Language Model: Attention and Transformers

- *When she tried to print her tickets, she found that the printer was out of toner. She went to the stationery store to buy more toner. It was very overpriced. After installing the toner into the printer, she finally printed her* 

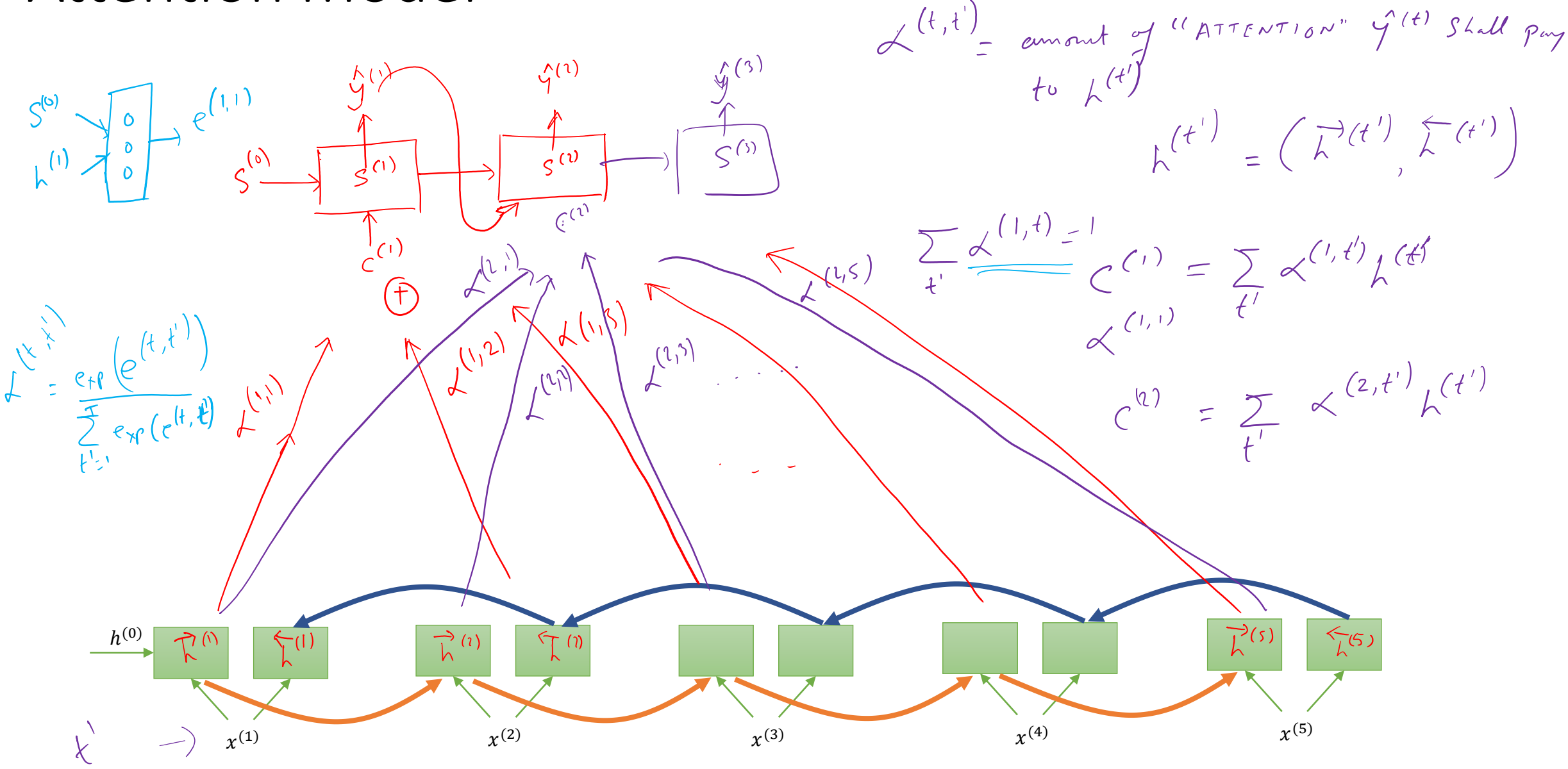
Sentiment Analysis

- I have bought and returned three of these units now. Each one has been defective, and finally I just gave up on returning the system. The DVD player constantly gives "Bad Disc" errors and skips if there is even the slightest smudge on a disc. The sound quality is very nice for the price, but since the player doesn't work, it's essentially useless. This is a complete rip-off at any price point

Sentiment Analysis

- I have bought and returned three of these units now. Each one has been defective, and finally I just gave up on returning the system. The DVD player constantly gives "Bad Disc" errors and skips if there is even the slightest smudge on a disc. The sound quality is very nice for the price, but since the player doesn't work, it's essentially useless. This is a complete rip-off at any price point

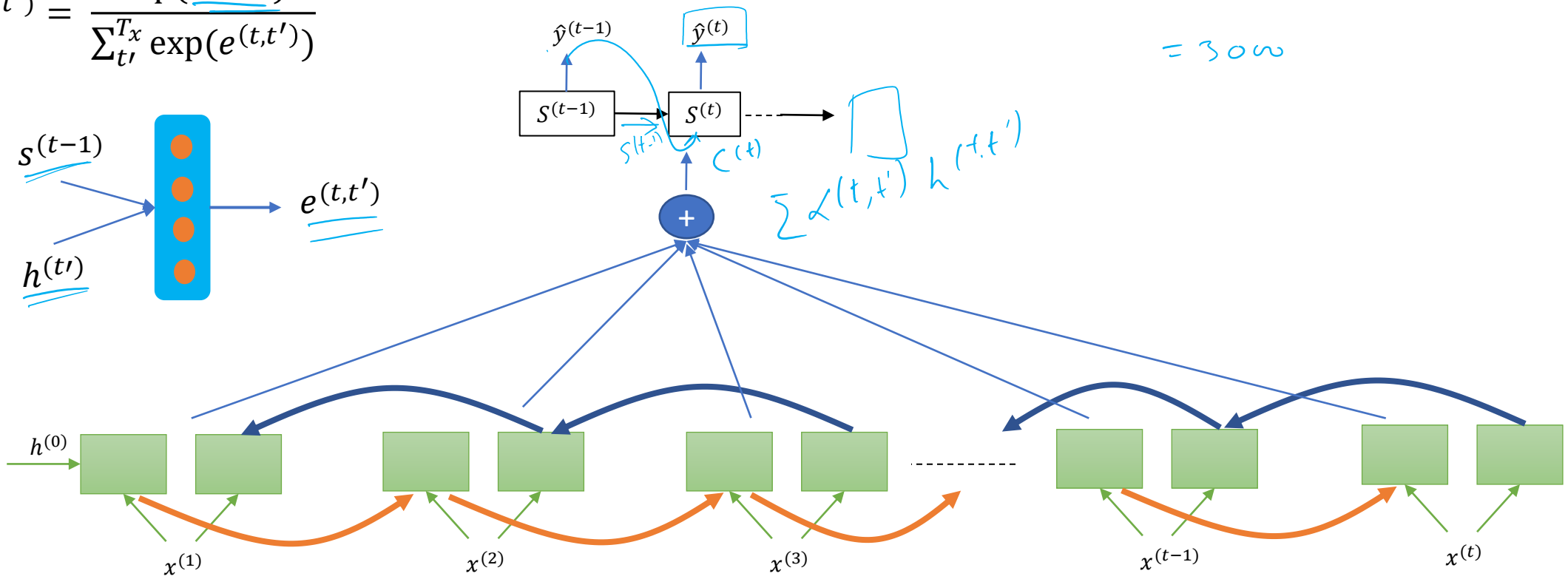
Attention Model



Attention Model

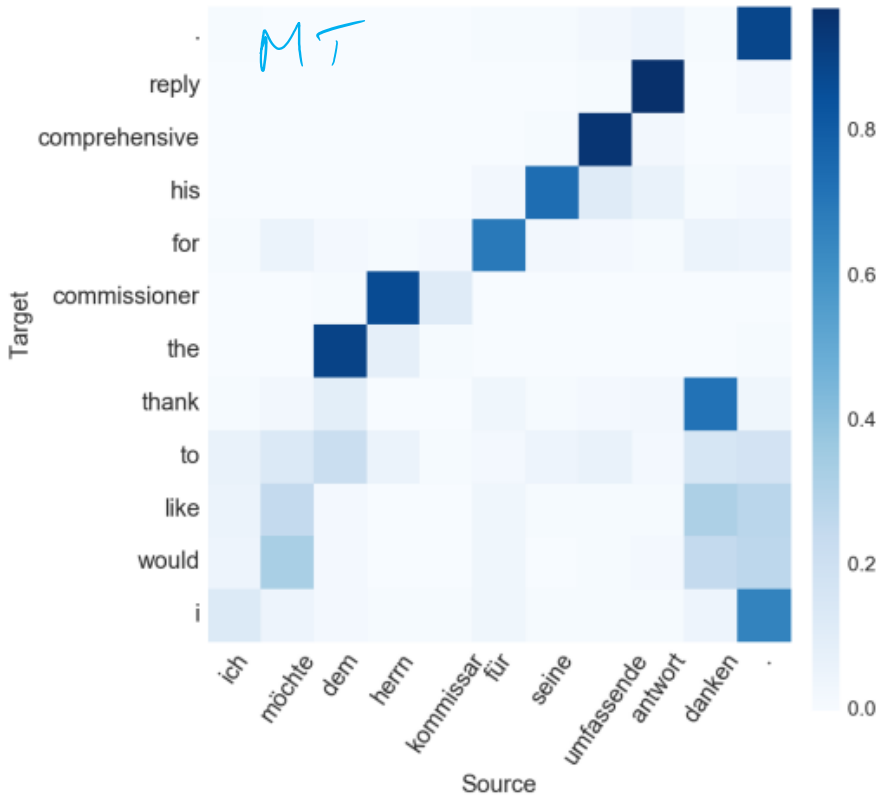
$\alpha^{(t,t')} = amount\ of\ attention\ y^{(t)}\ should\ pay\ to\ h^{(t')}$

$$\alpha^{(t,t')} = \frac{\exp(e^{(t,t')})}{\sum_{t'}^{T_x} \exp(e^{(t,t')})}$$

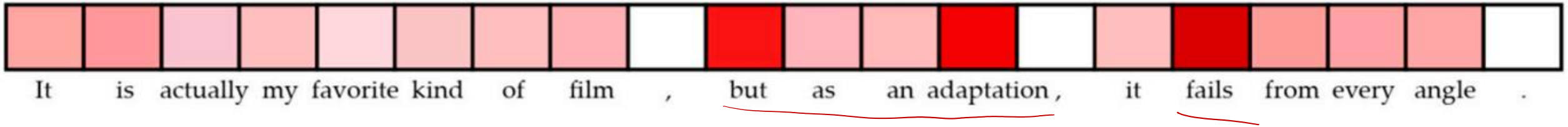


Attention Visualization

Machine Translation



Sentiment Classification



Transformers

Intuition

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 best models from the literature. We show that the Transformer generalizes well to other tasks by applying it successfully to English constituency parsing both with large and limited training data.

1 Introduction

Recurrent neural networks, long short-term memory [13] and gated recurrent [7] neural networks in particular, have been firmly established as state of the art approaches in sequence modeling and

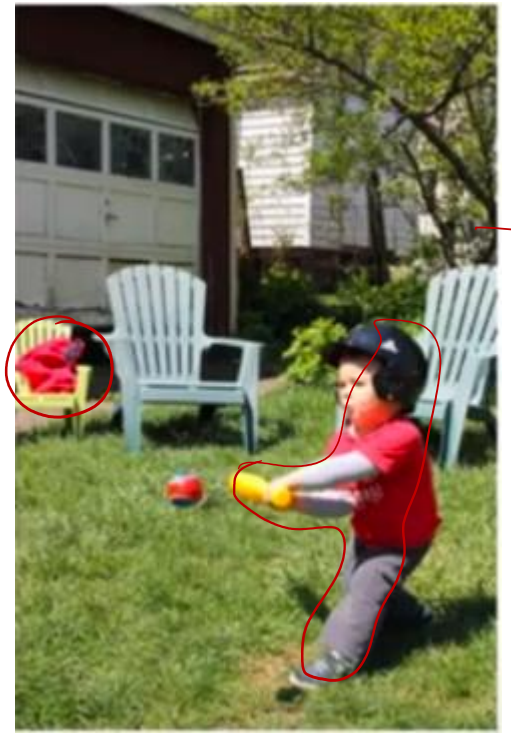
*Equal contribution. Listing order is random. Jakob proposed replacing RNNs with self-attention and started the effort to evaluate this idea. Ashish, with Illia, designed and implemented the first Transformer models and has been crucially involved in every aspect of this work. Noam proposed scaled dot-product attention, multi-head attention and the parameter-free position representation and became the other person involved in nearly every detail. Niki designed, implemented, tuned and evaluated countless model variants in our original codebase and tensor2tensor. Llion also experimented with novel model variants, was responsible for our initial codebase, and

MT and Language Models (Recap)

- Machine Translation: I like her → मैं उसे पसंद करता हूँ
- Language Model: I saw the man walking down the road
- Most of the tasks in NLP can be cast as some form of language modeling
- We can also design some architecture that uses pretty much exclusively some attention blocks and call that architecture a transformer

Attention

- Attention in Computer Vision
  - Used to highlight important parts of an Image that contribute to a desired output



- Attention in NLP
  - Aligned Machine Translation
  - Language Modeling

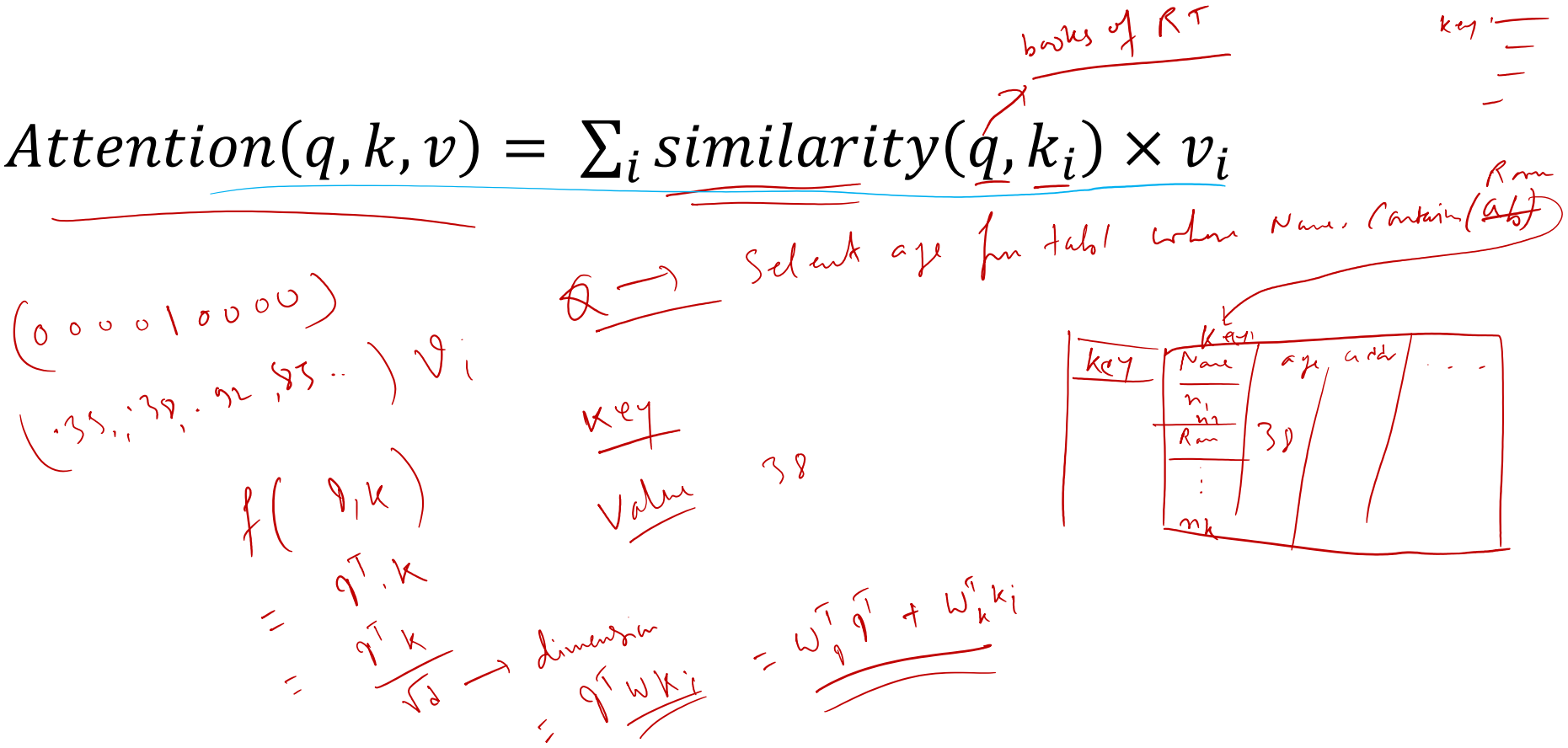
RNN and Transformers

Challenges with RNN	Transformer Networks
Long range dependencies	Facilitates long Range dependencies
Gradient Vanishing and Explosion	No Gradient Vanishing and Explosion
Large Number of Training Steps	Relatively Few Training Steps
Recurrence Prevents Parallel Computations	No recurrence → helps parallel processing

Attention Mechanism

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

- $Attention(q, k, v) = \sum_i similarity(q, k_i) \times v_i$

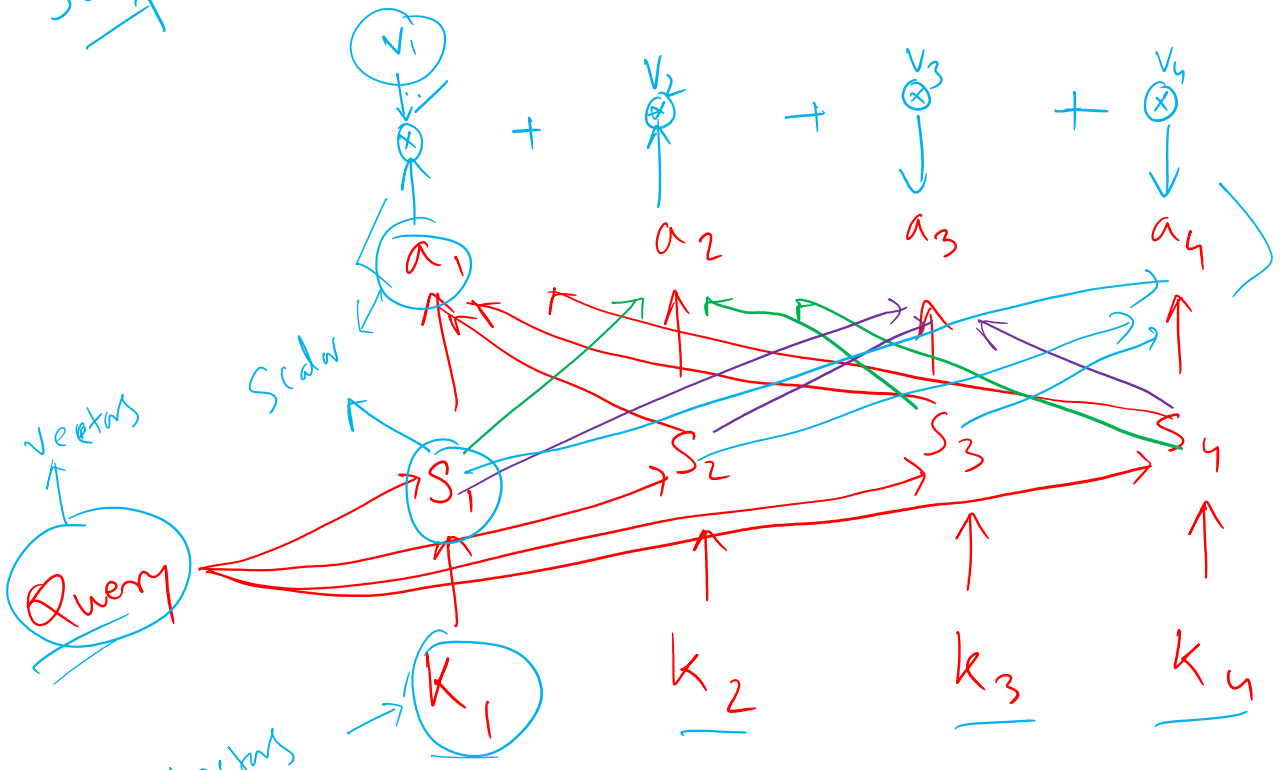


Self Attention

w $\hat{q}^{(i)}$

$$\sum (q^T k_i) \times v_i \longrightarrow \text{Vector}$$

$$a_i = \frac{\exp(s_i)}{\sum_j \exp(s_j)} \longrightarrow \text{Scalar}$$



$$s_i = f(q, k_i)$$

Similarity:

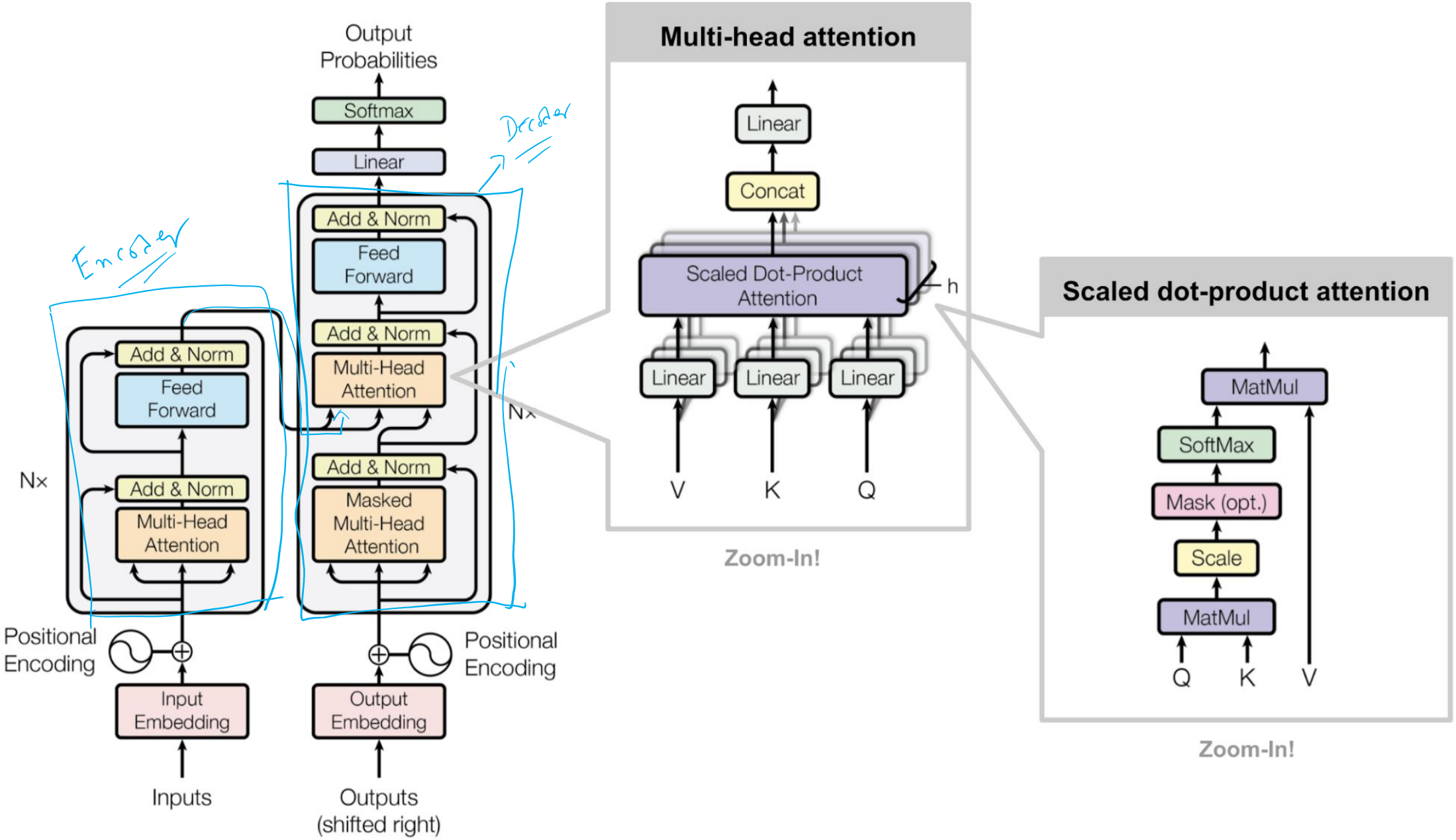
- $= q^T k_i$ → dot product
- $= \frac{q^T k_i}{\sqrt{d}}$ → Scaled dot product
- $= q^T W k_i$
- $= w_q^T q^T + w_k k_i$ → additive Sim.

Attention Mechanism

- Neural Architecture

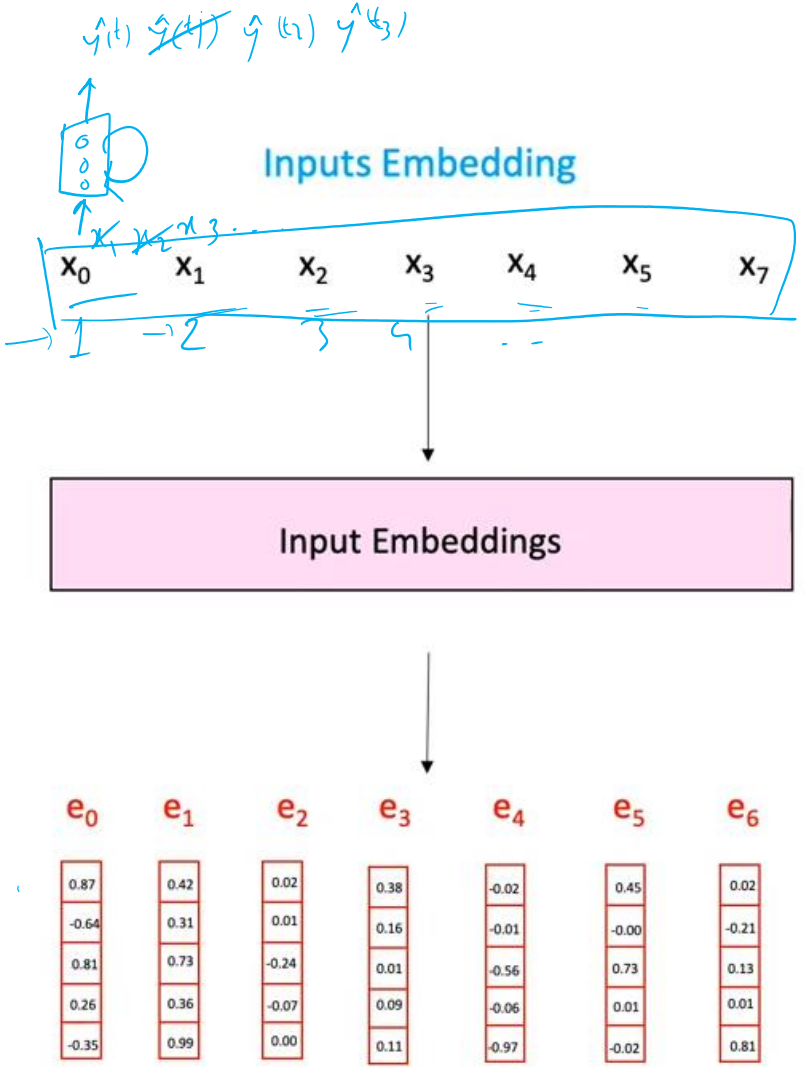
- Example:
  - Machine Translation
    - Query: S_i hidden vector for the ith output word
    - Key: h_j hidden vector for the jth input word
    - Value: h_j hidden vector for the jth input word

The Transformer Network


Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Nx
Nx
Positional Encoding
Positional Encoding
Input Embedding
Output Embedding
Inputs
Outputs (shifted right)
Multi-head attention
Linear
Concat
Scaled Dot-Product Attention
h
Linear
Linear
Linear
V
K
Q
Zoom-In!
Scaled dot-product attention
MatMul
SoftMax
Mask (opt.)
Scale
MatMul
Q
K
V
Zoom-In!

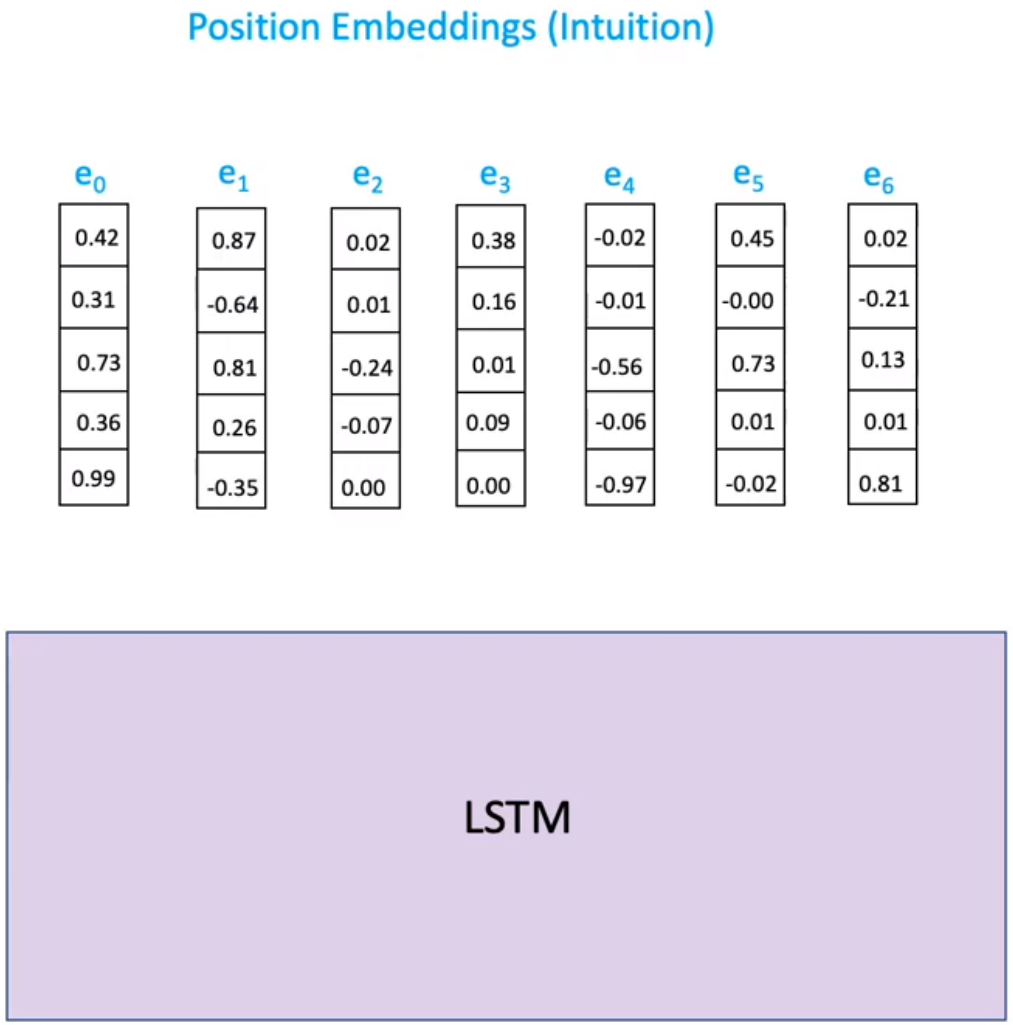
The Transformer Network

- Input Embedding



The Transformer Network

- Position Embedding



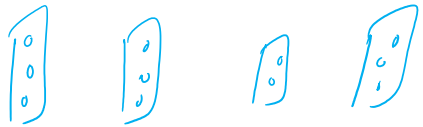
The Transformer Network

- Position Embedding

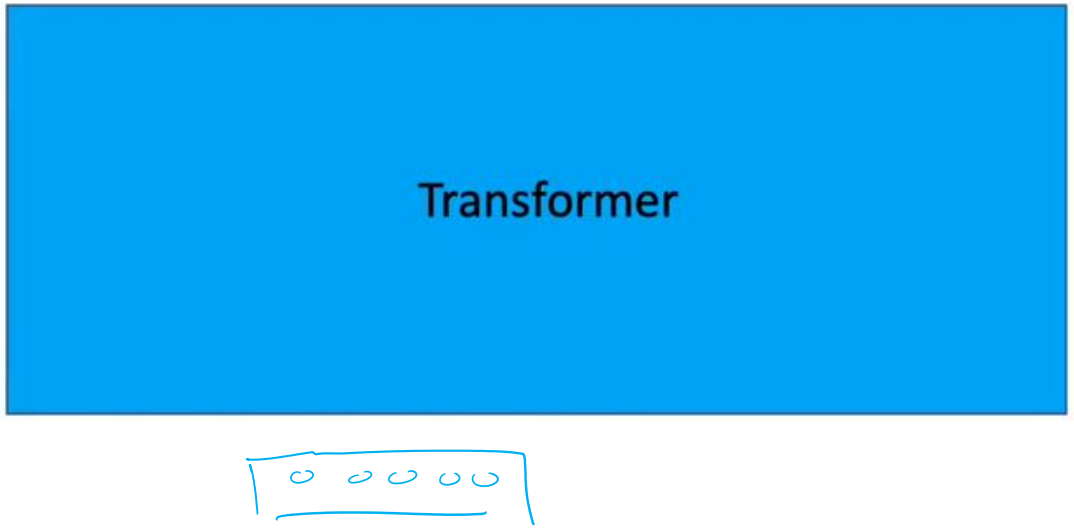
https://kazemnejad.com/blog/transformer_architecture_positional_encoding/

The Transformer Network

- Position Embedding



Even though she did not win the match, she was satisfied

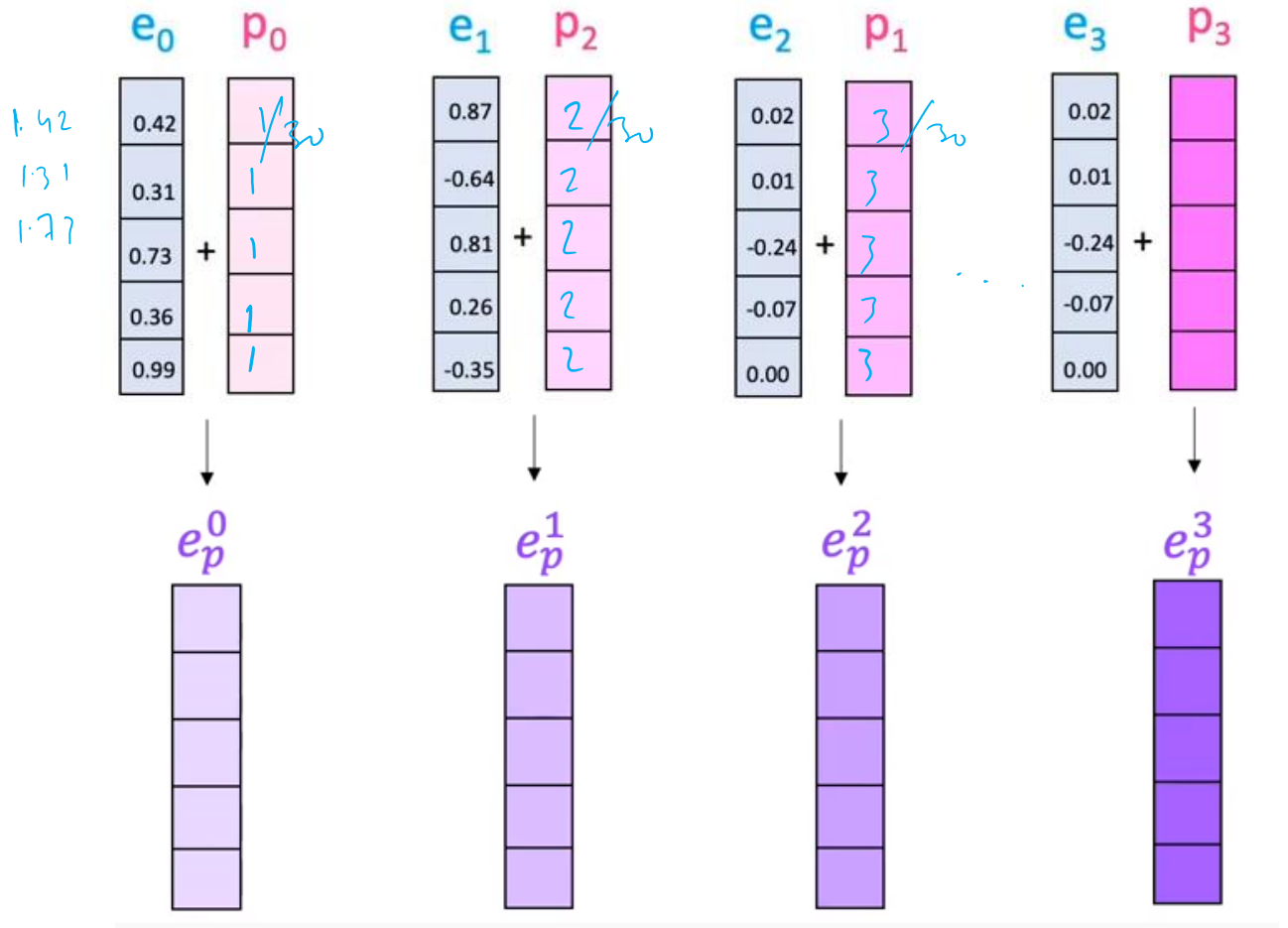


The Transformer Network

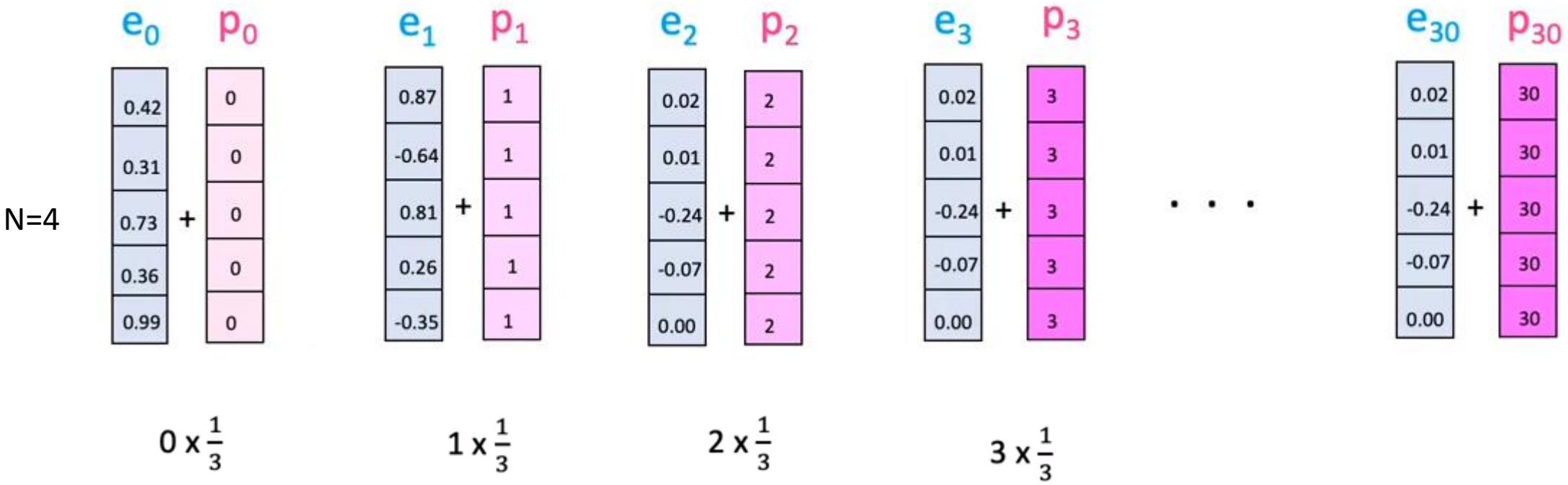
- Why Position Embedding matters?
  - Even though she did not win the match, she was satisfied
  - Even though she did win the match, she was not satisfied

The Transformer Network

- Position Embedding



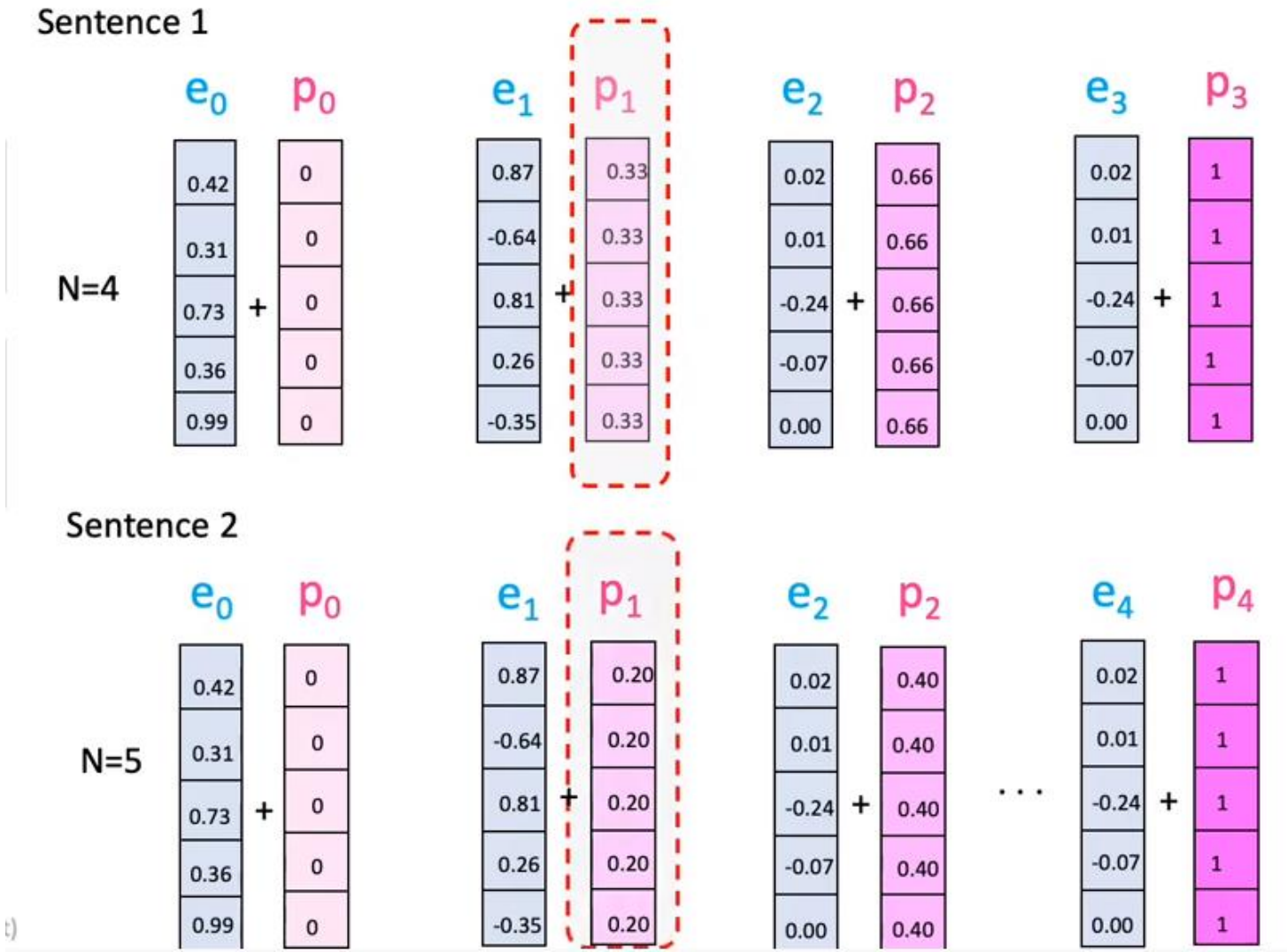
The Transformer Network: Position Embeddings



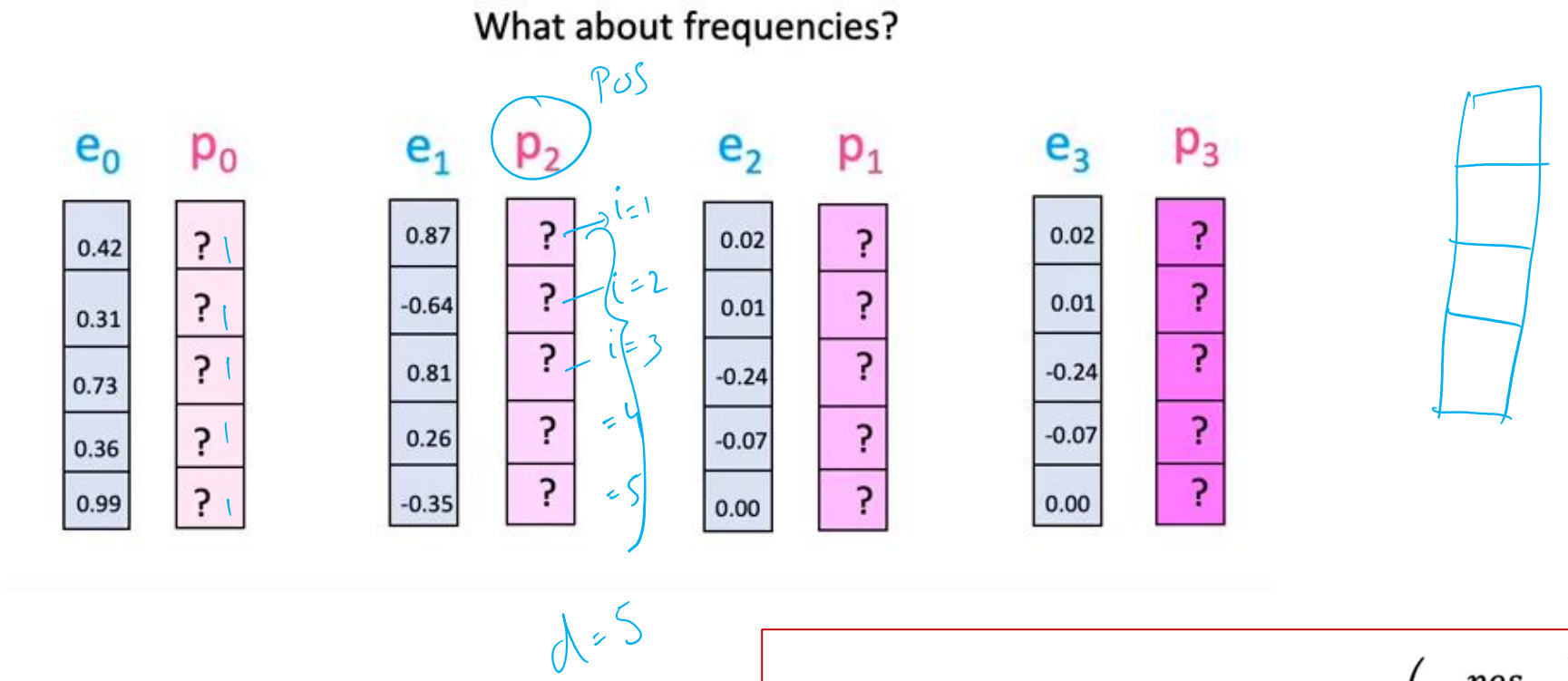
$$\frac{1}{N-1} = \frac{1}{3}$$

The Transformer Network: Position Embeddings

The Transformer Network: Position Embeddings



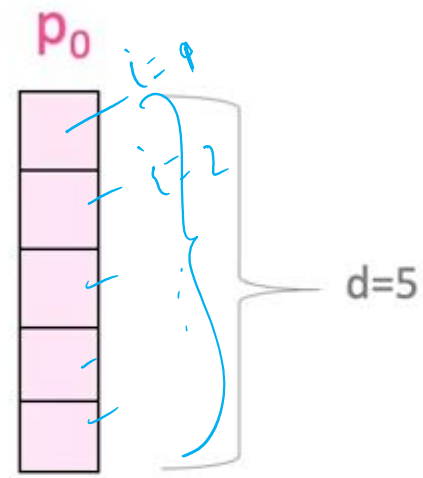
The Transformer Network: Position Embeddings



Vaswani et al 2017

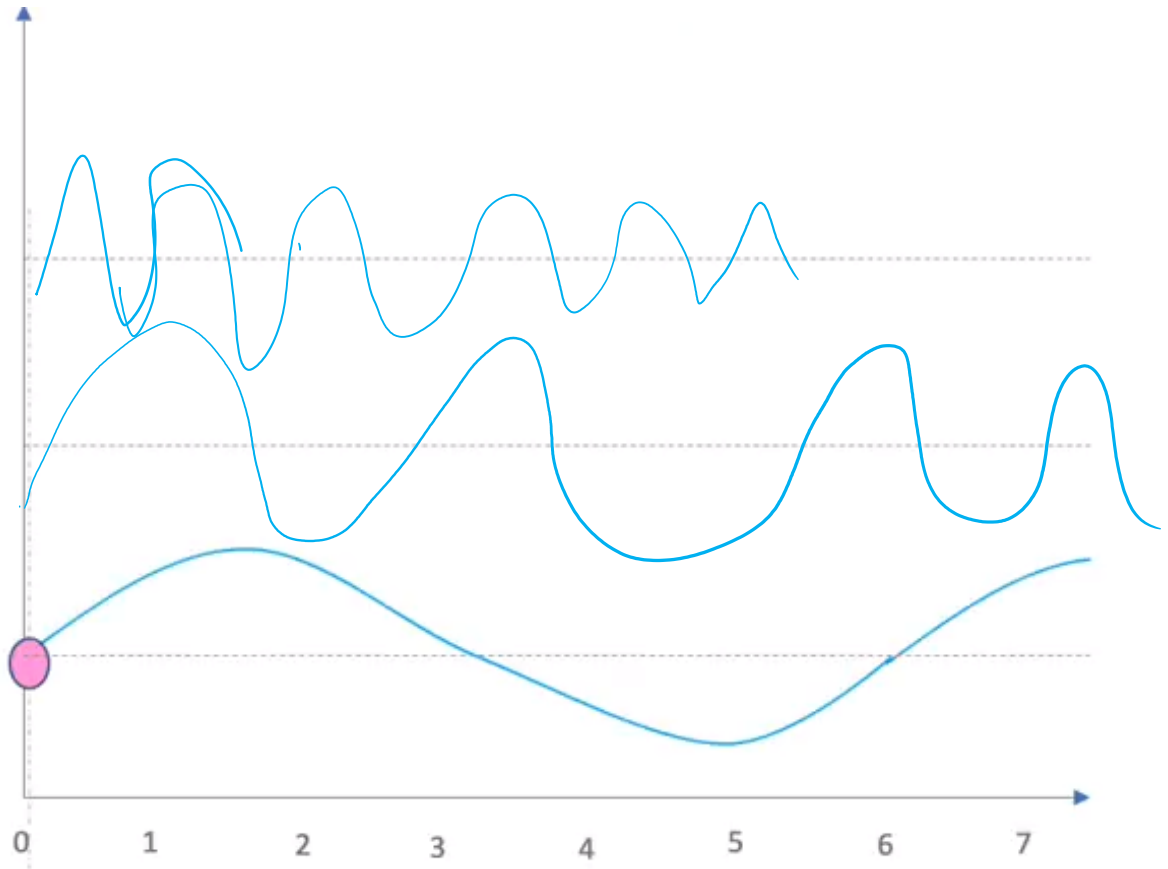
$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{\frac{2i}{d}}}\right)$$

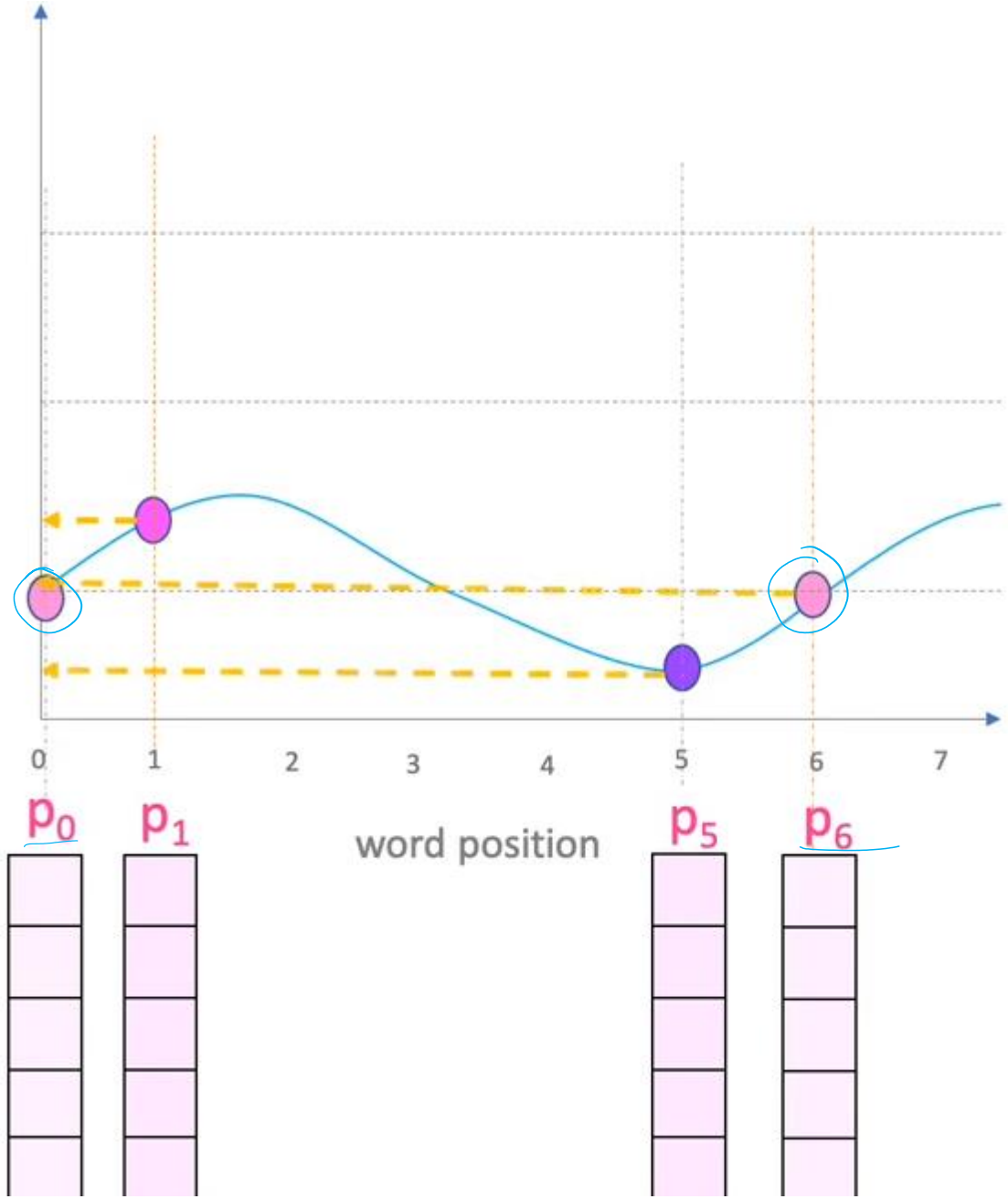
$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{\frac{2i}{d}}}\right)$$

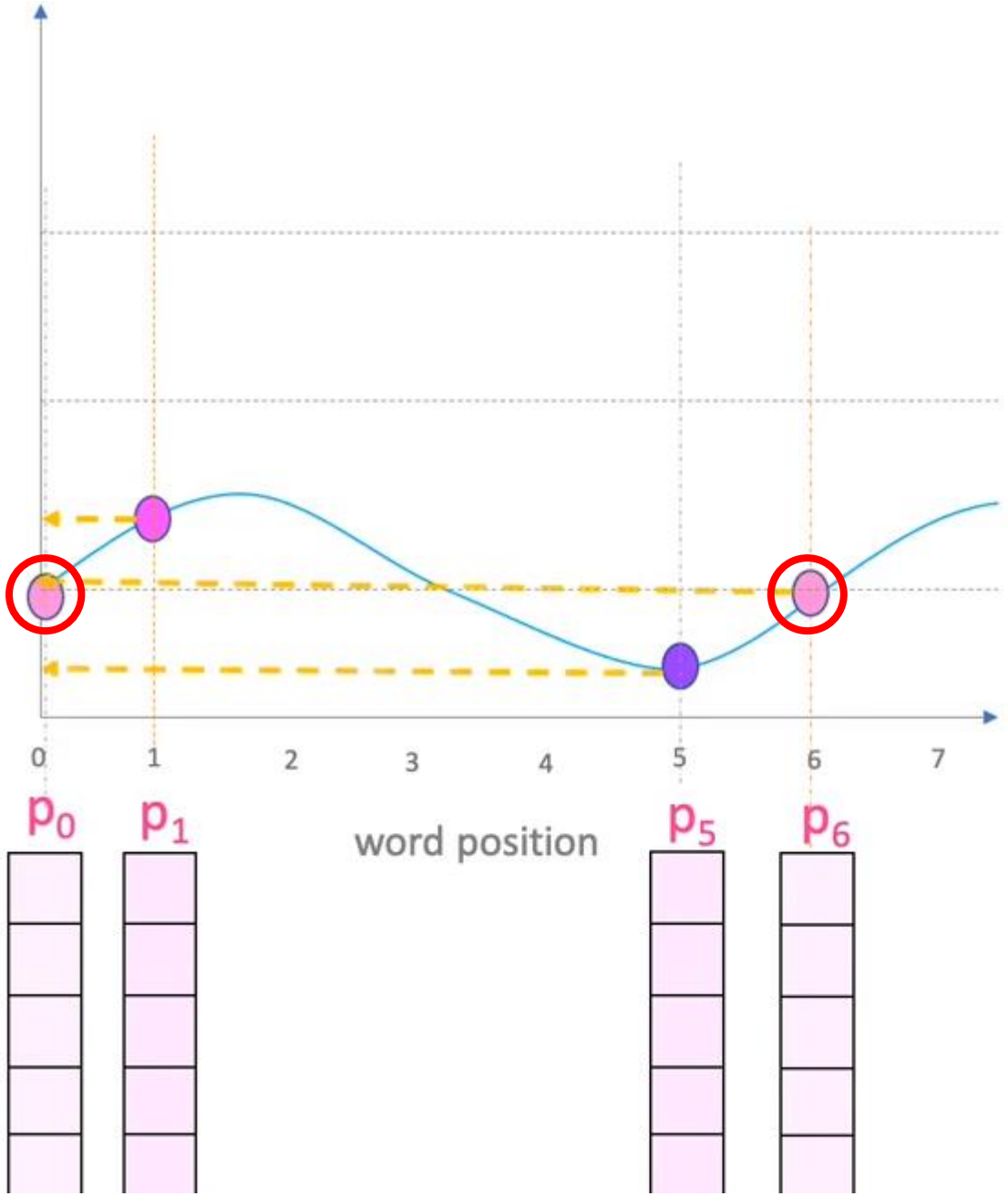


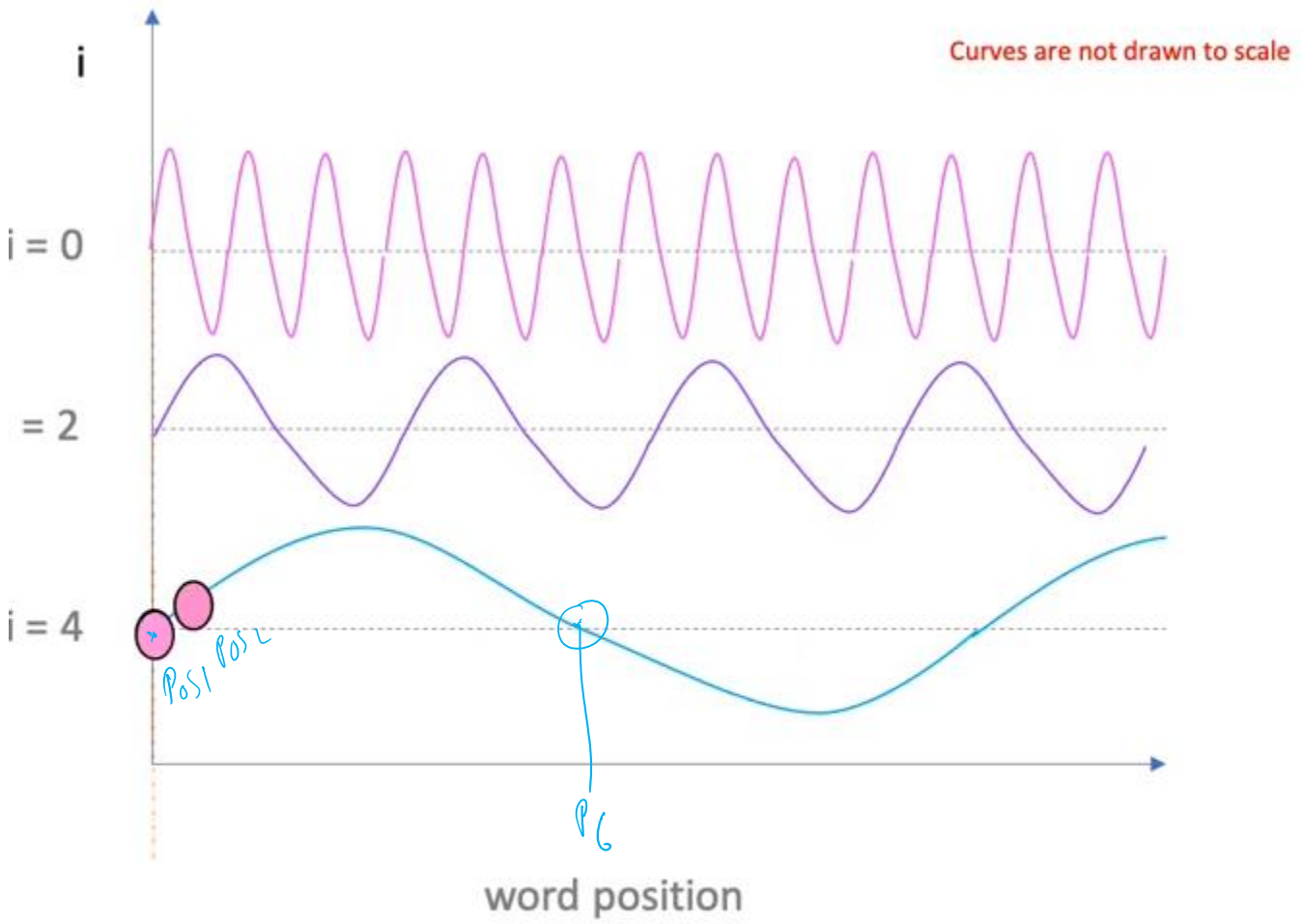
$pos = 0$

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{\frac{2i}{d}}}\right)$$



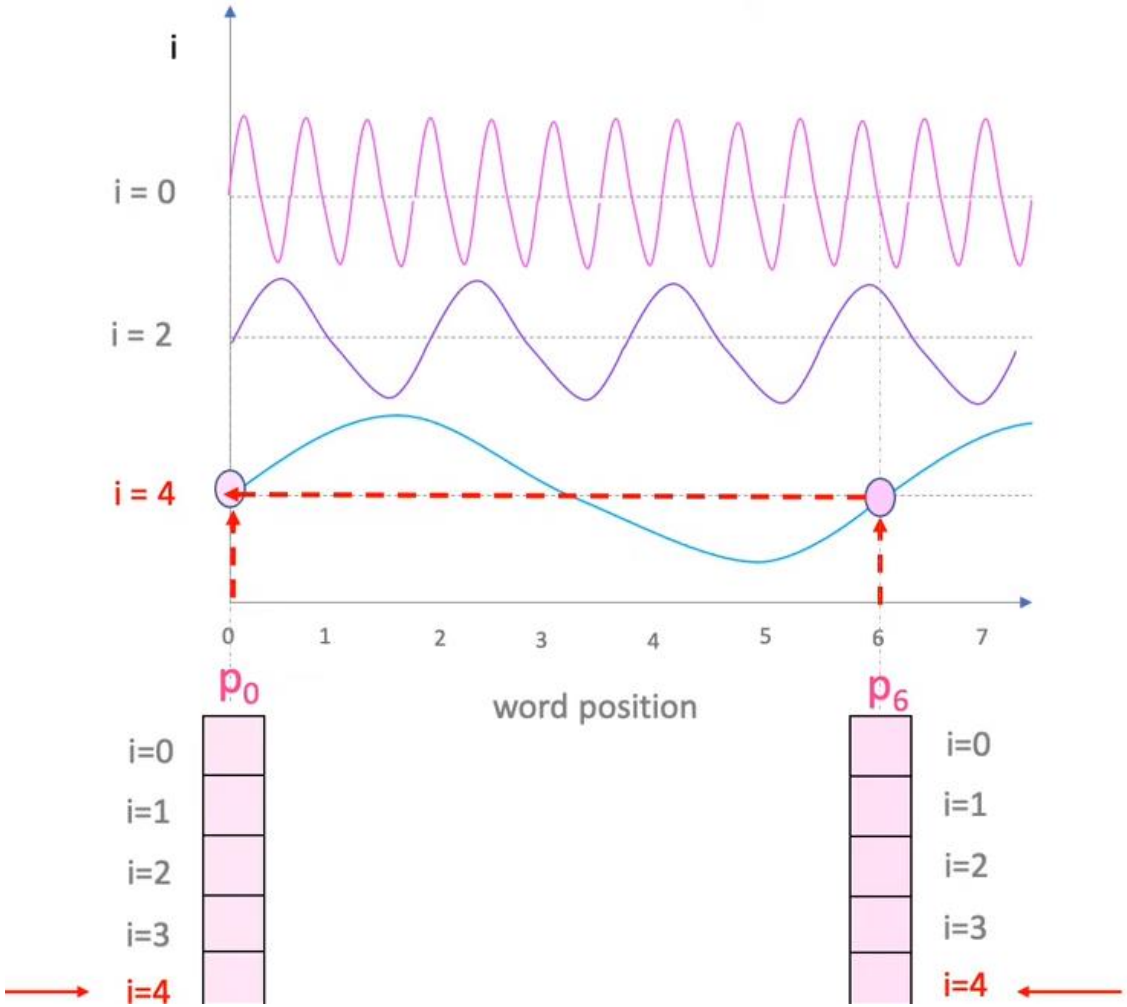

0
1
2
3
4
5
6
7
p_0
p_1
word position
p_5
p_6

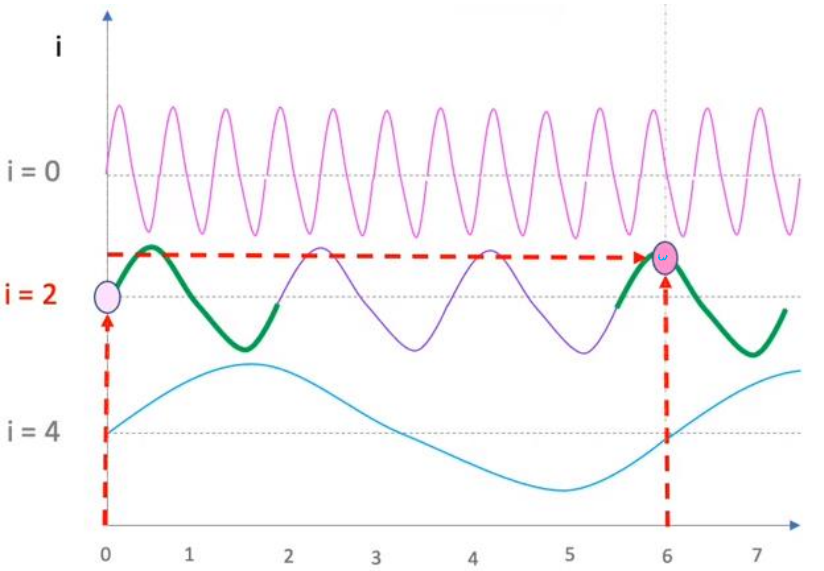

0
1
2
3
4
5
6
7
p_0
p_1
word position
p_5
p_6

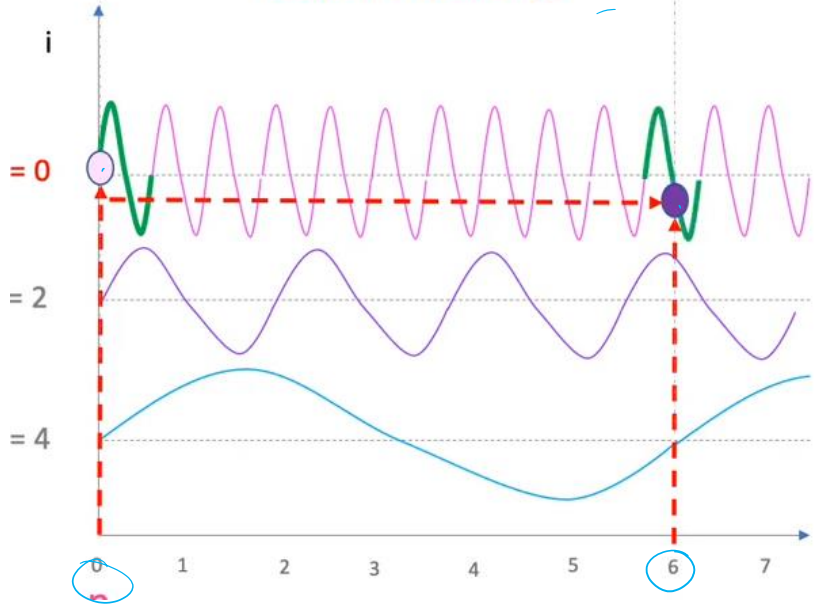


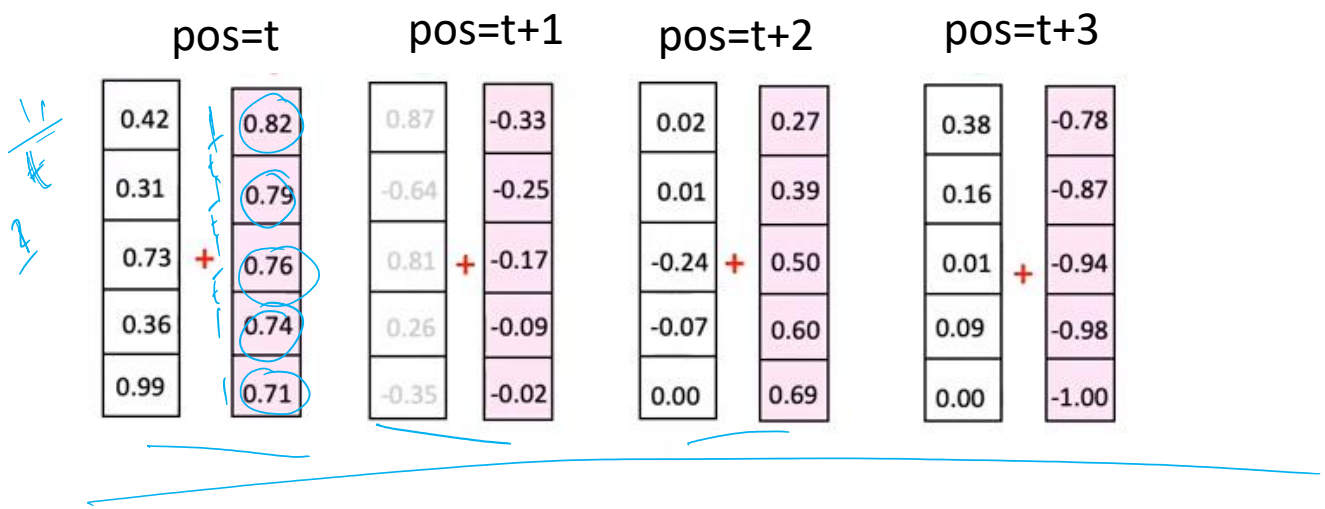
$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{\frac{2i}{d}}}\right)$$

i


i
i = 0
i = 2
i = 4
0 1 2 3 4 5 6 7
word position
p_0
p_6
i=0
i=1
i=2
i=3
i=4
i=0
i=1
i=2
i=3
i=4


i
i = 0
i = 2
i = 4
0 1 2 3 4 5 6 7

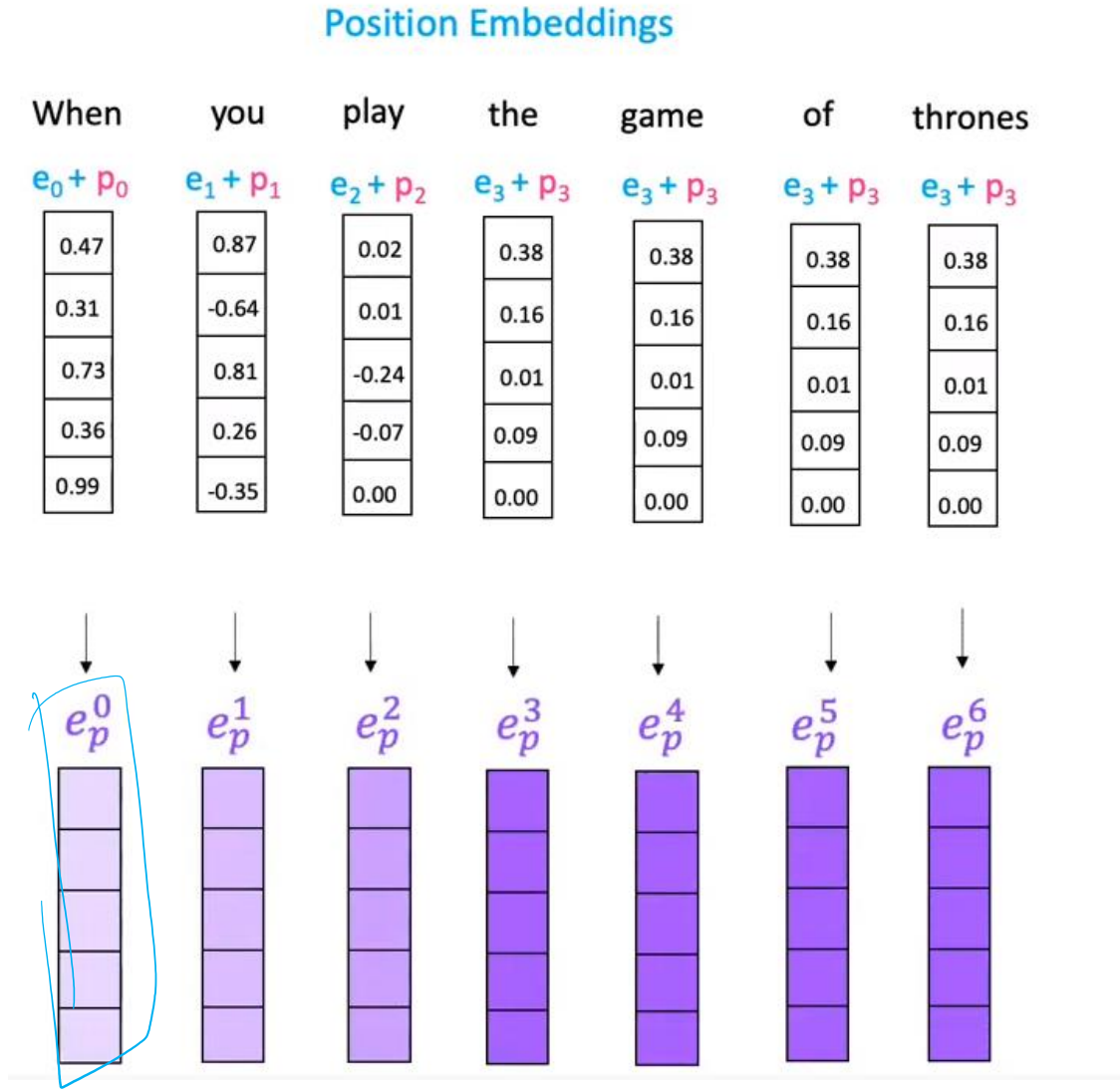

i
= 0
= 2
= 4
0 1 2 3 4 5 6 7

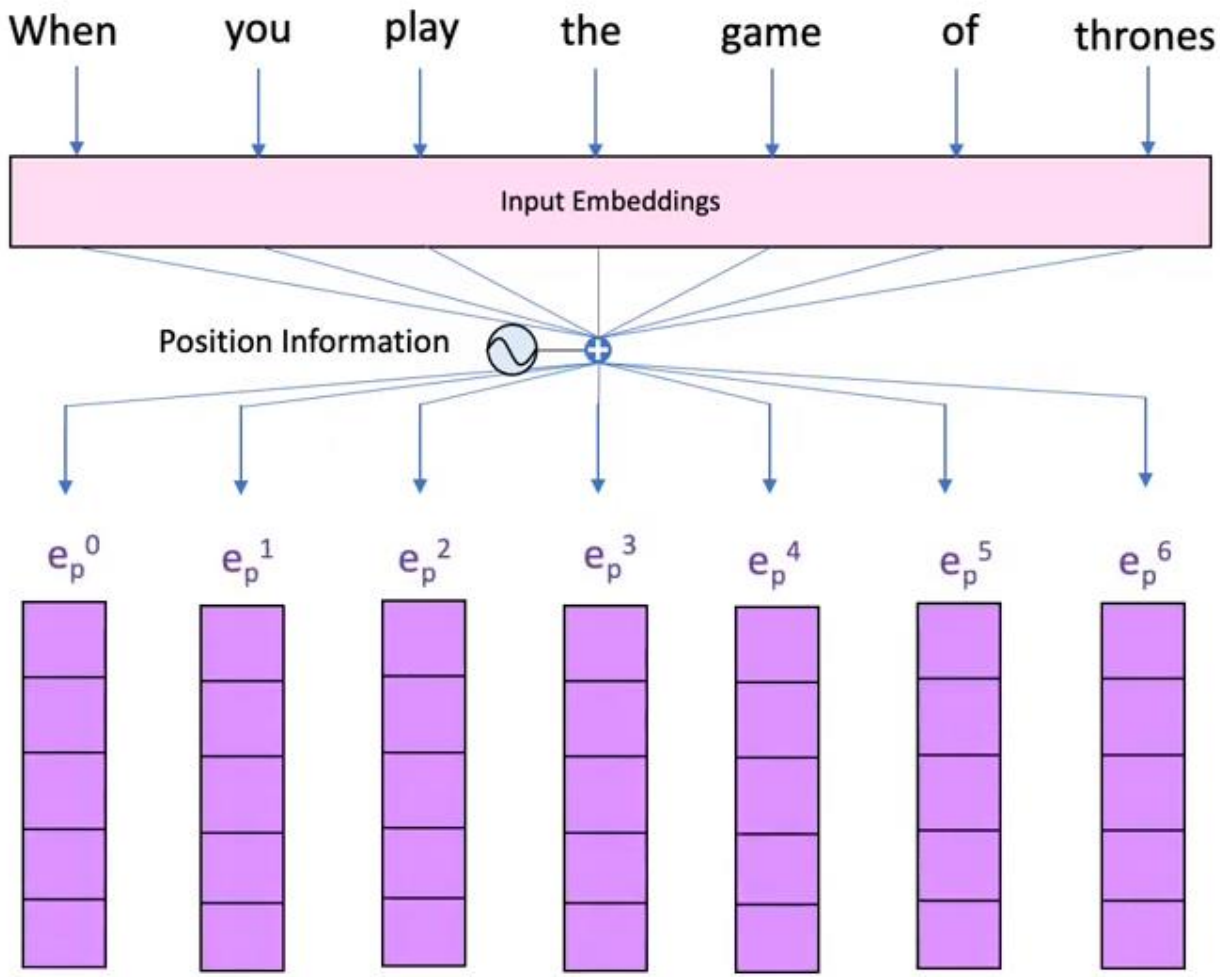


$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{\frac{2i}{d}}}\right)$$

$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{\frac{2i}{d}}}\right)$$

$$\overrightarrow{p_t} = \begin{bmatrix} \sin(\omega_1 . t) \\ \cos(\omega_1 . t) \\ \\ \sin(\omega_2 . t) \\ \cos(\omega_2 . t) \\ \\ \vdots \\ \\ \sin(\omega_{d/2} . t) \\ \cos(\omega_{d/2} . t) \end{bmatrix}_{d \times 1}$$


Position Embeddings
When
you
play
the
game
of
thrones
$e_0 + p_0$
$e_1 + p_1$
$e_2 + p_2$
$e_3 + p_3$
$e_3 + p_3$
$e_3 + p_3$
$e_3 + p_3$
0.47
0.31
0.73
0.36
0.99
0.87
-0.64
0.81
0.26
-0.35
0.02
0.01
-0.24
-0.07
0.00
0.38
0.16
0.01
0.09
0.00
0.38
0.16
0.01
0.09
0.00
0.38
0.16
0.01
0.09
0.00
0.38
0.16
0.01
0.09
0.00
e_p^0
e_p^1
e_p^2
e_p^3
e_p^4
e_p^5
e_p^6


When
you
play
the
game
of
thrones
Input Embeddings
Position Information
e_p^0
e_p^1
e_p^2
e_p^3
e_p^4
e_p^5
e_p^6

Query, Key & Value

Database

{Key, Value store}


```
{"order_100": {"items":"a1", "delivery_date":"a2", ...}},
{"order_101": {"items":"b1", "delivery_date":"b2", ...}},
{"order_102": {"items":"c1", "delivery_date":"c2", ...}},
{"order_103": {"items":"d1", "delivery_date":"d2", ...}},
{"order_104": {"items":"e1", "delivery_date":"e2", ...}},
{"order_105": {"items":"f1", "delivery_date":"f2", ...}},
{"order_106": {"items":"g1", "delivery_date":"g2", ...}},
{"order_107": {"items":"h1", "delivery_date":"h2", ...}},
{"order_108": {"items":"i1", "delivery_date":"i2", ...}},
{"order_109": {"items":"j1", "delivery_date":"j2", ...}},
{"order_110": {"items":"k1", "delivery_date":"k2", ...}}
```

Query, Key & Value

{Query: “Order details of order_104”}

OR

{Query: “Order details of order_106”}

Database

{Key, Value store}


```
{"order_100": {"items":"a1", "delivery_date":"a2", ...}},
{"order_101": {"items":"b1", "delivery_date":"b2", ...}},
{"order_102": {"items":"c1", "delivery_date":"c2", ...}},
{"order_103": {"items":"d1", "delivery_date":"d2", ...}},
{"order_104": {"items":"e1", "delivery_date":"e2", ...}},
{"order_105": {"items":"f1", "delivery_date":"f2", ...}},
{"order_106": {"items":"g1", "delivery_date":"g2", ...}},
{"order_107": {"items":"h1", "delivery_date":"h2", ...}},
{"order_108": {"items":"i1", "delivery_date":"i2", ...}},
{"order_109": {"items":"j1", "delivery_date":"j2", ...}},
{"order_110": {"items":"k1", "delivery_date":"k2", ...}}
```

{Query: “Order details of order_104”}

OR

{Query: “Order details of order_106”}

{Key, Value store}


```
{"order_100": {"items":"a1", "delivery_date":"a2", ...}},
{"order_101": {"items":"b1", "delivery_date":"b2", ...}},
{"order_102": {"items":"c1", "delivery_date":"c2", ...}},
{"order_103": {"items":"d1", "delivery_date":"d2", ...}},
{"order_104": {"items":"e1", "delivery_date":"e2", ...}},
{"order_105": {"items":"f1", "delivery_date":"f2", ...}},
{"order_106": {"items":"g1", "delivery_date":"g2", ...}},
{"order_107": {"items":"h1", "delivery_date":"h2", ...}},
{"order_108": {"items":"i1", "delivery_date":"i2", ...}},
{"order_109": {"items":"j1", "delivery_date":"j2", ...}},
{"order_110": {"items":"k1", "delivery_date":"k2", ...}}
```

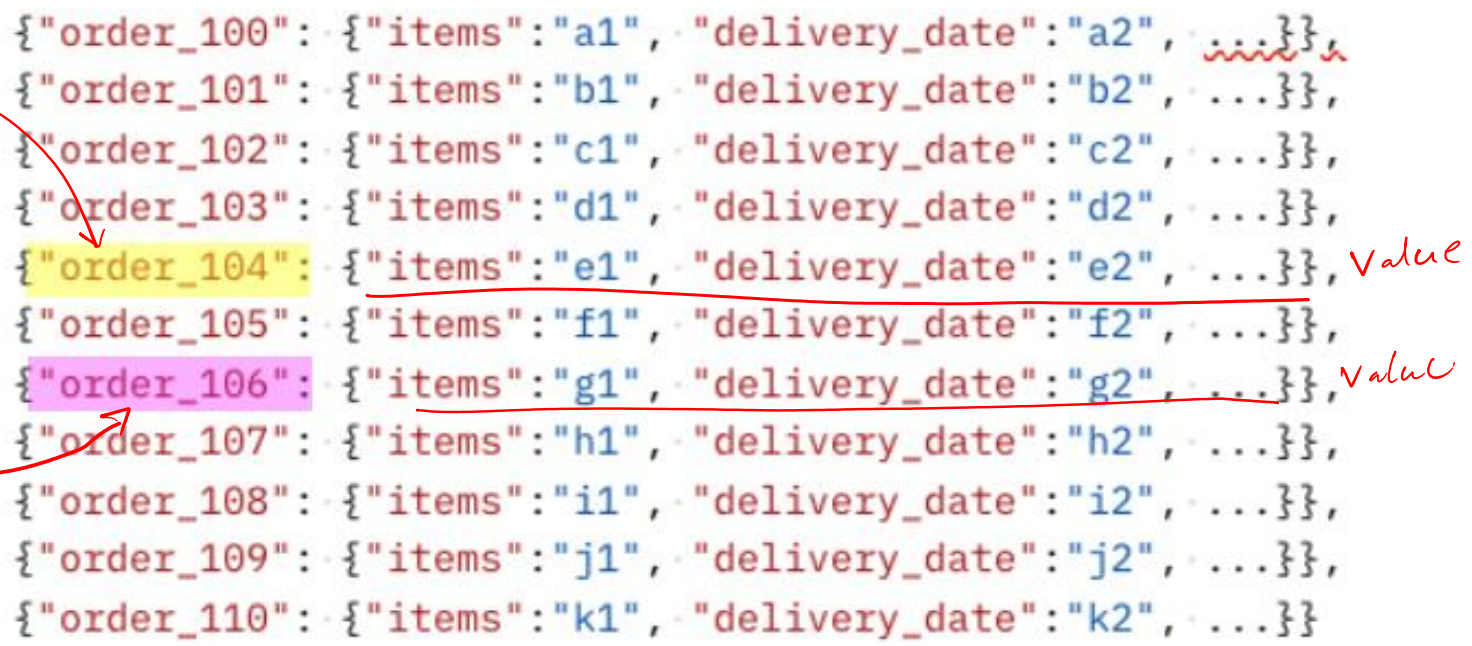
Query, Key & Value

{Query: “Order details of order_104”}

OR

{Query: “Order details of order_106”}

{Key, Value store}



Query, Key & Value

{Query: “Order details of order_104”}

OR

{Query: “Order details of order_106”}

{Key, Value store}


```
{"order_100": {"items":"a1", "delivery_date":"a2", ...}},
{"order_101": {"items":"b1", "delivery_date":"b2", ...}},
{"order_102": {"items":"c1", "delivery_date":"c2", ...}},
{"order_103": {"items":"d1", "delivery_date":"d2", ...}},
{"order_104": {"items":"e1", "delivery_date":"e2", ...}},
{"order_105": {"items":"f1", "delivery_date":"f2", ...}},
{"order_106": {"items":"g1", "delivery_date":"g2", ...}},
{"order_107": {"items":"h1", "delivery_date":"h2", ...}},
{"order_108": {"items":"i1", "delivery_date":"i2", ...}},
{"order_109": {"items":"j1", "delivery_date":"j2", ...}},
{"order_110": {"items":"k1", "delivery_date":"k2", ...}}
```

Query, Key & Value

Done at the same time !!

{Query: “Order details of order_104”}

OR

{Query: “Order details of order_106”}

{Key, Value store}


```
{"order_100": {"items":"a1", "delivery_date":"a2", ...}},
{"order_101": {"items":"b1", "delivery_date":"b2", ...}},
{"order_102": {"items":"c1", "delivery_date":"c2", ...}},
{"order_103": {"items":"d1", "delivery_date":"d2", ...}},
{"order_104": {"items":"e1", "delivery_date":"e2", ...}},
{"order_105": {"items":"f1", "delivery_date":"f2", ...}},
{"order_106": {"items":"g1", "delivery_date":"g2", ...}},
{"order_107": {"items":"h1", "delivery_date":"h2", ...}},
{"order_108": {"items":"i1", "delivery_date":"i2", ...}},
{"order_109": {"items":"j1", "delivery_date":"j2", ...}},
{"order_110": {"items":"k1", "delivery_date":"k2", ...}}
```

Query, Key & Value

{Query: “Order details of order_104”}

OR

{Query: “Order details of order_106”}


```
{"order_100": {"items":"a1", "delivery_date":"a2", ...}},
{"order_101": {"items":"b1", "delivery_date":"b2", ...}},
{"order_102": {"items":"c1", "delivery_date":"c2", ...}},
{"order_103": {"items":"d1", "delivery_date":"d2", ...}},
{"order_104": {"items":"e1", "delivery_date":"e2", ...}},
{"order_105": {"items":"f1", "delivery_date":"f2", ...}},
{"order_106": {"items":"g1", "delivery_date":"g2", ...}},
{"order_107": {"items":"h1", "delivery_date":"h2", ...}},
{"order_108": {"items":"i1", "delivery_date":"i2", ...}},
{"order_109": {"items":"j1", "delivery_date":"j2", ...}},
{"order_110": {"items":"k1", "delivery_date":"k2", ...}}
```


Query

1. Search for info

Key

1. Interacts directly with Queries
2. Distinguishes one object from another
3. Identify which object is the most relevant and by how much

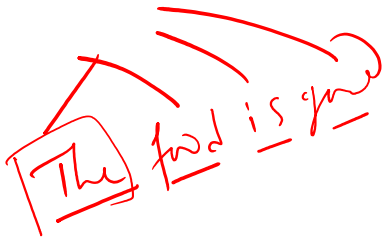
Value

1. Actual details of the object
2. More fine grained

Attention

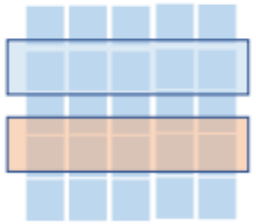
Query Key Value Store Key Value

Attention

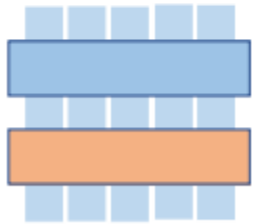


Query

Key Value Store



Key



Value

Attention

$$\alpha(t,t') \sum \left(\frac{\vec{Q}\cdot\vec{k}}{\sqrt{d}}\right)\vec{V}$$

Scaled dot-product.

Parallelizable !!!

Query

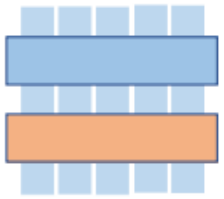
Q

Key Value Store

QK^T

Key

$softmax(\frac{QK^T}{\sqrt{d}})$

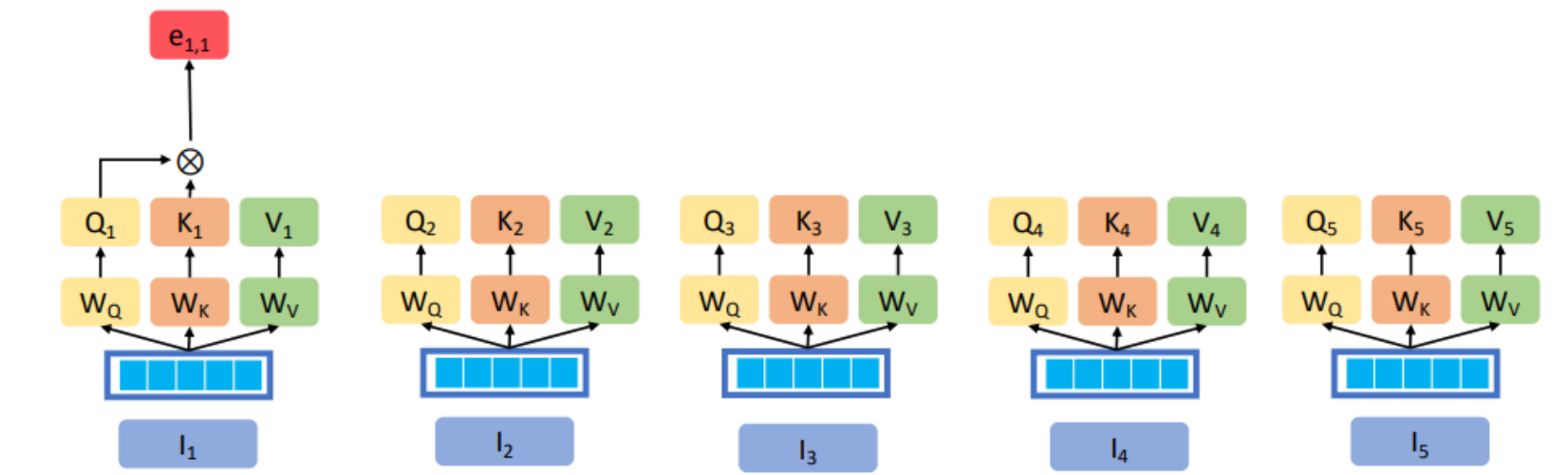


Value

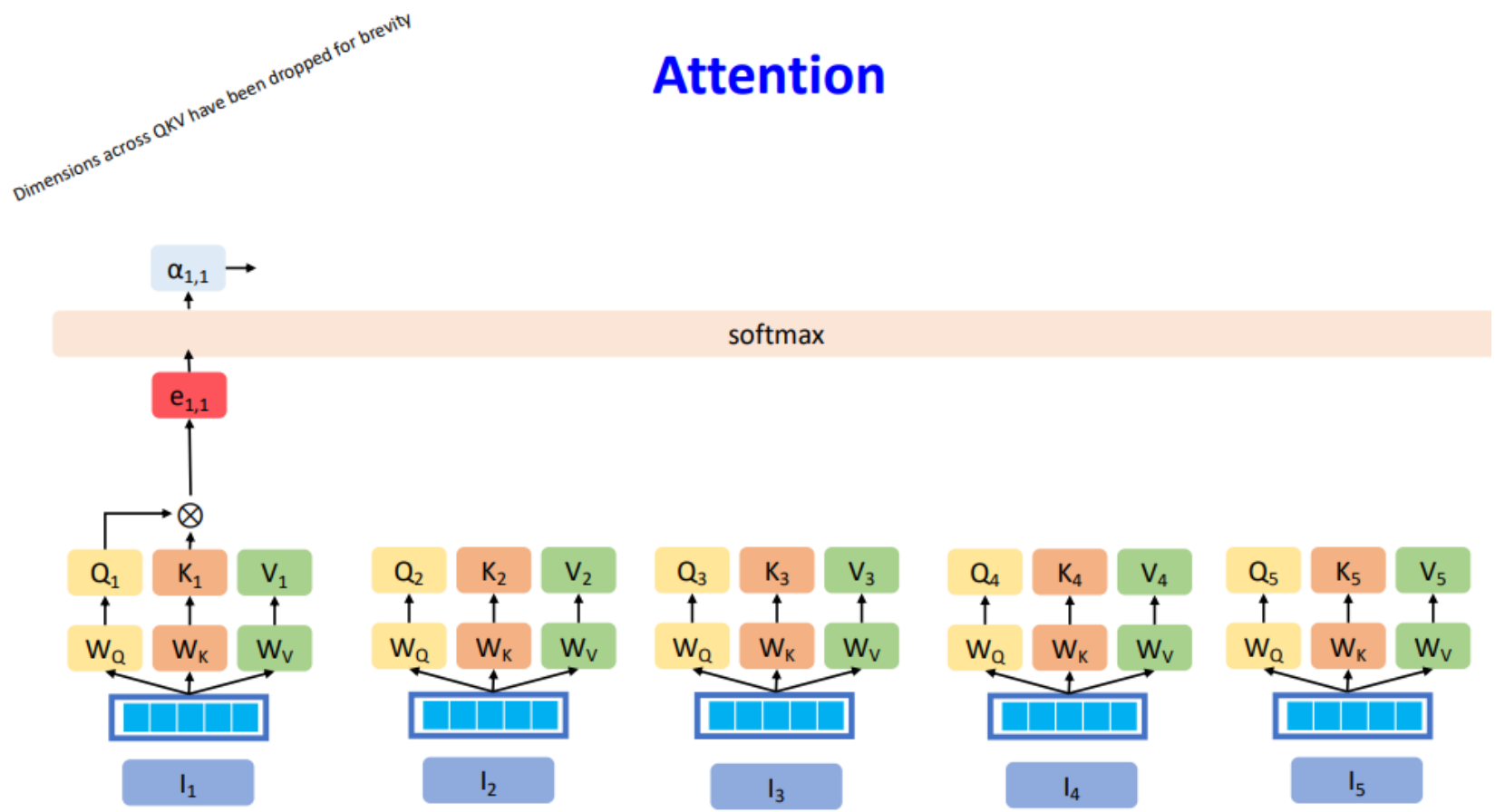
$softmax(\frac{QK^T}{\sqrt{d}})V$

Dimensions across QKV have been dropped for brevity

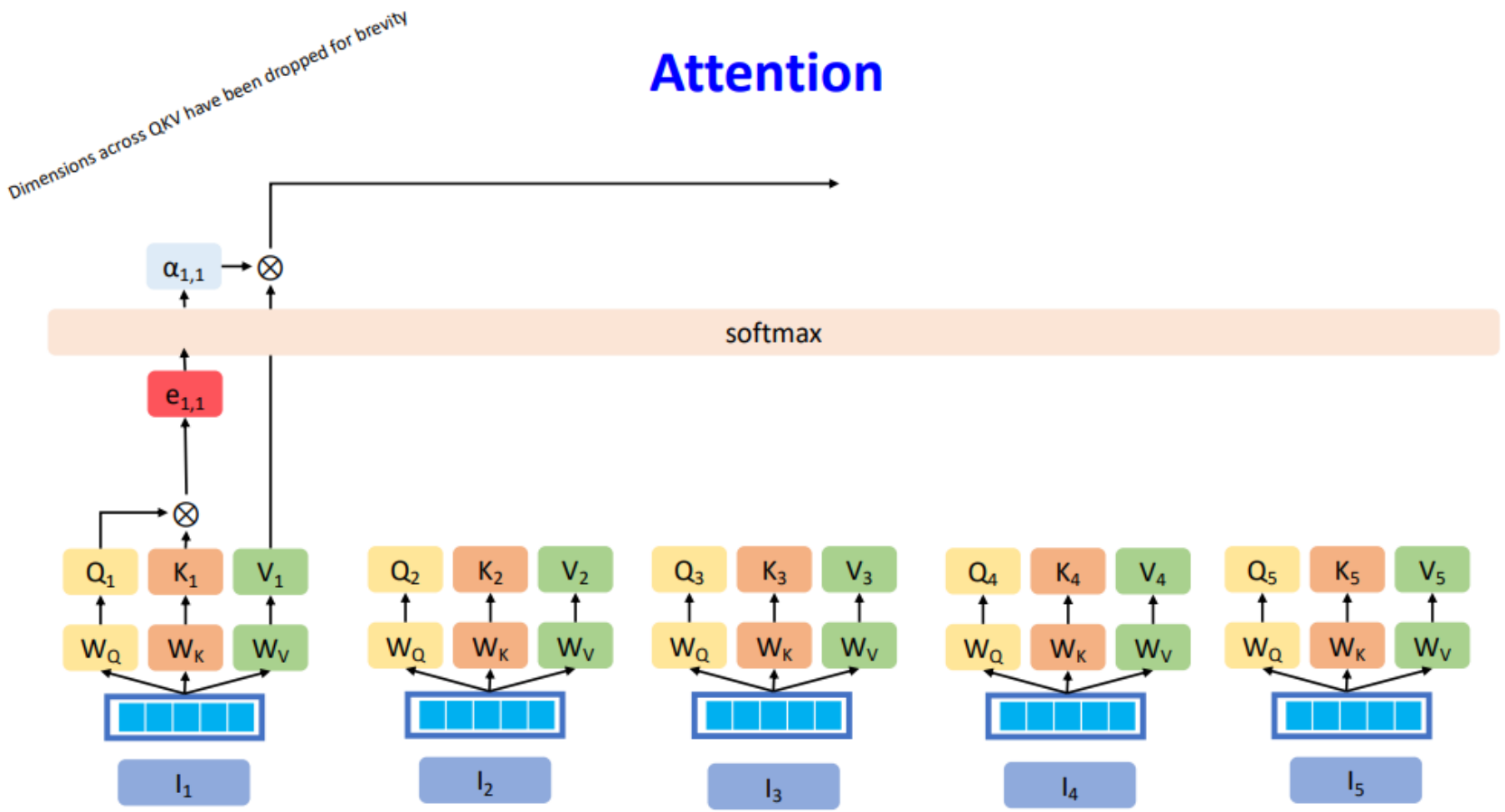
Attention



Attention

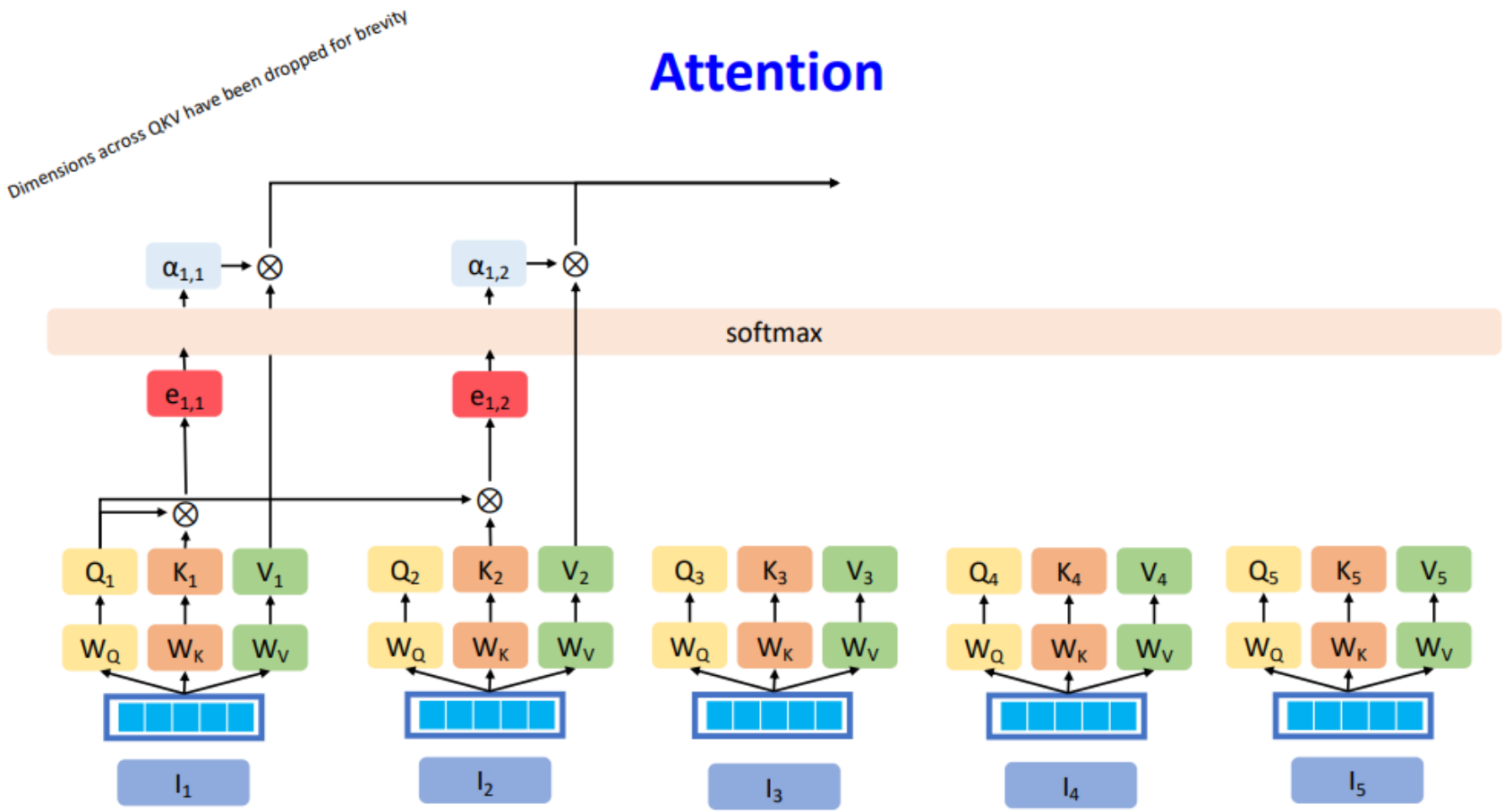

Dimensions across QKV have been dropped for brevity
$\alpha_{1,1}$
softmax
$e_{1,1}$
Q_1 K_1 V_1
Q_2 K_2 V_2
Q_3 K_3 V_3
Q_4 K_4 V_4
Q_5 K_5 V_5
W_Q W_K W_V
I_1 I_2 I_3 I_4 I_5

Attention

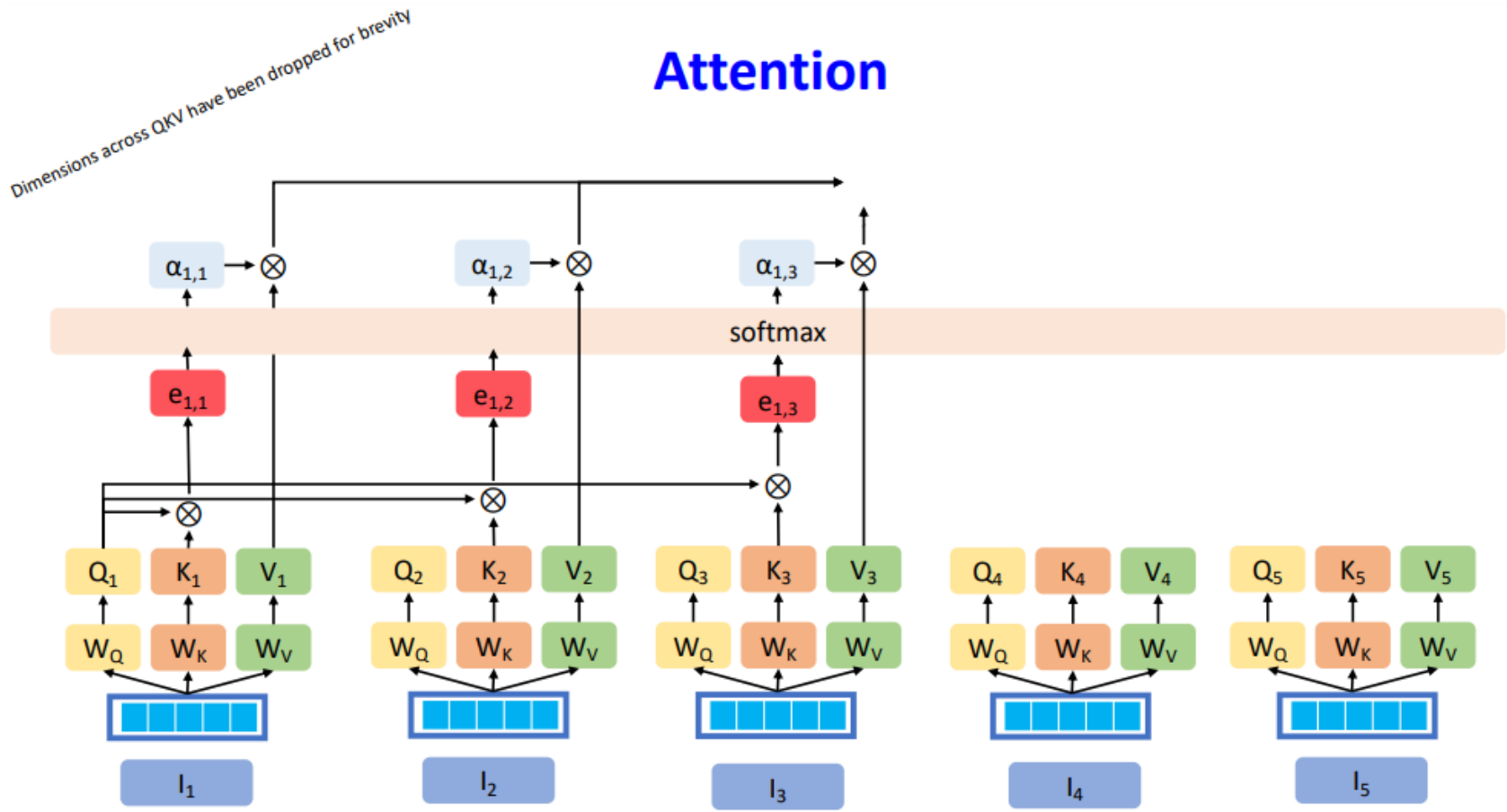


$A = (QK^T) \cdot V$

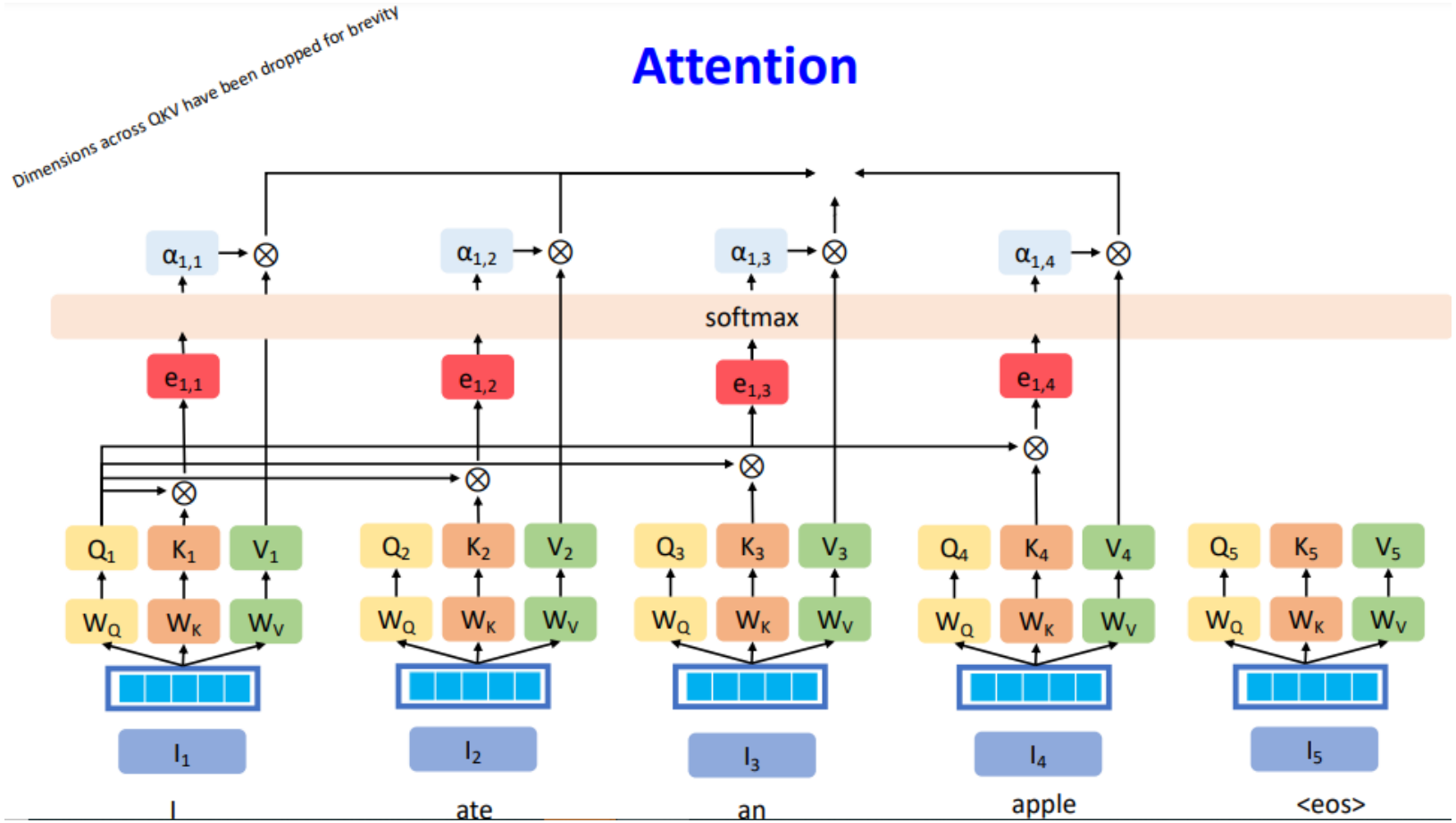
Attention


Dimensions across QKV have been dropped for brevity
$\alpha_{1,1}$
$\alpha_{1,2}$
softmax
$e_{1,1}$
$e_{1,2}$
Q_1 K_1 V_1
Q_2 K_2 V_2
Q_3 K_3 V_3
Q_4 K_4 V_4
Q_5 K_5 V_5
W_Q W_K W_V
I_1 I_2 I_3 I_4 I_5

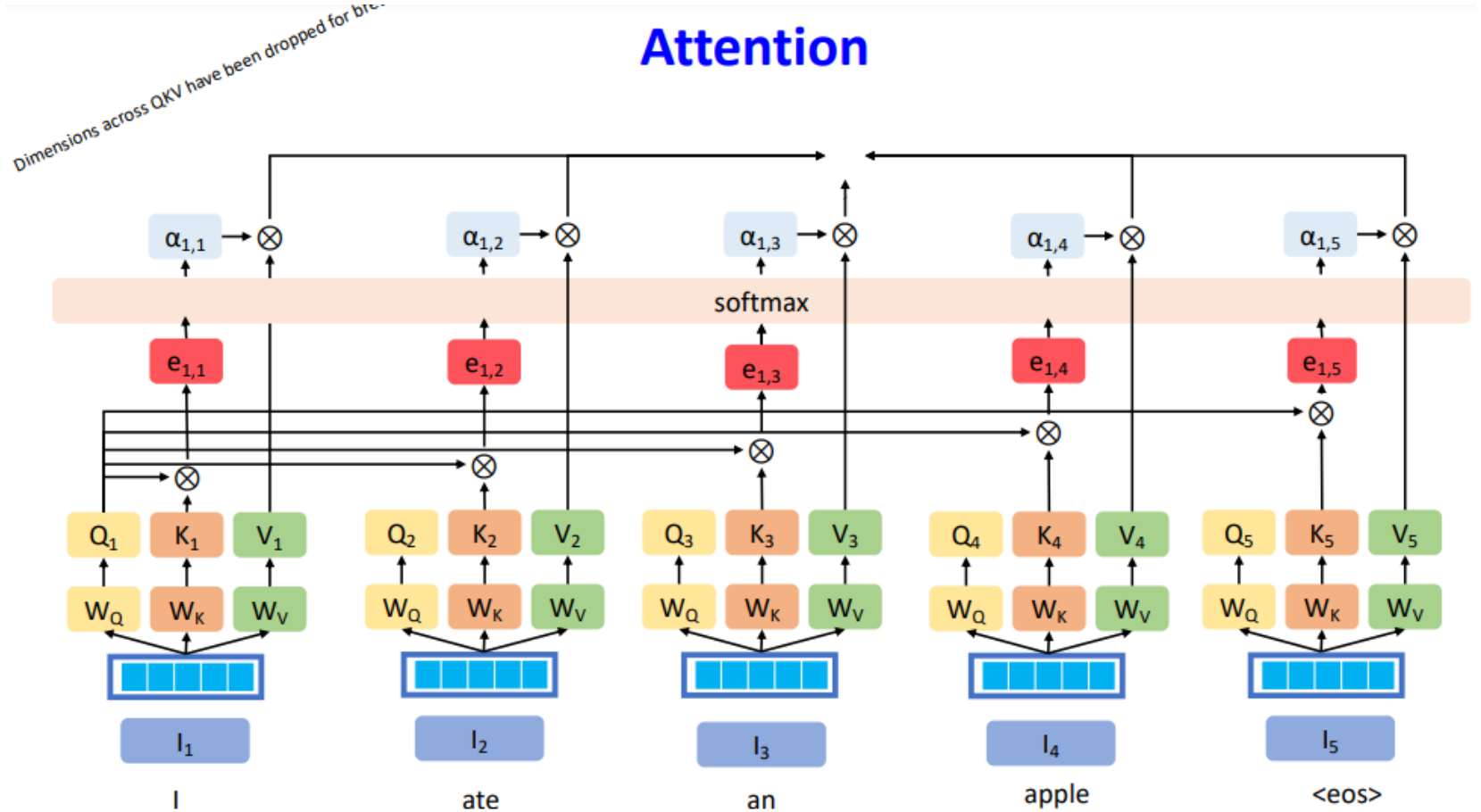
Attention


Dimensions across QKV have been dropped for brevity
α1,1
α1,2
α1,3
softmax
e1,1
e1,2
e1,3
Q1
K1
V1
Q2
K2
V2
Q3
K3
V3
Q4
K4
V4
Q5
K5
V5
WQ
WK
WV
I1
I2
I3
I4
I5

Attention

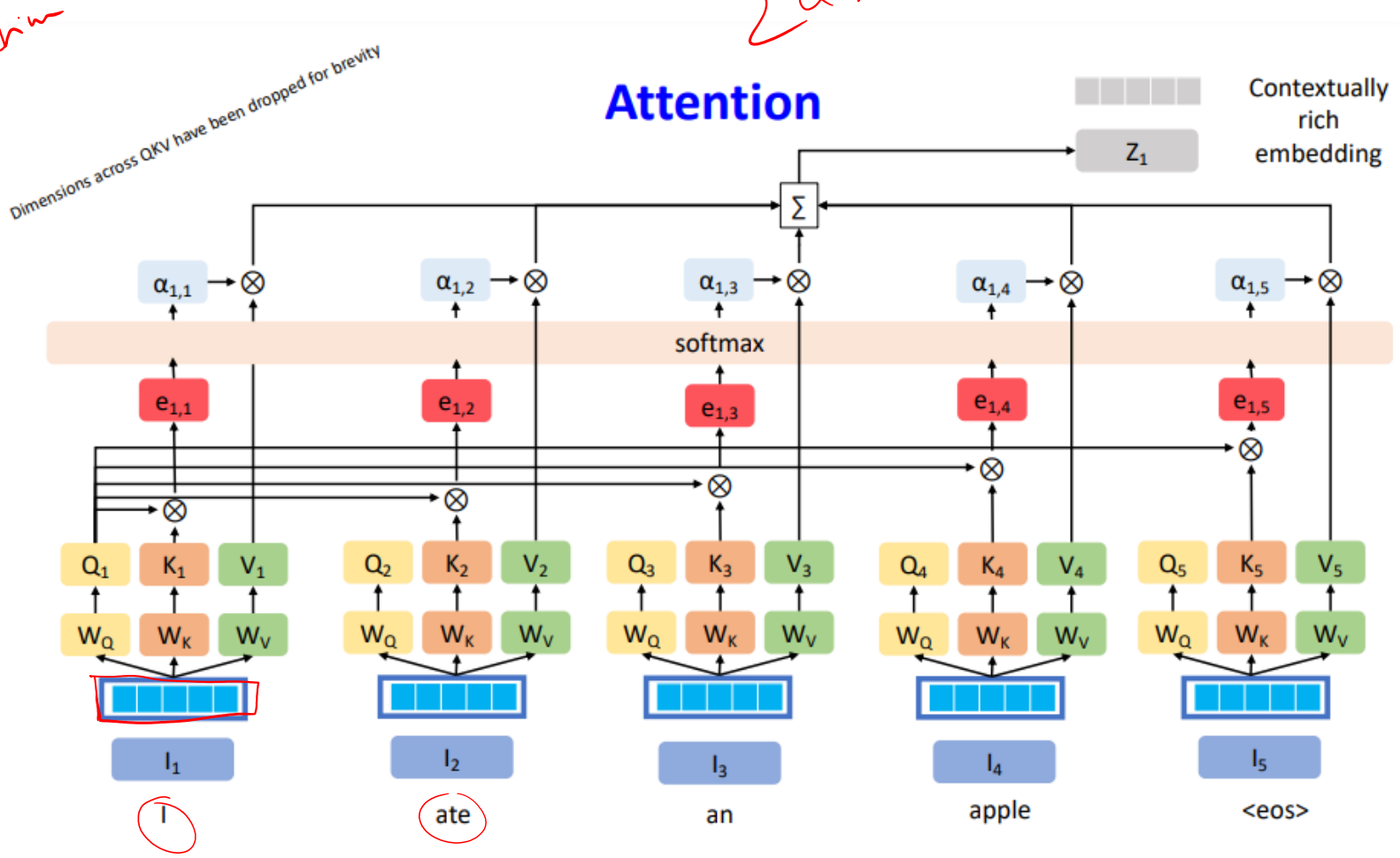

Dimensions across QKV have been dropped for brevity
α1,1
α1,2
α1,3
α1,4
softmax
e1,1
e1,2
e1,3
e1,4
Q1
K1
V1
Q2
K2
V2
Q3
K3
V3
Q4
K4
V4
Q5
K5
V5
WQ
WK
WV
I1
I2
I3
I4
I5
I
ate
an
apple
<eos>

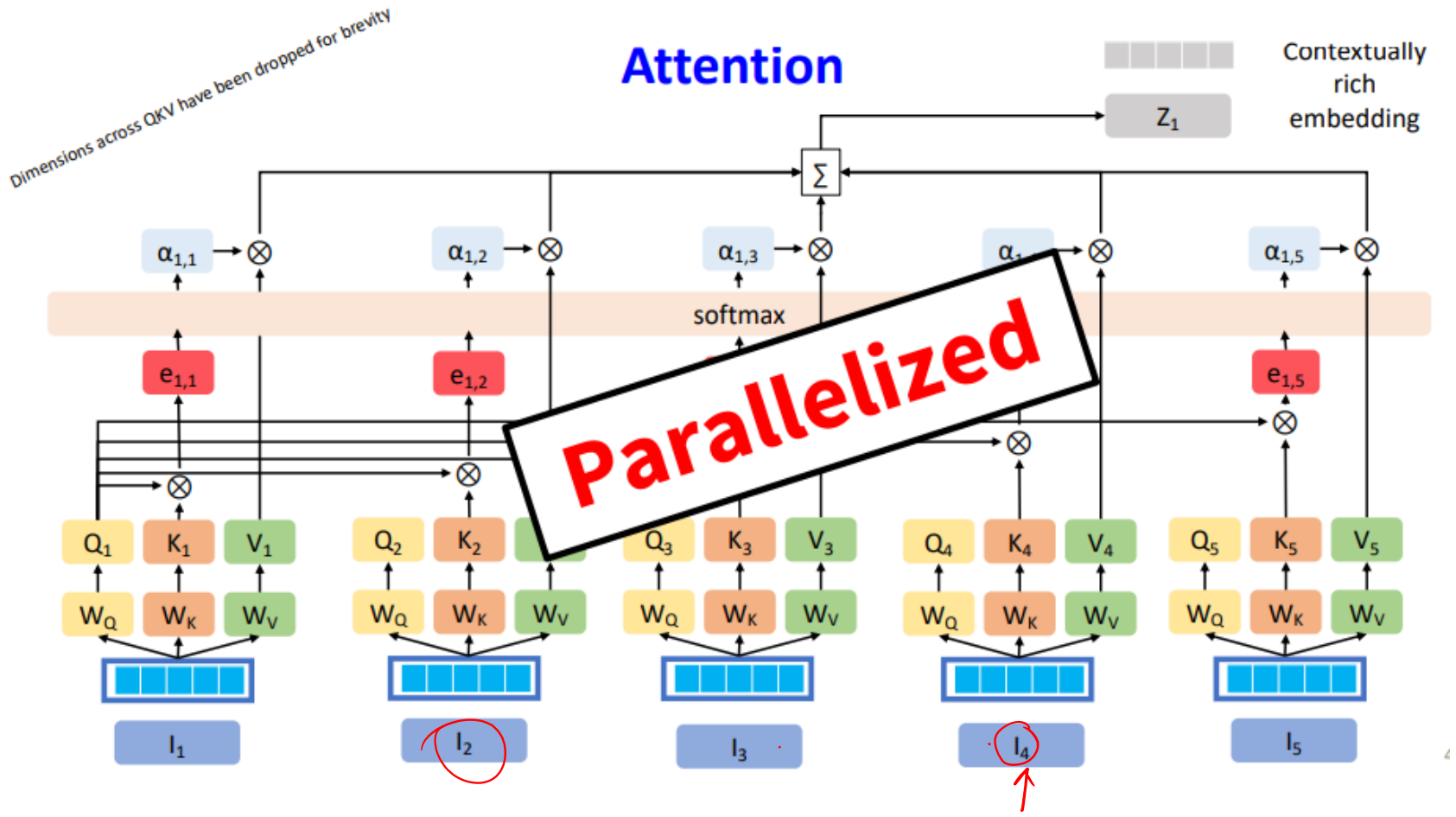
Attention


Dimensions across QKV have been dropped for bre
α1,1
α1,2
α1,3
α1,4
α1,5
softmax
e1,1
e1,2
e1,3
e1,4
e1,5
Q1
K1
V1
Q2
K2
V2
Q3
K3
V3
Q4
K4
V4
Q5
K5
V5
WQ
WK
WV
I1
I2
I3
I4
I5
I
ate
an
apple
<eos>

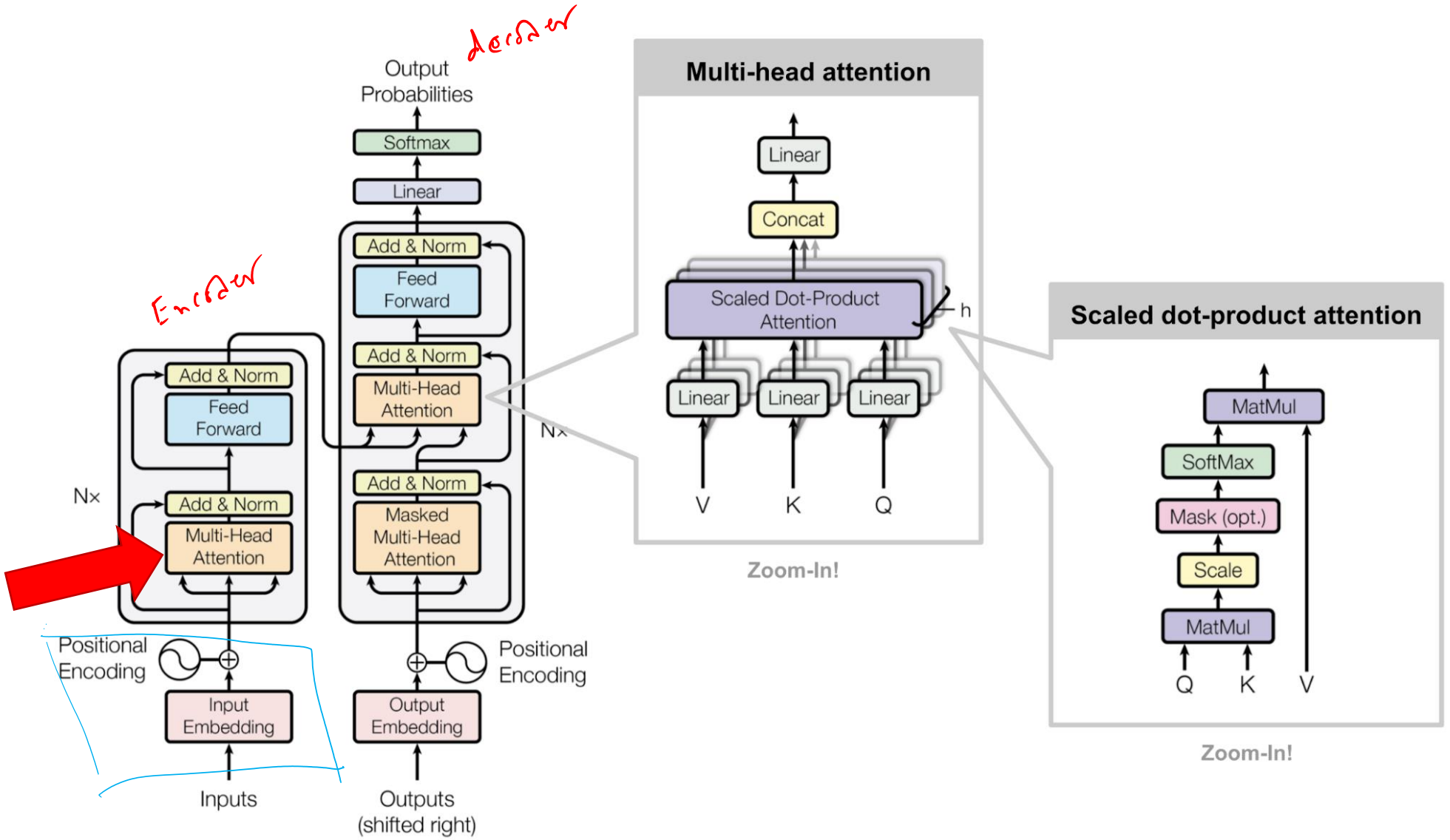
Self attention

$\sum QK^T \cdot V$



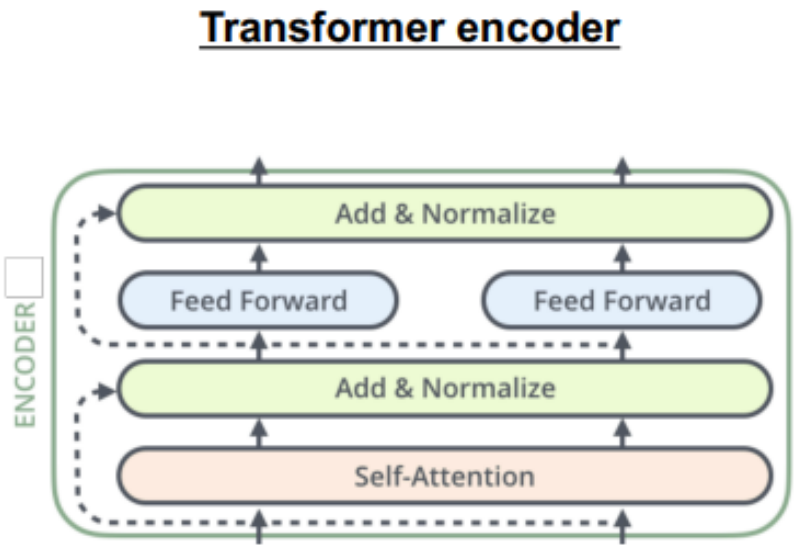

Dimensions across QKV have been dropped for brevity
Attention
Contextually rich embedding
Z_1
Σ
$\alpha_{1,1}$
$\alpha_{1,2}$
$\alpha_{1,3}$
$\alpha_{1,5}$
softmax
Parallelized
$e_{1,1}$
$e_{1,2}$
$e_{1,5}$
Q_1 K_1 V_1
Q_2 K_2
Q_3 K_3 V_3
Q_4 K_4 V_4
Q_5 K_5 V_5
W_Q W_K W_V
I_1
I_2
I_3
I_4
I_5

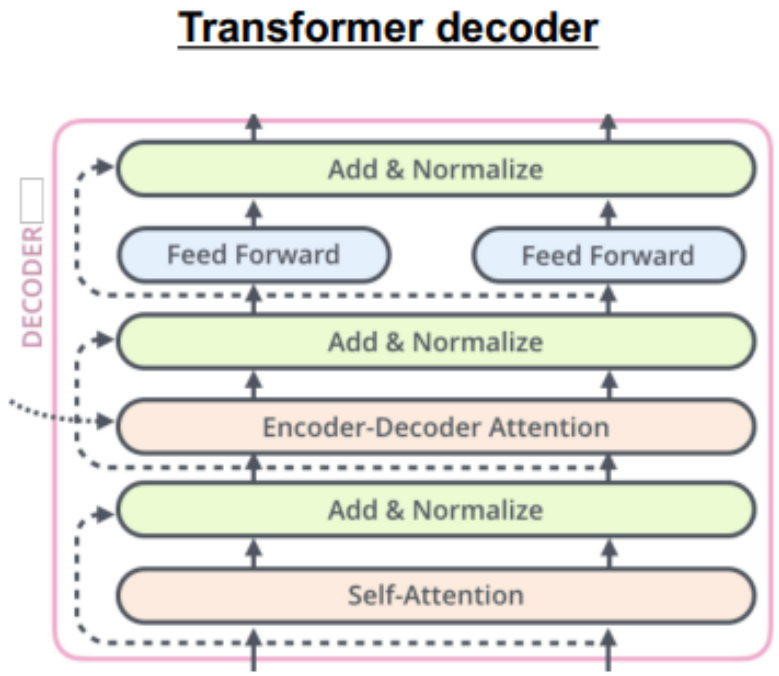
Multi-Head Attention



Transformers

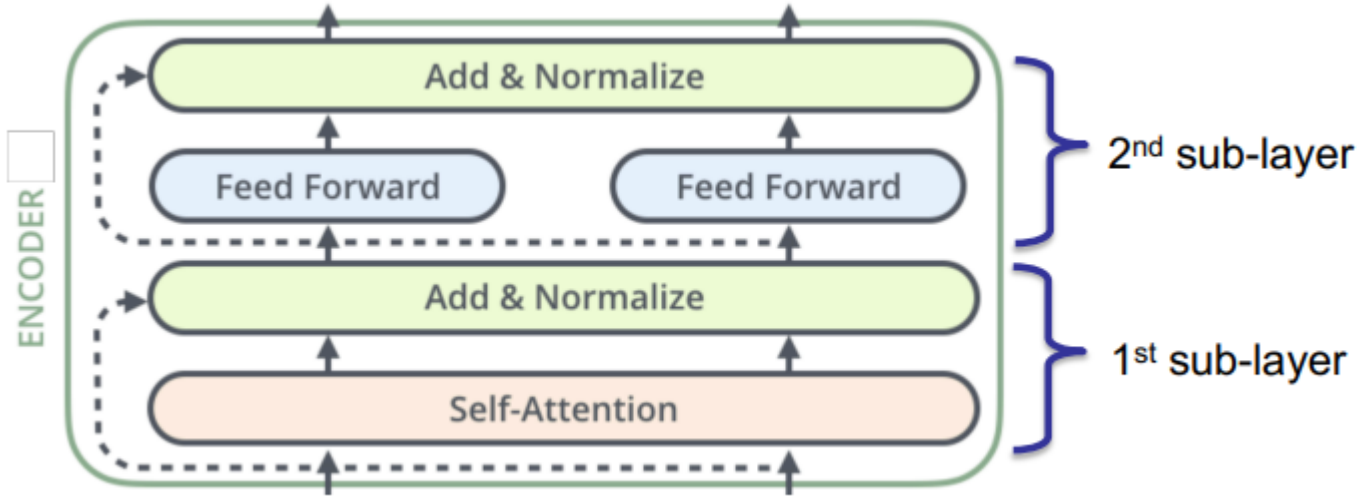
- An attention model with DL best practices!
- Originally introduced for machine translation, and now widely adopted for non-recurrent sequence encoding and decoding





Transformer encoder

- Transformer encoder consists of two sub-layers:
- 1st : Multi-head scaled dot-product self-attention
- 2nd : Position-wise feed forward
- Each sub-layer is followed by layer normalization and residual networks … and drop-outs are applied after each computation



Transformer encoder

- Let’s start from multi-head scaled dot-product self-attention:
- Scaled dot-product attention
- Multi-head attention
- self-attention (recap)

- First, non-normalized attention scores:

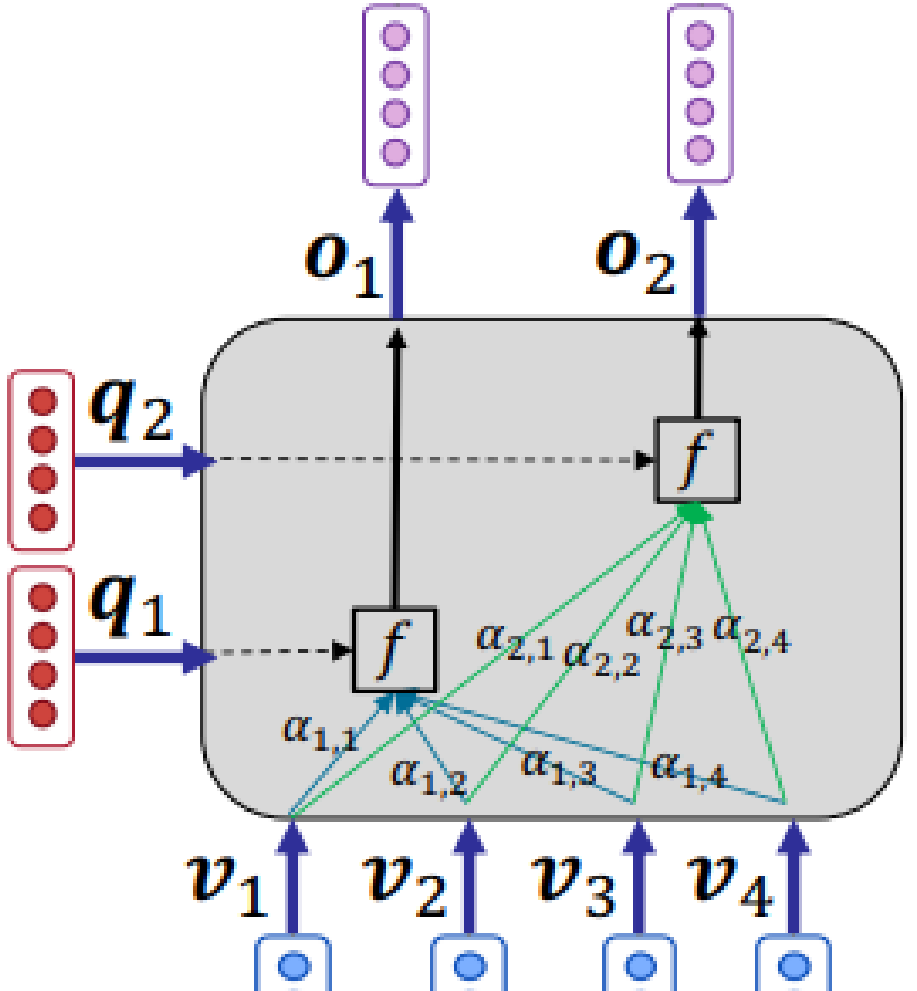
$$\tilde{\alpha}_{i,j} = \boldsymbol{q}_i \boldsymbol{v}_j^{\mathrm{T}}$$

  - $d = d_q = d_v$ dimension of vectors
  - has no parameter!

- Then, softmax over values:

$$\alpha_{i,j} = \mathrm{softmax}(\tilde{\boldsymbol{\alpha}}_i)_j$$

- Output (weighted sum): $\boldsymbol{o}_i = \sum_{j=1}^{|V|} \alpha_{i,j} \boldsymbol{v}_j$



Scaled dot-product attention

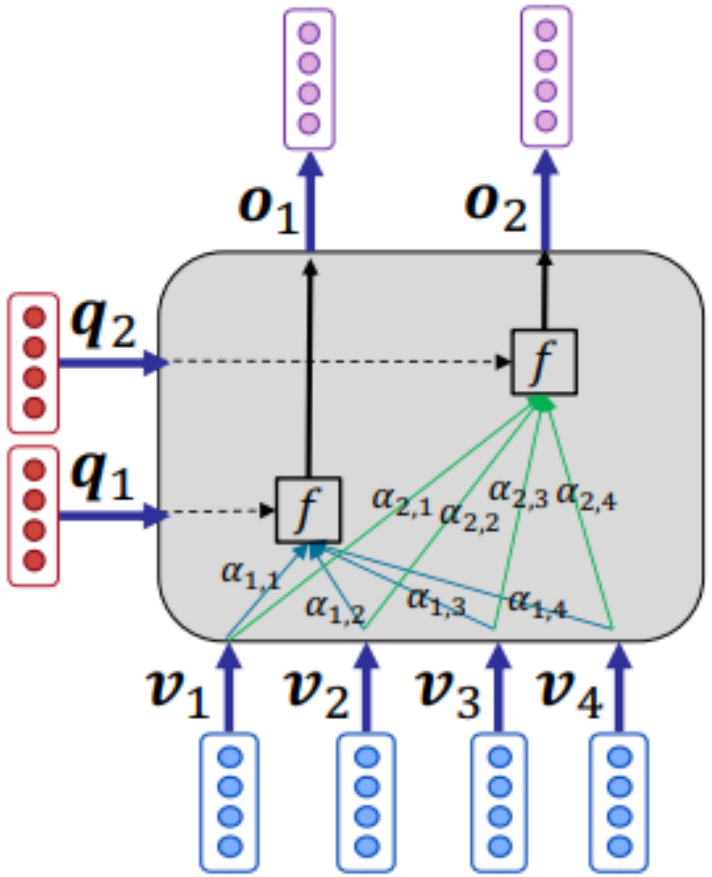
- Problem with basic doc-product attention:
  - As d gets large, the variance of $\tilde{\alpha}_{i,j}$ increases …
  - … this makes softmax very peaked for some values $\tilde{\boldsymbol{\alpha}}_i$ …
  - … and hence its gradient gets smaller
- Solution: normalize/scale $\tilde{\alpha}_{i,j}$ by size of d

Scaled dot-product attention

- Non-normalized attention scores:

$$\tilde{\alpha}_{i,j} = \frac{\boldsymbol{q}_i \boldsymbol{v}_j^{\mathrm{T}}}{\sqrt{d}}$$

- Softmax over values: $\alpha_{i,j} = \mathrm{softmax}(\tilde{\boldsymbol{\alpha}}_i)_j$
- Output (weighted sum): $\boldsymbol{o}_i = \sum_{j=1}^{|V|} \alpha_{i,j} \boldsymbol{v}_j$

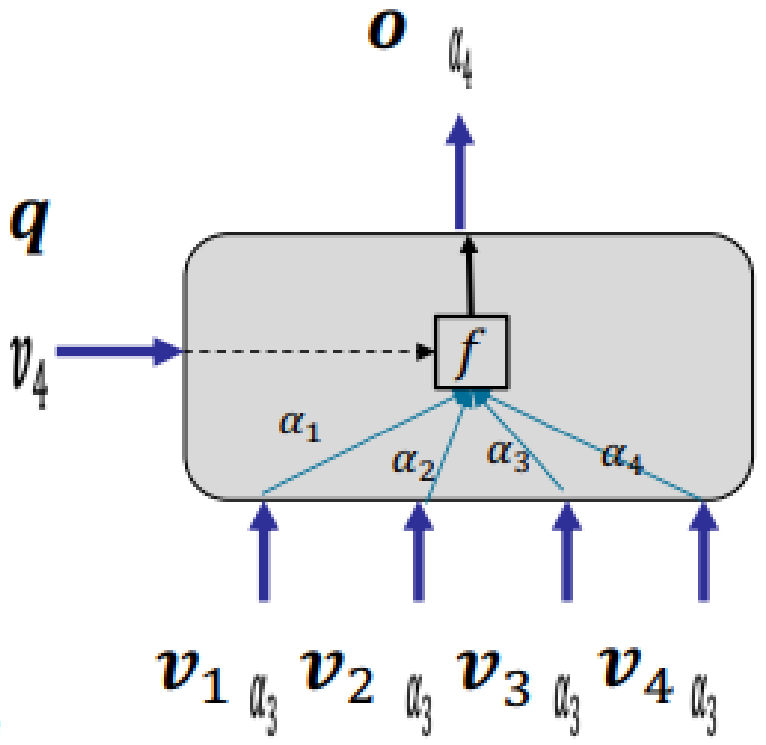


Problem with (single-head) attention

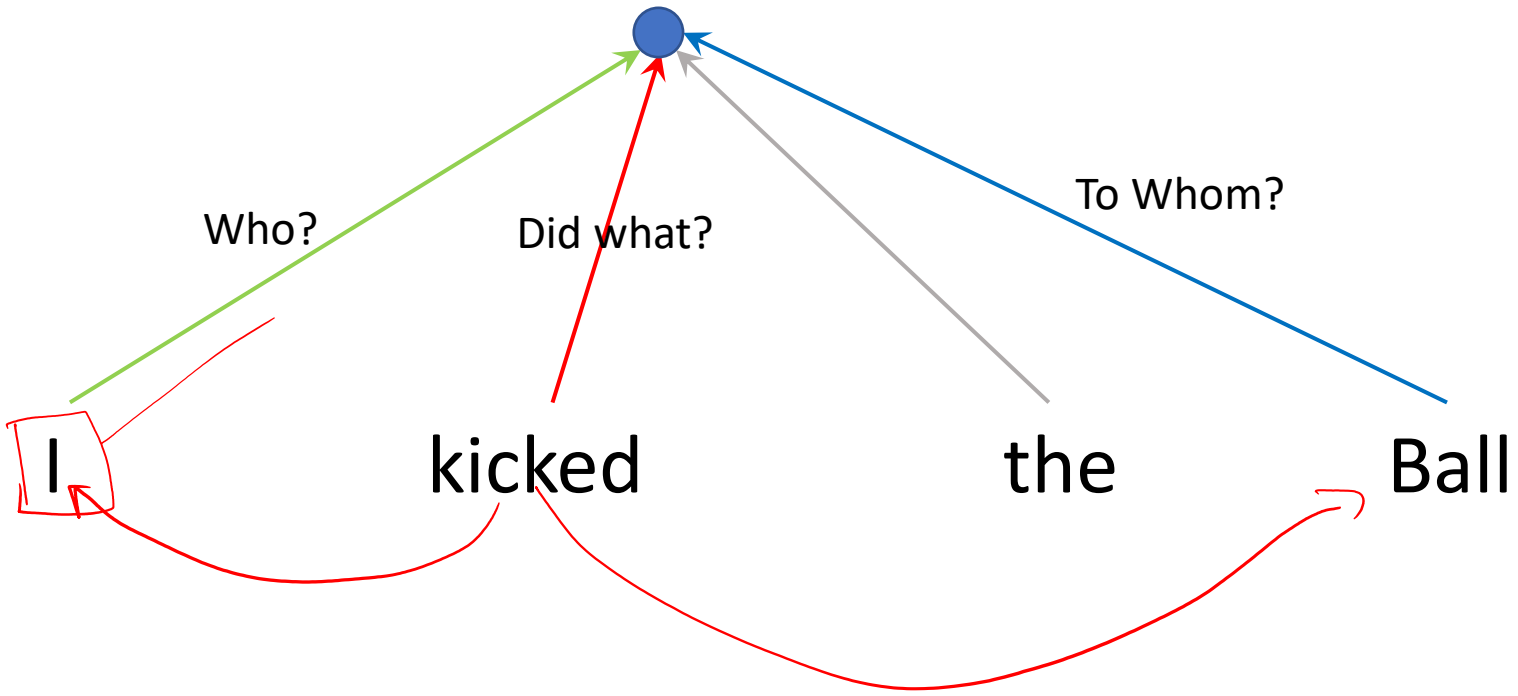
- In all attention networks so far, the final attention of query $\boldsymbol{q}$ on value vectors $\boldsymbol{V}$ are normalized with softmax
  - Recall that softmax makes the maximum value much higher than the other

$$\boldsymbol{z} = [1 \quad 2 \quad 5 \quad 6] \rightarrow \text{softmax}(\boldsymbol{z}) = [0.004 \quad 0.013 \quad 0.264 \quad 0.717]$$

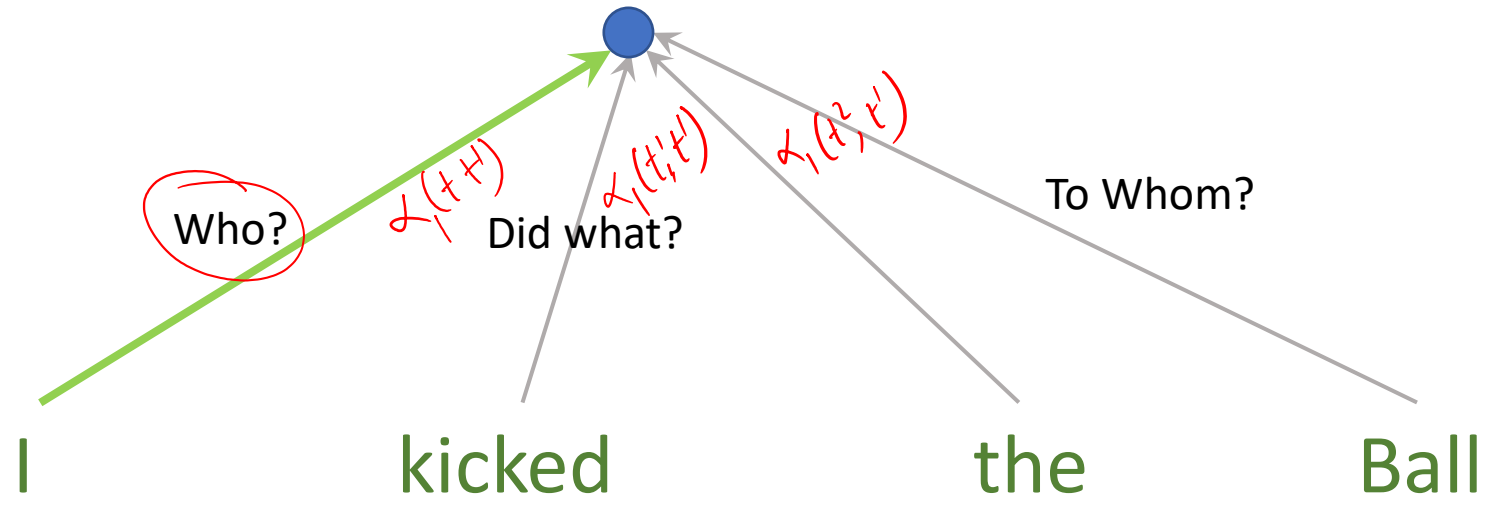
- Common in language, a word may be related to several other words in sequence, each through a specific concept
  - Like the relations of a verb to its subject and to its object
- However in a (single-head) attention network, all concepts are aggregated in one attention set
- Due to softmax, value vectors must compete for the attention of query vector → softmax bottleneck



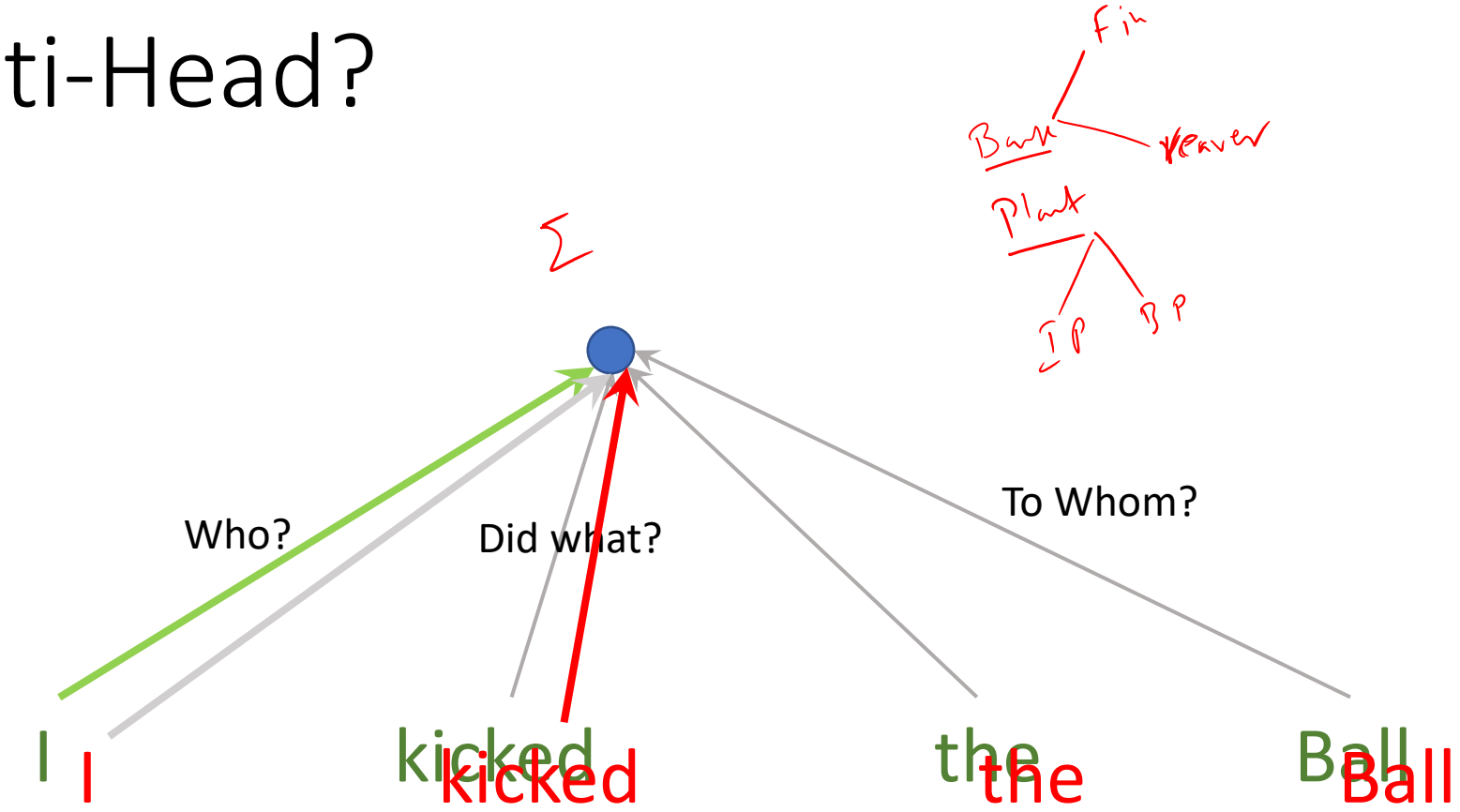
Multi-Head?



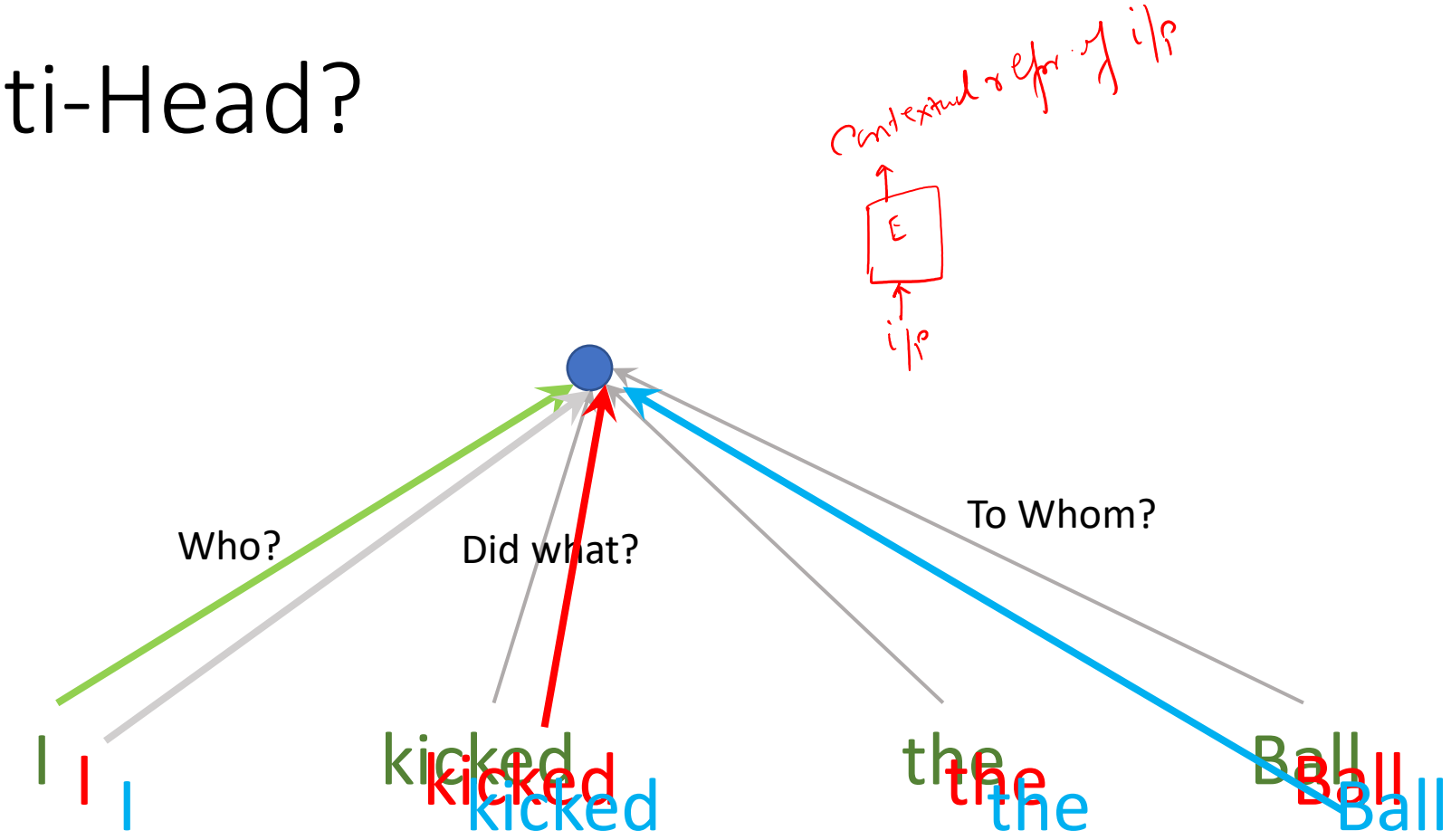
Multi-Head?



Multi-Head?

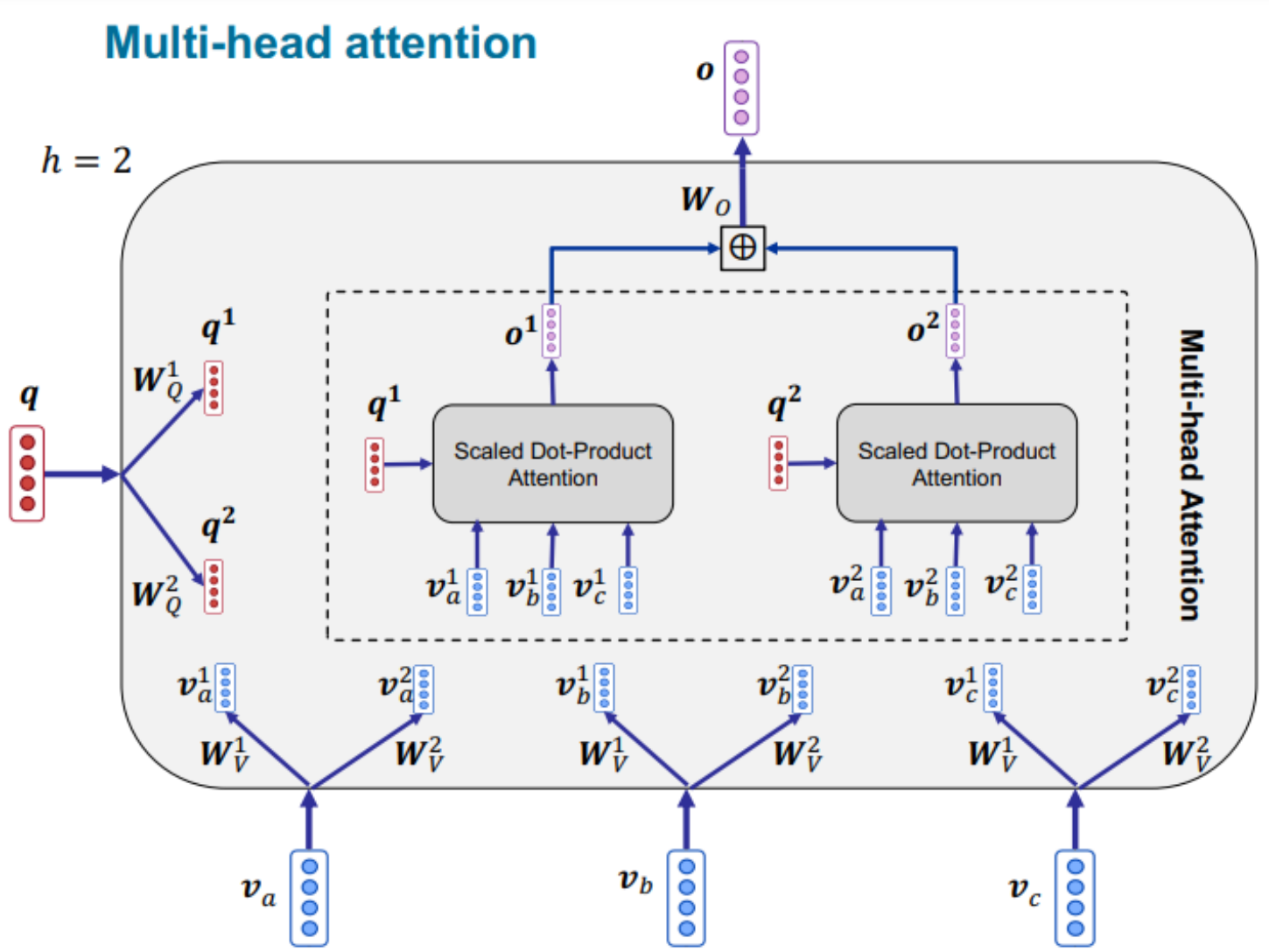


Multi-Head?

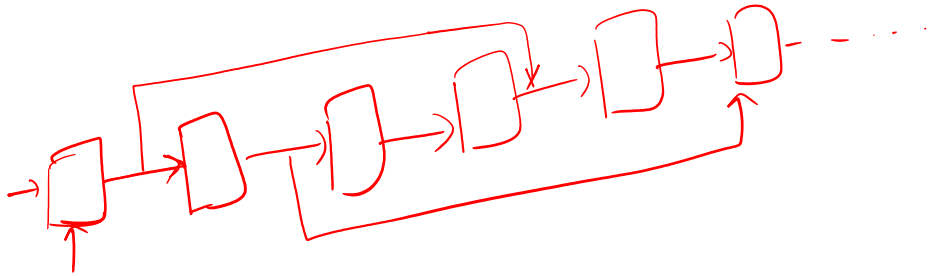

Context aware repr. of i/p
E
i/p
Who?
Did what?
To Whom?
I
I
I
kicked
kicked
kicked
the
the
the
Ball
Ball
Ball

Multi-head attention

- Multi-head attention approaches this by calculating multiple sets of attentions between queries and values
- In multi-head attention, each head (and each subspace) can specialize on capturing a specific kind of relations



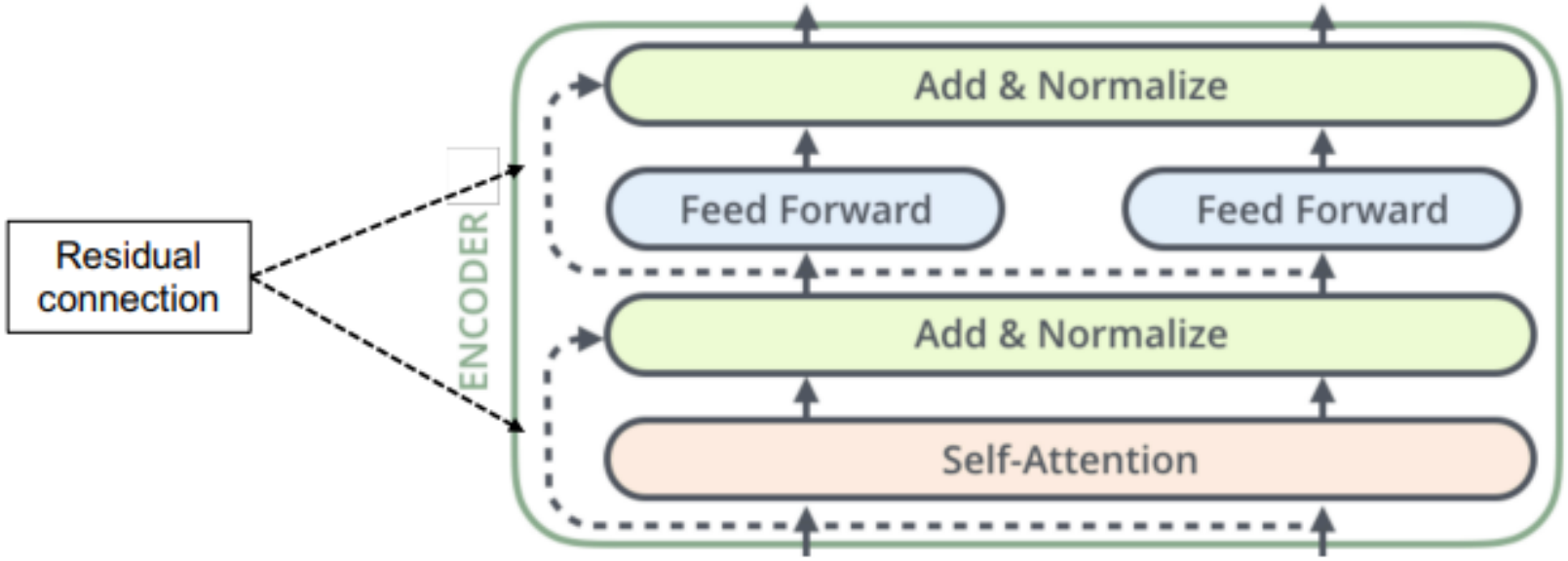
Residuals



- Residual (short-cut) connection:

$$\text{output} = f(x) + x$$

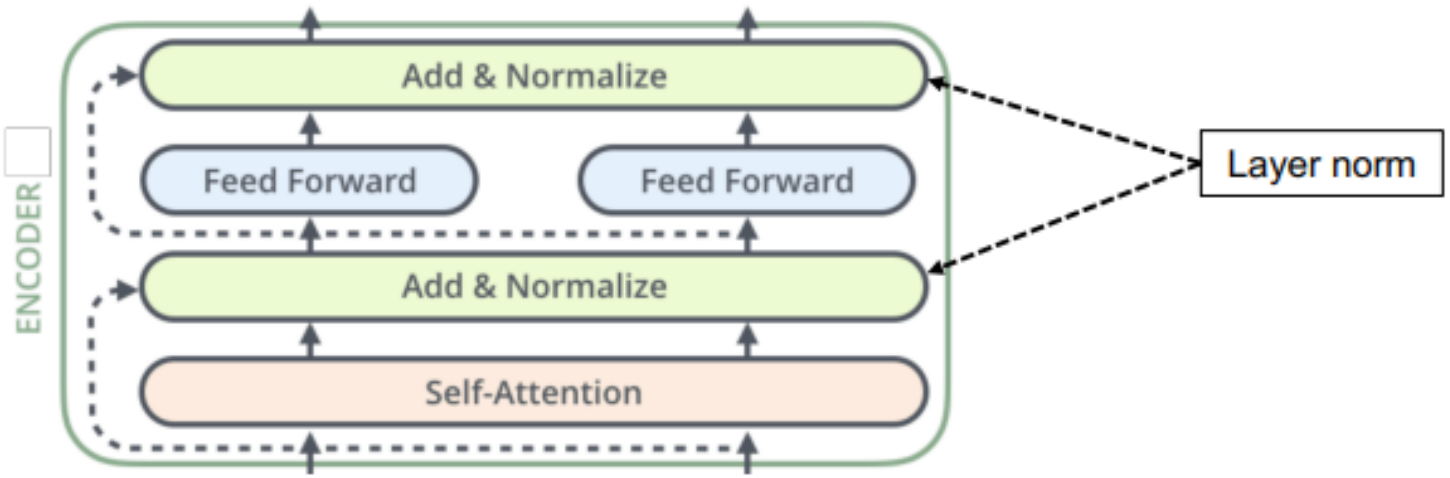
- Learn in detail:
  - He, Kaiming; Zhang, Xiangyu; Ren, Shaoqing; Sun, Jian (2016). "Deep Residual Learning for Image Recognition" . In proc. of CVPR
  - Srivastava, Rupesh Kumar; Greff, Klaus; Schmidhuber, Jürgen (2015). "Highway Networks". https://arxiv.org/pdf/1505.00387.pdf



Layer normalization

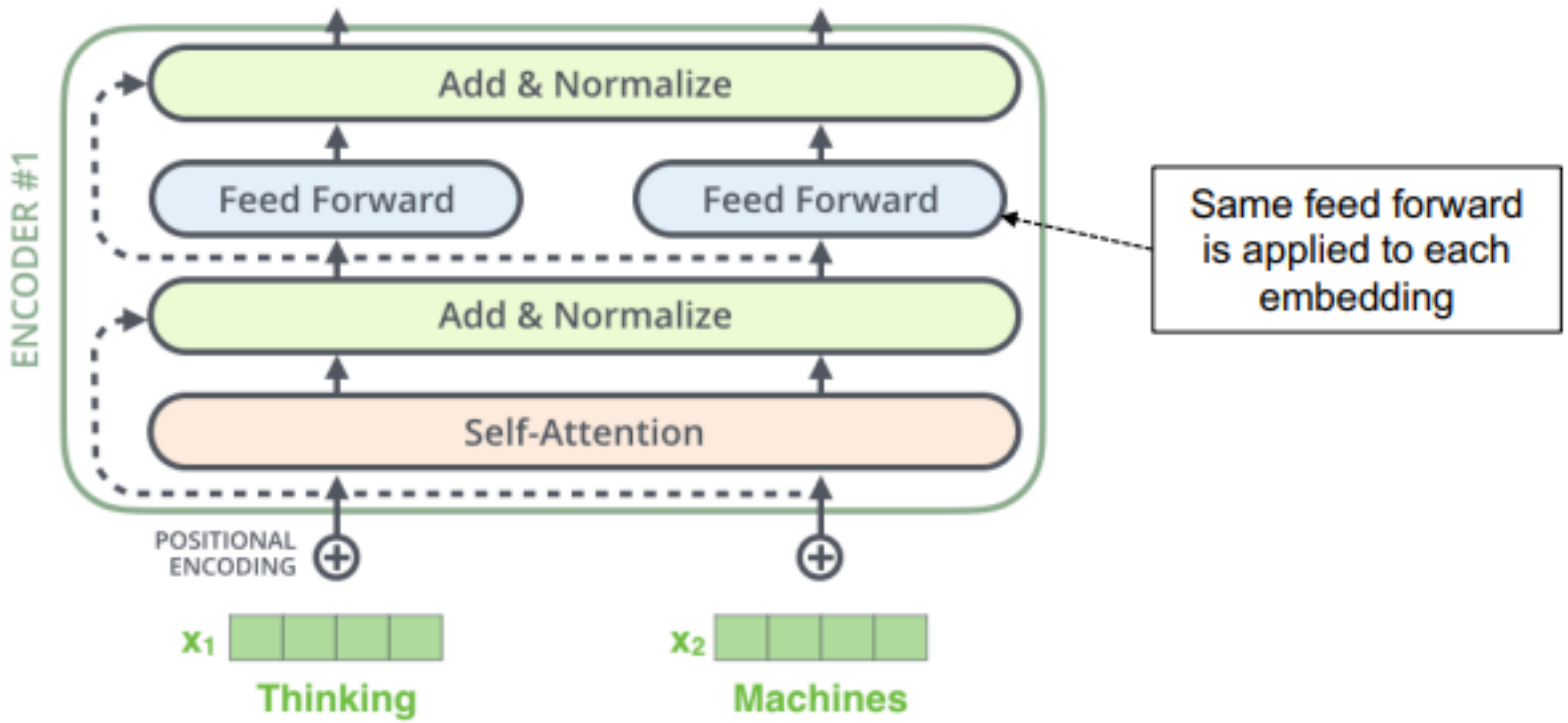
- Layer normalization changes each vector to have mean 0 and variance 1 …
  - … and learns two more parameter vectors per layer that set new means and variances for each dimension of the vectors

- Learn in detail:
  - Batch Normalization: Deep Learning book section 8.7.1 http://www.deeplearningbook.org/contents/optimization.html
  - Talk by Goodfellow https://www.youtube.com/watch?v=Xogn6veSyxA&feature=youtu.be
  - Paper: https://arxiv.org/pdf/1607.06450.pdf



Feed Forward on embedding

- In Transformers, a two-layer feed forward neural network (with ReLU) is applied to each embedding
  - With the feed forward network, the Transformers gain the capacity to learn non-linear transformations over each (contextualized) embedding



Encoders

Encoder

ENCODER

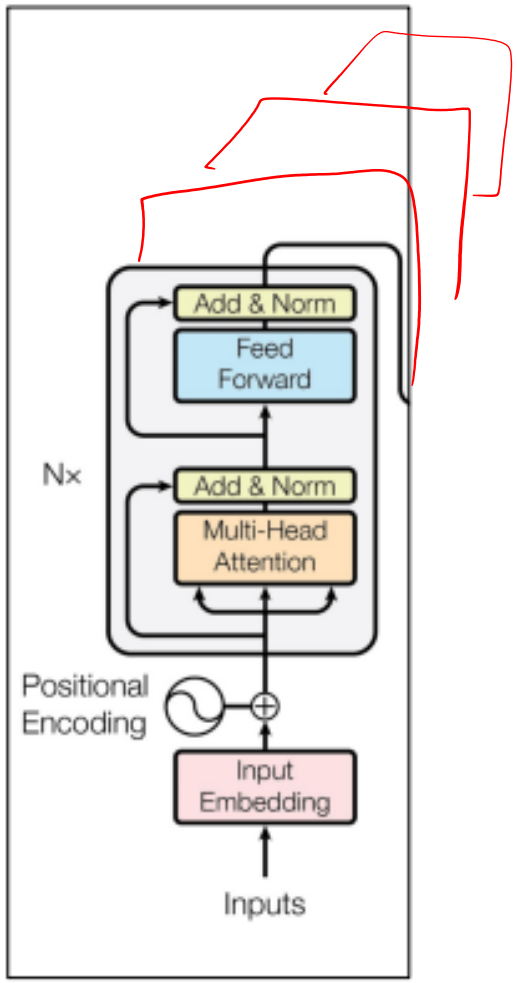
.
.
.

ENCODER

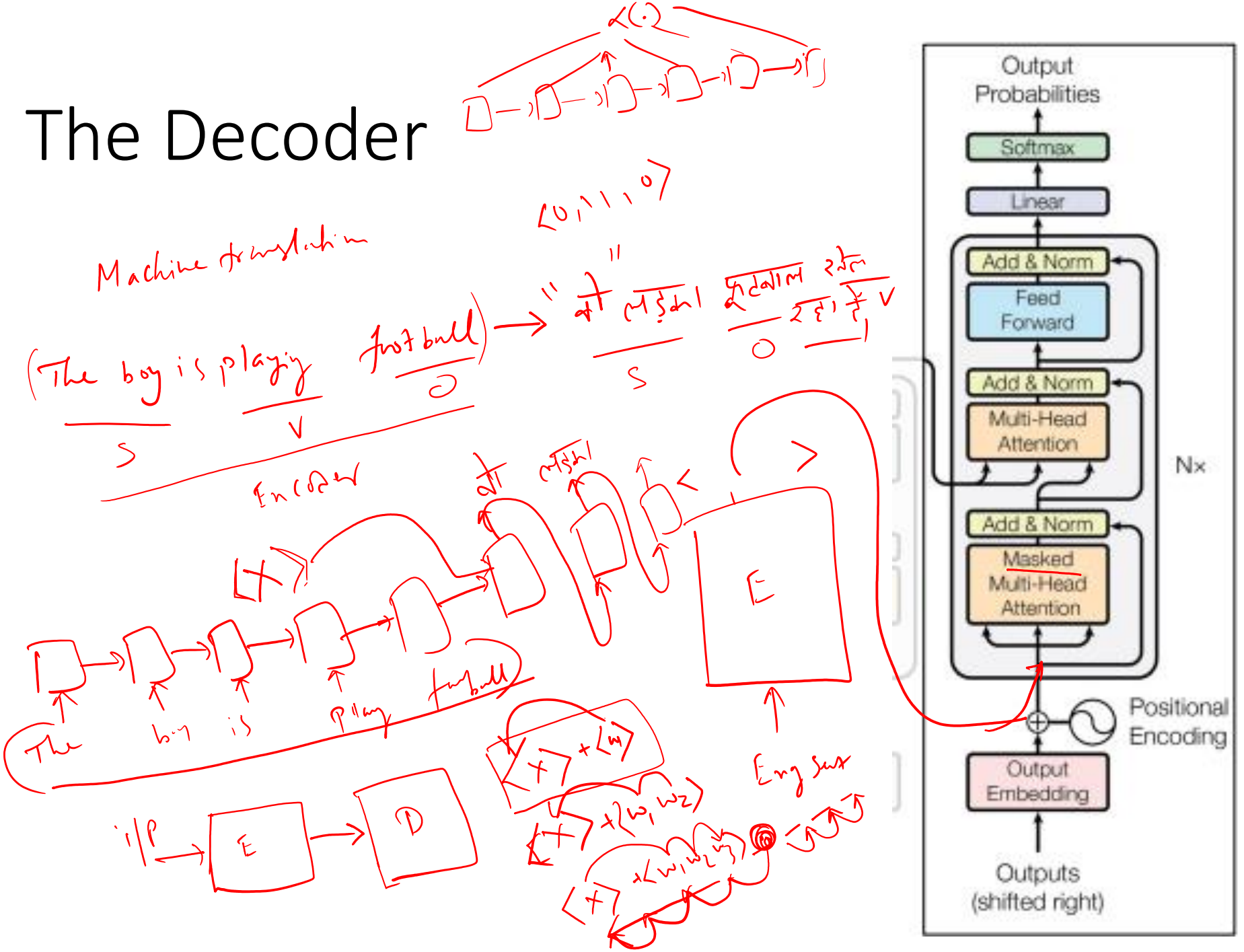
ENCODER

Input to Encoder$_{i+1}$

Output from Encoder$_i$



The Decoder


Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
Nx
Positional Encoding
Output Embedding
Outputs (shifted right)

Transformer encoder – summary

- Multi-head self-attention model followed by a feed-forward layer
- Benefits (as in attentions)
  - No locality bias - A long-distance context has “equal opportunity”
  - Friendly with high parallel computations of GPU

- Look here for self-teaching and the PyTorch implementation: http://nlp.seas.harvard.edu/2018/04/03/attention.html
- also available on Google Colab