Transformer models

Marta R. Costa-jussà

with slides from Peter Boem, Ashish Vaswani and Anna Huang

What are Transformers?

- transformer: any sequence-based model that primarly uses self-attention to propagate along the time dimension
- more broadly: any model that primarily uses self-attention to propagate information between basic units of our instansces
  - pixels
  - graph nodes
  - speech

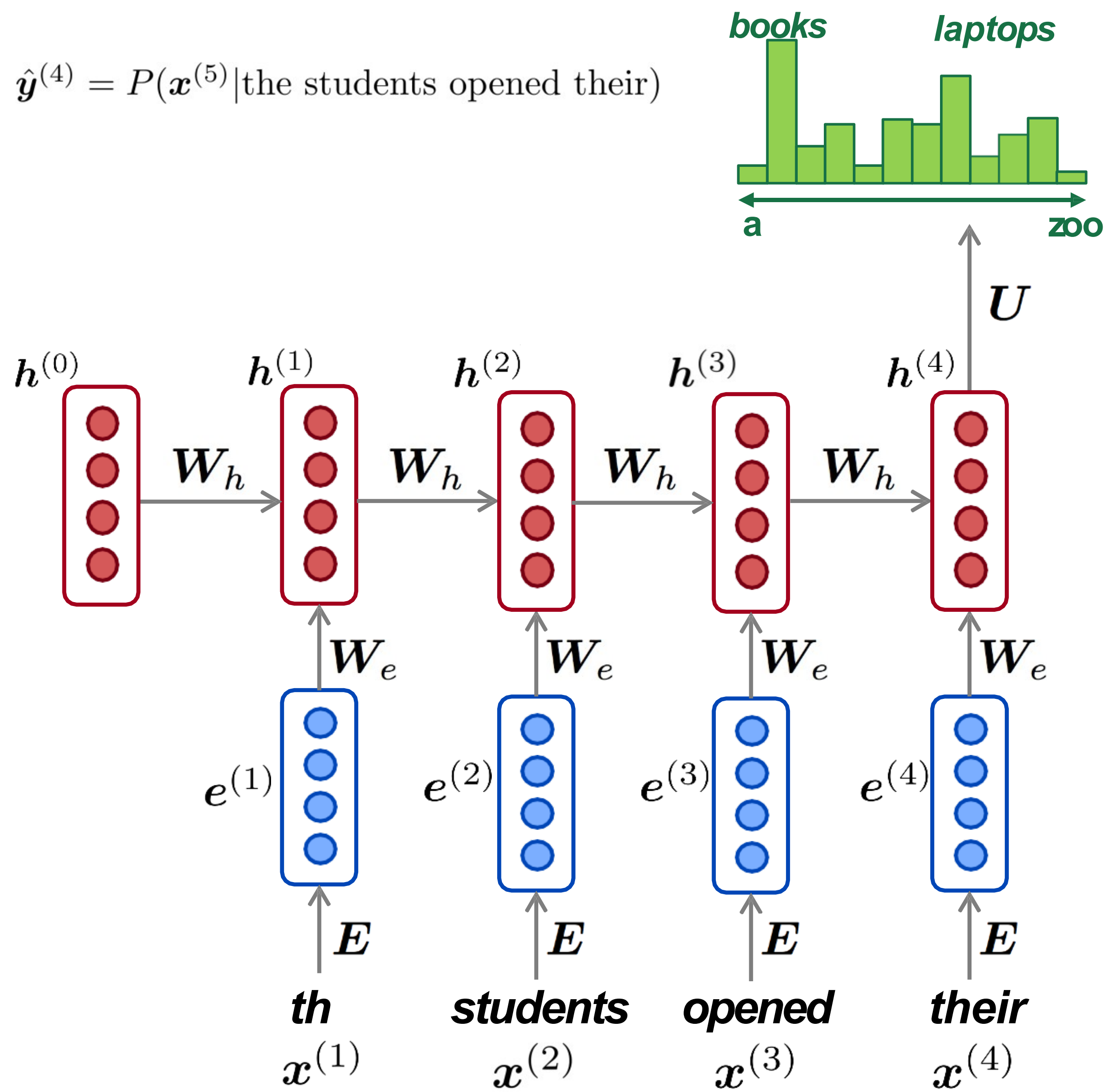
- motivation:
  - take advantage of all data available (parallelizable)
  - benefit from long -range dependencies

Outline

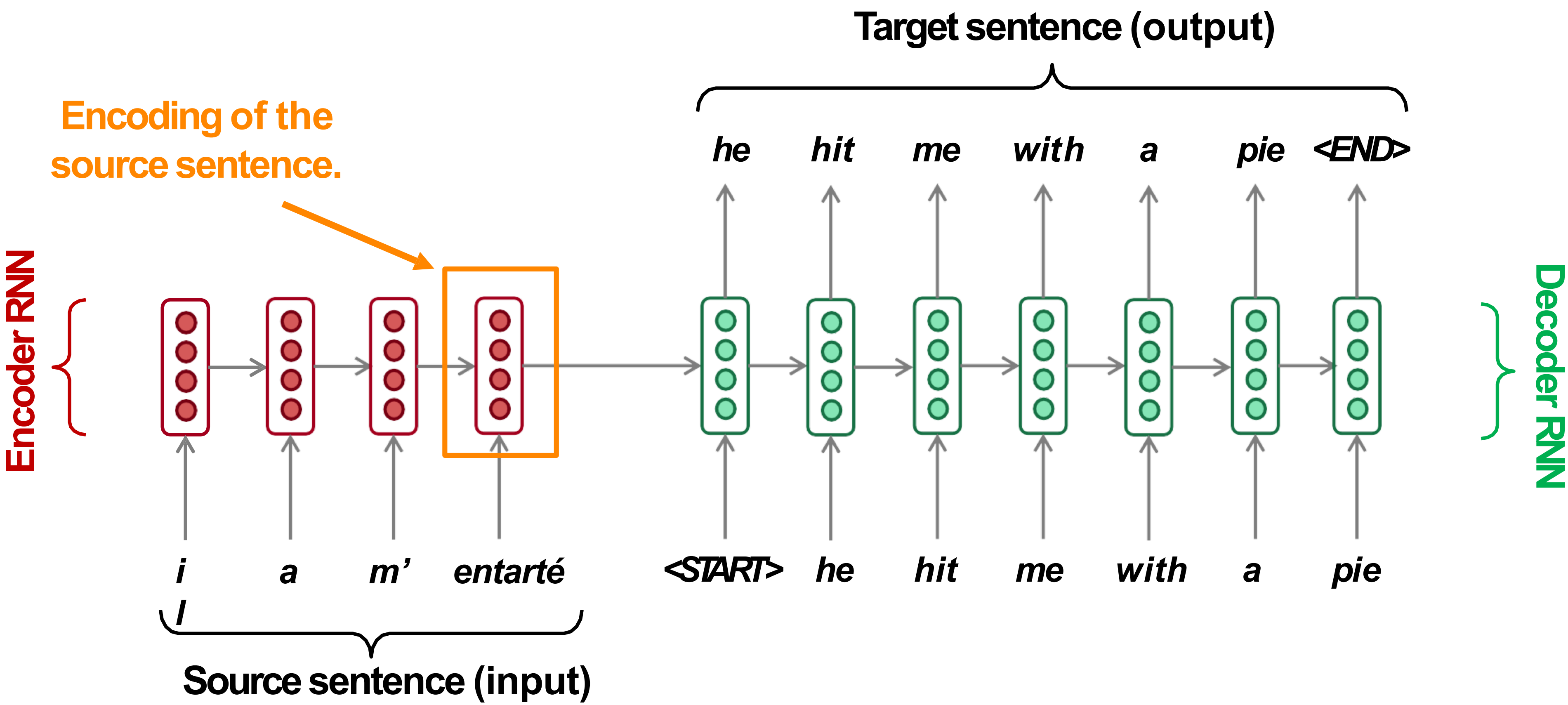
- Background: Language Modeling, Seq2Seq with Attention
- Key Concepts of the Transformer
  - Self Attention
  - Multi-Head Attention
- Position Information
- Transformer Layers/Blocks
- Encoder vs Decoder (Masking, Inter Attention, Softmax)

BACKGROUND

Background: Language modeling with RNN



Background: Seq2Seq



References

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

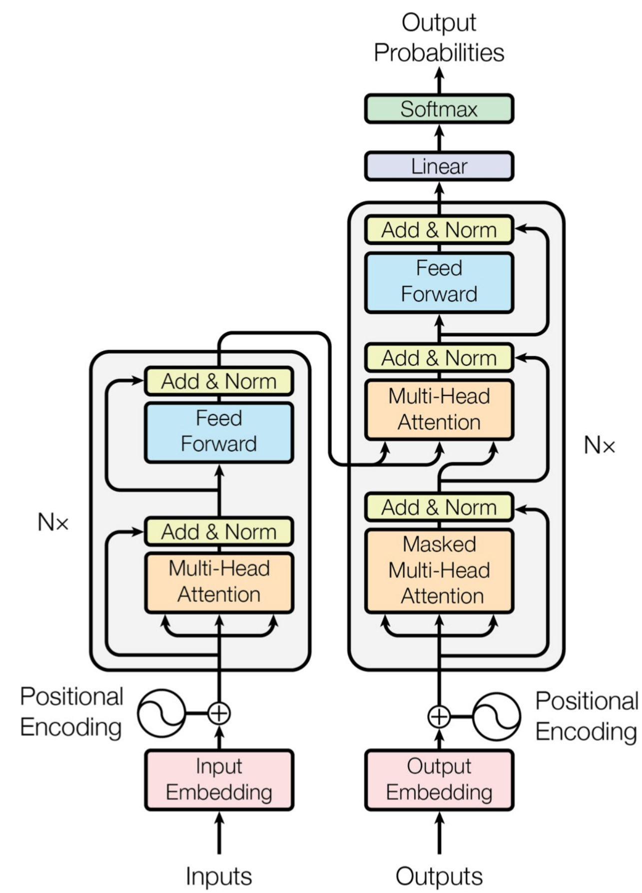
Abstract

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

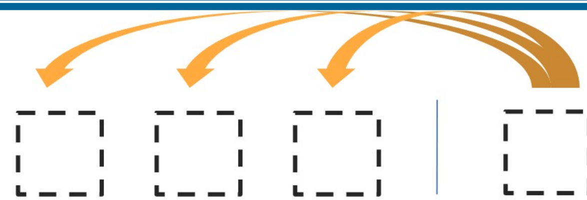
http://jalammar.github.io/illustrated-transformer/

TRANSFORMER

Complete picture

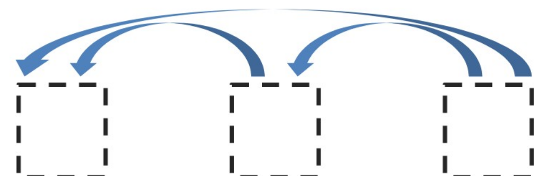


Different attentions

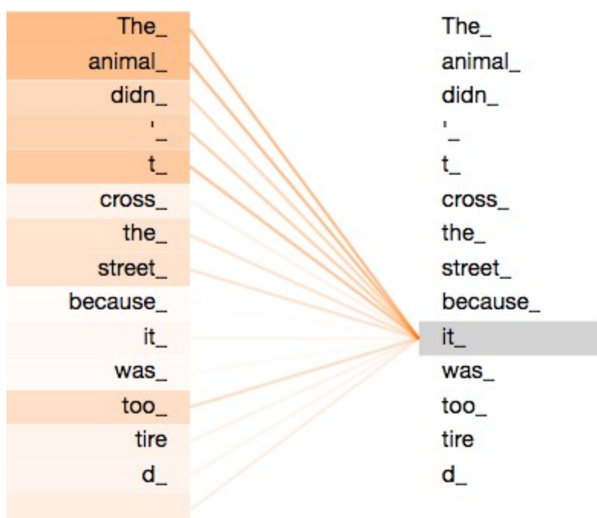


Encoder-Decoder Attention

Encoder Self-Attention

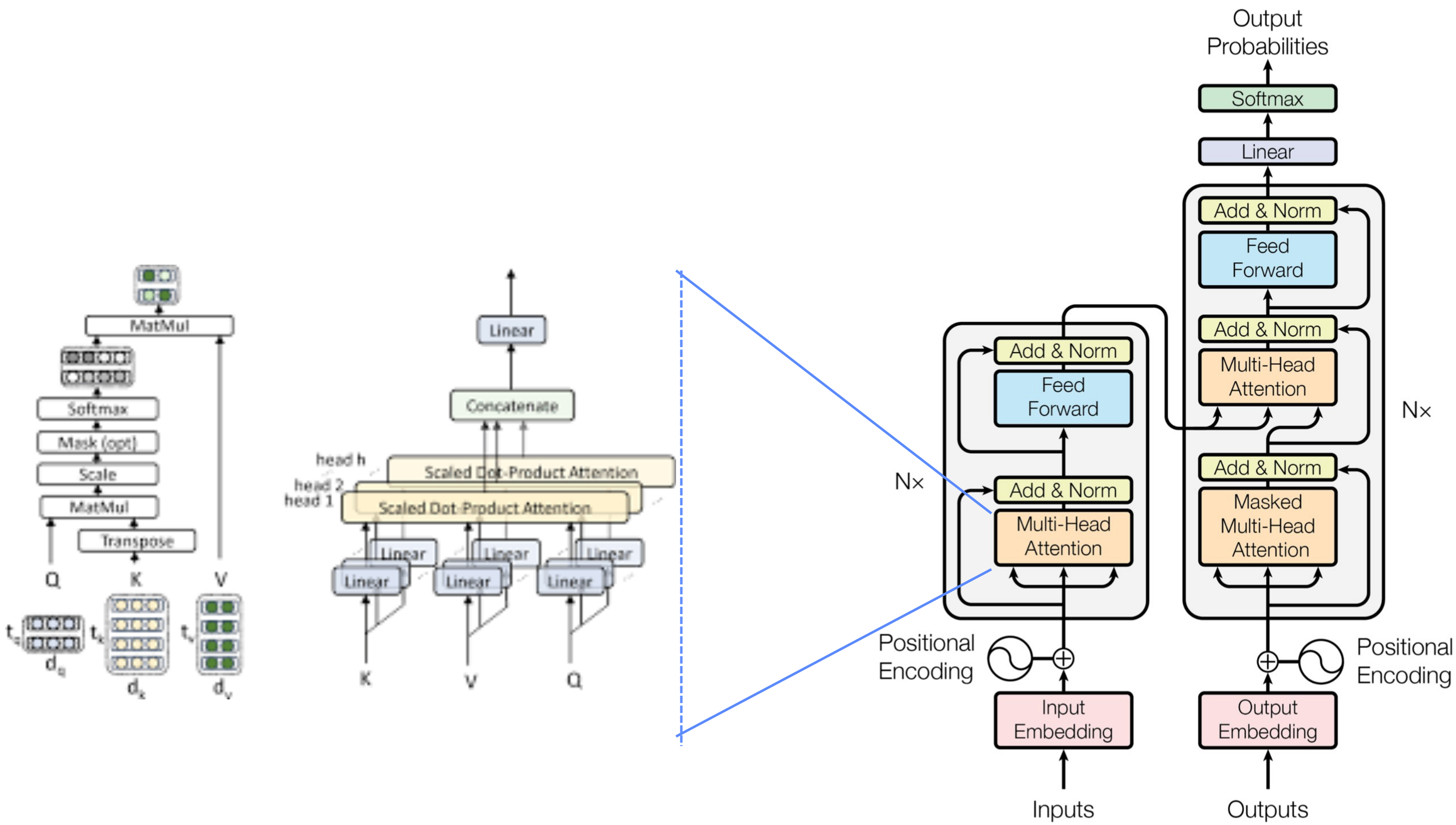


MaskedDecoder Self-Attention



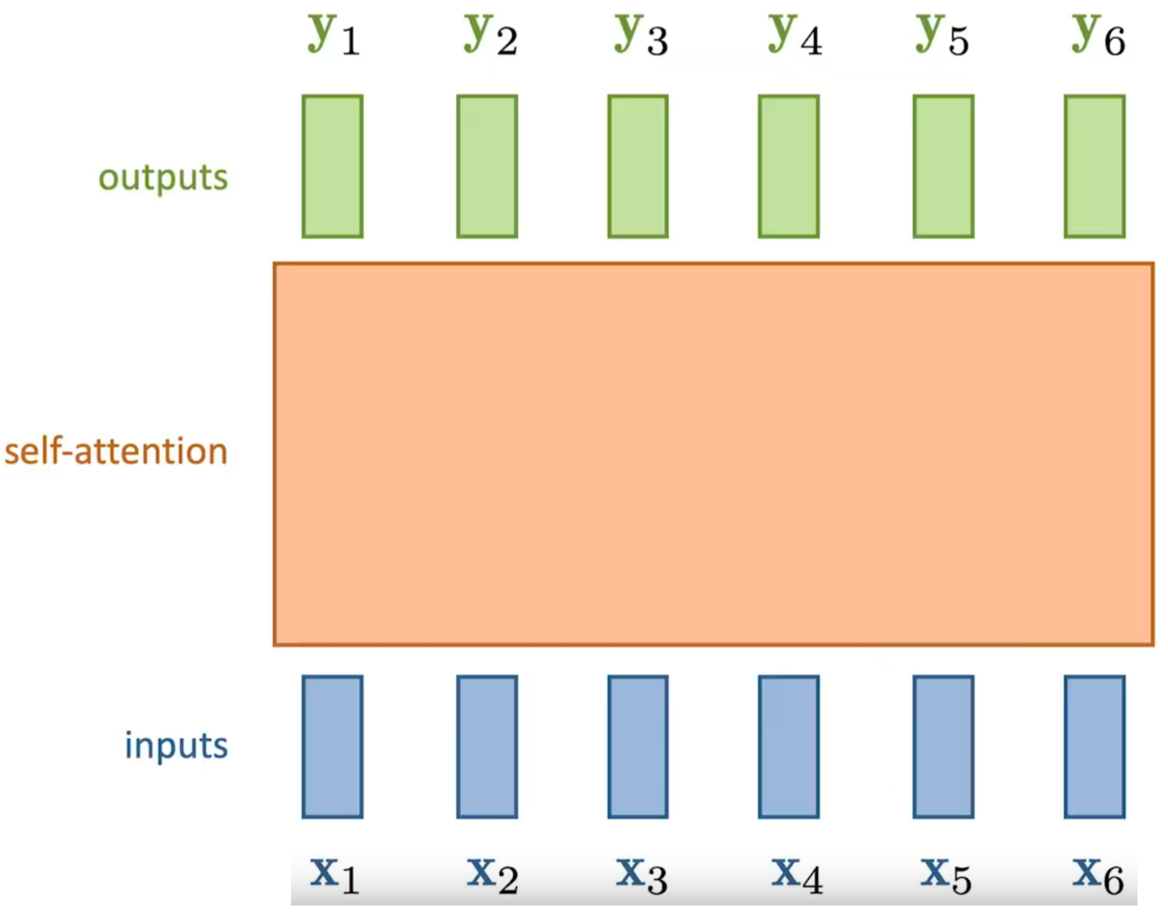
Key concepts

- Self-attention and multi-head attention



TRANSFORMER: SELF ATTENTION

Self-attention: step-by-step I: intuition



$$\mathbf{y}_i = \sum_j w_{ij} \mathbf{x}_j$$

not a parameter

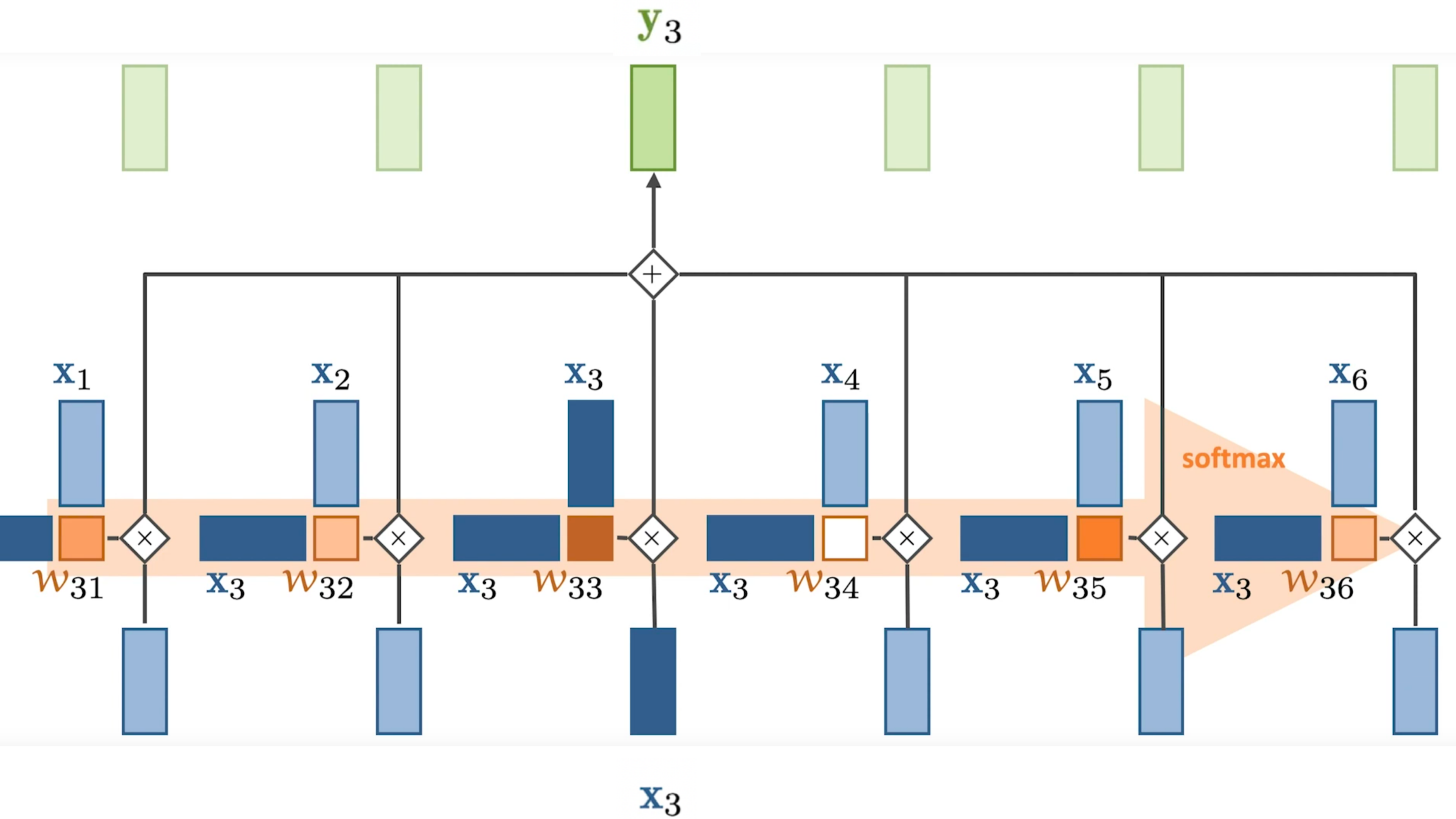
Self-attention: step-by-step II: equations

$$\mathbf{y}_i = \sum_j w_{ij} \mathbf{x}_j$$

$$w'_{ij} = \mathbf{x}_i{}^{\mathsf{T}} \mathbf{x}_j$$

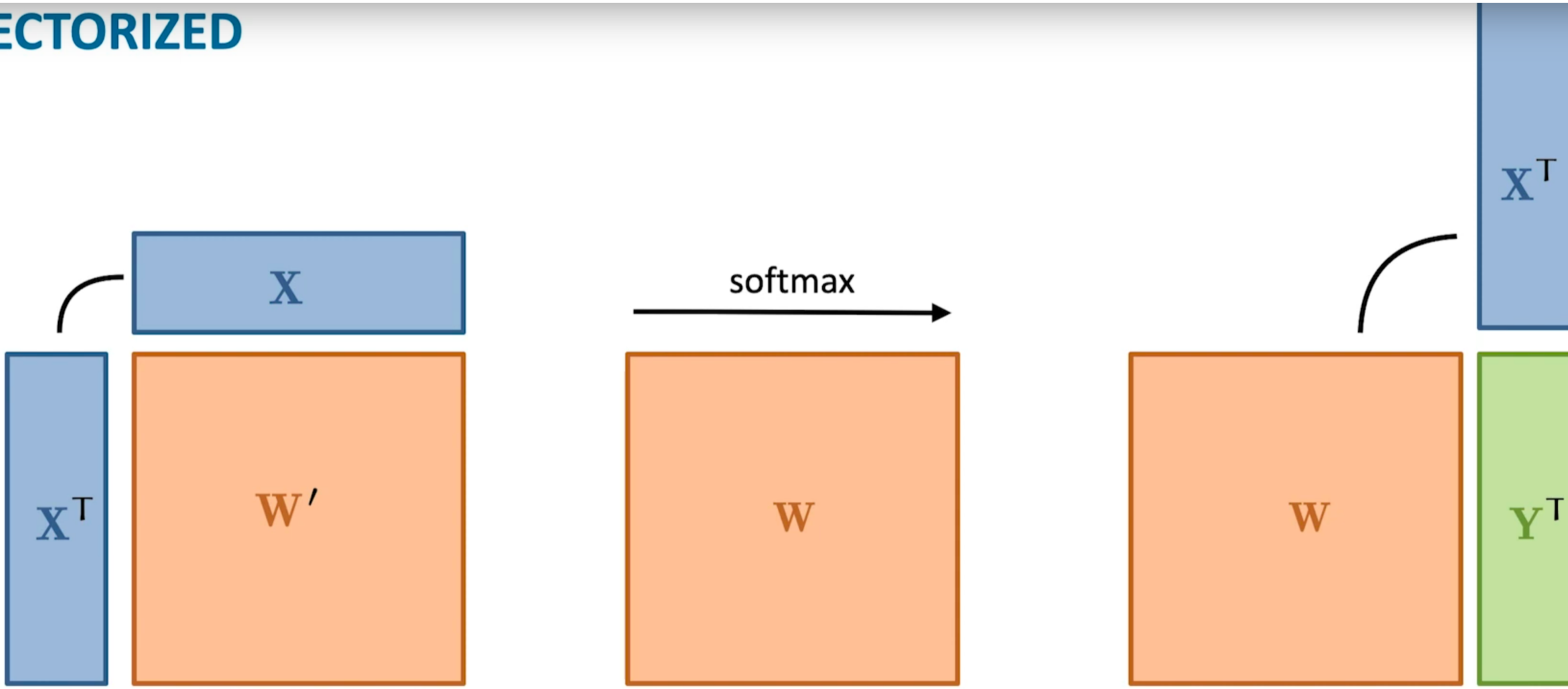
$$w_{ij} = \frac{\exp w'_{ij}}{\sum_j \exp w'_{ij}}$$

Self-attention: step-by-step III: graphically



Self-attention: step-by-step IV: vectorization

VECTORIZED



$$W' = X^T X \qquad W = \mathrm{softmax}(W') \qquad Y^T = W X^T$$

Self-attention: notes

In *simple* self-attention w_{ii} (x_i to y_i) usually has the most weight

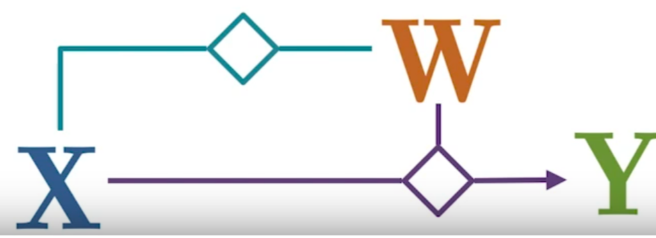
not a big problem, but we'll allow this to change later.

Simple self-attention has *no parameters*.

Whatever parameterized mechanism generates x_i (like an embedding layer) drives the self attention.

There is a linear operation between X and Y.

non-vanishing gradients through $Y = WX^T$, vanishing gradients through $W = \text{softmax}(X^TX)$.



best of two worlds:
linear and non-linear operations

Self-attention: notes

No problem looking far back into the sequence.

In fact, every input has the same distance to every output.

More of a *set model* than a *sequence model*. No access to the sequential information.

We'll fix by encoding the sequential structure into the embeddings. Details later.

Permutation equivariant.

for any permutation p of the input: $p(sa(\mathbf{X})) = sa(p(\mathbf{X}))$

Modifications to Self-Attention

- Scaled dot product
- Key, value and query transformations

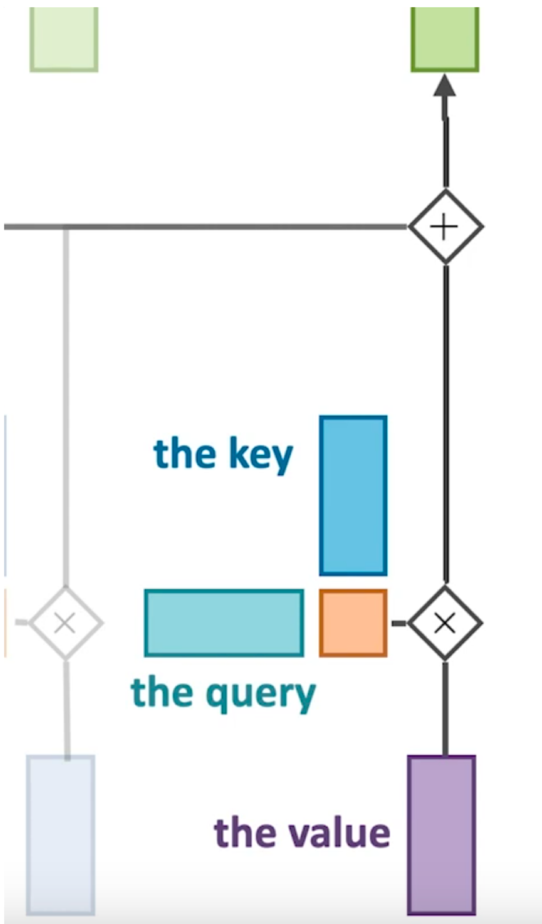
Scaled Dot Product

it keeps the weights within a certain range, not depending on the dimensionality of the vector

$$w'_{ij} = \frac{\mathbf{x}_i{}^T \mathbf{x}_j}{\sqrt{k}}$$

<- input dimension

Key, query, value



every vector occurs in 3 different positions
value: weighted sum that provides the output
query: input vector that corresponds to the current output matched against every other input vector
key: the vector that the query vector is matched against

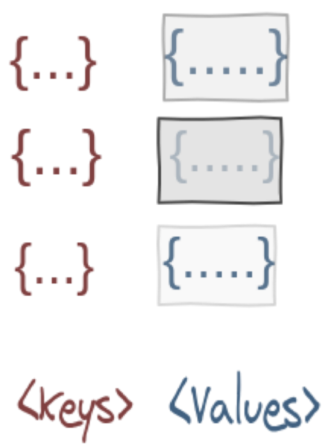
```
d = {'a' : 1, 'b' : 2, 'c' : 3}

d['b'] = 2
```

key value query

key	value
a	1
b	2
c	3

Keys, Queries and Values inspired in databases notation

Databases store information as pair of keys and values (K,V).



Example:

myfile.pdf

<Key>

<Value>

Figure: Nikhil Shah, “Attention? An Other Perspective! [Part 1]” (2020)

Keys, Queries and Values

The (K,Q,V) terminology used to retrieve information from databases is adopted to formulate attention

Attention is a mechanism to compute a context vector (c) for a query (Q) as a weighted sum of values (V).

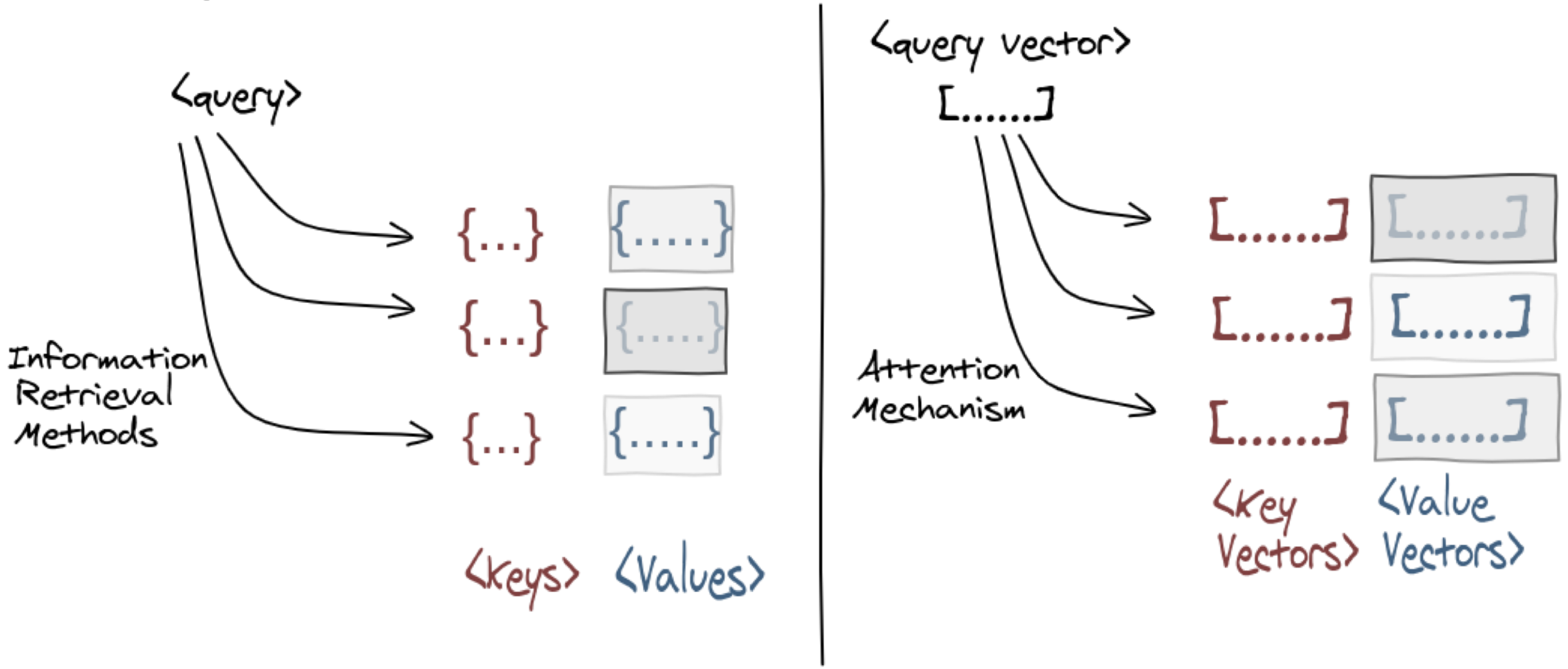


Figure: Nikhil Shah, "Attention? An Other Perspective! [Part 1]" (2020)

Linear Transformations

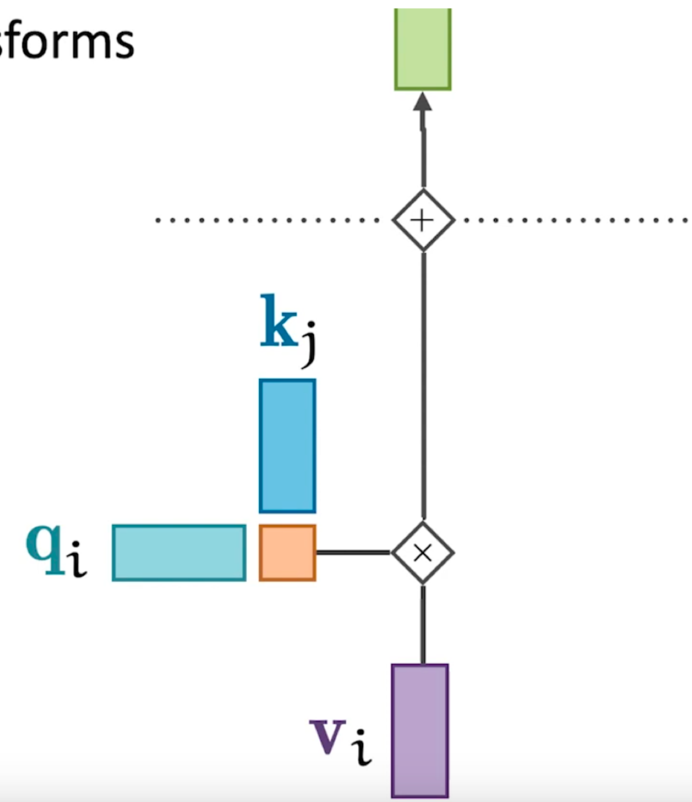
introduce matrices **K**, **Q**, **V** for linear transforms

and associated biases

$$\mathbf{k}_i = \mathbf{K}\mathbf{x}_i + \mathbf{b}_k$$

$$\mathbf{q}_i = \mathbf{Q}\mathbf{x}_i + \mathbf{b}_q$$

$$\mathbf{v}_i = \mathbf{V}\mathbf{x}_i + \mathbf{b}_v$$



Attention is Cheap!

	FLOPs
Self-Attention	$O(\text{length}^2 \cdot \text{dim})$
RNN (LSTM)	$O(\text{length} \cdot \text{dim}^2)$

specially attractive when your dim >> length (case of MT)

Question

- Given a query vector and two keys:

$q = [0.3, 0.2, 0.1]$

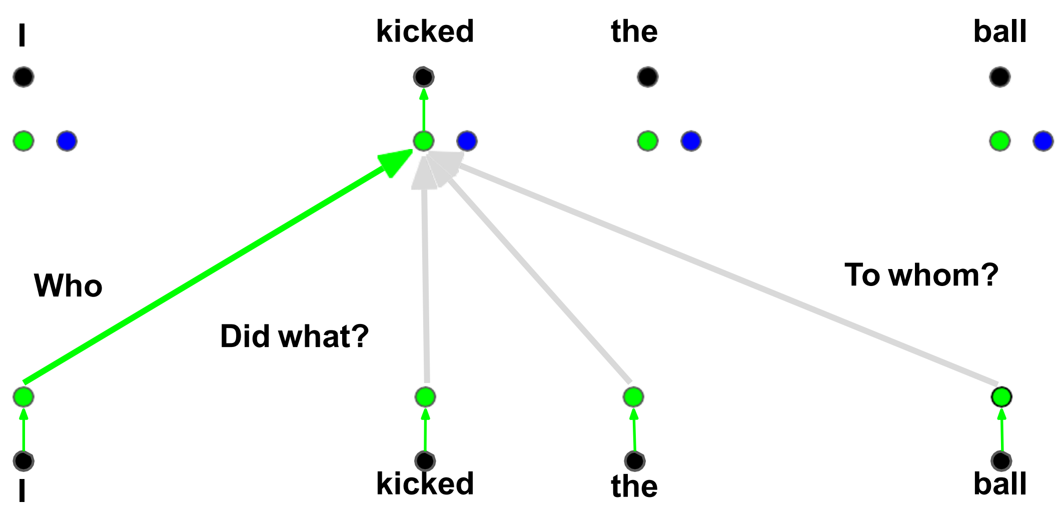
$k_1 = [0.1, 0.3, 0.1]$

$k_2 = [0.6, 0.4, 0.2]$

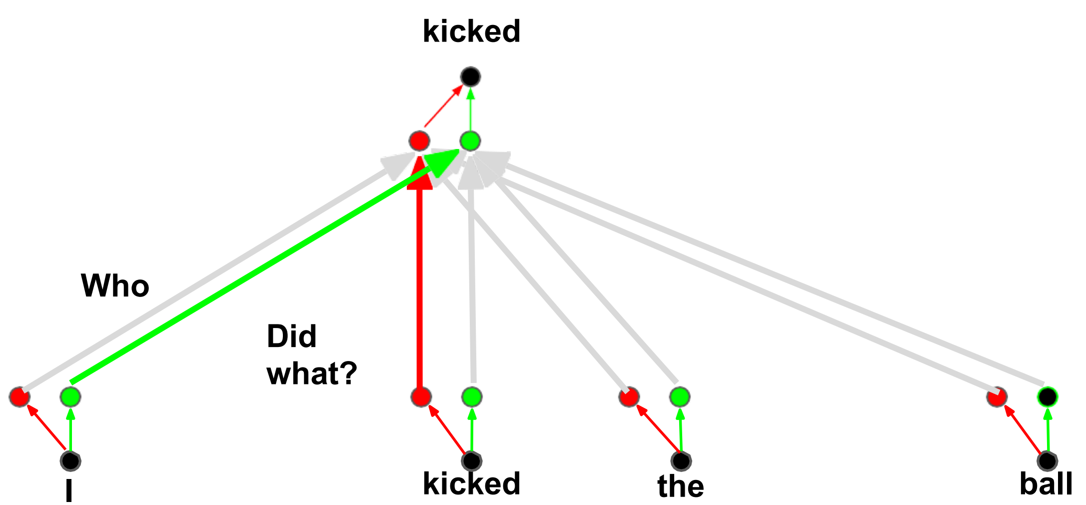
- What are the attention weights a_1 and a_2 computing the dot product ?
- What are the attention weights a_1 & a_2 when computing the scaled dot product ?
- Which key vector will receive more attention ?

Attention head: Who

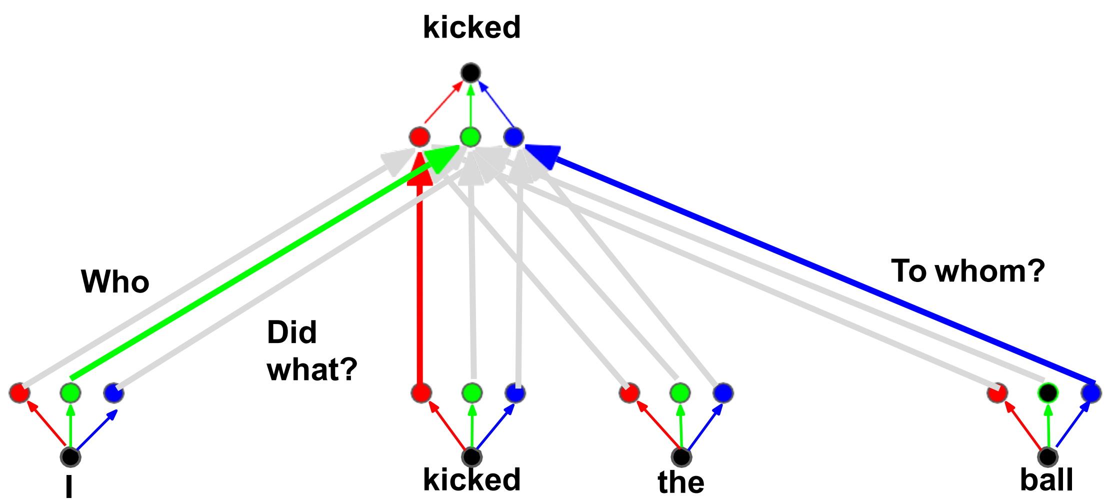
different words relate to each other by different relations



Attention head: Did What?

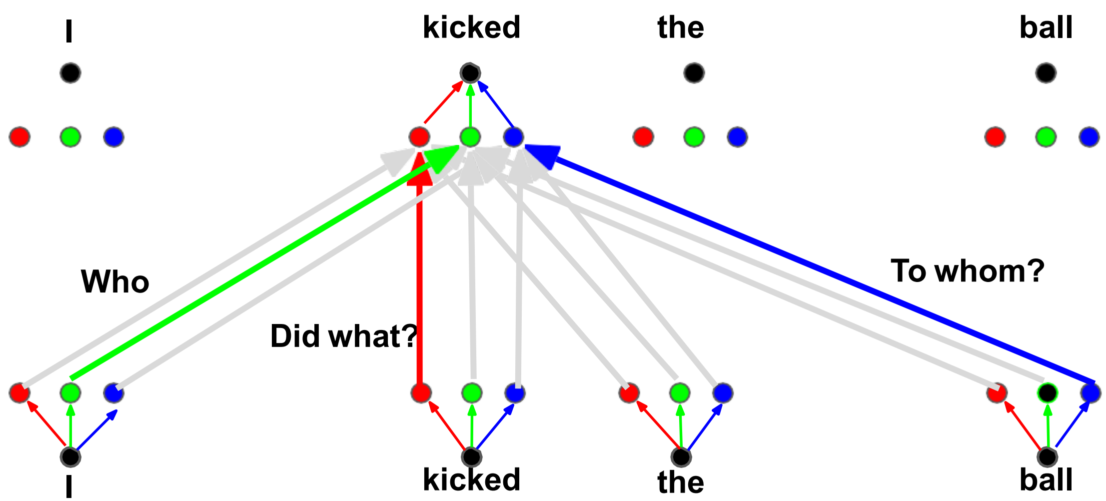


Attention head: To Whom?

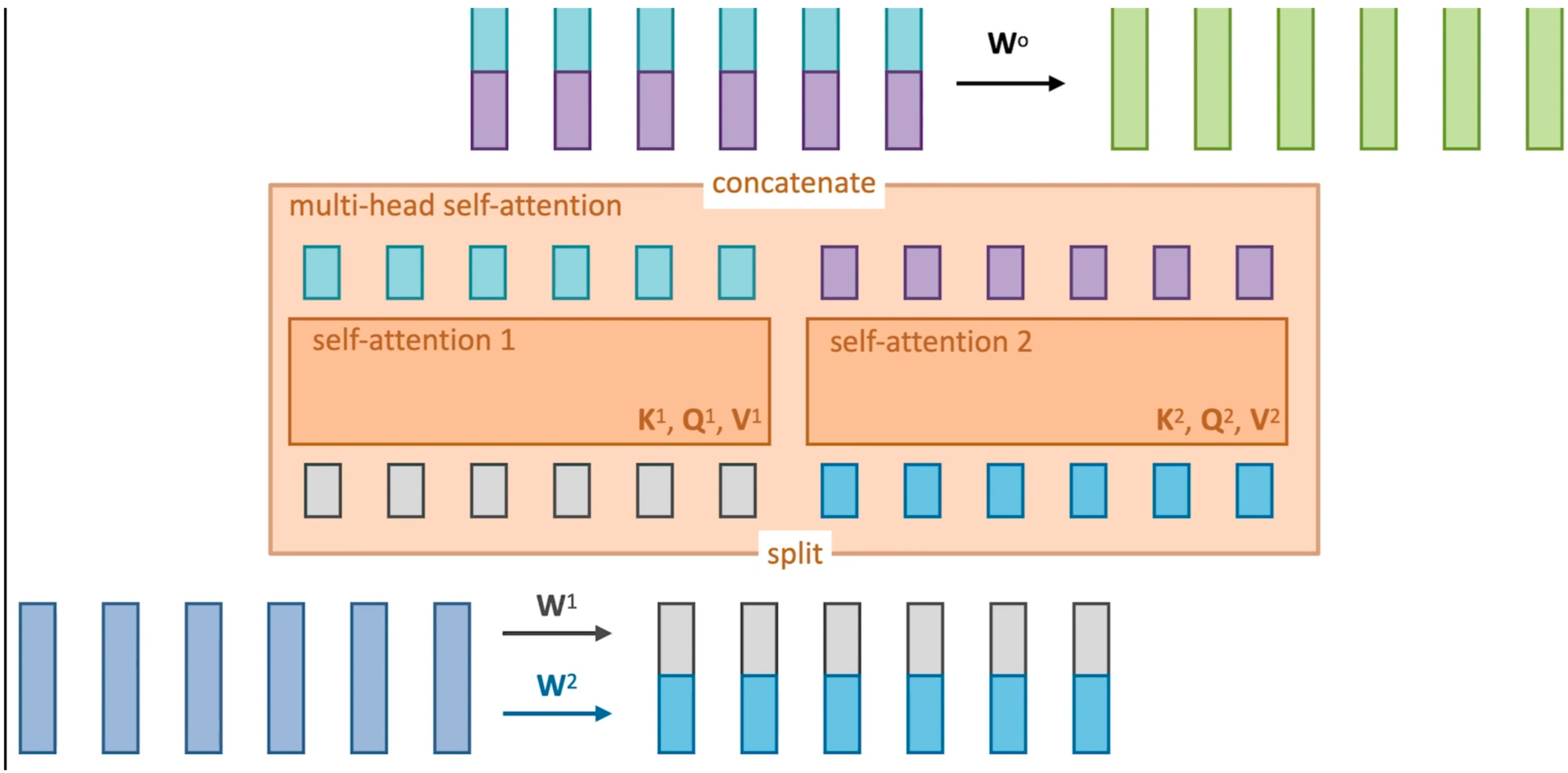


Parallel attention heads

To model all these different kinds of relation in one self-attention operation we split the self-attention into different heads which are basically self-attention layers applied in parallel



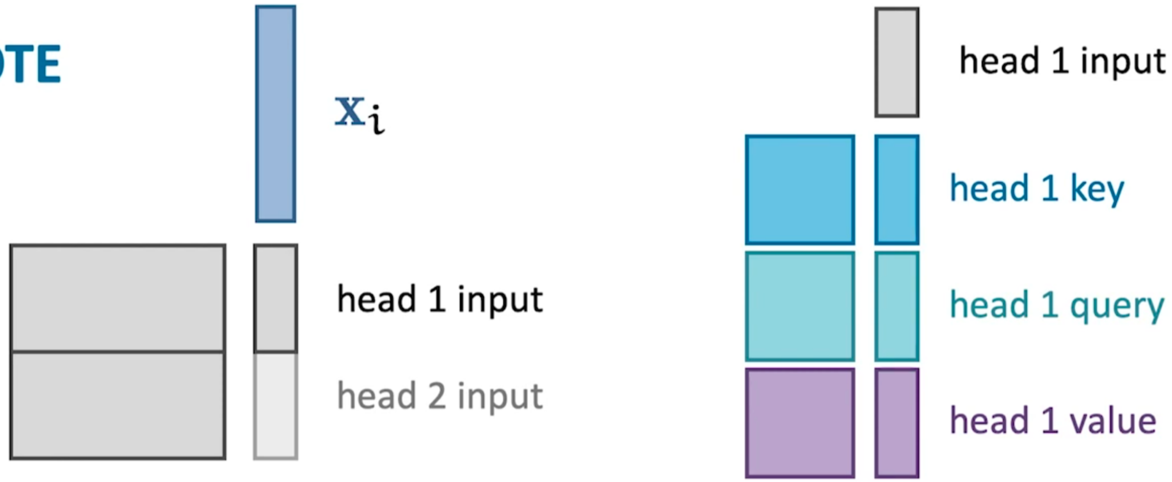
Multi-head attention

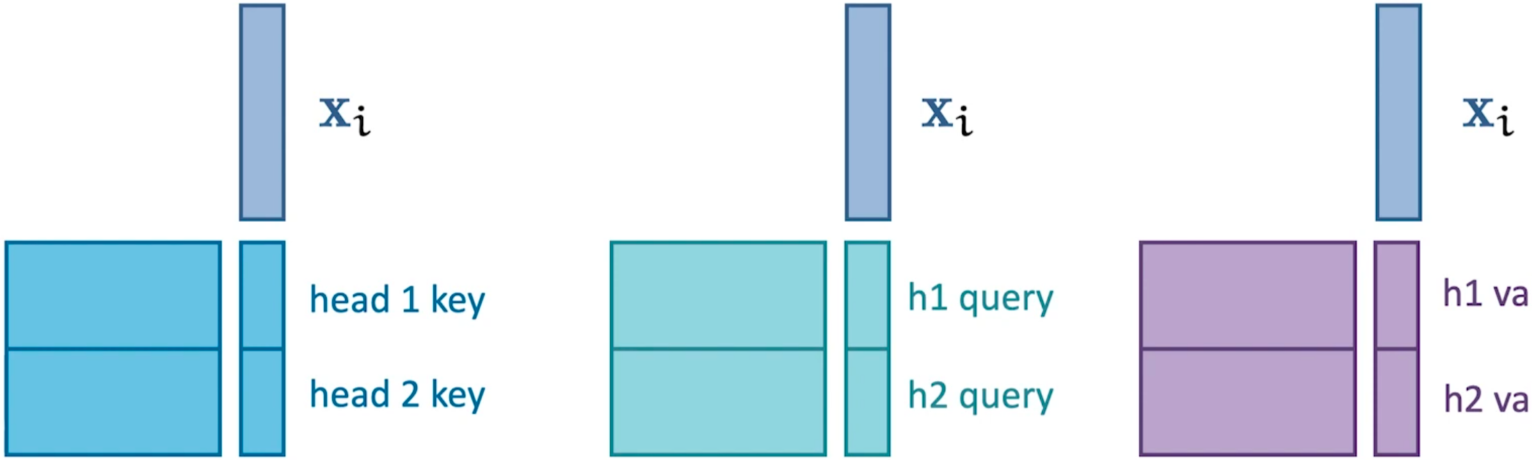


1. input sequence through linear operations to decrease dimensionality
2. each split of the input vector is fed into a head attention.

Multi-head attention

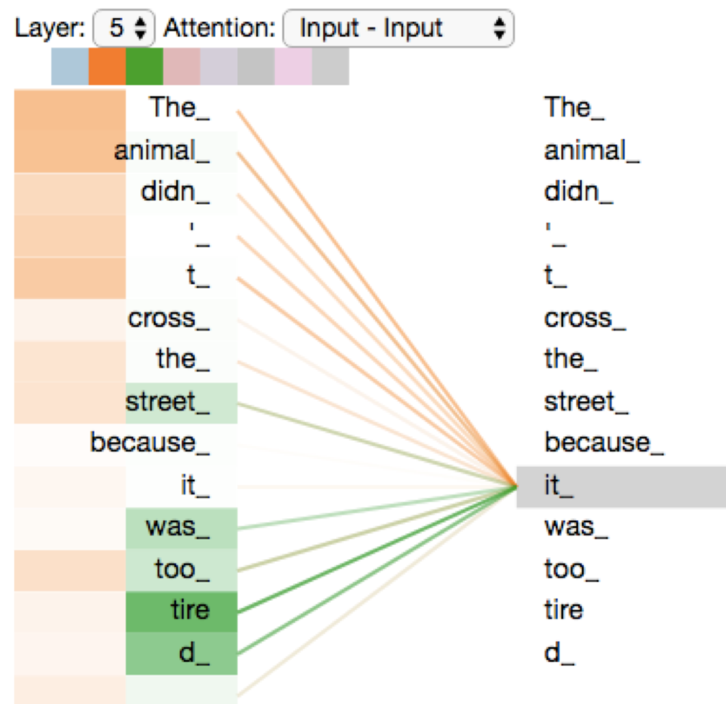
IMPLEMENTATION NOTE





Visualization

- As we encode the word "it", one attention head is focusing most on "the animal", while another is focusing on "tired" -- in a sense, the model's representation of the word "it" bakes in some of the representation of both "animal" and "tired".



https://www.youtube.com/watch?v=187JyiA4pyk

TRANSFORMER: POSITION INFORMATION

Relevance of position information

- This is not a real restaurant, it’s a filthy burger joint
- This is not a filthy Burger joint, it’s a real restaurant

The transformer contains no recurrence and no convolution.

We have to add positional information to the input word vectors

Methods:

Positional embeddings

Positional encodings

Positional embeddings

word embeddings:

$\mathbf{v}_{\text{the}}, \mathbf{v}_{\text{man}}, \mathbf{v}_{\text{pets}}, \mathbf{v}_{\text{cat}}, \mathbf{v}_{\text{again}}$

position embeddings:

$\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3, \mathbf{v}_4, \mathbf{v}_5, \ldots$

Question

- What is the problem with positional embeddings?

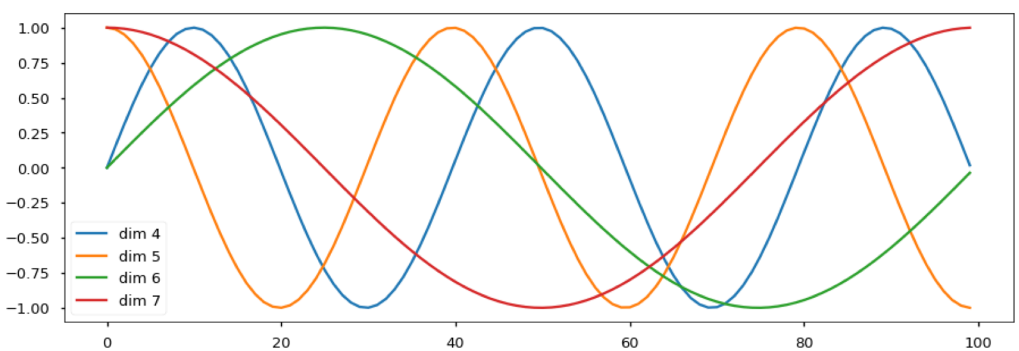
Positional Encoding

We can add positional encodings to the input word vectors:

- Fixed. A usual choice is sine and cosine functions of different frequencies, since it allow the model to attend by relative positions

$$PE(pos, dim) = \sin(\omega_i \cdot pos) \qquad if\ dim = 2i$$
$$PE(pos, dim) = \cos(\omega_i \cdot pos) \qquad if\ dim = 2i+1$$

$$\omega_i = \frac{1}{10000^{2 \cdot i / d_{embedding}}}$$



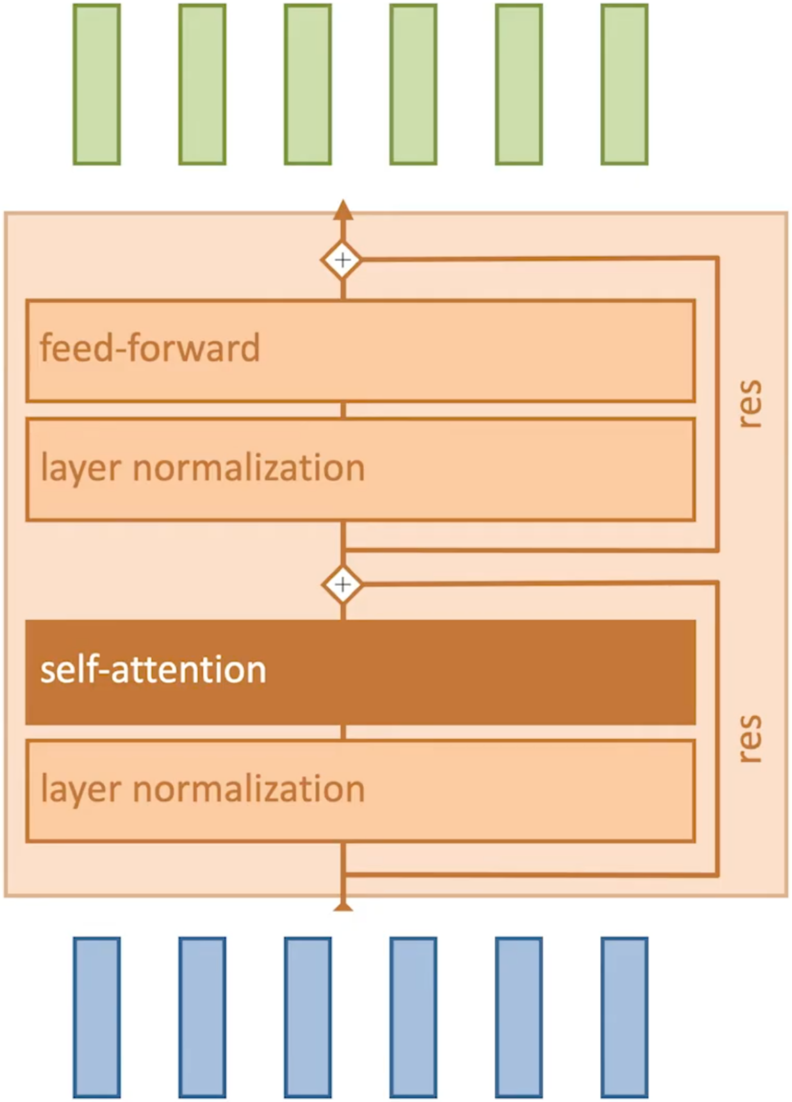
- *pos* is the position of the token in the sentence,
- *dim* the dimension of the embeddings
- *i* the position within the embedding.

Questions

- Why positional embeddings are summed with word embeddings instead of concatenation?
- Doesn't the position information get vanished once it reaches the upper layers?

TRANSFORMER: LAYERS/BLOCKS

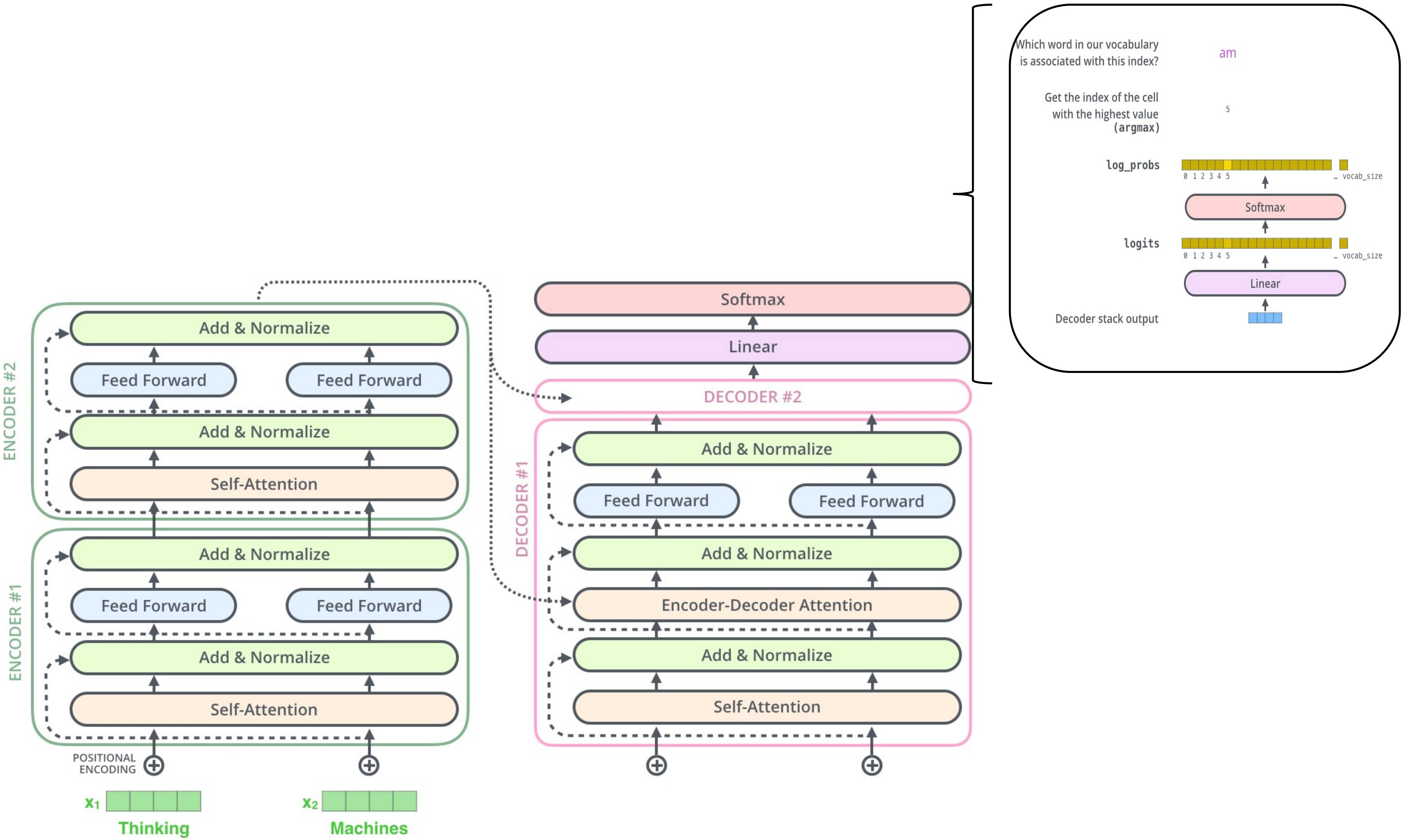
Transformer layers



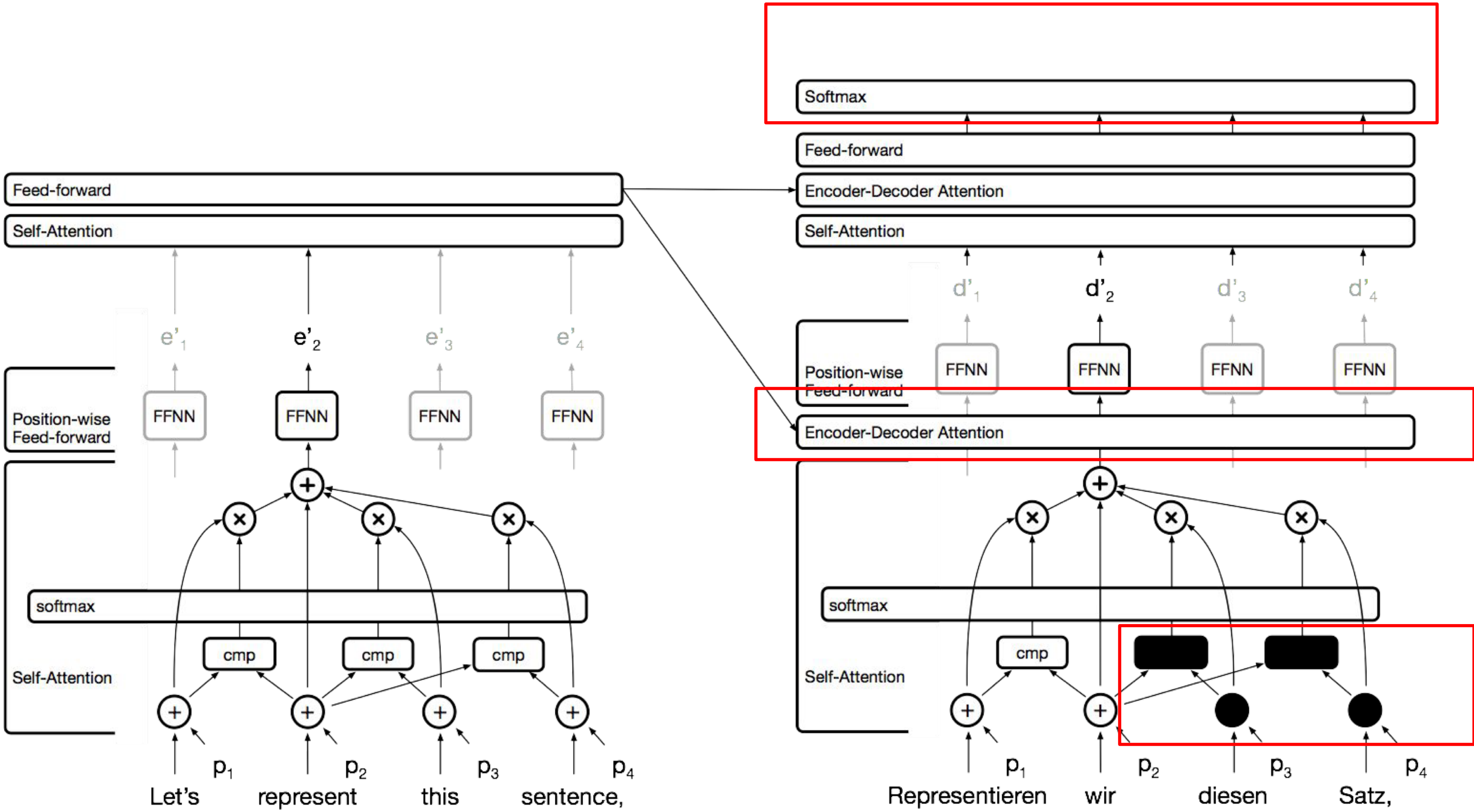
layer normalization

residual connections

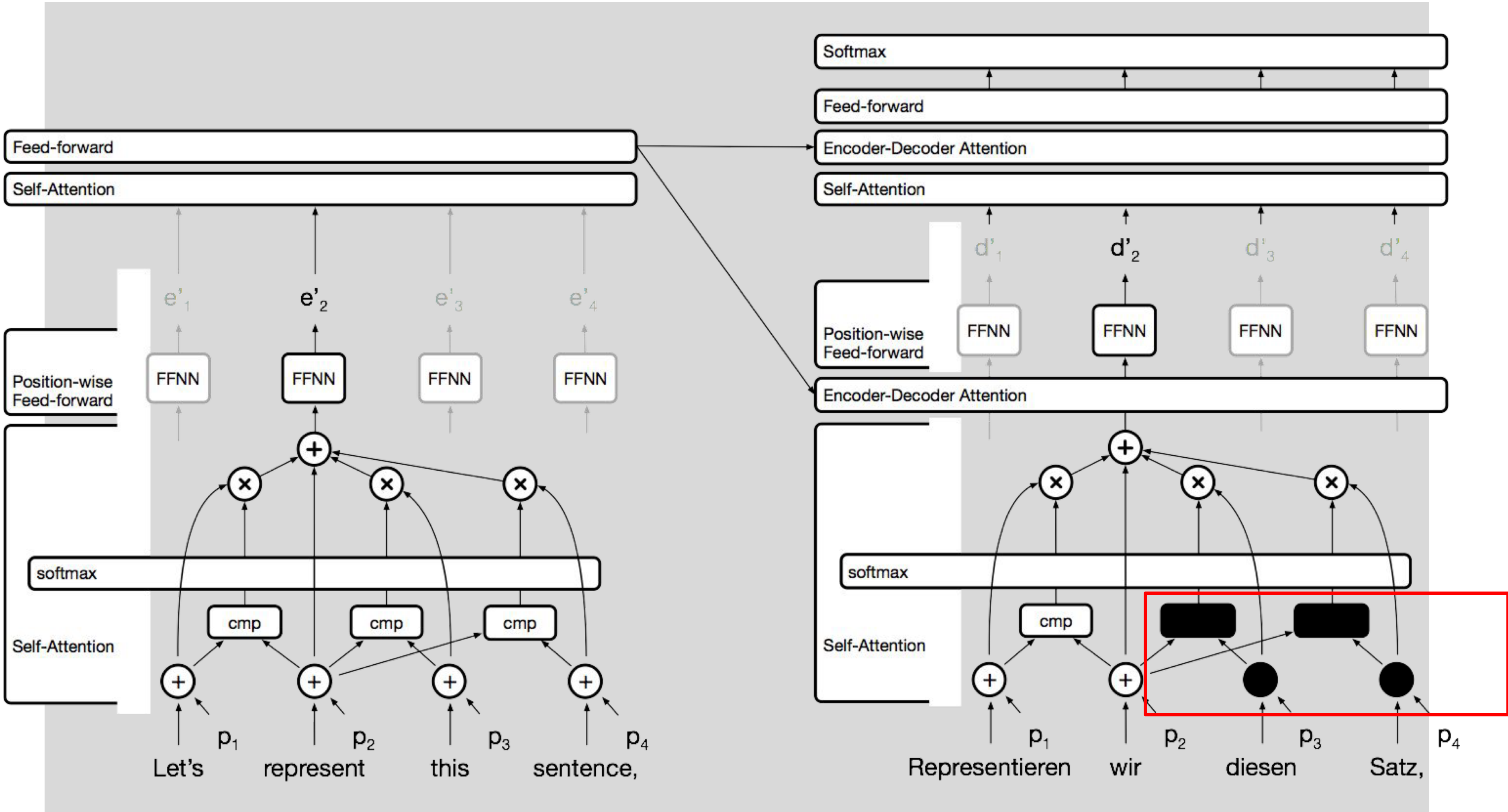
Transformer of 2 stacked encoders and decoders



The Transformer: Encoder vs Decoder layers



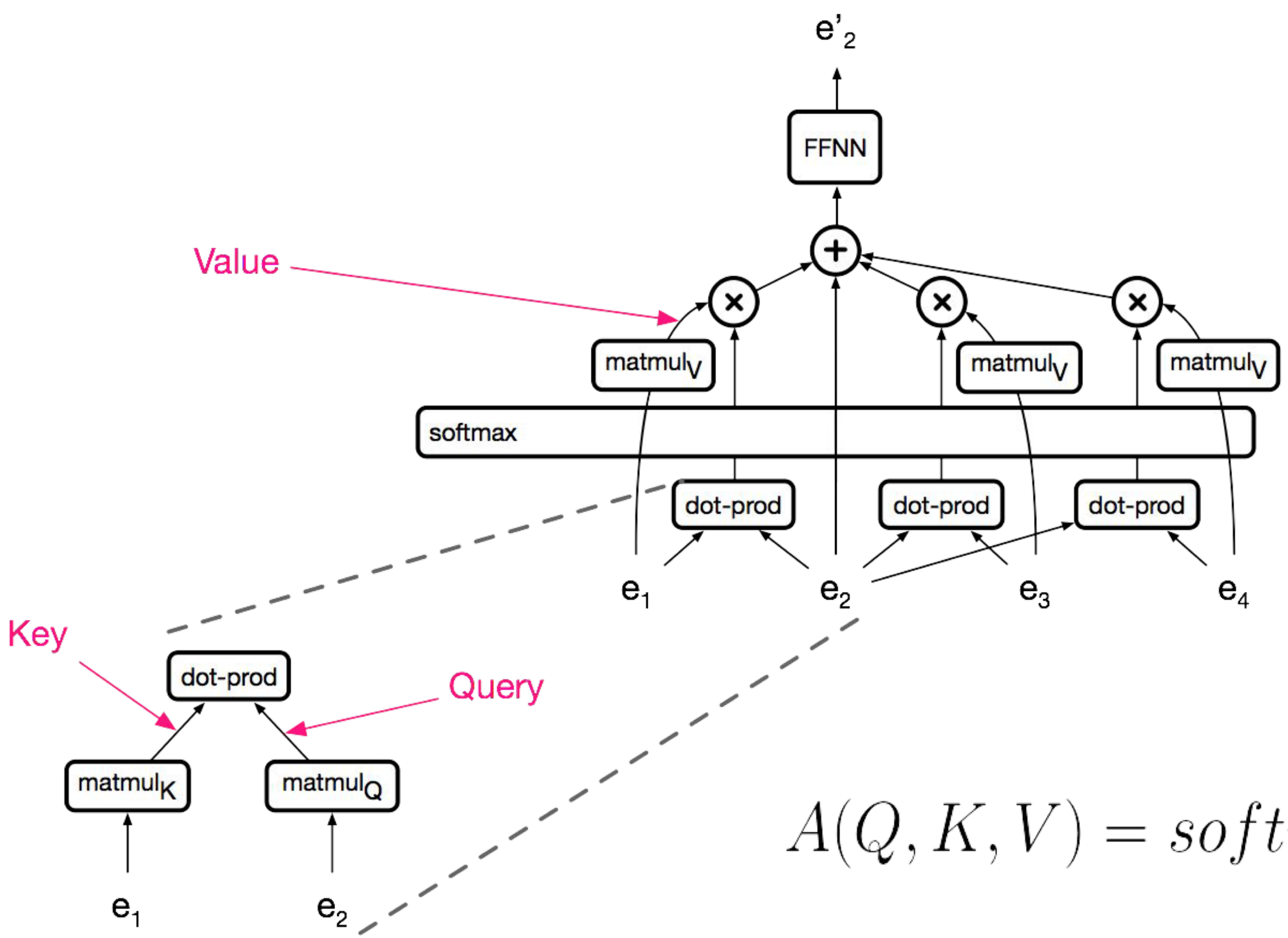
QUESTION:
What are we doing in the red square?



Encoder vs Decoder layers

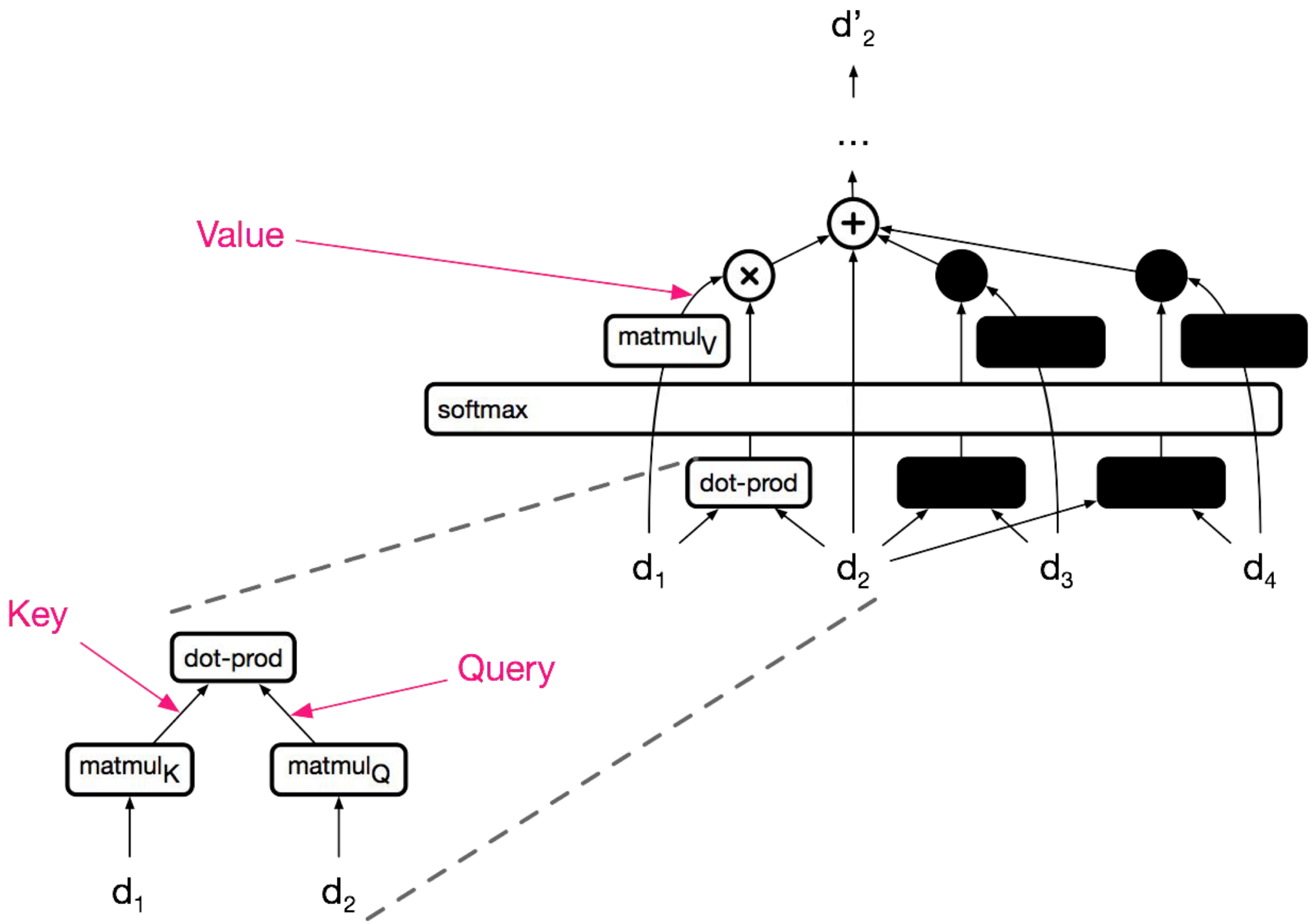
- Masking. The decoder cannot see *the future* when predicting the next word.
- Enc-Dec Attention. Queries are taken from the layer below it, but keys and values from the final output of the encoder.
- The decoder adds an additional linear and softmax layer (just as RNNs NMT)

Encoder Self-Attention (no masking)



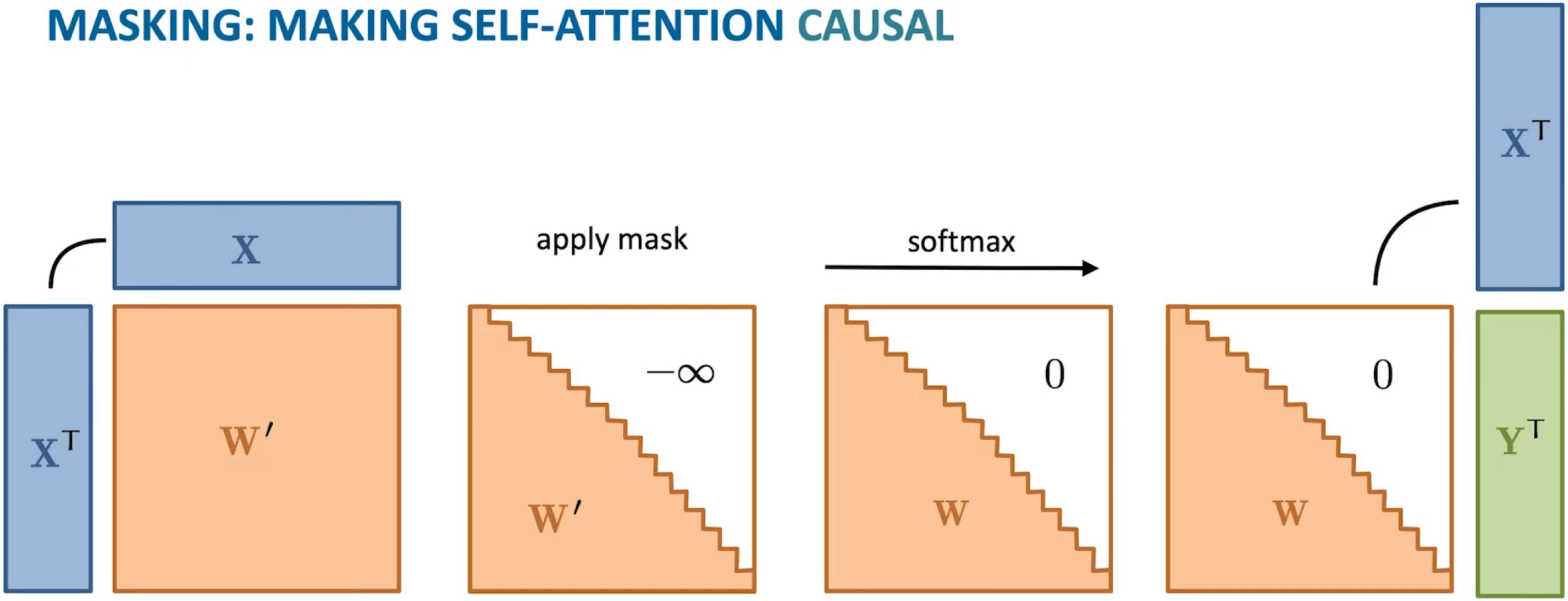
$$A(Q,K,V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$$

Decoder Self-Attention (with masking)



Masking

MASKING: MAKING SELF-ATTENTION CAUSAL

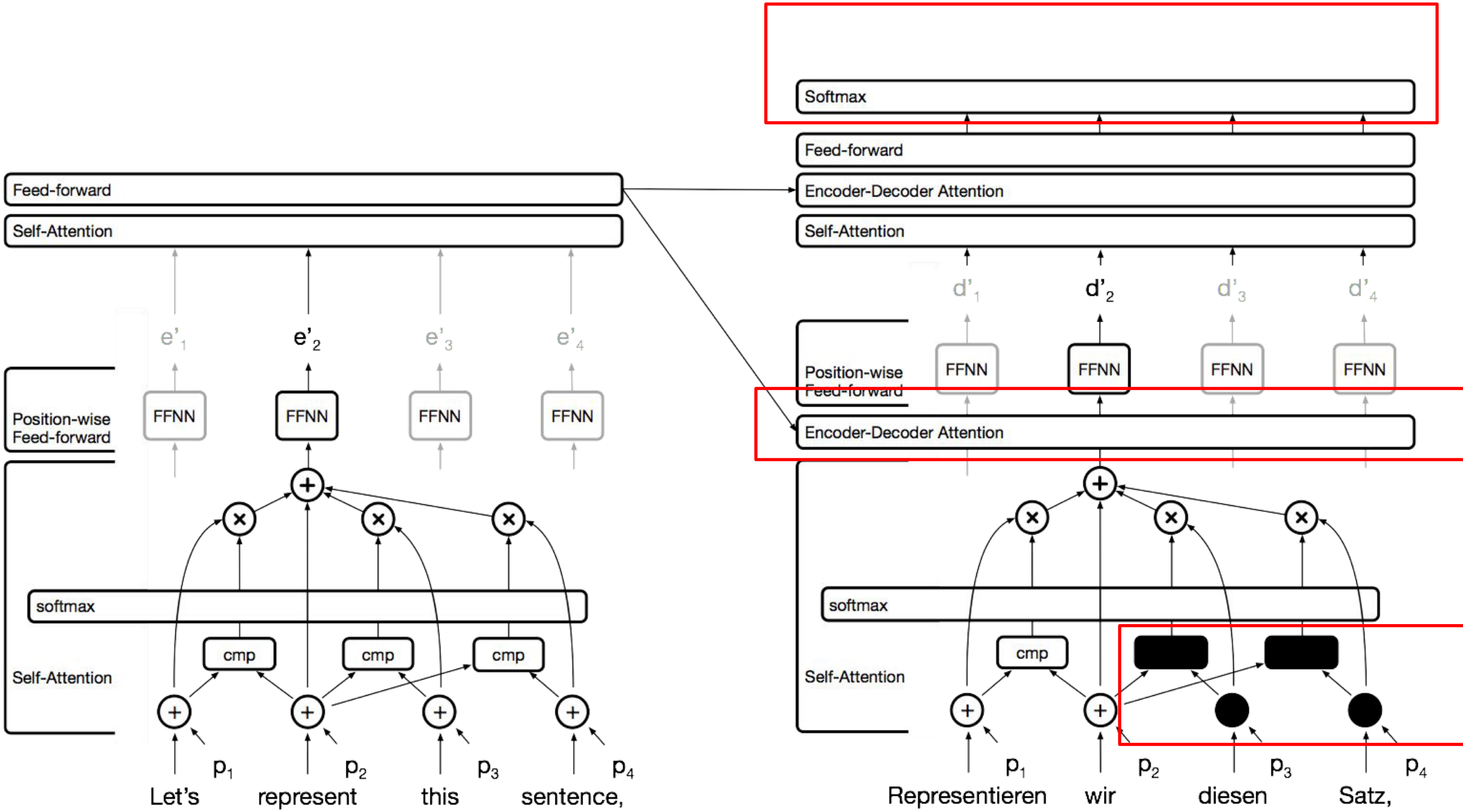


$W' = X^T X$ $\quad W'_{ij} \leftarrow -\infty \text{ if } j > i$ $\quad W = \text{softmax}(W')$ $\quad Y^T = W X^T$

Encoder vs Decoder layers

- Masking. The decoder cannot see *the future* when predicting the next word.
- Enc-Dec Attention. Queries are taken from the layer below it, but keys and values from the final output of the encoder.
- The decoder adds an additional linear and softmax layer (just as RNNs NMT)

The Transformer: Encoder vs Decoder layers



Complete picture

THE ORIGINAL TRANSFORMER

machine translation model

no recurrent layers or convolutions

encoder/decoder configuration

teacher forcing

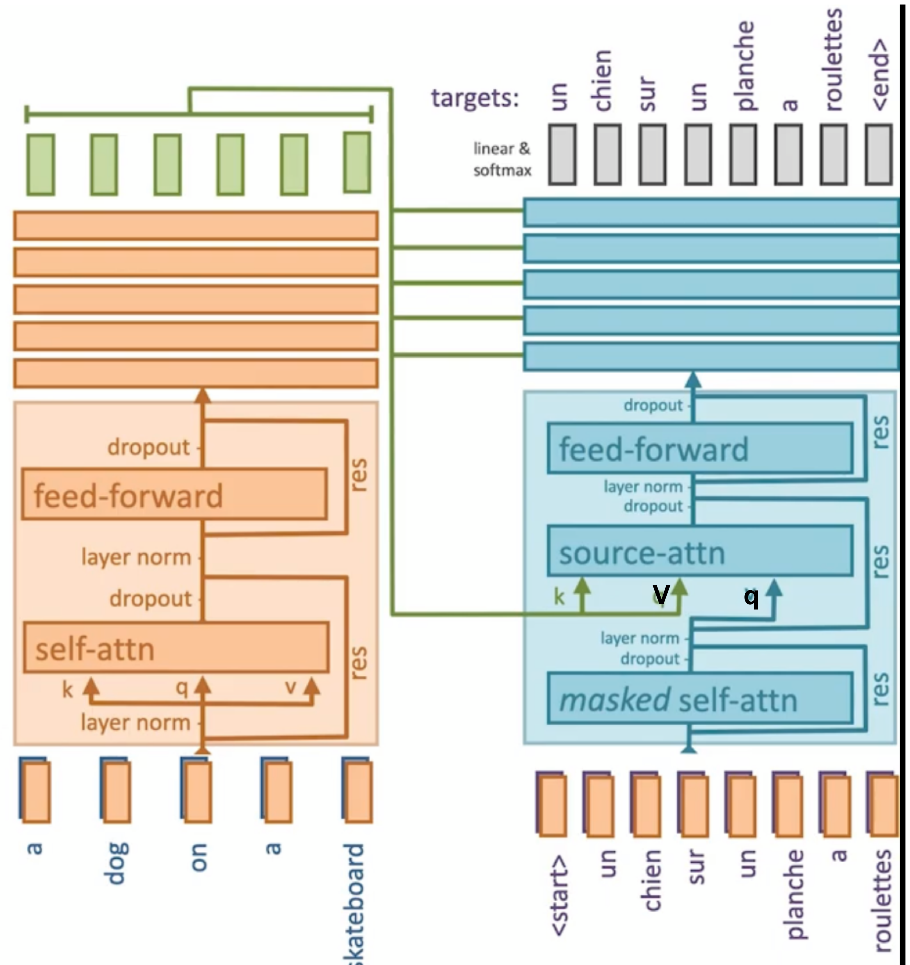
position *encoding*

512 dims, 8 heads, 2x6 blocks

FF: `Lin(512, 2048), relu, Lin(2048, 512)`

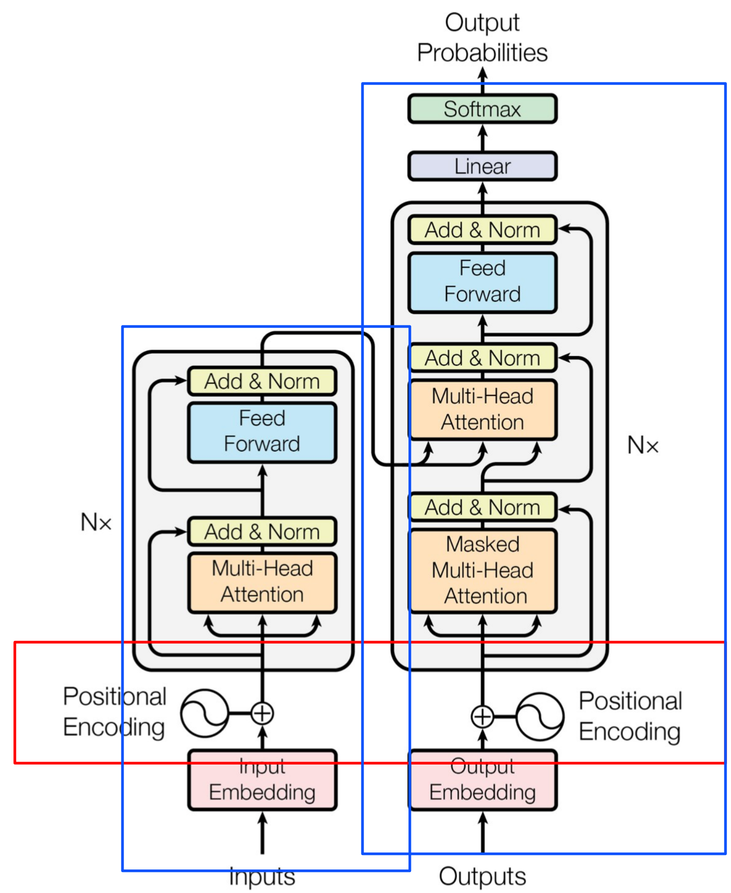
trained for 3.5 days on 8 GPUs

Attention Is All You Need, Vaswani et al, 2017.



Recap

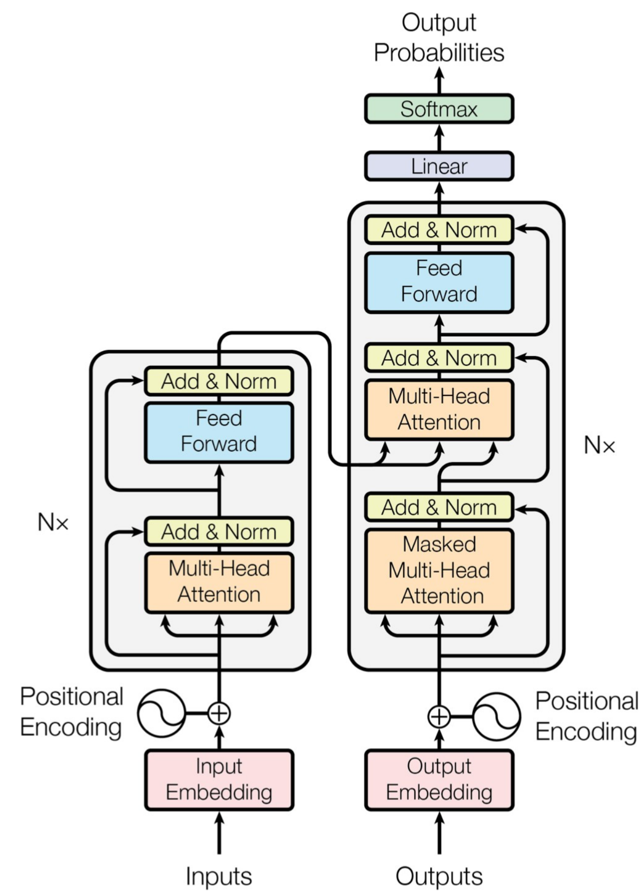
- Key Concepts of the Transformer
  - Self Attention
  - Multi-Head Attention
- Position Information
- Transformer Layers/Blocks
- Encoder vs Decoder (Masking, Inter Attention, Softmax)



Complete picture and recap

Self-Attention

- Constant ‘path length’ between any two positions.
- Unbounded memory.
- Trivial to parallelize (per layer).
- Models Self-Similarity.



Active Research Area

- Non autoregressive transformer (Gu and Bradbury et al., 2018)
- Improving Language Understanding by Generative Pre-Training (Radford, Narsimhan, Salimans, and Sutskever)
- BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding (Devlin, Chang, Lee, and Toutanova)
- Universal Transformers (ICLR 2019). Deghiani*, Gouws*, Vinyals, Uszkoreit, Kaiser.
- Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context (2019). Dai, Yang, Yang, Carbonell, Le, Salakhutdinov.