Transformers and Large Language Models

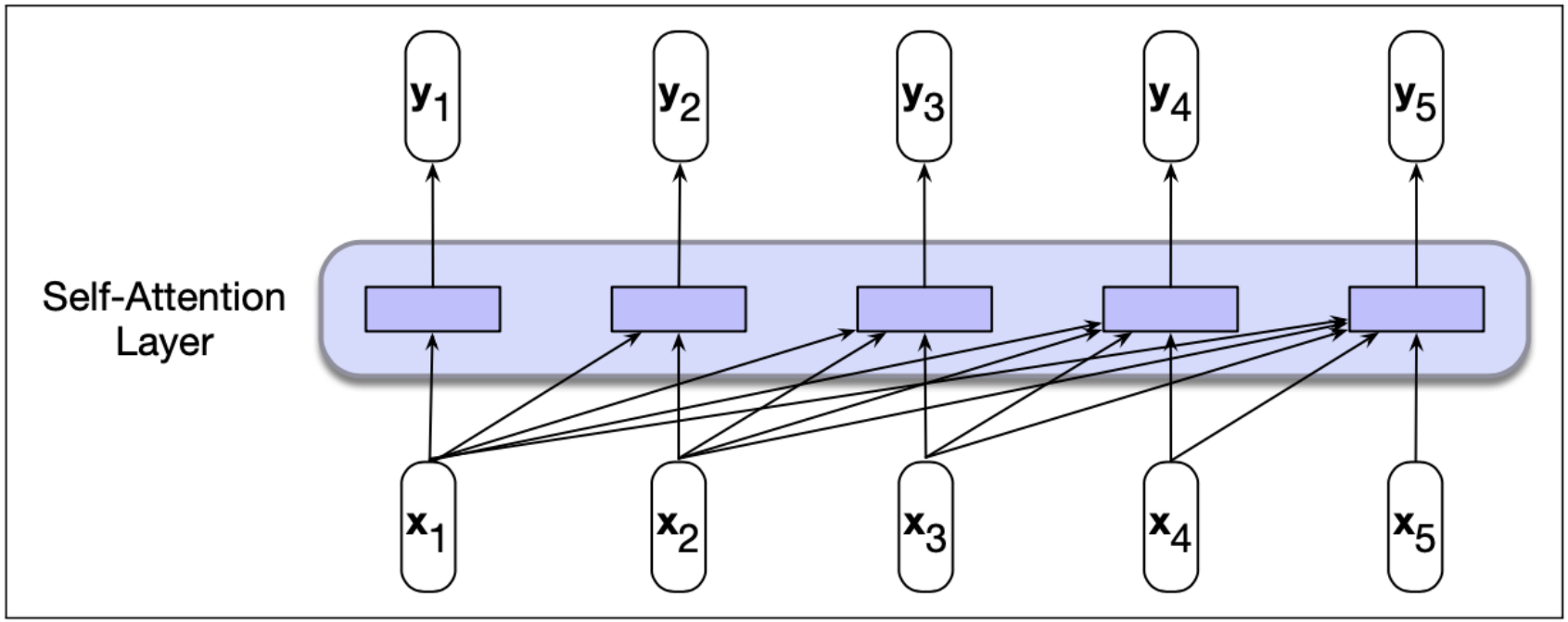
Erhard Hinrichs

Seminar für Sprachwissenschaft
Eberhard-Karls Universität Tübingen

Transformers

- Transformers are non-recurrent networks based on (self-)attention.
- Self-attention allows a network to directly extract and use information from arbitrarily large contexts without the need to pass them through intermediate recurrent connections as in RNNs.
- A self-attention layer maps input sequences to output sequences of the same length, using attention heads.
- Attention heads model how the surrounding words are relevant for the processing of the current word.

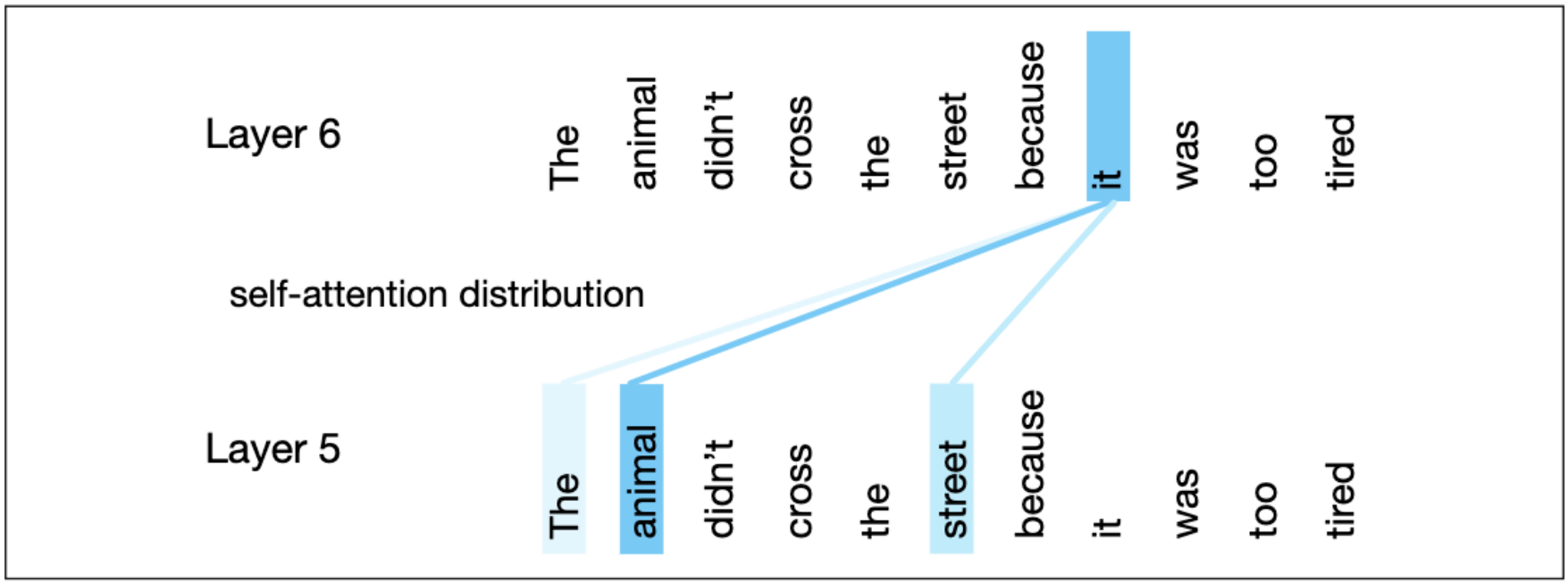
Self-Attention Layer



Relevant Linguistic Examples Motivating Self-Attention

1. The **keys** to the cabinet **are** on the table.
2. The **chicken** crossed the road because **it** wanted to get to the other side.
3. I walked along the **pond**, and noticed that one of the trees long the **bank** had fallen into the **water** after the storm.

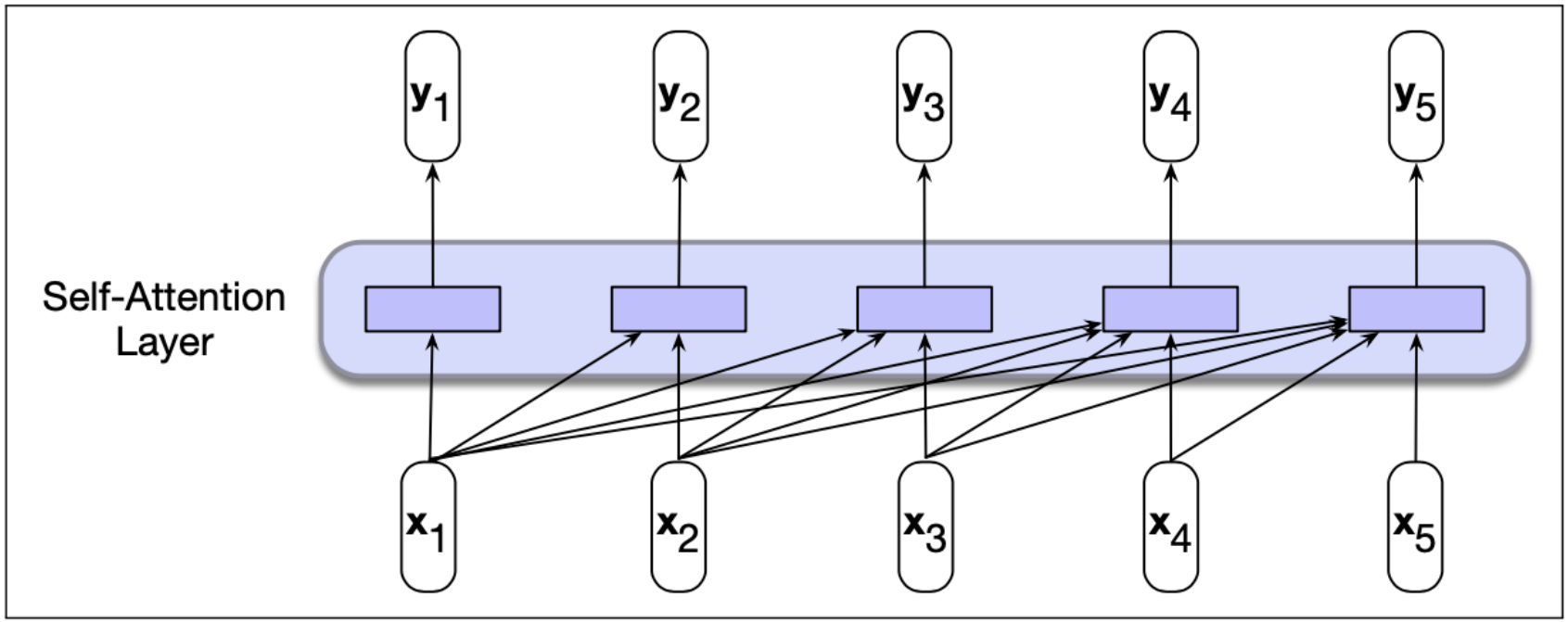
Self-Attention Weight Distribution



Flow of Information in a Self-Attention Layer

- When processing each item in the input of a self-attention layer, the model has access to all inputs up to and including the one under consideration, but no access to information about inputs beyond the current one.
- The computation performed for each item is independent of all the other computations. Hence, forward inference and training can proceed in parallel.

Self-Attention Layer



Self-Attention Networks: Transformers

$$\text{score}(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i \cdot \mathbf{x}_j \tag{1}$$

$$\alpha_{ij} = \text{softmax}(\text{score}(\mathbf{x}_i, \mathbf{x}_j)) \; \forall j \leq i \tag{2}$$

$$= \frac{\exp(\text{score}(\mathbf{x}_i, \mathbf{x}_j))}{\sum_{k=1}^{i} \exp(\text{score}(\mathbf{x}_i, \mathbf{x}_k))} \; \forall j \leq i \tag{3}$$

$$\mathbf{y}_i = \sum_{j \leq i} \alpha_{ij} \mathbf{x}_j \tag{4}$$

Self-Attention Networks: Different Roles

An input embedding can play three different roles:

- **query**: as the *current focus of attention* that is compared to all of the other preceding inputs.
- **key**: as *a preceding input* that is compared to the current focus of attention.
- **value** that is used to compute the output for the current focus of attention.

Self-Attention Networks: Transformers

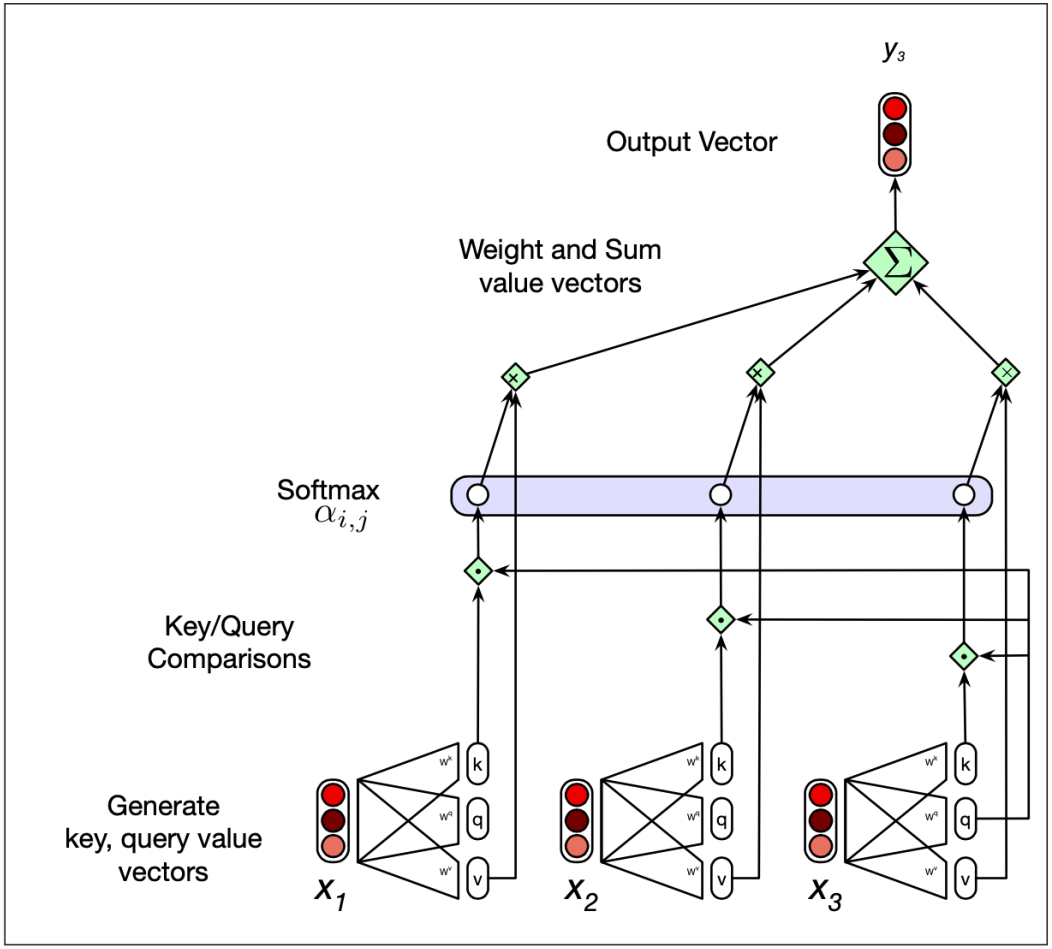
- The roles of query, key, and value are differentiated by three different weight matrices: $\mathbf{W^Q} \in \mathbb{R}^{d \times d'}$, $\mathbf{W^K} \in \mathbb{R}^{d \times d'}$, and $\mathbf{W^V} \in \mathbb{R}^{d \times d''}$.
- The inputs and outputs of transformers, as well as the intermediate vectors after the various layers, all have the same dimensionality $1 \times d$.

$$\mathbf{q}_i = \mathbf{W^Q}\mathbf{x}_i;\ \mathbf{k}_i = \mathbf{W^K}\mathbf{x}_i;\ \mathbf{v}_i = \mathbf{W^V}\mathbf{x}_i \tag{5}$$

$$\text{score}(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{q}_i \cdot \mathbf{k}_j \tag{6}$$

$$\mathbf{y}_i = \sum_{j \leq i} \alpha_{ij}\mathbf{v}_j \tag{7}$$

Calculation of an Output Embedding in a Self-Attention Layer



Self-Attention Networks: Further Adjustments

Scaling the dot product by the square root of the dimensionality

$$\text{score}(\mathbf{x}_i, \mathbf{x}_j) = \frac{\mathbf{q}_i \cdot \mathbf{k}_j}{\sqrt{d_k}} \tag{8}$$

Parallelizing the Cmputation: packing the input embeddings of the N tokens into a single matrix

$$\mathbf{Q} = \mathbf{X}\mathbf{W}^{\mathbf{Q}};\ \mathbf{K} = \mathbf{X}\mathbf{W}^{\mathbf{K}};\ \mathbf{V} = \mathbf{X}\mathbf{W}^{\mathbf{V}}; \tag{9}$$

Final Result: Reducing the Self-Attention Step for an Entire Sequence of N Tokens

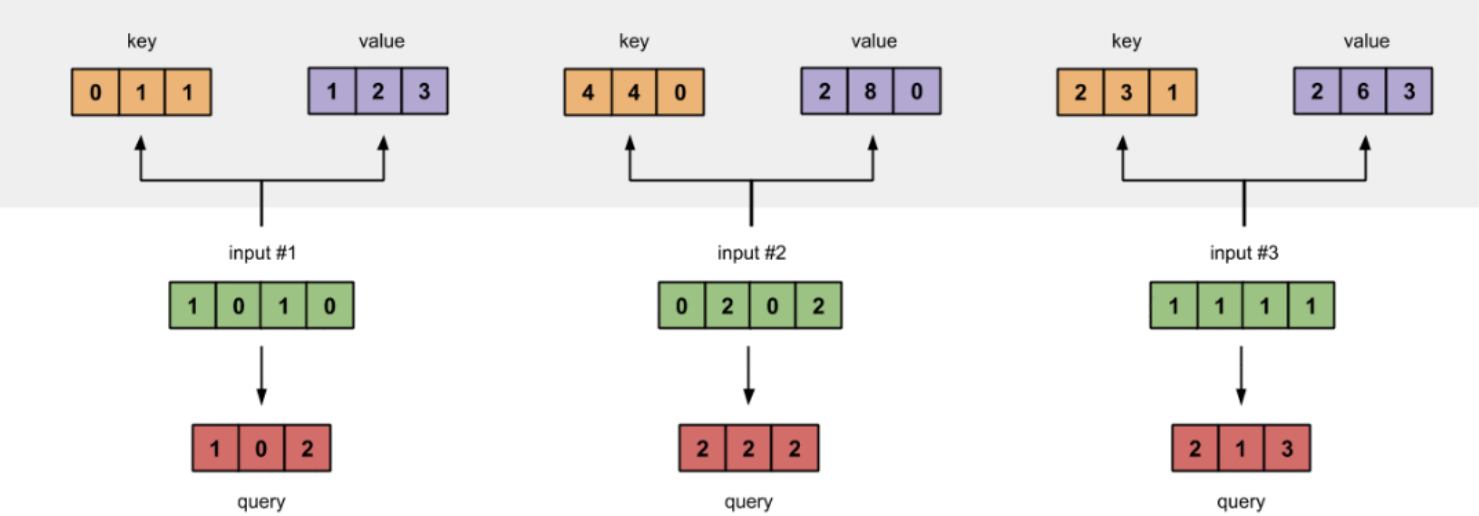
$$\textit{SelfAttention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right)\mathbf{V} \tag{10}$$

Working through an Example with 3 Input Vectors

1. Prepare inputs
2. Initialise weights
3. Derive key, query and value
4. Calculate attention scores for Input 1
5. Calculate softmax
6. Multiply scores with values
7. Sum weighted values to get Output 1
8. Repeat steps 4–7 for Input 2 and Input 3

This workflow and the illustrations in the next three slides are due to `https://towardsdatascience.com/illustrated-self-attention-2d627e33b20a`

Step 1-3


key
value
0 1 1
1 2 3
input #1
1 0 1 0
1 0 2
query
key
value
4 4 0
2 8 0
input #2
0 2 0 2
2 2 2
query
key
value
2 3 1
2 6 3
input #3
1 1 1 1
2 1 3
query

Initialize Weights for Query, Key, and Value

Weights for **key**:

```
[[0, 0, 1],
 [1, 1, 0],
 [0, 1, 0],
 [1, 1, 0]]
```

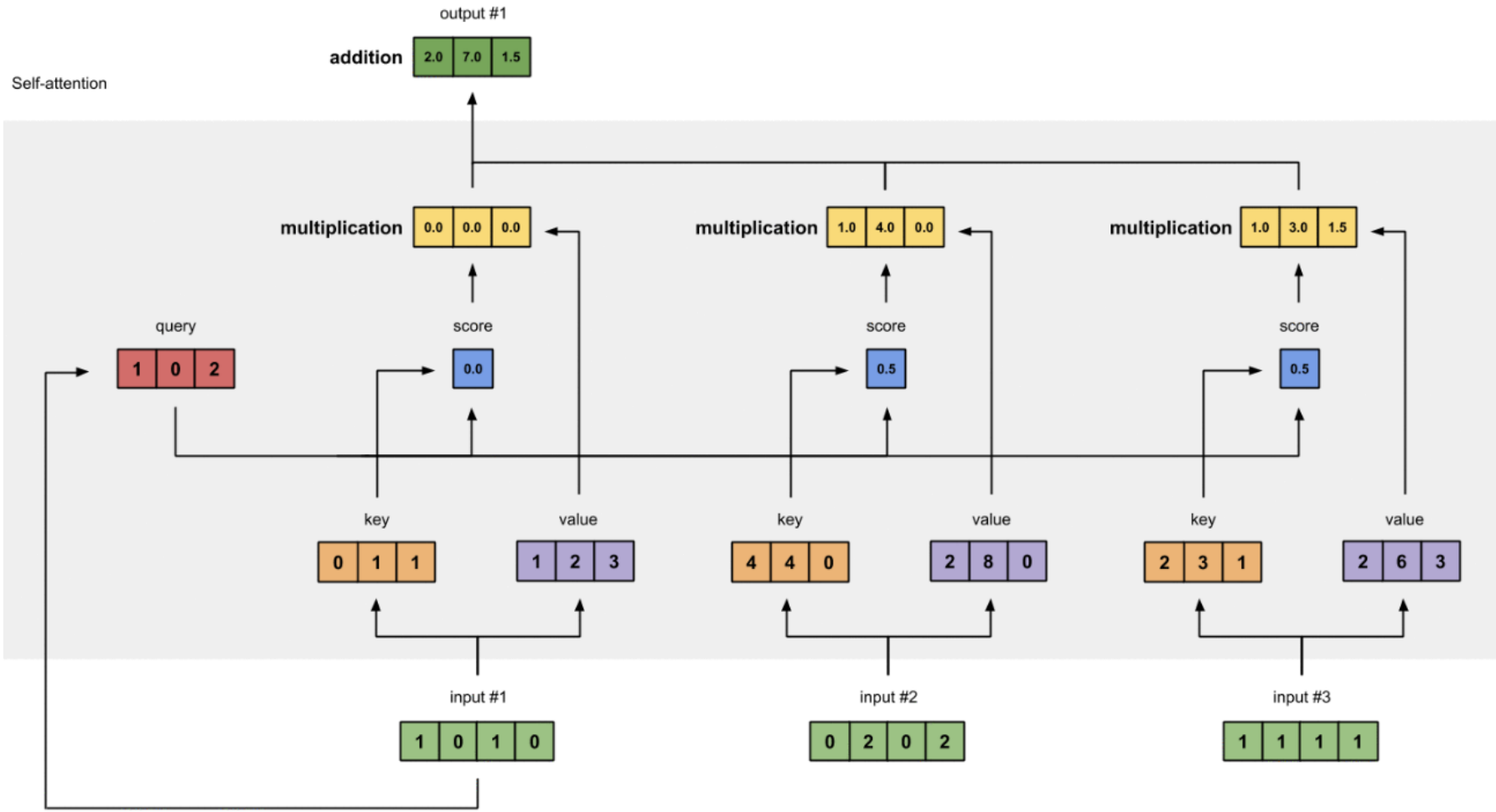
Weights for **query**:

```
[[1, 0, 1],
 [1, 0, 0],
 [0, 0, 1],
 [0, 1, 1]]
```

Weights for **value**:

```
[[0, 2, 0],
 [0, 3, 0],
 [1, 0, 3],
 [1, 1, 0]]
```

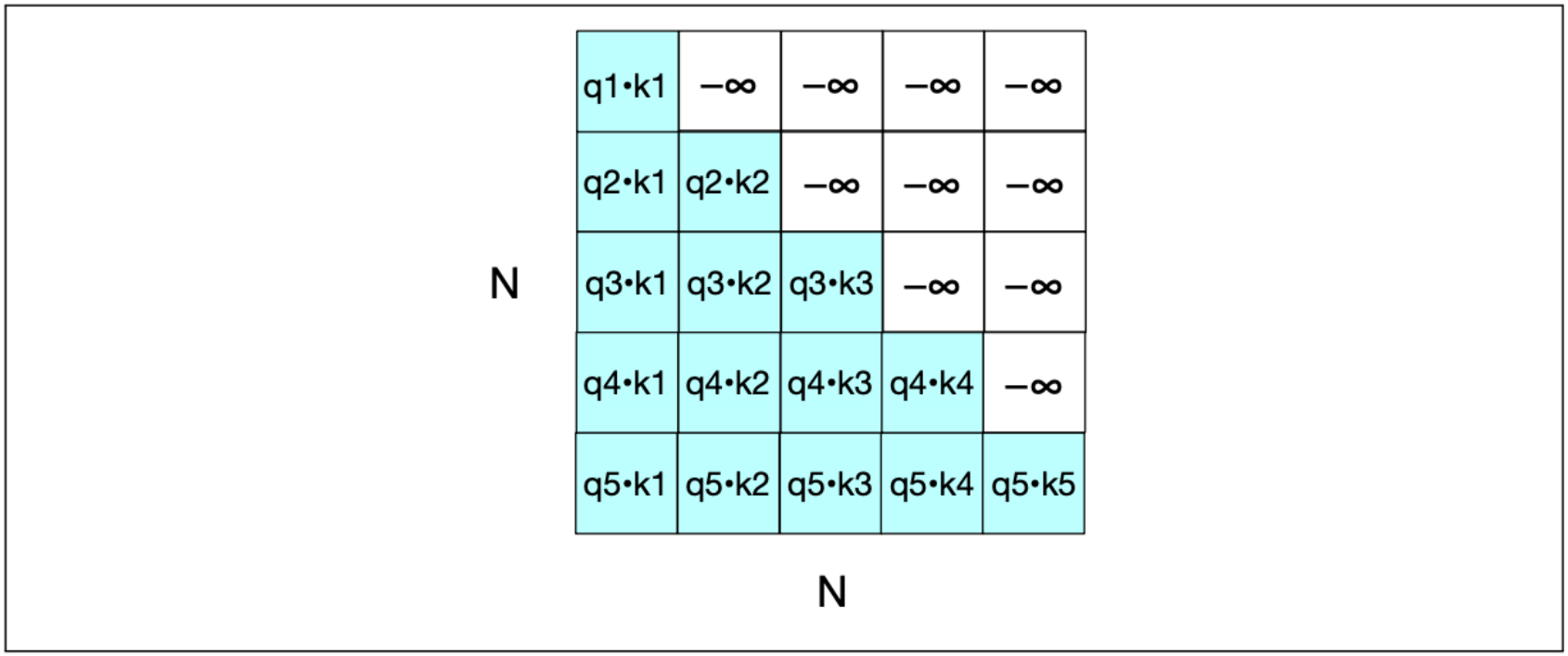
Calculate Attention Scores



Calculate Softmax

$$softmax(z_i) = \frac{exp(z_i)}{\sum_{j=1}^{k} exp(z_j)} \quad 1 \leq i \leq k \tag{11}$$

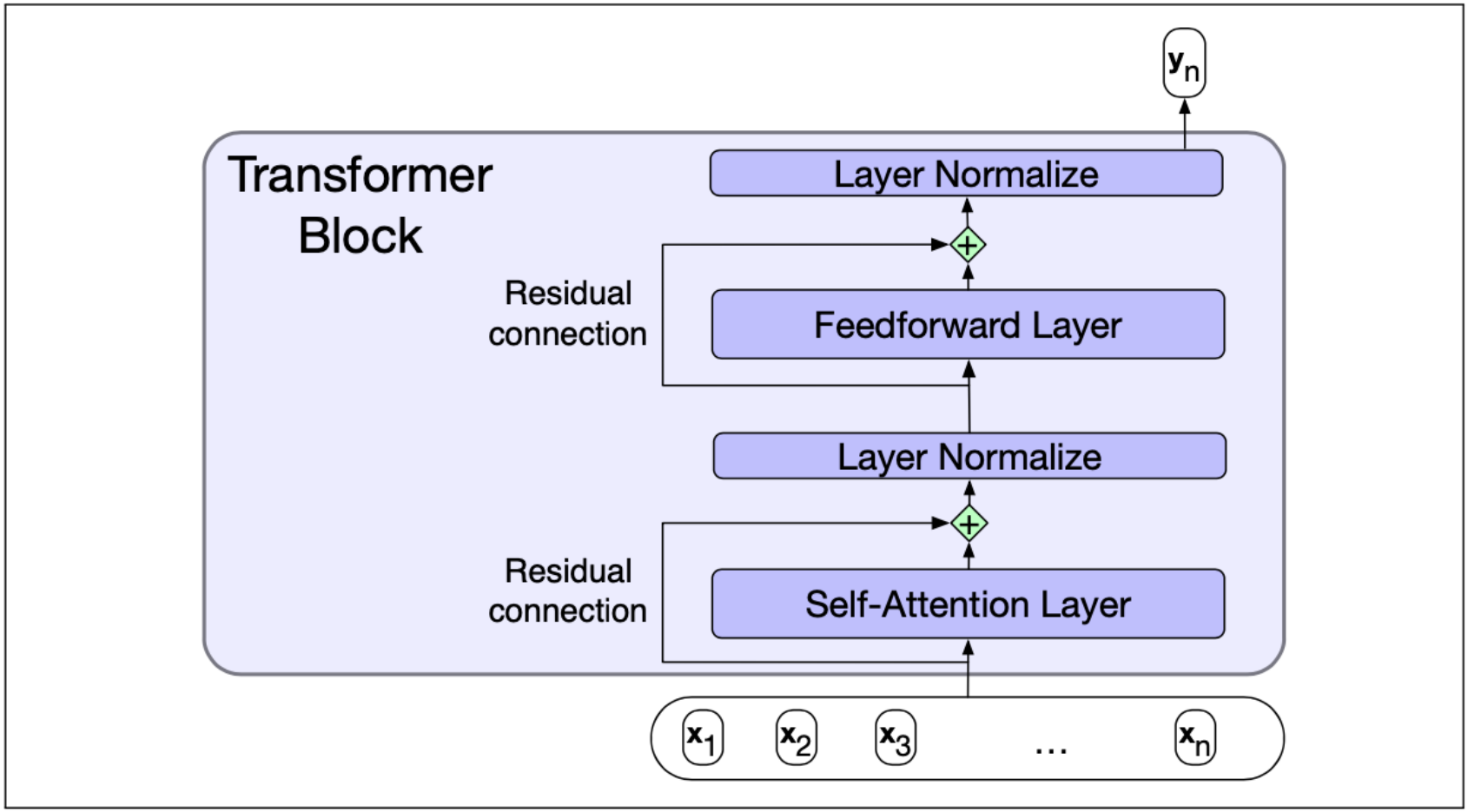
Zeroing Out the Upper Triangular Portion of a QT^T Matrix



Transformer Blocks

- A transformer block consists of a single attention layer followed by a feed-forward layer with residual connections and layer normalizations following each.
- Transformer blocks can be stacked to make deeper and more powerful networks.

Transformer Blocks



Transformer Blocks

$$\mathbf{z} = \text{LayerNorm}(\mathbf{x} + \text{SelfAttn}(\mathbf{x})) \tag{12}$$

$$\mathbf{y} = \text{LayerNorm}(\mathbf{z} + \text{FFNN}(\mathbf{z})) \tag{13}$$

$$\mu = \frac{1}{d_h}\sum_{i=1}^{d_h} x_i \tag{14}$$

$$\sigma = \sqrt{\frac{1}{d_h}\sum_{i=1}^{d_h}(x_i - \mu)^2} \tag{15}$$

$$\hat{\mathbf{x}} = \frac{(\mathbf{x} - \mu)}{\sigma} \tag{16}$$

$$\mathit{LayerNorm} = \gamma\,\hat{\mathbf{x}} + \beta \tag{17}$$

Multihead Attention Layers

- are sets of self-attention layers, each with its own set of key, query, and value matrices:
  - $\mathbf{W}_i^Q \in \mathbb{R}^{d \times d_k}$
  - $\mathbf{W}_i^K \in \mathbb{R}^{d \times d_k}$
  - $\mathbf{W}_i^V \in \mathbb{R}^{d \times d_v}$
- Each member of such a set of self-attention layers is called a **head**
- Each head gets multiplied by the inputs packed into $\mathbf{X}$ to produce
  - $\mathbf{W}_i^Q \in \mathbb{R}^{N \times d_k}$
  - $\mathbf{W}_i^K \in \mathbb{R}^{N \times d_k}$
  - $\mathbf{W}_i^V \in \mathbb{R}^{N \times d_v}$

Multihead Attention Layers

- The output of a multi-head layer with h heads consists of h vectors of shape $N \times d_v$
- These outputs are concatenated from each head and then, using a linear projection with weight matrix $\mathbf{W}_i^O \in \mathbb{R}^{hd_k \times d}$, reduced to $N \times d$ output.

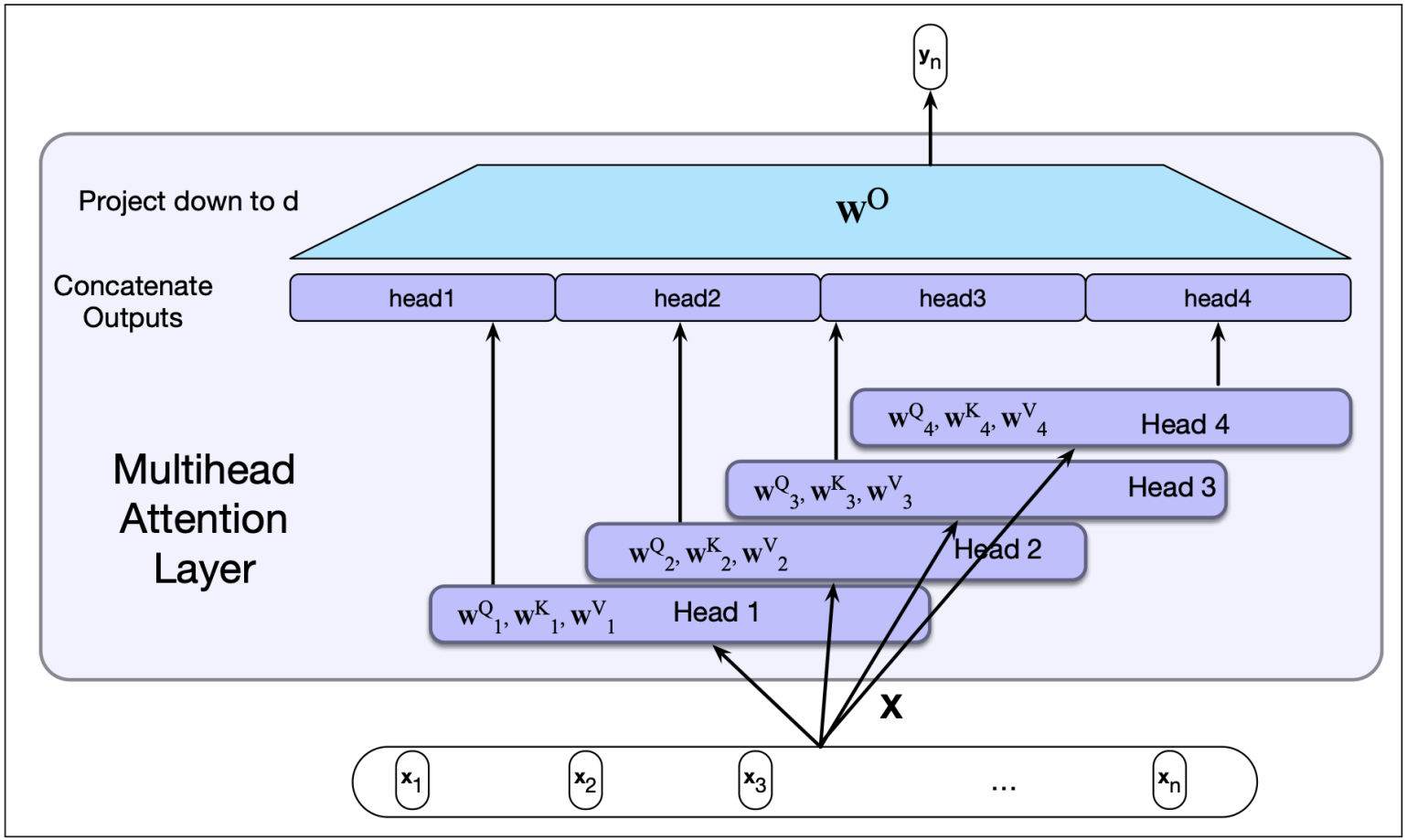
Multihead Attention

$$\mathit{MultiHeadAttn}(\mathbf{X}) = (\mathbf{head}_1 \oplus \mathbf{head}_2 \cdots \oplus \mathbf{head}_h)\mathbf{W}^O \tag{18}$$

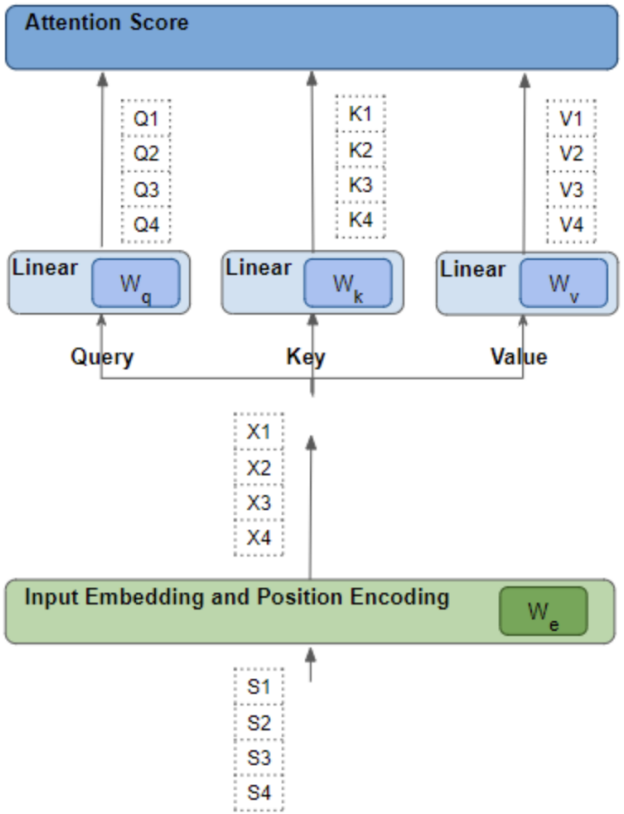
$$\mathbf{Q} = \mathbf{X}\mathbf{W}_i^Q;\ \mathbf{K} = \mathbf{X}\mathbf{W}_i^K;\ \mathbf{V} = \mathbf{X}\mathbf{W}_i^V \tag{19}$$

$$\mathbf{head}_i = \mathit{SelfAttention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) \tag{20}$$

Multihead Attention

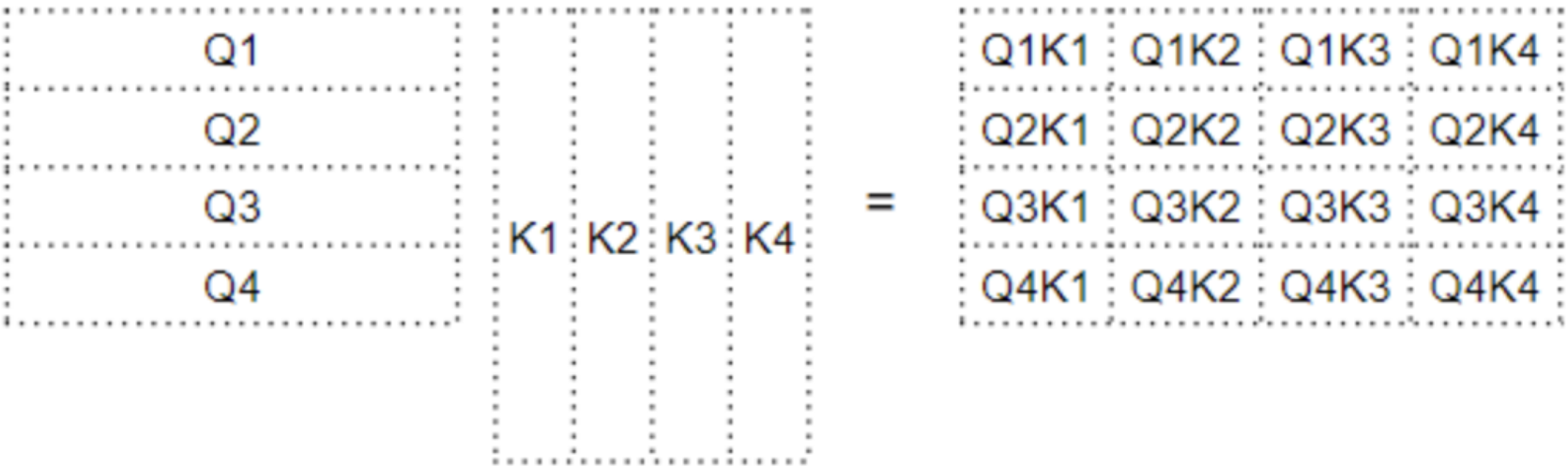


Multihead Attention: Visual Summary



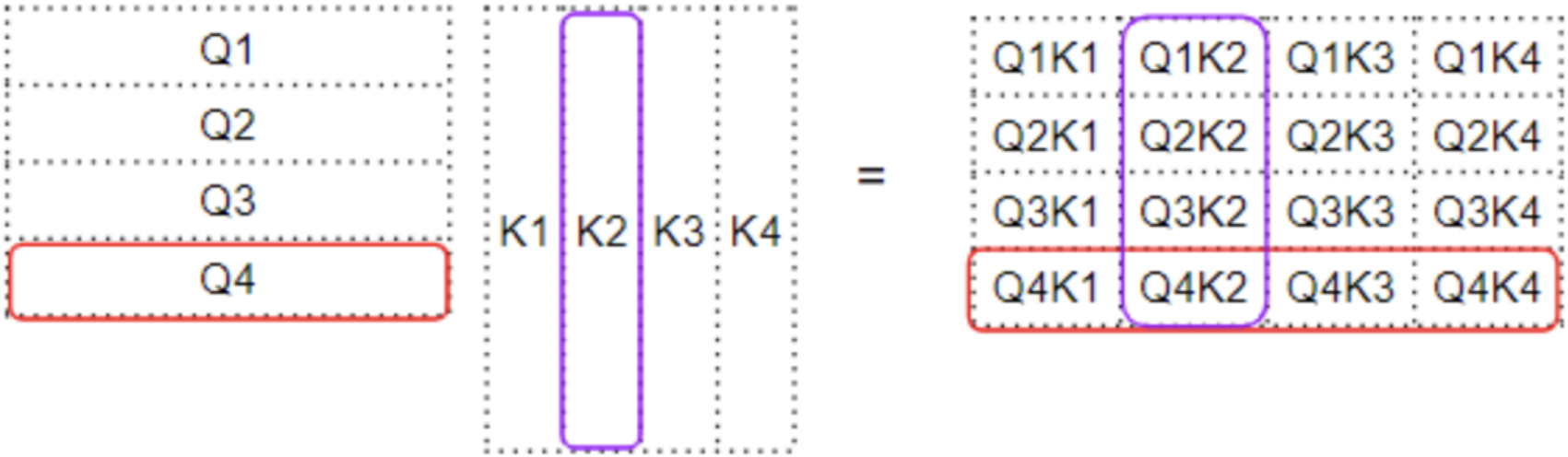
Slide due to Keitan Doshi, https://towardsdatascience.com/transformers-explained-visually-not-just-how-but-why-they-work-so-well-d840bd61a9d3

Multihead Attention: Dot Product between Query and Key matrices



Slide due to Keitan Doshi
https://towardsdatascience.com/transformers-explained-visually-not-just-how-but-why-they-work-so-well-d840bd61a9d3

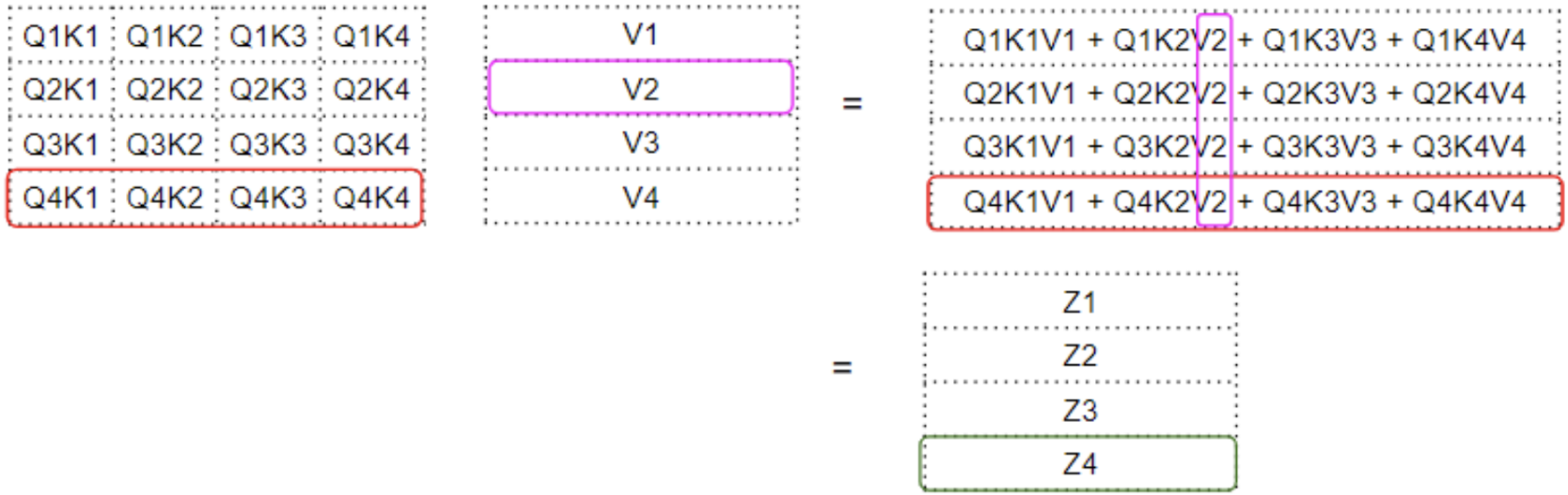
Multihead Attention: Dot Product between Query and Key matrices



Slide due to Keitan Doshi

`https://towardsdatascience.com/transformers-explained-visually-not-just-how-but-why-they-work-so-well-d840bd61a9d3`

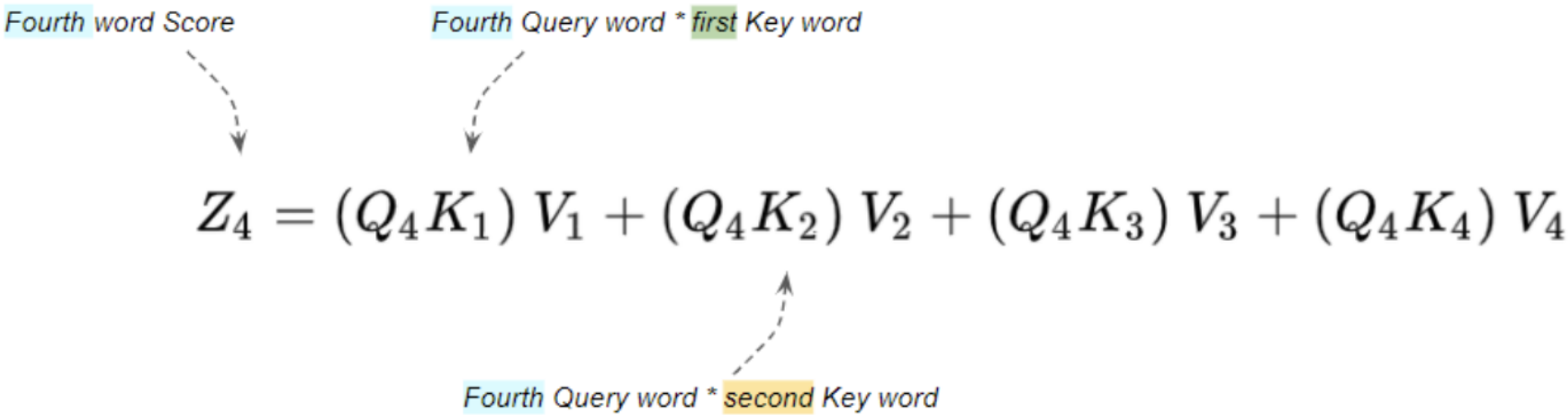
Dot Product between Query-Key and Value Matrices



Slide due to Keitan Doshi

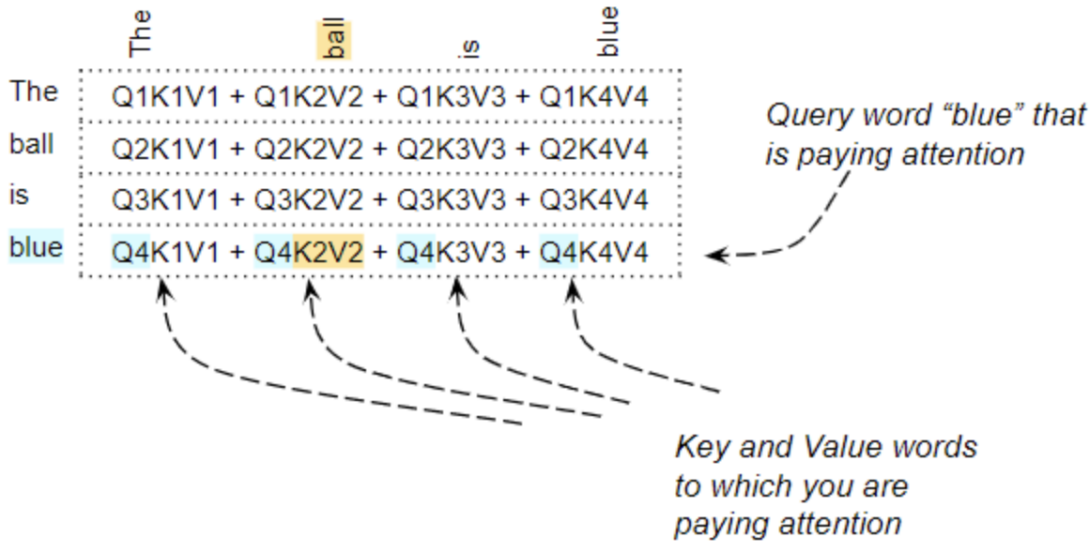
`https://towardsdatascience.com/transformers-explained-visually-not-just-how-but-why-they-work-so-well-d840bd61a9d3`

Attention Score for the word blue pays attention to every other word



Slide due to Keitan Doshi
`https://towardsdatascience.com/transformers-explained-visually-not-just-how-but-why-they-work-so-well-d840bd61a9d3`

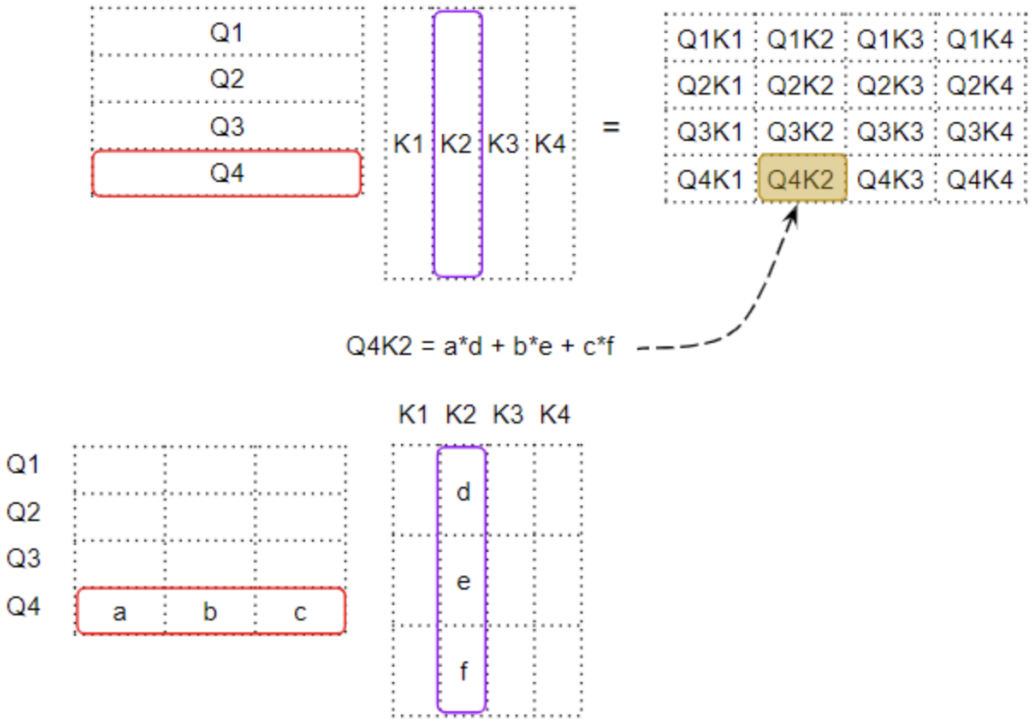
Dot Product between Query-Key and Value Matrices



Slide due to Keitan Doshi

`https://towardsdatascience.com/transformers-explained-visually-not-just-how-but-why-they-work-so-well-d840bd61a9d3`

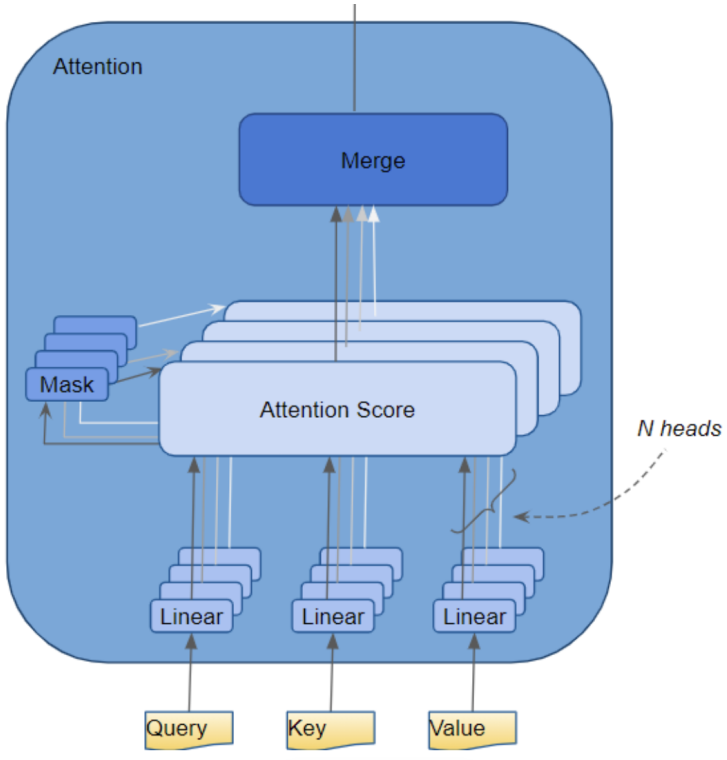
Each cell is a dot product between two word vectors



Slide due to Keitan Doshi

`https://towardsdatascience.com/transformers-explained-visually-not-just-how-but-`

Multihead Attention: Visual Summary

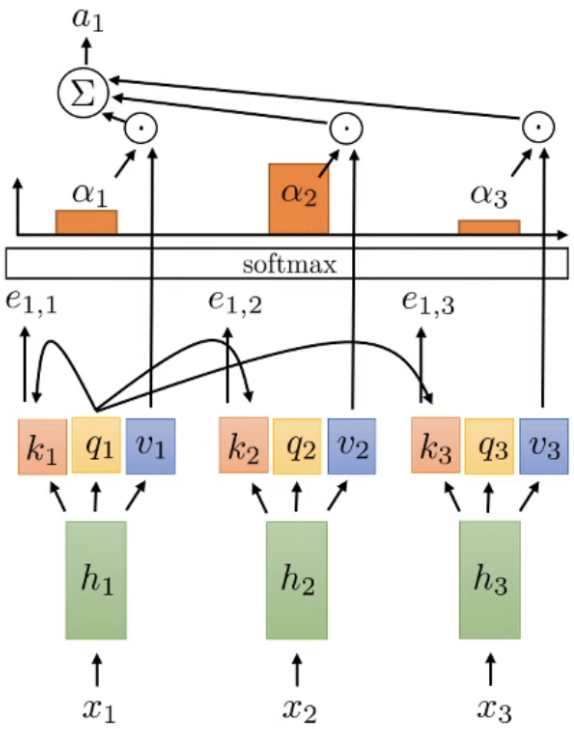


Slide due to Keitan Doshi
https://towardsdatascience.com/transformers-explained-visually-not-just-how-but-why-they-work-so-well-d840bd61a9d3

Modeling word order: positional embedding

- Transformers models do not come with built-in information about sequence order such as time-step information in an RNN
- Transformers models do not have any notion of relative or absolute positions of the tokens in a sequence.
- In order to model word order, positional embeddings can be combined with input embeddings.
- Positional embeddings that are specific to each position in an input sequence.

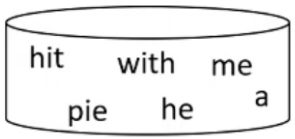
Positional encoding: what is the order?



what we see:

he hit me with a pie

what naïve self-attention sees:



a pie hit me with he

a hit with me he pie

he pie me with a hit

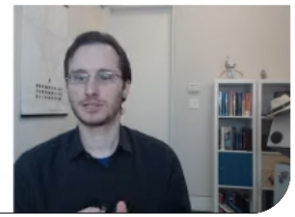
most alternative orderings are nonsense, but some change the meaning

in general the position of words in a sentence carries information!

Idea: add some information to the representation at the beginning that indicates where it is in the sequence!

$$h_t = f(x_t, t)$$

some function



Slide due to RAIL CS182: Lecture 12:Part 2: Transformers
https://www.youtube.com/watch?v=4AzsiCMw_-s

Positional encoding: sin/cos

Naïve positional encoding: just append t to the input $\quad \bar{x}_t = \begin{bmatrix} x_t \\ t \end{bmatrix}$

This is not a great idea, because **absolute** position is less important than **relative** position

I walk my dog every day

every single day I walk my dog

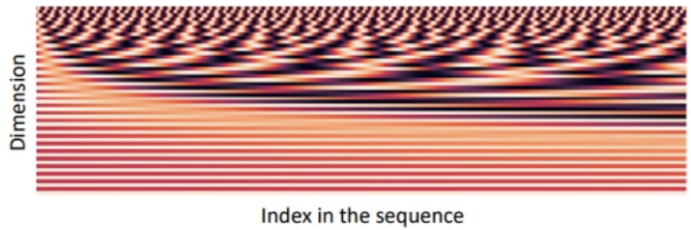
The fact that "my dog" is right after "I walk" is the important part, not its absolute position

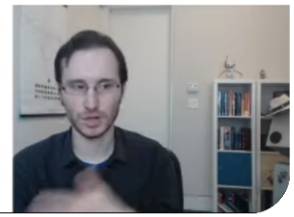
we want to represent **position** in a way that tokens with similar **relative** position have similar **positional encoding**

Idea: what if we use **frequency-based** representations?

$$p_t = \begin{bmatrix} \sin(t/10000^{2*1/d}) \\ \cos(t/10000^{2*1/d}) \\ \sin(t/10000^{2*2/d}) \\ \cos(t/10000^{2*2/d}) \\ \dots \\ \sin(t/10000^{2*\frac{d}{2}/d}) \\ \cos(t/10000^{2*\frac{d}{2}/d}) \end{bmatrix}$$

dimensionality of positional encoding

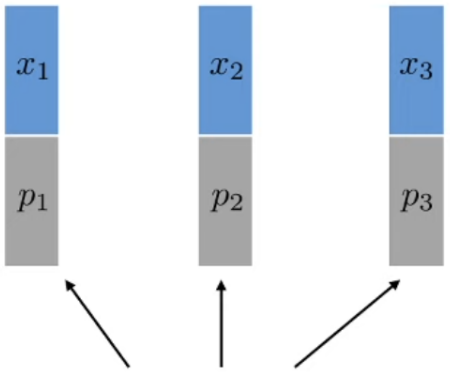




Slide due to RAIL CS182: Lecture 12:Part 2: Transformers
https://www.youtube.com/watch?v=4AzsiCMw_-s

Positional encoding: learned

Another idea: just learn a positional encoding



Different for every input sequence

The same learned values for every sequence

but different for different time steps

dimensionality

max sequence length

How many values do we need to learn?

$$P = [p_1, p_2, \ldots, p_T] \in R^{d \times T}$$

+ more flexible (and perhaps more optimal) than sin/cos encoding

+ a bit more complex, need to pick a max sequence length (and can't generalize beyond it)

Slide due to RAIL CS182: Lecture 12:Part 2: Transformers
`https://www.youtube.com/watch?v=4AzsiCMw_-s`

How to incorporate positional encoding?

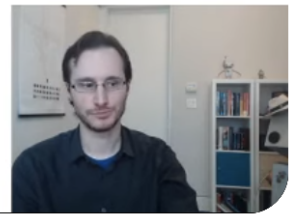
At each step, we have x_t and p_t

Simple choice: just concatenate them $\bar{x}_t = \begin{bmatrix} x_t \\ p_t \end{bmatrix}$

More often: just add after **embedding** the input

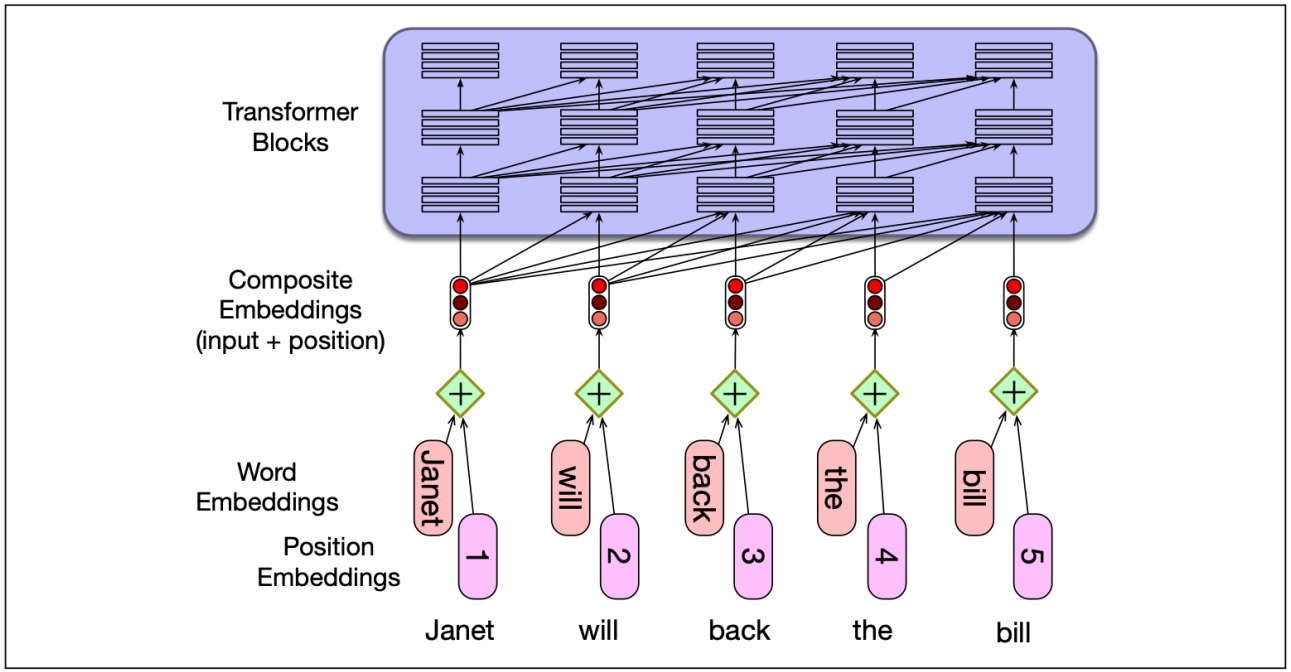
input to self-attention is $\text{emb}(x_t) + p_t$

some learned function (e.g., some fully connected layers with linear layers + nonlinearities)

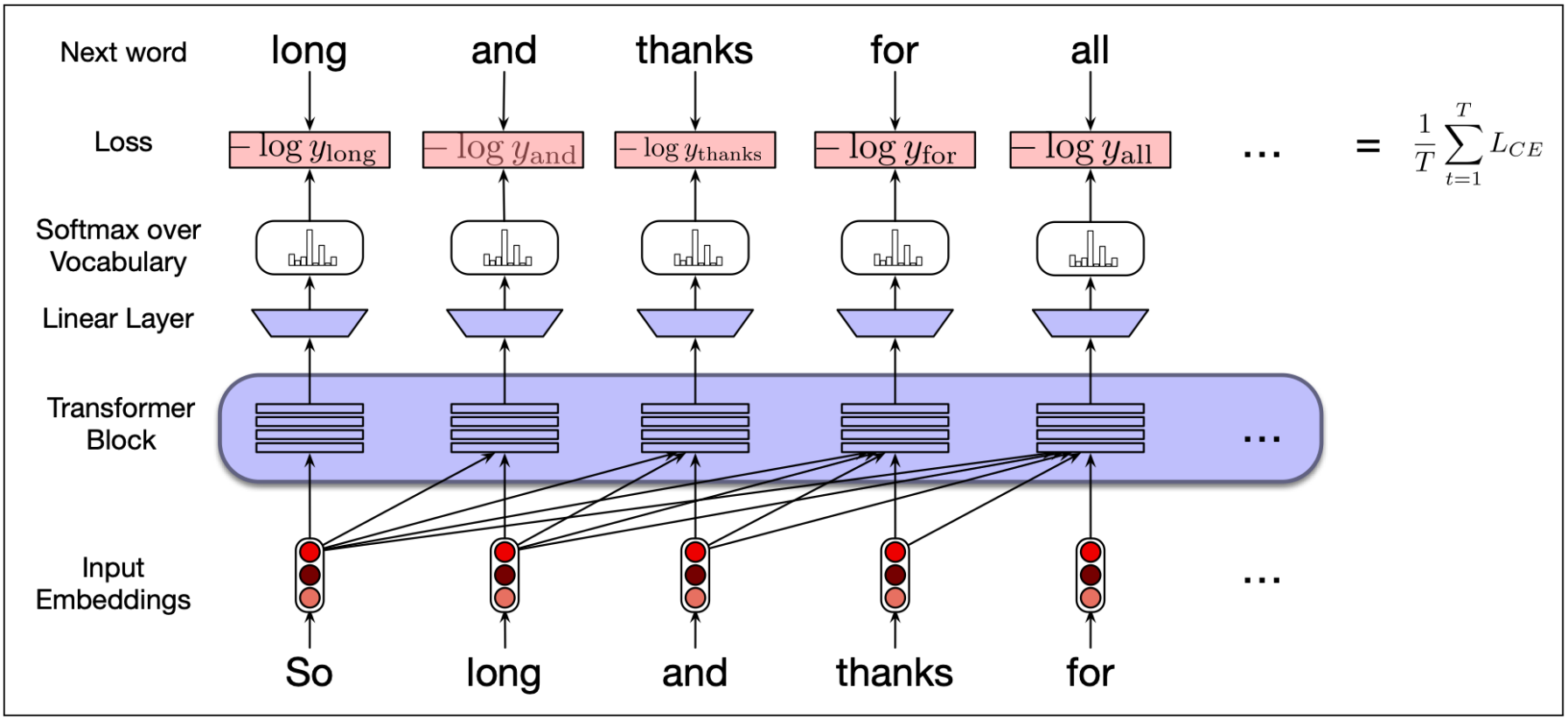


Slide due to RAIL CS182: Lecture 12:Part 2: Transformers
https://www.youtube.com/watch?v=4AzsiCMw_-s

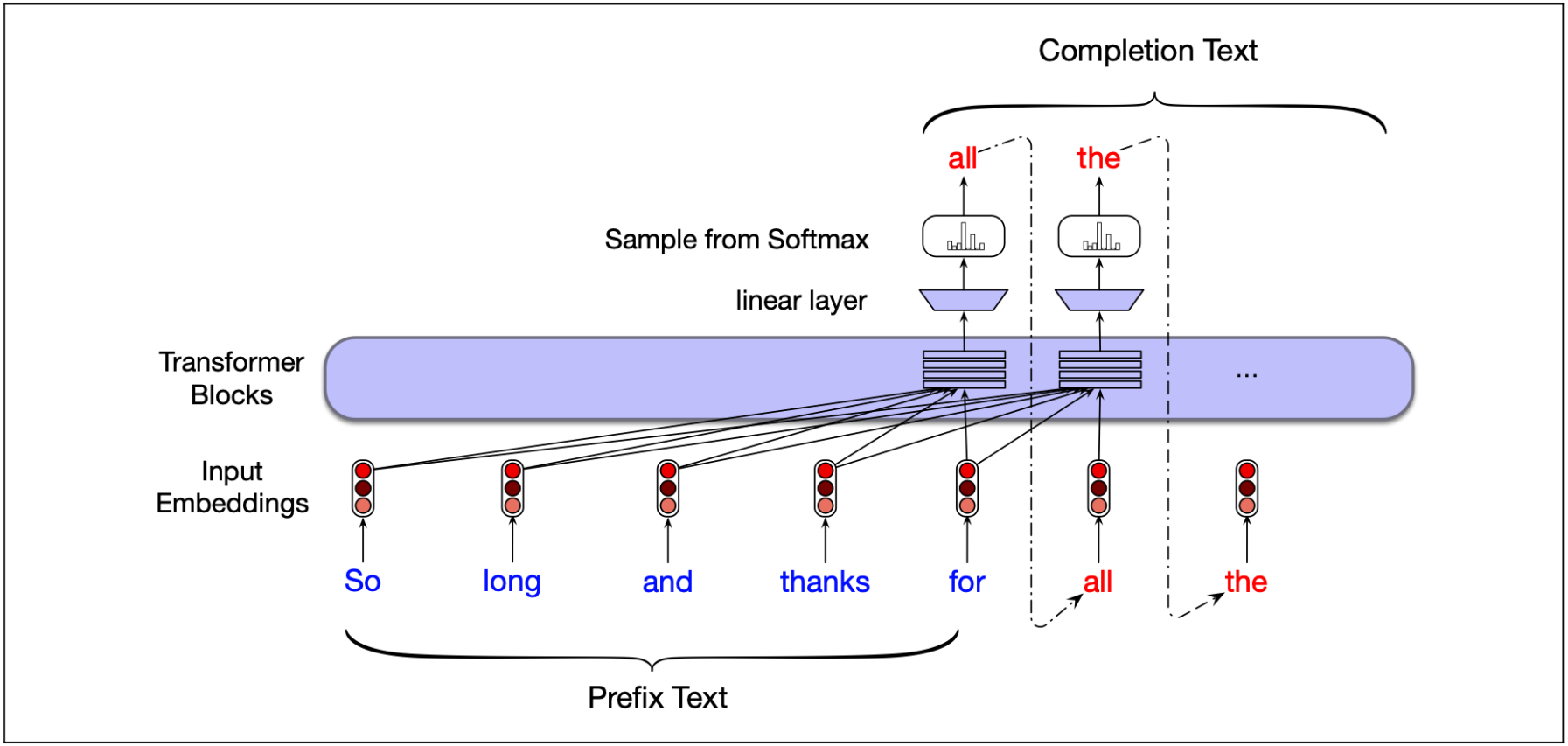
Multihead Attention



Transformers as Language Models



Contextual Generation and Summarization



Contextual Generation and Summarization

Original Article

The only thing crazier than a guy in snowbound Massachusetts boxing up the powdery white stuff and offering it for sale online? People are actually buying it. For $89, self-styled entrepreneur Kyle Waring will ship you 6 pounds of Boston-area snow in an insulated Styrofoam box – enough for 10 to 15 snowballs, he says. But not if you live in New England or surrounding states. “We will not ship snow to any states in the northeast!” says Waring’s website, ShipSnowYo.com. “We’re in the business of expunging snow!”

Contextual Generation and Summarization

Continued

His website and social media accounts claim to have filled more than 133 orders for snow – more than 30 on Tuesday alone, his busiest day yet. With more than 45 total inches, Boston has set a record this winter for the snowiest month in its history. Most residents see the huge piles of snow choking their yards and sidewalks as a nuisance, but Waring saw an opportunity.

Contextual Generation and Summarization

Continued

According to Boston.com, it all started a few weeks ago, when Waring and his wife were shoveling deep snow from their yard in Manchester-by-the-Sea, a coastal suburb north of Boston. He joked about shipping the stuff to friends and family in warmer states, and an idea was born. His business slogan: "Our nightmare is your dream!" At first, ShipSnowYo sold snow packed into empty 16.9-ounce water bottles for $19.99, but the snow usually melted before it reached its destination...

Contextual Generation and Summarization

Summary

Kyle Waring will ship you 6 pounds of Boston-area snow in an insulated Styrofoam box – enough for 10 to 15 snowballs, he says. But not if you live in New England or surrounding states.

Contextual Generation and Summarization