CSCE 5218 & 4930
Deep Learning

Transformers

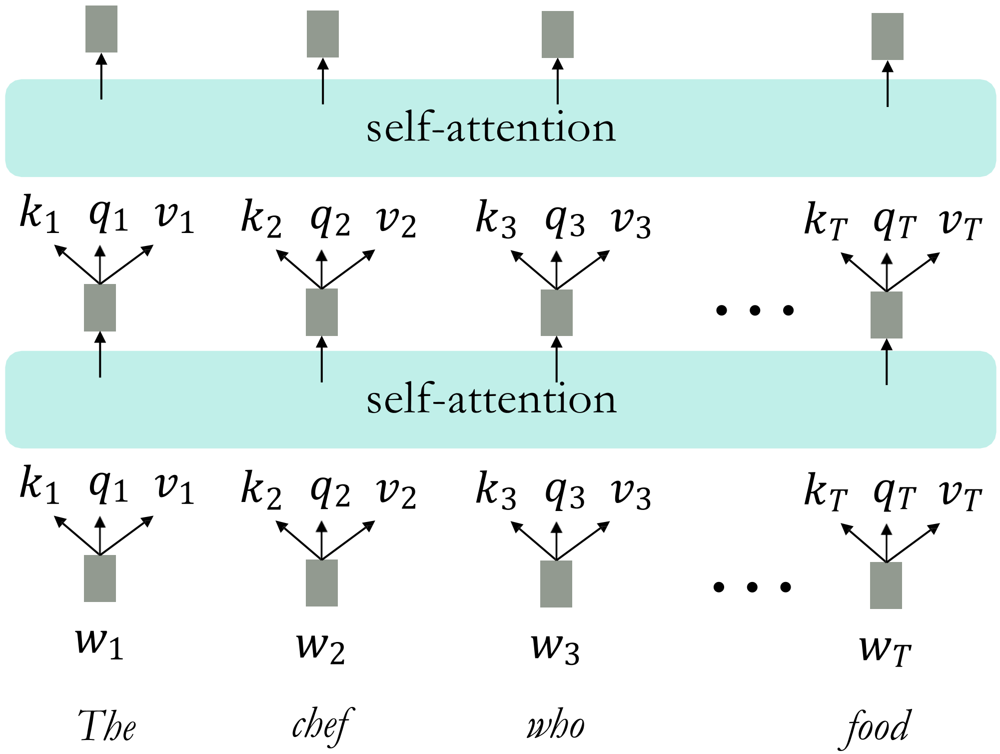
Slides adapted from Adriana Kovashka

Plan for this lecture

- Background
  - Context prediction, unsupervised learning
- Transformer models
  - Self-attention
  - Adapting self-attention for sequential data
  - The transformer architecture, encoder/decoder
- Transformers beyond language

Self-Attention

- In the diagram at the right, we have stacked self-attention blocks, like we might stack LSTM layers.
- Can self-attention be a drop-in replacement for recurrence?
- No. It has a few issues, which we'll go through.
- First, self-attention is an operation on **sets**. It has no inherent notion of order.



Self-attention doesn't know the order of its inputs.

Barriers and solutions for Self-Attention as a building block

Barriers

- Doesn't have an inherent notion of order!

Solutions

Fixing the first self-attention problem: Sequence order

- Since self-attention doesn't build in order information, we need to encode the order of the sentence in our keys, queries, and values.
- Consider representing each **sequence index** as a **vector**

$$p_i \in \mathbb{R}^d, \text{ for } i \in \{1,2,\ldots,T\} \text{ are position vectors}$$

- Don't worry about what the p_i are made of yet!
- Easy to incorporate this info into our self-attention block: just add the p_i to our inputs!
- Let v_i', k_i', q_i' be our old values, keys, and queries.

$$v_i = v_i' + p_i$$
$$q_i = q_i' + p_i$$
$$k_i = k_i' + p_i$$

In deep self-attention networks, we do this at the first layer! You could concatenate them as well, but people mostly just add…

Position representation vectors through sinusoids

- **Sinusoidal position representations:** concatenate sinusoidal functions of varying periods:

$$p_i = \begin{pmatrix} \sin(i/10000^{2*1/d}) \\ \cos(i/10000^{2*1/d}) \\ \vdots \\ \sin(i/10000^{2*\frac{d}{2}/d}) \\ \cos(i/10000^{2*\frac{d}{2}/d}) \end{pmatrix}$$

- Pros:
  - Periodicity indicates that maybe "absolute position" isn't as important
  - Maybe can extrapolate to longer sequences as periods restart!
- Cons:
  - Not learnable; also the extrapolation doesn't really work!

Image: https://timodenk.com/blog/linear-relationships-in-the-transformers-positional-encoding/

Barriers and solutions for Self-Attention as a building block

Barriers

- Doesn't have an inherent notion of order!
- No nonlinearities for deep learning! It's all just weighted averages

Solutions

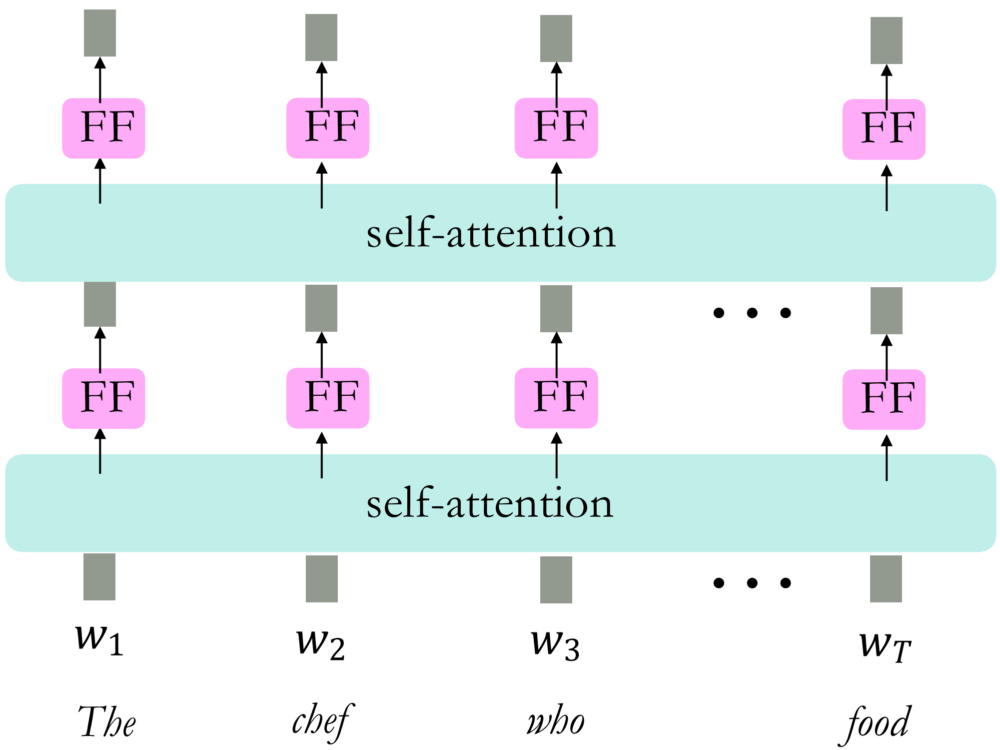
- Add position representations to the inputs

Adding nonlinearities in self-attention

- Note that there are no elementwise nonlinearities in self-attention; stacking more self-attention layers just re-averages **value** vectors

- Easy fix: add a **feed-forward network** to post-process each output vector.

$$
\begin{aligned}
m_i &= MLP(\text{output}_i) \\
&= W_2 * \text{ReLU}(W_1 \times \text{output}_i + b_1) + b_2
\end{aligned}
$$



Intuition: the FF network processes the result of attention

Barriers and solutions for Self-Attention as a building block

Barriers

- Doesn't have an inherent notion of order!
- No nonlinearities for deep learning magic! It's all just weighted averages
- Need to ensure we don't "look at the future" when predicting a sequence
  - Like in machine translation
  - Or language modeling

Solutions

- Add position representations to the inputs
- Easy fix: apply the same feedforward network to each self-attention output.

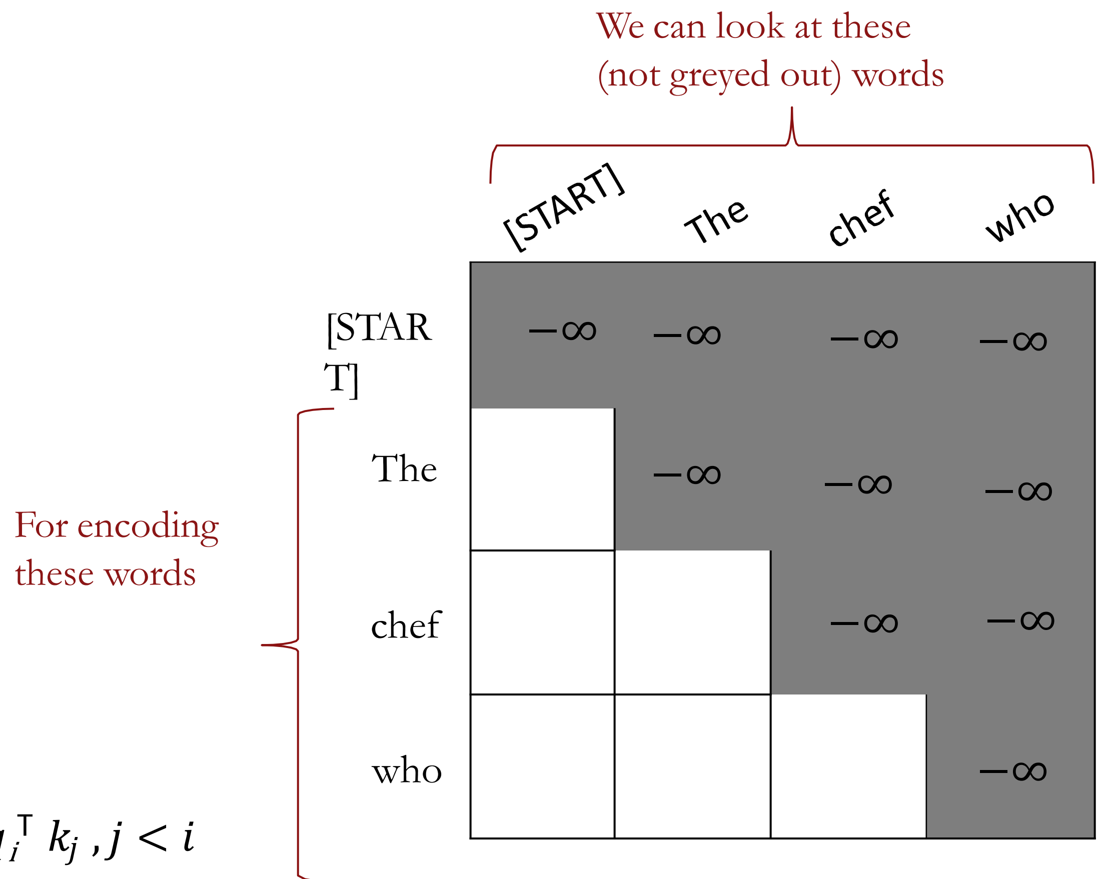
Masking the future in self-attention

- To use self-attention in **decoders**, we need to ensure we can't peek at the future.

- At every timestep, we could change the set of **keys and queries** to include only past words. (Inefficient!)

- To enable parallelization, we **mask out attention** to future words by setting attention scores to $-\infty$.

$$e_{ij} = \begin{cases} q_i^\top k_j, & j < i \\ -\infty, & j \geq i \end{cases}$$



Barriers and solutions for Self-Attention as a building block

Barriers

- Doesn't have an inherent notion of order!
- No nonlinearities for deep learning magic! It's all just weighted averages
- Need to ensure we don't "look at the future" when predicting a sequence
  - Like in machine translation
  - Or language modeling

- **Solutions**

- Add position representations to the inputs
- Easy fix: apply the same feedforward network to each self- attention output.
- Mask out the future by artificially setting attention weights to 0!

Necessities for a self-attention building block:

- **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.

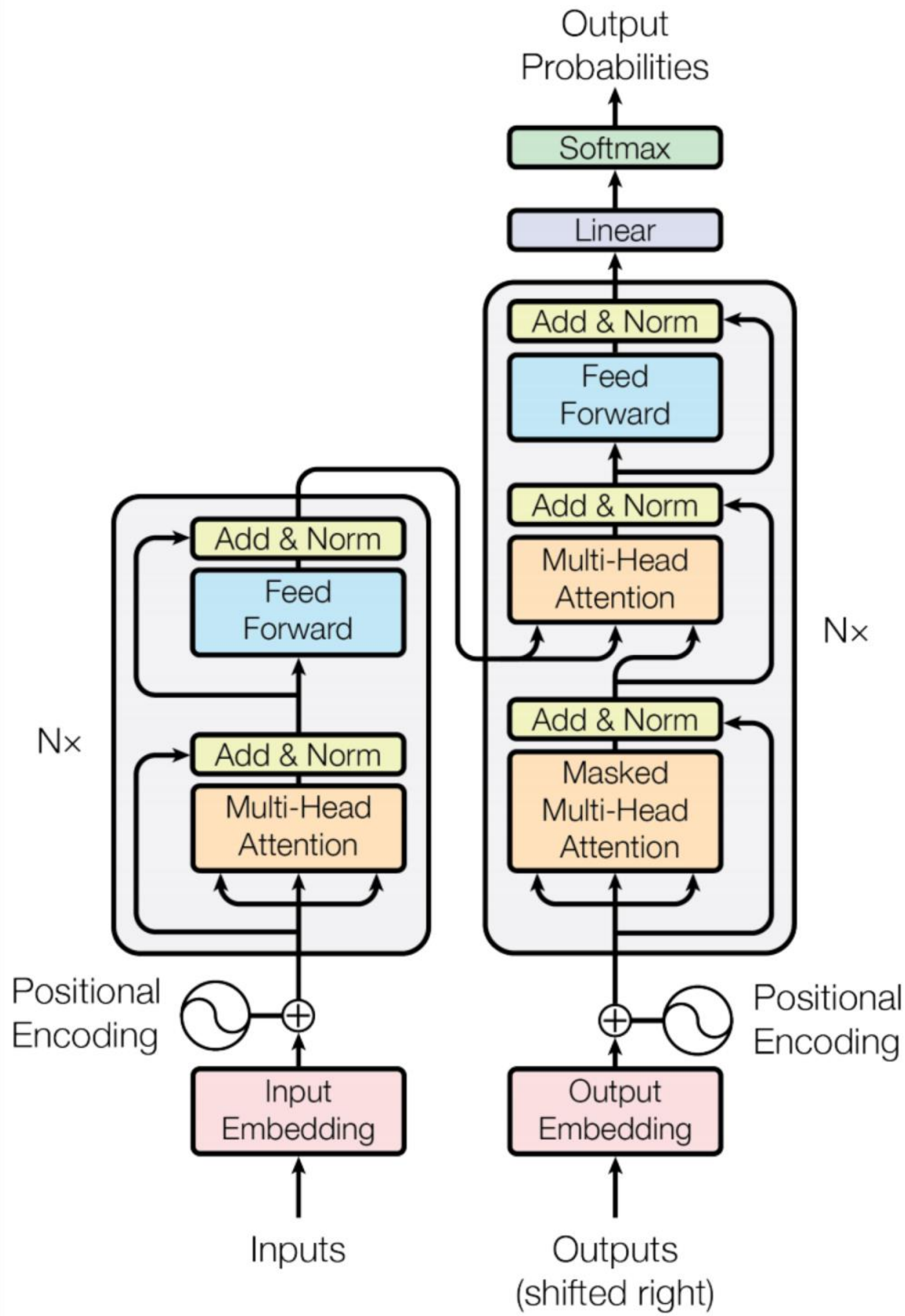
- That's it! But this is not the **Transformer** model we've been hearing about.

Transformer Overview

Attention is all you need. 2017. Aswani, Shazeer, Parmar, Uszkoreit, Jones, Gomez, Kaiser, Polosukhin
https://arxiv.org/pdf/1706.03762.pdf

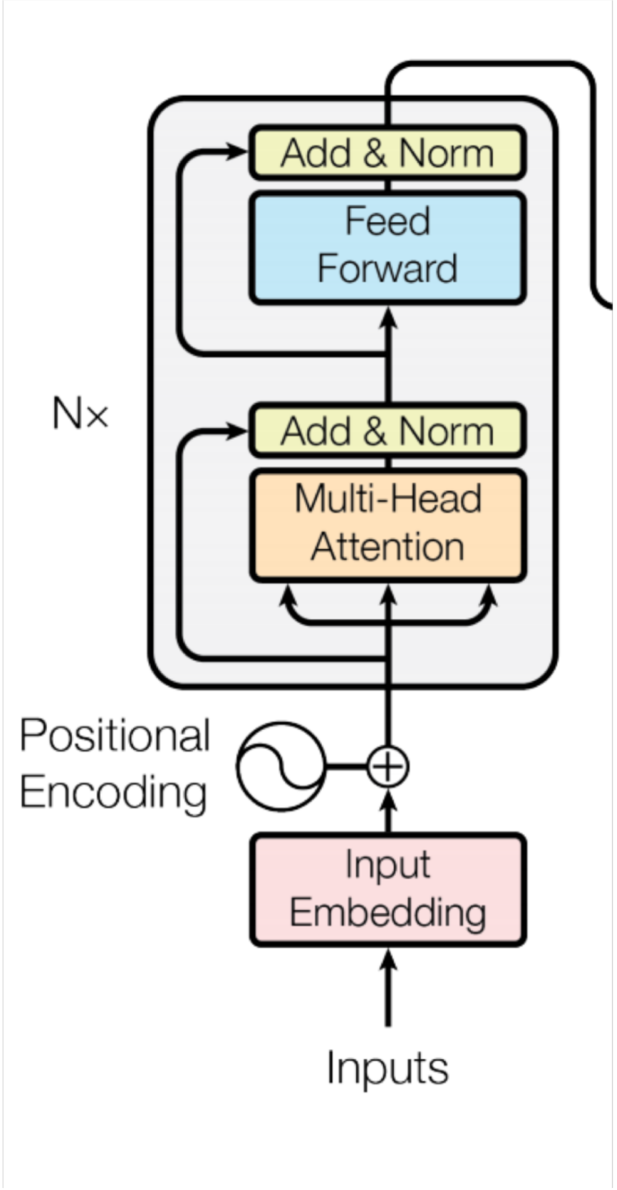
- Non-recurrent sequence-to- sequence encoder-decoder model
- Task: machine translation with parallel corpus
- Predict each translated word
- Final cost/error function is standard cross-entropy error on top of a softmax classifier

This and related figures from paper ⇑



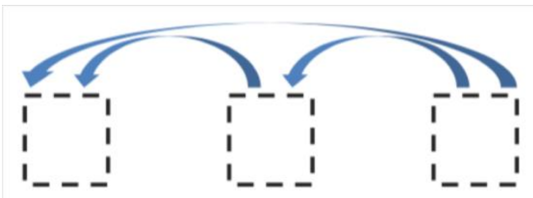
Transformer Encoder

- For encoder, at each block, we use the Q, K and V from the previous layer
- Blocks are repeated 6 times (in vertical stack)

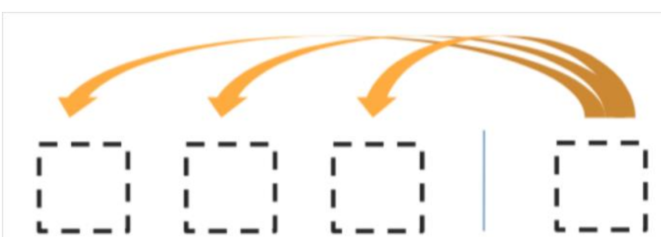


Transformer Decoder

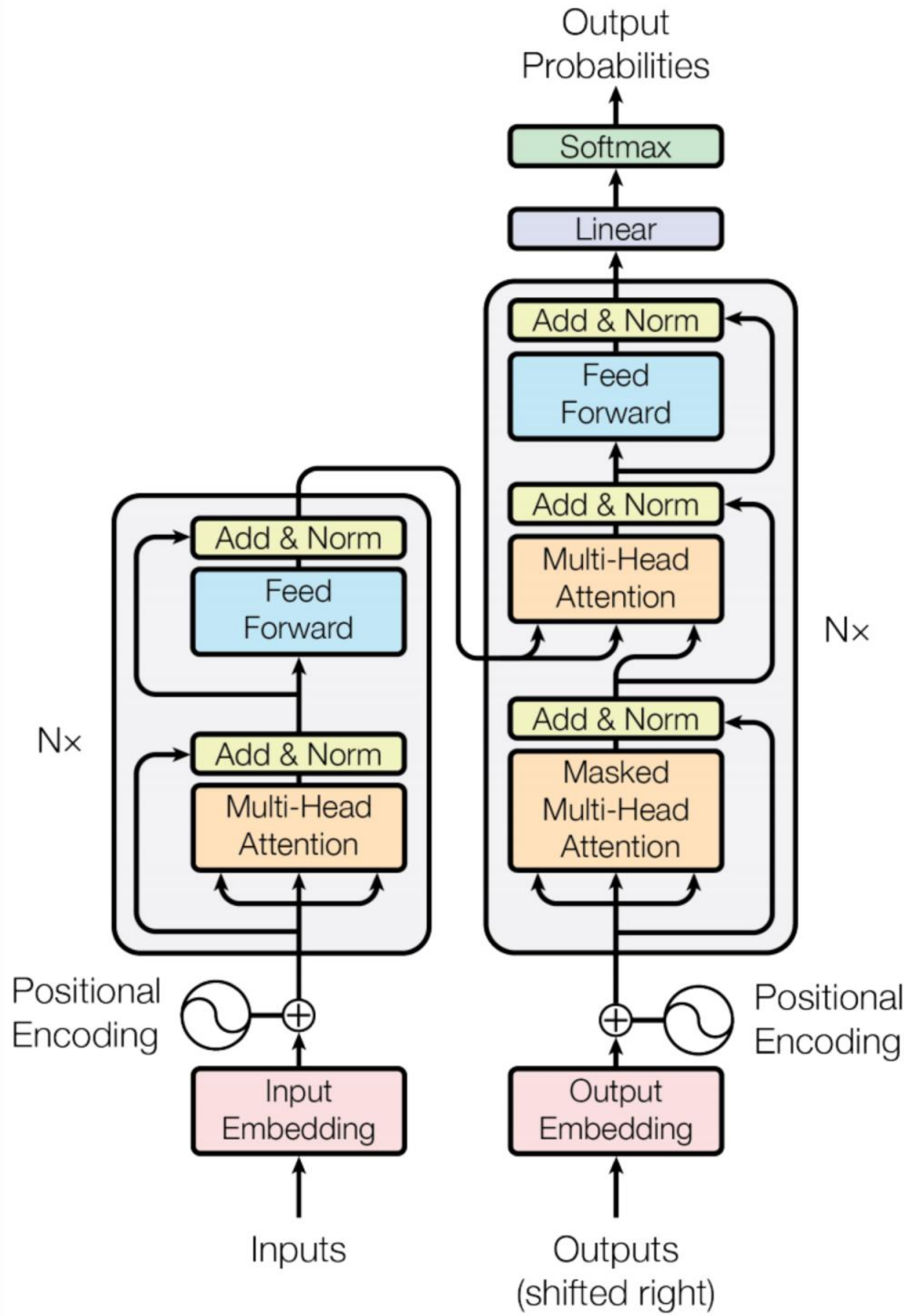
- 2 sublayer changes in decoder
- Masked decoder self-attention on previously generated outputs:



- Encoder-Decoder Attention, where queries come from previous decoder layer and keys and values come from output of encoder

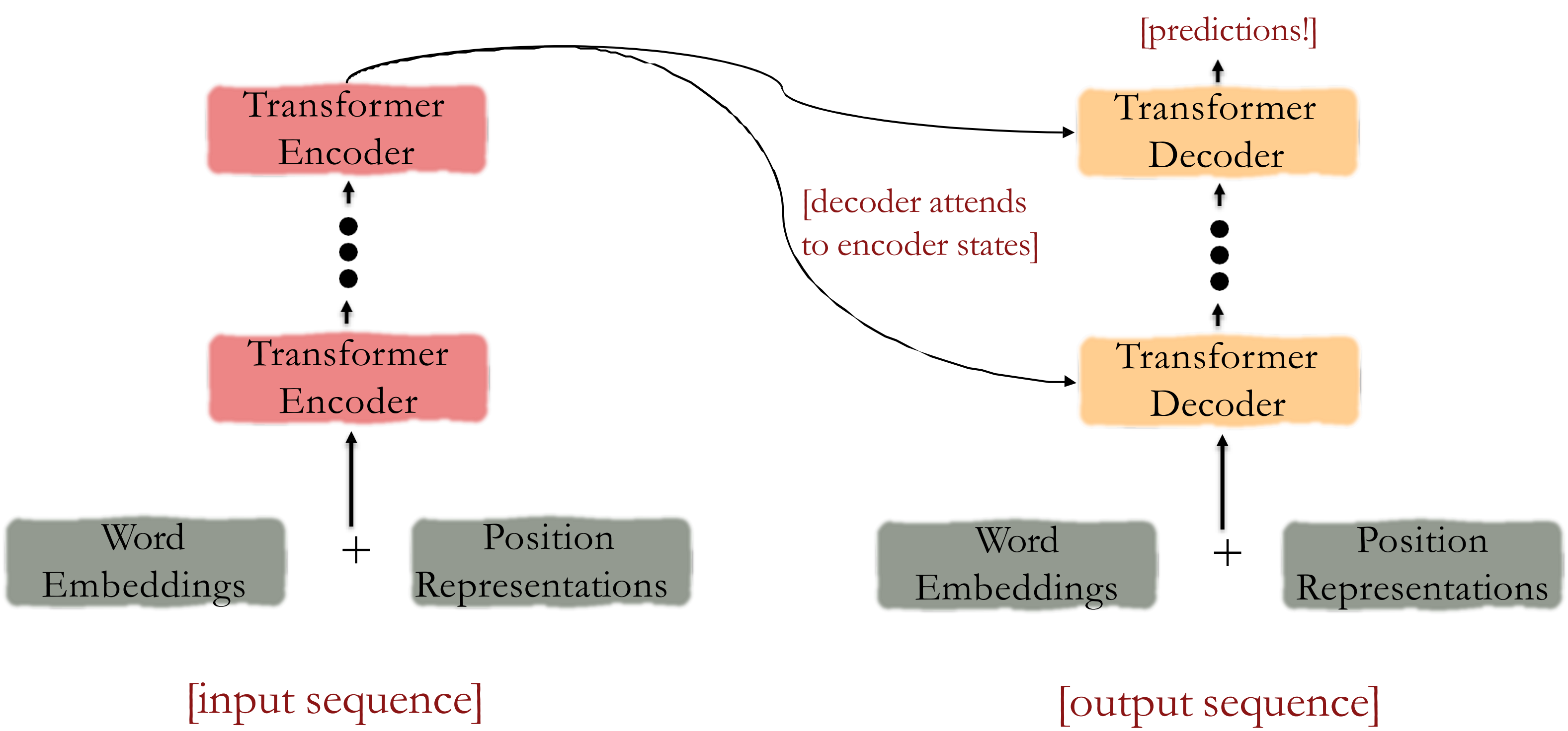


Blocks repeated 6 times also



The Transformer Encoder-Decoder [Vaswani et al., 2017]

Another look at the Transformer Encoder and Decoder Blocks at a high level



The Transformer Encoder-Decoder [Vaswani et al., 2017]

Next, let's look at the Transformer Encoder and Decoder Blocks

What's left in a Transformer Encoder Block that we haven't covered?

1. **Key-query-value attention:** How do we get the k, q, v vectors from a single word embedding?
2. **Multi-headed attention**: Attend to multiple places in a single layer!
3. **Tricks to help with training!**
    1. Residual connections
    2. Layer normalization
    3. Scaling the dot product
    4. These tricks **don't improve** what the model is able to do; they help improve the training process

The Transformer Encoder:
Dot-Product Attention

- Inputs: a query q and a set of key-value (k-v) pairs to an output
- Query, keys, values, and output are all vectors

- Output is weighted sum of values, where
- Weight of each value is computed by an inner product of query and corresponding key
- Queries and keys have same dimensionality d_k, value have d_v

$$A(q, K, V) = \sum_i \frac{e^{q \cdot k_i}}{\sum_j e^{q \cdot k_j}} v_i$$

The Transformer Encoder:
Dot-Product Attention – Matrix notation

- When we have multiple queries q, we stack them in a matrix Q:

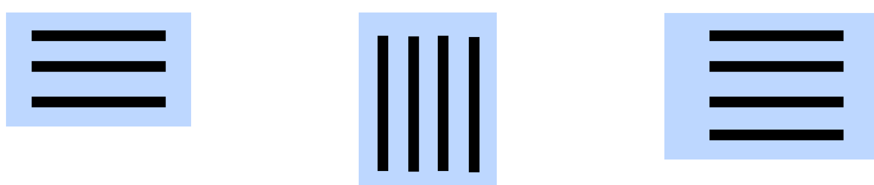
$$A(q, K, V) = \sum_i \frac{e^{q \cdot k_i}}{\sum_j e^{q \cdot k_j}} v_i$$

- Becomes:

$$A(Q, K, V) = softmax(QK^T)V$$

$[\,|Q|\ \mathrm{x}\ d_k]\ \mathrm{x}\ [d_k\ \mathrm{x}\ |K|\,]\ \mathrm{x}\ [\,|K|\ \mathrm{x}\ d_v]$

softmax
row-wise



$= [\,|Q|\ \mathrm{x}\ d_v]$

The Transformer Encoder: **Key-Query-Value Attention**

- We saw that self-attention is when keys, queries, and values come from the same source. The Transformer does this in a particular way:
  - Let $x_1, \ldots, x_T$ be input vectors to the Transformer encoder; $x_i \in \mathbb{R}^d$
- Then keys, queries, values are:
  - $k_i = Kx_i$, where $K \in \mathbb{R}^{d \times d}$ is the key matrix.
  - $q_i = Qx_i$, where $\mathrm{Q} \in \mathbb{R}^{d \times d}$ is the query matrix.
  - $v_i = Vx_i$, where $\mathrm{V} \in \mathbb{R}^{d \times d}$ is the value matrix.
- These matrices allow *different aspects* of the x vectors to be used/emphasized in each of the three roles.

The Transformer Encoder: **Key-Query-Value Attention**

- Let's look at how key-query-value attention is computed, in matrices.
  - Let $X = [x_1; \dots; x_T] \in \mathbb{R}^{T\times d}$ be the concatenation of input vectors.
  - First, note that $XK \in \mathbb{R}^{T\times d}$, $XQ \in \mathbb{R}^{T\times d}$, $XV \in \mathbb{R}^{T\times d}$.
  - The output is defined as **output** $=$ softmax$(XQ(XK)^\top) \times XV$.

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

$$XQ \quad K^\top X^\top = XQK^\top X^\top \in \mathbb{R}^{T\times T}$$

All pairs of attention scores!

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

$$\mathrm{softmax}\left(XQK^\top X^\top\right) \; XV = \quad \text{output} \in \mathbb{R}^{T\times d}$$

The Transformer Encoder: **Multi-headed attention**

- What if we want to look in multiple places in the sentence at once?
  - For word i, self-attention "looks" where $x^\top Q^\top K x_j$ 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, $Q_P, K_P, V_P \in \mathbb{R}^{d \times \frac{d}{h}}$, where h is the number of attention heads, and P ranges from 1 to h.
- Each attention head performs attention independently:
  - $\text{output}_P = \text{softmax}\left(XQ_P K_P^\top X^\top\right) * XV_P$, where $\text{output}_P \in \mathbb{R}^{d/h}$
- Then the outputs of all the heads are combined!
  - $\text{output} = Y[\text{output}_1; \ldots; \text{output}_h]$, where $Y \in \mathbb{R}^{d \times d}$

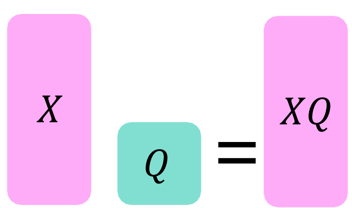
- Each head gets to "look" at different things, and construct value vectors differently.

The Transformer Encoder: **Multi-headed 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, $Q_P, K_P, V_P \in \mathbb{R}^{d \times \frac{d}{h}}$, where h is the number of attention heads, and P ranges from 1 to h.

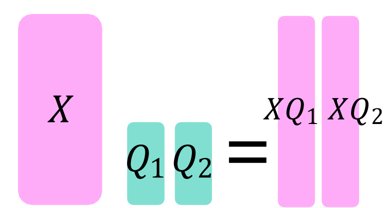
Single-head attention

(just the query matrix)



Multi-head attention

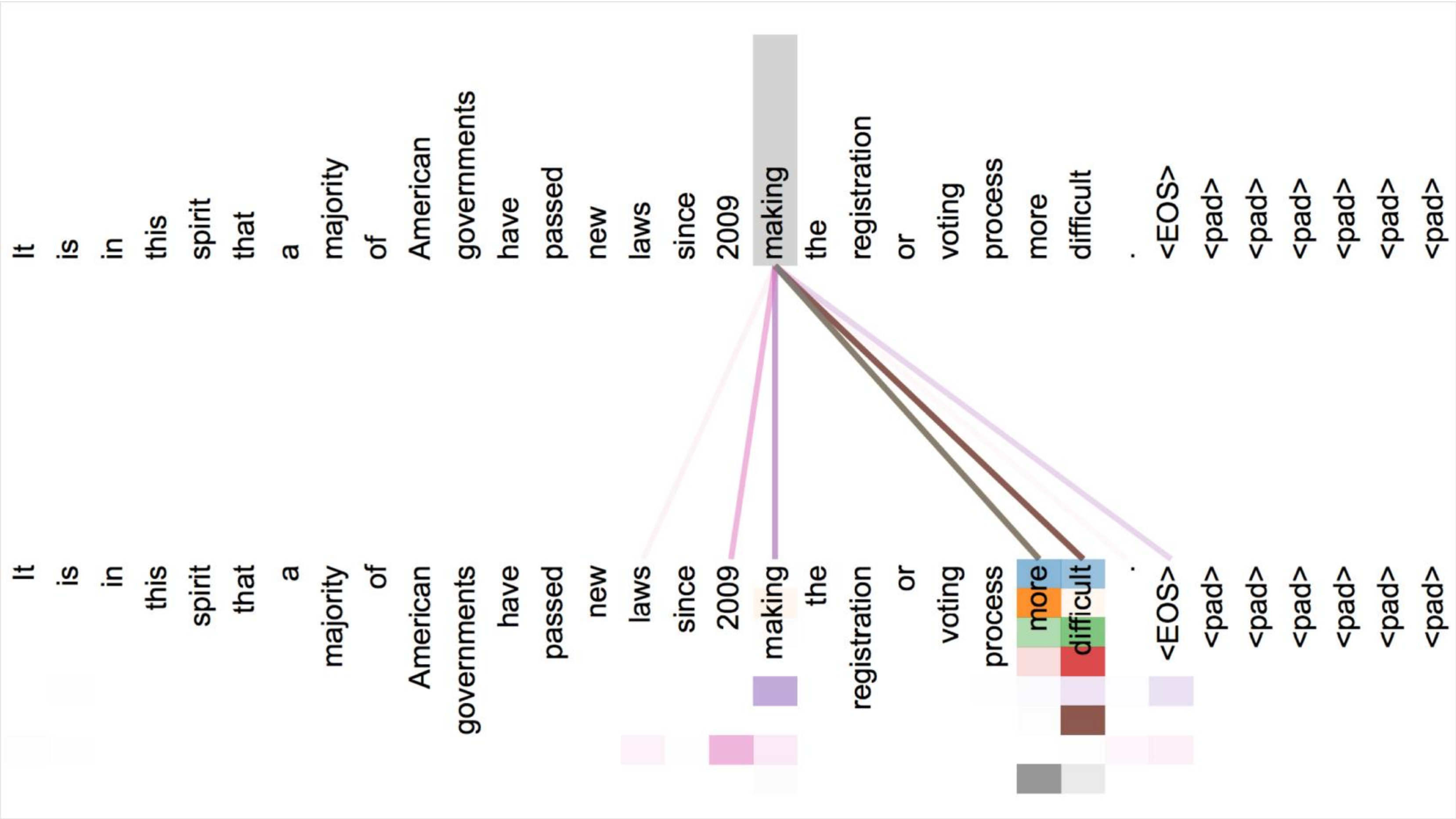
(just two heads here)



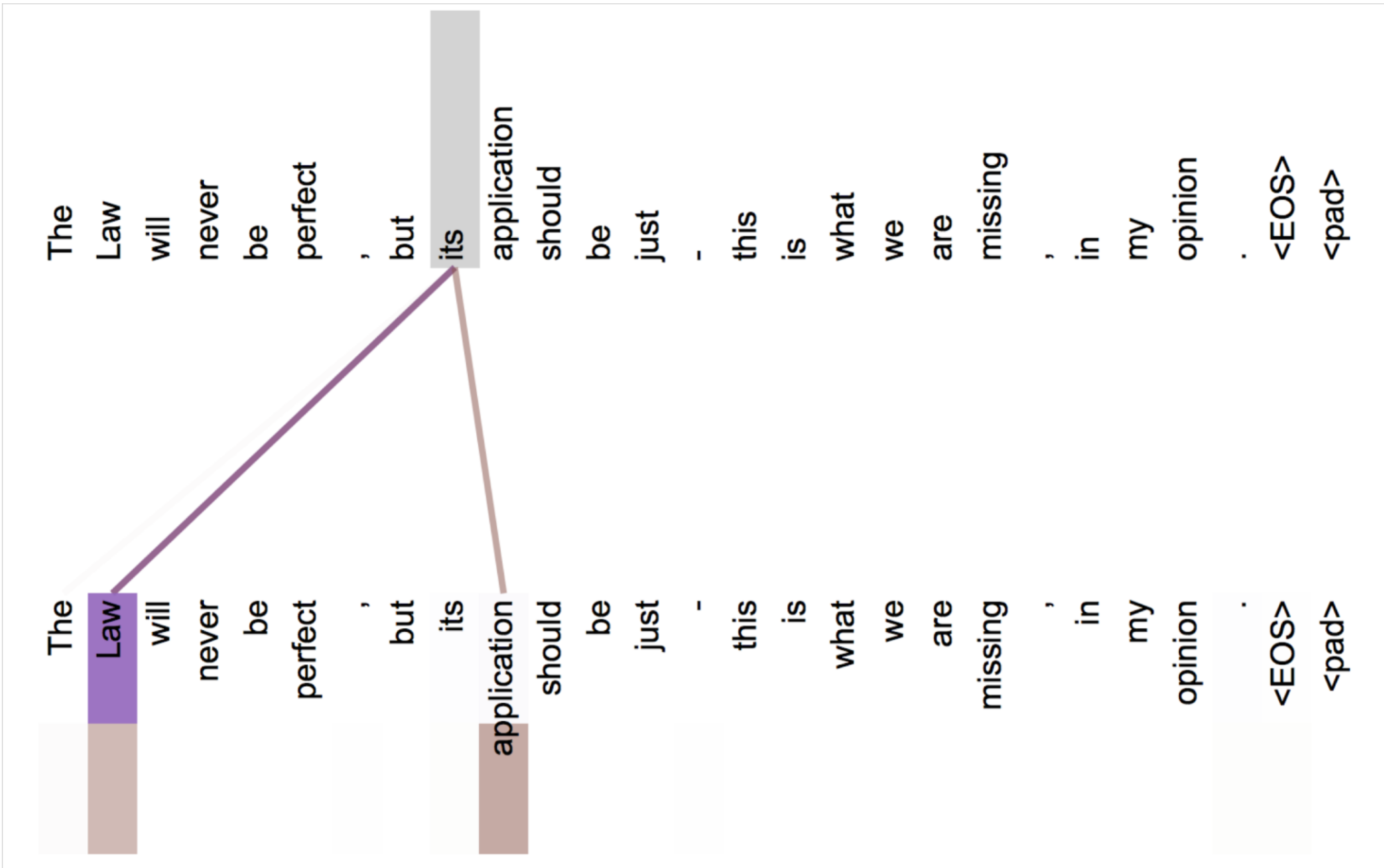
Same amount of computation as single-head self-attention!

Attention visualization in layer 5

- Words start to pay attention to other words in sensible ways



Attention visualization: Implicit anaphora resolution

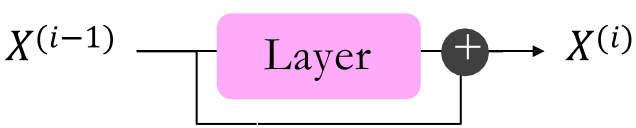


In 5th layer. Isolated attentions from just the word 'its' for attention heads 5 and 6. Note that the attentions are very sharp for this word.

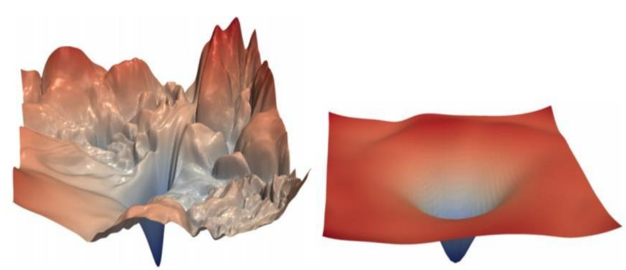
The Transformer Encoder:
Residual connections [He et al., 2016]

- **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)



  - Residual connections are thought to make the loss landscape considerably smoother (thus easier training!)



[no residuals] [residuals]

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

The Transformer Encoder: **Layer normalization** [Ba et al., 2016]

- **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 $\mu = \Sigma_{j=1}^{d} x_j$; this is the mean; $\mu \in \mathbb{R}$.
  - Let $\sigma = \sqrt{\frac{1}{d}\sum_{j=1}^{d}(x - \mu)^2}$; 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

The Transformer Encoder: **Scaled Dot Product** [Vaswani et al., 2017]

- **“Scaled Dot Product”** attention is a final variation to aid in Transformer training.
- When dimensionality d becomes large, dot products between vectors become large, inputs to the softmax function can be large, making gradients small.

- Instead of the self-attention function we’ve seen:

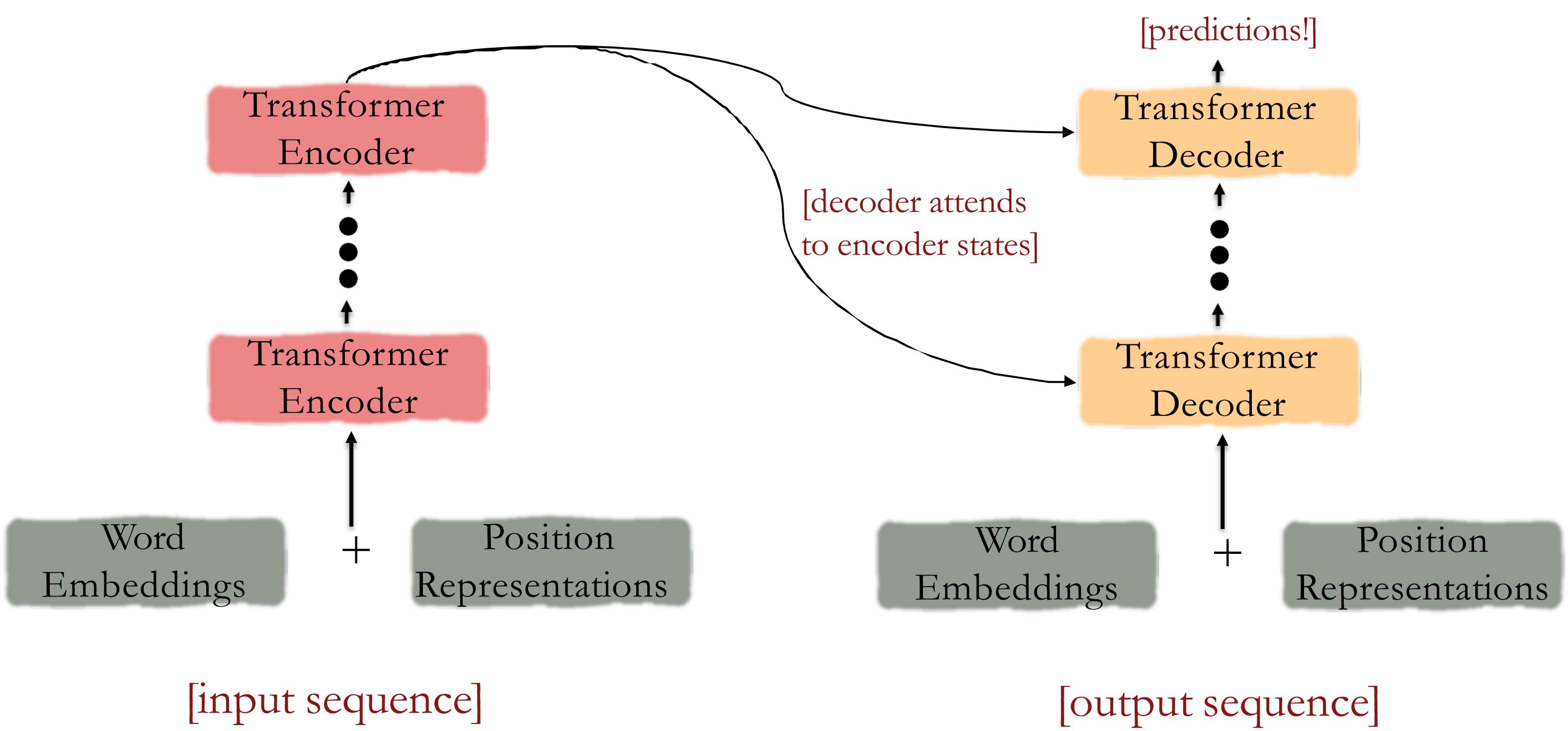
$$\text{output}_P = \text{softmax}\left(XQ_P K_P^\top X^\top\right) * XV_P$$

- 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.)

$$\text{output}_P = \text{softmax}\left(\frac{XQ_P K_P^\top X^\top}{\sqrt{d/h}}\right) * XV_P$$

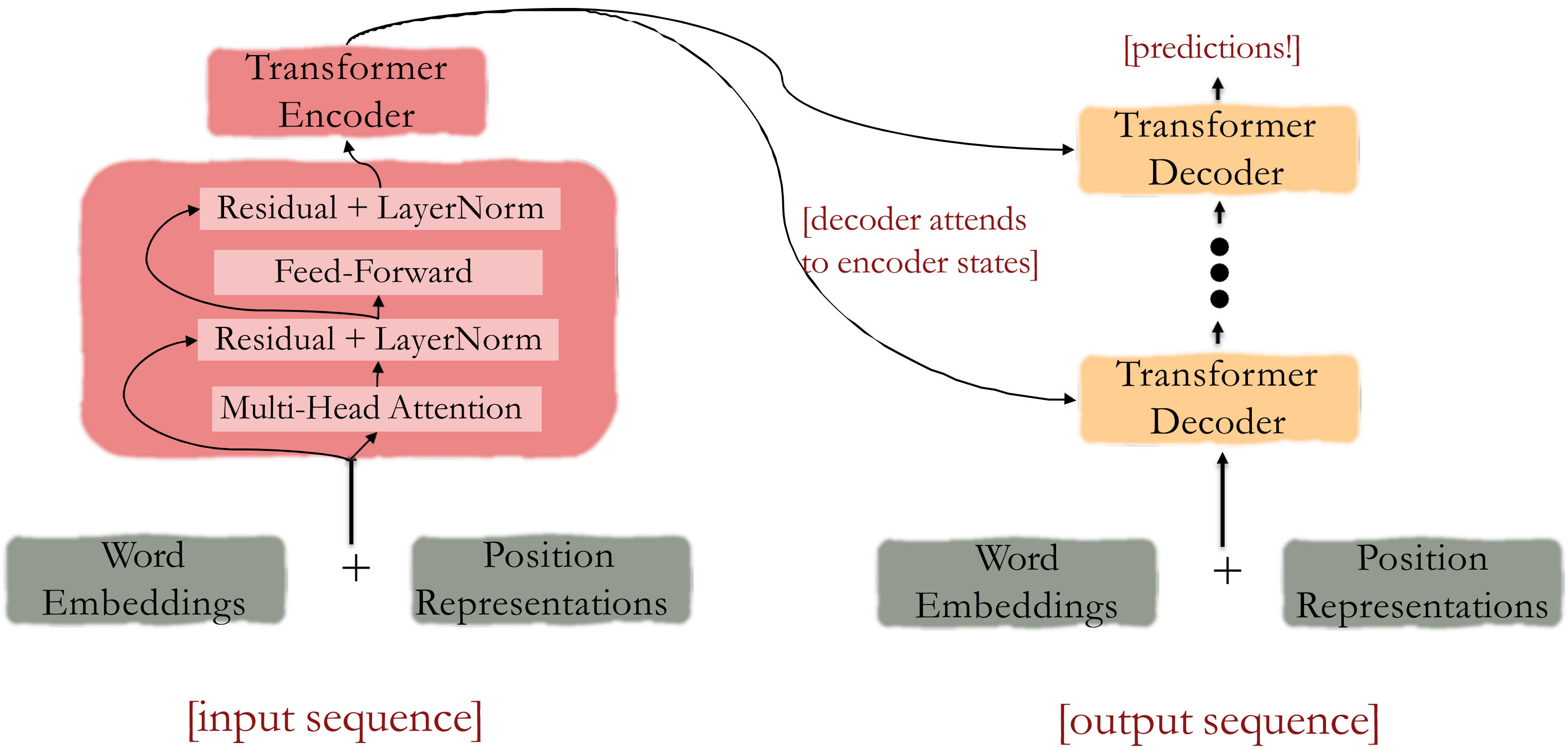
The Transformer Encoder-Decoder
[Vaswani et al., 2017]

Looking back at the whole model, zooming in on an Encoder block:



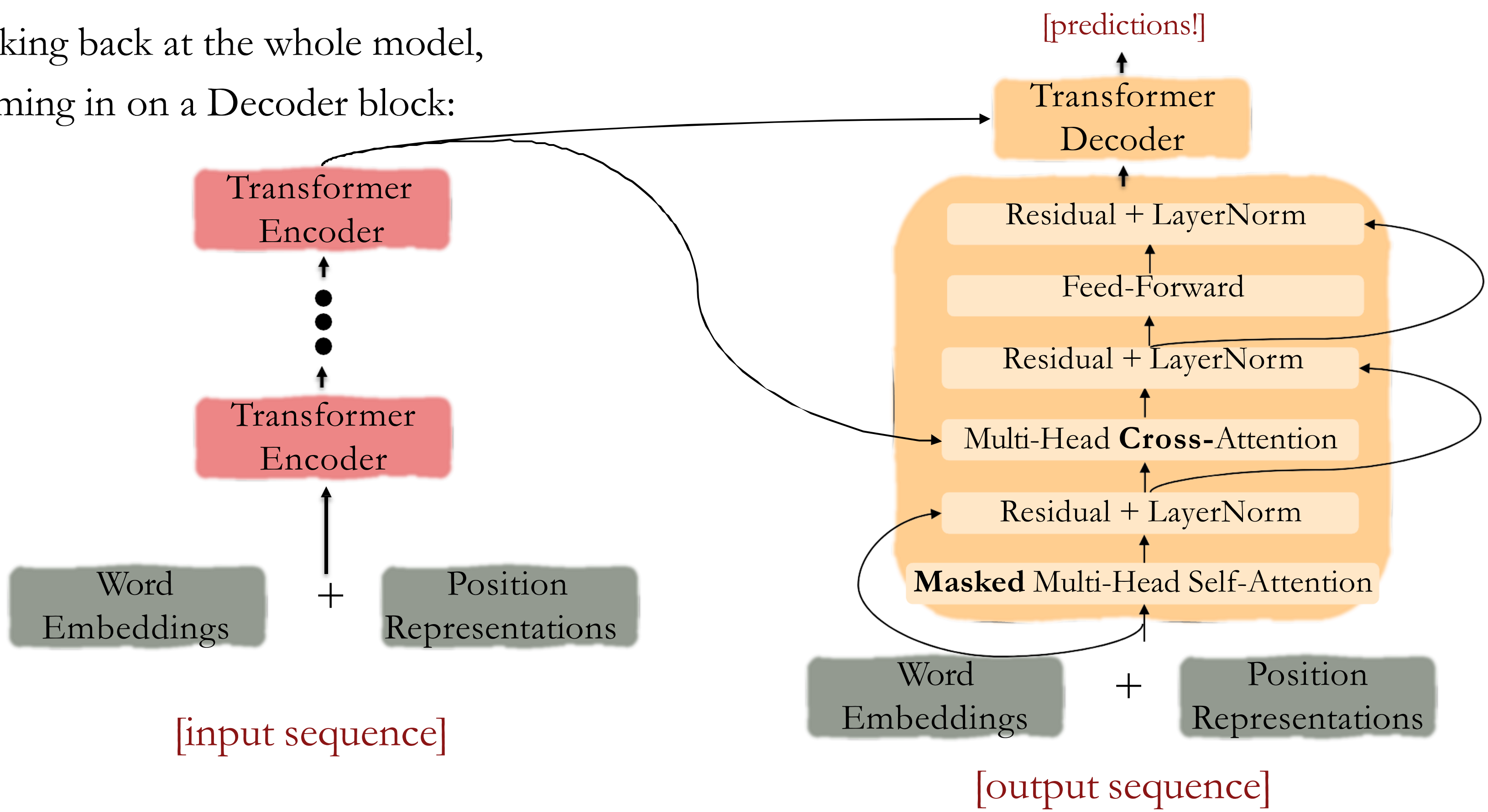
The Transformer Encoder-Decoder [Vaswani et al., 2017]

Looking back at the whole model, zooming in on an Encoder block:



The Transformer Encoder-Decoder [Vaswani et al., 2017]

Looking back at the whole model, zooming in on a Decoder block:



The Transformer Decoder: **Cross-attention (details)**

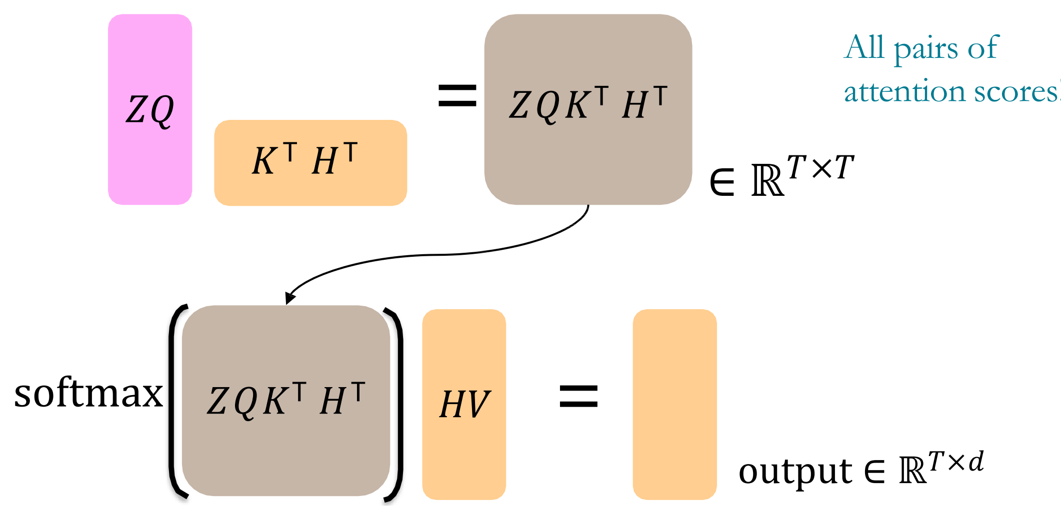
- We saw self-attention is when keys, queries, and values come from the same source.
- In the decoder, we have attention that looks more like what we saw last week.
- Let $h_1, \dots, h_T$ be **output** vectors from the Transformer **encoder**; $x_i \in \mathbb{R}^d$
- Let $z_1, \dots, z_T$ be input vectors from the Transformer **decoder**, $z_i \in \mathbb{R}^d$
- Then keys and values are drawn from the **encoder** (like a memory):
  - $k_i = K h_i$, $v_i = V h_i$.
- And the queries are drawn from the **decoder**, $q_i = Q z_i$.

The Transformer Encoder: **Cross-attention (details)**

- Let's look at how cross-attention is computed, in matrices.
  - Let $\mathrm{H} = [h_1; \dots; h_T] \in \mathbb{R}^{T \times d}$ be the concatenation of encoder vectors.
  - Let $\mathrm{Z} = [z_1; \dots; z_T] \in \mathbb{R}^{T \times d}$ be the concatenation of decoder vectors.
  - The output is defined as $\text{output} = \ \ \text{softmax}(ZQ(HK)\ \ ) \times HV$.

First, take the query-key dot products in one matrix multiplication: $ZQ\ \ (HK)$

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



Great Results with Transformers

Next, document generation!

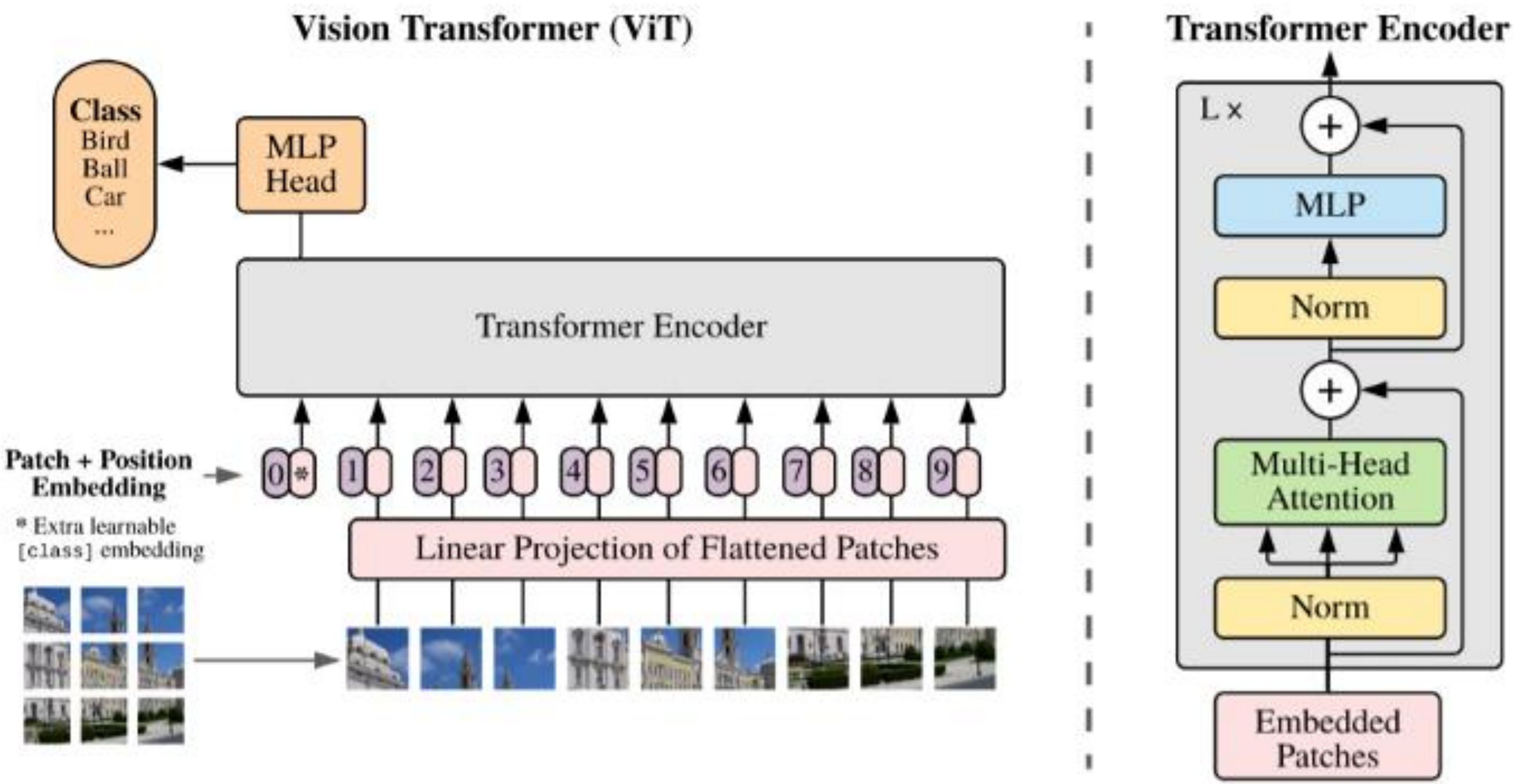
Model	Test perplexity	ROUGE-L
seq2seq-attention, $L = 500$	5.04952	12.7
Transformer-ED, $L = 500$	2.46645	34.2
Transformer-D, $L = 4000$	2.22216	33.6
Transformer-DMCA, no MoE-layer, $L = 11000$	2.05159	36.2
Transformer-DMCA, MoE-128, $L = 11000$	1.92871	37.9
Transformer-DMCA, MoE-256, $L = 7500$	1.90325	38.8

The old standard

Transformers all the way down.

[Liu et al., 2018]; WikiSum dataset

Transformers in vision



Dosovitskiy, ICLR 2021, https://github.com/google-research/vision_transformer https://www.youtube.com/watch?v=TrdevFK_am4

Cross-modal transformers

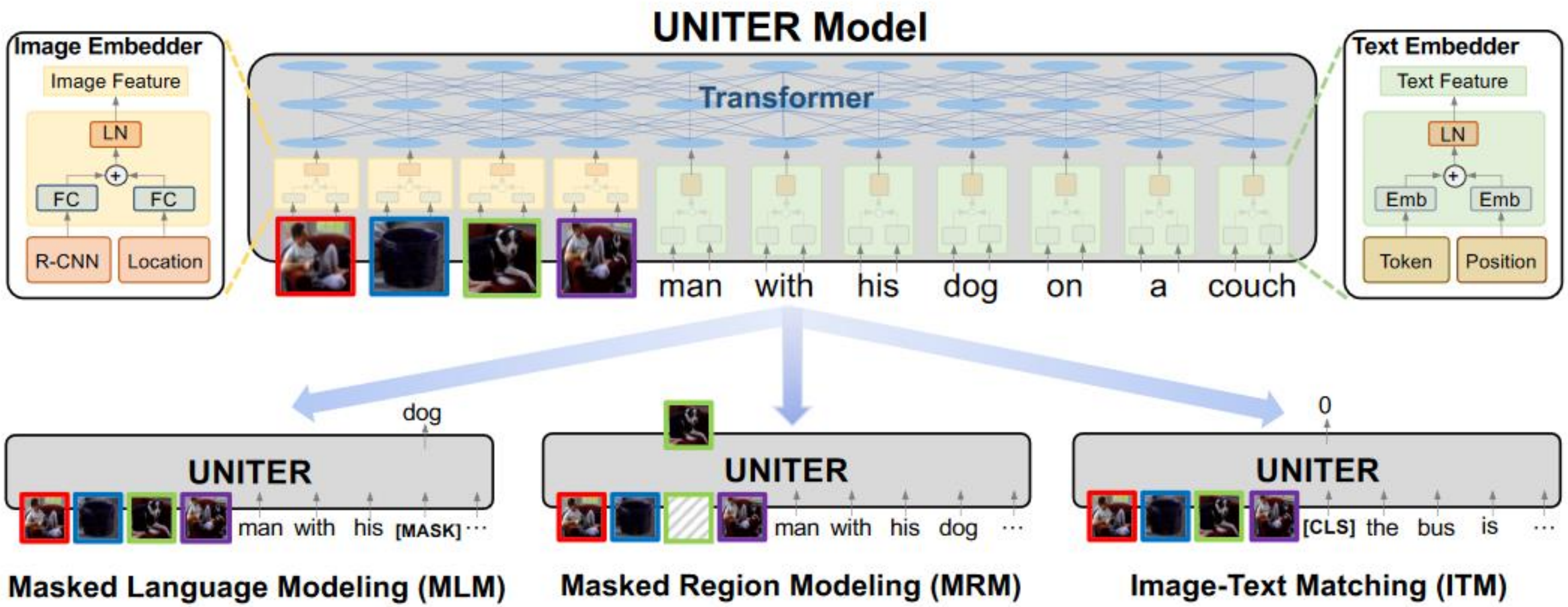


Figure 1: Overview of the proposed UNITER model (best viewed in color), consisting of an Image Embedder, a Text Embedder and a multi-layer self-attention Transformer, learned through three pre-training tasks.

Chen et al., "UNITER: Learning UNiversal Image-TExt Representations", arxiv 2019

Cross-modal transformers

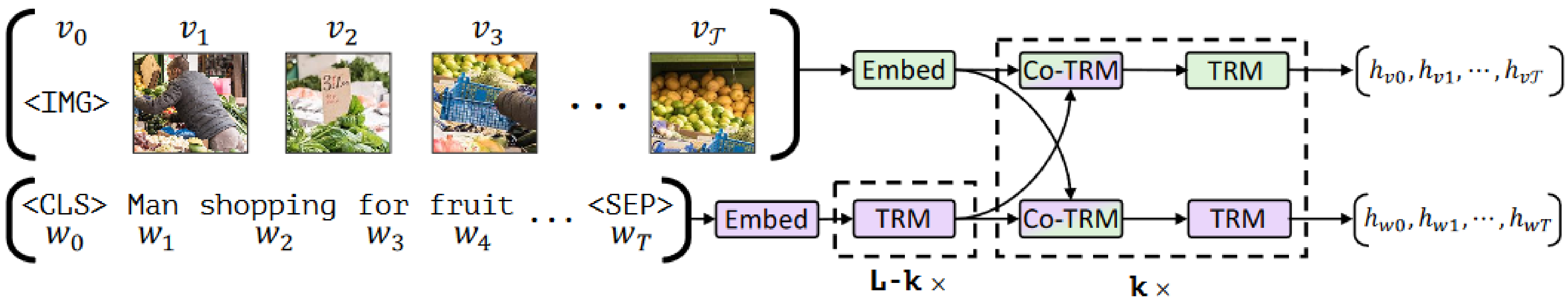


Figure 1: Our ViLBERT model consists of two parallel streams for visual (green) and linguistic (purple) processing that interact through novel co-attentional transformer layers. This structure allows for variable depths for each modality and enables sparse interaction through co-attention. Dashed boxes with multiplier subscripts denote repeated blocks of layers.

Lu et al., "ViLBERT: Pretraining Task-Agnostic Visiolinguistic Representations for Vision-and-Language Tasks", NeurIPS 2019

Cross-modal transformers

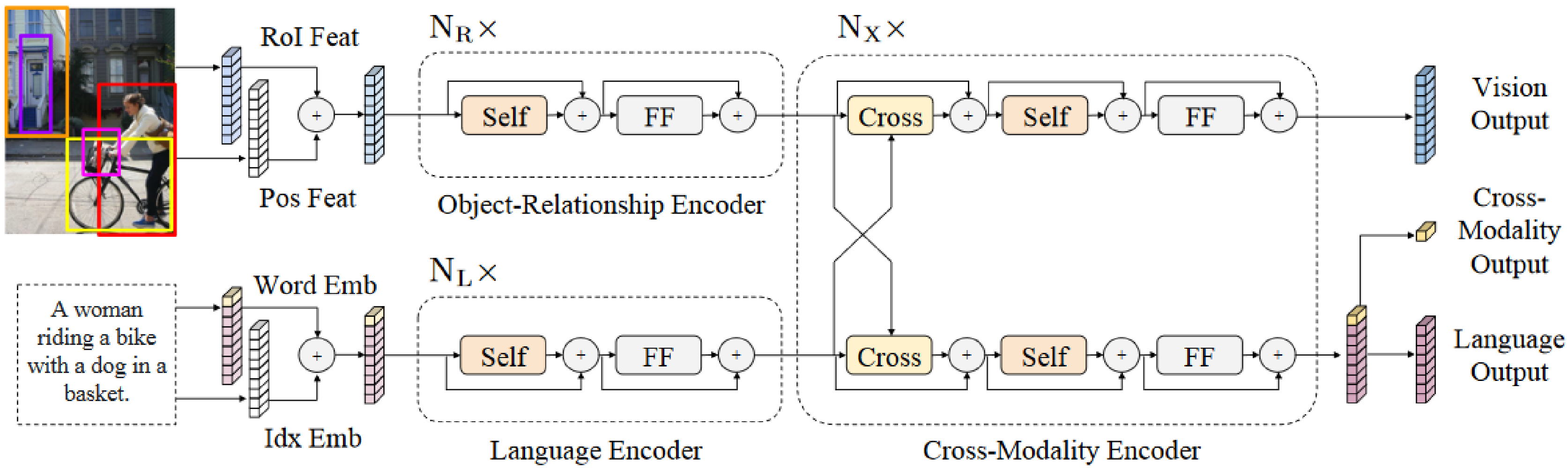


Figure 1: The LXMERT model for learning vision-and-language cross-modality representations. 'Self' and 'Cross' are abbreviations for self-attention sub-layers and cross-attention sub-layers, respectively. 'FF' denotes a feed-forward sub-layer.

Tan and Bansal, "LXMERT: Learning Cross-Modality Encoder Representationsfrom Transformers", EMNLP 2019