CS388: Natural Language Processing
Lecture 7: Transformers

The University of Texas at Austin

Slides credit: Greg Durrett

Administration

- HW2 will be out on Thursday

Recap: Attention

Step 1: Compute scores for each key

keys k_i

[1, 0] [1, 0] [0, 1] [1, 0]

0 0 1 0

query: q = [0, 1] (we want to find 1s)

$s_i = k_i^T q$

0 0 1 0

Step 2: softmax the scores to get probabilities α

0 0 1 0 => (1/6, 1/6, 1/2, 1/6) if we assume e=3

Step 3: compute output values by multiplying embs. by alpha + summing

result = $\text{sum}(\alpha_i e_i)$ = 1/6 [1, 0] + 1/6 [1, 0] + 1/2 [0, 1] + 1/6 [1, 0] = [1/2, 1/2]

Recap: Self-Attention

$$E = \begin{pmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 1 & 0 \end{pmatrix}$$

$$W^Q = \begin{matrix} 0 & 1 \\ 0 & 1 \end{matrix}$$

$$W^K = \begin{matrix} 10 & 0 \\ 0 & 10 \end{matrix}$$

$$Q = E\,(W^Q) = \begin{pmatrix} 0 & 1 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \end{pmatrix}$$

$$K = E\,(W^K) = \begin{pmatrix} 10 & 0 \\ 10 & 0 \\ 0 & 10 \\ 10 & 0 \end{pmatrix}$$

Scores $S = QK^T$ $\quad S_{ij} = q_i \cdot k_j$

len x len = (len x d) x (d x len)

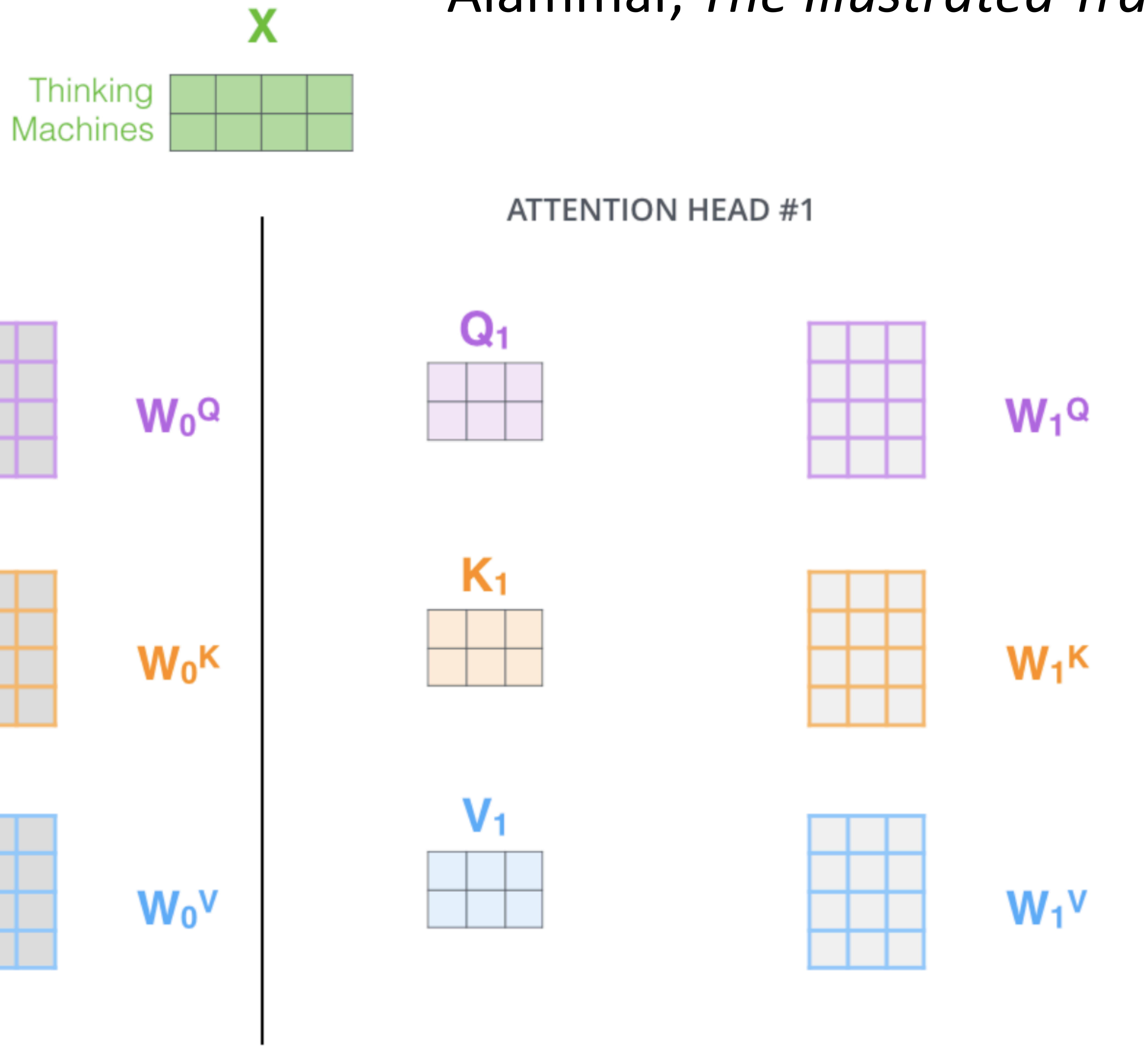
Final step: softmax to get attentions A, then output is AE

*technically it's A (EW^V), using a values matrix V = EW^V

Recap: Multi-head Self-Attention

Just duplicate the whole computation with different weights:

Alammar, *The Illustrated Transformer*



Transformers

Architecture

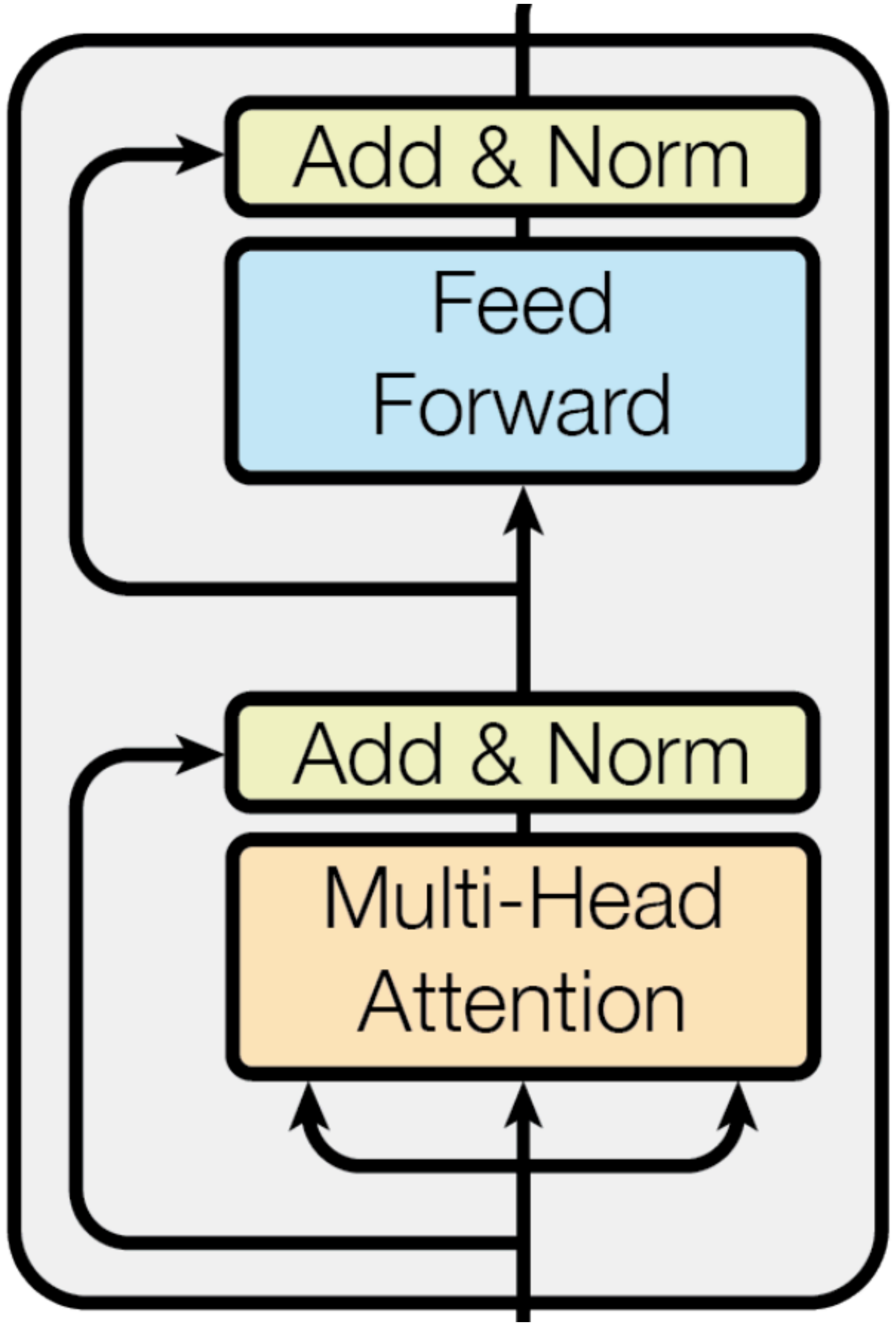
- Alternate multi-head self-attention with feedforward layers that **operate over each word individually**

ReLU

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

  - These feedforward layers are where most of the parameters are

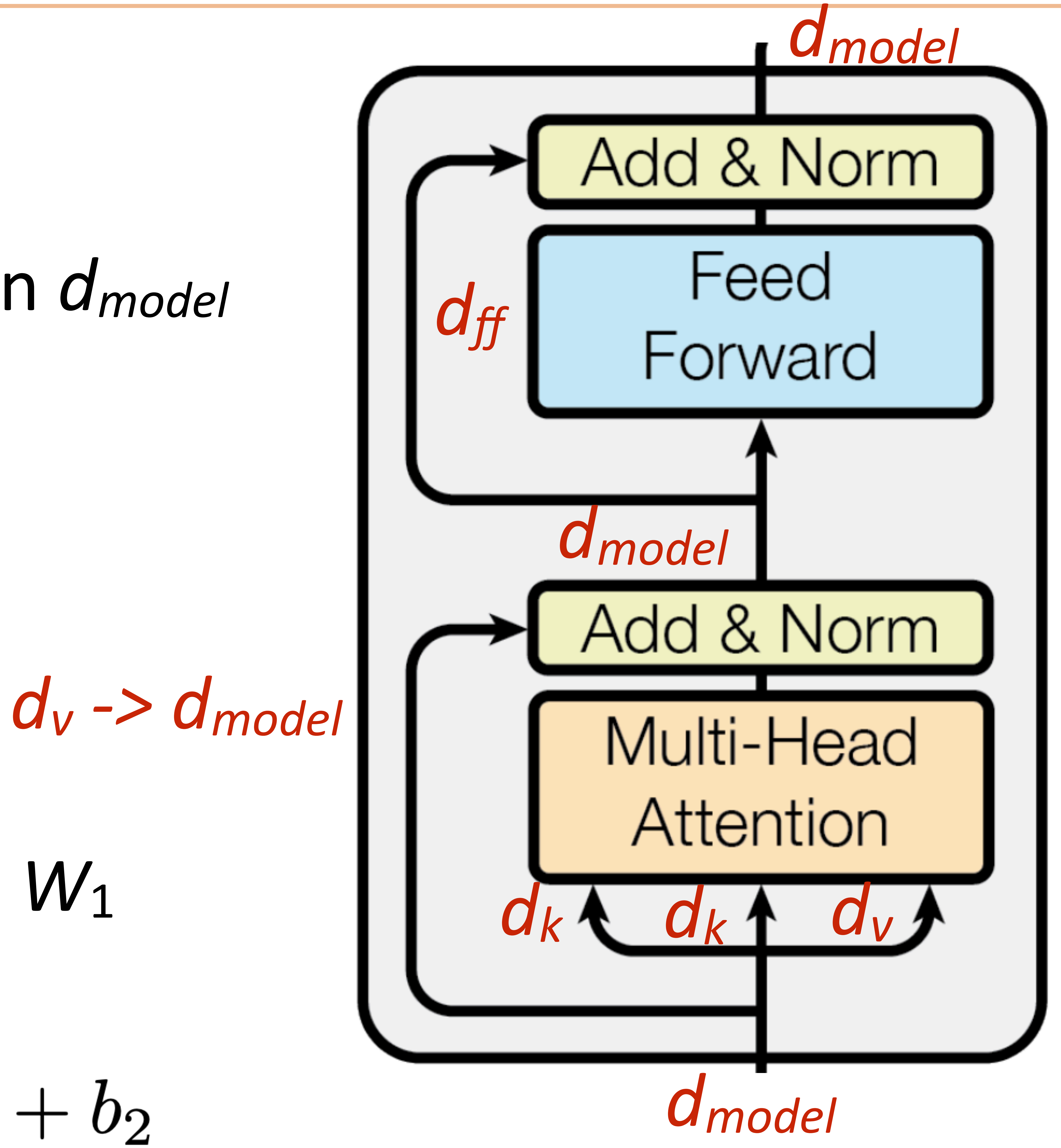
- Residual connections in the model: input of a layer is added to its output
- Layer normalization: controls the scale of different layers in very deep networks (not needed in the assignment)



Dimensions

- Vectors: d_{model}
- Queries/keys: d_k , always smaller than d_{model}
- Values: separate dimension d_v , output is multiplied by W^O which is d_v x d_{model} so we can get back to d_{model} before the residual
- FFN can explode the dimension with W_1 and collapse it back with W_2

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



Vaswani et al. (2017)

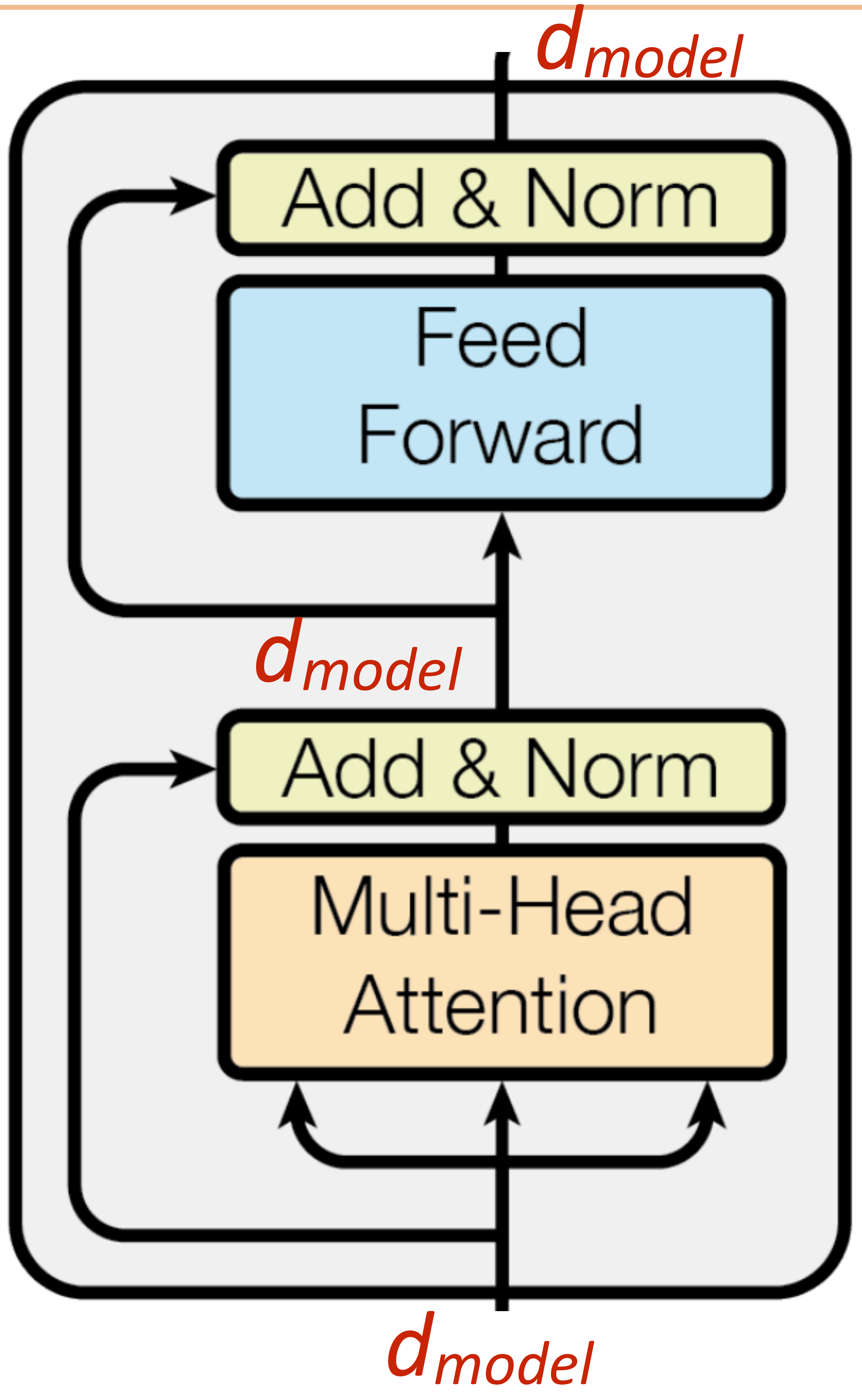
Transformer Architecture

	N	d_{model}	d_{ff}	h	d_k	d_v
base	6	512	2048	8	64	64

- From Vaswani et al.

Model Name	n_{params}	n_{layers}	d_{model}	n_{heads}	d_{head}
GPT-3 Small	125M	12	768	12	64
GPT-3 Medium	350M	24	1024	16	64
GPT-3 Large	760M	24	1536	16	96
GPT-3 XL	1.3B	24	2048	24	128
GPT-3 2.7B	2.7B	32	2560	32	80
GPT-3 6.7B	6.7B	32	4096	32	128
GPT-3 13B	13.0B	40	5140	40	128
GPT-3 175B or "GPT-3"	175.0B	96	12288	96	128

- From GPT-3; d_{head} is our d_k

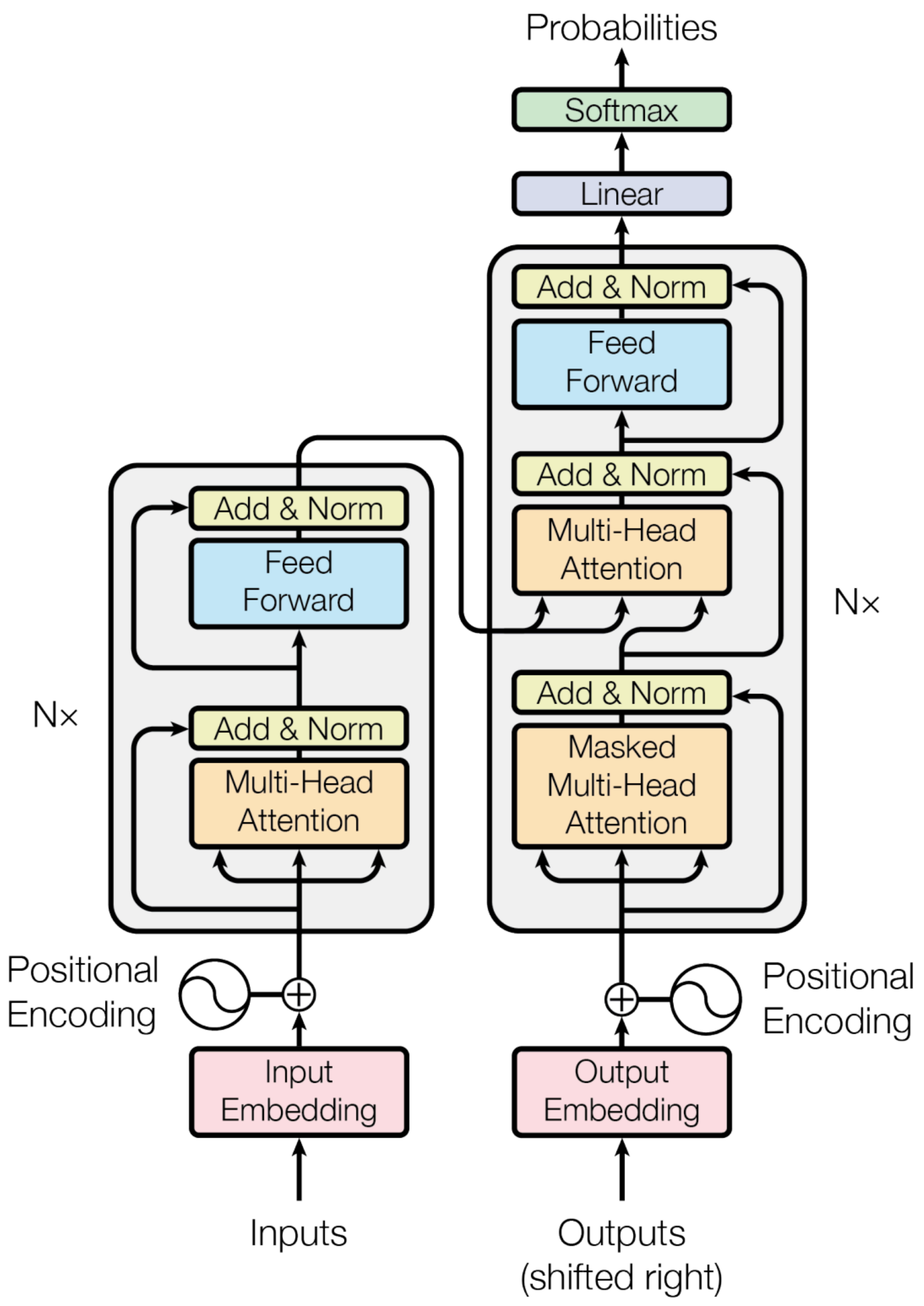


Transformer Architecture

1	description	FLOPs / update	% FLOPS MHA	% FLOPS FFN	% FLOPS attn	% FLOPS logit
8	OPT setups					
9	760M	4.3E+15	35%	44%	14.8%	5.8%
10	1.3B	1.3E+16	32%	51%	12.7%	5.0%
11	2.7B	2.5E+16	29%	56%	11.2%	3.3%
12	6.7B	1.1E+17	24%	65%	8.1%	2.4%
13	13B	4.1E+17	22%	69%	6.9%	1.6%
14	30B	9.0E+17	20%	74%	5.3%	1.0%
15	66B	9.5E+17	18%	77%	4.3%	0.6%
16	175B	2.4E+18	17%	80%	3.3%	0.3%

Credit: Stephen Roller on Twitter

Transformers: Complete Model

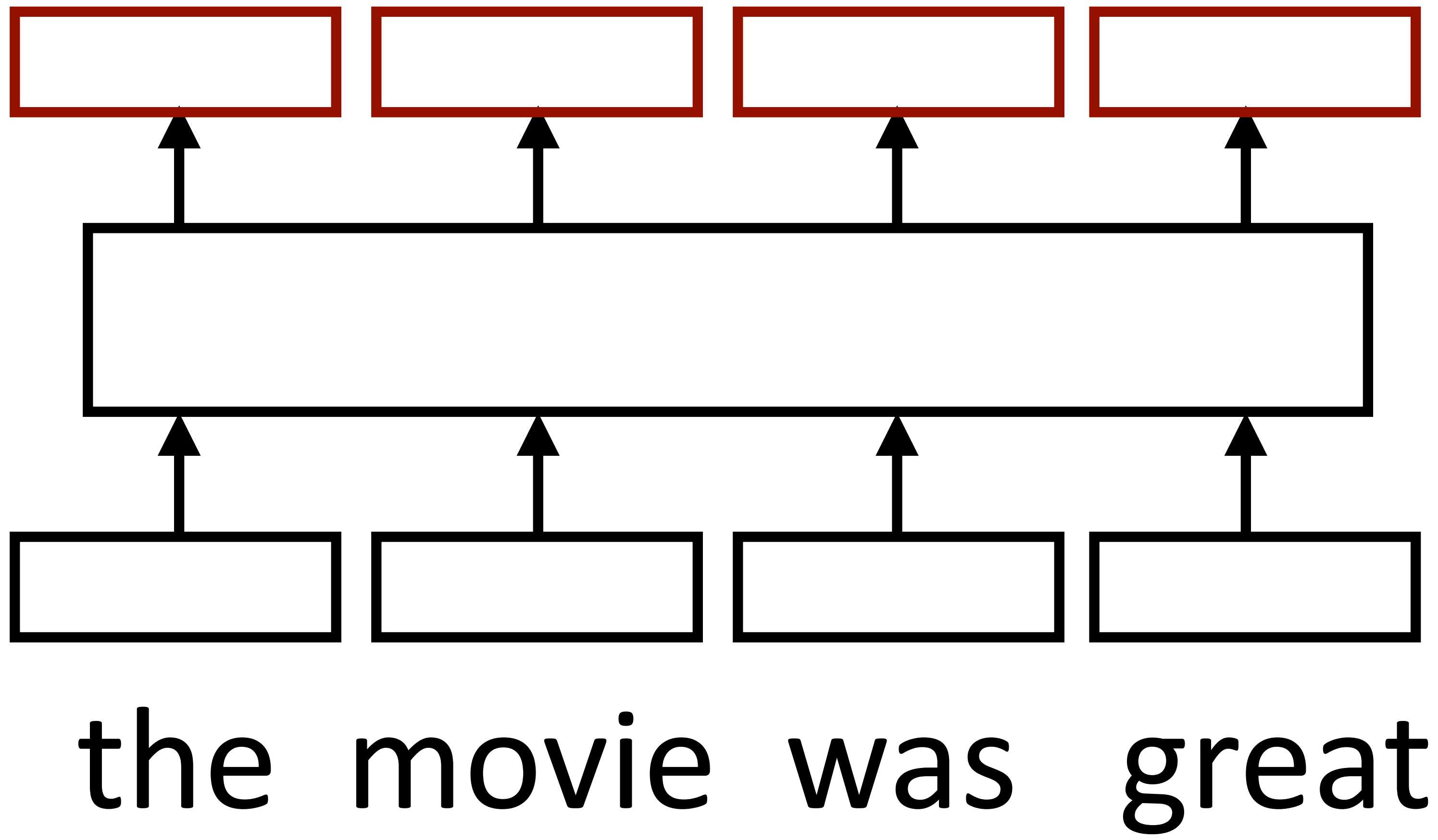


- Original Transformer paper presents an **encoder-decoder** model
- Right now we don't need to think about both of these parts — will return in the context of MT
- Can turn the encoder into a decoder-only model through use of a triangular causal attention mask (only allow attention to previous tokens)

Vaswani et al. (2017)

Using Transformers

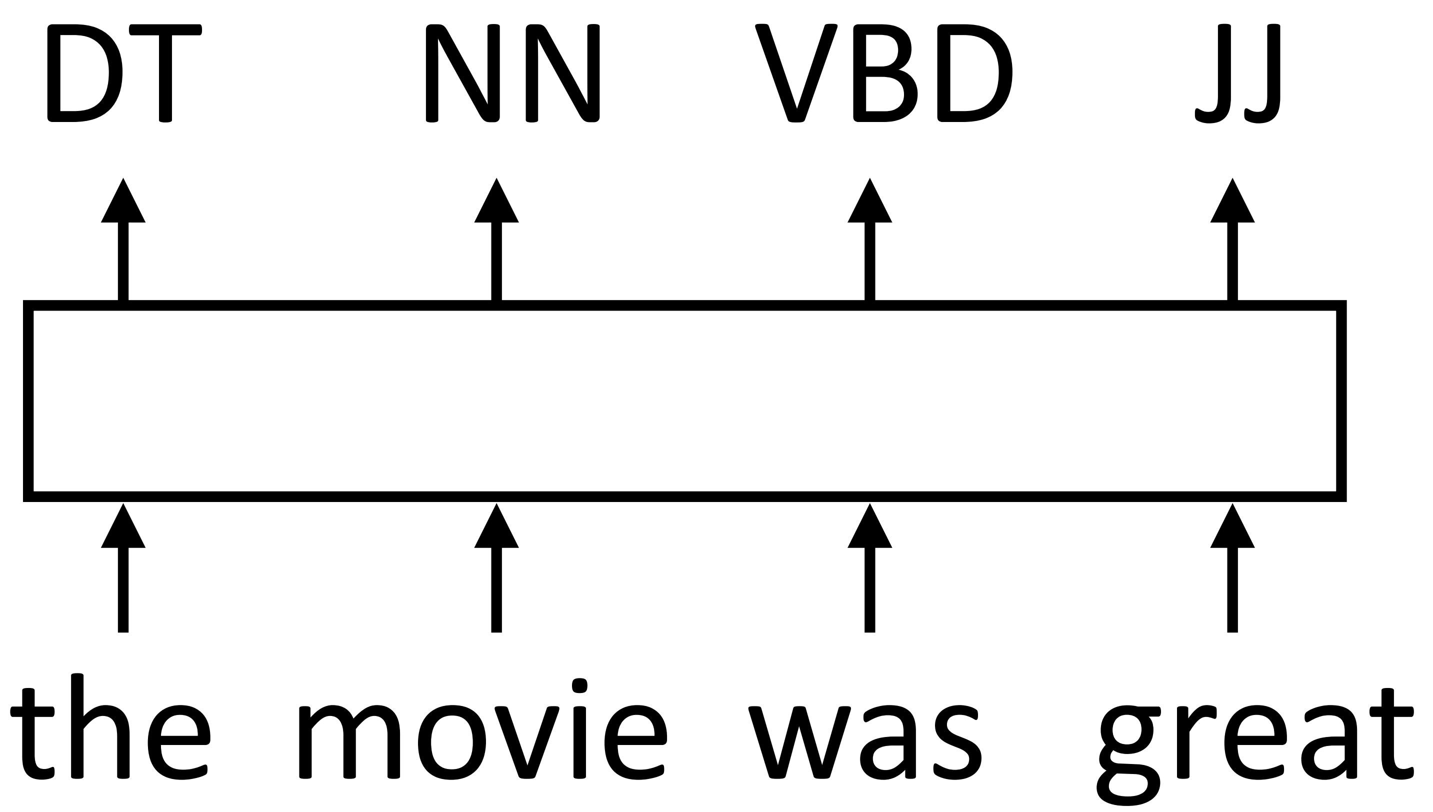
What do Transformers produce?



- Encoding of each word — can pass this to another layer to make a prediction (like predicting the next word for language modeling)
- Like RNNs, Transformers can be viewed as a transformation of a sequence of vectors into a sequence of context-dependent vectors

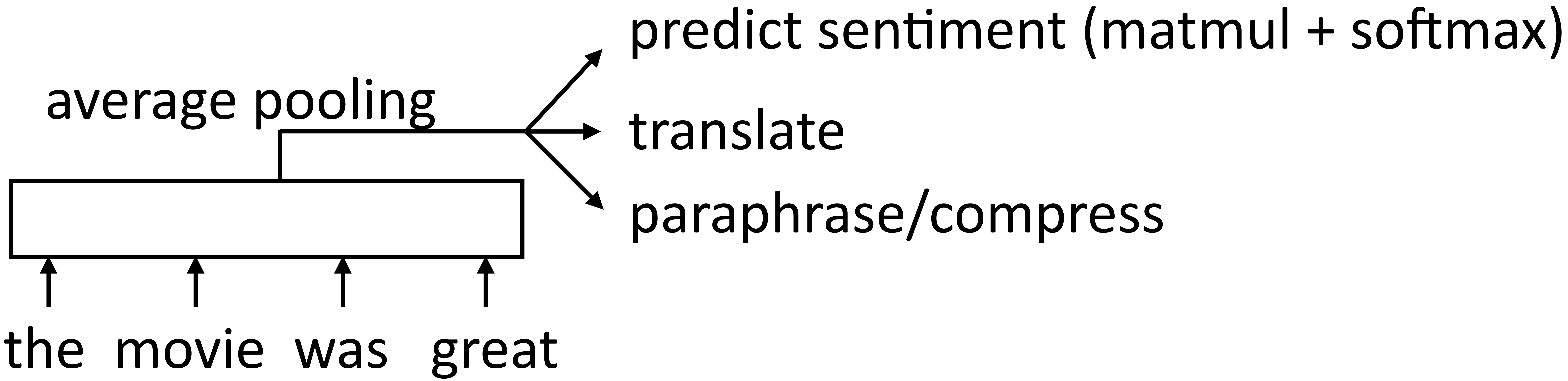
Transformer Uses

- Transducer: make some prediction for each element in a sequence

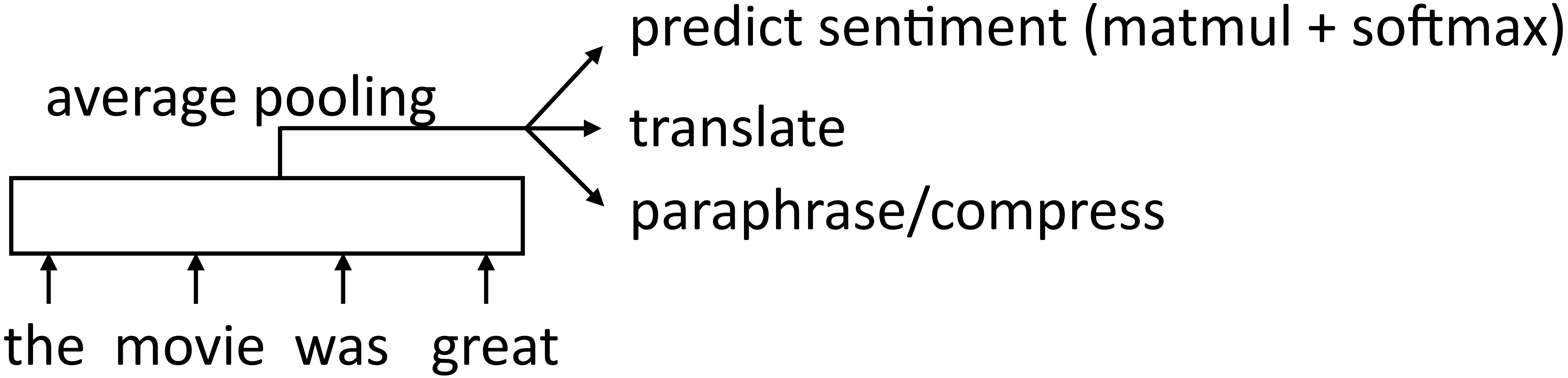


output **y** = score for each tag, then softmax

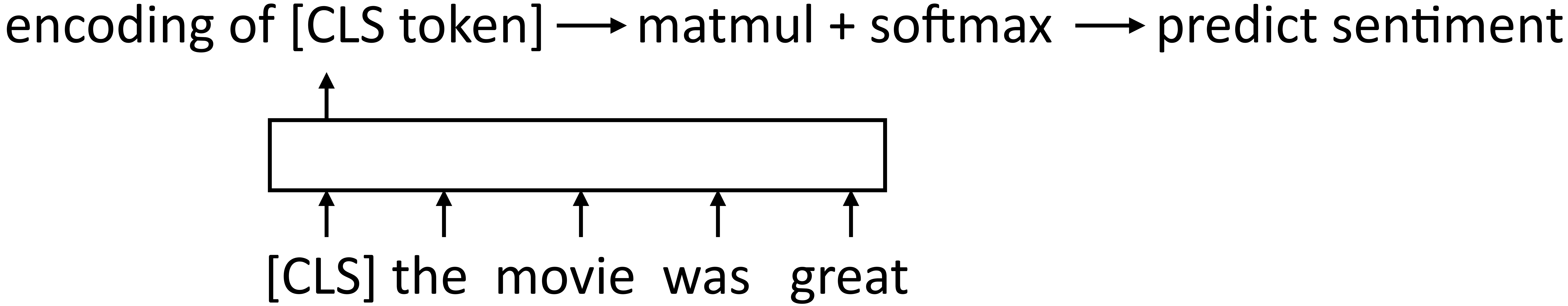
- Classifier: encode a sequence into a fixed-sized vector and classify that



Transformer Uses

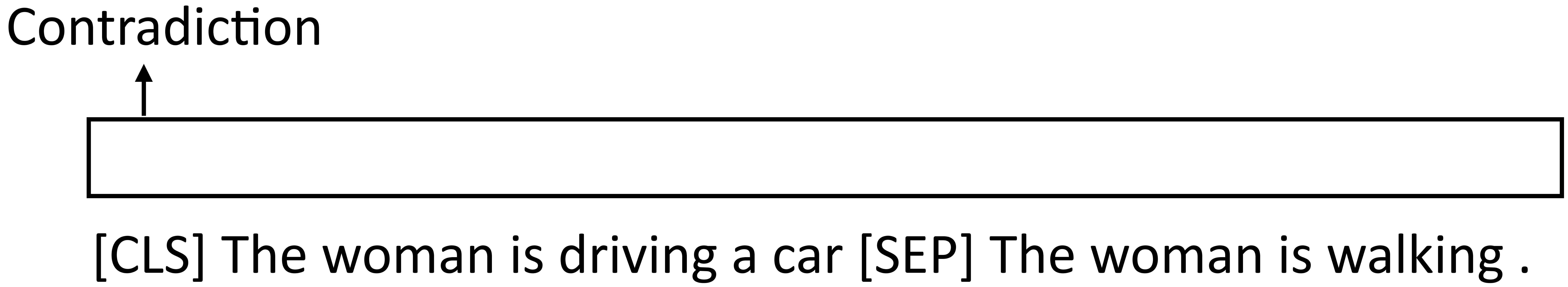


- Alternative: use a placeholder [CLS] token at the start of the sequence. Because [CLS] attends to everything with self-attention, it can do the pooling for you!



Transformer Uses

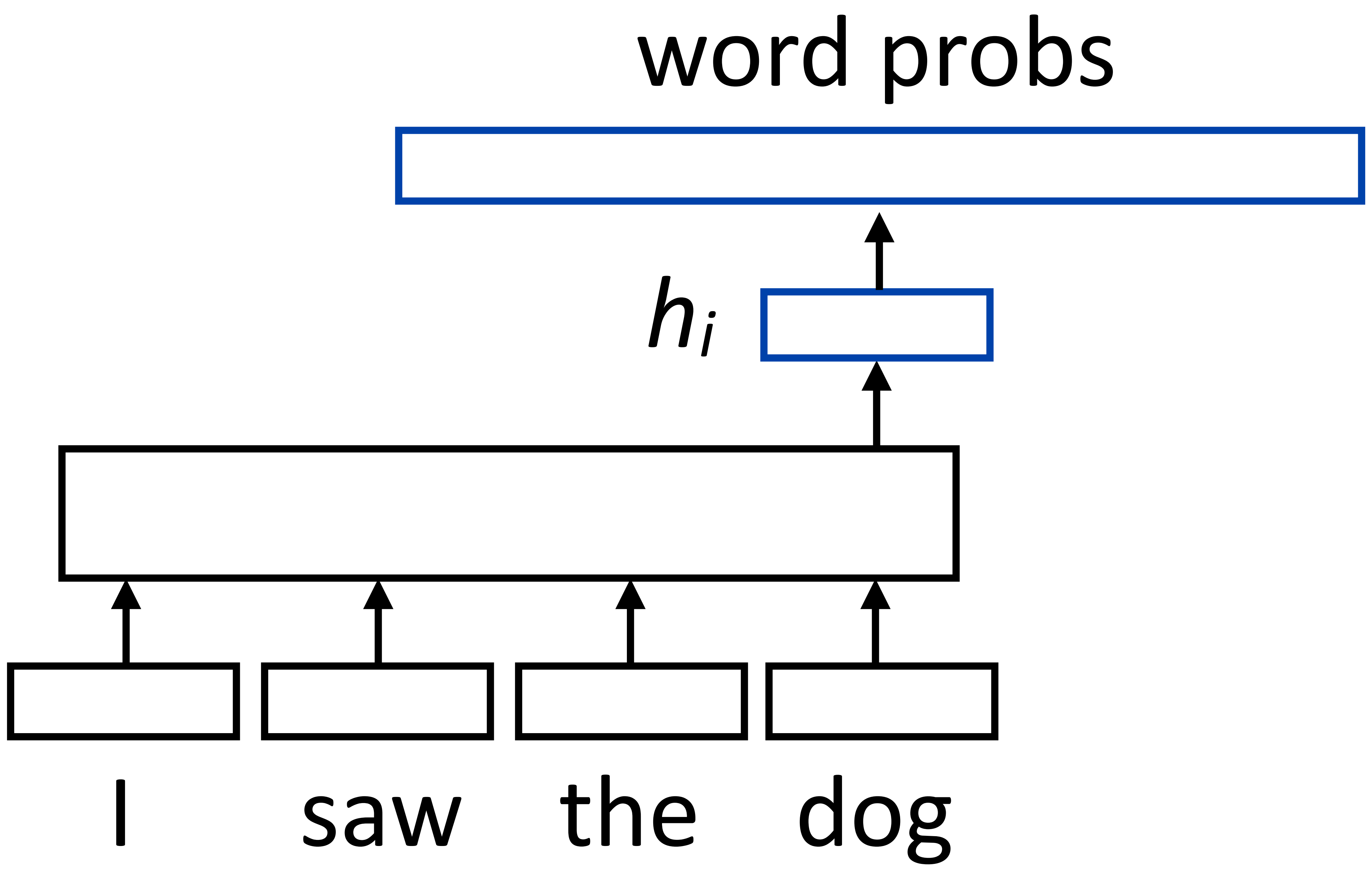
- Sentence **pair** classifier: feed in two sentences and classify something about their relationship



- Why might Transformers be particularly good at sentence **pair** tasks?

Transformer Language Modeling

Transformer Language Modeling



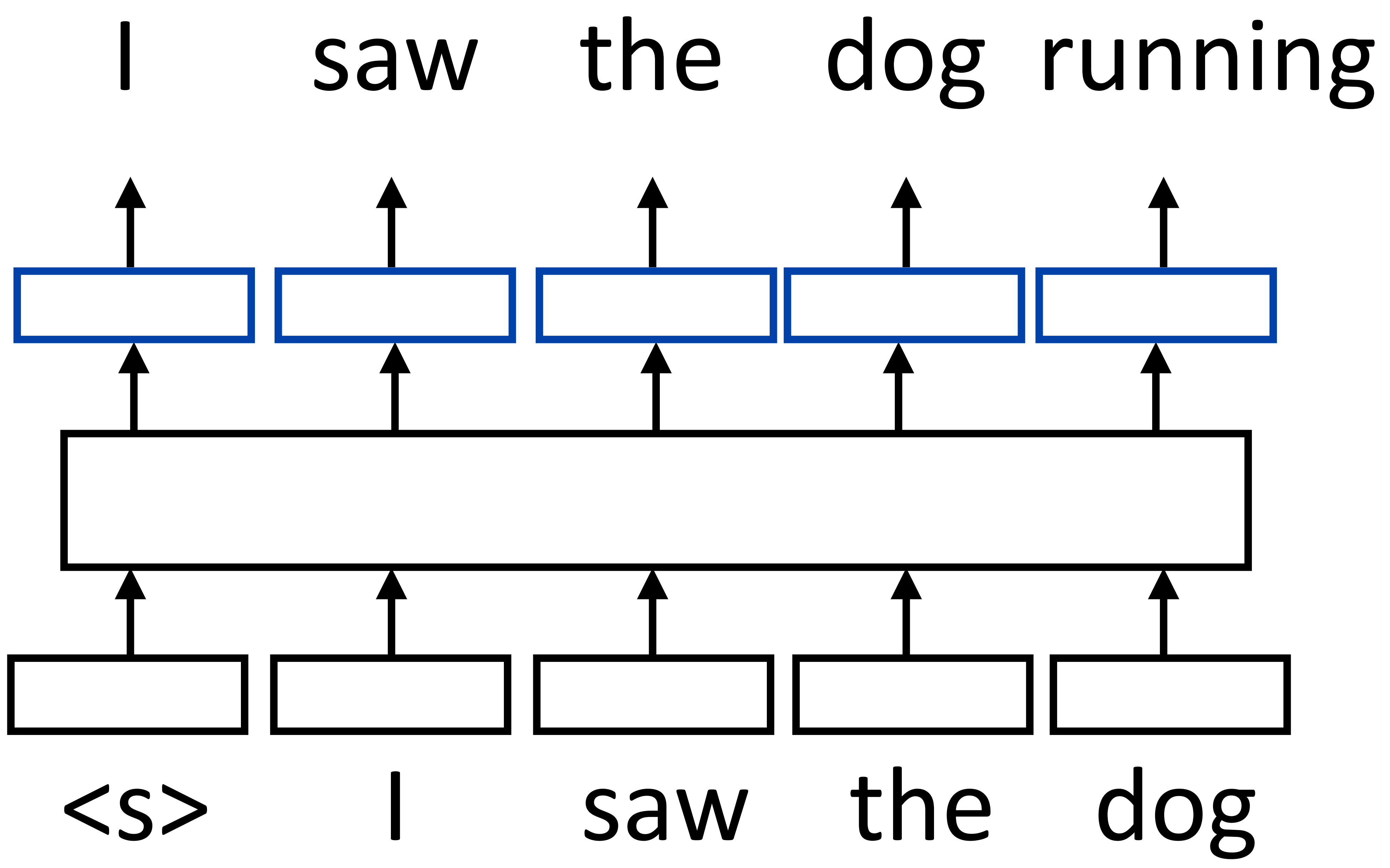
$$P(w|\text{context}) = \frac{\exp(\mathbf{w} \cdot \mathbf{h_i})}{\sum_{w'} \exp(\mathbf{w'} \cdot \mathbf{h_i})}$$

equivalent to

$$P(w|\text{context}) = \text{softmax}(W\mathbf{h}_i)$$

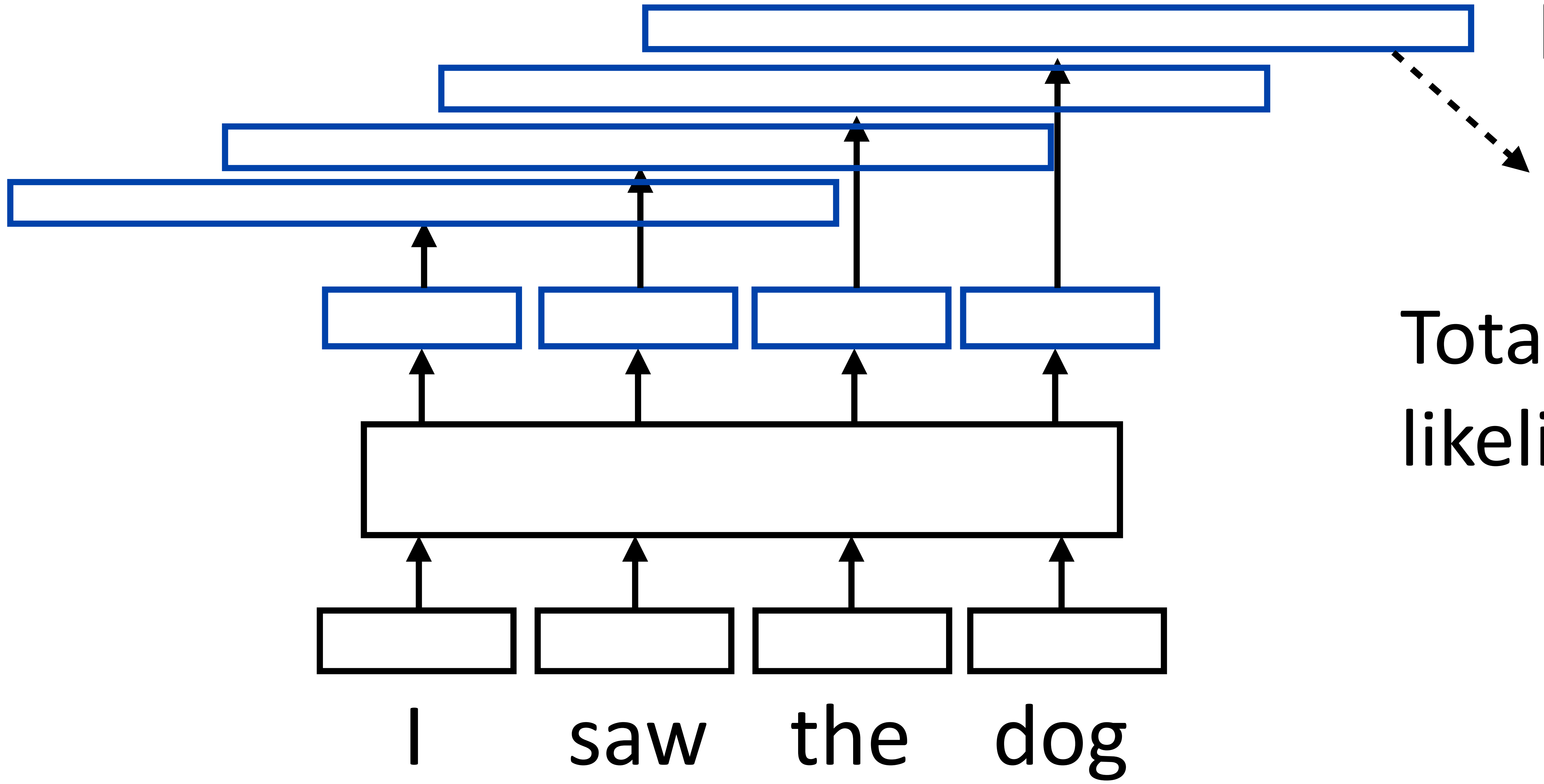
- *W* is a (vocab size) x (hidden size) matrix; linear layer in PyTorch (rows are word embeddings)

Training Transformer LMs



- Input is a sequence of words, output is those words shifted by one,
- Allows us to train on predictions across several timesteps simultaneously (similar to batching but this is NOT what we refer to as batching)

Training Transformer LMs



P(w|context)

loss = — log P(w*|context)

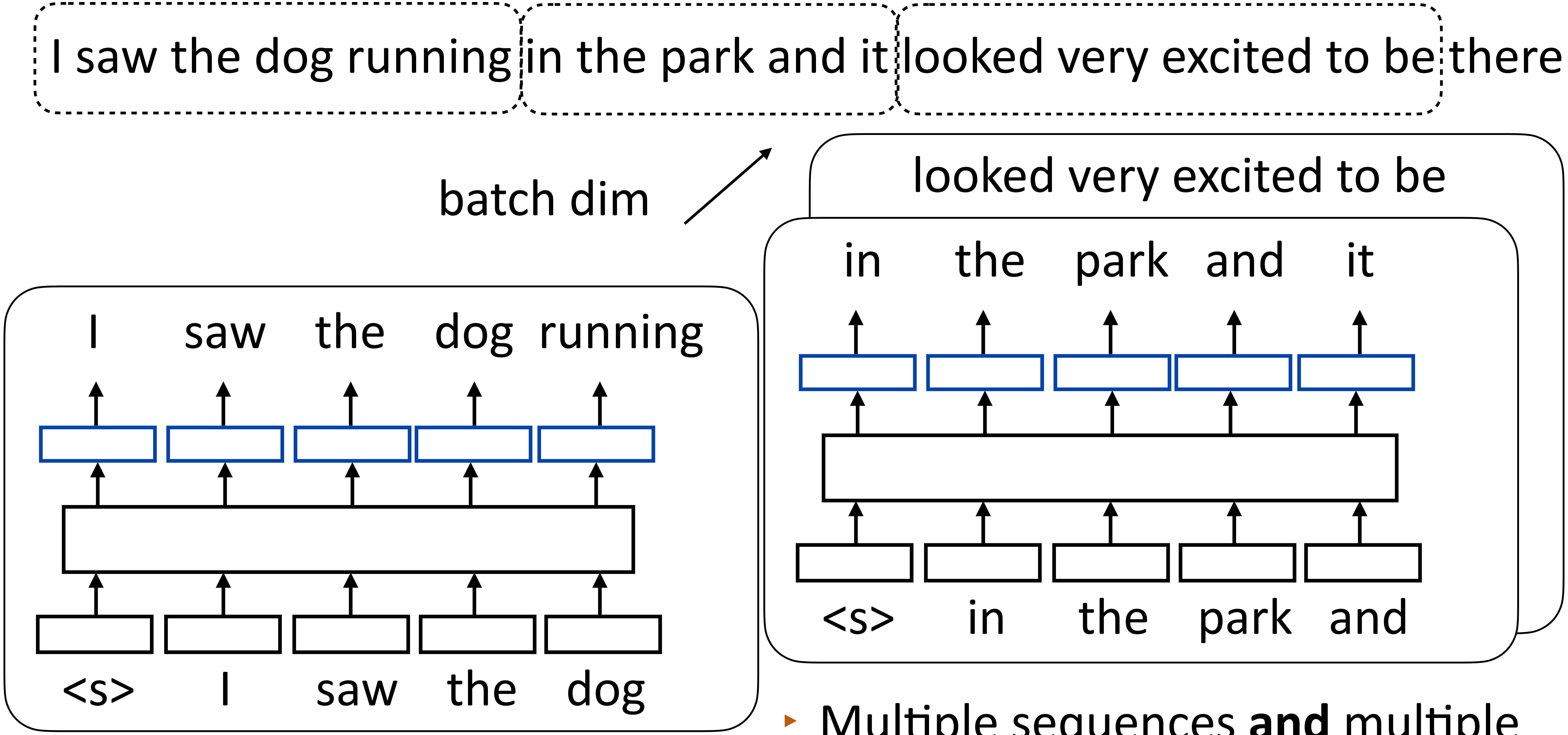
Total loss = sum of negative log likelihoods at each position

```
loss_fcn = nn.NLLLoss()
loss += loss_fcn(log_probs, ex.output_tensor)
```

[seq len, num output classes] [seq len]

- Batching is a little tricky with NLLLoss: need to collase [batch, seq len, num classes] to [batch * seq len, num classes].

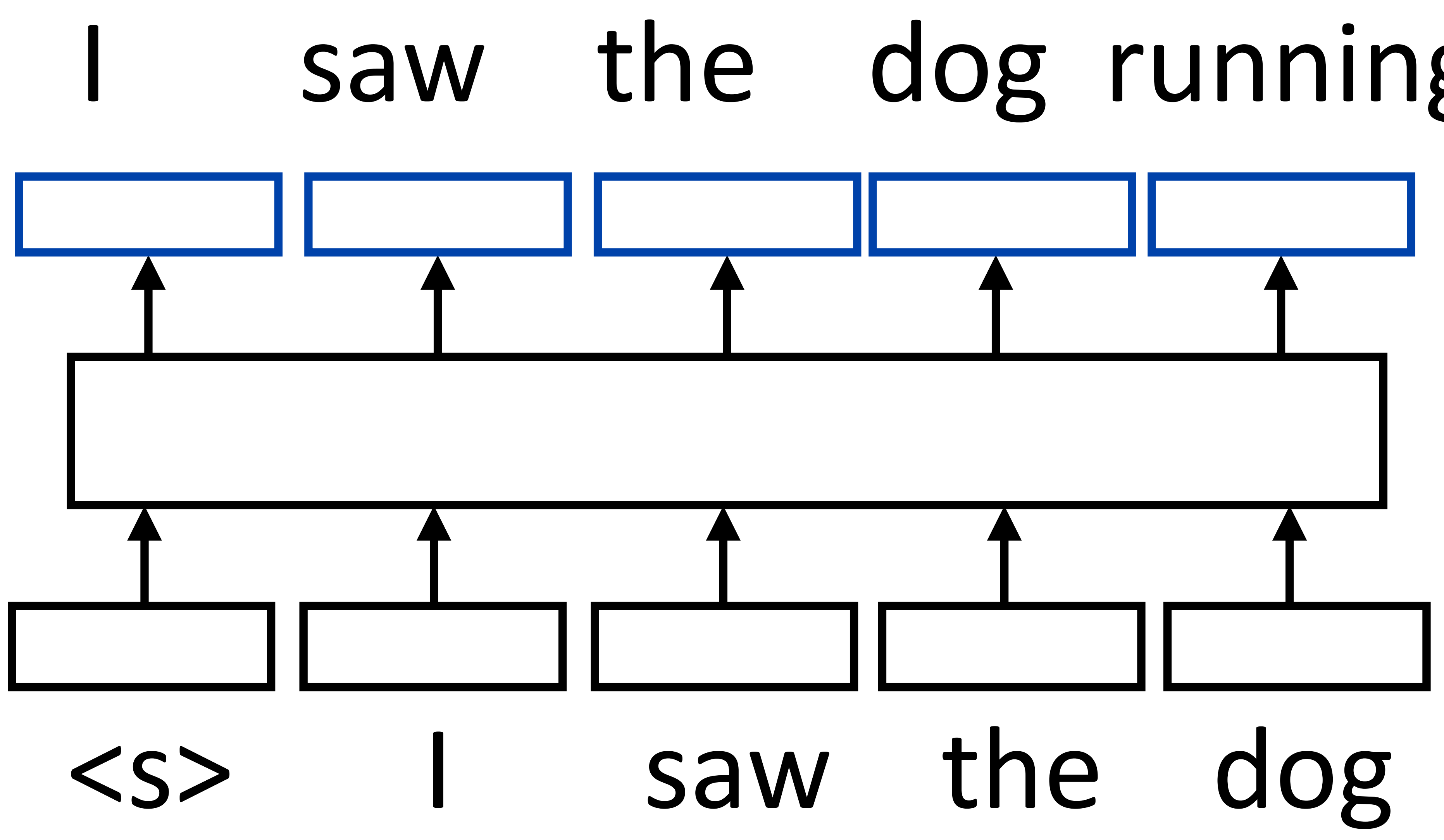
Batched LM Training



- Multiple sequences **and** multiple timesteps per sequence

A Small Problem with Transformer LMs

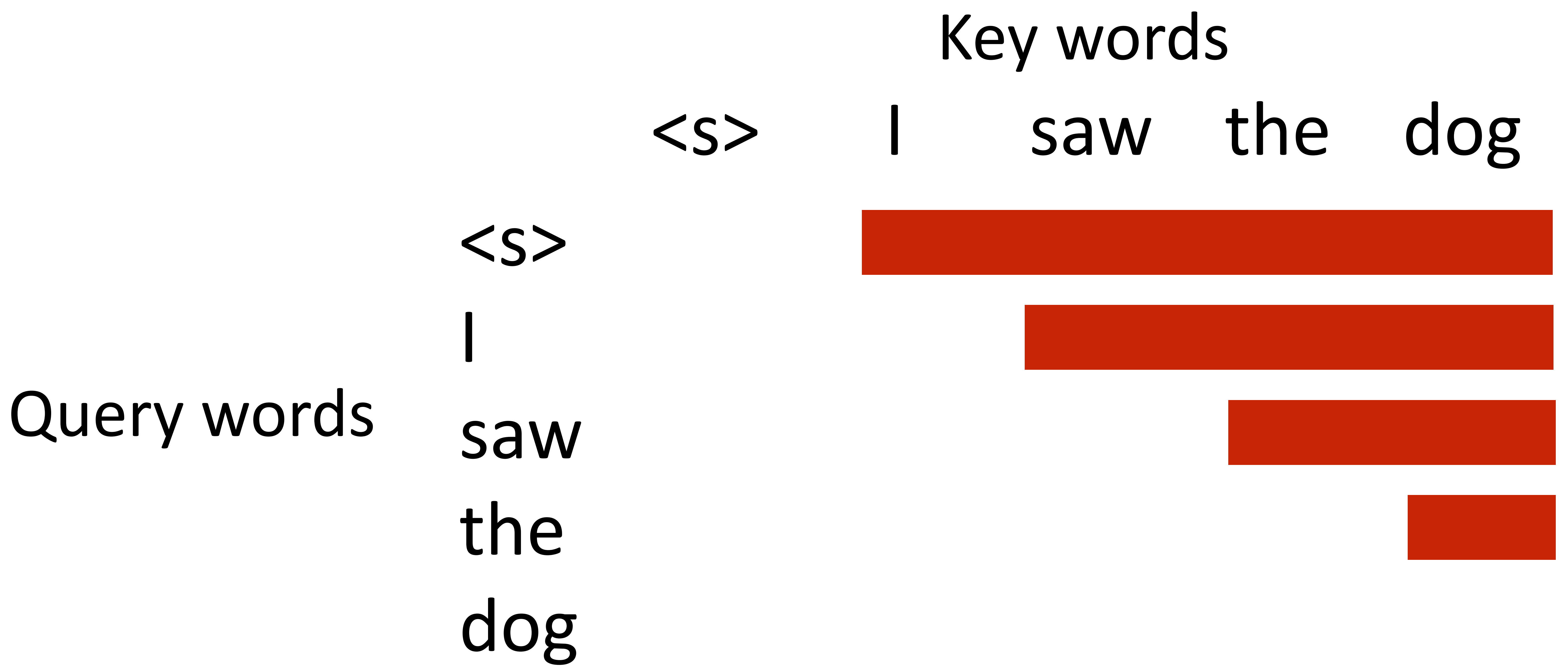
- This Transformer LM as we've described it will *easily* achieve perfect accuracy. Why?



- With standard self-attention: "I" attends to "saw" and the model is "cheating". How do we ensure that this doesn't happen?

Attention Masking

- What do we want to prohibit?

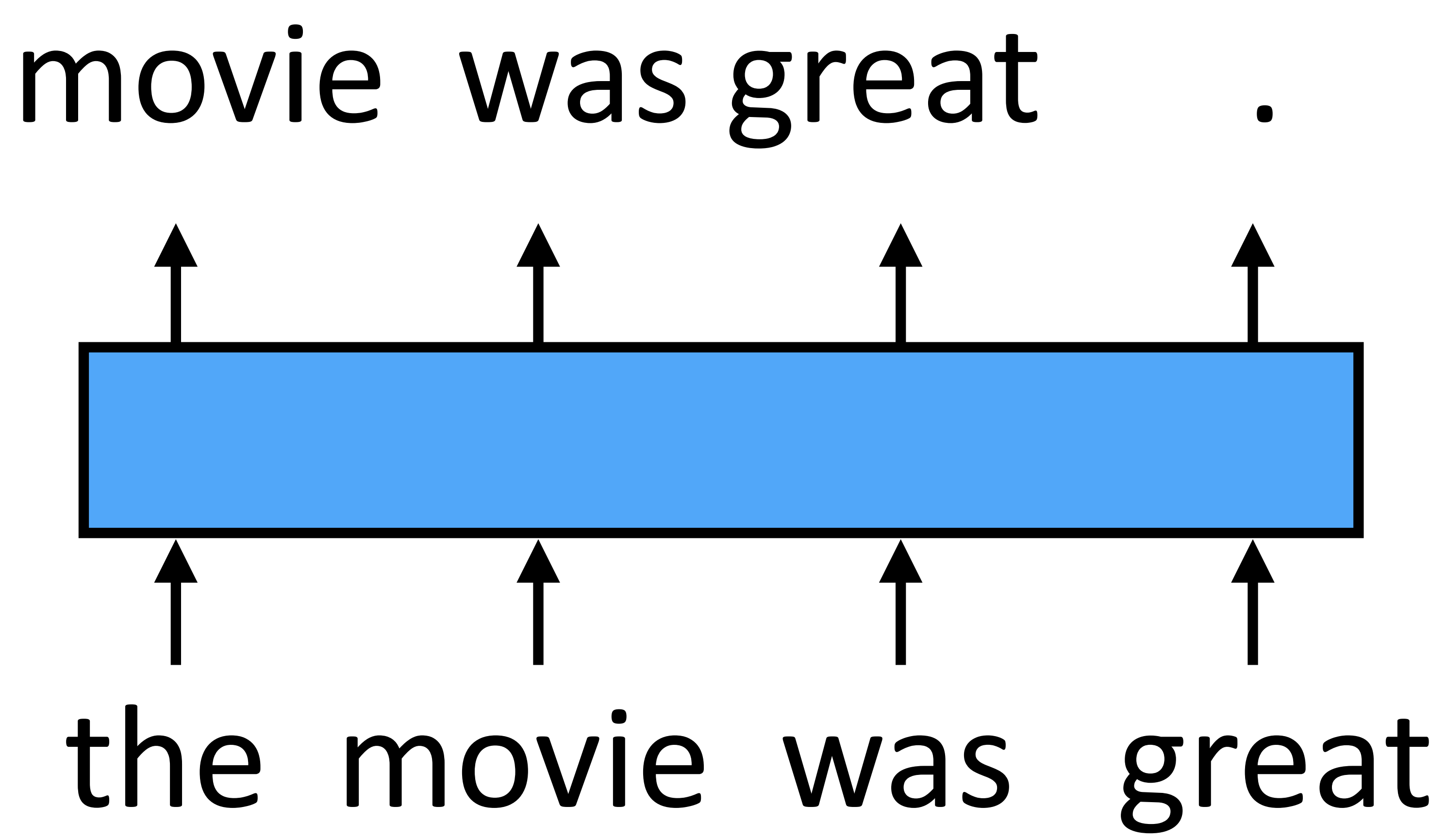


- We want to mask out everything in red (an upper triangular matrix)
- If you implement an LM and you jump to 100% train accuracy and 0% dev, 9/10 times this is why

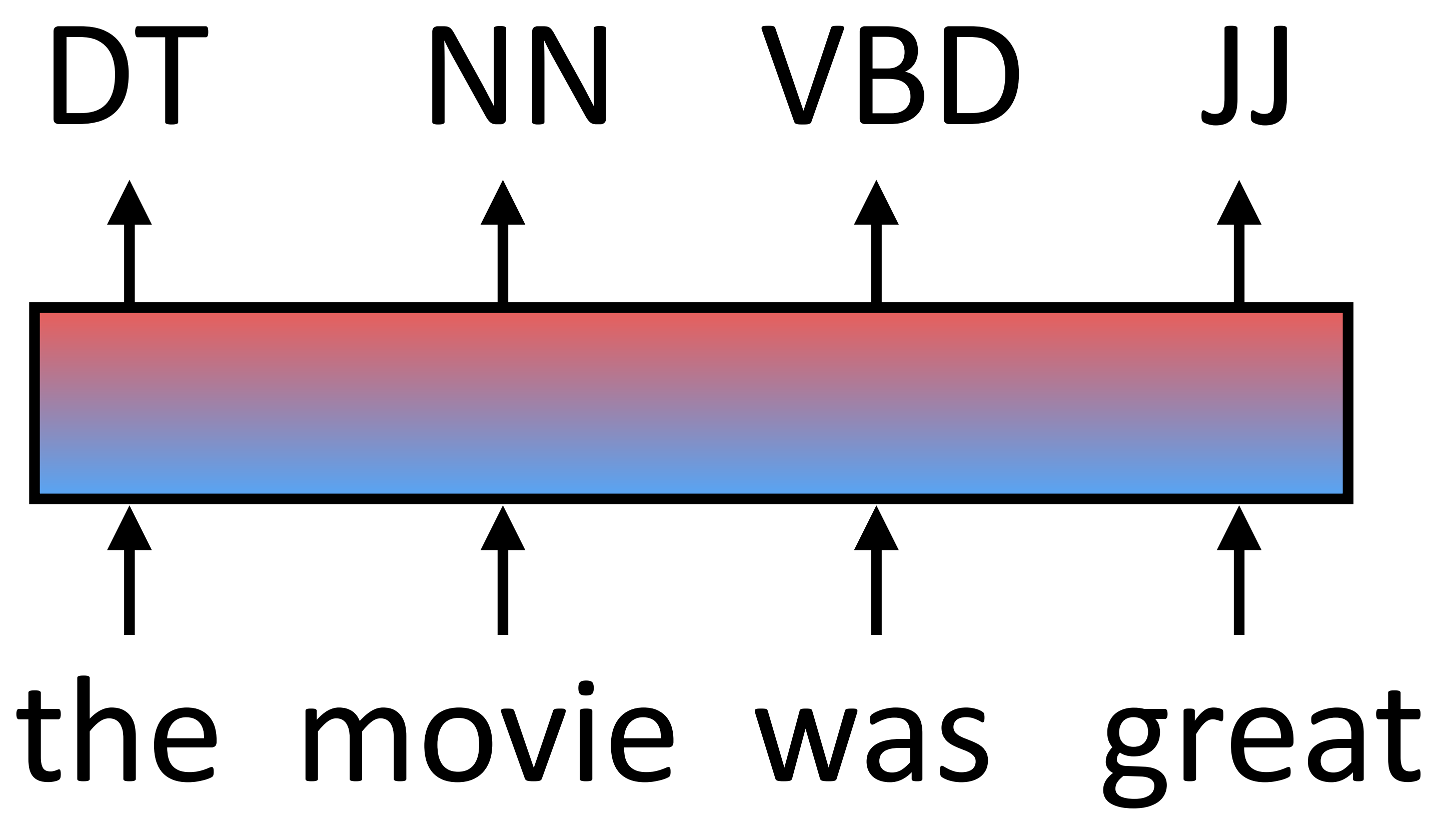
Preview: Pre-training and BERT

- Transformers are usually large and you don't want to train them for each new task

Train on language modeling…



then "fine-tune" that model on your target task with a new classification layer



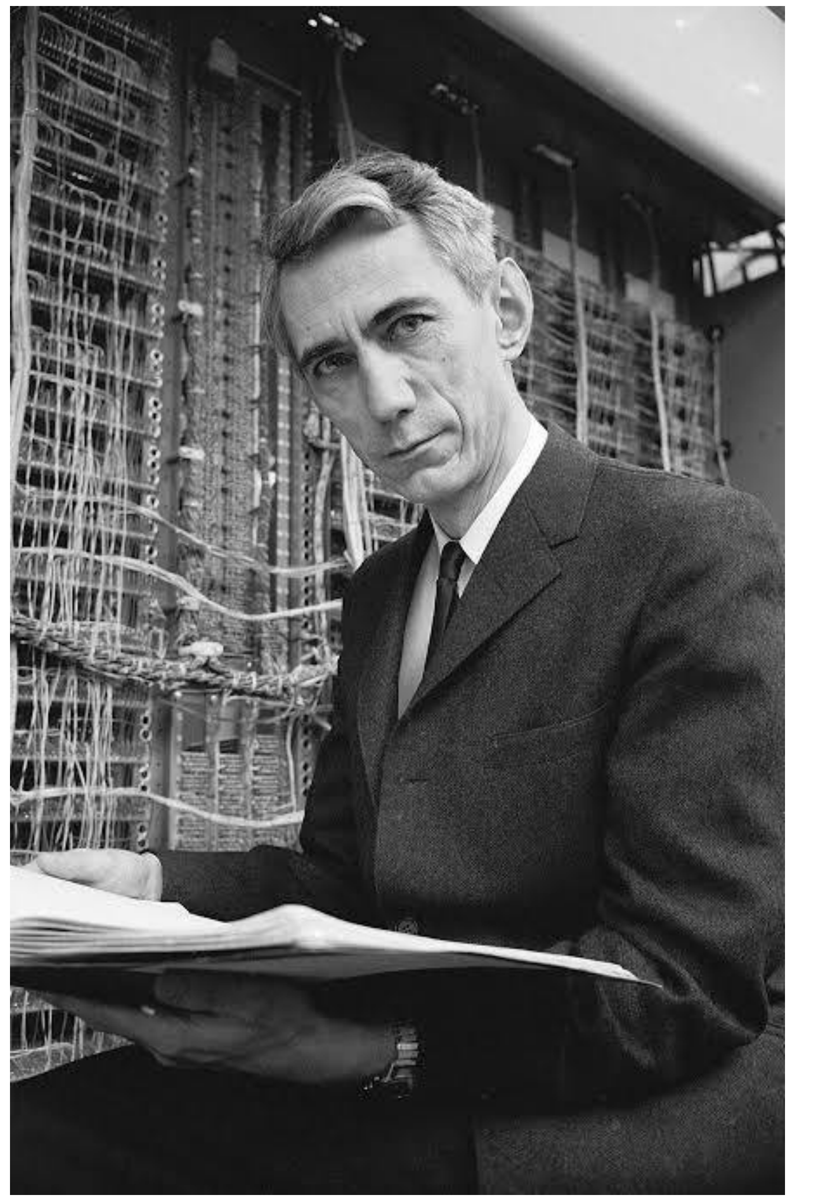
LM Evaluation

- Accuracy doesn't make sense — predicting the next word is generally impossible so accuracy values would be very low
- Evaluate LMs on the likelihood of held-out data (averaged to normalize for length)

$$\frac{1}{n}\sum_{i=1}^{n}\log P(w_i|w_1,\ldots,w_{i-1})$$

- Perplexity: exp(average negative log likelihood). Lower is better
  - Suppose we have probs 1/4, 1/3, 1/4, 1/3 for 4 predictions
  - Avg NLL (base e) = 1.242 Perplexity = 3.464 <== geometric mean of denominators

Information Theory



- PPL is $e^{H(p)} = e^{-\sum_i p_i \ln p_i}$
- Where H(p) is the entropy
- How many bits do I need to describe the distribution?
- Imagine 2 car colors: red and blue.
  - Uniform distribution: 50 red, 50 blue

$$H = -(0.5 \ln 0.5 + 0.5 \ln 0.5) = \ln 2 = 0.693$$

  - Skewed: 90 red, 10 blue

$$H = -(0.9 \ln 0.9 + 0.1 \ln 0.1) = 0.325$$

What does this mean?

- Now imagine: my model says 50-50, but the true distribution is 90-10

$$H = -(0.9 \ln 0.5 + 0.1 \ln 0.5) = 0.693$$

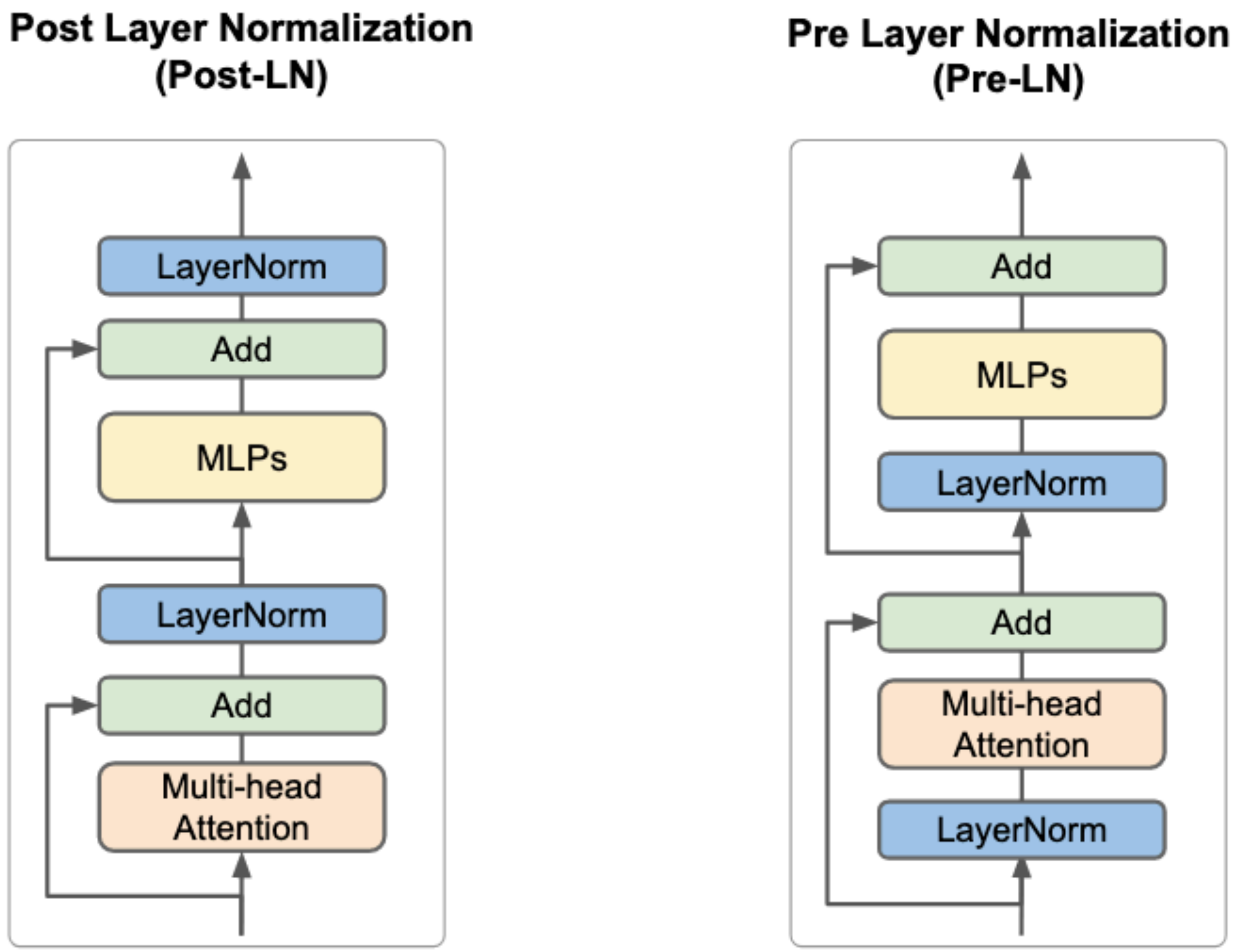
- Also called surprisal: remember this way. If I told you a coin was fair (you had 50-50 model in your head) then you'd be surprised to see a 90-10 distribution

- The better your language model, the less "surprised" it is to see real data
  - Untrained model: ~uniform distribution over tokens
  - Trained model: distribution that captures real tokens with high probability

Transformer Extensions

Pre-Norm

- Layer norm: residual —> layernorm
- Pre-norm: before addition.



https://deeprevision.github.io/posts/001-transformer/

$$\textsc{PostNorm}: \quad \boldsymbol{x}_{\ell+1} = \textsc{LayerNorm}(\boldsymbol{x}_\ell + F_\ell(\boldsymbol{x}_\ell)). \quad (2)$$

$$\textsc{PreNorm}: \quad \boldsymbol{x}_{\ell+1} = \boldsymbol{x}_\ell + F_\ell(\textsc{LayerNorm}(\boldsymbol{x}_\ell)). \quad (3)$$

Nguyen and Salazar, 2019

Scaling Laws

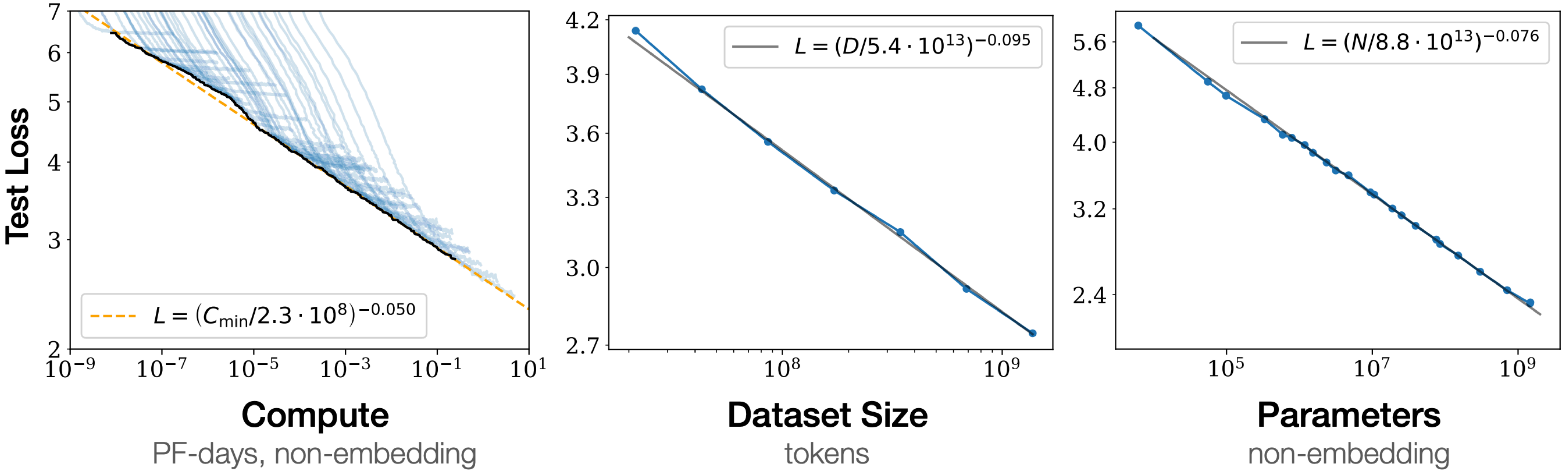


Figure 1 Language modeling performance improves smoothly as we increase the model size, datasetset size, and amount of compute[2] used for training. For optimal performance all three factors must be scaled up in tandem. Empirical performance has a power-law relationship with each individual factor when not bottlenecked by the other two.

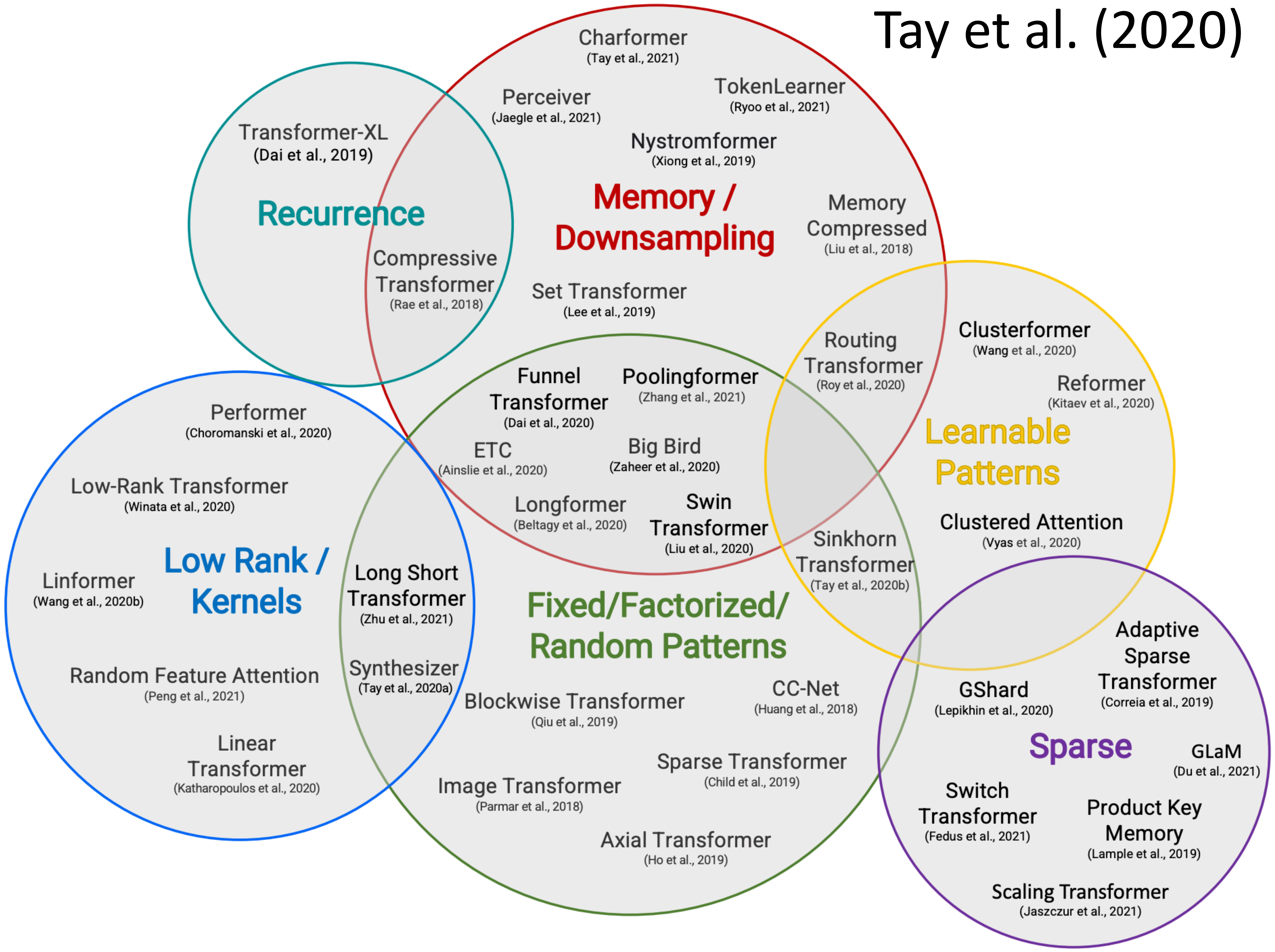
- Transformers scale really well!

Kaplan et al. (2020)

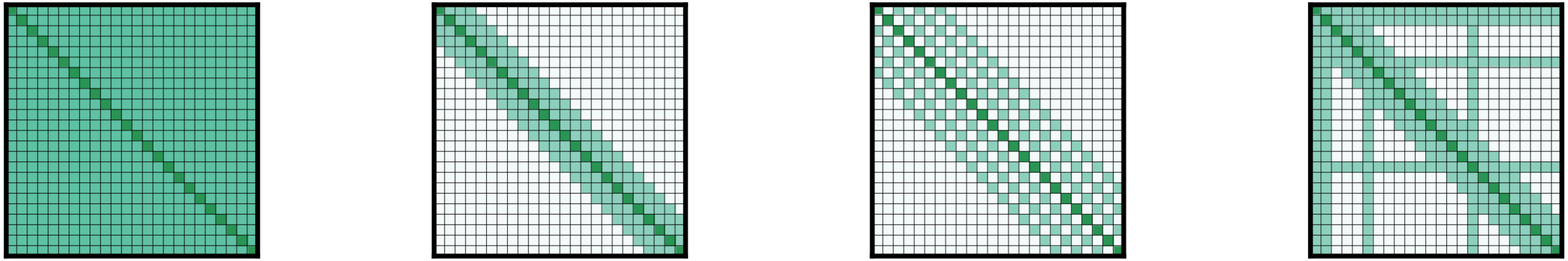
Transformer Runtime

- Even though most parameters and FLOPs are in feedforward layers, Transformers are still limited by quadratic complexity of self-attention
- Many ways proposed to handle this

Tay et al. (2020)



Longformer



(a) Full n^2 attention (b) Sliding window attention (c) Dilated sliding window (d) Global+sliding window

Figure 2: Comparing the full self-attention pattern and the configuration of attention patterns in our Longformer.

- Use several pre-specified self-attention patterns that limit the number of operations while still allowing for attention over a reasonable set of things
- Scales to 4096-length sequences

Beltagy et al. (2021)

Mixture-of-Experts

- Old idea in ML: combine a bunch of “expert” networks into an ensemble
- Recall: FFNN is where most of the weights live (OPT 175B: 80% of FLOPs)
- Can we make FFNN sparse?
  - Think: what’s the advantage of sparsity?

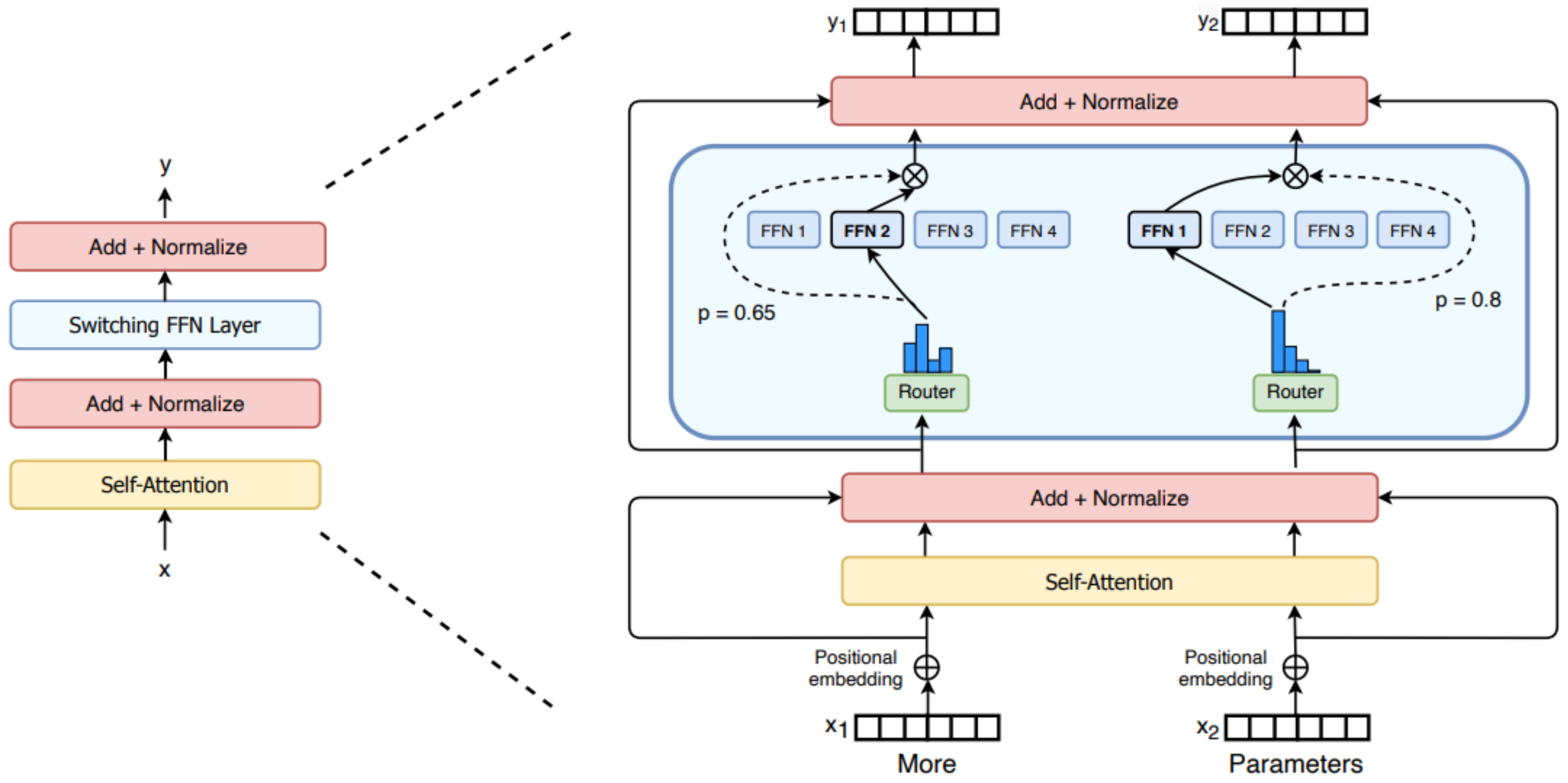
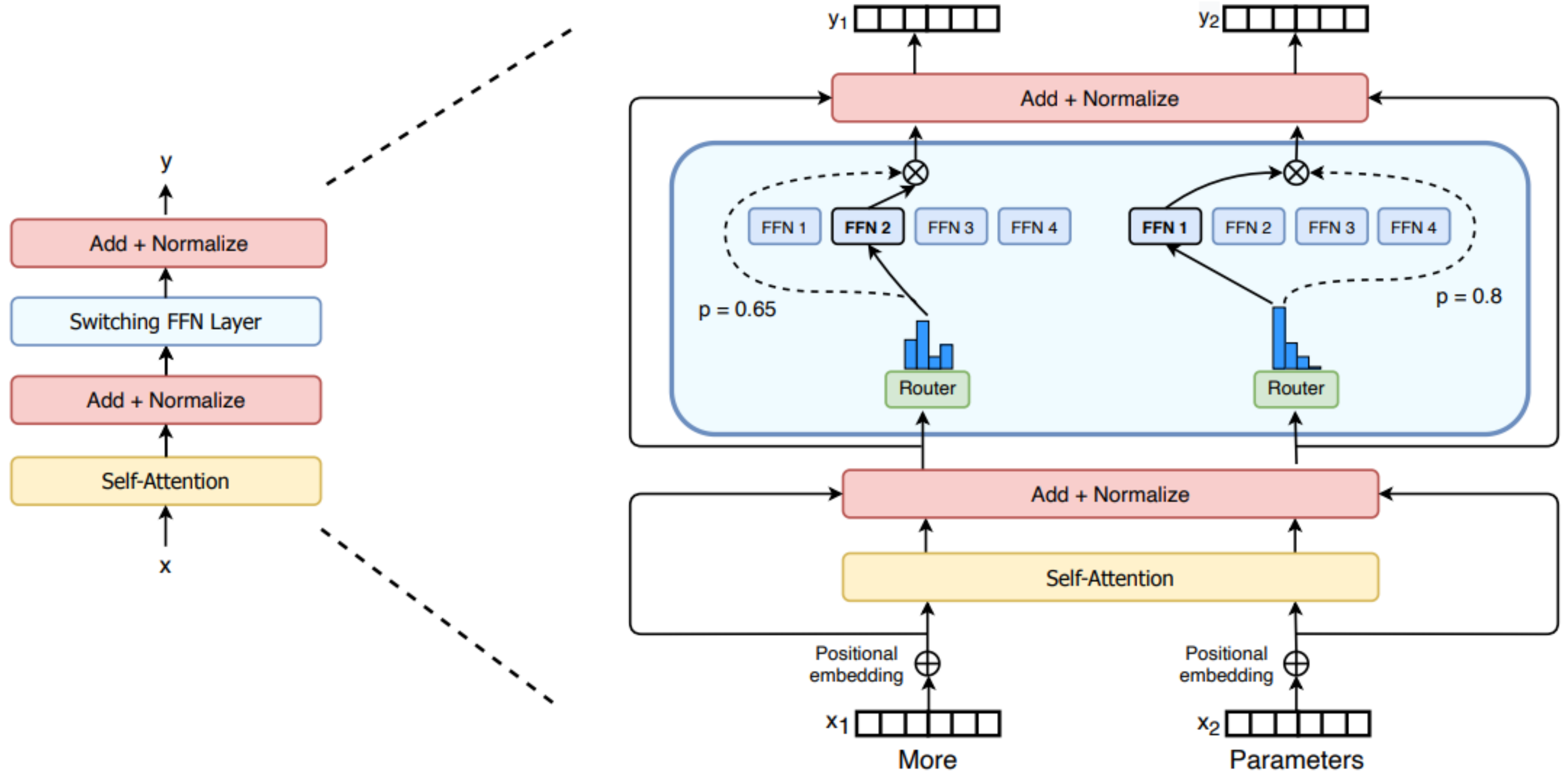


Figure 2: Illustration of a Switch Transformer encoder block. We replace the dense feed forward network (FFN) layer present in the Transformer with a sparse Switch FFN layer (light blue). The layer operates independently on the tokens in the sequence. We diagram two tokens (x_1 = “More” and x_2 = “Parameters” below) being routed (solid lines) across four FFN experts, where the router independently routes each token. The switch FFN layer returns the output of the selected FFN multiplied by the router gate value (dotted-line).

Fedus et al. (2021)

Mixture-of-Experts

- MoE idea: route each token to different "expert" subnetwork
- Why is this more efficient?
  - Each token only requires smaller # of FFNN params —> fewer FLOPs
  - Trade-off: memory requirement. FLOPs are lower but you need to keep experts in VRAM. Routing adds communication overhead.



Fedus et al. (2021)

- Many frontier models you interact with are (likely) MoEs!

Frontiers

- Will come back later in the semester when we talk about efficiency in LLMs

- Engineering-based approaches like Flash Attention and Linear Attention (which supports the "basic" Transformer) have superseded changing the Transformer model itself

Vision, Other Modalities, and RL

- Vision/Multimodal: ViT, VLMs, Omni-modal models
  Treat images/audio/video/time-series data/scans/etc. as a sequence of tokens. Model separately or with language.

- RL: treat action as a sequence problem (P(action | context)). Decision Transformer: does reinforcement learning by Transformer-based modeling over a series of actions. This is the basis for most "agents".

- Transformers are now being used all over AI, even outside of NLP.

Dosovitskiy (2020), Ramesh et al. (2021), Chen et al. (2021)

Takeaways

- Transformers are going to be the foundation for the much of the rest of this class and are a ubiquitous architecture nowadays

- Many details to get right, many ways to tweak and extend them, but core idea is the multi-head self attention and their ability to contextualize items in sequences