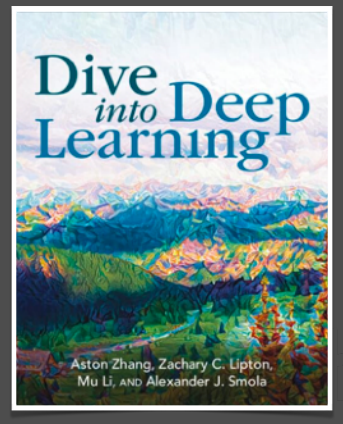
ROBOT INTELLIGENCE LAB

Introduction to Deep Learning

Attention and Transformers

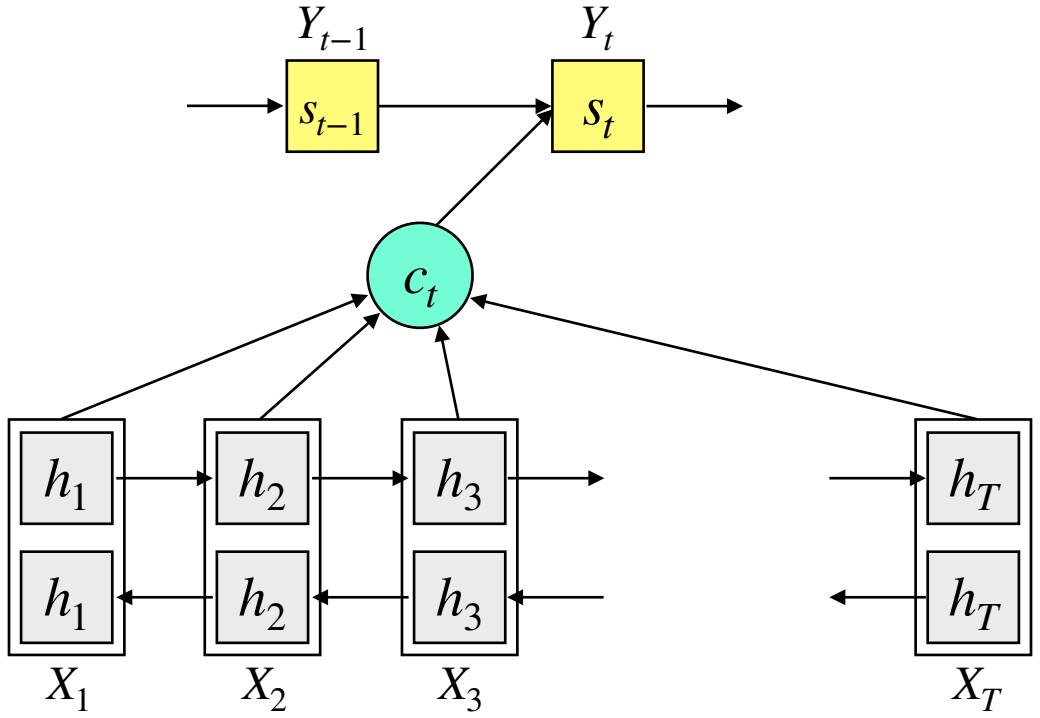
Sungjoon Choi, Korea University



ROBOT INTELLIGENCE LAB

Attention Mechanism

Learning to Align and Translate



- The decoder: $p(y_t | y_1, \ldots, y_{t-1}, x_{1:T}) = g(y_{t-1}, s_{t-1}, c_t)$ where

$$c_i = \sum_{j=1}^{T} \alpha_{ij} h_j, \; \alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T} \exp(e_{ik})}, \text{ and } e_{ij} = a(s_{i-1}, h_j).$$

- Note that e_{ij} is an **alignment model** which scores how well the inputs around position j and the output at position i match.

Queries, Keys, and Values

ROBOT INTELLIGENCE LAB

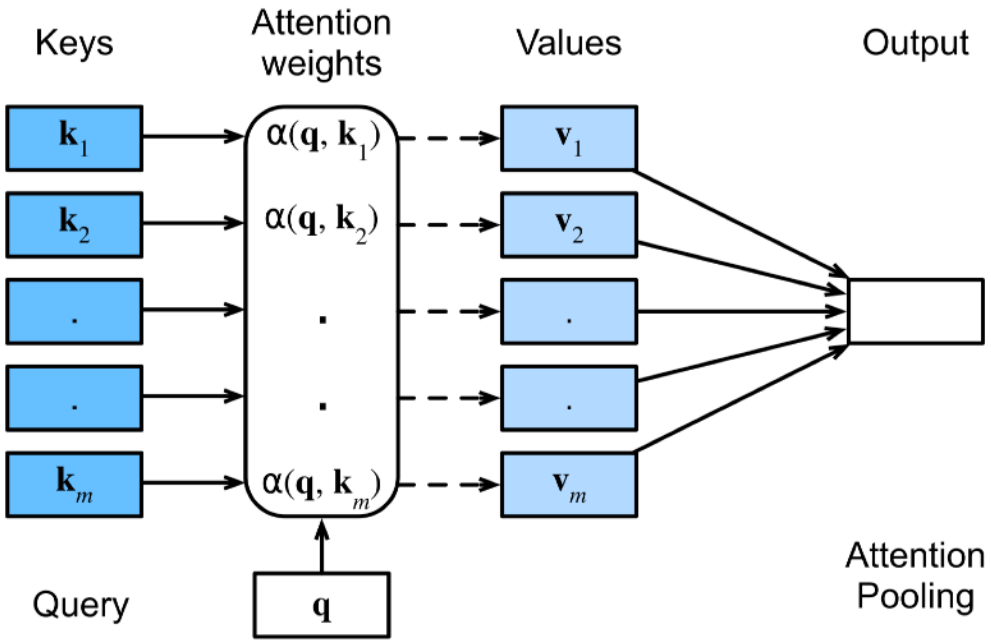
- Our main focus is on handling varying sizes with varying information content.
- In the simplest form, the databases are collections of keys (k) and values (v).
- The following are some desired properties:
  - We can design queries q that operate on (k, v) pairs in such a manner as to be valid regardless of the database size (varying sizes).
  - The same query can receive different answers according to the contents of the database.
- Consider the following: denote by $\mathcal{D} = \{(k_1, v_1), \ldots, (k_m, v_m)\}$ a database with m tuples of keys and values. denote by q a query.
- Then, the QKV attention is defined as

$$\text{Attention}(q; \mathcal{D}) = \sum_{i=1}^{m} \alpha(q, k_i) v_i.$$

where $\alpha(q, k)_i \in \mathbb{R}$ are scalar attention weights.

- The operation itself is typically referred to as **attention pooling**.

Queries, Keys, and Values

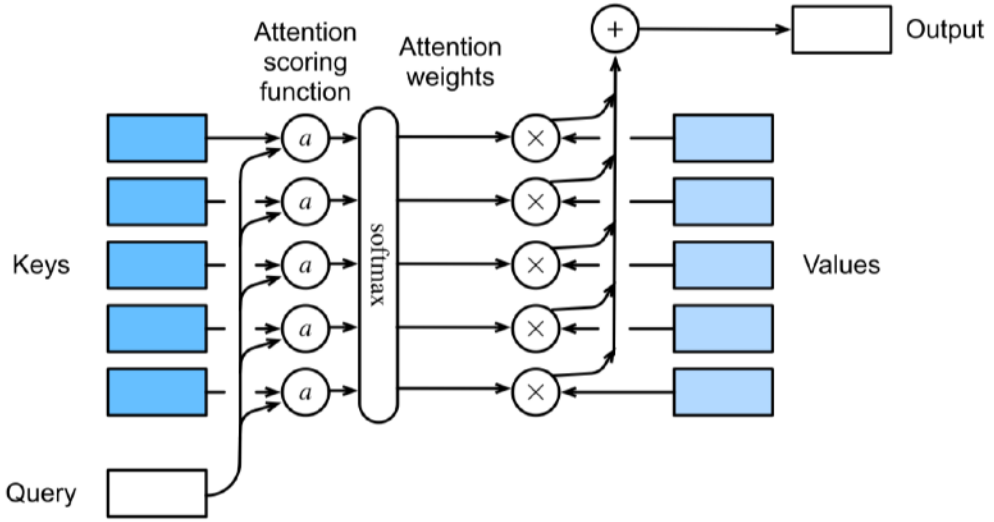


- A common strategy to ensure the weights are nonnegative and sum up to one is via:

$$\alpha(q, k_i) = \frac{\exp(a(q, k_i))}{\sum_j \exp(a(q, k_j))}$$

where we can use any function $a(q, k) \in \mathbb{R}$ for the attention scoring function with the softmax operation.

Dot Product Attention

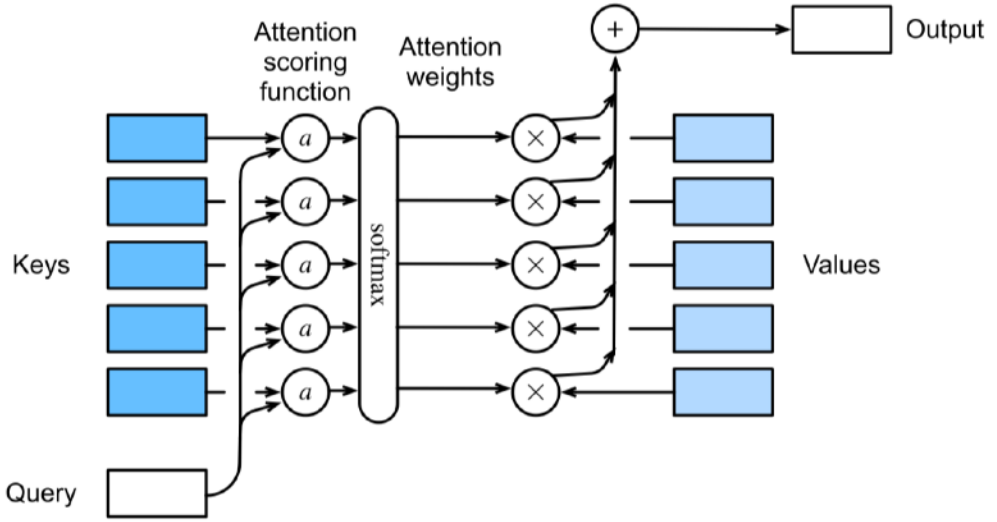


- Let's review the attention scoring function (before the softmax) from the Gaussian kernel:

$$a(q, k_i) = -\frac{1}{2}\|q - k_i\|^2 = q^T k_i - \frac{1}{2}\|k_i\|^2 - \frac{1}{2}\|q\|^2$$

  - First, note that the last term ($\|q\|^2$) depends on q only; hence, it may not be necessary for the purpose of querying.
  - Second, if batch and layer normalization are used, $\|k\|^2 \approx$ const., hence, this term may be dropped.
- Hence, we may simply use the inner product, $a(q, k_i) = q^T k_i$ with the benefit of computational efficiency.

Dot Product Attention



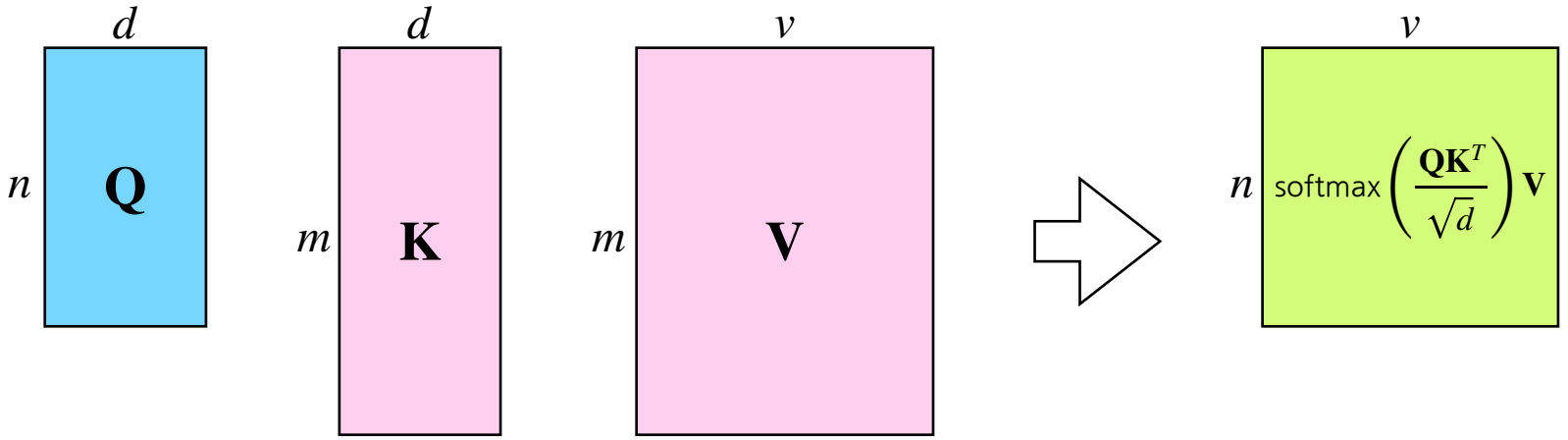
- As the dot-product between two d-dimensional random vectors with zero mean and unit variance has zero mean and variance d, we may rescale the dot-product by $1/\sqrt{d}$:

$$a(q, k_i) = q^T k_i / \sqrt{d}$$

- Combining all together, we have the following attention weights:

$$\alpha(q, k_i) = \text{softmax}(a(q, k_i)) = \frac{\exp(q^T k_i / \sqrt{d})}{\sum_{j=1} \exp(q^T k_j / \sqrt{d})}$$

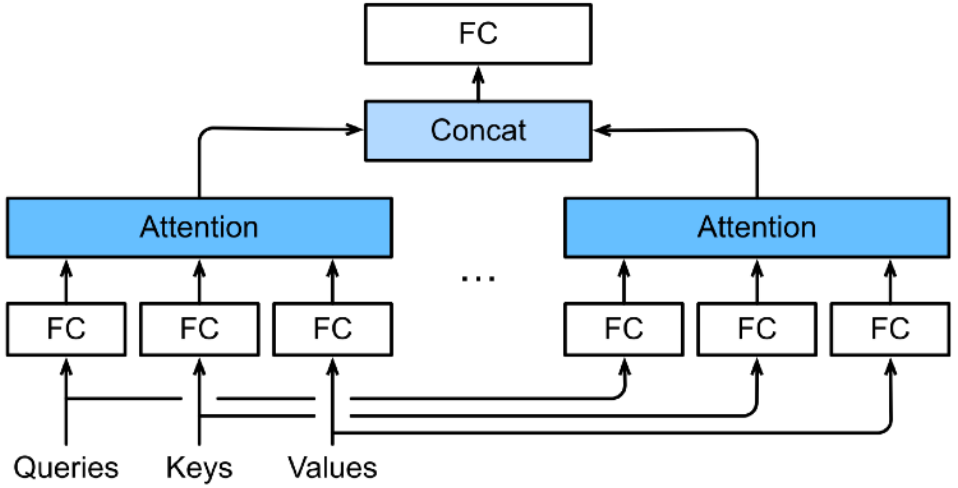
Scaled Dot-Product Attention



- In practice, we often think in minibatches for computational efficiency.
- Suppose we have n queries and m key-value pairs, where queries and keys are of length d and values are of length v.
- The scaled dot-product attention of queries $\mathbf{Q} \in \mathbb{R}^{n\times d}$, keys $\mathbf{K} \in \mathbb{R}^{m\times d}$, and values $\mathbf{V} \in \mathbb{R}^{m\times v}$ thus can be written as

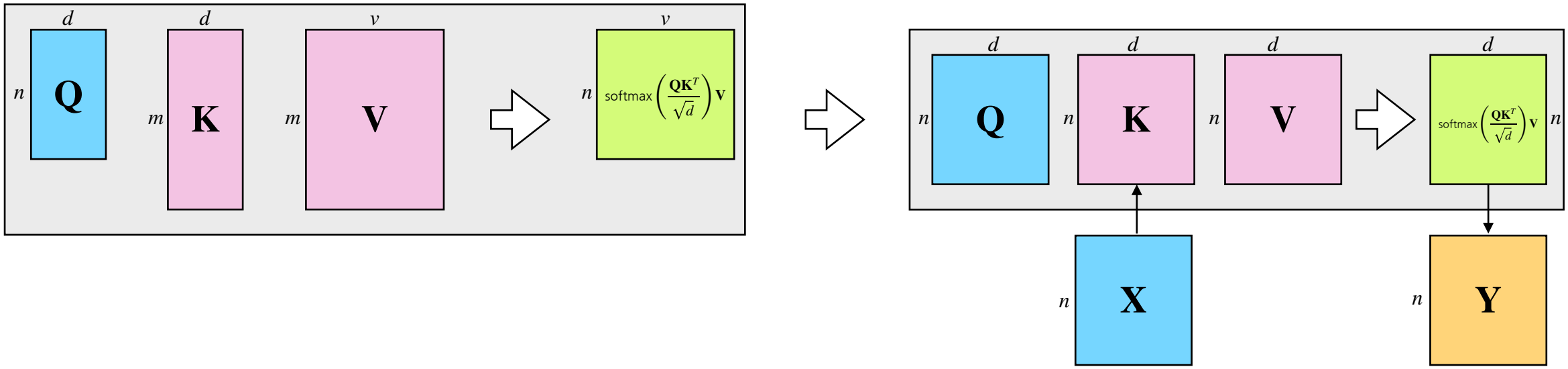
$$\text{softmax}\left(\frac{\mathbf{QK}^T}{\sqrt{d}}\right)\mathbf{V} \in \mathbb{R}^{n\times v}$$

Multi-Head Attention



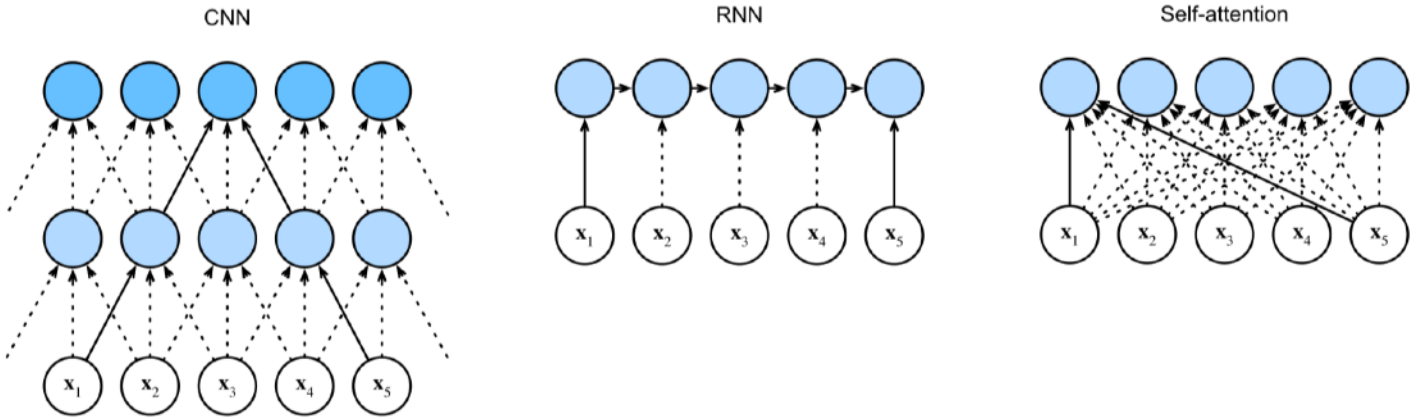
- In practice, given the same set of queries, keys, and values, we may want our model to combine knowledge from different behaviors of the same attention mechanism.
- Instead of performing a single attention pooling, queries, keys, and values can be transformed with h independently learned linear projections.
  - These h projected queries, keys, and values are fed into attention pooling in parallel.
  - In the end, h attention pooling outputs are concatenated and transformed with another learned linear projection to produce the final output.
- This design is called multi-head attention, where each of the h attention pooling outputs is a head.

Self-Attention



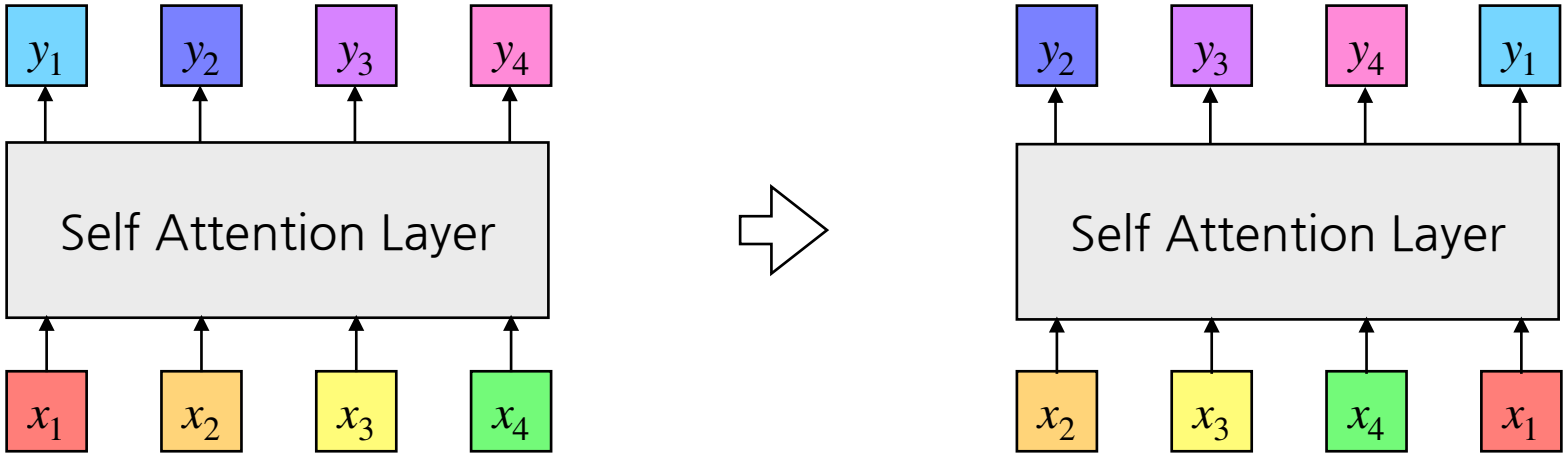
- Given a sequence of input tokens $x_1, \ldots, x_n$ where any $x_i \in \mathbb{R}^d$ $(1 \leq i \leq n)$, its self-attention outputs a sequence of the same length $y_1, \ldots, y_n$.
- While the original scaled dot product mechanism may have different lengths between queries and keys, they should be the same for the **self-attention mechanism**.

Comparisons



- Let's compare architectures for mapping a sequence of n tokens to another sequence of equal length, where each input or output token is represented by a d-dimensional vector.
- Consider a convolutional layer with kernel size k. The computational complexity is $O(knd^2)$ as the number of input and output channels are both d. There are $O(1)$ sequential operations and the maximum path length (dependencies, shorter the better) is $O(n/k)$.
- When updating the hidden state of RNNs, multiplication of the $d \times d$ weight matrix leads to the computational complexity of $O(d^2)$ and the sequence length is n, the computation complexity of RNNs is $O(nd^2)$. There are $O(n)$ sequential operations and the maximum path length is also $O(n)$.
- In self-attention, the scale dot-product attention has a $O(n^2d)$ computational complexity as it multiplies an $n \times d$ matrix with a $d \times n$ matrix. As all operations are done in parallel there are $O(1)$ sequential operations and the maximum path length is $O(1)$.

Permutation Invariance



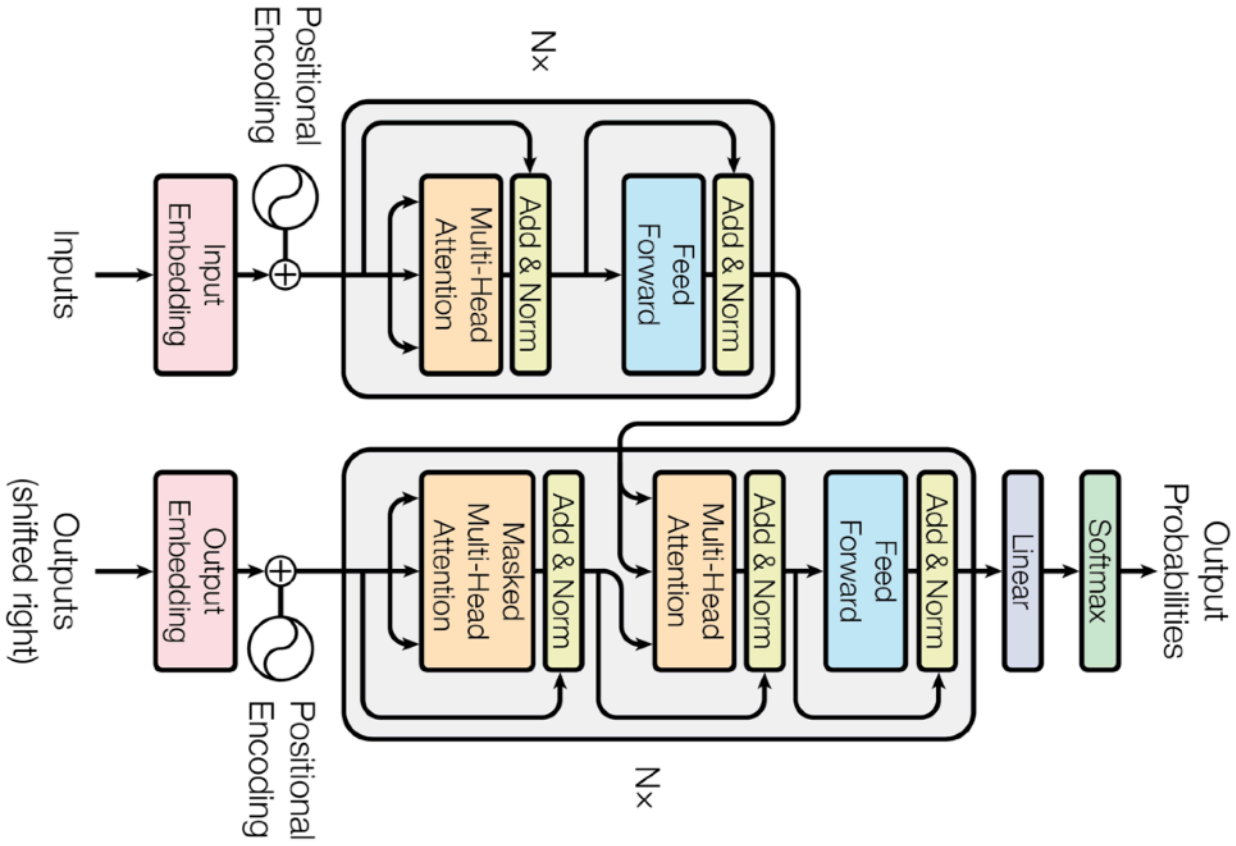
- Unlike RNNs, which recurrently process tokens of a sequence one by one, self-attention ditches sequential operations in favor of parallel computation.
- Note, however, that self-attention by itself does not preserve the order of the sequence (aka permutation invariance).
- The dominant approach for preserving information about the order of tokens is to represent this to the model as an additional input associated with each token.
    - These inputs are called **positional encodings**, and they can either be learned or fixed a priori.

ROBOT INTELLIGENCE LAB

Transformer

"Attention is All You Need," 2017

Transformer

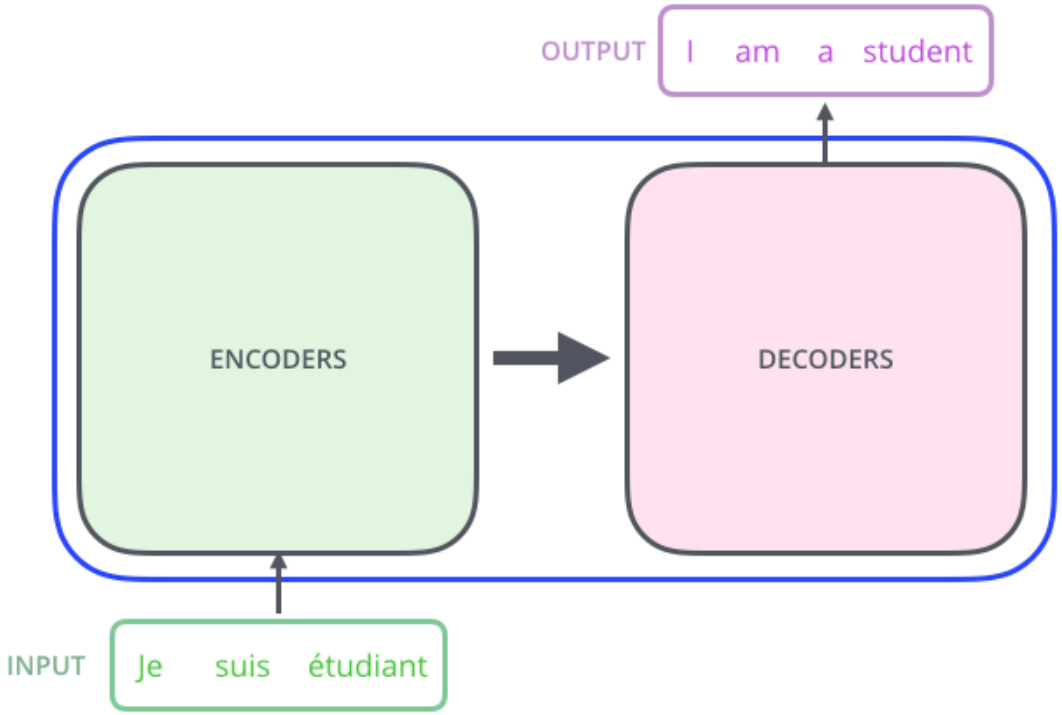


Transformer is the first sequence transduction model based entirely on attention.

Bird's eye view

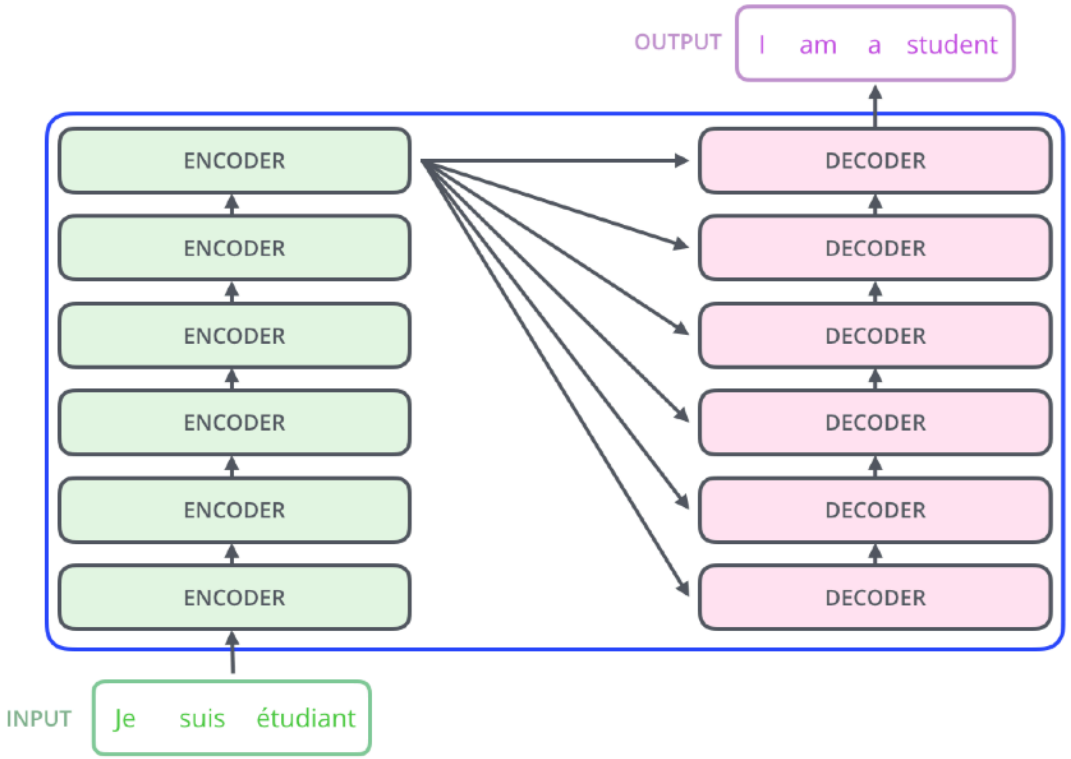
http://jalammar.github.io/illustrated-transformer/

Glide down a little



It consists of an **encoder** and a **decoder**

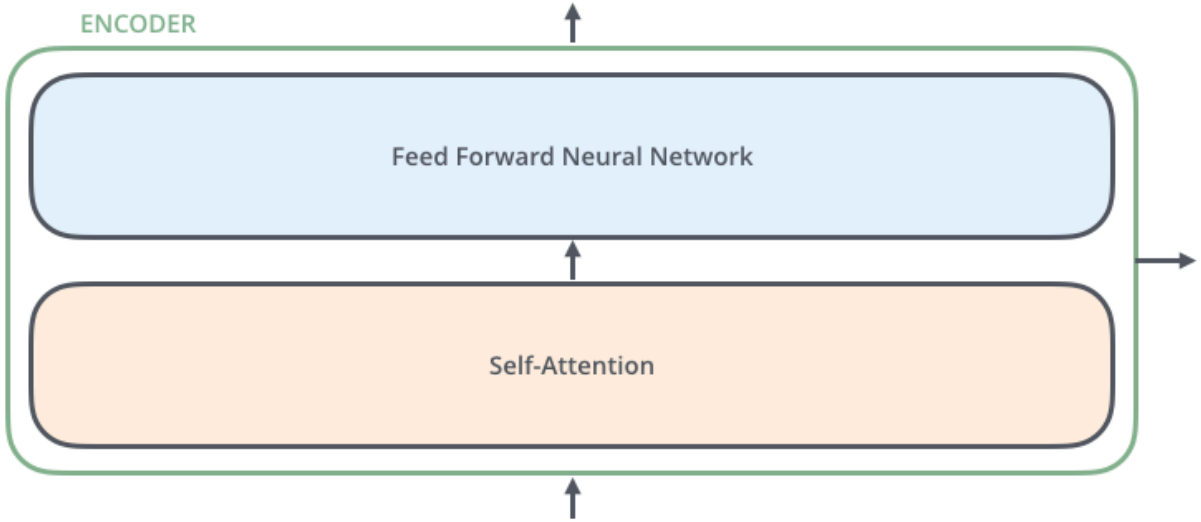
Glide down a little more



What information is being set from an **encoder** to a **decoder**?

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

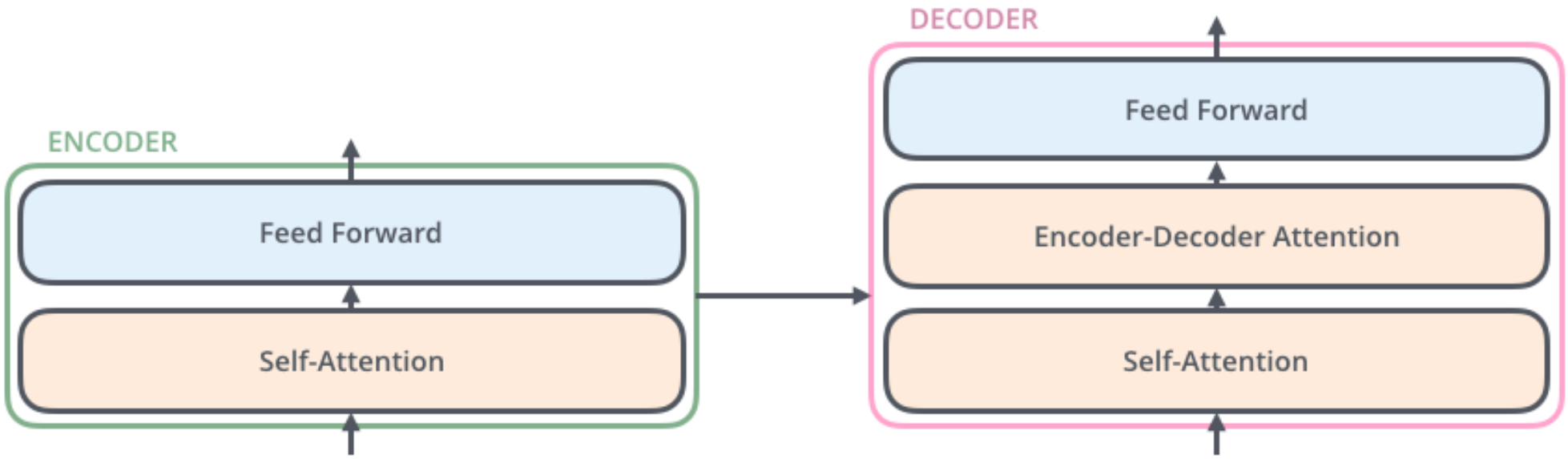
Encoder



Each encoder has an identical structure where the **Self-Attention** layer is the cornerstone of Transformer.

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

Encoder-Decoder



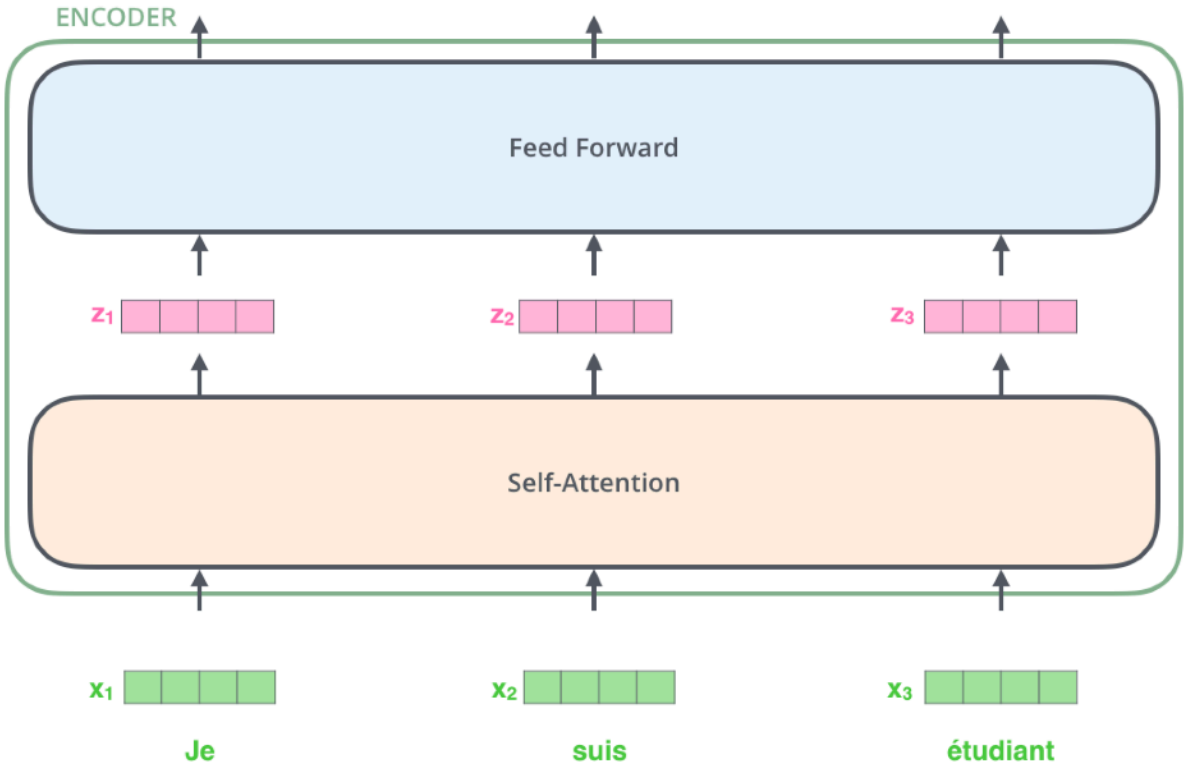
Decoder has an additional structure, **Encoder-Decoder Attention**, along with self-attention block

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

Input

Each input (e.g., word) is represented with embedding vectors.

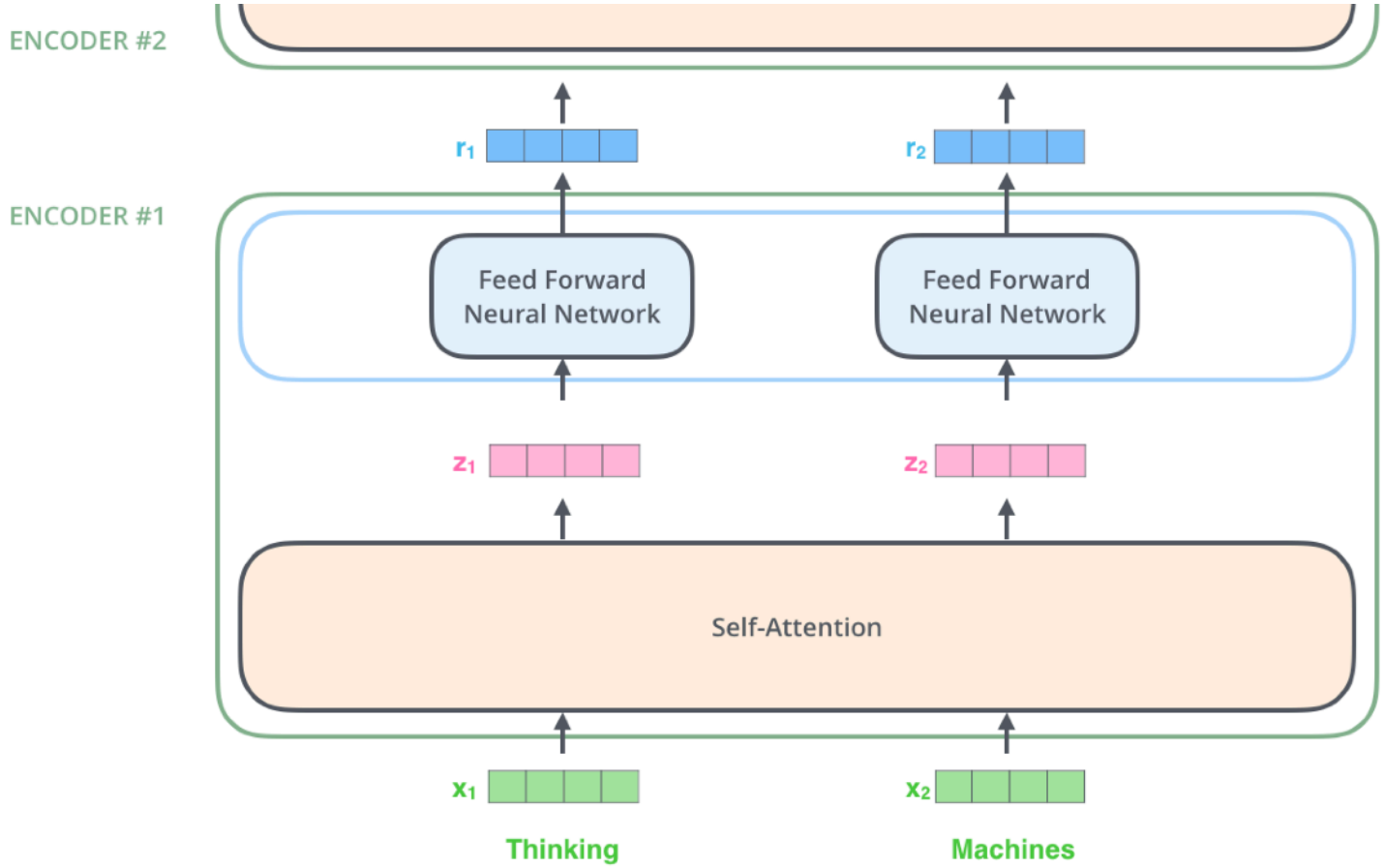
Encoder



Each input is encoded by first passing through a **Self-Attention** layer followed by a **Feed-Forward** layer

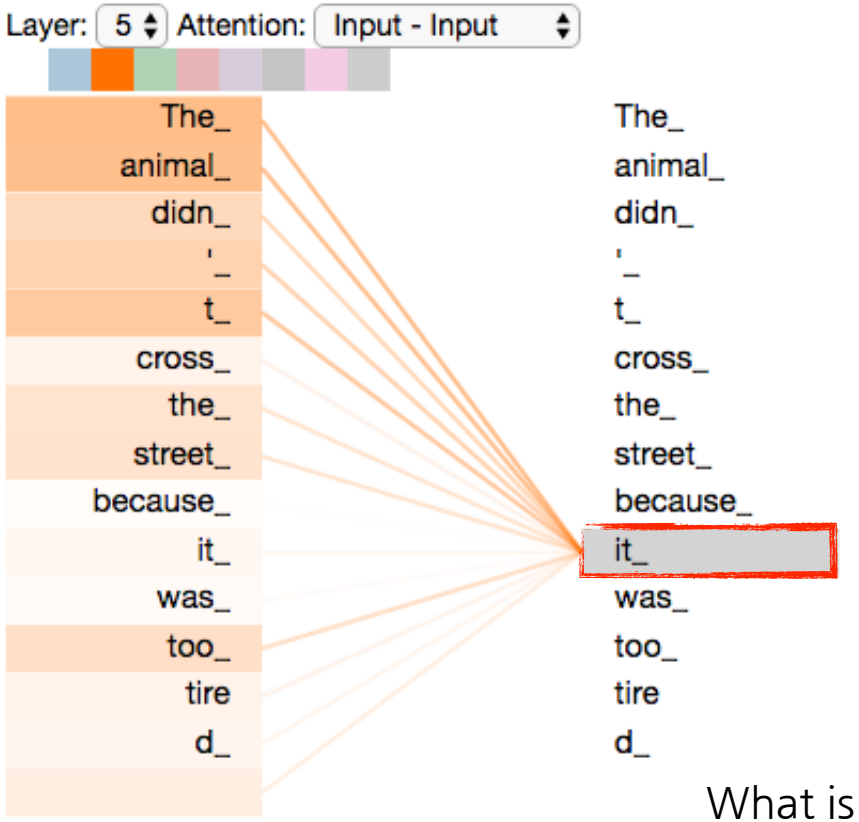
http://jalammar.github.io/illustrated-transformer/

Encoder



Suppose we have two words, **Thinking** and **Machines**, as an input sequence.

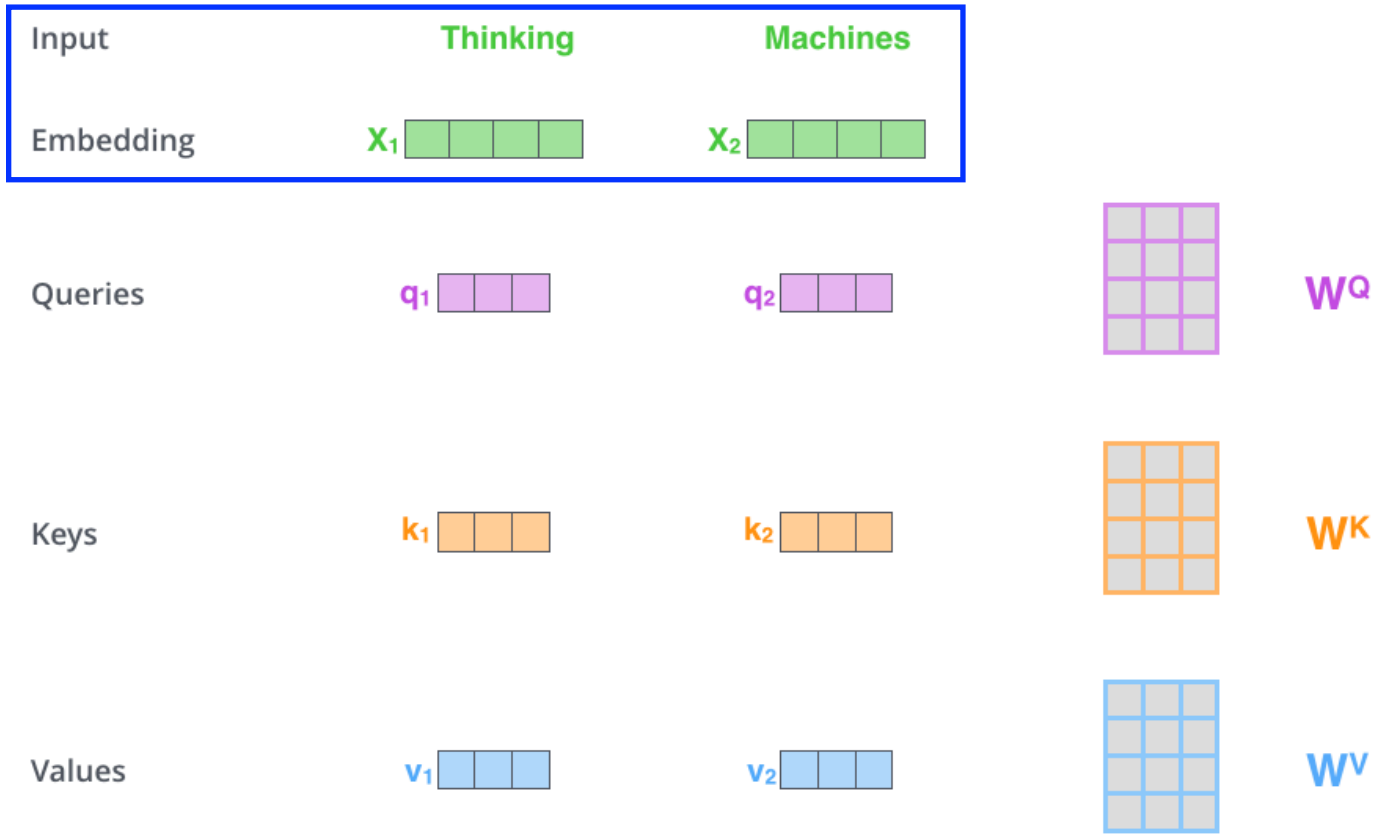
Self Attention



What is it referring to?

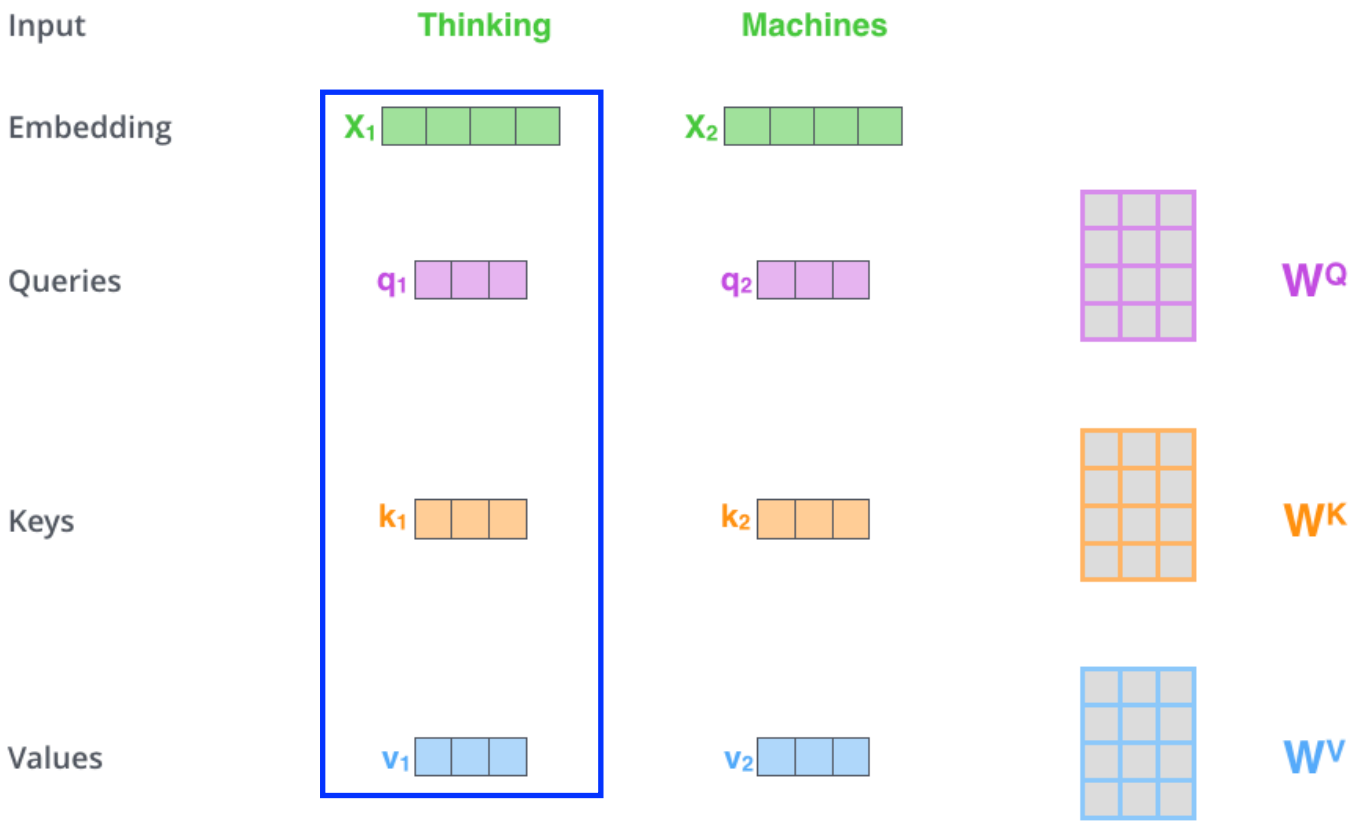
"The animal didn't cross the street because it was too tired"

Encoder



Each input is represented with an **embedding vector** (e.g., Word2Vec)

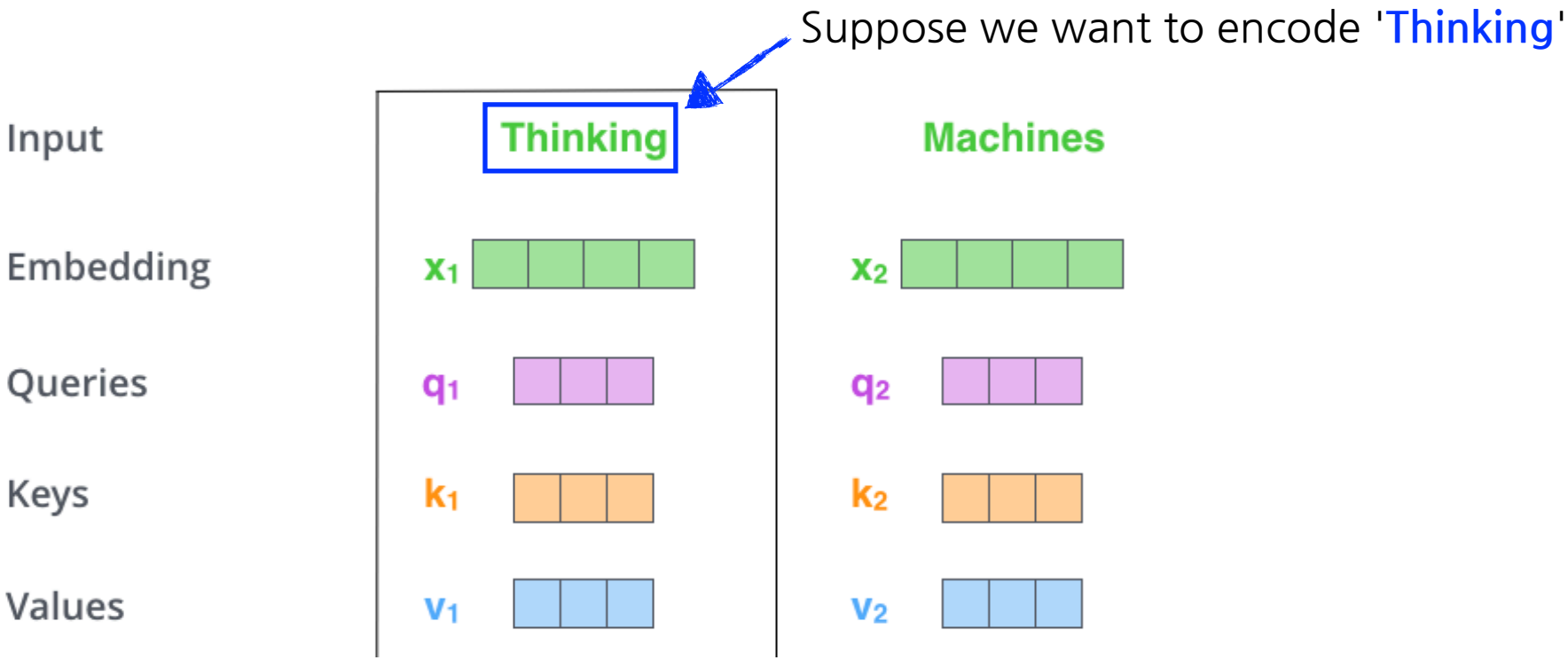
Encoder



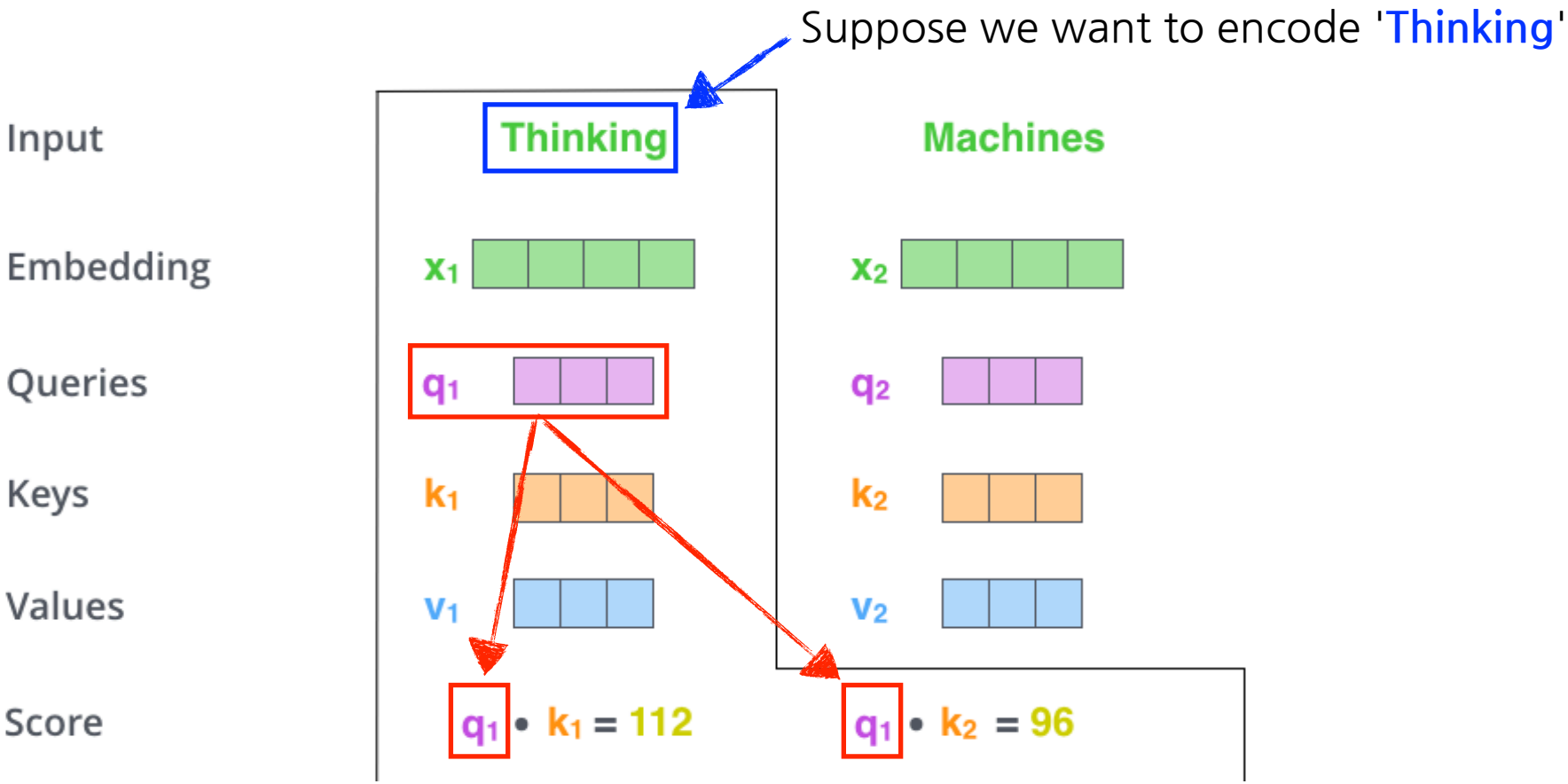
Query, **Key**, and **Value** vectors are computed from the **embedding** vector using three different networks.

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

Encoding 'Thinking'

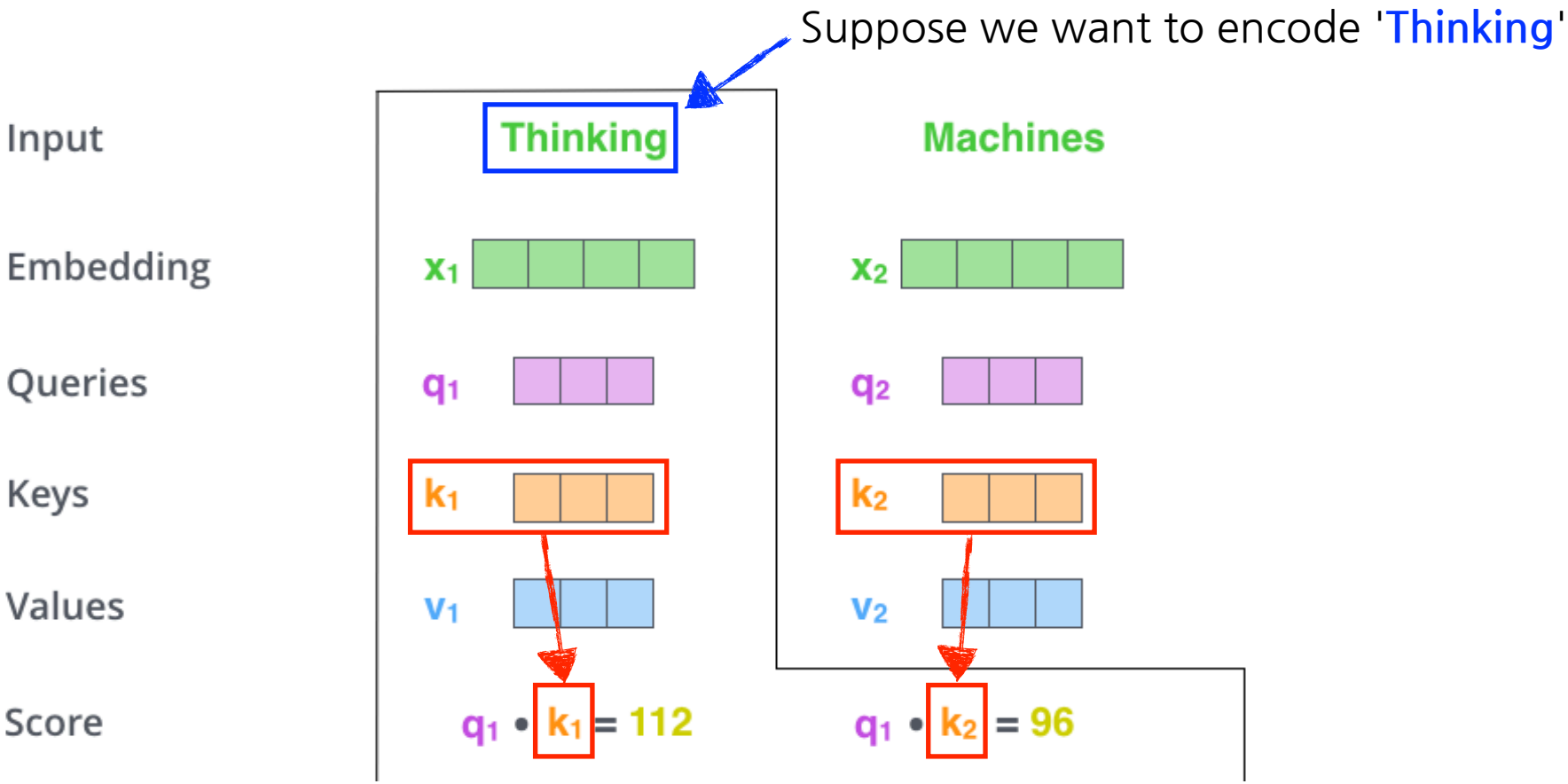


Encoding 'Thinking'



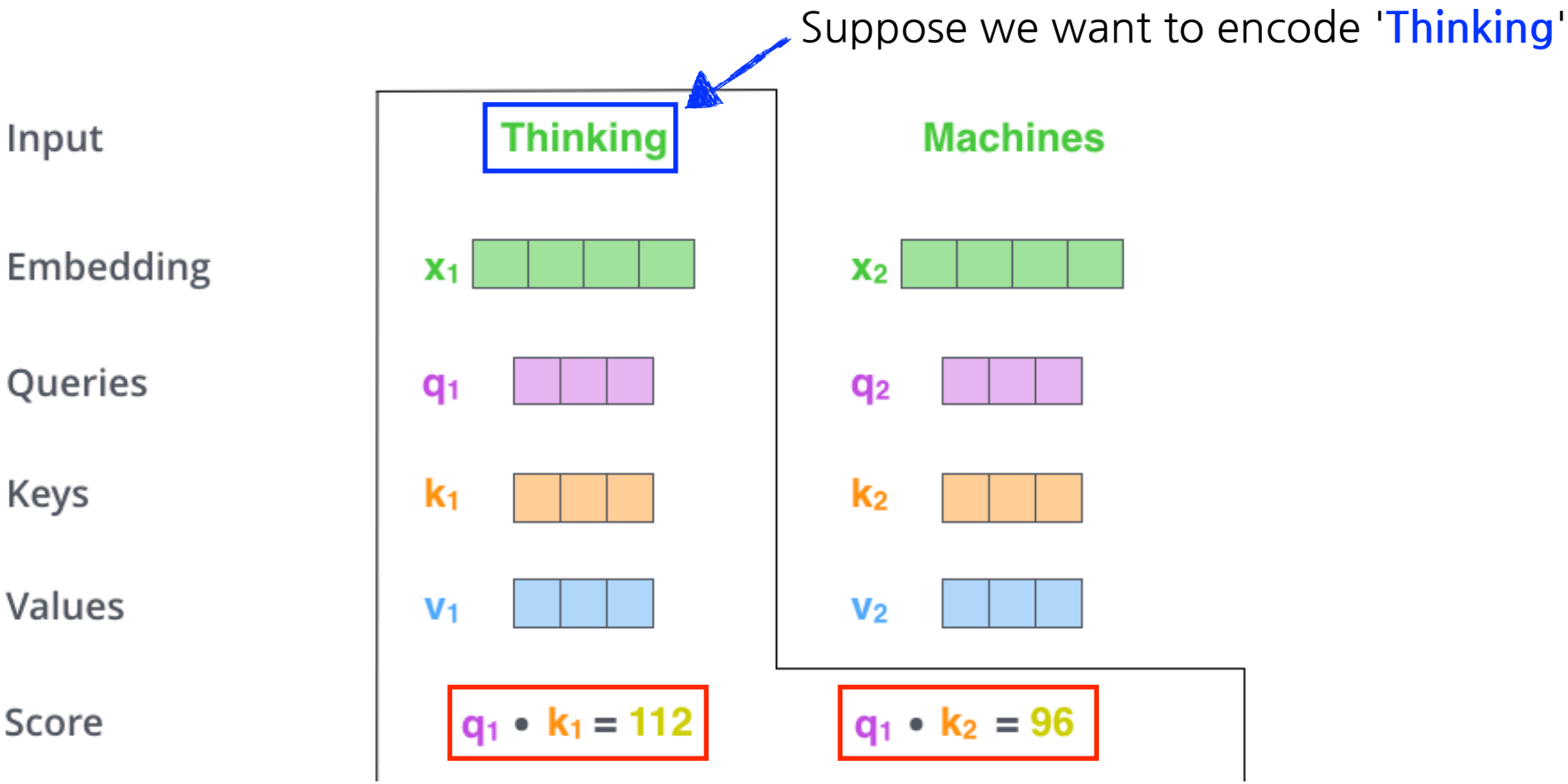
Query vector comes from the encoding word (i.e., Thinking)

Encoding 'Thinking'



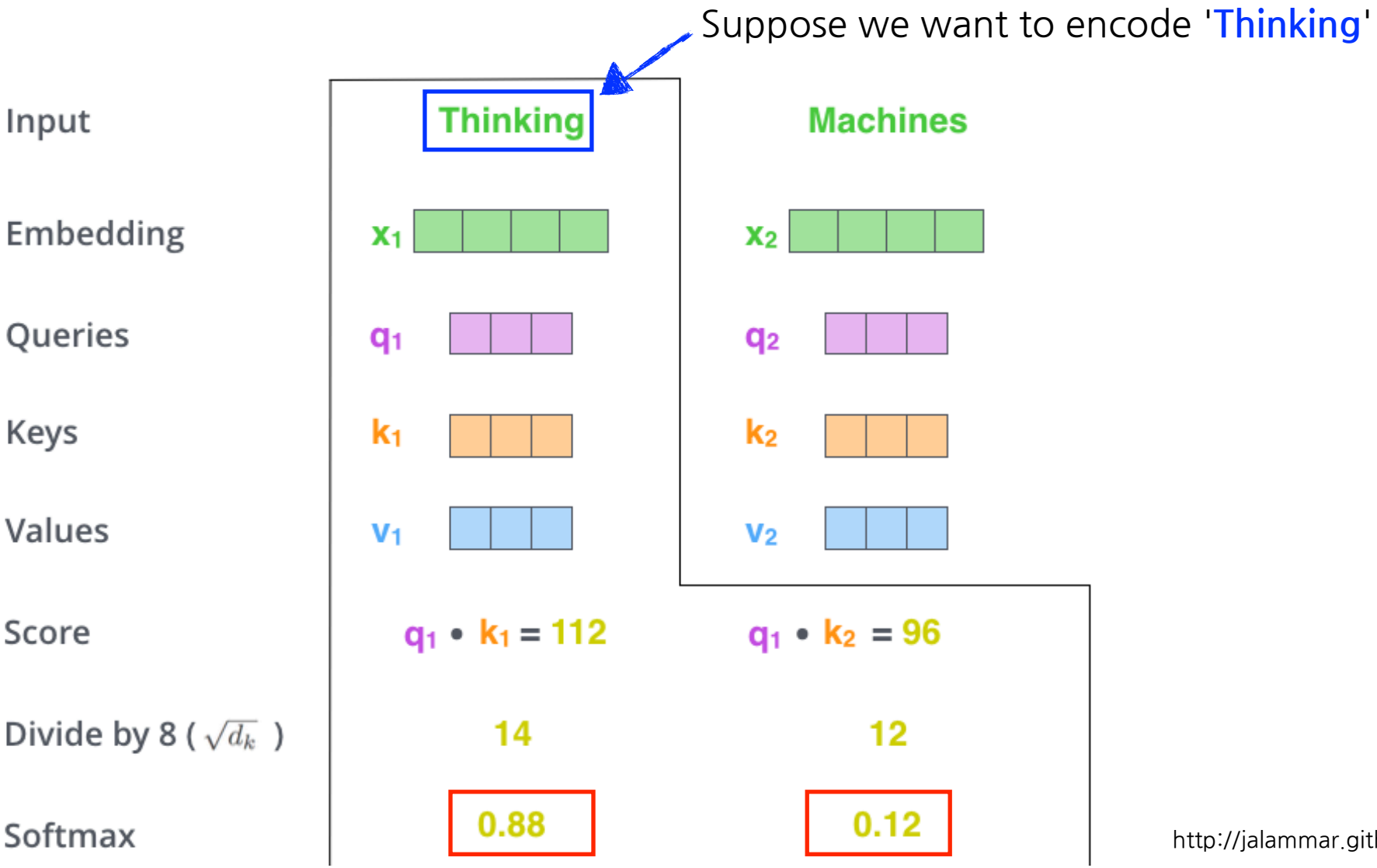
Key vector comes from the corresponding target words (i.e., **Thinking** and **Machines**)

Encoding 'Thinking'

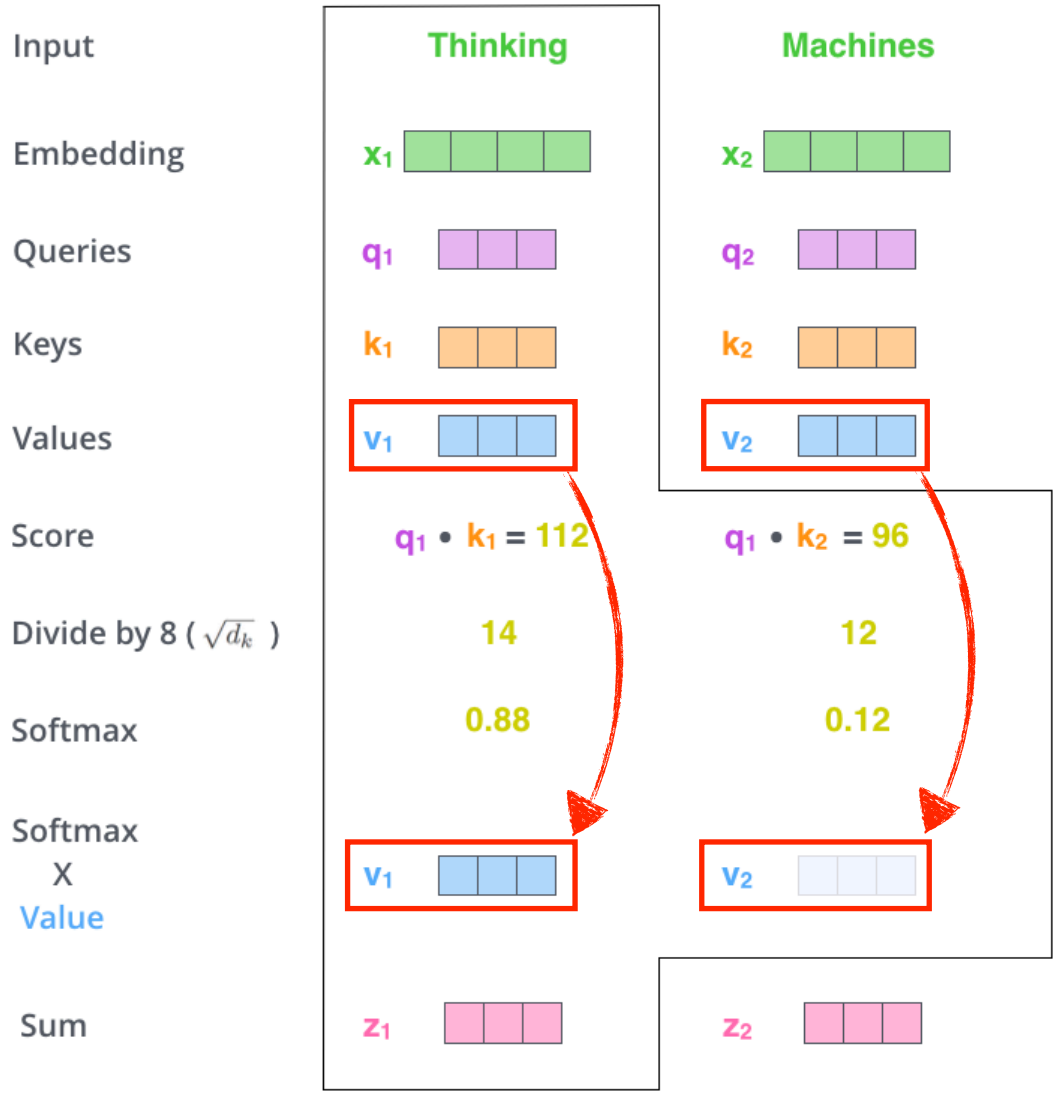


The score of each input is computed from the inner products between **query** and **key** vectors

Encoding 'Thinking'

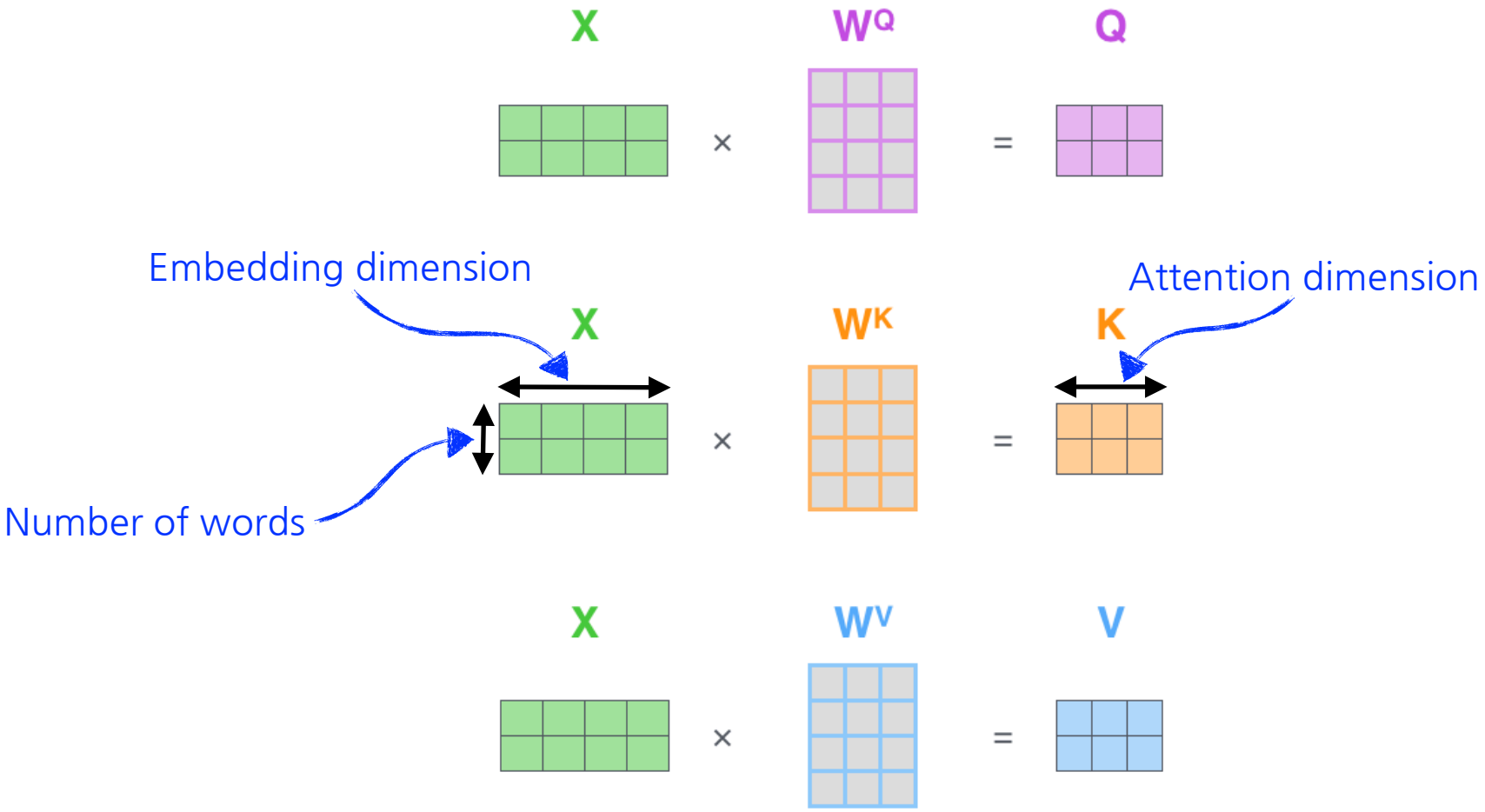


Encoding 'Thinking'

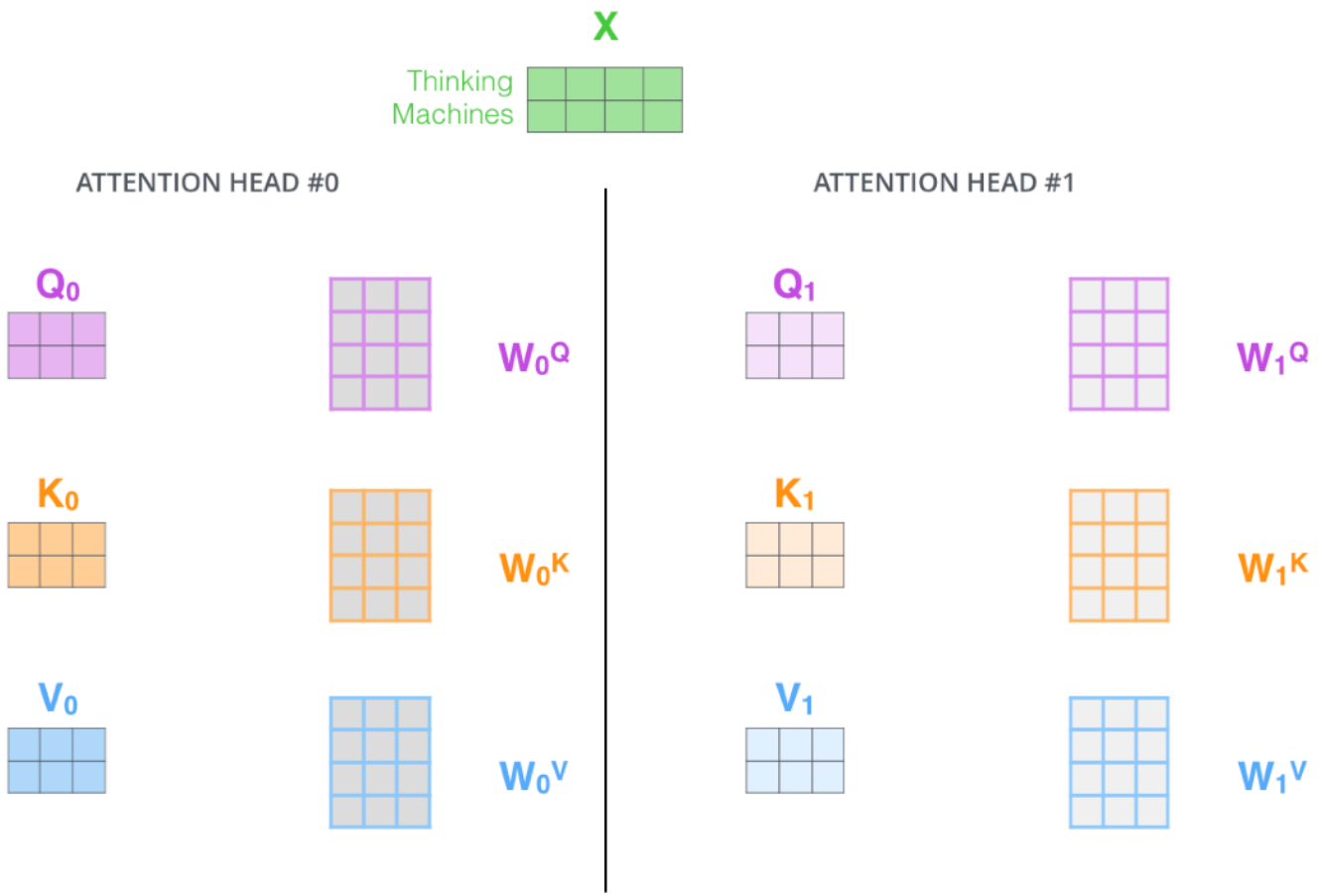


http://jalammar.github.io/illustrated-transformer/

Computing Q, K, and V



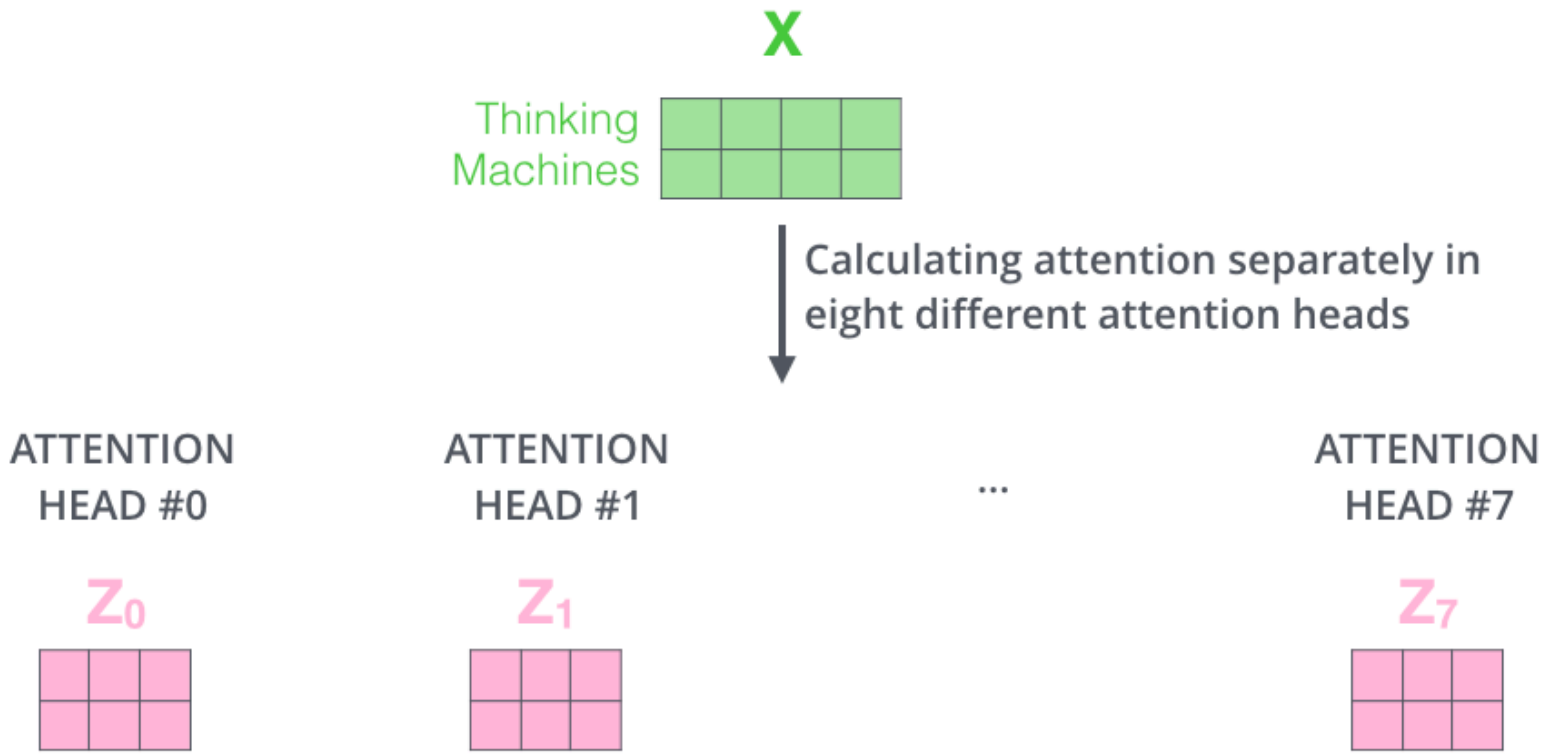
Multi-Head Attention (MHA)



Multi-Head Attention (MHA) allows to focus on different positions (attentions)

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

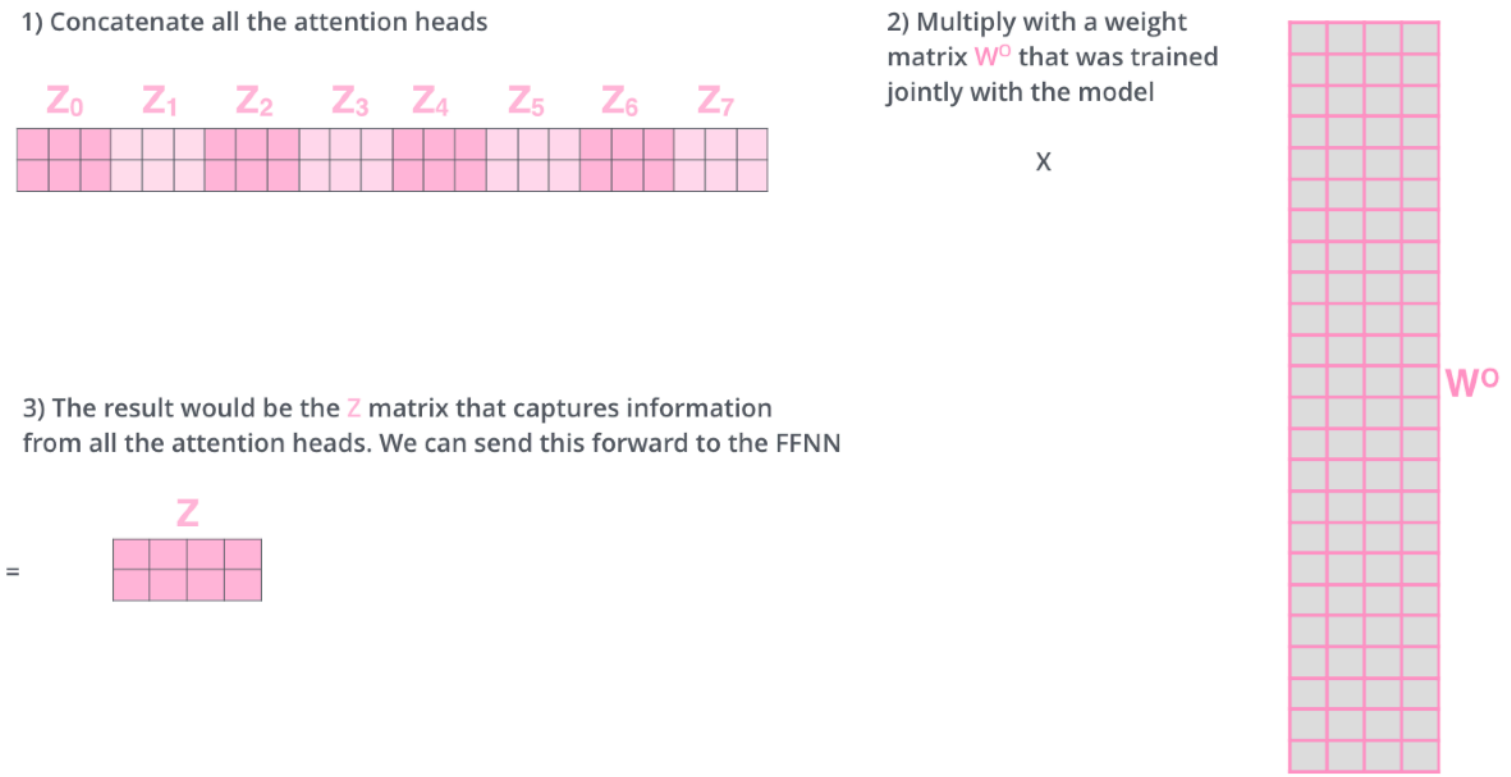
Multi-Head Attention (MHA)



How do we accumulate multiple attention results from MHA?

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

Simple Linear Map



First **concatenate** and then pass it through a (learnable) linear map to reduce the dimension

Summary

1) This is our input sentence*

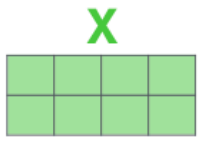
2) We embed each word*

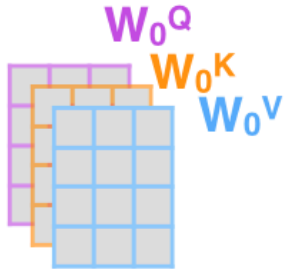
3) Split into 8 heads. We multiply X or R with weight matrices

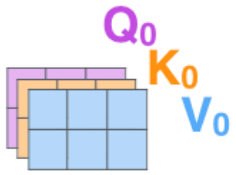
4) Calculate attention using the resulting Q/K/V matrices

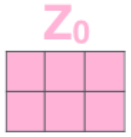
5) Concatenate the resulting Z matrices, then multiply with weight matrix W^O to produce the output of the layer

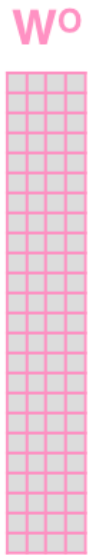
Thinking Machines

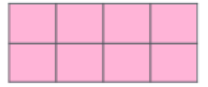




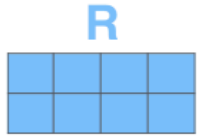


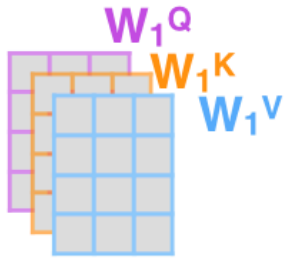


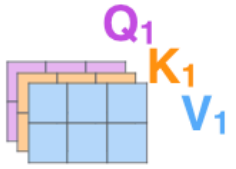


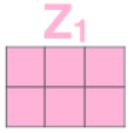


* In all encoders other than #0, we don't need embedding. We start directly with the output of the encoder right below this one





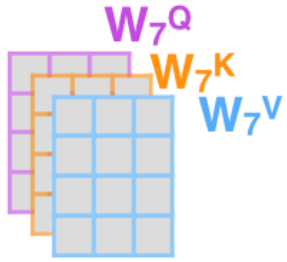


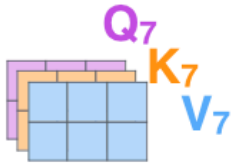


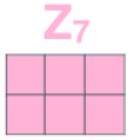
...

...

...





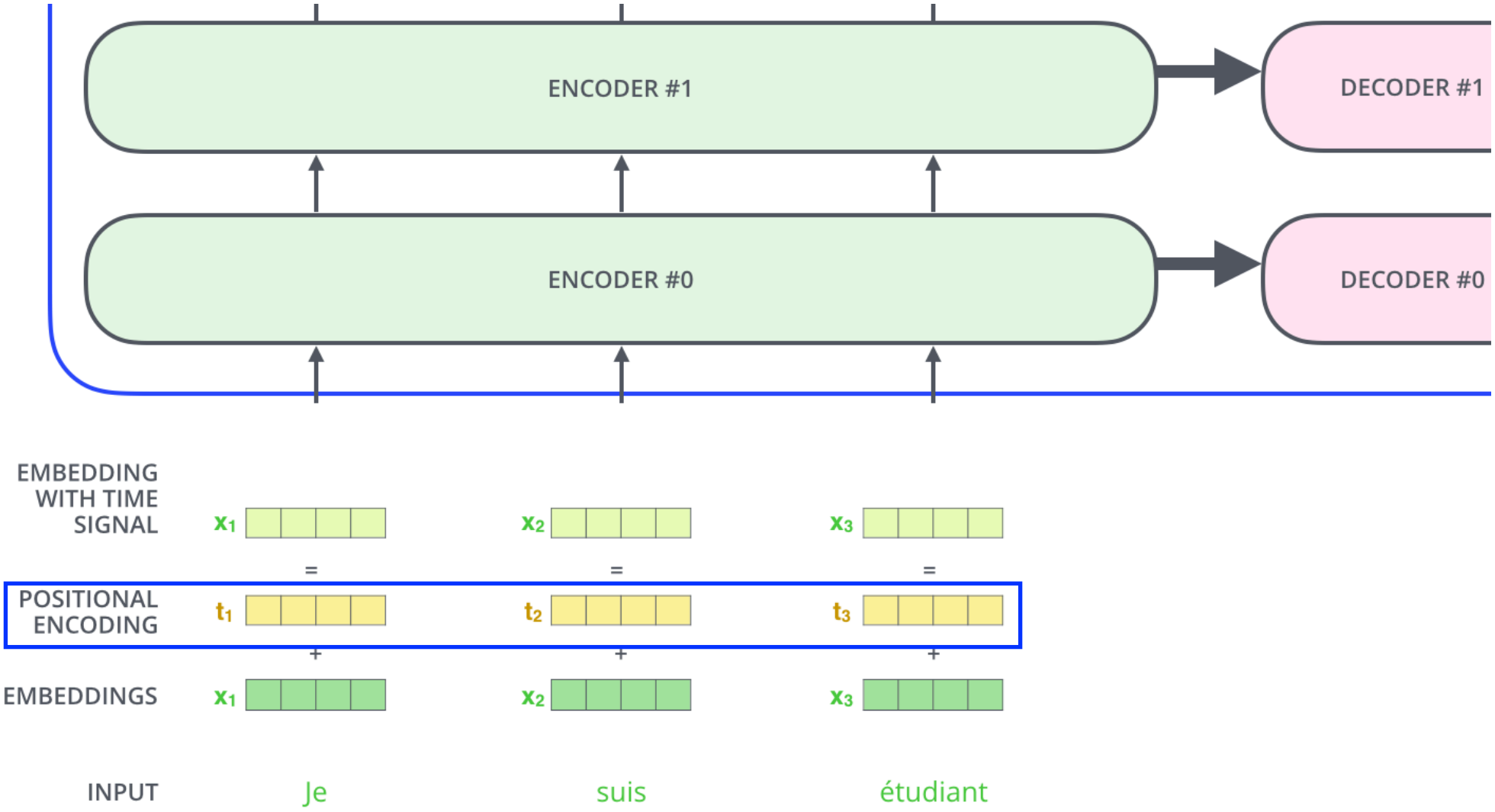


Ordering?

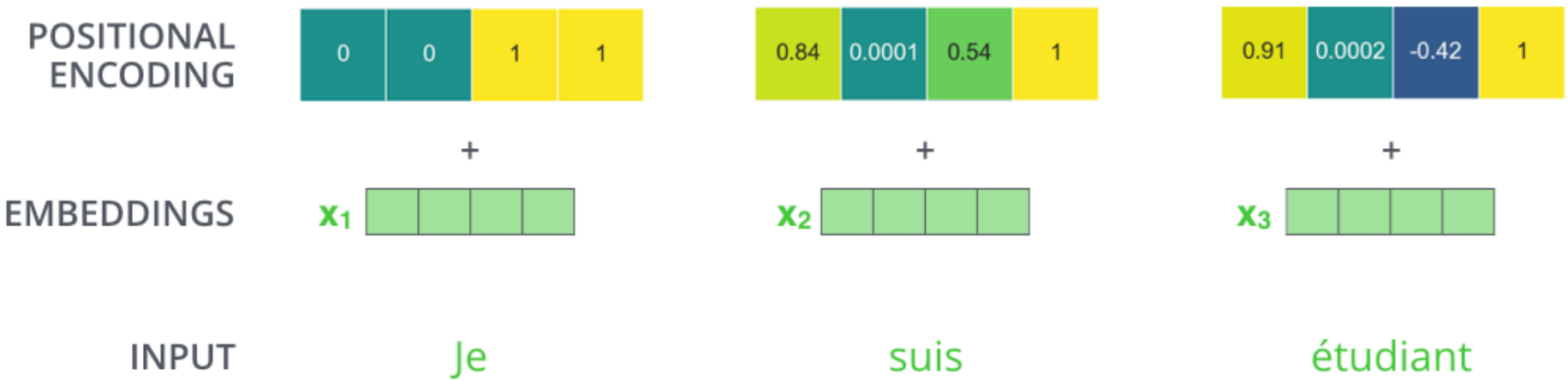
ROBOT INTELLIGENCE LAB

Is Transformer **input-order dependent**?

Positional Encoding

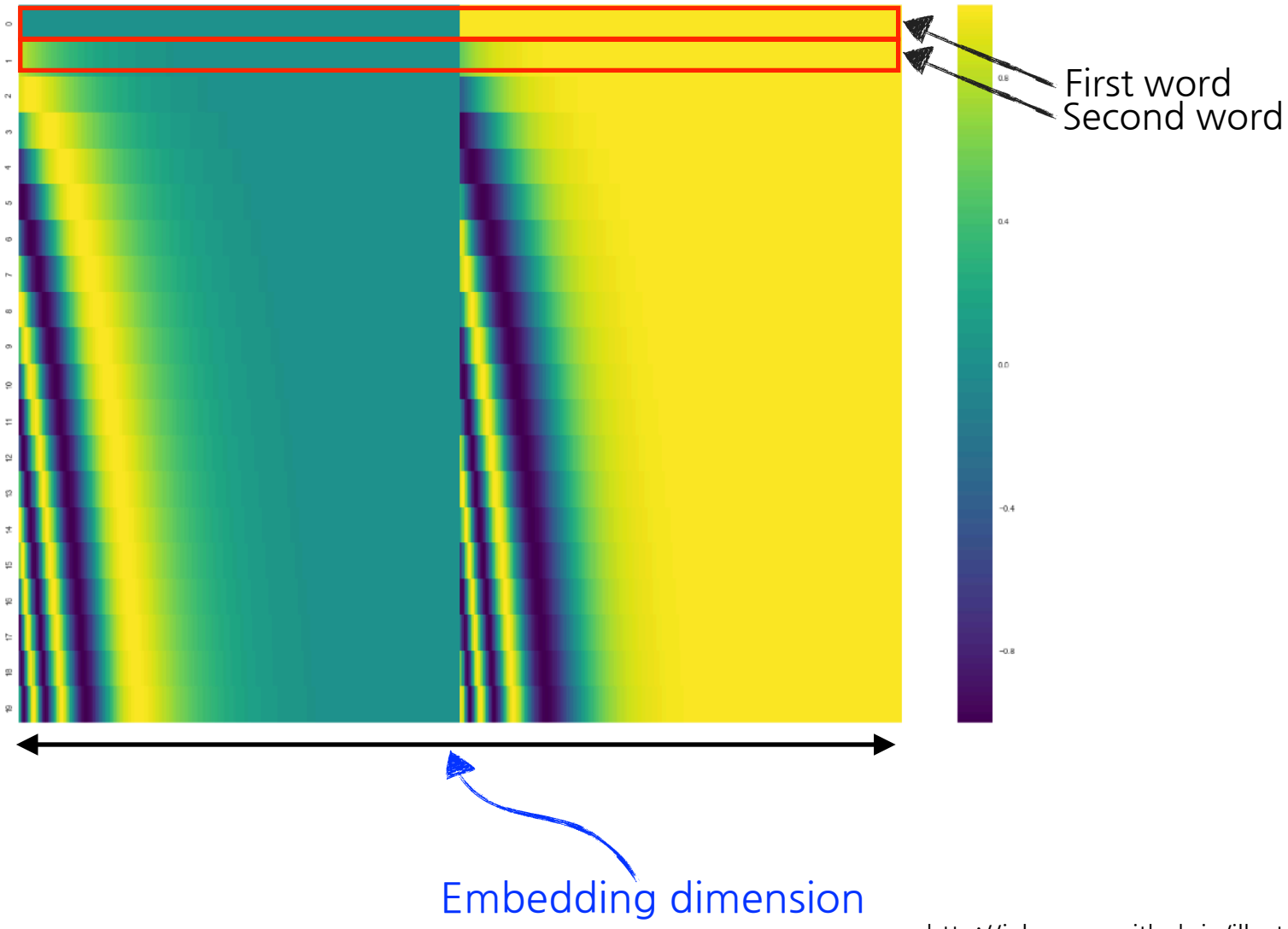


Positional Encoding



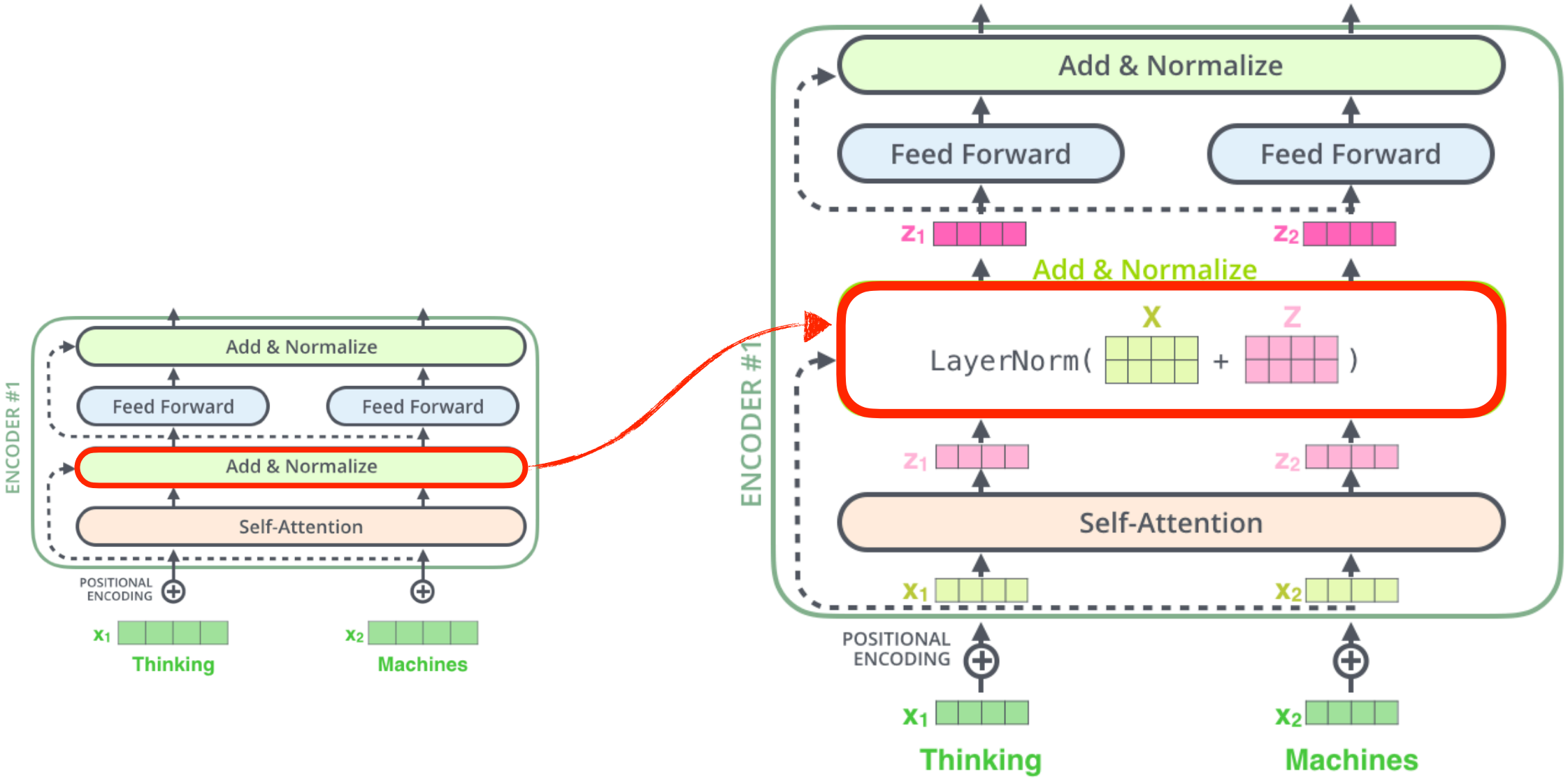
http://jalammar.github.io/illustrated-transformer/

Positional Encoding



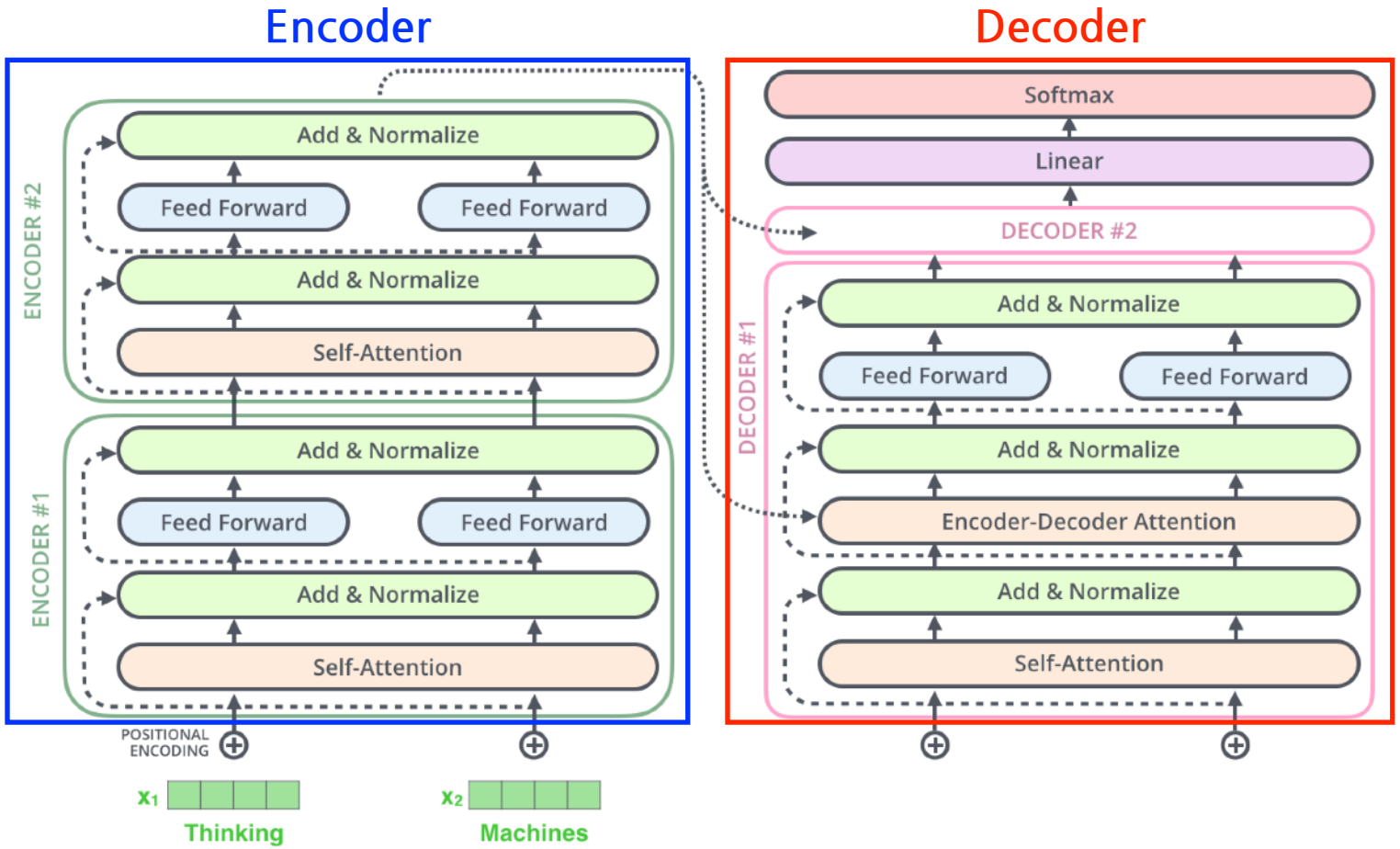
http://jalammar.github.io/illustrated-transformer/

Encoder



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

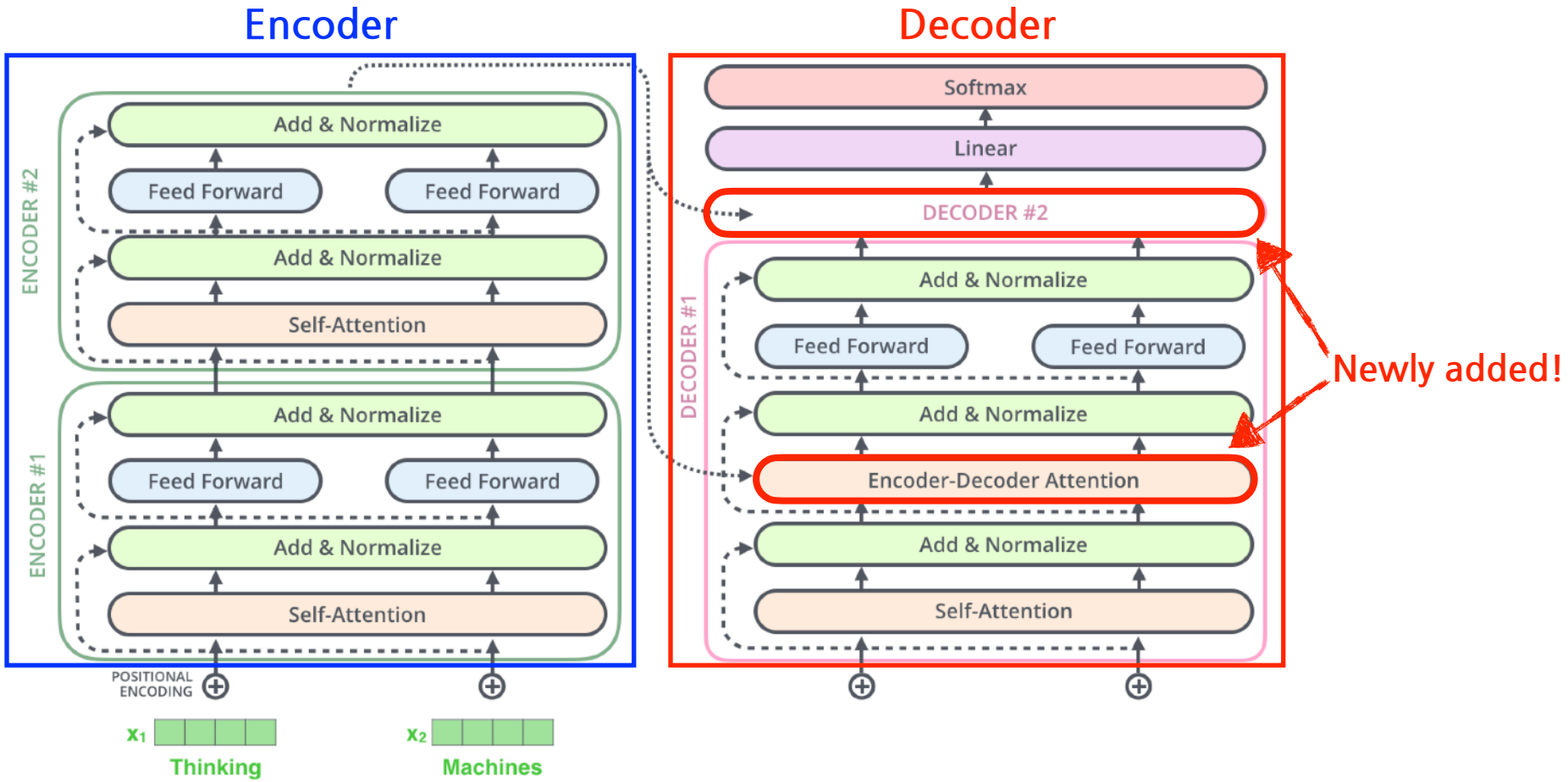
Encoder-Decoder



Both Encoder and Decoder have similar architectures.

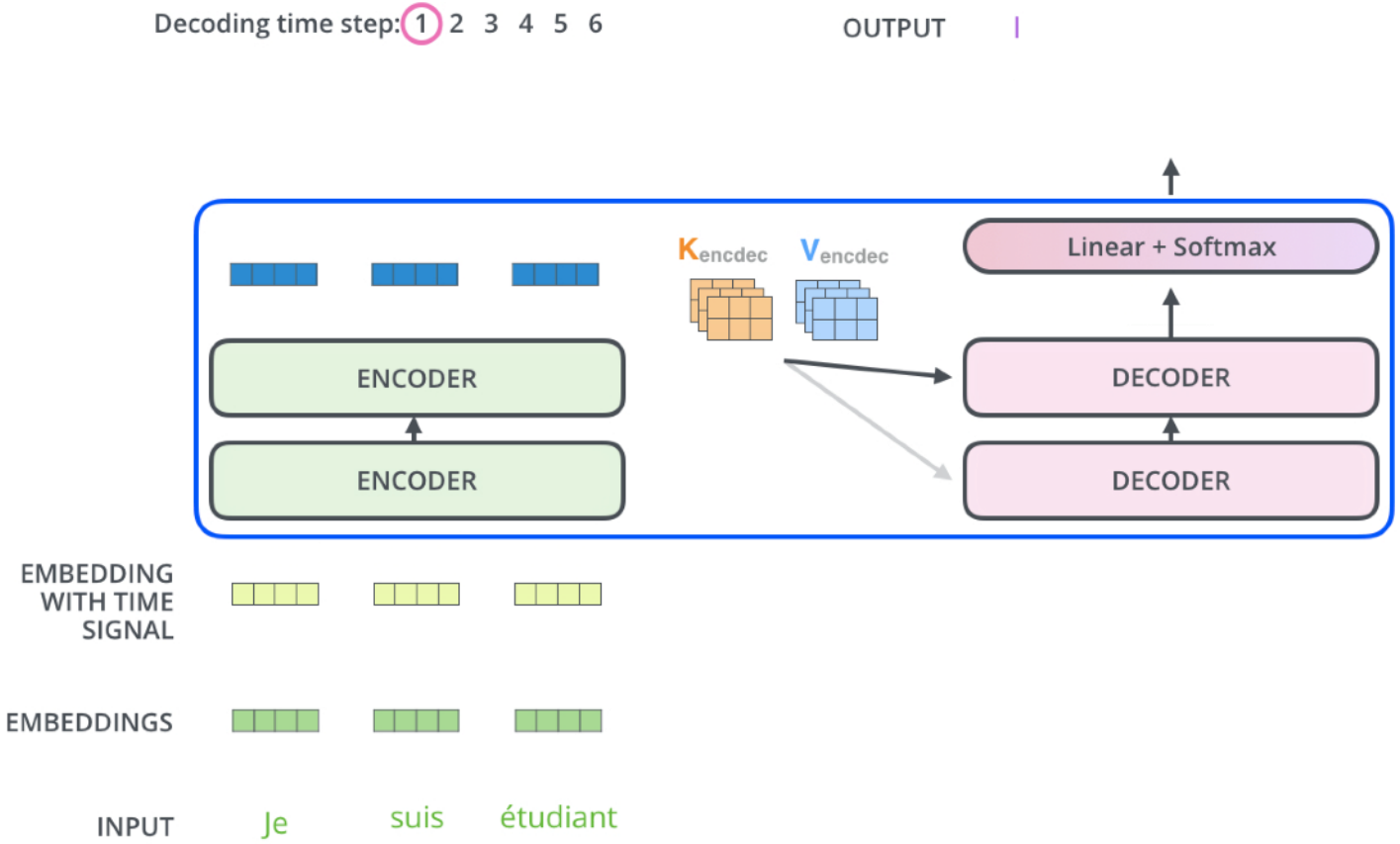
http://jalammar.github.io/illustrated-transformer/

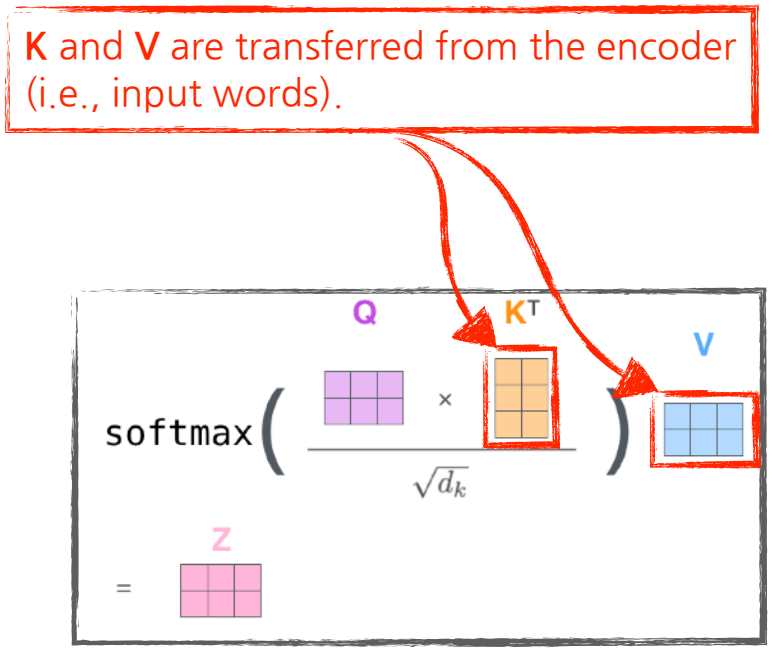
Encoder-Decoder



What is the information being sent from Encoder to Decoder?

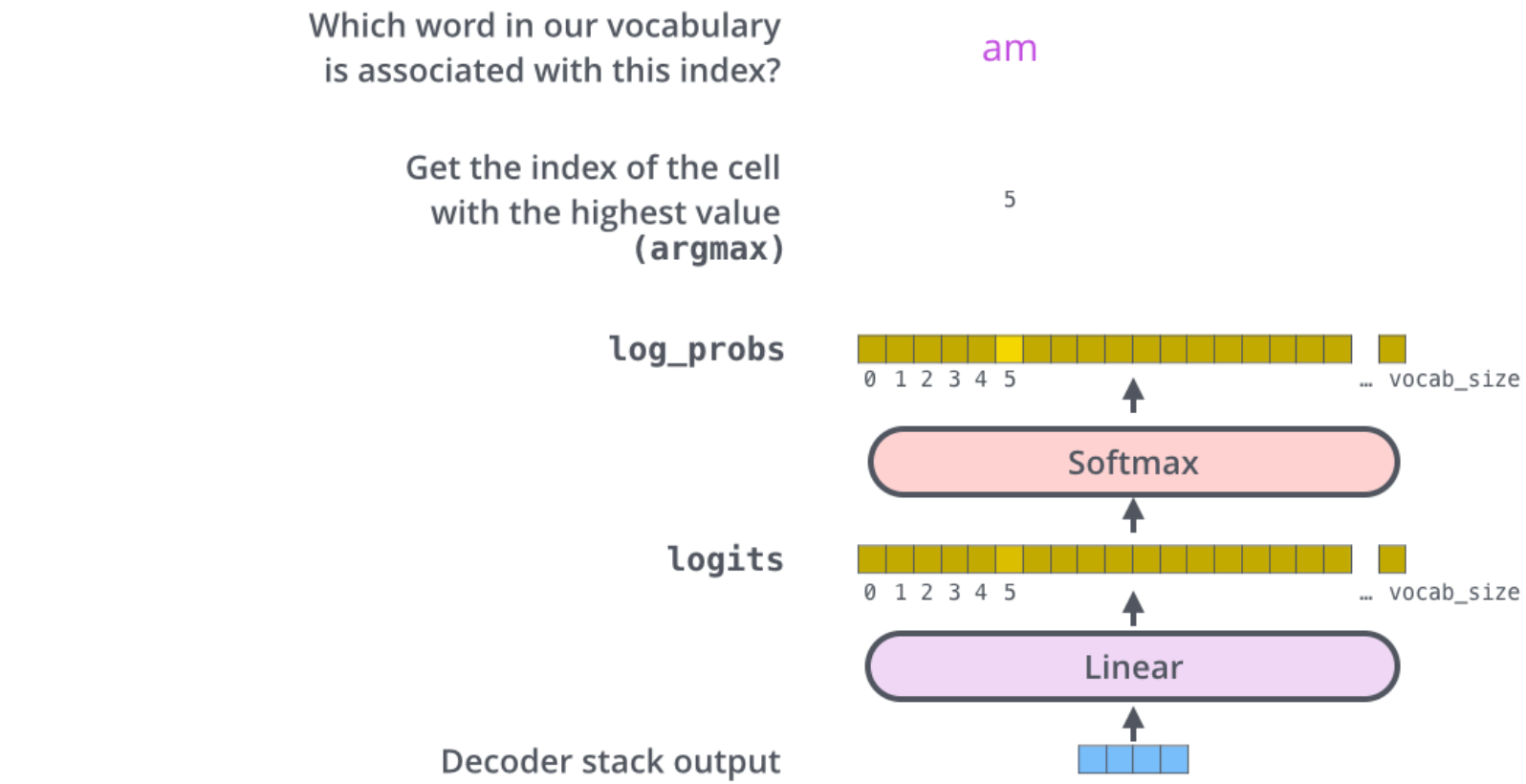
Encoder-Decoder





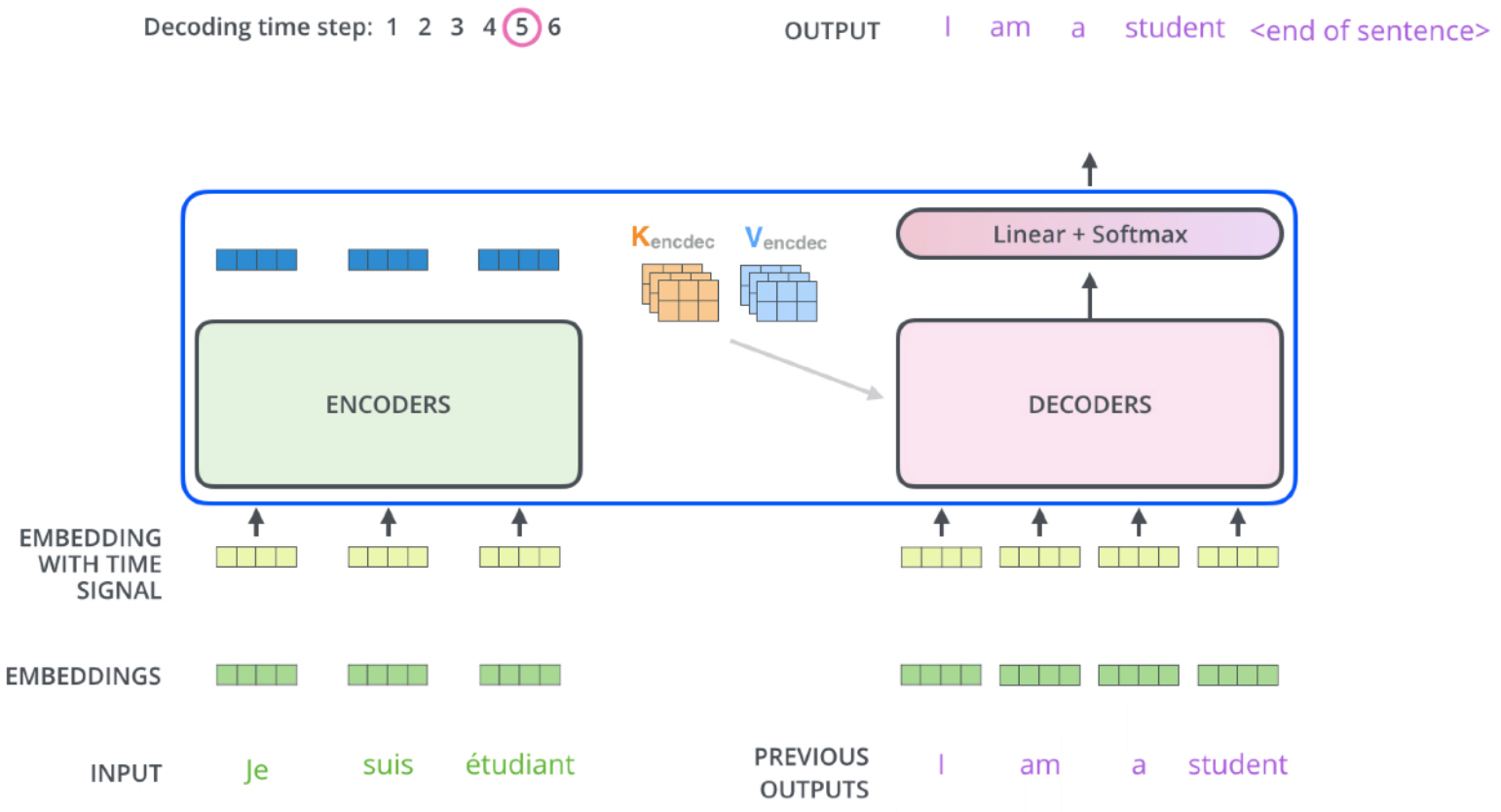
Key (K) and **Value (V)** vectors are transferred from Encoder to Decoder

Decoder



Final layer converts the stack of decoder outputs to the distribution over outputs (e.g., words)

Decoder



Output sequence is generated in an autoregressive manner

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

ROBOT INTELLIGENCE LAB

ViT

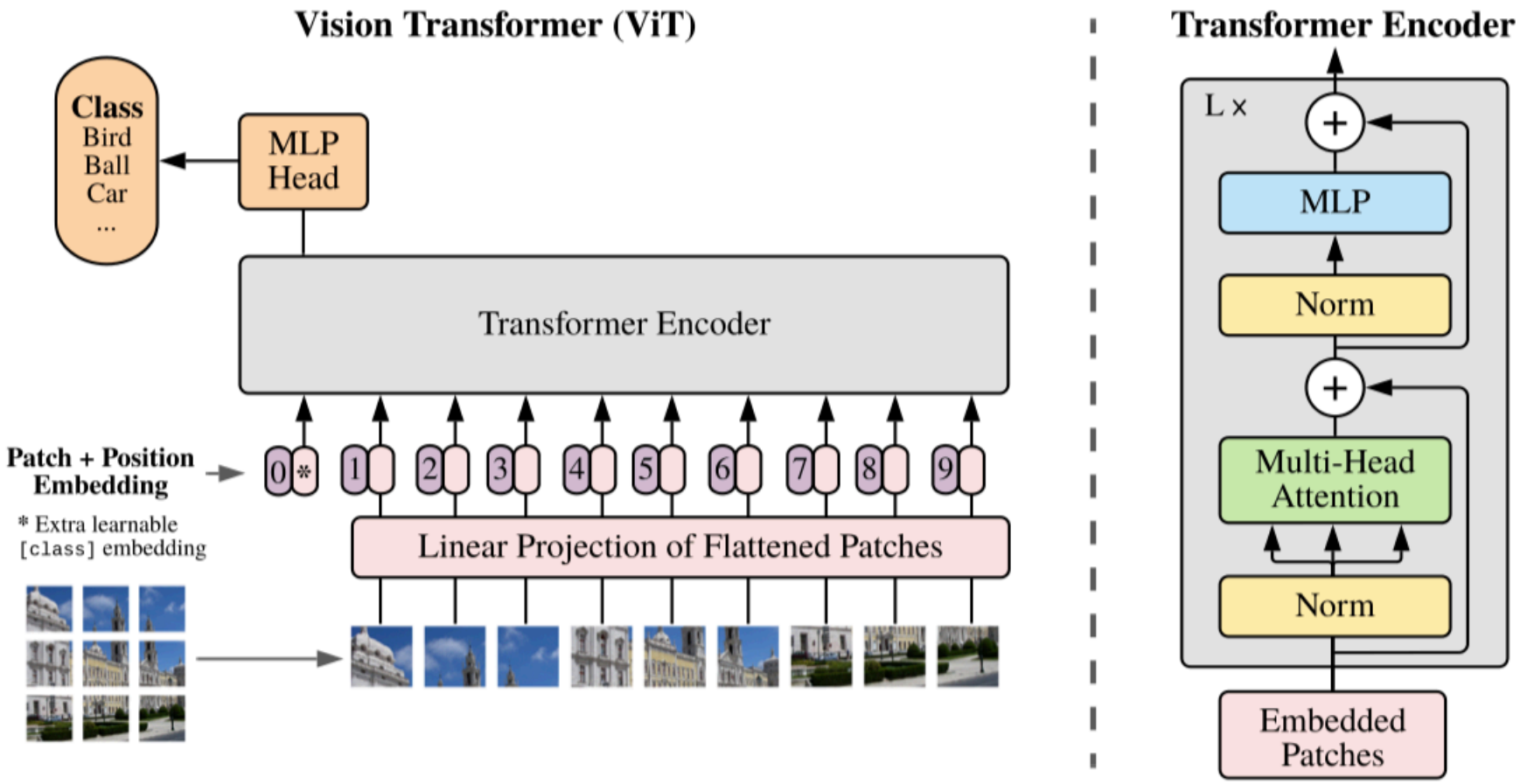
"AN IMAGE IS WORTH 16 X 16 WORDS :TRANSFORMERS FOR IMAGE RECOGNITION AT SCALE," 2021

Vision Transformer (ViT)

ROBOT INTELLIGENCE LAB

Can we use **Transformer** for image classification tasks?

ViT


Vision Transformer (ViT)
Class
Bird
Ball
Car
...
MLP Head
Transformer Encoder
Patch + Position Embedding
* Extra learnable [class] embedding
0 * 1 2 3 4 5 6 7 8 9
Linear Projection of Flattened Patches
Transformer Encoder
L x
MLP
Norm
Multi-Head Attention
Norm
Embedded Patches

ViT Details

ROBOT INTELLIGENCE LAB

- To handle 2D images, the image $\mathbf{x} \in \mathbb{R}^{H \times W \times C}$ into sequence of flattened 2D patches $\mathbf{x}_p \in \mathbb{R}^{N \times (P^2 C)}$ where (H, W) is the resolution of the original image, $N = HW/P^2$ is the number of patches, C is the number of channels, (P, P) is the resolution of each image patch.
  - A simple linear projection is used to map a patch to a token (latent vector), $\mathbf{z}_i = \mathbf{x}_i \mathbf{E}$ where $\mathbf{z}_i \in \mathbb{R}^D$, $\mathbf{x}_i \in \mathbb{R}^{P^2 C}$, and $\mathbf{E} \in \mathbb{R}^{(P^2 C) \times D}$.
- A learnable embedding $\mathbf{z}_0^0$ is prepended to the sequence of embedded patches, whose state at the output of the Transformer encoder, $\mathbf{z}_L^0$, serves as the image representation $\mathbf{y}$.
- Positional embeddings are added to the patch embeddings to retain positional information.
  - Standard **learnable** 1D position embeddings are used.
  - More advanced 2D-aware position embeddings did not show significant performance gains.

Results

ROBOT INTELLIGENCE LAB

	Ours-JFT (ViT-H/14)	Ours-JFT (ViT-L/16)	Ours-I21k (ViT-L/16)	BiT-L (ResNet152x4)	Noisy Student (EfficientNet-L2)
ImageNet	**88.55** ± 0.04	87.76 ± 0.03	85.30 ± 0.02	87.54 ± 0.02	88.4/88.5*
ImageNet ReaL	**90.72** ± 0.05	90.54 ± 0.03	88.62 ± 0.05	90.54	90.55
CIFAR-10	**99.50** ± 0.06	99.42 ± 0.03	99.15 ± 0.03	99.37 ± 0.06	–
CIFAR-100	**94.55** ± 0.04	93.90 ± 0.05	93.25 ± 0.05	93.51 ± 0.08	–
Oxford-IIIT Pets	**97.56** ± 0.03	97.32 ± 0.11	94.67 ± 0.15	96.62 ± 0.23	–
Oxford Flowers-102	99.68 ± 0.02	**99.74** ± 0.00	99.61 ± 0.02	99.63 ± 0.03	–
VTAB (19 tasks)	**77.63** ± 0.23	76.28 ± 0.46	72.72 ± 0.21	76.29 ± 1.70	–
TPUv3-core-days	2.5k	0.68k	0.23k	9.9k	12.3k

Table 2: Comparison with state of the art on popular image classification benchmarks. We report mean and standard deviation of the accuracies, averaged over three fine-tuning runs. Vision Transformer models pre-trained on the JFT-300M dataset outperform ResNet-based baselines on all datasets, while taking substantially less computational resources to pre-train. ViT pre-trained on the smaller public ImageNet-21k dataset performs well too. *Slightly improved 88.5% result reported in Touvron et al. (2020).

Effect of Pre-Training

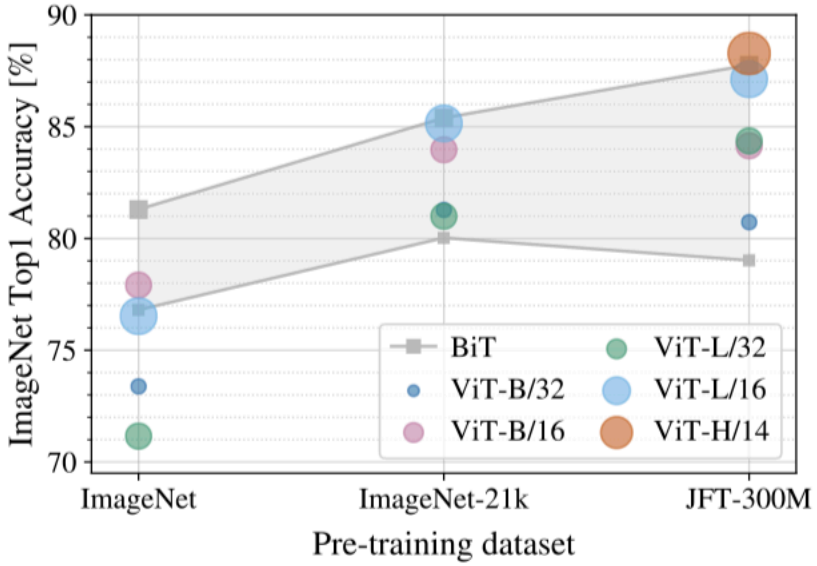


Figure 3: Transfer to ImageNet. While large ViT models perform worse than BiT ResNets (shaded area) when pre-trained on small datasets, they shine when pre-trained on larger datasets. Similarly, larger ViT variants overtake smaller ones as the dataset grows.


ROBOT INTELLIGENCE LAB