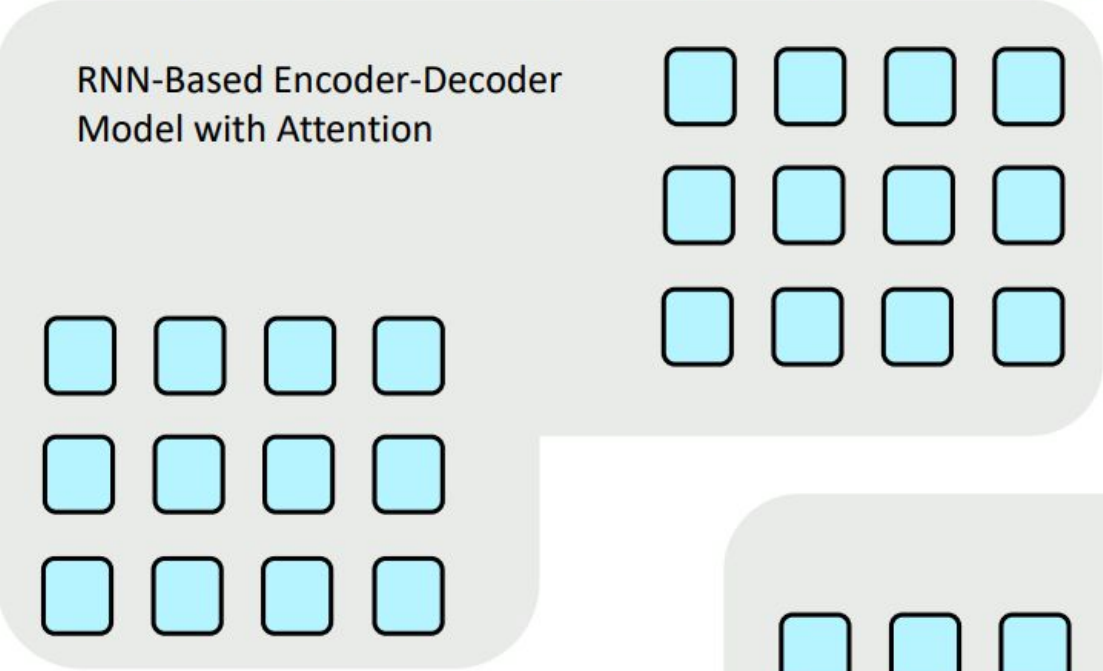
This is CS4045!

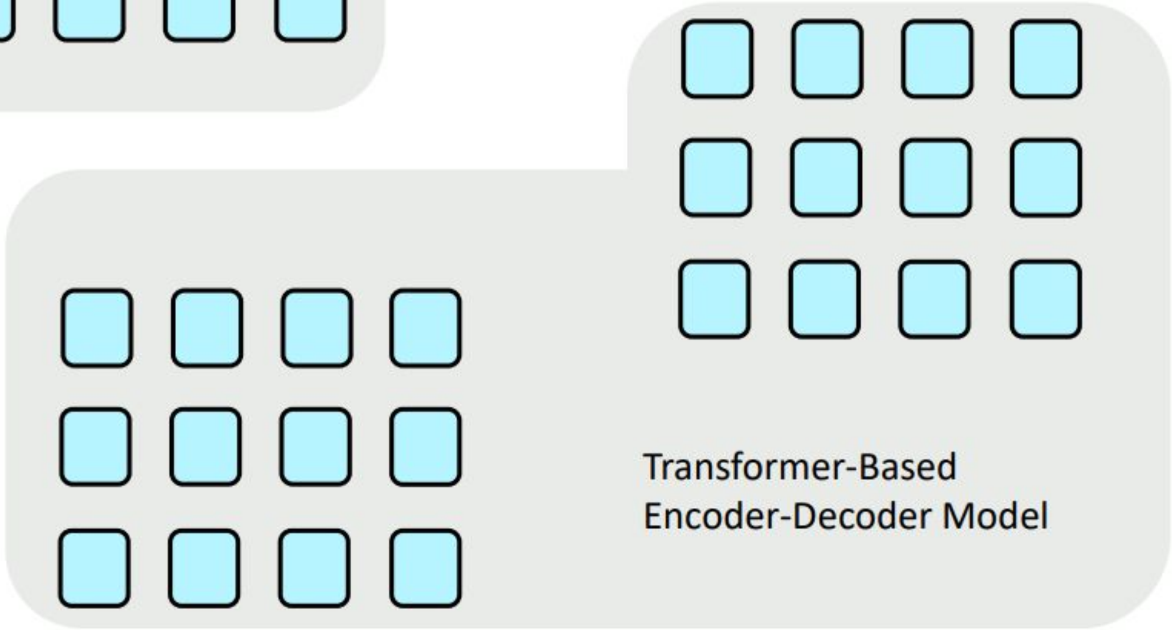
GCR : dxuxugo

Computational Dependencies for Recurrence vs. Attention



Transformer Advantages:

- Number of unparallelizable operations does not increase with sequence length.
- Each "word" interacts with each other, so maximum interaction distance: O(1).



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

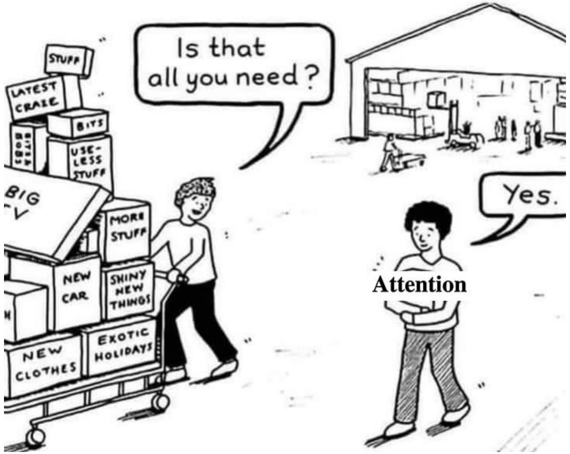
Łukasz Kaiser* Google Brain lukaszkaiser@google.com

Illia Polosukhin* ‡ illia.polosukhin@gmail.com

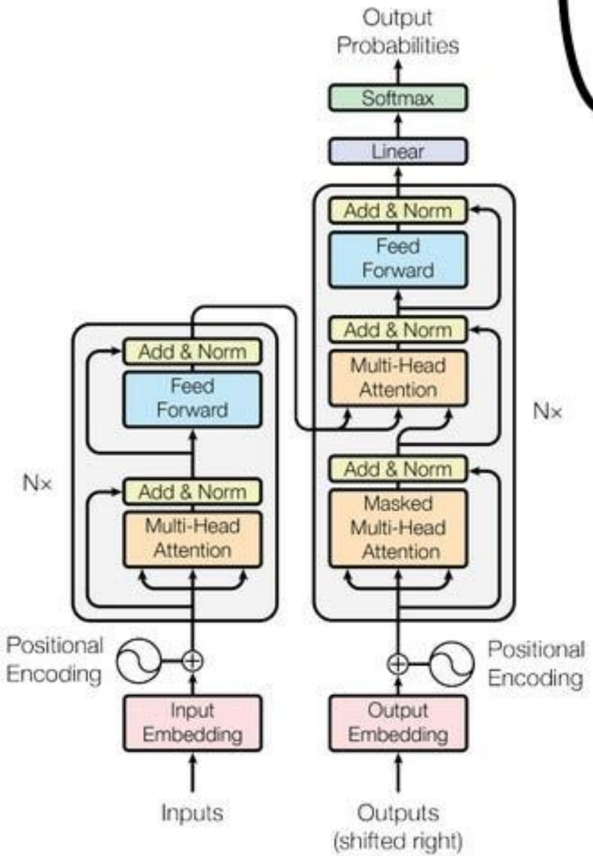

ATTENTION IS ALL YOU NEED?
FALSE. YOU NEED WATER AND RATIONS.
imgflip.com

Attention

Vaswani, Ashish, et al. "Attention is all you need." Advances in neural information processing systems. 2017.



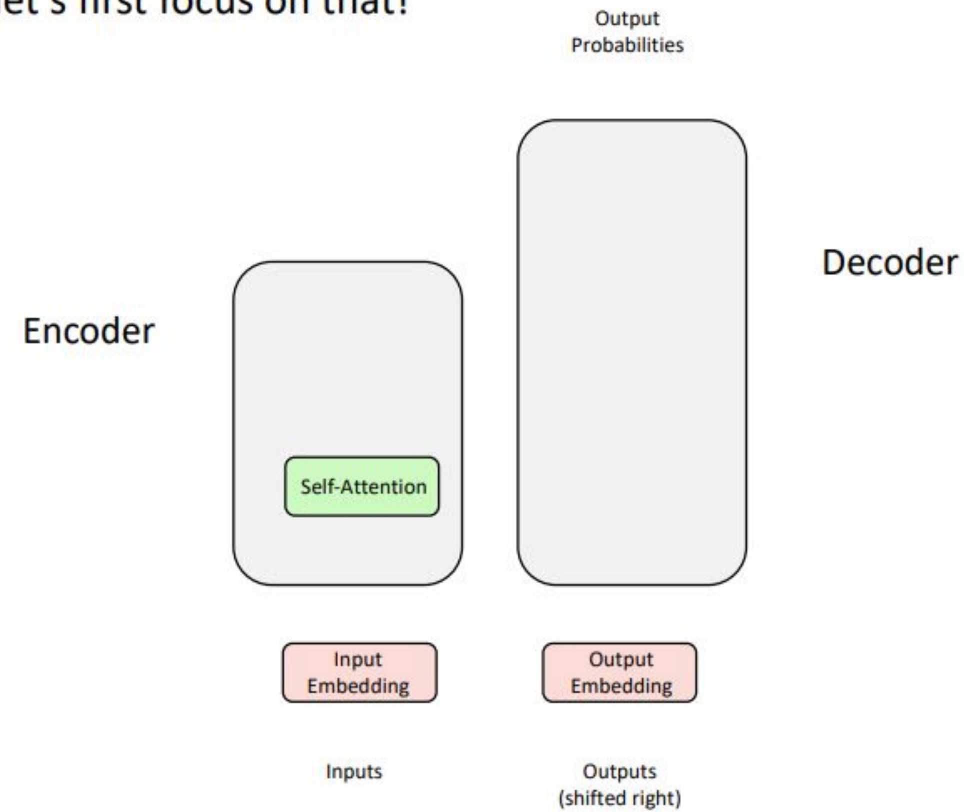
- Sequential models ingest the input one word or one token at the time. And so, as as if each unit was like a bottleneck to the flow of information.
- Transformer ingest an entire sentence all at the same time.
- Attention mechanism + CNN like parallelism.


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


COME, LET US UNDERSTAND THE UNCOOL TRANSFORMERS!!!
- OPTIMUS PRIME

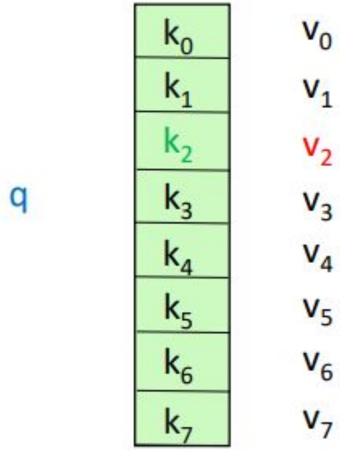
Encoder: Self-Attention

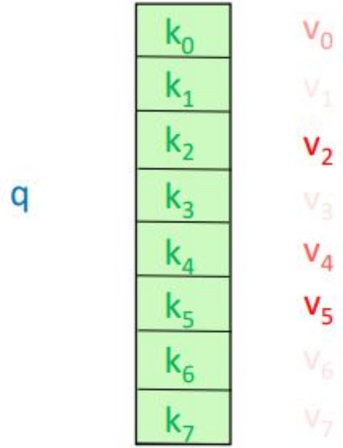
Self-Attention is the core building block of Transformer, so let's first focus on that!



Intuition for Attention Mechanism

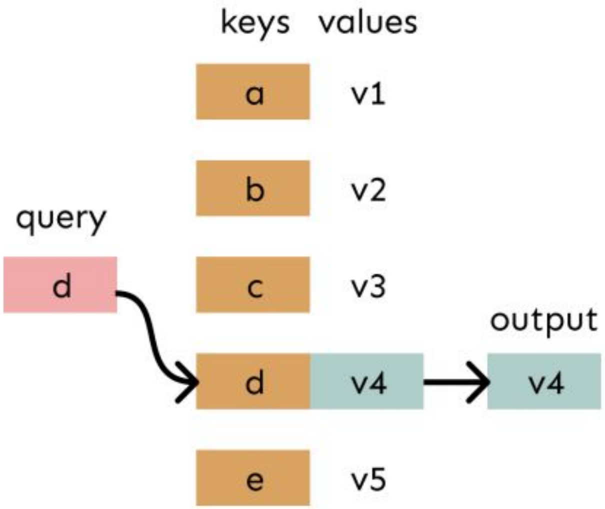
- Let's think of attention as a "fuzzy" or approximate hashtable:
  - To look up a **value**, we compare a **query** against **keys** in a table.
  - In a hashtable (shown on the bottom left):
    - Each **query** (hash) maps to exactly one **key**-**value** pair.
  - In (self-)attention (shown on the bottom right):
    - Each **query** matches each **key** to varying degrees.
    - We return a sum of **values** weighted by the **query**-**key** match.

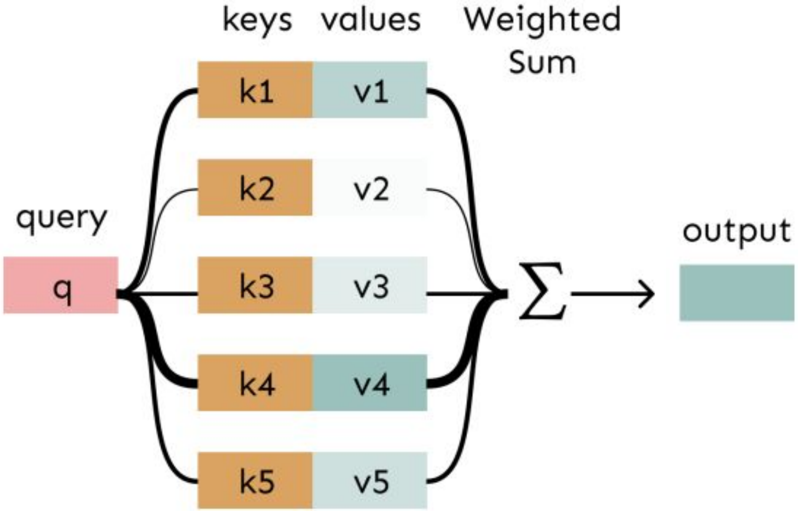




In a **lookup table**, we have a table of **keys** that map to **values**. The **query** matches one of the keys, returning its value.

In **attention**, the **query** matches all **keys** *softly*, to a weight between 0 and 1. The keys' **values** are multiplied by the weights and summed.





Recipe for Self-Attention in the Transformer Encoder

- Step 1: For each word x_i , calculate its **query**, **key**, and **value**.

$$q_i = W^Q x_i \quad k_i = W^K x_i \quad v_i = W^V x_i$$

- Step 2: Calculate attention score between **query and keys.**

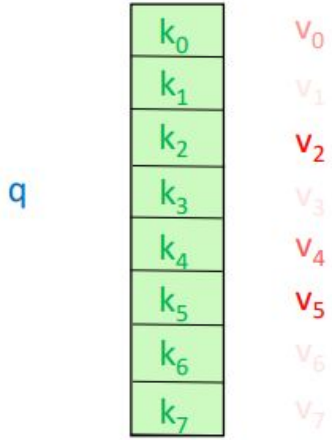
$$e_{ij} = q_i \cdot k_j$$

- Step 3: Take the softmax to normalize attention scores.

$$\alpha_{ij} = softmax(e_{ij}) = \frac{exp(e_{ij})}{\sum_k exp(e_{ik})}$$

- Step 4: Take a weighted sum of **values**.

$$Output_i = \sum_j \alpha_{ij} v_j$$



Recipe for (Vectorized) Self-Attention in the Transformer Encoder

- Step 1: With embeddings stacked in X, calculate **queries**, **keys**, and **values**.

$$Q = XW^Q \quad K = XW^K \quad V = XW^V$$

- Step 2: Calculate attention scores between **query and keys.**

$$E = QK^T$$

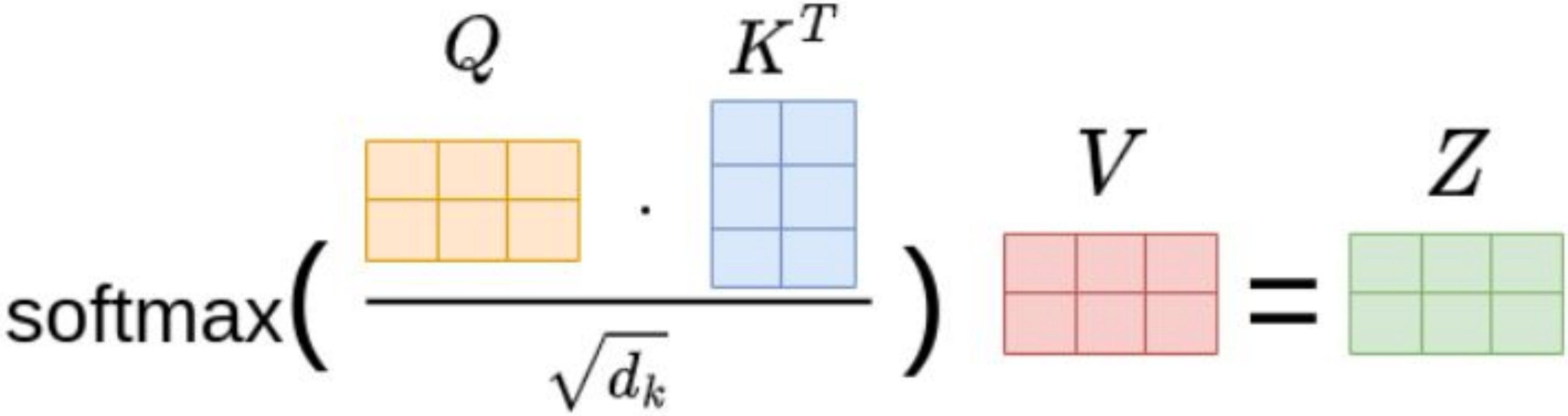
- Step 3: Take the softmax to normalize attention scores.

$$A = softmax(E)$$

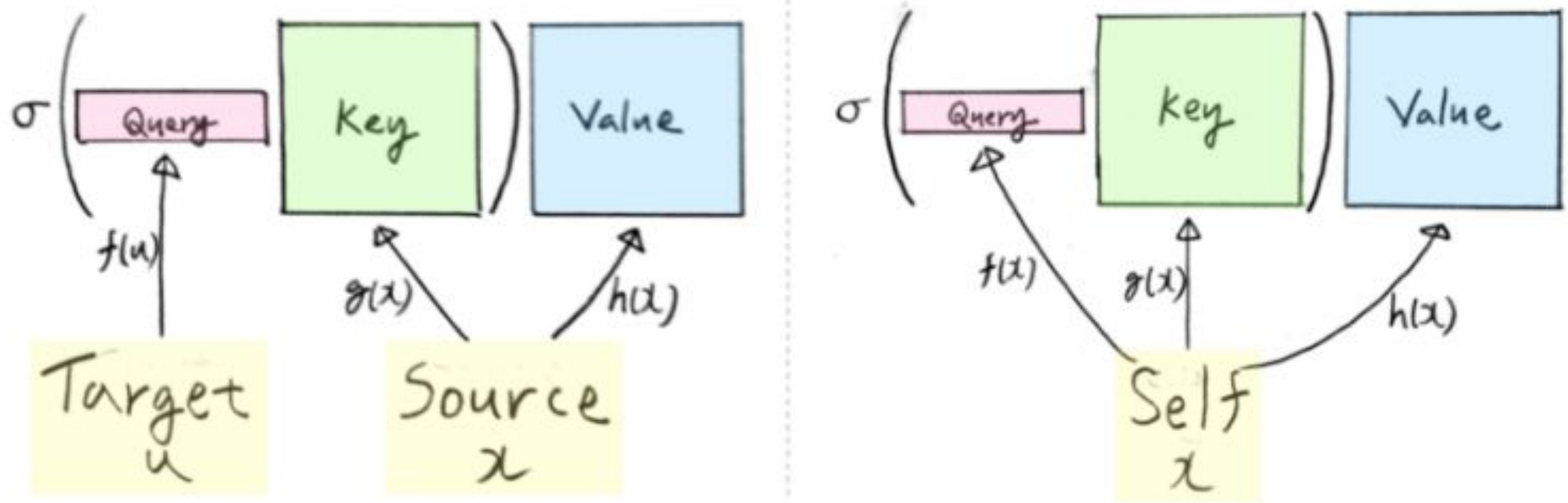
- Step 4: Take a weighted sum of **values**.

$$Output = AV$$

$$\boxed{Output = softmax\left(QK^T\right)V}$$

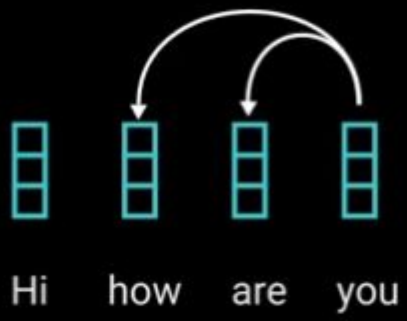

softmax
Q
K^T
$\sqrt{d_k}$
V
Z

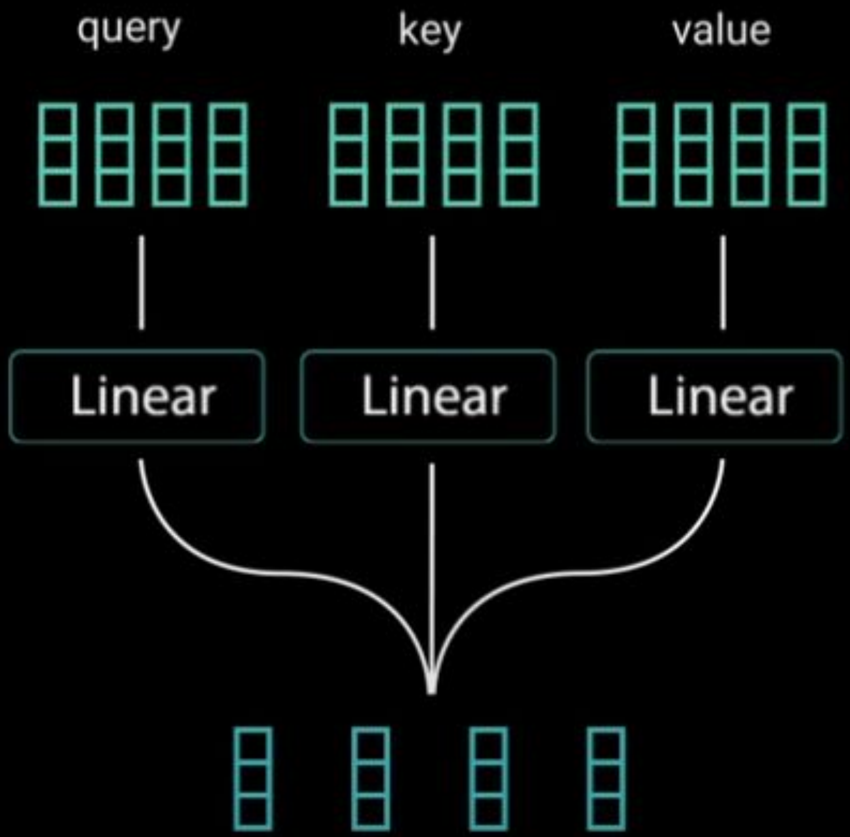
(Source-Target-Attention) (Self-Attention)



3. Multi-headed Attention

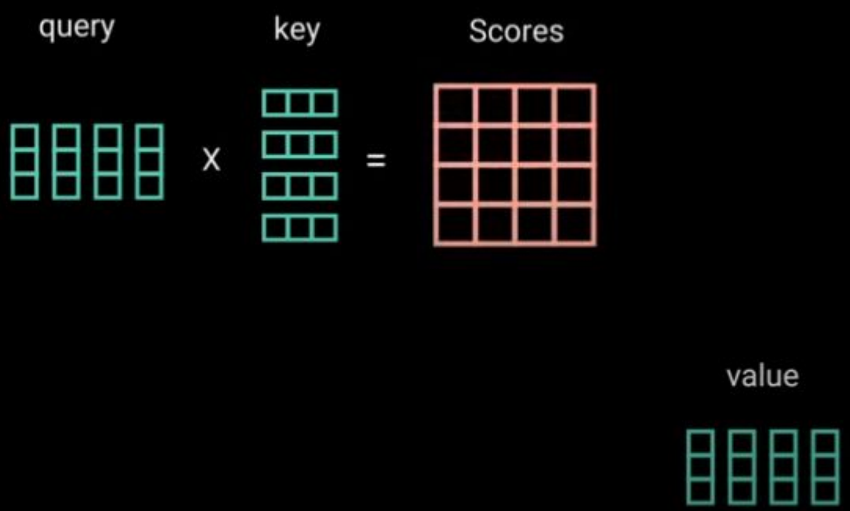
3.1. Self-Attention





3. Multi-headed Attention

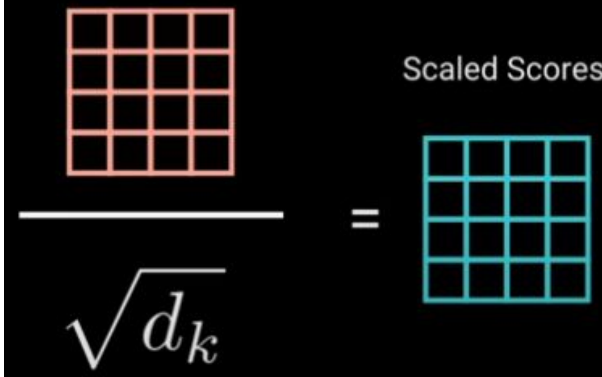
3.1. Self-Attention



	Hi	how	are	you
Hi	98	27	10	12
how	27	89	31	67
are	10	31	91	54
you	12	67	54	92

3. Multi-headed Attention

3.1. Self-Attention



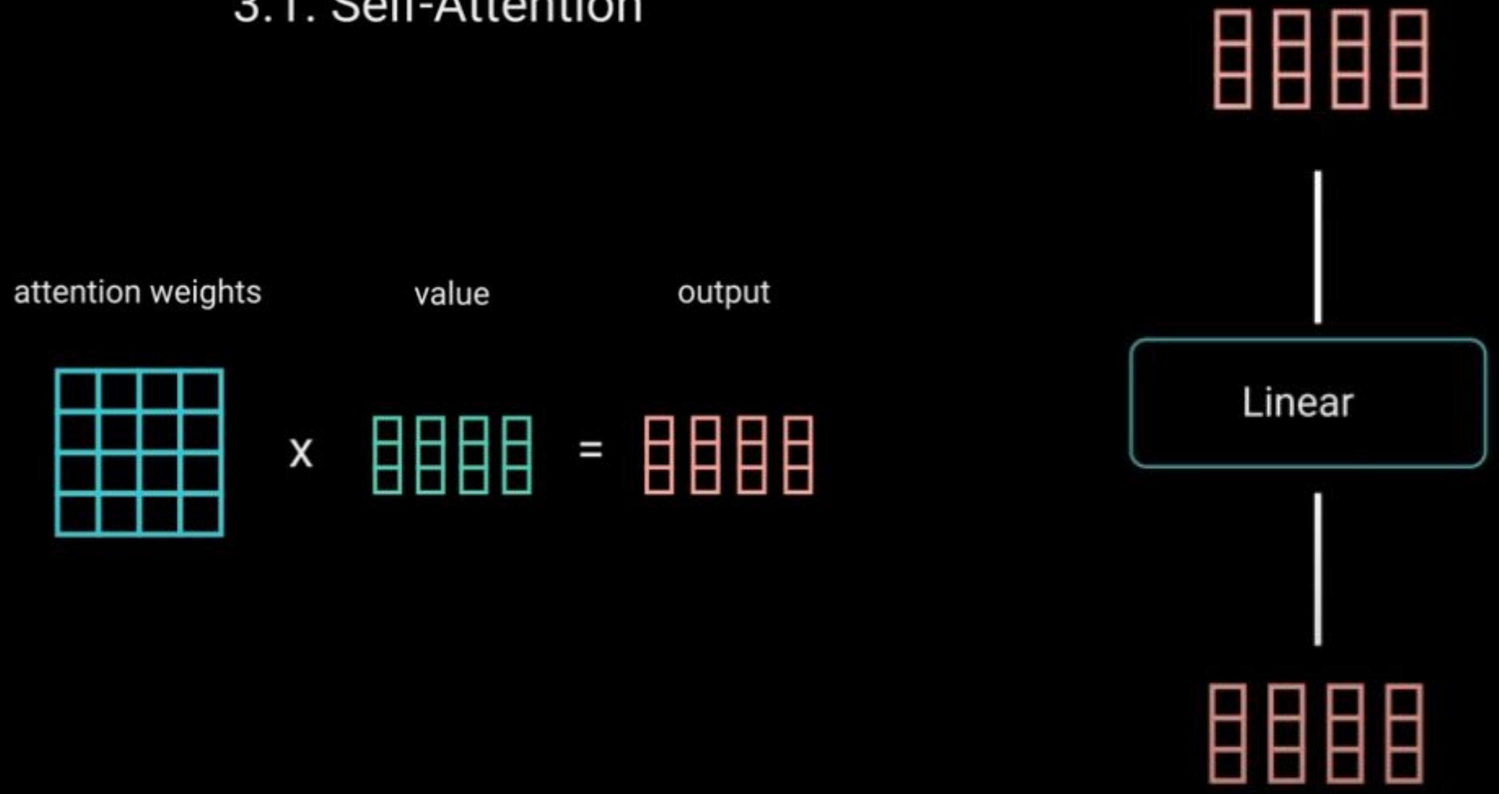
Softmax() =

	Hi	how	are	you
Hi	0.7	0.1	0.1	0.1
how	0.1	0.6	0.2	0.1
are	0.1	0.3	0.6	0.1
you	0.1	0.3	0.3	0.3

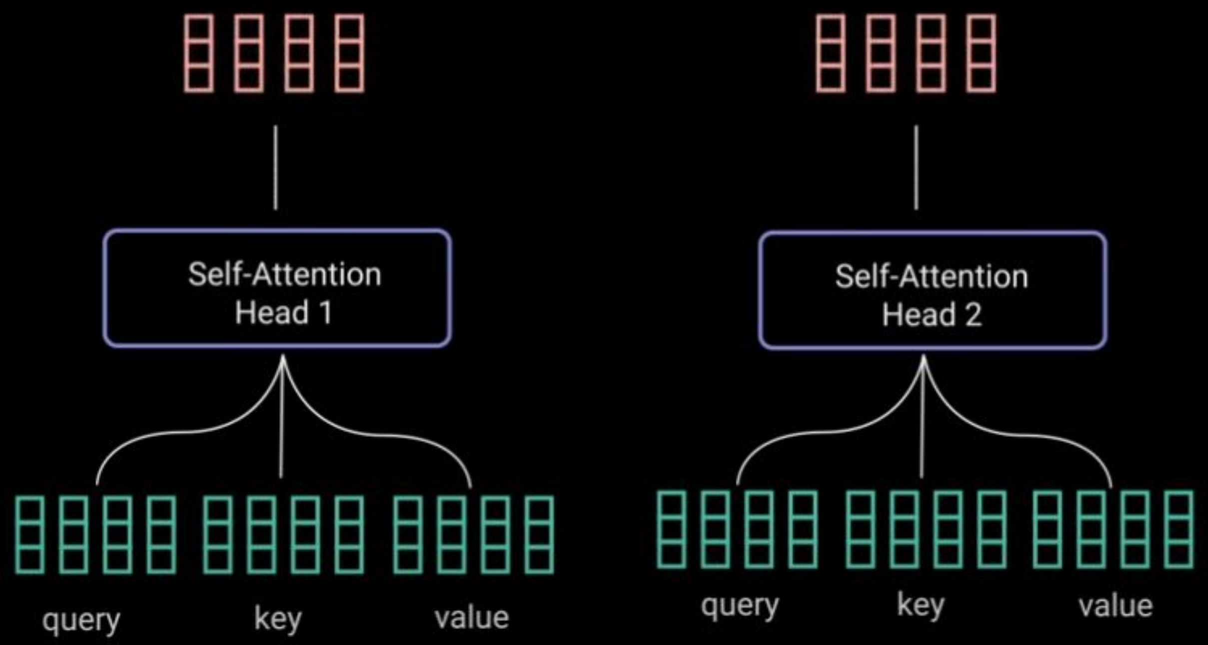
$$softmax(x)_i = \frac{exp(x_i)}{\sum_j exp(x_j))}$$

3. Multi-headed Attention

3.1. Self-Attention

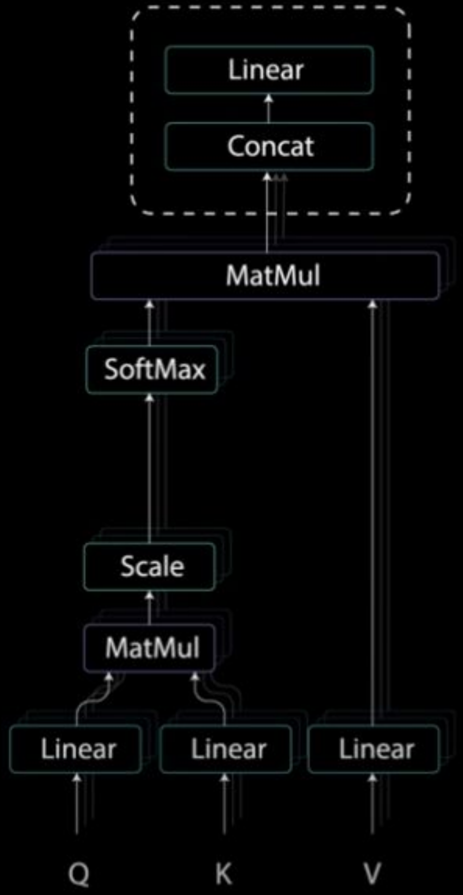


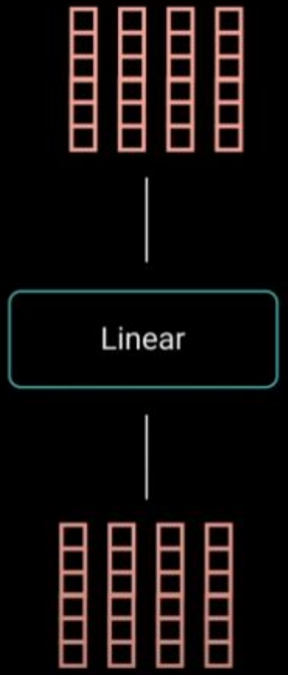
3. Multi-headed Attention



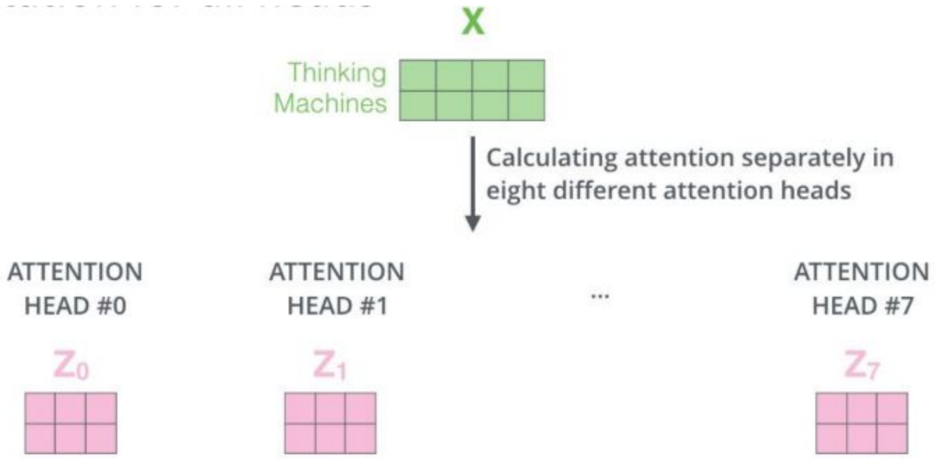
N = 2

3. Multi-headed Attention

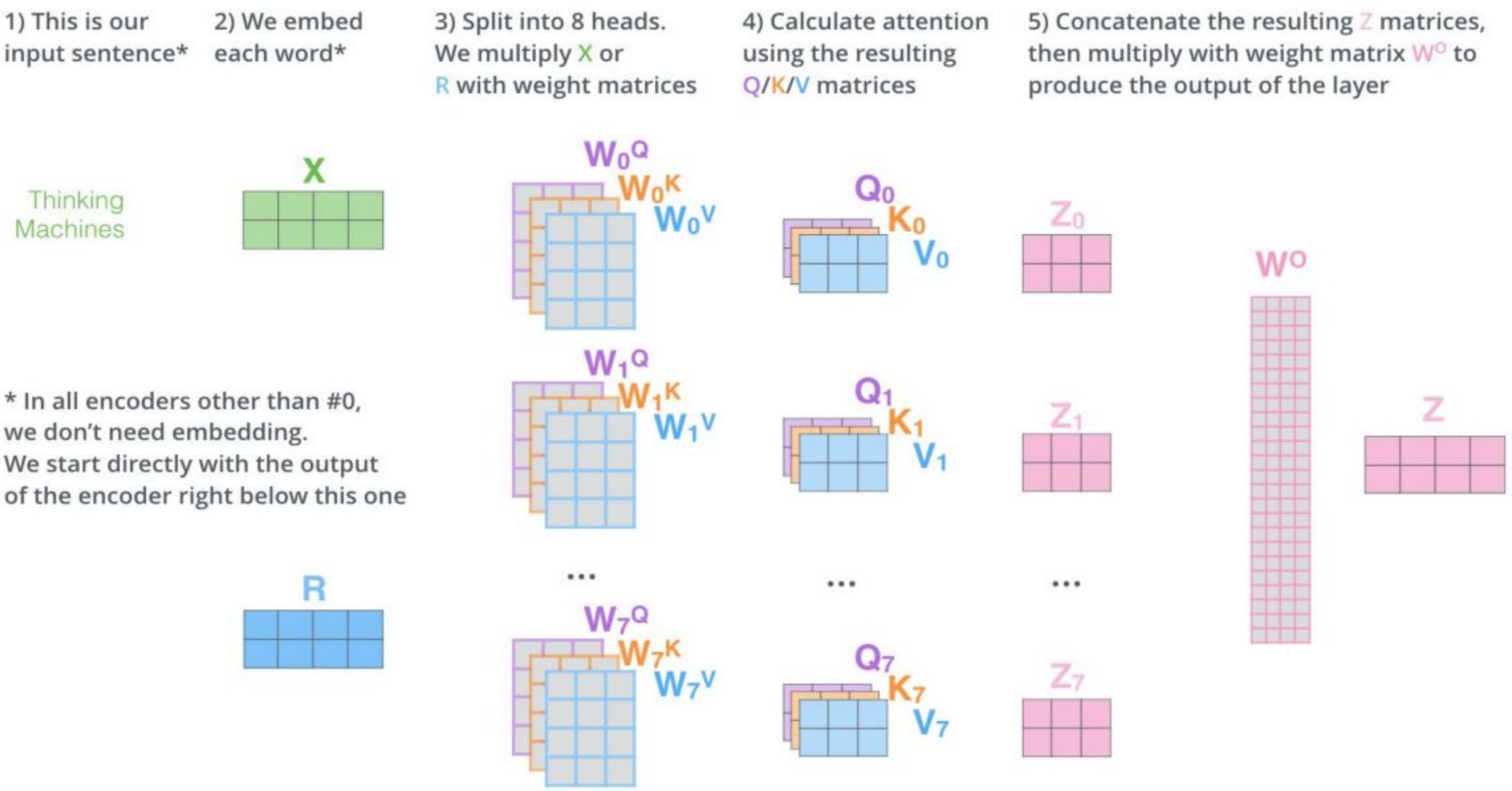




Multi-Head Attention



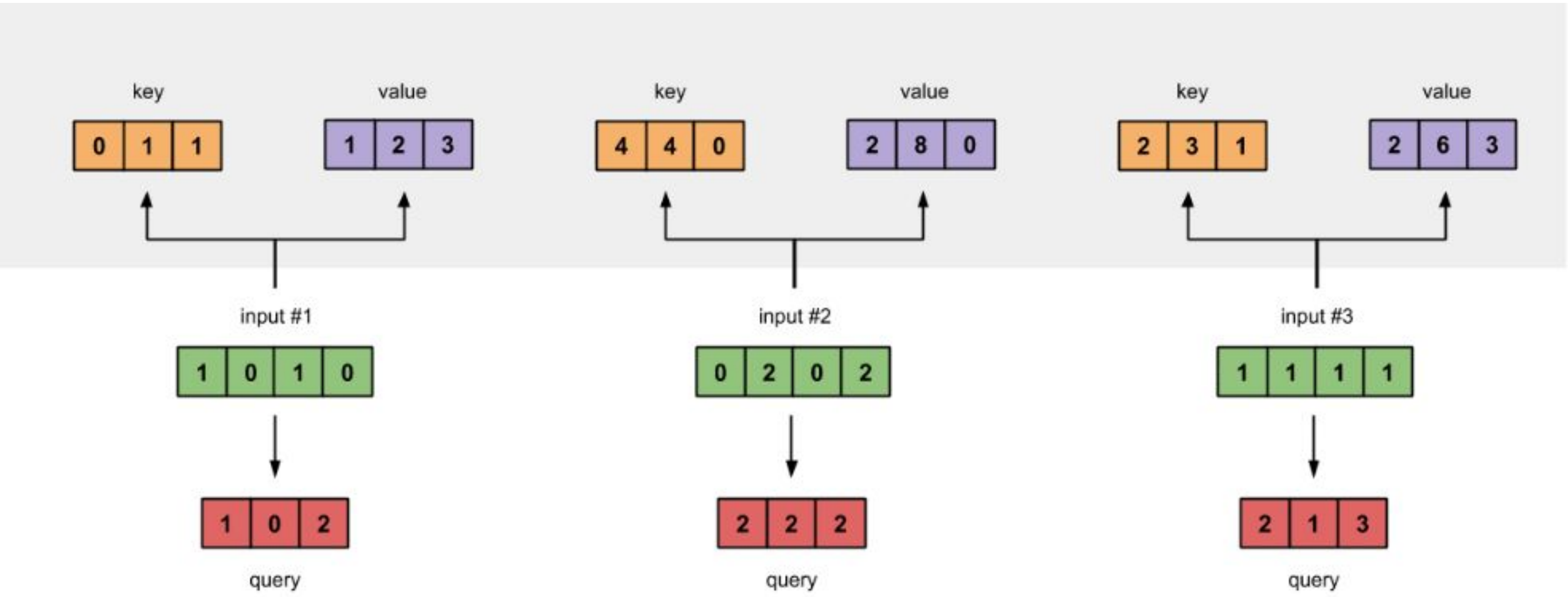
- Doing self-Attention multiple times (Multi-Head).
- As if you are asking a different query about the input multiple times.
- Parallel computation for all heads


1) This is our input sentence*
2) We embed each word*
3) Split into 8 heads. We multiply X or R with weight matrices
4) Calculate attention using the resulting Q/K/V matrices
5) Concatenate the resulting Z matrices, then multiply with weight matrix W^O to produce the output of the layer
Thinking Machines
X
* In all encoders other than #0, we don't need embedding. We start directly with the output of the encoder right below this one
R
W_0^Q
W_0^K
W_0^V
W_1^Q
W_1^K
W_1^V
...
W_7^Q
W_7^K
W_7^V
Q_0
K_0
V_0
Q_1
K_1
V_1
...
Q_7
K_7
V_7
Z_0
Z_1
...
Z_7
W^O
Z

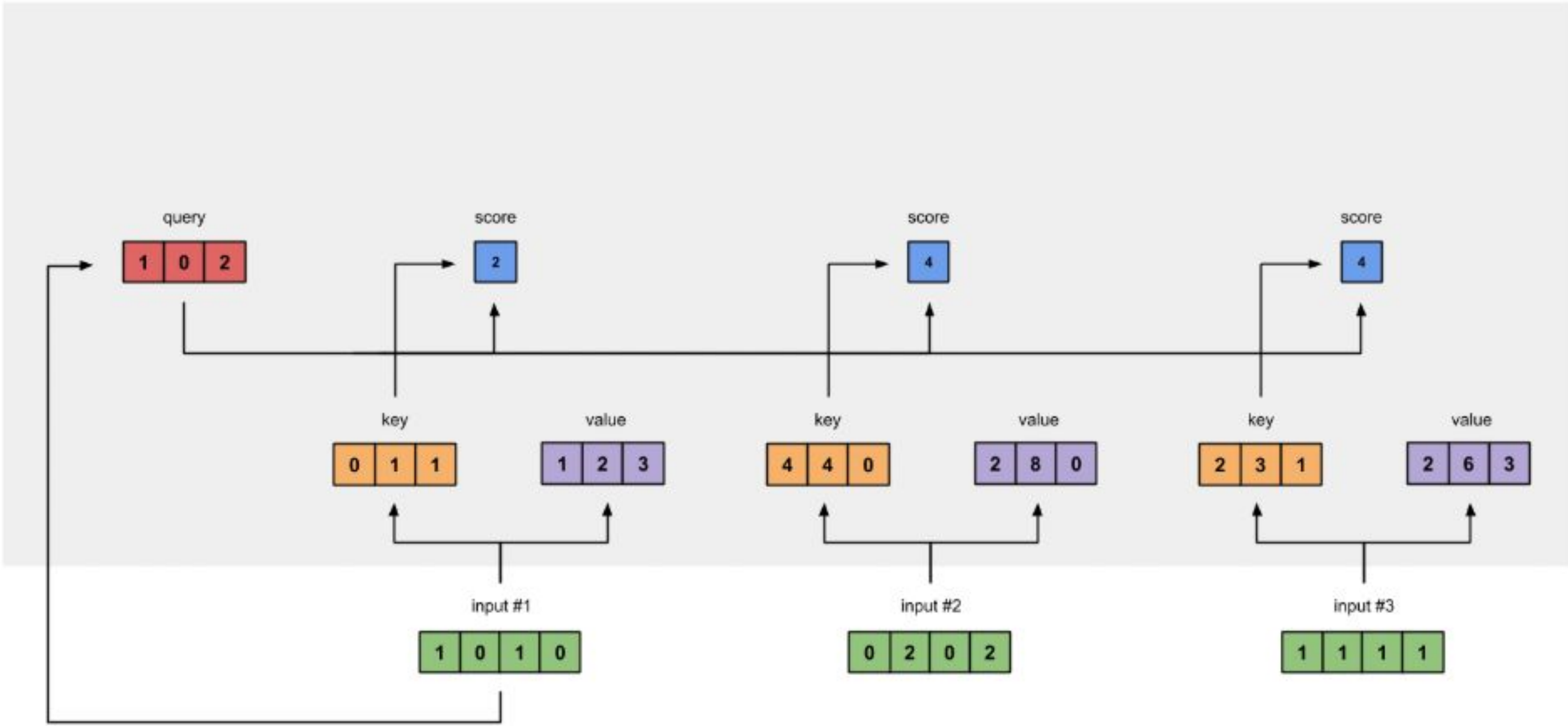
Self Attention Numerical

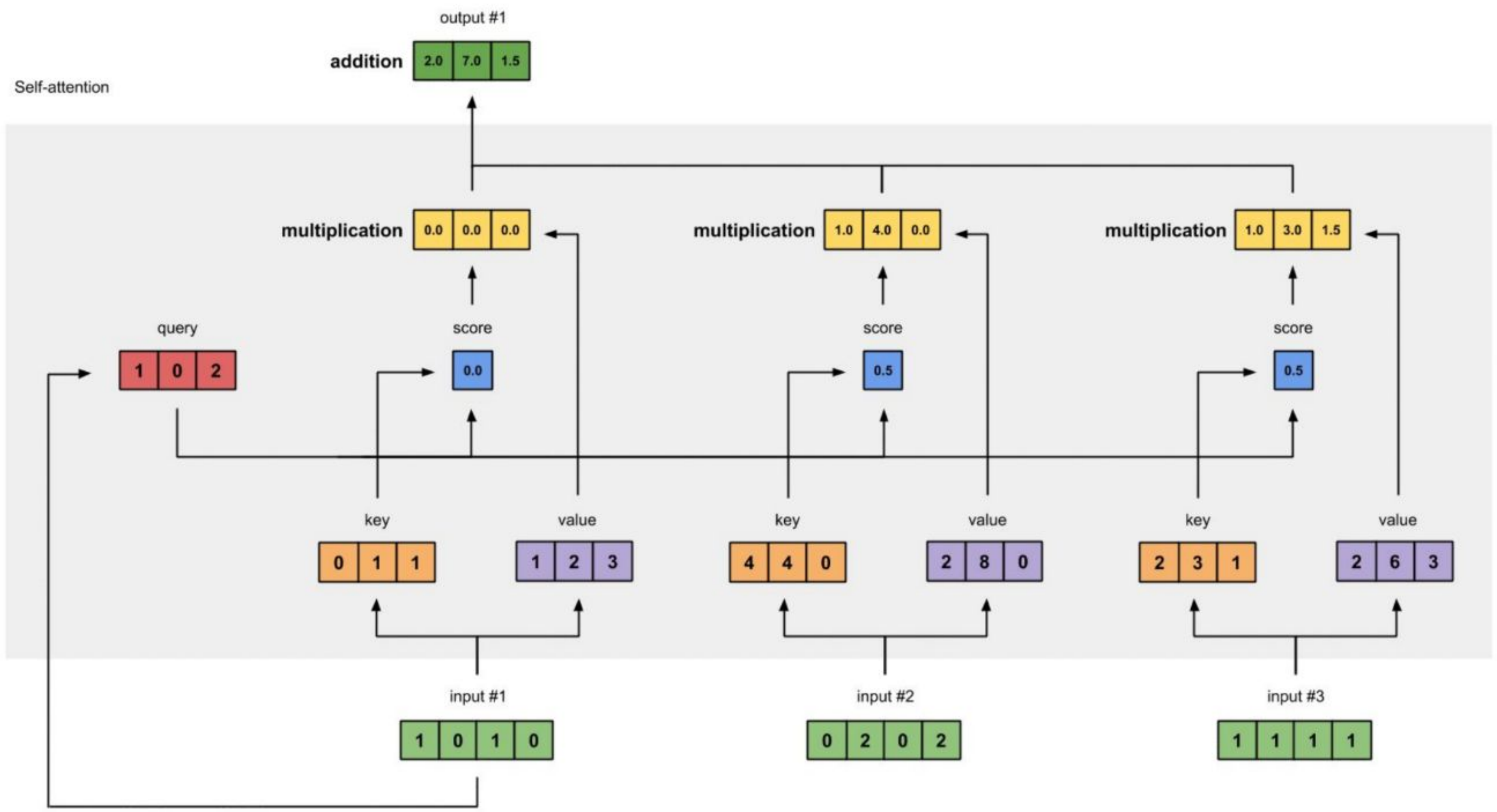
```
x = [
  [1, 0, 1, 0], # Input 1
  [0, 2, 0, 2], # Input 2
  [1, 1, 1, 1]  # Input 3
 ]
```

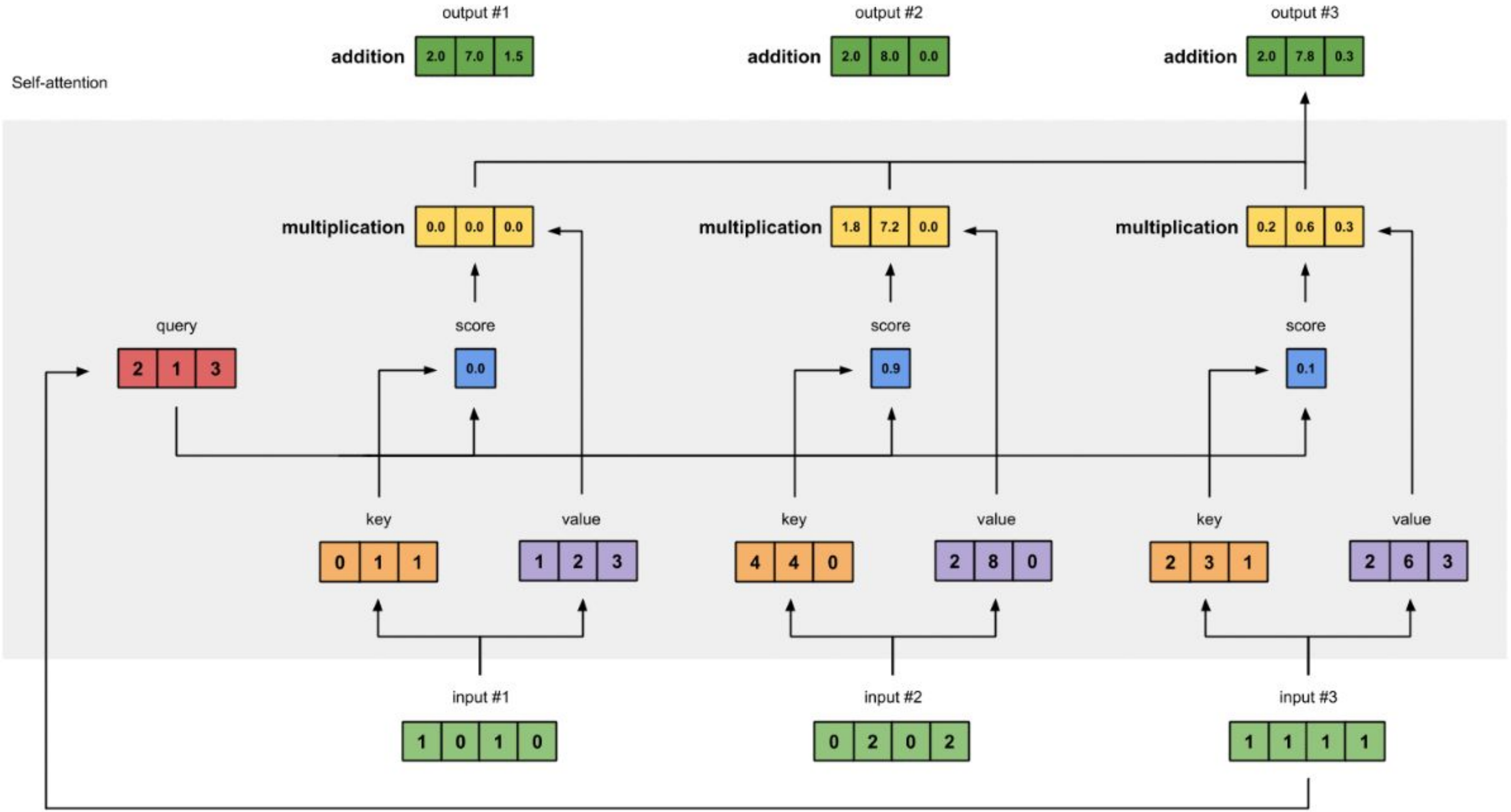
```
w_key = [
  [0, 0, 1],
  [1, 1, 0],
  [0, 1, 0],
  [1, 1, 0] ]
w_query = [
  [1, 0, 1],
  [1, 0, 0],
  [0, 0, 1],
  [0, 1, 1] ]
w_value = [
  [0, 2, 0],
  [0, 3, 0],
  [1, 0, 3],
  [1, 1, 0] ]
```


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

Self-attention

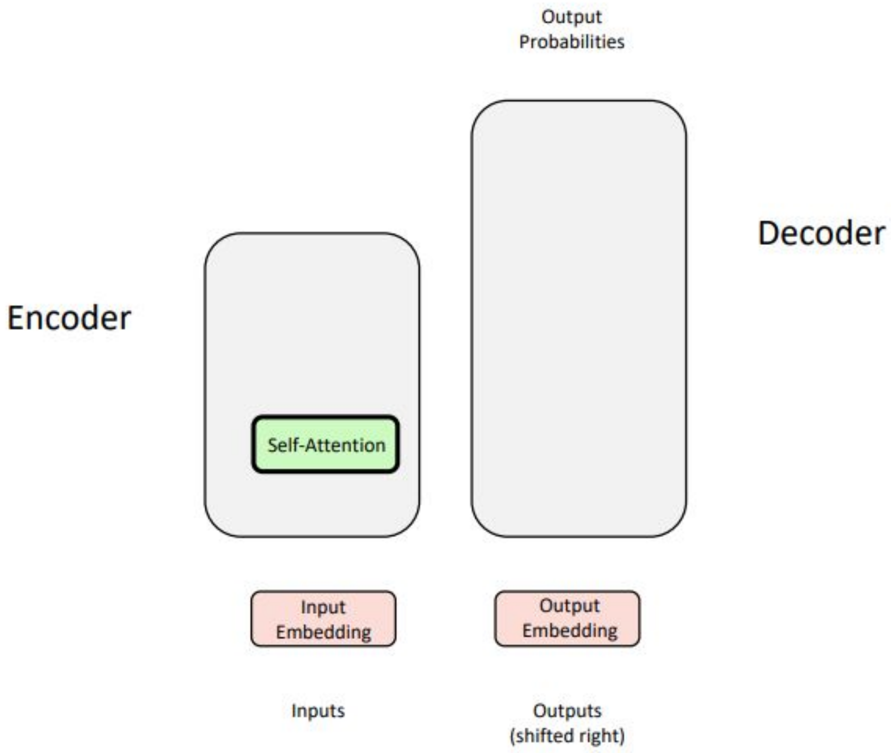



output #1
addition
2.0
7.0
1.5
Self-attention
multiplication
0.0
0.0
0.0
multiplication
1.0
4.0
0.0
multiplication
1.0
3.0
1.5
query
1
0
2
score
0.0
score
0.5
score
0.5
key
0
1
1
value
1
2
3
key
4
4
0
value
2
8
0
key
2
3
1
value
2
6
3
input #1
1
0
1
0
input #2
0
2
0
2
input #3
1
1
1
1



Self-attention
output #1
addition
2.0 7.0 1.5
output #2
addition
2.0 8.0 0.0
output #3
addition
2.0 7.8 0.3
multiplication
0.0 0.0 0.0
multiplication
1.8 7.2 0.0
multiplication
0.2 0.6 0.3
query
2 1 3
score
0.0
score
0.9
score
0.1
key
0 1 1
value
1 2 3
key
4 4 0
value
2 8 0
key
2 3 1
value
2 6 3
input #1
1 0 1 0
input #2
0 2 0 2
input #3
1 1 1 1

What We Have So Far: (Encoder) Self-Attention!

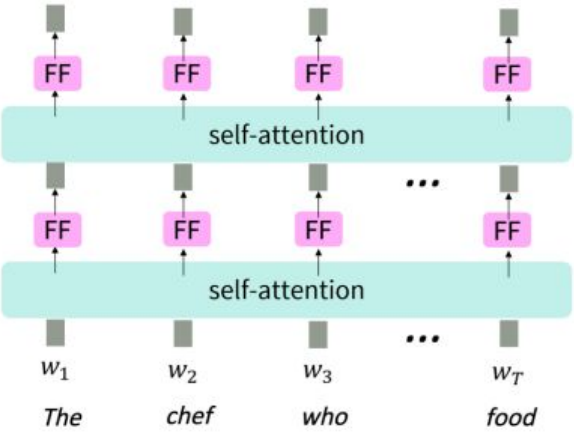


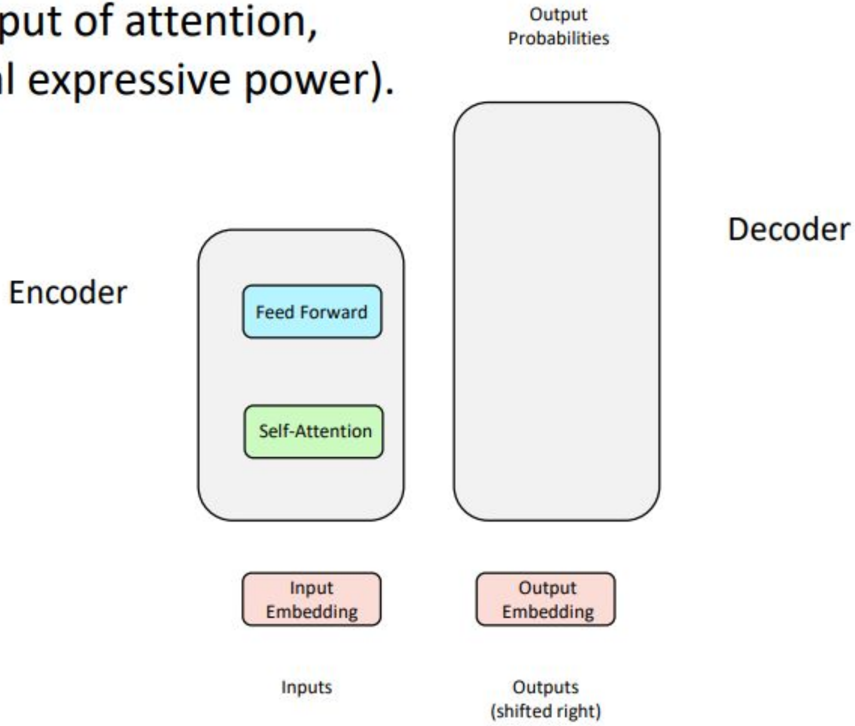
But attention isn't quite all you need!

- **Problem:** Since there are no element-wise non-linearities, self-attention is simply performing a re-averaging of the value vectors.
- **Easy fix:** Apply a feedforward layer to the output of attention, providing non-linear activation (and additional expressive power).

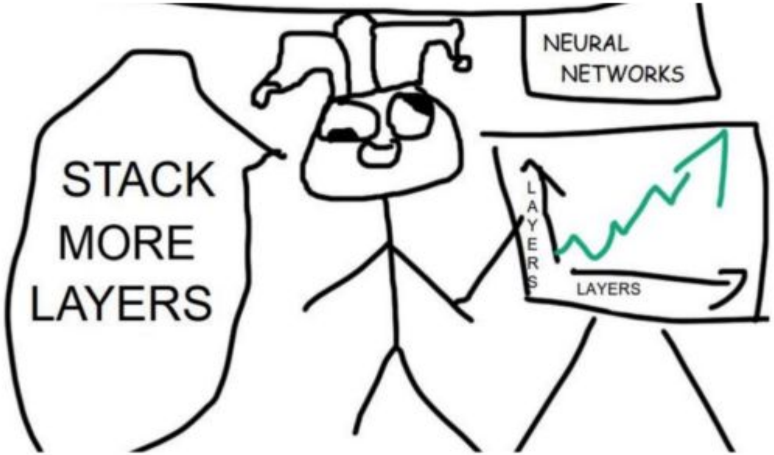
Equation for Feed Forward Layer

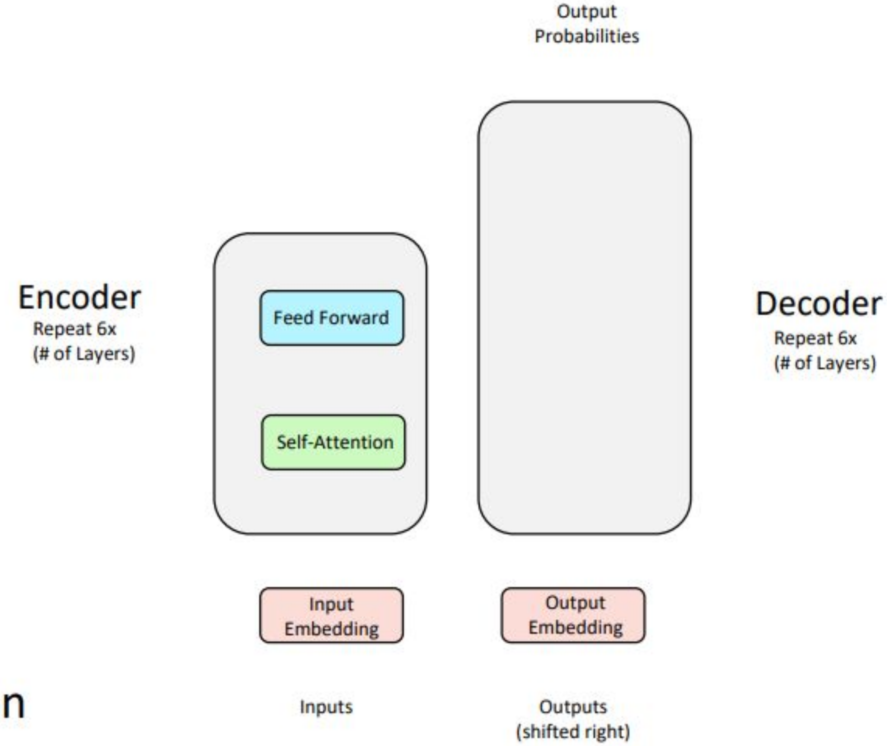
$$
\begin{aligned}
m_i &= MLP(\text{output}_i) \\
&= W_2 * \text{ReLU}(W_1 \times \text{output}_i + b_1) + b_2
\end{aligned}
$$





But how do we make this work for deep networks?





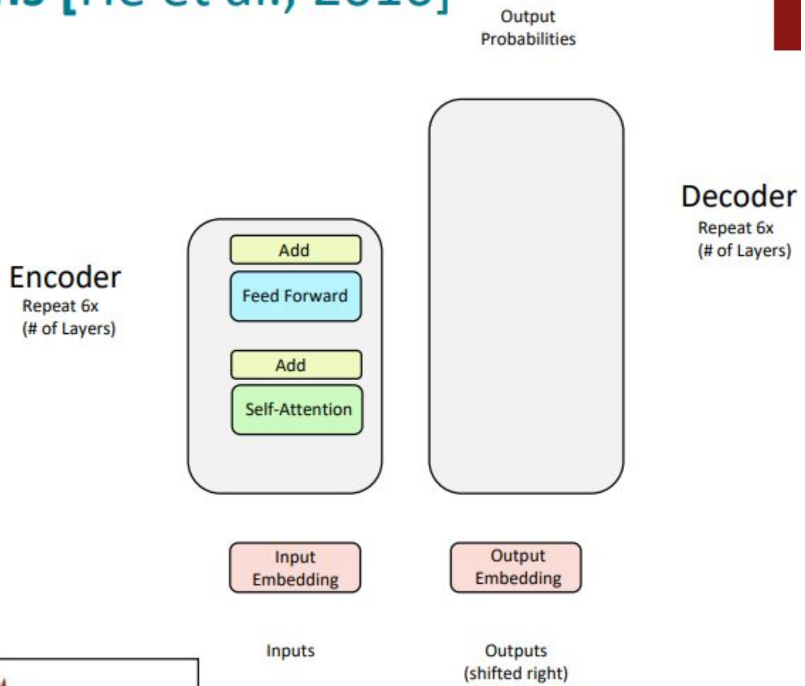
Training Trick #1: Residual Connections
Training Trick #2: LayerNorm
Training Trick #3: Scaled Dot Product Attention

Training Trick #1: Residual Connections [He et al., 2016]

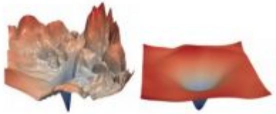
- Residual connections are a simple but powerful technique from computer vision.
- Deep networks are surprisingly bad at learning the identity function!
- Therefore, directly passing "raw" embeddings to the next layer can actually be very helpful!

$$x_\ell = F(x_{\ell-1}) + x_{\ell-1}$$

- This prevents the network from "forgetting" or distorting important information as it is processed by many layers.



Residual connections are also thought to smooth the loss landscape and make training easier!



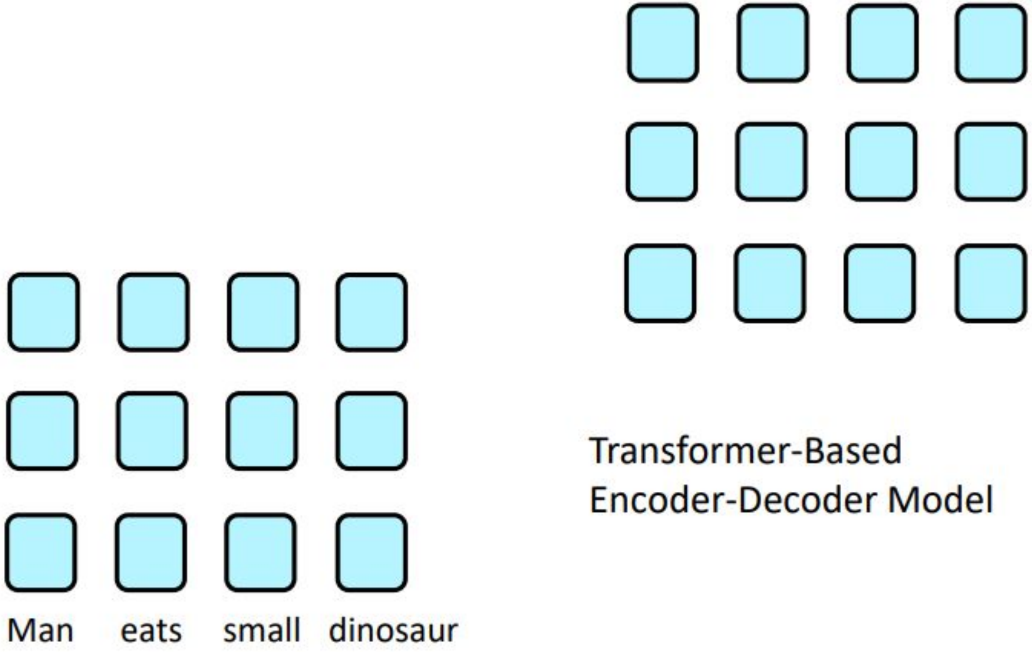
[no residuals] [residuals]

[Loss landscape visualization, Li et al., 2018, on a ResNet]

Major issue!

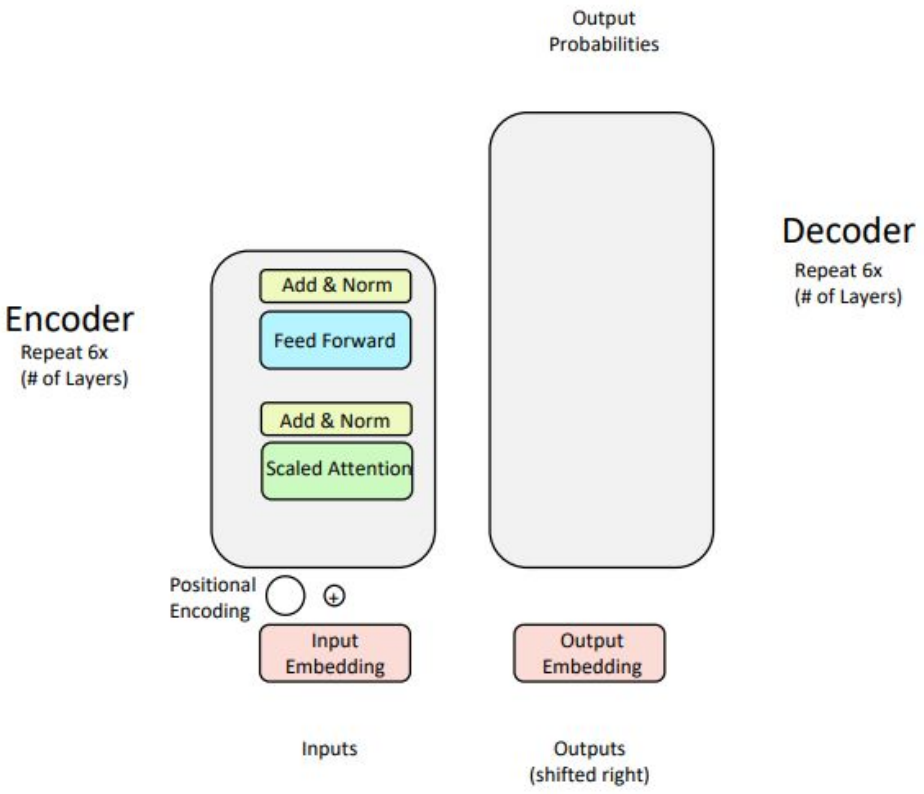
- We're almost done with the Encoder, but we have a major problem! Has anyone spotted it?
- Consider this sentence:
  - "Man eats small dinosaur."

$$Output = softmax\left(QK^T / \sqrt{d_k}\right)V$$



Transformer-Based Encoder-Decoder Model

Solution: Inject Order Information through Positional Encodings!



Fixing the first self-attention problem: **sequence order**

- Since self-attention doesn't build in order information, we need to encode the order of the sentence in our keys, queries, and values.
- Consider representing each **sequence index** as a **vector**

$p_i \in \mathbb{R}^d$, for $i \in \{1,2,\dots,T\}$ are position vectors

- Don't worry about what the p_i are made of yet!
- Easy to incorporate this info into our self-attention block: just add the p_i to our inputs!
- Let $\tilde{v}_i$ $\tilde{k}_i$, $\tilde{q}_i$ be our old values, keys, and queries.

$$
\begin{aligned}
v_i &= \tilde{v}_i + p_i \\
q_i &= \tilde{q}_i + p_i \\
k_i &= \tilde{k}_i + p_i
\end{aligned}
$$

In deep self-attention networks, we do this at the first layer! You could concatenate them as well, but people mostly just add…

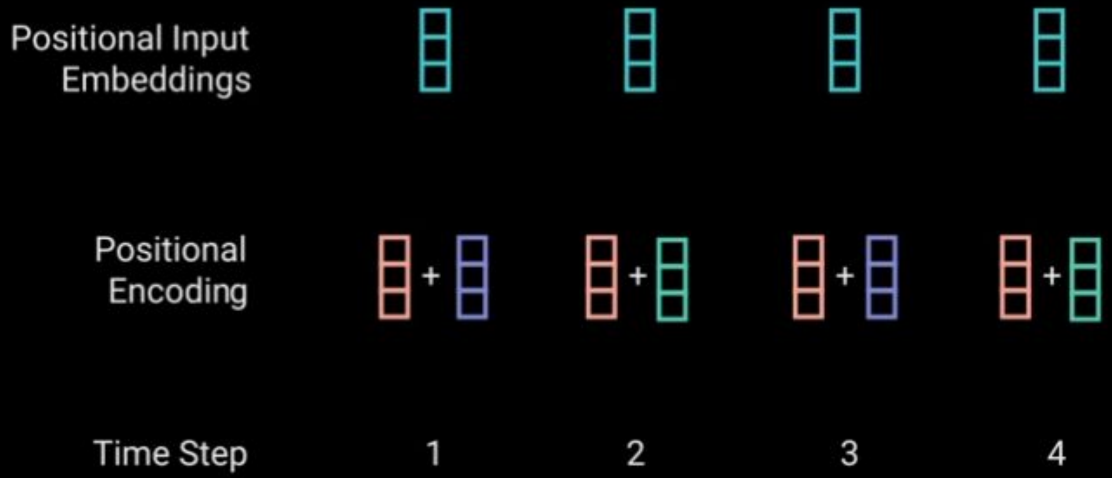
Positional Encoding

- Account for the order of the words in the input sequence.
- Adds a vector to each input embedding. These vectors follow a specific pattern that helps determine the position of each word.

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{model}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{model}})$$

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

2. Positional Encoding

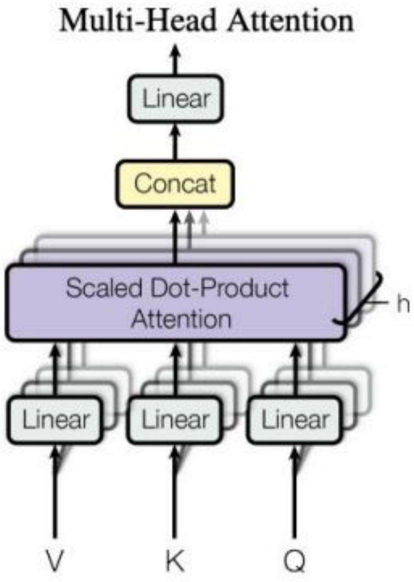


$$PE(pos, 2i+1) = cos(\frac{pos}{10000^{2i/dmodel}})$$

$$PE(pos, 2i) = sin(\frac{pos}{10000^{2i/dmodel}})$$

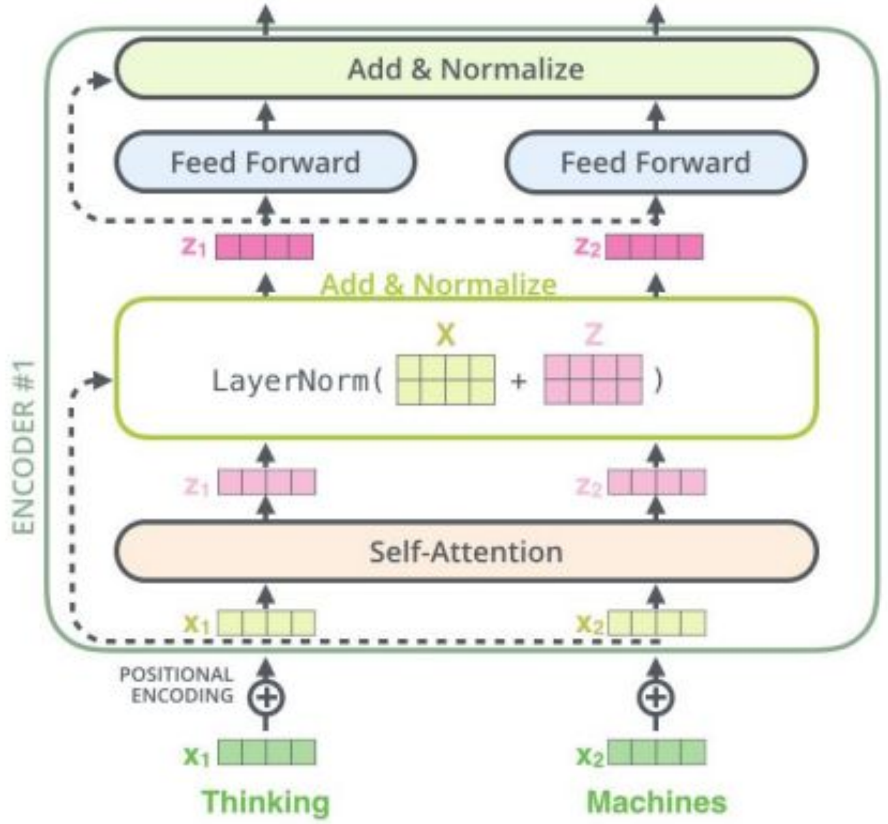
Multi-Headed Self-Attention: k heads are better than 1!

- **High-Level Idea:** Let's perform self-attention multiple times in parallel and combine the results.

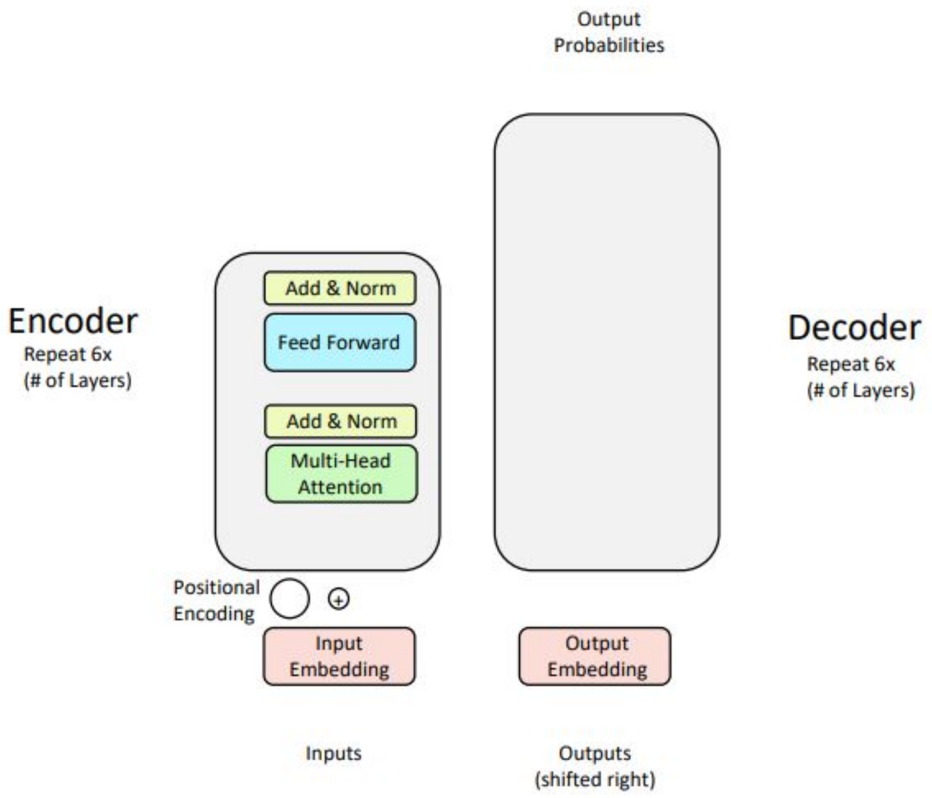


[Vaswani et al. 2017]

Wizards of the Coast, Artist: Todd Lockwood

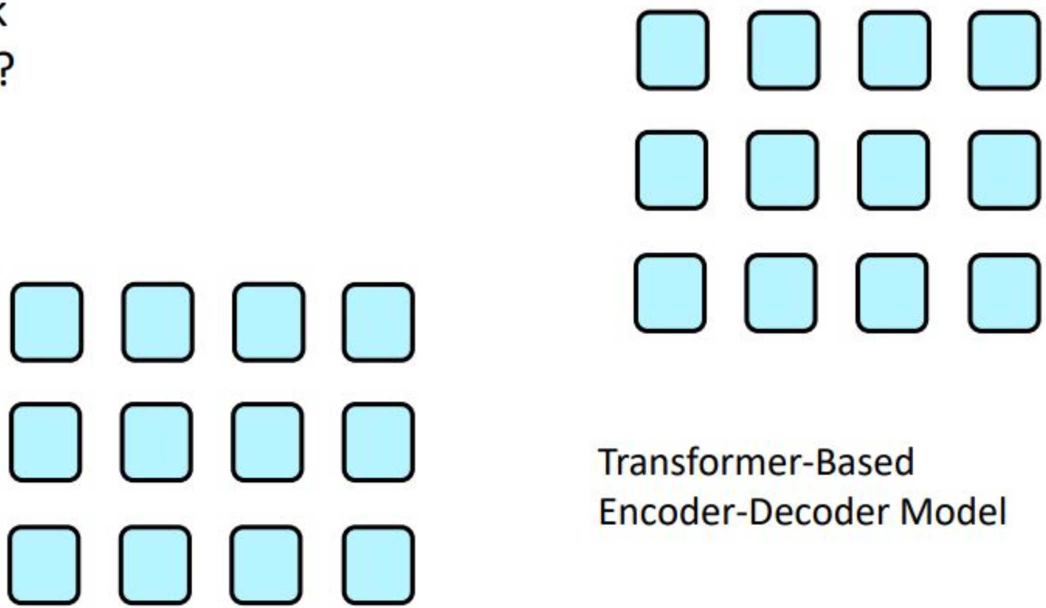

Add & Normalize
Feed Forward
Feed Forward
z1
z2
Add & Normalize
X
Z
LayerNorm(+)
ENCODER #1
z1
z2
Self-Attention
x1
x2
POSITIONAL
ENCODING
x1
x2
Thinking
Machines

Yay, we've completed the Encoder! Time for the Decoder...



Decoder: Masked Multi-Head Self-Attention

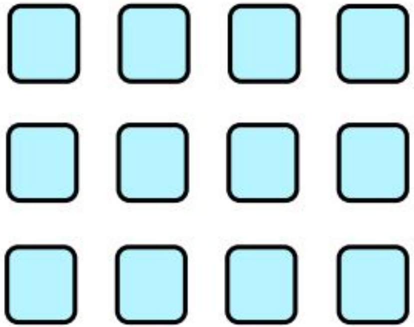
- **Problem:** How do we keep the decoder from cheating? If we have a language modeling objective, can't the network just look ahead and "see" the answer?

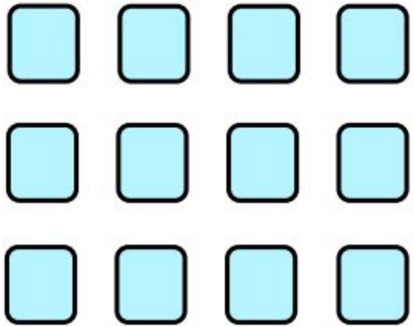


Transformer-Based
Encoder-Decoder Model

Decoder: Masked Multi-Head Self-Attention

- **Problem:** How do we keep the decoder from "cheating"? If we have a language modeling objective, can't the network just look ahead and "see" the answer?
- **Solution:** Masked Multi-Head Attention. At a high-level, we hide (mask) information about future tokens from the model.



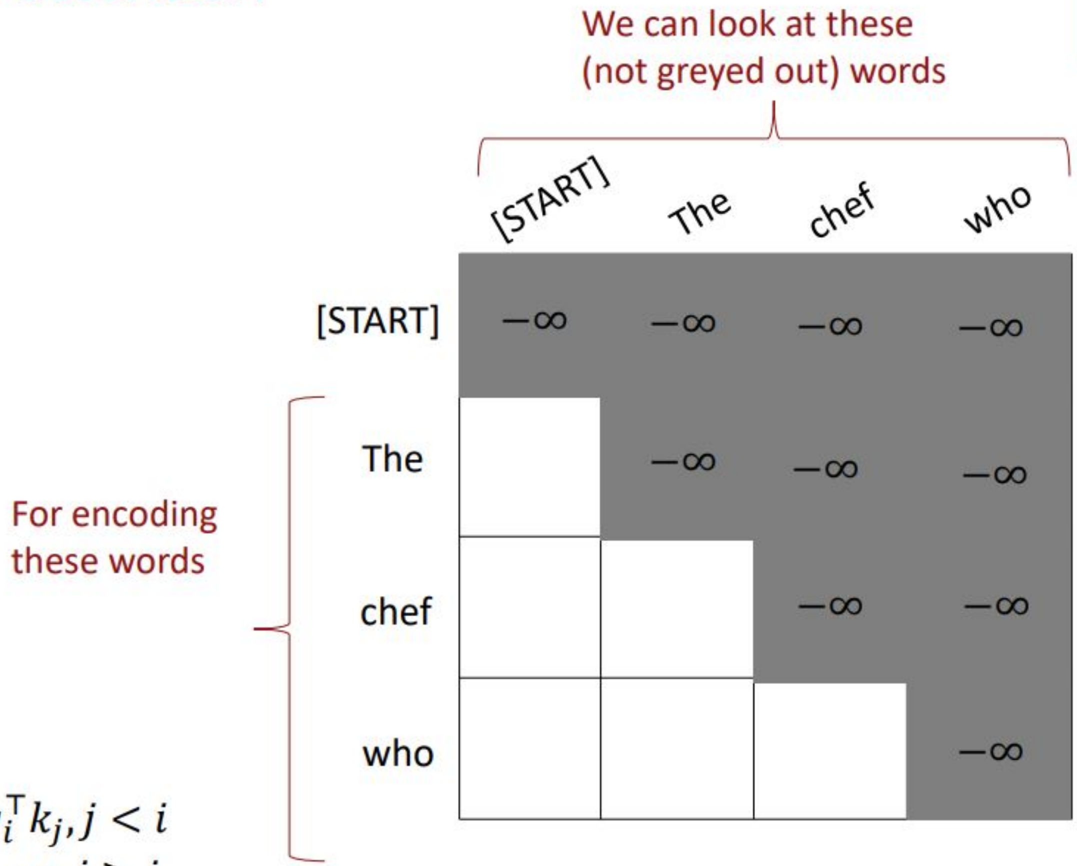


Transformer-Based
Encoder-Decoder Model

Masking the future in self-attention

- To use self-attention in **decoders**, we need to ensure we can't peek at the future.
- At every timestep, we could change the set of **keys and queries** to include only past words. (Inefficient!)
- To enable parallelization, we **mask out attention** to future words by setting attention scores to $-\infty$.

$$e_{ij} = \begin{cases} q_i^\top k_j, j < i \\ -\infty, j \geq i \end{cases}$$



Transformer Numerical

Input sentence: "Transformers transforming our lives"
Output sentence: “For sure”

The embedding matrix for the words is represented as follows:

"Transformers": [0.1, 0.2, 0.3]
"transforming": [0.4, 0.5, 0.6]
"our": [0.7, 0.8, 0.9]
"lives": [1.0, 1.1, 1.2]
"For": [0.4, 0.1, 0.8]
"sure": [0.9, 0.7, 0.2]

Assuming the weight matrices:

$$W_q = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 2 \\ 7 & 1 & 9 \end{bmatrix}$$

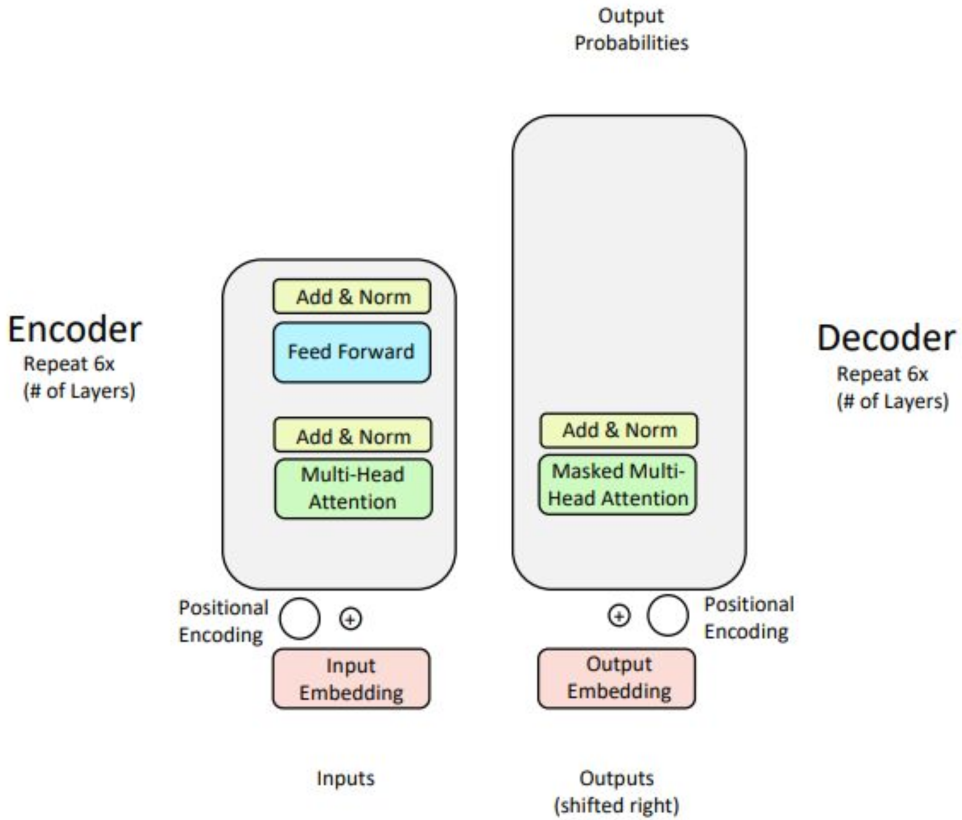
$$W_k = \begin{bmatrix} 1 & 6 & 9 \\ 7 & 3 & 1 \\ 9 & 2 & 1 \end{bmatrix}$$

$$W_v = \begin{bmatrix} 2 & 4 & 6 \\ 8 & 0 & 2 \\ 1 & 6 & 8 \end{bmatrix}$$

For simplicity, assume dk is 1.

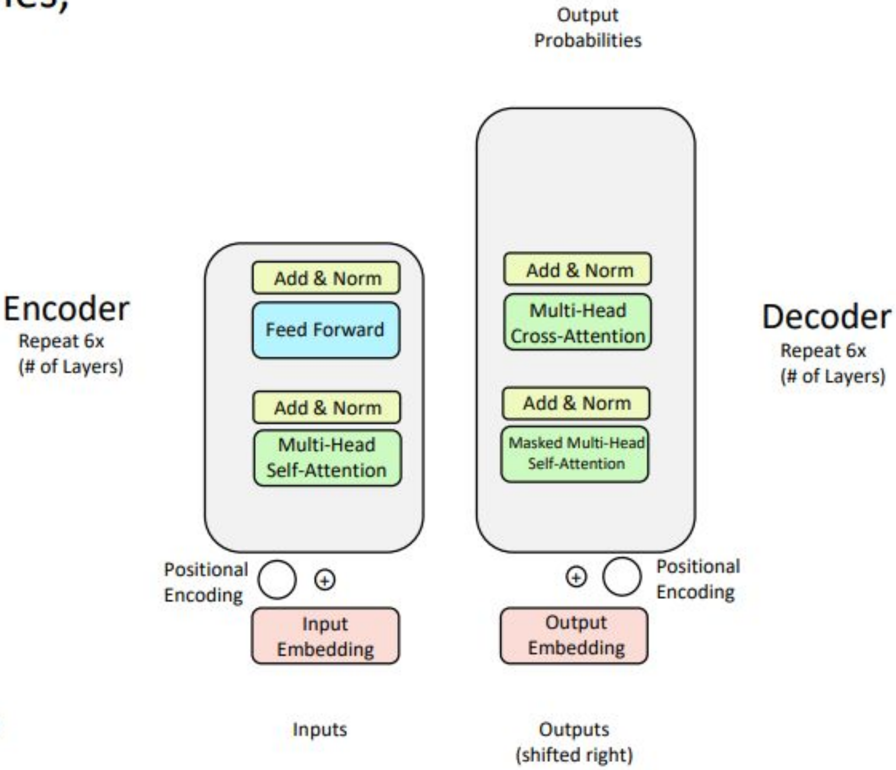
1. Calculate masked self-attention for the above input sentence using the provided weight matrices. Show all steps clearly.
2. Calculate cross-attention for the above input and output sentence using the provided weight matrices. Show all steps clearly.

Decoder: Masked Multi-Headed Self-Attention



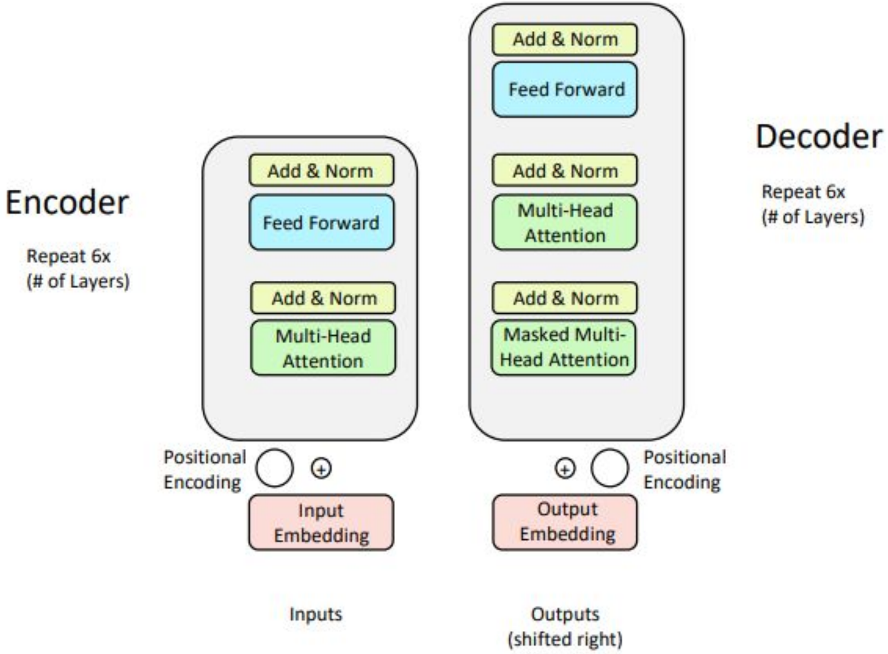
Encoder-Decoder Attention

- We saw that self-attention is when keys, queries, and values come from the same source.
- In the decoder, we have attention that looks more like what we saw last week.
- Let $h_1, \dots, h_T$ be **output** vectors **from** the Transformer **encoder**; $x_i \in \mathbb{R}^d$
- Let $z_1, \dots, z_T$ be input vectors from the Transformer **decoder**, $z_i \in \mathbb{R}^d$
- Then keys and values are drawn from the **encoder** (like a memory):
  - $k_i = Kh_i, v_i = Vh_i$.
- And the queries are drawn from the **decoder**, $q_i = Qz_i$.



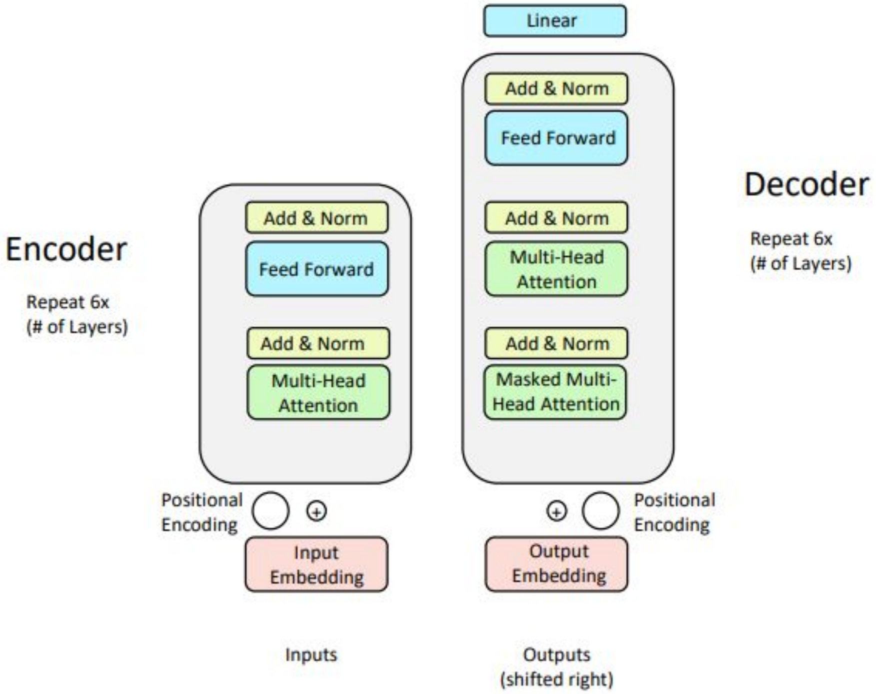
Decoder: Finishing touches!

- Add a feed forward layer (with residual connections and layer norm)



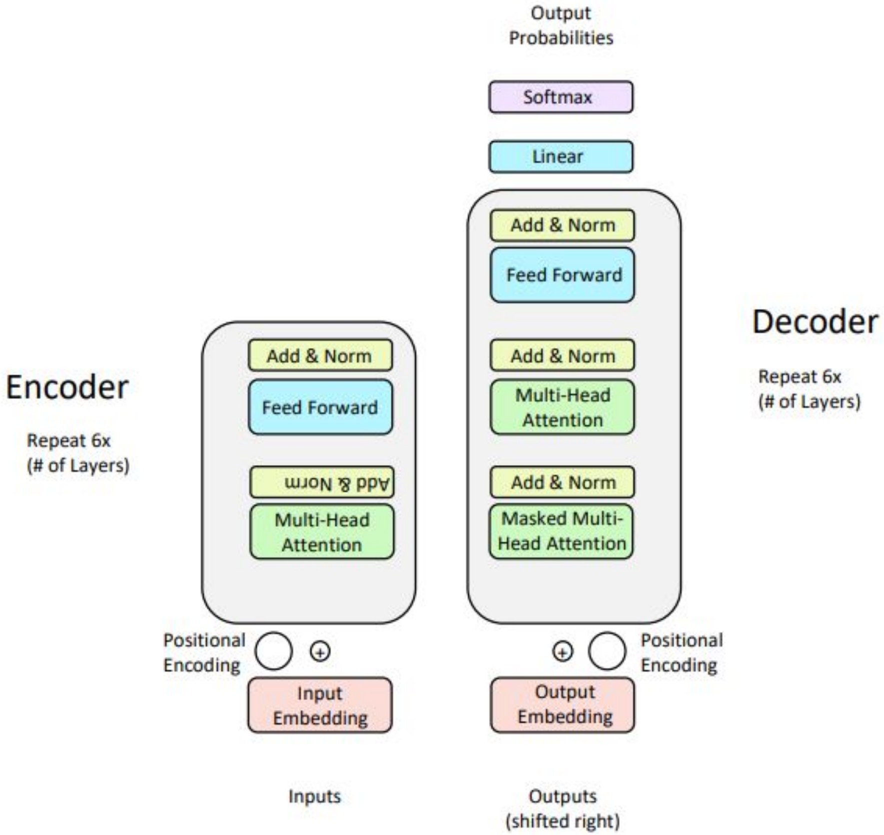
Decoder: Finishing touches!

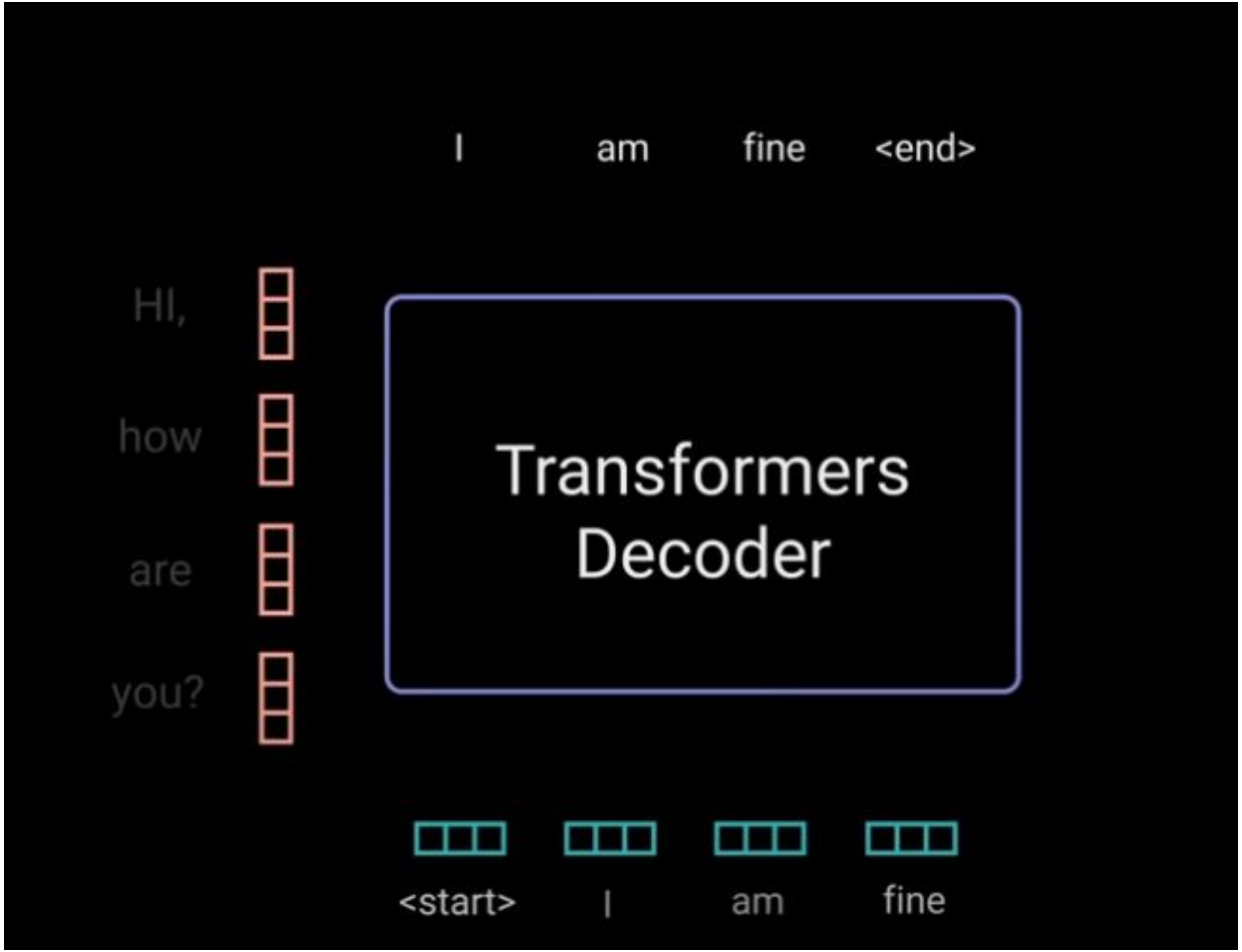
- Add a feed forward layer (with residual connections and layer norm)
- Add a final linear layer to project the embeddings into a much longer vector of length vocab size (logits)

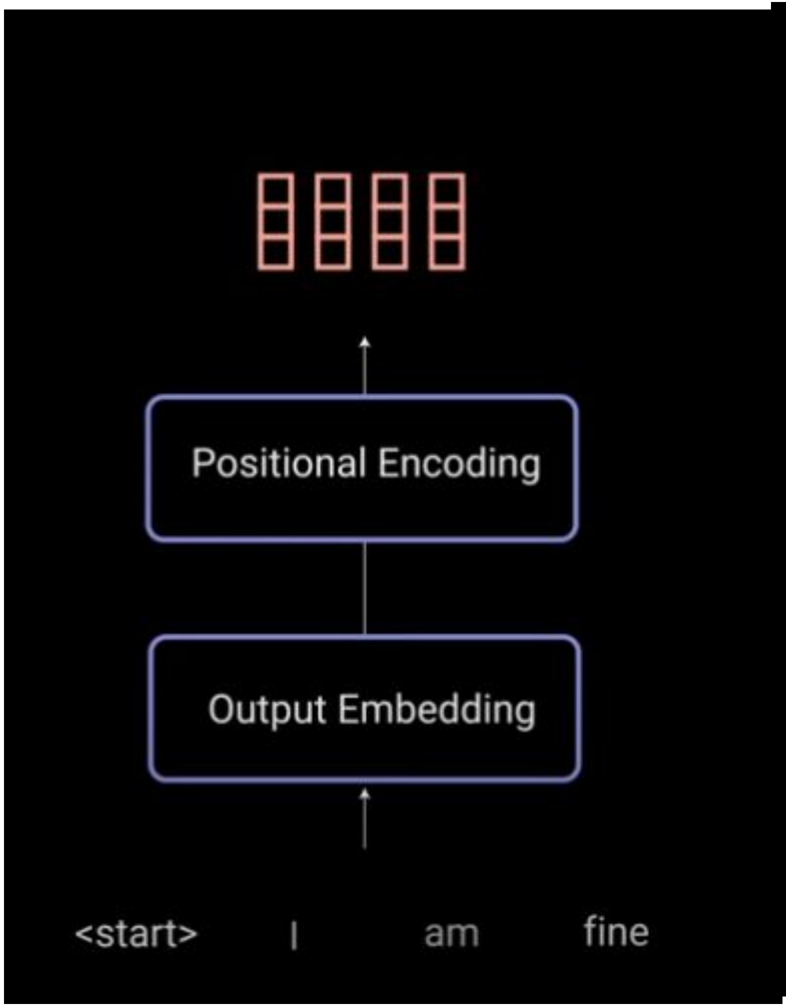


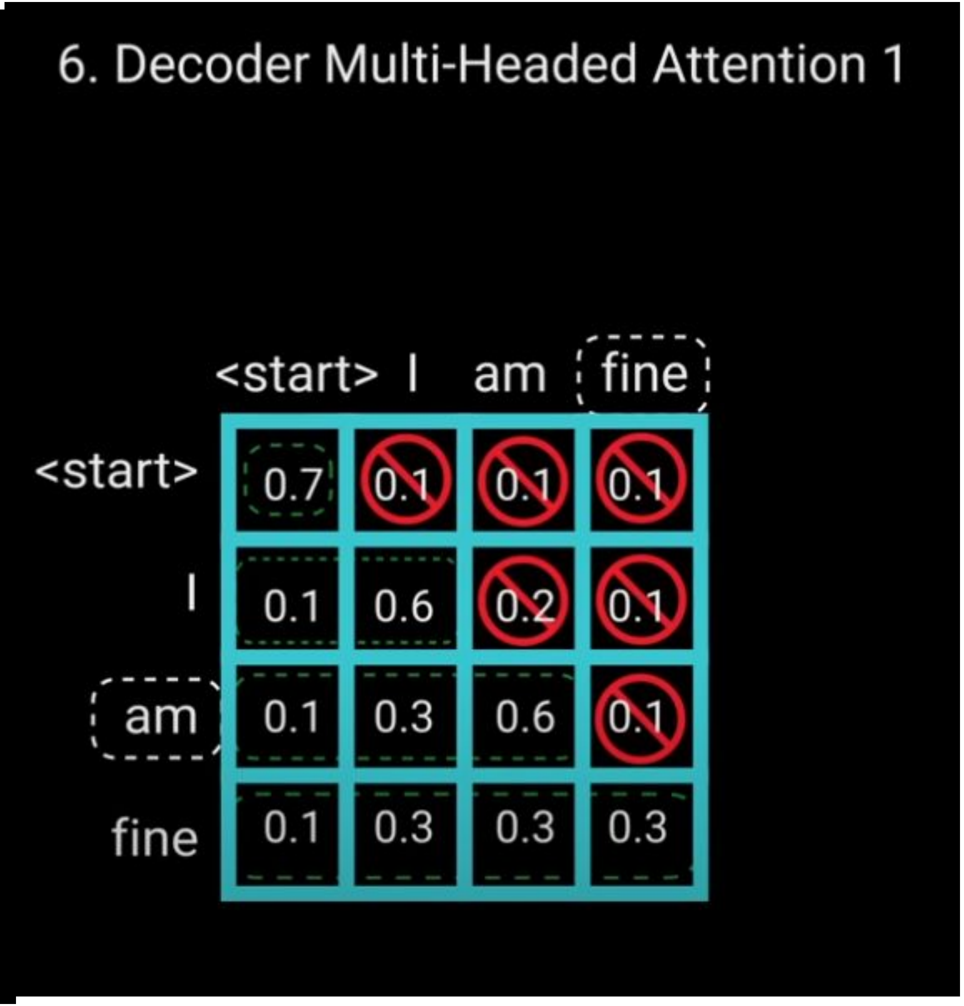
Decoder: Finishing touches!

- Add a feed forward layer (with residual connections and layer norm)
- Add a final linear layer to project the embeddings into a much longer vector of length vocab size (logits)
- Add a final softmax to generate a probability distribution of possible next words!




I
am
fine
<end>
HI,
how
are
you?
Transformers
Decoder
<start>
I
am
fine


Positional Encoding
Output Embedding
<start>
I
am
fine


6. Decoder Multi-Headed Attention 1
<start> I am fine
<start> 0.7 0.1 0.1 0.1
I 0.1 0.6 0.2 0.1
am 0.1 0.3 0.6 0.1
fine 0.1 0.3 0.3 0.3

6. Decoder Multi-Headed Attention 1

6.1. Look-Ahead Mask

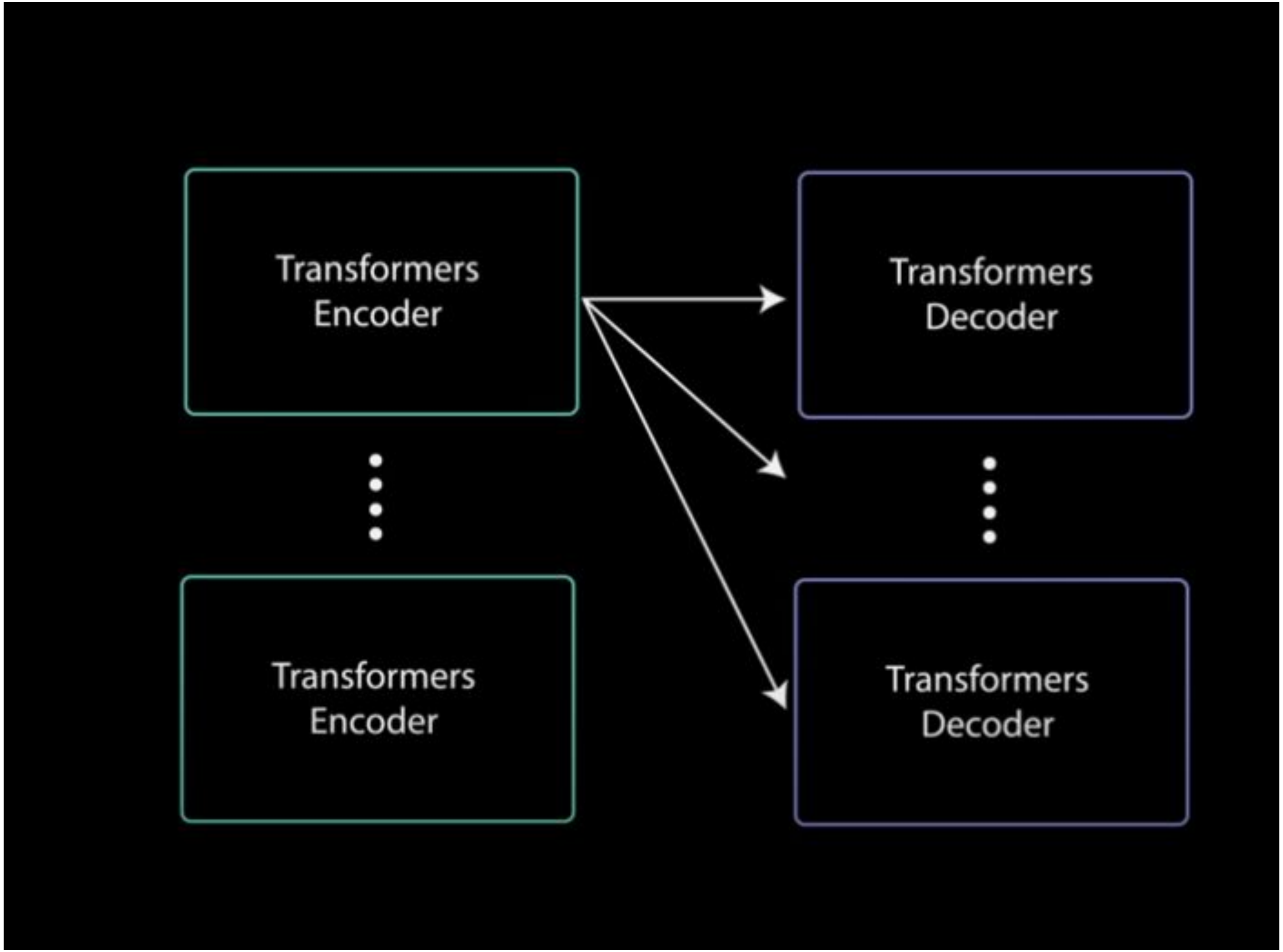
Scaled Scores + Look-Ahead Mask = Masked Scores

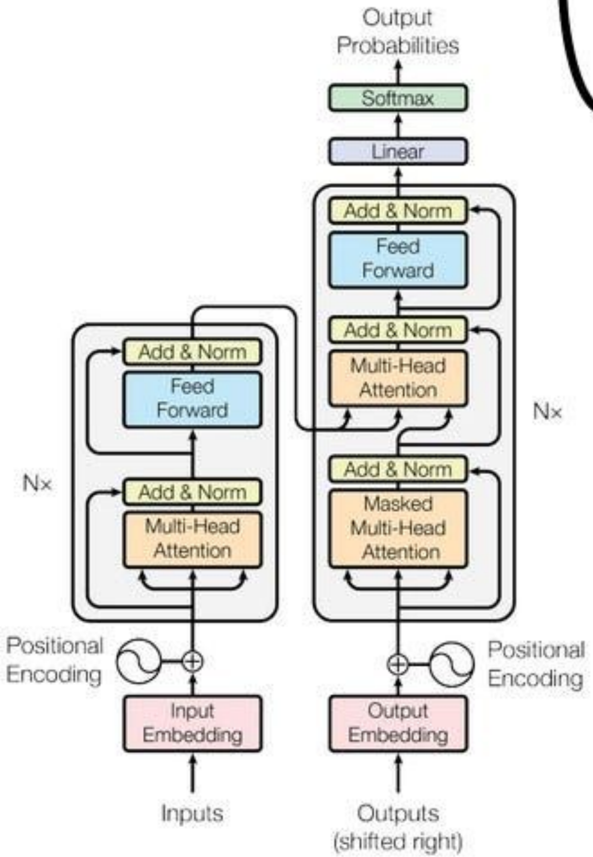
$$\begin{bmatrix} 0.7 & 0.1 & 0.1 & 0.1 \\ 0.1 & 0.6 & 0.2 & 0.1 \\ 0.1 & 0.3 & 0.6 & 0.1 \\ 0.1 & 0.3 & 0.3 & 0.3 \end{bmatrix} + \begin{bmatrix} 0 & -inf & -inf & -inf \\ 0 & 0 & -inf & -inf \\ 0 & 0 & 0 & -inf \\ 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0.7 & -inf & -inf & -inf \\ 0.1 & 0.6 & -inf & -inf \\ 0.1 & 0.3 & 0.6 & -inf \\ 0.1 & 0.3 & 0.3 & 0.3 \end{bmatrix}$$

6. Decoder Multi-Headed Attention 1

6.1. Look-Ahead Mask

$$
\text{Softmax}\left(\begin{bmatrix} 0.7 & \text{-inf} & \text{-inf} & \text{-inf} \\ 0.1 & 0.6 & \text{-inf} & \text{-inf} \\ 0.1 & 0.3 & 0.6 & \text{-inf} \\ 0.1 & 0.3 & 0.3 & 0.3 \end{bmatrix}\right) = \begin{array}{c c} & \begin{array}{cccc} \text{<start>} & \text{I} & \text{am} & \text{fine} \end{array} \\ \begin{array}{r} \text{<start>} \\ \text{I} \\ \text{am} \\ \text{fine} \end{array} & \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0.37 & 0.62 & 0 & 0 \\ 0.26 & 0.31 & 0.43 & 0 \\ 0.21 & 0.26 & 0.26 & 0.26 \end{bmatrix} \end{array}
$$


Transformers
Encoder
Transformers
Encoder
Transformers
Decoder
Transformers
Decoder


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


COME, LET US UNDERSTAND
THE UNCOOL
TRANSFORMERS!!!
- OPTIMUS PRIME

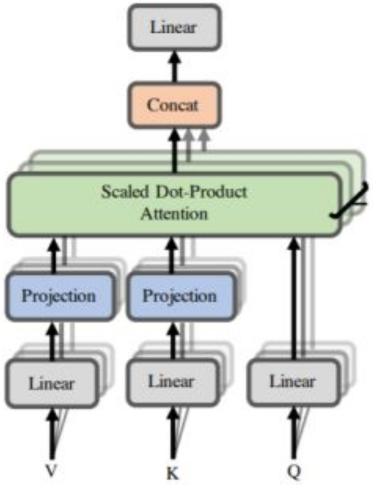
What would we like to fix about the Transformer?

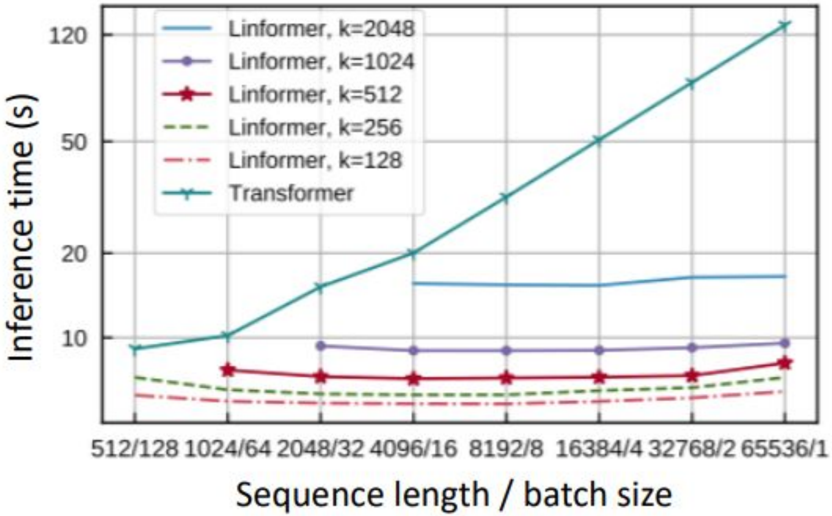
- **Quadratic compute in self-attention (today)**:
  - Computing all pairs of interactions means our computation grows **quadratically** with the sequence length!
  - For recurrent models, it only grew linearly!
- **Position representations**:
  - Are simple absolute indices the best we can do to represent position?
  - Relative linear position attention [Shaw et al., 2018]
  - Dependency syntax-based position [Wang et al., 2019]

Recent work on improving on quadratic self-attention cost

- Considerable recent work has gone into the question, *Can we build models like Transformers without paying the* $O(T^2)$ *all-pairs self-attention cost?*
- For example, **Linformer** [Wang et al., 2020]

Key idea: map the sequence length dimension to a lower-dimensional space for values, keys

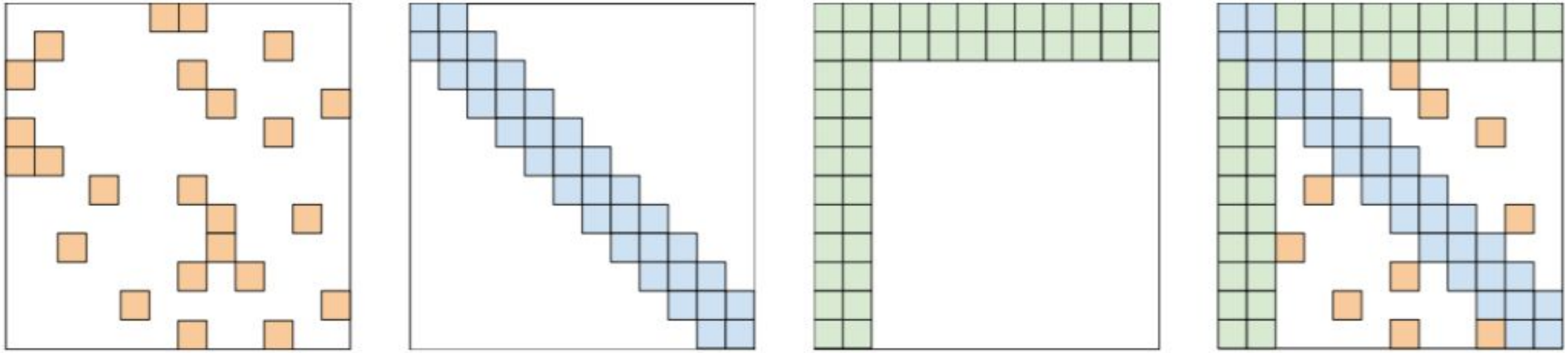




Recent work on improving on quadratic self-attention cost

- Considerable recent work has gone into the question, *Can we build models like Transformers without paying the $O(T^2)$ all-pairs self-attention cost?*
- For example, **BigBird** [Zaheer et al., 2021]

Key idea: replace all-pairs interactions with a family of other interactions, **like local windows**, **looking at everything**, and **random interactions**.



(a) Random attention (b) Window attention (c) Global Attention (d) BIGBIRD

References for Self Attention

https://towardsdatascience.com/transformers-in-action-attention-is-all-you-need-ac10338a023a

https://web.stanford.edu/class/cs224n/slides/cs224n-2022-lecture09-transformers.pdf

https://pretteyandnerdy.wordpress.com/2019/04/26/attention-is-all-you-need/

https://www.youtube.com/watch?v=4Bdc55j80l8

https://colab.research.google.com/drive/1rPk3ohrmVclqhH7uQ7qys4oznDdAhpzF#scrollTo=DZ96EoE1Bvat