SFU NatLangLab

CMPT 413/713: Natural Language Processing

Contextualized Word Embeddings

Spring 2024

2024-02-12

Adapted from slides from Danqi Chen and Karthik Narasimhan
(with some content from slides from Chris Manning and Abigail See)

Overview

Contextualized Word Representations

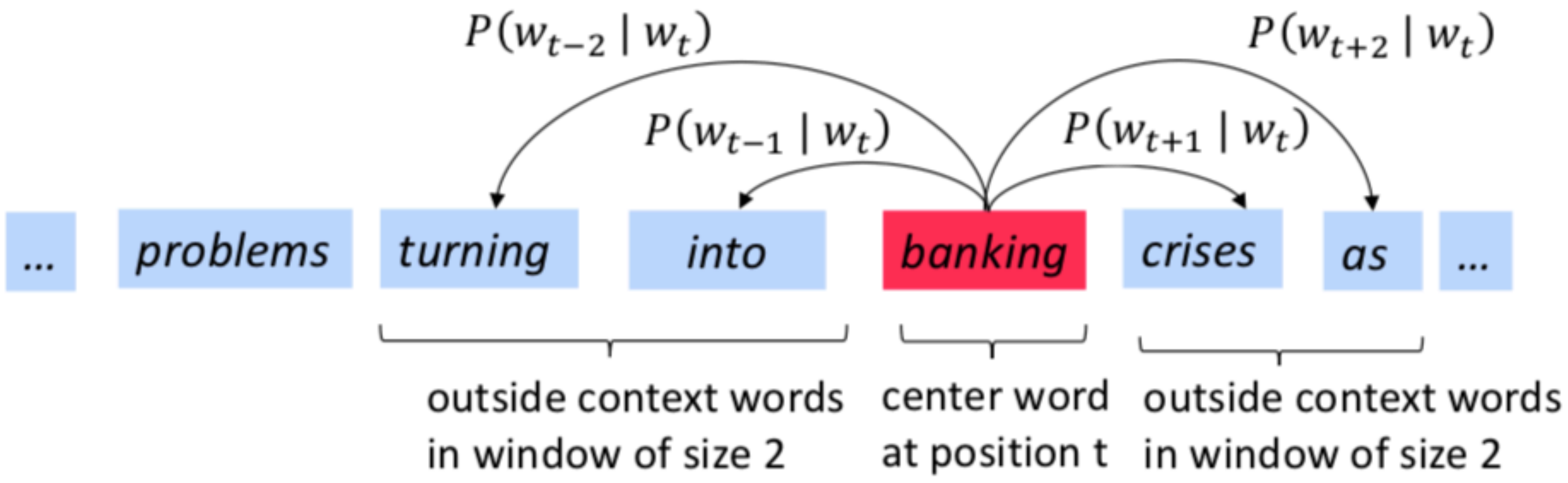
- ELMo = **E**mbeddings from **L**anguage **Mo**dels

Deep contextualized word representations
https://arxiv.org › cs ▾
by ME Peters - 2018 - Cited by 1683 - Related articles
Deep contextualized word representations. ... Our **word** vectors are learned functions of the internal states of a **deep** bidirectional language model (biLM), which is pre-trained on a large text corpus.

- BERT = **B**idirectional **E**ncoder **R**epresentations from **T**ransformers

BERT: Pre-training of Deep Bidirectional Transformers for ...
https://arxiv.org › cs ▾
by J Devlin - 2018 - Cited by 2259 - Related articles
Oct 11, 2018 - Unlike recent language representation models, **BERT** is designed to pre-train deep ... As a result, the pre-trained **BERT** model can be fine-tuned with just one additional output ... Which authors of this **paper** are endorsers?

Recap: word2vec



word = “sweden”

Word	Cosine distance
norway	0.760124
denmark	0.715460
finland	0.620022
switzerland	0.588132
belgium	0.585835
netherlands	0.574631
iceland	0.562368
estonia	0.547621
slovenia	0.531408

What's wrong with word2vec?

- One vector for each word type

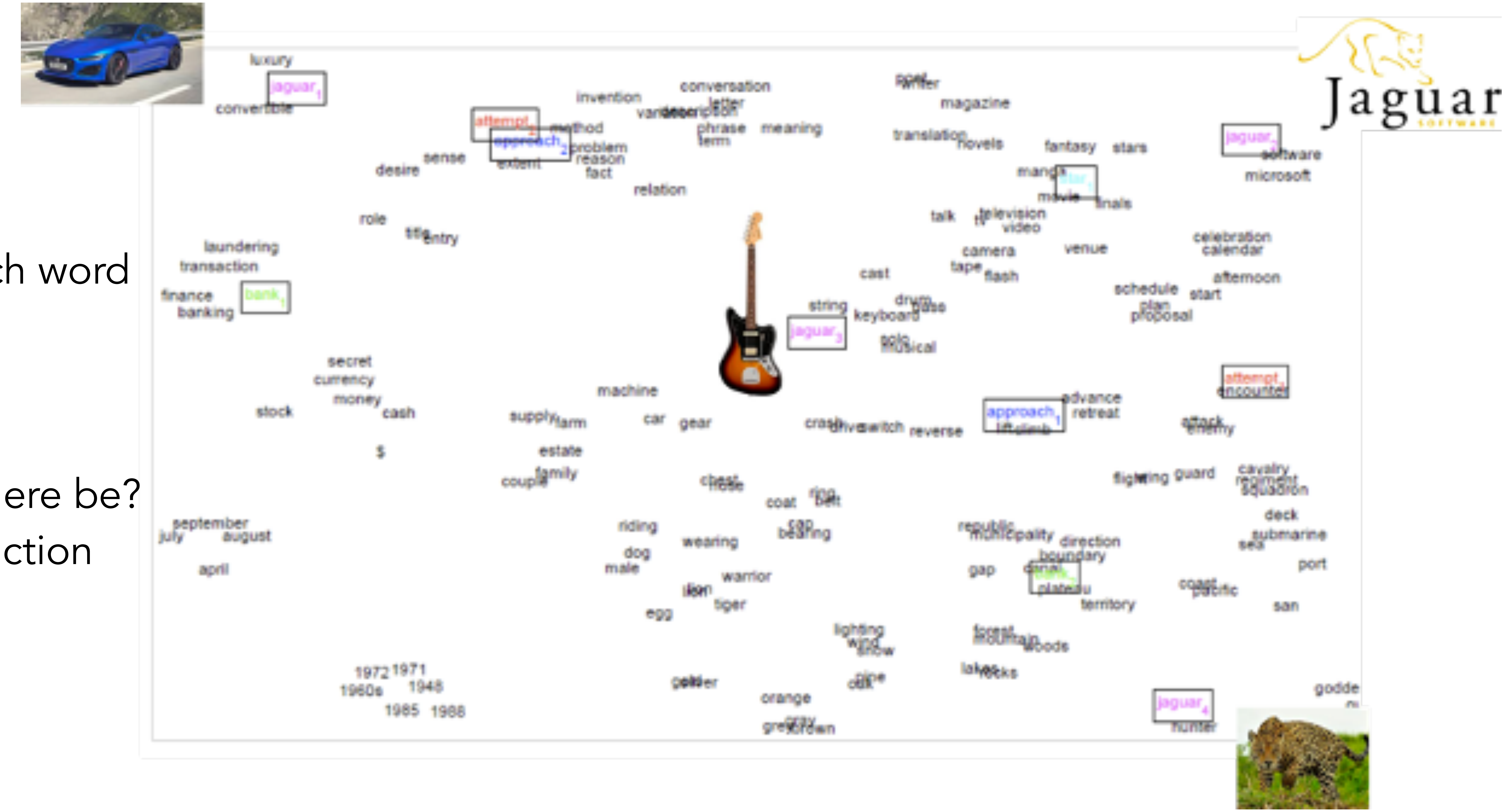
$$v(\text{bank}) = \begin{pmatrix} -0.224 \\ 0.130 \\ -0.290 \\ 0.276 \end{pmatrix}$$

- Complex characteristics of word use: semantics, syntactic behavior, and connotations

- Polysemous words, e.g., bank, mouse

mouse^1 : .... a *mouse* controlling a computer system in 1968.
mouse^2 : .... a quiet animal like a *mouse*
bank^1 : ...a *bank* can hold the investments in a custodial account ...
bank^2 : ...as agriculture burgeons on the east *bank*, the river ...

Sense embeddings



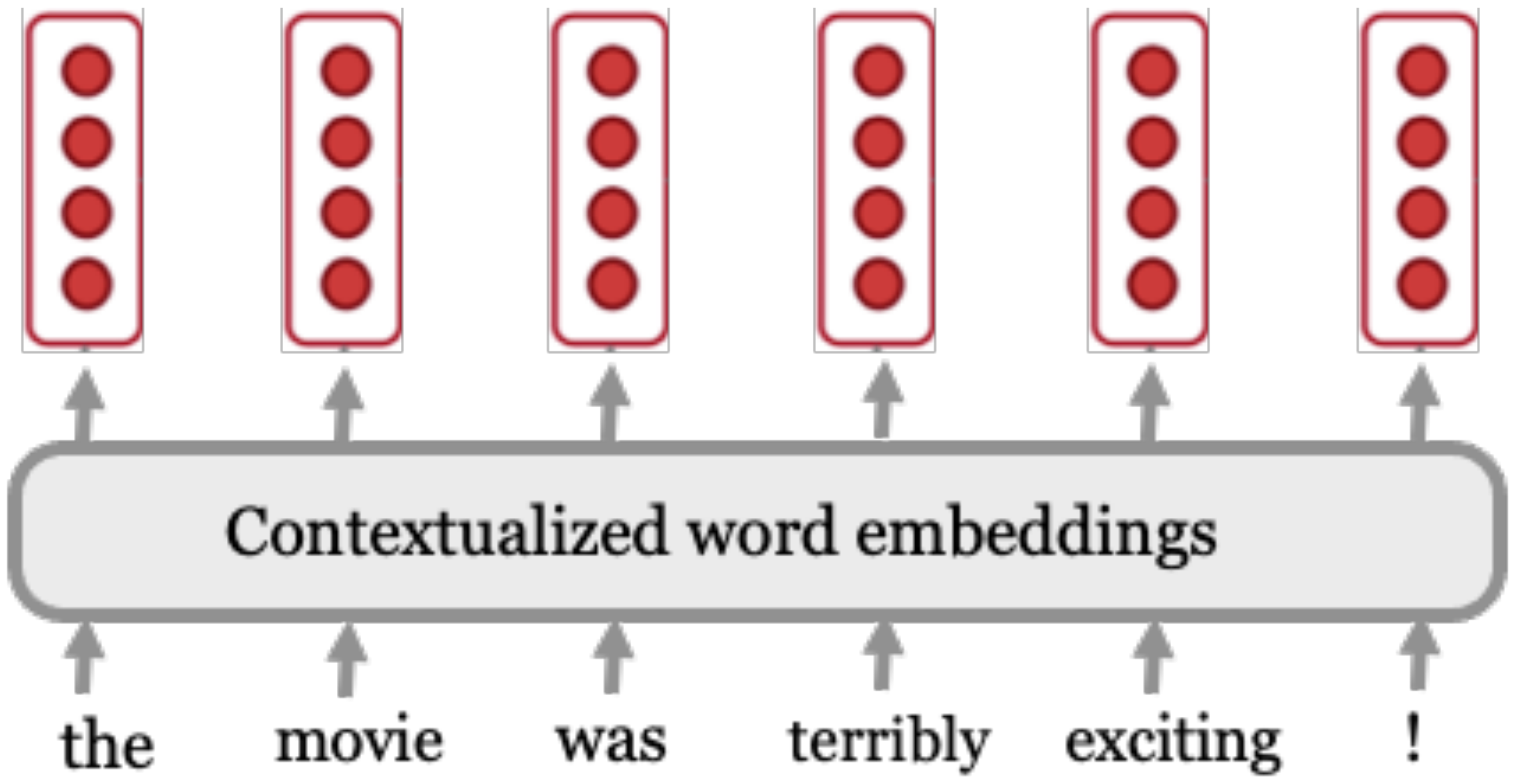
- Multiple embeddings for each word
- One embedding per sense

But

- How many senses should there be?
- Is there always a clear distinction between senses?

Contextualized word embeddings

Let's build a vector for each word conditioned on its context!



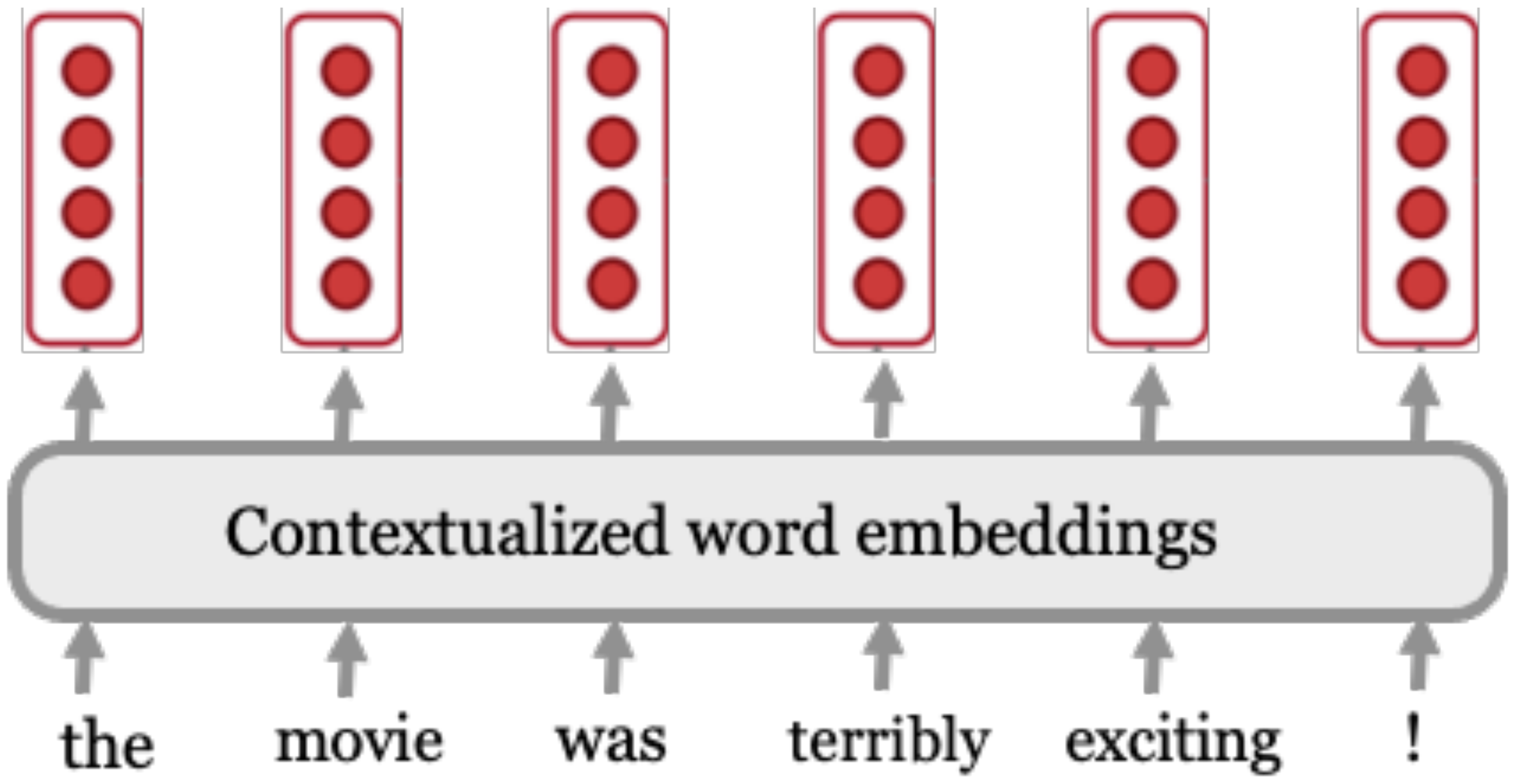
$f: (w_1, w_2, \dots, w_n) \longrightarrow \mathbf{x}_1, \dots, \mathbf{x}_n \in \mathbb{R}^d$

Note: this is different from sentence embeddings where we get one embedding for the entire sentence.

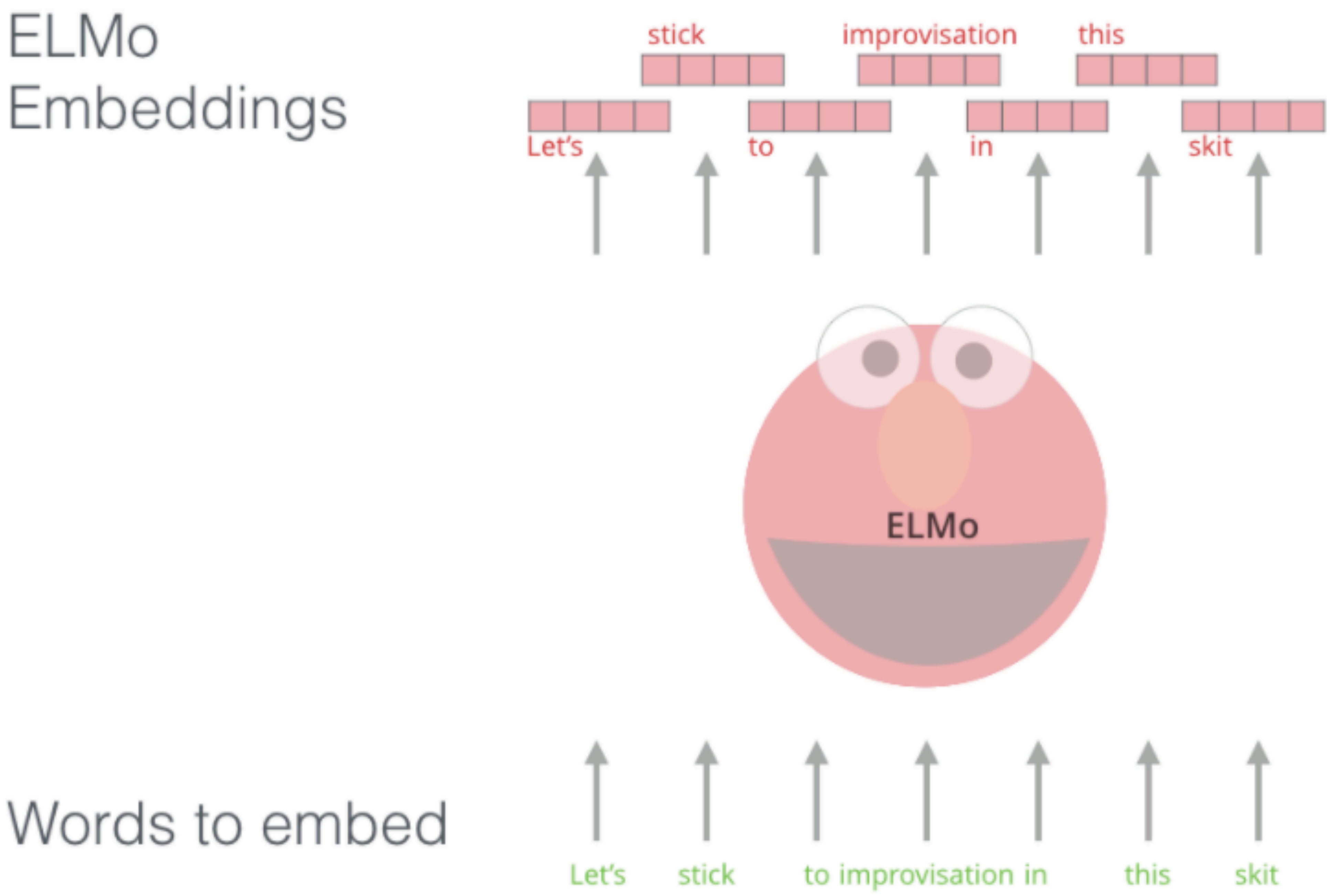
$g: (w_1, w_2, \dots, w_n) \longrightarrow \boldsymbol{s} \in \mathbb{R}^d$

Contextualized word embeddings

Let's build a vector for each word conditioned on its context!



$$f: (w_1, w_2, \ldots, w_n) \longrightarrow \mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^d$$



Contextualized word embeddings

Example sentences with the word play:

1. Chico Ruiz made a spectacular play on Alusik's grounder {. . . }
2. Olivia De Havilland signed to do a Broadway play for Garson {. . . }
3. Kieffer was commended for his ability to hit in the clutch , as well as his all-round excellent play {. . . }
4. {. . . } they were actors who had been handed fat roles in a successful play {. . . }
5. Concepts play an important role in all aspects of cognition {. . . }

Want *v*(play), the vector corresponding to the word play to be different for each of the sentences, with similar senses having similar vectors.

Which of the sentences (2-5) would should have an embedding most similar to sentence 1?

Contextualized word embeddings

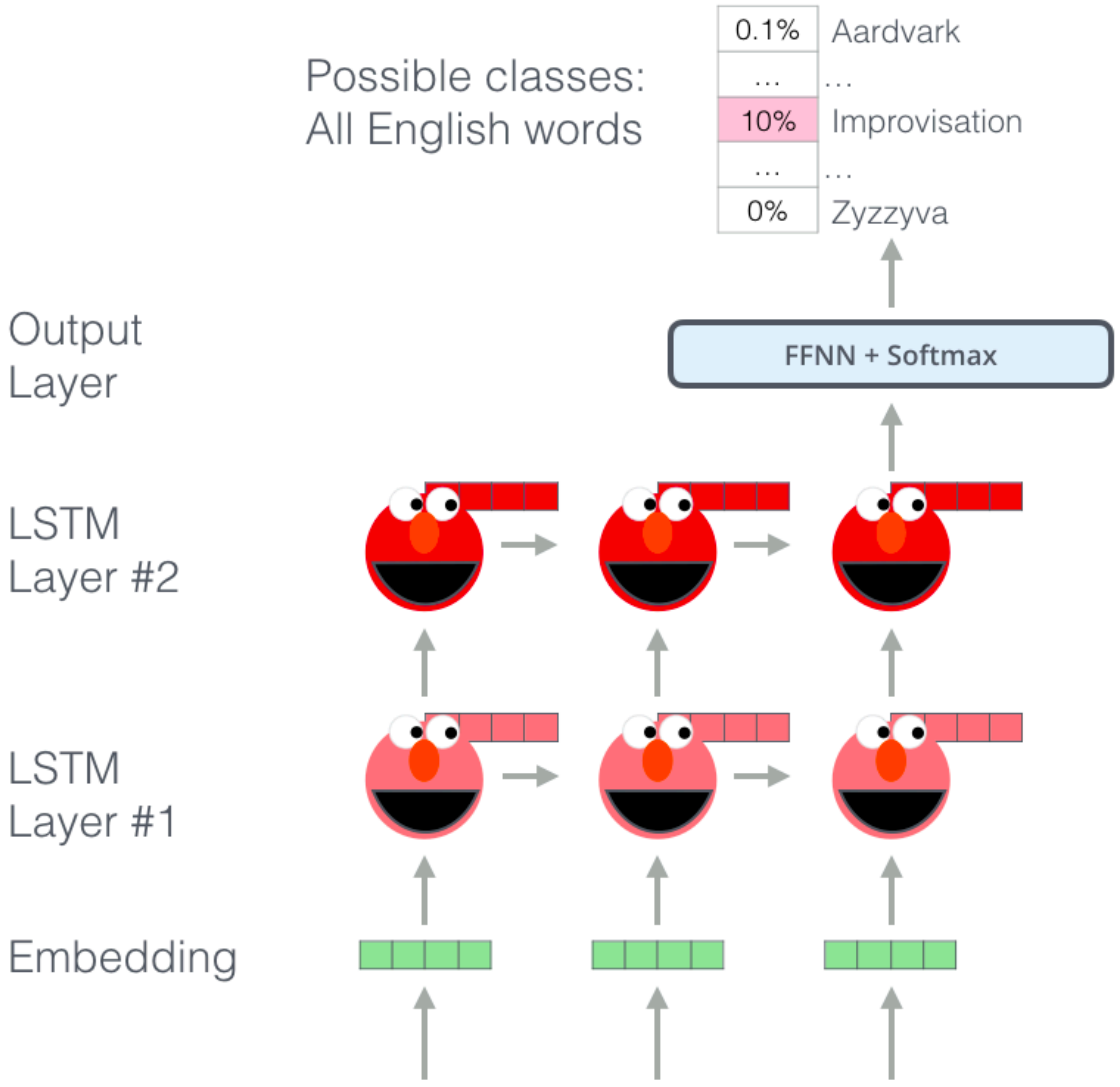
	Source	Nearest Neighbors
GloVe	play	playing, game, games, played, players, plays, player, Play, football, multiplayer
biLM (from ELMo)	Chico Ruiz made a spectacular play on Alusik 's grounder {. . . }	Kieffer , the only junior in the group , was commended for his ability to hit in the clutch , as well as his all-round excellent play .
	Olivia De Havilland signed to do a Broadway play for Garson {. . . }	{. . . } they were actors who had been handed fat roles in a successful play , and had talent enough to fill the roles competently , with nice understatement .

different senses

(Peters et al, 2018): Deep contextualized word representations

ELMo

- NAACL'18: Deep contextualized word representations
- Key idea:
  - Train two stacked LSTM-based language model on some large corpus
  - Use the hidden states of the LSTM for each token to compute a vector representation of each word



ELMo

Pretrain LM

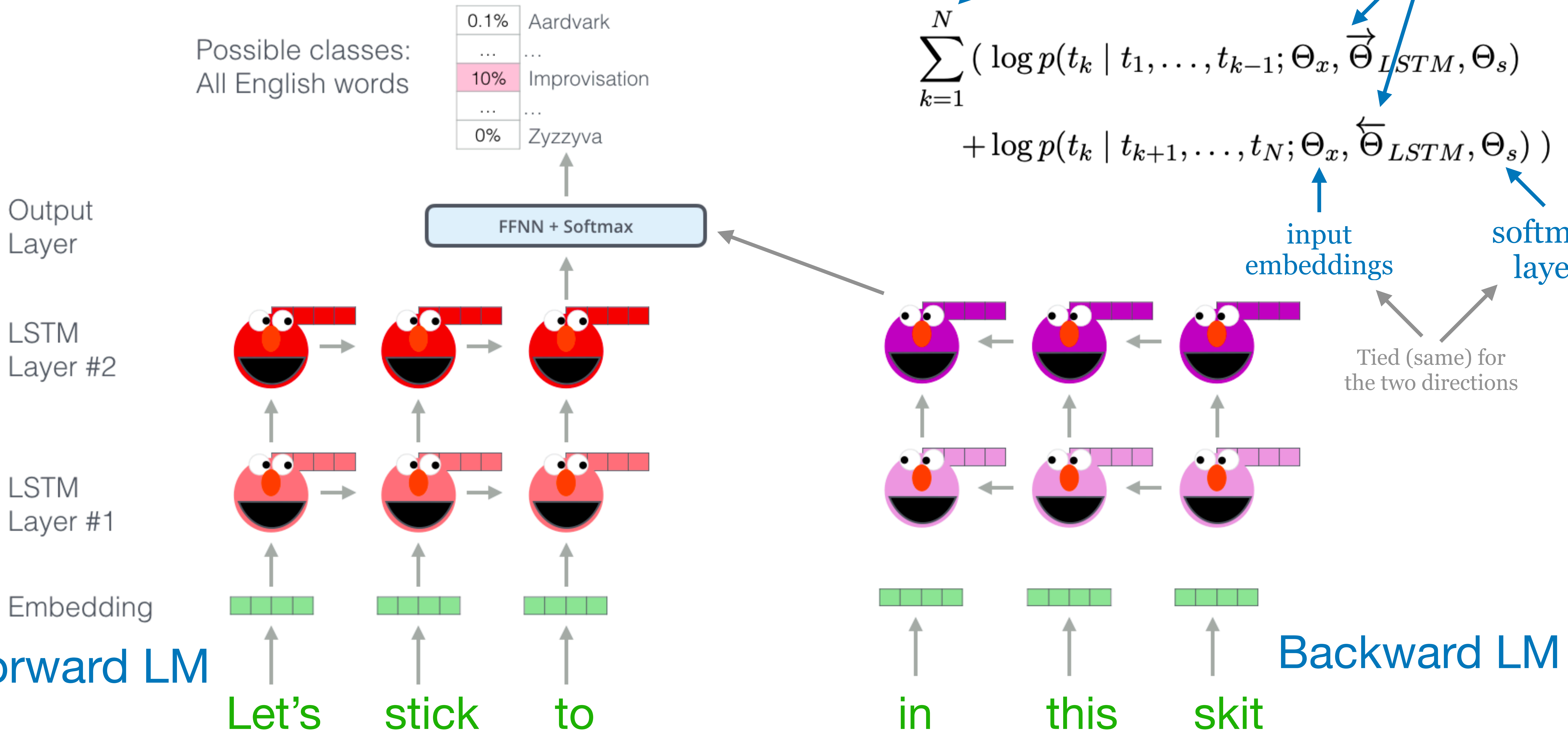
tokens in the sentence

LSTM parameters

$$\sum_{k=1}^{N} \Big(\log p(t_k \mid t_1, \ldots, t_{k-1}; \Theta_x, \overrightarrow{\Theta}_{LSTM}, \Theta_s) + \log p(t_k \mid t_{k+1}, \ldots, t_N; \Theta_x, \overleftarrow{\Theta}_{LSTM}, \Theta_s) \Big)$$

input embeddings

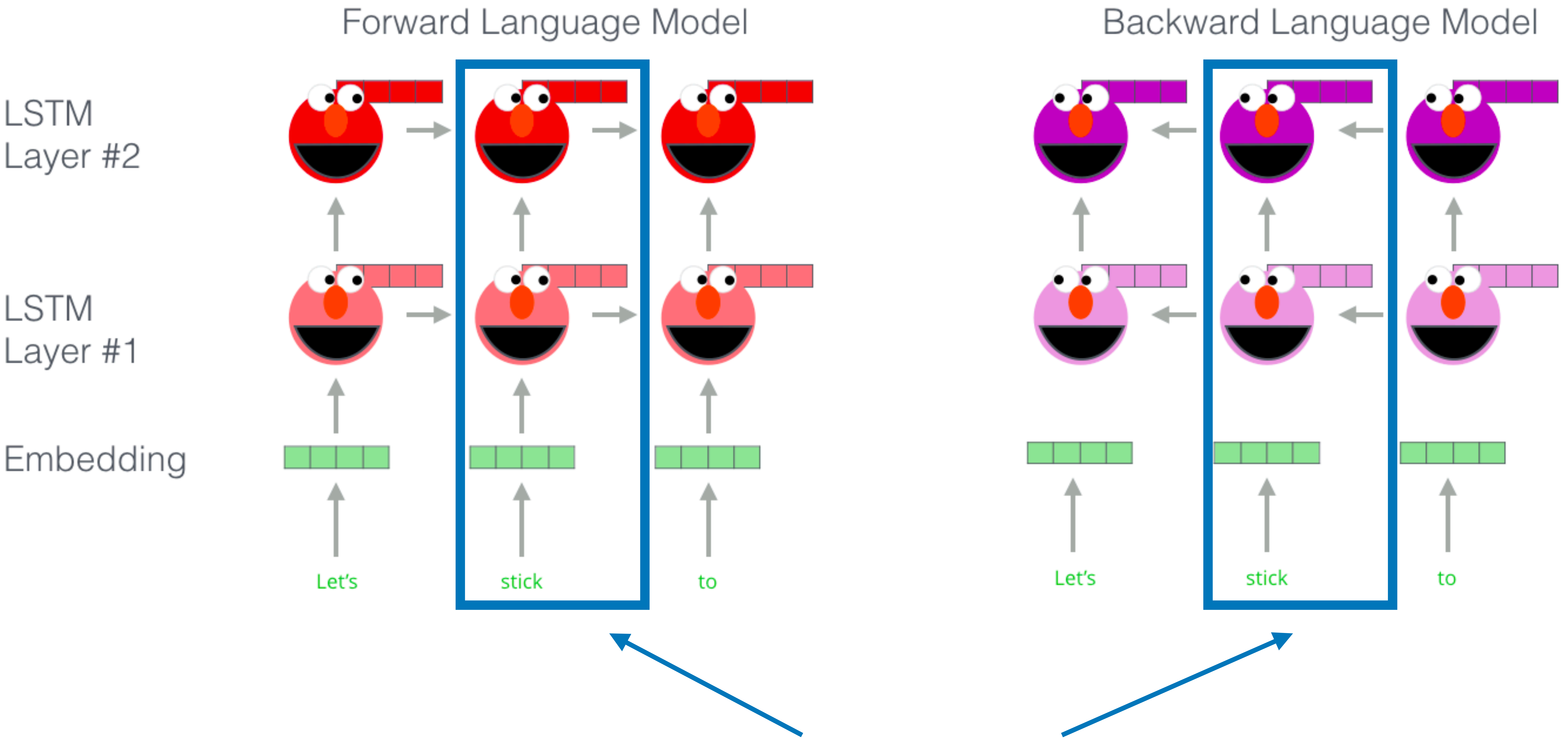
softmax layer



(figure credit: [Jay Alammar](http://jalammar.github.io/illustrated-bert/)
http://jalammar.github.io/illustrated-bert/)

ELMo

After training LM

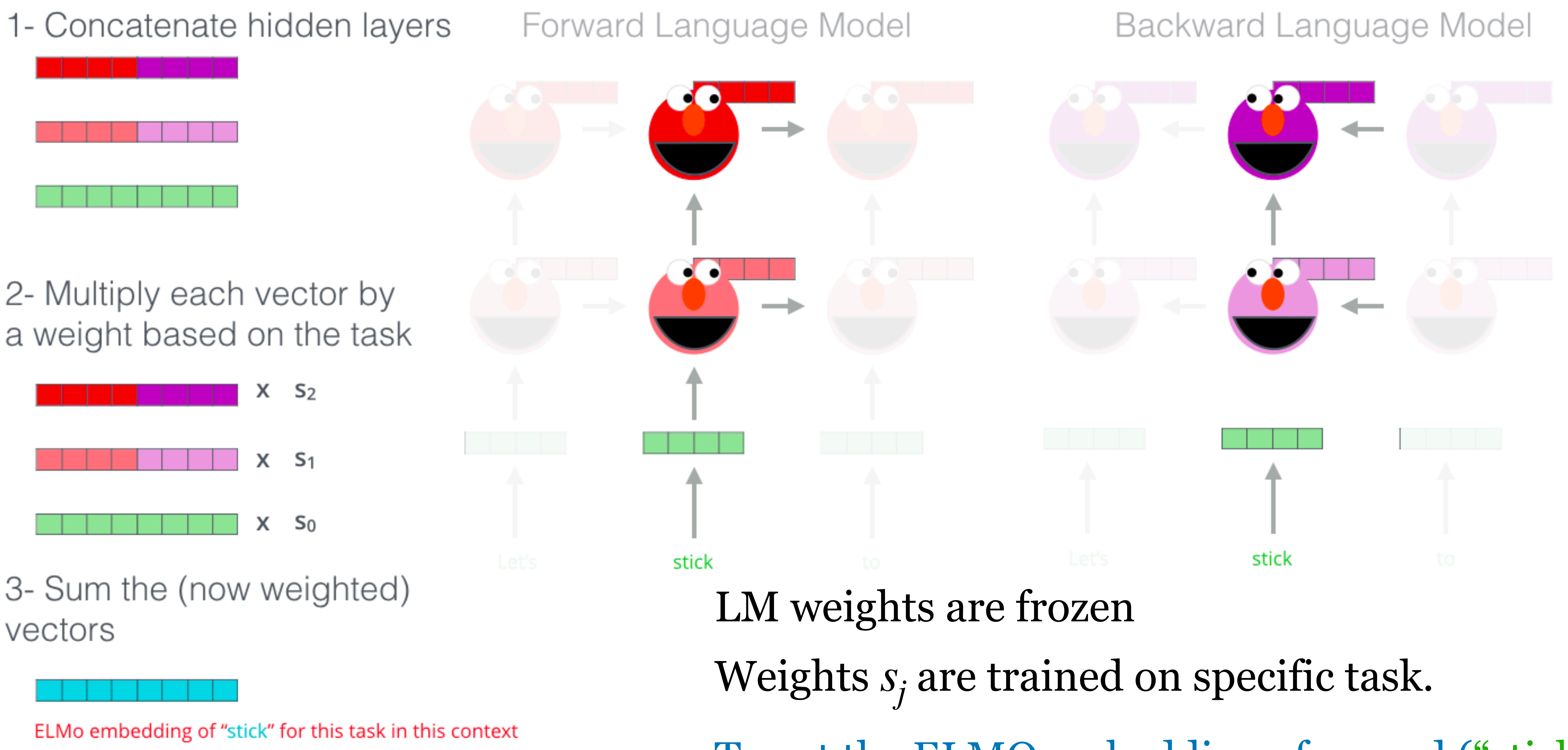


To get the ELMO embedding of a word ("stick"):

Concatenate forward and backward embeddings and take weighted sum of layers

(figure credit: Jay Alammar http://jalammar.github.io/illustrated-bert/)

ELMo



LM weights are frozen

Weights s_j are trained on specific task.

To get the ELMO embedding of a word ("stick"):

Concatenate forward and backward embeddings and take weighted sum of layers

(figure credit: Jay Alammar http://jalammar.github.io/illustrated-bert/)

Summary: How to get ELMo embedding?

Input embeddings　　Hidden state

$$R_k = \{\mathbf{x}_k^{LM}, \overrightarrow{\mathbf{h}}_{k,j}^{LM}, \overleftarrow{\mathbf{h}}_{k,j}^{LM} \mid j = 1, \ldots, L\}$$ ← L is # of layers

$$= \{\mathbf{h}_{k,j}^{LM} \mid j = 0, \ldots, L\},$$

Token representation → $\mathbf{h}_{k,0}^{LM} = \mathbf{x}_k^{LM}$, $\mathbf{h}_{k,j}^{LM} = [\overrightarrow{\mathbf{h}}_{k,j}^{LM}; \overleftarrow{\mathbf{h}}_{k,j}^{LM}]$ ← hidden states

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

Task specific learnable parameters →

- γ^{task}: allows the task model to scale the entire ELMo vector
- s_j^{task}: softmax-normalized weights across layers
- To use: plug ELMo into any (neural) NLP model: freeze all the LMs weights and change the input representation to:

$$[\mathbf{x}_k; \mathbf{ELMo}_k^{task}]$$

(could also insert into higher layers)

More details

- Forward and backward LMs: 2 layers each
- Use character CNN to build initial word representation
    - 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
- A residual connection from the first to second layer
- Trained 10 epochs on 1B Word Benchmark

ELMo: pre-training and use

Data: 10 epoches on 1B Word Benchmark (trained on single sentences)

Pre-training time: 2 weeks on 3 NVIDIA GTX 1080 GPUs

• Much lower time cost if we used V100s / Google's TPUs but still hundreds of dollars in compute cost to train once

• Larger BERT models trained on more data costs $10k+

How to apply ELMo in practice?

• Take the embeddings and feed them into any neural models just like word2vec

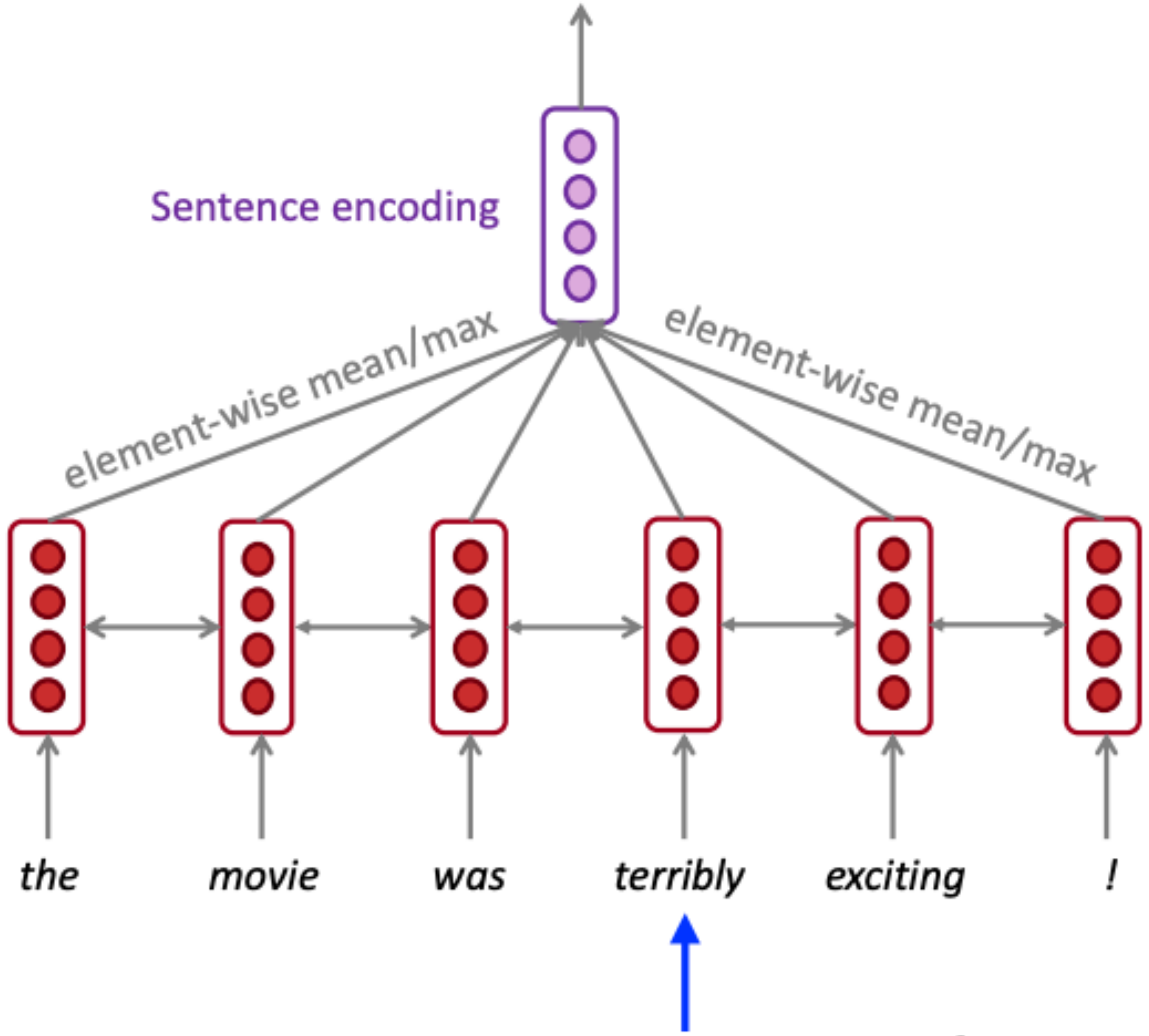
$f : (w_1, w_2, \ldots, w_n) \rightarrow \mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^d$

• The LM's hidden states are fixed and not updated during the downstream use (only the scaling and softmax weights are learned)

• Common practice: concatenate word2vec/GloVe with ELMo

ELMo: pre-training and use

Example: A BiLSTM model for sentiment classification

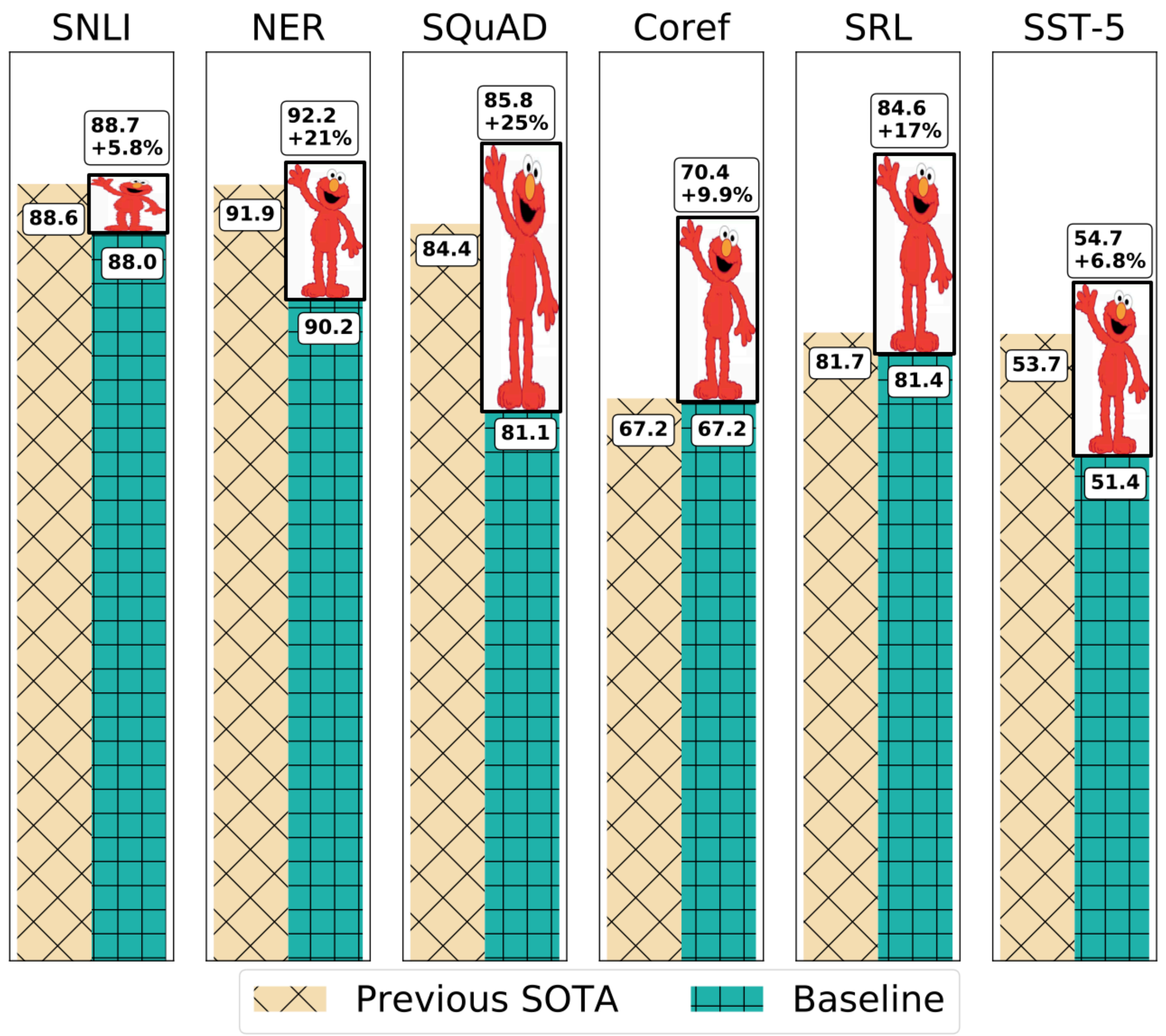


$[e(\text{terribly}); \text{ELMo}_4^{\text{task}}]$

Experimental results

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%

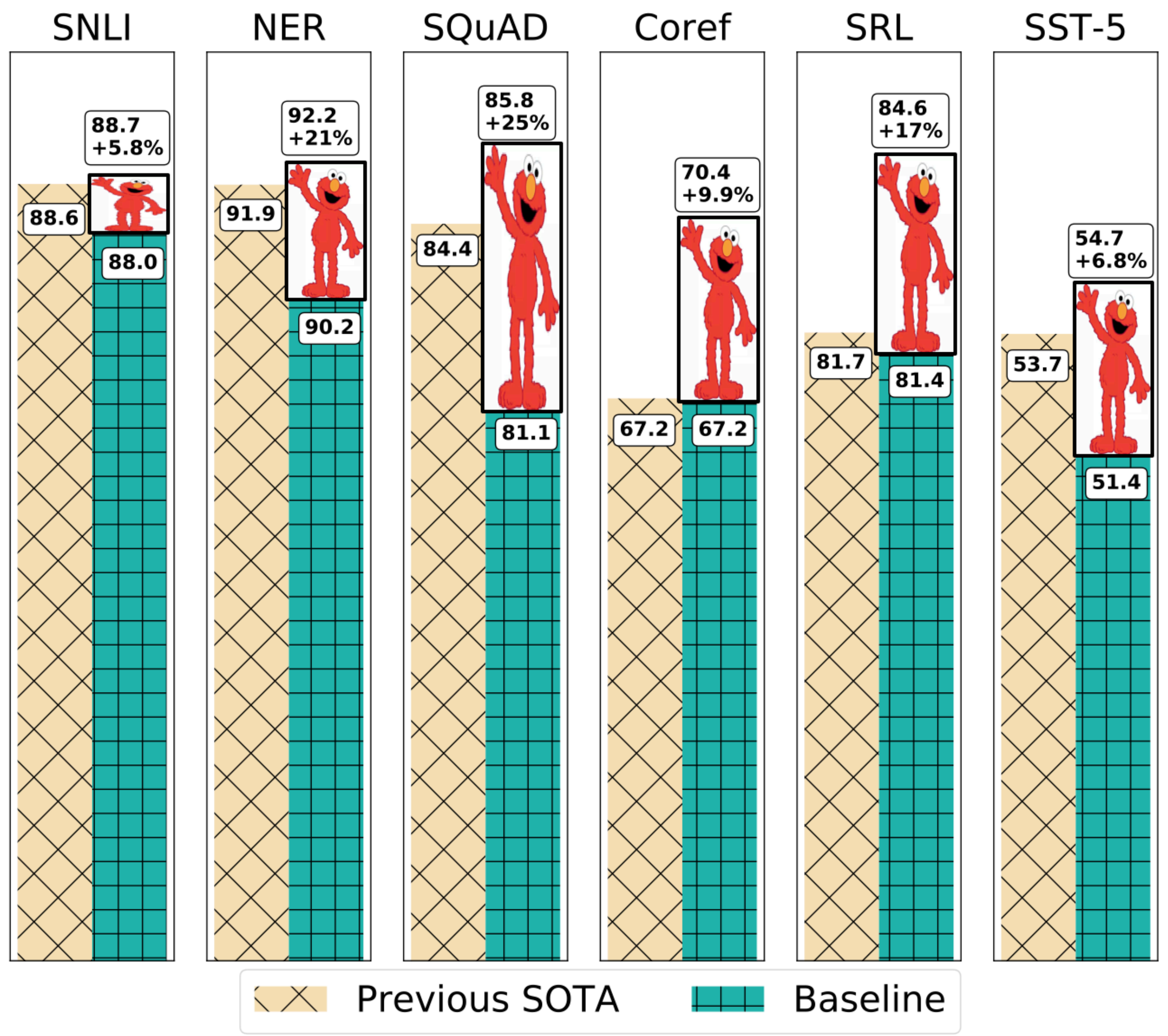
- SQuAD: question answering
- SNLI: natural language inference
- SRL: semantic role labeling
- Coref: coreference resolution
- NER: named entity recognition
- SST-5: sentiment analysis



Experimental results

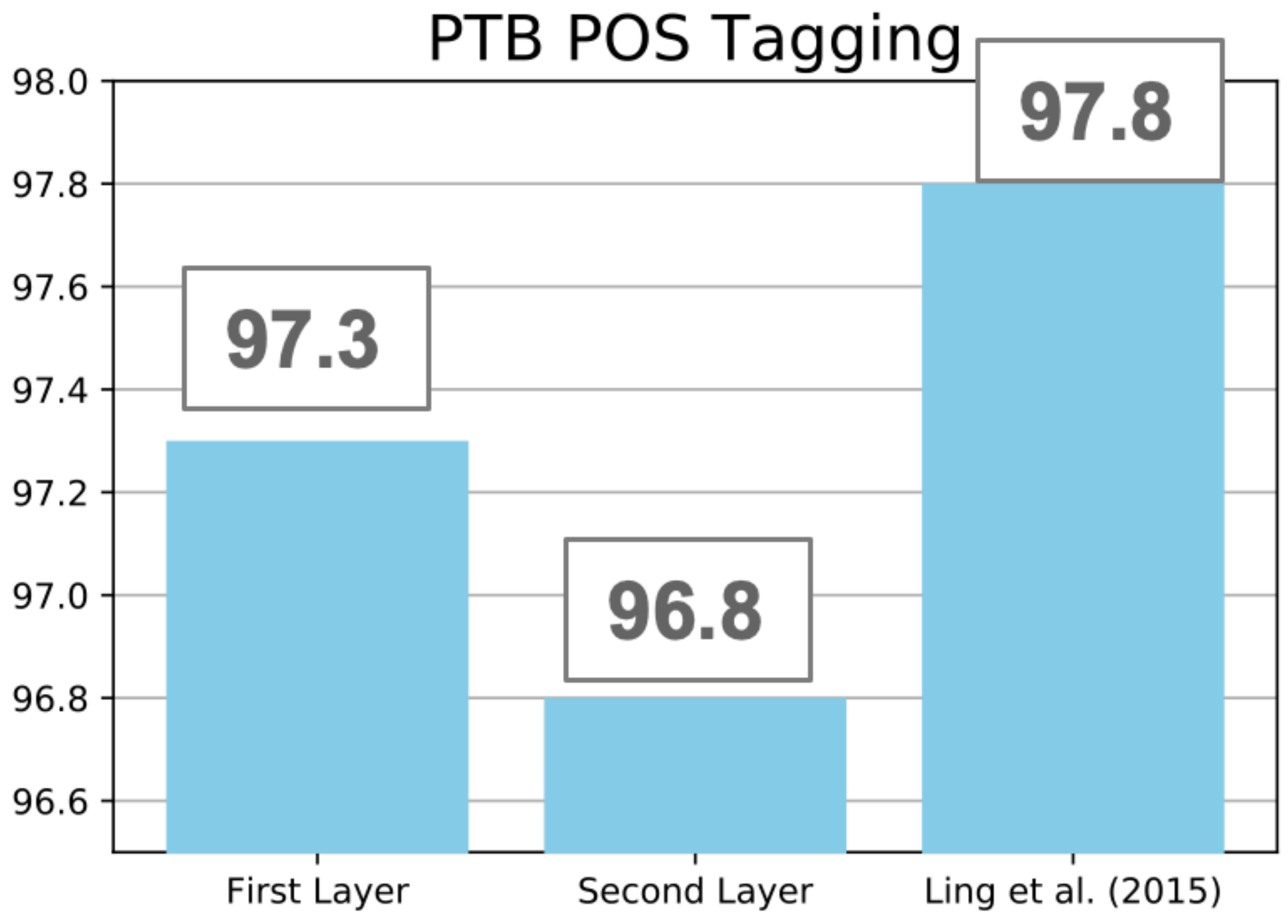
Task	Previous SOTA		Our baseline	ELMo + baseline	Increase (absolute/ relative)
SQuAD	Liu et al. (2017)	84.4	81.1	85.8	4.7 / 24.9%
SNLI	Chen et al. (2017)	88.6	88.0	88.7 ± 0.17	0.7 / 5.8%
SRL	He et al. (2017)	81.7	81.4	84.6	3.2 / 17.2%
Coref	Lee et al. (2017)	67.2	67.2	70.4	3.2 / 9.8%
NER	Peters et al. (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%

- SQuAD: question answering
- SNLI: natural language inference
- SRL: semantic role labeling
- Coref: coreference resolution
- NER: named entity recognition
- SST-5: sentiment analysis



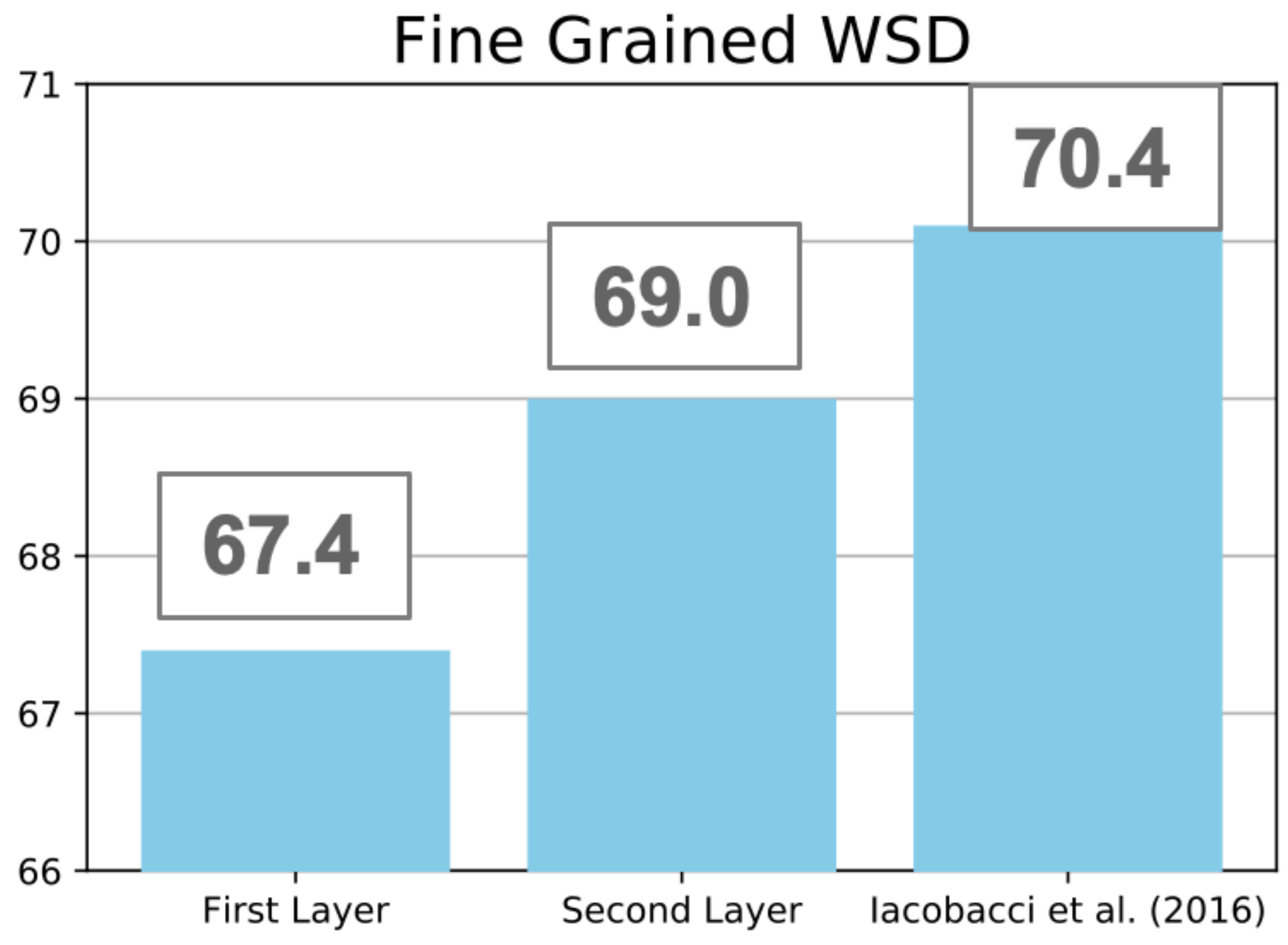
Intrinsic Evaluation

syntactic information



First Layer > Second Layer

semantic information



Second Layer > First Layer

syntactic information is better represented at lower layers while semantic information is captured at higher layers

Use ELMo in practice

https://allennlp.org/elmo

Pre-trained ELMo Models

Model	Link(Weights/Options File)		# Parameters (Millions)	LSTM Hidden Size/Output size	# Highway Layers>
Small	weights	options	13.6	1024/128	1
Medium	weights	options	28.0	2048/256	1
Original	weights	options	93.6	4096/512	2
Original (5.5B)	weights	options	93.6	4096/512	2

```
from allennlp.modules.elmo import Elmo, batch_to_ids

options_file = "https://allennlp.s3.amazonaws.com/models/elmo/2x409
weight_file = "https://allennlp.s3.amazonaws.com/models/elmo/2x4096

# Compute two different representation for each token.
# Each representation is a linear weighted combination for the
# 3 layers in ELMo (i.e., charcnn, the outputs of the two BiLSTM))
elmo = Elmo(options_file, weight_file, 2, dropout=0)

# use batch_to_ids to convert sentences to character ids
sentences = [['First', 'sentence', '.'], ['Another', '.']]
character_ids = batch_to_ids(sentences)

embeddings = elmo(character_ids)
```

Also available in TensorFlow

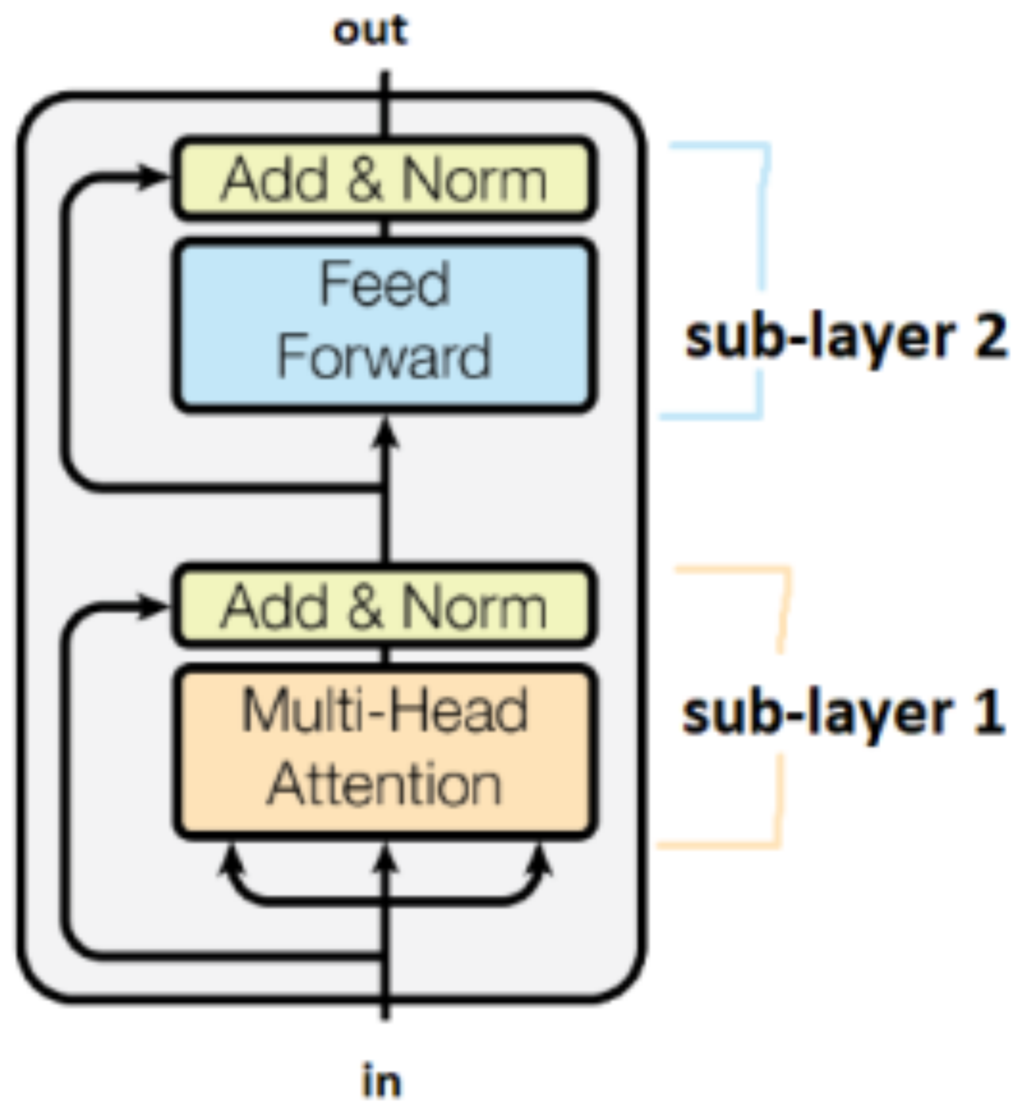
BERT

- First released in Oct 2018.
- NAACL'19: BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

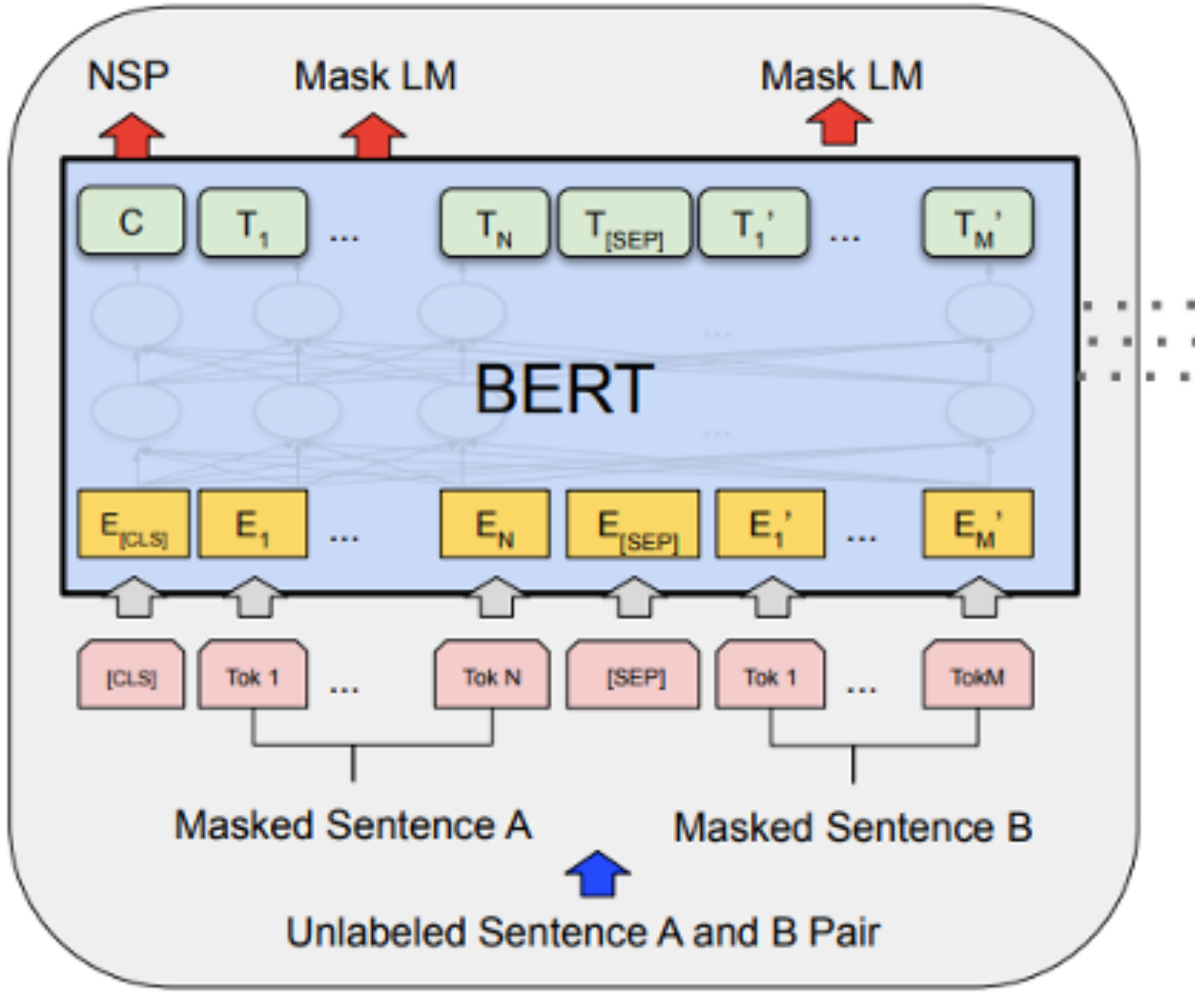
How is BERT different from ELMo?

- Use Transformers instead of LSTMs
- Trained on segments of text (512 word-piece tokens)
- Use a bidirectional encoder instead of two independent LSTMs from both directions
- The weights are not frozen (use fine-tuning for downstream tasks)
- Two new pre-training objectives

BERT



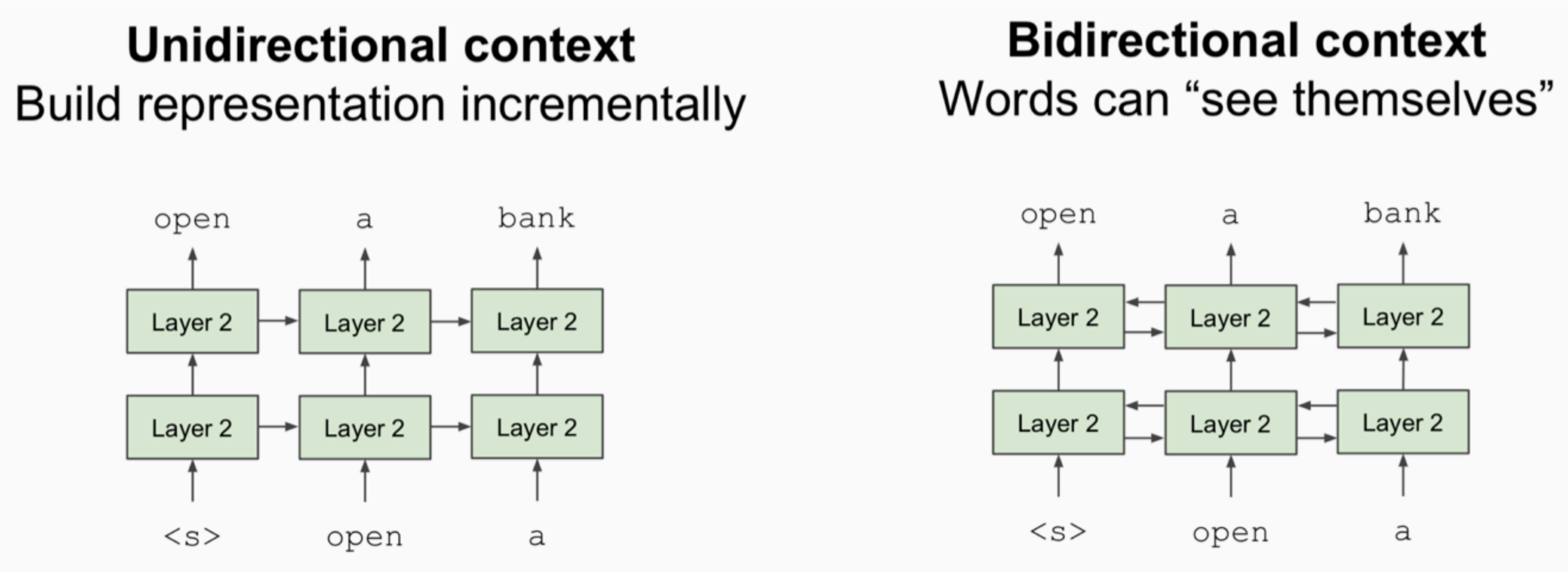
- Transformer Encoder
- Two training objectives
  - Masked Language Modeling
  - Next Sentence Prediction



Pre-training

Bidirectional encoders

- Language models only use left context or right context (although ELMo used two independent LMs from each direction).
- Language understanding is bidirectional



Masked language models (MLMs)

- Solution: Mask out 15% of the input words, and then predict the masked words

- Too little masking: too expensive to train
- Too much masking: not enough context

Masked language models (MLMs)

A little more complex
(don't always replace with `[MASK]`):

Example: `my dog is hairy`, we replace the word hairy

- 80% of time: replace word with `[MASK]` token

  `my dog is [MASK]`

- 10% of time: replace word with random word

  `my dog is apple`

- 10% of time: keep word unchanged to bias representation toward actual observed word

  `my dog is hairy`

Because `[MASK]` is never seen when BERT is used...

Next sentence prediction (NSP)

Always sample two sentences, predict whether the second sentence is followed after the first one.

Input = [CLS] the man went to [MASK] store [SEP]

he bought a gallon [MASK] milk [SEP]

Label = IsNext

Input = [CLS] the man [MASK] to the store [SEP]

penguin [MASK] are flight ##less birds [SEP]

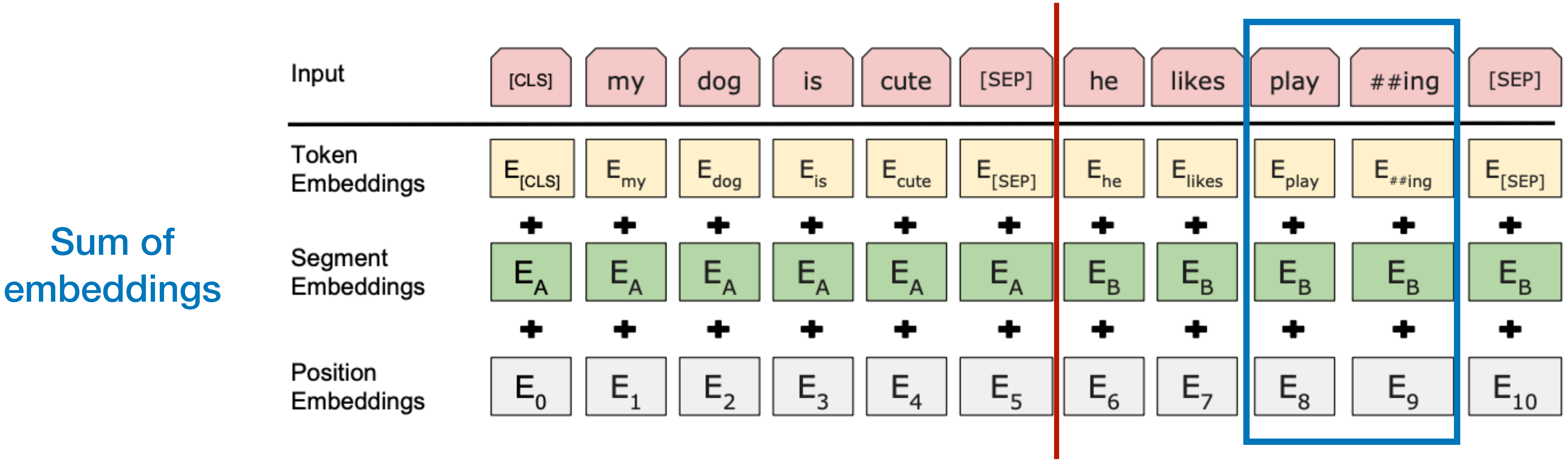
Label = NotNext

Recent papers show that NSP is not necessary…

(Joshi*, Chen* et al, 2019) :SpanBERT: Improving Pre-training by Representing and Predicting Spans
(Liu et al, 2019): RoBERTa: A Robustly Optimized BERT Pretraining Approach

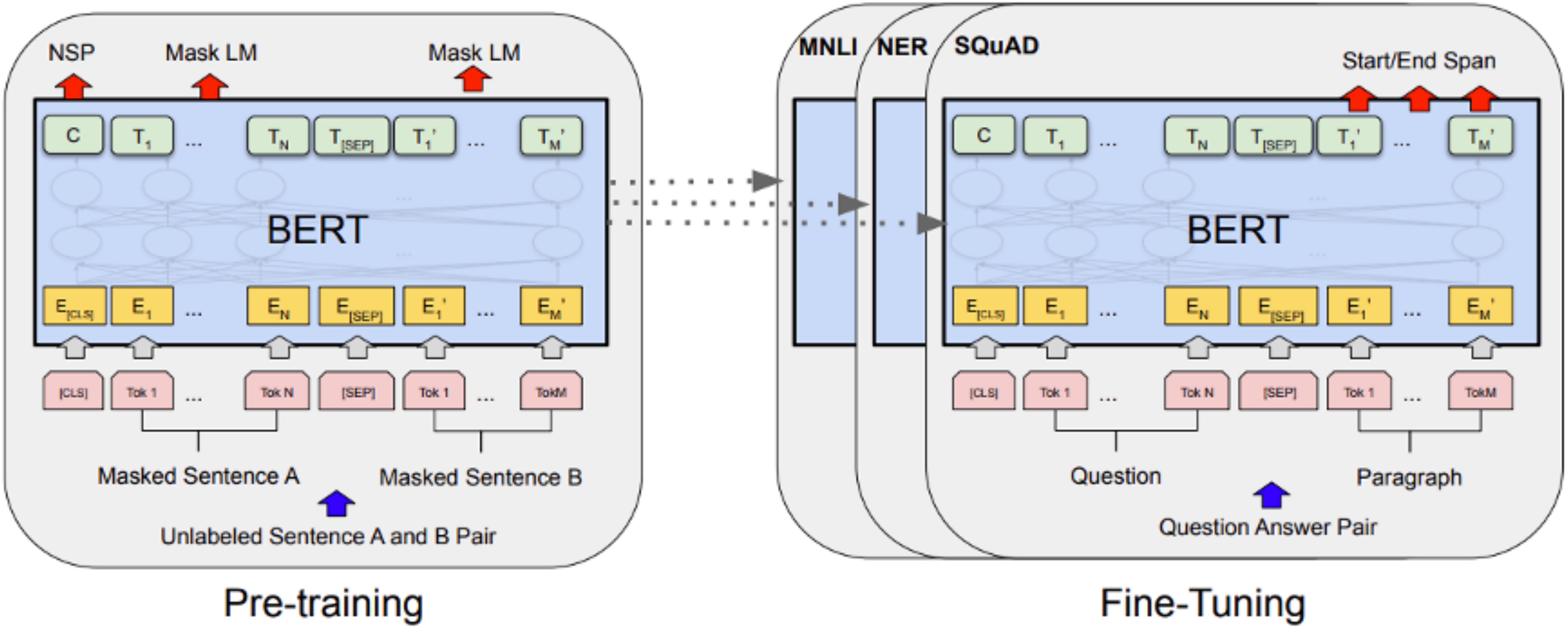
More details

- Input representations



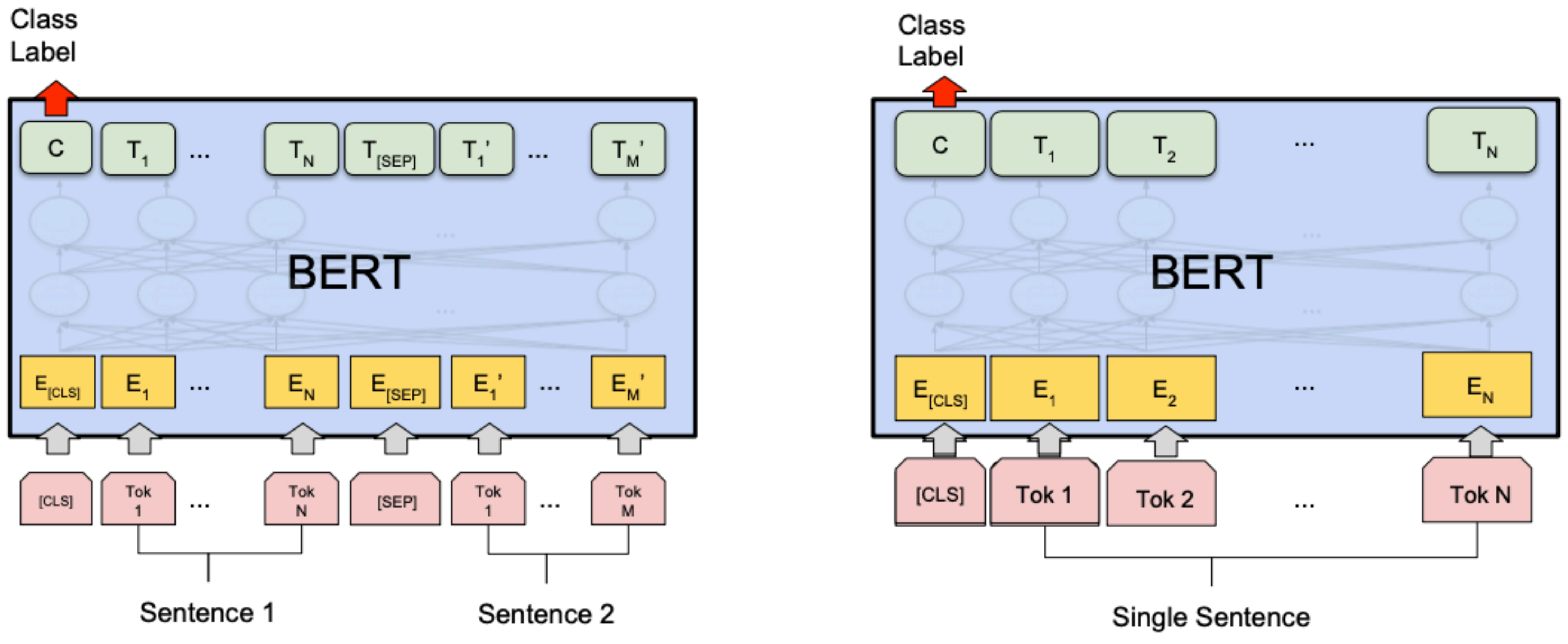
- Use word pieces instead of words: playing => play ##ing (30K token vocabulary)
- Segment length: 512 tokens
- Trained 40 epochs on Wikipedia (2.5B tokens) + BookCorpus (0.8B tokens)
- Released two model sizes: BERT_base, BERT_large

Pre-training and fine-tuning



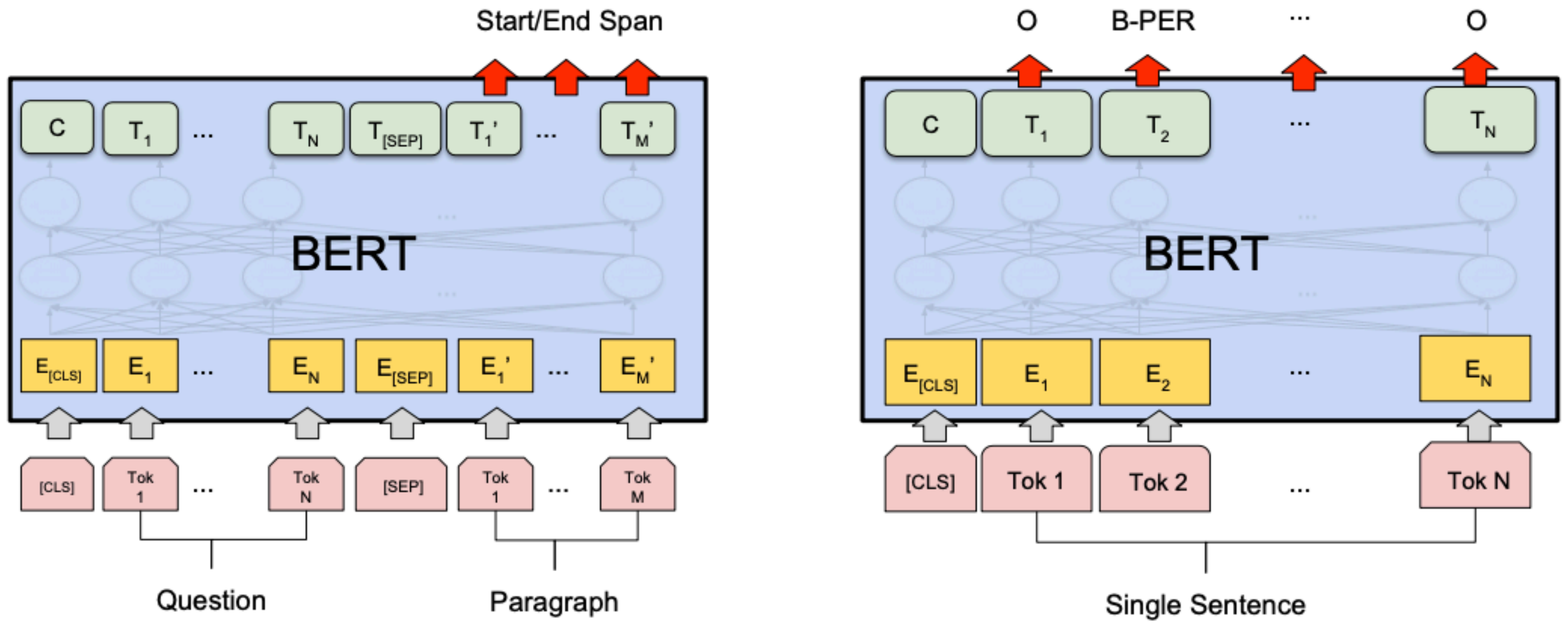
Key idea: **all** the weights are fine-tuned on downstream tasks

Applications



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

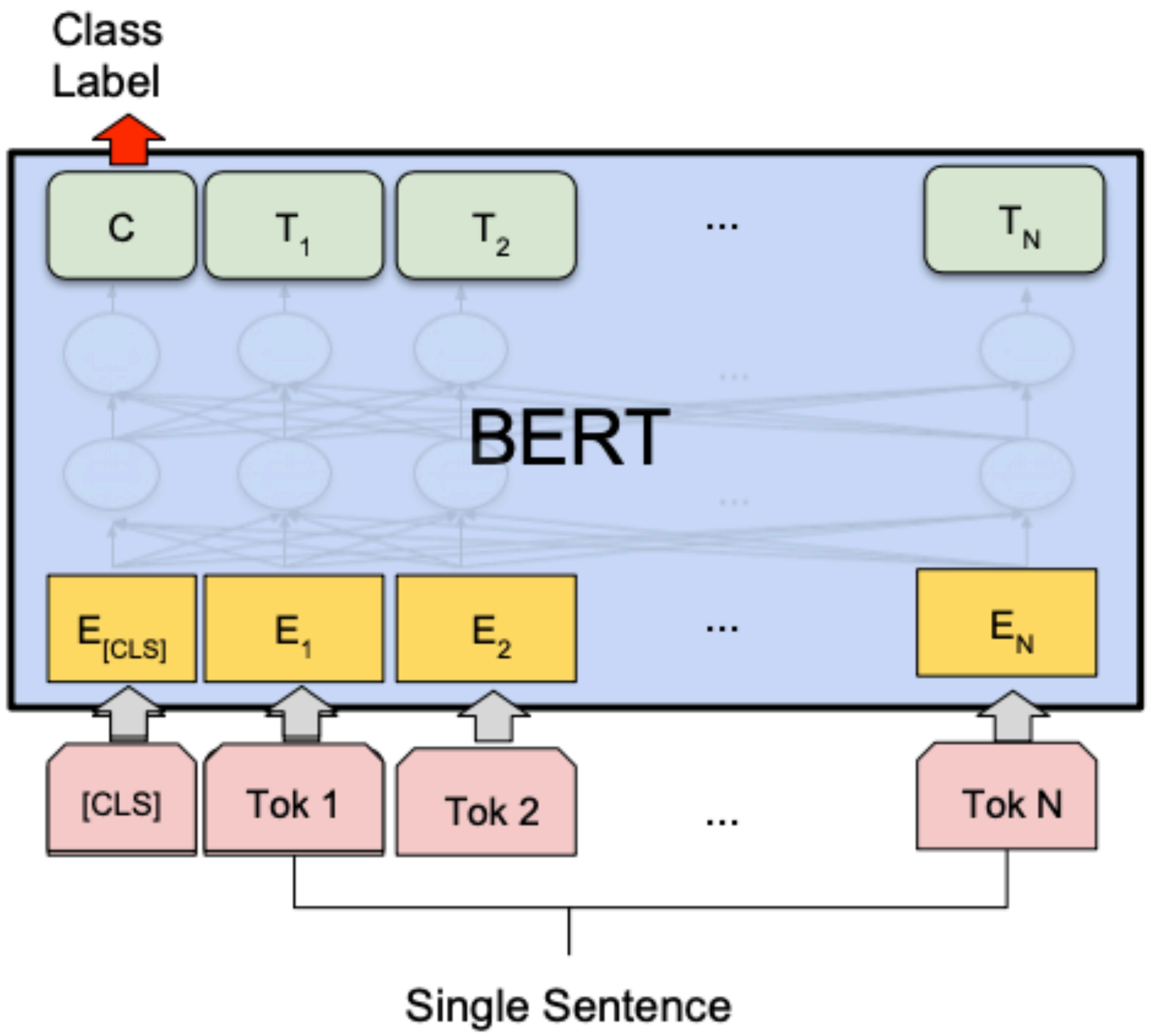
(b) Single Sentence Classification Tasks: SST-2, CoLA



(c) Question Answering Tasks: SQuAD v1.1

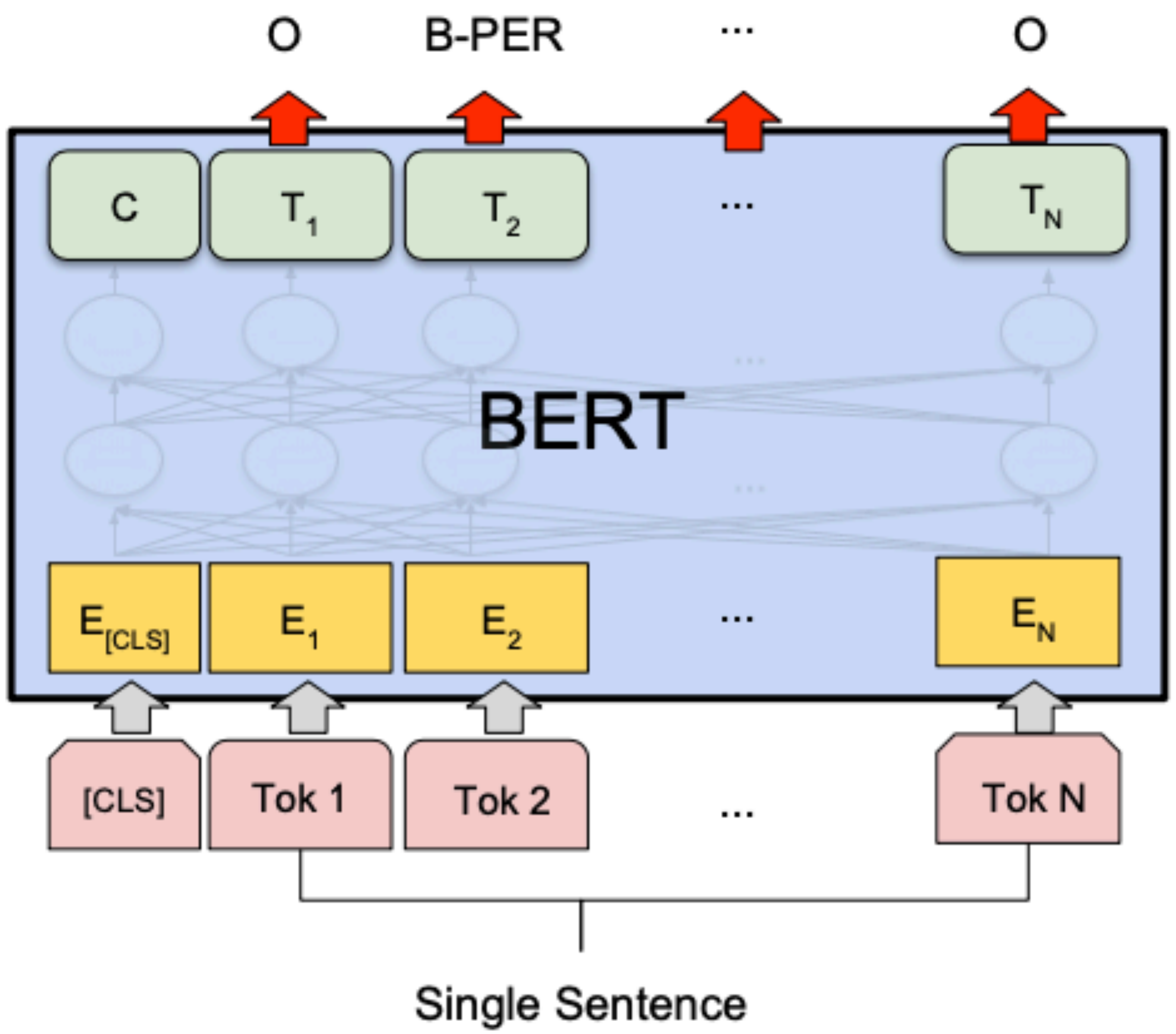
(d) Single Sentence Tagging Tasks: CoNLL-2003 NER

Applications



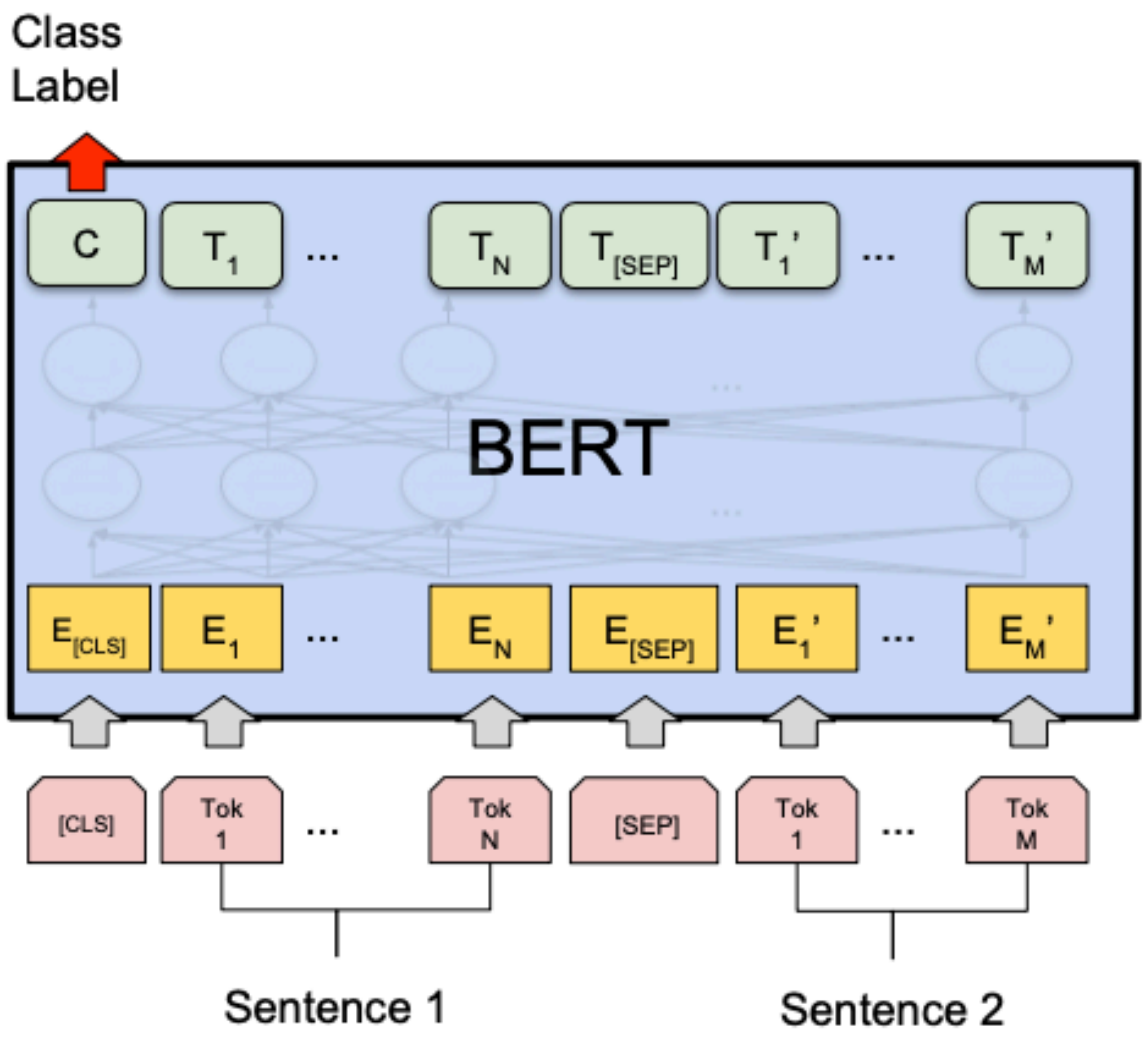
(b) Single Sentence Classification Tasks:
SST-2, CoLA

Applications



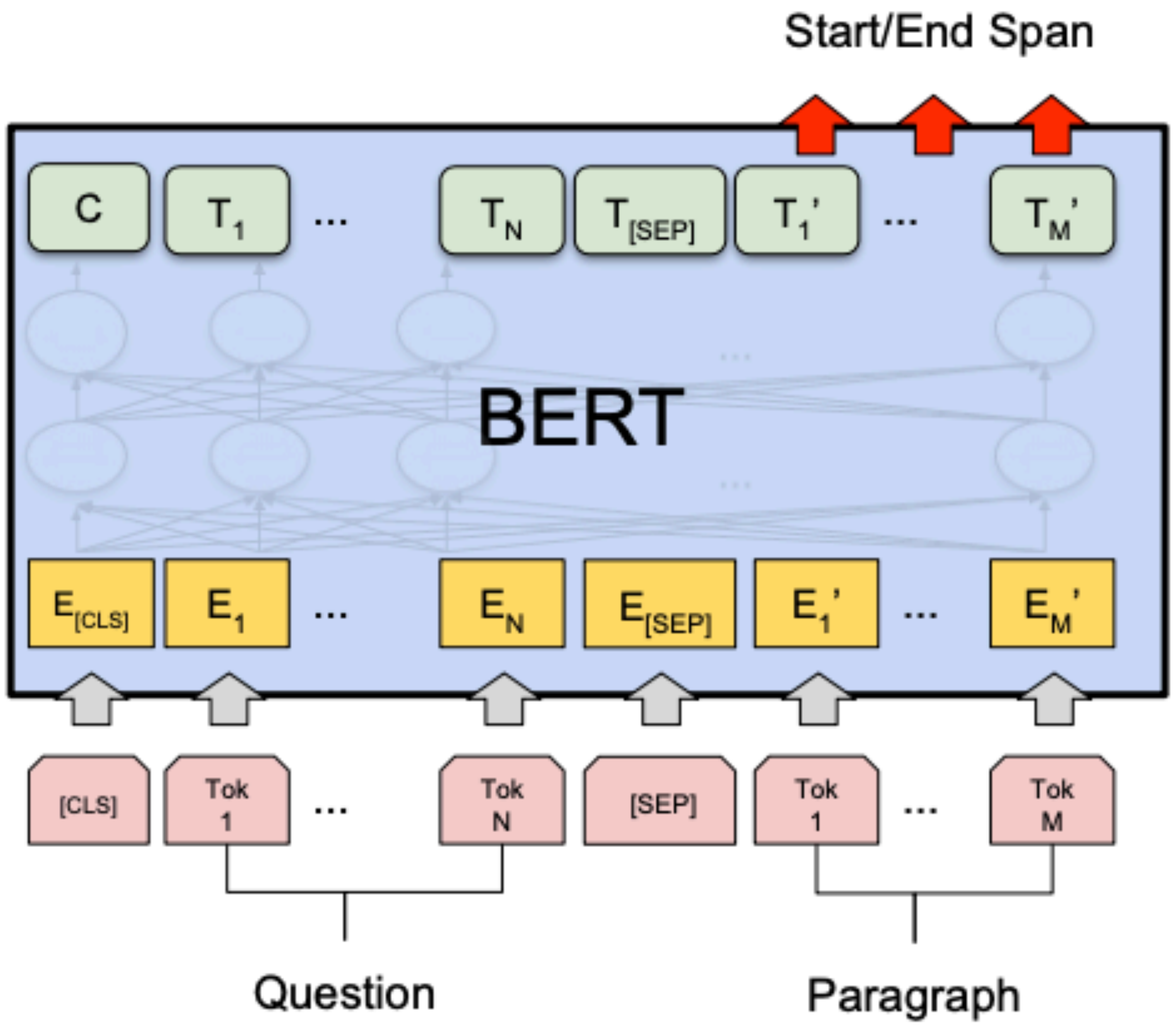
(d) Single Sentence Tagging Tasks:
CoNLL-2003 NER

Applications



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

Applications



(c) Question Answering Tasks:
SQuAD v1.1

BERT Details

Two models were released:

- BERT-base: 12 layers, 768-dim hidden states, 12 attention heads, 110 million params.
- BERT-large: 24 layers, 1024-dim hidden states, 16 attention heads, 340 million params.

Trained on:

- BooksCorpus (800 million words)
- English Wikipedia (2,500 million words)

Pretraining is expensive and impractical on a single GPU.

- BERT was pretrained with 64 TPU chips for a total of 4 days.
- (TPUs are special tensor operation acceleration hardware)

Finetuning is practical and common on a single GPU

- “Pretrain once, finetune many times.”

Experimental results

BiLSTM: 63.9

Entailment

- MNLI: multilingual NLI
- QNLI: NLI with SQuAD data
- RTE: Textual Entailment

Similarity

- QQP: Quora Question Pairs
- STS-B: Semantic Textual Similarity
- MRPC: MS Research Paraphrase Corpus

Other

- SST-2: sentiment analysis
- CoLA: Linguistic acceptability
- SQuAD: question answering

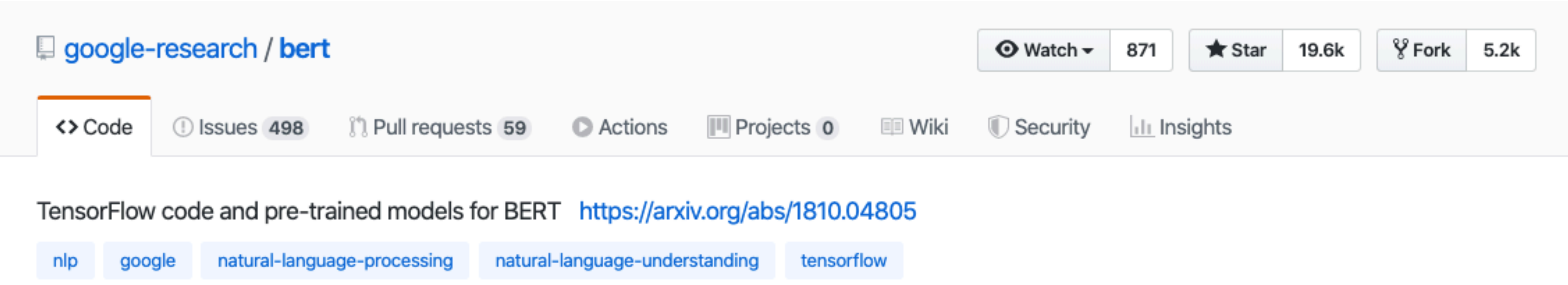
System	MNLI-(m/mm)	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE	**Average**
	392k	363k	108k	67k	8.5k	5.7k	3.5k	2.5k	-
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.9	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	88.1	91.3	45.4	80.0	82.3	56.0	75.2
$\text{BERT}_{\text{BASE}}$	84.6/83.4	71.2	90.1	93.5	52.1	85.8	88.9	66.4	79.6
$\text{BERT}_{\text{LARGE}}$	**86.7/85.9**	**72.1**	**91.1**	**94.9**	**60.5**	**86.5**	**89.3**	**70.1**	**81.9**

Model	**data**	**bsz**	**steps**	**SQuAD** (v1.1/2.0)	**MNLI-m**	**SST-2**
RoBERTa						
with BOOKS + WIKI	16GB	8K	100K	93.6/87.3	89.0	95.3
+ additional data (§3.2)	160GB	8K	100K	94.0/87.7	89.3	95.6
+ pretrain longer	160GB	8K	300K	94.4/88.7	90.0	96.1
+ pretrain even longer	160GB	8K	500K	**94.6/89.4**	**90.2**	**96.4**
$\text{BERT}_{\text{LARGE}}$						
with BOOKS + WIKI	13GB	256	1M	90.9/81.8	86.6	93.7
$\text{XLNet}_{\text{LARGE}}$						
with BOOKS + WIKI	13GB	256	1M	94.0/87.8	88.4	94.4
+ additional data	126GB	2K	500K	94.5/88.8	89.8	95.6

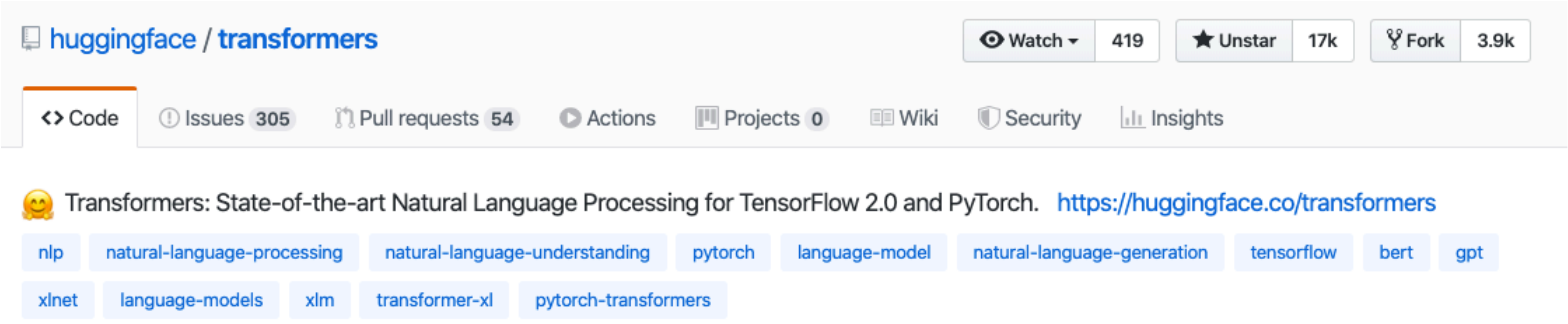
(Wang et al, 2018): GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding

Use BERT in practice

TensorFlow: https://github.com/google-research/bert



PyTorch: https://github.com/huggingface/transformers



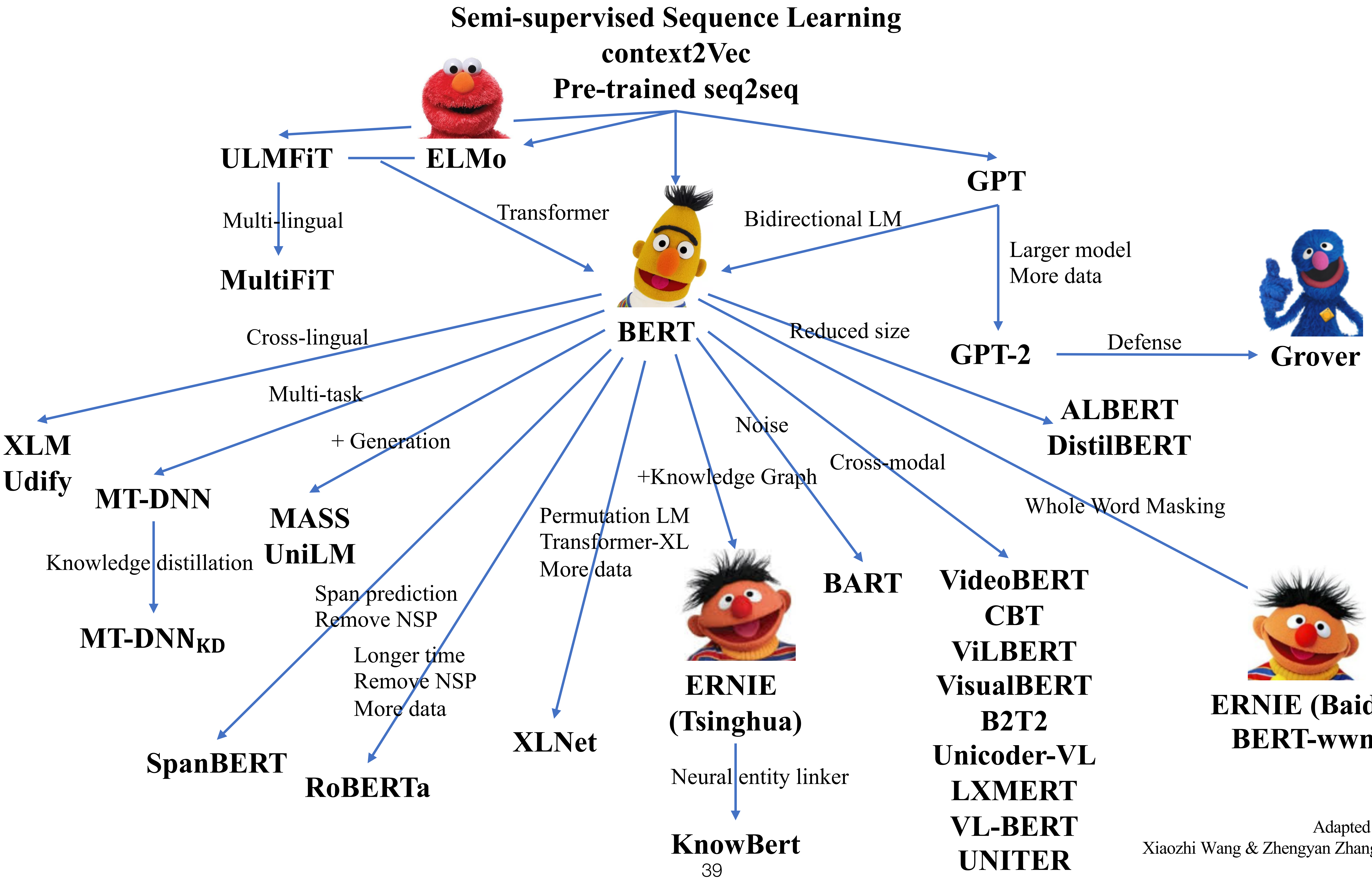
Contextualized word embeddings in context

- TagLM (Peters et, 2017)
- CoVe (McCann et al. 2017)
- ULMfit (Howard and Ruder, 2018)
- **ELMo (Peters et al, 2018)**
- OpenAI GPT (Radford et al, 2018)
- **BERT (Devlin et al, 2018)**
- OpenAI GPT-2 (Radford et al, 2019)
- XLNet (Yang et al, 2019)
- SpanBERT (Joshi et al, 2019)
- RoBERTa (Liu et al, 2019)
- ALBERT (Lan et al, 2019)
- DistilBERT (Sanh et al, 2019)
- ELECTRA (Clark et al, 2020)
- …

Transformers

https://github.com/huggingface/transformers

See https://huggingface.co/transformers/ for more information and models



Adapted from slide by
Xiaozhi Wang & Zhengyan Zhang @THUNLP

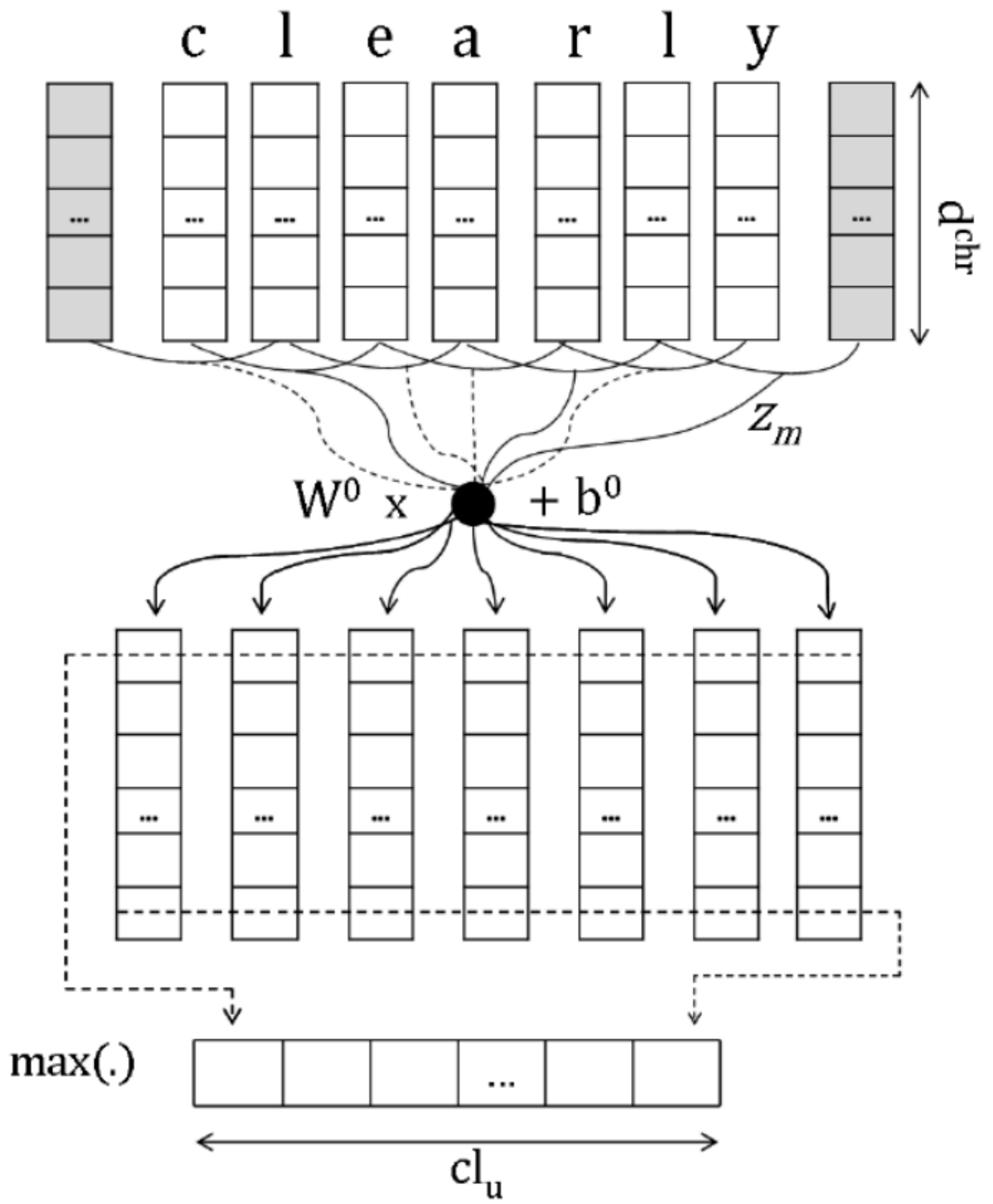
Subword modeling

Subword modeling

- Why subword modeling?
  - Captures morphology
  - Helps with OOV words
    - New words, spelling variants, misspellings, and noisy text
- Ways of incorporating subword modeling
  - Use subwords (word-pieces) as tokens
  - Hybrid architecture where part of the word embeddings come from subword modeling
- Used in most SOTA NLP methods
  - Character CNNs in ELMo
  - BPE (Byte Pair Encoding) in original Transformer paper
  - Wordpiece / sentence piece (in BERT)

NN over characters to build word representations

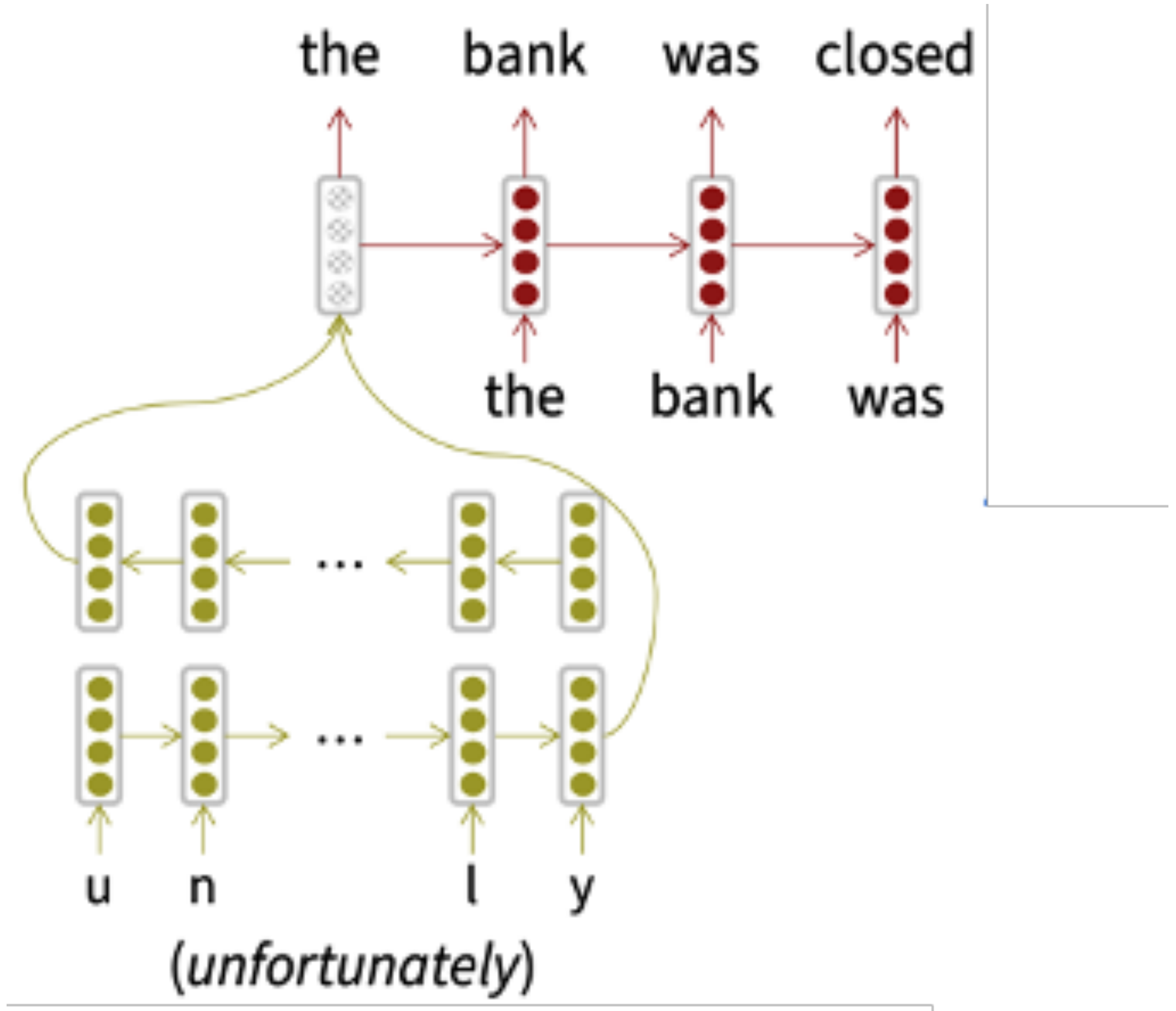
- Convolution over characters to generate word embeddings



Fixed window of word embeddings used for PoS tagging

Learning Character-level Representations for Part-of-Speech Tagging (Dos Santos and Zadrozny 2014)

- Same objective as word2vec but with characters
- Bi-directional LSTM to compute embedding



Char2vec: A joint model for word embedding and word morphology (Cao and Rei, 2016)

Byte Pair Encoding

- Originally a compression algorithm
  - Bottom up clustering
  - Most frequent byte pair -> a new byte
  - For words, replace bytes with character ngrams
- Automatically build vocabulary
  - Vocabulary is pieces of words (or character ngrams)
  - Deterministic algorithm that finds the common longest pieces of words to use in vocabulary

Byte Pair Encoding

count	bigram
5 + 2	l o
5 + 2	o w
2 + 6	w e
2	e r
6	n e
6	e w
6 + 3	e s
6 + 3	s t
3	w i
3	i d
3	d e

- A word (character ngram) segmentation algorithm
  - Start with a vocabulary of characters
  - Take most frequent ngram pair -> add the new ngram the to vocabulary

Dictionary

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

Vocabulary

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

Start with all characters in vocabulary

Byte Pair Encoding

count	bigram
5 + 2	l o
5 + 2	o w
2	w e
2	e r
6	n e
6	e w
6	w es
$\boxed{6 + 3}$	$\boxed{\text{es t}}$
3	w i
3	i d
3	d es

- A word (character ngram) segmentation algorithm
  - Start with a vocabulary of characters
  - Take most frequent ngram pair -> add the new ngram the to vocabulary

Dictionary

5 l o w
2 l o w e r
6 n e w **es** t
3 w i d **es** t

Vocabulary

l, o, w, e, r, n, w, s, t, i, d, **es**

Add a pair **(e,s)** with frequency 9

Byte Pair Encoding

count	bigram
$\boxed{5 + 2}$	$\boxed{l\ o}$
5 + 2	o w
2	w e
2	e r
6	n e
6	e w
6	w est
3	w i
3	i d
3	d est

- A word (character ngram) segmentation algorithm
  - Start with a vocabulary of characters
  - Take most frequent ngram pair -> add the new ngram the to vocabulary

Dictionary

5	l o w
2	l o w e r
6	n e w **est**
3	w i d **est**

Vocabulary

l, o, w, e, r, n, w, s, t, i, d, es, **est**

Add a pair **(es,t)** with frequency 9

Byte Pair Encoding

count	bigram
5 + 2	lo w
2	w e
2	e r
6	n e
6	e w
6	w est
3	w i
3	i d
3	d est

- A word (character ngram) segmentation algorithm
  - Start with a vocabulary of characters
  - Take most frequent ngram pair -> add the new ngram the to vocabulary

Dictionary

5	**lo** w
2	**lo** w e r
6	n e w est
3	w i d est

Vocabulary

l, o, w, e, r, n, w, s, t, i, d, es, est, **lo**

Add a pair **(l,o)** with frequency 7

Byte Pair Encoding

- When to stop
  - Have a target vocabulary size and stop when you reach it
- Deterministic, common longest piece segmentation of words
- Segmentation is only within words already identified by some prior tokenizers
- Automatically decide vocabulary to use (vocabulary is pieces of words - character ngrams)

Wordpiece/Sentencepiece

- Used in Google NMT
  - V1: wordpiece — only tokenizes inside words
  - V2: sentencepiece — works directly on raw text (use special token _ for whitespace)
- Difference way to select what ngram to add
  - Choose n-gram that maximally reduces perplexity
  - Greedy approximation to maximizing the language model log likelihood

Wordpiece/Sentencepiece

- Used in Google NMT
  - V1: wordpiece
  - V2: sentencepiece
- Variant of wordpiece model is used in BERT
  - Common words are in vocabulary: at, Fairfax, 1910s
  - Other words built from workpieces: Hypatia = h ##yp ##ati ##a