Transformers for Language Processing and Vision

IE 643

Lecture 23

Nov 1, 2024.

Transformers

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com

Noam Shazeer* Google Brain noam@google.com

Niki Parmar* Google Research nikip@google.com

Jakob Uszkoreit* Google Research usz@google.com

Llion Jones* Google Research llion@google.com

Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu

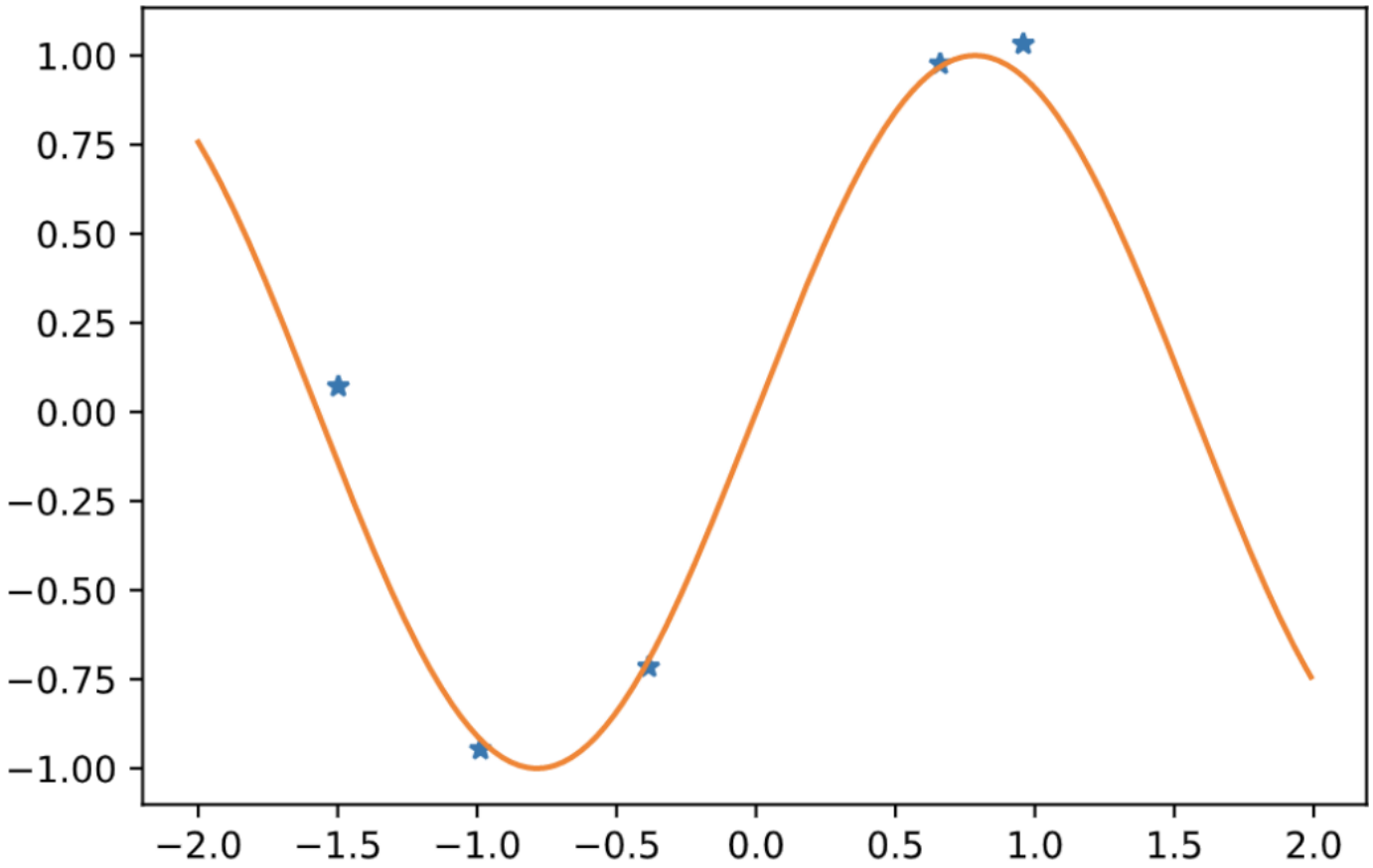
Łukasz Kaiser* Google Brain lukaszkaiser@google.com

Illia Polosukhin* ‡ illia.polosukhin@gmail.com

Transformers - Idea from history

Watson-Nadaraya Estimator

- Given data set $\{(x^i, y^i)\}_{i=1}^n$, with $x^i \in \mathbb{R}$, $y^i \in \mathbb{R}$, determine label y for a sample x.

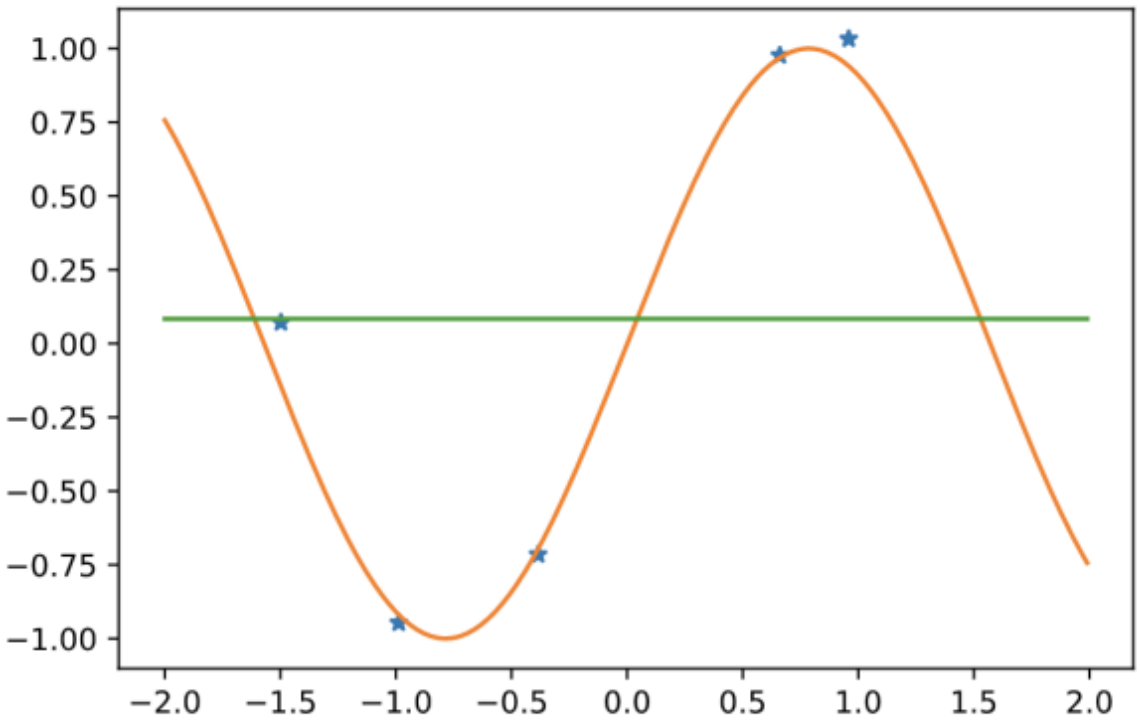


Source: http://alex.smola.org/talks/ICML19-attention.pdf

Transformers - Idea from history

Watson-Nadaraya Estimator

- Given data set $\{(x^i, y^i)\}_{i=1}^n$, with $x^i \in \mathbb{R}$, $y^i \in \mathbb{R}$, determine label y for a sample x.
- Naive estimator: $y = \frac{1}{m} \sum_{i=1}^n y^i$

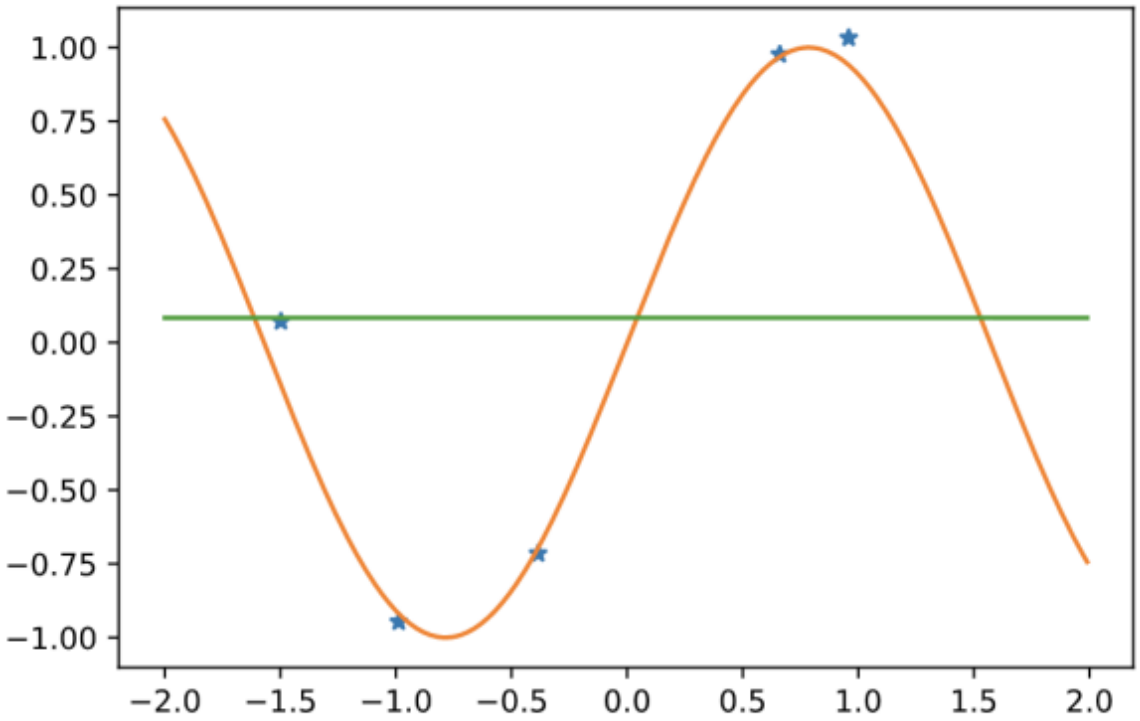


Source: http://alex.smola.org/talks/ICML19-attention.pdf

Transformers - Idea from history

Watson-Nadaraya Estimator

- Given data set $\{(x^i, y^i)\}_{i=1}^n$, with $x^i \in \mathbb{R}$, $y^i \in \mathbb{R}$, determine label y for a sample x.
- Naive estimator: $y = \frac{1}{m} \sum_{i=1}^n y^i$
- Improved estimator: $y = \sum_{i=1}^n \alpha(x^i, x) y^i$

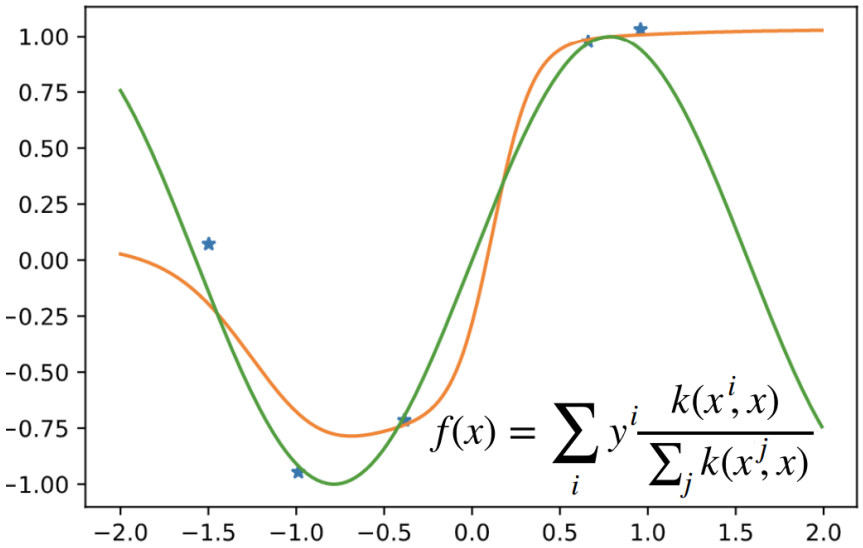


Source: http://alex.smola.org/talks/ICML19-attention.pdf

Transformers - Idea from history

Watson-Nadaraya Estimator

- Given data set $\{(x^i, y^i)\}_{i=1}^n$, with $x^i \in \mathbb{R}$, $y^i \in \mathbb{R}$, determine label y for a sample x.
- Naive estimator: $y = \frac{1}{m}\sum_{i=1}^n y^i$
- Improved estimator: $y = \sum_{i=1}^n \alpha(x^i, x) y^i$
- Weights $\alpha(x^i, x)$ can be considered to be attention score of i-th sample.

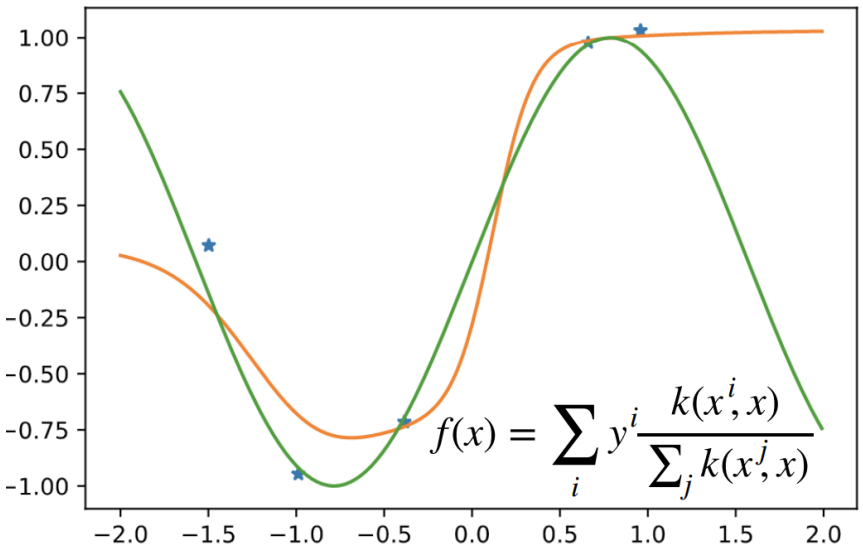


Source: http://alex.smola.org/talks/ICML19-attention.pdf

Transformers - Idea from history

Watson-Nadaraya Estimator

- Given data set $\{(x^i, y^i)\}_{i=1}^n$, with $x^i \in \mathbb{R}$, $y^i \in \mathbb{R}$, determine label y for a sample x.
- Naive estimator: $y = \frac{1}{m} \sum_{i=1}^n y^i$
- Improved estimator: $y = \sum_{i=1}^n \alpha(x^i, x) y^i$
- Weights $\alpha(x^i, x)$ can be considered to be attention score of i-th sample.
- Popular example: $\alpha(x^i, x) = k(x^i, x) / \sum_{j=1}^n k(x^j, x)$ where $k(x^i, x)$ is some kernel function based score for i-th sample.



Transformers - Idea from history

Watson-Nadaraya Estimator

- In the expression $y = \sum_{i=1}^{n} \alpha(x^i, x) y^i$,
  - x can be called **query**
  - x^i can be called **key**
  - y^i can be called **value**

Transformers - Architecture

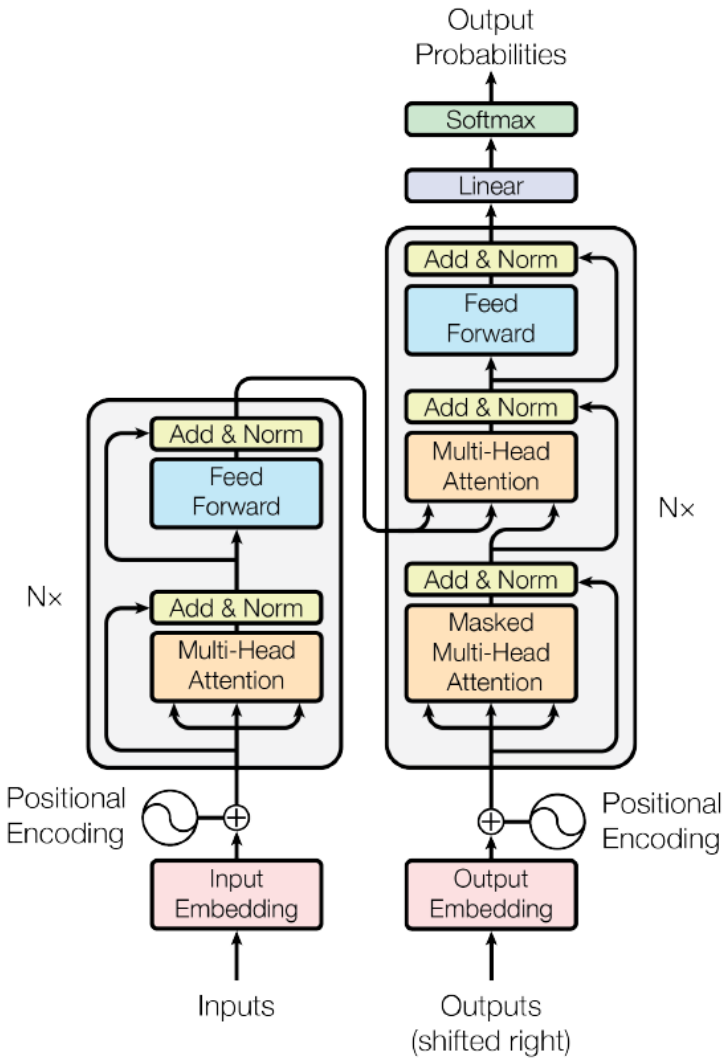
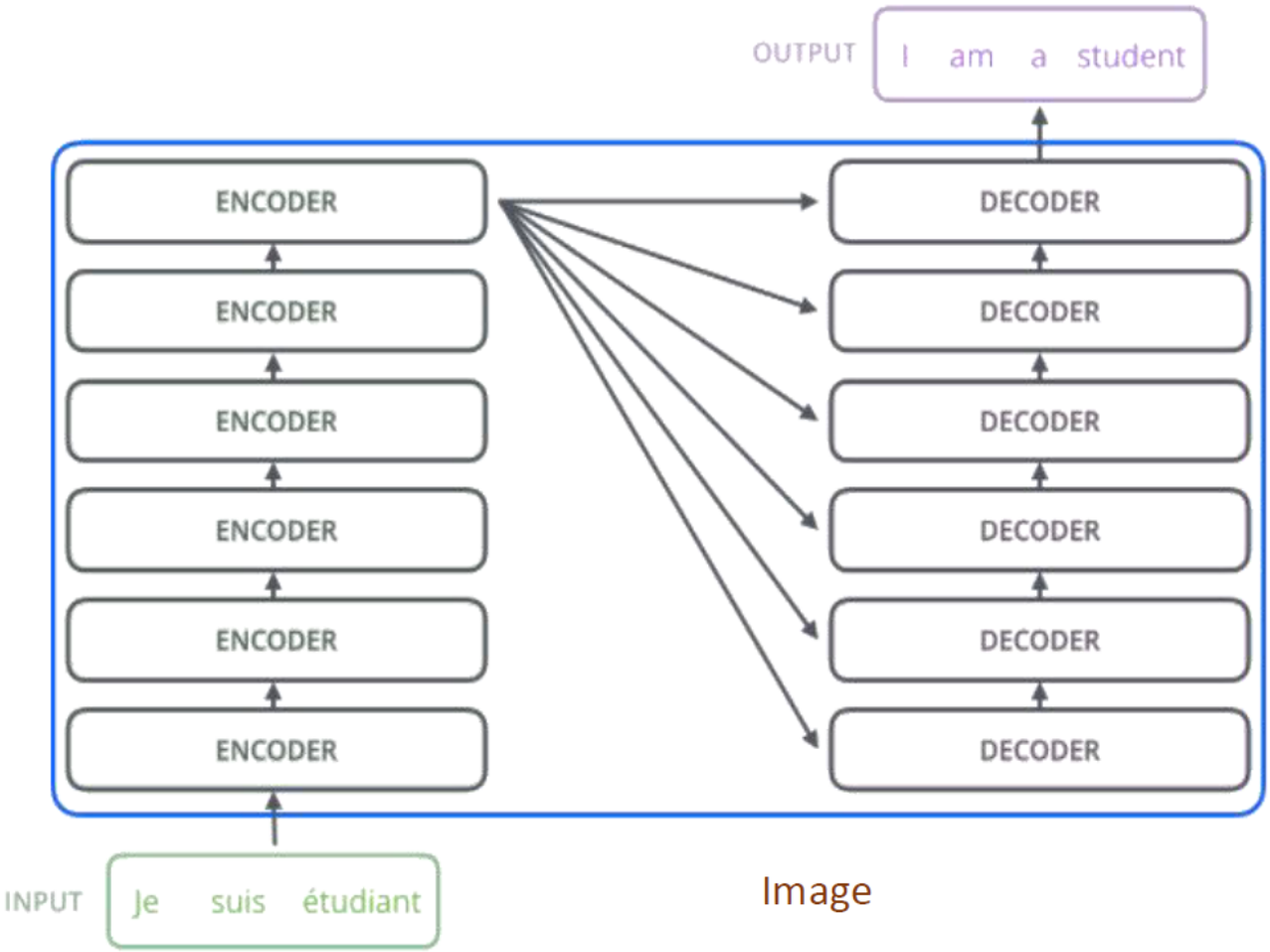


Figure : The Transformer - model architecture.

Transformers - Blocks in Architecture



Image

Image Source: The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time. (jalammar.github.io)

Transformers - Word vector representations

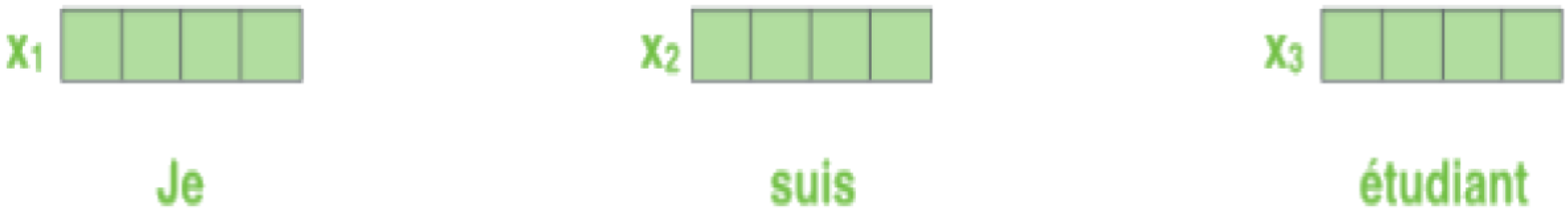


Image Source: The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time. (jalammar.github.io)

Transformers - Positional Encoding

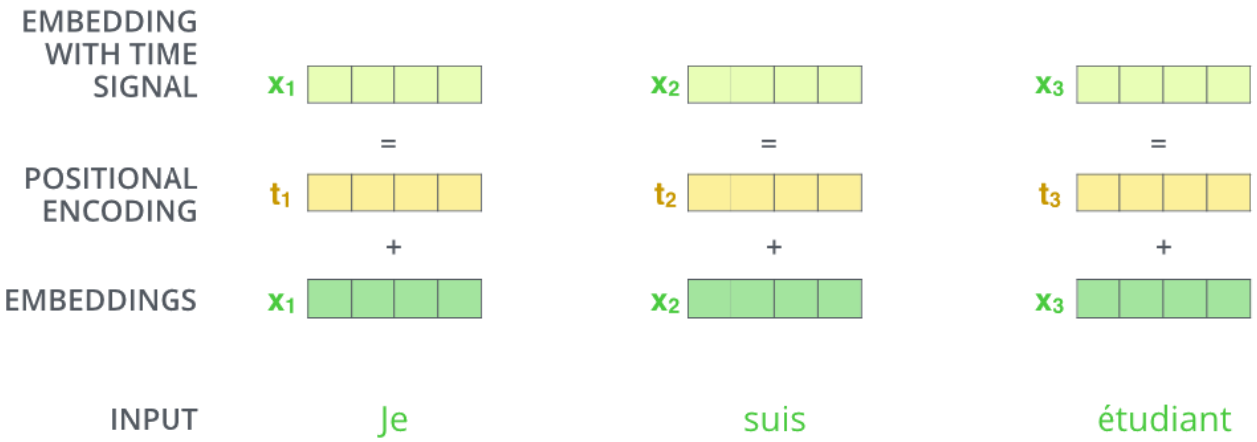


Image Source: The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time. (jalammar.github.io)

Transformers

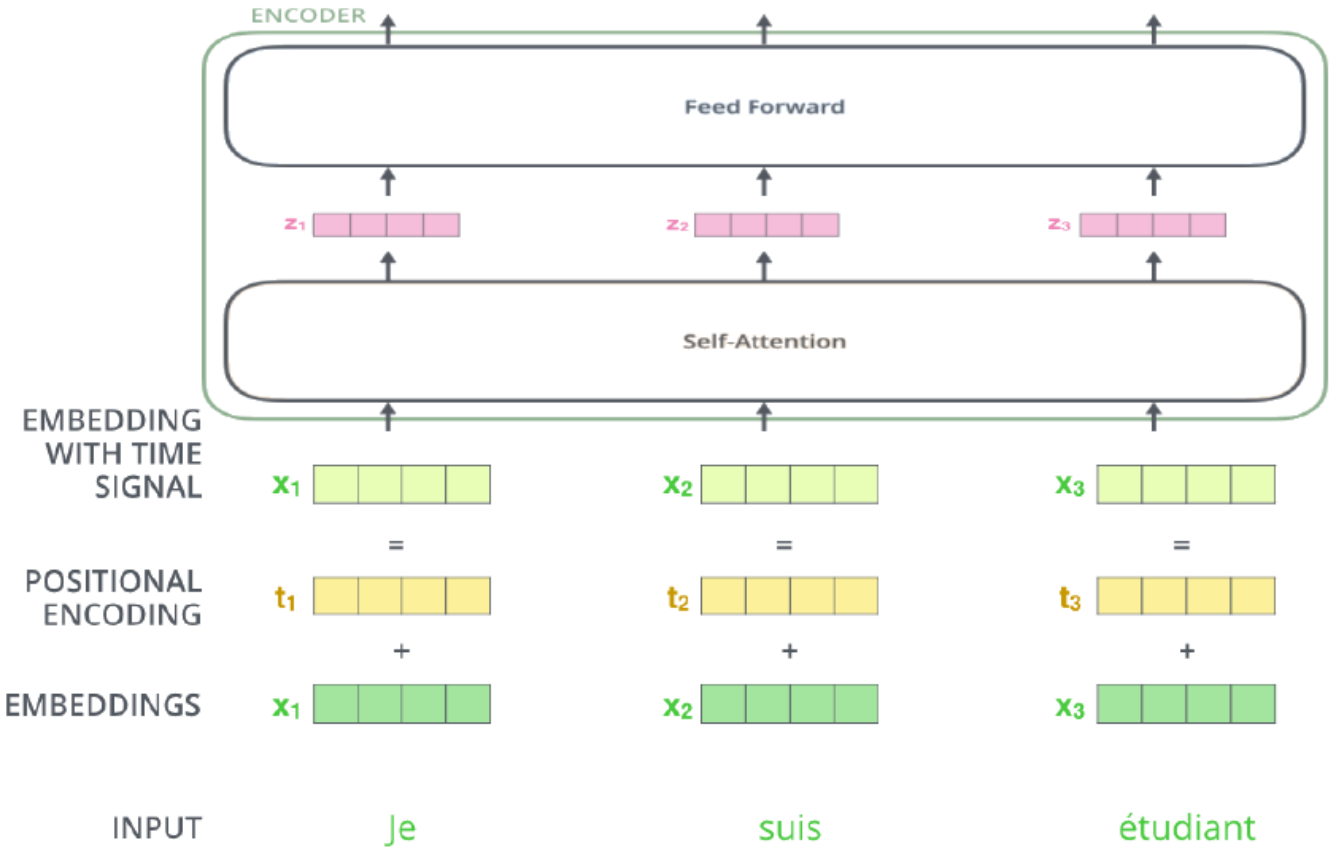


Image Source: The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time. (jalammar.github.io)

Transformers - Action of First Encoder

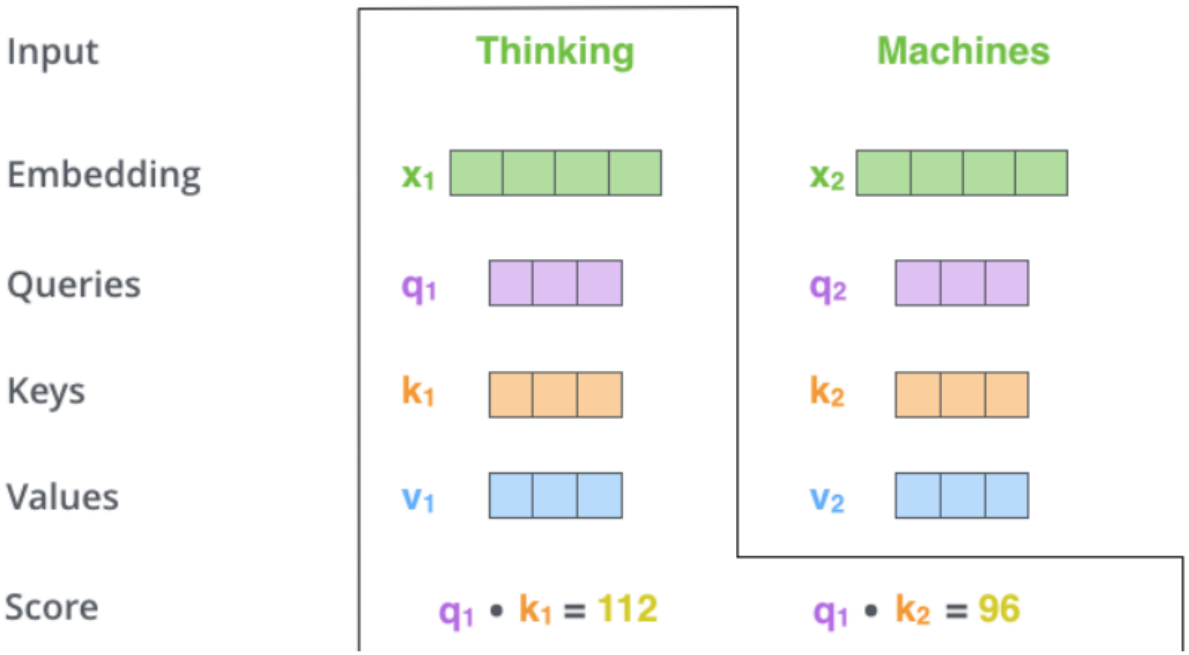


Image Source: The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time. (jalammar.github.io)

Transformers: Self-attention

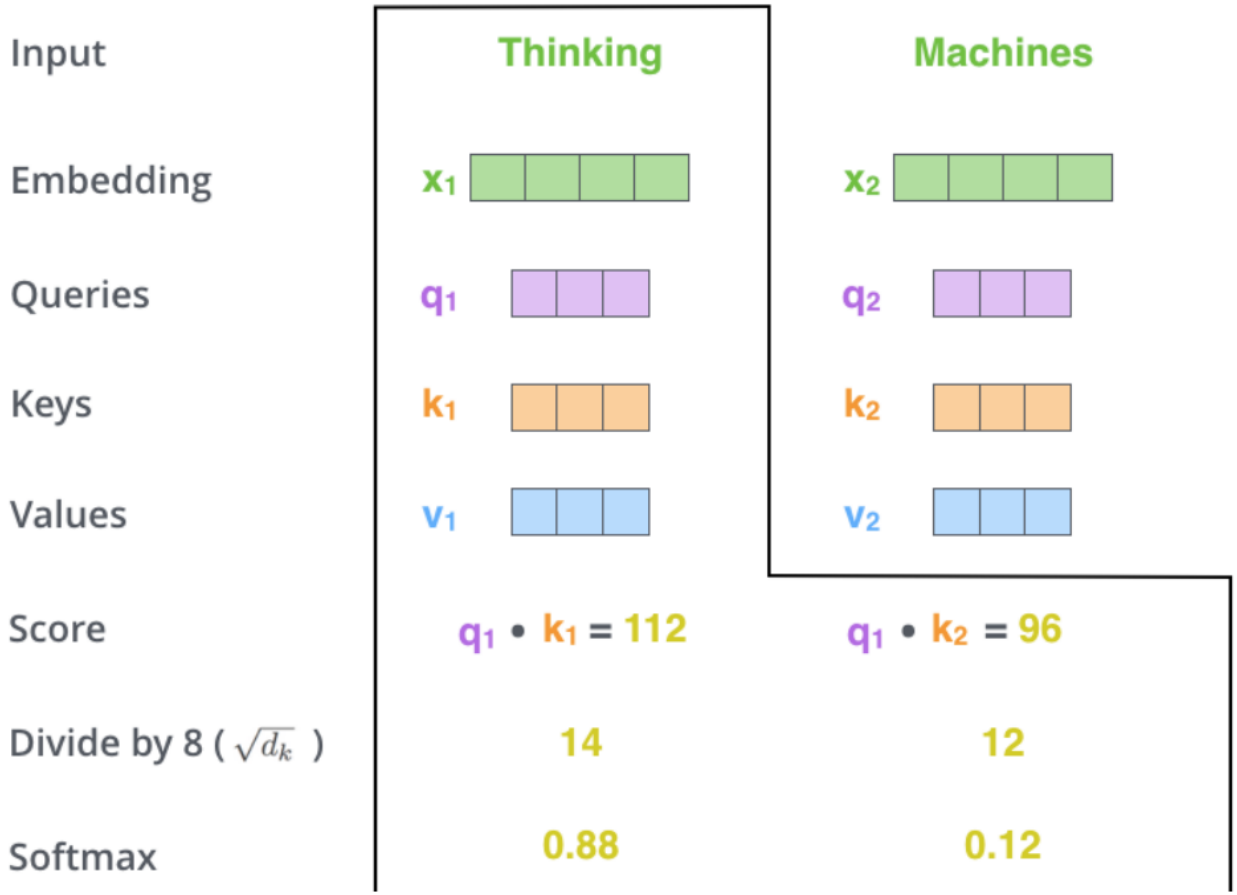
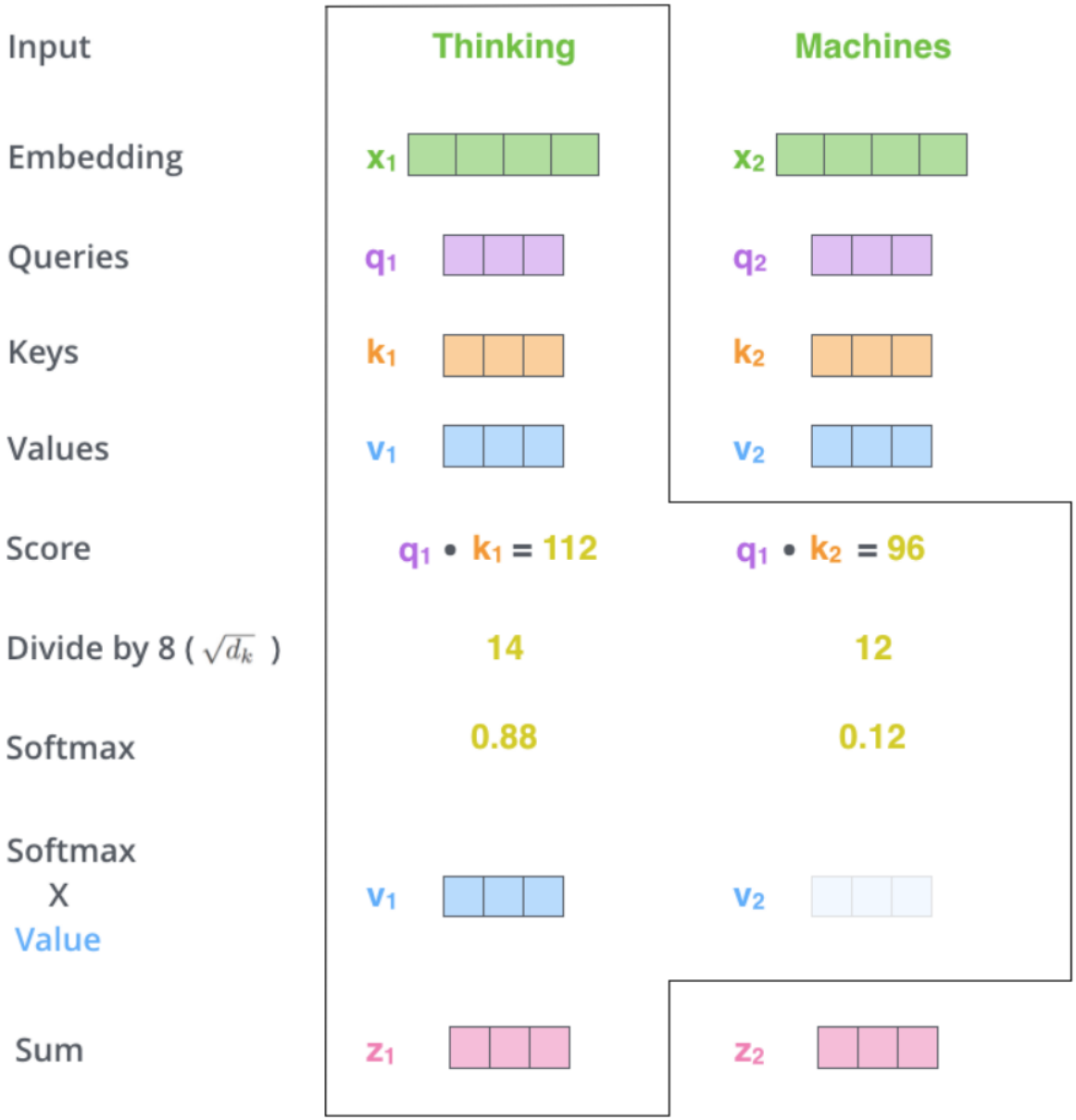


Image Source: The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time. (jalammar.github.io)

Transformers: Self-attention



Transformers: Self-attention

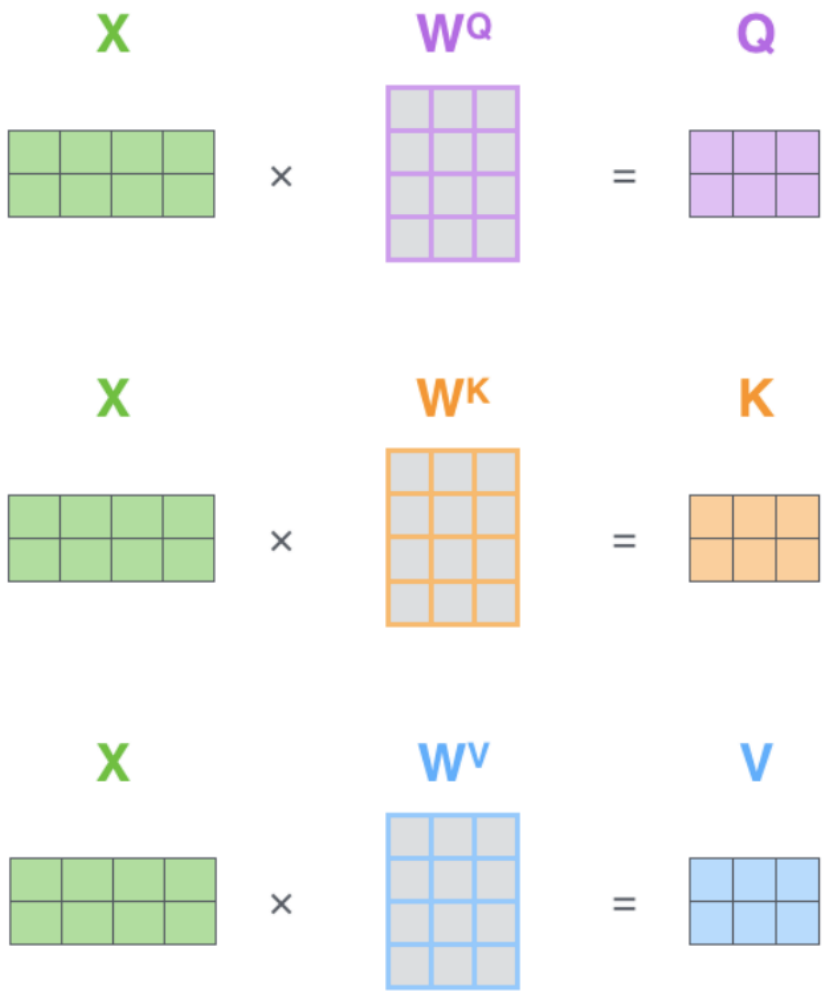


Image Source: The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time. (jalammar.github.io)

Transformers: Self-attention

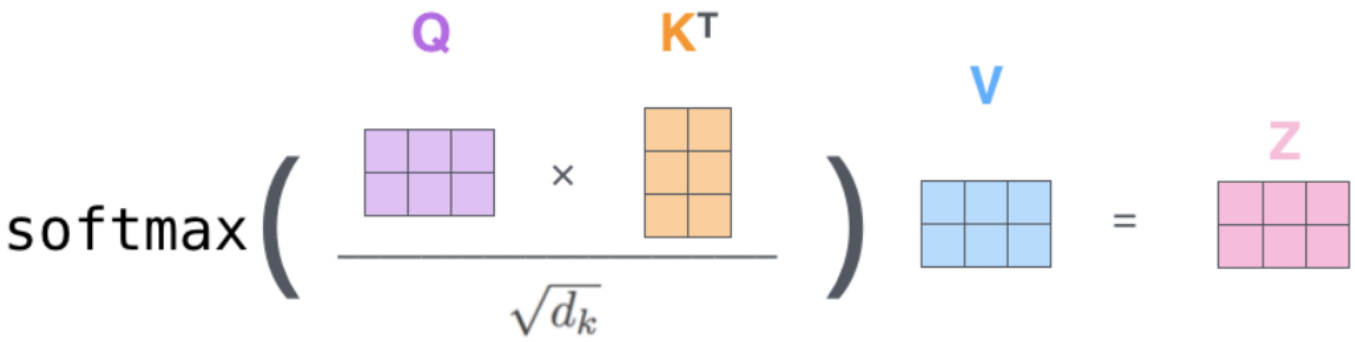
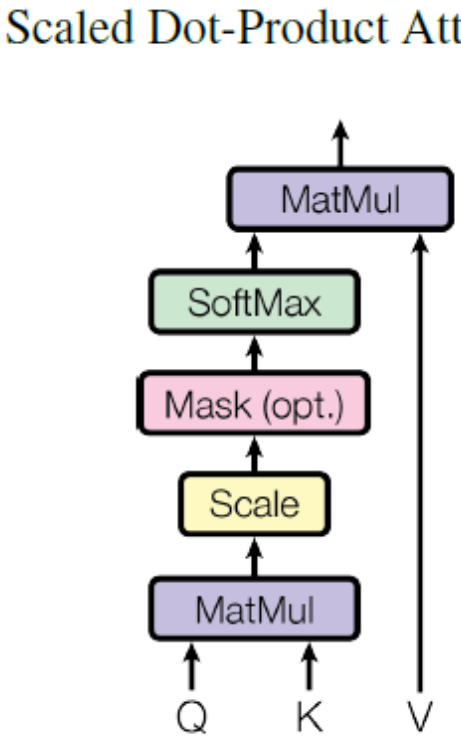


Image Source: The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time. (jalammar.github.io)

Transformers: Self-attention



$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

Transformers: Multi-head attention

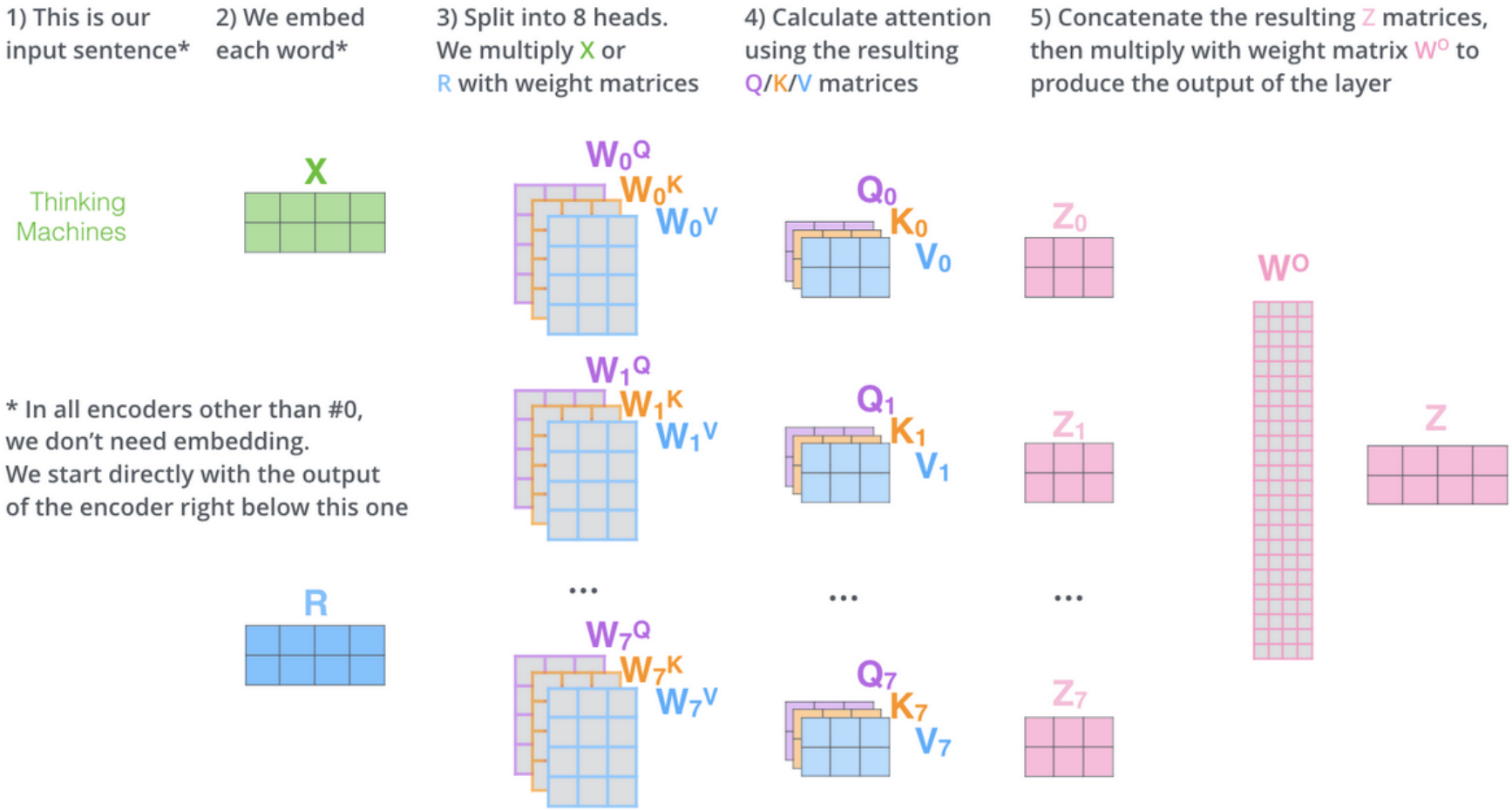
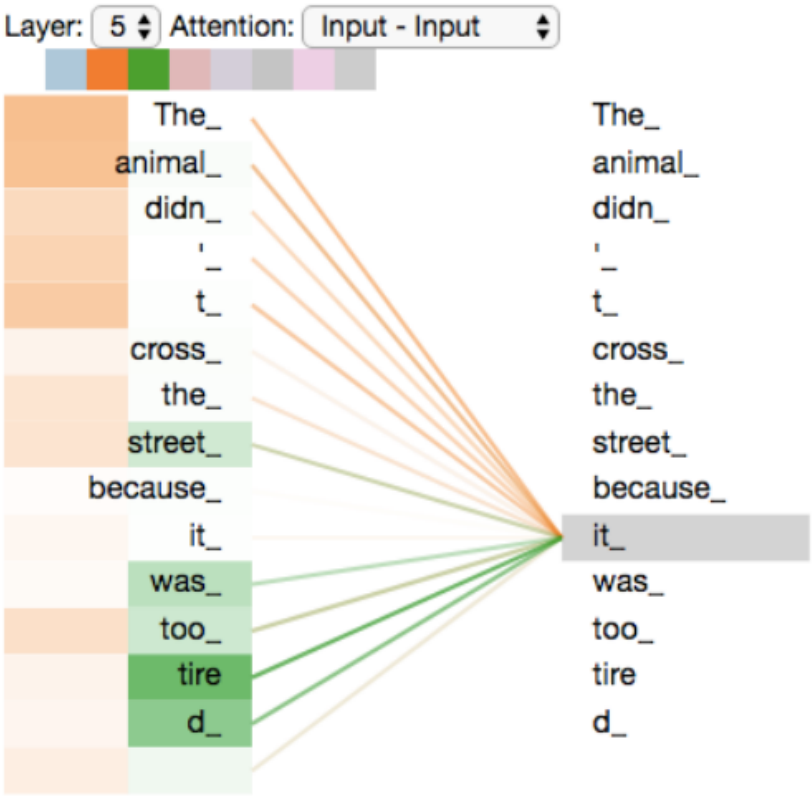
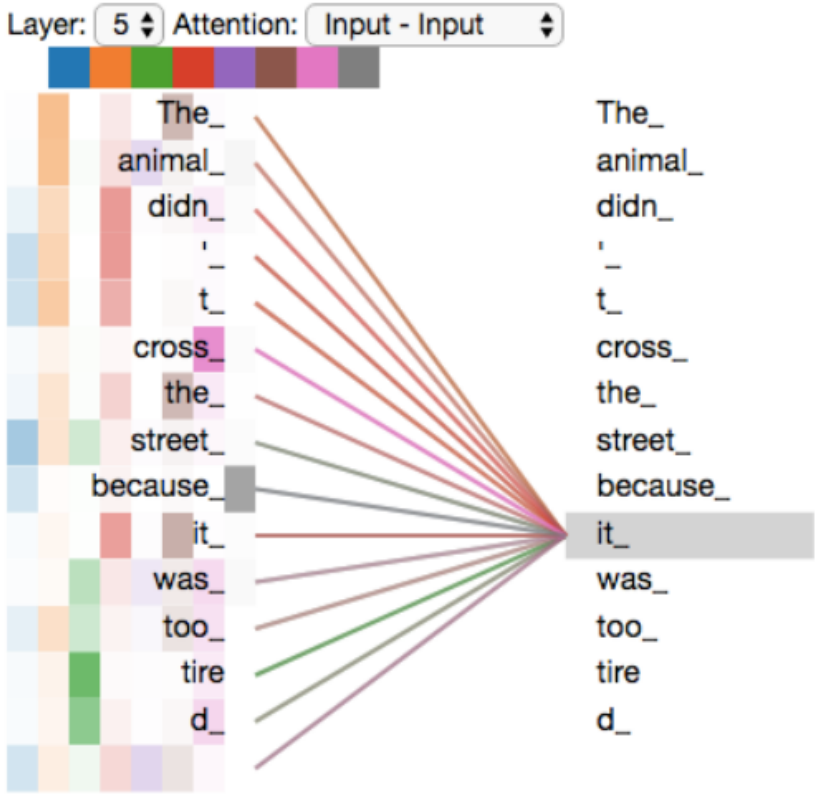


Image Source: The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time. (jalammar.github.io)

Transformers: Multi-head self-attention Example





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

Transformers: Full Encoder Block

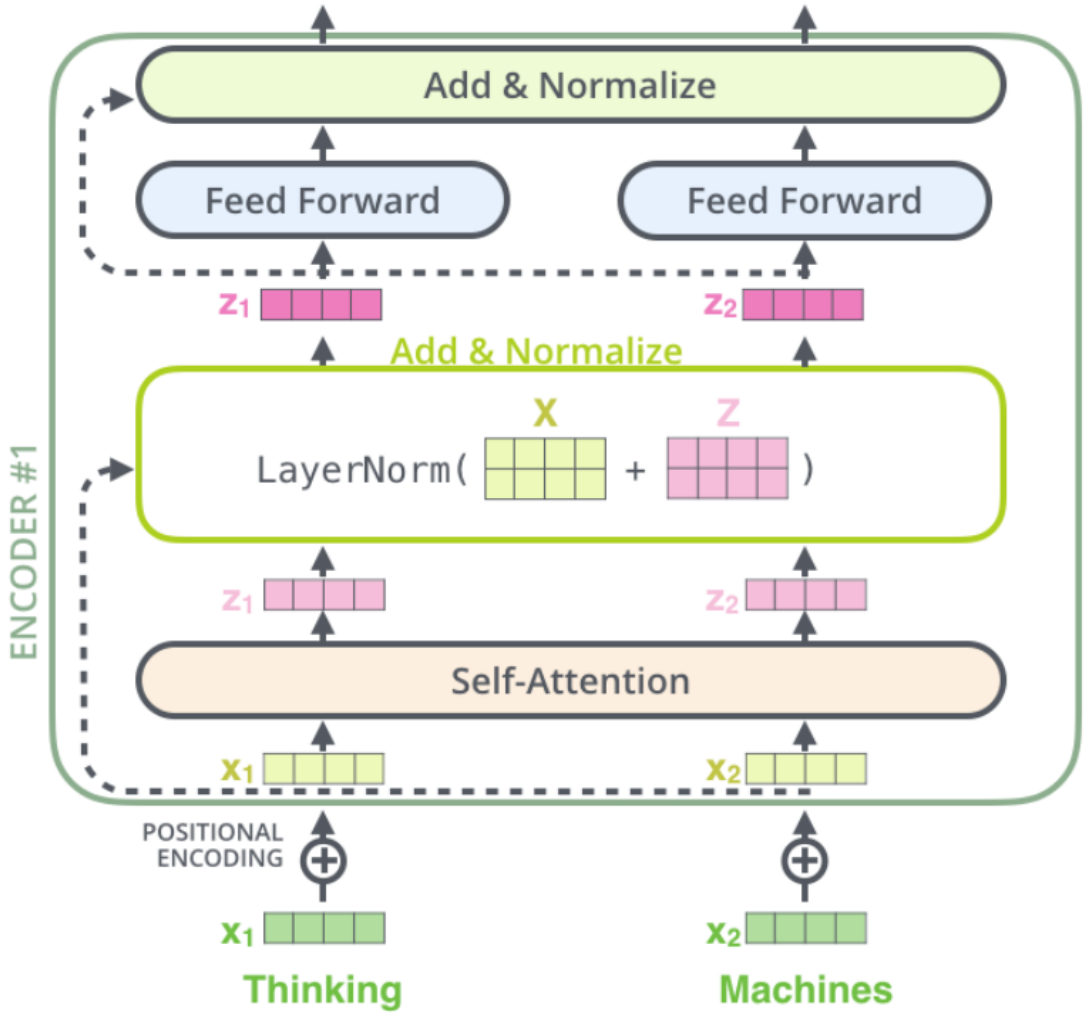


Image Source: The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time. (jalammar.github.io)

Transformers: Full Encoder Block

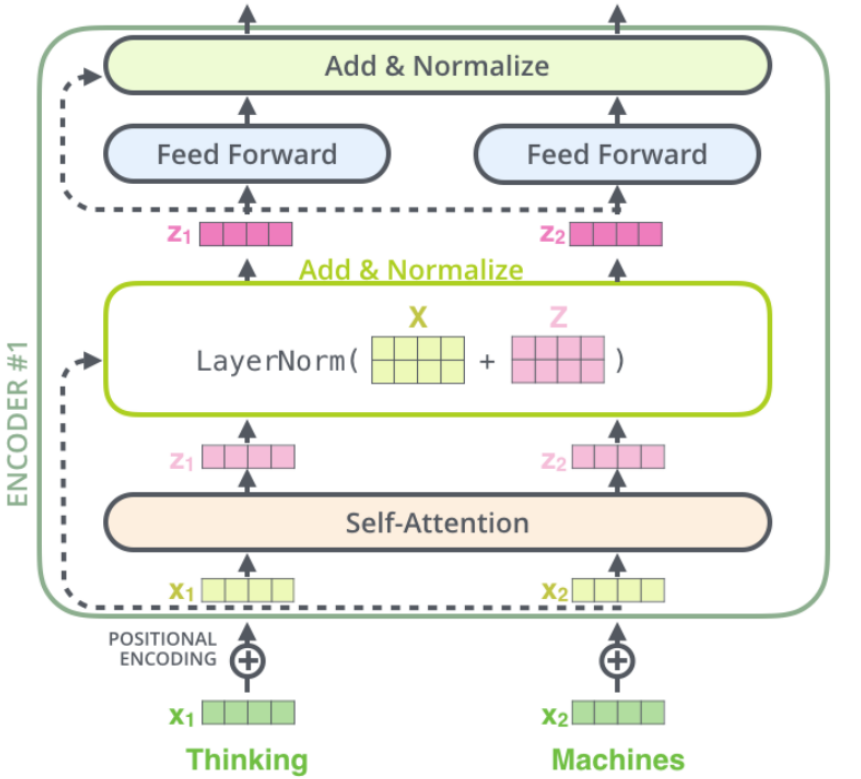


Image Source: The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time. (jalammar.github.io)

Layer Normalization:

$$\mathbf{h}^t = f\left[\frac{\mathbf{g}}{\sigma^t} \odot \left(\mathbf{a}^t - \mu^t\right) + \mathbf{b}\right] \qquad \mu^t = \frac{1}{H}\sum_{i=1}^{H} a_i^t \qquad \sigma^t = \sqrt{\frac{1}{H}\sum_{i=1}^{H}\left(a_i^t - \mu^t\right)^2}$$

J. L. Ba, J. R. Kiros and G. E. Hinton. Layer normalization. https://arxiv.org/pdf/1607.06450.pdf

Transformers: Encoder-Decoder Connections

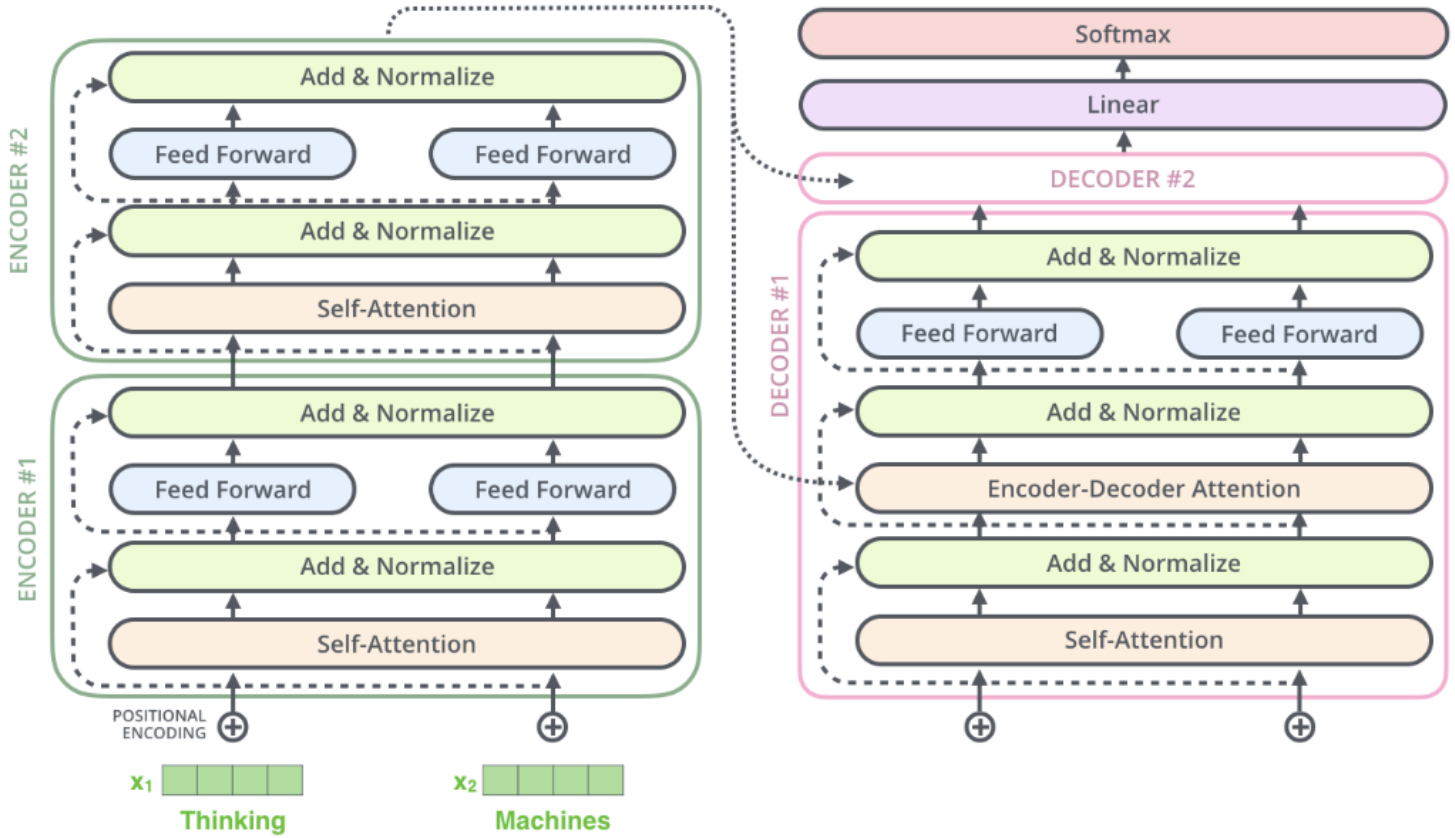


Image Source: The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time. (jalammar.github.io)

Transformers: Decoding

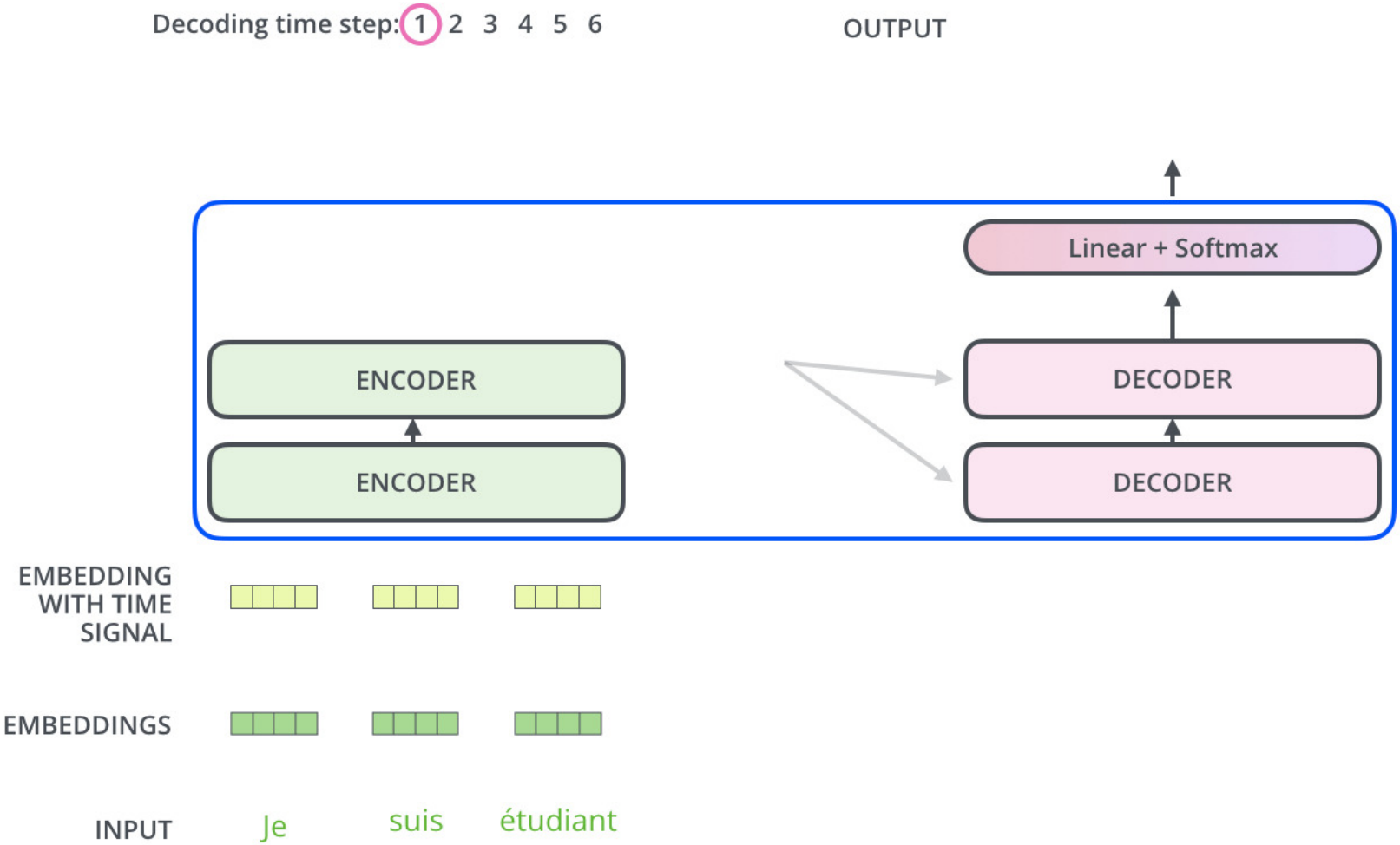


Image credit: https://jalammar.github.io/illustrated-transformer/

Transformers: Decoding

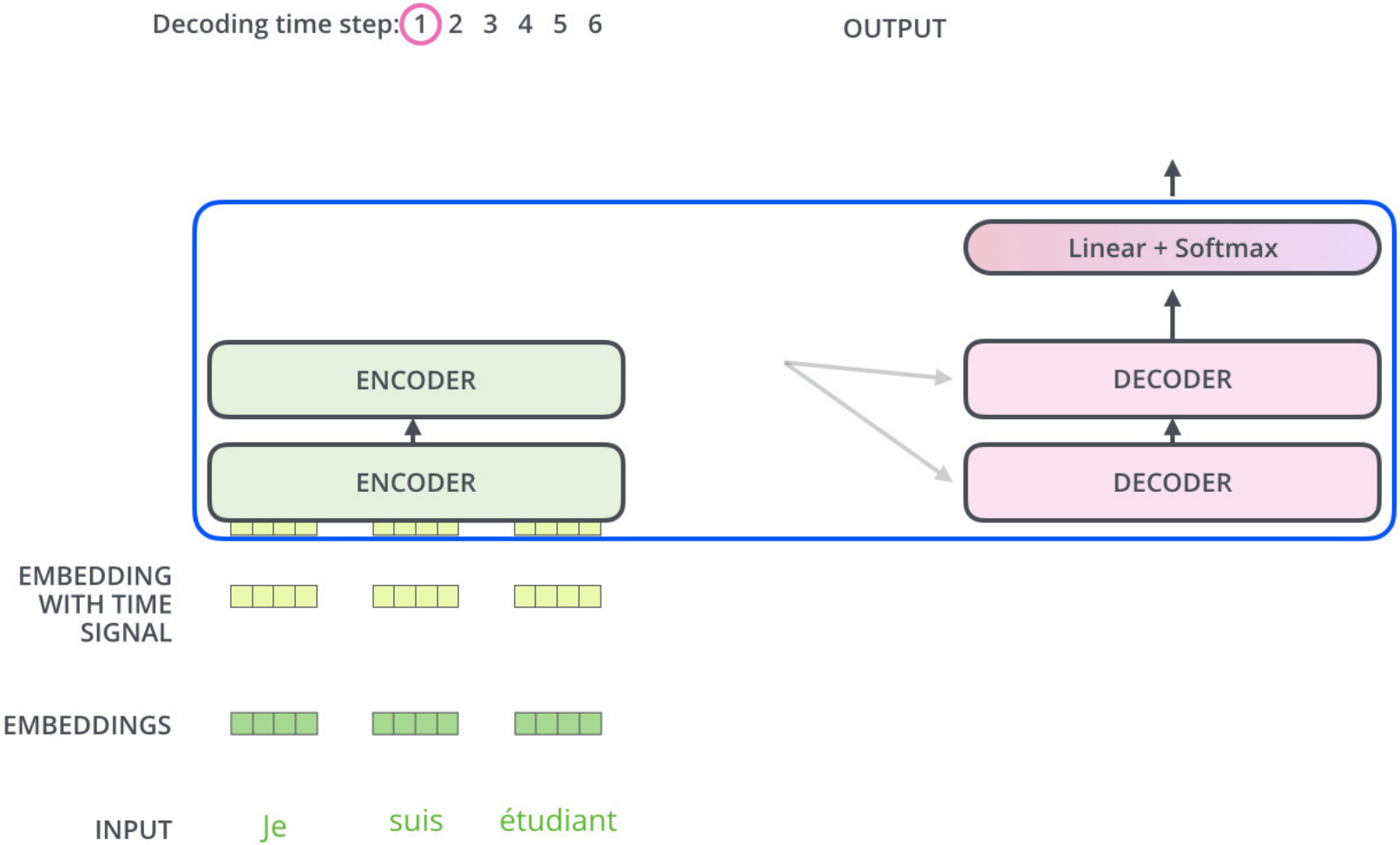


Image credit: https://jalammar.github.io/illustrated-transformer/

Transformers: Decoding

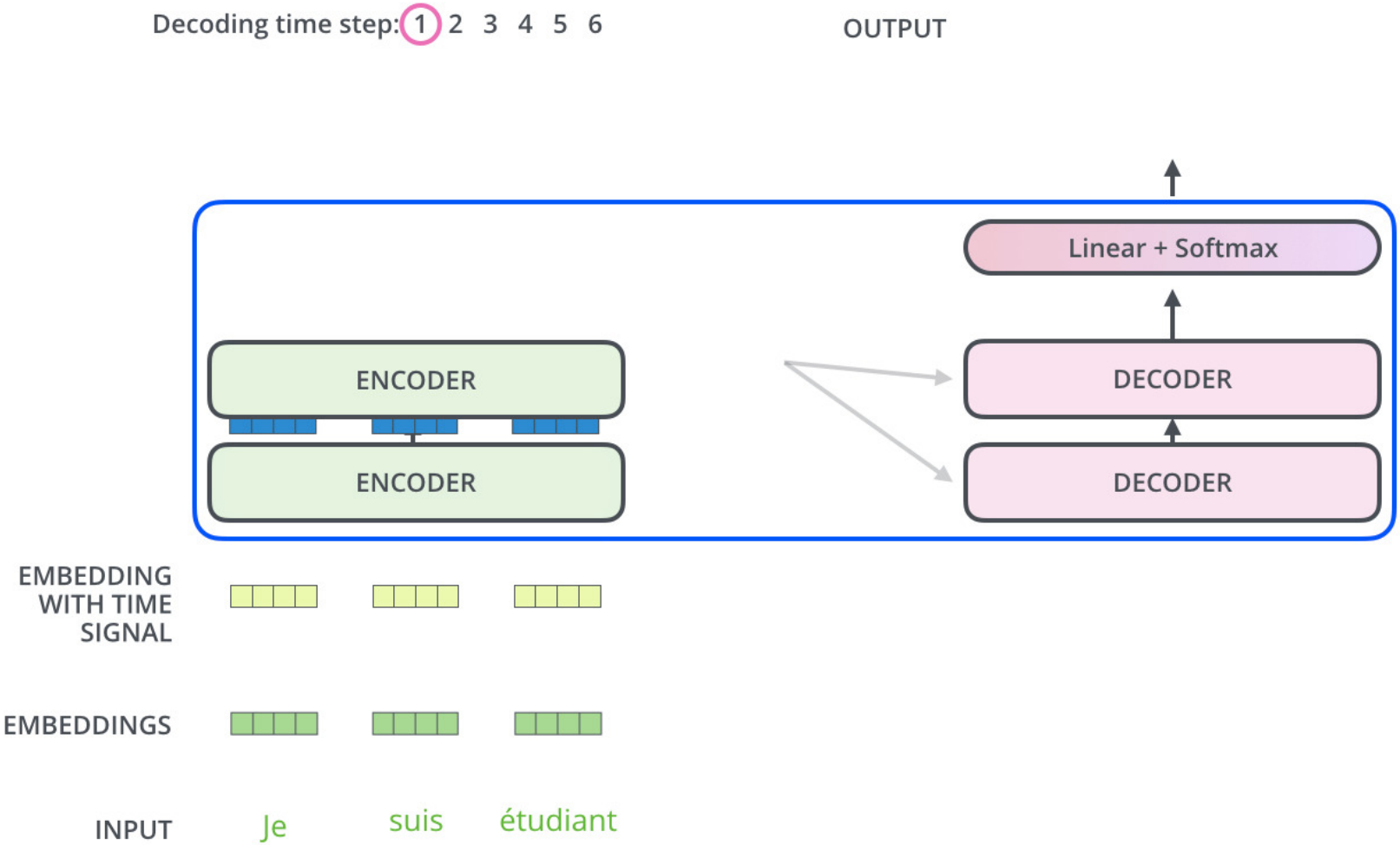


Image credit: https://jalammar.github.io/illustrated-transformer/

Transformers: Decoding

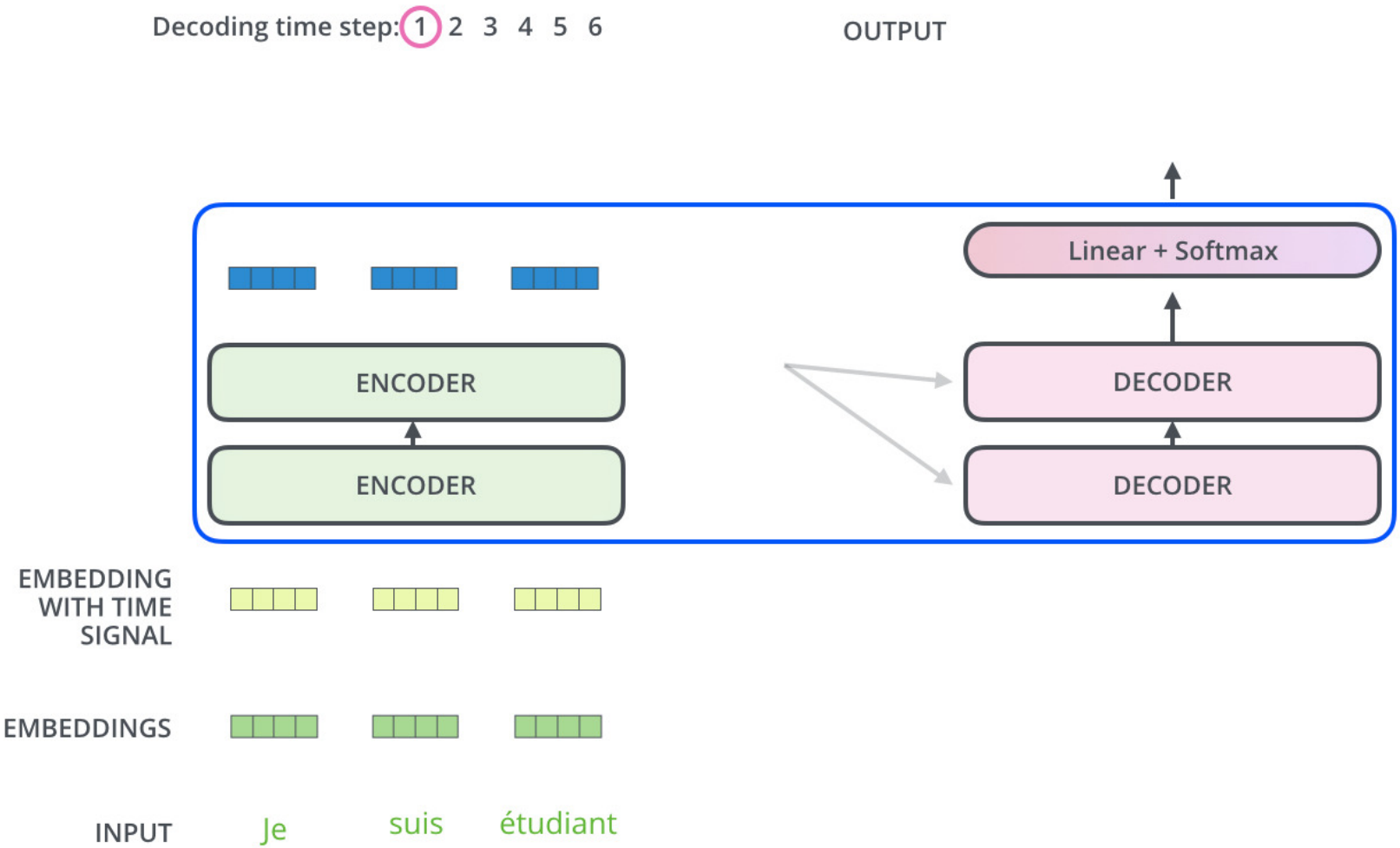


Image credit: https://jalammar.github.io/illustrated-transformer/

Transformers: Decoding

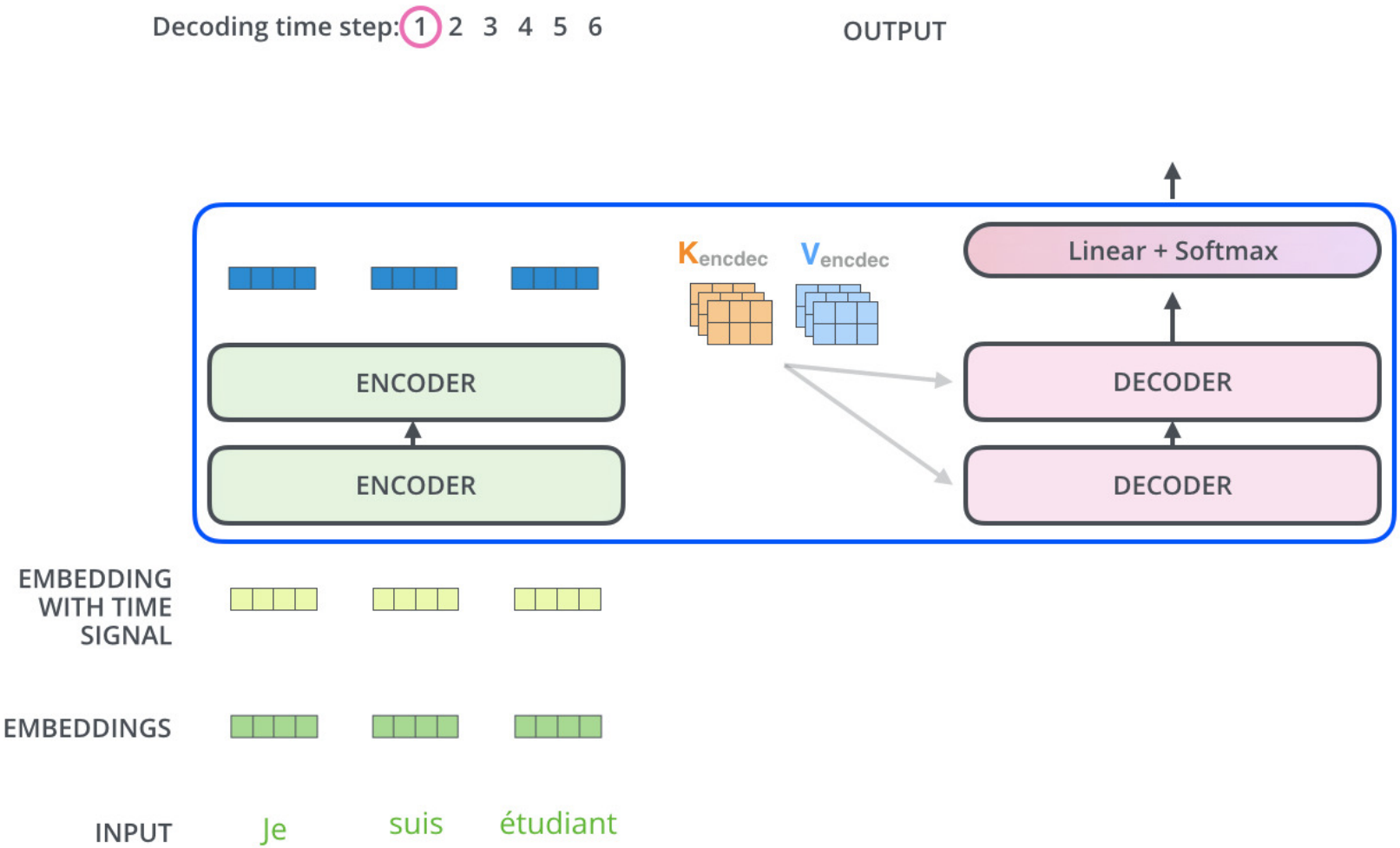


Image credit: https://jalammar.github.io/illustrated-transformer/

Transformers: Decoding

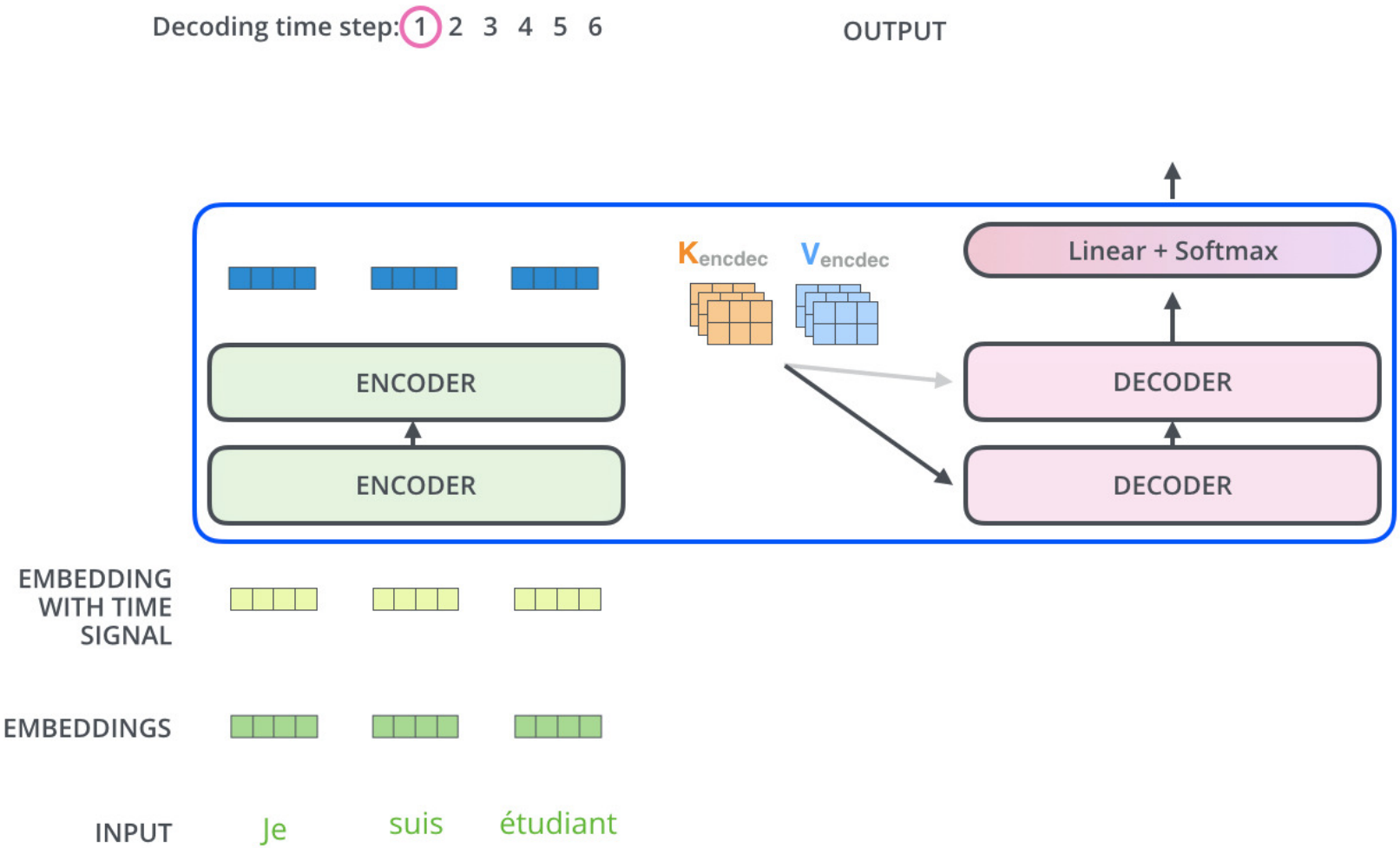


Image credit: https://jalammar.github.io/illustrated-transformer/

Transformers: Decoding

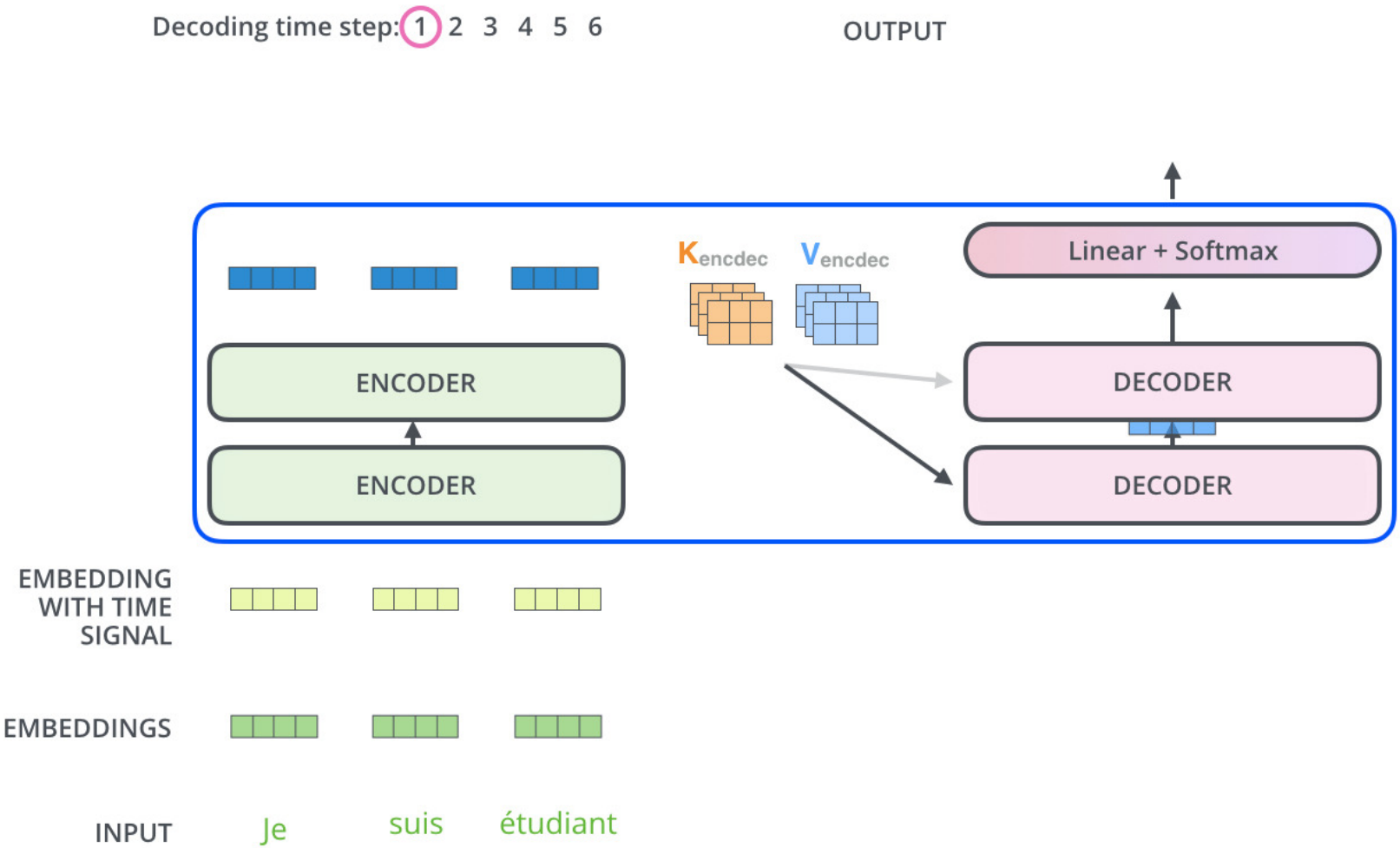


Image credit: https://jalammar.github.io/illustrated-transformer/

Transformers: Decoding

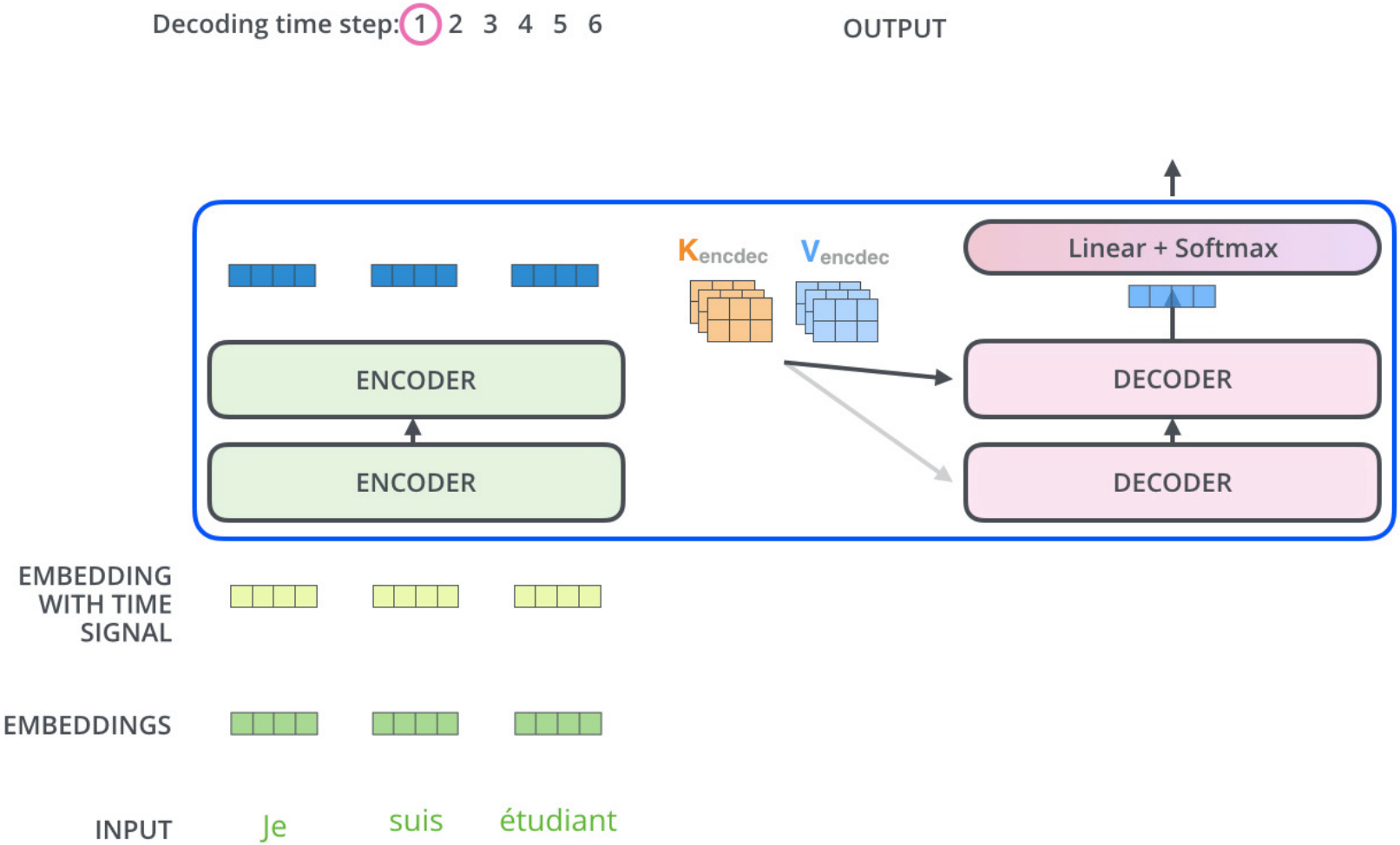


Image credit: https://jalammar.github.io/illustrated-transformer/

Transformers: Decoding

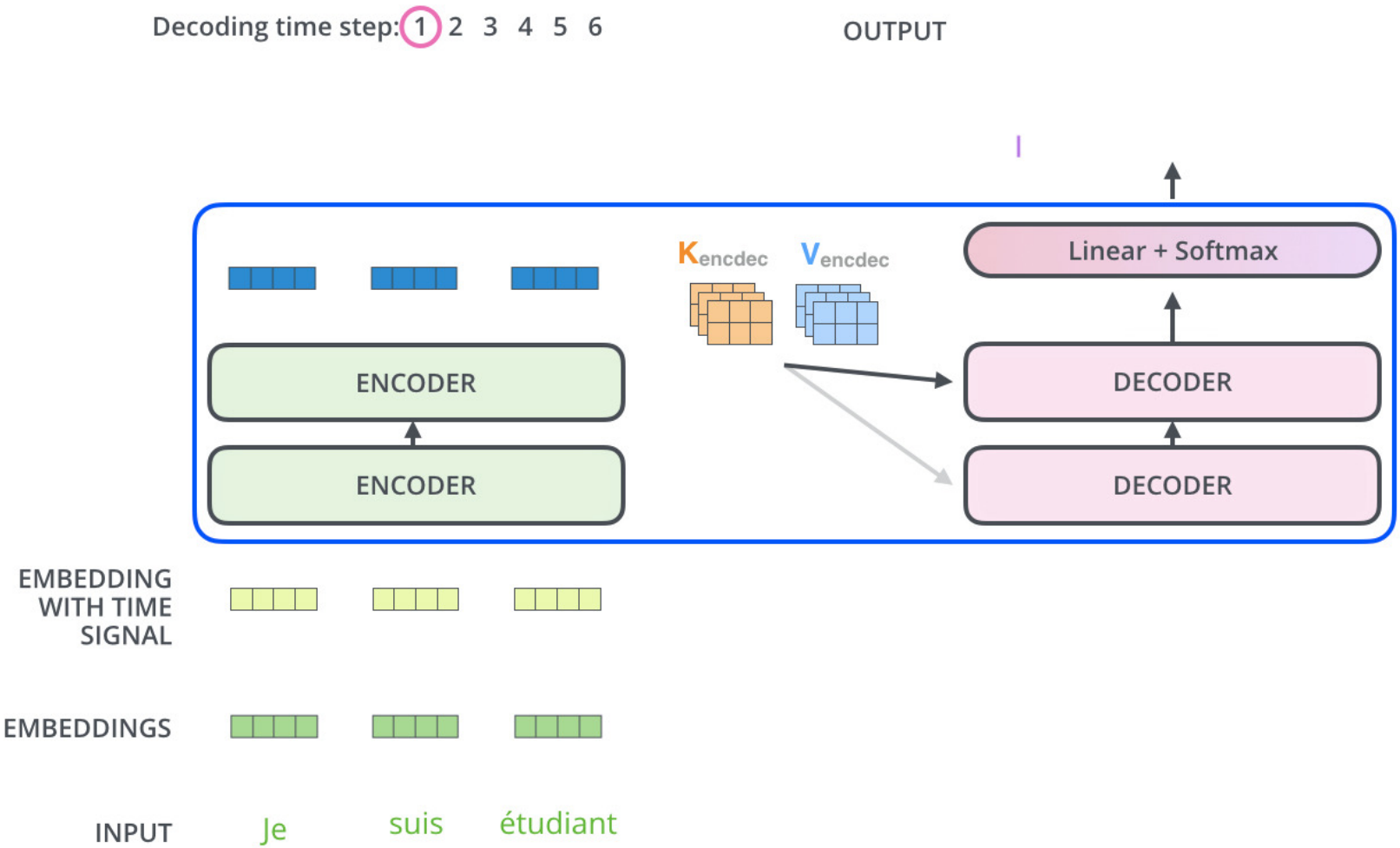


Image credit: https://jalammar.github.io/illustrated-transformer/

Transformers: Decoding

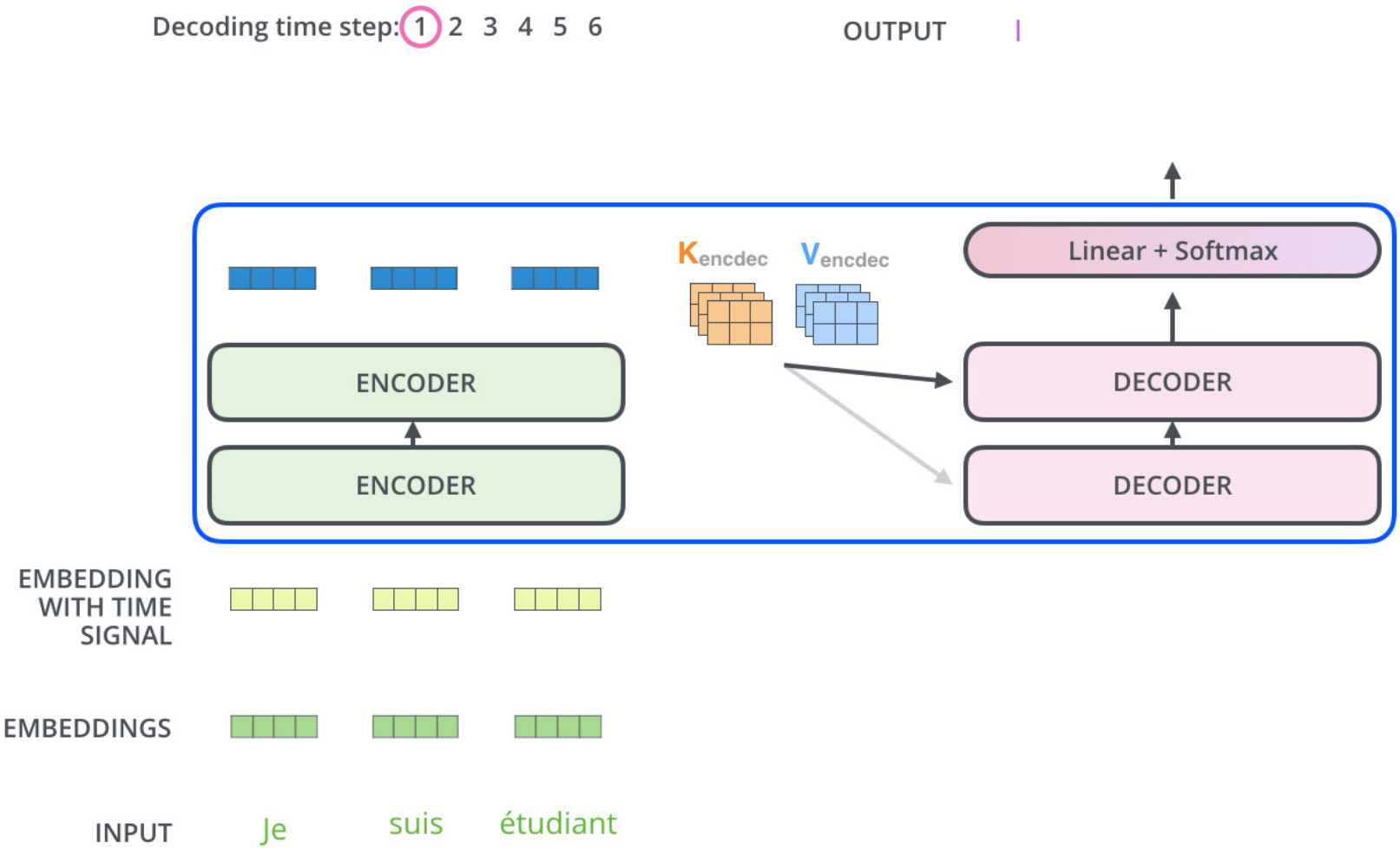


Image credit: https://jalammar.github.io/illustrated-transformer/

Transformers: Decoding

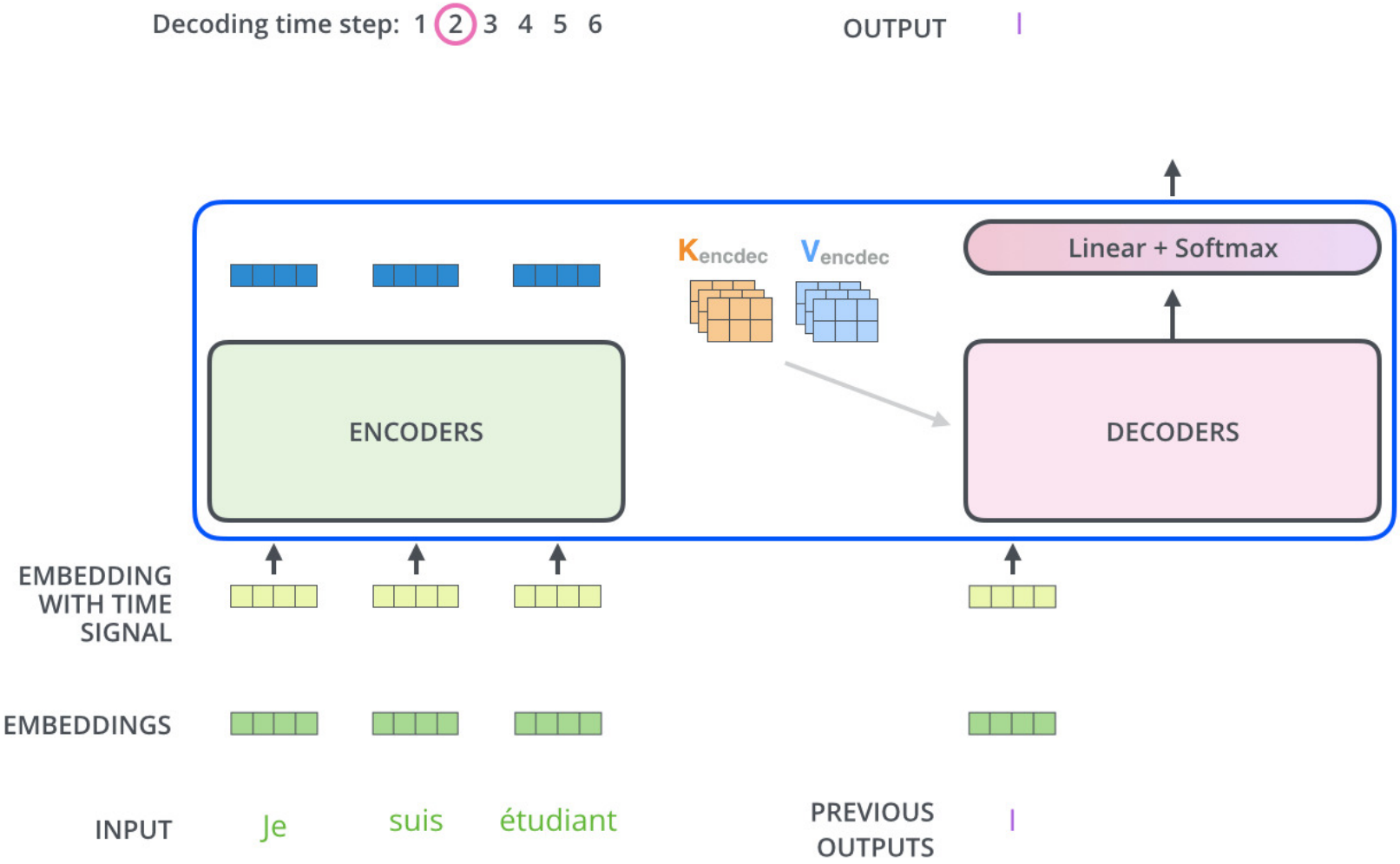


Image credit: https://jalammar.github.io/illustrated-transformer/

Transformers: Decoding

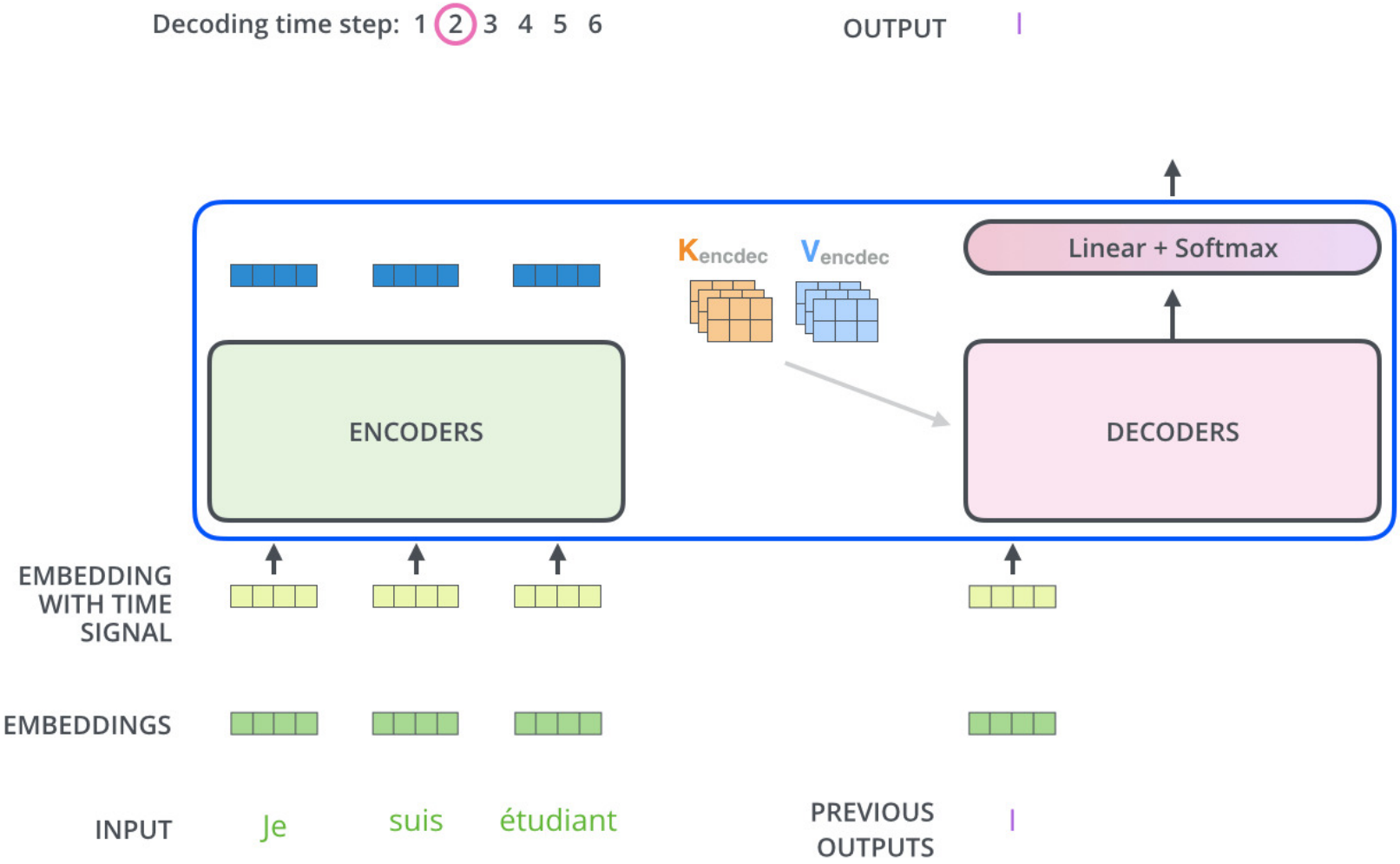


Image credit: https://jalammar.github.io/illustrated-transformer/

Transformers: Decoding

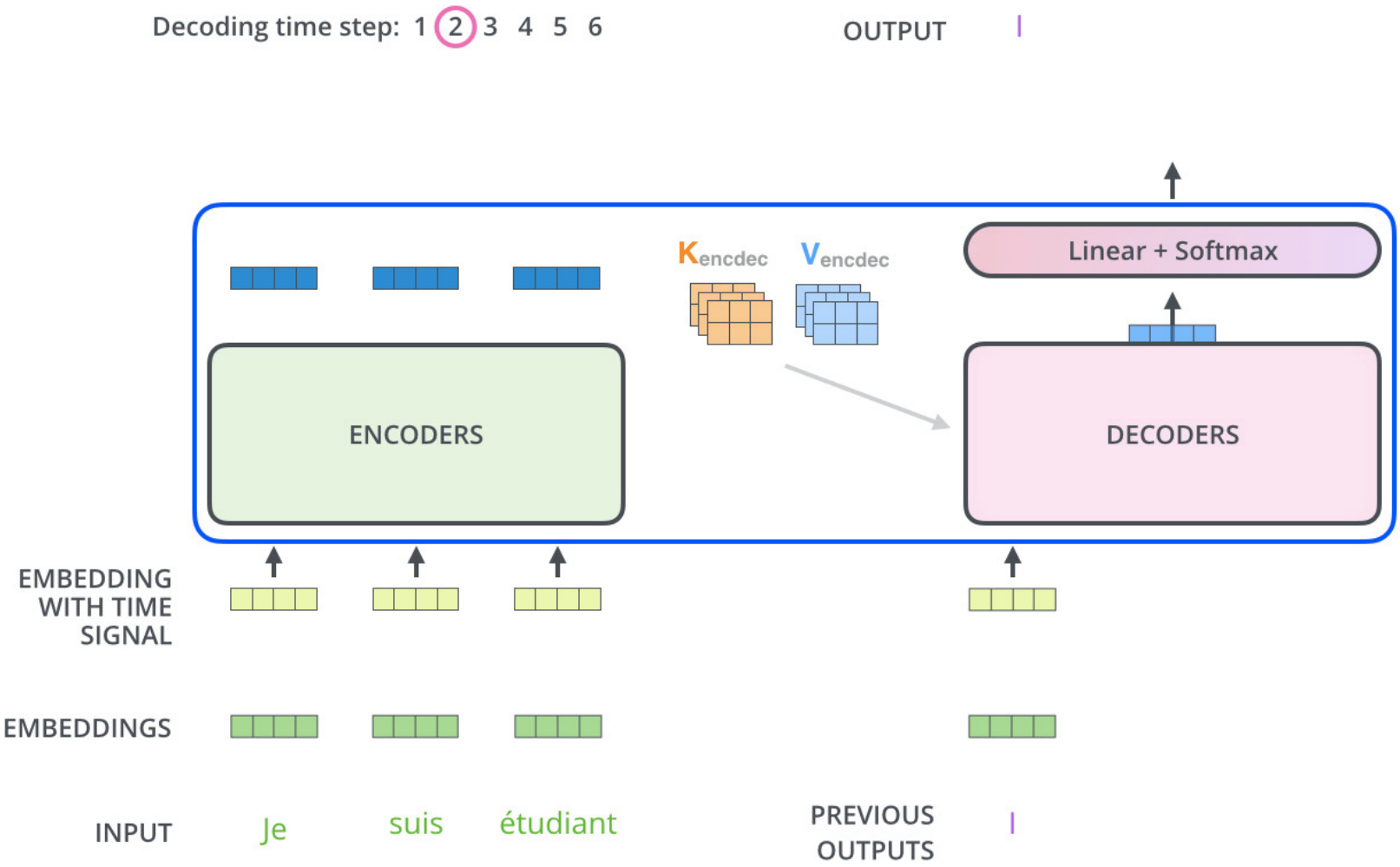


Image credit: https://jalammar.github.io/illustrated-transformer/

Transformers: Decoding

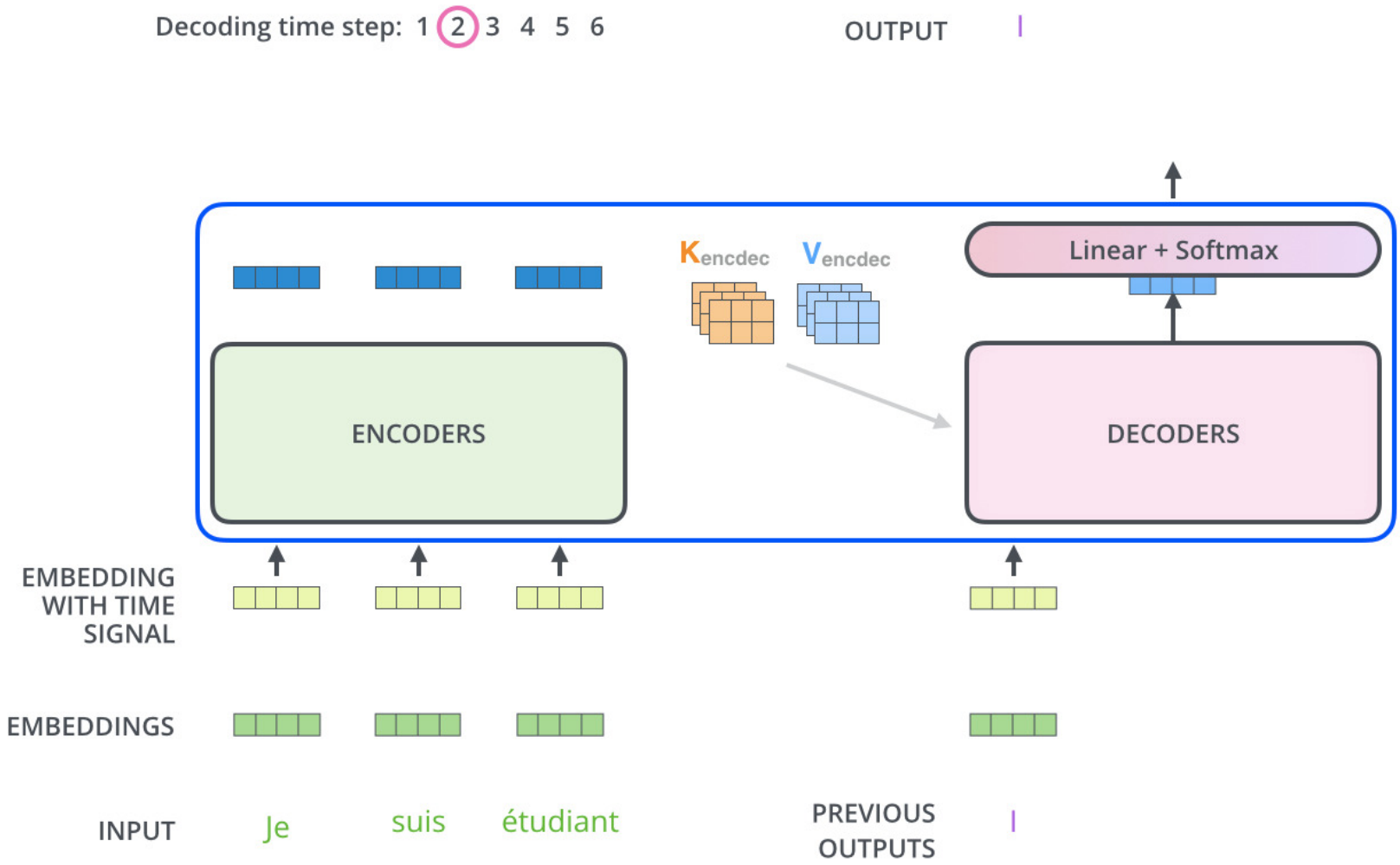


Image credit: https://jalammar.github.io/illustrated-transformer/

Transformers: Decoding

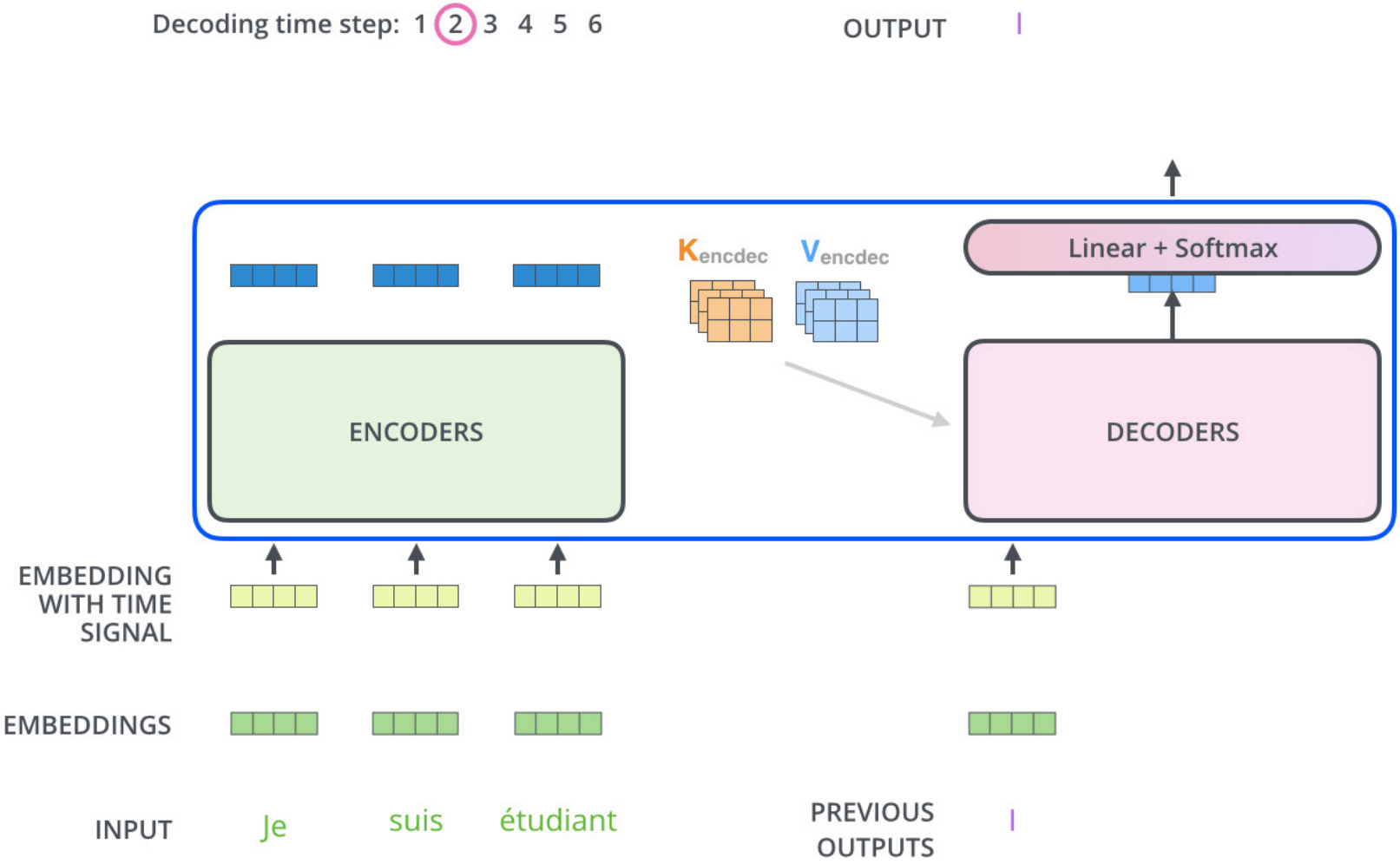


Image credit: https://jalammar.github.io/illustrated-transformer/

Transformers: Decoding

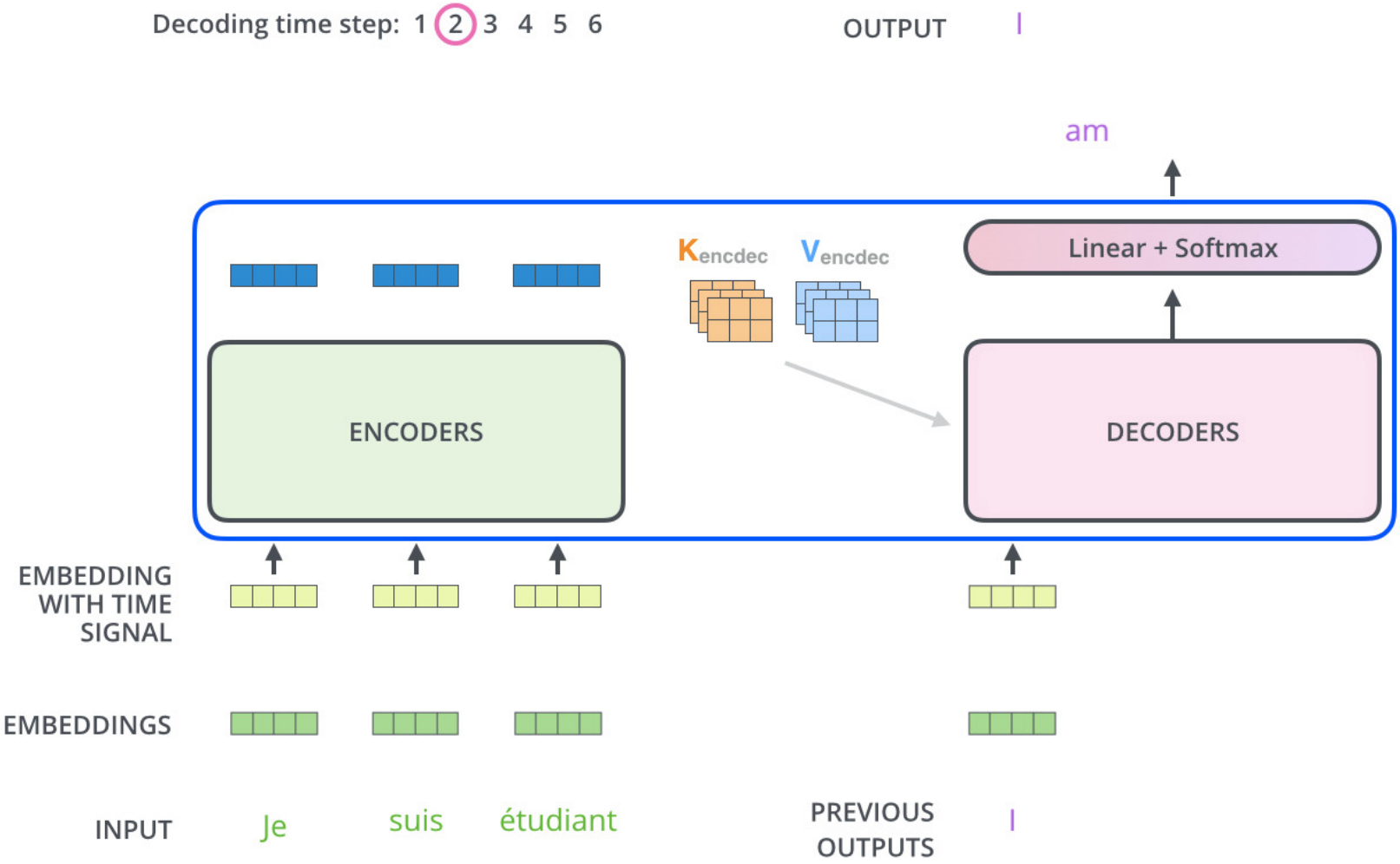


Image credit: https://jalammar.github.io/illustrated-transformer/

Transformers: Decoding

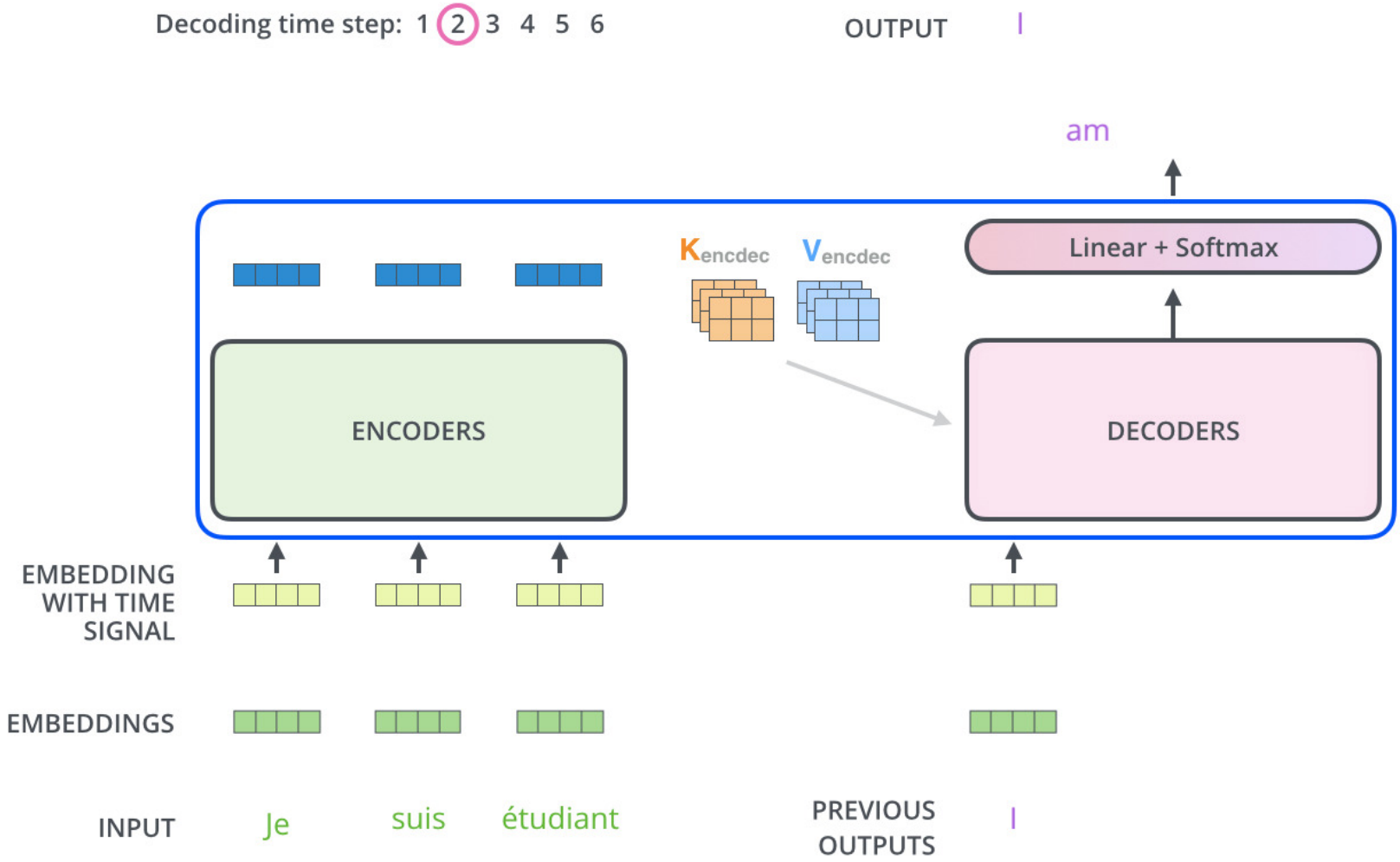


Image credit: https://jalammar.github.io/illustrated-transformer/

Transformers: Decoding

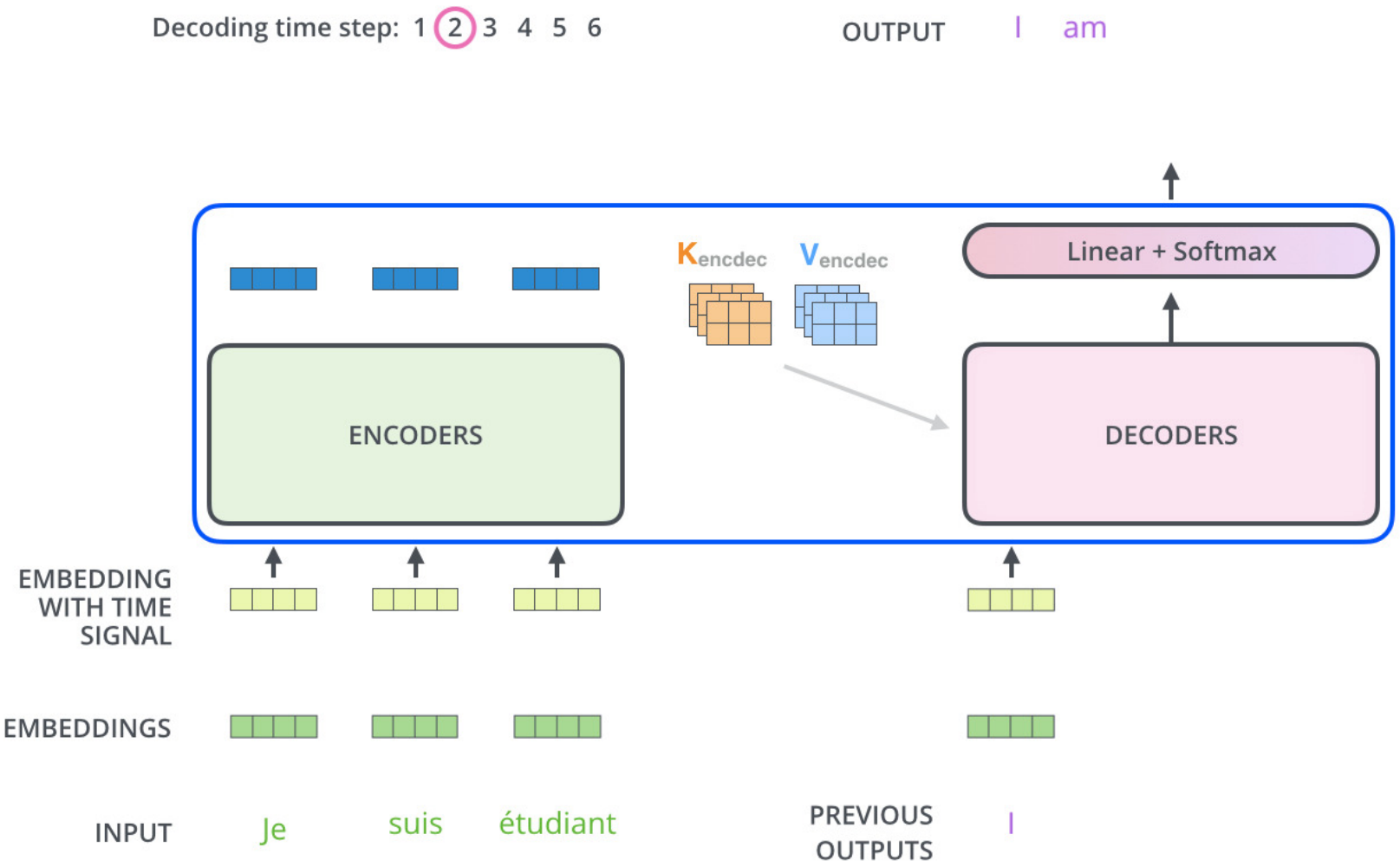


Image credit: https://jalammar.github.io/illustrated-transformer/

Transformers: Decoding

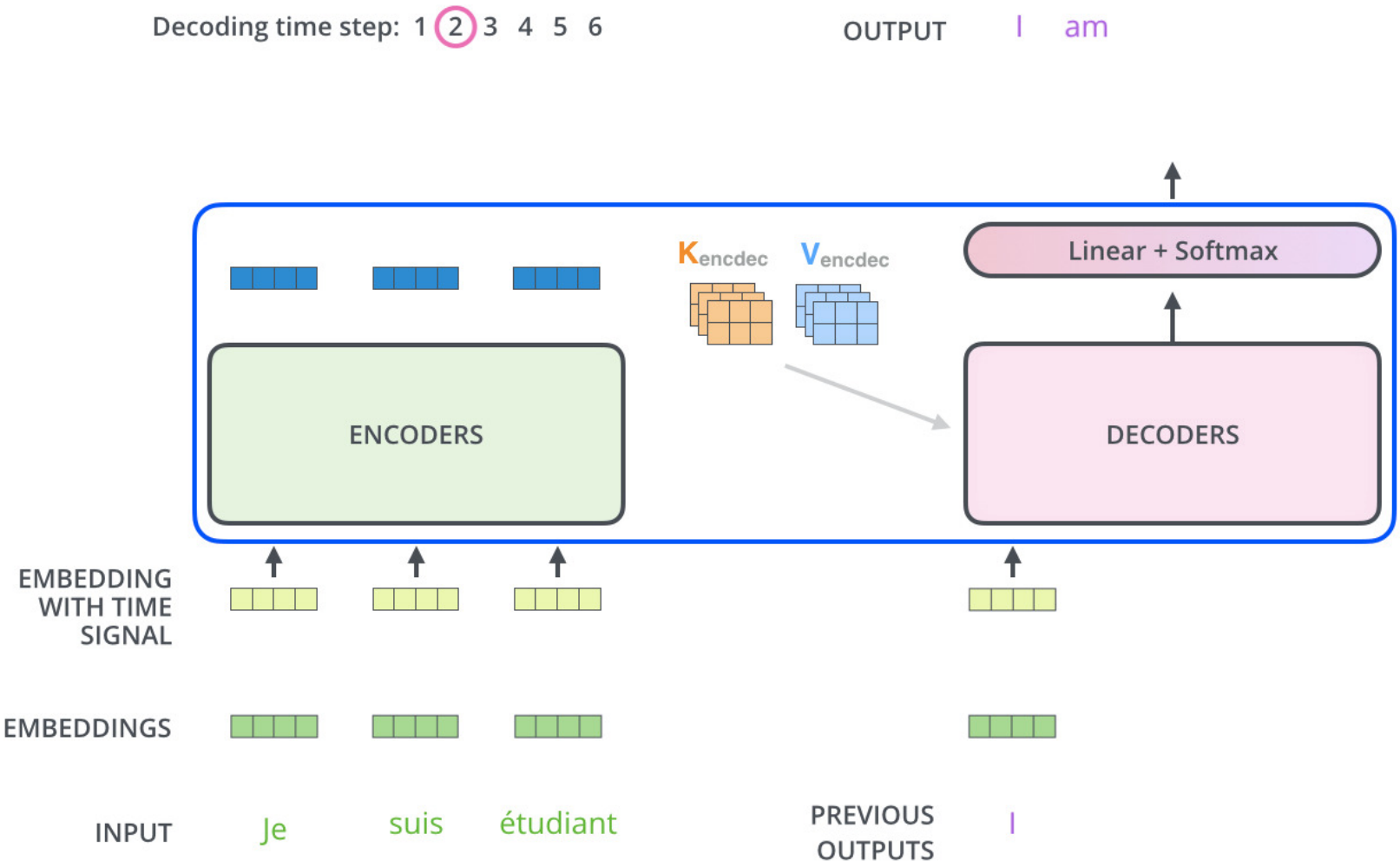


Image credit: https://jalammar.github.io/illustrated-transformer/

Transformers: Decoding

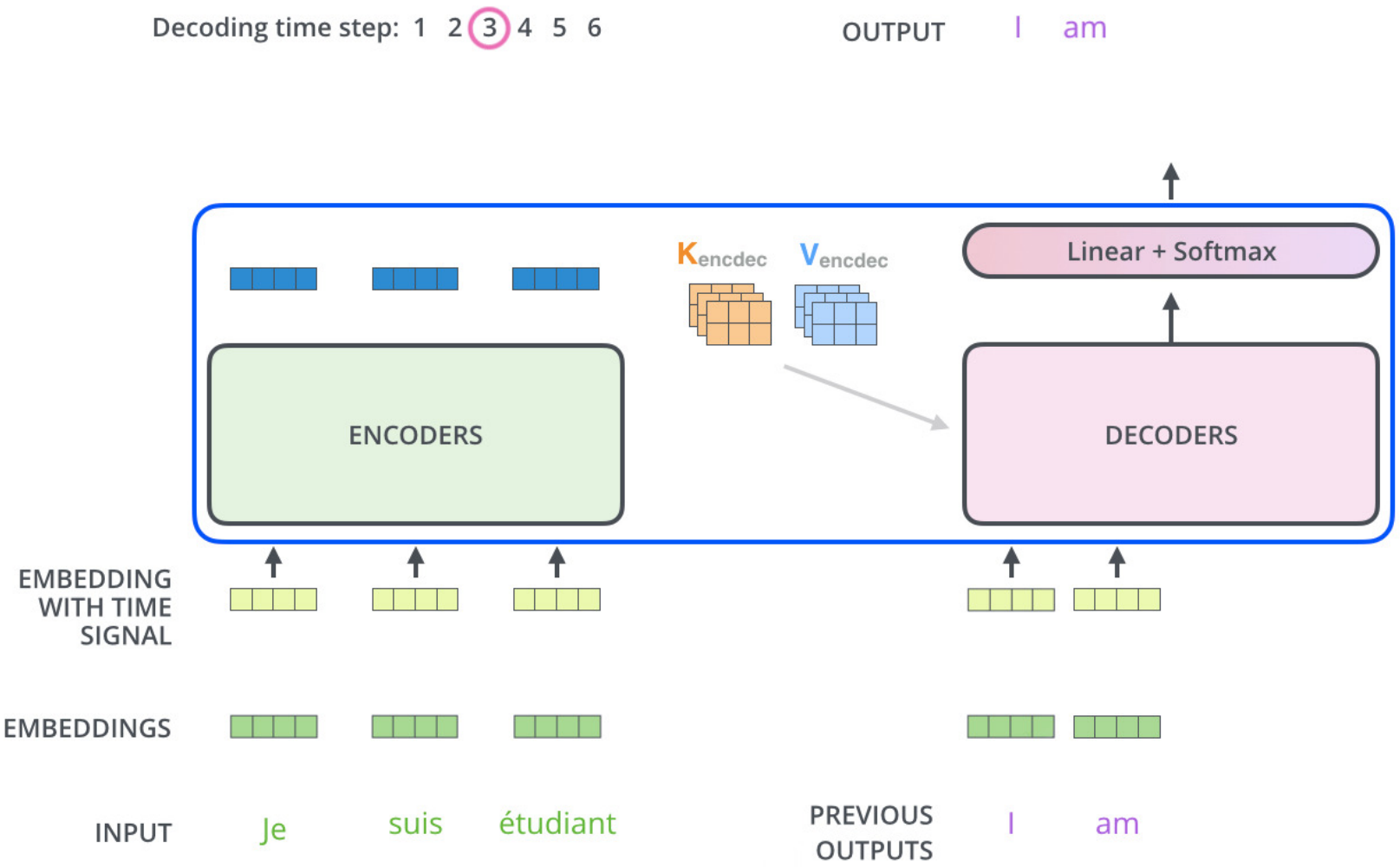


Image credit: https://jalammar.github.io/illustrated-transformer/

Transformers: Decoding

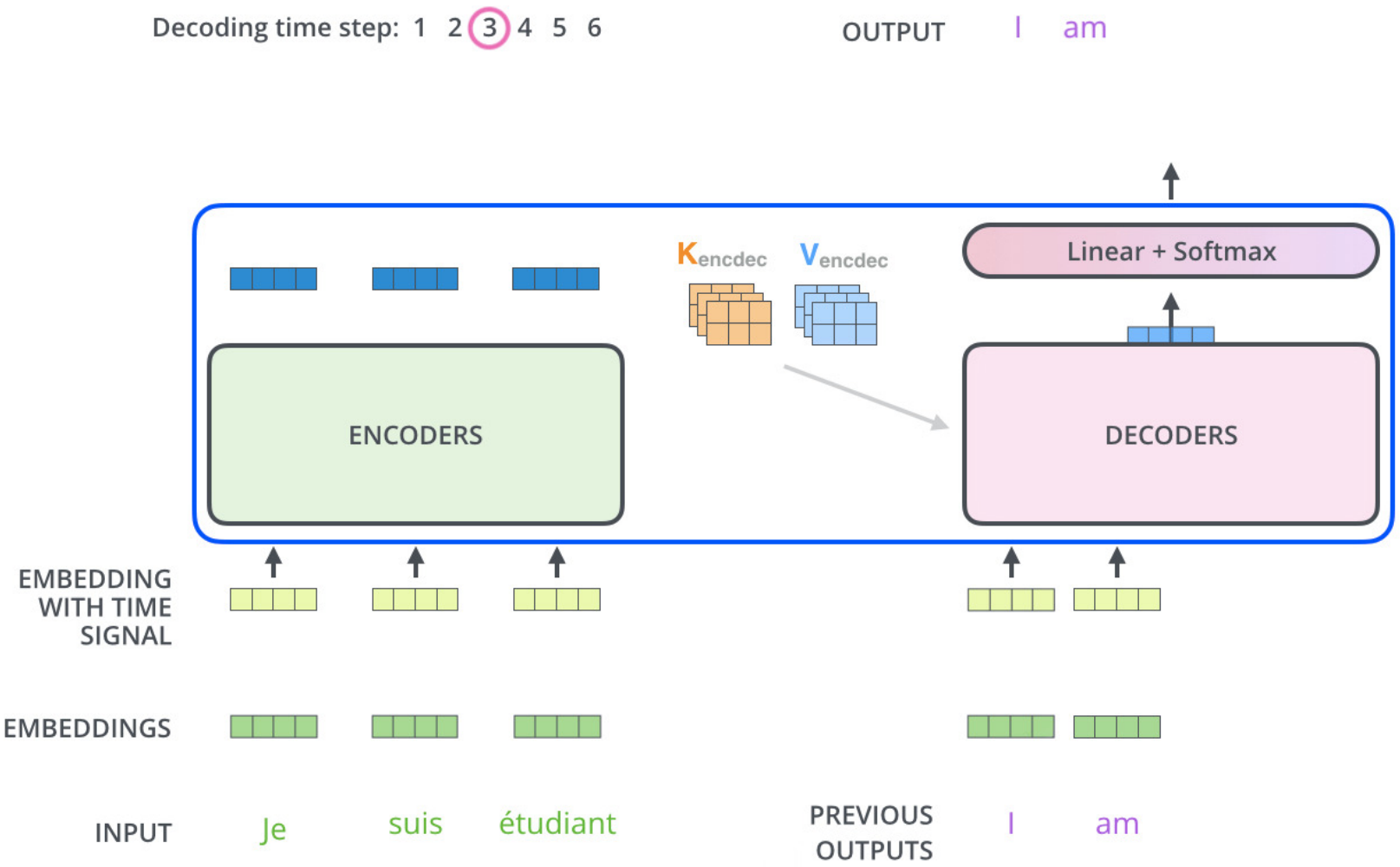


Image credit: https://jalammar.github.io/illustrated-transformer/

Transformers: Decoding

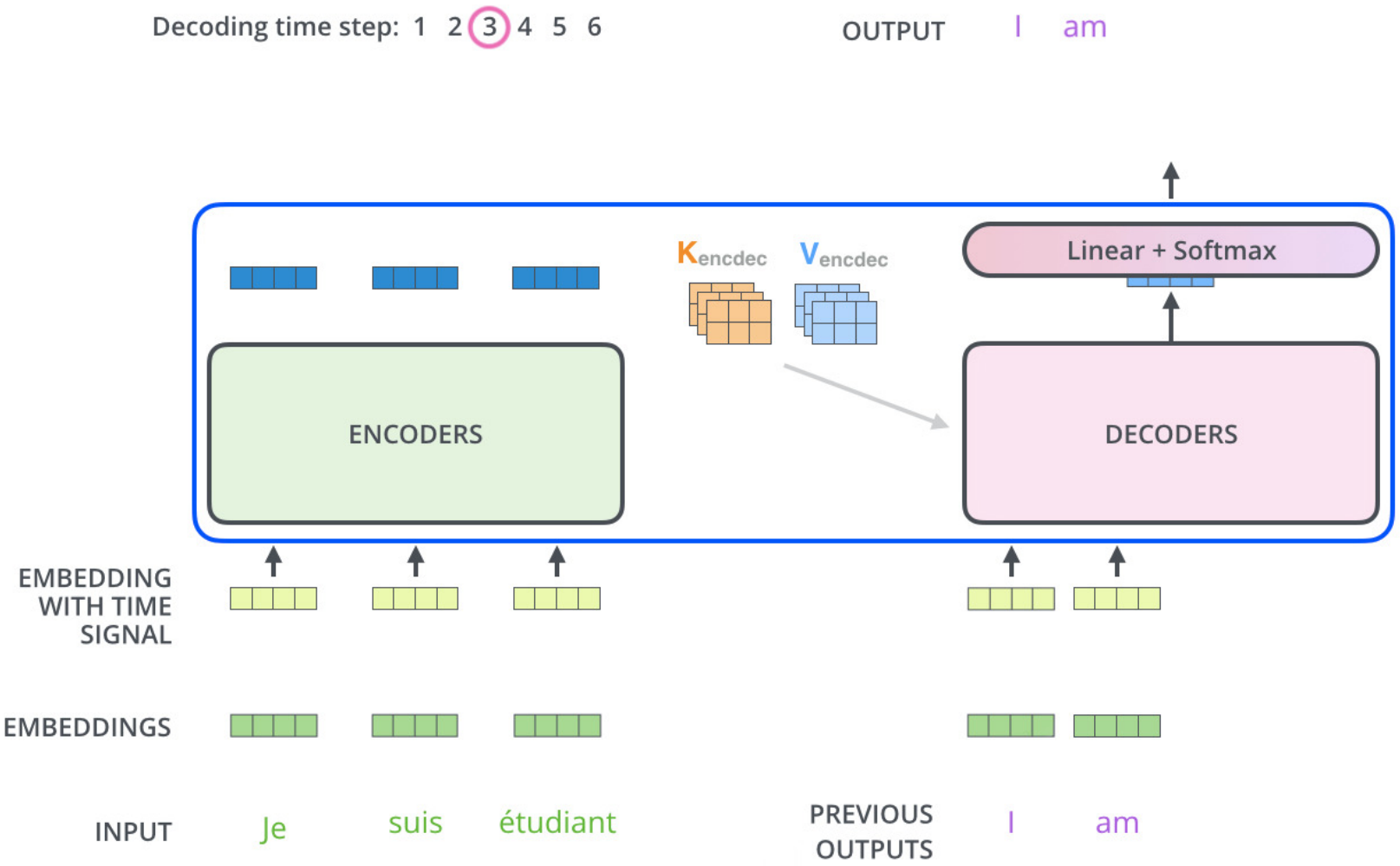


Image credit: https://jalammar.github.io/illustrated-transformer/

Transformers: Decoding

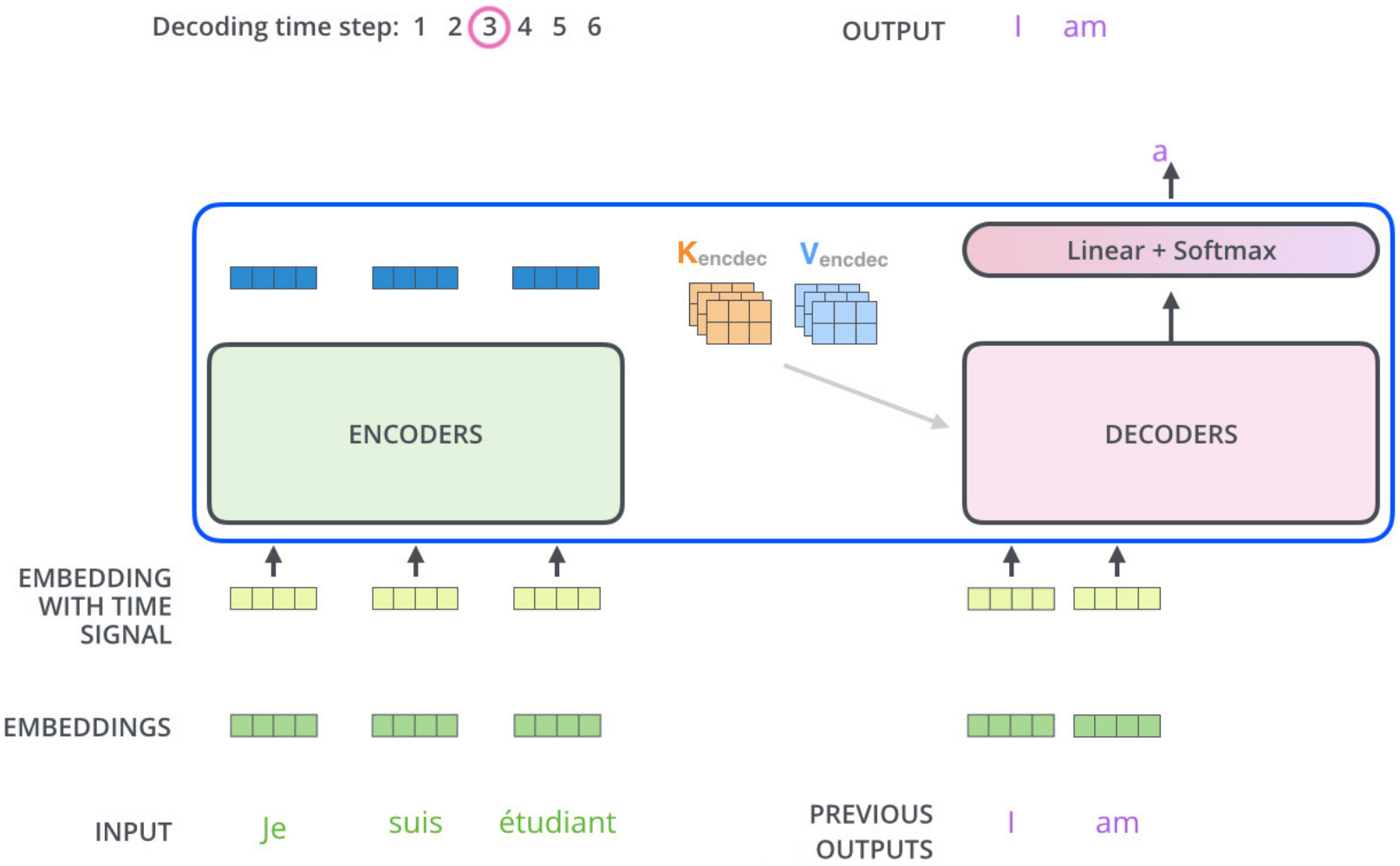


Image credit: https://jalammar.github.io/illustrated-transformer/

Transformers: Decoding

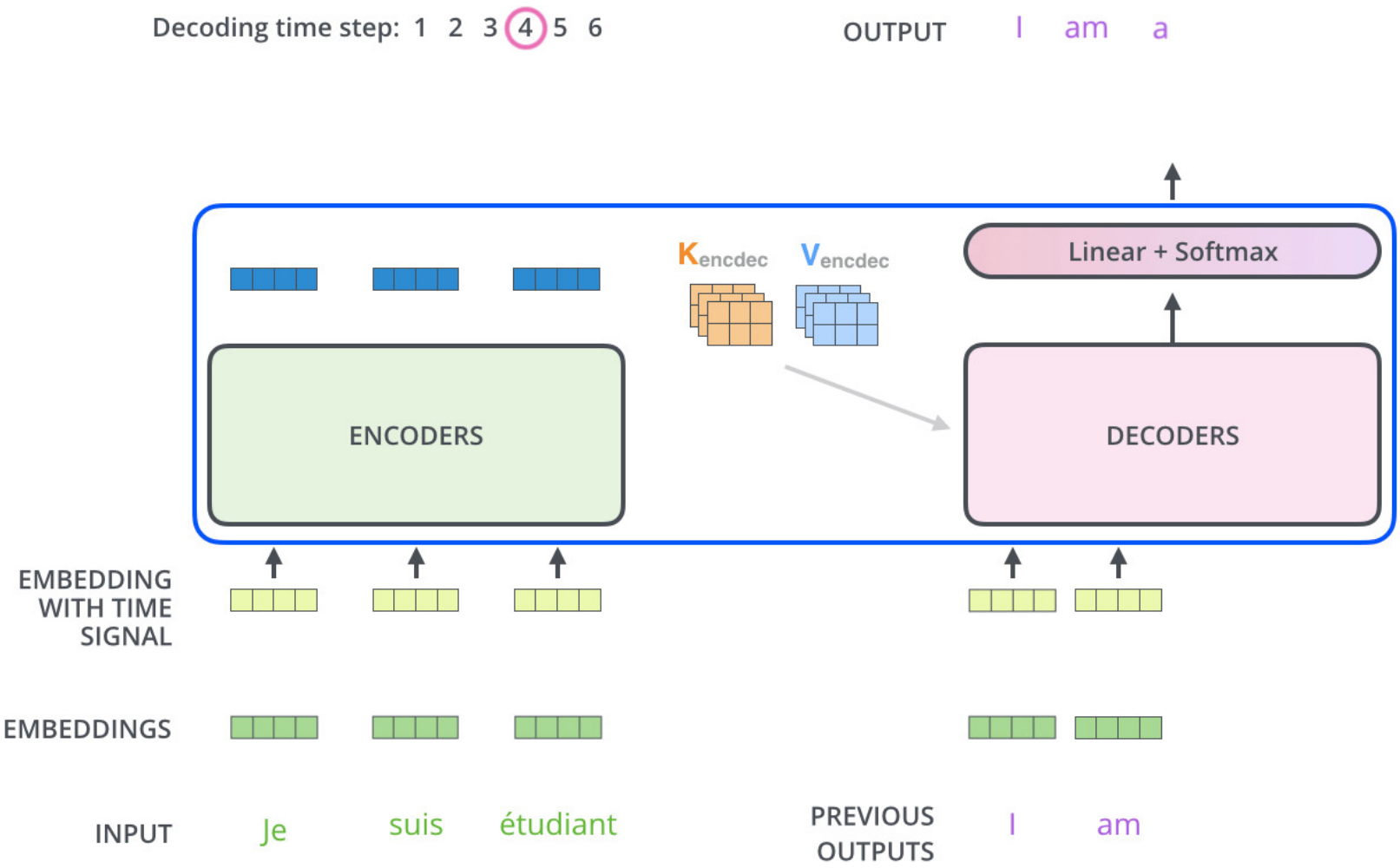


Image credit: https://jalammar.github.io/illustrated-transformer/

Transformers: Decoding

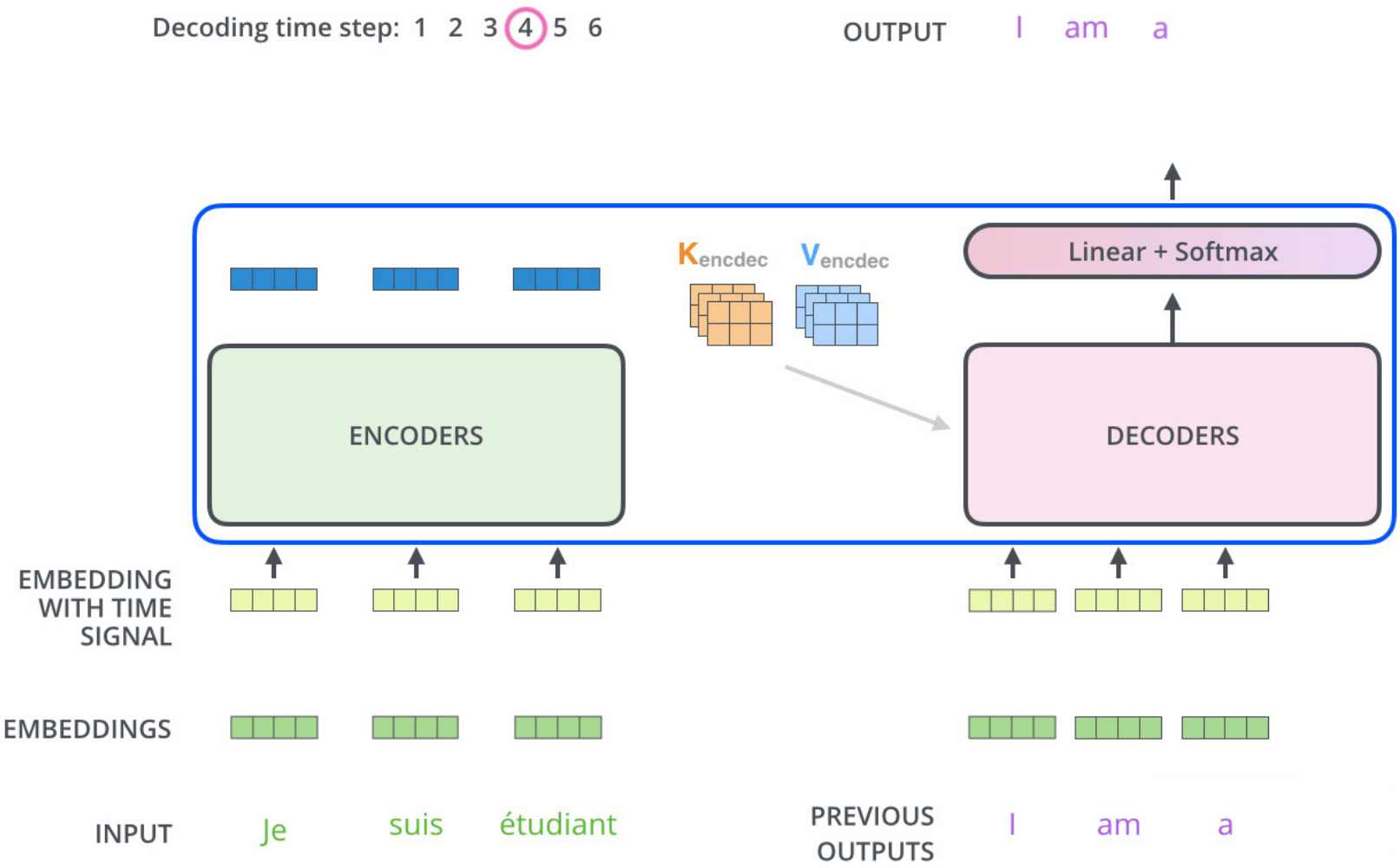


Image credit: https://jalammar.github.io/illustrated-transformer/

Transformers: Decoding

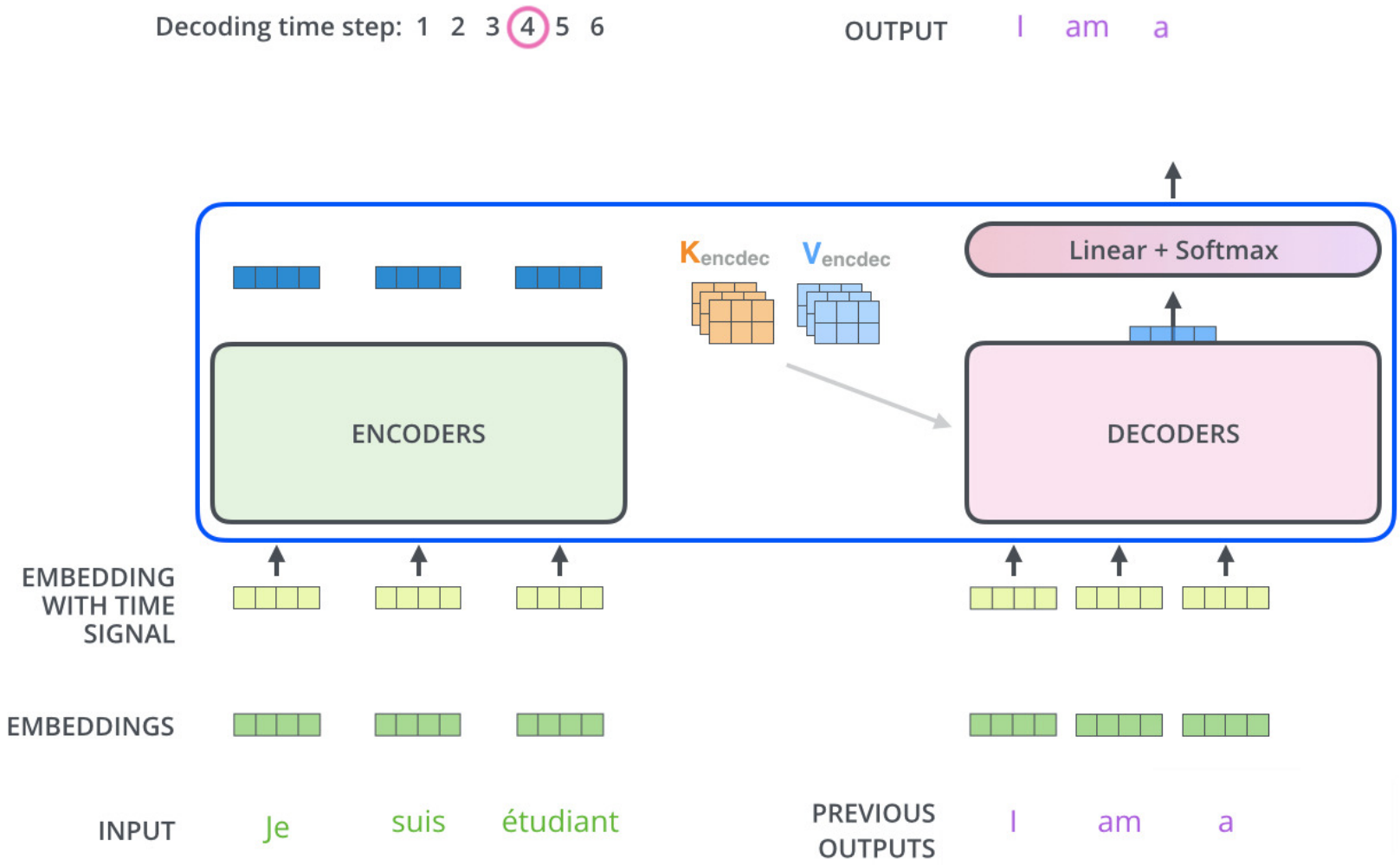


Image credit: https://jalammar.github.io/illustrated-transformer/

Transformers: Decoding

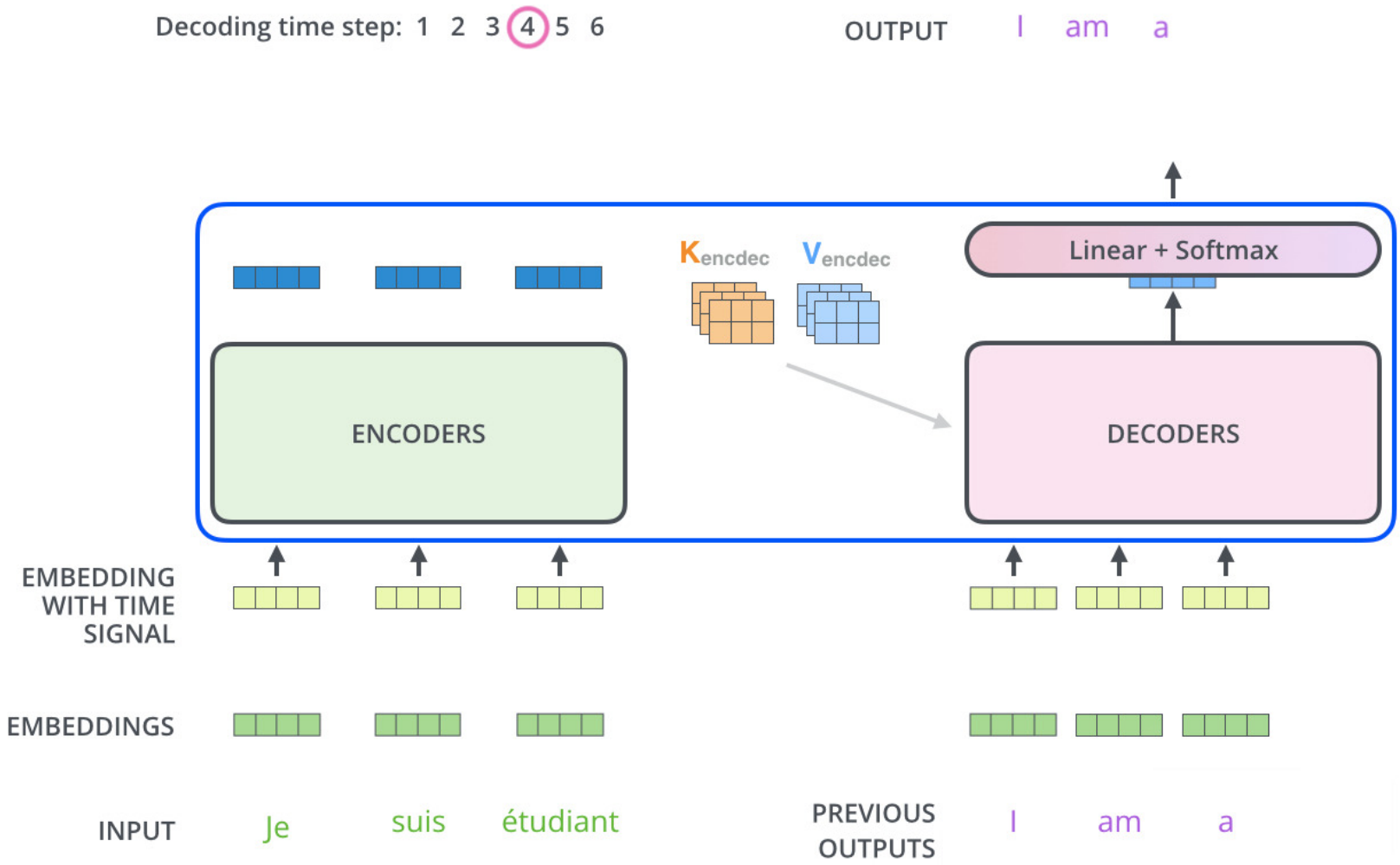


Image credit: https://jalammar.github.io/illustrated-transformer/

Transformers: Decoding

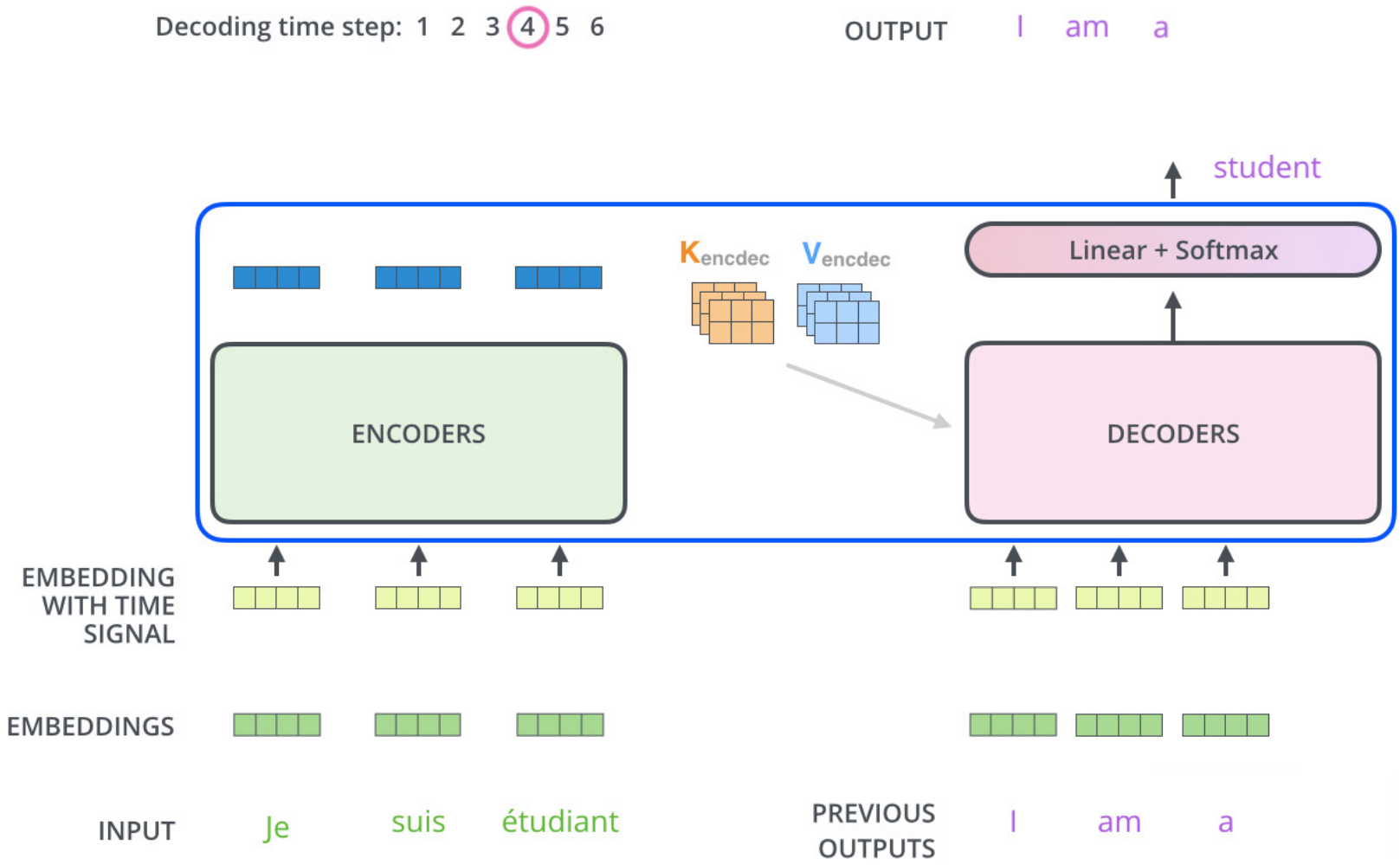


Image credit: https://jalammar.github.io/illustrated-transformer/

Transformers: Decoding

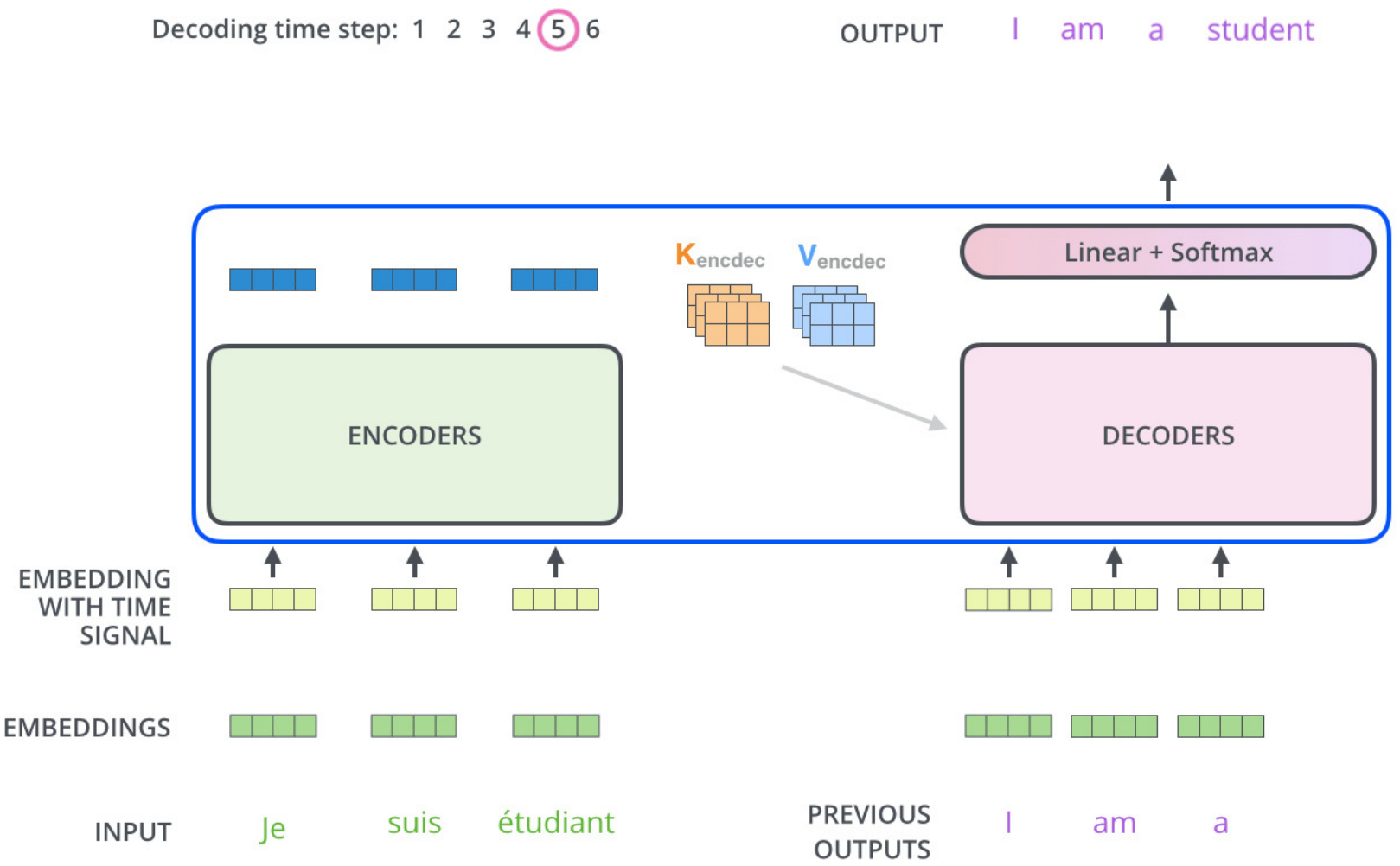


Image credit: https://jalammar.github.io/illustrated-transformer/

Transformers: Decoding

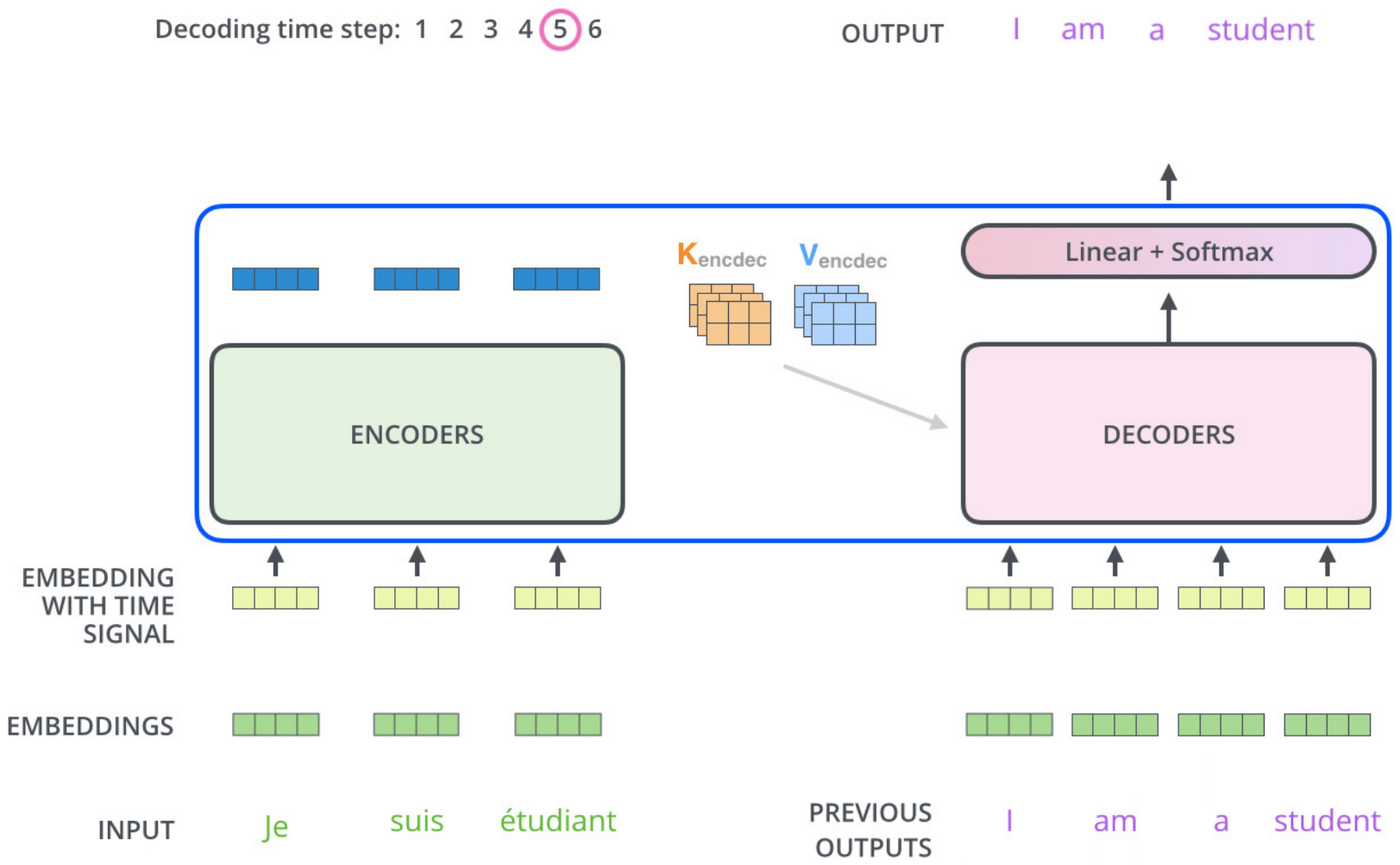


Image credit: https://jalammar.github.io/illustrated-transformer/

Transformers: Decoding

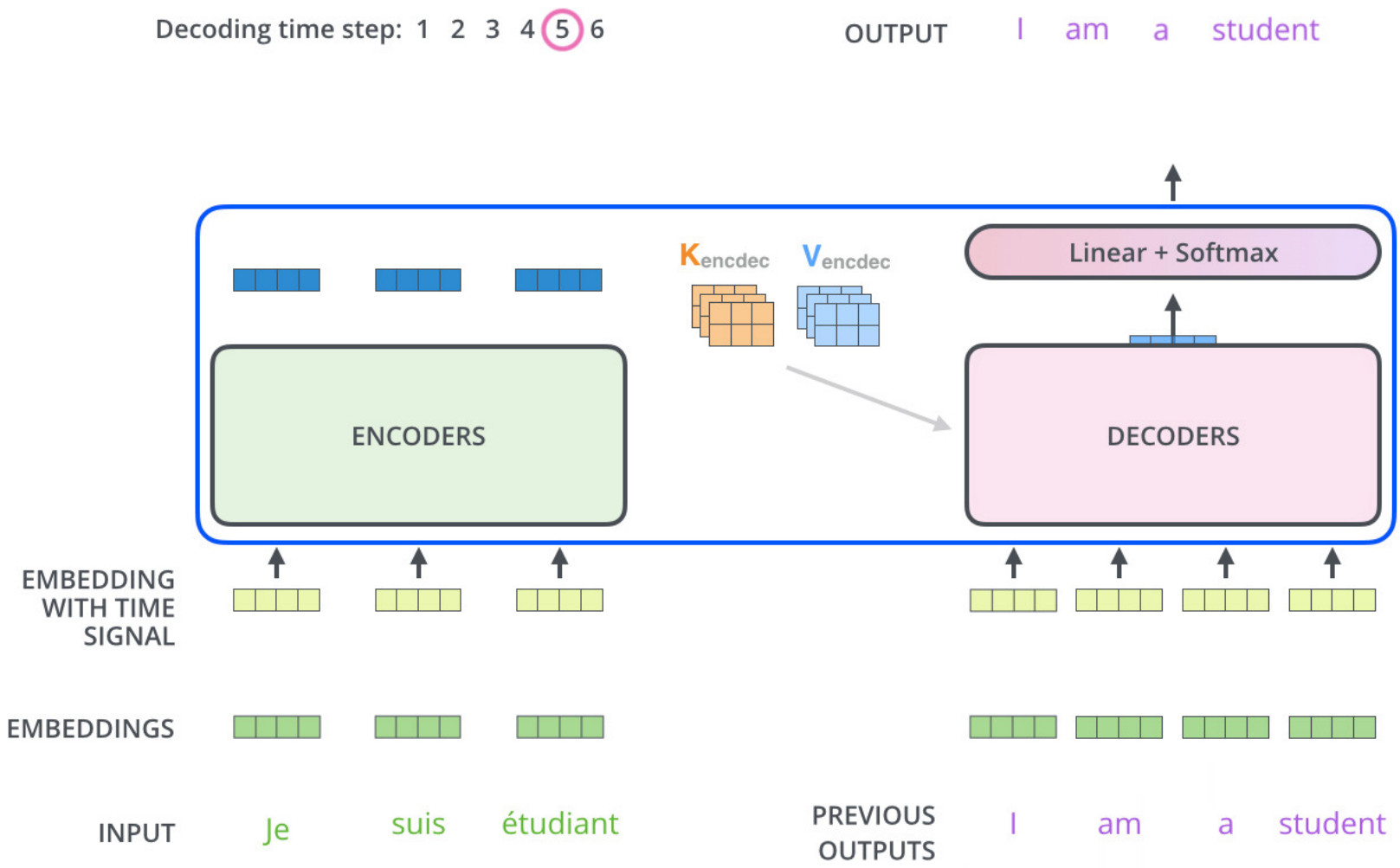


Image credit: https://jalammar.github.io/illustrated-transformer/

Transformers: Decoding

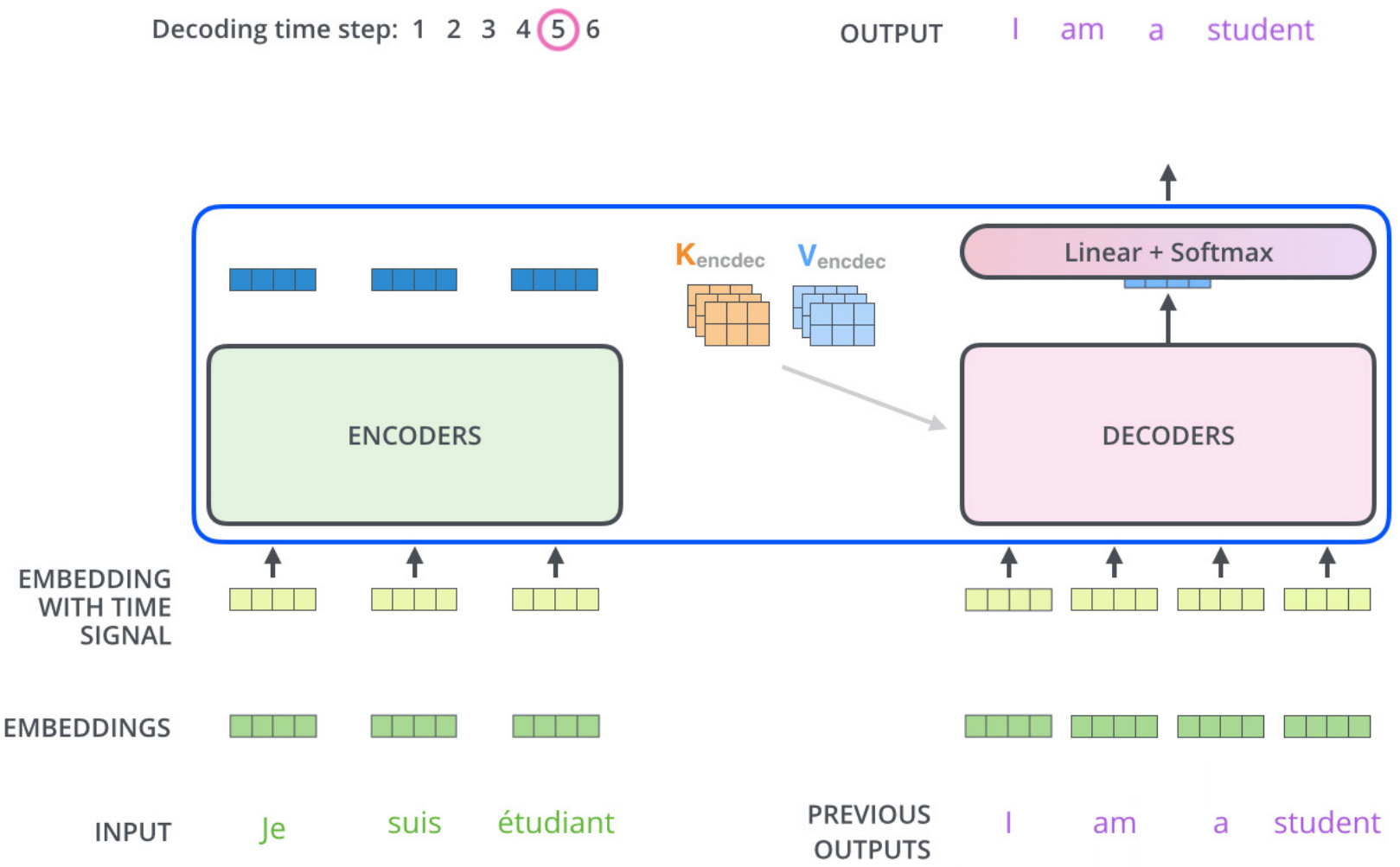


Image credit: https://jalammar.github.io/illustrated-transformer/

Transformers: Decoding

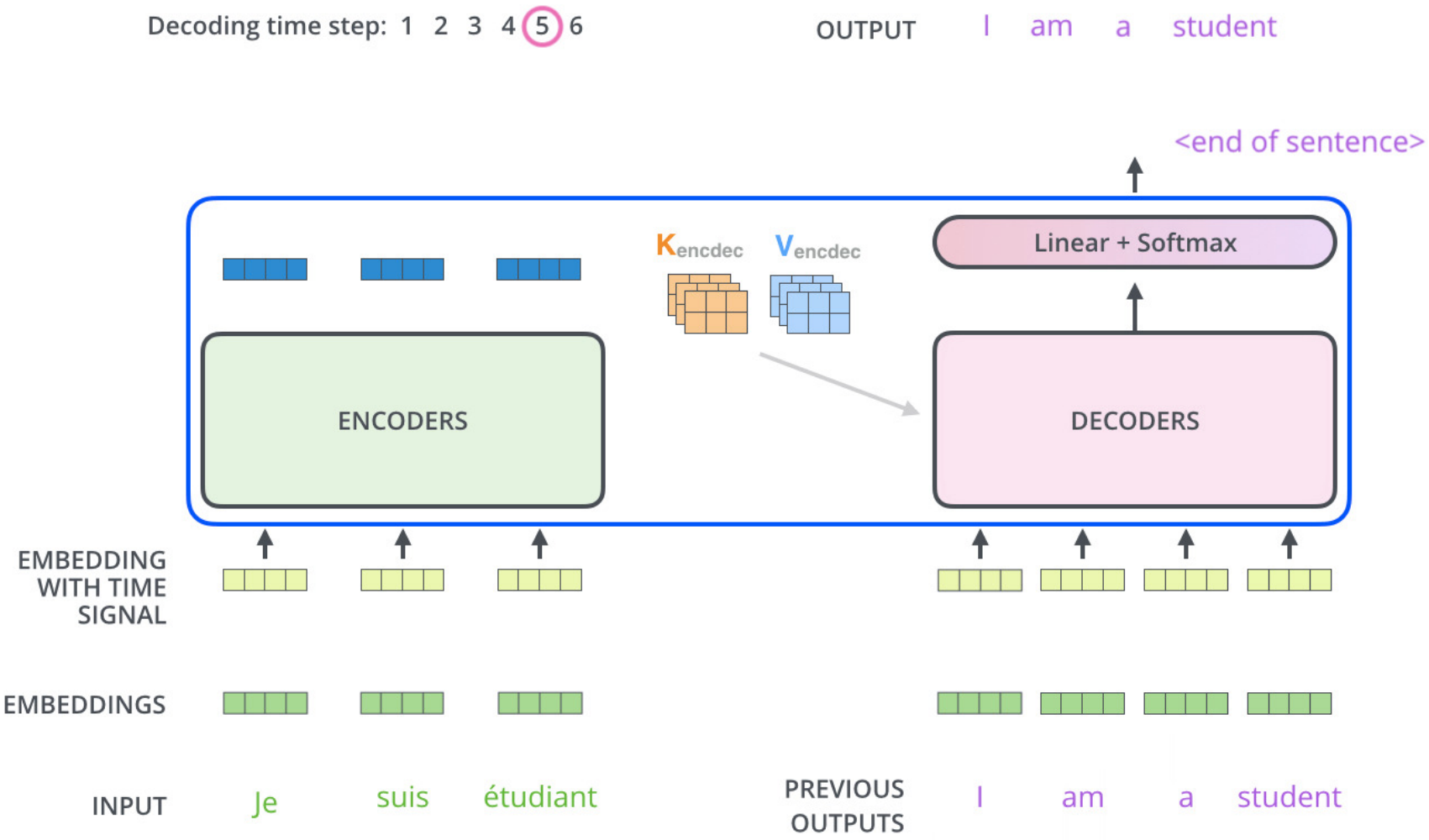


Image credit: https://jalammar.github.io/illustrated-transformer/

Transformers: Decoding

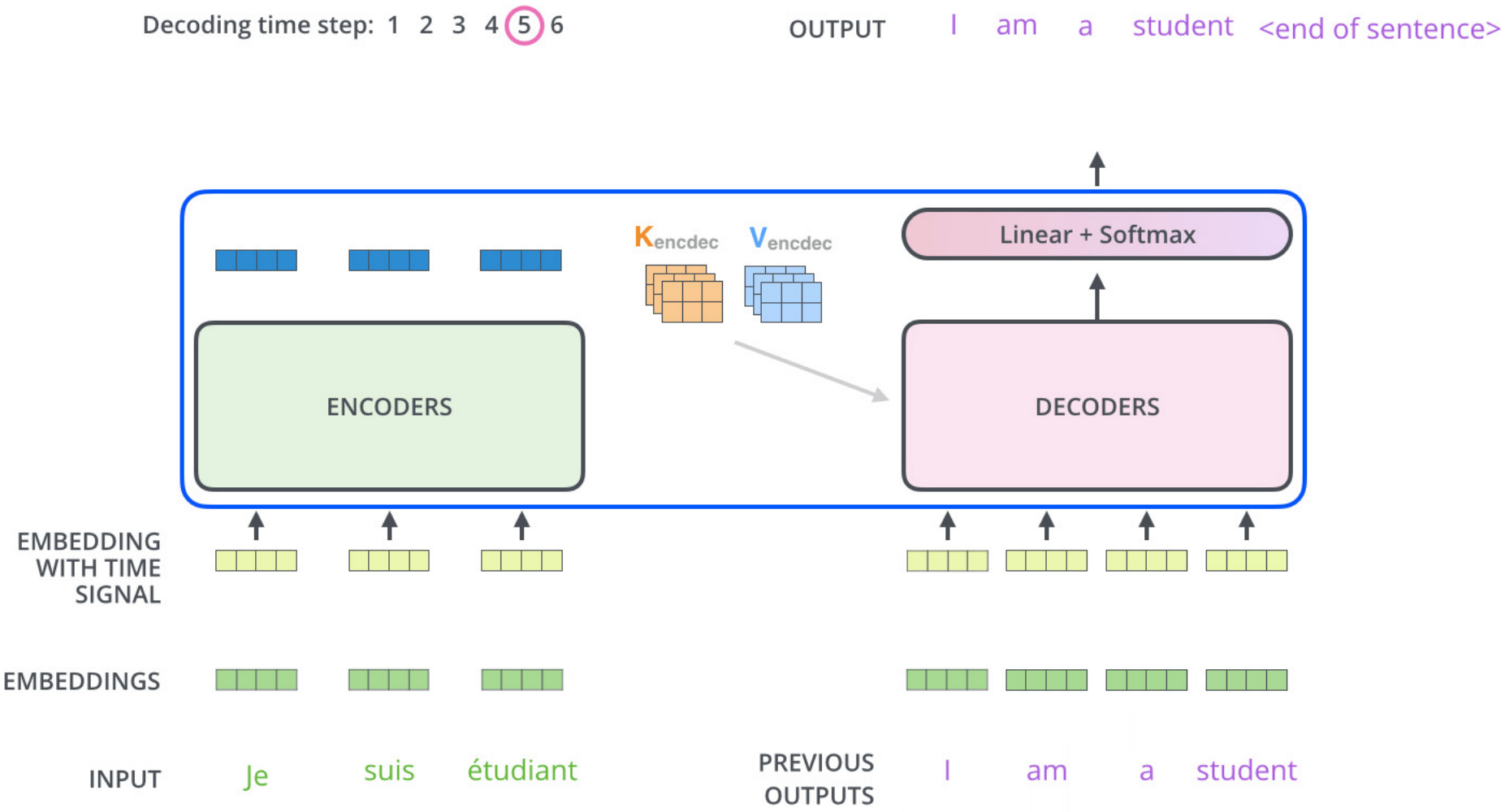


Image credit: https://jalammar.github.io/illustrated-transformer/

Transformers: Results on translation

Table : The Transformer achieves better BLEU scores than previous state-of-the-art models on the English-to-German and English-to-French newstest2014 tests at a fraction of the training cost.

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [38]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	**$3.3 \cdot 10^{18}$**	
Transformer (big)	**28.4**	**41.8**	$2.3 \cdot 10^{19}$	

Transformers - Positional Encoding

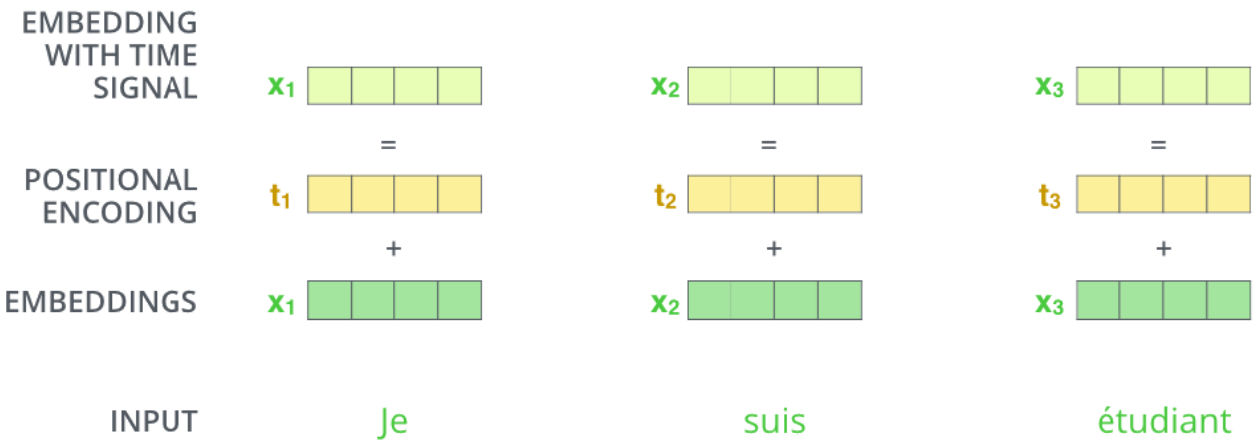


Image Source: The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time. (jalammar.github.io)

Exercise: Check the positional encoding scheme used in the paper and understand why the scheme was chosen.

Visual Transformer

AN IMAGE IS WORTH 16X16 WORDS: TRANSFORMERS FOR IMAGE RECOGNITION AT SCALE

Alexey Dosovitskiy[*,†], **Lucas Beyer**[*], **Alexander Kolesnikov**[*], **Dirk Weissenborn**[*], **Xiaohua Zhai**[*], **Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, Neil Houlsby**[*,†]

[*]equal technical contribution, [†]equal advising

Google Research, Brain Team

{adosovitskiy, neilhoulsby}@google.com

Visual Transformer

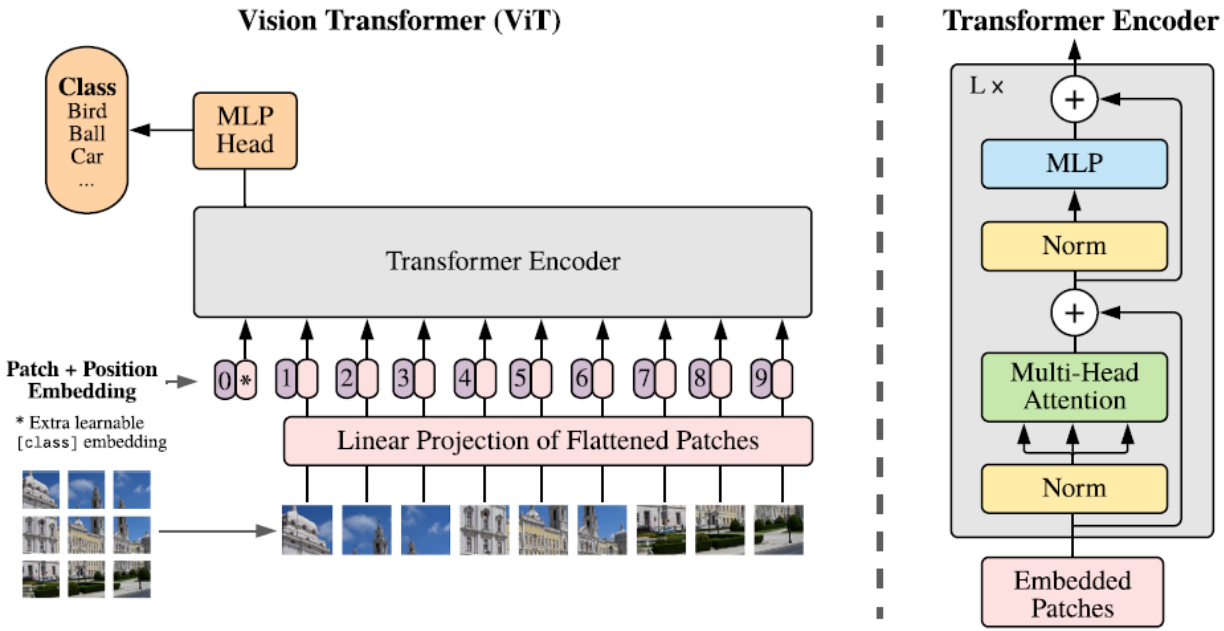


Figure 1: Model overview. We split an image into fixed-size patches, linearly embed each of them, add position embeddings, and feed the resulting sequence of vectors to a standard Transformer encoder. In order to perform classification, we use the standard approach of adding an extra learnable "classification token" to the sequence. The illustration of the Transformer encoder was inspired by Vaswani et al. (2017).

Visual Transformer

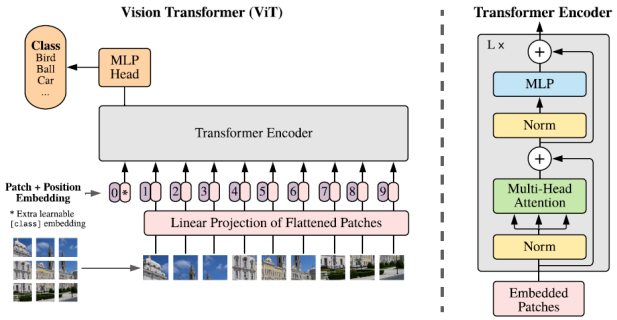


Figure 1: Model overview. We split an image into fixed-size patches, linearly embed each of them, add position embeddings, and feed the resulting sequence of vectors to a standard Transformer encoder. In order to perform classification, we use the standard approach of adding an extra learnable "classification token" to the sequence. The illustration of the Transformer encoder was inspired by Vaswani et al. (2017).

1. Split an image into (equal-sized) patches

Visual Transformer

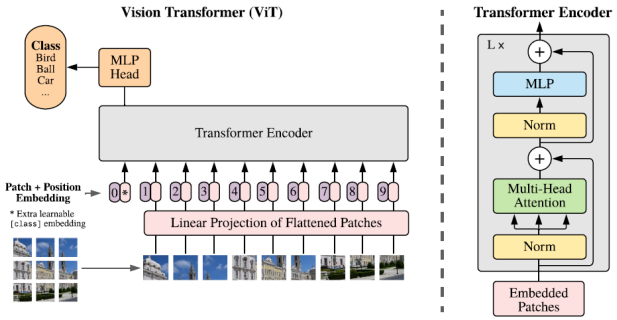


Figure 1: Model overview. We split an image into fixed-size patches, linearly embed each of them, add position embeddings, and feed the resulting sequence of vectors to a standard Transformer encoder. In order to perform classification, we use the standard approach of adding an extra learnable "classification token" to the sequence. The illustration of the Transformer encoder was inspired by Vaswani et al. (2017).

1. Split an image into (equal-sized) patches
2. Flatten the patches

Visual Transformer

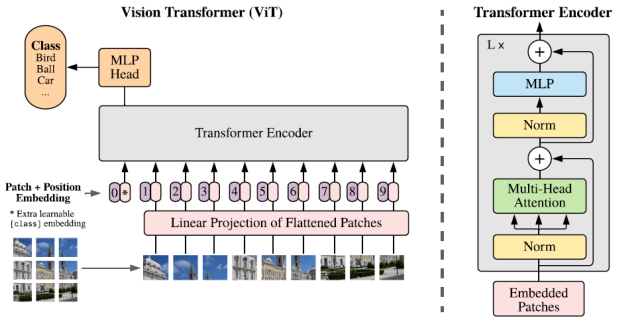


Figure 1: Model overview. We split an image into fixed-size patches, linearly embed each of them, add position embeddings, and feed the resulting sequence of vectors to a standard Transformer encoder. In order to perform classification, we use the standard approach of adding an extra learnable "classification token" to the sequence. The illustration of the Transformer encoder was inspired by Vaswani et al. (2017).

1. Split an image into (equal-sized) patches
2. Flatten the patches
3. Perform linear embedding of the flattened patches

Visual Transformer

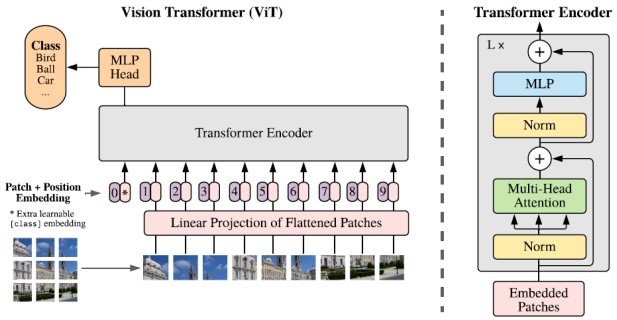


Figure 1: Model overview. We split an image into fixed-size patches, linearly embed each of them, add position embeddings, and feed the resulting sequence of vectors to a standard Transformer encoder. In order to perform classification, we use the standard approach of adding an extra learnable "classification token" to the sequence. The illustration of the Transformer encoder was inspired by Vaswani et al. (2017).

1. Split an image into (equal-sized) patches
2. Flatten the patches
3. Perform linear embedding of the flattened patches
4. Feed the sequence of linear embeddings (optionally added with positional encodings) to a transformer encoder

Visual Transformer

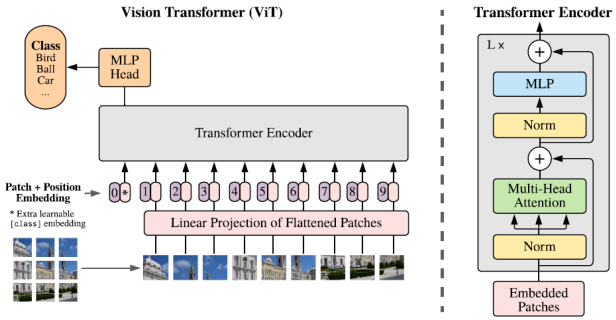


Figure 1: Model overview. We split an image into fixed-size patches, linearly embed each of them, add position embeddings, and feed the resulting sequence of vectors to a standard Transformer encoder. In order to perform classification, we use the standard approach of adding an extra learnable "classification token" to the sequence. The illustration of the Transformer encoder was inspired by Vaswani et al. (2017).

1. Split an image into (equal-sized) patches
2. Flatten the patches
3. Perform linear embedding of the flattened patches
4. Feed the sequence of linear embeddings (optionally added with positional encodings) to a transformer encoder
5. Pre-train the model with image labels (large data set with full supervision)

Visual Transformer

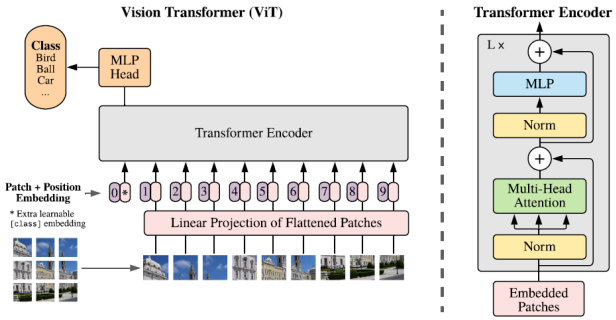


Figure 1: Model overview. We split an image into fixed-size patches, linearly embed each of them, add position embeddings, and feed the resulting sequence of vectors to a standard Transformer encoder. In order to perform classification, we use the standard approach of adding an extra learnable "classification token" to the sequence. The illustration of the Transformer encoder was inspired by Vaswani et al. (2017).

1. Split an image into (equal-sized) patches
2. Flatten the patches
3. Perform linear embedding of the flattened patches
4. Feed the sequence of linear embeddings (optionally added with positional encodings) to a transformer encoder
5. Pre-train the model with image labels (large data set with full supervision)
6. Fine-tune on a downstream classification task on a smaller data set.

Visual Transformer

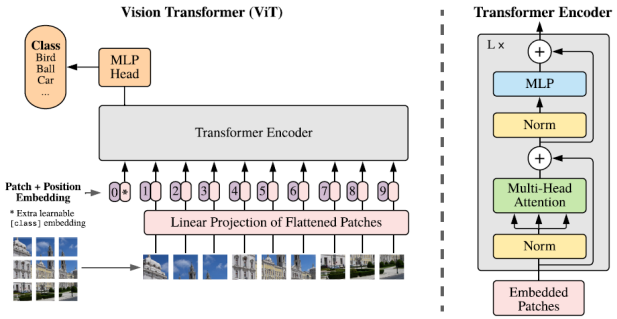


Figure 1: Model overview. We split an image into fixed-size patches, linearly embed each of them, add position embeddings, and feed the resulting sequence of vectors to a standard Transformer encoder. In order to perform classification, we use the standard approach of adding an extra learnable "classification token" to the sequence. The illustration of the Transformer encoder was inspired by Vaswani et al. (2017).

- Input: 2D image $\mathbf{x} \in \mathbb{R}^{H \times W \times C}$ where (H, W) is the resolution of image in terms of height and width, and C denotes the number of channels.

Visual Transformer

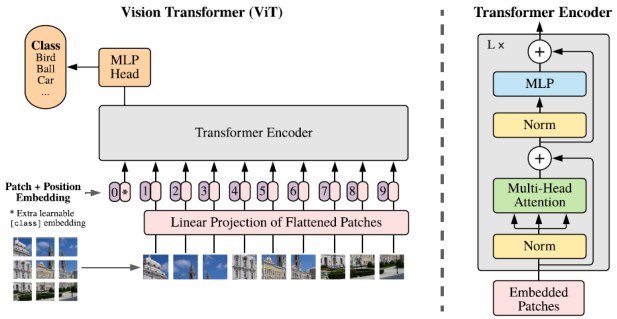


Figure 1: Model overview. We split an image into fixed-size patches, linearly embed each of them, add position embeddings, and feed the resulting sequence of vectors to a standard Transformer encoder. In order to perform classification, we use the standard approach of adding an extra learnable "classification token" to the sequence. The illustration of the Transformer encoder was inspired by Vaswani et al. (2017).

- Input: 2D image $\mathbf{x} \in \mathbb{R}^{H \times W \times C}$ where (H, W) is the resolution of image in terms of height and width, and C denotes the number of channels.
- Input: 2D patch size P. Hence each patch is square of shape (P, P).

Visual Transformer

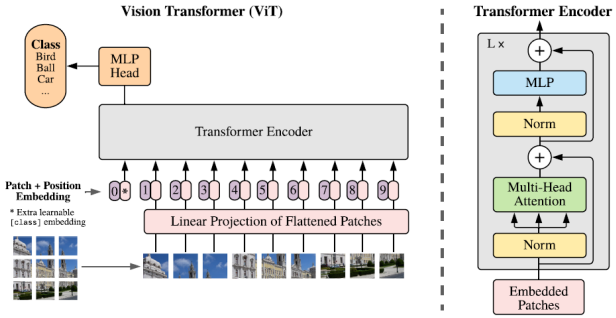


Figure 1: Model overview. We split an image into fixed-size patches, linearly embed each of them, add position embeddings, and feed the resulting sequence of vectors to a standard Transformer encoder. In order to perform classification, we use the standard approach of adding an extra learnable "classification token" to the sequence. The illustration of the Transformer encoder was inspired by Vaswani et al. (2017).

- Input: 2D image $\mathbf{x} \in \mathbb{R}^{H \times W \times C}$ where (H, W) is the resolution of image in terms of height and width, and C denotes the number of channels.
- Input: 2D patch size P. Hence each patch is square of shape (P, P).
- Output: N patches denoted x_p, where j-th patch $x_p^j \in \mathbb{R}^{P \times P \times C}$.

Visual Transformer

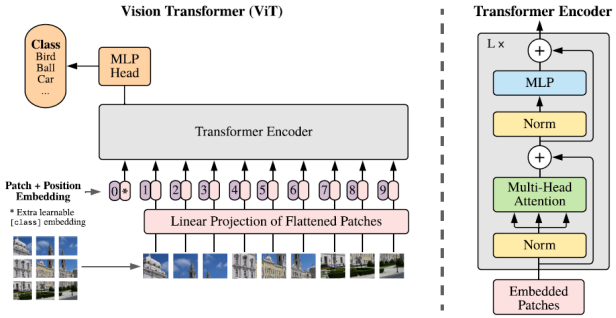


Figure 1: Model overview. We split an image into fixed-size patches, linearly embed each of them, add position embeddings, and feed the resulting sequence of vectors to a standard Transformer encoder. In order to perform classification, we use the standard approach of adding an extra learnable "classification token" to the sequence. The illustration of the Transformer encoder was inspired by Vaswani et al. (2017).

- Input: 2D image $\mathbf{x} \in \mathbb{R}^{H \times W \times C}$ where (H, W) is the resolution of image in terms of height and width, and C denotes the number of channels.
- Input: 2D patch size P. Hence each patch is square of shape (P, P).
- Output: N patches denoted x_p, where j-th patch $x_p^j \in \mathbb{R}^{P \times P \times C}$.
- (Note: $N = HW/P^2$, hence we assume that both H and W are divisible by P)

Visual Transformer

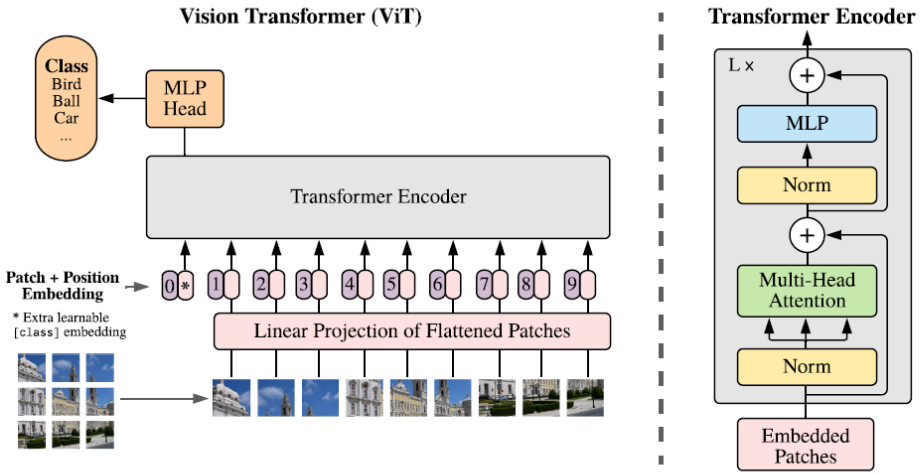


Figure 1: Model overview. We split an image into fixed-size patches, linearly embed each of them, add position embeddings, and feed the resulting sequence of vectors to a standard Transformer encoder. In order to perform classification, we use the standard approach of adding an extra learnable "classification token" to the sequence. The illustration of the Transformer encoder was inspired by Vaswani et al. (2017).

- A learnable embedding z_0^0 is attached to sequence of embeddings at 0-th encoder block.
- The state z_L^0 at L-th block serves as representation of class label y.
- During pre-training and fine-tuning, a classification head is attached to z_L^0.
- Classification head is a MLP with single hidden layer at pre-training and with no hidden layer at fine-tuning.

Visual Transformer

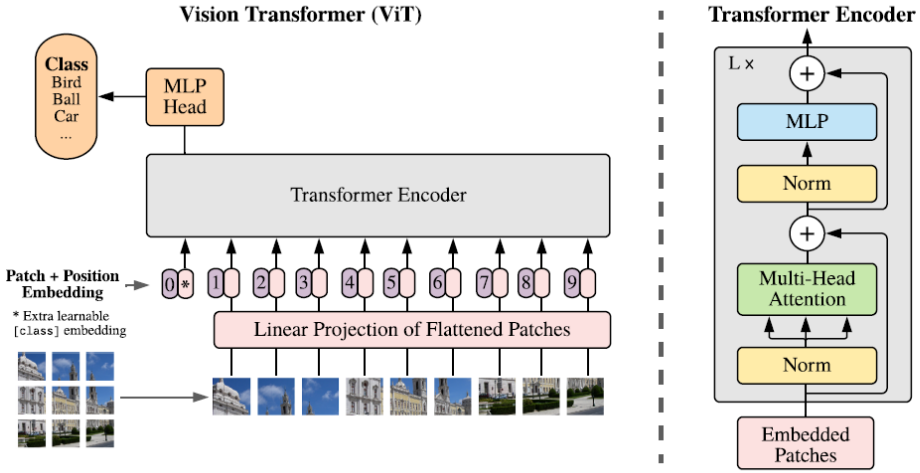


Figure 1: Model overview. We split an image into fixed-size patches, linearly embed each of them, add position embeddings, and feed the resulting sequence of vectors to a standard Transformer encoder. In order to perform classification, we use the standard approach of adding an extra learnable "classification token" to the sequence. The illustration of the Transformer encoder was inspired by Vaswani et al. (2017).

- 1D positional encoding used.
- 2D positional encoding did not give much advantage.

Visual Transformer

$$\mathbf{z}_0 = [\mathbf{x}_{\text{class}};\ \mathbf{x}_p^1\mathbf{E};\ \mathbf{x}_p^2\mathbf{E};\ \cdots\ ;\ \mathbf{x}_p^N\mathbf{E}] + \mathbf{E}_{pos}, \qquad \mathbf{E} \in \mathbb{R}^{(P^2 \cdot C) \times D},\ \mathbf{E}_{pos} \in \mathbb{R}^{(N+1) \times D}$$

$$\mathbf{z}'_\ell = \text{MSA}(\text{LN}(\mathbf{z}_{\ell-1})) + \mathbf{z}_{\ell-1}, \qquad \ell = 1 \ldots L$$

$$\mathbf{z}_\ell = \text{MLP}(\text{LN}(\mathbf{z}'_\ell)) + \mathbf{z}'_\ell, \qquad \ell = 1 \ldots L$$

$$\mathbf{y} = \text{LN}(\mathbf{z}_L^0)$$

Visual Transformer

$$\mathbf{z}_0 = [\mathbf{x}_{\text{class}}; \mathbf{x}_p^1\mathbf{E}; \mathbf{x}_p^2\mathbf{E}; \cdots; \mathbf{x}_p^N\mathbf{E}] + \mathbf{E}_{pos}, \qquad \mathbf{E} \in \mathbb{R}^{(P^2 \cdot C) \times D}, \ \mathbf{E}_{pos} \in \mathbb{R}^{(N+1) \times D}$$

$$\mathbf{z}'_\ell = \text{MSA}(\text{LN}(\mathbf{z}_{\ell-1})) + \mathbf{z}_{\ell-1}, \qquad \ell = 1 \ldots L$$

$$\mathbf{z}_\ell = \text{MLP}(\text{LN}(\mathbf{z}'_\ell)) + \mathbf{z}'_\ell, \qquad \ell = 1 \ldots L$$

$$\mathbf{y} = \text{LN}(\mathbf{z}_L^0)$$

Layer Normalization:

$$\mathbf{h}^t = f\left[\frac{\mathbf{g}}{\sigma^t} \odot (\mathbf{a}^t - \mu^t) + \mathbf{b}\right] \qquad \mu^t = \frac{1}{H}\sum_{i=1}^{H} a_i^t \qquad \sigma^t = \sqrt{\frac{1}{H}\sum_{i=1}^{H}(a_i^t - \mu^t)^2}$$

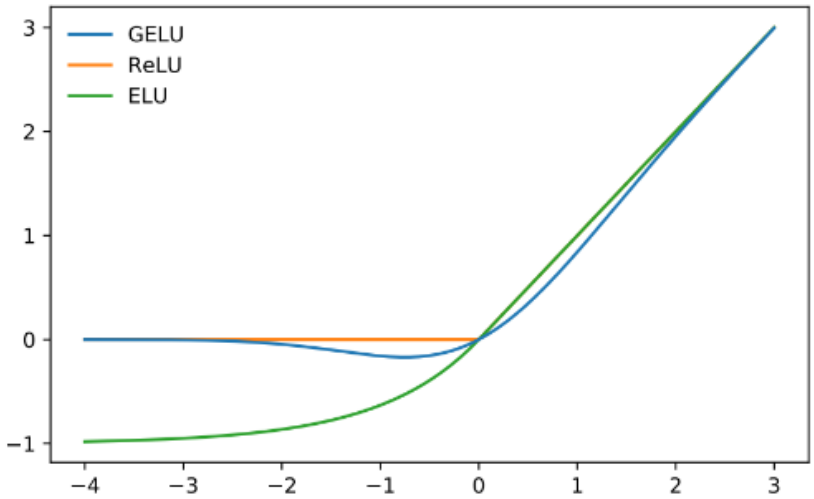
J. L. Ba, J. R. Kiros and G. E. Hinton. Layer normalization.
https://arxiv.org/pdf/1607.06450.pdf

Visual Transformer

$$
\begin{aligned}
\mathbf{z}_0 &= [\mathbf{x}_{\text{class}};\, \mathbf{x}_p^1\mathbf{E};\, \mathbf{x}_p^2\mathbf{E};\, \cdots;\, \mathbf{x}_p^N\mathbf{E}] + \mathbf{E}_{pos}, && \mathbf{E} \in \mathbb{R}^{(P^2 \cdot C) \times D},\ \mathbf{E}_{pos} \in \mathbb{R}^{(N+1) \times D} \\
\mathbf{z}'_\ell &= \text{MSA}(\text{LN}(\mathbf{z}_{\ell-1})) + \mathbf{z}_{\ell-1}, && \ell = 1 \ldots L \\
\mathbf{z}_\ell &= \text{MLP}(\text{LN}(\mathbf{z}'_\ell)) + \mathbf{z}'_\ell, && \ell = 1 \ldots L \\
\mathbf{y} &= \text{LN}(\mathbf{z}_L^0)
\end{aligned}
$$

- MLP uses two layers with GeLU non-linearity.
- $GeLU(x) = xP(X \leq x)$ for a random variable X. Typically $X \sim \mathcal{N}(0, 1)$.

Visual Transformer



- $ReLU(x) = \max\{0, x\}$.
- $ELU(x; \alpha) = \begin{cases} z & \text{if } z > 0 \\ \alpha(e^z - 1) & \text{if } z \leq 0 \end{cases}$
- $GeLU(x) = x\Phi(x)$ where $\Phi(x)$ is the cdf associated with $\mathcal{N}(0, 1)$.

Visual Transformer

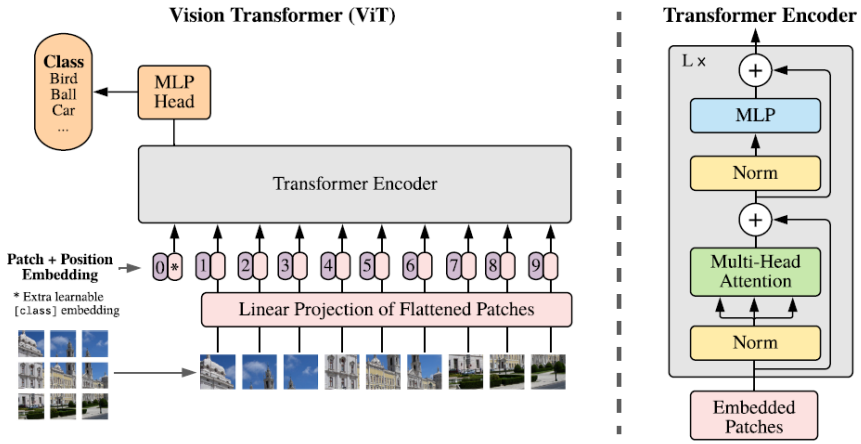


Figure 1: Model overview. We split an image into fixed-size patches, linearly embed each of them, add position embeddings, and feed the resulting sequence of vectors to a standard Transformer encoder. In order to perform classification, we use the standard approach of adding an extra learnable "classification token" to the sequence. The illustration of the Transformer encoder was inspired by Vaswani et al. (2017).

- Inductive bias in convolutions due to local two-dimensional neighborhood structure, and translation equivariance **not available** in visual transformer.
- MLP might be useful to capture local information.
- Self-attention is generally global.
- Positional encodings do not capture 2D positional information, hence spatial relations should be learned from scratch.

Visual Transformer

Model	Layers	Hidden size D	MLP size	Heads	Params
ViT-Base	12	768	3072	12	86M
ViT-Large	24	1024	4096	16	307M
ViT-Huge	32	1280	5120	16	632M

Table 1: Details of Vision Transformer model variants.

Visual Transformer

	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).

Note: JFT data set is an inhouse data set of Google with more than billion images.

Visual Transformer

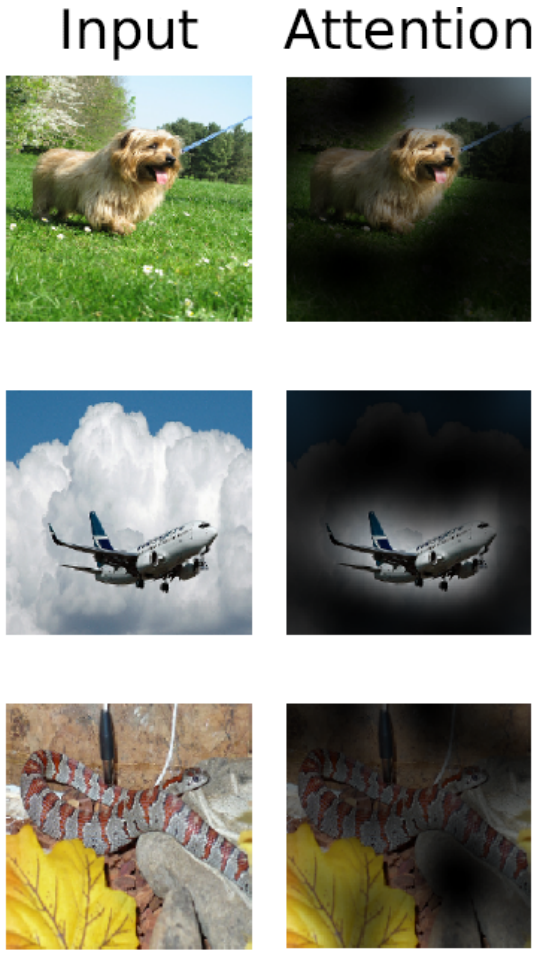


Figure : Representative examples of attention from the output token to the input space.