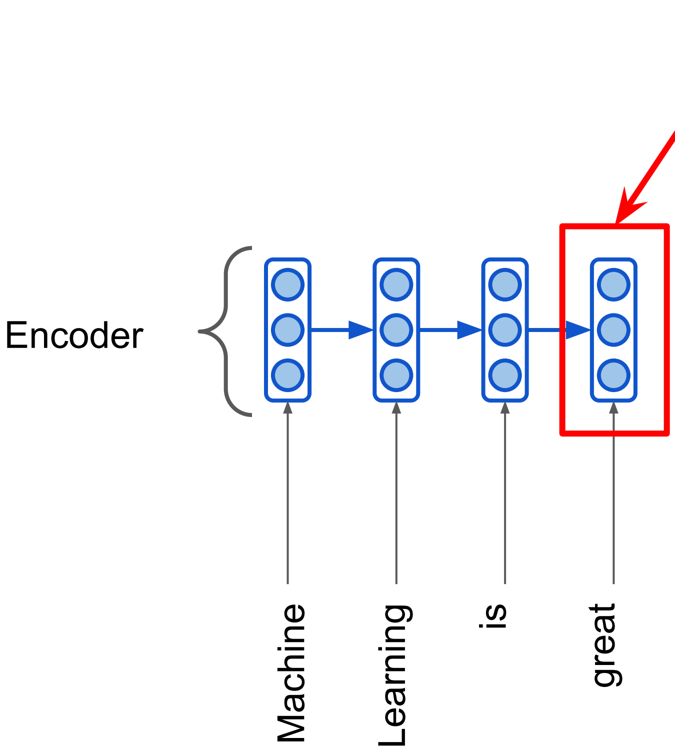
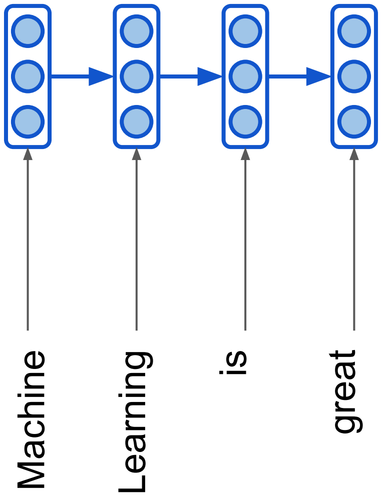
Seq2seq NMT

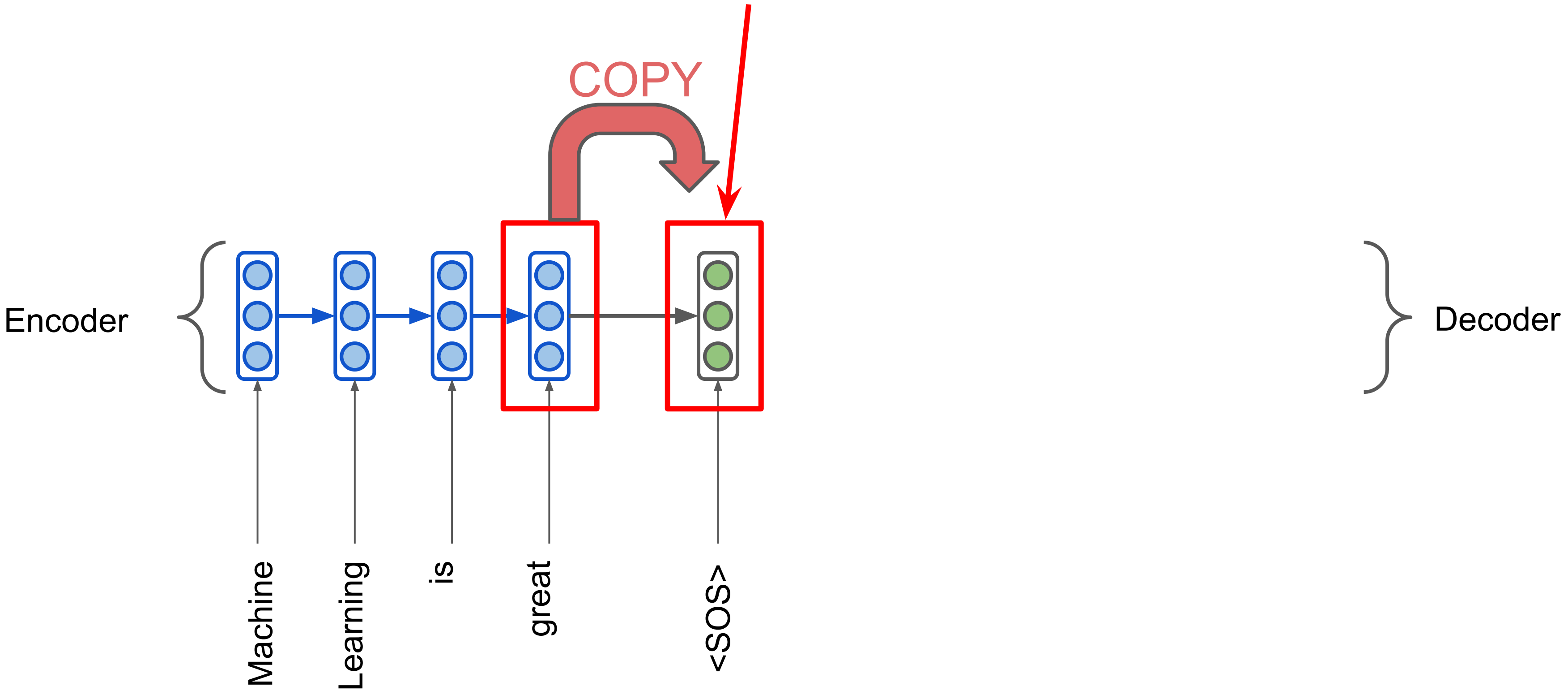
This state encodes
the whole sentence



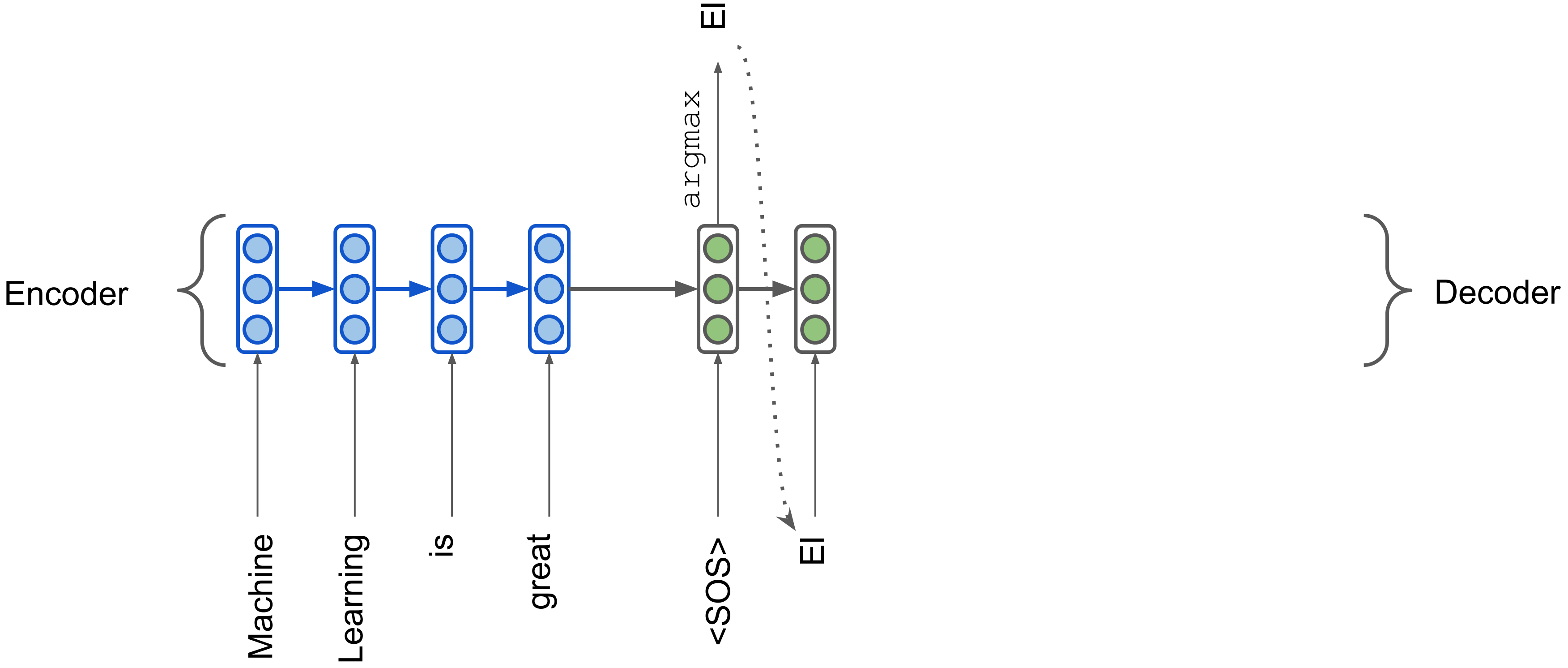

Machine
Learning
is
great

Seq2seq NMT

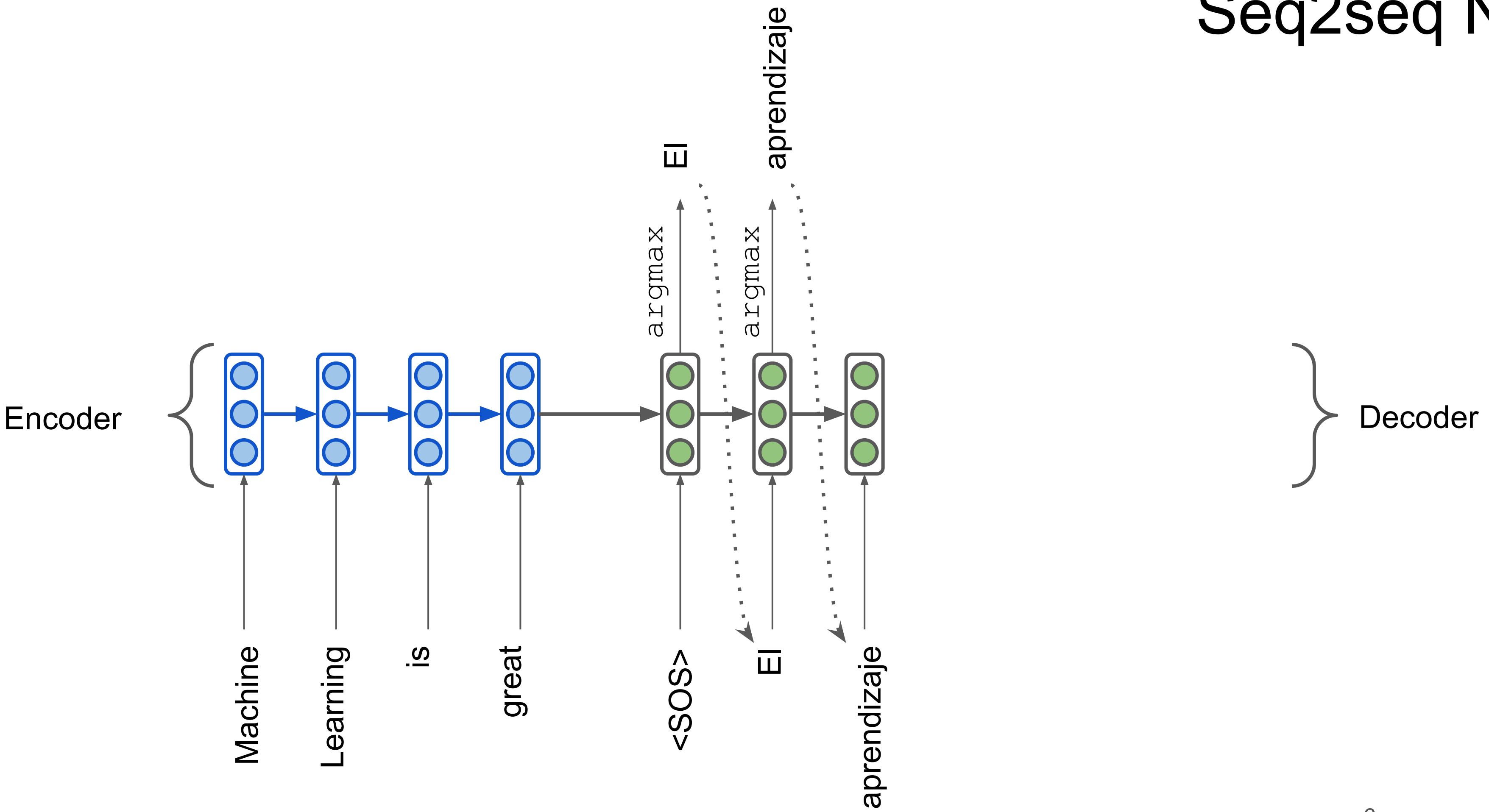
Forwarded as initial hidden state to decoder



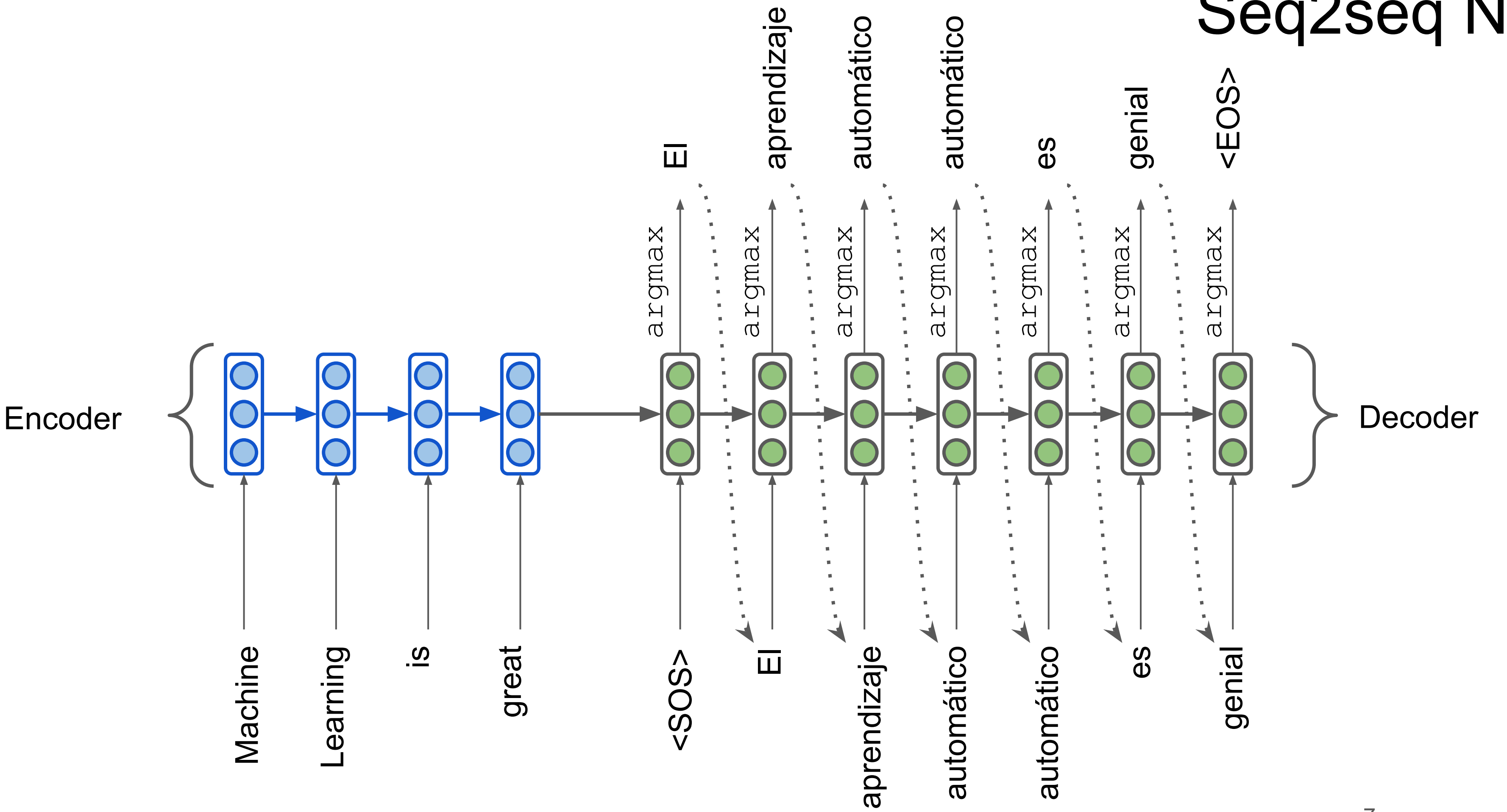
Seq2seq NMT



Seq2seq NMT

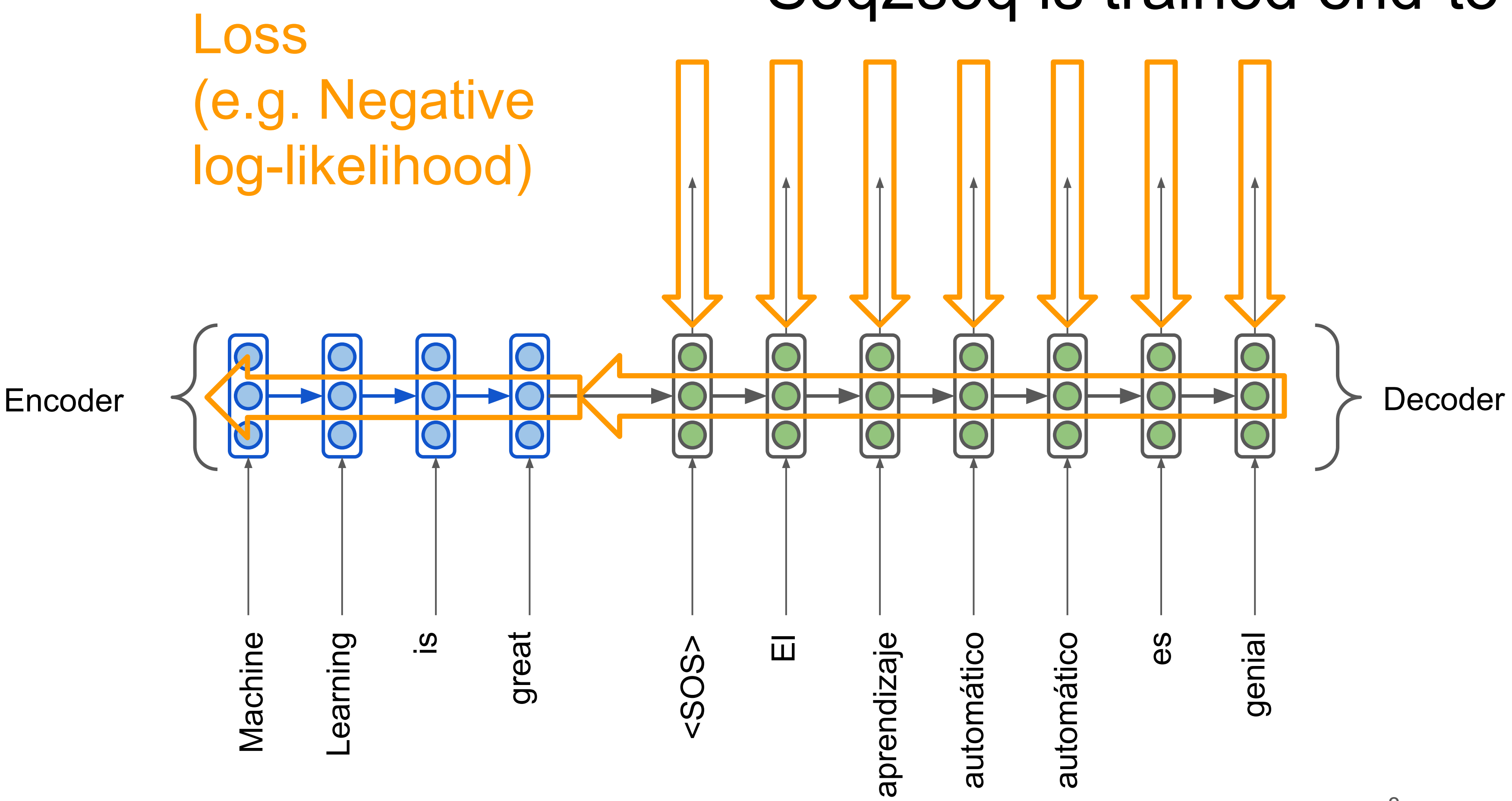

Encoder
Decoder
Machine
Learning
is
great
<SOS>
El
aprendizaje
argmax
argmax
El
aprendizaje

Seq2seq NMT

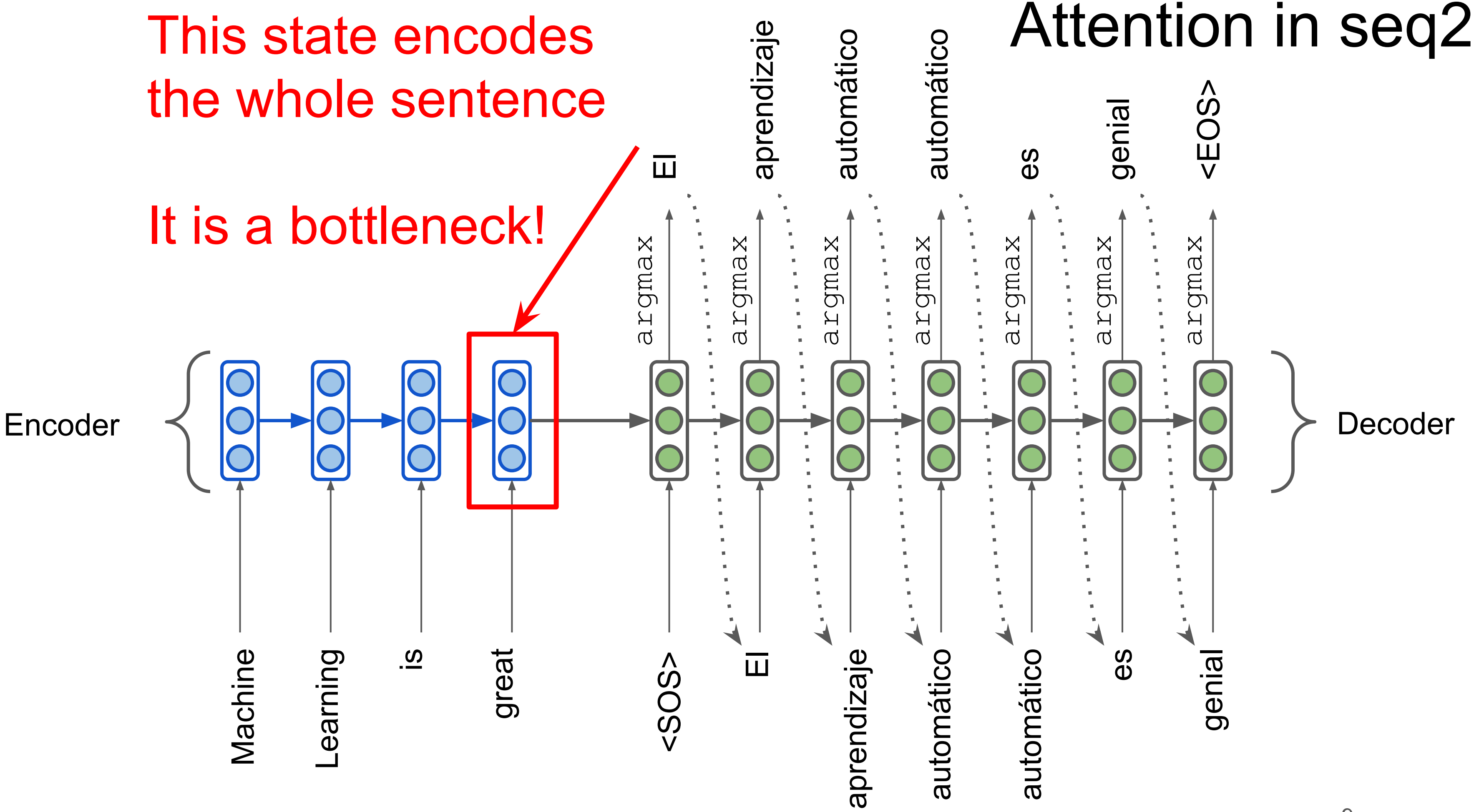


Encoder
Decoder
Machine
Learning
is
great
<SOS>
El
aprendizaje
automático
automático
es
genial
argmax
argmax
argmax
argmax
argmax
argmax
argmax
El
aprendizaje
automático
automático
es
genial
<EOS>

Seq2seq is trained end-to-end



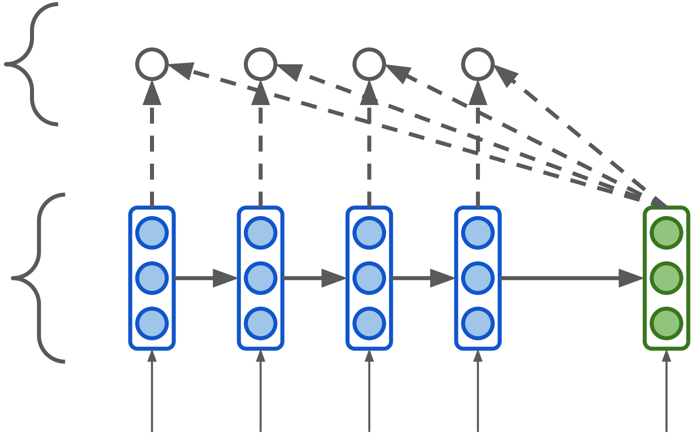
Attention in seq2seq



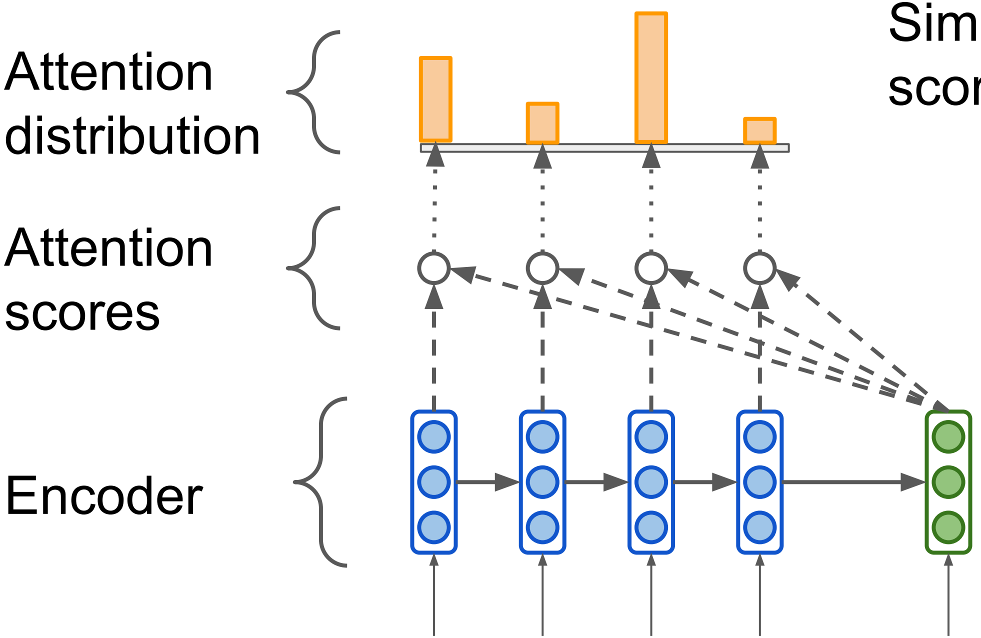
Seq2seq with attention

Attention scores

Encoder

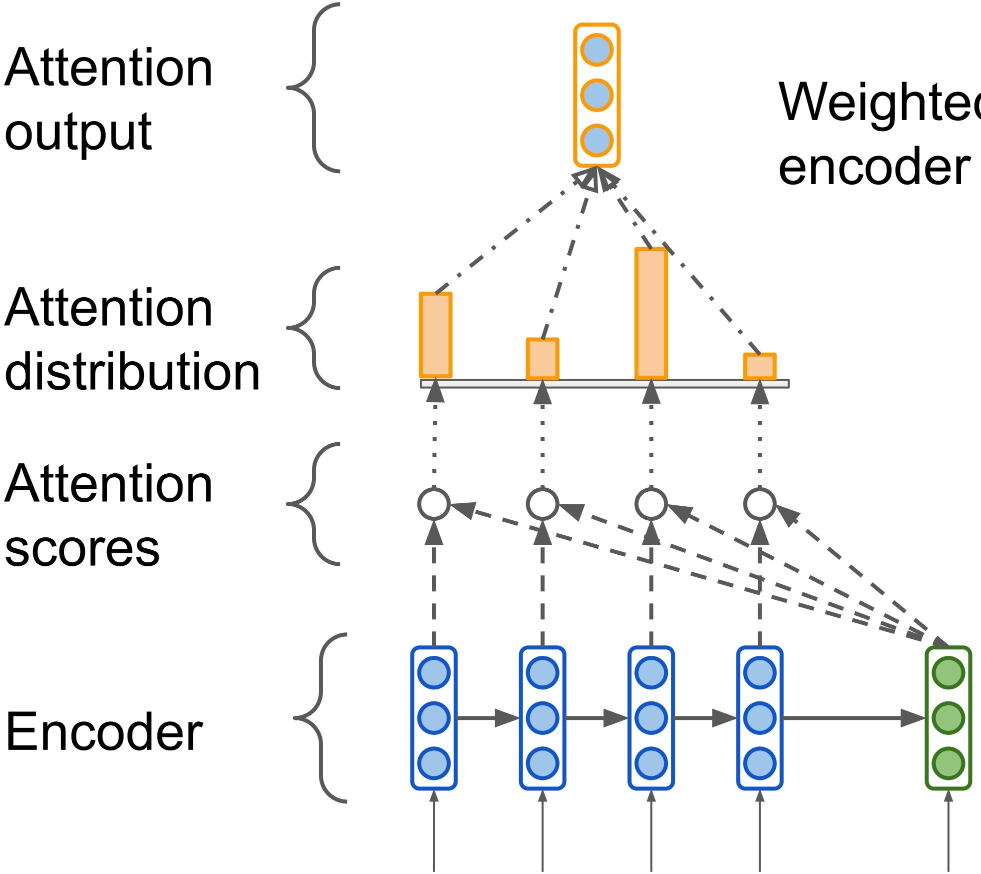


Seq2seq with attention



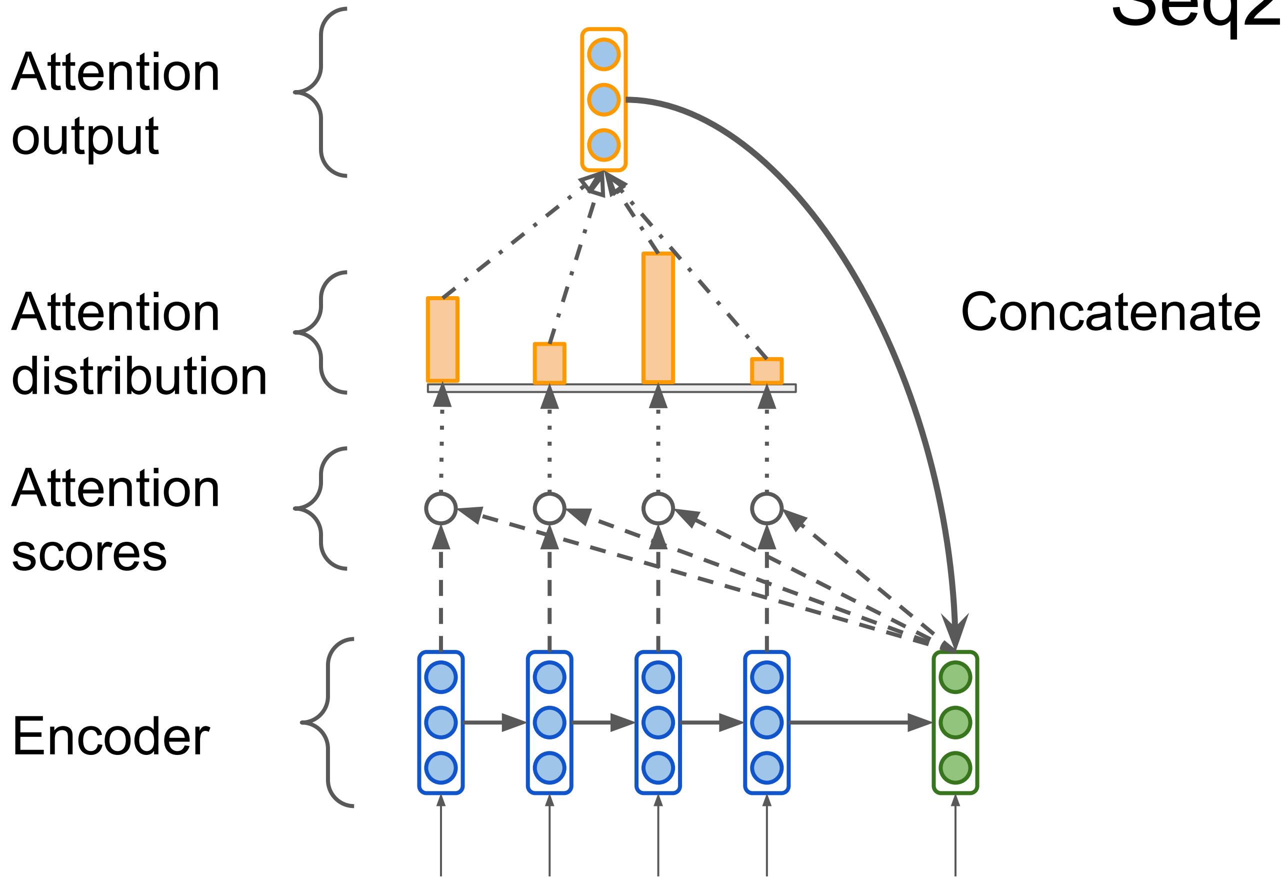
Simply apply softmax to scores

Seq2seq with attention

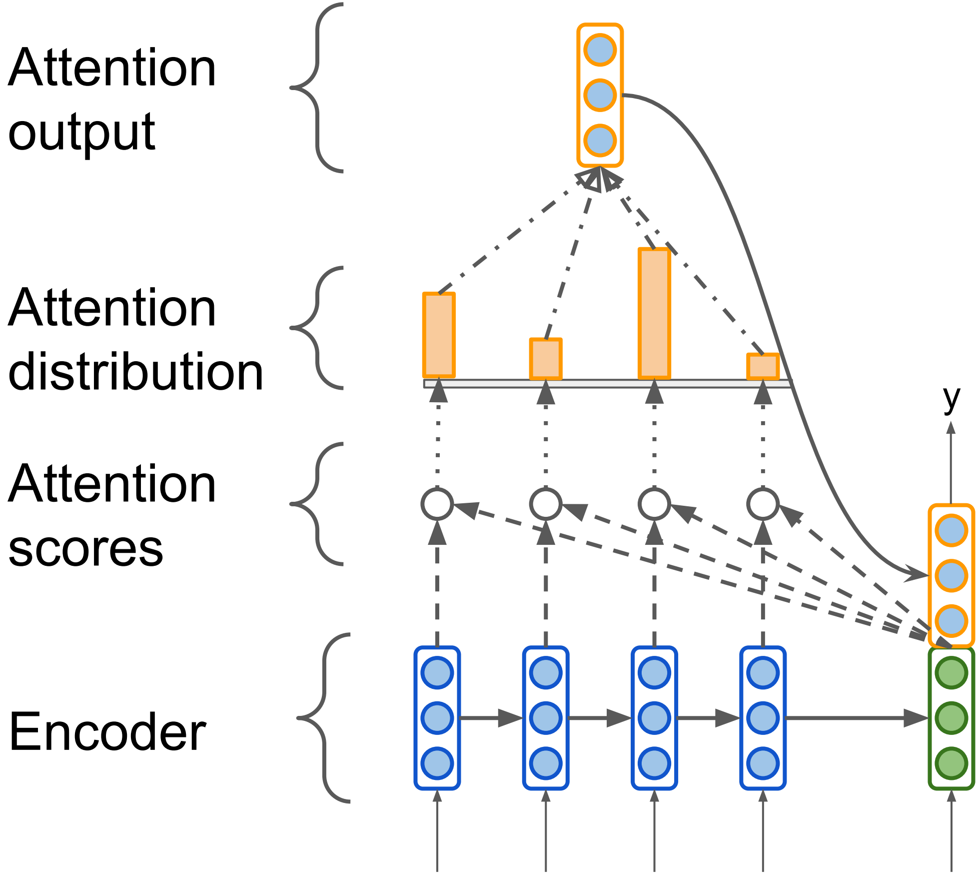


Weighted sum of all encoder states

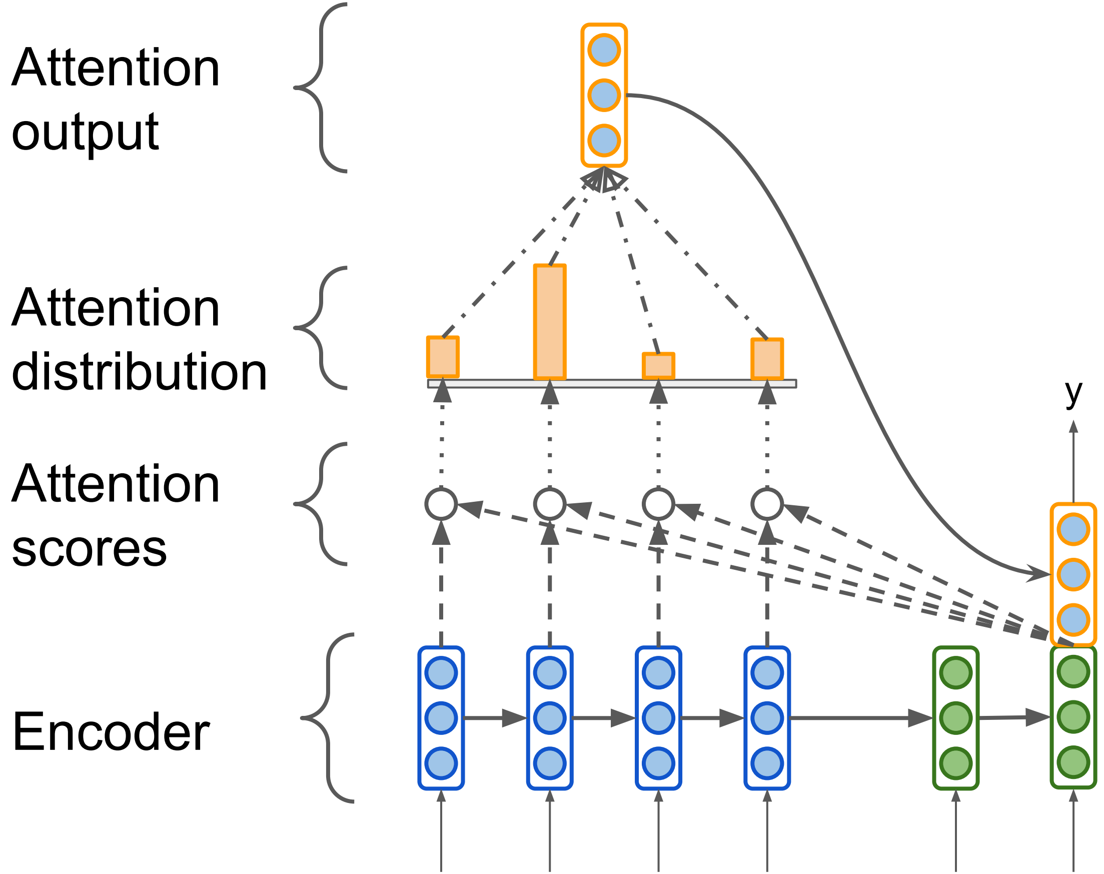
Seq2seq with attention



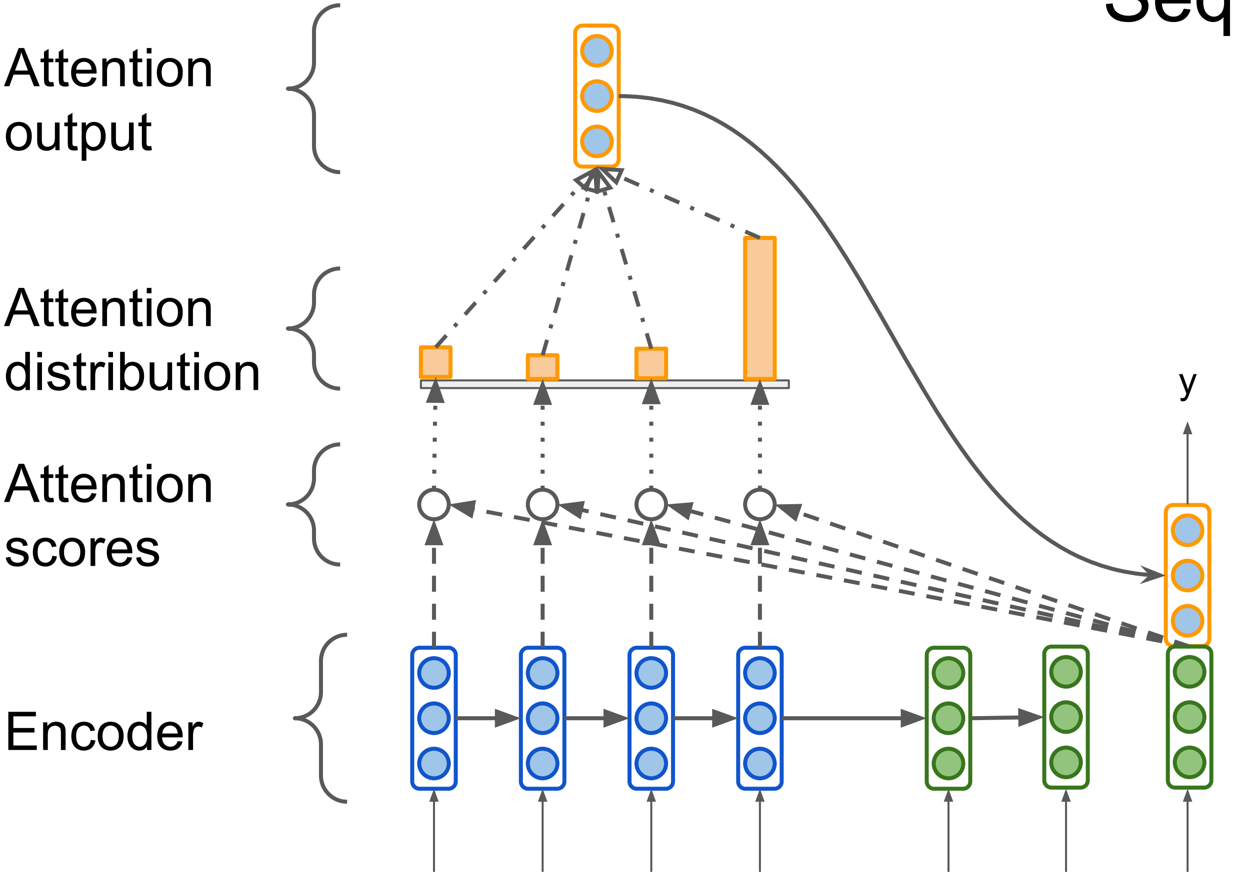
Seq2seq with attention



Seq2seq with attention



Seq2seq with attention



Attention in equations

Denote encoder hidden states $\mathbf{h}_1, \ldots, \mathbf{h}_N \in \mathbb{R}^k$

and decoder hidden state at time step t $\mathbf{s}_t \in \mathbb{R}^k$

The attention scores $\mathbf{e}^t$ can be computed as dot product

$$\mathbf{e}^t = [\mathbf{s}^T \mathbf{h}_1, \ldots, \mathbf{s}^T \mathbf{h}_N]$$

Then the attention vector is a linear combination of encoder states

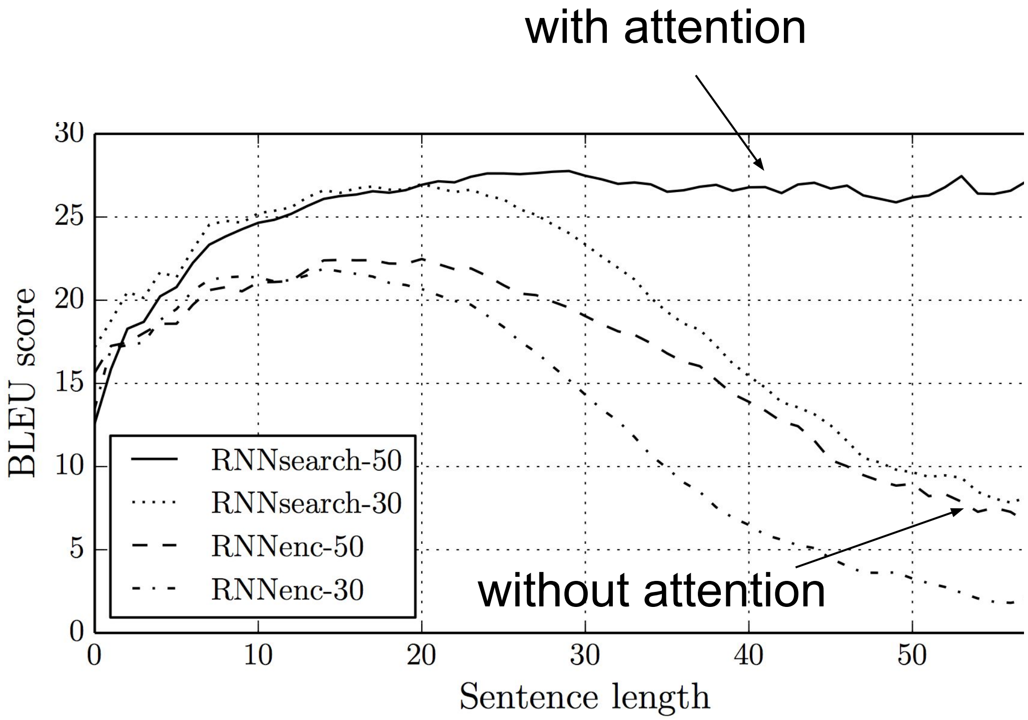
$$\mathbf{a}_t = \sum_{i=1}^{N} \boldsymbol{\alpha}_i^t \mathbf{h}_i \in \mathbb{R}^k$$, where $\boldsymbol{\alpha}_t = \mathrm{softmax}(\mathbf{e}_t)$

Attention variants

- Basic dot-product (the one discussed before): $\boldsymbol{e}_i = \boldsymbol{s}^T \boldsymbol{h}_i \in \mathbb{R}$
- Multiplicative attention: $\boldsymbol{e}_i = \boldsymbol{s}^T \boldsymbol{W} \boldsymbol{h}_i \in \mathbb{R}$
  - $\boldsymbol{W} \in \mathbb{R}^{d_2 \times d_1}$ - weight matrix
- Additive attention: $\boldsymbol{e}_i = \boldsymbol{v}^T \tanh(\boldsymbol{W}_1 \boldsymbol{h}_i + \boldsymbol{W}_2 \boldsymbol{s}) \in \mathbb{R}$
  - $\boldsymbol{W}_1 \in \mathbb{R}^{d_3 \times d_1}, \boldsymbol{W}_2 \in \mathbb{R}^{d_3 \times d_2}$ - weight matrices
  - $\boldsymbol{v} \in \mathbb{R}^{d_3}$ - weight vector

Attention advantages

- “Free” word alignment
- Better results on long sequences



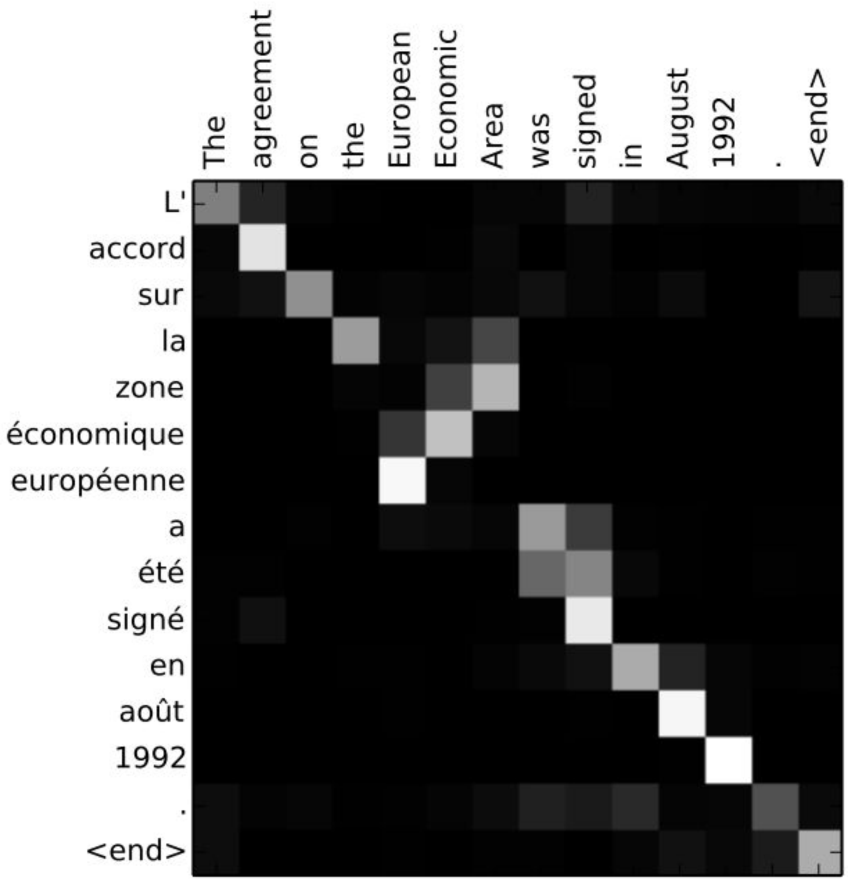


Image source: [Neural Machine Translation by Jointly Learning to Align and Translate]

The Transformer

The Transformer

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

The Transformer

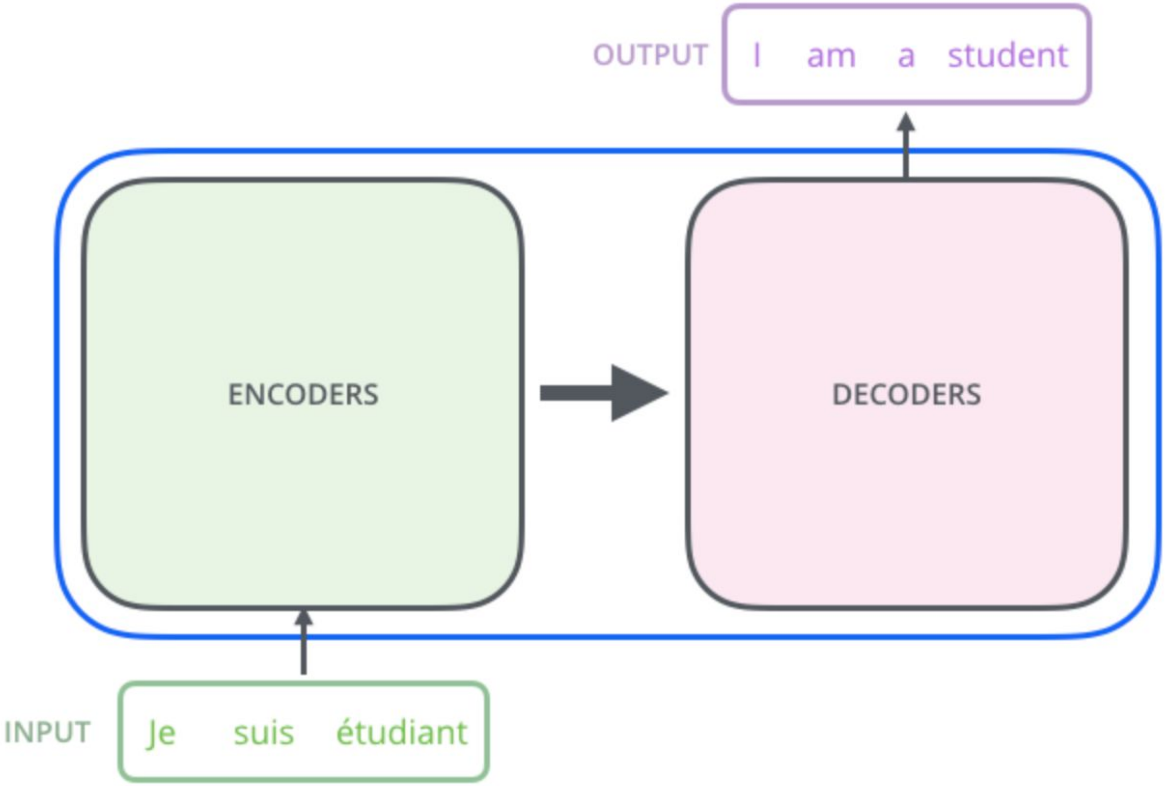


Image source: https://jalammar.github.io/illustrated-transformer/

The Transformer

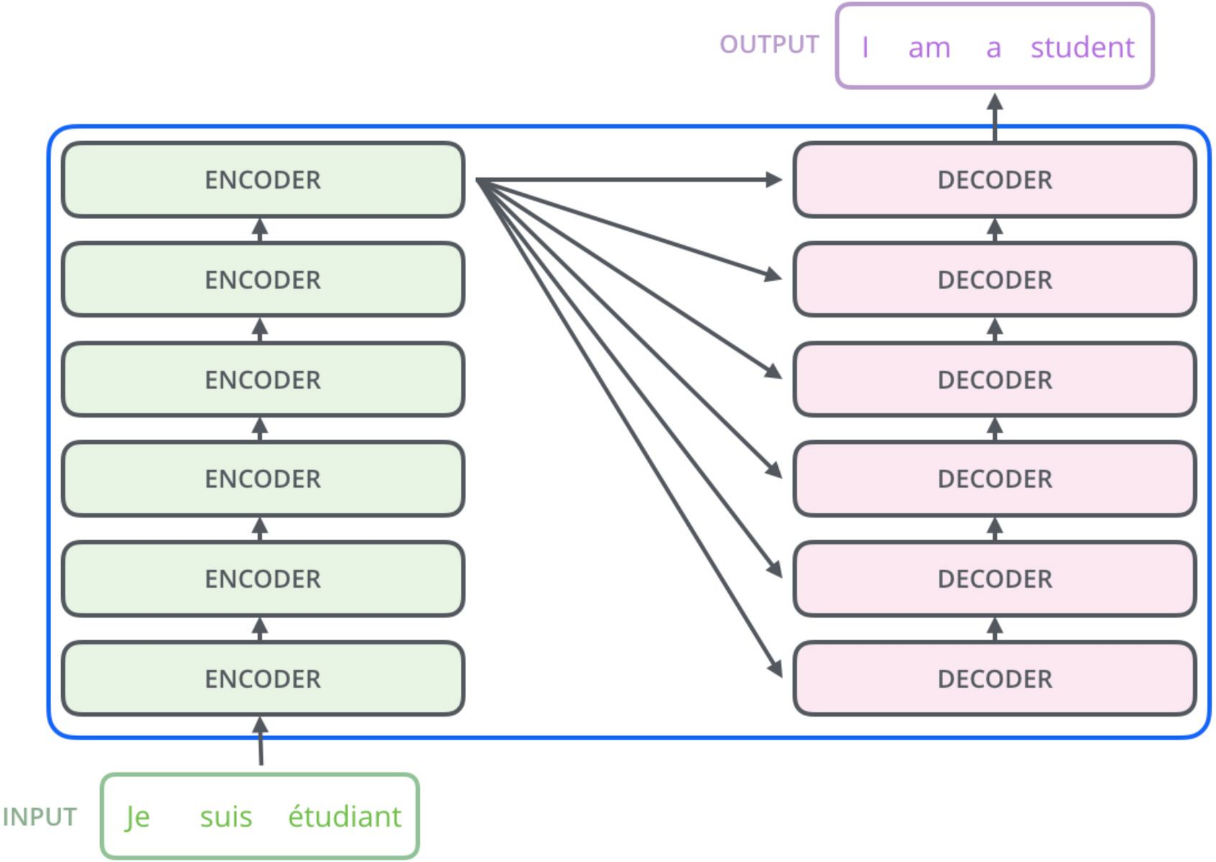
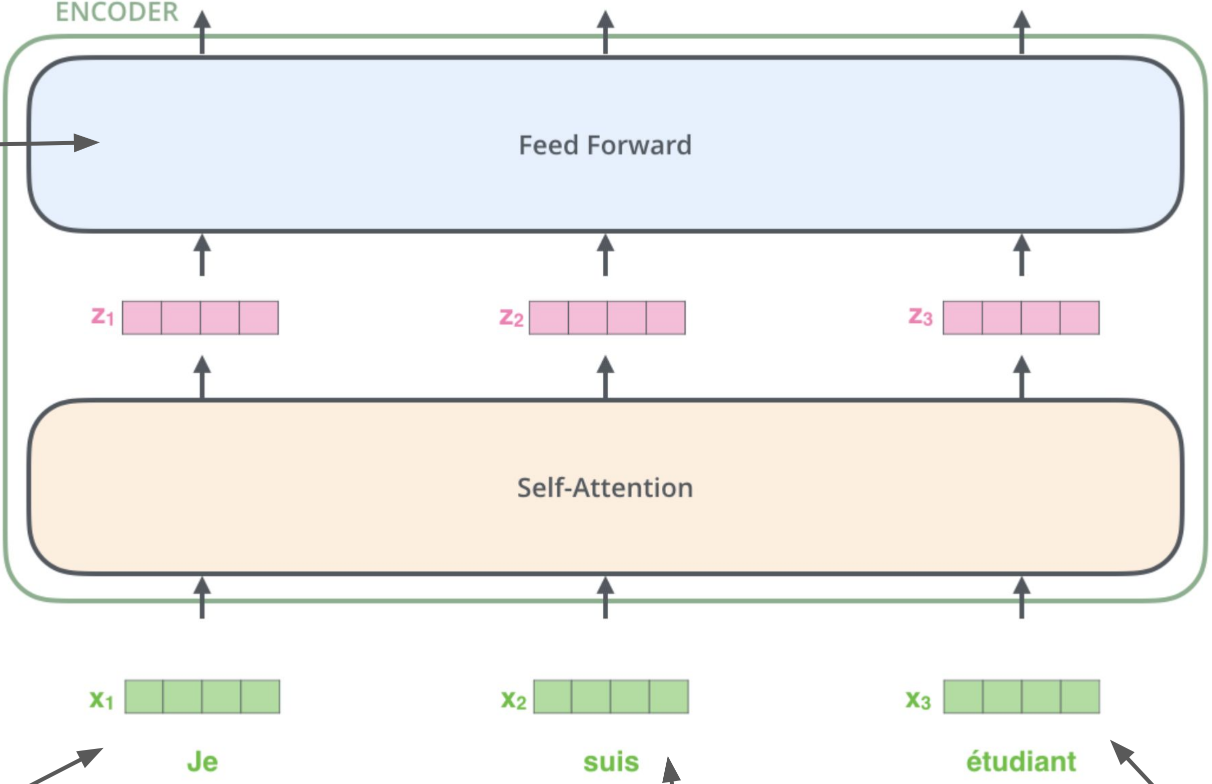


Image source: https://jalammar.github.io/illustrated-transformer/

The Encoder Side

Can be parallelized

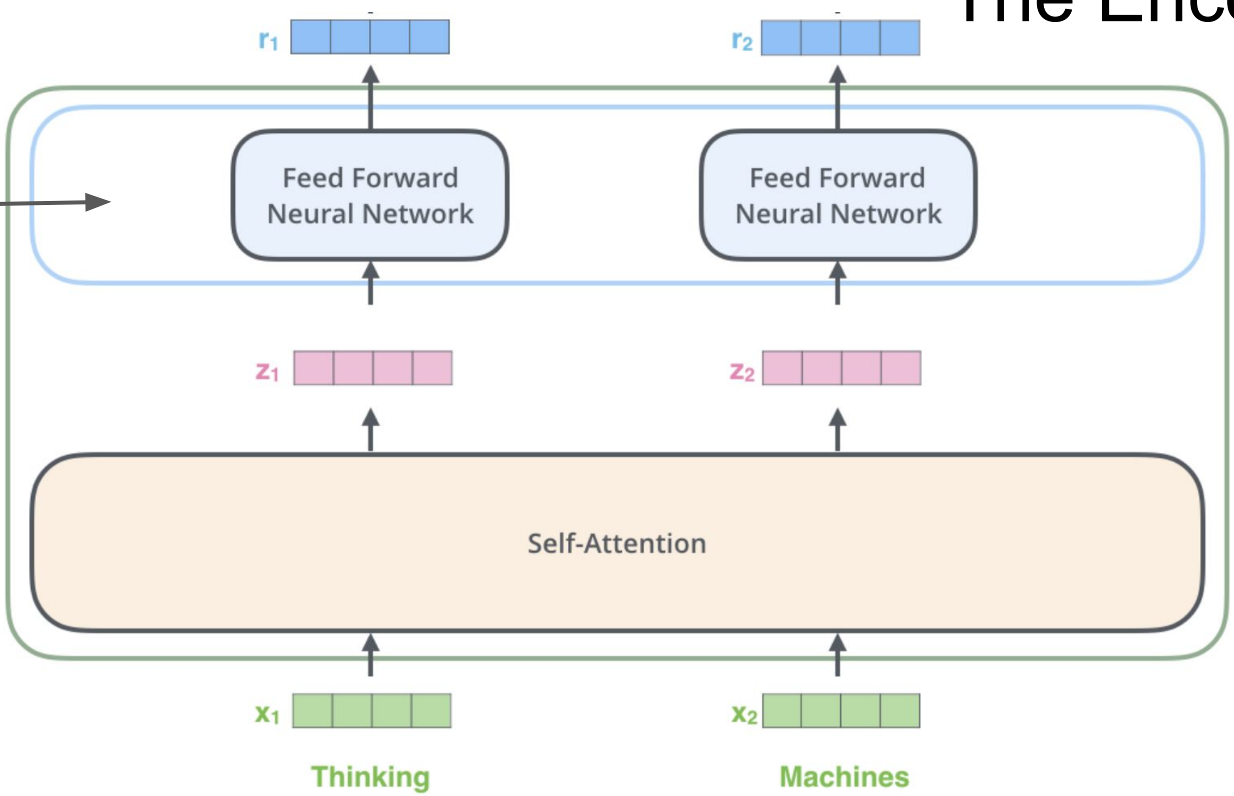


the word in each position flows through its own path in the encoder

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

The Encoder Side

Can be parallelized



the word in each position flows through its own path in the encoder

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

The Transformer: quick overview

- Proposed in 2017 in paper Attention is All You Need by Ashish Vaswani et al.
- No recurrent or convolutional layers, only attention
- Beats seq2seq in machine translation task
  - *28.4 BLEU on the WMT 2014 English-to-German translation task*
- Much faster
- Uses **self-attention** concept

Self-Attention

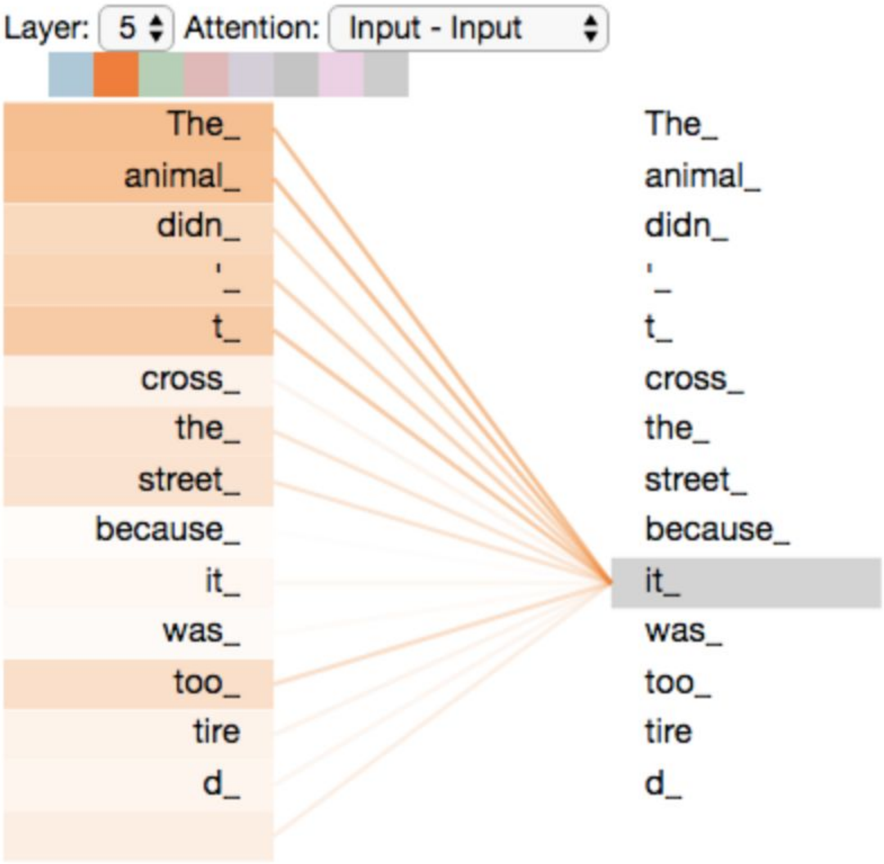
Self-Attention at a High Level

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

- What does “it” in this sentence refer to?
- We want self-attention to associate “it” with “animal”

- Self-attention is the method the Transformer uses to bake the “understanding” of other relevant words into the one we’re currently processing

Self-Attention at a High Level



Self-Attention: detailed explanation

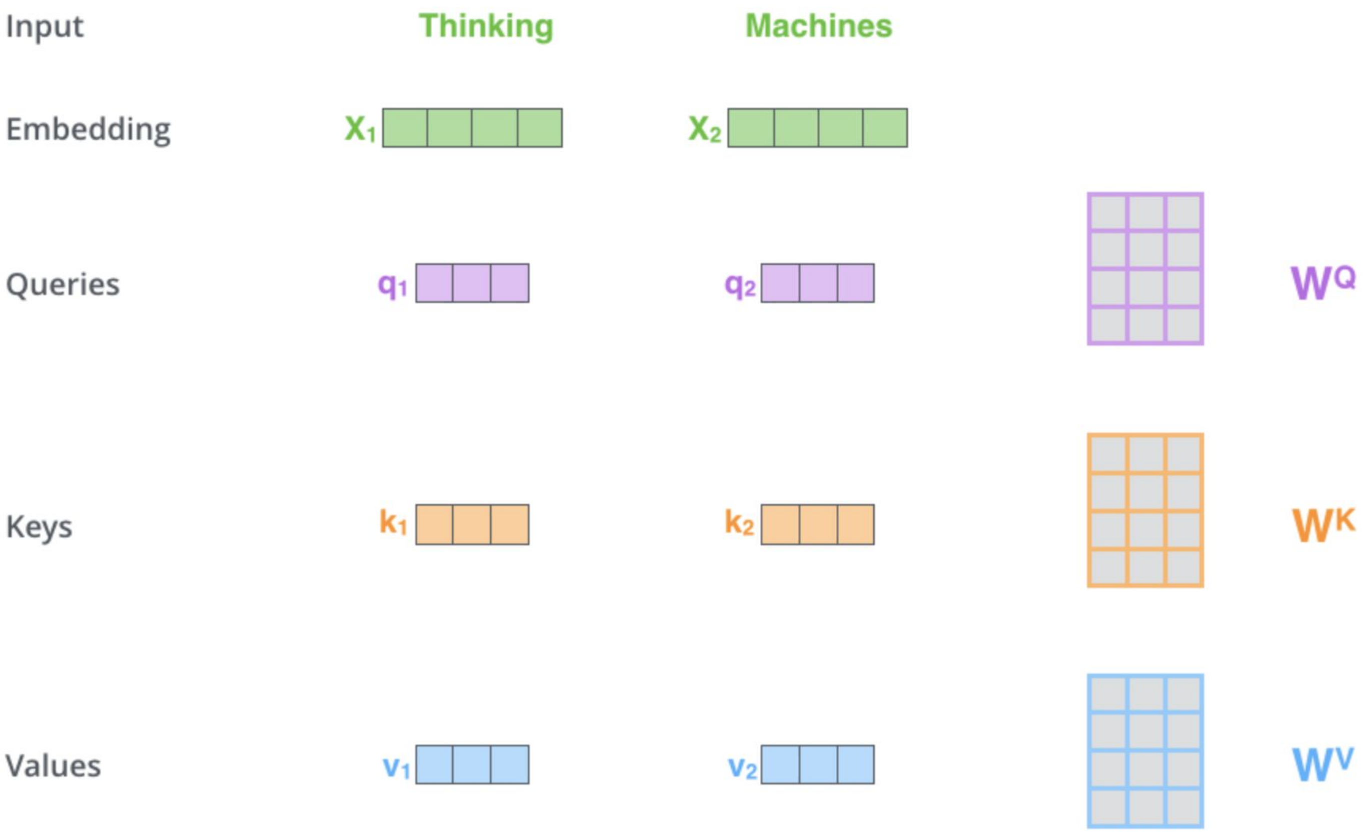


Image source: https://jalammar.github.io/illustrated-transformer/

Self-Attention: detailed explanation

STEP 1:

create 3 vectors
(**query**, **key**, **value**)

from each of the encoder's input vectors

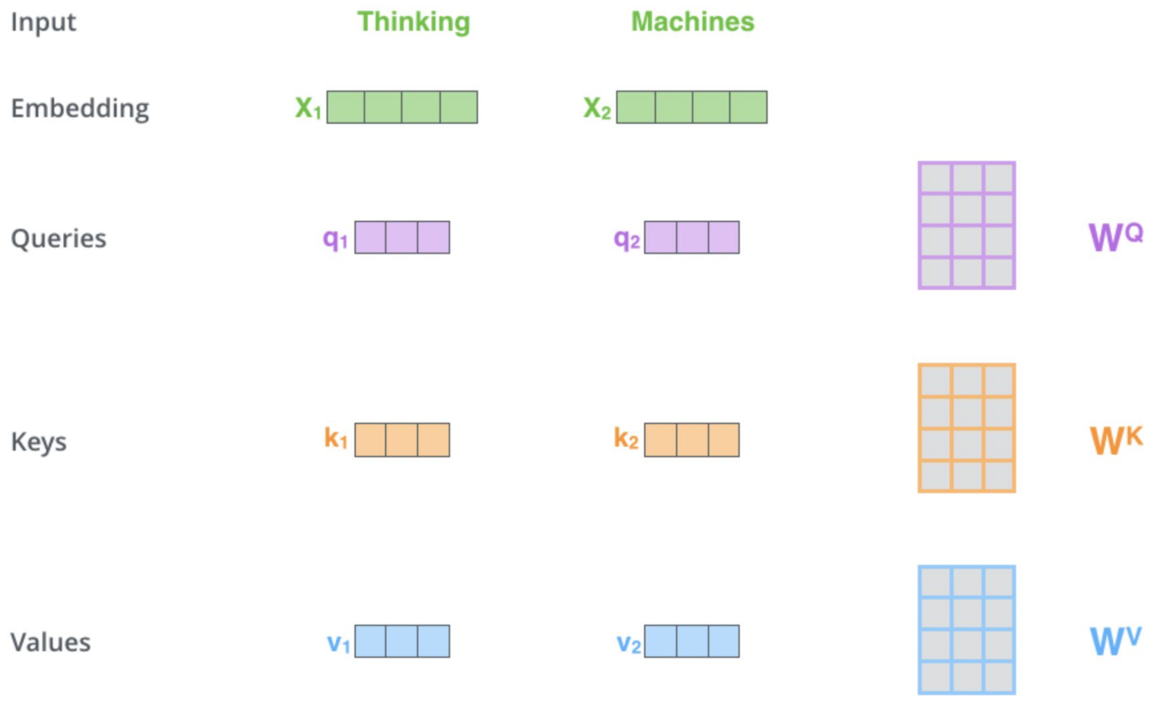


Image source: https://jalammar.github.io/illustrated-transformer/

Self-Attention: detailed explanation

What are the **query**, **key**, **value** vectors?

They’re abstractions that are useful for

calculating and thinking about attention.

Self-Attention: detailed explanation

STEP 2:

calculate a score

(score each word of the input sentence against the current word)

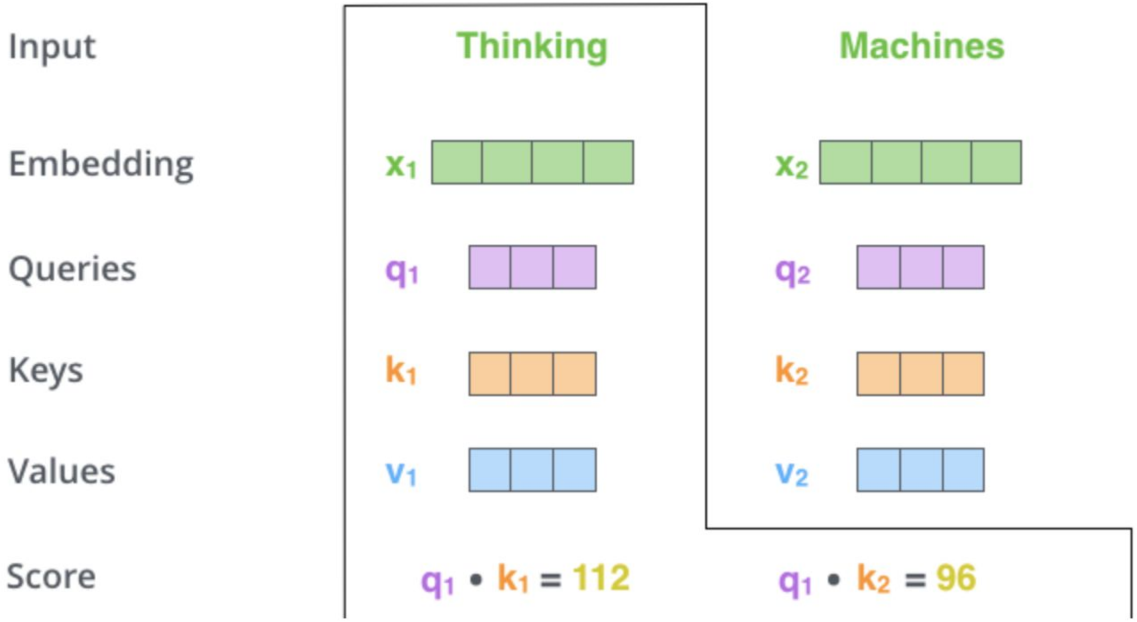


Image source: https://jalammar.github.io/illustrated-transformer/

Self-Attention: detailed explanation

STEP 3:

divide the scores by 8

(the square root of the dimension of the key vectors)

STEP 4:

softmax

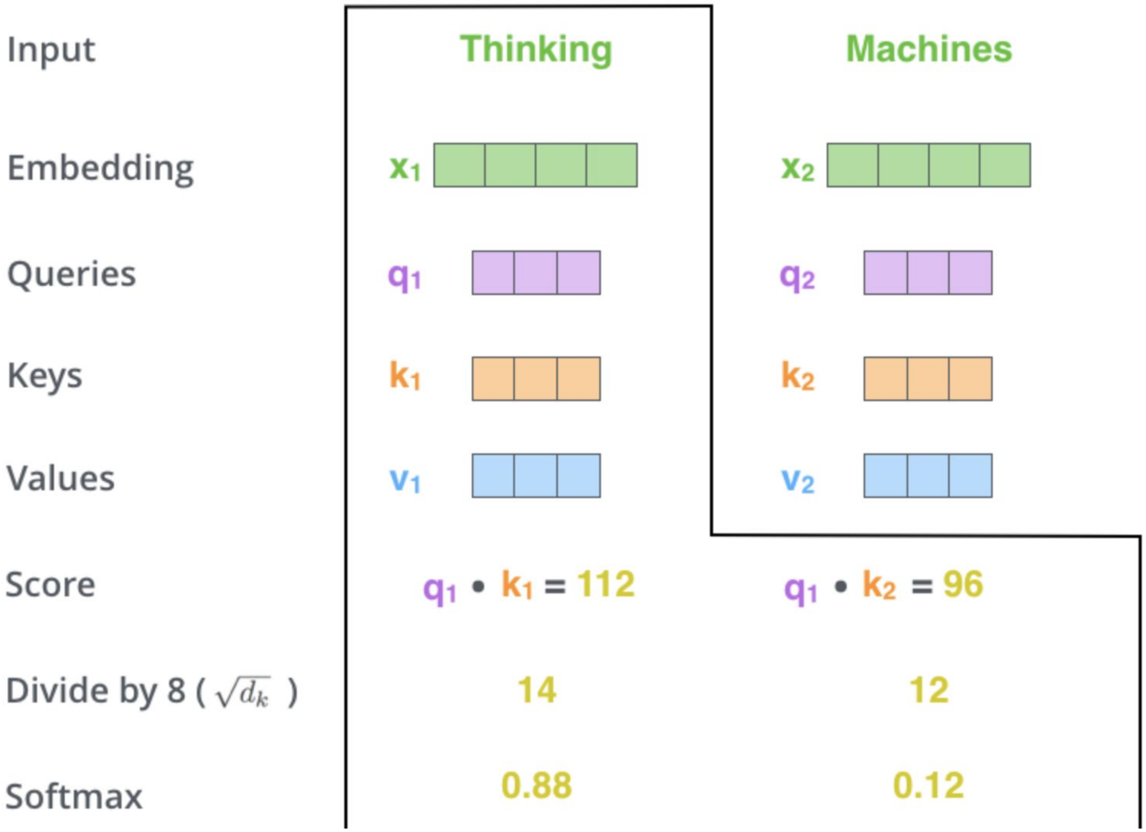


Image source: https://jalammar.github.io/illustrated-transformer/

Self-Attention: detailed explanation

STEP 5:

multiply each value vector by the softmax score

STEP 6:

sum up the weighted value vectors

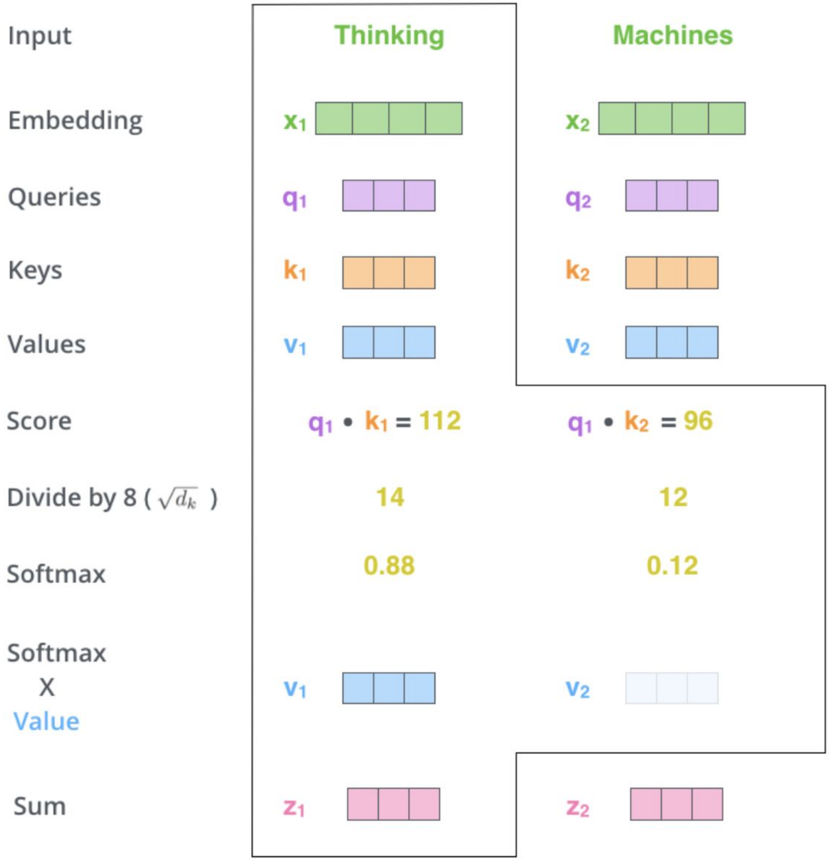
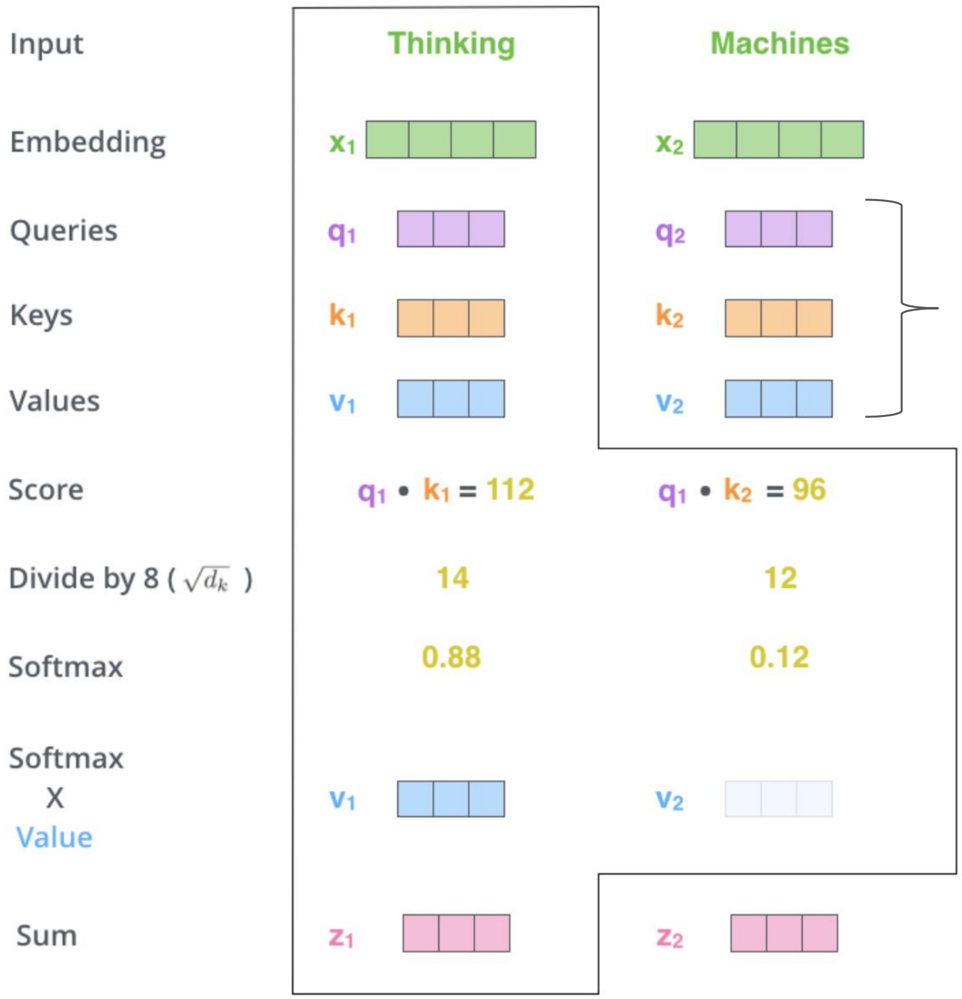


Image source: https://jalammar.github.io/illustrated-transformer/

Self-Attention



STEP 1: create Query, Key, Value

STEP 2: calculate scores

STEP 3: divide by $\sqrt{d_k}$

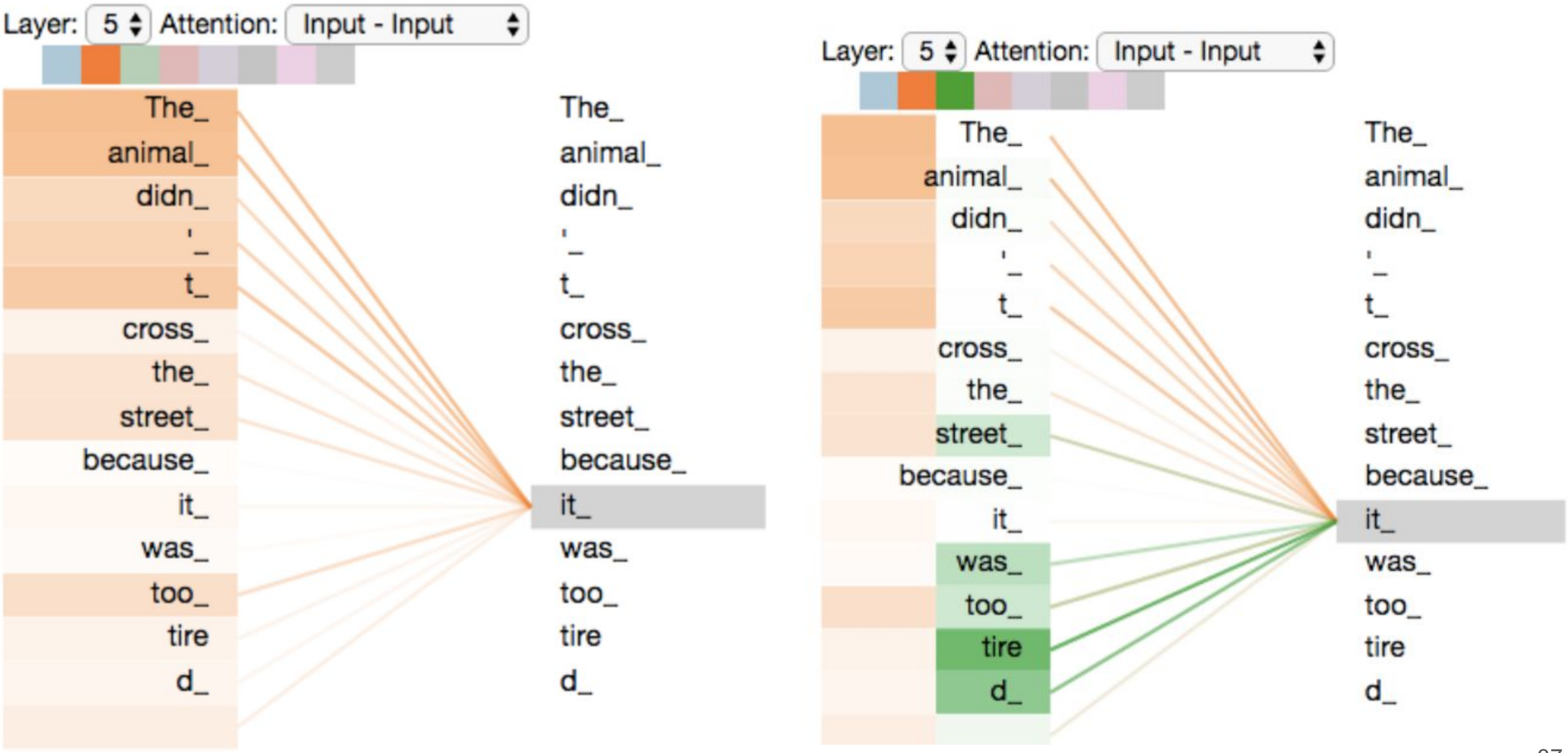
STEP 4: softmax

STEP 5: multiply each value vector by the softmax score

STEP 6: sum up the weighted value vectors

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

Multi-Head Attention

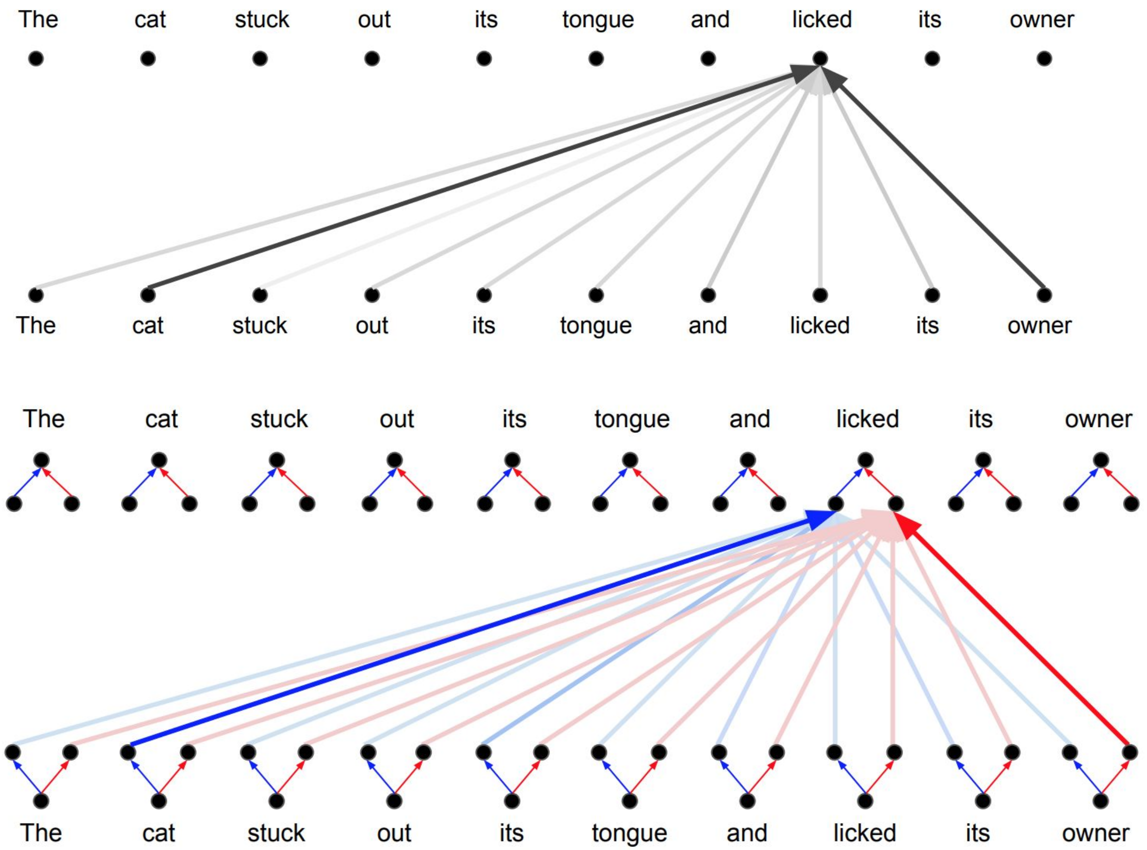


Attention vs. Multi-Head Attention

<u>Attention:</u> a weighted average

<u>Multi-Head Attention:</u>

parallel attention layers with different linear transformations on input and output.



Performance: WMT 2014 BLEU

	EN-DE	EN-FR
GNMT (orig)	24.6	39.9
ConvSeq2Seq	25.2	40.5
Transformer*	**28.4**	**41.8**

*Transformer models trained >3x faster than the others.

Research Challenges

- Constant ‘path length’ between any two positions.
- Unbounded memory.
- Trivial to parallelize (per layer).
- Models Self-Similarity.
- Relative attention provides expressive timing, equivariance, and extends naturally to graphs.

Positional Encoding

Positional Encoding: why sin and cos?

$$\vec{p_t}^{(i)} = f(t)^{(i)} \quad = \begin{cases} \sin(\omega_k t), & \text{if } i = 2k \\ \cos(\omega_k t), & \text{if } i = 2k+1 \end{cases}$$

$$\omega_k = \frac{1}{10000^{2k/d}}$$

$$\vec{p_t} = \begin{bmatrix} \sin(\omega_1.t) \\ \cos(\omega_1.t) \\ \\ \sin(\omega_2.t) \\ \cos(\omega_2.t) \\ \\ \vdots \\ \\ \sin(\omega_{d/2}.t) \\ \cos(\omega_{d/2}.t) \end{bmatrix}_{d \times 1}$$

t stays for position in the original sequence
k is the index of the element in the positional vector

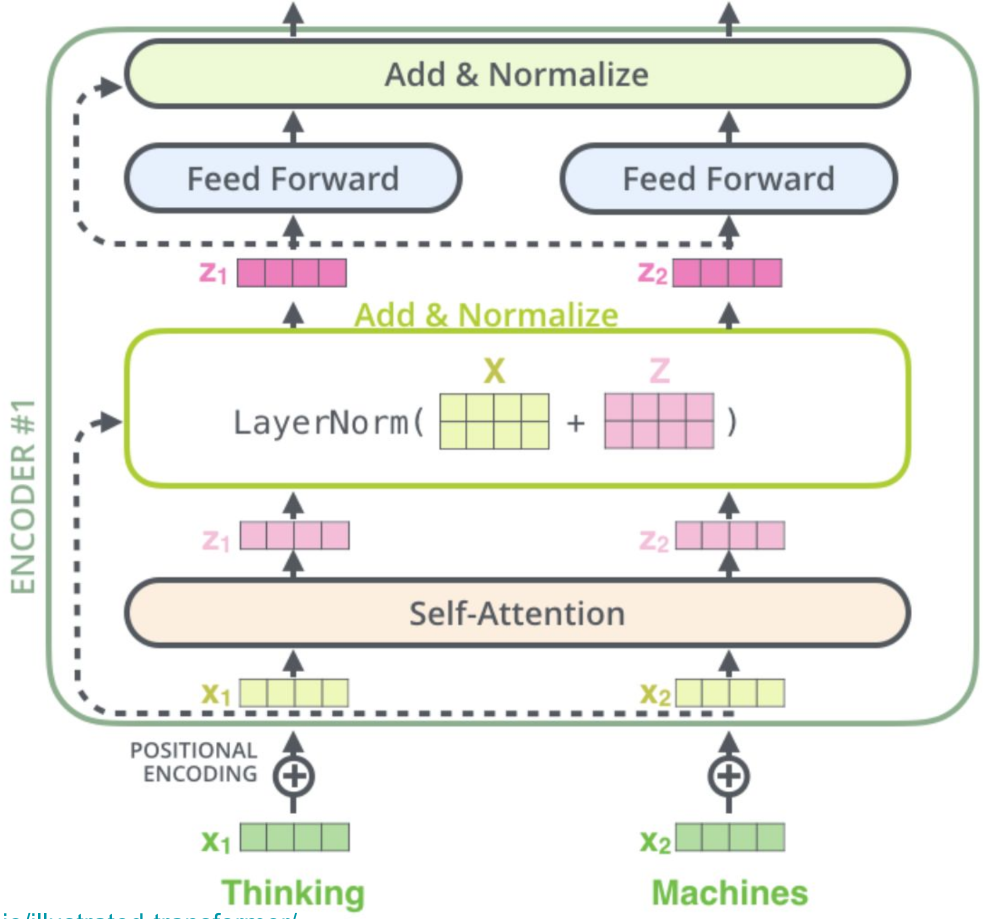
Image source: https://kazemnejad.com/blog/transformer_architecture_positional_encoding/

Layer Normalization

Layer Normalization

Like BatchNorm

but normalize along all features representing latent vector

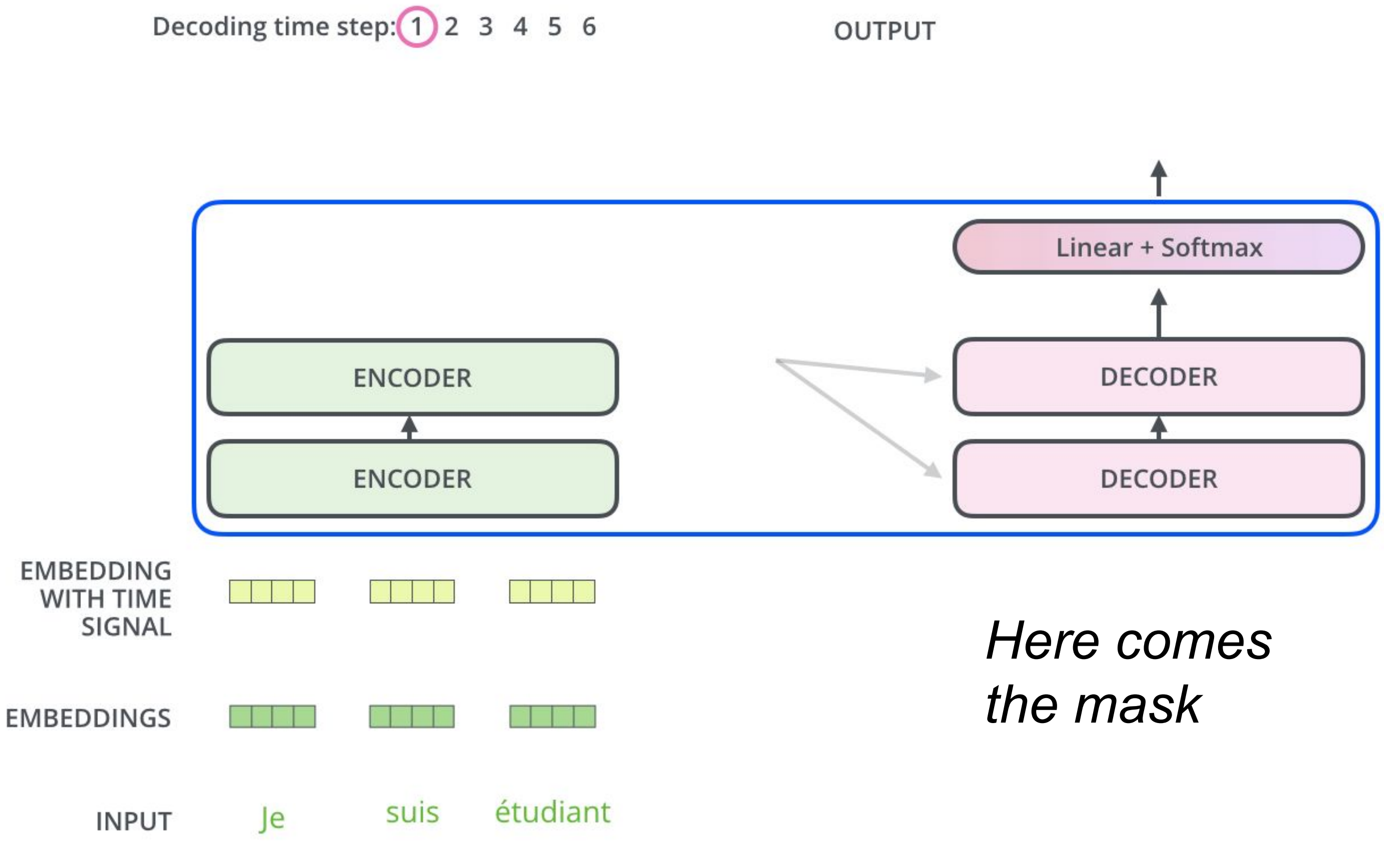


More info:
[Layer Normalization]()

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

The Decoder

The Decoder Side



Here comes the mask

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

The Decoder Side

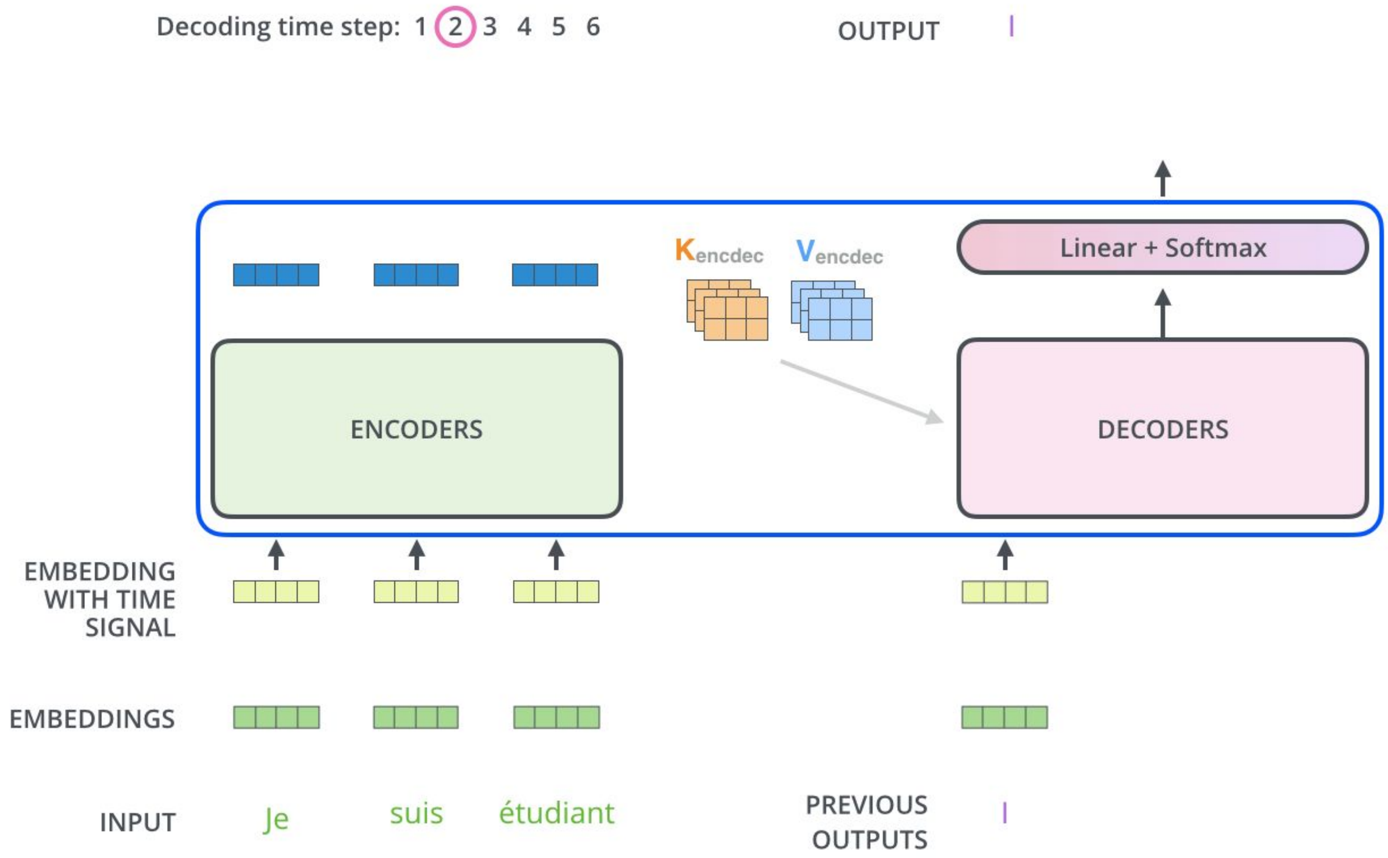


Image source: https://jalammar.github.io/illustrated-transformer/

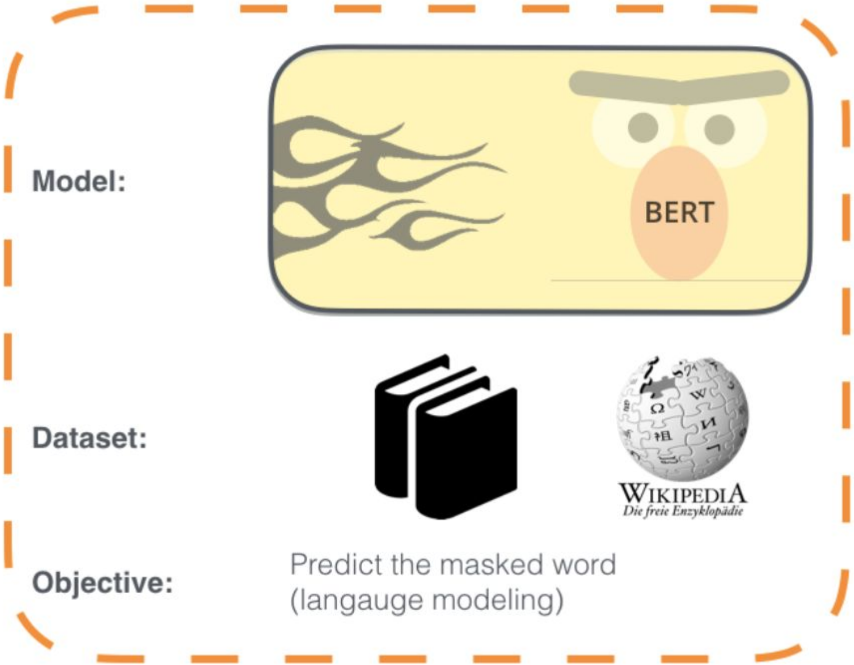
BERT

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

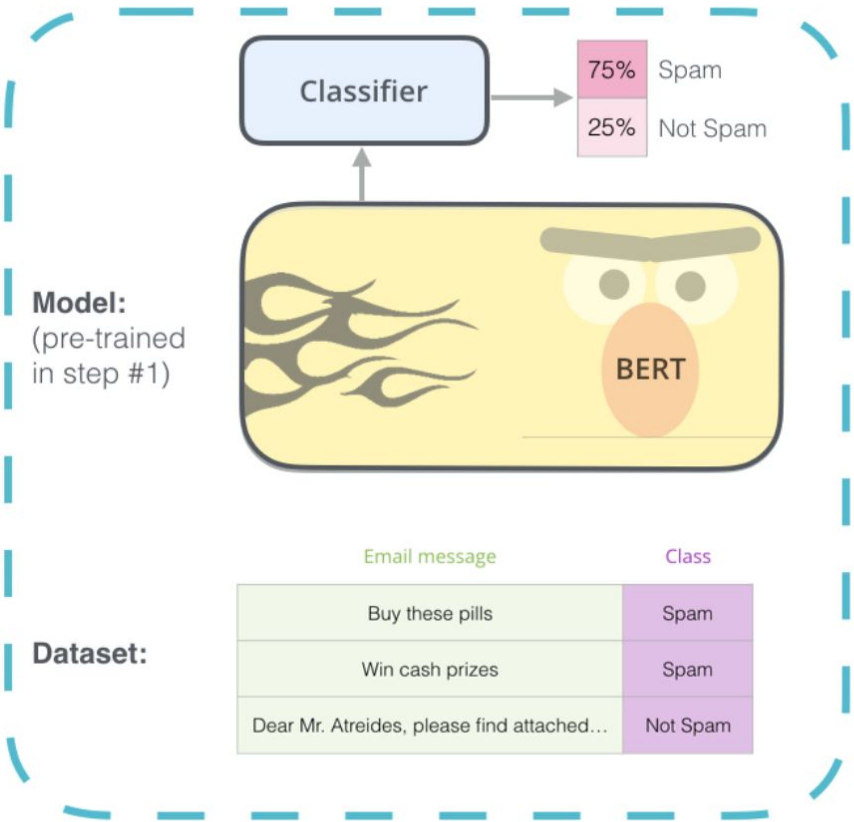


Image source: http://jalammar.github.io/illustrated-bert/

BERT

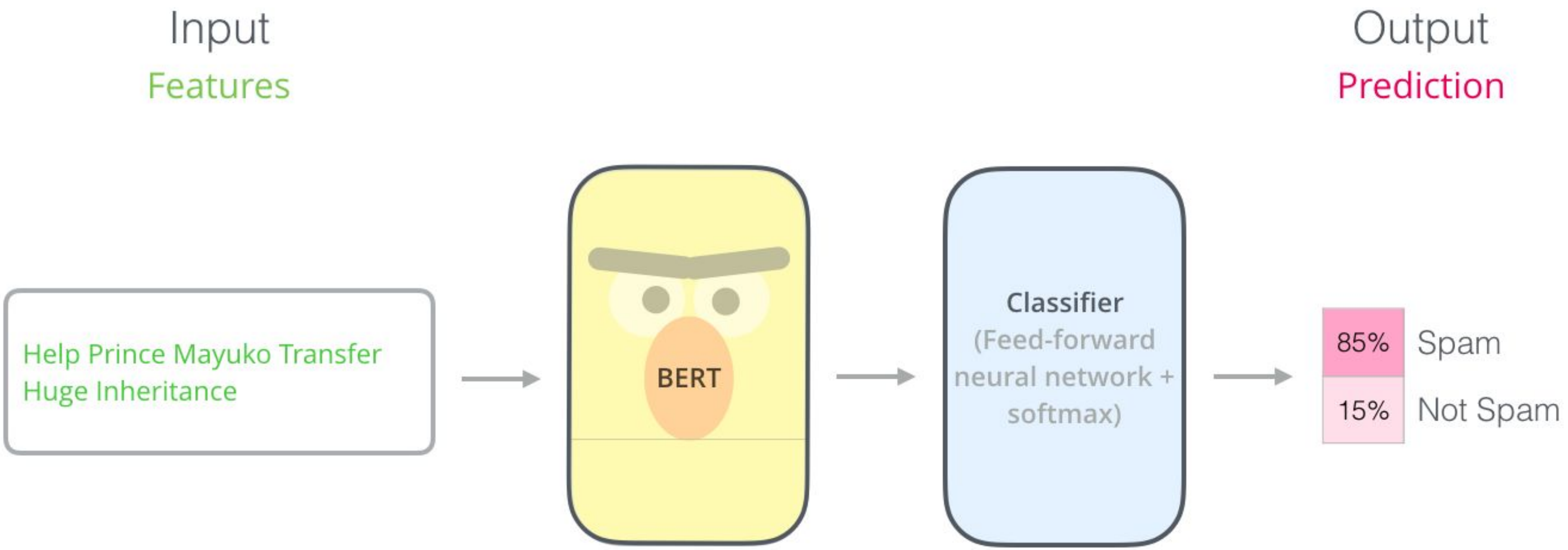


Image source: http://jalammar.github.io/illustrated-bert/

BERT: base and large

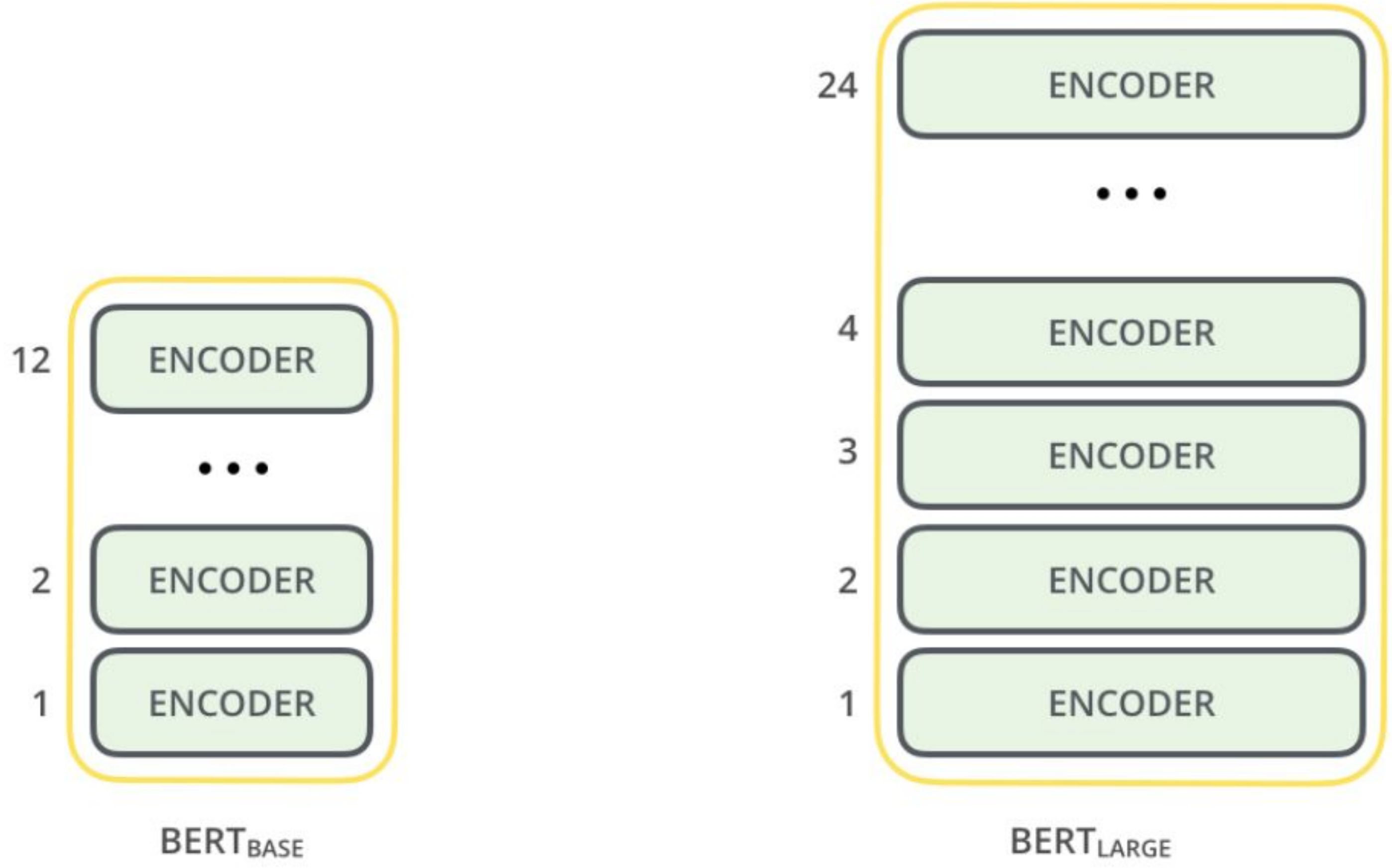


Image source: http://jalammar.github.io/illustrated-bert/

BERT vs. Transformer

			
		Base BERT	**Large BERT**
Encoders	6	12	24
Units in FFN	512	768	1024
Attention Heads	8	12	16

Image source: http://jalammar.github.io/illustrated-bert/

Model inputs

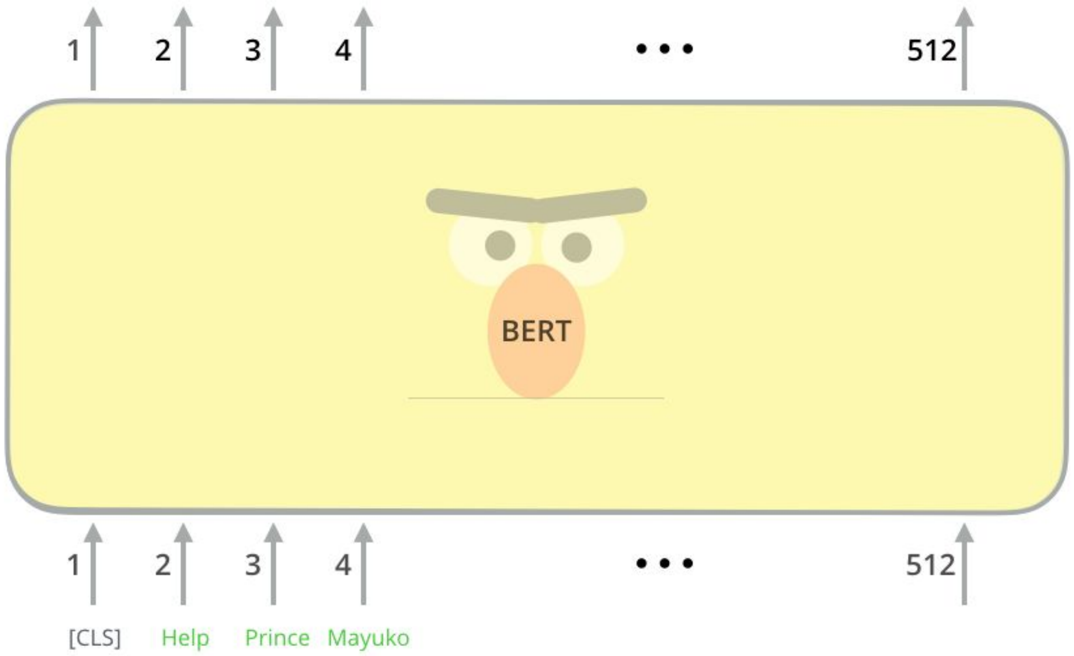
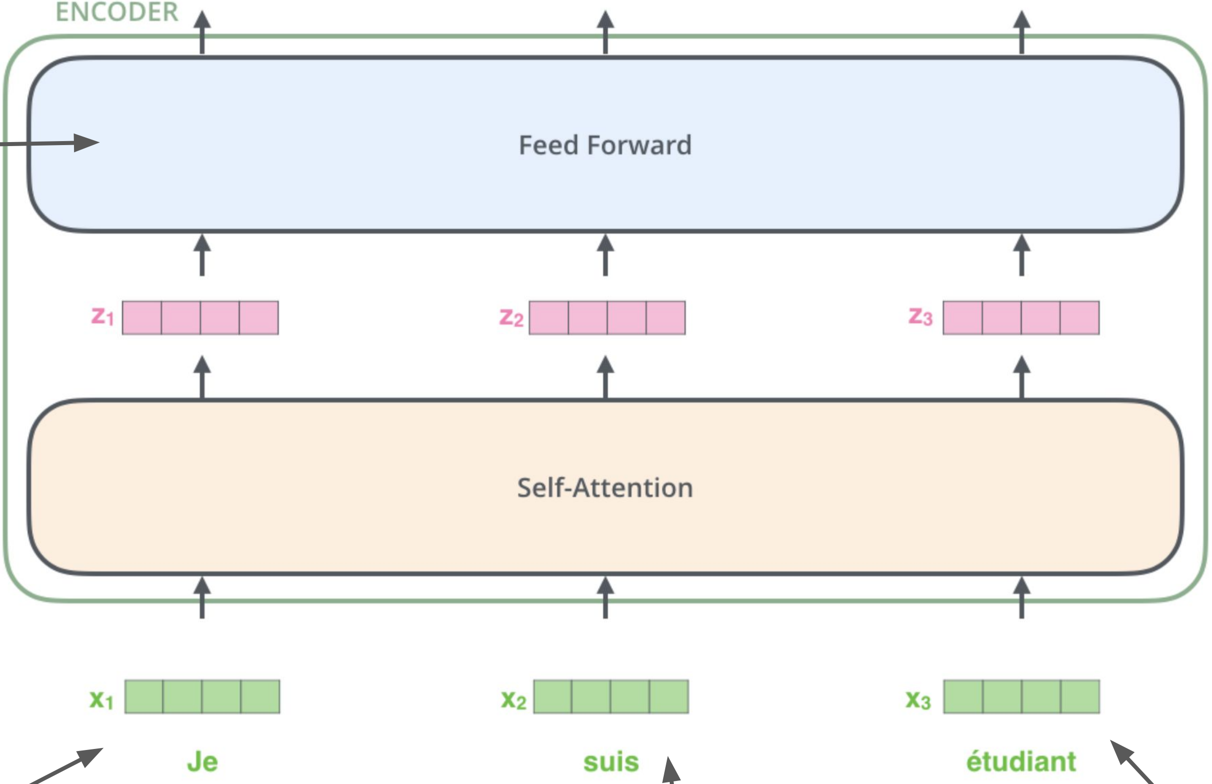


Image source: http://jalammar.github.io/illustrated-bert/

Transformer Block in BERT

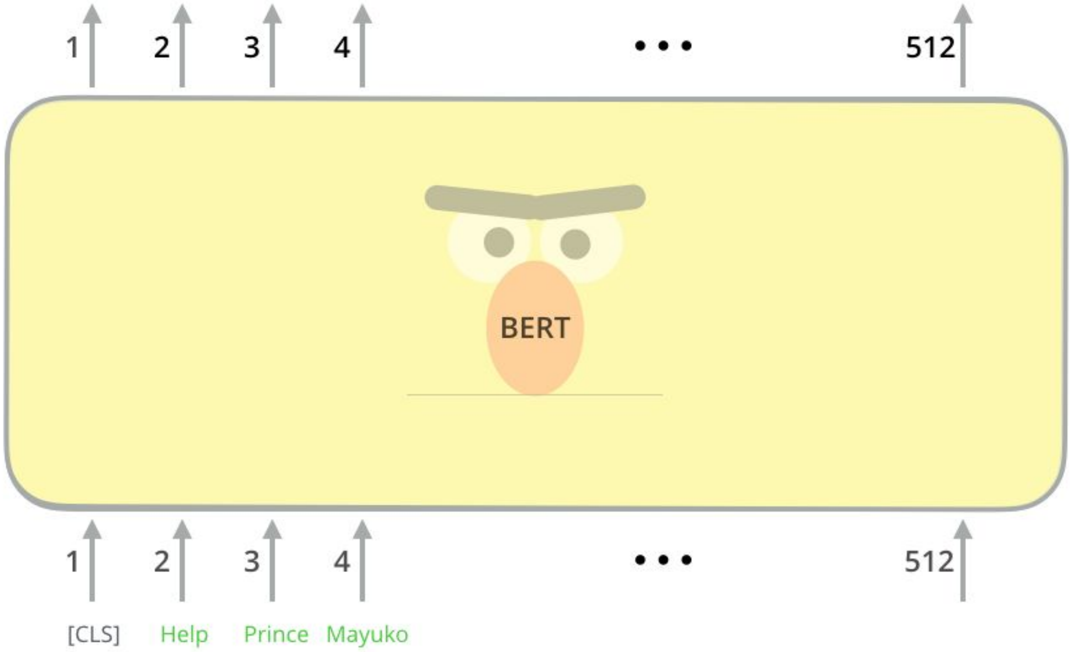
Can be parallelized



the word in each position flows through its own path in the encoder

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

Model inputs

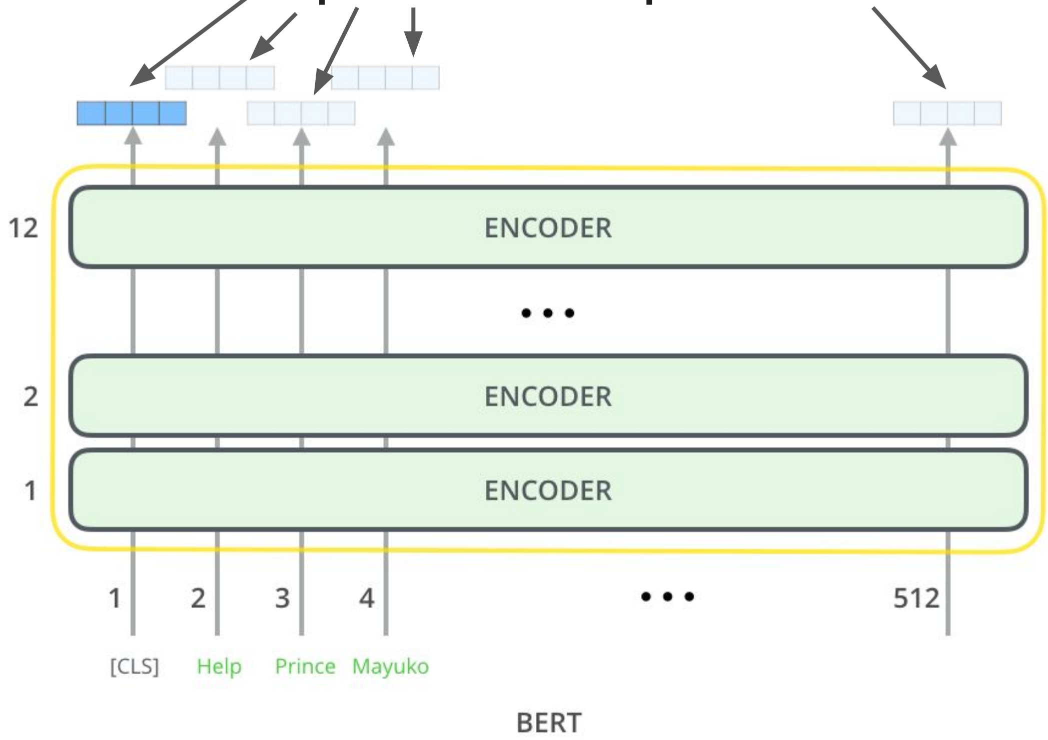


Identical to the Transformer up until this point

Why is BERT so special?

Model outputs

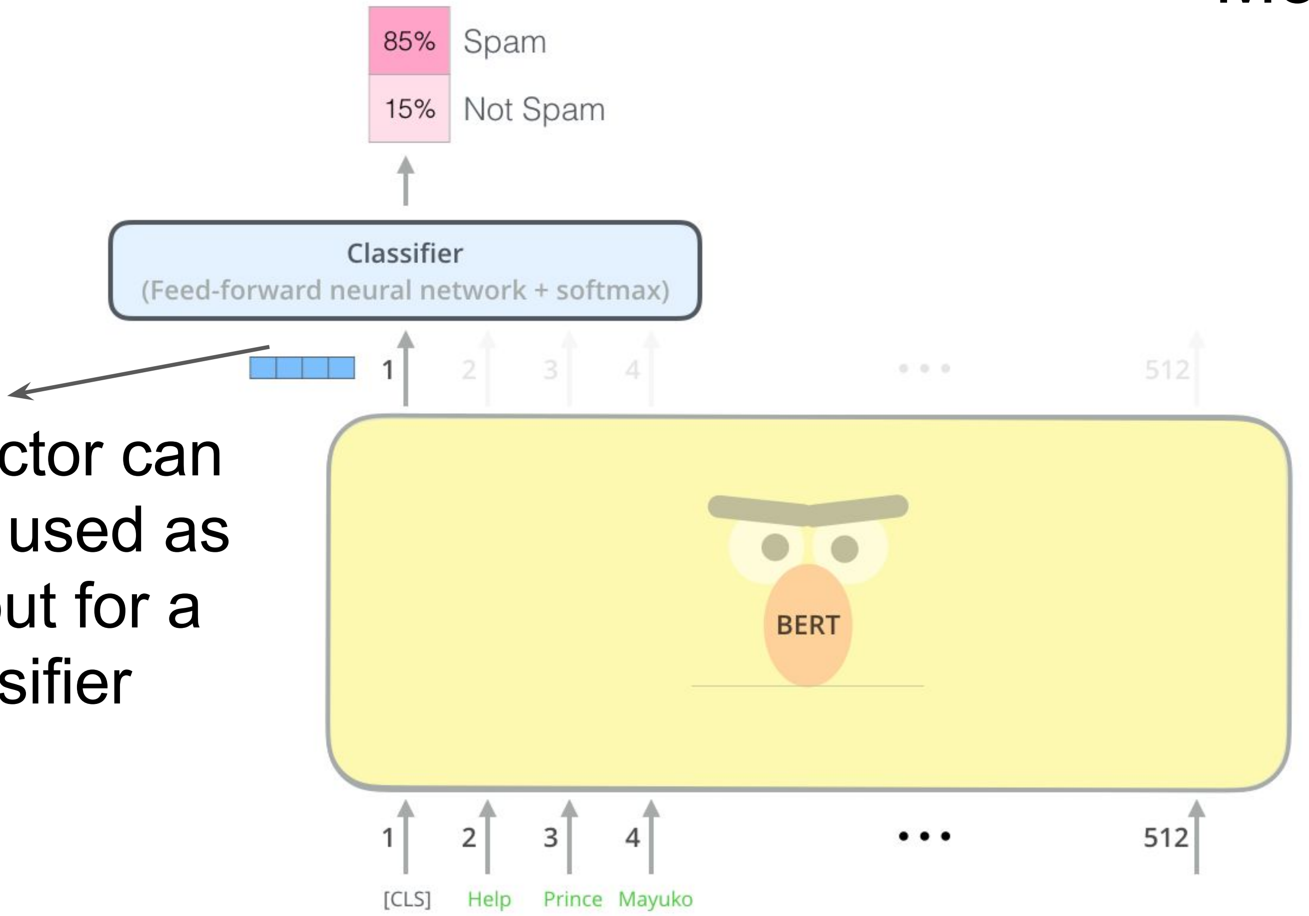
Each position outputs a vector



For sentence classification we focus on the first position (that we passed [CLS] token to)

Image source: http://jalammar.github.io/illustrated-bert/

Model inputs



This vector can now be used as the input for a classifier

Image source: http://jalammar.github.io/illustrated-bert/

BERT: pre-training

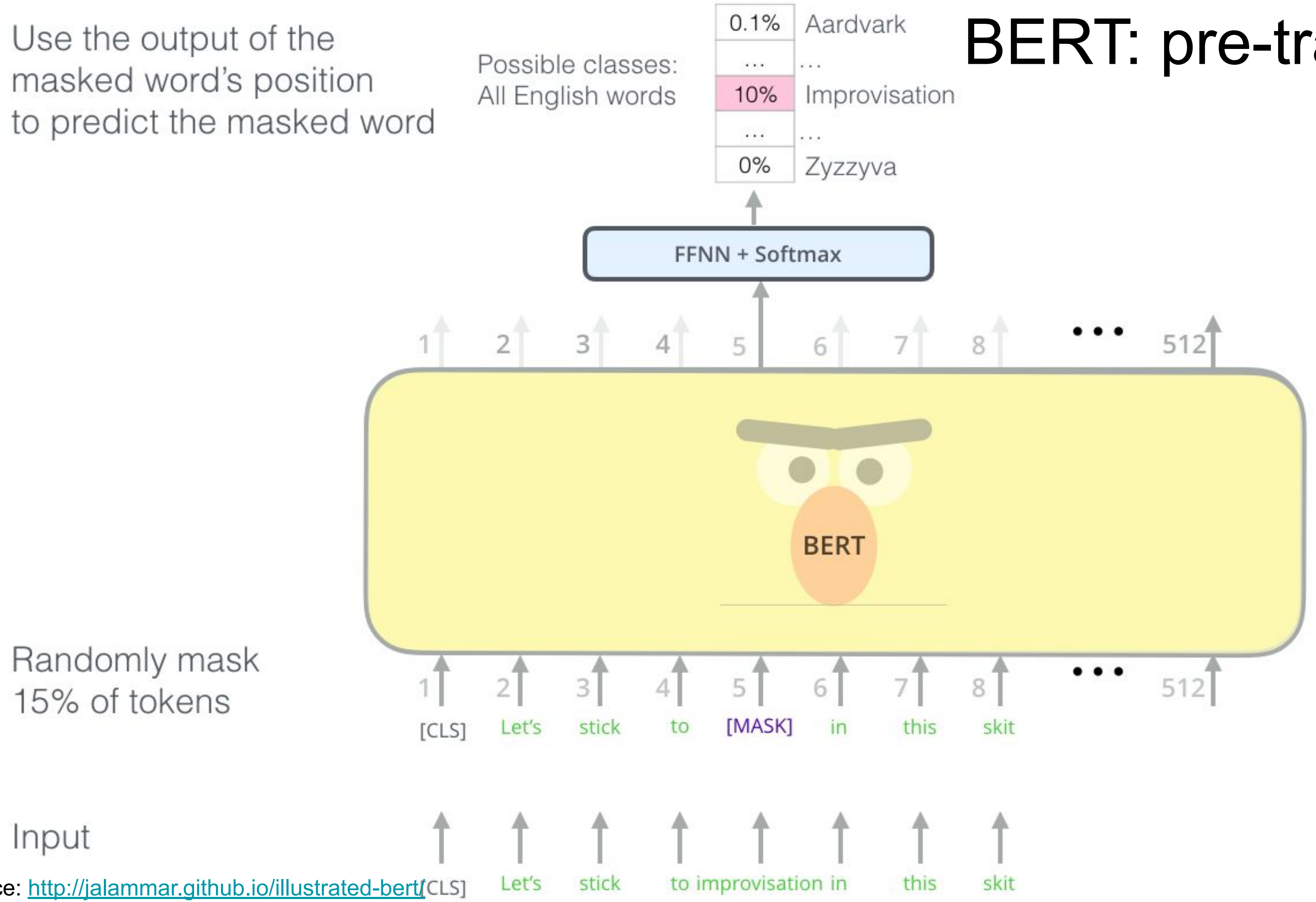


Image source: http://jalammar.github.io/illustrated-bert/

BERT: pre-training

- “Masked Language Model” approach

- To make BERT better at handling relationships between multiple sentences, the pre-training process includes an additional task:

  “Given two sentences (A and B), is B likely to be the sentence that follows A, or not?”

BERT: pre-training

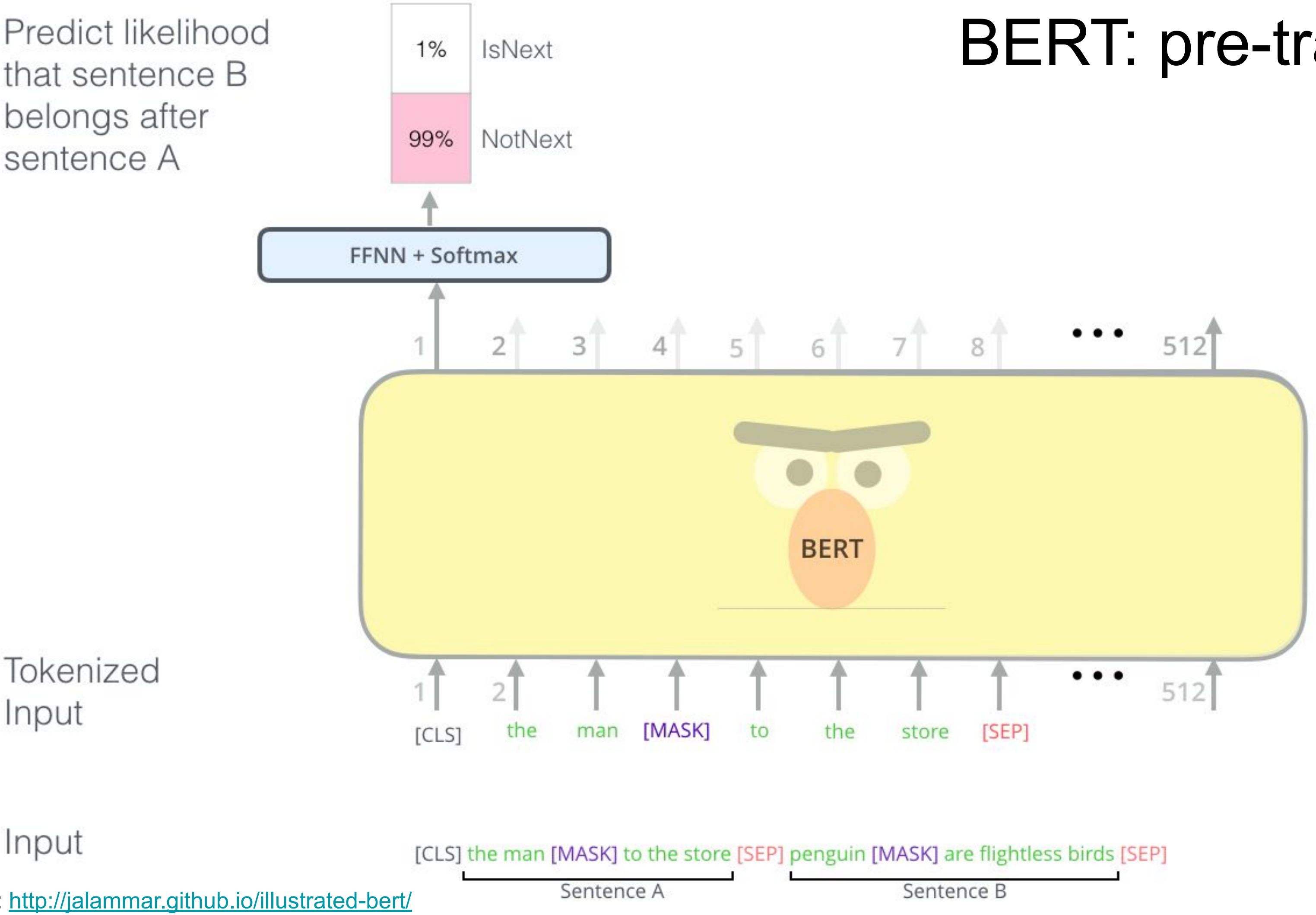


Image source: http://jalammar.github.io/illustrated-bert/

- [Transformer](#)
- [OpenAI Transformer](#)
- [ELMO](#)
- [BERT](#)
- [BERTology](#)
- [GPT](#)
- [GPT-2](#)
- [GPT-3](#)

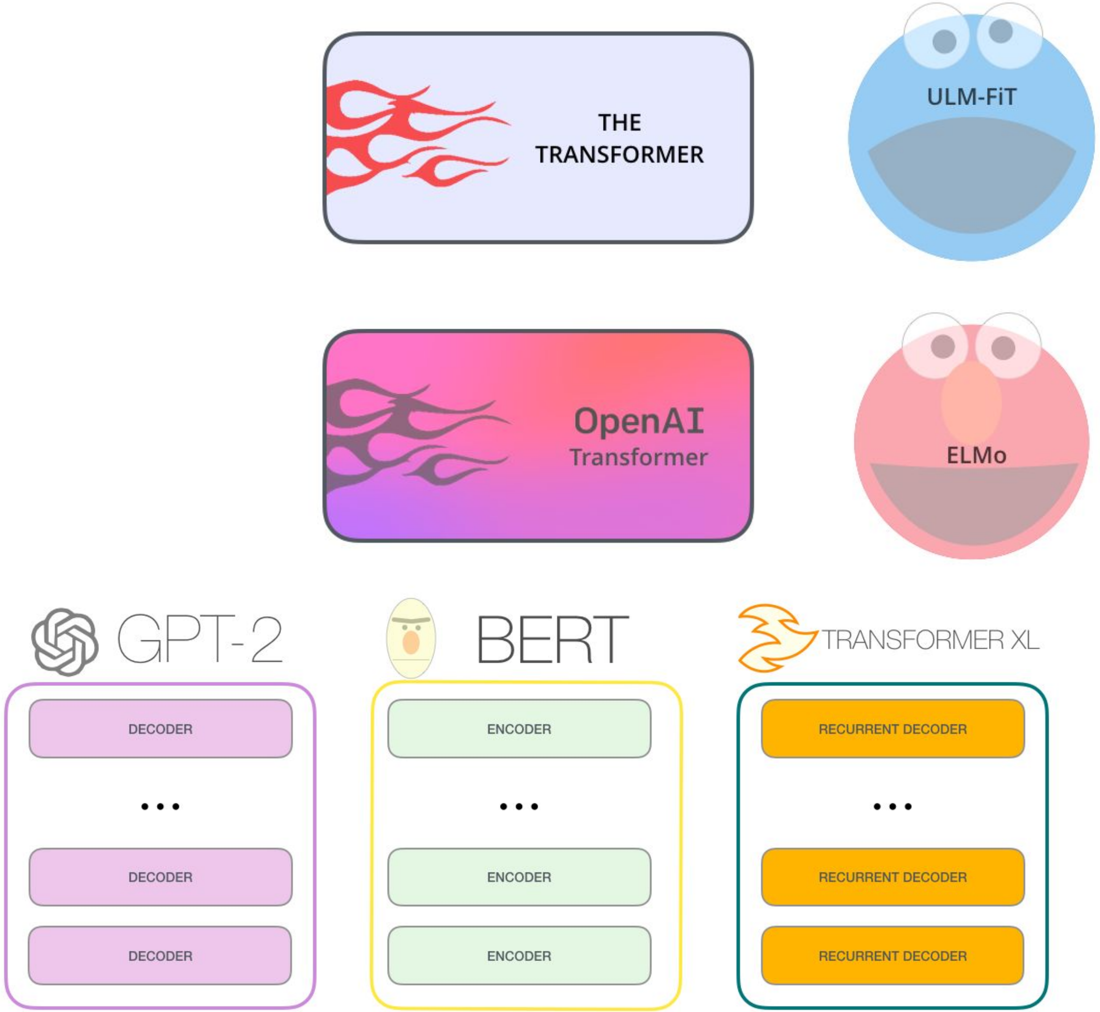


Image sources: http://jalammar.github.io/illustrated-bert/
http://jalammar.github.io/illustrated-gpt2/

Outro

- Transformer is novel and very powerful architecture
- It is worth it to understand how Self-Attention works
- BERT is variant of Decoders from Transformer for variety of tasks
- GPT are even bigger and better in metrics but they are made by corporations