Transformer

Wei Xu

(many slides from Greg Durrett)

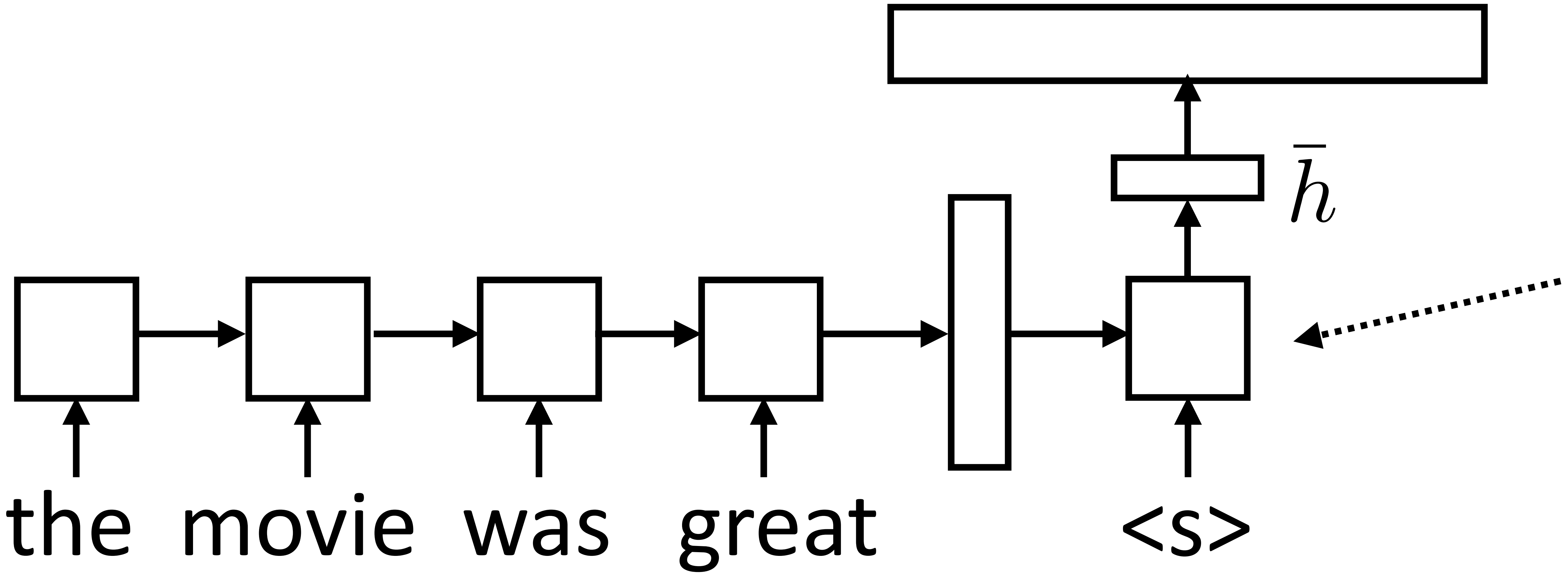
Recap: Encoder-Decoder

- Generate next word conditioned on previous word as well as hidden state

- W size is |vocab| x |hidden state|, softmax over entire vocabulary

$$P(y_i|\mathbf{x}, y_1, \ldots, y_{i-1}) = \mathrm{softmax}(W\bar{h})$$

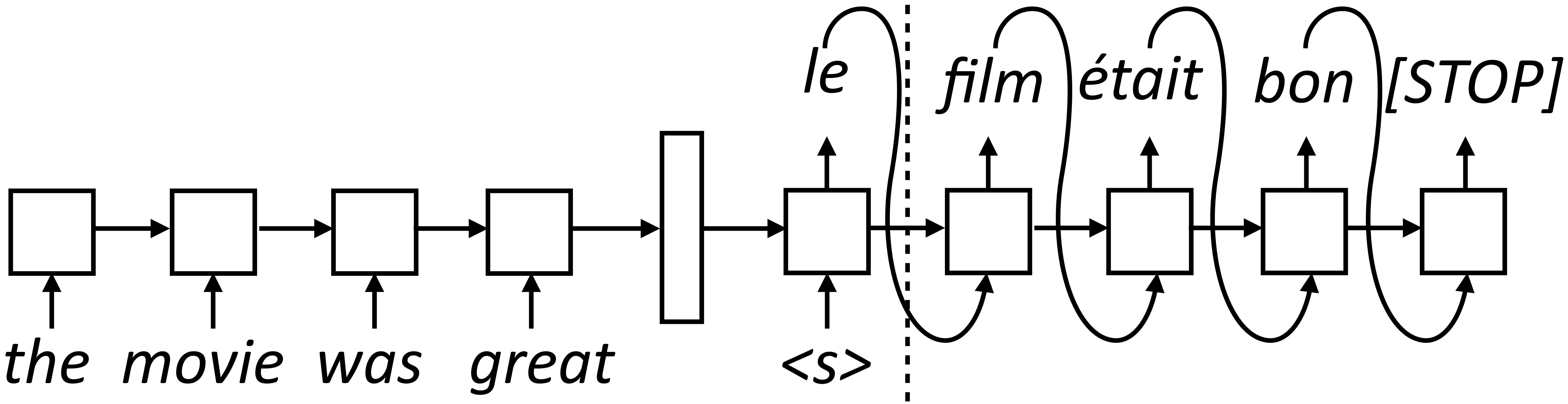
$$P(\mathbf{y}|\mathbf{x}) = \prod_{i=1}^{n} P(y_i|\mathbf{x}, y_1, \ldots, y_{i-1})$$



Decoder has separate parameters from encoder, so this can learn to be a language model (produce a plausible next word given current one)

Recap: Greedy Decoding

- Generate next word conditioned on previous word as well as hidden state

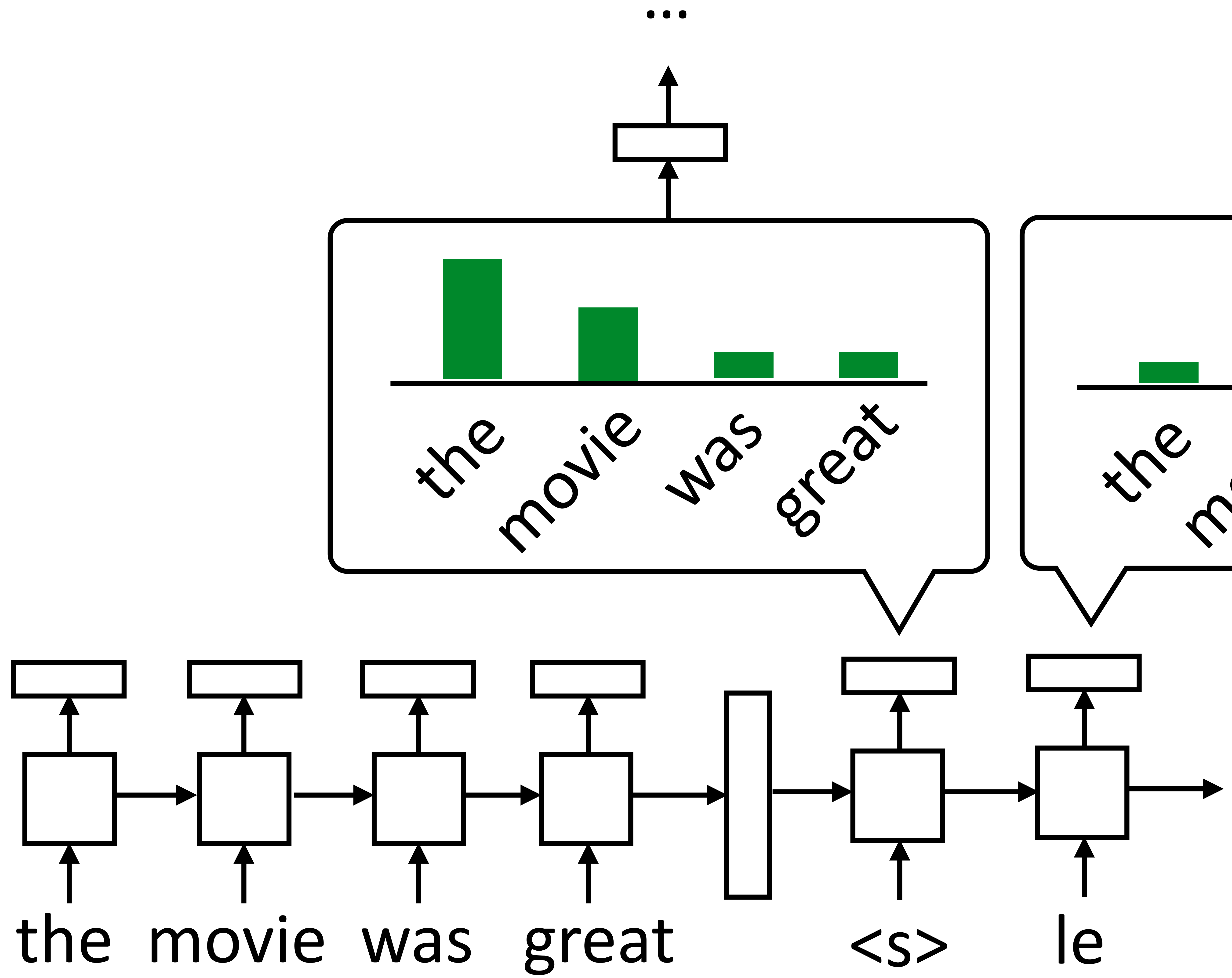


- During inference: need to compute the argmax over the word predictions and then feed that to the next RNN state. This is **greedy decoding**

$$P(y_i|\mathbf{x}, y_1, \ldots, y_{i-1}) = \mathrm{softmax}(W\bar{h}_i)$$

$$y_{\mathrm{pred}} = \mathrm{argmax}_y P(y|\mathbf{x}, y_1, \ldots, y_{i-1})$$

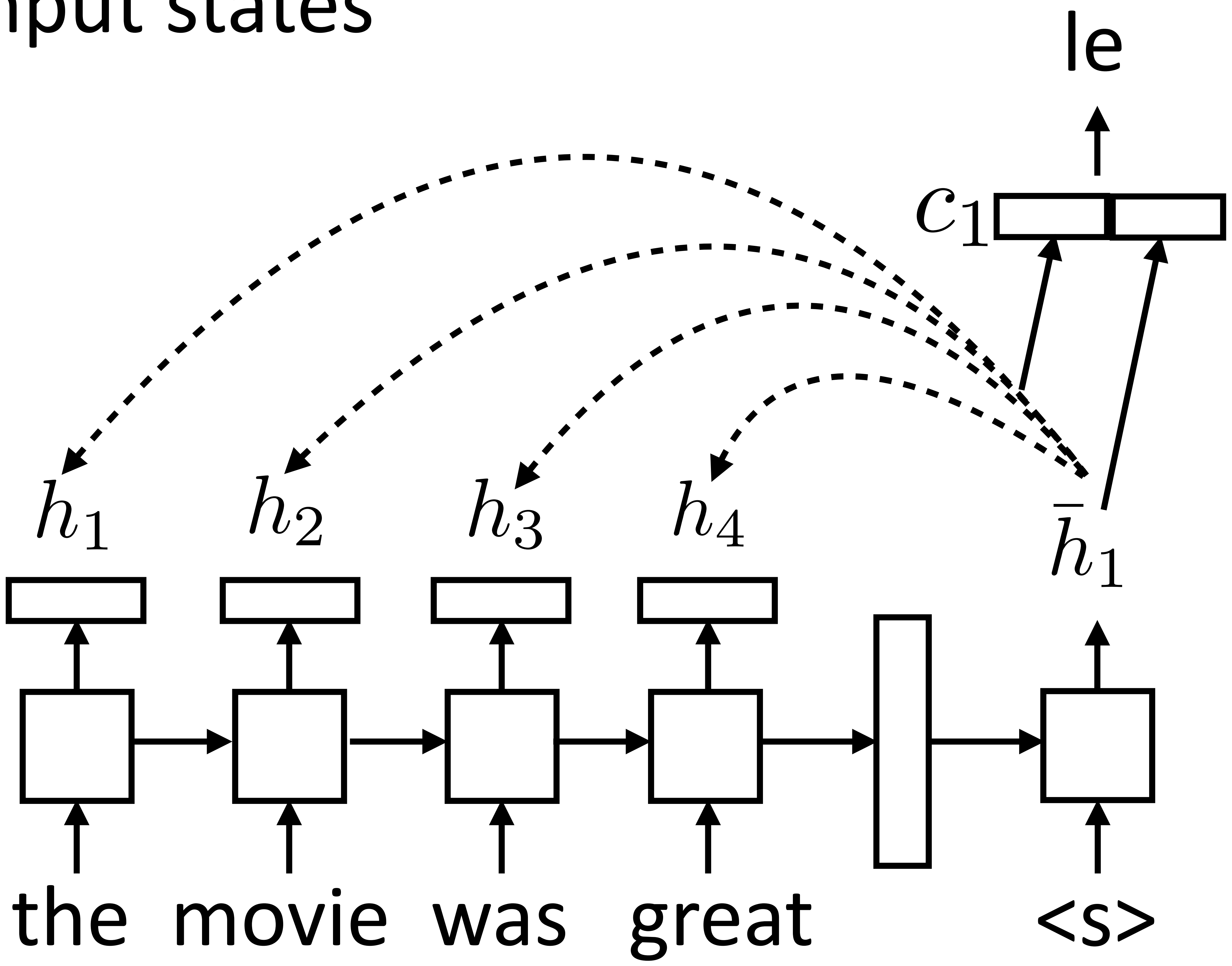
Recap: Attention



- At each decoder state, compute a distribution over source inputs based on current decoder state
- Use the weighted sum of input tokens to predict output

Recap: Attention

- For each decoder state, compute weighted sum of input states



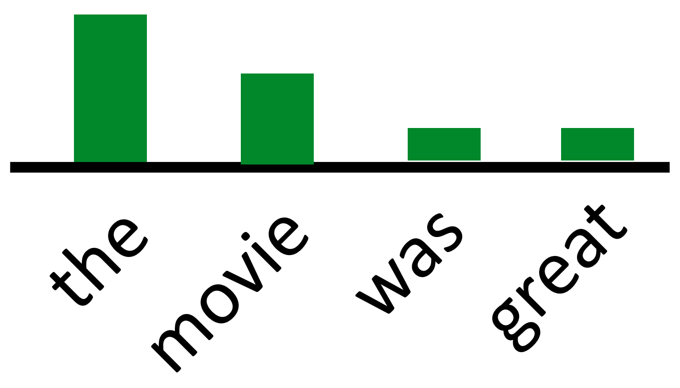
- No attn: $P(y_i|\mathbf{x}, y_1, \ldots, y_{i-1}) = \mathrm{softmax}(W\bar{h}_i)$

$$P(y_i|\mathbf{x}, y_1, \ldots, y_{i-1}) = \mathrm{softmax}(W[c_i; \bar{h}_i])$$

$$c_i = \sum_j \alpha_{ij} h_j$$

- Weighted sum of input hidden states (vector)

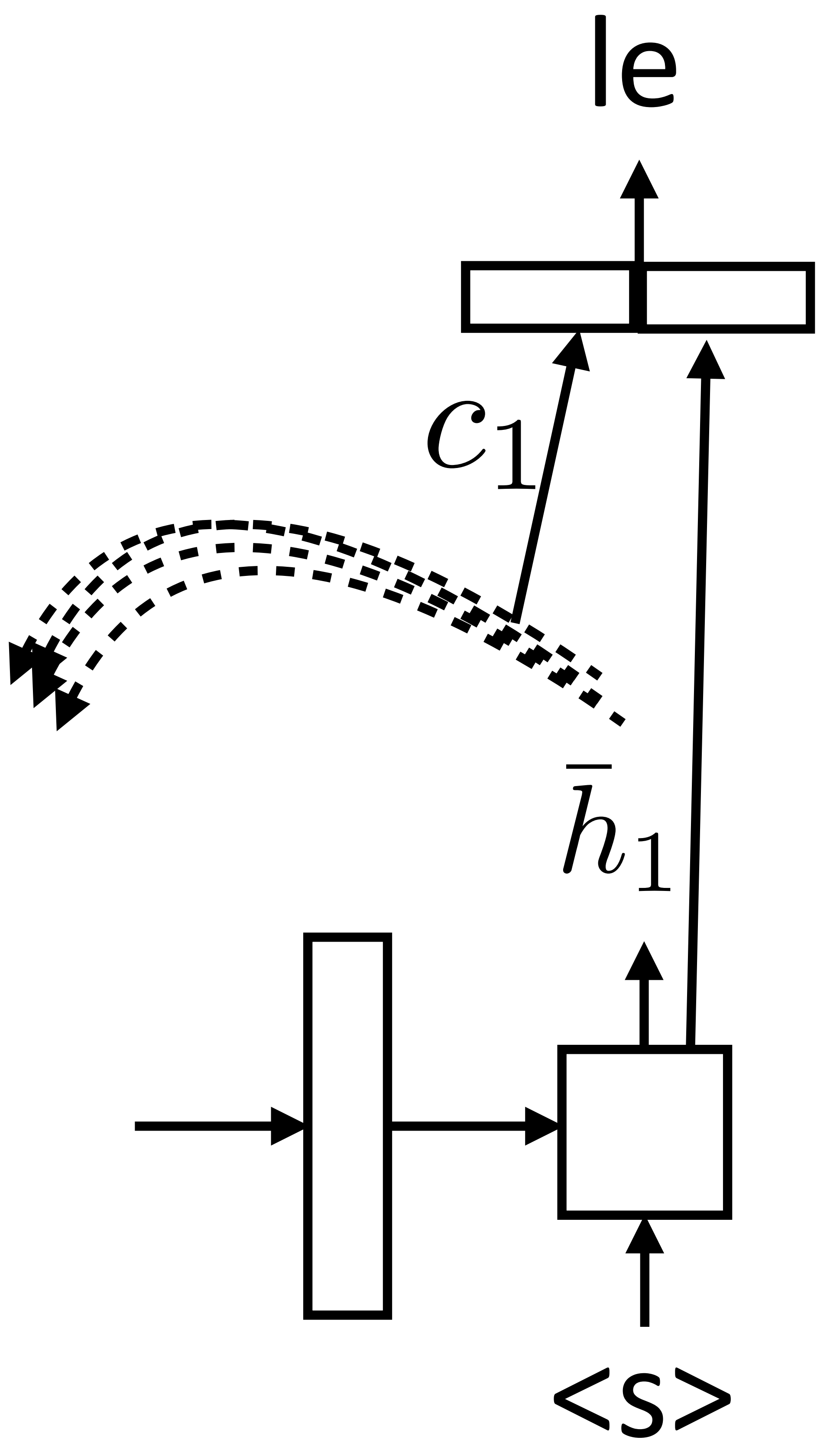
$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{j'} \exp(e_{ij'})}$$



$$e_{ij} = f(\bar{h}_i, h_j)$$

- Some function f (e.g., dot product)

Recap: Attention



$$c_i = \sum_j \alpha_{ij} h_j$$

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{j'} \exp(e_{ij'})}$$

$$e_{ij} = f(\bar{h}_i, h_j)$$

$$f(\bar{h}_i, h_j) = \tanh(W[\bar{h}_i; h_j])$$

- Bahdanau+ (2014): additive

$$f(\bar{h}_i, h_j) = \bar{h}_i \cdot h_j$$

- Luong+ (2015): dot product

$$f(\bar{h}_i, h_j) = \bar{h}_i^\top W h_j$$

- Luong+ (2015): bilinear

- Note that this all uses outputs of hidden layers

Transformers

Attention is All You Need

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 best models from the literature. We show that the Transformer generalizes well to other tasks by applying it successfully to English constituency parsing both with large and limited training data.

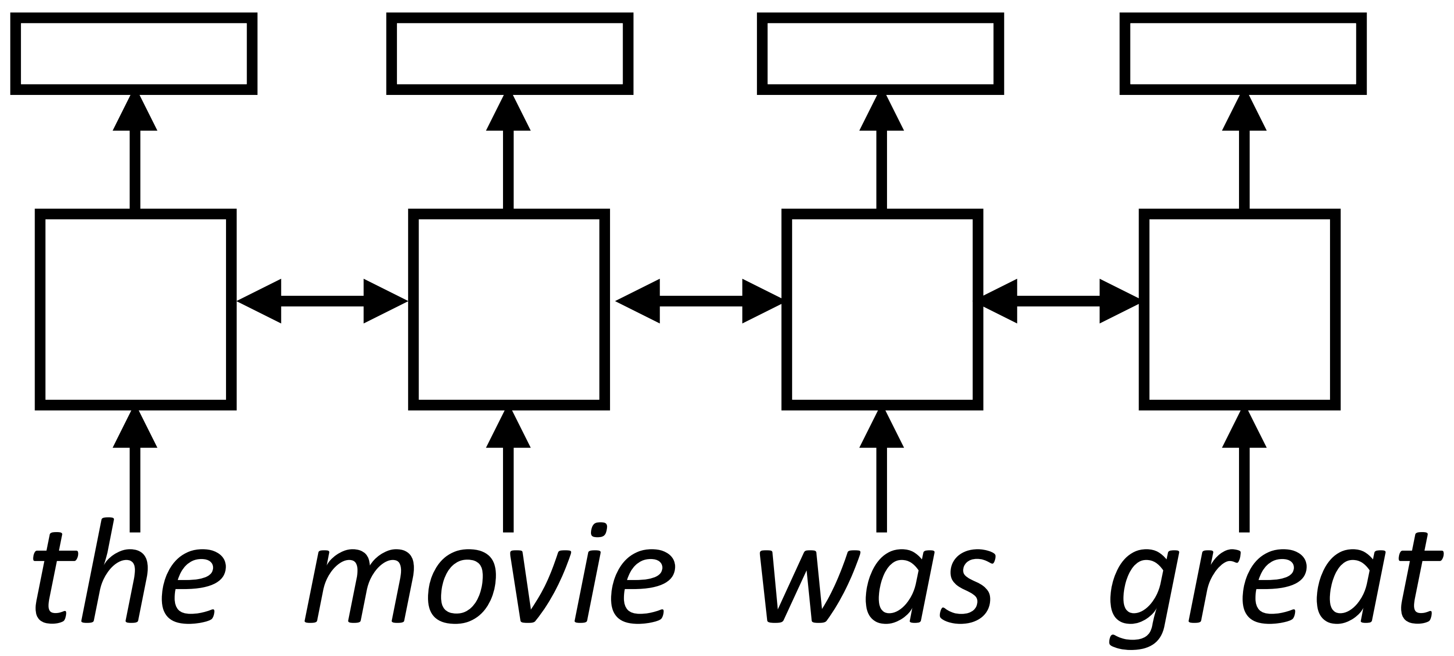
Vaswani et al. (2017)

Readings

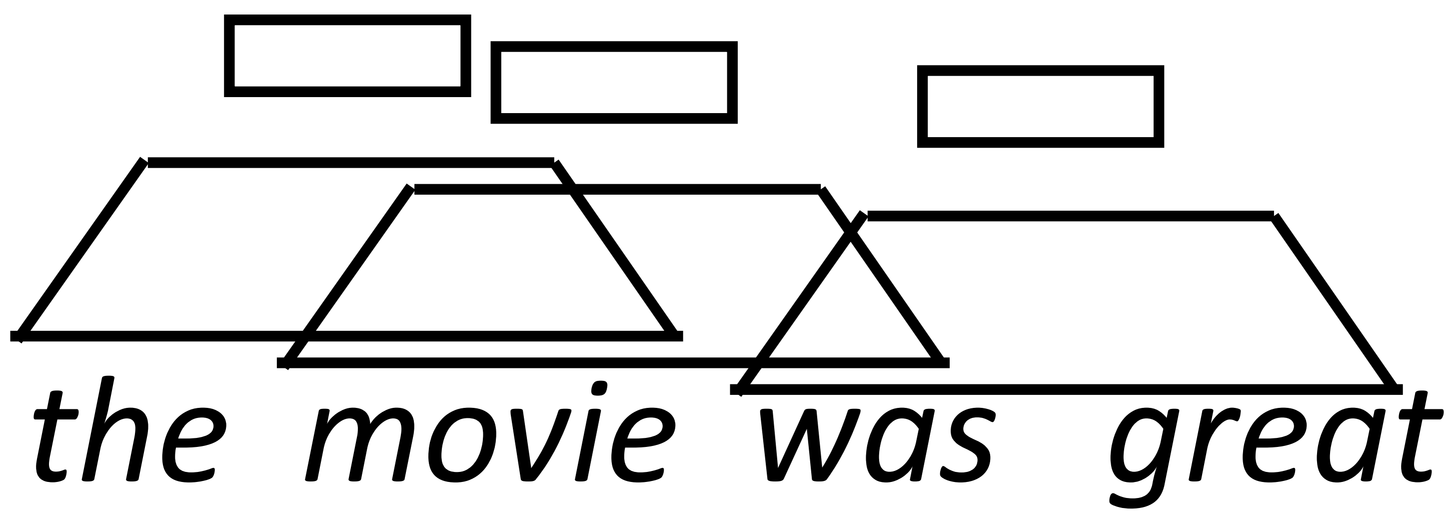
- "The Annotated Transformer" by Sasha Rush
  https://nlp.seas.harvard.edu/2018/04/03/attention.html
- "The Illustrated Transformer" by Jay Lamar
  http://jalammar.github.io/illustrated-transformer/
- Jurafsky+Martin Chapter 9

Sentence Encoders

- LSTM abstraction: maps each vector in a sentence to a new, context-aware vector



- CNNs do something similar with filters



- Attention can give us a third way to do this

Vaswani et al. (2017)

Self-Attention

- Assume we're using GloVe/word2vec embeddings — what do we want our neural network to do?

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

- Q: What words need to be contextualized here?

Vaswani et al. (2017)

Self-Attention

- Assume we're using GloVe/word2vec embeddings — what do we want our neural network to do?

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

- What words need to be contextualized here?
  - Pronouns need to look at antecedents
  - Ambiguous words should look at context
  - Words should look at syntactic parents/children
- Problem: LSTMs and CNNs don't do this

Vaswani et al. (2017)

Self-Attention

- Want:

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

- LSTMs/CNNs: tend to look at local context

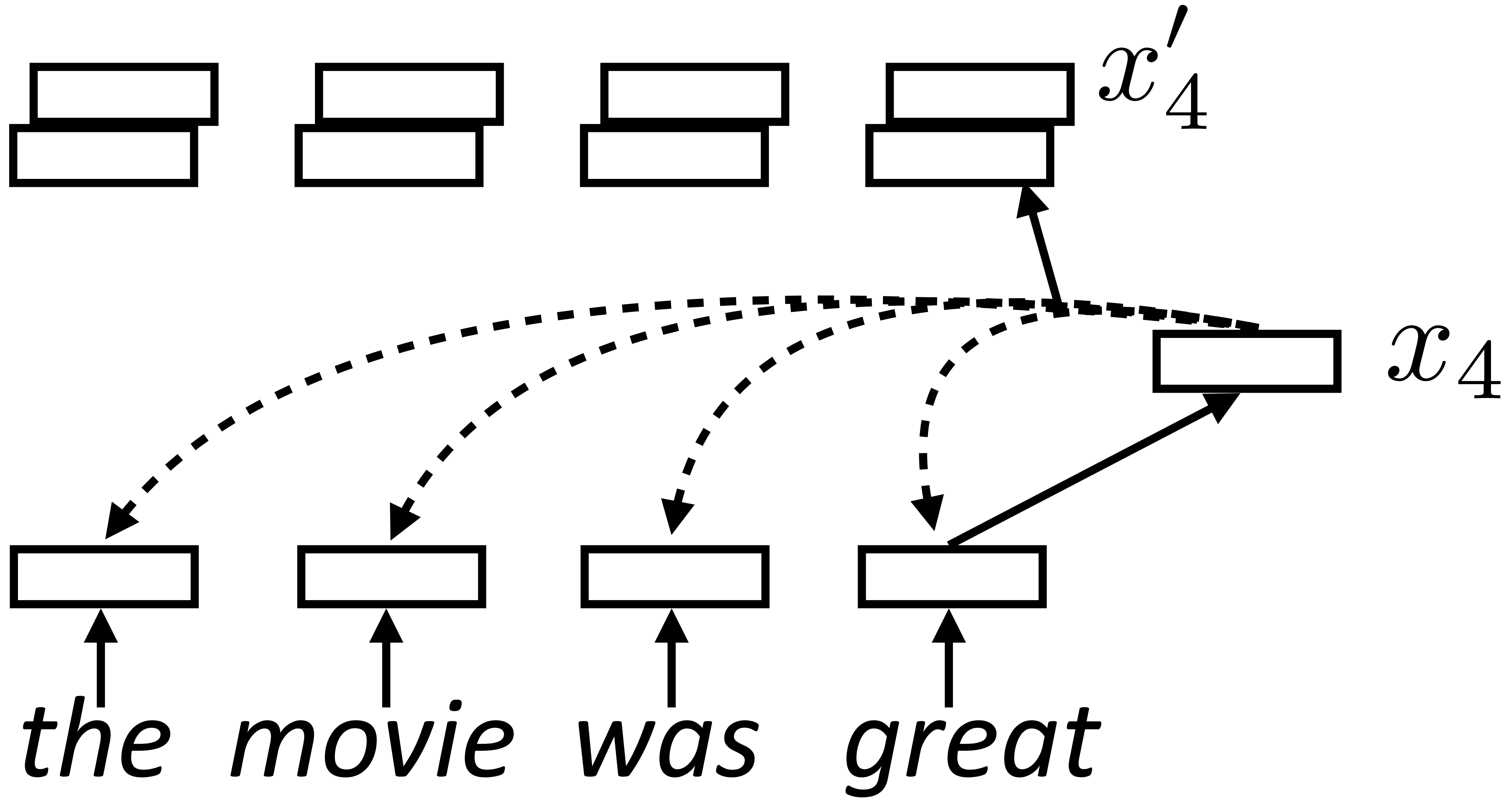
The ballerina is very excited that she will dance in the show.

- To appropriately contextualize embeddings, we need to pass information over long distances dynamically for each word

Vaswani et al. (2017)

Self-Attention

- Each word forms a "query" which then computes attention over each word



$$\alpha_{i,j} = \text{softmax}(x_i^\top x_j) \quad \text{scalar}$$

$$x'_i = \sum_{j=1}^{n} \alpha_{i,j} x_j \quad \text{vector = sum of scalar * vector}$$

- Multiple "heads" analogous to different convolutional filters. Use parameters W_k and V_k to get different attention values + transform vectors

$$\alpha_{k,i,j} = \text{softmax}(x_i^\top W_k x_j) \qquad x'_{k,i} = \sum_{j=1}^{n} \alpha_{k,i,j} V_k x_j$$

Vaswani et al. (2017)

What can self-attention do?

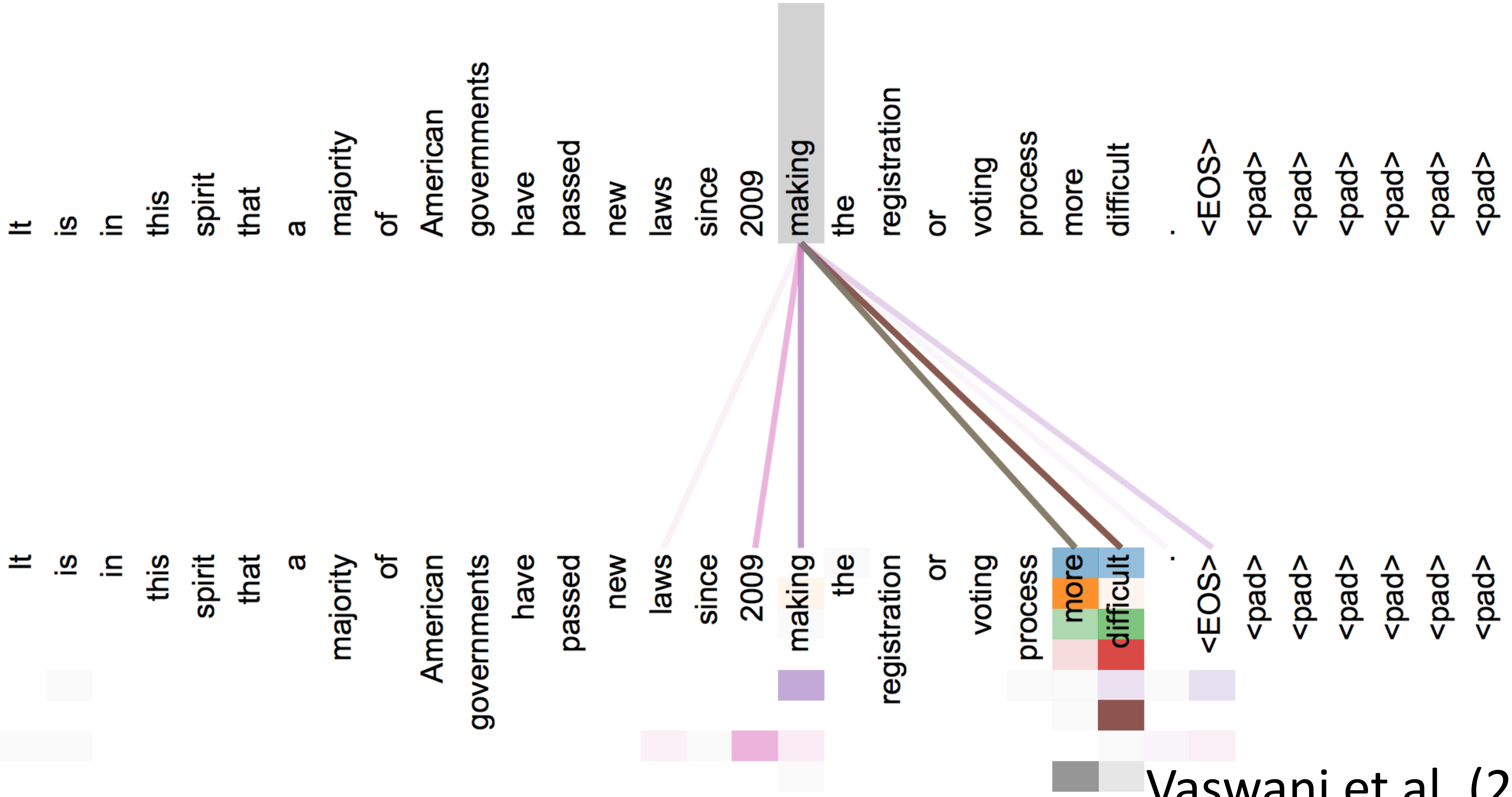
The ballerina is very excited that she will dance in the show.

0	0.5	0	0	0.1	0.1	0	0.1	0.2	0	0	0
0	0.1	0	0	0	0	0	0	0.5	0	0.4	0

- Attend nearby + to semantically related terms
- Why multiple heads? Softmaxes end up being peaked, single distribution cannot easily put weight on multiple things

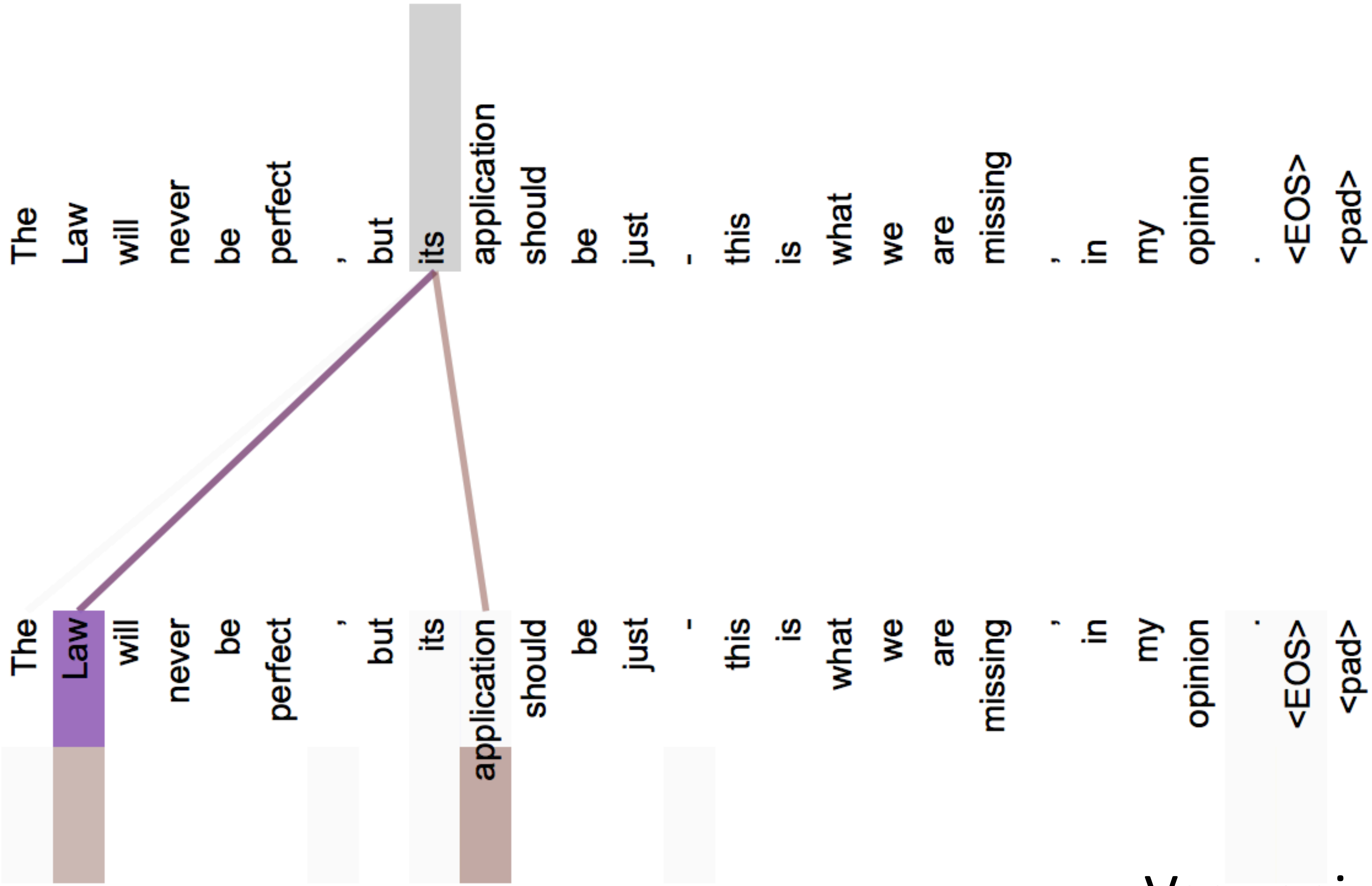
Vaswani et al. (2017)

Visualization



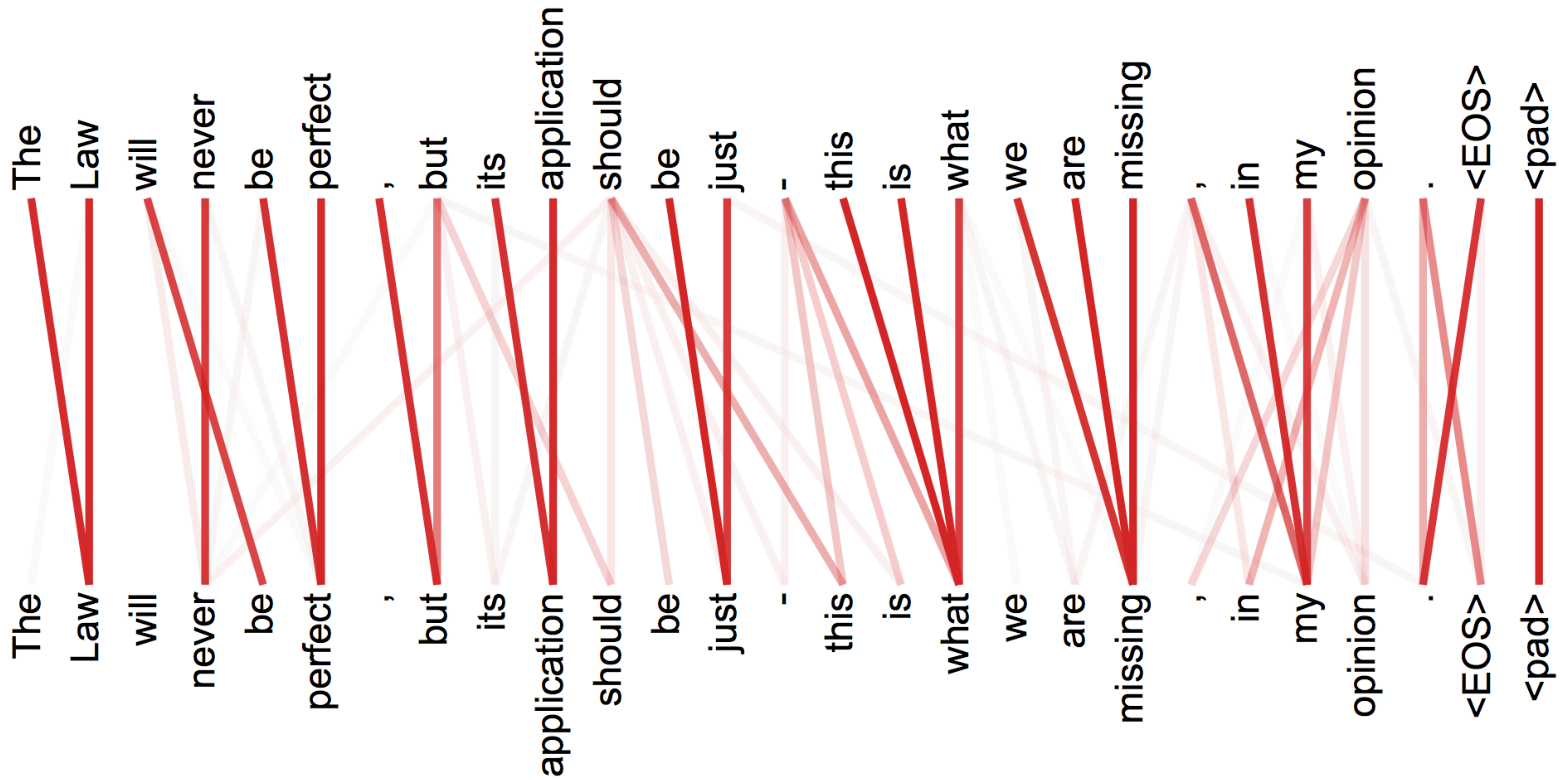
Vaswani et al. (2017)

Visualization



Vaswani et al. (2017)

Visualization



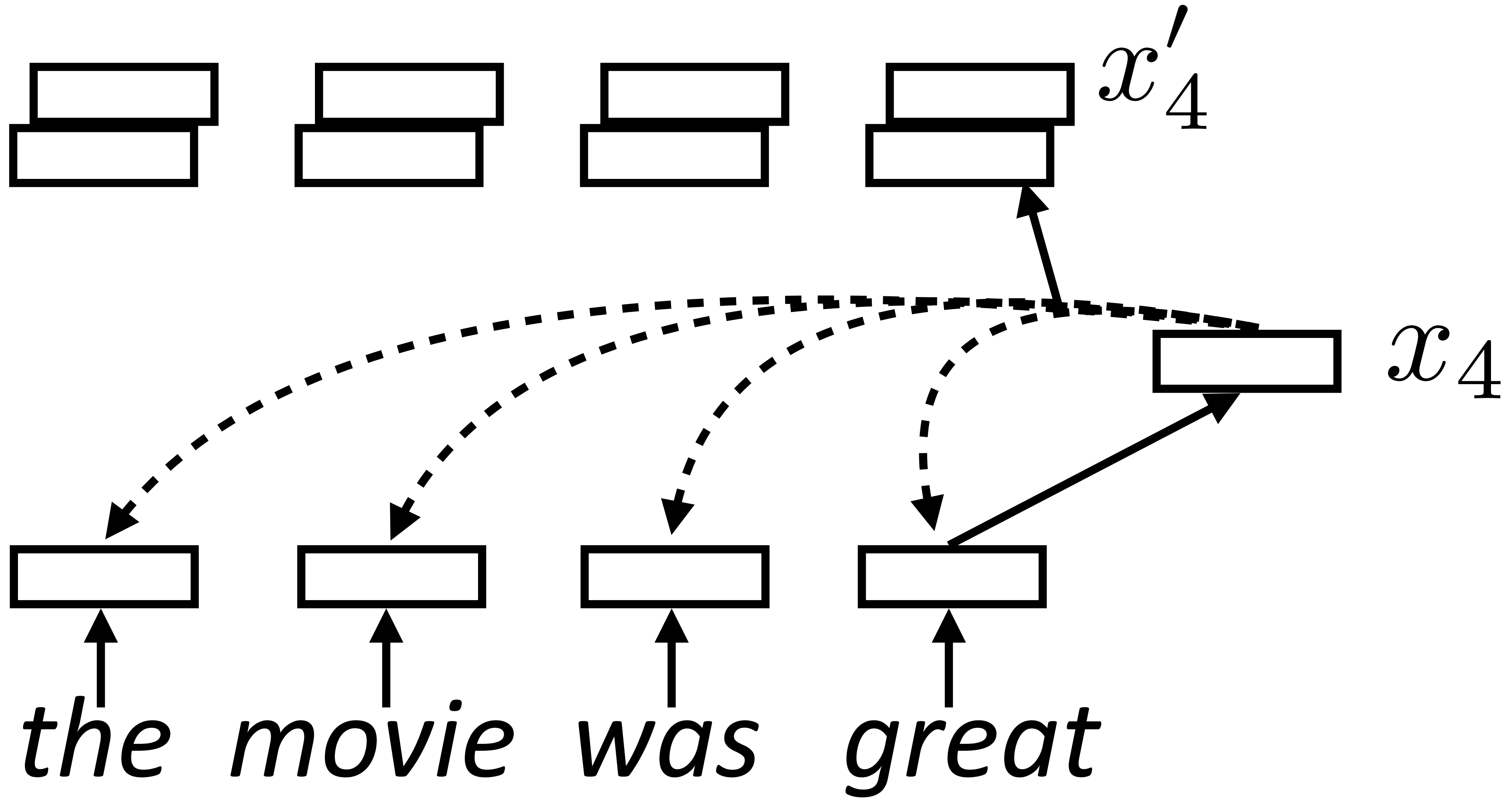
Vaswani et al. (2017)

Self-Attention

- Each word forms a "query" which then computes attention over each word

$$\alpha_{i,j} = \text{softmax}(x_i^\top x_j) \quad \text{scalar}$$

$$x'_i = \sum_{j=1}^{n} \alpha_{i,j} x_j \quad \text{vector = sum of scalar * vector}$$



- Multiple "heads" analogous to different convolutional filters. Use parameters W_k and V_k to get different attention values + transform vectors

$$\alpha_{k,i,j} = \text{softmax}(x_i^\top W_k x_j) \qquad x'_{k,i} = \sum_{j=1}^{n} \alpha_{k,i,j} V_k x_j$$

Vaswani et al. (2017)

Multi-Head Self Attention

- Multiple "heads" analogous to different convolutional filters
- Let X = [sent len, embedding dim] be the input sentence
- Query $Q = XW^Q$: these are like the **decoder hidden state** in attention
- Keys $K = XW^K$: these control what gets attended to, along with the query
- Values $V = XW^V$: these vectors get summed up to form the output

$$\mathrm{Attention}(Q, K, V) = \mathrm{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

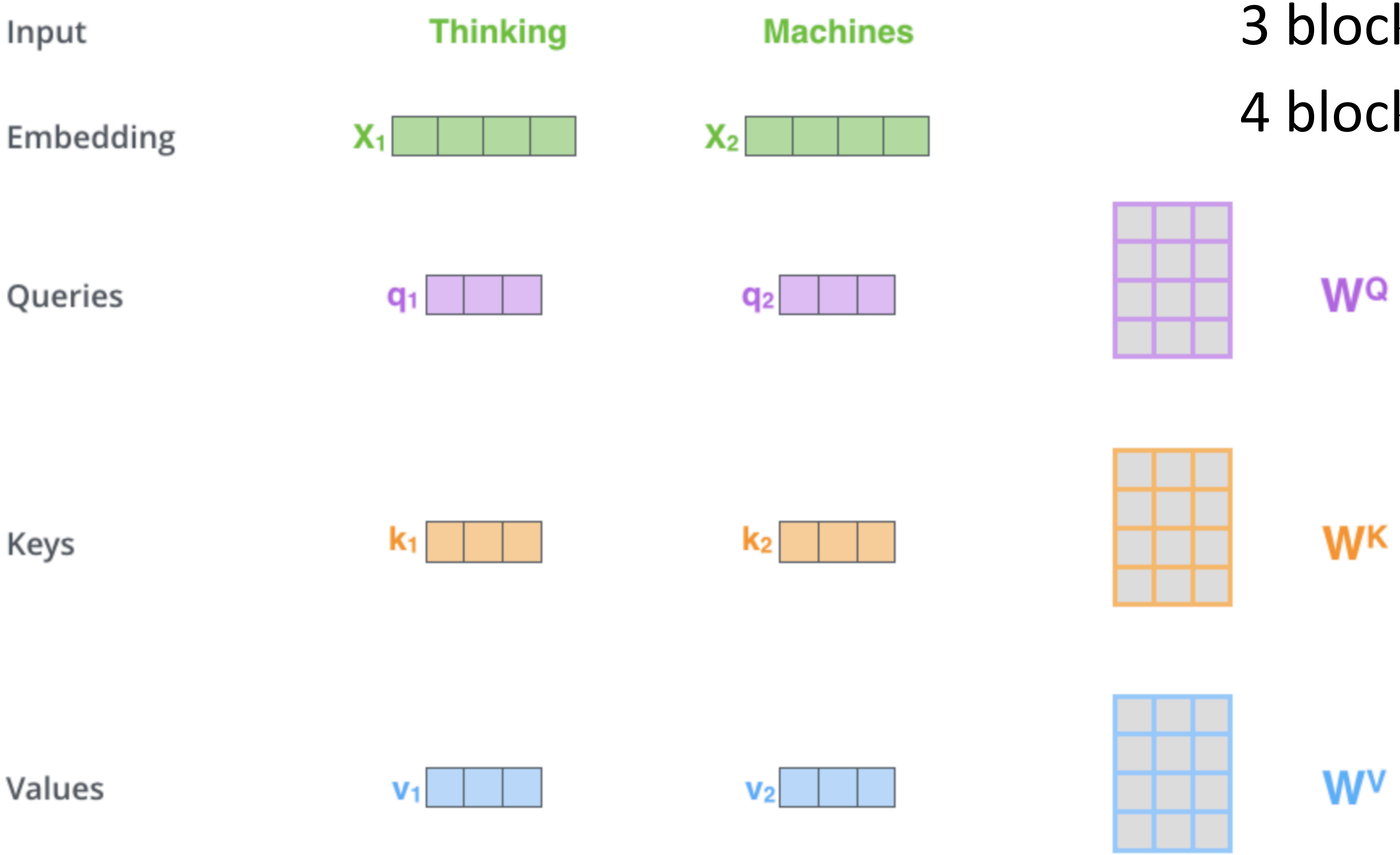
dim of keys

Vaswani et al. (2017)

Multi-Head Self Attention

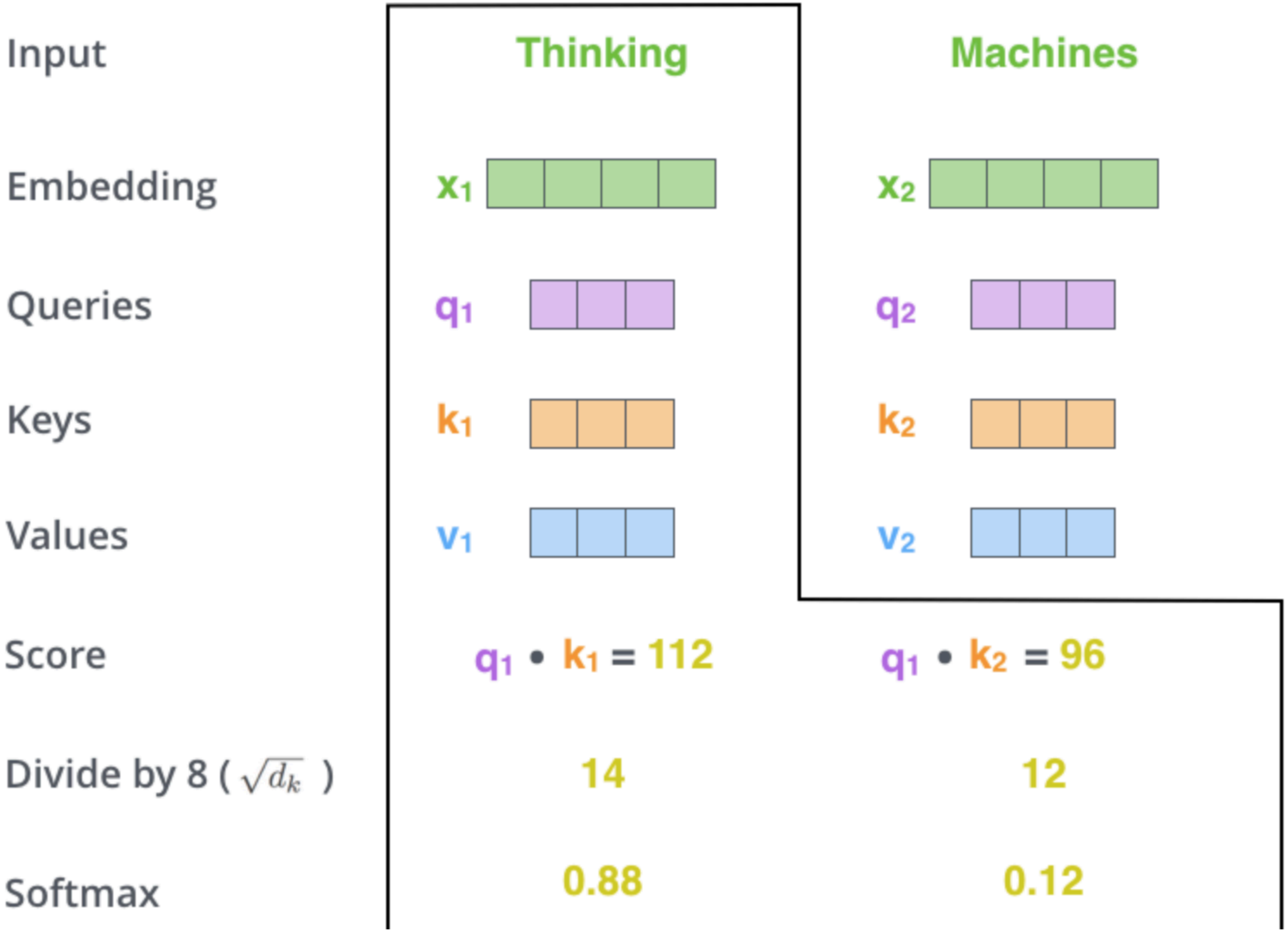
3 blocks - 64 dim (d_k)

4 blocks - 512 dim (d_x)



Credit: Alammar, *The Illustrated Transformer*

Multi-Head Self Attention



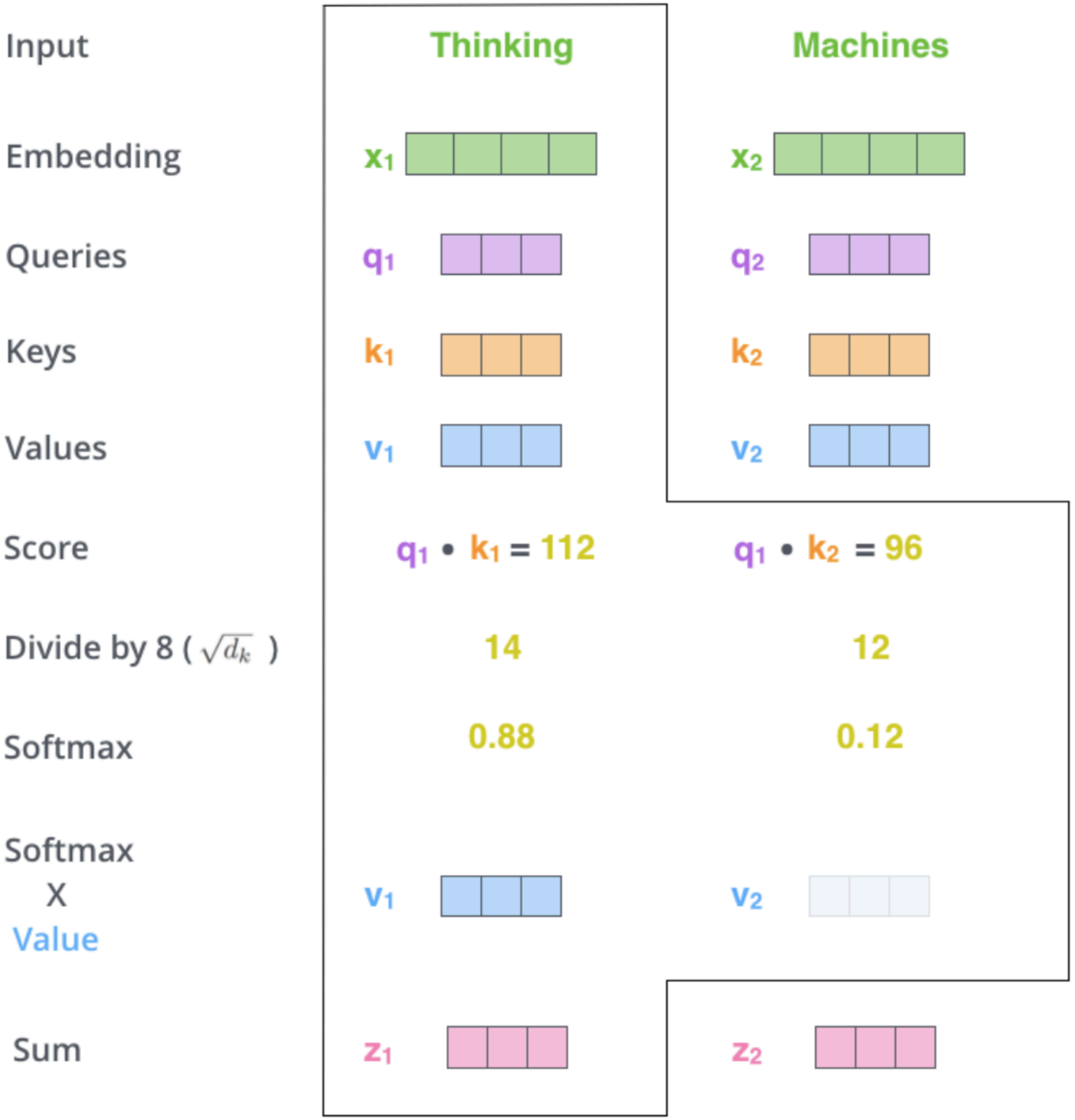
3 blocks - 64 dim (d_k)

4 blocks - 512 dim (d_x)

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

Credit: Alammar, *The Illustrated Transformer*

Multi-Head Self Attention



3 blocks - 64 dim (d_k)

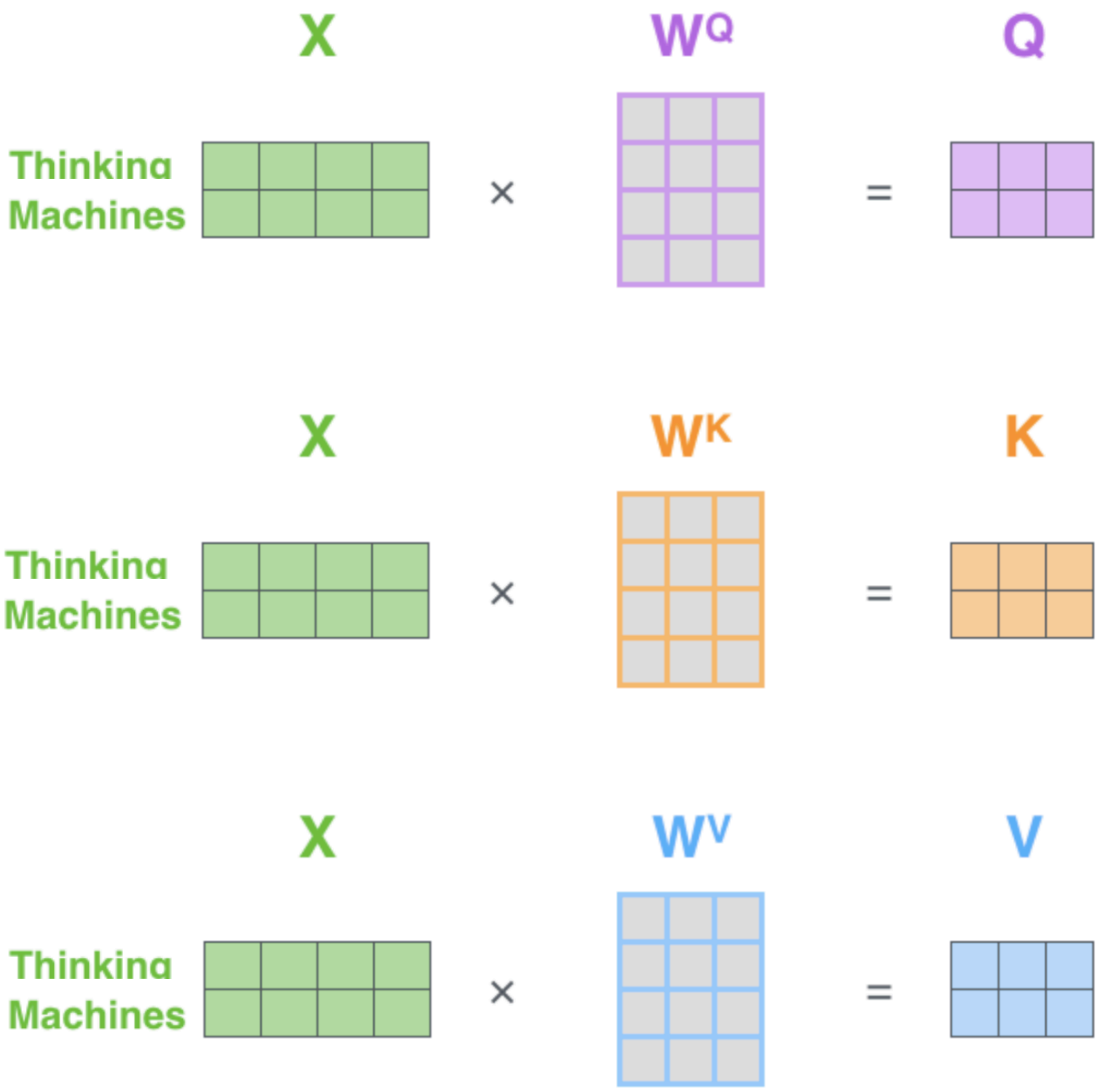
4 blocks - 512 dim (d_x)

$$\mathrm{Attention}(Q,K,V) = \mathrm{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

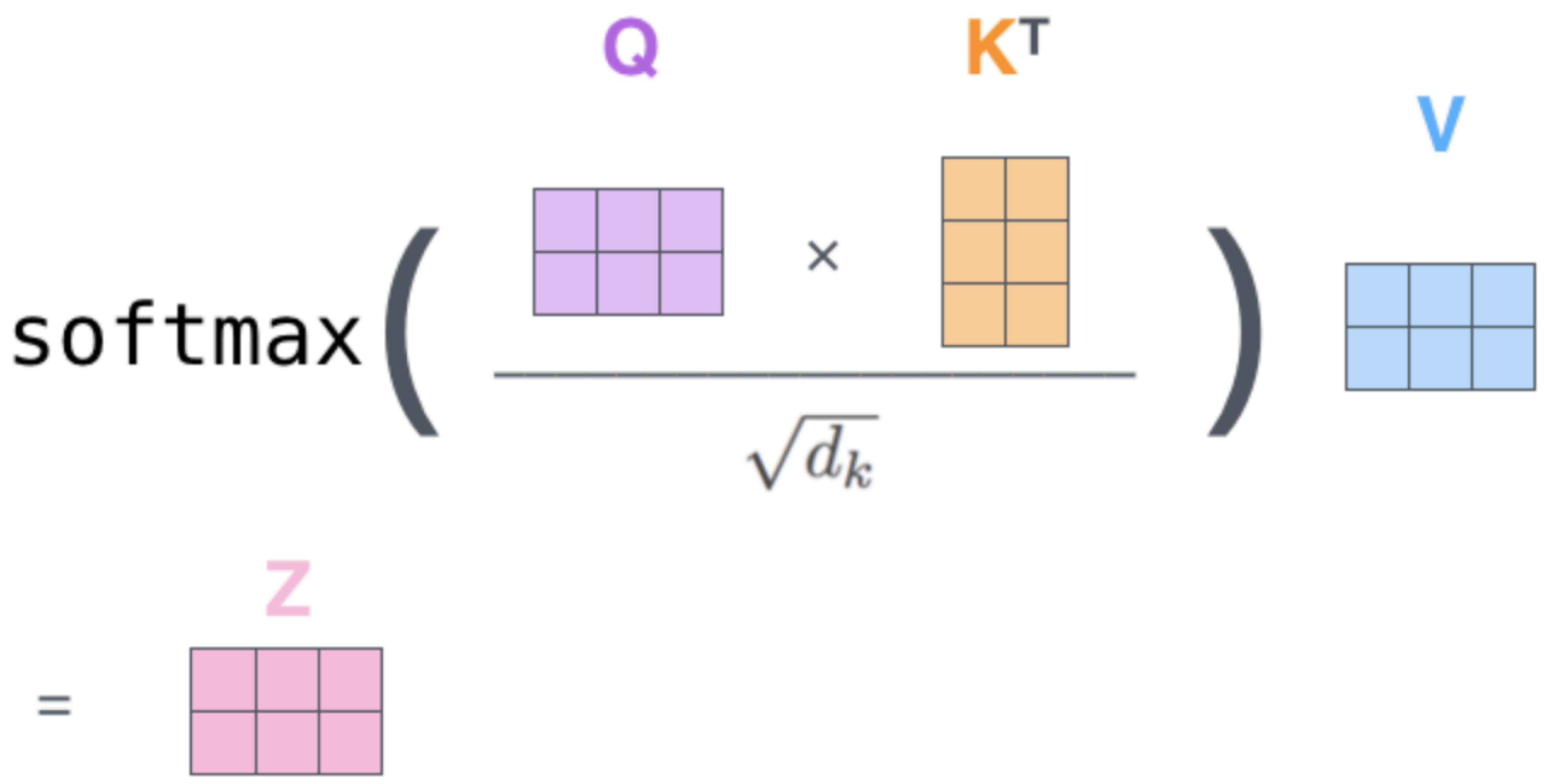
Credit: Alammar, *The Illustrated Transformer*

Multi-Head Self Attention

every row in X is a word in input sent



sent len x sent len (attn for each word to each other)

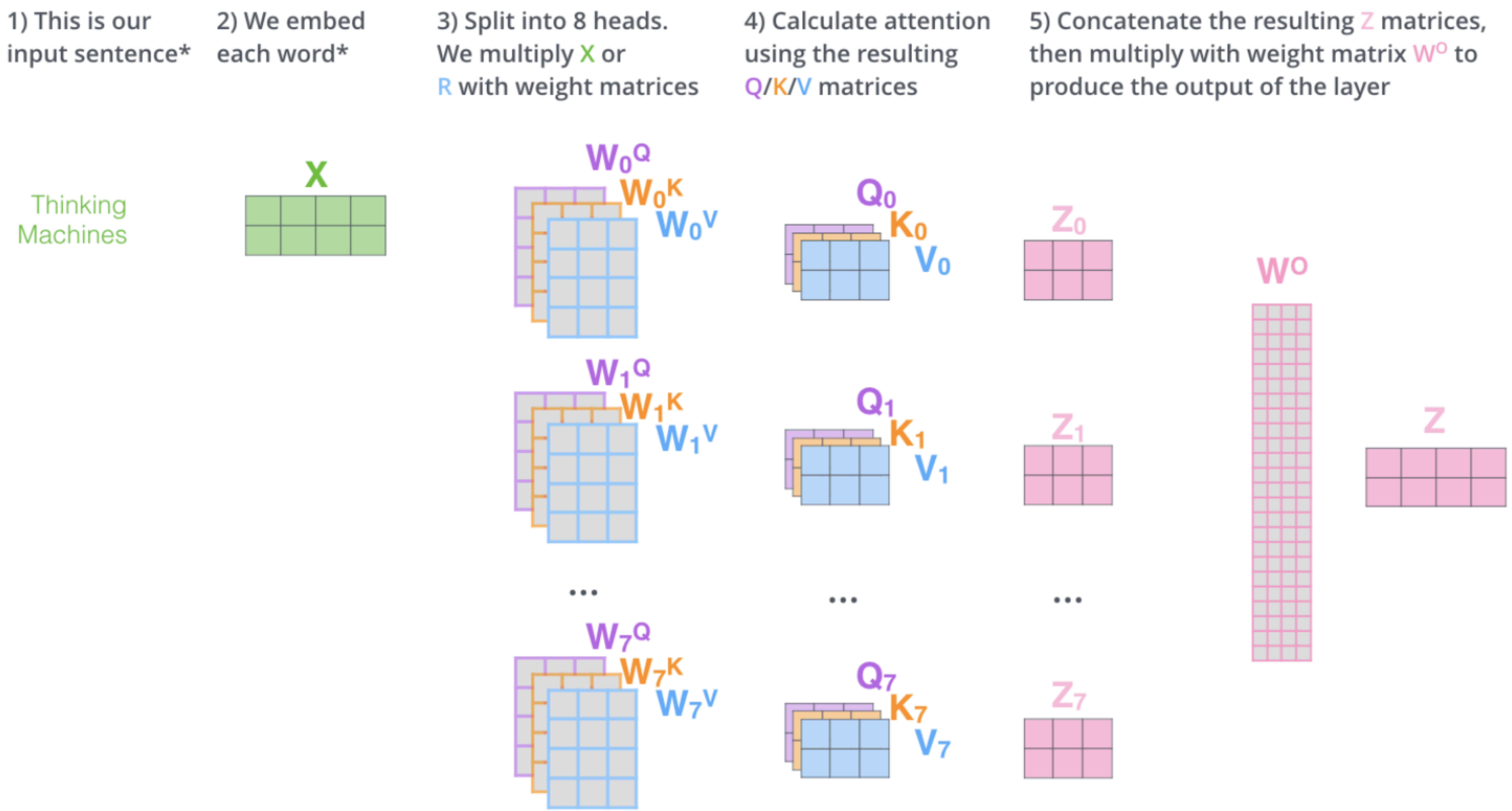


sent len x hidden dim

Z is a weighted combination of V rows

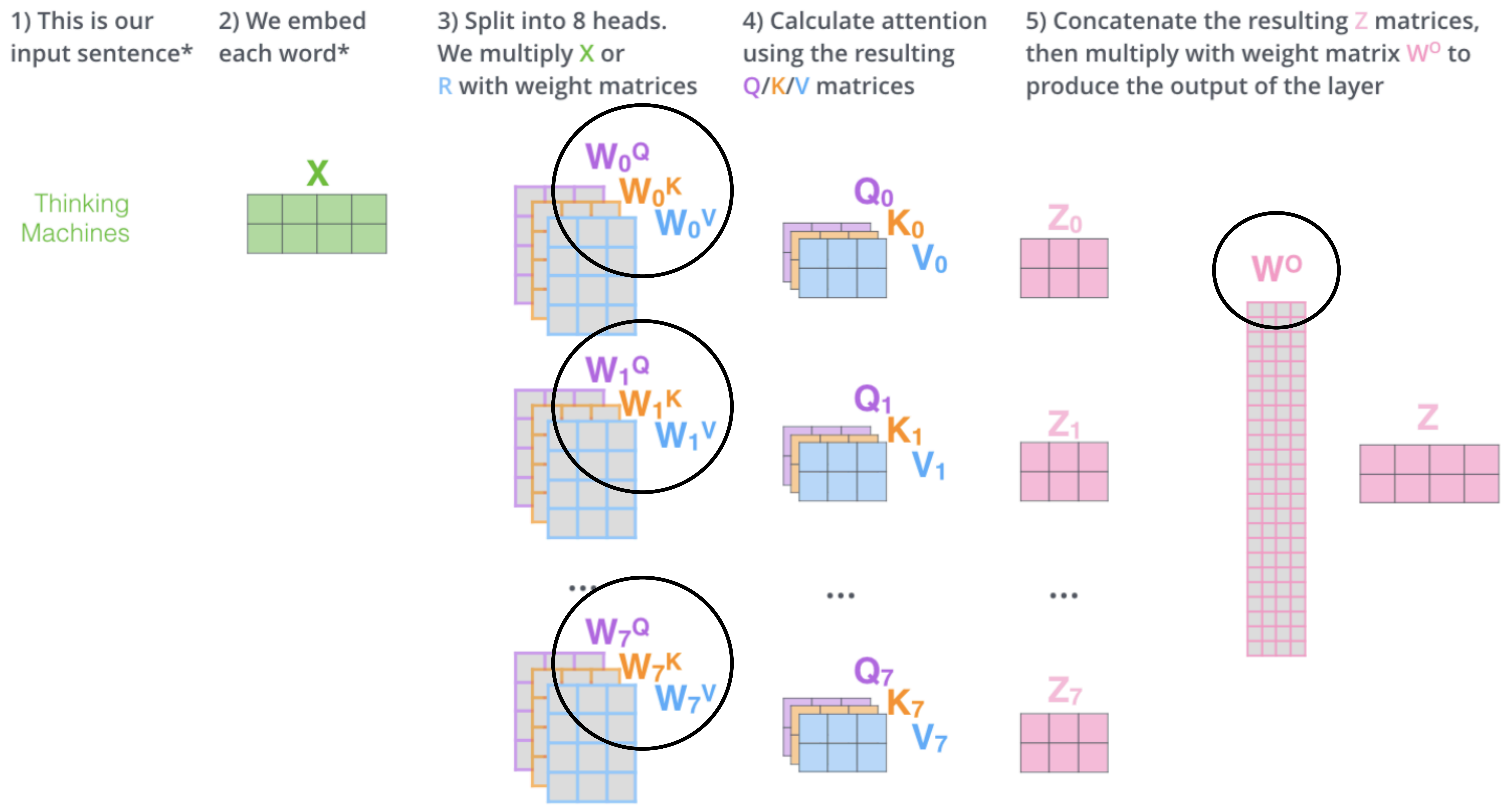
Credit: Alammar, *The Illustrated Transformer*

Multi-Head Self Attention



Credit: Alammar, *The Illustrated Transformer*

Multi-Head Self Attention



Credit: Alammar, *The Illustrated Transformer*

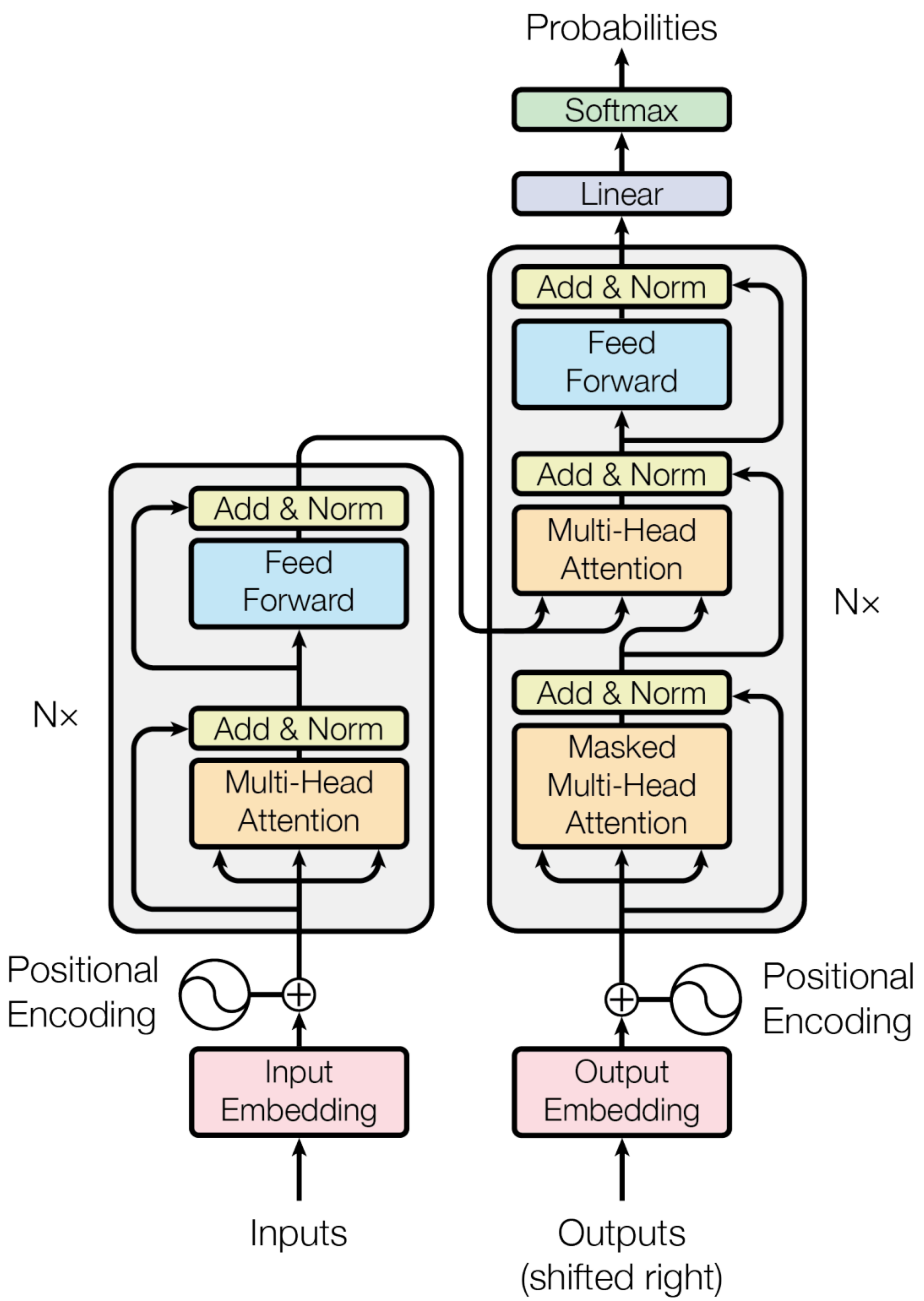
Properties of Self-Attention

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

- *n* = sentence length, *d* = hidden dim, *k* = kernel size, *r* = restricted neighborhood size
- **Quadratic complexity**, but O(1) sequential operations (not linear like in RNNs) and O(1) "path" for words to inform each other

Vaswani et al. (2017)

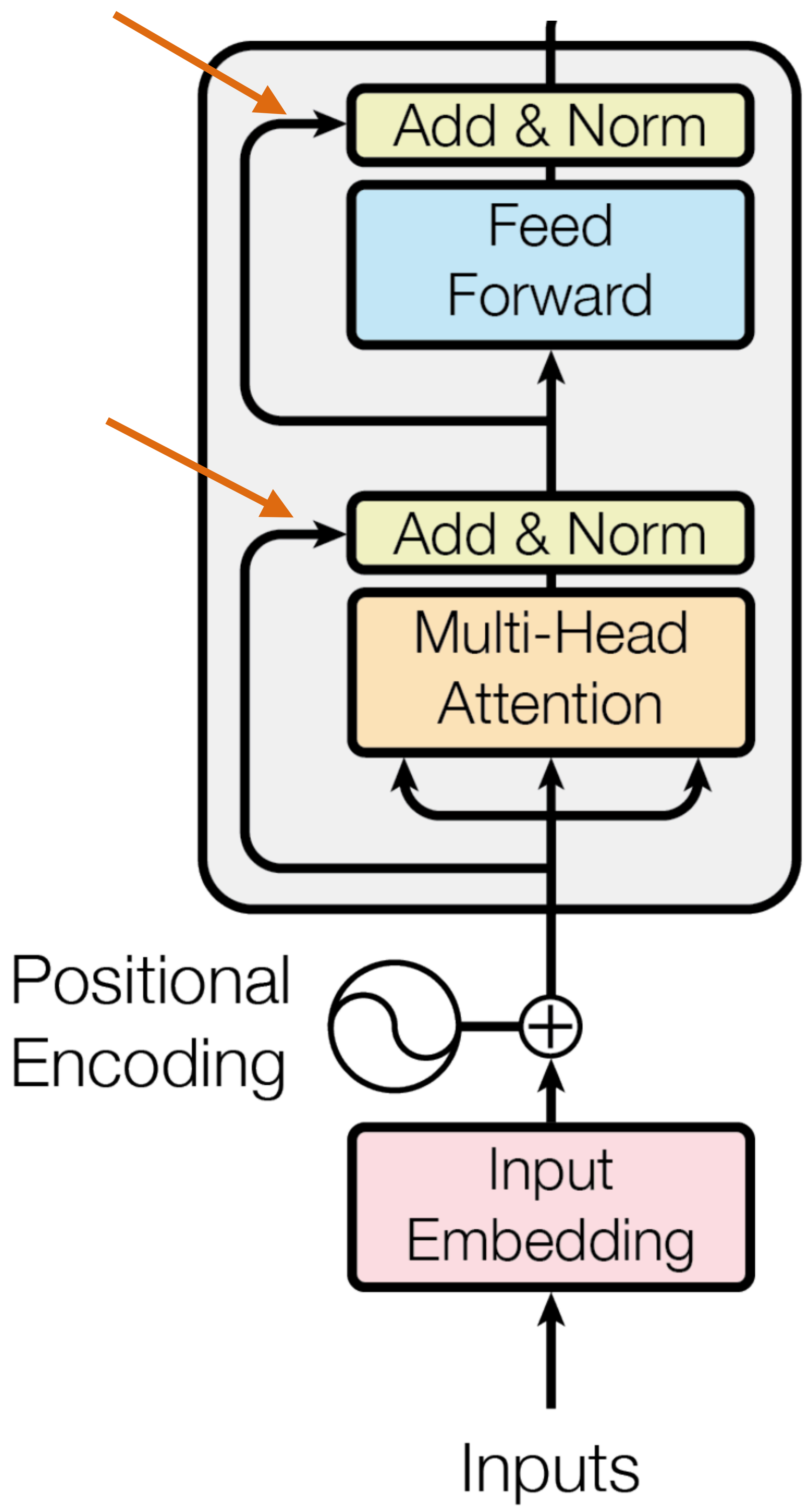
Transformers for MT: Complete Model



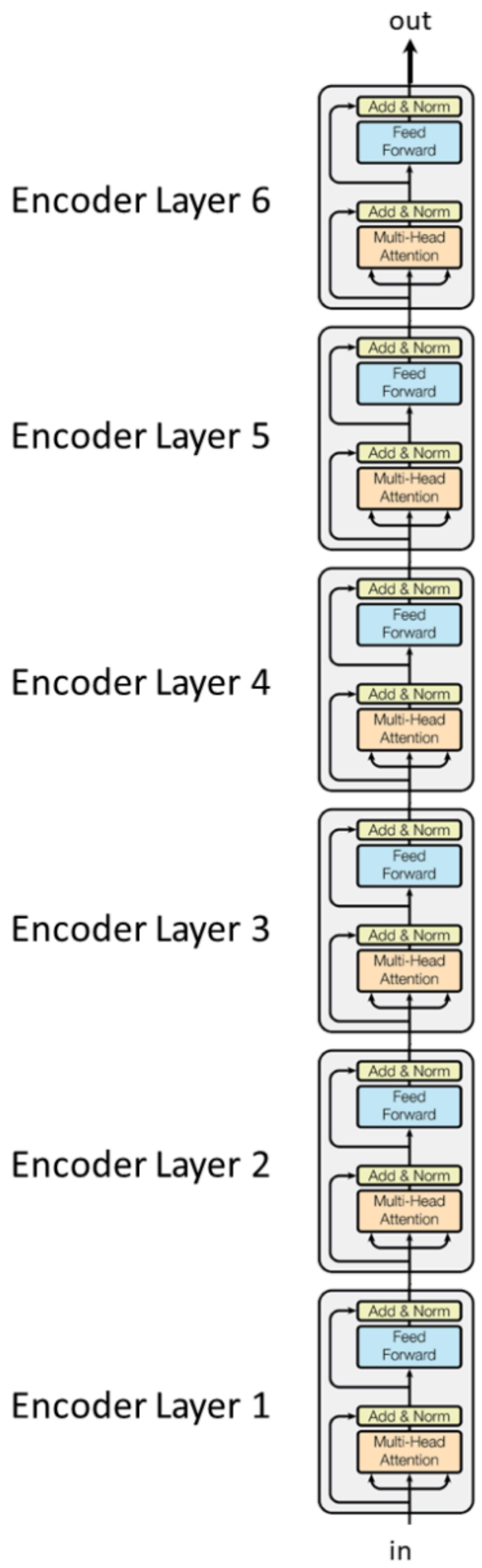
- Encoder and decoder are both transformers
- Decoder alternates attention over the output and attention over the input as well
- Decoder consumes the previous generated tokens but has *no recurrent state*

Vaswani et al. (2017)

Transformers



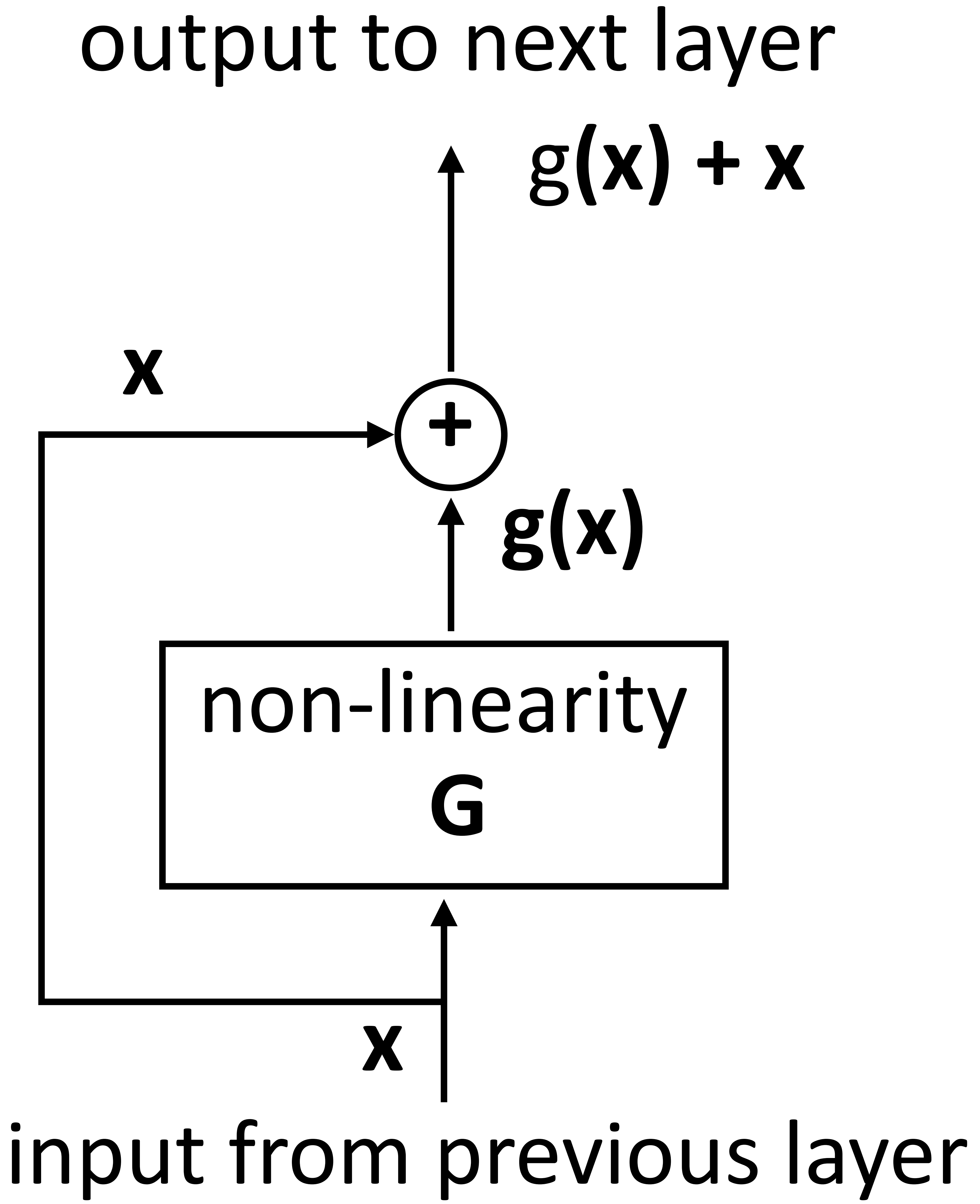
- Alternate multi-head self-attention layers and feedforward layers
- Residual connections let the model "skip" each layer — these are particularly useful for training deep networks



Vaswani et al. (2017)

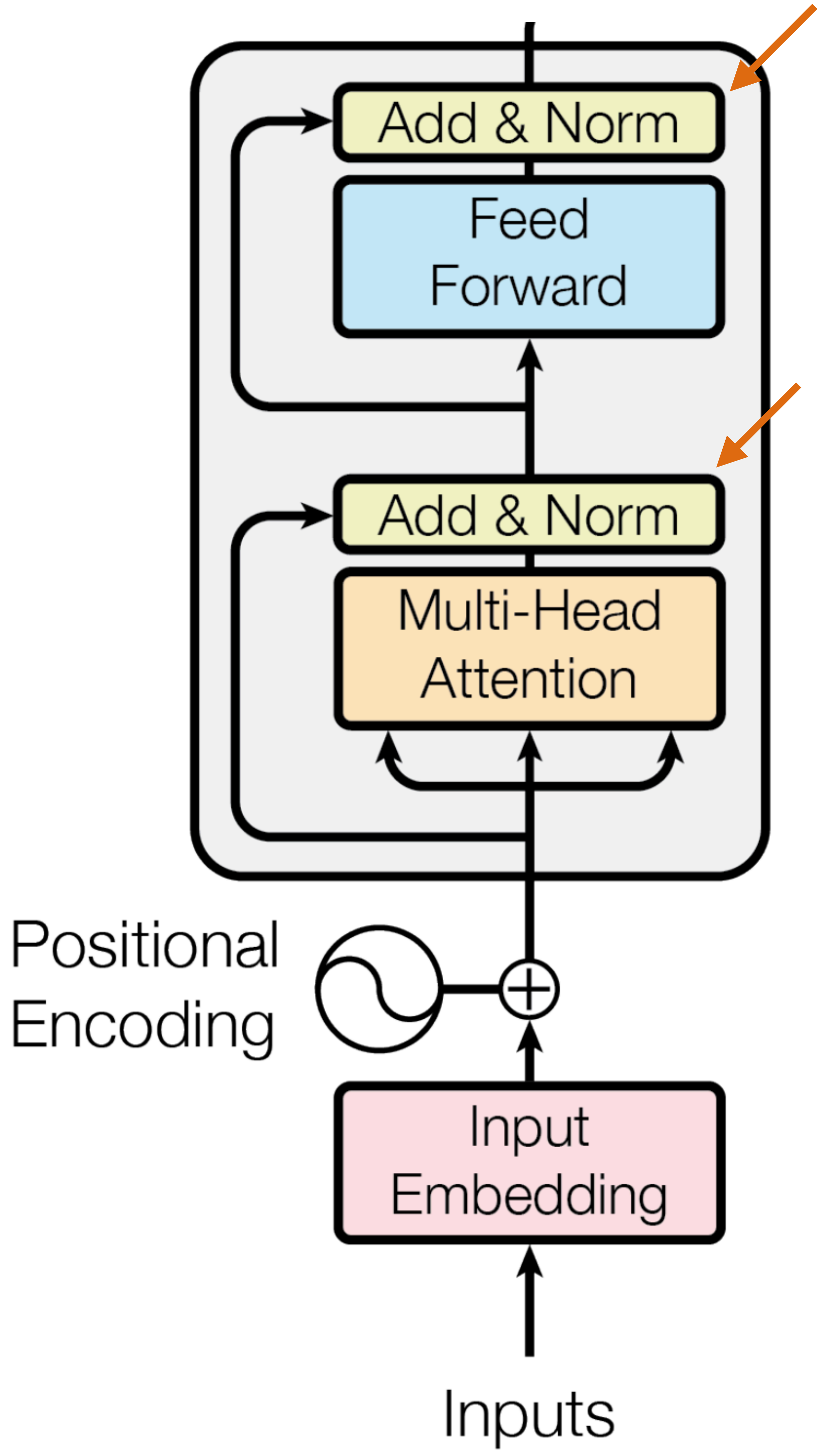
Residual Connections

- allow gradients to flow through a network directly, without passing through non-linear activation functions

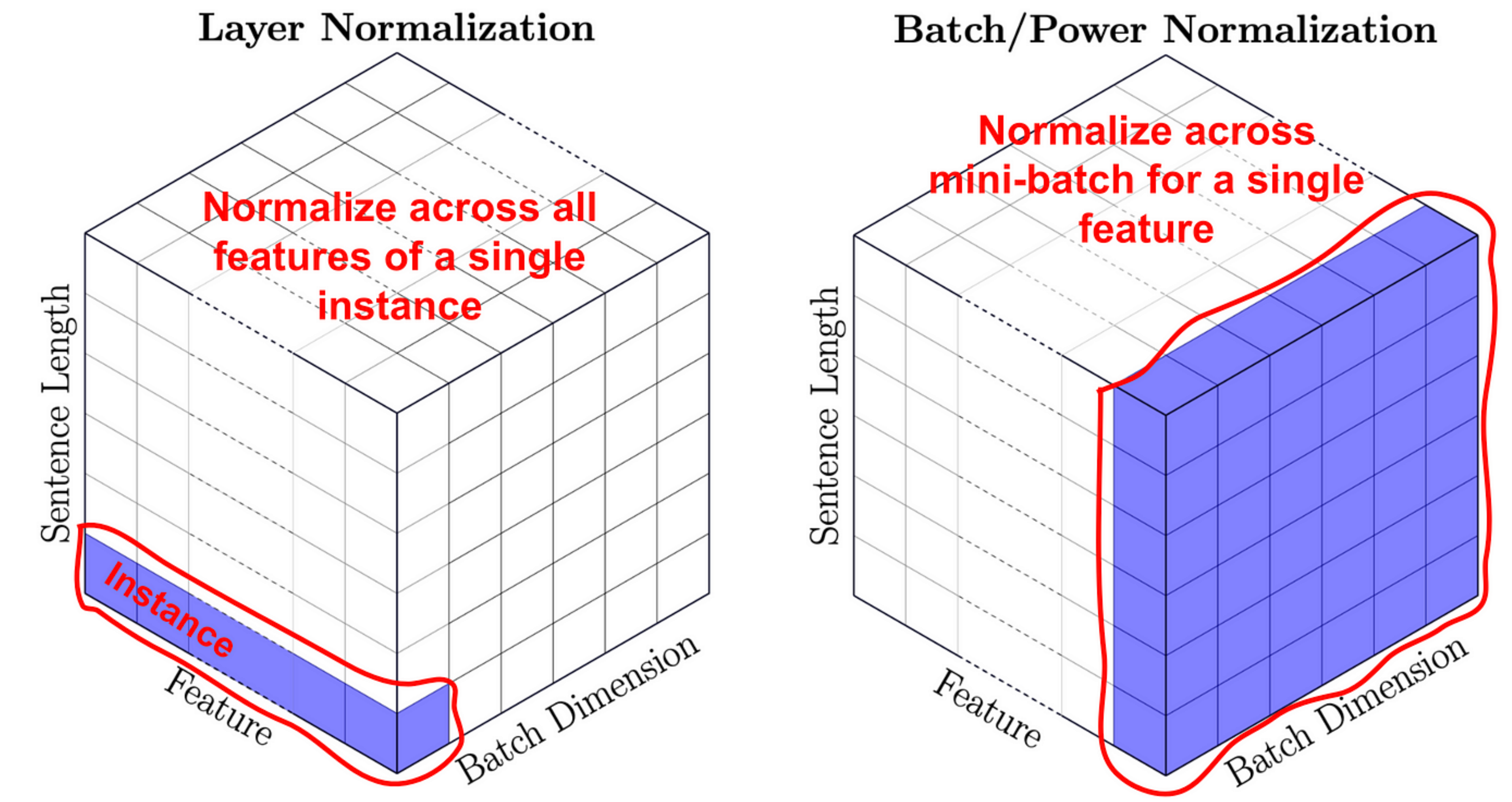


He et al. (2015)

Layer Normalization



- subtract mean, divide by variance



Ba et al. (2016)

Image Credit: Shahwar Alam Naqi

Batch Normalization

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \text{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \text{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

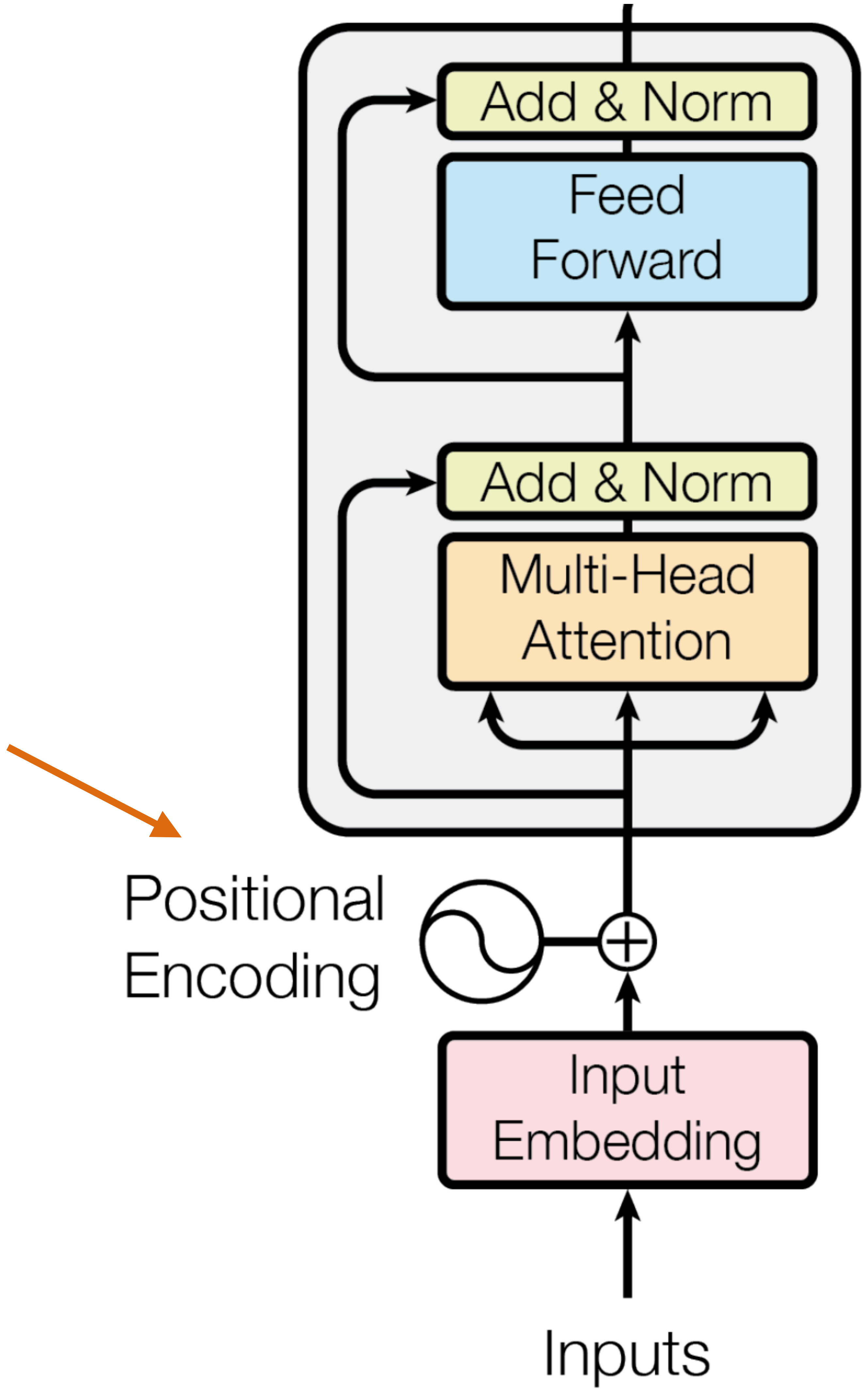
Transformers: Position Sensitivity

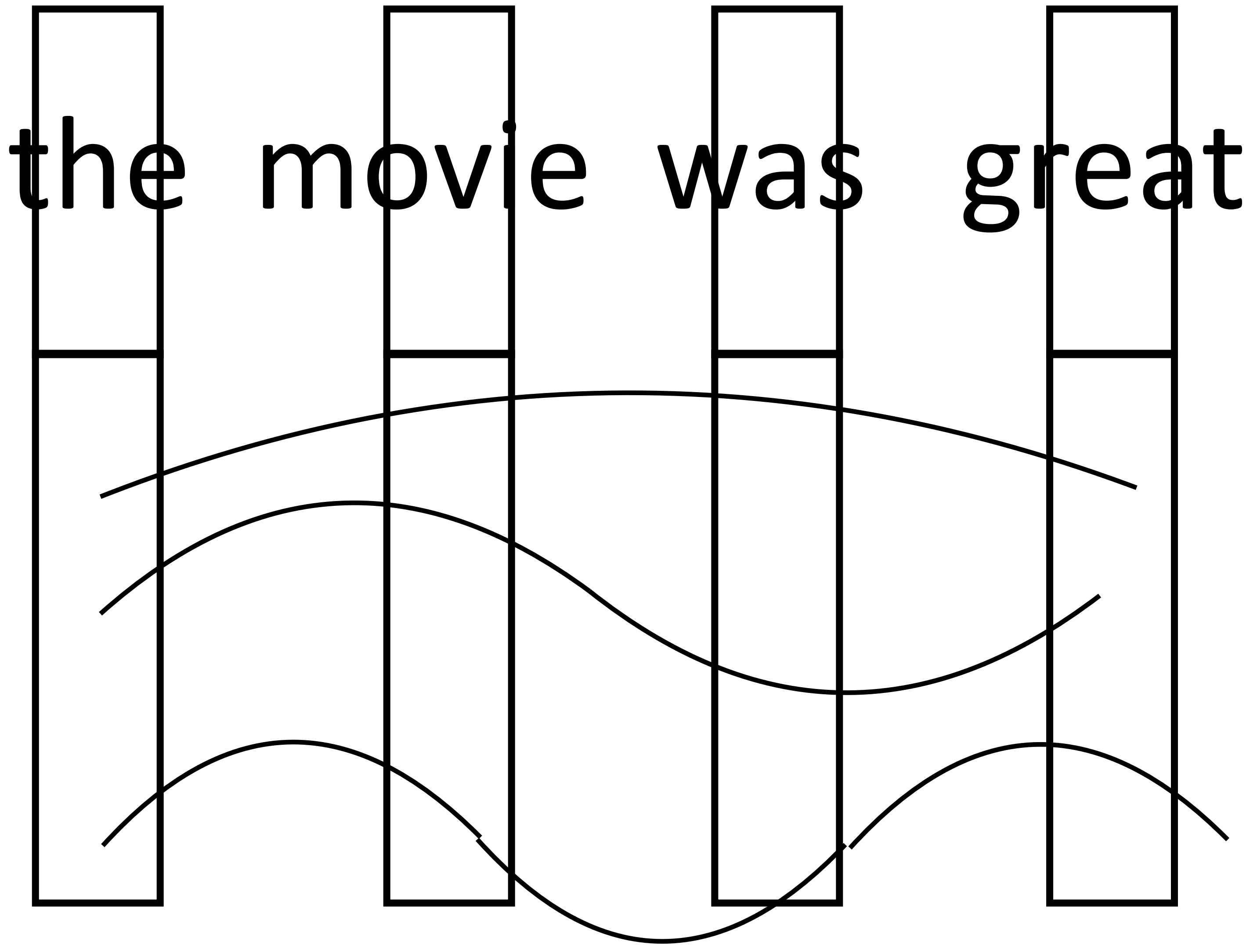
The ballerina is very excited that she will dance in the show.

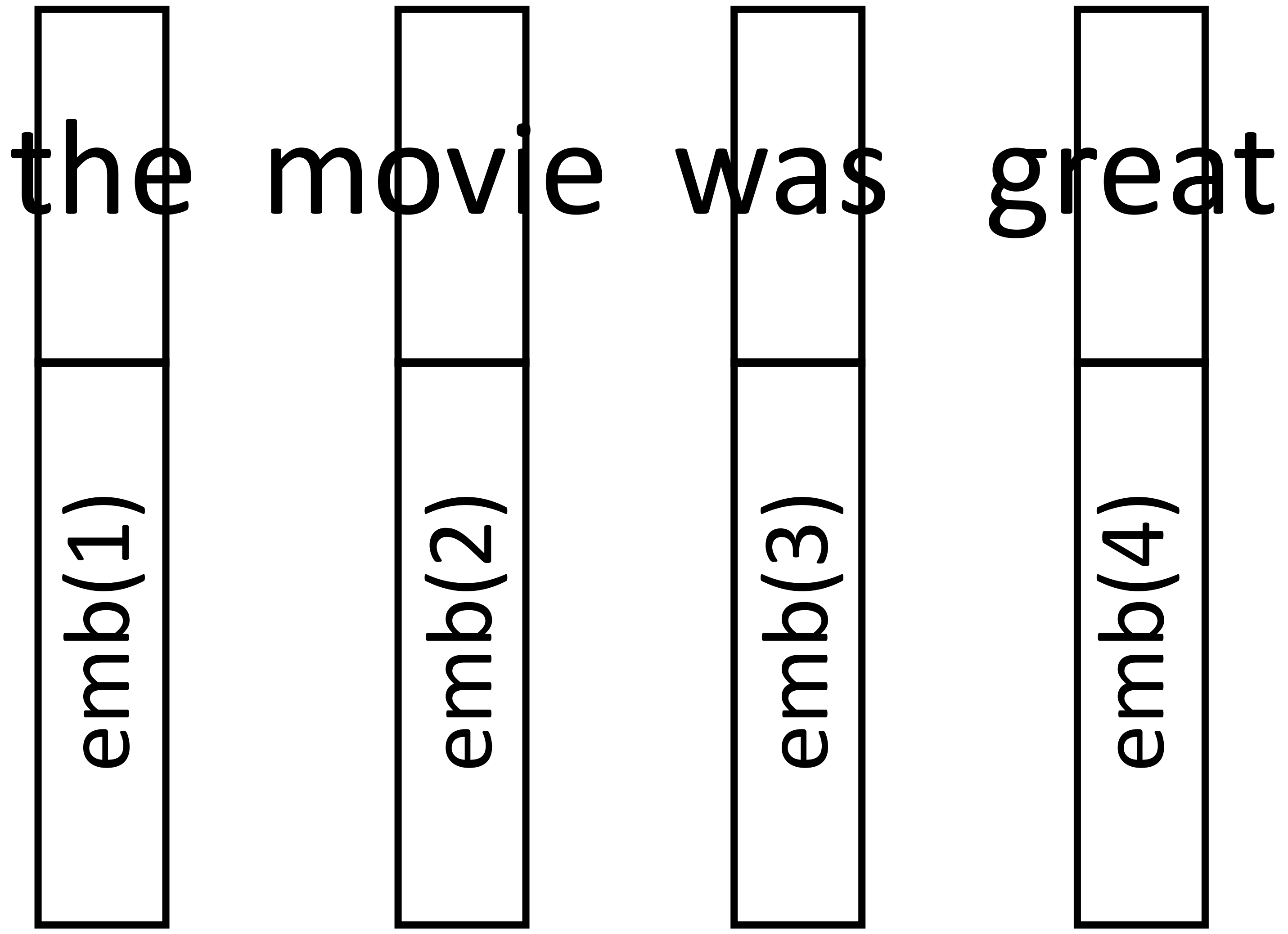
- If this is in a longer context, we want words to attend *locally*
- But transformers have *no notion of position* by default

Vaswani et al. (2017)

Transformers



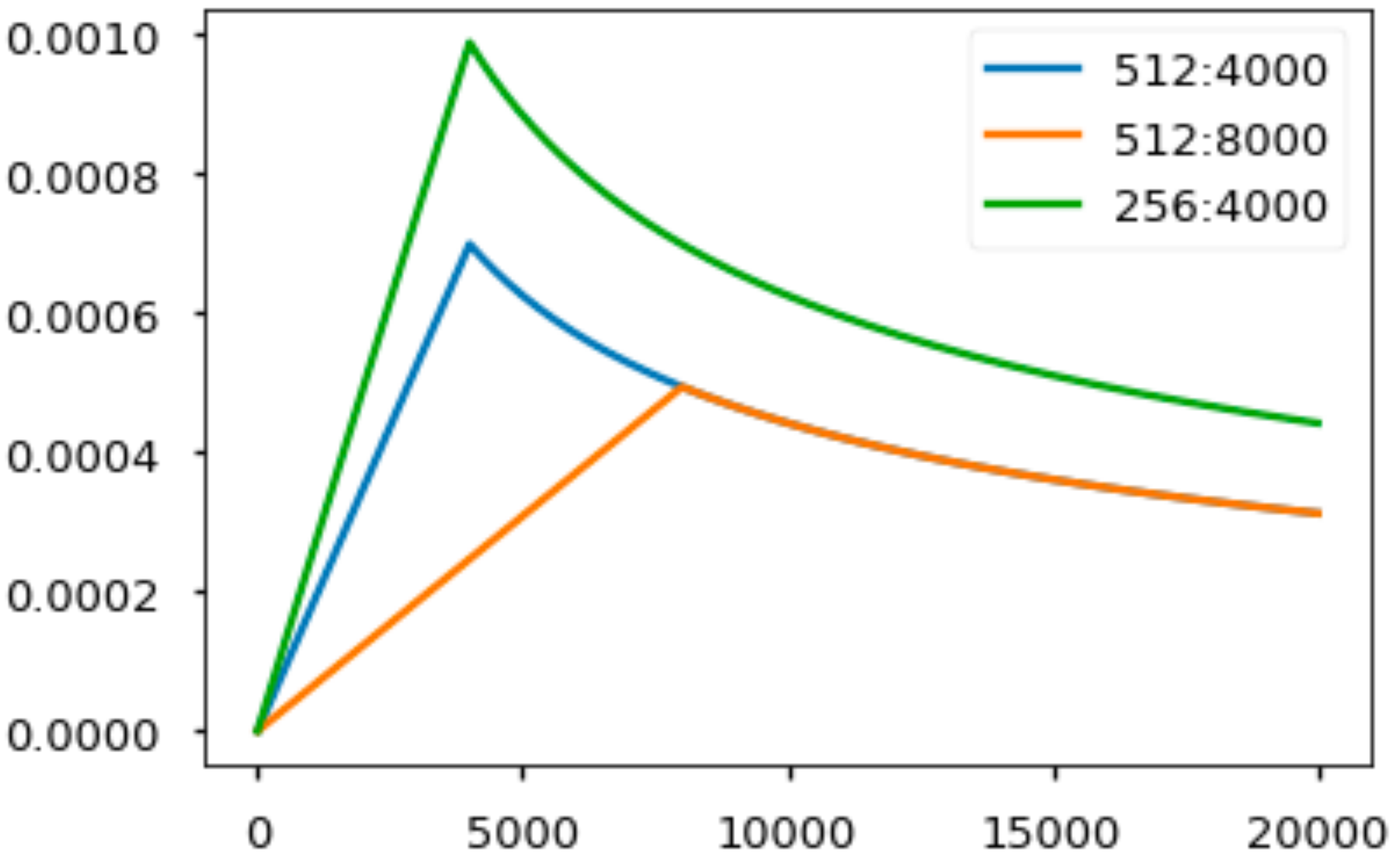




- Augment word embedding with position embeddings, each dim is a sine/cosine wave of a different frequency. Closer points = higher dot products
- Works essentially as well as just encoding position as a one-hot vector

Vaswani et al. (2017)

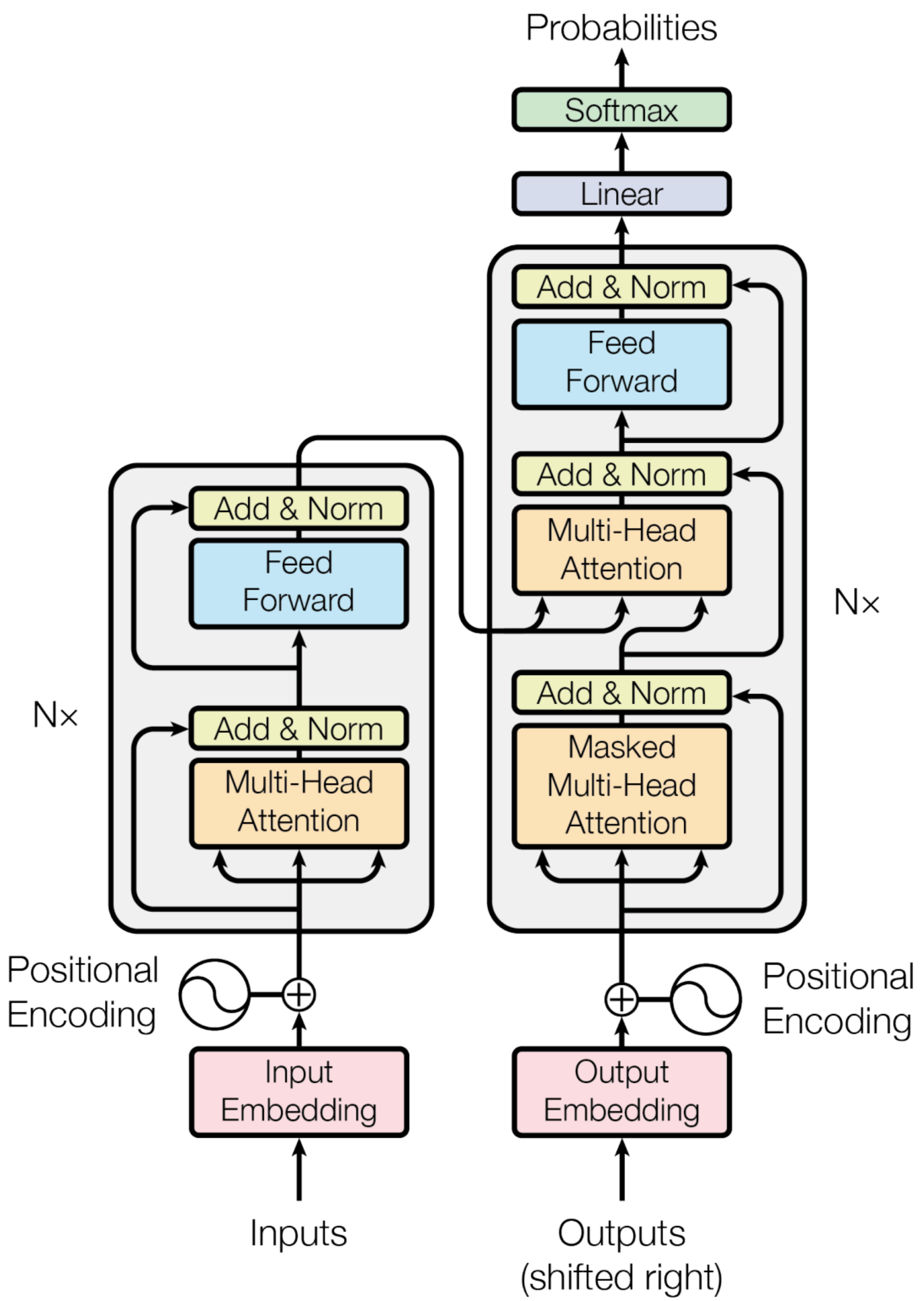
Transformers



- Adam optimizer with varied learning rate over the course of training
- Linearly increase for warmup, then decay proportionally to the inverse square root of the step number
- This part is very important!

Vaswani et al. (2017)

Transformers for MT: Complete Model



- Many other details to get it to work: residual connections, layer normalization, positional encoding, optimizer with learning rate schedule, label smoothing ....

Vaswani et al. (2017)

Transformers

Model	BLEU	
	EN-DE	EN-FR
ByteNet [18]	23.75	
Deep-Att + PosUnk [39]		39.2
GNMT + RL [38]	24.6	39.92
ConvS2S [9]	25.16	40.46
MoE [32]	26.03	40.56
Deep-Att + PosUnk Ensemble [39]		40.4
GNMT + RL Ensemble [38]	26.30	41.16
ConvS2S Ensemble [9]	26.36	**41.29**
Transformer (base model)	27.3	38.1
Transformer (big)	**28.4**	**41.8**

- big = 6 layers, 1000 dim for each token, 16 heads, base = 6 layers + other params halved

Vaswani et al. (2017)

Useful Resources

nn.Transformer:

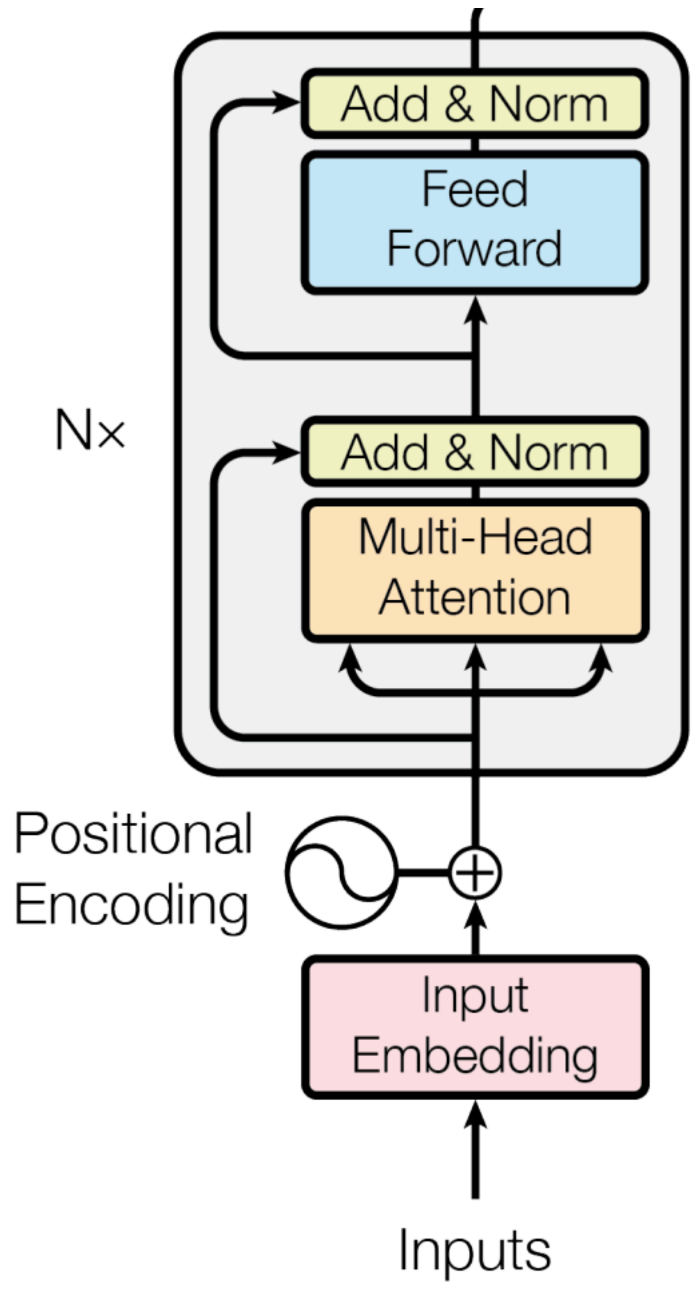
```
>>> transformer_model = nn.Transformer(nhead=16, num_encoder_layers=12)
>>> src = torch.rand((10, 32, 512))
>>> tgt = torch.rand((20, 32, 512))
>>> out = transformer_model(src, tgt)
```

nn.TransformerEncoder:

```
>>> encoder_layer = nn.TransformerEncoderLayer(d_model=512, nhead=8)
>>> transformer_encoder = nn.TransformerEncoder(encoder_layer, num_layers=6)
>>> src = torch.rand(10, 32, 512)
>>> out = transformer_encoder(src)
```

Other Transformer Variations

- Multilayer transformer networks consist of interleaved self-attention and feedforward sublayers.
- Could ordering the sublayers in a different pattern lead to better performance?



(a) Interleaved Transformer

(b) Sandwich Transformer

Figure 1: A transformer model (a) is composed of interleaved self-attention (green) and feedforward (purple) sublayers. Our sandwich transformer (b), a reordering of the transformer sublayers, performs better on language modeling. Input flows from left to right.

Press et al. (2020)

Other Transformer Variations

- Mixture of Expert (MoE) Transformer, e.g., used in massively multilingual MT

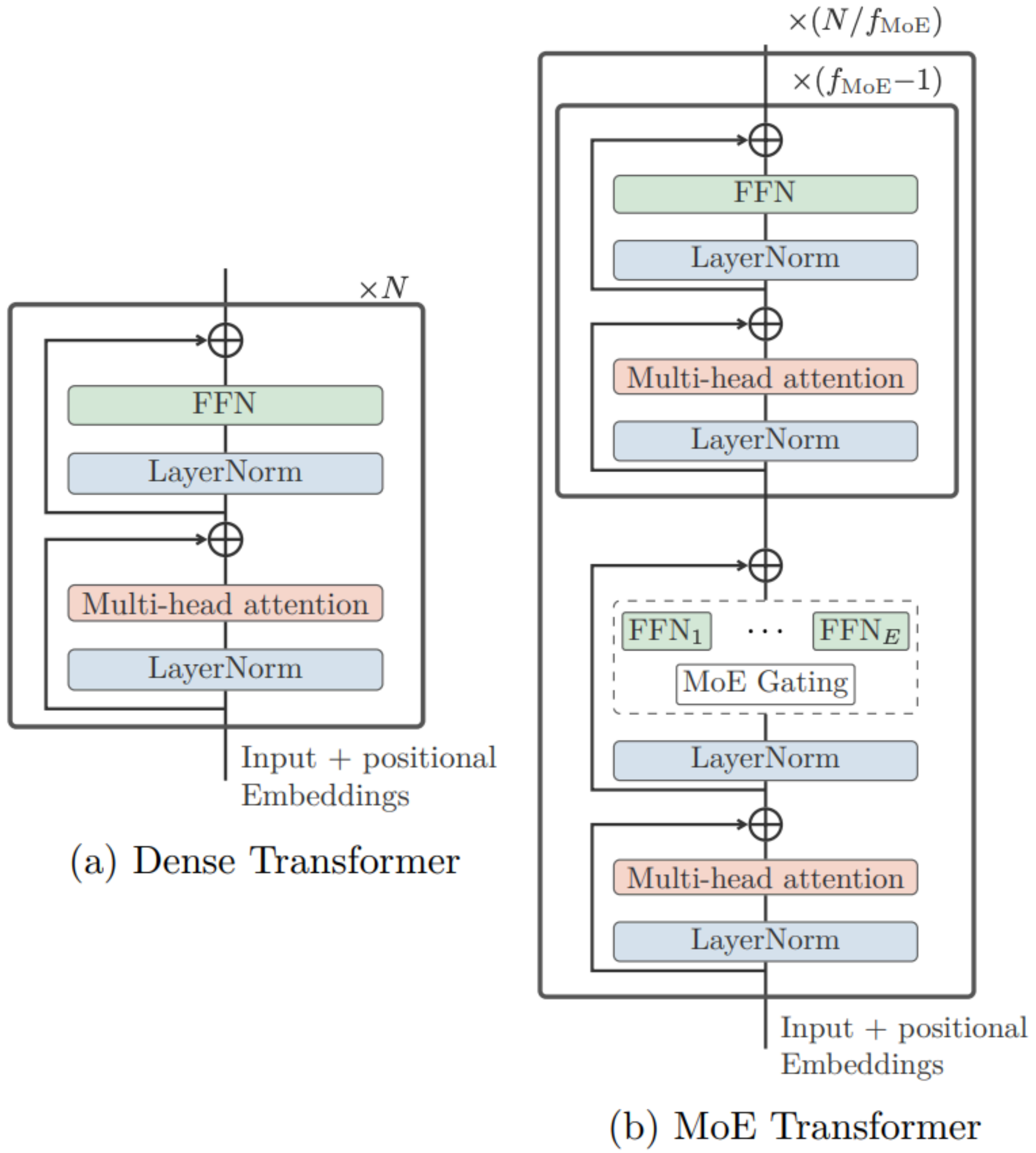
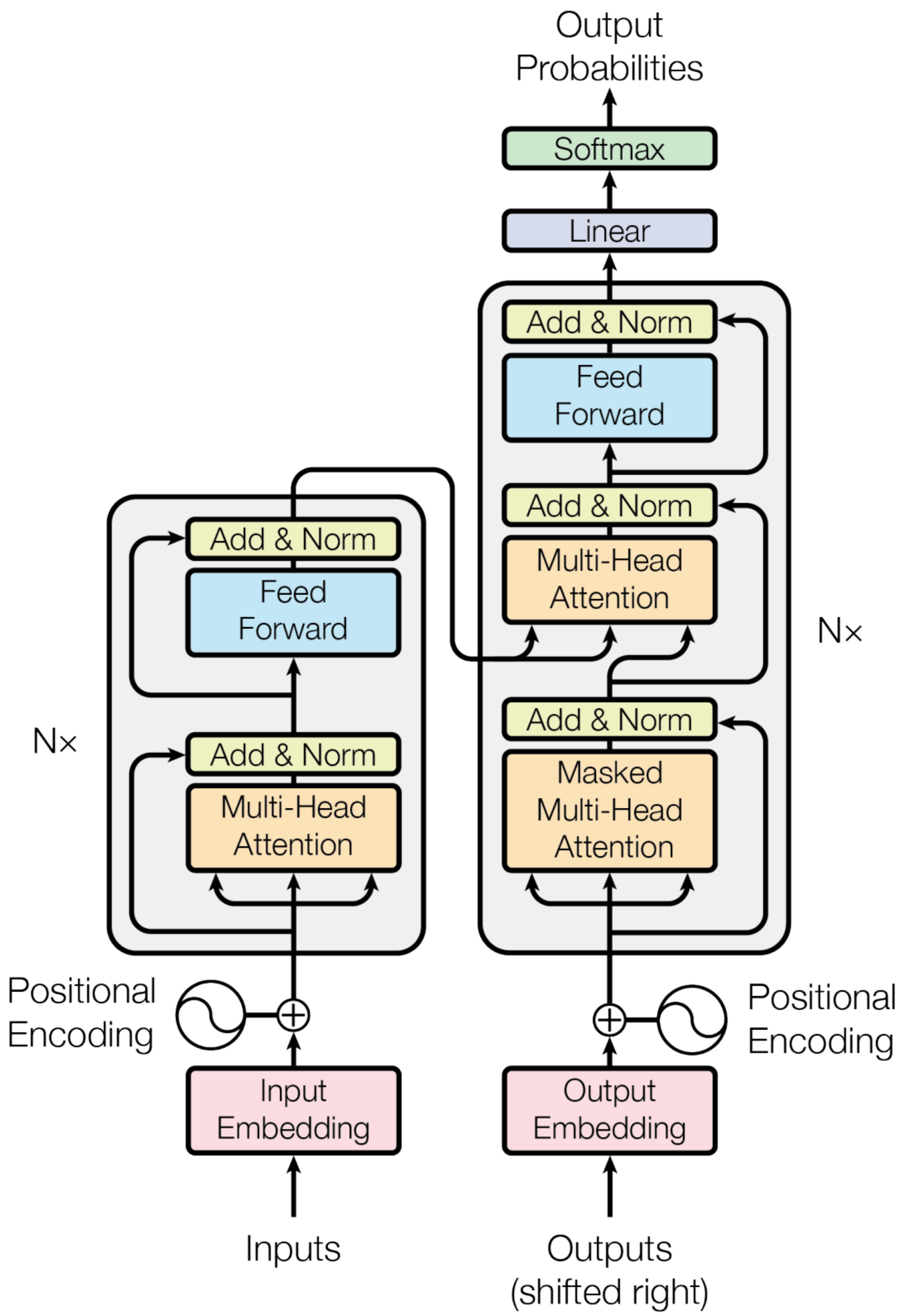


Figure 16: **Illustration of a Transformer encoder with MoE layers** inserted at a 1:f_{MoE} frequency. Each MoE layer has E experts and a gating network responsible for dispatching tokens.

Eigen el al. (2013), Shazeer et al. (2017), NLLB (2022)

Summary: Transformer Uses

- Supervised: transformer can replace LSTM as encoder, decoder, or both; such as in machine translation and natural language generation tasks.



- Encoder and decoder are both transformers
- Decoder consumes the previous generated token (and attends to input), but has *no recurrent state*

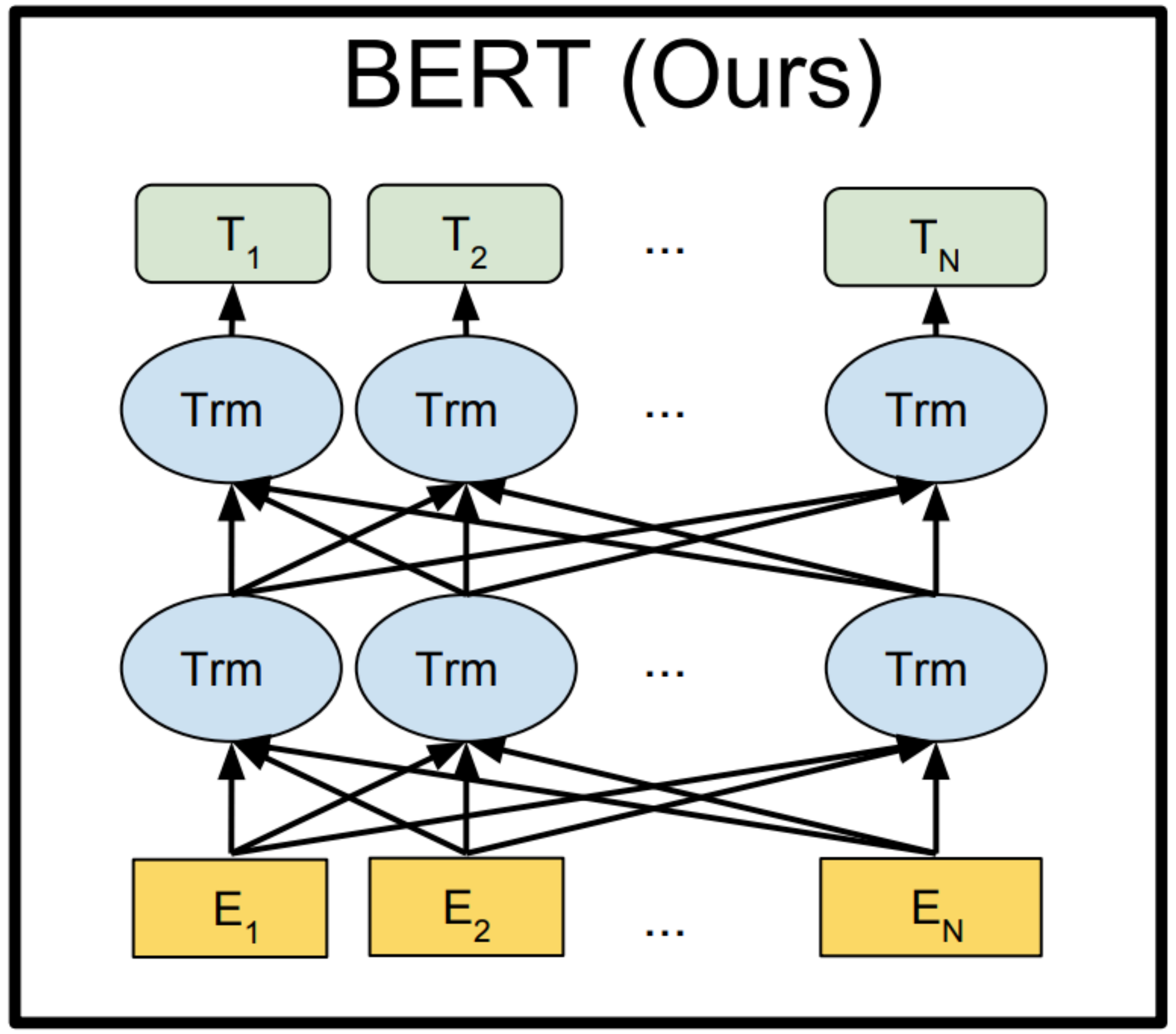
Vaswani et al. (2017)

Summary: Transformer Uses

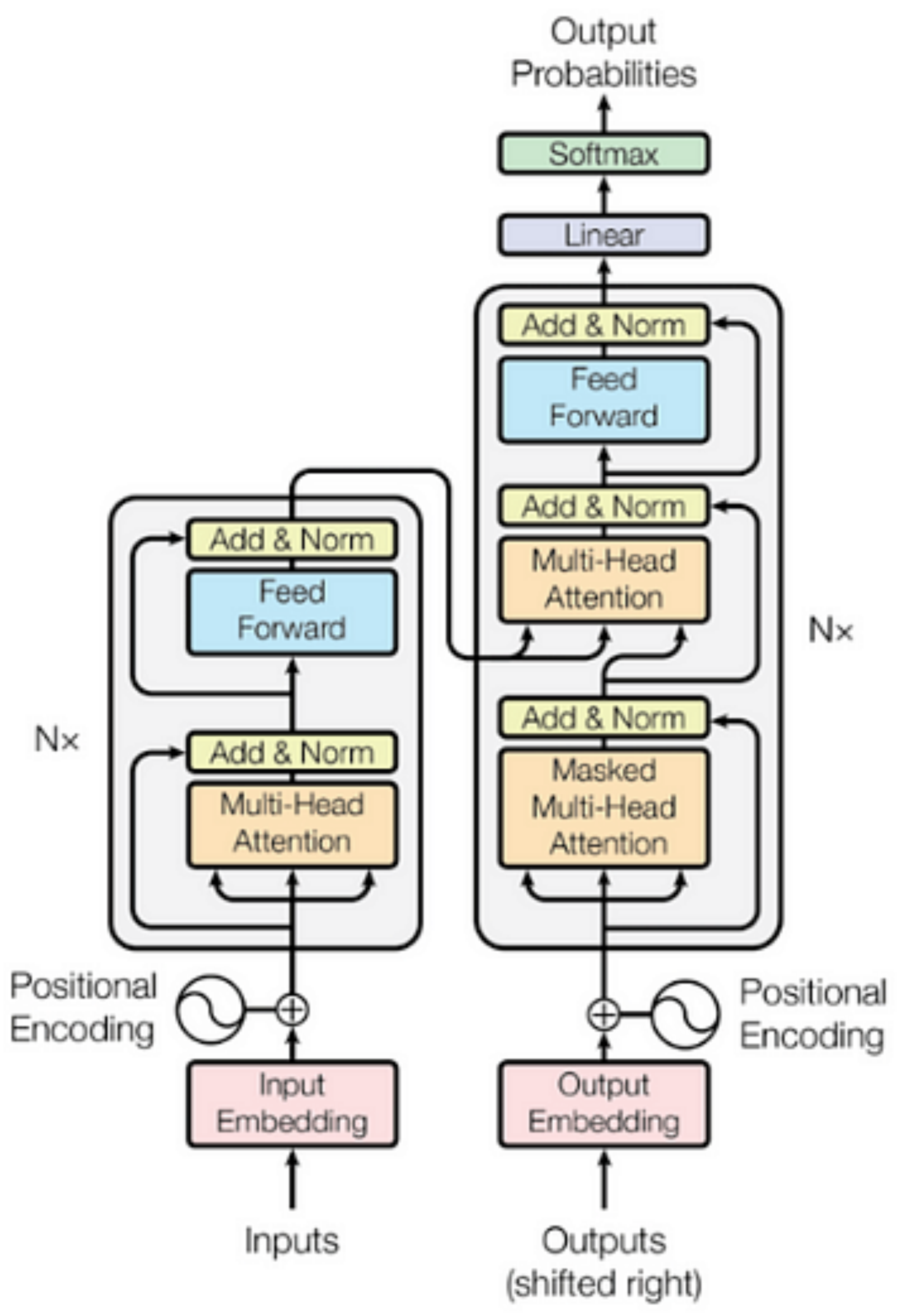
- Unsupervised: transformers work better than LSTM for unsupervised pre-training of embeddings — predict word given context words

- BERT (Bidirectional Encoder Representations from Transformers): pretraining transformer language models similar to ELMo (based on LSTM)

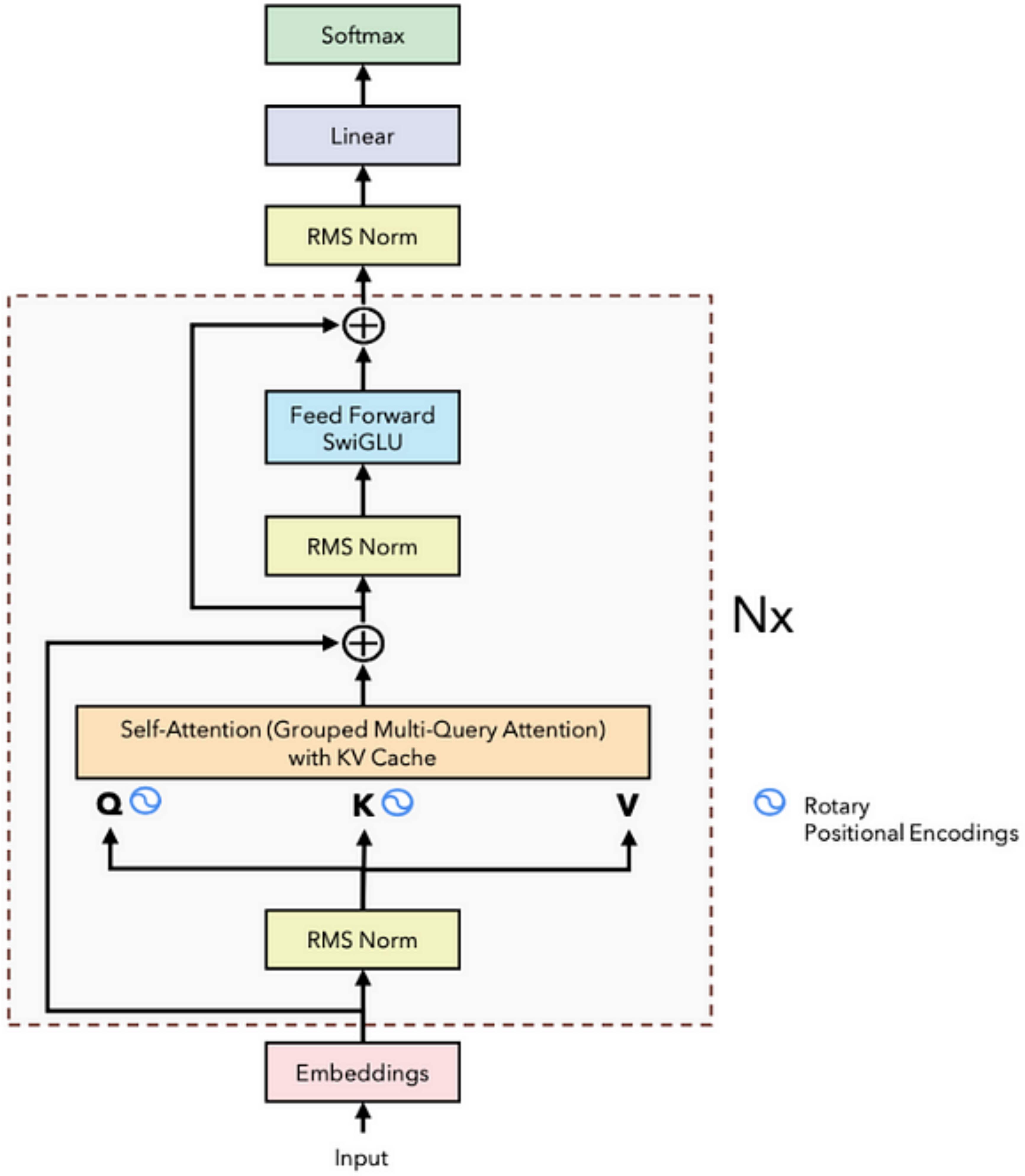
- Stronger than similar methods, SOTA on ~11 tasks (including NER — 92.8 F1)



Transformer as in LLaMA-3



Transformer



LLaMA