UNIVERSITY OF TECHNOLOGY
IN THE EUROPEAN CAPITAL OF CULTURE
CHEMNITZ

Neurocomputing

Transformers

Julien Vitay
Professur für Künstliche Intelligenz - Fakultät für Informatik

1 - 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

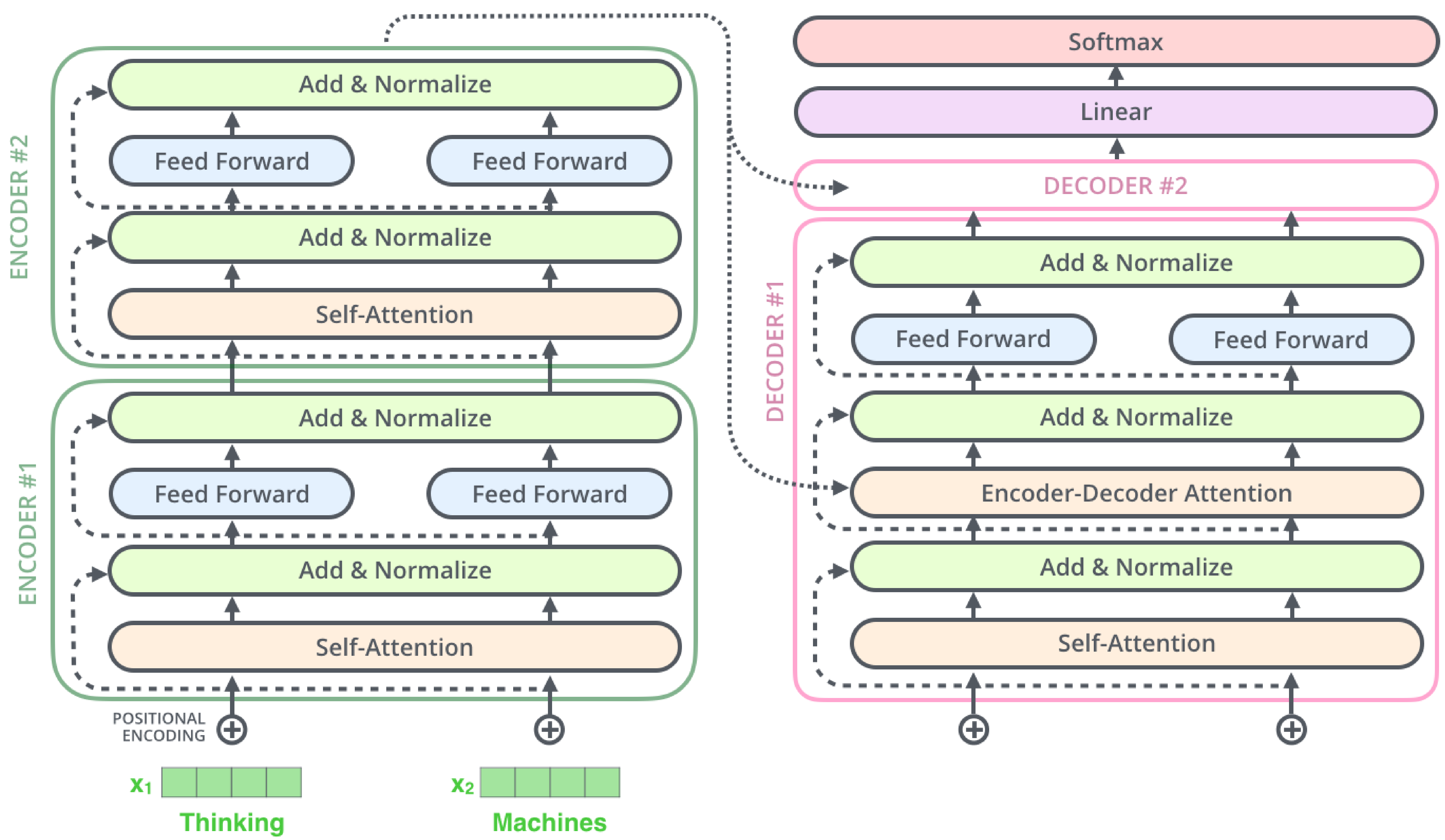
Łukasz Kaiser[*]
Google Brain
lukaszkaiser@google.com

Illia Polosukhin[*] [‡]
illia.polosukhin@gmail.com

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., et al. (2017). **Attention Is All You Need**. arXiv:1706.03762.

Transformer networks

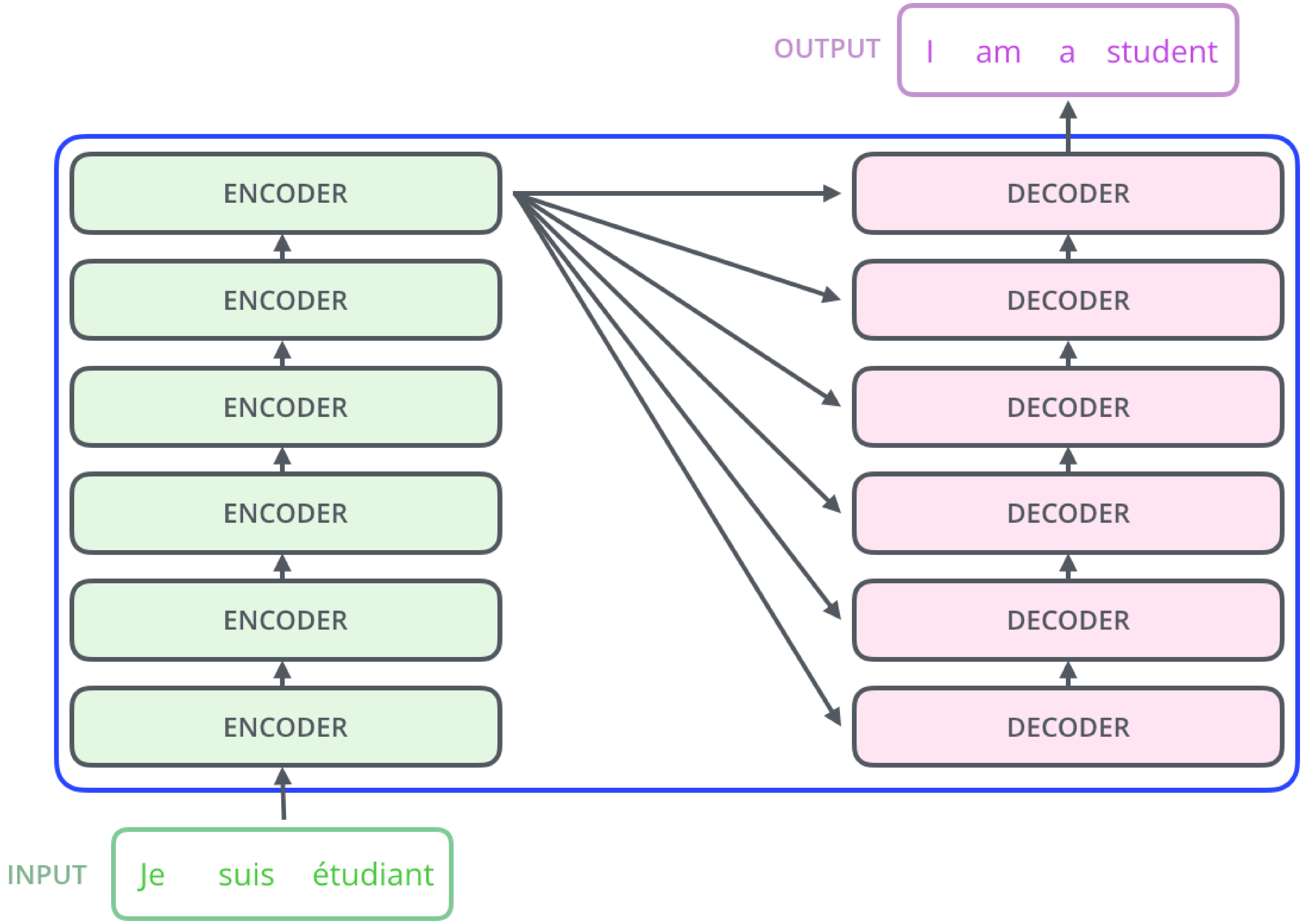
- Attentional mechanisms are so powerful that recurrent networks are not even needed anymore.
- **Transformer networks** use **self-attention** in a purely feedforward architecture and outperform recurrent architectures.
- Used in Google BERT and OpenAI GPT-3 for text understanding (e.g. search engine queries) and generation.



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

Transformer networks

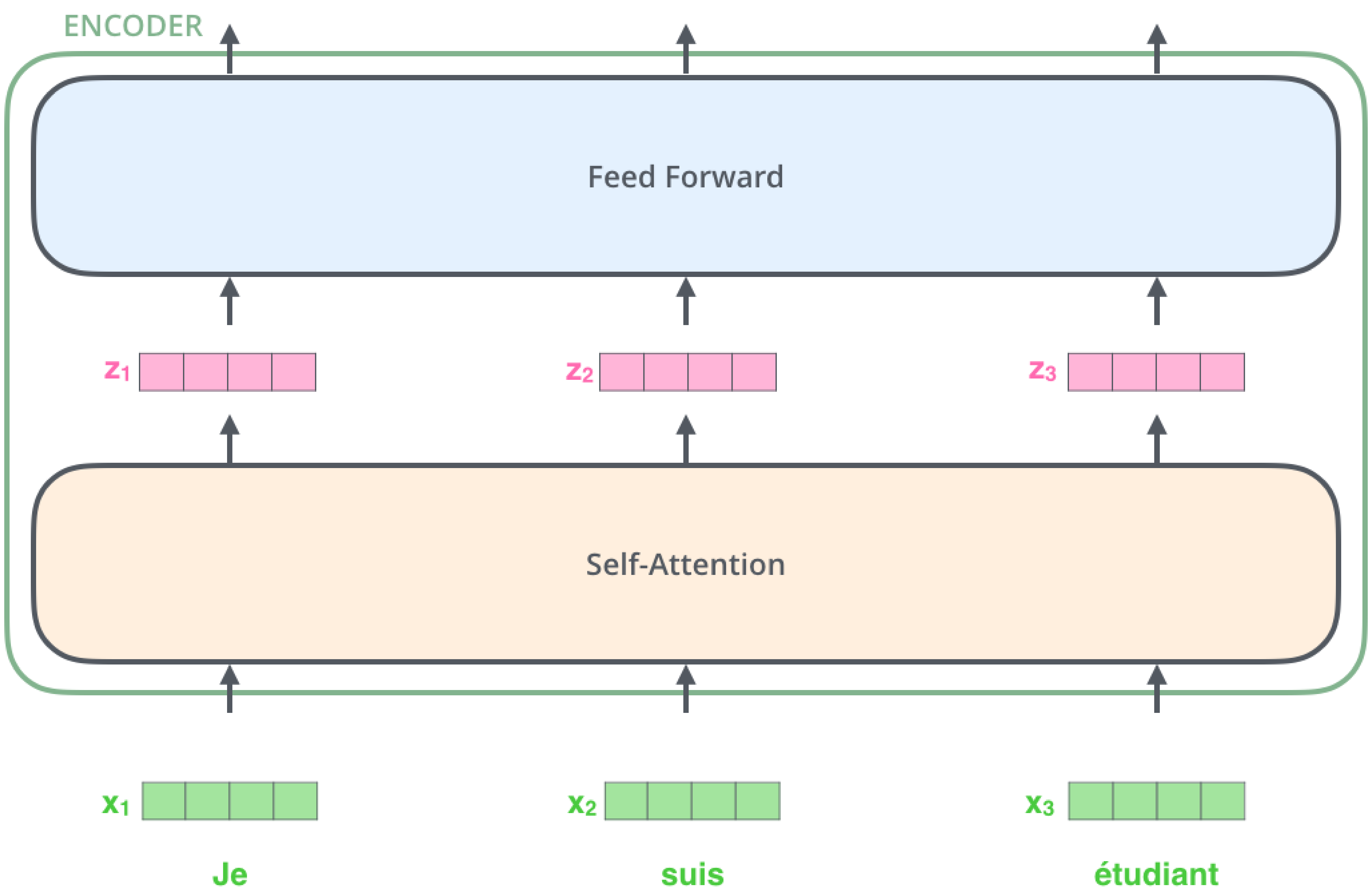
- Transformer networks use an **encoder-decoder** architecture, each with 6 stacked layers.



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

Encoder layer

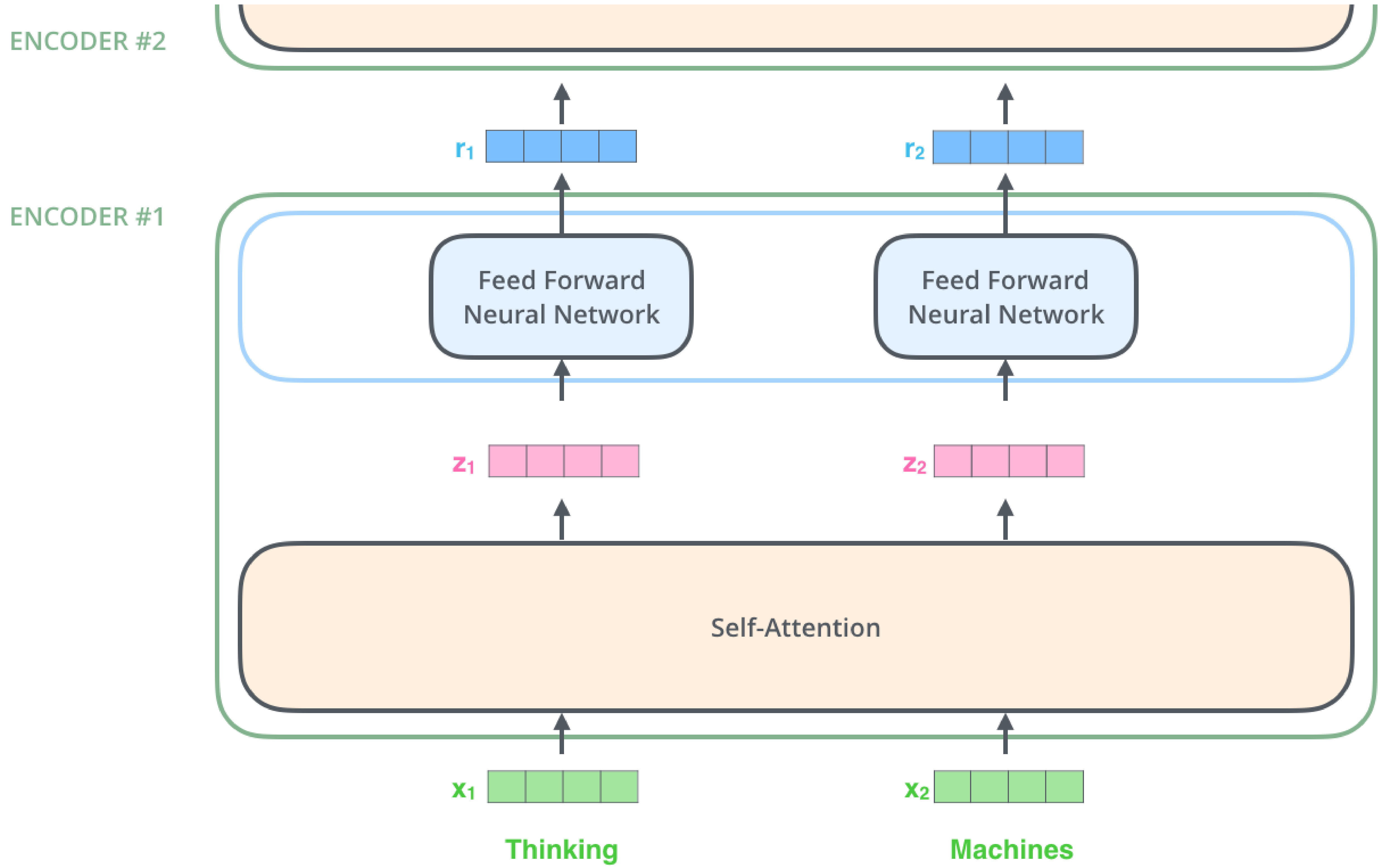
- Each layer of the encoder processes the n words of the input sentence **in parallel**.
- Word embeddings (as in word2vec) of dimension 512 are used as inputs (but learned end-to-end).



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

Encoder layer

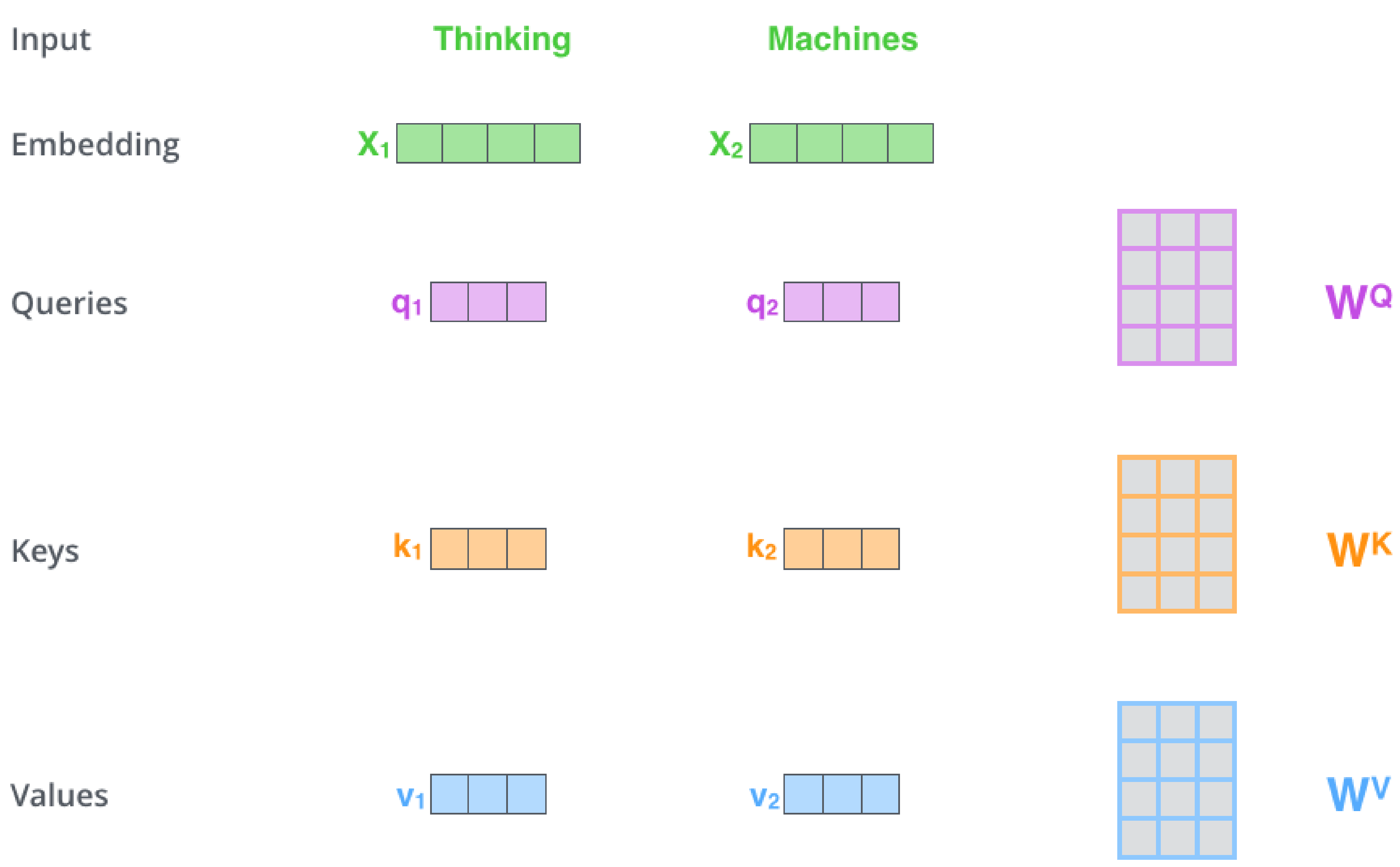
- Two operations are performed on each word embedding $\mathbf{x}_i$:
  - self-attention vector $\mathbf{z}_i$ depending on the other words.
  - a regular feedforward layer to obtain a new representation $\mathbf{r}_i$ (shared among all words).



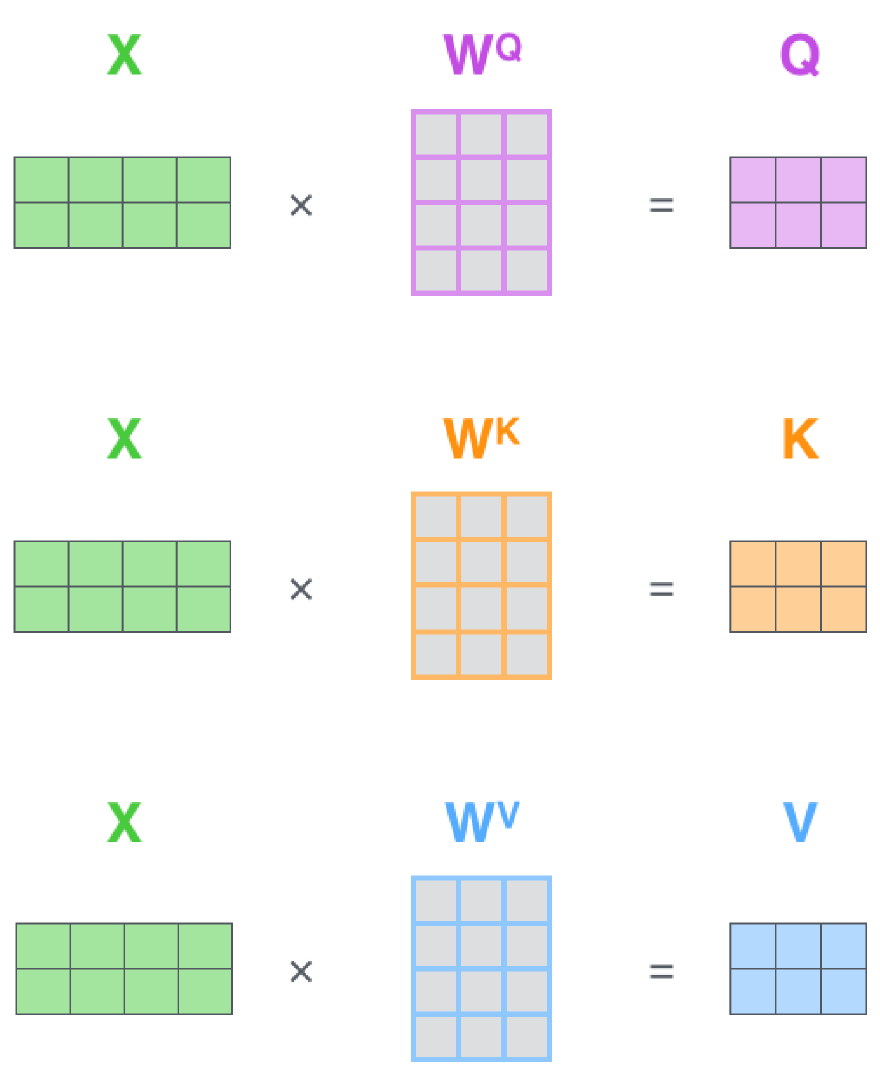
Source: http://jalammar.github.io/illustrated-transformer/

Self-attention

- The first step of self-attention is to compute for each word three vectors of length $d_k = 64$ from the embeddings $\mathbf{x}_i$ or previous representations $\mathbf{r}_i$ (d = 512).
  - The **query** $\mathbf{q}_i$ using W^Q.
  - The **key** $\mathbf{k}_i$ using W^K.
  - The **value** $\mathbf{v}_i$ using W^V.



- This operation can be done in parallel over all words:



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

Self-attention

- Why query / key / value? This a concept inspired from recommendation systems / databases.
- A Python dictionary is a set of key / value entries:

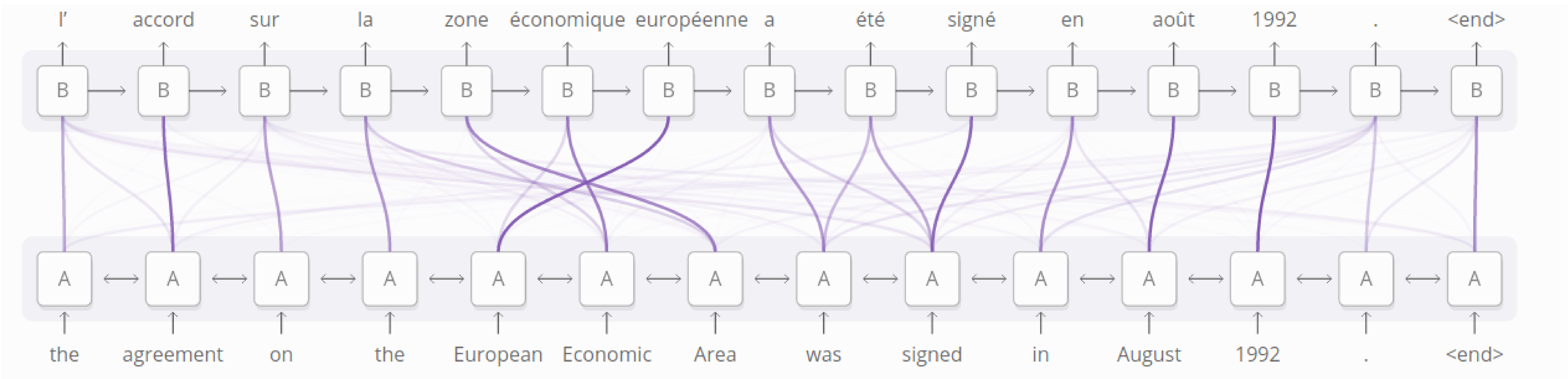
```
tel = {
    'jack': 4098,
    'sape': 4139
}
```

- The query would ask the dictionary to iterate over all entries and return the value associated to the key **equal or close to** the query.

```
tel['jacky'] # 4098
```

- This would be some sort of **fuzzy** dictionary.

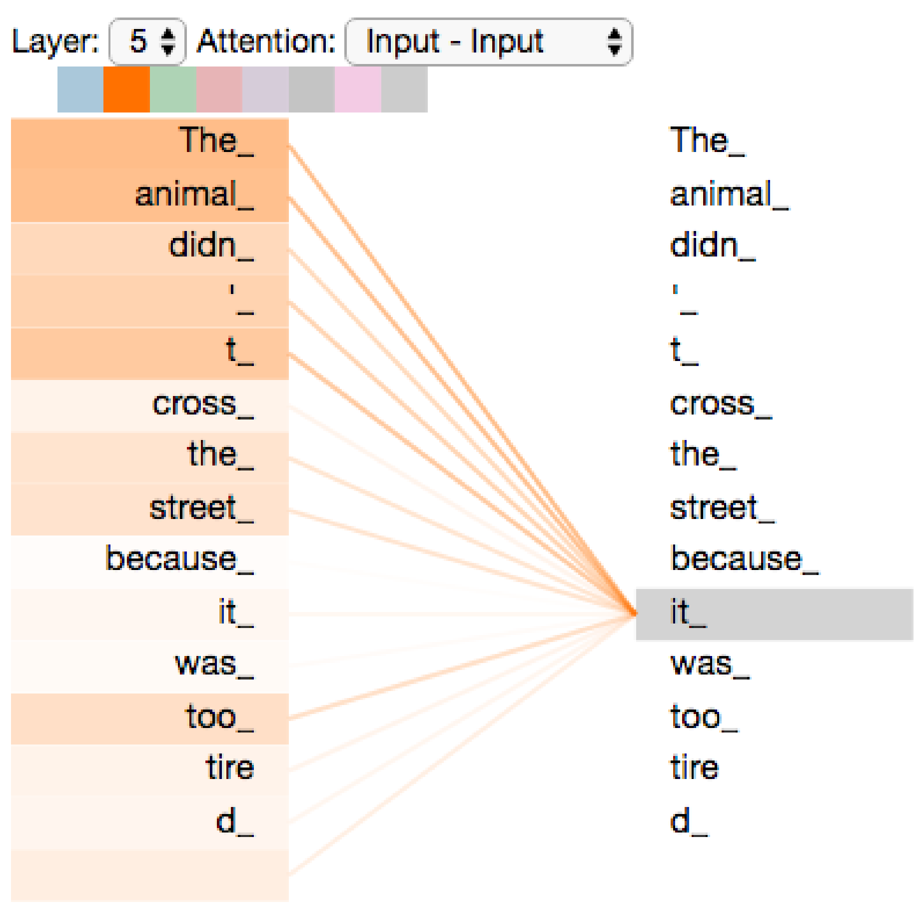
Self-attention



- In attentional RNNs, the attention scores were used by each word generated by the decoder to decide which **input word** is relevant.
- If we apply the same idea to the **same sentence** (self-attention), the attention score tells how much words of the same sentence are related to each other (context).

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

- The goal is to learn a representation for the word `it` that contains information about `the animal`, not `the street`.



Dzmitry Bahdanau, Kyunghyun Cho, Yoshua Bengio (2014). Neural Machine Translation by Jointly Learning to Align and Translate. arXiv:1409.0473

Self-attention

- Each word $\mathbf{x}_i$ of the sentence generates its query $\mathbf{q}_i$, key $\mathbf{k}_i$ and value $\mathbf{v}_i$.

- For all other words $\mathbf{x}_j$, we compute the **match** between the query $\mathbf{q}_i$ and the keys $\mathbf{k}_j$ with a dot product:

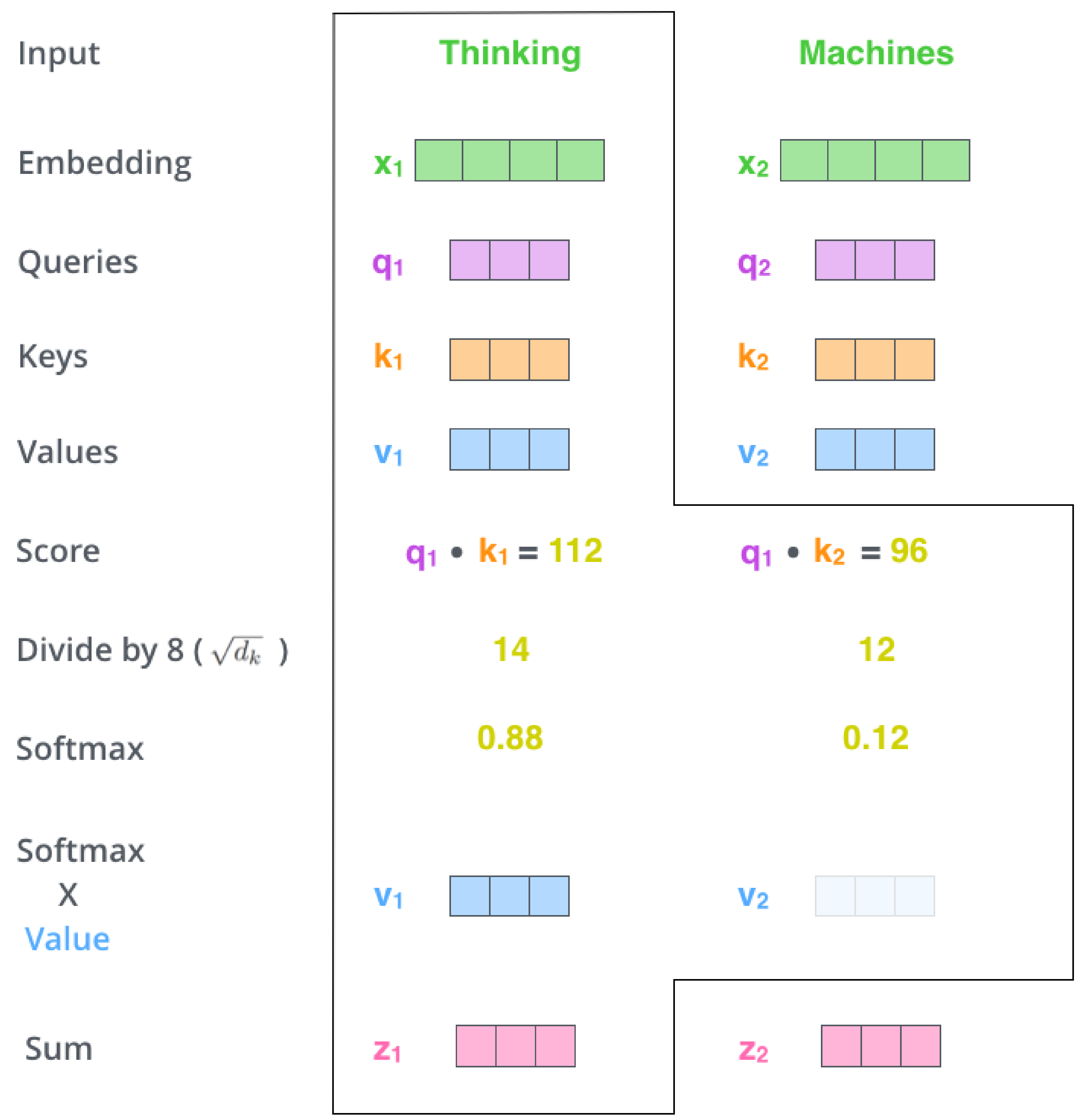
$$e_{i,j} = \mathbf{q}_i^T \, \mathbf{k}_j$$

- We normalize the scores by dividing by $\sqrt{d_k} = 8$ and apply a softmax:

$$a_{i,j} = \text{softmax}(\frac{\mathbf{q}_i^T \, \mathbf{k}_j}{\sqrt{d_k}})$$

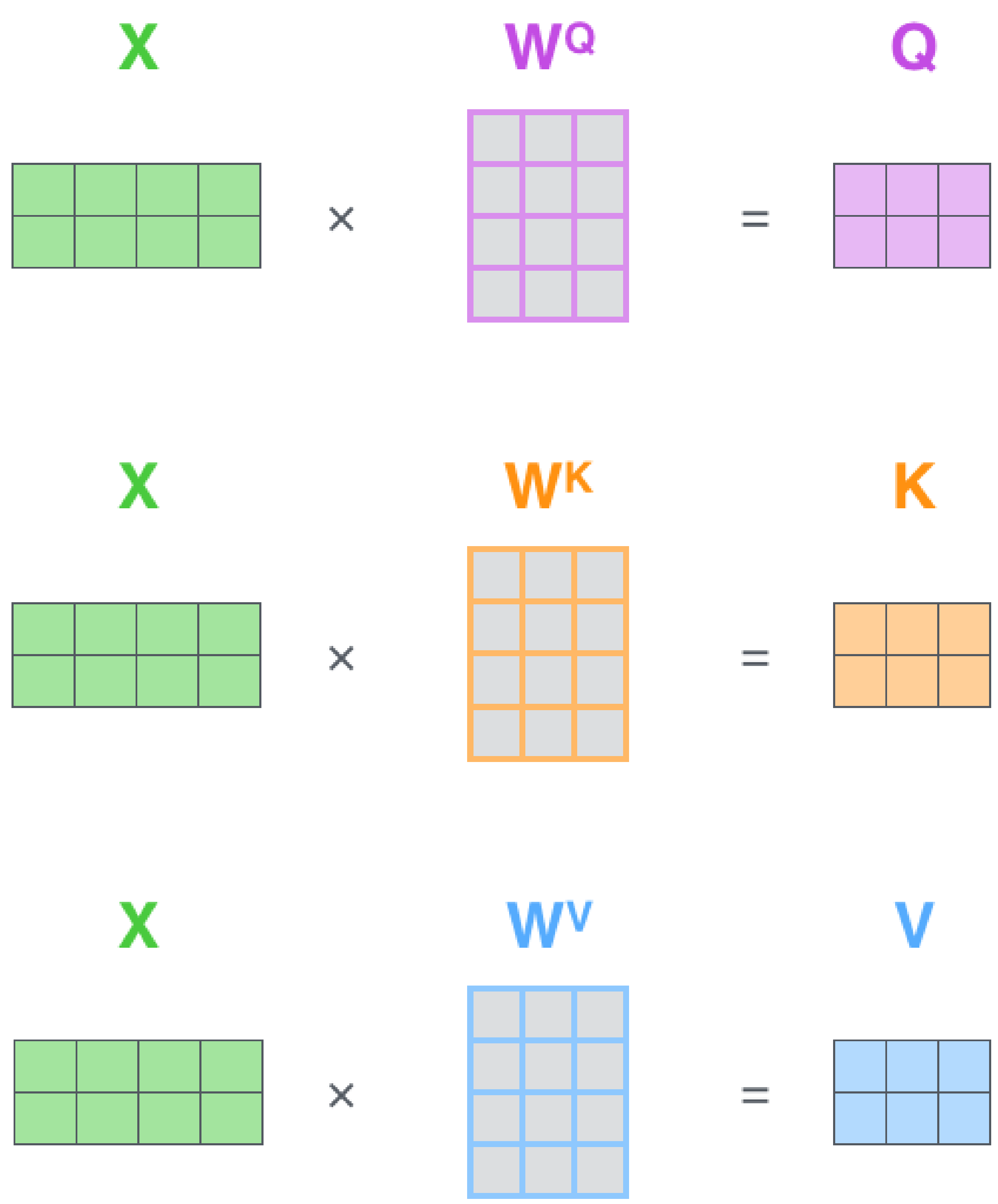
- The new representation $\mathbf{z}_i$ of the word $\mathbf{x}_i$ is a weighted sum of the values of all other words, weighted by the attention score:

$$\mathbf{z}_i = \sum_j a_{i,j} \, \mathbf{v}_j$$



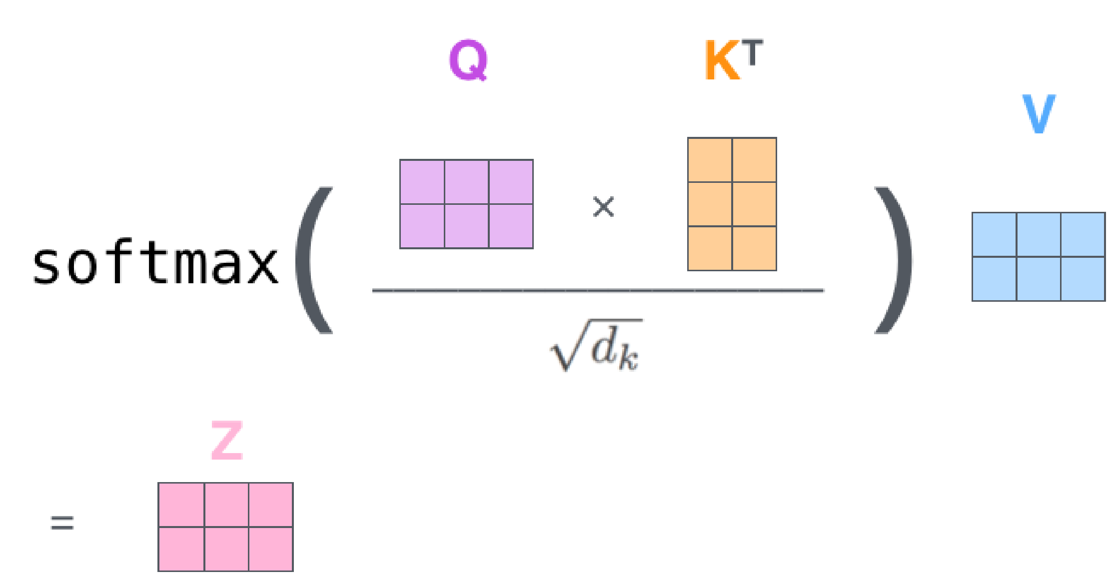
Source: http://jalammar.github.io/illustrated-transformer/

Self-attention



- If we concatenate the word embeddings into a $n \times d$ matrix X, self-attention only implies matrix multiplications and a row-based softmax:

$$
\begin{cases}
Q = X \times W^Q \\
K = X \times W^K \\
V = X \times W^V \\
Z = \text{softmax}(\dfrac{Q \times K^T}{\sqrt{d_k}}) \times V
\end{cases}
$$

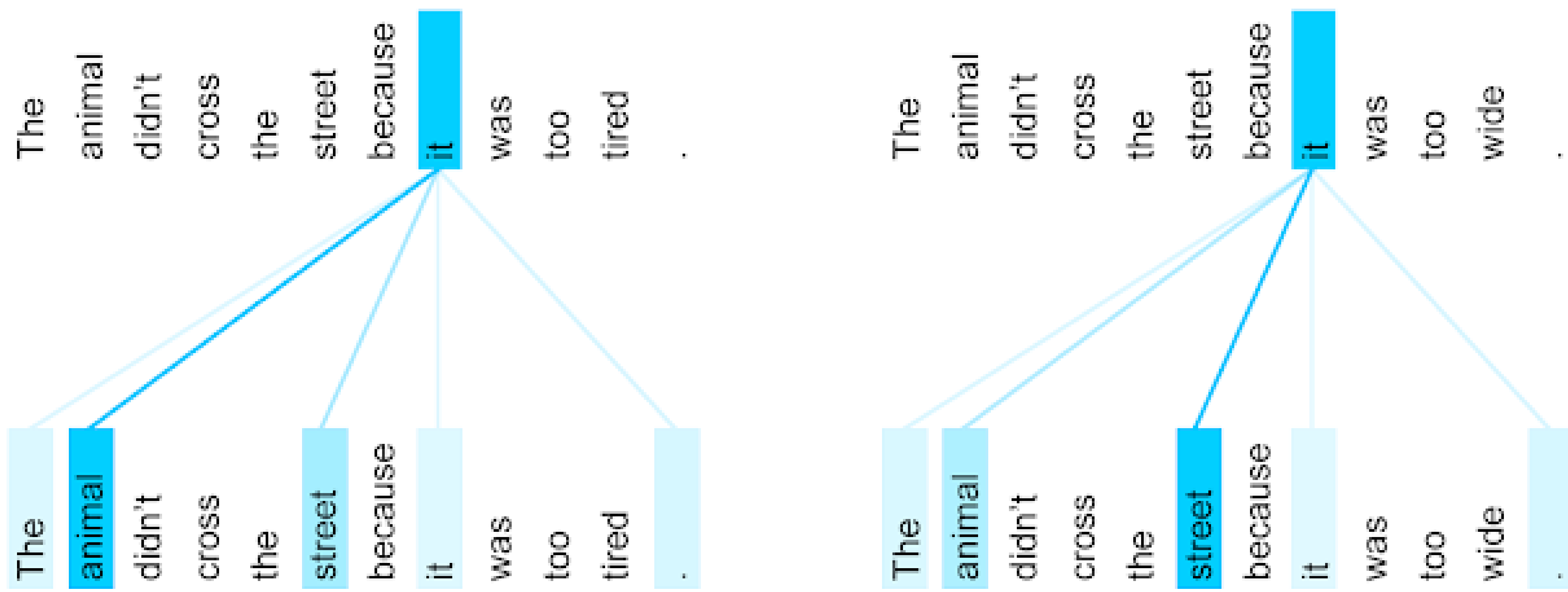


Source: http://jalammar.github.io/illustrated-transformer/

- Note 1: everything is differentiable, backpropagation will work.
- Note 2: the weight matrices do not depend on the length n of the sentence.

Multi-headed self-attention

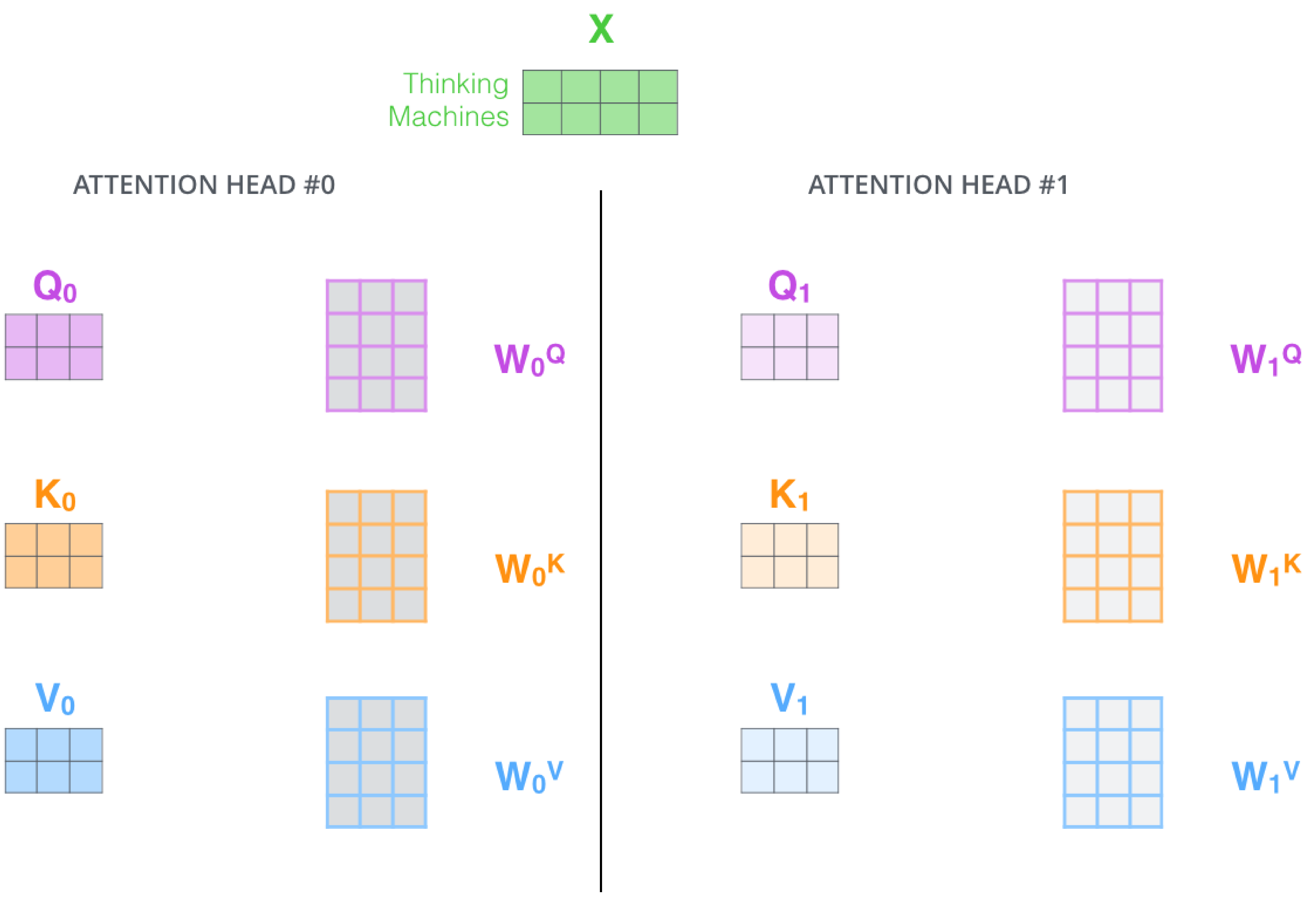
- In the sentence *The animal didn't cross the street because it was too tired.*, the new representation for the word `it` will hopefully contain features of the word `animal` after training.
- But what if the sentence was *The animal didn't cross the street because it was too **wide**.*? The representation of `it` should be linked to `street` in that context.
- This is not possible with a single set of matrices W^Q, W^K and W^V, as they would average every possible context and end up being useless.



Source: https://ai.googleblog.com/2017/08/transformer-novel-neural-network.html

Multi-headed self-attention

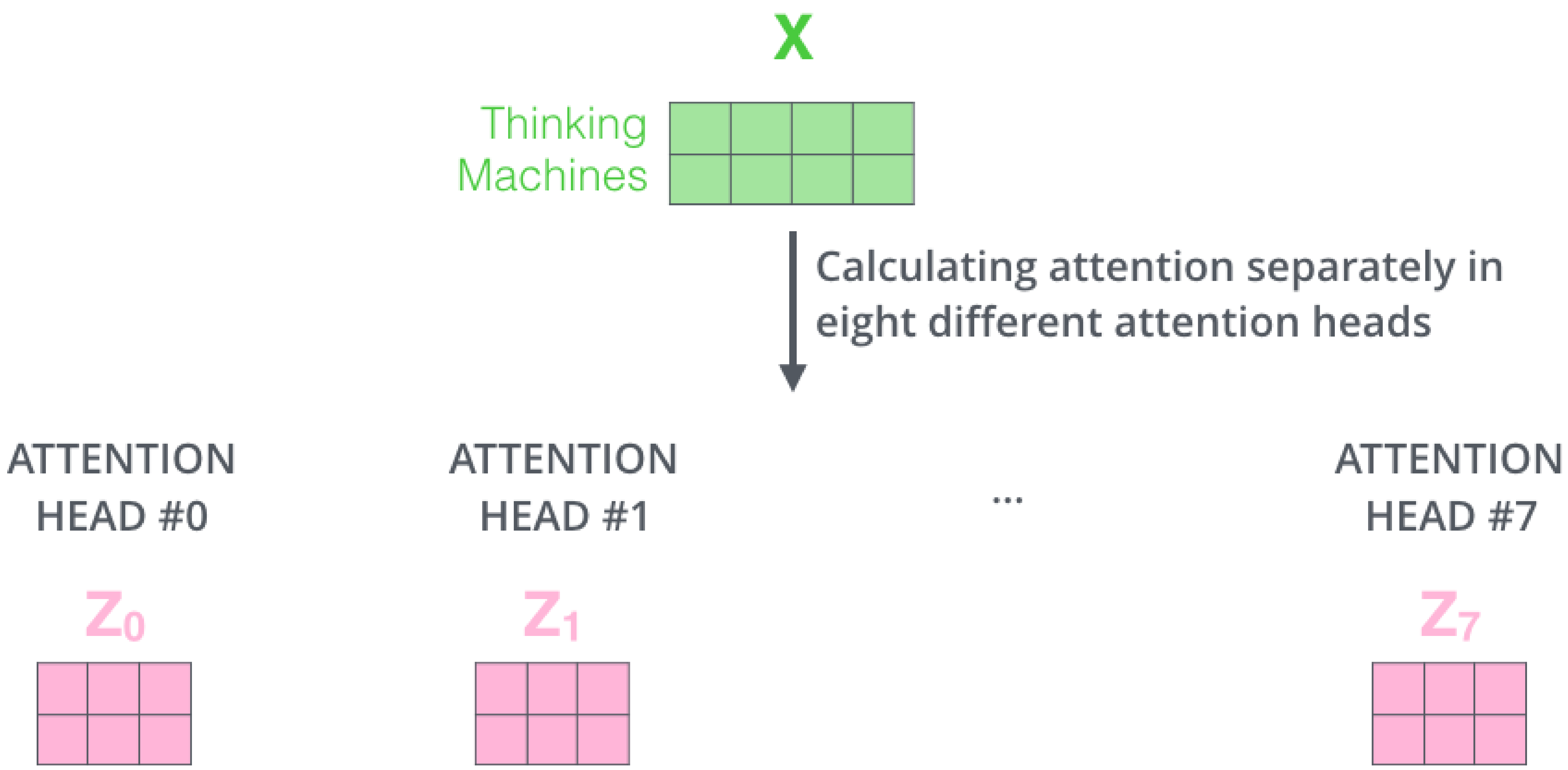
- The solution is to use **multiple attention heads** ($h=8$) with their own matrices W_k^Q, W_k^K and W_k^V.



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

Multi-headed self-attention

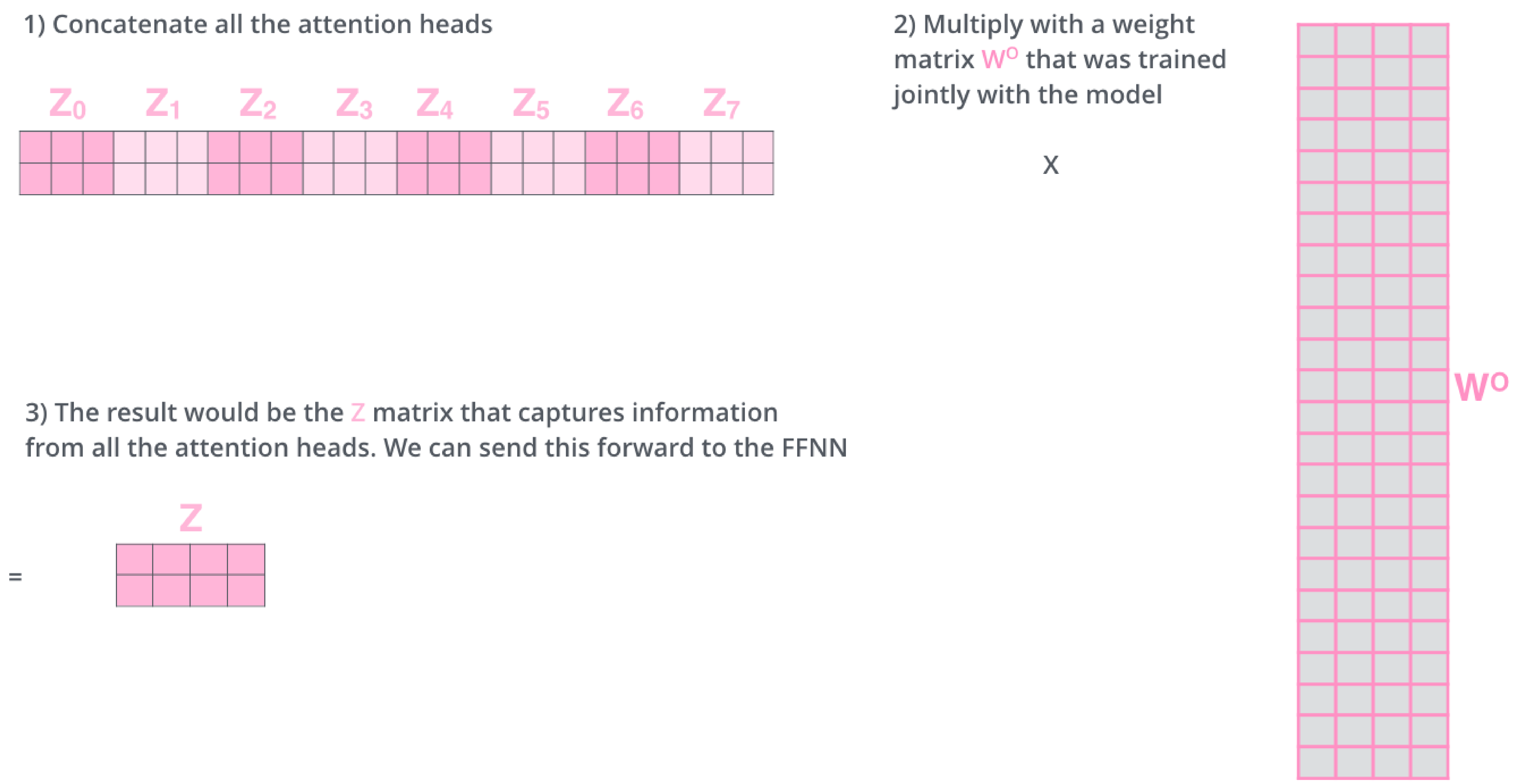
- Each **attention head** will output a vector $\mathbf{z}_i$ of size $d_k = 64$ for each word.
- How do we combine them?



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

Multi-headed self-attention

- The proposed solution is based on **ensemble learning** (stacking):
  - let another matrix W^O decide which attention head to trust...
  - 8×64 rows, 512 columns.



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

Multi-headed self-attention

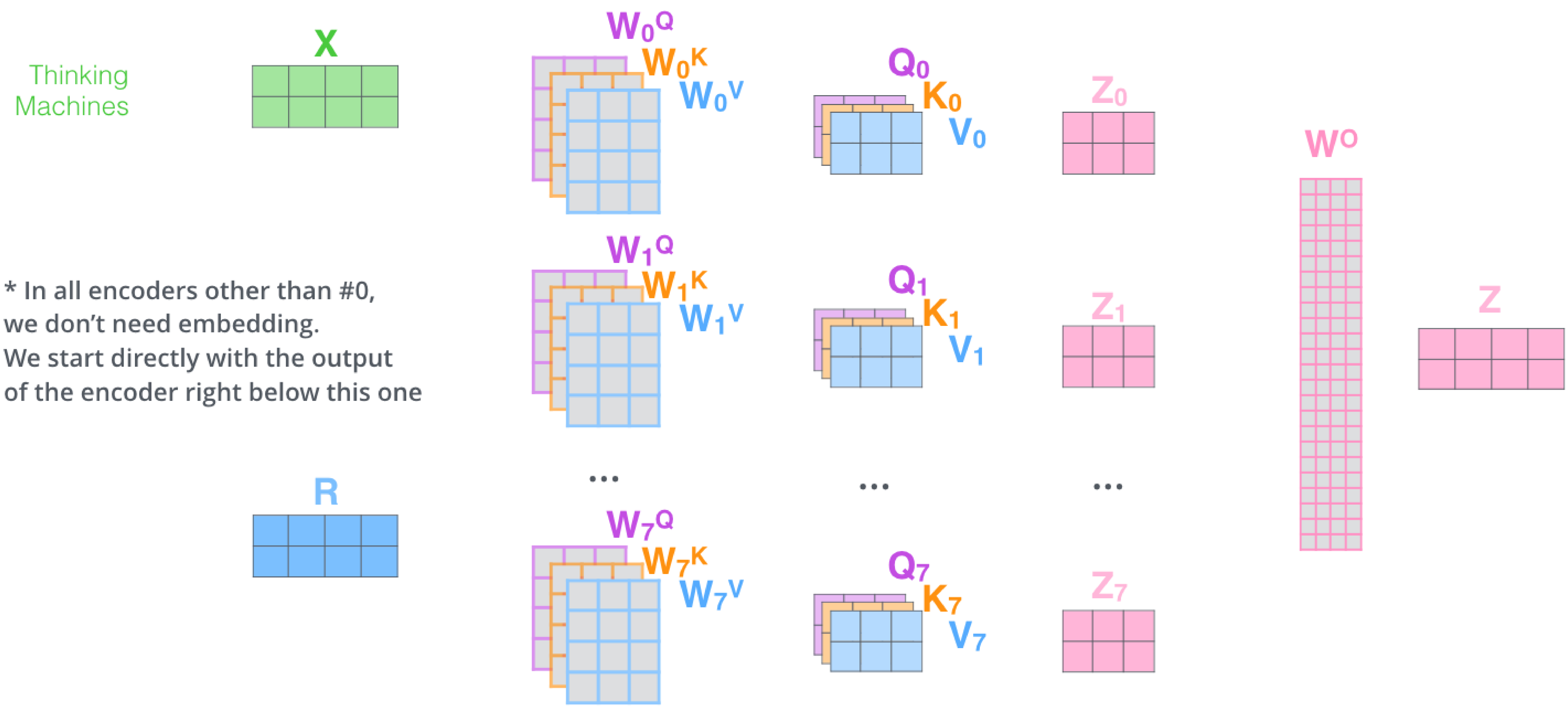
1) This is our input sentence*

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

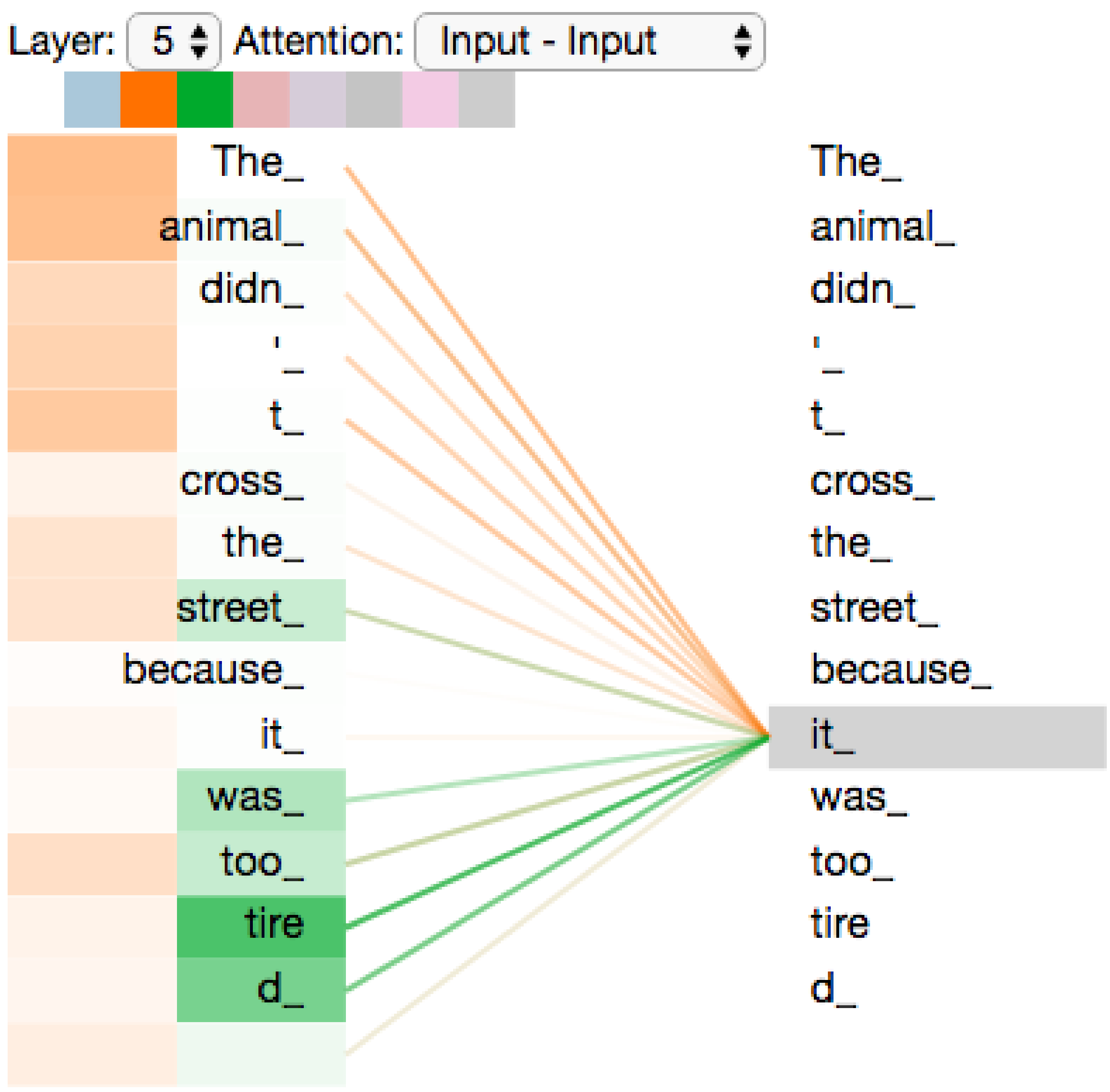


* In all encoders other than #0, we don't need embedding. We start directly with the output of the encoder right below this one

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

Multi-headed self-attention

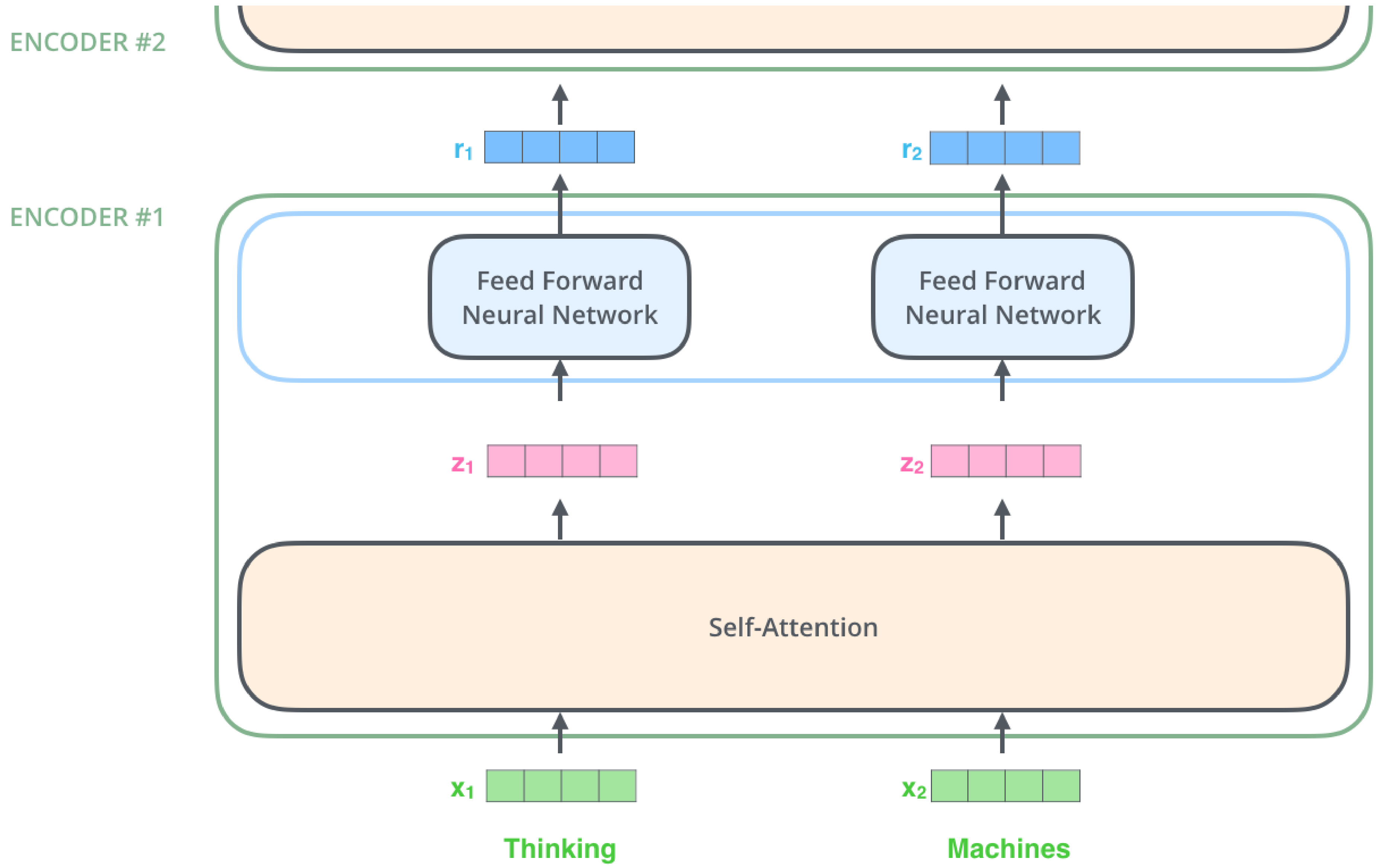
- Each attention head learns a different context:
  - `it` refers to `animal`.
  - `it` refers to `street`.
  - etc.
- The original transformer paper in 2017 used 8 attention heads.
- OpenAI's GPT-3 uses 96 attention heads…



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

Encoder layer

- Multi-headed self-attention produces a vector $\mathbf{z}_i$ for each word of the sentence.
- A regular feedforward MLP transforms it into a new representation $\mathbf{r}_i$.
    - one input layer with 512 neurons.
    - one hidden layer with 2048 neurons and a ReLU activation function.
    - one output layer with 512 neurons.
- The same NN is applied on all words, it does not depend on the length n of the sentence.



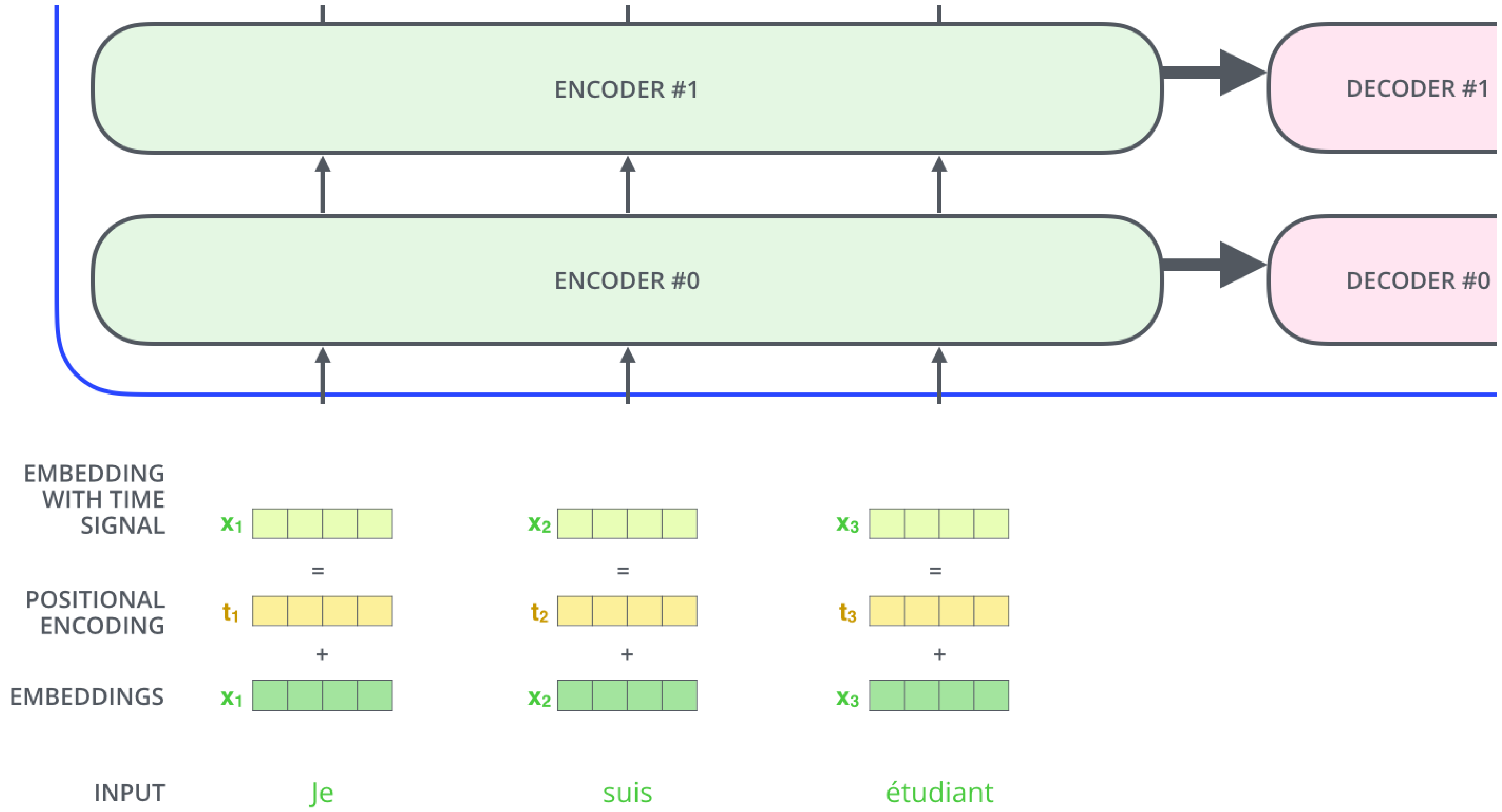
Source: http://jalammar.github.io/illustrated-transformer/

Positional encoding

- As each word is processed in parallel, the order of the words in the sentence is lost.

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

- We need to explicitly provide that information in the **input** using positional encoding.
- A simple method would be to append an index $i = 1, 2, \ldots, n$ to the word embeddings, but it is not very robust.



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

Positional encoding

- If the elements of the 512-d embeddings are numbers between 0 and 1, concatenating an integer between 1 and n will unbalance the dimensions.
- Normalizing that integer between 0 and 1 requires to know n in advance, this introduces a maximal sentence length...
- How about we append the binary code of that integer?

0 :	0	0	0	0	8 :	1	0	0	0
1 :	0	0	0	1	9 :	1	0	0	1
2 :	0	0	1	0	10 :	1	0	1	0
3 :	0	0	1	1	11 :	1	0	1	1
4 :	0	1	0	0	12 :	1	1	0	0
5 :	0	1	0	1	13 :	1	1	0	1
6 :	0	1	1	0	14 :	1	1	1	0
7 :	0	1	1	1	15 :	1	1	1	1

Source: https://kazemnejad.com/blog/transformer_architecture_positional_encoding/

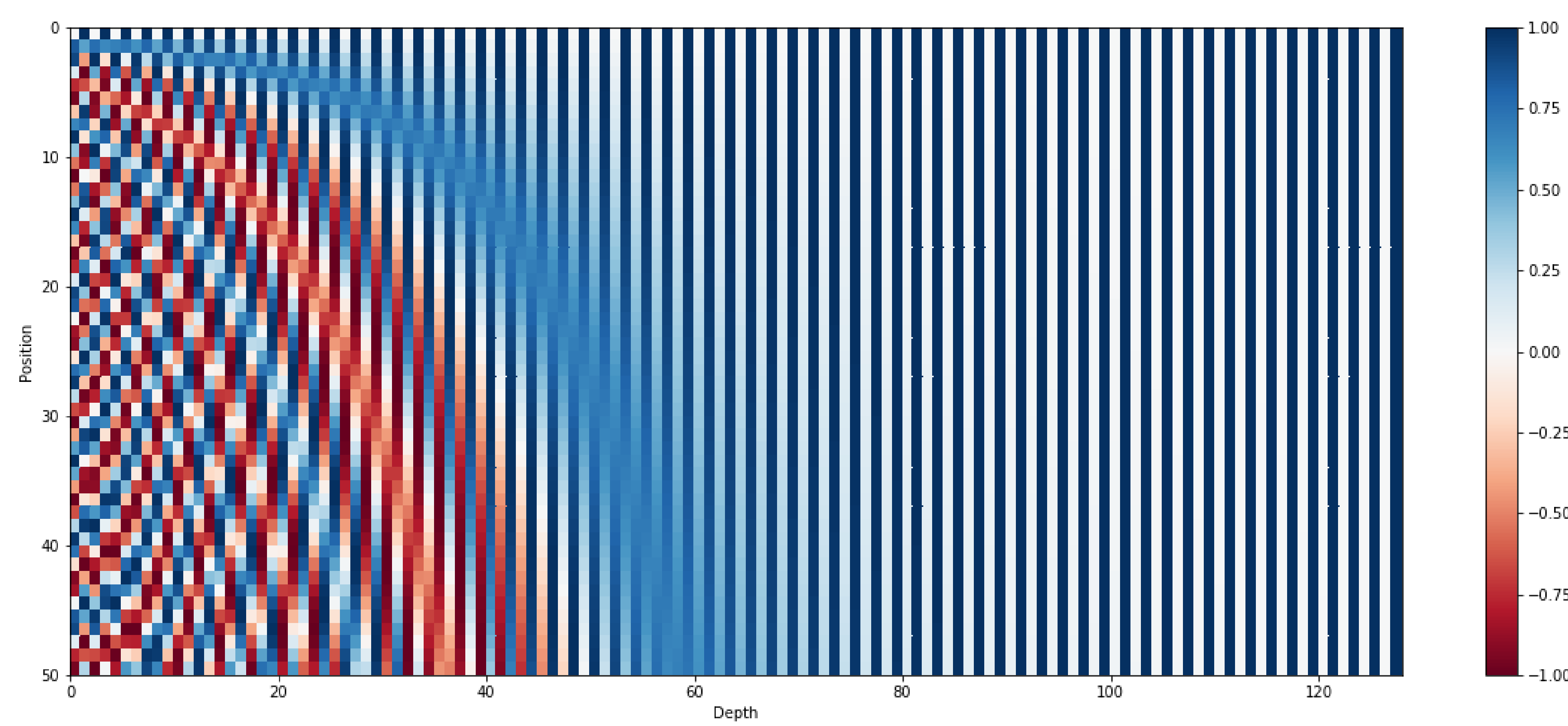
- Sounds good, we have numbers between 0 and 1, and we just need to use enough bits to encode very long sentences.
- However, representing a binary value (0 or 1) with a 64 bits floating number is a waste of computational resources.

Positional encoding

- We can notice that the bits of the integers oscillate at various frequencies:
    - the lower bit oscillates every number.
    - the bit before oscillates every two numbers.
    - etc.

0 :	0	0	0	0	8 :	1	0	0	0
1 :	0	0	0	1	9 :	1	0	0	1
2 :	0	0	1	0	10 :	1	0	1	0
3 :	0	0	1	1	11 :	1	0	1	1
4 :	0	1	0	0	12 :	1	1	0	0
5 :	0	1	0	1	13 :	1	1	0	1
6 :	0	1	1	0	14 :	1	1	1	0
7 :	0	1	1	1	15 :	1	1	1	1

- We could also represent the position of a word using sine and cosine functions at different frequencies (Fourier basis).
- We create a vector, where each element oscillates at increasing frequencies.
- The “code” for each position in the sentence is unique.

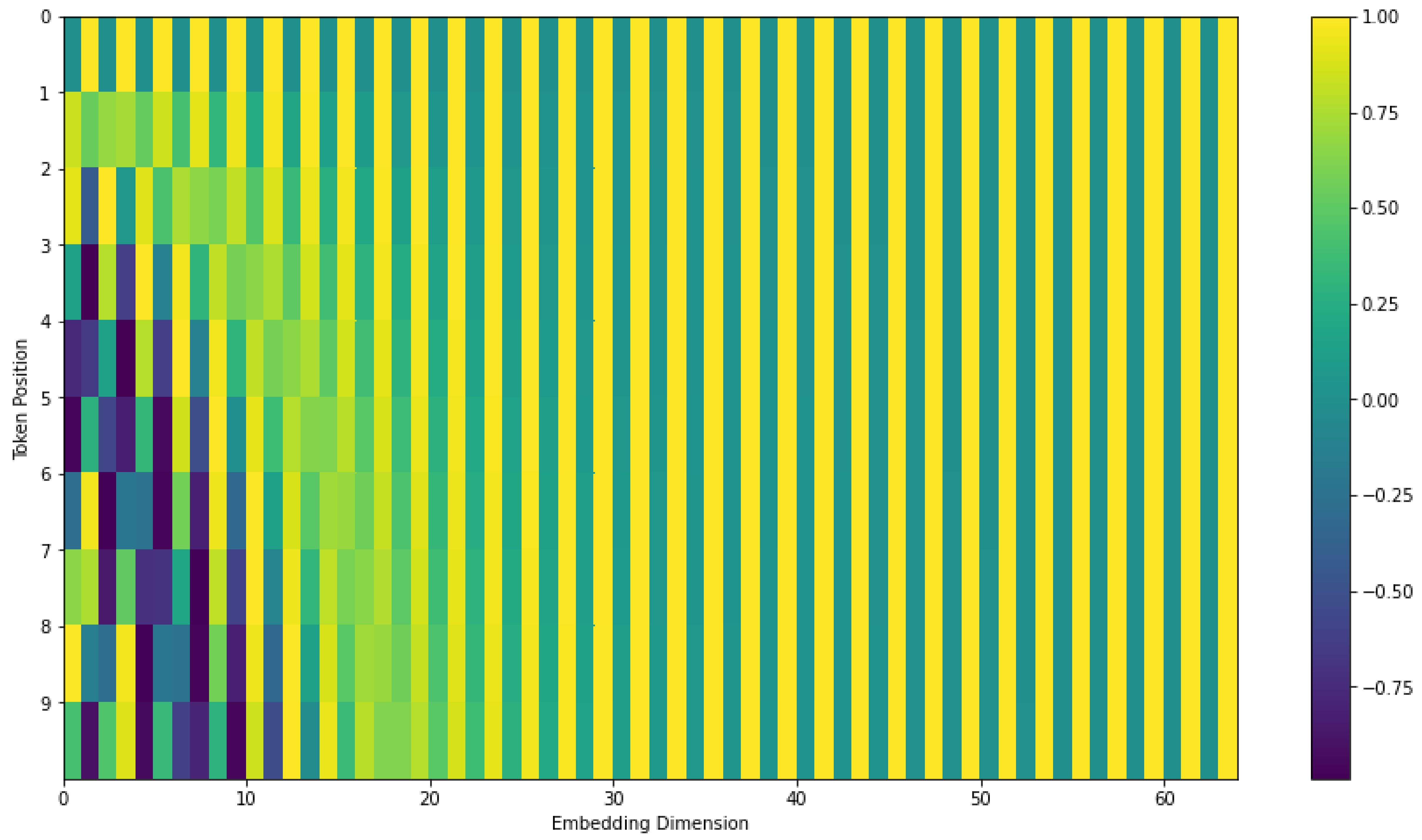


Source: https://kazemnejad.com/blog/transformer_architecture_positional_encoding/

Positional encoding

- In practice, a 512-d vector is created using sine and cosine functions.

$$\begin{cases} t(\text{pos}, 2i) = \sin(\dfrac{\text{pos}}{10000^{2i/512}}) \\ t(\text{pos}, 2i+1) = \cos(\dfrac{\text{pos}}{10000^{2i/512}}) \end{cases}$$

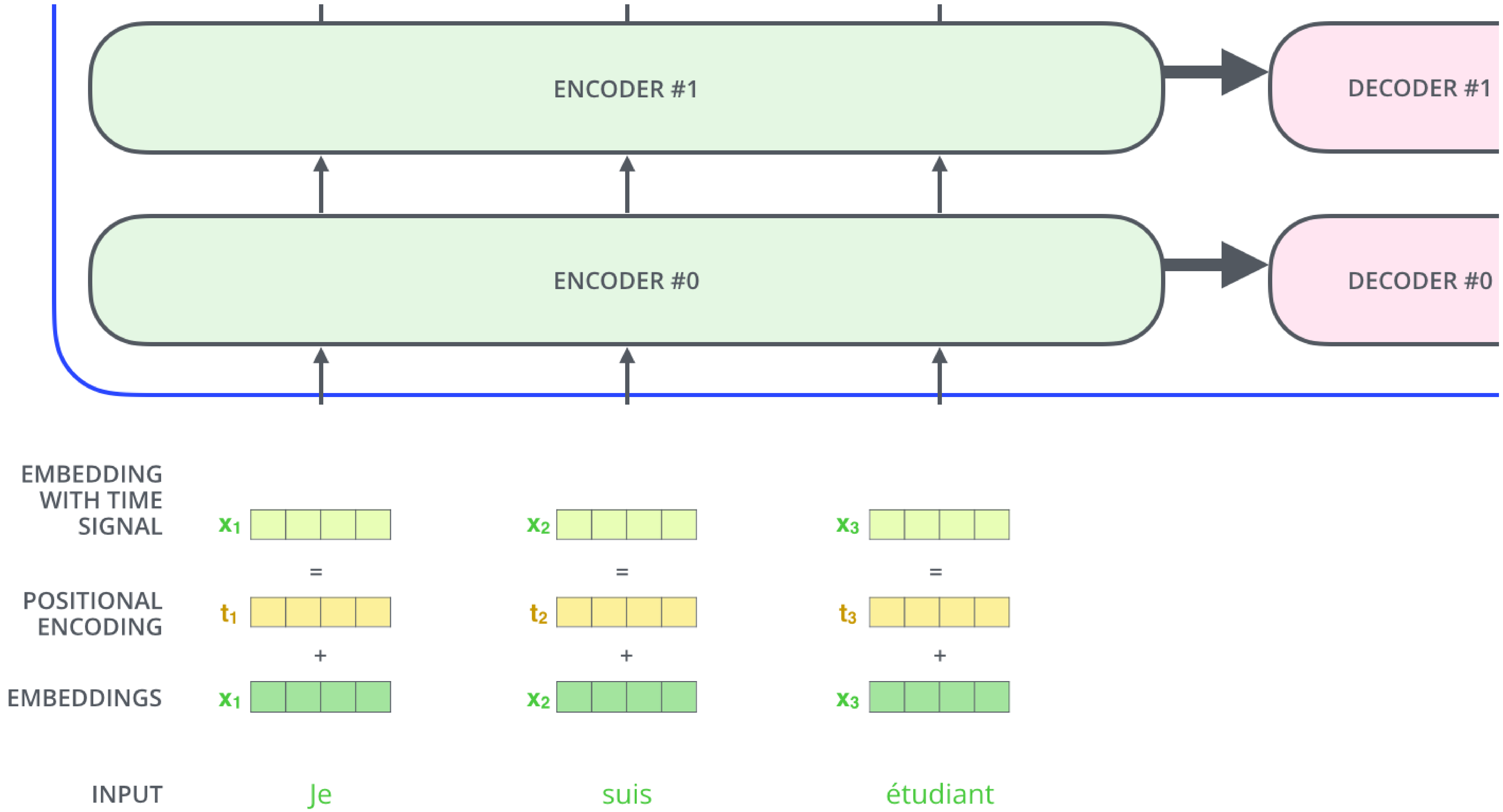


Source: http://jalammar.github.io/illustrated-transformer/

Positional encoding

- The positional encoding vector is **added** element-wise to the embedding, not concatenated!

$$\mathbf{x}_i = \mathbf{x}_i^{\text{embed}} + \mathbf{t}_i$$



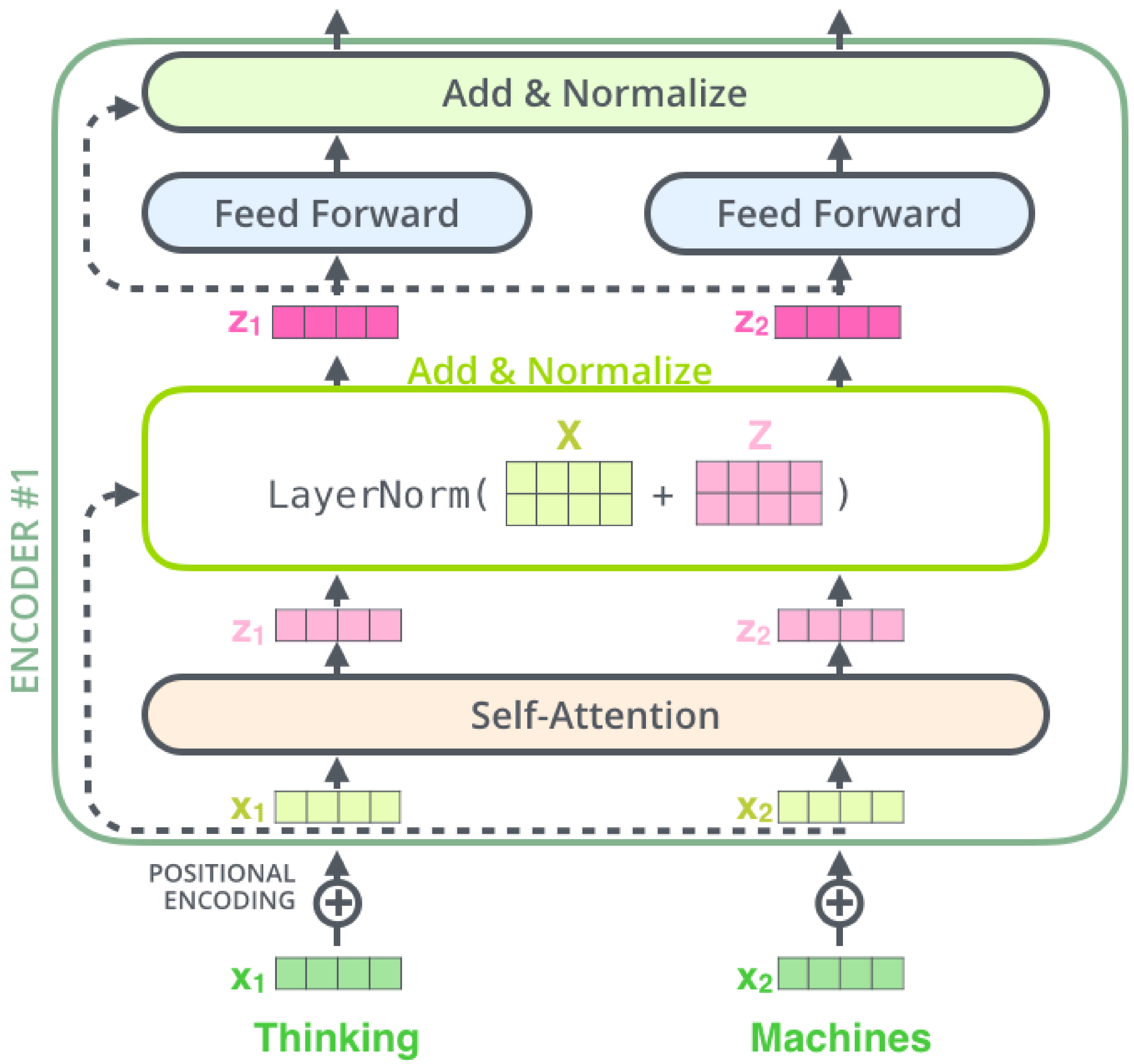
Source: http://jalammar.github.io/illustrated-transformer/

Encoder layer

- Last tricks of the encoder layers:
    - skip connections (residual layer)
    - layer normalization
- The input X is added to the output of the multi-headed self-attention and normalized (zero mean, unit variance).
- **Layer normalization** (Ba et al., 2016) is an alternative to batch normalization, where the mean and variance are computed over single vectors, not over a minibatch:

$$\mathbf{z} \leftarrow \frac{\mathbf{z} - \mu}{\sigma}$$

with $\mu = \frac{1}{d} \sum_{i=1}^d z_i$ and $\sigma = \frac{1}{d} \sum_{i=1}^d (z_i - \mu)^2$.

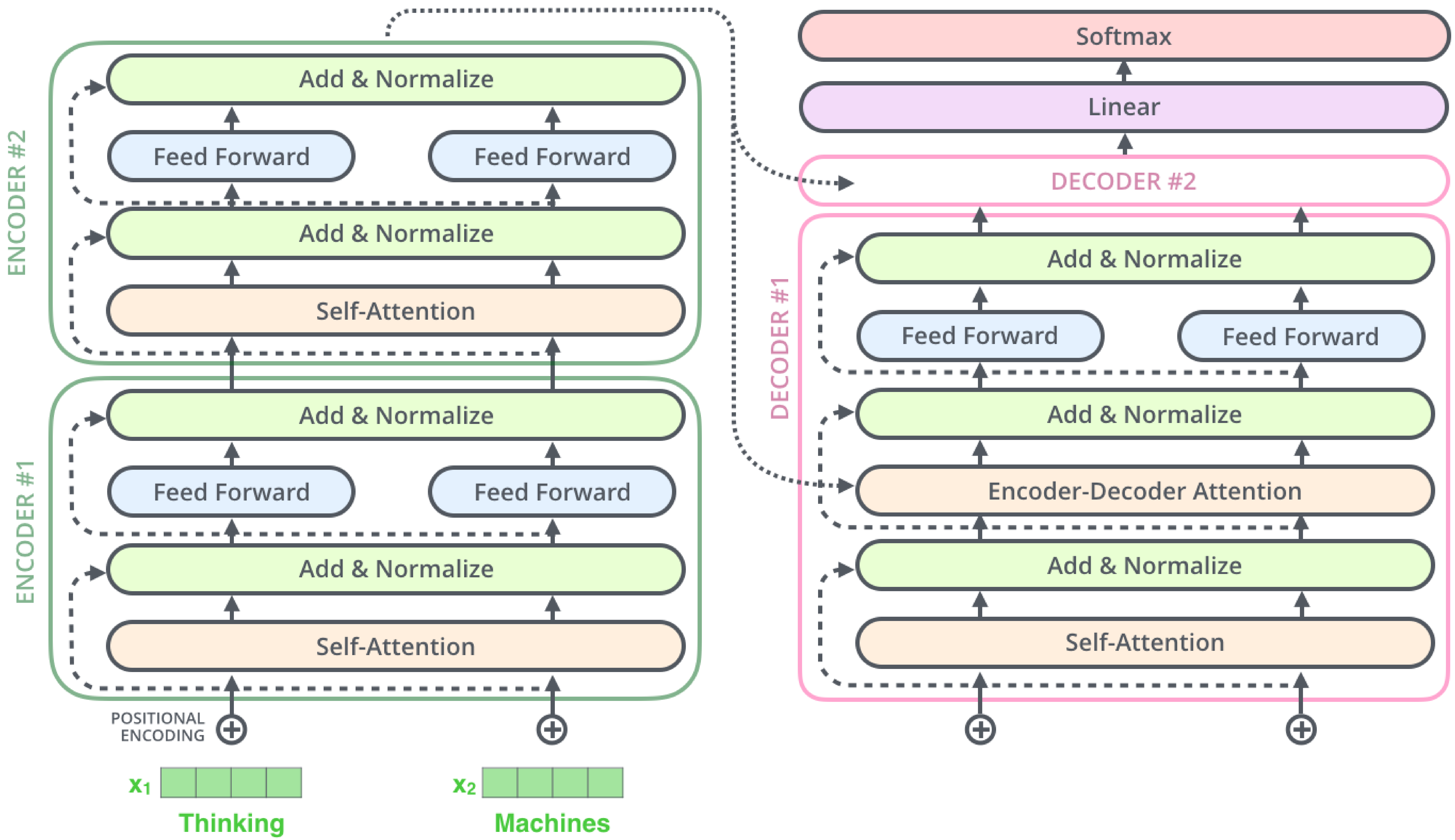


Source: http://jalammar.github.io/illustrated-transformer/

- The feedforward network also uses a skip connection and layer normalization.

Encoder

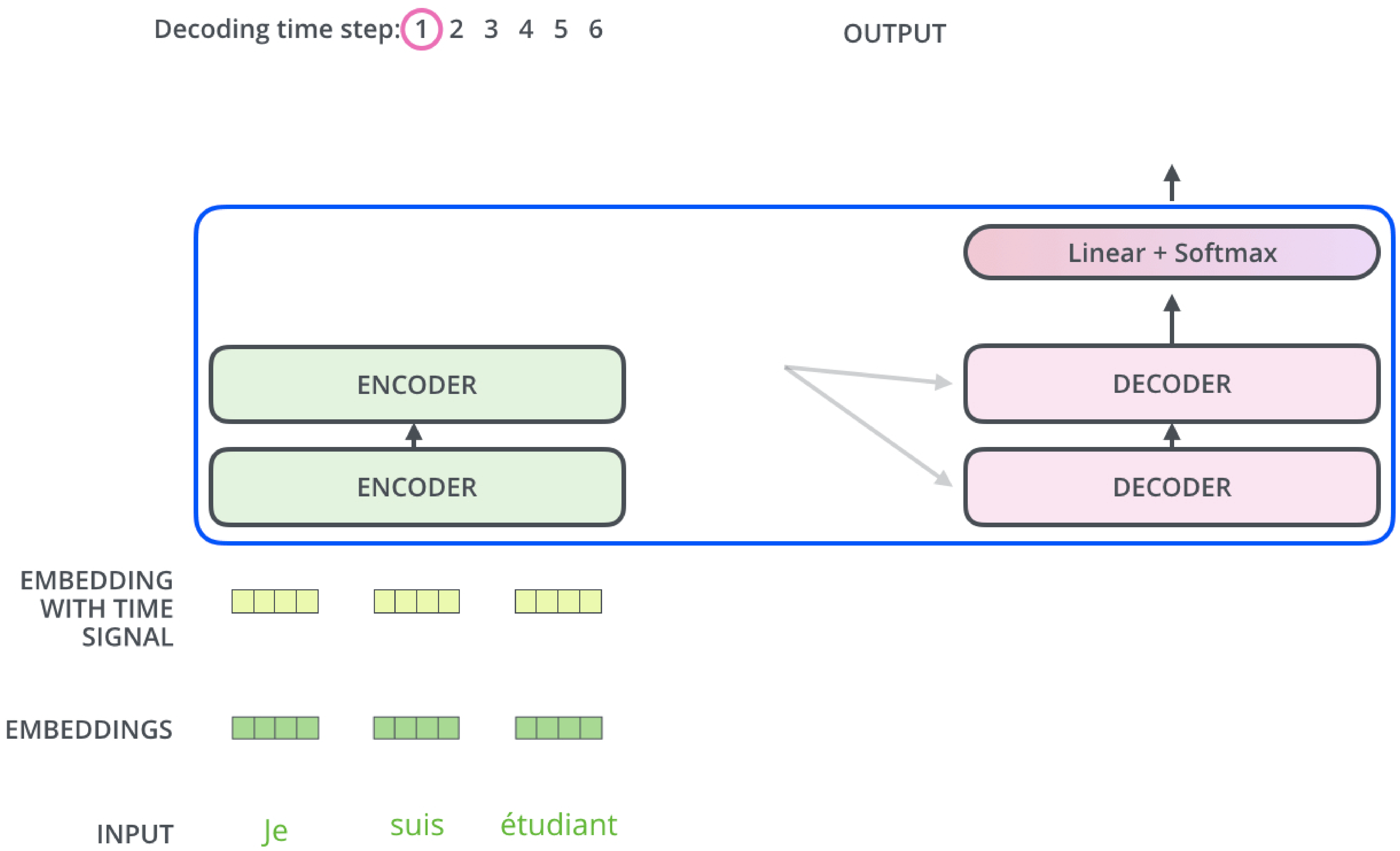
- We can now stack 6 (or more, 96 in GPT-3) of these encoder layers and use the final representation of each word as an input to the decoder.



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

Decoder

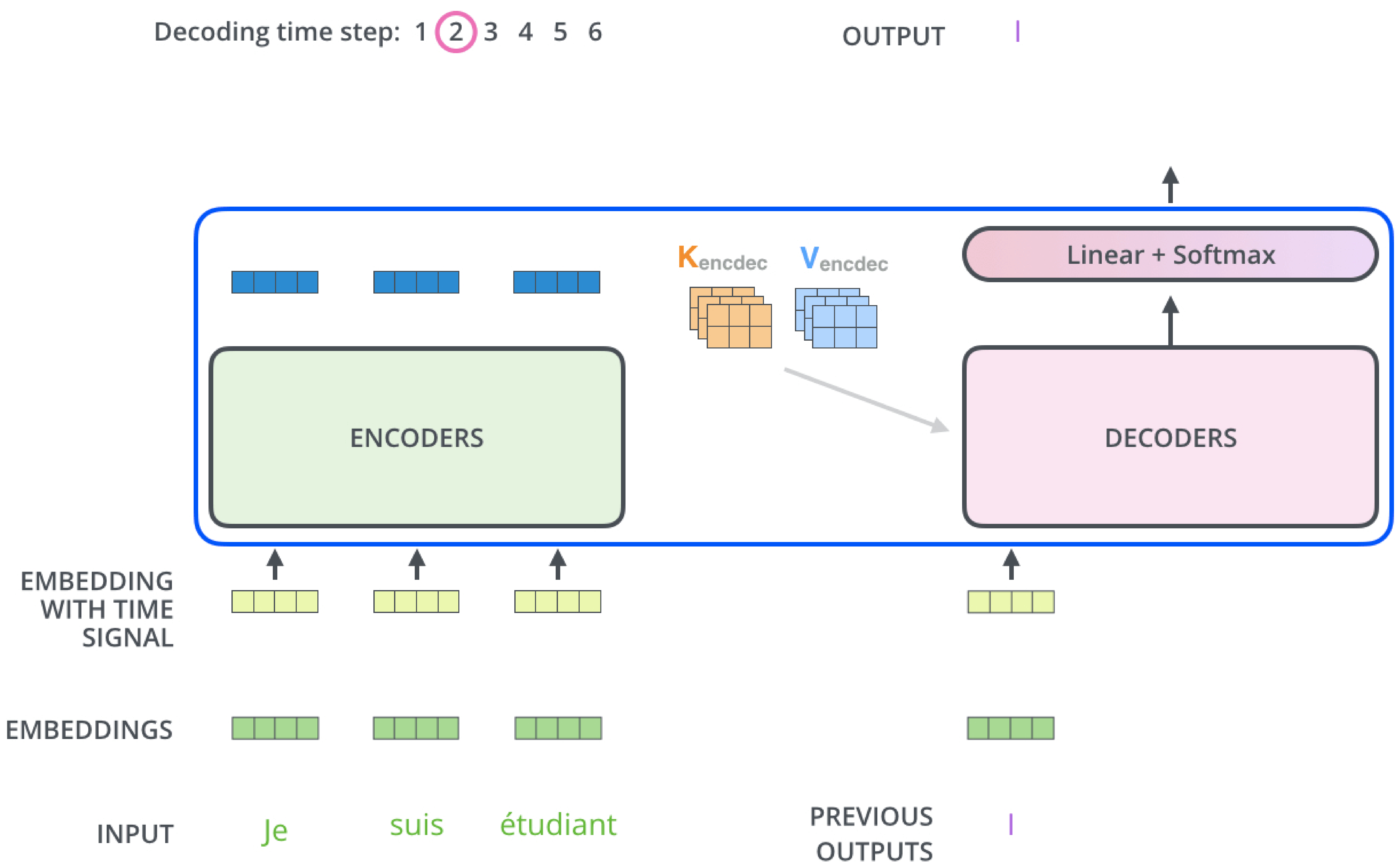
- In the first step of decoding, the final representations of the encoder are used as query and value vectors of the decoder to produce the first word.
- The input to the decoder is a “start of sentence” symbol.



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

Decoder

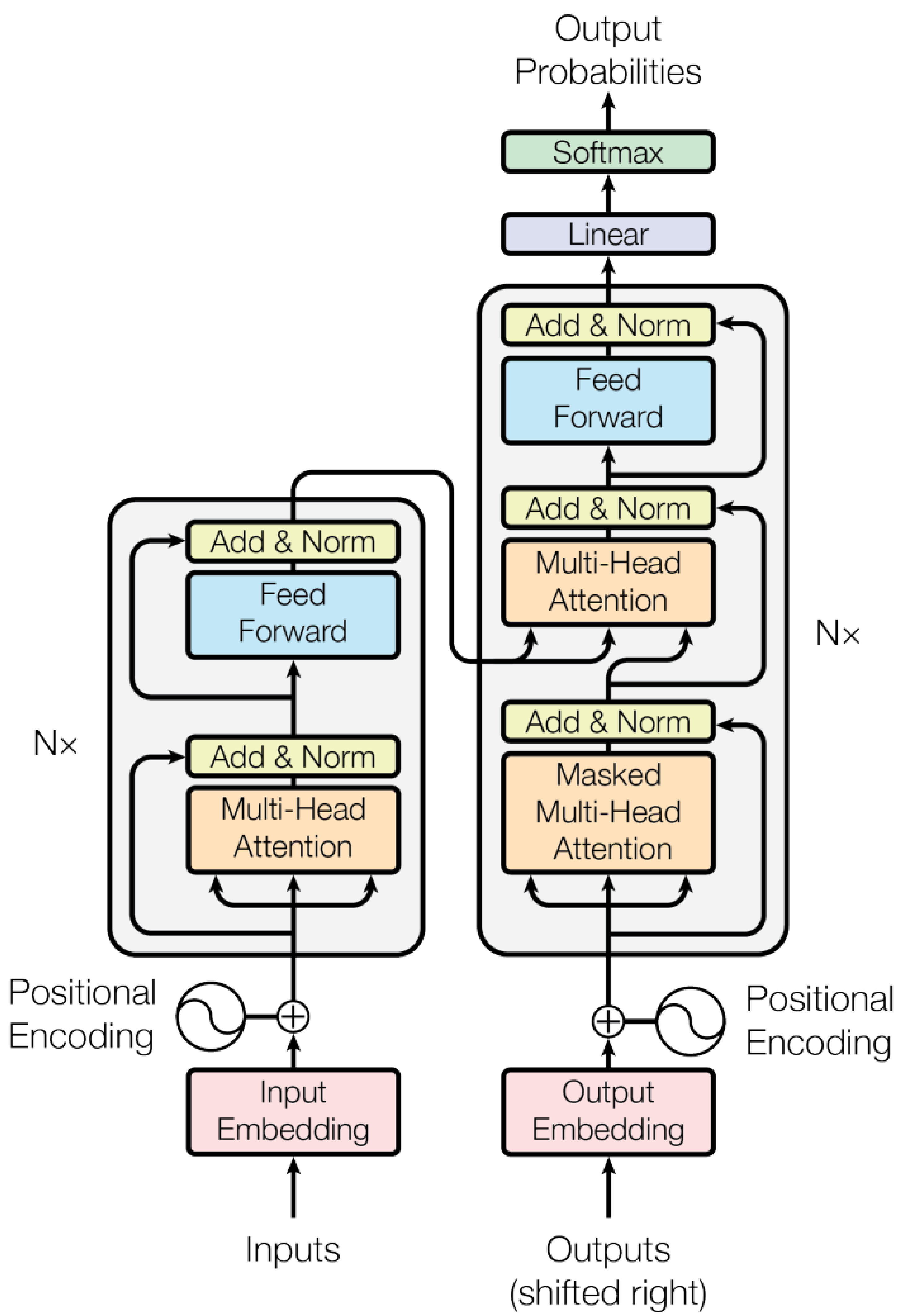
- The decoder is **autoregressive**: it outputs words one at a time, using the previously generated words as an input.



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

Decoder layer

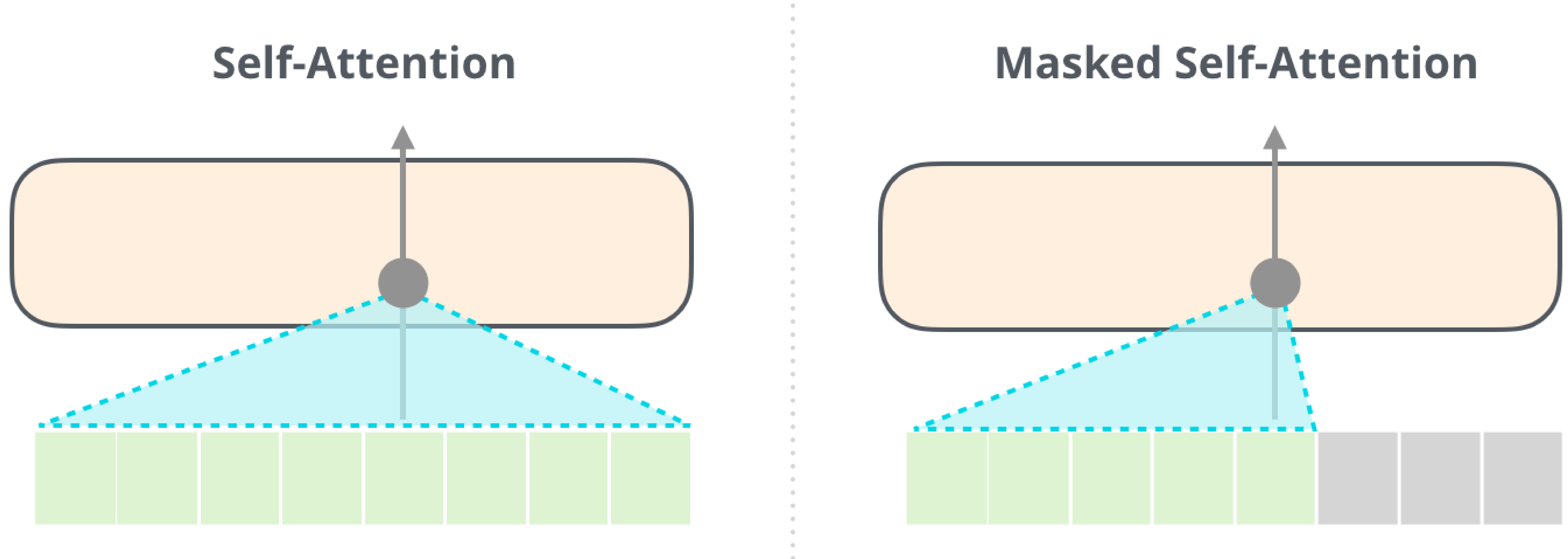
- Each decoder layer has two multi-head attention sub-layers:
  - A self-attention sub-layer with query/key/values coming from the generated sentence.
  - An **encoder-decoder** attention sub-layer, with the query coming from the generated sentence and the key/value from the encoder.
- The encoder-decoder attention is the regular attentional mechanism used in seq2seq architectures.
- Apart from this additional sub-layer, the same residual connection and layer normalization mechanisms are used.



Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., et al. (2017). **Attention Is All You Need**. arXiv:1706.03762.

Masked self-attention

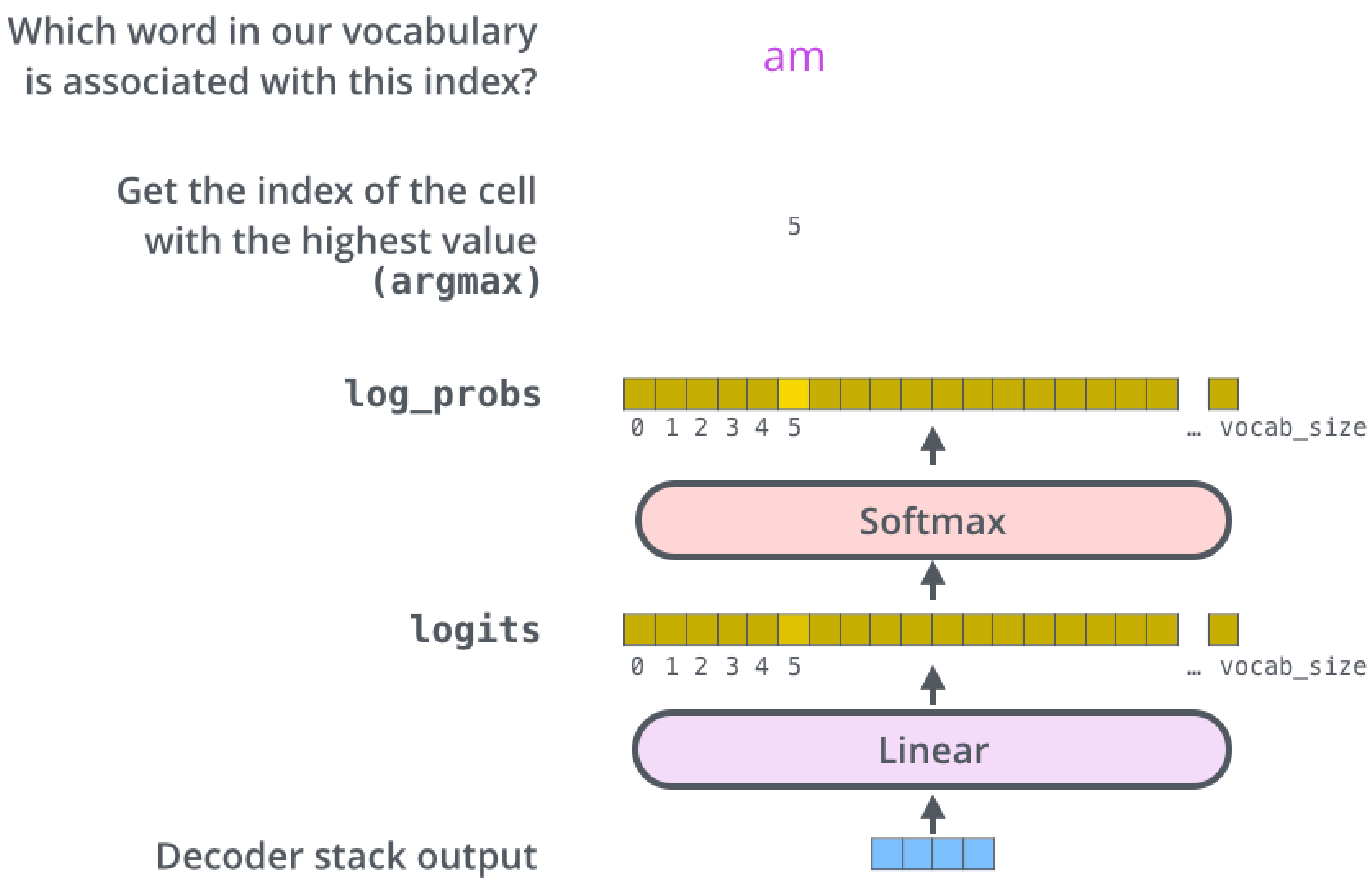
- When the sentence has been fully generated (up to the `<eos>` symbol), **masked self-attention** has to applied in order for a word in the middle of the sentence to not “see” the solution in the input when learning.
- As usual, learning occurs on minibatches of sentences, not on single words.



Source: https://jalammar.github.io/illustrated-gpt2/

Output

- The output of the decoder is a simple softmax classification layer, predicting the one-hot encoding of the word using a vocabulary (`vocab_size=25000`).



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

Training procedure

- The transformer is trained on the WMT datasets:
  - English-French: 36M sentences, 32000 unique words.
  - English-German: 4.5M sentences, 37000 unique words.
- Cross-entropy loss, Adam optimizer with scheduling, dropout. Training took 3.5 days on 8 P100 GPUs.
- The sentences can have different lengths, as the decoder is autoregressive.
- The transformer network beat the state-of-the-art performance in translation with less computations and without any RNN.

Table 2: 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.

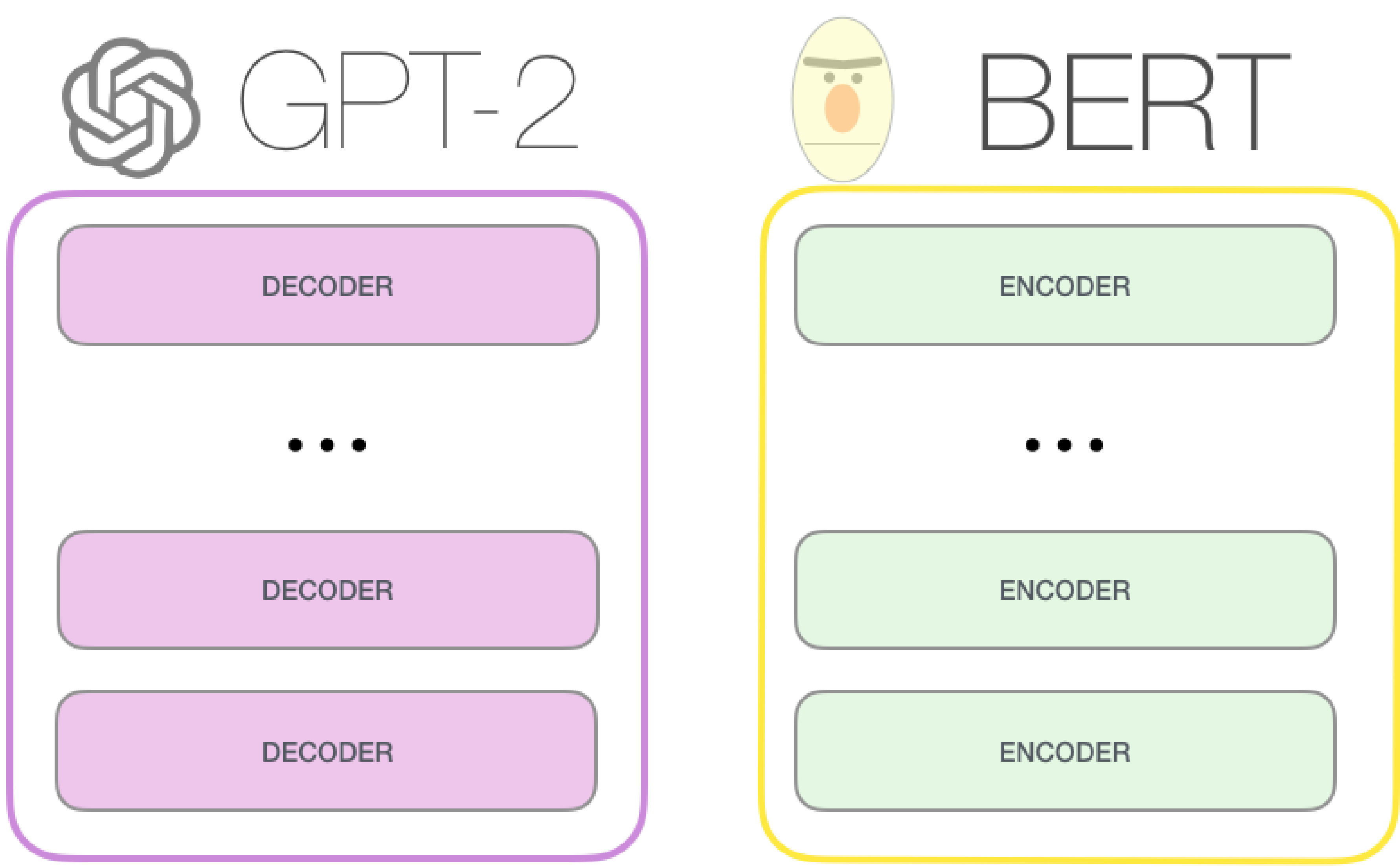
Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
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	$\mathbf{3.3 \cdot 10^{18}}$	
Transformer (big)	**28.4**	**41.8**	$2.3 \cdot 10^{19}$	

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., et al. (2017). **Attention Is All You Need**. arXiv:1706.03762.

2 - Self-supervised transformers

Transformer-based language models

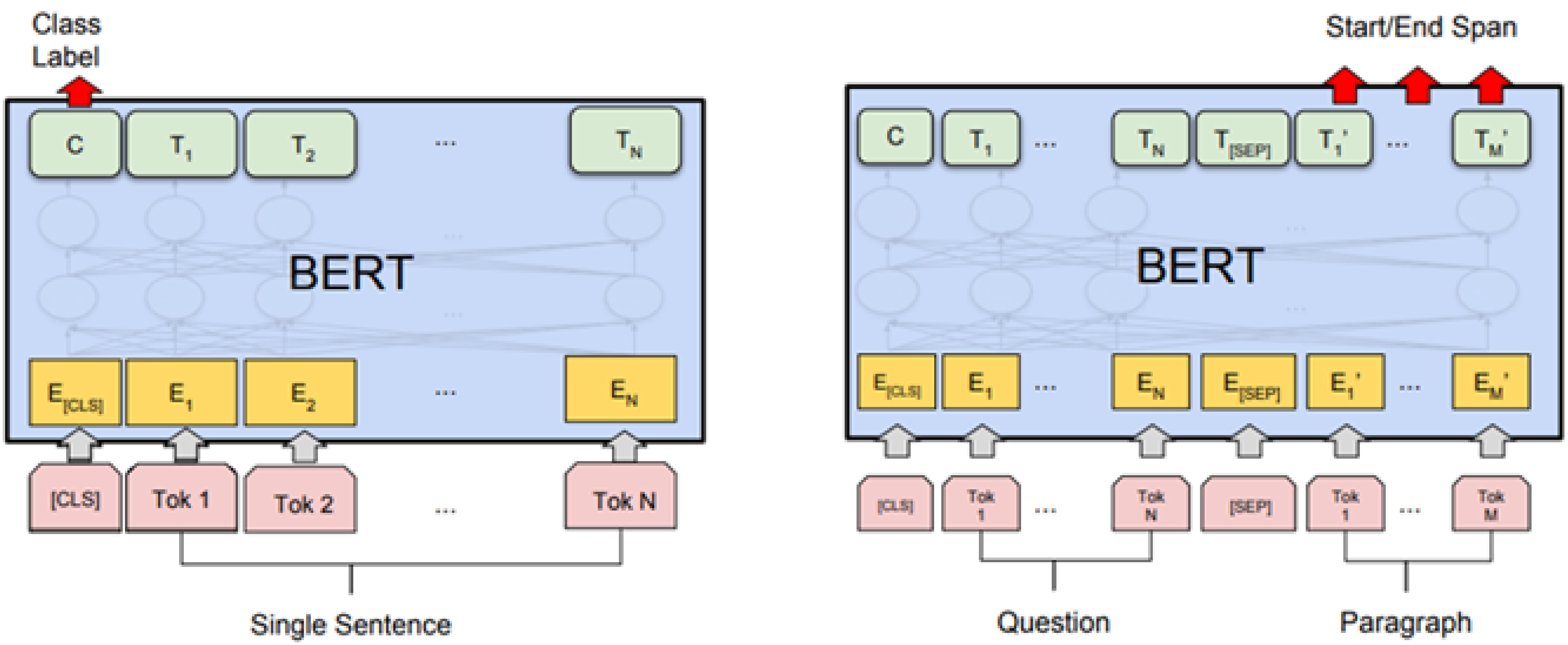
- The Transformer is considered as the **AlexNet** moment of natural language processing (NLP).
- However, it is limited to supervised learning of sentence-based translation.
- Two families of architectures have been developed from that idea to perform all NLP tasks using **unsupervised pretraining** or **self-supervised training**:
  - BERT (Bidirectional Encoder Representations from Transformers) from Google.
  - GPT (Generative Pre-trained Transformer) from OpenAI https://openai.com/blog/better-language-models/.



Source: https://jalammar.github.io/illustrated-gpt2/

BERT - Bidirectional Encoder Representations from Transformers

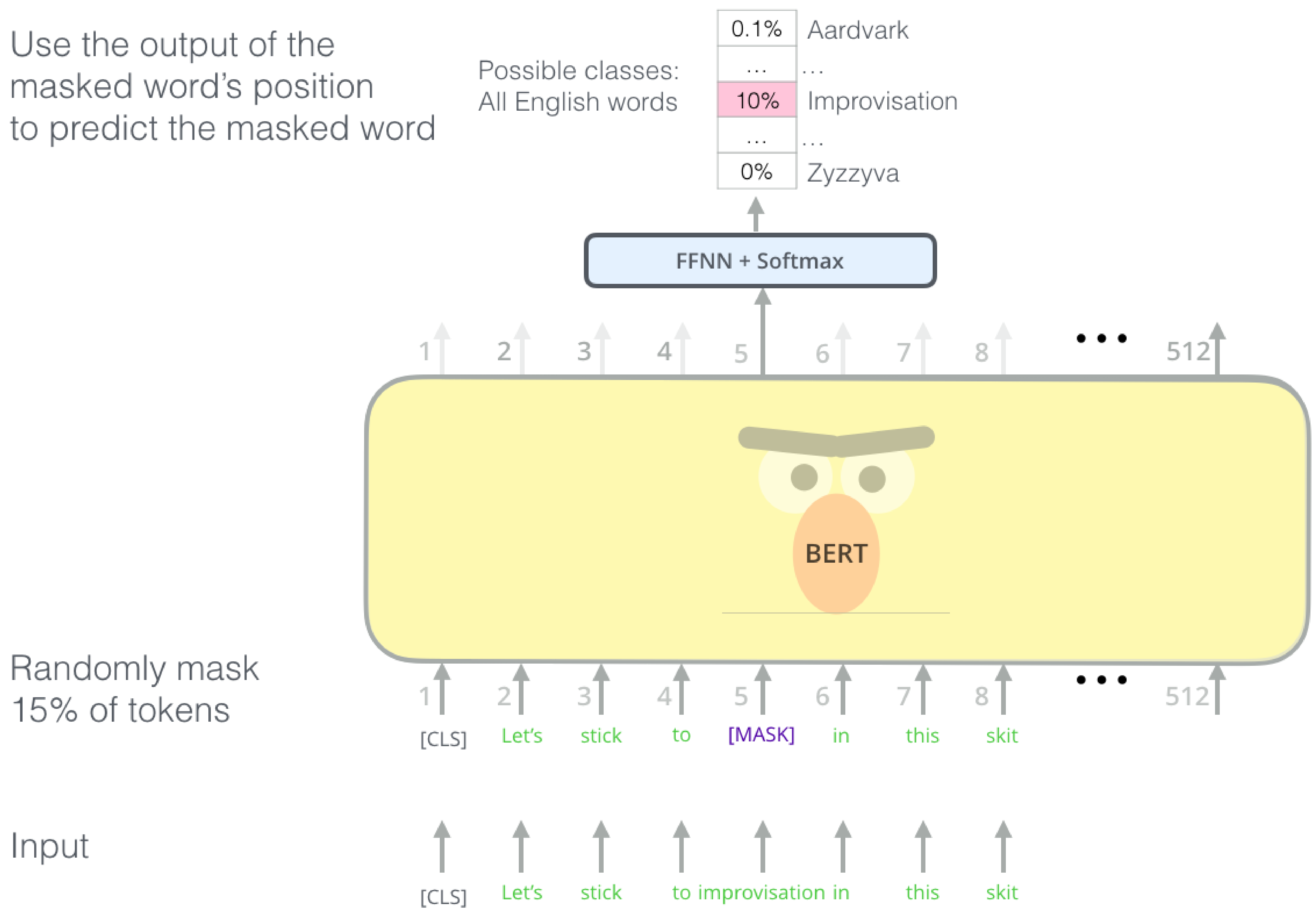
- BERT only uses the encoder of the transformer (12 layers, 12 attention heads, $d = 768$).
- BERT is pretrained on two different unsupervised tasks before being fine-tuned on supervised tasks.



Devlin J, Chang M-W, Lee K, Toutanova K. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv:181004805

BERT - Bidirectional Encoder Representations from Transformers

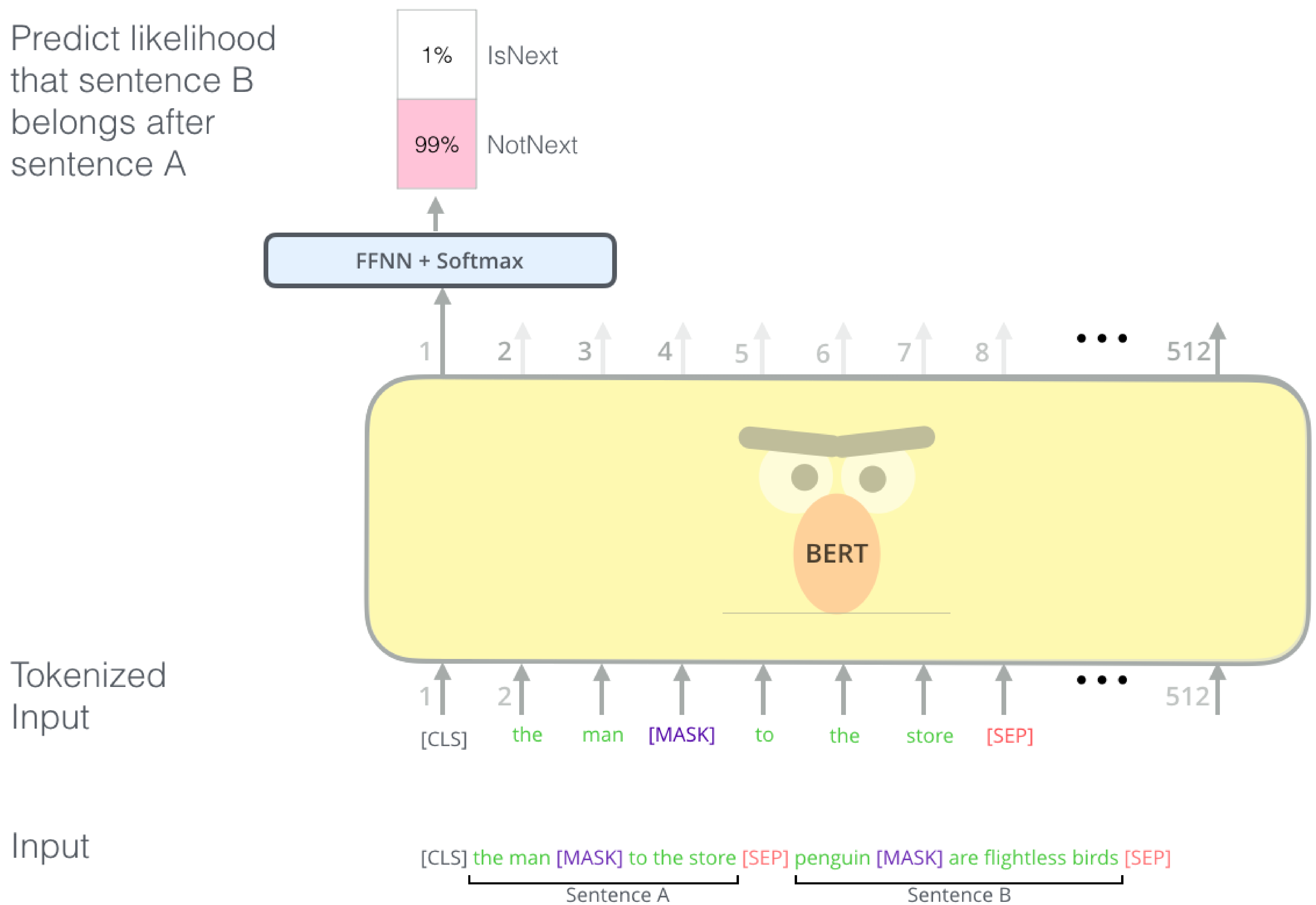
- **Task 1:** Masked language model. Sentences from BooksCorpus and Wikipedia (3.3G words) are presented to BERT during pre-training, with 15% of the words masked.
- The goal is to predict the masked words from the final representations using a shallow FNN.



Source: https://jalammar.github.io/illustrated-bert/

BERT - Bidirectional Encoder Representations from Transformers

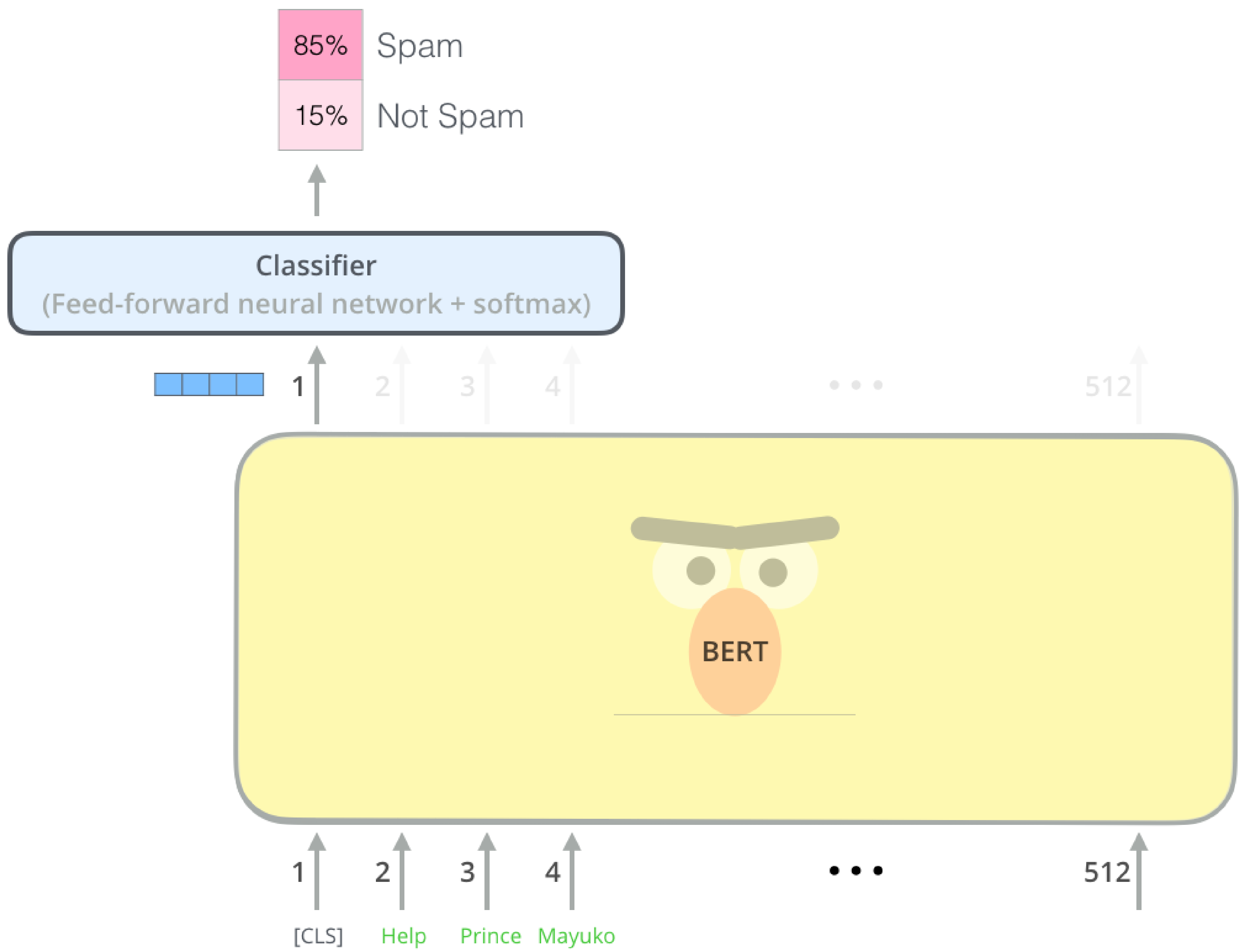
- **Task 2:** Next sentence prediction. Two sentences are presented to BERT.
- The goal is to predict from the first representation whether the second sentence should follow the first.



Source: https://jalammar.github.io/illustrated-bert/

BERT - Bidirectional Encoder Representations from Transformers

- Once BERT is pretrained, one can use **transfer learning** with or without fine-tuning from the high-level representations to perform:
  - sentiment analysis / spam detection
  - question answering



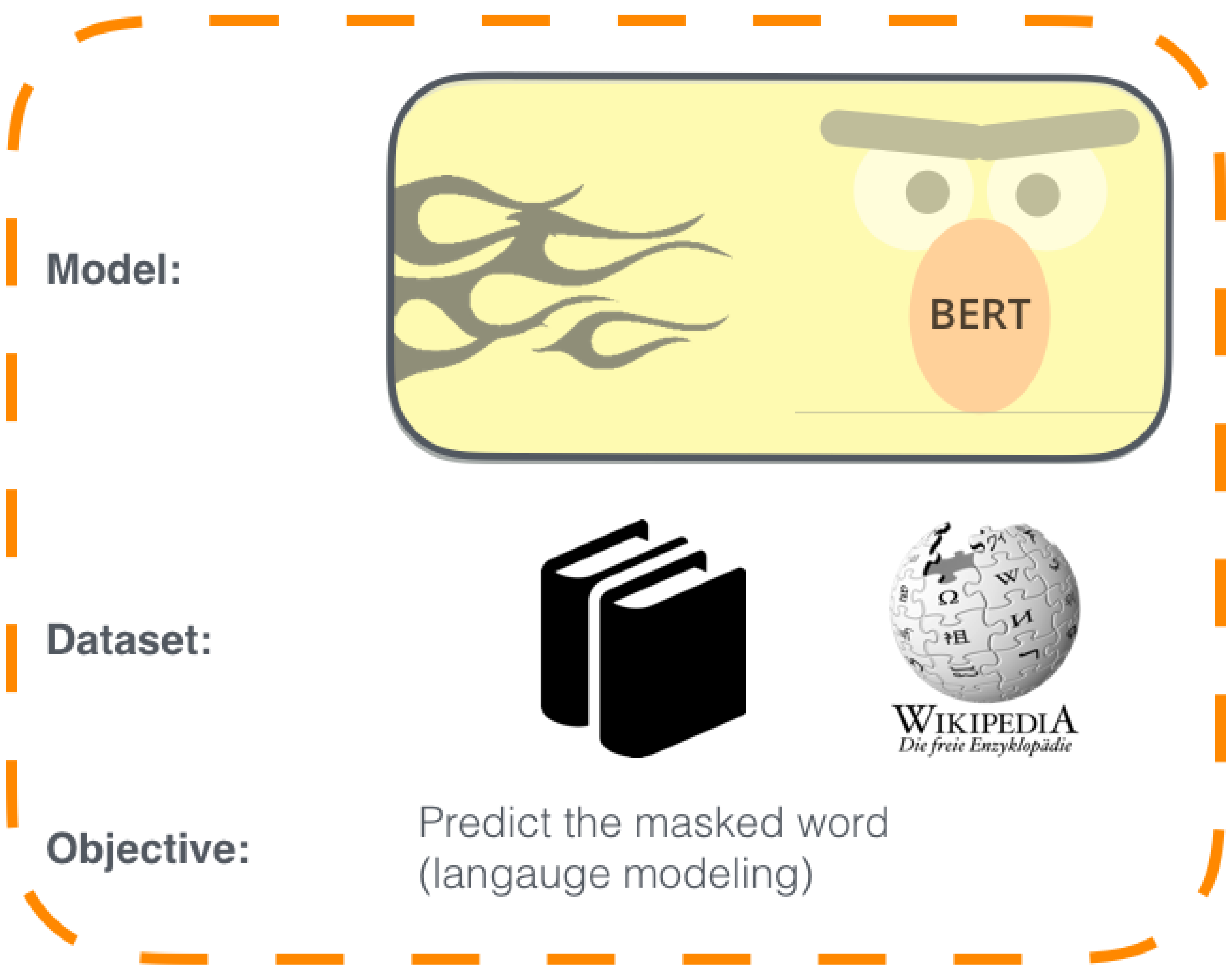
Source: https://jalammar.github.io/illustrated-bert/

BERT - Bidirectional Encoder Representations from Transformers

1 - Semi-supervised training on large amounts of text (books, wikipedia..etc).

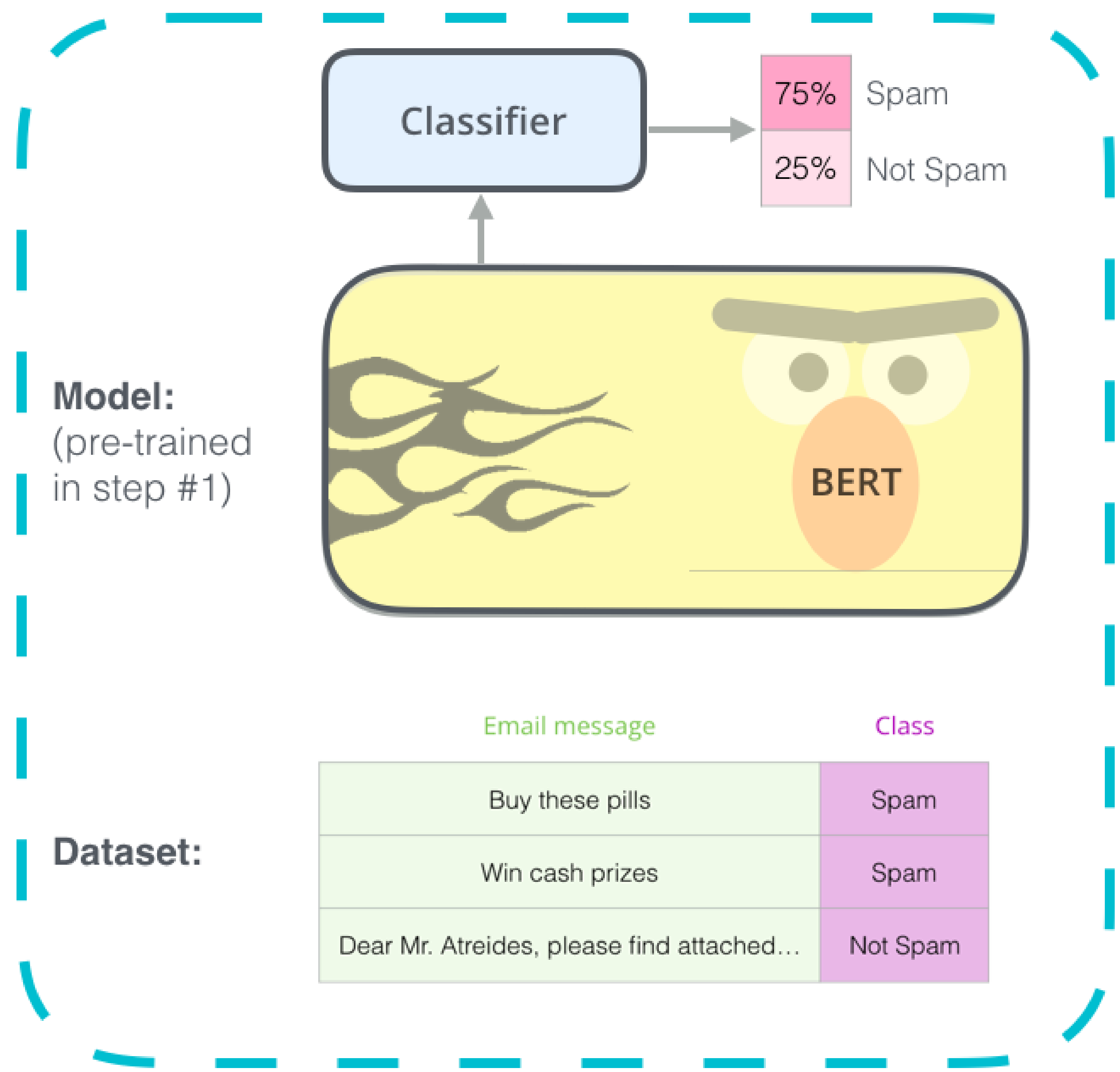
The model is trained on a certain task that enables it to grasp patterns in language. By the end of the training process, BERT has language-processing abilities capable of empowering many models we later need to build and train in a supervised way.

Semi-supervised Learning Step



2 - Supervised training on a specific task with a labeled dataset.

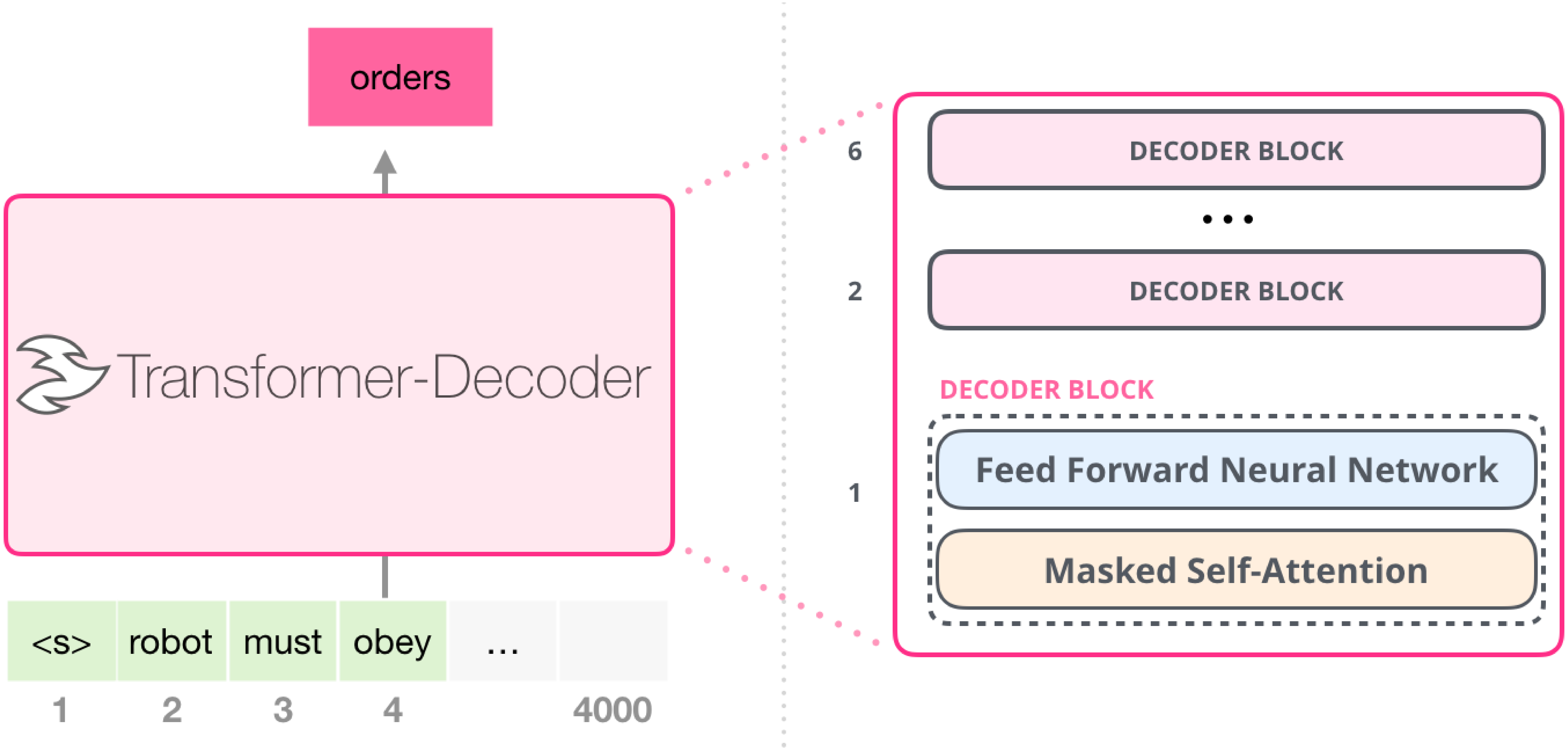
Supervised Learning Step



Source: https://jalammar.github.io/illustrated-bert/

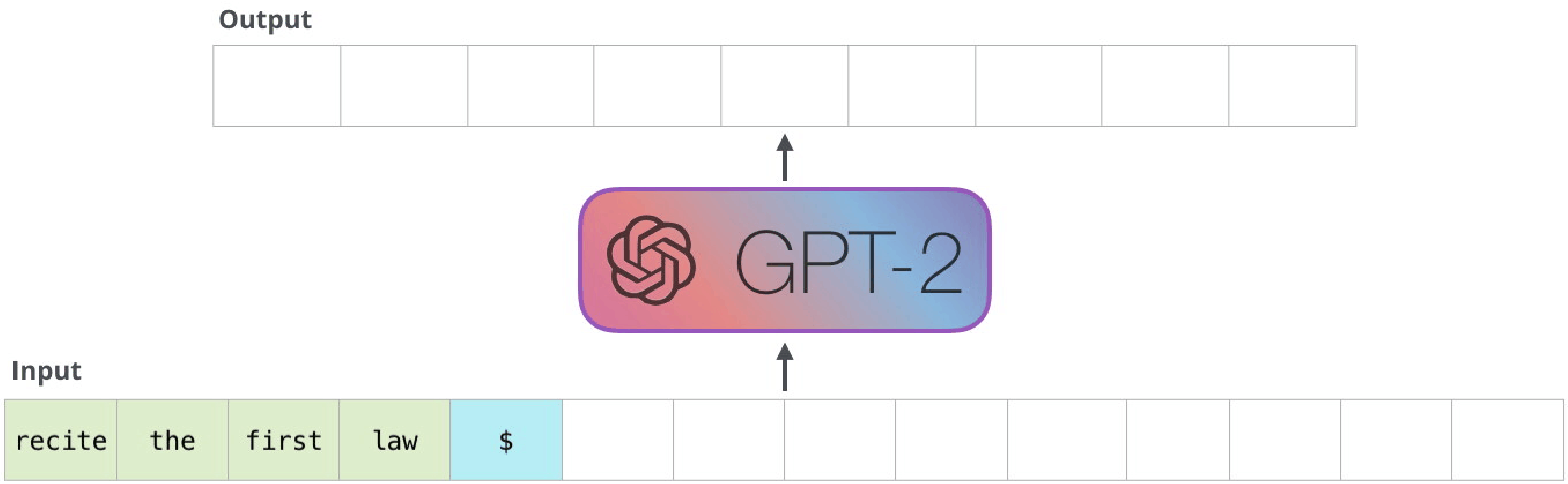
GPT - Generative Pre-trained Transformer

- As the Transformer, GPT is an **autoregressive** language model learning to predict the next word using only the transformer's **decoder**.



Source: https://jalammar.github.io/illustrated-gpt2/

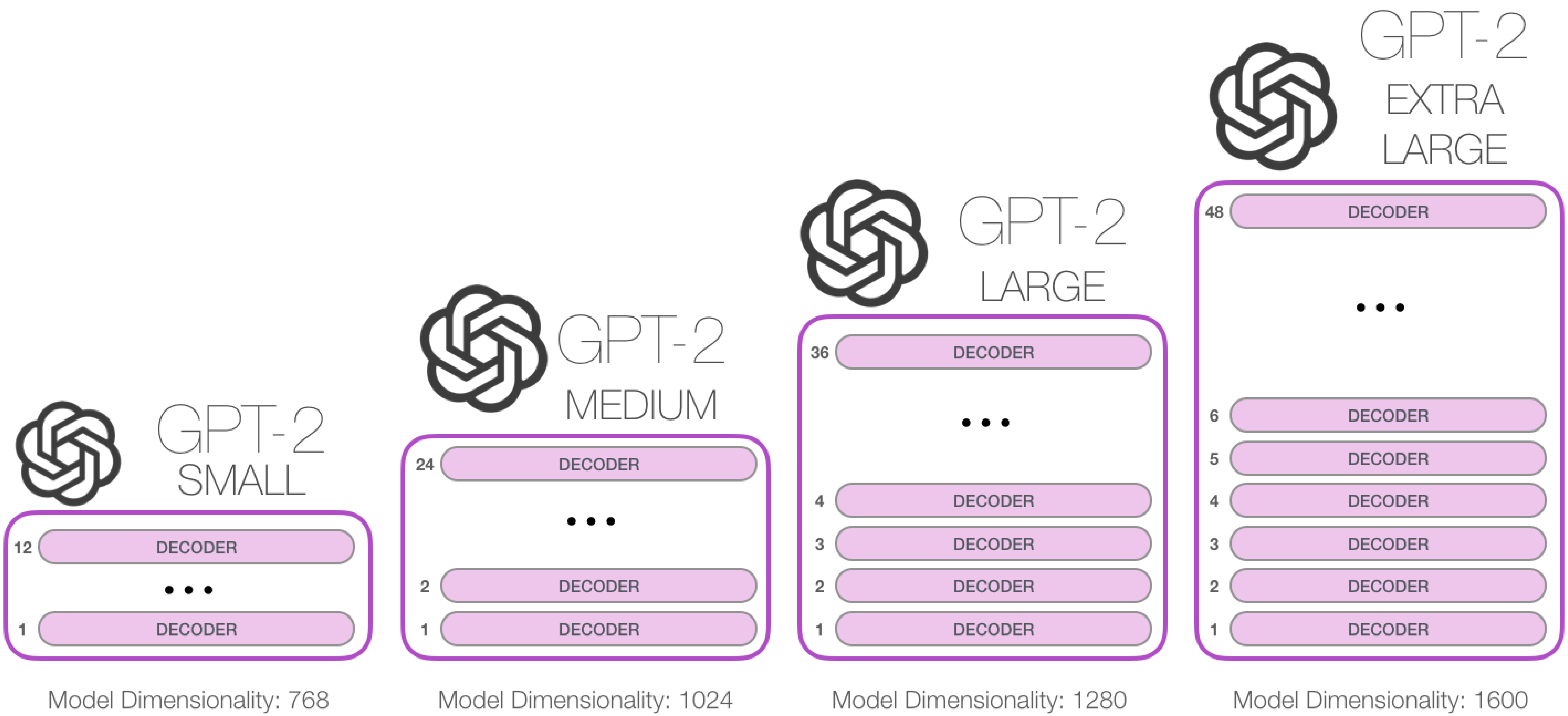
GPT - Generative Pre-trained Transformer



Source: https://jalammar.github.io/illustrated-gpt2/

GPT - Generative Pre-trained Transformer

- GPT-2 comes in various sizes, with increasing performance.
- GPT-3 is even bigger, with 175 **billion** parameters and a much larger training corpus.



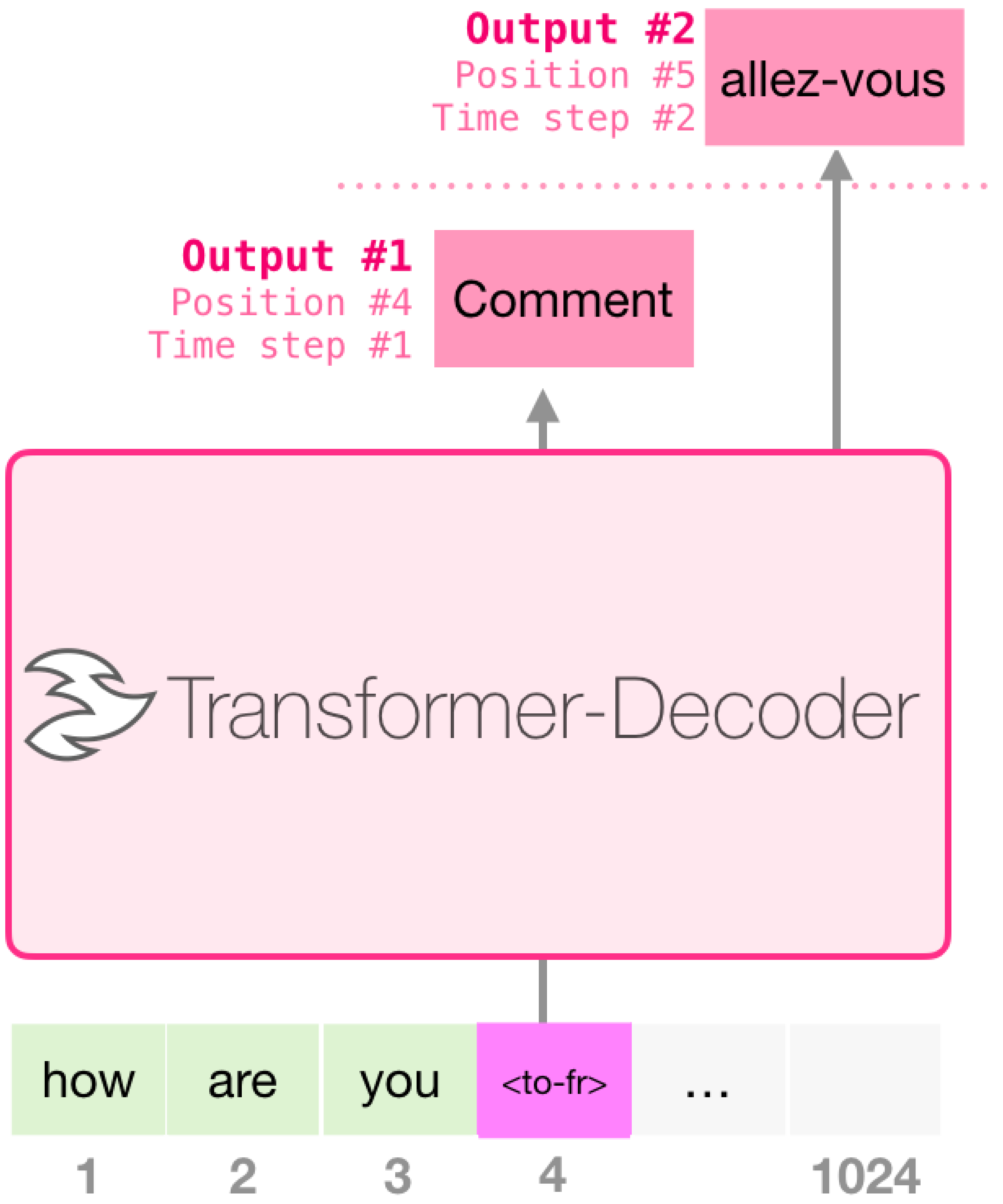
Source: https://jalammar.github.io/illustrated-gpt2/

GPT - Generative Pre-trained Transformer

- GPT can be fine-tuned (transfer learning) to perform **machine translation**.

Training Dataset

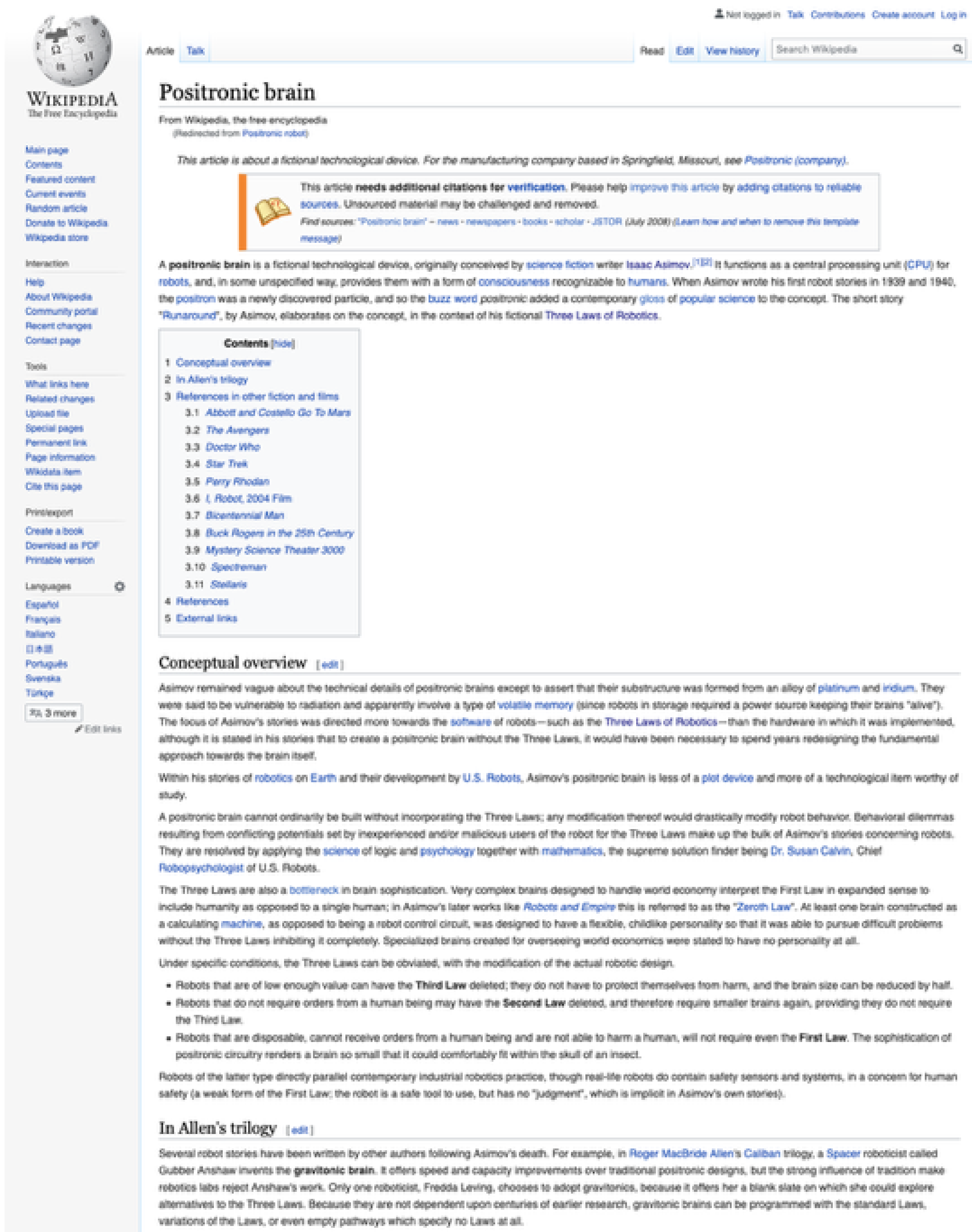
I	am	a	student	<to-fr>	je	suis	étudiant
let	them	eat	cake	<to-fr>	Qu'ils	mangent	de
good	morning	<to-fr>	Bonjour				

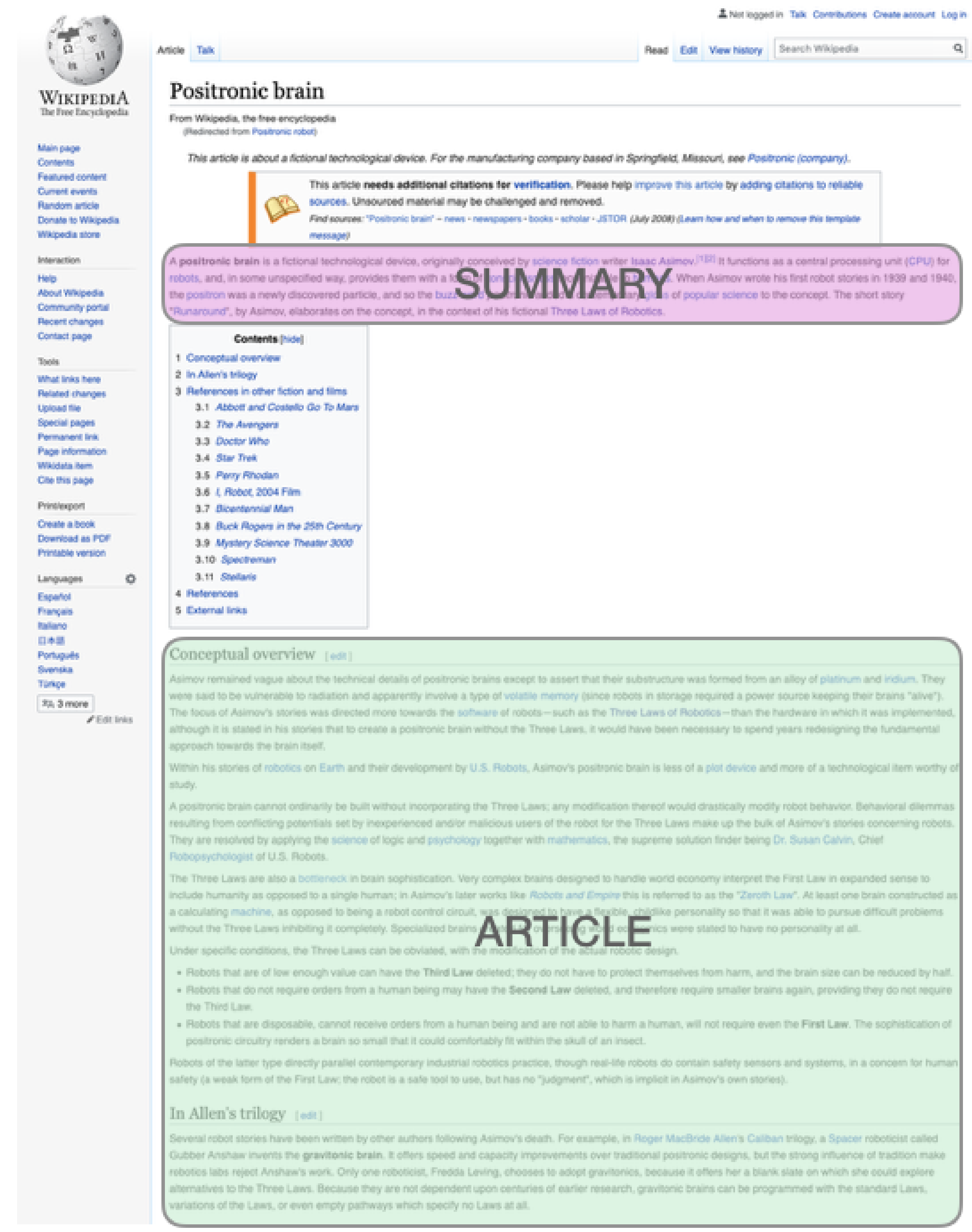


Source: https://jalammar.github.io/illustrated-gpt2/

GPT - Generative Pre-trained Transformer

- GPT can be fine-tuned to summarize Wikipedia articles.

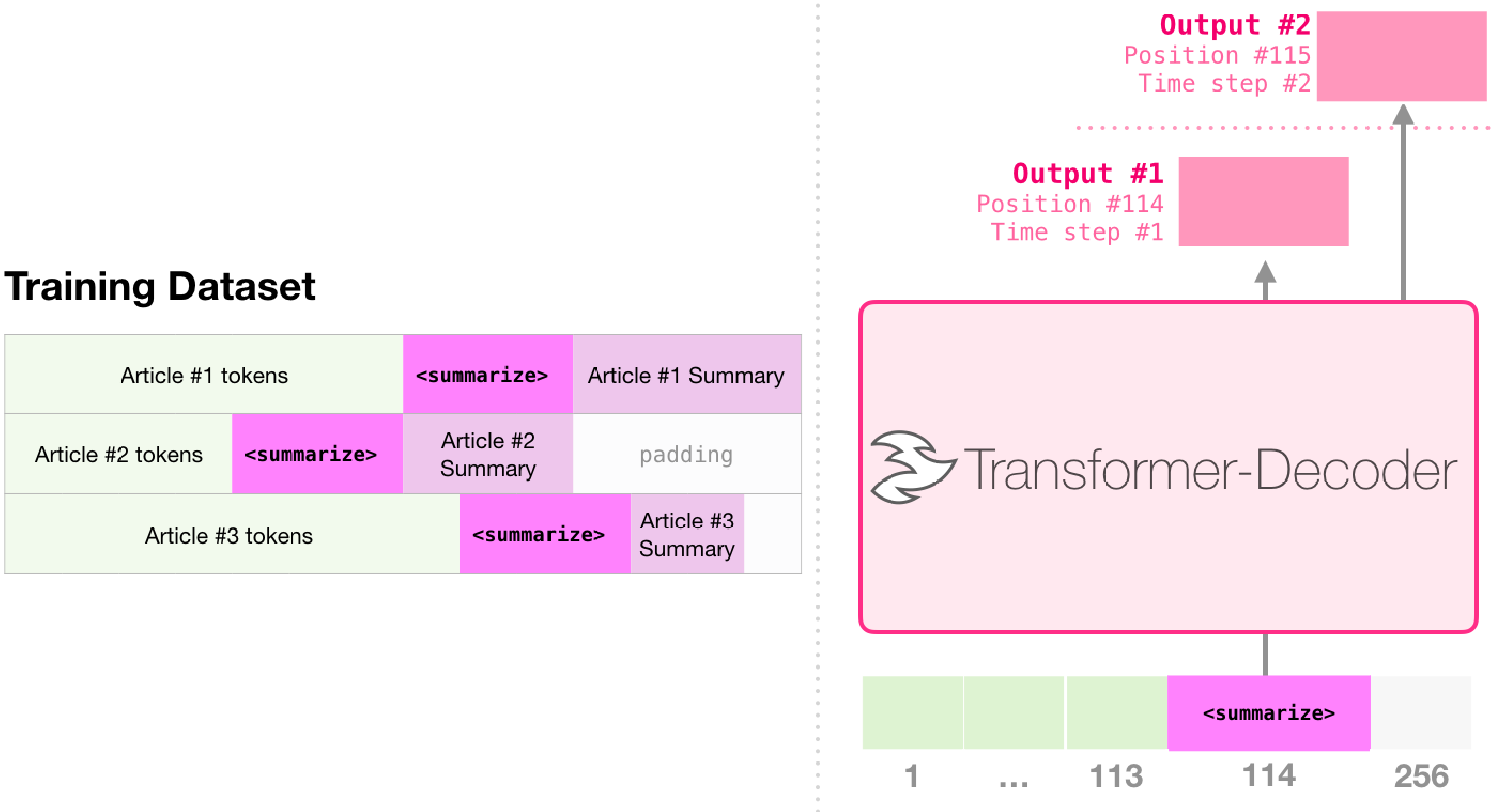




Source: https://jalammar.github.io/illustrated-gpt2/

GPT - Generative Pre-trained Transformer

- GPT can be fine-tuned to summarize Wikipedia articles.



Source: https://jalammar.github.io/illustrated-gpt2/

Try transformers at https://huggingface.co/

```
pip install transformers
```

Write With Transformer `distil-gpt2`

Shuffle initial text   Trigger autocomplete or `tab`   Select suggestion `↑` `↓` and `enter`   Cancel suggestion `esc`

Save & Publish

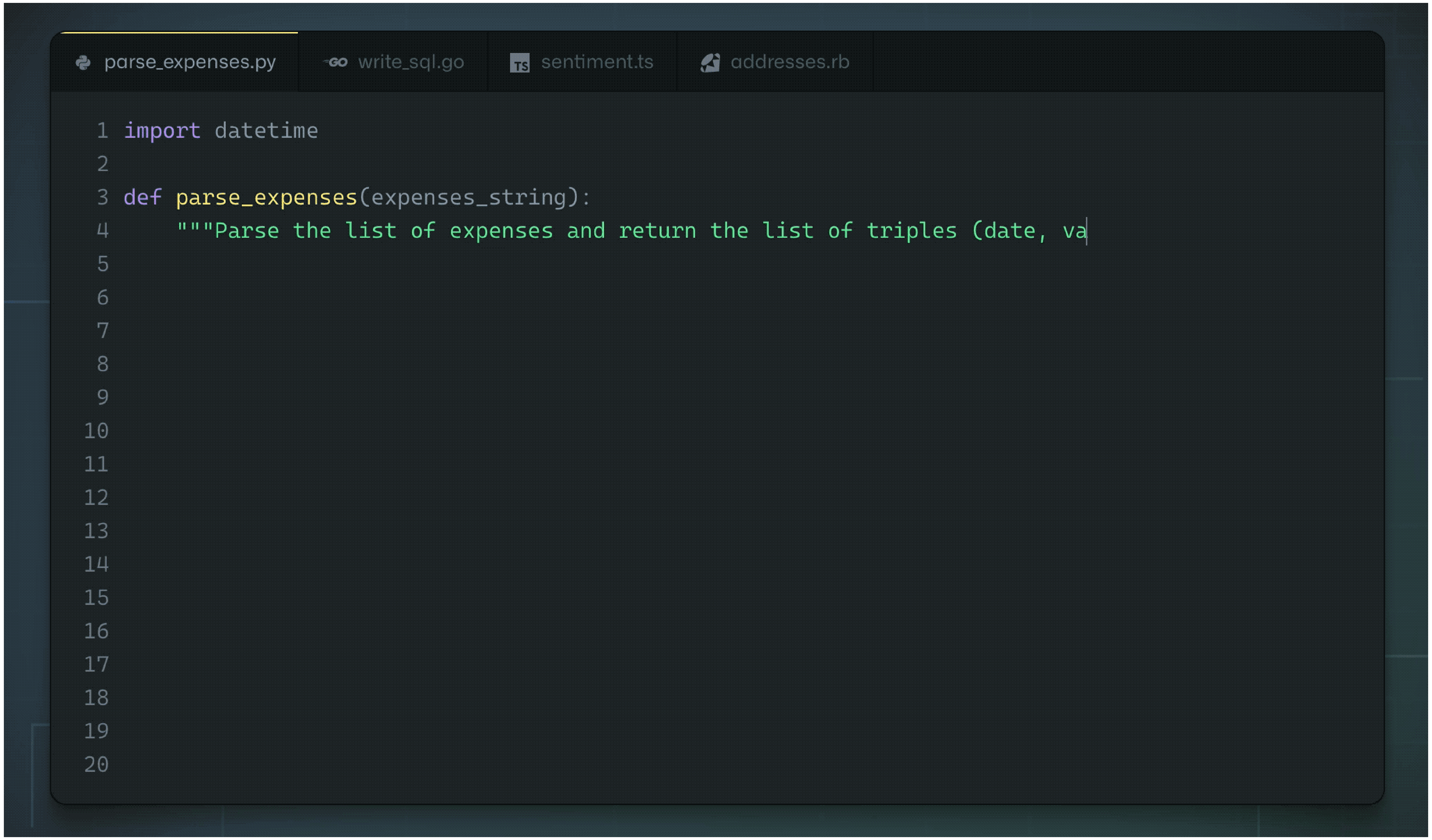
Neurocomputing is

- the leading topic of the next century.
- now more popular than a year ago, wit…
- a new field of study that explores the w…

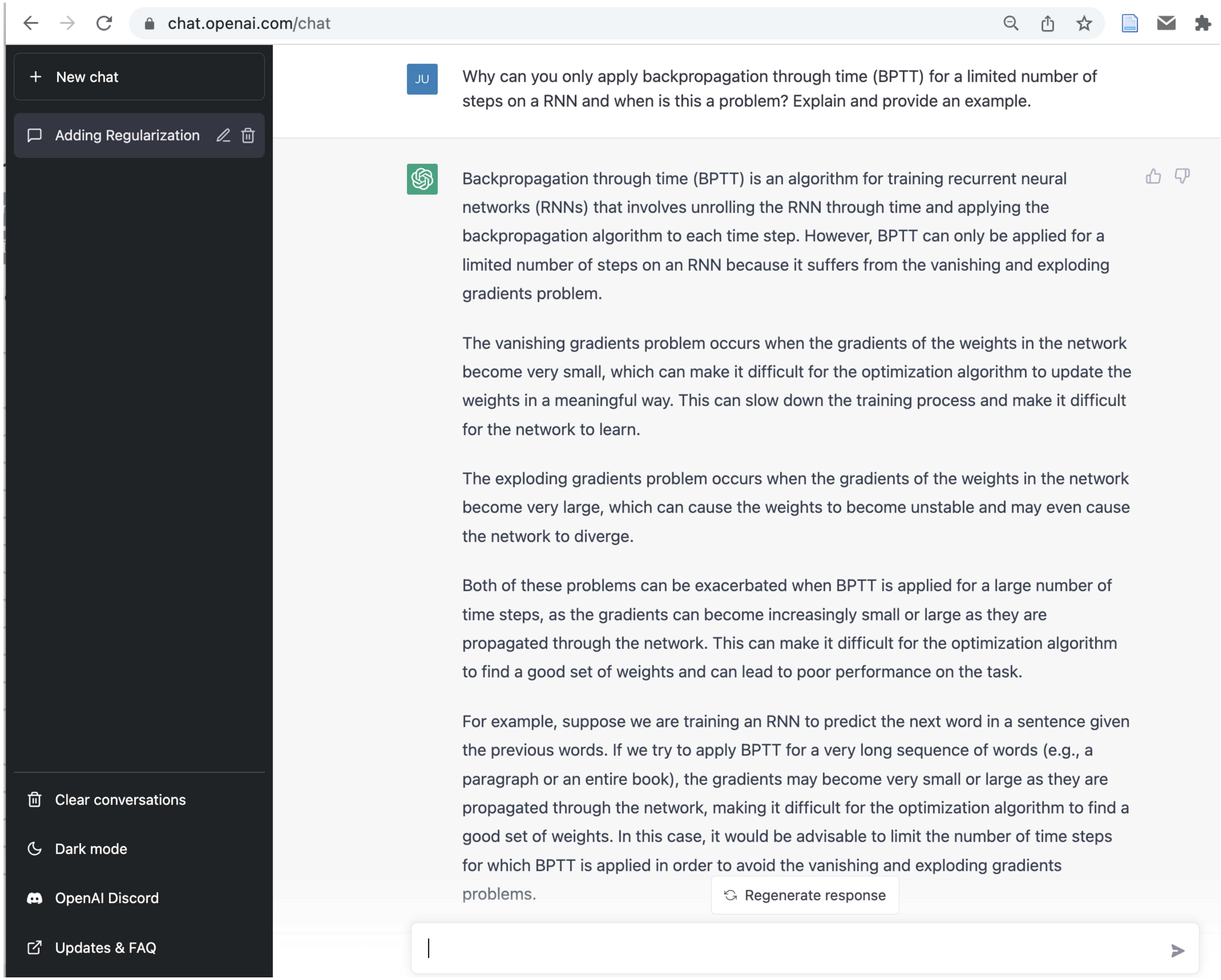
Github copilot

- Github and OpenAI trained a GPT-3-like architecture on the available open source code.
- Copilot is able to “autocomplete” the code based on a simple comment/docstring.

https://copilot.github.com/



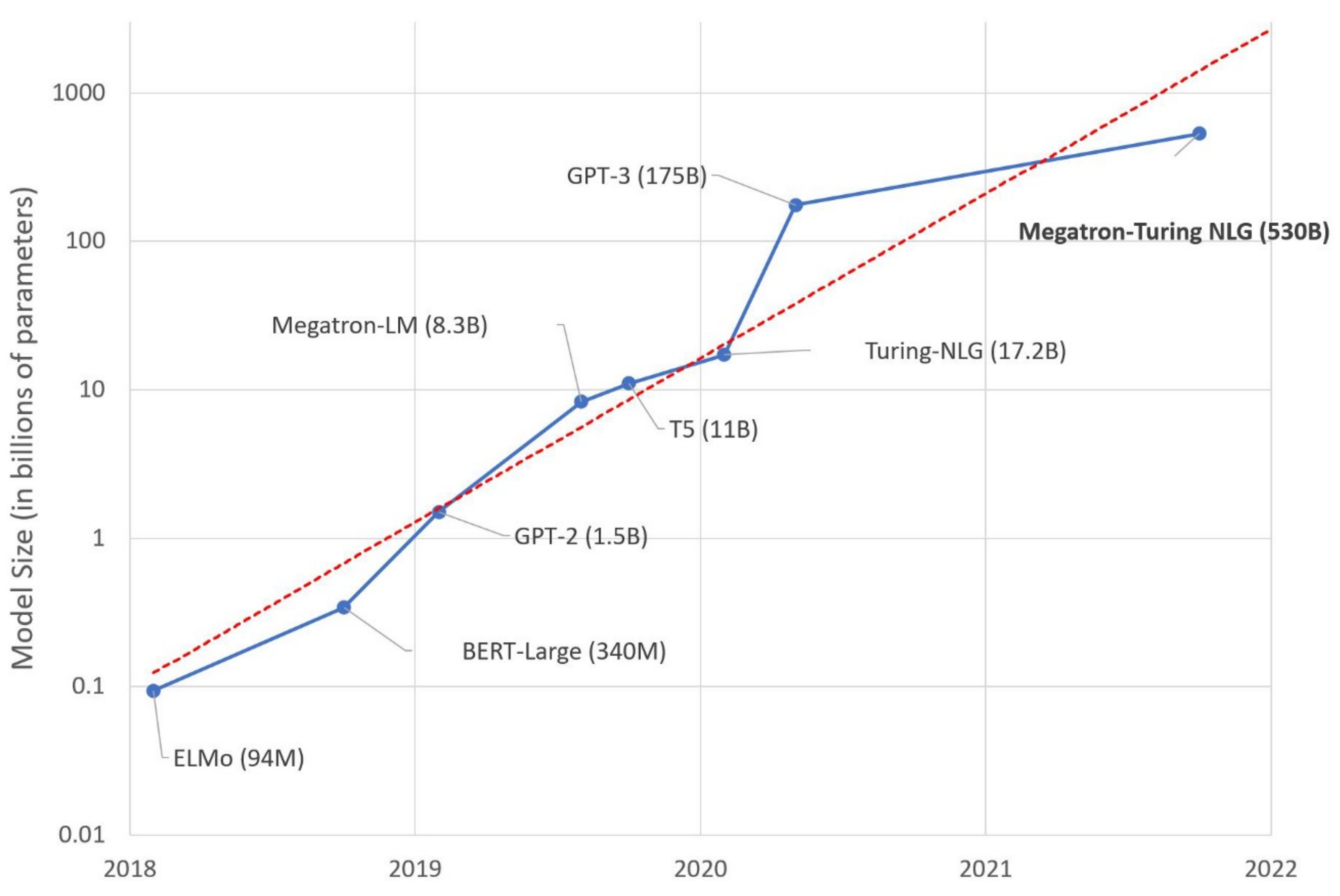
ChatGPT



https://chat.openai.com/

Transformers and NLP

- All NLP tasks (translation, sentence classification, text generation) are now done using **Large Language Models** (LLM), **self-supervisedly** pre-trained on huge corpuses.
- BERT can be used for feature extraction, while GPT is more generative.
- Transformer architectures seem to **scale**: more parameters = better performance. Is there a limit?



Source: https://julsimon.medium.com/large-language-models-a-new-moores-law-66623de5631b

- The price to pay is that these models are very expensive to train (training one instance of GPT-3 costs 12M$) and to use (GPT-3 is only accessible with an API).
- Many attempts have been made to reduce the size of these models while keeping a satisfying performance.
  - DistilBERT, RoBERTa, BART, T5, XLNet...
- See https://medium.com/mlearning-ai/recent-language-models-9fcf1b5f17f5

3 - Vision transformers

Vision transformer (ViT)

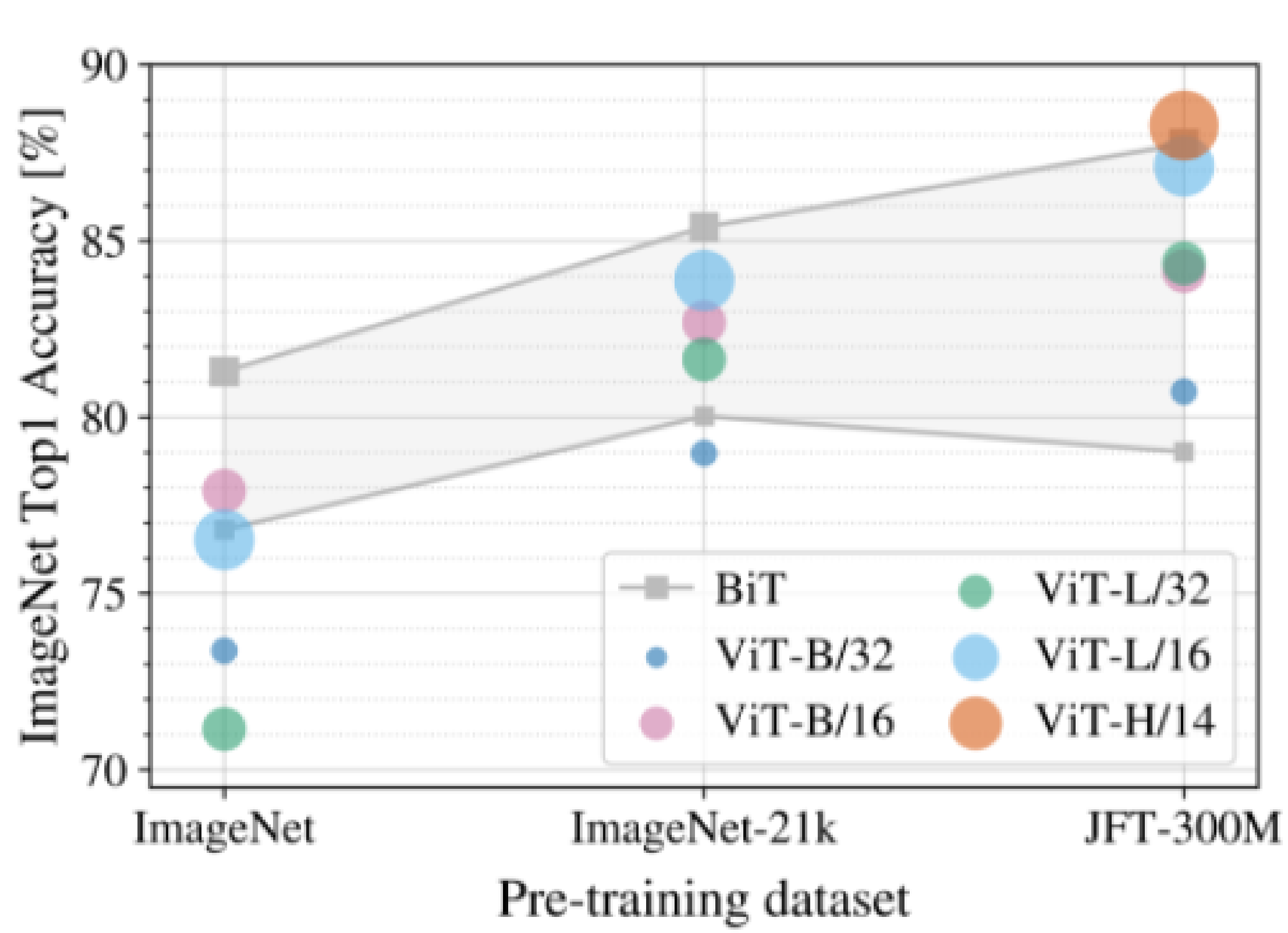
- The transformer architecture can also be applied to computer vision, by splitting images into a **sequence** of small patches (16x16).
- The sequence of vectors can then be classified by the output of the transformer using labels.

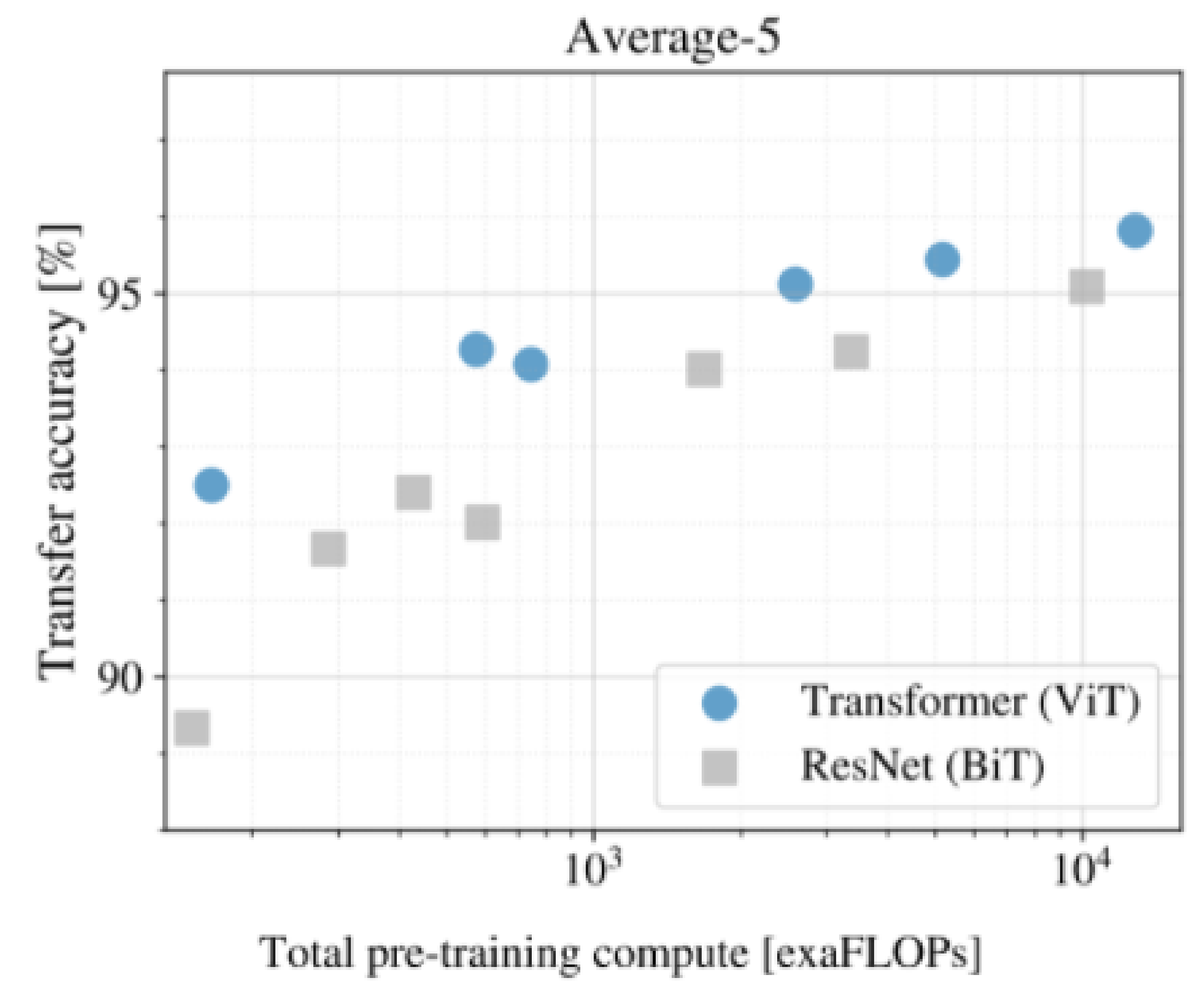
Source: https://ai.googleblog.com/2020/12/transformers-for-image-recognition-at.html

Dosovitskiy et al. (2021). An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. arXiv:201011929

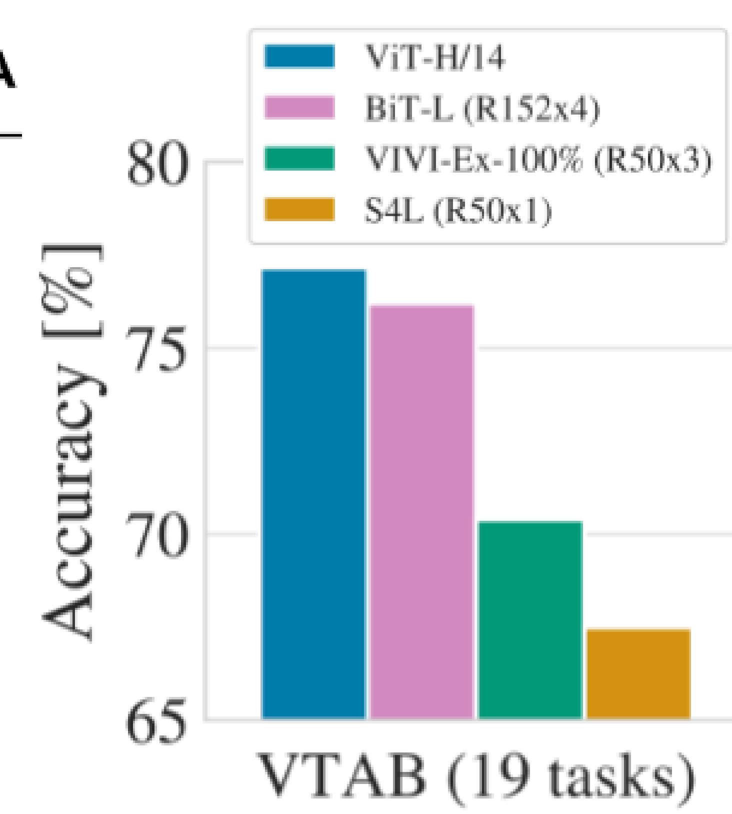
Vision transformer (ViT)

- The Vision Transformer (ViT) outperforms state-of-the-art CNNs while requiring less computations.





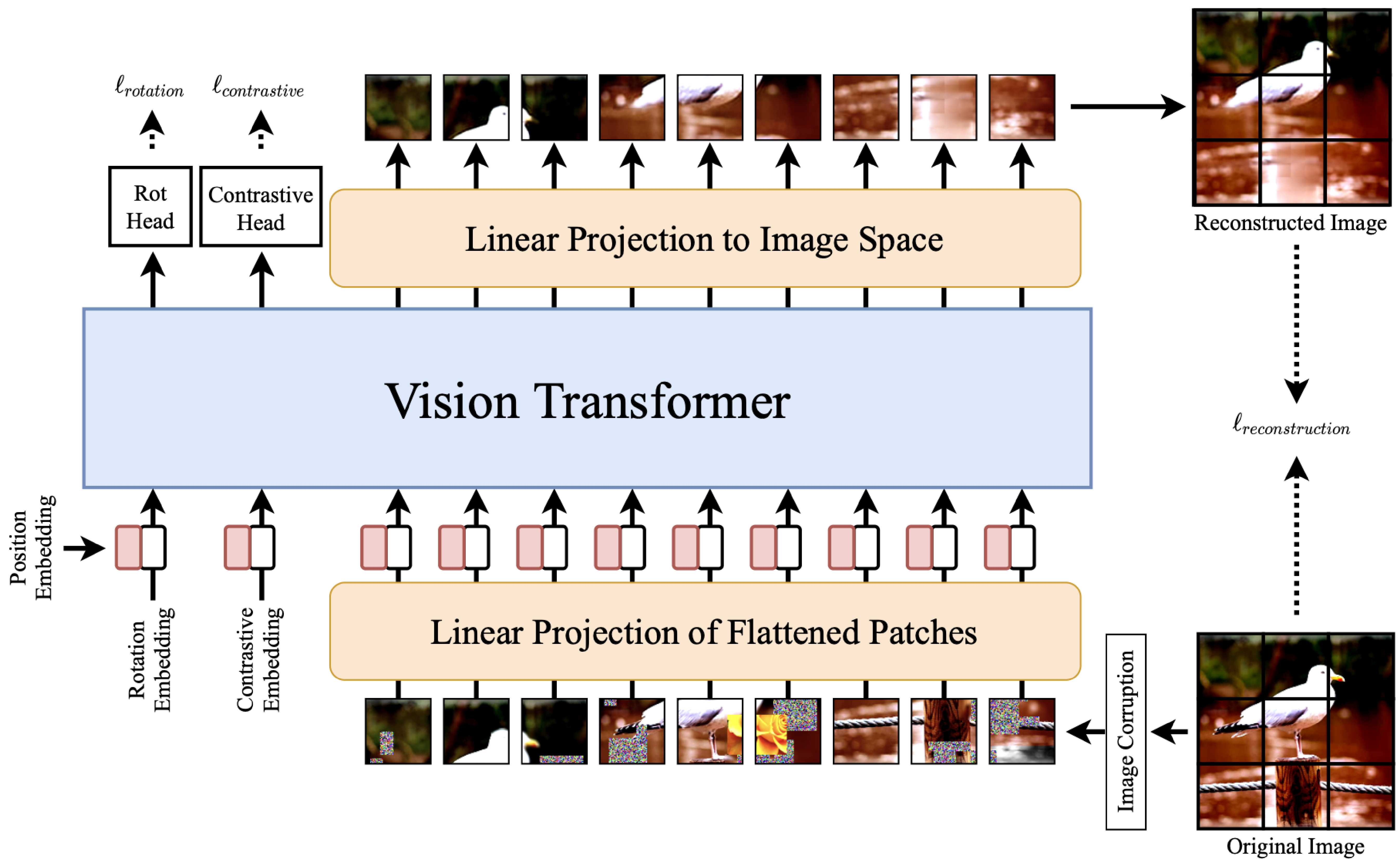
	ViT-H	Previous SOTA
ImageNet	88.55	88.5
ImageNet-ReaL	90.72	90.55
Cifar-10	99.50	99.37
Cifar-100	94.55	93.51
Pets	97.56	96.62
Flowers	99.68	99.63



https://ai.googleblog.com/2020/12/transformers-for-image-recognition-at.html

Self-supervised Vision Transformer (SiT)

- ViT only works on big supervised datasets (ImageNet). Can we benefit from self-supervised learning as in BERT or GPT?
- The Self-supervised Vision Transformer (SiT) has an denoising autoencoder-like structure, reconstructing corrupted patches autoregressively.



Self-supervised Vision Transformer (SiT)

- Self-supervised learning is possible through from **data augmentation** techniques.
- Various corruptions (masking, replacing, color distortion, blurring) are applied to the input image, but SiT must reconstruct the original image (denoising autoencoder).

Original Image | Random Drop | Random Replace | Colour Distortion | Blurring | Grey-scale

- An auxiliary **rotation loss** forces SiT to predict the orientation of the image (e.g. 30°). Another auxiliary **contrastive loss** ensures that high-level representations are different for different images.

Method	Backbone	Linear Evaluation			Domain Transfer	
		CIFAR10	CIFAR100	Tiny-ImageNet	C100→C10	C10 →C100
DeepCluster [19]	ResNet-32	$43.31\% \pm 0.62$	$20.44\% \pm 0.80$	$11.64\% \pm 0.21$	$43.39\% \pm 1.84$	$18.37\% \pm 0.41$
RotationNet [23]	ResNet-32	$62.00\% \pm 0.79$	$29.02\% \pm 0.18$	$14.73\% \pm 0.48$	$52.22\% \pm 0.70$	$27.02\% \pm 0.20$
Deep InfoMax [20]	ResNet-32	$47.13\% \pm 0.45$	$24.07\% \pm 0.05$	$17.51\% \pm 0.15$	$45.05\% \pm 0.24$	$23.73\% \pm 0.04$
SimCLR [8]	ResNet-32	$77.02\% \pm 0.64$	$42.13\% \pm 0.35$	$25.79\% \pm 0.4$	$65.59\% \pm 0.76$	$36.21\% \pm 0.16$
SimCLR [8]	ResNet-56	$78.75\% \pm 0.24$	$44.33\% \pm 0.48$	n/a	$66.19\% \pm 0.80$	$36.79\% \pm 0.45$
Relational Reasoning [21]	ResNet-32	$74.99\% \pm 0.07$	$46.17\% \pm 0.16$	$30.54\% \pm 0.42$	$67.81\% \pm 0.42$	$41.50\% \pm 0.35$
Relational Reasoning [21]	ResNet-56	$77.51\% \pm 0.00$	$47.90\% \pm 0.27$	n/a	$68.66\% \pm 0.21$	$42.19\% \pm 0.28$
SiT (ours) - Linear projection	Transformer	$81.98\% \pm 0.24$	$54.31\% \pm 0.13$	$40.35\% \pm 0.27$	$73.79\% \pm 0.15$	$55.72\% \pm 0.13$
SiT (ours) - Non-Linear projection	Transformer	$\mathbf{83.50}\% \pm 0.11$	$\mathbf{57.75}\% \pm 0.21$	$\mathbf{43.06}\% \pm 0.14$	$\mathbf{75.52}\% \pm 0.11$	$\mathbf{57.89}\% \pm 0.14$

Atito, S., Awais, M., and Kittler, J. (2021). SiT: Self-supervised vIsion Transformer. arXiv:2104.03602

Self-distillation with no labels (DINO)

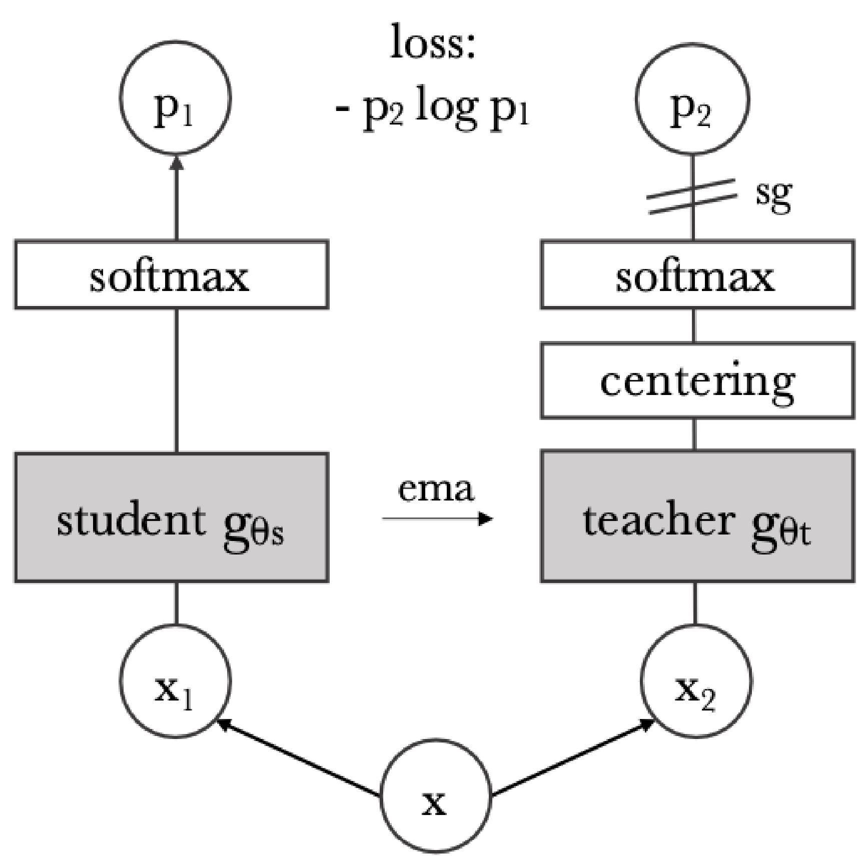
- A recent approach for self-supervised learning has been proposed by Facebook AI researchers using **self-distillation**.
- The images are split into **global** and **local patches** at different scales.
- Global patches contain label-related information (whole objects) while local patches contain finer details.

Davide Coccomini | 2021

Source: https://towardsdatascience.com/on-dino-self-distillation-with-no-labels-c29e9365e382

Self-distillation with no labels (DINO)

- The idea of **self-distillation** in DINO is to use two similar ViT networks to classify the patches.
- The **teacher** network gets the global views as an input, while the **student** network get both the local and global ones.
- Both have a MLP head to predict the softmax probabilities, but do **not** use any labels.



- The student tries to imitate the output of the teacher, by minimizing the **cross-entropy** (or KL divergence) between the two probability distributions.
- The teacher slowly integrates the weights of the student (momentum or exponentially moving average ema):

$$\theta_\text{teacher} \leftarrow \beta \, \theta_\text{teacher} + (1 - \beta) \, \theta_\text{student}$$

Caron et al. (2021). Emerging Properties in Self-Supervised Vision Transformers. arXiv:2104.14294

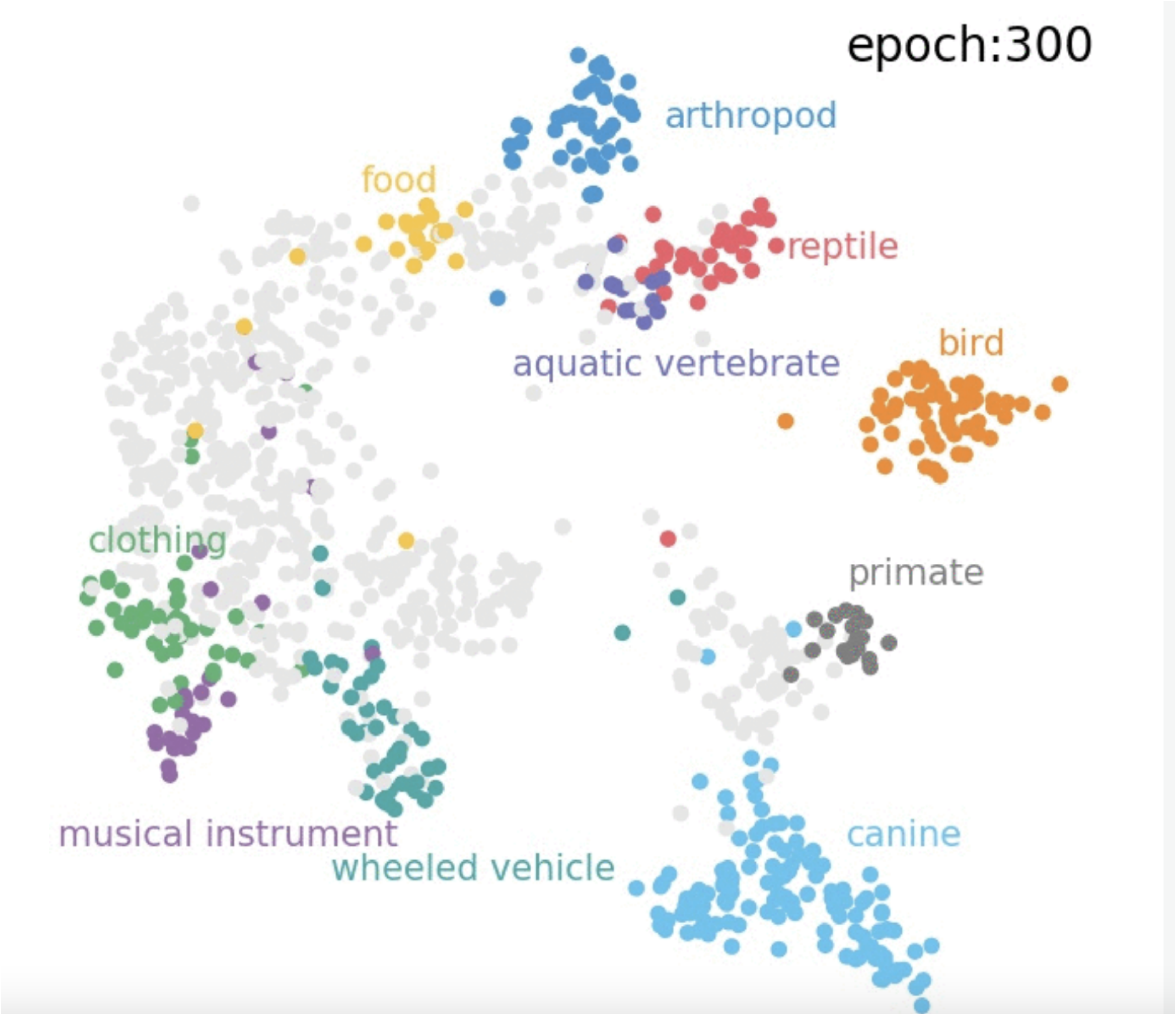
Self-distillation with no labels (DINO)

Source: https://ai.facebook.com/blog/dino-paws-computer-vision-with-self-supervised-transformers-and-10x-more-efficient-training/

Caron, M., Touvron, H., Misra, I., Jégou, H., Mairal, J., Bojanowski, P., et al. (2021). Emerging Properties in Self-Supervised Vision Transformers. arXiv:2104.14294

Self-distillation with no labels (DINO)

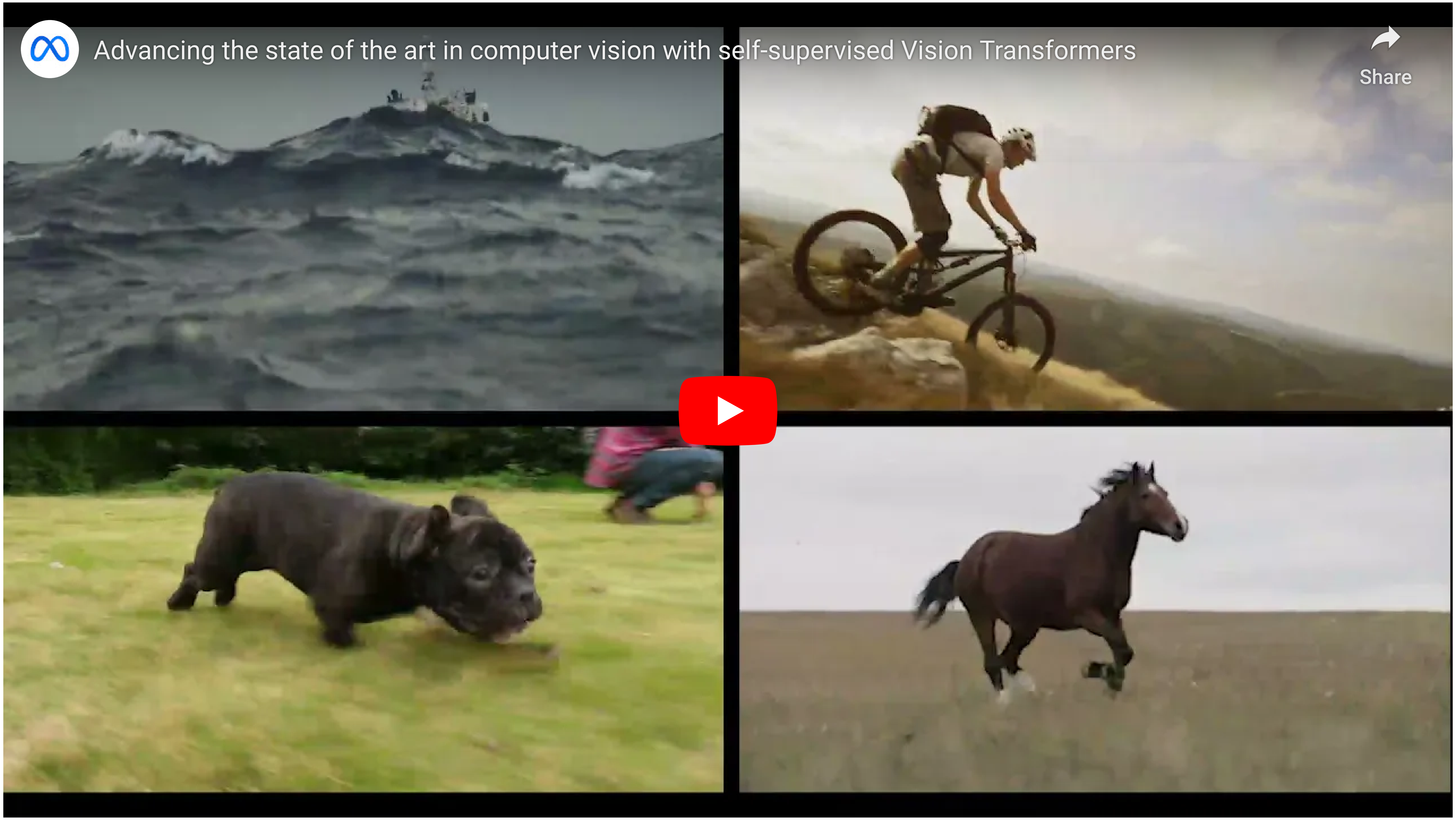
- The predicted classes do not matter when pre-training, as there is no ground truth.
- The only thing that matters is the **high-level representation** of an image before the softmax output, which can be used for transfer learning.
- Self-distillation forces the representations to be meaningful at both the global and local scales, as the teacher gets global views.
- ImageNet classes are already separated in the high-level representations: a simple kNN (k-nearest neighbour) classifier achieves 74.5% accuracy (vs. 79.3% for a supervised ResNet50).



https://ai.facebook.com/blog/dino-paws-computer-vision-with-self-supervised-transformers-and-10x-more-efficient-training

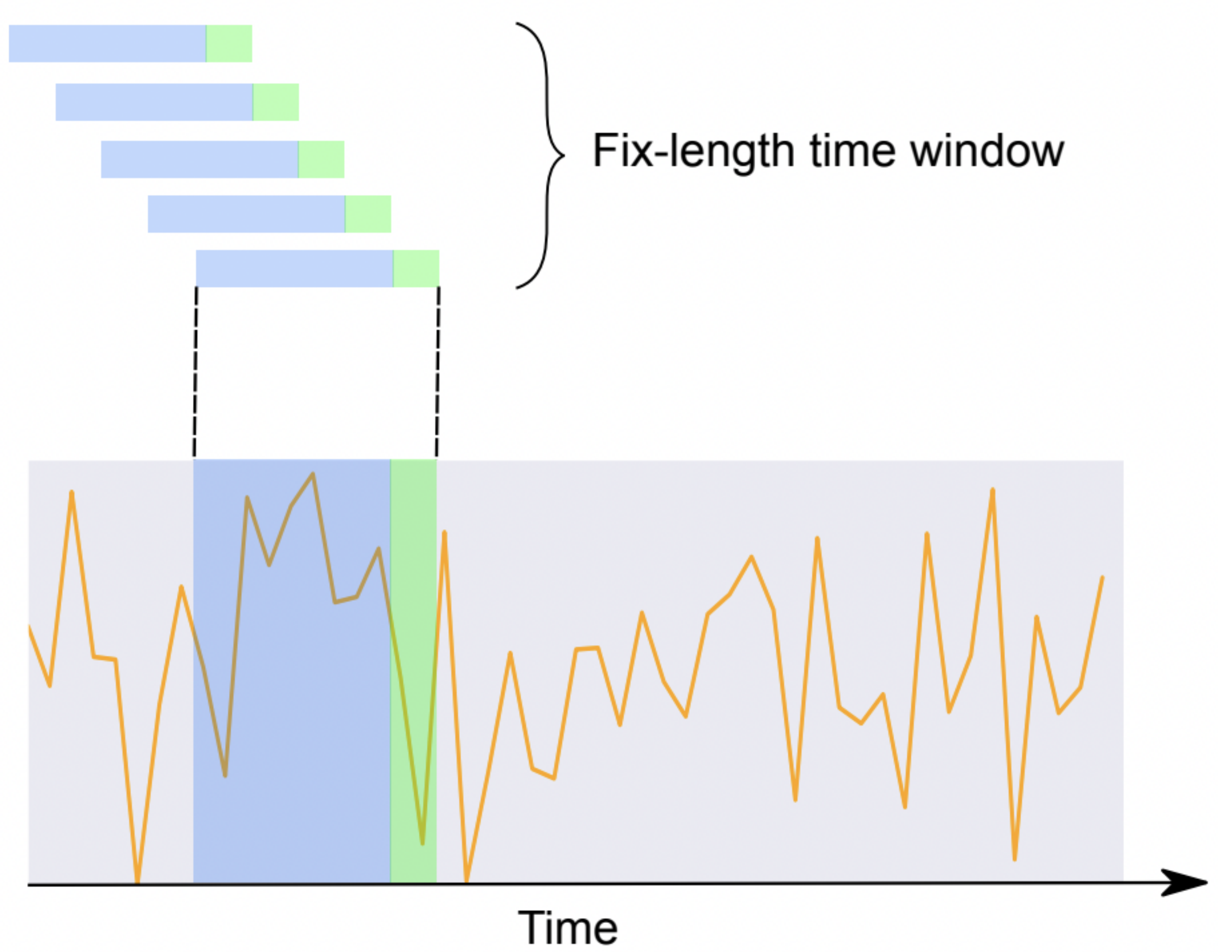
Self-distillation with no labels (DINO)

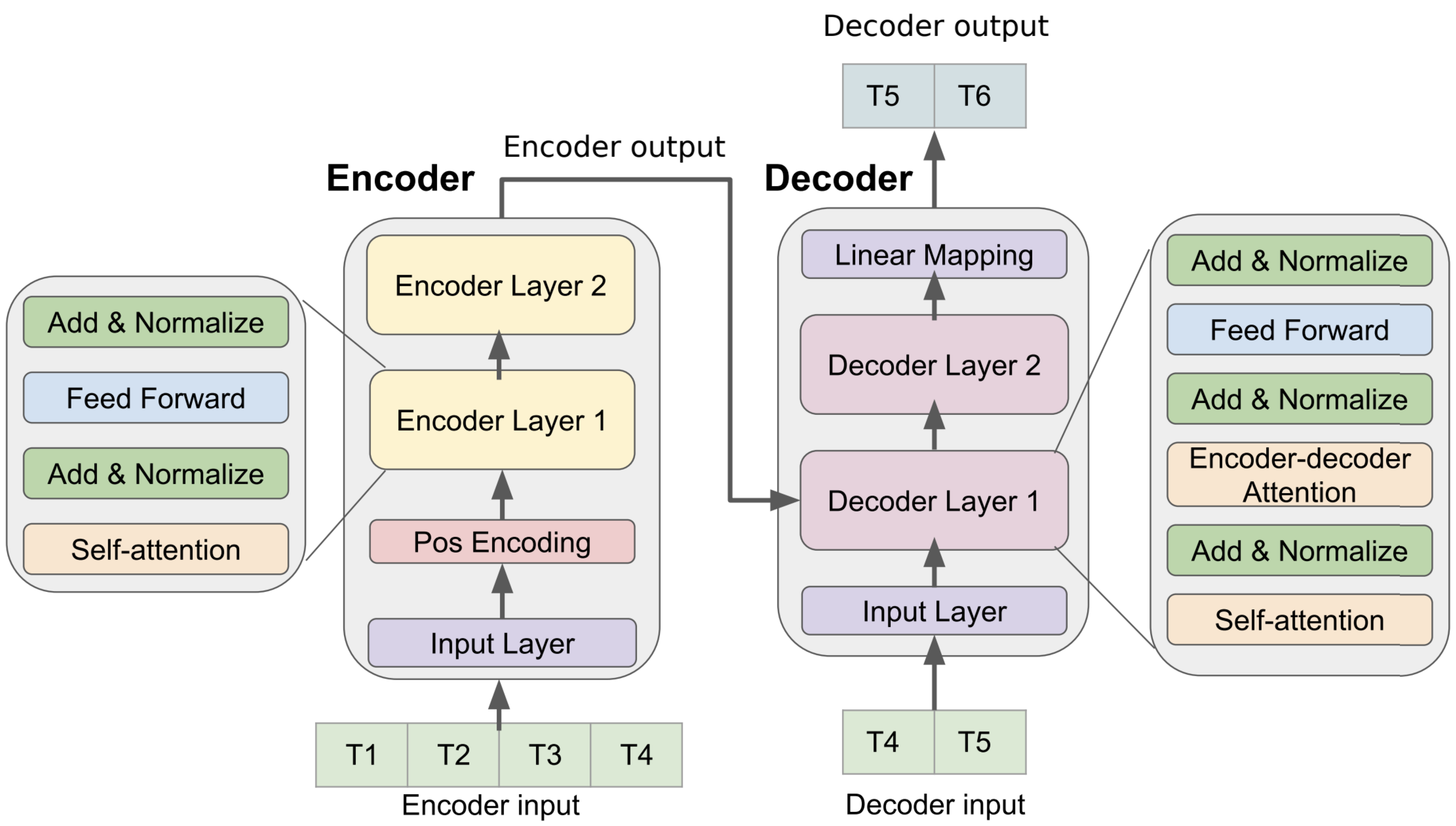
- More interestingly, by looking at the self-attention layers, one can obtain saliency maps that perform **object segmentation** without ever having been trained to!



Transformer for time series

- Transformers can also be used for time-series classification or forecasting instead of RNNs.
- Example: weather forecasting, market prices, etc.





Wu, N., Green, B., Ben, X., and O'Banion, S. (2020). Deep Transformer Models for Time Series Forecasting: The Influenza Prevalence Case. arXiv:2001.08317

References

- Various great blog posts by Jay Alammar to understand attentional networks, transformers, etc:

https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/

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

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

https://jalammar.github.io/illustrated-gpt2/

- Application of transformers outside NLP:

https://medium.com/swlh/transformers-are-not-only-for-nlp-cd837c9f175

- Extensions of the Vision Transformer:

https://theaisummer.com/transformers-computer-vision/