UNIVERSITY OF GEORGIA

CSCI 4360/6360 Data Science II

Transformers

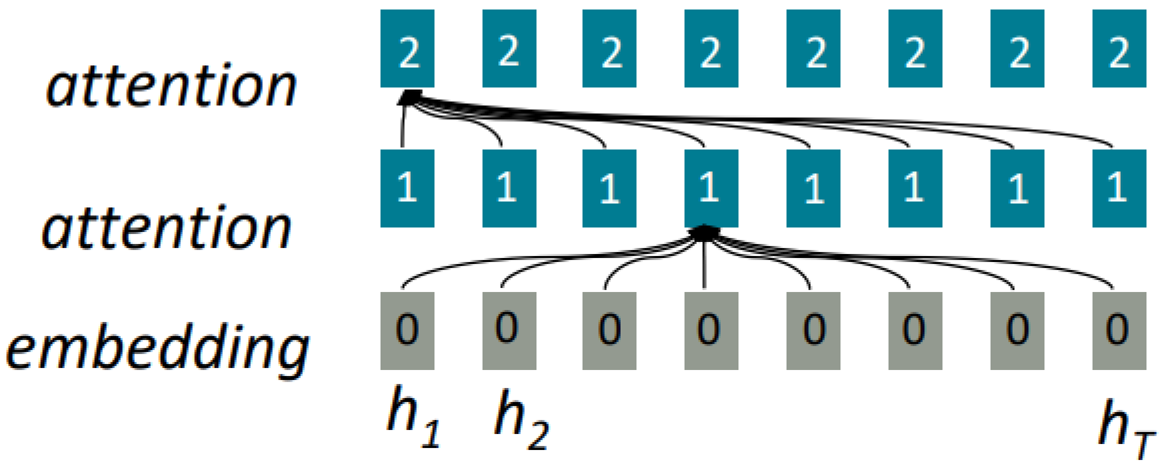
Ninghao Liu
Assistant Professor
School of Computing
University of Georgia

The majority of contents are adopted from Stanford CS224N/Ling284. (Abigail See and Richard Socher)

Where we left off

Self-Attention:

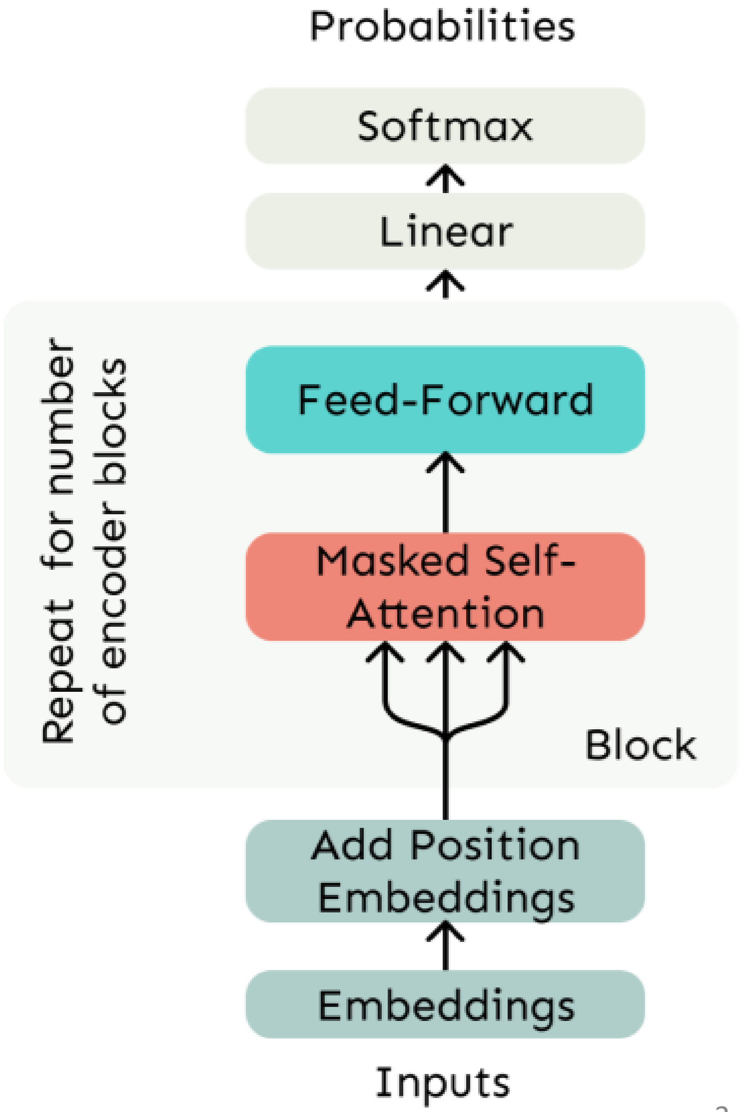
- Words attend to themselves.
- A word plays the role of query, key, and value simultaneously.



All words attend to all words in previous layer; most arrows here are omitted

Where we left off

- **Self-attention**:
  - the basis of the method.
- **Position representations**:
  - Specify the sequence order, since self-attention is an unordered function of its inputs.
- **Nonlinearities**:
  - At the output of the self-attention block
  - Frequently implemented as a simple feed-forward network.
- **Masking**:
  - In order to parallelize operations while not looking at the future.
  - Keeps information about the future from “leaking” to the past.

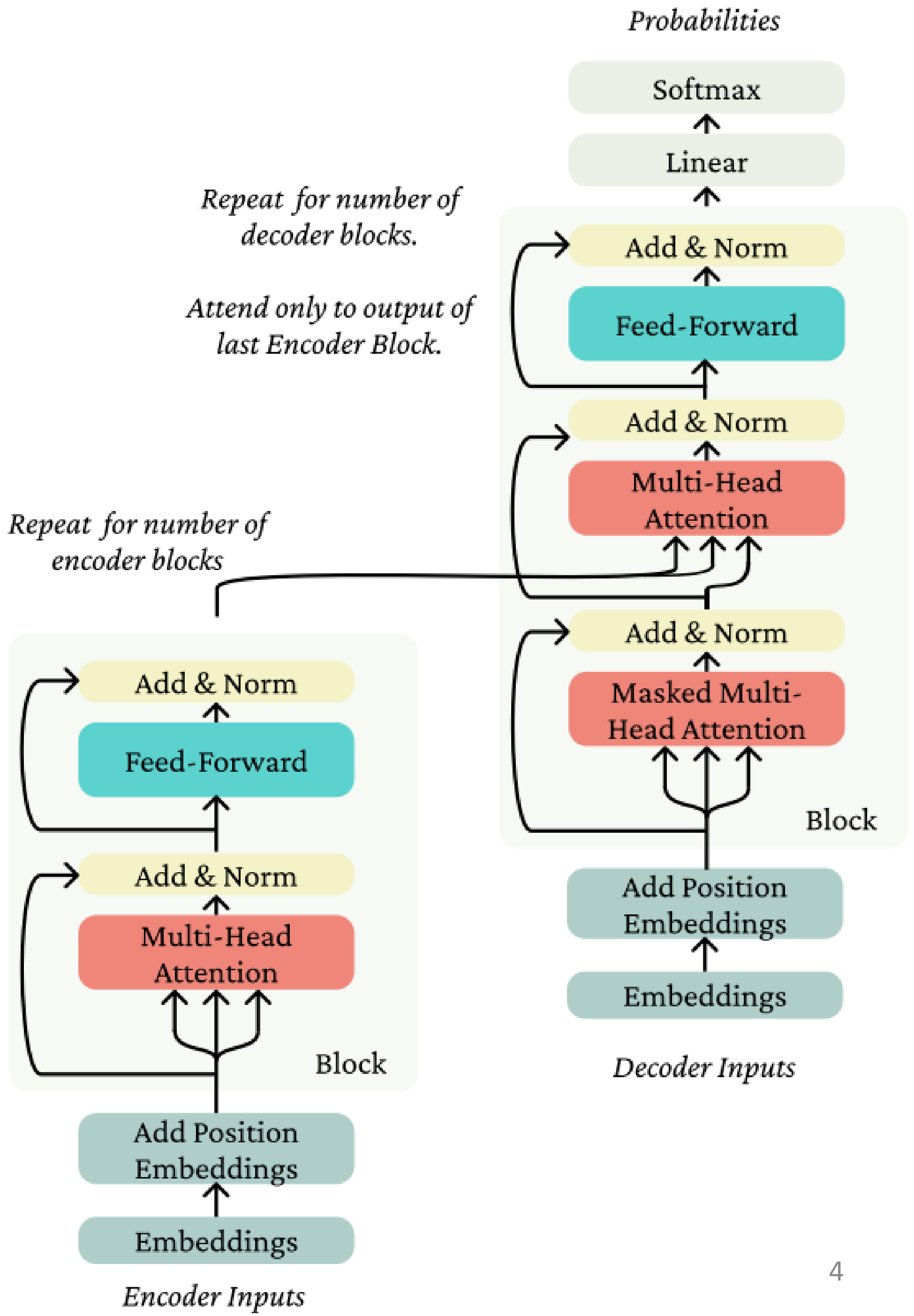


Transformer - Overview

Key points

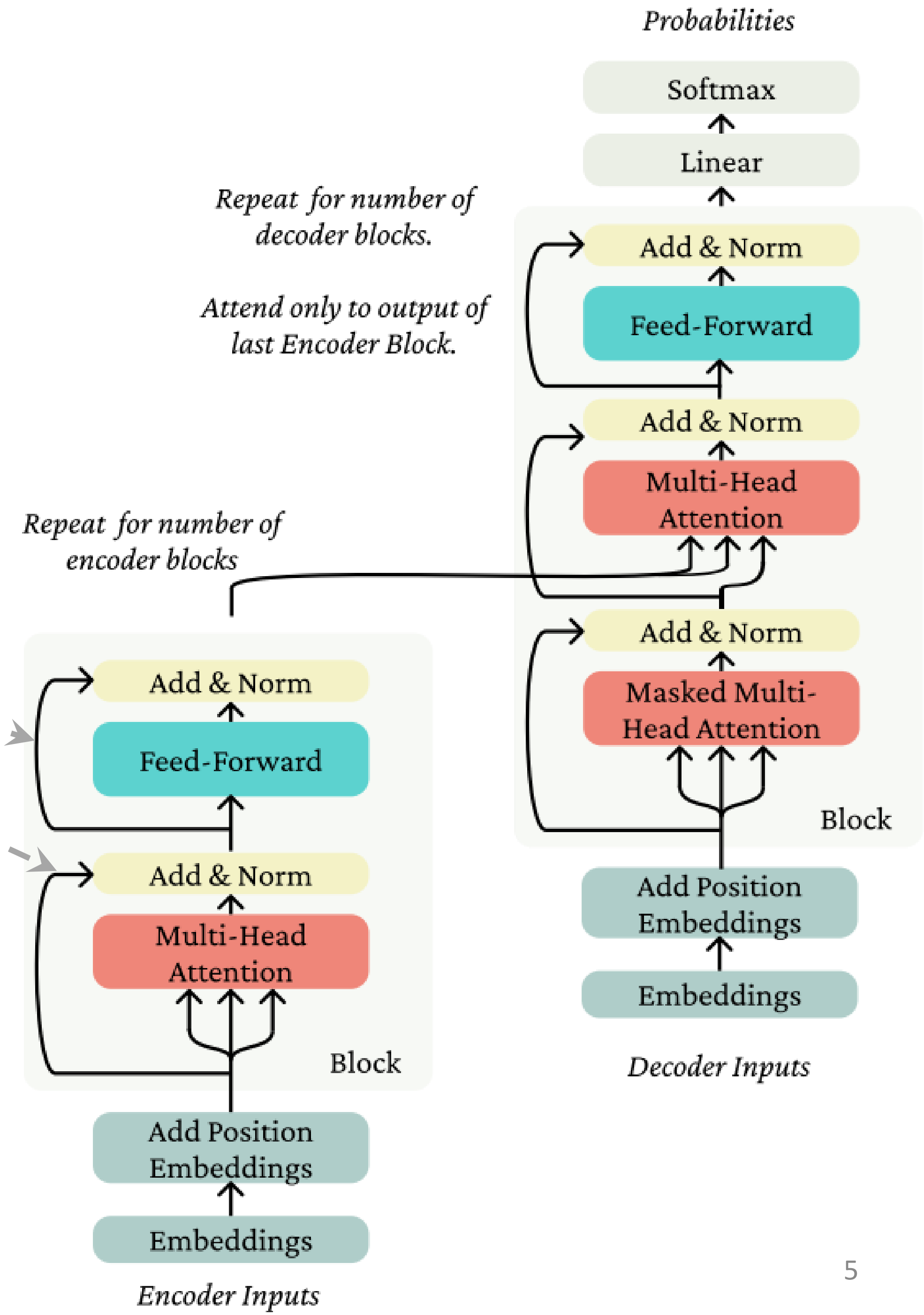
- Originally, an Encoder-Decoder architecture

- Self-Attention Block
  - Self-attention mechanism
  - Position embeddings
  - Nonlinearity in FF layers
  - Masking (optional)



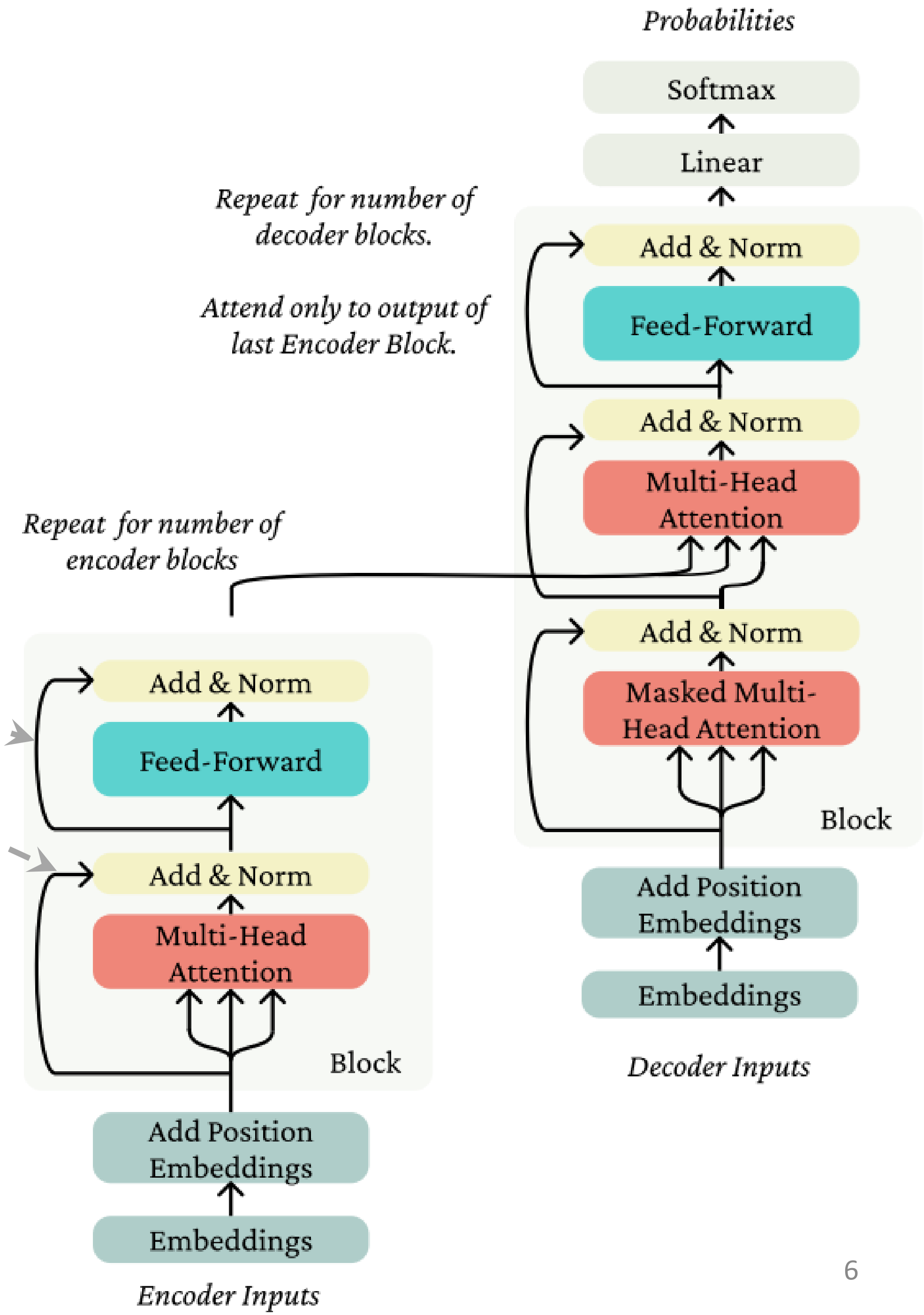
Transformer - Overview

- What is “Multi-Head Attention”?
- What is this?
- What is “Norm”



Transformer - Overview

- **What is "Multi-Head Attention"?**
- What is this?
- What is "Norm"

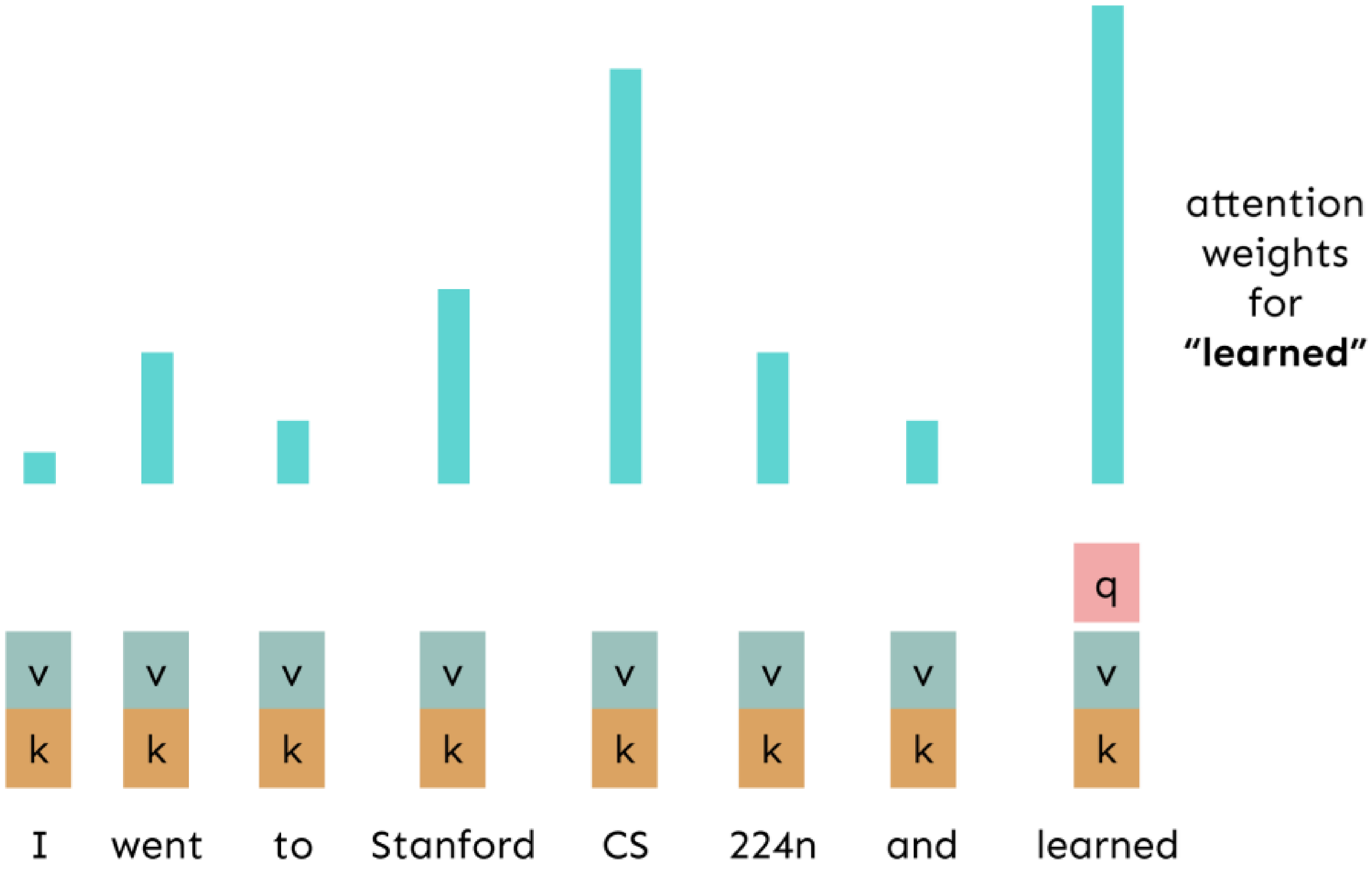


Multi-Head Self-Attention

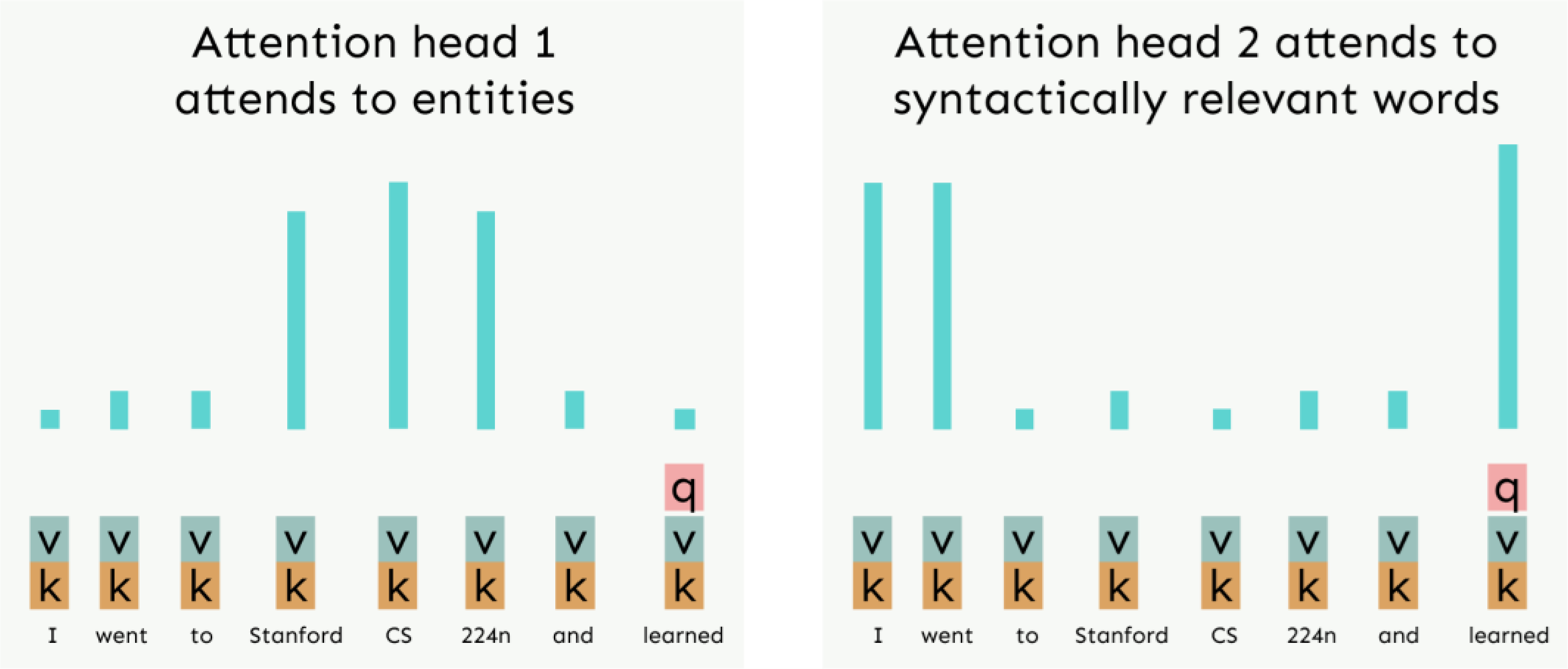
- Multi-Head Self-Attention
  - “Self-Attention” + “Multi-Head”

Multi-Head Self-Attention

Recall the Self-Attention Hypothetical Example



Multi-Head Self-Attention



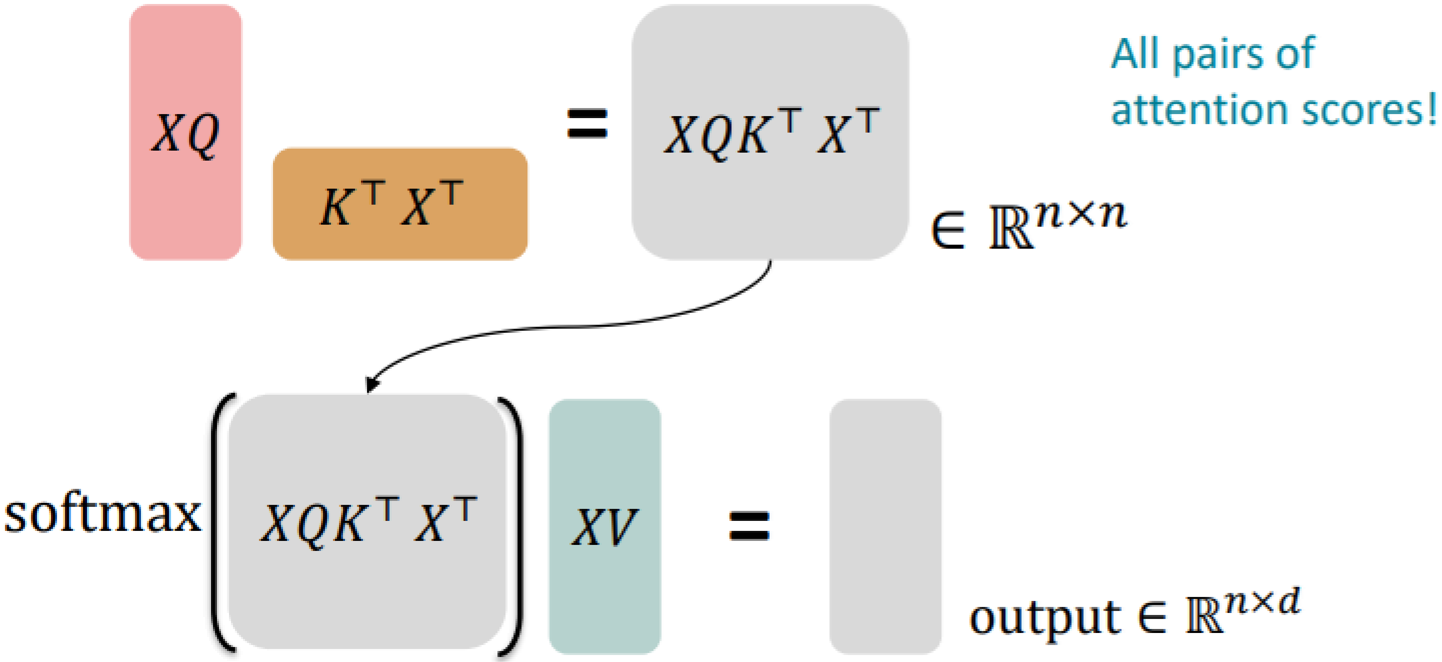
I went to Stanford CS 224n and learned

Single-Head Self-Attention

- Let's look at how key-query-value attention is computed, in matrices.
  - Let $X = [x_1; \ldots; x_n] \in \mathbb{R}^{n \times d}$ be the concatenation of input vectors.
  - First, note that $XK \in \mathbb{R}^{n \times d}$, $XQ \in \mathbb{R}^{n \times d}$, $XV \in \mathbb{R}^{n \times d}$.
  - The output is defined as $\text{output} = \text{softmax}(XQ(XK)^\top)XV \in\in \mathbb{R}^{n \times d}$.

First, take the query-key dot products in one matrix multiplication: $XQ(XK)^\top$

Next, softmax, and compute the weighted average with another matrix multiplication.



Multi-Head Self-Attention

- What if we want to look in multiple places in the sentence at once?
  - For word i, self-attention "looks" where $x_i^\top Q^\top K x_j$ is high, but maybe we want to focus on different j for different reasons?
- We'll define **multiple attention "heads"** through multiple Q,K,V matrices
- Let, $Q_\ell, K_\ell, V_\ell \in \mathbb{R}^{d \times \frac{d}{h}}$, where h is the number of attention heads, and ℓ ranges from 1 to h.
- Each attention head performs attention independently:
  - $\text{output}_\ell = \text{softmax}\left(XQ_\ell K_\ell^\top X^\top\right) * XV_\ell$, where $\text{output}_\ell \in \mathbb{R}^{d/h}$
- Then the outputs of all the heads are combined!
  - $\text{output} = [\text{output}_1; \ldots; \text{output}_h]Y$, where $Y \in \mathbb{R}^{d \times d}$

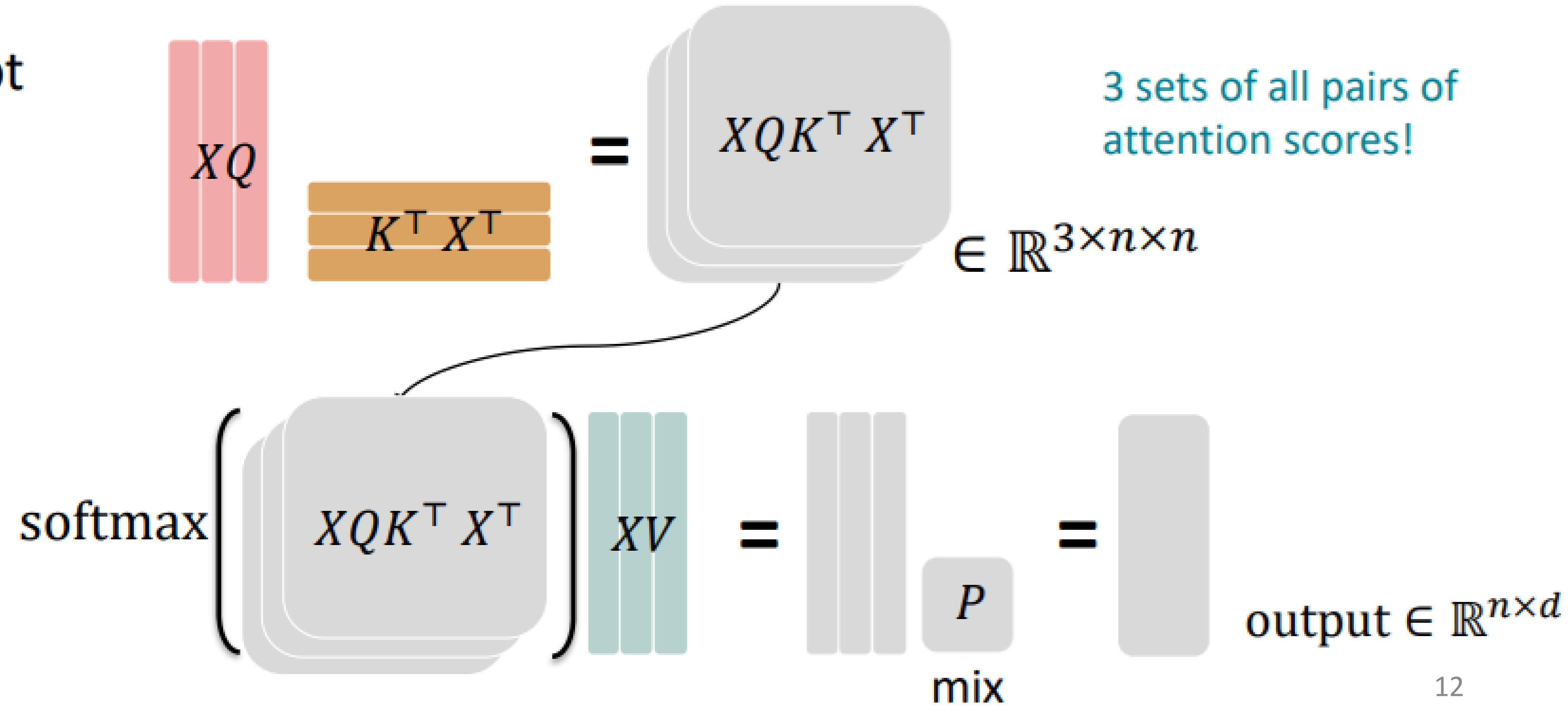
- Each head gets to "look" at different things, and construct value vectors differently.

Multi-Head Self-Attention

- Even though we compute h many attention heads, it's not really more costly.
  - We compute $XQ \in \mathbb{R}^{n\times d}$, and then reshape to $\mathbb{R}^{n\times h\times d/h}$. (Likewise for XK, XV.)
  - Then we transpose to $\mathbb{R}^{h\times n\times d/h}$; now the head axis is like a batch axis.
  - Almost everything else is identical, and the **matrices are the same sizes.**

First, take the query-key dot products in one matrix multiplication: $XQ(XK)^\top$

Next, softmax, and compute the weighted average with another matrix multiplication.



Scaled Dot Product

- **“Scaled Dot Product”** attention aids in training.
- When dimensionality d becomes large, dot products between vectors tend to become large.
  - Because of this, inputs to the softmax function can be large, making the gradients small.
- Instead of the self-attention function we’ve seen:

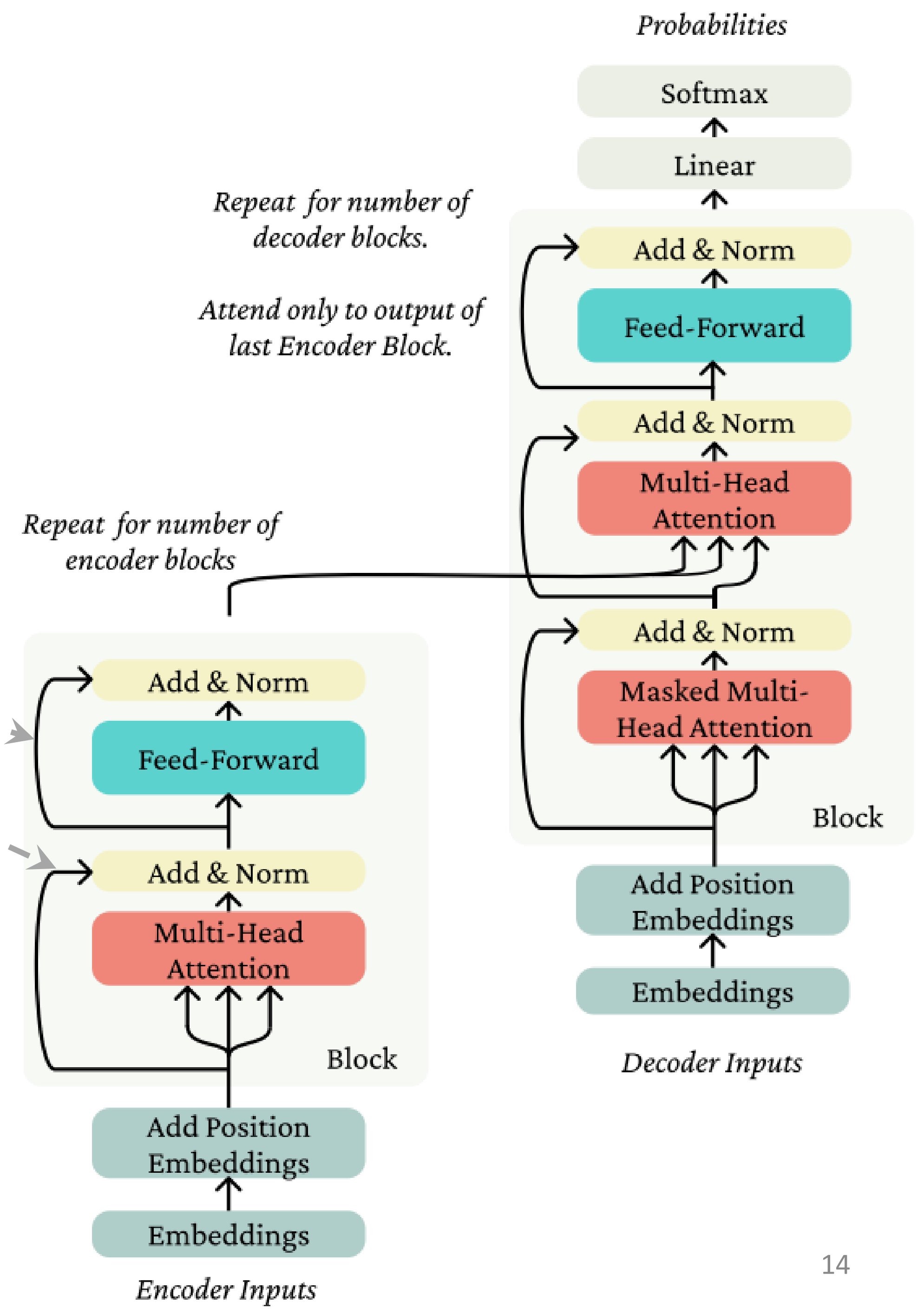
$$\mathrm{output}_\ell = \mathrm{softmax}\left(XQ_\ell K_\ell^\top X^\top\right) * XV_\ell$$

- We divide the attention scores by $\sqrt{d/h}$, to stop the scores from becoming large just as a function of d/h (The dimensionality divided by the number of heads.)

$$\mathrm{output}_\ell = \mathrm{softmax}\left(\frac{XQ_\ell K_\ell^\top X^\top}{\sqrt{d/h}}\right) * XV_\ell$$

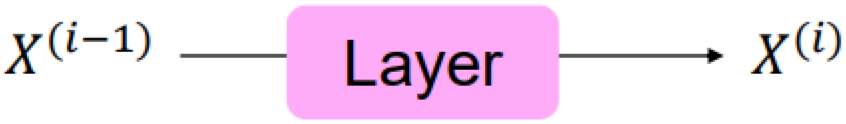
Transformer - Overview

- What is “Multi-Head Attention”?
- **What is this?**
- What is “Norm”



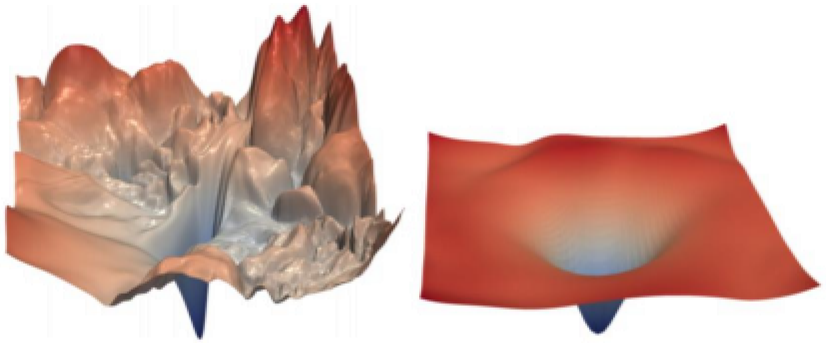
Residual Connection

- **Residual connections** are a trick to help models train better.
  - Instead of $X^{(i)} = \text{Layer}(X^{(i-1)})$ (where i represents the layer)



  - We let $X^{(i)} = X^{(i-1)} + \text{Layer}(X^{(i-1)})$ (so we only have to learn "the residual" from the previous layer)

  - Gradient is **great** through the residual connection; it's 1!
  - Bias towards the identity function!



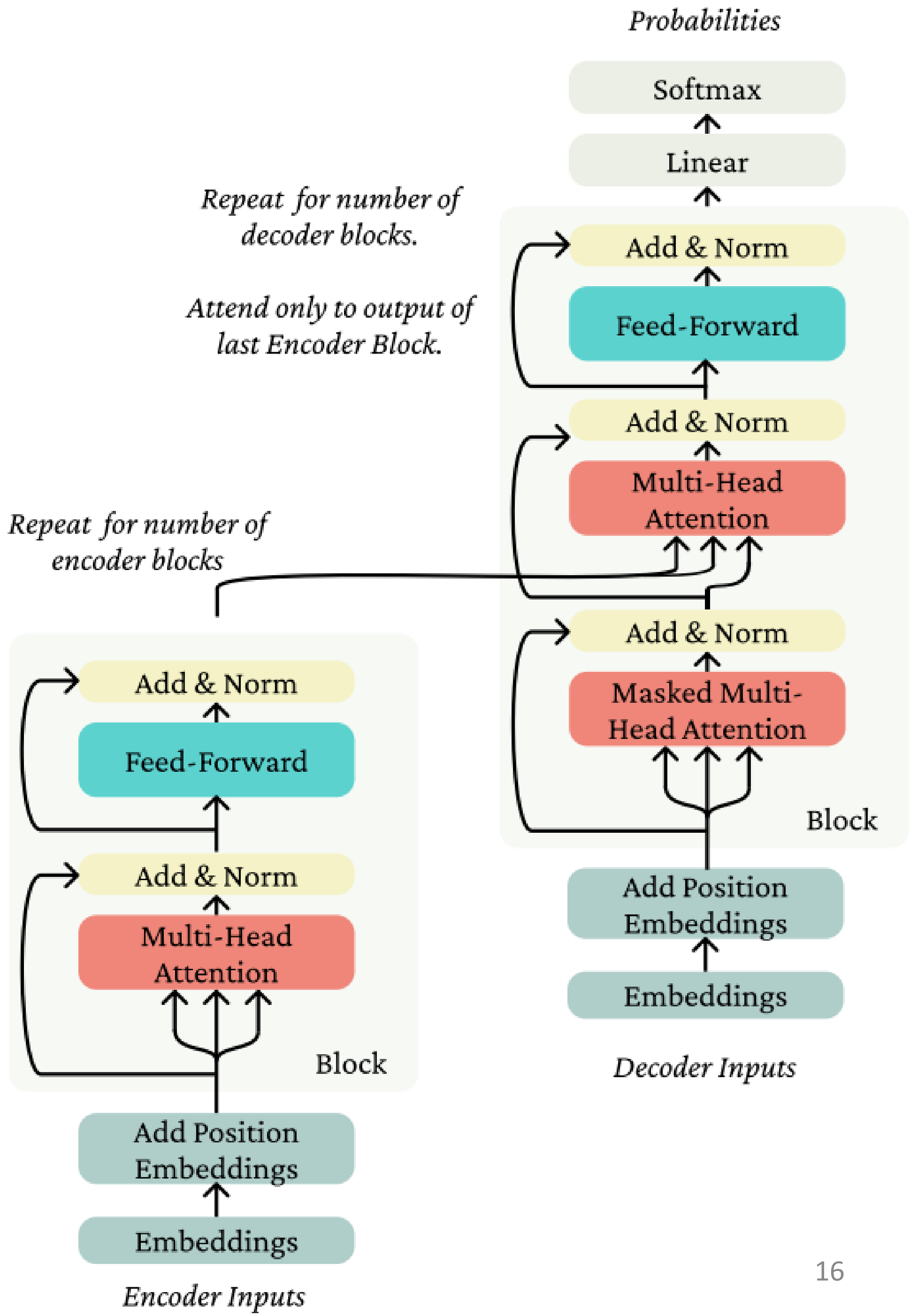
[no residuals] [residuals]

[Loss landscape visualization, Li et al., 2018, on a ResNet]

He, Kaiming, et al. "Deep residual learning for image recognition." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016.

Transformer - Overview

- What is "Multi-Head Attention"?
- What is this?
- **What is "Norm"**



Layer Normalization

- **Layer normalization** is a trick to help models train faster.
- Idea: cut down on uninformative variation in hidden vector values by normalizing to unit mean and standard deviation **within each layer**.
  - LayerNorm's success may be due to its normalizing gradients [Xu et al., 2019]
- Let $x \in \mathbb{R}^d$ be an individual (word) vector in the model.
- Let $\mu = \sum_{j=1}^{d} x_j$; this is the mean; $\mu \in \mathbb{R}$.
- Let $\sigma = \sqrt{\frac{1}{d} \sum_{j=1}^{d} (x_j - \mu)^2}$; this is the standard deviation; $\sigma \in \mathbb{R}$.
- Let $\gamma \in \mathbb{R}^d$ and $\beta \in \mathbb{R}^d$ be learned "gain" and "bias" parameters. (Can omit!)
- Then layer normalization computes:

$$\text{output} = \frac{x - \mu}{\sqrt{\sigma} + \epsilon} * \gamma + \beta$$

Normalize by scalar mean and variance

Modulate by learned elementwise gain and bias

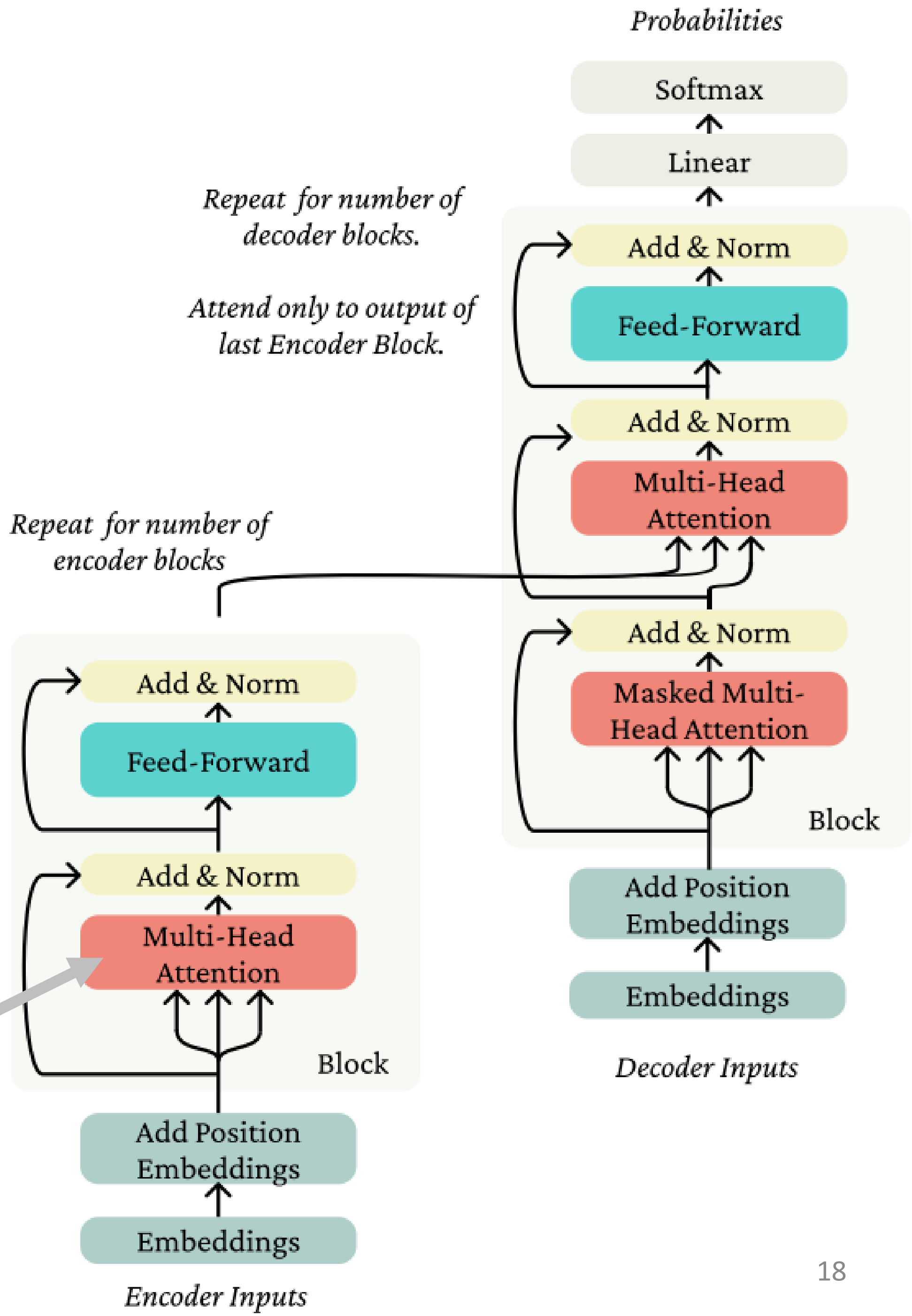
Transformer – Wrap Up

Key points

- Encoder - Decoder architecture

- But! the encoder and decoder can be used **separately**!

No Masking!

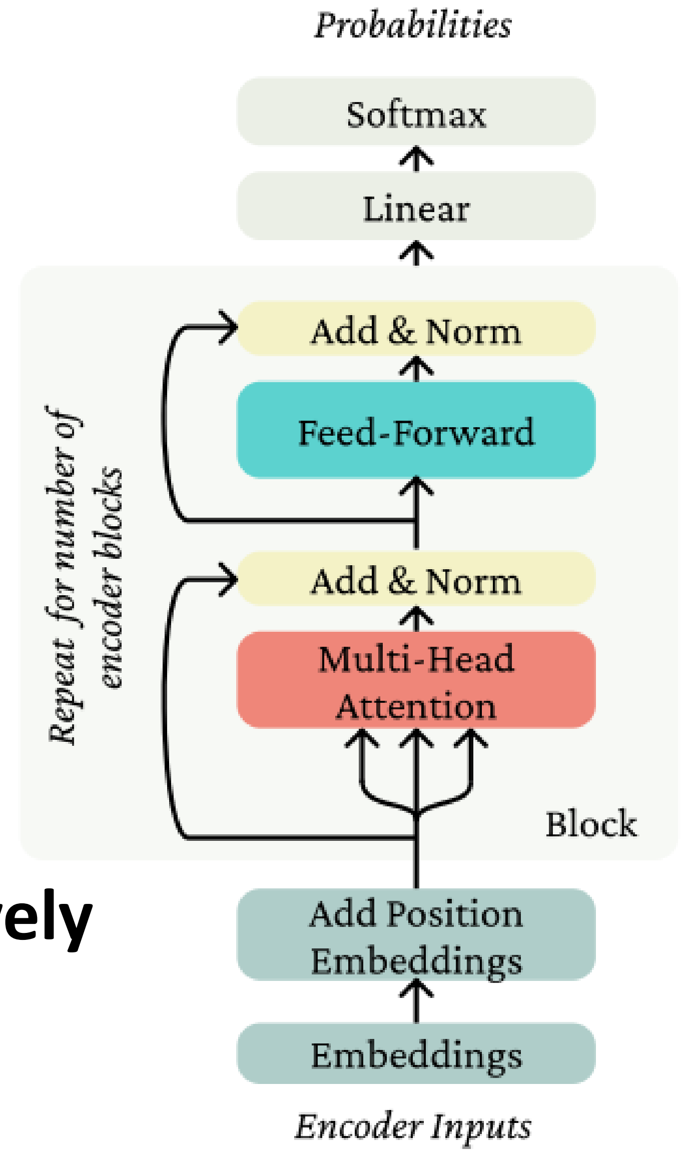


Transformer – Wrap Up

- Each Block consists of:
  - Self-attention
  - Add & Norm
  - Feed-Forward
  - Add & Norm

A Transformer Encoder is great when:

- You **aren't** trying to generate text **autoregressively** (there's no masking in the encoder so each position index can see the whole sequence,);
- Famous examples: BERT, RoBERTa.



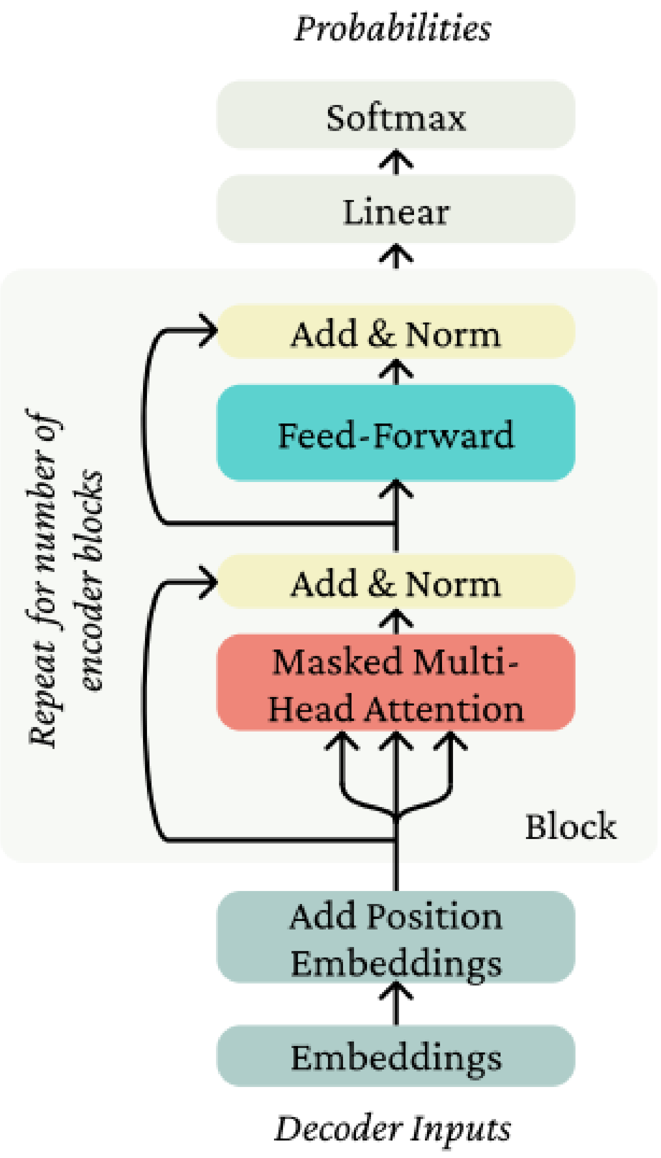
Transformer Encoder

Transformer – Wrap Up

Decoder is very similar to Encoder.

- To build an **autoregressive language model**, one uses a Transformer Decoder.
- Using future masking at each application of self-attention.

Famous example: GPT-2, GPT-3, BLOOM.



Transformer Decoder

Transformer – Wrap Up

When to use the whole Encoder - Decoder architecture?

- When we'd like bidirectional context on something (e.g., **article summarization**, **translation**)
- Such an architecture has been found to provide better performance than decoder-only models