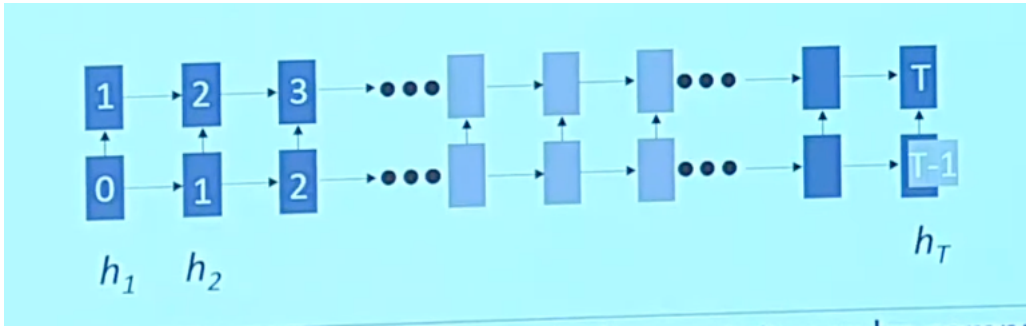
Lecture 9 - Self- Attention and Transformers

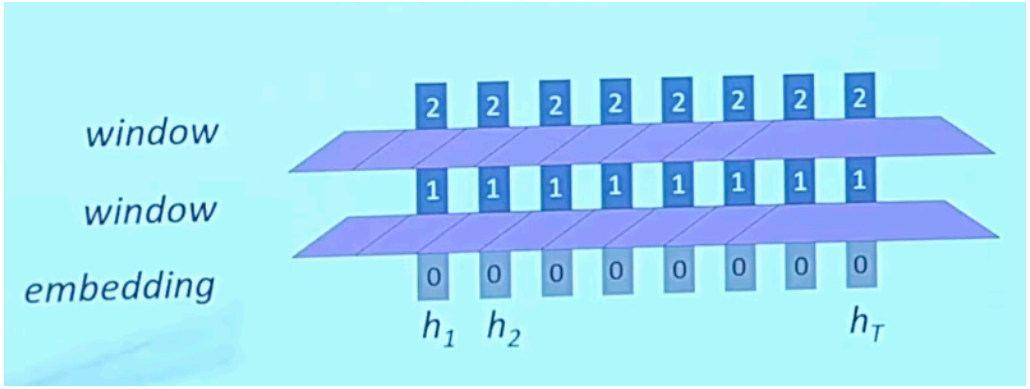
**Lecture Plan**
From recurrence to attention based NLP models
Introducing the Transformer model
Great results with Transformers
Drawbacks and variants of Transformer

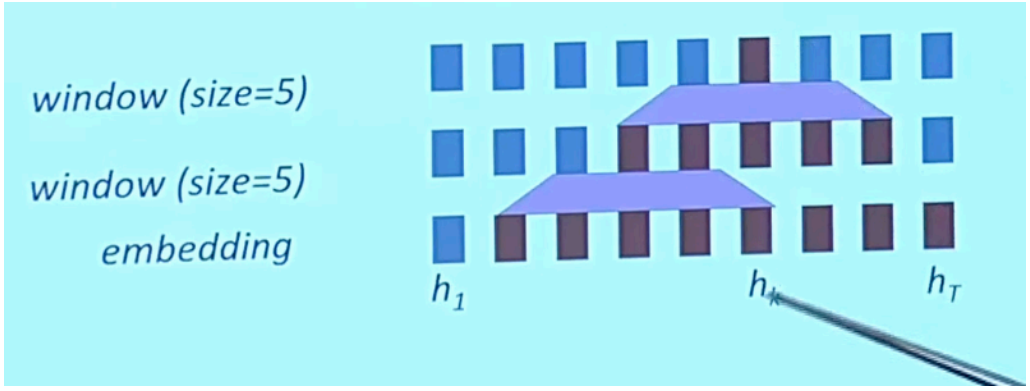
**Issues with recurrent models : Linear interactgion distance**
Problem : RNN take O(sequence length) steps for distant word pairs to interact

**Issues with recurrent models : Lack of parallelizability**



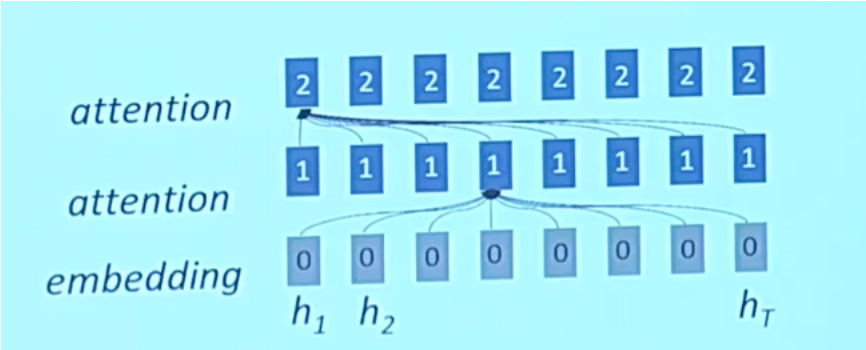

If not recurrence, then what? How about word windows?
Word window models aggregates total contexts



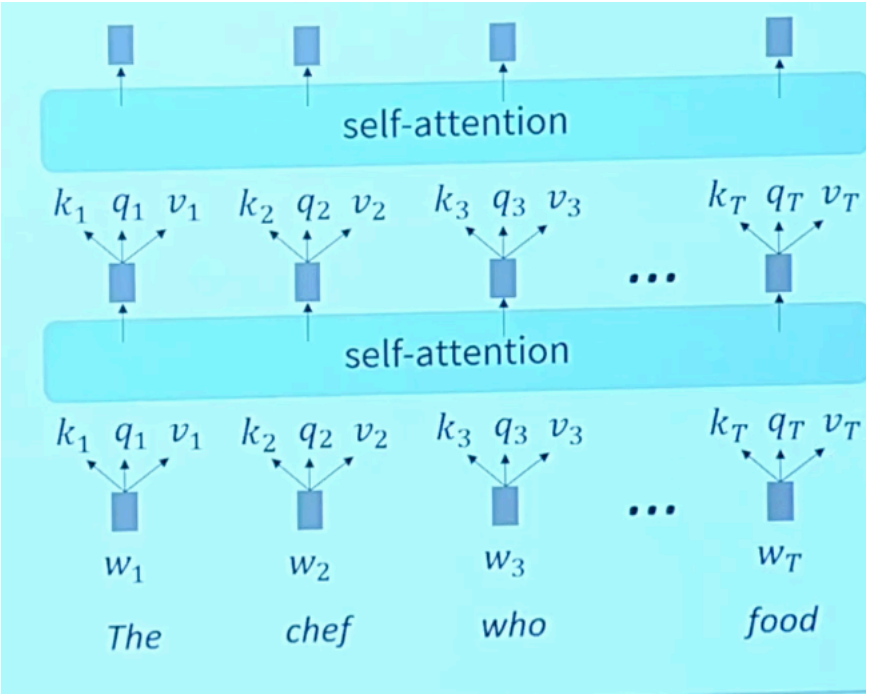
**If not recurrence, then what? How about attention?**

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



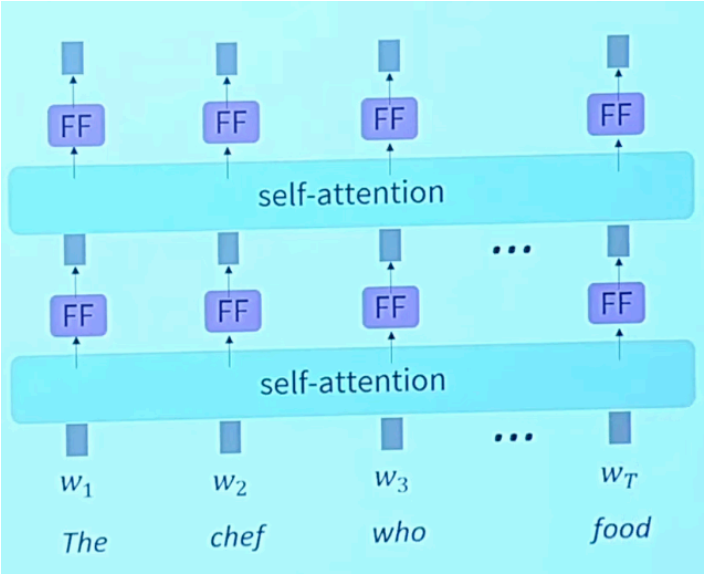

**Self-Attention**
Queries, keys and values
Ex) if the output of the previous layer is x1,…,xt
We could let vi = ki = qi = xi

**Self-attention as an NLP building block**



**Finxing the first self-attention problem : sequence order**

**Adding nonlinearities in self attention**




**Masking the future in self-attention**

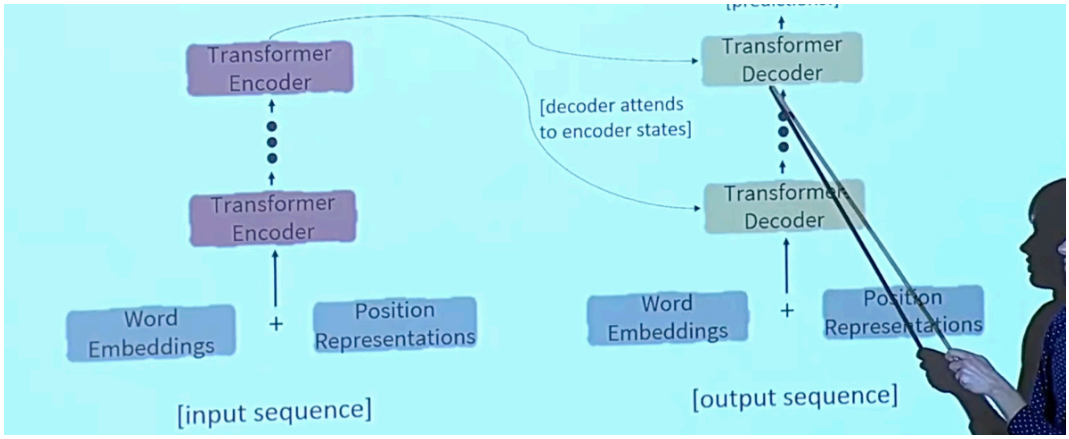
At every timestep, we could change the set of keys and queries to include only past words
To enable parallelization, we mask out attention to future words by setting attention scores to -inf.

**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

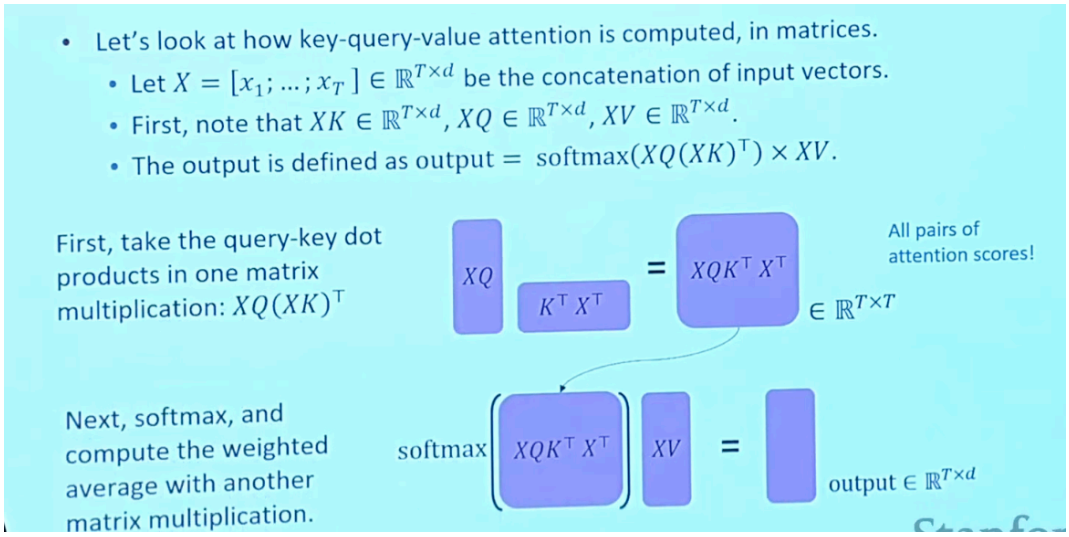



What's left in a Transformer Encoder Block that we haven't covered?

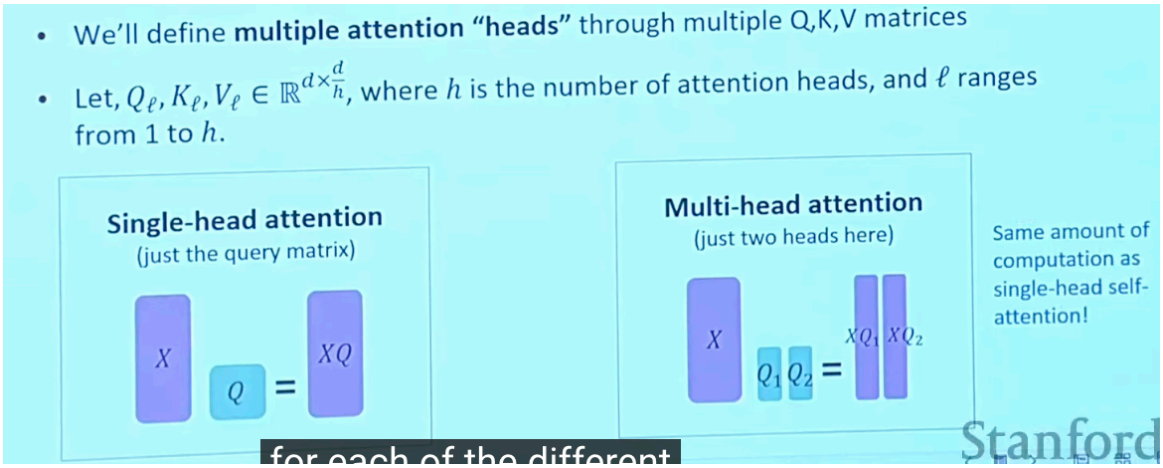
1. Key-query-value attention : How do we get the k, q, v vectors from a single word embedding?
2. Multi-headed attention : Attend to multiple places single layer
3. Tricks to help with training
    1. Residual connections
    2. Layer normalization
    3. Scaling the dot product

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

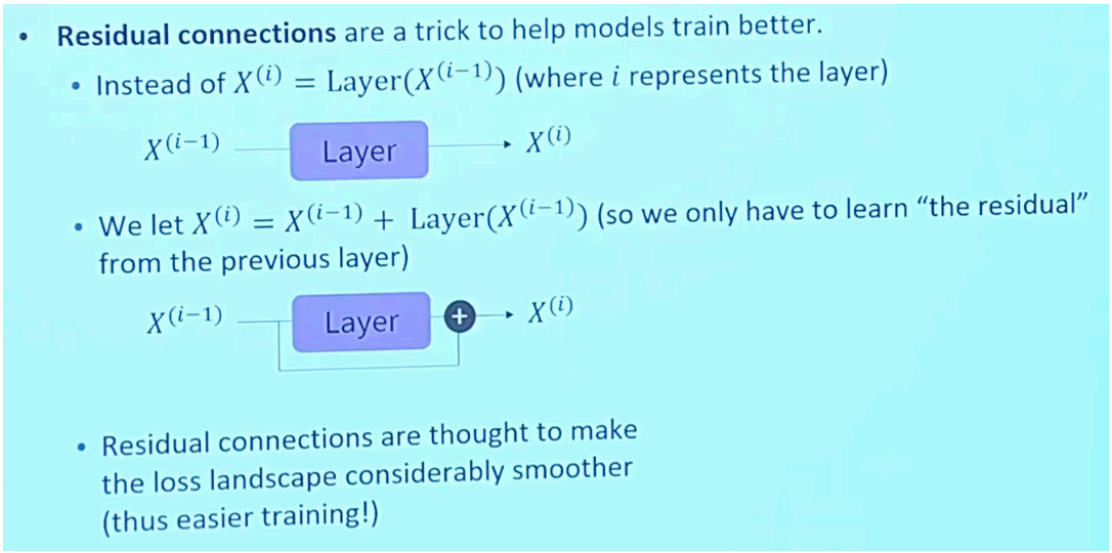
X1,…,xt be input vectors to the Transformer encoder
Then keys, queries, values akre
Ki = Kxi, where K in R(dxd) is the key matrix
Qi = Qxi, where Q in R(dxd) is query matrix
Vi = Vxi, where V iin R(dxd) is value matrix



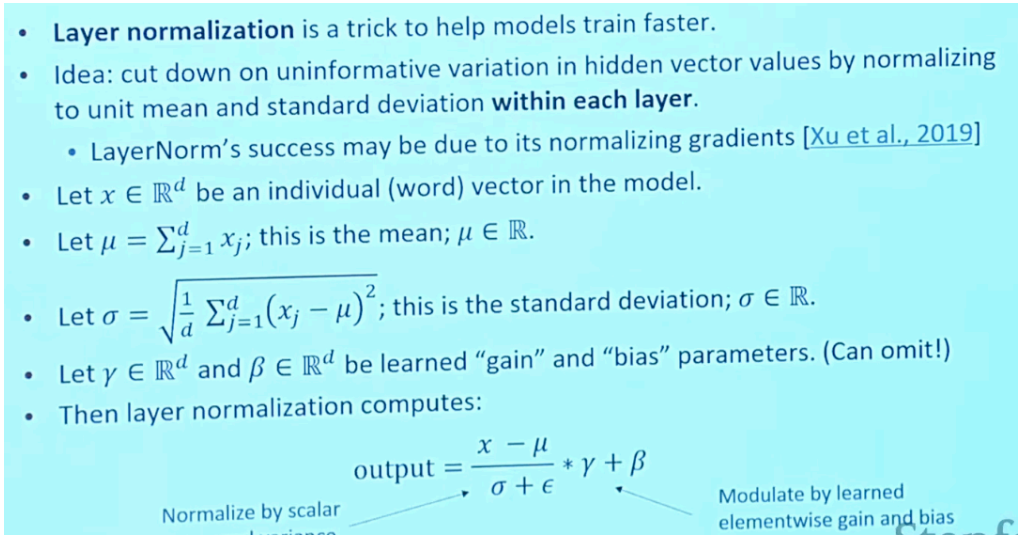
## The Transformer Encoder: Multi-headed attention




## The Transformer Encoder : Residual connections



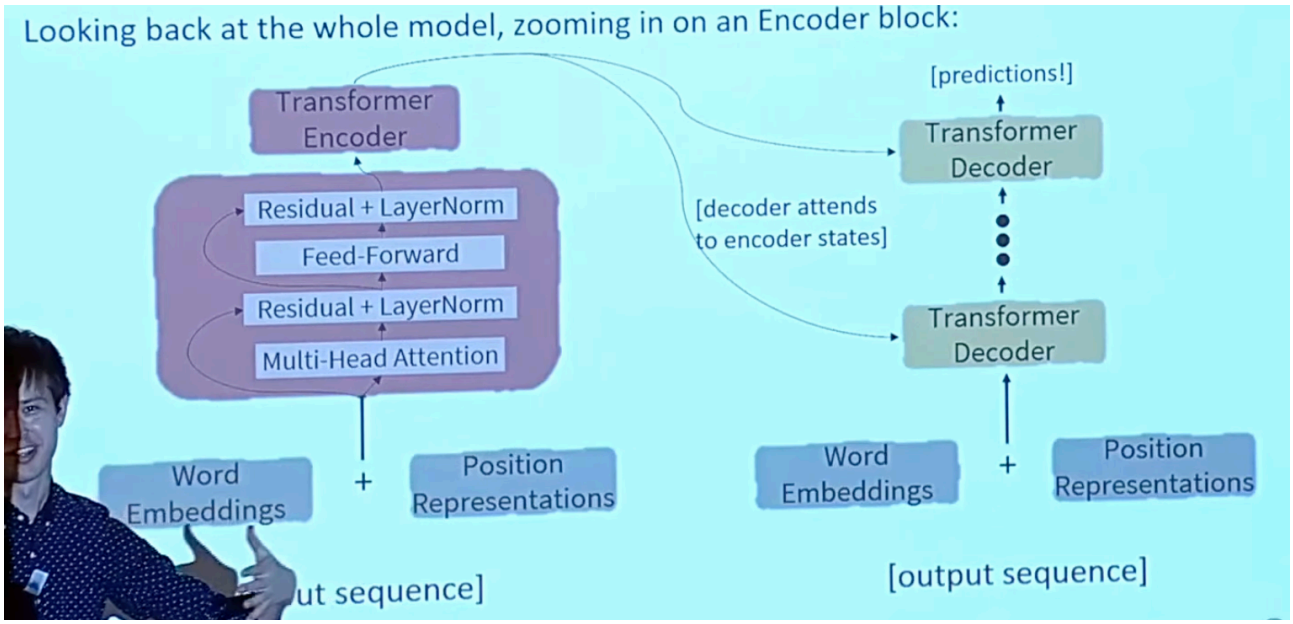

## The Transformer Encoder : Layer normalization

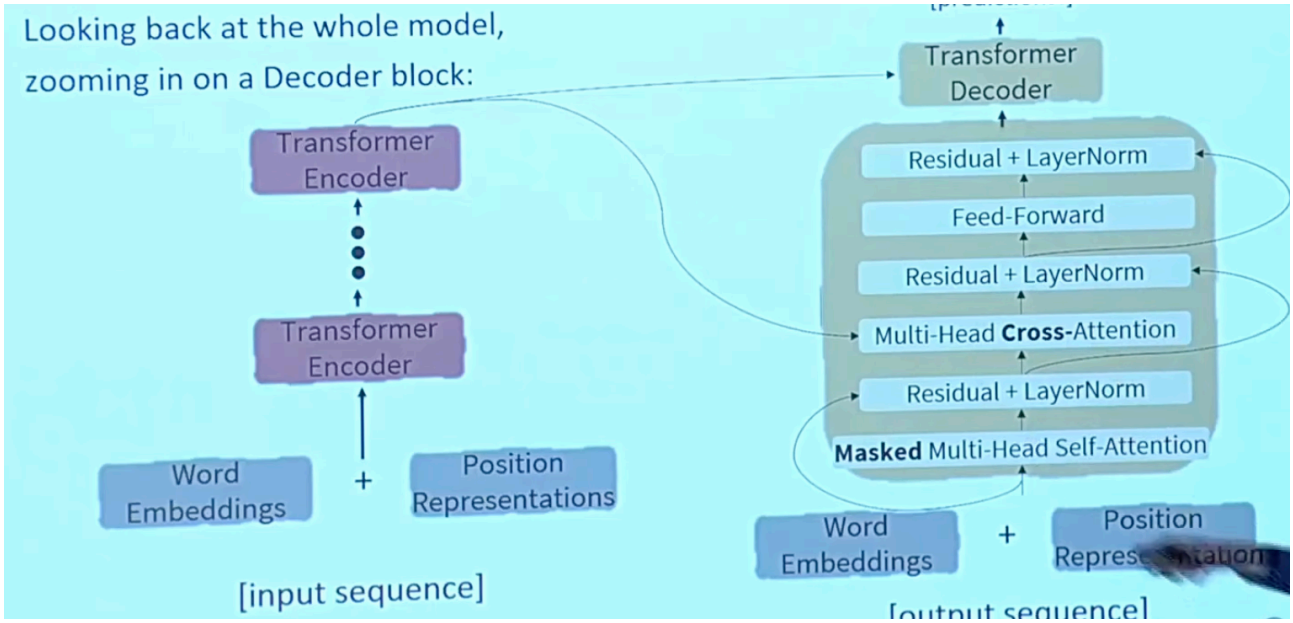


## The Transformer Encoder : Scaled Dot product

- 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.)

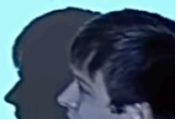
$$\text{output}_\ell = \text{softmax}\left(\frac{XQ_\ell K_\ell^\top X^\top}{\sqrt{d/h}}\right) * XV_\ell$$



## The Transformer Decoder : Cross-attention

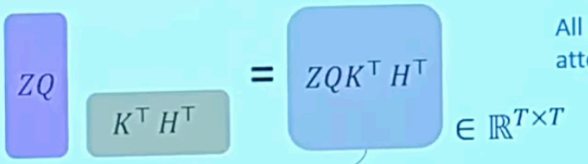
- We saw that self-attention is when keys, queries, and values come from the same source.
- In the decoder, we have attention that looks more like what we saw last week.
- Let $h_1, \dots, h_T$ be **output** vectors **from** the Transformer **encoder**; $x_i \in \mathbb{R}^d$
- Let $z_1, \dots, z_T$ 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**, $q_i = Qz_i$.



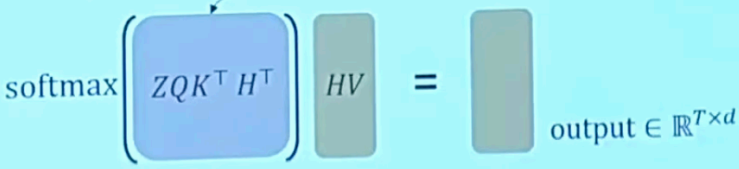
## The Transformer Encoder : Cross-attention

- Let's look at how cross-attention is computed, in matrices.
  - Let $\mathrm{H} = [h_1; \dots; h_T] \in \mathbb{R}^{T \times d}$ 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 output $= \mathrm{softmax}(ZQ(HK)^\top) \times HV$.

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




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




Stanfo