Deep Learning in Applications

Lecture 6:
How NLP Cracked Transfer Learning

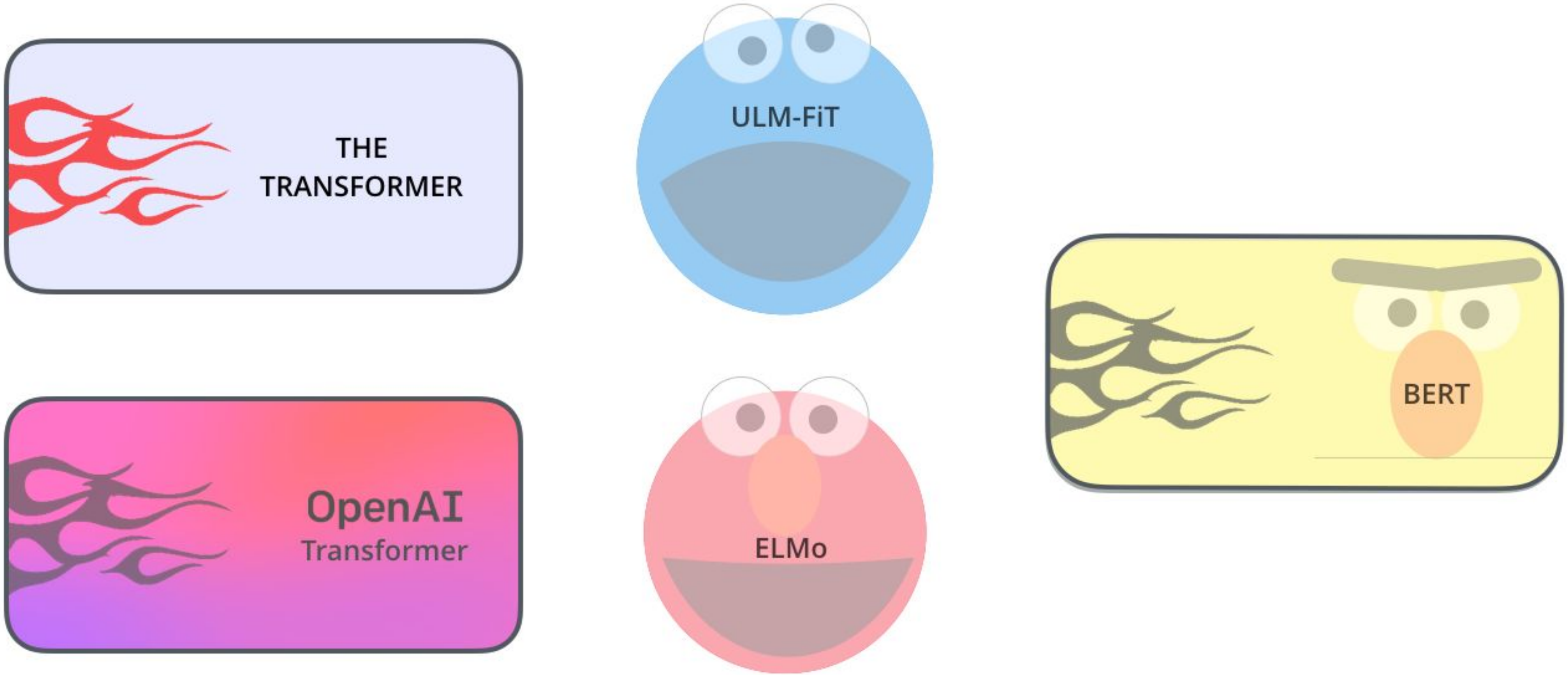
Anastasia Ianina

Harbour.Space University
15.07.2019, Barcelona, Spain

Outline

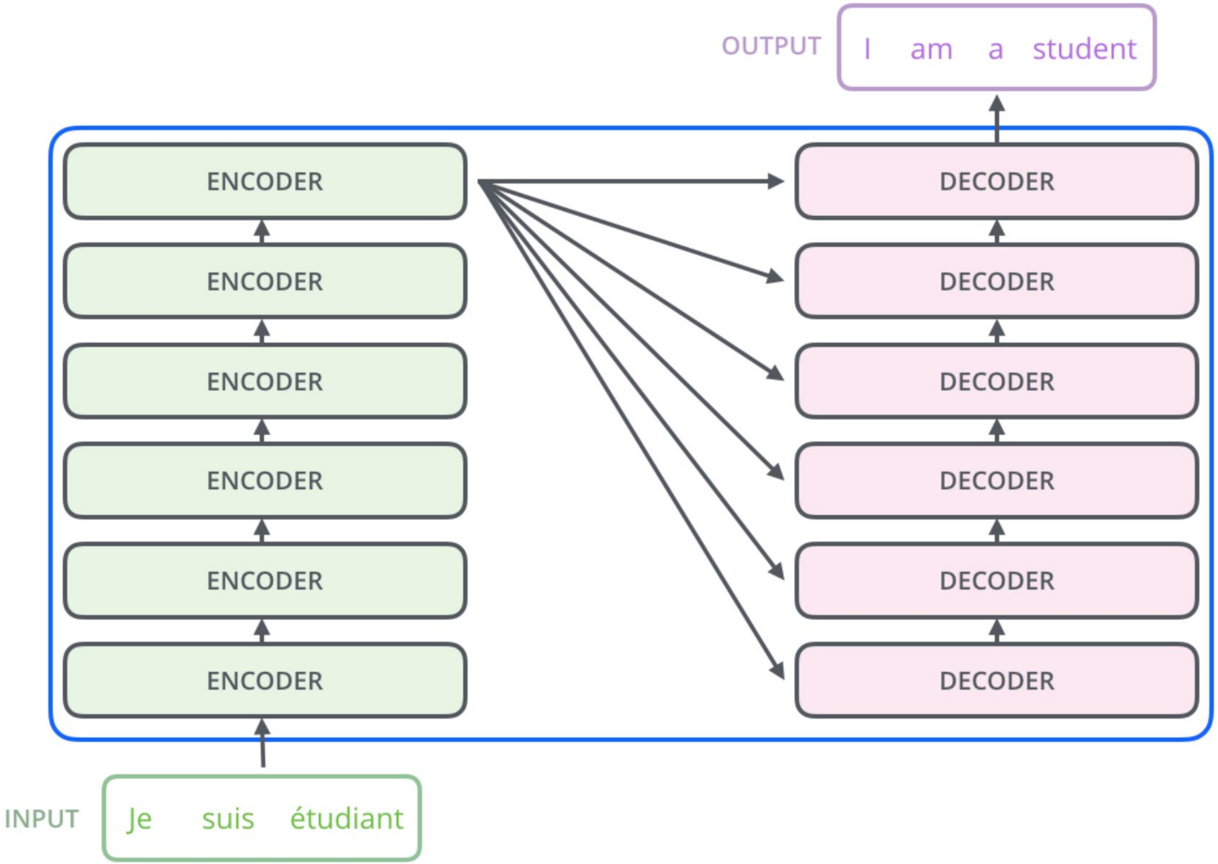
1. Transformer: recap
2. OpenAI Transformer
3. ELMO
4. BERT
5. Q & A

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/

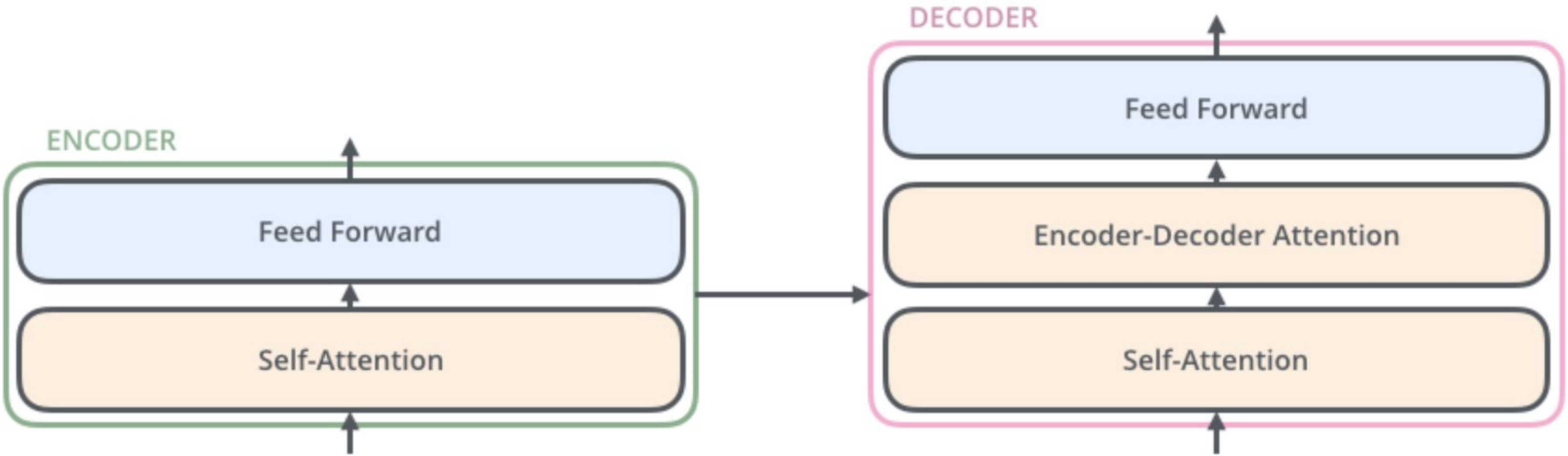

THE
TRANSFORMER
OpenAI
Transformer
ULM-FiT
ELMo
BERT

The Transformer: recap

The Transformer

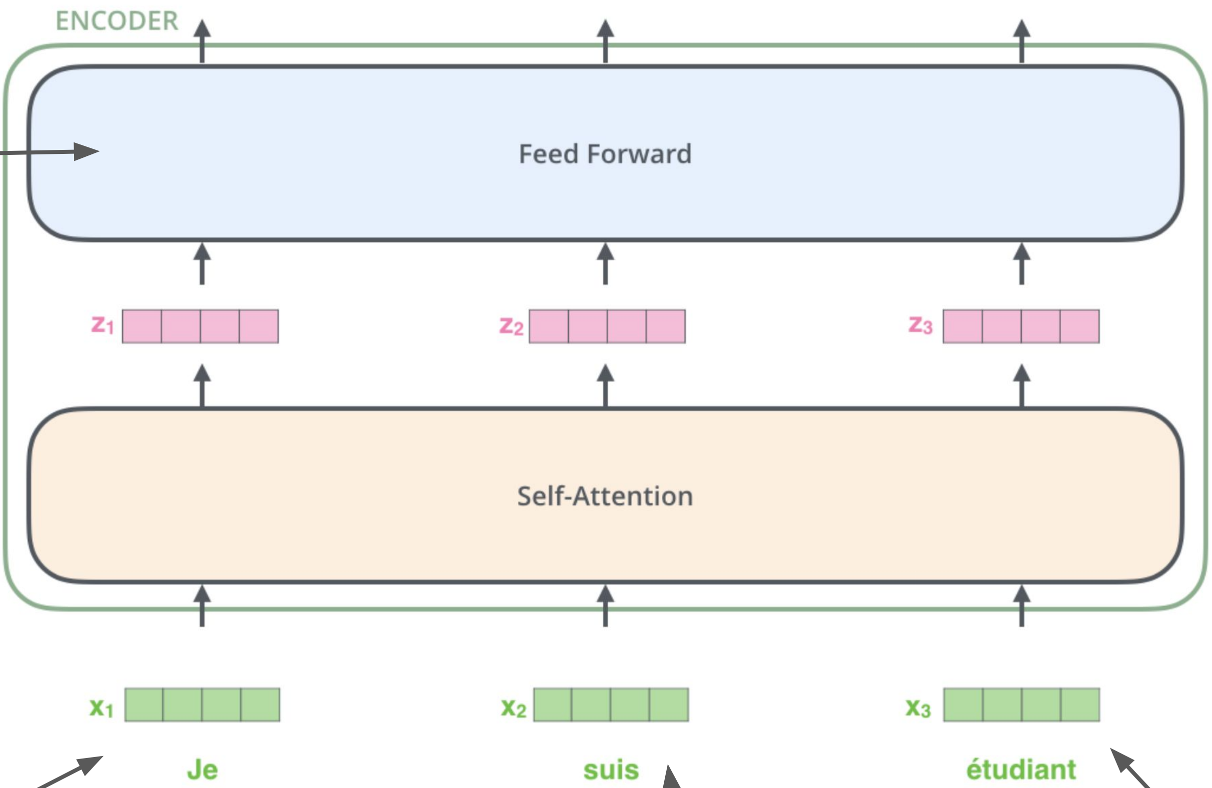


The Transformer

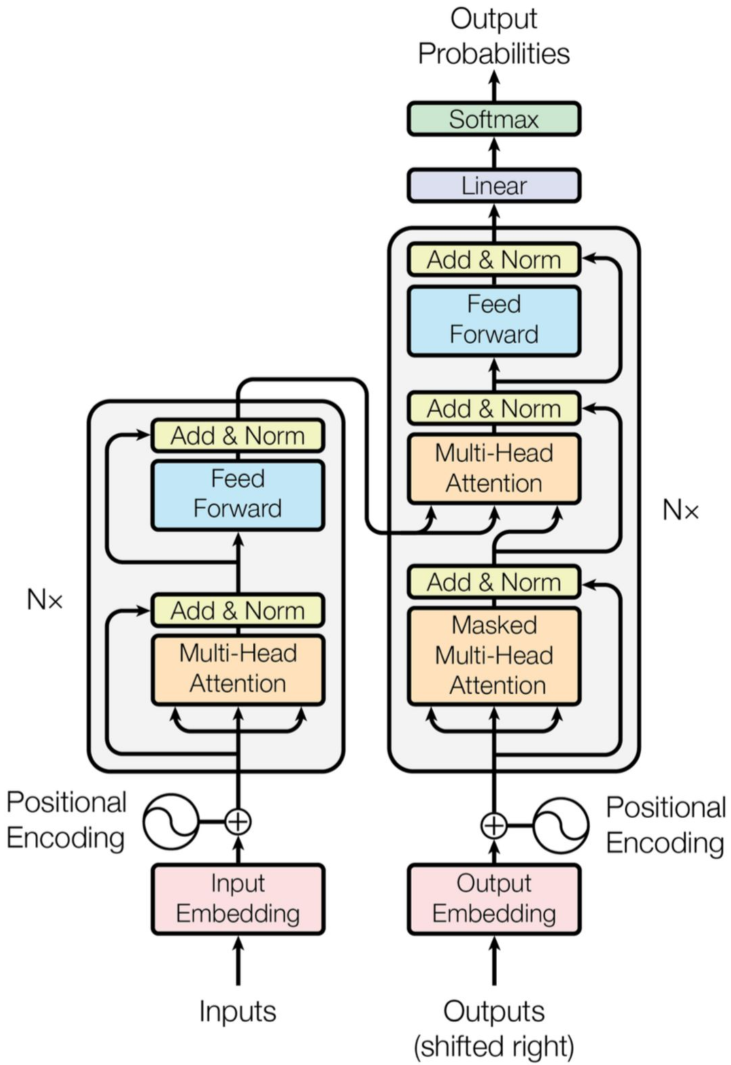


The Transformer

Can be parallelized



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

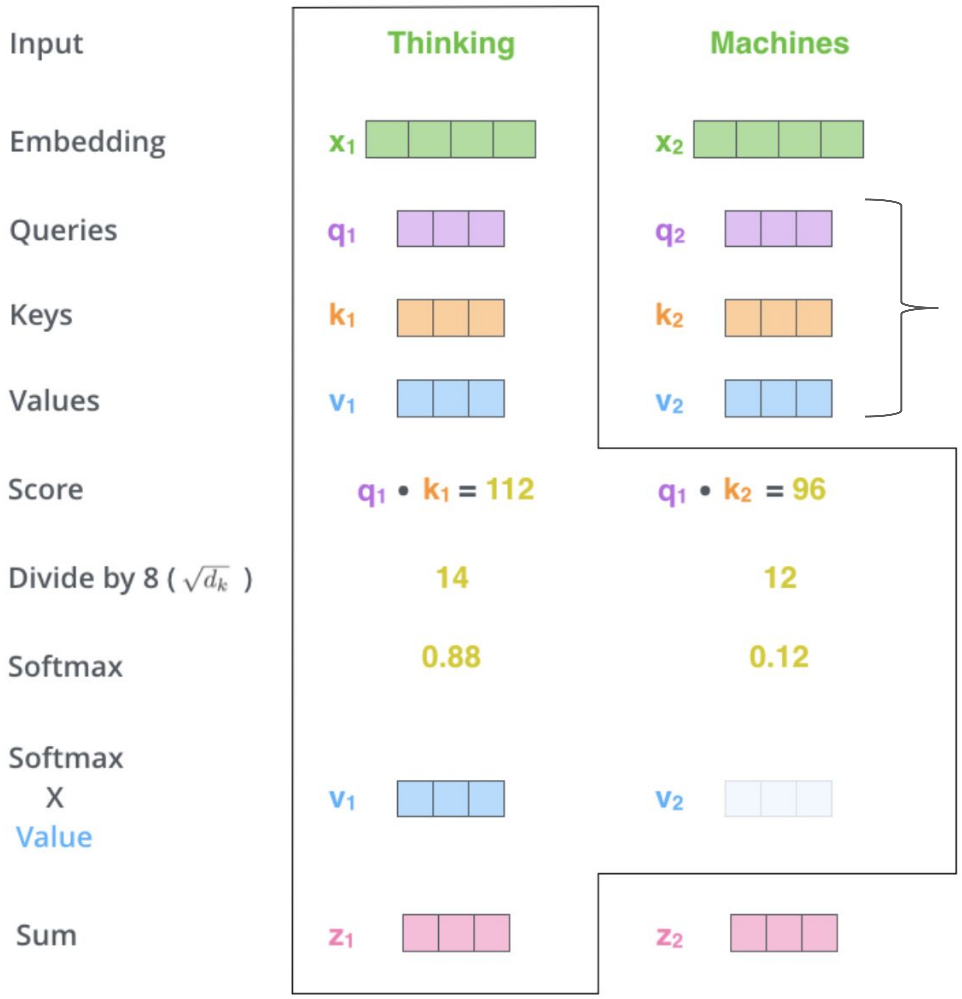


The Transformer: recap

- 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



<u>STEP 1:</u> create Query, Key, Value

<u>STEP 2:</u> calculate scores

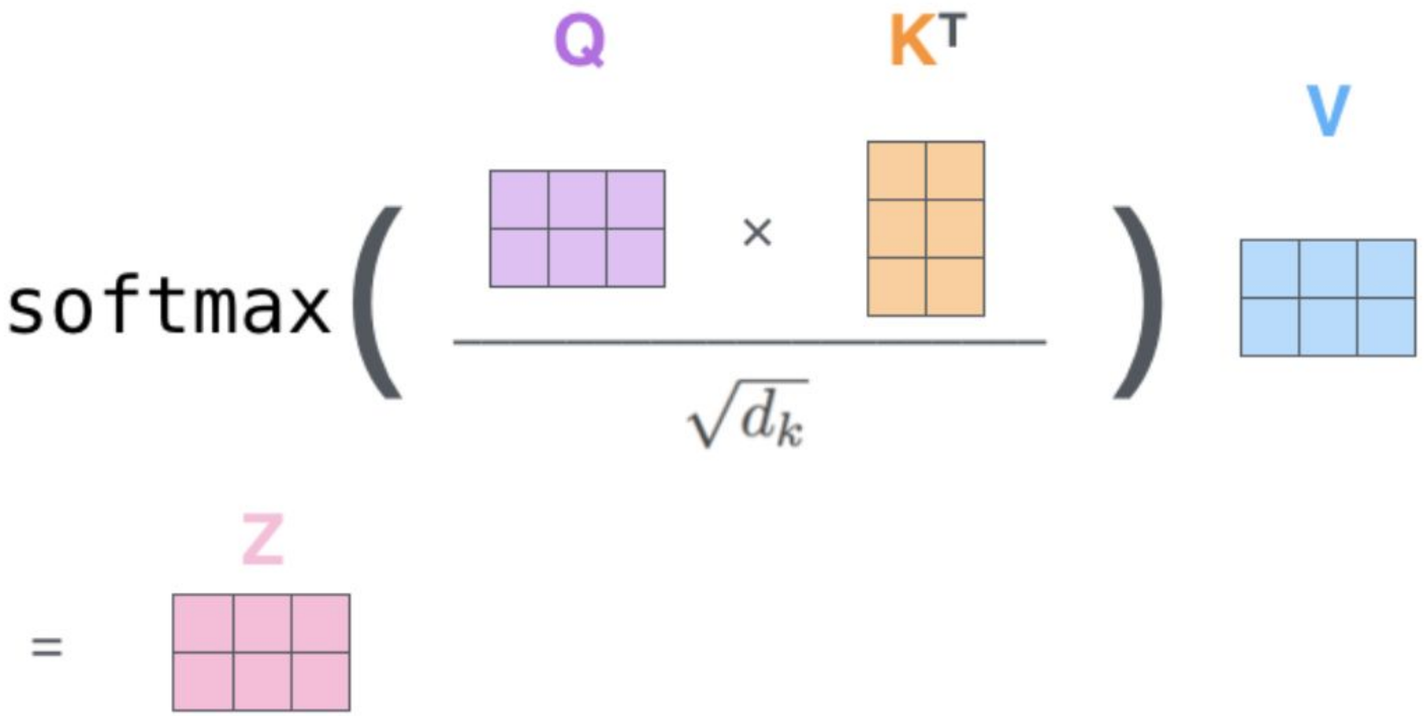
<u>STEP 3:</u> divide by $\sqrt{d_k}$

<u>STEP 4:</u> softmax

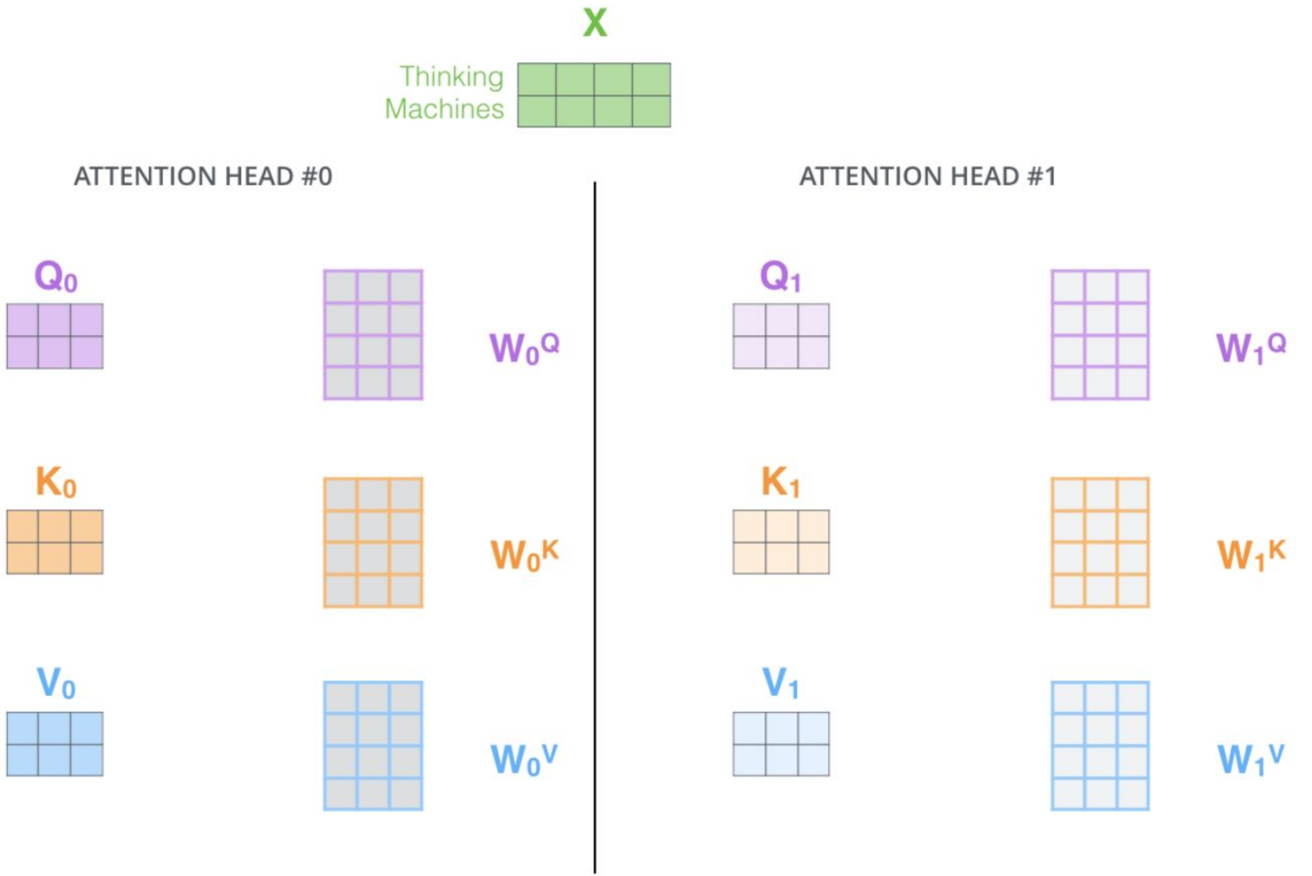
<u>STEP 5:</u> multiply each value vector by the softmax score

<u>STEP 6:</u> sum up the weighted value vectors

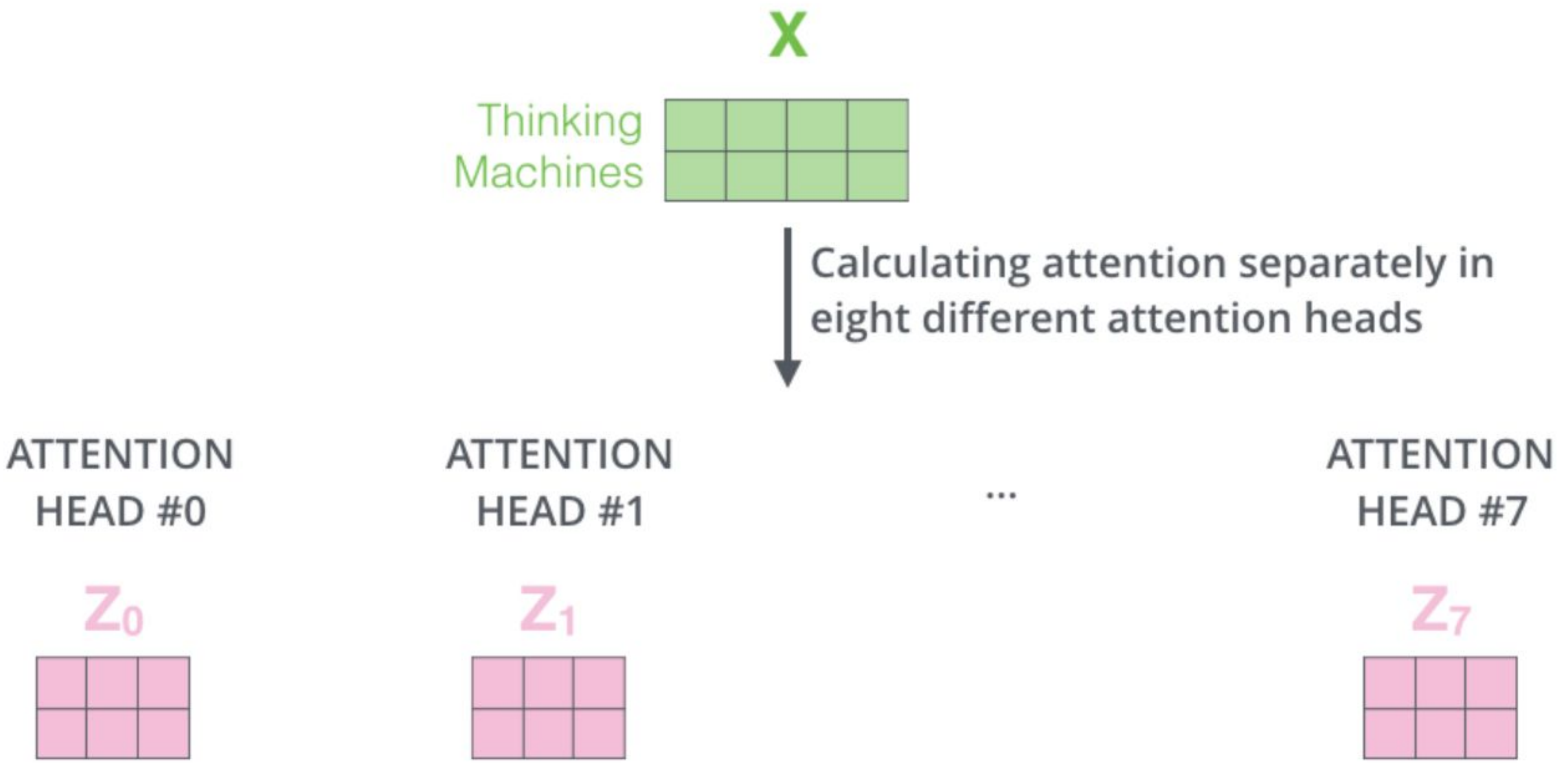
Self-Attention: Matrix Calculation



Multi-Head Attention



Multi-Head Attention



Multi-Head Attention

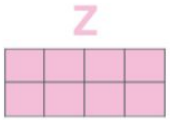
1) Concatenate all the attention heads

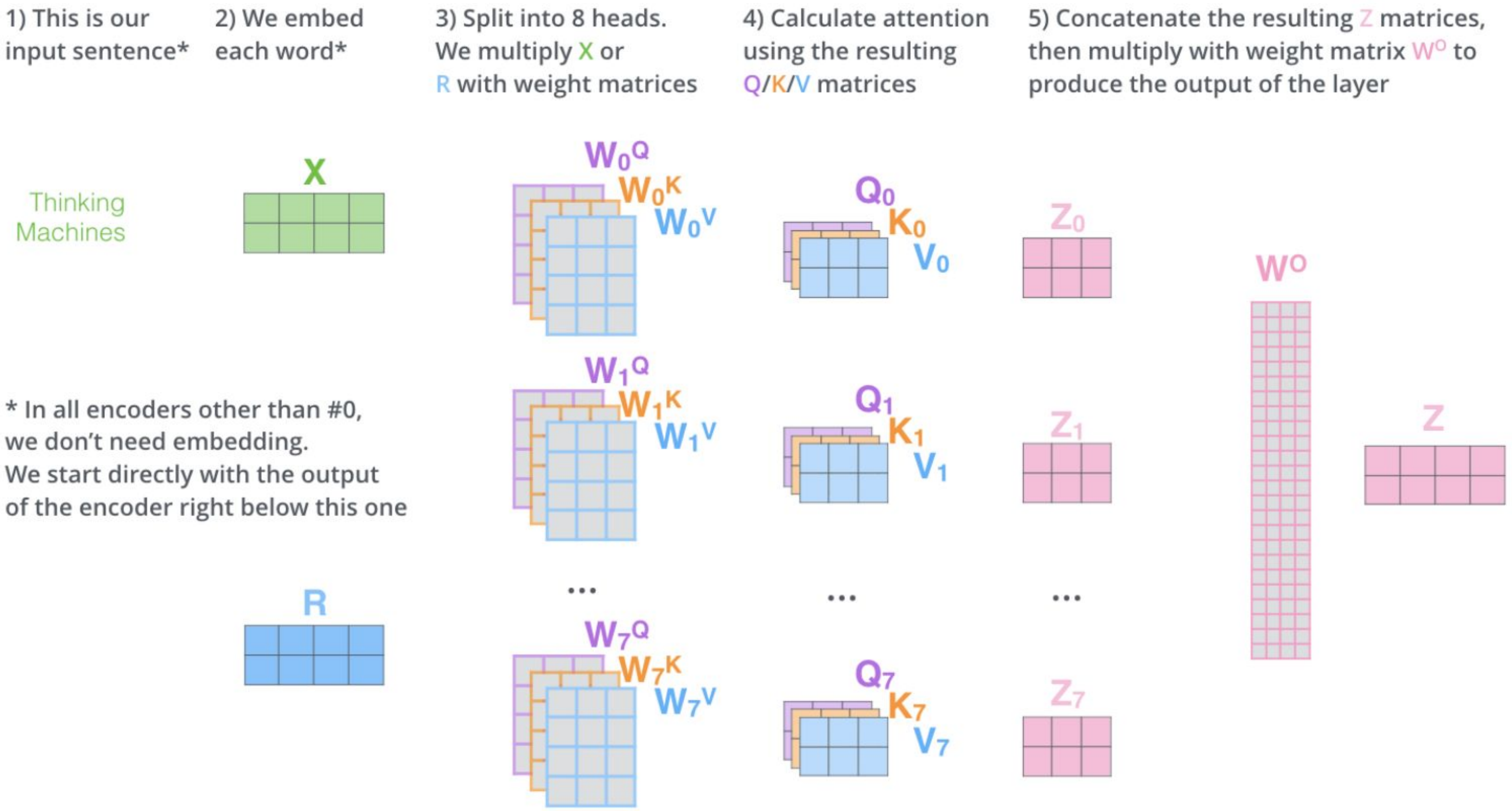
2) Multiply with a weight matrix W^O that was trained jointly with the model

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

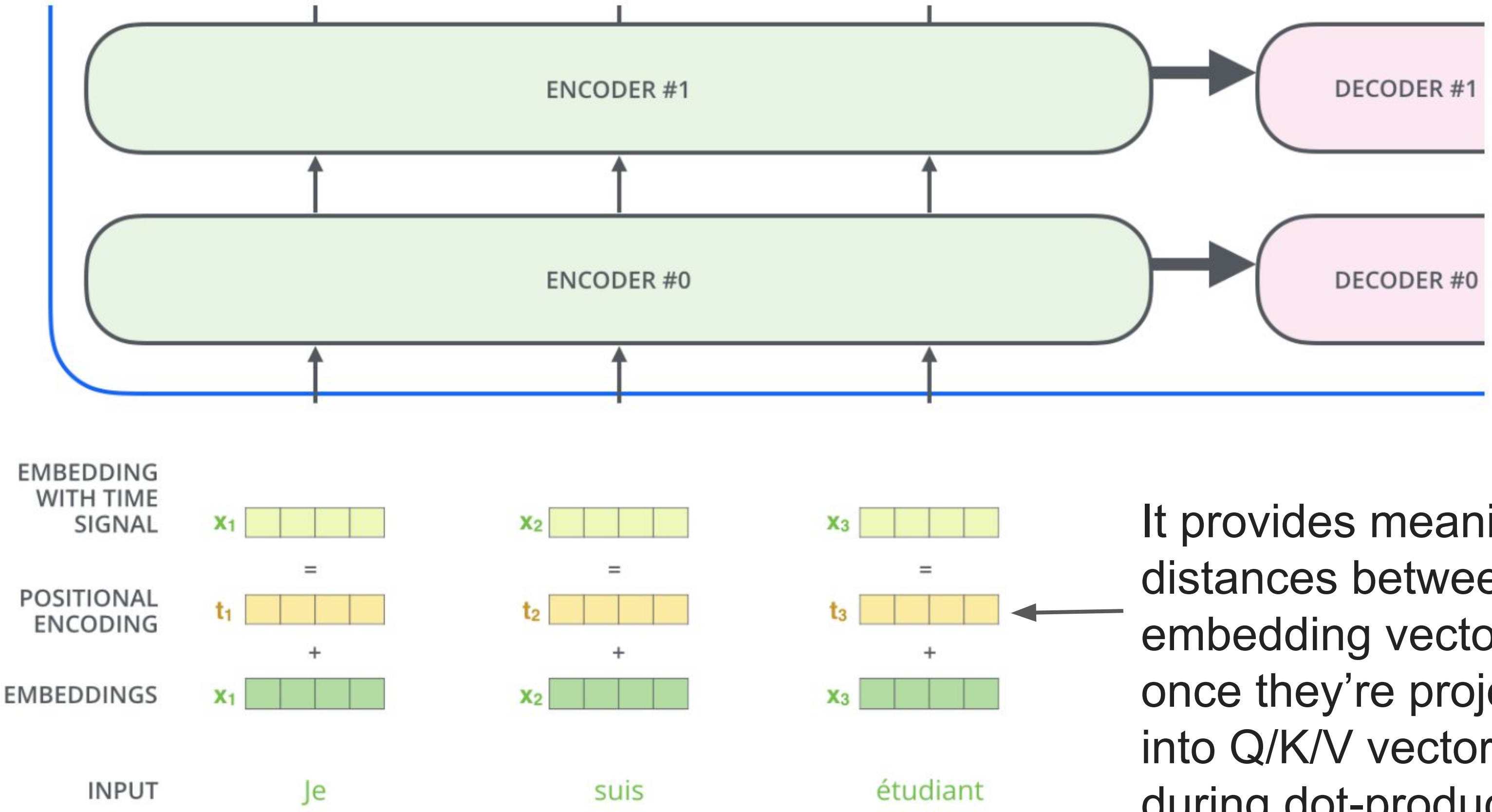
=




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

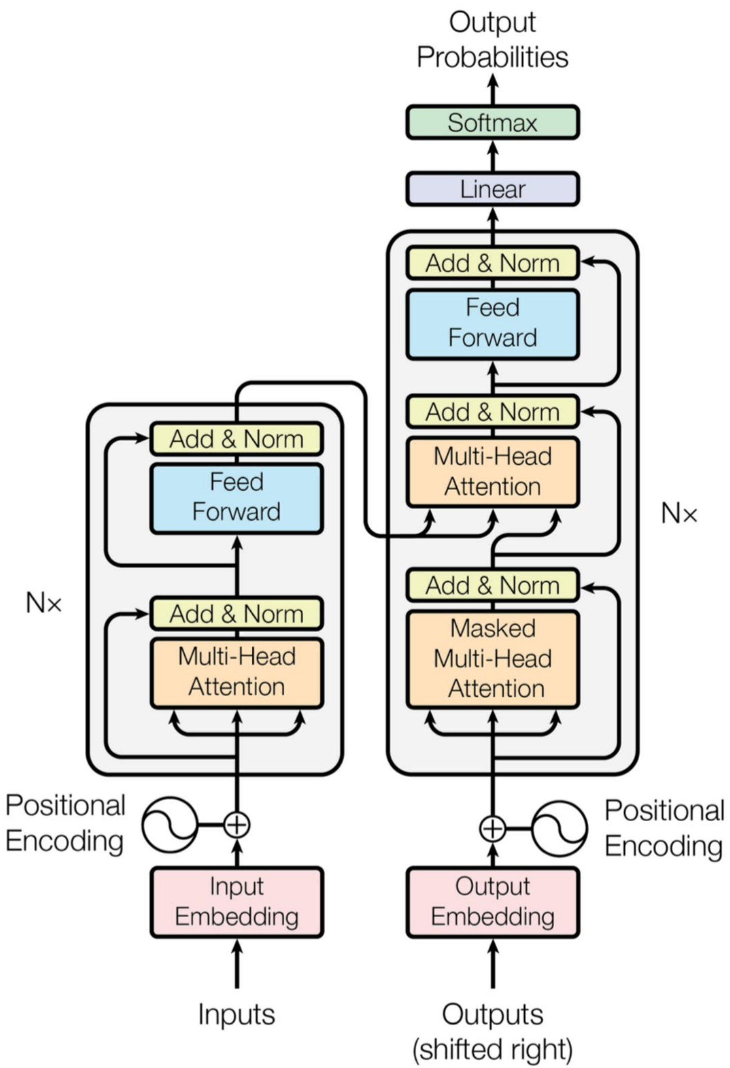


It provides meaningful distances between the embedding vectors once they're projected into Q/K/V vectors and during dot-product attention

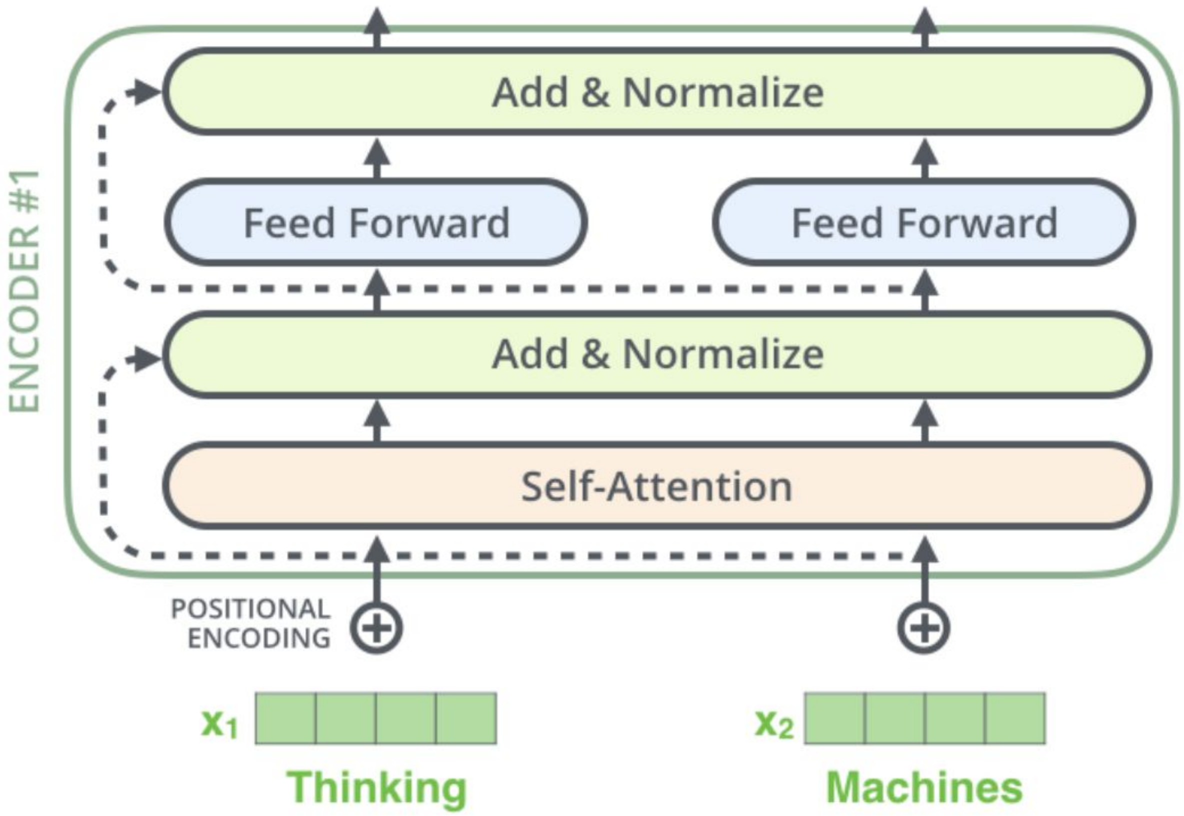
Positional Encoding

Layer Normalization

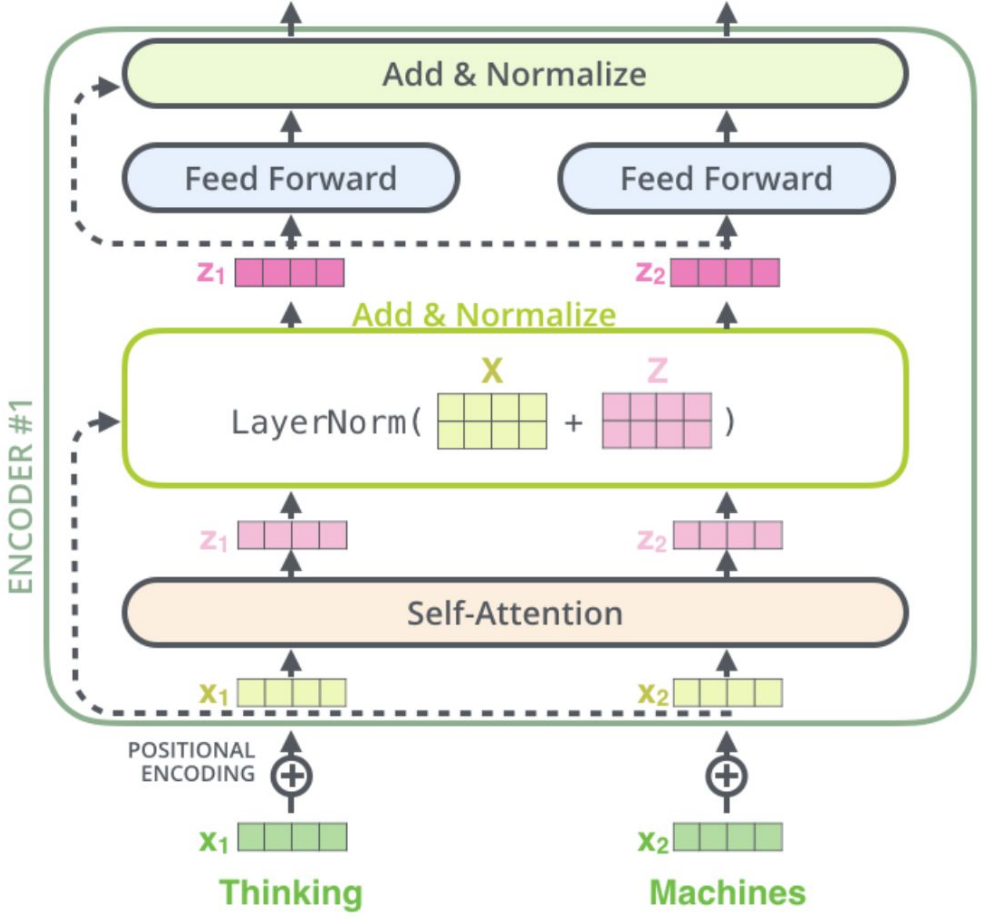
The Transformer: recap



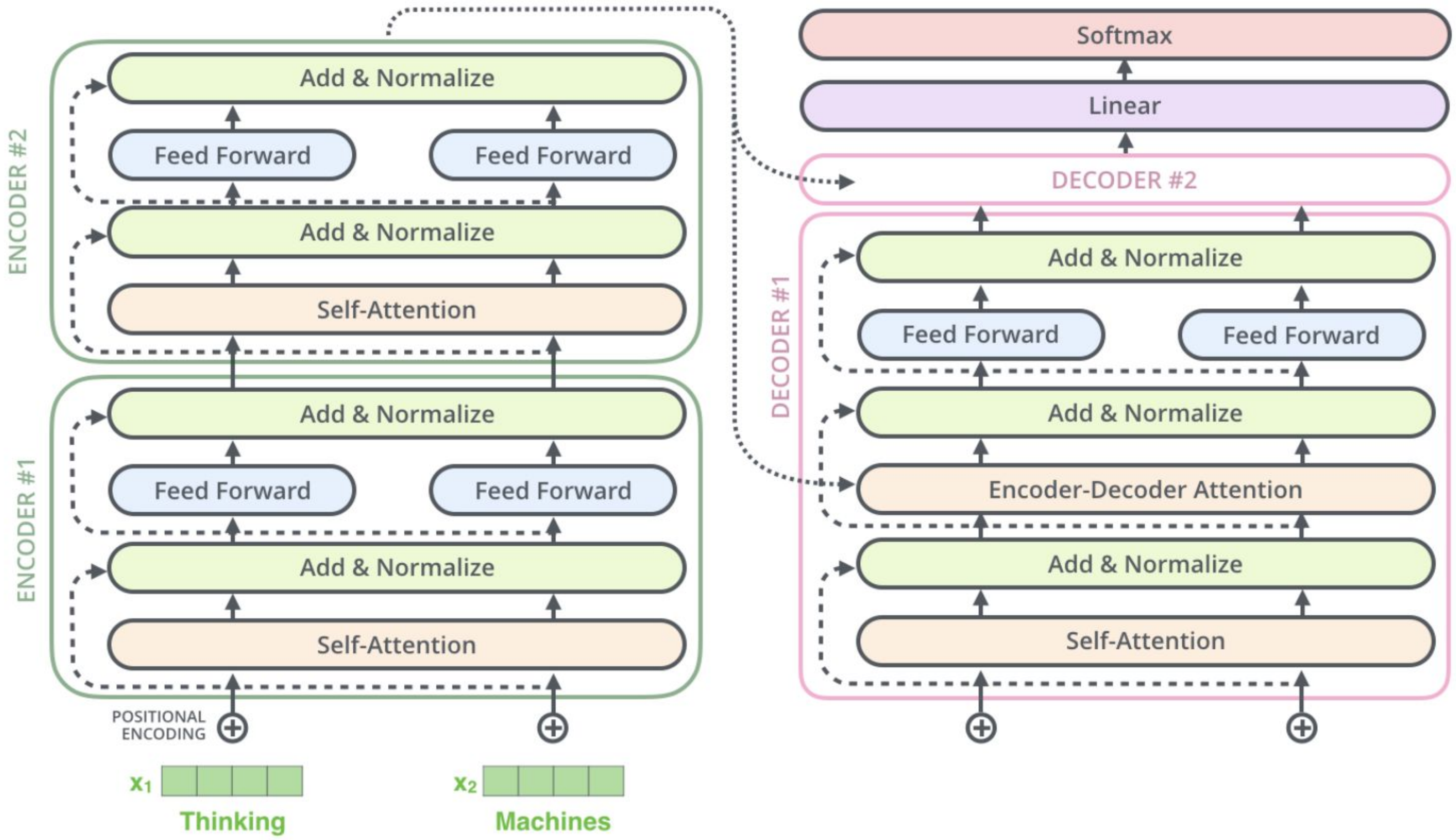
Layer Normalization



Layer Normalization

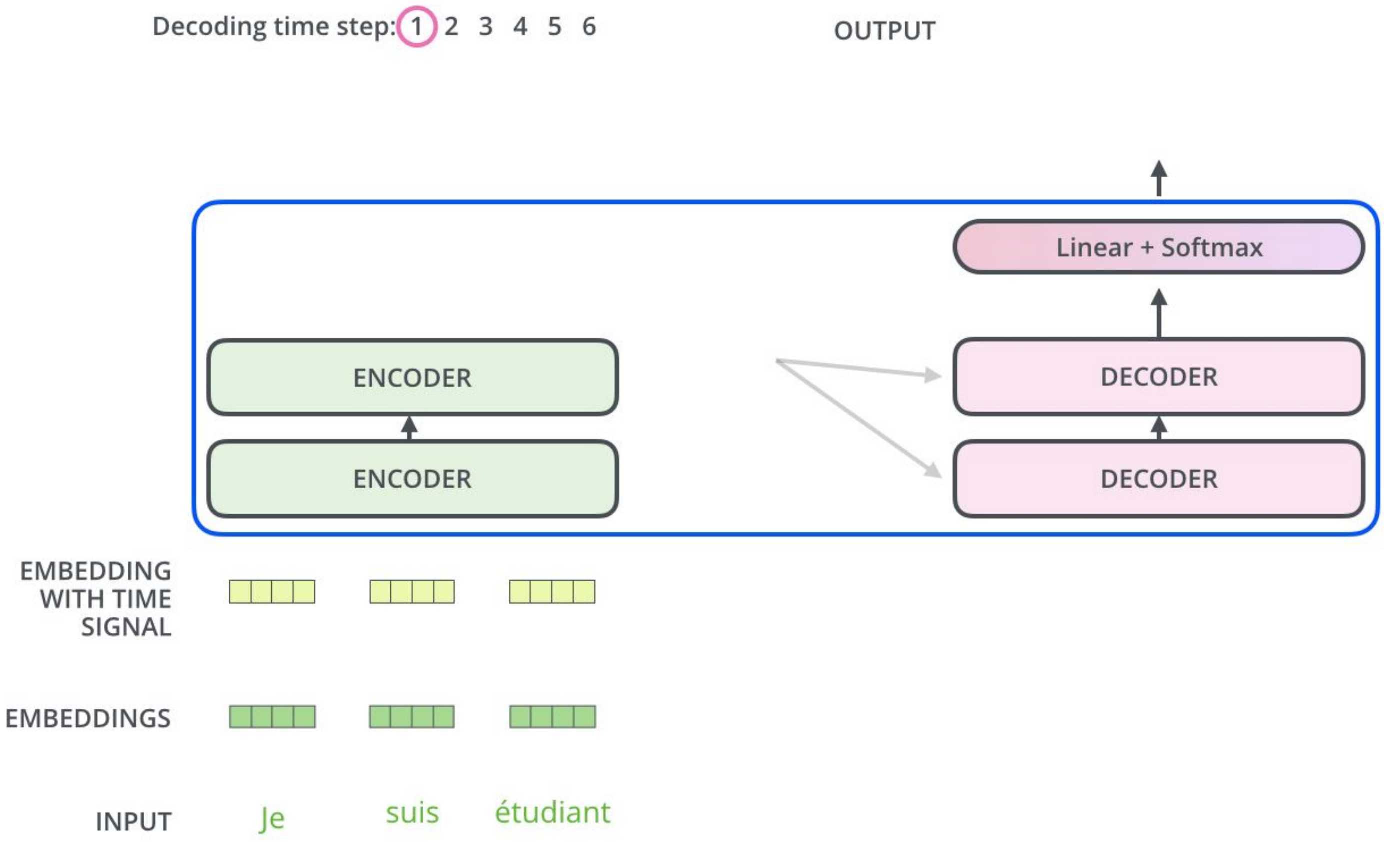


Layer Normalization

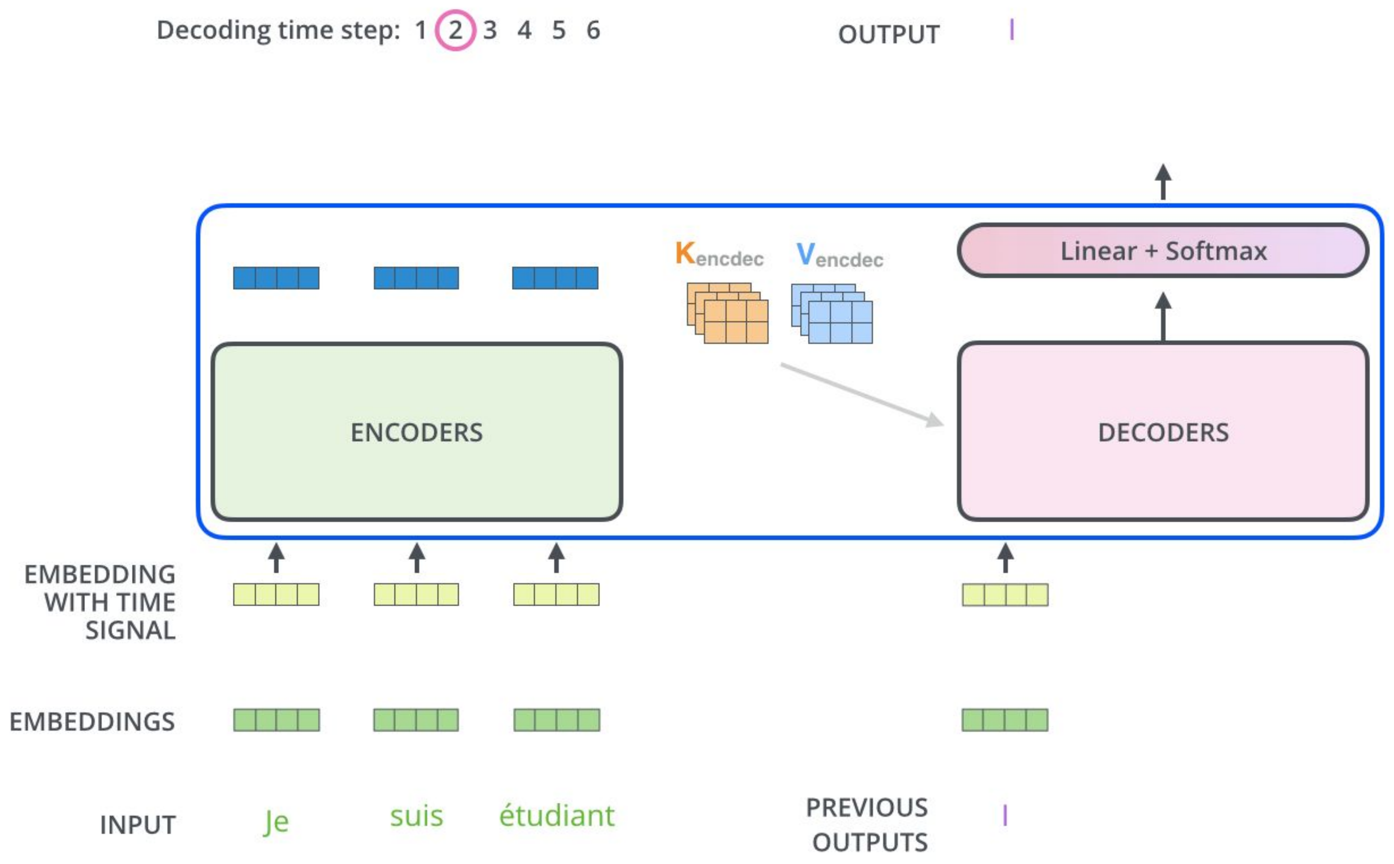


The Decoder

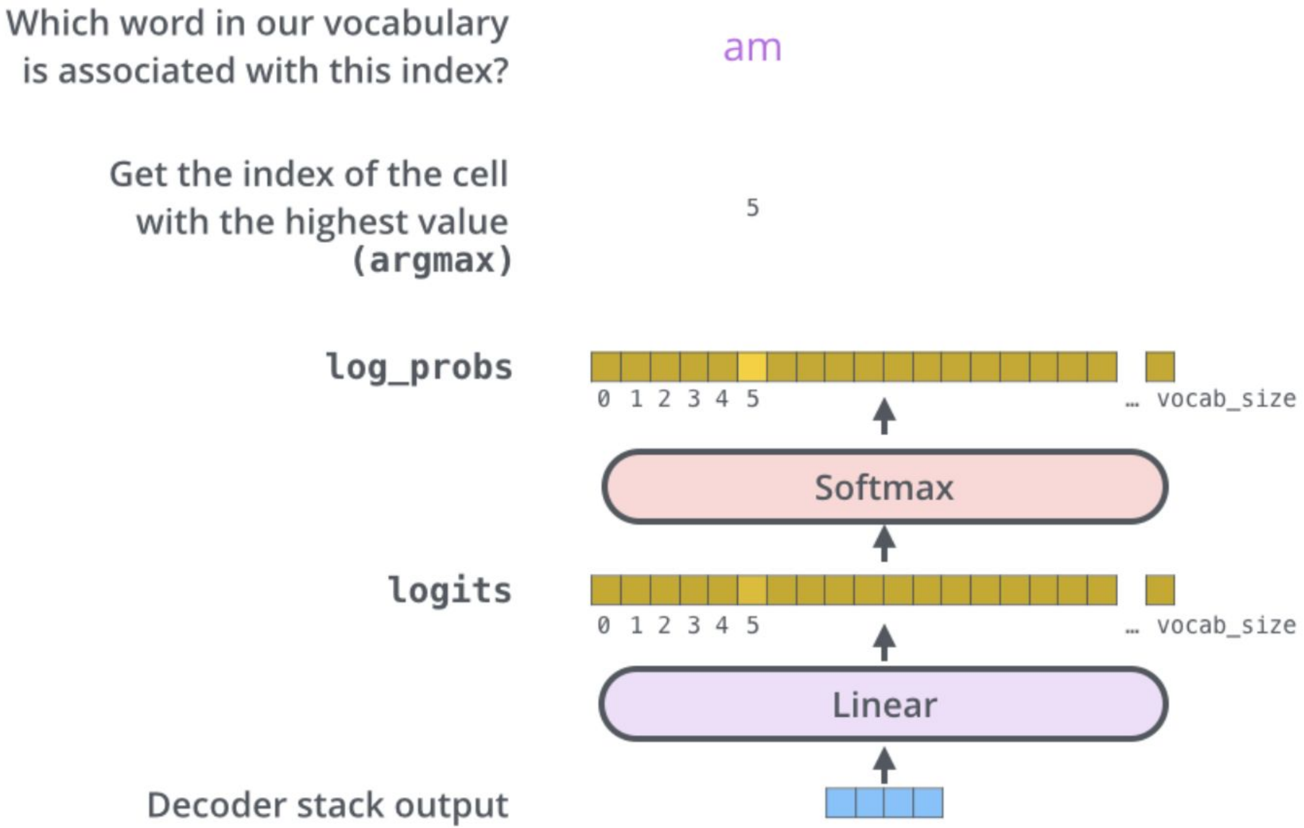
The Decoder Side



The Decoder Side



The Decoder Side



Recap of Training

Recap of 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

Example: one-hot encoding of our output vocabulary

Recap of Training

Target Model Outputs

Output Vocabulary:	a	am	I	thanks	student	<eos>
position #1	0.0	0.0	1.0	0.0	0.0	0.0
position #2	0.0	1.0	0.0	0.0	0.0	0.0
position #3	1.0	0.0	0.0	0.0	0.0	0.0
position #4	0.0	0.0	0.0	0.0	1.0	0.0
position #5	0.0	0.0	0.0	0.0	0.0	1.0
	a	am	I	thanks	student	<eos>

Trained Model Outputs

Output Vocabulary:	a	am	I	thanks	student	<eos>
position #1	0.01	0.02	0.93	0.01	0.03	0.01
position #2	0.01	0.8	0.1	0.05	0.01	0.03
position #3	0.99	0.001	0.001	0.001	0.002	0.001
position #4	0.001	0.002	0.001	0.02	0.94	0.01
position #5	0.01	0.01	0.001	0.001	0.001	0.98
	a	am	I	thanks	student	<eos>

Loss Function

Kullback-Leibler Divergence:

$$D_{KL}(P||Q) = \sum_{x} p(x) log \frac{p(x)}{q(x)}$$

Cross-Entropy:

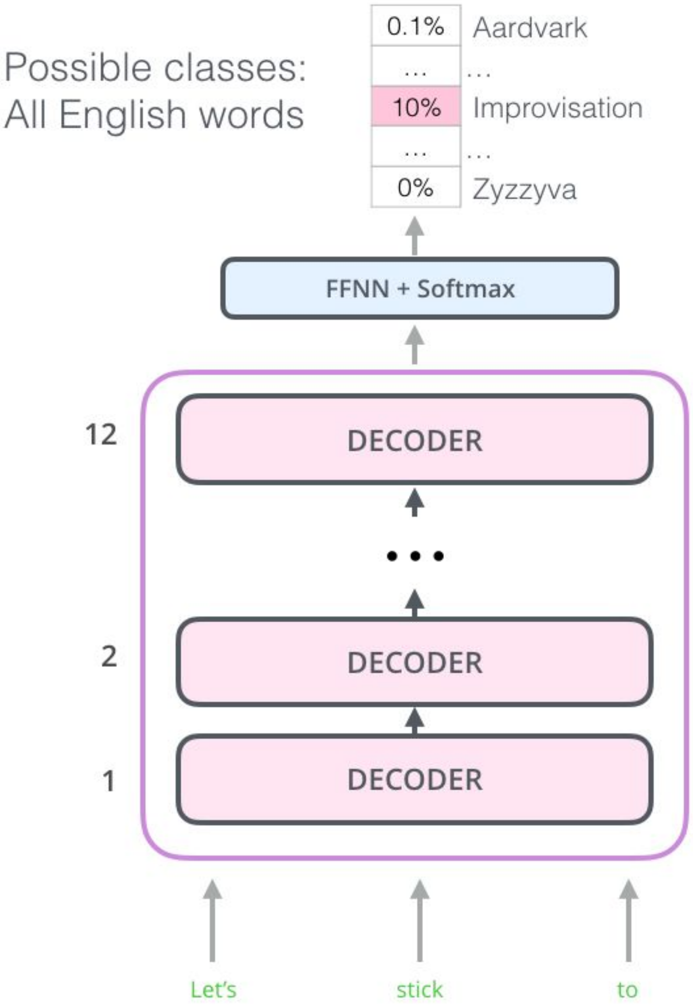
$$H(p, q) = -\sum_{x} p(x) \log q(x)$$

OpenAI Transformer: Pre-training Decoder for Language Modeling

OpenAI Transformer

- The Encoder-Decoder structure of the transformer made it perfect for machine translation
- But what about sentence classification?
- **Main goal: pre-train a language model that can be fine-tuned for other tasks**

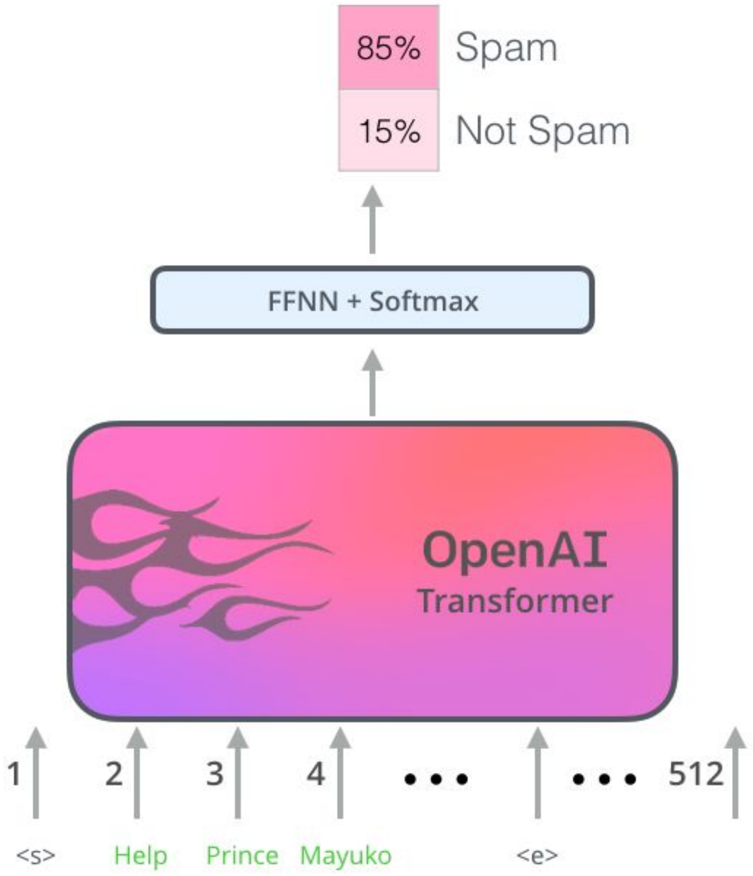
OpenAI Transformer



Differences from vanilla Transformer:

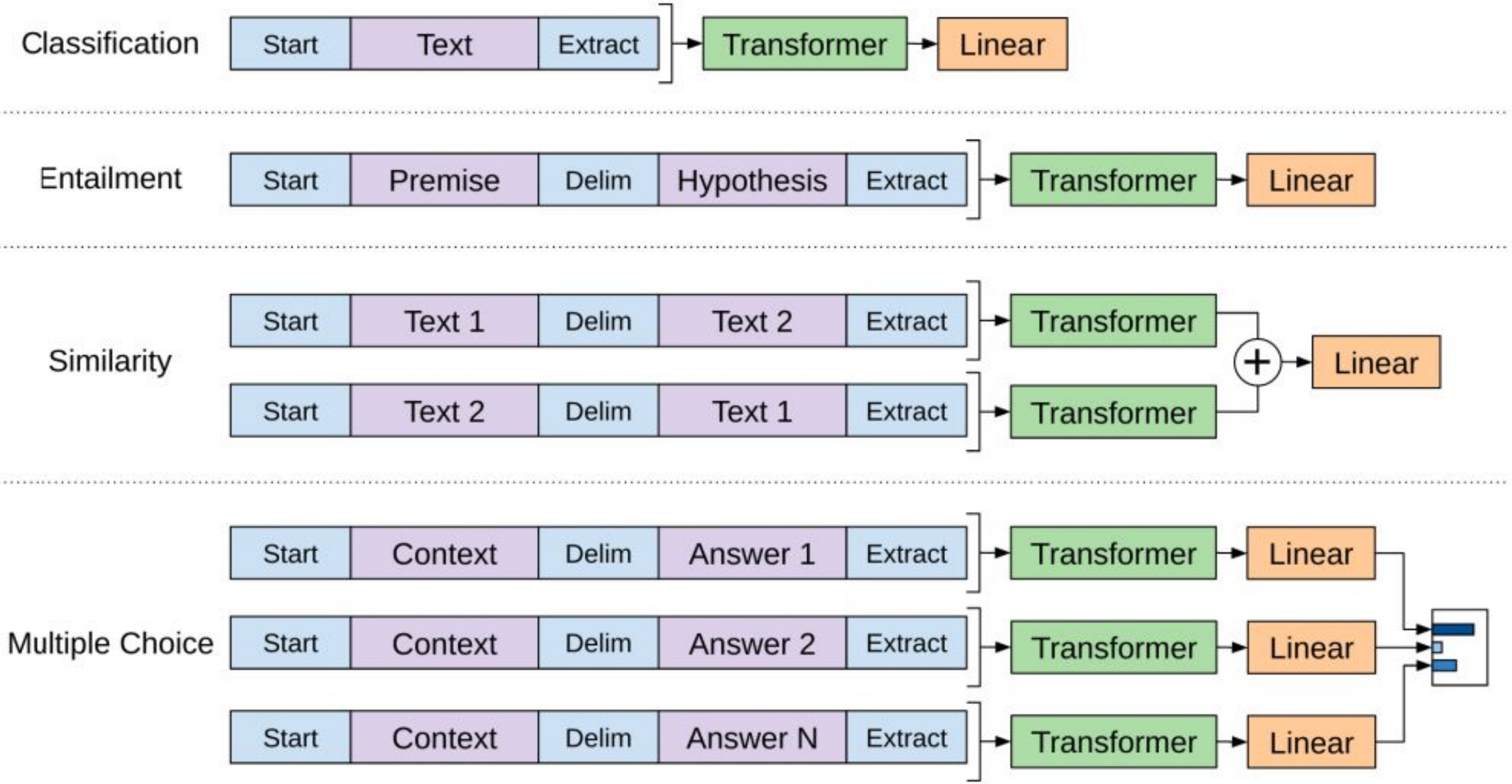
- no encoder
- decoder layers would not have the encoder-decoder attention sublayer
- Pre-train the model on predicting the next word using massive (unlabeled) datasets

OpenAI Transformer



- During pre-training phase layers have been tuned to reasonably handle language
- Now let’s use it for downstream tasks (e.g. sentence classification)

Input transformations for different tasks



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

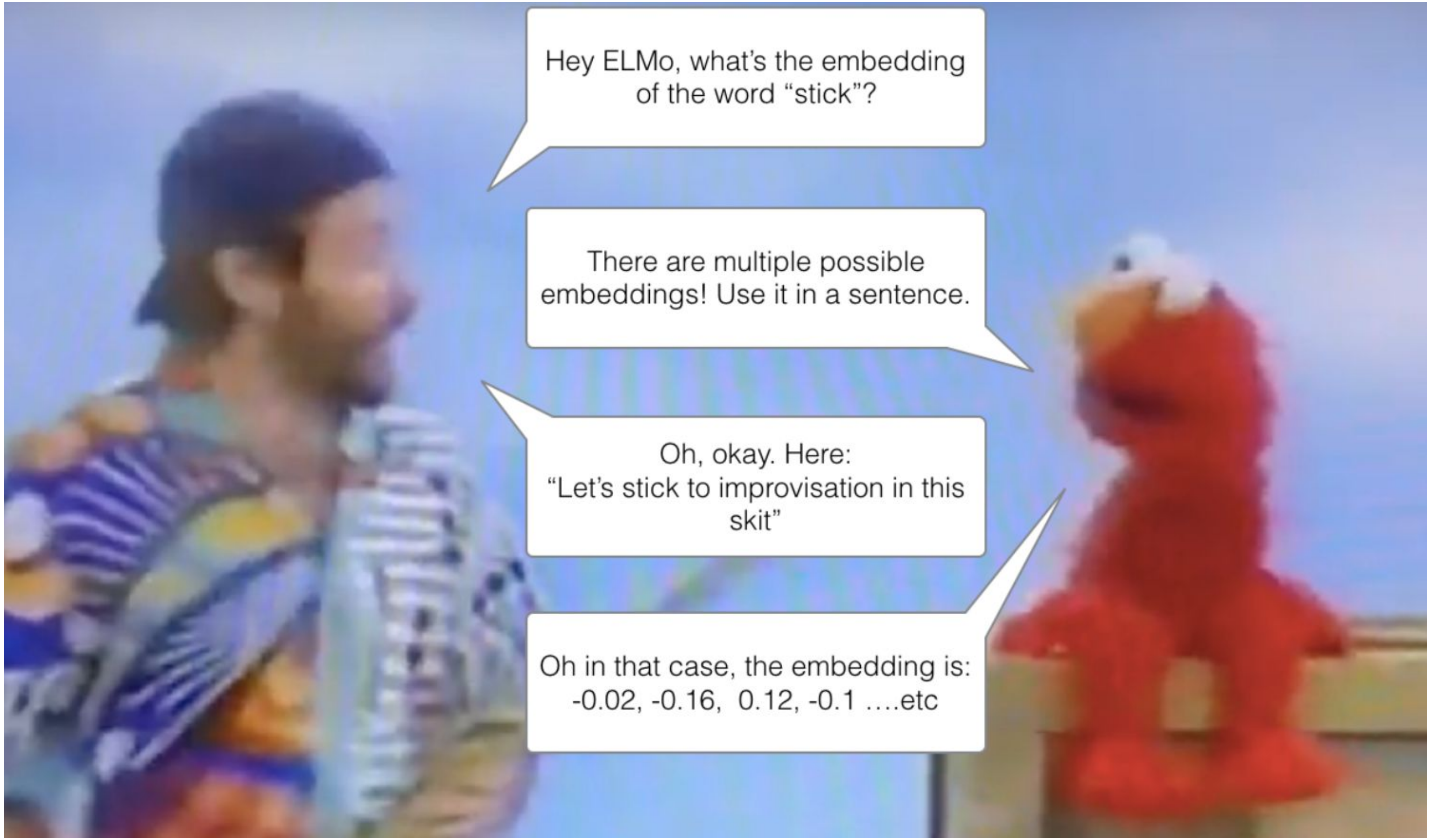
ELMo

What does it stand for?

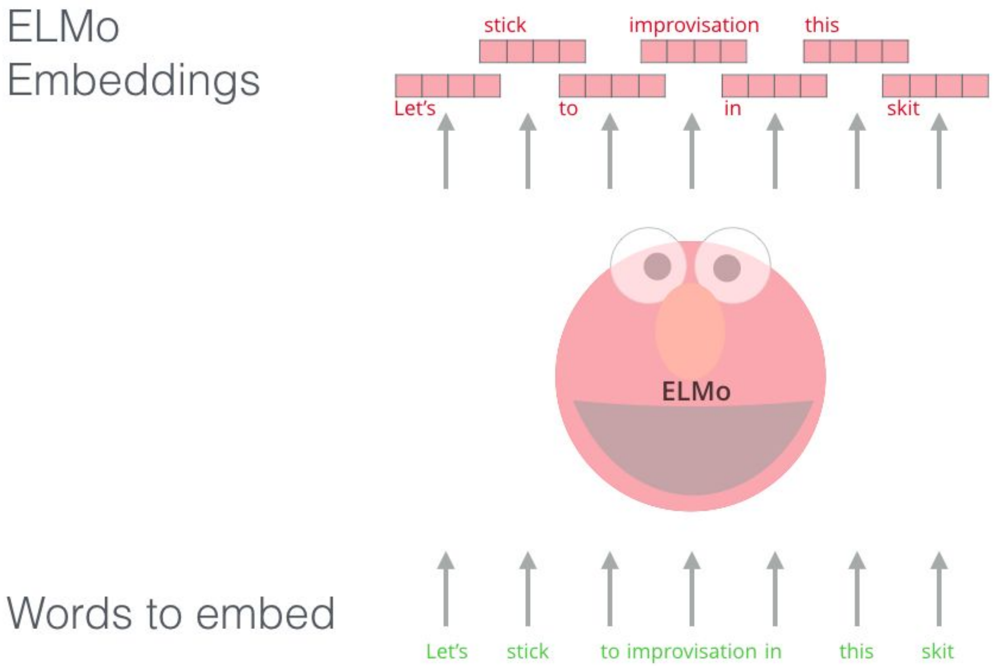
1. **E**xpedited **L**abour **M**arket **O**pinion
2. **E**lectric **L**ight **M**achine **O**rganization
3. **E**nough **L**et’s **M**ove **O**n

4. **Embeddings from Language Models**

ELMo: contextualized word embeddings

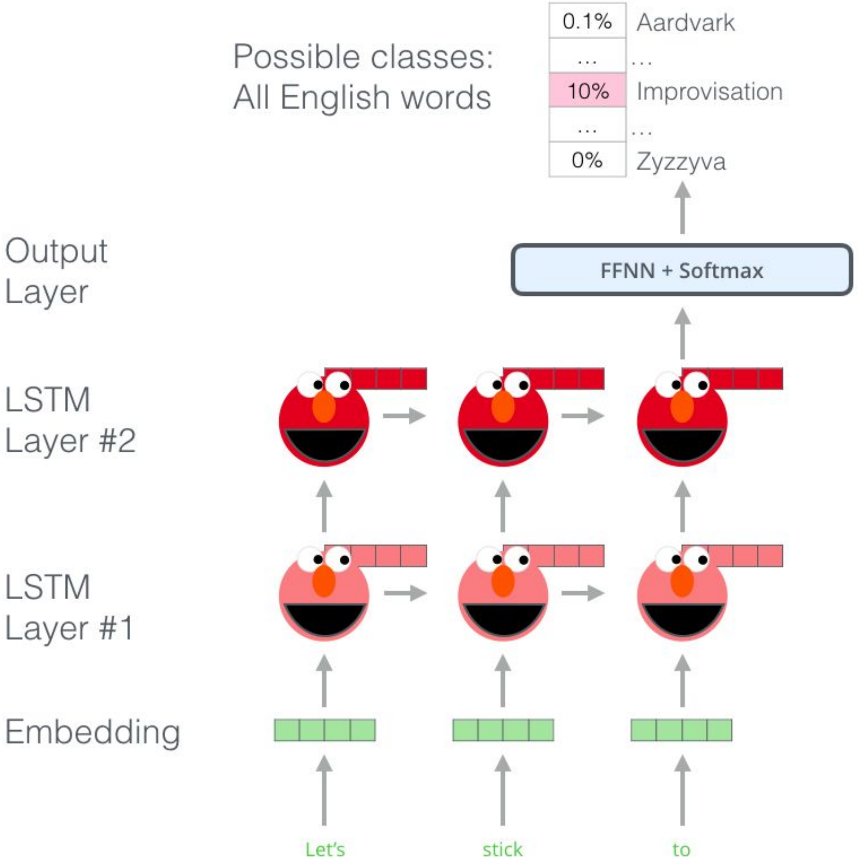


ELMo: Contextualized word embeddings



- 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:

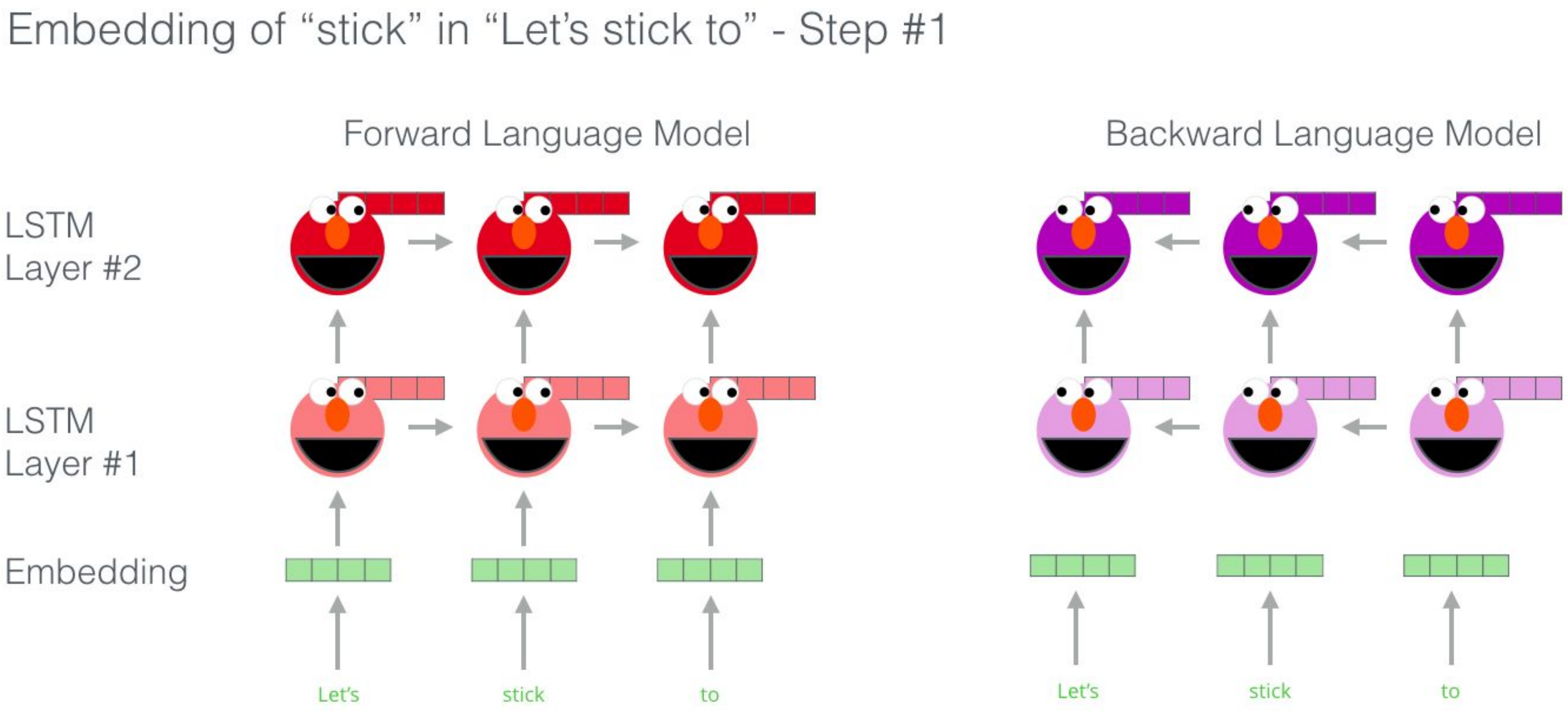
$$p(t_1, t_2, \ldots, t_N) = \prod_{k=1}^{N} p(t_k \mid t_1, t_2, \ldots, t_{k-1})$$

- Backward:

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

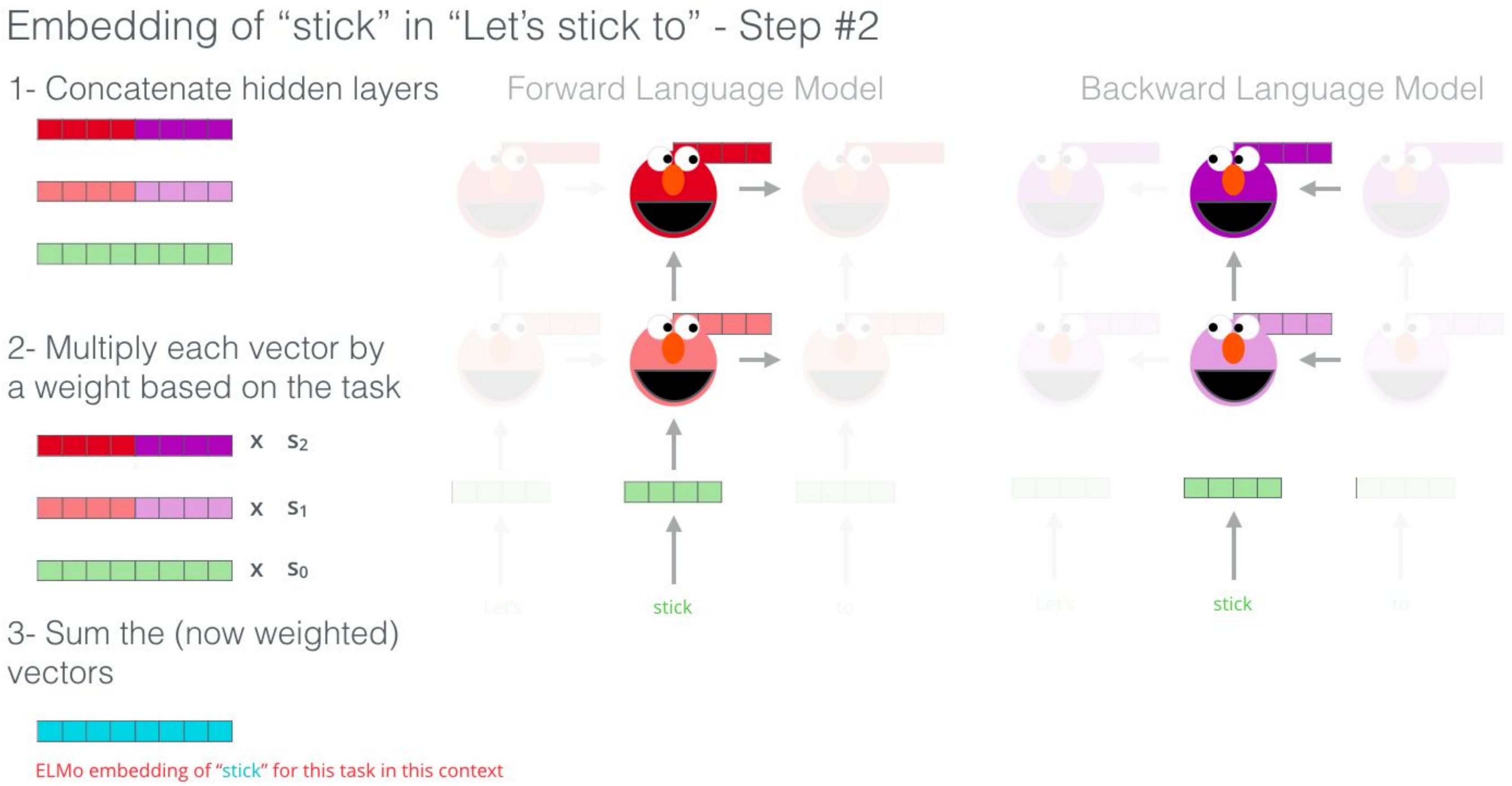
LSTM predicts next word in both directions to build biLMs

ELMo: main pipeline



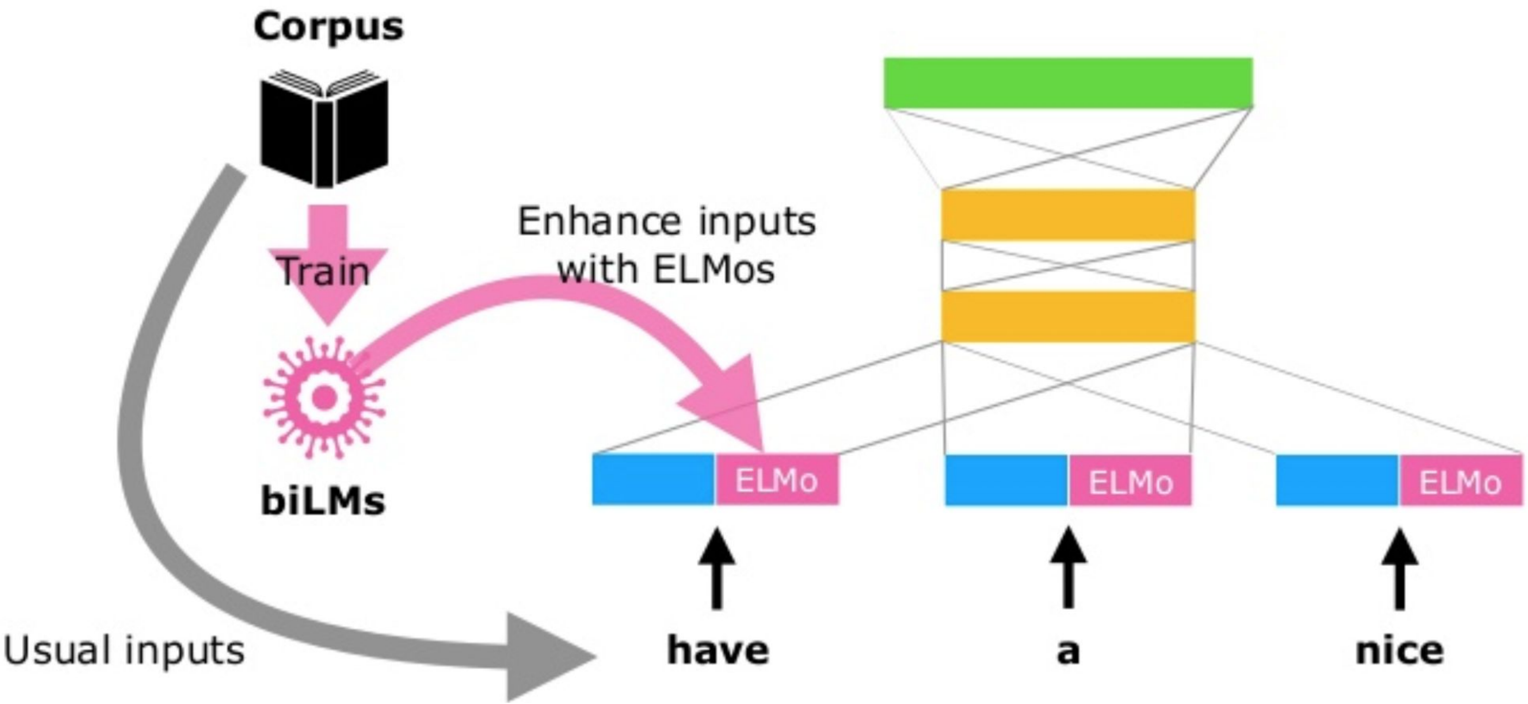
ELMo: main pipeline

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



ELMo

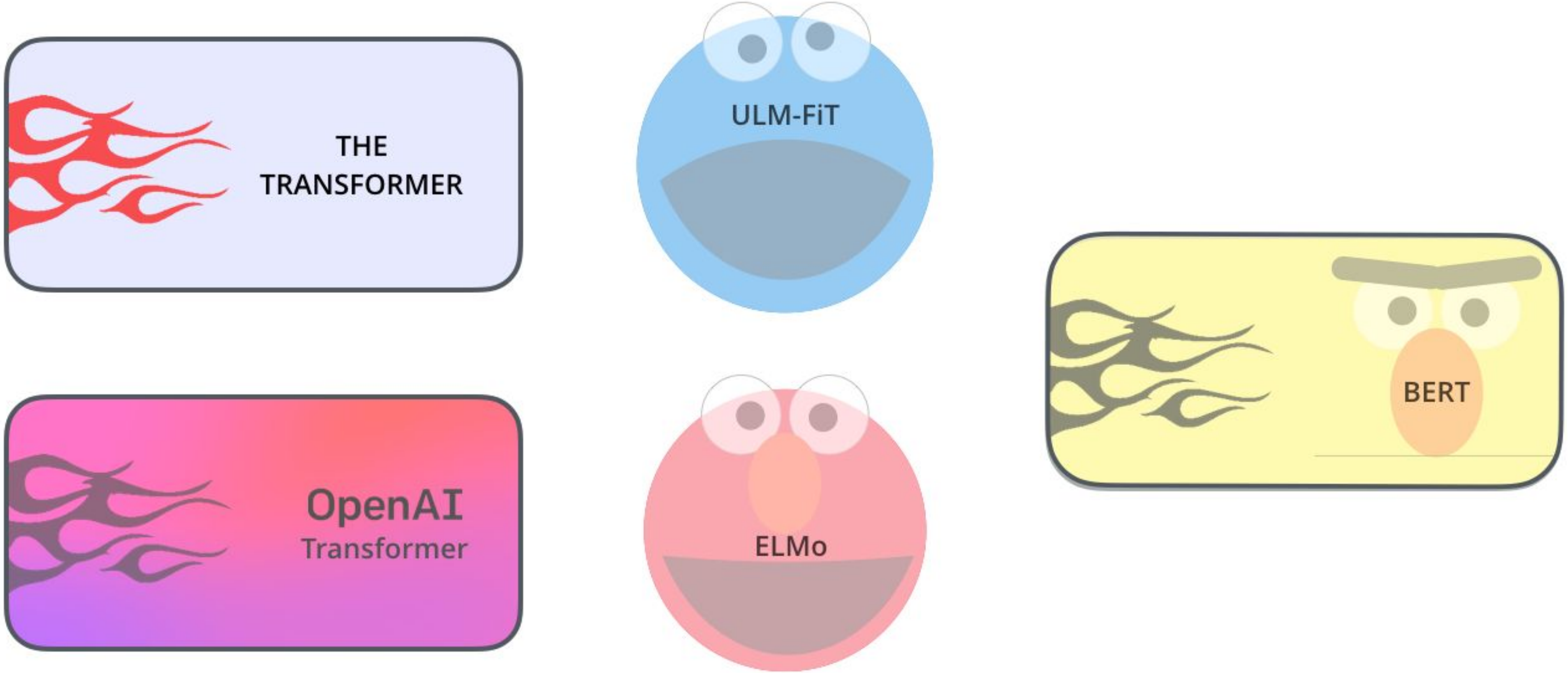
ELMo can be integrated to almost all neural NLP tasks with simple concatenation to the embedding layer



ELMo: overview

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

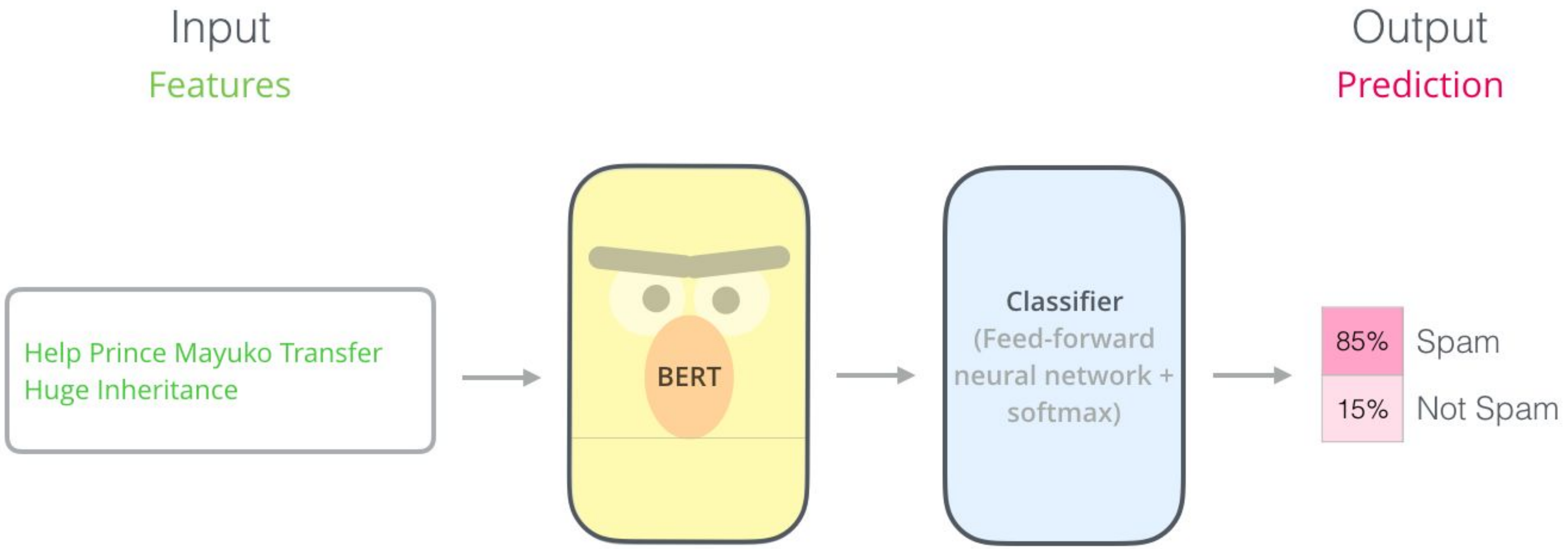
- Higher levels seems to catch semantics while lower layer probably capture syntactic features


THE
TRANSFORMER
ULM-FiT
BERT
OpenAI
Transformer
ELMo

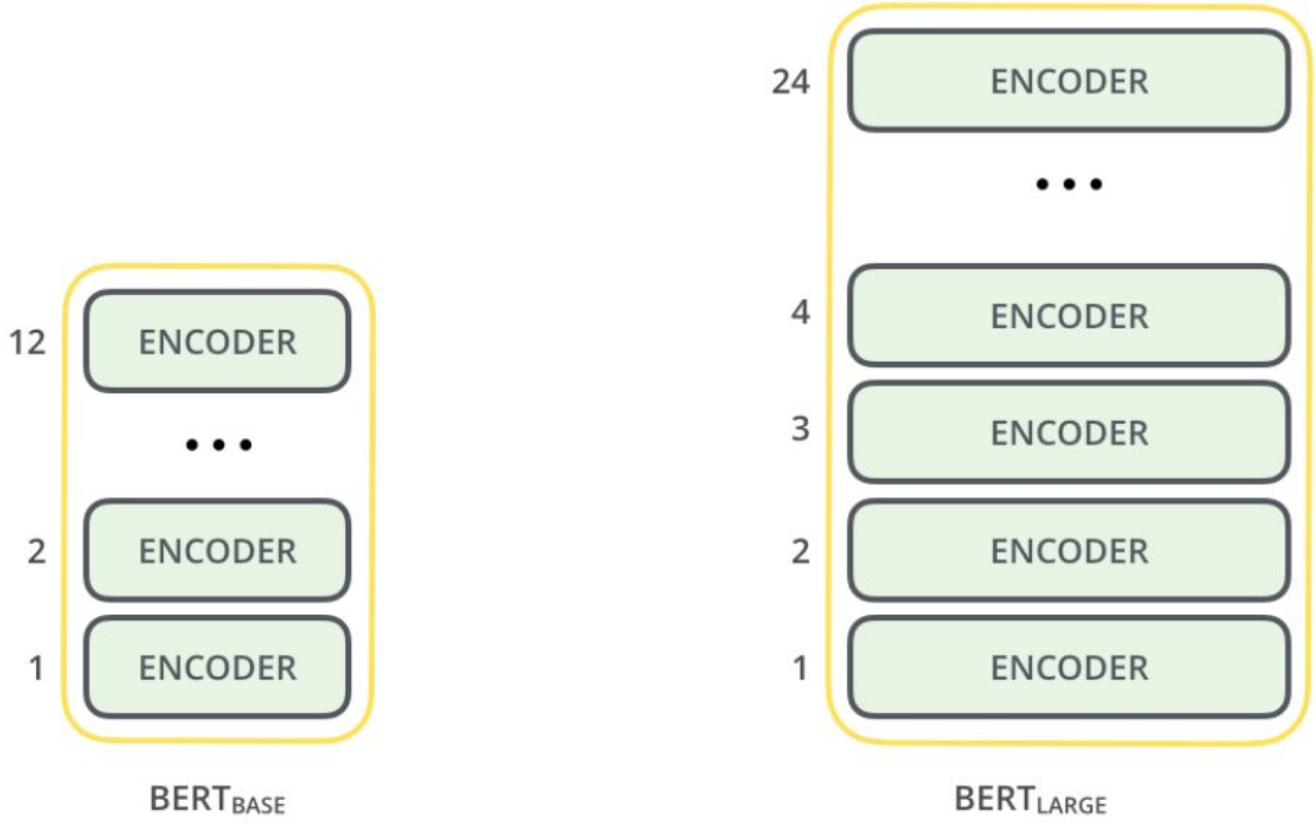
BERT

Bidirectional Encoder Representations from Transformers

BERT


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

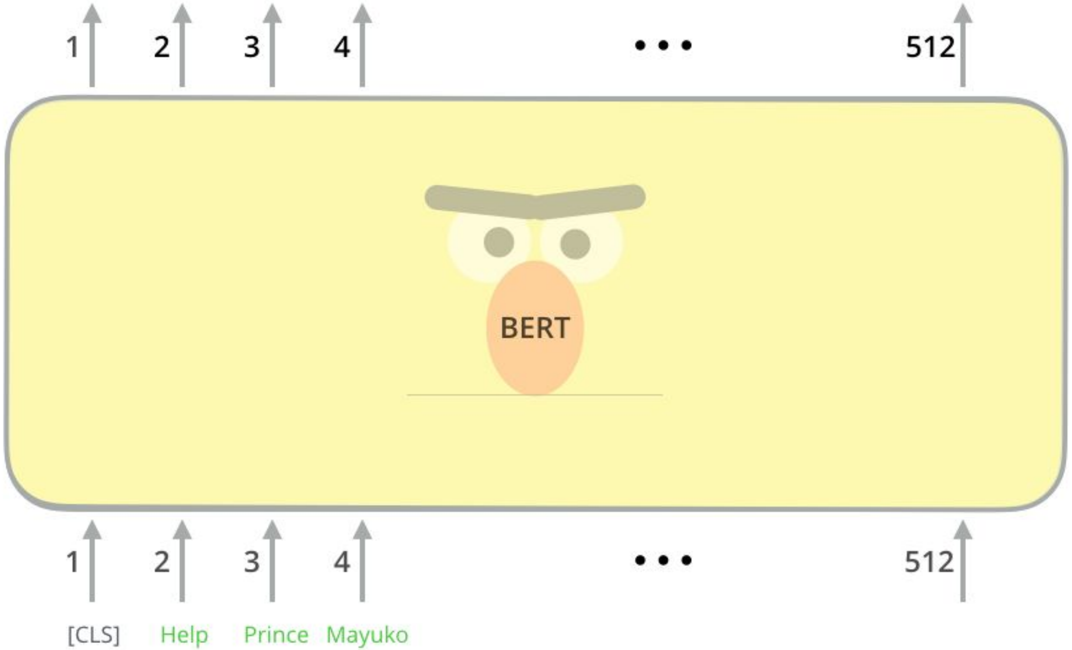
BERT: base and large



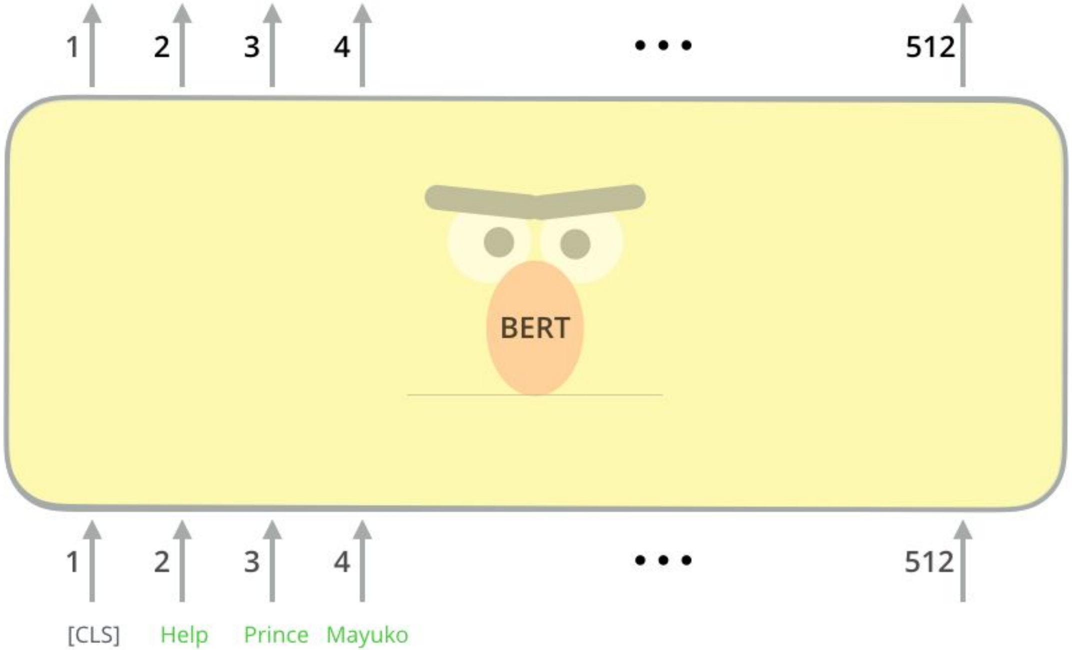
BERT vs. Transformer

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

Model inputs

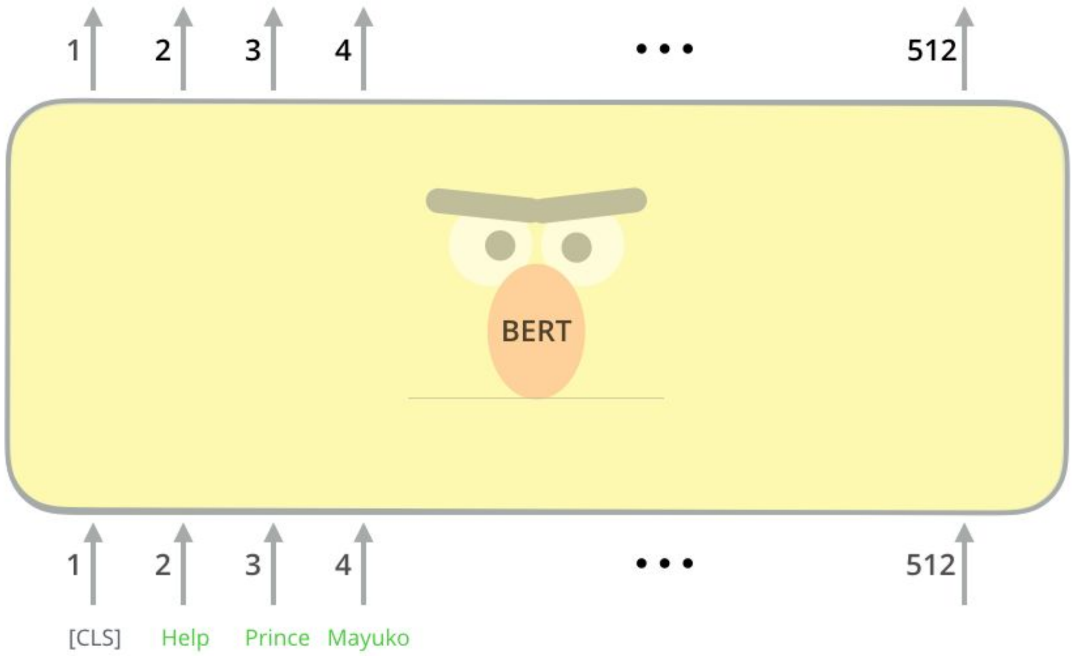


Model inputs



Identical to the Transformer up until this point

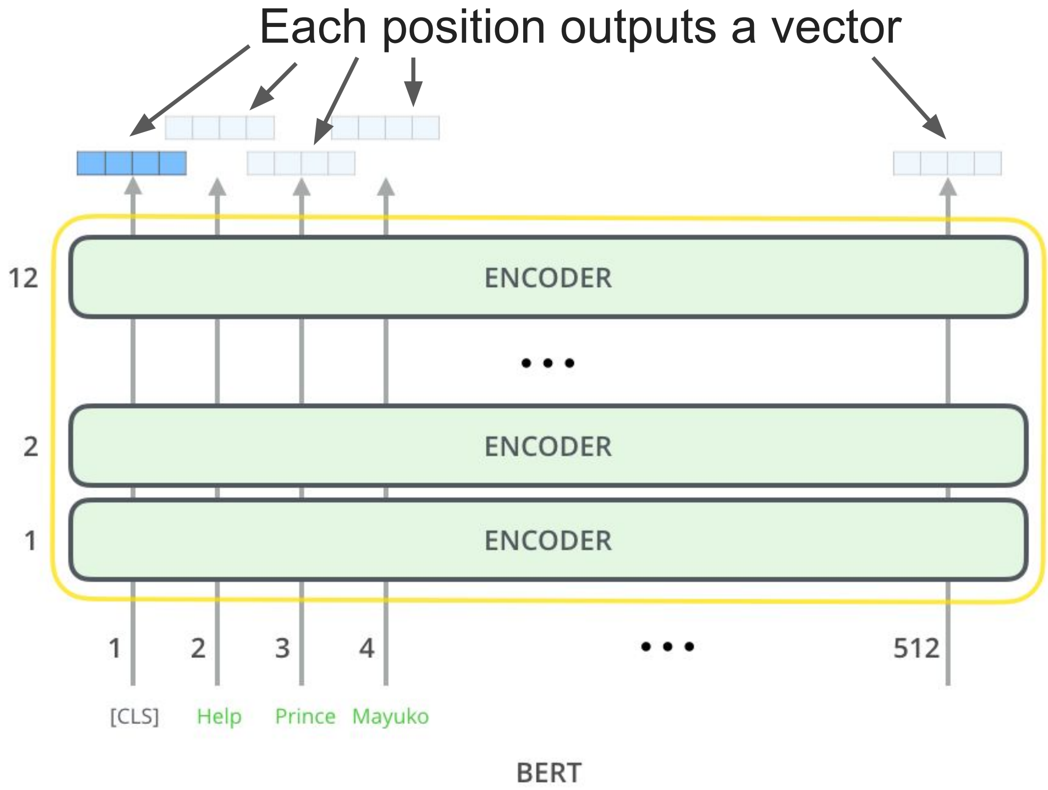
Model inputs



Identical to the Transformer up until this point

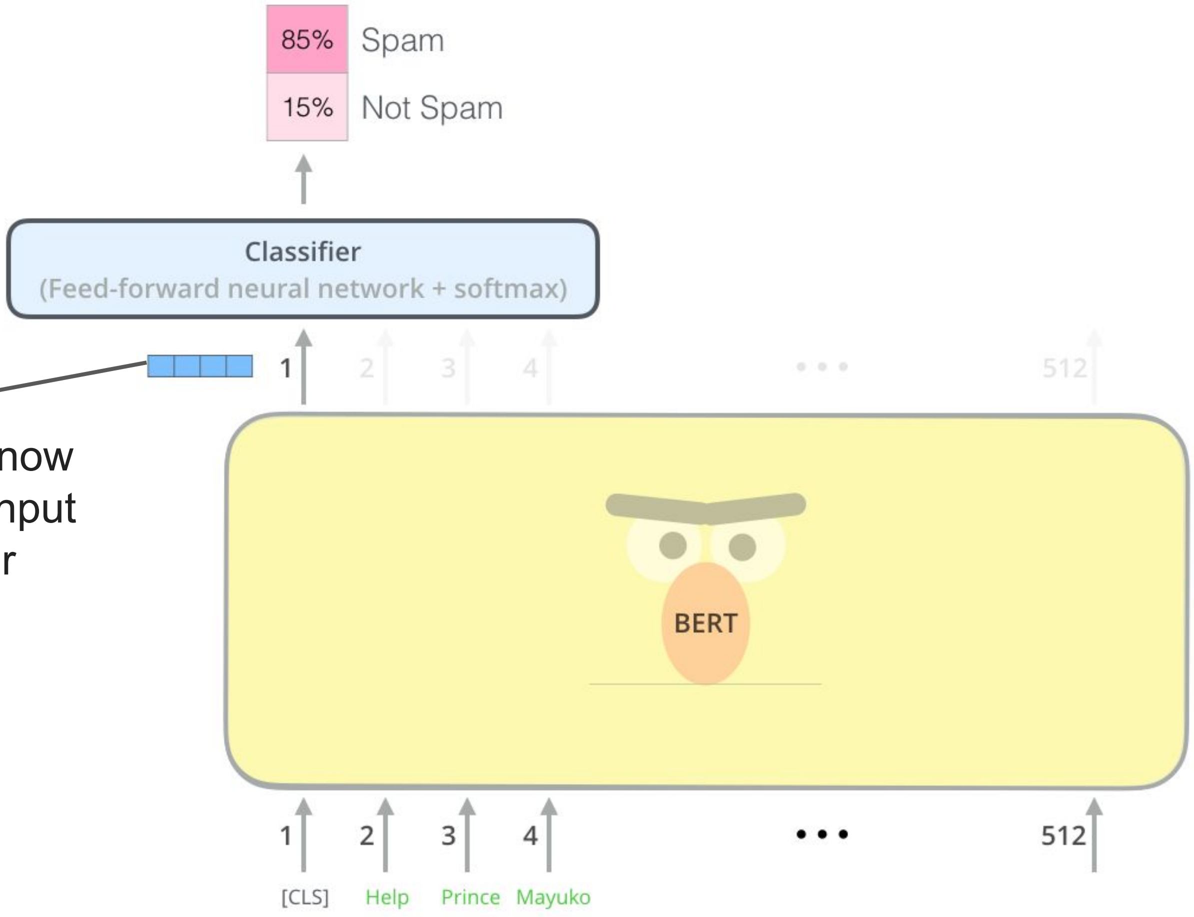
Why is BERT so special?

Model outputs



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

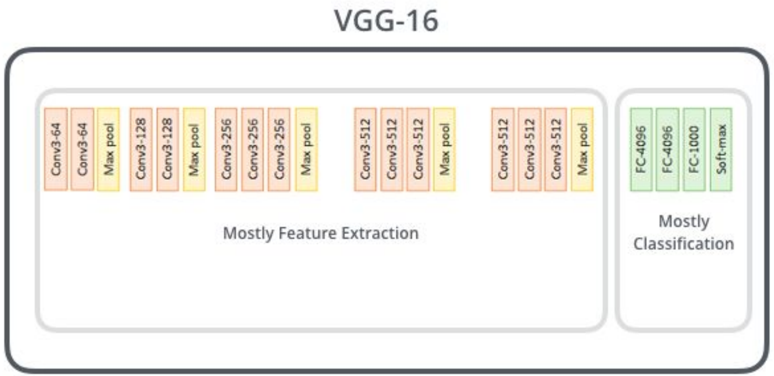
Model inputs



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

Similar to CNN concept!

Input
Features



Output
Prediction

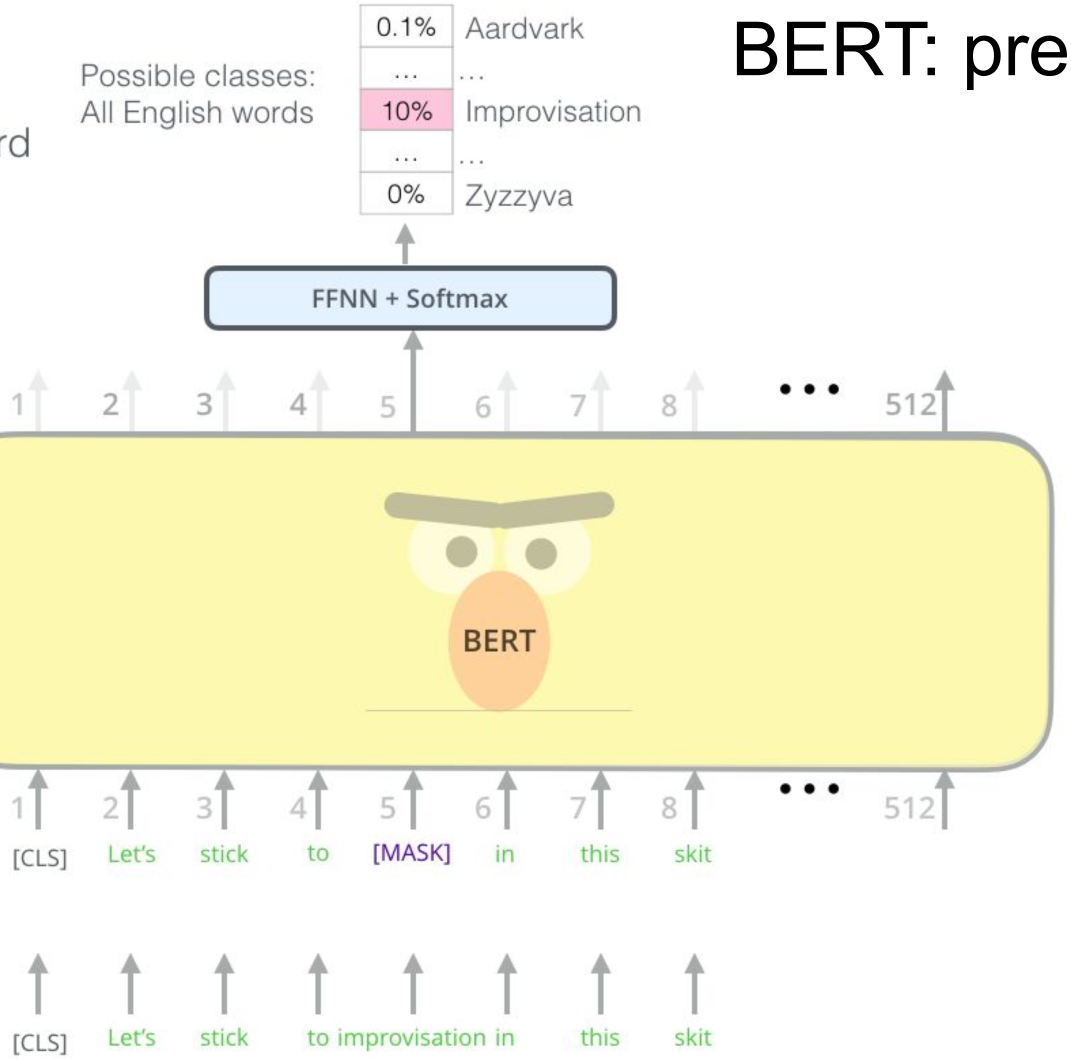
0.2%	Kit fox
0.1%	English setter
95%	Egyptian cat
1%	Great Dane
	...
0%	Hotdog

BERT: pre-training

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

Randomly mask 15% of tokens

Input



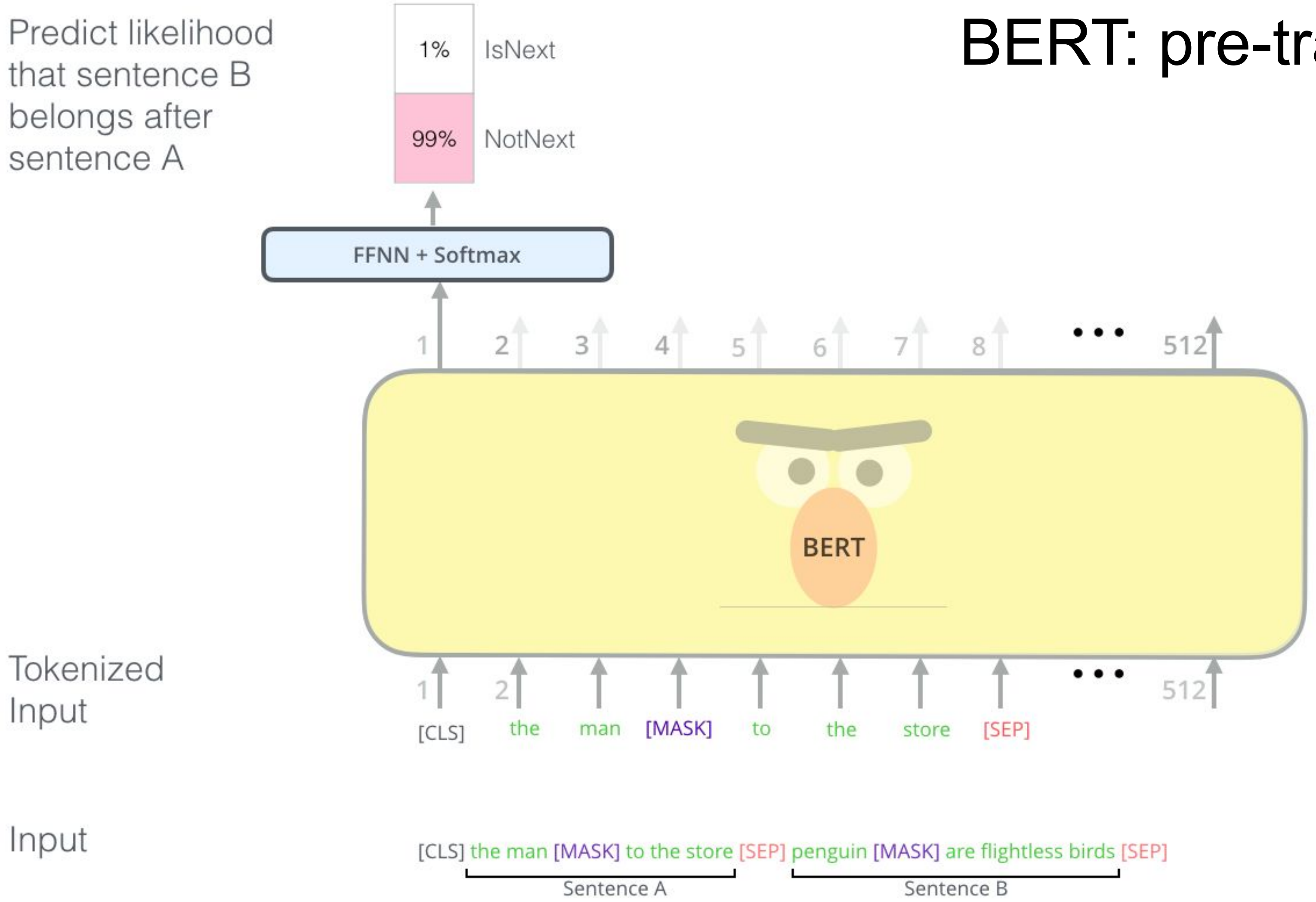
BERT: pre-training

- “Masked Language Model” approach

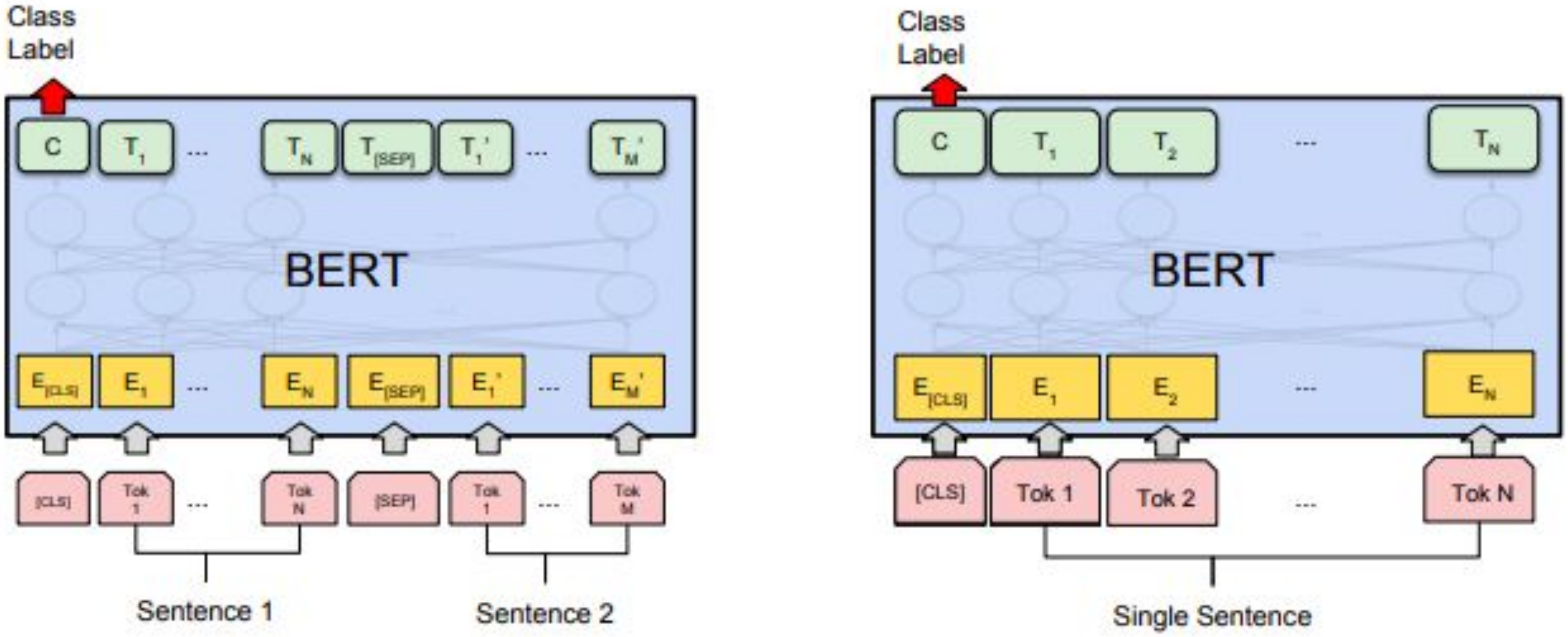
- To make BERT better at handling relationships between multiple sentences, the pre-training process includes an additional task:

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

BERT: pre-training



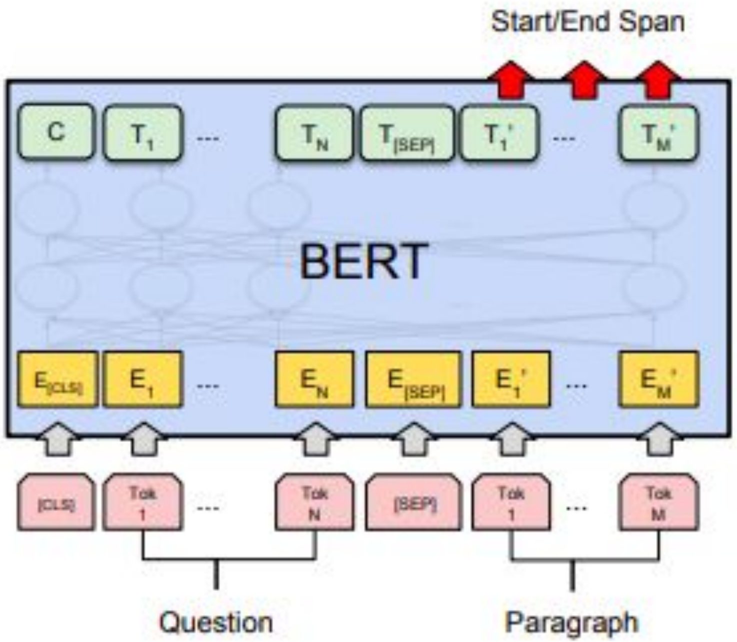
BERT: fine-tuning for different tasks



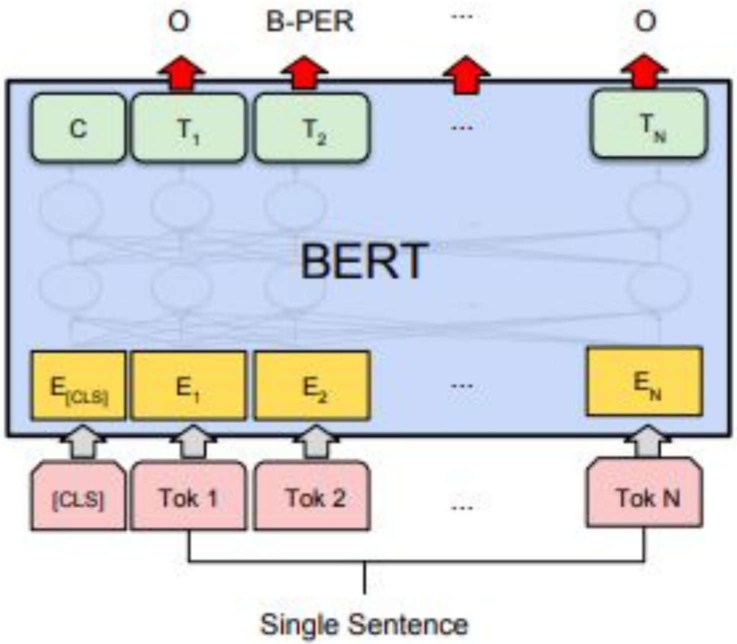
(a) Sentence Pair Classification Tasks: MNLI, QQP, QNLI, STS-B, MRPC, RTE, SWAG

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

BERT: fine-tuning for different tasks

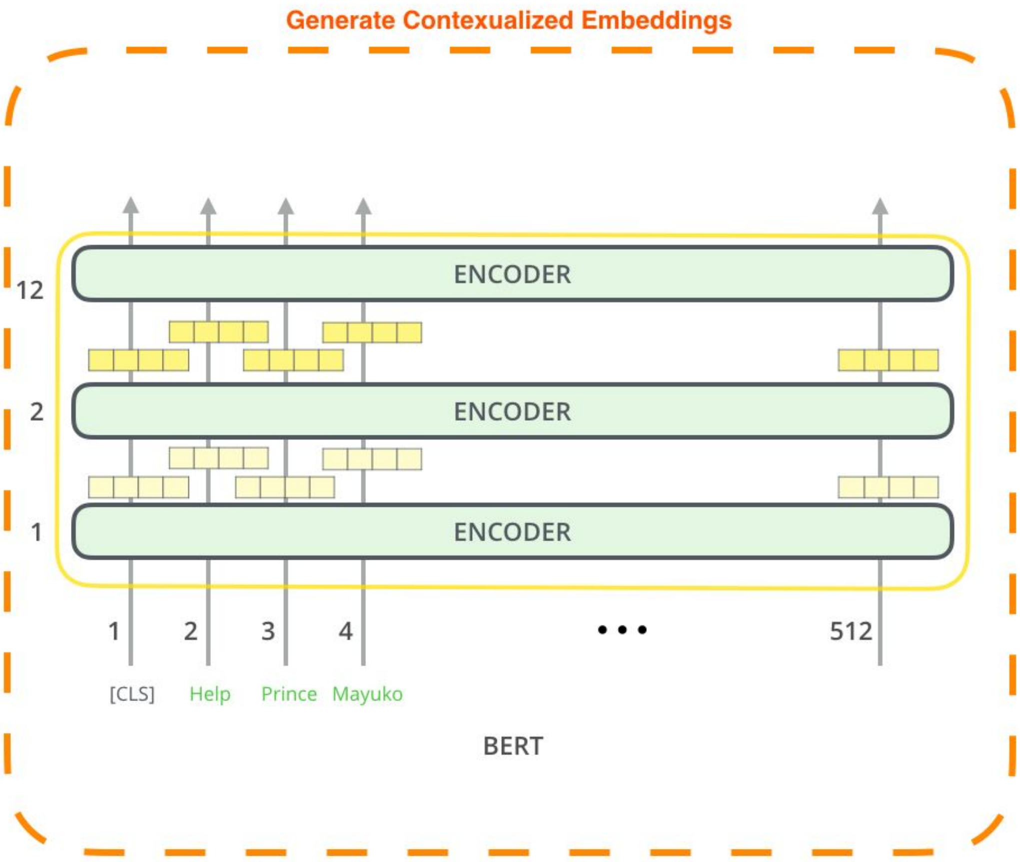


(c) Question Answering Tasks:
SQuAD v1.1

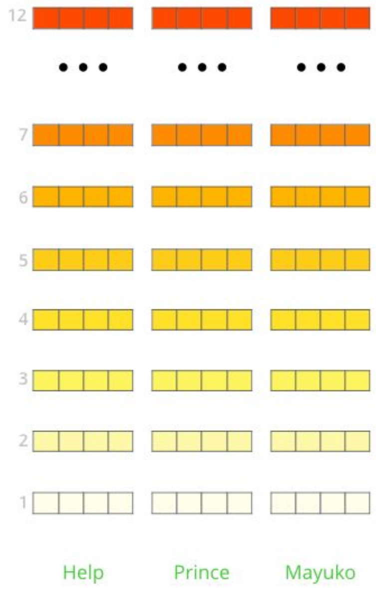


(d) Single Sentence Tagging Tasks:
CoNLL-2003 NER

BERT for feature extraction

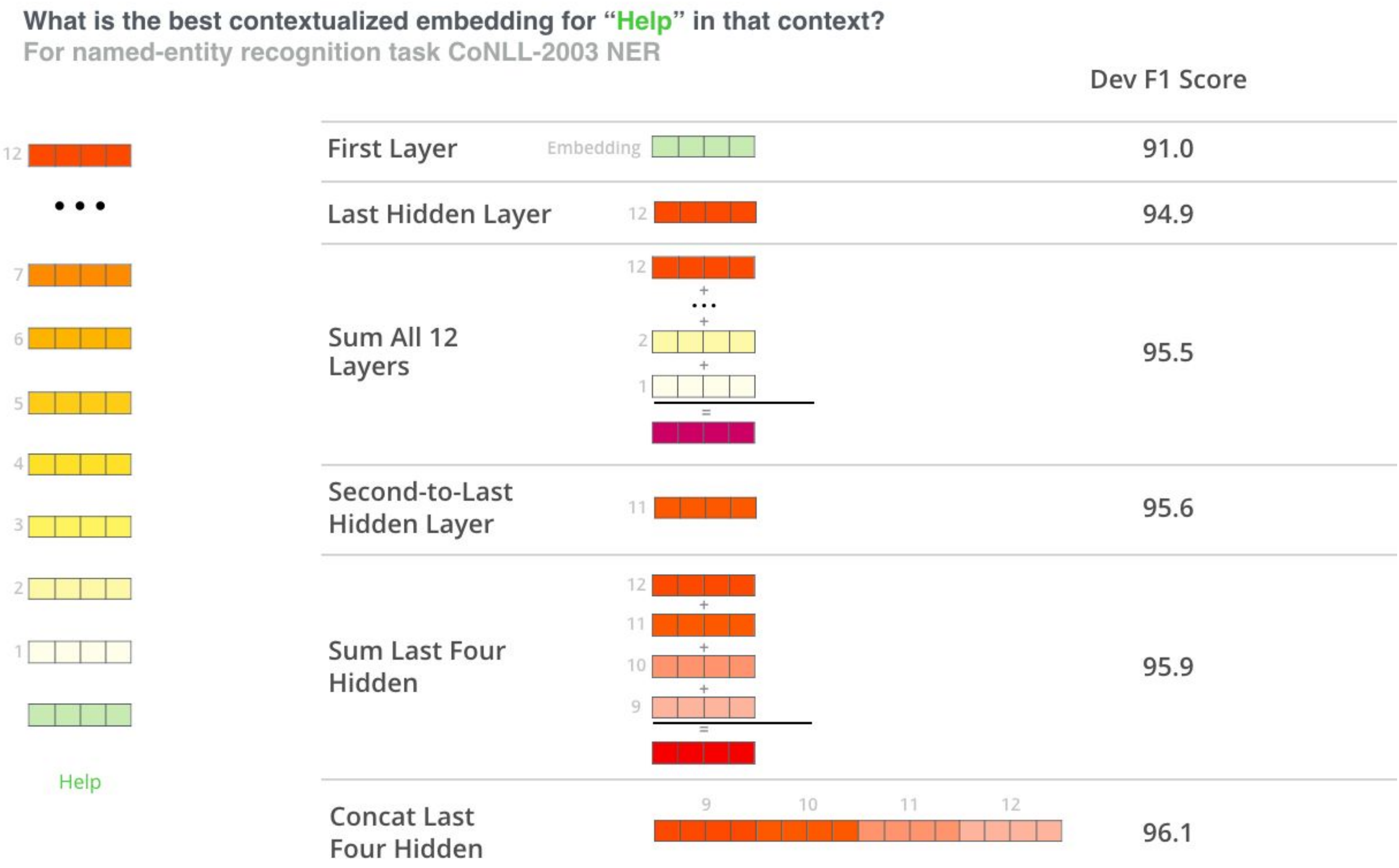


The output of each encoder layer along each token's path can be used as a feature representing that token.



But which one should we use?

BERT for feature extraction

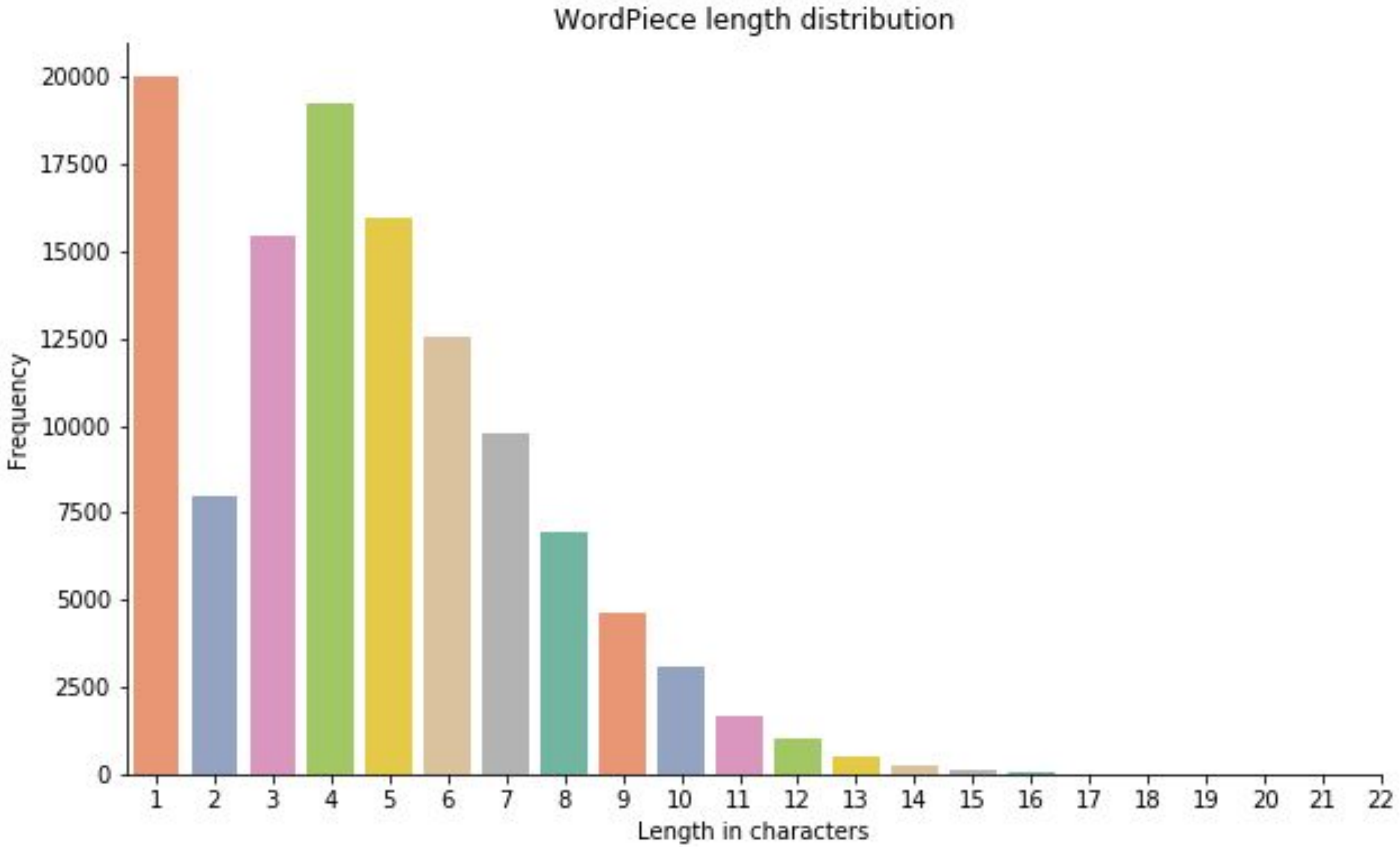


BERT: tokenization

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

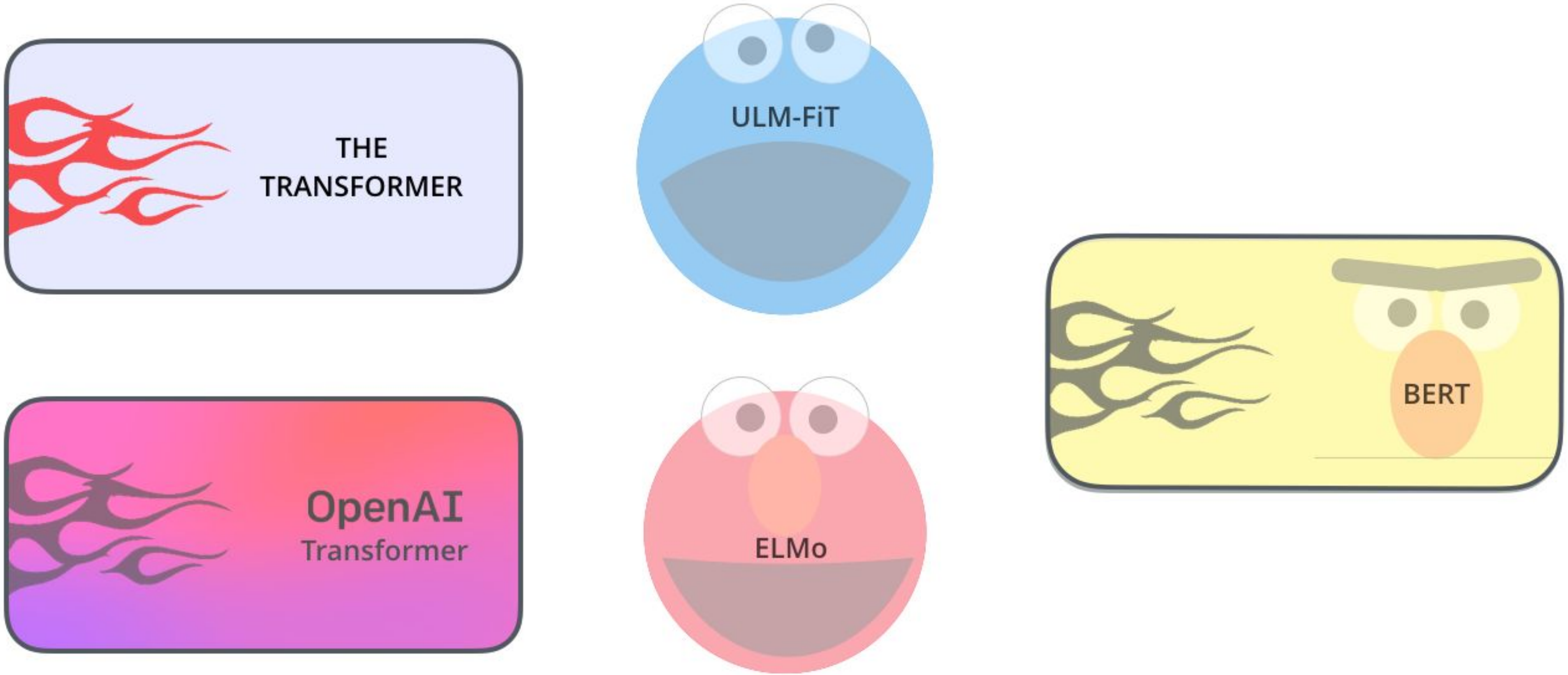
- Single model for 104 languages with a large shared vocabulary (119,547 WordPiece model)
- Non-word-initial units are prefixed with ##
- The first 106 symbols: constants like PAD and UNK
- 36.5% of the vocabulary are non-initial word pieces
- The alphabet consists of 9,997 unique characters that are 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.

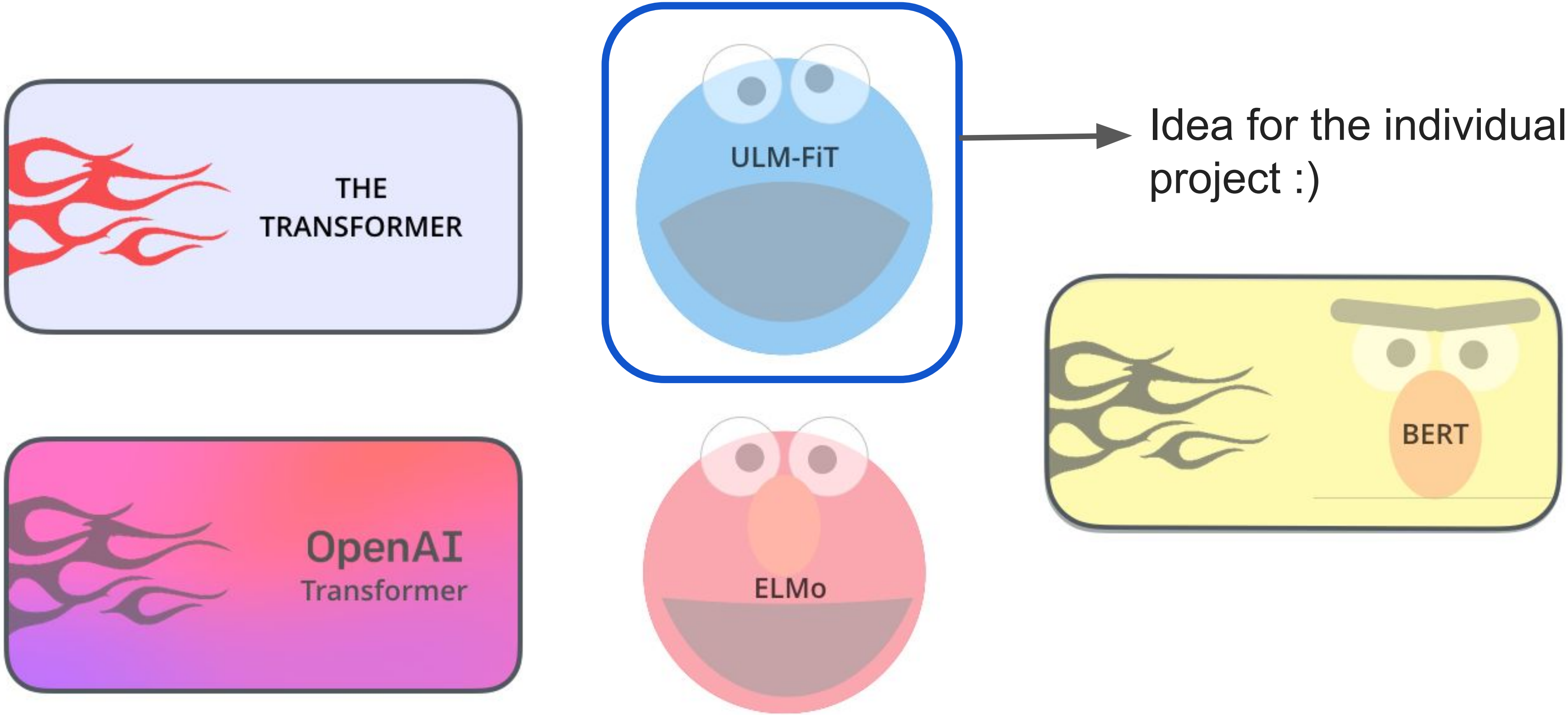
BERT: tokenization



BERT: overview

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


THE
TRANSFORMER
OpenAI
Transformer
ULM-FiT
ELMo
BERT


THE TRANSFORMER
ULM-FiT
Idea for the individual project :)
OpenAI Transformer
ELMo
BERT