

# Transformers and sequence- to-sequence learning

CS 685, Spring 2022

**Mohit Iyyer**

College of Information and Computer Sciences  
University of Massachusetts Amherst

*some slides from Emma Strubell*

# From last time

- Project proposals due Friday 2/18
- Quiz 1 due Monday 2/21

# 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  $f_1, f_2, \dots, f_n$  produce English translation  $e$  with tokens  $e_1, e_2, \dots, e_m$
- **real goal:** compute  $\arg \max_e p(e | f)$

# This is an instance of *conditional language modeling*

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

Just like we've seen before, except we additionally condition our prediction of the next word on some other input (here, the French sentence)

# seq2seq models

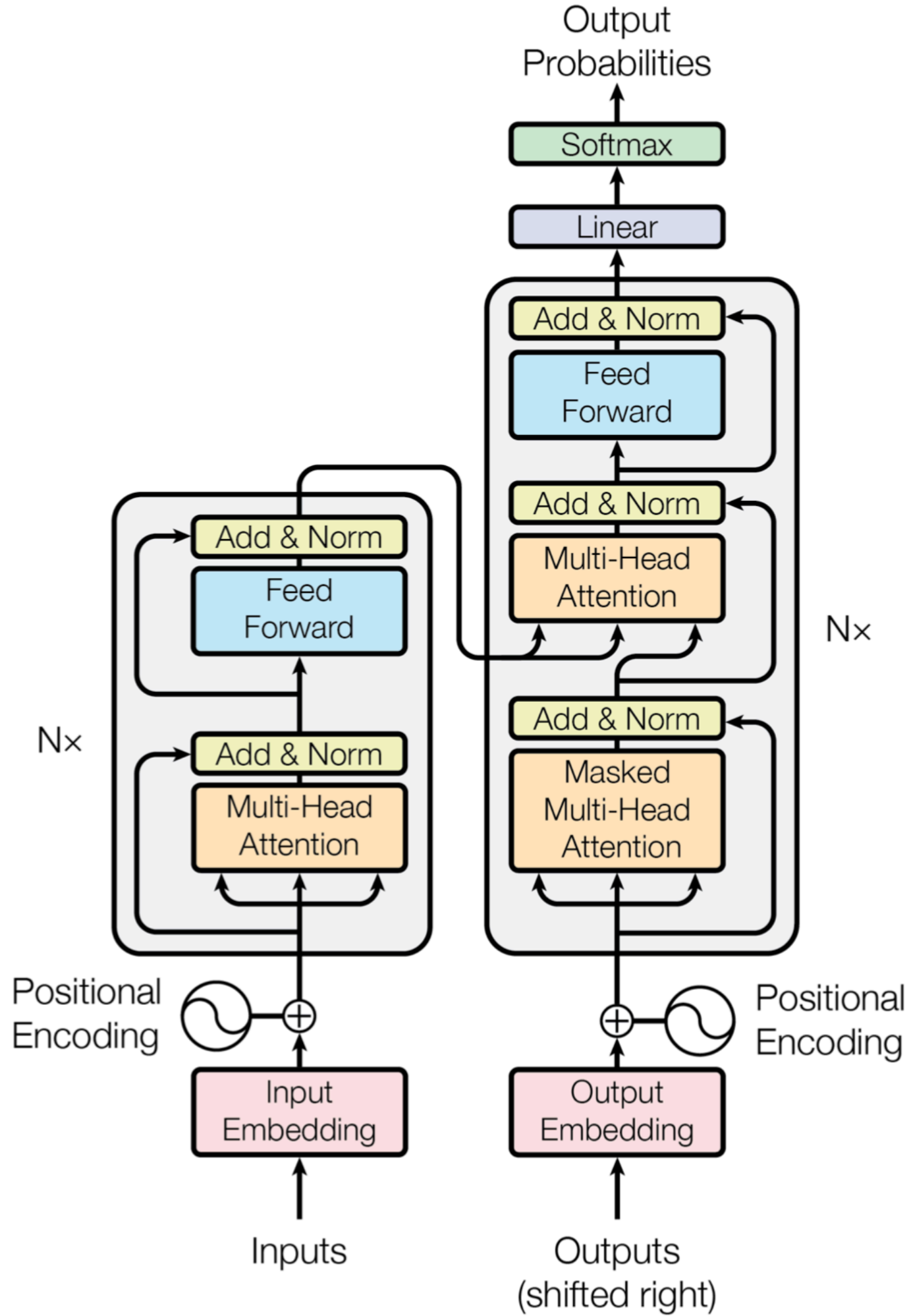
- use two different neural networks to model

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

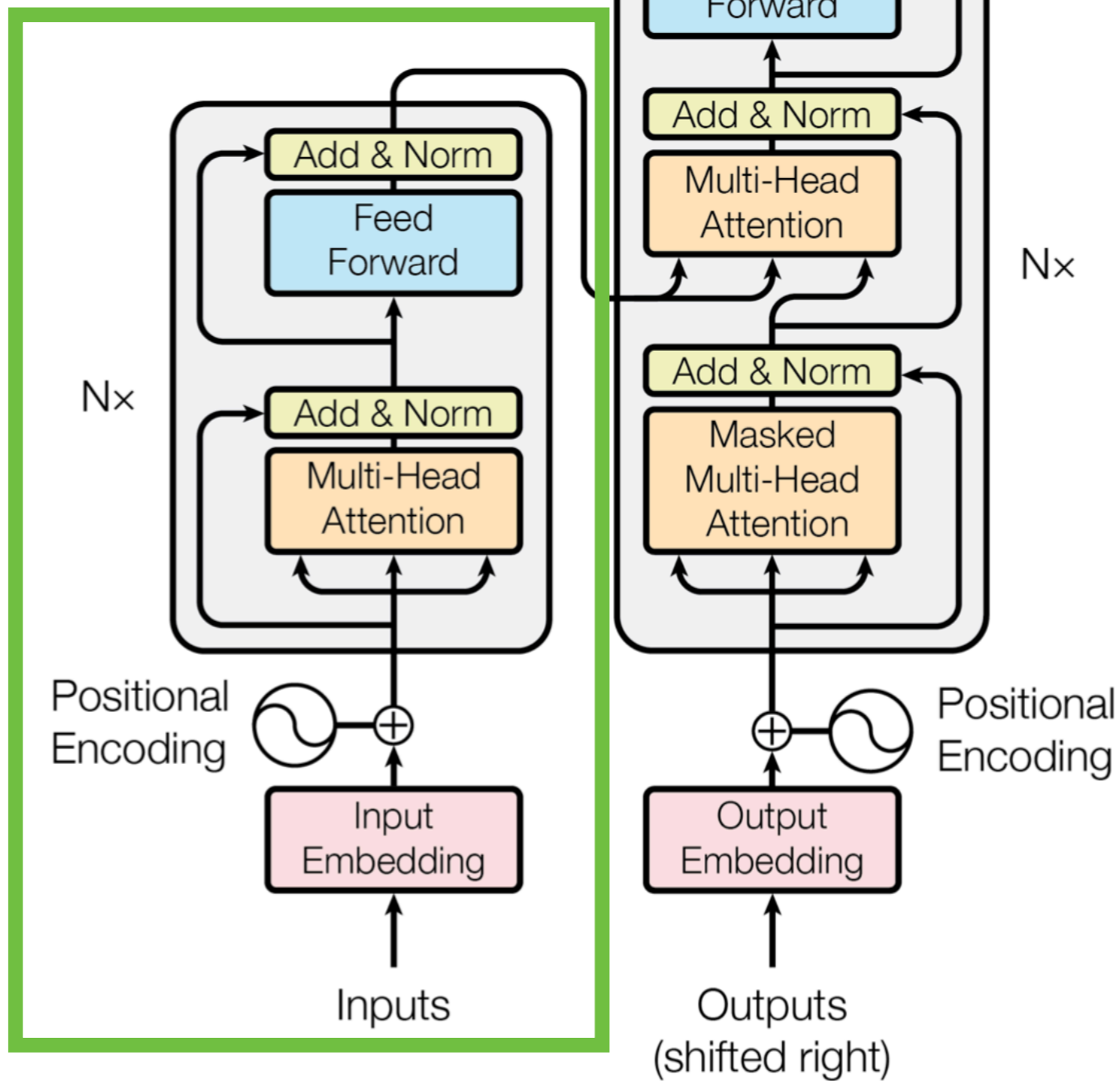
- first we have the *encoder*, which encodes the French sentence  $f$
- then, we have the *decoder*, which produces the English sentence  $e$

iPad

Introduce seq2seq on iPad too

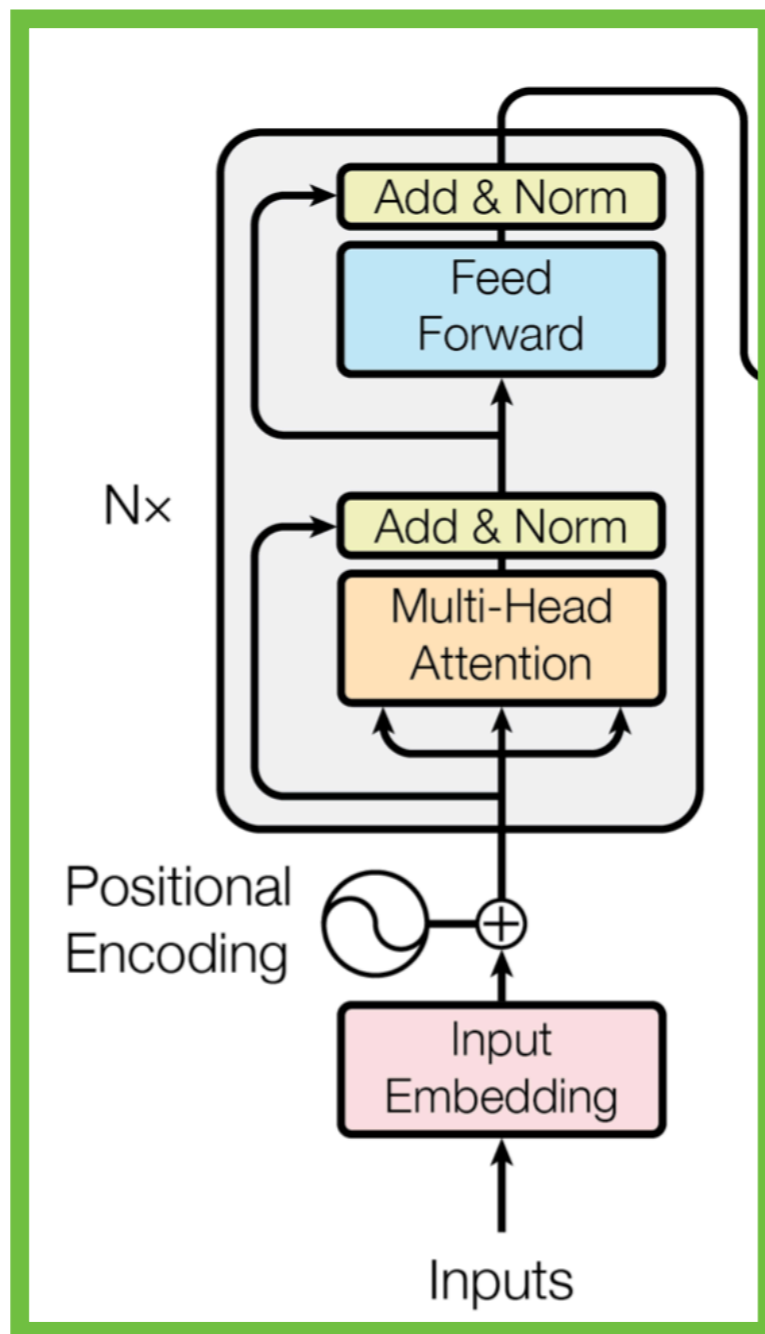


*encoder*

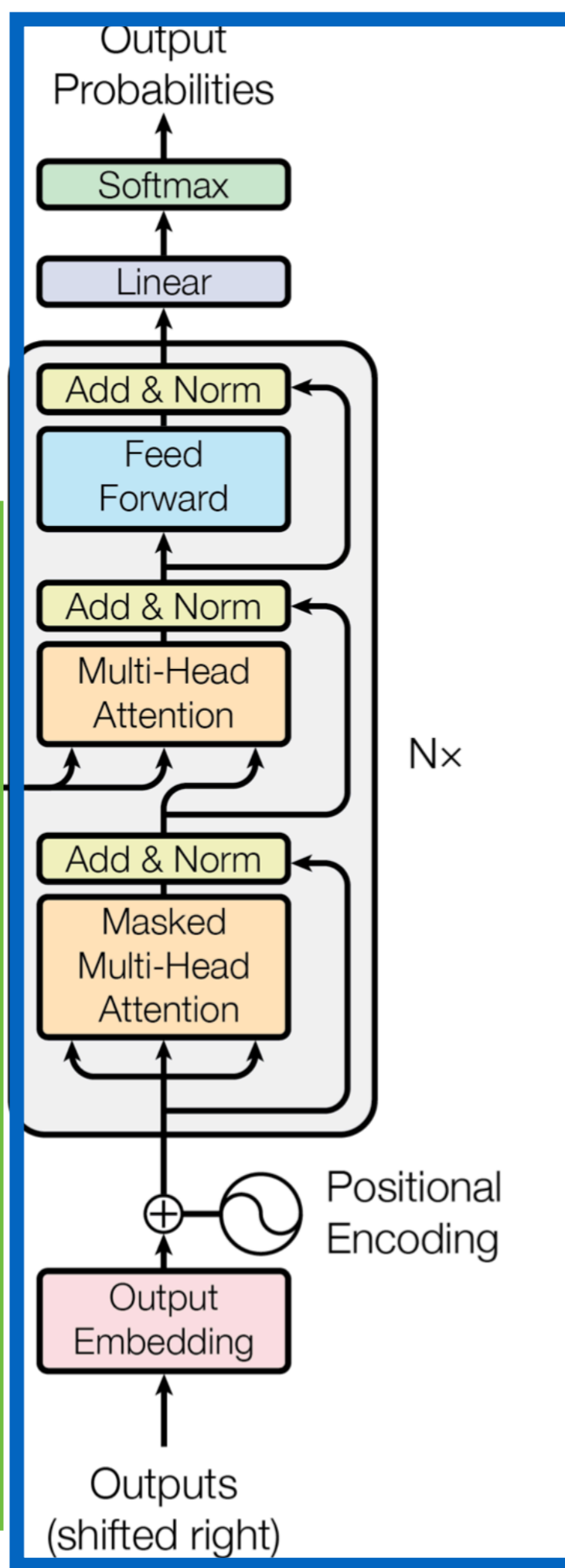




*encoder*

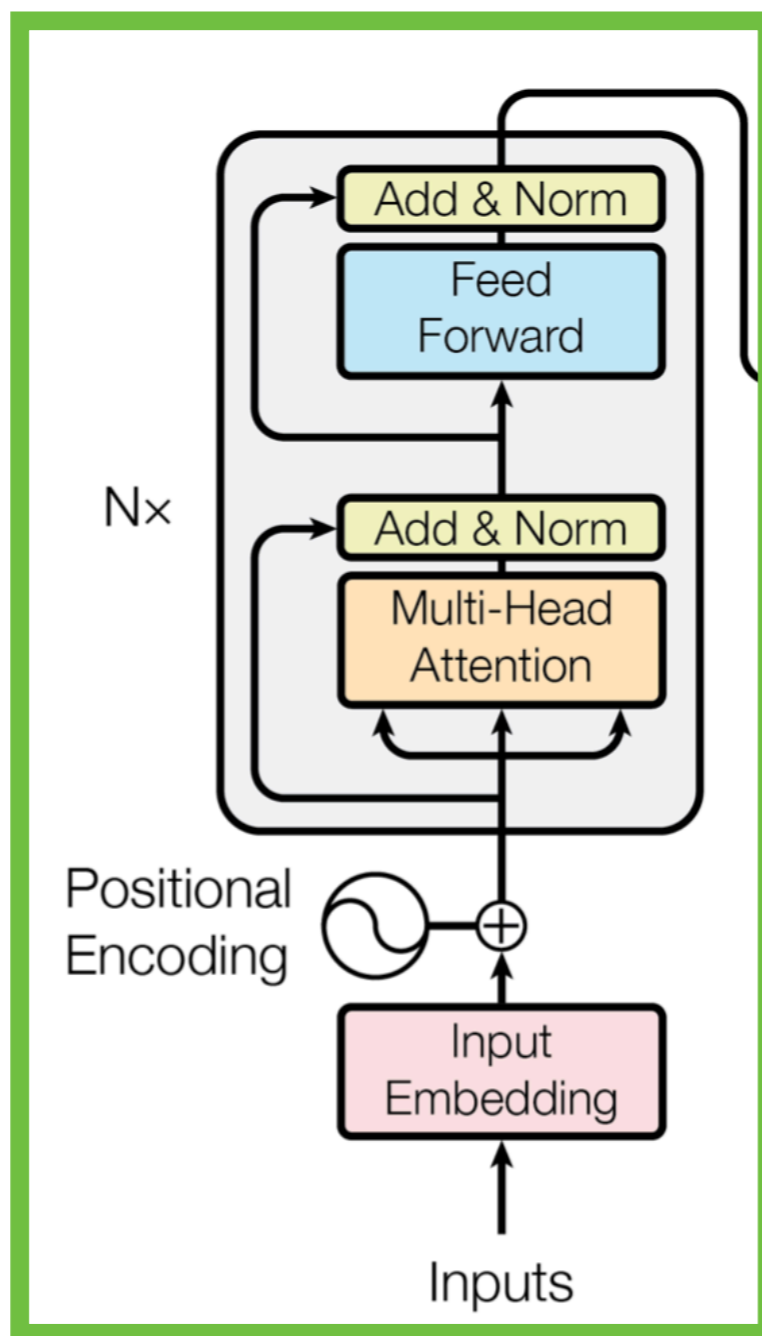


*decoder*

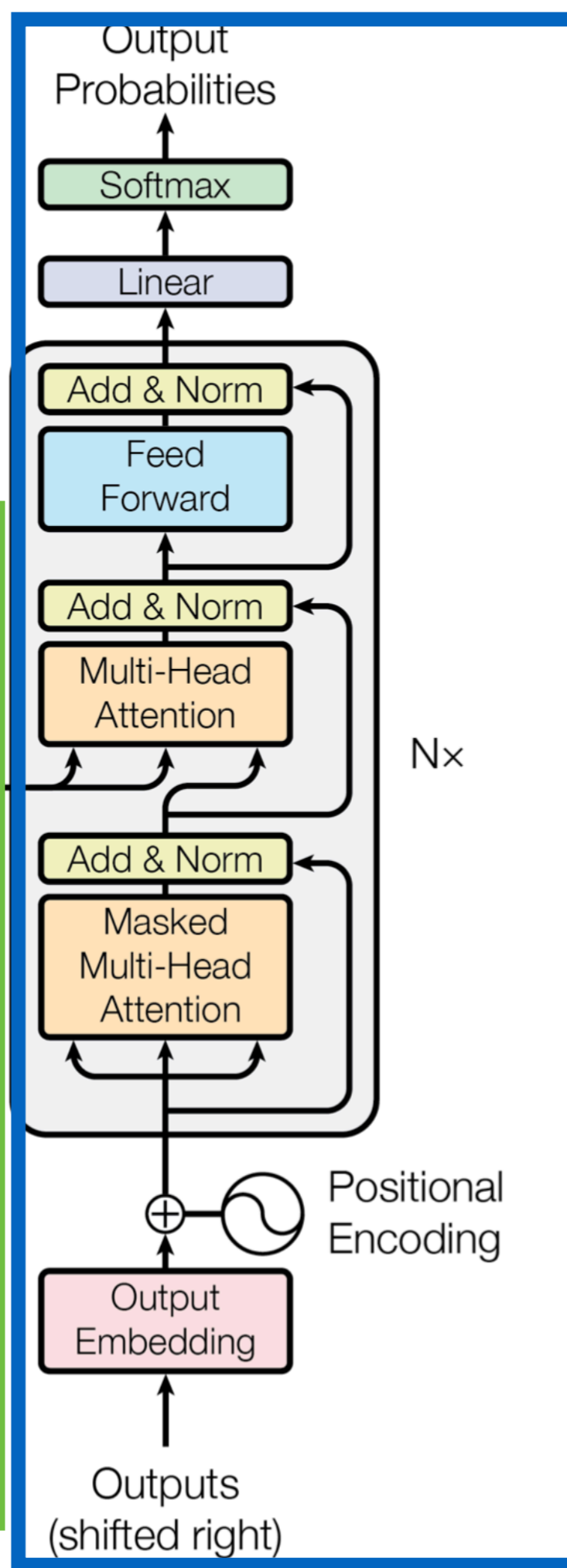


So far we've just talked about self-attention... what is all this other stuff?

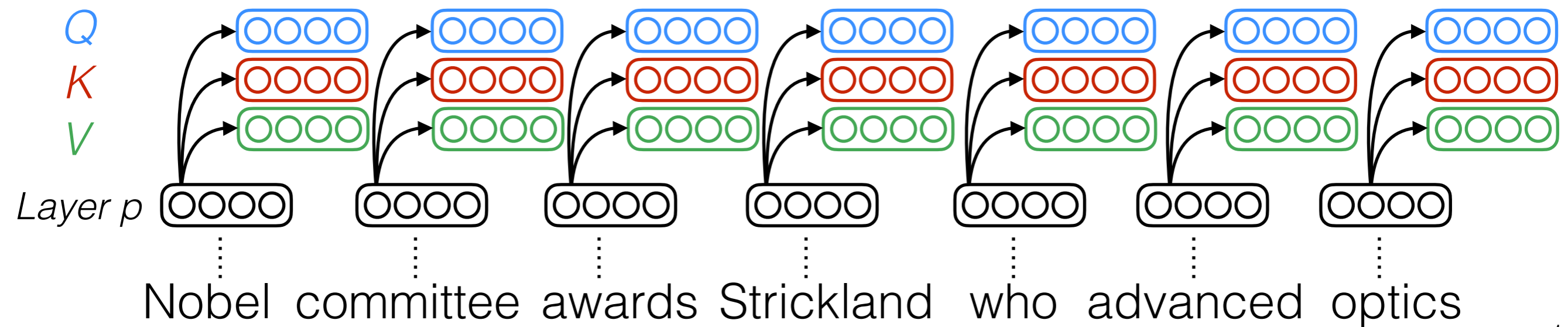
*encoder*



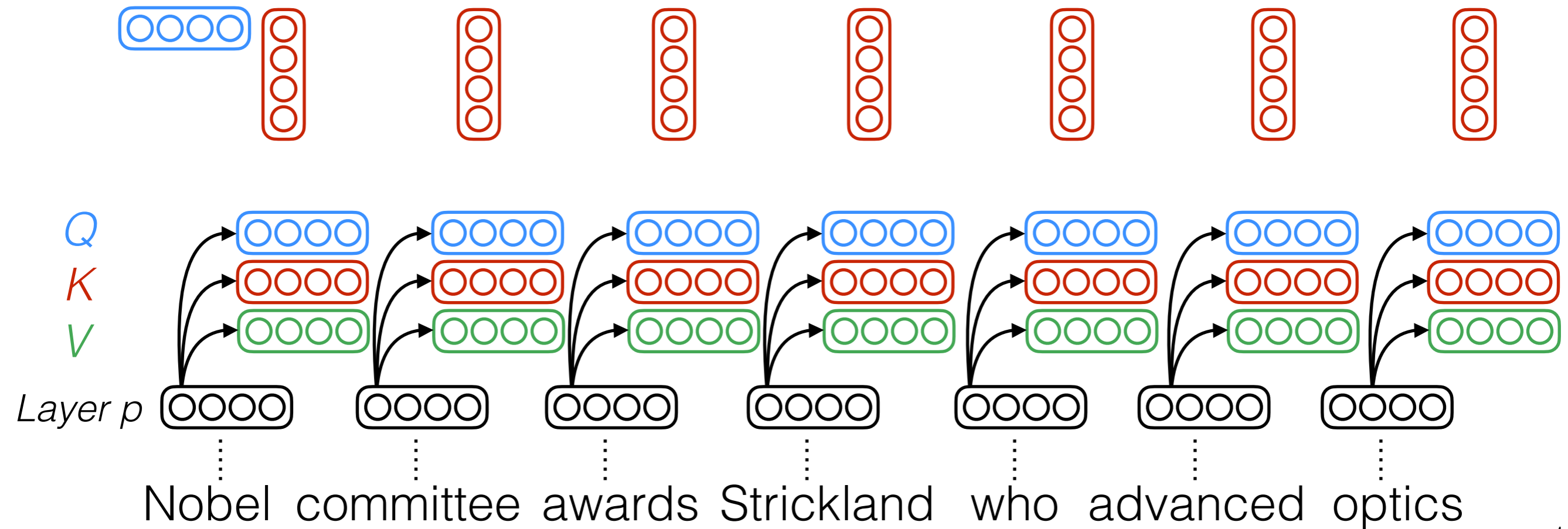
*decoder*



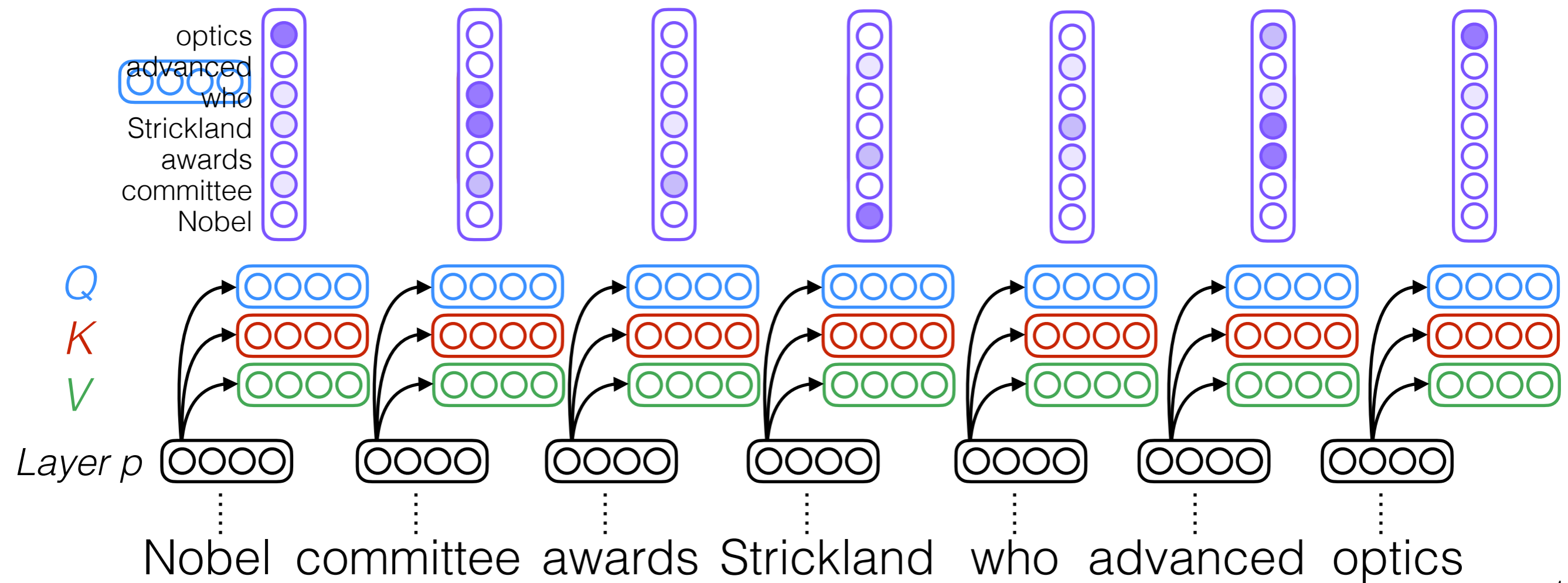
# Self-attention (in encoder)



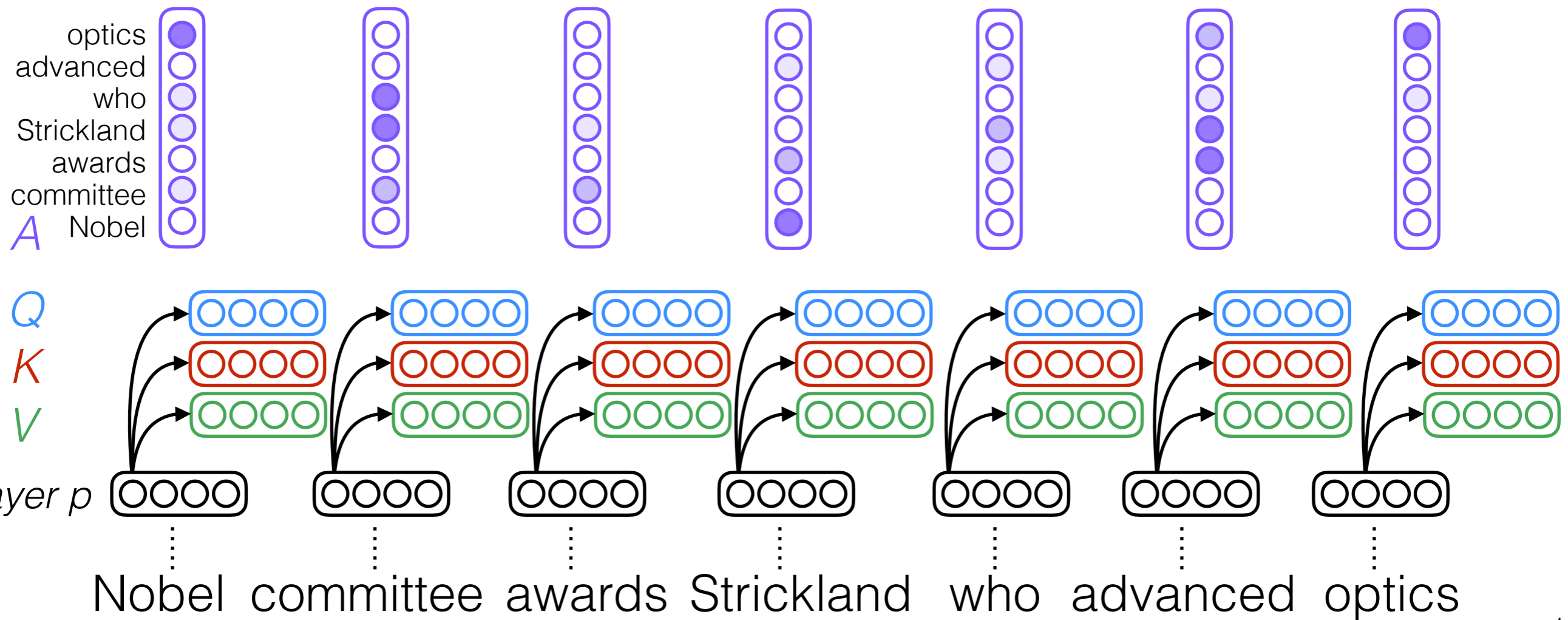
# Self-attention (in encoder)



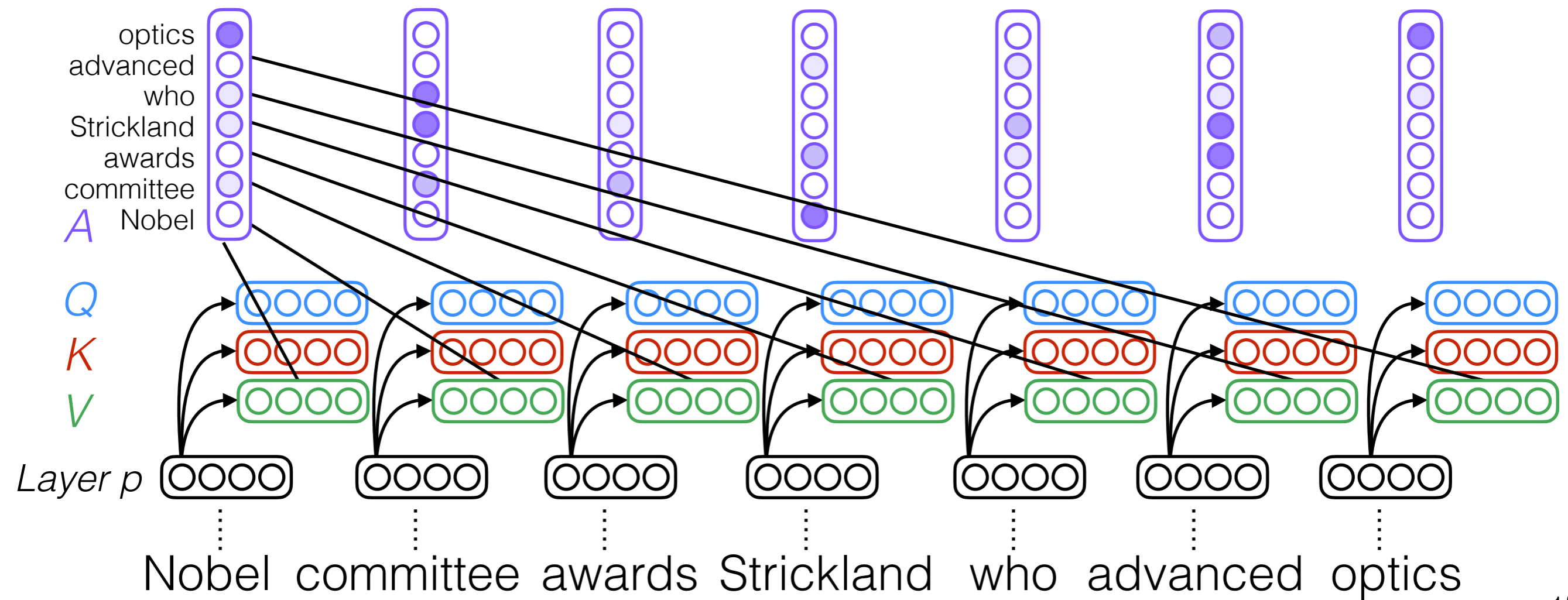
# Self-attention (in encoder)



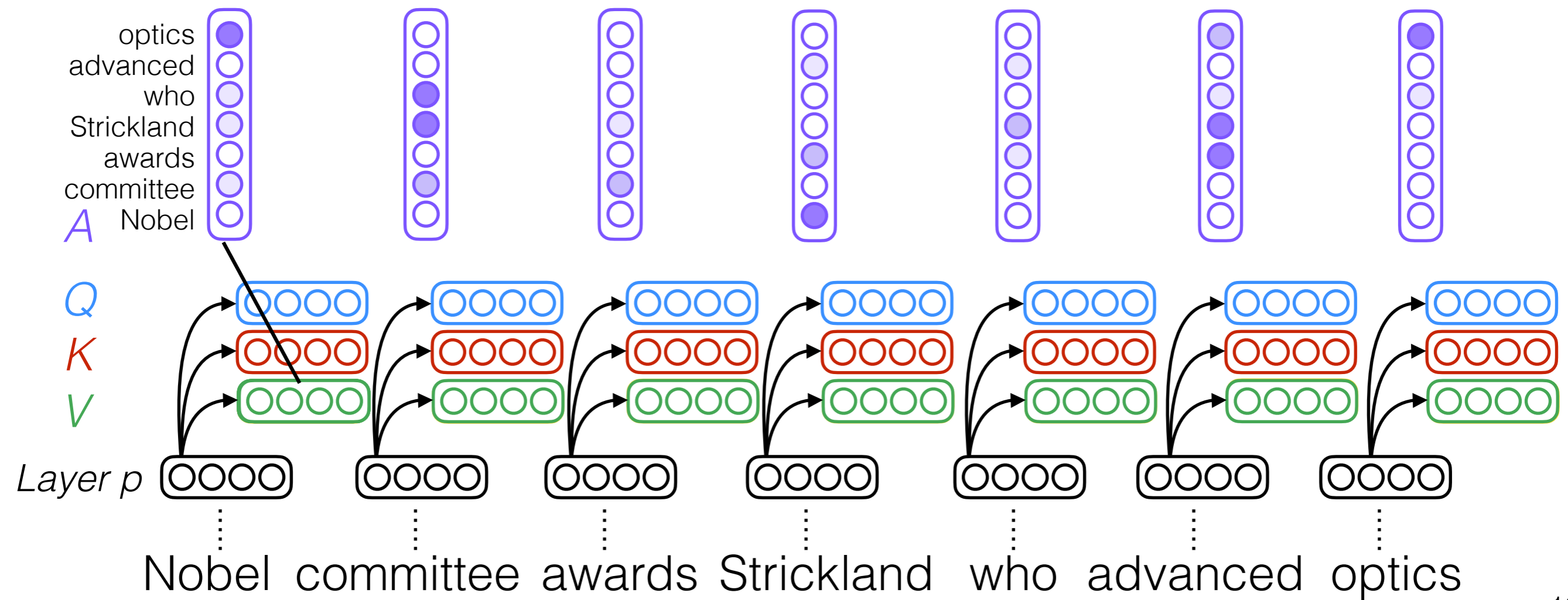
# Self-attention (in encoder)



# Self-attention (in encoder)

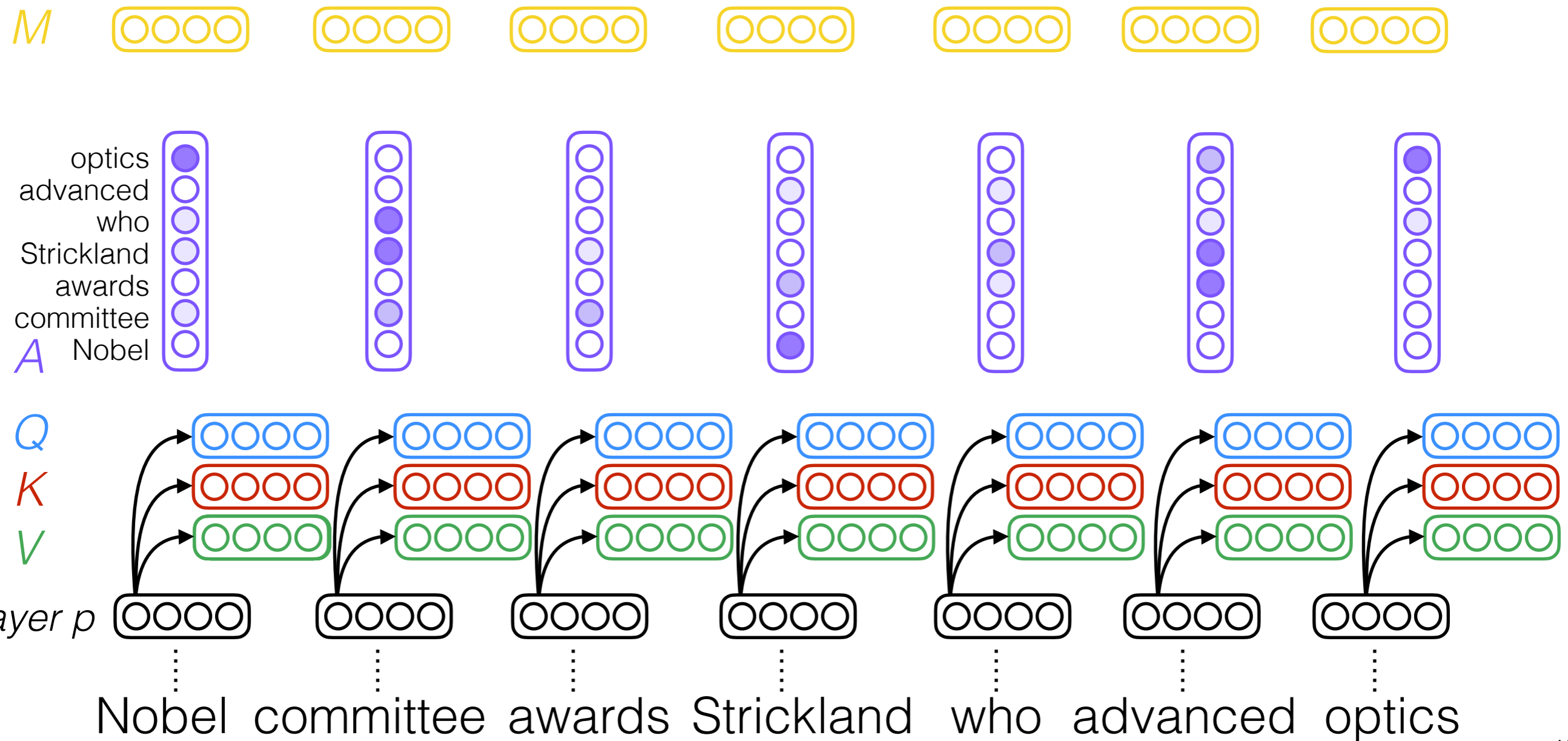


# Self-attention (in encoder)

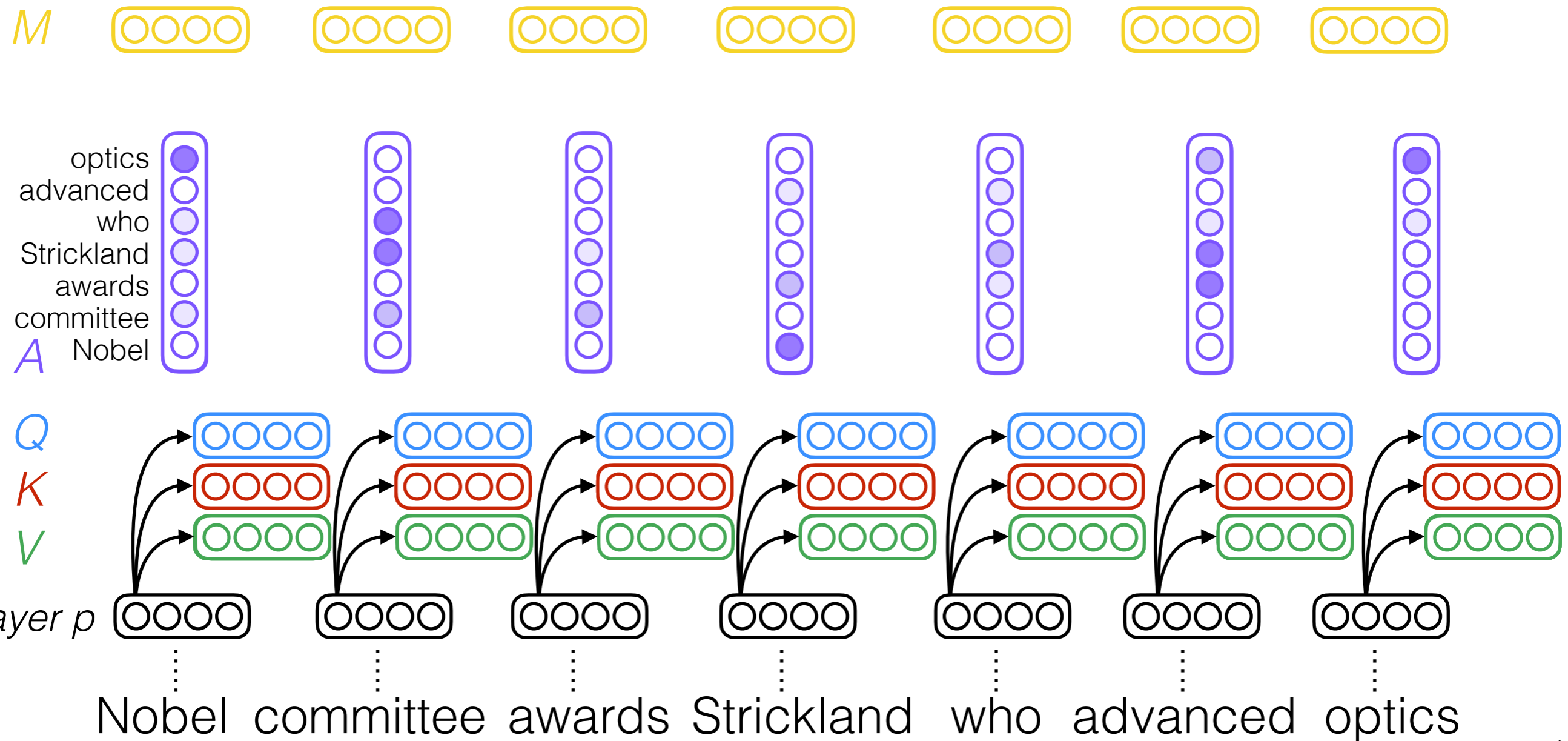




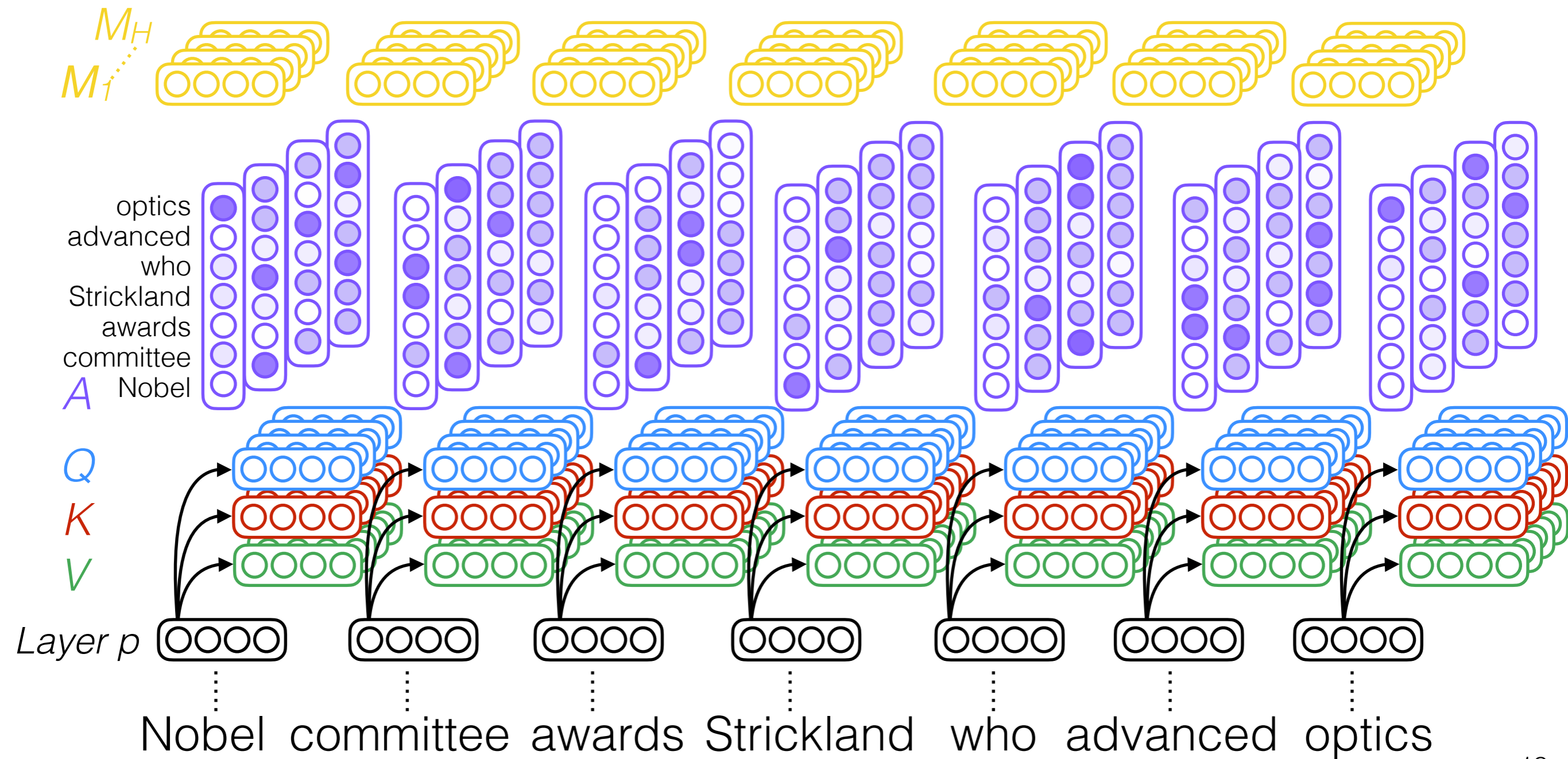
# Self-attention (in encoder)



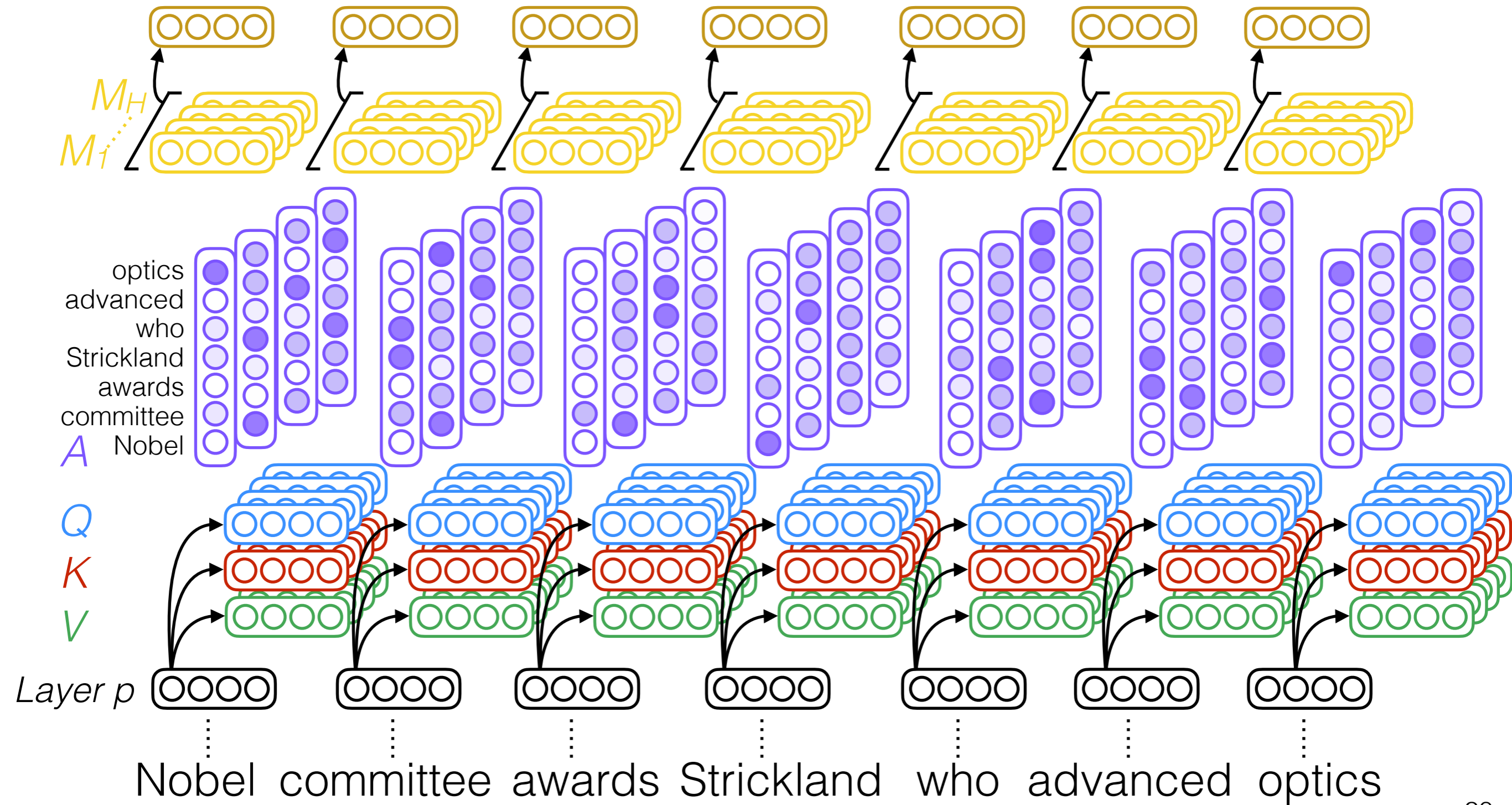
# Self-attention (in encoder)



# Multi-head self-attention

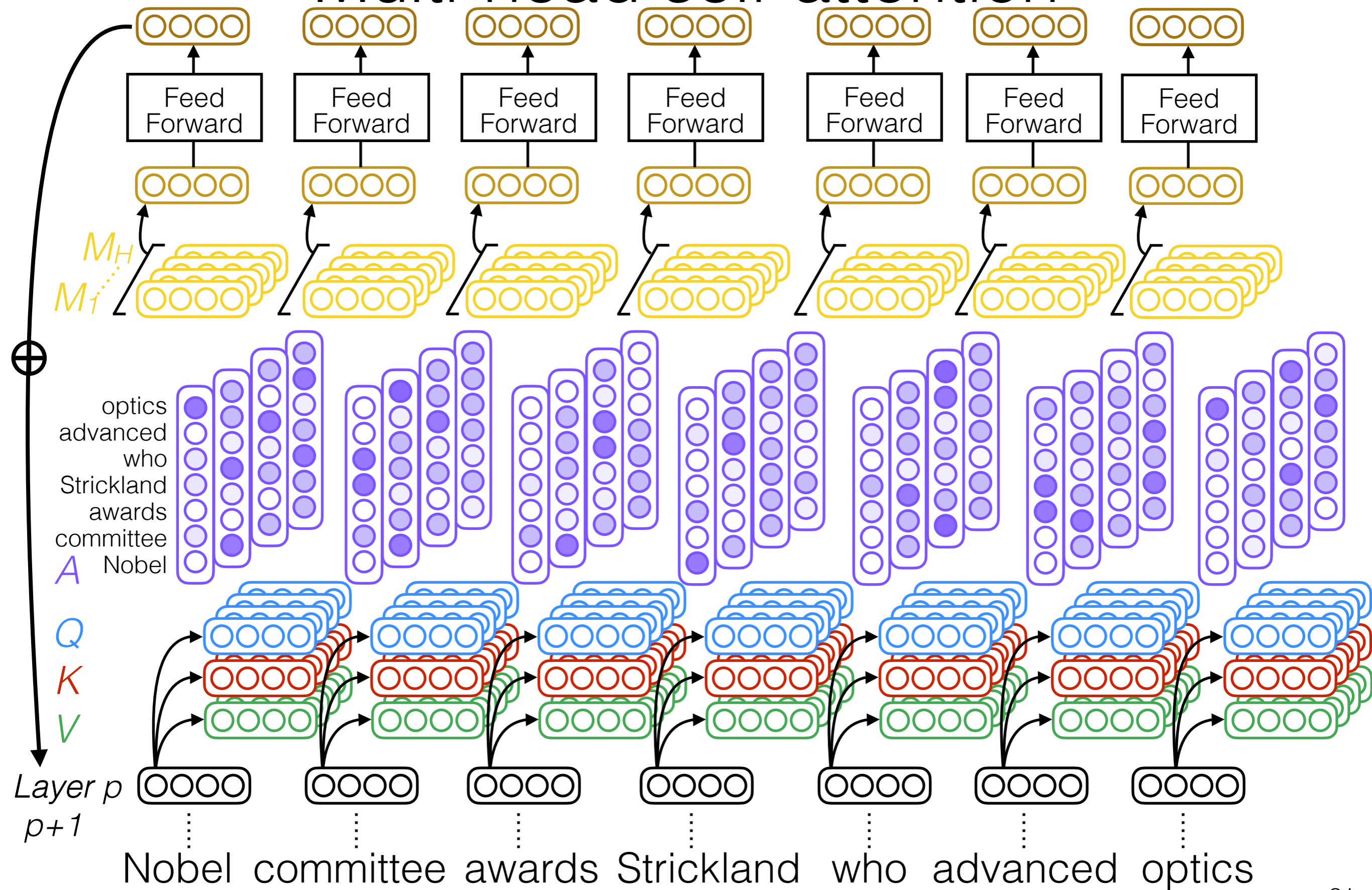


# Multi-head self-attention

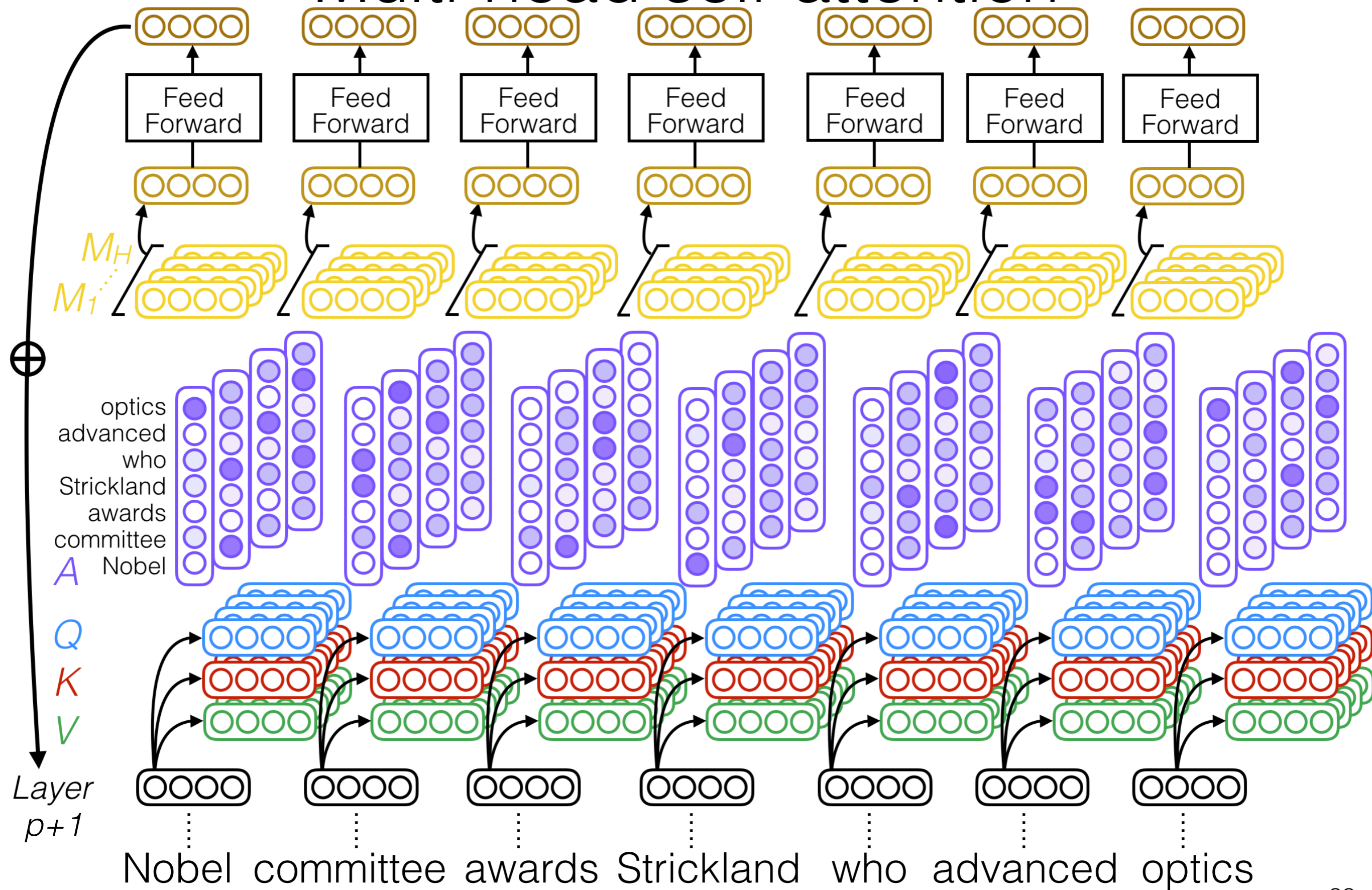




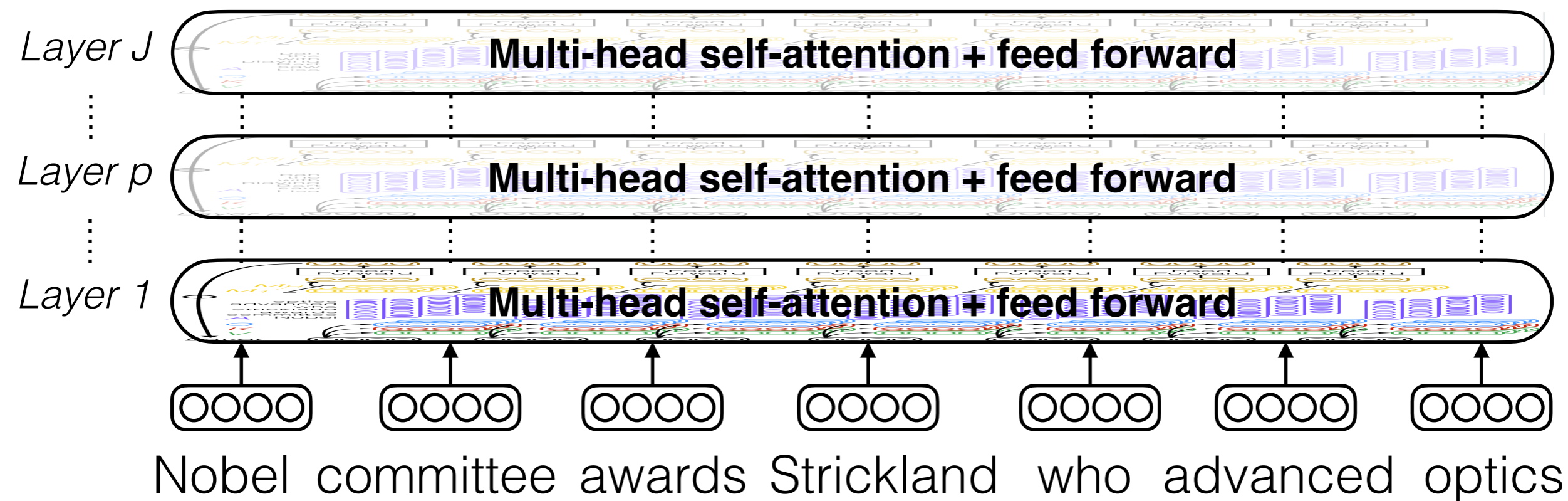
# Multi-head self-attention



# Multi-head self-attention

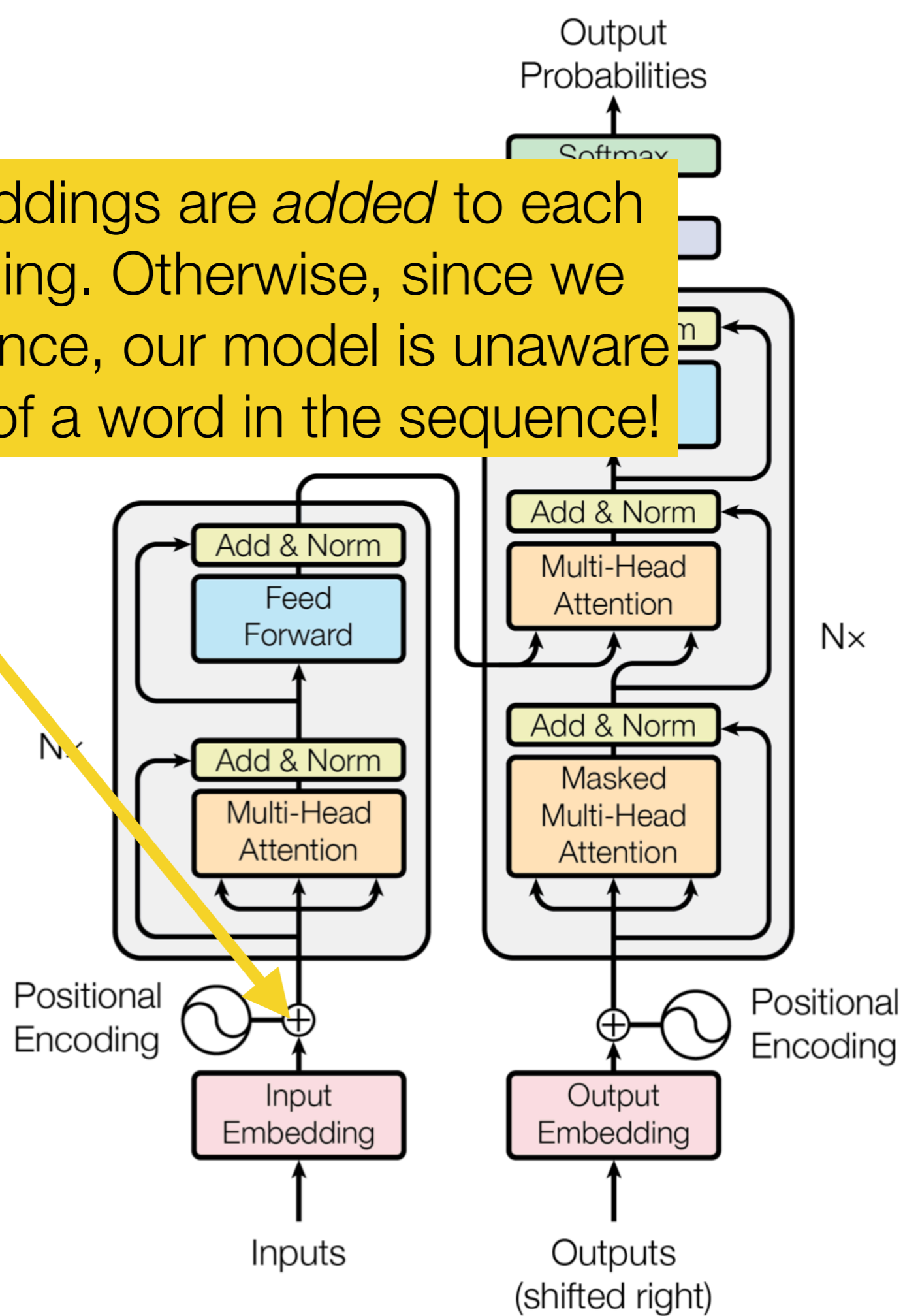


# Multi-head self-attention



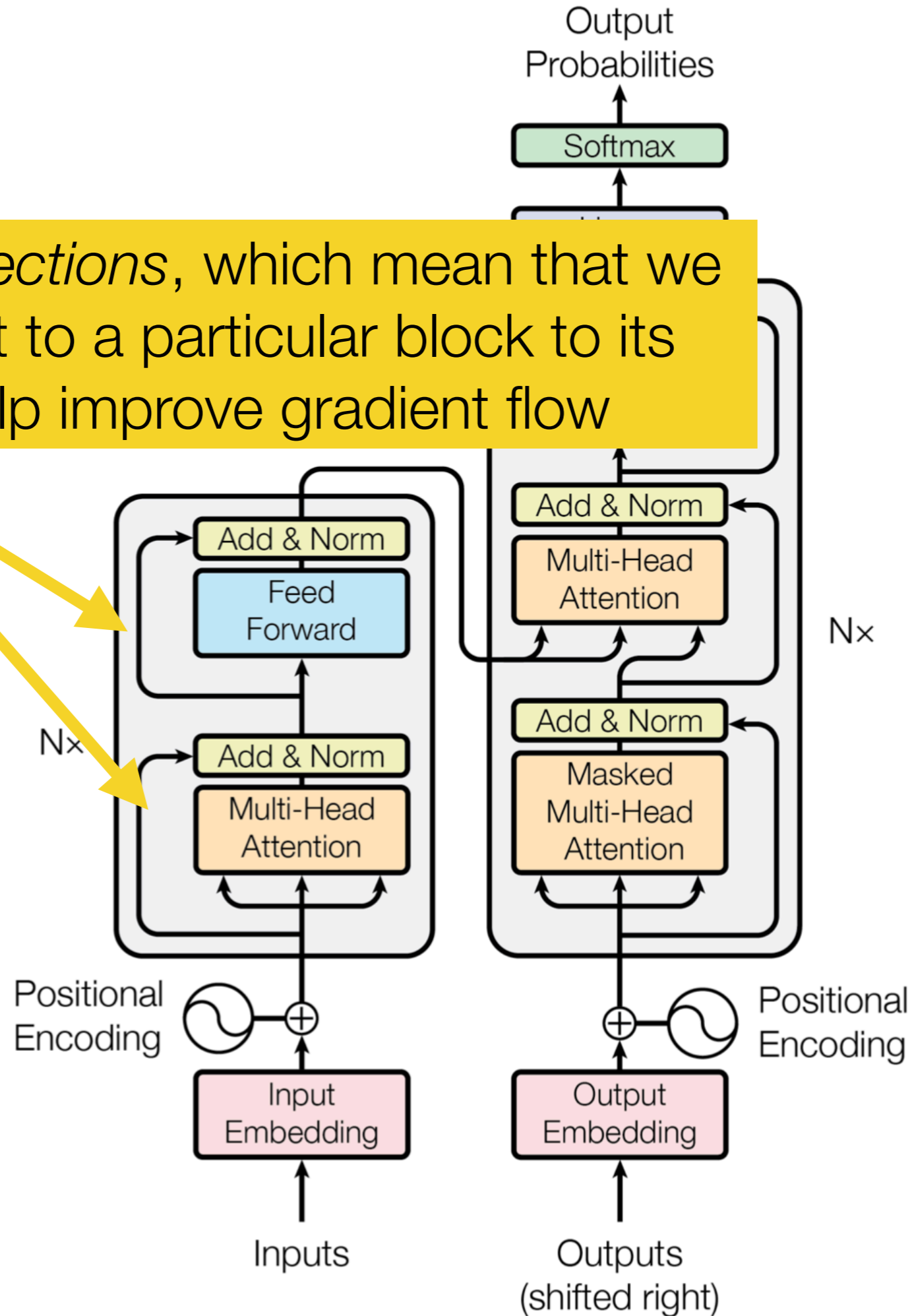


Position embeddings are *added* to each word embedding. Otherwise, since we have no recurrence, our model is unaware of the position of a word in the sequence!

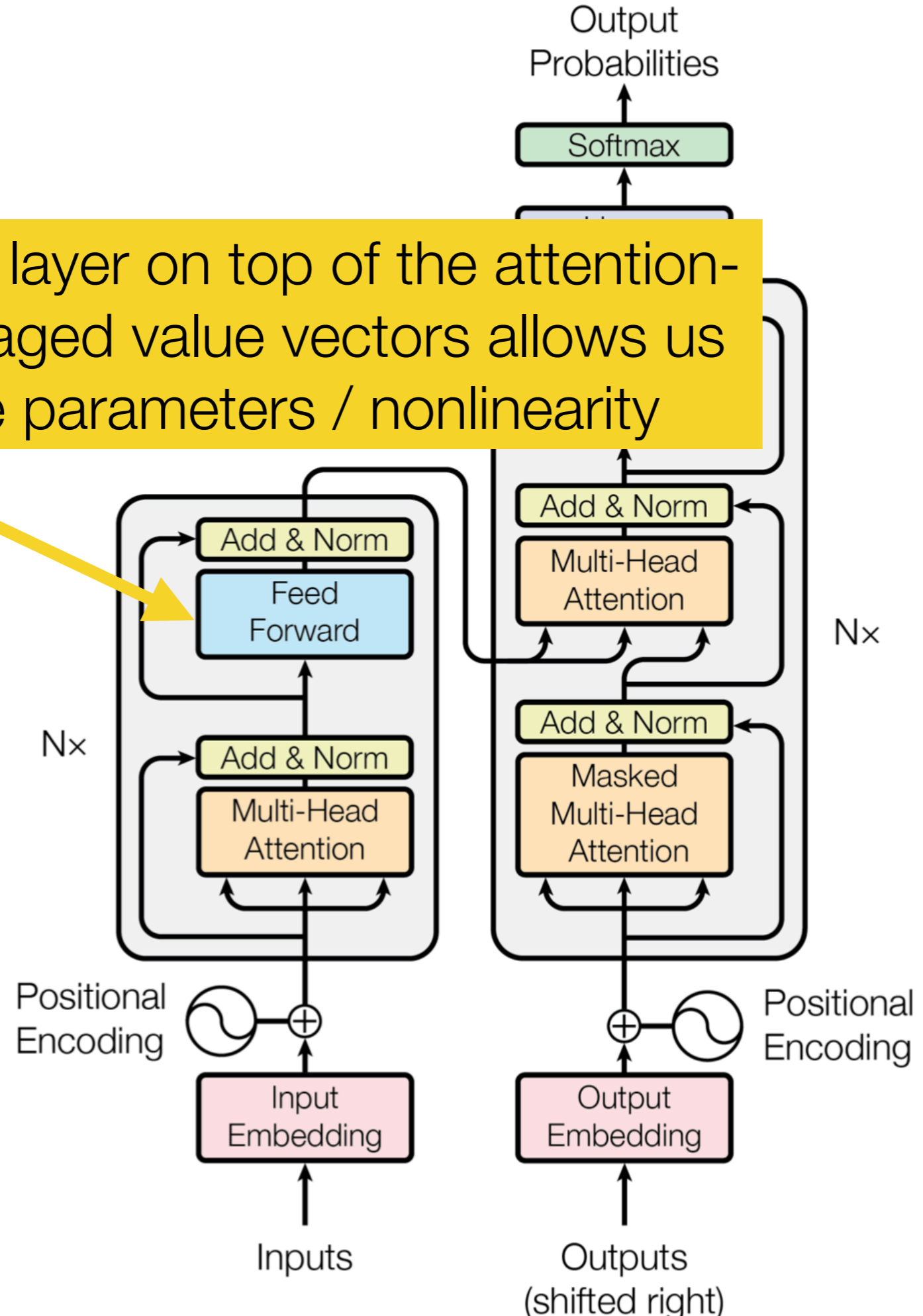




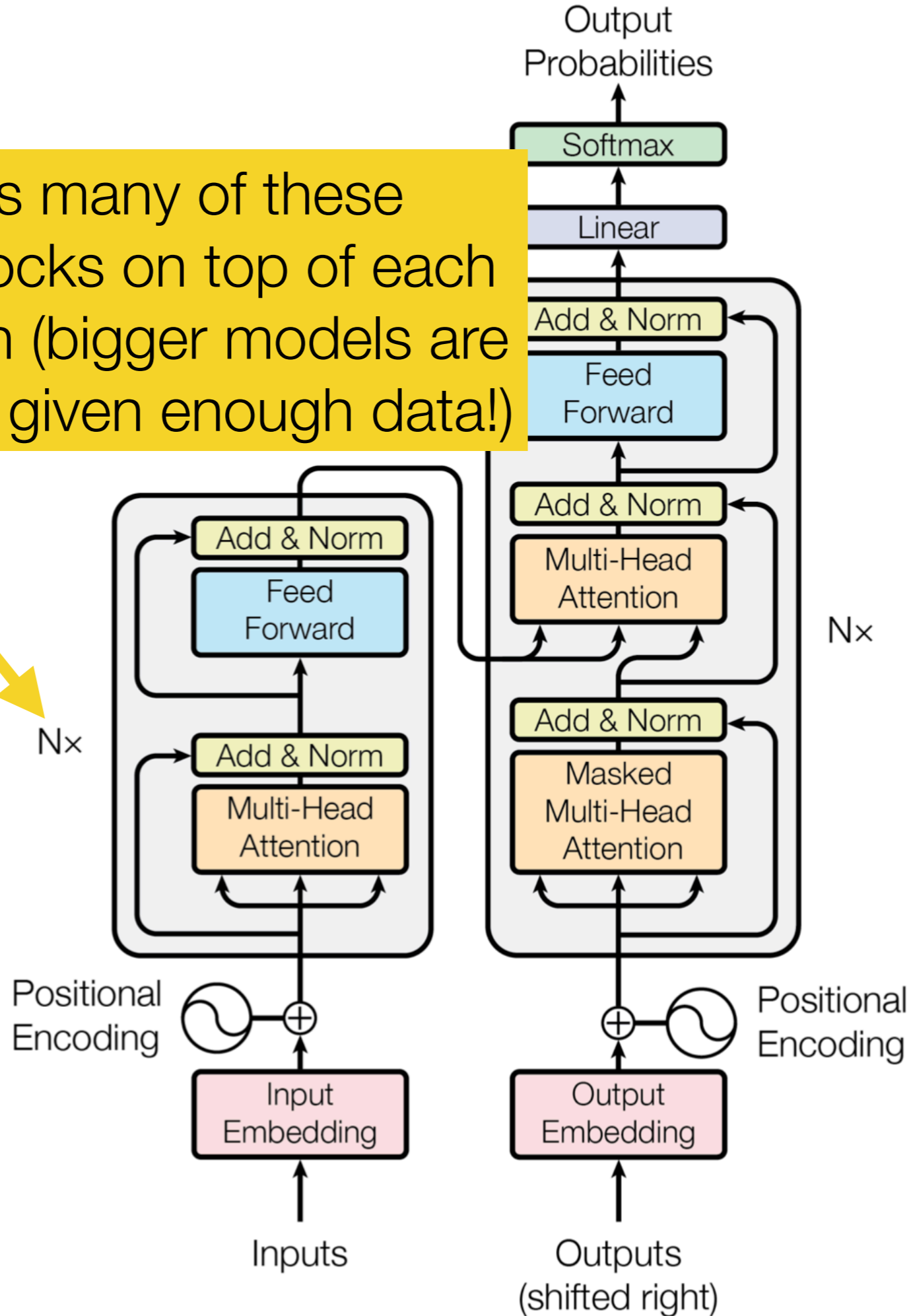
*Residual connections, which mean that we add the input to a particular block to its output, help improve gradient flow*



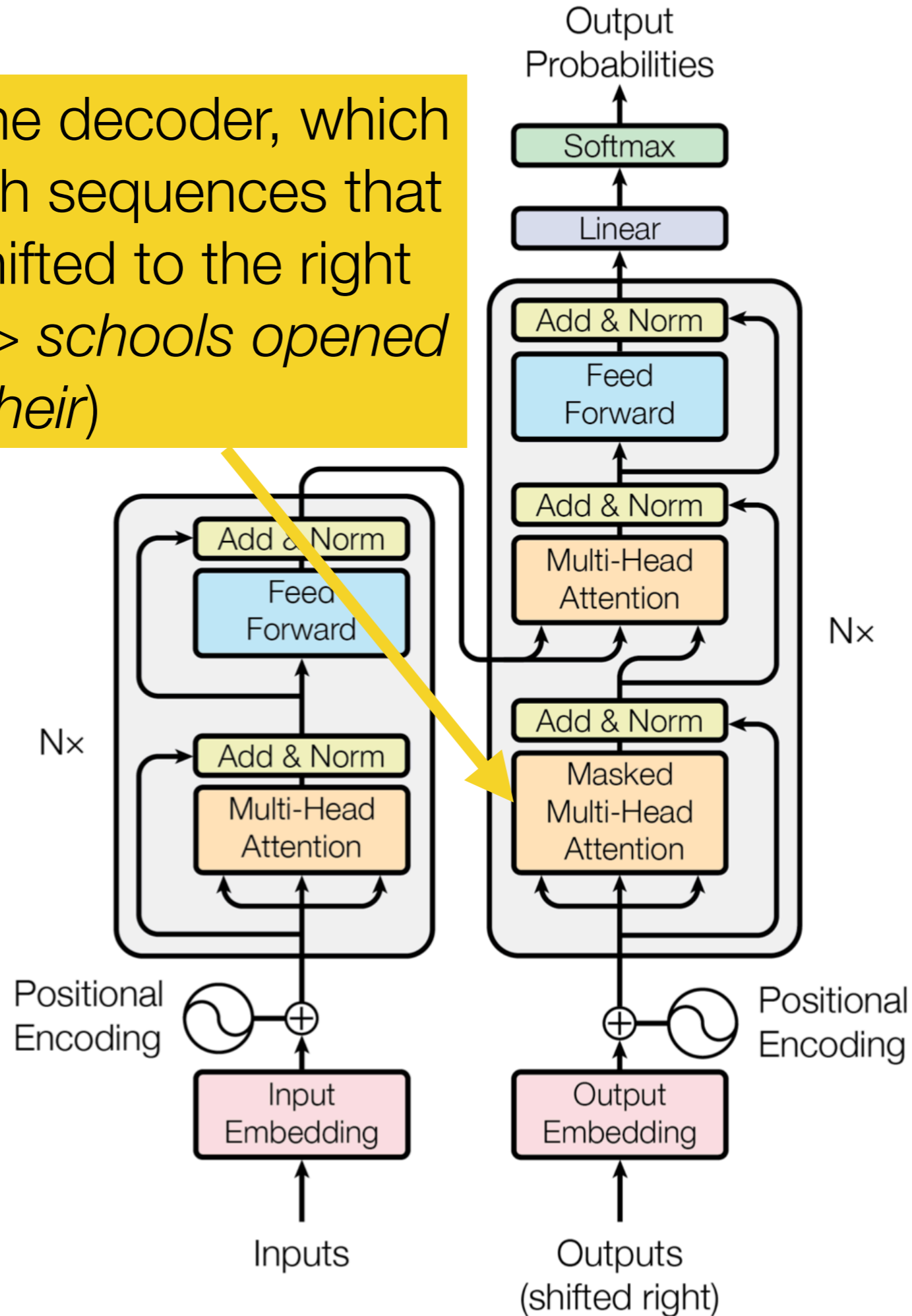
A feed-forward layer on top of the attention-weighted averaged value vectors allows us to add more parameters / nonlinearity



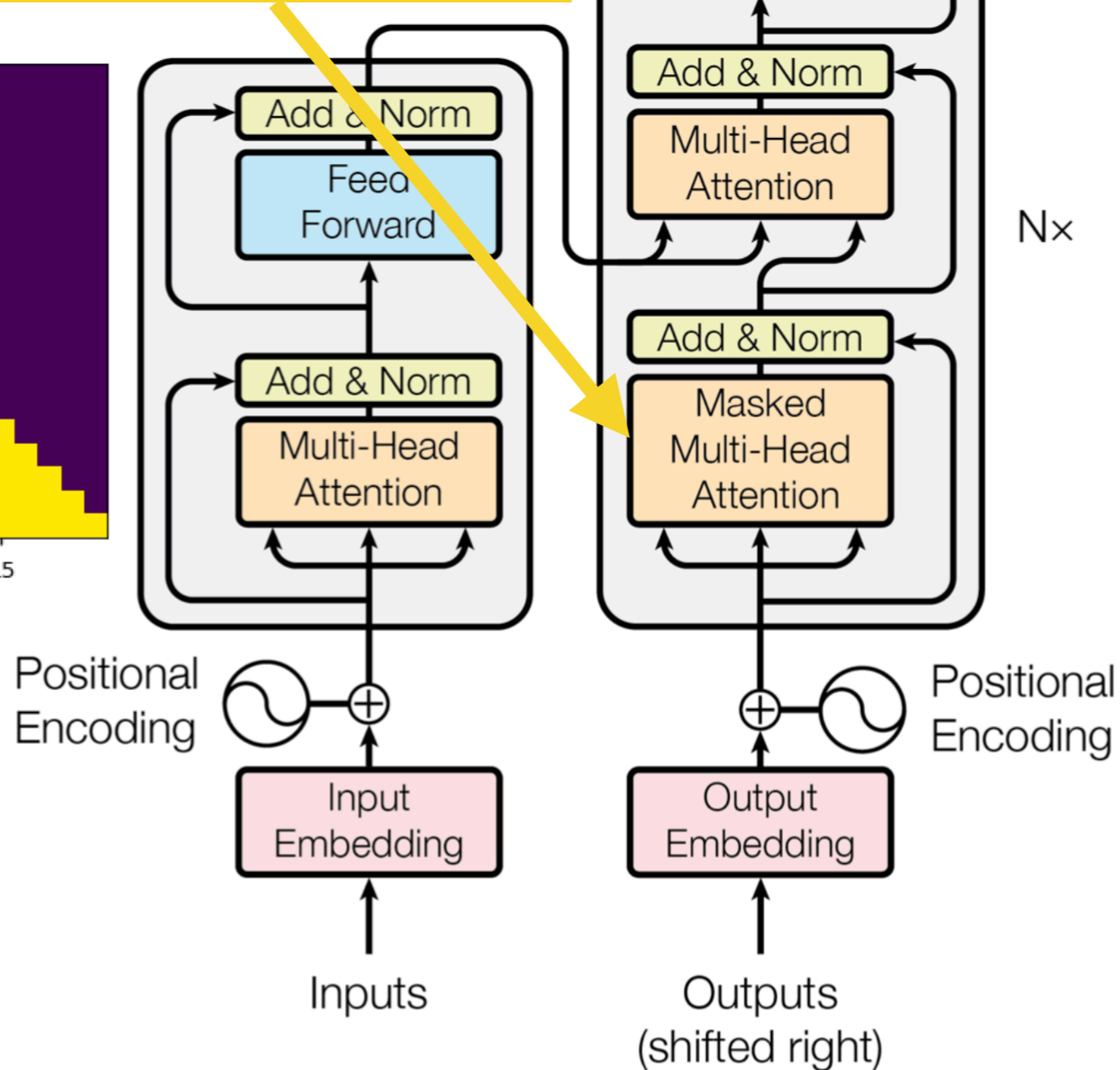
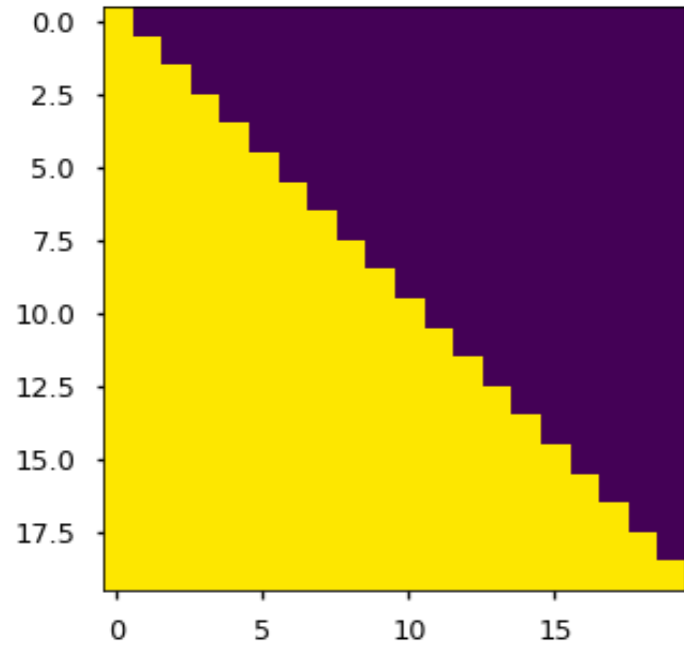
We stack as many of these *Transformer* blocks on top of each other as we can (bigger models are generally better given enough data!)



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

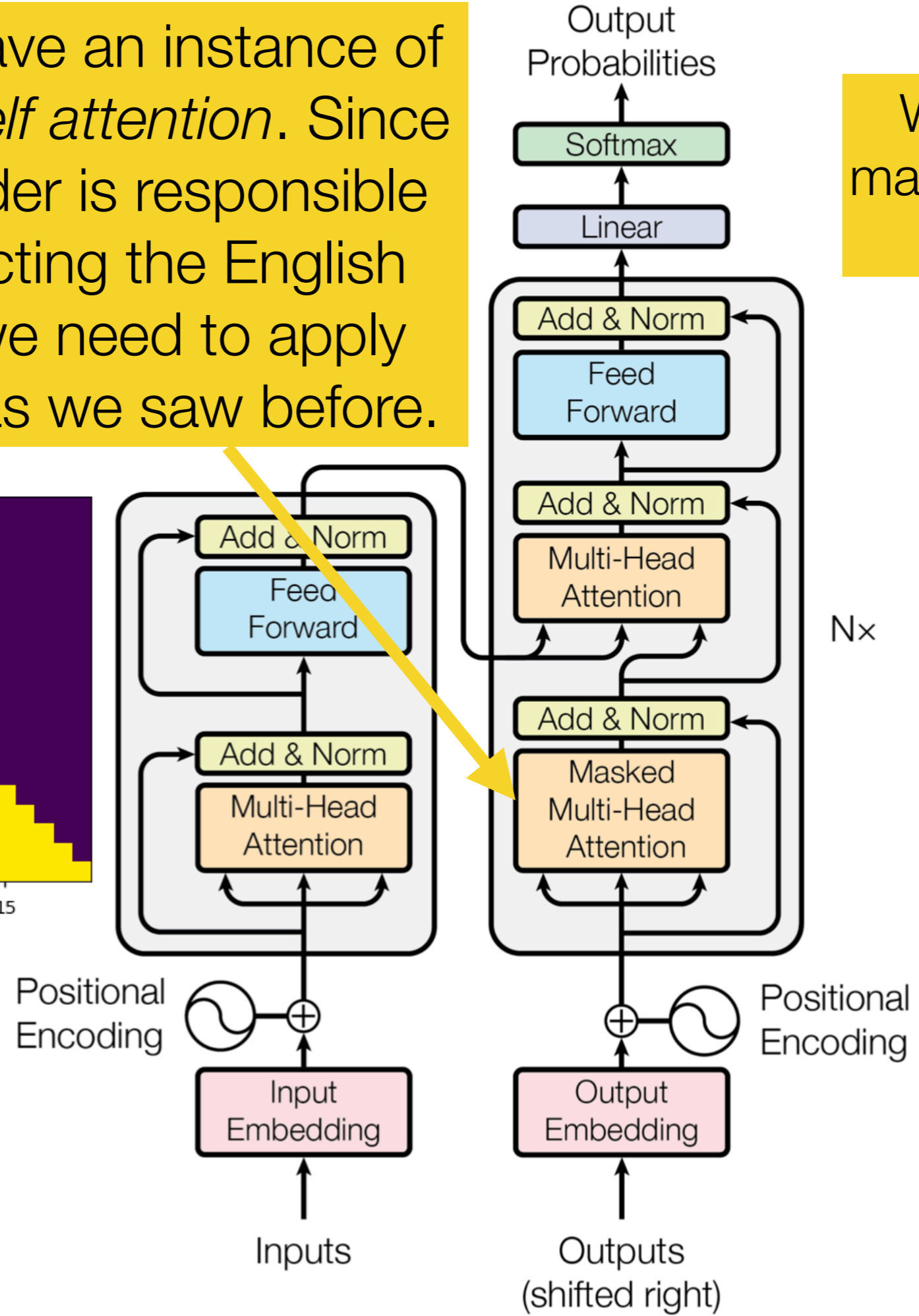
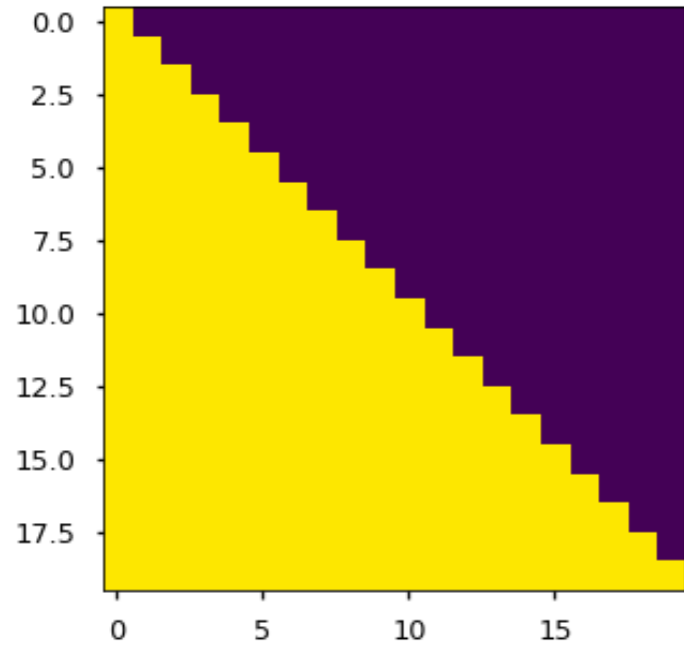


We first have an instance of *masked self attention*. Since the decoder is responsible for predicting the English words, we need to apply masking as we saw before.



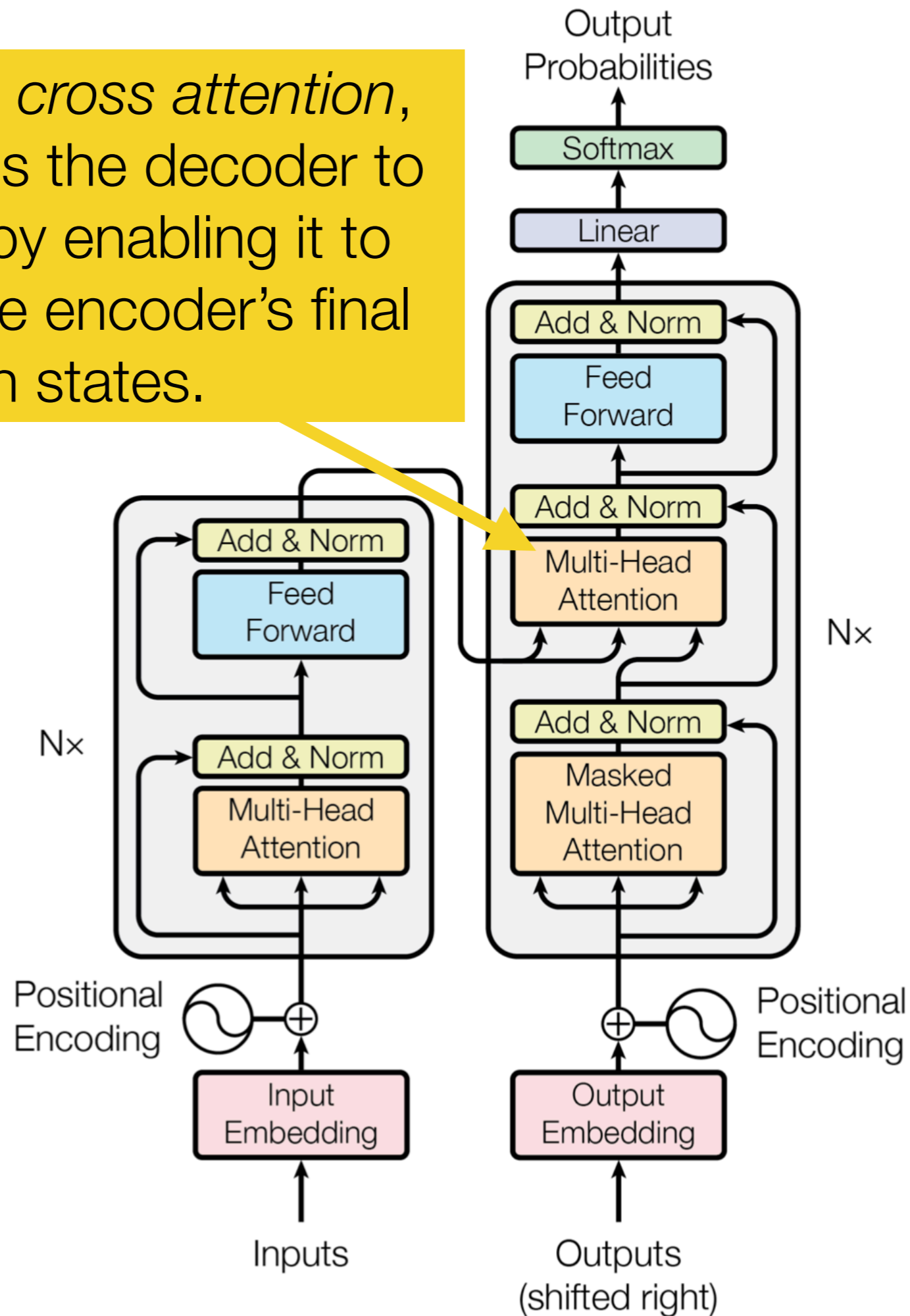
We first have an instance of *masked self attention*. Since the decoder is responsible for predicting the English words, we need to apply masking as we saw before.

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

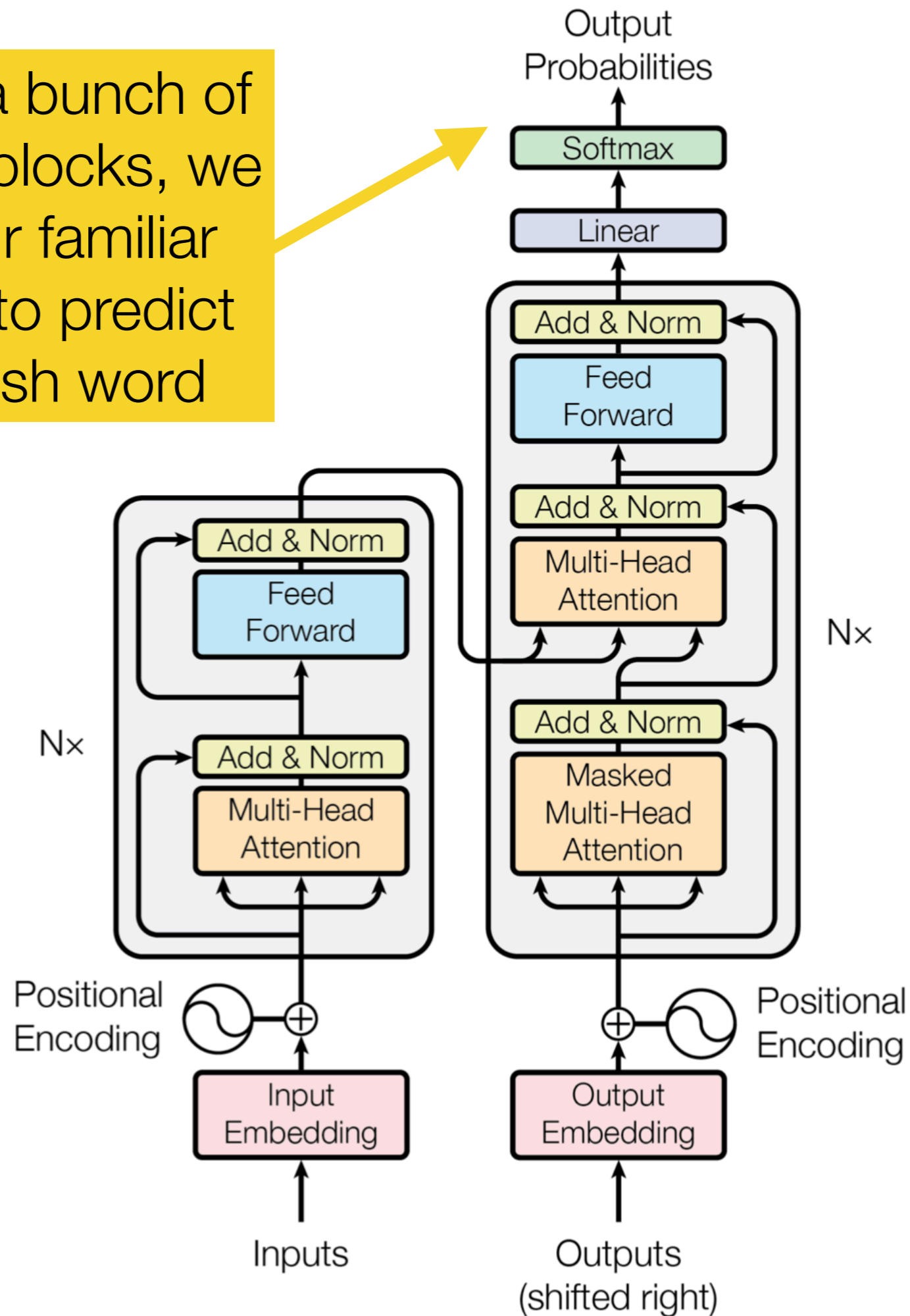




Now, we have *cross attention*, which connects the decoder to the encoder by enabling it to attend over the encoder's final hidden states.

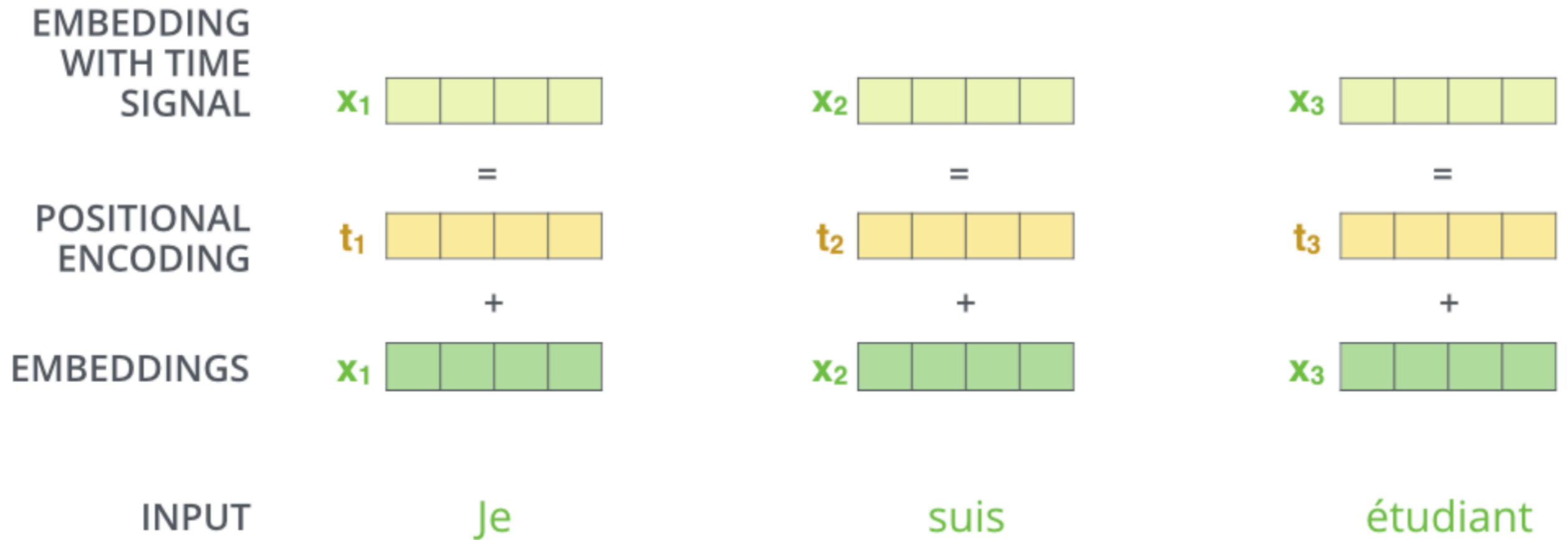


After stacking a bunch of these decoder blocks, we finally have our familiar Softmax layer to predict the next English word





# 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

# Transformer positional encoding

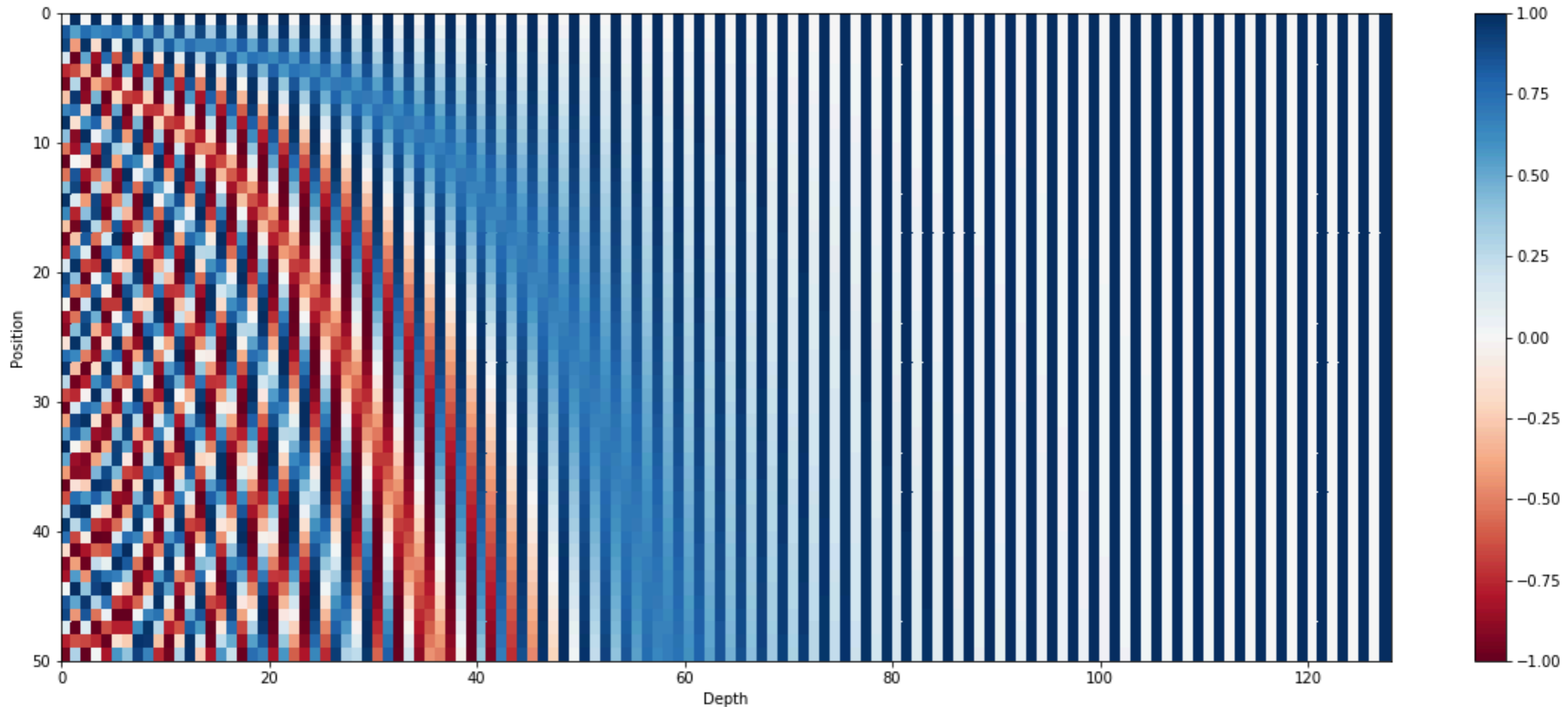
$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{model}}}\right)$$

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

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)*



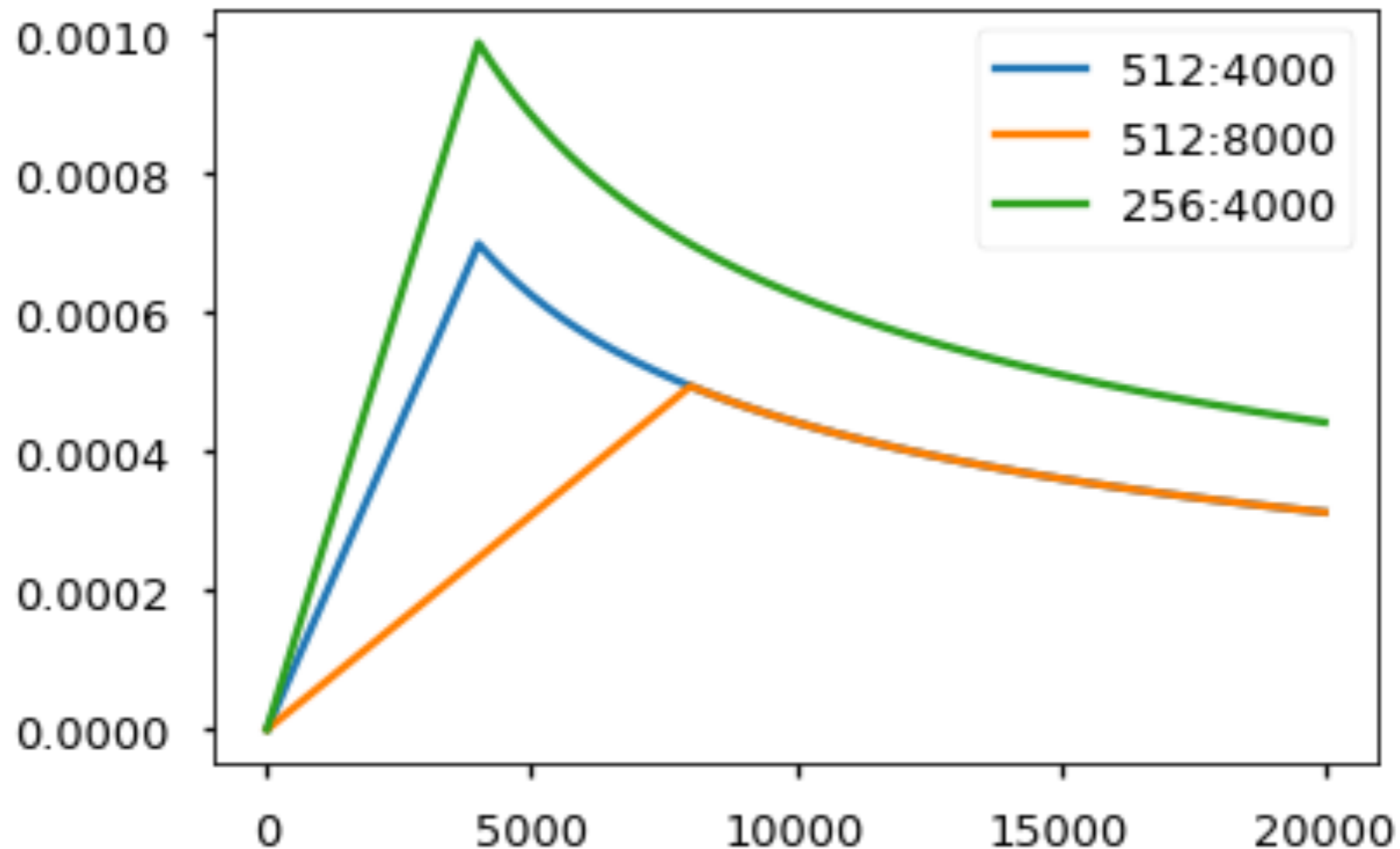
Despite the intuitive flaws, many models these days use *learned positional embeddings* (i.e., they cannot generalize to longer sequences, but this isn't a big deal for their use cases)

# 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 \_\_\_\_\_**

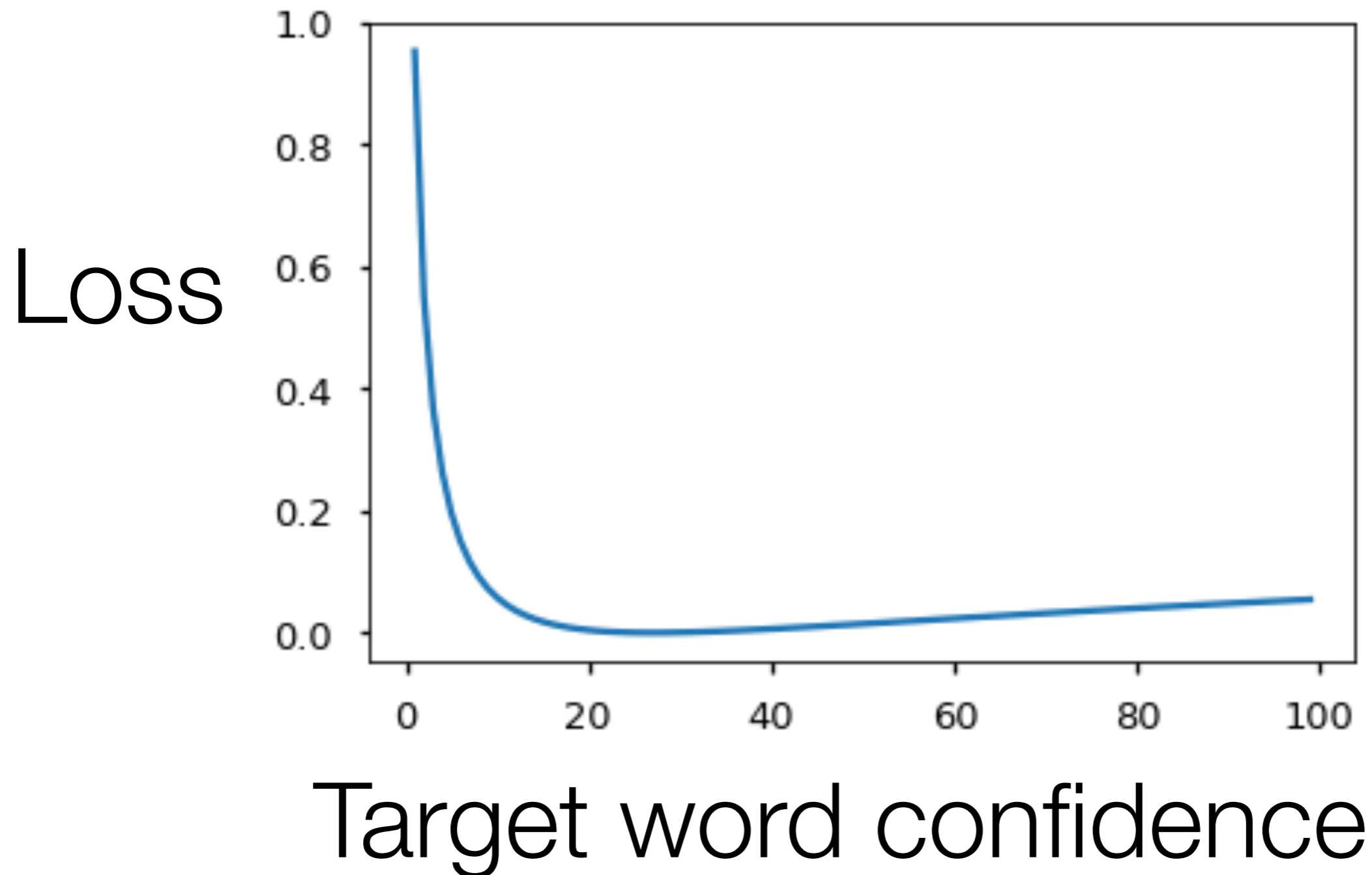
cats    TV    notes    took    sofa

0    0    1    0    0

0.025    0.025    0.9    0.025    0.025

with label smoothing

Get penalized for  
overconfidence!

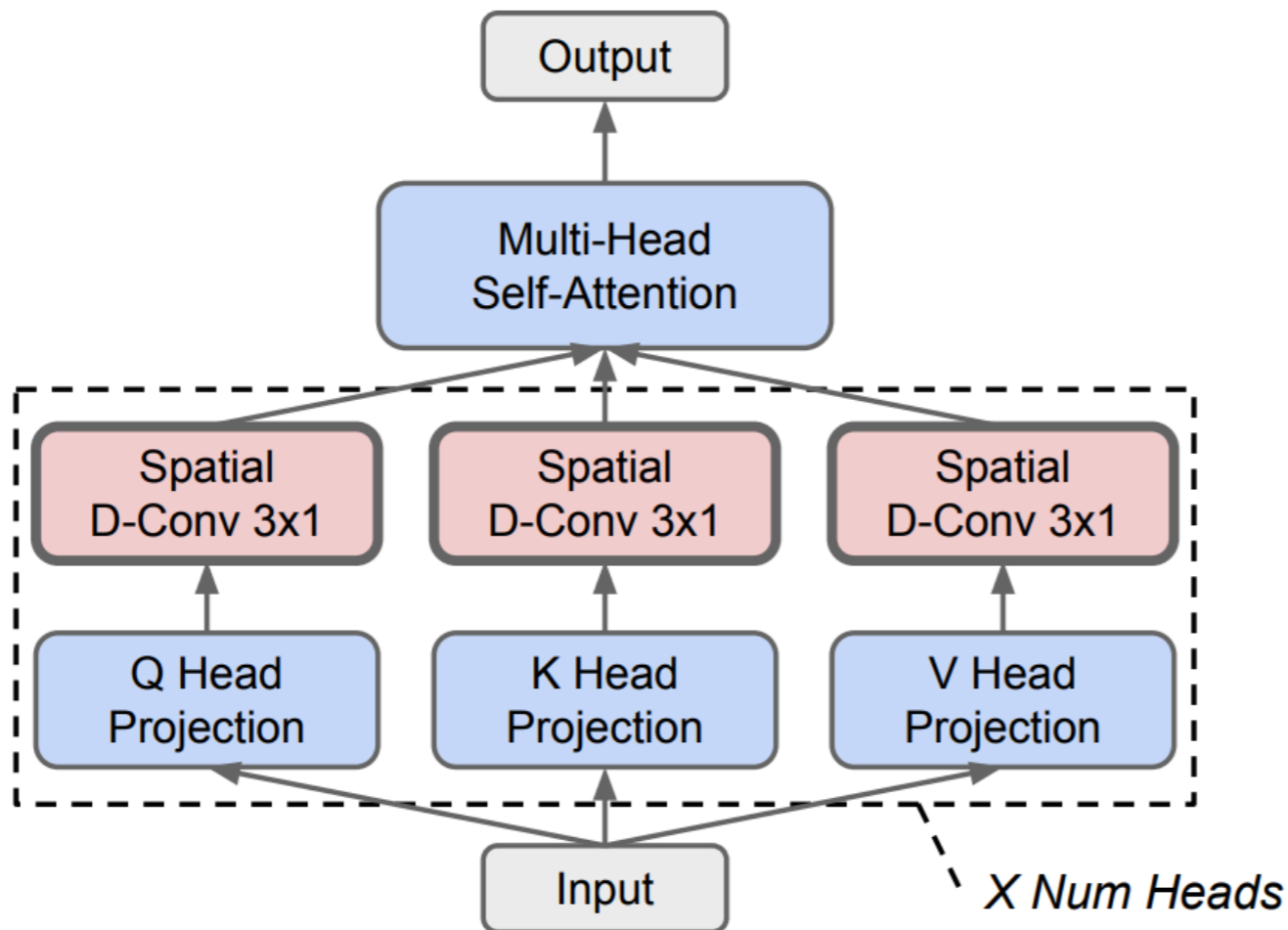


# Why these decisions?

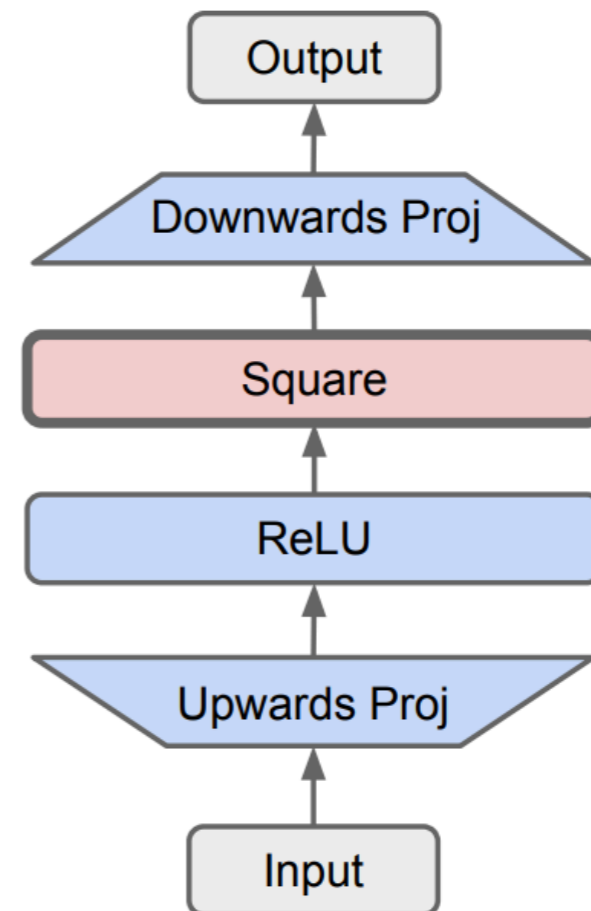
Unsatisfying answer: they empirically worked well.

Neural architecture search finds even better Transformer variants:

Multi-DConv-Head Attention (MDHA)



Squared ReLU in Feed Forward Block



# 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)

# Coming up!

- Transfer learning via Transformer models like BERT
- Tokenization (word vs subword vs character/byte)
- Prompt-based learning
- Efficient / long-range Transformers
- Downstream tasks