

CS378: Natural Language Processing

Lecture 17: Transformers for Language Modeling, Implementation

Greg Durrett



TEXAS

The University of Texas at Austin

Transformers

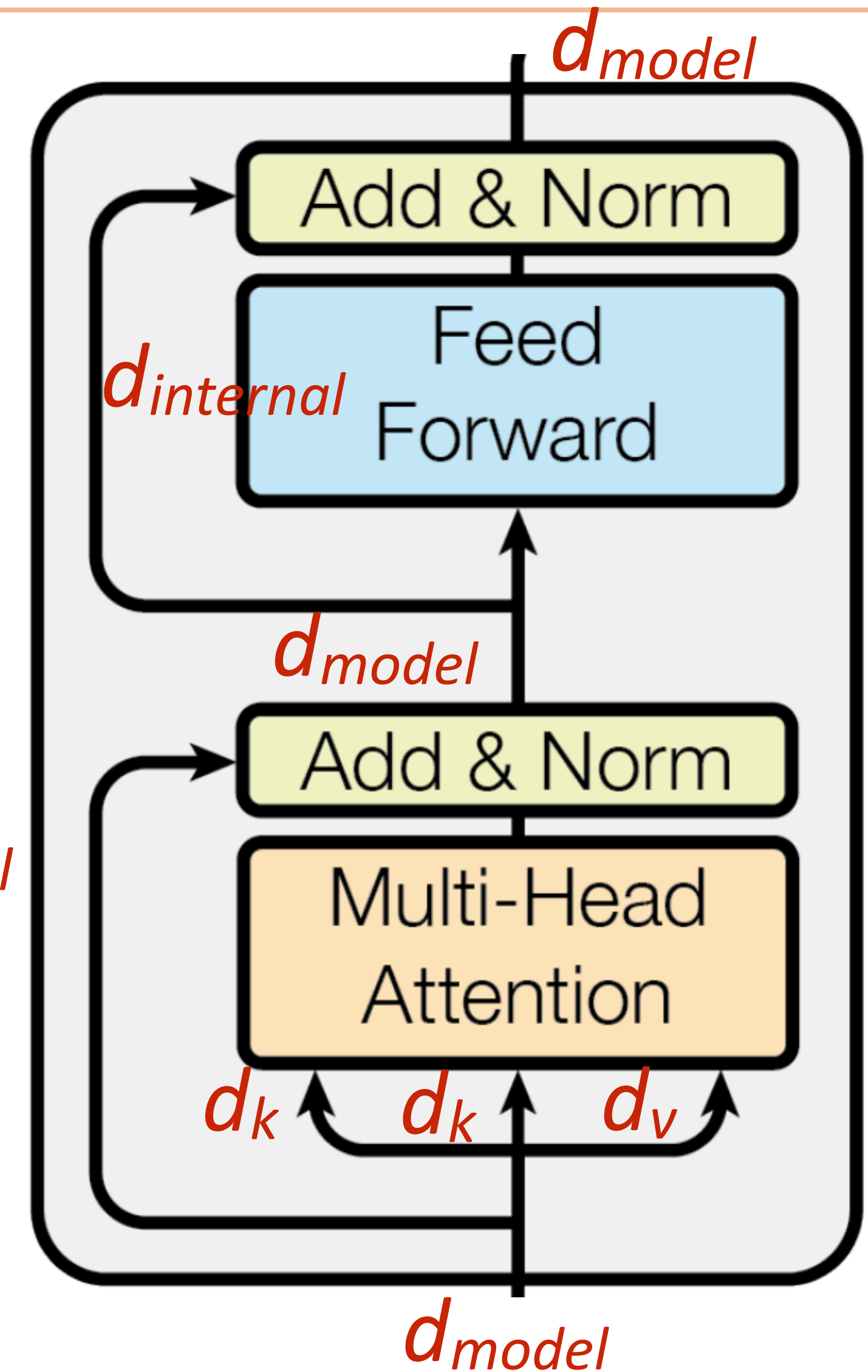


Dimensions

- ▶ 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 $d_v \times d_{model}$ 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

$d_v \rightarrow d_{model}$

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





Transformers: Position Sensitivity

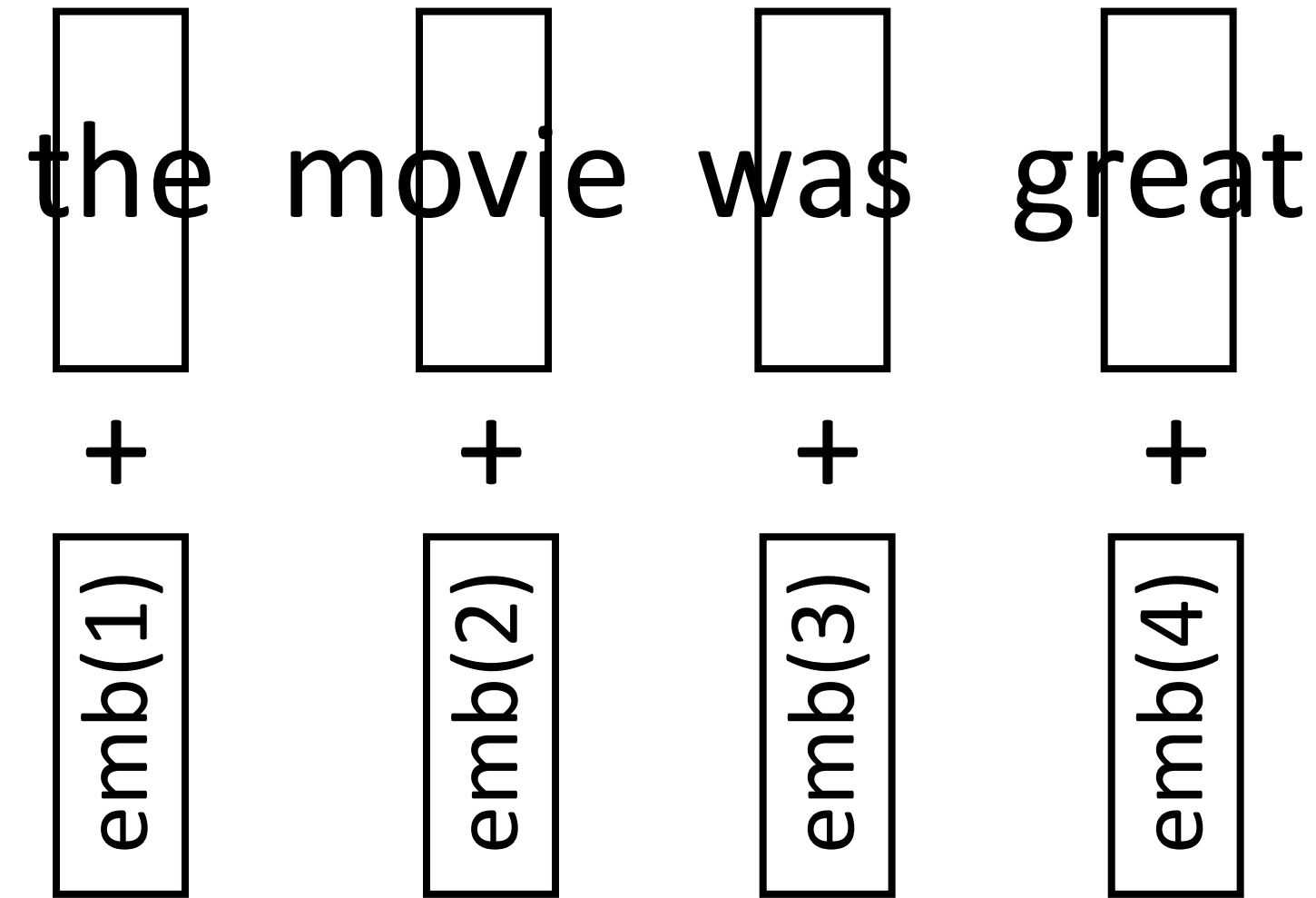
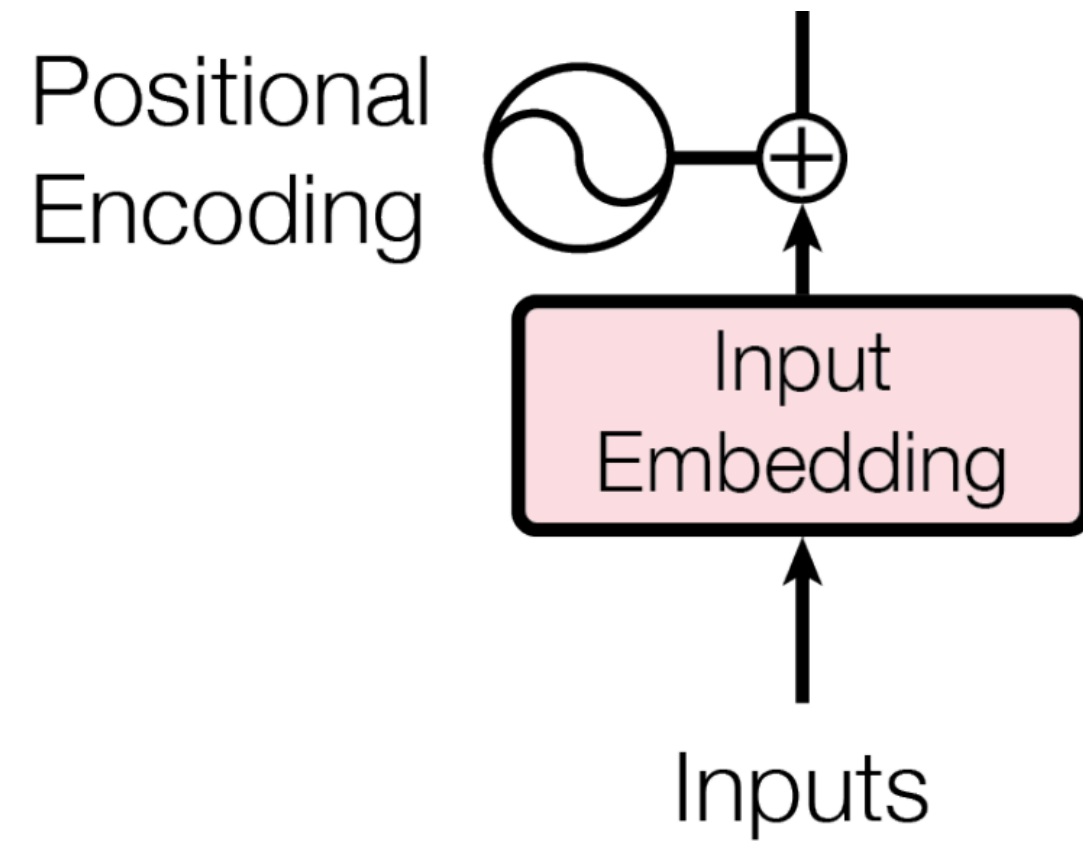
*The ballerina is very excited that **she** will dance in the **show**.*

The diagram illustrates attention weights between the words 'she' and 'show'. A blue arrow points from 'she' to 'The ballerina is very excited that', and another blue arrow points from 'she' to 'will dance in the'. A red arrow points from 'show' to 'The ballerina is very excited that', and another red arrow points from 'show' to 'will dance in the'.

- ▶ If this is in a longer context, we want words to attend *locally*
- ▶ But transformers have *no notion of position* by default



Transformers: Position Sensitivity



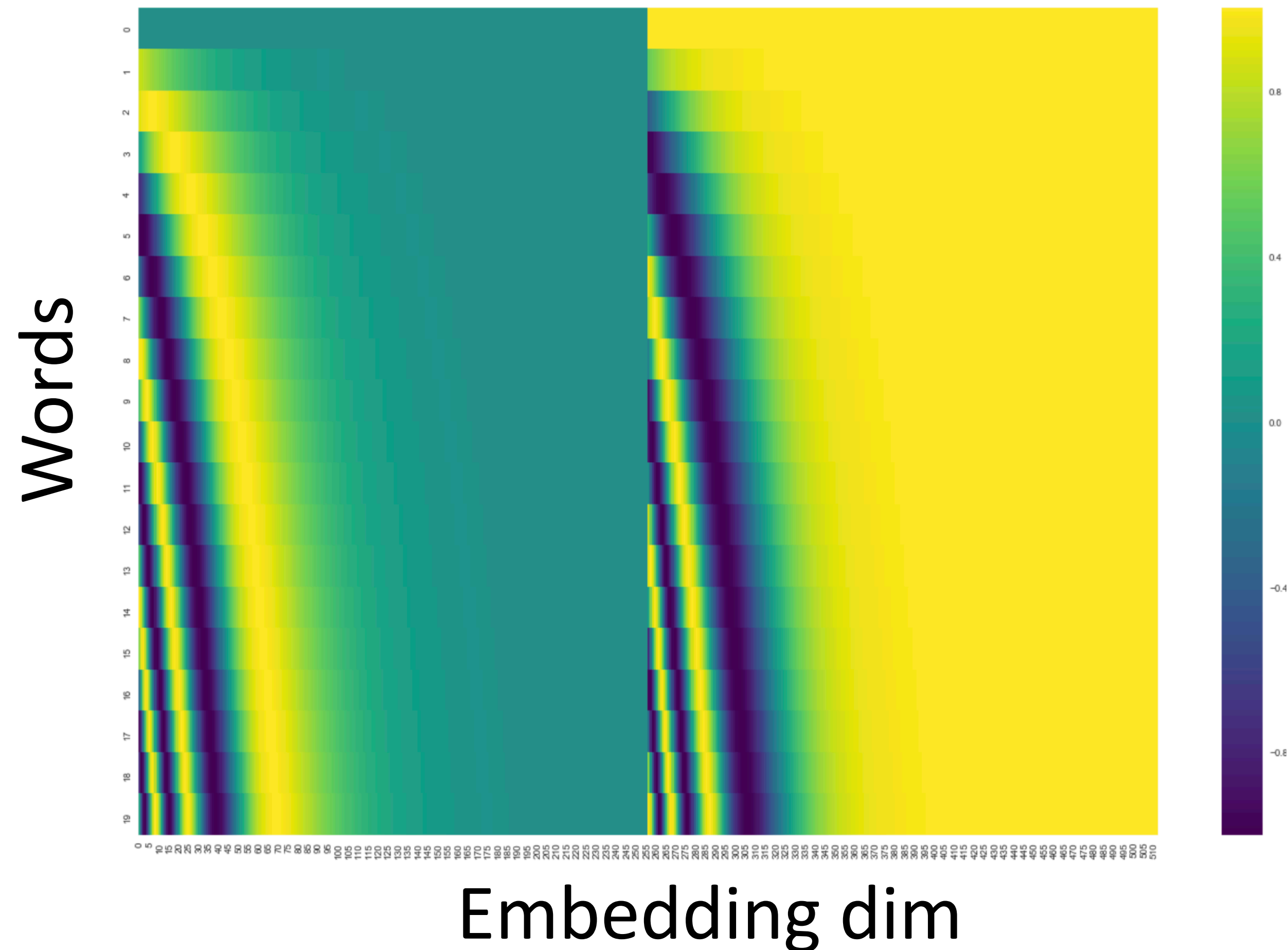
- ▶ Encode each sequence position as an integer, add it to the word embedding vector
- ▶ Why does this work?



Transformers

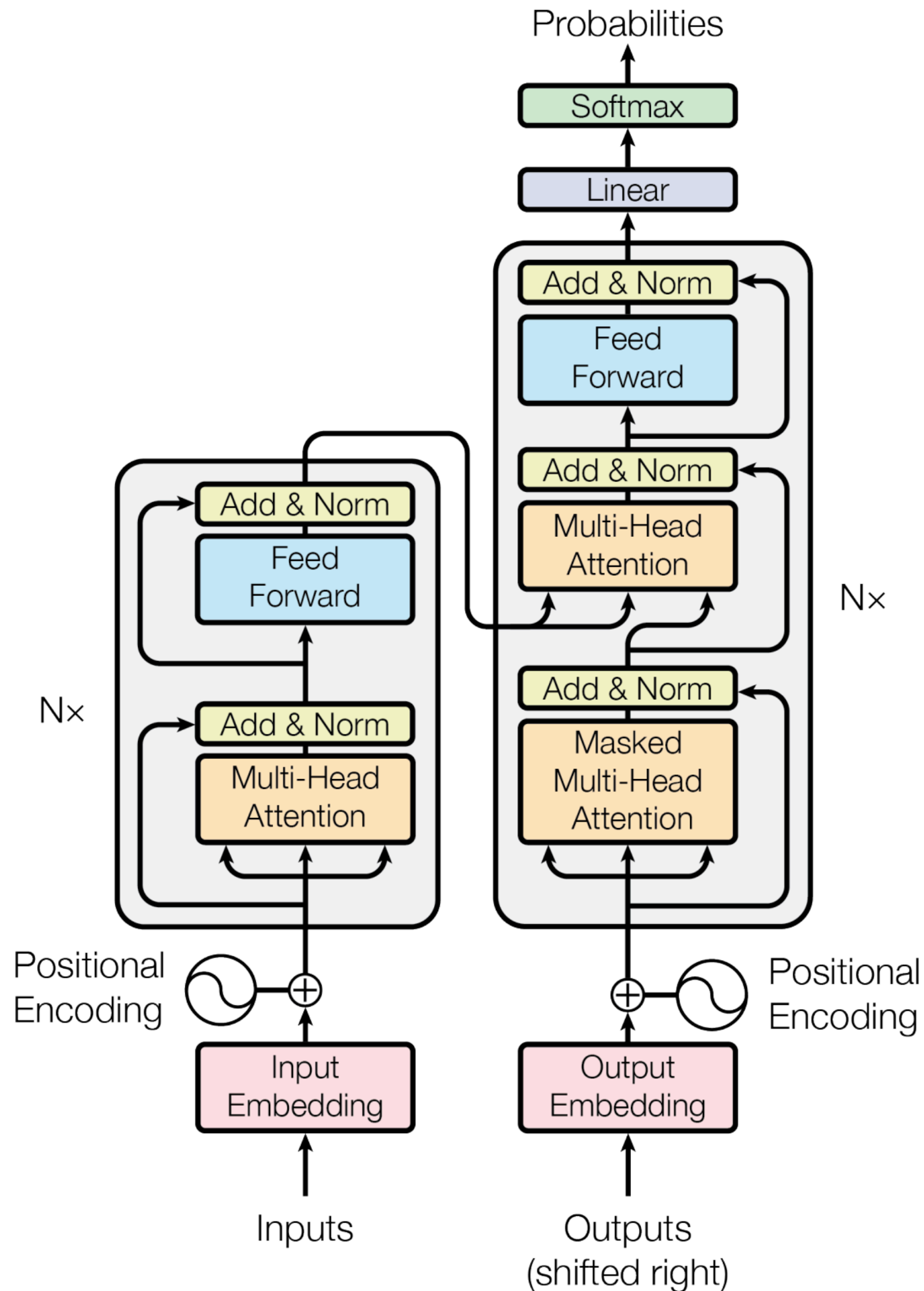
Alammar, *The Illustrated Transformer*

- ▶ Alternative from Vaswani et al.: sines/cosines of different frequencies (closer words get higher dot products by default)





Transformers: Complete Model

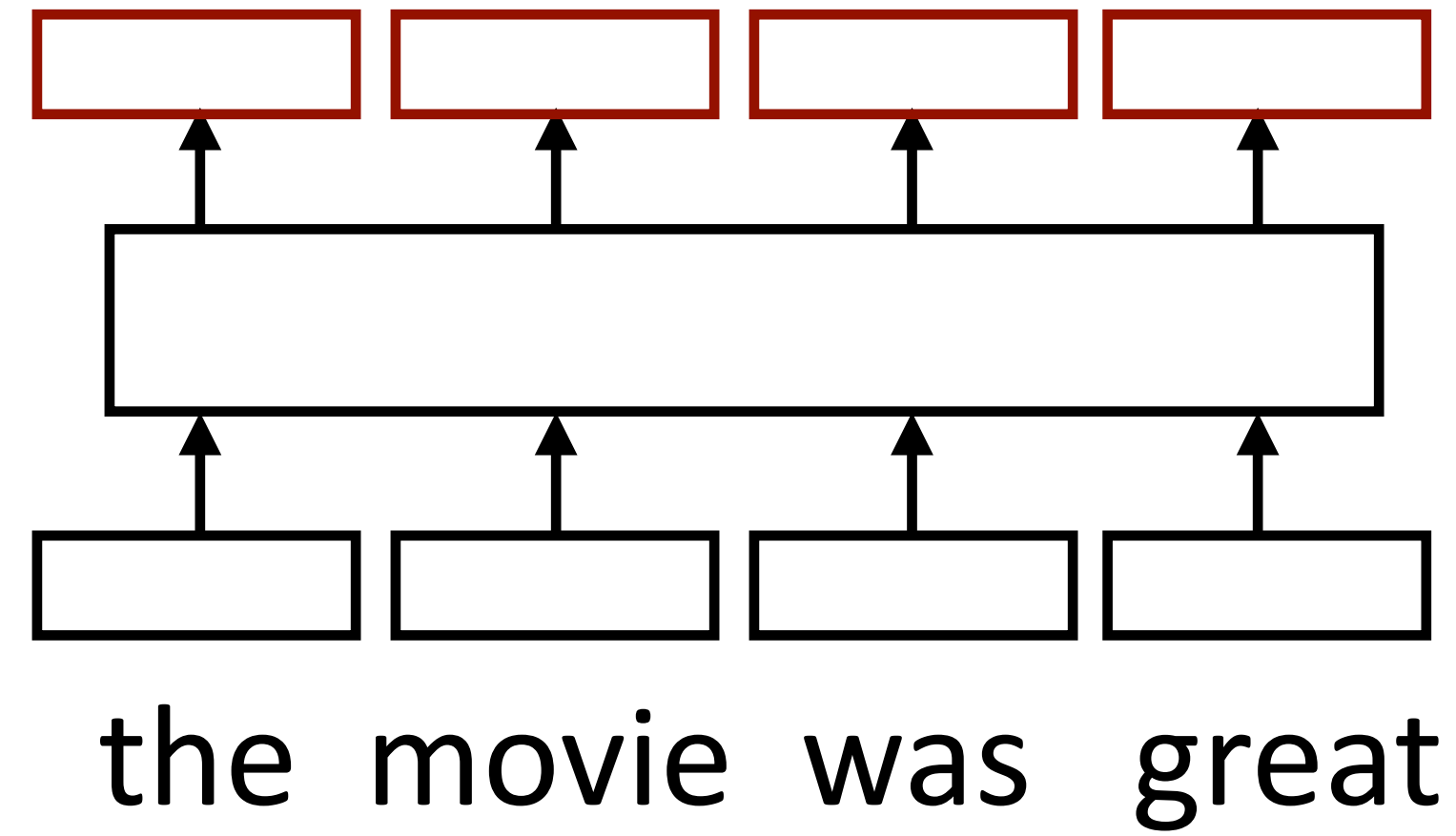


- ▶ Original Transformer paper presents an **encoder-decoder** model
- ▶ Right now we don't need to think about both of these parts — will return in the context of MT
- ▶ Decoder differs because each token only attends to those coming before it. Can do this with an **attention mask**

Using Transformers



What do Transformers produce?

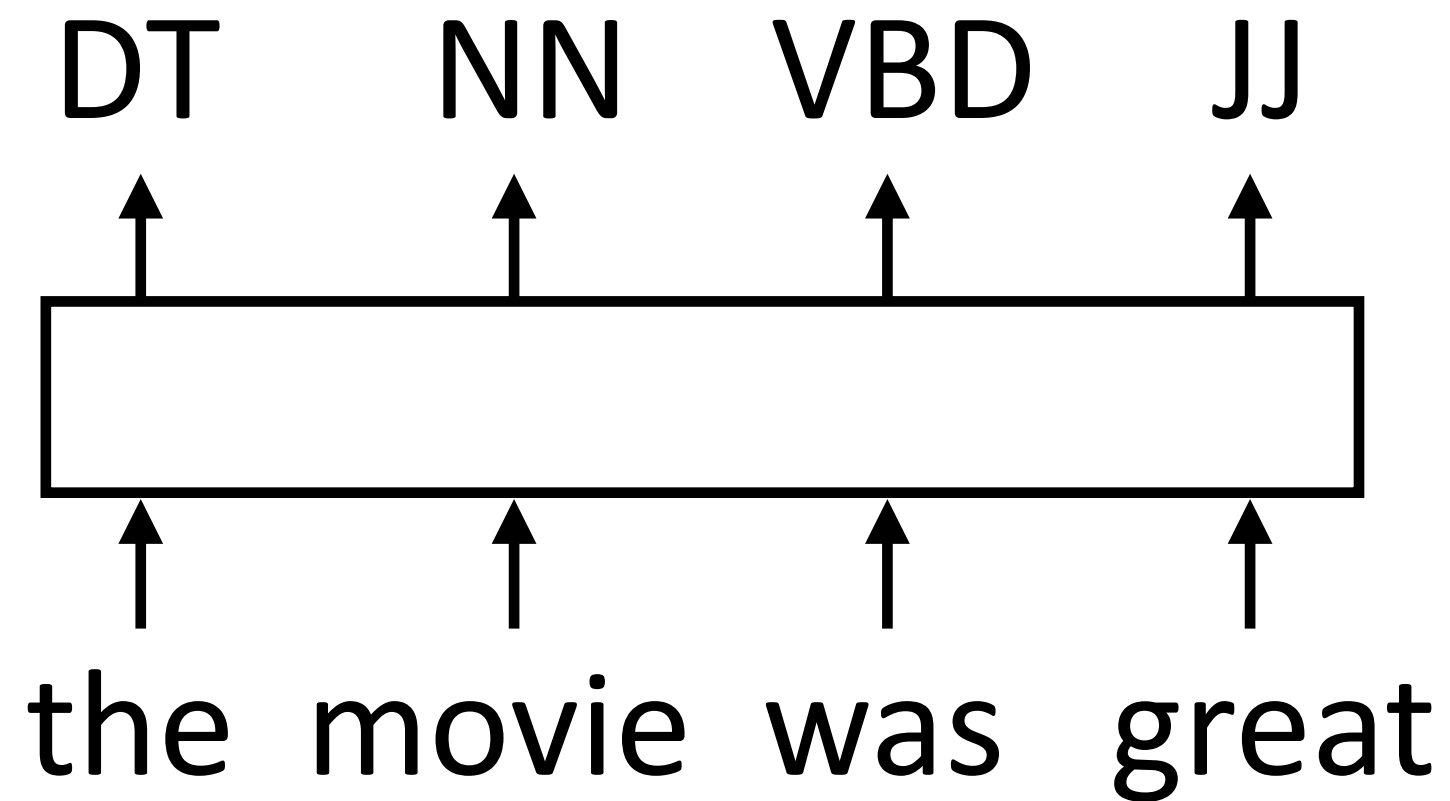


- ▶ **Encoding of each word** — can pass this to another layer to make a prediction (like predicting the next word for language modeling)
- ▶ Like RNNs, Transformers can be viewed as a transformation of a sequence of vectors into a sequence of context-dependent vectors



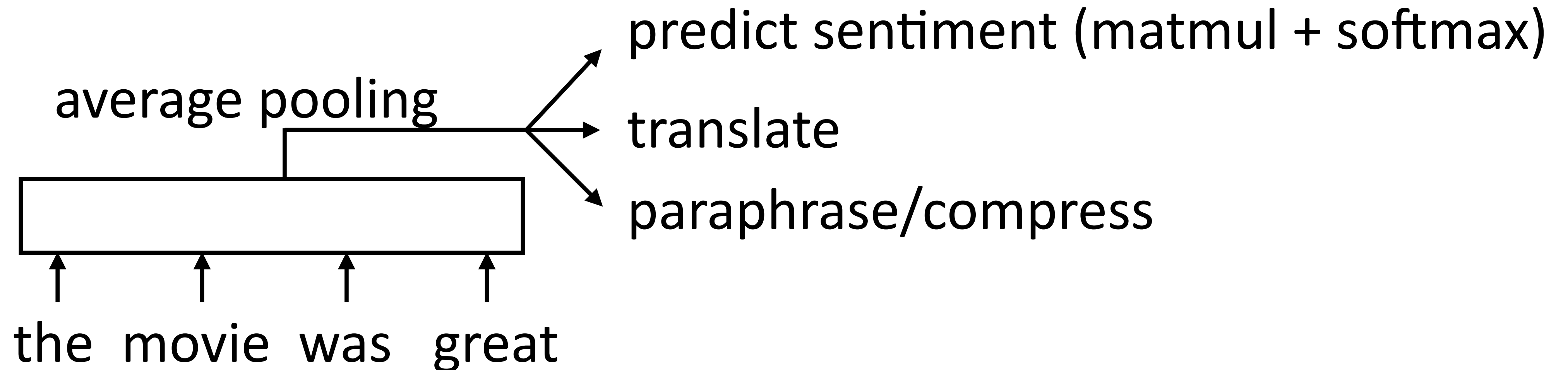
Transformer Uses

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



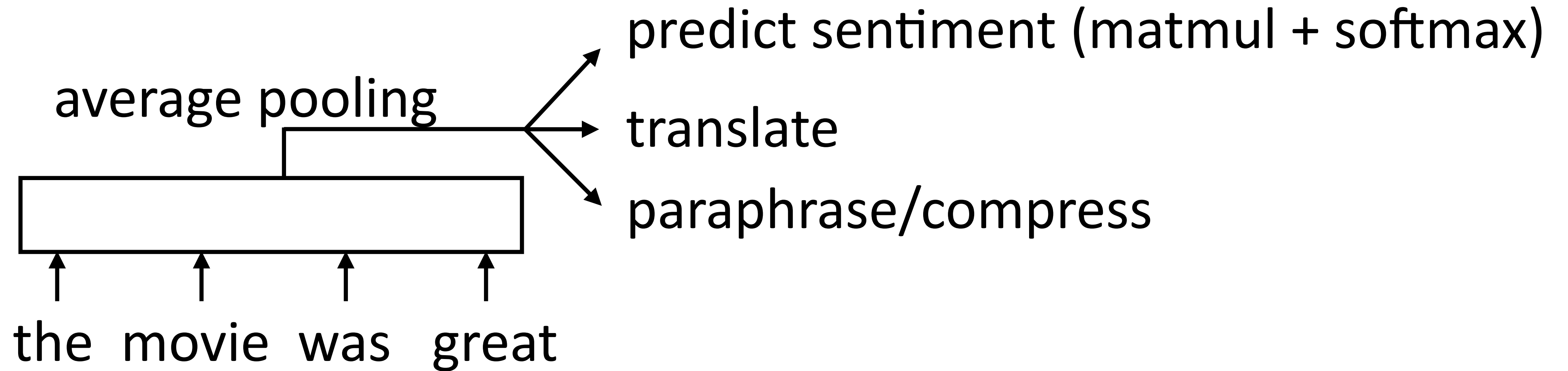
output y = score for each tag, then softmax

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



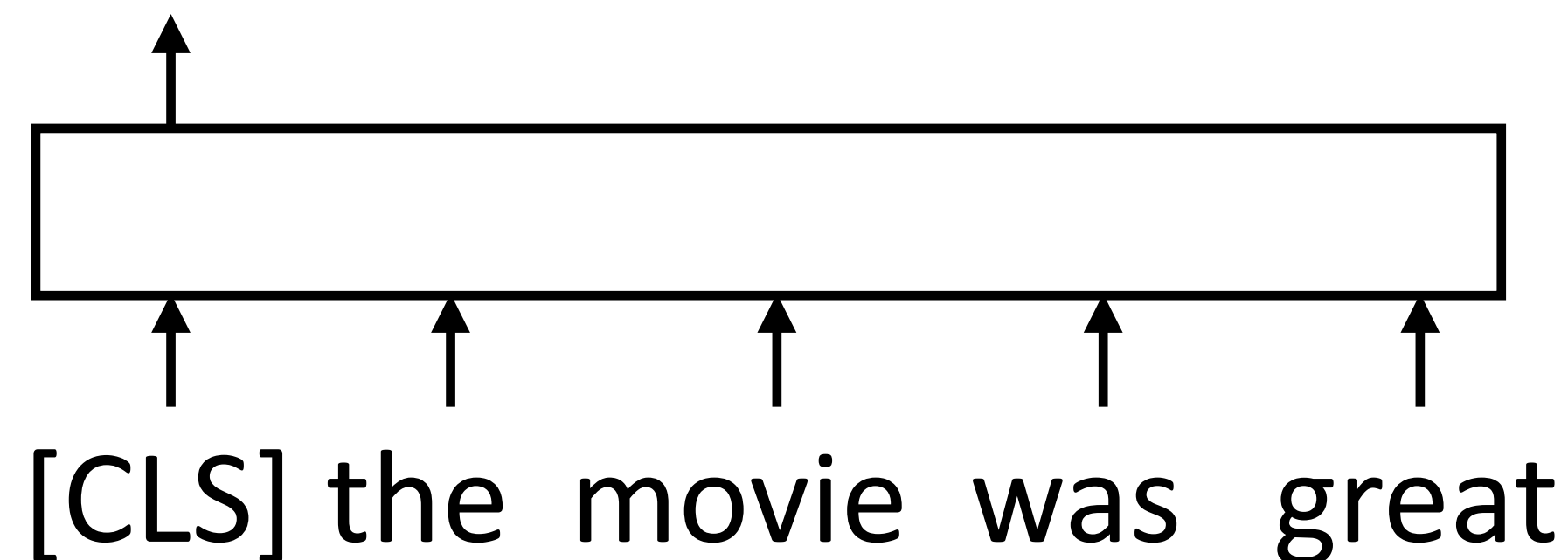


Transformer Uses



- ▶ 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!

encoding of [CLS token] \rightarrow matmul + softmax \rightarrow predict sentiment





Transformer Uses

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

Contradiction



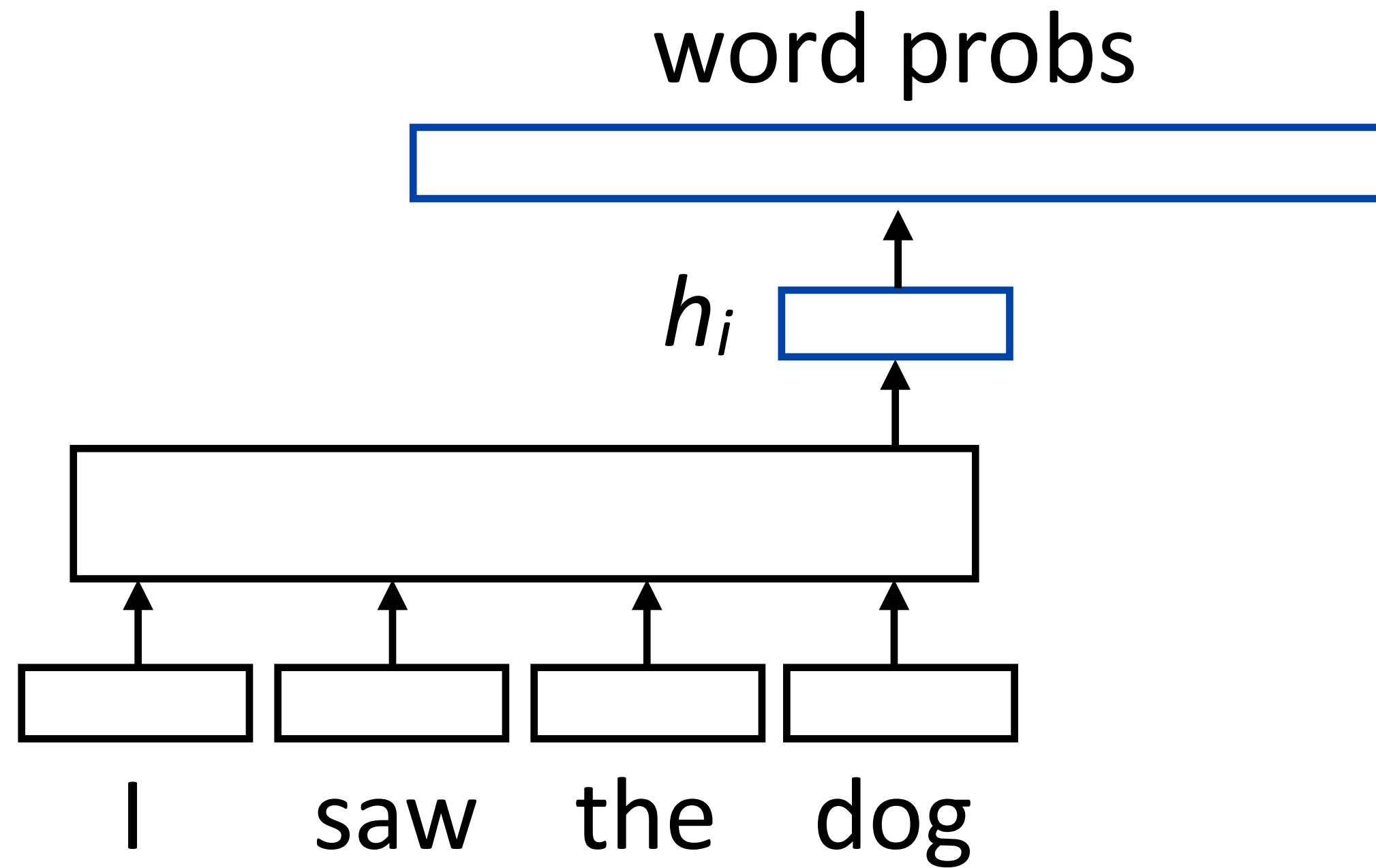
[CLS] The woman is driving a car [SEP] The woman is walking .

- ▶ Why might Transformers be particularly good at sentence **pair** tasks compared to something like a DAN?

Transformer Language Modeling



Transformer Language Modeling



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

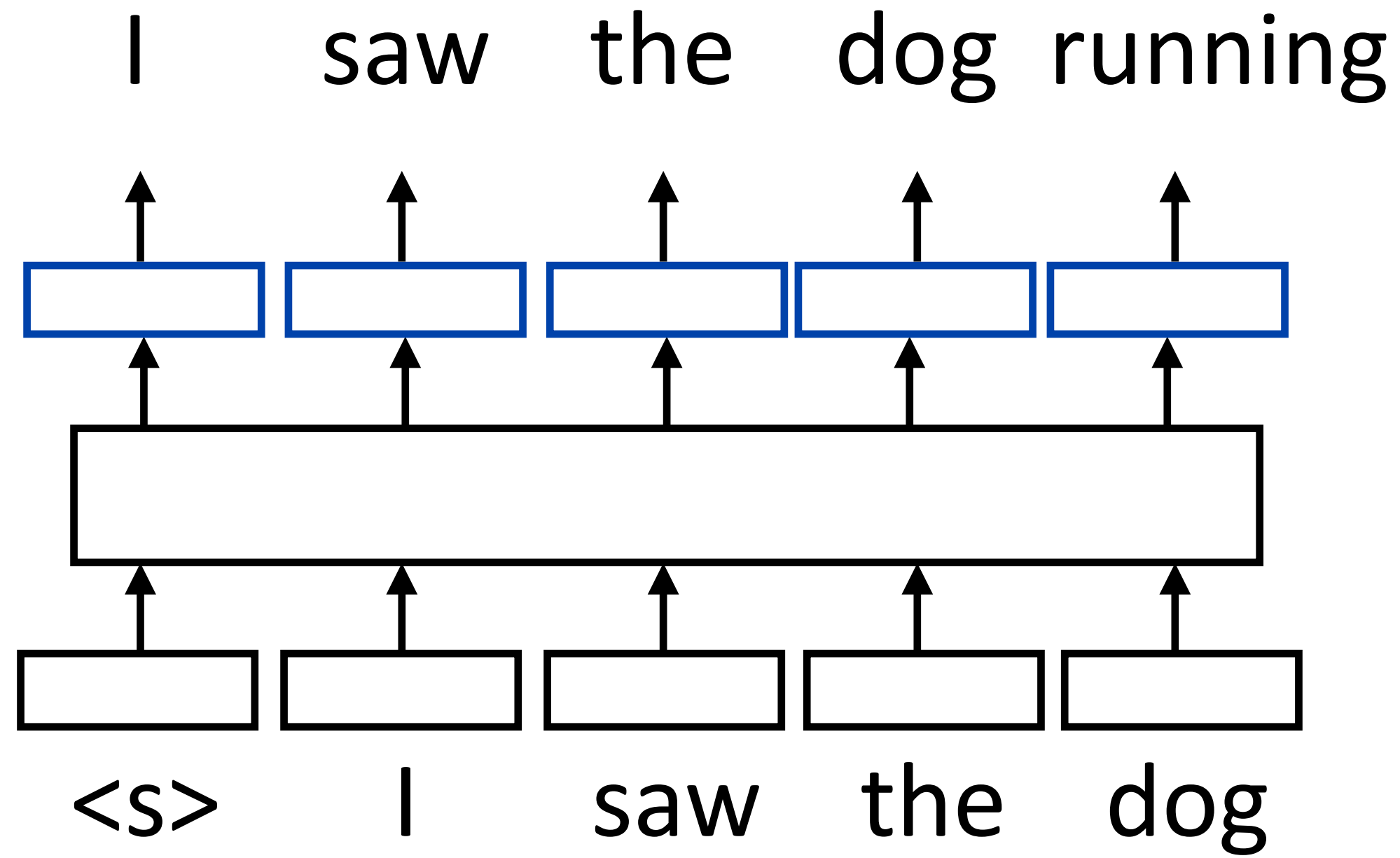
equivalent to

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

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



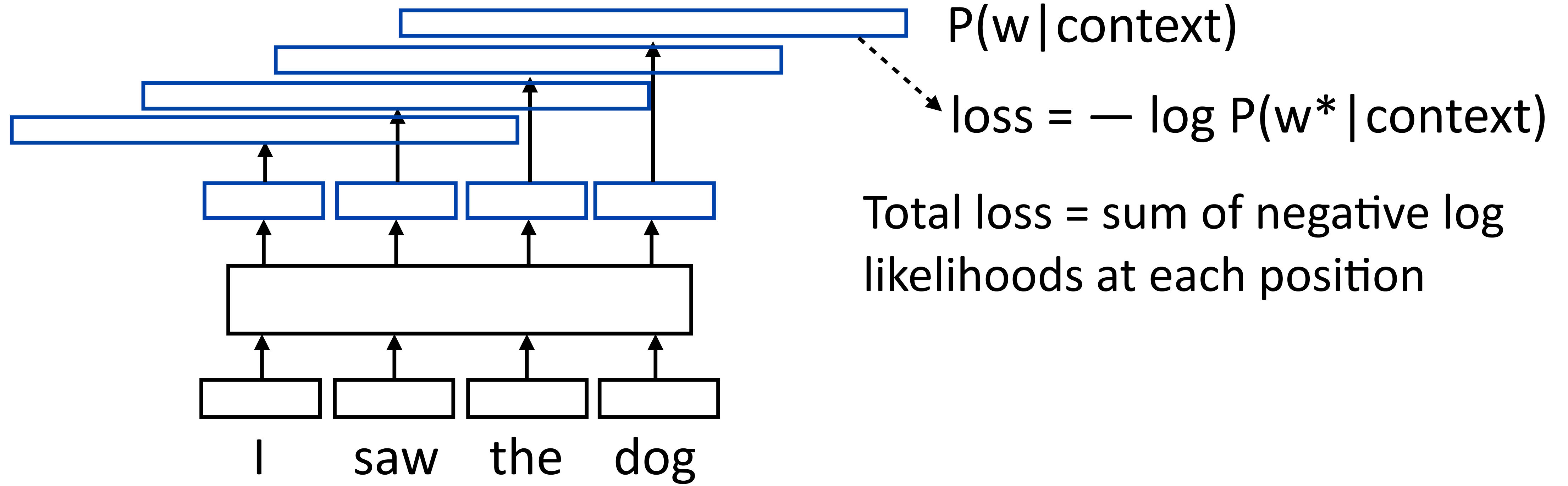
Training Transformer LMs



- ▶ 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



```
loss_fcn = nn.NLLLoss()
```

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

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

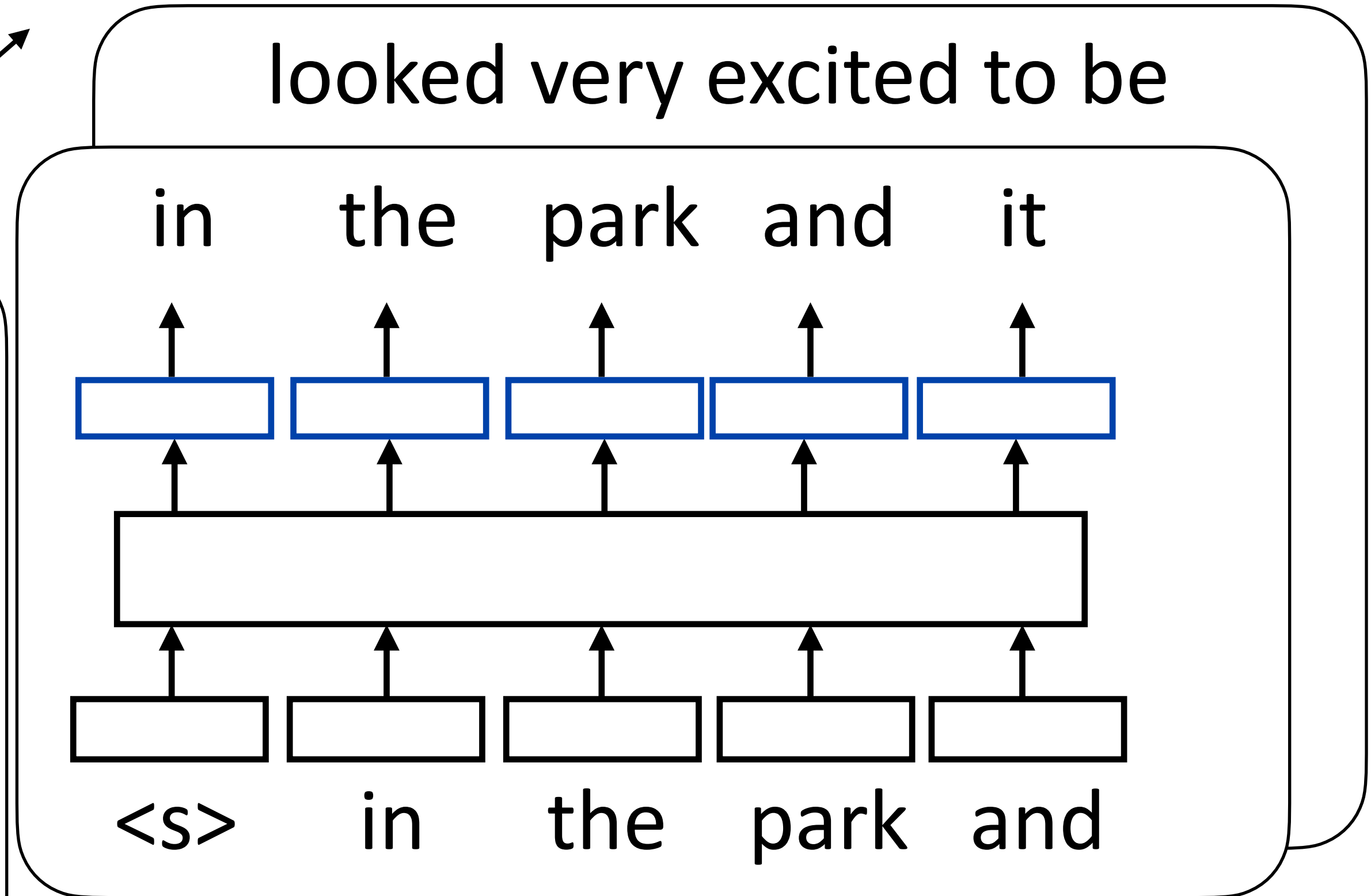
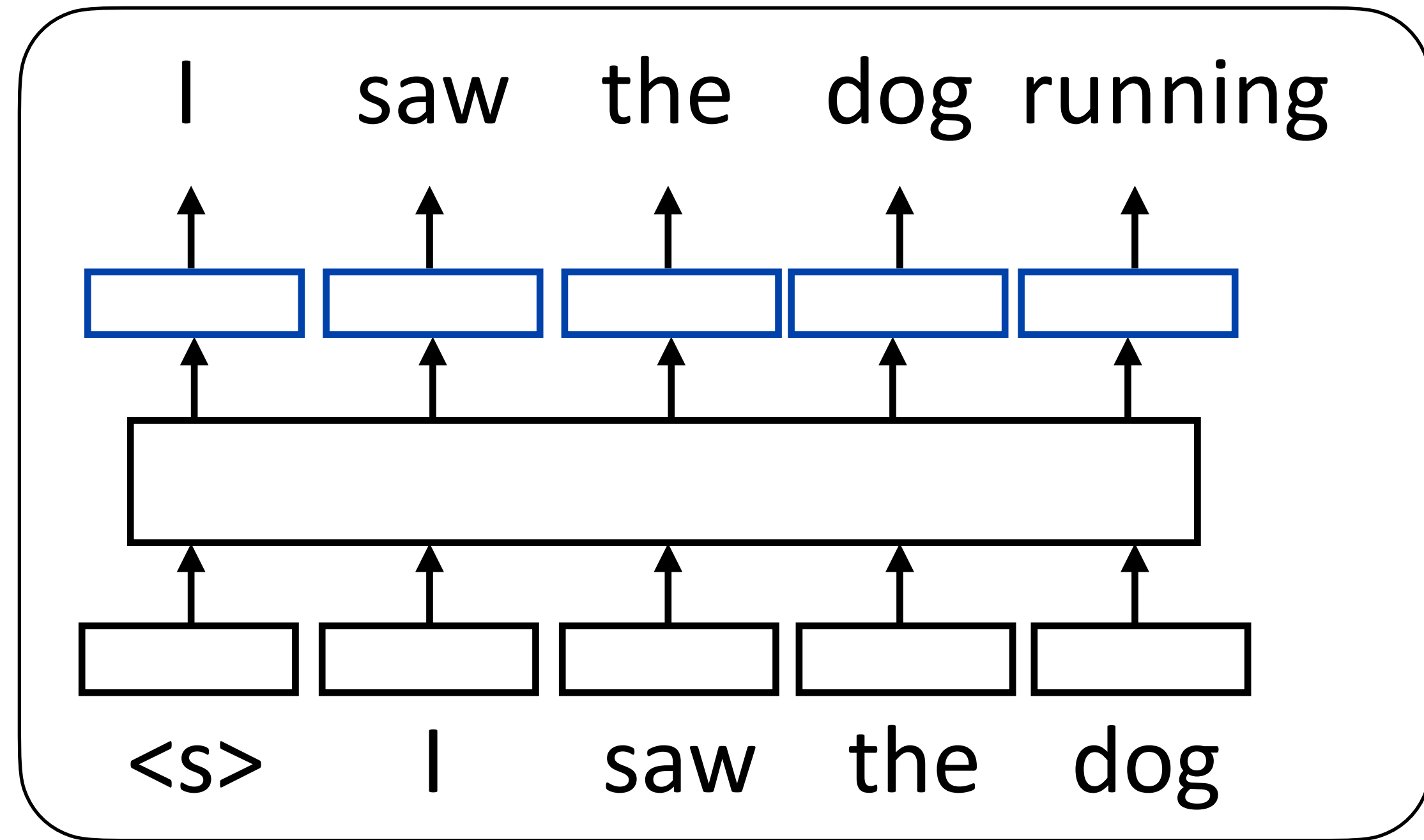
- ▶ Batching is a little tricky with NLLoss: need to collapse [batch, seq len, num classes] to [batch * seq len, num classes]. You do not need to batch



Batched LM Training

I saw the dog running in the park and it looked very excited to be there

batch dim

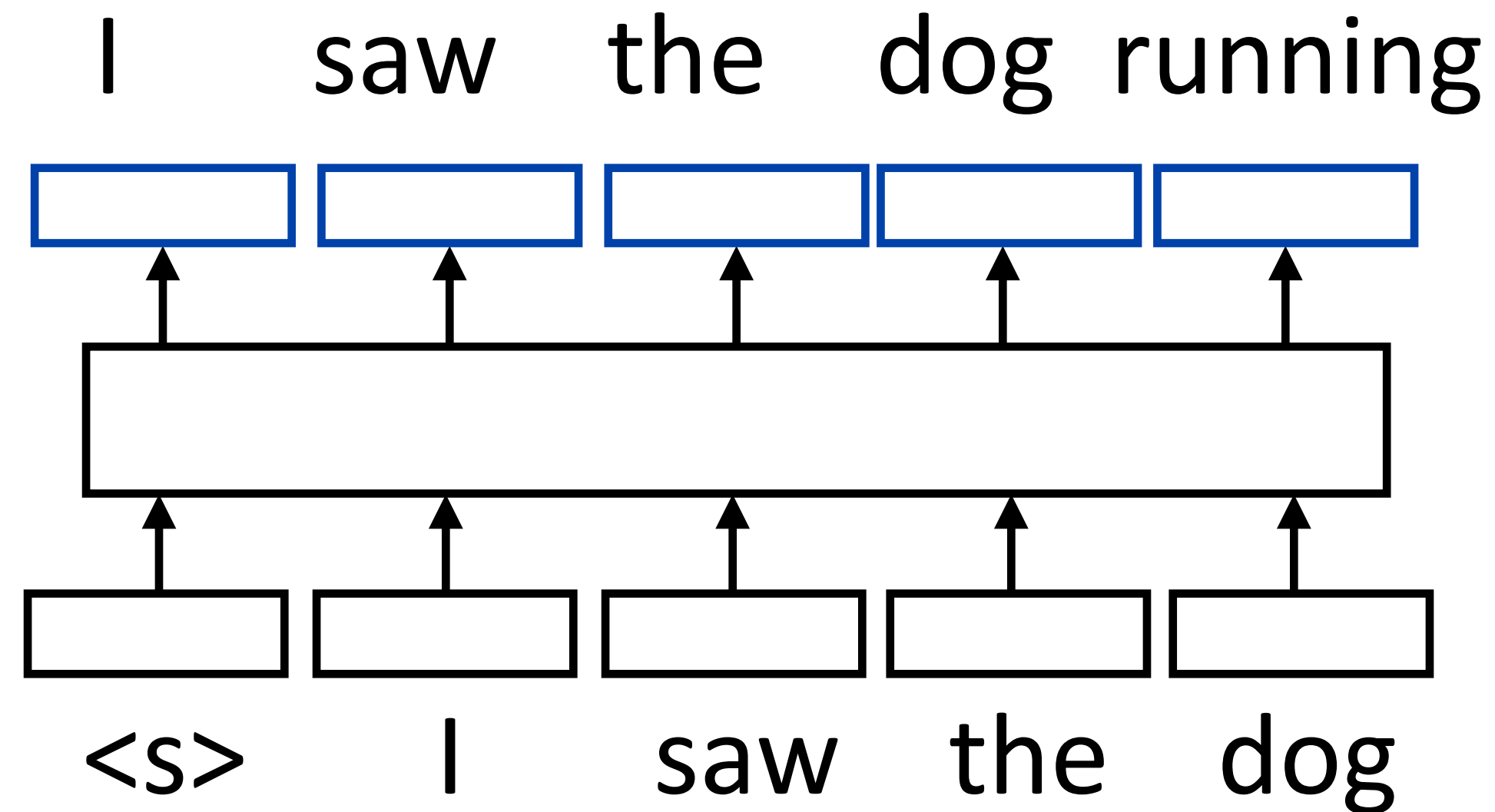


- ▶ Multiple sequences **and** multiple timesteps per sequence



A Small Problem with Transformer LMs

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

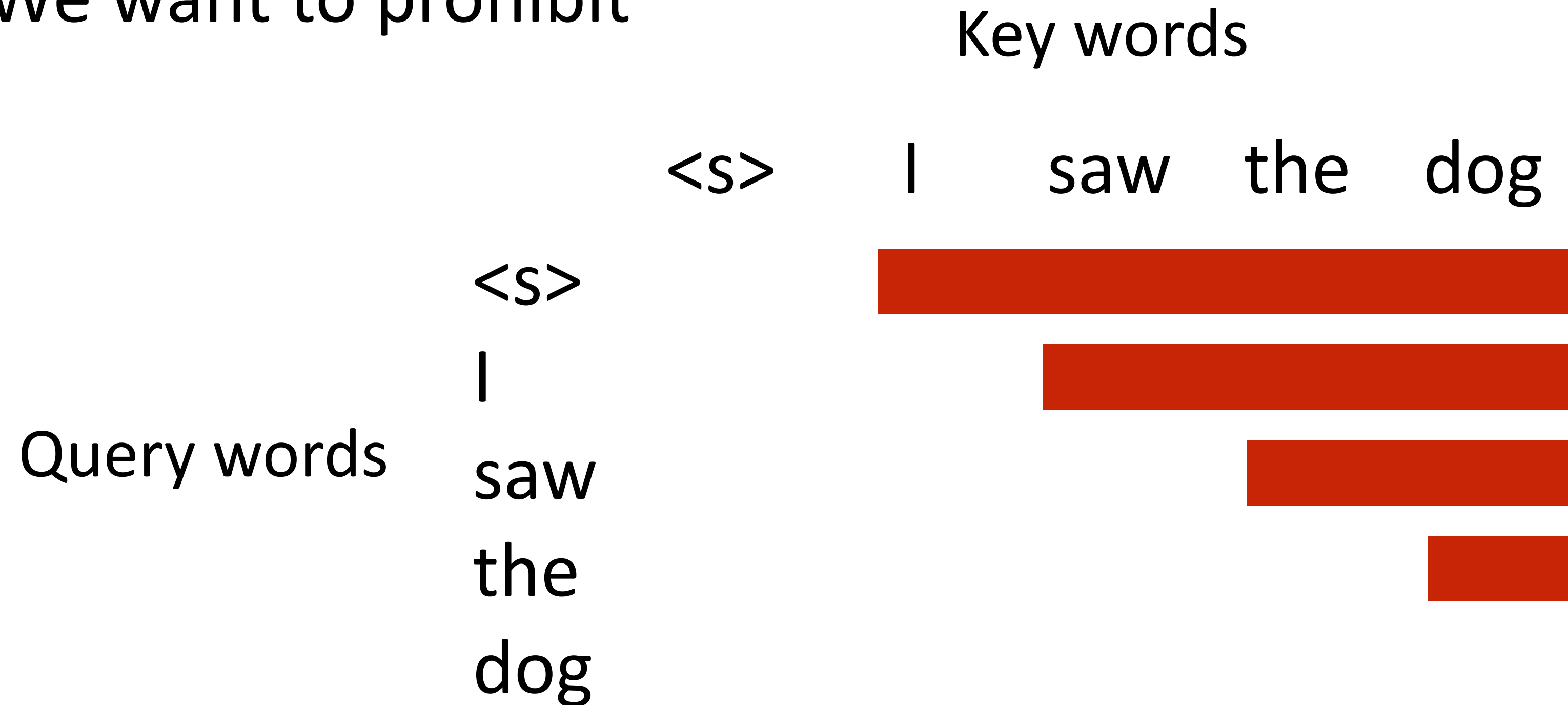


- ▶ With standard self-attention: “I” attends to “saw” and the model is “cheating”. How do we ensure that this doesn't happen?



Attention Masking

- ▶ We want to prohibit



- ▶ We want to mask out everything in red (an upper triangular matrix)



Implementing in PyTorch

- ▶ `nn.TransformerEncoder` can be built out of `nn.TransformerEncoderLayers`, can accept an input and a mask for language modeling:

```
# Inside the module; need to fill in size parameters
layers = nn.TransformerEncoderLayer([...])
transformer_encoder = nn.TransformerEncoder(encoder_layers, num_layers=[...])
[. . .]
# Inside forward(): puts negative infinities in the red part
mask = torch.triu(torch.ones(len, len) * float('-inf'), diagonal=1)
output = transformer_encoder(input, mask=mask)
```

- ▶ **You cannot use these for Part 1, only for Part 2**



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)

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

- ▶ Perplexity: $\exp(\text{average negative log likelihood})$. 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 \Leftarrow geometric mean of denominators

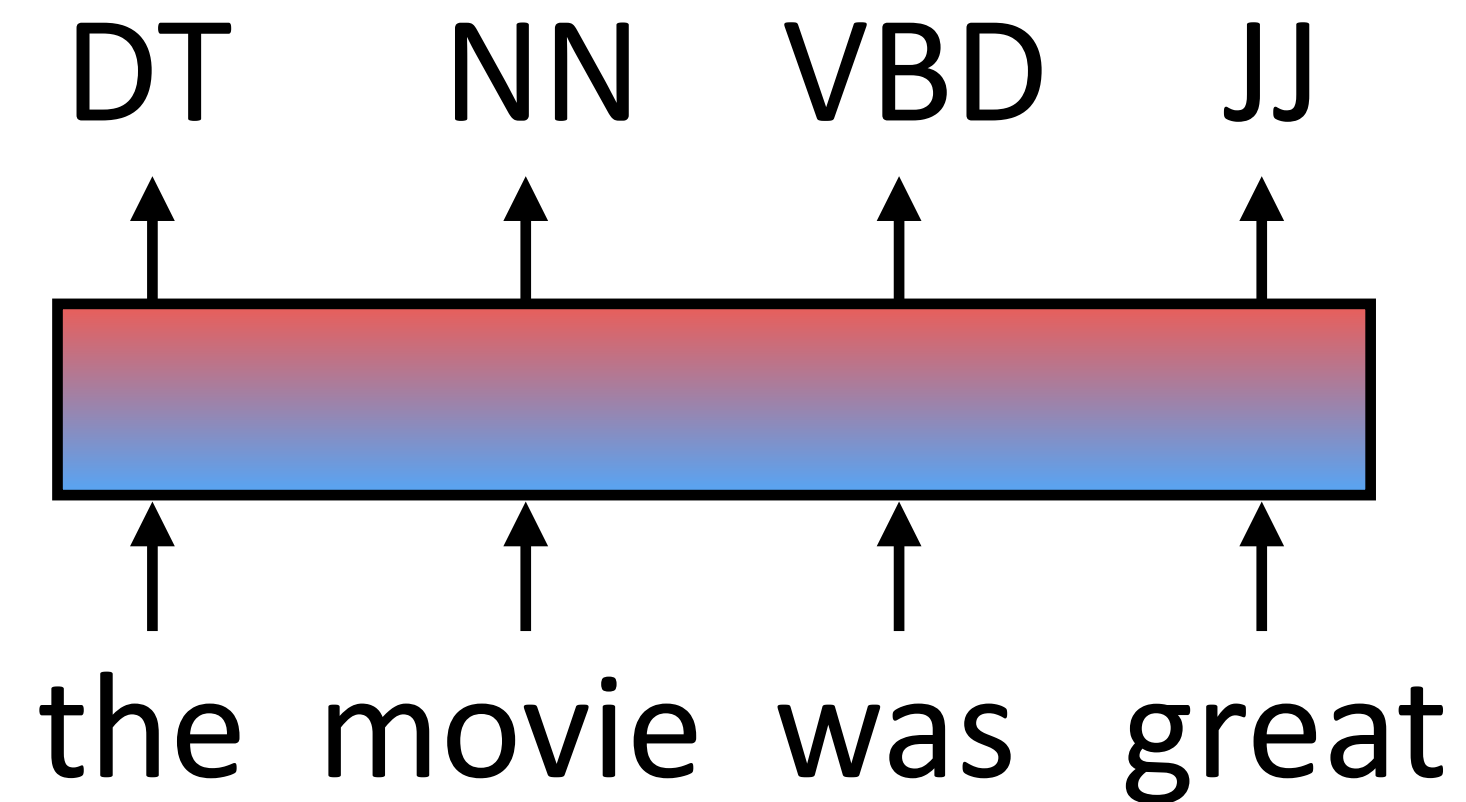
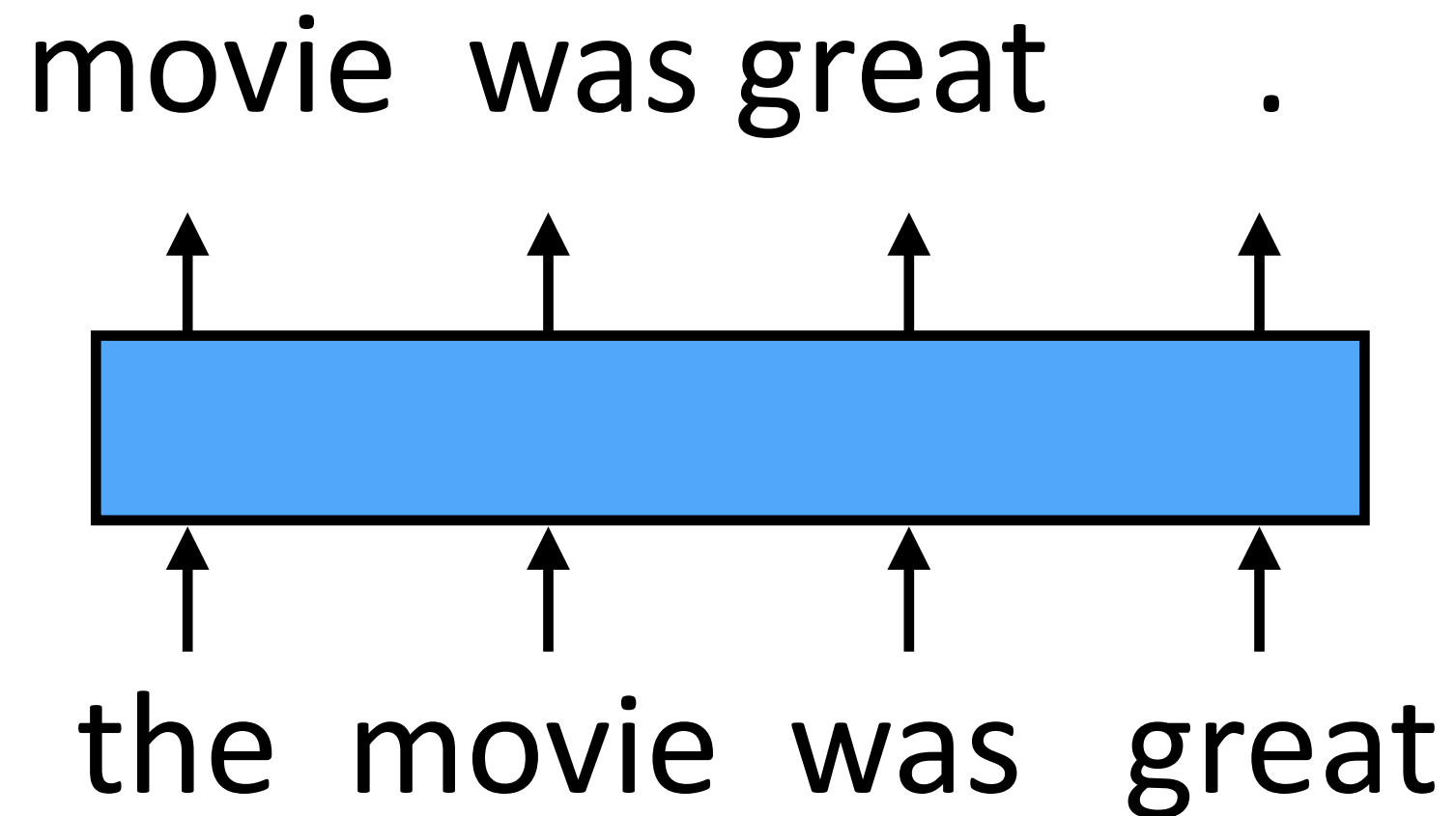


Preview: Pre-training and BERT

- ▶ 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



Transformer Extensions



Scaling Laws

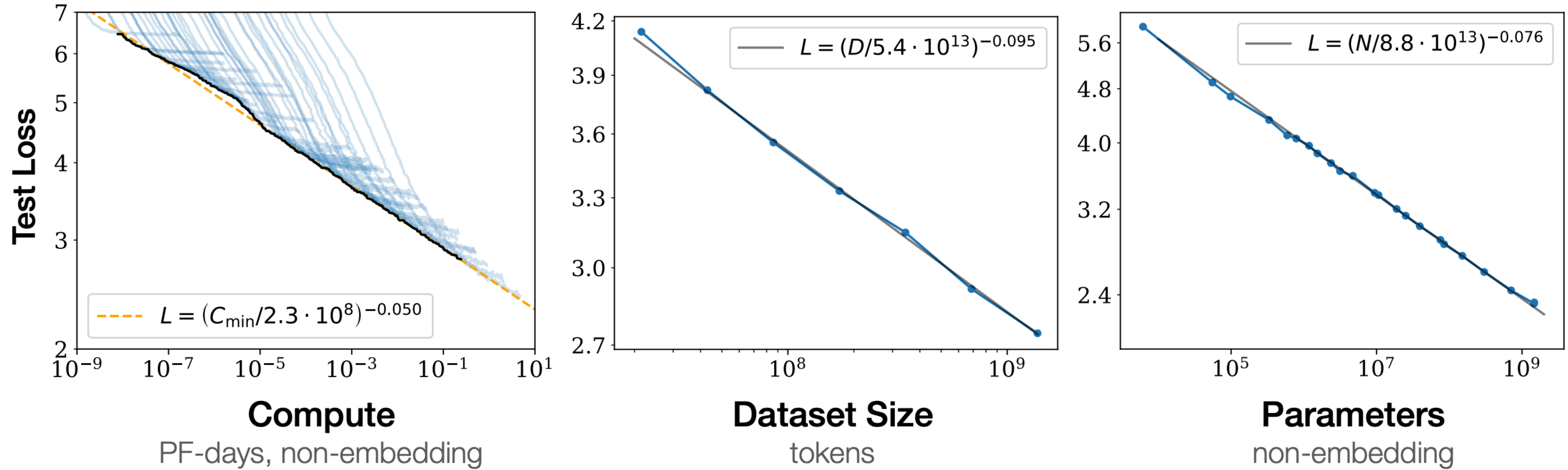


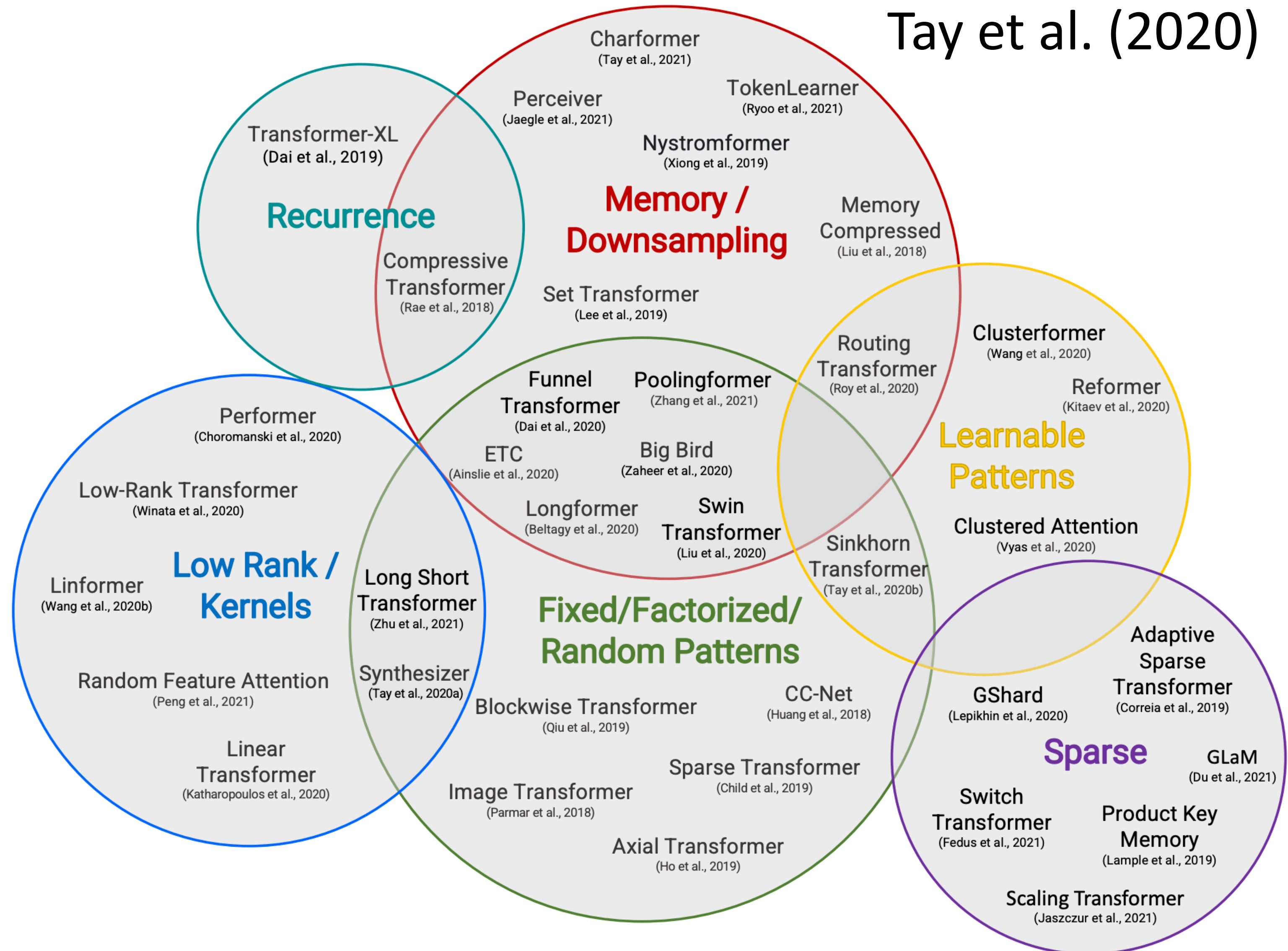
Figure 1 Language modeling performance improves smoothly as we increase the model size, dataset size, and amount of compute² used for training. For optimal performance all three factors must be scaled up in tandem. Empirical performance has a power-law relationship with each individual factor when not bottlenecked by the other two.

- ▶ Transformers scale really well!



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



Performers

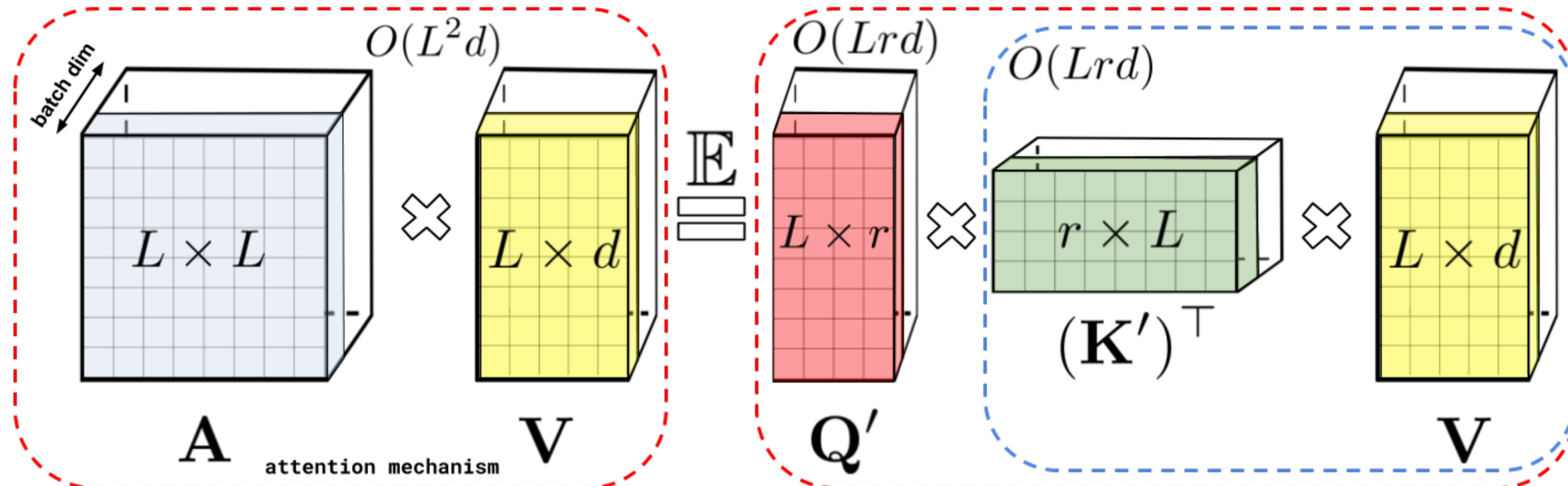
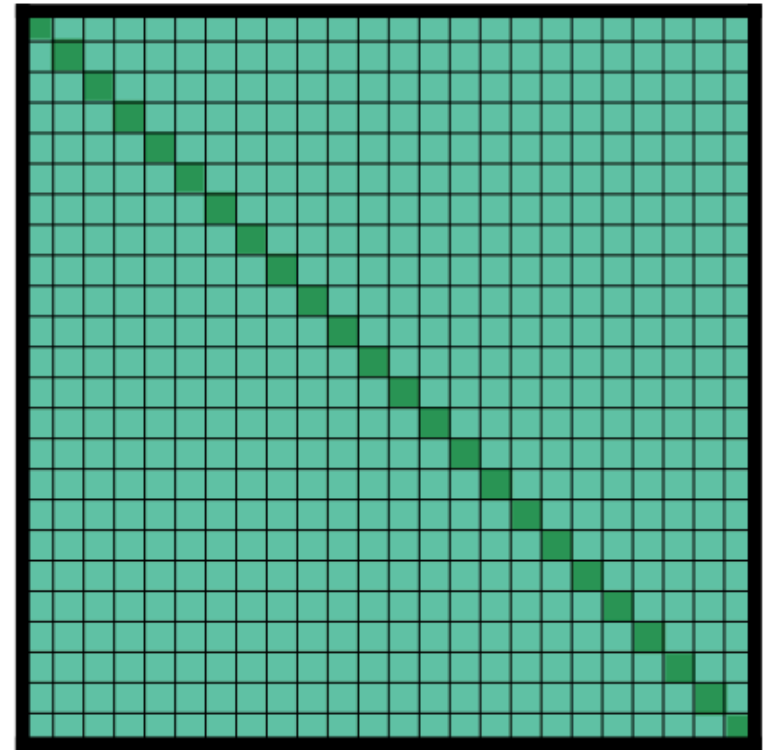


Figure 1: Approximation of the regular attention mechanism $\mathbf{A}\mathbf{V}$ (before \mathbf{D}^{-1} -renormalization) via (random) feature maps. Dashed-blocks indicate order of computation with corresponding time complexities attached.

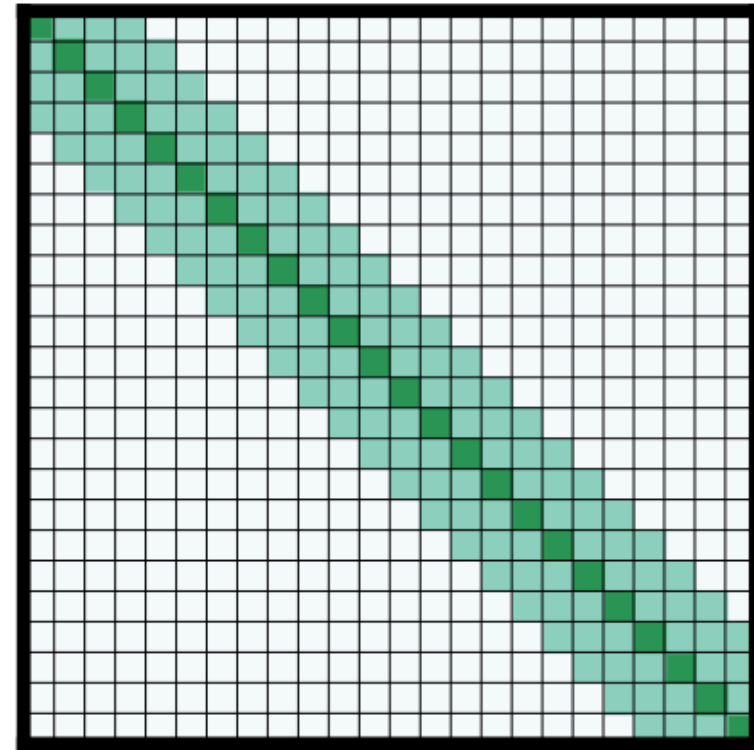
- ▶ No more len^2 term, but we are fundamentally approximating the self-attention mechanism (cannot form \mathbf{A} and take the softmax)



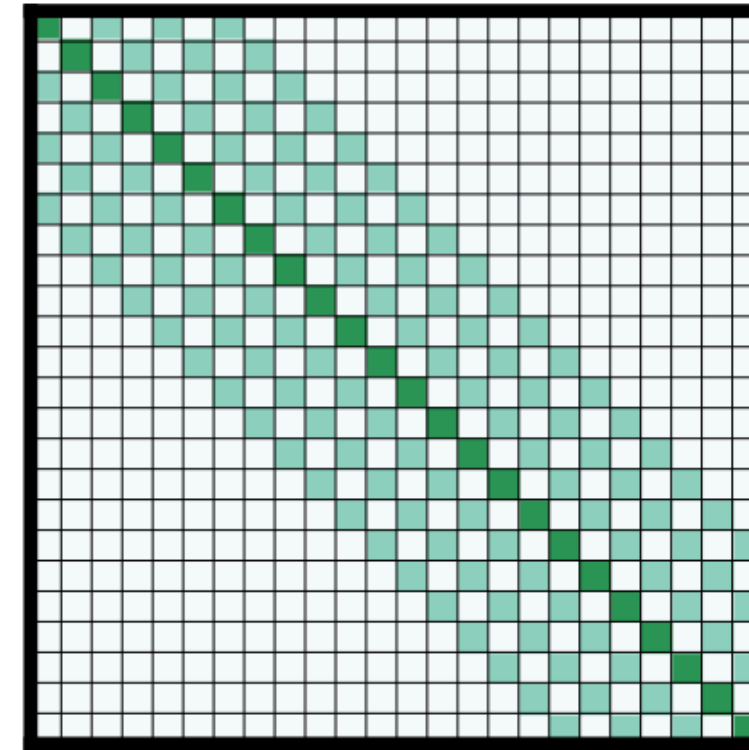
Longformer



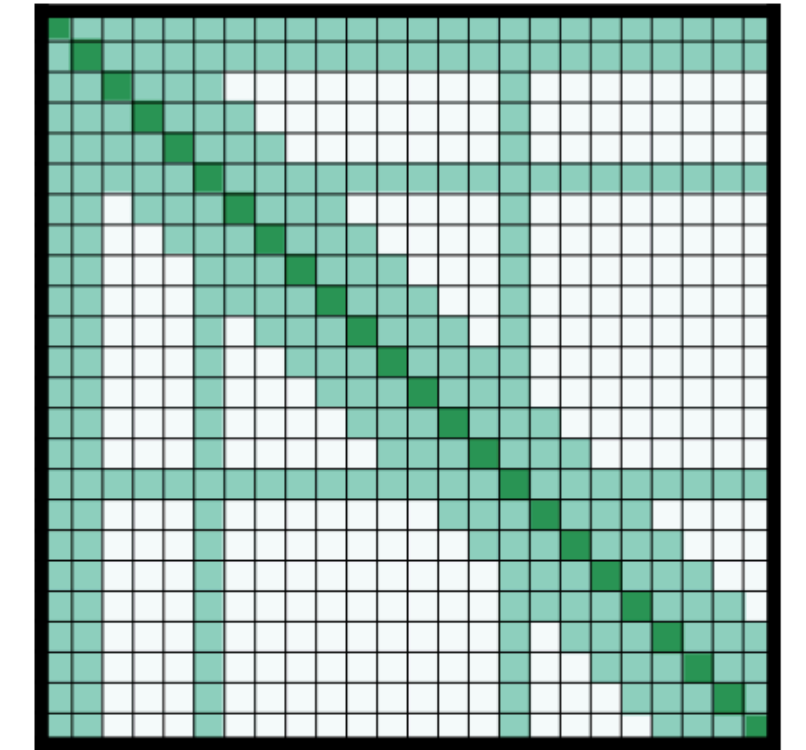
(a) Full n^2 attention



(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



Vision and RL

- ▶ DALL-E 1: learns a discrete “codebook” and treats an image as a sequence of visual tokens which can be modeled autoregressively, then decoded back to an image
- ▶ Decision Transformer: does reinforcement learning by Transformer-based modeling over a series of actions
- ▶ Transformers are now being used all over AI



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
- ▶ Next: machine translation and seq2seq models (conditional language modeling)