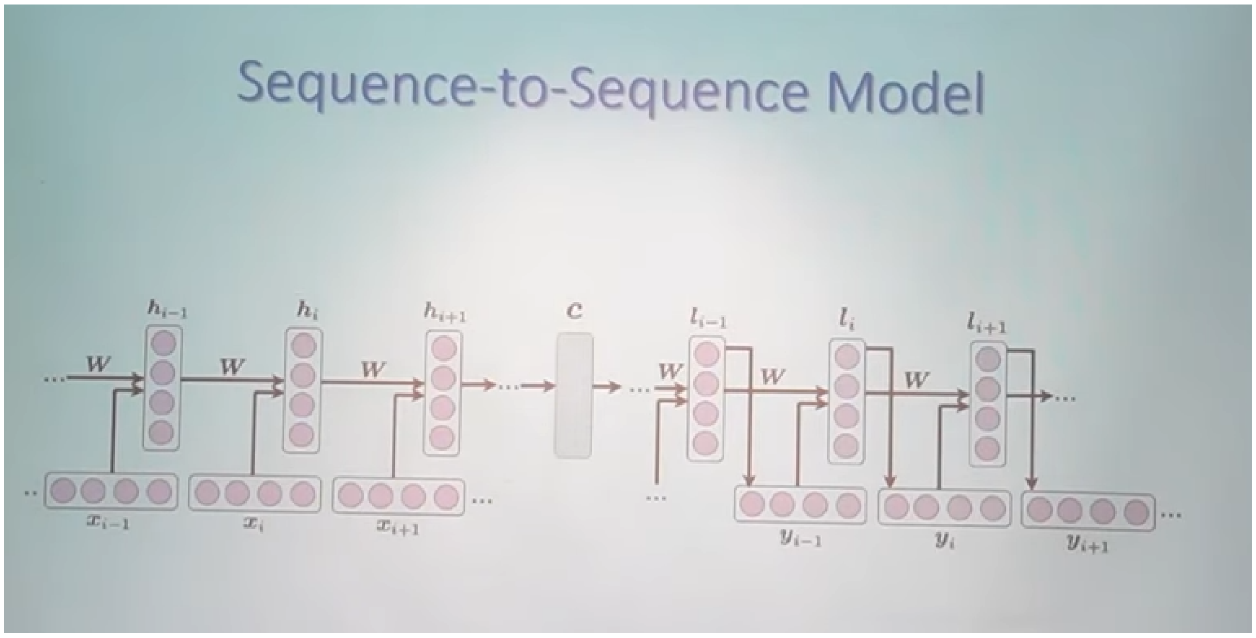
Lecture 9: Attention mechanism, self-attention, S2S

S2S model using RNN based Encoder-Decoder

Consider the word elephant. There are many different representations of the idea/abstraction of elephant. It can be represented in different languages or an image also. But the core idea/abstraction of an elephant is the same. So the encoder, encodes the word elephant into a vector(context vector) which carries an abstraction of an elephant and a decoder decodes this into different forms like a different language or an image.

Both Encoder and Decoder can be made using a RNNs. In this S2S model, a single context vector is passed to the decoder.



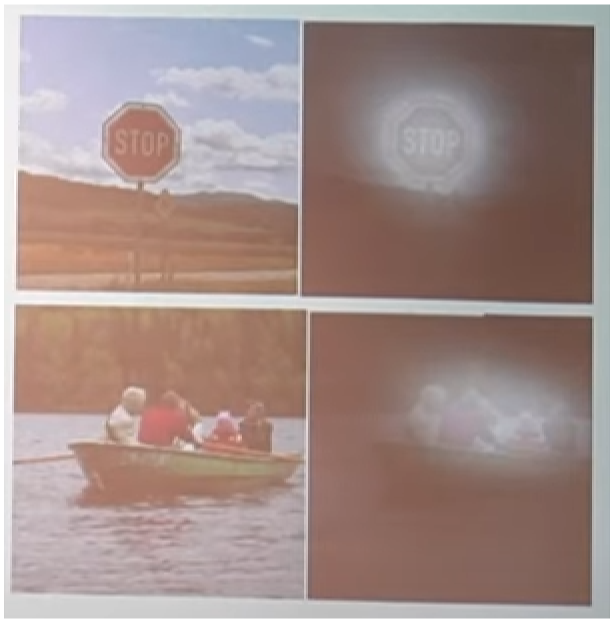
However, there are some problems with RNNs.

1. Long-range dependencies
   If we have a long sentence, then the dependency of the hidden vector obtained at the end on the initial words in the sentence is negligible i.e. we have forgotten the beginning of the sentence.
2. The sequential nature of the model architecture prevents parallelization.

Attention

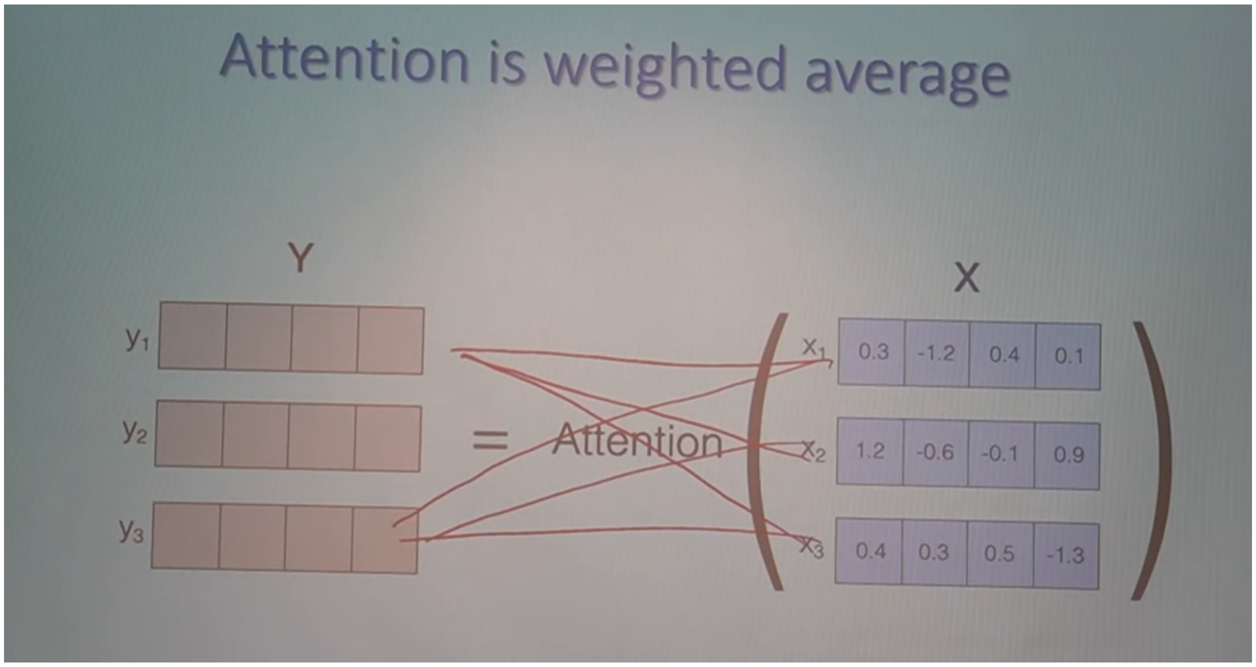
Directing your focus towards important parts when processing data. This is the basic idea behind attention.

> Attention is a fancy name for weighted average.

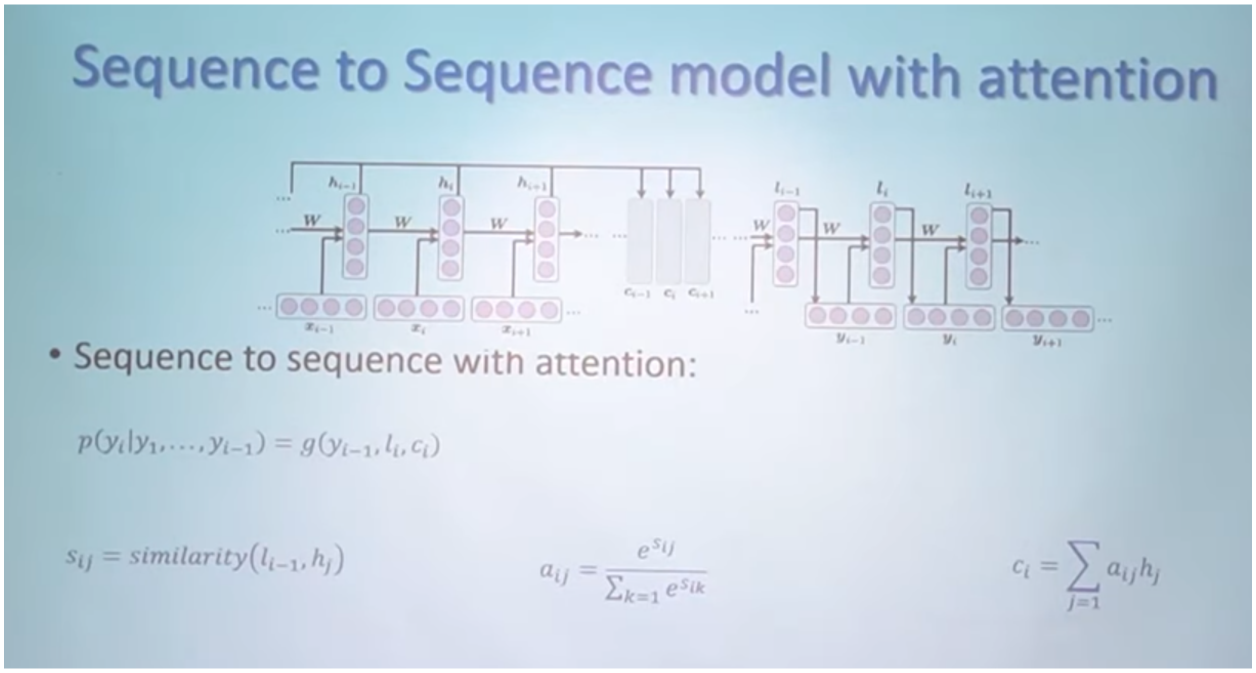


Here instead of a single context vector, multiple context vectors are are formed. Each context vector is a weighted combination of hidden vectors. These weighted sums are formed on the basis of what the decoder wants to predict at that moment.

If the first part of the sentence is important, then higher weights are given to h1. If the last part of the sentence is important, then higher weights are given to h3 and so on. So depending on what it wants to predict, it chooses the context vector. So the various weighted sums of hidden vectors are present as context vectors for the decoder. Depending on what it wants to predict, it chooses the context vector.



What we essentially want to compute is the conditional probability of a word being the next word, given the words you have seen so far. That probability is given by the function that represents the Sequence to Sequence model. We use similarity score and softmax function to decide the weights for each of the hidden vectors in the encoder. Then we add up these weighted hidden vectors to give the context vector. The function mentioned above uses this context vector along with the previous word and the current hidden vector in the decoder to give us the conditional probability.



Sequence to Sequence model with attention

- Sequence to sequence with attention:

$$p(y_i|y_1,\ldots,y_{i-1}) = g(y_{i-1}, l_i, c_i)$$

$$s_{ij} = similarity(l_{i-1}, h_j) \qquad a_{ij} = \frac{e^{s_{ij}}}{\sum_{k=1} e^{s_{ik}}} \qquad c_i = \sum_{j=1} a_{ij} h_j$$

For each yi, ci is computed at that time itself.

Note: For the entire vocabulary we have distribution probability.
To the function gi we give as input the previous word(yi), the output generated for current state(li) and the context vector ci which takes into account the previous words.
In RNN based S2S, we have access to c after a chain of computations. In the attention based model, we have direct access to ci.

Self Attention

Terminology: Key, Query and Value
From database literature.

We have a key, value table and a query. Corresponding to the query we want a value.

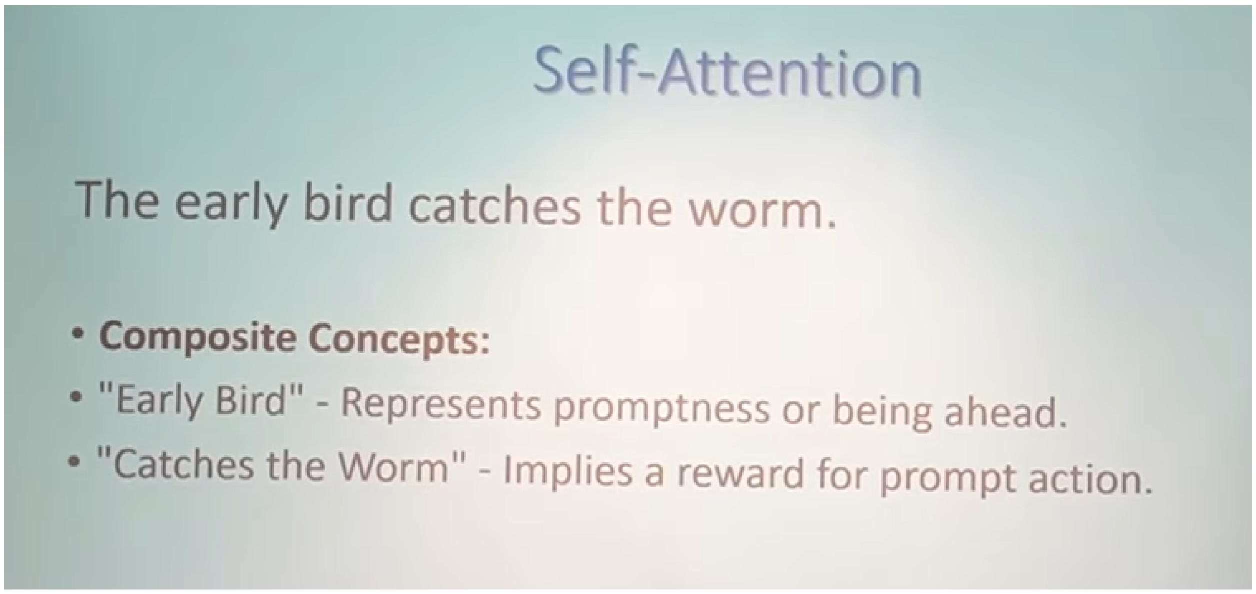
In some cases our query matches exactly with a key and we get the value. But in some other cases our query doesn’t exactly match with any key. In that case we calculate the similarity score of the query with each value, find softmax coefficients and take weighted average of the values with these coefficients. We get the value corresponding to our query.

This is exactly what we do in self attention.

Intuition:

Understanding individual words is not sufficient to understand the sentence.

We need to know how the words relate to each other.

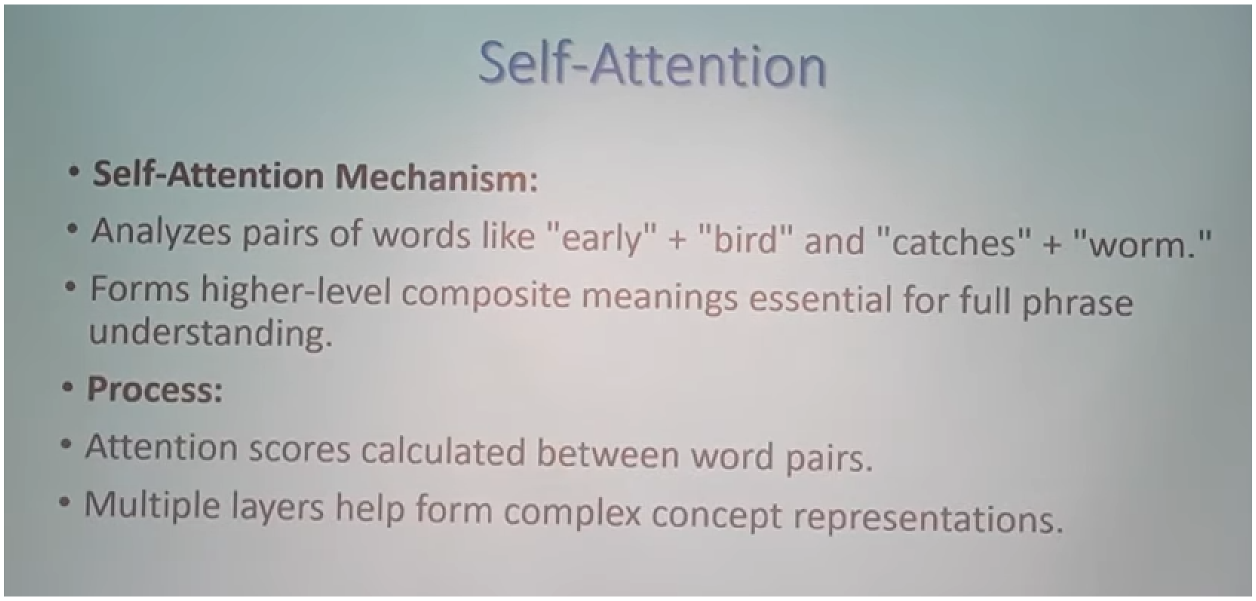


Self-Attention

The early bird catches the worm.

- **Composite Concepts:**
- "Early Bird" - Represents promptness or being ahead.
- "Catches the Worm" - Implies a reward for prompt action.

Here Early Bird doesn’t exactly refer to a bird and catches the worm doesn’t mean literally catching a worm.



Self-Attention

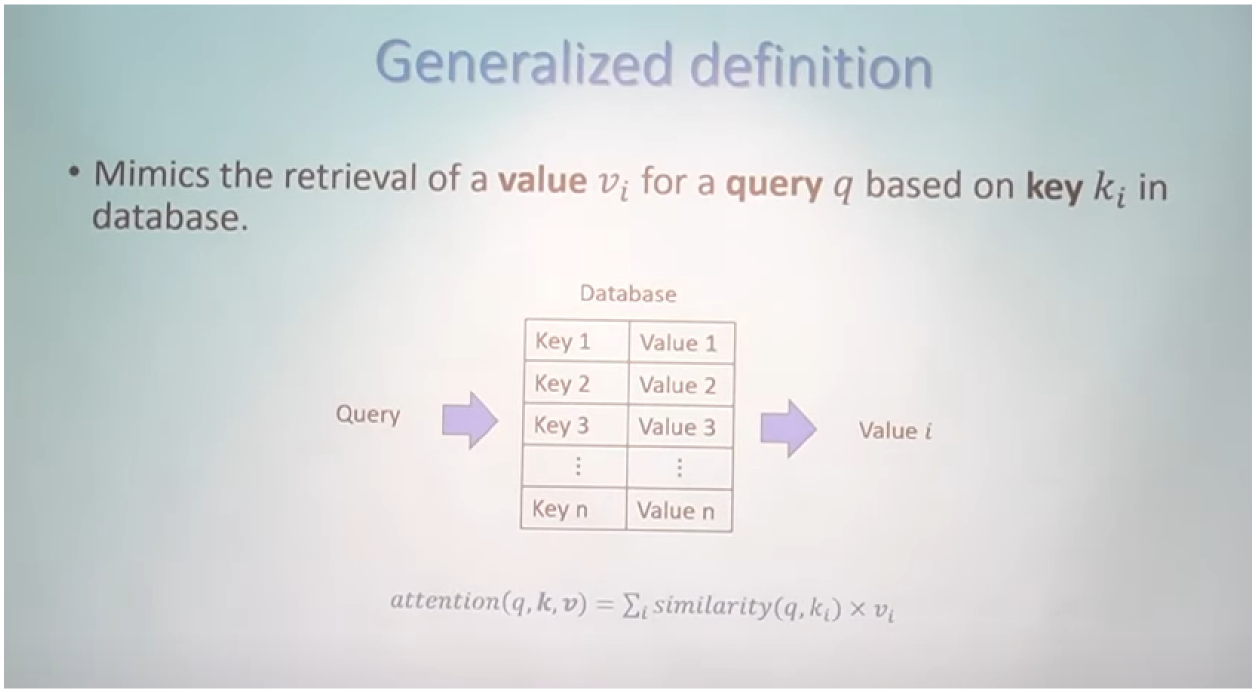
- **Self-Attention Mechanism:**
- Analyzes pairs of words like "early" + "bird" and "catches" + "worm."
- Forms higher-level composite meanings essential for full phrase understanding.
- **Process:**
- Attention scores calculated between word pairs.
- Multiple layers help form complex concept representations.

Generalized definition of Attention

Finding the value for a query on the basis of similarity with keys.

To find the value, we take the weighted average of values based on these similarity values.

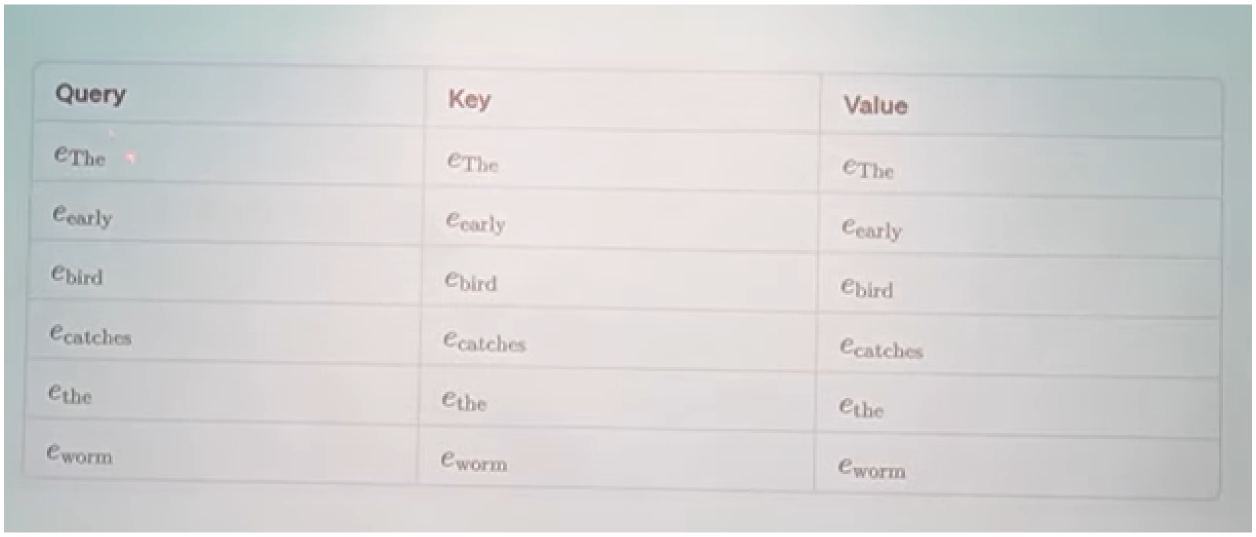
We call it generalized because it works for both, exact and approximate match.


Generalized definition

- Mimics the retrieval of a **value** v_i for a **query** q based on **key** k_i in database.

$$attention(q, \mathbf{k}, \mathbf{v}) = \sum_i similarity(q, k_i) \times v_i$$

In case of words, we keep all words as query, key and value.



Query	Key	Value
e_{The}	e_{The}	e_{The}
e_{early}	e_{early}	e_{early}
e_{bird}	e_{bird}	e_{bird}
e_{catches}	e_{catches}	e_{catches}
e_{the}	e_{the}	e_{the}
e_{worm}	e_{worm}	e_{worm}

Here we look at the vector representation of the words.

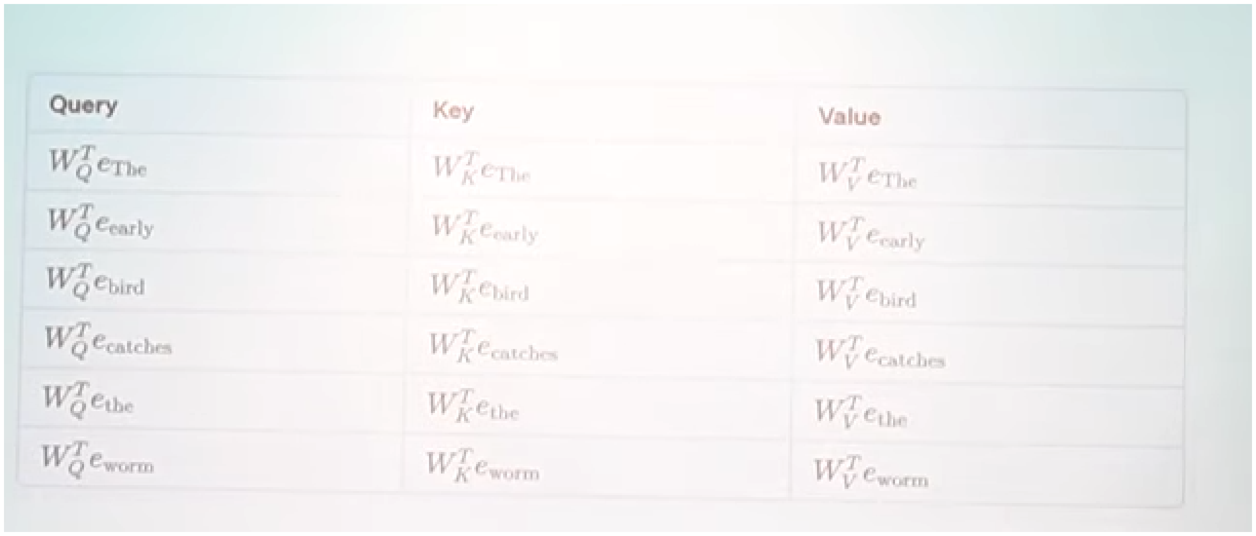
Now these vectors do not inherently capture the similarity between the words ‘early’ and ‘bird’ and neither do they capture the meaning of the phrase, which is ‘being prompt’.

So the vectors have to learn this through the large corpus of data fed to the model. There are two ways to learn this:

1. Change the vectors
2. Change the representation of vectors.

Changing the vectors will lead us to a problem. The problem would be that ‘early’ and ‘bird’ are similar in this particular context but are dissimilar in some other context.

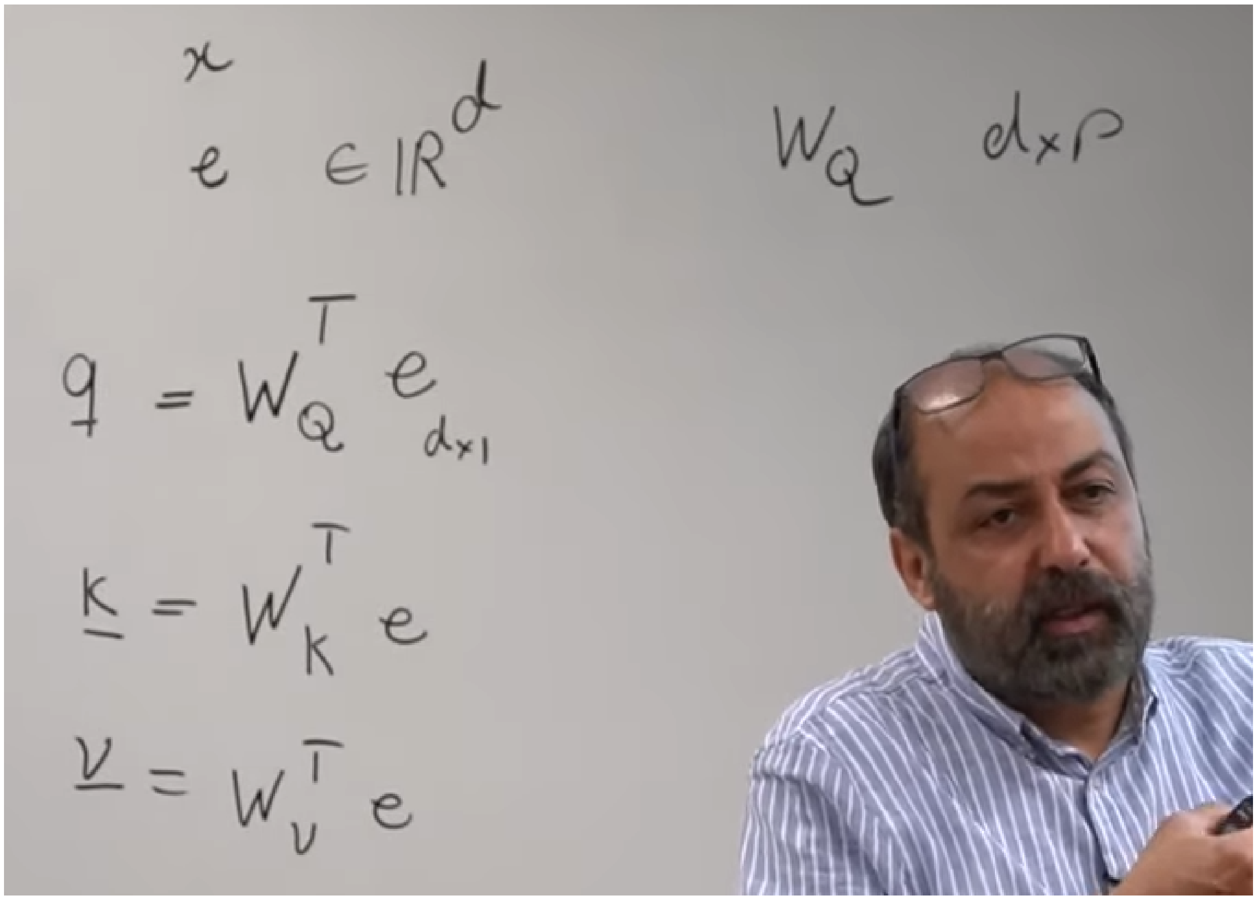
So we need to change the way we represent the vectors.



Query	Key	Value
$W_Q^T e_{\text{The}}$	$W_K^T e_{\text{The}}$	$W_V^T e_{\text{The}}$
$W_Q^T e_{\text{early}}$	$W_K^T e_{\text{early}}$	$W_V^T e_{\text{early}}$
$W_Q^T e_{\text{bird}}$	$W_K^T e_{\text{bird}}$	$W_V^T e_{\text{bird}}$
$W_Q^T e_{\text{catches}}$	$W_K^T e_{\text{catches}}$	$W_V^T e_{\text{catches}}$
$W_Q^T e_{\text{the}}$	$W_K^T e_{\text{the}}$	$W_V^T e_{\text{the}}$
$W_Q^T e_{\text{worm}}$	$W_K^T e_{\text{worm}}$	$W_V^T e_{\text{worm}}$

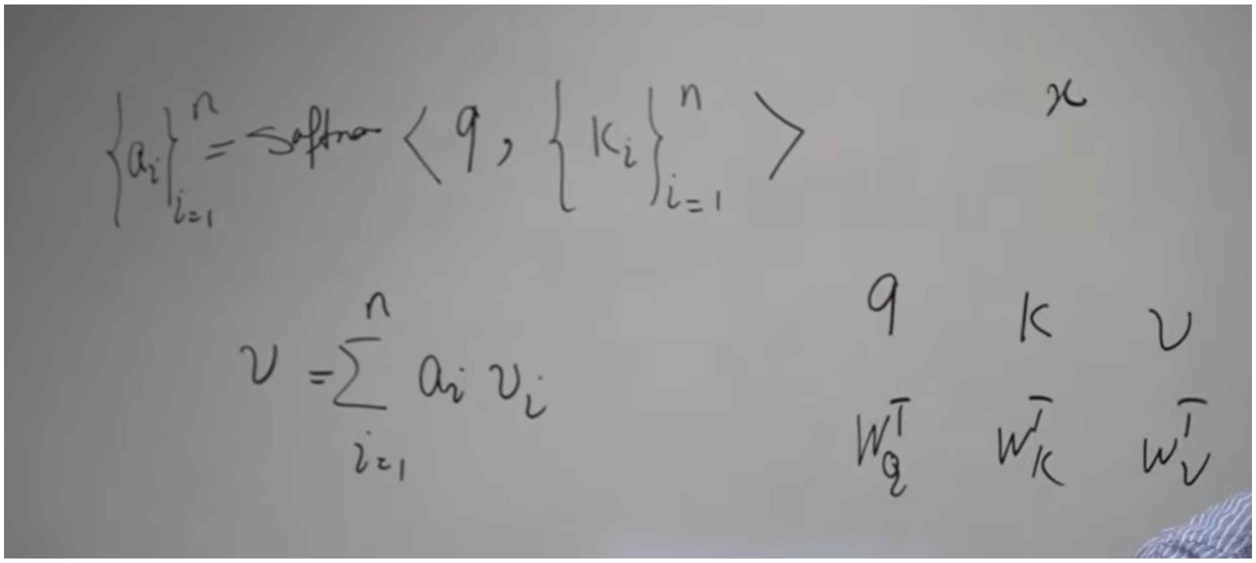
T here means transpose of matrix W

Now we will keep a different representation for each word. Word ‘bird’ will have a different representation as a query, as a key and as a value. That can be easily done transforming the vectors using different matrices.


$\overset{x}{e} \in \mathbb{R}^d$
$W_Q \quad d \times p$
$q = W_Q^T e_{d\times 1}$
$\underline{k} = W_K^T e$
$\underline{v} = W_v^T e$

x is word for which e is a vector representation. e is of dimension d.
We define 3 matrices Wq, Wk and Wv. Take their transpose and multiply by e to get vector q, k and v of dimension p.
q is query vector, k is key vector and v is value vector

So now when we find similarity between words, we are finding similarity score between q of query word and k of all words. It is through back-propagation that we learn e(vector rep of word) and these matrices Wq, Wk and Wv. They are unknown initially.

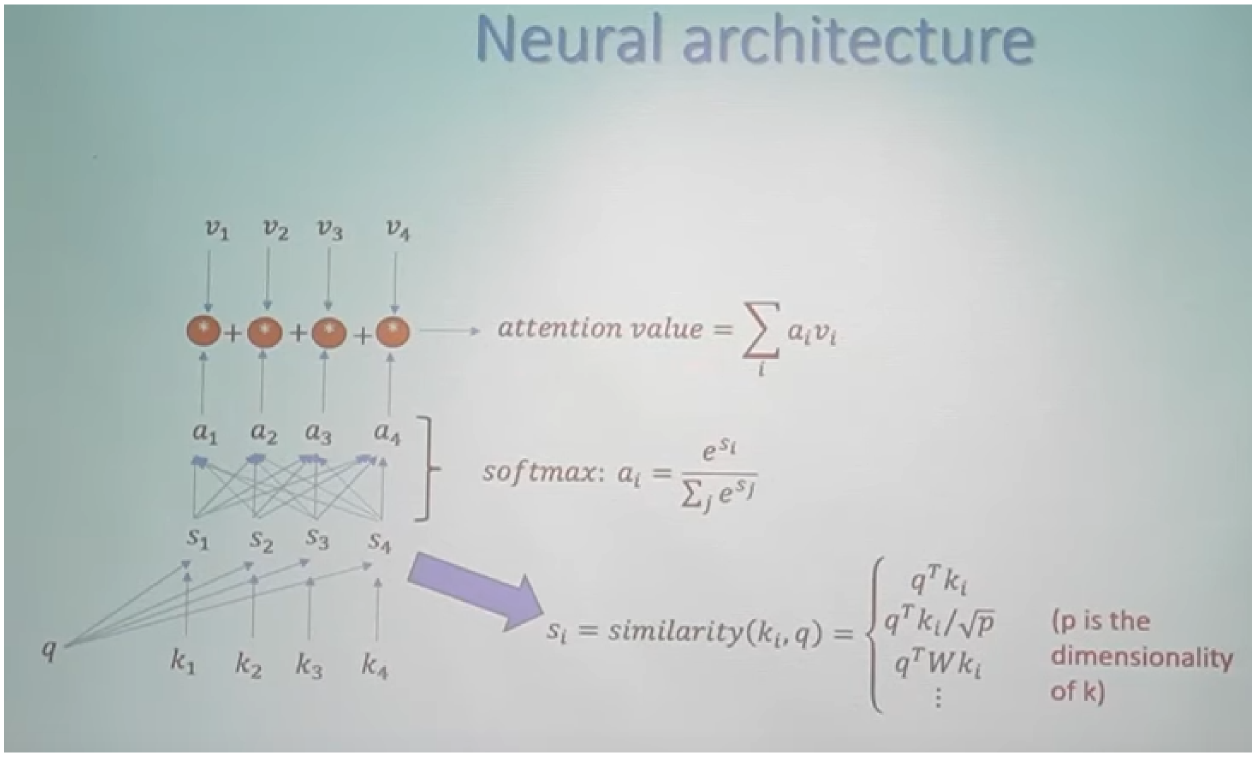


$$\{a_i\}_{i=1}^{n} = \text{softmax} \langle q, \{k_i\}_{i=1}^{n} \rangle \qquad x$$

$$v = \sum_{i=1}^{n} a_i v_i$$

$$q \qquad k \qquad v$$

$$W_q^T \qquad W_k^T \qquad W_v^T$$



Neural architecture

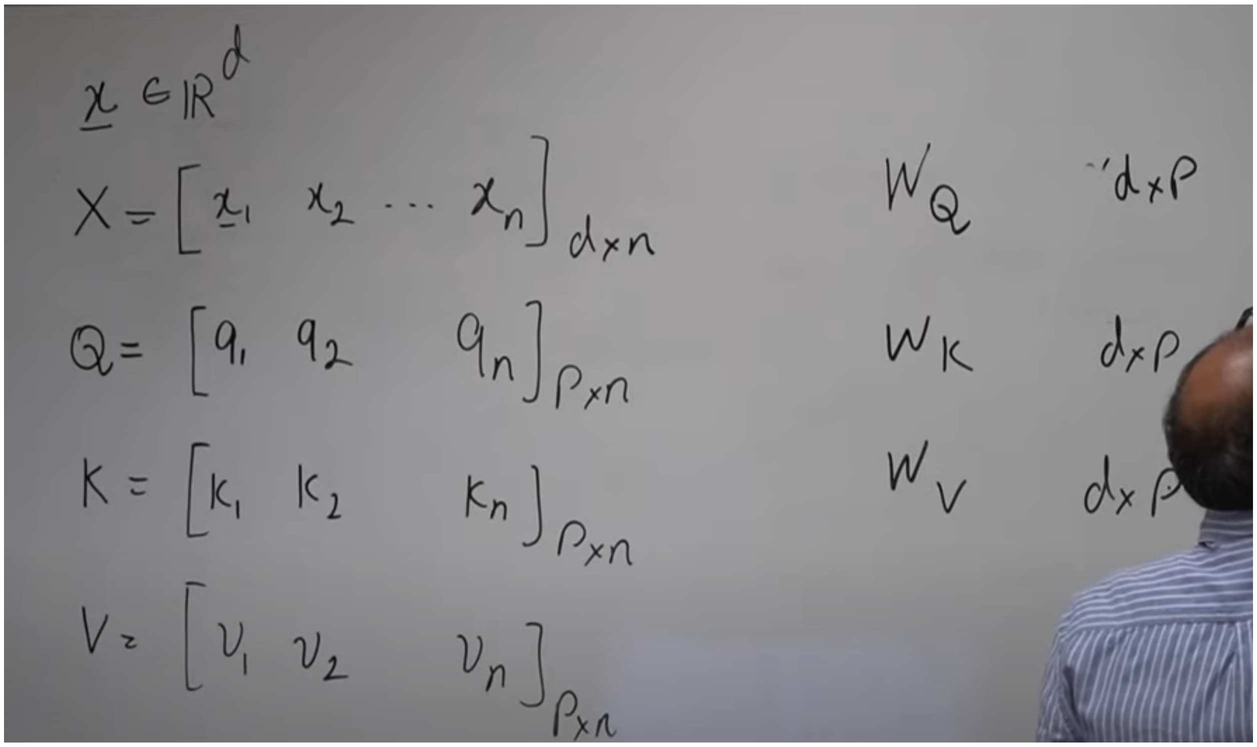
$$attention\ value = \sum_i a_i v_i$$

$$softmax\colon\ a_i = \frac{e^{s_i}}{\sum_j e^{s_j}}$$

$$s_i = similarity(k_i, q) = \begin{cases} q^T k_i \\ q^T k_i / \sqrt{p} \\ q^T W k_i \\ \vdots \end{cases}$$

(p is the dimensionality of k)

The second method is the most commonly used similarity score. Dot product of query and key followed by normalization.



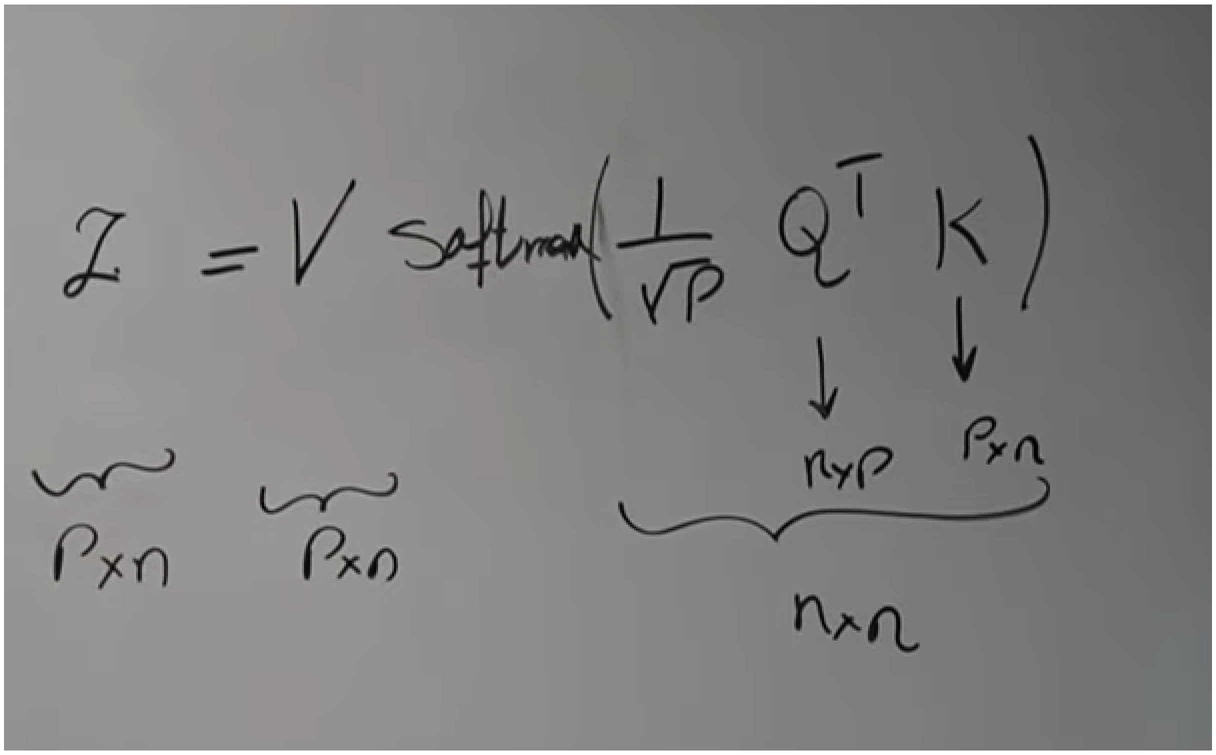
$\underline{x} \in \mathbb{R}^d$

$X = \begin{bmatrix} \underline{x}_1 & x_2 & \cdots & x_n \end{bmatrix}_{d\times n}$ $\qquad W_Q \quad d\times p$

$Q = \begin{bmatrix} q_1 & q_2 & & q_n \end{bmatrix}_{p\times n}$ $\qquad W_K \quad d\times p$

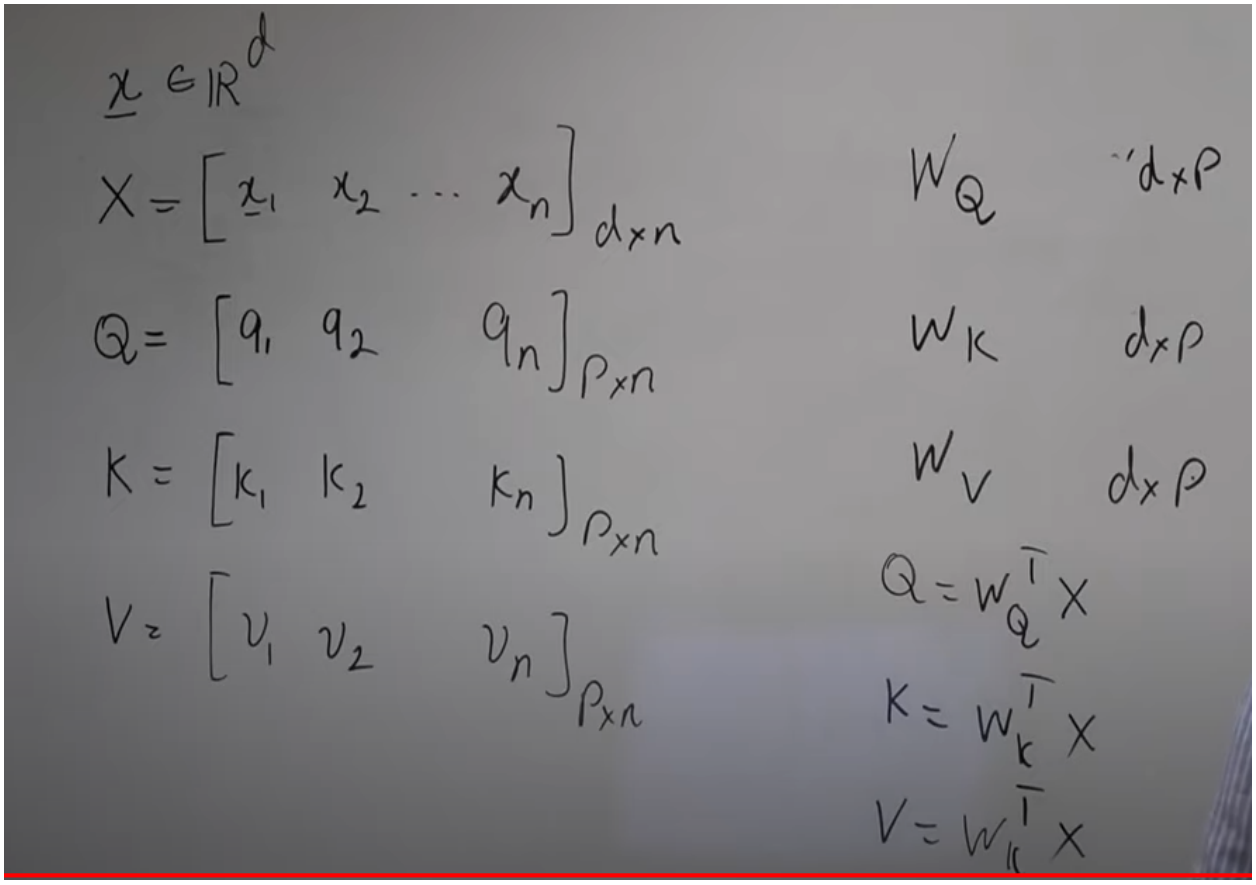
$K = \begin{bmatrix} k_1 & k_2 & & k_n \end{bmatrix}_{p\times n}$ $\qquad W_V \quad d\times p$

$V = \begin{bmatrix} v_1 & v_2 & & v_n \end{bmatrix}_{p\times n}$



$$\underbrace{Z}_{p\times n} = \underbrace{V}_{p\times n}\ \text{softmax}\Big(\underbrace{\frac{1}{\sqrt{p}}\ \underset{\substack{\downarrow \\ n\times p}}{Q^T}\ \underset{\substack{\downarrow \\ p\times n}}{K}}_{n\times n}\Big)$$

Self Attention in matrix form



$\underline{x} \in \mathbb{R}^d$

$X = [\underline{x}_1 \; x_2 \; \dots \; x_n]_{d \times n}$

$Q = [q_1 \; q_2 \; \; q_n]_{p \times n}$

$K = [k_1 \; k_2 \; \; k_n]_{p \times n}$

$V = [v_1 \; v_2 \; \; v_n]_{p \times n}$

$W_Q \quad d \times p$

$W_K \quad d \times p$

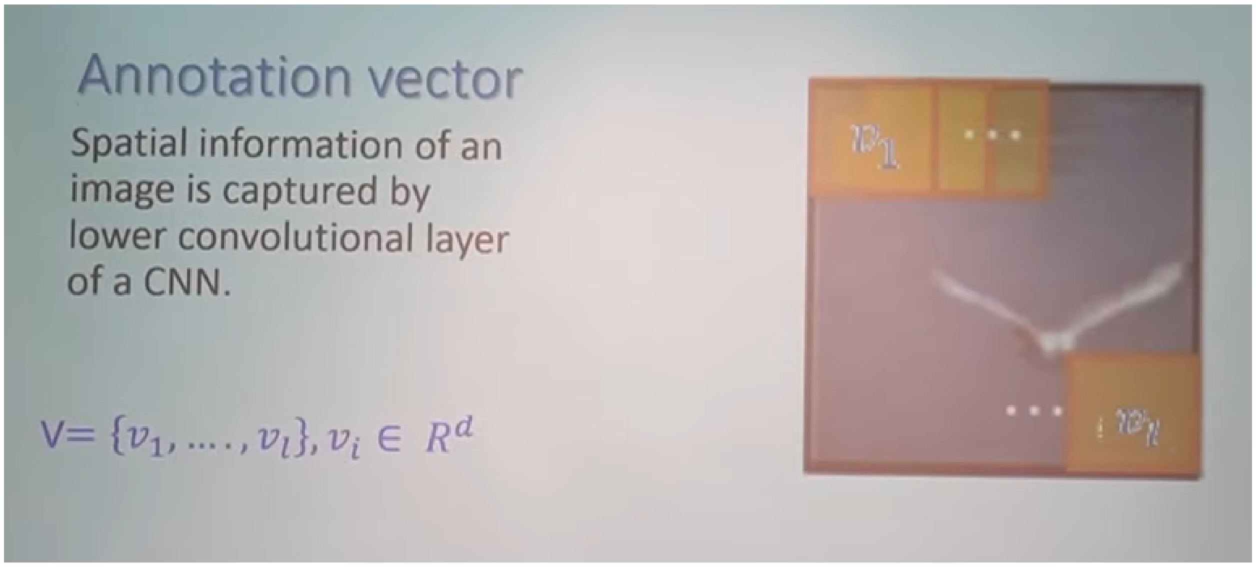
$W_V \quad d \times p$

$Q = W_Q^T X$

$K = W_K^T X$

$V = W_V^T X$

Attention in computer vision



Annotation vector

Spatial information of an image is captured by lower convolutional layer of a CNN.

$V = \{v_1, \dots, v_l\}, v_i \in R^d$

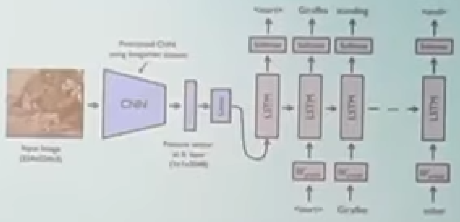
Model

$$v = \{v_1, \dots, v_l\}, v_i \in R^d$$

$$s_{ti} = f_{att}(v_i, h_{t-1})$$

$$a_{ti} = \frac{e^{s_{ti}}}{\sum_j e^{s_{tj}}}$$

$$z_t = \Phi(\{v_i\}, \{a_i\})$$


CNN
LSTM
LSTM
LSTM
LSTM
Giraffes
standing
Giraffes