Attention

And Transformers

Aaron Mueller
CAS CS 505: Introduction to Natural Language Processing
Spring 2026
Boston University

Neural Networks Take Over NLP

- Before 2012, most NLP systems did *not* use neural networks
  - Statistical methods
  - SVMs

- **2012:** AlexNet **[Krizhevsky et al.]** (in computer vision)

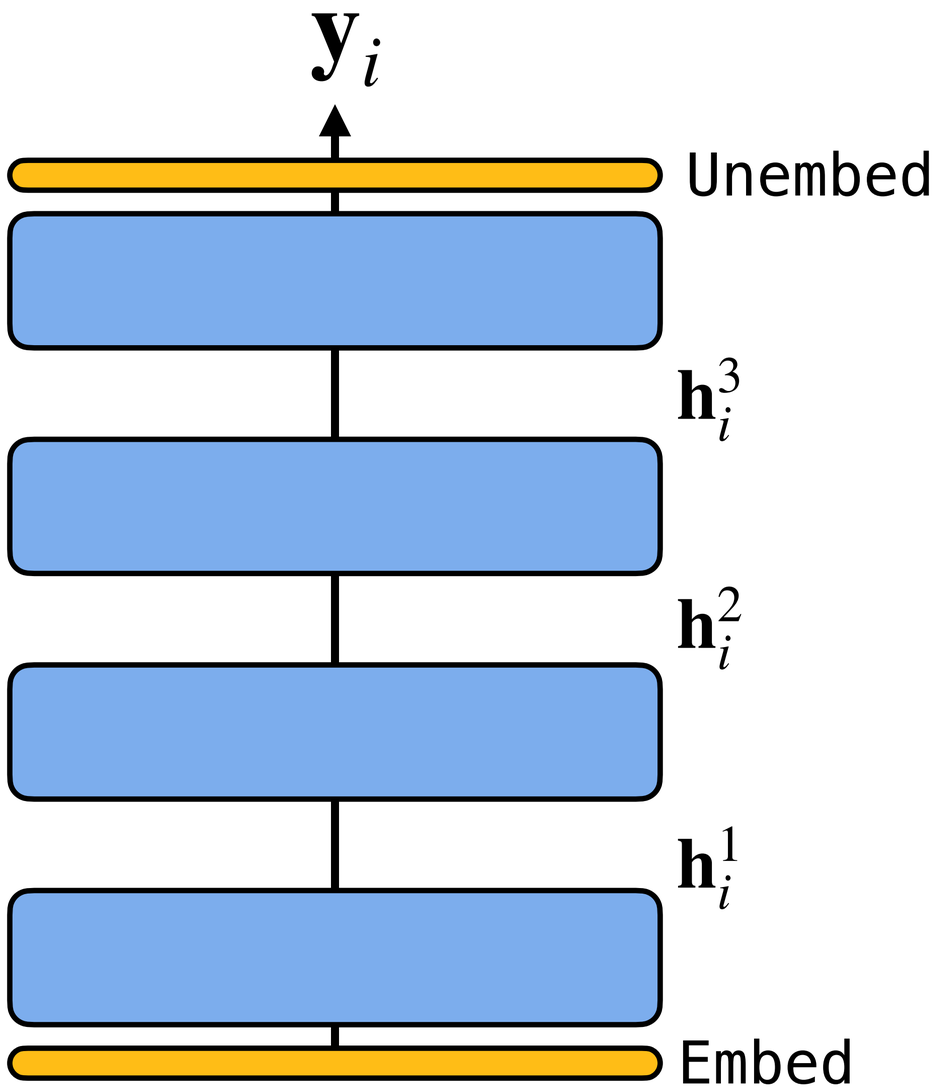
- **2014:** The first neural machine translation system outperforming statistical methods. *This is what put recurrent neural networks on the map for NLP.*

$\mathbf{x} =$ Translate this to French: The cat sat on the mat. Le chat s’est assis sur le tapis.

$\mathbf{h}^{\ell}$: Hidden representation at output of layer ℓ

x_i: The ith token of input $\mathbf{x}$

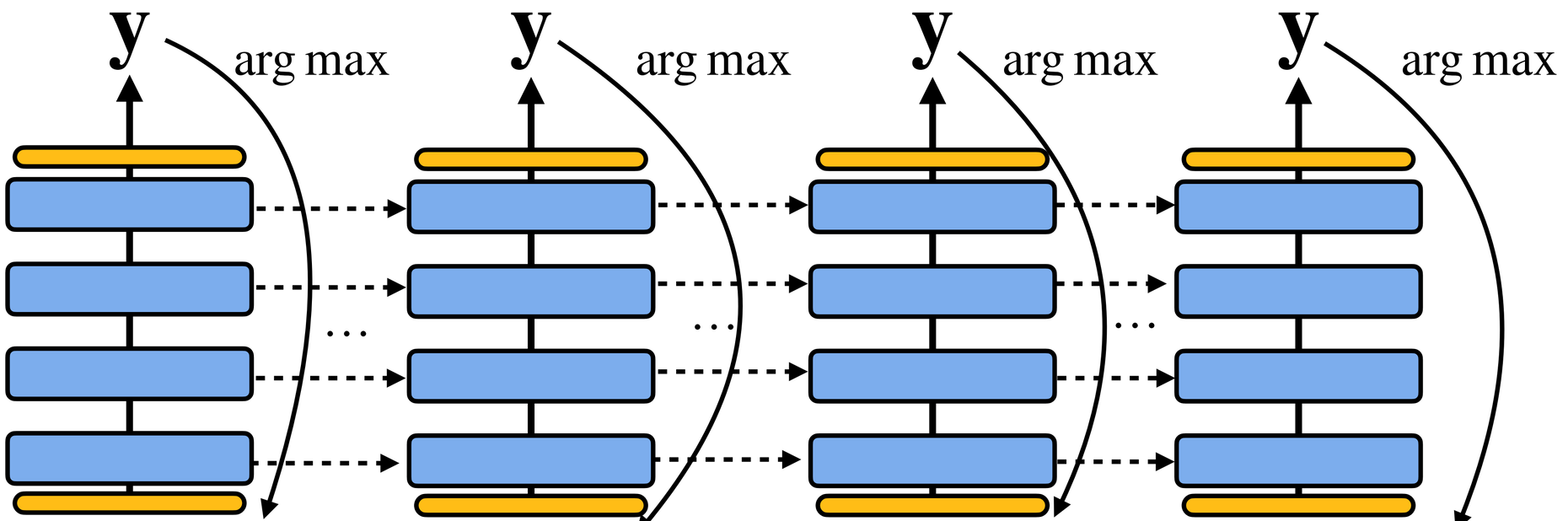
$\mathbf{y}$: The output logits vector



$\mathbf{x} =$ Translate (x_1) late (x_2) this (x_3) to … French: The cat sat on the mat. Le chat s'est assis sur le tapis. (x_n)

RNN Sequence Models

Autoregressive generation:



$\mathbf{x} =$ Translate this to French: The cat sat on the mat. Le chat _____

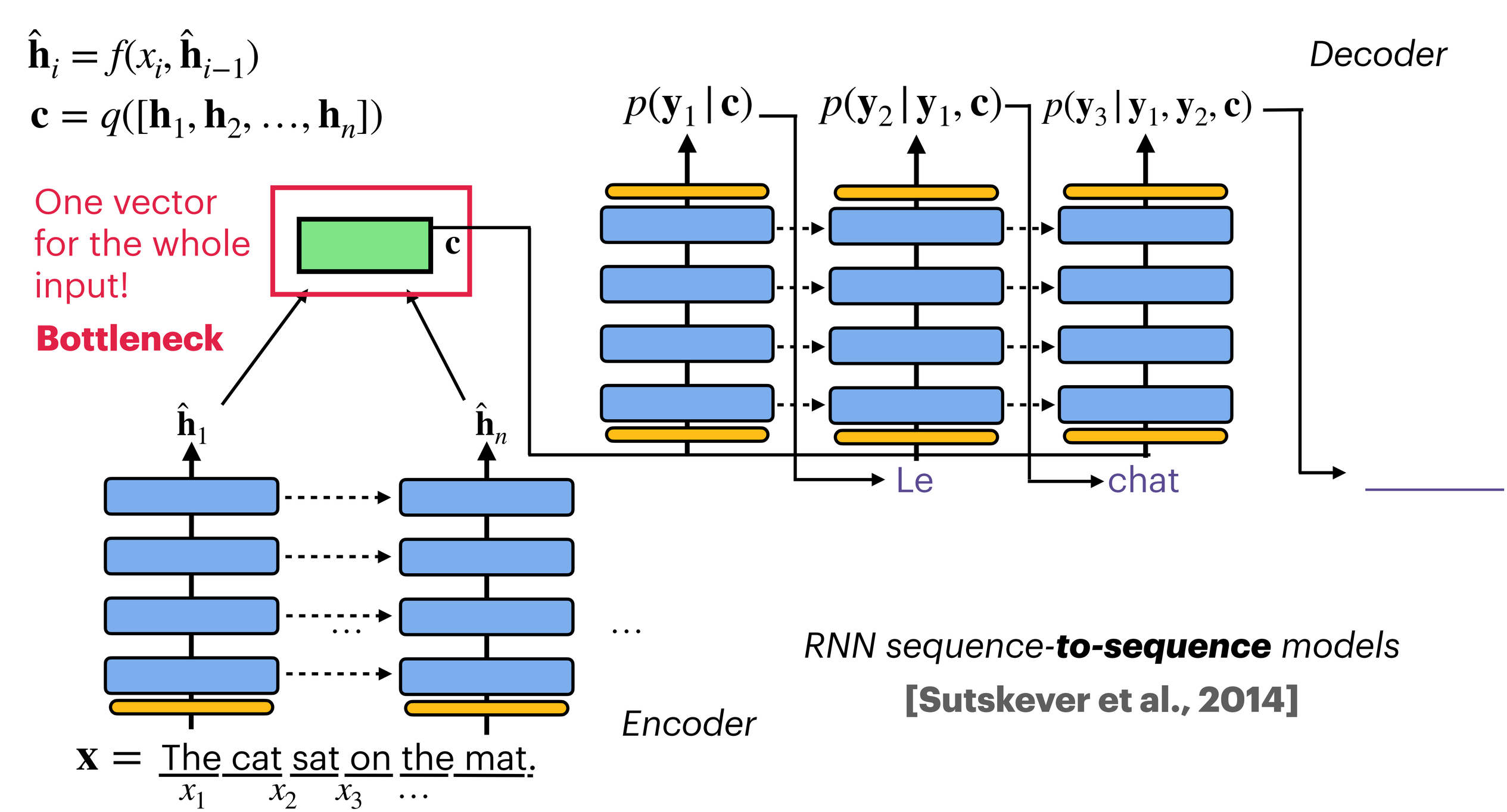
x_1 x_2 x_3 …

$$\hat{\mathbf{h}}_i = f(x_i, \hat{\mathbf{h}}_{i-1})$$

$$\mathbf{c} = q([\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_n])$$

One vector for the whole input!

Bottleneck



*RNN sequence-**to-sequence** models*

[Sutskever et al., 2014]

Ilya Sutskever et al. (2014). "Sequence to sequence learning with neural networks." *NeurIPS*.

Reprise: *How much can you cram into a vector?*

- If $\mathbf{x}$ is a word, $\mathbf{c}$ will probably work fine
- If $\mathbf{x}$ is a sentence, you'll probably lose some information
- If $\mathbf{x}$ is a document, good luck

(Editor's note: Actually, we do this—and often! But if you can avoid it, you probably should.)

"You can't cram the meaning of a whole %&!$# sentence into a single $&!# vector!"*

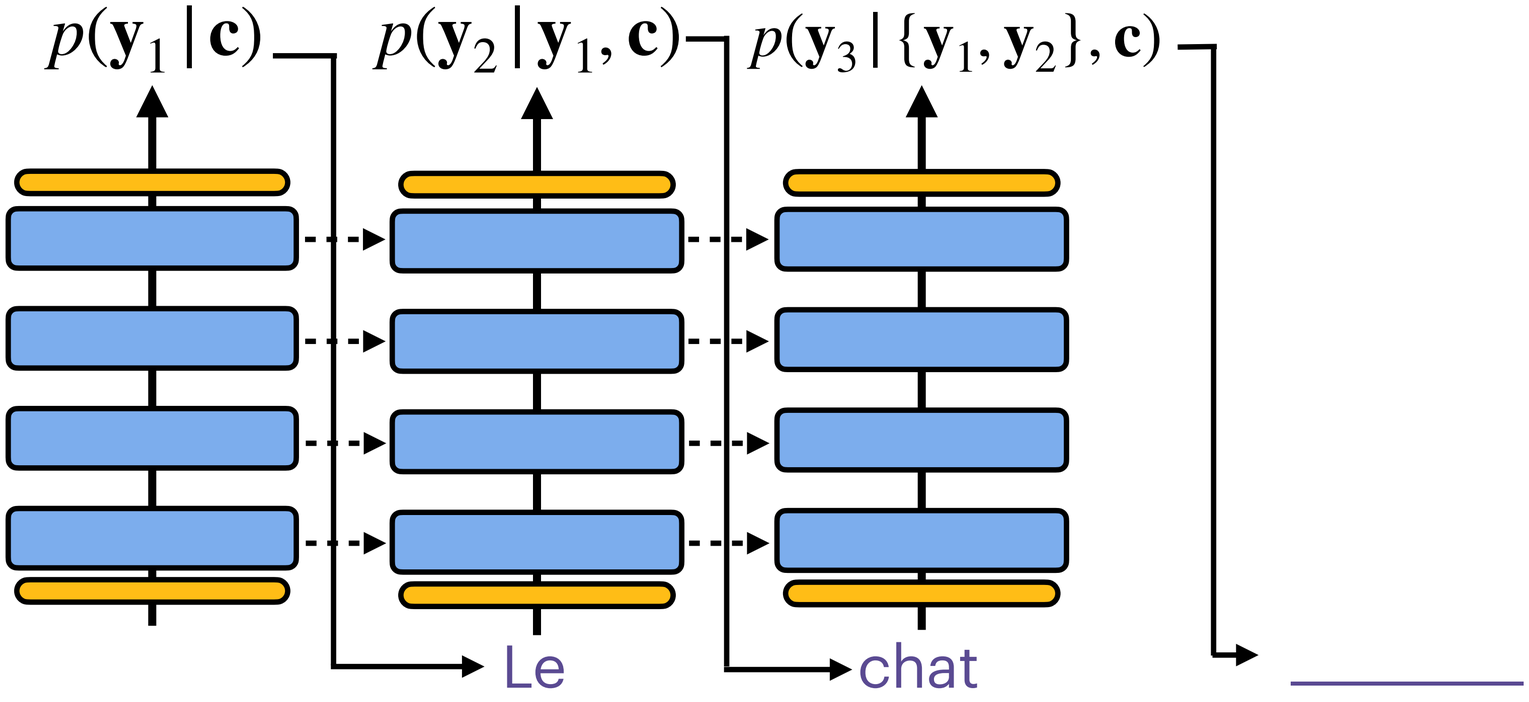
—Raymond Mooney, 2014

The Vanishing Gradient Problem

$$\frac{\partial \mathbf{h}_i}{\partial W_{\mathbf{hh}}} = \sum_{i=1}^{n} (W_{hh}^{\top})^{n-i} \mathbf{h}_i$$

$$\cdots$$

$$\frac{\partial \mathbf{h}_i}{\partial W_{\mathbf{xh}}} = \sum_{i=1}^{n} (W_{hh}^{\top})^{n-i} \mathbf{x}_i$$



Attention

- Idea: what if we had a separate context vector $\mathbf{c}_i$ for each decoding step?

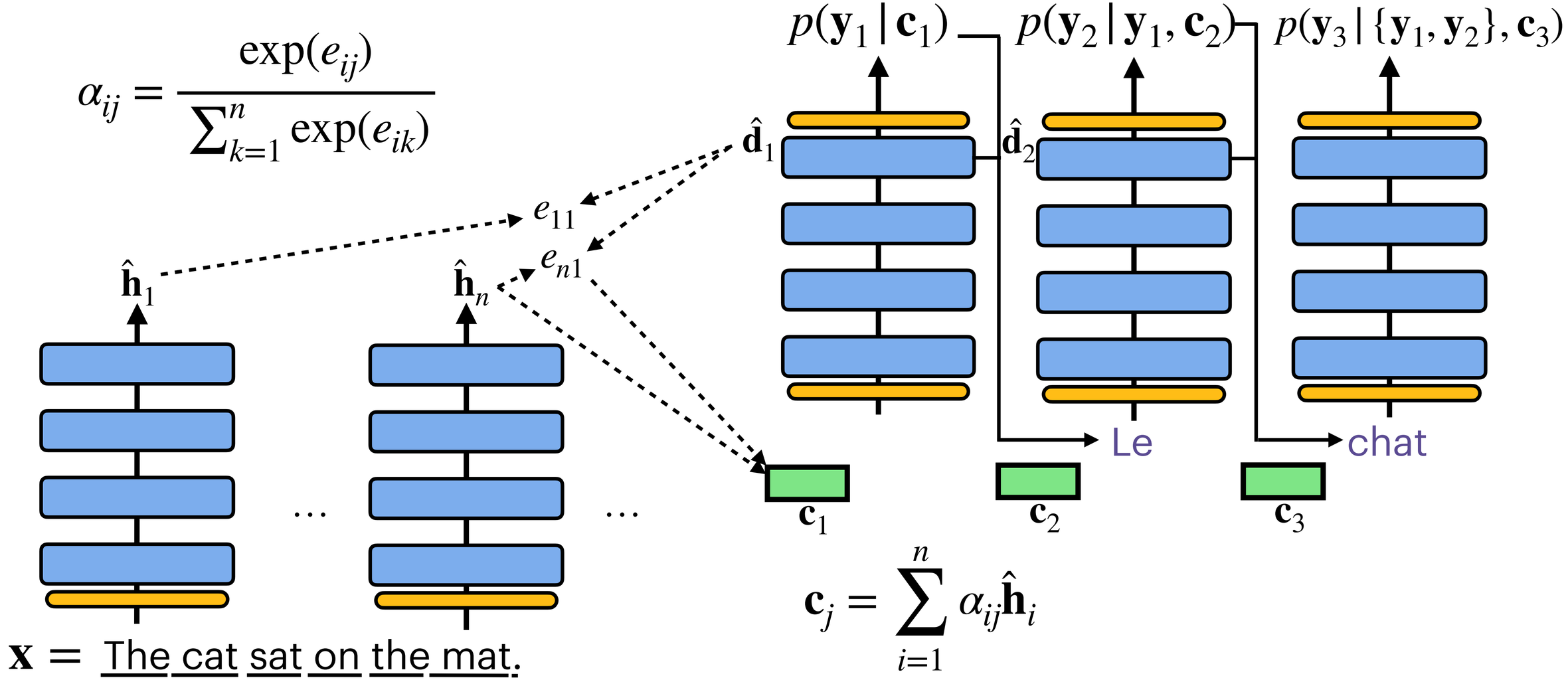
- At each decoding timestep, the model can learn by itself which parts of the context to pay attention to

Attention

Bahdanau et al., 2015

$$e_{ij} = a(\hat{\mathbf{h}}_i, \hat{\mathbf{d}}_j)$$

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{n} \exp(e_{ik})}$$

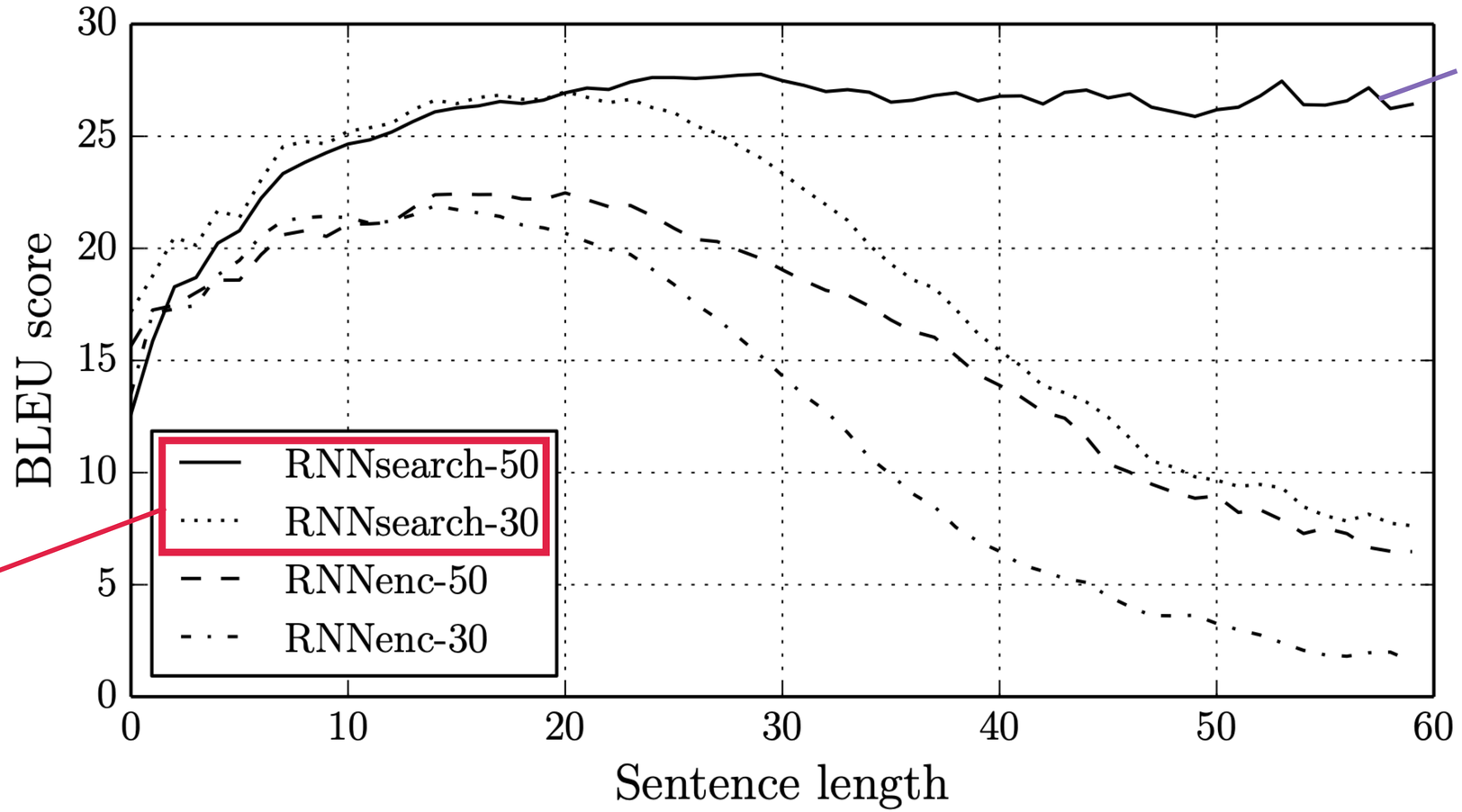


$$\mathbf{c}_j = \sum_{i=1}^{n} \alpha_{ij} \hat{\mathbf{h}}_i$$

Dzimitry Bahdanau et al. (2015). "Neural machine translation by jointly learning to align and translate." *ICLR*.

Advantages of Attention

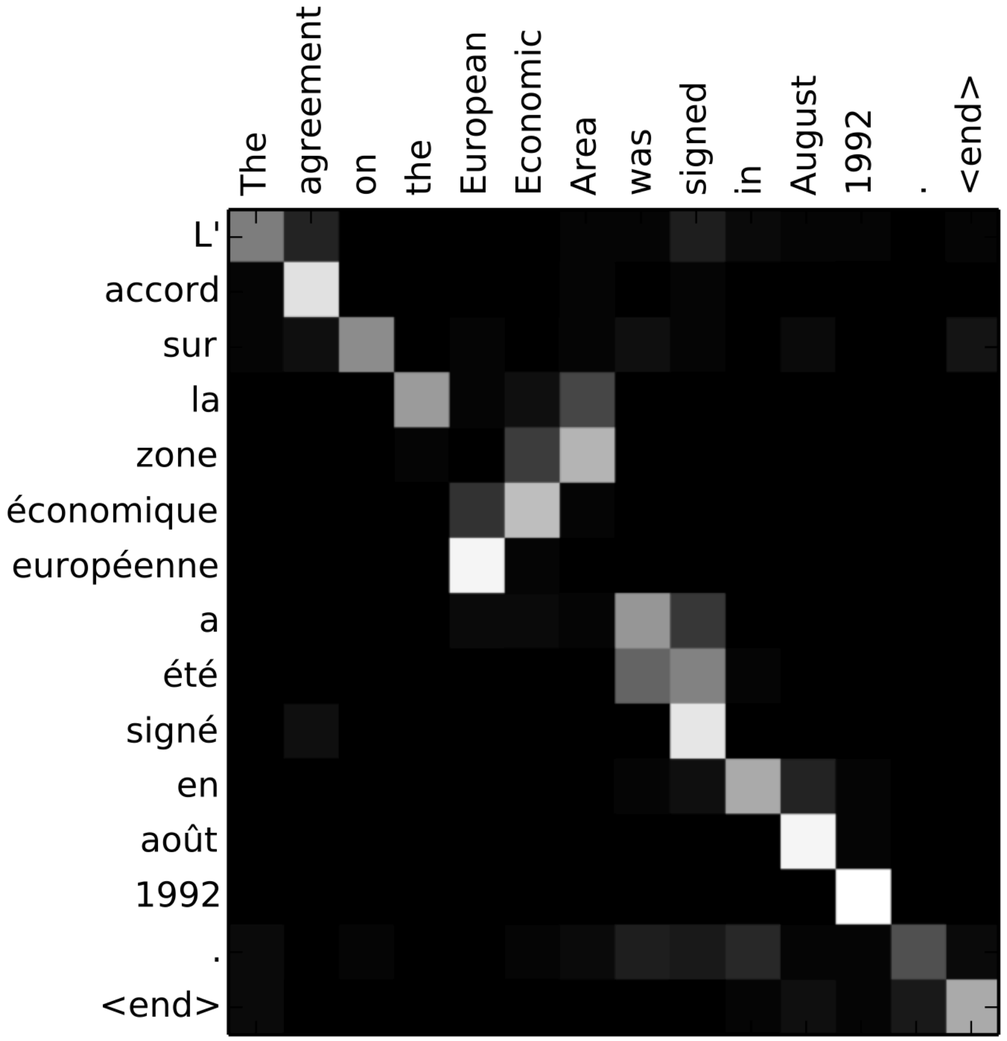
- Attention solves the context bottleneck problem
  - Model can look at any part of the context without losing anything
- Attention helps address vanishing gradients
  - Provides a gradient shortcut to distant previous timesteps
- Attention can sometimes make it easier to understand a model's decisions

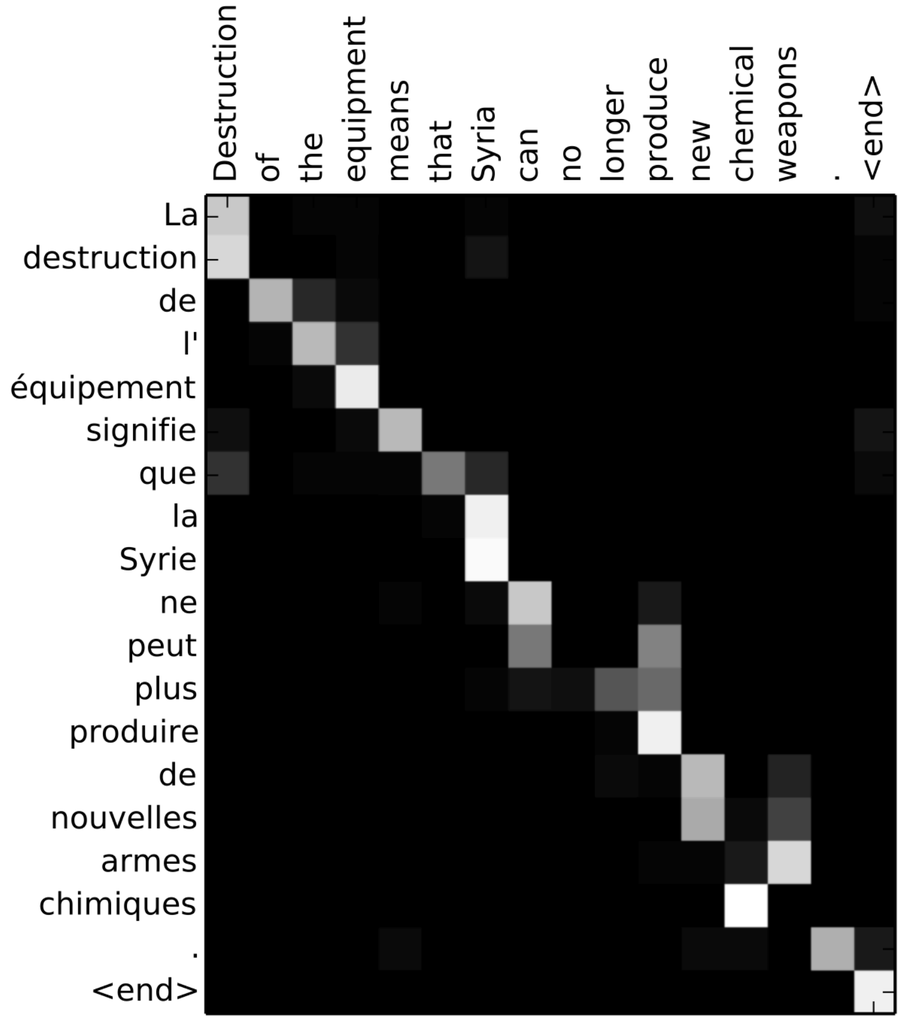


Length generalization!

with attention

α_{ij} attends to meaningful signals!





Summary of Attention (pre-Transformer)

- Adding attention to encoder-decoder models significantly improves their long-context performance
- This achieved state-of-the-art performance when it was released, and put neural networks on the map for most machine translation researchers (and NLP researchers)
- The attention mechanism seems powerful—how important is the rest of the encoder-decoder to achieving high performance?

Attention Is All You Need

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones,
Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin

NeurIPS 2017

Motivation

- Recurrent models are inherently sequential:
  - Relating information from x_i to x_j requires $O(j - i)$ operations, and cannot be parallelized. This makes training on large datasets very slow.

- *How can we make this more efficient?*
  - Let's *just* use attention, and get rid of the recurrence.

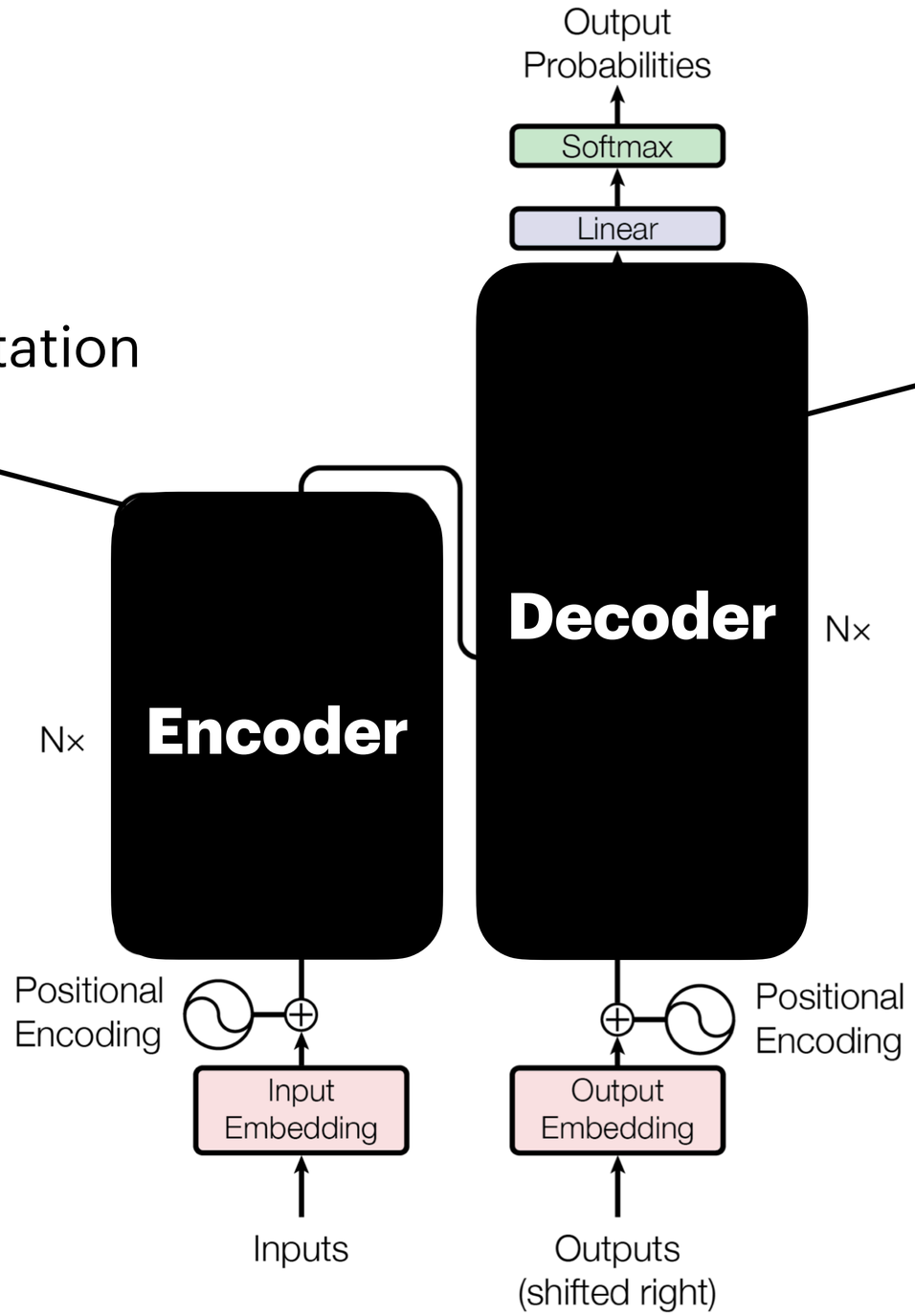
Intuition

- You can think of self-attention as doing two things:
    1. Figuring out which tokens relate to each other
    2. Figuring out what information to move between token positions

- Without recurrence, how do we preserve things like *sequence order*?
    - **Positional embeddings**
    - **Masking**
    - **Feed-forward layers**

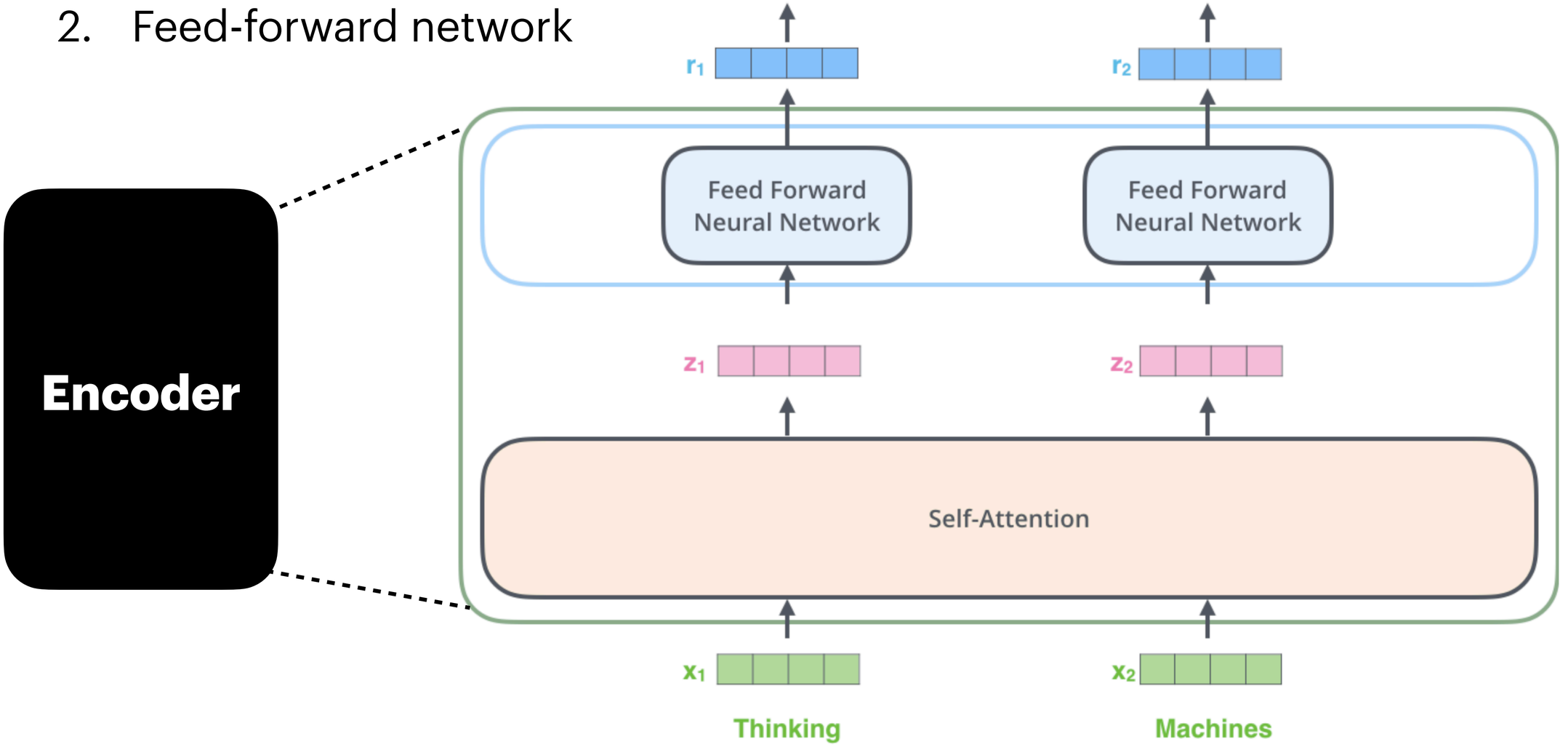
Reads the left *and* right context, yields representation of current token

Reads only the left context; generates the next token



Two main parts:

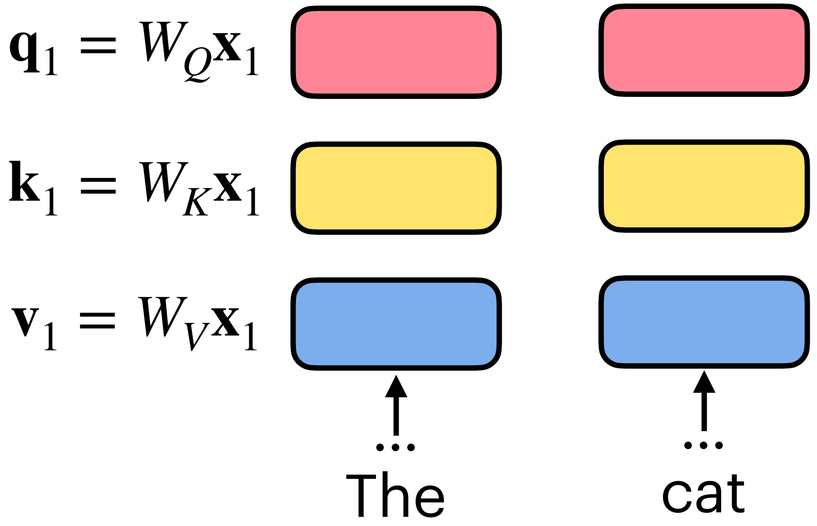
1. Self-attention
2. Feed-forward network



Self-attention

1. Derive three vectors from hidden state $\mathbf{x}_1$: the **query**, **key**, and **value**

Intuition: Think of retrieval. We have a query (input), which is compared to all keys. We retrieve the value(s) associated with the key(s) most similar to the query.



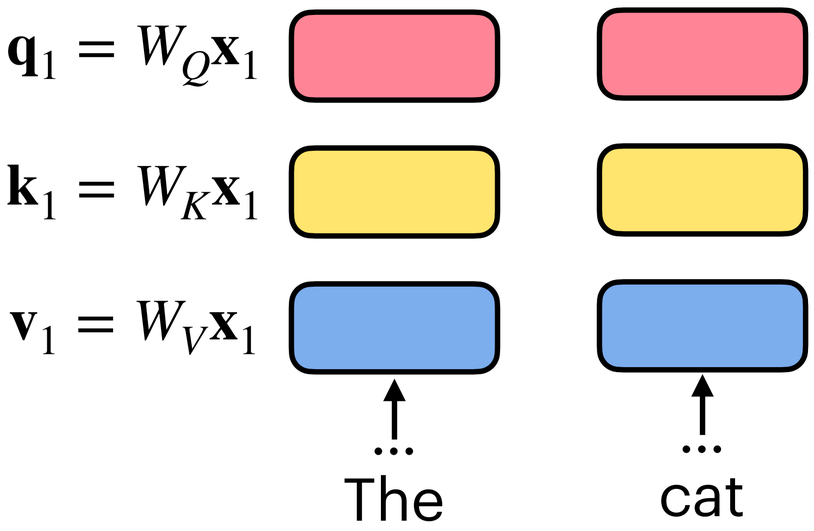
2. For all i, j, dot query of $\mathbf{x}_i$ with key of $\mathbf{x}_j$:

Attention score → $A_{ij} = \mathbf{q}_i \cdot \mathbf{k}_j$

3. Divide by square root of the dimensionality of $\mathbf{k}$ and softmax:

Attention weight → $\alpha_{ij} = \text{softmax}\left(\frac{A_{ij}}{\sqrt{d_\mathbf{k}}}\right)$

Self-attention



2. For all i, j, dot query of $\mathbf{x}_i$ with key of $\mathbf{x}_j$:

$$A_{ij} = \mathbf{q}_i \cdot \mathbf{k}_j$$

3. Divide by square root of the dimensionality of $\mathbf{k}$ and softmax:

$$\alpha_{ij} = \text{Softmax}\left(\frac{A_{ij}}{\sqrt{d_{\mathbf{k}}}}\right)$$

4. For all j, multiply α_{ij} by $\mathbf{v}_j$ and sum:

$$\mathbf{z}_i = \sum_{j=1}^{n} \alpha_{ij}\mathbf{v}_j$$

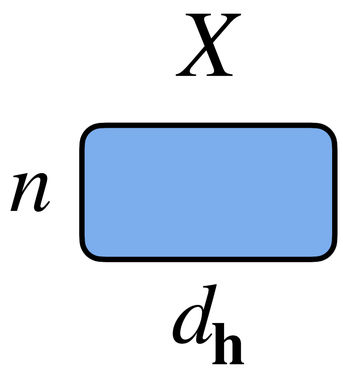
This is the output of the self-attention layer for token $\mathbf{x}_i$

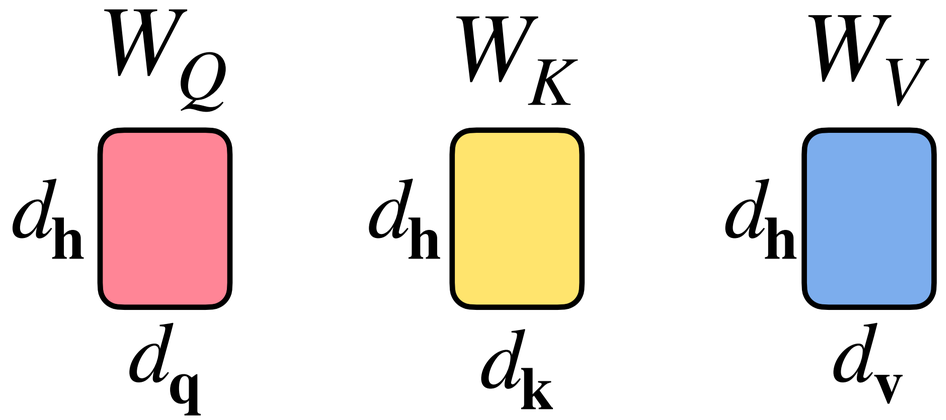
Self-attention

Batching attention computations

In practice, we want to be able to do all of this in parallel for all token positions. Let's do this in matrix form:

$$X = [\mathbf{x}_1; \mathbf{x}_2; \ldots; \mathbf{x}_n]$$





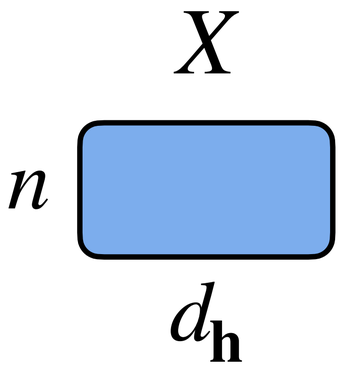
$$Z = \text{Softmax}\left(\frac{QK^\top}{\sqrt{d_\mathbf{k}}}\right)V$$

$$Q = XW_Q \qquad V = XW_V$$

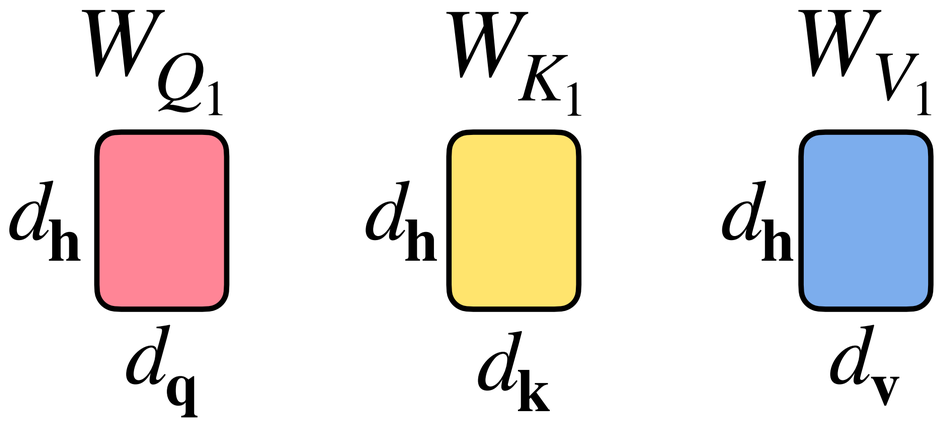
$$K = XW_K$$

Multi-head Self-attention

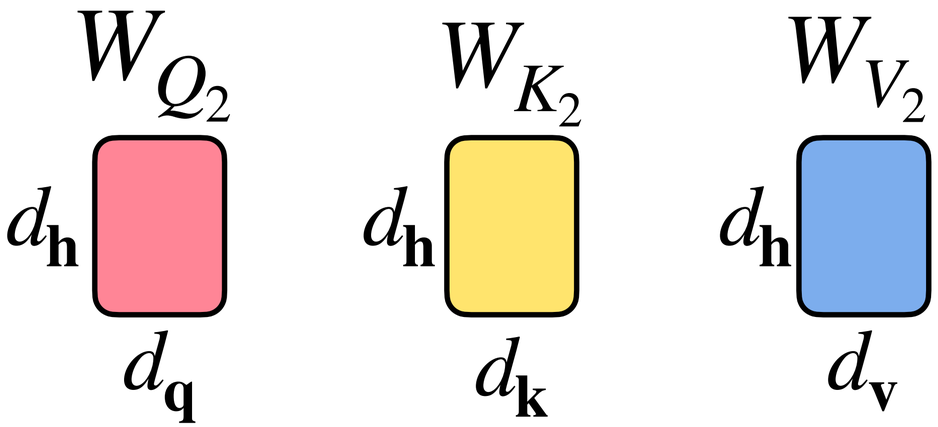
Now let's add more attention heads!



Head 1:



Head 2:

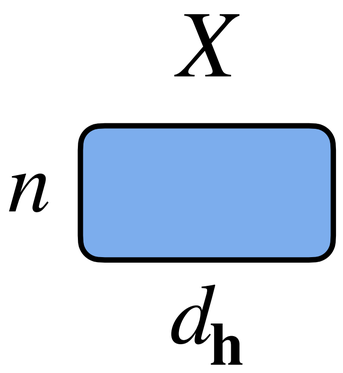


$$Z_1 = \text{Softmax}\left(\frac{Q_1 K_1^\top}{\sqrt{d_\mathbf{k}}}\right) V_1$$

$$Z_2 = \text{Softmax}\left(\frac{Q_2 K_2^\top}{\sqrt{d_\mathbf{k}}}\right) V_2$$

Multi-head Self-attention

Now let's add more attention heads!



But wait, the model expects the output of the attention block to be the size of a single Z. How do we convert all these Z_i into a Z-shaped matrix?

Concatenate then project:

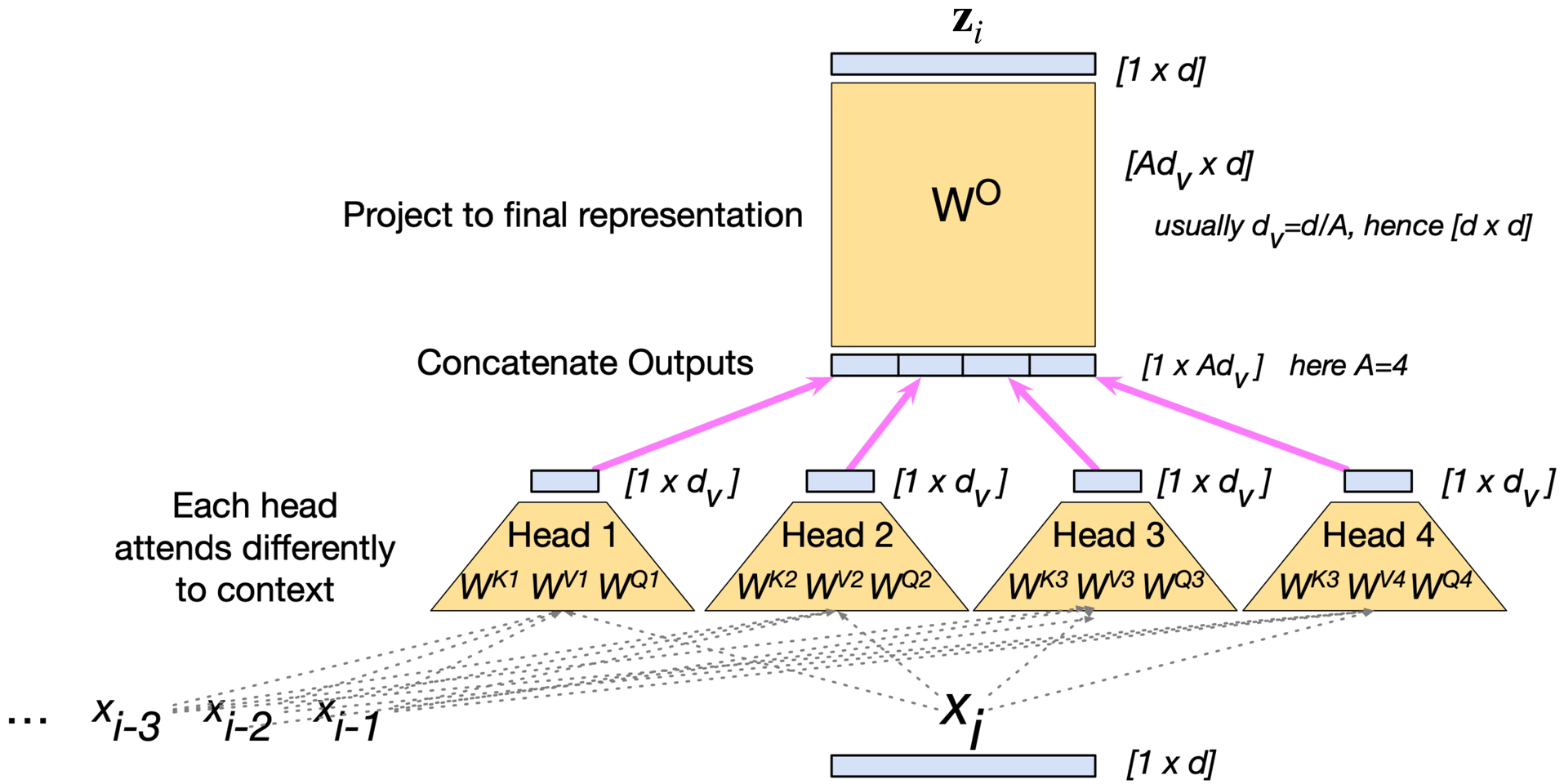
$$Z_1 = \text{Softmax}\left(\frac{Q_1 K_1^\top}{\sqrt{d_{\mathbf{k}}}}\right) V_1$$

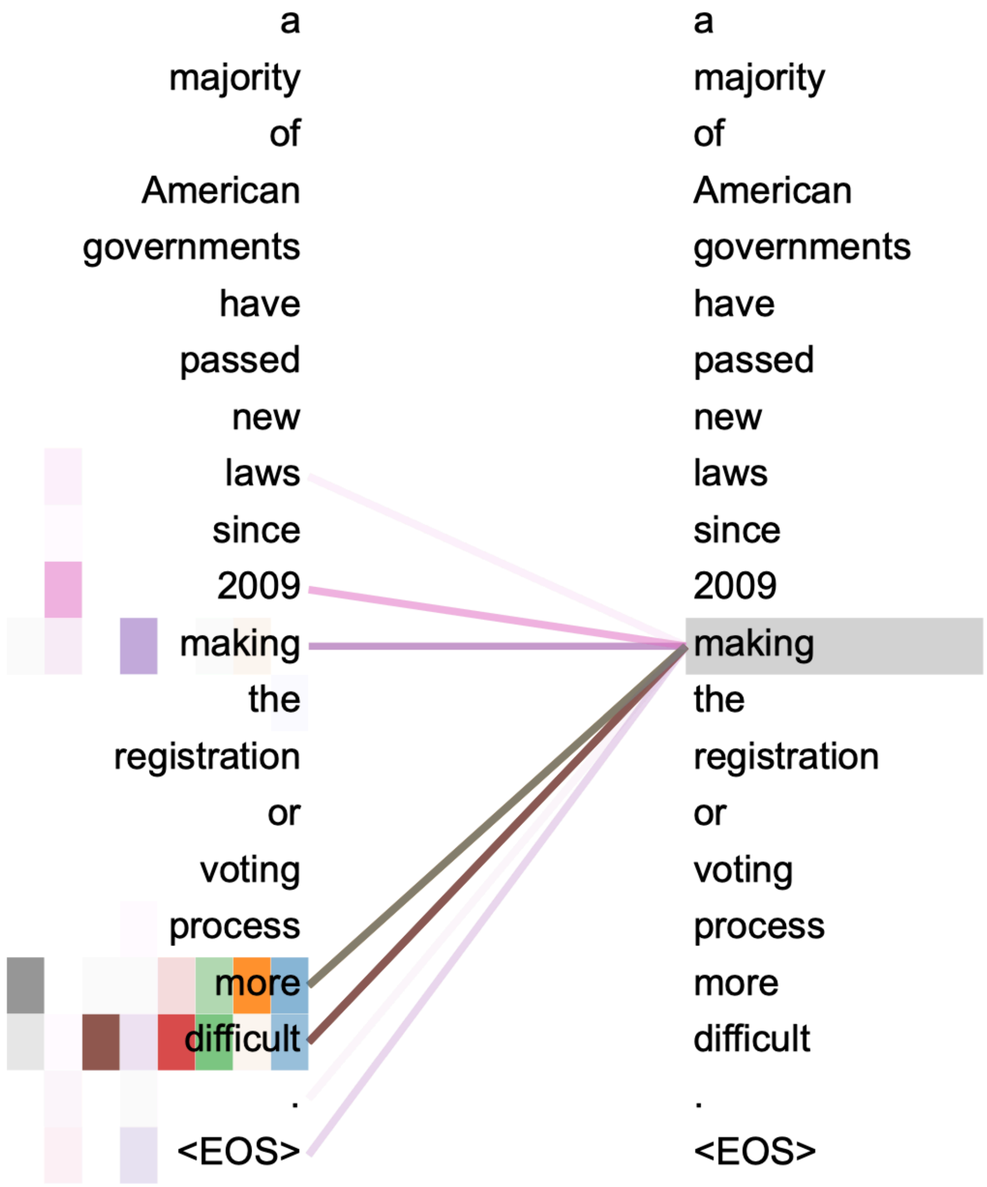
$$Z_2 = \text{Softmax}\left(\frac{Q_2 K_2^\top}{\sqrt{d_{\mathbf{k}}}}\right) V_2$$

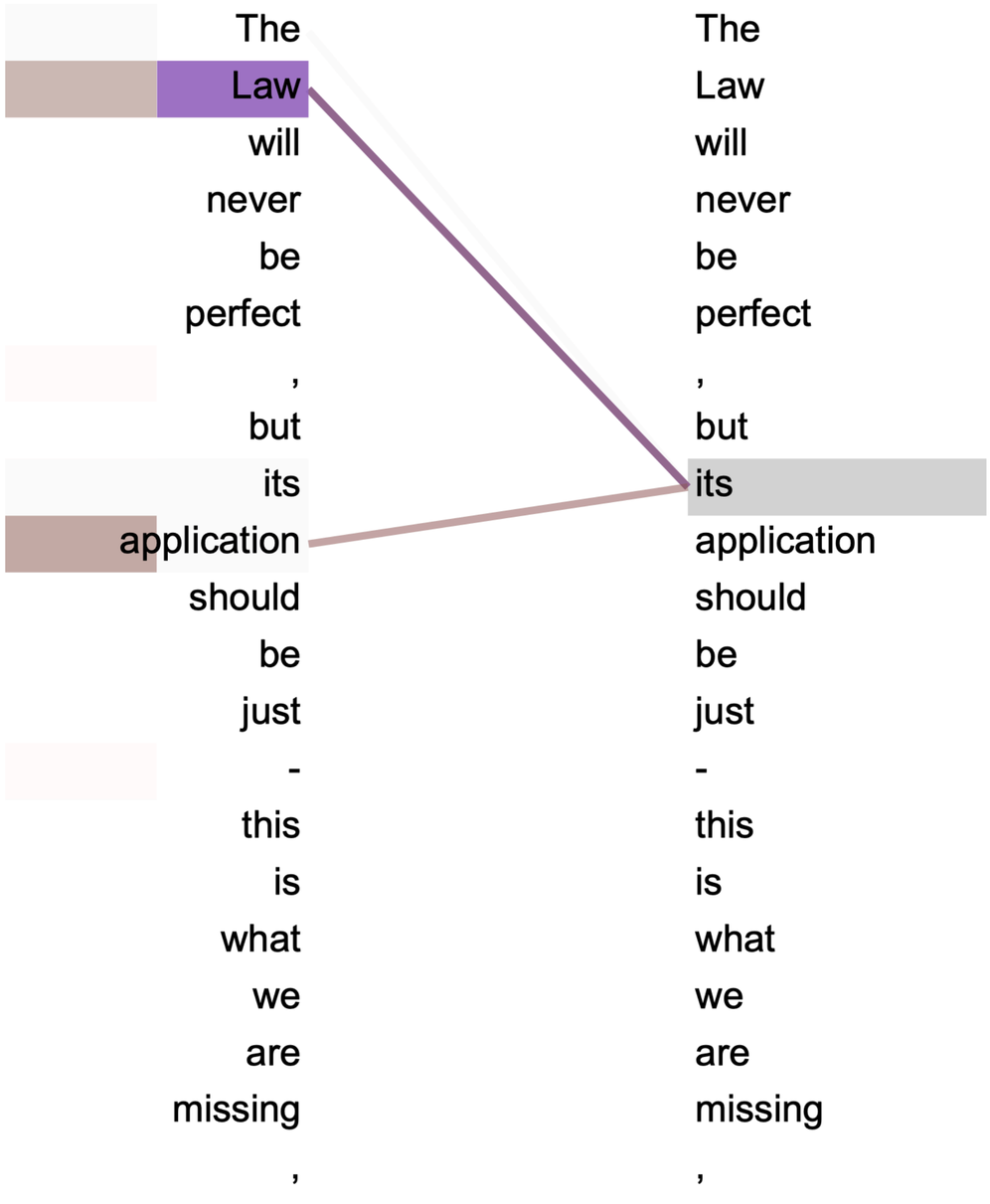
$$Z = \begin{array}{c} Z_1 \quad Z_2 \\ \square\square \end{array} W_O$$

Output projection

Multi-head Self-attention




a
majority
of
American
governments
have
passed
new
laws
since
2009
making
the
registration
or
voting
process
more
difficult
.
<EOS>
a
majority
of
American
governments
have
passed
new
laws
since
2009
making
the
registration
or
voting
process
more
difficult
.
<EOS>


The
Law
will
never
be
perfect
,
but
its
application
should
be
just
-
this
is
what
we
are
missing
,
The
Law
will
never
be
perfect
,
but
its
application
should
be
just
-
this
is
what
we
are
missing
,

Contextual Representations

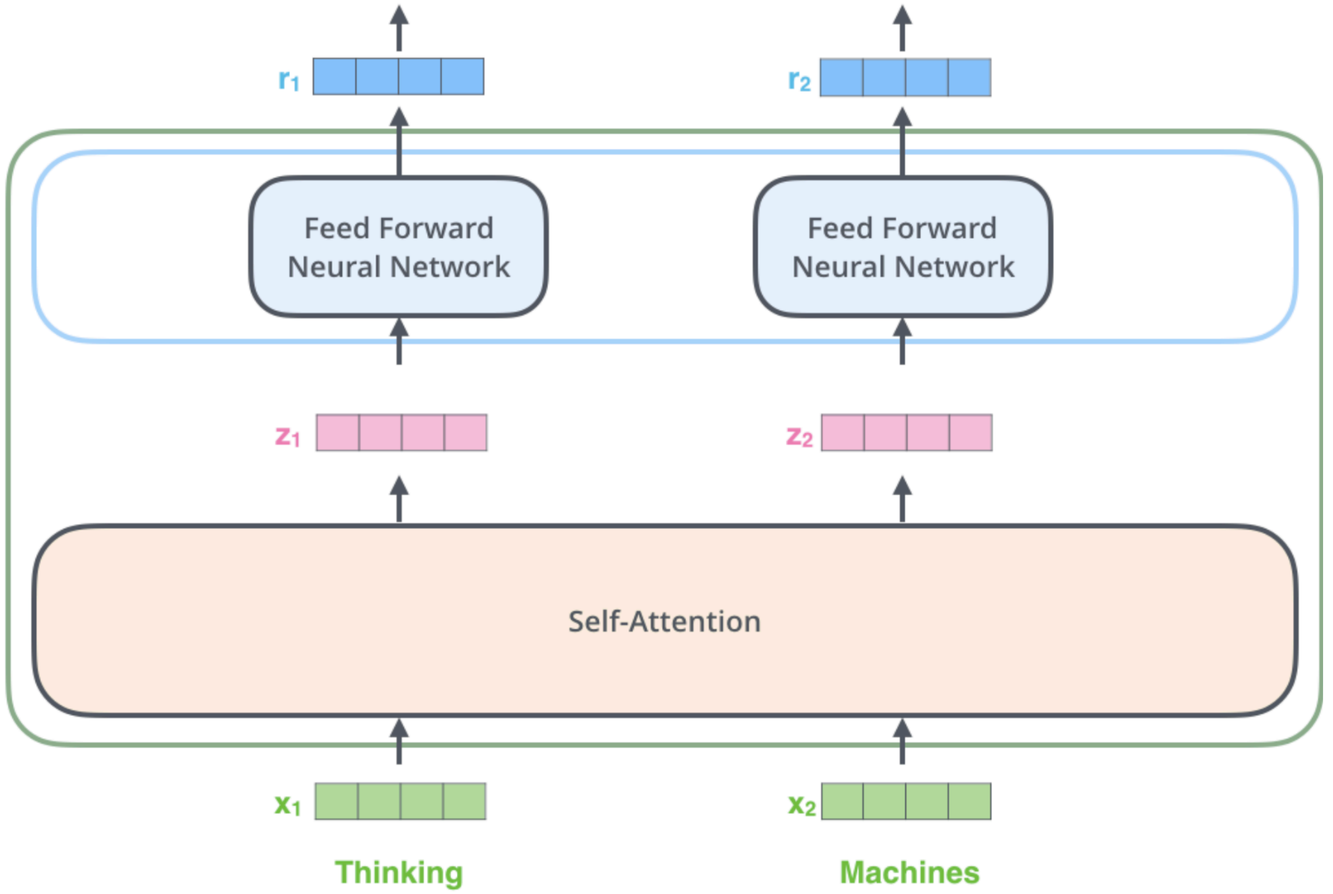
- Because they receive information from other tokens, token representations $\mathbf{r}_i$ after a Transformer block carry contextual information!

E.g., $\mathbf{r}_i$ can distinguish between:

Let's go to the **park**

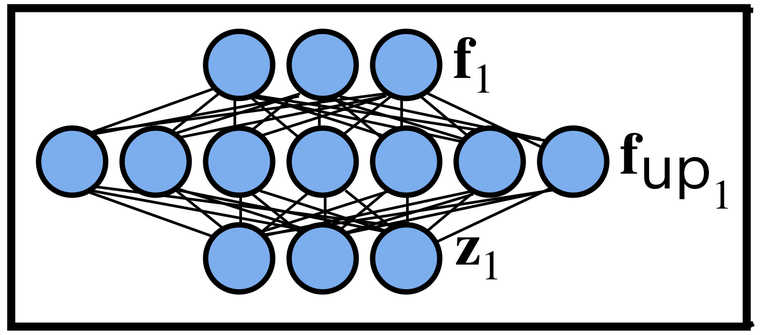
Let me **park** my car

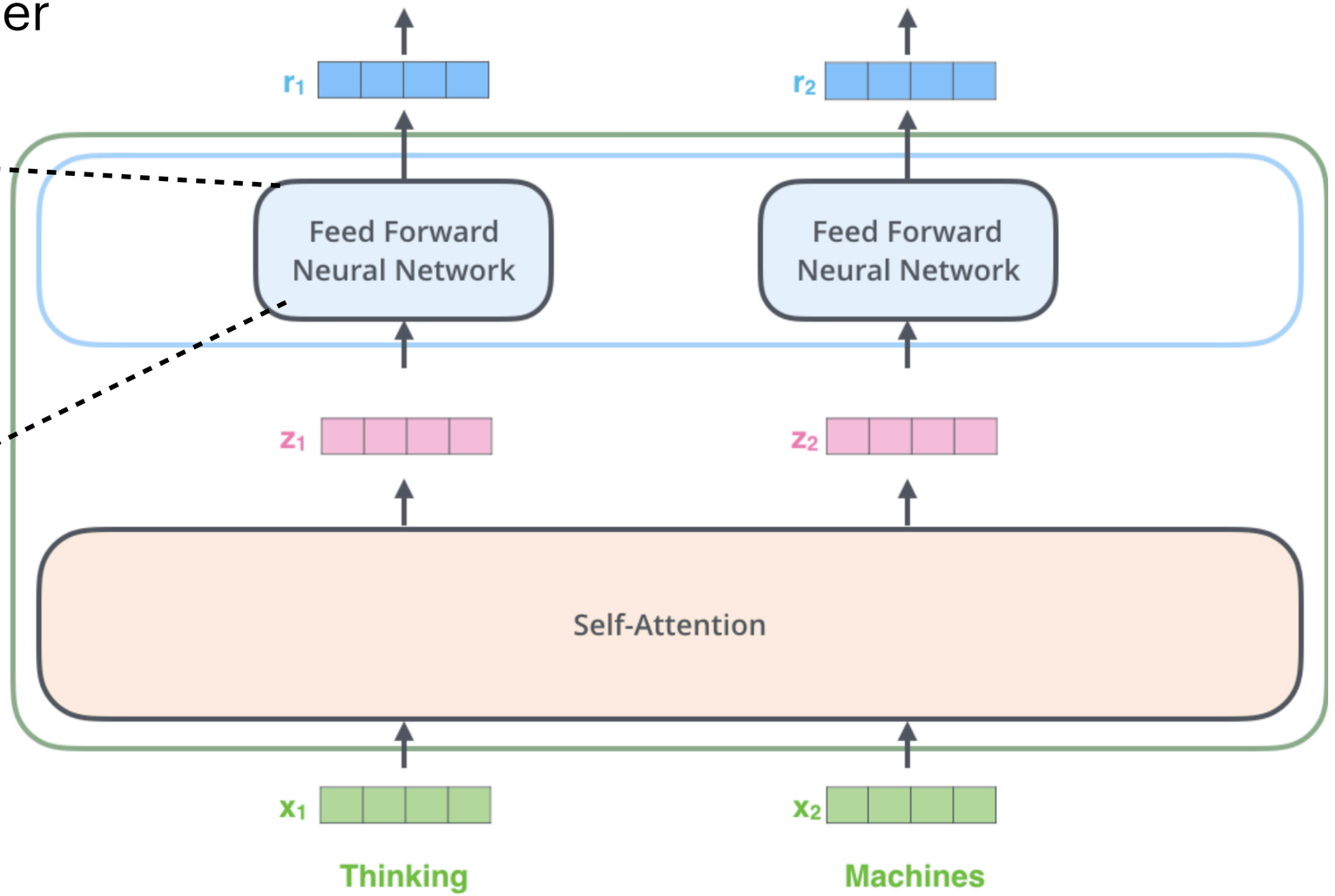
Let's **park** that for now



Attention is totally linear... so where do the non-linearities come from?

We add feed-forward layers after the attention:

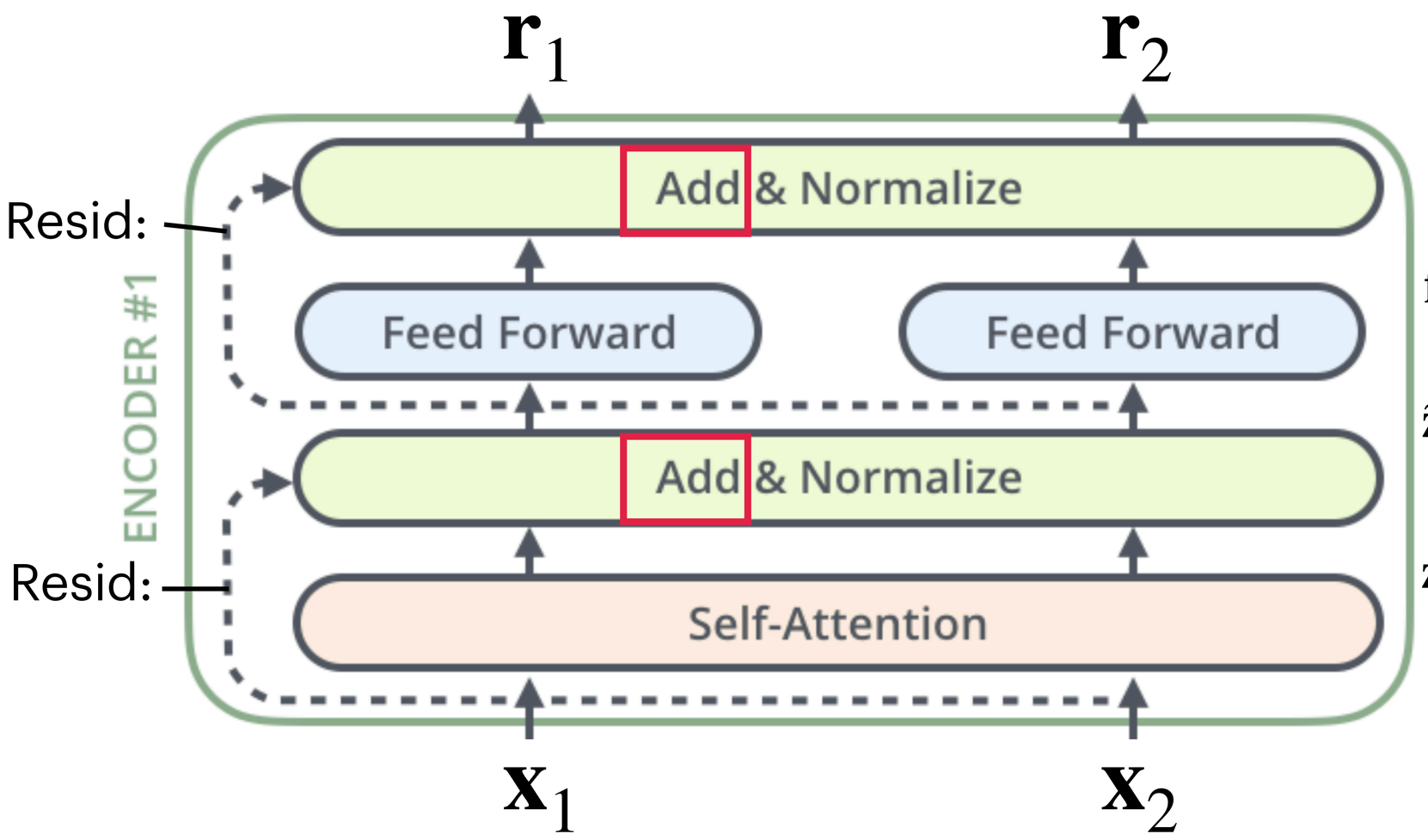




$$\mathbf{f}_{\text{up}} = \text{ReLU}(\mathbf{z}_1 W_{\text{up}} + \mathbf{b}_1)$$

$$\mathbf{f}_1 = \mathbf{f}_{\text{up}} W_{\text{down}} + \mathbf{b}_2$$

Residual Connections



$\mathbf{f}_j$

$\tilde{\mathbf{z}}_j$

$\mathbf{z}_j$

$R = \text{LayerNorm}(\tilde{Z} + F)$

$\tilde{Z} = \text{LayerNorm}(X + Z)$

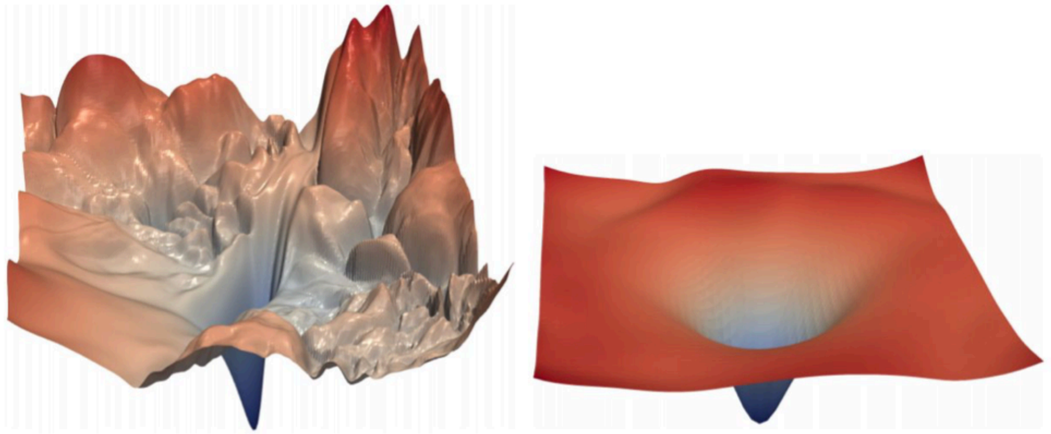
Residual connections help prevent vanishing gradients. Gives gradients more direct paths from later layers to earlier layers.

Why residual connections?

1. It helps gradients not vanish as easily.

2. It makes the loss landscape much easier to optimize over:

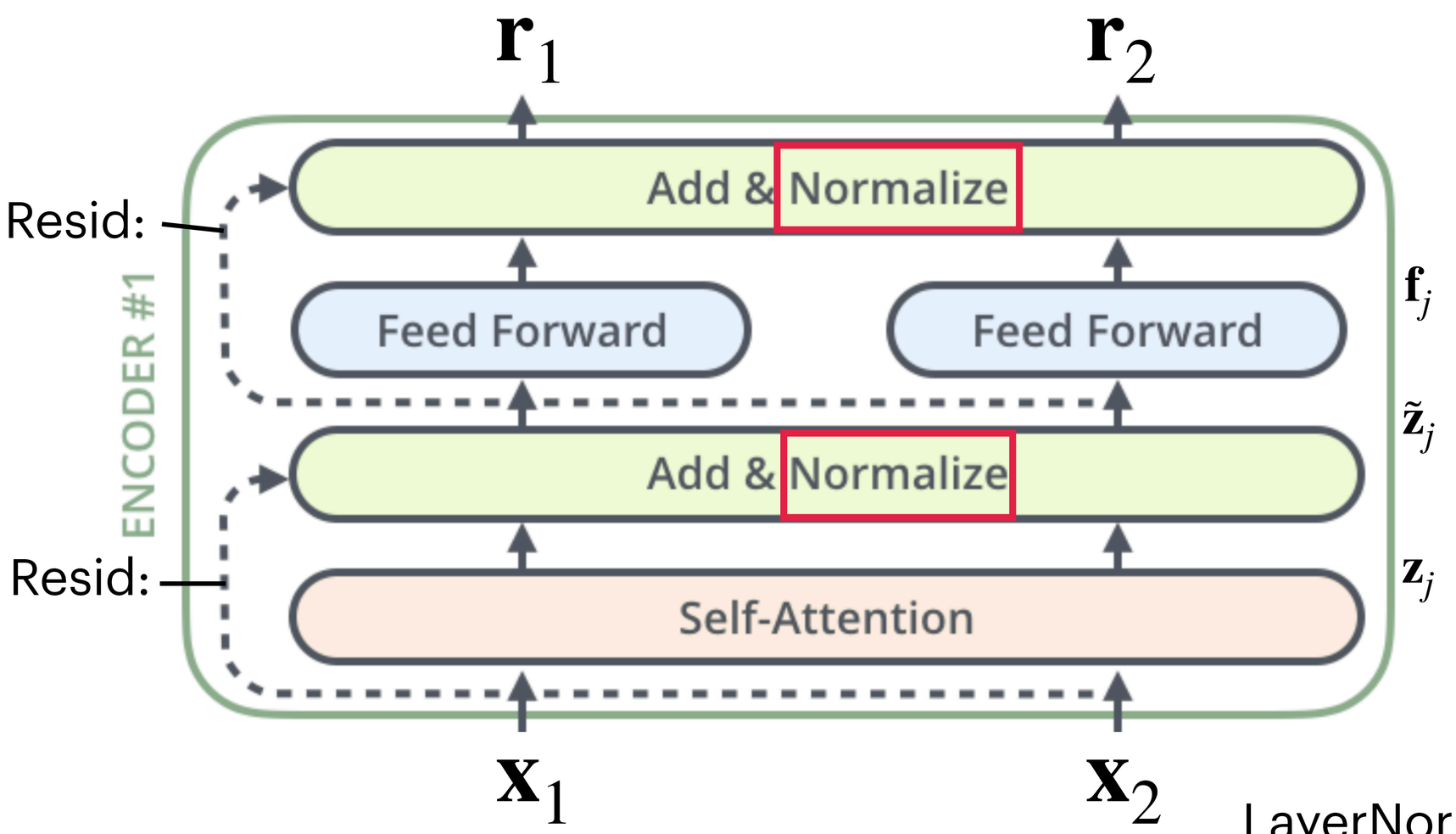
Loss landscapes:



(no residuals) (with residuals)

[Li et al., 2018]

LayerNorm



$$R = \text{LayerNorm}(\tilde{Z} + F)$$

$$\tilde{Z} = \text{LayerNorm}(X + Z)$$

Learnable parameters

Mean

$$\text{LayerNorm}(\mathbf{v}) = \gamma \frac{\mathbf{v} - \mu}{\sigma} + \beta$$

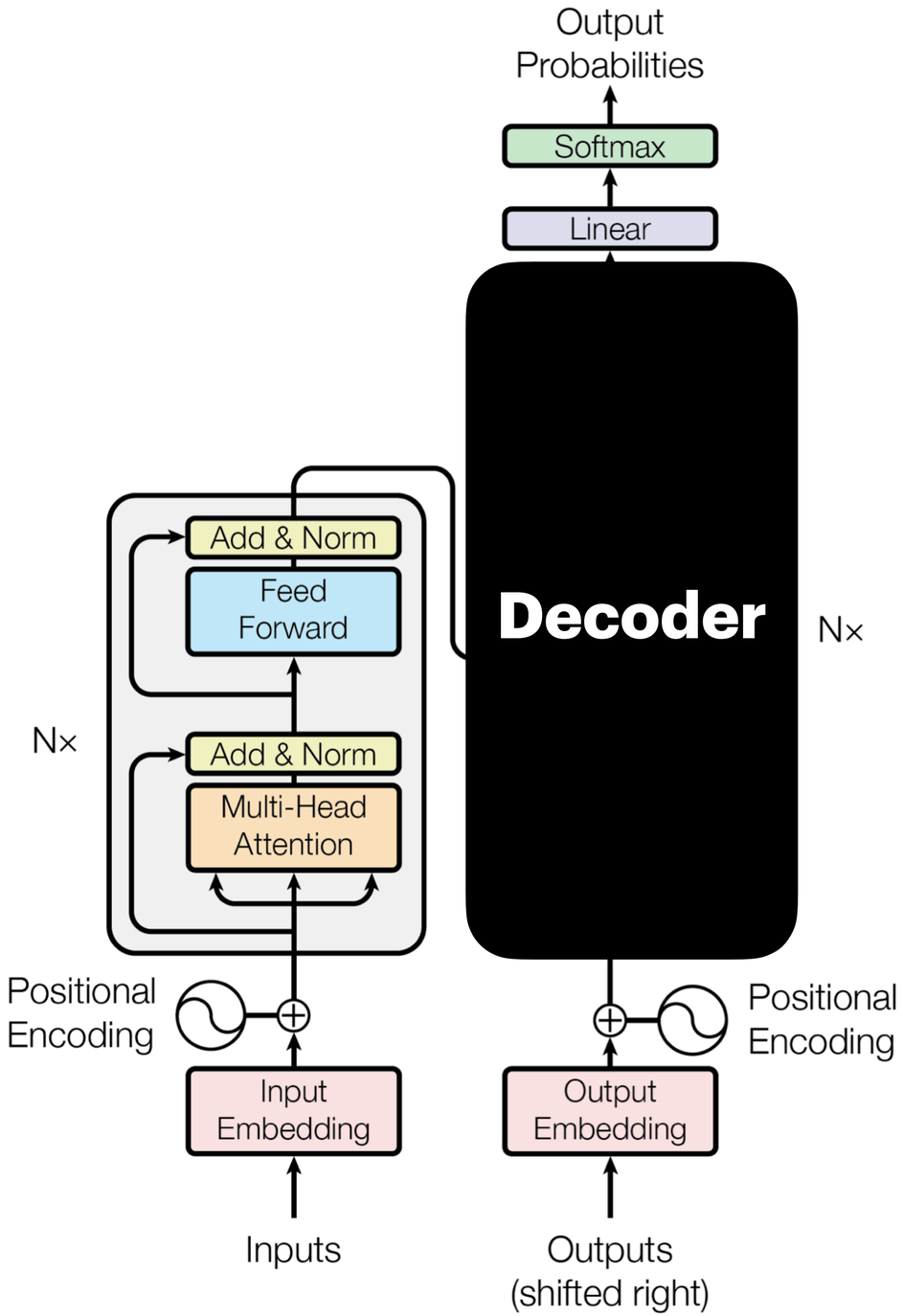
Std. dev.

Why LayerNorm?

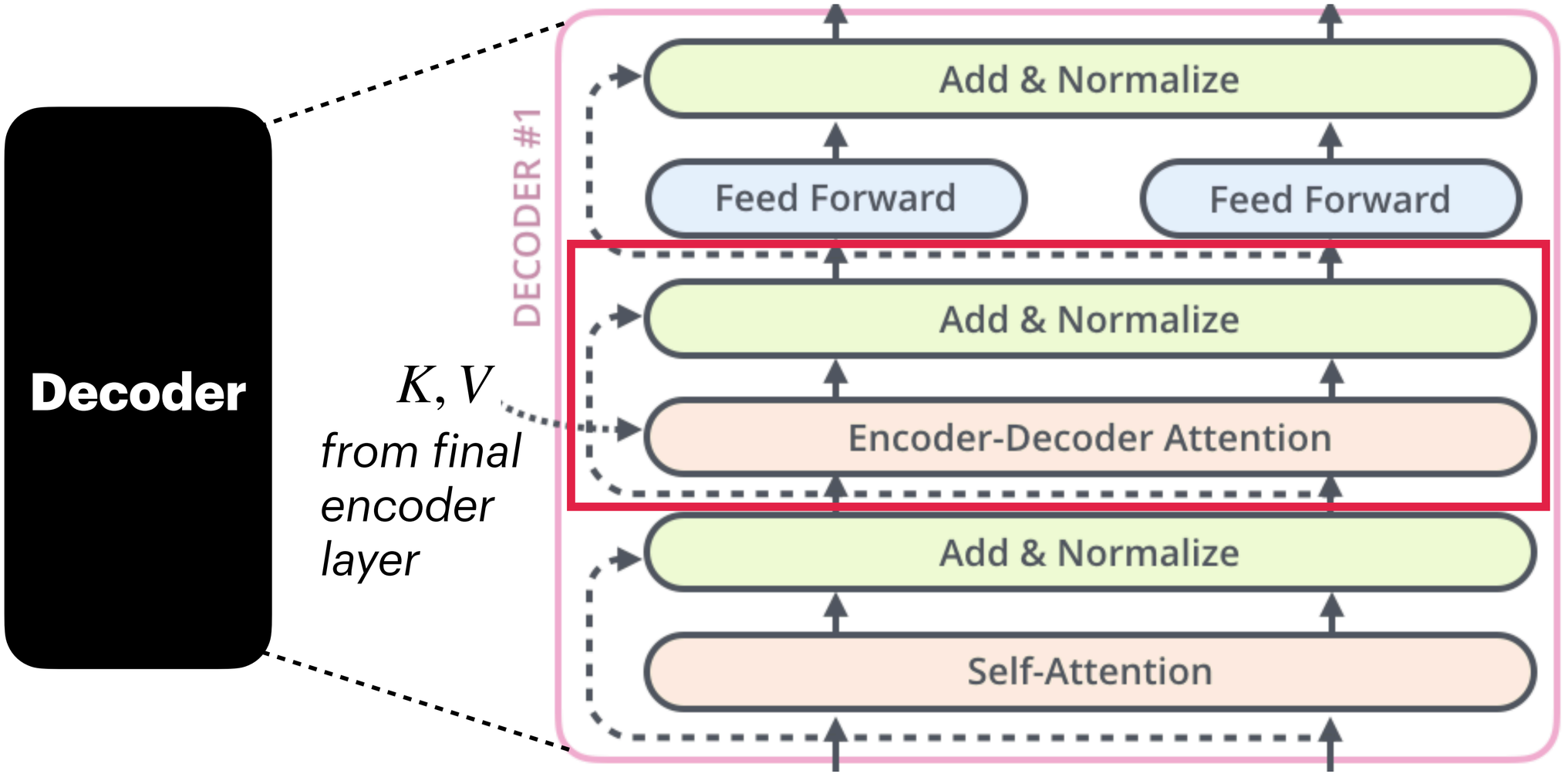
- It helps models train faster.
  - Cuts down on uninformative variance in hidden vectors within each layer.
  - Helps normalize gradients.
- The parameters γ and β help determine how much we should normalize:

$$\gamma \frac{\mathbf{v} - \mu}{\sigma} + \beta$$

"gain" "offset"


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

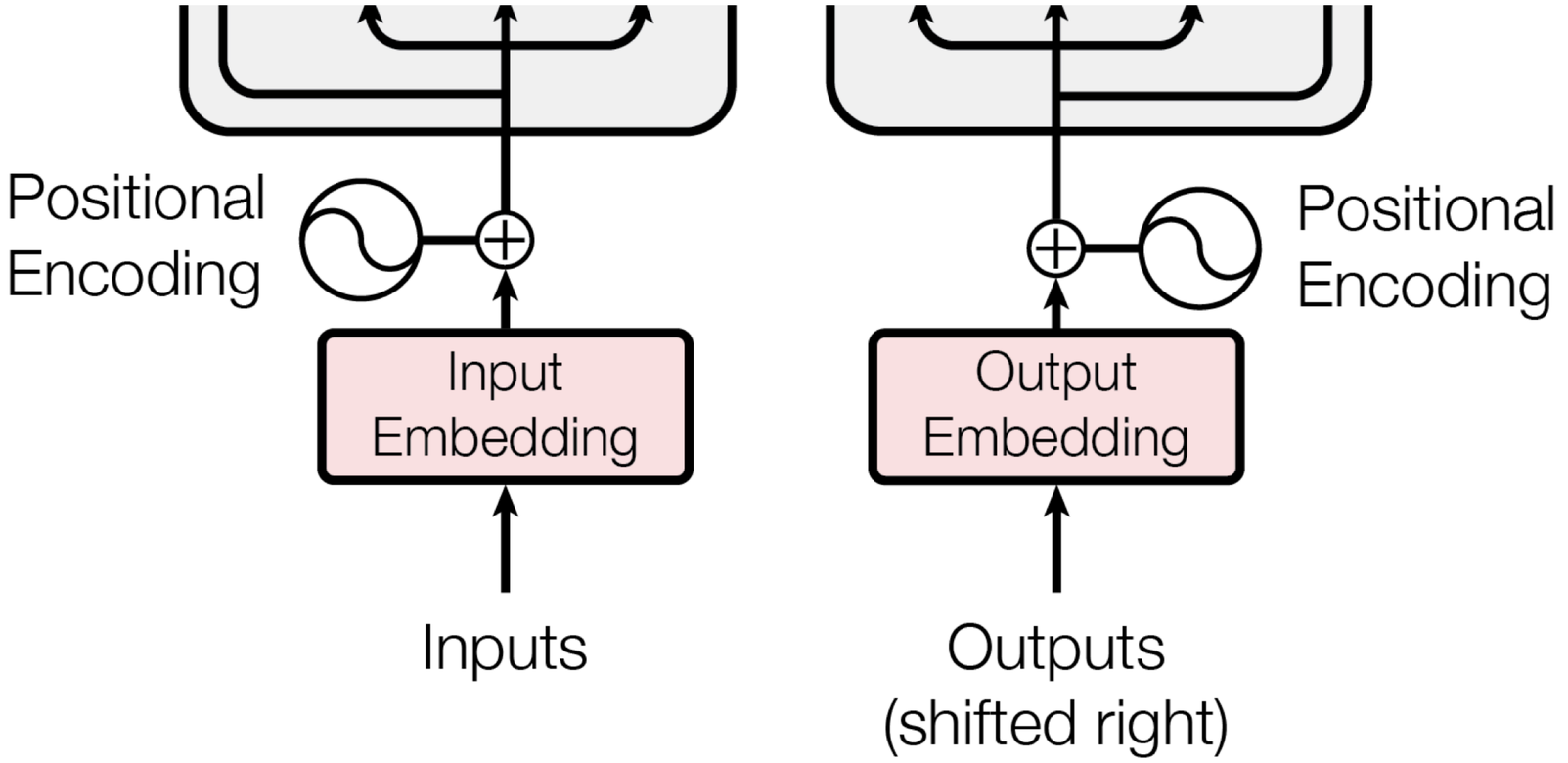
The Decoder Block



Positional Embeddings

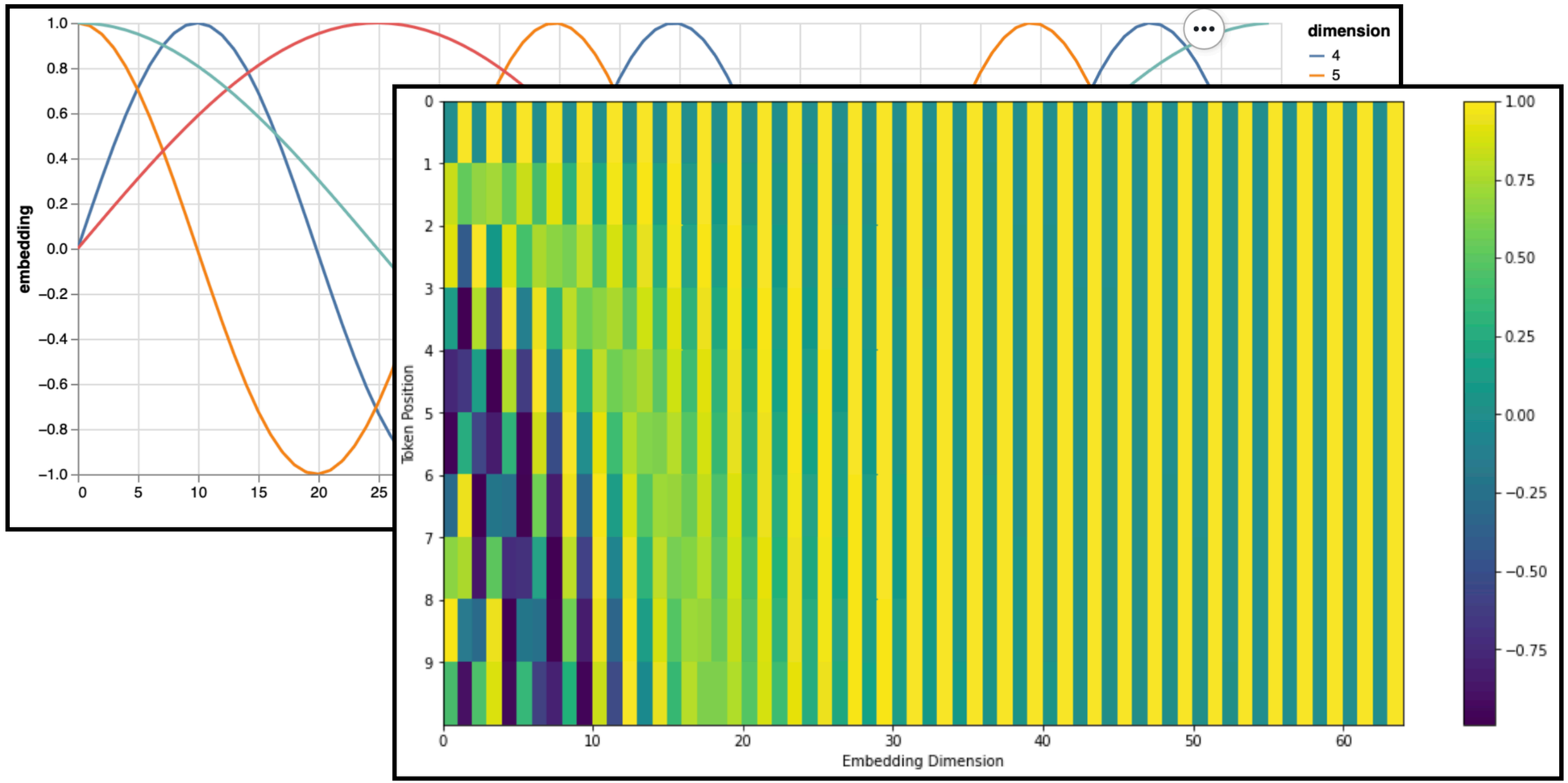
"Since our model contains no recurrence and no convolution, in order for the model to make use of the order of the sequence, we must inject some information about the relative or absolute position of the tokens in the sequence."

—Vaswani et al. (2017)



Positional Embeddings

Encode position with sine/cosine:



Positional Embeddings

n: position in sequence

i: index in embedding vector

d: dimensionality of embedding vector

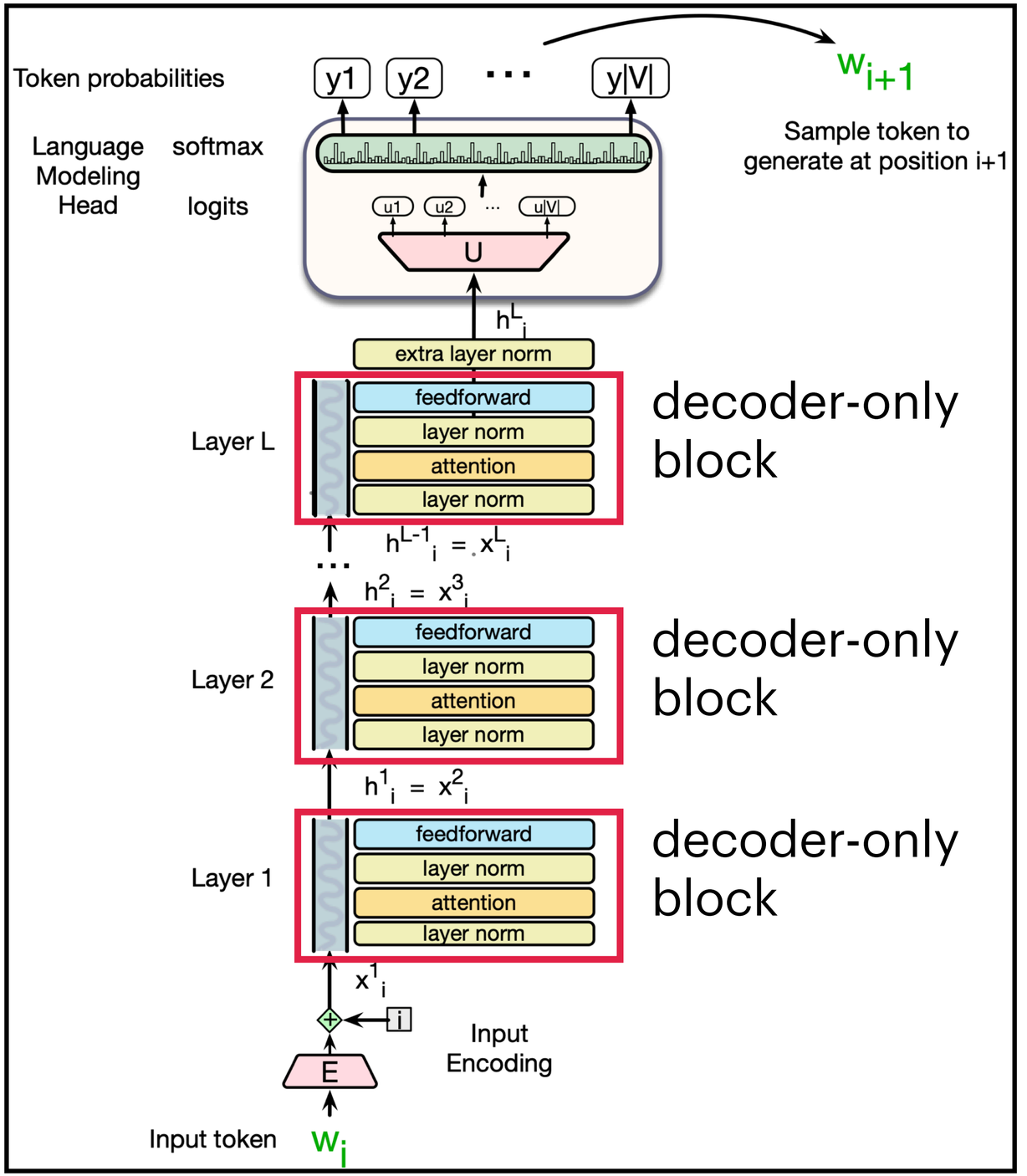
Even indices: $P(n, 2i) = \sin\left(\frac{n}{10000^{2i/d}}\right)$

Odd indices: $P(n, 2i+1) = \cos\left(\frac{n}{10000^{2i/d}}\right)$

Why this over just using the position integer?

- Integers aren't really inherently meaningful
- Sines/cosines specify position well via frequency and phase

- In practice, encoder-decoder models are rare these days.
- **Decoder-only models** are very common. How can we set these up without the encoder, and without encoder-decoder attention?
- Very simple: it looks almost just like the encoder block, but with a **causal mask.**



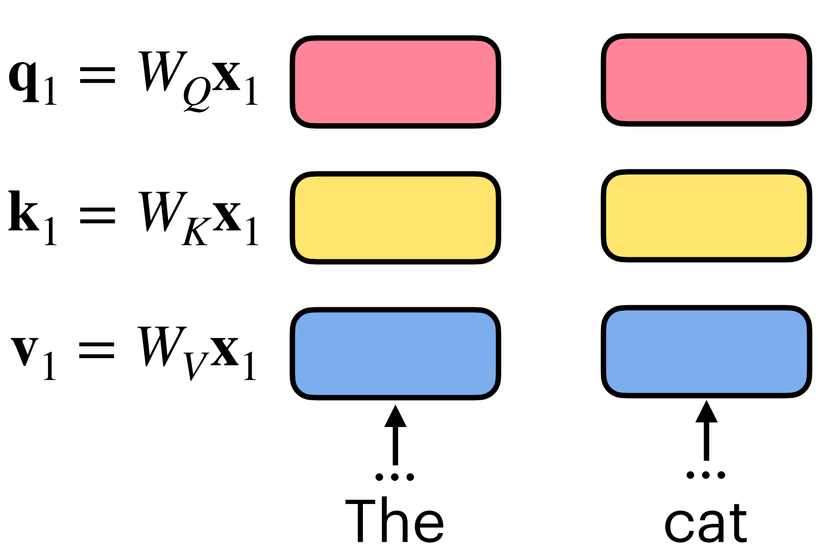
Decoder-*only* Transformer Blocks

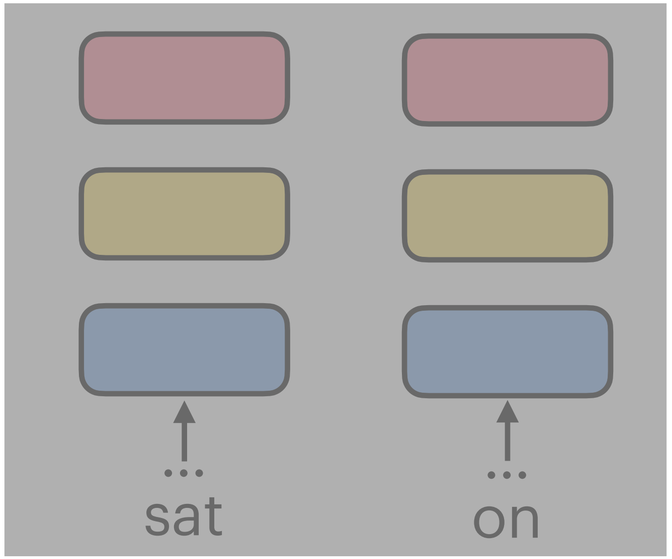
Encoder: $\alpha_{ij} = \text{Softmax}\left(\frac{A_{ij}}{\sqrt{d_{\mathbf{k}}}}\right)$

Decoder: before the softmax, mask out everything above the diagonal (i.e., set those elements to $-\infty$)

At position i, mask out attention to all tokens at positions $> i$

"causal mask"

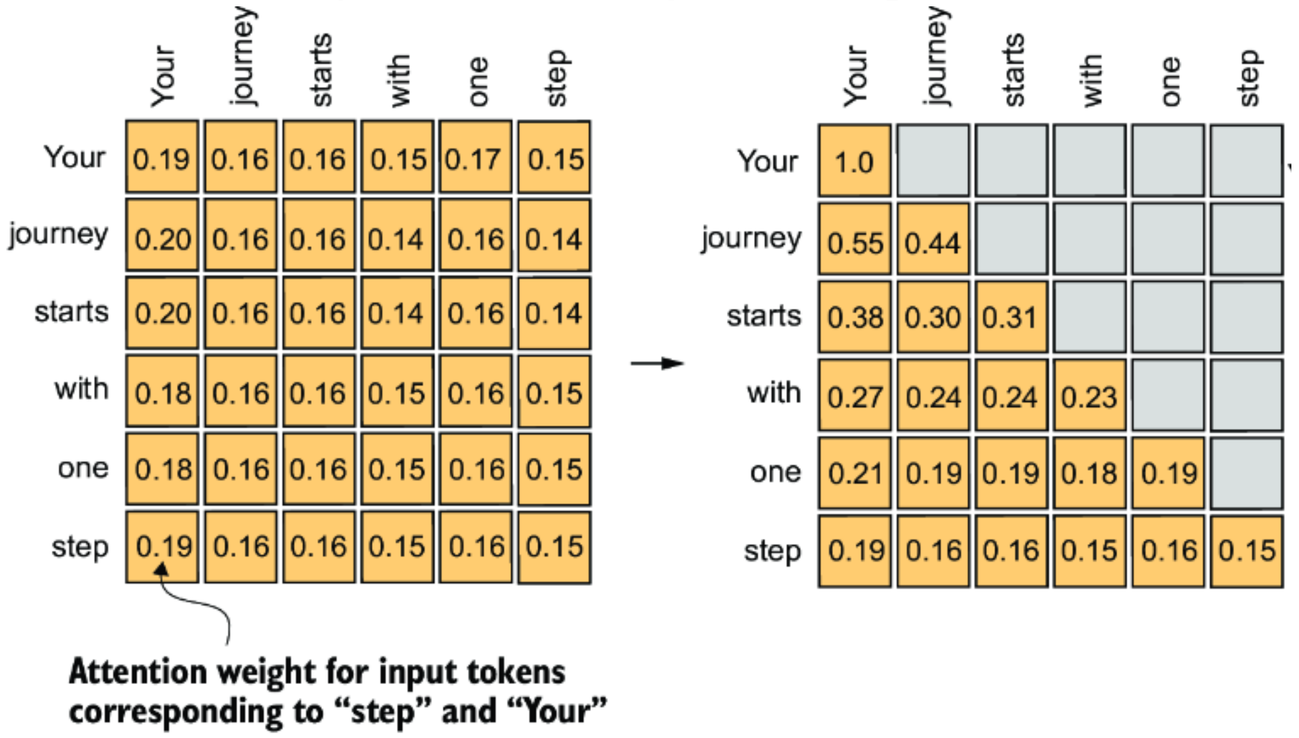




$$\text{Softmax}\left(\begin{bmatrix} 1.1 & -\infty & -\infty & -\infty \\ -0.1 & 0.5 & -\infty & -\infty \\ 0.5 & 1.2 & 3.3 & -\infty \\ 5.1 & 2.9 & -0.3 & 4.2 \end{bmatrix}\right)$$

Why a causal mask?

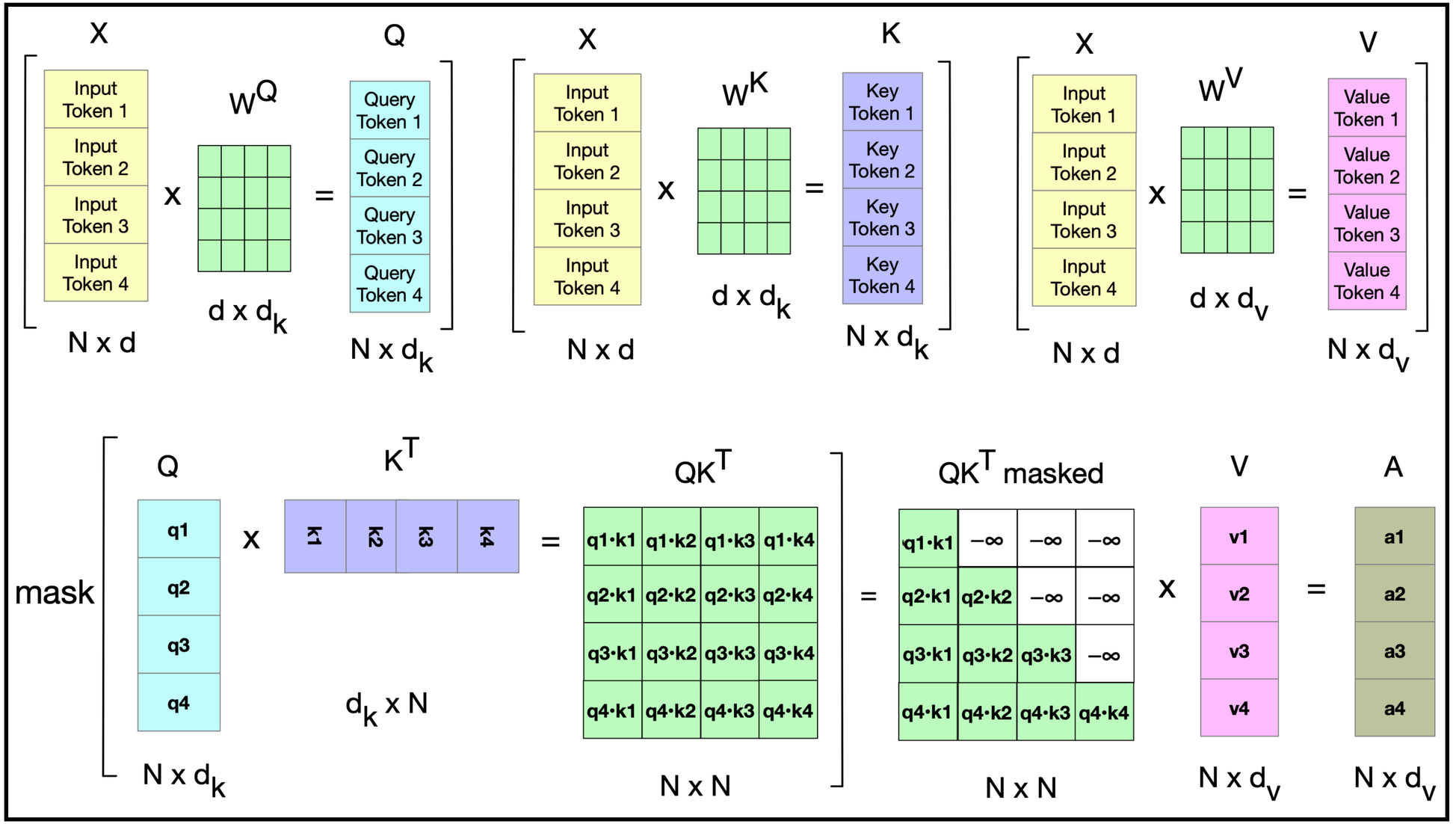
- The causal mask is what turns the Transformer into a generative language model.
  - Left-to-right information flow
- Without it, the model could just cheat by looking ahead at the next tokens.



Self-attention with the Causal Mask

These are all of the computations needed for one attention head.

Multi-head attention is just a bunch of these performed in parallel and then recombined.



Mathematical Summary

Given a sequence of token embeddings X, a Transformer block does the following:

(The textbook puts the LayerNorms *before* the MHA and FFNs instead of after. This works a bit better in practice; for the HW, feel free to do either.)

$$T^1 = \text{MultiHeadAttention}(X)$$

$$T^2 = T^1 + X$$

$$T^3 = \text{LayerNorm}(T^2)$$

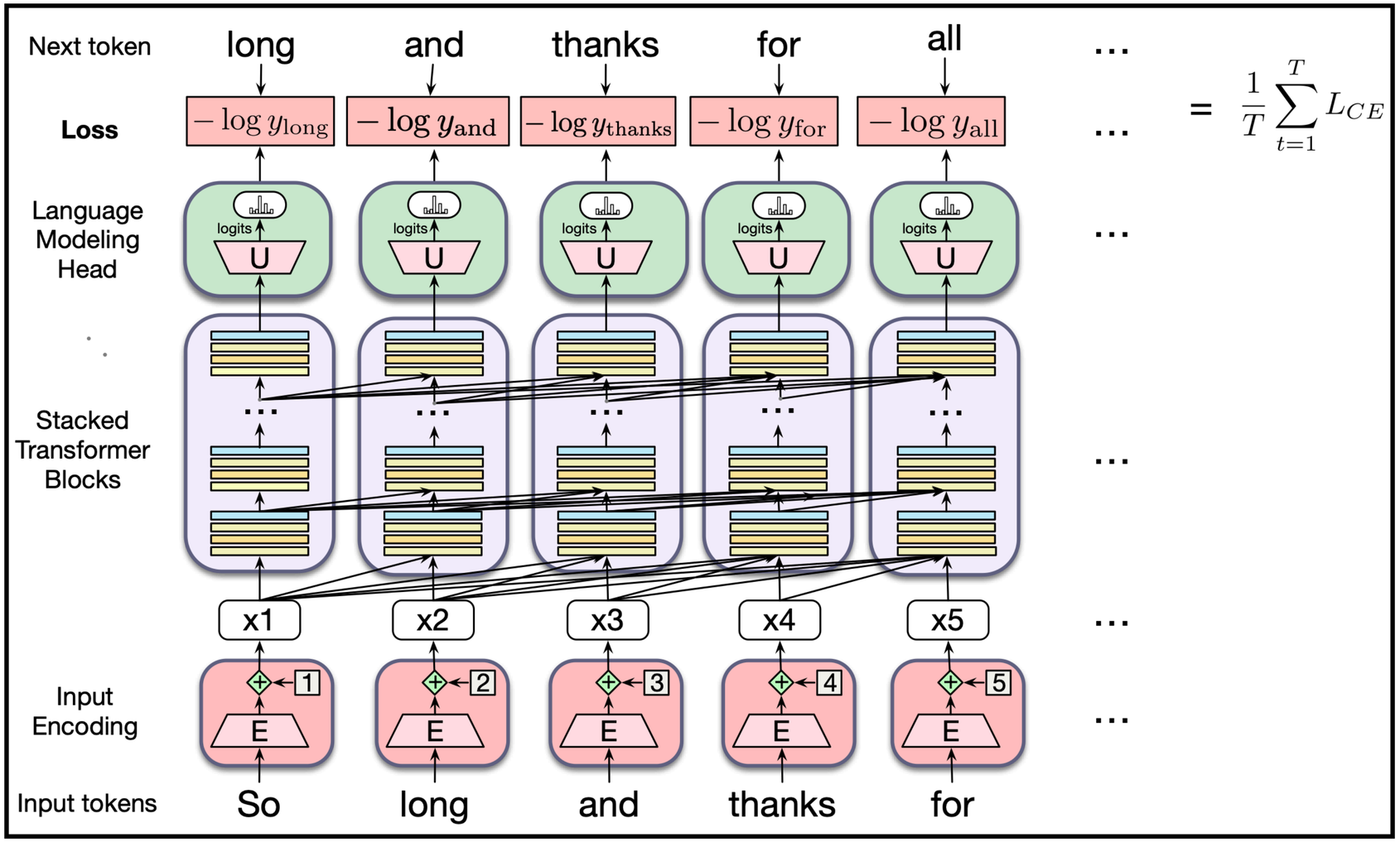
$$T^4 = \text{FFN}(T^3)$$

$$T^5 = T^4 + T^3$$

$$R = \text{LayerNorm}(T^5)$$

Training

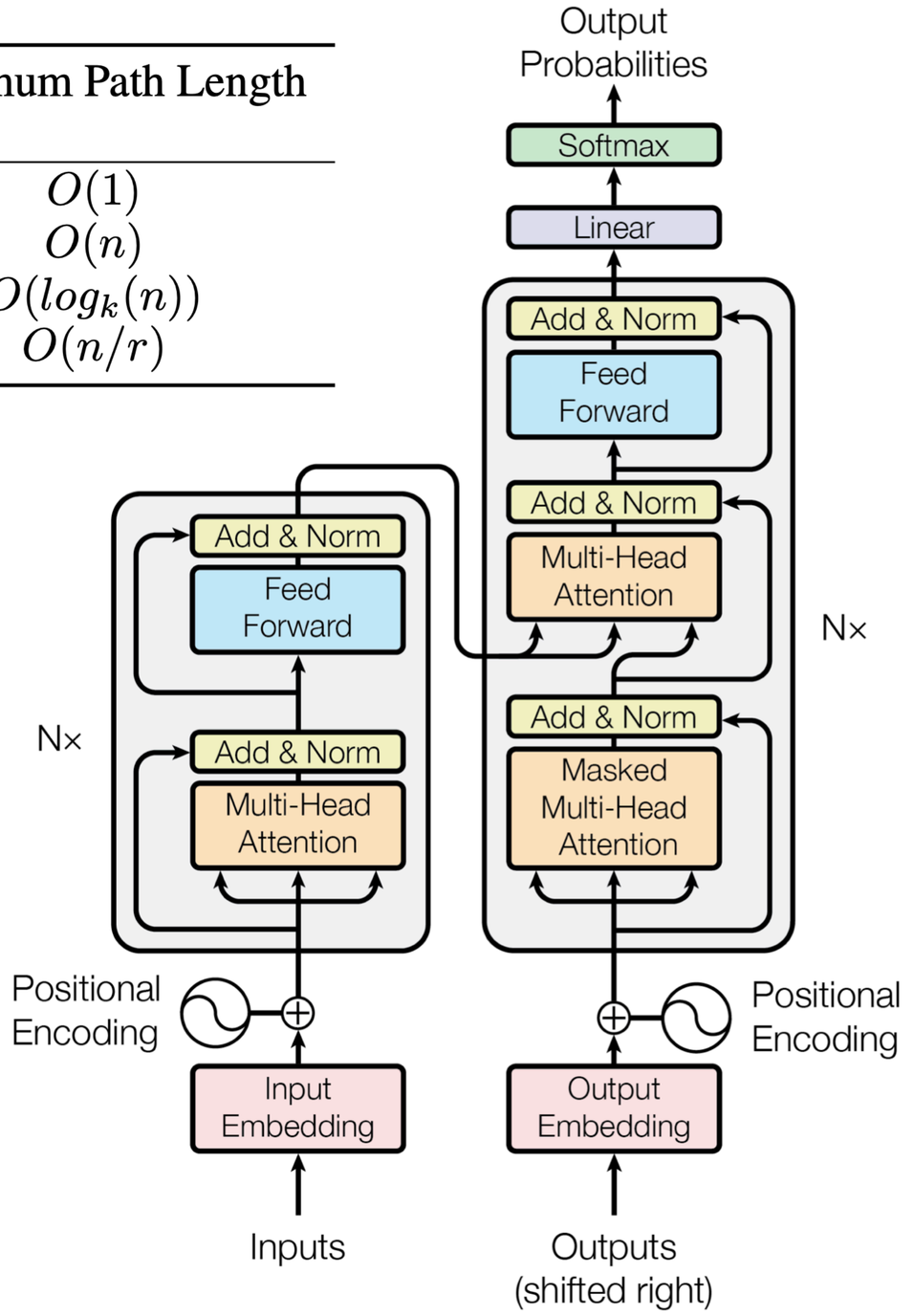
- Same cross-entropy loss as usual
- You can process each training item (each token) *in parallel*
- We usually pack as many sequences as we can into the context window (4096 tokens for GPT-4)



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)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

Transformers are technically more compute-intensive than RNNs, but they can be massively parallelized, making Transformers much faster to train in practice.

"As [a] side benefit, self-attention could yield more interpretable models."



Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [38]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	**$3.3 \cdot 10^{18}$**	
Transformer (big)	**28.4**	**41.8**	$2.3 \cdot 10^{19}$	

Transformers also perform well on downstream tasks

...using an order-of-magnitude fewer FLOPs than past methods

Impact

- With transformers, we can train neural networks that are just as good as before, but using orders-of-magnitude less compute
- Transformers significantly reduced the problem of long sequence modeling
  - (Granted, memory requirements are now quadratic w.r.t. sequence length)
- Still the dominant architecture in language modeling
- Look around us:




Some Drawbacks

- **Quadratic dependencies w.r.t. length**: we now need $O(n^2)$ compute as sequences increase in length.
  - With recurrent models, it was linear!

- The **position representations** can almost certainly be improved.

Quadratic Dependencies

- Attention is highly parallelizable, but consider the fact that every token has pairwise interactions with every other token:

$$XQ \times K^\top X^\top = XQK^\top X^\top$$

- This means we need to perform $O(n^2 d)$ operations, where n is context length and d is the dimensionality.
- This becomes prohibitive when working with long documents.
- People have tried to remove this dependency, but methods that do so don't usually perform as well.