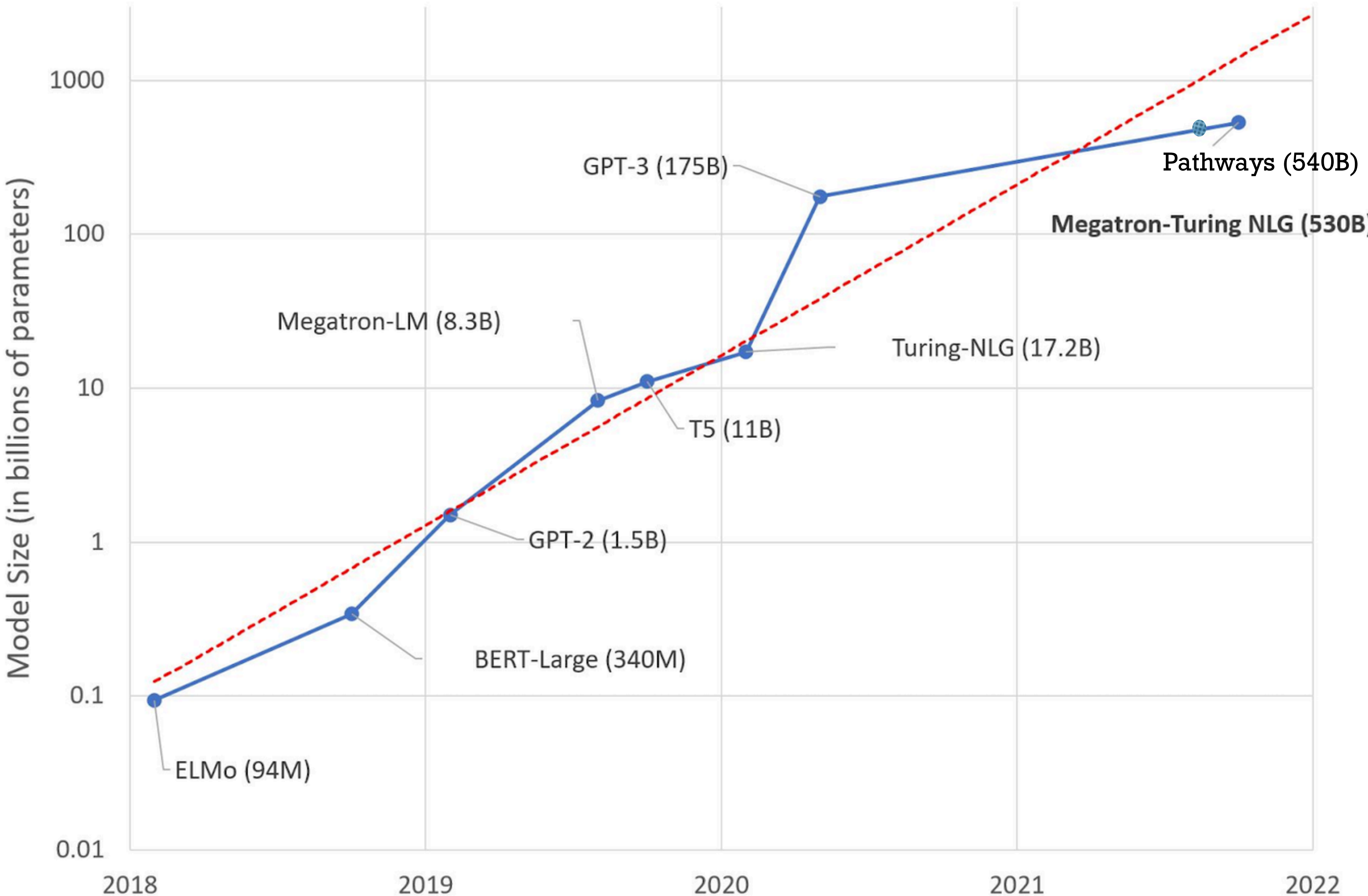
le chat rouge . . . <ENG> The cat . . .

The Costs of Deep Learning

OpenAI’s Transformer LMs

- GPT (Jun 2018): 117 million parameters, trained on 13GB of data (~1 billion tokens)
- GPT2 (Feb 2019): 1.5 billion parameters, trained on 40GB of data
- GPT3 (July 2020): 175 billion parameters, ~500GB data (300 billion tokens)

Models keep getting larger



https://huggingface.co/blog/large-language-models

These models are really expensive!

Megatron (530 billion parameters), Microsoft's GPT-3 competitor, cost around **$100 million** to train

These models are really expensive!

www.lesswrong.com/posts/midXmMb2Xg37F2Kgn/new-scaling-laws-for-large-language-models

Model	Size (# Parameters)	Training Tokens
LaMDA (Thoppilan et al., 2022)	137 Billion	168 Billion
GPT-3 (Brown et al., 2020)	175 Billion	300 Billion
Jurassic (Lieber et al., 2021)	178 Billion	300 Billion
Gopher (Rae et al., 2021)	280 Billion	300 Billion
MT-NLG 530B (Smith et al., 2022)	530 Billion	270 Billion
Pathways (Chowdhery et al. 2022)	540 Billion	780 Billion

By contrast: children are exposed to around ~28,470,000* words in their critical language acquisition period.

* my back-of-envelope calculation from L1 acquisition studies where children are recorded 12 hr/day

These models are really expensive!

Consumption	CO_2e (lbs)
Air travel, 1 person, NY↔SF	1984
Human life, avg, 1 year	11,023
American life, avg, 1 year	36,156
Car, avg incl. fuel, 1 lifetime	126,000
Training one model (GPU)	
NLP pipeline (parsing, SRL)	39
w/ tuning & experiments	78,468
Transformer (big)	192
w/ neural arch. search	626,155

Table 1: Estimated CO_2 emissions from training common NLP models, compared to familiar consumption.[1]

Strubell, Ganesh, & McCallum (2019)

Emma Strubell

These models are really expensive!

BERT-L (340 million parameters) had a **carbon footprint** equivalent to a trans-American flight.

And remember:

Microsoft Megatron has 530 **billion** parameters...

Google Pathways has 540 **billion** parameters...