ROBOT INTELLIGENCE LAB

Self-Supervised Learning

Transformer-based SSL

Sungjoon Choi, Korea University

kakaobrain

Content

ROBOT INTELLIGENCE LAB

- Transformer
- ViT
- DINO
- EsViT

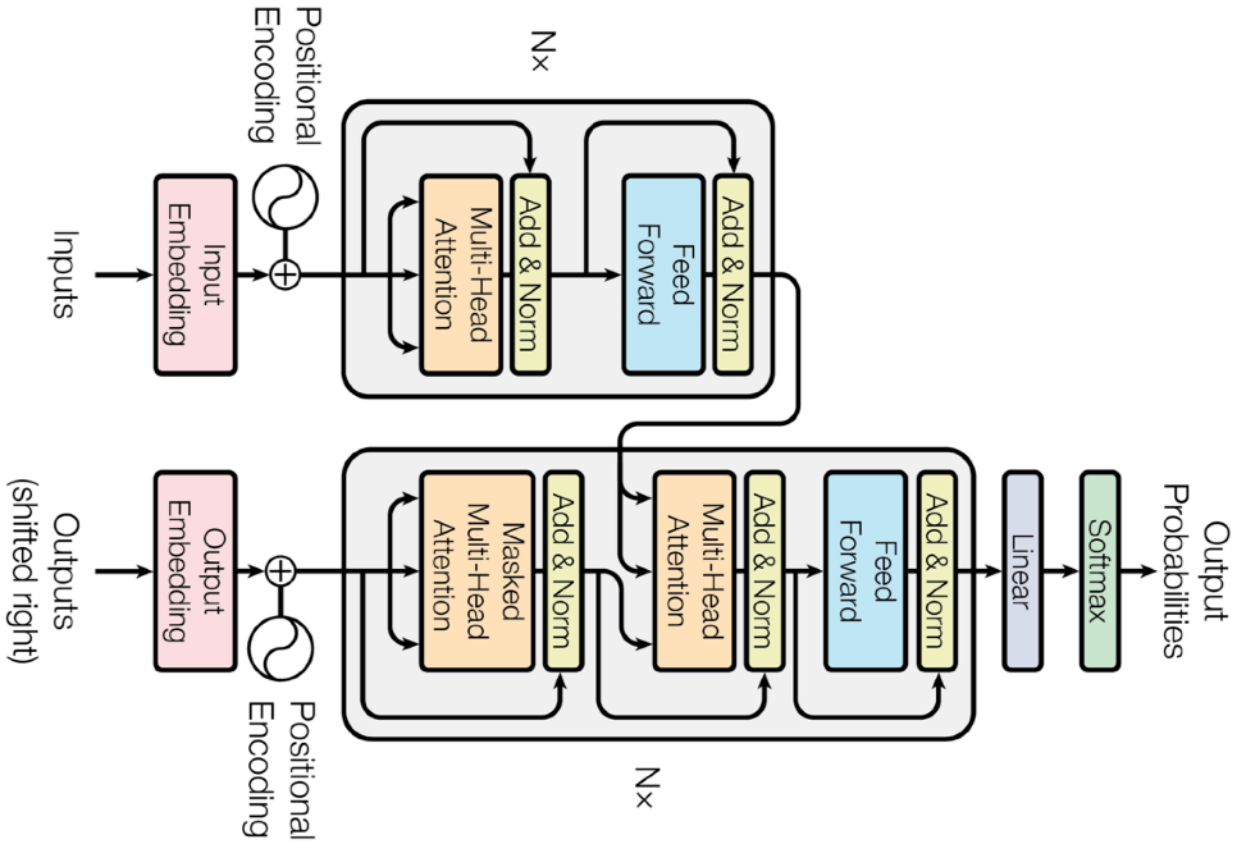
ROBOT INTELLIGENCE LAB

Transformer

"Attention is All You Need," 2017

Transformer

ROBOT INTELLIGENCE LAB



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

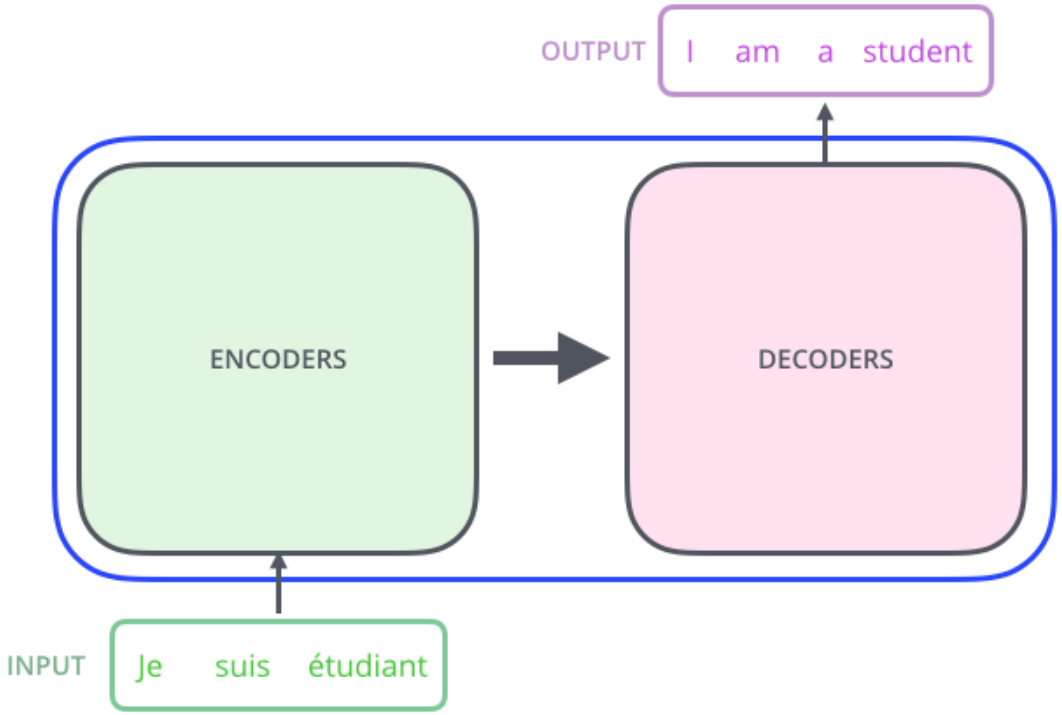
Bird's eye view

ROBOT INTELLIGENCE LAB

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

Glide down a little

ROBOT INTELLIGENCE LAB

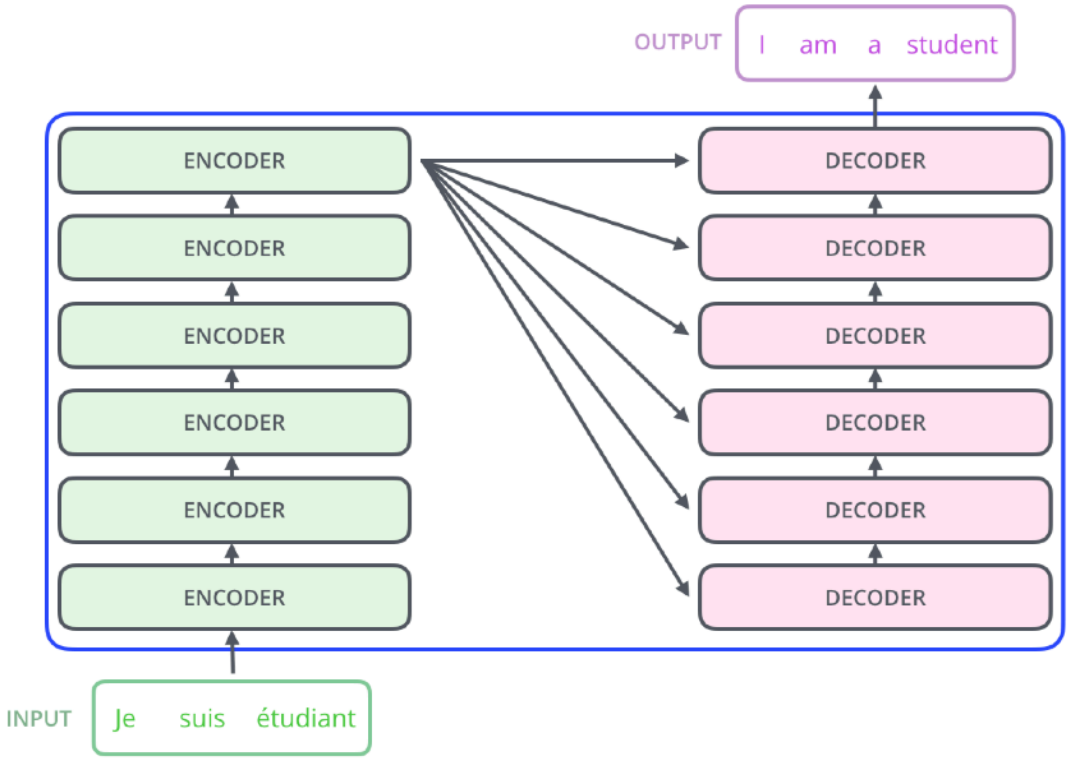


It consists of an **encoder** and a **decoder**

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

Glide down a little more

ROBOT INTELLIGENCE LAB

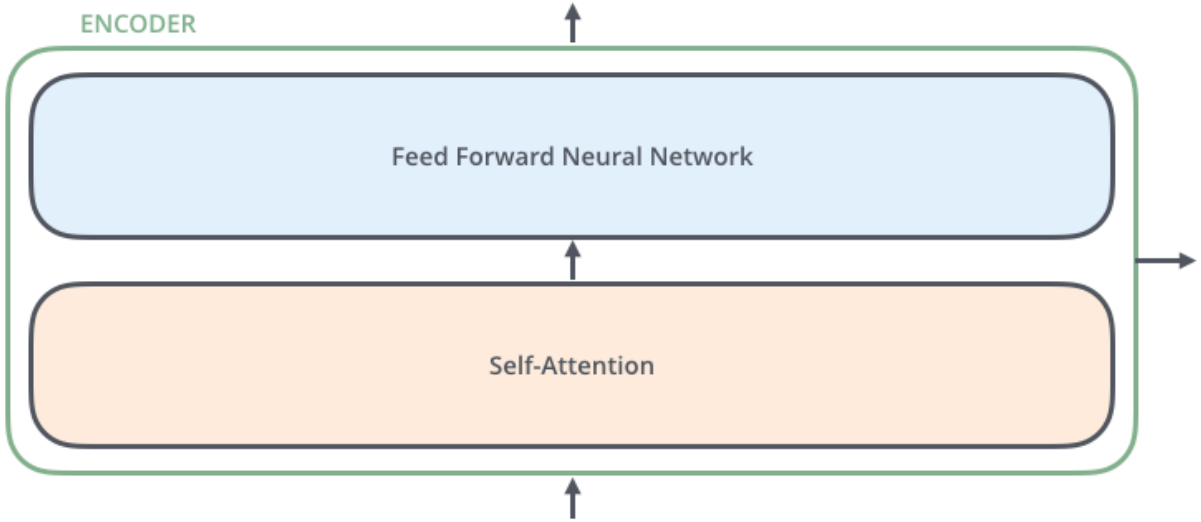


What information is being set from an **encoder** to a **decoder**?

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

Encoder

ROBOT INTELLIGENCE LAB

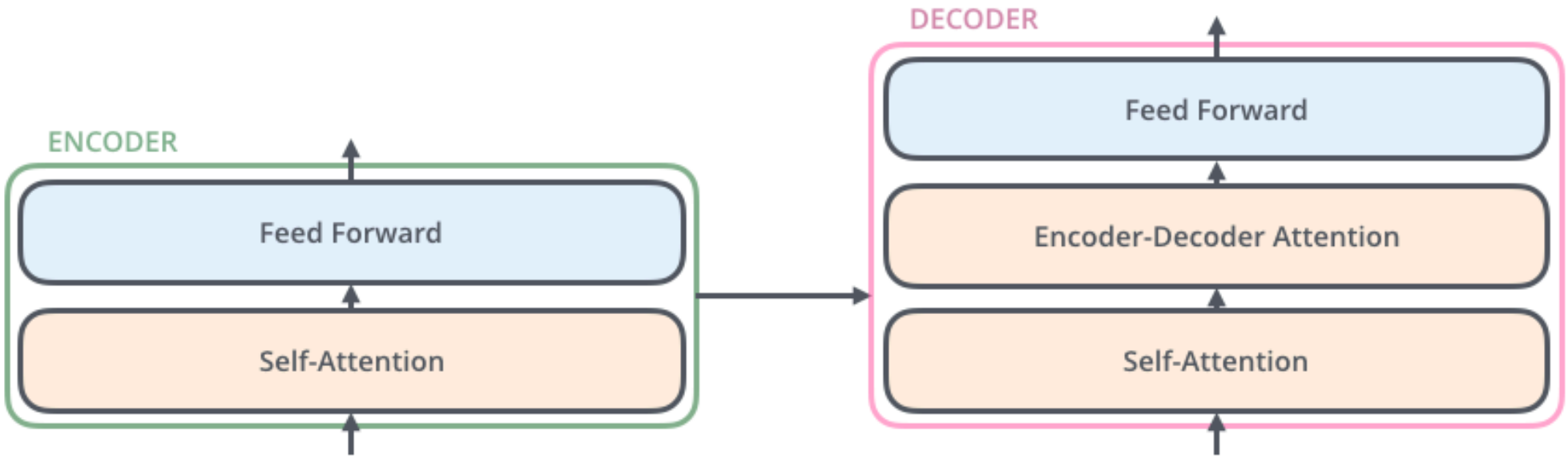


Each encoder has an identical structure where the **Self-Attention** layer is the cornerstone of Transformer.

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

Encoder-Decoder

ROBOT INTELLIGENCE LAB



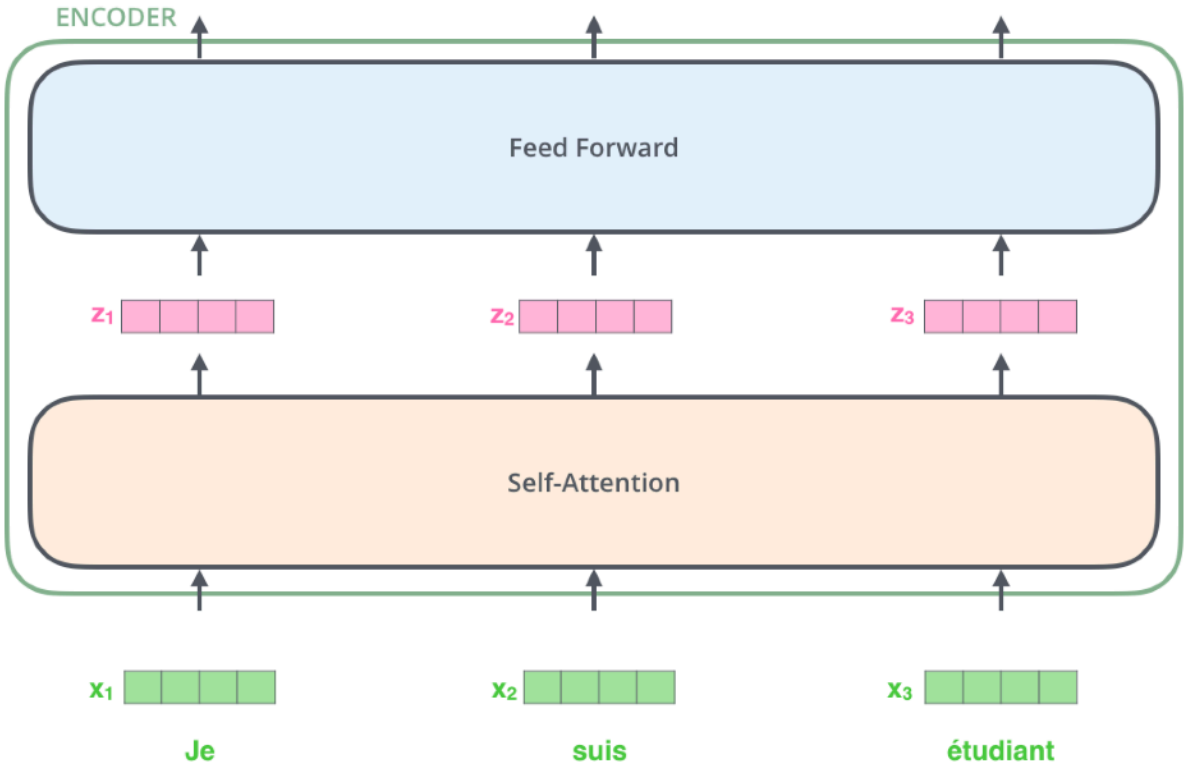
Decoder has an additional structure, **Encoder-Decoder Attention**, along with self-attention block

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

Input

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

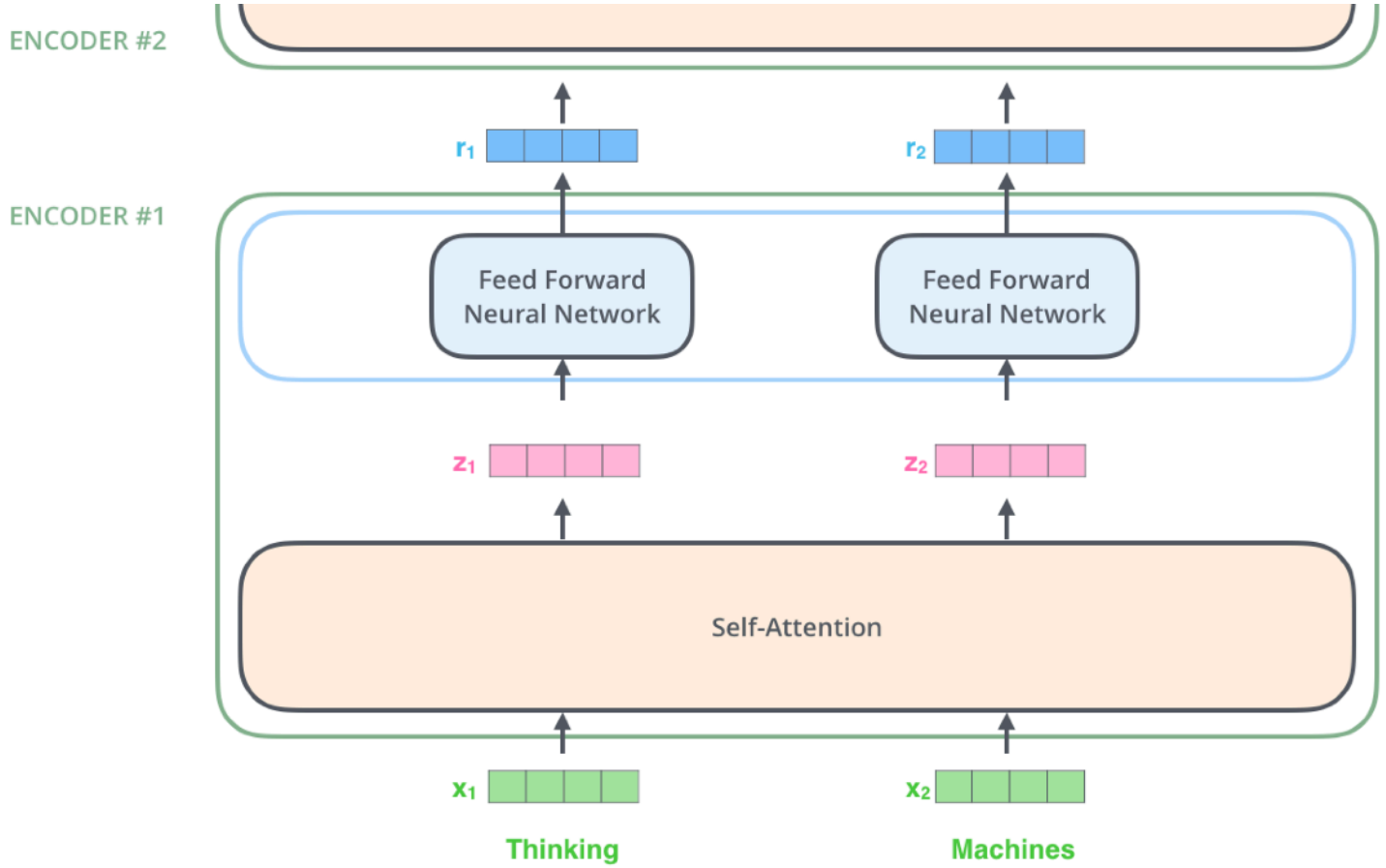
Encoder



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

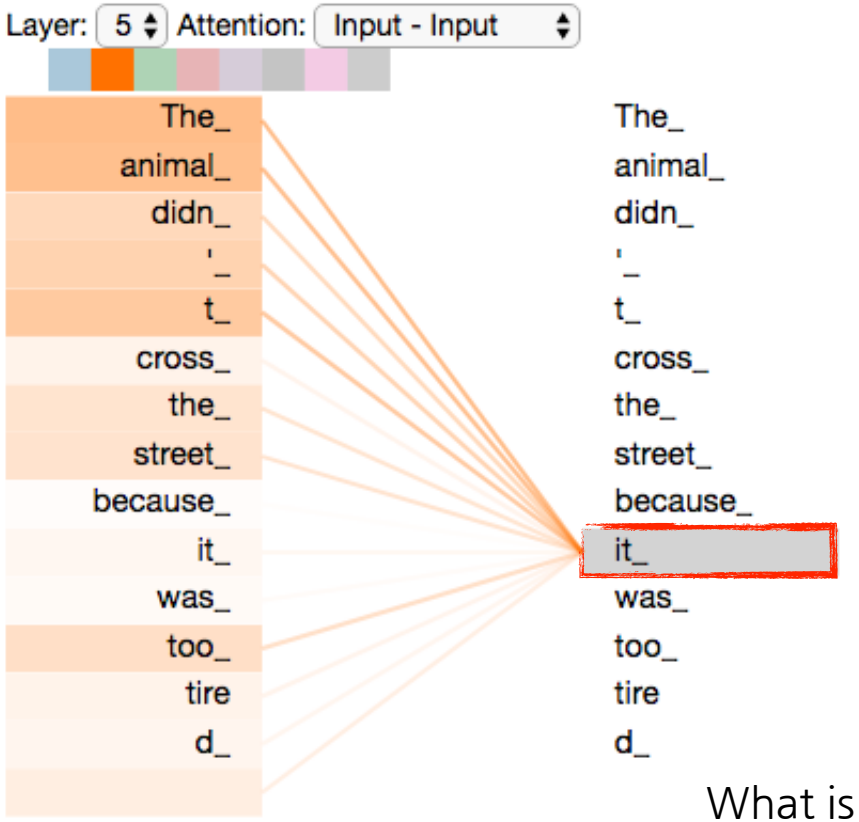


Suppose we have two words, **Thinking** and **Machines**, as an input sequence.

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

Self Attention

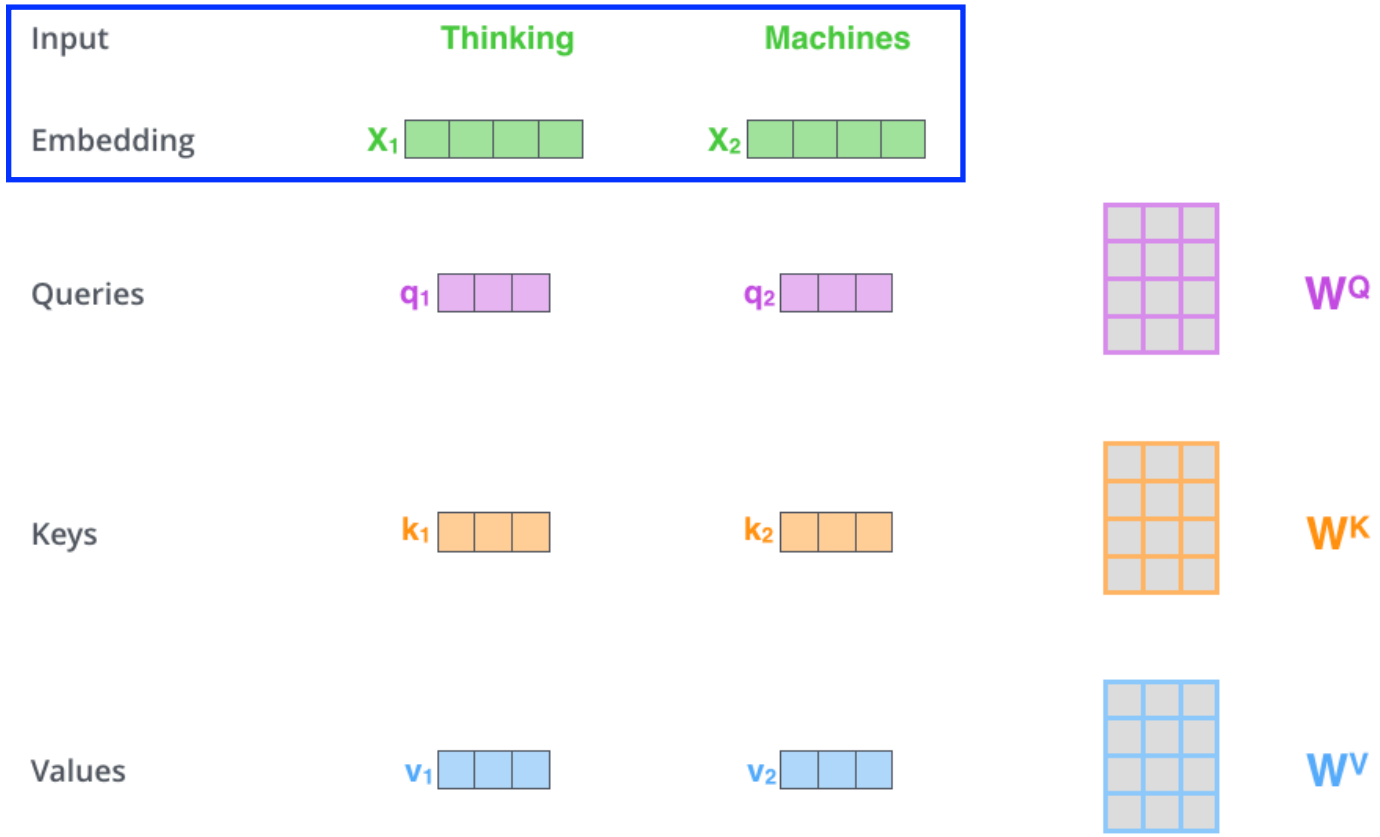
ROBOT INTELLIGENCE LAB



What is it referring to?

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

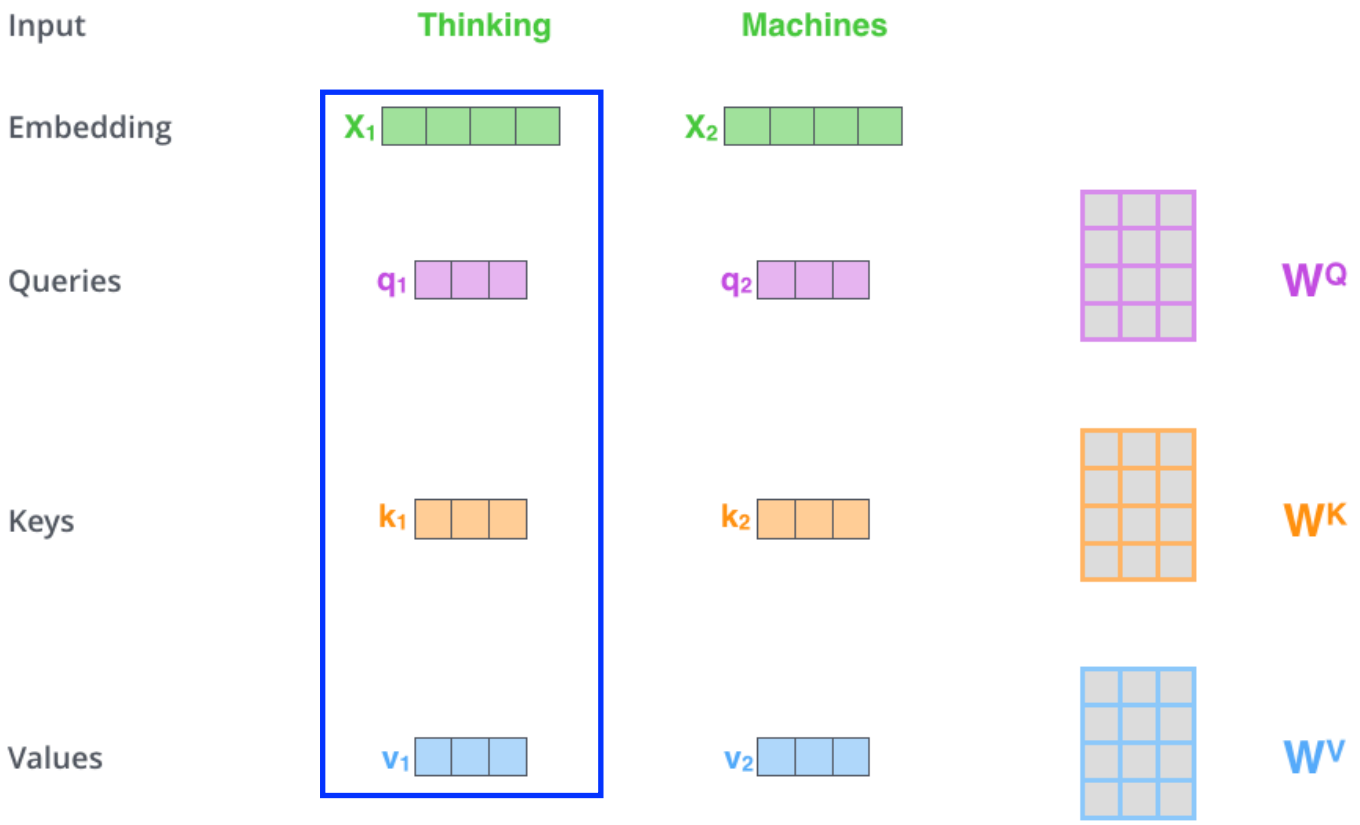
Encoder



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

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

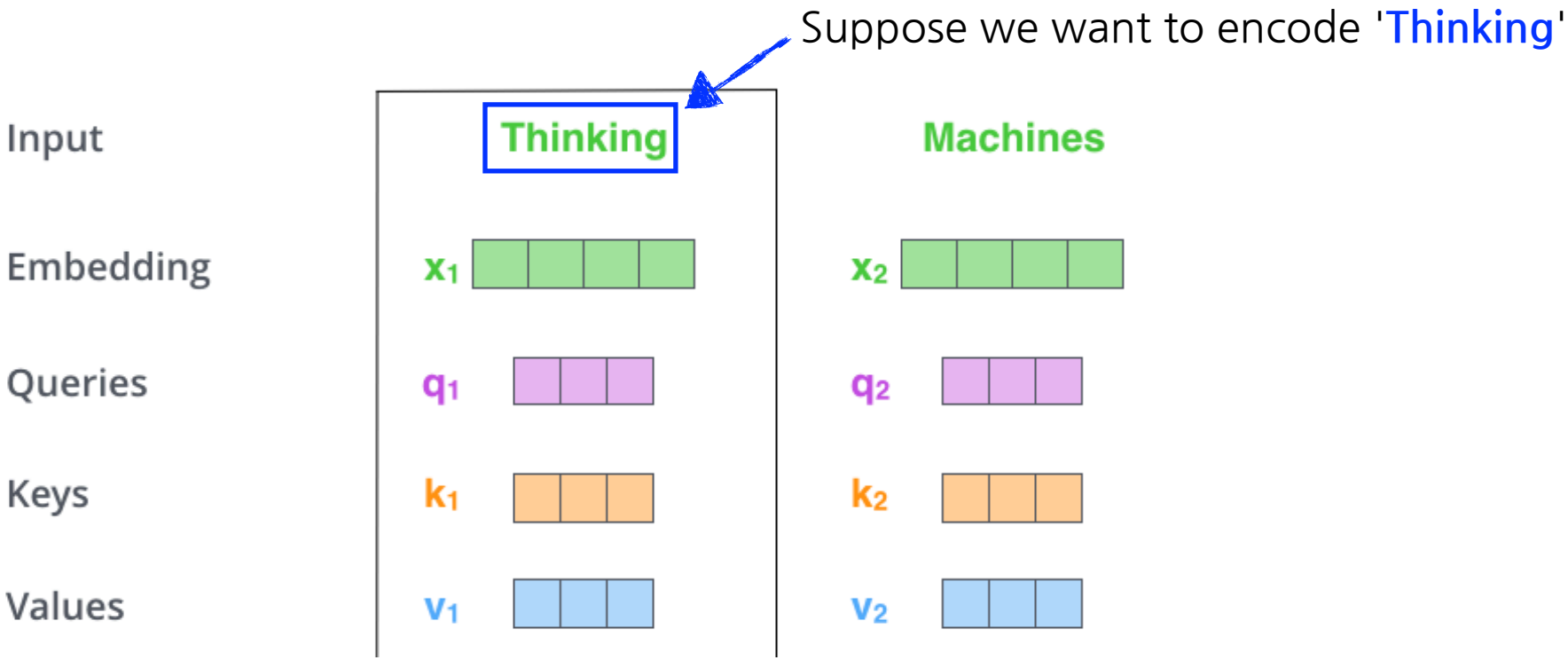
Encoder



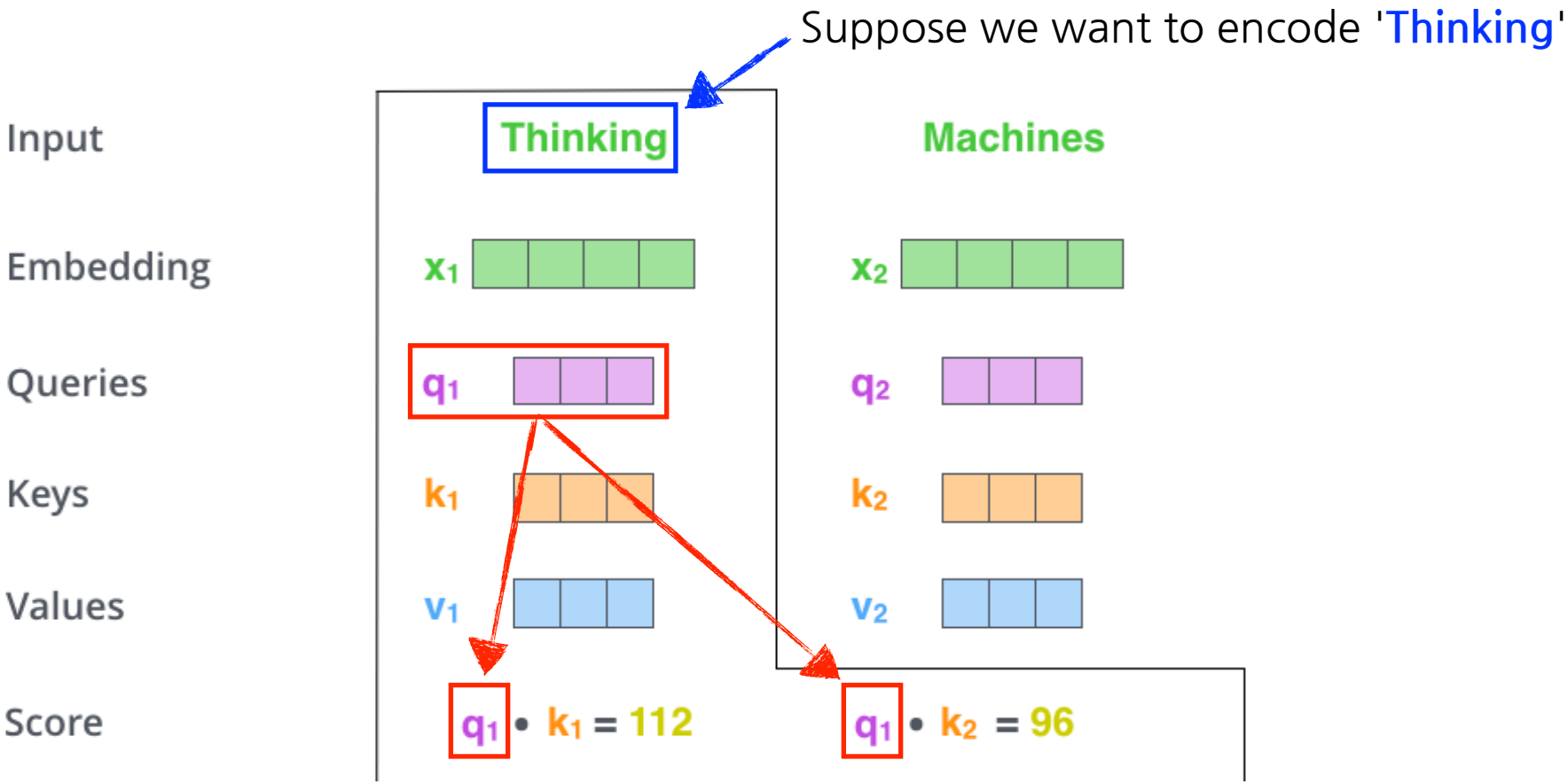
Query, **Key**, and **Value** vectors are computed from the **embedding** vecotor using three different networks.

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

Encoding 'Thinking'

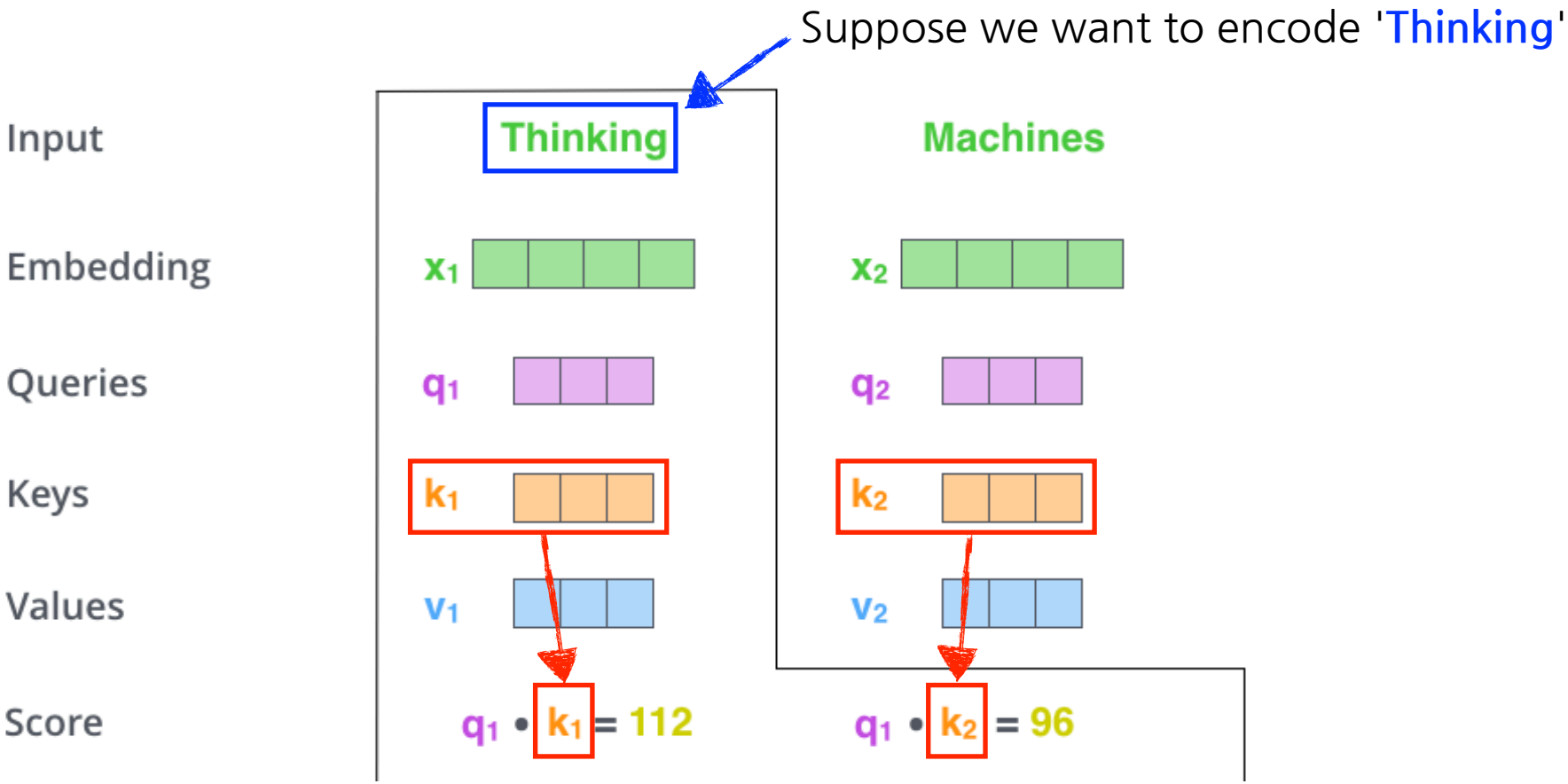


Encoding 'Thinking'



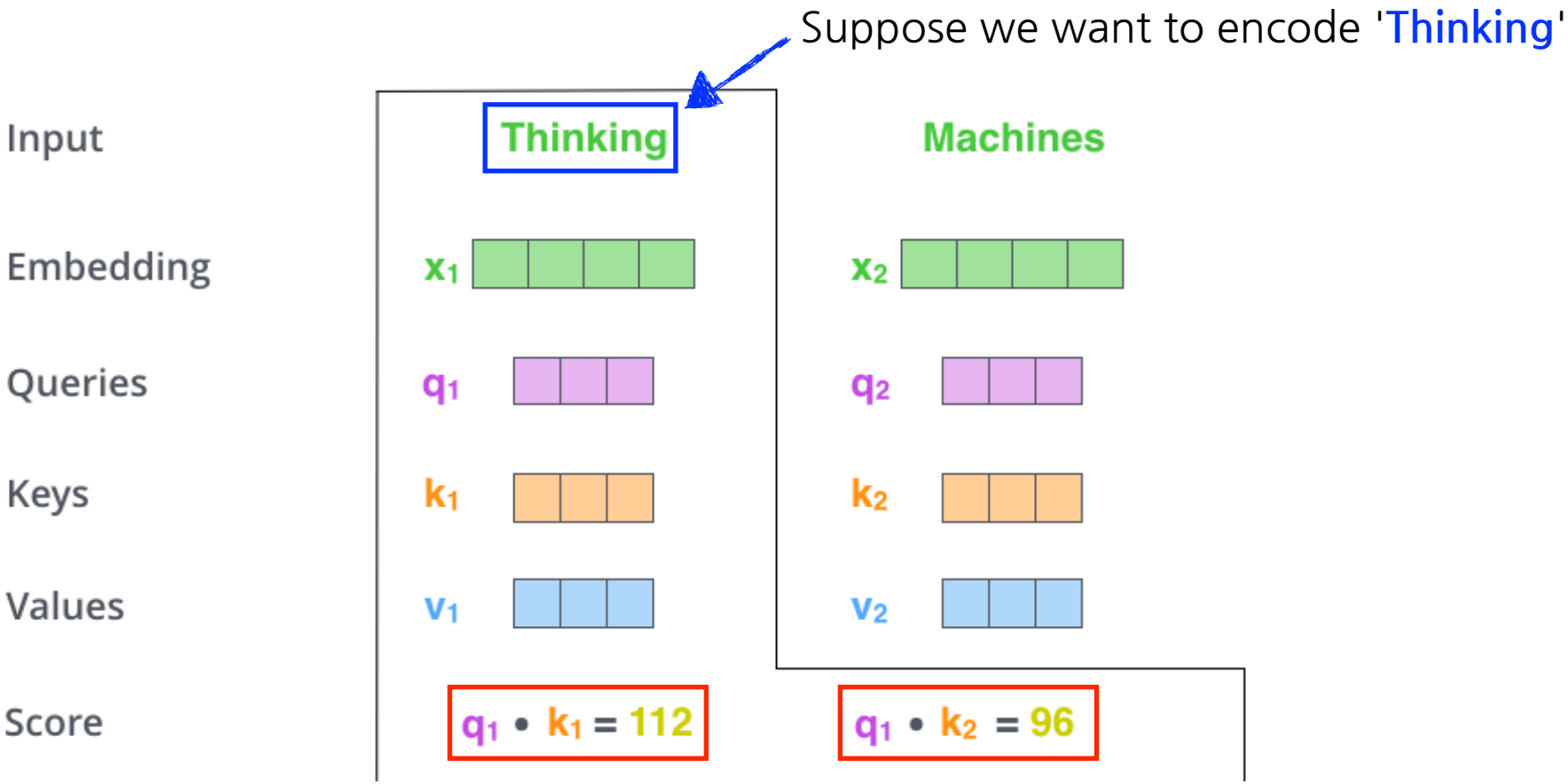
Query vector comes from the encoding word (i.e., Thinking)

Encoding 'Thinking'



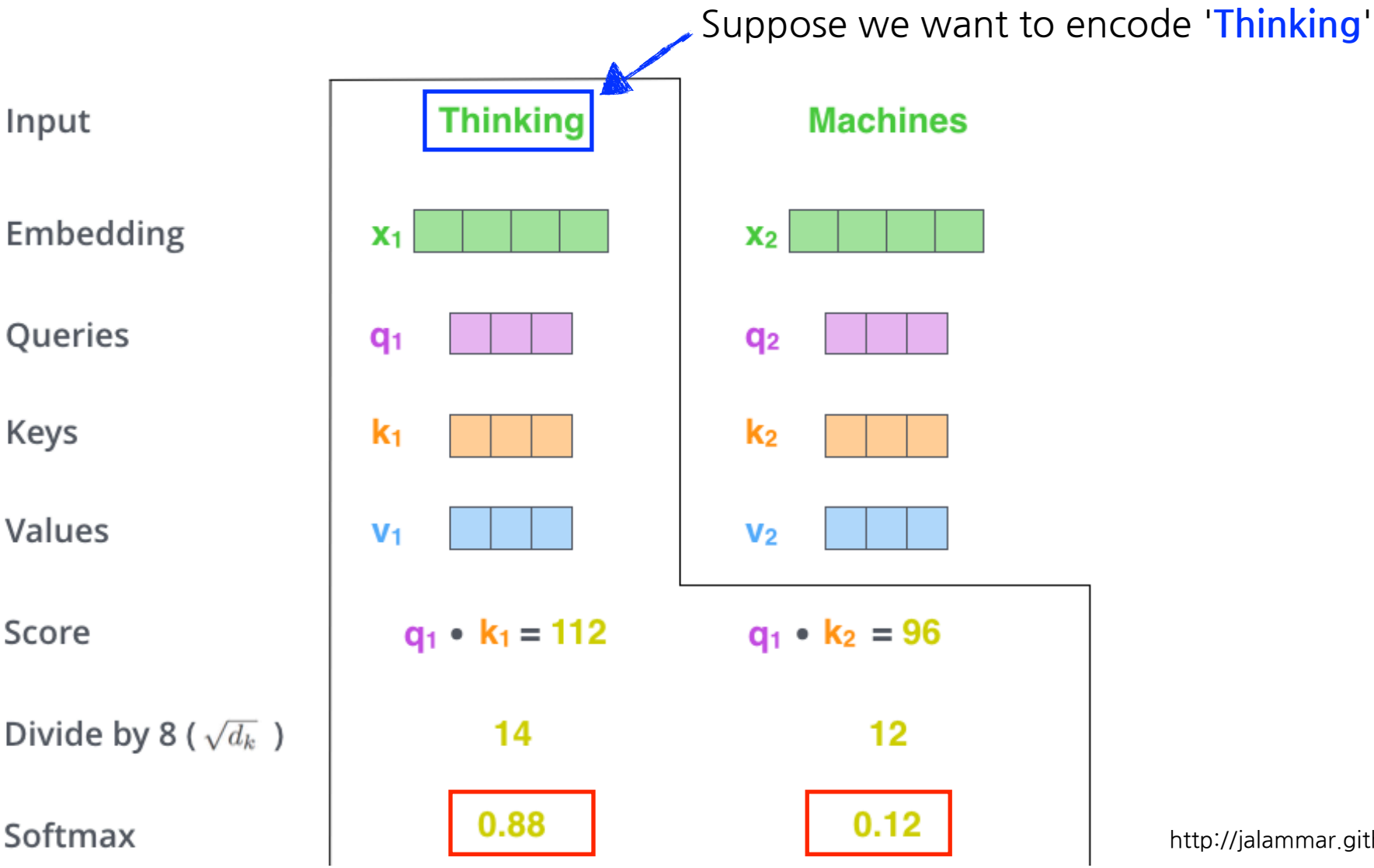
Key vector comes from the corresponding target words (i.e., **Thinking** and **Machines**)

Encoding 'Thinking'



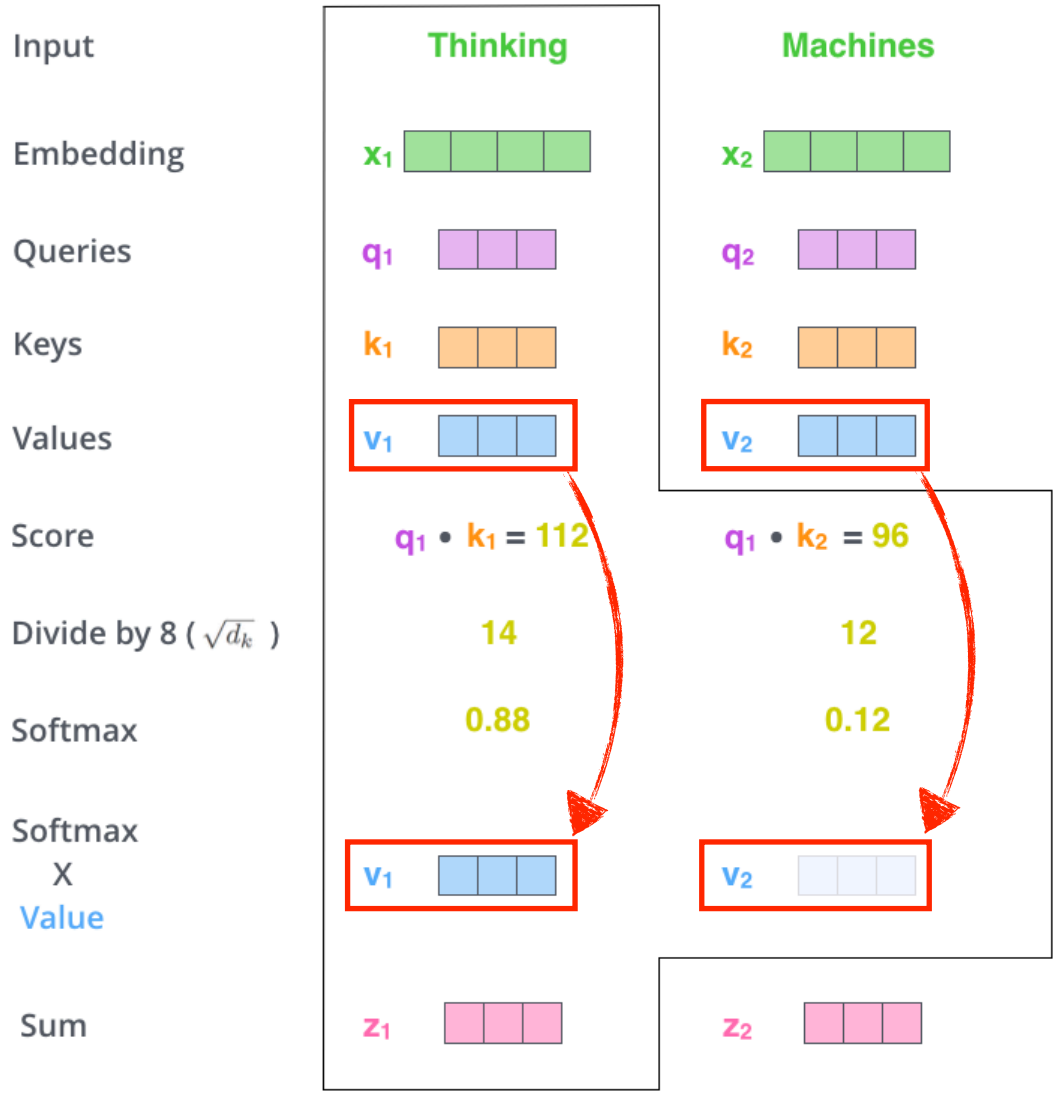
The score of each input is computed from the inner products between **query** and **key** vectors

Encoding 'Thinking'



Encoding 'Thinking'

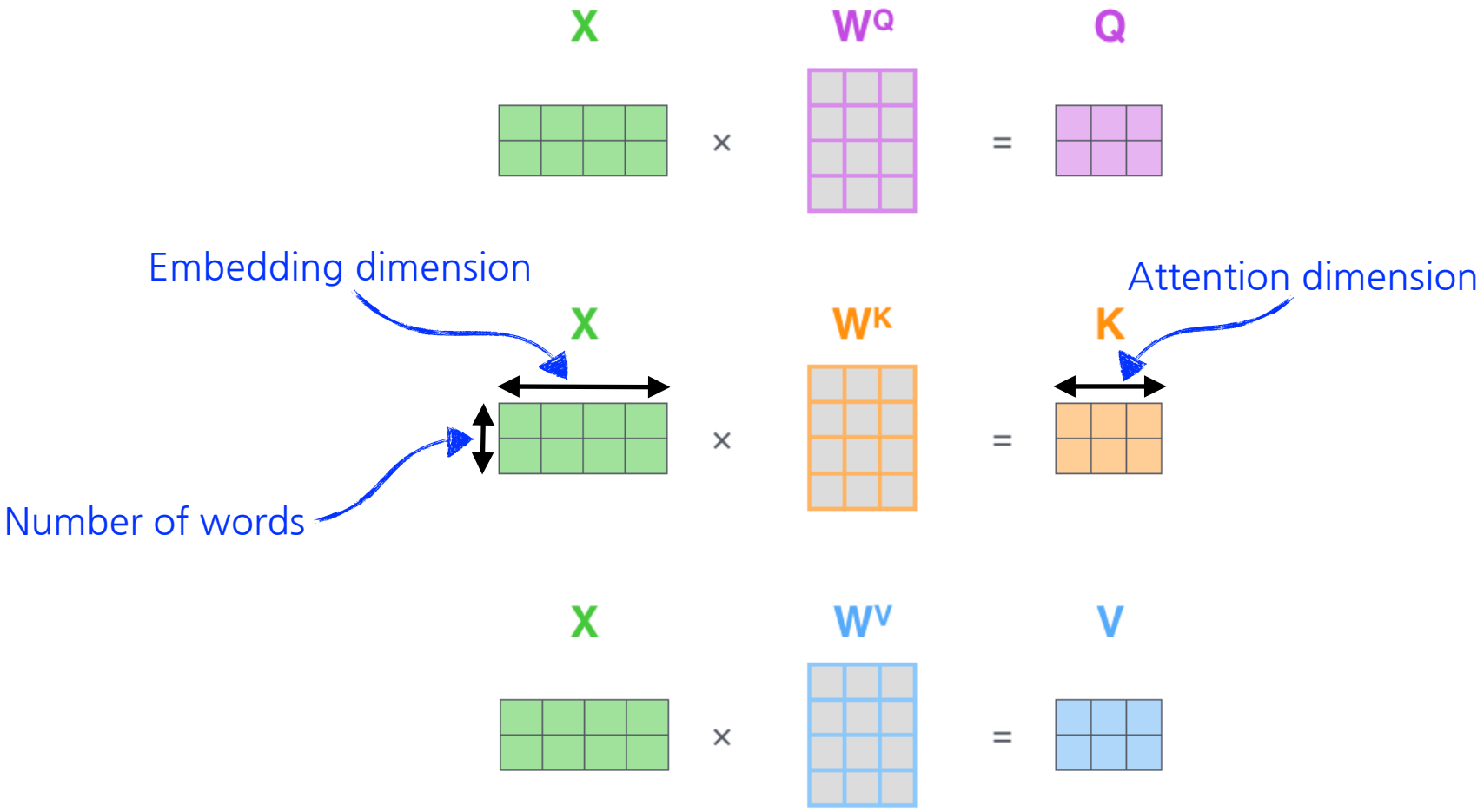
ROBOT INTELLIGENCE LAB



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

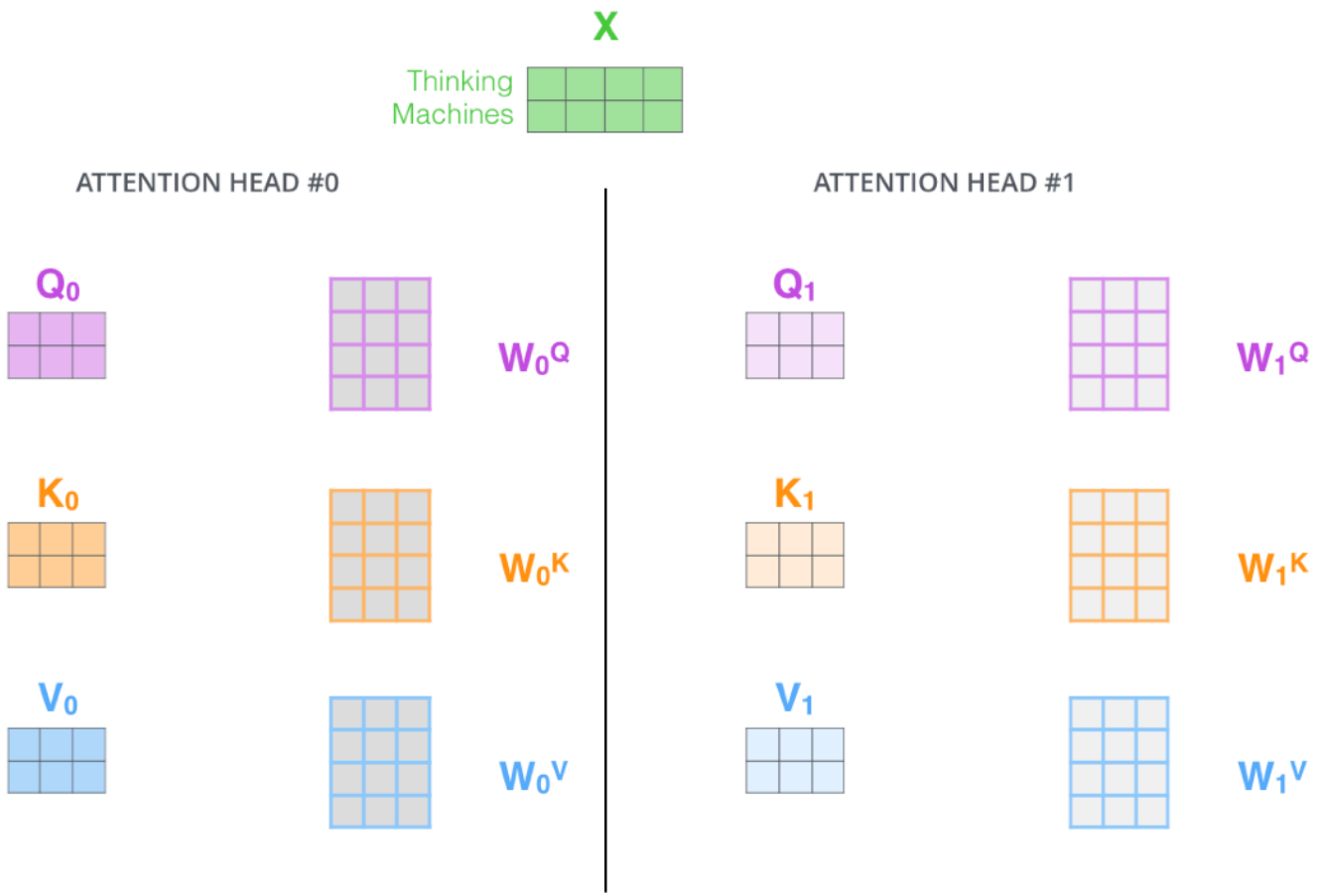
Computing Q, K, and V

ROBOT INTELLIGENCE LAB



Multi-Head Attention (MHA)

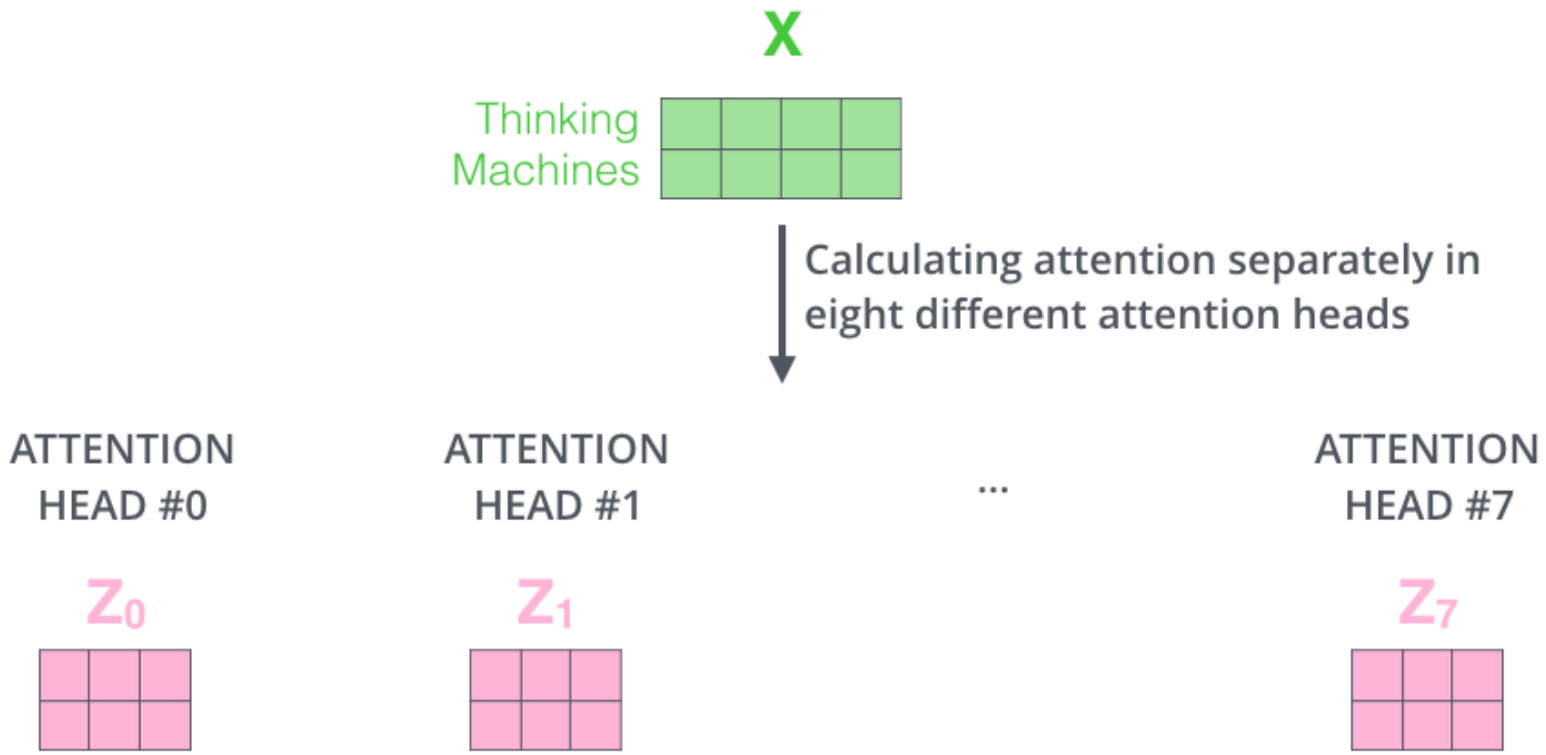
ROBOT INTELLIGENCE LAB



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

Multi-Head Attention (MHA)

ROBOT INTELLIGENCE LAB



How do we accumulate multiple attention results from MHA?

Simple Linear Map

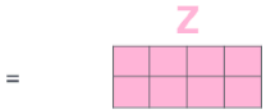
ROBOT INTELLIGENCE LAB

1) Concatenate all the attention heads

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



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

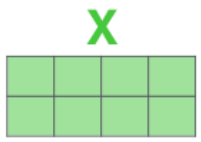
Summary

ROBOT INTELLIGENCE LAB

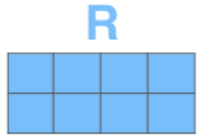
1) This is our input sentence*

Thinking
Machines

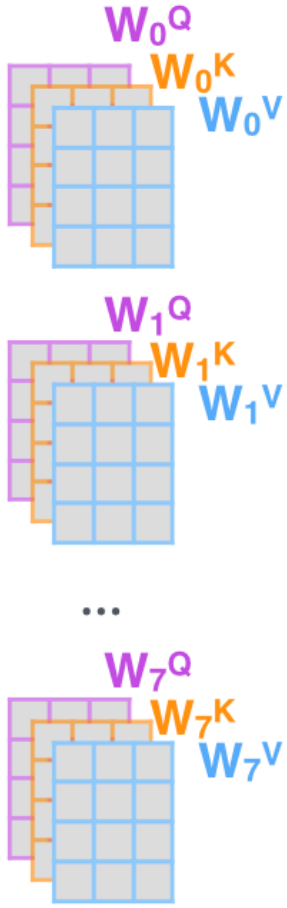
2) We embed each word*



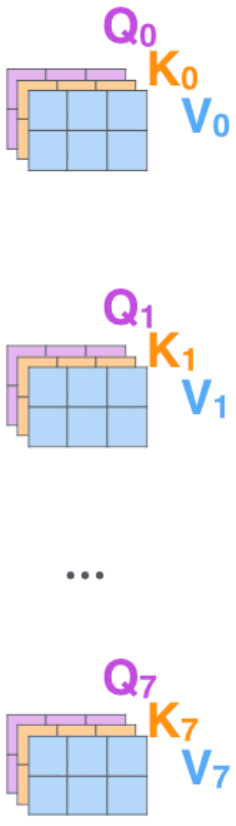
* In all encoders other than #0, we don't need embedding. We start directly with the output of the encoder right below this one



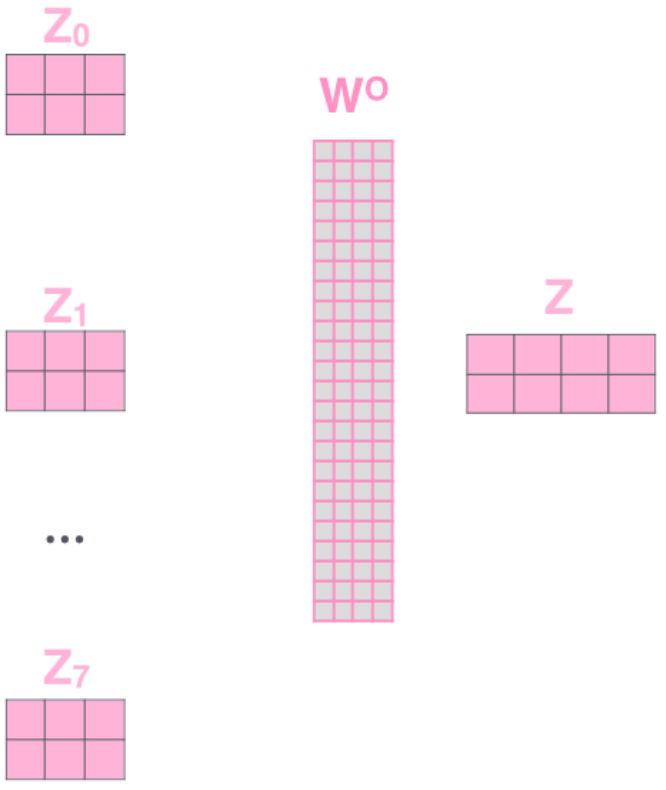
3) Split into 8 heads. We multiply X or R with weight matrices



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



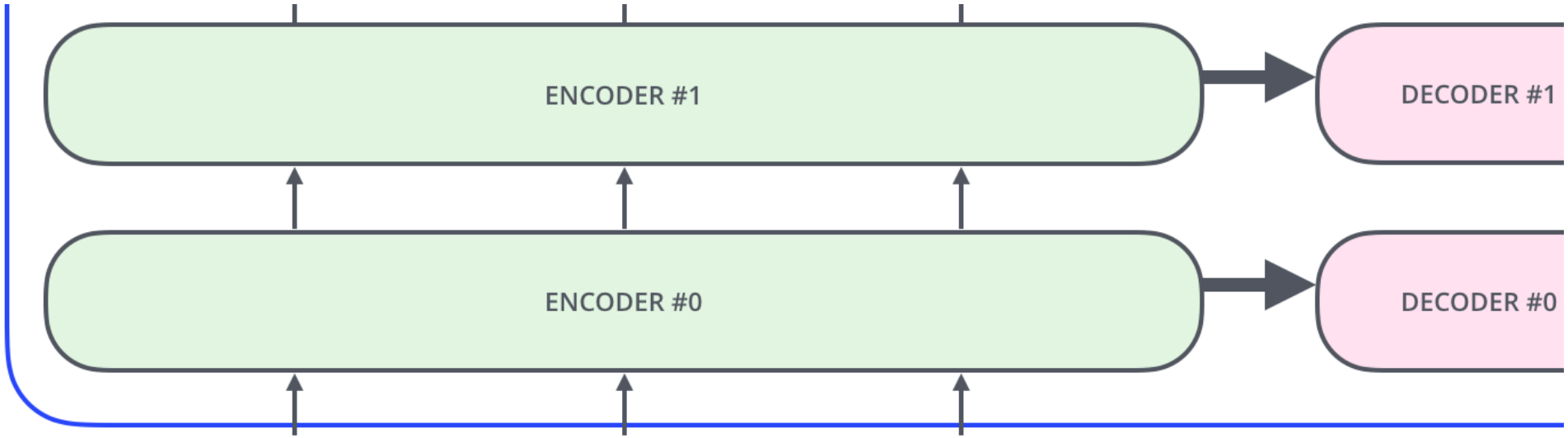
Ordering?

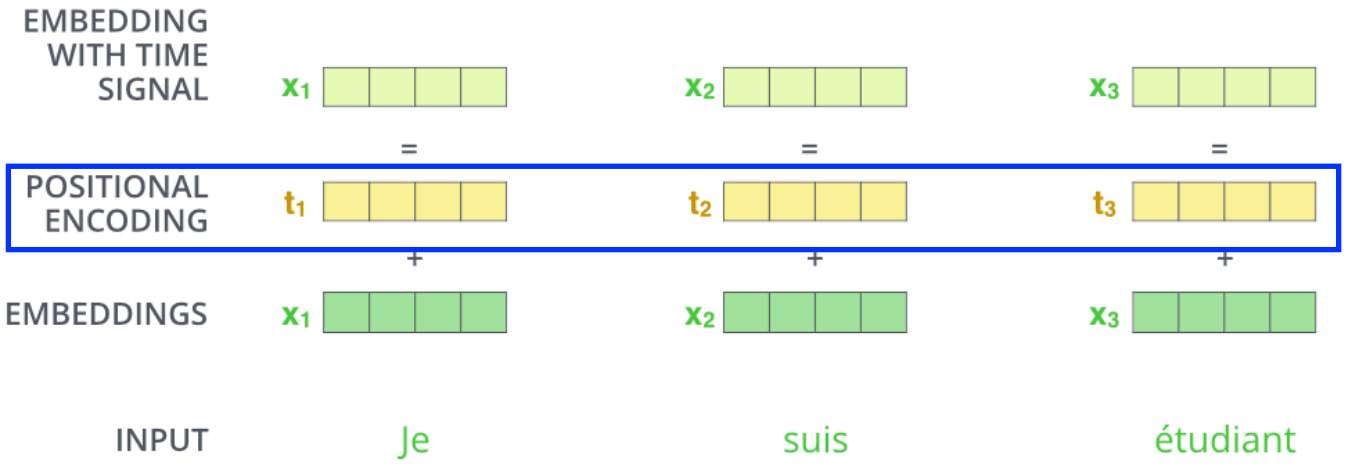
ROBOT INTELLIGENCE LAB

Is Transformer **input-order dependent**?

Positional Encoding

ROBOT INTELLIGENCE LAB

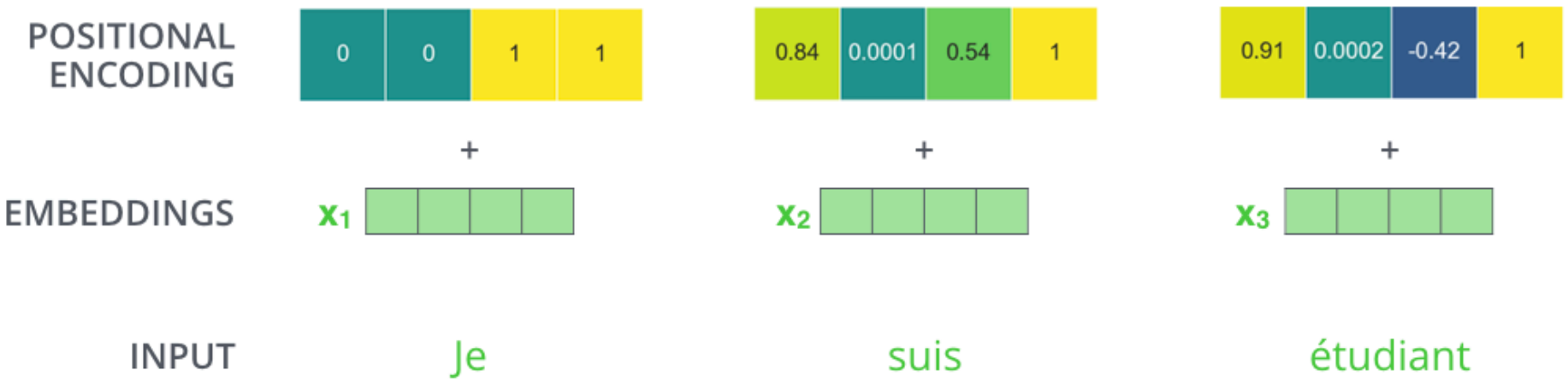




http://jalammar.github.io/illustrated-transformer/

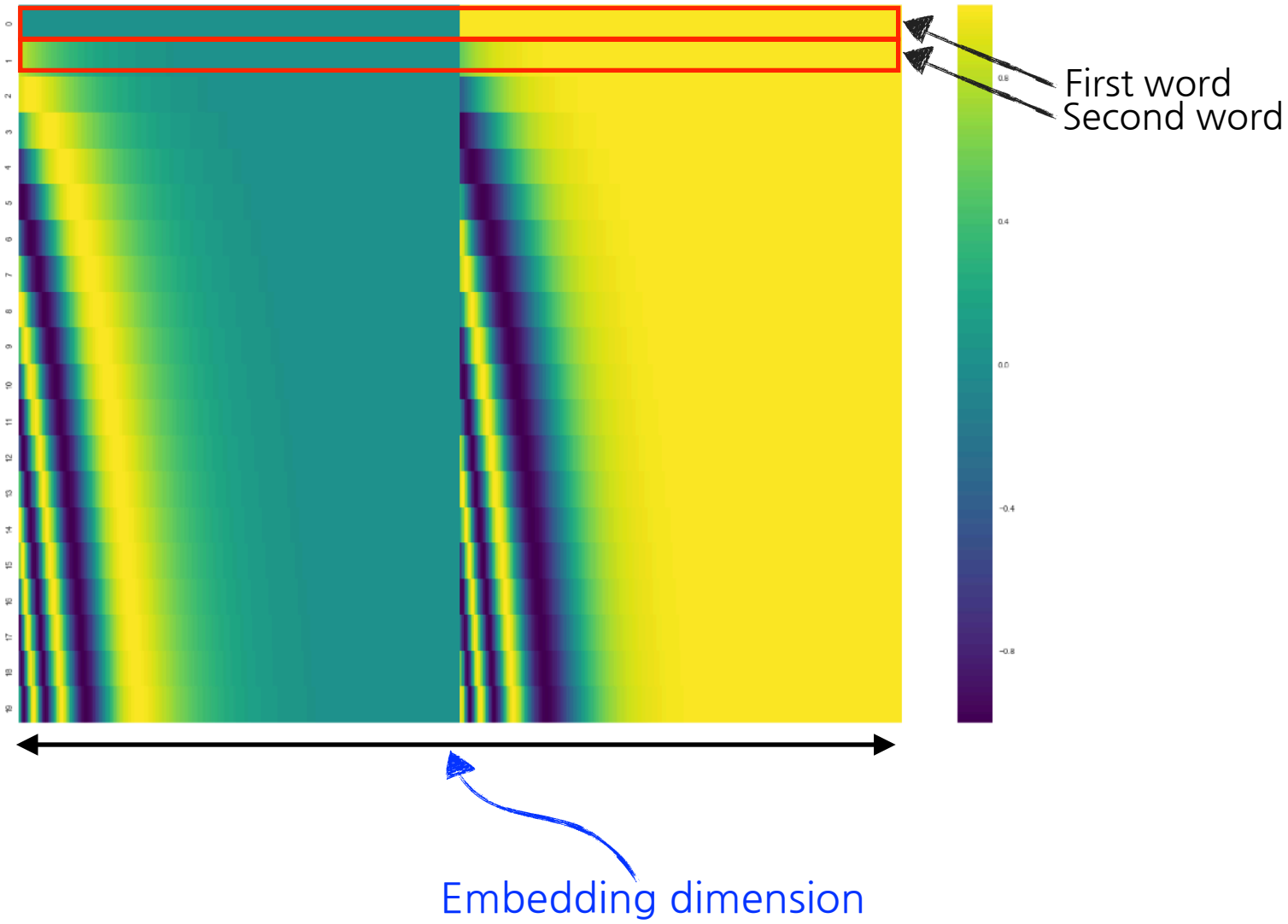
Positional Encoding

ROBOT INTELLIGENCE LAB



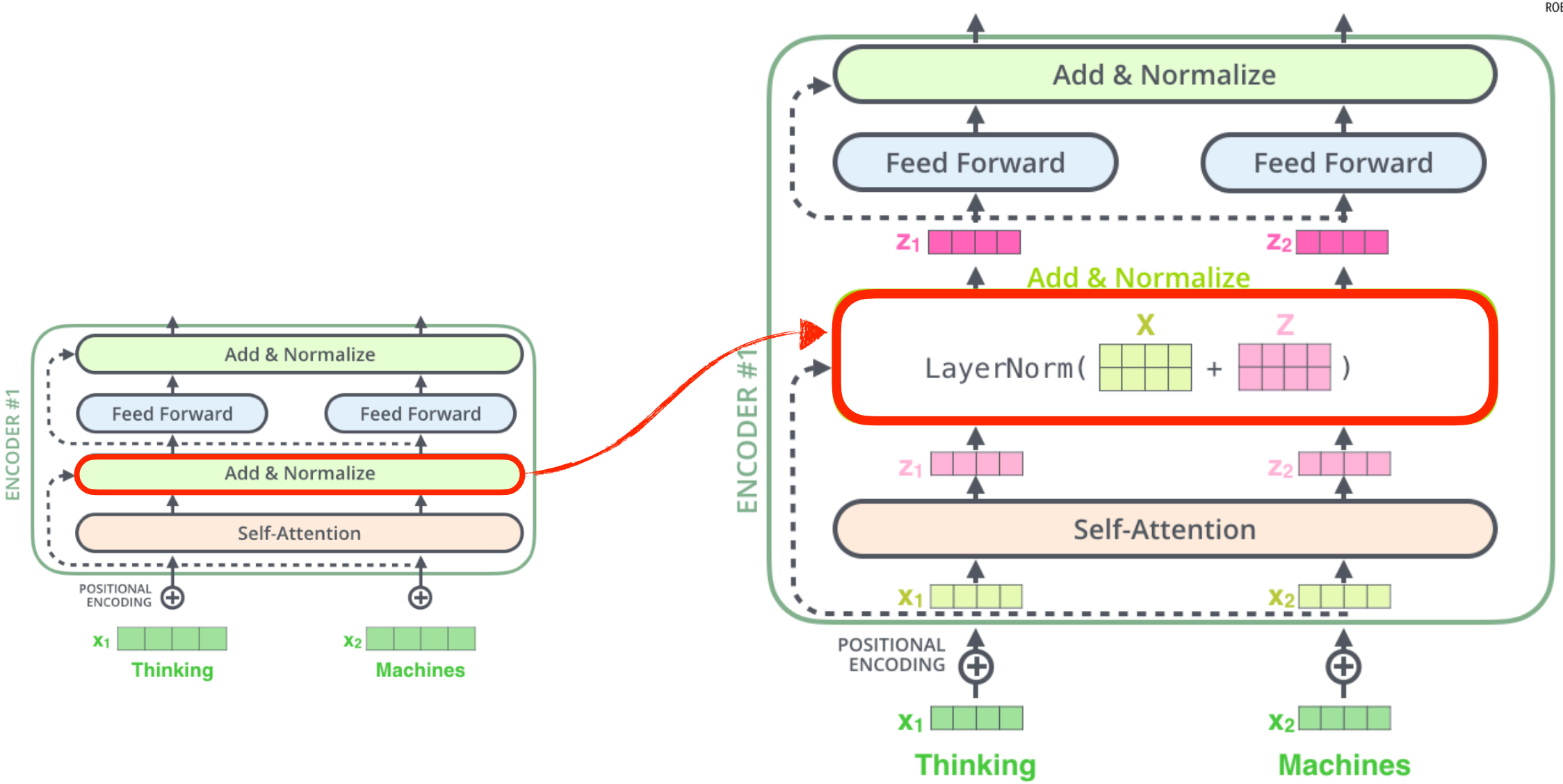
Positional Encoding

ROBOT INTELLIGENCE LAB



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

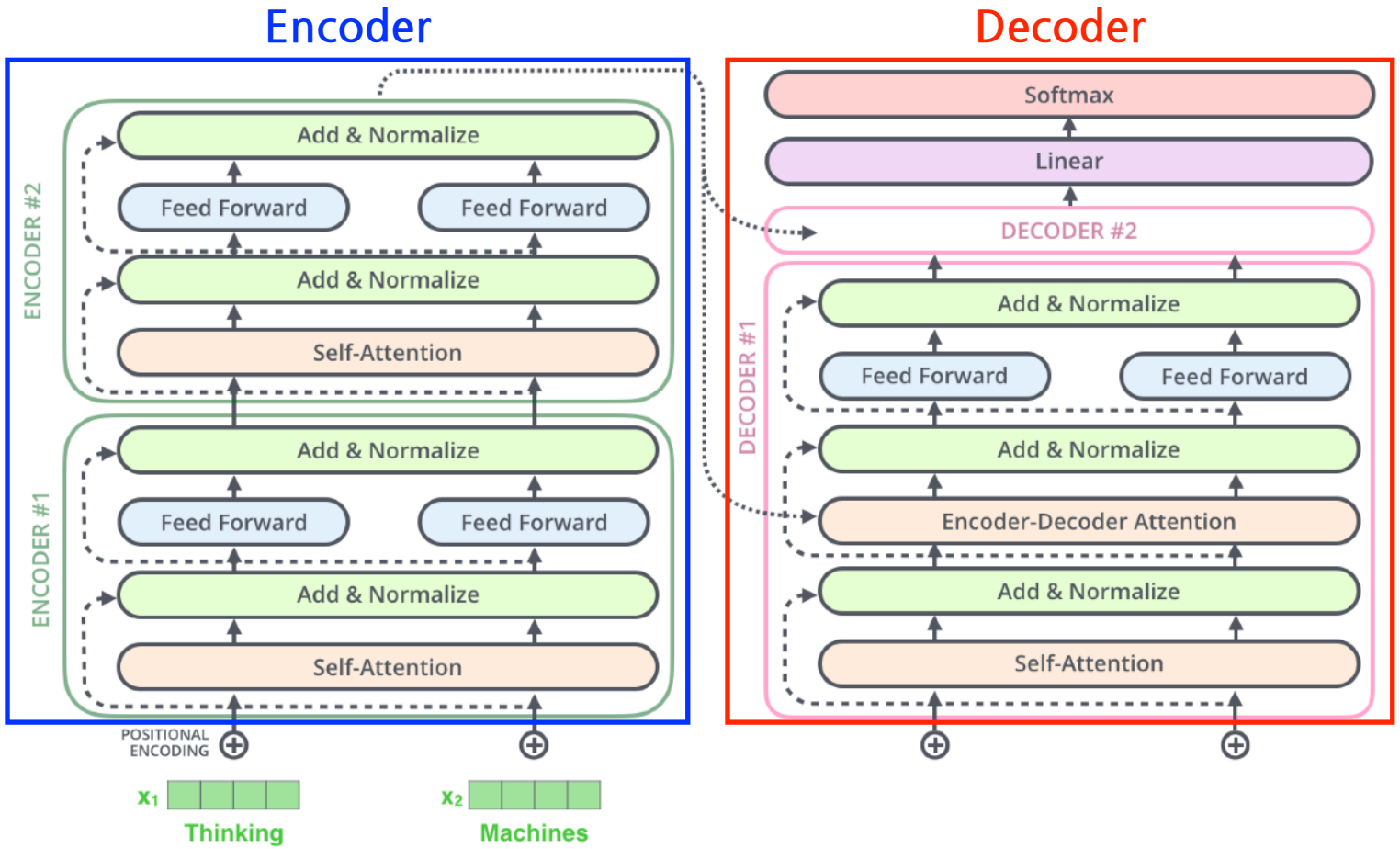
Encoder



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

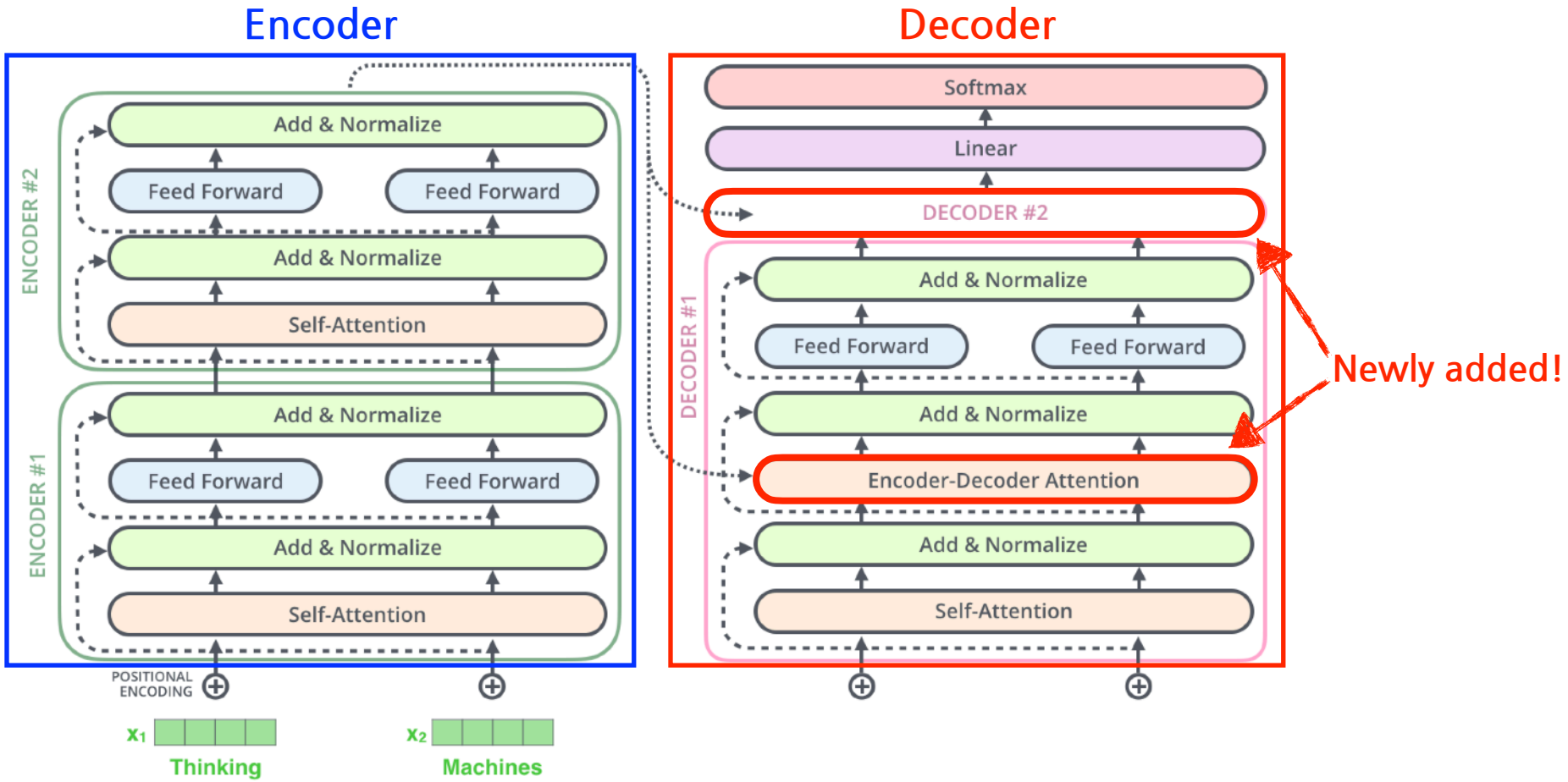
Encoder-Decoder

ROBOT INTELLIGENCE LAB



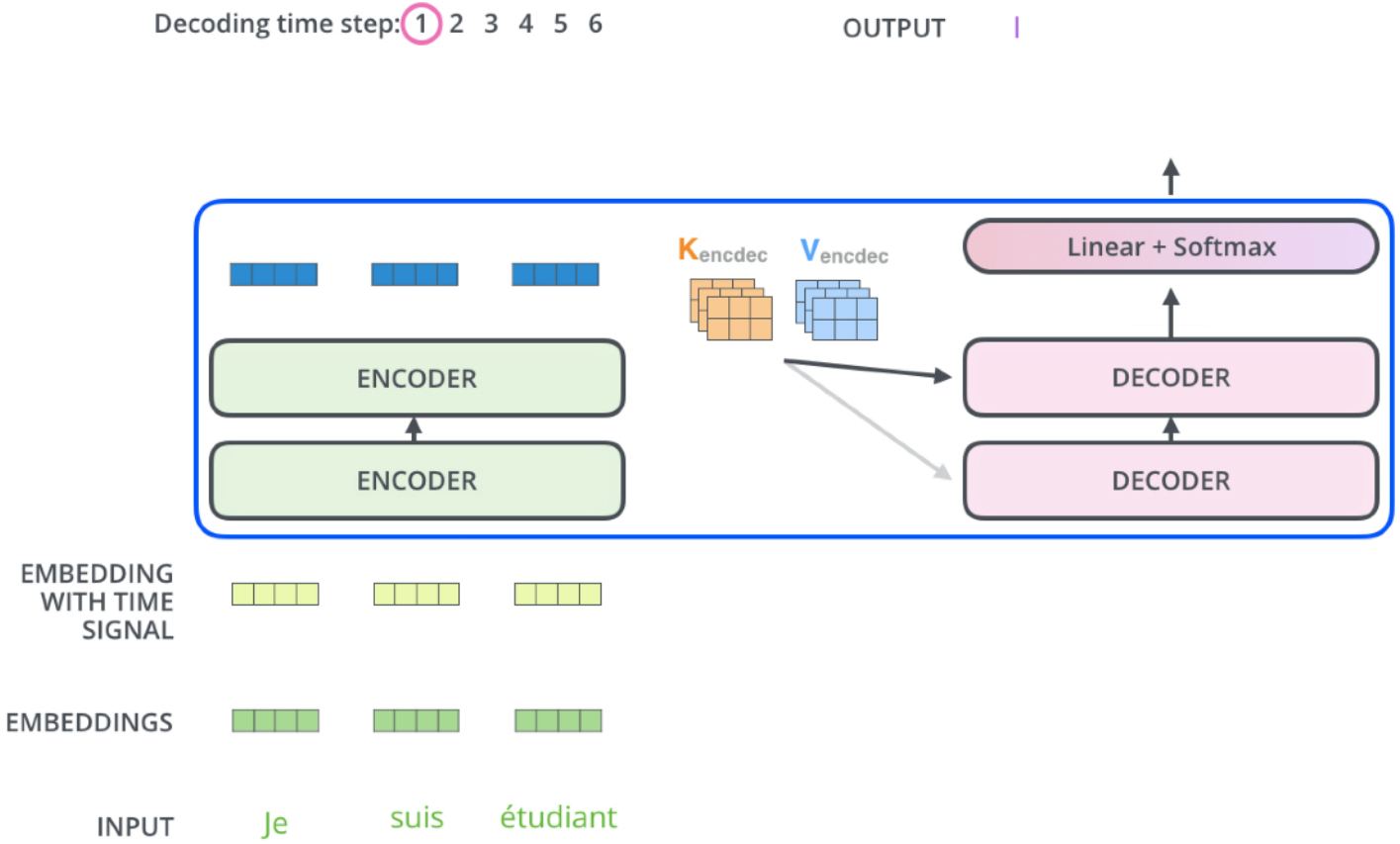
Both Encoder and Decoder have similar architectures.

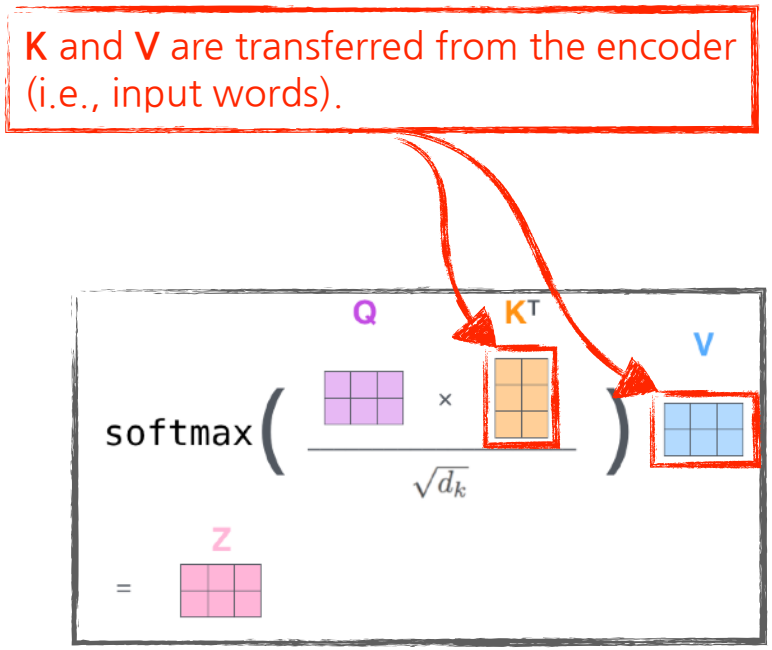
Encoder-Decoder



What is the information being sent from Encoder to Decoder?

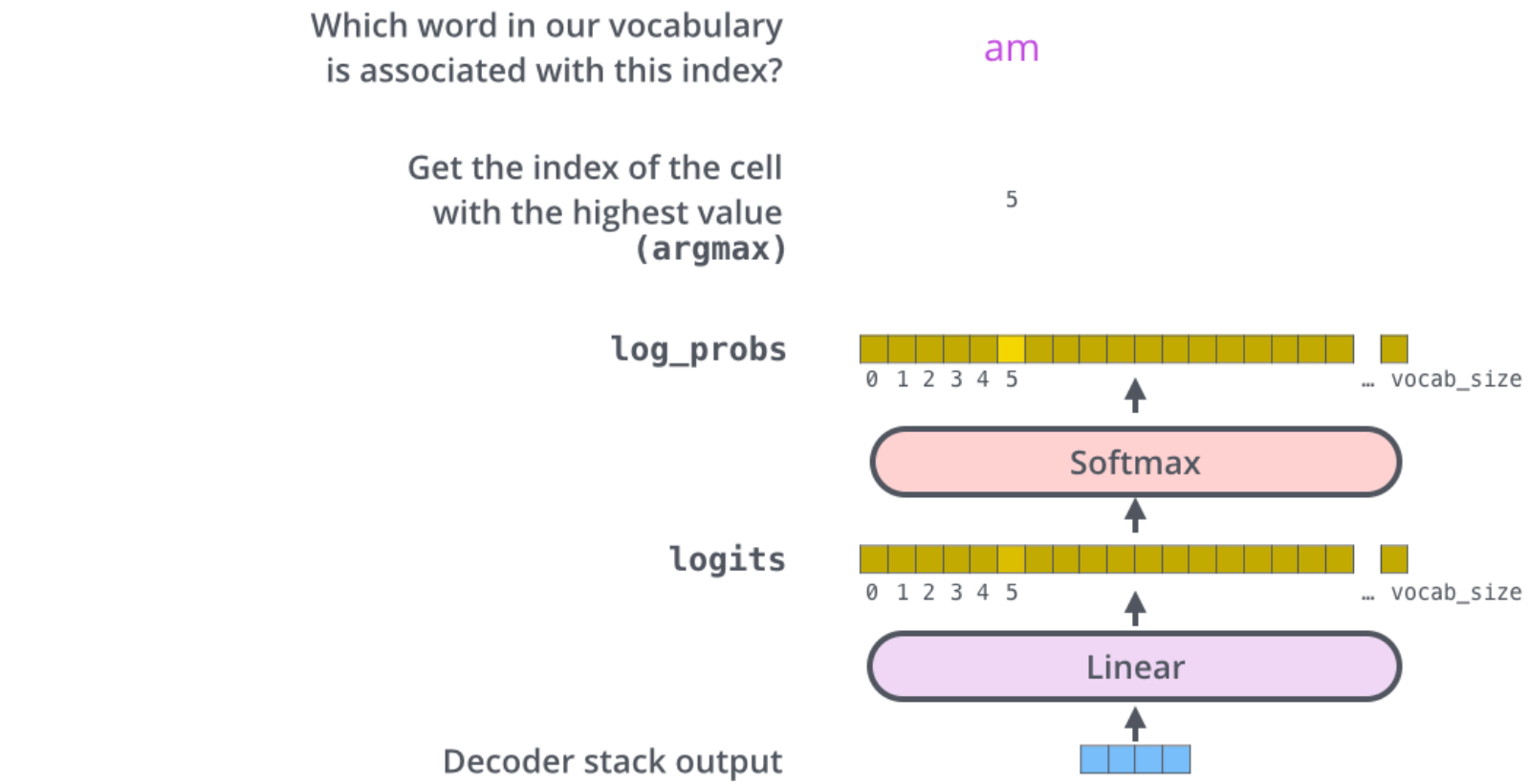
Encoder-Decoder





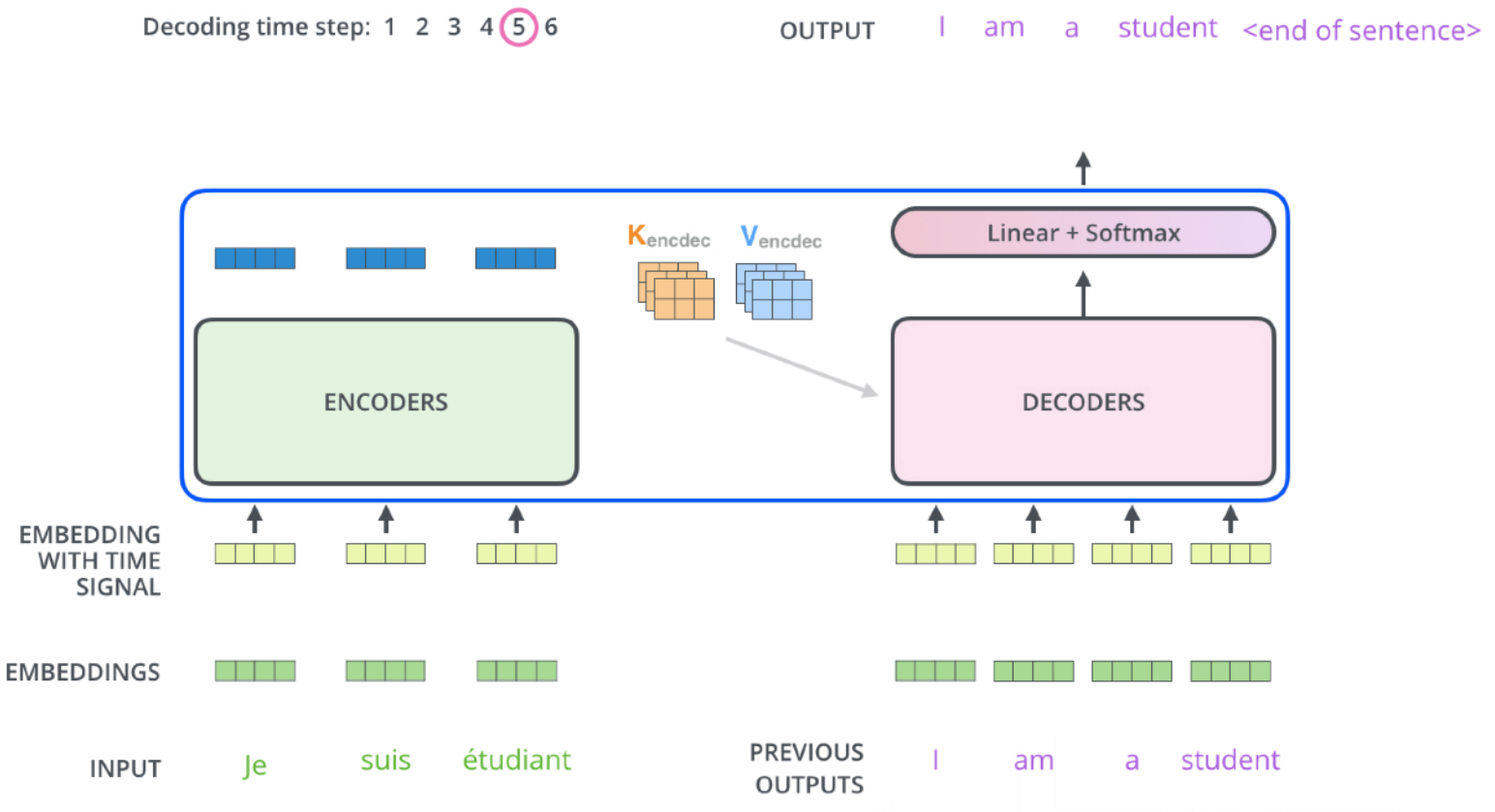
Key (K) and **Value (V)** vectors are transferred from Encoder to Decoder

Decoder



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

Decoder



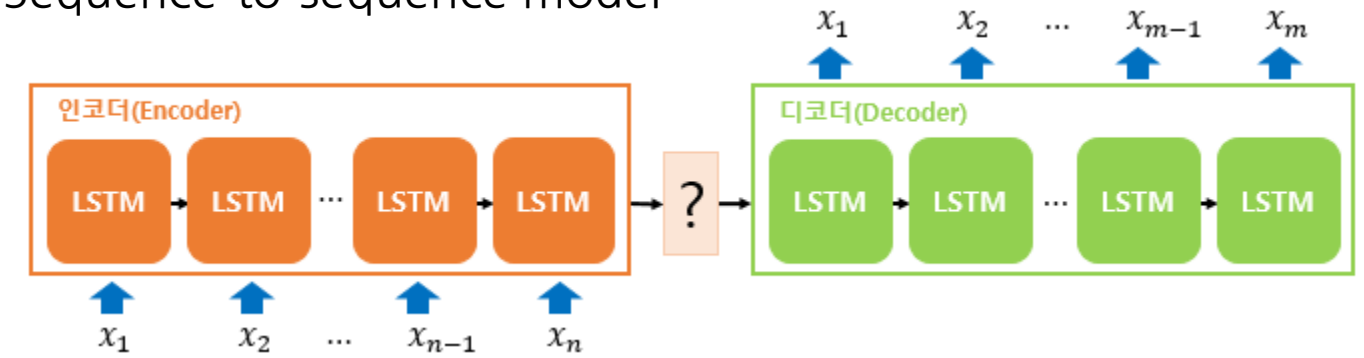
Output sequence is generated in an autoregressive manner

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

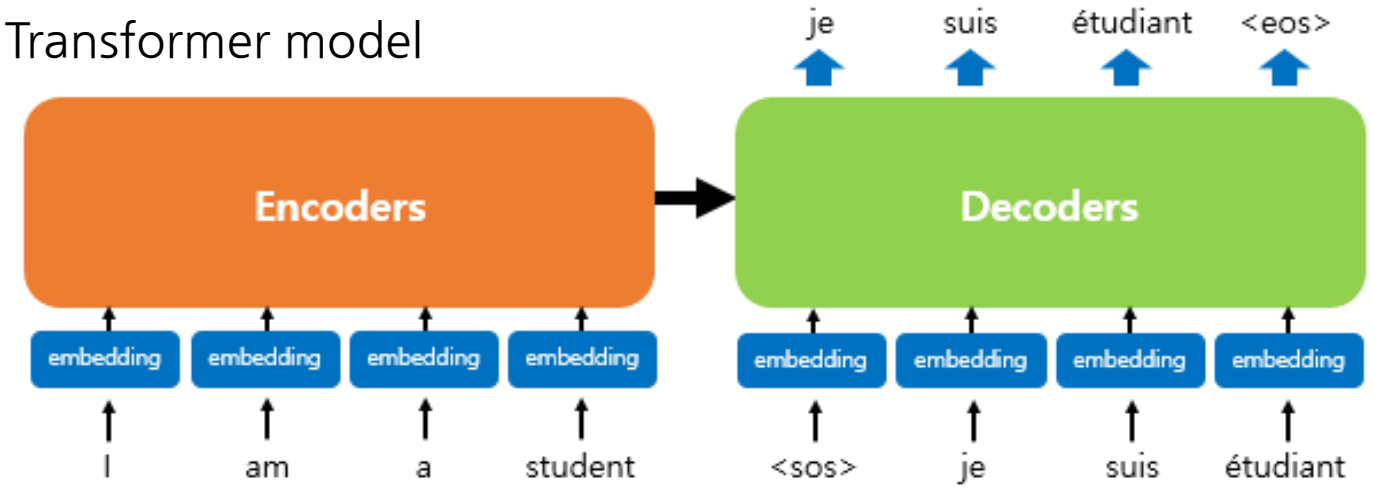
Seq2Seq vs. Transformer

ROBOT INTELLIGENCE LAB

Sequence-to-sequence model



Transformer model



ROBOT INTELLIGENCE LAB

ViT

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

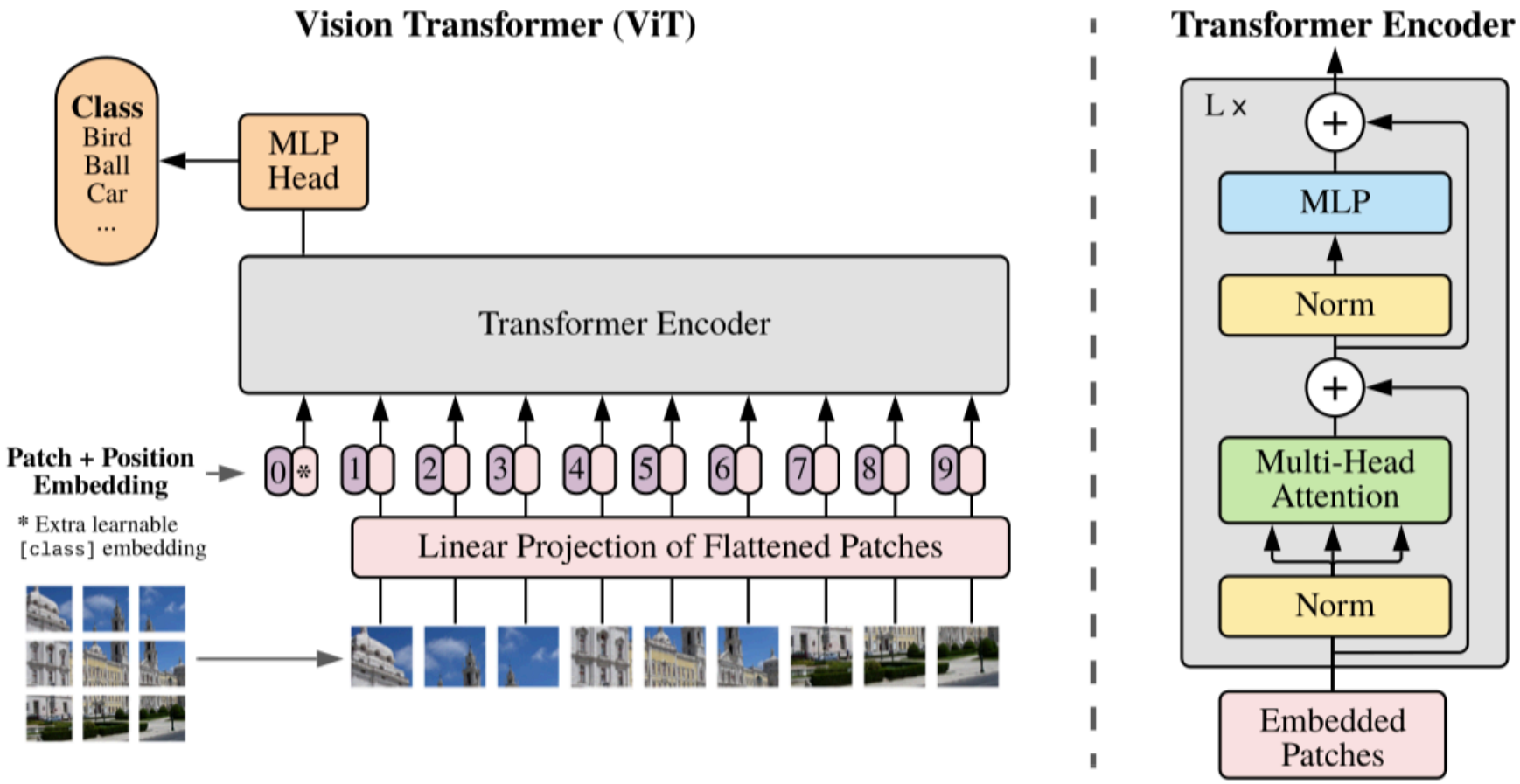
Vision Transformer (ViT)

ROBOT INTELLIGENCE LAB

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

ViT

ROBOT INTELLIGENCE LAB



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

ROBOT INTELLIGENCE LAB

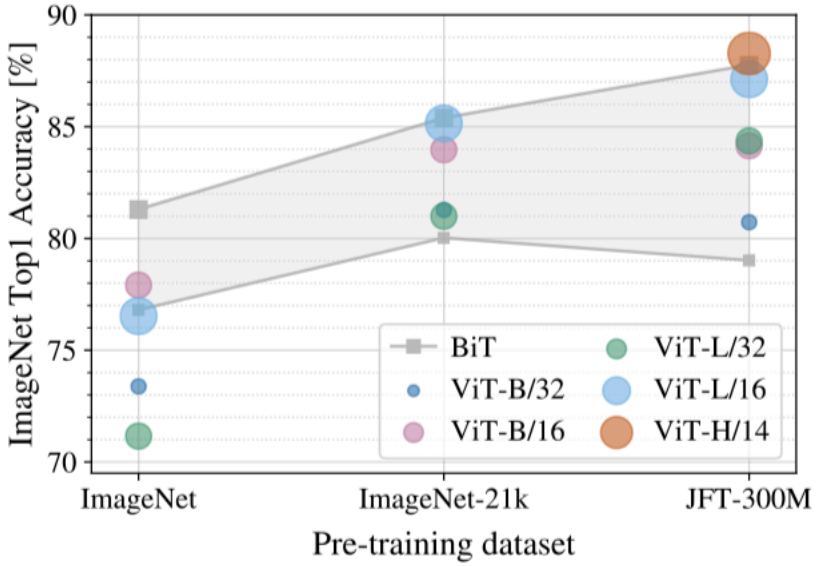


Figure 3: Transfer to ImageNet. While large ViT models perform worse than BiT ResNets (shaded area) when pre-trained on small datasets, they shine when pre-trained on larger datasets. Similarly, larger ViT variants overtake smaller ones as the dataset grows.

ROBOT INTELLIGENCE LAB

DINO

"Emerging Properties in Self-Supervised Vision Transformers," 2021

DINO

ROBOT INTELLIGENCE LAB

Self-DIstillation with NO labels (DINO)

DINO

ROBOT INTELLIGENCE LAB

- Does self-supervised learning provides new properties to Vision Transformer (ViT) compared to convolutional networks (convnets)?
    1. Self-supervised ViT features contain explicit information about the **semantic segmentation** of an image.
    2. These features are significantly useful for k**-NN classifiers** (i.e., 78.3% top-1 on ImageNet)

Semantic Segmentation

ROBOT INTELLIGENCE LAB

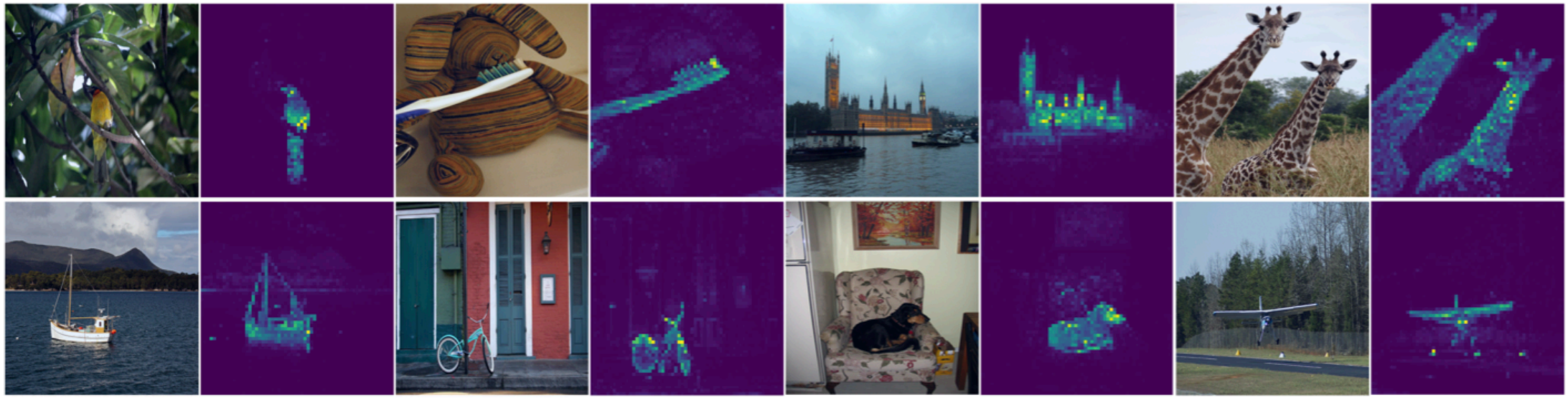


Figure 1: **Self-attention from a Vision Transformer with** 8×8 **patches trained with no supervision.** We look at the self-attention of the `[CLS]` token on the heads of the last layer. This token is not attached to any label nor supervision. These maps show that the model automatically learns class-specific features leading to unsupervised object segmentations.

Training Objectives

ROBOT INTELLIGENCE LAB

- DINO maintains two identical networks, teacher and student networks, where the SSL objective is to train the student network to predict the output of the teacher network.
  - The teacher network is updated via EMA of the student network (similar to MoCo).
  - The outputs of both student and teacher are passed through softmax and CE loss is used for prediction.
  - The output of teacher network is centered (batch-wise) before passing softmax.

DINO

ROBOT INTELLIGENCE LAB

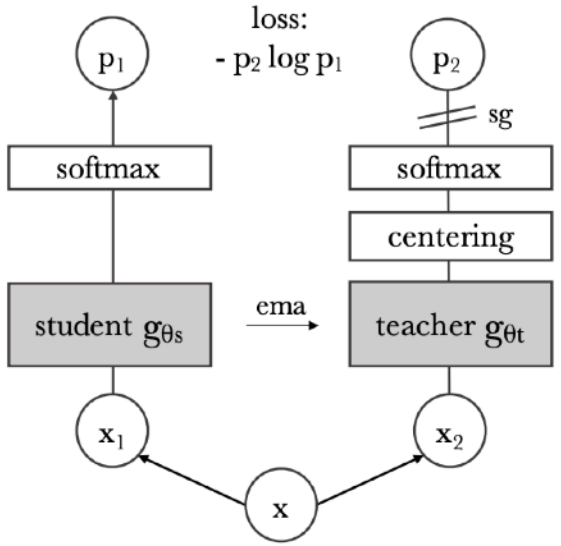


Figure 2: **Self-distillation with no labels.** We illustrate DINO in the case of one single pair of views (x_1, x_2) for simplicity. The model passes two different random transformations of an input image to the student and teacher networks. Both networks have the same architecture but different parameters. The output of the teacher network is centered with a mean computed over the batch. Each networks outputs a K dimensional feature that is normalized with a temperature softmax over the feature dimension. Their similarity is then measured with a cross-entropy loss. We apply a stop-gradient (sg) operator on the teacher to propagate gradients only through the student. The teacher parameters are updated with an exponential moving average (ema) of the student parameters.

DINO

ROBOT INTELLIGENCE LAB

Algorithm 1 DINO PyTorch pseudocode w/o multi-crop.

```
# gs, gt: student and teacher networks
# C: center (K)
# tps, tpt: student and teacher temperatures
# l, m: network and center momentum rates
gt.params = gs.params
for x in loader: # load a minibatch x with n samples
    x1, x2 = augment(x), augment(x) # random views

    s1, s2 = gs(x1), gs(x2) # student output n-by-K
    t1, t2 = gt(x1), gt(x2) # teacher output n-by-K

    loss = H(t1, s2)/2 + H(t2, s1)/2
    loss.backward() # back-propagate

    # student, teacher and center updates
    update(gs) # SGD
    gt.params = l*gt.params + (1-l)*gs.params
    C = m*C + (1-m)*cat([t1, t2]).mean(dim=0)

def H(t, s):
    t = t.detach() # stop gradient
    s = softmax(s / tps, dim=1)
    t = softmax((t - C) / tpt, dim=1) # center + sharpen
    return - (t * log(s)).sum(dim=1).mean()
```

Knowledge Distillation

ROBOT INTELLIGENCE LAB

- Knowledge distillation is a learning paradigm where we train a student network g_{θ_i}, parametrized by θ_s and θ_t respectively.
- Given an input image $\mathbf{x}$, both network output probability distribution over K dimension is obtained via a softmax function with temperature τ.

$$P(\mathbf{x})^{(i)} = \frac{\exp(g_\theta(\mathbf{x})^{(i)}/\tau)}{\sum_{k=1}^{K}\exp(g_\theta(\mathbf{x})^{(k)}/\tau)}$$

- Local-to-global Correspondences

$$\min_{\theta_s} \sum_{x\in\{x_1^g, x_2^g\}} \sum_{x'\in V, x'\neq x} H(P_t(x), P_s(x'))$$

  - where $\{x_1^g, x_2^g\}$ are two global view at resolution 224^2 covering a large ($> 50\,\%$) area of the original image and V is a set of serval local views of resolution 96^2 overing only small areas ($< 50\,\%$) of the original image.

Results

Table 2: **Linear and k-NN classification on ImageNet.** We report top-1 accuracy for linear and k-NN evaluations on the validation set of ImageNet for different self-supervised methods. We focus on ResNet-50 and ViT-small architectures, but also report the best results obtained across architectures. * are run by us. We run the k-NN evaluation for models with official released weights. The throughput (im/s) is calculated on a NVIDIA V100 GPU with 128 samples per forward. Parameters (M) are of the feature extractor.

Method	Arch.	Param.	im/s	Linear	k-NN
Supervised	RN50	23	1237	79.3	79.3
SCLR [12]	RN50	23	1237	69.1	60.7
MoCov2 [15]	RN50	23	1237	71.1	61.9
InfoMin [67]	RN50	23	1237	73.0	65.3
BarlowT [81]	RN50	23	1237	73.2	66.0
OBoW [27]	RN50	23	1237	73.8	61.9
BYOL [30]	RN50	23	1237	74.4	64.8
DCv2 [10]	RN50	23	1237	75.2	67.1
SwAV [10]	RN50	23	1237	**75.3**	65.7
DINO	RN50	23	1237	**75.3**	**67.5**
Supervised	ViT-S	21	1007	79.8	79.8
BYOL* [30]	ViT-S	21	1007	71.4	66.6
MoCov2* [15]	ViT-S	21	1007	72.7	64.4
SwAV* [10]	ViT-S	21	1007	73.5	66.3
DINO	ViT-S	21	1007	**77.0**	**74.5**
Comparison across architectures					
SCLR [12]	RN50w4	375	117	76.8	69.3
SwAV [10]	RN50w2	93	384	77.3	67.3
BYOL [30]	RN50w2	93	384	77.4	–
DINO	ViT-B/16	85	312	78.2	76.1
SwAV [10]	RN50w5	586	76	78.5	67.1
BYOL [30]	RN50w4	375	117	78.6	–
BYOL [30]	RN200w2	250	123	79.6	73.9
DINO	ViT-S/8	21	180	79.7	**78.3**
SCLRv2 [13]	RN152w3+SK	794	46	79.8	73.1
DINO	ViT-B/8	85	63	**80.1**	77.4

Results

ROBOT INTELLIGENCE LAB

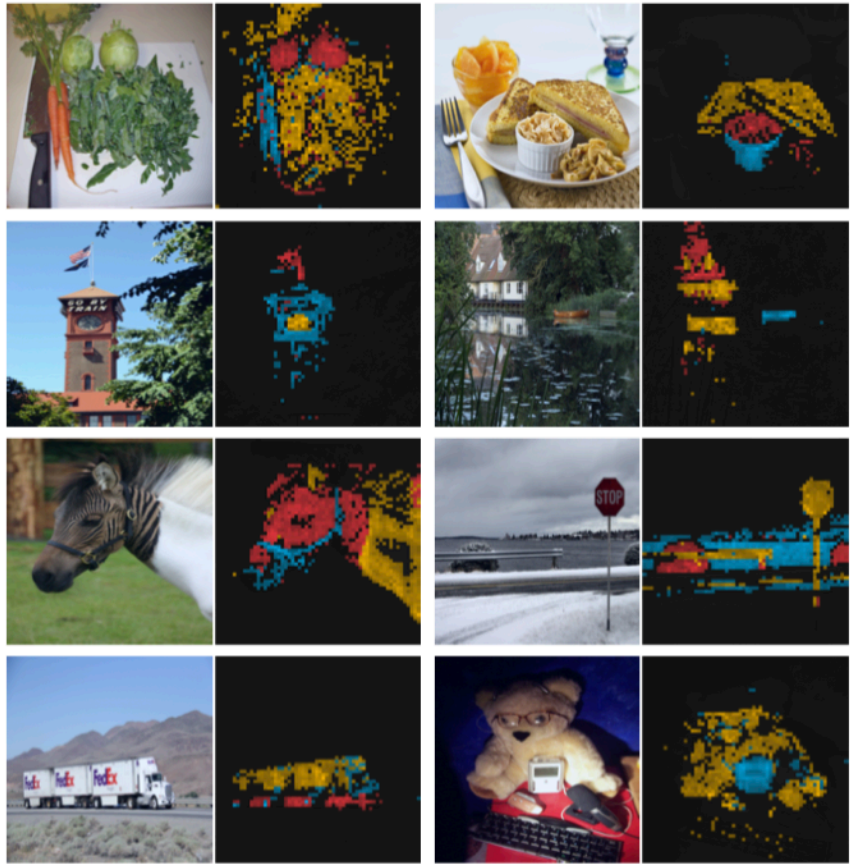
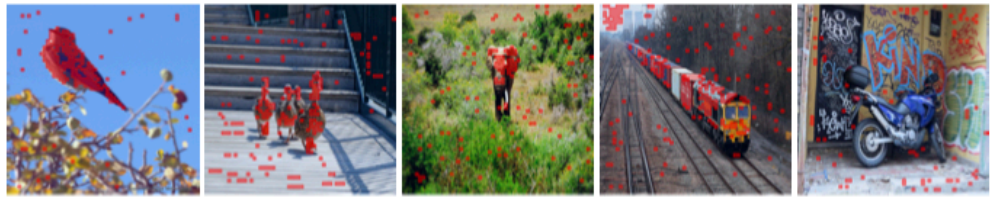


Figure 3: **Attention maps from multiple heads.** We consider the heads from the last layer of a ViT-S/8 trained with DINO and display the self-attention for `[CLS]` token query. Different heads, materialized by different colors, focus on different locations that represents different objects or parts (more examples in Appendix).

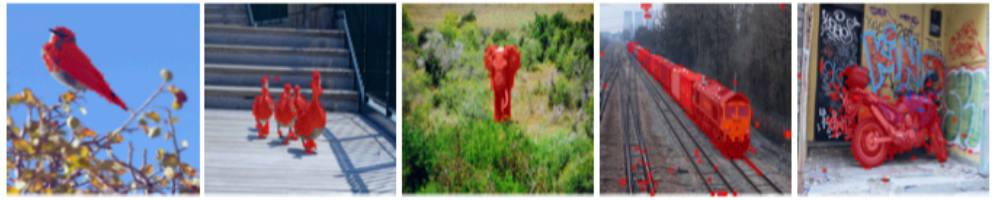
Results

ROBOT INTELLIGENCE LAB

Supervised



DINO



	Random	Supervised	DINO
ViT-S/16	22.0	27.3	45.9
ViT-S/8	21.8	23.7	44.7

Figure 4: **Segmentations from supervised versus DINO.** We visualize masks obtained by thresholding the self-attention maps to keep 60% of the mass. On top, we show the resulting masks for a ViT-S/8 trained with supervision and DINO. We show the best head for both models. The table at the bottom compares the Jaccard similarity between the ground truth and these masks on the validation images of PASCAL VOC12 dataset.

Ablation Study

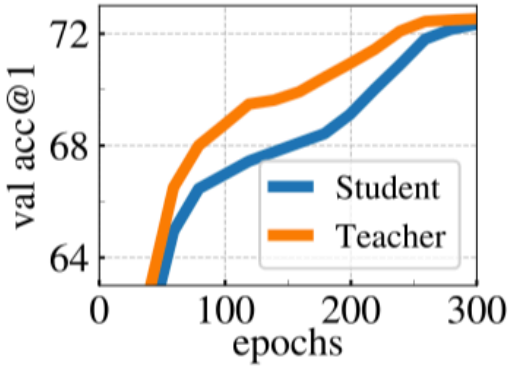
ROBOT INTELLIGENCE LAB

Table 7: **Important component for self-supervised ViT pre-training.** Models are trained for 300 epochs with ViT-S/16. We study the different components that matter for the k-NN and linear ("Lin.") evaluations. For the different variants, we highlight the differences from the default DINO setting. The best combination is the momentum encoder with the multicrop augmentation and the cross-entropy loss. We also report results with BYOL [30], MoCo-v2 [15] and SwAV [10].

	Method	Mom.	SK	MC	Loss	Pred.	k-NN	Lin.
1	DINO	✓	✗	✓	CE	✗	72.8	76.1
2		✗	✗	✓	CE	✗	0.1	0.1
3		✓	✓	✓	CE	✗	72.2	76.0
4		✓	✗	✗	CE	✗	67.9	72.5
5		✓	✗	✓	MSE	✗	52.6	62.4
6		✓	✗	✓	CE	✓	71.8	75.6
7	BYOL	✓	✗	✗	MSE	✓	66.6	71.4
8	MoCov2	✓	✗	✗	INCE	✗	62.0	71.6
9	SwAV	✗	✓	✓	CE	✗	64.7	71.8

SK: Sinkhorn-Knopp, MC: Multi-Crop, Pred.: Predictor
CE: Cross-Entropy, MSE: Mean Square Error, INCE: InfoNCE

Ablation Study



Teacher	Top-1
Student copy	0.1
Previous iter	0.1
Previous epoch	66.6
Momentum	72.8

Figure 6: Top-1 accuracy on ImageNet validation with k-NN classifier. **(left)** Comparison between the performance of the momentum teacher and the student during training. **(right)** Comparison between different types of teacher network. The momentum encoder leads to the best performance but is not the only viable option.

ROBOT INTELLIGENCE LAB

EsViT

"Efficient Self-supervised Vision Transformers for Representation Learning," 2021

EsViT

Efficient self-supervised Vision Transformers (EsViT)

EsViT

ROBOT INTELLIGENCE LAB

- EsVit combines multi-stage architectures for Transformer and region matching pre-training.
  - **Multi-stage architecture** with sparse self-attentions can reduce modeling complexity but with a cost of losing the ability to capture fine-grained correspondences between image regions.
  - A new pre-training task of **region matching** is proposed which allows the model to capture fine-grained region dependencies.
- EsVit achieves 81.3% top-1 on the ImageNet linear probe evaluation.

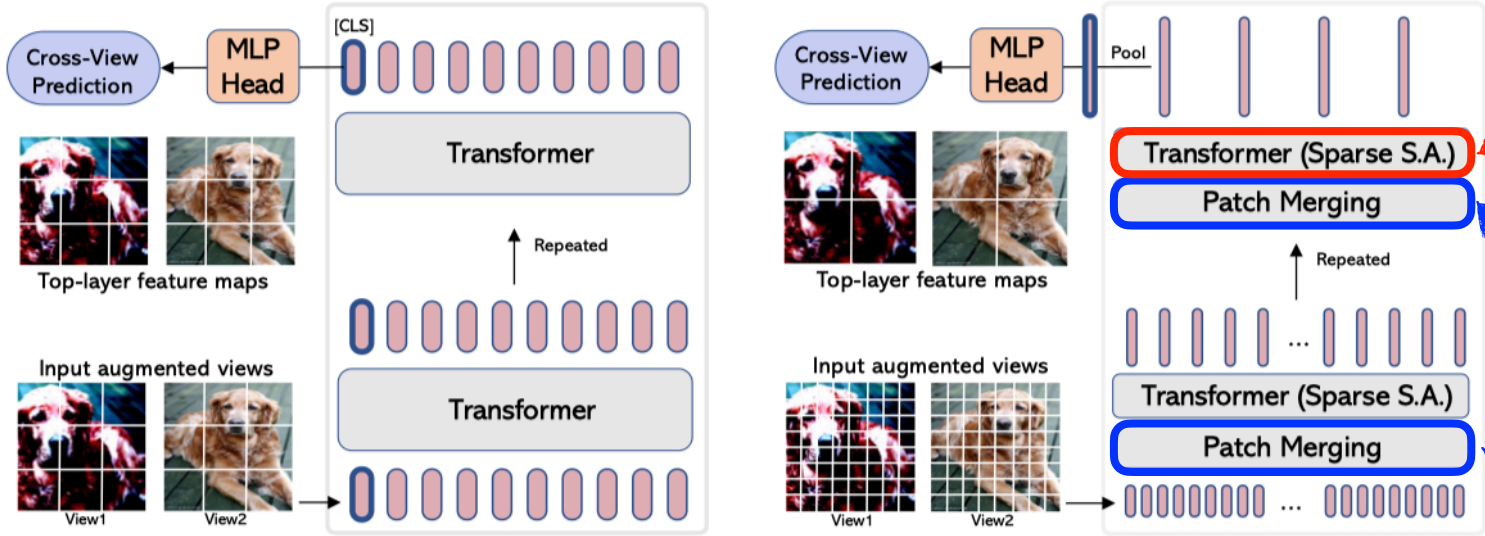
EsViT

ROBOT INTELLIGENCE LAB

EsVit = Multi-stage Swin Transformer + DINO + Region Matching

Architecture

ROBOT INTELLIGENCE LAB



(a) Baseline monolithic architecture (b) Proposed multi-stage architecture

Figure 2: Architecture comparison. (a) The traditional monolithic transformer architecture. For all layers, the transformer blocks share the same network configurations and input token sequence sizes are the same. (b) The multi-stage Transformer architecture organizes an input image into a long sequence of smaller patches, sparse self-attentions (S.A.) are utilized at early stages to maintain model expressiveness while reducing computational complexity; The neighboring tokens at an intermediate layer are gradually merged, constituting a short sequence to ease the compute burden of self-attention at late stages.

Sparse self-attention splits the feature maps into non-overlapping local windows, and self-attention is performed within each local window.

It concatenates the features of each group of 2×2 neighboring patches, and applies linear projection on $4C$-dim concatenated feature to $2C$.

It splits an input RGB image into non-overlapping patches (tokens).

Swin Transformer

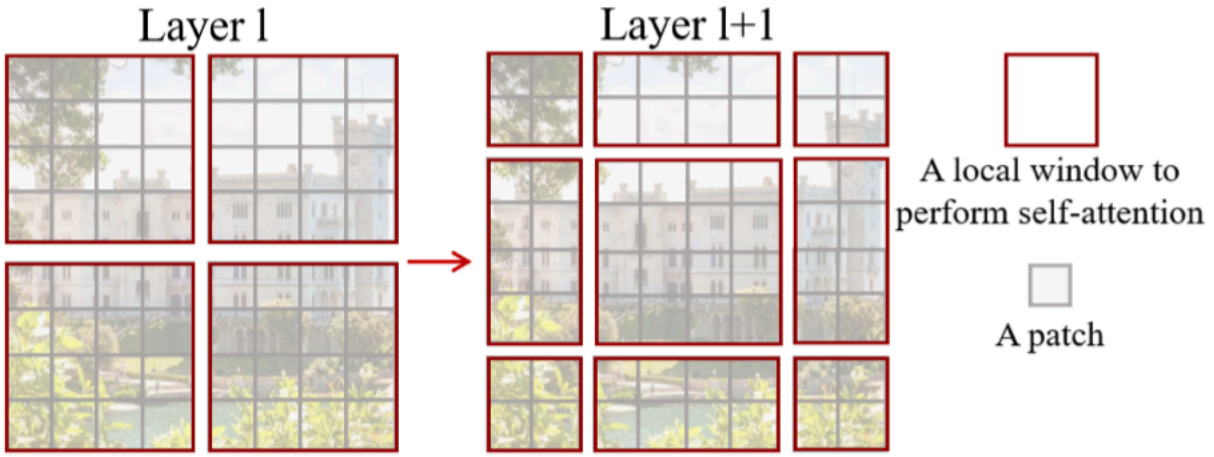
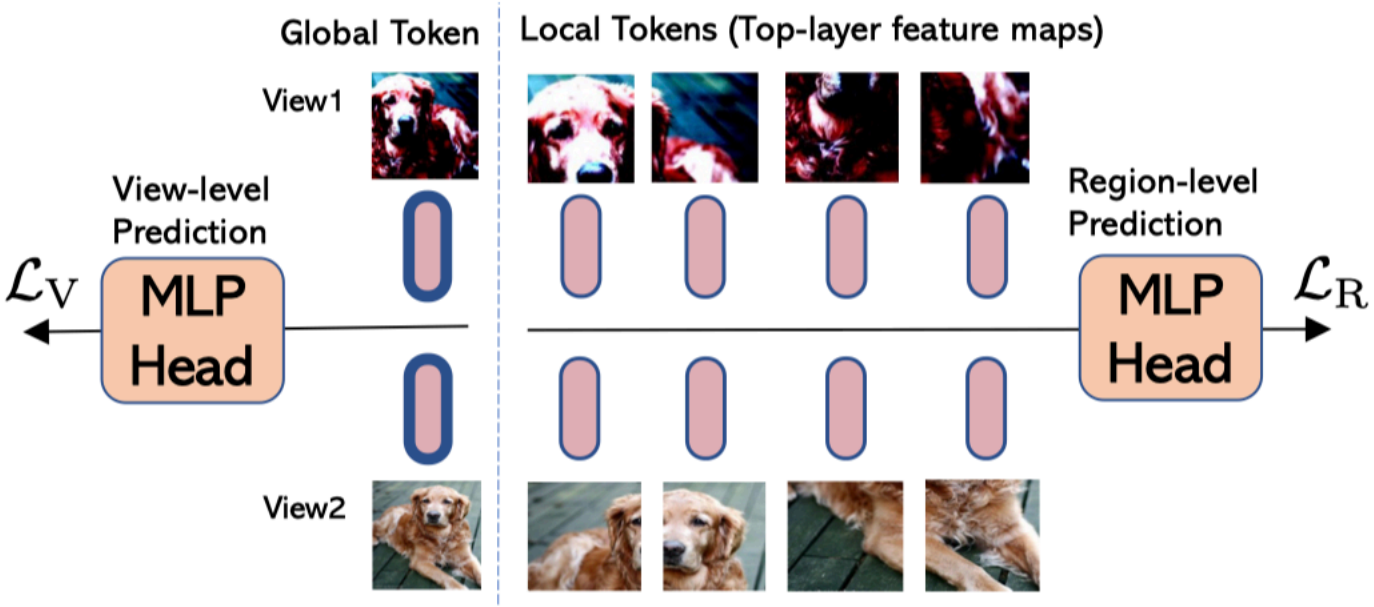


Figure 2. An illustration of the *shifted window* approach for computing self-attention in the proposed Swin Transformer architecture. In layer l (left), a regular window partitioning scheme is adopted, and self-attention is computed within each window. In the next layer $l + 1$ (right), the window partitioning is shifted, resulting in new windows. The self-attention computation in the new windows crosses the boundaries of the previous windows in layer l, providing connections among them.

Region Matching



- Co-occurrences/structures between local features

$$\mathscr{L}_R = \frac{1}{|\mathscr{P}|} \sum_{(s,t)\in\mathscr{P}} \mathscr{M}_R(s,t) \text{ with } \mathscr{M}_R(s,t) = -\frac{1}{T}\sum_{i=1}^{T} p_{j^*} \log p_i \text{ and } j^* = \arg\max_j \frac{z_i^T z_j}{\|z_i\|\|z_j\|}$$

where $p_i = h'(z_i)$ and $p_j = h'(z_j)$ are the probability outputs of a new MLP head h' over the local feature of student $z_i \in \mathbf{z}_s$ and teacher $z_j \in \mathbf{z}_t$, respectively.

Results

ROBOT INTELLIGENCE LAB

Method	#Parameters ↓	Throughput (Image/s) ↑	Linear ↑	k-NN ↑
SoTA SSL methods with Big ConvNets				
SwAV, RN50w5 [5]	586	76	78.5	67.1
BYOL, RN200w2 [25]	250	123	79.6	73.9
SimCLR-v2, RN152w3+SK [10]	794	46	79.8	73.1
Skyline methods with excessively long sequences for self-attentions				
DINO, DeiT-S/8 [6]	21	180	79.7	78.3
DINO, ViT-B/8 [6]	85	63	80.1	77.4
MoCo-v3, ViT-B-BN/7 [12]	85	∼63	79.5	-
MoCo-v3, ViT-L-BN/7 [12]	304	∼17	81.0	-
iGPT, iGPT-XL [8]	6801	-	72.0	-
EsViT, Swin-S/$W=14$	49	383	80.8	79.1
EsViT, Swin-B/$W=14$	87	254	**81.3**	**79.3**
Transformer-based SSL, with moderate sequence length for self-attentions				
Masked Patch Pred., ViT-B/16 [19]	85	312	79.9†	-
DINO, DeiT-S/16 [6]	21	1007	77.0	74.5
DINO, ViT-B/16 [6]	85	312	78.2	76.1
MoCo-v3, ViT-B/16 [12]	85	312	76.7	-
MoCo-v3, ViT-H-BN/16 [12]	632	∼32	79.1	-
MoBY, Swin-T [64]	28	808	75.1	-
EsViT, Swin-T	28	808	78.1	75.7
EsViT, Swin-S	49	467	79.5	77.7
EsViT, Swin-B	87	297	**80.4**	**78.9**

Table 1: Comparison with SoTA across different architectures on ImageNet linear probing. ViT-BN is ViT that has BatchNorm [21], and "/P" denotes a patch size of $P \times P$. "∼" indicates through-puts estimated by comparing different papers, detailed in Appendix. † The mask patch prediction in [19] is pre-trained on JFT-300M and end-to-end fine-tuned in ImageNet, which we append as a reference.

Results

ROBOT INTELLIGENCE LAB

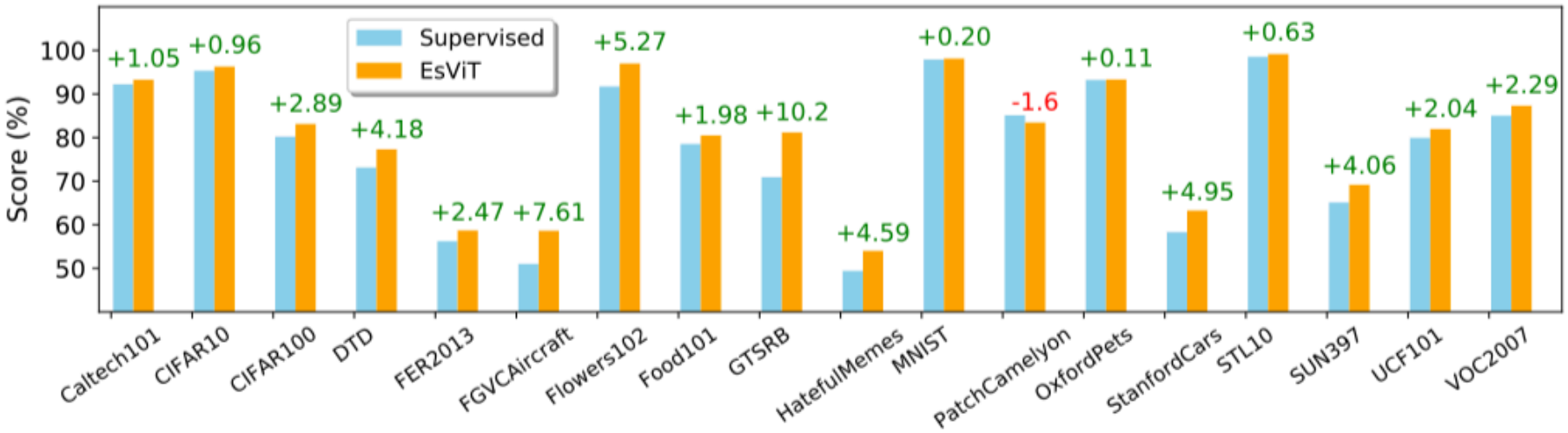


Figure 5: Transfer learning to a wide variety of datasets. Fitting a linear classifier on EsViT's features outperforms using its supervised counterpart on 17 out of 18 datasets.

Thank You