How NLP Cracked Transfer Learning

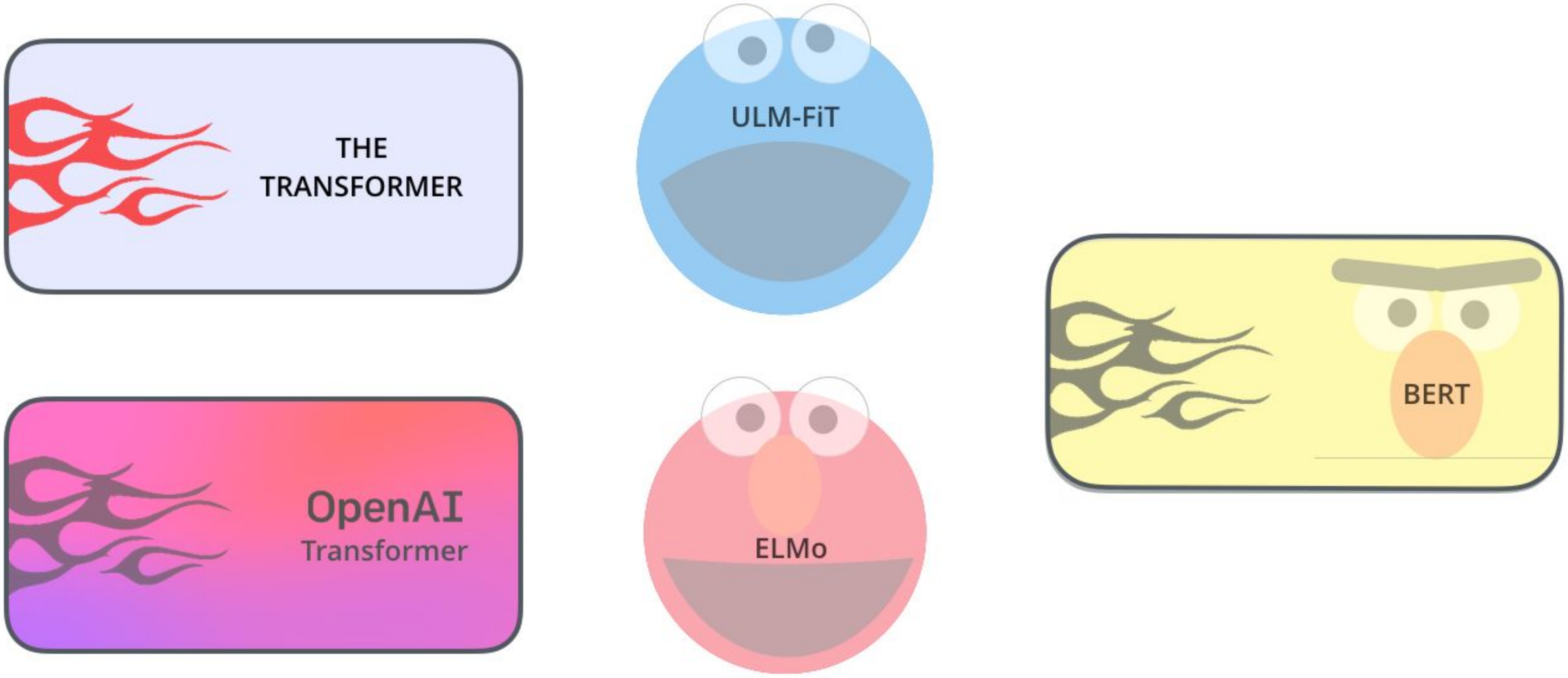
Anastasia Ianina

Harbour.Space, Barcelona

Outline

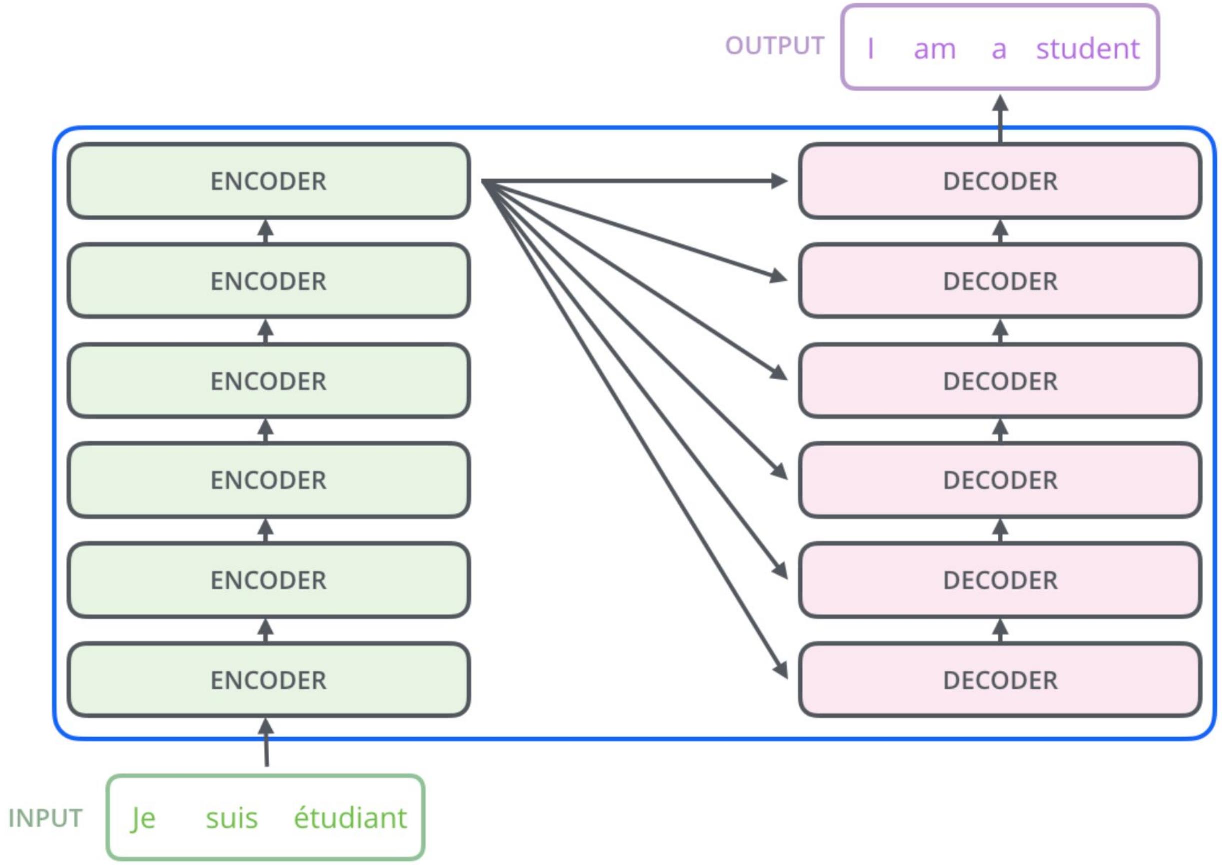
1. Transformer: recap
2. OpenAI Transformer
3. ELMO
4. BERT
5. BERTology

Based on: http://web.stanford.edu/class/cs224n/slides/cs224n-2019-lecture13-contextual-representations.pdf
https://jalammar.github.io/illustrated-transformer/
http://jalammar.github.io/illustrated-bert/

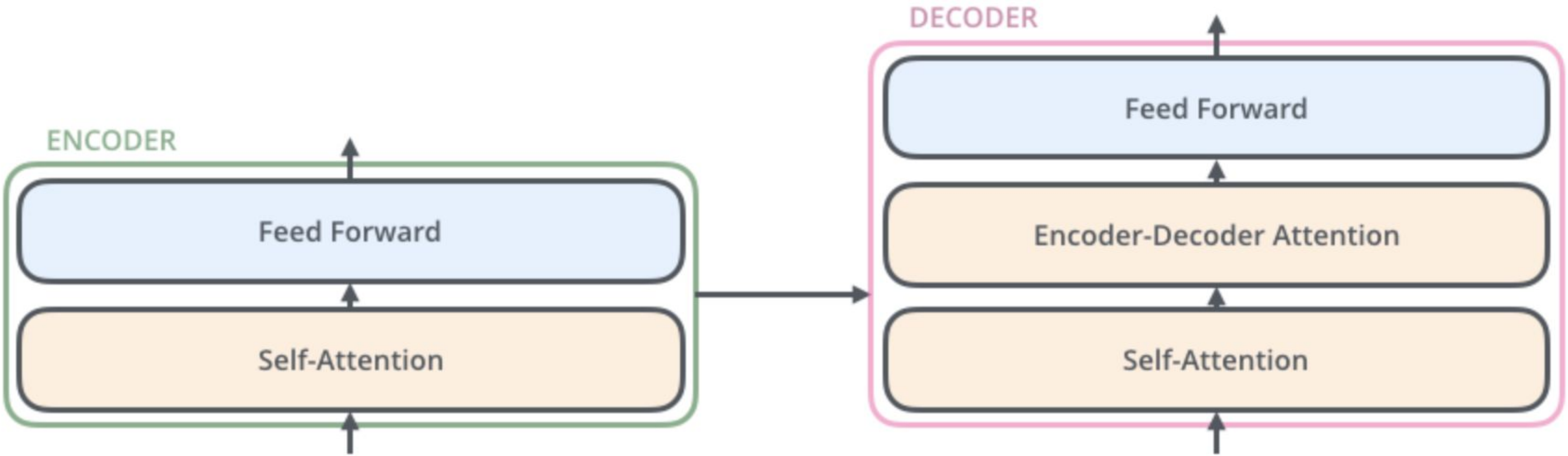

THE TRANSFORMER
ULM-FiT
BERT
OpenAI Transformer
ELMo

The Transformer: recap

The Transformer

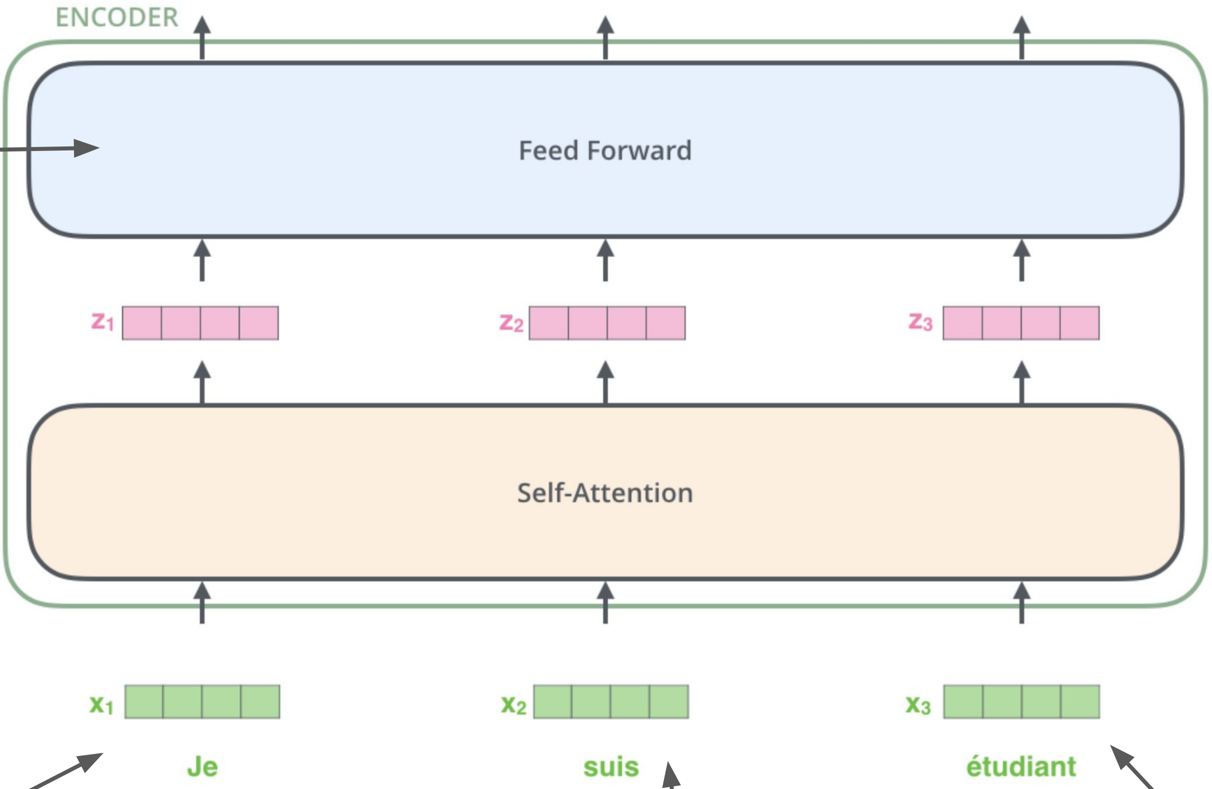


The Transformer



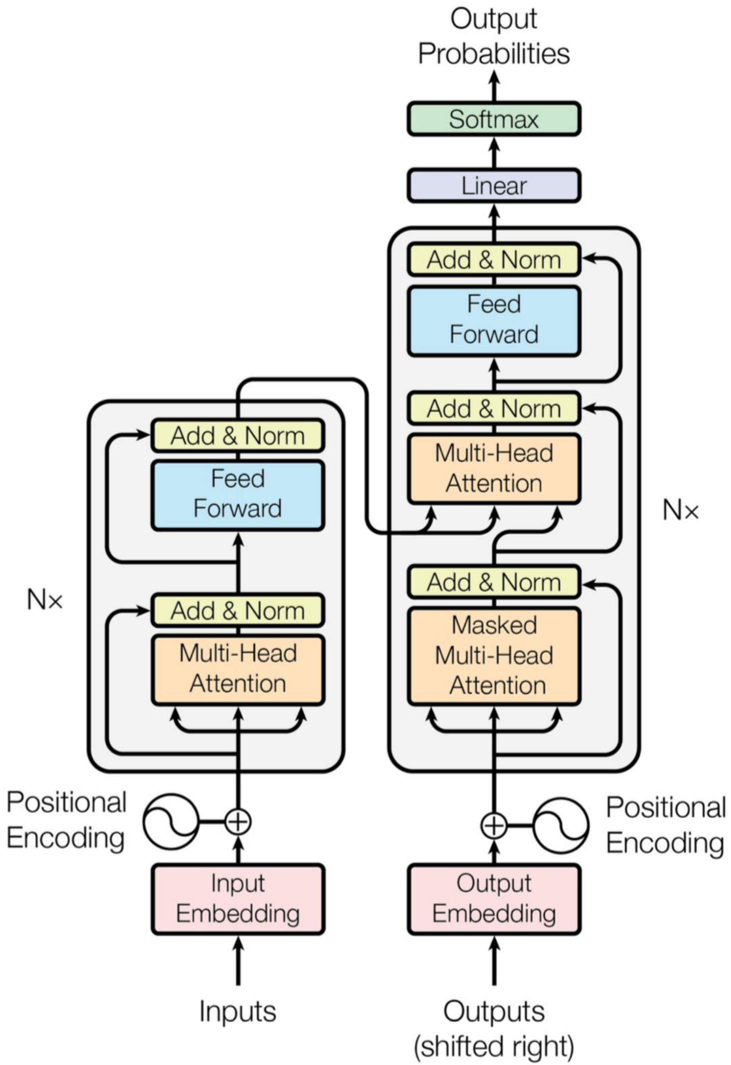
The Transformer

Can be parallelized



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

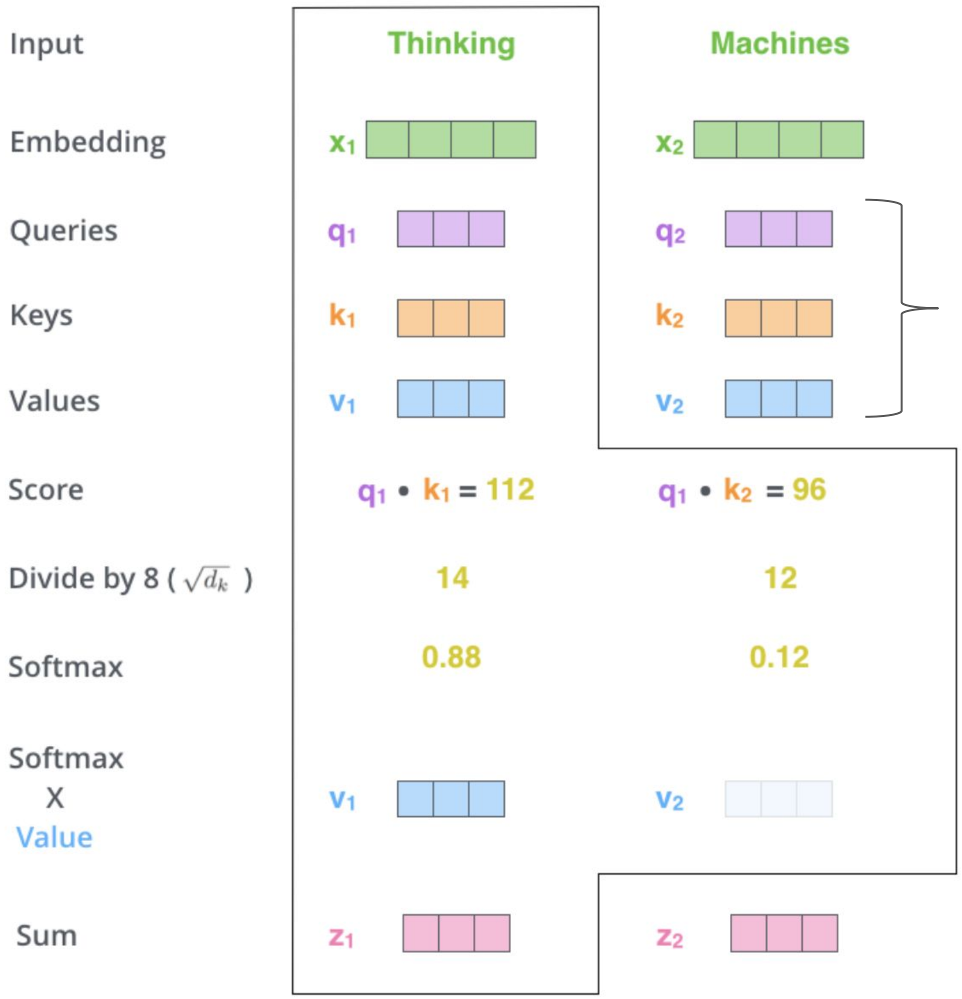
The Transformer: recap



- Proposed in the paper "Attention is All You Need" (Ashish Vaswani et al.)
- No recurrent or convolutional neural networks -> just attention
- Uses Multi-Head **self-attention** concept

Self-Attention: recap

Self-Attention in Detail



STEP 1: create Query, Key, Value

STEP 2: calculate scores

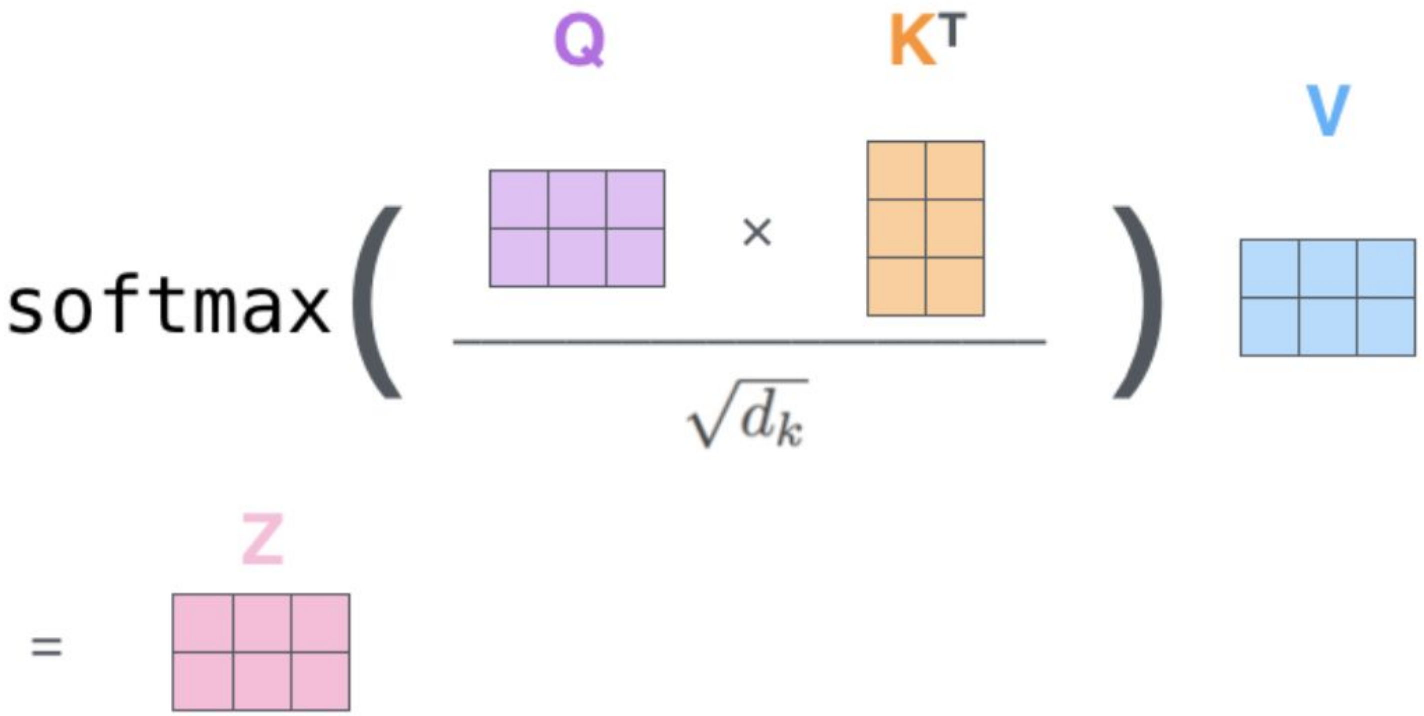
STEP 3: divide by $\sqrt{d_k}$

STEP 4: softmax

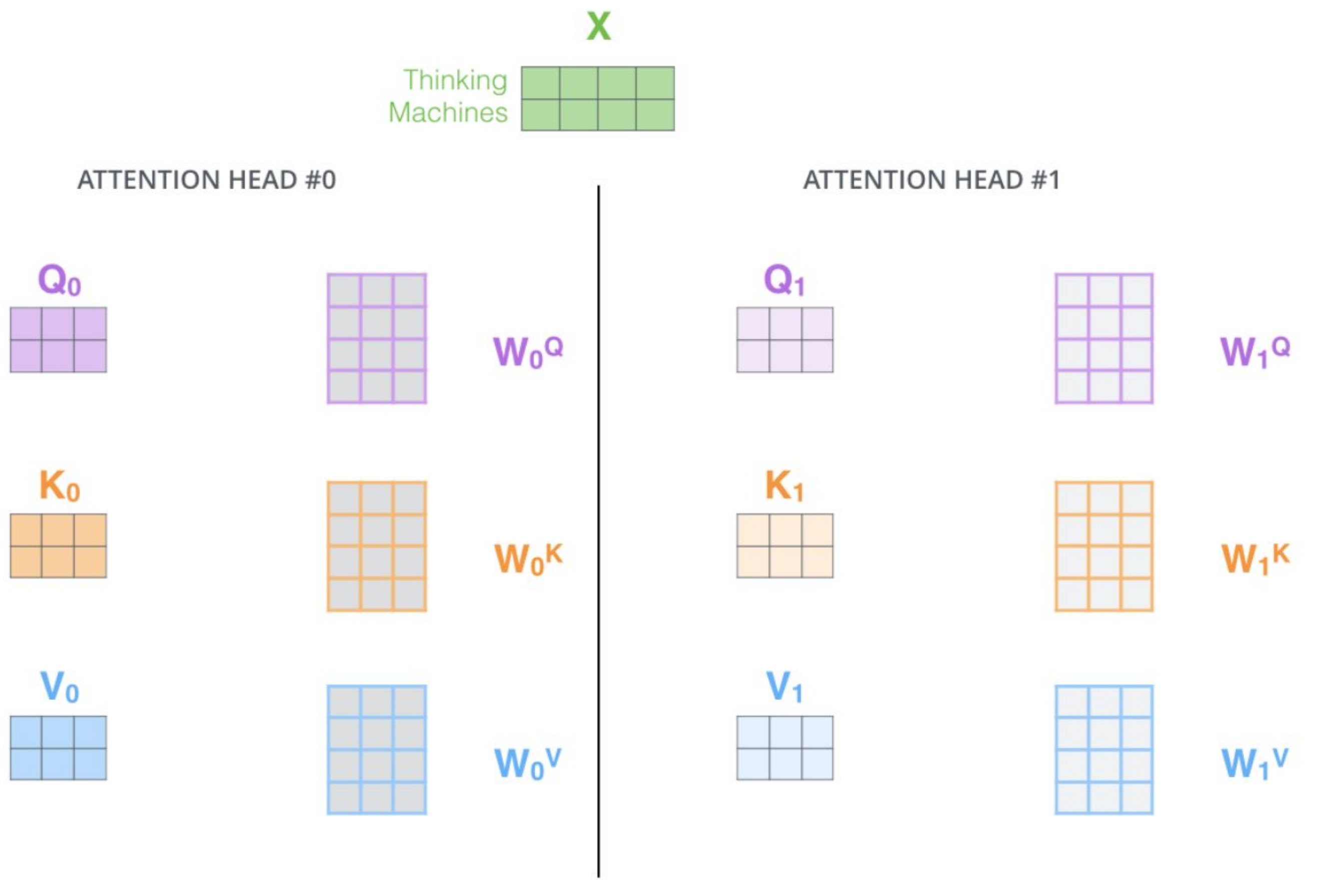
STEP 5: multiply each value vector by the softmax score

STEP 6: sum up the weighted value vectors

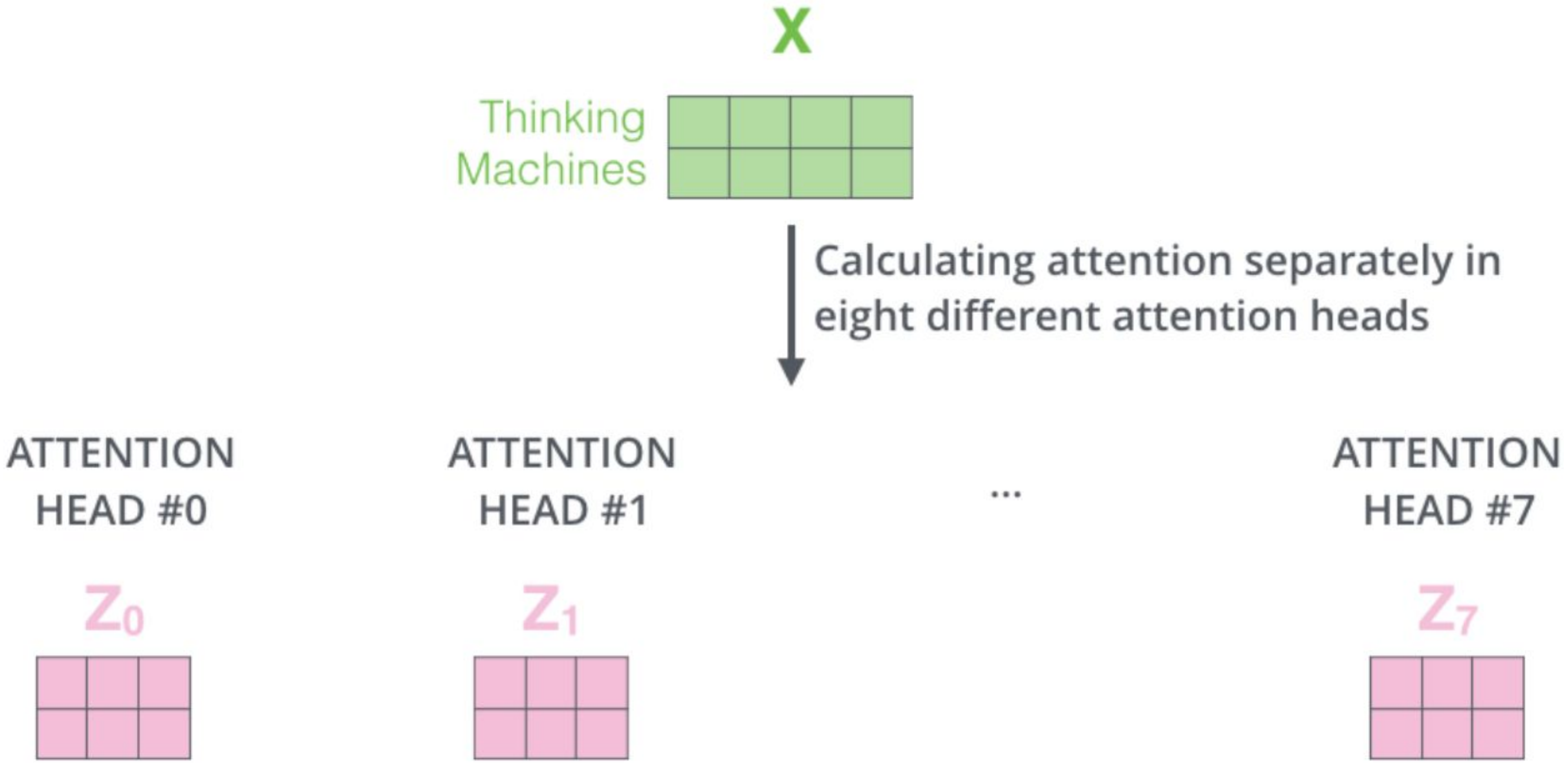
Self-Attention: Matrix Calculation



Multi-Head Attention



Multi-Head Attention



Multi-Head Attention

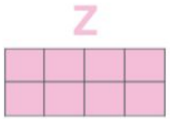
1) Concatenate all the attention heads

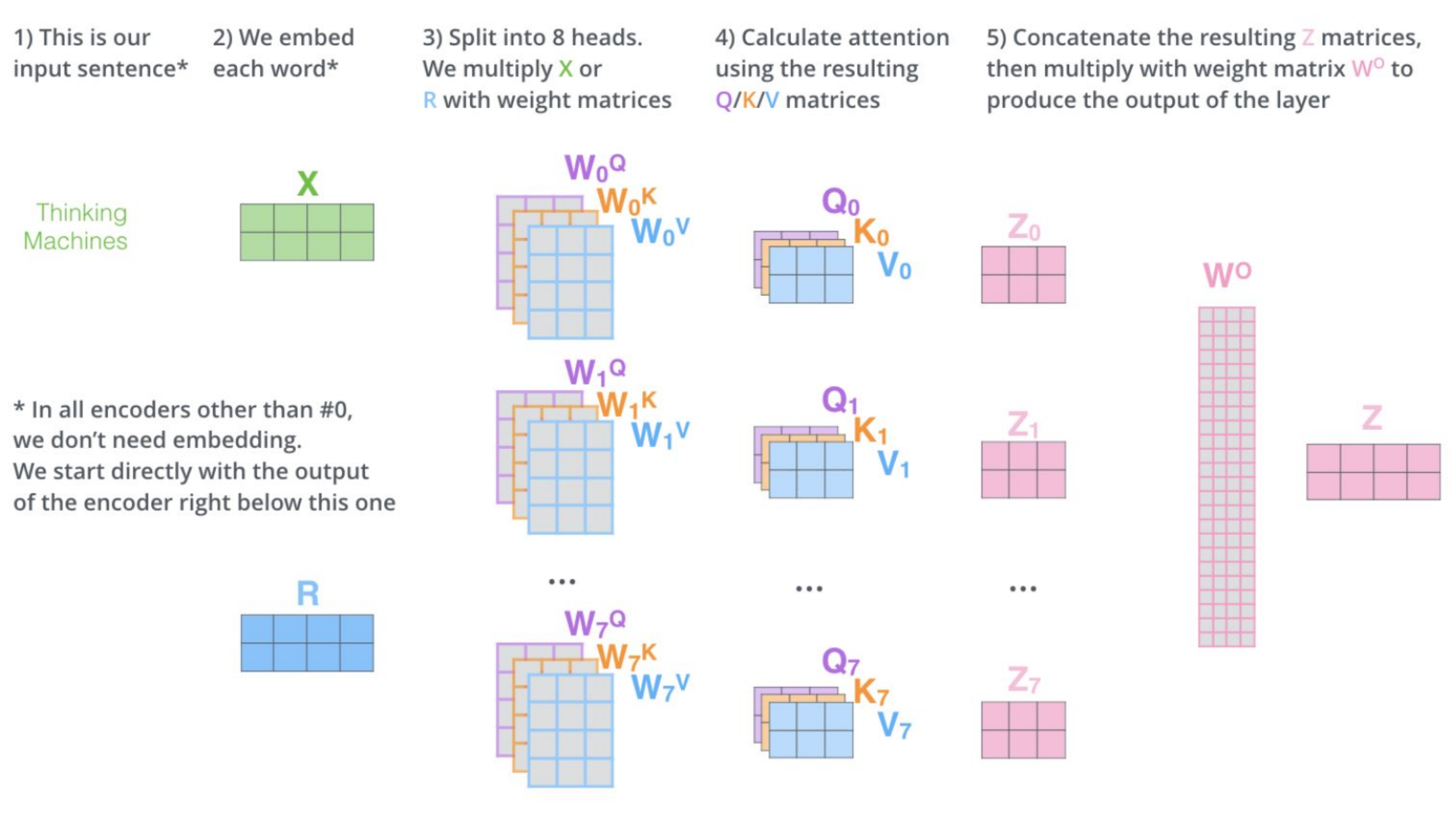
2) Multiply with a weight matrix W^O that was trained jointly with the model

X

3) The result would be the Z matrix that captures information from all the attention heads. We can send this forward to the FFNN

=



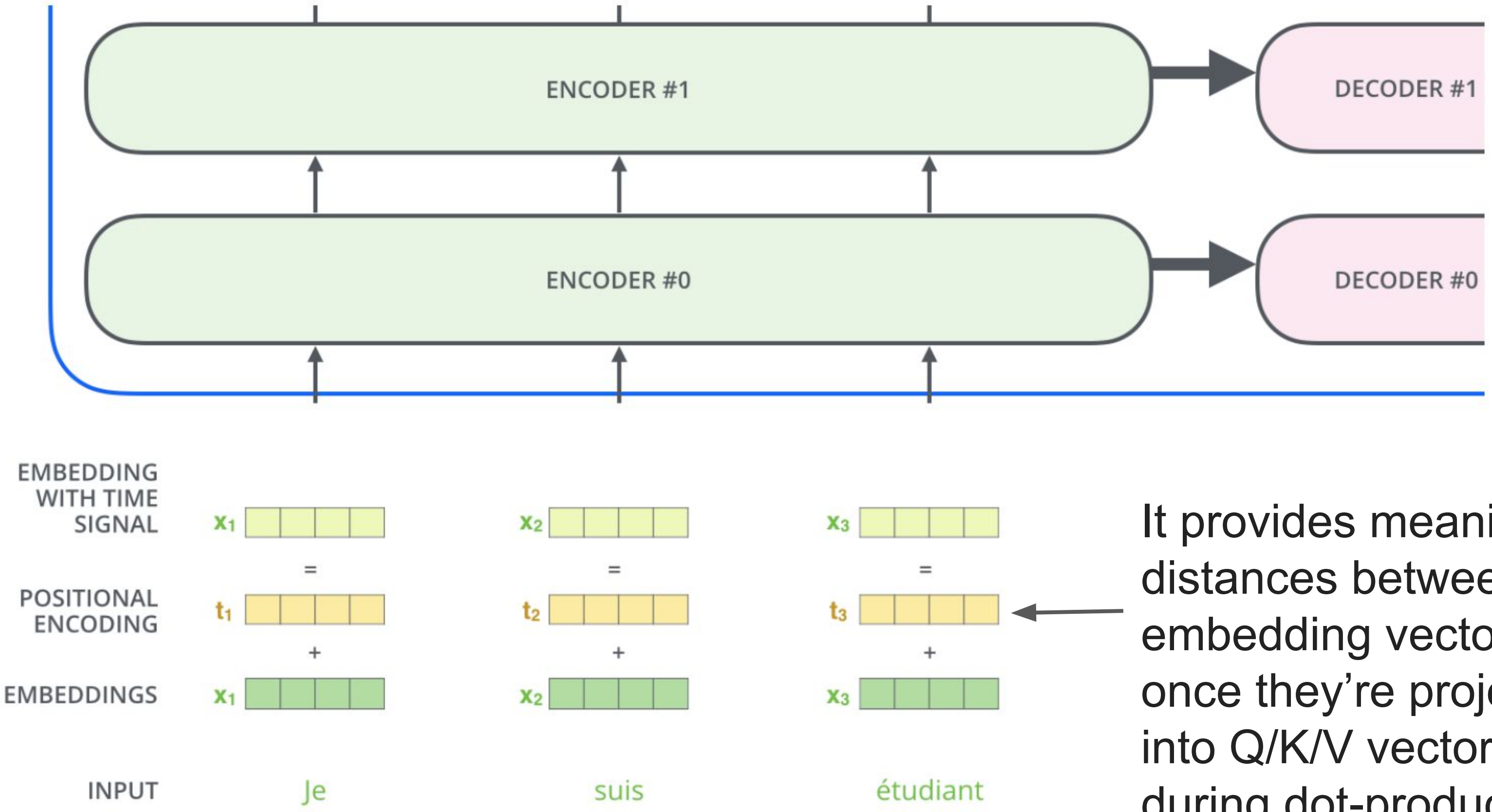

1) This is our input sentence*
2) We embed each word*
3) Split into 8 heads. We multiply X or R with weight matrices
4) Calculate attention using the resulting Q/K/V matrices
5) Concatenate the resulting Z matrices, then multiply with weight matrix W^O to produce the output of the layer
Thinking Machines
X
R
* In all encoders other than #0, we don't need embedding. We start directly with the output of the encoder right below this one
W_0^Q
W_0^K
W_0^V
W_1^Q
W_1^K
W_1^V
W_7^Q
W_7^K
W_7^V
Q_0
K_0
V_0
Q_1
K_1
V_1
Q_7
K_7
V_7
Z_0
Z_1
Z_7
...
W^O
Z

Positional Encoding

Positional encoding requirements

- Positional encoding should be unique for every position in the sequence
- Distance between two same positions should be preserved with sequences of different length
- The positional encoding should be deterministic
- *It would be great if it would work with long sequences (longer than any sequence in the training set)*

Positional Encoding



It provides meaningful distances between the embedding vectors once they're projected into Q/K/V vectors and during dot-product attention

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/

Positional Encoding

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

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

Positional Encoding

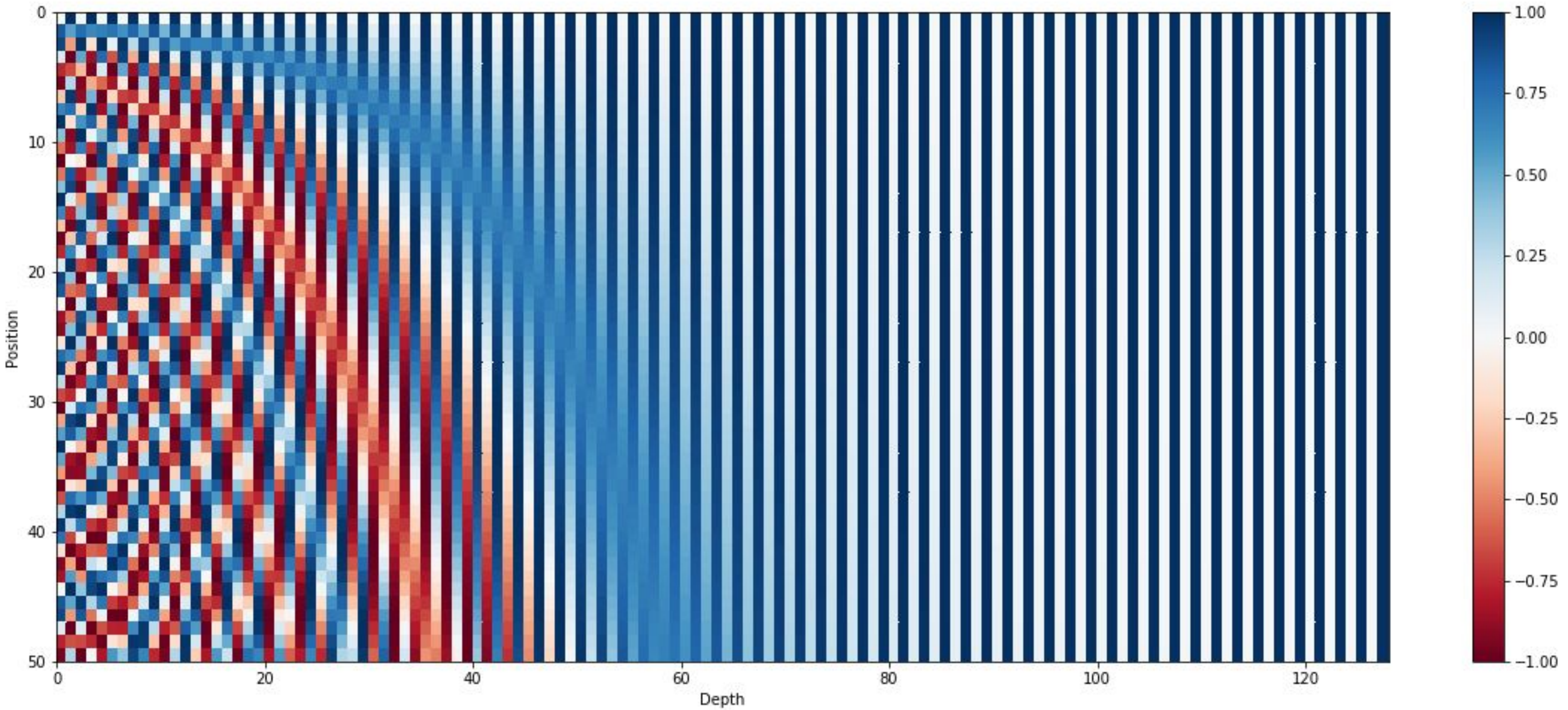


Image source: https://kazemnejad.com/blog/transformer_architecture_positional_encoding/

Positional Encoding: why sin and cos?

We chose this function because we hypothesized it would allow the model to easily learn to attend by relative positions, since for any fixed offset k, PEpos+k can be represented as a linear function of PEpos.

$$M \begin{bmatrix} \sin(\omega_k t) \\ \cos(\omega_k t) \end{bmatrix} = \begin{bmatrix} \sin(\omega_k (t + \phi)) \\ \cos(\omega_k (t + \phi)) \end{bmatrix}$$

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

Positional Encoding: why sin and cos?

$$\begin{bmatrix} u_1 & v_1 \\ u_2 & v_2 \end{bmatrix} \begin{bmatrix} \sin(\omega_k t) \\ \cos(\omega_k t) \end{bmatrix} = \begin{bmatrix} \sin(\omega_k (t+\phi)) \\ \cos(\omega_k (t+\phi)) \end{bmatrix}$$

$$\begin{bmatrix} u_1 & v_1 \\ u_2 & v_2 \end{bmatrix} \begin{bmatrix} \sin(\omega_k t) \\ \cos(\omega_k t) \end{bmatrix} = \begin{bmatrix} \sin(\omega_k t)\cos(\omega_k \phi) + \cos(\omega_k t)\sin(\omega_k \phi) \\ \cos(\omega_k t)\cos(\omega_k \phi) - \sin(\omega_k t)\sin(\omega_k \phi) \end{bmatrix}$$

$$M_{\phi,k} = \begin{bmatrix} \cos(\omega_k \phi) & \sin(\omega_k \phi) \\ -\sin(\omega_k \phi) & \cos(\omega_k \phi) \end{bmatrix}$$

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

Positional Encoding

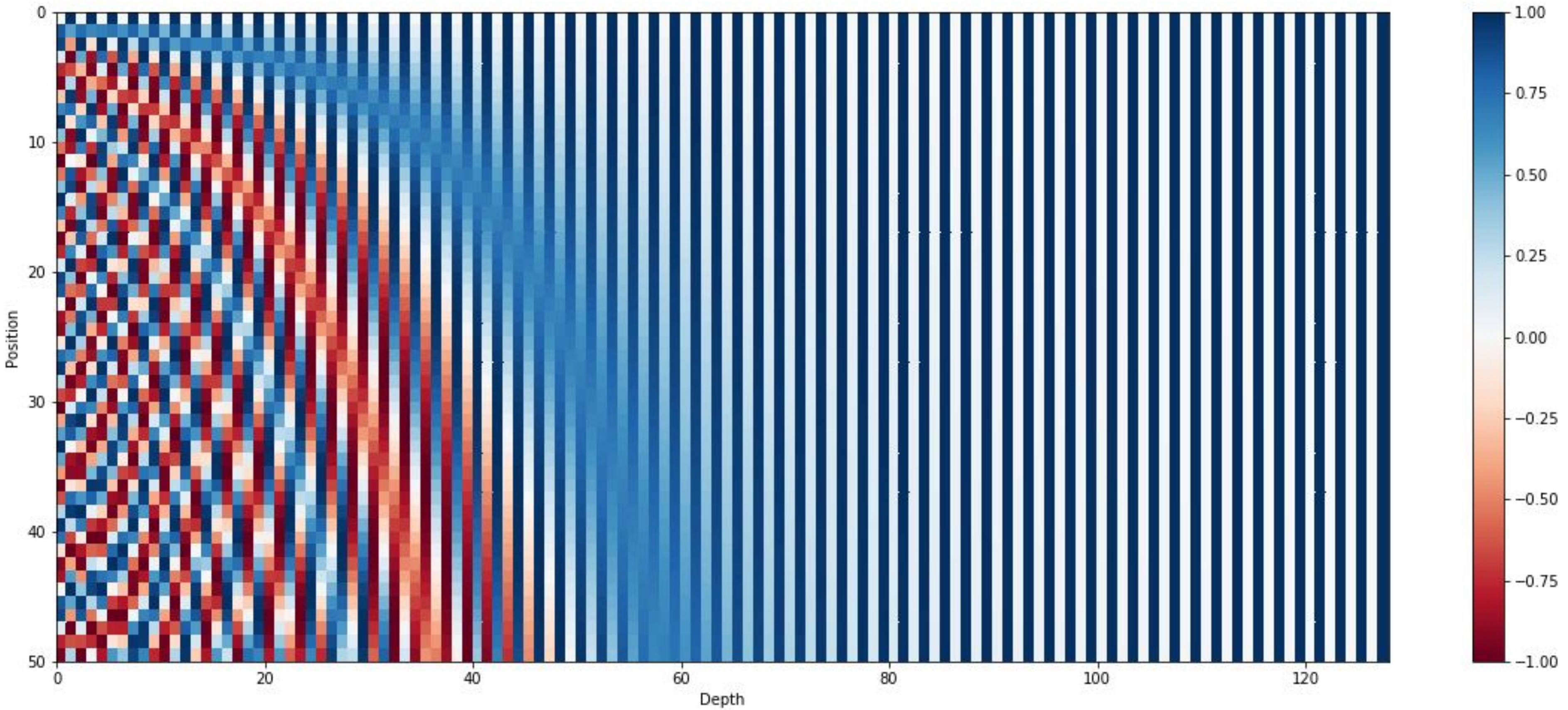


Image source: https://kazemnejad.com/blog/transformer_architecture_positional_encoding/

ELMo: context that matters

ELMo: contextualized word embeddings

"Why not give it an embedding based on the context it's used in – to both capture the word meaning in that context as well as other contextual information?"

Peters et. al., 2017, McCann et. al., 2017,
and yet again Peters et. al., 2018 in the ELMo paper

ELMo - deep contextualized word representations

ELMo

What does it stand for?

1. **E**xpedited **L**abour **M**arket **O**pinion
2. **E**lectric **L**ight **M**achine **O**rganization
3. **E**nough **L**et’s **M**ove **O**n

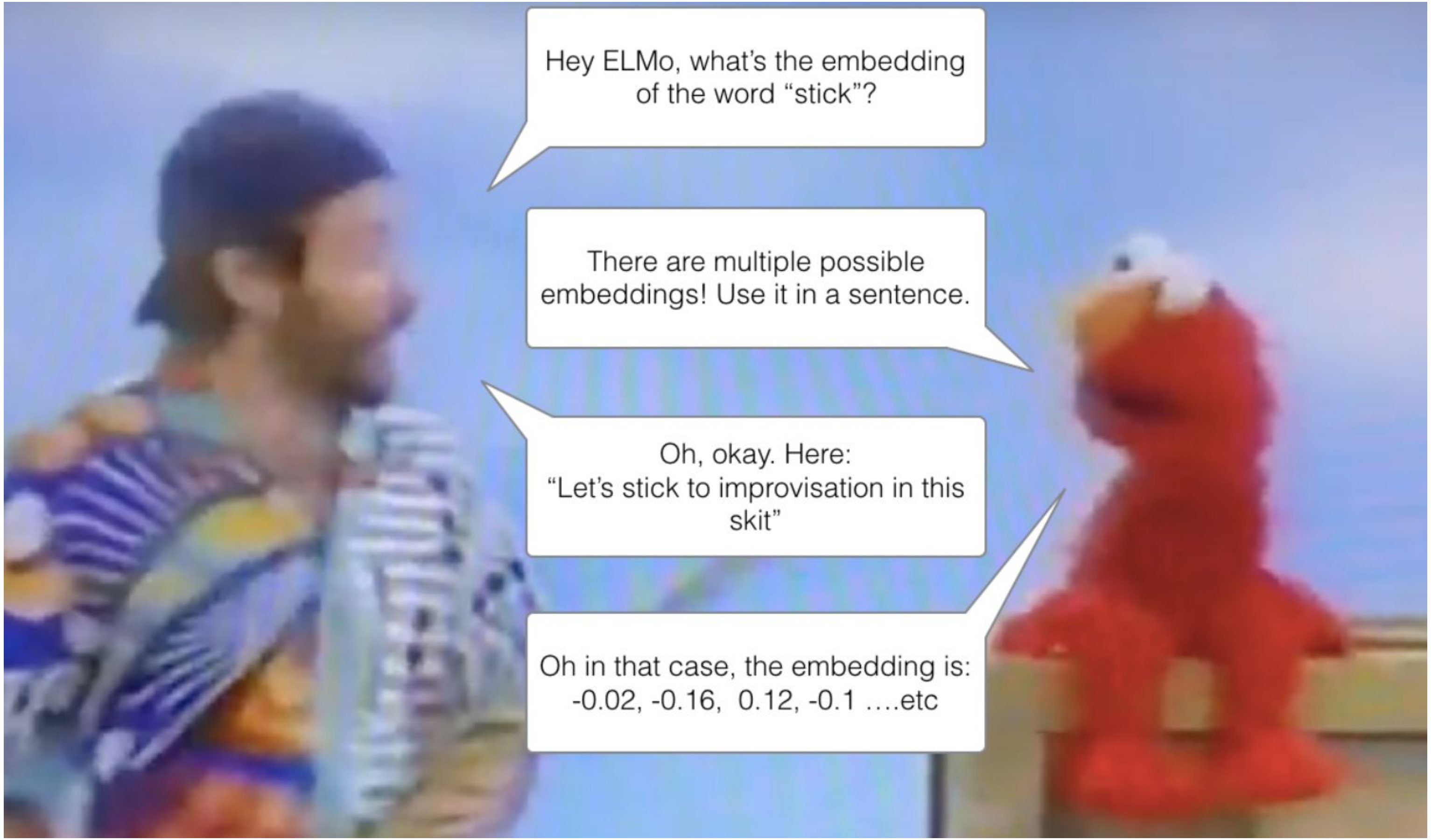
ELMo

What does it stand for?

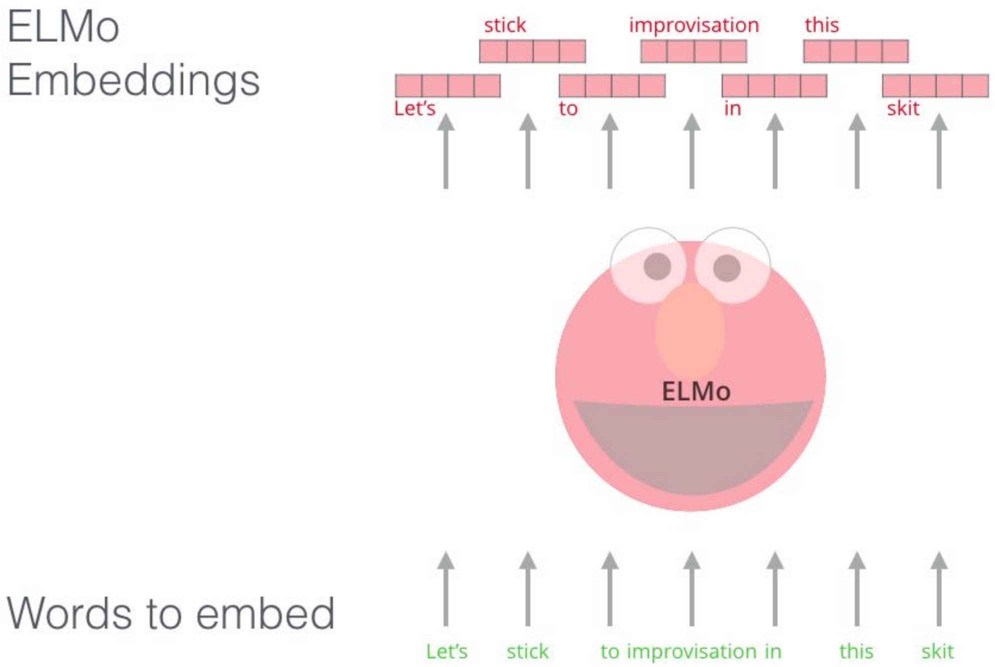
1. **E**xpedited **L**abour **M**arket **O**pinion
2. **E**lectric **L**ight **M**achine **O**rganization
3. **E**nough **L**et’s **M**ove **O**n

4. **Embeddings from Language Models**

ELMo: contextualized word embeddings

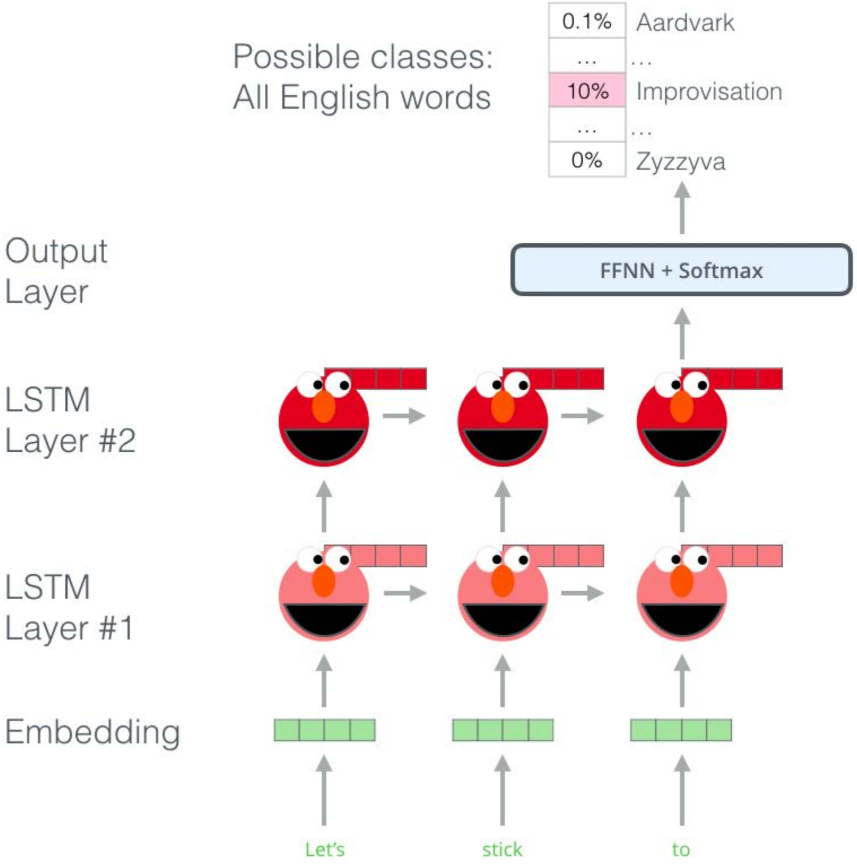


ELMo: Contextualized word embeddings



- uses a bi-directional LSTM trained on Language Modeling task
- a model can learn without labels

Bidirectional Language Models (biLMs)



biLMs consist of forward and backward LMs:

- forward:

$$p(t_1, t_2, \ldots, t_N) = \prod_{k=1}^{N} p(t_k \mid t_1, t_2, \ldots, t_{k-1})$$

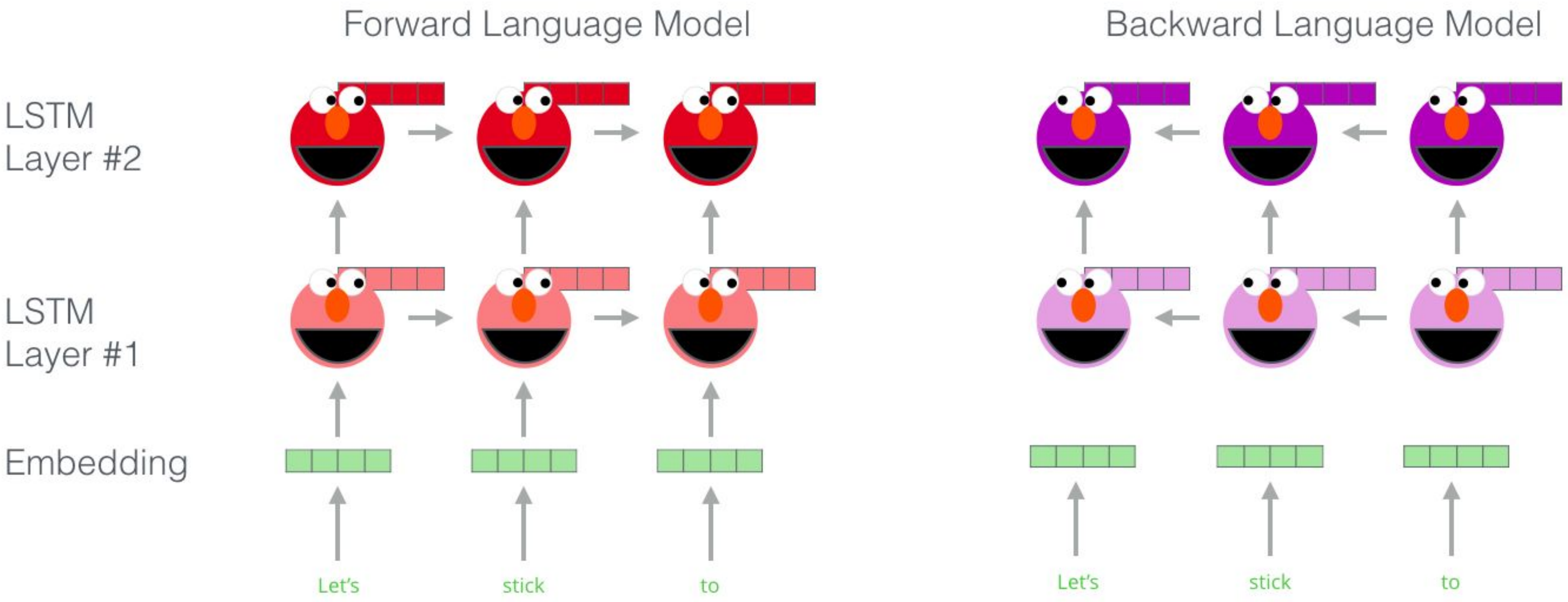
- Backward:

$$p(t_1, t_2, \ldots, t_N) = \prod_{k=1}^{N} p(t_k \mid t_{k+1}, t_{k+2}, \ldots, t_N)$$

LSTM predicts next word in both directions to build biLMs

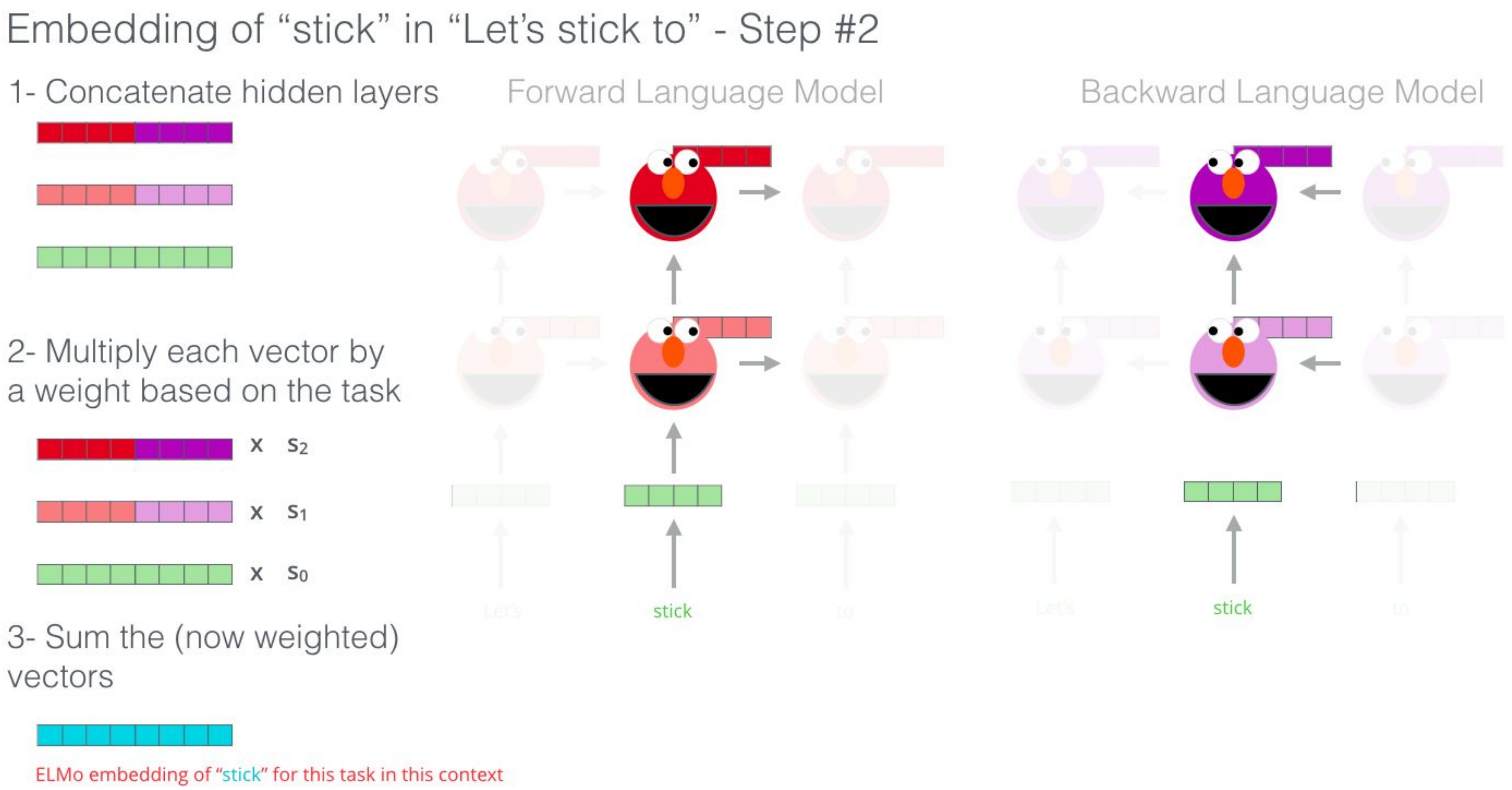
ELMo: main pipeline

Embedding of “stick” in “Let’s stick to” - Step #1



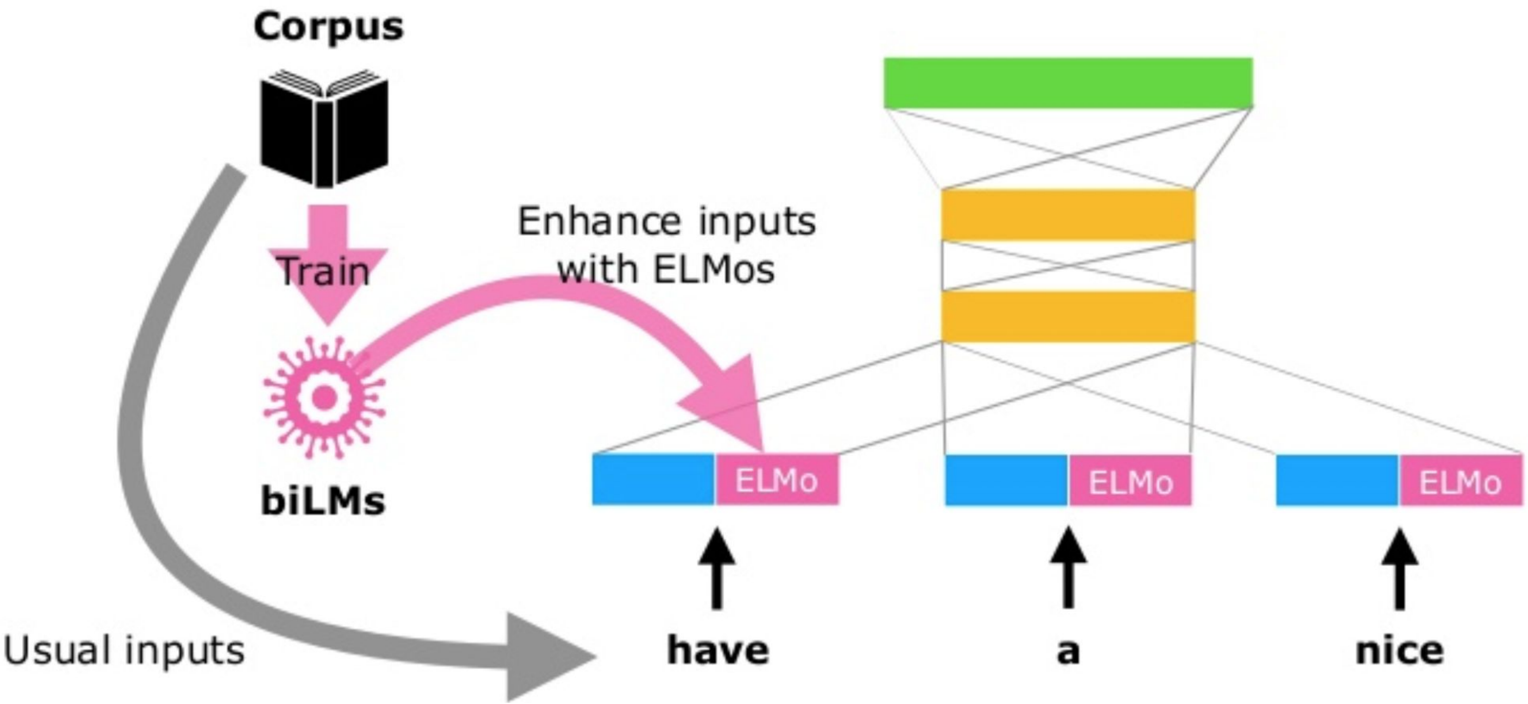
ELMo: main pipeline

ELMo represents a word as a linear combination of corresponding hidden layers:



ELMo

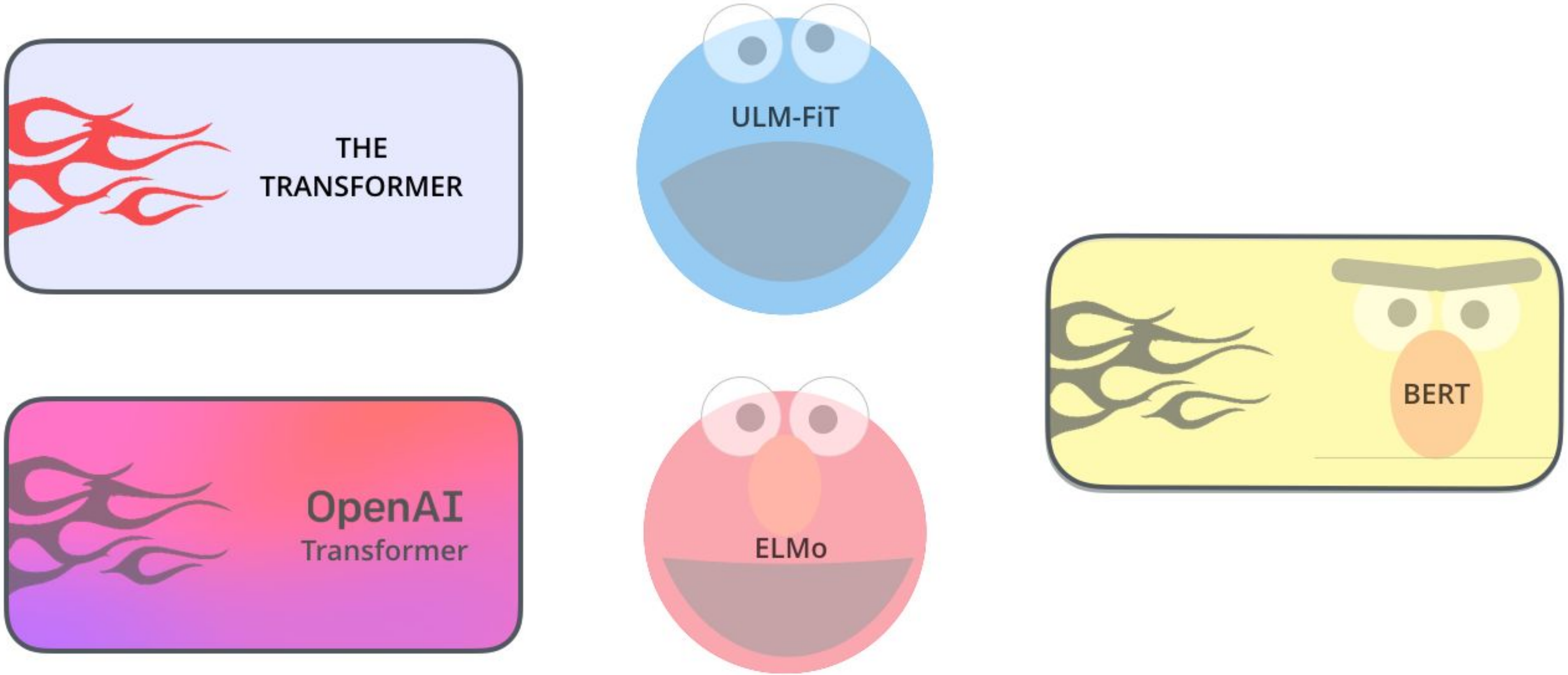
ELMo can be integrated to almost all neural NLP tasks with simple concatenation to the embedding layer



ELMo: overview

- Pretrained ELMo models: http://allennlp.org/elmo
- AllenNLP is a library on the top of PyTorch

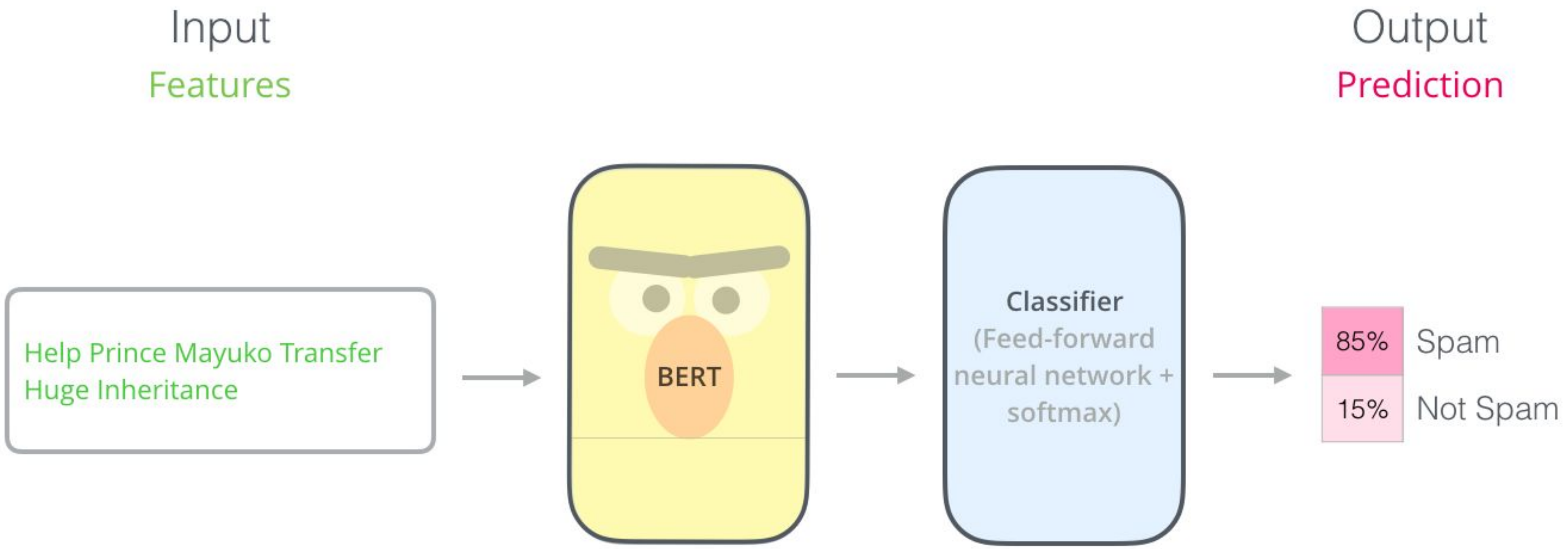
- Higher levels seems to catch semantics while lower layer probably capture syntactic features


THE
TRANSFORMER
ULM-FiT
BERT
OpenAI
Transformer
ELMo

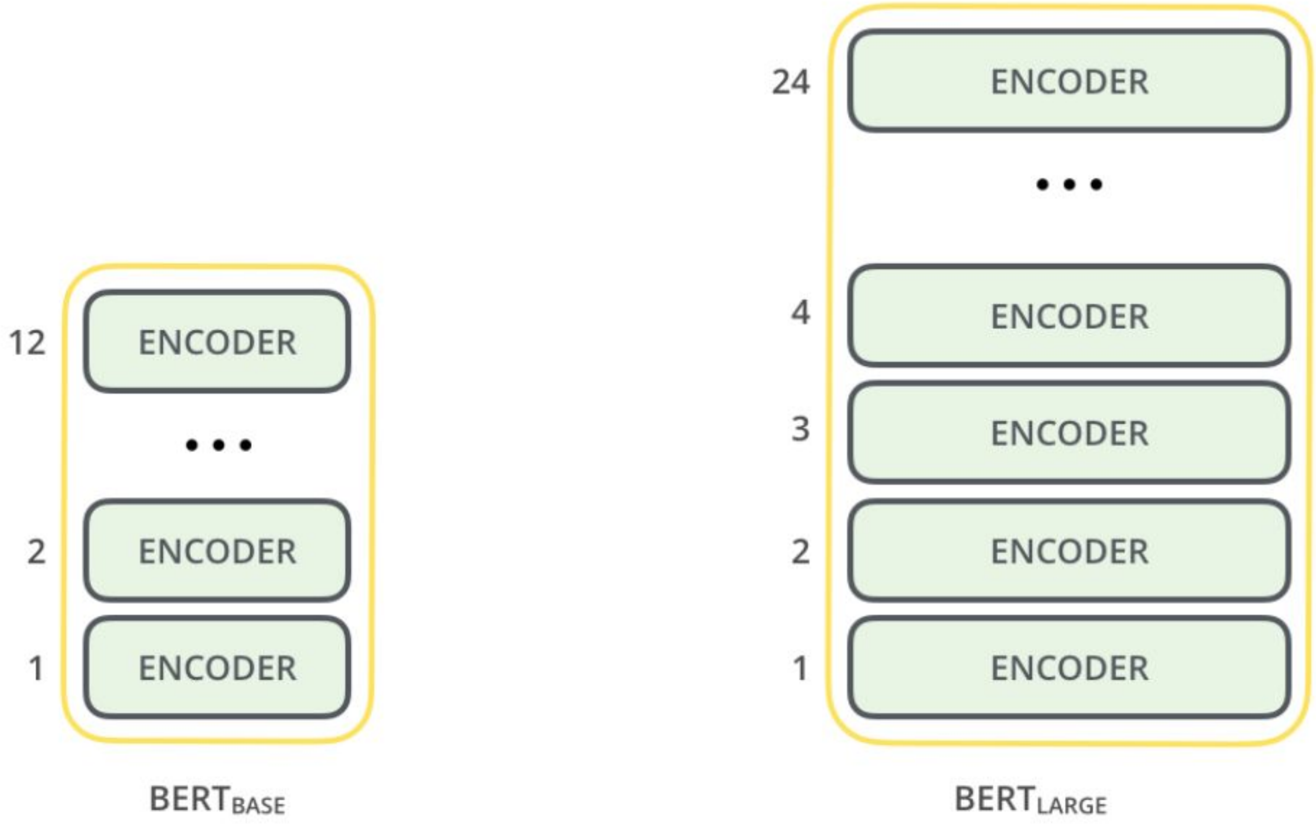
BERT

Bidirectional Encoder Representations from Transformers

BERT


Input
Features
Help Prince Mayuko Transfer Huge Inheritance
BERT
Classifier
(Feed-forward neural network + softmax)
Output
Prediction
85% Spam
15% Not Spam

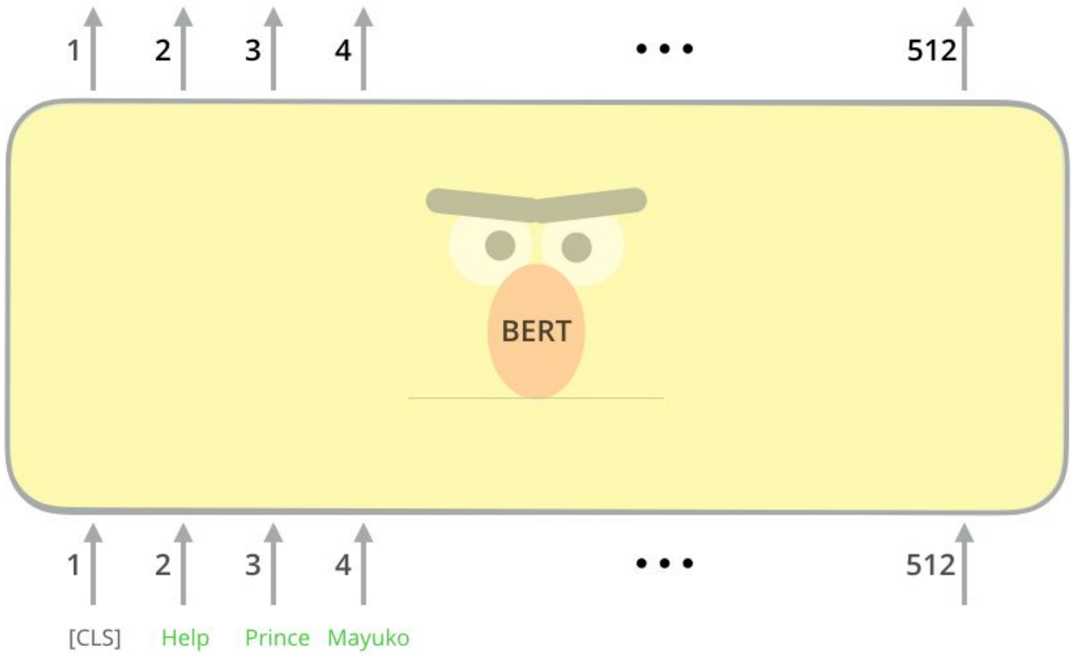
BERT: base and large



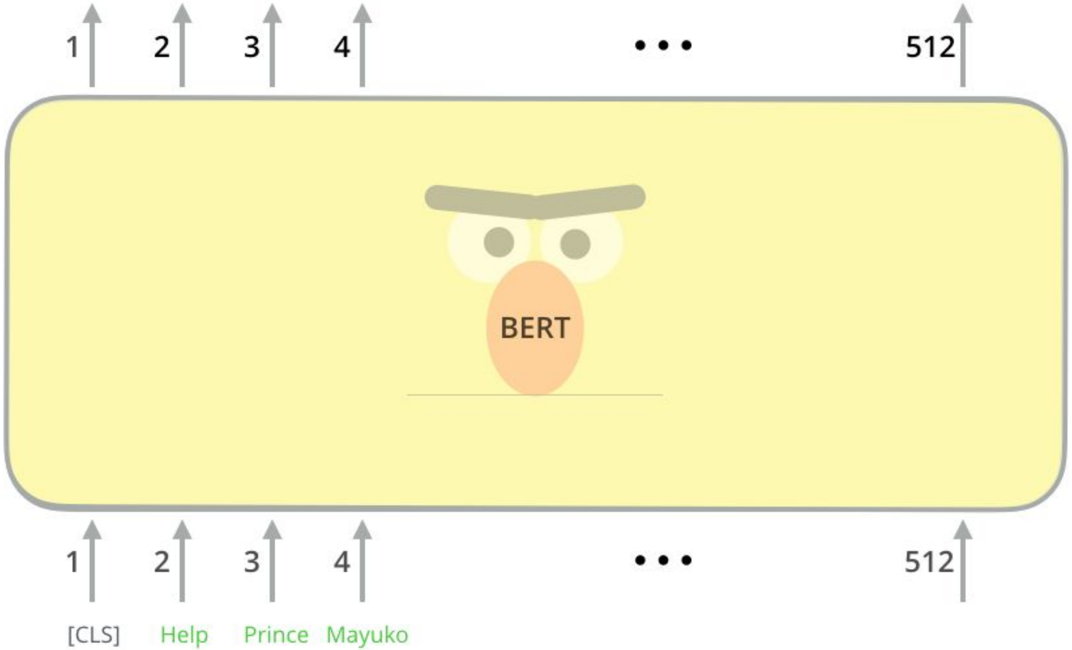
BERT vs. Transformer

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

Model inputs

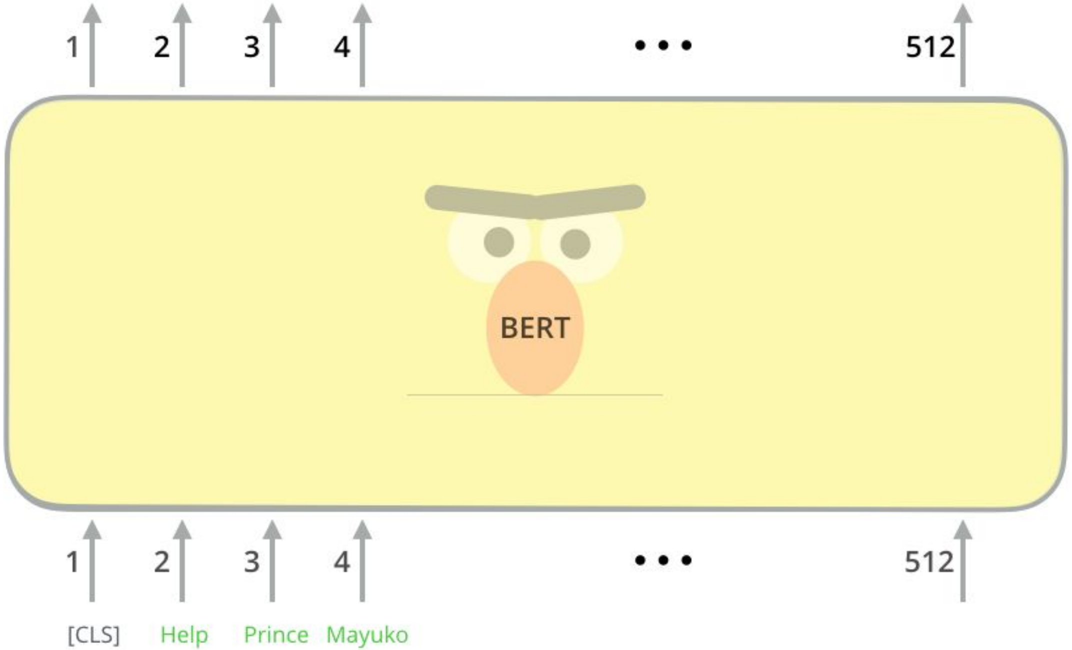


Model inputs



Identical to the Transformer up until this point

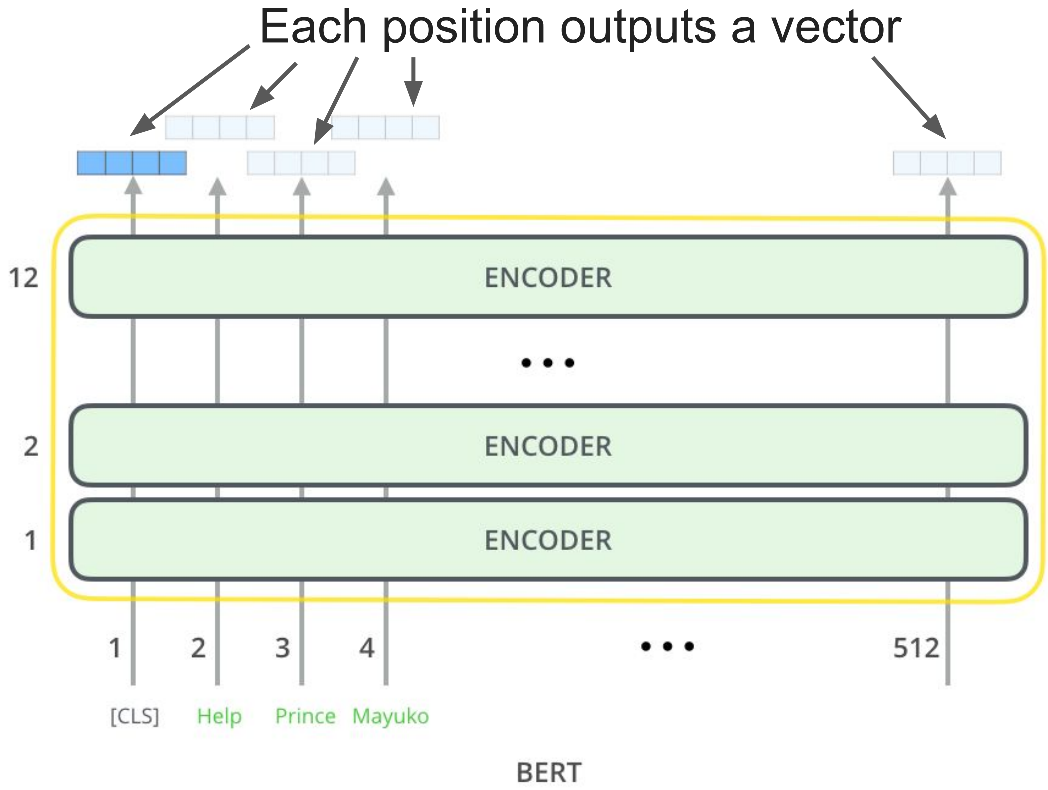
Model inputs



Identical to the Transformer up until this point

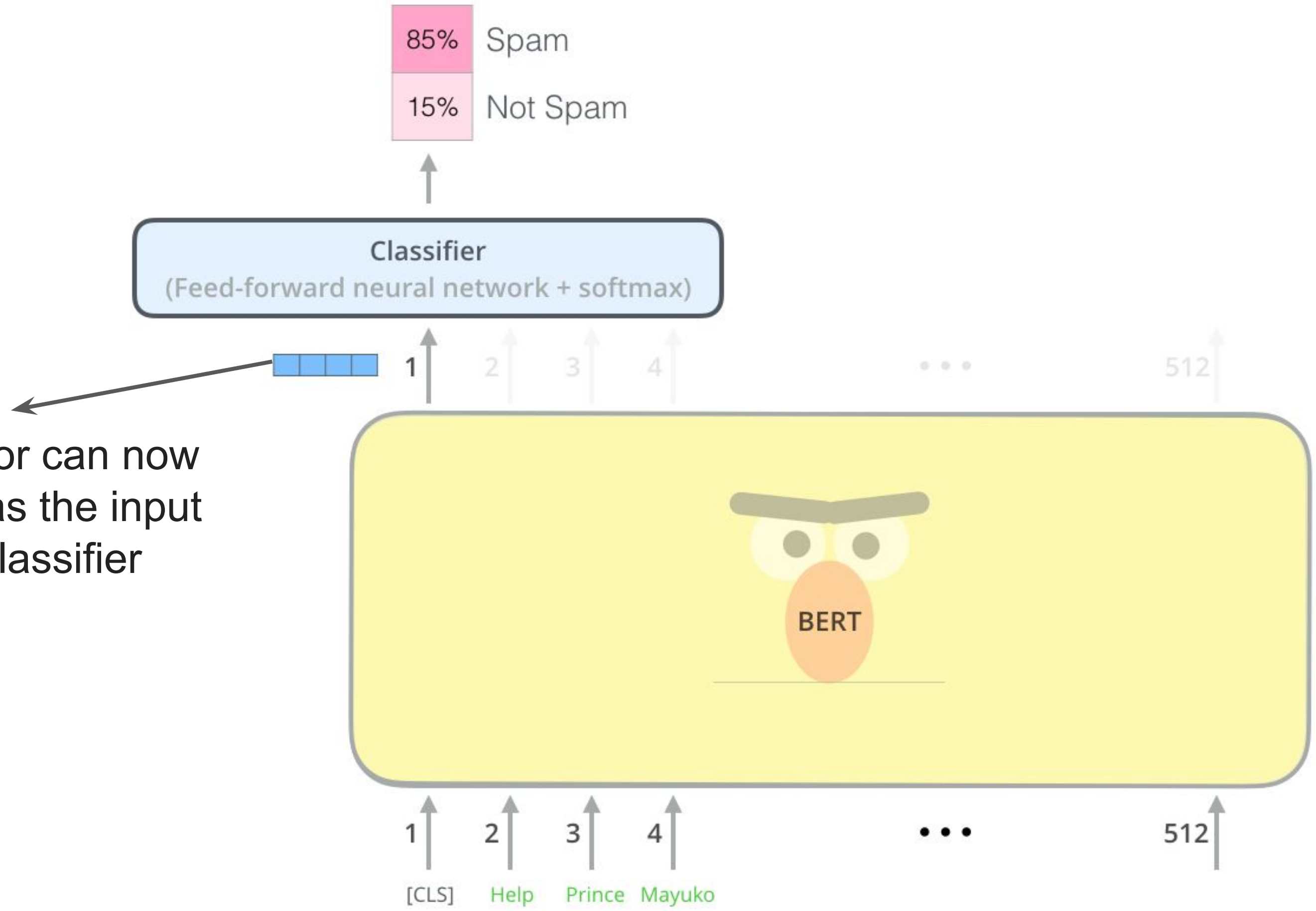
Why is BERT so special?

Model outputs



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

Model inputs

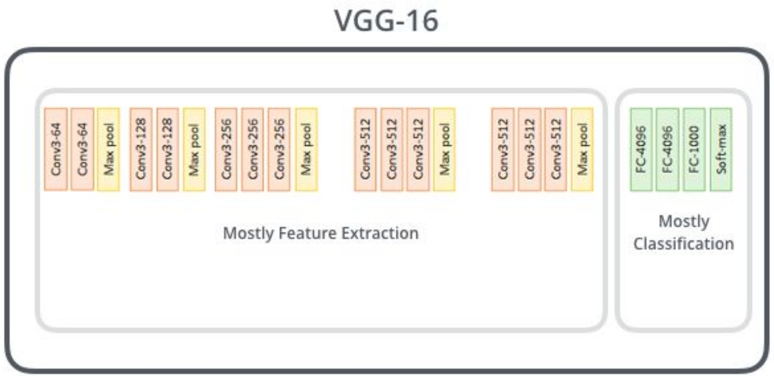


This vector can now be used as the input for a classifier

Similar to CNN concept!

Input

Features



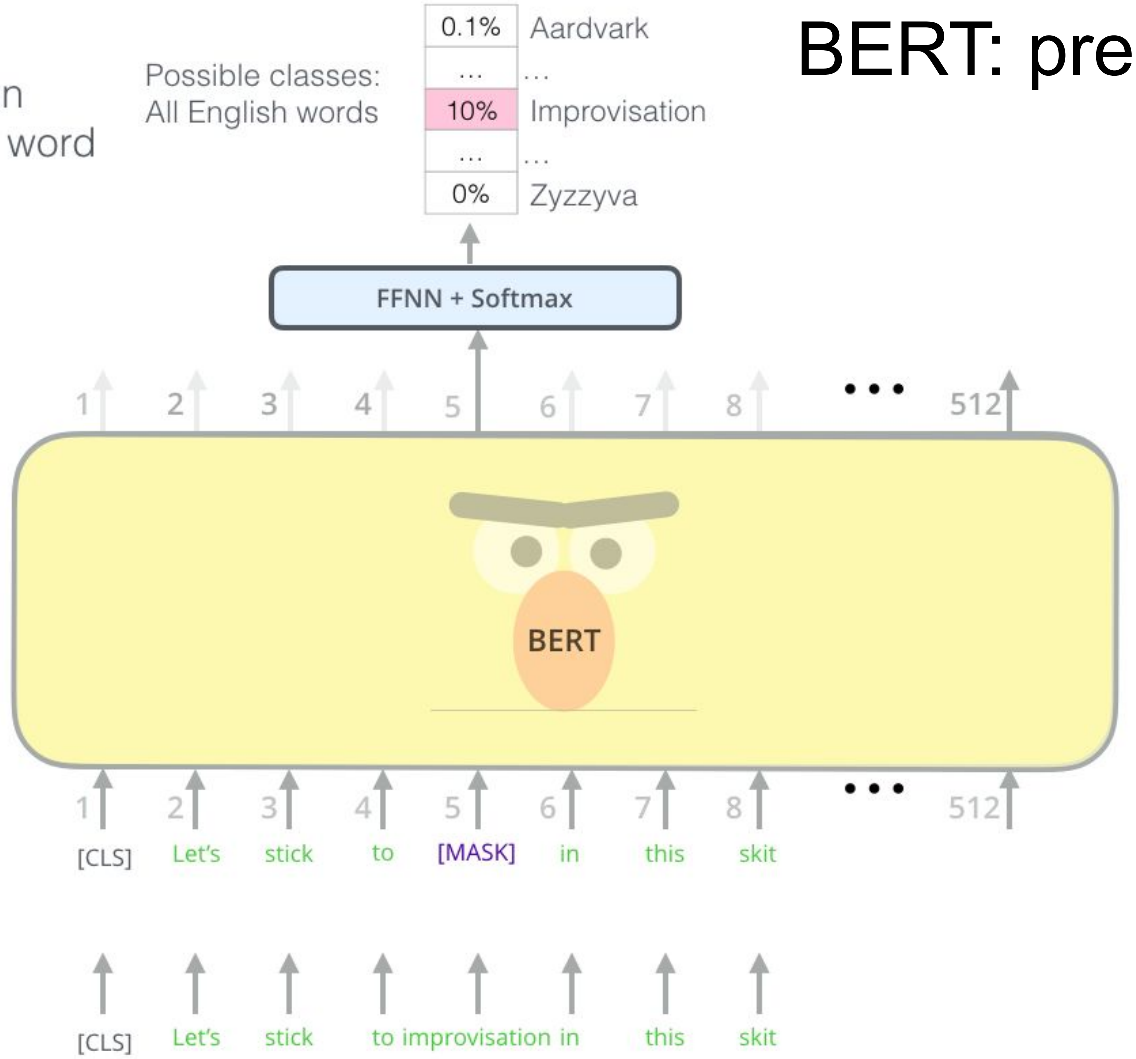
Output

Prediction

0.2%	Kit fox
0.1%	English setter
95%	Egyptian cat
1%	Great Dane
	...
0%	Hotdog

BERT: pre-training

Use the output of the masked word's position to predict the masked word



Randomly mask 15% of tokens

Input

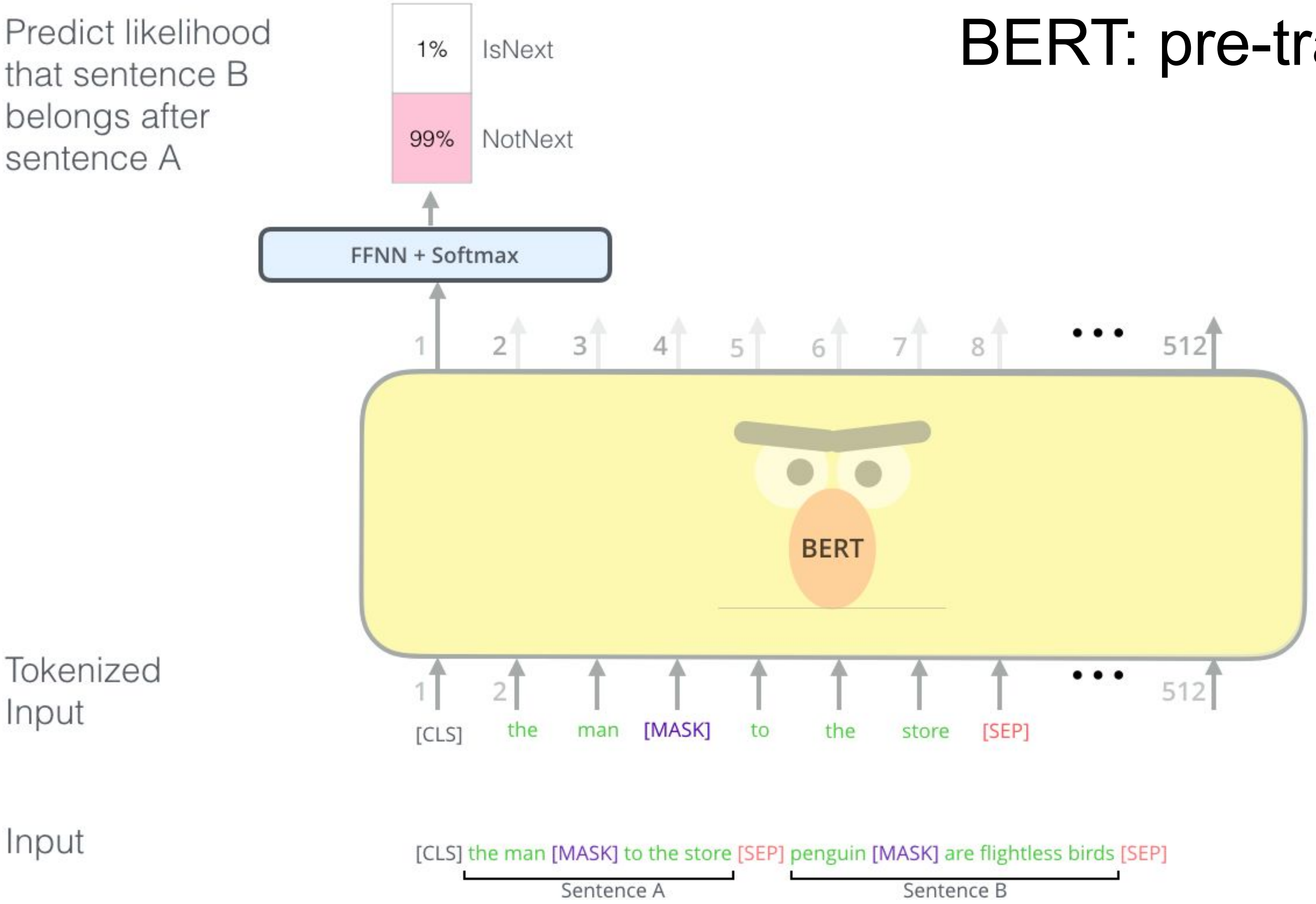
BERT: pre-training

- “Masked Language Model” approach

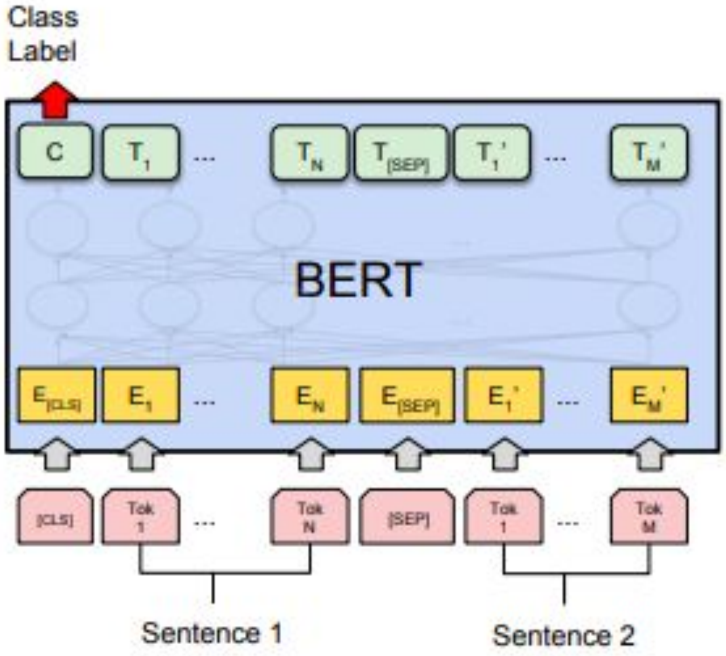
- To make BERT better at handling relationships between multiple sentences, the pre-training process includes an additional task:

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

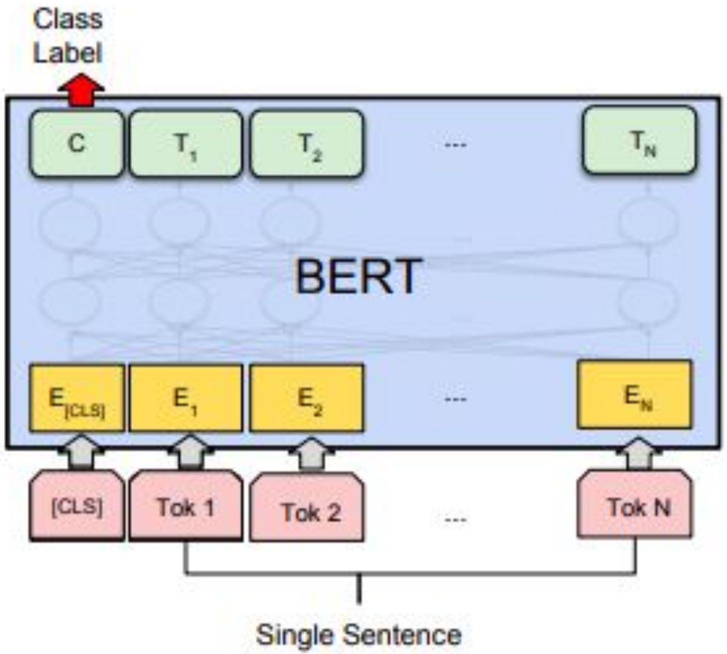
BERT: pre-training



BERT: fine-tuning for different tasks

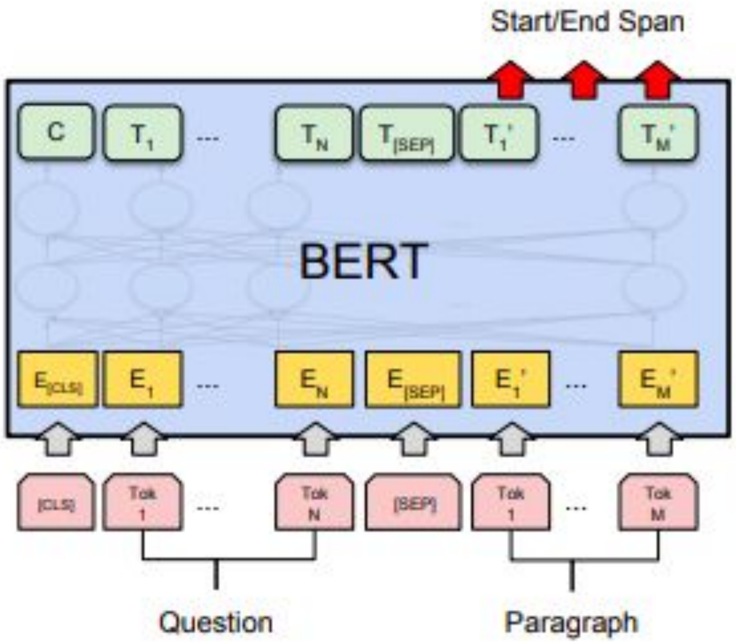


(a) Sentence Pair Classification Tasks: MNLI, QQP, QNLI, STS-B, MRPC, RTE, SWAG

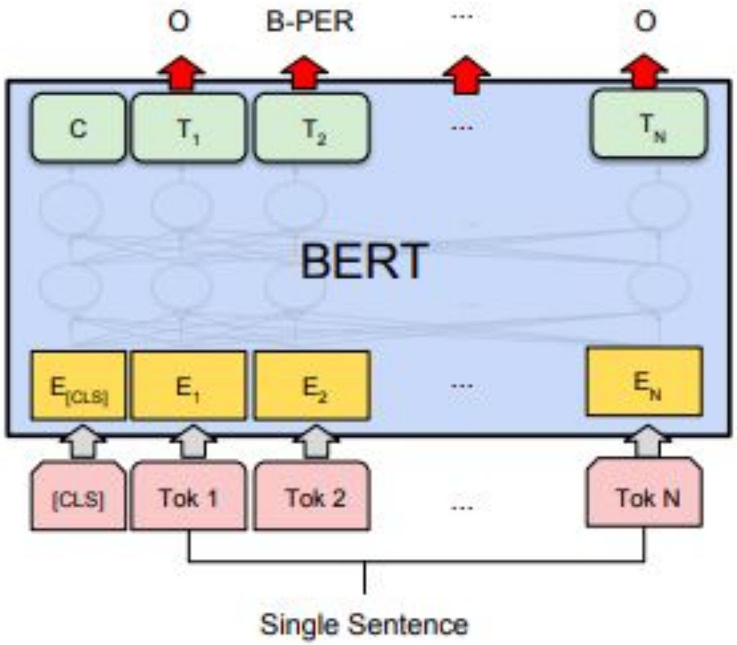


(b) Single Sentence Classification Tasks: SST-2, CoLA

BERT: fine-tuning for different tasks

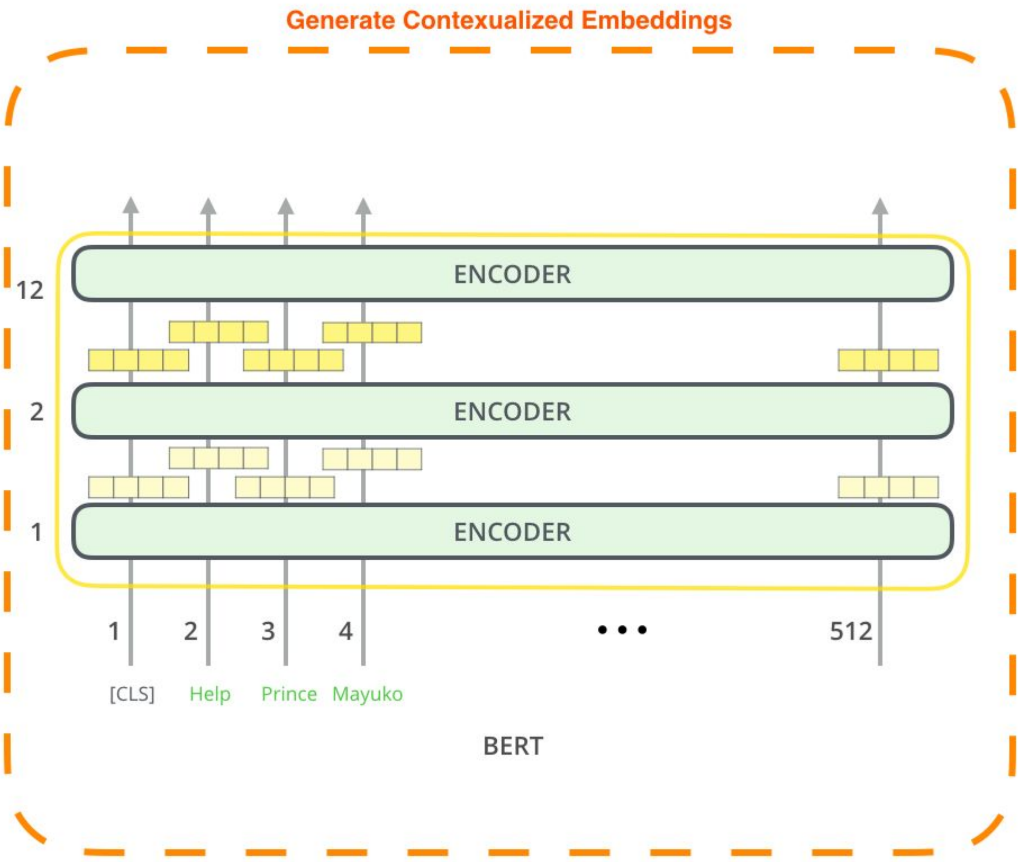


(c) Question Answering Tasks:
SQuAD v1.1

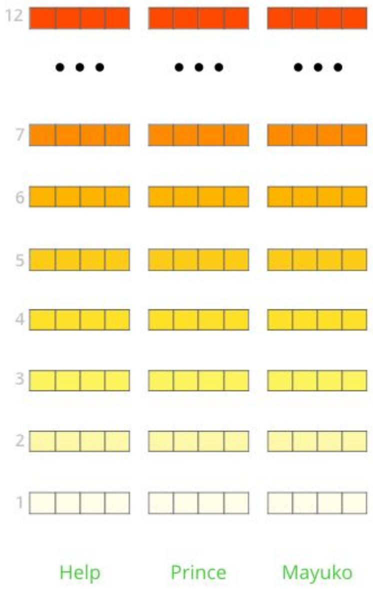


(d) Single Sentence Tagging Tasks:
CoNLL-2003 NER

BERT for feature extraction

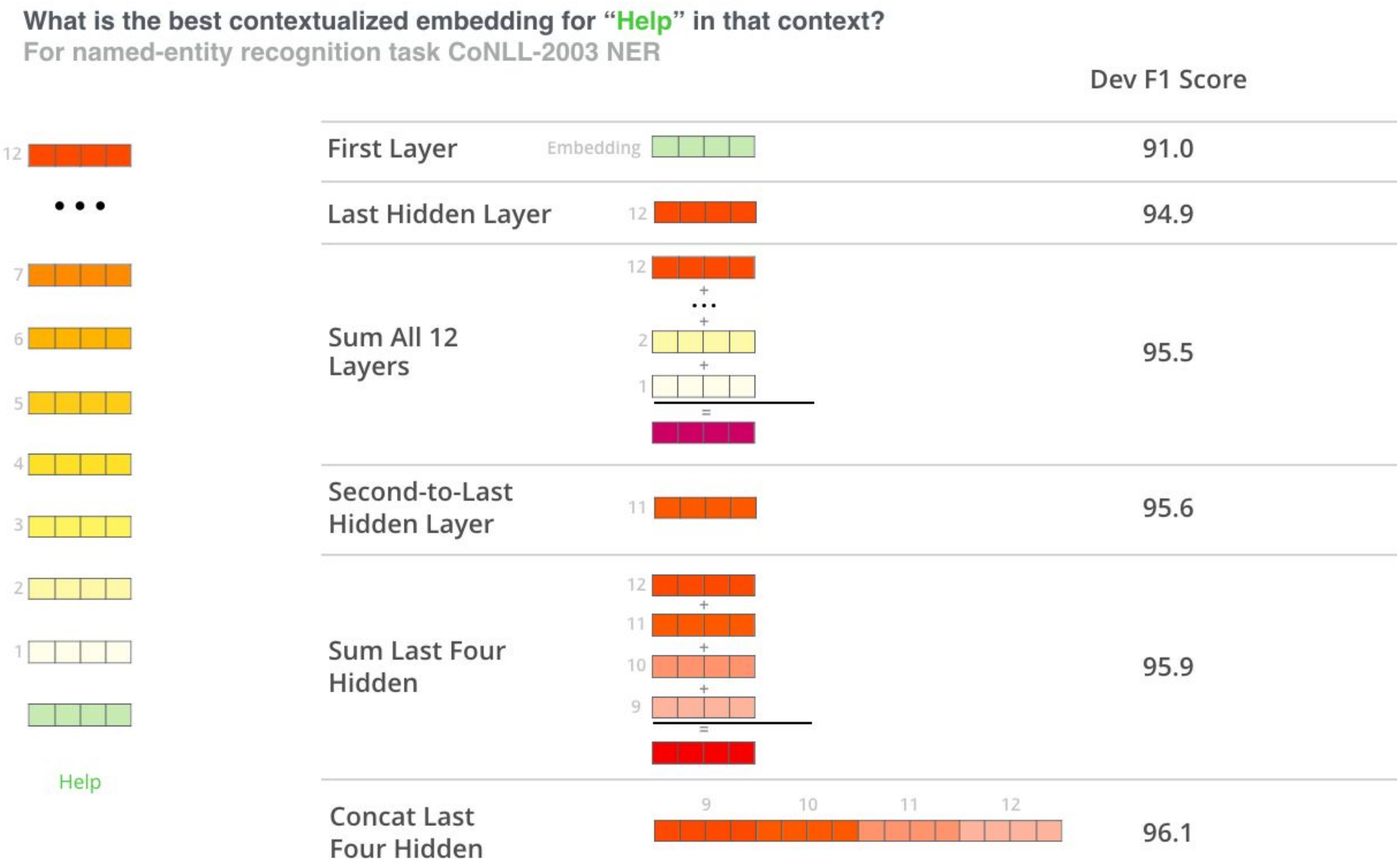


The output of each encoder layer along each token's path can be used as a feature representing that token.



But which one should we use?

BERT for feature extraction

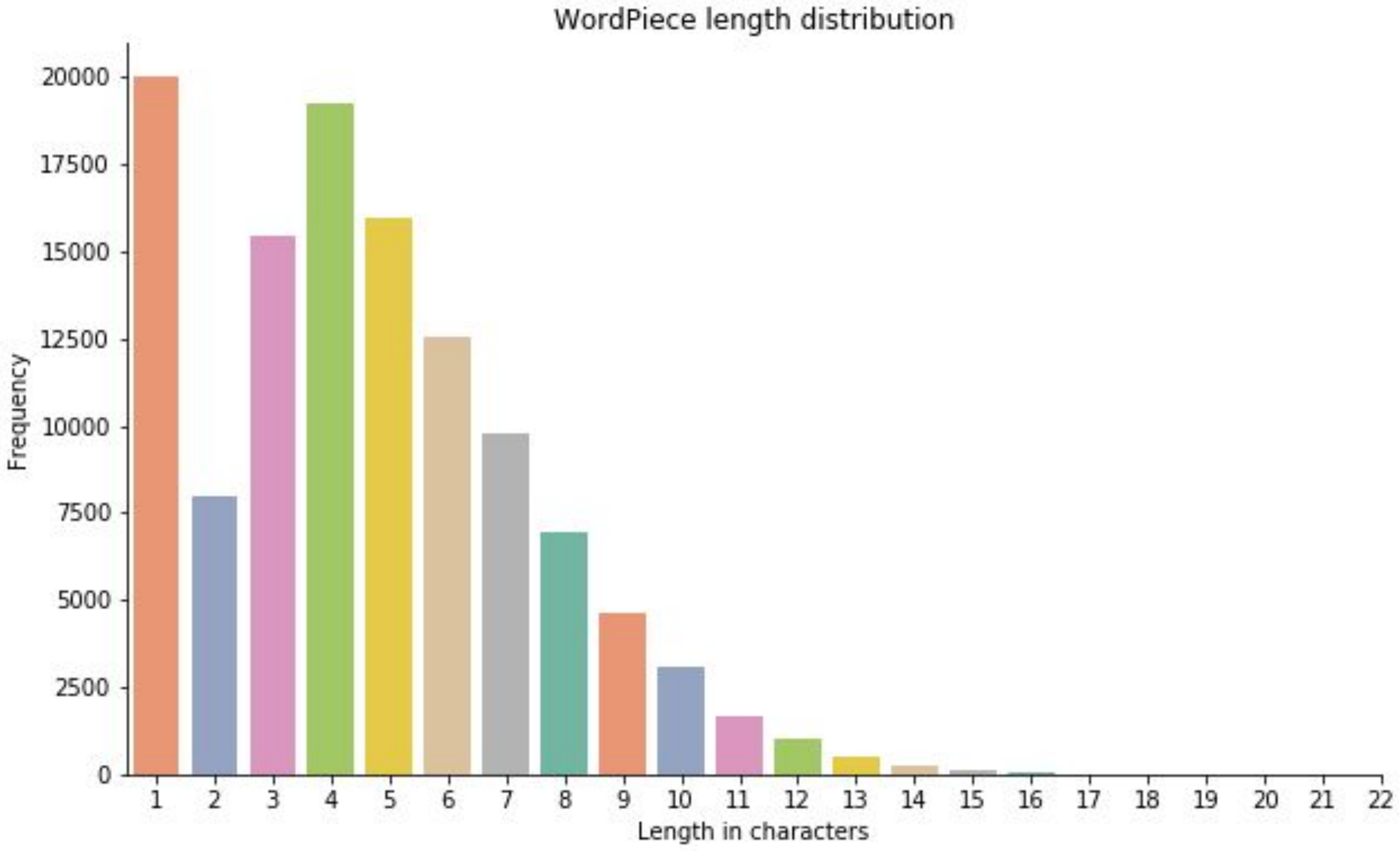


BERT: tokenization

<u>Example:</u> Unaffable -> un, ##aff, ##able

- Single model for 104 languages with a large shared vocabulary (119,547 WordPiece model)
- Non-word-initial units are prefixed with ##
- The first 106 symbols: constants like PAD and UNK
- 36.5% of the vocabulary are non-initial word pieces
- The alphabet consists of 9,997 unique characters that are defined as word-initial (C) and continuation symbols (##C), which together make up 19,994 word pieces
- The rest are multicharacter word pieces of various length.

BERT: tokenization

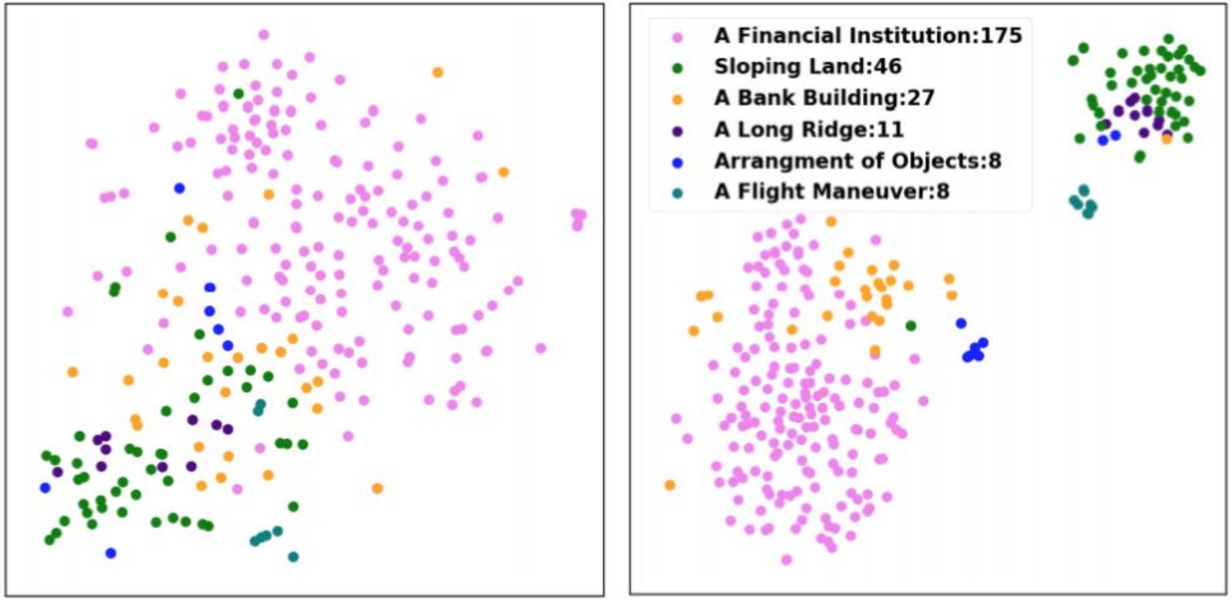


BERT: overview

- BERT repo
- Try out BERT on TPU
- WordPieces Tokenizer
- PyTorch Implementation of BERT

BERT vs. ELMO (Word Sense Disambiguation problem)

T-SNE plots of different senses of ‘bank’

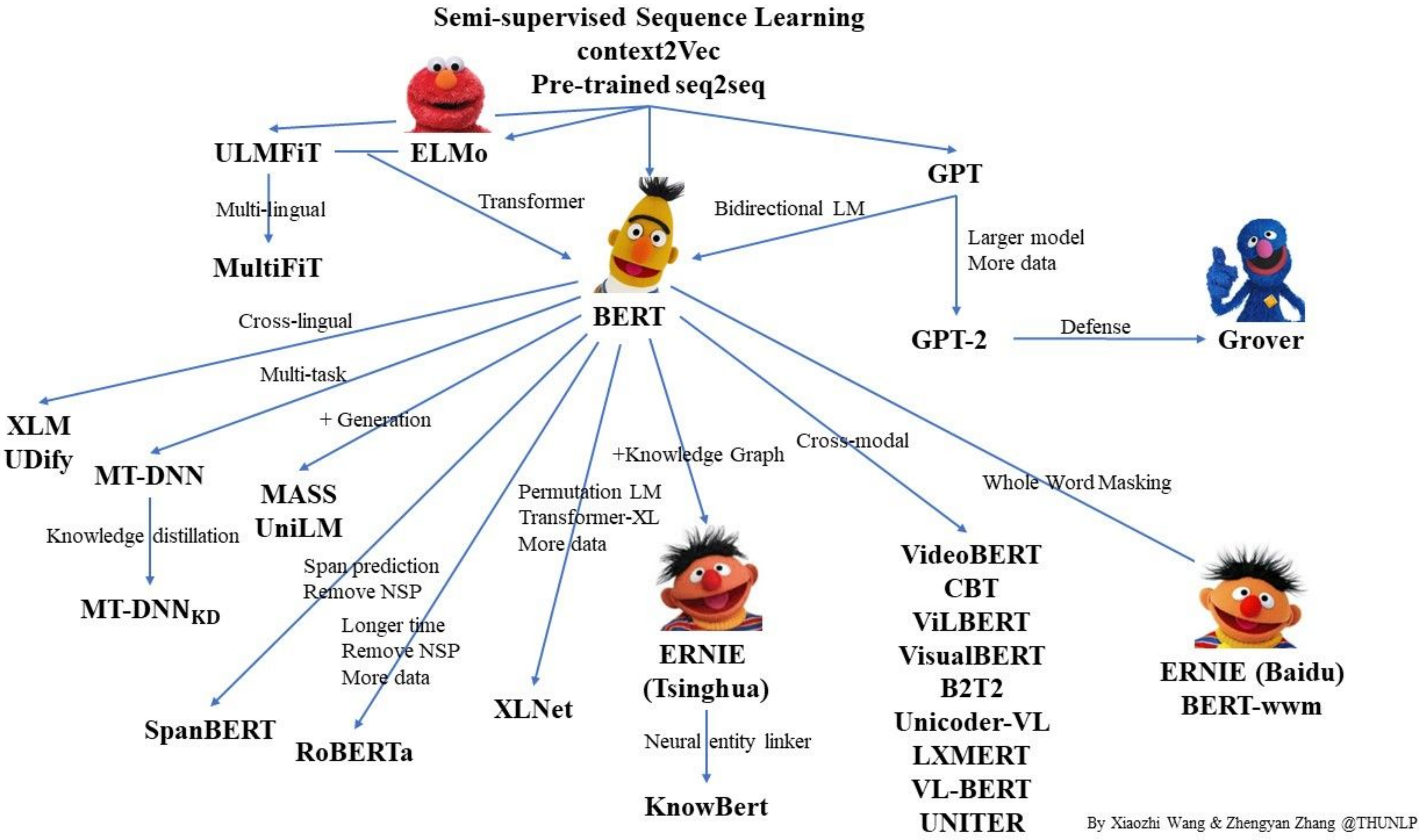


(c) ELMo (a) BERT

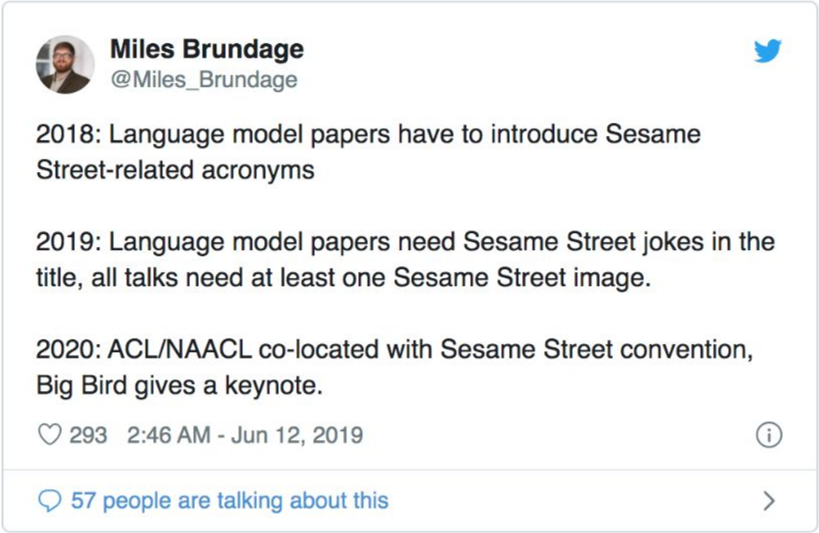
BERTology

BERTology paper

BERTology



BERTology



Miles Brundage
@Miles_Brundage

2018: Language model papers have to introduce Sesame Street-related acronyms

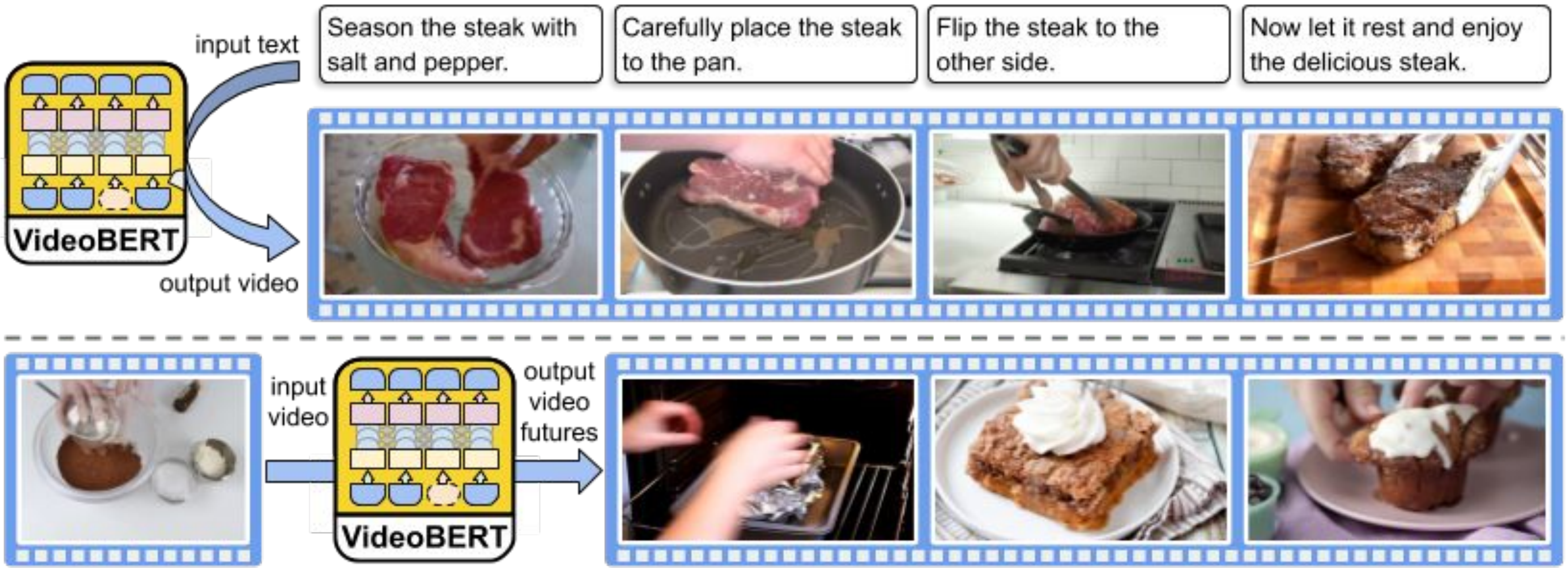
2019: Language model papers need Sesame Street jokes in the title, all talks need at least one Sesame Street image.

2020: ACL/NAACL co-located with Sesame Street convention, Big Bird gives a keynote.

293 2:46 AM - Jun 12, 2019

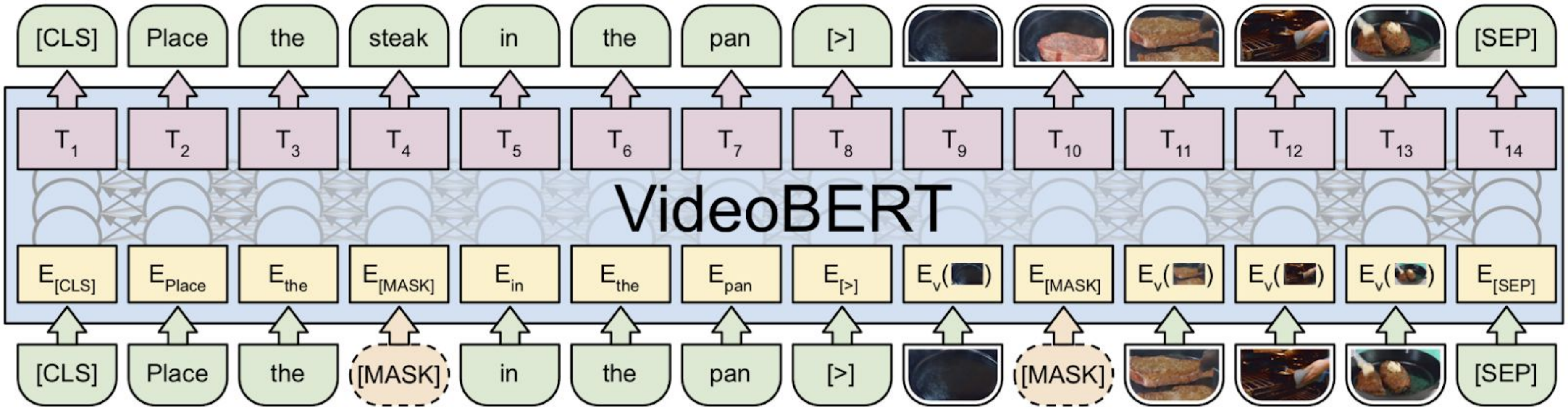
57 people are talking about this

VideoBERT (ICCV 2019)



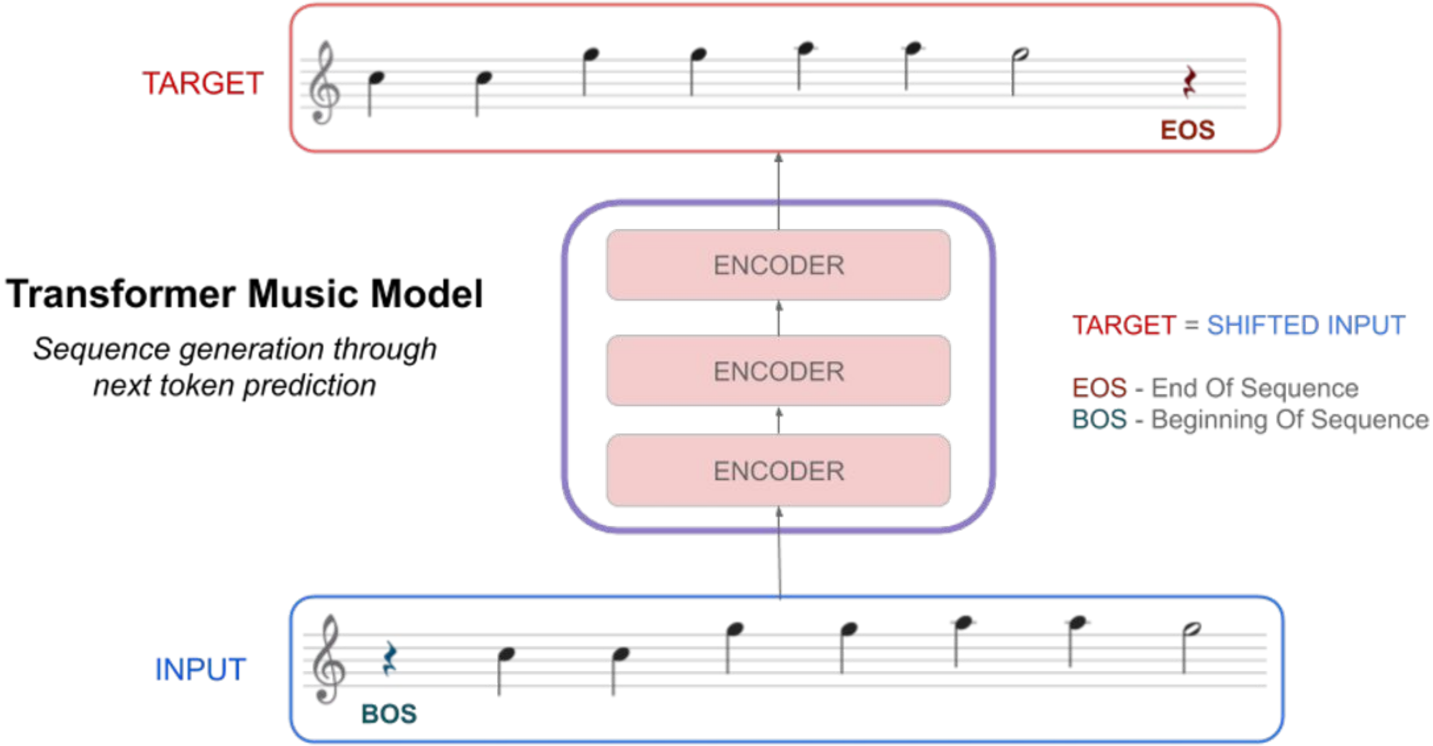
https://arxiv.org/abs/1904.01766
https://ai.googleblog.com/2019/09/learning-cross-modal-temporal.html

VideoBERT (ICCV 2019)



https://arxiv.org/abs/1904.01766
https://ai.googleblog.com/2019/09/learning-cross-modal-temporal.html

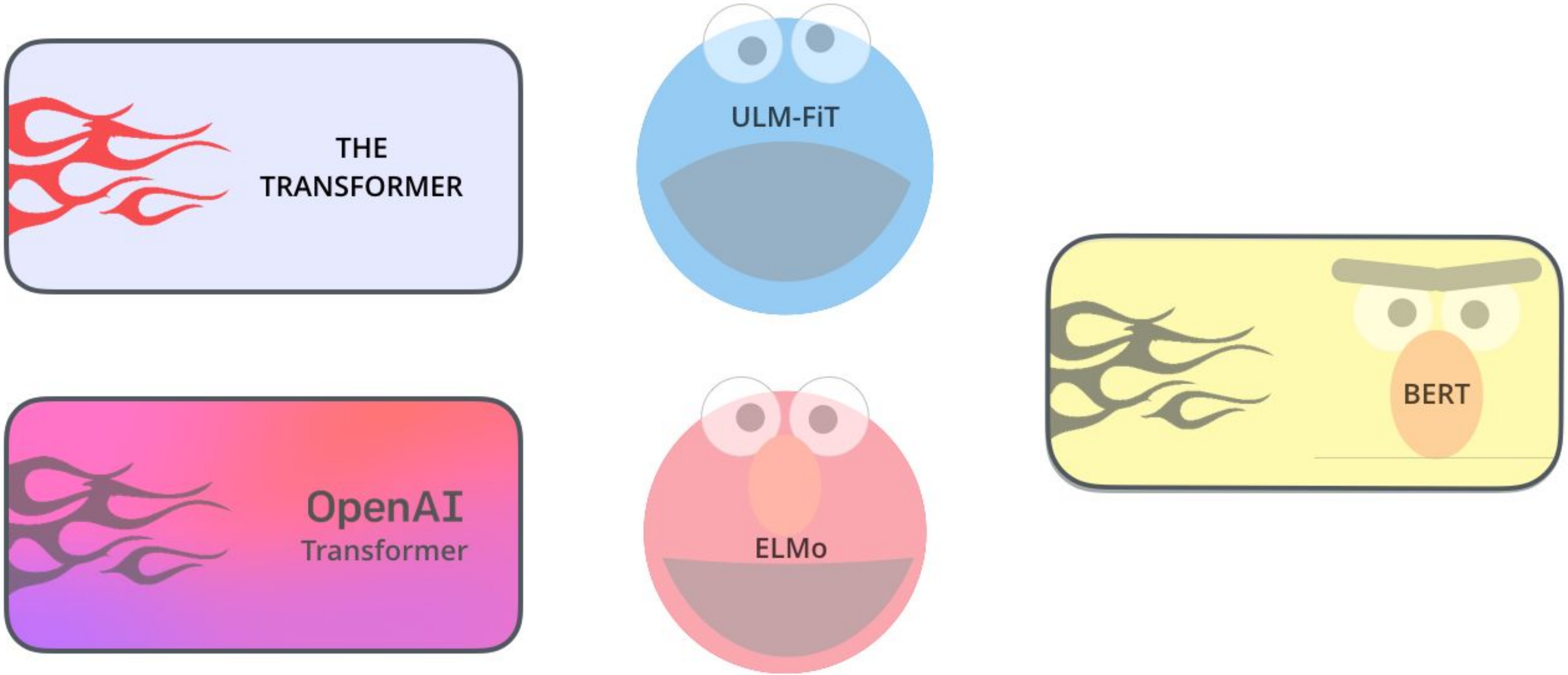
Music Transformer (ICLR 2019)



https://arxiv.org/abs/1809.04281
https://magenta.tensorflow.org/music-transformer

Fun demos to play with

[Get a neural network to autocomplete your thoughts]
[And yet another auto-completion tool]

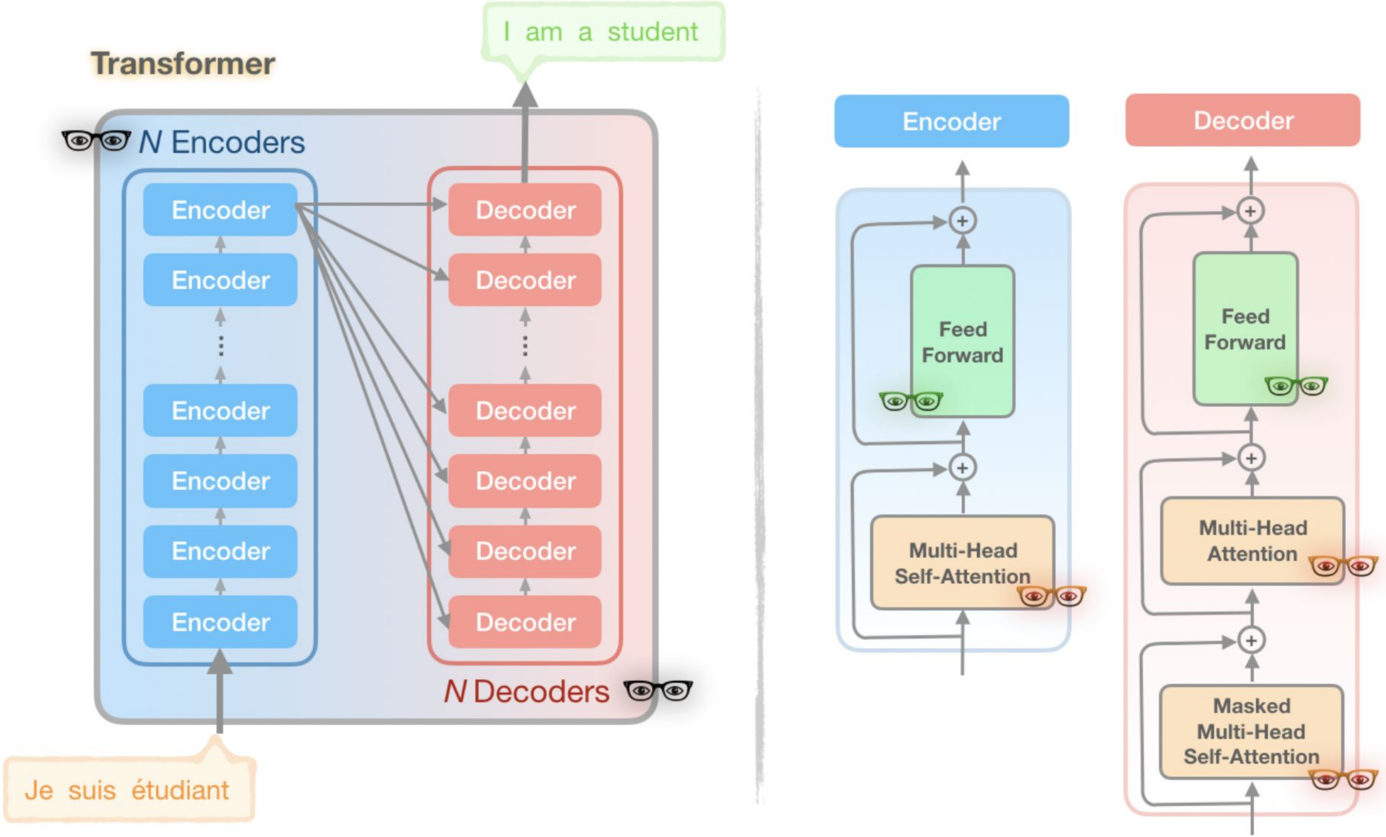

THE
TRANSFORMER
OpenAI
Transformer
ULM-FiT
ELMo
BERT

The Reformer

Reformer: The Effective Transformer (ICLR 2020)

Transformer-based models have a problem:

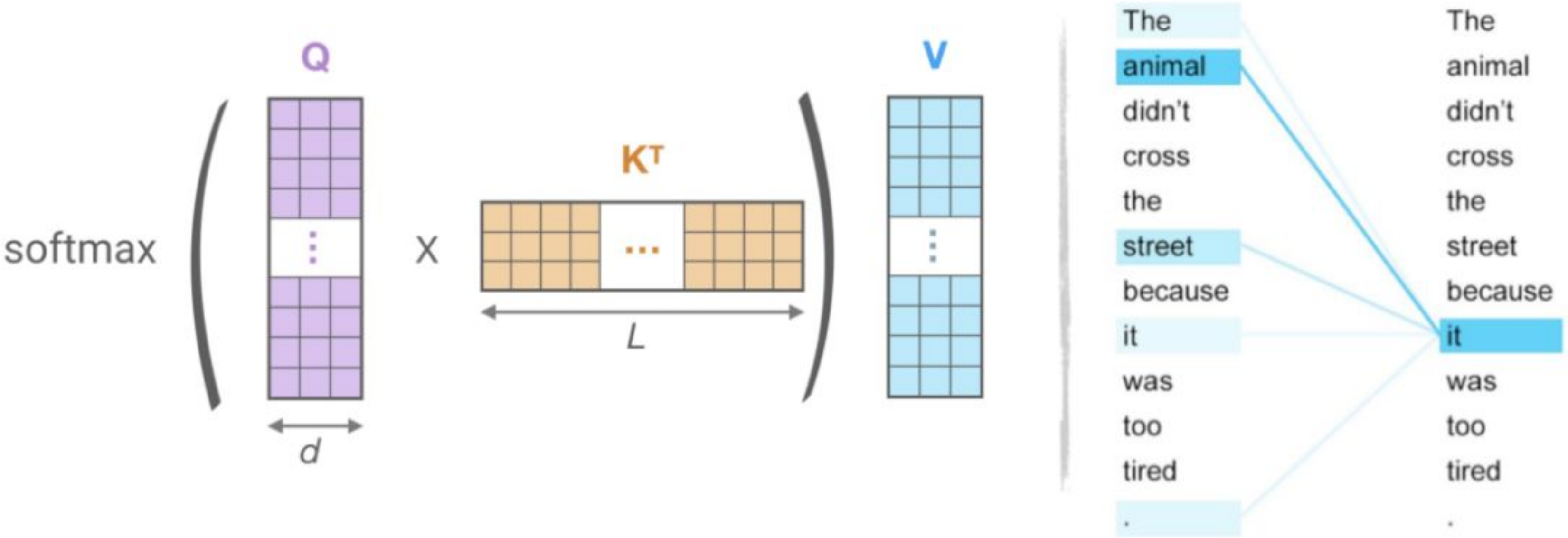
- They require lots of GPUs to train
  - even cannot be fine-tuned on a single GPU



- Problem 1 (**Red** 👓): Attention computation
- Problem 2 (**Black** 👓): Large number of layers
- Problem 3 (**Green** 👓): Depth of feed-forward layers

https://towardsdatascience.com/illustrating-the-reformer-393575ac6ba0

Attention computation



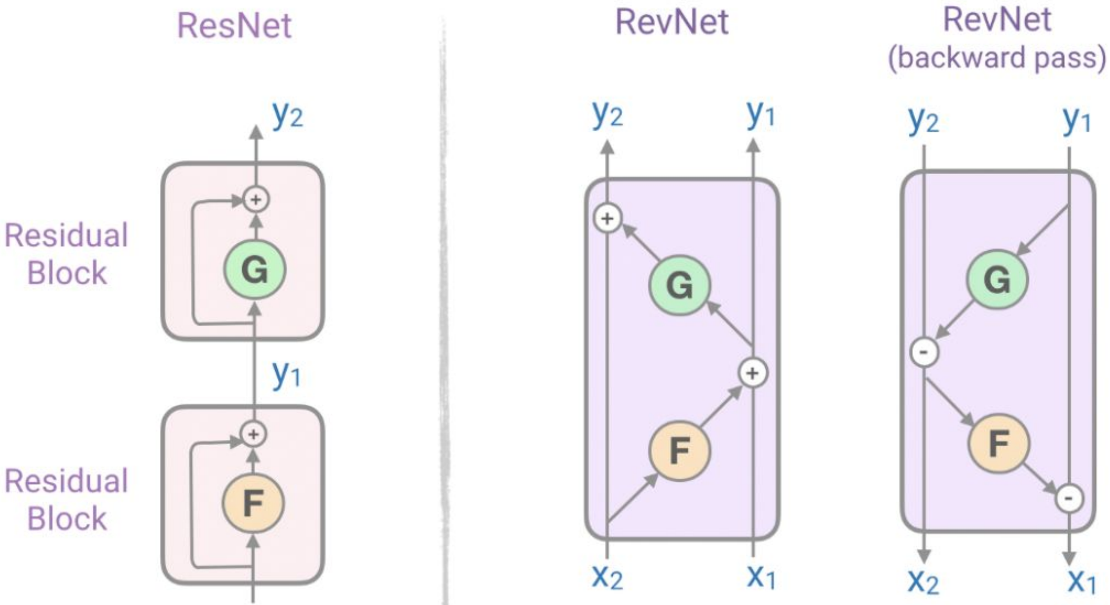
- Replace dot-product attention with locality-sensitive hashing (LSH)
  - changes the complexity from $O(L^2)$ to $O(L \log L)$

https://towardsdatascience.com/illustrating-the-reformer-393575ac6ba0

Locality-sensitive hashing for attention computation

- LSH - an *<u>efficient</u>* and *<u>approximate</u>* way of nearest neighbors search in high dimensional datasets.
- The main idea behind LSH is to select hash functions such that for two points ‘p’ and ‘q’, if ‘q’ is close to ‘p’ then with good enough probability we have ‘hash(q) == hash(p)’.

Reversible Transformer



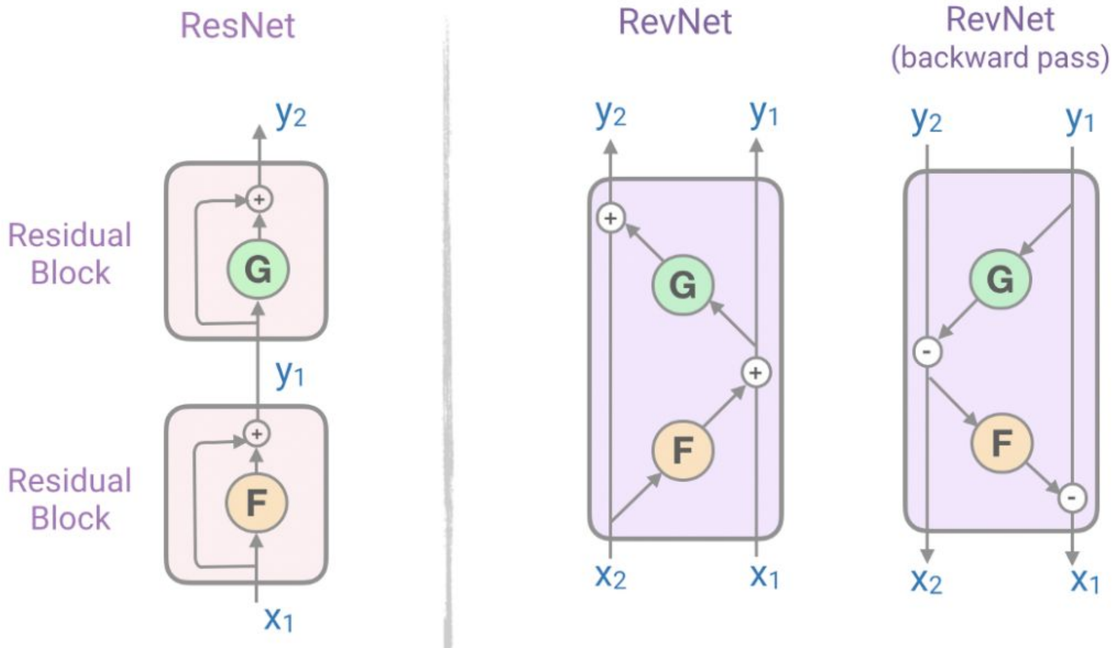
$$y_1 = x_1 + F(x_1)$$
$$y_2 = y_1 + G(y_1)$$

$$y_1 = x_1 + F(x_2)$$
$$y_2 = x_2 + G(y_1)$$

$$x_2 = y_2 - G(y_1)$$
$$x_1 = y_1 - F(x_2)$$

https://towardsdatascience.com/illustrating-the-reformer-393575ac6ba0

Reversible Transformer



$$y_1 = x_1 + F(x_1)$$
$$y_2 = y_1 + G(y_1)$$

$$y_1 = x_1 + F(x_2)$$
$$y_2 = x_2 + G(y_1)$$

$$x_2 = y_2 - G(y_1)$$
$$x_1 = y_1 - F(x_2)$$

- F - self-attention block
- G - feed-forward layer

Profit: storing activations only once during the training process

https://towardsdatascience.com/illustrating-the-reformer-393575ac6ba0

Chunking

Computations in feed-forward layers are independent across positions in a sequence => the computations for the forward and backward passes can be split into chunks.

$$Y_2 = \left[Y_2^{(1)}; \ldots; Y_2^{(c)}\right] = \left[X_2^{(1)} + \text{FeedForward}(Y_1^{(1)}); \ldots; X_2^{(c)} + \text{FeedForward}(Y_1^{(c)})\right]$$

Chunking in the forward pass computation [Image is taken from the Reformer paper]

https://towardsdatascience.com/illustrating-the-reformer-393575ac6ba0