Advanced Prediction Models

Today’s Outline

- Recap of Attention in Sequence to Sequence Models
- Transformer Architecture and Self-Attention
- Transfer Learning using a pre-trained NLP model
- BERT and related architectures

Attention in Seq2Seq

What does attention mean?

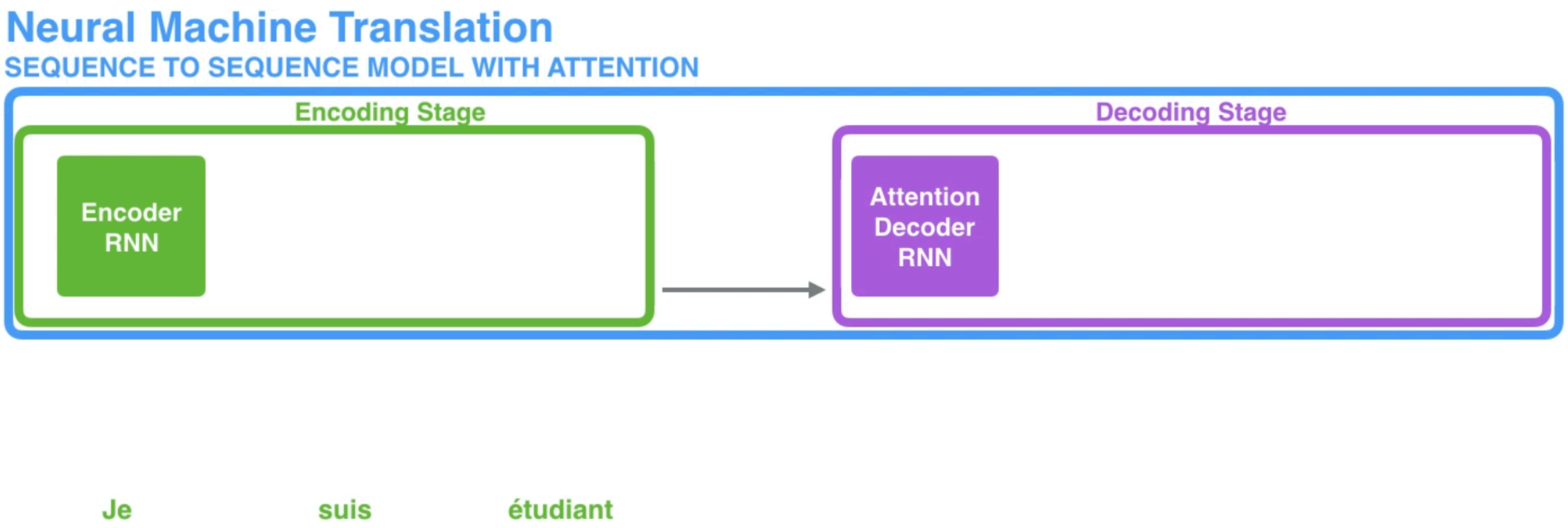
- Attend to certain steps/parts of the input sequence while deciding/predicting the current output (of the output sequence)

- By 'attend', we just mean that the prediction of the current output depends on **specific** parts of the input sequence.

- Next, we will revisit the attention mechanism in a neural machine translation sequence to sequence modelling setting.

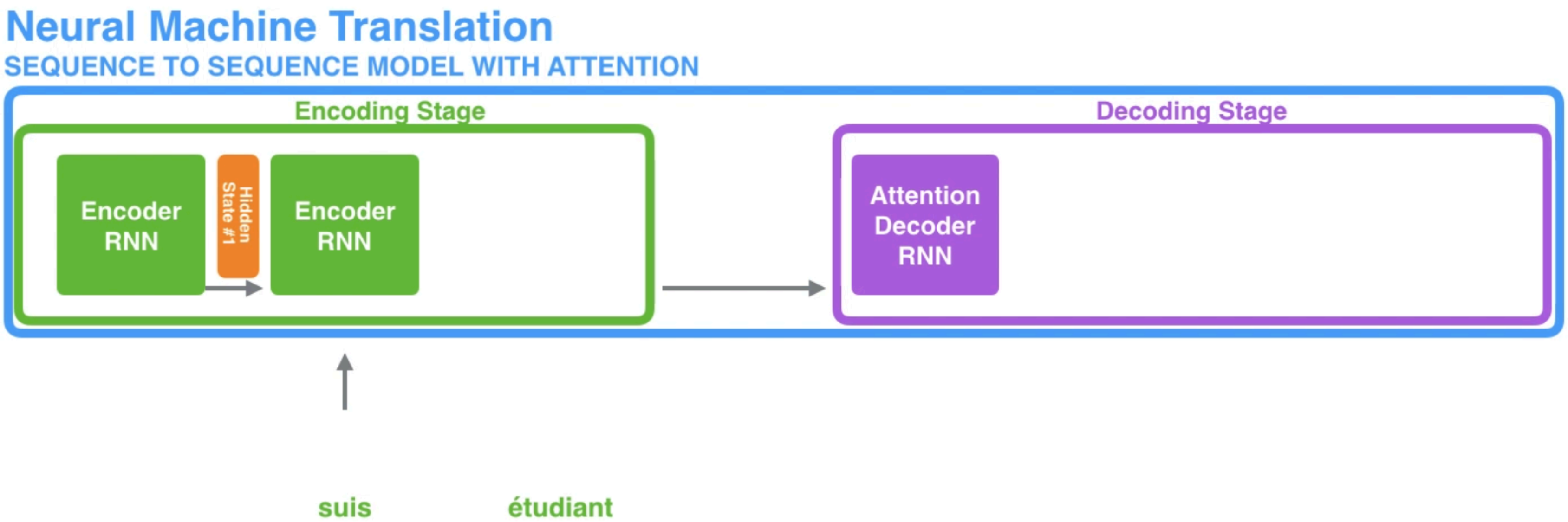
[1]Reference: N

Sequence to Sequence Modeling



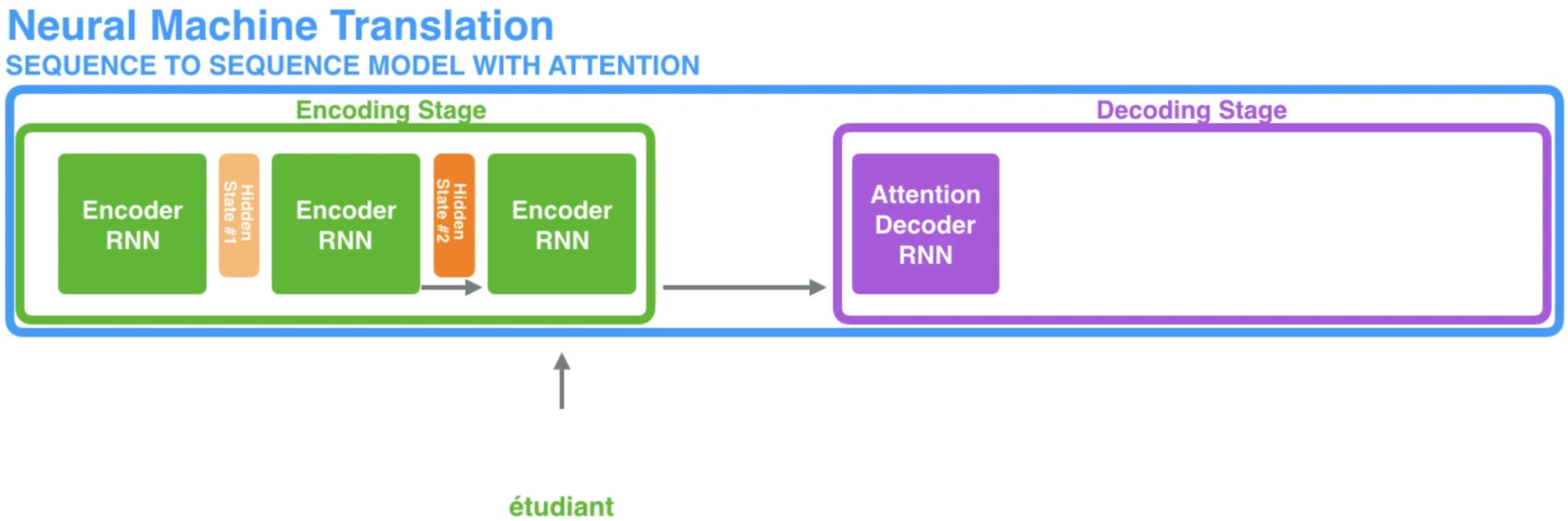
[1]Reference: https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/

Sequence to Sequence Modeling



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

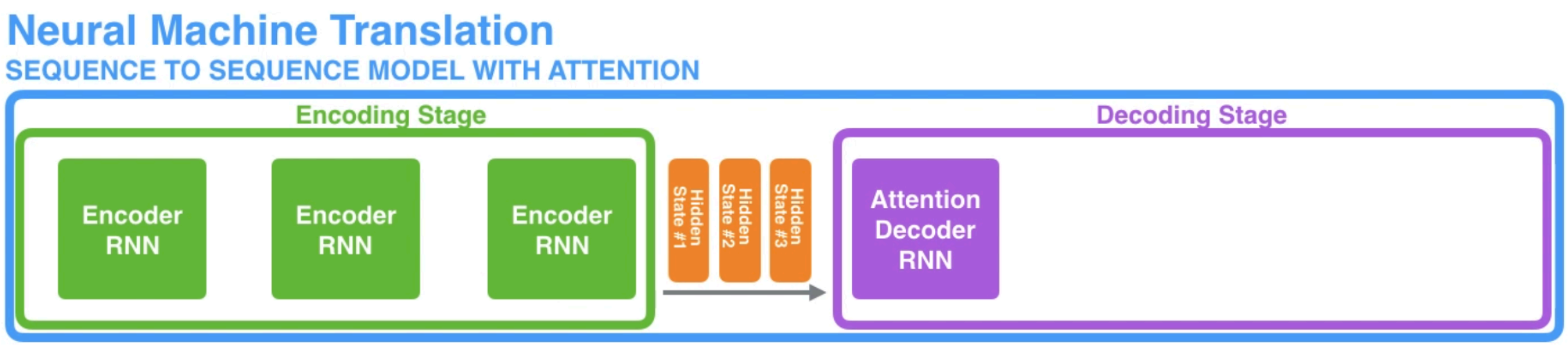
Sequence to Sequence Modeling



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

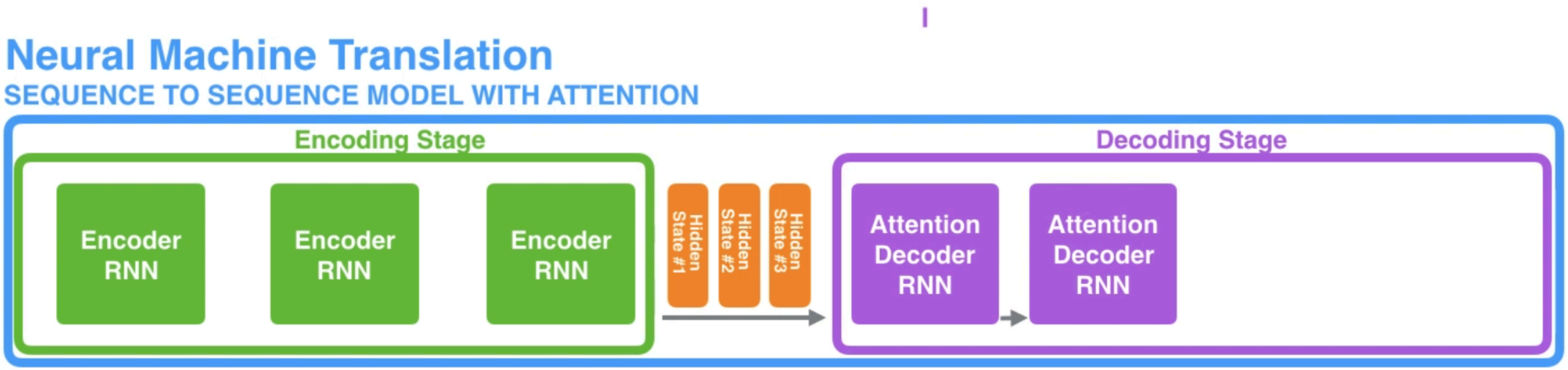
Sequence to Sequence Modeling

Collect the hidden layer outputs from all RNN steps



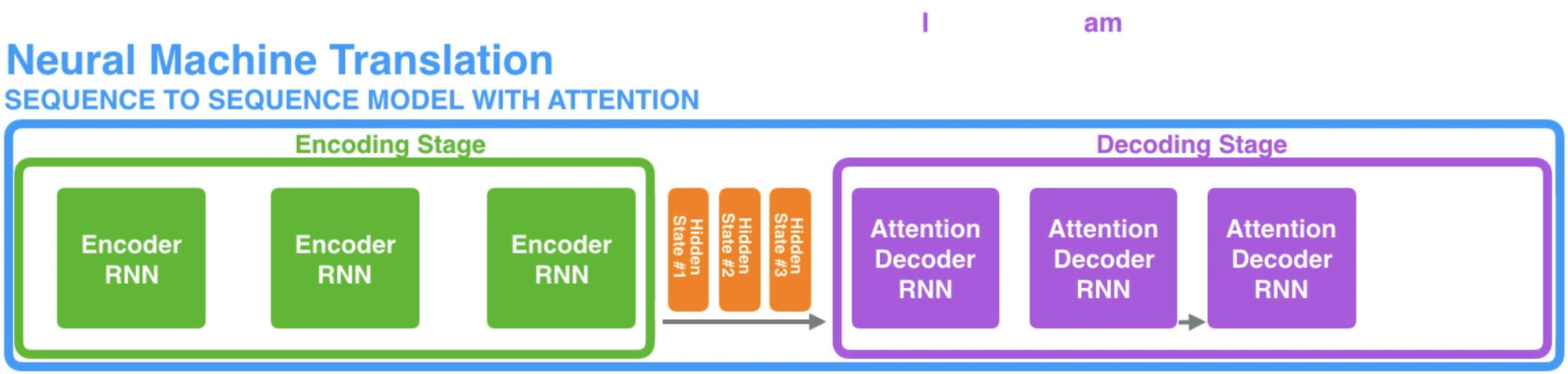
[1]Reference: https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/

Sequence to Sequence Modeling



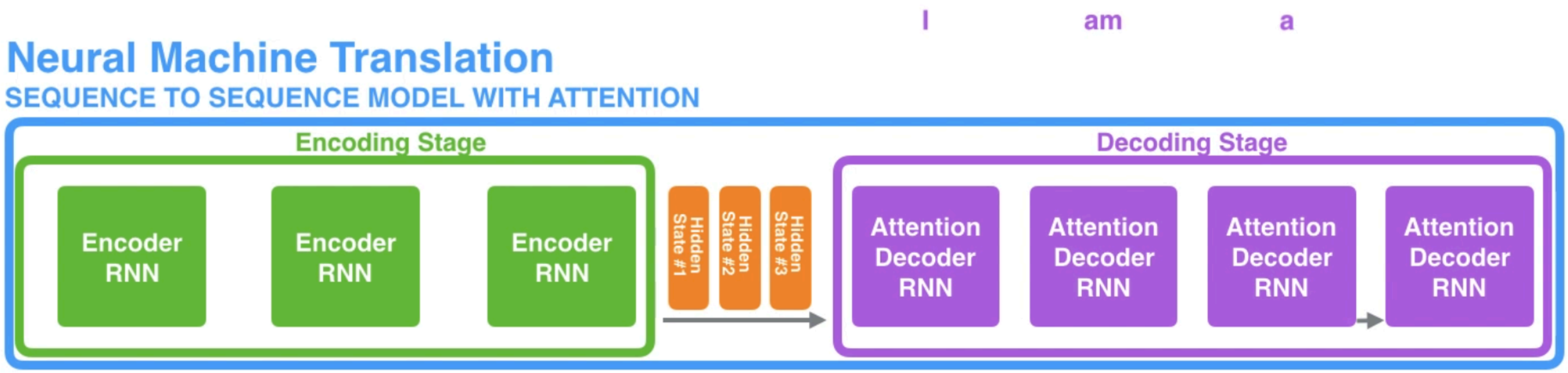
[1]Reference: https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/

Sequence to Sequence Modeling



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

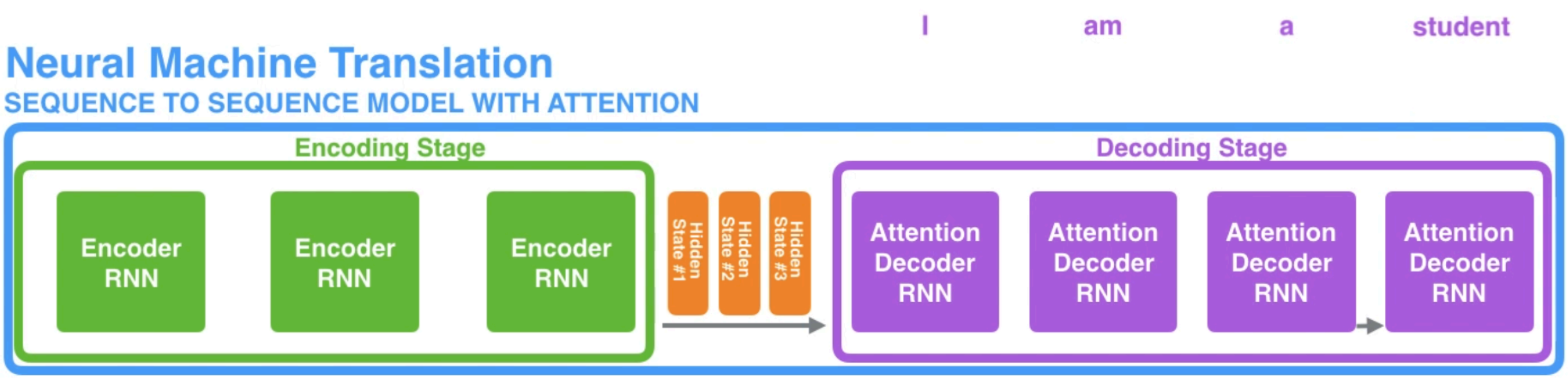
Sequence to Sequence Modeling



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

Sequence to Sequence Modeling

While decoding happens sequentially, each RNN step involves attention!



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

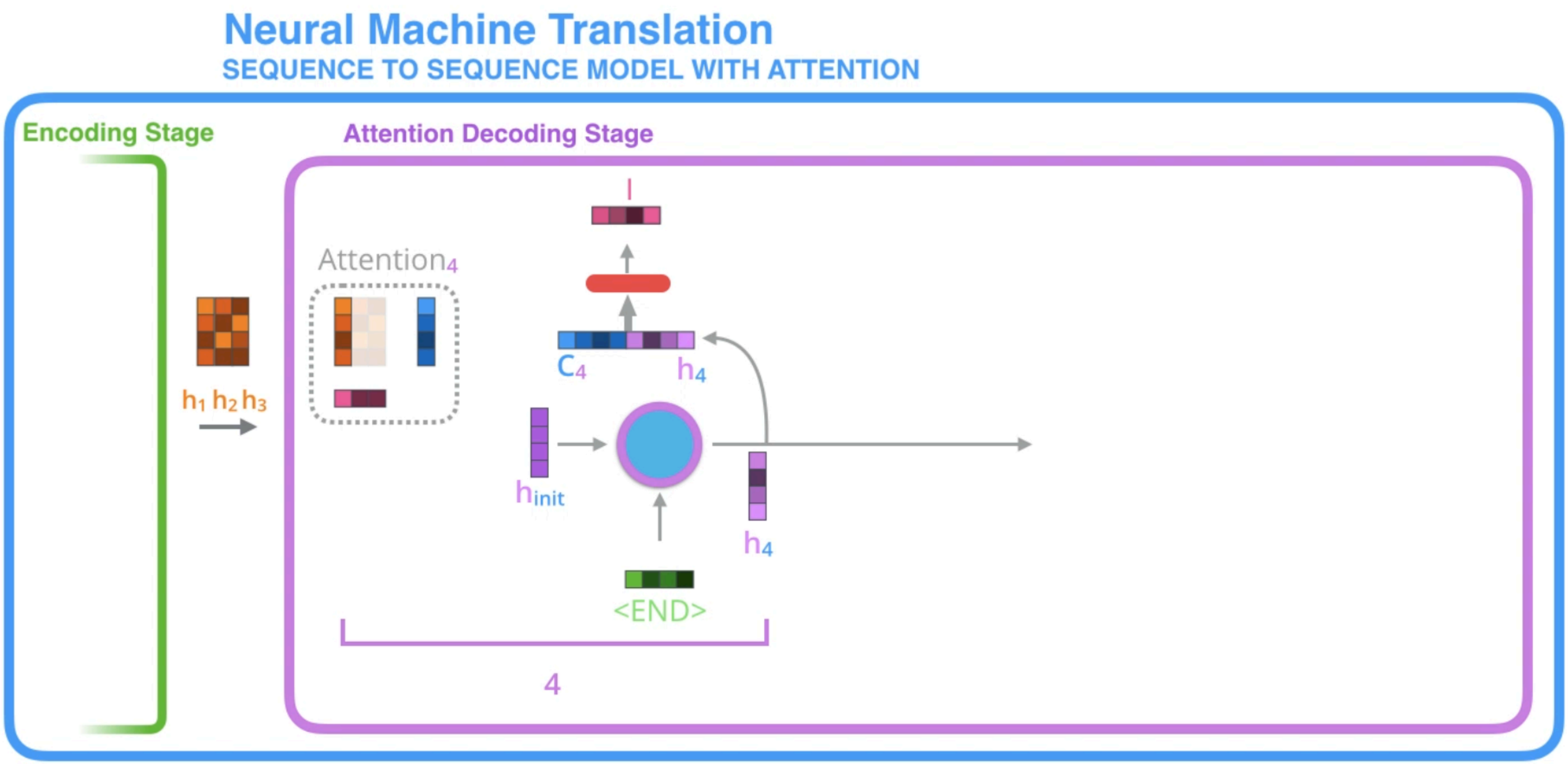
Attention in Seq2Seq

A combination of encoder vectors is concatenated with the decoder hidden vector

Recall that score generating function has its own parameters and uses the decoder hidden vector

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

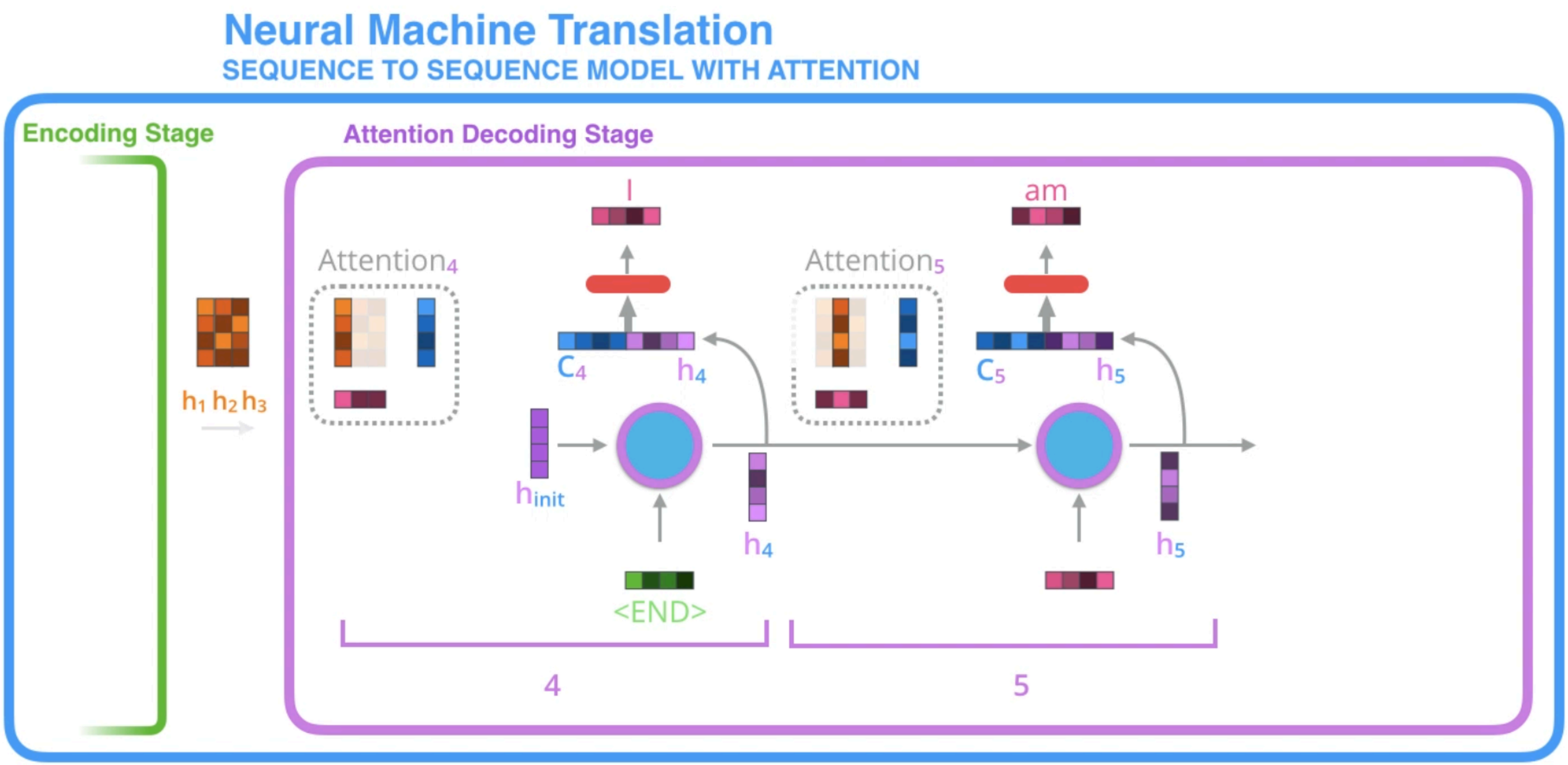
Attention in Seq2Seq



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

Attention in Seq2Seq

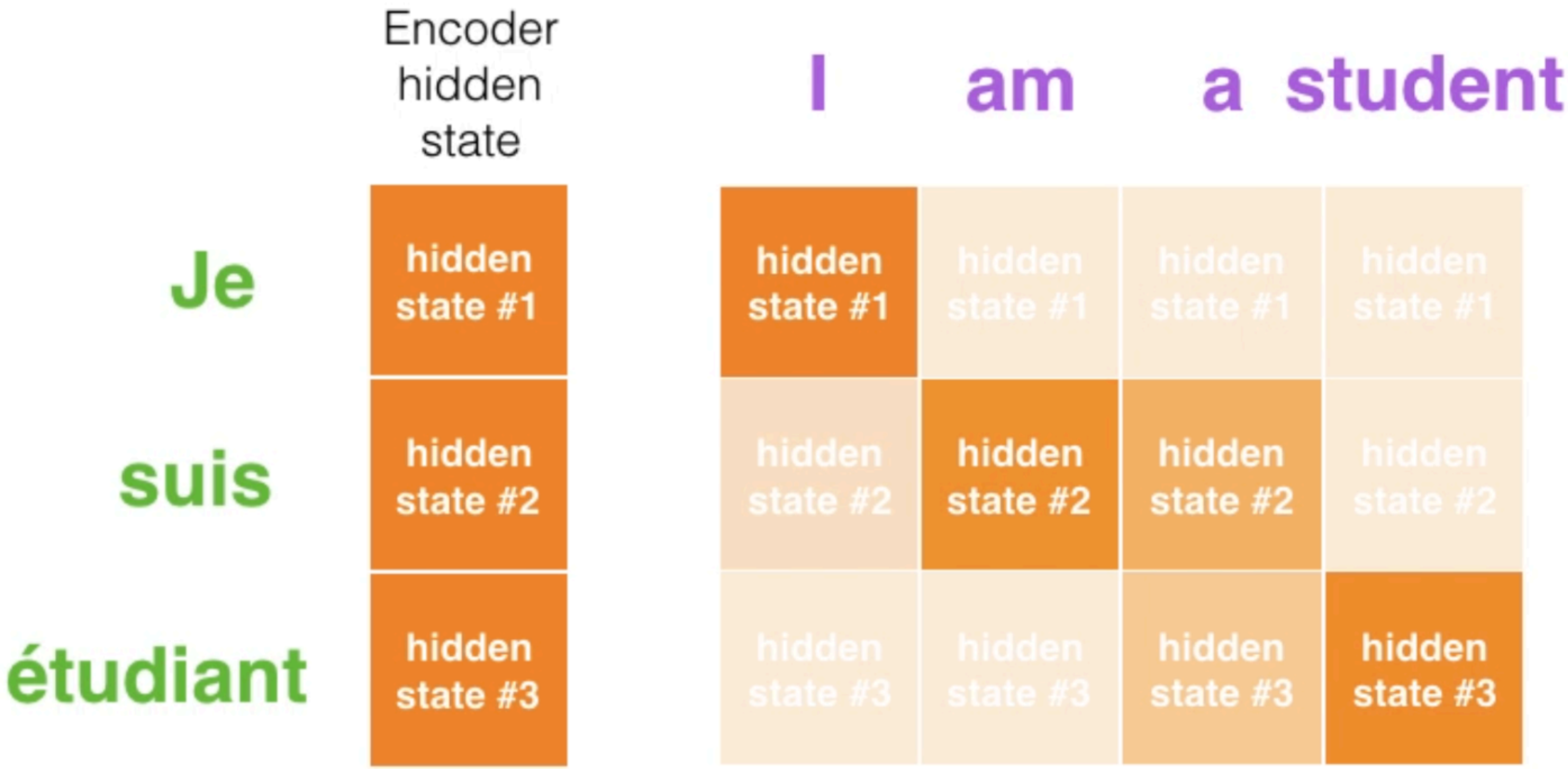
Because the attention mechanism spans across encoder and decoder, we will refer to it as encoder-decoder attention



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

Attention in Seq2Seq

An example illustrating the attention scores: in this example, the attention scores seem quite natural

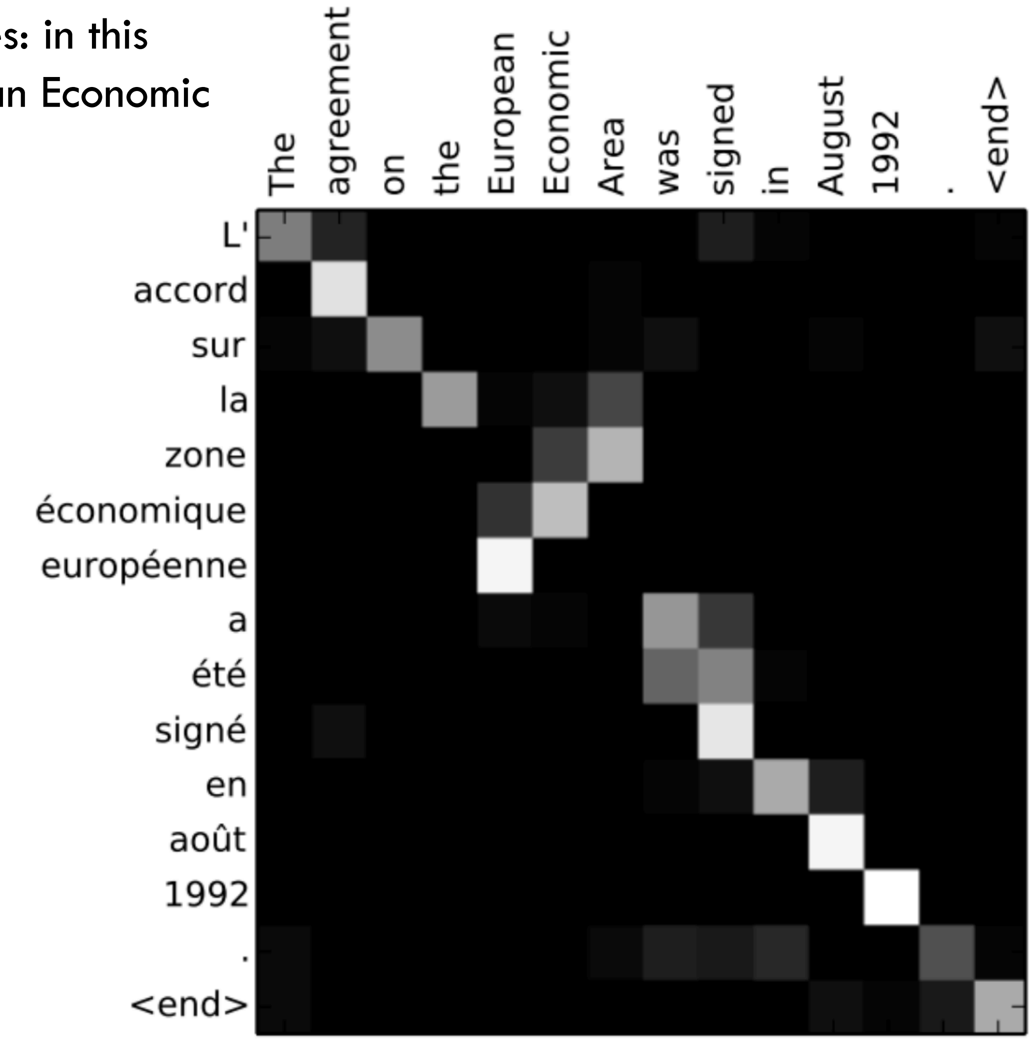


[1]Reference: https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/ . Also see https://github.com/tensorflow/nmt

Attention in Seq2Seq

An example illustrating the attention scores: in this example, the attention scores for 'European Economic Area' capture the non-triviality

Attention visualization – example of the alignments between source and target sentences



[1]Reference: Bahdanau et al., 2015

Questions?

Today’s Outline

- Recap of Attention in Sequence to Sequence Models
- Transformer Architecture and Self-Attention
- Transfer Learning using a pre-trained NLP model
- BERT and related architectures

Transformer Architecture and Self-Attention

Transformer

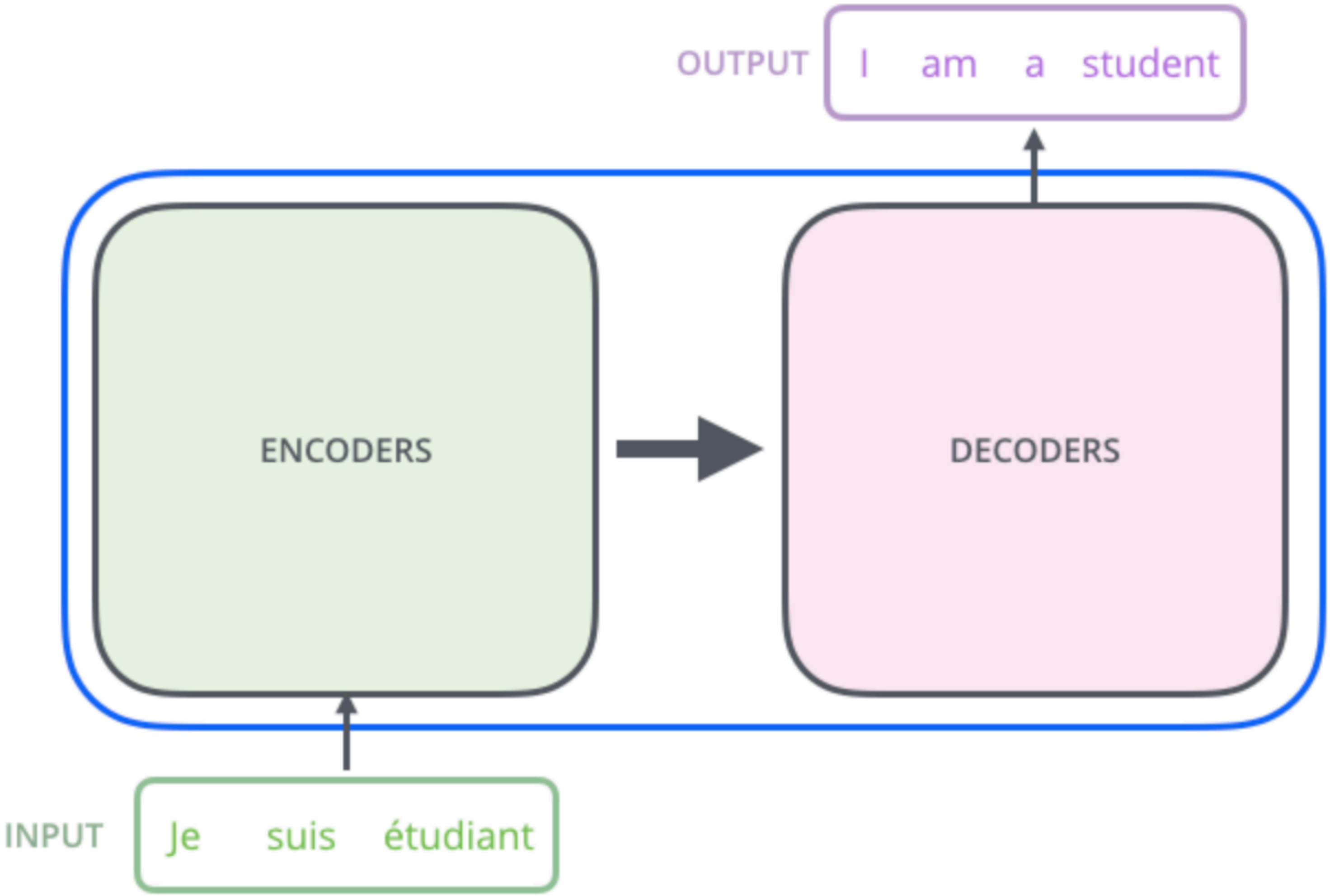
Is a new model/architecture (from 2017) that we will look at that can also be used for sequence to sequence modelling

Attention (same idea as before, but a slightly different take) is a core component of this model

Lets start again with neural machine translation as the running application.

[1]Reference: https://jalammar.github.io/illustrated-transformer/

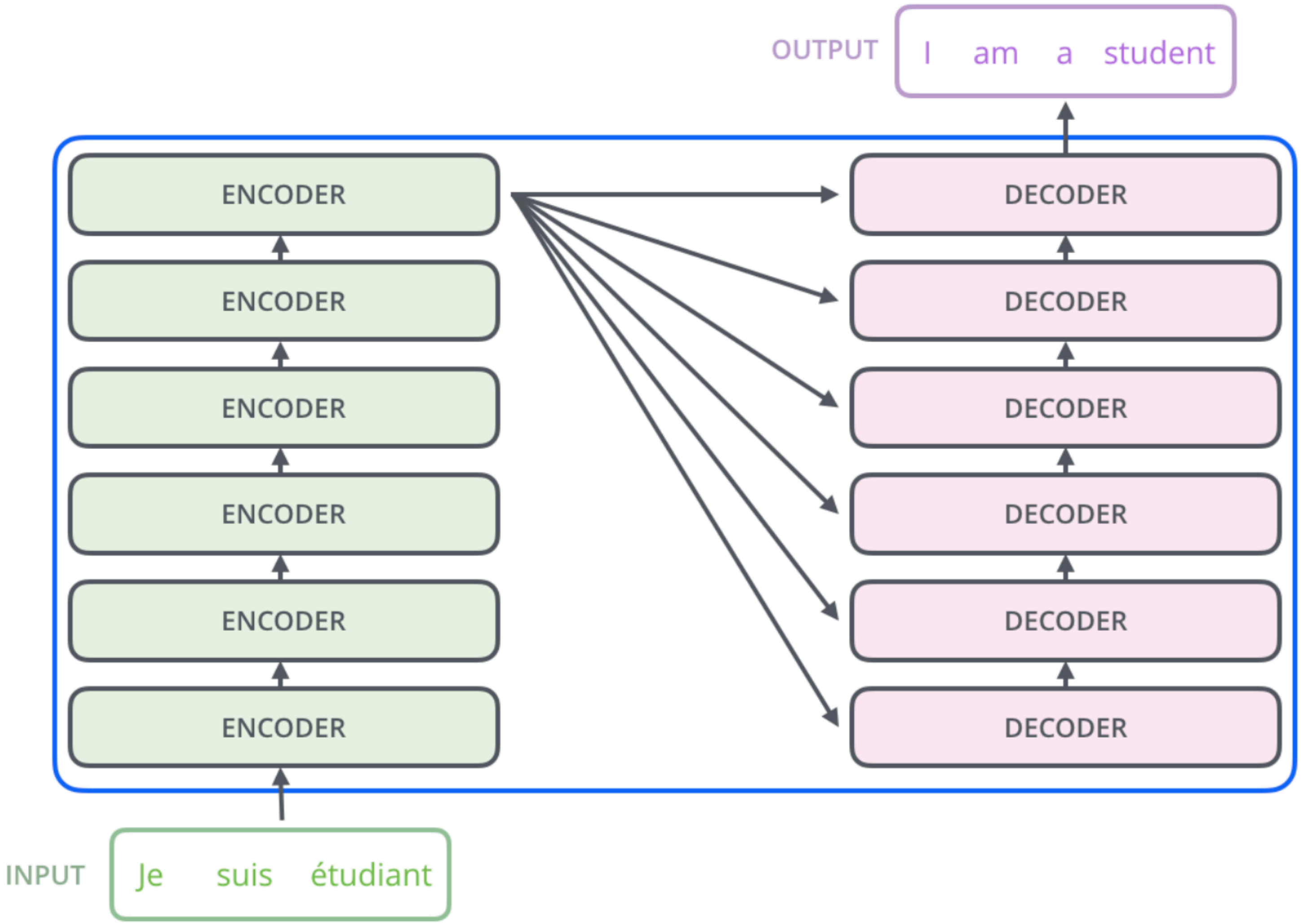
Transformer



[1]Reference: https://jalammar.github.io/illustrated-transformer/

Transformer

Lets now see the insides of both the encoder and the decoder in order.

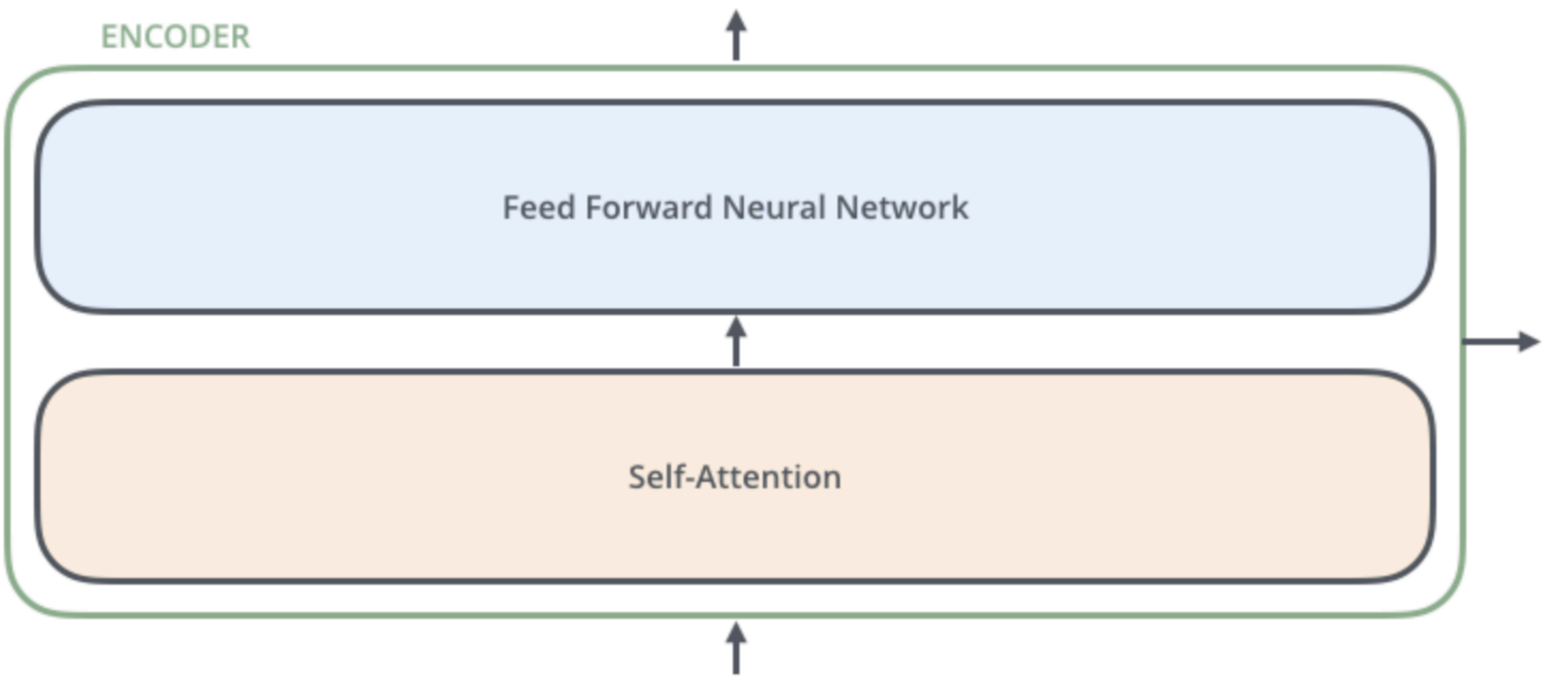


[1]Reference: https://jalammar.github.io/illustrated-transformer/

Transformer : Encoder

In particular, lets focus on one encoder unit.

The number of such units is fixed and does not depend on the input sequence length (thanks to parallel treatment of all input elements/words, as we will see later).



[1]Reference: https://jalammar.github.io/illustrated-transformer/

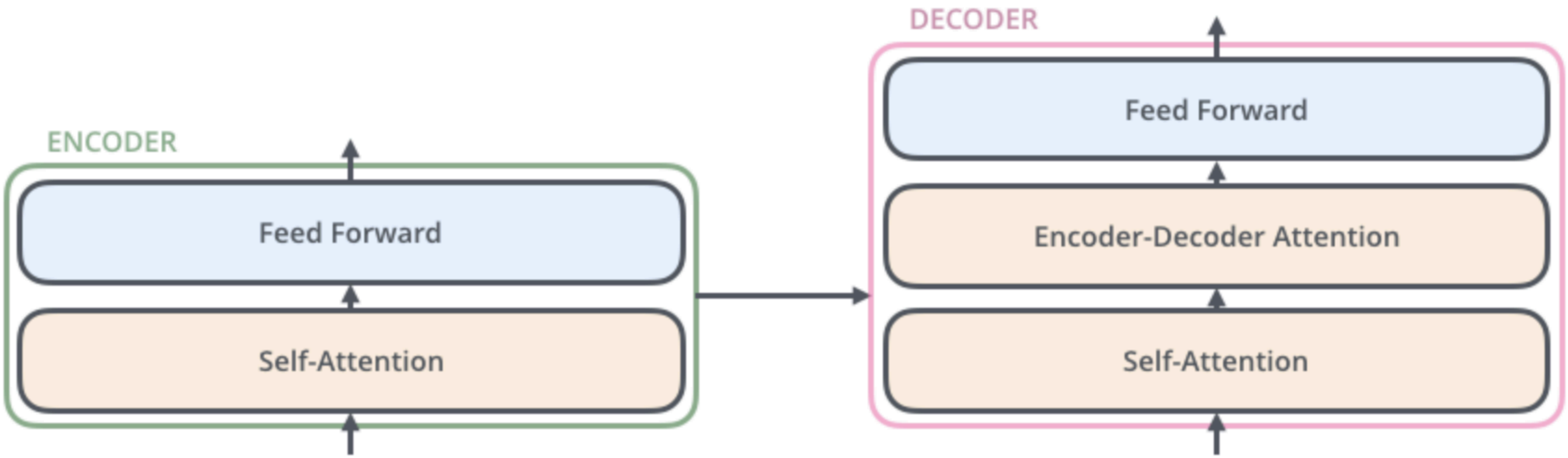
Transformer : Encoder

The encoder units don't share weights.

Have two sub-layers: self-attention followed by a fully connected/feed forward (FF) layer.

All words will be input at the same time. So the FF layer's weights are reused across words

(Decoder has self-attention as well as the classical encoder-decoder attention)

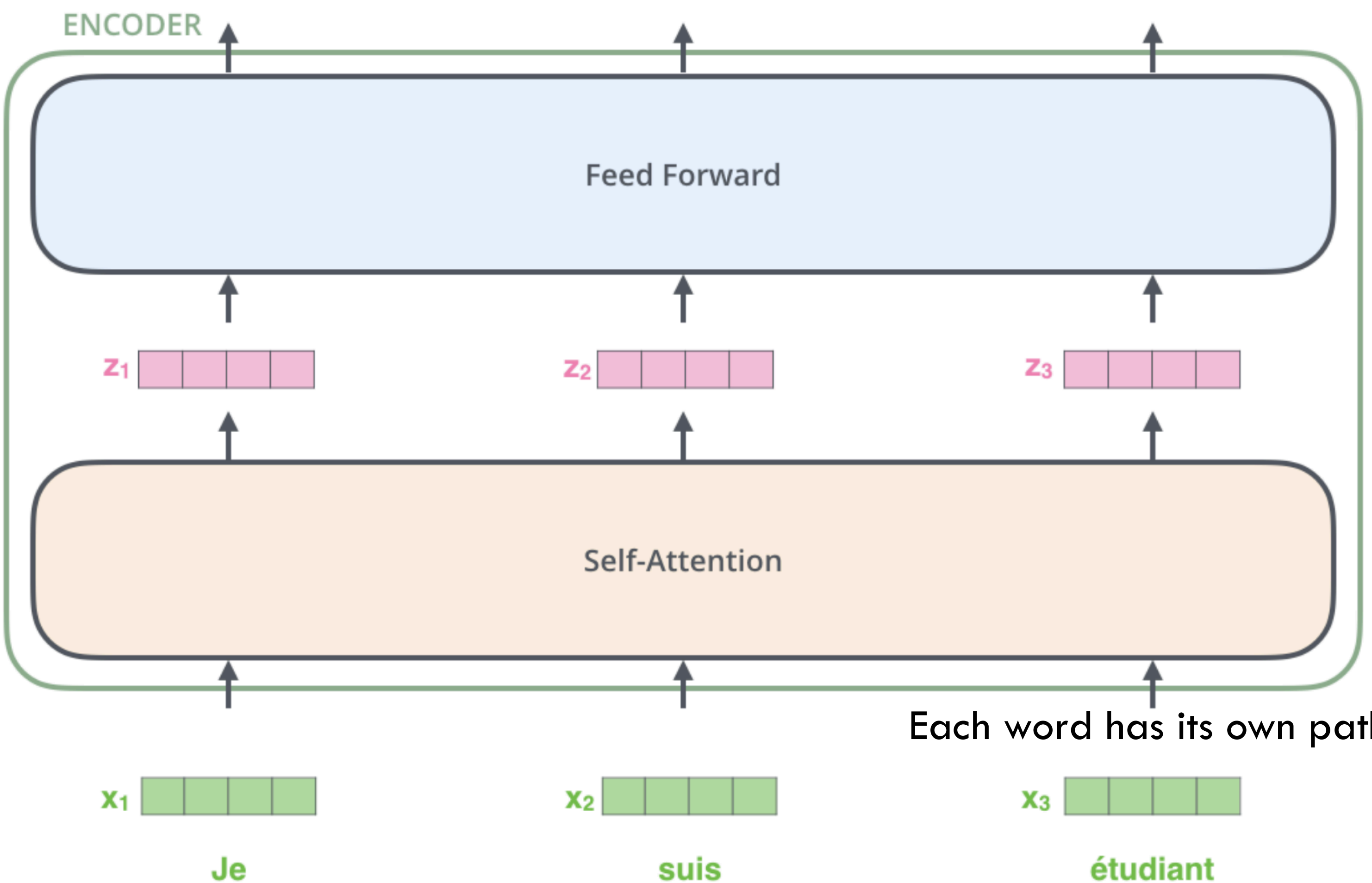


Self-attention helps 'transform' inputs taking other inputs into consideration

[1]Reference: https://jalammar.github.io/illustrated-transformer/

Transformer : Encoder

Lets start with vector embeddings of words (fix some max length say 20) at the lowest/starting layer. Say the embedding size is 512.
In other layers, its not embeddings but vector outputs of previous encoder units



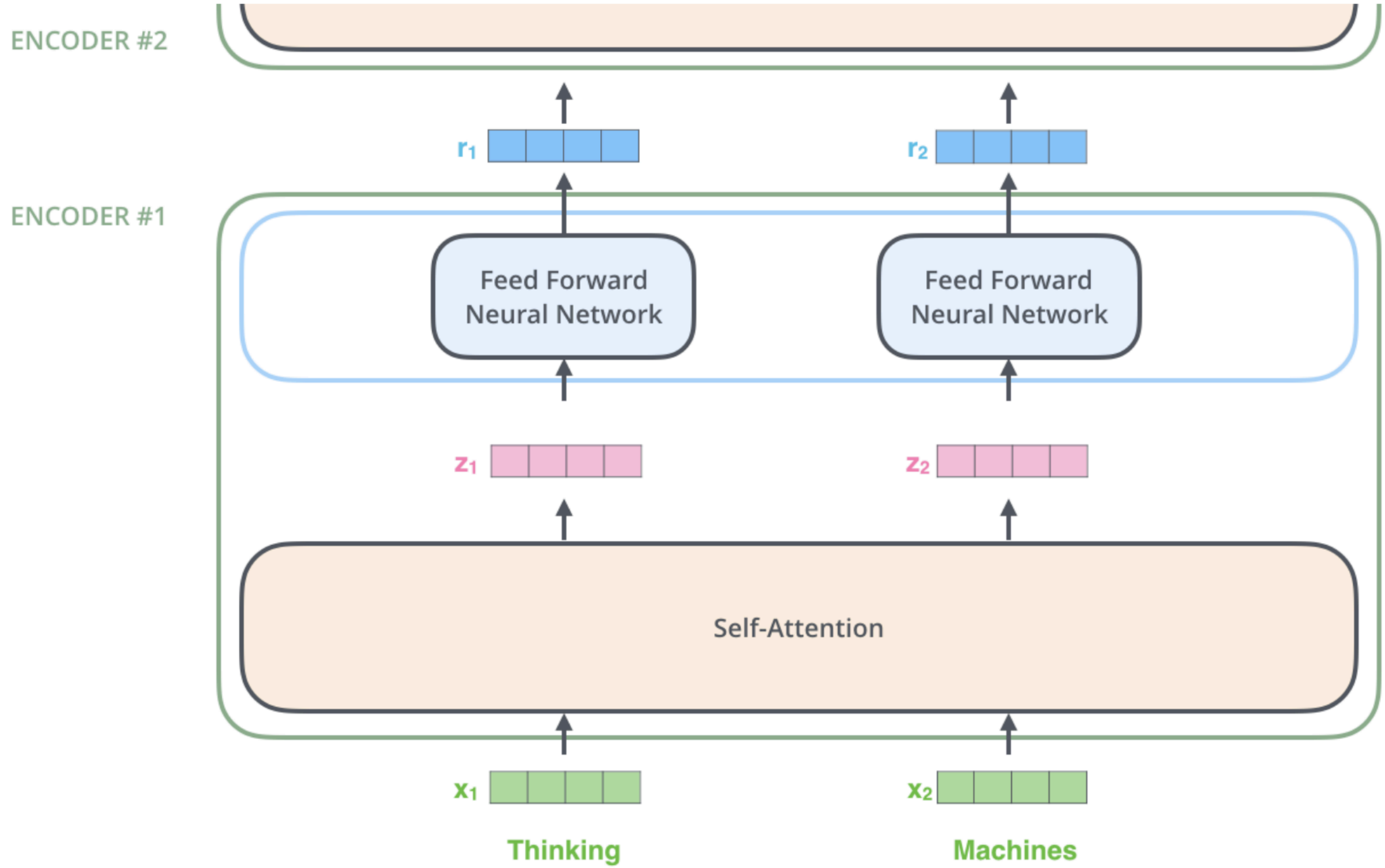
Each word has its own path till the end

[1]Reference: https://jalammar.github.io/illustrated-transformer/

Transformer : Encoder

Transformation of each word depends on transformations of other words in each encoding unit

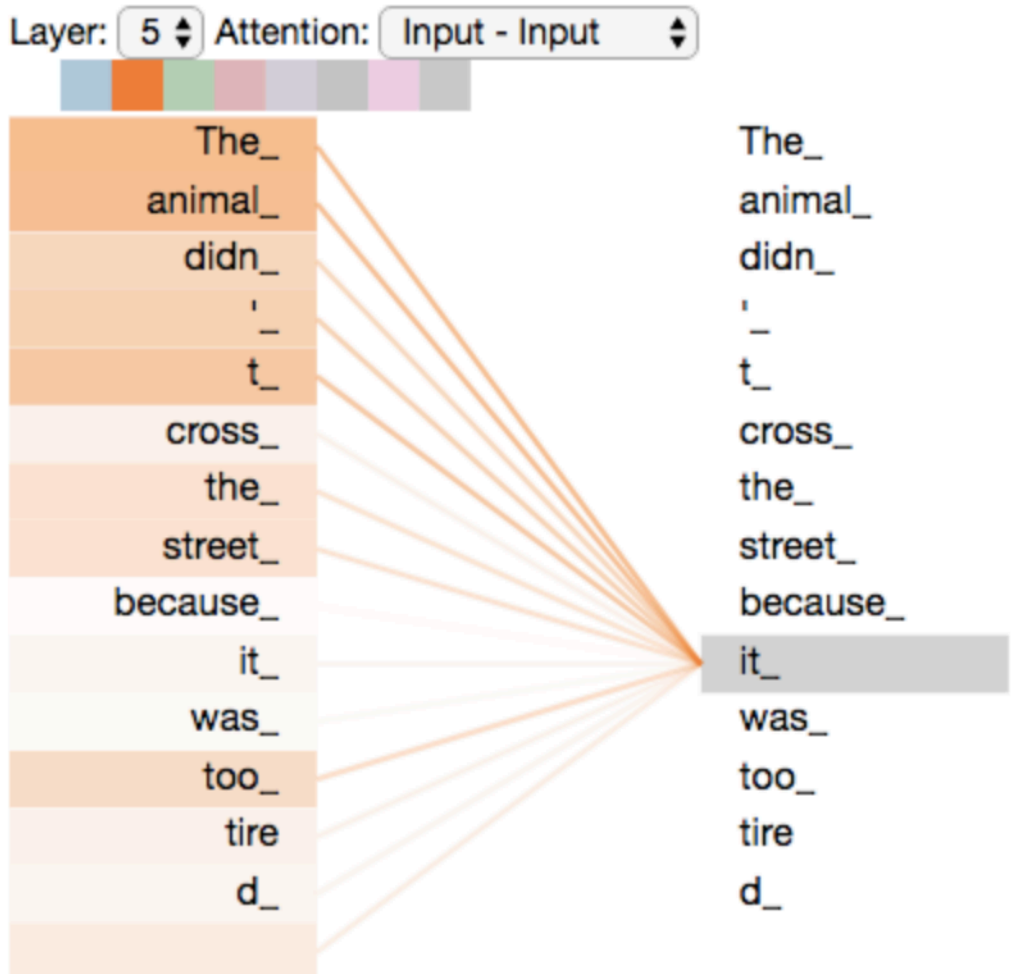
E.g.: z_1 depends on x_1 and x_2



[1]Reference: https://jalammar.github.io/illustrated-transformer/

Transformer : Self-Attention

Self-attention: while processing the word 'it' below, attend to words related to it

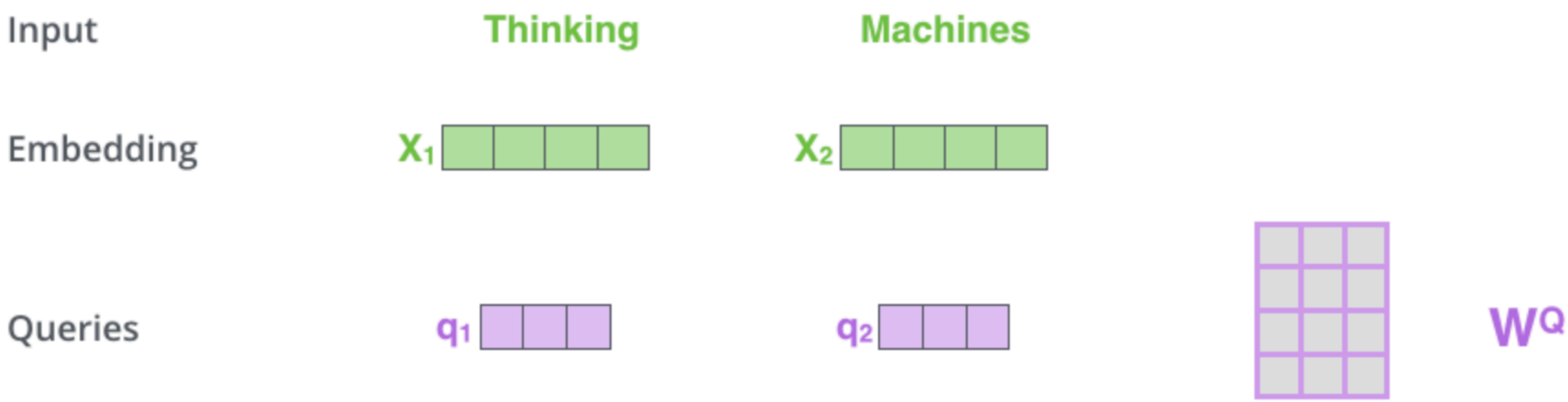


Self-attention: helps better encode the word 'it'. Analogous to cell state changes in LSTMs.

[1]Reference: https://jalammar.github.io/illustrated-transformer/

Transformer : Self-Attention

Create three vectors for each input: q, k and v using three matrices (that need to be learned)



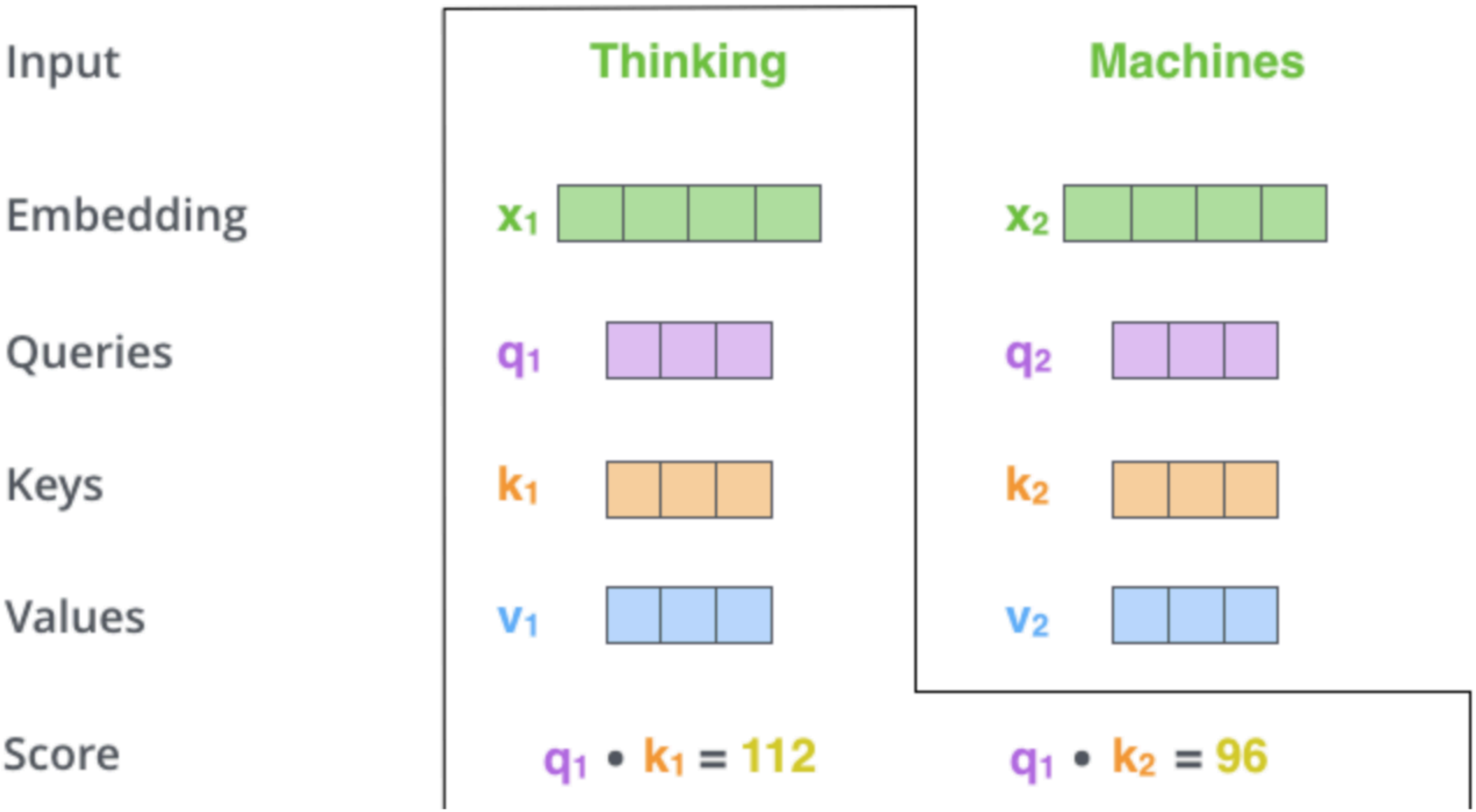
Their dimension is typically chosen to be smaller, e.g., 64

The use of names query, key and value is for interpretation (just like in LSTMs)

[1]Reference: https://jalammar.github.io/illustrated-transformer/

Transformer : Self-Attention

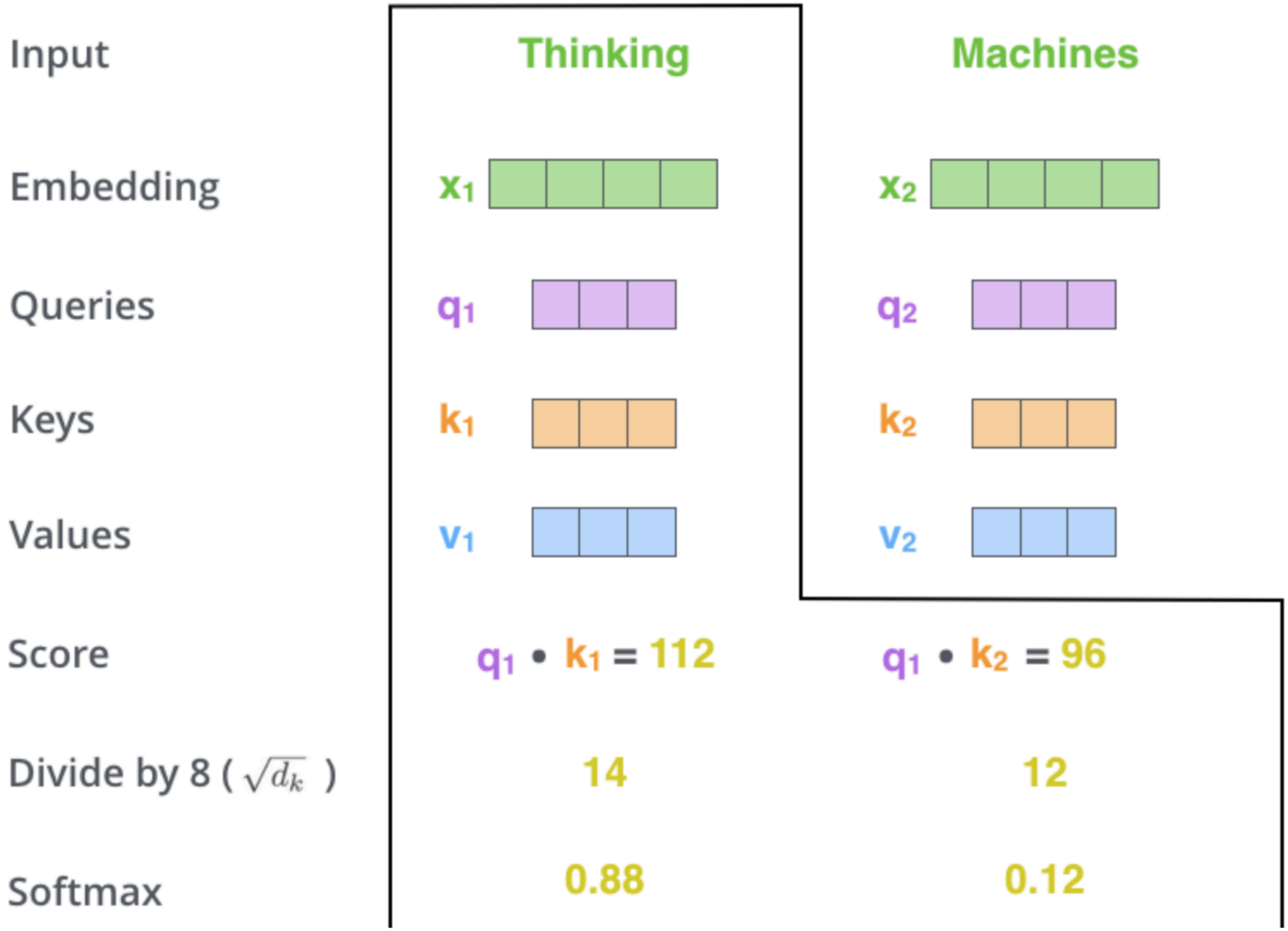
After create the three vectors, a set of scores per input word are calculated.



[1]Reference: https://jalammar.github.io/illustrated-transformer/

Transformer : Self-Attention

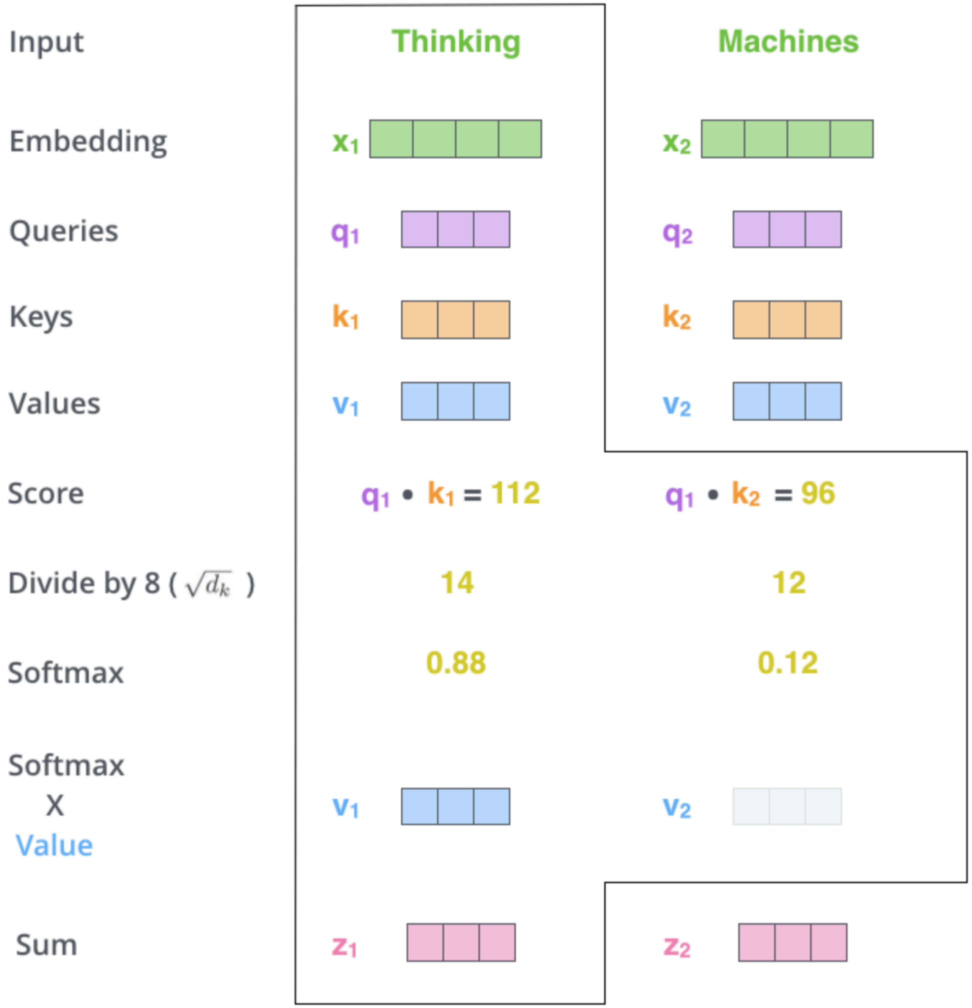
These scores are then normalized. They determine how much attention is needed on itself and other words (e.g., Machines below).



[1]Reference: https://jalammar.github.io/illustrated-transformer/

Transformer : Self-Attention

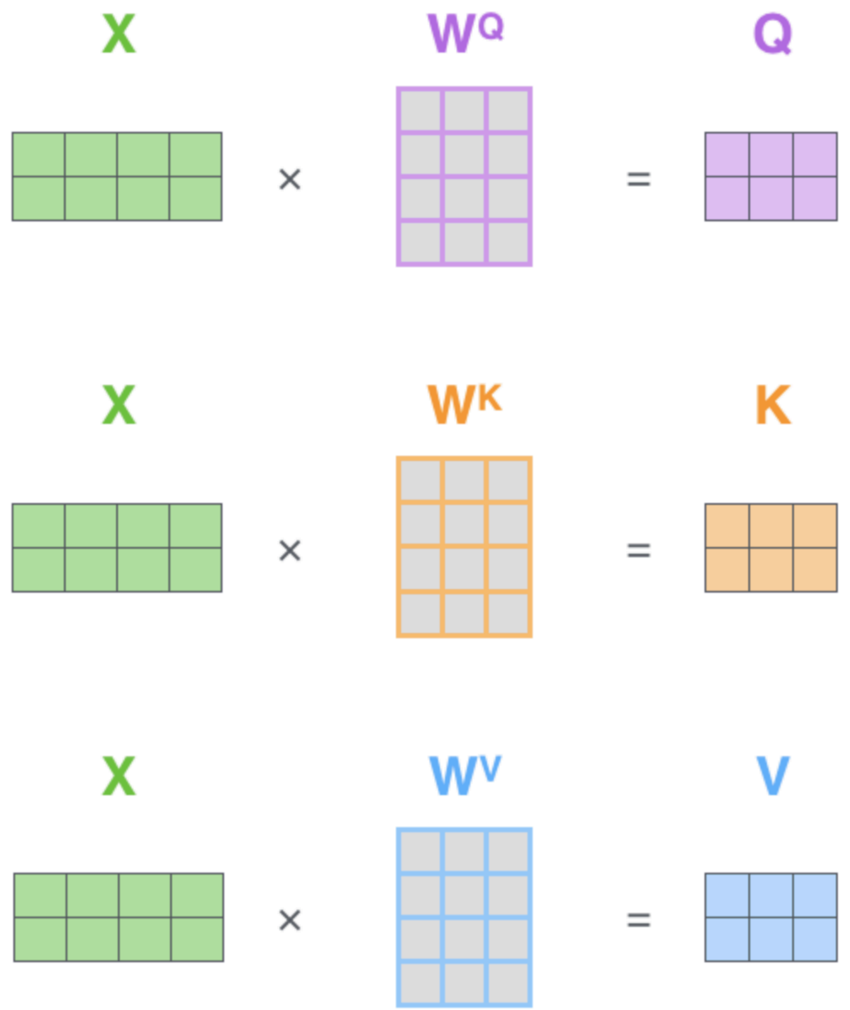
A score weighted sum of 'value' vectors is the output encoding of the input in this encoder unit



[1]Reference: https://jalammar.github.io/illustrated-transformer/

Transformer : Self-Attention

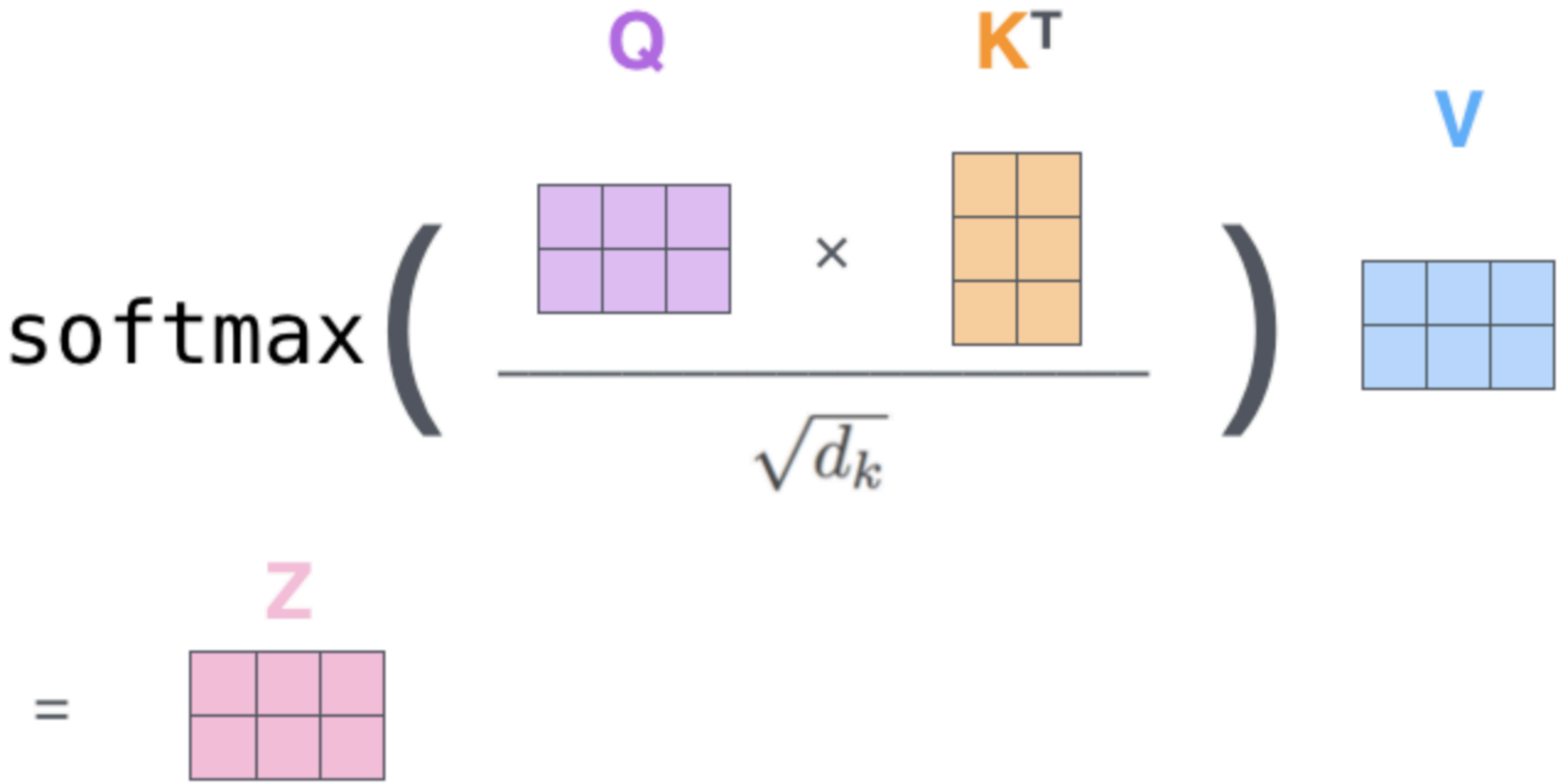
The computation of queries, keys and values actually happens via matrix multiplication.



[1]Reference: https://jalammar.github.io/illustrated-transformer/

Transformer : Self-Attention

Similarly, the scores are also computed using matrix-matrix multiplications.

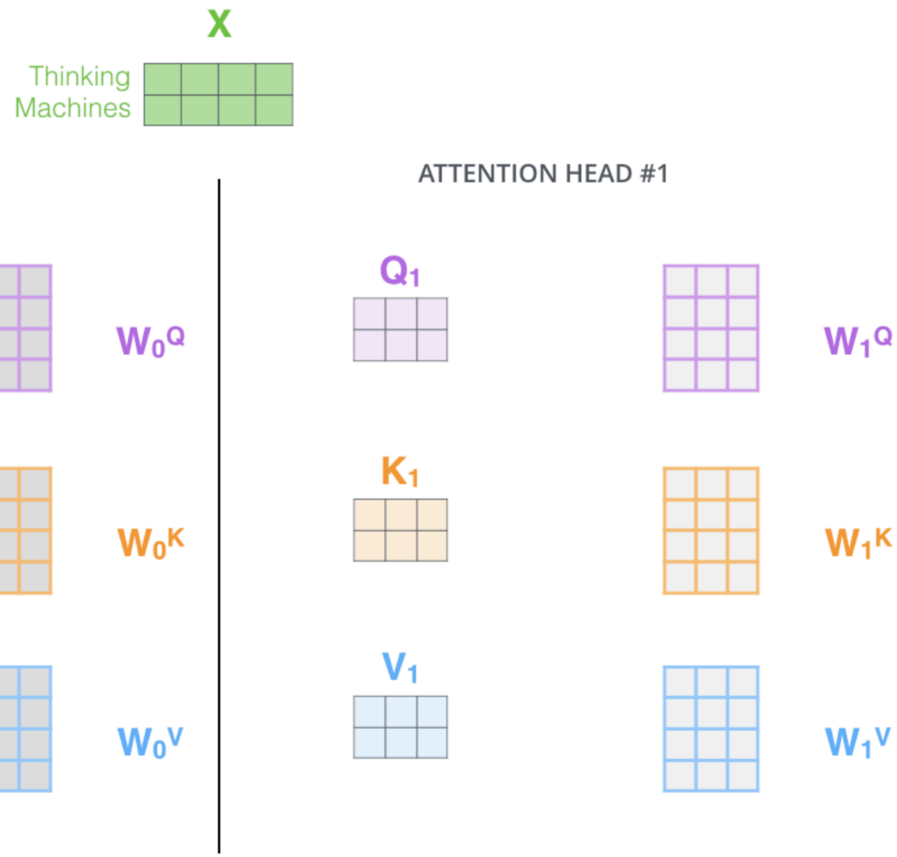


Exercise: This should seem similar to the encoder-decoder attention we saw in seq2seq models

[1]Reference: https://jalammar.github.io/illustrated-transformer/

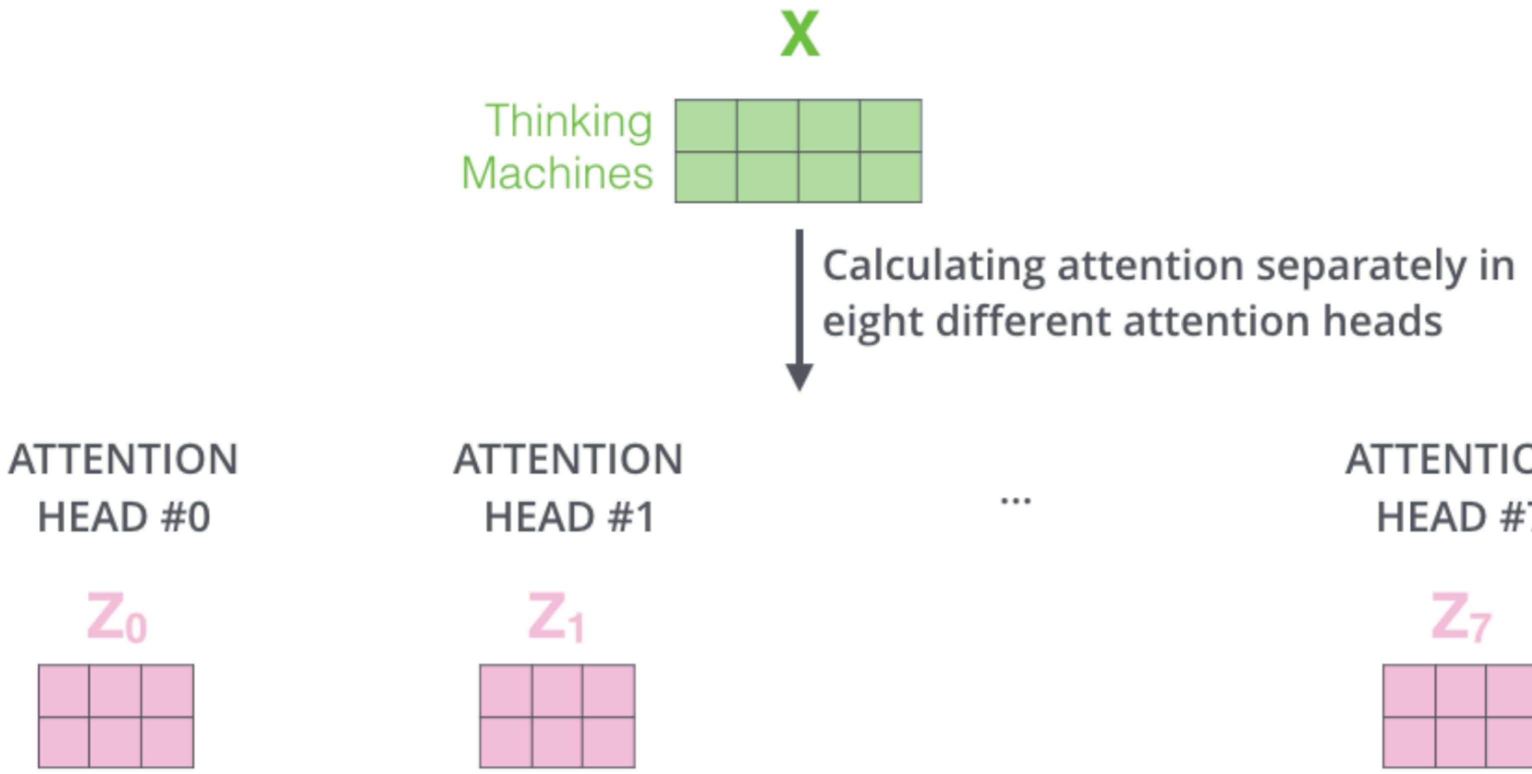
Transformer : Multi-Headed Attention

Multi-headed attention: doing multiple attention computations in parallel.
Empirically validated.
How it might help: (a) model can better focus on multiple inputs, (b) get 'different' representations (basically we can get different 'output embeddings' due to random initializations)



[1]Reference: https://jalammar.github.io/illustrated-transformer/

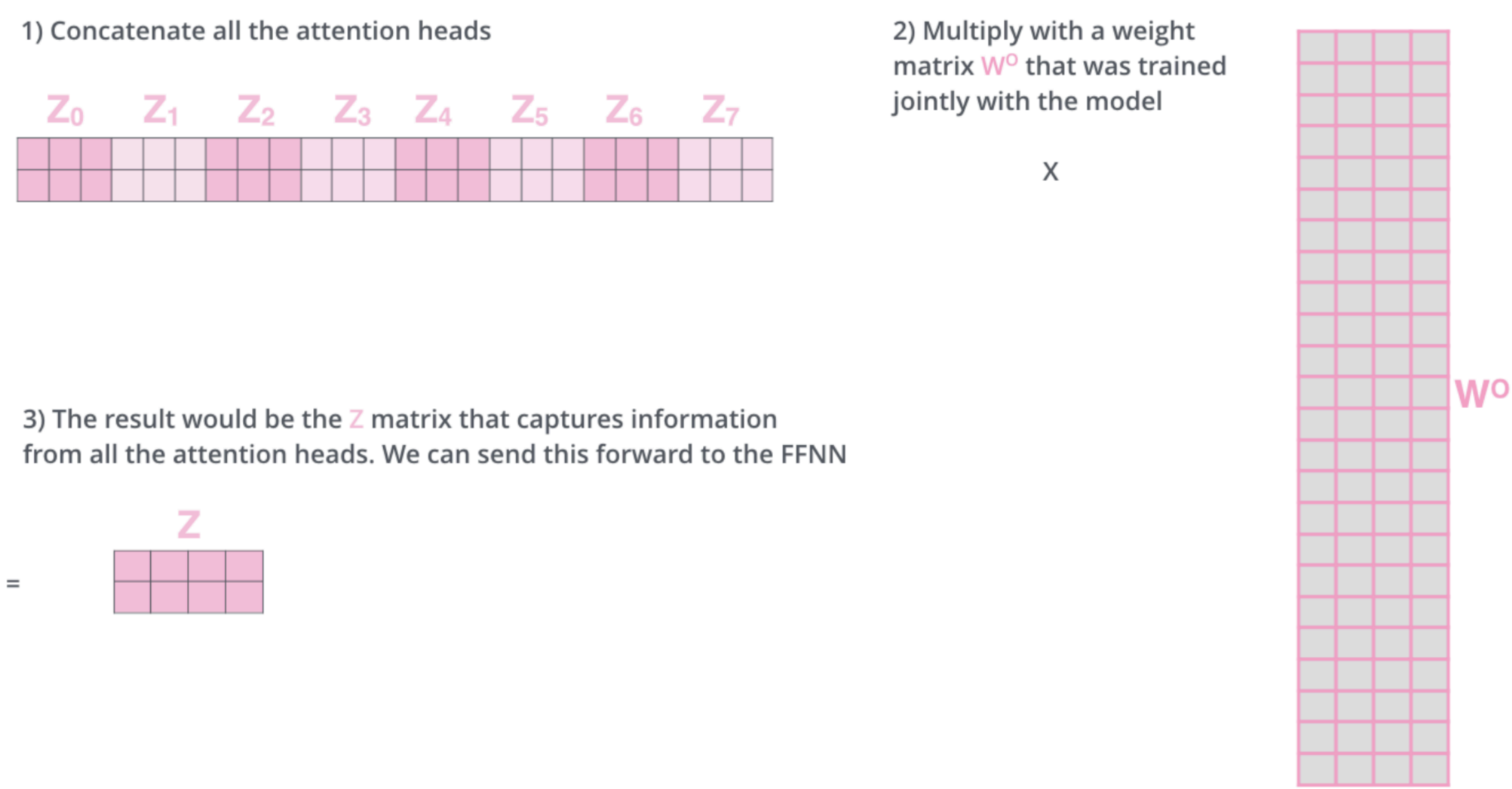
Transformer : Multi-Headed Attention



[1]Reference: https://jalammar.github.io/illustrated-transformer/

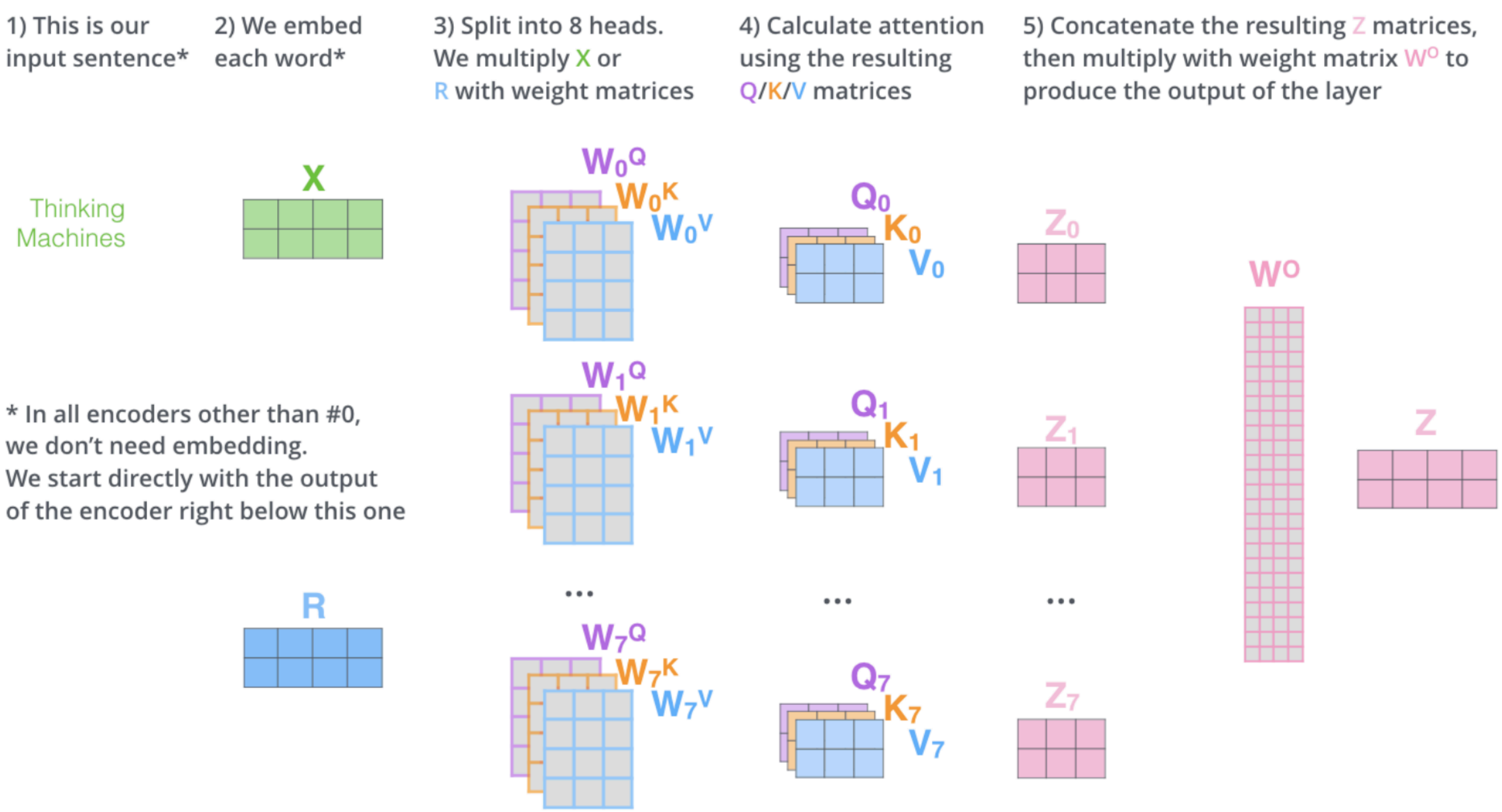
Transformer : Multi-Headed Attention

Need to merge these vectors before passing onto the FF layer: concatenate and transform.



[1]Reference: https://jalammar.github.io/illustrated-transformer/

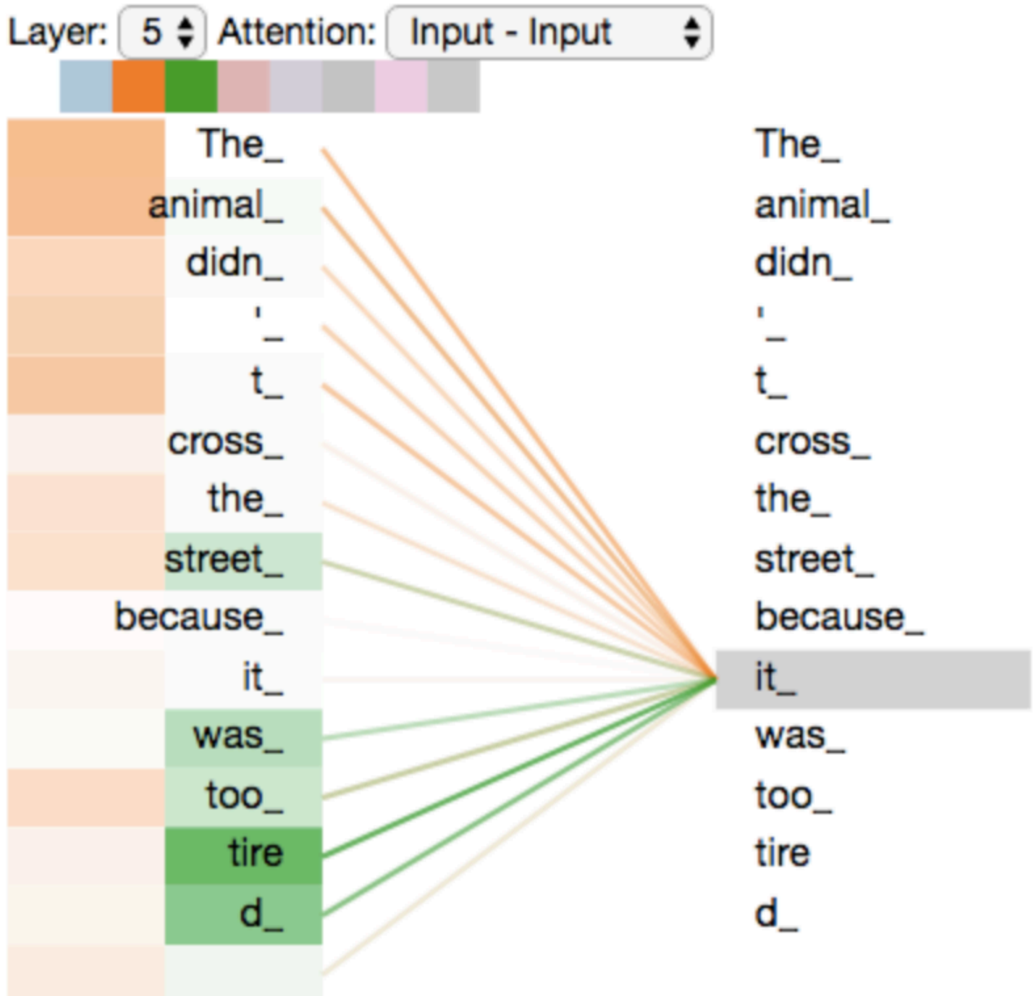
Transformer : Multi-Headed Attention



[1]Reference: https://jalammar.github.io/illustrated-transformer/

Transformer : Multi-Headed Attention

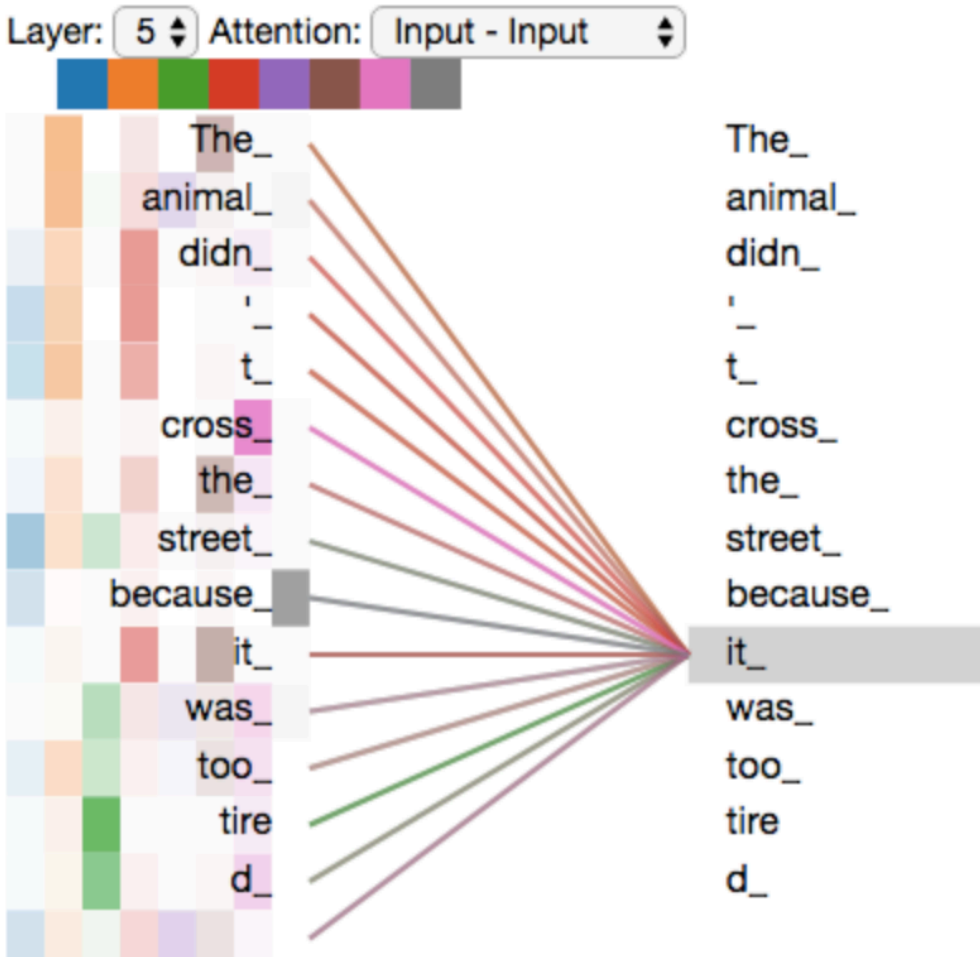
Two heads are focusing on two different related words. The orange head focuses on animal, the green head focusses on tired.



[1]Reference: https://jalammar.github.io/illustrated-transformer/

Transformer : Multi-Headed Attention

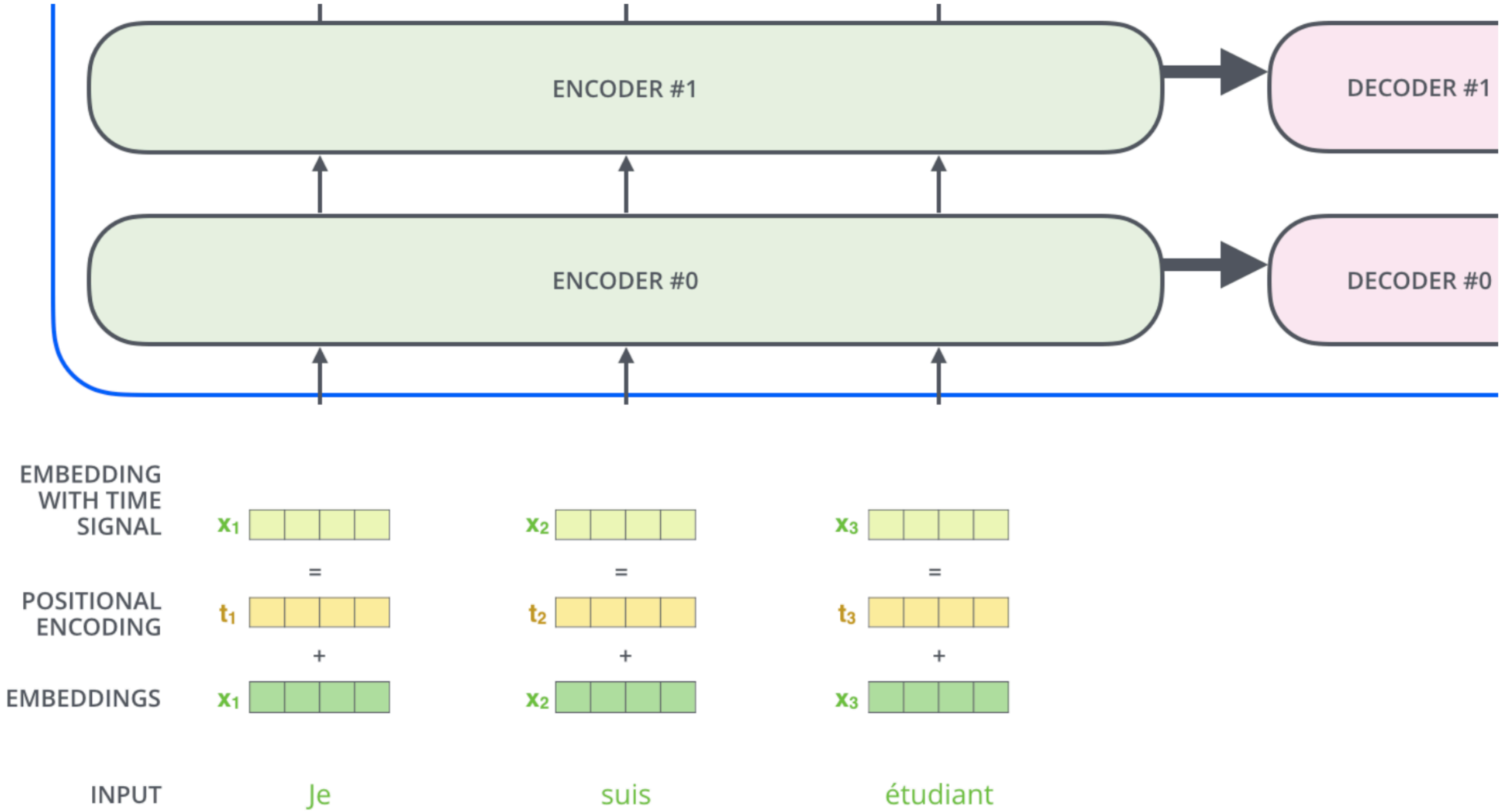
Visualizing all heads does not give an interpretation. But it works well!



[1]Reference: https://jalammar.github.io/illustrated-transformer/

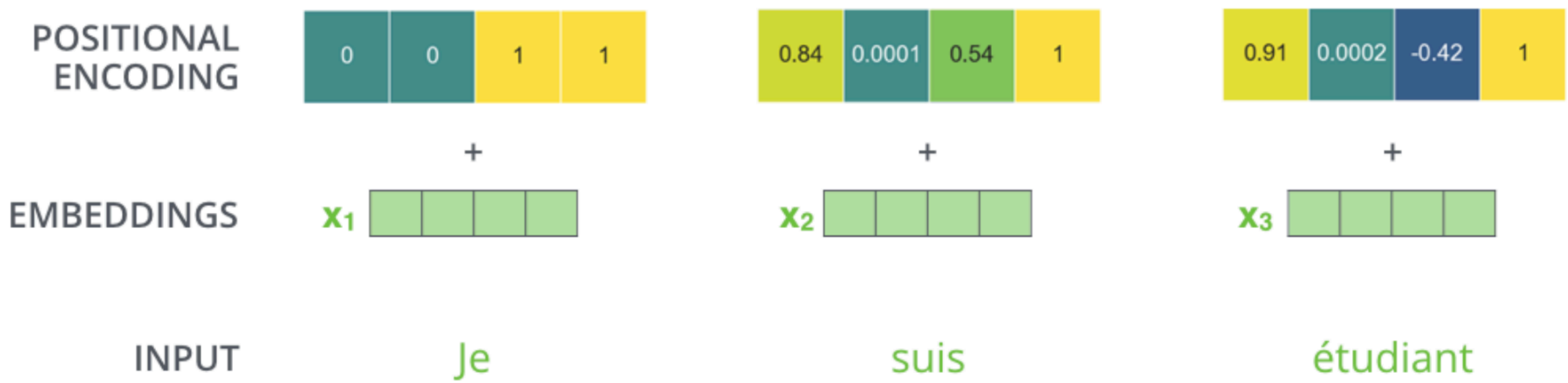
Transformer : Positional Encoding

We need to account for word ordering. We can do that by adding a 'disambiguating vector' to each embedding.



[1]Reference: https://jalammar.github.io/illustrated-transformer/

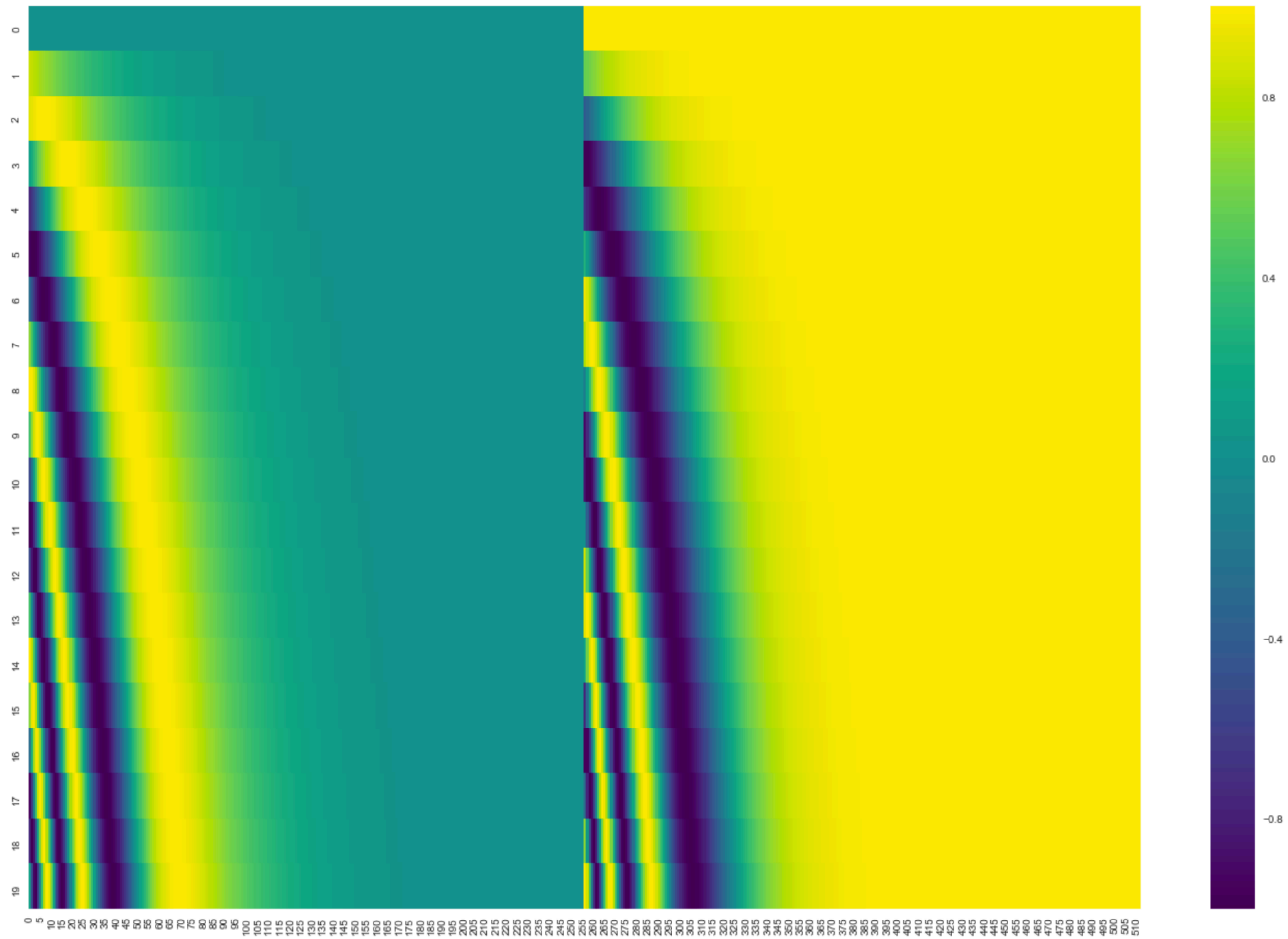
Transformer : Positional Encoding



[1]Reference: https://jalammar.github.io/illustrated-transformer/

Transformer : Positional Encoding

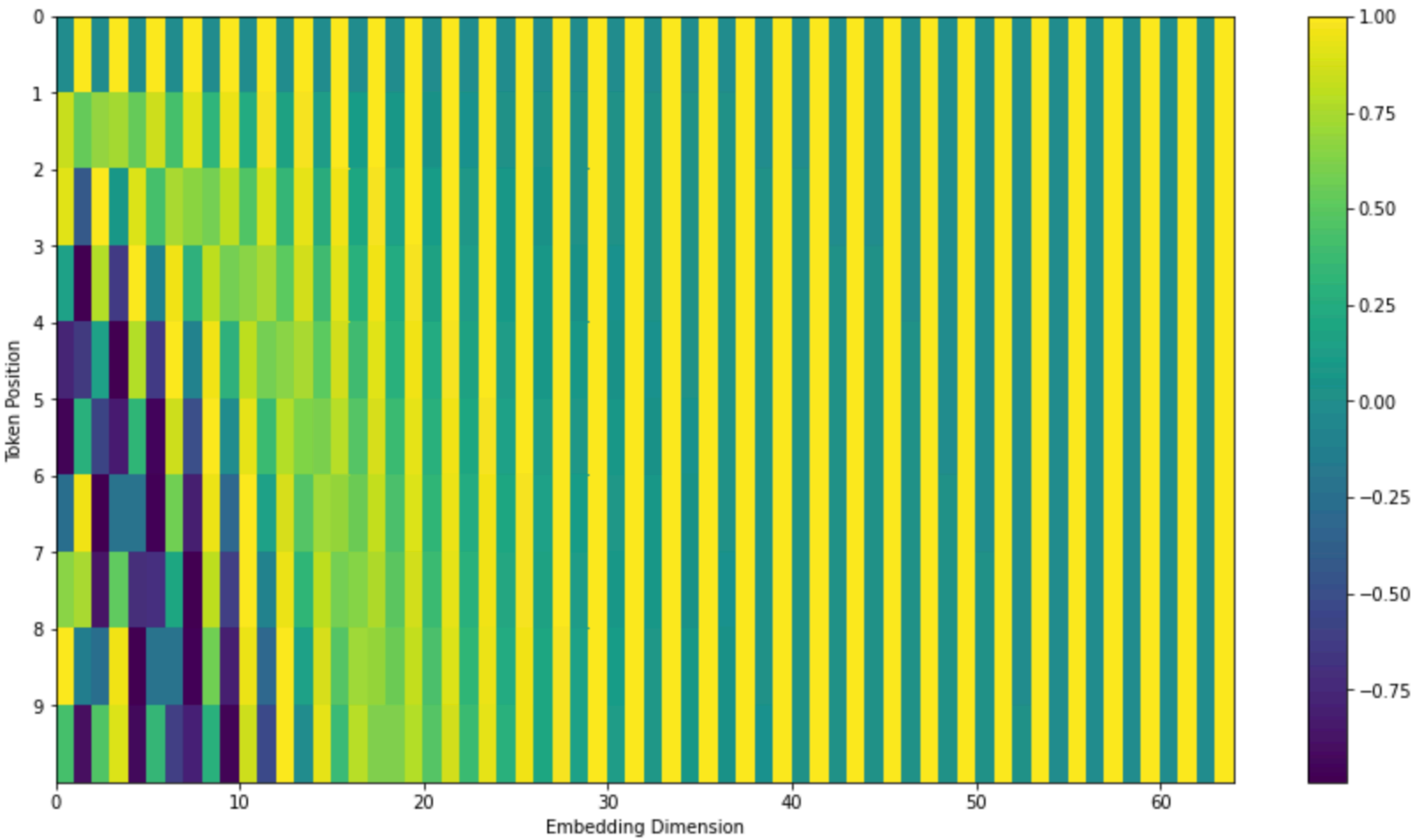
Each row below is an example positional encoding vector (512 dimensional, from Transformer2Transformer).



[1]Reference: https://jalammar.github.io/illustrated-transformer/

Transformer : Positional Encoding

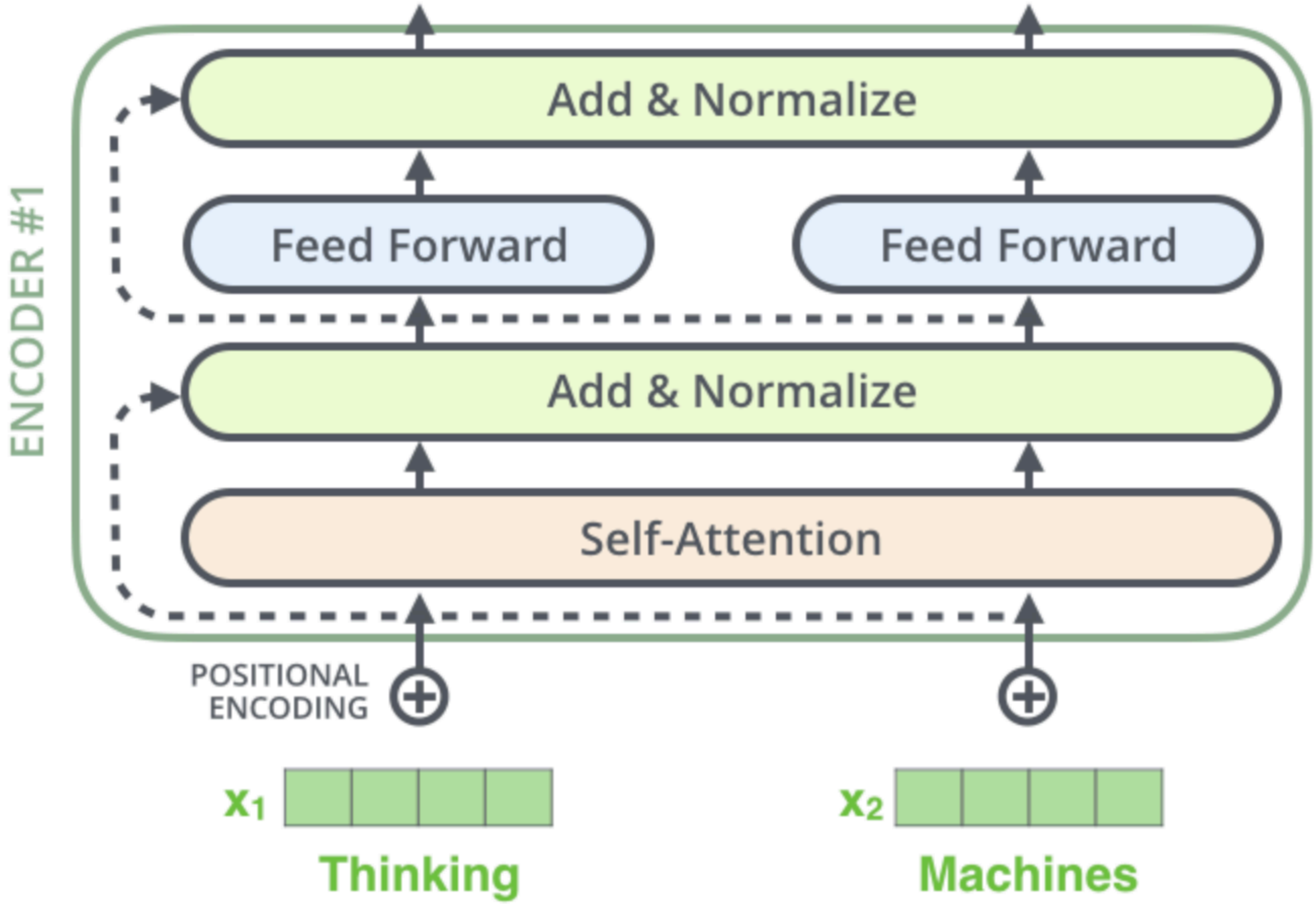
Each row below is an example positional encoding vector (64 dimensional).



[1]Reference: https://jalammar.github.io/illustrated-transformer/

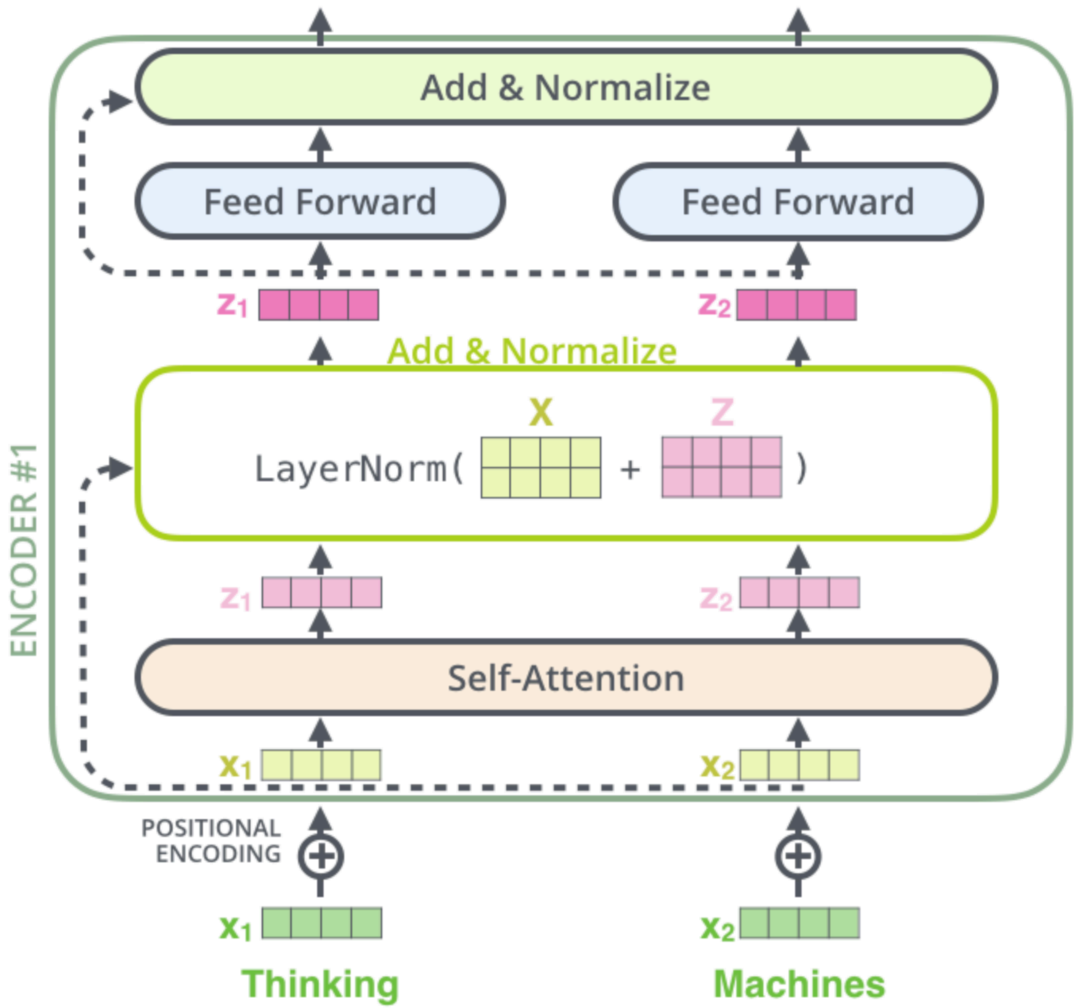
Transformer : Residual Connections

Each sub-layer has residual connections that skip it, and these get normalized with processed vectors.



[1]Reference: https://jalammar.github.io/illustrated-transformer/

Transformer : Residual Connections



[1]Reference: https://jalammar.github.io/illustrated-transformer/

Transformer : Residual Connections

Layer Normalization

Jimmy Lei Ba, Jamie Ryan Kiros, Geoffrey E. Hinton

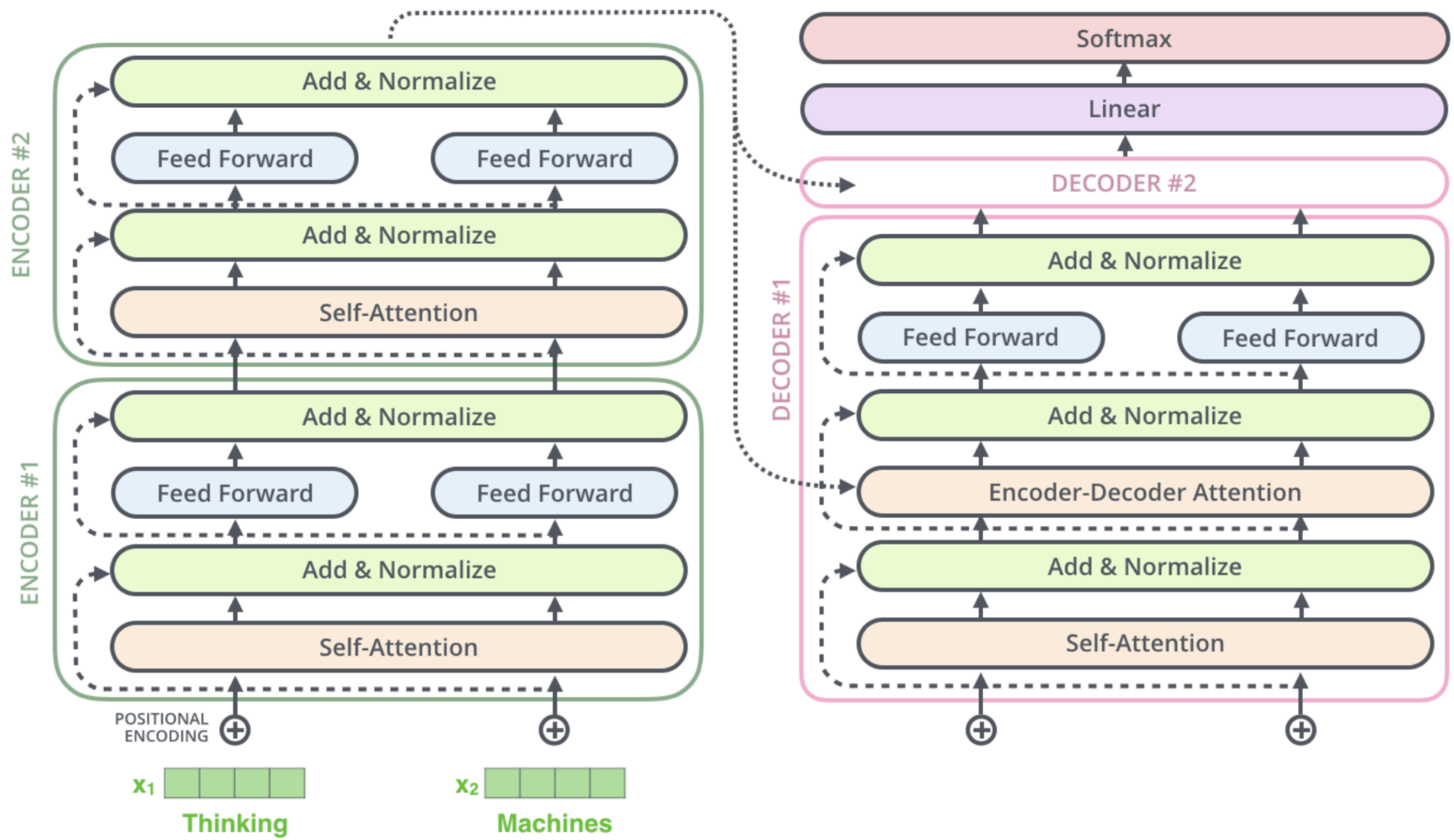
Download PDF

Training state-of-the-art, deep neural networks is computationally expensive. One way to reduce the training time is to normalize the activities of the neurons. A recently introduced technique called batch normalization uses the distribution of the summed input to a neuron over a mini-batch of training cases to compute a mean and variance which are then used to normalize the summed input to that neuron on each training case. This significantly reduces the training time in feed-forward neural networks. However, the effect of batch normalization is dependent on the mini-batch size and it is not obvious how to apply it to recurrent neural networks. In this paper, we transpose batch normalization into layer normalization by computing the mean and variance used for normalization from all of the summed inputs to the neurons in a layer on a single training case. Like batch normalization, we also give each neuron its own adaptive bias and gain which are applied after the normalization but before the non-linearity. Unlike batch normalization, layer normalization performs exactly the same computation at training and test times. It is also straightforward to apply to recurrent neural networks by computing the normalization statistics separately at each time step. Layer normalization is very effective at stabilizing the hidden state dynamics in recurrent networks. Empirically, we show that layer normalization can substantially reduce the training time compared with previously published techniques.

[1] Reference: https://arxiv.org/abs/1607.06450

Transformer : Residual Connections

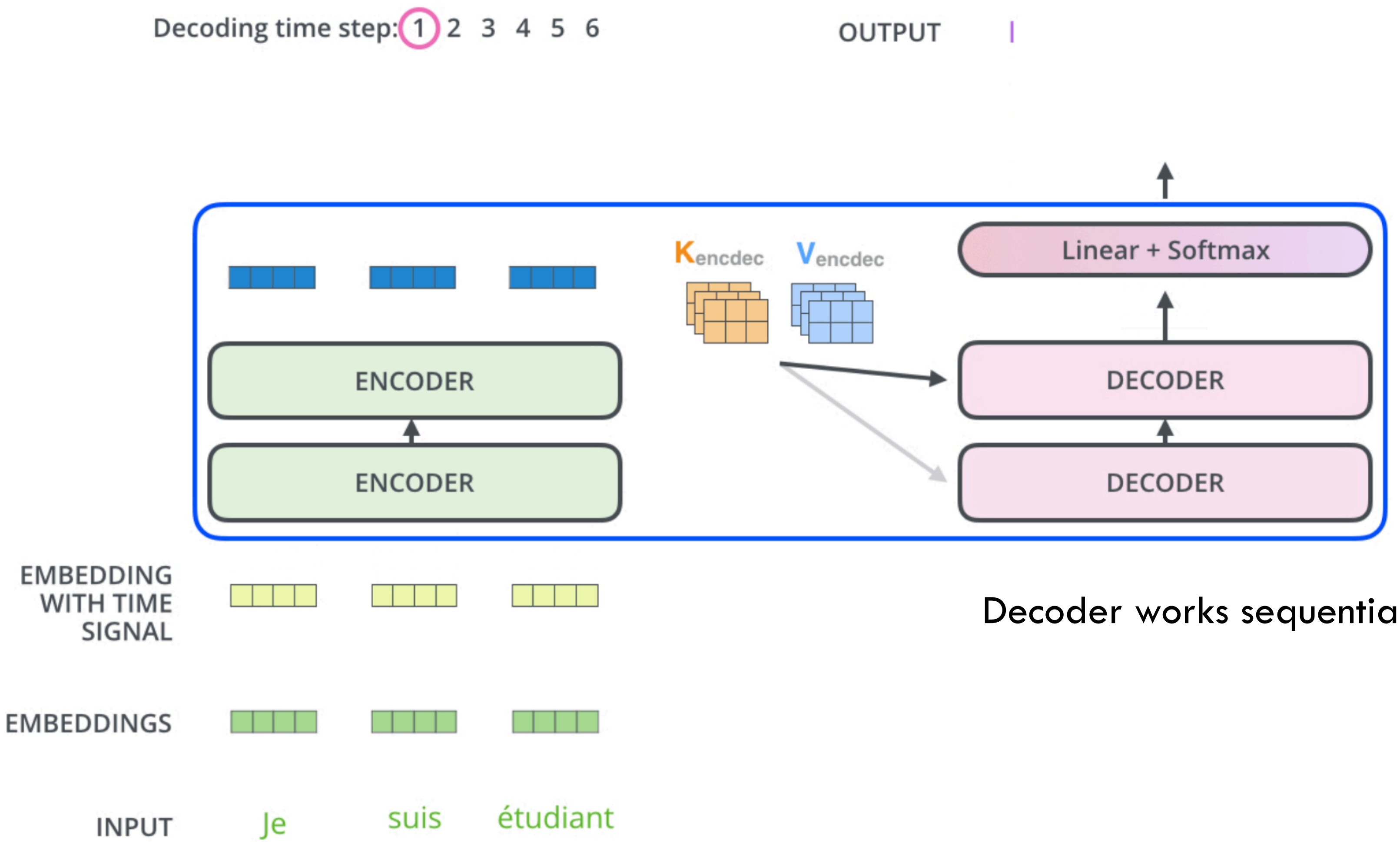
A 2-layer transfer example is below. Layer-normalization is part of the decoder unit as well.



[1]Reference: https://jalammar.github.io/illustrated-transformer/

Transformer : Decoder

The outputs of the top encoder unit are transformed to keys and values. These will be used in **all** encoder-decoder attention sub-layers.

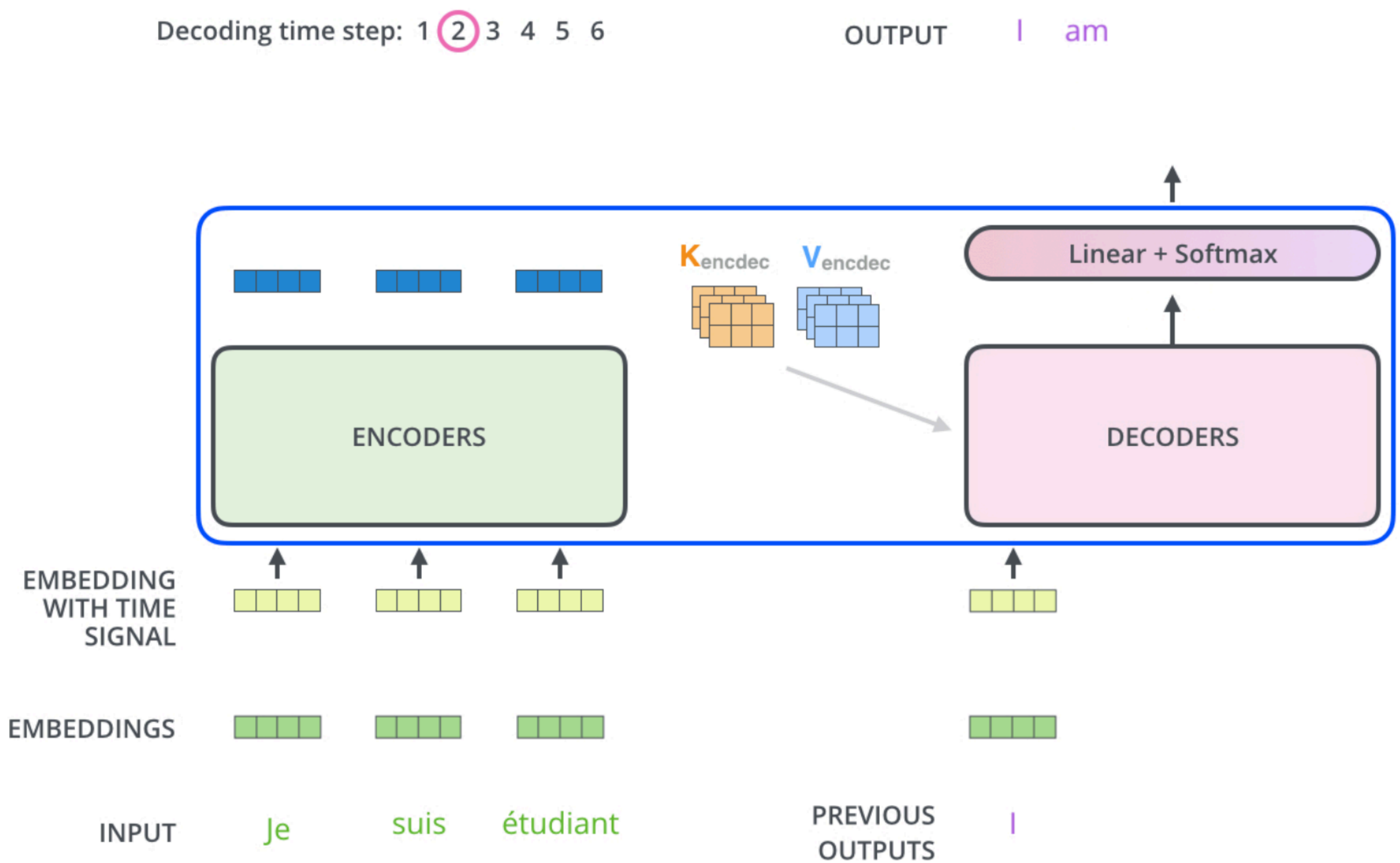


Decoder works sequentially

[1]Reference: https://jalammar.github.io/illustrated-transformer/

Transformer : Decoder

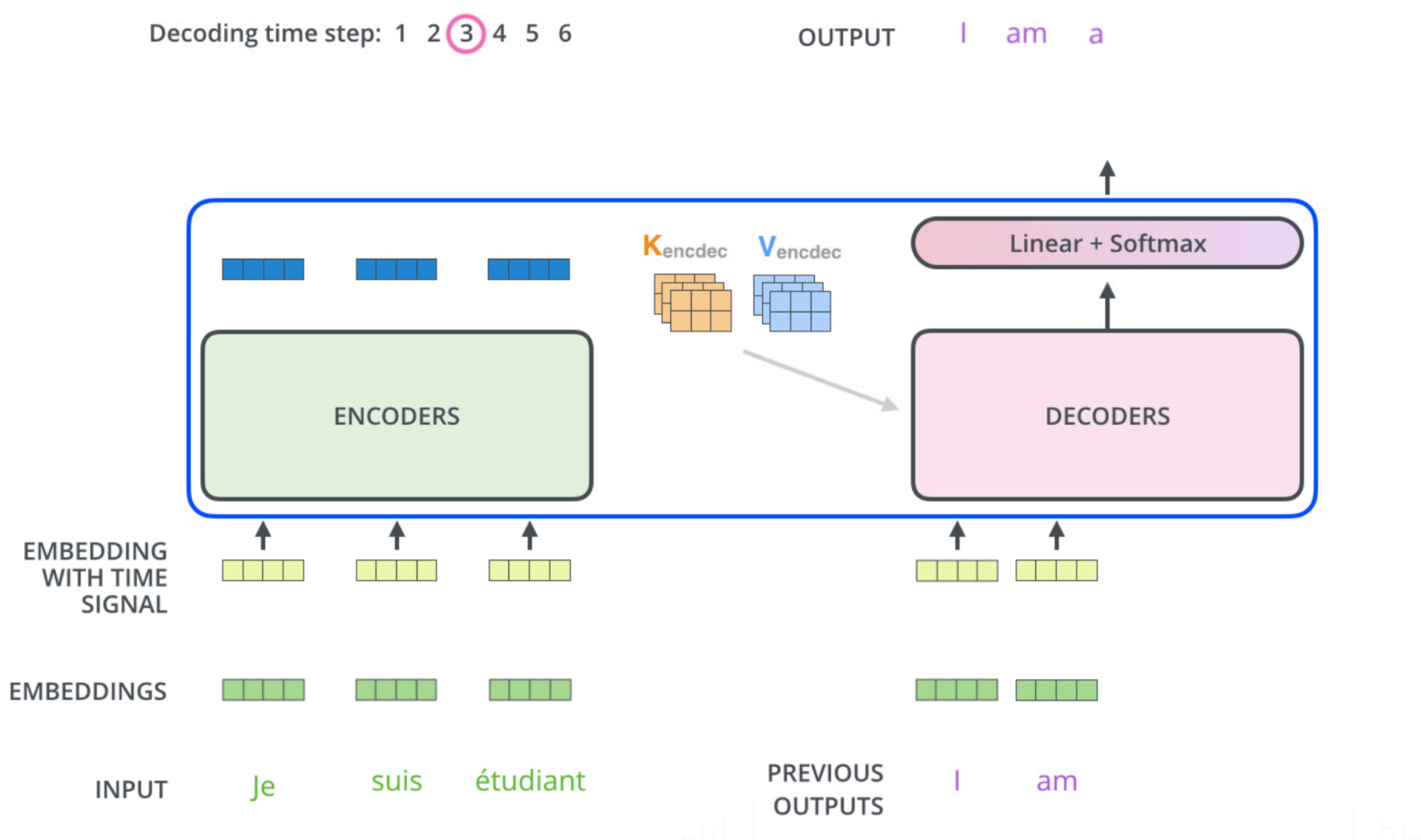
Decoder self-attention: can only attend to previous words. Achieved by **masking**



[1]Reference: https://jalammar.github.io/illustrated-transformer/

Transformer : Decoder

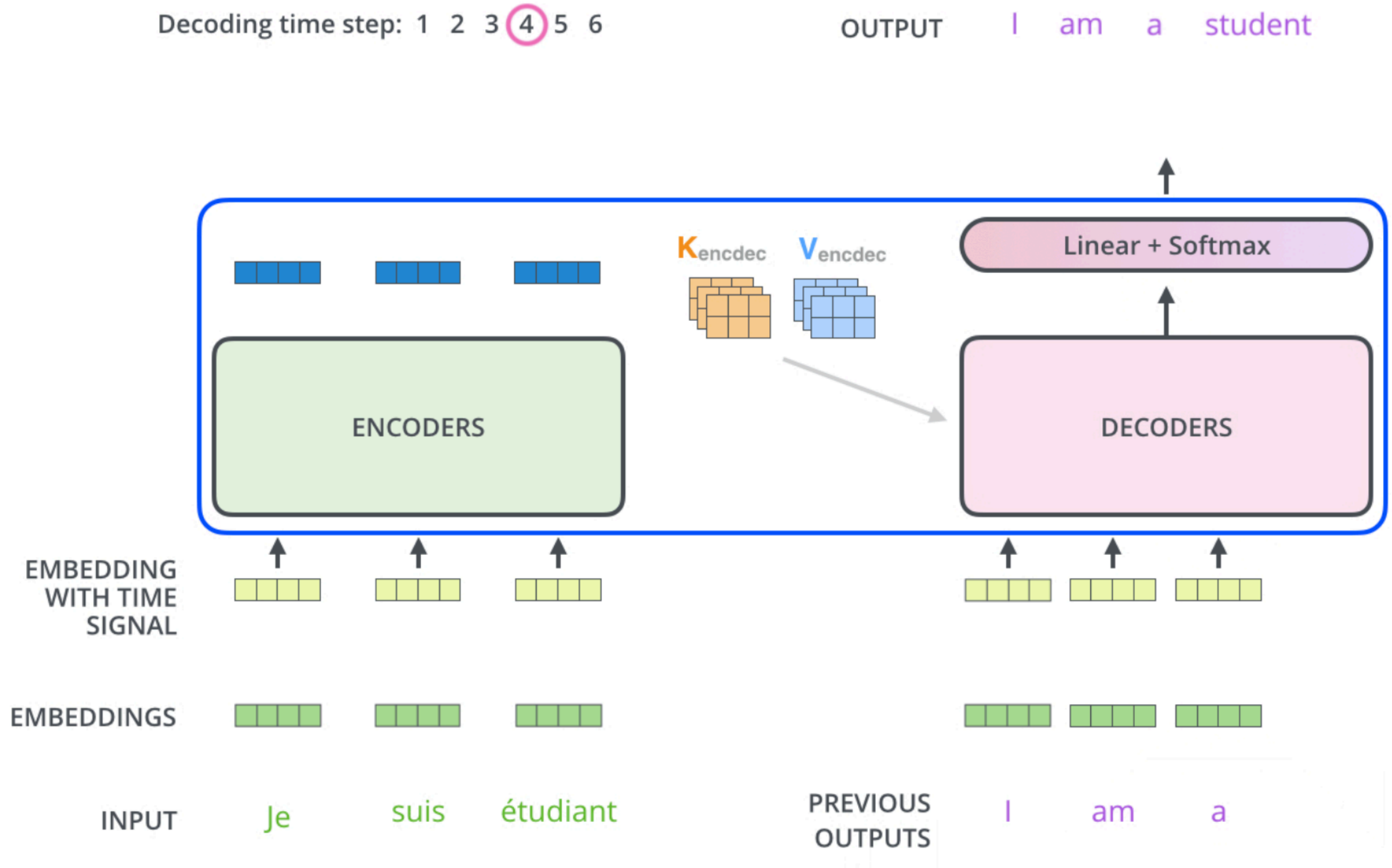
Decoder's encoder-decoder attention: queries are generated from below (sub)-layers.



[1]Reference: https://jalammar.github.io/illustrated-transformer/

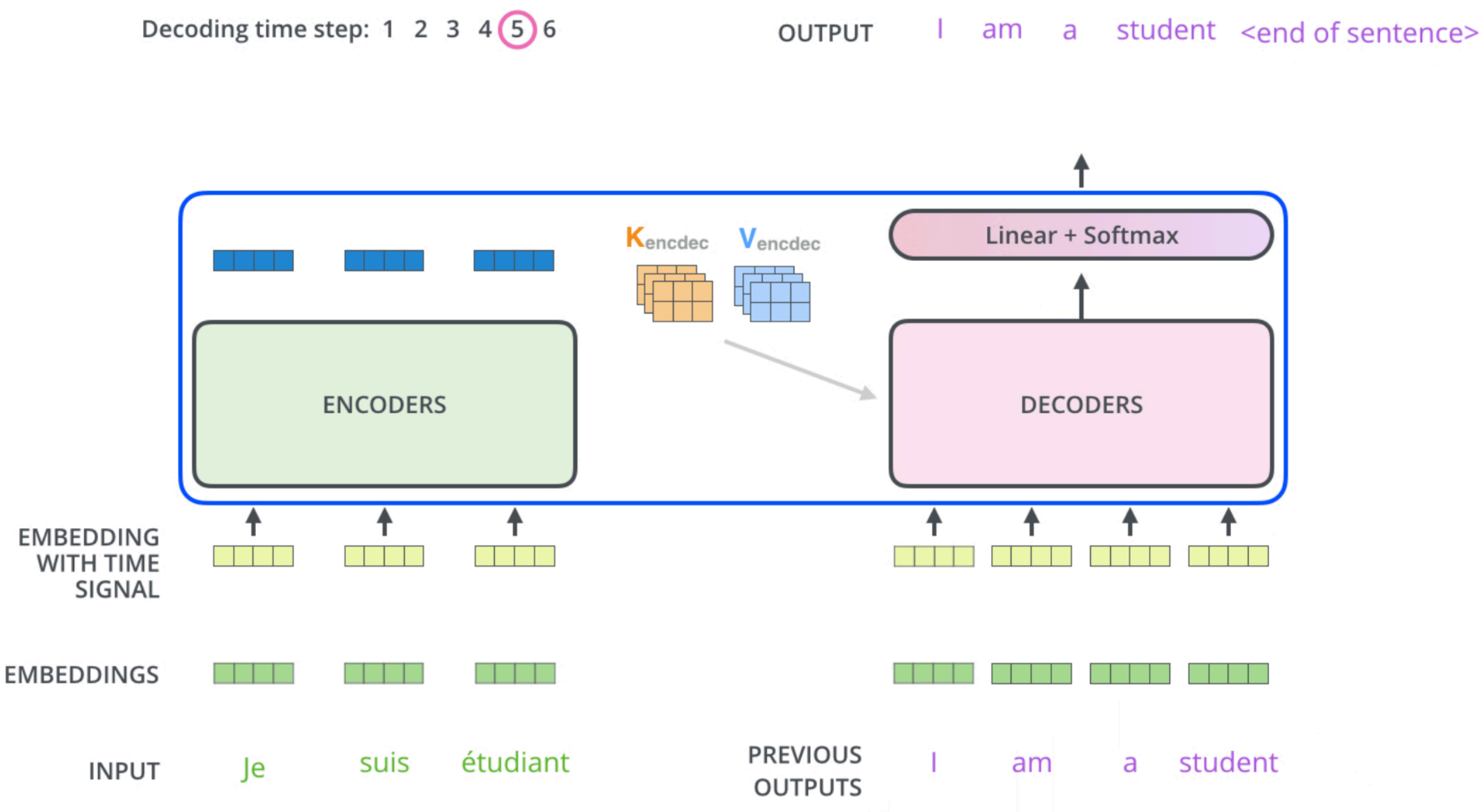
Transformer : Decoder

Decoder outputs are fed back sequentially. They are also embedded and positional encoding is added.



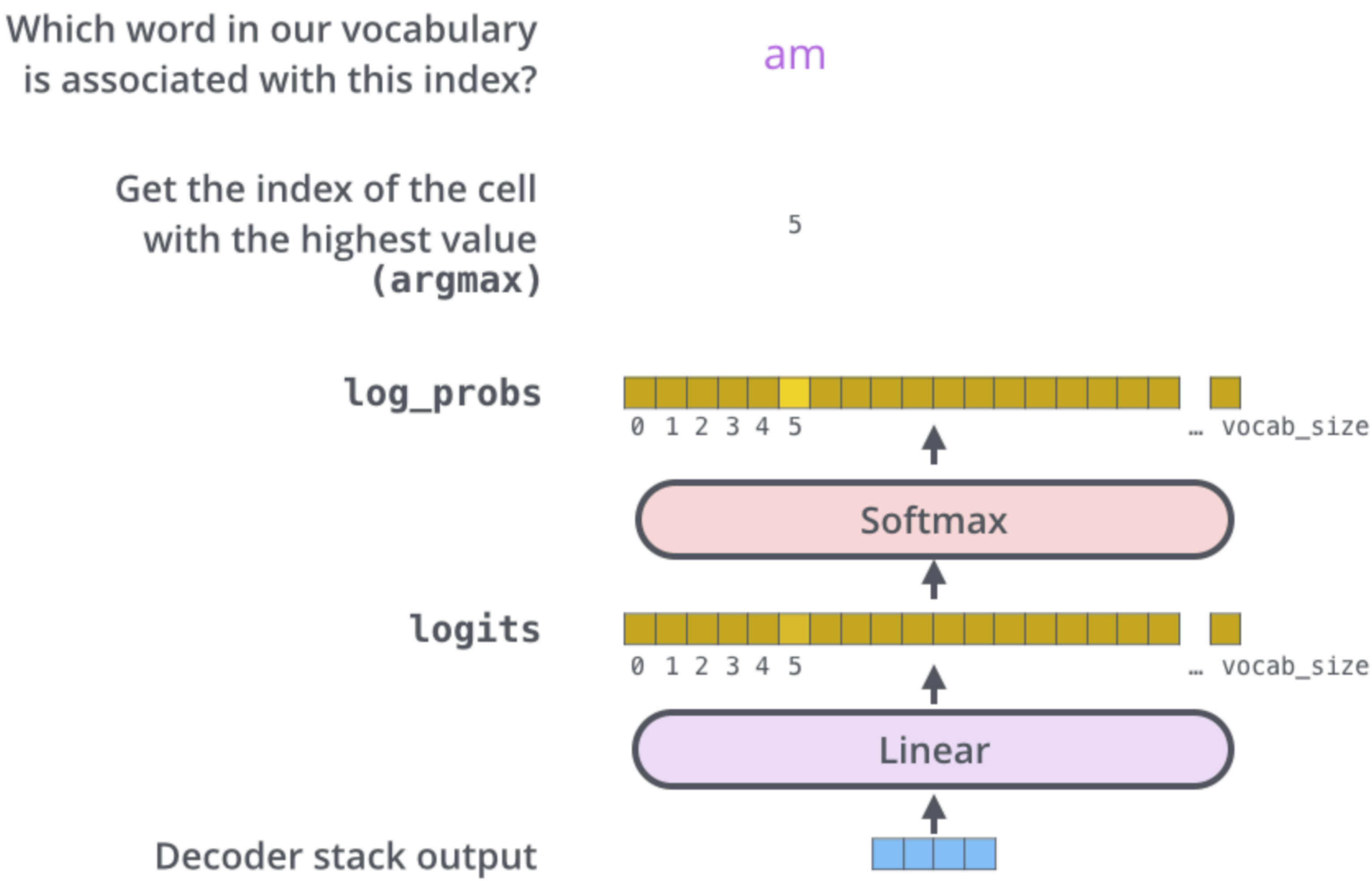
[1]Reference: https://jalammar.github.io/illustrated-transformer/

Transformer : Decoder



[1]Reference: https://jalammar.github.io/illustrated-transformer/

Transformer : Final Layer



[1]Reference: https://jalammar.github.io/illustrated-transformer/

Transformer : Training

The output is compared to ground truth translation after the forward pass.
E.g.: Consider a 6 word vocabulary as shown below.

Output Vocabulary

WORD	a	am	I	thanks	student	<eos>
INDEX	0	1	2	3	4	5

[1]Reference: https://jalammar.github.io/illustrated-transformer/

Transformer : Training

Output Vocabulary

WORD	a	am	I	thanks	student	<eos>
INDEX	0	1	2	3	4	5

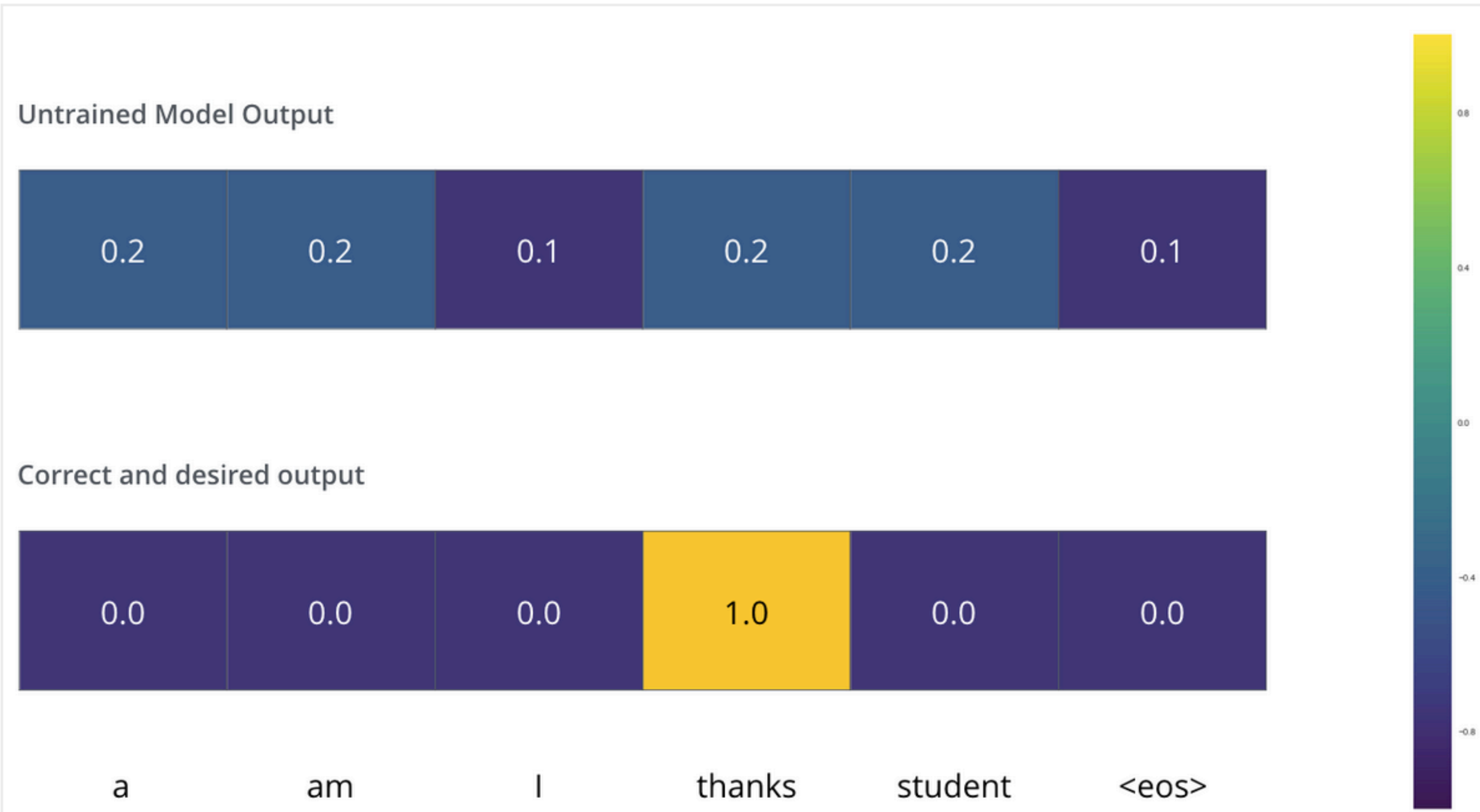
One-hot encoding of the word “am”

0.0	1.0	0.0	0.0	0.0	0.0

[1]Reference: https://jalammar.github.io/illustrated-transformer/

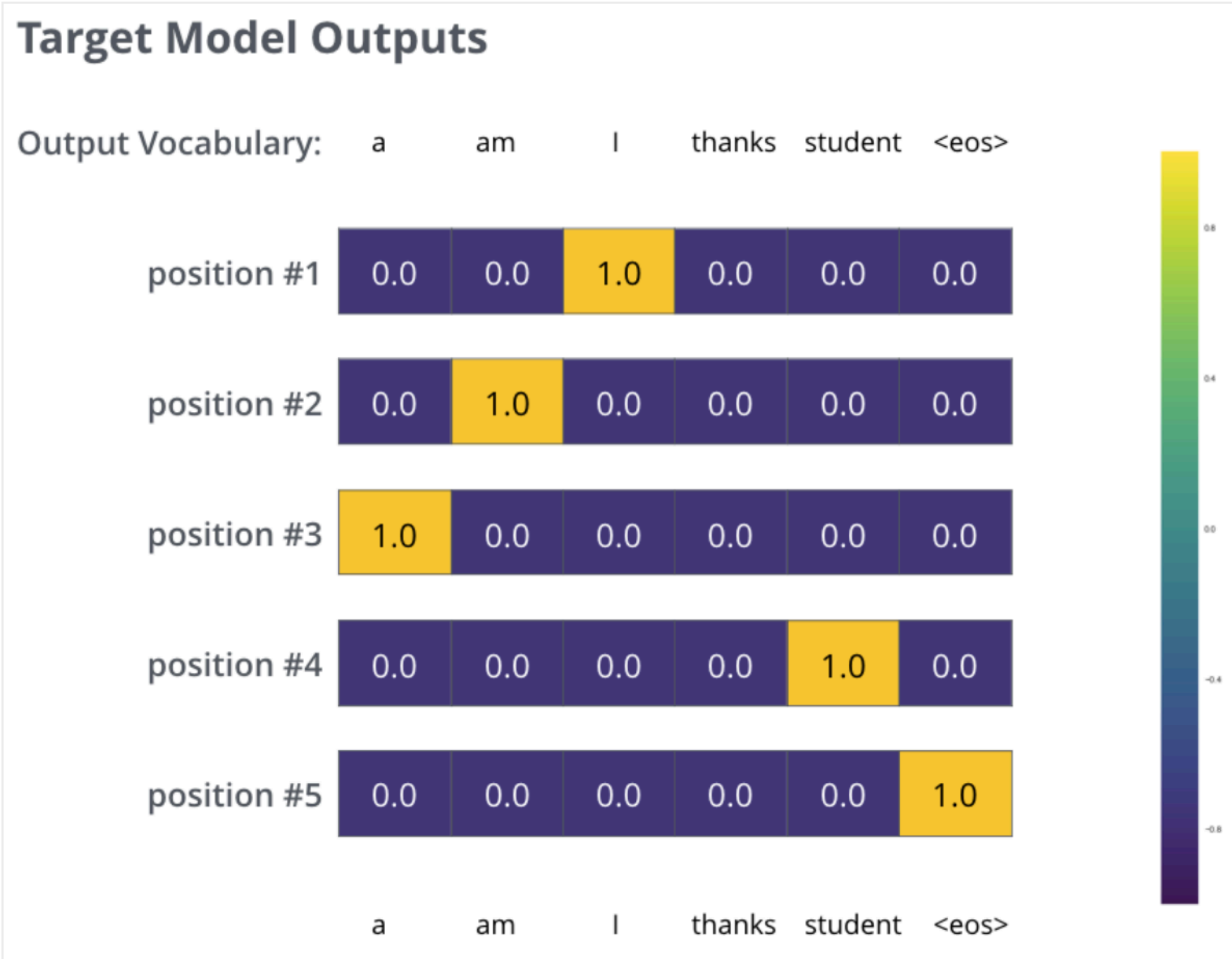
Transformer : Training

Use cross-entropy loss, summed across all outputs.



[1]Reference: https://jalammar.github.io/illustrated-transformer/

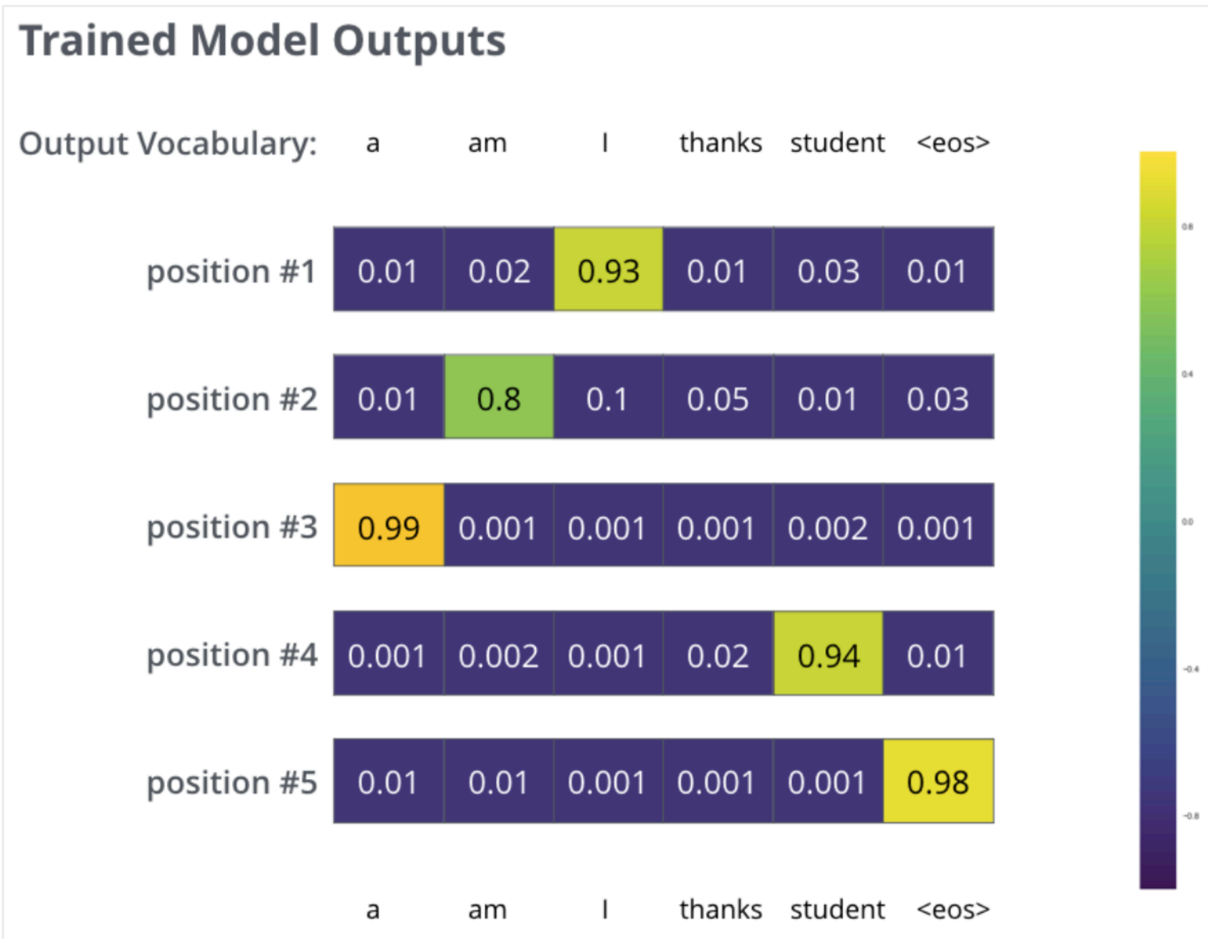
Transformer : Training



[1]Reference: https://jalammar.github.io/illustrated-transformer/

Transformer : Training

After training, output probabilities at each position should reflect the translated sentence's word. Do beam-search decoding, probabilistic decoding or greedy decoding as needed.



[1]Reference: https://jalammar.github.io/illustrated-transformer/

Questions?

Today's Outline

- Recap of Attention in Sequence to Sequence Models
- Transformer Architecture and Self-Attention
- Transfer Learning using a pre-trained NLP model
- BERT and related architectures

Transfer Learning in NLP

Bidirectional Encoder Representations from Transformers

- Key idea: transfer learning
- Similar to how we can use pre-trained models in vision, we can use pre-trained models for language
- We have seen this idea before:
  - Word2Vec
  - Glove
- Since 2018, embeddings generated using transformer based pre-trained models have further improved the state of the art for multiple NLP tasks.
- Lets first see how to use such a model (distillBERT) in a classification task.
  - BERT stands for **B**idirectional **E**ncoder **R**epresentations from **T**ransformers

[1]Reference: https://github.com/google-research/bert and https://arxiv.org/abs/1910.01108

Pre-trained BERT Embeddings

- BERT has been used in versatile products such as Google Search.
  - "… the biggest leap forward in the past five years, and one of the biggest leaps forward in the history of Search."
- For us, we want to use BERT (or distillBERT) in a specific NLP task.
  - Lets pick a movie review classification task

[1]Reference: https://jalammar.github.io/a-visual-guide-to-using-bert-for-the-first-time/ and https://www.blog.google/products/search/search-language-understanding-bert/

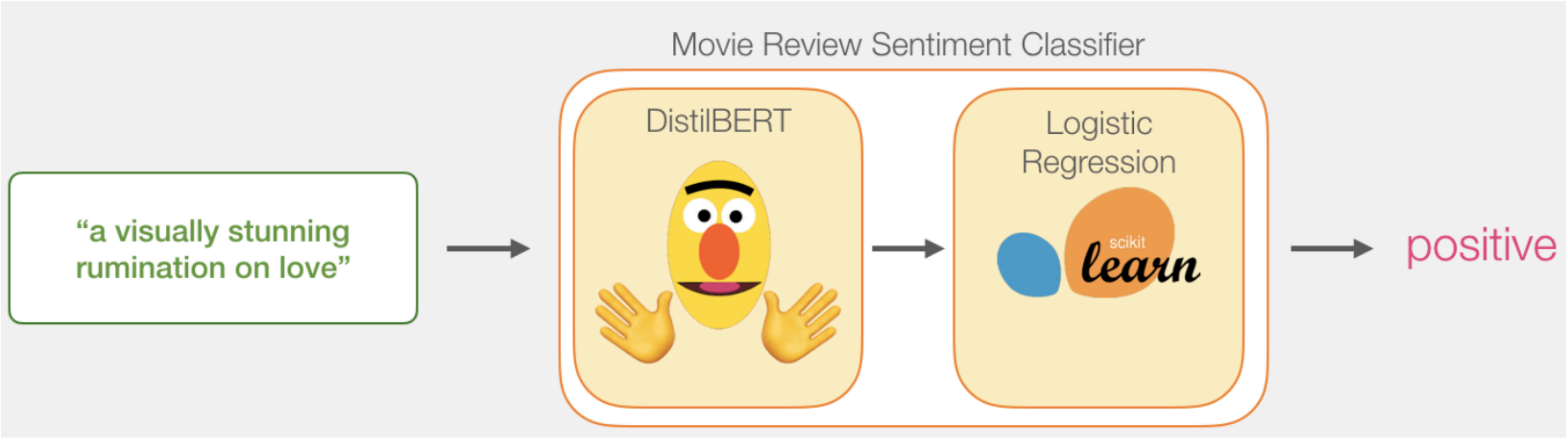
Pre-trained BERT Embeddings

sentence	label
a stirring , funny and finally transporting re imagining of beauty and the beast and 1930s horror films	1
apparently reassembled from the cutting room floor of any given daytime soap	0
they presume their audience won't sit still for a sociology lesson	0
this is a visually stunning rumination on love , memory , history and the war between art and commerce	1
jonathan parker 's bartleby should have been the be all end all of the modern office anomie films	1

[1]Reference: https://jalammar.github.io/a-visual-guide-to-using-bert-for-the-first-time/ and https://colab.research.google.com/github/jalammar/jalammar.github.io/blob/master/notebooks/bert/A_Visual_Notebook_to_Using_BERT_for_the_First_Time.ipynb

Pre-trained BERT Embeddings

The ‘review embedding’ that will be passed on the logistic regression model will be of size 768.

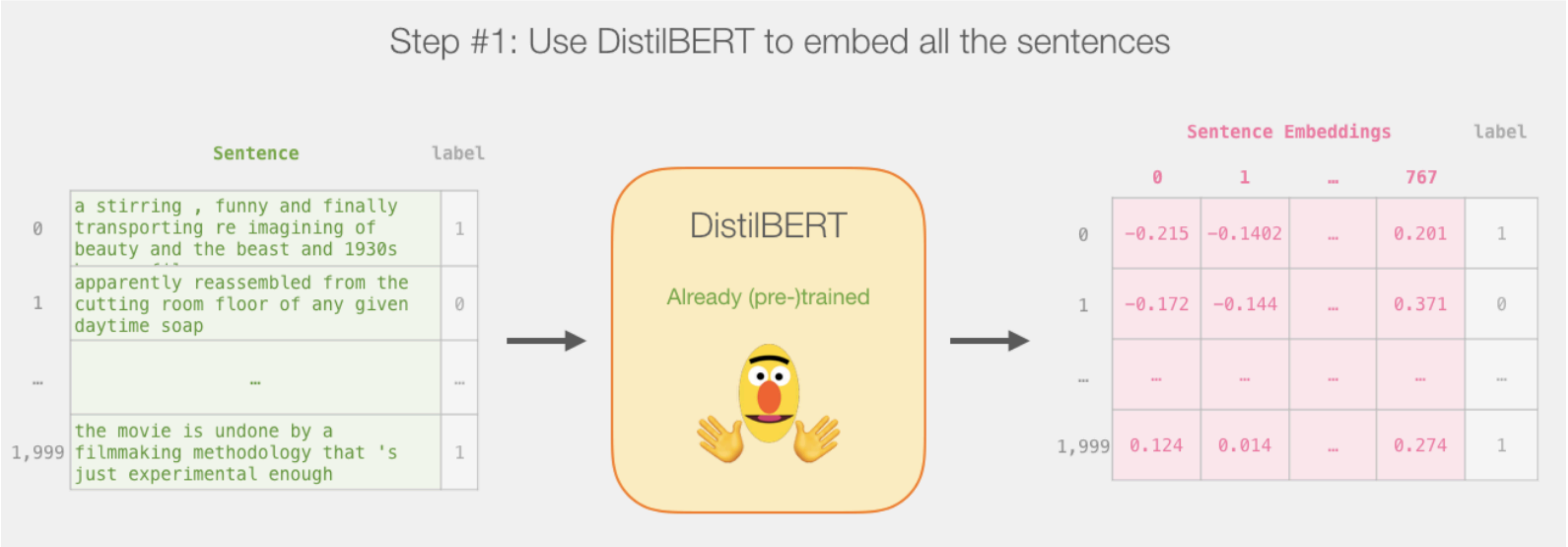


[1]Reference: https://jalammar.github.io/a-visual-guide-to-using-bert-for-the-first-time/

Pre-trained BERT Embeddings

The embedding vector is the output of the first position (associated with the so called [CLS] token) among multiple positions (recall transformer encoder)

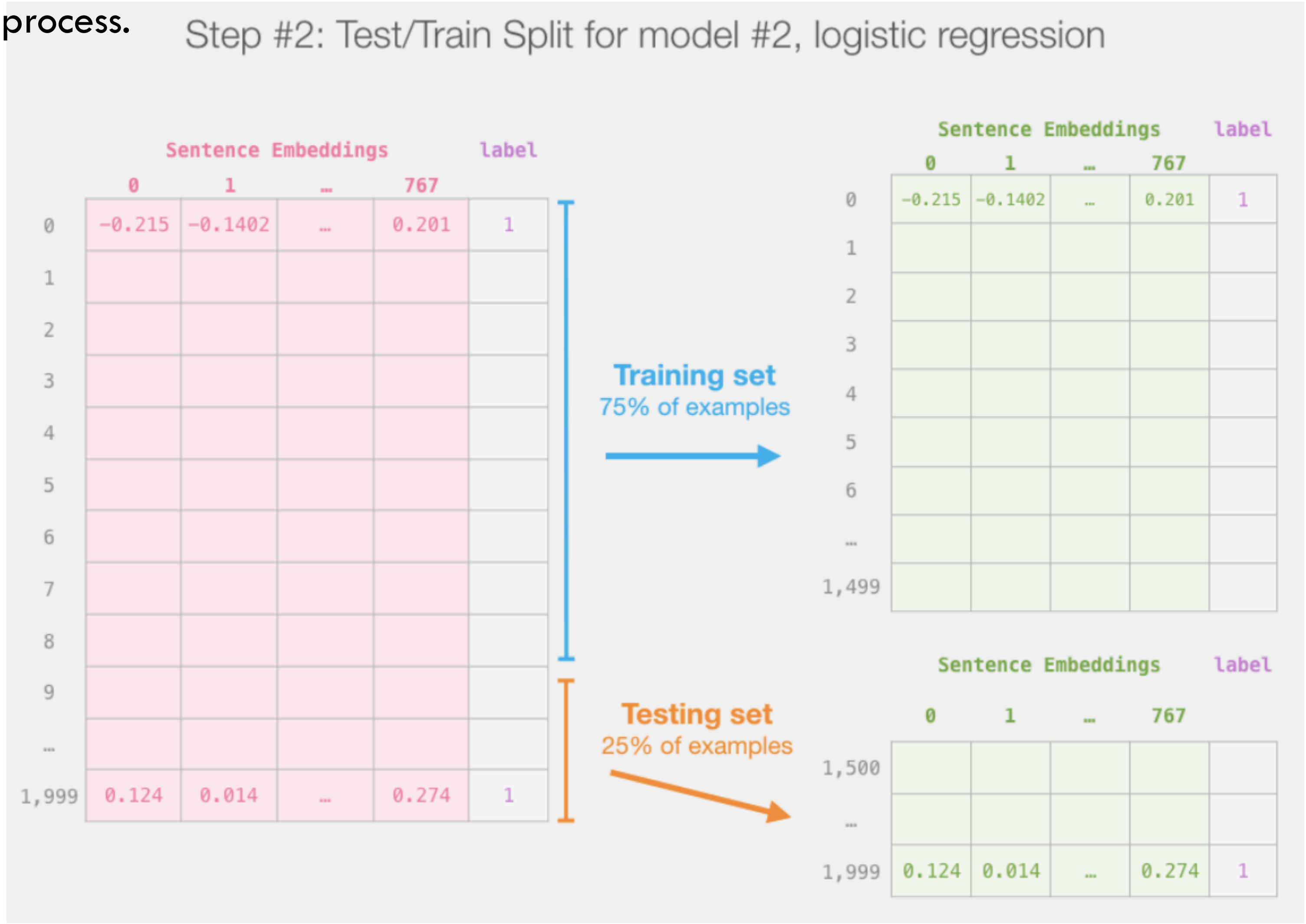
DistilBERT has been pretrained on English using a suitable learning task and a large dataset



[1]Reference: https://jalammar.github.io/a-visual-guide-to-using-bert-for-the-first-time/

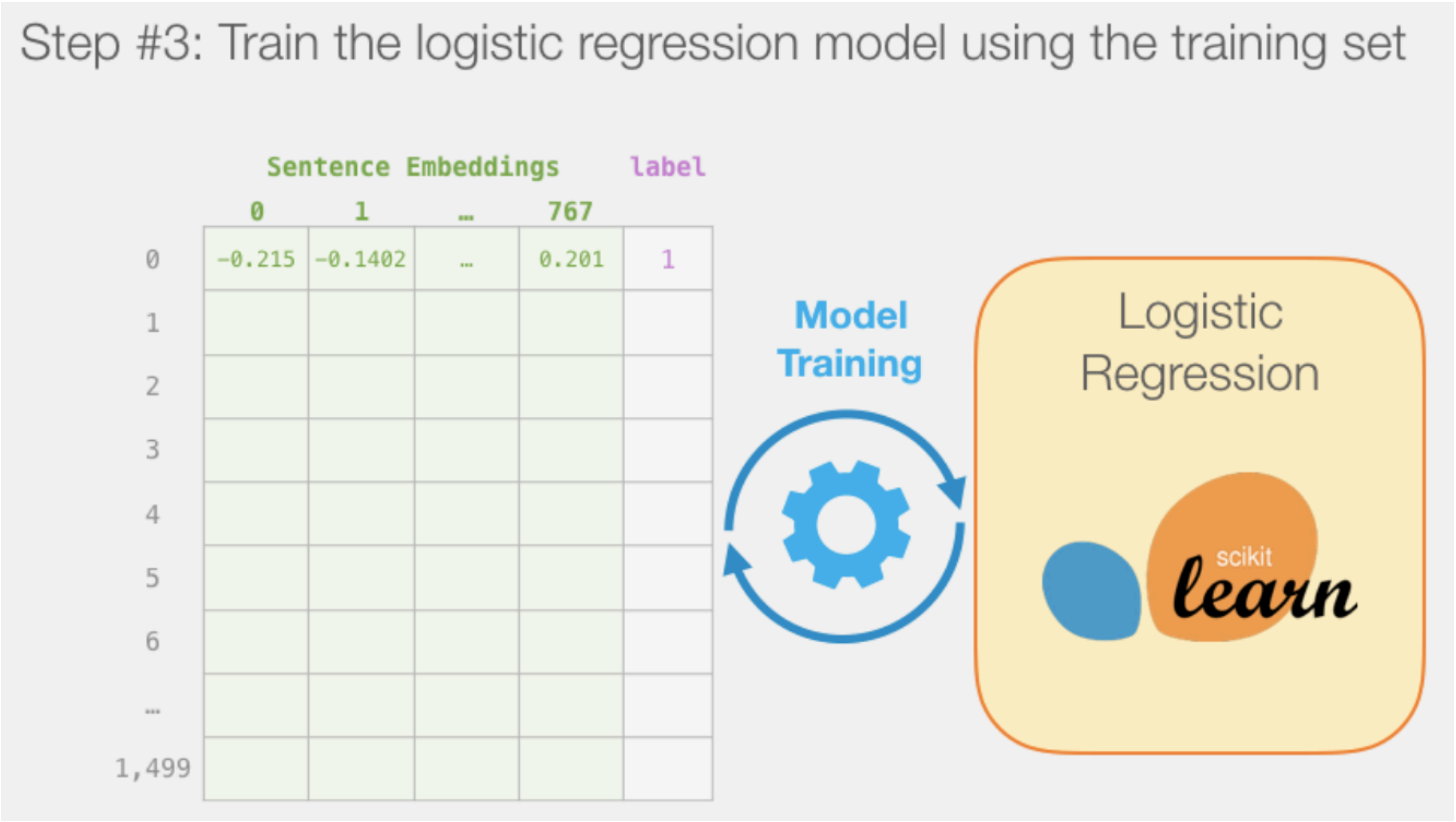
Pre-trained BERT Embeddings

Lets say we have 2000 examples. Once the embeddings are generated, we can just follow the usual ML process.



[1]Reference: https://jalammar.github.io/a-visual-guide-to-using-bert-for-the-first-time/

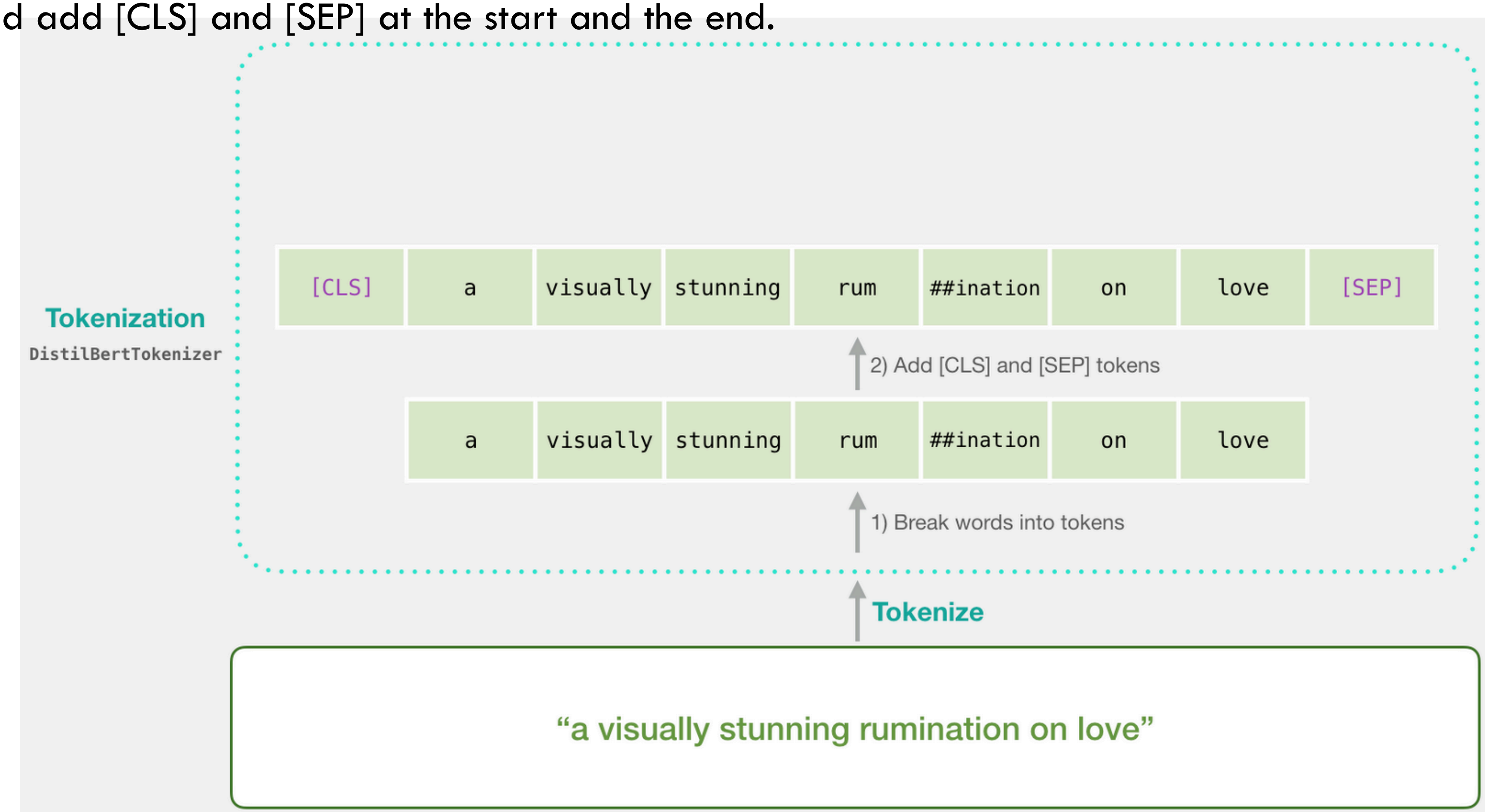
Pre-trained BERT Embeddings



[1]Reference: https://jalammar.github.io/a-visual-guide-to-using-bert-for-the-first-time/

Pre-trained BERT Embeddings

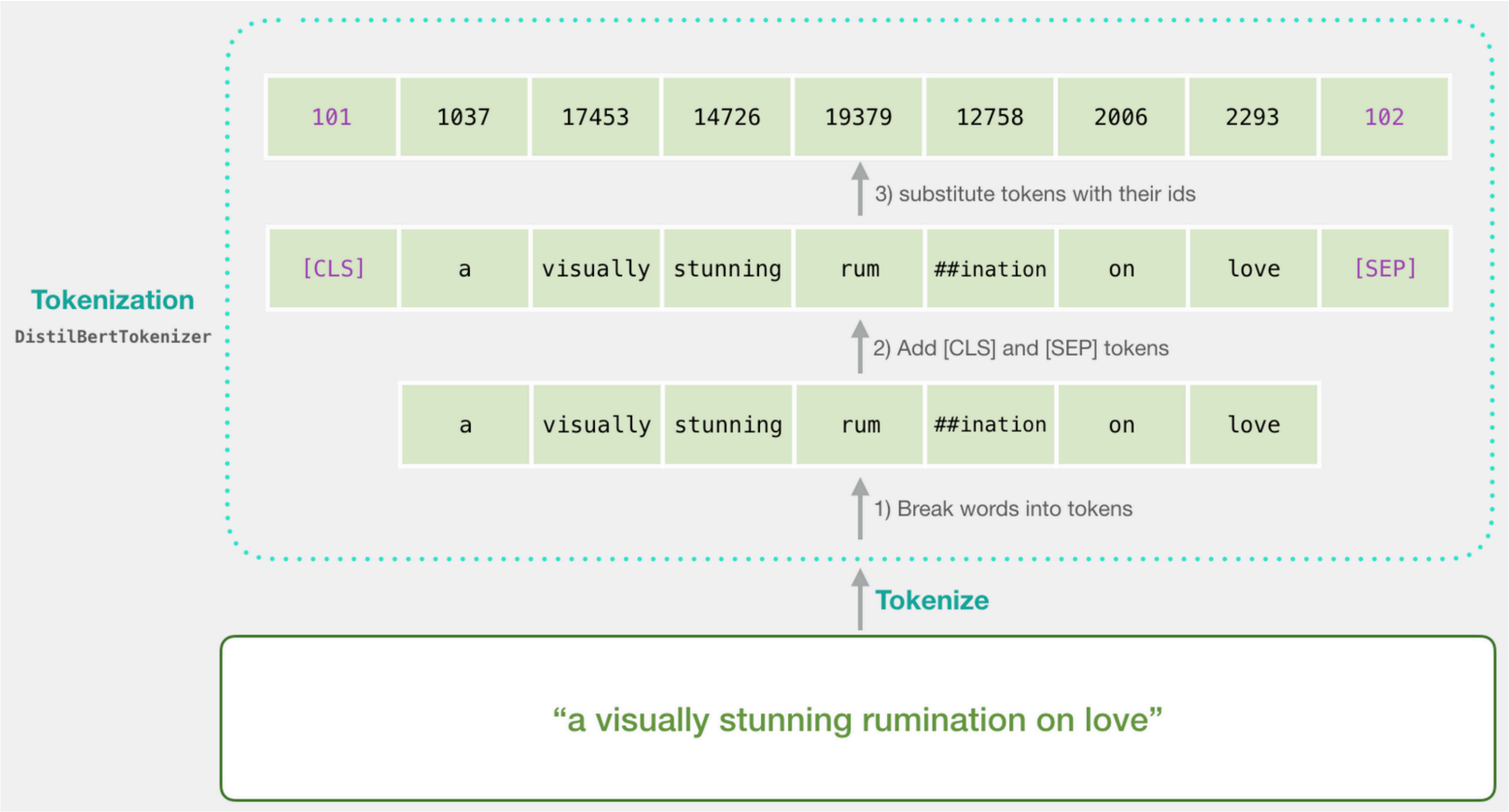
Lets focus on a single prediction with a trained model. We need to 'tokenize' our input sentence and add [CLS] and [SEP] at the start and the end.



[1]Reference: https://jalammar.github.io/a-visual-guide-to-using-bert-for-the-first-time/

Pre-trained BERT Embeddings

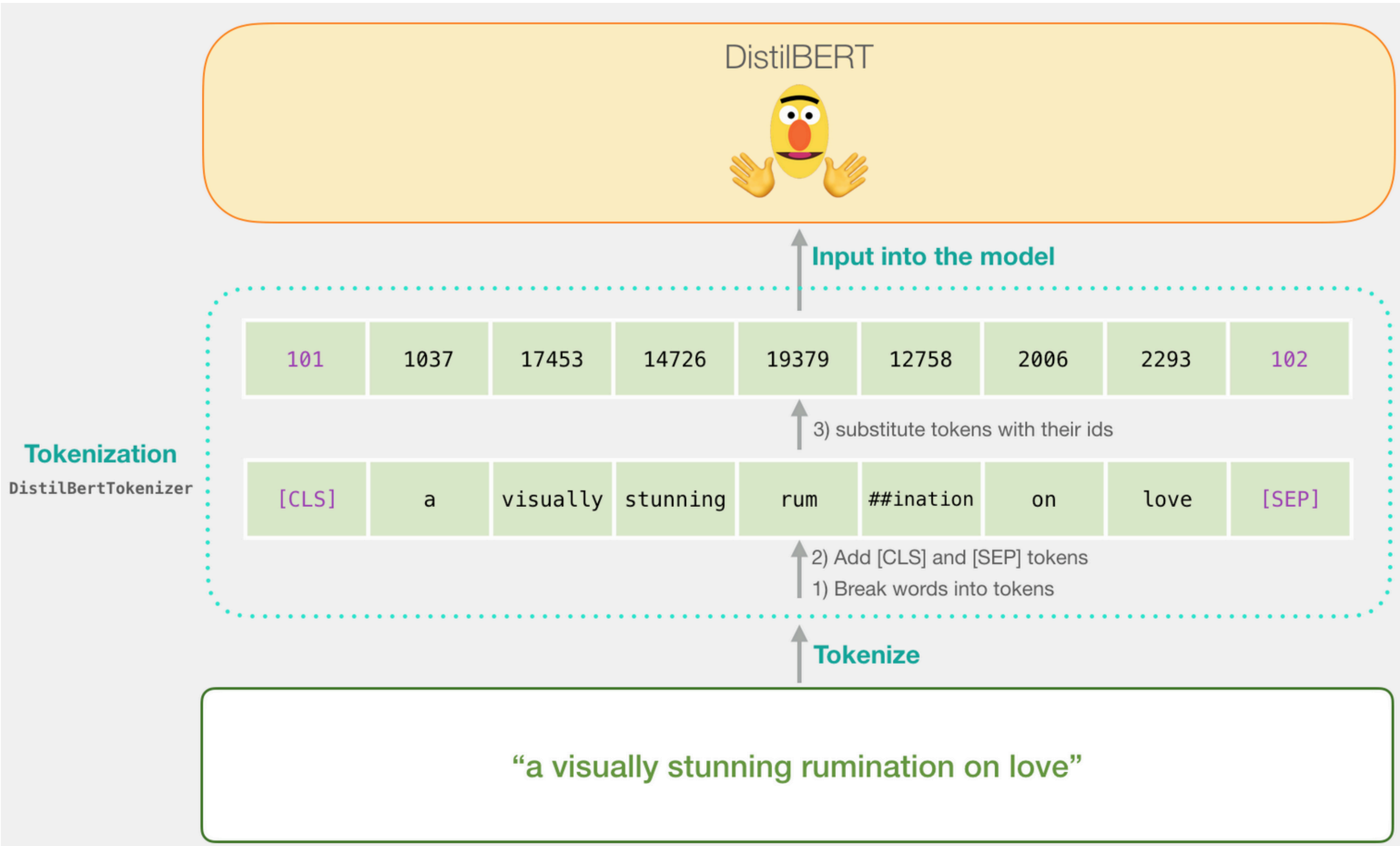
After tokenization, we are left with a sequence of token ids.




```
tokenizer.encode("a visually stunning rumination on love", add_special_tokens=True)
```


[1]Reference: https://jalammar.github.io/a-visual-guide-to-using-bert-for-the-first-time/

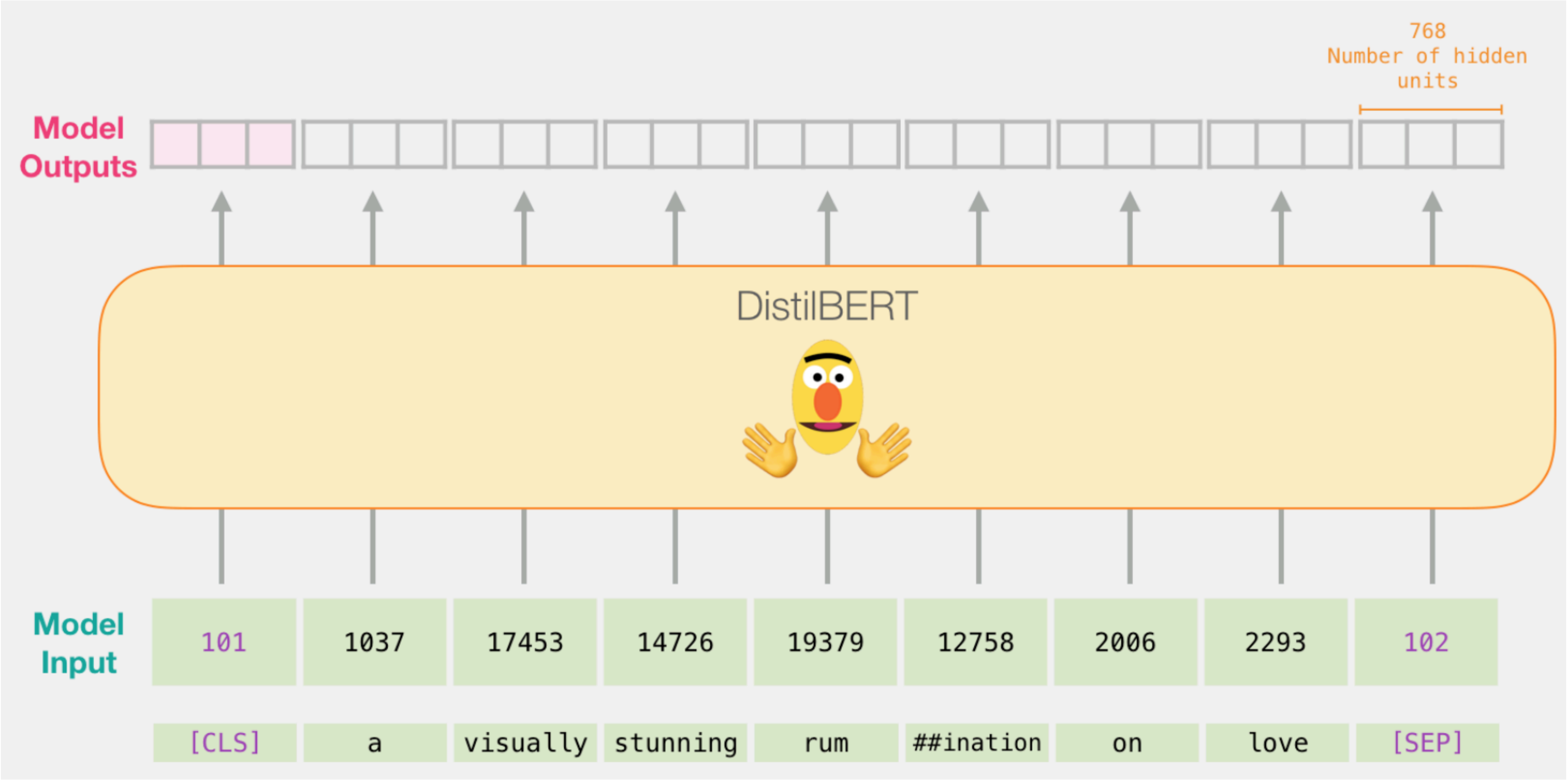
Pre-trained BERT Embeddings



[1] Reference: https://jalammar.github.io/a-visual-guide-to-using-bert-for-the-first-time/

Pre-trained BERT Embeddings

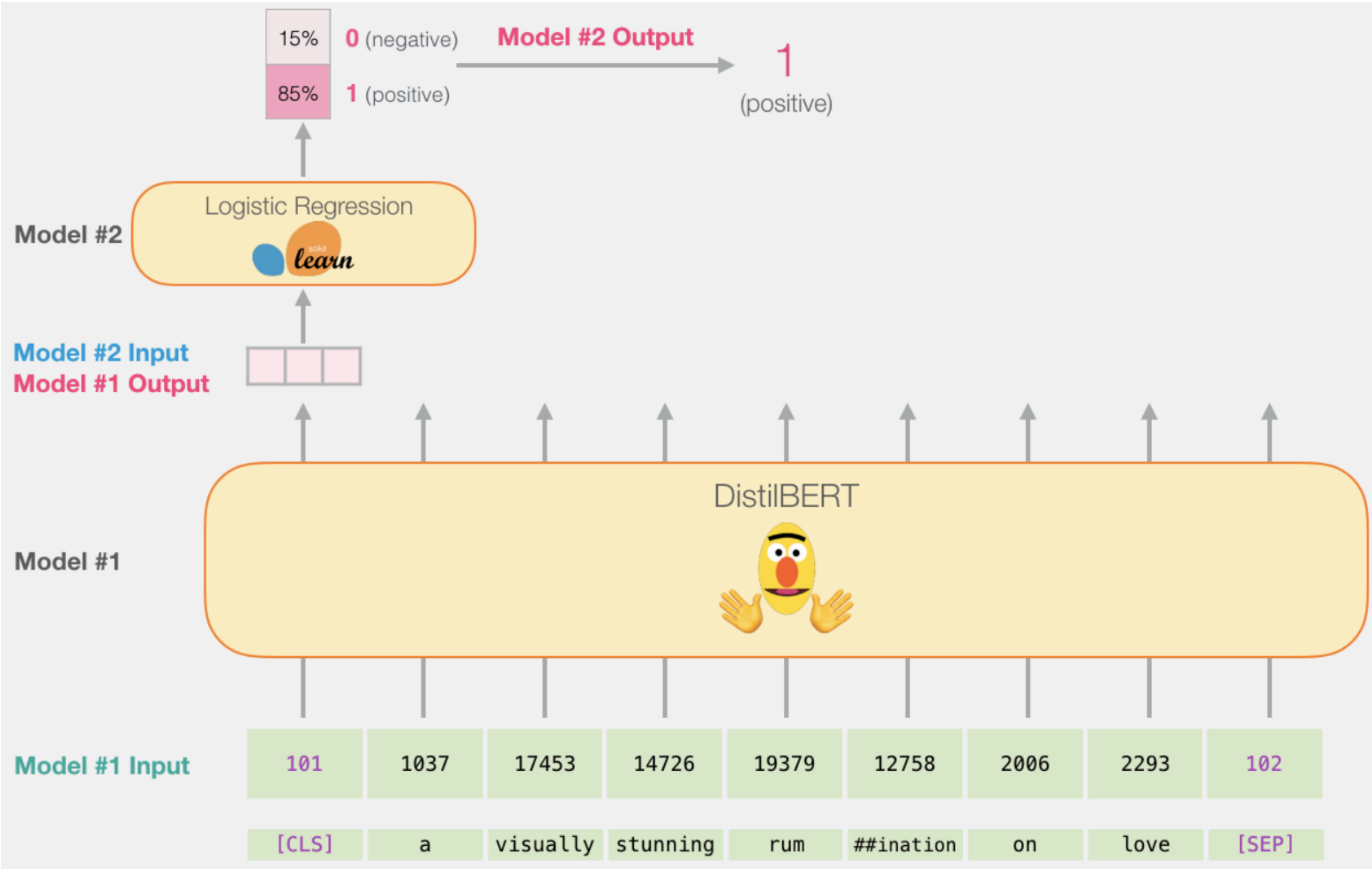
This sequence is passed through DistillBERT (again, think of this as a transformer encoder. We will look at key details later).



[1]Reference: https://jalammar.github.io/a-visual-guide-to-using-bert-for-the-first-time/

Pre-trained BERT Embeddings

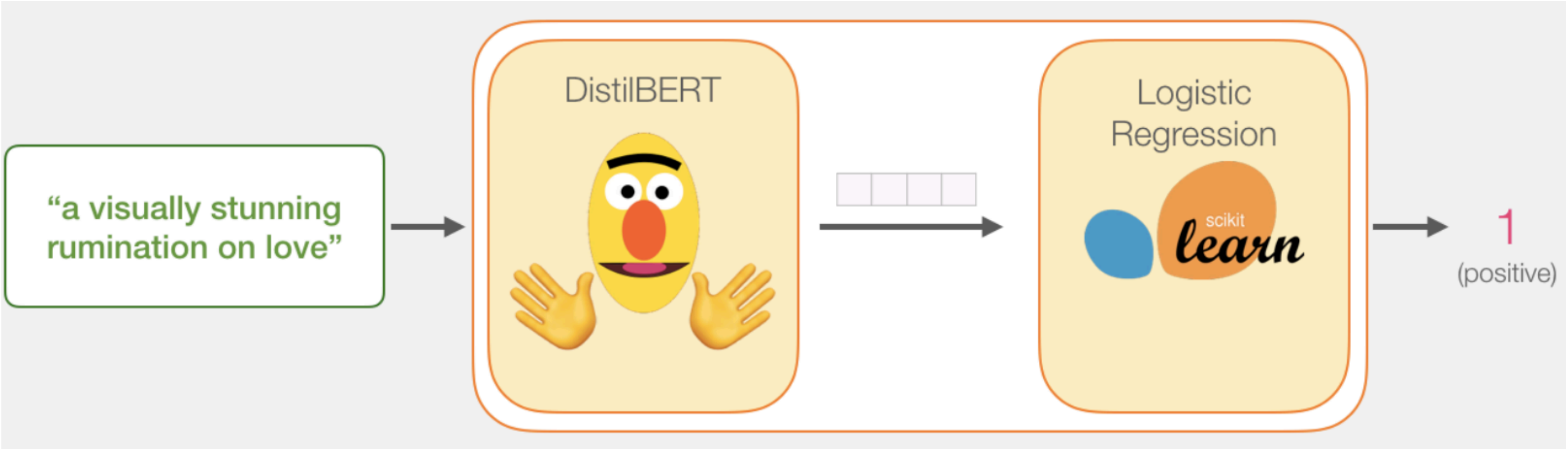
As mentioned before, we only use the vector corresponding to the first dimension.



[1] Reference: https://jalammar.github.io/a-visual-guide-to-using-bert-for-the-first-time/

Pre-trained BERT Embeddings

The rest of the process is standard ML workflow: cross-validated training or training after a train-test split.



[1]Reference: https://jalammar.github.io/a-visual-guide-to-using-bert-for-the-first-time/

Pre-trained BERT Embeddings

The code is as follows:

```
import numpy as np
import pandas as pd
import torch
import transformers as ppb # pytorch transformers
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
```

```
model_class, tokenizer_class, pretrained_weights = (ppb.DistilBertModel, ppb.Distil
BertTokenizer, 'distilbert-base-uncased')

## Want BERT instead of distilBERT? Uncomment the following line:
#model_class, tokenizer_class, pretrained_weights = (ppb.BertModel, ppb.BertTokeniz
er, 'bert-base-uncased')

# Load pretrained model/tokenizer
tokenizer = tokenizer_class.from_pretrained(pretrained_weights)
model = model_class.from_pretrained(pretrained_weights)
```


[1]Reference: https://jalammar.github.io/a-visual-guide-to-using-bert-for-the-first-time/

Pre-trained BERT Embeddings

```
df = pd.read_csv('https://github.com/clairett/pytorch-sentiment-classification/raw/master/data/SST2/train.tsv', delimiter='\t', header=None)
```

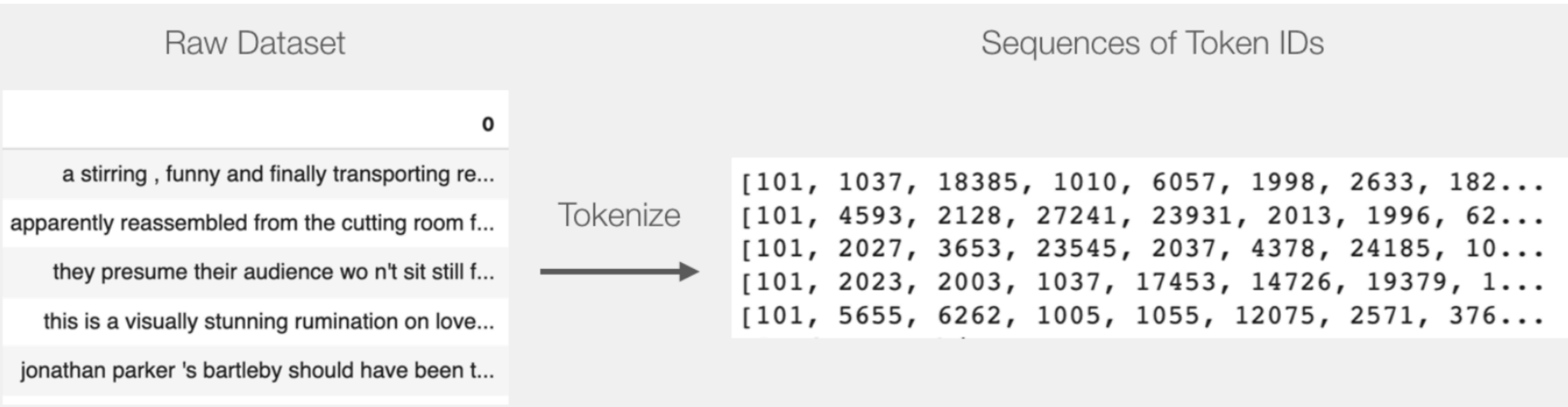
	0	1
0	a stirring , funny and finally transporting re...	1
1	apparently reassembled from the cutting room f...	0
2	they presume their audience wo n't sit still f...	0
3	this is a visually stunning rumination on love...	1
4	jonathan parker 's bartleby should have been t...	1

[1]Reference: https://jalammar.github.io/a-visual-guide-to-using-bert-for-the-first-time/ and https://github.com/clairett/pytorch-sentiment-classification/

Pre-trained BERT Embeddings

We are applying tokenization over all training data.

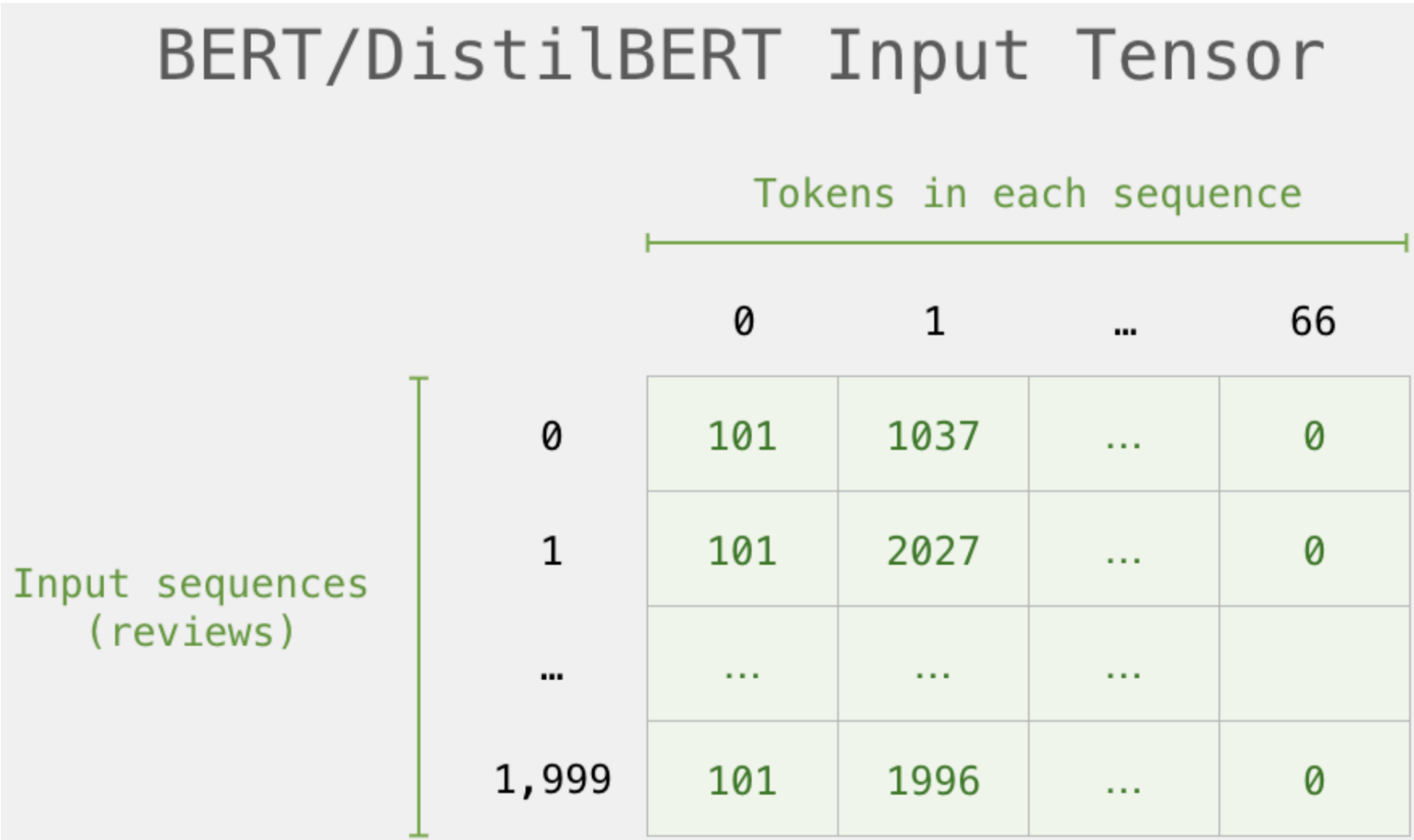
```
tokenized = df[0].apply((lambda x: tokenizer.encode(x, add_special_tokens=True)))
```



[1] Reference: https://jalammar.github.io/a-visual-guide-to-using-bert-for-the-first-time/

Pre-trained BERT Embeddings

We will pad short sentences with token 0. The largest sentence length is 66.

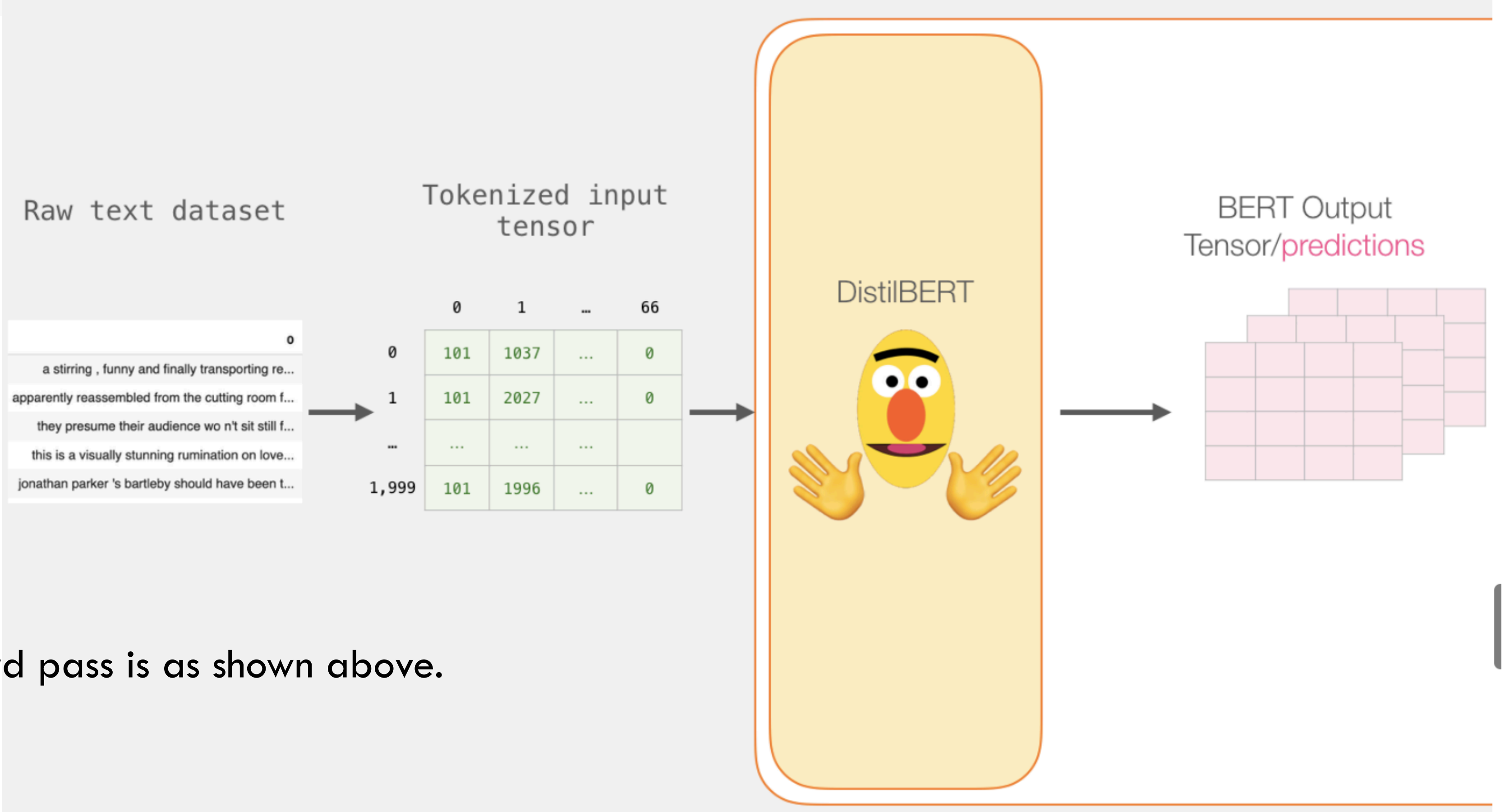


[1]Reference: https://jalammar.github.io/a-visual-guide-to-using-bert-for-the-first-time/

Pre-trained BERT Embeddings

```
input_ids = torch.tensor(np.array(padded))

with torch.no_grad():
    last_hidden_states = model(input_ids)
```

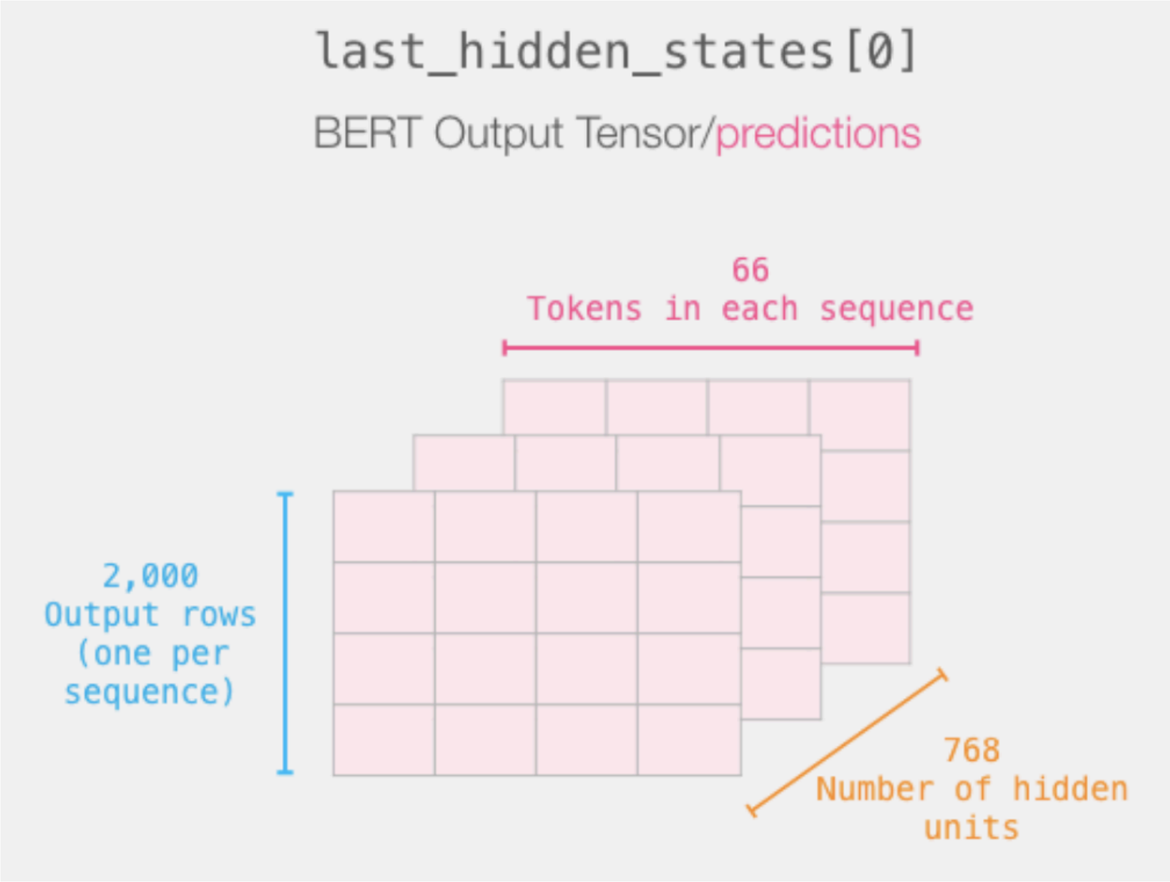


The forward pass is as shown above.

[1]Reference: https://jalammar.github.io/a-visual-guide-to-using-bert-for-the-first-time/

Pre-trained BERT Embeddings

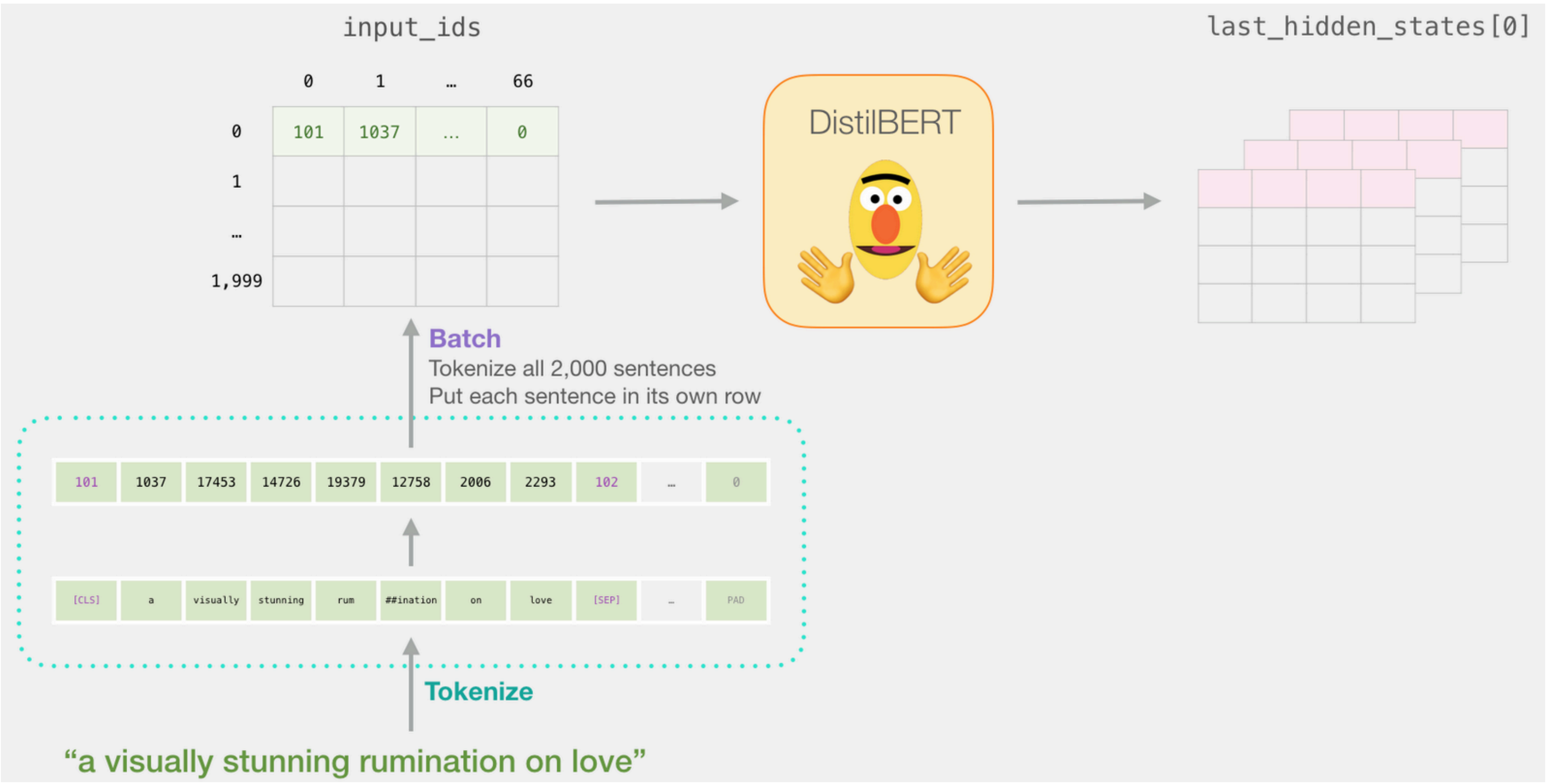
The output variable has a shape (#examples, max no of tokens, number of hidden units)
So, 2000 * 66 * 768.



[1]Reference: https://jalammar.github.io/a-visual-guide-to-using-bert-for-the-first-time/

Pre-trained BERT Embeddings

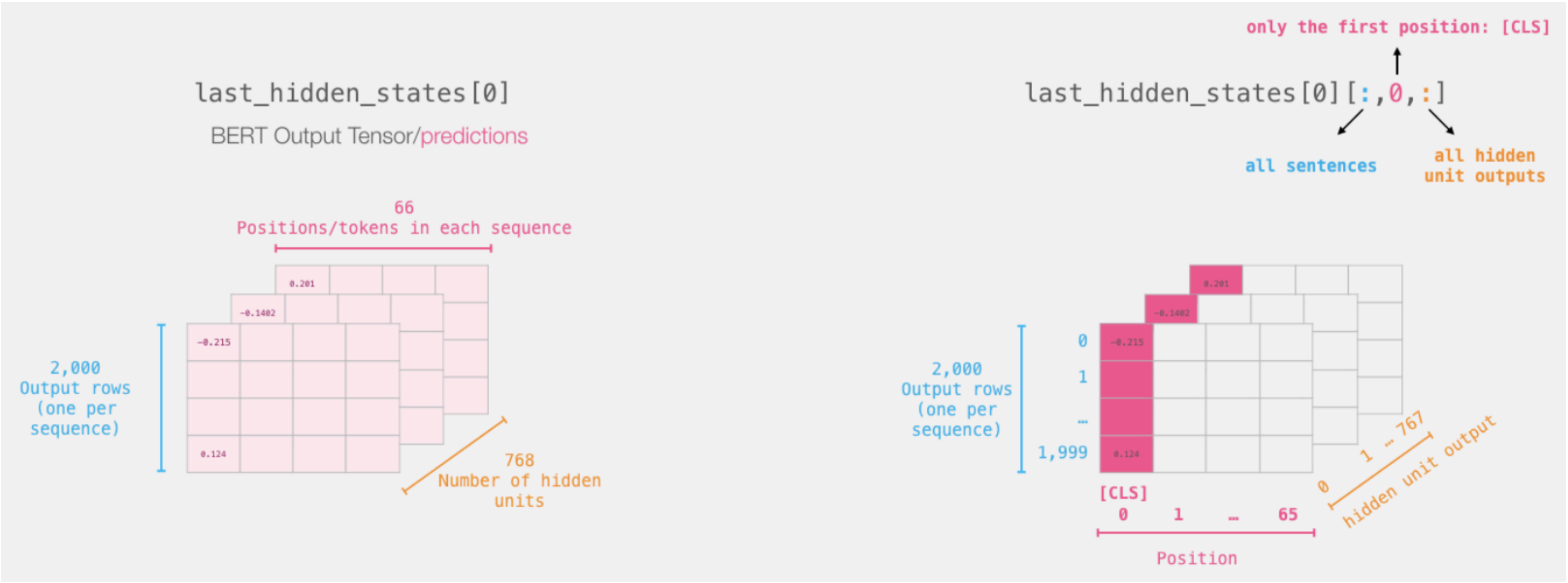
Here is an illustration for a single example (the first one).



[1]Reference: https://jalammar.github.io/a-visual-guide-to-using-bert-for-the-first-time/

Pre-trained BERT Embeddings

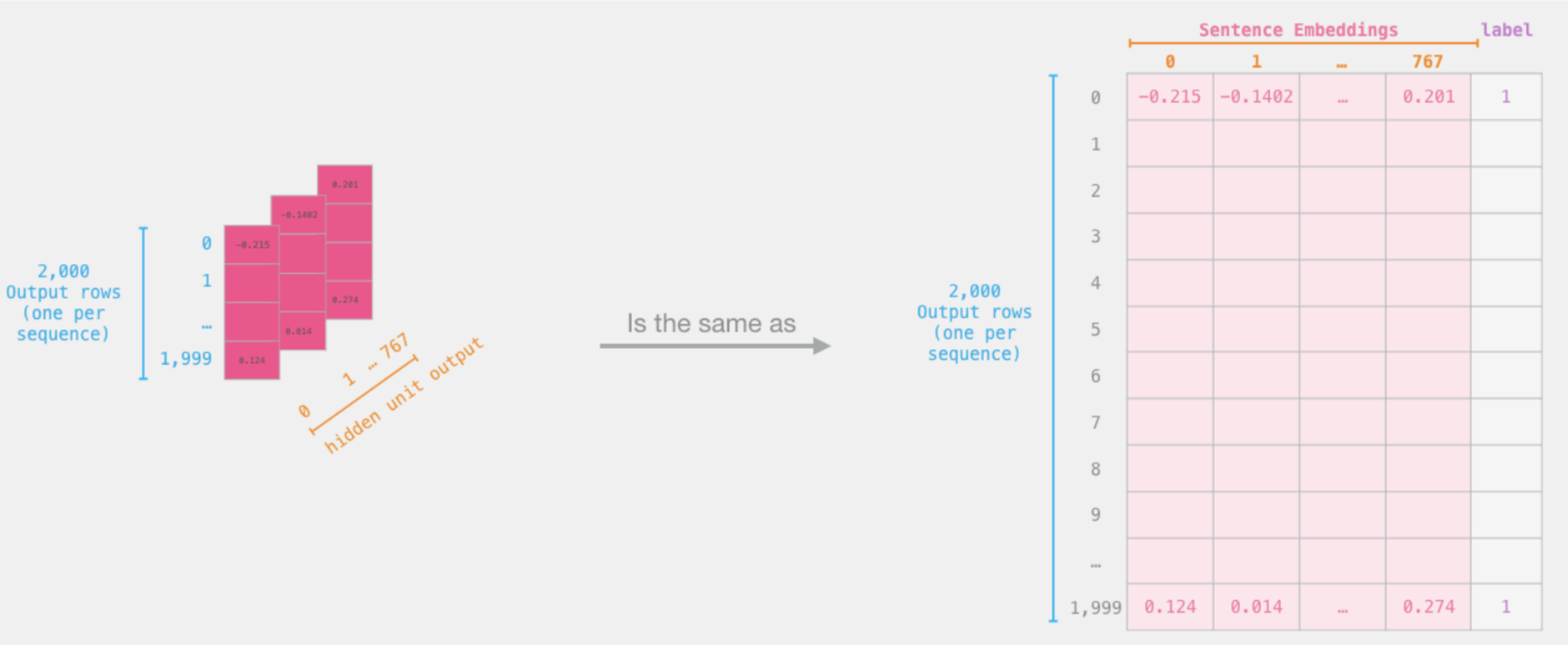
We only need the output vector corresponding to the first position/token. That part of the output tensor is highlighted.



[1]Reference: https://jalammar.github.io/a-visual-guide-to-using-bert-for-the-first-time/

Pre-trained BERT Embeddings

```
# Slice the output for the first position for all the sequences, take all hidden unit outputs
features = last_hidden_states[0][:,0,:].numpy()
```

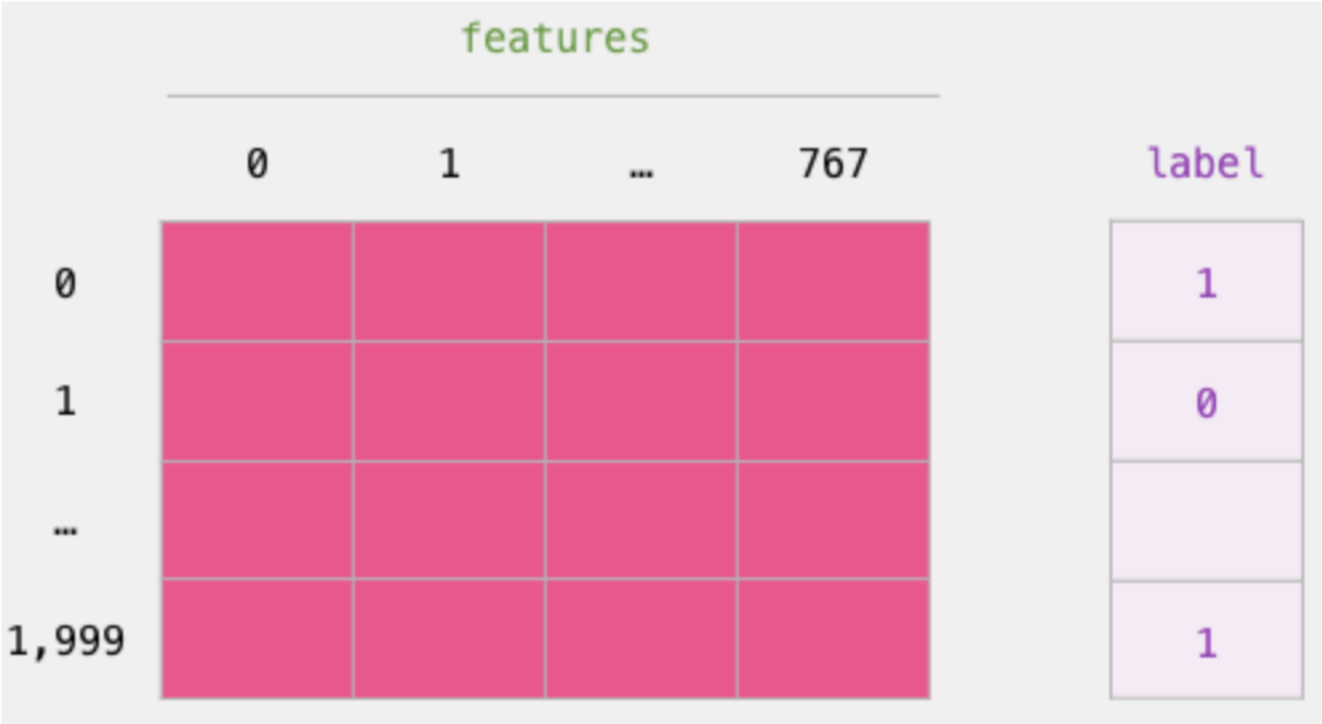


Reference[1]: https://jalammar.github.io/a-visual-guide-to-using-bert-for-the-first-time/

Pre-trained BERT Embeddings

See https://huggingface.co/transformers/examples.html for more example that not only use pre-trained models as feature extractors, but also fine-tune them.

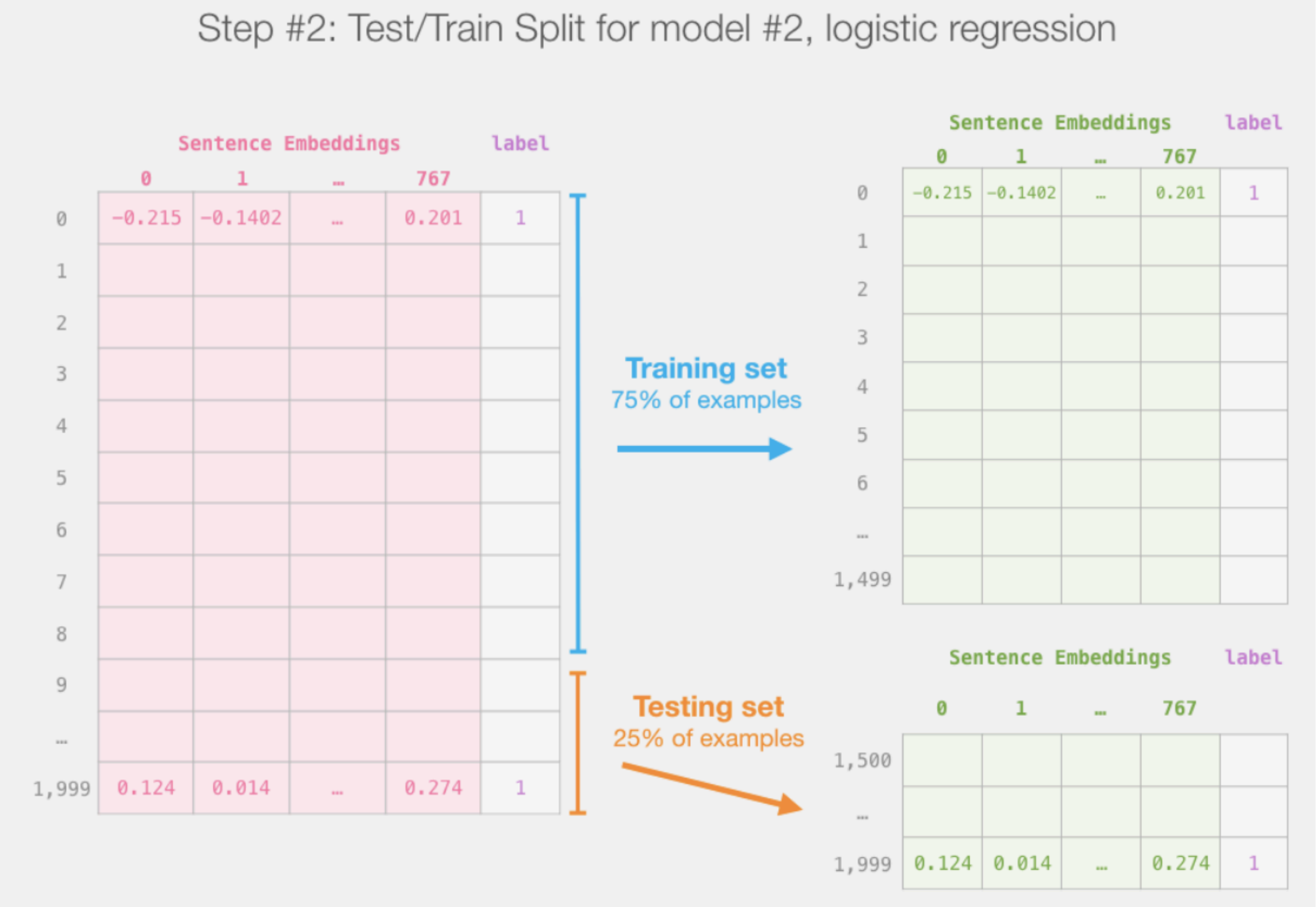
Its standard ML from this point out for our running example (movie review classification).



[1]Reference: https://jalammar.github.io/a-visual-guide-to-using-bert-for-the-first-time/

Pre-trained BERT Embeddings

```
labels = df[1]
train_features, test_features, train_labels, test_labels = train_test_split(features, labels)
```



[1]Reference: https://jalammar.github.io/a-visual-guide-to-using-bert-for-the-first-time/

Questions?

Today's Outline

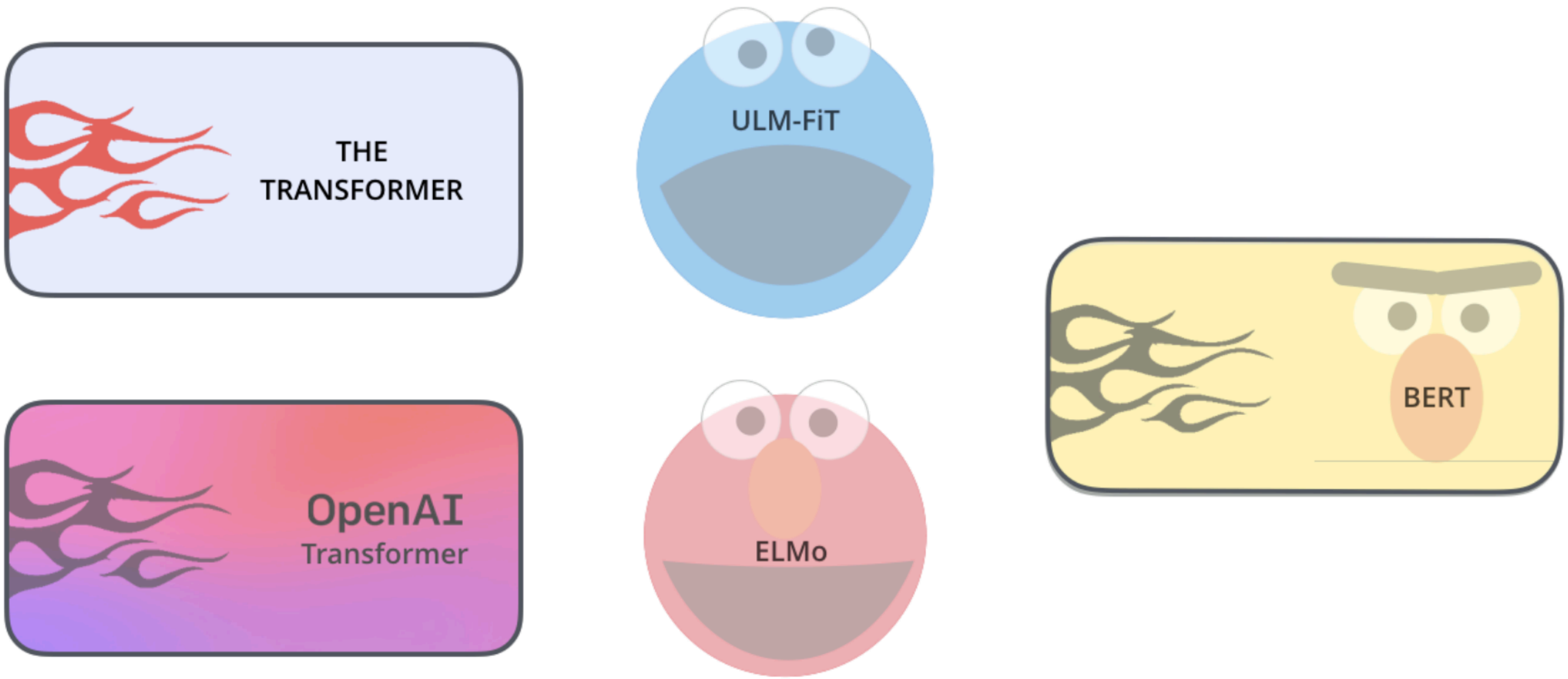
- Recap of Attention in Sequence to Sequence Models
- Transformer Architecture and Self-Attention
- Transfer Learning using a pre-trained NLP model
- BERT and related architectures

BERT and Friends

BERT

BERT, or Bidirectional Encoder Representations from Transformers, is a new method of pre-training language representations which obtains state-of-the-art results on a wide array of Natural Language Processing (NLP) tasks.

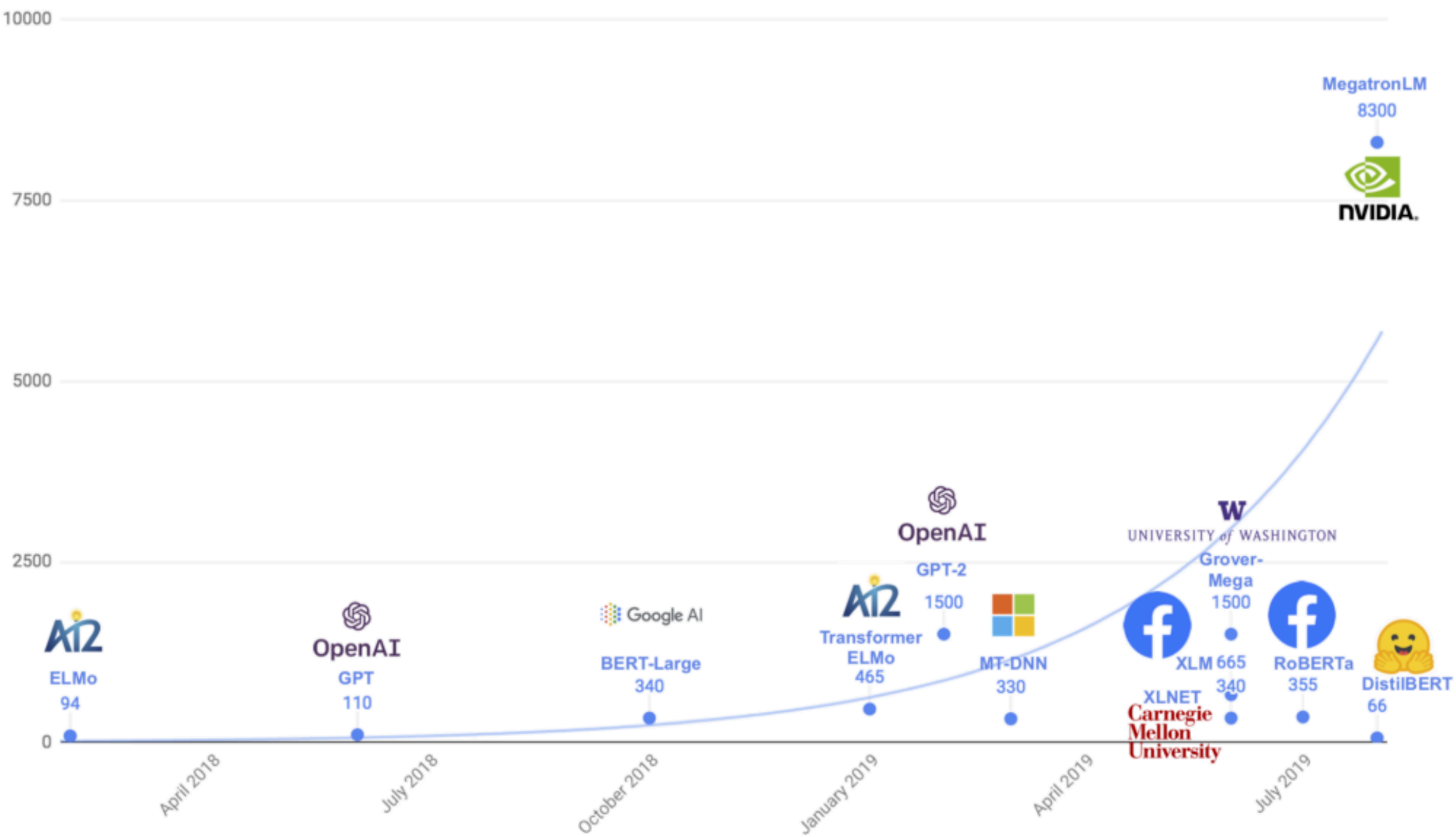
There are many models out there.



[1]Reference: https://jalammar.github.io/illustrated-bert/ and https://arxiv.org/abs/1810.04805

BERT

- GPT-3



[1]Reference: https://medium.com/huggingface/distilbert-8cf3380435b5 and https://en.wikipedia.org/wiki/GPT-3

BERT

Here are the ExactMatch (EM) and F1 scores evaluated on the test set of SQuAD v1.1.

Rank	Model	EM	F1
	Human Performance *Stanford University* (Rajpurkar et al. '16)	82.304	91.221
12 Oct 05, 2018	BERT (single model) *Google AI Language* https://arxiv.org/abs/1810.04805	85.083	91.835

- **S**tanford **Q**uestion **A**nswering **D**ataset (SQuAD) is a reading comprehension dataset, consisting of questions posed by crowdworkers on a set of Wikipedia articles, where the answer to every question is a segment of text, or *span*, from the corresponding reading passage, or the question might be unanswerable.

[1]Reference: https://rajpurkar.github.io/SQuAD-explorer/

BERT

SQuAD v1.1 Leaderboard (Oct 8th 2018)	Test EM	Test F1
1st Place Ensemble - BERT	**87.4**	**93.2**
2nd Place Ensemble - nlnet	86.0	91.7
1st Place Single Model - BERT	**85.1**	**91.8**
2nd Place Single Model - nlnet	83.5	90.1

And several natural language inference tasks:

System	MultiNLI	Question NLI	SWAG
BERT	**86.7**	**91.1**	**86.3**
OpenAI GPT (Prev. SOTA)	82.2	88.1	75.0

Plus many other tasks.

Moreover, these results were all obtained with almost no task-specific neural network architecture design.

[1] Reference: https://github.com/google-research/bert

BERT

- A general-purpose "language understanding" model on a large text corpus (like Wikipedia)
- Use the model for downstream NLP tasks that we care about (like question answering)

- BERT outperforms previous methods because it is the first *unsupervised*, *deeply bidirectional* system for pre-training NLP
  - Unsupervised: trained only on plain text (no metadata)

[1]Reference: https://github.com/google-research/bert

BERT

- Pre-trained representations can also either be *context-free* or *contextual*, and contextual representations can further be *unidirectional* or *bidirectional*.
- Context-free models such as [word2vec]() or [GloVe]() generate a single "word embedding" representation
- Contextual models instead generate a representation of each word that is based on the other words in the sentence.
- Bidirectionality: BERT represents words using both its left and right context

[1] Reference: https://github.com/google-research/bert

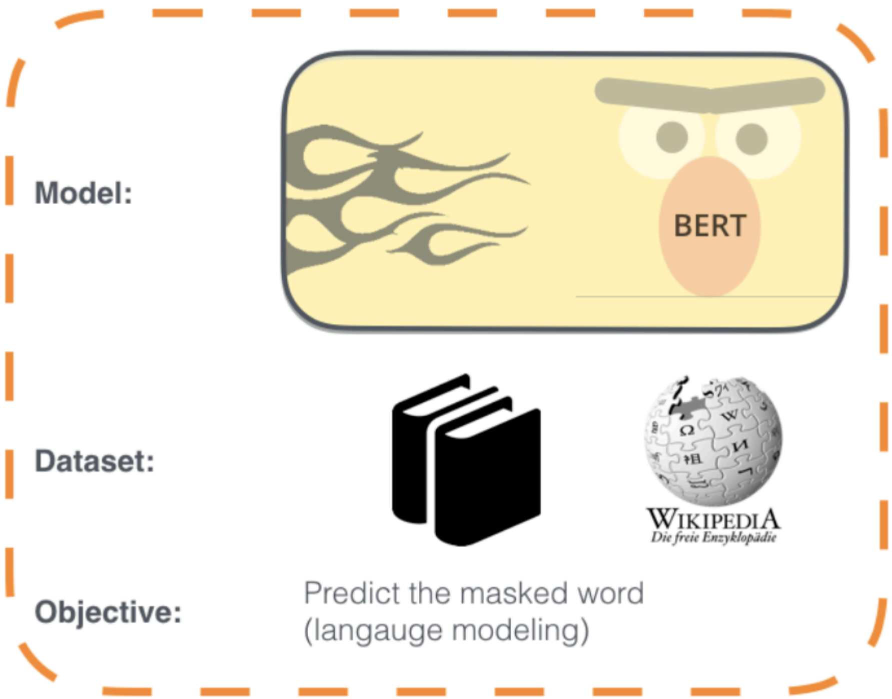
BERT

Two step process

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.

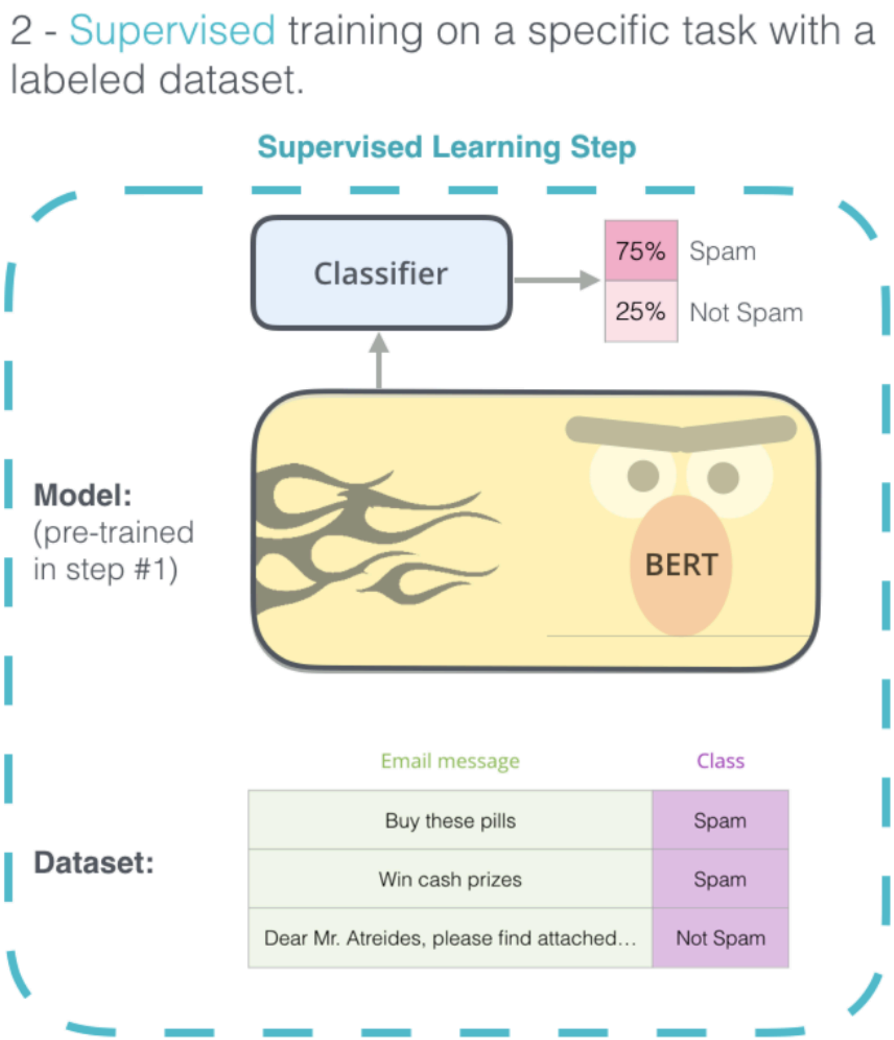
Semi-supervised Learning Step



[1]Reference: https://jalammar.github.io/illustrated-bert/

BERT

The second step is problem specific

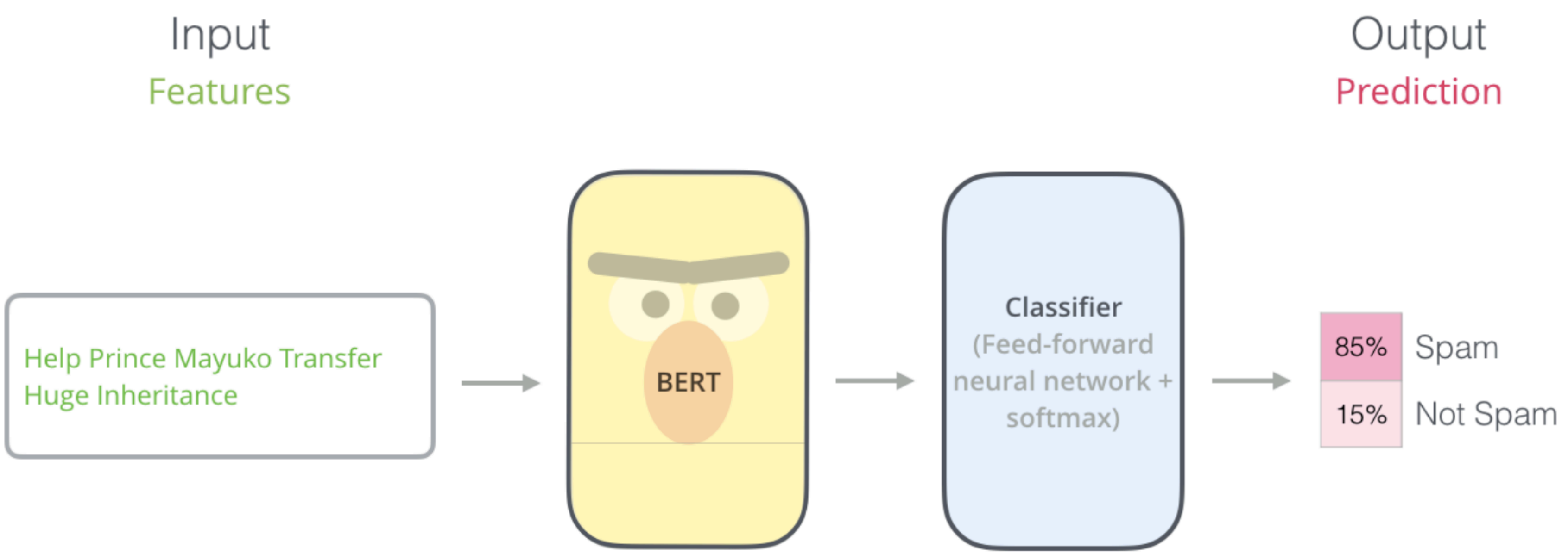


[1]Reference: https://jalammar.github.io/illustrated-bert/

BERT

For instance, BERT can be used for classification (we saw this in detail earlier)

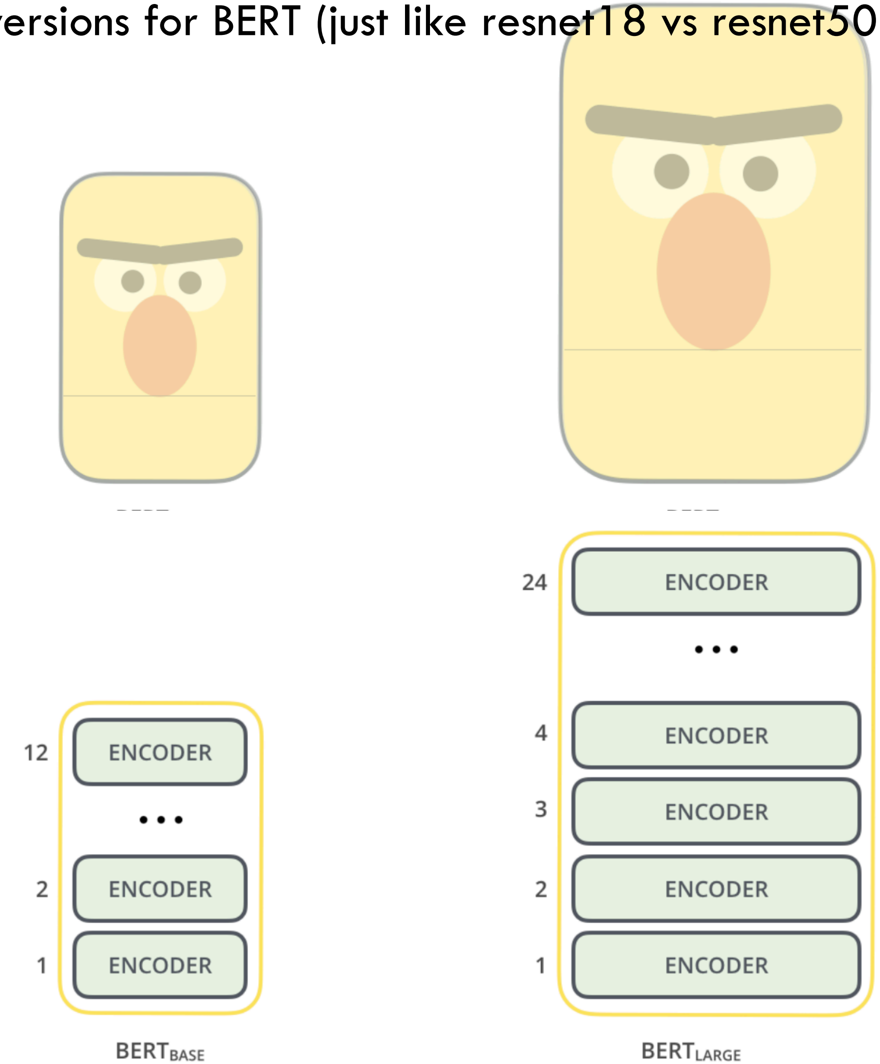
If the BERT parameters are also changed, this would be considered fine-tuning (we saw this for vision)



[1]Reference: https://jalammar.github.io/illustrated-bert/

BERT

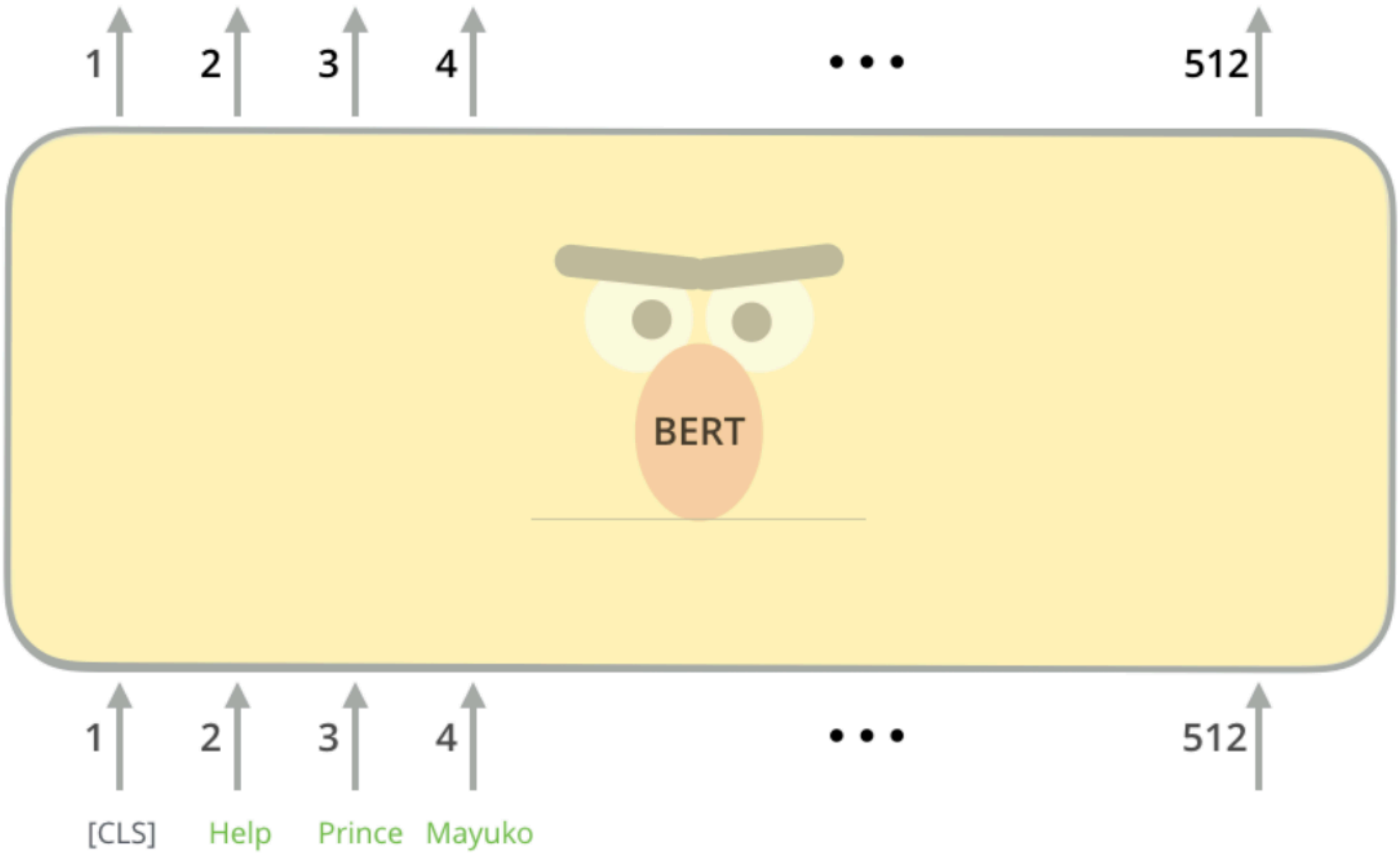
There are two pre-trained versions for BERT (just like resnet18 vs resnet50 or vgg16 vs vgg19)



[1]Reference: https://jalammar.github.io/illustrated-bert/

BERT

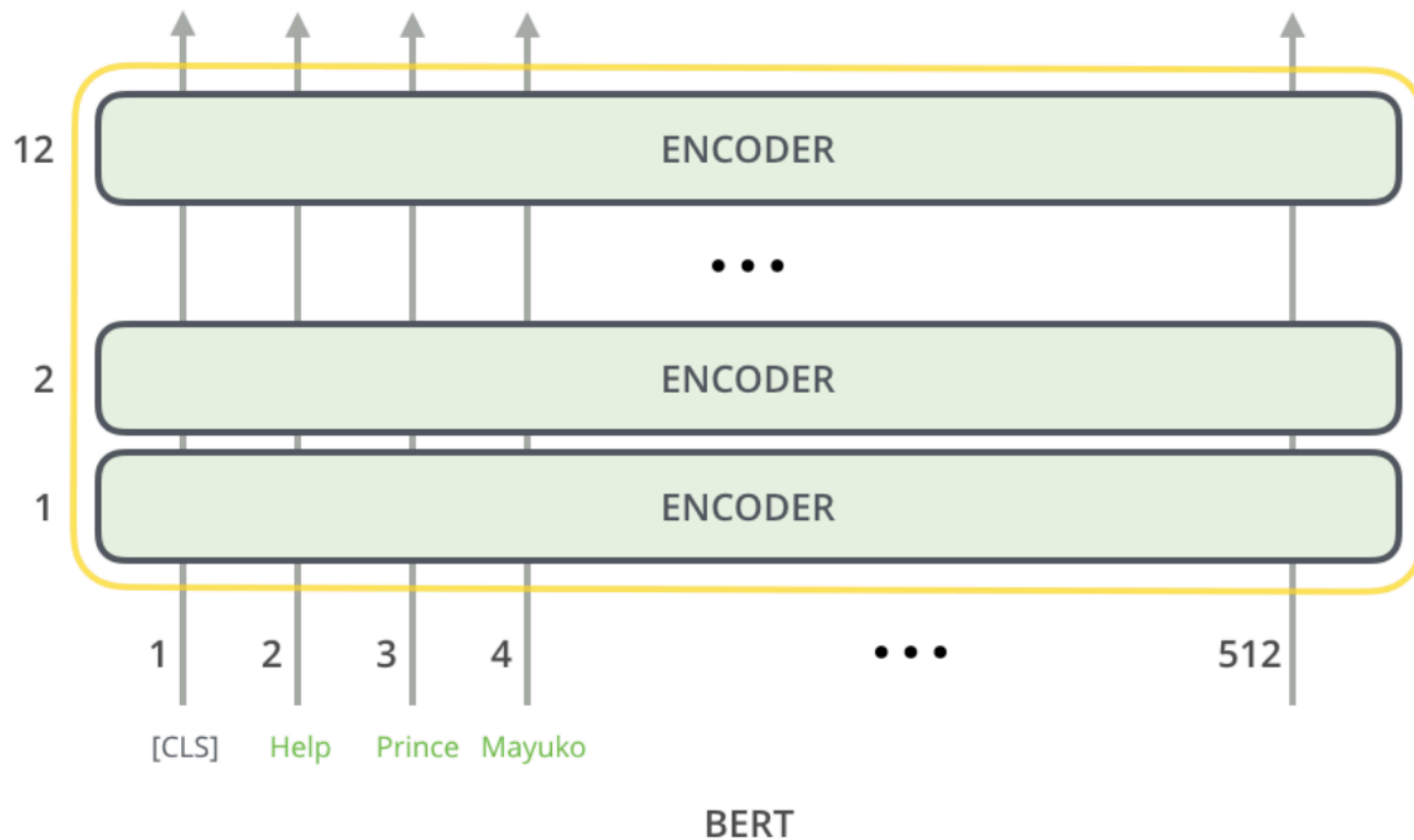
The encoder units/layers (also called transformer blocks) is 12 or 24. FF networks have 768 or 1024 hidden units. The number of attention heads is 12 or 16. (vs 6 units, 512 units, 8 heads before)



[1]Reference: https://jalammar.github.io/illustrated-bert/

BERT

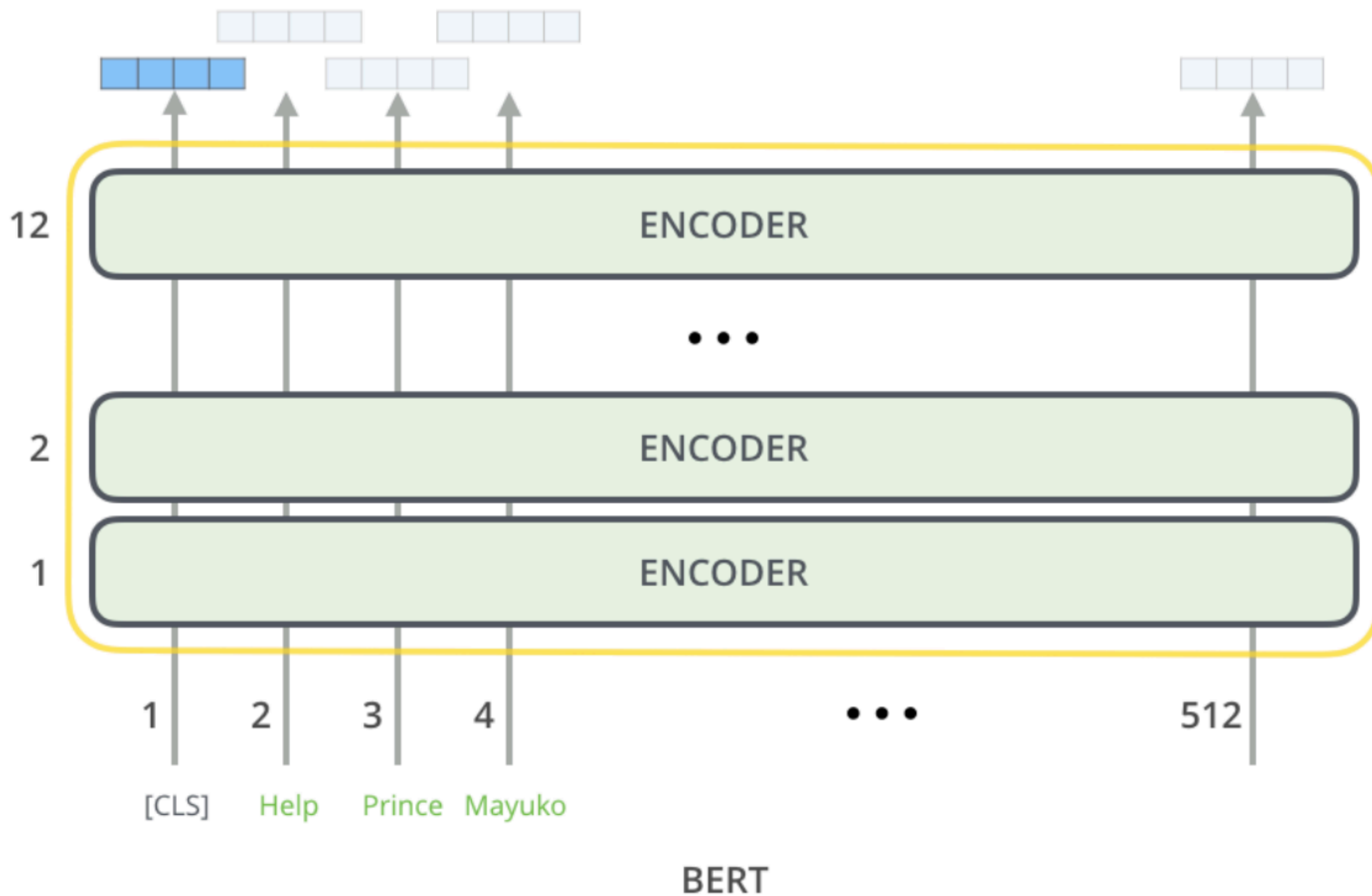
First input is a special symbol (cls means classification). Architecture same as Transformer so far.



[1]Reference: https://jalammar.github.io/illustrated-bert/

BERT

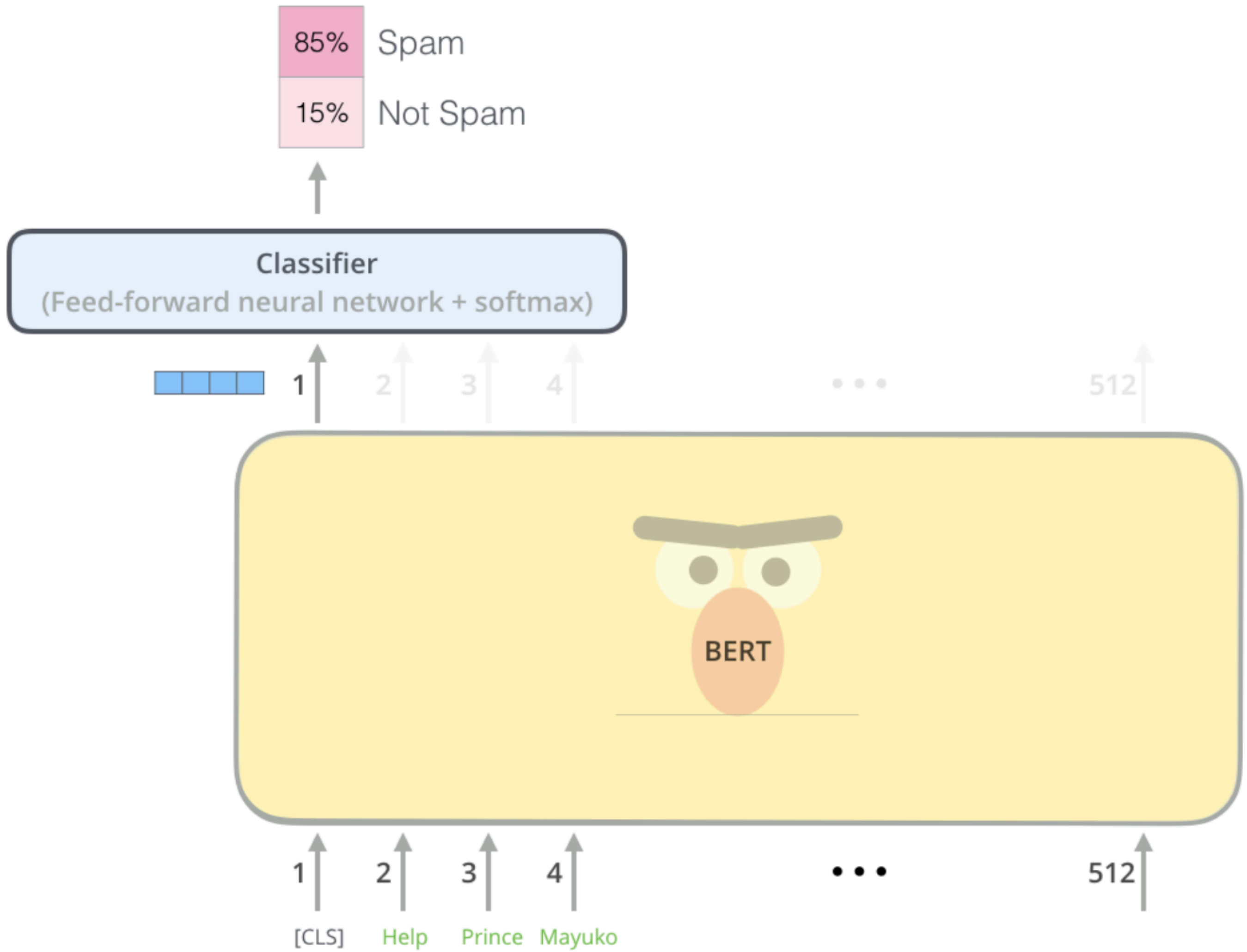
Each position outputs a 768 dim vector in BERT base.



[1]Reference: https://jalammar.github.io/illustrated-bert/

BERT

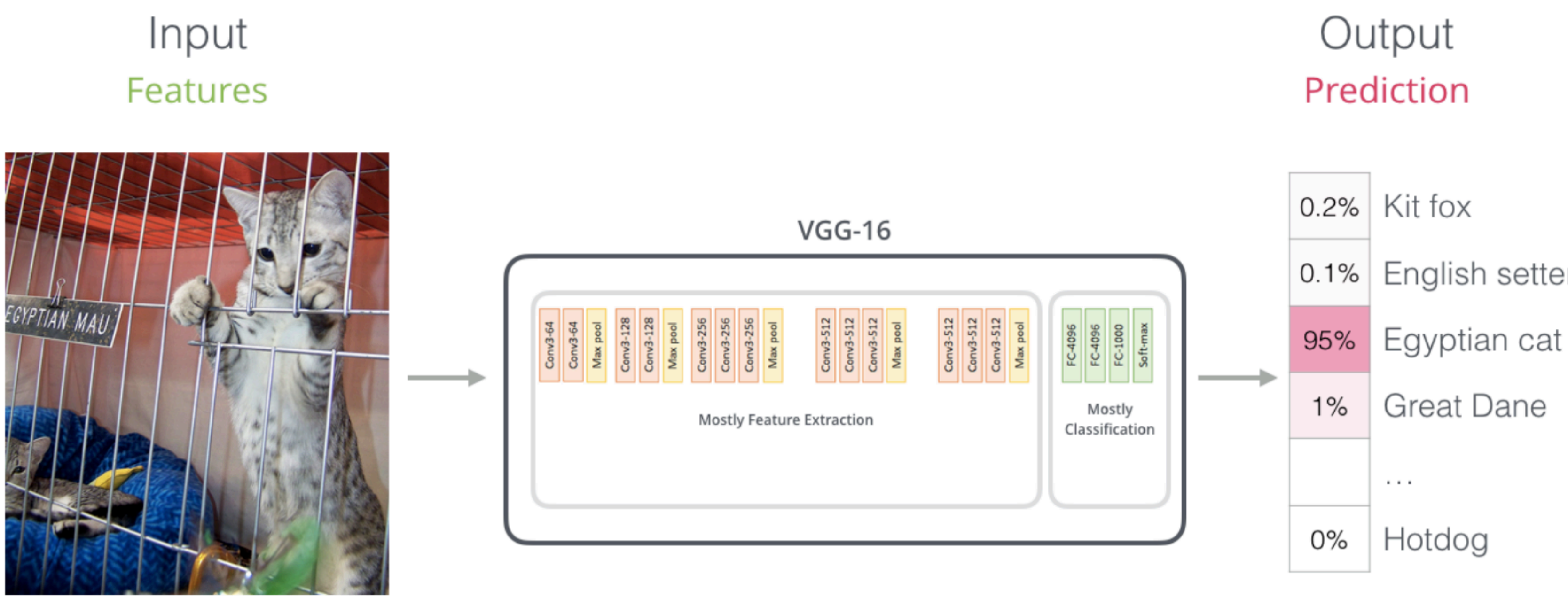
For classification, as we saw earlier, we use only the first vector.



[1]Reference: https://jalammar.github.io/illustrated-bert/

BERT

Similar to a CNN classificer (CNN layers followed by a fully connected layer)



[1] Reference: https://jalammar.github.io/illustrated-bert/

ELMo

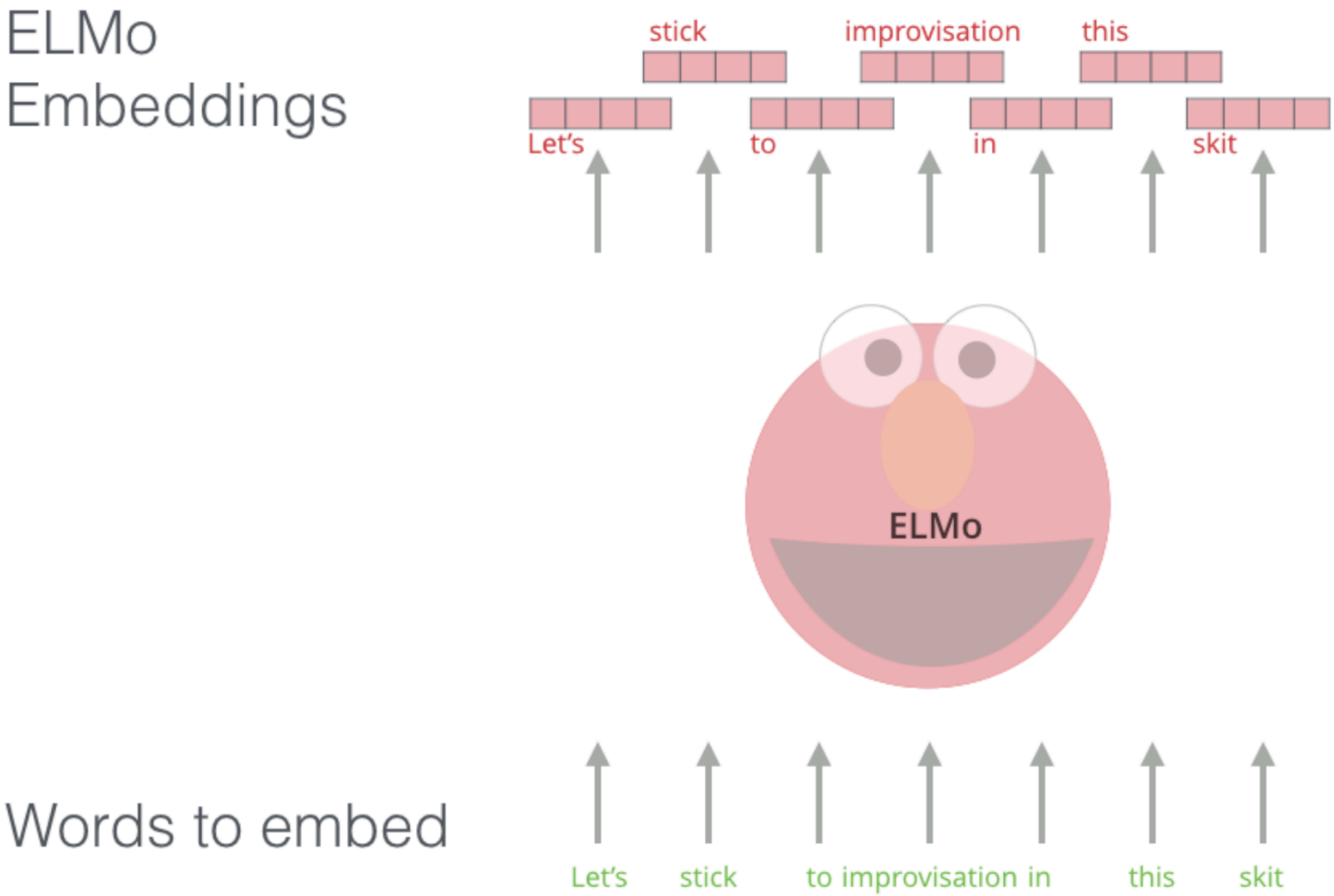
- A word can have different meaning depending on its context
- This was not captured in word2vec and Glove for instance.

- ELMo (2018) produces contextualized word embeddings.

[1]Reference: https://jalammar.github.io/illustrated-bert/ and https://arxiv.org/pdf/1802.05365.pdf

ELMo

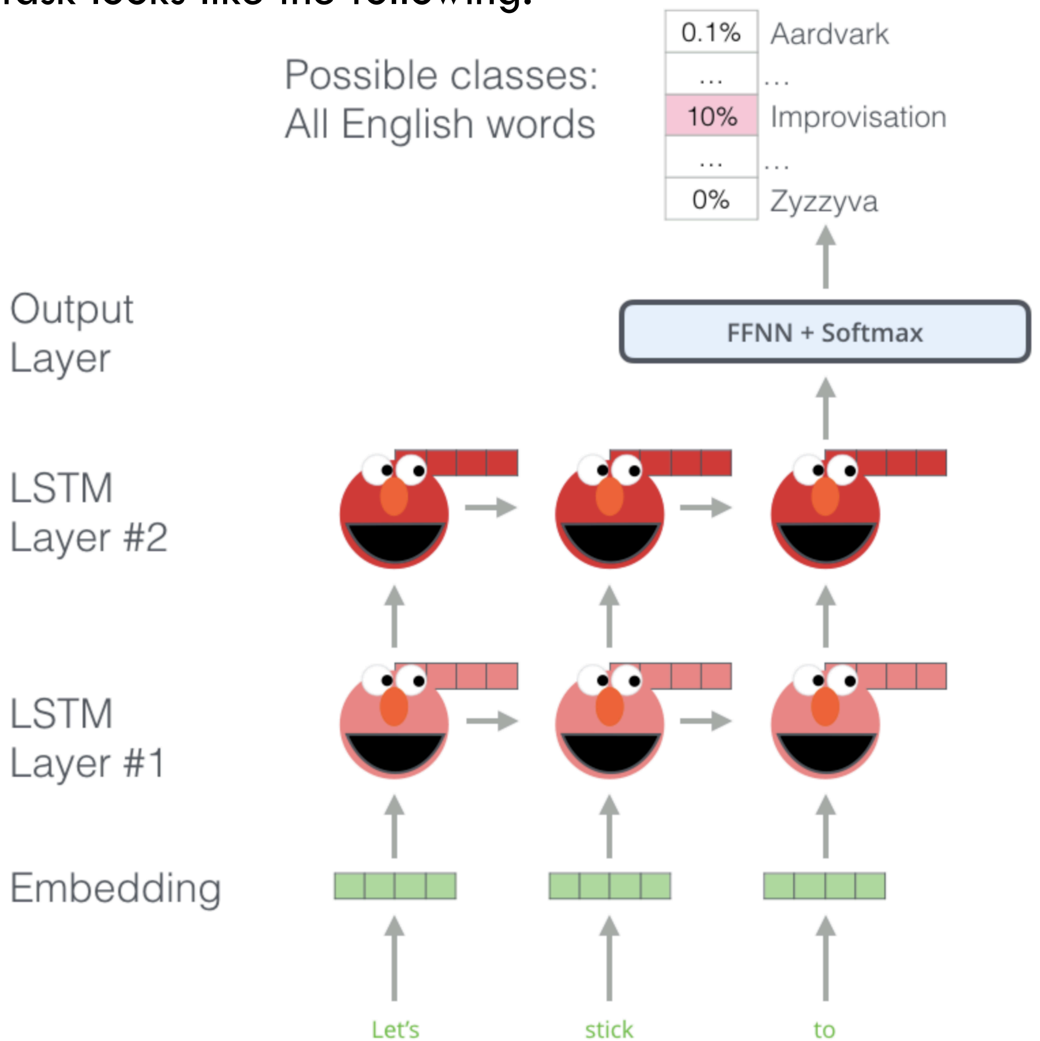
Look at the entire sentence before embedding each word in the sentence. Based on LSTMs. Trained as a language model.



[1]Reference: https://jalammar.github.io/illustrated-bert/

ELMo

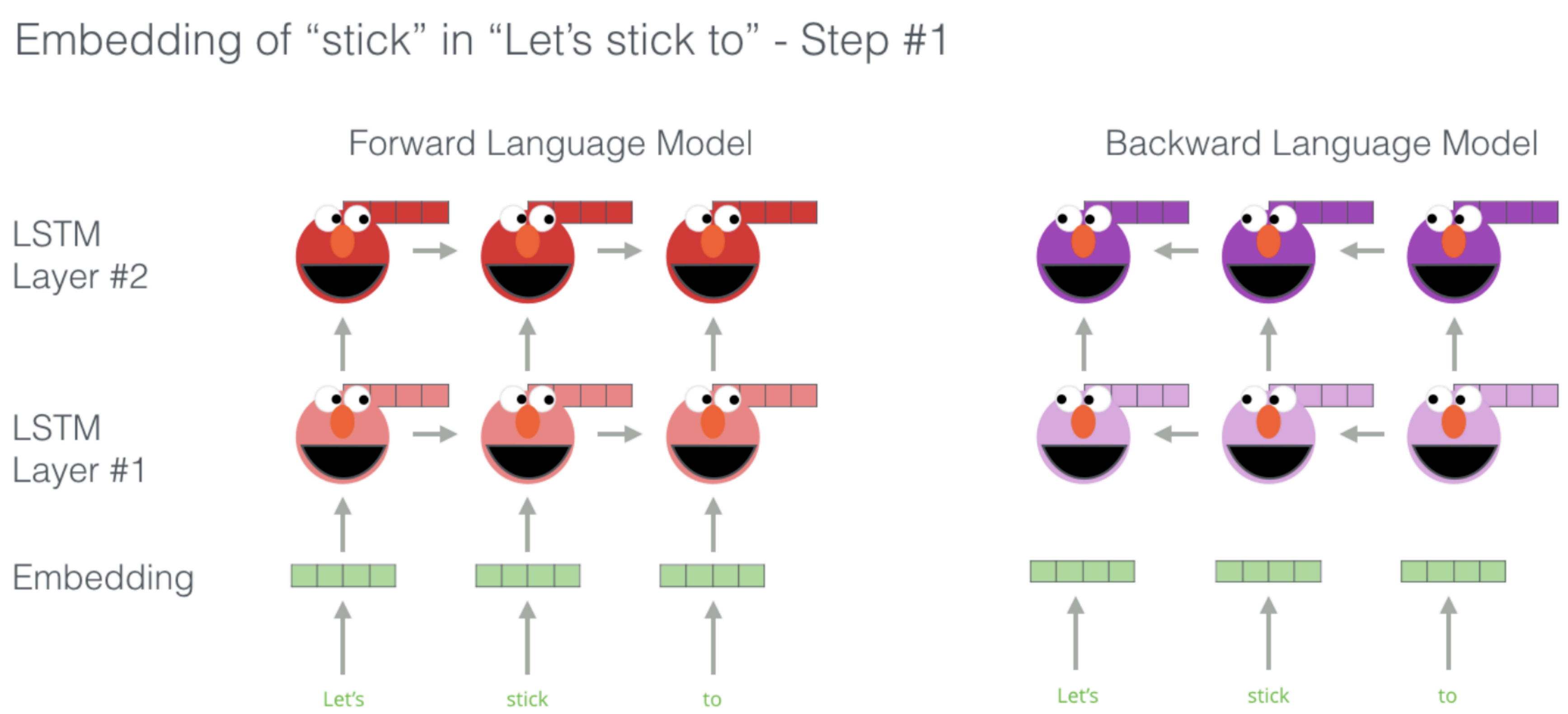
Language modelling task looks like the following:



[1]Reference: https://jalammar.github.io/illustrated-bert/

ELMo

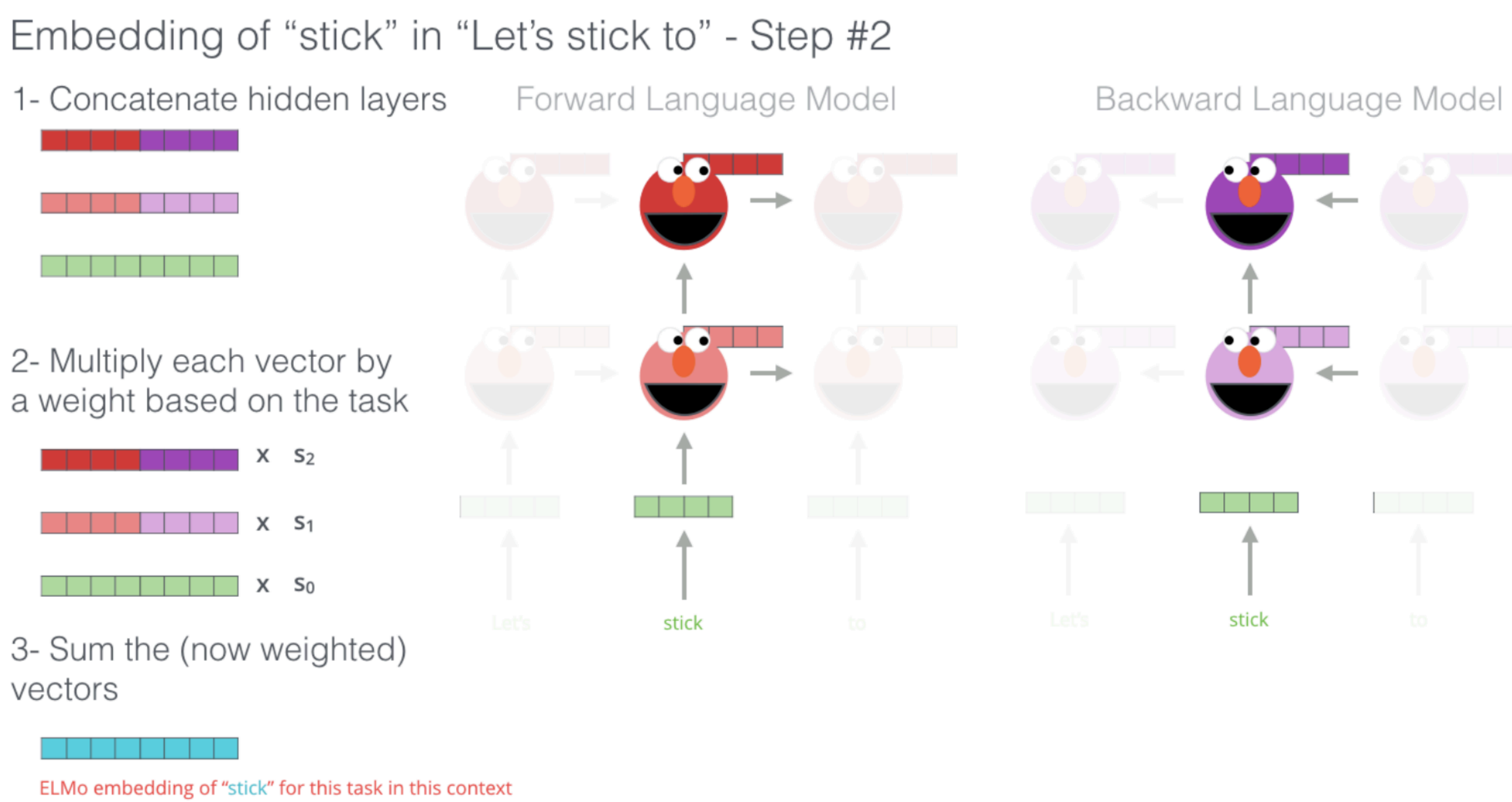
The hidden vectors computed in the forward pass are used for generating embeddings.



[1]Reference: https://jalammar.github.io/illustrated-bert/

ELMo

ELMo is actually a bidirectional LSTM. The hidden vectors are aggregated to get the embedding.



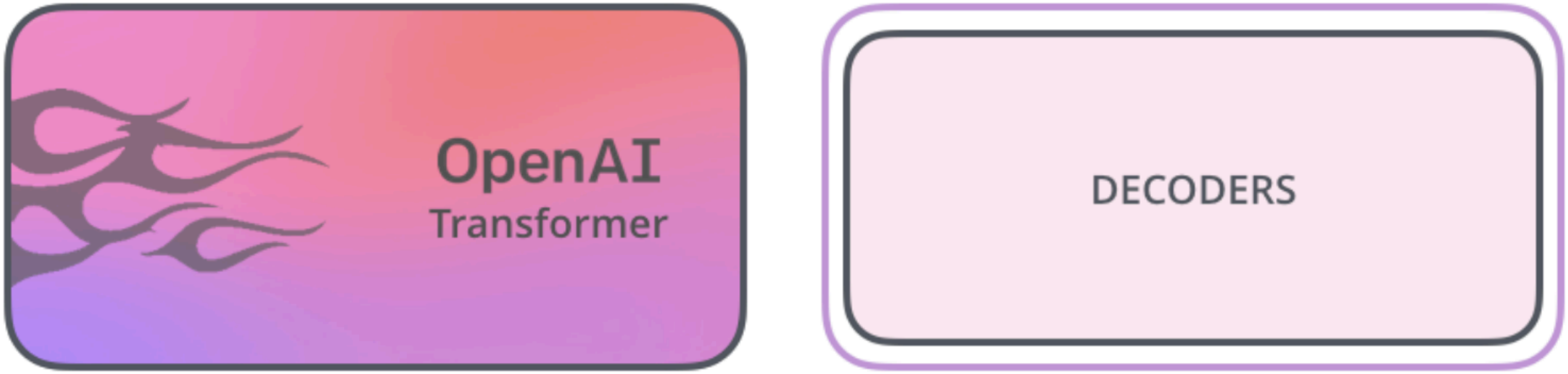
In addition to embeddings, the model parameters can also be changed later on (ULM-FiT)

[1]Reference: https://jalammar.github.io/illustrated-bert/

OpenAI Transformer

Transformers are able to capture long-term dependencies better than LSTMs (empirical)

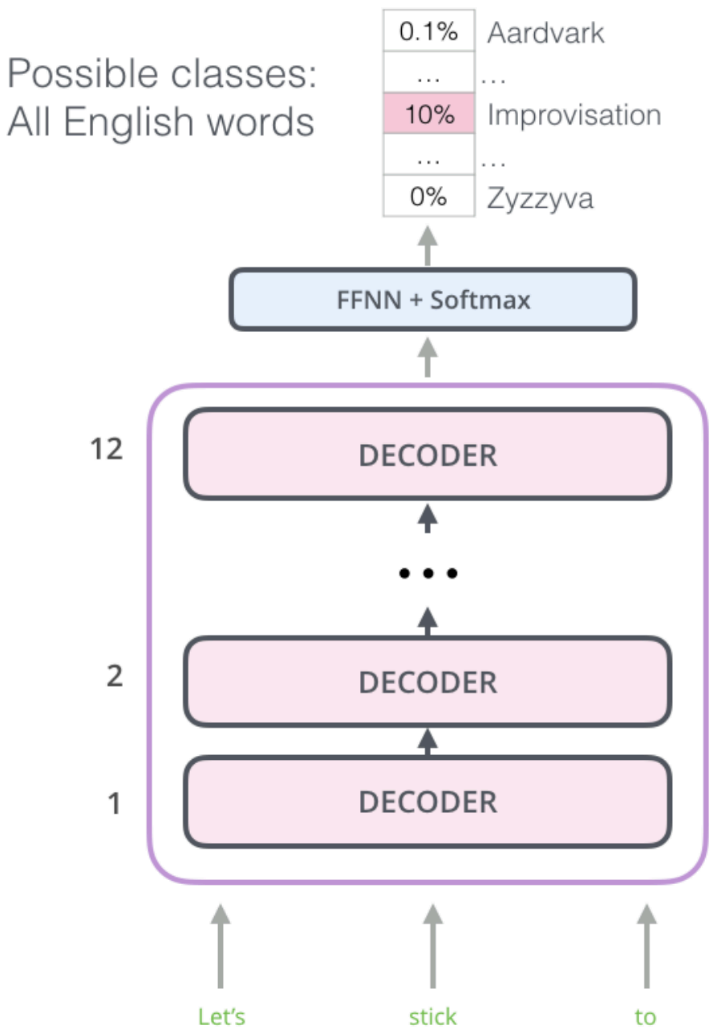
Use just the decoder for language modelling. Can predict the next word and masks future tokens.



[1]Reference: https://jalammar.github.io/illustrated-bert/

OpenAI Transformer

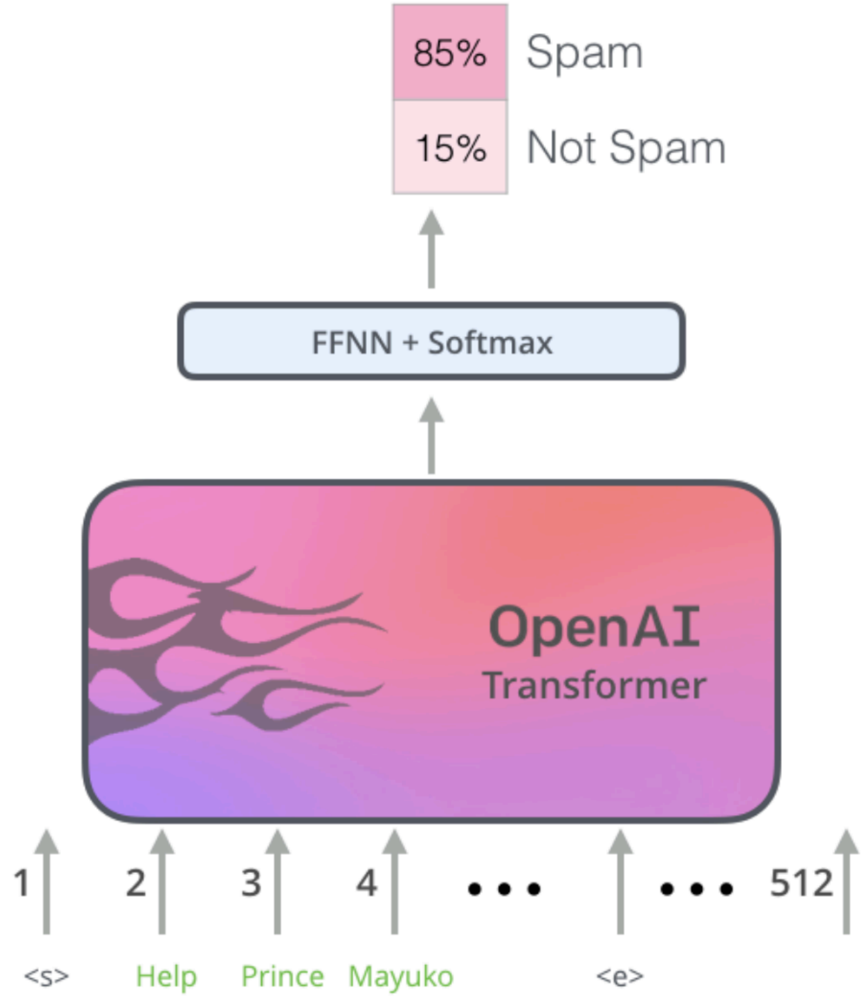
Has 12 decoder units (the encoder-decoder attention is removed).



[1]Reference: https://jalammar.github.io/illustrated-bert/

OpenAI Transformer

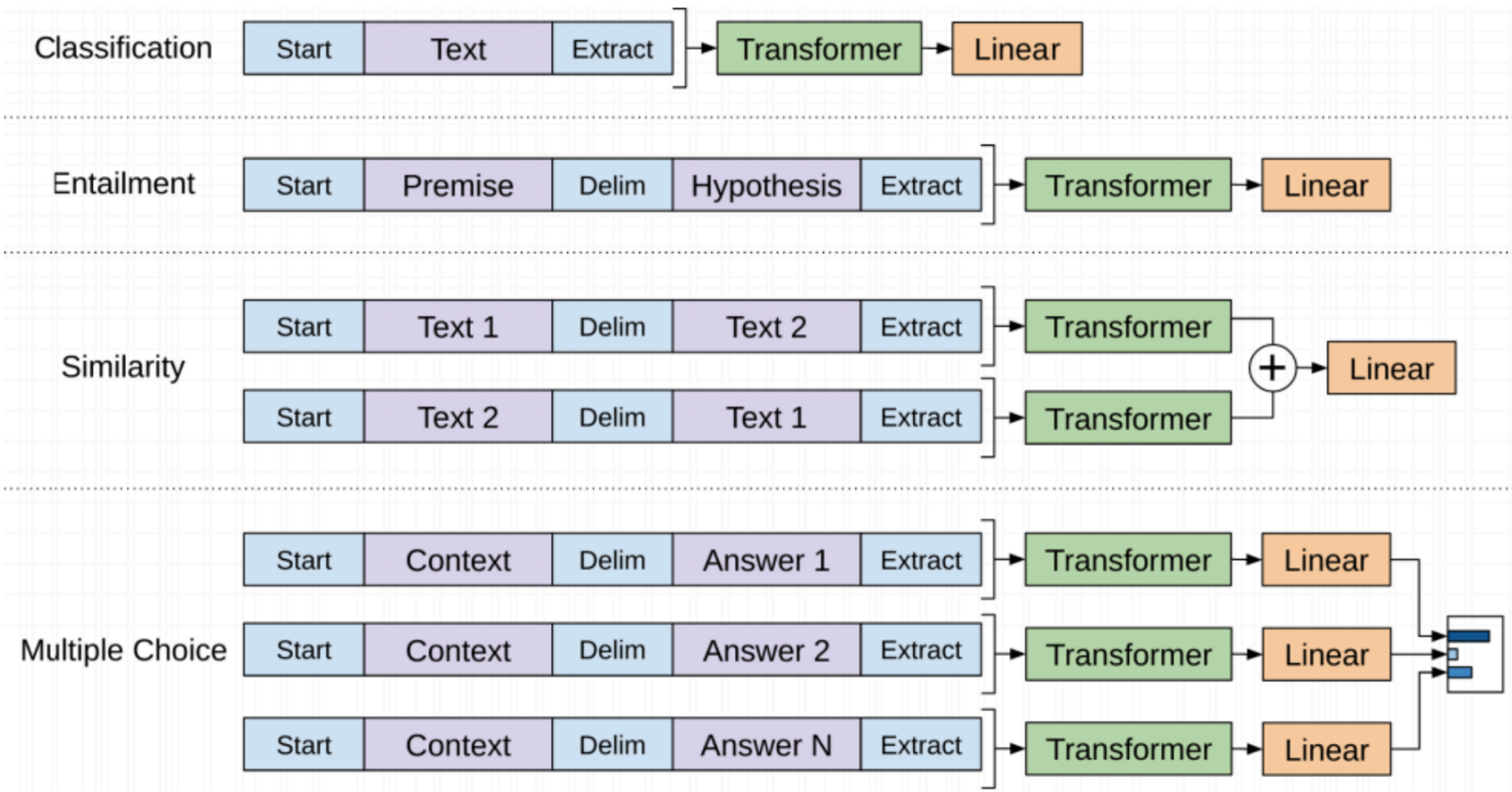
Can then be used for downstream NLP tasks.



[1]Reference: https://jalammar.github.io/illustrated-bert/

OpenAI Transformer

Suitably processing the input can allow the OpenAI Transformer to be used for various tasks



[1]Reference: https://jalammar.github.io/illustrated-bert/

BERT

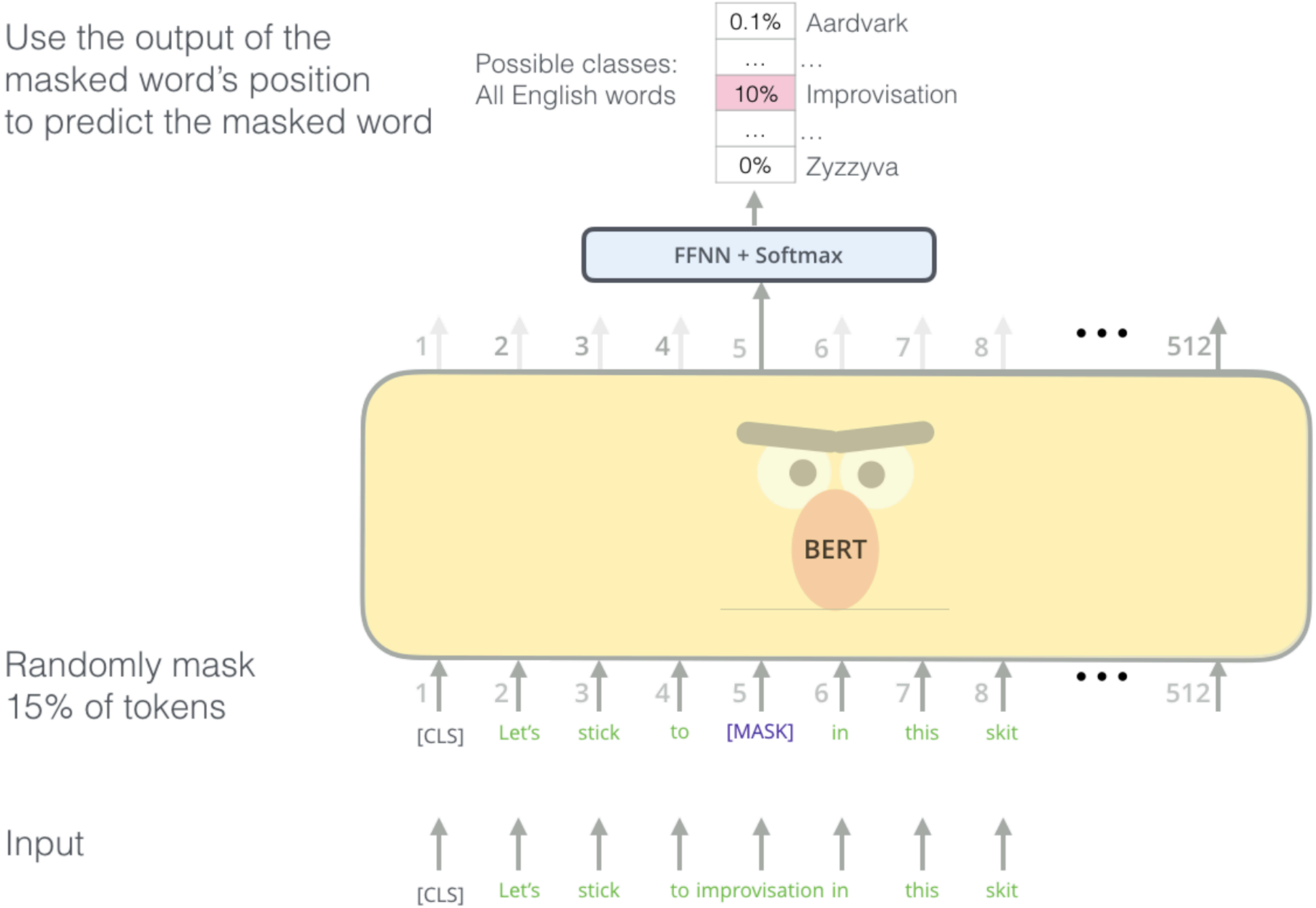
- ELMo was bi-directional but OpenAI Transformer was not

- The next natural idea (that lead to BERT) is whether a transformer-based model can look both forward and backward while predicting the next word.

[1]Reference: https://jalammar.github.io/illustrated-bert/

BERT

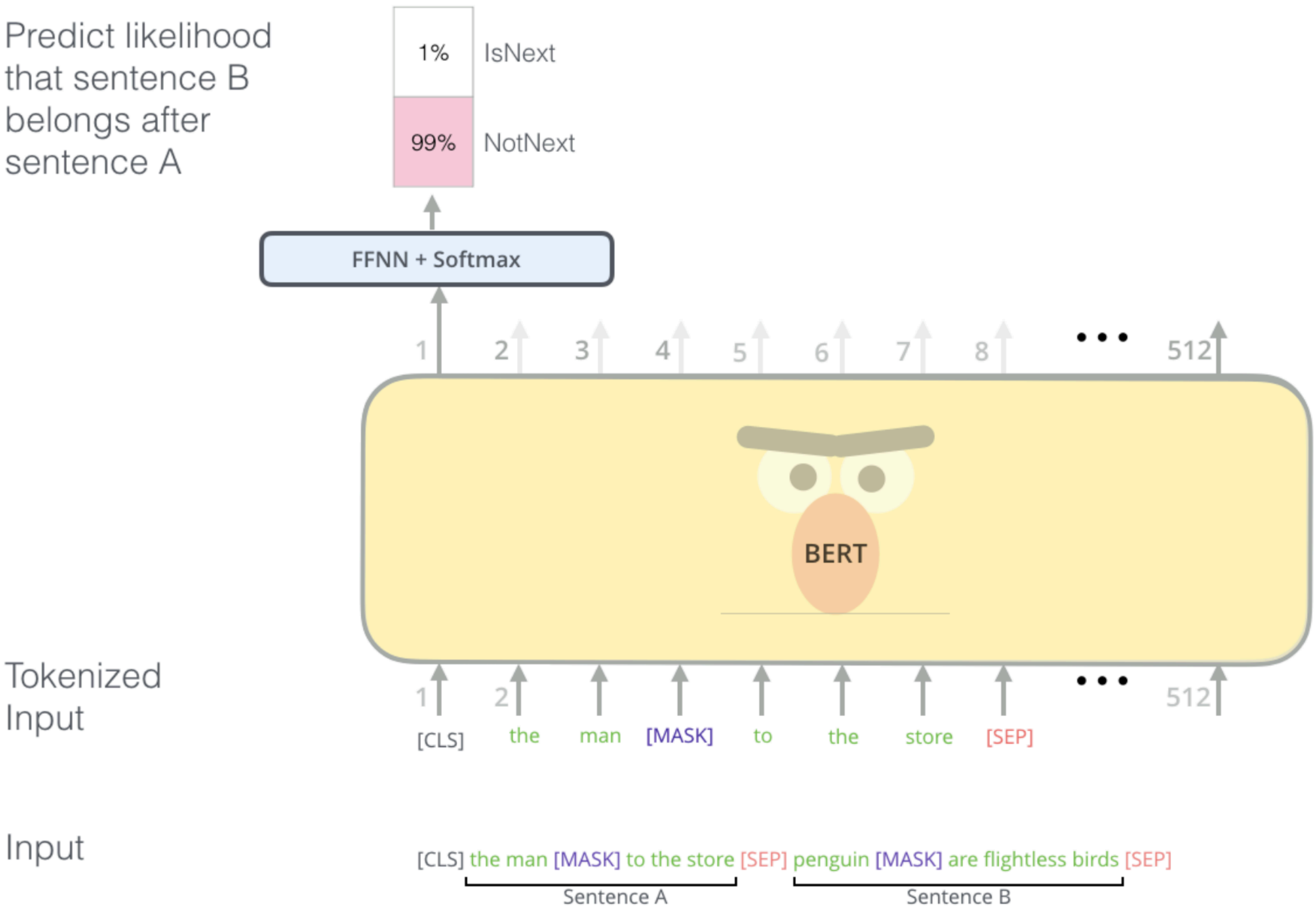
The key idea is to use **masks and encoders.** We need to prevent word from seeing itself.
We will skip much of the details here about masking.



[1]Reference: https://jalammar.github.io/illustrated-bert/

BERT

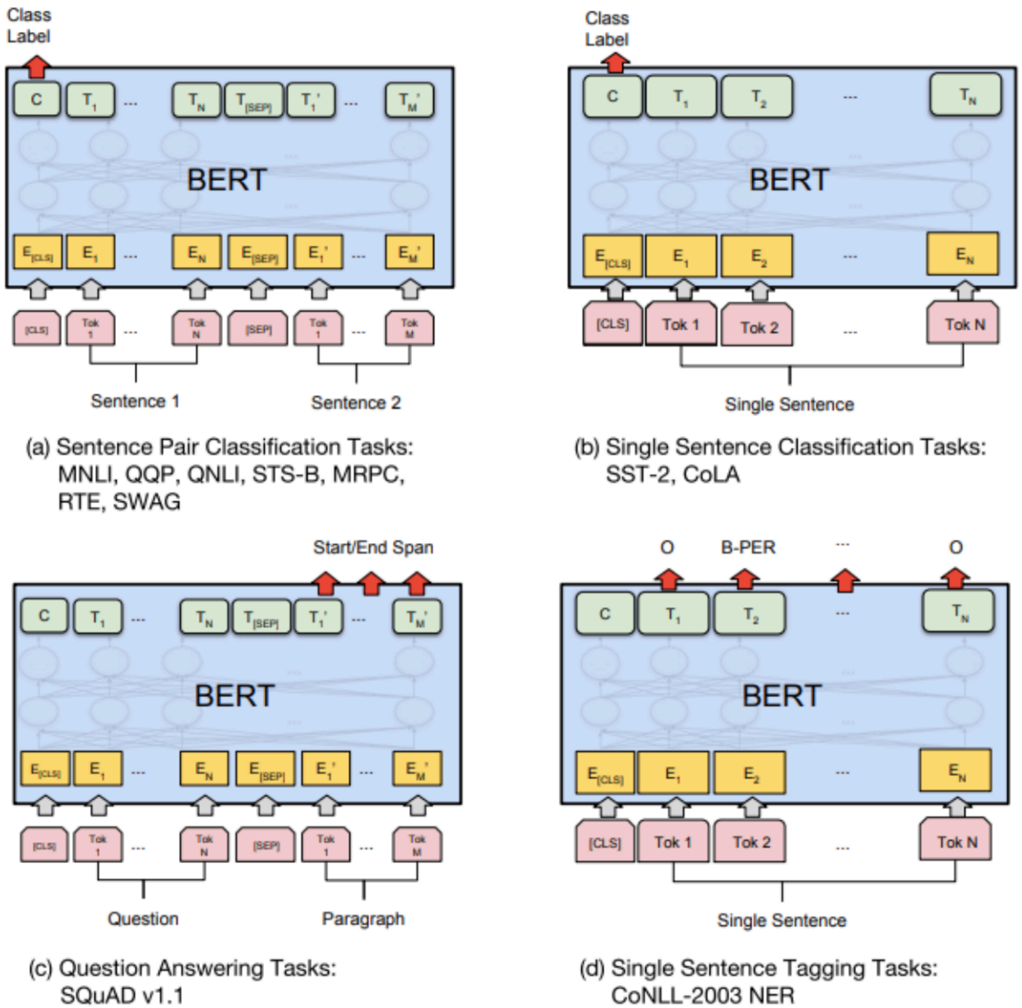
In addition to language modelling, BERT also pre-trains on sentence sequencing task.



[1]Reference: https://jalammar.github.io/illustrated-bert/

BERT

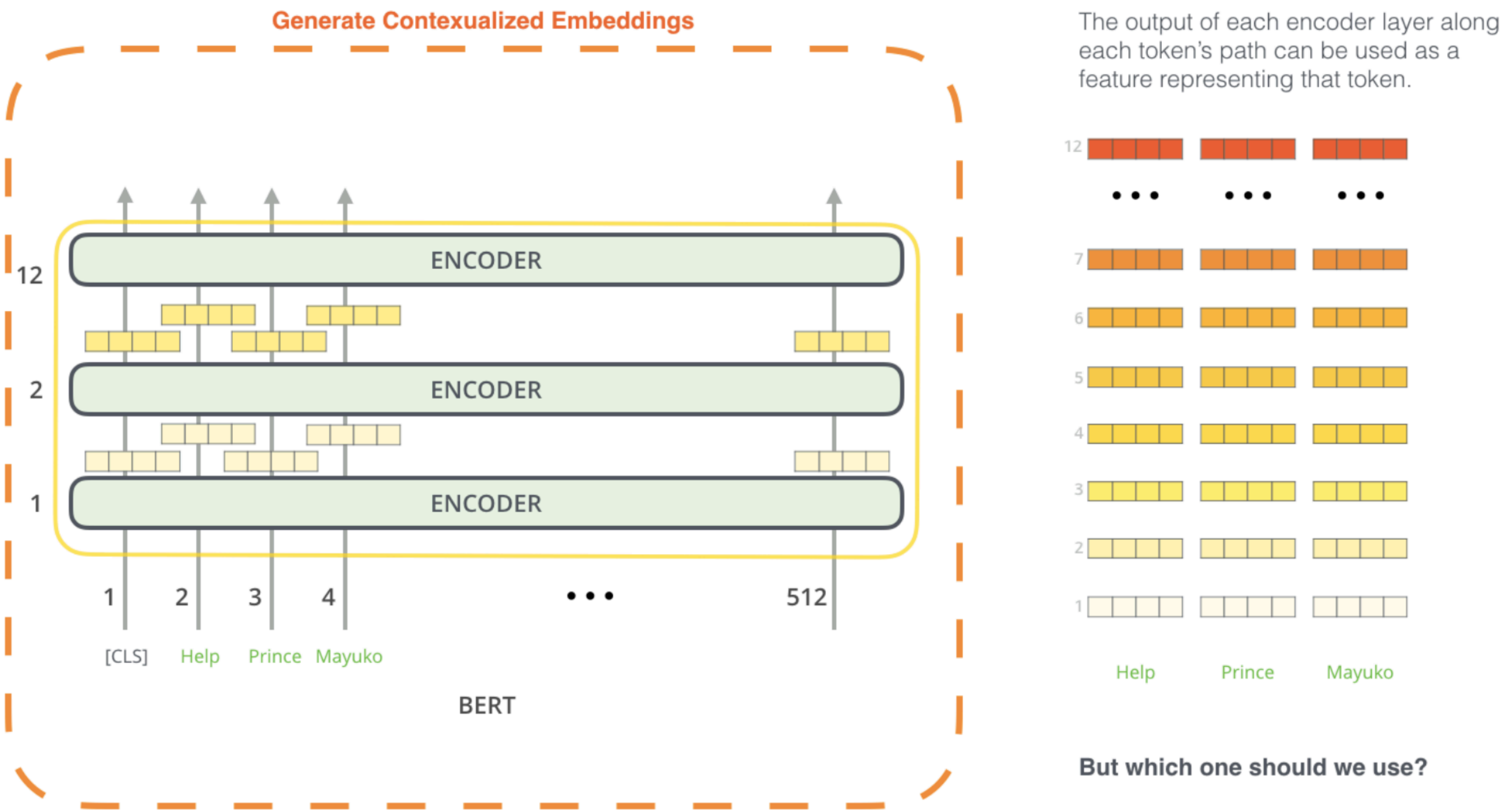
Pre-trained BERT can be used for other tasks (beyond classification) as well:



[1]Reference: https://jalammar.github.io/illustrated-bert/

BERT

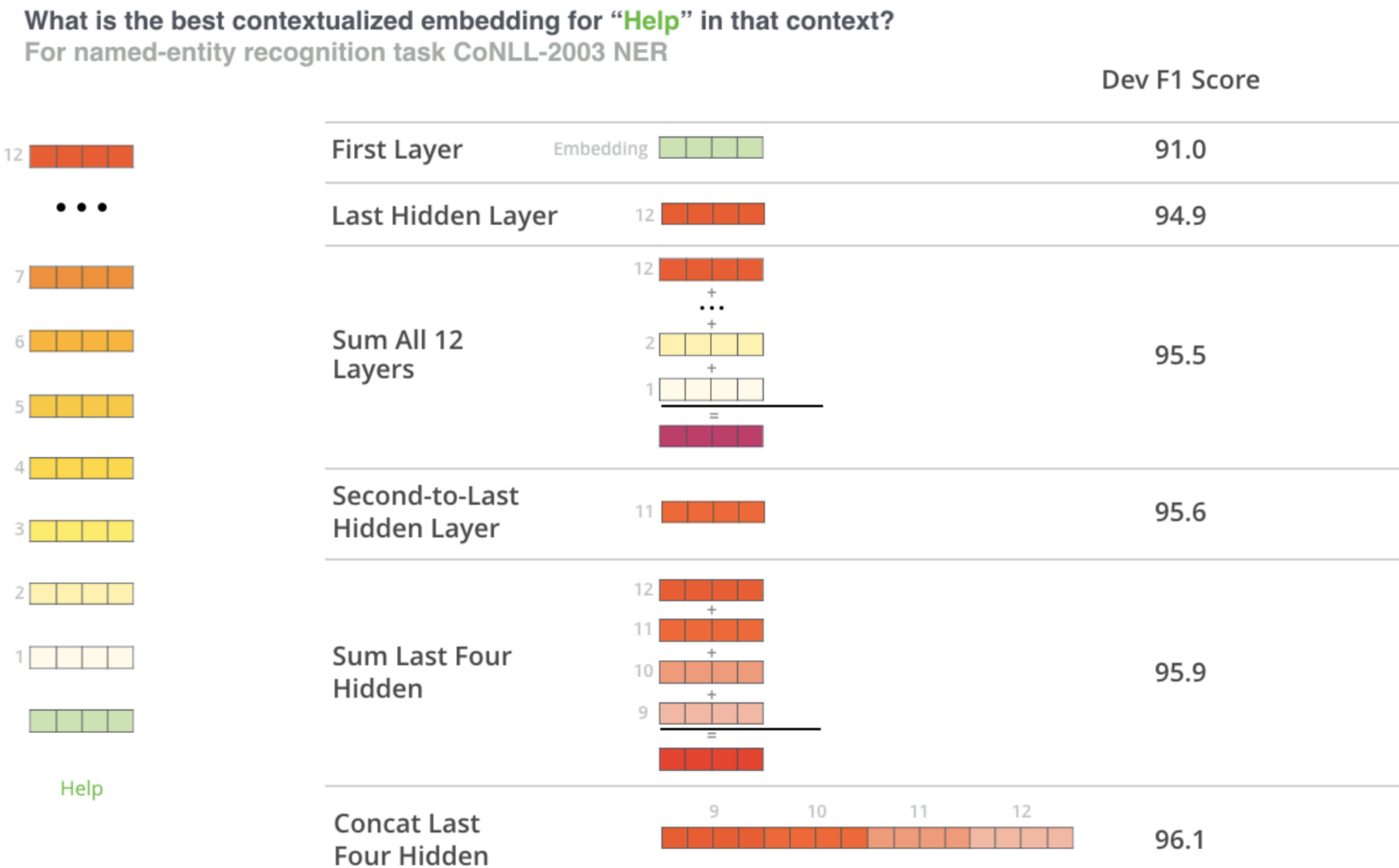
BERT can be used as a word embedding model like ELMo. The embeddings are contextual.



[1]Reference: https://jalammar.github.io/illustrated-bert/

BERT

The choice of which hidden vector to use as the word-embedding can be data driven.



[1] Reference: https://jalammar.github.io/illustrated-bert/

Questions?

Summary

- Self-attention is the key building block of transformer variants
- Transformer based encoders can be used for contextualized embeddings of words
- BERT and related architectures can be used to improve many NLP tasks. This is similar to using pre-trained vision models (e.g., resnet50). Finetuning can also be done.
- Readily available pre-trained models alleviate the need for compute heavy resources in application specific ML projects

- Exercise: BERT finetuning tutorial on Google Colab
  - https://colab.research.google.com/github/tensorflow/tpu/blob/master/tools/colab/bert_finetuning_with_cloud_tpus.ipynb