Transformers and sequence-to-sequence learning

Guest Lecture, Fall 2021

Andrew Drozdov
College of Information and Computer Sciences
University of Massachusetts Amherst

some slides from Emma Strubell and Mohit Iyyer

Overview

- Background: Word Embeddings and Bi-RNNs (link)

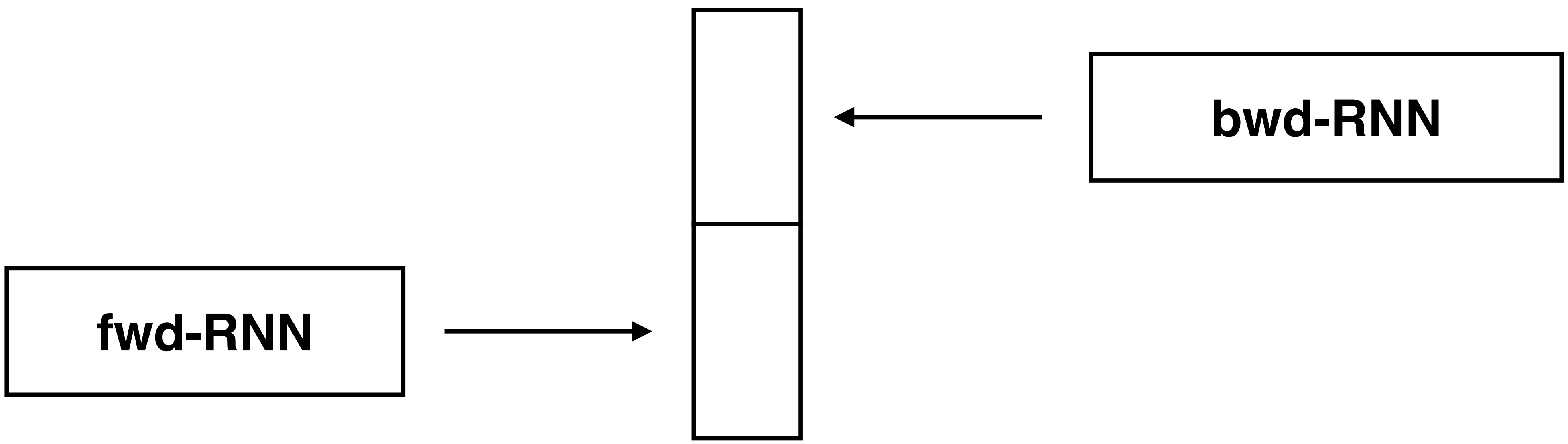
- Word Embeddings: **semantically rich** and **static**.

  Doesn’t capture multiple word meanings well.

  “bank”
  - verb: put basketball in hoop
  - noun: place to deposit money
  - noun: a water structure

Overview

- Background: Word Embeddings and Bi-RNNs (link)
- Word Embeddings: **semantically rich** and **static**.
- Bi-RNNs: **incorporates context** and **sequential**.



Few had the impact of Sondheim, shaping modern musicals.

Figure inspired by Wei Xu's example with ballet dancer

Overview

- Background: Word Embeddings and Bi-RNNs (link)
- Word Embeddings: **semantically rich** and **static**.
- Bi-RNNs: **incorporates context** and **sequential**.
- What about methods that are rich, contextual, and more flexible than sequential Bi-RNNs?

  Today's lecture: **Transformers**

  Running example: sequence-to-sequence learning

sequence-to-sequence learning

Used when inputs and outputs are both sequences of words (e.g., machine translation, summarization)

- we'll use French (*f*) to English (*e*) as a running example
- **goal**: given French sentence *f* with tokens f_1, f_2, … f_n produce English translation *e* with tokens e_1, e_2, … e_m
- **real goal**: compute $\arg\max_{e} p(e \mid f)$

This is an instance of *conditional language modeling*

$$
\begin{aligned}
p(e \mid f) &= p(e_1, e_2, \ldots, e_m \mid f) \\
&= p(e_1 \mid f) \cdot p(e_2 \mid e_1, f) \cdot p(e_3 \mid e_2, e_1, f) \cdot \ldots \\
&= \prod_{i=1}^{m} p(e_i \mid e_1, \ldots, e_{i-1}, f)
\end{aligned}
$$

Just like in LM, except we additionally condition our prediction of the next word on some other input (here, the French sentence)

seq2seq models

- use two different neural networks to model

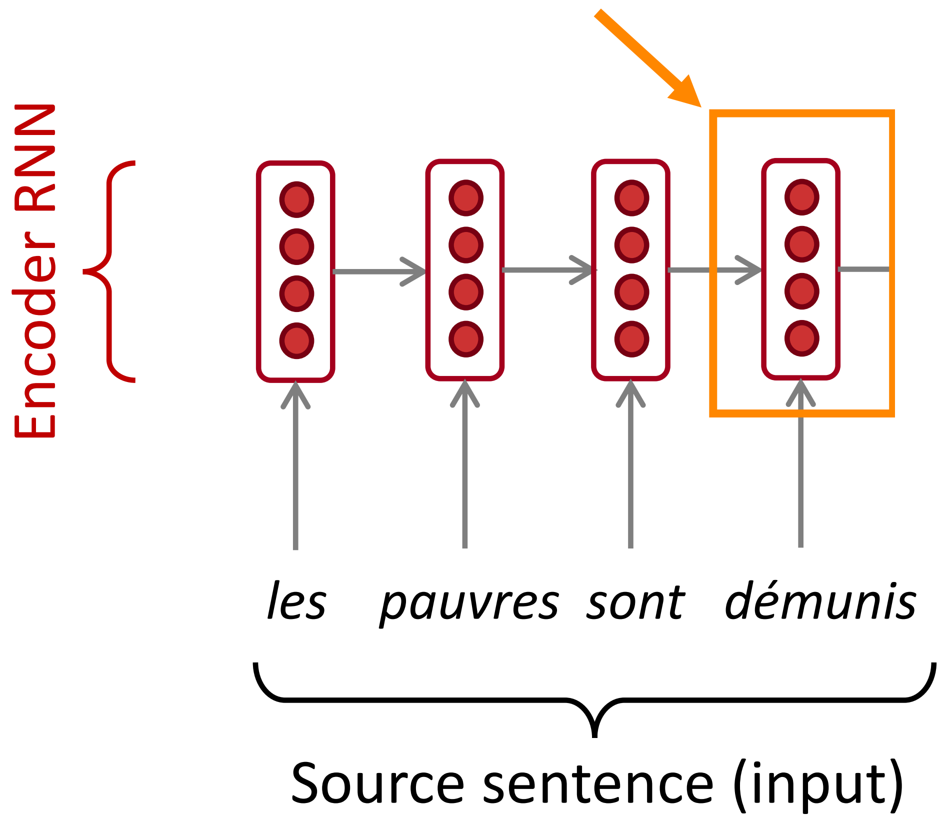
$$\prod_{i=1}^{L} p(e_i \mid e_1, \ldots, e_{i-1}, f)$$

- first we have the ***encoder***, which encodes the French sentence *f*
- then, we have the ***decoder***, which produces the English sentence *e*

Neural Machine Translation (NMT)

The sequence-to-sequence model

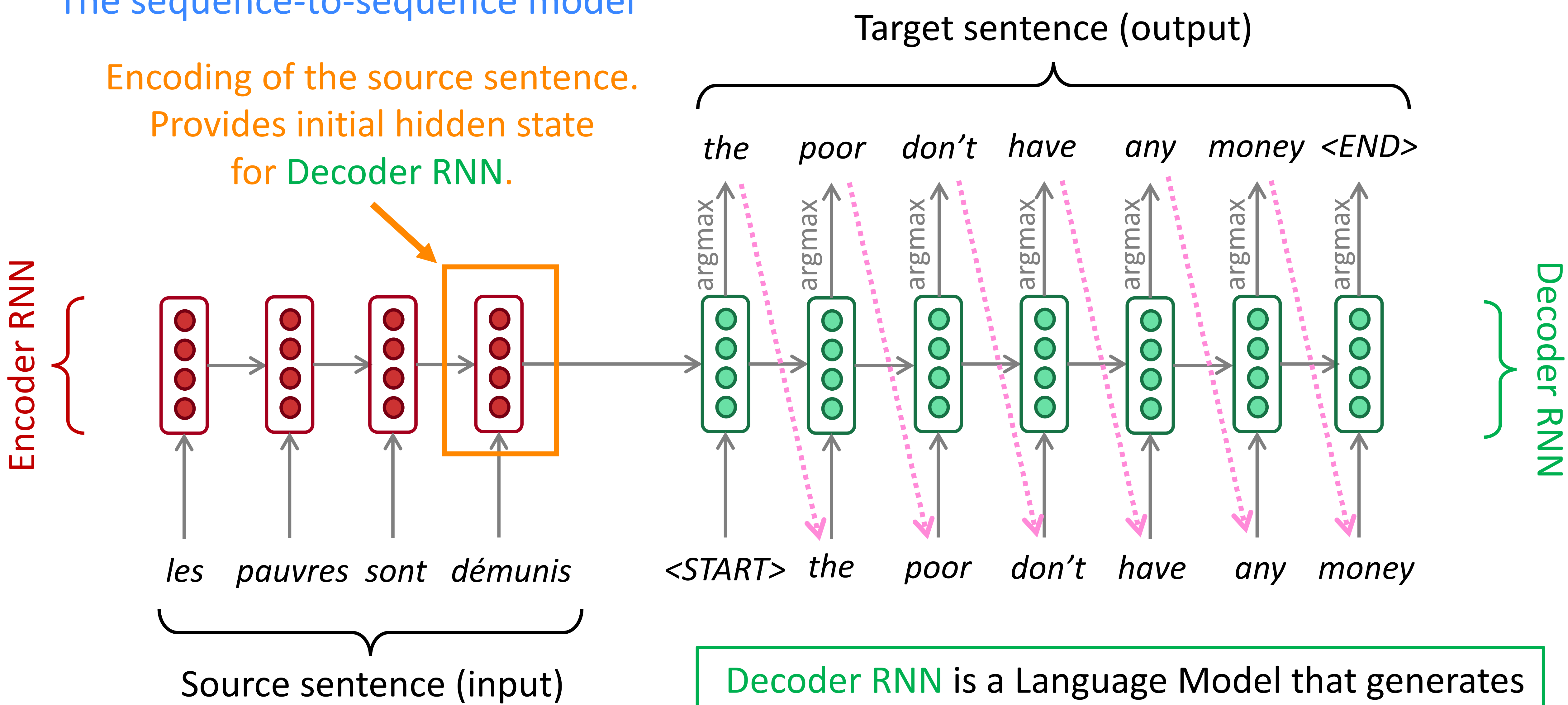
Encoding of the source sentence.
Provides initial hidden state
for Decoder RNN.



Encoder RNN produces an encoding of the source sentence.

Neural Machine Translation (NMT)

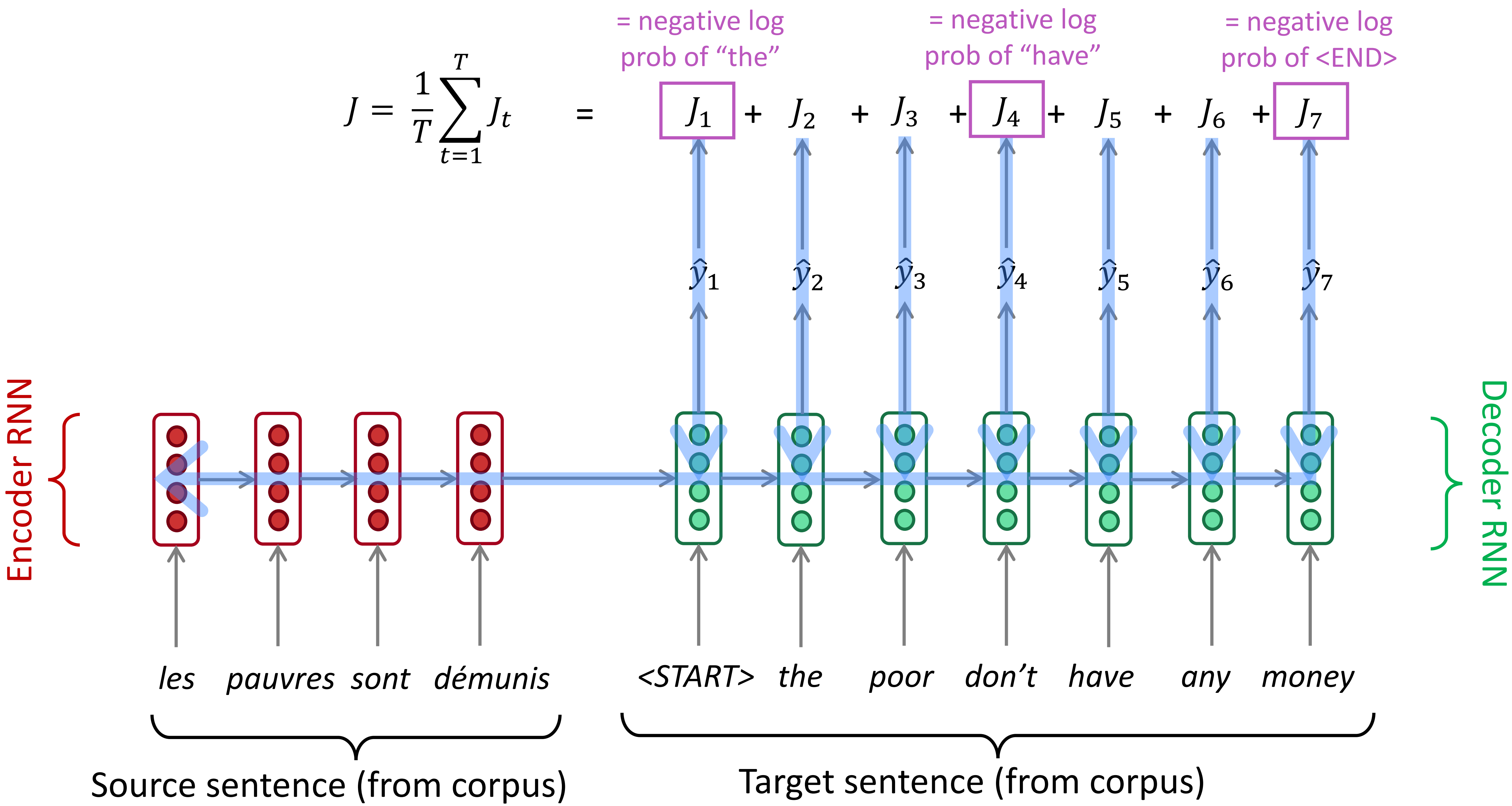
The sequence-to-sequence model



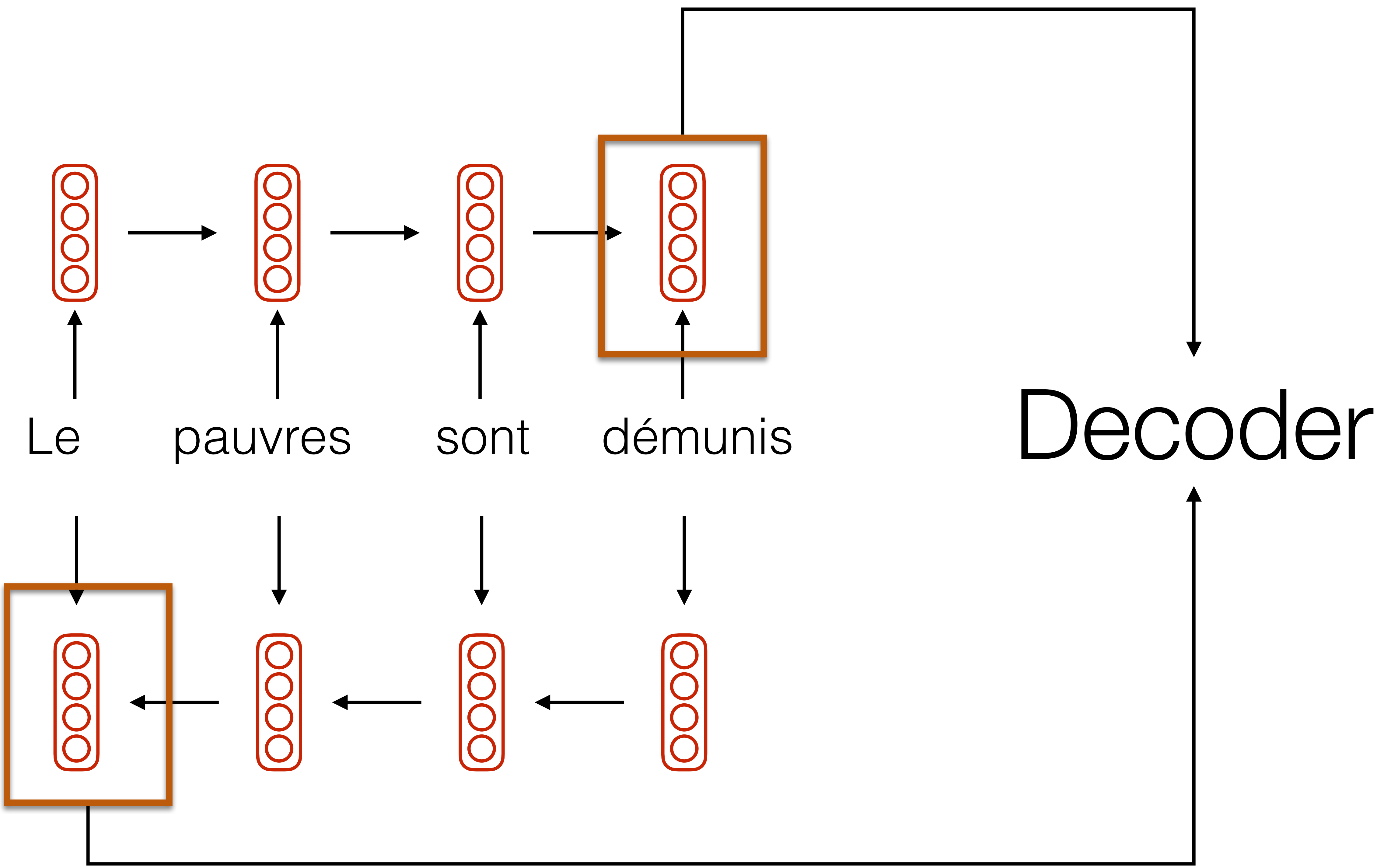
Decoder RNN is a Language Model that generates target sentence conditioned on encoding.

Encoder RNN produces an encoding of the source sentence.

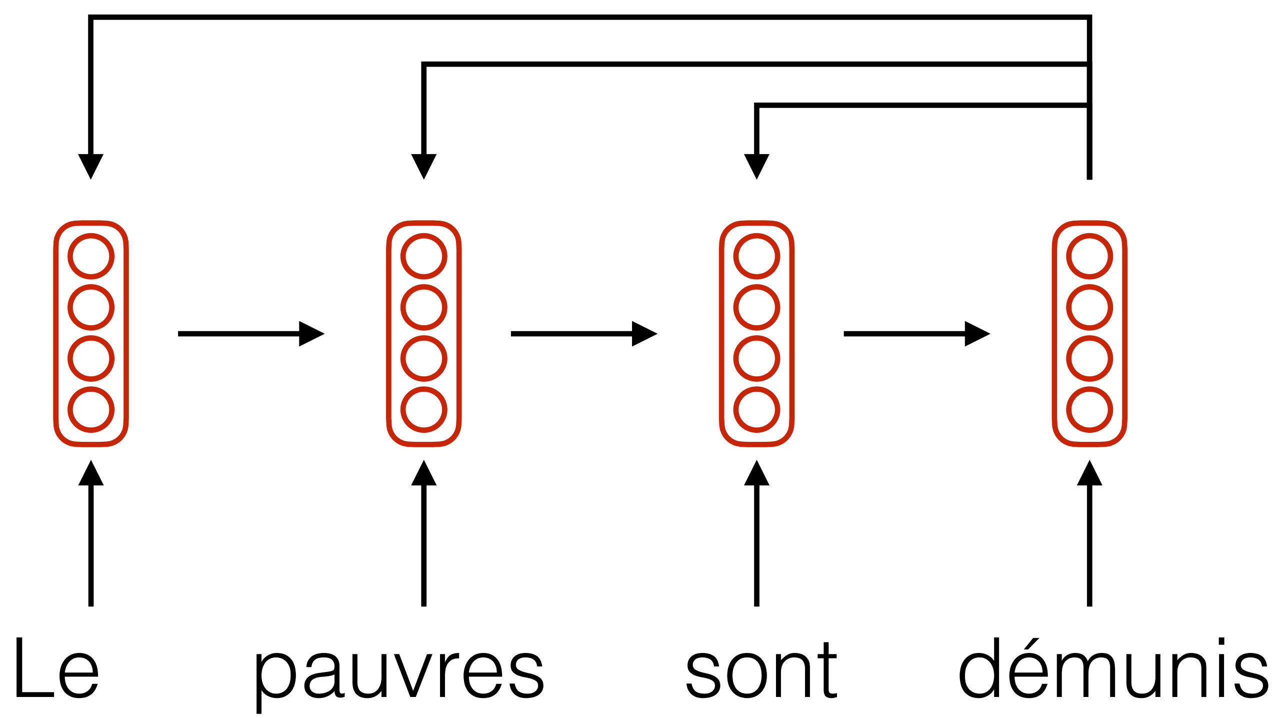
Training a Neural Machine Translation system



Improved Translation: Bi-RNN



Improved Translation: Attention



$$\tilde{h}_i = f(x_i, h_{i-1})$$

$$\alpha_{i,j} = \tilde{h}_i^\top h_j$$

$$v_i = \Sigma_{j<i}\ \alpha_{i,j}\ h_j$$

$$h_i = g(\tilde{h}_i, v_i)$$

Sequence-to-Sequence w/ RNN

Train Time	**Test Time**
Encoder	
• Runs iteratively, bi-directional.	
Decoder	
• Conditioned on full source + decoder history.	
• Runs iteratively, left-to-right.	
• Input is from “teacher forcing”.	• Input is from “own predictions”.

The “sequence-to-sequence” learning setup
(or “encoder-decoder”) is very natural for RNN.

How would this work with a transformer?

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 that include an encoder and a decoder. 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 41.8 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the

Transformers (Vaswani et al., 2017)

- Paper Title: "Attention Is All You Need"

- What is attention?

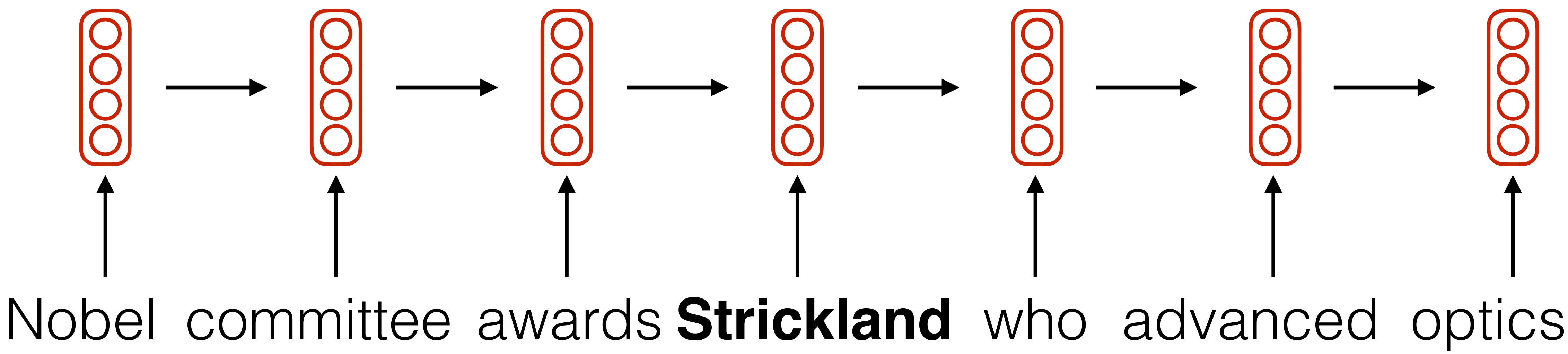
  Attention w/ RNN: Each word attends to words in its history.
  Self-attention: Each word attends to every other word.

- What do we *not* need? Recurrence

- How do we replace it? (Parikh et al., '16)

  Next slide, computational complexity of attention…
  Then we'll get to transformers.

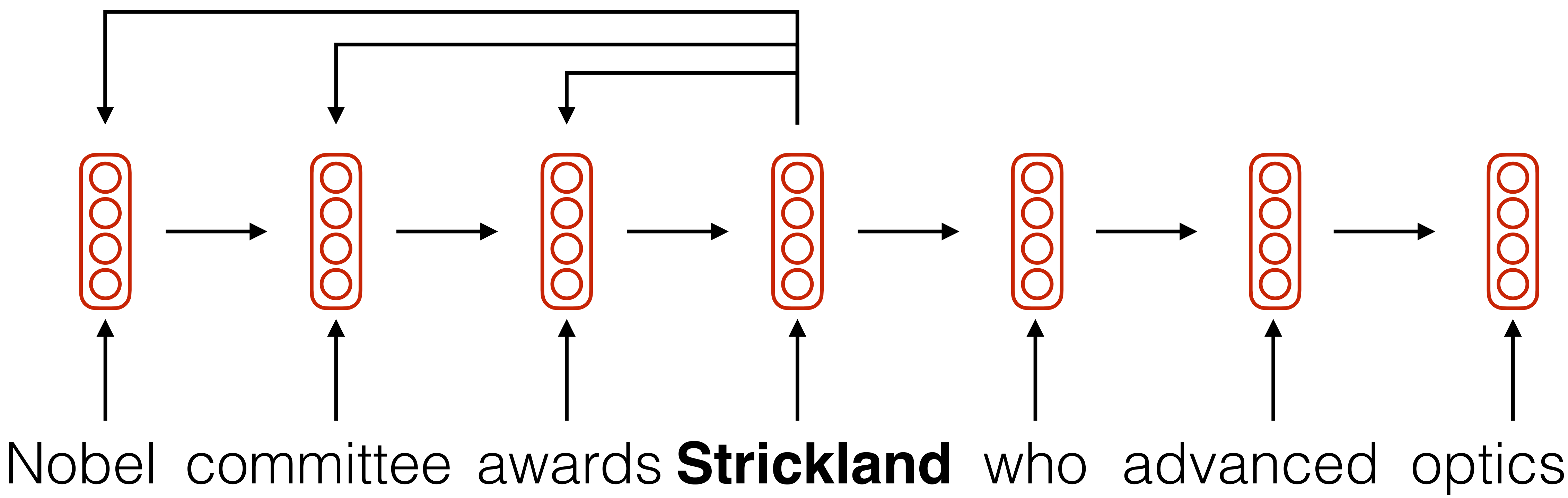
Review: Attention w/ RNN



One word: $O(D^2)$

D - embedding dimension
L - sequence length
Matrix Vector Multiplication is $O(D^2)$, Vector Vector Multiplication is $O(D)$

Review: Attention w/ RNN

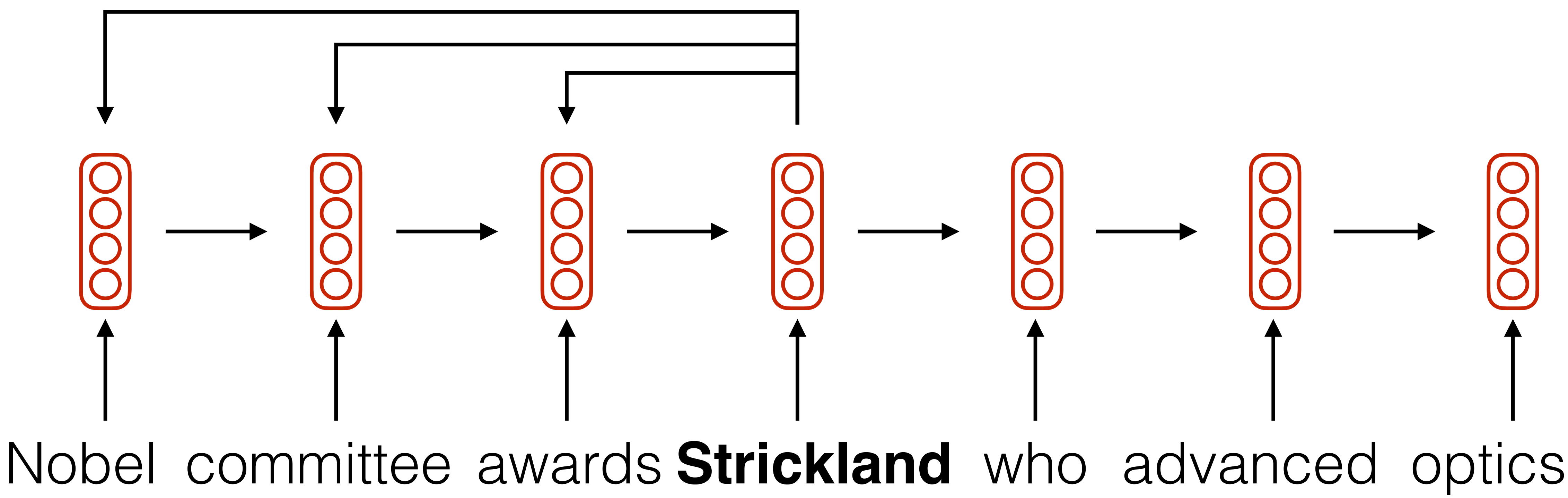


One word: $O(D^2)$

One word + Attention: $O(D^2+LD)$

D - embedding dimension
L - sequence length
Matrix Vector Multiplication is $O(D^2)$, Vector Vector Multiplication is $O(D)$

Review: Attention w/ RNN



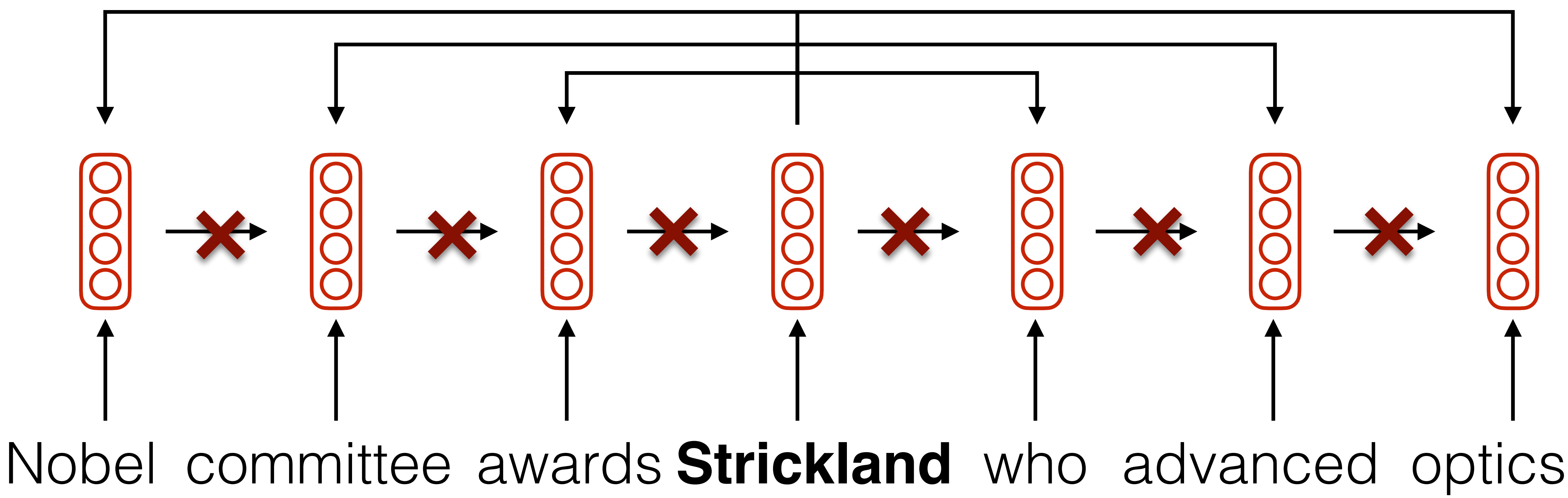
One word: $O(D^2)$

One word + Attention: $O(D^2+LD)$

Full sentence + Attention: $O(LD^2+L^2D)$

D - embedding dimension
L - sequence length
Matrix Vector Multiplication is $O(D^2)$, Vector Vector Multiplication is $O(D)$

Self-Attention ~~w/ RNN~~



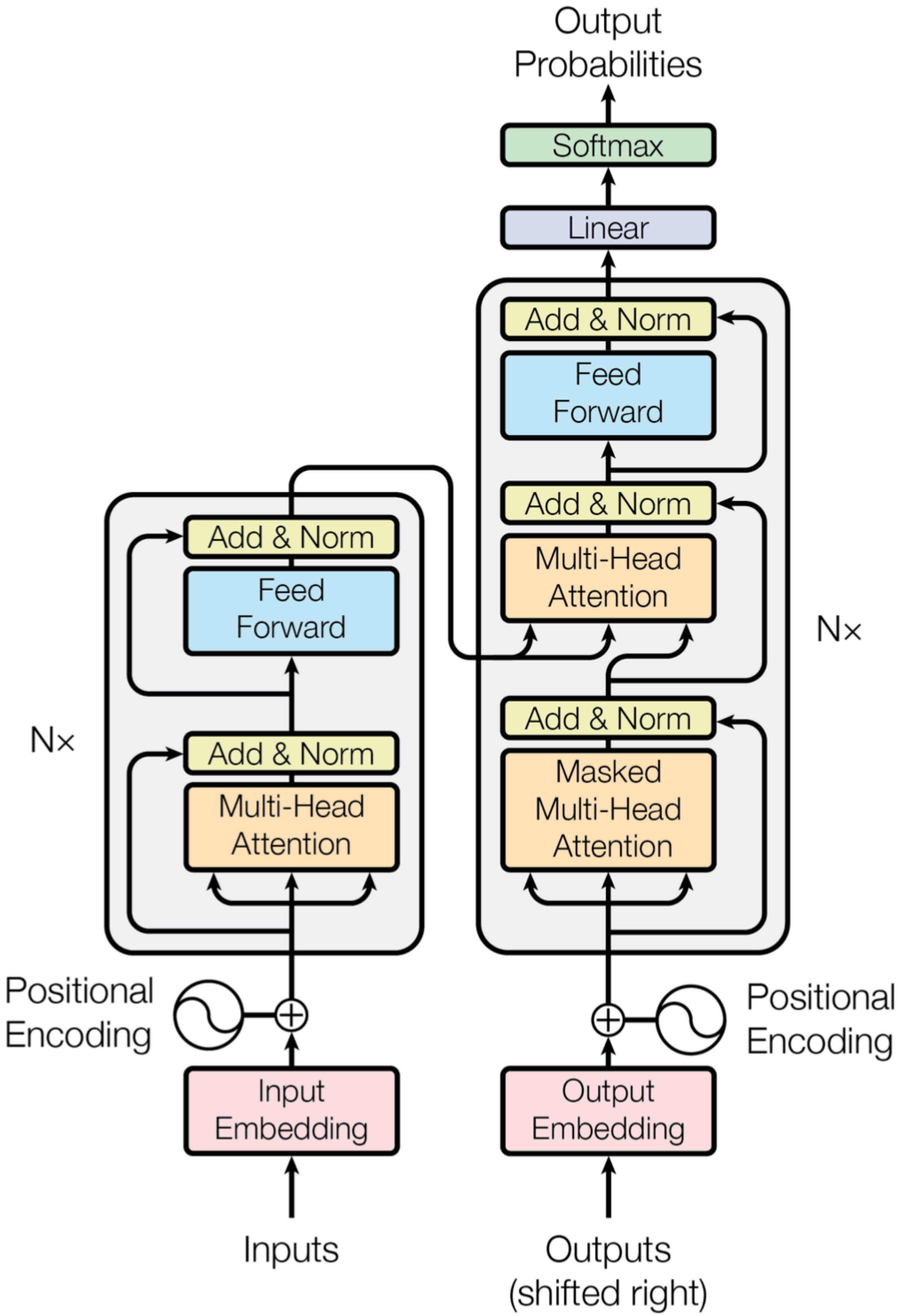
One word: $O(D^2)$

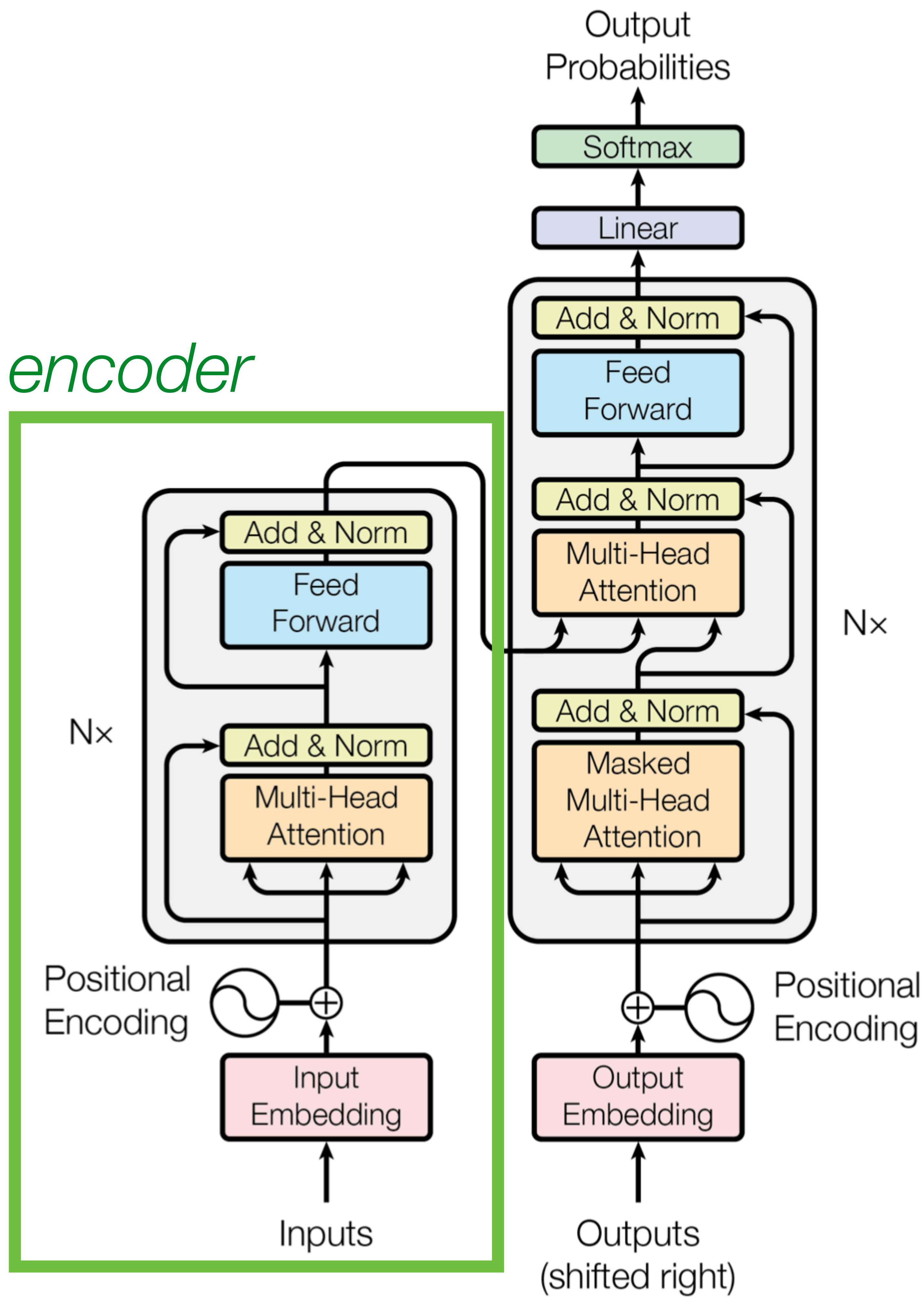
One word + Attention: $O(D^2+LD)$

Full sentence + Attention: $O(LD^2+L^2D)$

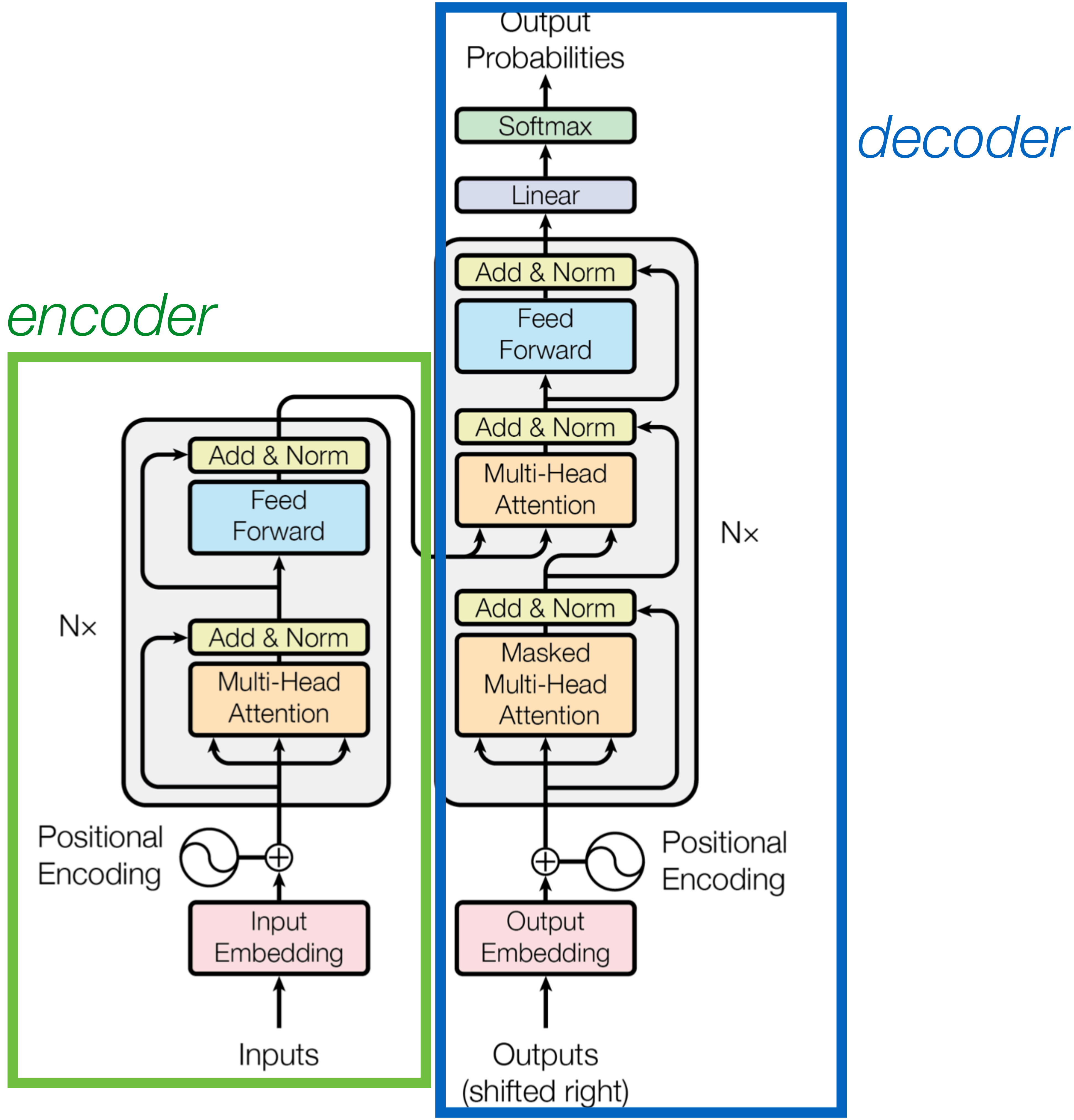
D - embedding dimension
L - sequence length
Matrix Vector Multiplication is $O(D^2)$, Vector Vector Multiplication is $O(D)$

How to turn self-attention into a viable model? Transformers!

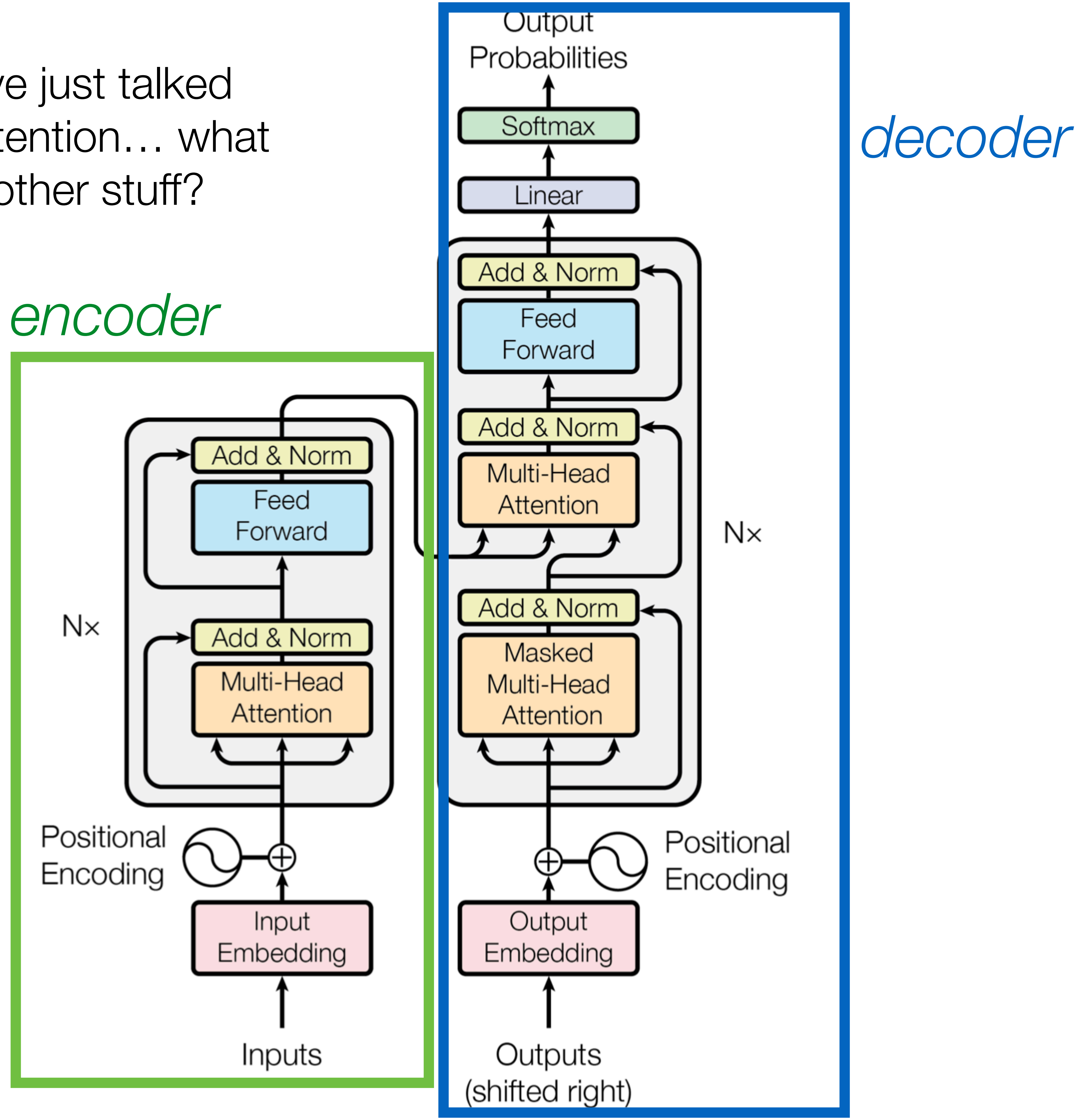

Output
Probabilities
Softmax
Linear
Add & Norm
Feed
Forward
Add & Norm
Multi-Head
Attention
Nx
Add & Norm
Masked
Multi-Head
Attention
Add & Norm
Feed
Forward
Nx
Add & Norm
Multi-Head
Attention
Positional
Encoding
Positional
Encoding
Input
Embedding
Output
Embedding
Inputs
Outputs
(shifted right)



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


encoder
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

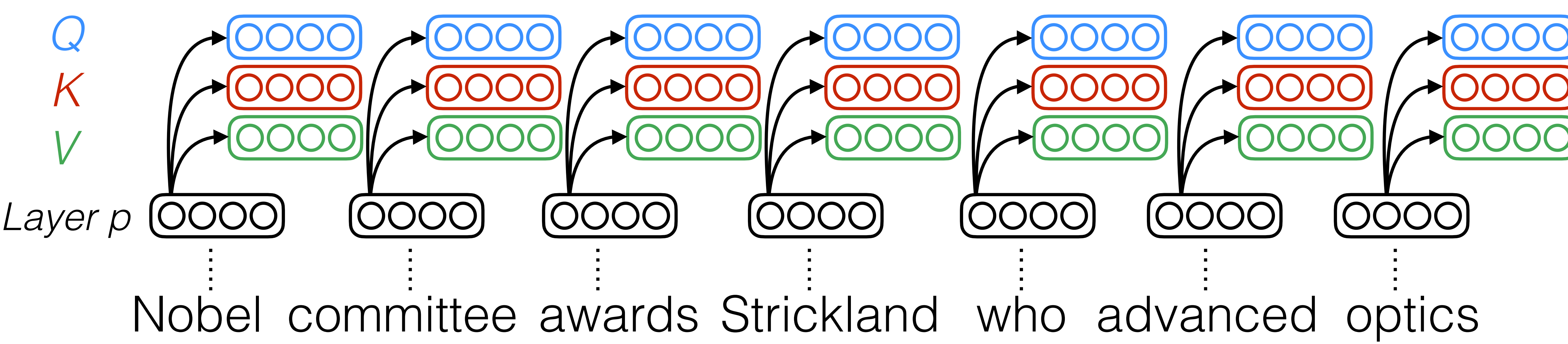
So far we've just talked about self-attention… what is all this other stuff?



Slides by Emma Strubell!

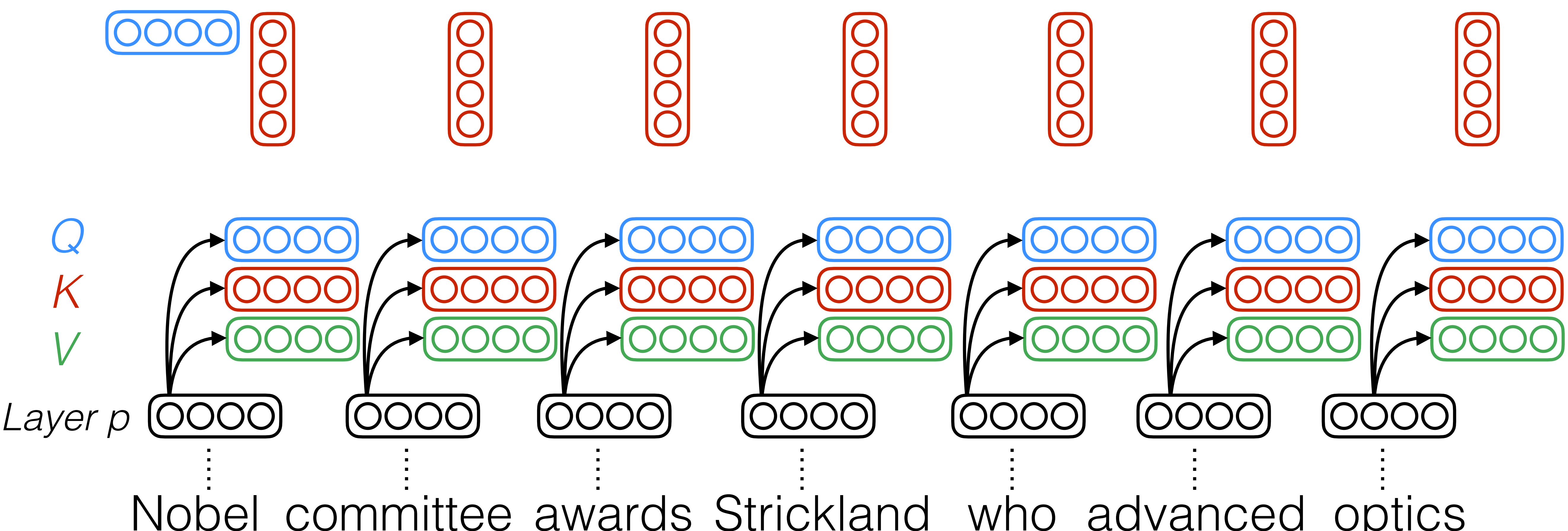
[Vaswani et al. 2017]

Self-attention (in encoder)



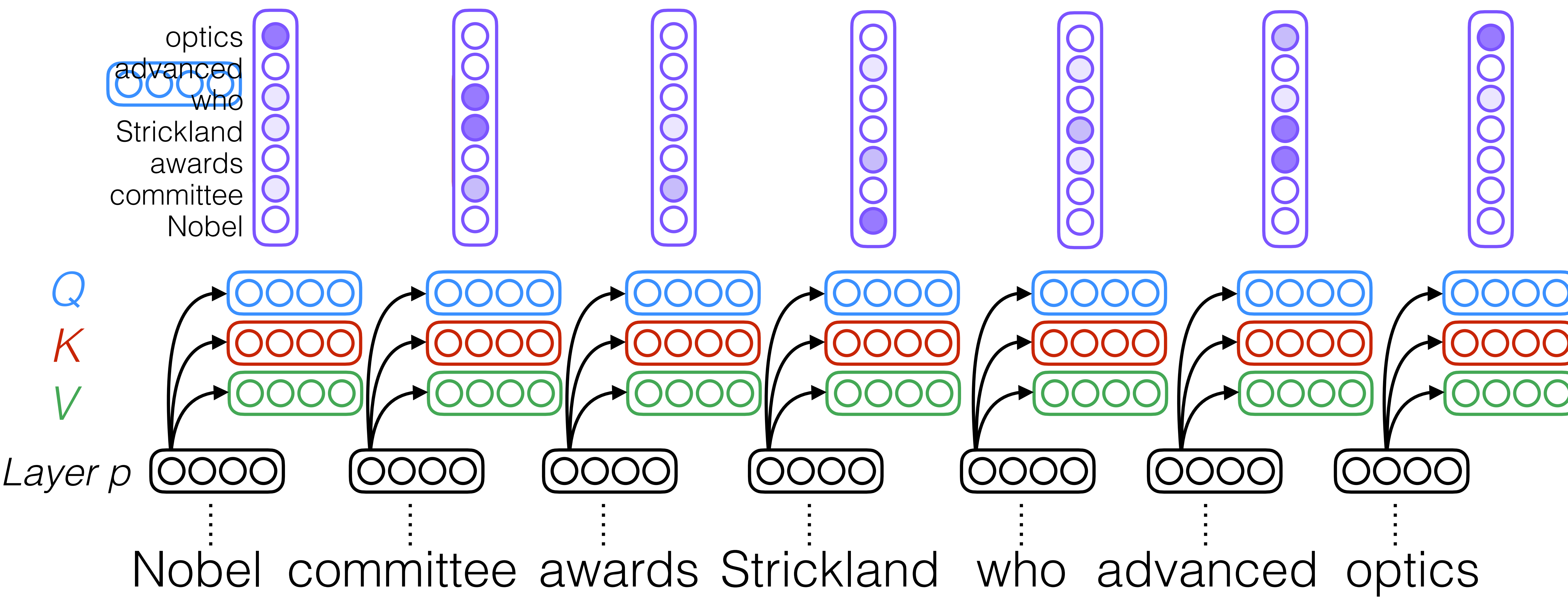
[Vaswani et al. 2017]

Self-attention (in encoder)



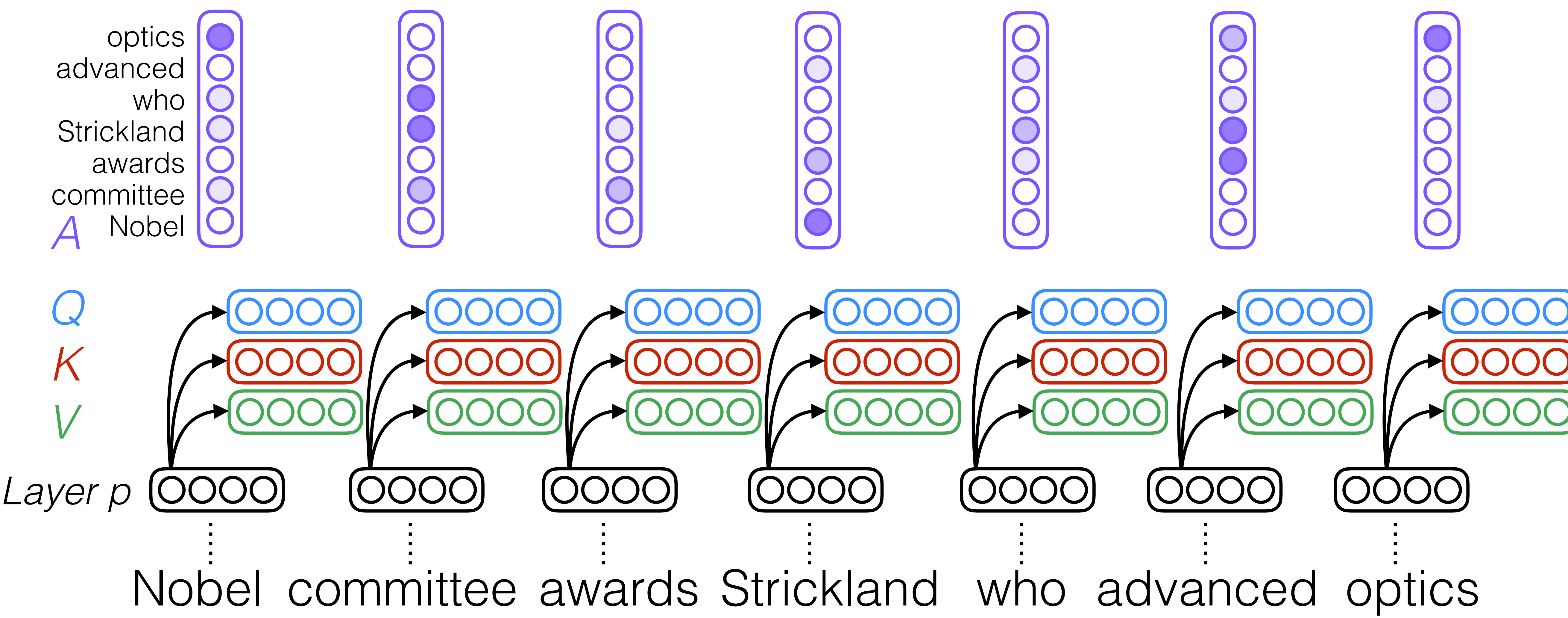
[Vaswani et al. 2017]

Self-attention (in encoder)

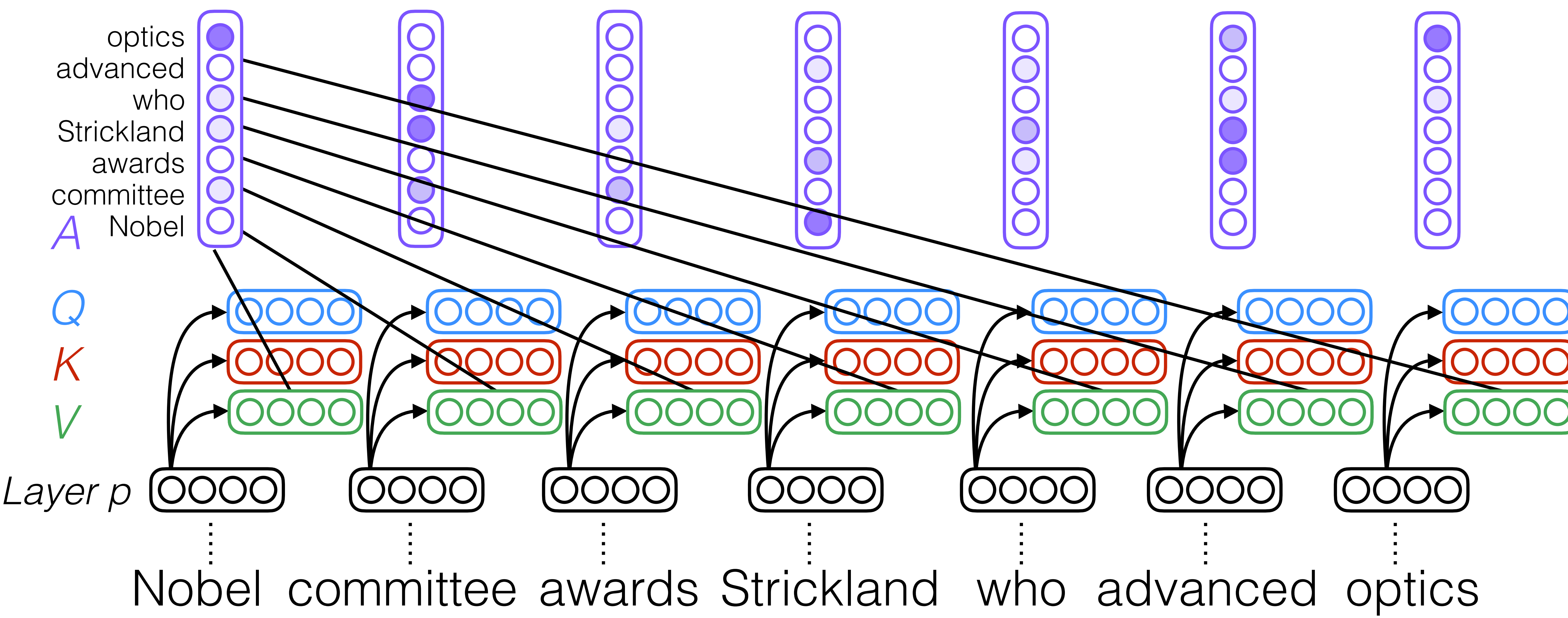


[Vaswani et al. 2017]

Self-attention (in encoder)

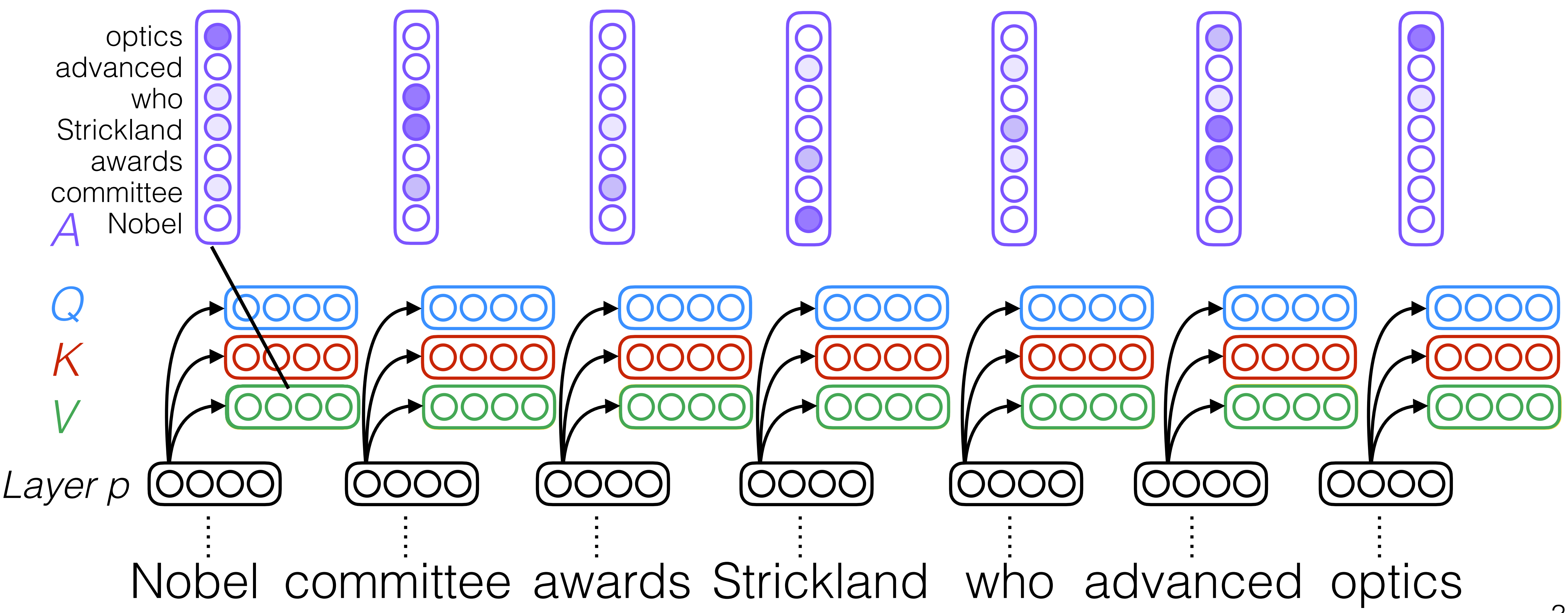


Self-attention (in encoder)



[Vaswani et al. 2017]

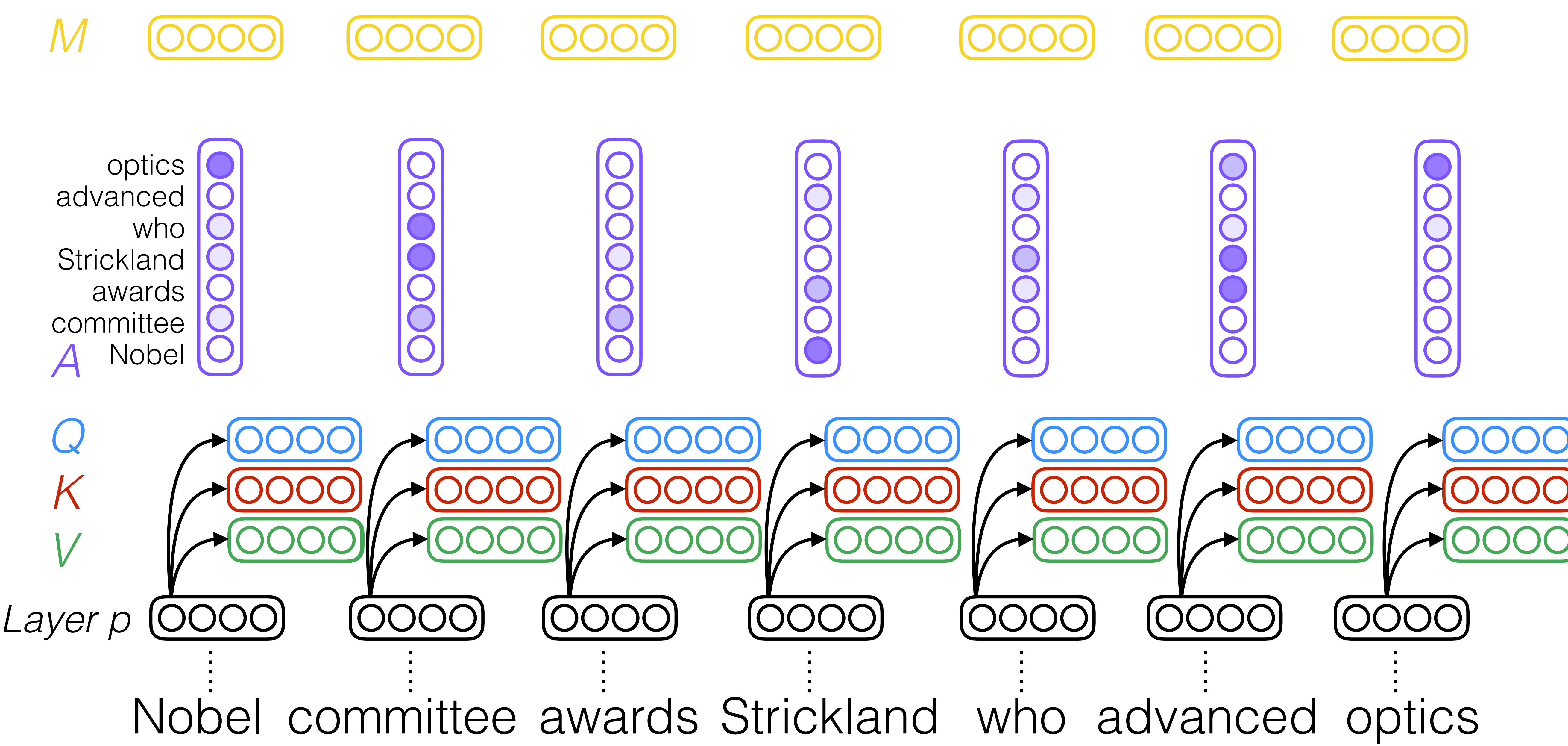
Self-attention (in encoder)



Slides by Emma Strubell!

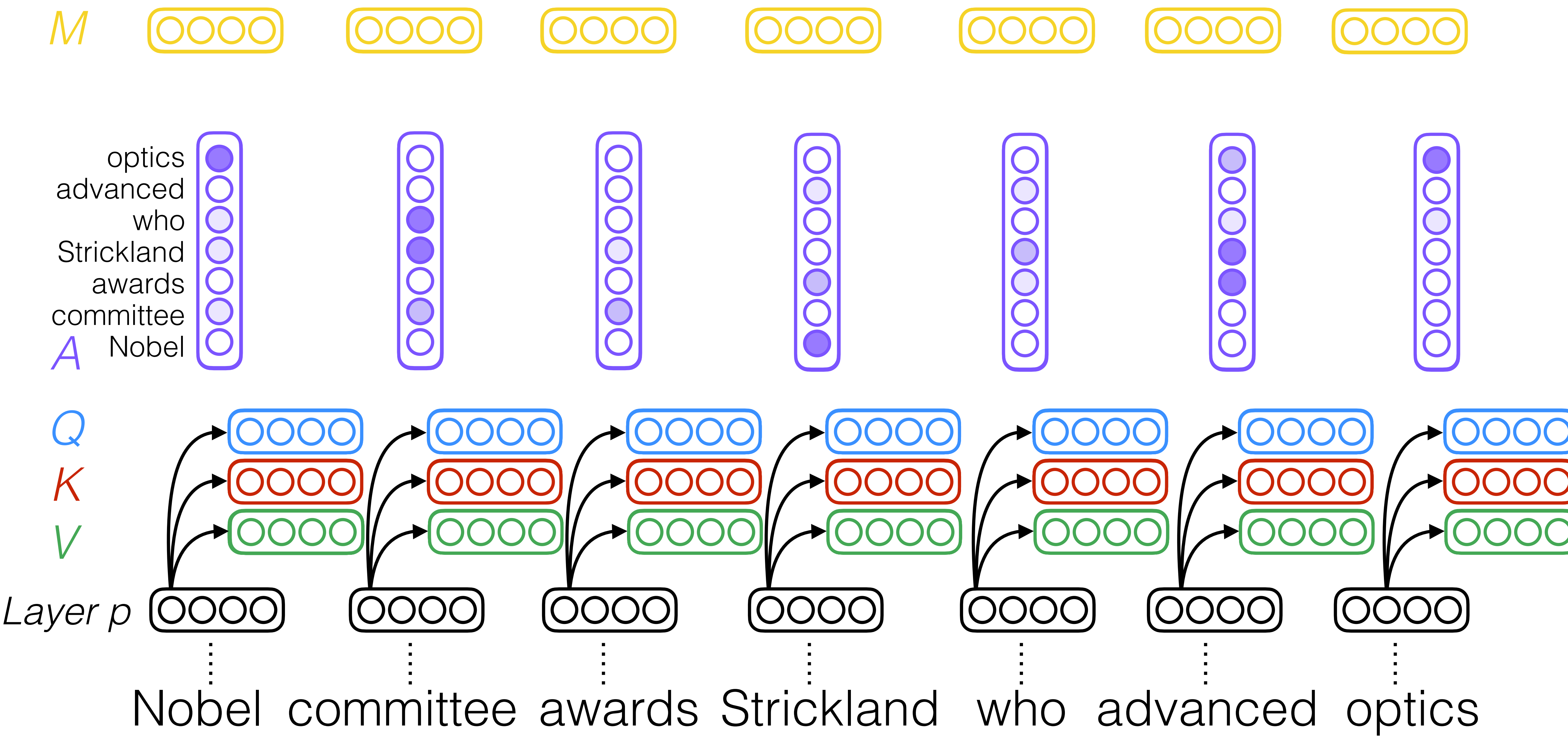
[Vaswani et al. 2017]

Self-attention (in encoder)



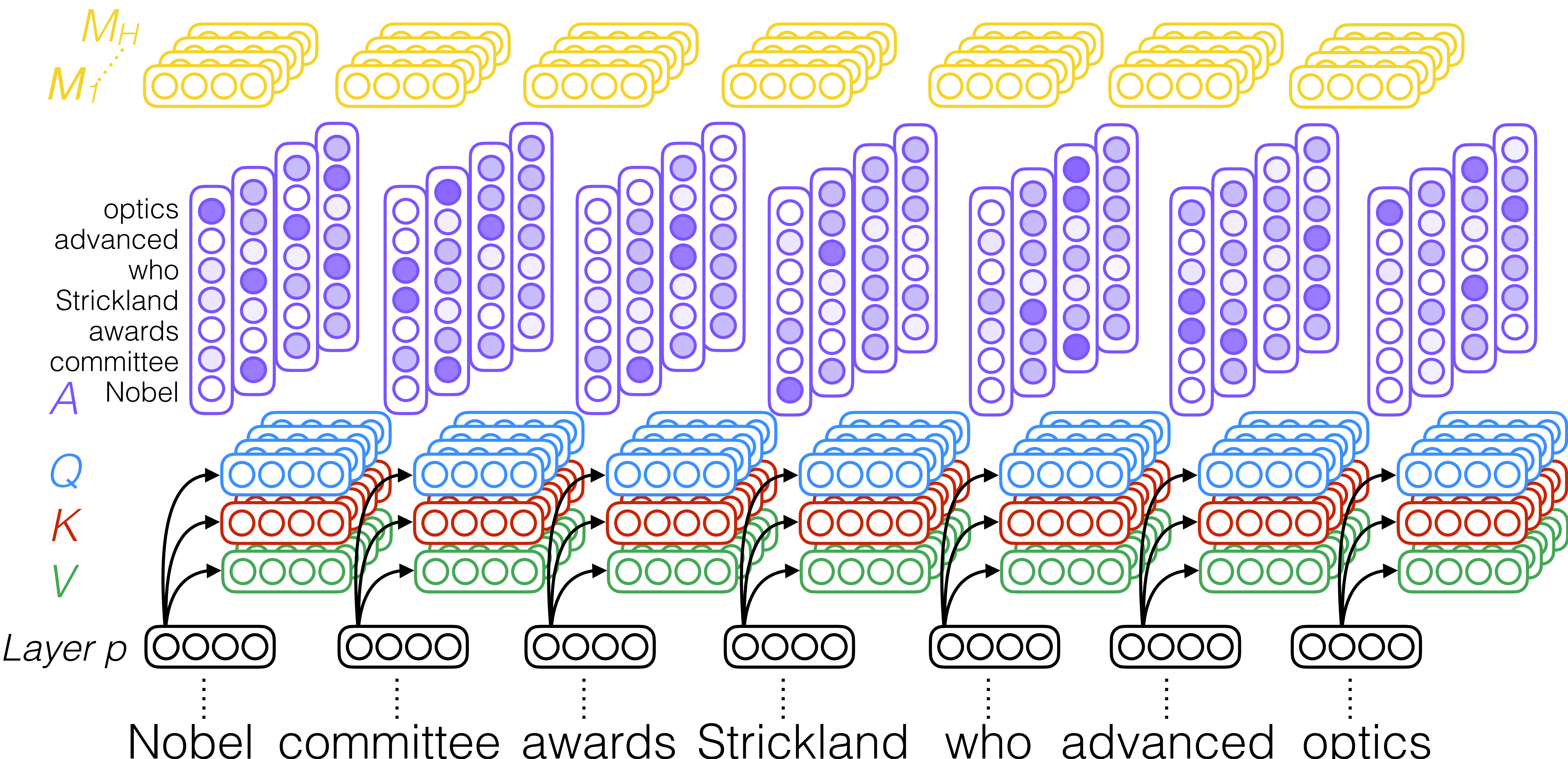
[Vaswani et al. 2017]

Self-attention (in encoder)



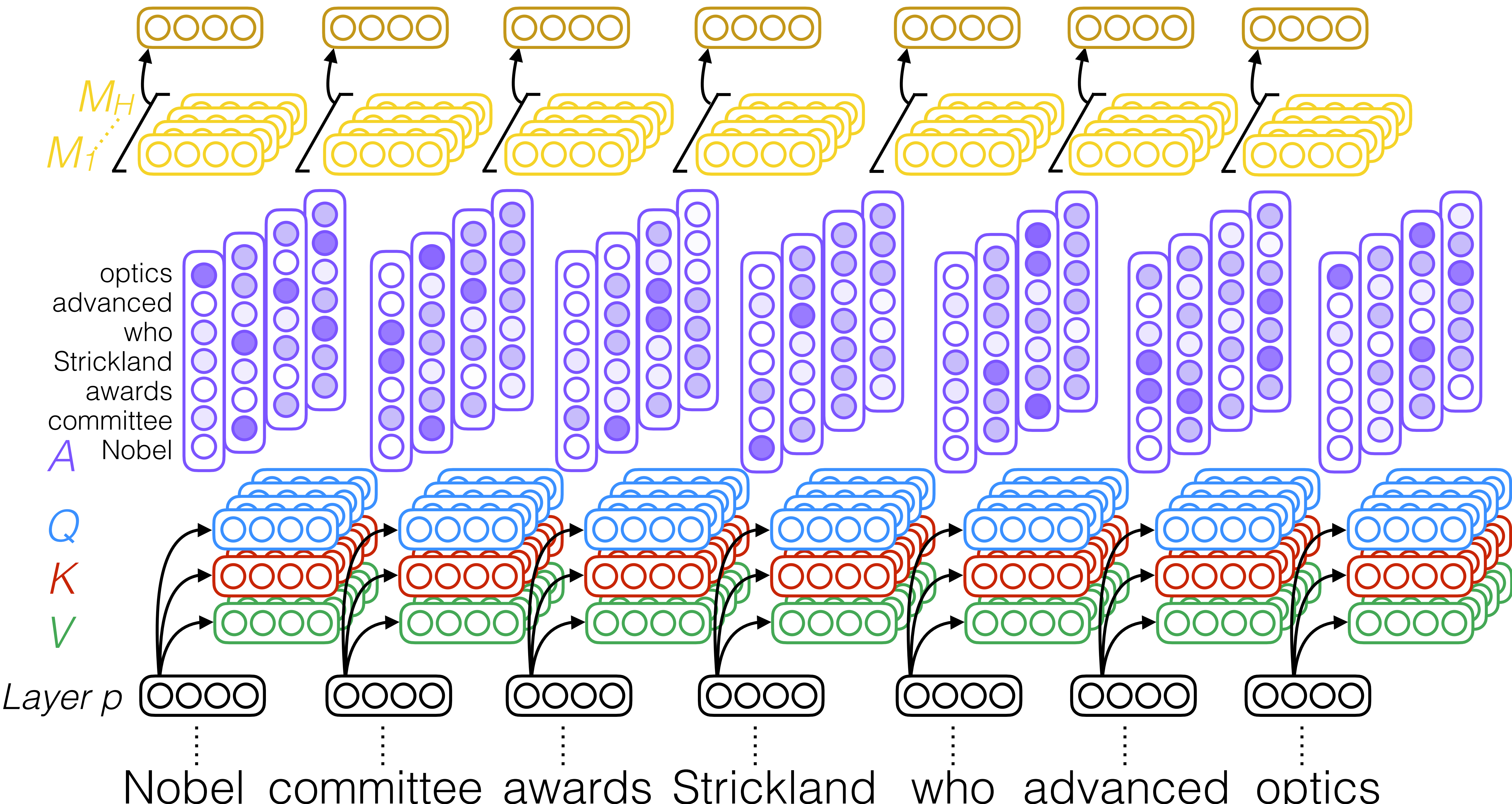
[Vaswani et al. 2017]

Multi-head self-attention



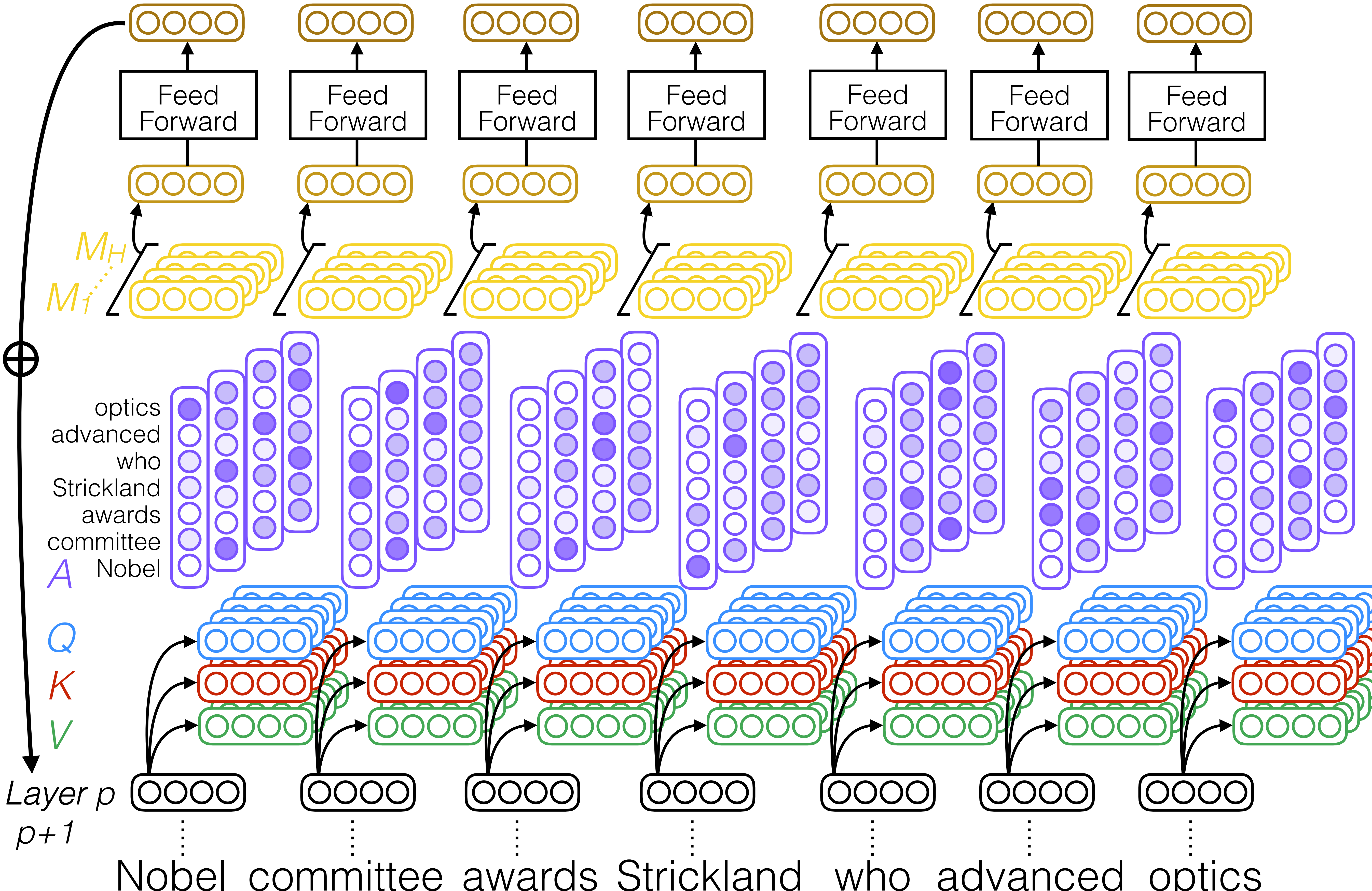
[Vaswani et al. 2017]

Multi-head self-attention



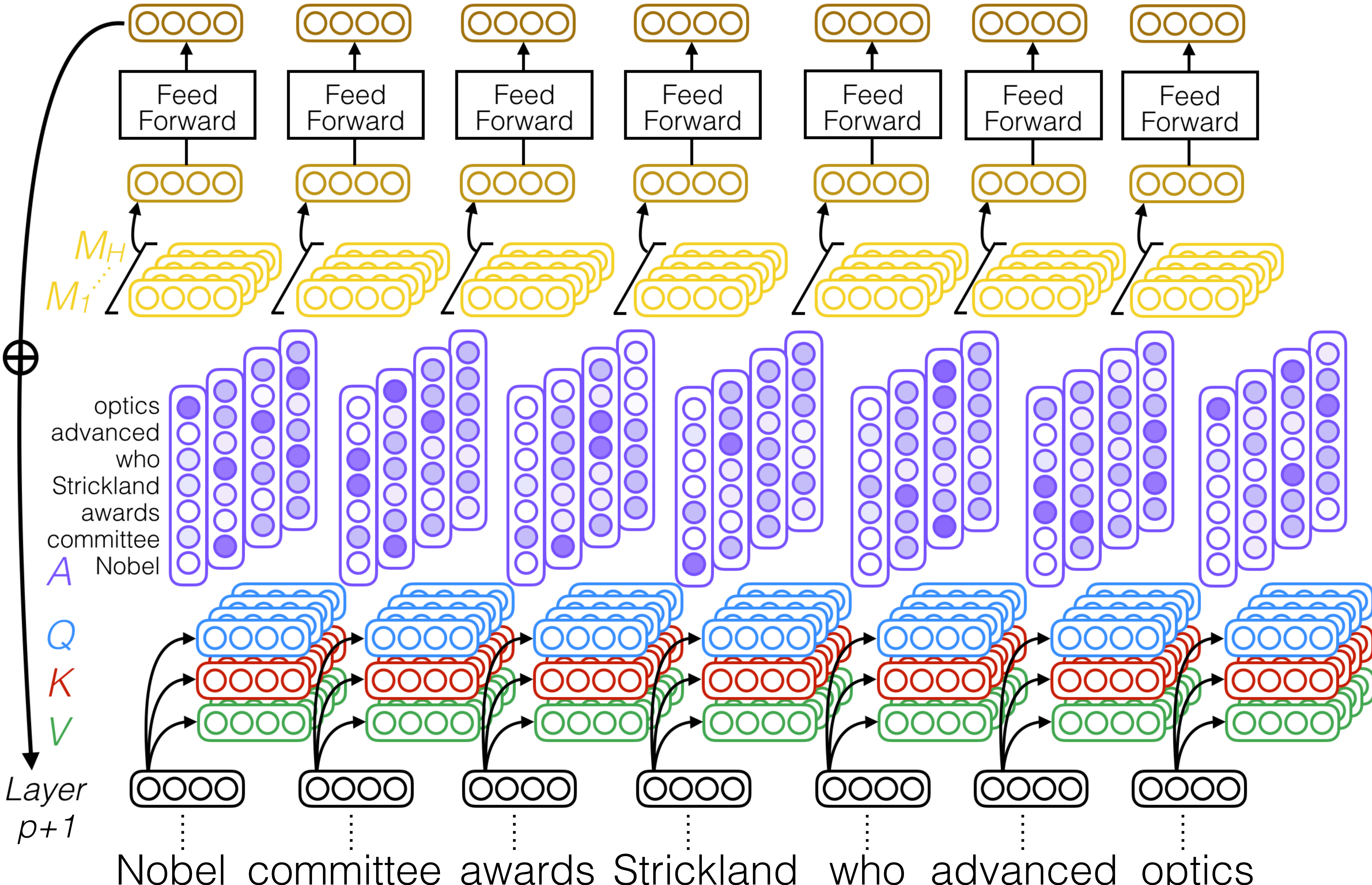
[Vaswani et al. 2017]

Multi-head self-attention



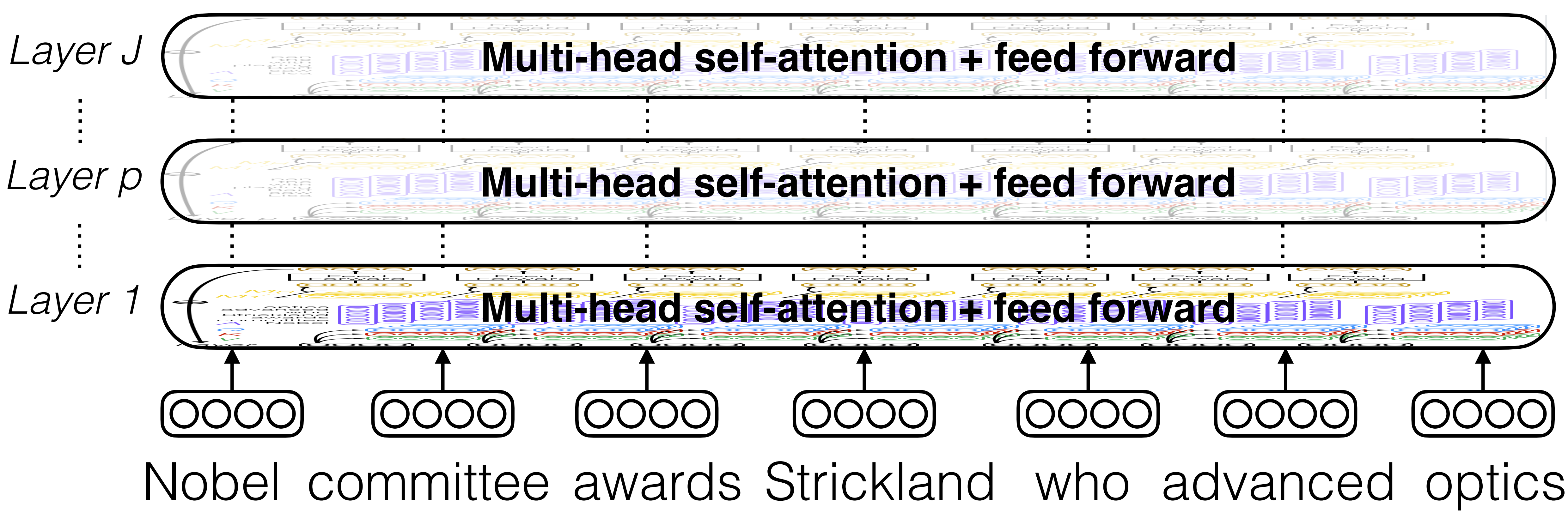
[Vaswani et al. 2017]

Multi-head self-attention

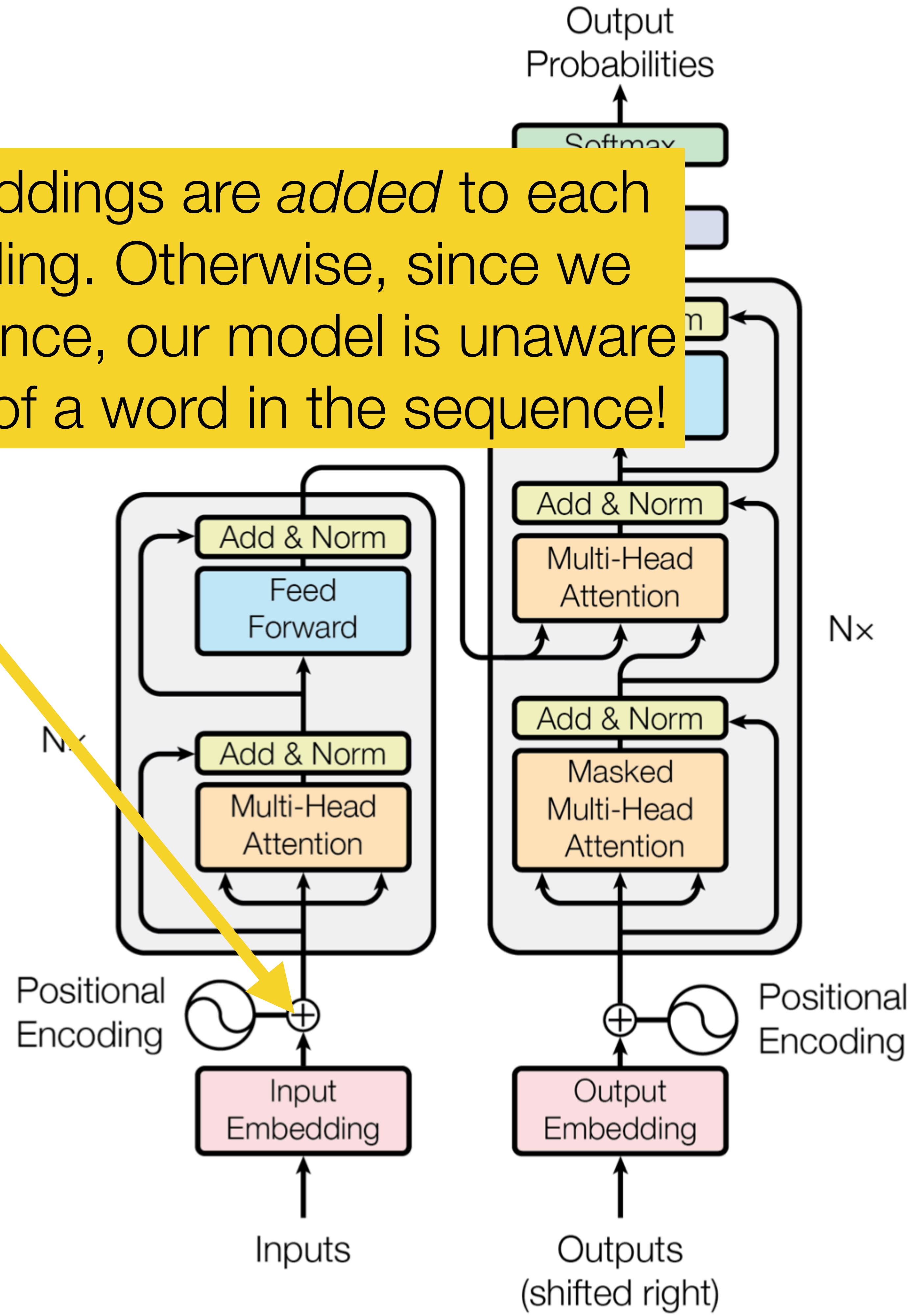


[Vaswani et al. 2017]

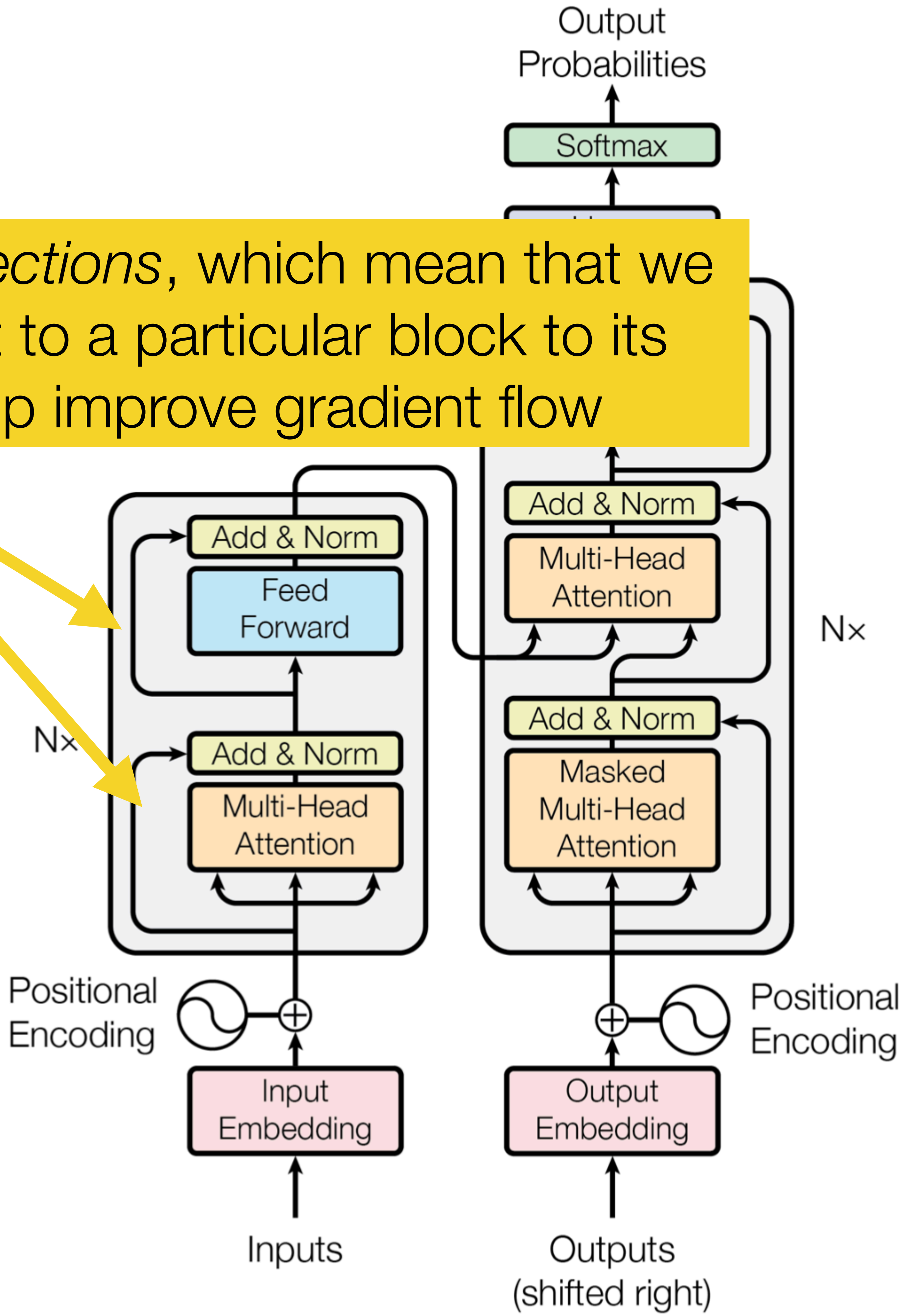
Multi-head self-attention



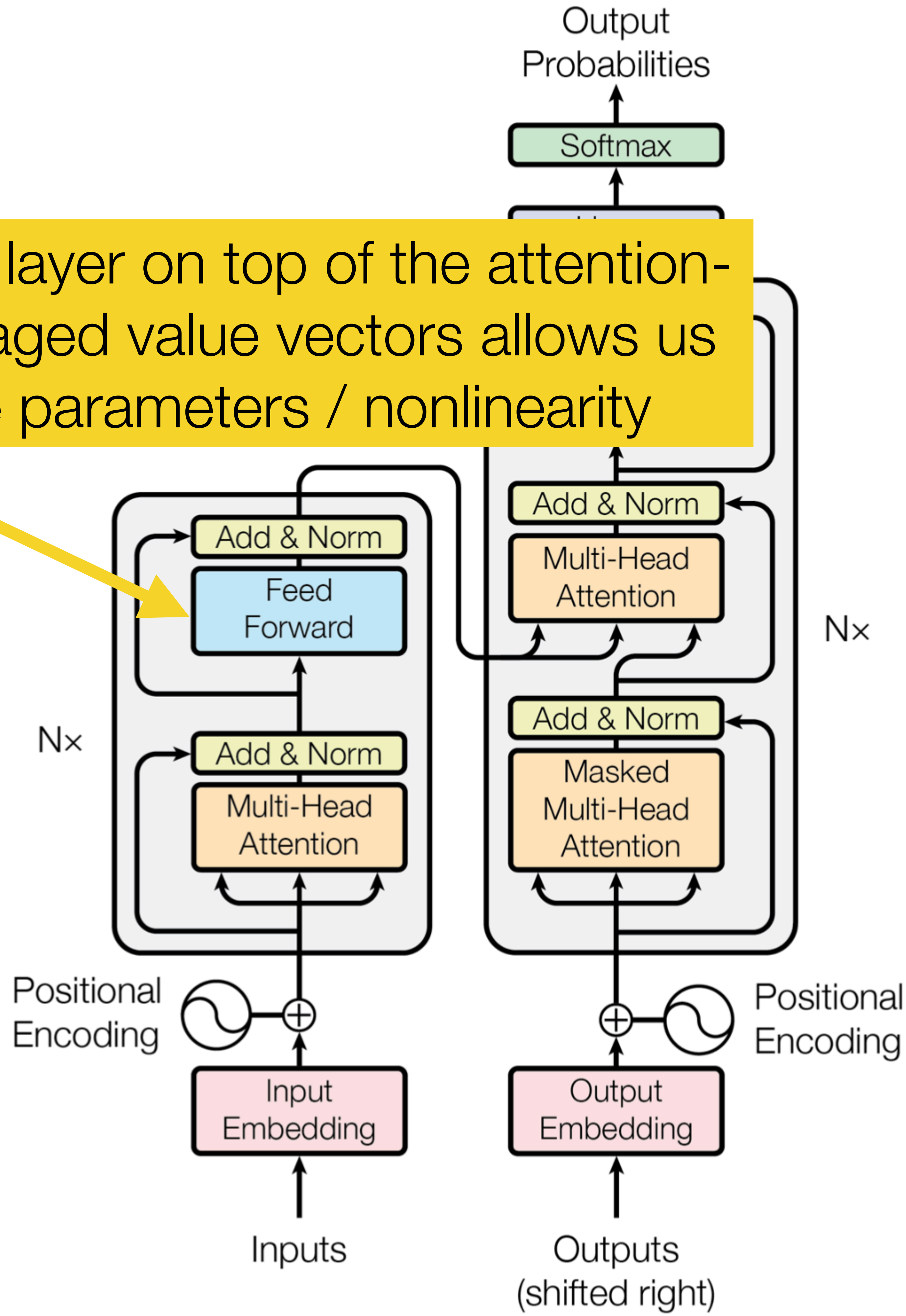
Position embeddings are *added* to each word embedding. Otherwise, since we have no recurrence, our model is unaware of the position of a word in the sequence!



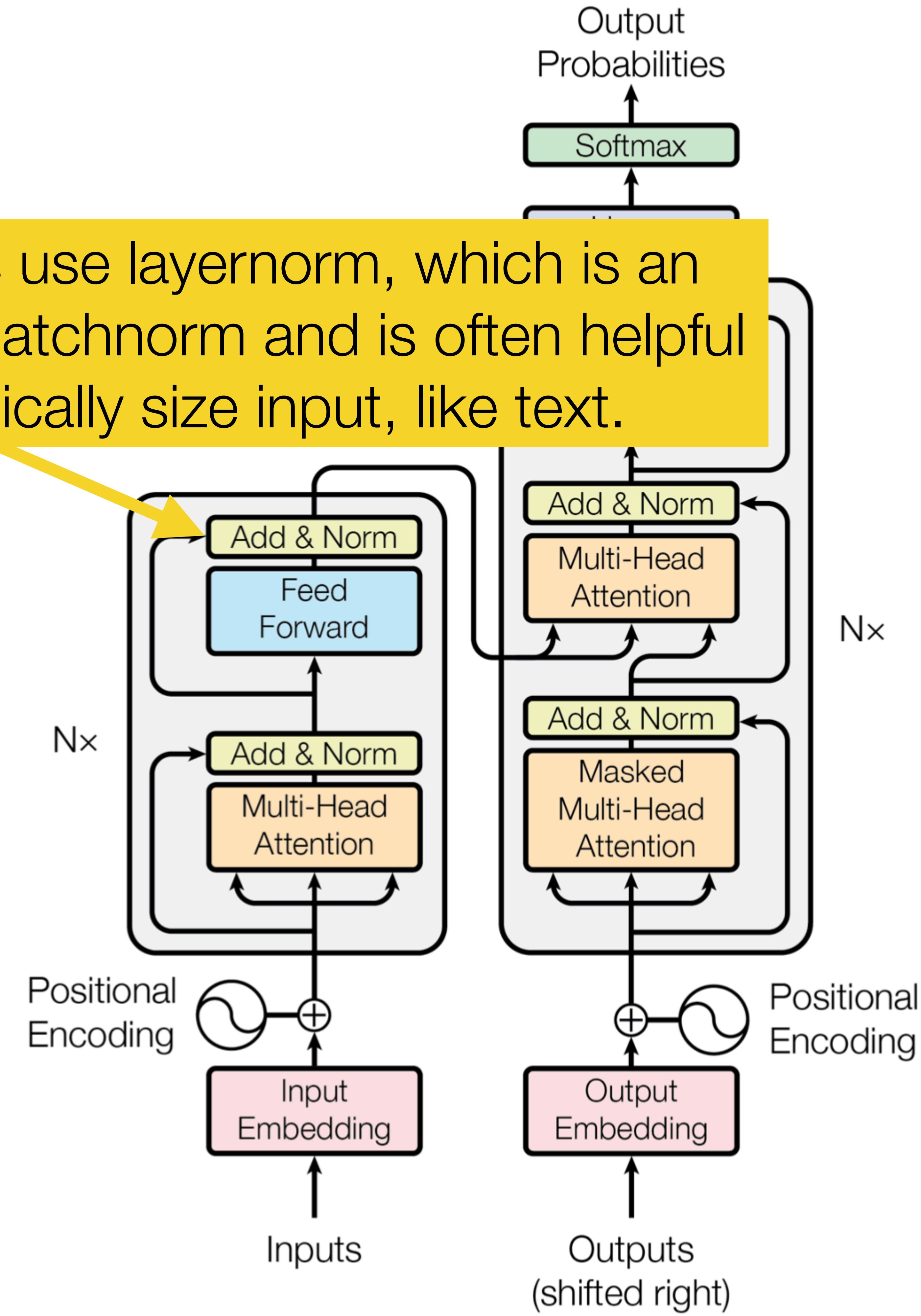
Residual connections, which mean that we add the input to a particular block to its output, help improve gradient flow



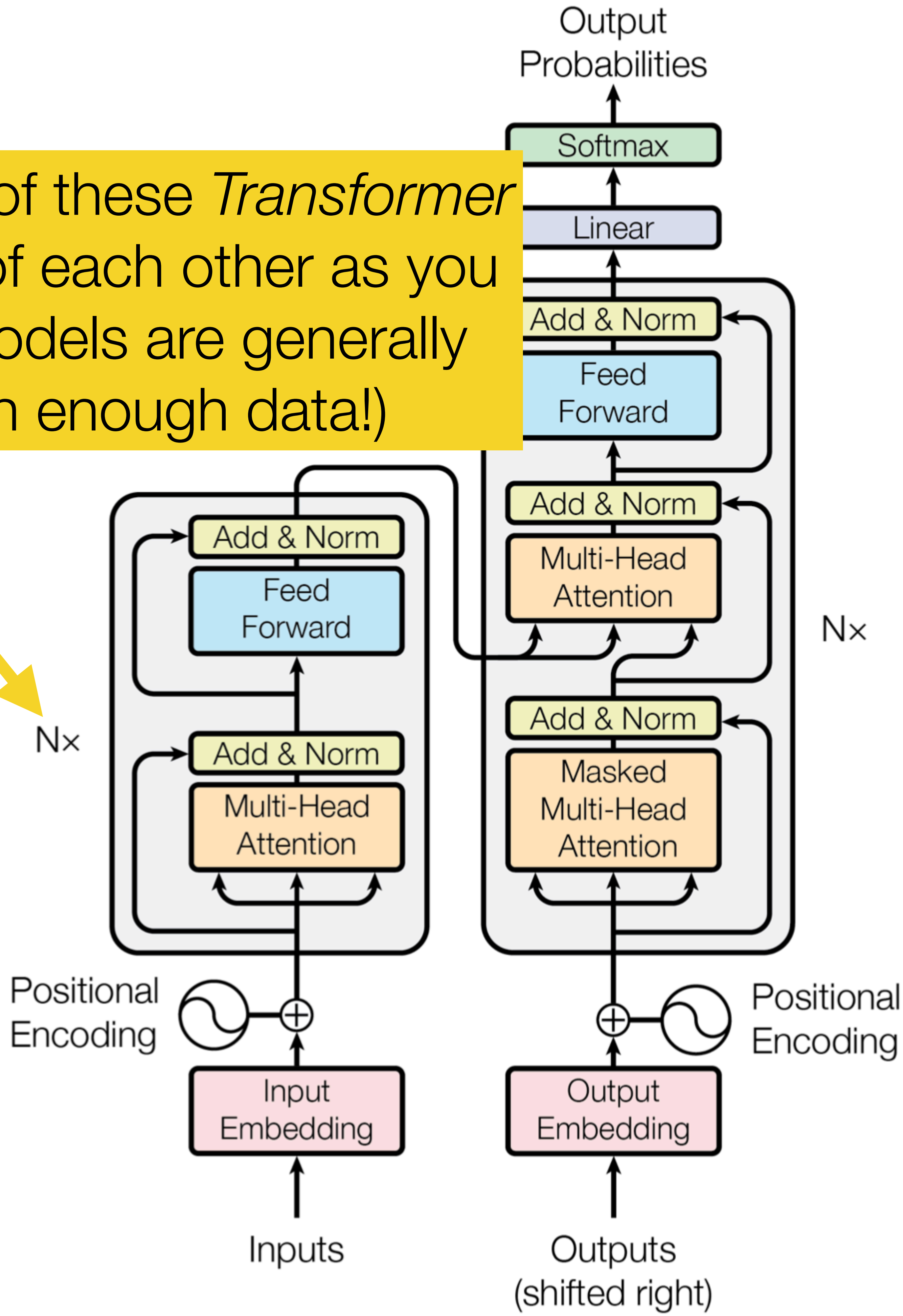
A feed-forward layer on top of the attention-weighted averaged value vectors allows us to add more parameters / nonlinearity



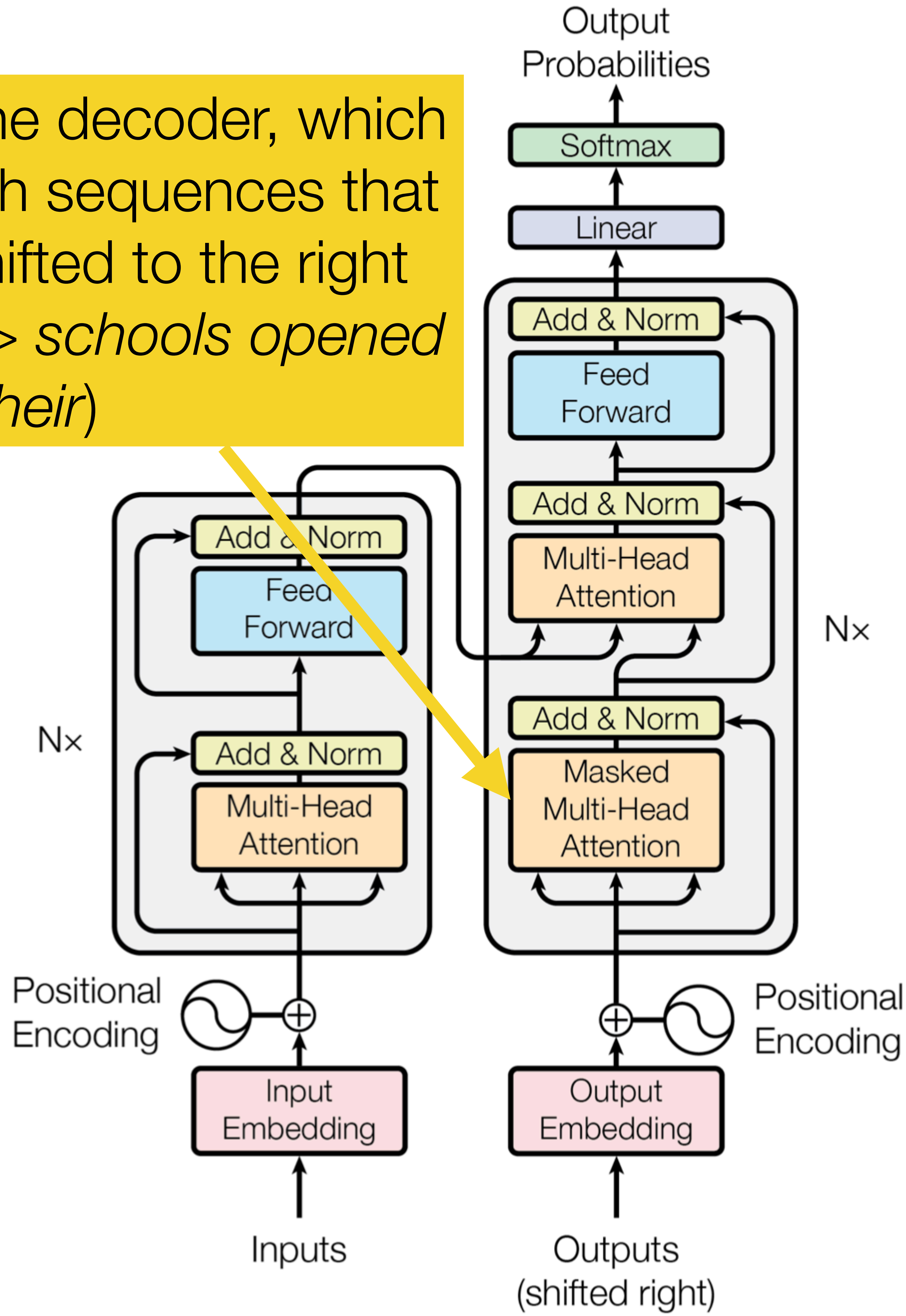
Transformers use layernorm, which is an alternative to batchnorm and is often helpful for dynamically size input, like text.



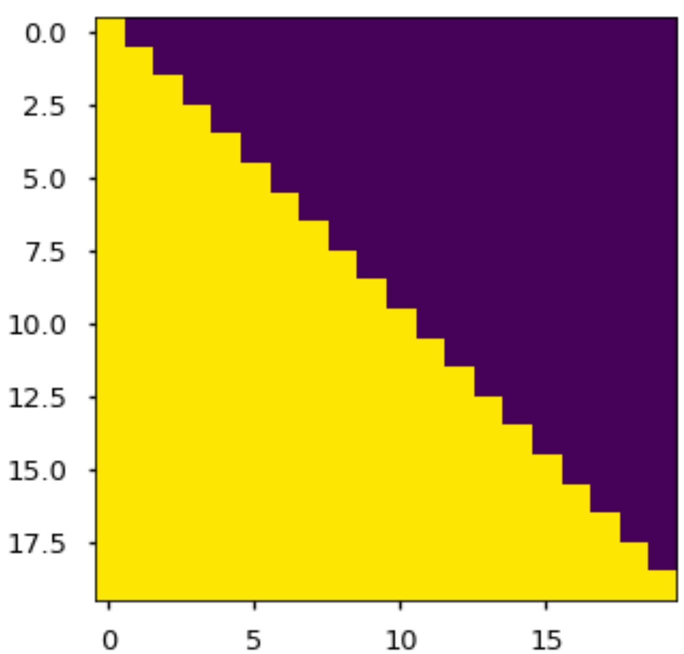
Stack as many of these *Transformer* blocks on top of each other as you can (bigger models are generally better given enough data!)

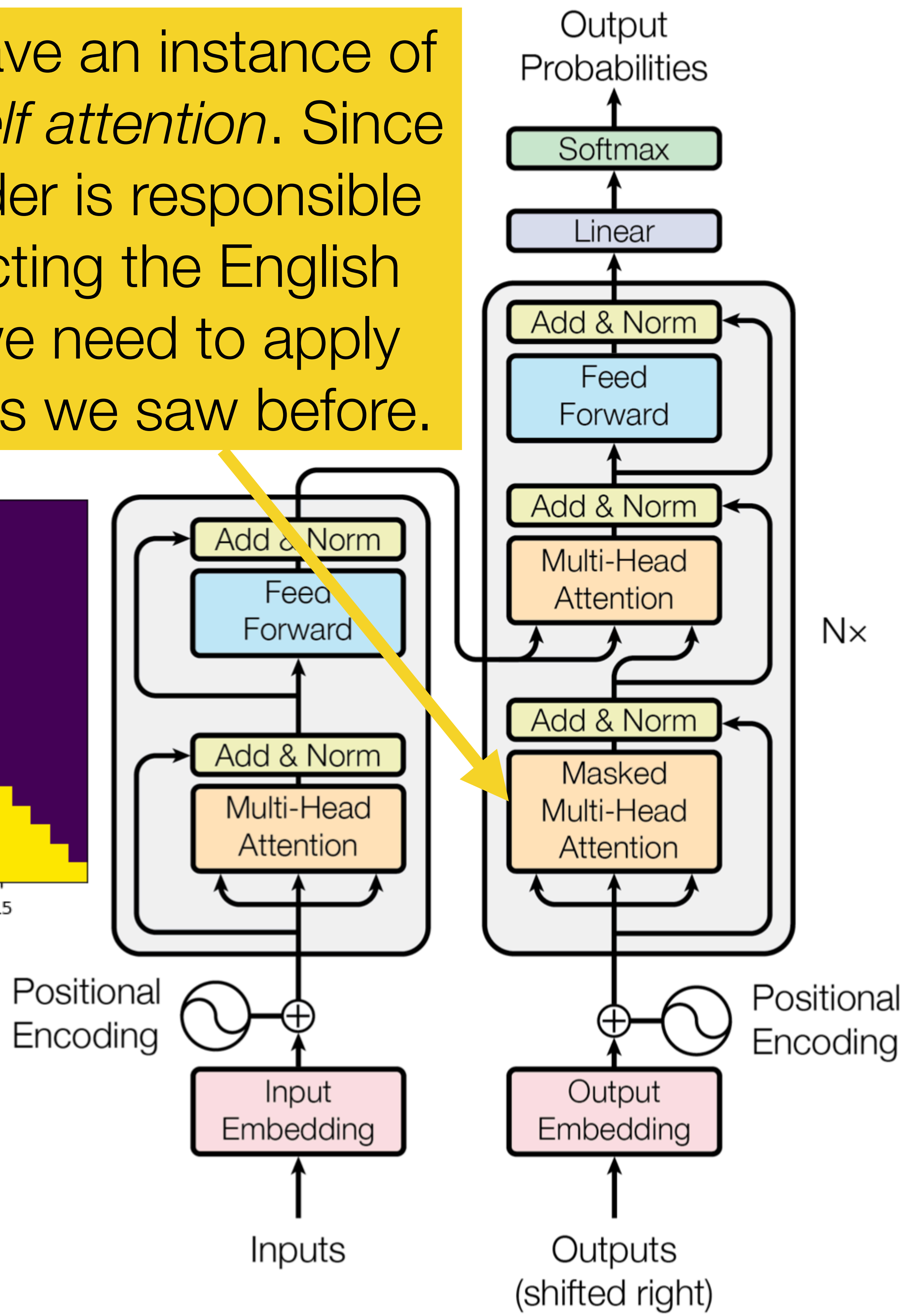


Moving onto the decoder, which takes in English sequences that have been shifted to the right (e.g., *<START> schools opened their*)

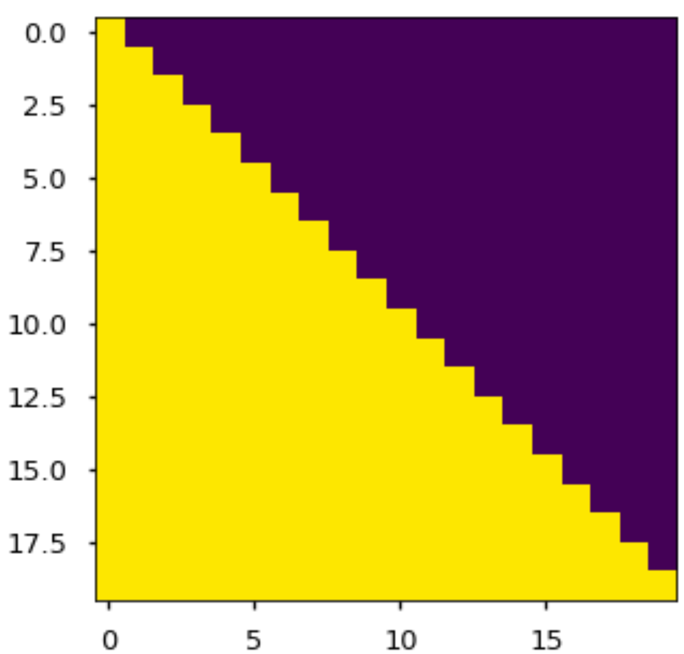


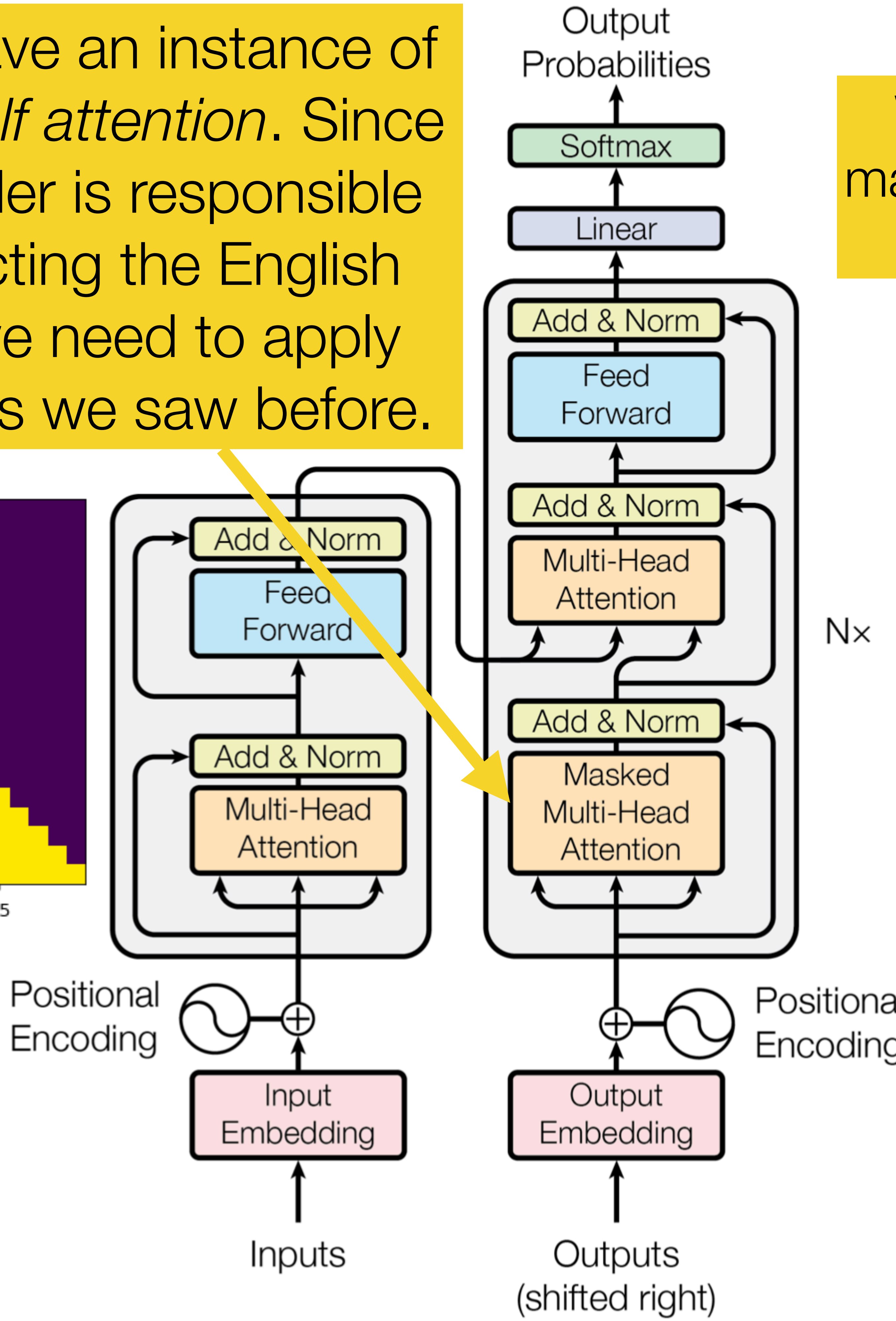
We first have an instance of *masked self attention*. Since the decoder is responsible for predicting the English words, we need to apply masking as we saw before.





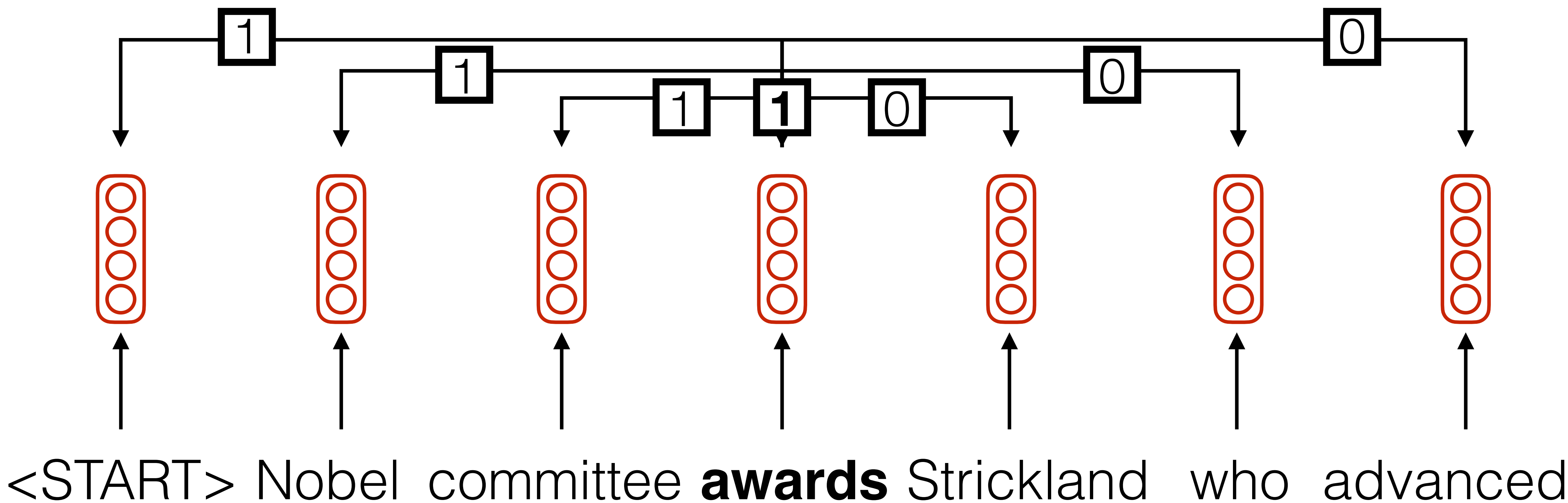
We first have an instance of *masked self attention*. Since the decoder is responsible for predicting the English words, we need to apply masking as we saw before.





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

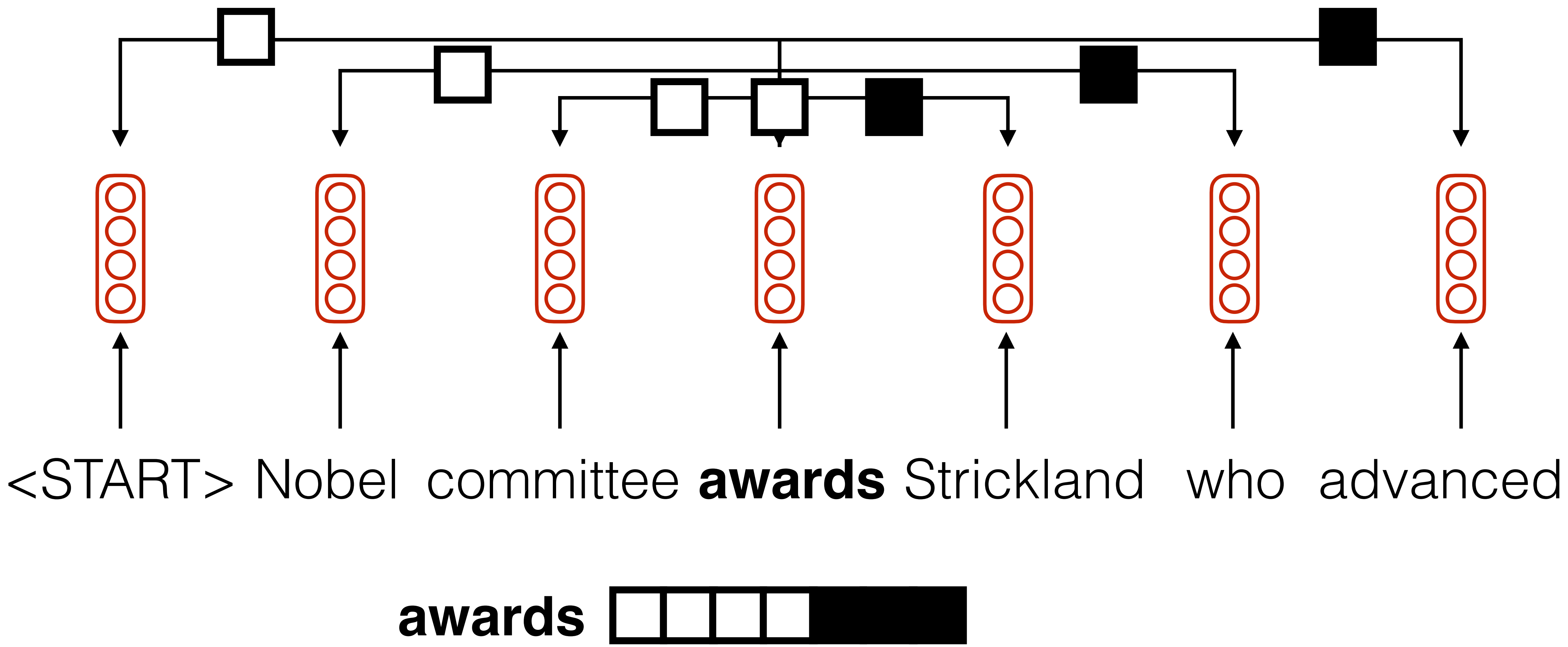
Masked Self-Attention



awards | 1 | 1 | 1 | **1** | 0 | 0 | 0 |

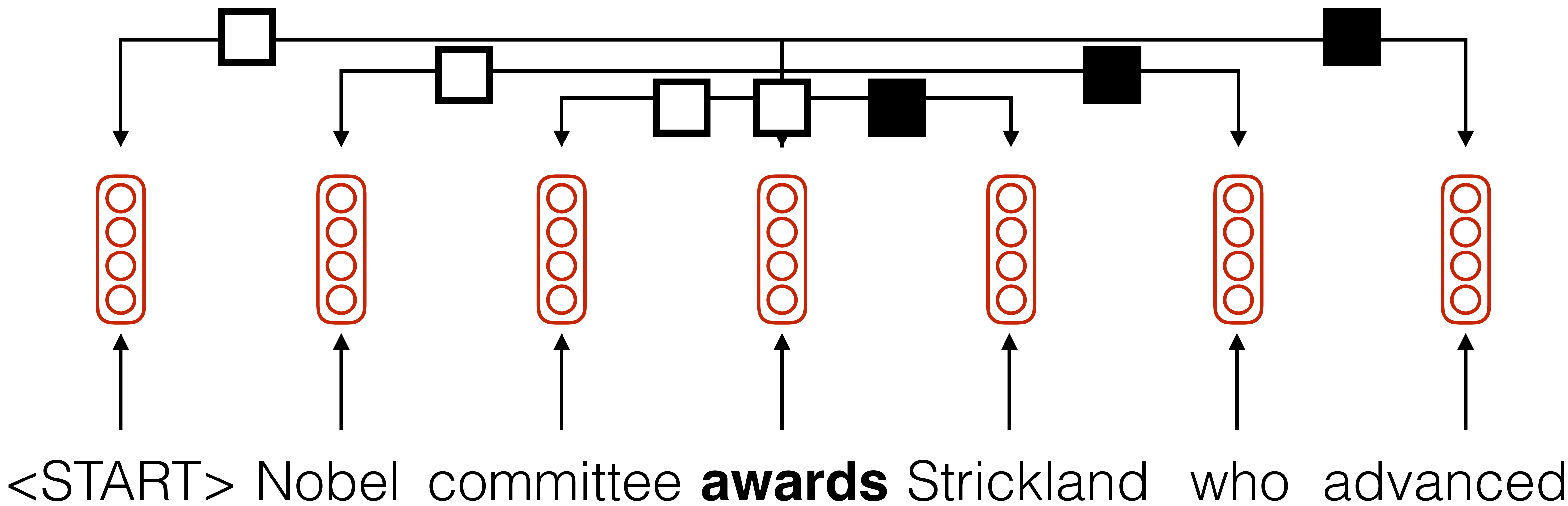
$$h_i^p = \sum_j \alpha_{i,j}^p \, h_j^{p-1}$$

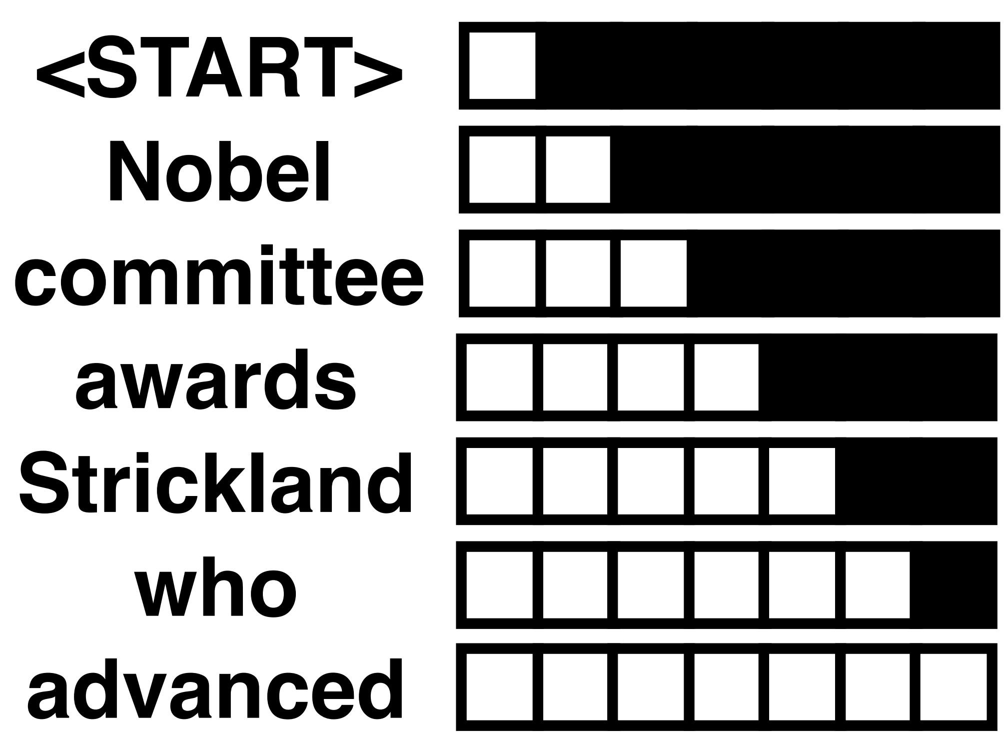
Masked Self-Attention

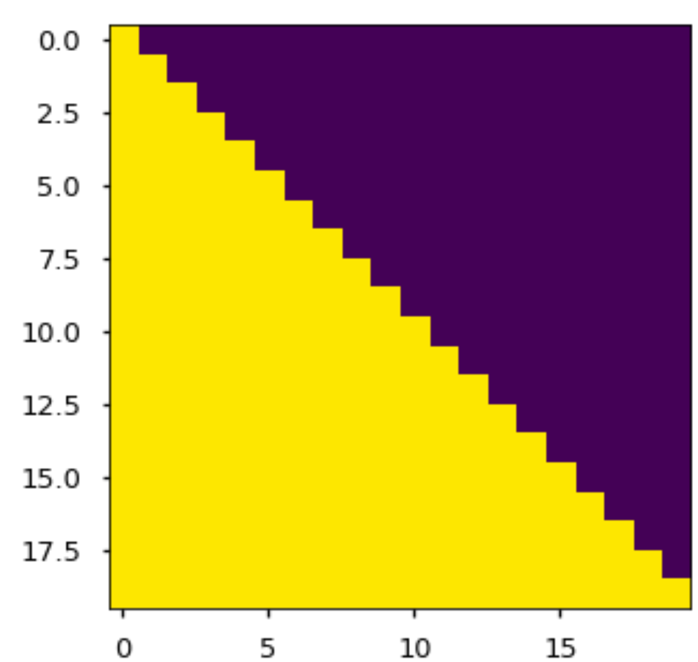


$$h_i^p = \sum_j \alpha_{i,j}^p \, h_j^{p-1}$$

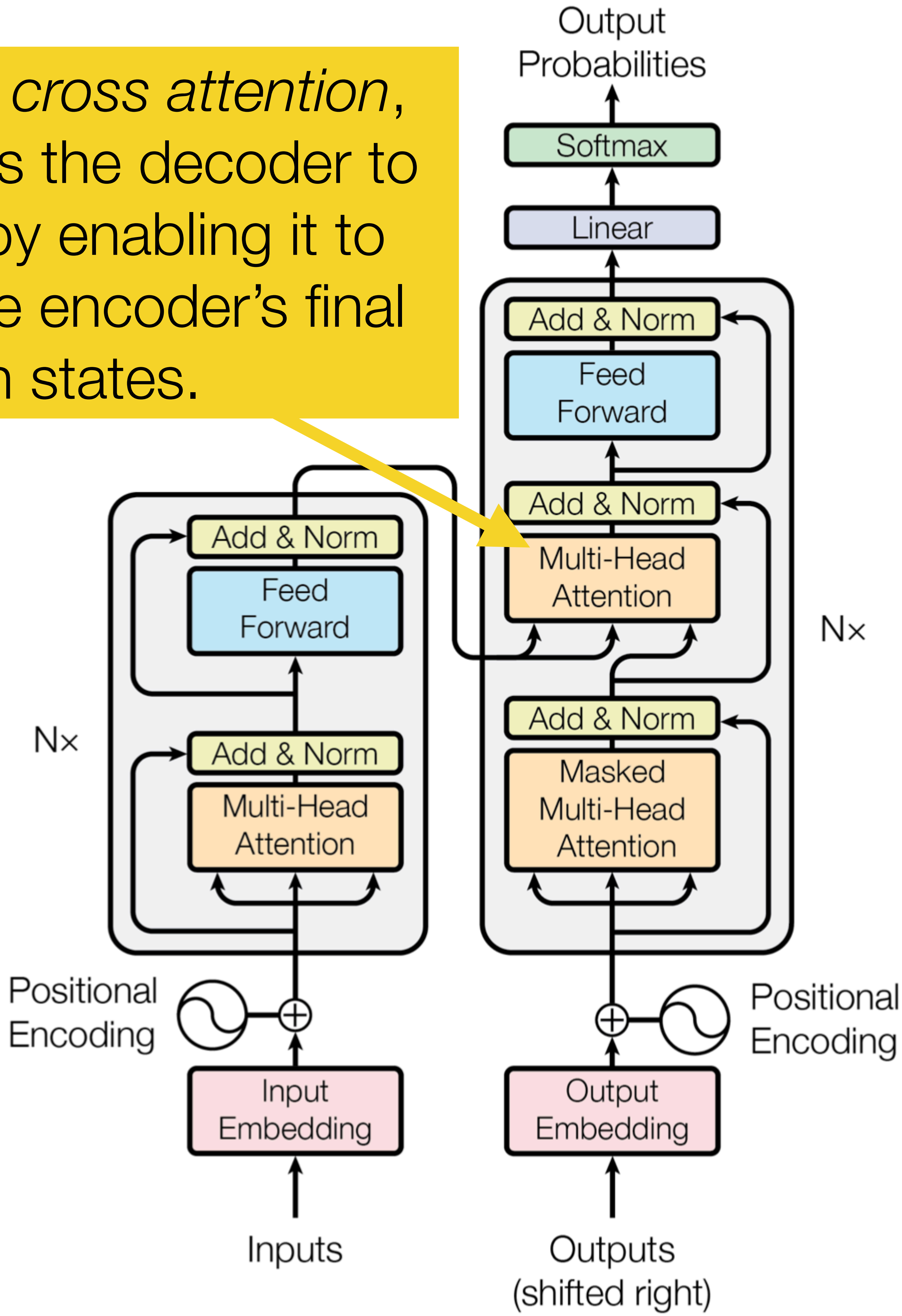
Masked Self-Attention



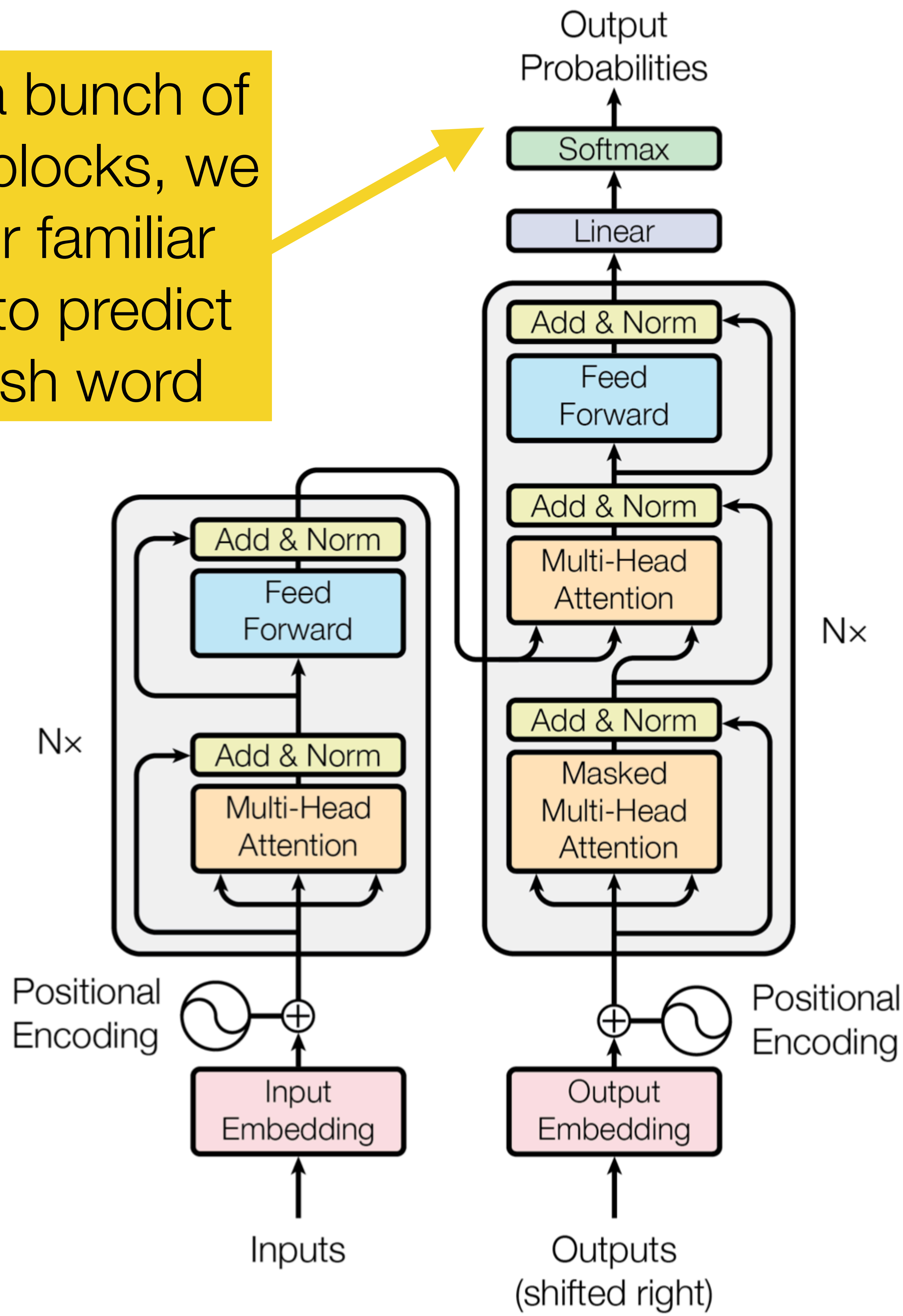




Now, we have *cross attention*, which connects the decoder to the encoder by enabling it to attend over the encoder's final hidden states.



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

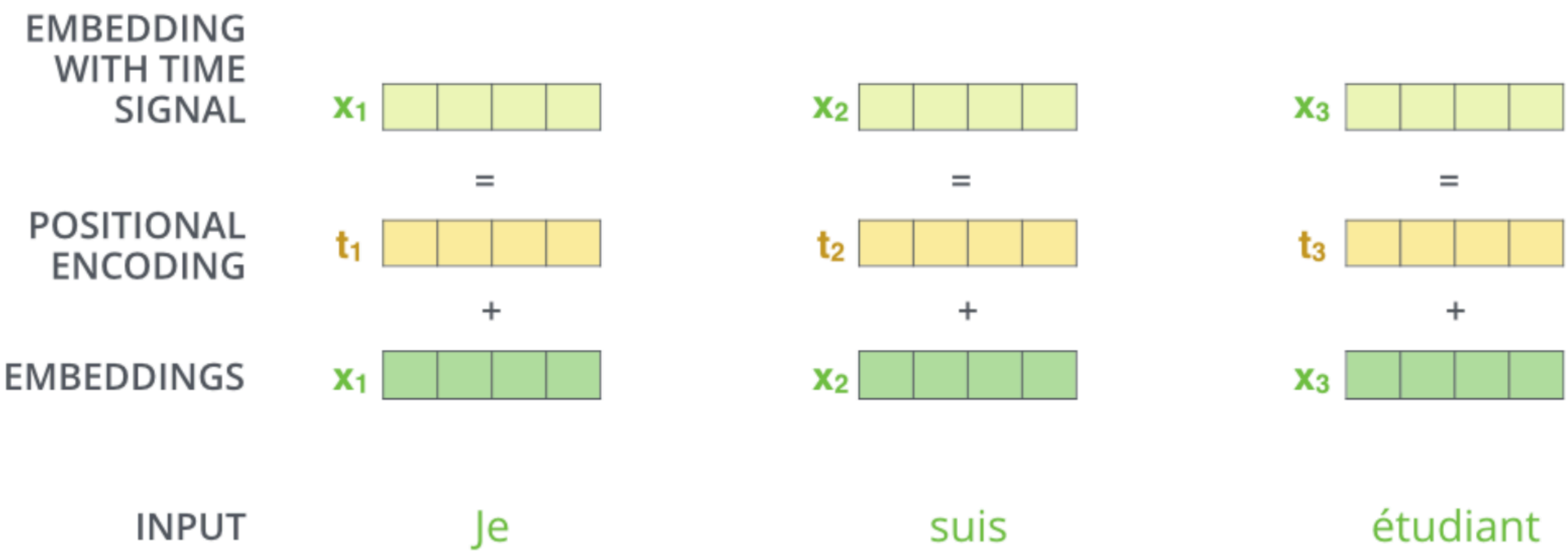


Sequence-to-Sequence w/ Transformers ~~RNNs~~

Train Time	**Test Time**
Encoder	
• Runs parallel ~~iteratively~~, joint ~~bi-directional~~.	
Decoder	
• Conditioned on full source + decoder history. • Runs parallel ~~iteratively~~, left-to-right. • Input is from “teacher forcing”.	• Runs iteratively, left-to-right. • Input is from “own predictions”.

Don’t forget, transformers require some tricks…

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

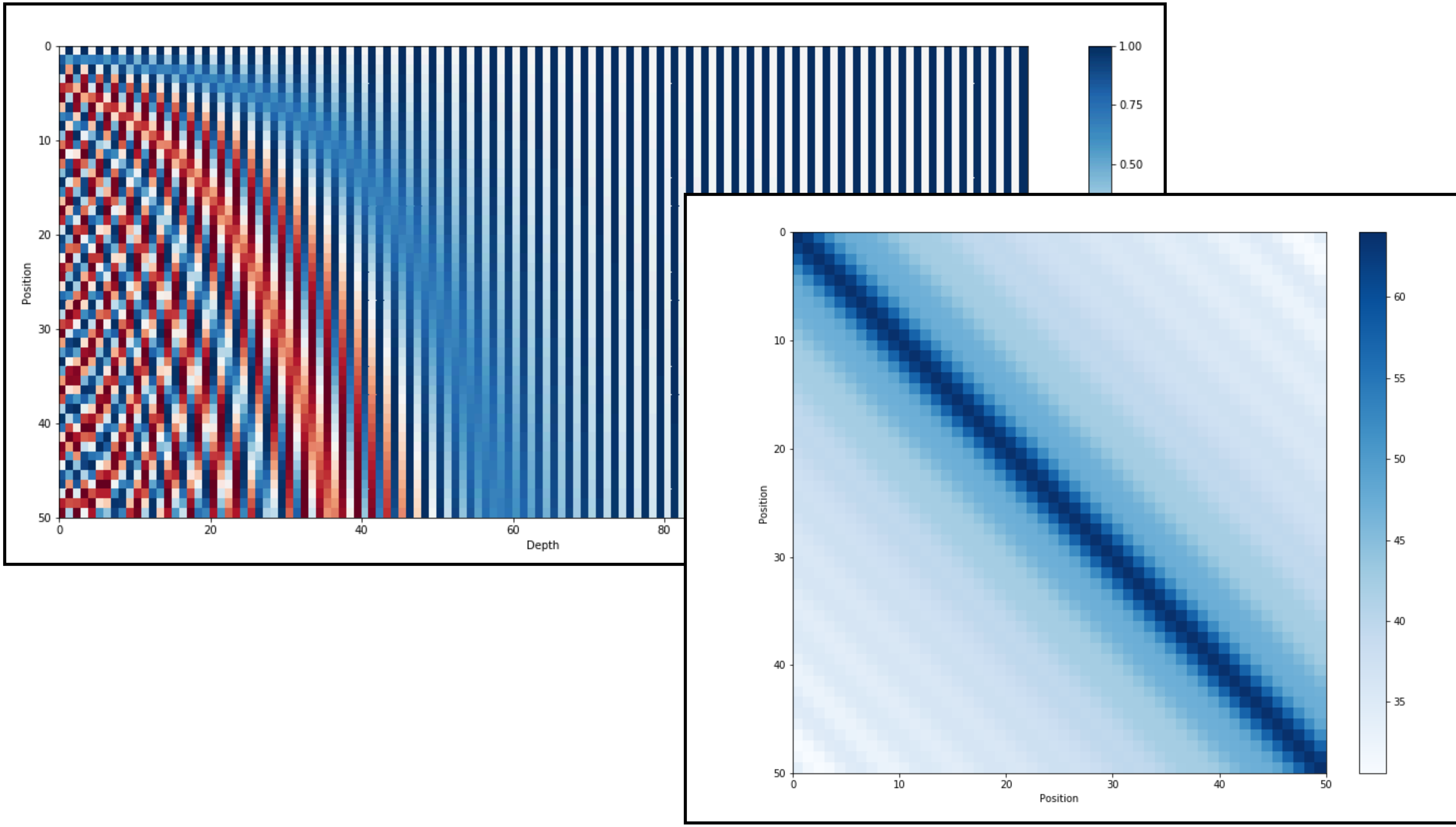
$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{model}}}\right)$$

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

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

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)

How to train transformers?

- Language Model: GPT (Summer 2018)

Few had the impact of Sondheim, shaping modern musicals.

Few had the impact of <u>Sondheim</u>

How to train transformers?

- Language Model: GPT (Summer 2018)
- Masked Language Model: BERT (Fall 2018)

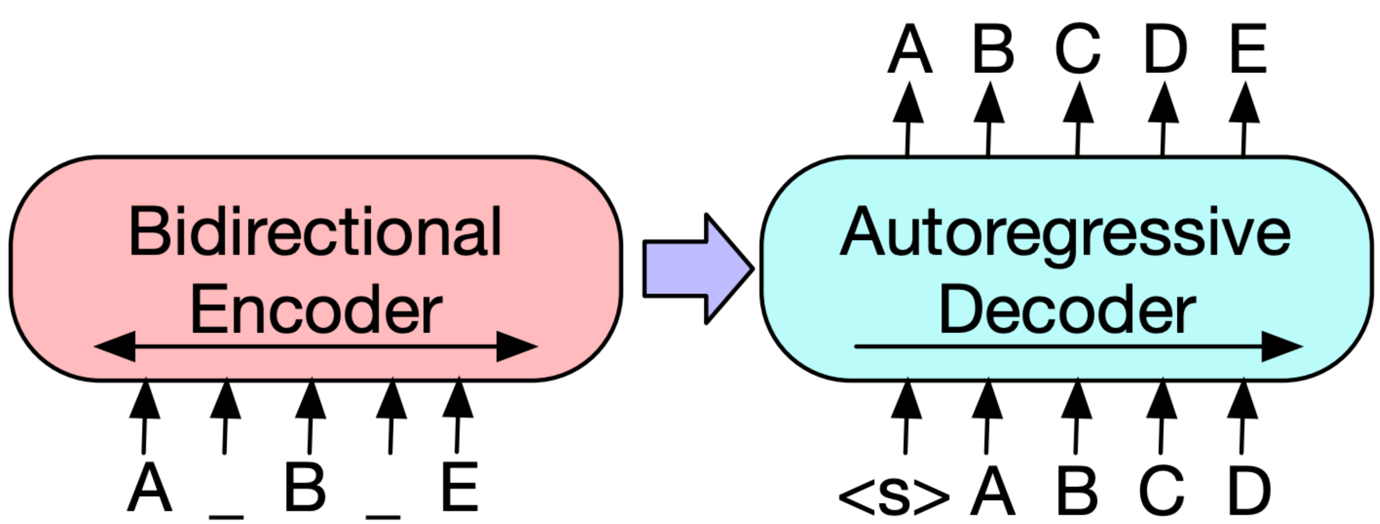
  For 15% of words,
  mask 80% of the time,
  swap 10% of the time,
  leave 10% of the time.

Few had the impact of Sondheim, shaping modern musicals.

<u>ocean</u> had the <u>[MASK]</u> of <u>Sondheim</u>, shaping <u>[MASK]</u> musicals.

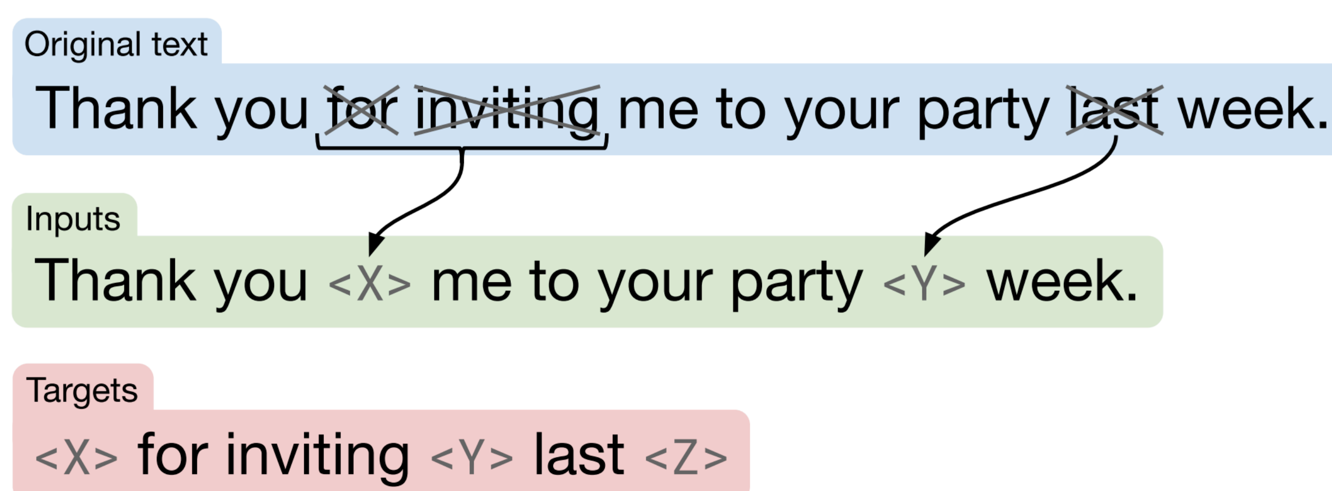
How to train transformers?

- Language Model: GPT (Summer 2018)
- Masked Language Model: BERT (Fall 2018)
- Autoregressive MLM: BART (Fa 2019), MASS (Su ’19)



How to train transformers?

- Language Model: GPT (Su ’18)
- Masked Language Model: BERT (Fa ’18)
- Autoregressive MLM: BART (Fa ’19), MASS (Su ’19)
- Sentinel Autoregressive MLM: T5 (Fa ’19)



Other tricks…

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.

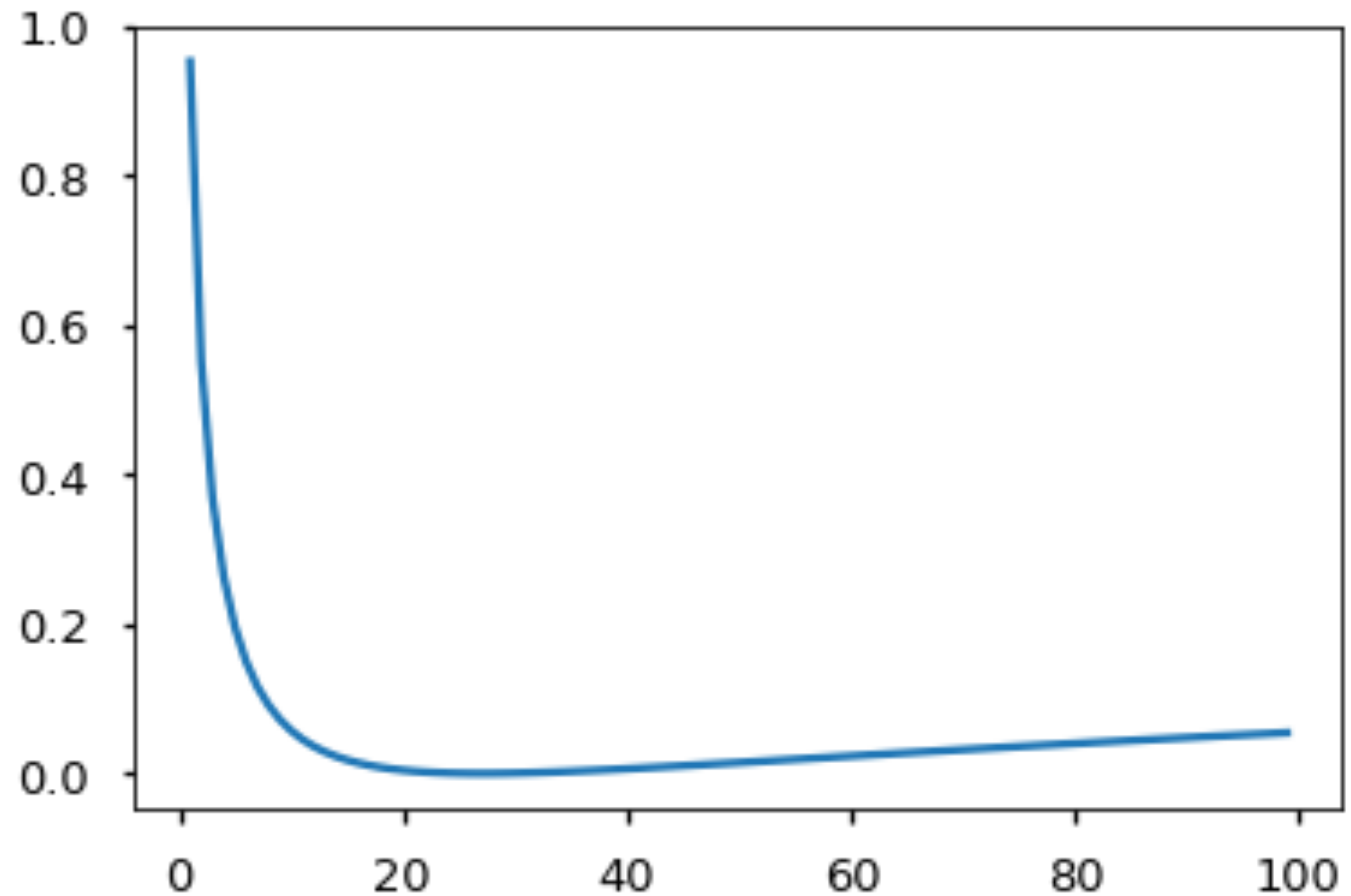
> *We implement label smoothing using the KL div loss. Instead of using a one-hot target distribution, we create a distribution that has `confidence` of the correct word and the rest of the `smoothing` mass distributed throughout the vocabulary.*

I went to class and took ___

cats	TV	notes	took	sofa
0	0	1	0	0
0.025	0.025	0.9	0.025	0.025

with label smoothing

Get penalized for overconfidence!



Loss

Target word confidence

https://nlp.seas.harvard.edu/2018/04/03/attention.html

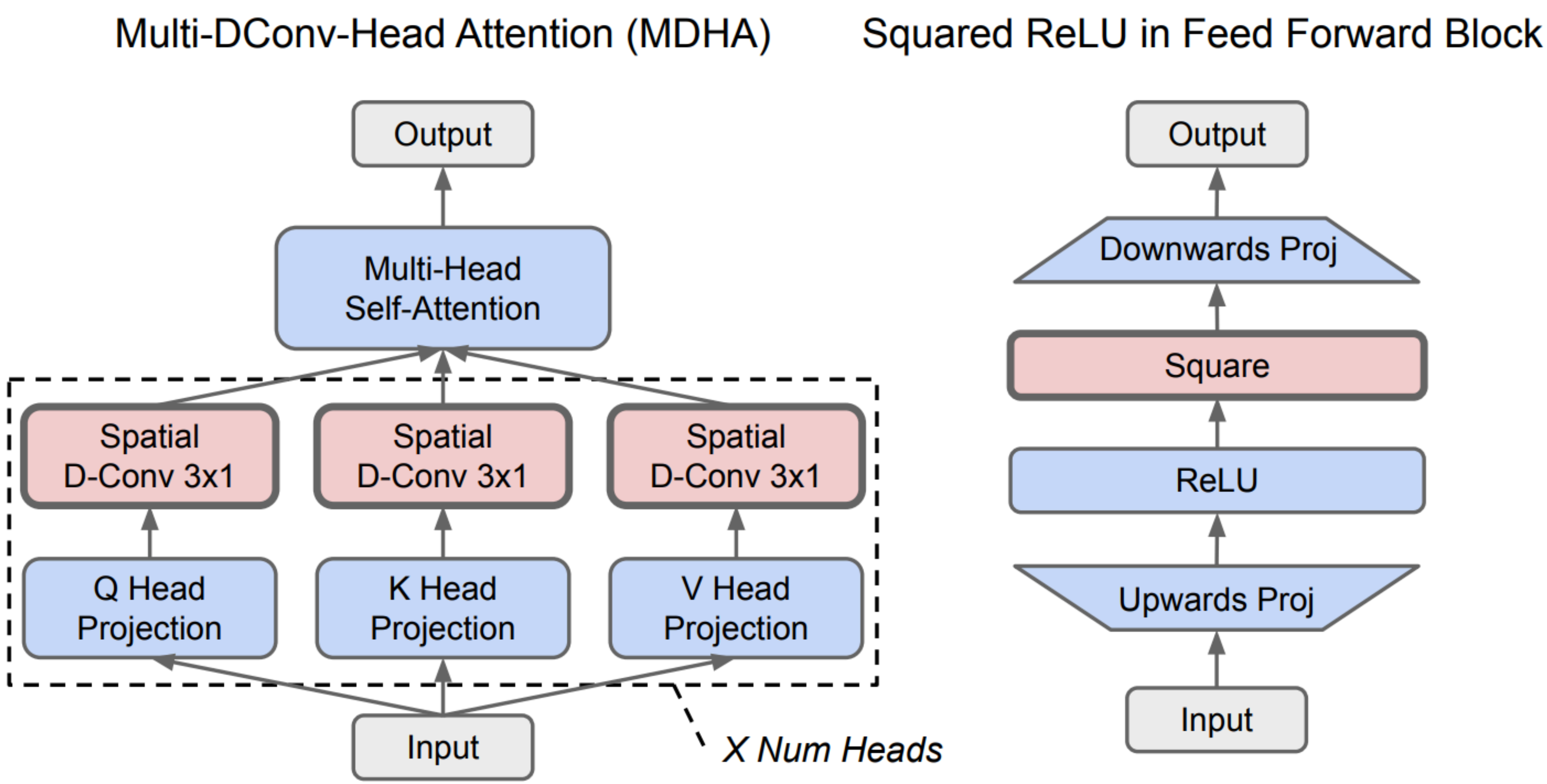
And more details…

- Tokenization (i.e. subwords) — Out of scope.
- Beam Search - It’s hard to do inference.
- Model Averaging - Easy way to ensemble seq2seq.

Why these decisions?

Unsatisfying answer: they empirically worked well.

Neural architecture search finds even better Transformer variants:



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