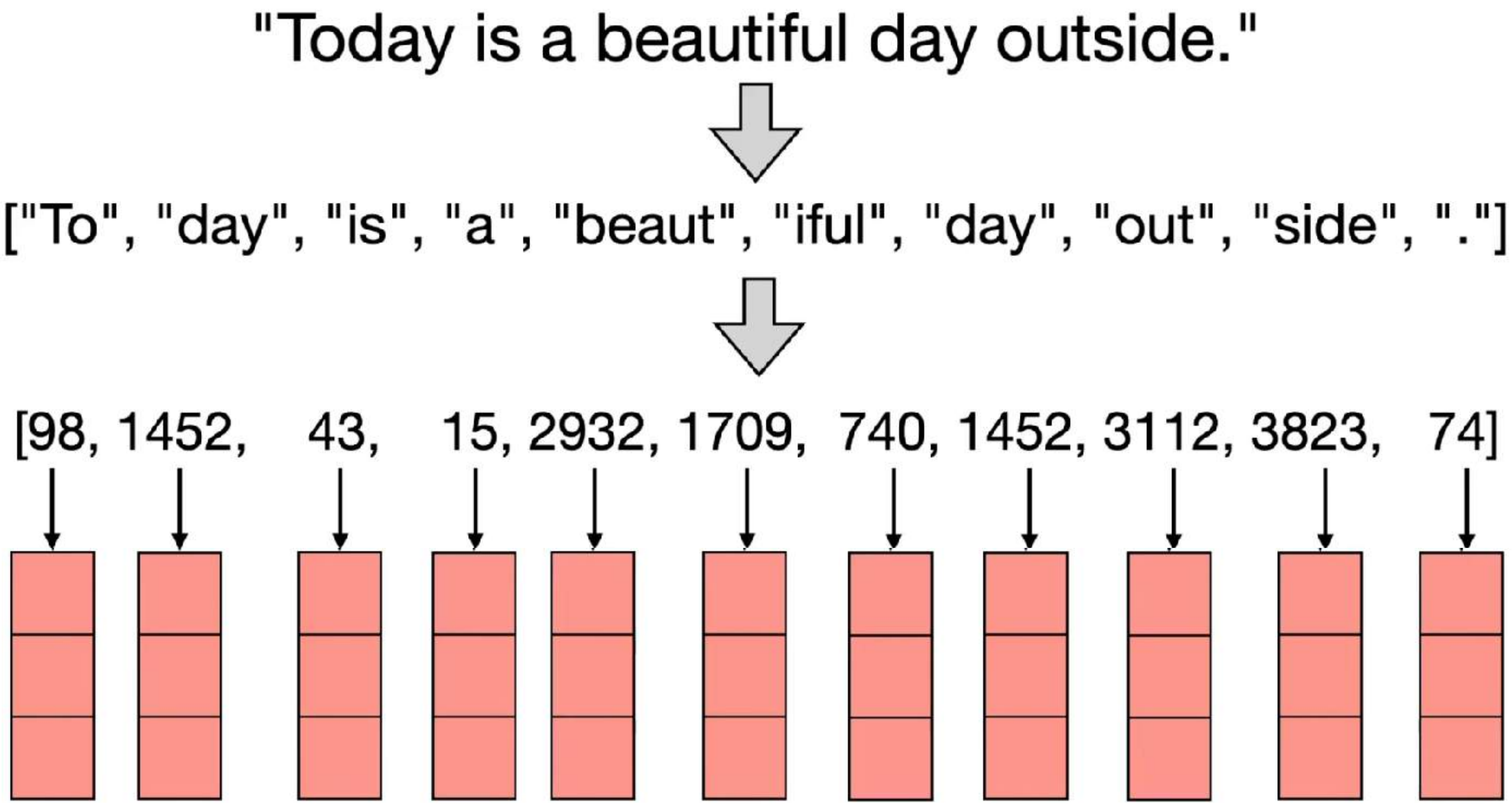
Tokenisation for Transformers & Contextual Embeddings

Introduction to NLP

Rahul Mishra
IIIT-Hyderabad
March 22, 2024

Byte Pair Encoding



Byte Pair Encoding

Instead of

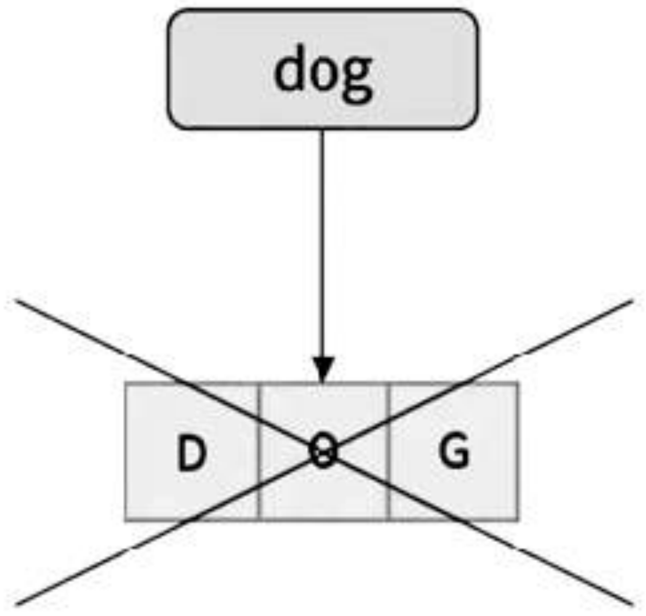
- white-space segmentation
- single-character segmentation

Use the data to tell us how to tokenize.

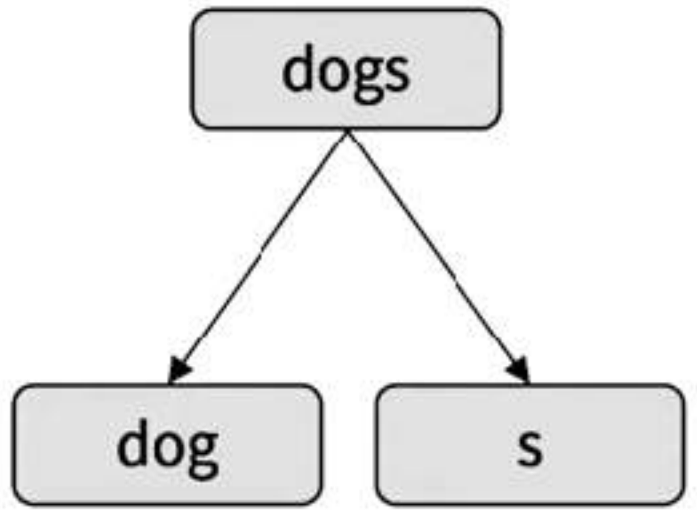
Subword tokenization (because tokens can be parts of words as well as whole words)

Byte Pair Encoding

Frequently used words should not be split into smaller subwords



Rare words should be decomposed into meaningful subwords.



Word-based tokenization

Very large vocabularies

Large quantity of out-of-vocabulary tokens

Loss of meaning across very similar words

Character-based tokenization

Very long sequences

Less meaningful individual tokens

Byte Pair Encoding

Three common algorithms:

- **Byte-Pair Encoding (BPE)** (Sennrich et al., 2016)
- **Unigram language modeling tokenization** (Kudo, 2018)
- **WordPiece** (Schuster and Nakajima, 2012)

All have 2 parts:

- A token **learner** that takes a raw training corpus and induces a vocabulary (a set of tokens).
- A token **segmenter** that takes a raw test sentence and tokenizes it according to that vocabulary

Byte Pair Encoding

Let vocabulary be the set of all individual characters

= {A, B, C, D,…, a, b, c, d….}

Repeat:

- Choose the two symbols that are most frequently adjacent in the training corpus (say 'A', 'B')
- Add a new merged symbol 'AB' to the vocabulary
- Replace every adjacent 'A' 'B' in the corpus with 'AB'.

Until *k* merges have been done.

BPE Token Learning Algo

function BYTE-PAIR ENCODING(strings C, number of merges k) **returns** vocab V

$V \leftarrow$ all unique characters in C # initial set of tokens is characters
for $i = 1$ **to** k **do** # merge tokens til k times
  $t_L, t_R \leftarrow$ Most frequent pair of adjacent tokens in C
  $t_{NEW} \leftarrow t_L + t_R$ # make new token by concatenating
  $V \leftarrow V + t_{NEW}$ # update the vocabulary
  Replace each occurrence of t_L, t_R in C with t_{NEW} # and update the corpus
return V

BPE Token Learner

Original (very fascinating🙄) corpus:

low low low low low lowest lowest newer newer newer newer newer newer wider wider wider new new

Add end-of-word tokens, resulting in this vocabulary:

vocabulary

_, d, e, i, l, n, o, r, s, t, w

corpus representation

5	l o w _
2	l o w e s t _
6	n e w e r _
3	w i d e r _
2	n e w _

BPE Token Learner

corpus		vocabulary
5	l o w _	_, d, e, i, l, n, o, r, s, t, w
2	l o w e s t _	
6	n e w e r _	
3	w i d e r _	
2	n e w _	

Merge e r to er

corpus		vocabulary
5	l o w _	_, d, e, i, l, n, o, r, s, t, w, er
2	l o w e s t _	
6	n e w er _	
3	w i d er _	
2	n e w _	

BPE Token Learner

corpus

5 l o w _
2 l o w e s t _
6 n e w er_
3 w i d er_
2 n e w _

vocabulary

_, d, e, i, l, n, o, r, s, t, w, er, er_

Merge n e to ne

corpus

5 l o w _
2 l o w e s t _
6 ne w er_
3 w i d er_
2 ne w _

vocabulary

_, d, e, i, l, n, o, r, s, t, w, er, er_, ne

BPE Token Learner

The next merges are:

Merge	Current Vocabulary
(ne, w)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new
(l, o)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new, lo
(lo, w)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new, lo, low
(new, er_)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new, lo, low, newer_
(low, _)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new, lo, low, newer_, low_

BPE Token Segmenter

On the test data, run each merge learned from the training data:

- Greedily
- In the order we learned them
- (test frequencies don't play a role)

So: merge every e r to er, then merge er _ to er_, etc.

Result:

- Test set "n e w e r _" would be tokenized as a full word
- Test set "l o w e r _" would be two tokens: "low er_"

BPE Properties

Usually include frequent words

And frequent subwords

- Which are often morphemes like *-est* or *–er*

A **morpheme** is the smallest meaning-bearing unit of a language

- *unlikeliest* has 3 morphemes *un-*, *likely*, and *-est*

WordPiece and SentencePiece

$$\frac{\boxed{P(tok_1, tok_2)}}{P(tok_1)P(tok_2)} \rightarrow \text{Pair probability}$$

Japanese

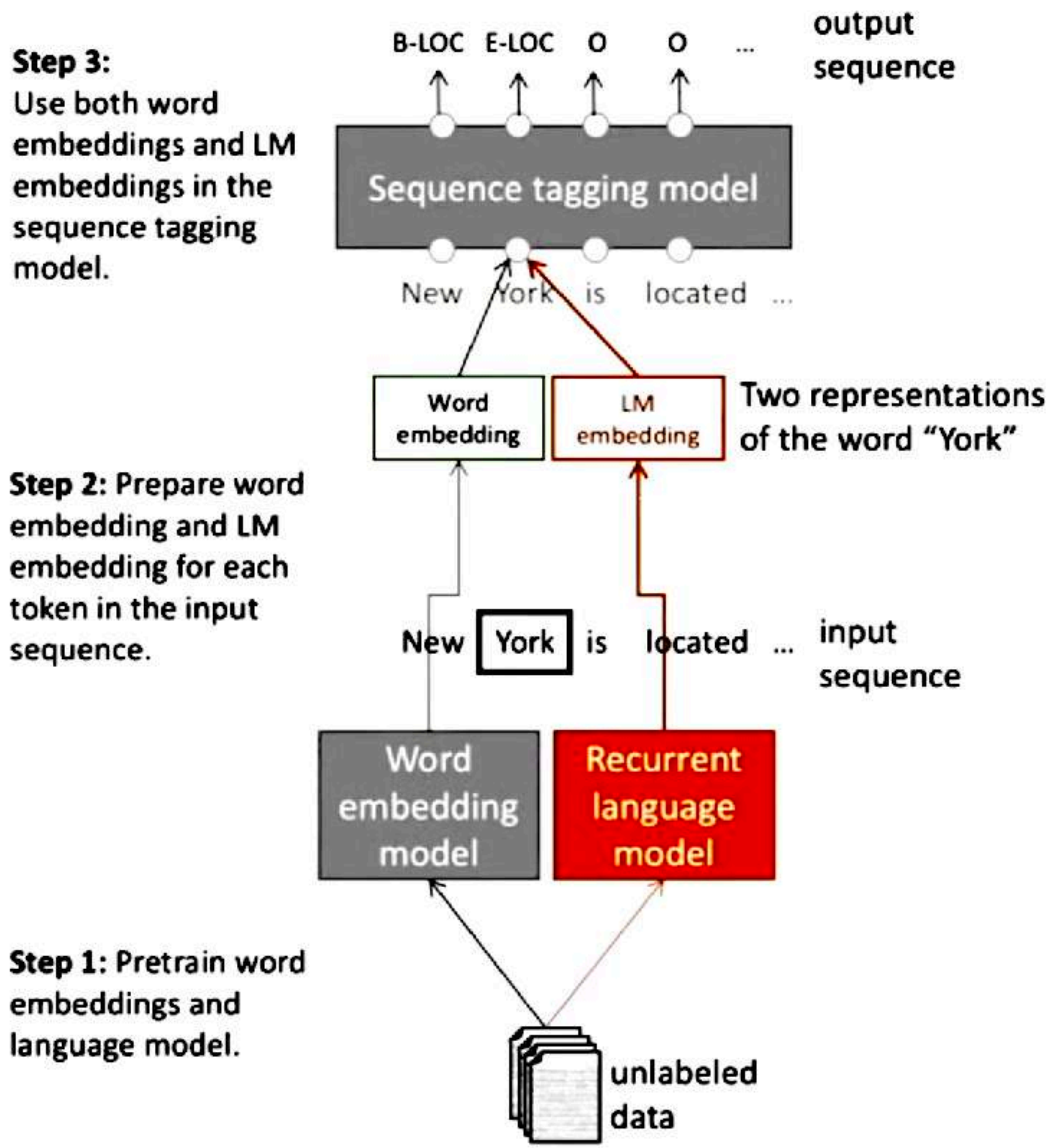
トークナイザーについて楽しく学んでいただければ幸いです

Chinese

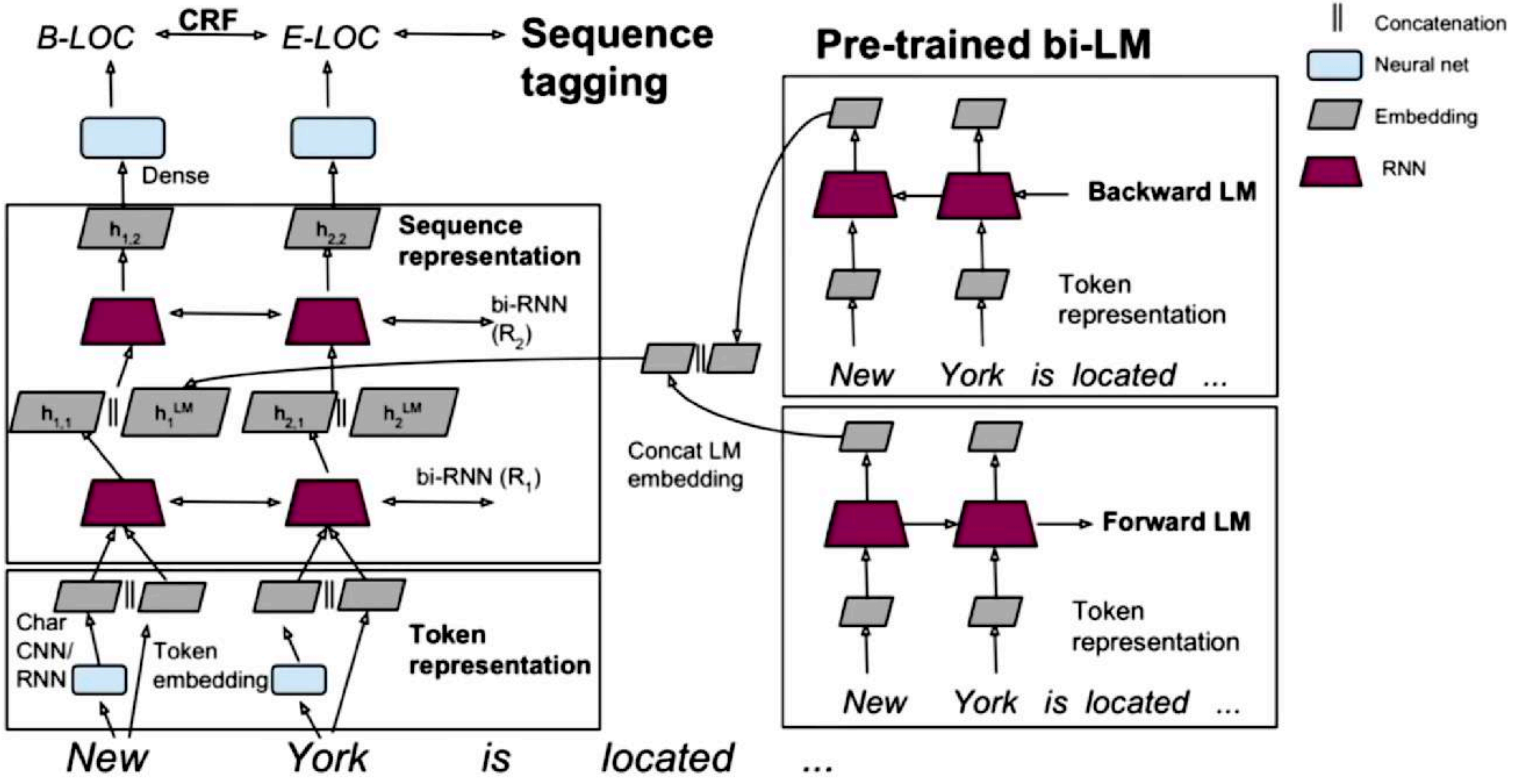
我希望您在学习分词器过程中度过愉快的时光

How do you pretokenize such languages?

TagLM (Pre-ELMo) Peters et al. 2017



TagLM (Pre-ELMo) Peters et al. 2017



$$\mathbf{h}_{k,1} = [\overrightarrow{\mathbf{h}}_{k,1}; \overleftarrow{\mathbf{h}}_{k,1}; \mathbf{h}_k^{LM}].$$

ELMo: Embeddings from Language Models

Train a bidirectional LM

Aim at performant but not overly large LM:

- Use 2 biLSTM layers
- Use character CNN to build initial word representation (only)
  - 2048 char n-gram filters and 2 highway layers, 512 dim projection
- User 4096 dim hidden/cell LSTM states with 512 dim projections to next input
- Use a residual connection
- Tie parameters of token input and output (softmax) and tie these between forward and backward LMs

Best Paper award at NAACL 2018

ELMo: Embeddings from Language Models

- ELMo learns task-specific combination of biLM representations
- This is an innovation that improves on just using top layer of LSTM stack

$$\begin{aligned} R_k &= \{\mathbf{x}_k^{LM}, \overrightarrow{\mathbf{h}}_{k,j}^{LM}, \overleftarrow{\mathbf{h}}_{k,j}^{LM} \mid j = 1, \ldots, L\} \\ &= \{\mathbf{h}_{k,j}^{LM} \mid j = 0, \ldots, L\}, \end{aligned}$$

$$\mathbf{ELMo}_k^{task} = E(R_k; \Theta^{task}) = \gamma^{task} \sum_{j=0}^{L} s_j^{task} \mathbf{h}_{k,j}^{LM}$$

- γ^{task} scales overall usefulness of ELMo to task;
- $\mathbf{s}^{\text{task}}$ are softmax-normalized mixture model weights

ELMo

ELMo	ELMo in BiLSTM	2018	92.22
TagLM Peters	LSTM BiLM in BiLSTM tagger	2017	91.93
Ma + Hovy	BiLSTM + char CNN + CRF layer	2016	91.21
Tagger Peters	BiLSTM + char CNN + CRF layer	2017	90.87
Ratinov + Roth	Categorical CRF+Wikipeda+word cls	2009	90.80
Finkel et al.	Categorical feature CRF	2005	86.86
IBM Florian	Linear/softmax/TBL/HMM ensemble, gazettes++	2003	88.76
Stanford	MEMM softmax markov model	2003	86.07

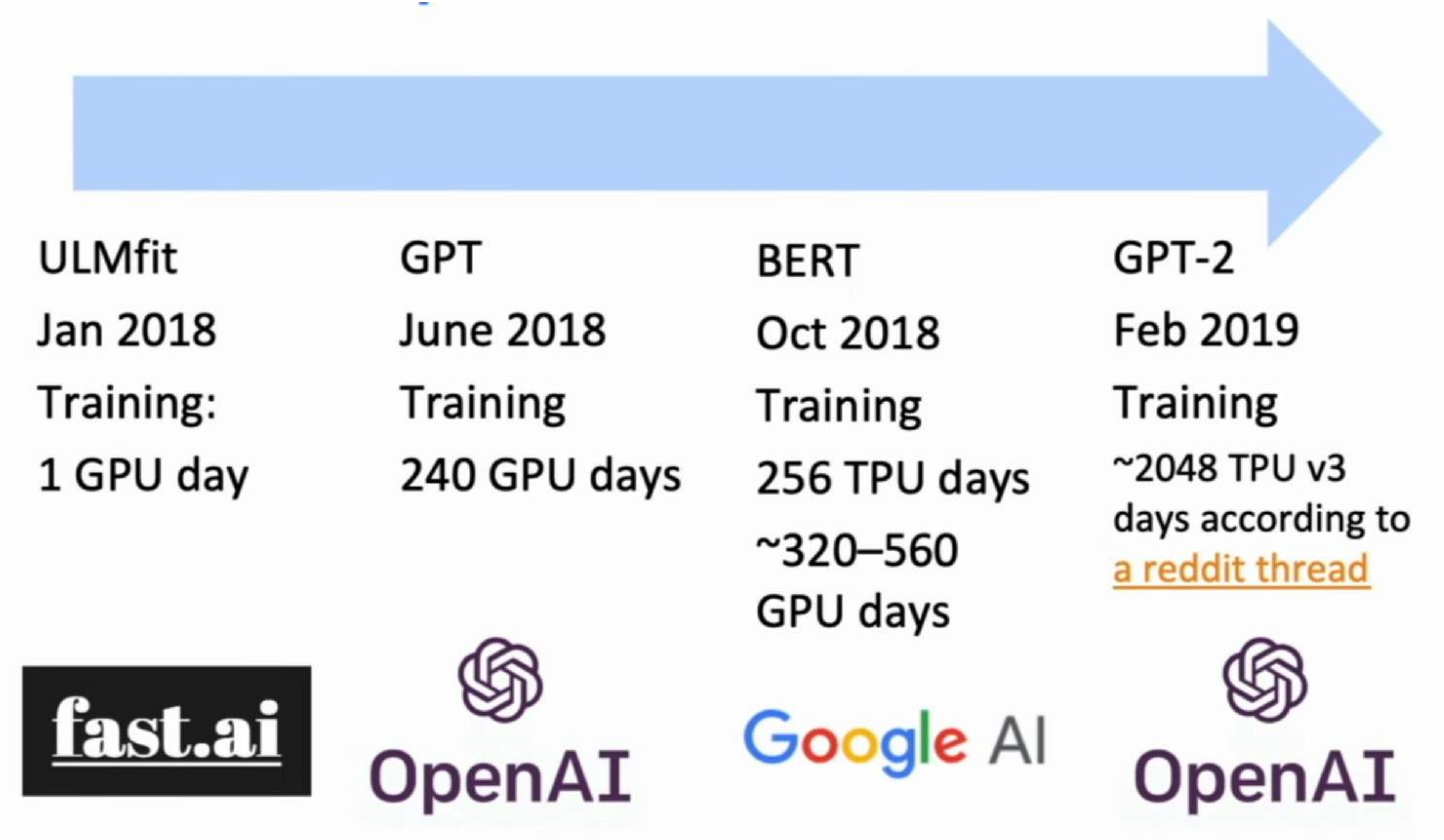
TASK	PREVIOUS SOTA		OUR BASELINE	ELMO + BASELINE	INCREASE (ABSOLUTE/ RELATIVE)
SQuAD	Liu et al. (2017)	84.4	81.1	85.8	4.7 / 24.9%
SNLI	Chen et al. (2017)	88.6	88.0	88.7 ± 0.17	0.7 / 5.8%
SRL	He et al. (2017)	81.7	81.4	84.6	3.2 / 17.2%
Coref	Lee et al. (2017)	67.2	67.2	70.4	3.2 / 9.8%
NER	Peters et al. (2017)	91.93 ± 0.19	90.15	92.22 ± 0.10	2.06 / 21%
SST-5	McCann et al. (2017)	53.7	51.4	54.7 ± 0.5	3.3 / 6.8%

ELMo layers

The two biLSTM NLM layers have differentiated uses/meanings

- Lower layer is better for lower-level syntax, etc.
  - Part-of-speech tagging, syntactic dependencies, NER
- Higher layer is better for higher-level semantics
  - Sentiment, Semantic role labeling, question answering, SNLI

ULMfit


ULMfit
Jan 2018
Training:
1 GPU day
fast.ai
GPT
June 2018
Training
240 GPU days
OpenAI
BERT
Oct 2018
Training
256 TPU days
~320–560
GPU days
Google AI
GPT-2
Feb 2019
Training
~2048 TPU v3
days according to
a reddit thread
OpenAI

BERT

Problem: Language models only use left context *or* right context, but language understanding is bidirectional.

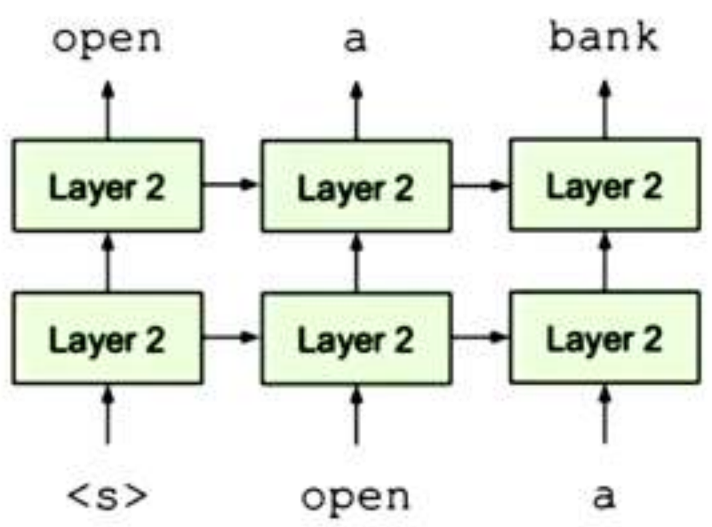
Why are LMs unidirectional?

Reason 1: Directionality is needed to generate a well-formed probability distribution.

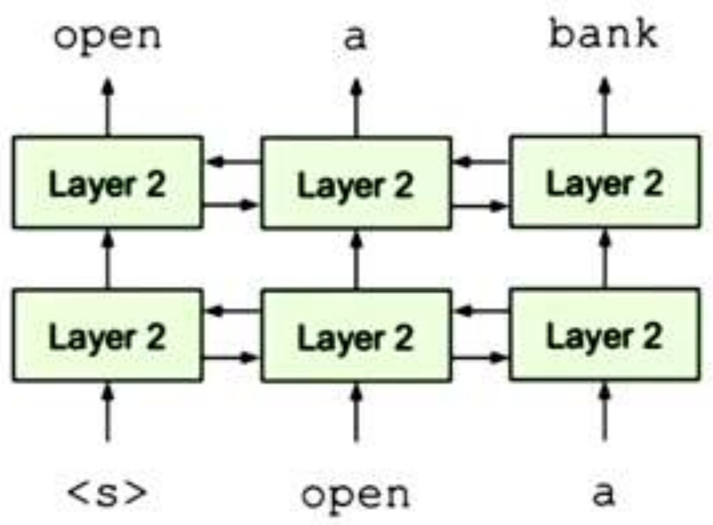
- We don't care about this.

Reason 2: Words can "see themselves" in a bidirectional encoder.

Unidirectional context
Build representation incrementally



Bidirectional context
Words can "see themselves"



BERT

Solution: Mask out *k*% of the input words, and then predict the masked words

- They always use *k* = 15%

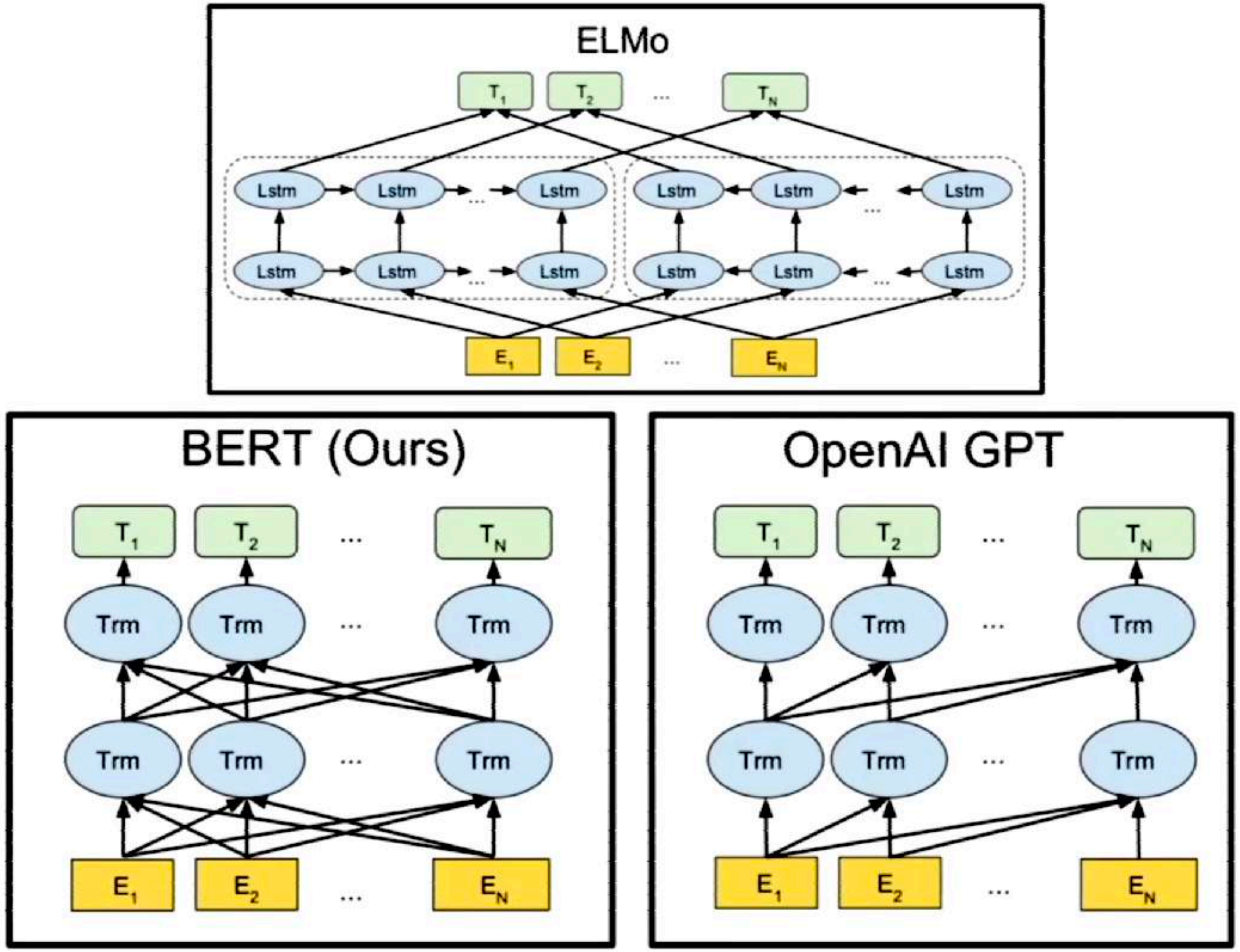
store ↑ gallon ↑

the man went to the [MASK] to buy a [MASK] of milk

Too little masking: Too expensive to train

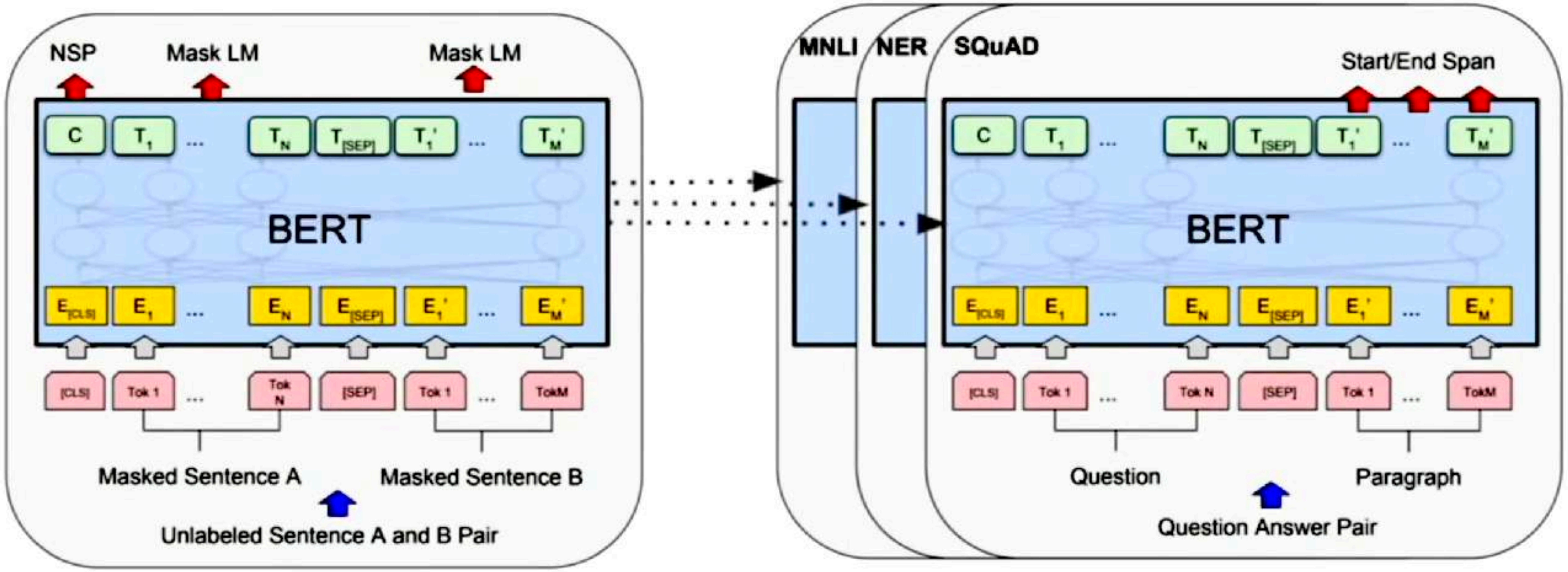
Too much masking: Not enough context

BERT



BERT Fine Tuning

- Simply learn a classifier built on the top layer for each task that you fine tune for



Pre-training

Fine-Tuning

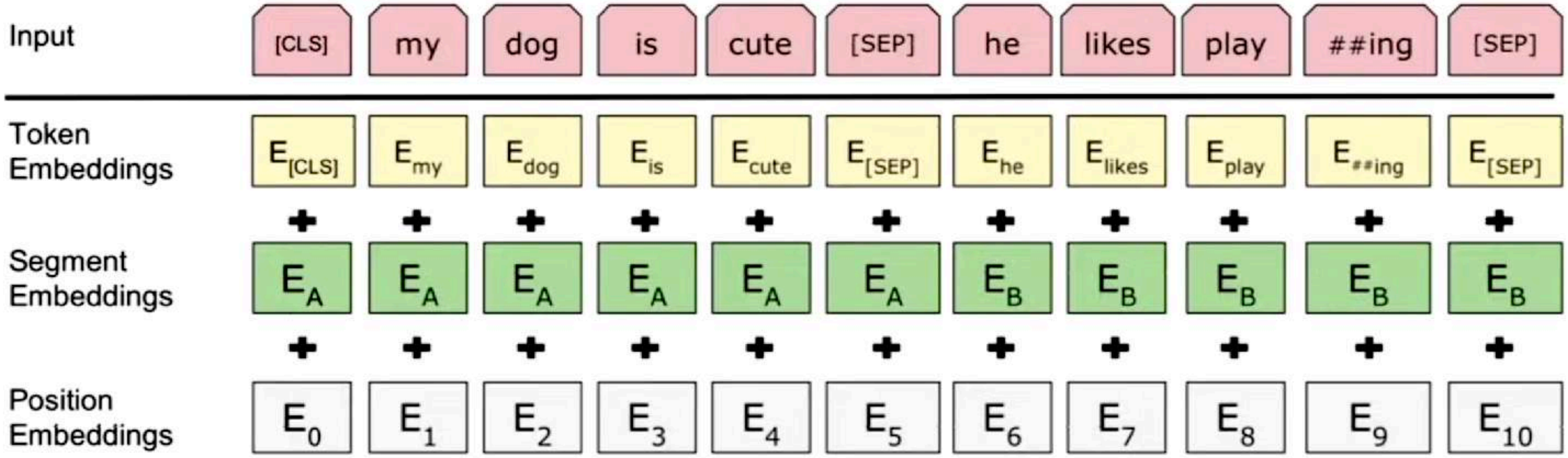
BERT Next Sentence Prediction

- To learn *relationships* between sentences, predict whether Sentence B is actual sentence that proceeds Sentence A, or a random sentence

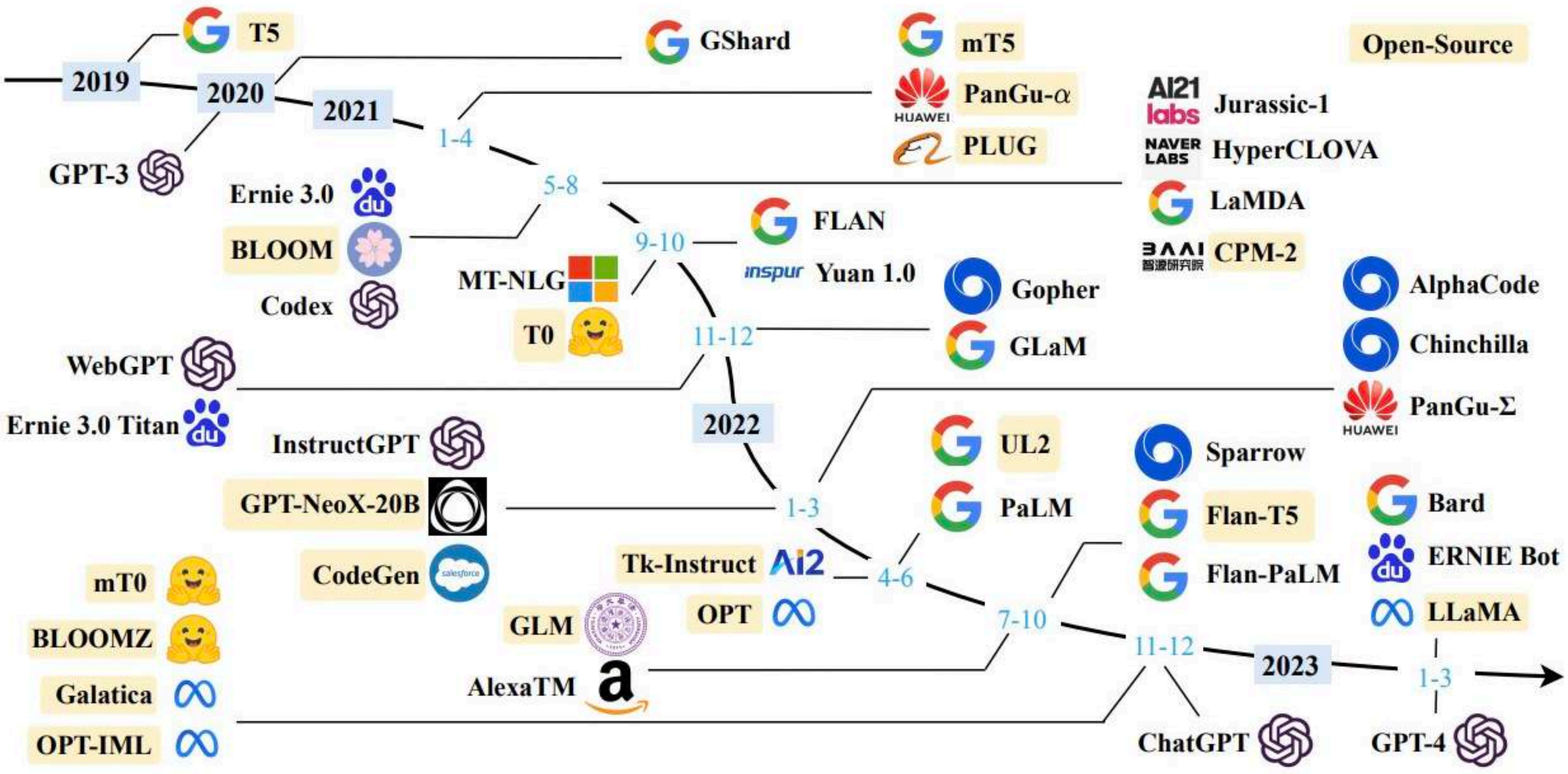
Sentence A = `The man went to the store.`
Sentence B = `He bought a gallon of milk.`
Label = `IsNextSentence`

Sentence A = `The man went to the store.`
Sentence B = `Penguins are flightless.`
Label = `NotNextSentence`

BERT Sent Pair Encoding



Token embeddings are word pieces
Learned segmented embedding represents each sentence
Positional embedding is as for other Transformer architectures


2019
2020
2021
1-4
5-8
9-10
11-12
2022
1-3
4-6
7-10
11-12
2023
1-3
T5
GPT-3
GShard
mT5
PanGu-α
HUAWEI
PLUG
Open-Source
Jurassic-1
AI21 labs
HyperCLOVA
NAVER LABS
LaMDA
CPM-2
Ernie 3.0
BLOOM
Codex
MT-NLG
T0
FLAN
Yuan 1.0
inspur
Gopher
GLaM
AlphaCode
Chinchilla
PanGu-Σ
HUAWEI
WebGPT
Ernie 3.0 Titan
InstructGPT
GPT-NeoX-20B
CodeGen
salesforce
UL2
PaLM
Sparrow
Flan-T5
Flan-PaLM
Bard
ERNIE Bot
LLaMA
Tk-Instruct
OPT
GLM
AlexaTM
mT0
BLOOMZ
Galatica
OPT-IML
ChatGPT
GPT-4