Natural Language Processing Series

[1]

Transformers

Prof. Khaled Mostafa El-Sayed
khaledms@fci-cu.edu.eg

Faculty of Computers and Artificial Intelligence
Cairo University

Aug 2020

Transformers

Processing Sequential Data

Transformer Building Blocks

Self Attention Mechanism

Multi-Head Attention

Parallelization Implementation Issues

Positional Information

Building Transformer Block

Transformer Decoder

Transformer Encoder as a Classifier

What is Next ?

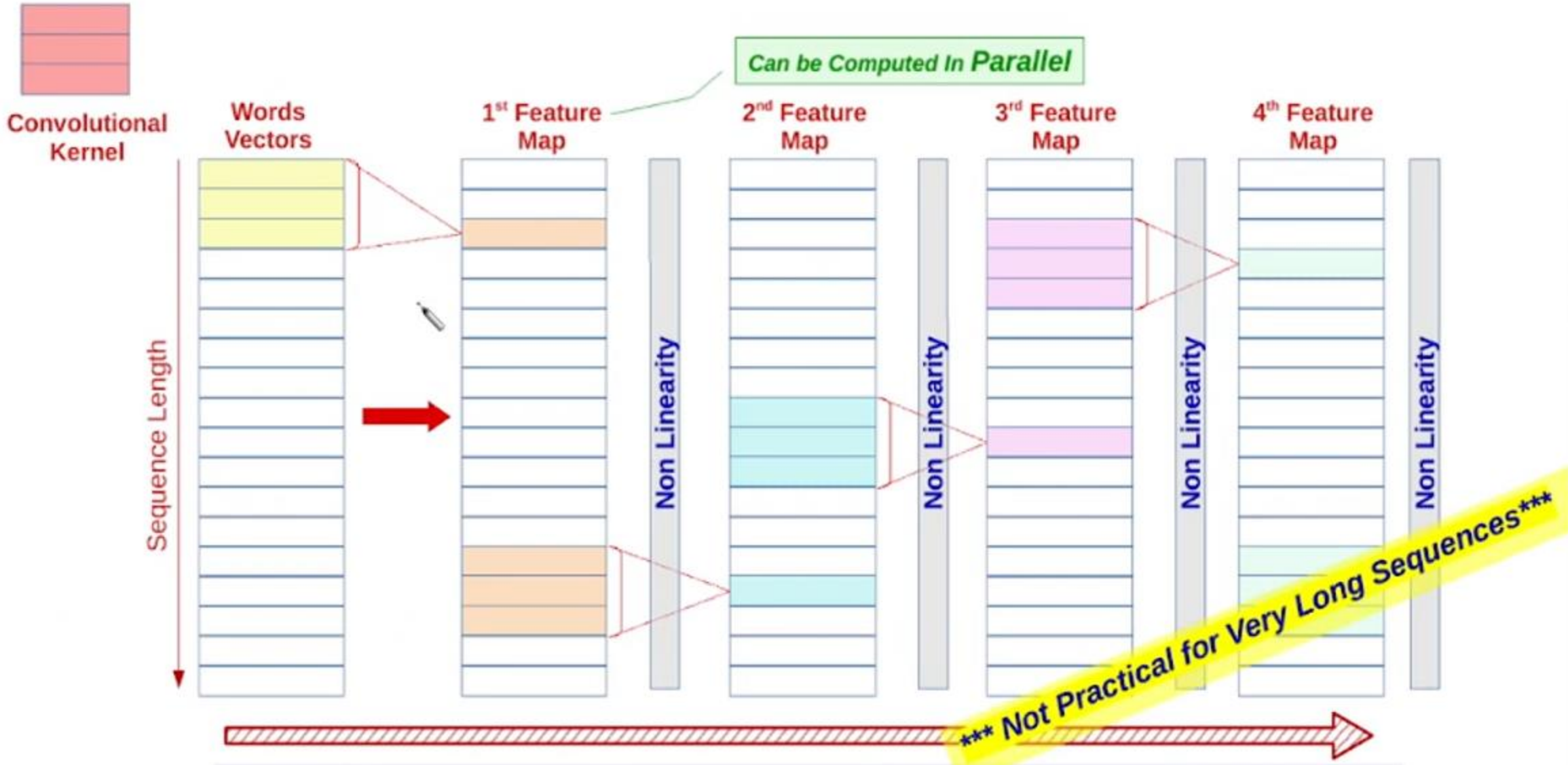
Processing Sequential Data

1-D Convolutional Neural Network (1-D CNN)

Recurrent Neural Network (RNN)

1-D Convolutional Neural Network

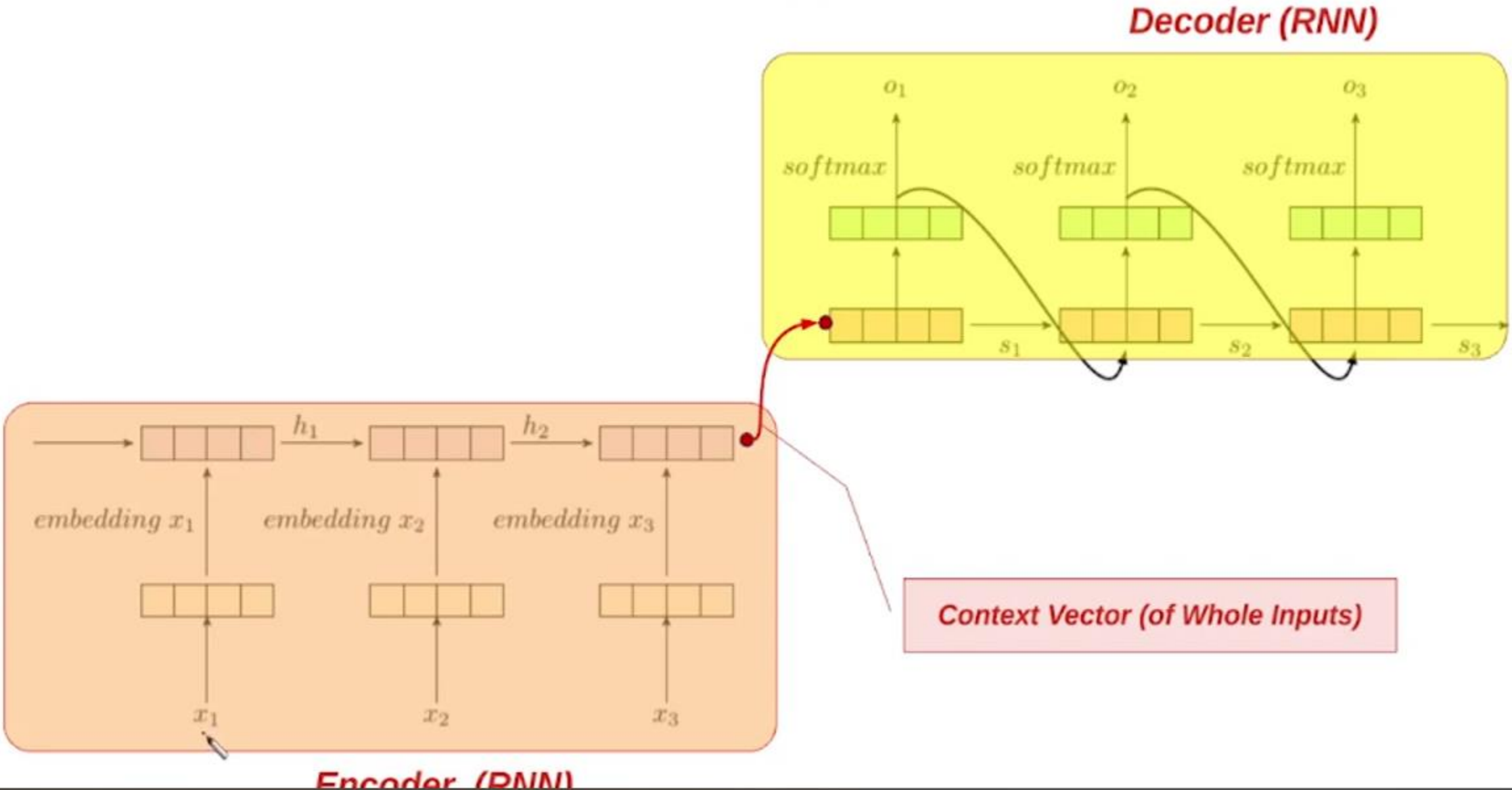
[4]



As Word Sequence Length Increases, **we need to stack many convolution Layers.**

Traditional RNN Seq2Seq model

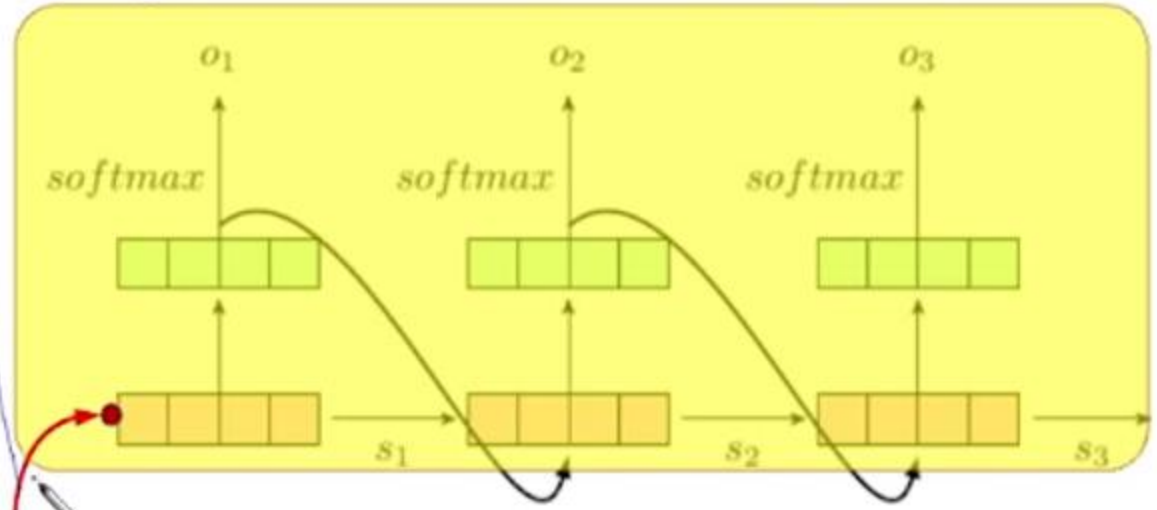
[5]



Traditional RNN Seq2Seq model (Challenges)

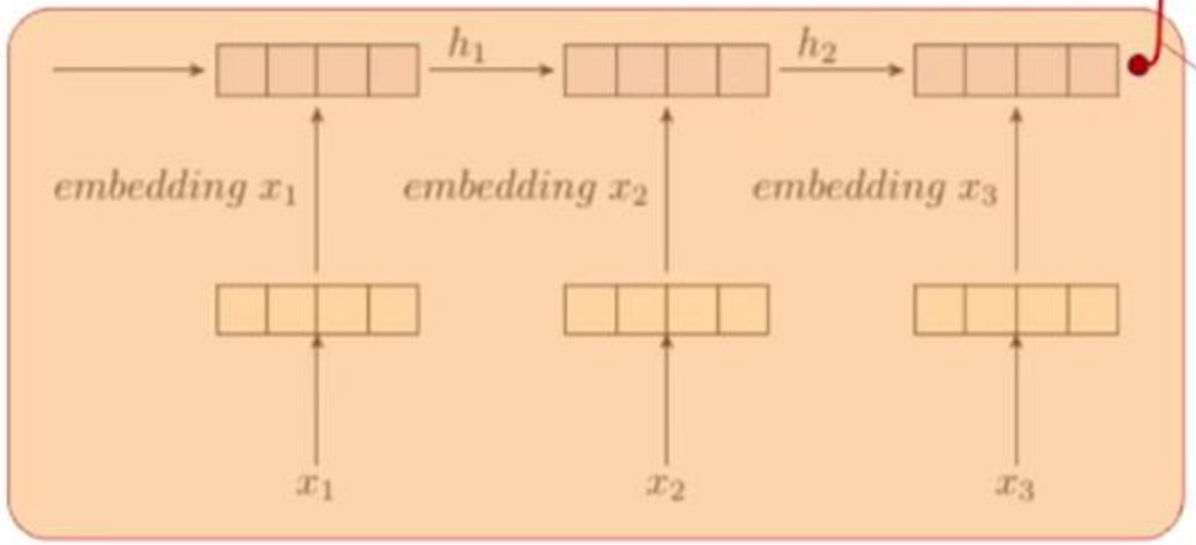
[6]

Decoder (RNN)



[1] Bottleneck

The meaning of the entire input sequence is Expected to be captured by a single context vector with fixed dimensionality

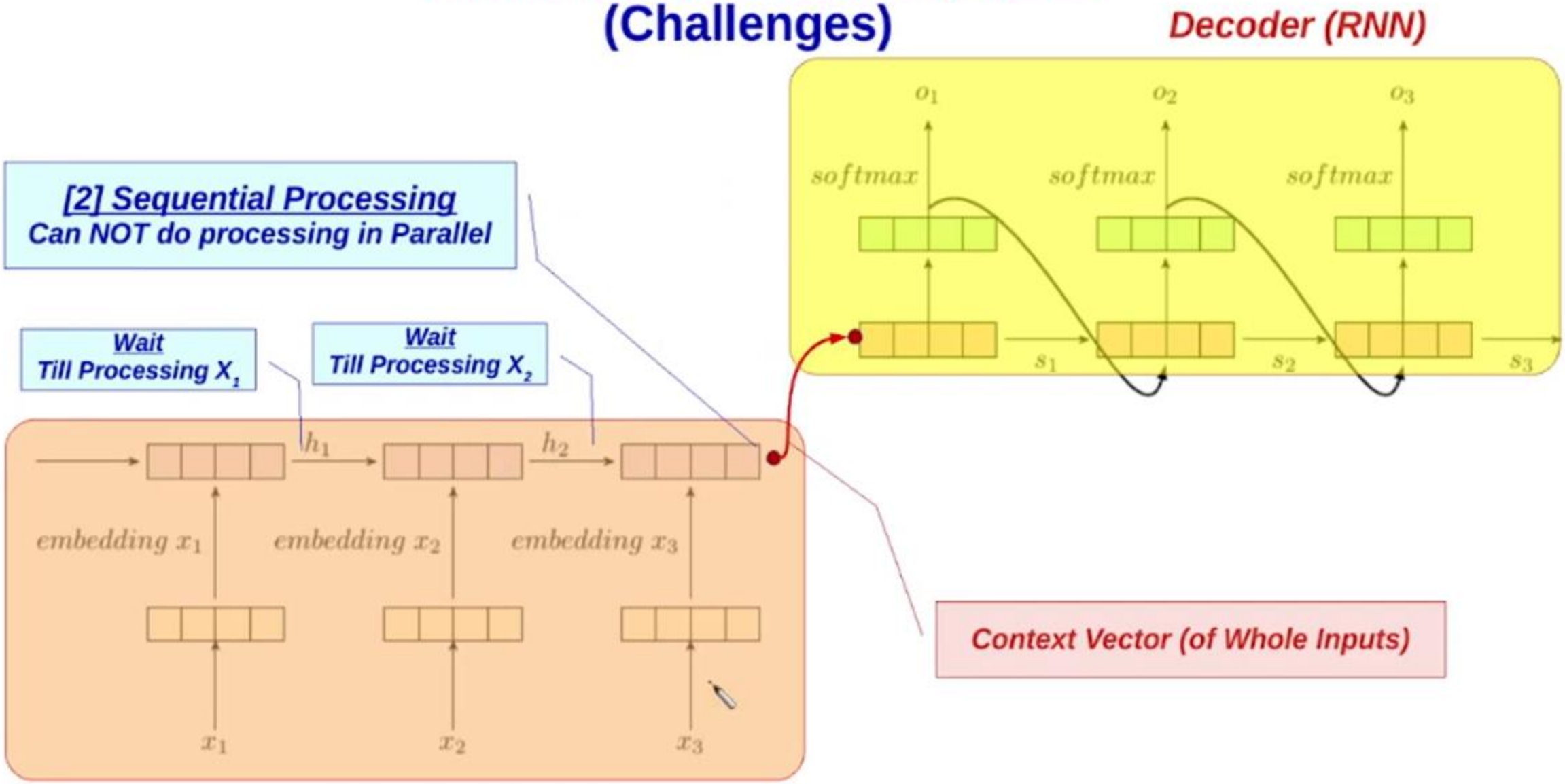


Context Vector (of Whole Inputs)

Encoder (RNN)

Traditional RNN Seq2Seq model (Challenges)

[7]



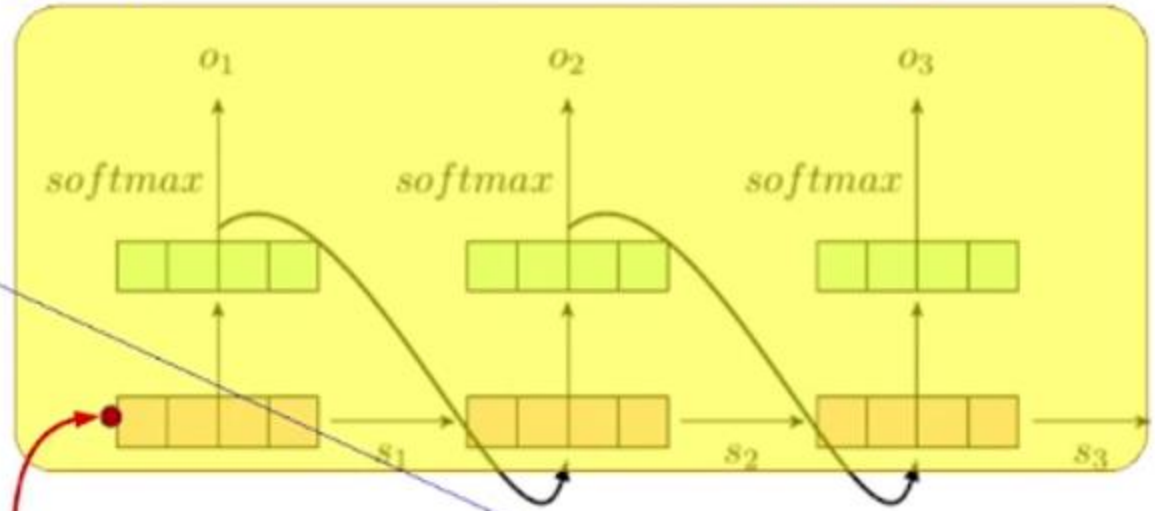
Traditional RNN Seq2Seq model (Challenges)

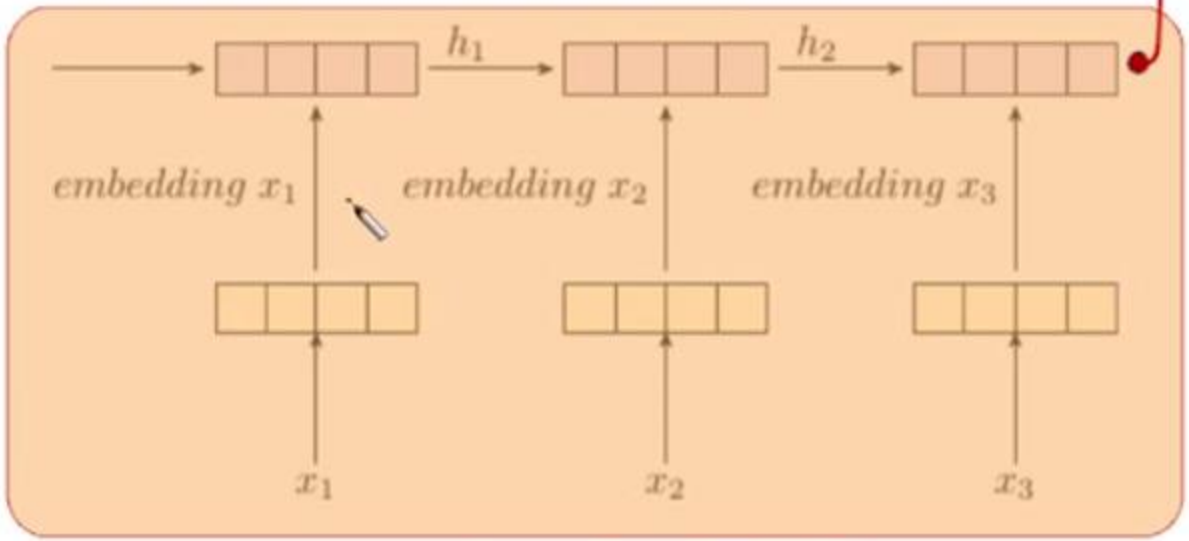
[8]

[3] Very Long Sequence
Vanishing Gradient

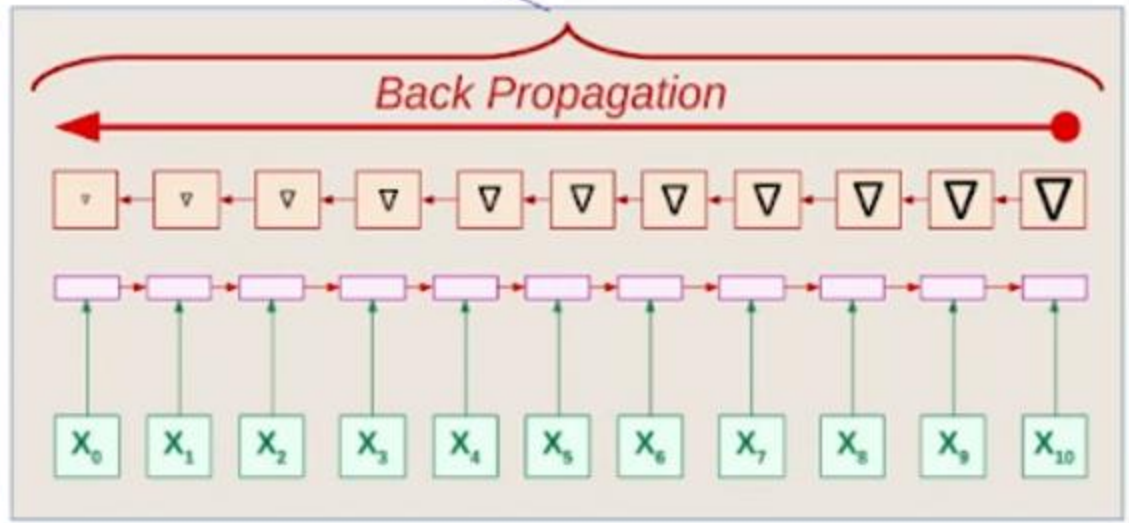
Short Sequence: RNN
Long Sequence: LSTM, GRU
Very Long Sequence: FAIL

Decoder (RNN)





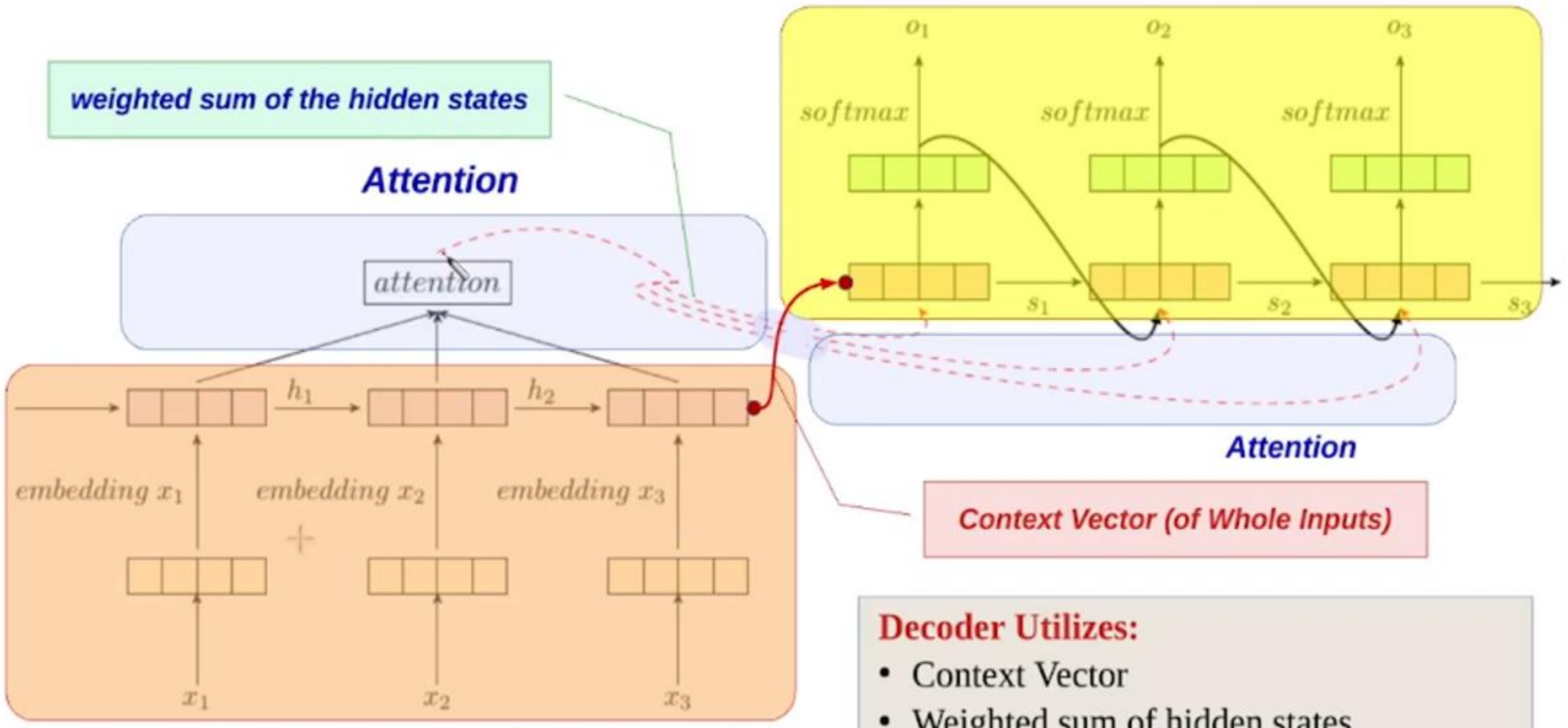
Encoder (RNN)



RNN Seq2Seq model [with Attention]

[To overcome Bottleneck challenge]

[9]



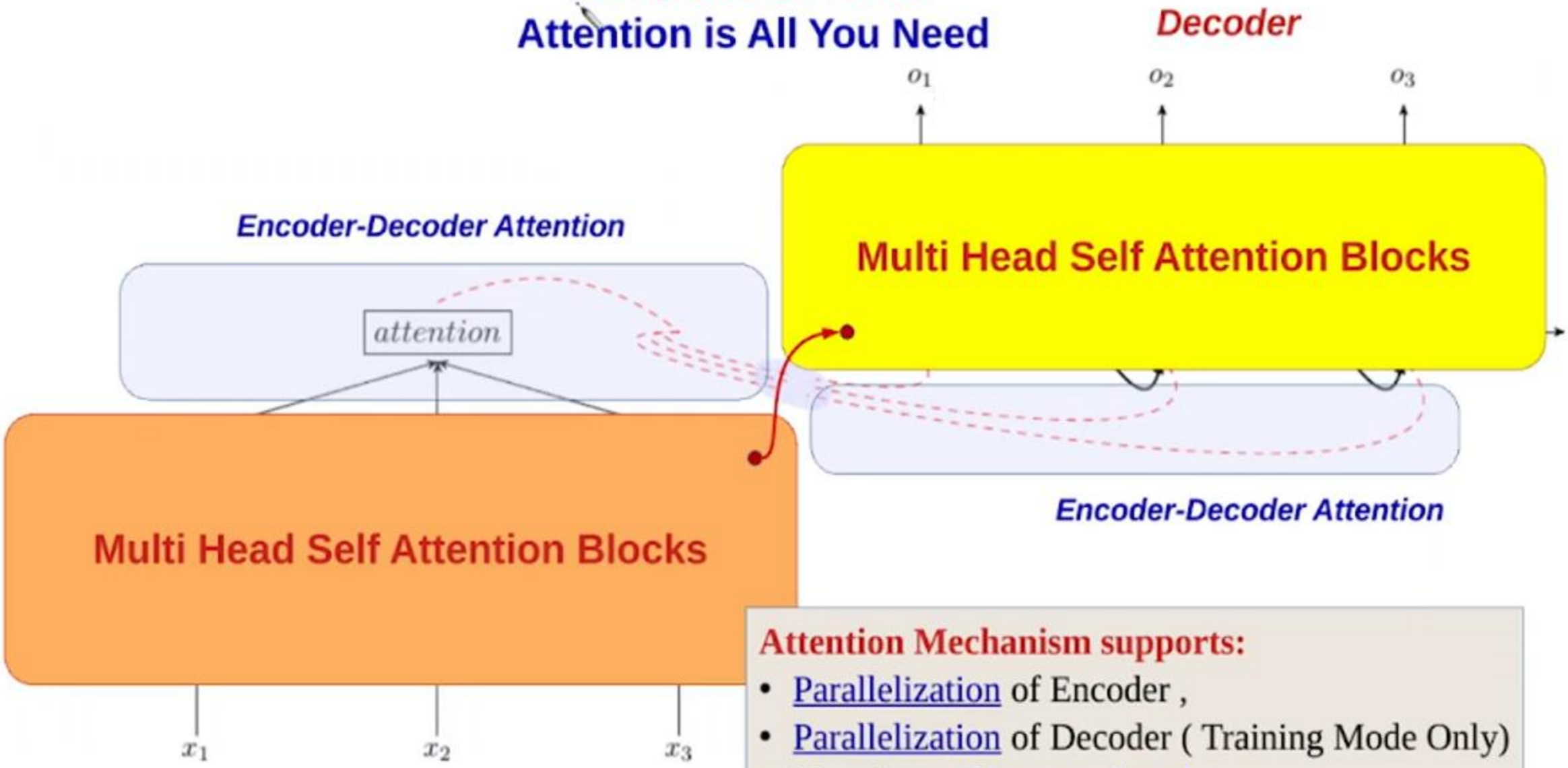
Decoder Utilizes:

- Context Vector
- Weighted sum of hidden states

[10]

Transformer

Attention is All You Need



Attention Mechanism supports:

- Parallelization of Encoder ,
- Parallelization of Decoder (Training Mode Only)
- Very Long Sequence length

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com

Noam Shazeer* Google Brain noam@google.com

Niki Parmar* Google Research nikip@google.com

Jakob Uszkoreit* Google Research usz@google.com

Llion Jones* Google Research llion@google.com

Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu

Łukasz Kaiser* Google Brain lukaszkaiser@google.com

Illia Polosukhin* ‡ illia.polosukhin@gmail.com

Abstract

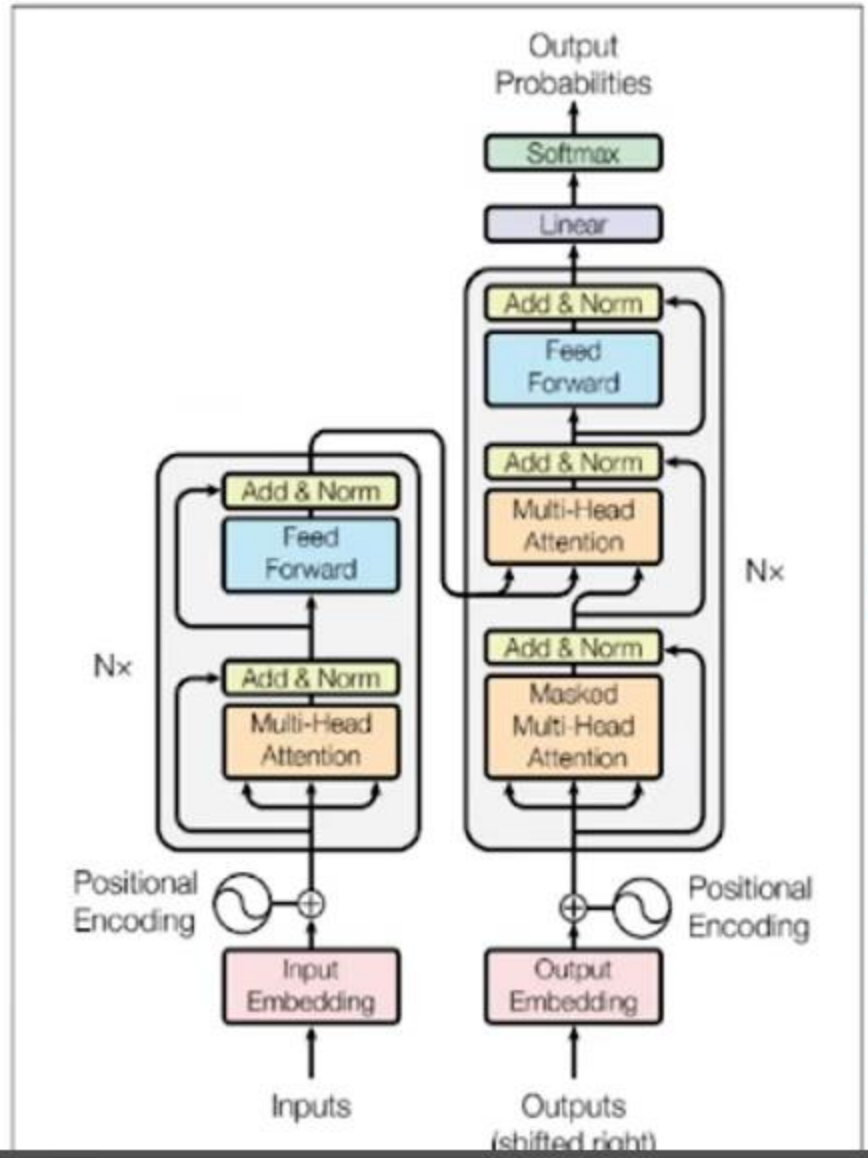
The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.8 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the

[12]

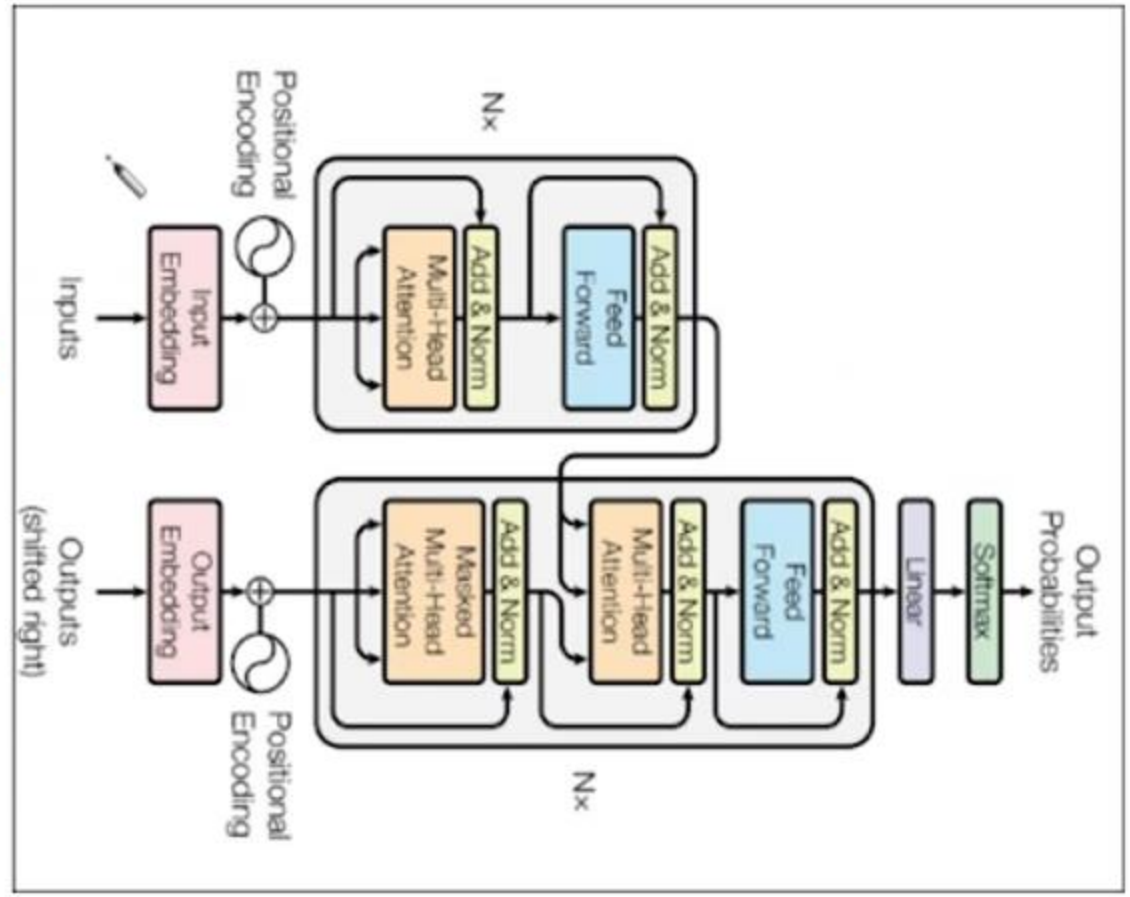
Transformer Building Blocks

Encoder – Decoder Transformer

[13]



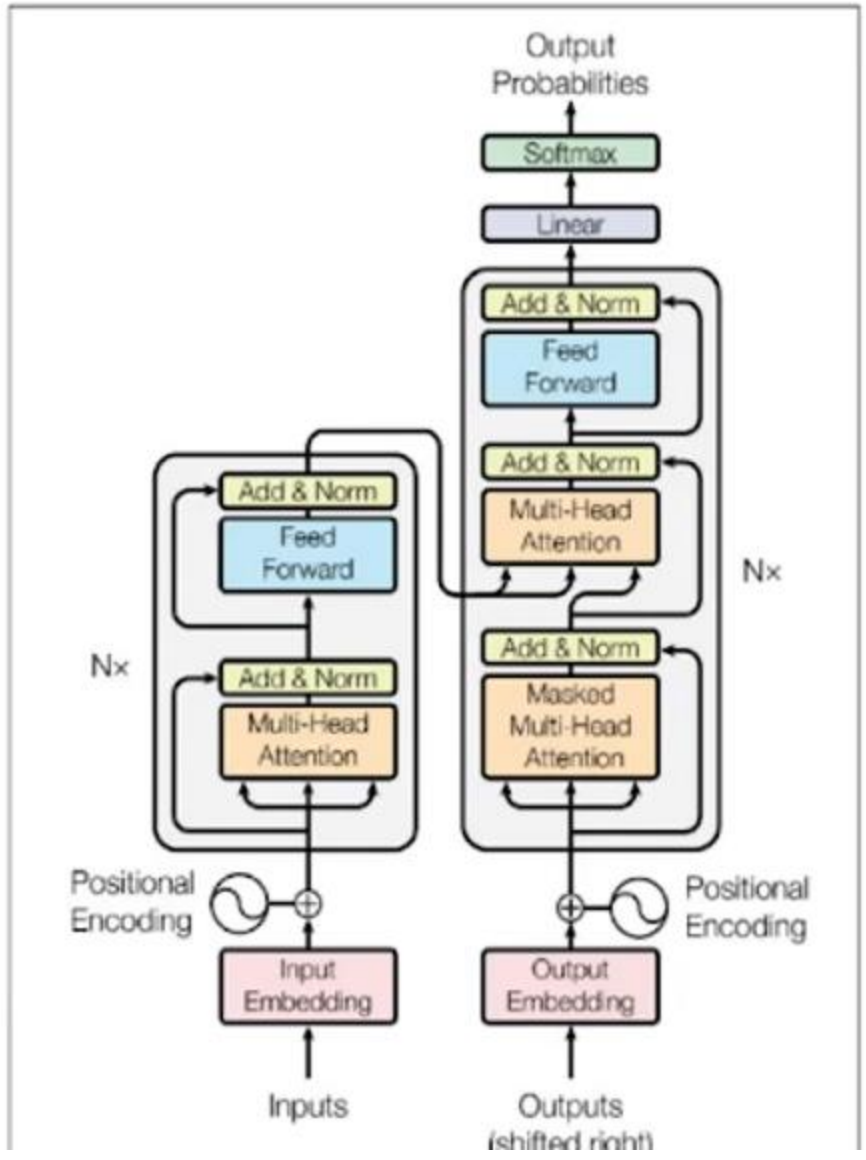
Orientation is not the Issue

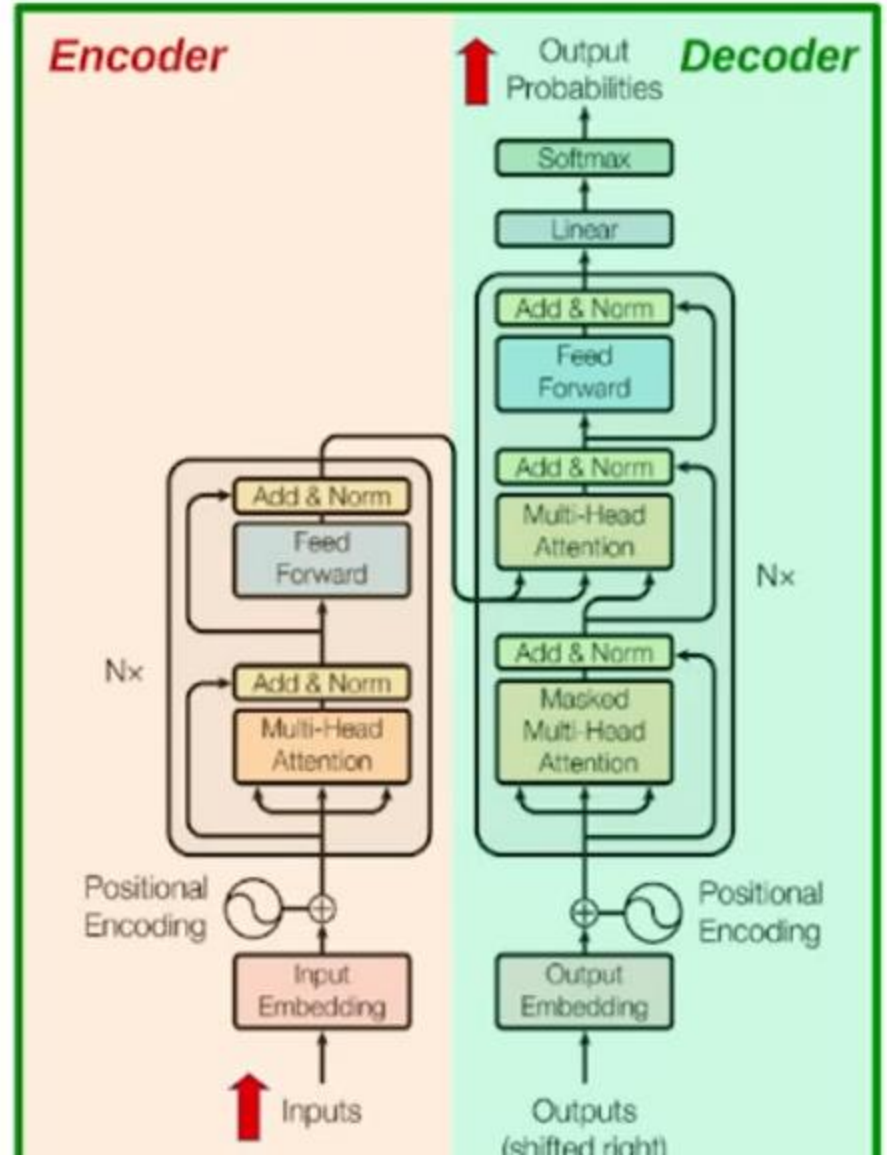


Let us focus on the Building blocks

Encoder – Decoder Transformer

[14]





Encoder – Decoder Transformer

[15]

Decoder Block

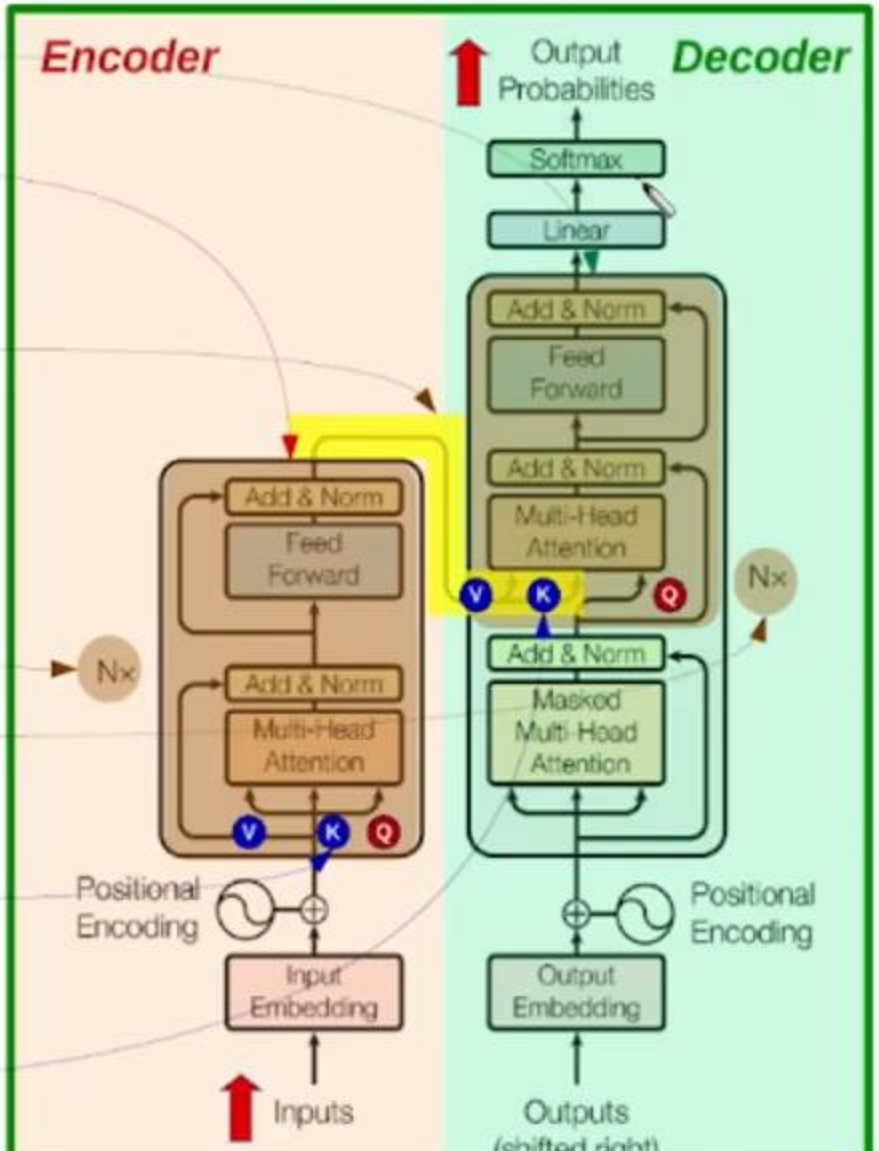
Encoder Block

Encoder Decoder Attention
From Last Encoder Block
To ALL Decoder Cascaded Blocks

N Cascaded Blocks
In Both Encoder and Decoder

Inputs to Encoder Block (V,K,Q)
All from Previous Block

Inputs to Decoder Block (V,K,Q)
(V,K) from Attention, (Q) from Previous Block



Encoder – Decoder Transformer

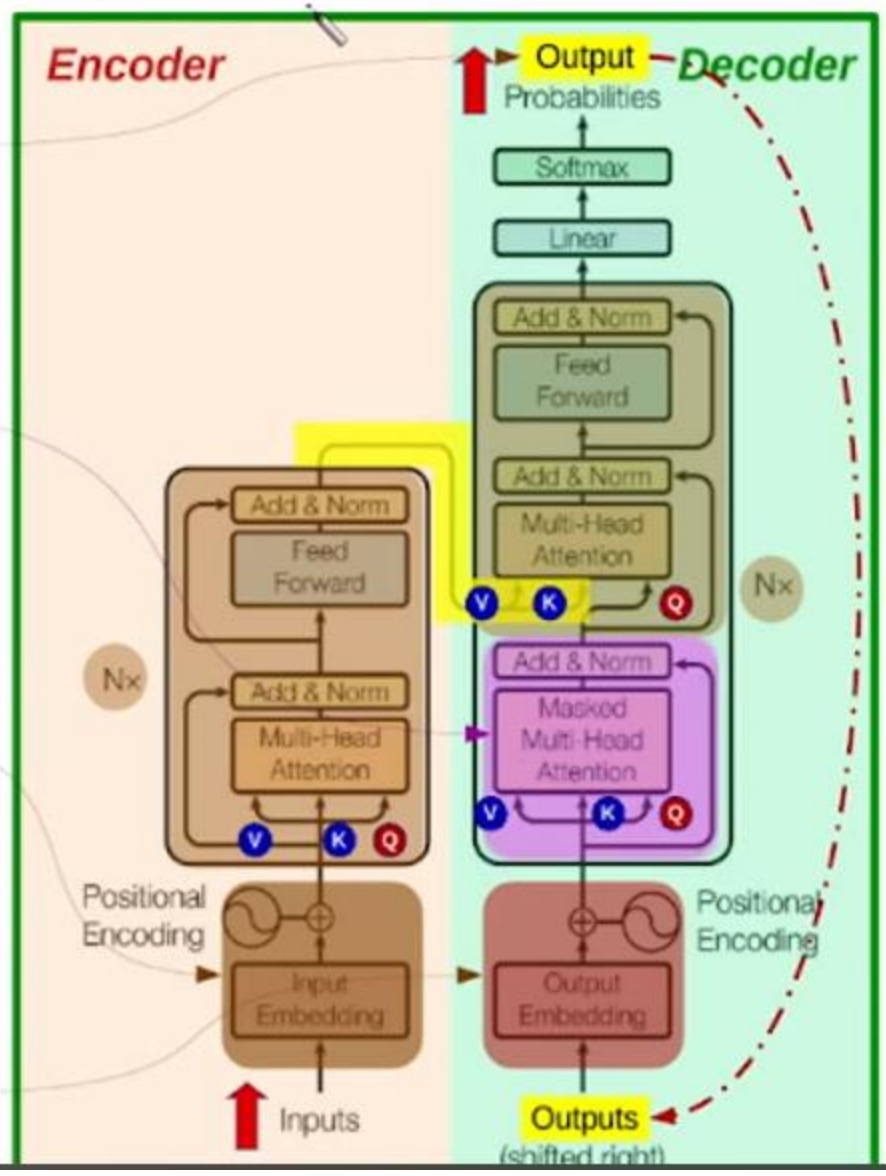
[16]

Decoder Output (at step "t-1")
Is supplied as Decoder Input (at step "t")

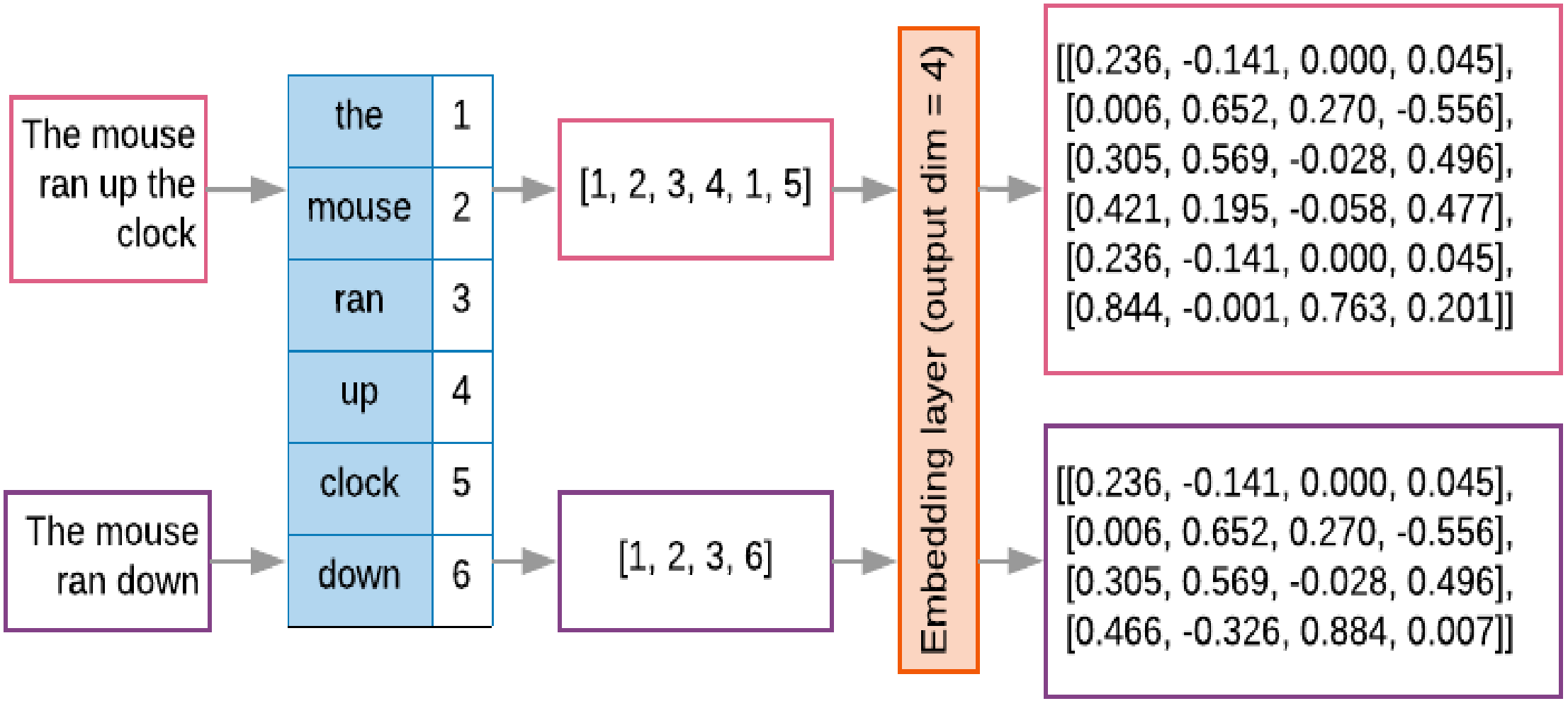
First Decoder Block
No Input from Encoder Attention

Input Positional Encoder
(Position of Input Word within the Input Sequence)

Output Positional Encoder
(Position of Output Word within the Output Sequence)



Token and embedding



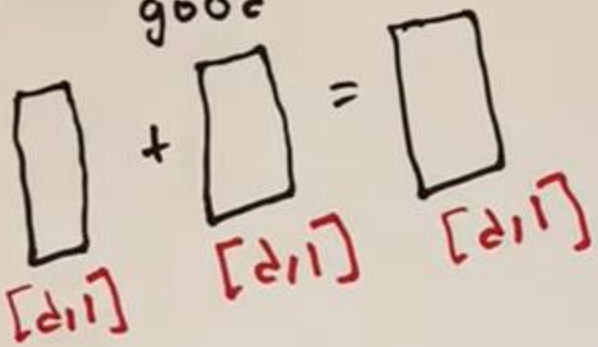
d=4

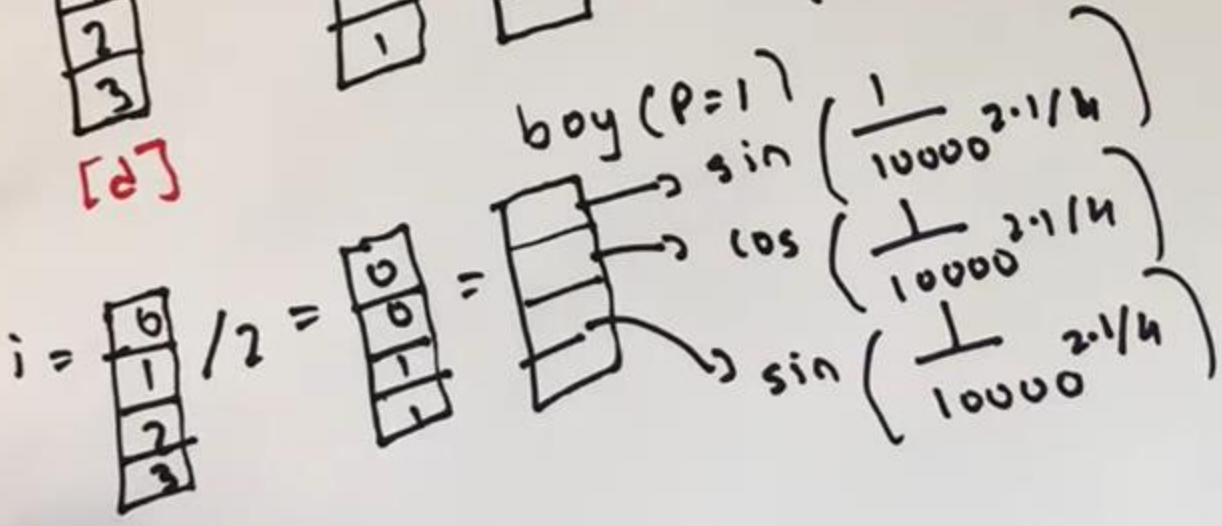
good boy
p=0 p=1

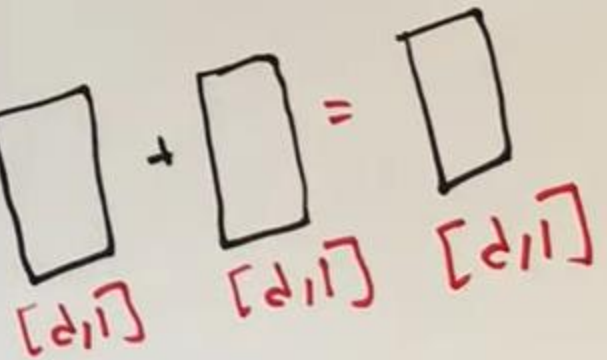
$PE(p,2i) = \sin\left(\frac{p}{10000^{2i/d}}\right)$

$PE(p,2i+1) = \cos\left(\frac{p}{10000^{2i/d}}\right)$

good





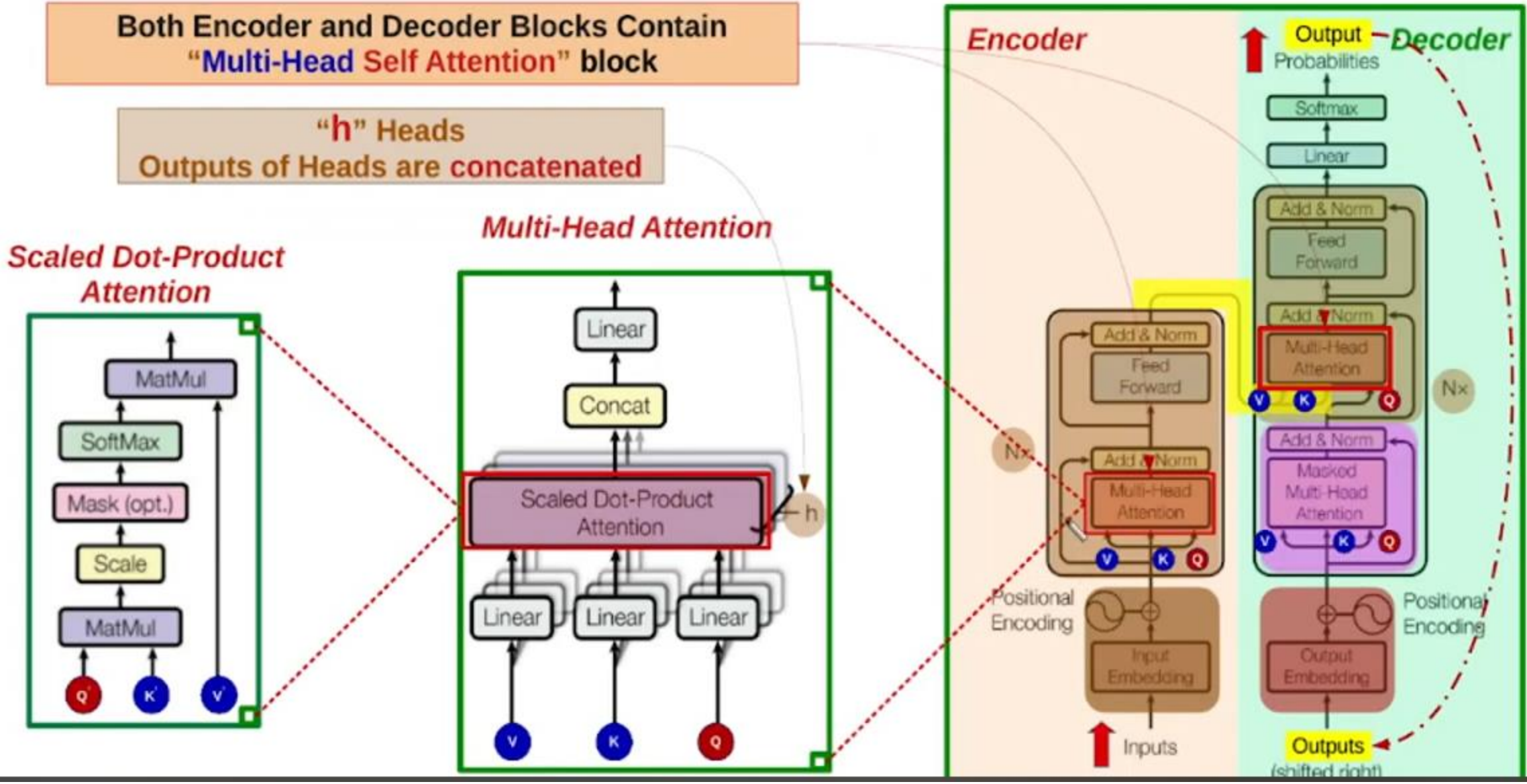


Encoder – Decoder Transformer

[17]

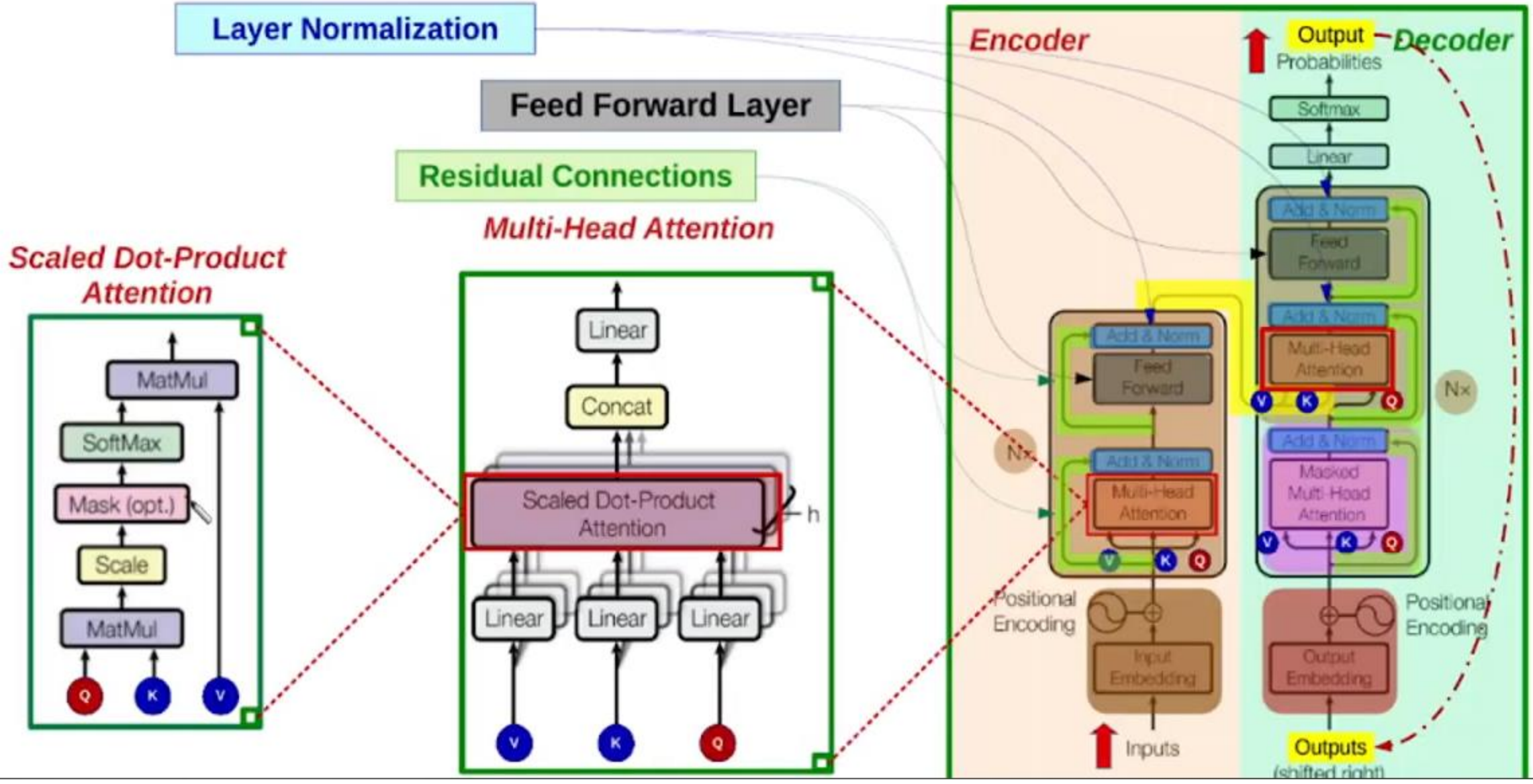
Both Encoder and Decoder Blocks Contain "Multi-Head Self Attention" block

"h" Heads
Outputs of Heads are concatenated



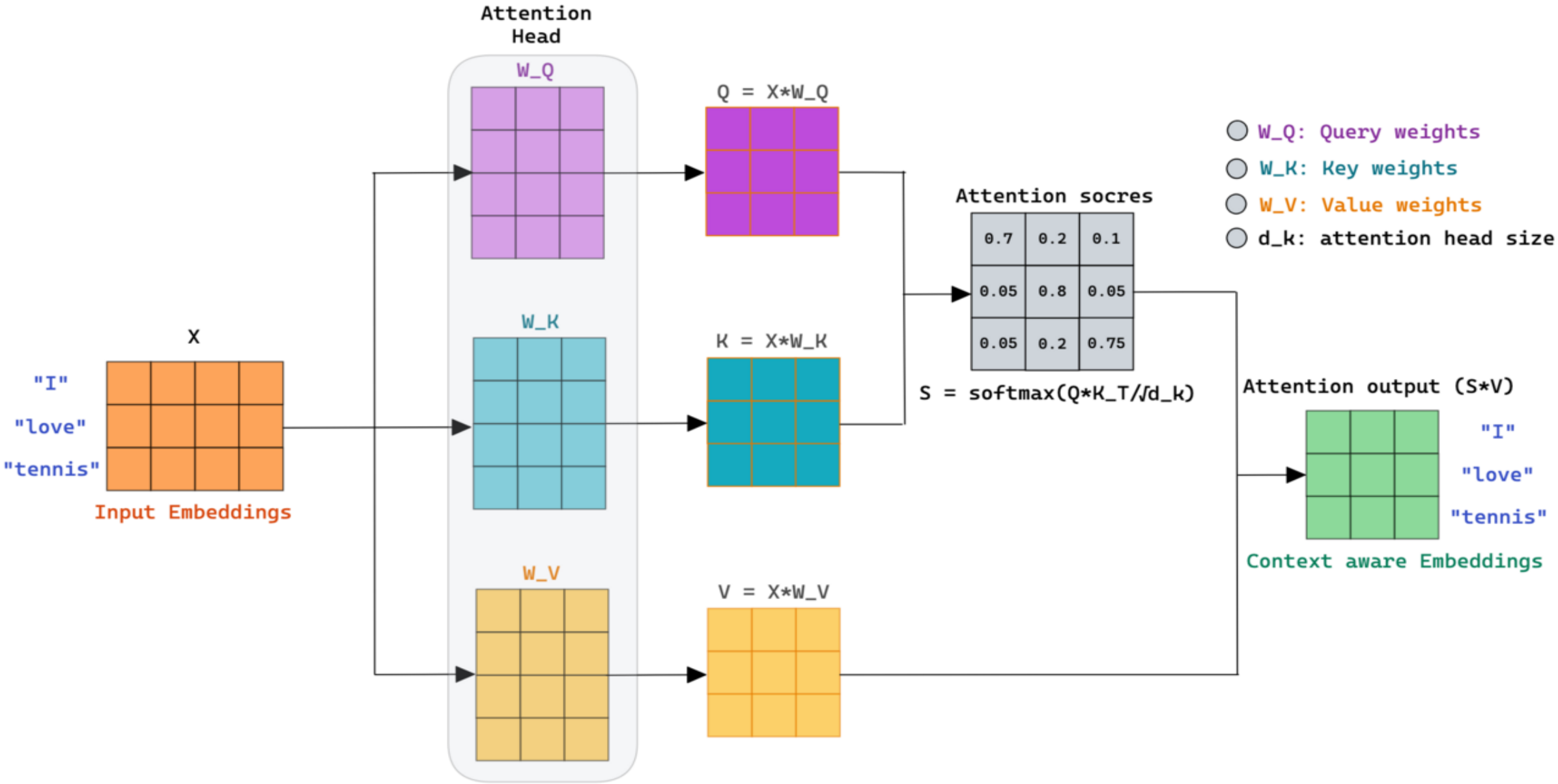
Encoder – Decoder Transformer

[18]



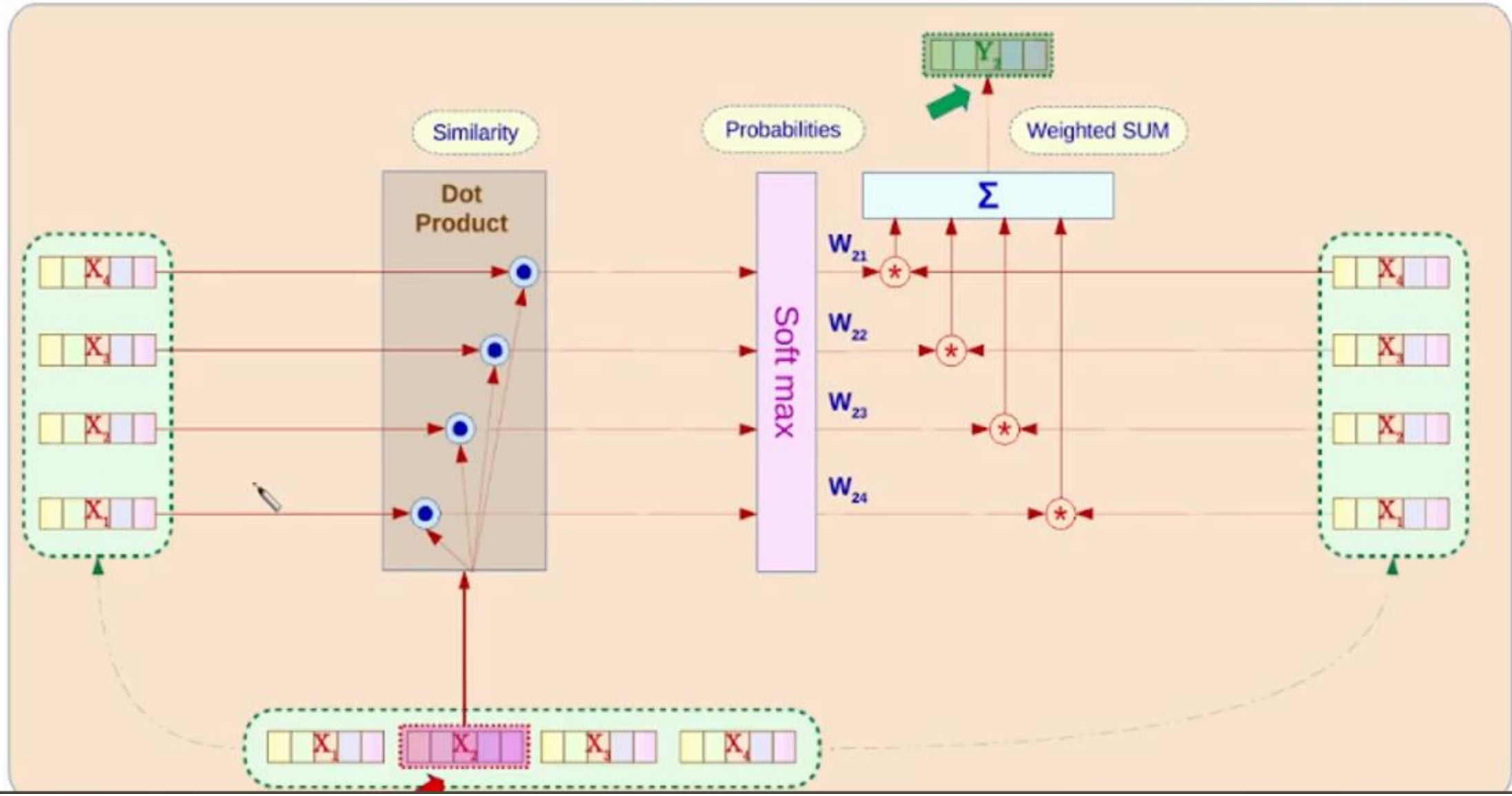
Self Attention Mechanism

Self-attention

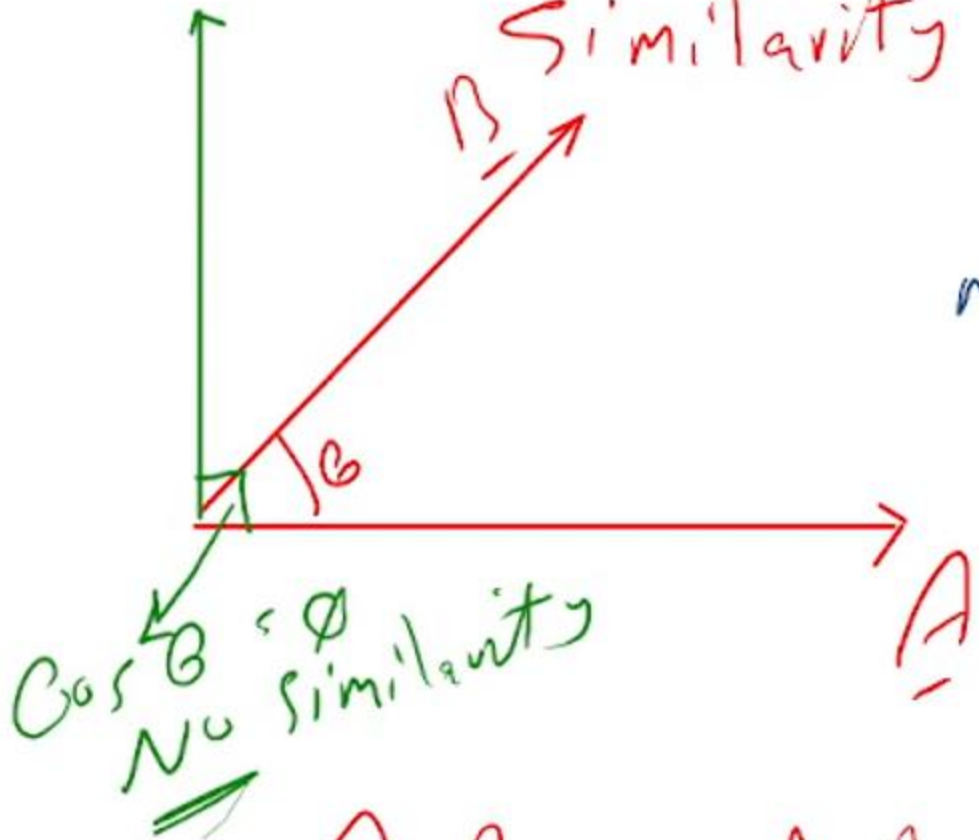


Self Attention Mechanism (Fundamental Operation)

(Self Attention of word X_2 w.r.t All words in Sentence (X_1, X_2, X_3, X_4)

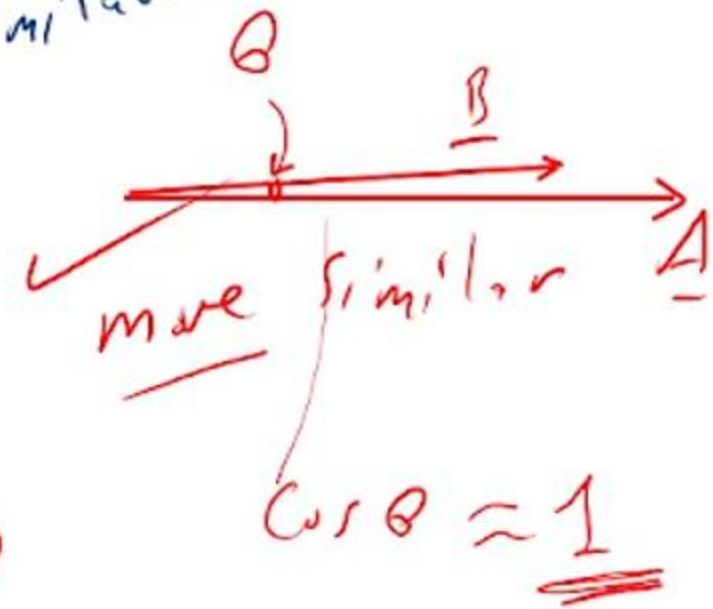


Similarity of vectors



$$\cos\theta = \frac{\underline{A} \cdot \underline{B}}{|A||B|}$$

measure of similarity



$$\underline{A} \cdot \underline{B} = AB\cos\theta$$

Dot product

$\cos\theta \ll$

COS Similarity

$X_2 \longrightarrow Y_2 \Rightarrow$ related to context words (around X_2)

$Y_2 = \underbrace{w_1 X_1 + w_2 X_2 + w_3 X_3 + w_4 X_4}$ weighted sum

similarity of $X_1, X_2 \longrightarrow w_1$

" $X_3, X_2 \longrightarrow w_3$

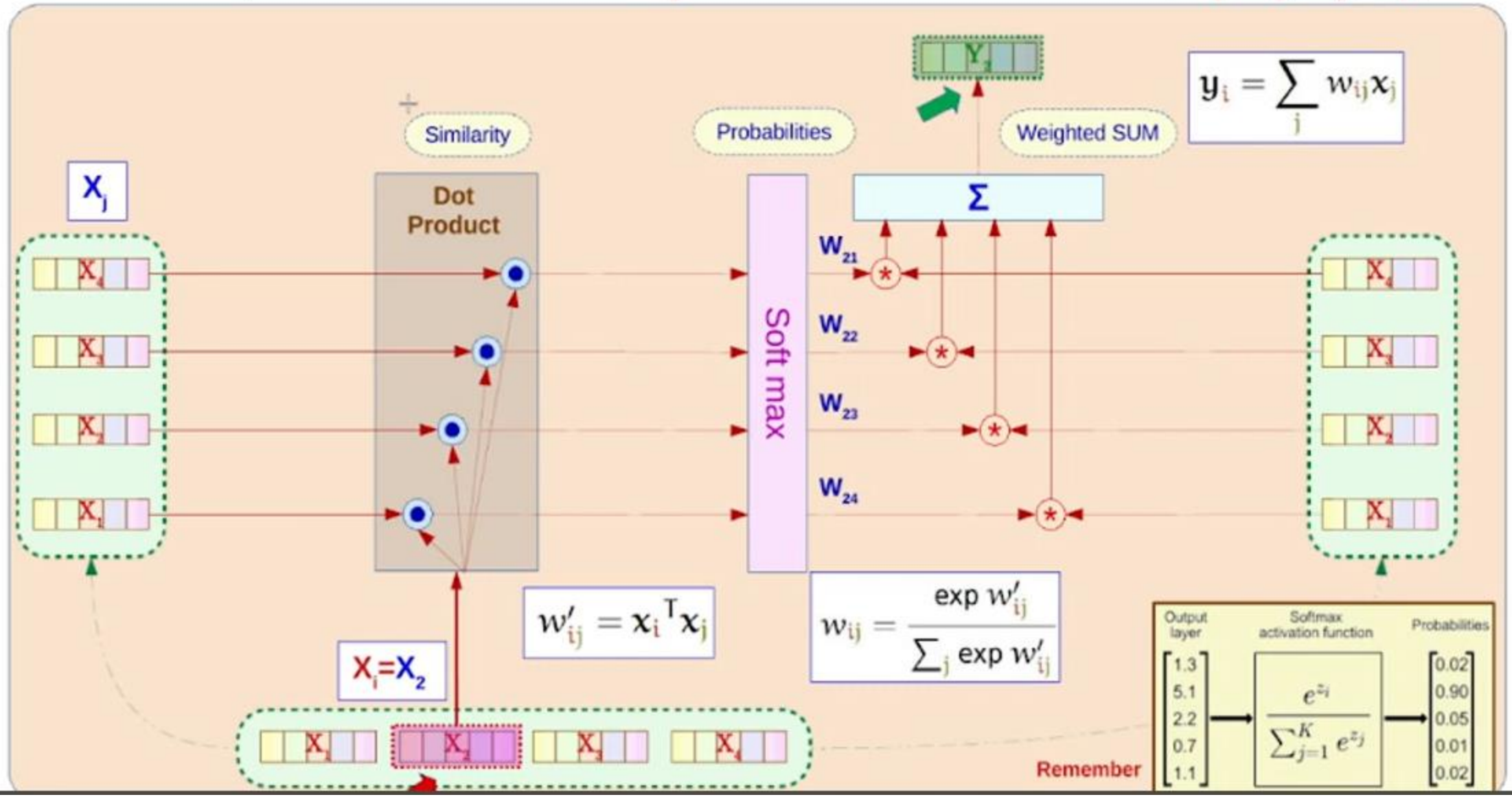
" $X_4, X_2 \longrightarrow w_4$

" $X_2, X_2 \longrightarrow w_2$

sum = 1

Self Attention Mechanism (Fundamental Operation)

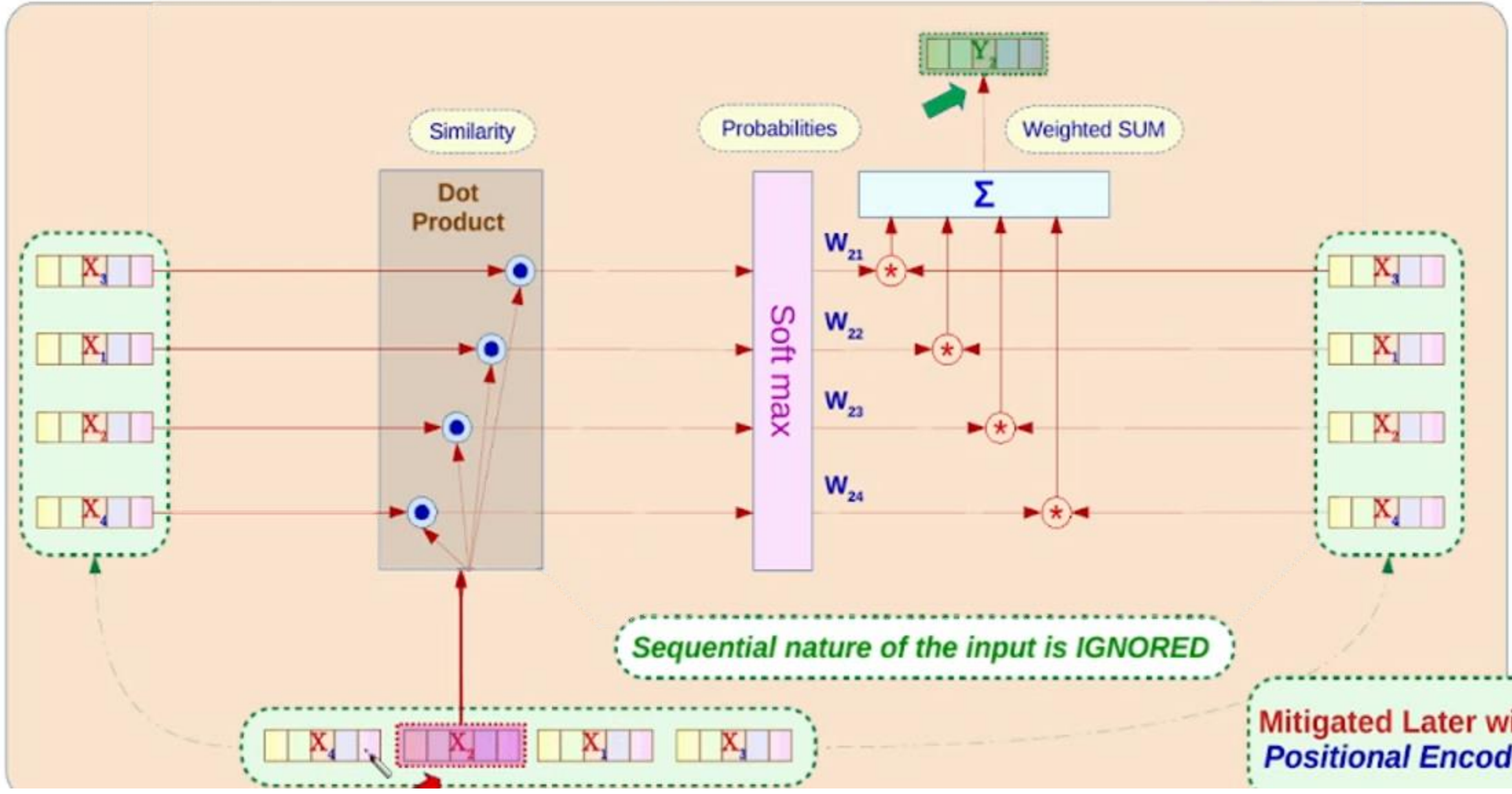
(Self Attention of word X_2 w.r.t All words in Sentence (X_1, X_2, X_3, X_4)



[23]

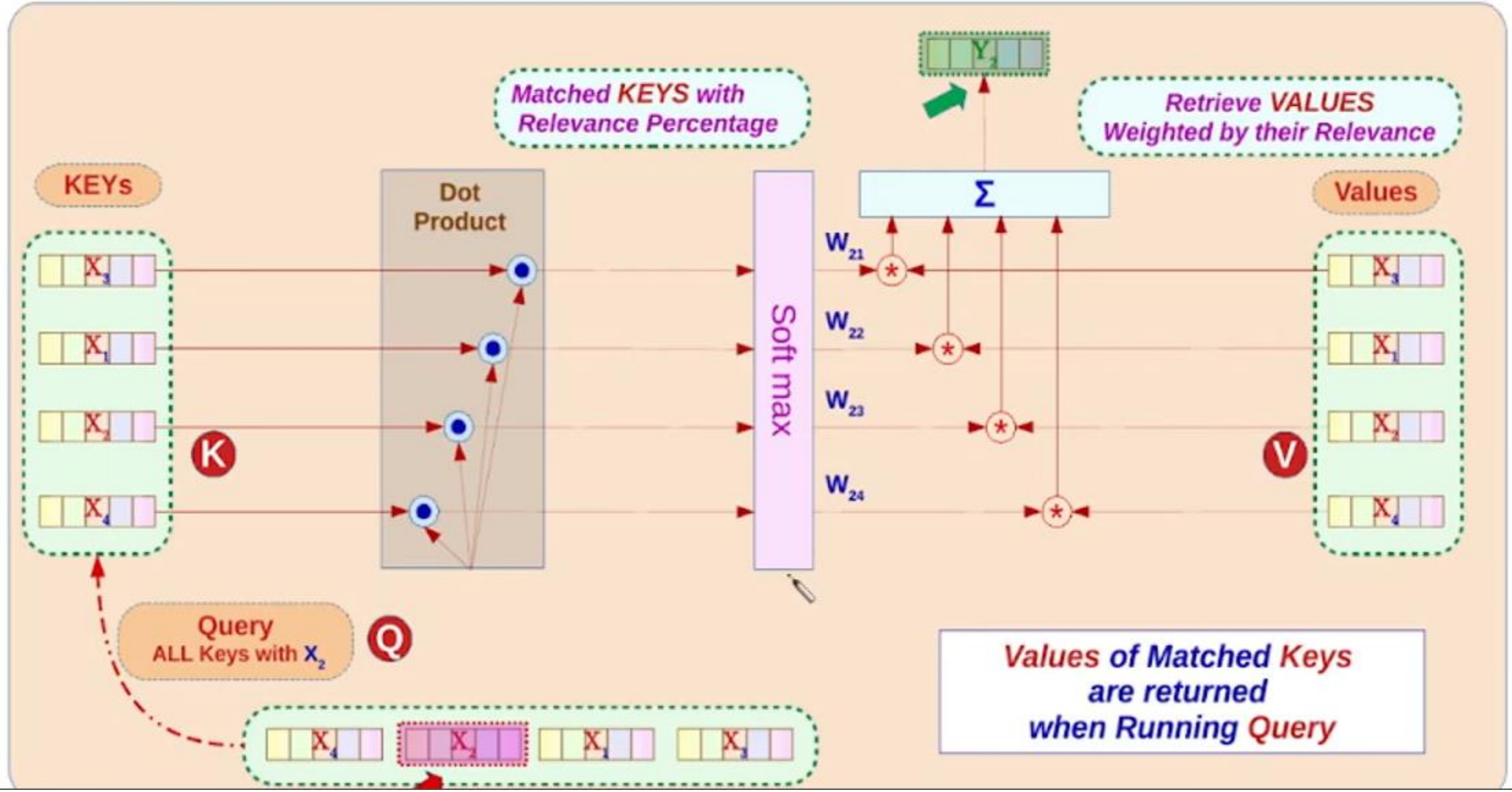
Self Attention Mechanism (Not Sensitive to Words Order)

(Self Attention of word X_2 w.r.t All words in Sentence (X_4, X_2, X_1, X_3)

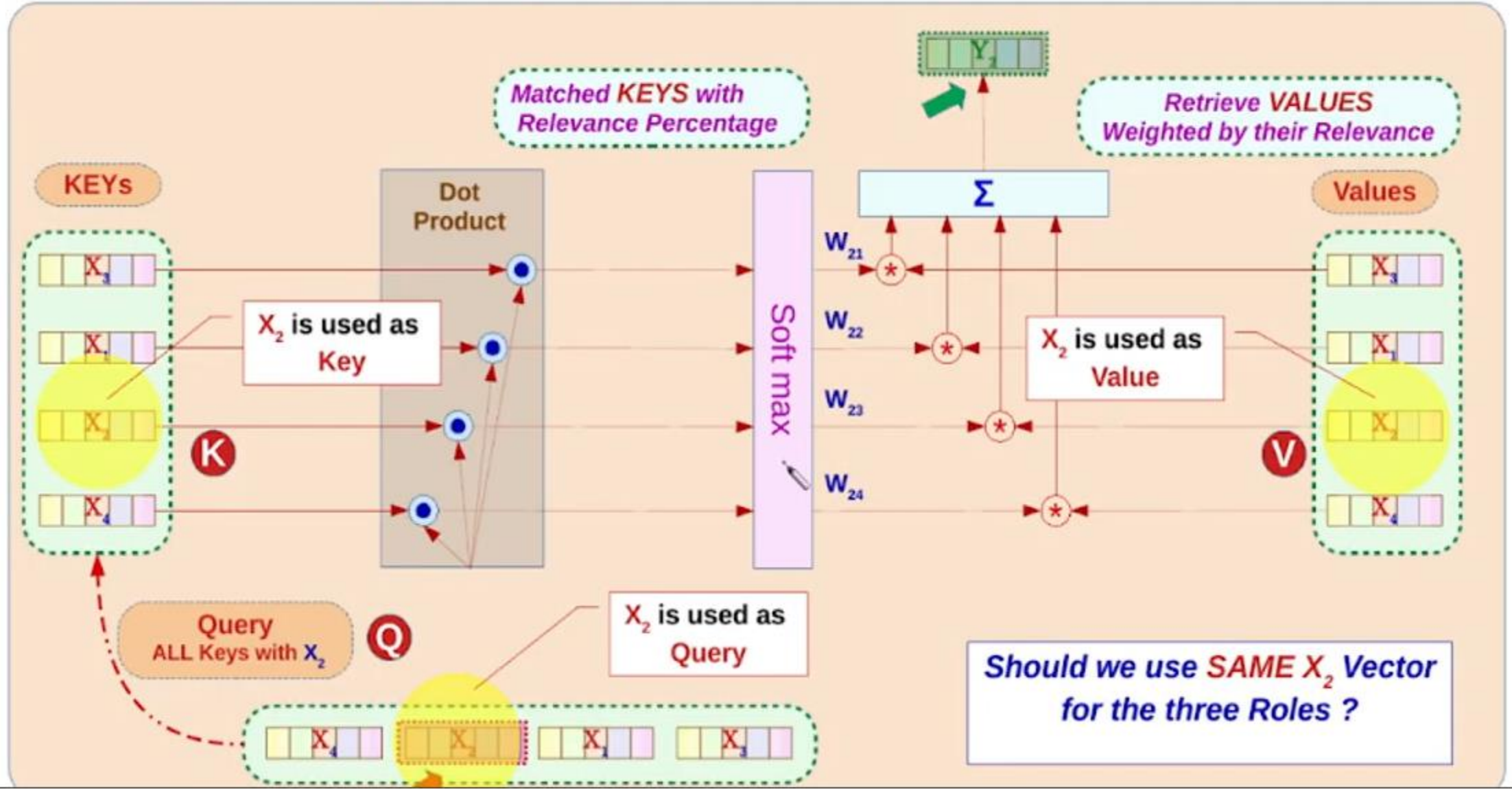


Self Attention Mechanism (as Query and {Key-Value} Dictionary)

(Query using X_2 All Keys in Dictionary and retrieve Corresponding Values

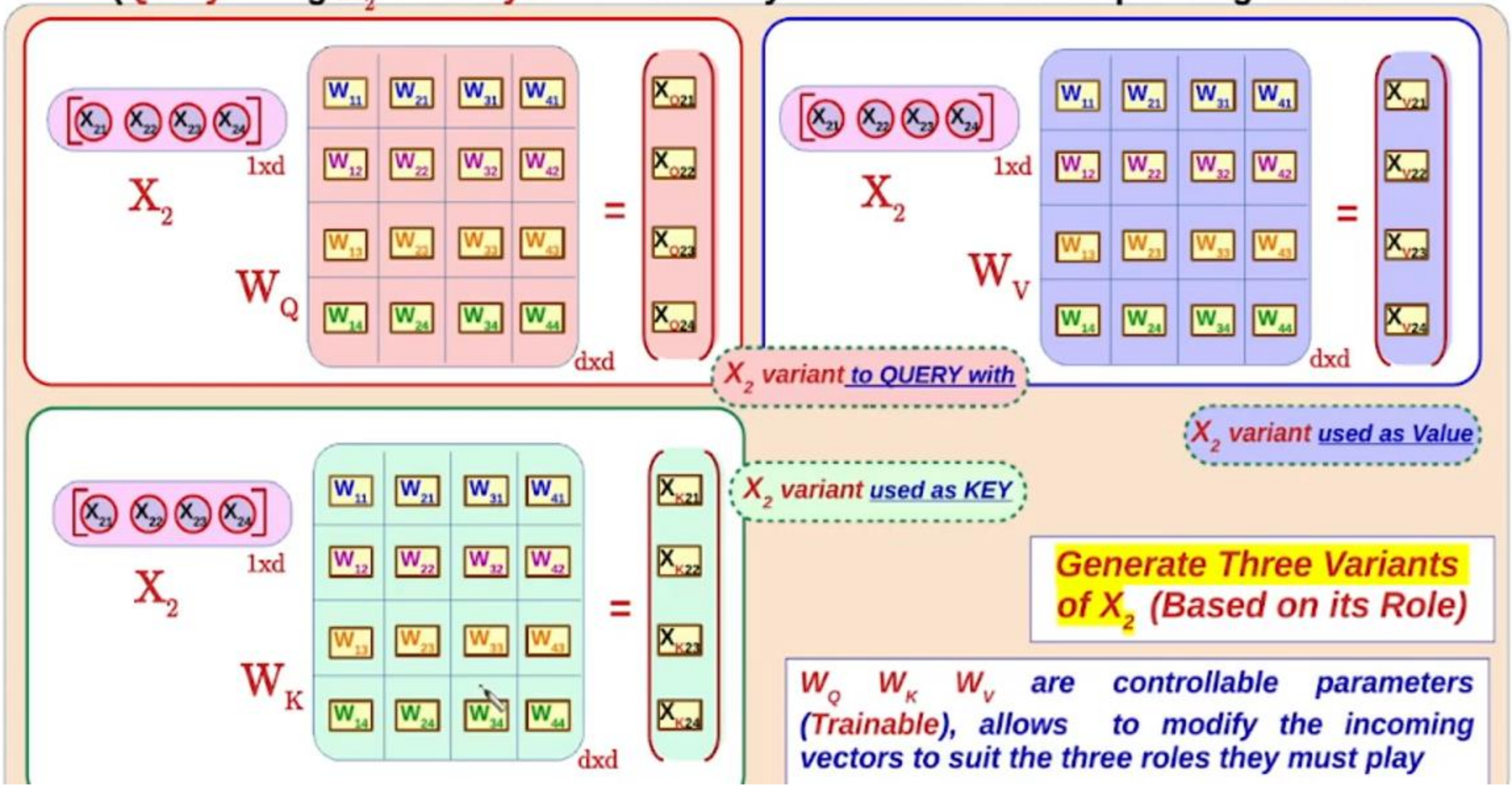


Self Attention Mechanism (Input Word plays Different Roles)
(Query using X_2 All Keys in Dictionary and retrieve Corresponding Values



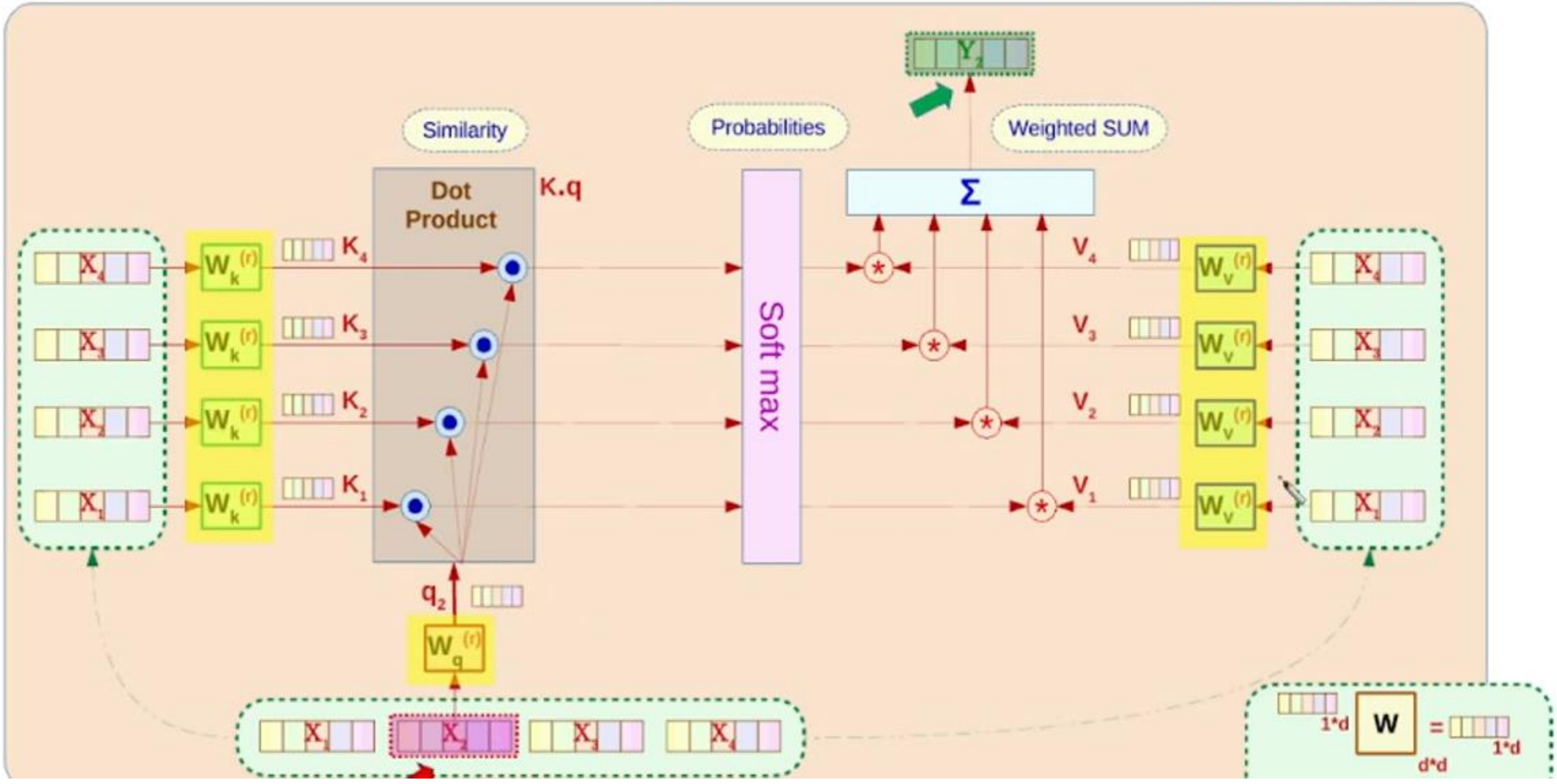
Self Attention Mechanism (Linear Transformation of Inputs)

(Query using X_2 All Keys in Dictionary and retrieve Corresponding Values



[27]

Self Attention Mechanism (Linear Transformation of Inputs)
(Self Attention of word X_2 w.r.t All words in Sentence (X_1, X_2, X_3, X_4)



Self Attention Mechanism
(Embedding Vector Dimension and Softmax Sensitivity)

$$w'_{ij} = x_i^T x_j$$

$$[X_i]_{1xd}$$

d ↑

↑ $w'_{ij} = x_i^T x_j$

As Dimension of Embedding "d" Increases the Calculated weight value "w" increases

$[X_i]_{1xd}$ SoftMax $w_{ij} = \frac{\exp w'_{ij}}{\sum_j \exp w'_{ij}}$

High Weight Values kill the gradient, and slow down learning

d ↓

w'_{ij}	SoftMax
1	0.0321
2	0.0871
3	0.2369
4	0.6439

For small "d"

d ↑

w'_{ij}	SoftMax
5	0.0000
10	0.0000
15	0.0067
20	0.9933

For medium "d"

d ↑↑

w'_{ij}	SoftMax
10	0.0000
20	0.0000
30	0.0000
40	1.0000

For large "d"

Self Attention Mechanism
(Embedding Vector Dimension and Softmax Sensitivity)

$$w'_{ij} = x_i^T x_j$$

$[X_i]_{1xd}$

d ↑ ↑ $w'_{ij} = x_i^T x_j$

*As Dimension of Embedding "**d**" Increases the Calculated weight value "**w**" increases*

Scale down "Weights" Values
before applying SoftMax

***Scaling Factor** should be related to Embedding Dimension "**d**"*

$[X_i]_{1xd}$ SoftMax $w_{ij} = \frac{\exp w'_{ij}}{\sum_j \exp w'_{ij}}$

High Weight Values kill the gradient, and slow down learning

d ↓

1	SoftMax	0.0321
2		0.0871
3		0.2369
4		0.6439

w'_{ij} For small "d"

d ↑

5	SoftMax	0.0000
10		0.0000
15		0.0067
20		0.9933

w'_{ij} For medium "d"

d ↑↑

10	SoftMax	0.0000
20		0.0000
30		0.0000
40		1.0000

w'_{ij} For large "d"

Self Attention Mechanism

(Scaling based on Square Root of Embedding Vector Dimension)

V_1 V_2

V_1: $X_1, X_2, X_3, X_4, X_5, X_6, X_7, X_8, X_9, X_{10}, X_{11}, X_{12}$ — $1 \times d_1$

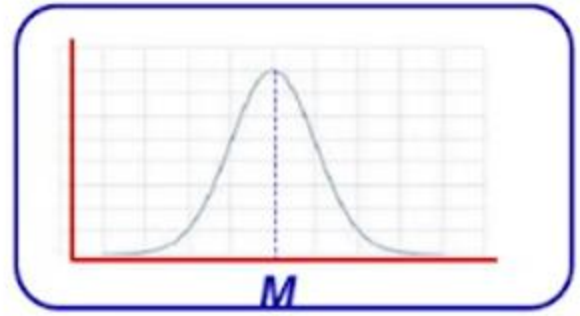
V_2: X_1, X_2, X_3, X_4 — $1 \times d_2$

Average Values of elements of V_1 = M

Average Values of elements of V_2 = M

Vector Length of V_1 *(all Ms)* = $V_1^T V_1 = M\sqrt{d_1}$

Vector Length of V_2 *(all Ms)* = $V_2^T V_2 = M\sqrt{d_2}$



$\sqrt{d}$ *is a good choice for scaling*

$\sqrt[4]{d}$ *Some times is used*

Remember:

"d" is

Dimension of Embedding

Not

length of input Sequence

$\text{SoftMax}\left[w'_{ij}\right]$

$\text{SoftMax}\left[\frac{w'_{ij}}{\sqrt{d}}\right]$

d=100, $\sqrt{d}$ =10

10	SoftMax →	0.0000
20		0.0000
30		0.0000
40		1.0000

10	/ $\sqrt{d}$ →	1	Scaled →	2	SoftMax →	0.0321
20		2		3		0.0871
30		3		4		0.2369
40		4				0.6439

$$\begin{bmatrix} 5 & 5 & 5 & 5 \end{bmatrix} \begin{bmatrix} 5 \\ 5 \\ 5 \\ 5 \end{bmatrix} = \sqrt{5^2 + 5^2 + 5^2 + 5^2} = \sqrt{4 * 5^2}$$

$$5\sqrt{4} \rightarrow d$$

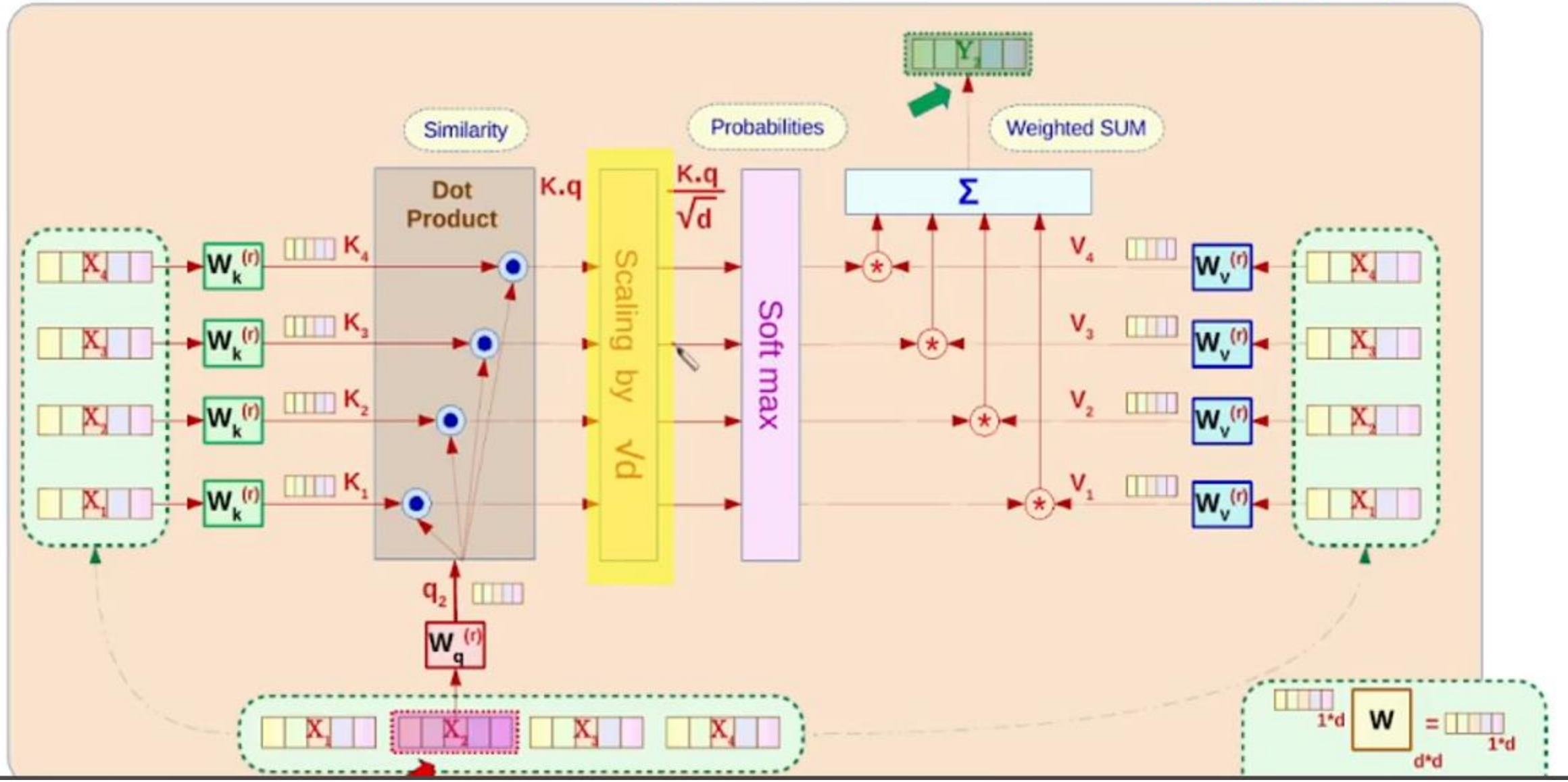
$$\begin{bmatrix} 5 & 5 & 5 & \cdots & 5 \end{bmatrix} \begin{bmatrix} 5 \\ 5 \\ 5 \\ 5 \\ 5 \\ 5 \\ 5 \\ 5 \end{bmatrix}_{100} = \sqrt{\underbrace{5^2 + 5^2 + \cdots + 5^2}_{100}} = \sqrt{100 \times 5^2}$$

$$= 5\sqrt{100} \rightarrow d$$

[31]

Self Attention Mechanism (Adding Scaling)

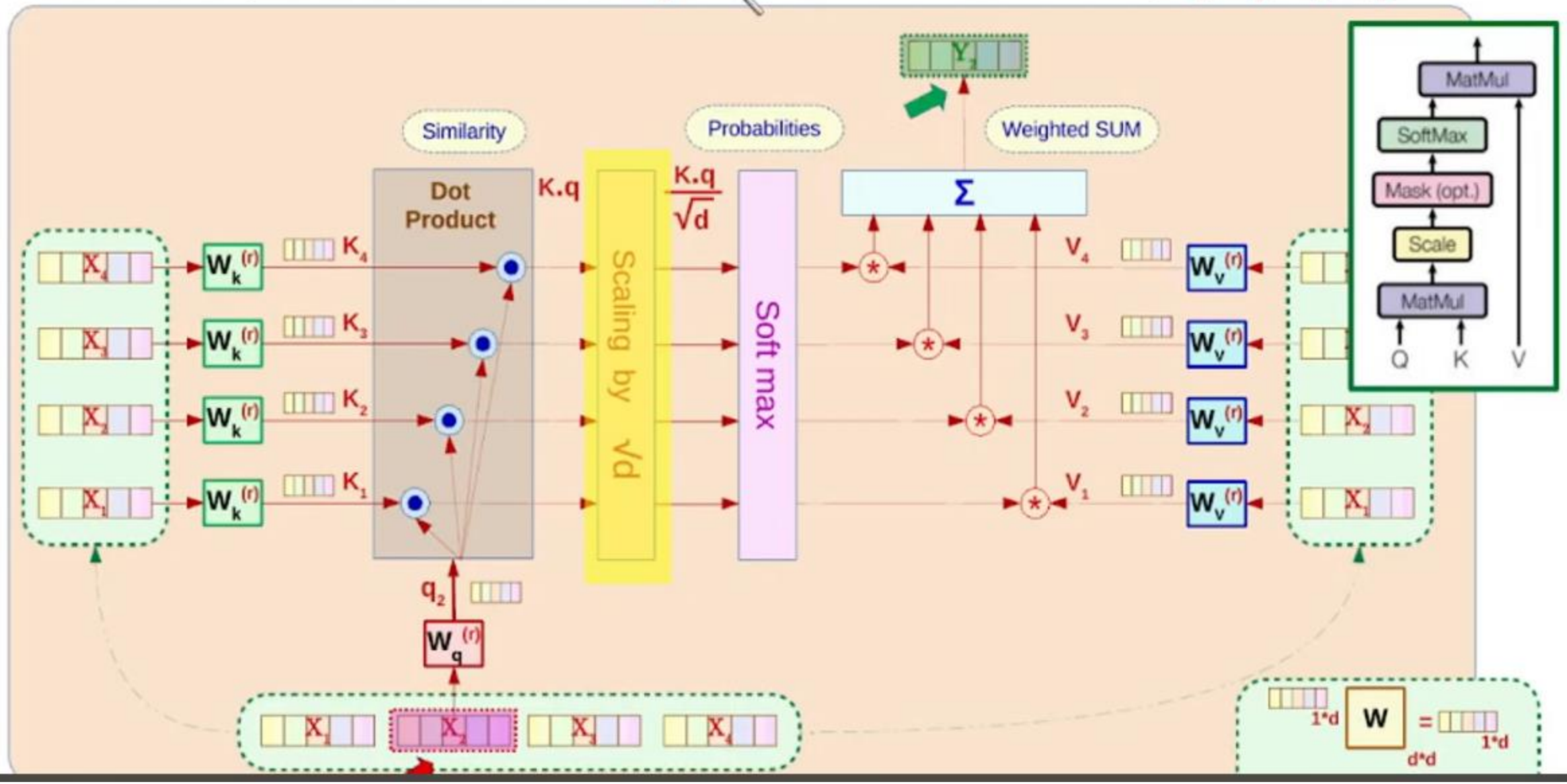
(Self Attention of word X_2 w.r.t All words in Sentence (X_1, X_2, X_3, X_4)

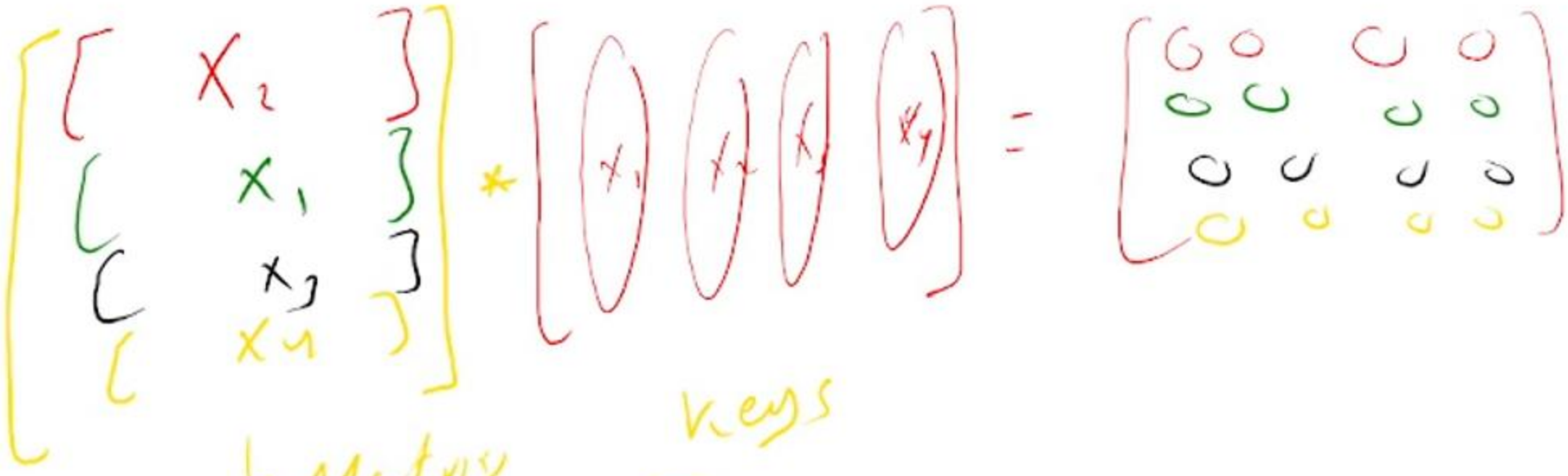


[32]

Self Attention Mechanism (Adding Scaling)

(Self Attention of word X_2 w.r.t All words in Sentence (X_1, X_2, X_3, X_4)



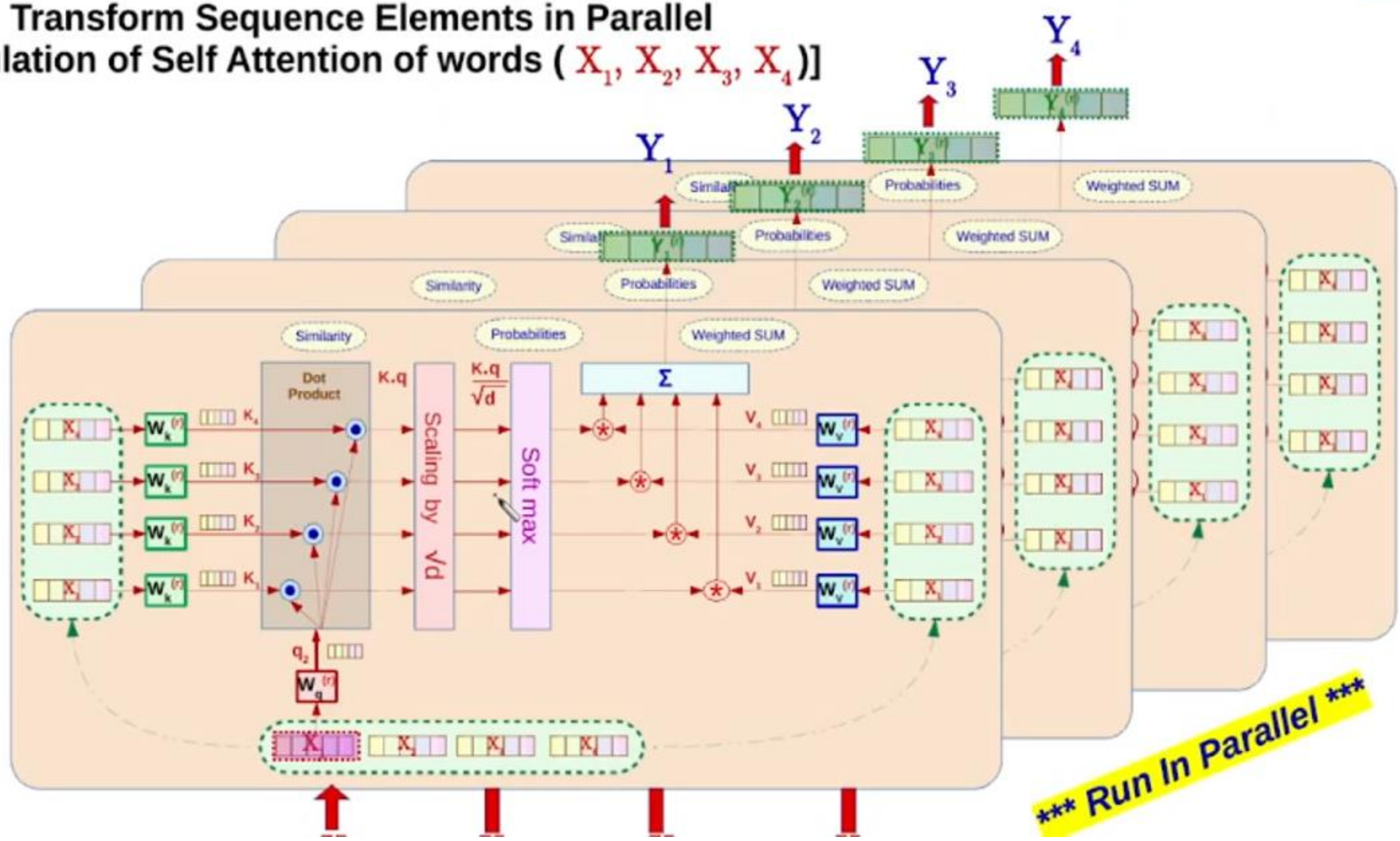


to get similarity of All words with other words

At one time

Parallel Processing
Transform Sequence Elements in Parallel
[Calculation of Self Attention of words (X_1, X_2, X_3, X_4)]

[33]



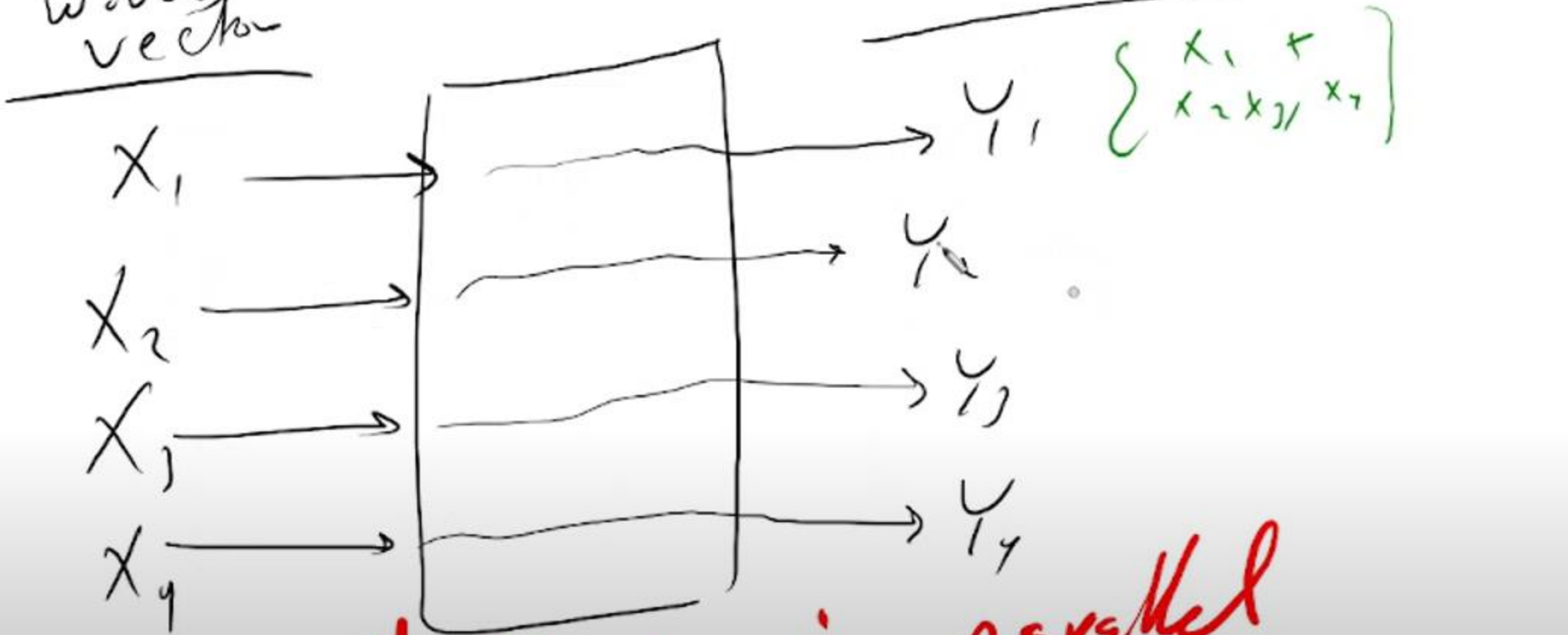
Multi-Head Attention

Wide Architecture

Narrow Architecture

individual word vectors

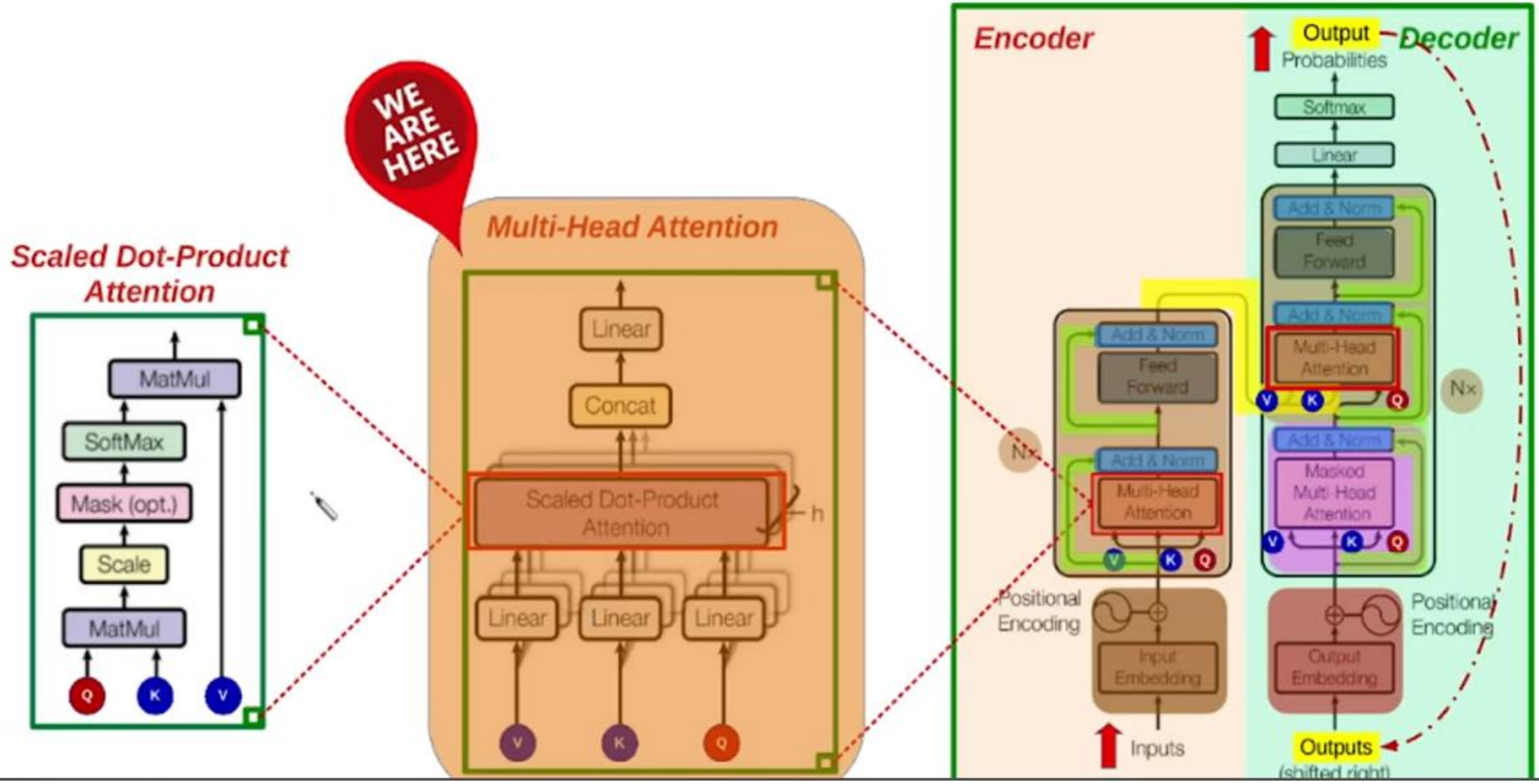
word vectors that Embed the context words



all Done in parallel

Encoder – Decoder Transformer

[35]



Multi-Head Self Attention

Combining several self attention mechanisms give the self attention greater power of discrimination

Each Head is indexed with "**r**"

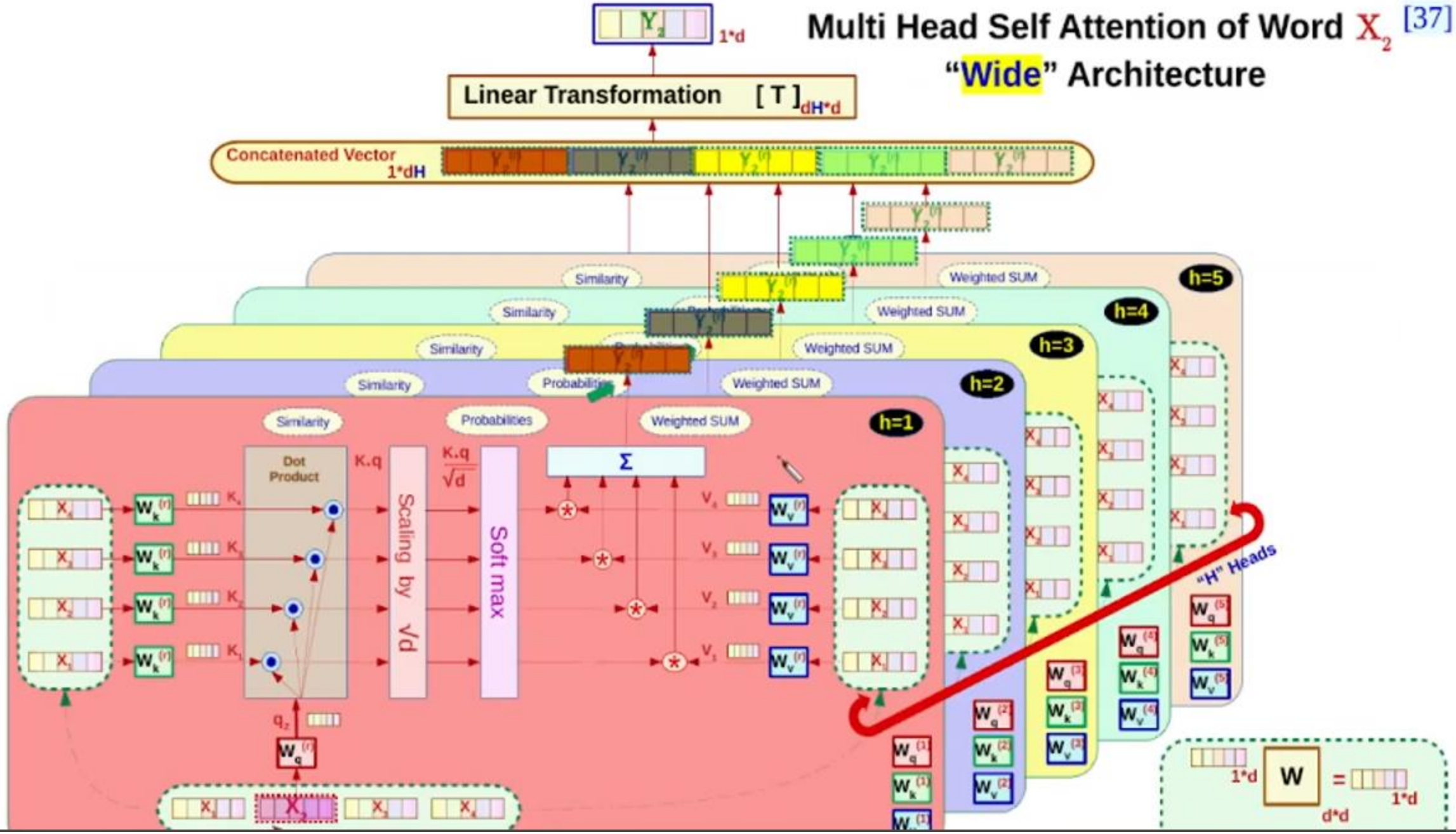
Each head has its own matrices W^r_q , W^r_k , W^r_v .

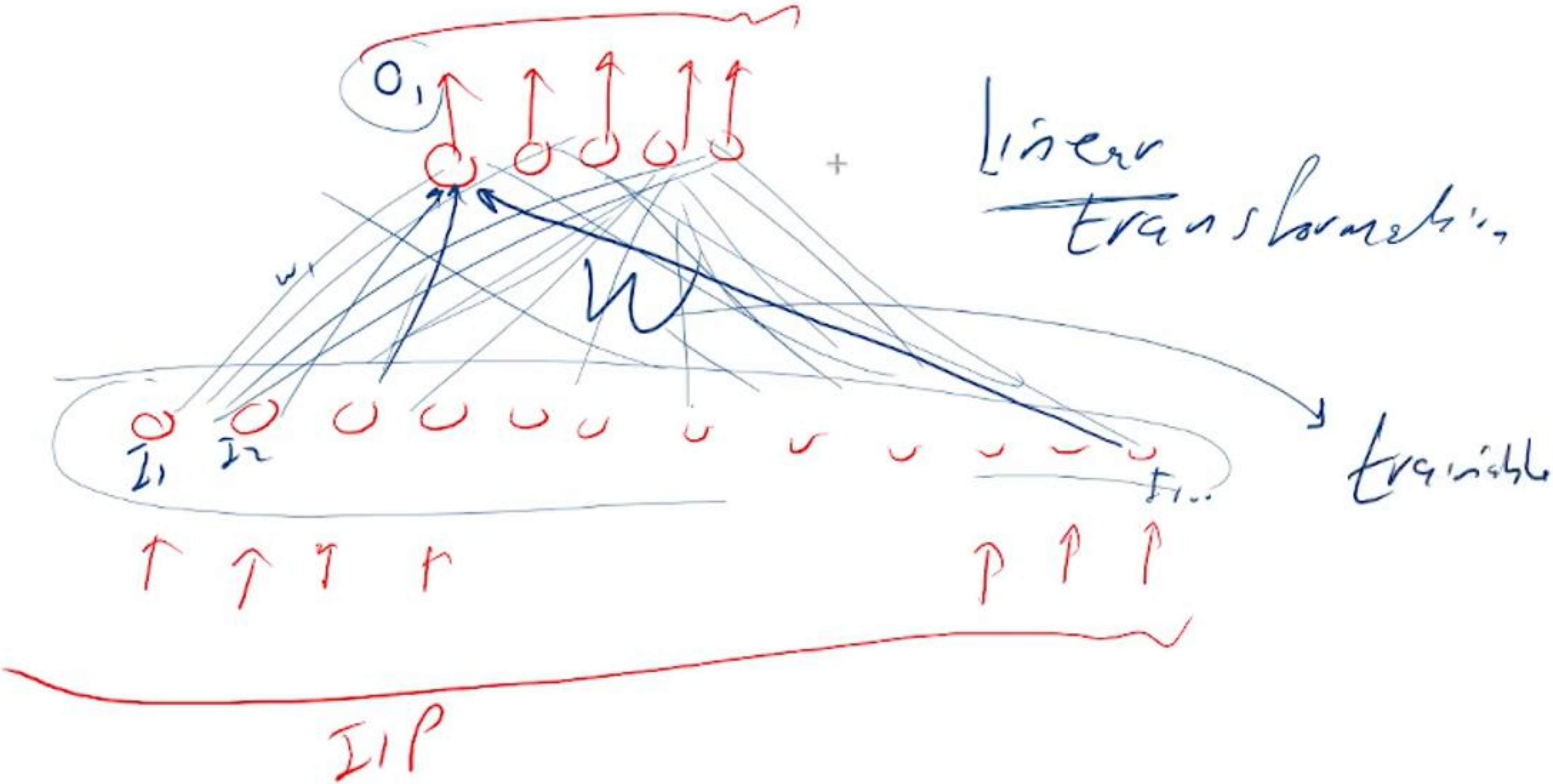
For input **X** each attention head produces a different output vector Y^r.

All Output vectors are concatenated , and pass through a **linear transformation** to reduce the dimension back to original **X** dimension.

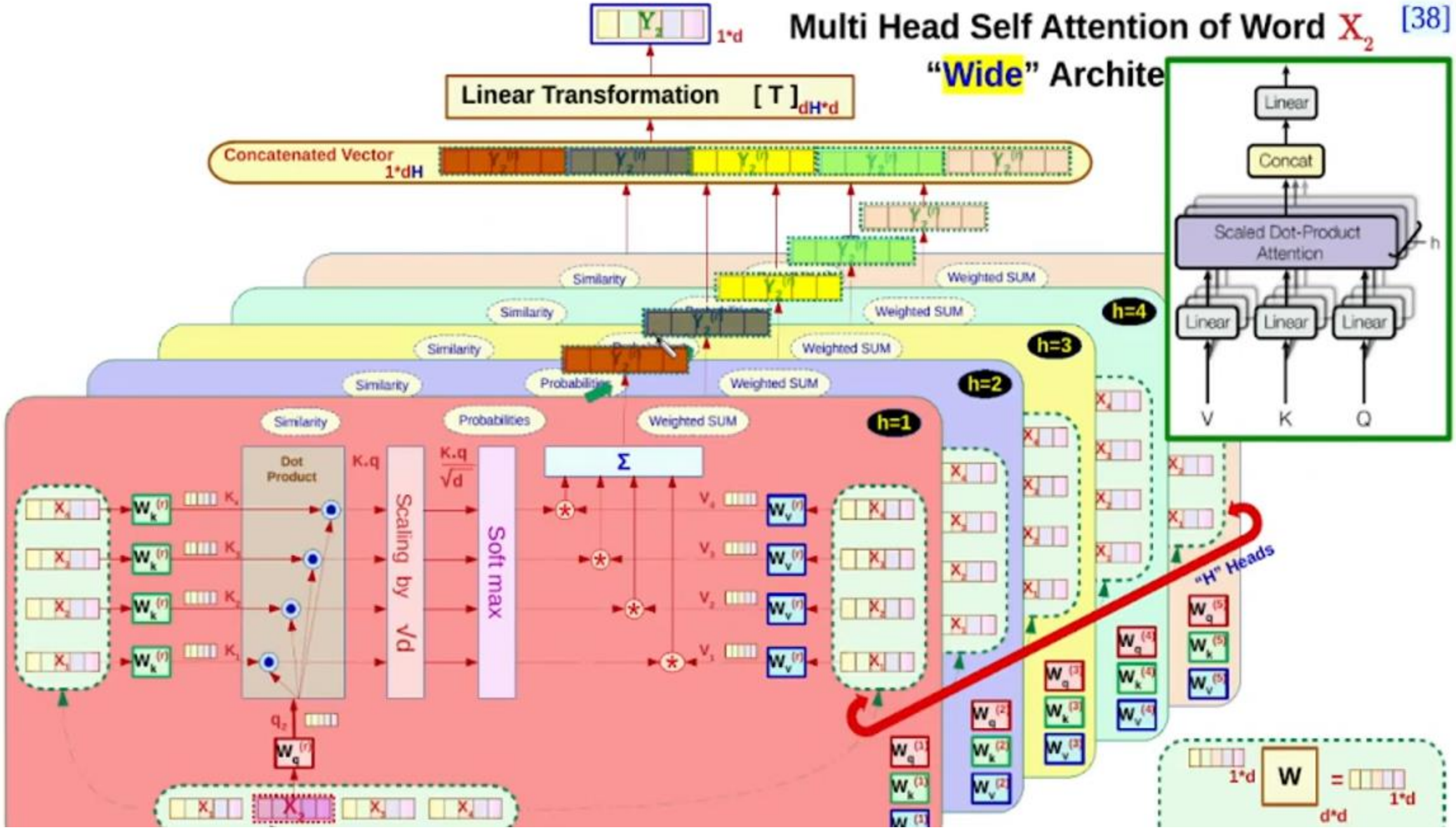
Multi Head Self Attention of Word X_2 [37]

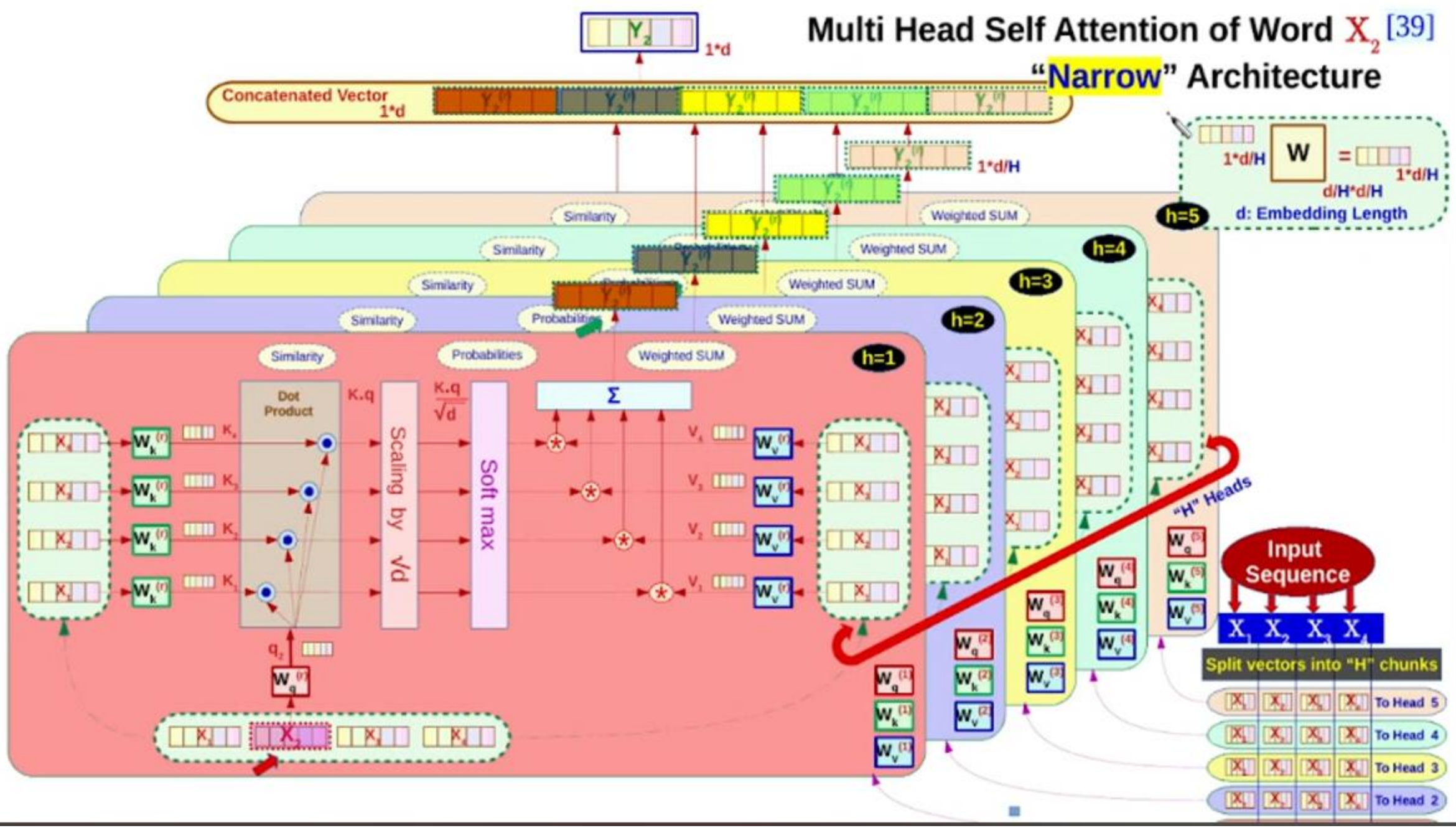
"Wide" Architecture





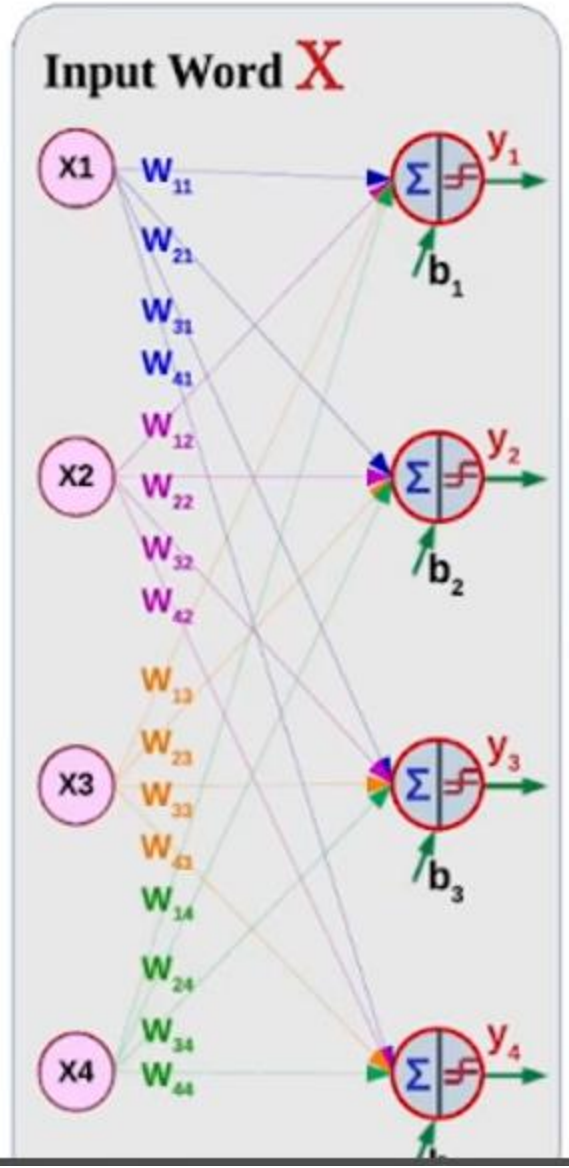
O1
w1
W
I1
I2
I/P
Linear Transformation
trainable


Multi Head Self Attention of Word X_2
[38]
"Wide" Archite
1*d
Linear Transformation [T]dH*d
Concatenated Vector 1*dH
Linear
Concat
Scaled Dot-Product Attention
h
Linear
Linear
Linear
V
K
Q
Similarity
Weighted SUM
Probabilities
h=4
h=3
h=2
h=1
Dot Product
K.q
Scaling by √d
Soft max
Σ
"H" Heads
1*d
W
d*d
=
1*d


Multi Head Self Attention of Word X_2 [39]
"Narrow" Architecture
Concatenated Vector 1*d
1*d
1*d/H
1*d/H
W
d/H*d/H
1*d/H
d: Embedding Length
h=5
h=4
h=3
h=2
h=1
Similarity
Probabilities
Weighted SUM
Dot Product
K.q
Scaling by √d
Soft max
Σ
"H" Heads
Input Sequence
Split vectors into "H" chunks
To Head 5
To Head 4
To Head 3
To Head 2

Parallelization Implementation Issues

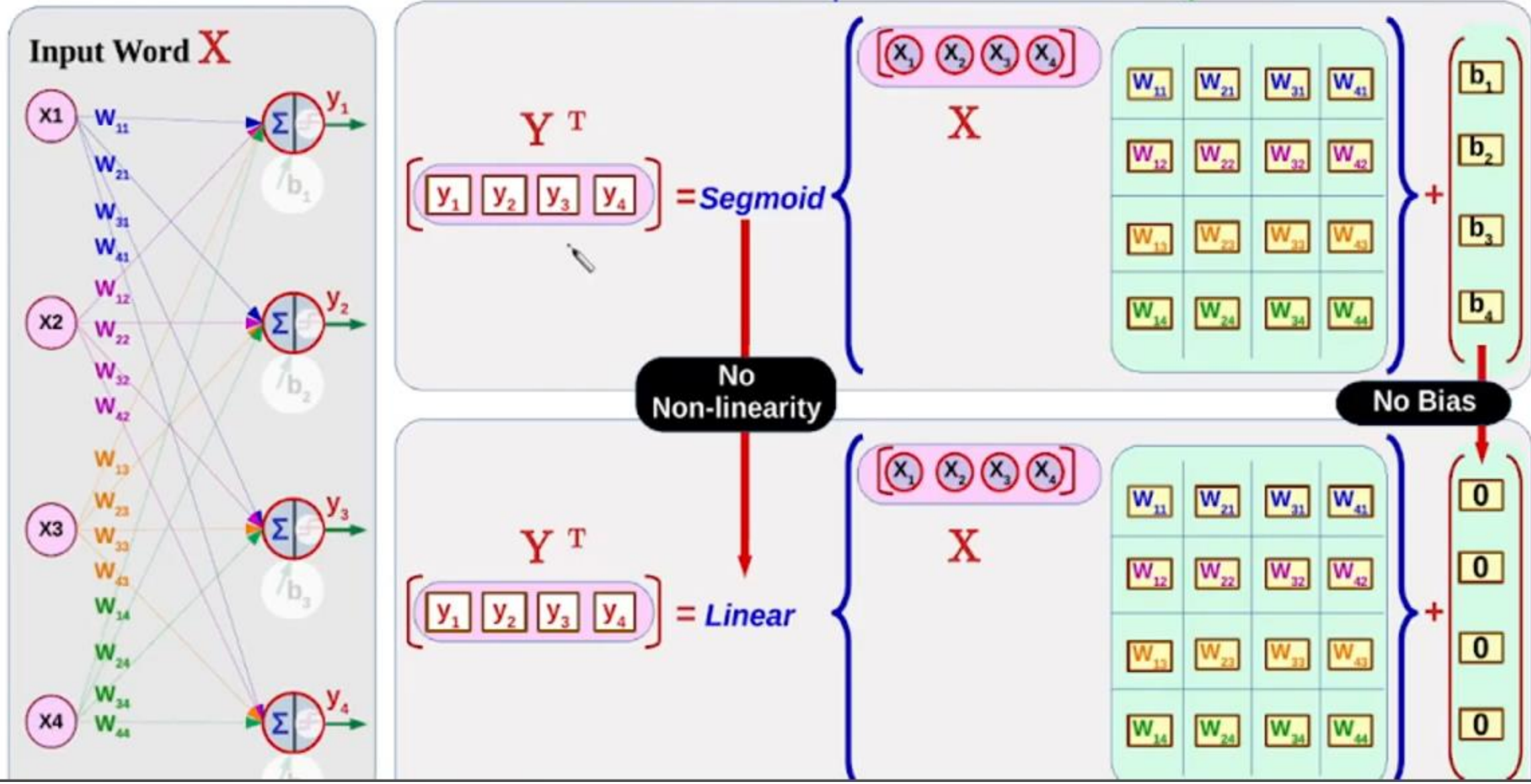
Using FF NN for Multiplication of a Vector by a Matrix



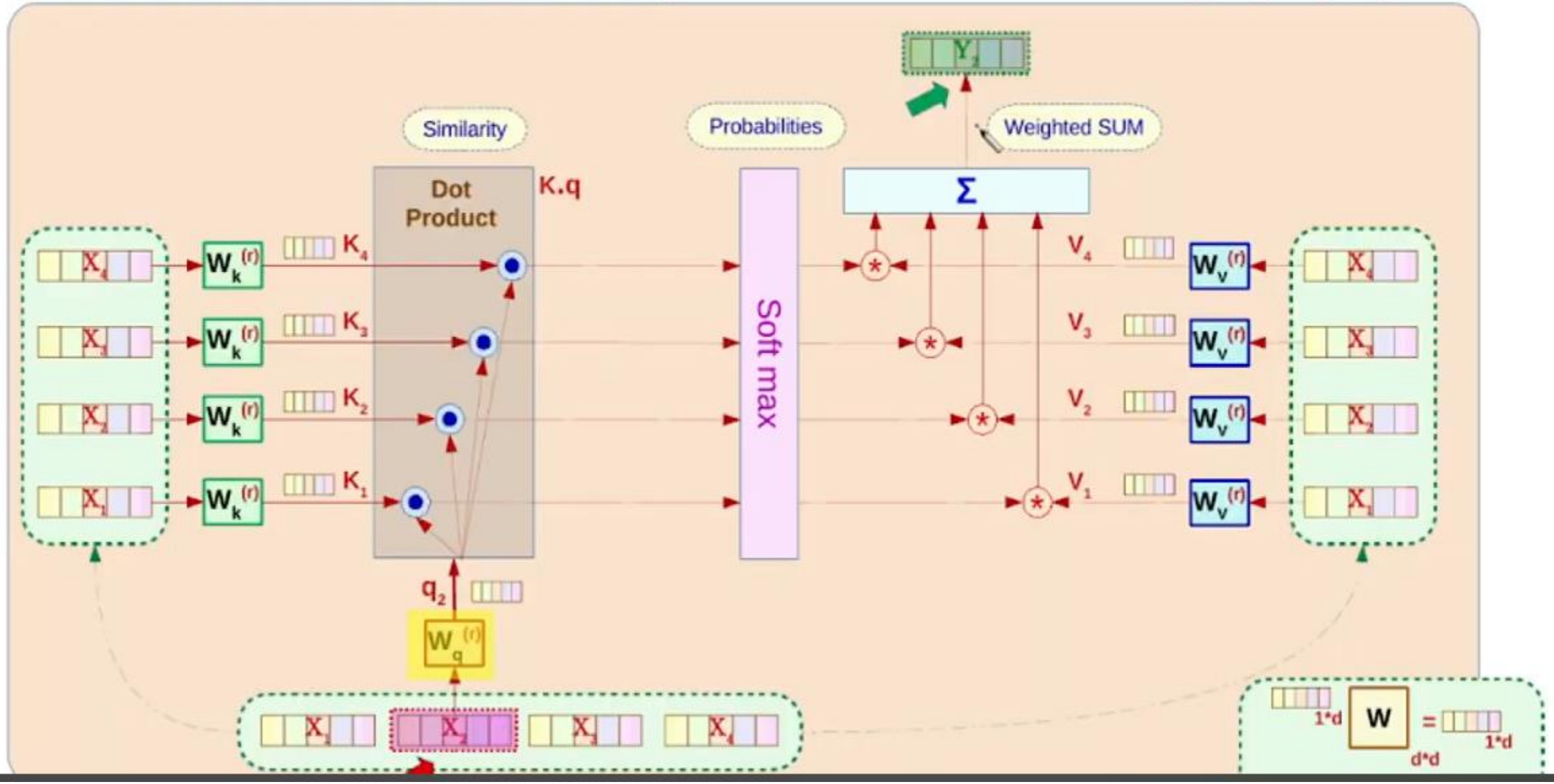
$$Y^T = \begin{bmatrix} y_1 & y_2 & y_3 & y_4 \end{bmatrix} = Segmoid\left\{ \begin{bmatrix} x_1 & x_2 & x_3 & x_4 \end{bmatrix} X \begin{bmatrix} w_{11} & w_{21} & w_{31} & w_{41} \\ w_{12} & w_{22} & w_{32} & w_{42} \\ w_{13} & w_{23} & w_{33} & w_{43} \\ w_{14} & w_{24} & w_{34} & w_{44} \end{bmatrix} \right\} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix}$$

Using FF NN for Multiplication of ONE Vector by a Matrix

Example: Multiply Vector of Current Word X by W_q (Query) *{Remember: W_q is Squared Matrix}*

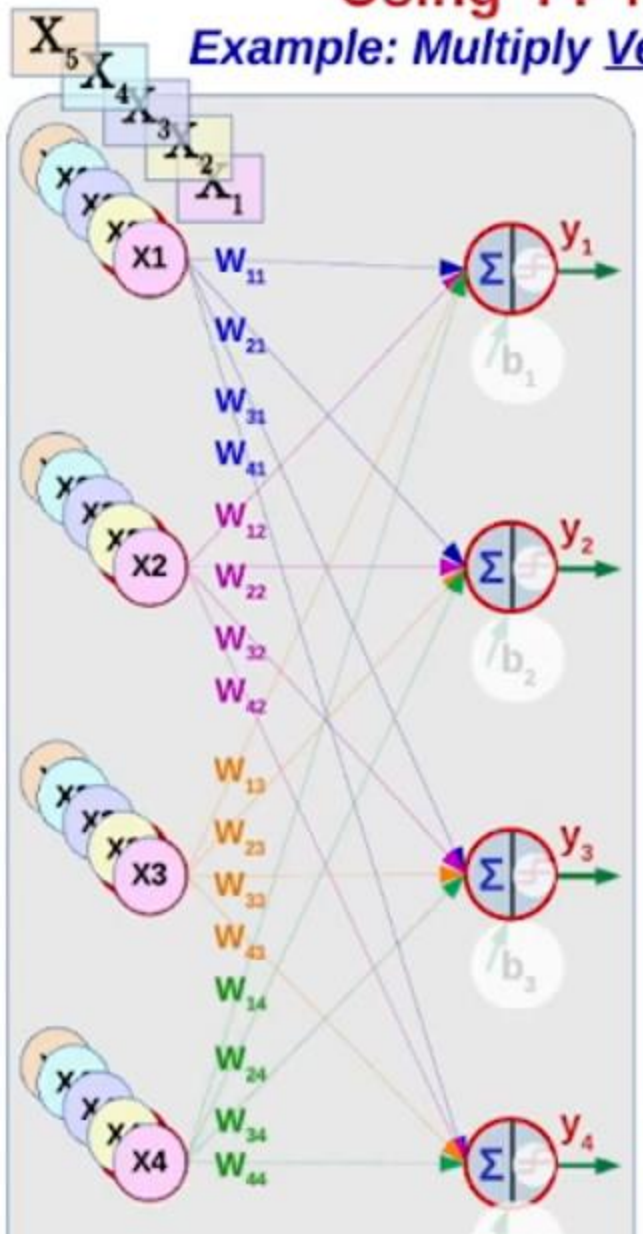


Self Attention Mechanism (Linear Transformation of Inputs)
(Self Attention of word X_2 w.r.t All words in Sentence (X_1, X_2, X_3, X_4)

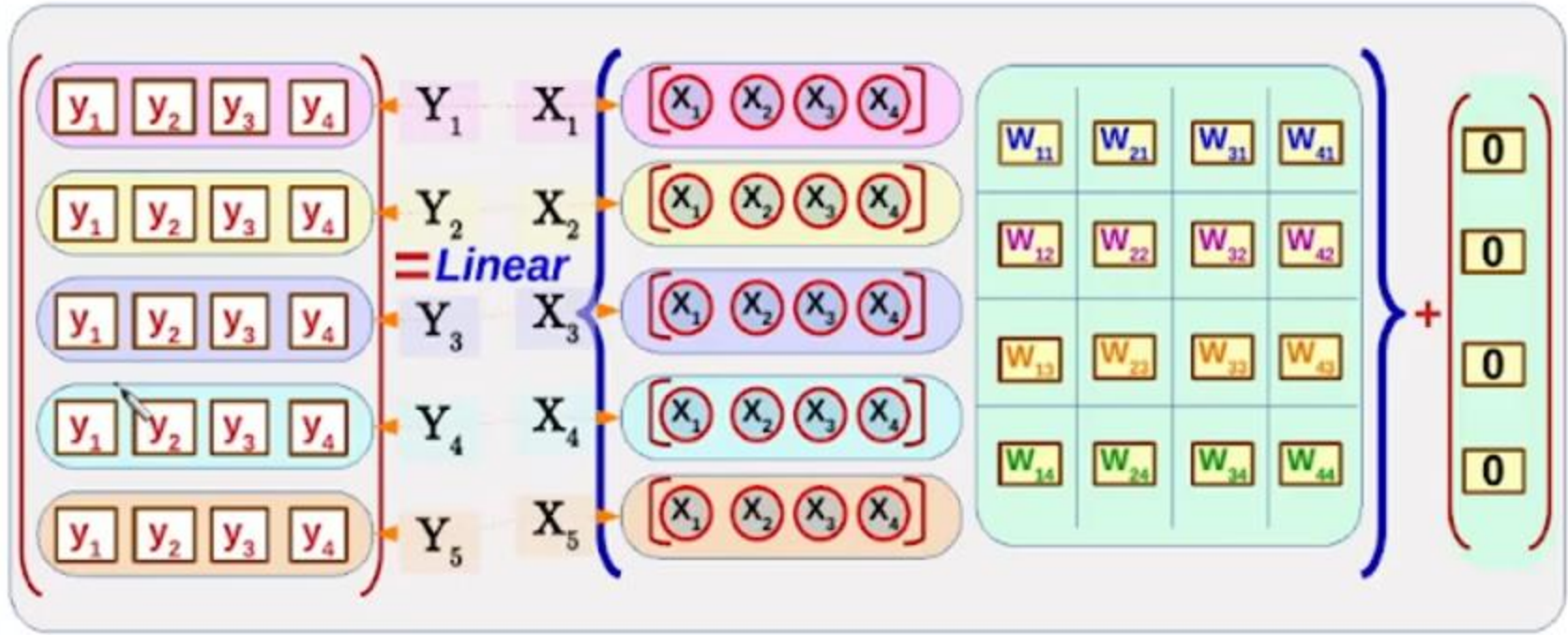


Using FF NN for Multiplication of Multiple Vectors by a Matrix

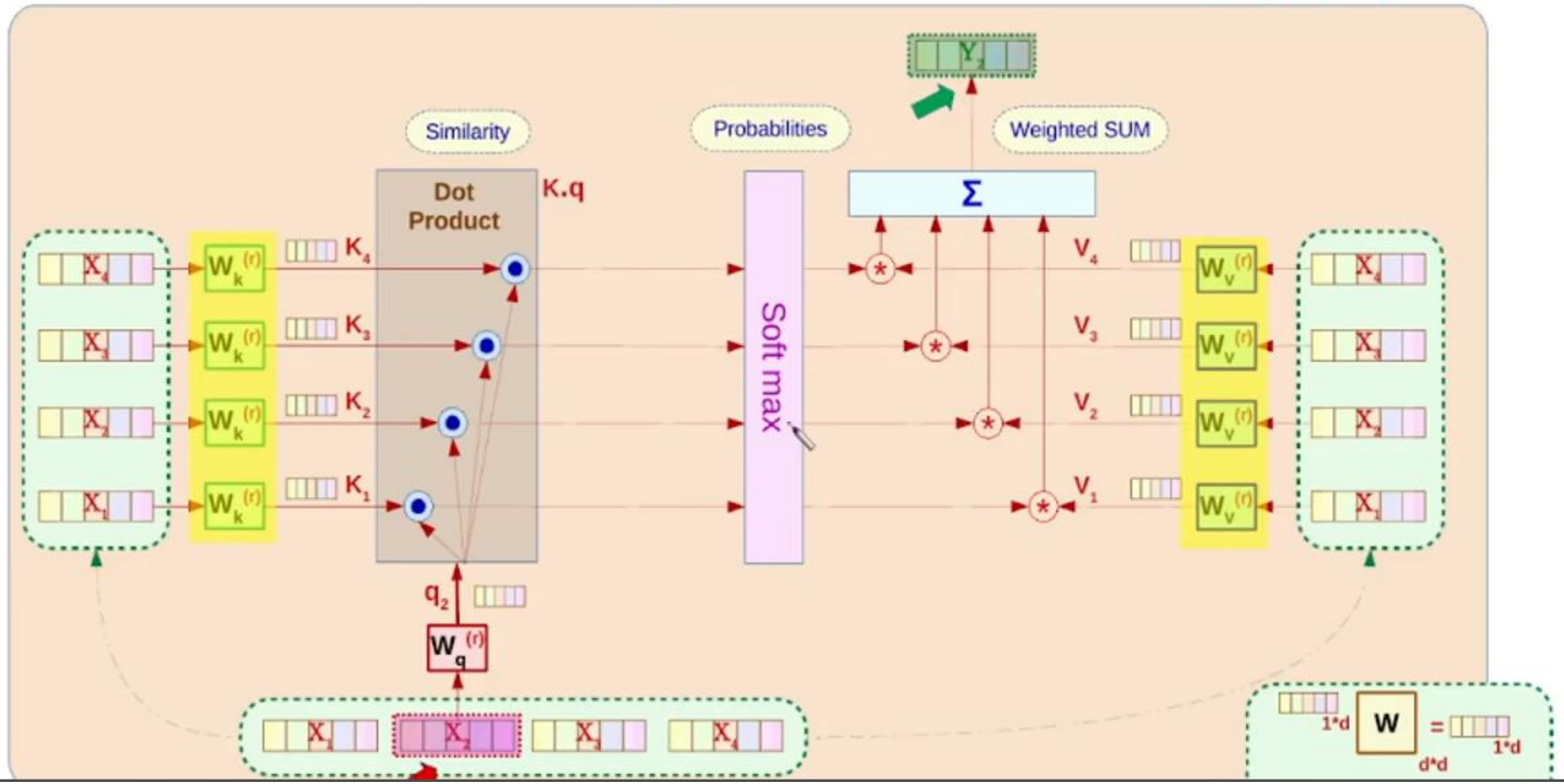
Example: Multiply Vectors of ALL Words by W_k (Key) or W_v (Value) { W_k W_v is Squared Matrix}



Design FF NN with the following Considerations:
Activation: *Linear*
Bias: *Zeros*
Inputs: *All Words Within Same Paragraph (X_1, X_2, X_3, X_4, X_5)*

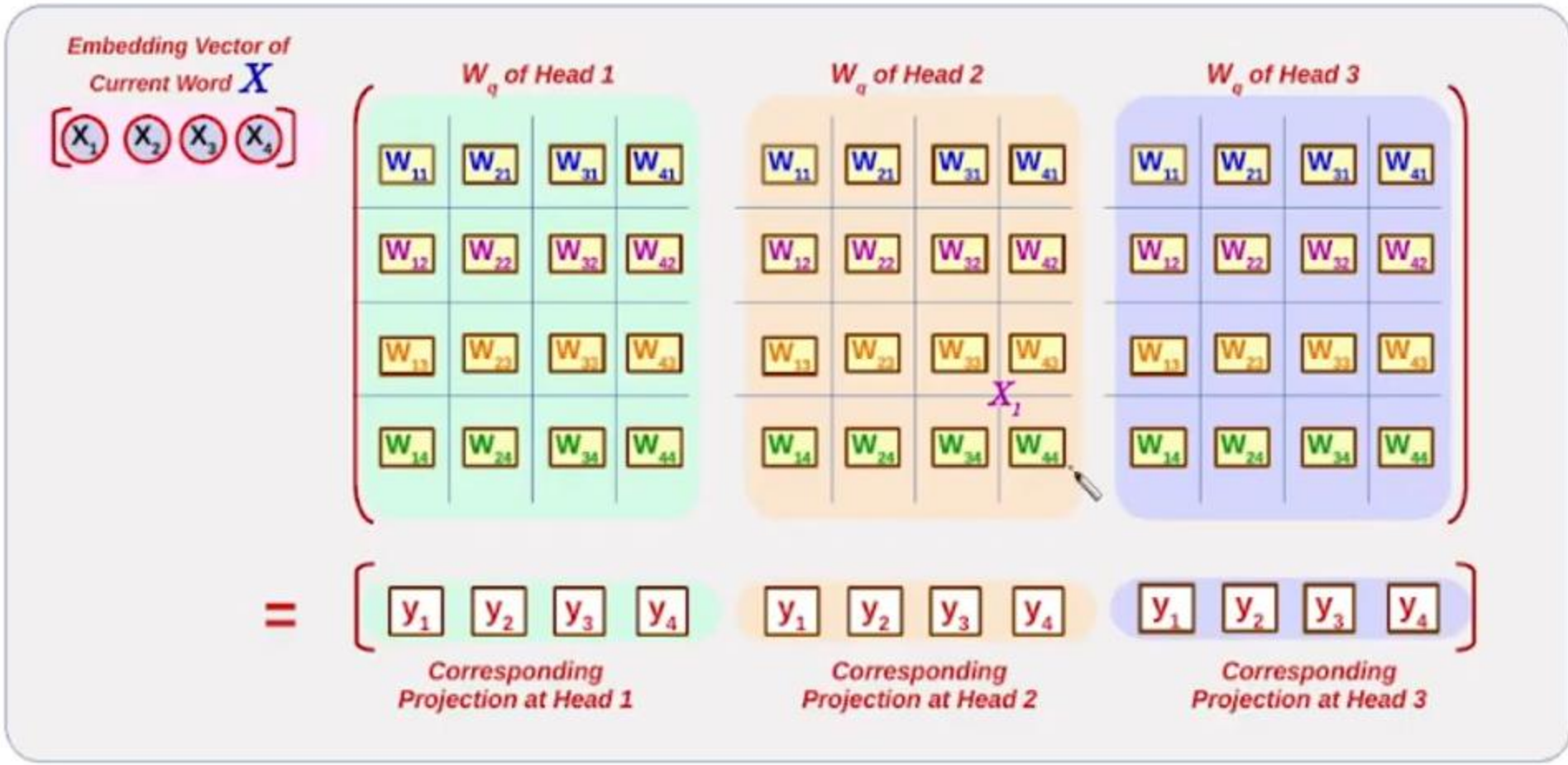


Self Attention Mechanism (Linear Transformation of Inputs)
(Self Attention of word X_2 w.r.t All words in Sentence (X_1, X_2, X_3, X_4)



Using FF NN for Multiplication of Vector by a Multiple Matrices

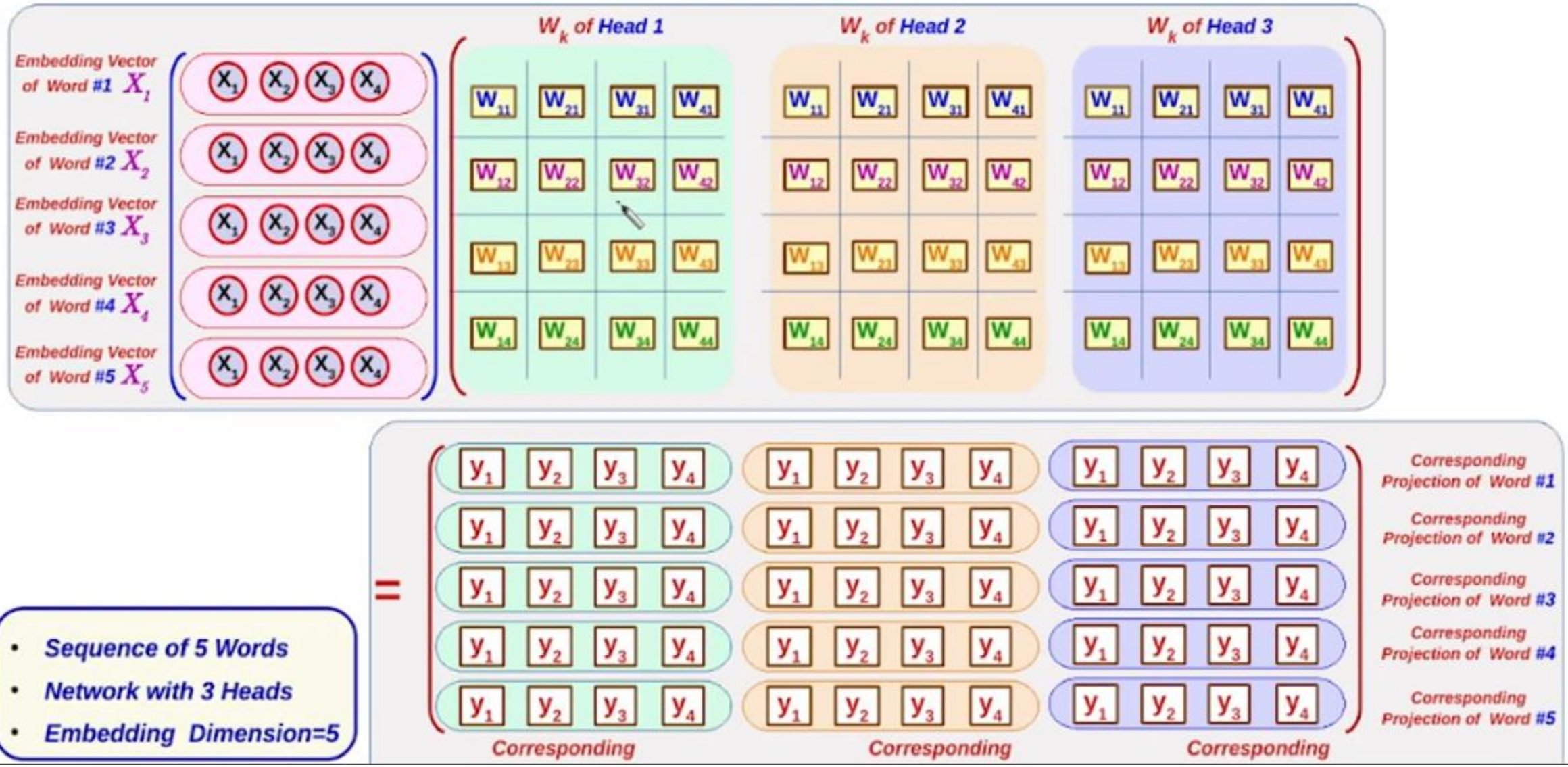
Example: Multiply Vector of Current Word by Multiple (different) W_q (Queries of Multiple Heads)



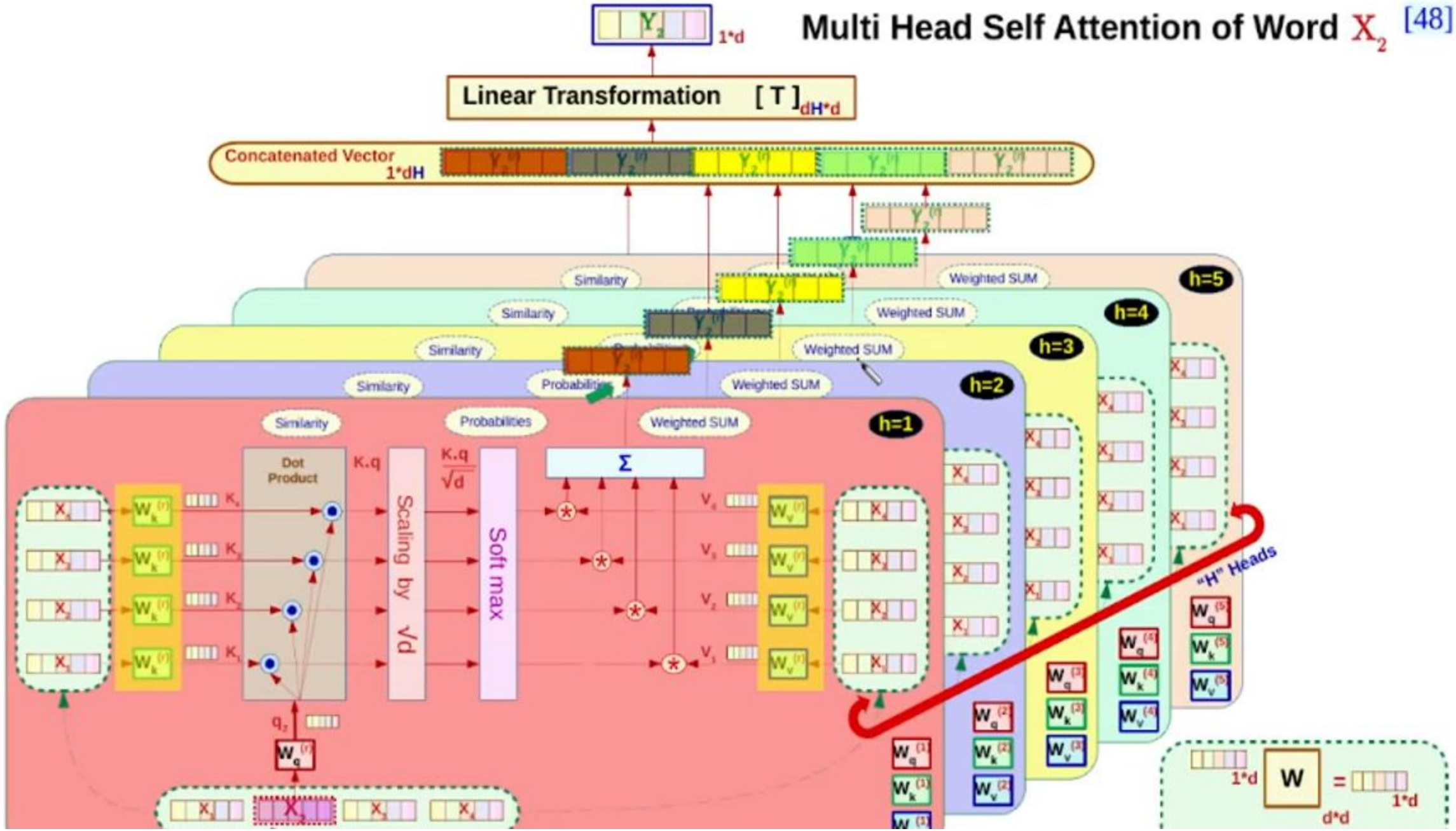
Using FF NN for Multiplication of Multiple Vectors by Multiple Matrices

[47]

Usage: Multiply Vectors of All Words by Multiple (different) W_k (Keys of Multiple Heads)



- *Sequence of 5 Words*
- *Network with 3 Heads*
- *Embedding Dimension=5*


Multi Head Self Attention of Word X2 [48]
1*d
Linear Transformation [T]dH*d
Concatenated Vector 1*dH
Similarity
Weighted SUM
Probabilities
h=5
h=4
h=3
h=2
h=1
Dot Product
K.q
Scaling by √d
Soft max
Σ
"H" Heads
1*d
W
d*d
1*d

Adding Position Information

[1] Position Encoding

[2] Position Embedding

Encoder – Decoder Transformer

[50]