

Lecture 9 - Self- Attention and Transformers

Lecture Plan

From recurrence to attention based NLP models

Introducing the Transformer model

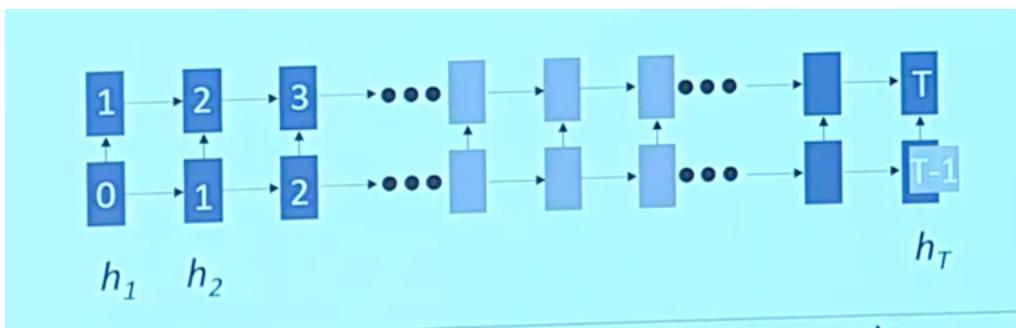
Great results with Transformers

Drawbacks and variants of Transformer

Issues with recurrent models : Linear interaction distance

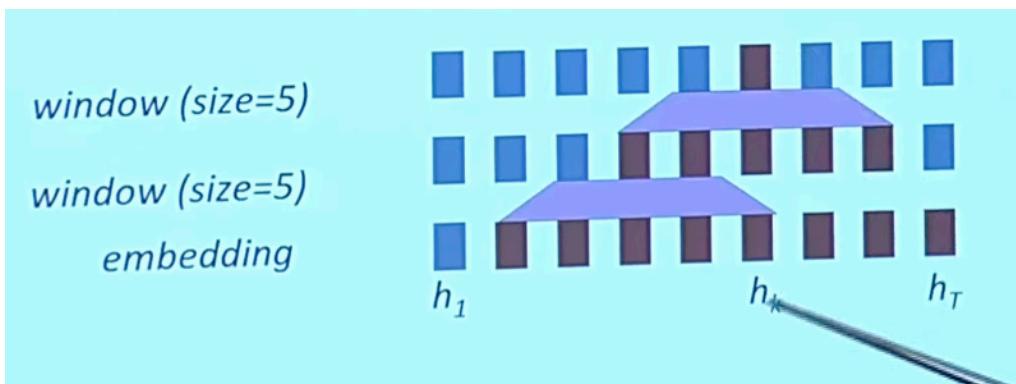
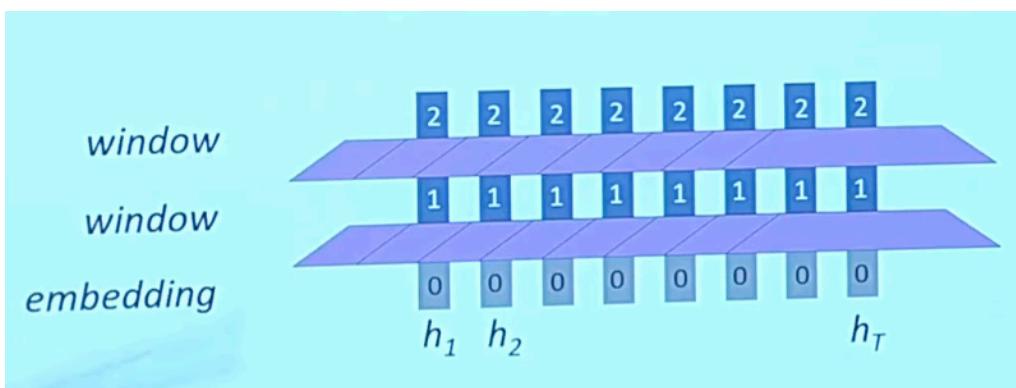
Problem : RNN take $O(\text{sequence length})$ steps for distant word pairs to interact

Issues with recurrent models : Lack of parallelizability



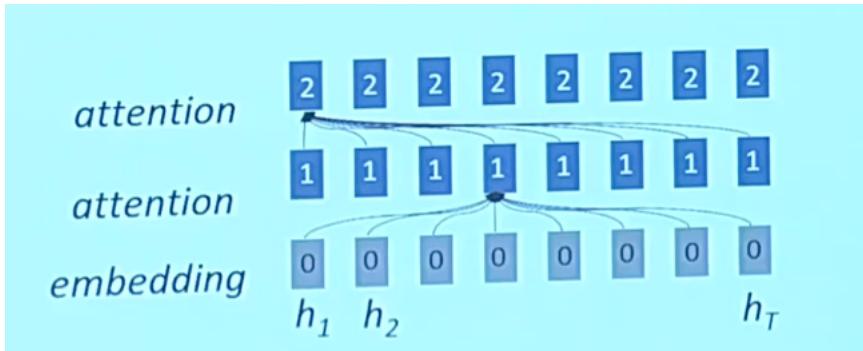
If not recurrence, then what? How about word windows?

Word window models aggregates total contexts



If not recurrence, then what? How about attention?

Attention treats each word's representation as a query to access and incorporate information from a set of values



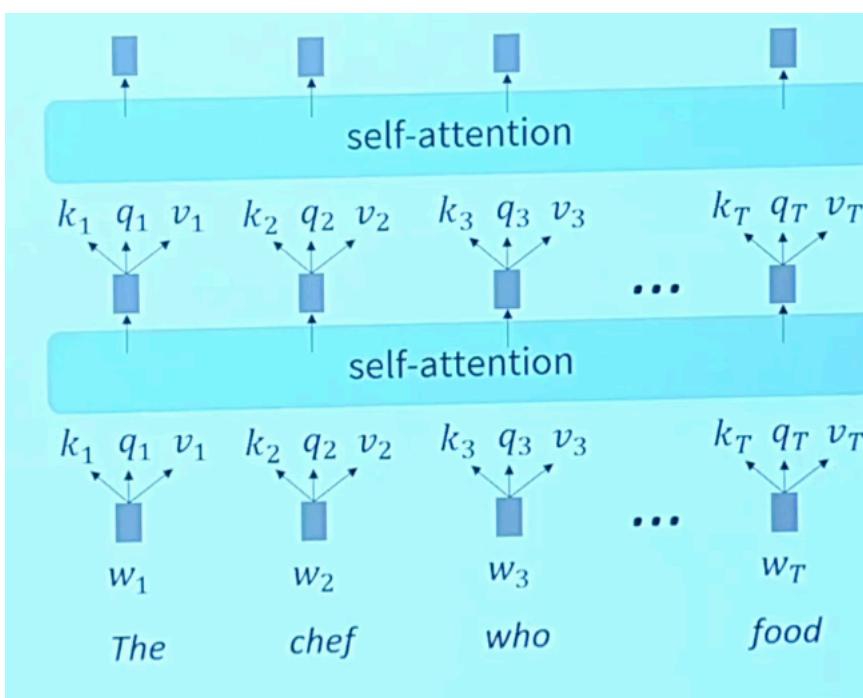
Self-Attention

Queries, keys and values

Ex) if the output of the previous layer is x_1, \dots, x_T

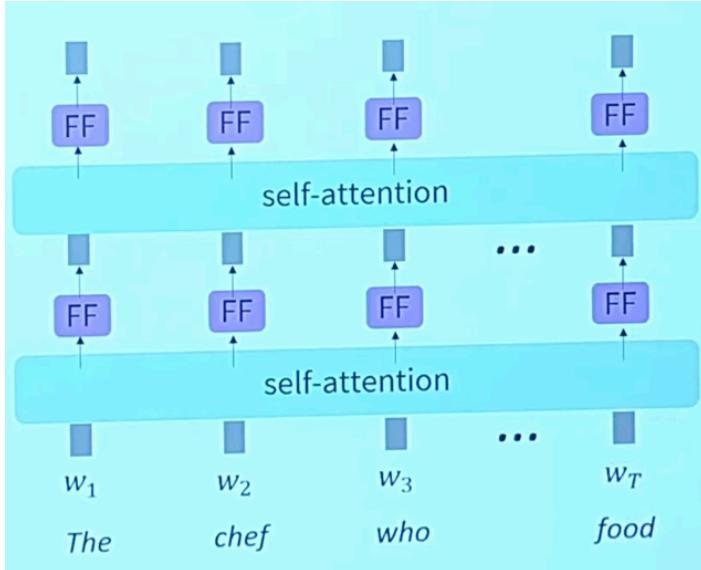
We could let $v_i = k_i = q_i = x_i$

Self-attention as an NLP building block



Fixing the first self-attention problem : sequence order

Adding nonlinearities in self attention



Masking the future in self-attention

At every timestep, we could change the set of keys and queries to include only past words

To enable parallelization, we mask out attention to future words by setting attention scores to -inf.

Necessities for a self-attention building block

Self-attention

Position representation

Nonlinearities

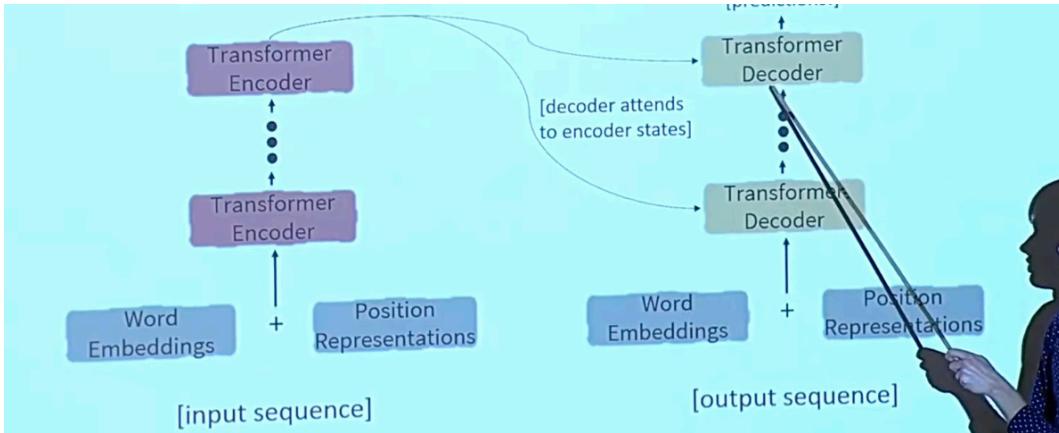
- at the output of the self attention block
- frequency implemented as a simple feed-forward network

Masking

- in order to parallelize operations while not looking at the future
- keeps information about the future from leaking to the past

2. Introducing the Transformer model

The Transformer Encoder-Decoder



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 single layer
3. Tricks to help with training
 1. Residual connections
 2. Layer normalization
 3. Scaling the dot product

The Transformer Encoder : Key-Query-Value Attention

X_1, \dots, x_T be input vectors to the Transformer encoder

Then keys, queries, values akre

$K_i = Kx_i$, where K in $\mathbb{R}^{d \times d}$ is the key matrix

$Q_i = Qx_i$, where Q in $\mathbb{R}^{d \times d}$ is query matrix

$V_i = Vx_i$, where V in $\mathbb{R}^{d \times d}$ is value matrix

- 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 $\text{output} = \text{softmax}(XQ(XK)^T) \times XV$.

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

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

All pairs of attention scores!

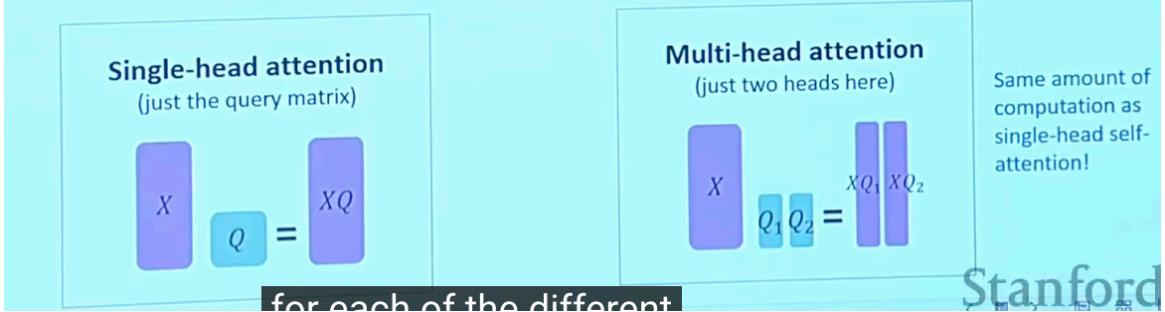
Next, softmax, and compute the weighted average with another matrix multiplication.

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

Step 2

The Transformer Encoder: Multi-headed attention

- 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 .



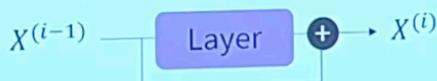
Stanford

The Transformer Encoder : Residual connections

- **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!)

The Transformer Encoder : 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:

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

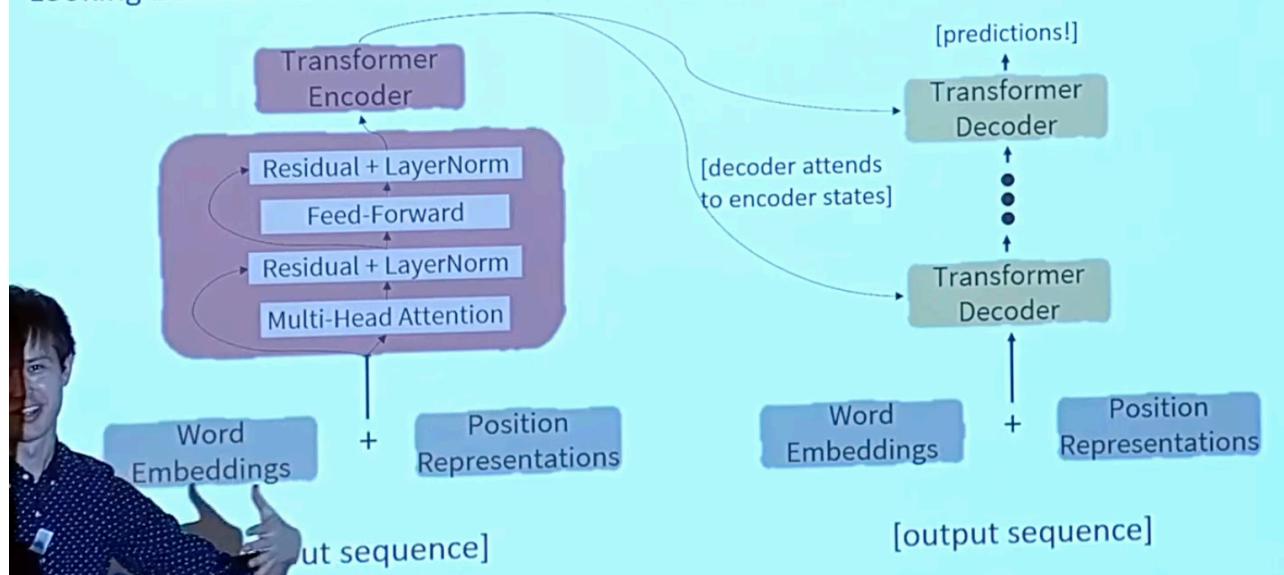
Normalize by scalar Modulate by learned elementwise gain and bias

The Transformer Encoder : Scaled Dot product

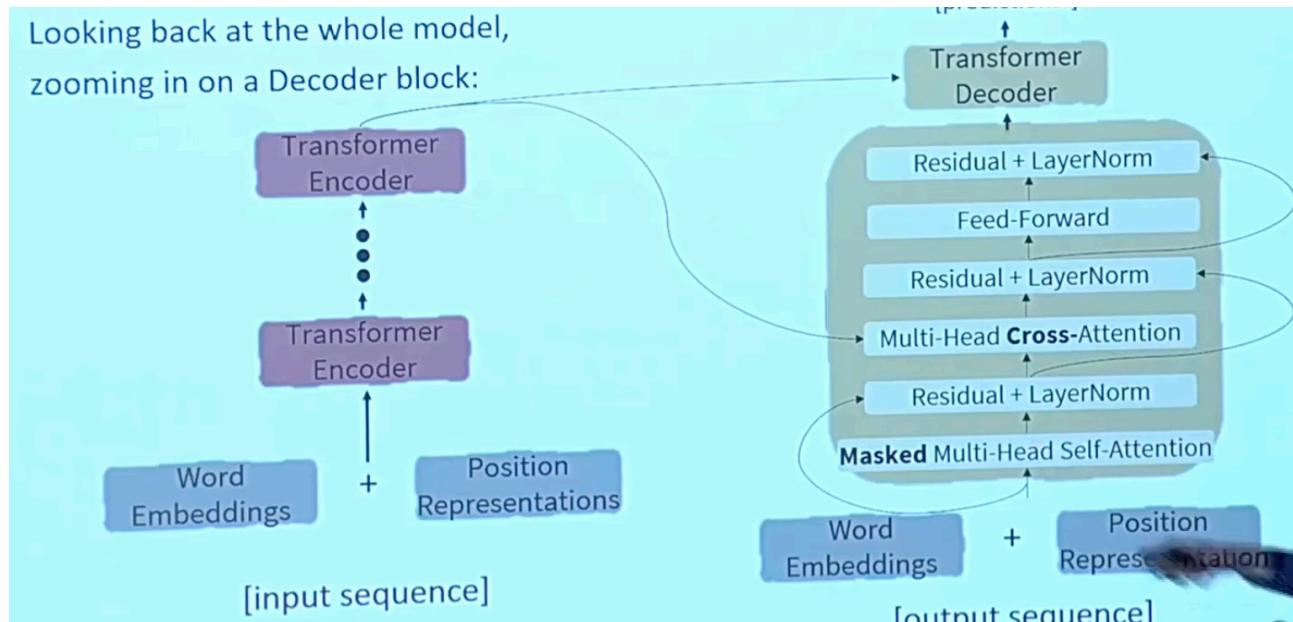
- We divide the attention scores by $\sqrt{d/h}$, to stop the scores from becoming large just as a function of d/h (The dimensionality divided by the number of heads.)

$$\text{output}_\ell = \text{softmax}\left(\frac{XQ_\ell K_\ell^T X^T}{\sqrt{d/h}}\right) * X V_\ell$$

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



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



The Transformer Decoder : Cross-attention

- We saw that 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):
 - $k_i = Kh_i, v_i = Vh_i$.
- And the queries are drawn from the **decoder**, $q_i = Qz_i$.



The Transformer Encoder : Cross-attention

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

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

$$ZQ \begin{pmatrix} K^\top H^\top \end{pmatrix} = ZQK^\top H^\top \in \mathbb{R}^{T \times T}$$

All pairs of attention scores!

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

$$\text{softmax} \begin{pmatrix} ZQK^\top H^\top \end{pmatrix} \begin{pmatrix} HV \end{pmatrix} = \text{output} \in \mathbb{R}^{T \times d}$$

Stanfor

