Computer Vision

Usman Nazir

2011

(Excerpt from a representative paper in applied computer vision)

4. Computational Scene Classification

In this section we explore how discriminable the SUN categories are with a variety of image features and kernels paired with 1 vs. all support vector machines.

4.1. Image Features and Kernels

We selected or designed several state-of-art features that are potentially useful for scene classification. GIST features [21] are proposed specifically for scene recognition tasks. Dense SIFT features are also found to perform very well at the 15-category dataset [17]. We also evaluate sparse SIFTs as used in "Video Google" [27]. HOG features provide excellent performance for object and human recognition tasks [4, 9], so it is interesting to examine their utility for scene recognition. While SIFT is known to be very good at finding repeated image content, the self-similarity descriptor (SSIM) [26] relates images using their internal layout of local self-similarities. Unlike GIST, SIFT, and HOG, which are all local gradient-based approaches, SSIM may provide a distinct, complementary measure of scene layout that is somewhat appearance invariant. As a baseline, we also include Tiny Images [28], color histograms and straight line histograms. To make our color and texton histograms more invariant to scene layout, we also build histograms for specific geometric classes as determined by [13]. The geometric classification of a scene is then itself used as a feature, hopefully being invariant to appearance but responsive to layout.

GIST: The GIST descriptor [21] computes the output energy of a bank of 24 filters. The filters are Gabor-like filters tuned to 8 orientations at 4 different scales. The square output of each filter is then averaged on a 4×4 grid.

HOG2x2: First, histogram of oriented edges (HOG) descriptors [4] are densely extracted on a regular grid at steps of 8 pixels. HOG features are computed using the code available online provided by [9], which gives a 31-dimension descriptor for each node of the grid. Then, 2×2 neighboring HOG descriptors are stacked together to form a descriptor with 124 dimensions. The stacked descriptors spatially overlap. This 2×2 neighbor stacking is important because the higher feature dimensionality provides more descriptive power. The descriptors are quantized into 300 visual words by k-means. With this visual word representation, three-level spatial histograms are computed on grids of 1×1, 2×2 and 4×4. Histogram intersection[17] is used to define the similarity of two histograms at the same pyramid level for two images. The kernel matrices at the three levels are normalized by their respective means, and linearly combined together using equal weights.

Dense SIFT: As with HOG2x2, SIFT descriptors are densely extracted [17] using a flat rather than Gaussian window at two scales (4 and 8 pixel radii) on a regular grid at steps of 5 pixels. The three descriptors are stacked together for each HSV color channels, and quantized into 300 visual words by k-means, and spatial pyramid histograms are used as kernels[17].

:S

Geometry Specific Histograms: Inspired by "Illumination Context" [16], we build color and texton histograms for each geometric class (ground, vertical, porous, and sky). Specifically, for each color and texture sample, we weight its contribution to each histogram by the probability that it belongs to that geometric class. These eight histograms are compared with χ^2 distance after normalization.

S

… and it still doesn't exactly work

All other AI areas were totally different

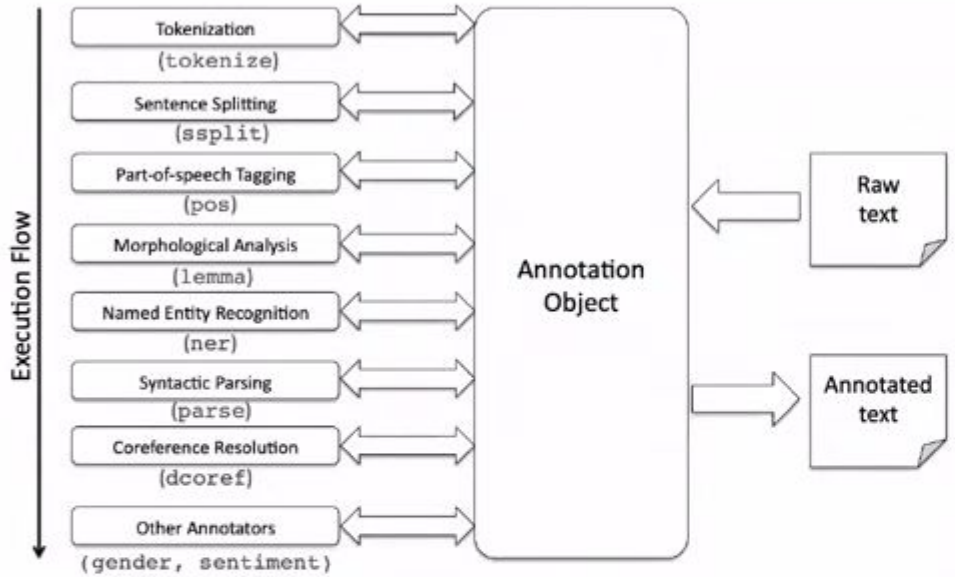


Figure 1: Overall system architecture: Raw text is put into an Annotation object and then a sequence of Annotators add information in an analysis pipeline. The resulting Annotation, containing all the analysis information added by the Annotators, can be output in XML or plain text forms.

In SCFG, the grammar specifies *two* output strings for each pr
a correspondence between strings via the use of *co-indexed* nonter
fragment of SCFG grammar.

$$\begin{aligned}
\text{NP} &\longrightarrow \text{DT}_{[1]}\text{NPB}_{[2]} \;/\; \text{DT}_{[1]}\text{NPB}_{[2]} \\
\text{NPB} &\longrightarrow \text{JJ}_{[1]}\text{NN}_{[2]} \;/\; \text{JJ}_{[1]}\text{NN}_{[2]} \\
\text{NPB} &\longrightarrow \text{NPB}_{[1]}\text{JJ}_{[2]} \;/\; \text{JJ}_{[2]}\text{NPB}_{[1]} \\
\text{DT} &\longrightarrow \text{the} \;/\; \varepsilon \\
\text{JJ} &\longrightarrow \text{strong} \;/\; \text{呼啸} \\
\text{JJ} &\longrightarrow \text{north} \;/\; \text{北} \\
\text{NN} &\longrightarrow \text{wind} \;/\; \text{风}
\end{aligned}$$

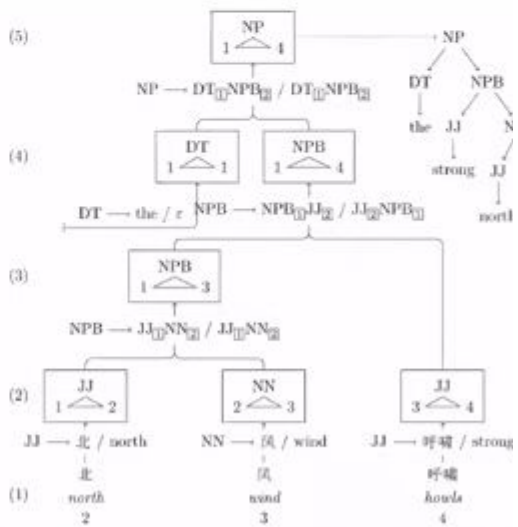


Fig. 16. An illustration of SCFG decoding, which is equivalent to parsing the source la
source word and associate it with a span. (2) Apply SCFG rules that match the target
infer larger spans from smaller spans. (4) We can also infer target language words w
in the source language, if our grammar contains SCFG rules that produce these words
in the source language. (5) Read off the tree in target-language order.

2012

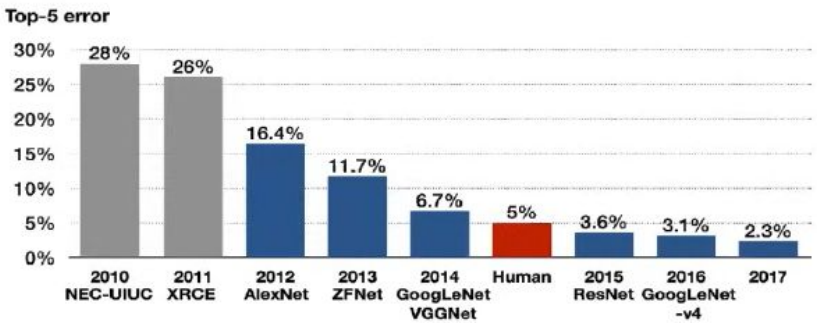
then this happened…

ImageNet Classification with Deep Convolutional Neural Networks

Alex Krizhevsky University of Toronto kriz@cs.utoronto.ca

Ilya Sutskever University of Toronto ilya@cs.utoronto.ca

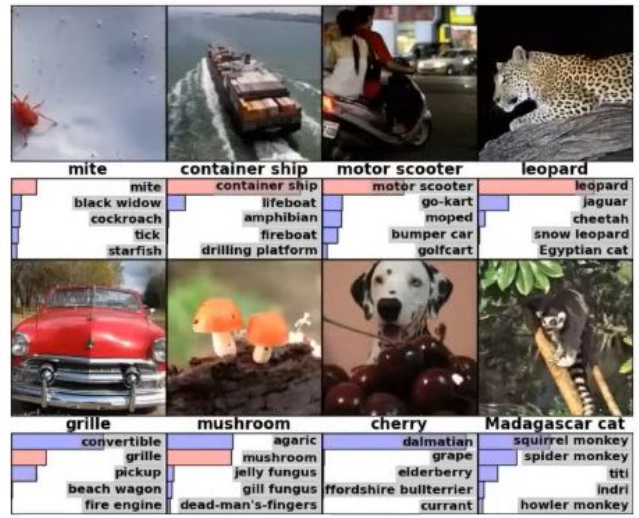
Geoffrey E. Hinton University of Toronto hinton@cs.utoronto.ca

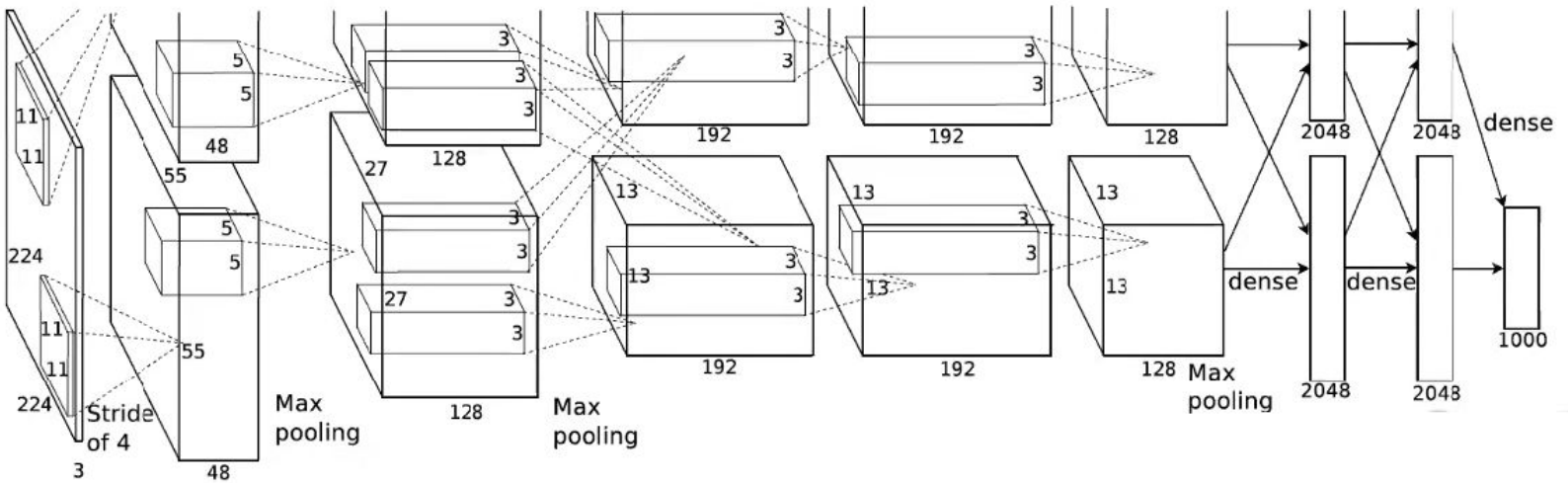


ImageNet Challenge

IMAGENET

- 1,000 object classes (categories).
- Images:
  - 1.2 M train
  - 100k test.

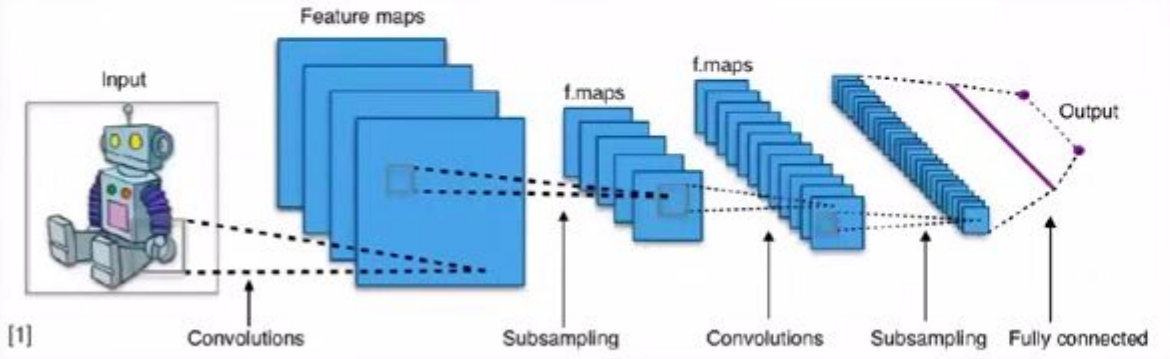




Neural nets started to take over AI. They all looked different…

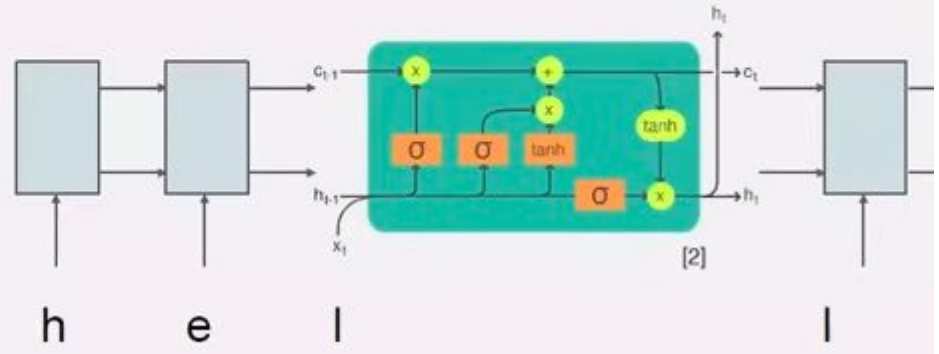
Computer Vision

Convolutional NNs (+ResNets)



Natural Lang.

Recurrent NNs (+LST

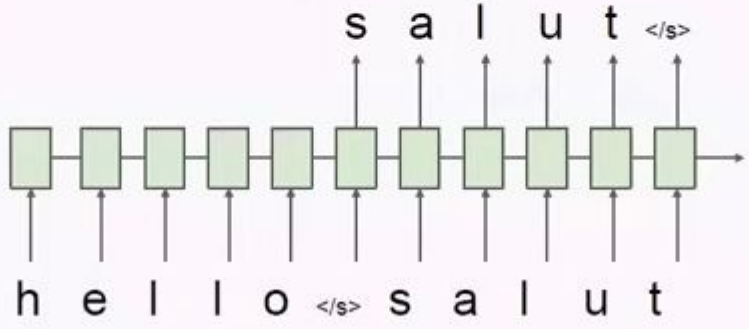


Speech

Deep Belief Nets (+non-DL)

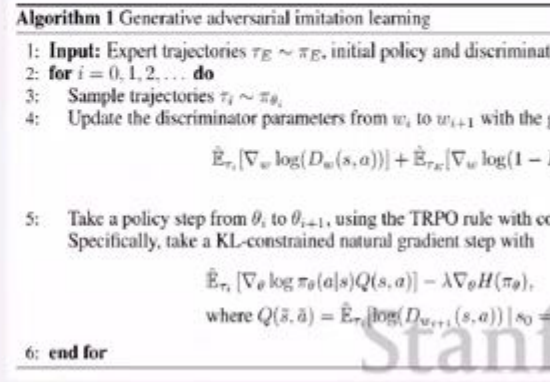
Translation

Seq2Seq



RL

BC/GAIL



Algorithm 1 Generative adversarial imitation learning

1: **Input:** Expert trajectories $\tau_E \sim \pi_E$, initial policy and discriminat
2: **for** $i = 0, 1, 2, \ldots$ **do**
3: Sample trajectories $\tau_i \sim \pi_{\theta_i}$
4: Update the discriminator parameters from w_i to w_{i+1} with the

$$\hat{\mathbb{E}}_{\tau_i}[\nabla_w \log(D_w(s,a))] + \hat{\mathbb{E}}_{\tau_E}[\nabla_w \log(1 -$$

5: Take a policy step from θ_i to θ_{i+1}, using the TRPO rule with c
Specifically, take a KL-constrained natural gradient step with

$$\hat{\mathbb{E}}_{\tau_i}[\nabla_\theta \log \pi_\theta(a|s) Q(s,a)] - \lambda \nabla_\theta H(\pi_\theta),$$

where $Q(\bar{s}, \bar{a}) = \hat{\mathbb{E}}_{\tau_i}[\log(D_{w_{i+1}}(s,a)) \mid s_0 =$

6: **end for**

[1] CNN image CC-BY-SA by Aphex34 for Wikipedia https://commons.wikimedia.org/wiki/File:Typical_cnn.png
[2] RNN image CC-BY-SA by GChe for Wikipedia https://commons.wikimedia.org/wiki/File:The_LSTM_Cell.svg

Steven Feng

Stani

slid

2017

Transformers

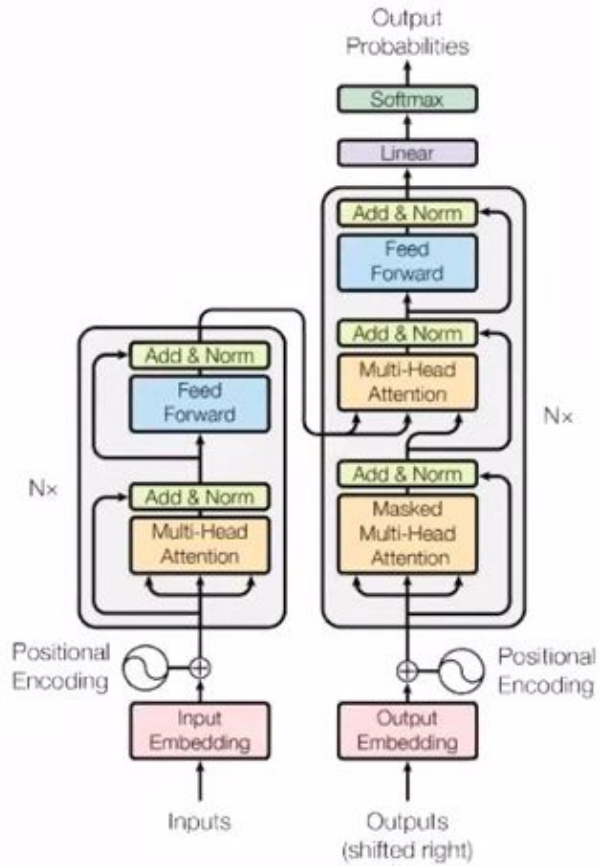
Usman Nazir

2017

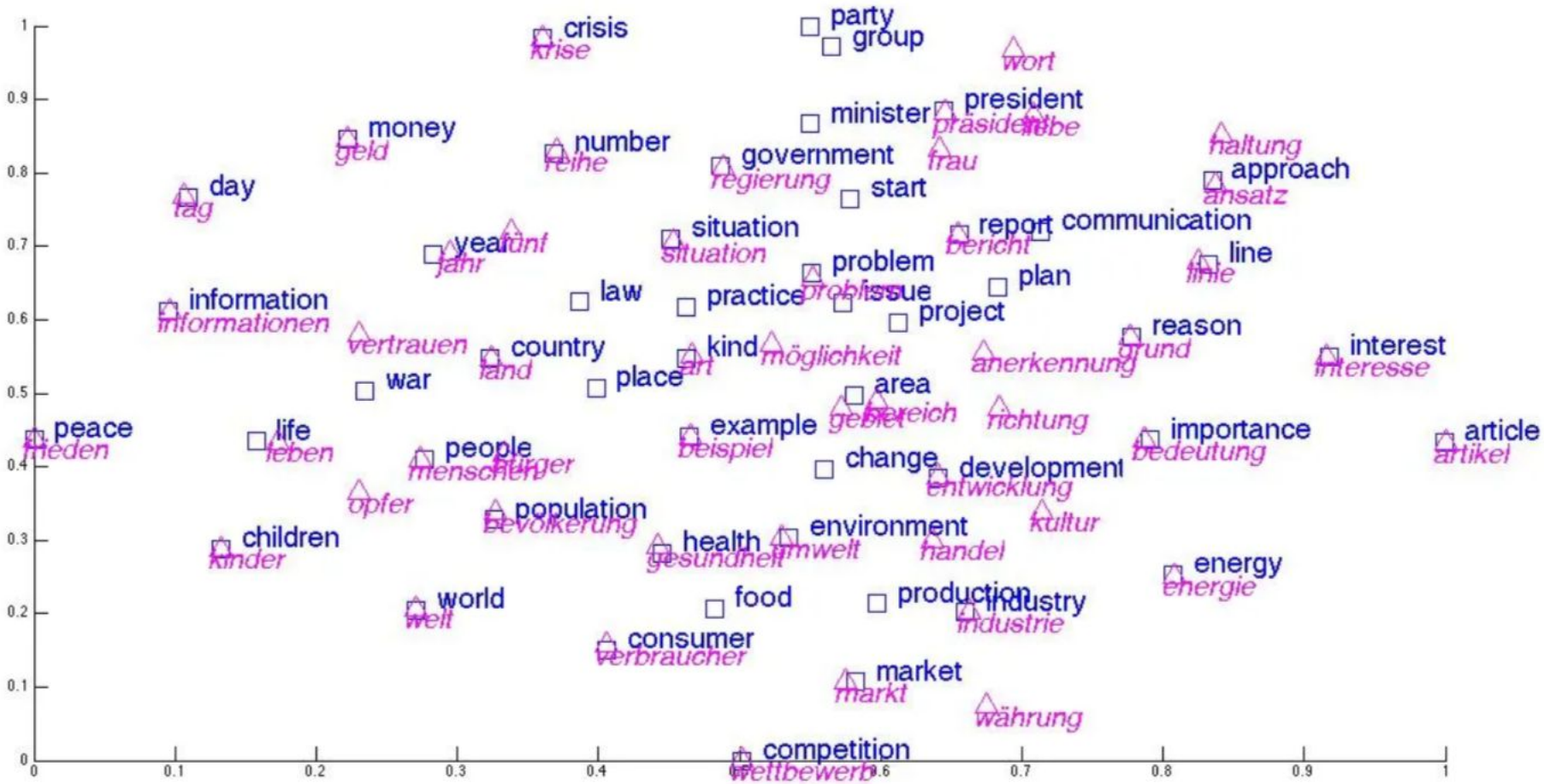
Attention Is All You Need

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin

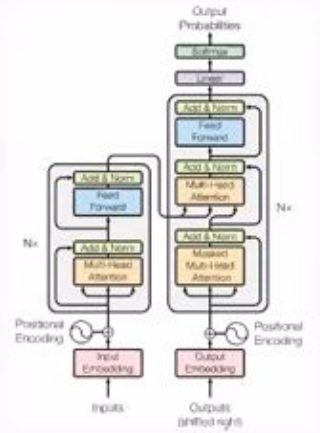
A very unassuming looking machine translation paper…



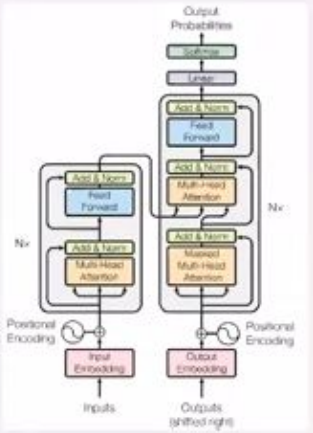
Embedding or dimensionality reduction is more efficient way of representation



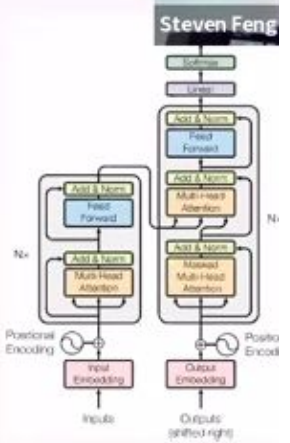
Computer Vision



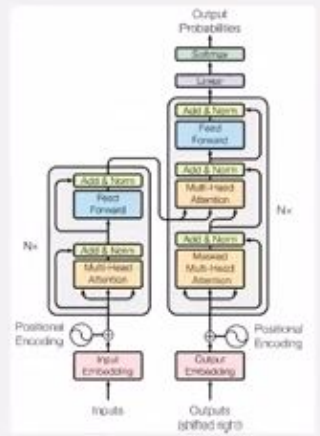
Natural Lang. Proc.



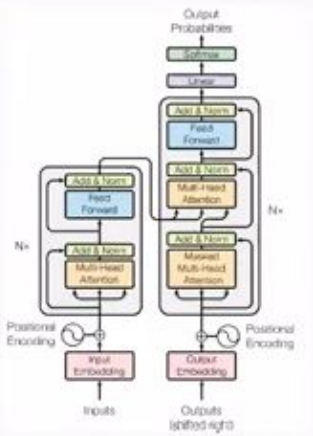
Reinf. L



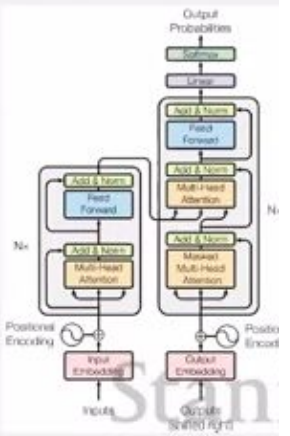
Speech



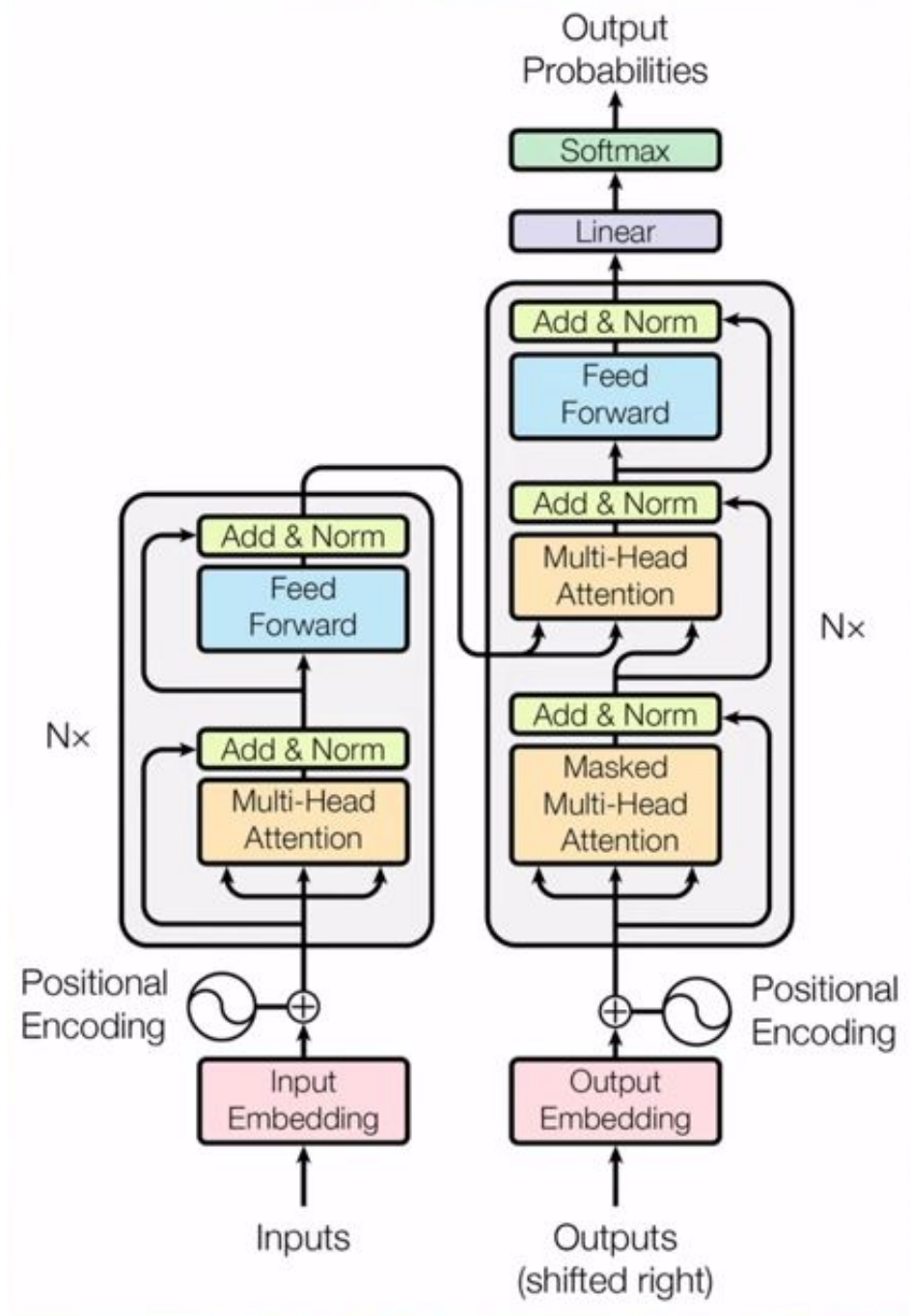
Translation

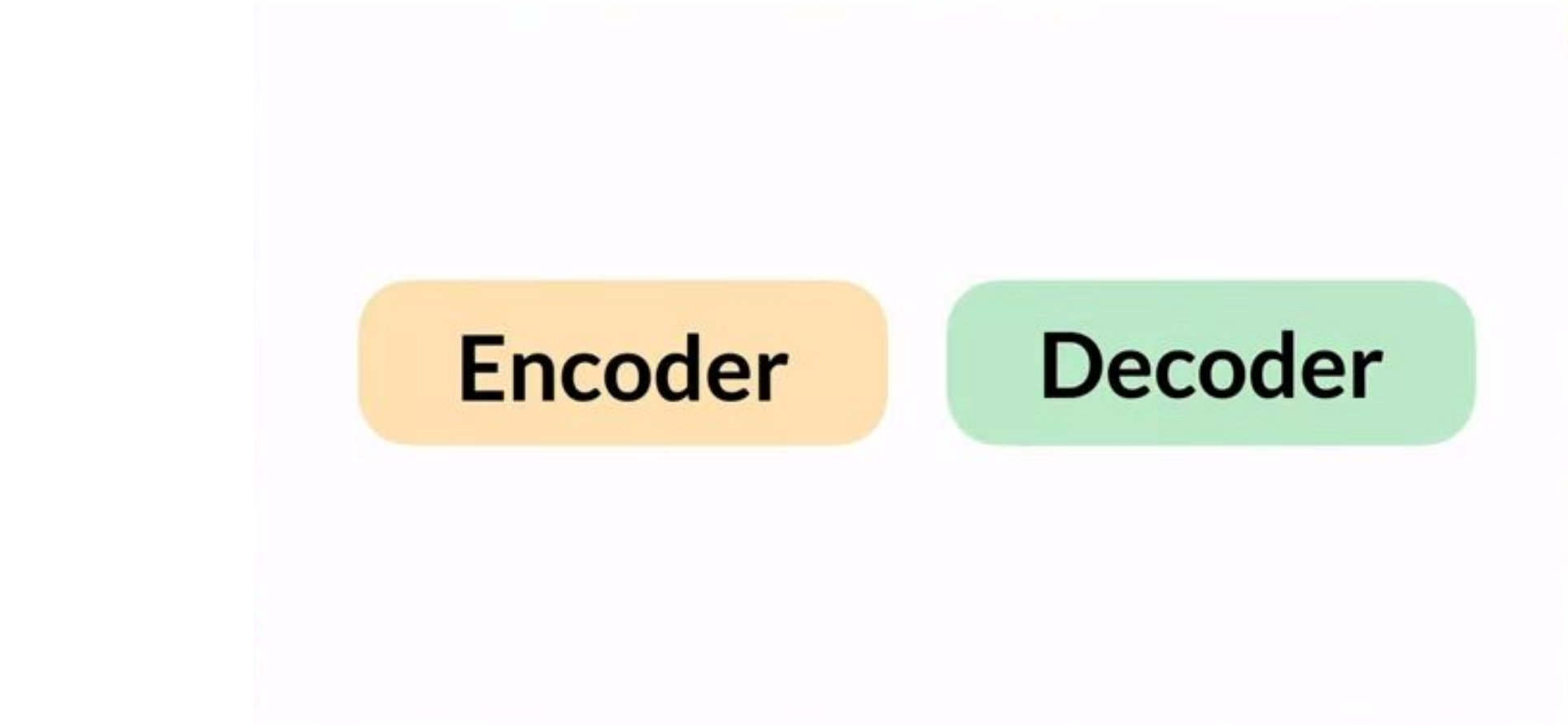


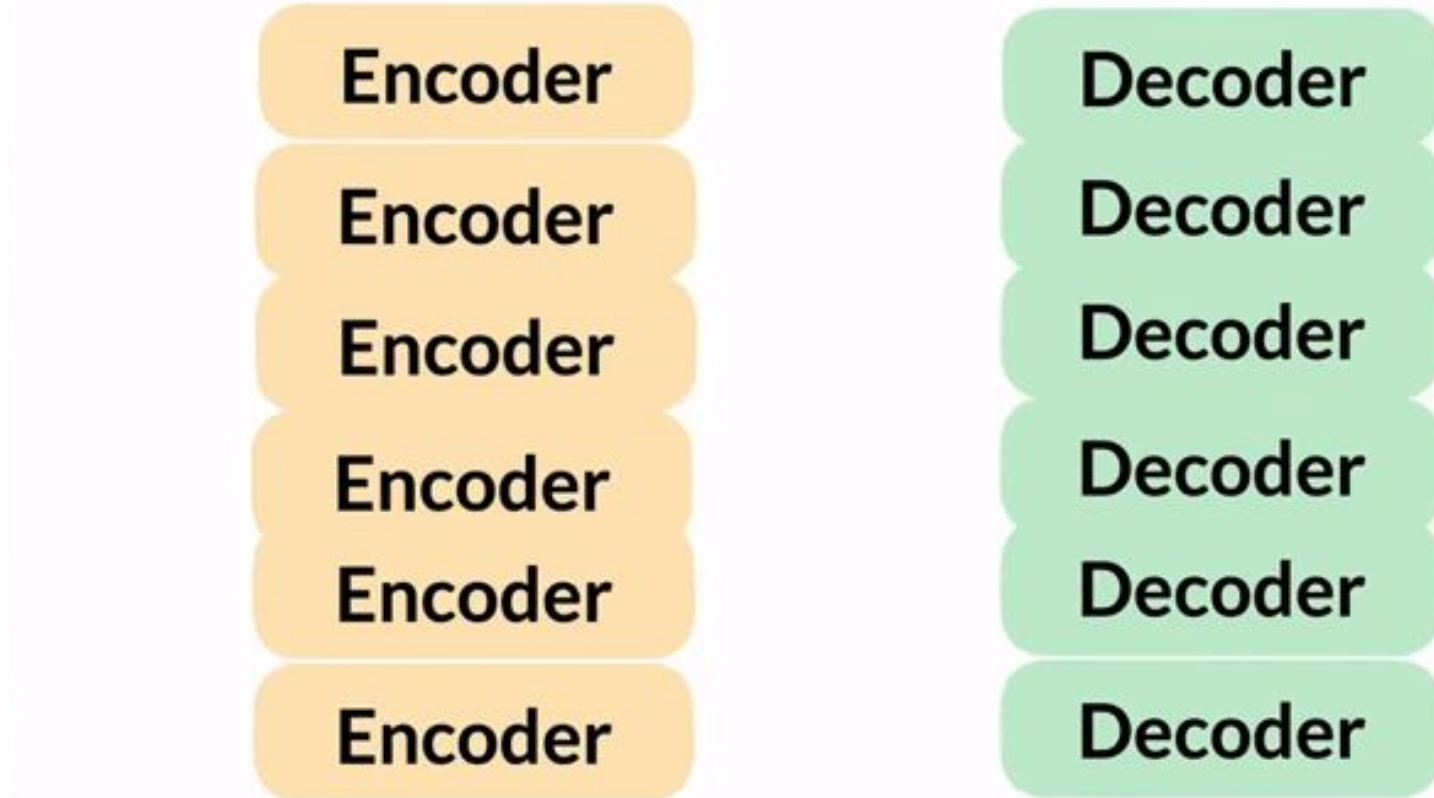
Graphs/Sci

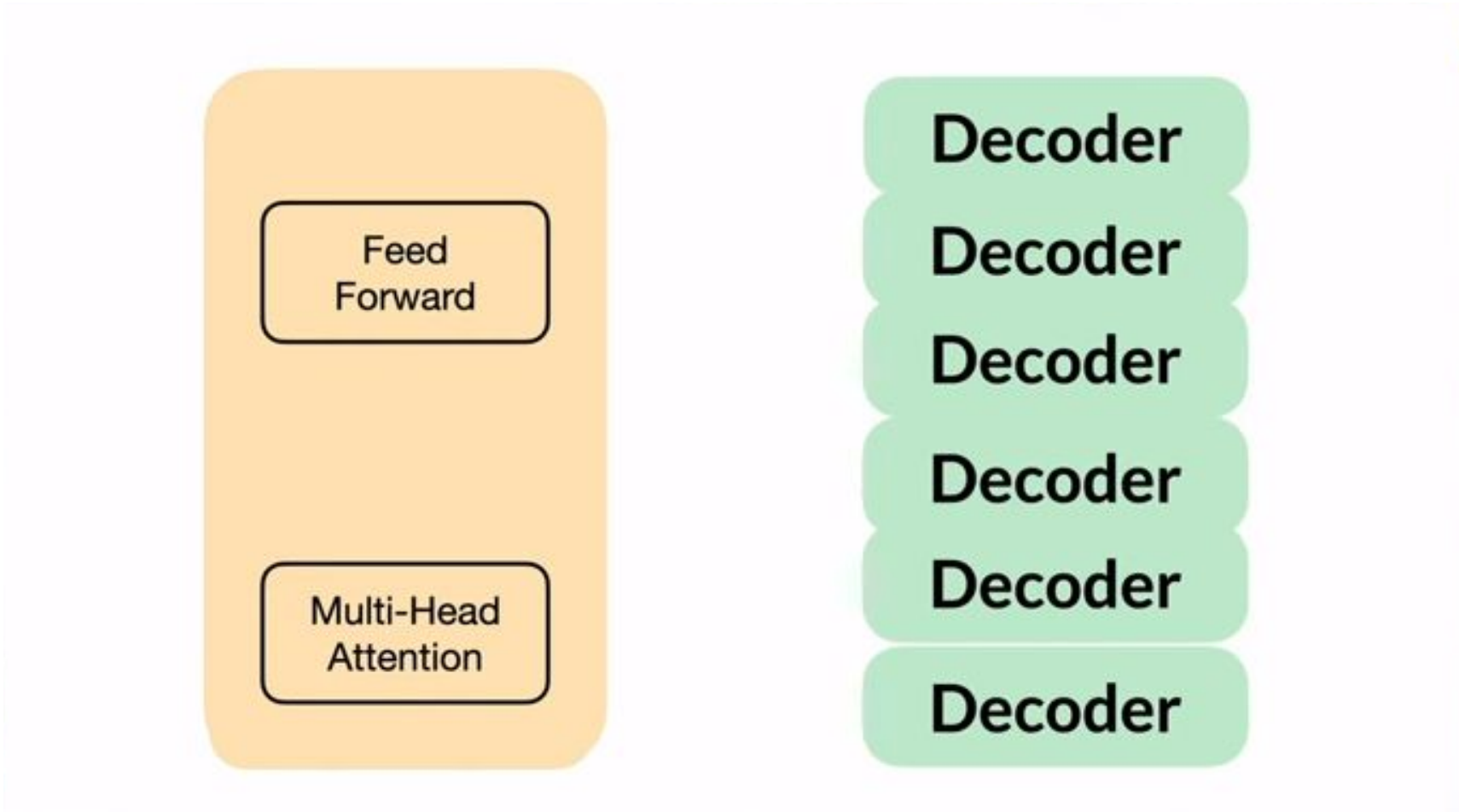


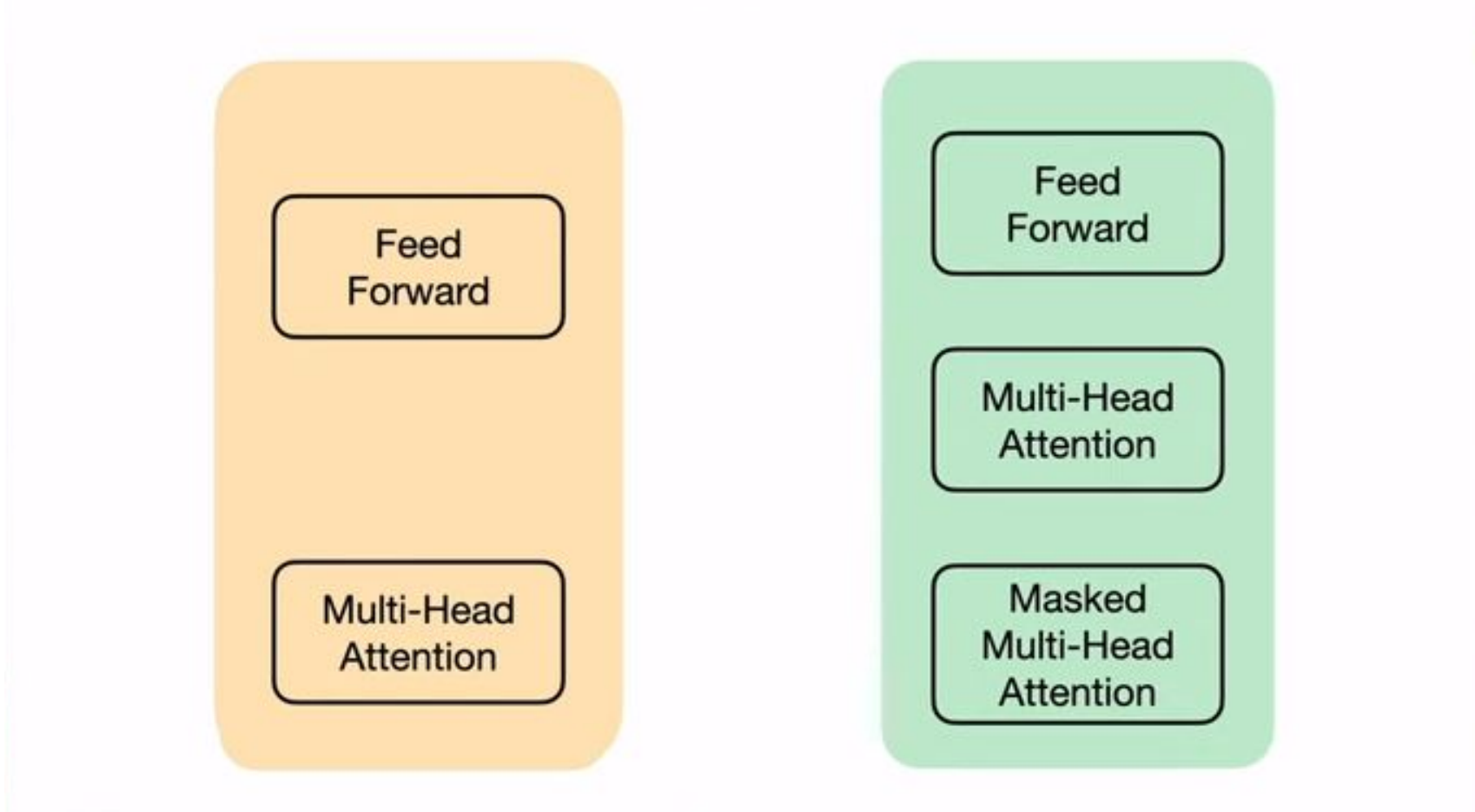
Transformer image source: "Attention Is All You Need" paper

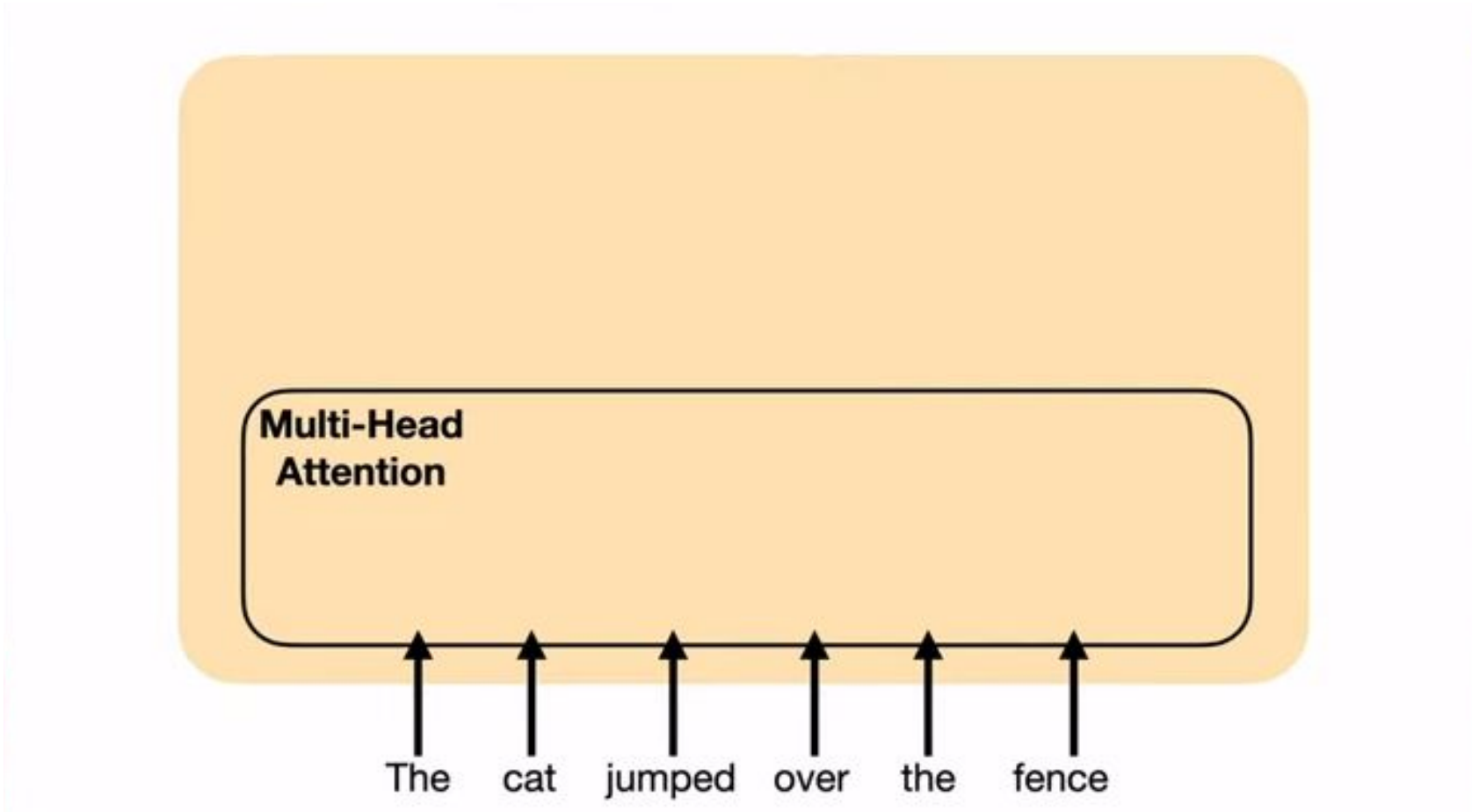

Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
Nx
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Nx
Positional Encoding
Positional Encoding
Input Embedding
Output Embedding
Inputs
Outputs (shifted right)

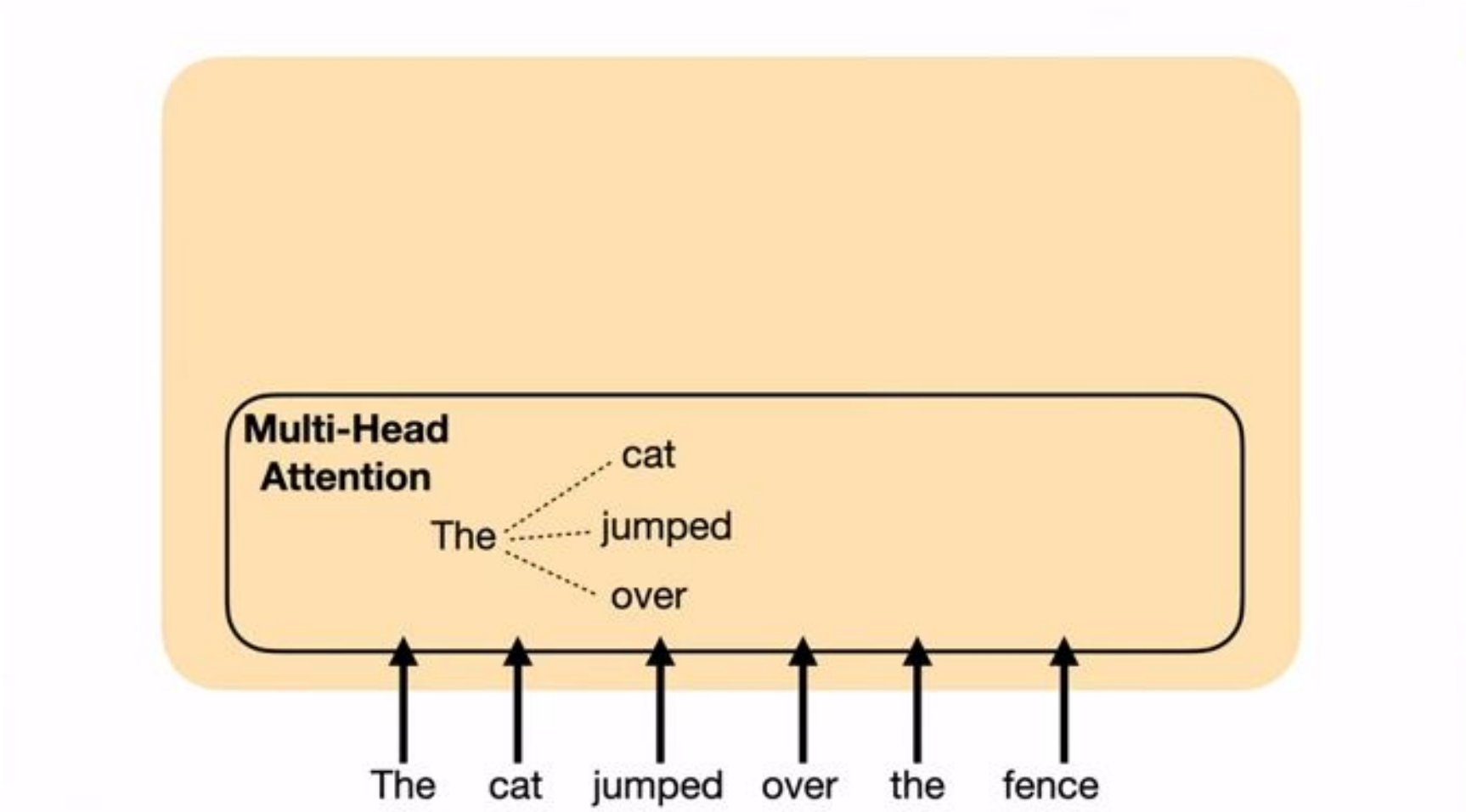

Encoder
Decoder

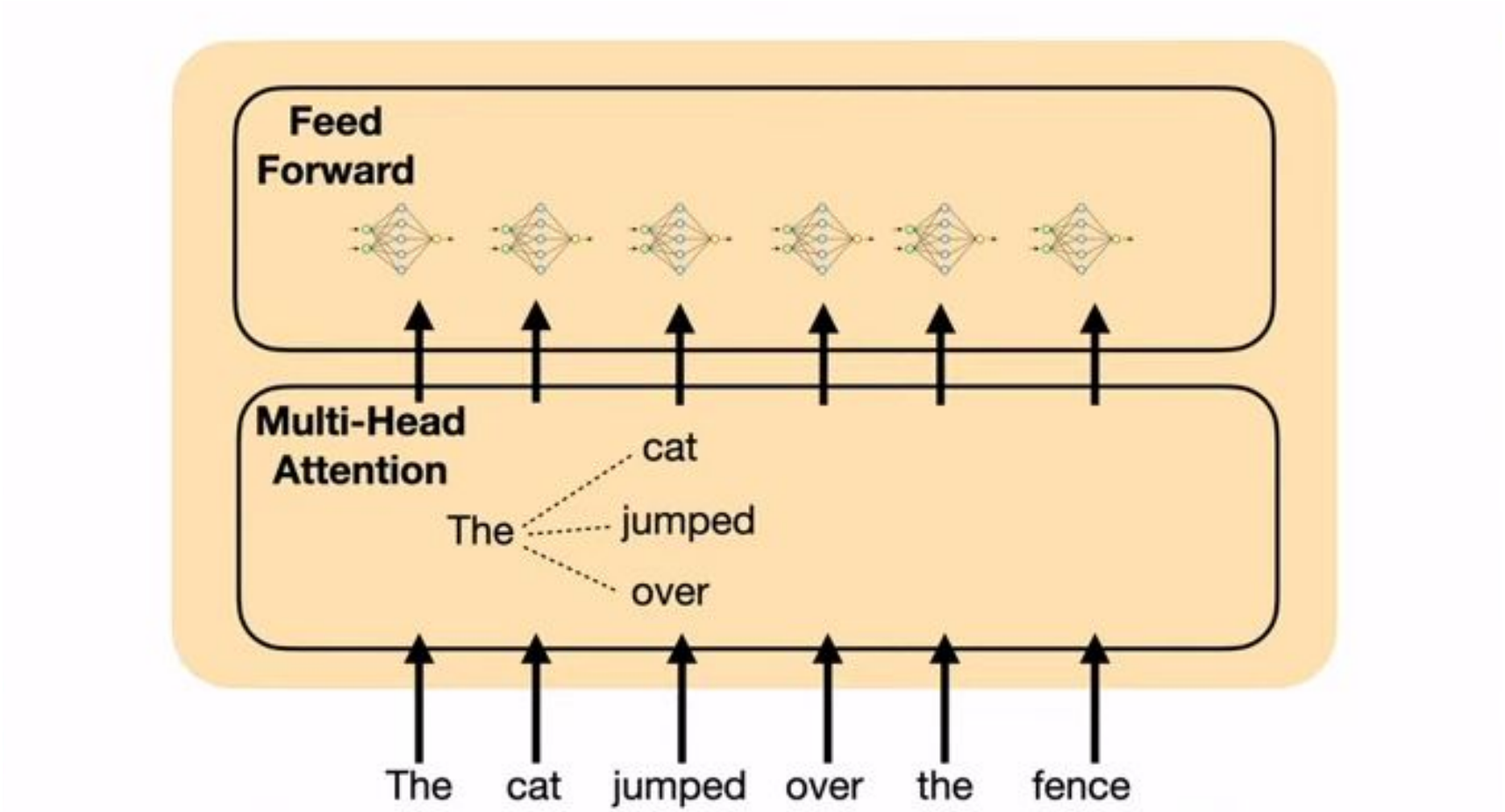

Encoder
Encoder
Encoder
Encoder
Encoder
Encoder
Decoder
Decoder
Decoder
Decoder
Decoder
Decoder

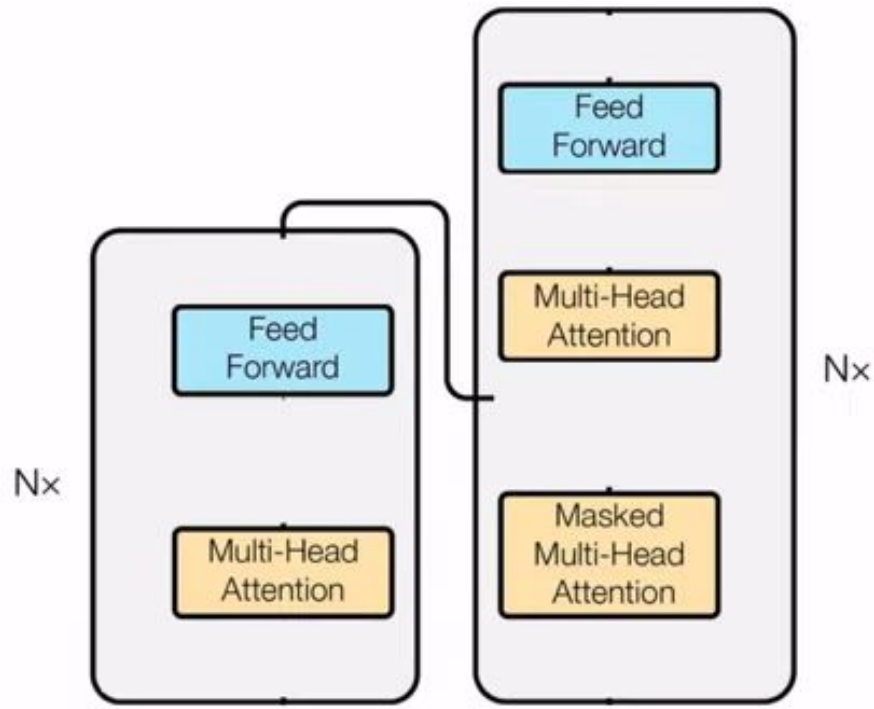

Feed Forward
Multi-Head Attention
Decoder
Decoder
Decoder
Decoder
Decoder
Decoder

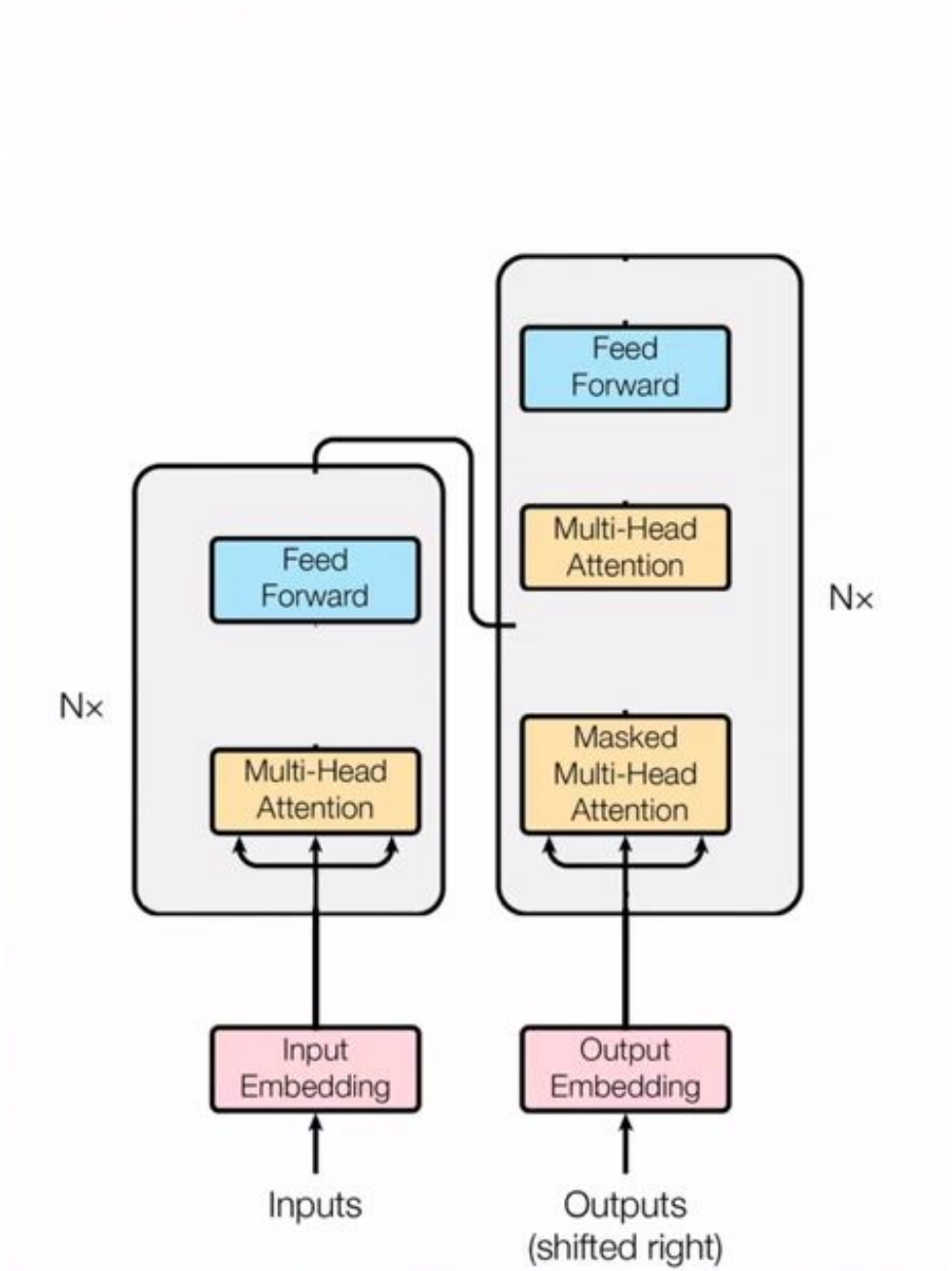

Feed Forward
Multi-Head Attention
Feed Forward
Multi-Head Attention
Masked Multi-Head Attention

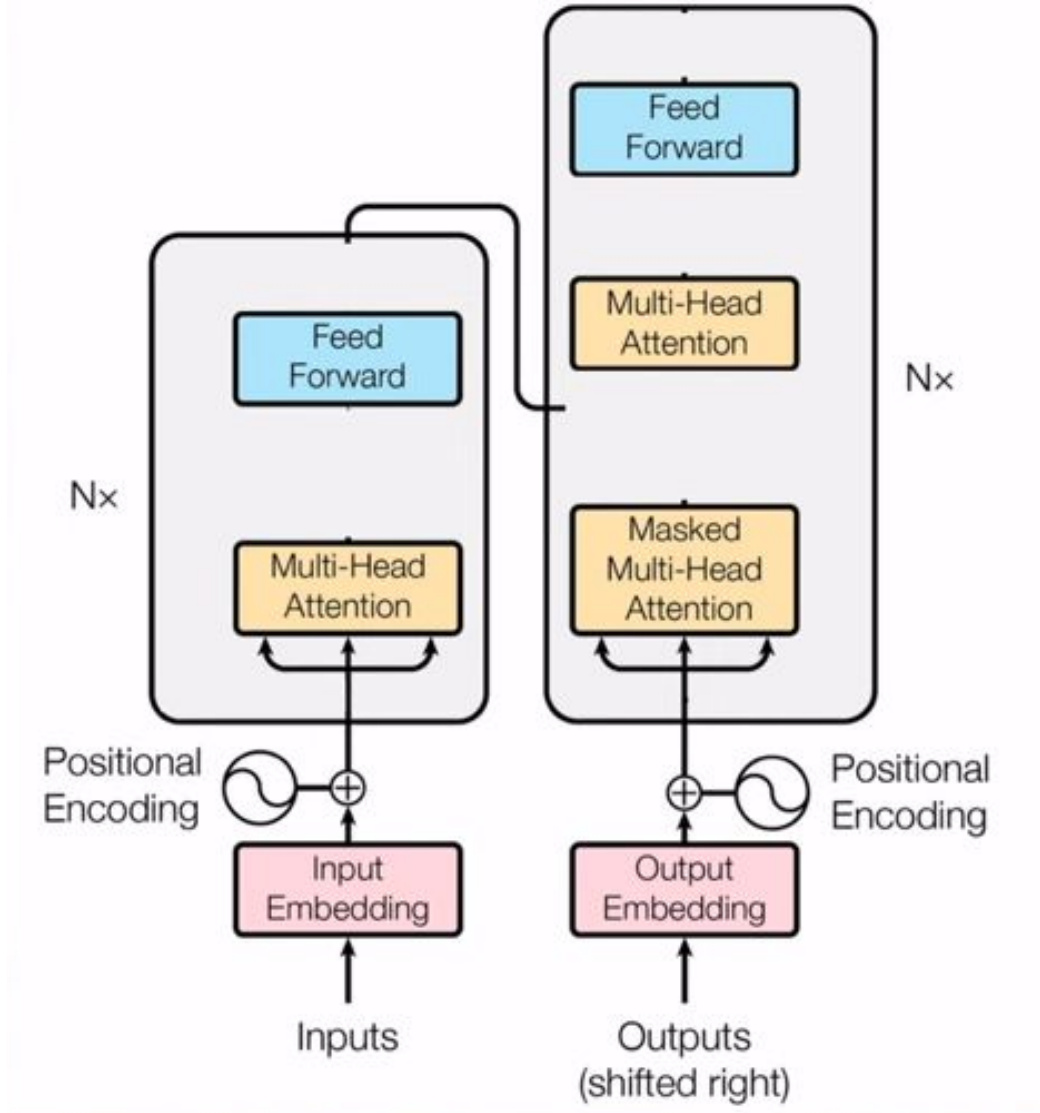

Multi-Head
Attention
The
cat
jumped
over
the
fence

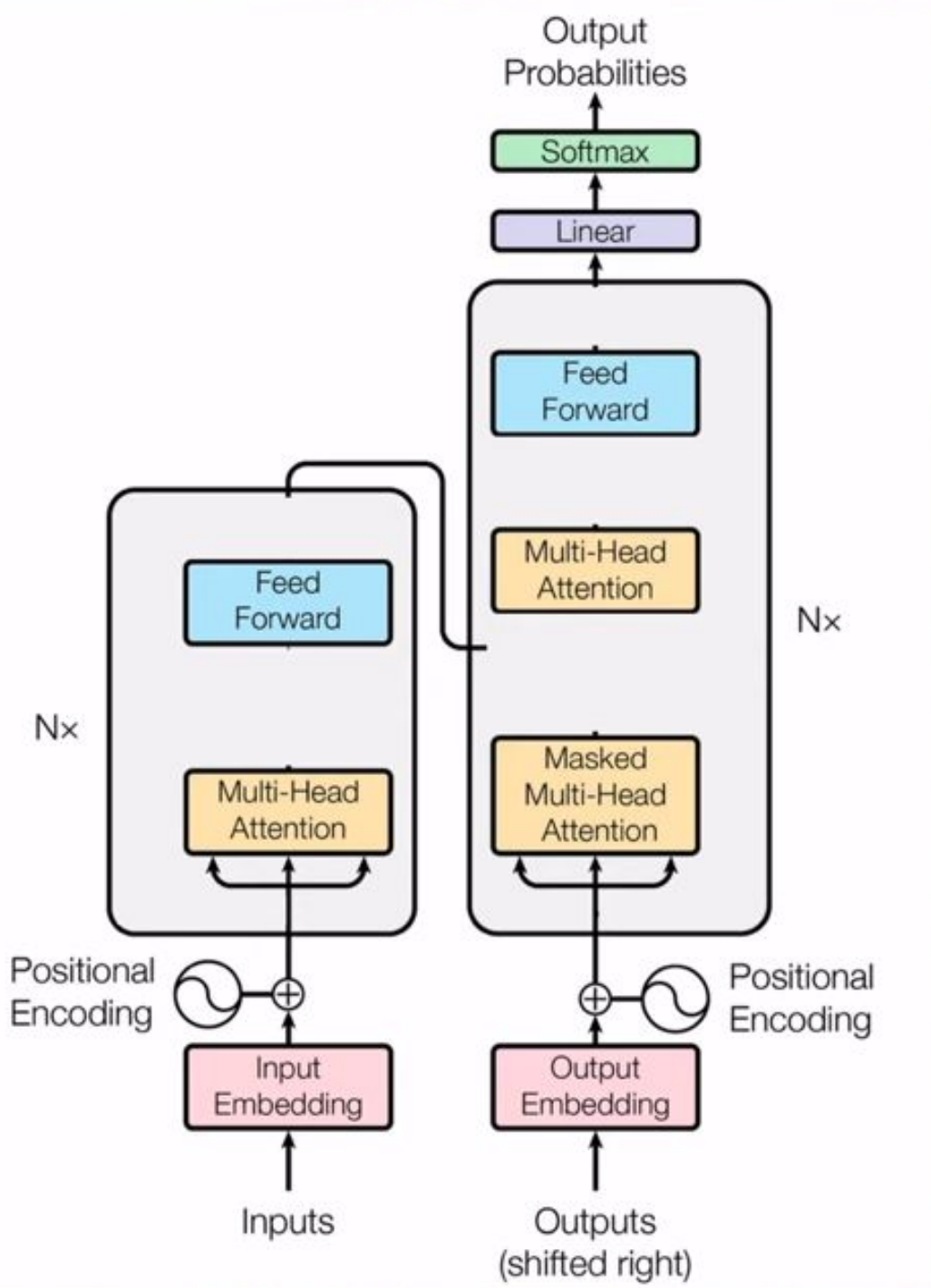

Multi-Head
Attention
The
cat
jumped
over
The
cat
jumped
over
the
fence

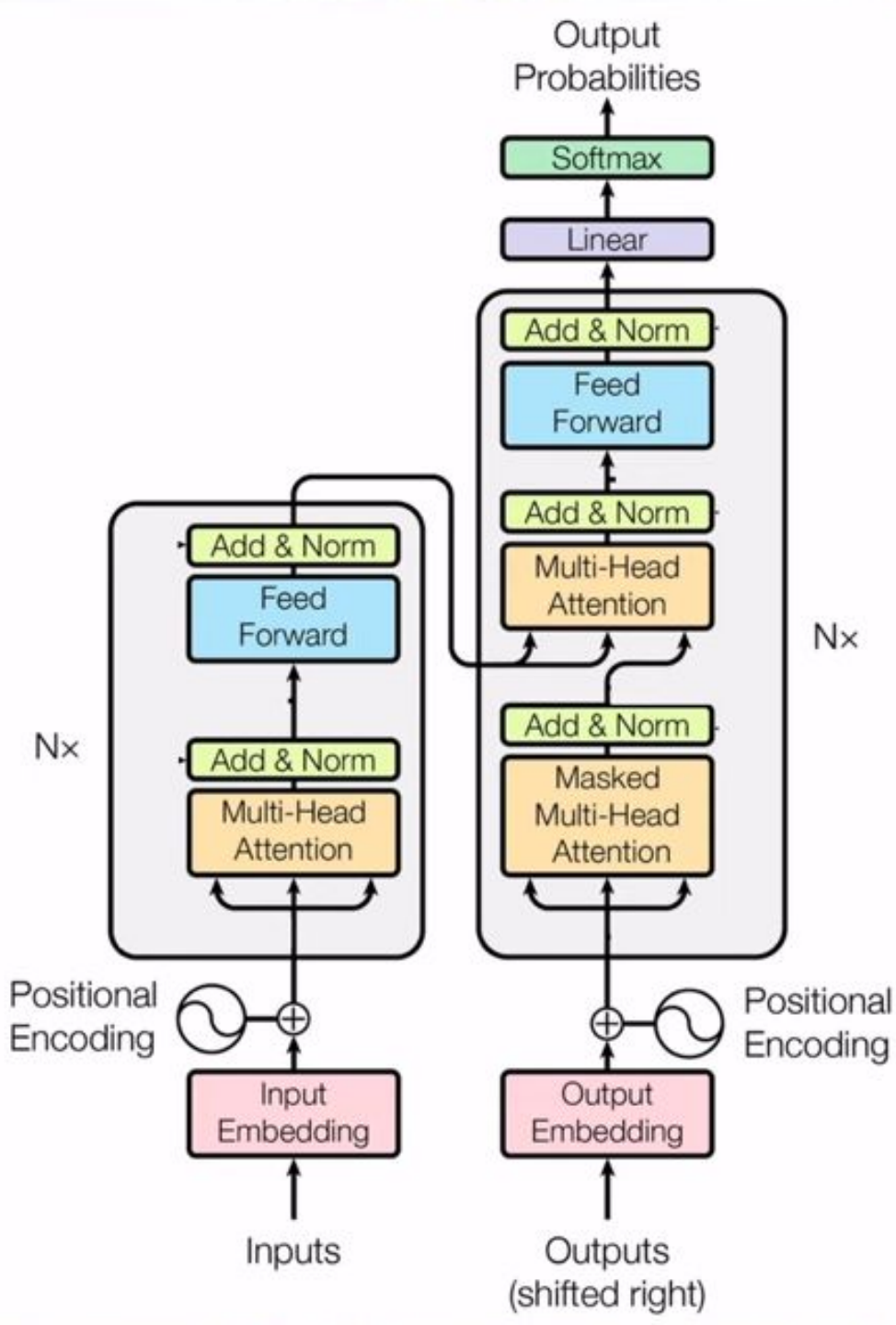

Feed
Forward
Multi-Head
Attention
cat
The
jumped
over
The
cat
jumped
over
the
fence

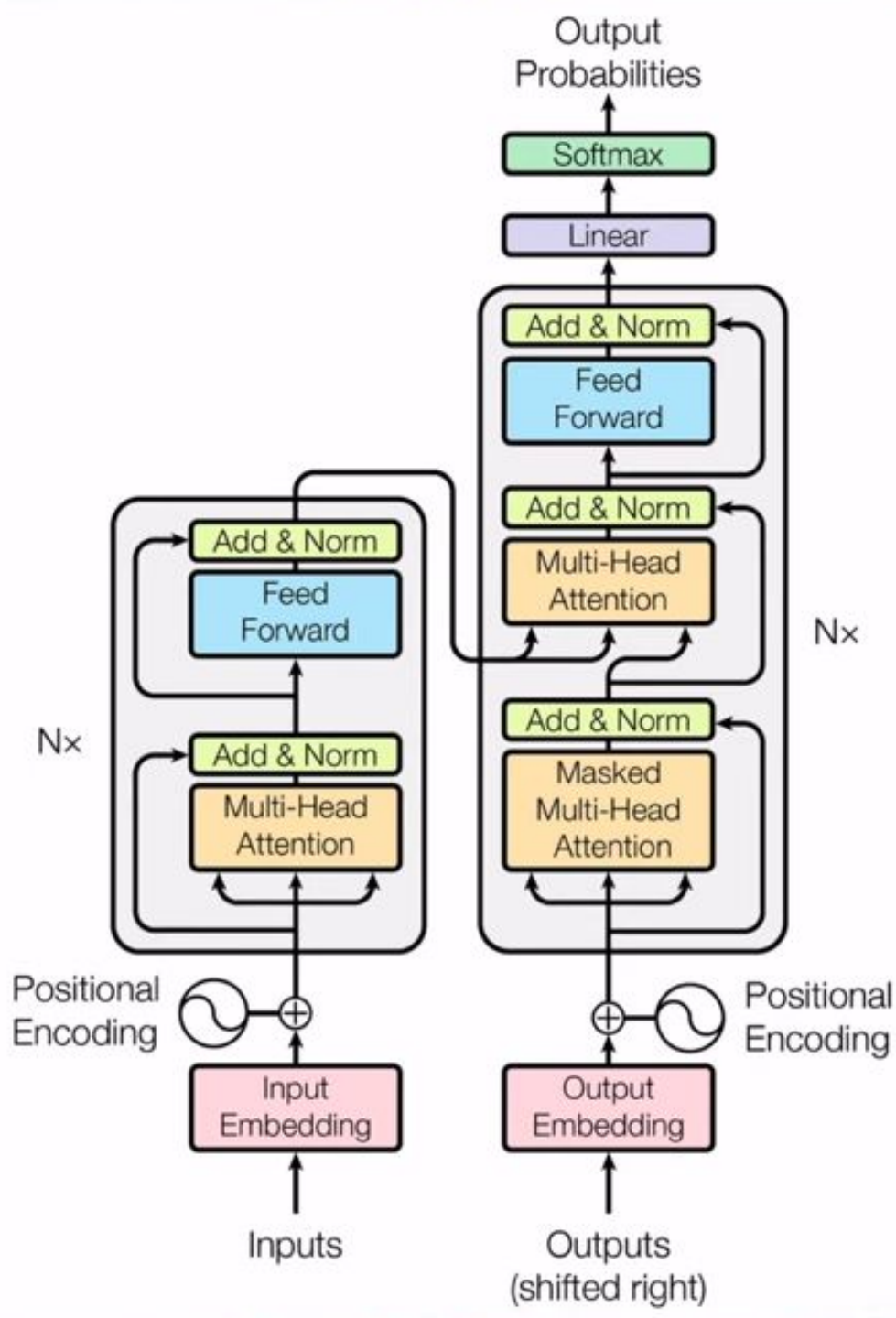

Feed
Forward
Multi-Head
Attention
Masked
Multi-Head
Attention
Nx
Feed
Forward
Multi-Head
Attention
Nx

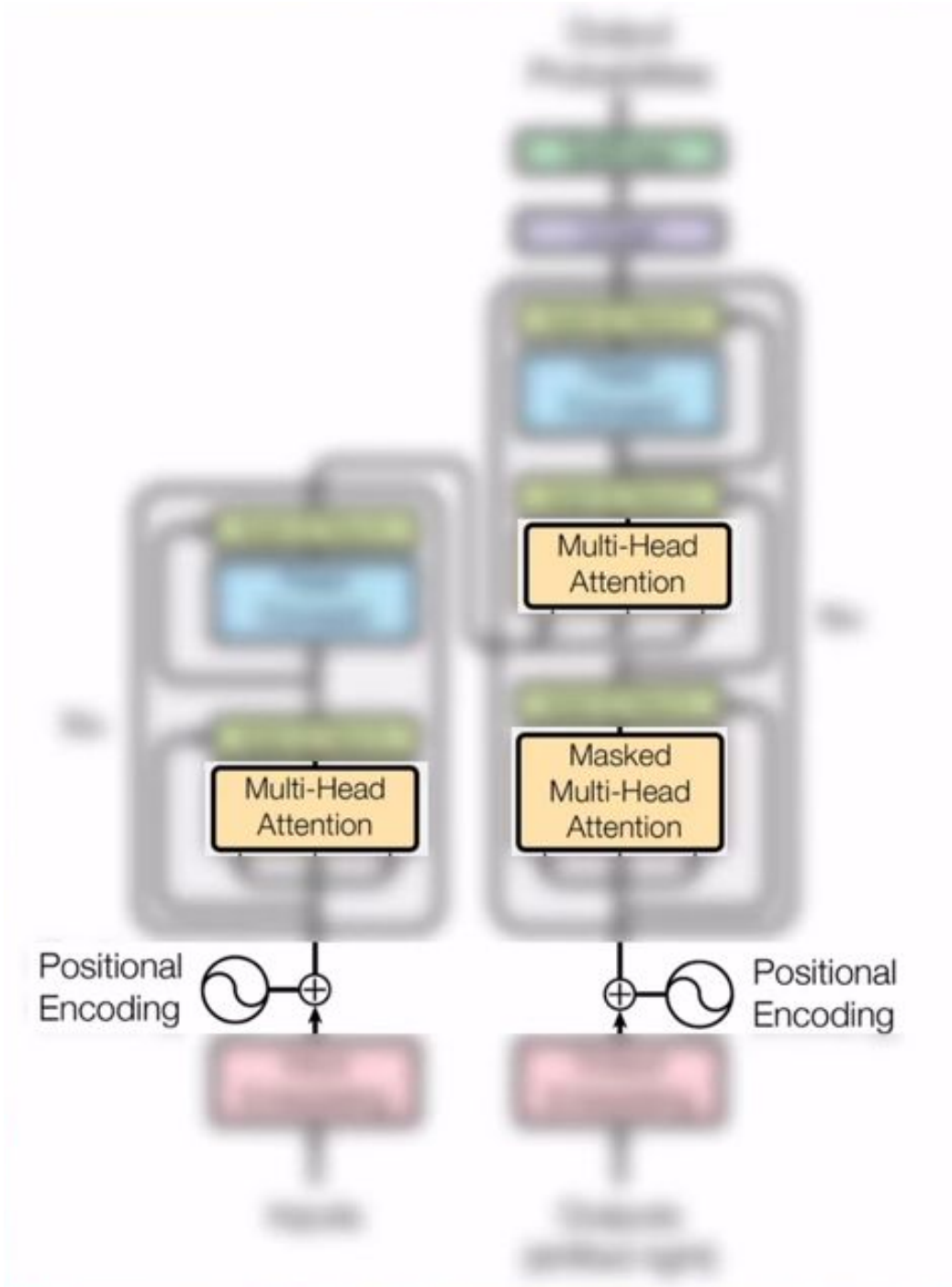

Feed Forward
Multi-Head Attention
Masked Multi-Head Attention
Nx
Feed Forward
Multi-Head Attention
Nx
Input Embedding
Output Embedding
Inputs
Outputs (shifted right)


Feed Forward
Multi-Head Attention
Masked Multi-Head Attention
Nx
Feed Forward
Multi-Head Attention
Nx
Positional Encoding
Positional Encoding
Input Embedding
Output Embedding
Inputs
Outputs (shifted right)

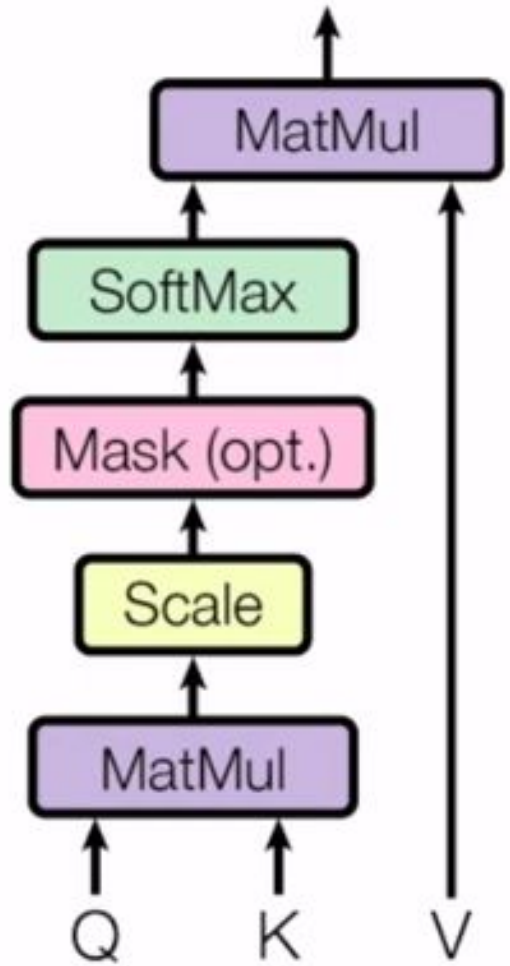

Output Probabilities
Softmax
Linear
Feed Forward
Multi-Head Attention
Masked Multi-Head Attention
Nx
Feed Forward
Multi-Head Attention
Nx
Positional Encoding
Positional Encoding
Input Embedding
Output Embedding
Inputs
Outputs (shifted right)

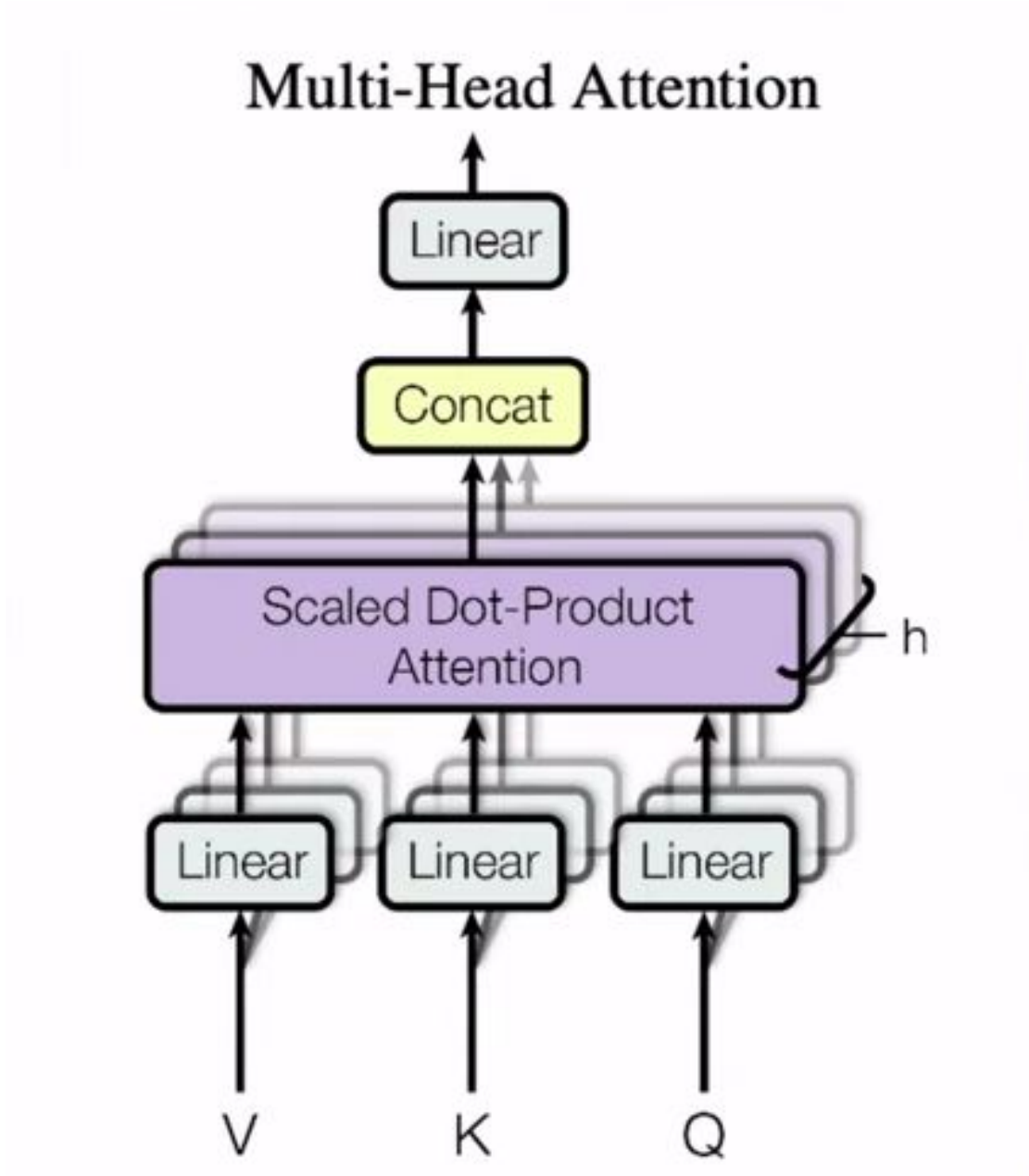

Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
Nx
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Nx
Positional Encoding
Positional Encoding
Input Embedding
Output Embedding
Inputs
Outputs (shifted right)


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)

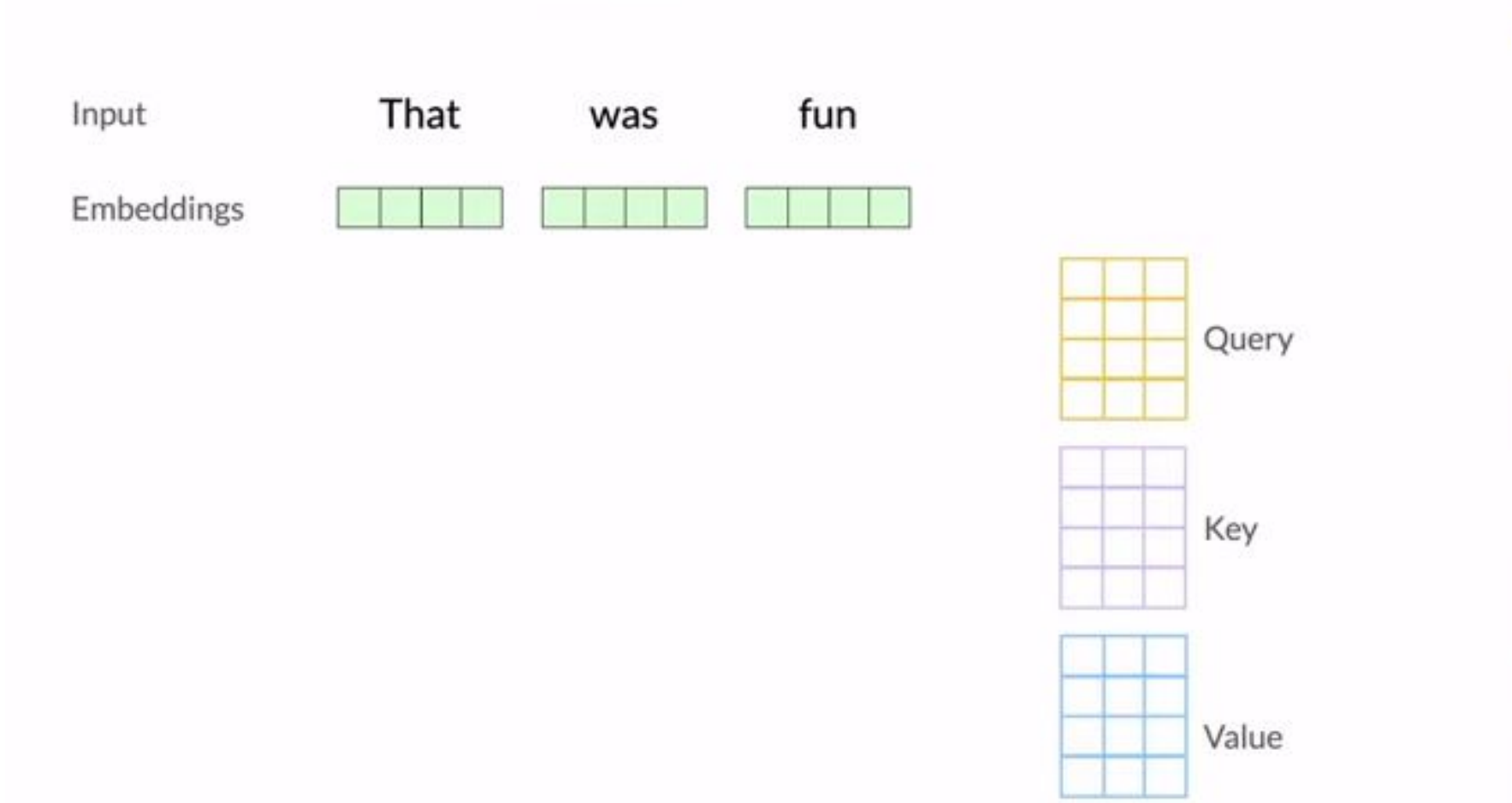

Multi-Head Attention
Masked Multi-Head Attention
Multi-Head Attention
Positional Encoding
Positional Encoding

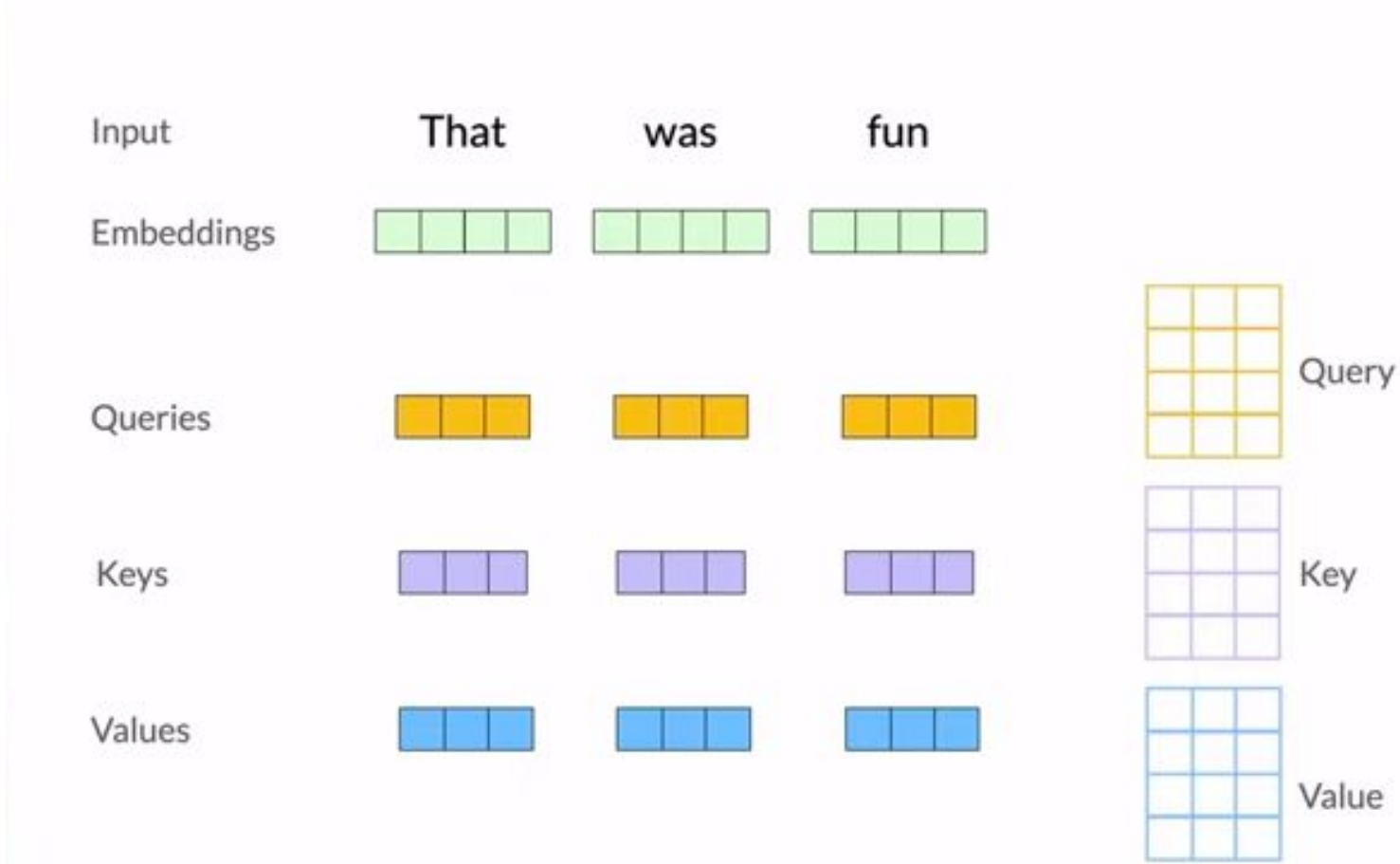
Scaled Dot-Product Attention

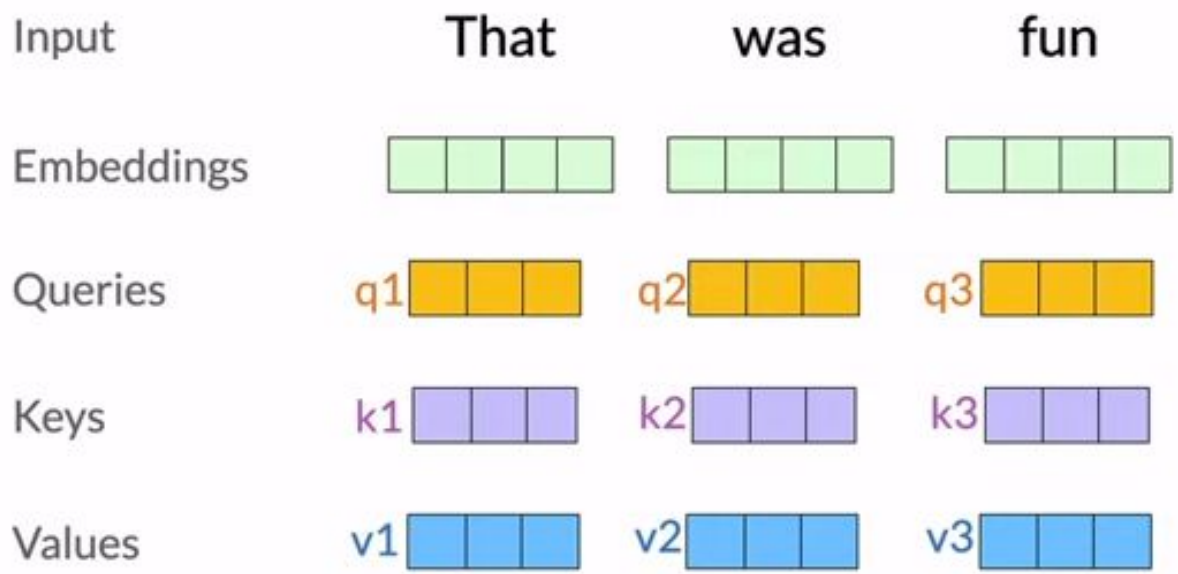


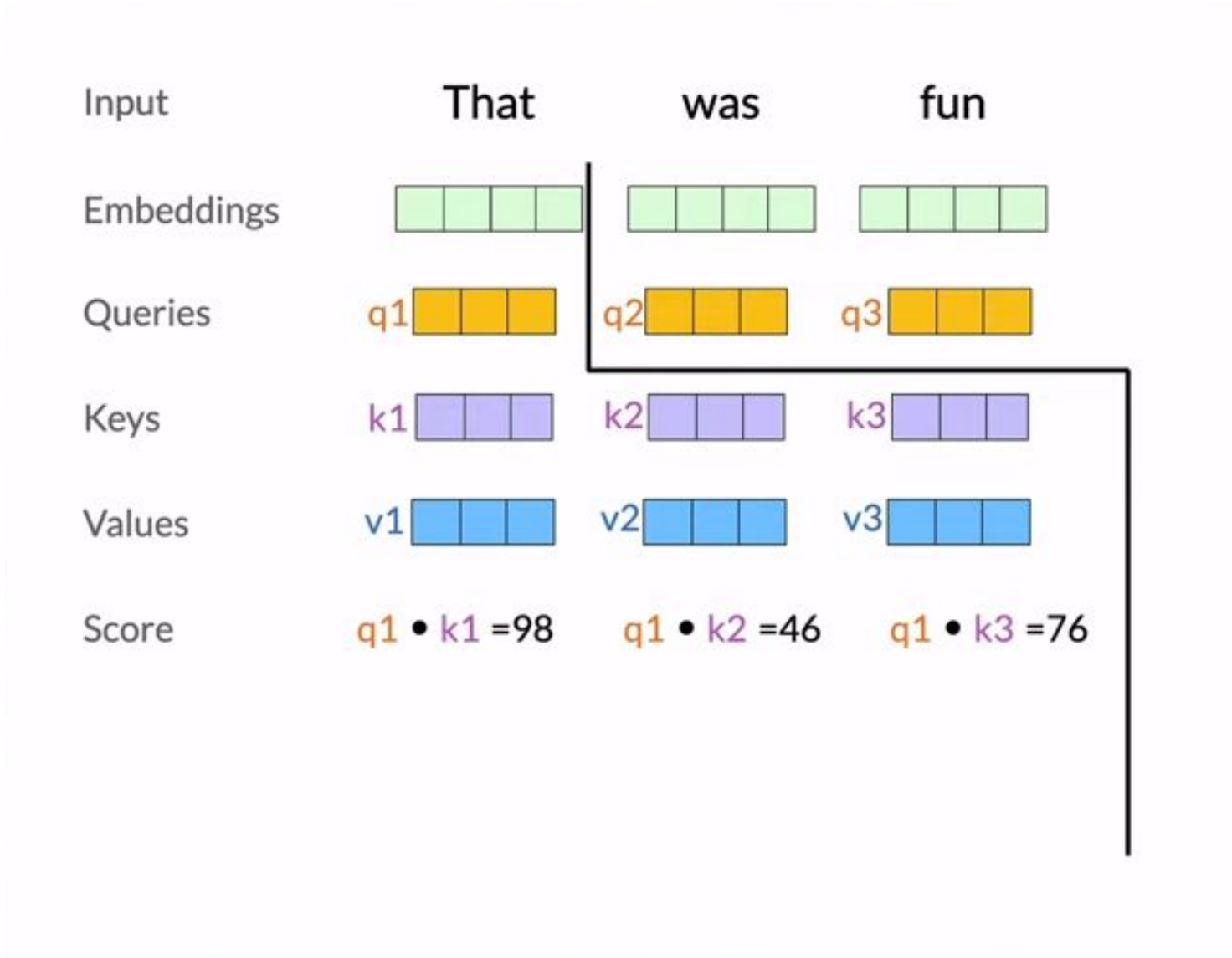

Multi-Head Attention
Linear
Concat
Scaled Dot-Product Attention
h
Linear
Linear
Linear
V
K
Q

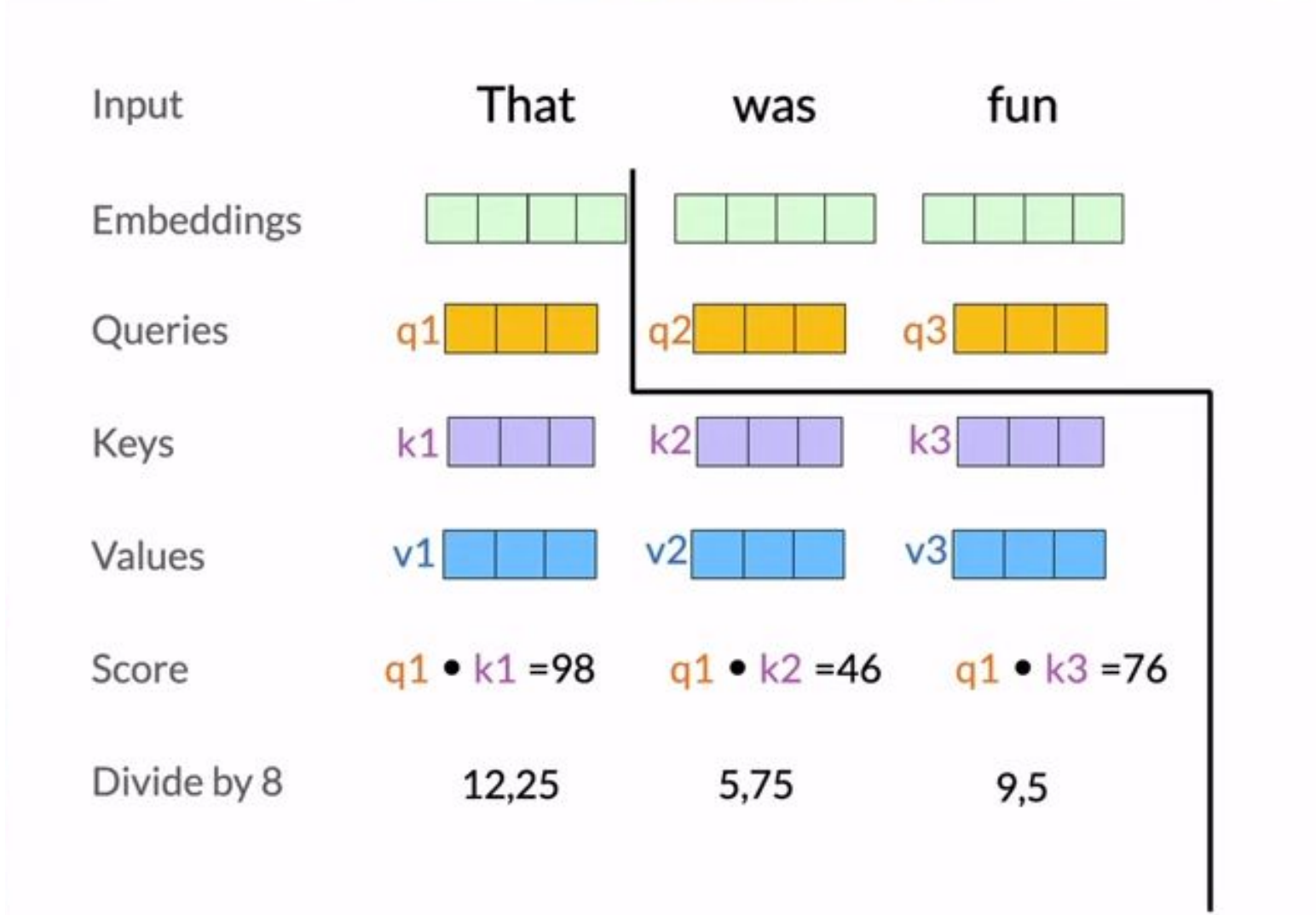

Input
That
was
fun
Embeddings

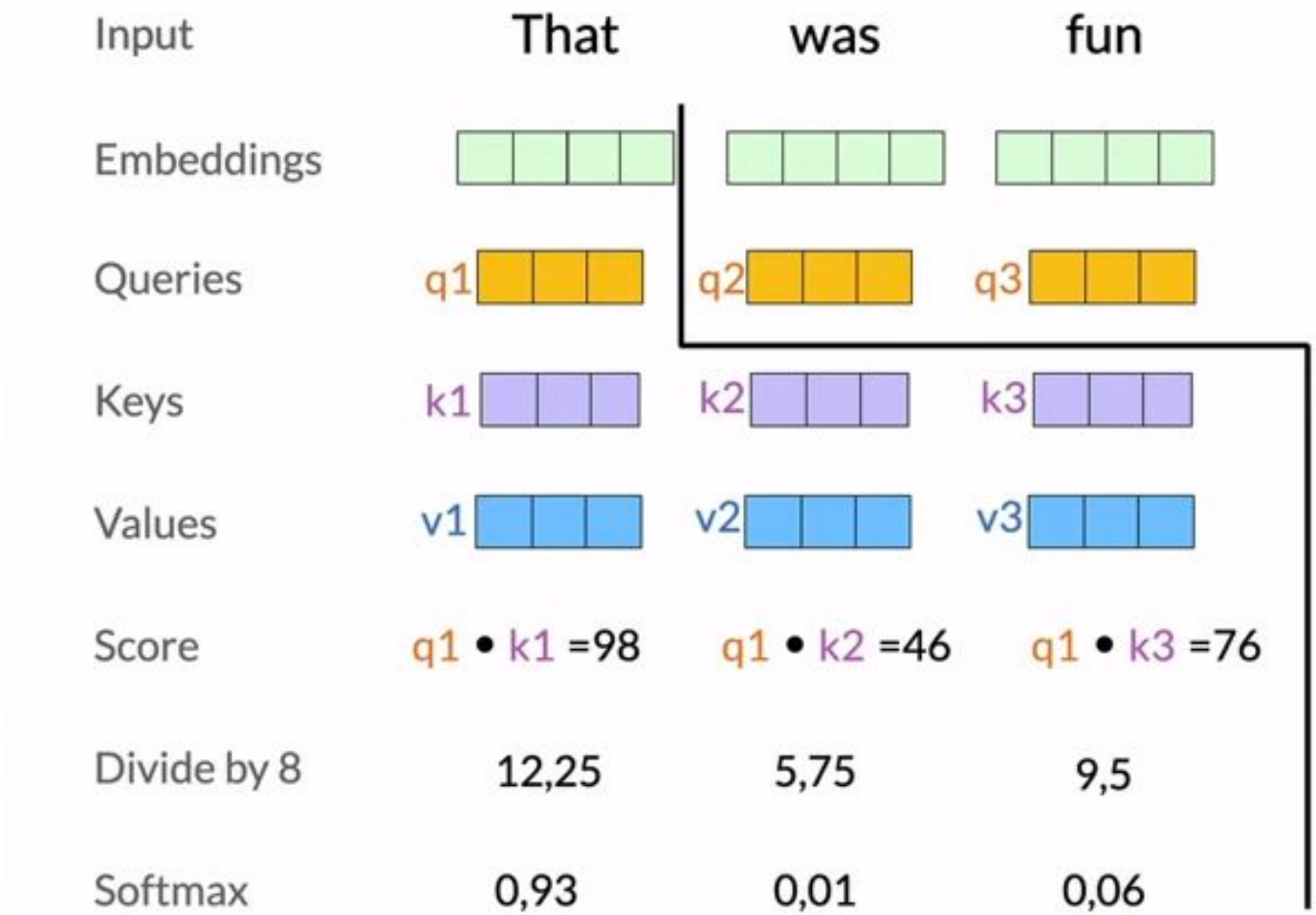

Input
That
was
fun
Embeddings
Query
Key
Value

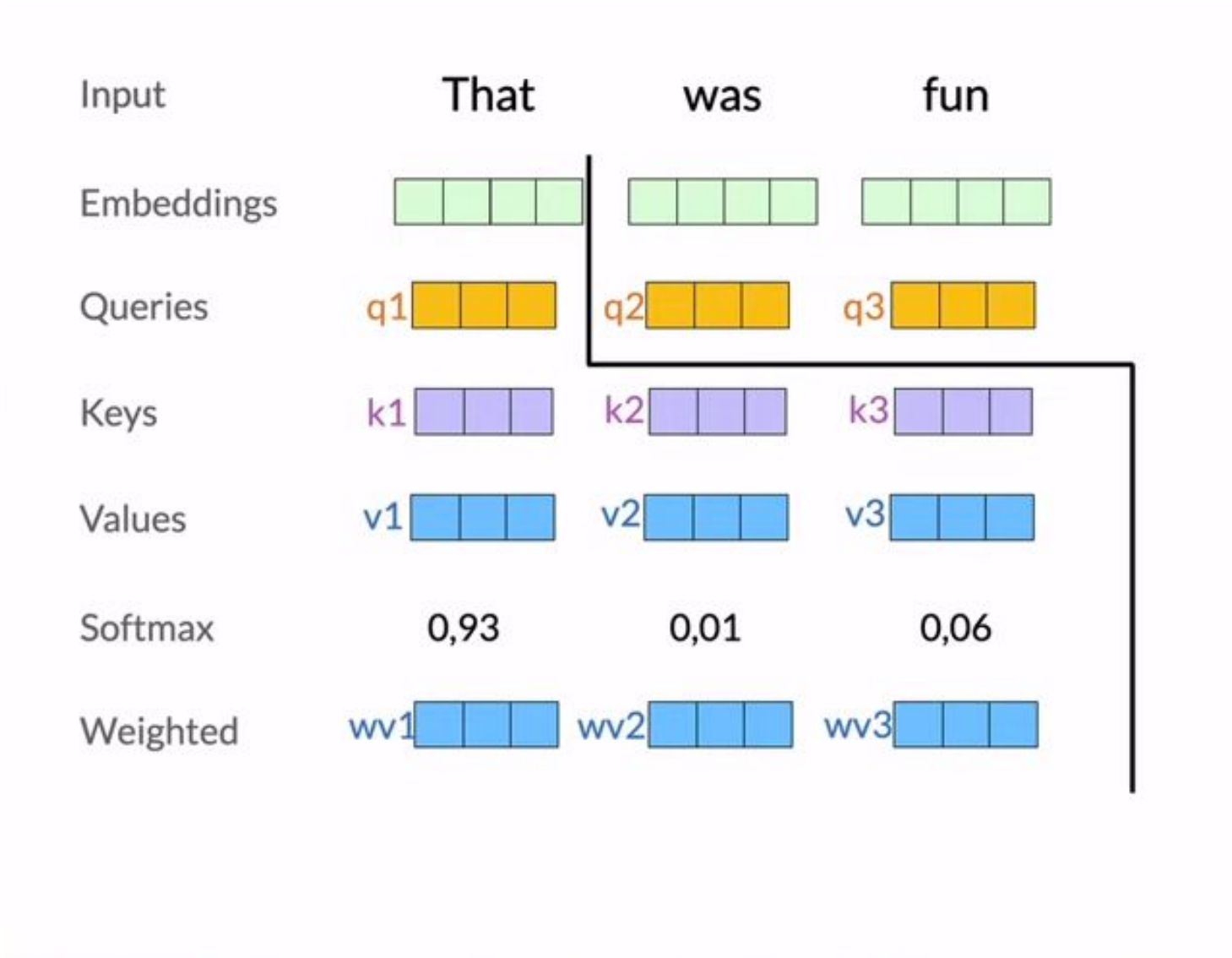

Input
That
was
fun
Embeddings
Queries
Keys
Values
Query
Key
Value

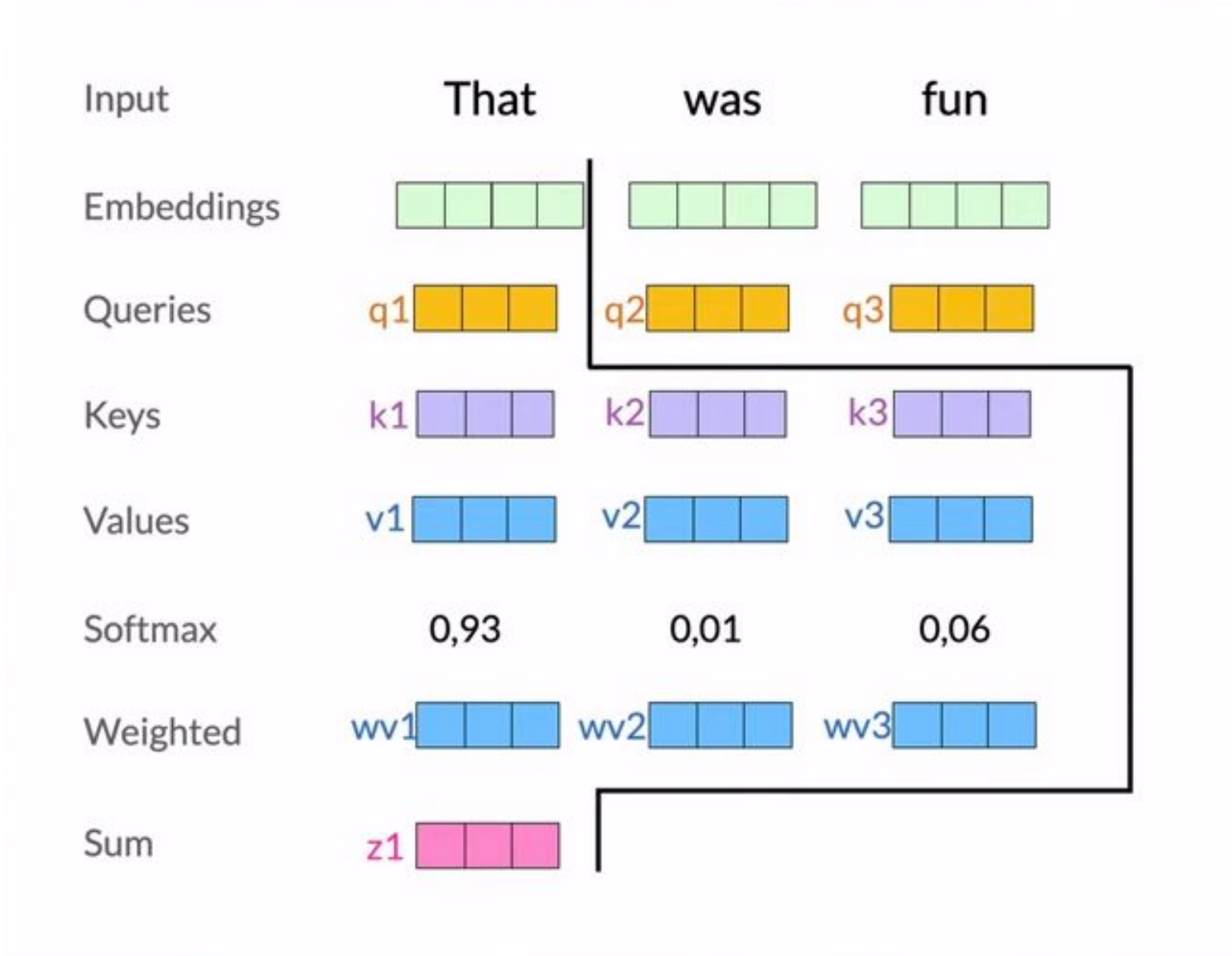

Input
That
was
fun
Embeddings
Queries
q1
q2
q3
Keys
k1
k2
k3
Values
v1
v2
v3

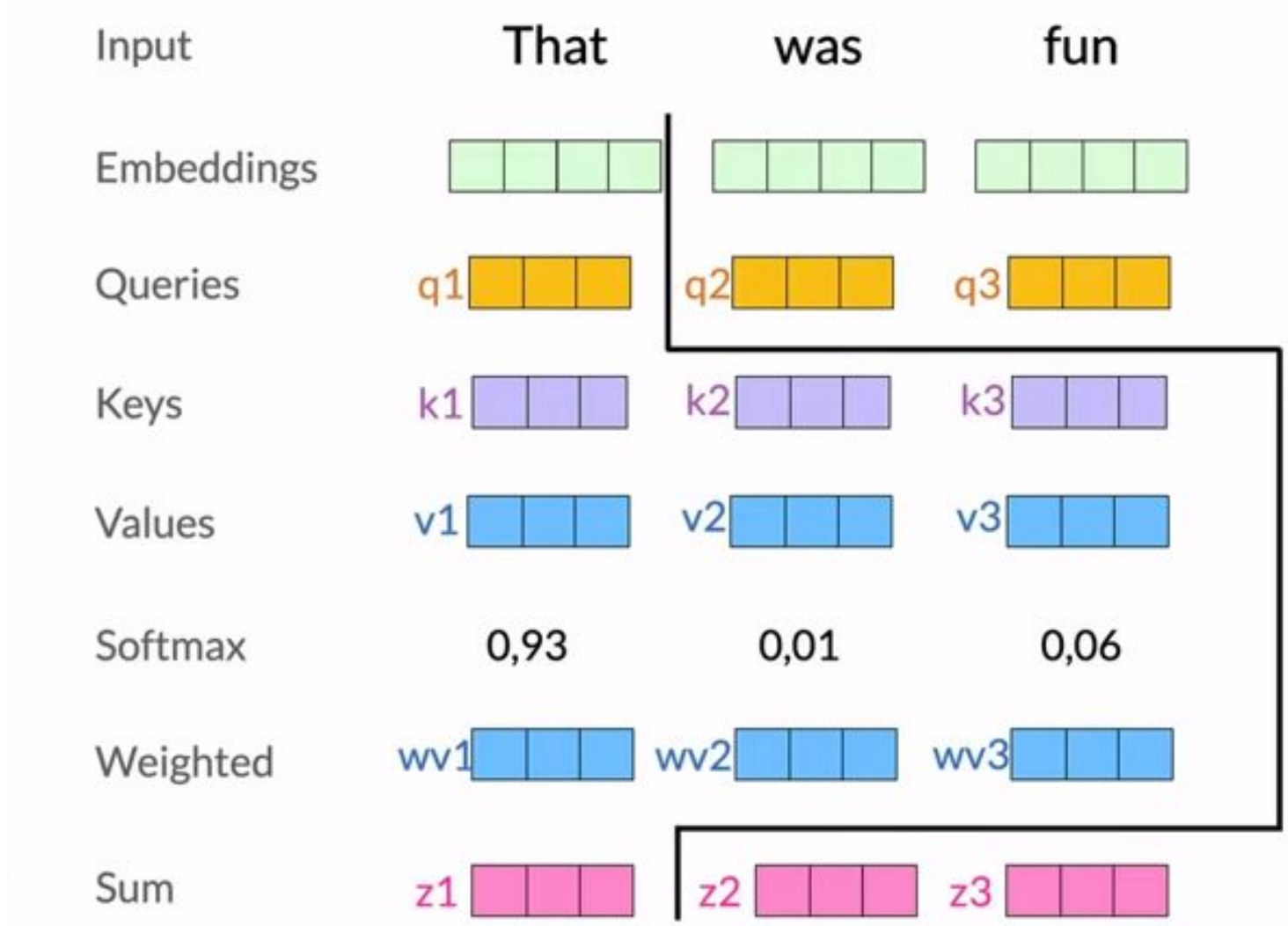

Input
That
was
fun
Embeddings
Queries
q1
q2
q3
Keys
k1
k2
k3
Values
v1
v2
v3
Score
q1 • k1 =98
q1 • k2 =46
q1 • k3 =76

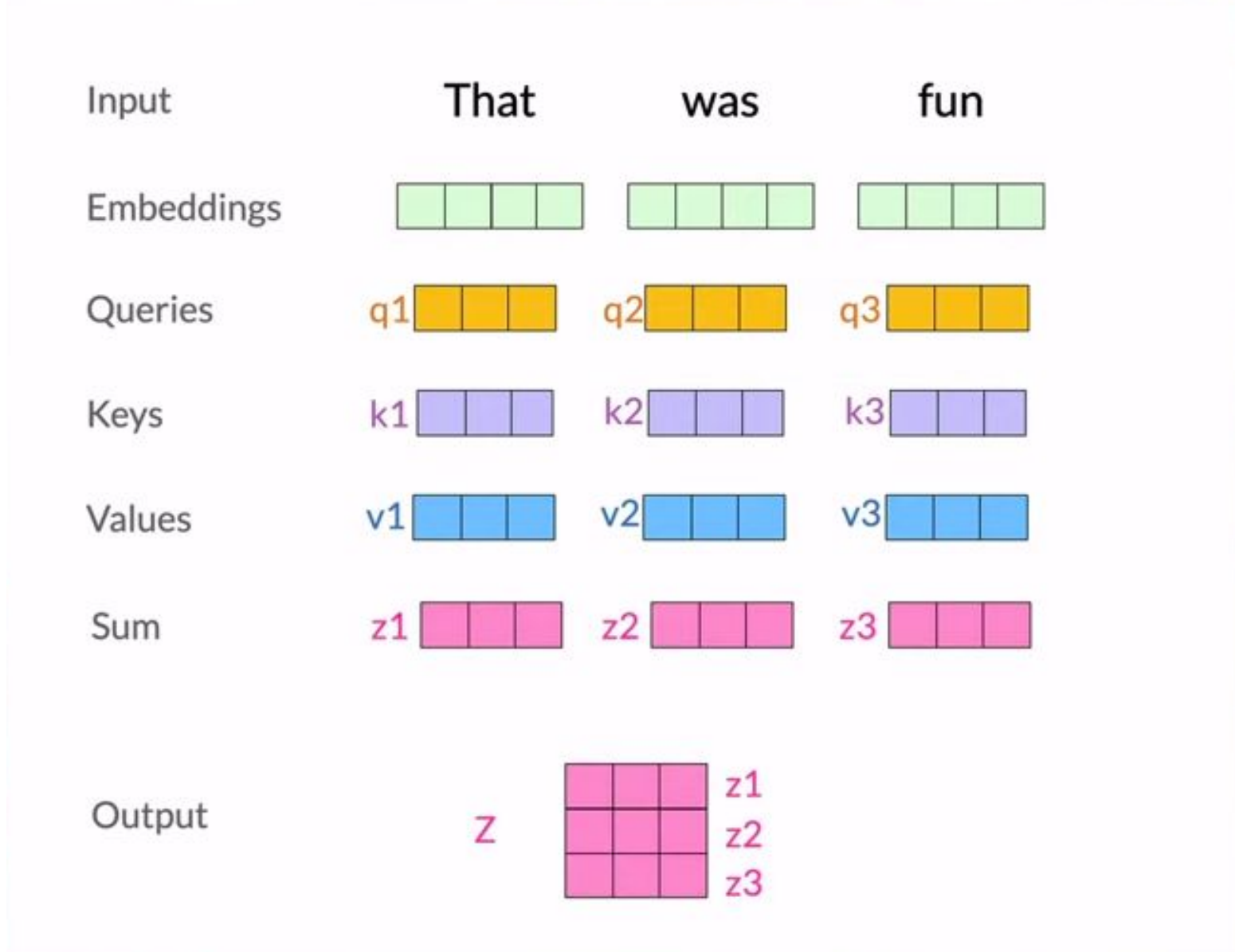

Input
That
was
fun
Embeddings
Queries
q1
q2
q3
Keys
k1
k2
k3
Values
v1
v2
v3
Score
q1 • k1 =98
q1 • k2 =46
q1 • k3 =76
Divide by 8
12,25
5,75
9,5


Input
That
was
fun
Embeddings
Queries
q1
q2
q3
Keys
k1
k2
k3
Values
v1
v2
v3
Score
q1 • k1 =98
q1 • k2 =46
q1 • k3 =76
Divide by 8
12,25
5,75
9,5
Softmax
0,93
0,01
0,06

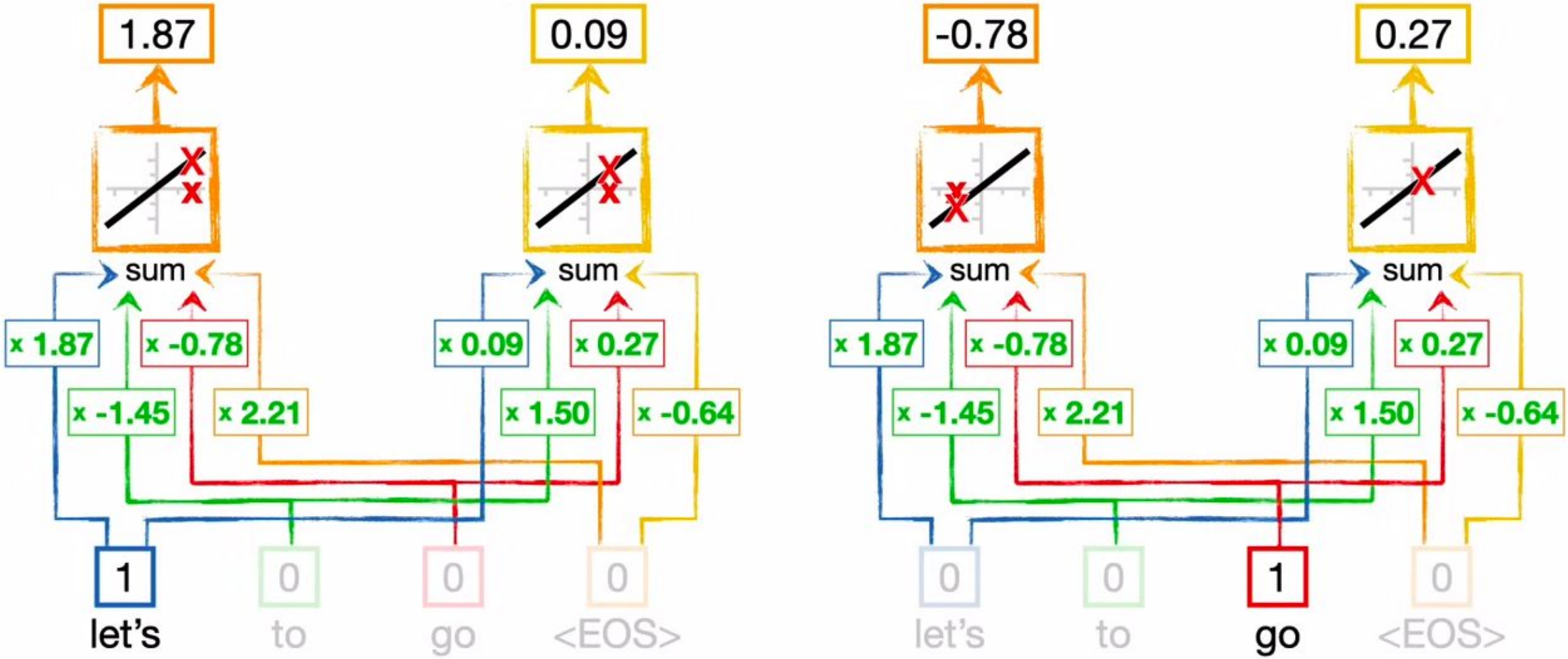

Input
That
was
fun
Embeddings
Queries
q1
q2
q3
Keys
k1
k2
k3
Values
v1
v2
v3
Softmax
0,93
0,01
0,06
Weighted
wv1
wv2
wv3

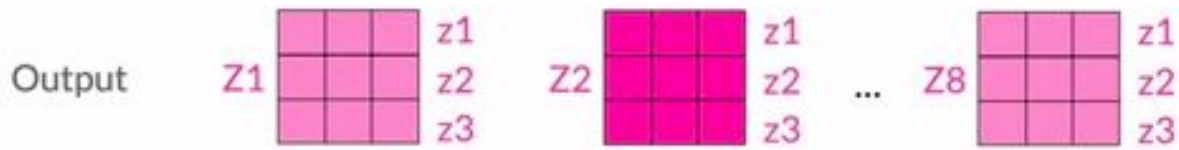

Input
That
was
fun
Embeddings
Queries
q1
q2
q3
Keys
k1
k2
k3
Values
v1
v2
v3
Softmax
0,93
0,01
0,06
Weighted
wv1
wv2
wv3
Sum
z1

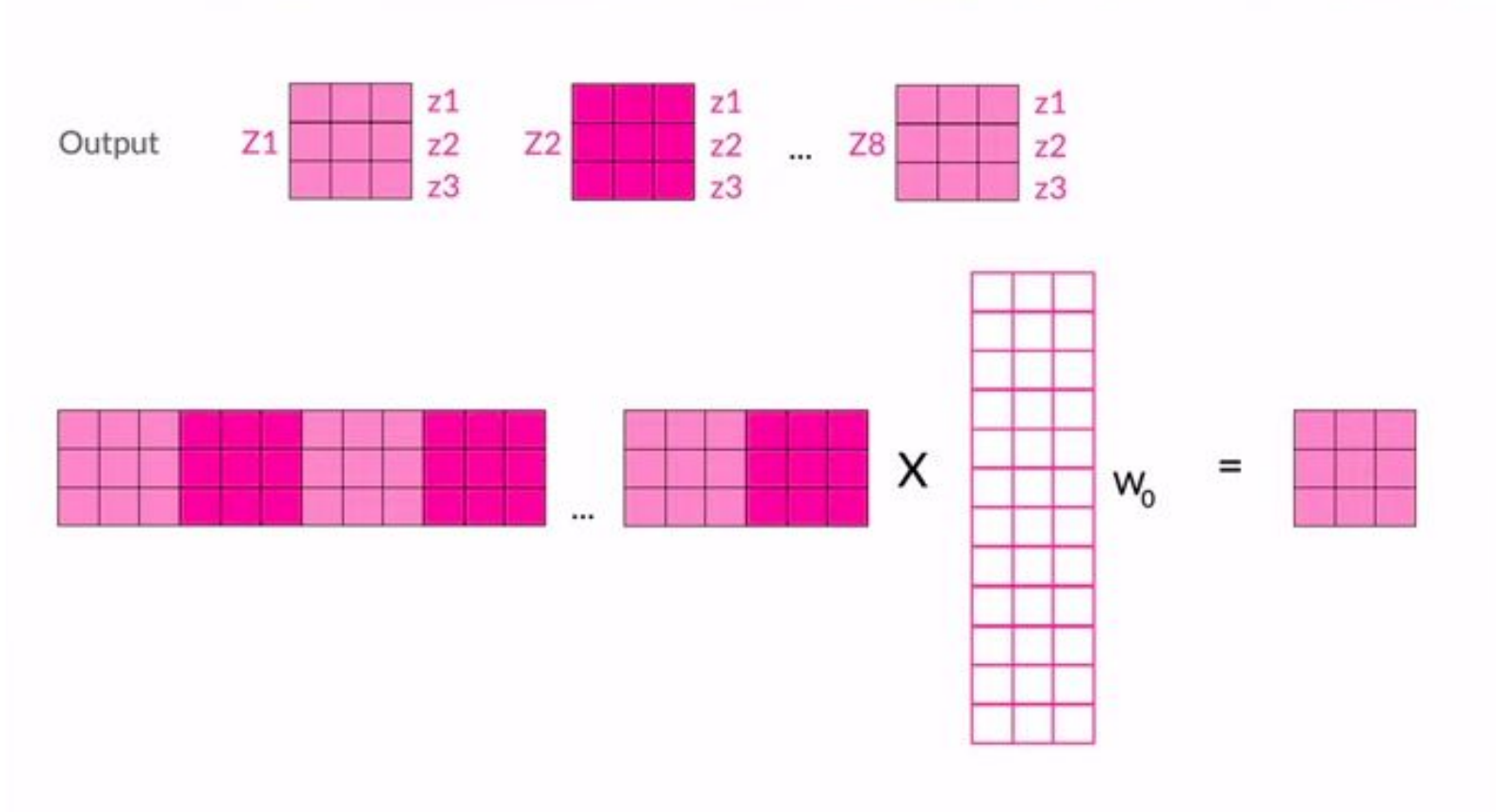

Input
That
was
fun
Embeddings
Queries
q1
q2
q3
Keys
k1
k2
k3
Values
v1
v2
v3
Softmax
0,93
0,01
0,06
Weighted
wv1
wv2
wv3
Sum
z1
z2
z3

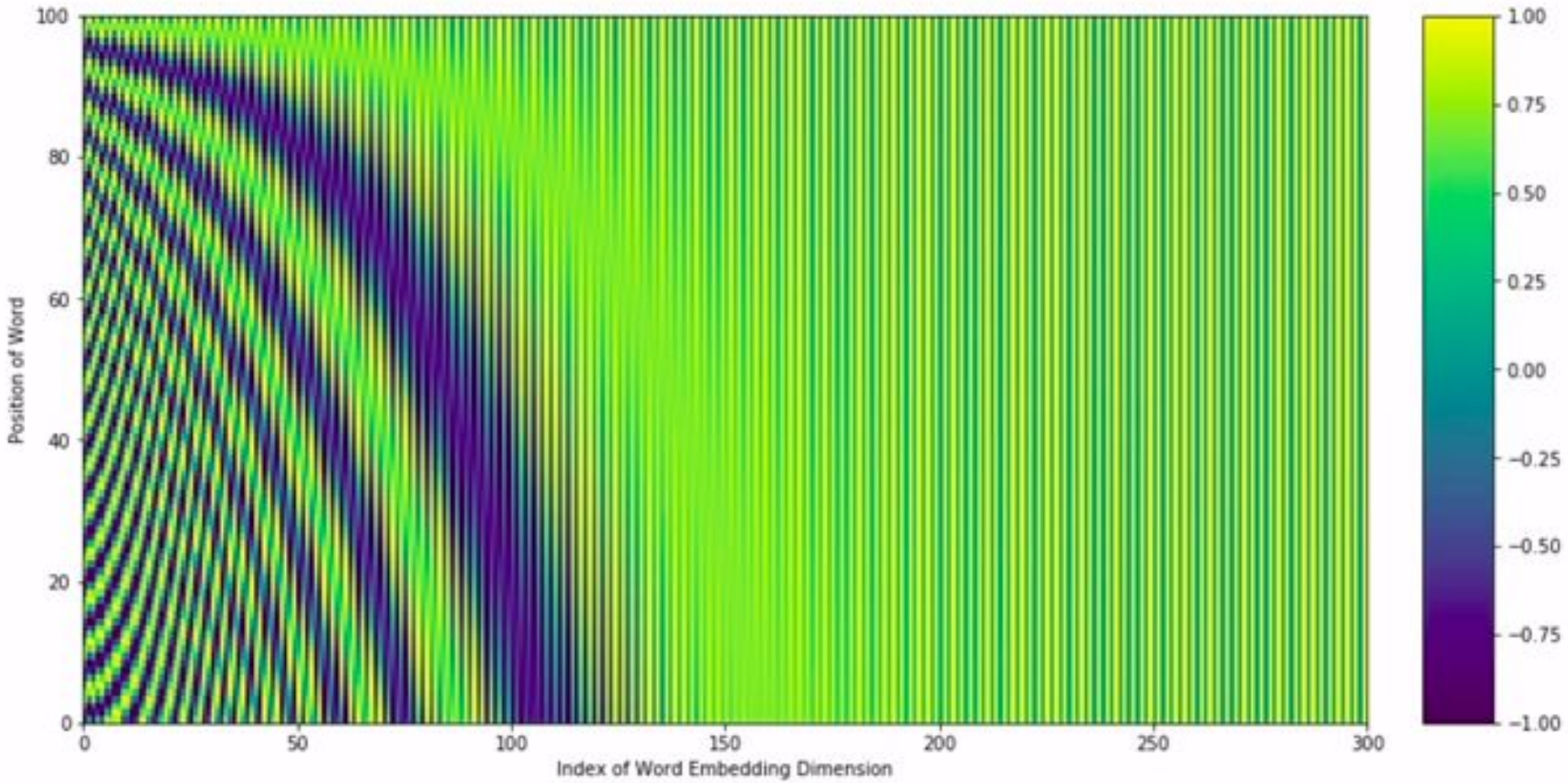

Input
That
was
fun
Embeddings
Queries
q1
q2
q3
Keys
k1
k2
k3
Values
v1
v2
v3
Sum
z1
z2
z3
Output
Z
z1
z2
z3

This means that regardless of how long the input sentence is, we just copy and use the exact same **Word Embedding** network for each word or symbol.

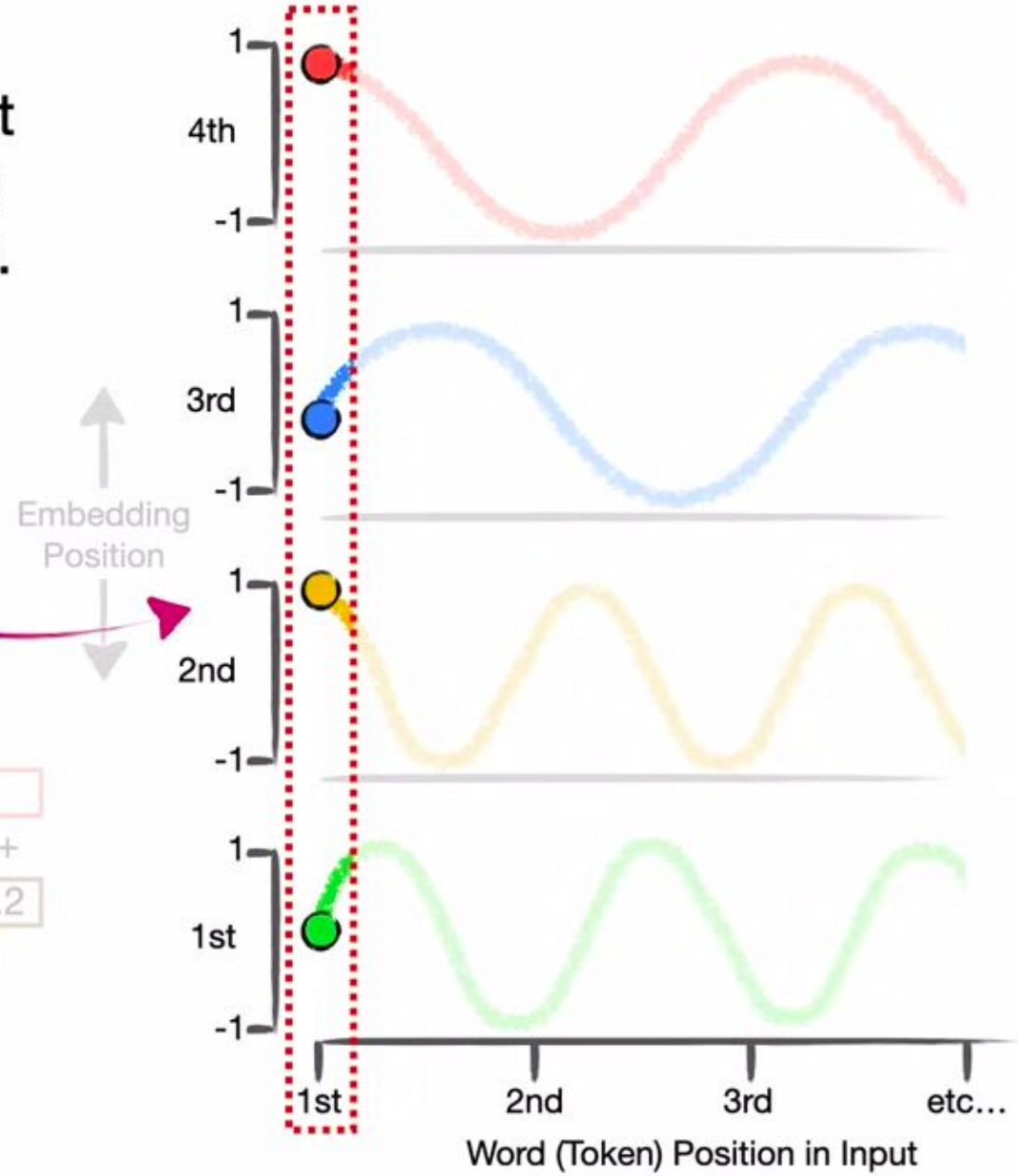



Output
Z1
z1
z2
z3
Z2
z1
z2
z3
...
Z8
z1
z2
z3


Output
Z1
z1
z2
z3
Z2
z1
z2
z3
...
Z8
z1
z2
z3
...
X
w_0
=

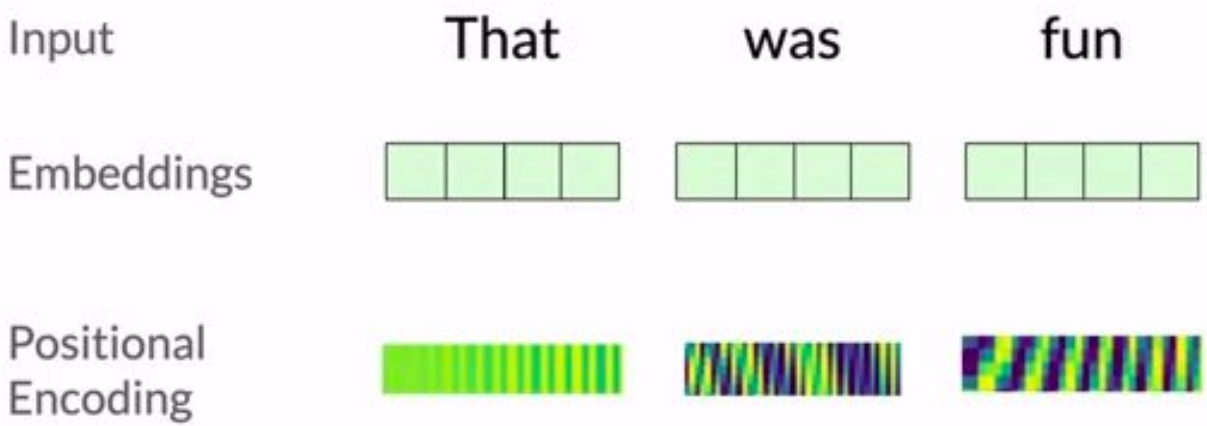

Position of Word
Index of Word Embedding Dimension
0
20
40
60
80
100
0
50
100
150
200
250
300
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00

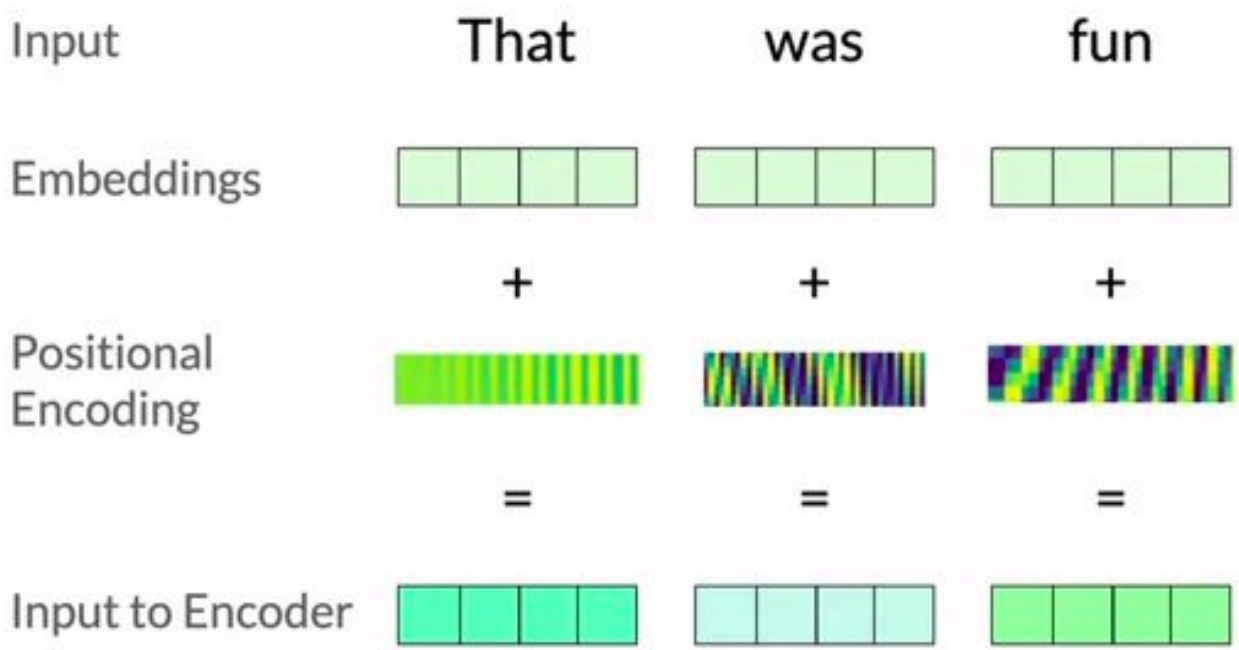
Thus, the position values for the first word come from the corresponding y-axis coordinates on the squiggles.

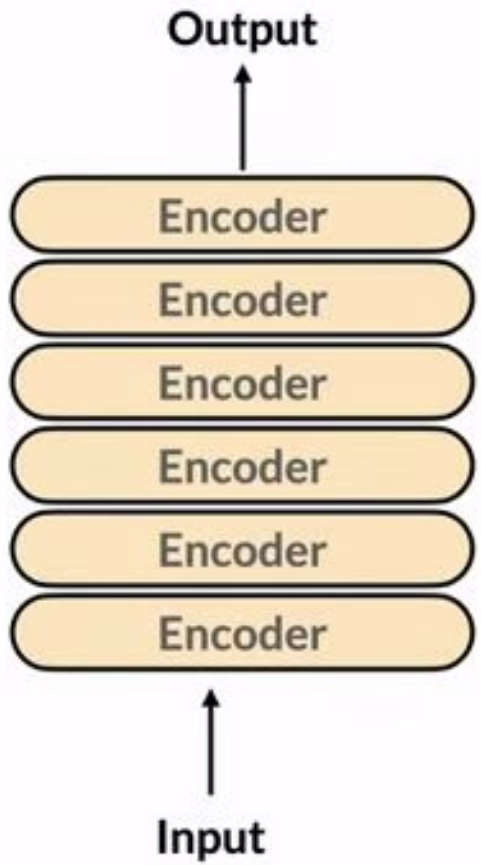


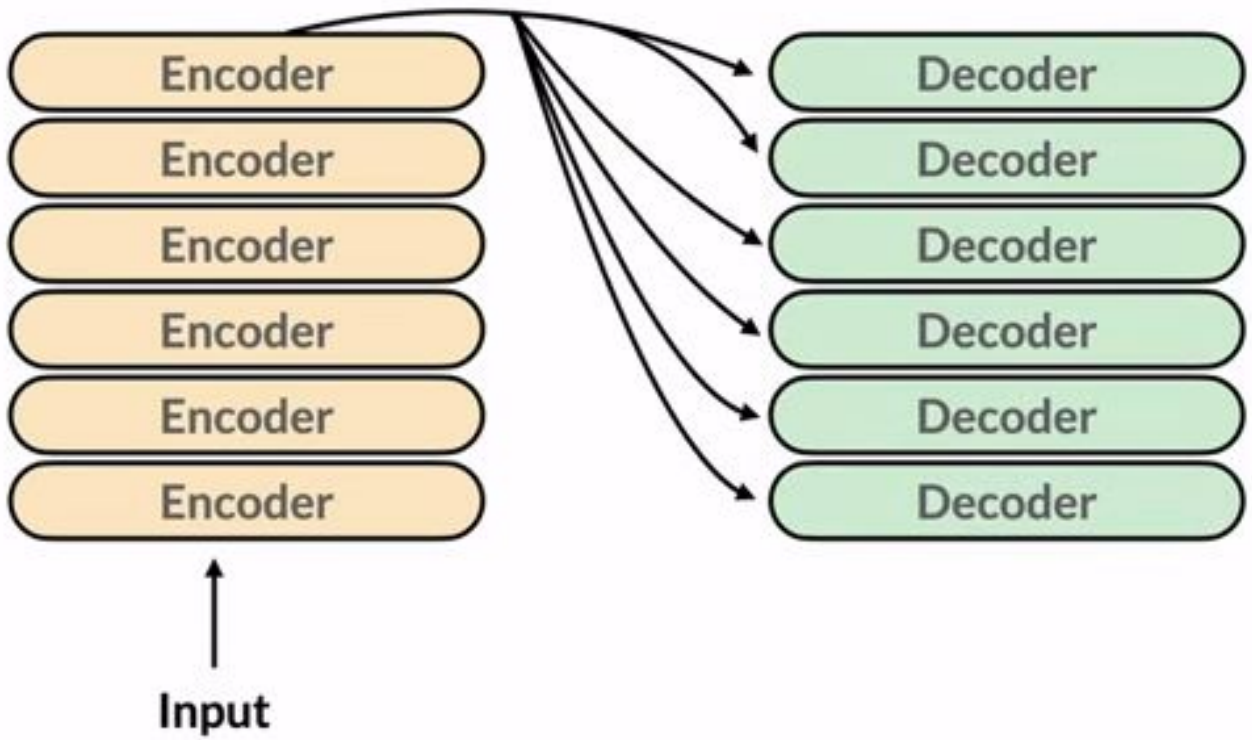
0: 0 0 0 0
1: 0 0 0 1
2: 0 0 1 0
3: 0 0 1 1
4: 0 1 0 0
5: 0 1 0 1
6: 0 1 1 0
7: 0 1 1 1

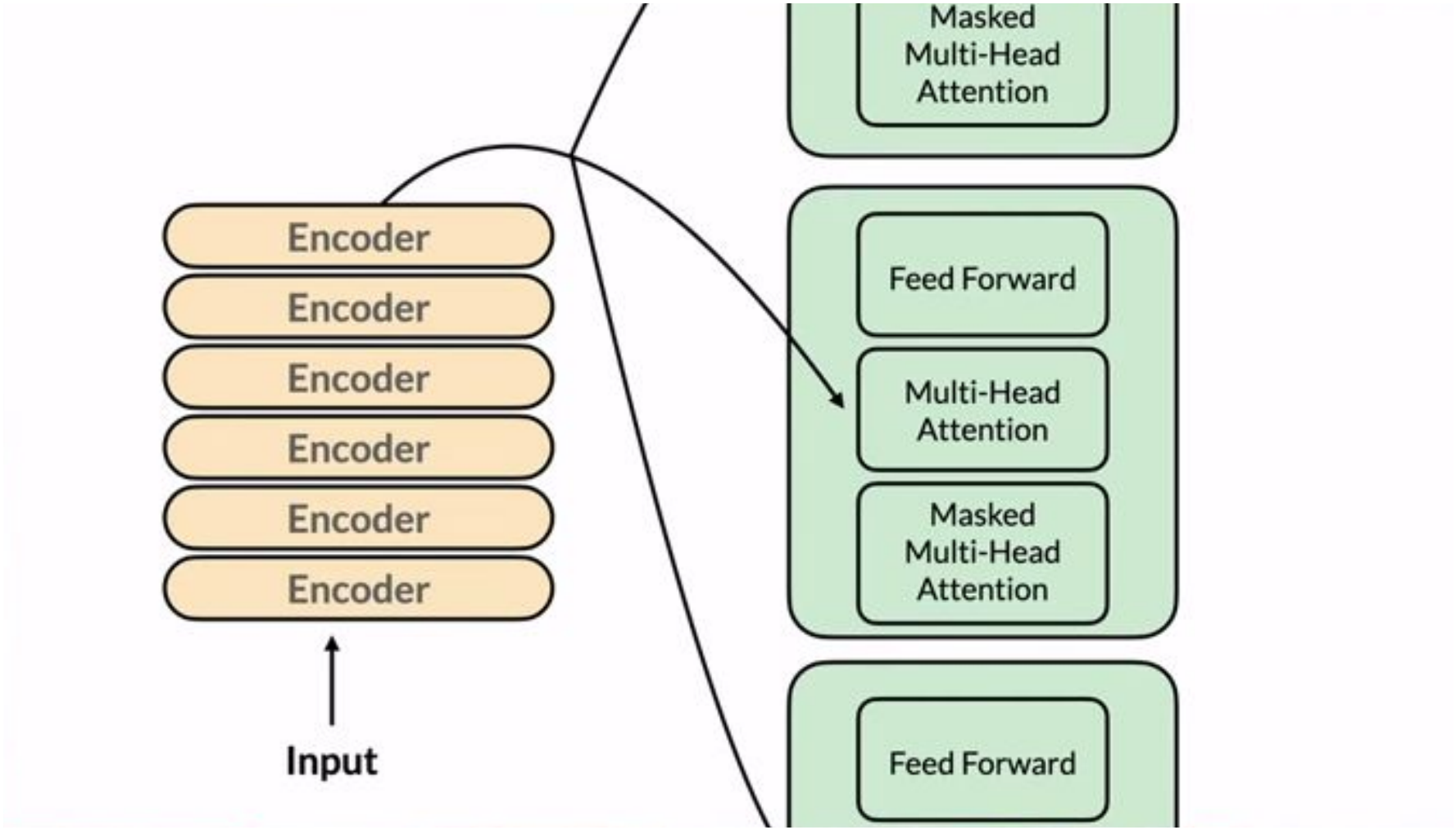
7: 1 0 0 0
8: 1 0 0 1
9: 1 0 1 0
10: 1 0 1 1
11: 1 1 0 0
12: 1 1 0 1
13: 1 1 1 0
14: 1 1 1 1

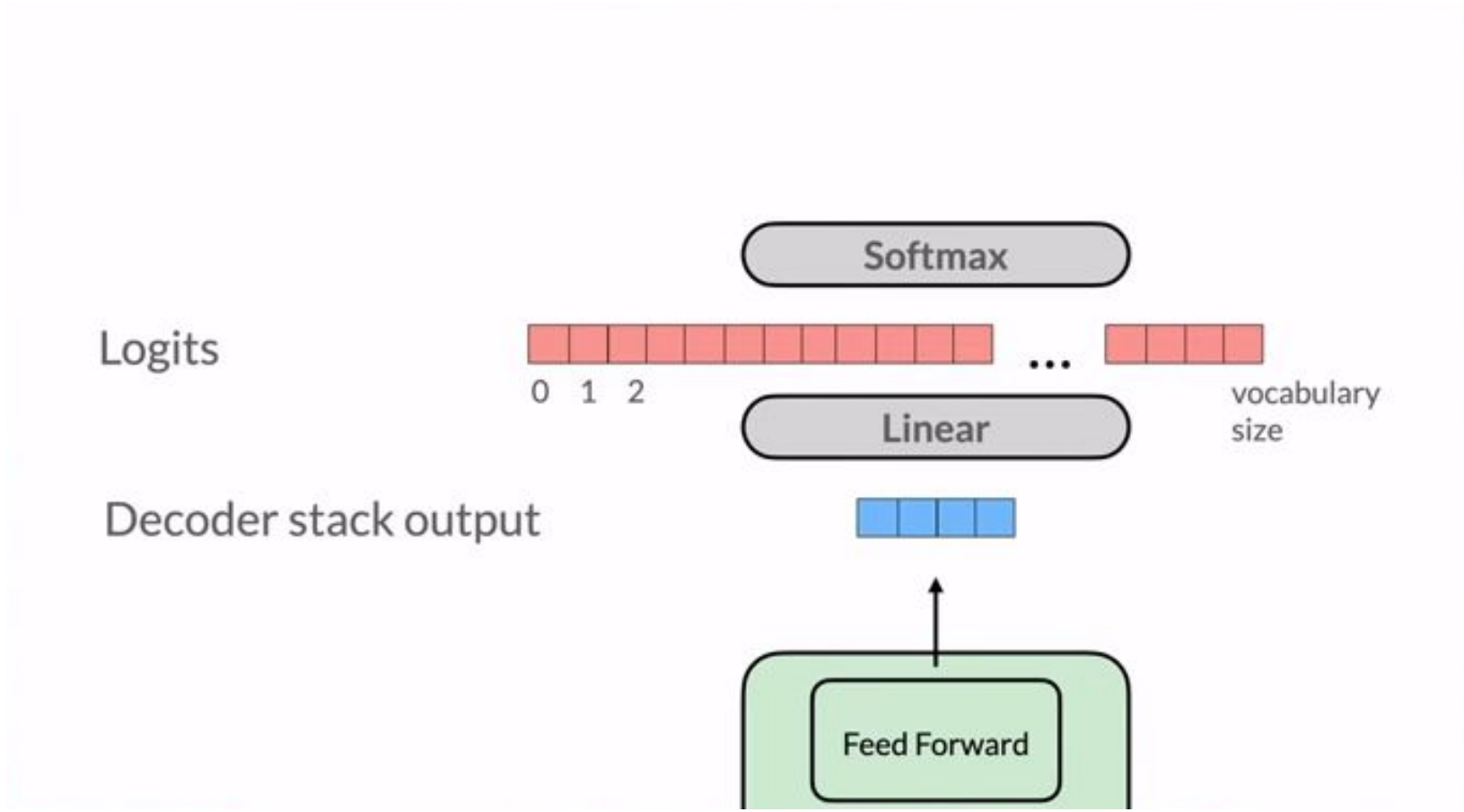

Input
That
was
fun
Embeddings
Positional Encoding

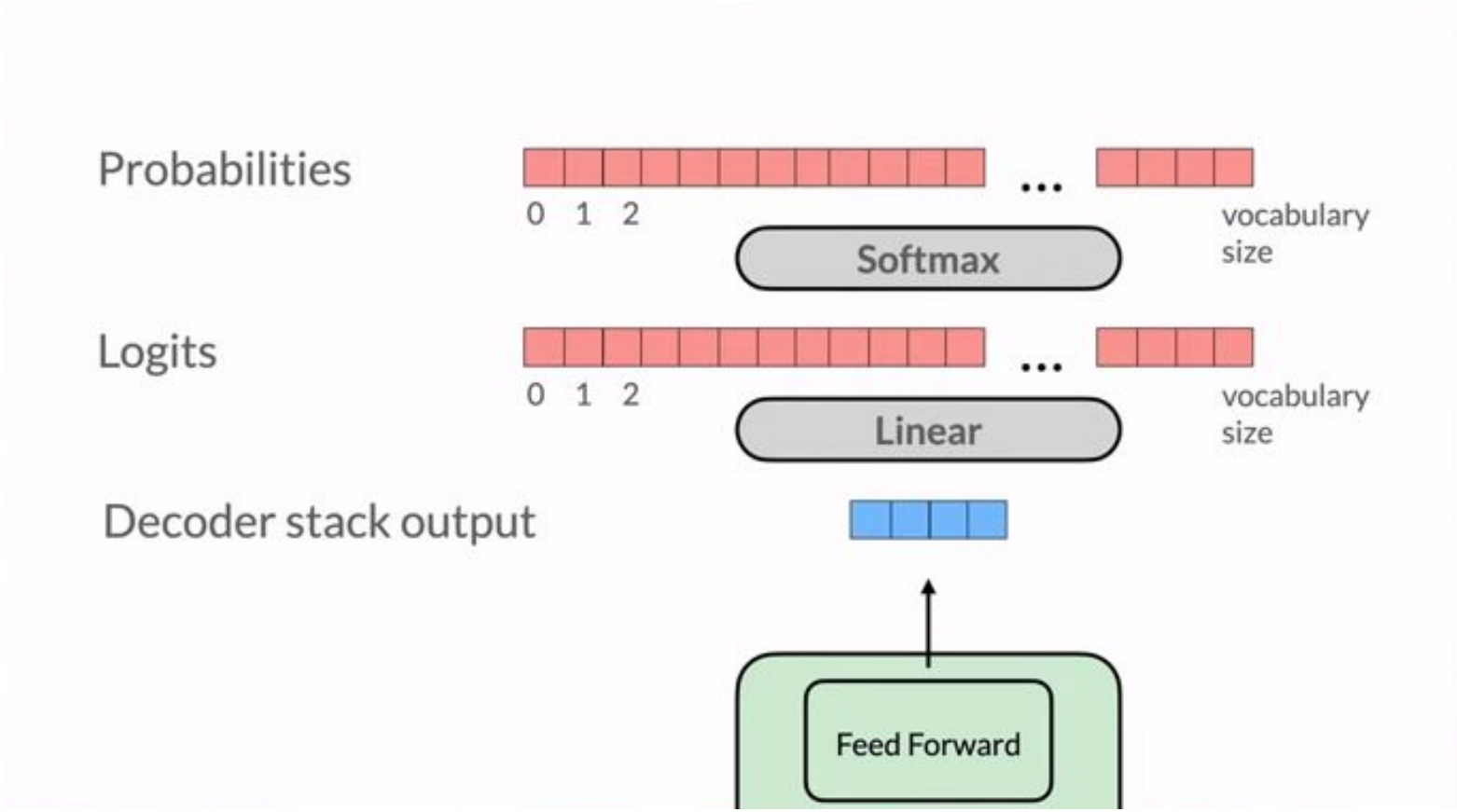

Input
That
was
fun
Embeddings
+
+
+
Positional Encoding
=
=
=
Input to Encoder


Output
Encoder
Encoder
Encoder
Encoder
Encoder
Encoder
Input


Encoder
Encoder
Encoder
Encoder
Encoder
Encoder
Decoder
Decoder
Decoder
Decoder
Decoder
Decoder
Input


Masked
Multi-Head
Attention
Encoder
Encoder
Encoder
Encoder
Encoder
Encoder
Feed Forward
Multi-Head
Attention
Masked
Multi-Head
Attention
Feed Forward
Input

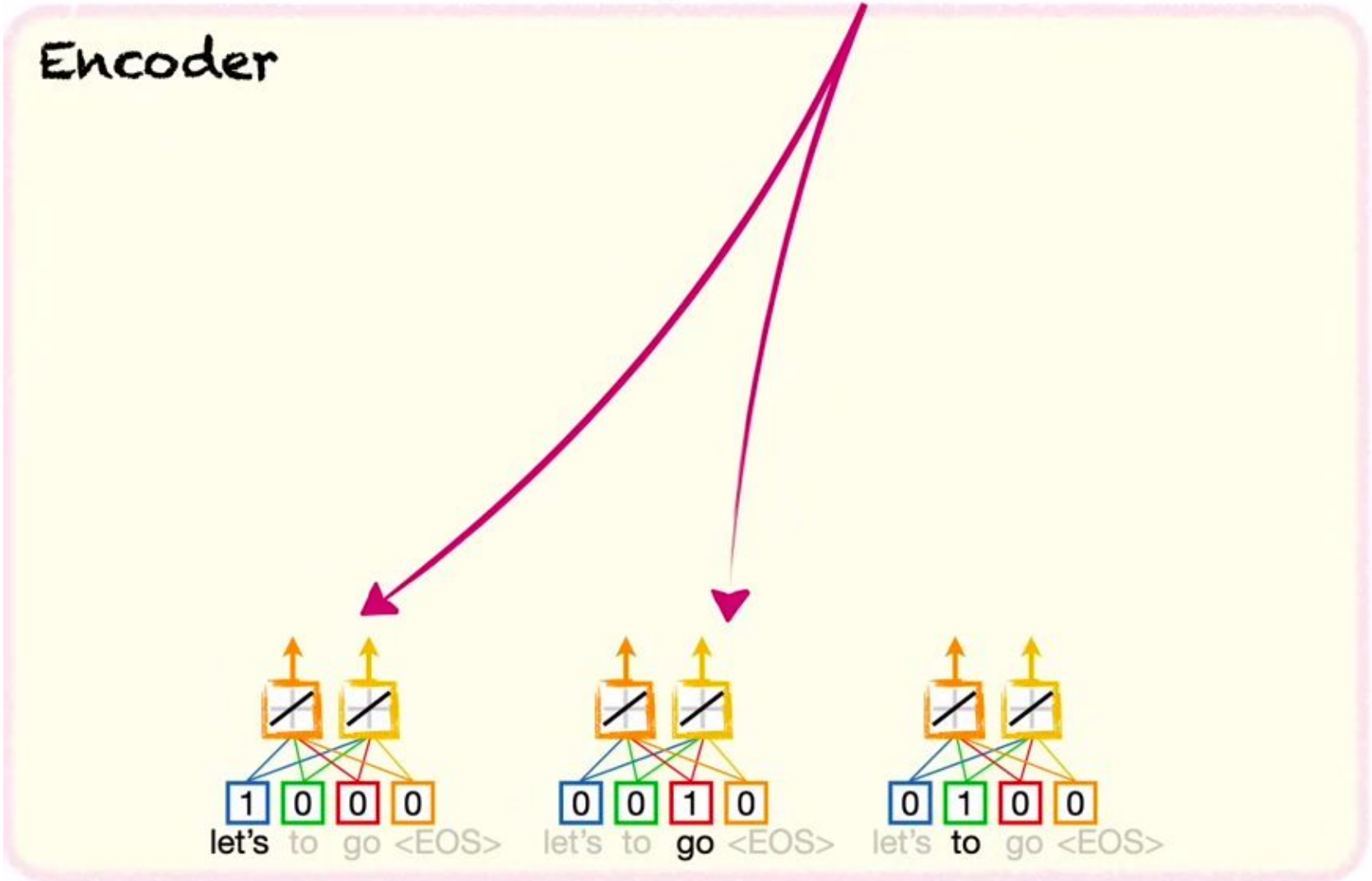

Softmax
Logits
0 1 2
...
vocabulary size
Linear
Decoder stack output
Feed Forward


Probabilities
0 1 2
...
vocabulary size
Softmax
Logits
0 1 2
...
vocabulary size
Linear
Decoder stack output
Feed Forward

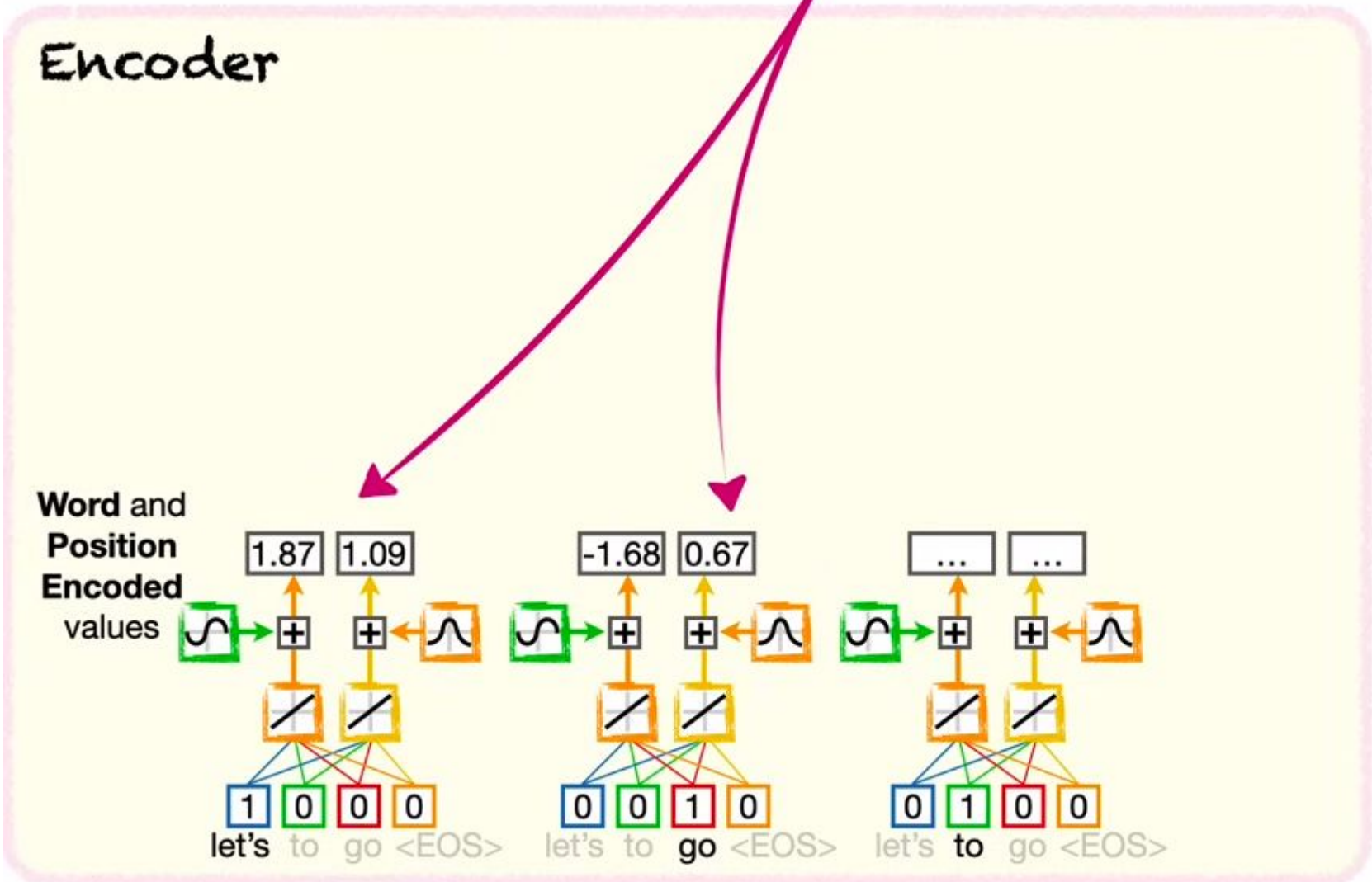
Reading Assignment

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

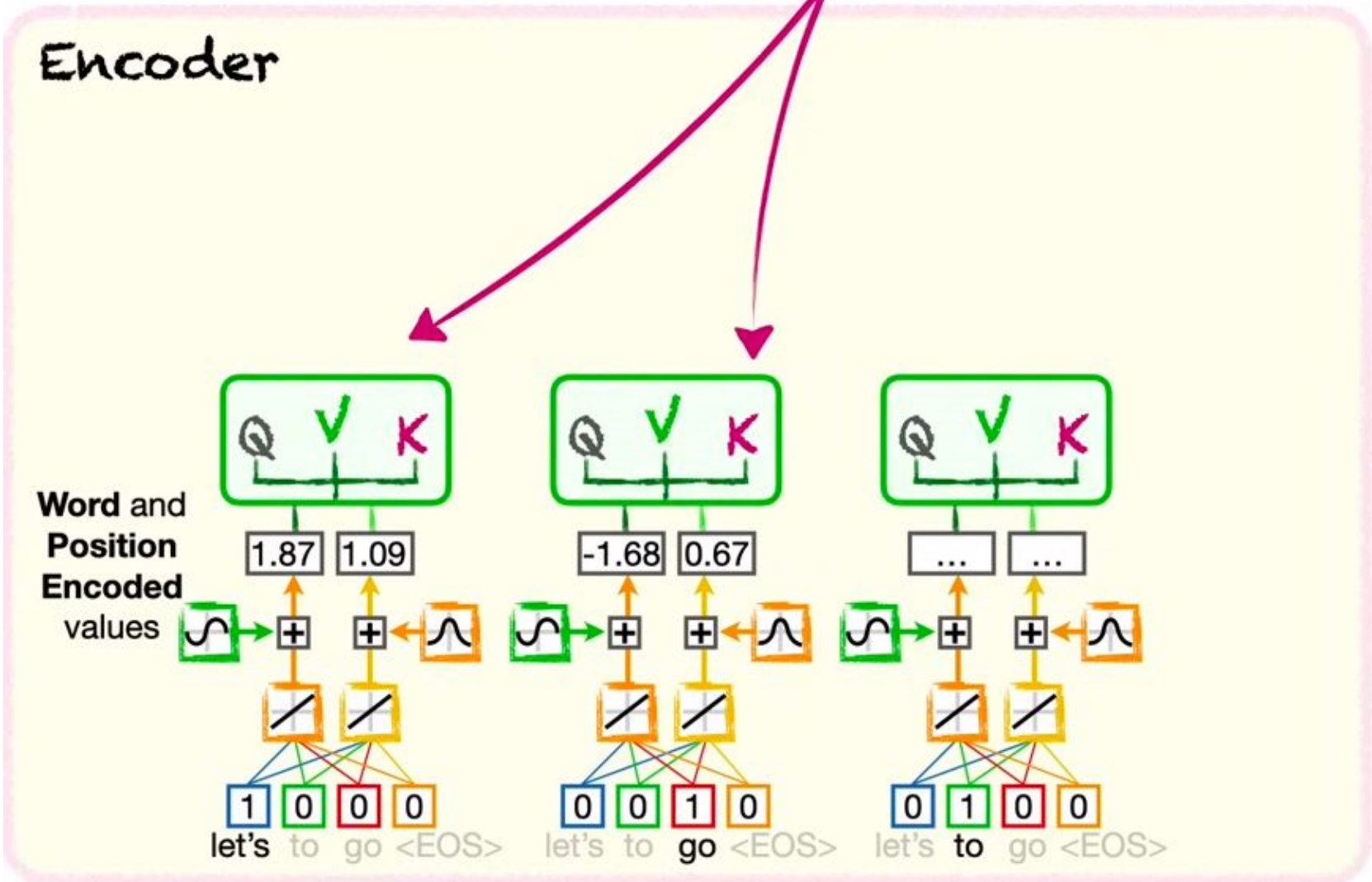
For example, we can calculate the **Word Embeddings** on different processors at the same time…



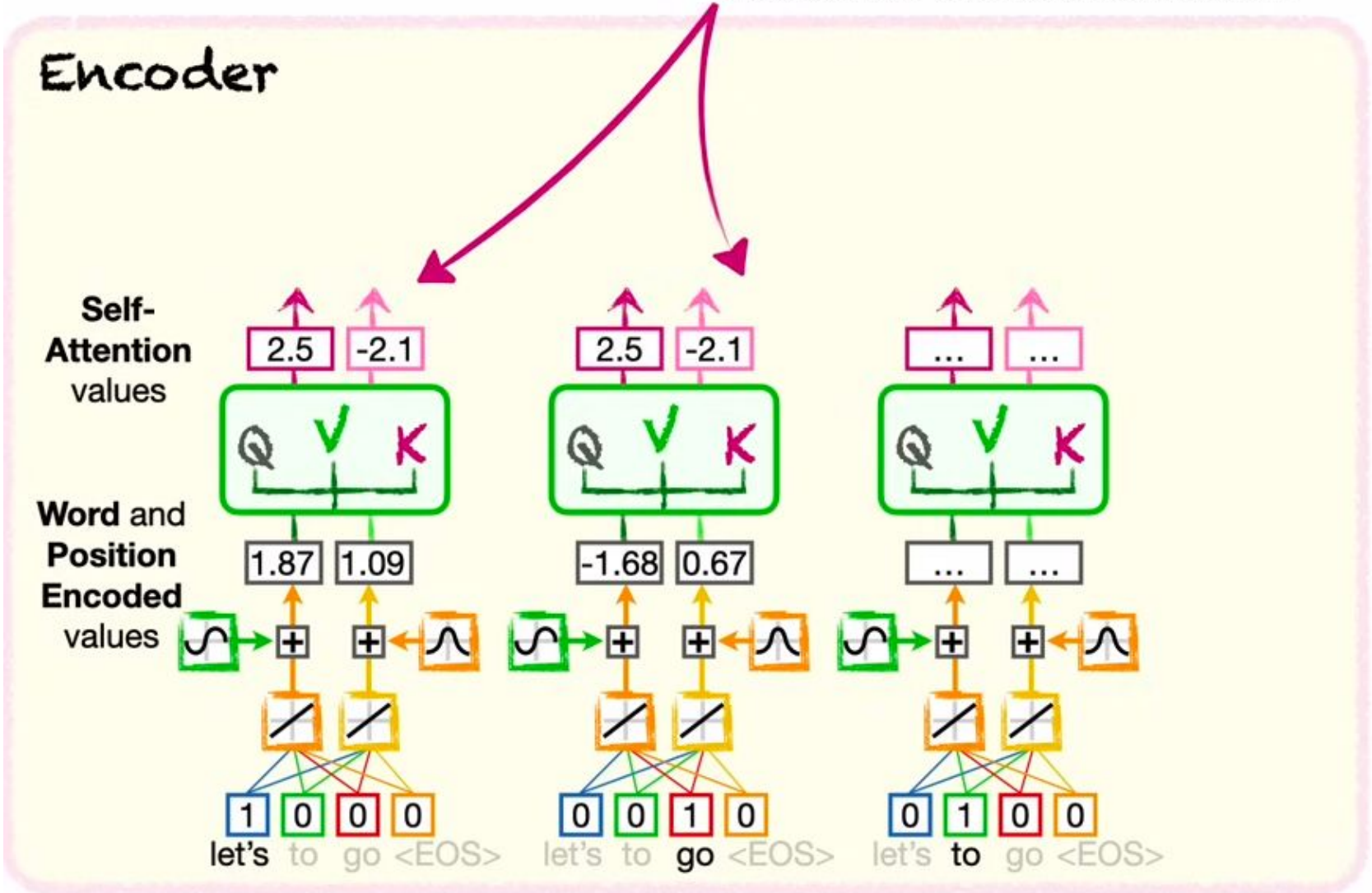
…and then add the **Positional Encoding** at the same time…



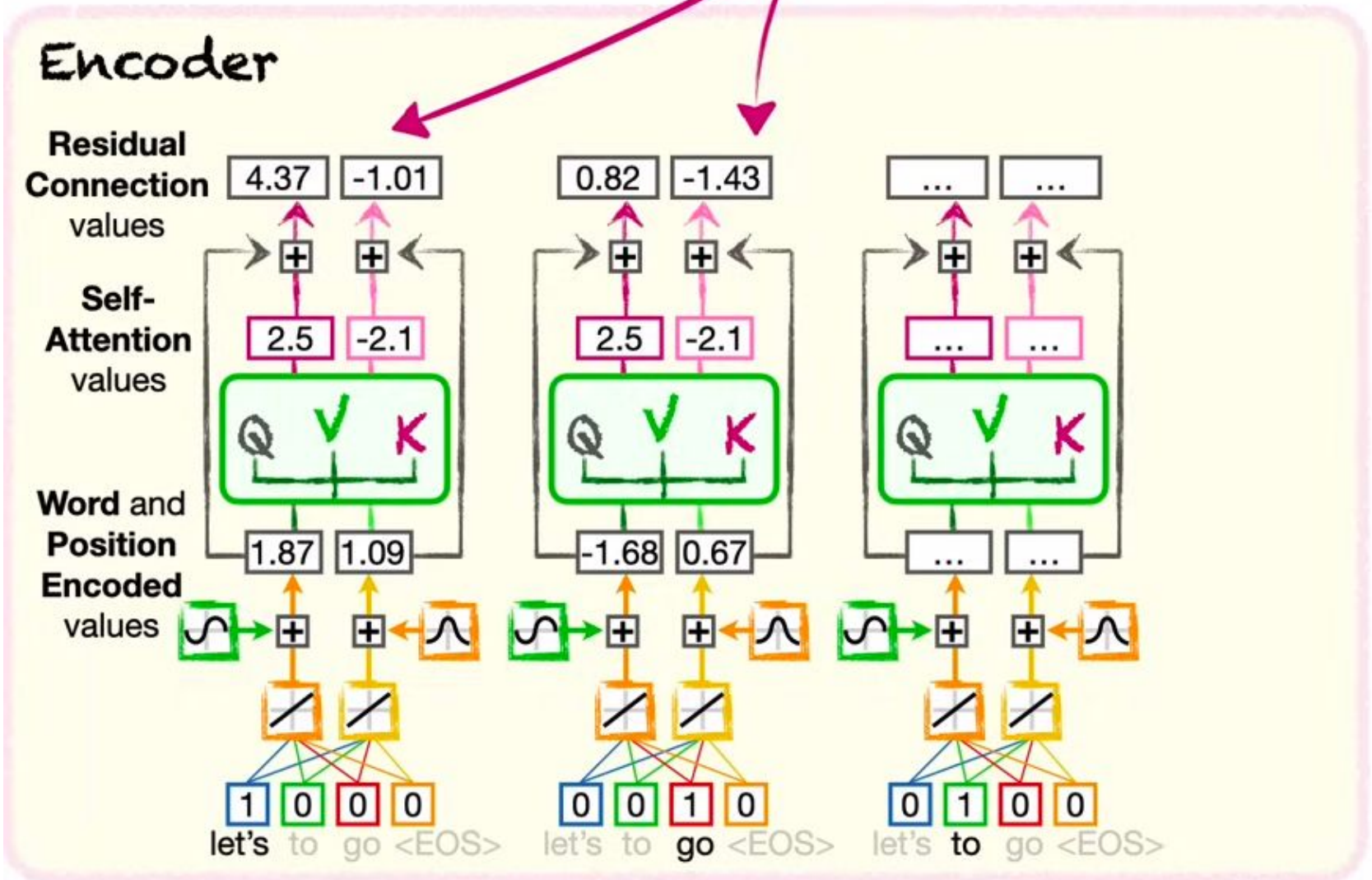
…and then calculate the **Queries**, **Keys** and **Values** at the same time…



…and, once that is done, we can calculate the **Self-Attention** values at the same time…



…and, lastly, we can calculate the **Residual Connections** at the same time.



Doing all of the computations at the same time, rather than doing them sequentially for each word…