Attention & Transformers

Mausam
IIT Delhi

(some figures taken from Jay Alammar's blog)

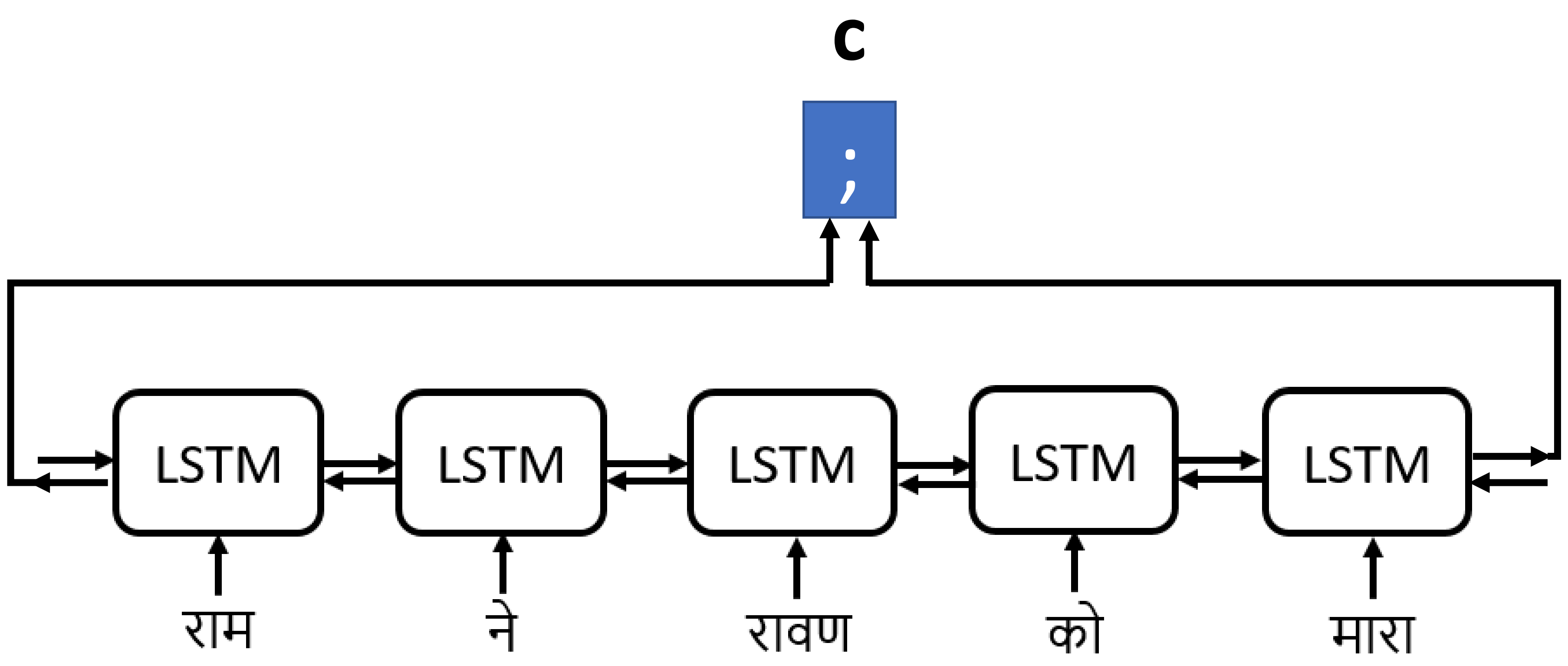
Attention

Sentence Representation

*You can’t cram the meaning of the whole *%#@ing sentence in a single *%#@ing vector.*

- Encoding a single vector is too restrictive.
  Instead of producing a single vector for the sentence, produce one vector for each word.
- But, eventually need 1 vector.
  Multiple vectors → Single vector
  Sum/Avg operators give equal importance to each input
- We dynamically decide which input is more/less important for a task.
- Create a weighted sum to reflect this variation: Attention
- query (q): decides importance of each input
  attention weights (α_i): normalized importance of input
  unnormalized attention weights ($\bar{\alpha}_i$): intermediate step to compute α_i
  attended summary: weighted avg of input with α weights

LSTM Encoder



Encoder
(Bi-LSTM)

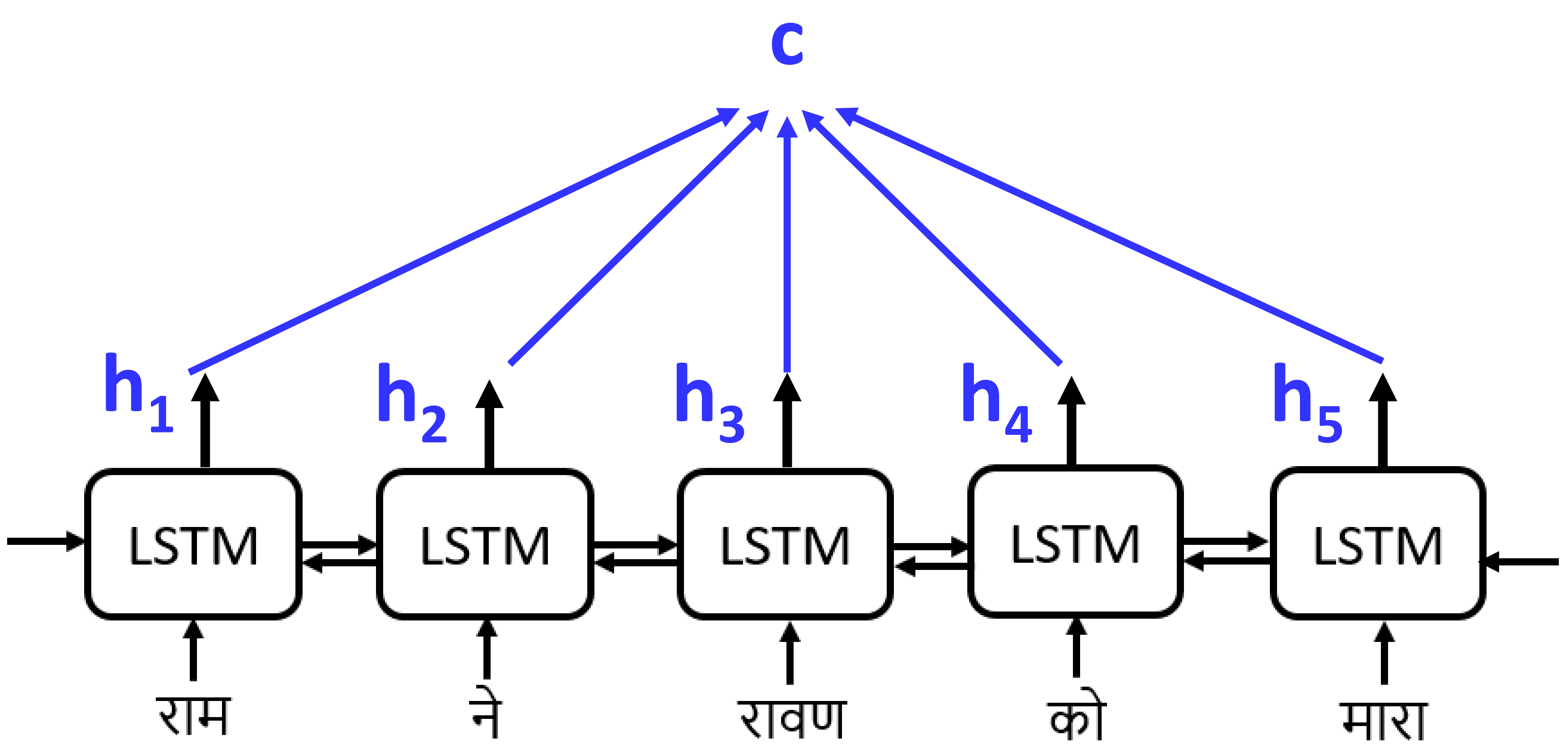
Multiple Encoded Vectors → Single Summary

$$h_{1:T} = \text{biLSTM}(x_{1:T})$$

$$c = attend(h_{1:T}, q)$$

$$c = \sum_{i=1}^{T} \alpha_i . h_i$$

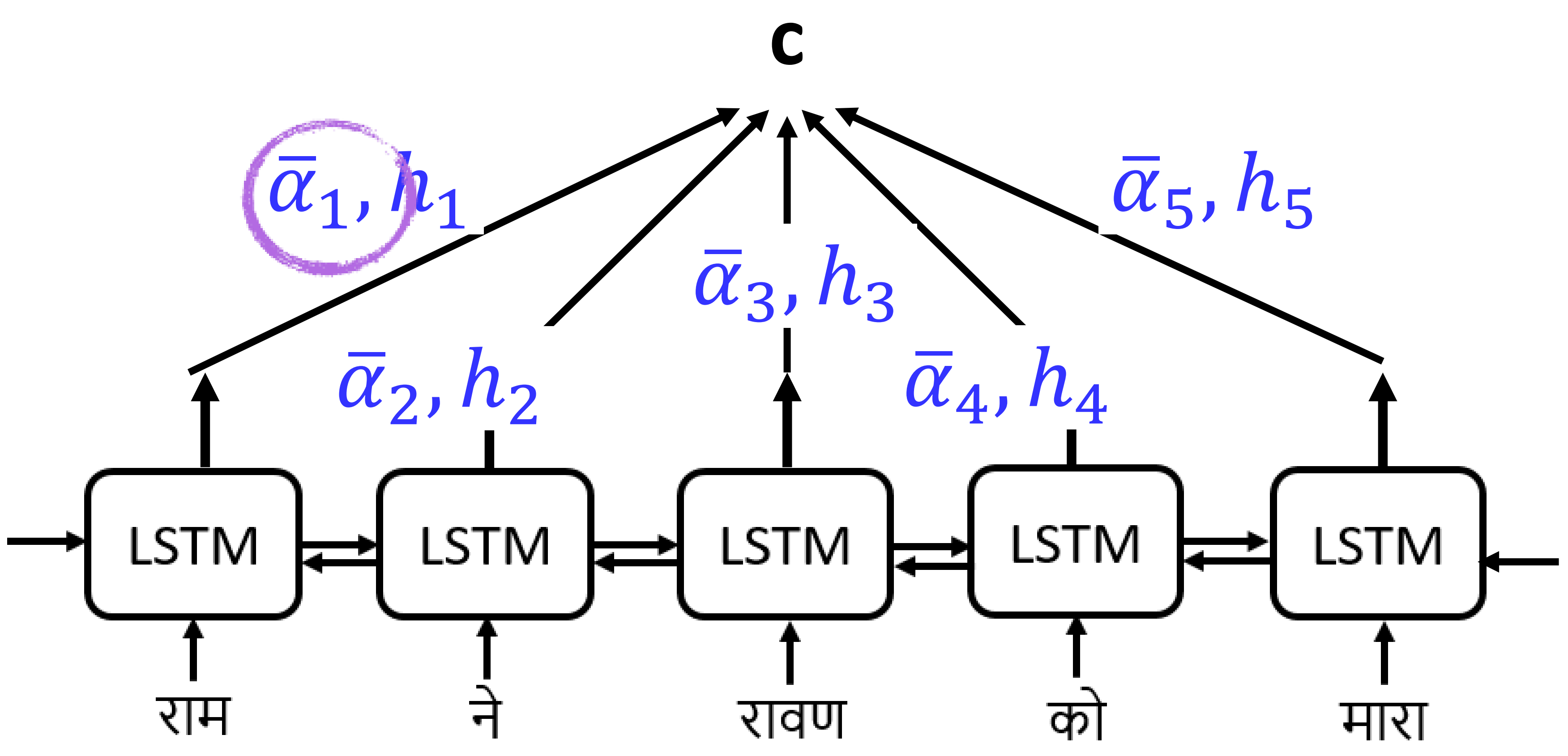
Need to convert h_is to c



Multiple Encoded Vectors → Single Summary

$$c = \sum_{i=1}^{T} \alpha_i . h_i$$

$$\alpha_{1:T} = \text{softmax}(\bar{\alpha}_1, \bar{\alpha}_2, \dots, \bar{\alpha}_T)$$

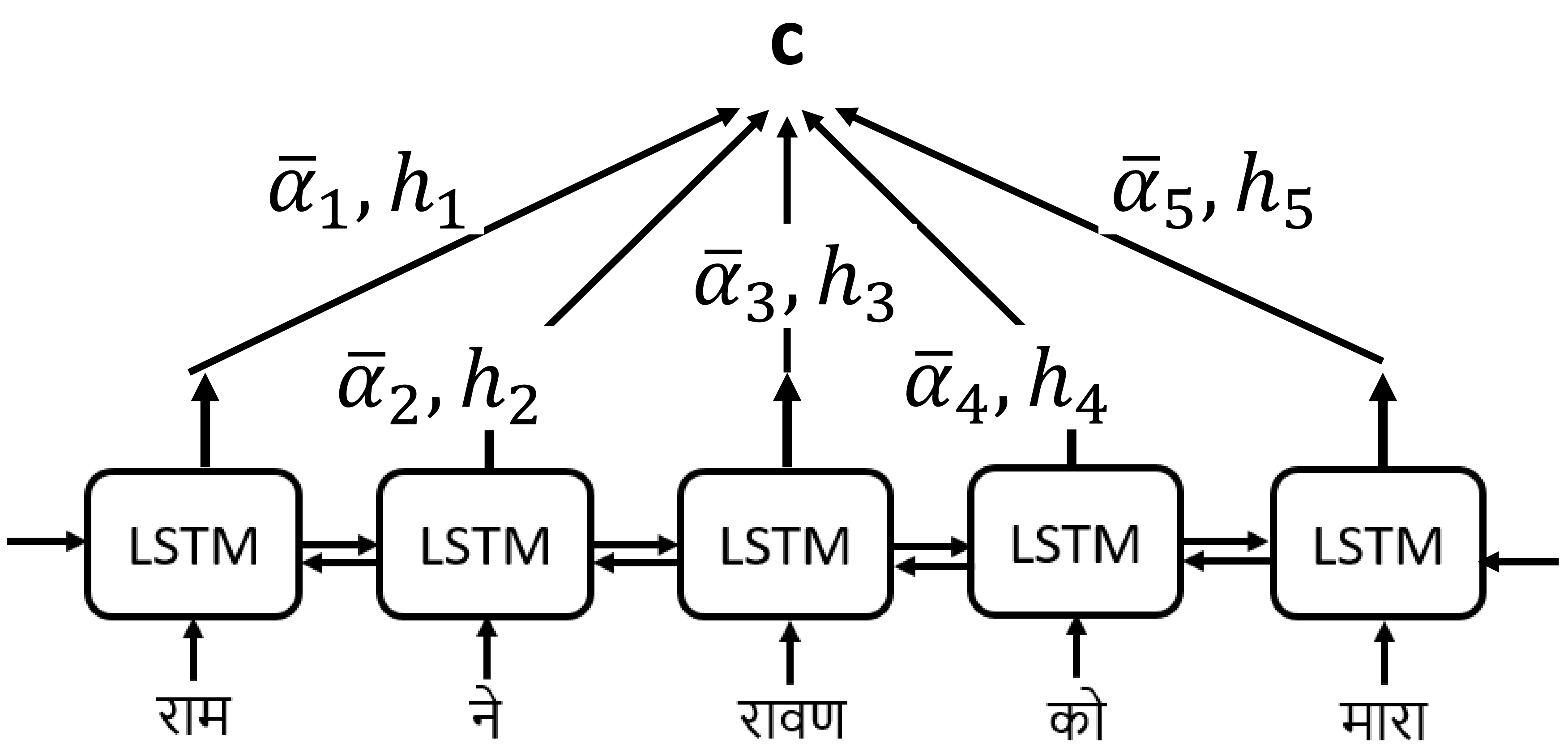


Multiple Encoded Vectors → Single Summary

$$c = \sum_{i=1}^{T} \alpha_i . h_i$$

$$\alpha_{1:T} = \text{softmax}(\bar{\alpha}_1, \bar{\alpha}_2, \dots, \bar{\alpha}_T\ )$$

$$\bar{\alpha}_i = \phi^{\text{att}}(q, h_i)$$



Attention: Encoding

$$c = \sum_{i=1}^{T} \alpha_i . h_i$$

$$h_{1:T} = \text{biLSTM}_{enc}(x_{1:T})$$

$$\alpha = \text{softmax}(\bar{\alpha}_1, \dots, \bar{\alpha}_T)$$

$$\bar{\alpha}_i = \phi^{\text{att}}(q, h_i)$$

what is ϕ^{att}?

what is q?

Attention Functions ϕ^{att}

- Bahadanau Attention: $\phi^{\mathrm{att}}(q,h) = \mathrm{u.g(Wq + W'h + b)}$
- Luong Attention: $\phi^{\mathrm{att}}(q,h) = \mathrm{q.h}$
- Scaled Dot Product Attention: $\phi^{\mathrm{att}}(q,h) = \frac{\mathrm{q.h}}{\sqrt{d}}$
- Bilinear Attention: $\phi^{\mathrm{att}}(q,h) = \mathrm{hWq}$

Additive vs Multiplicative

While the two are similar in theoretical complexity, dot-product attention is much faster and more space-efficient in practice, since it can be implemented using highly optimized matrix multiplication code.

While for small values of d_k the two mechanisms perform similarly, additive attention outperforms dot product attention without scaling for larger values of d_k [3]. We suspect that for large values of d_k, the dot products grow large in magnitude, pushing the softmax function into regions where it has extremely small gradients [4]. To counteract this effect, we scale the dot products by $\frac{1}{\sqrt{d_k}}$.

d is the dimensionality of q and h $\frac{q.h}{\sqrt{d}}$

Scaled dot product attention

Paper's Justification:

To illustrate why the dot products get large, assume that the components of q and h are independent random variables with mean 0 and variance 1 → Then their dot product, q · h has mean 0 and variance d

Attention: Encoding

$$c = \sum_{i=1}^{T} \alpha_i . h_i$$

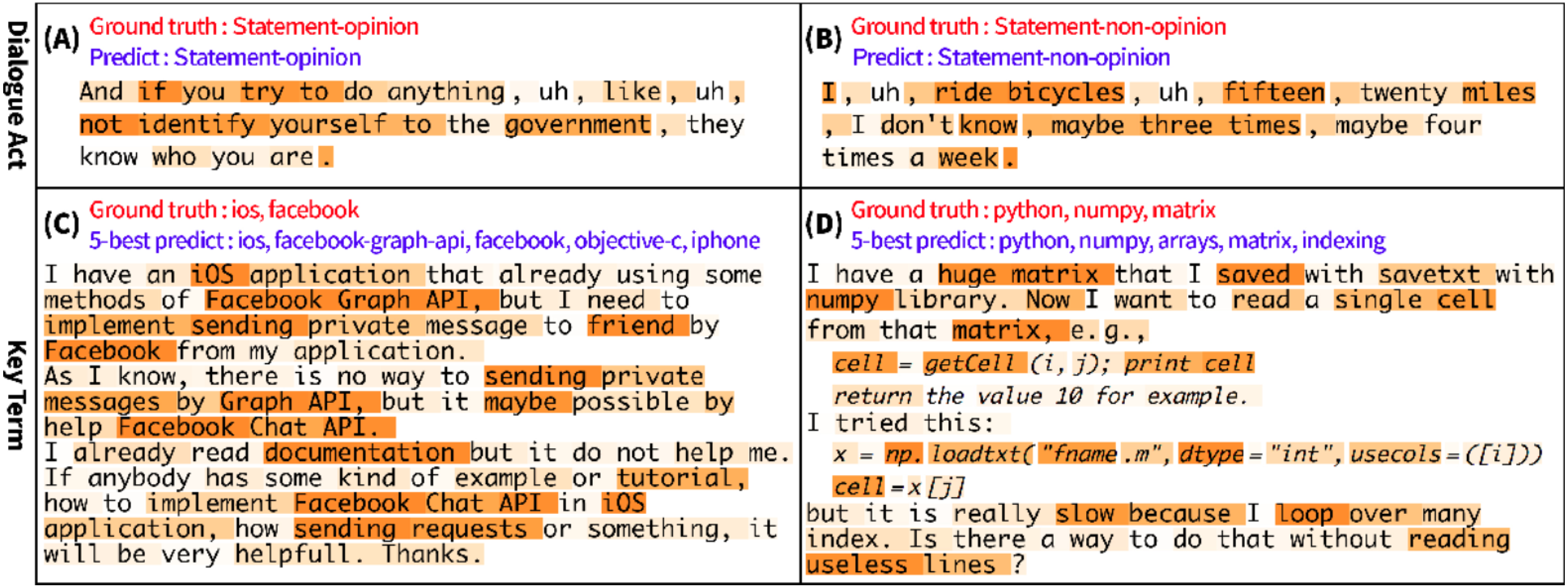
$$h_{1:T} = \text{biLSTM}_{enc}(x_{1:T})$$

$$\alpha = \text{softmax}(\bar{\alpha}_1, \ldots, \bar{\alpha}_T)$$

$$\bar{\alpha}_i = \phi^{\text{att}}(q, h_i)$$

what is q?

Attention and/vs Interpretation



Published in INTERSPEECH 2016

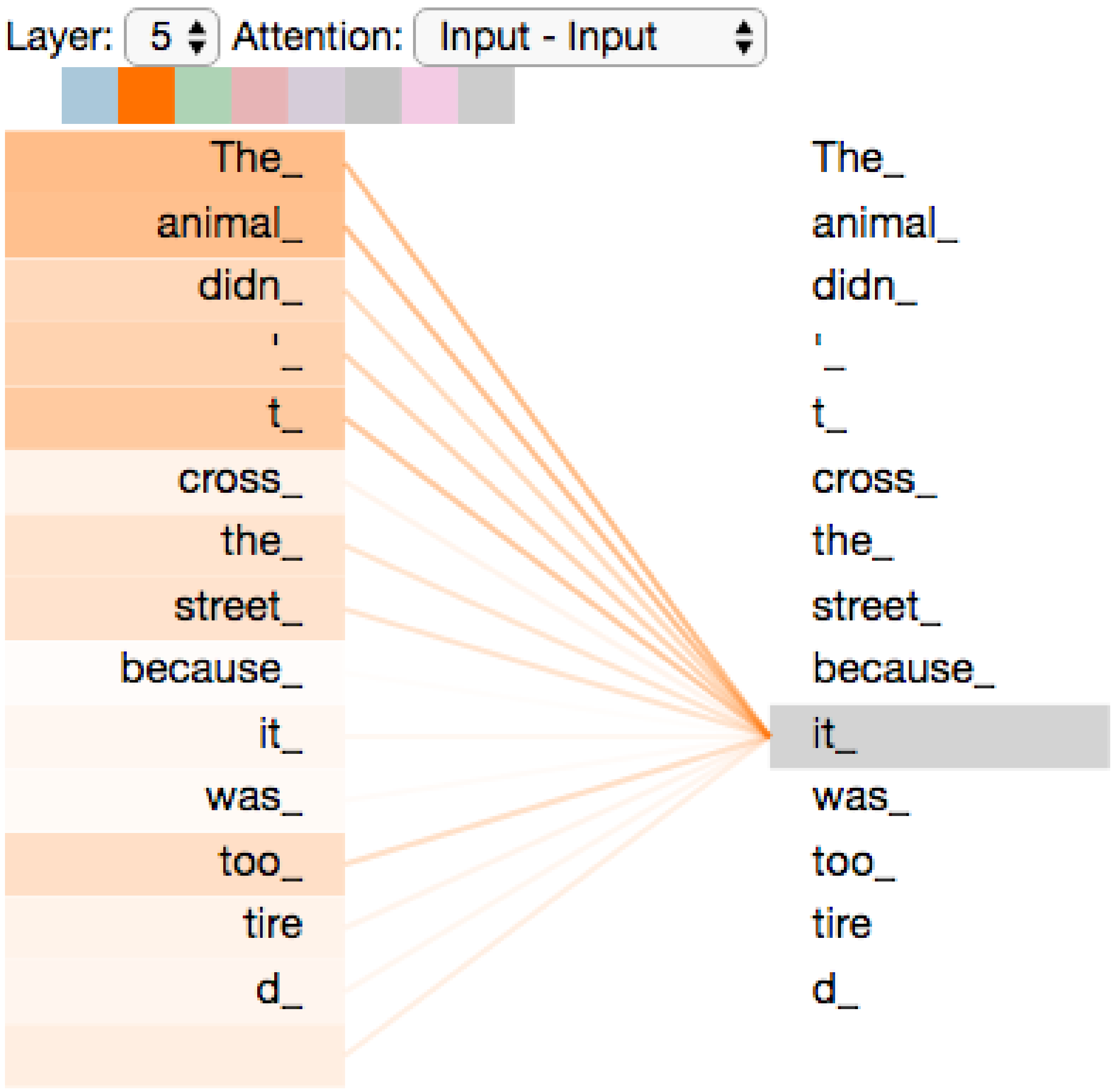
Neural Attention Models for Sequence Classification: Analysis and Application to Key Term Extraction and Dialogue Act Detection

S. Shen, Hung-yi Lee

Multi-head Key-Value Self Attention

Self-attention (single-head, high-level)

"`The animal didn't cross the street because it was too tired`"



There is no external query q.

The input is also the query.

Many approaches:

https://ruder.io/deep-learning-nlp-best-practices/

Transformers: query q is another $x_{j:}$ $\phi^{att}(x_j,x_i)$

Attention: Encoding (h → x)

$$c = \sum_{i=1}^{T} \alpha_i . x_i$$

$$\alpha = \mathrm{softmax}(\bar{\alpha}_1, \dots, \bar{\alpha}_T)$$

$$\bar{\alpha}_i = \phi^{\mathrm{att}}(q, x_i)$$

Attention: Encoding

$$c = \sum_{i=1}^{T} \alpha_i . x_i$$

$$\alpha = \text{softmax}(\bar{\alpha}_1, \dots, \bar{\alpha}_T)$$

$$\bar{\alpha}_i = \phi^{\text{att}}(q, x_i)$$

Each vector (x) playing two roles

(1) computing importance
(2) weighted sum

Key-Value Attention

- Project an input vector x_i into two vectors
  k: key vector $k_i = W^K x_i$
  v: value vector $v_i = W^V x_i$

- Use key vector for computing attention

  $\phi^{att}(q, x_i) = \phi^{att}(q, k_i) = \frac{k_i . q}{\sqrt{d}}$ //scaled multiplicative

- Use value vector for computing attended summary

$$c = \sum_{i=1}^{T} \alpha_i . v_i$$

Key-Value Single-Head Self Attention

- Project an input vector x_i into three vectors
  k: key vector: $k_i = W^K x_i$
  v: value vector: $v_i = W^v x_i$
  q: query vector: $q_i = W^Q x_i$

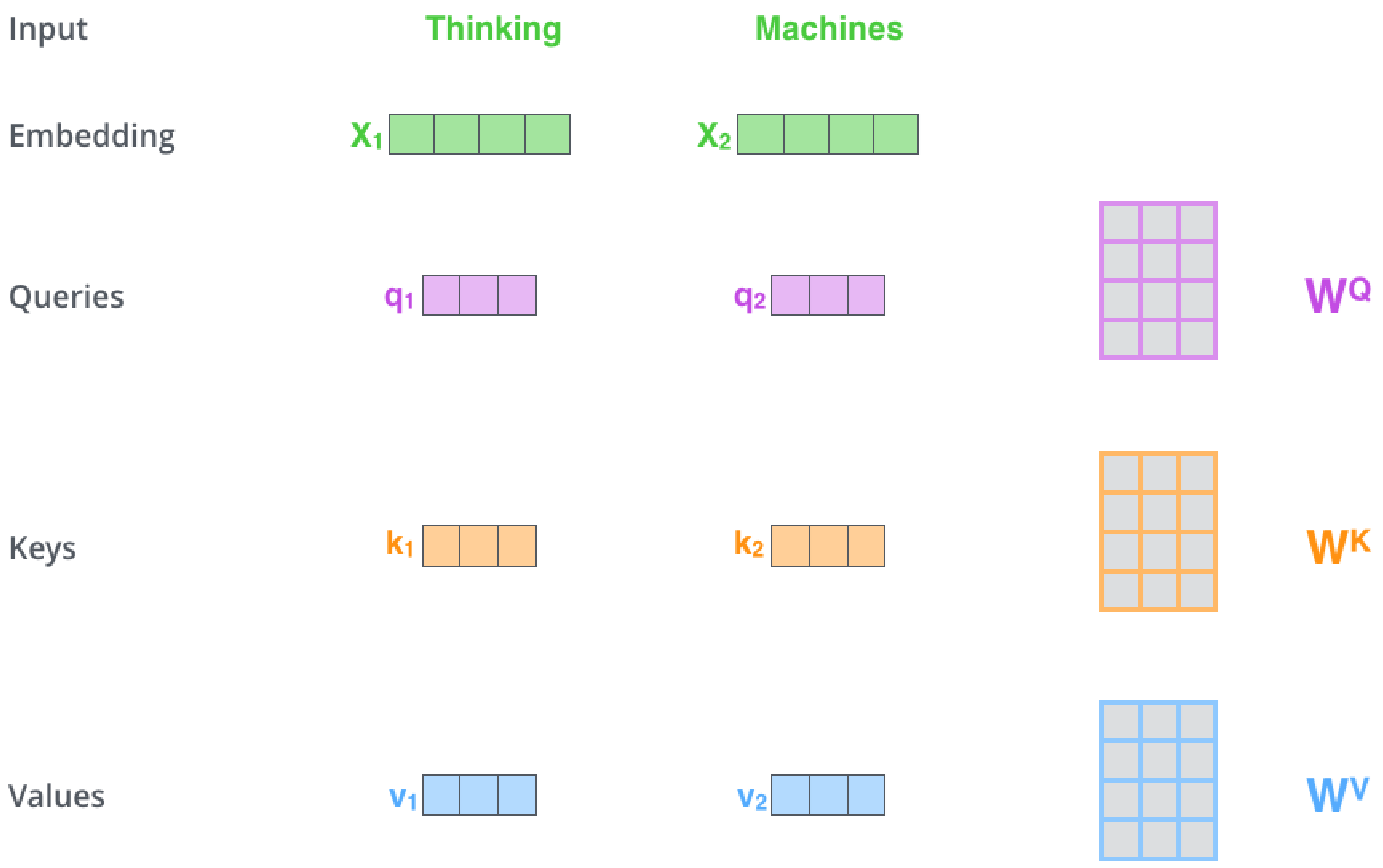
- Use key and query vectors for computing attention of i^{th} word at word j

  $\phi^{att}(x_j; x_i) = \frac{k_i . q_j}{\sqrt{d}}$ //scaled multiplicative

- Use value vector for computing attended summary

$$c^j = \sum_{i=1}^{T} \alpha_i . v_i$$

Key-Value Single-Head Self Attention



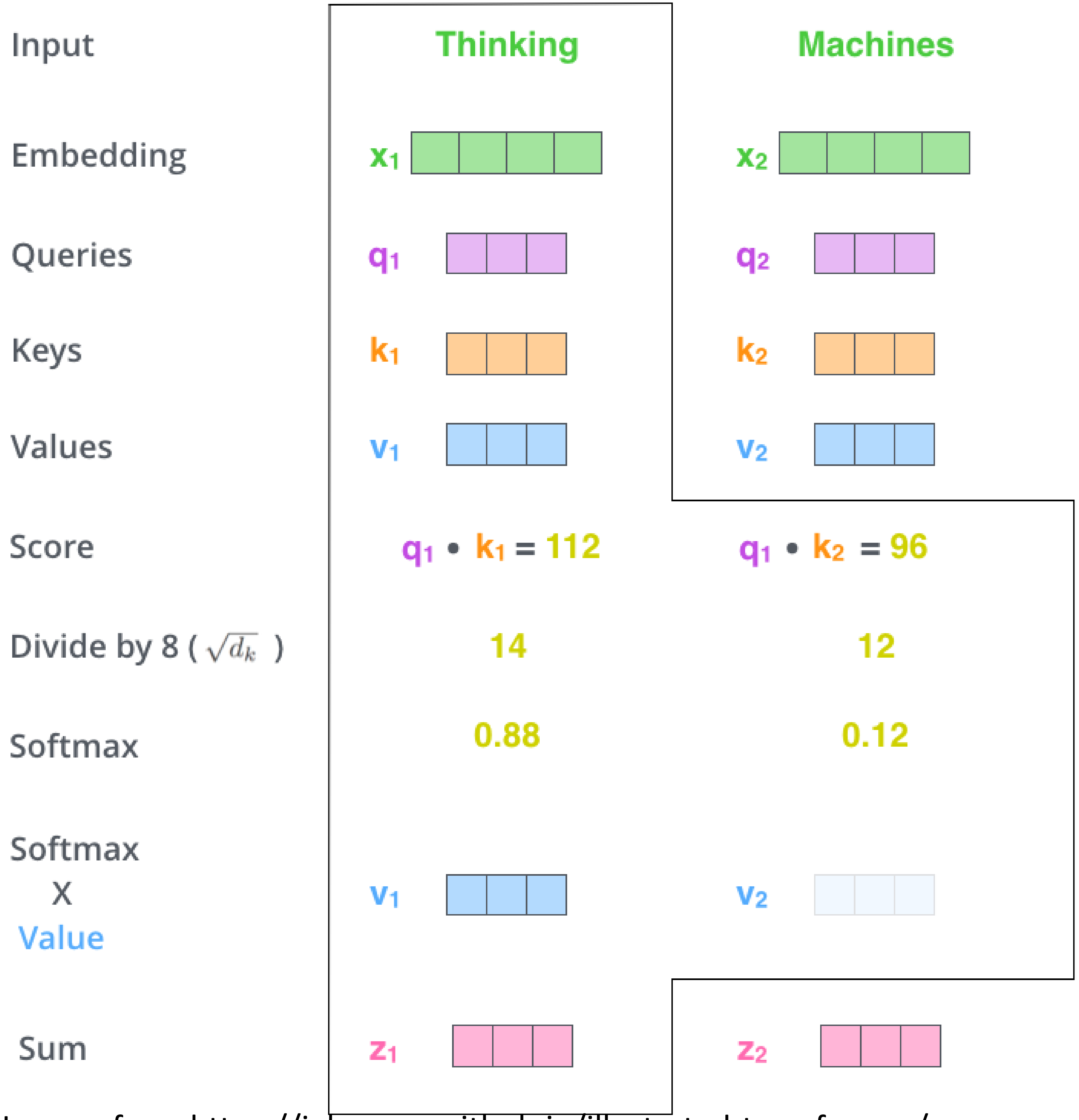
Creation of query, key and value vectors by multiplying by trained weight matrices

Separation of Value and Key and Query

Matrix multiplications are quite efficient and can be done in aggregated manner

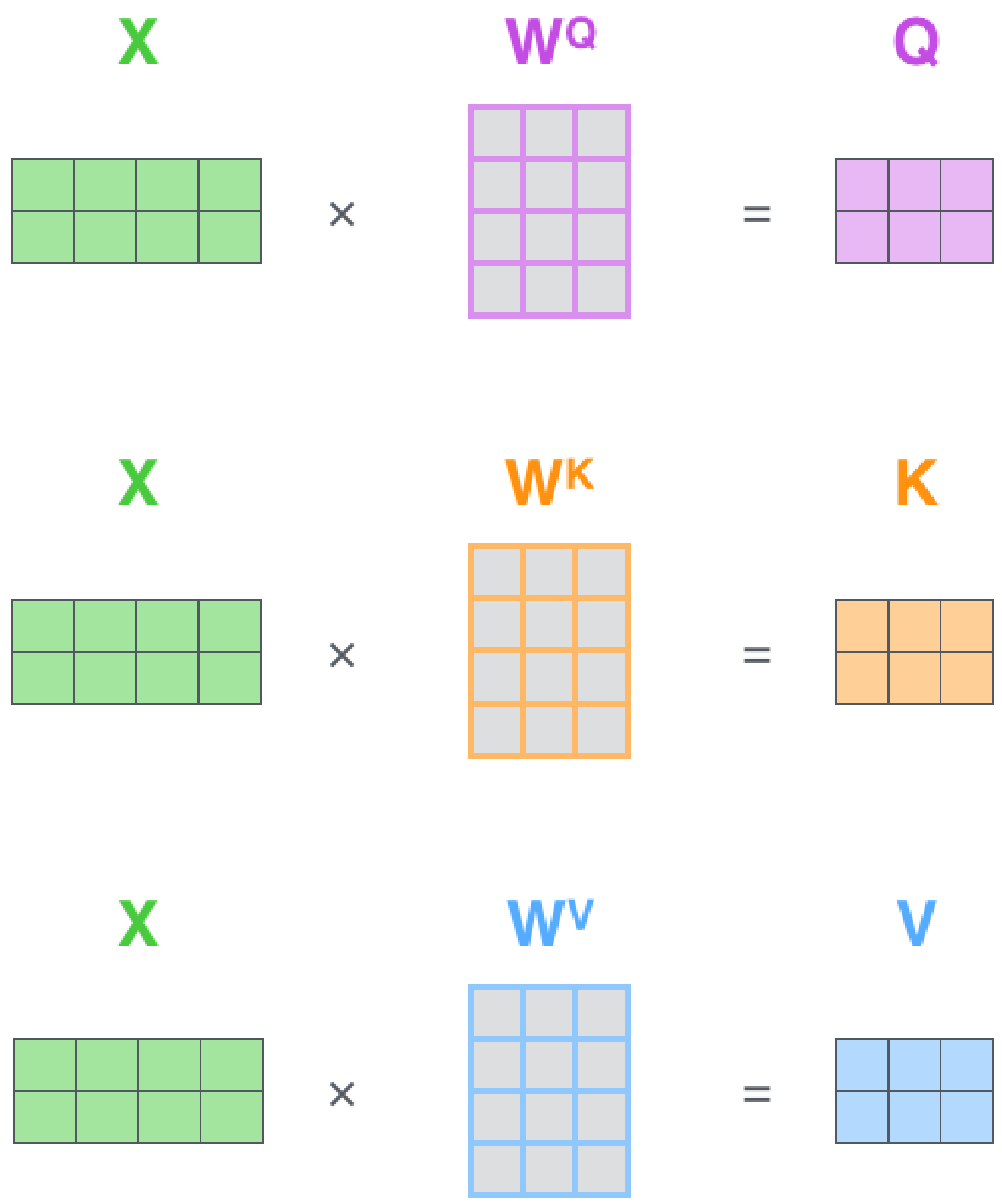
Images from https://jalammar.github.io/illustrated-transformer/

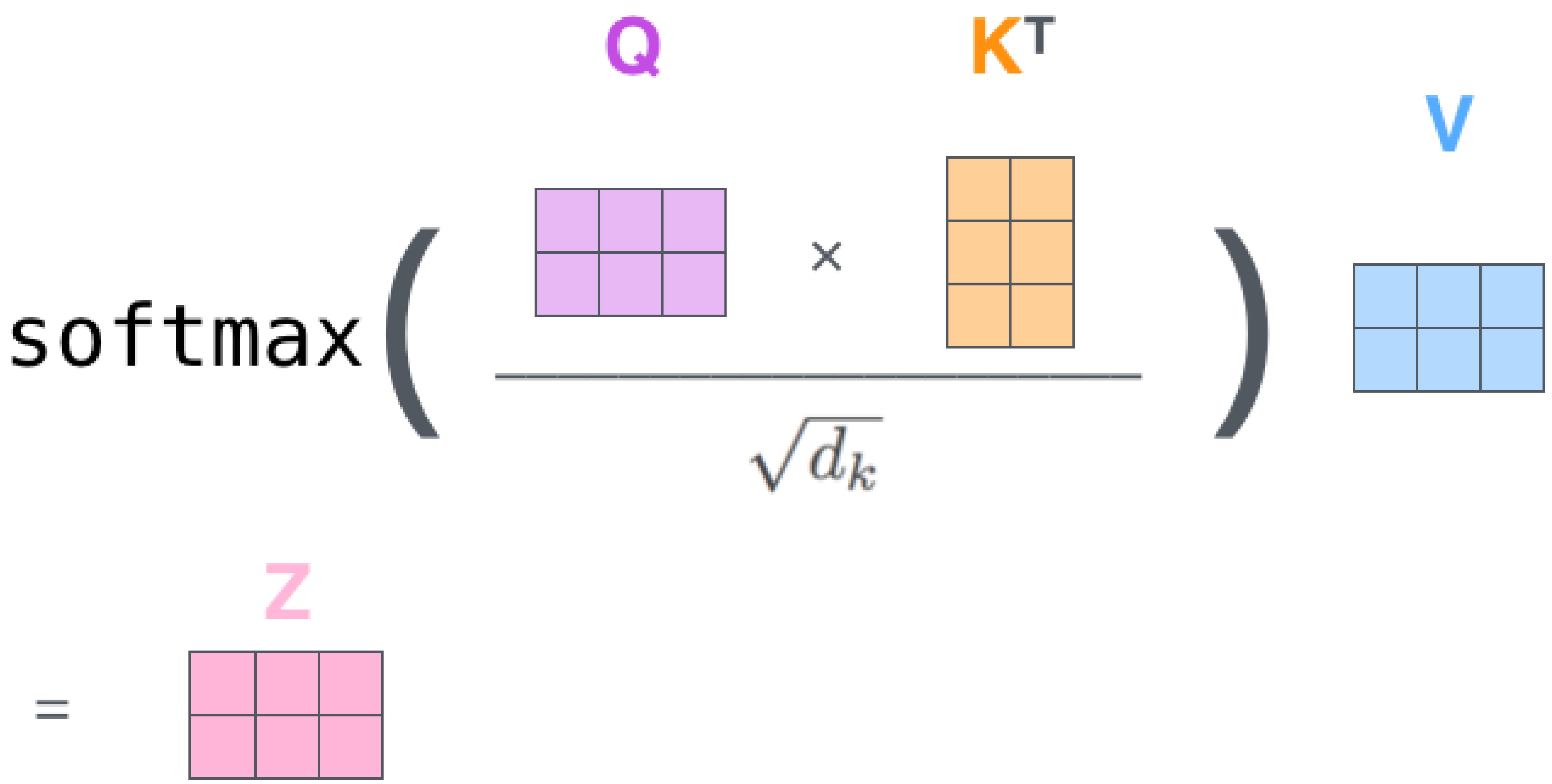
Key-Value Single-Head Self Attention



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

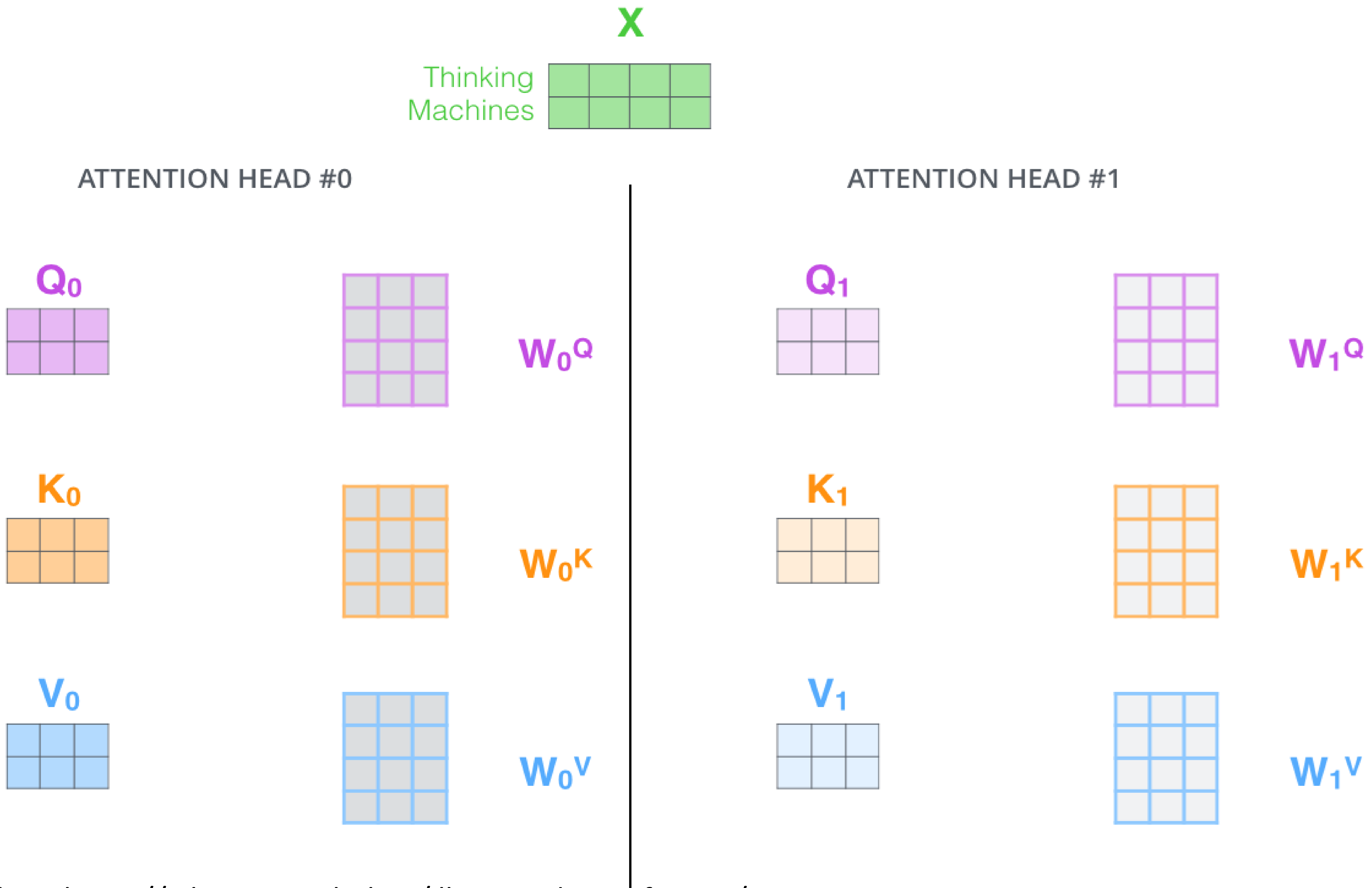
Key-Value Single-Head Self Attention





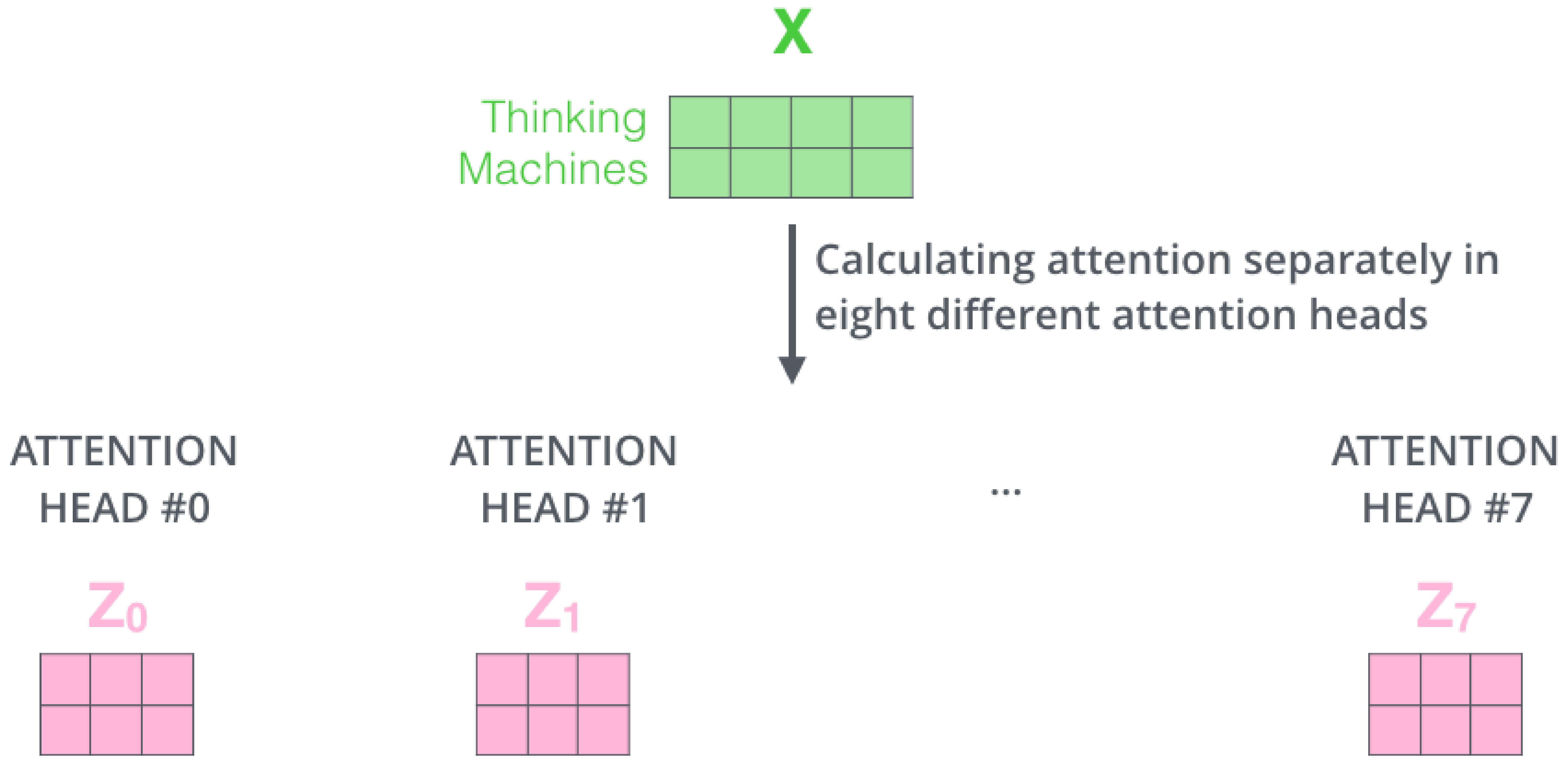
Images from https://jalammar.github.io/illustrated-transformer/

Key-Value Multi-Head Self Attention



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

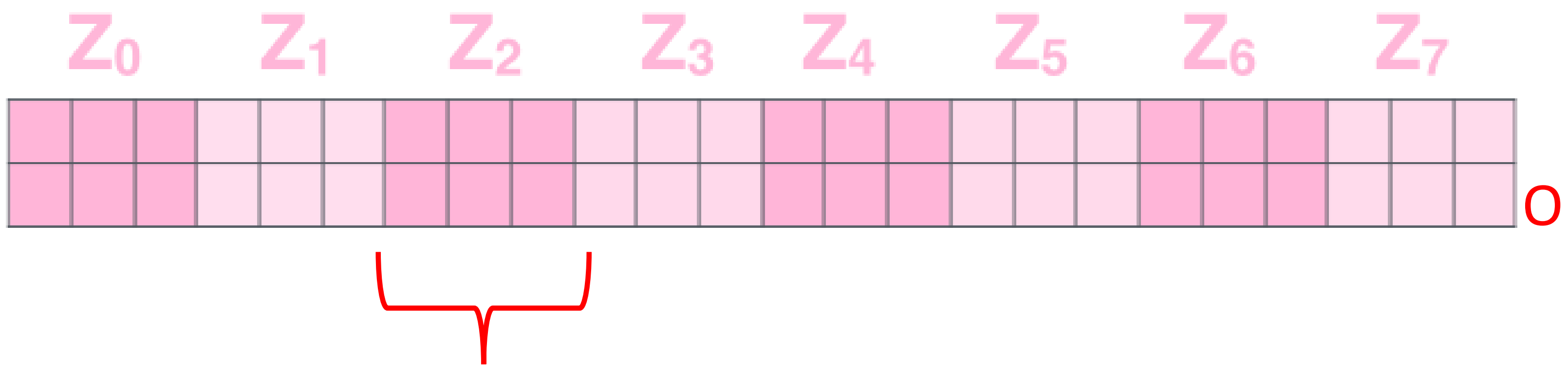
Multi-Head Attention



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

Multi-Head Attended Vector → Output

1) Concatenate all the attention heads

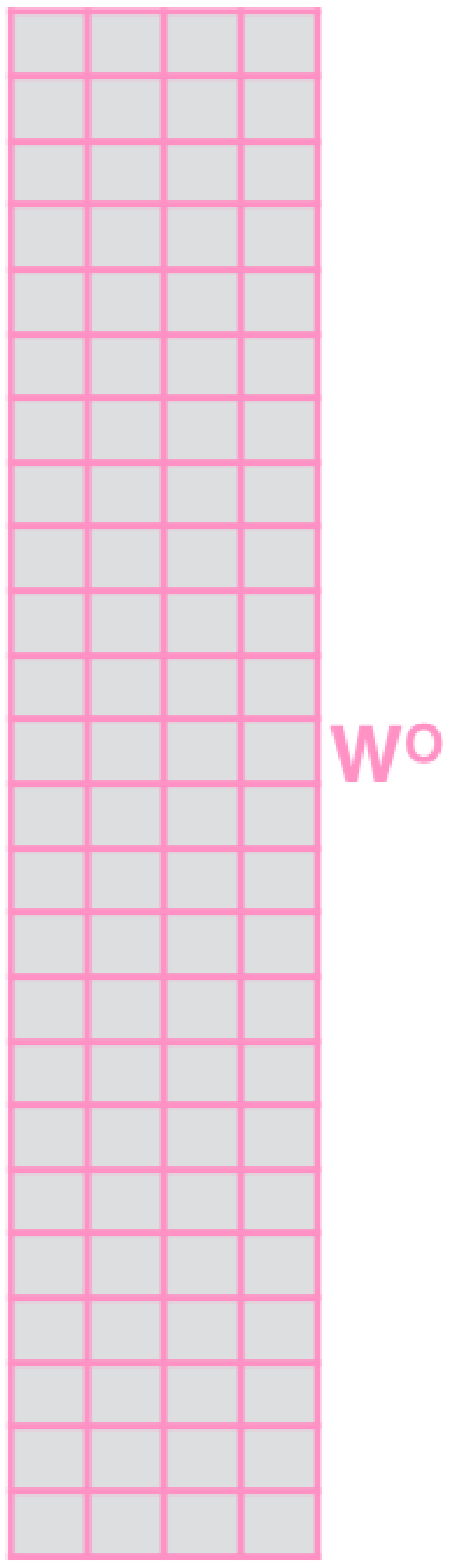


One for each word

One for each attention head

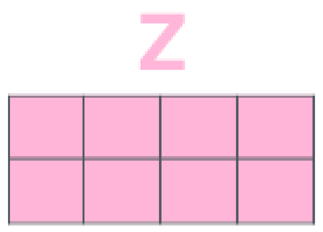
2) Multiply with a weight matrix W^O that was trained jointly with the model

X



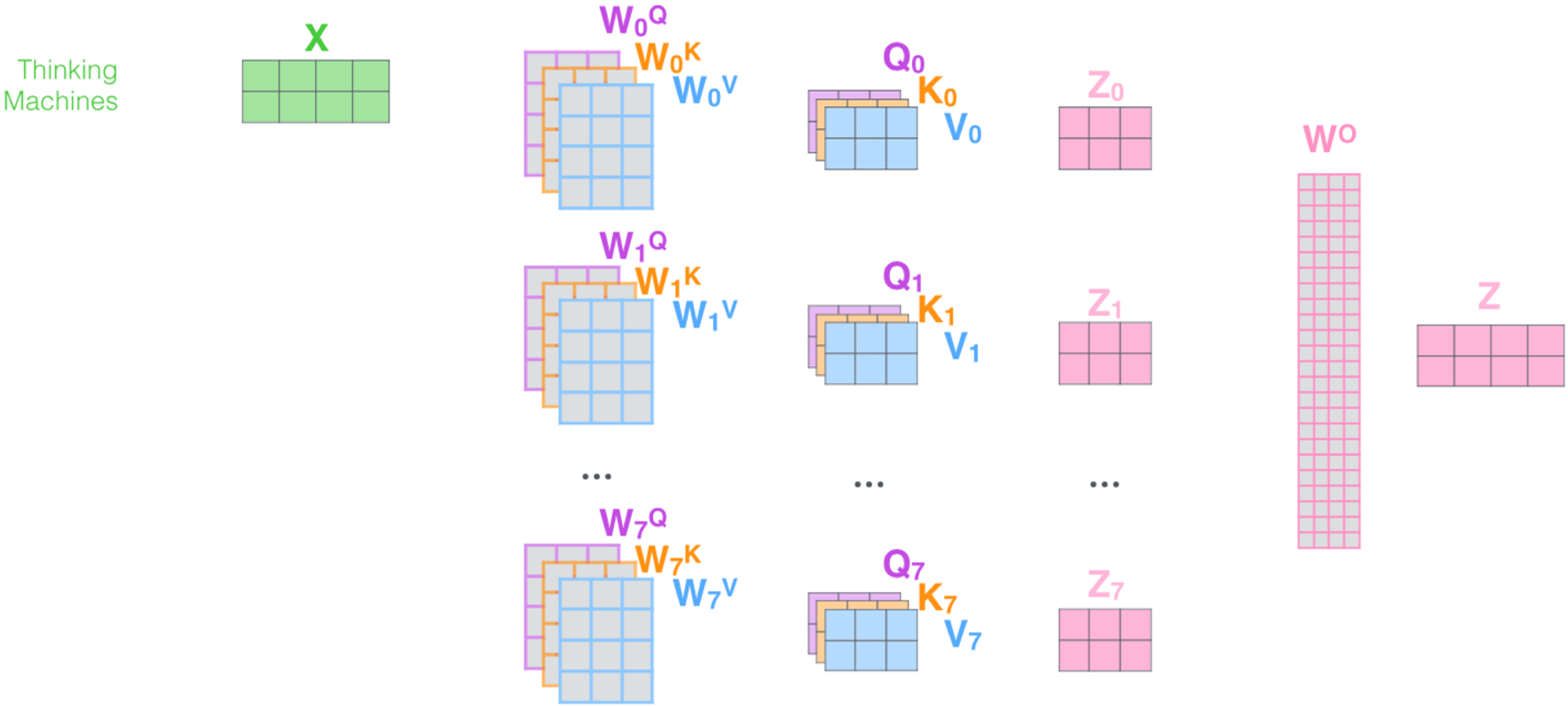
3) The result would be the Z matrix that captures information from all the attention heads. We can send this forward to the FFNN

=



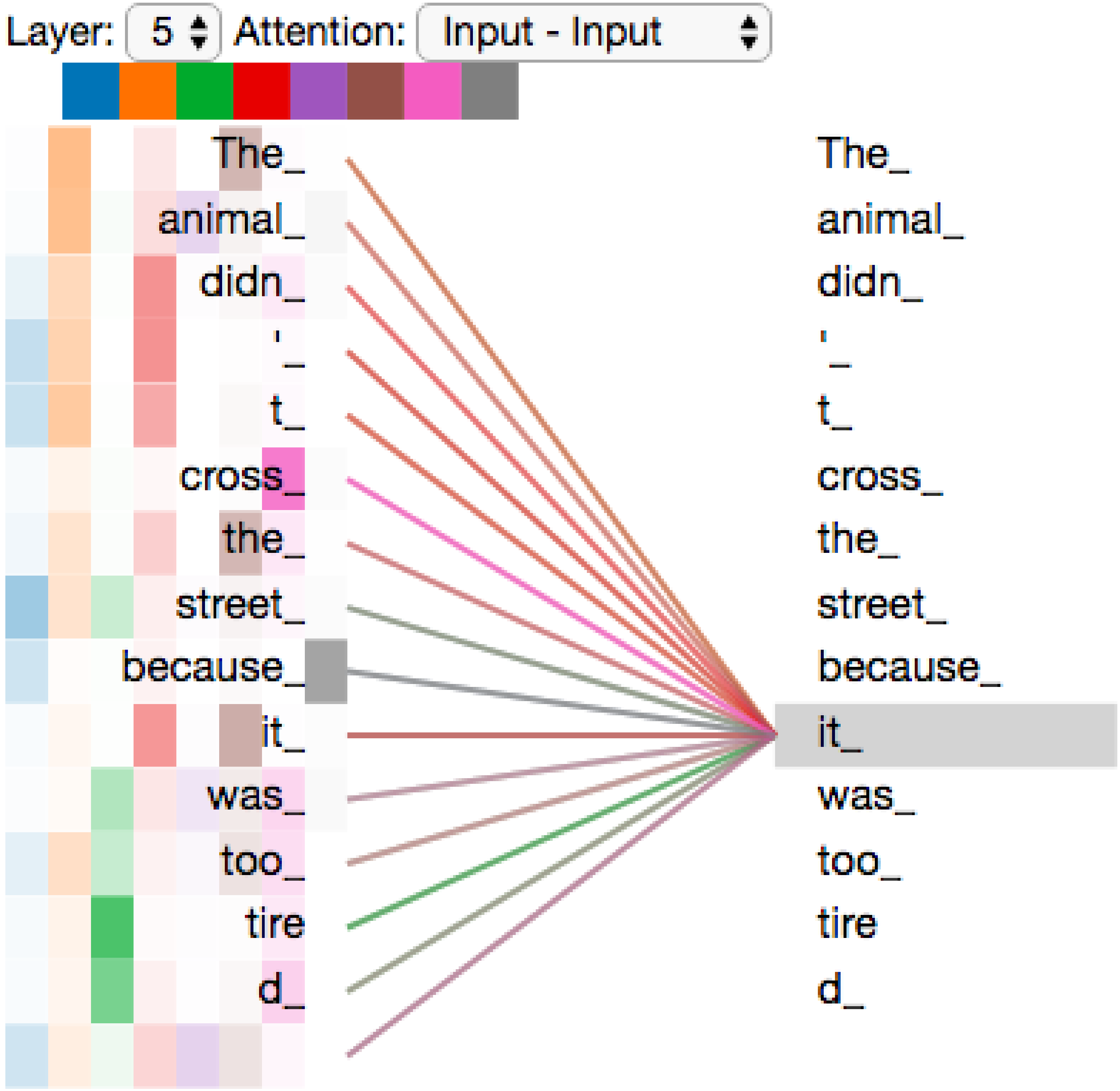
Images from https://jalammar.github.io/illustrated-transformer/

Key-Value Multi-Head Self Attention (summary)



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

Multi-head Self attention visualisation (Interpretable?!)



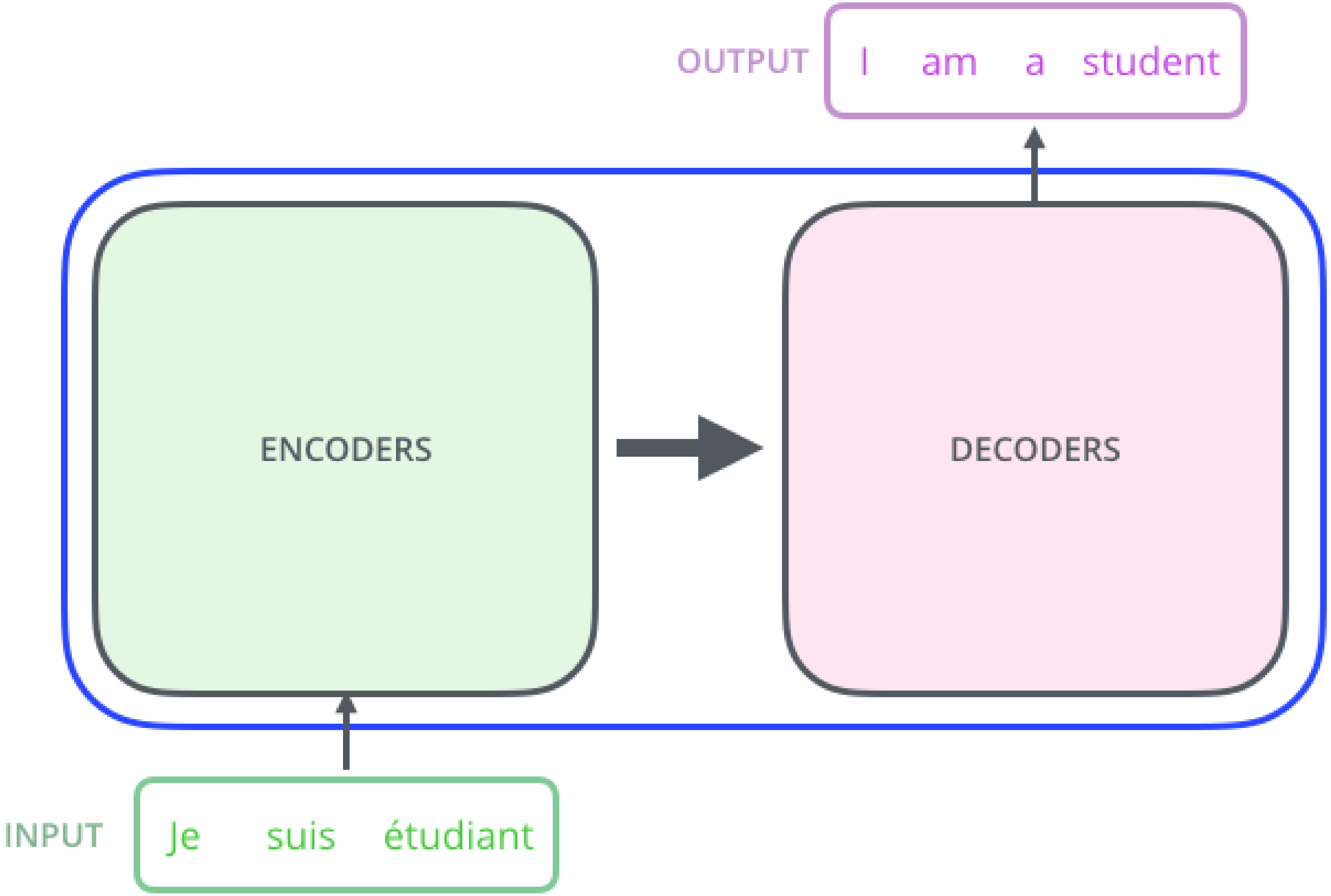
Images from https://jalammar.github.io/illustrated-transformer/

Transformer Encoders

Motivation

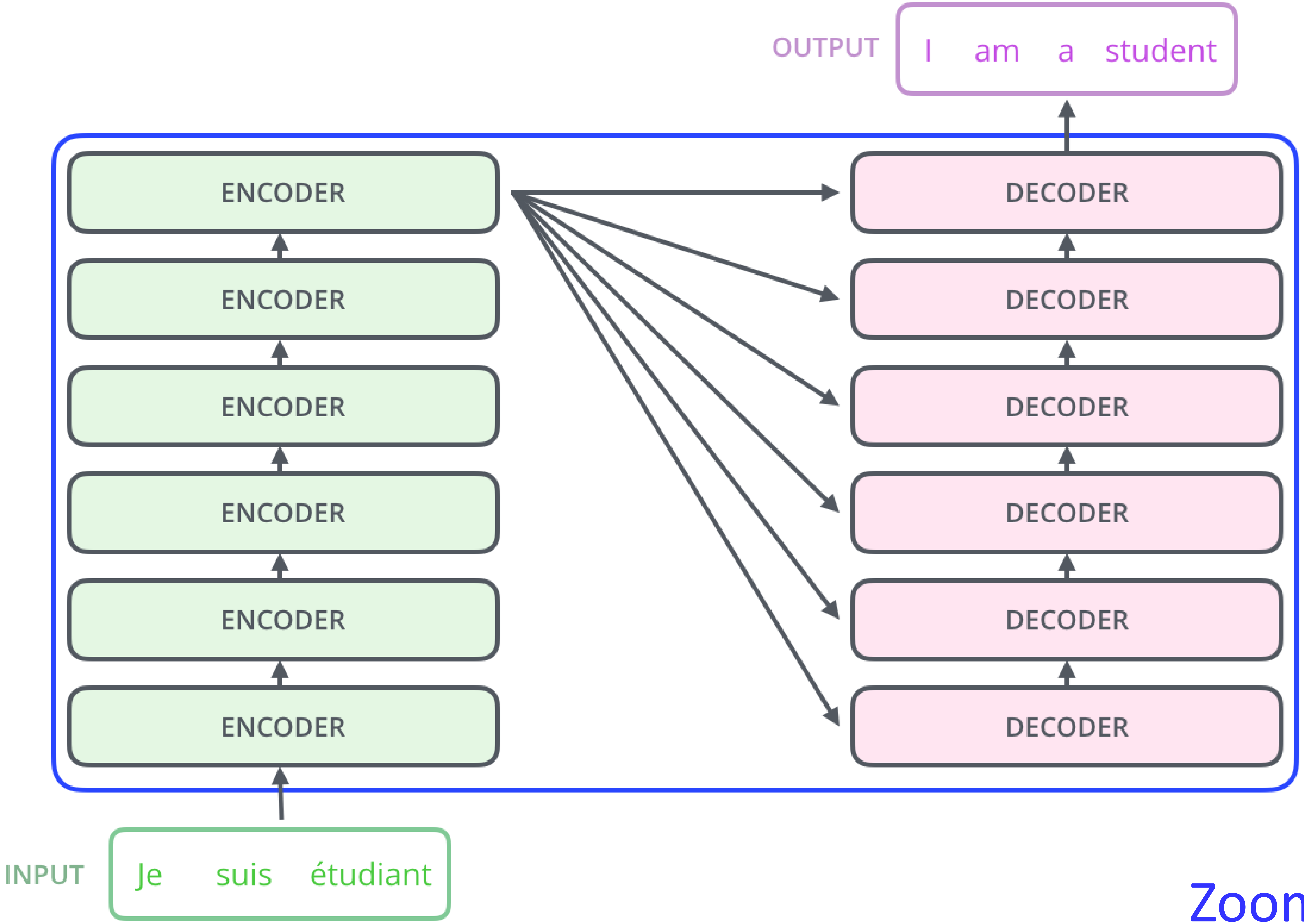
- Recurrence is powerful but
  - Issues with learnability: vanishing gradients
  - Issues with remembering long sentences
  - Issues with scalability:
    - backpropagation time high due to sequentiality in sentence length
  - Issues with scalability:
    - can’t be parallelized even at test time – O(sentence length)

- Remove recurrence: only use attention
  “Attention is All You Need”



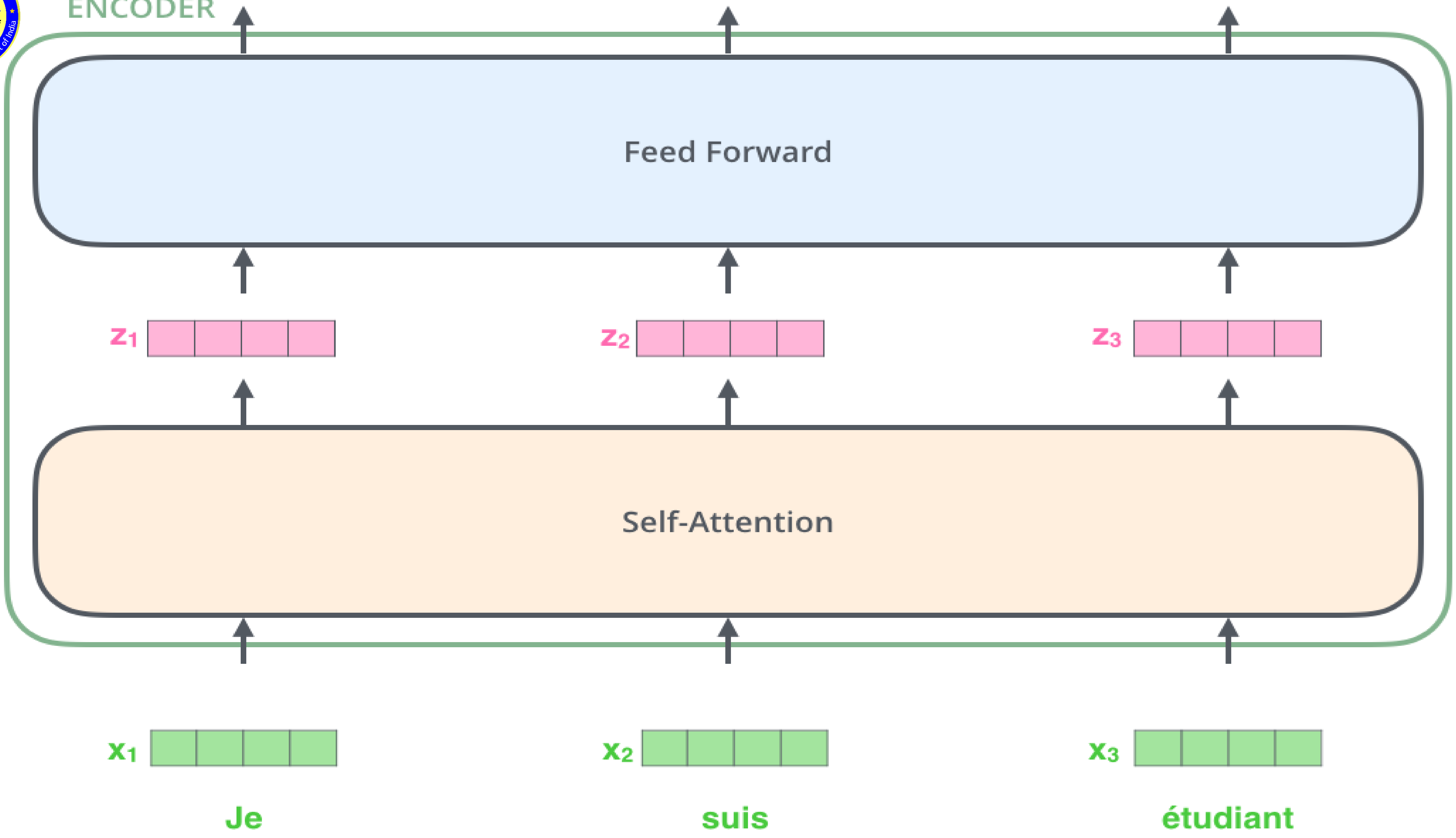
We focus only on encoder for now… (decoder is an extension of sequence decoders)

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



Zooming in...

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



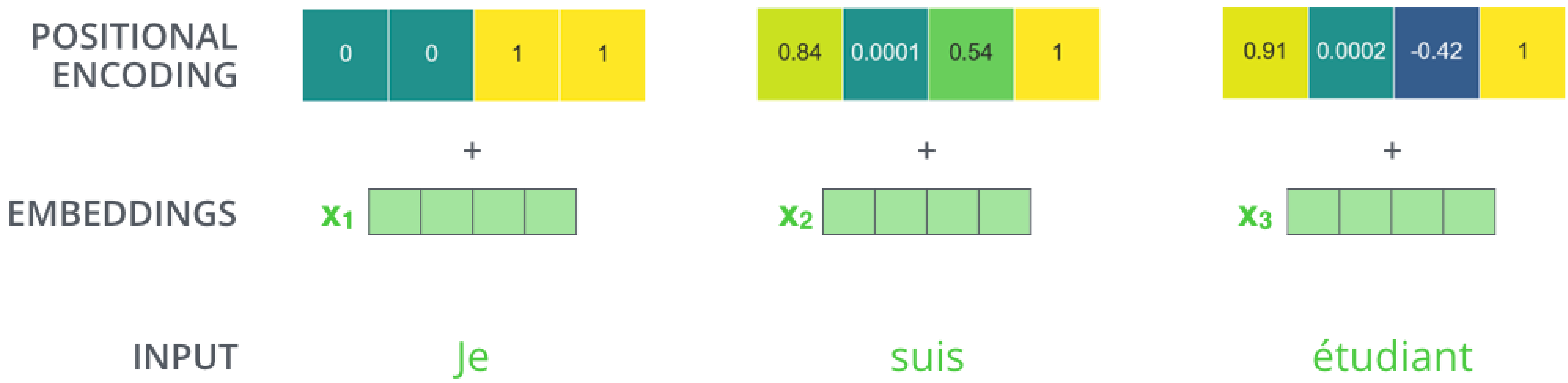
Can you see a fundamental limitation?

Encoders have same architecture but different weights…

Zooming in further…

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

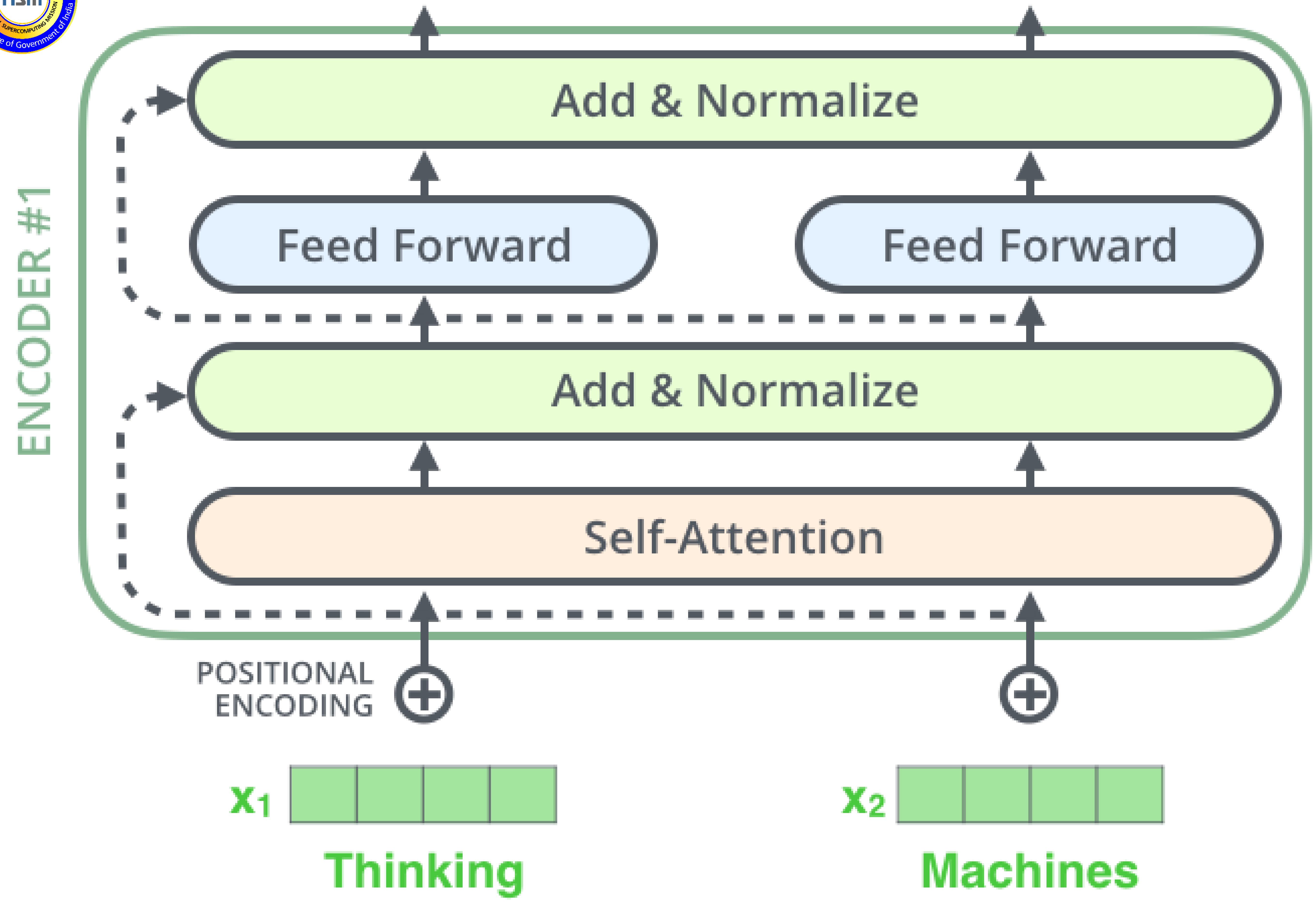
A note on Positional embeddings



Positional embeddings can be extended to any sentence length but if any test input is longer than all training inputs then we will face issues.

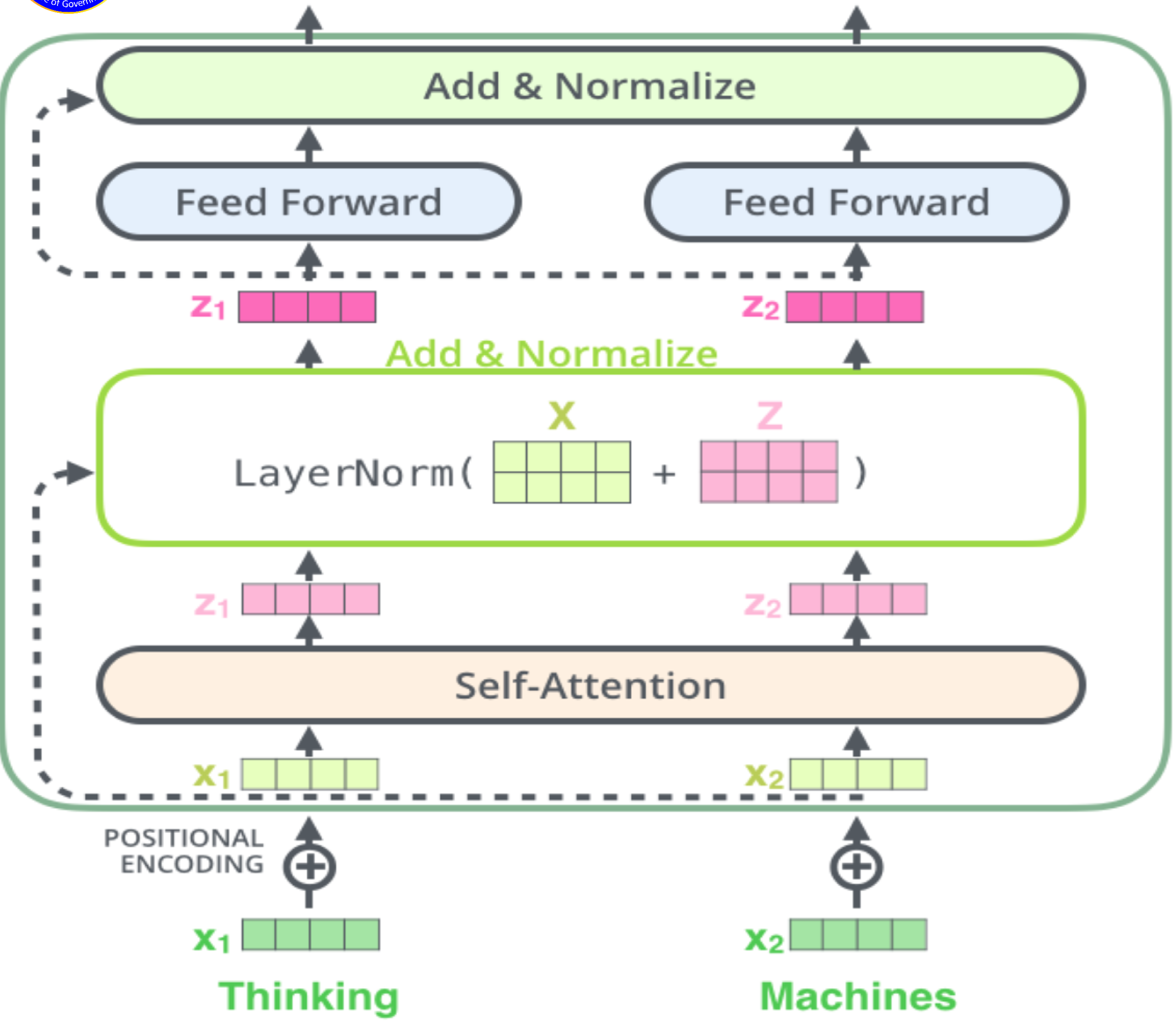
Solution: use a functional form (as in Transformer paper – sinuisoidal encoding)

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



Adding residual connections...

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



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

The residual connections help the network train, by allowing gradients to flow through the networks directly.

The layer normalizations stabilize the network -- substantially reducing the training time necessary.

$$z = \text{LayerNorm}(x + z) = \gamma \frac{x + z - \mu}{\sigma} + \beta$$

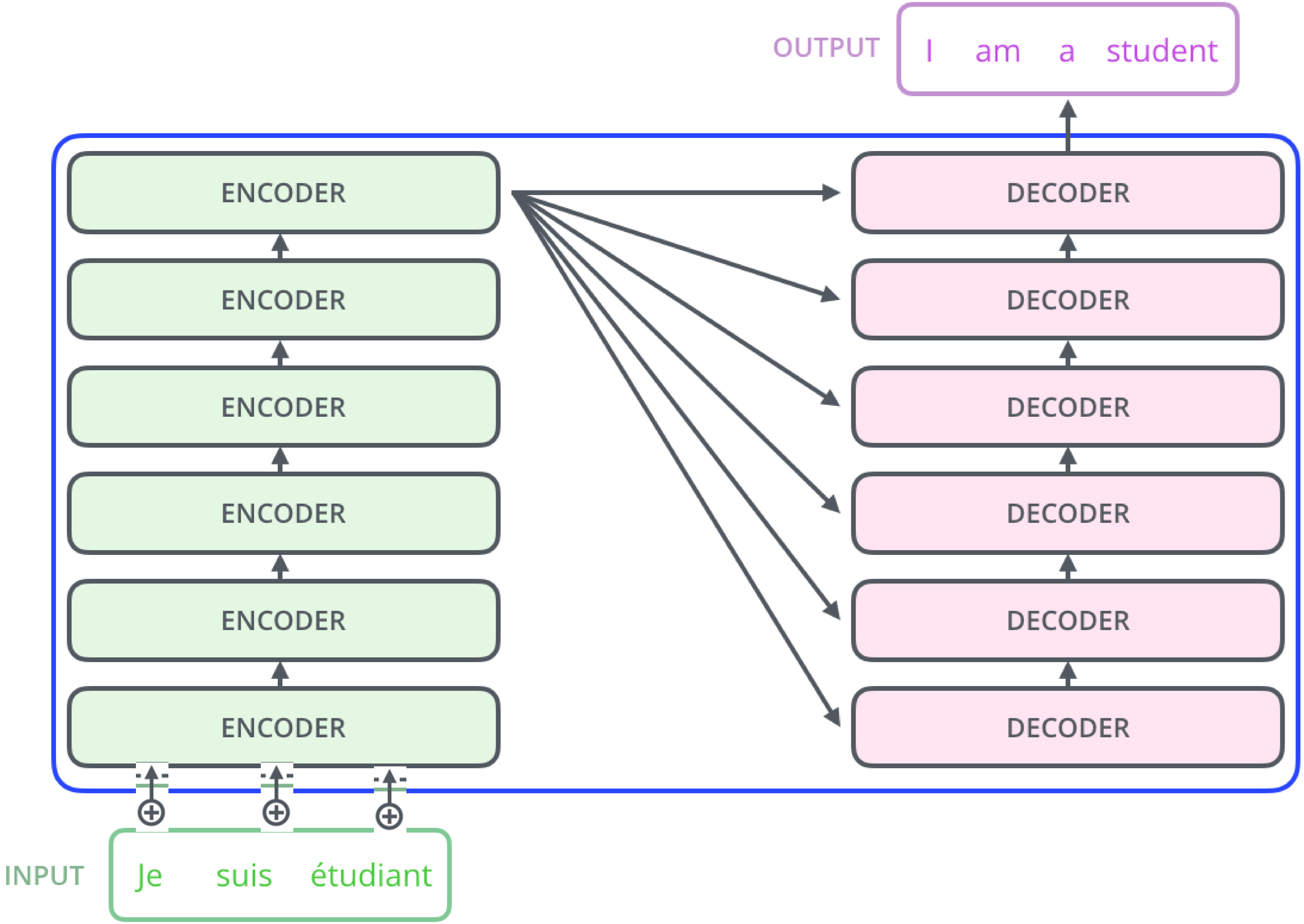
The pointwise feedforward layer is used to project the attention outputs potentially giving it a richer representation.

Regularization

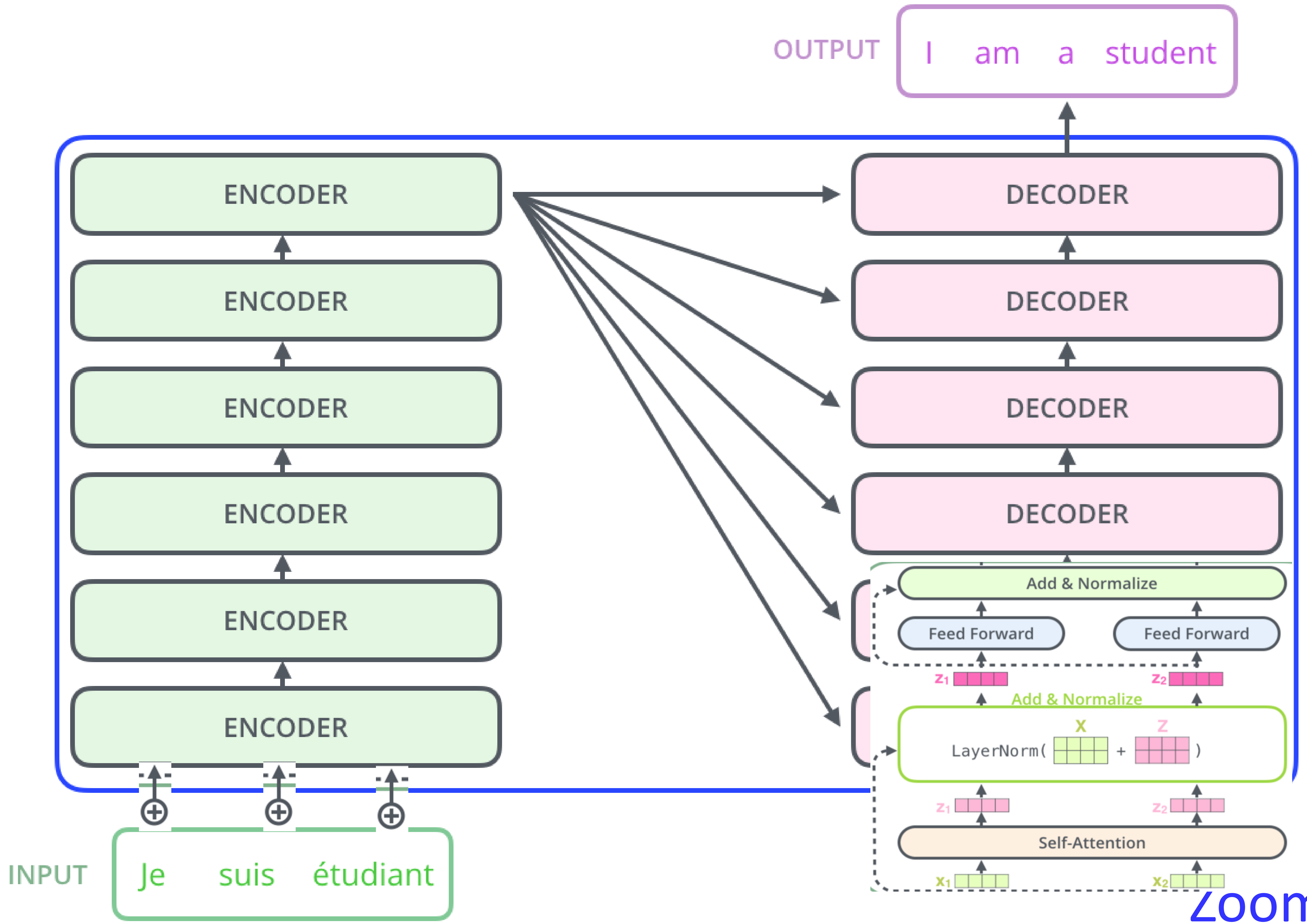
Residual dropout: Dropout added to the the output of each sublayer, before it is added to the input of the sublayer and normalized

Label Smoothing: During training label smoothing was employed. This hurts perplexity, as the model learns to be more unsure, but improves accuracy and BLEU score. (skip for now)

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

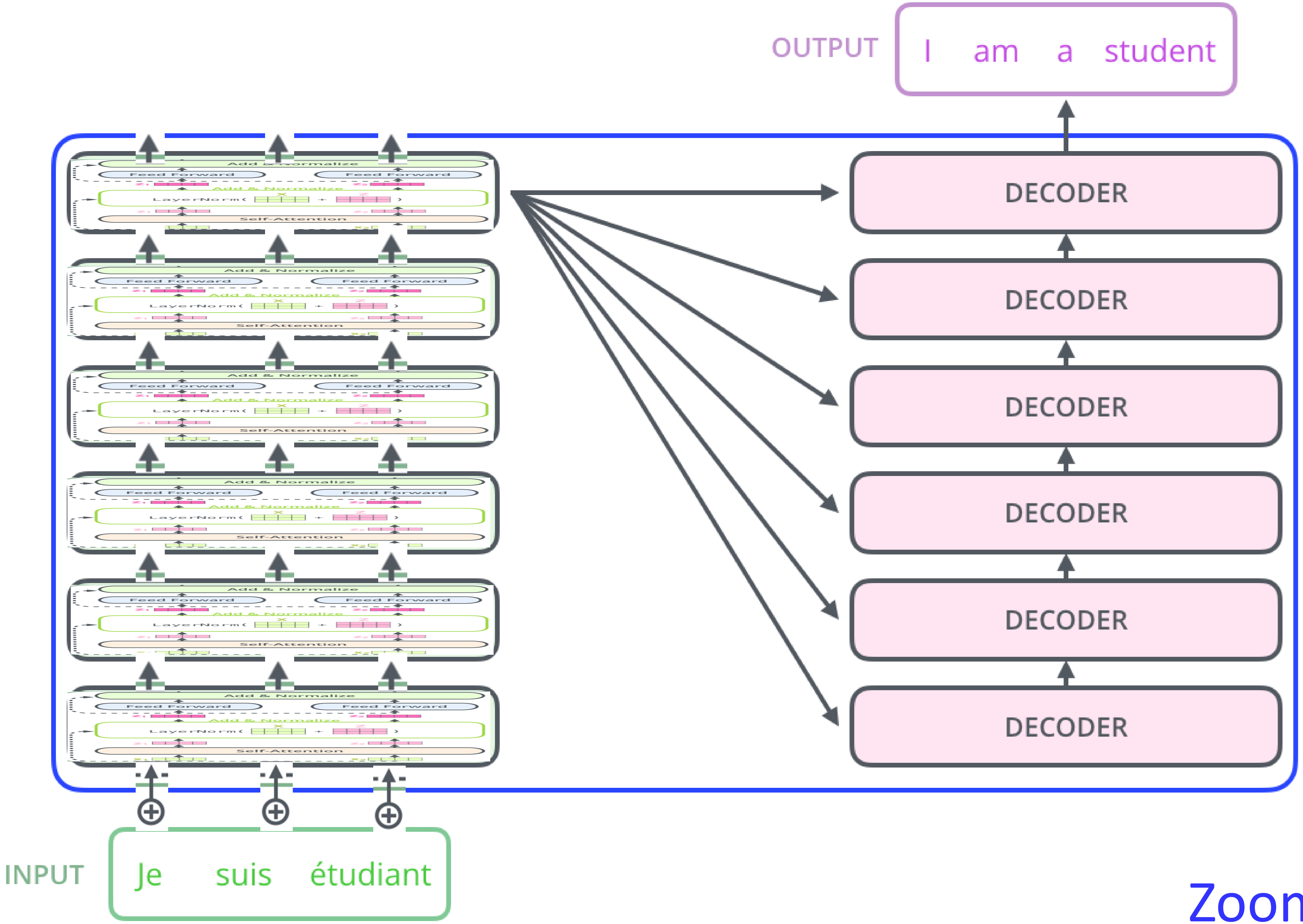


Images from https://jalammar.github.io/illustrated-transformer/



Zooming in...

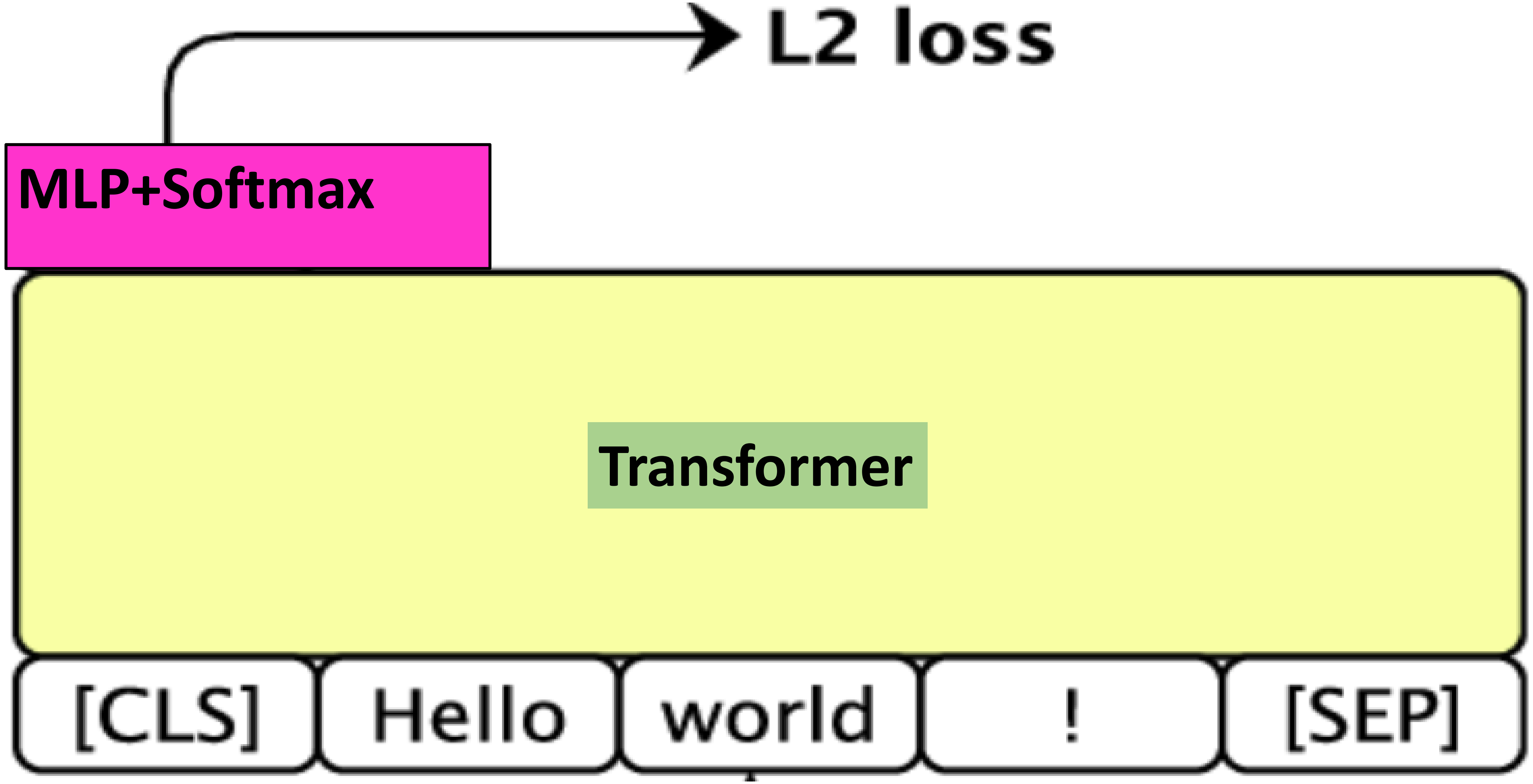
Images from https://jalammar.github.io/illustrated-transformer/



Zooming in...

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

Use of [CLS] for Text Classification



Pros

- Current state-of-the-art.
- Enables deep architectures
- Easier learning of long-range dependencies
- Can be efficiently parallelized
- Gradients don’t suffer from vanishing gradients

Cons

Huge number of parameters so

- Very data hungry
- Takes a long time to train
- Memory inefficient

Other issues

- Keeping sentence length limited
- How to ensure multi-head attention has diverse perspectives.