COS 484: Natural Language Processing

Contextualized Word Embeddings

Fall 2019

Overview

Contextualized Word Representations

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

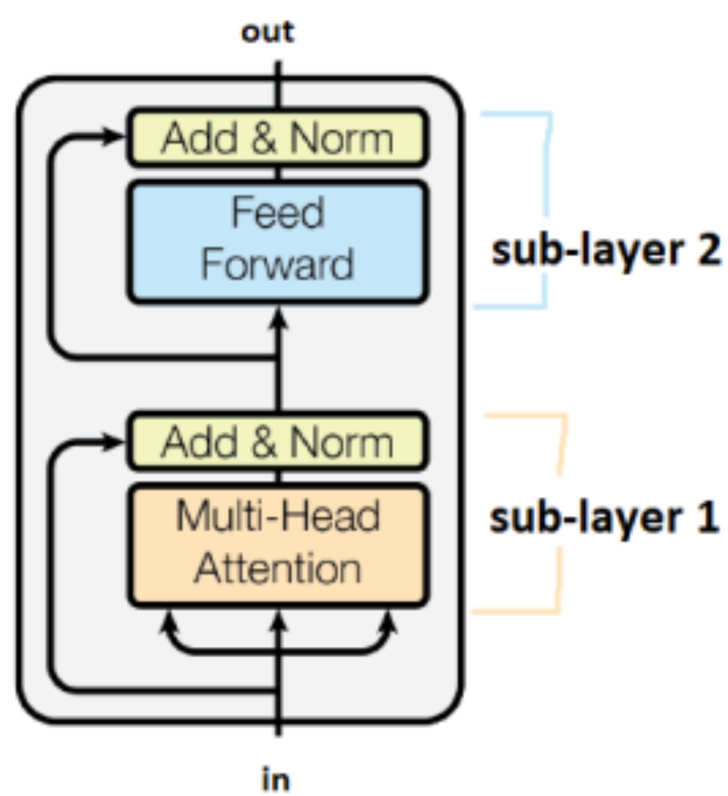
Deep contextualized word representations
https://arxiv.org › cs
by ME Peters - 2018 - Cited by 1683 - Related articles
Deep contextualized word representations. ... Our **word** vectors are learned functions of the internal states of a **deep** bidirectional language model (biLM), which is pre-trained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

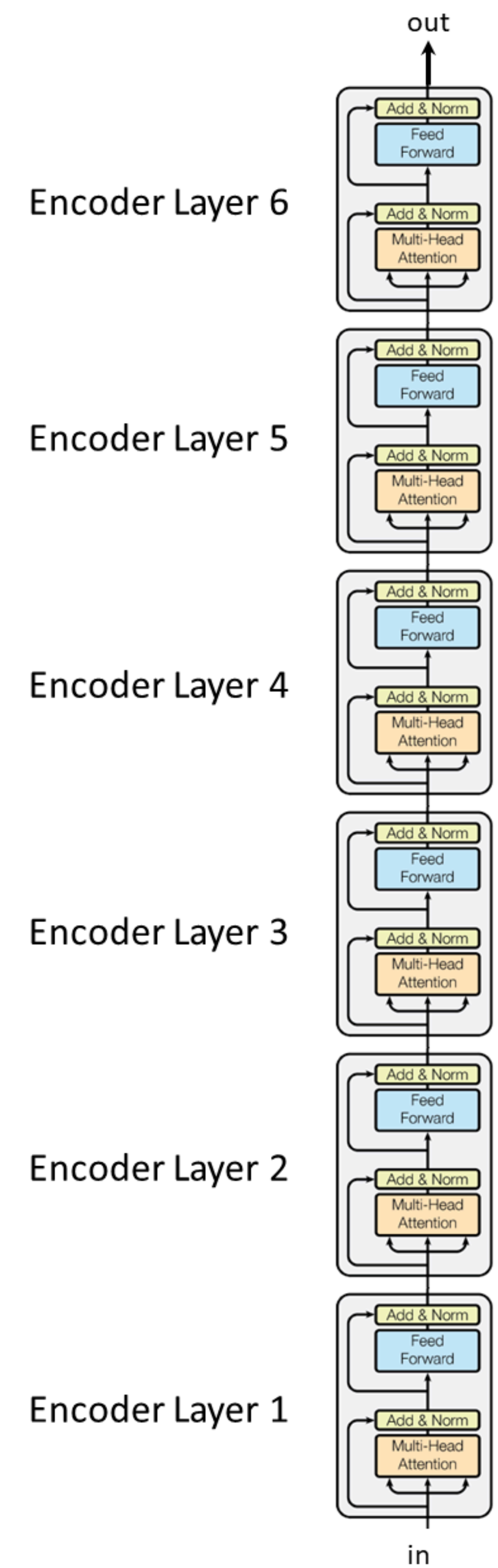
- BERT = **B**idirectional **E**ncoder **R**epresentations from **T**ransformers

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?

Overview

- Transformers





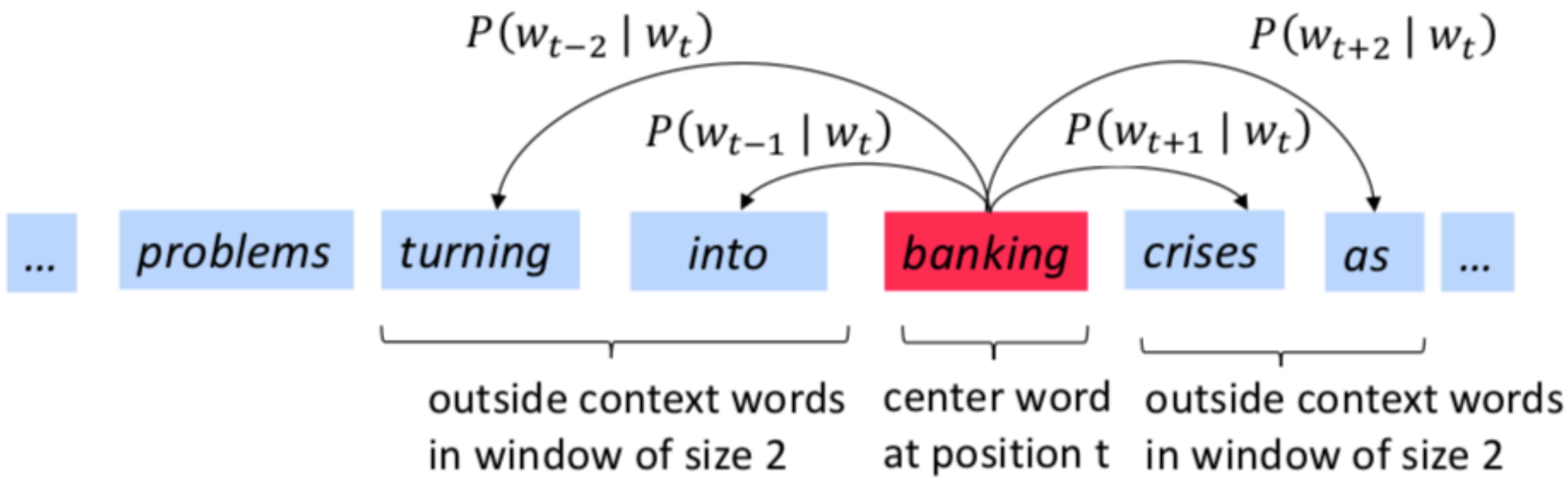
Attention Is All You Need

https://arxiv.org › cs ▾

by A Vaswani - 2017 - Cited by 4323 - Related articles

Jun 12, 2017 - **Attention Is All You Need**. The dominant sequence transduction models are based on complex recurrent or convolutional neural networks in an encoder-decoder configuration. The best performing models also connect the encoder and decoder through an **attention** mechanism.

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

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

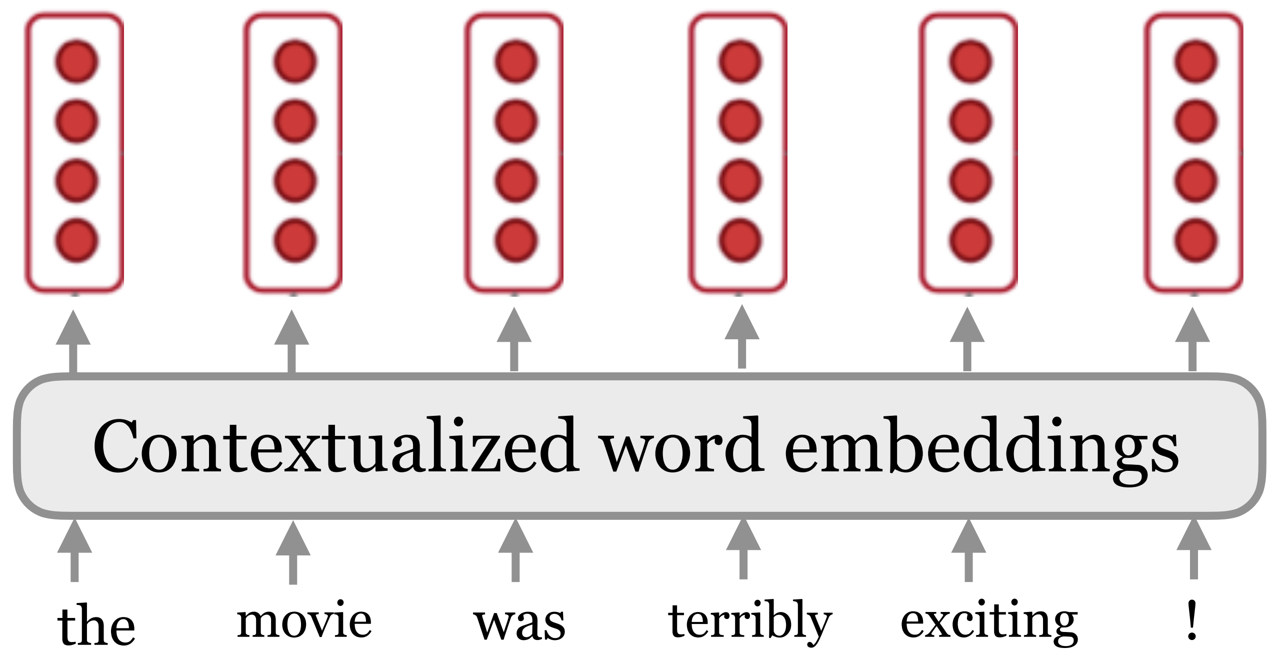
- Complex characteristics of word use: semantics, syntactic behavior, and connotations

- 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 ...

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

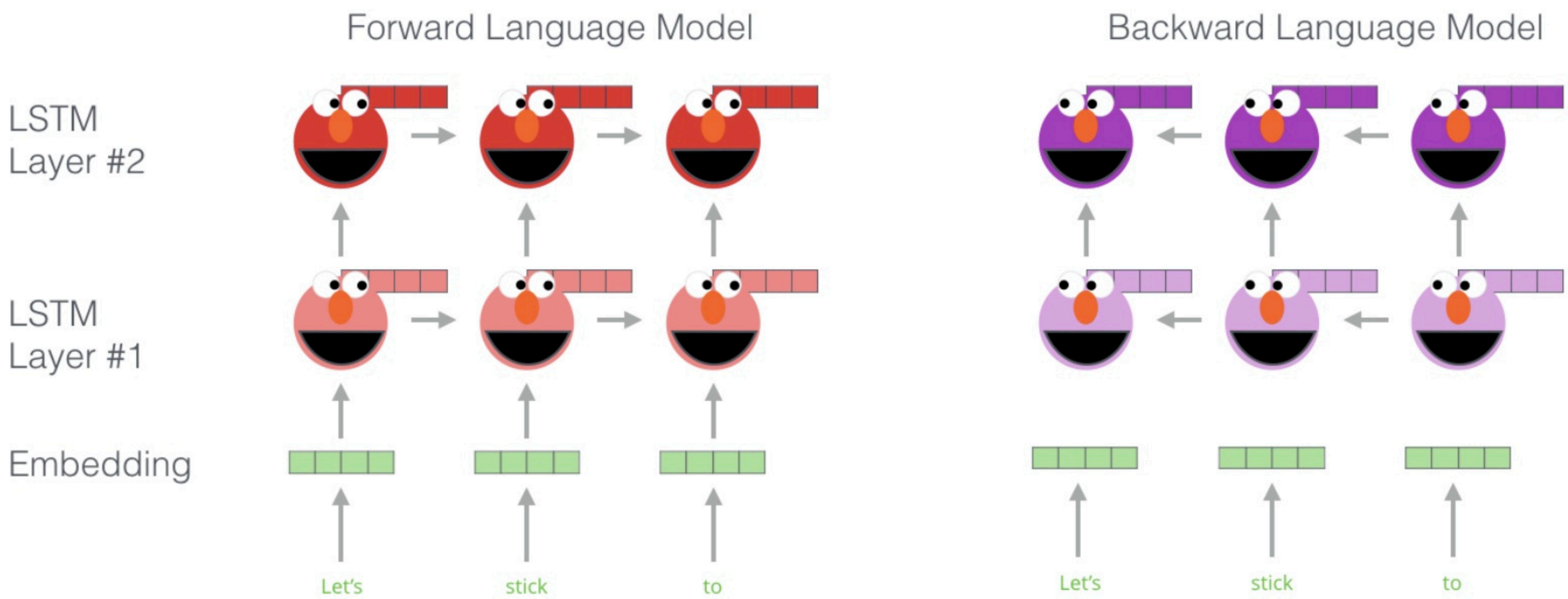
	Source	Nearest Neighbors
GloVe	play	playing, game, games, played, players, plays, player, Play, football, multiplayer
biLM	Chico Ruiz made a spectacular <u>play</u> on Alusik 's grounder {...}	Kieffer , the only junior in the group , was commended for his ability to hit in the clutch , as well as his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

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

ELMo

- NAACL'18: Deep contextualized word representations
- Key idea:
  - Train an 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



words in the sentence

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

input

softmax

How to use ELMo?

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

$$\mathbf{h}_{k,0}^{lM} = \mathbf{x}_k^{LM}, \mathbf{h}_{k,j}^{LM} = [\overrightarrow{\mathbf{h}}_{k,j}^{LM}; \overleftarrow{\mathbf{h}}_{k,j}^{LM}]$$

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

- γ^{task}: allows the task model to scale the entire ELMo vector
- s_j^{task}: softmax-normalized weights across layers
- 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)

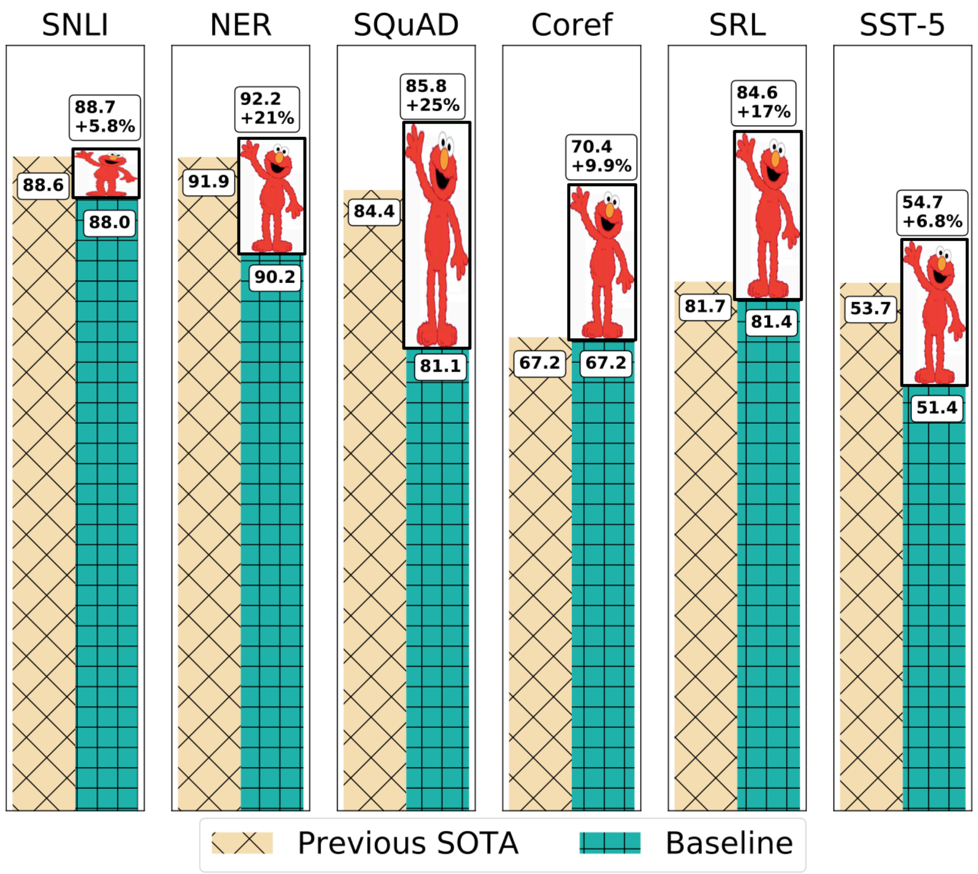
More details

- 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

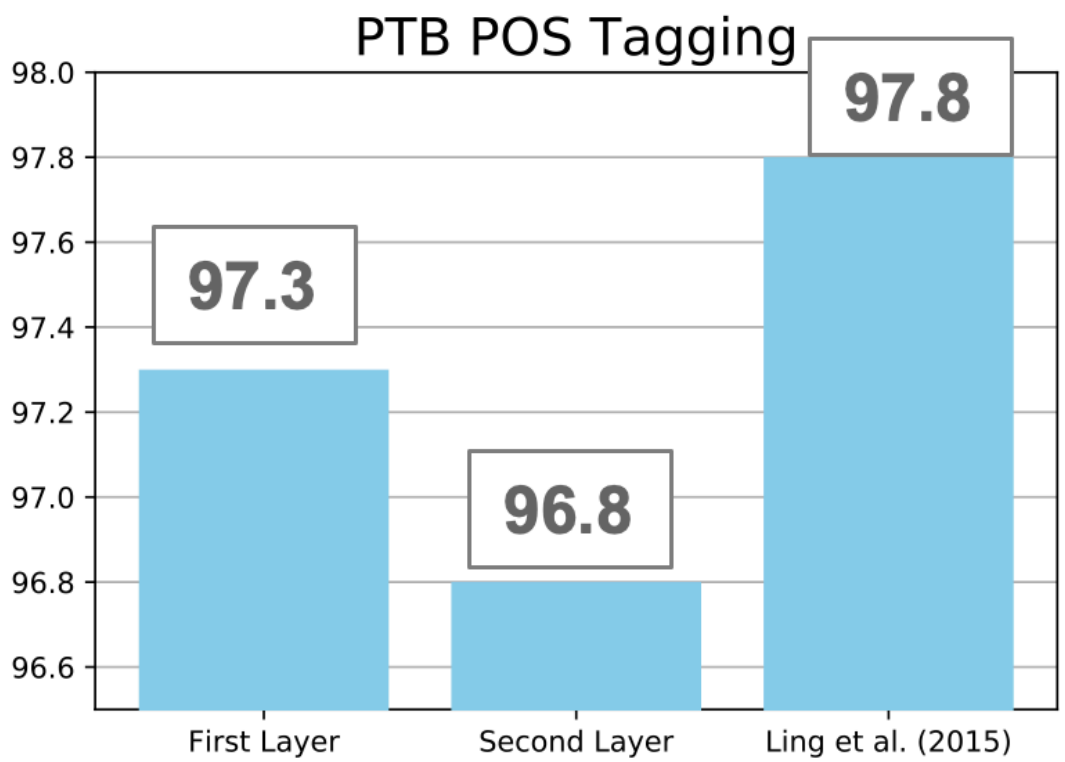
Experimental results

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%

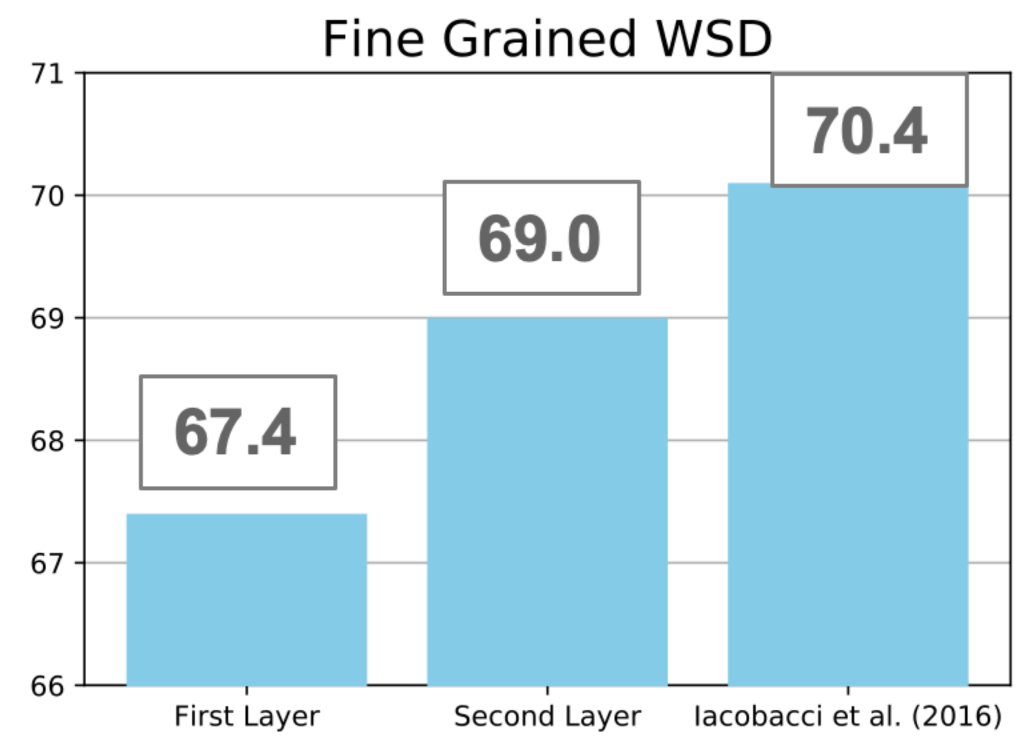
- SQuAD: question answering
- SNLI: natural language inference
- SRL: semantic role labeling
- Coref: coreference resolution
- NER: named entity recognition
- SST-5: sentiment analysis



Intrinsic Evaluation



First Layer > Second Layer



Second Layer > First Layer

syntactic information is better represented at lower layers
while semantic information is captured a 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?

#1. Unidirectional context vs bidirectional context

#2. LSTMs vs Transformers (will talk later)

#3. The weights are not freezed, called fine-tuning

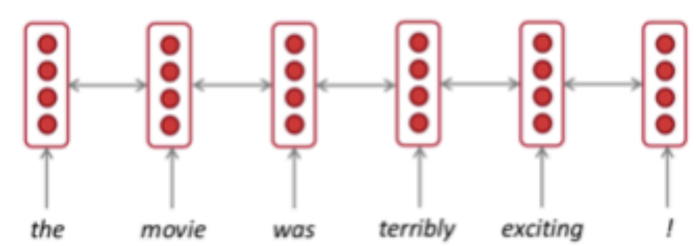
Bidirectional encoders

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

Lecture 9:

Bidirectional RNNs

Bidirectionality is important in language representations:

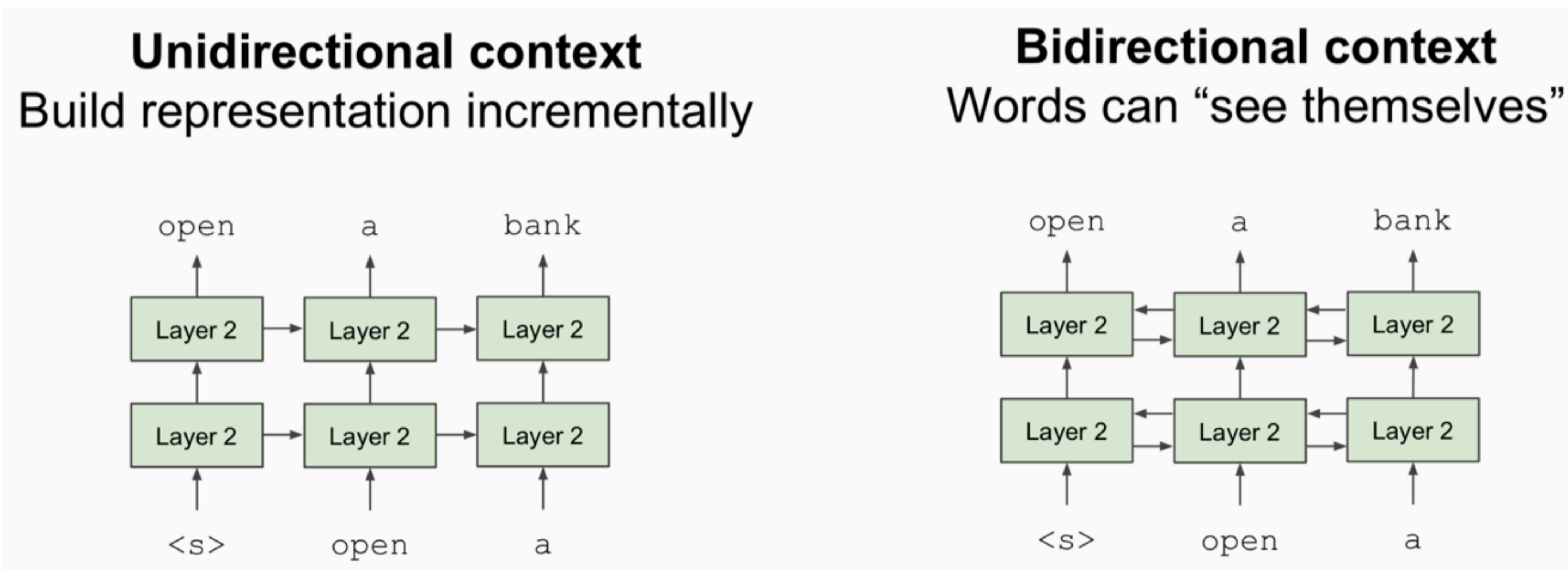


terribly:
- left context "the movie was"
- right context "exciting !"

Why are LMs unidirectional?

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

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

Masked language models (MLMs)

A little more complication:

- Rather than *always* replacing the chosen words with `[MASK]`, the data generator will do the following:
- 80% of the time: Replace the word with the `[MASK]` token, e.g., `my dog is hairy` → `my dog is [MASK]`
- 10% of the time: Replace the word with a random word, e.g., `my dog is hairy` → `my dog is apple`
- 10% of the time: Keep the word unchanged, e.g., `my dog is hairy` → `my dog is hairy`. The purpose of this is to bias the representation towards the actual observed word.

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]`

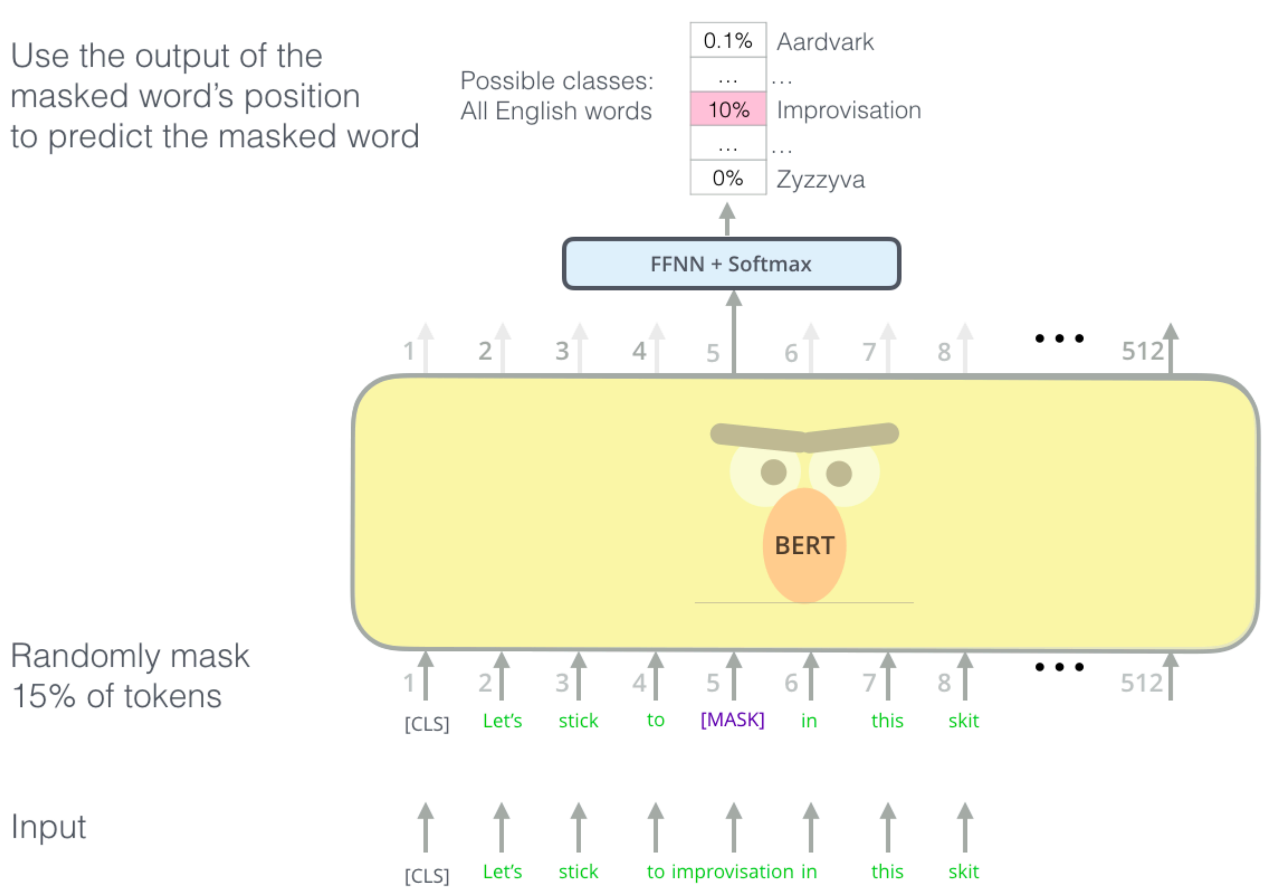
`penguin [MASK] are flight ##less birds [SEP]`

Label = `NotNext`

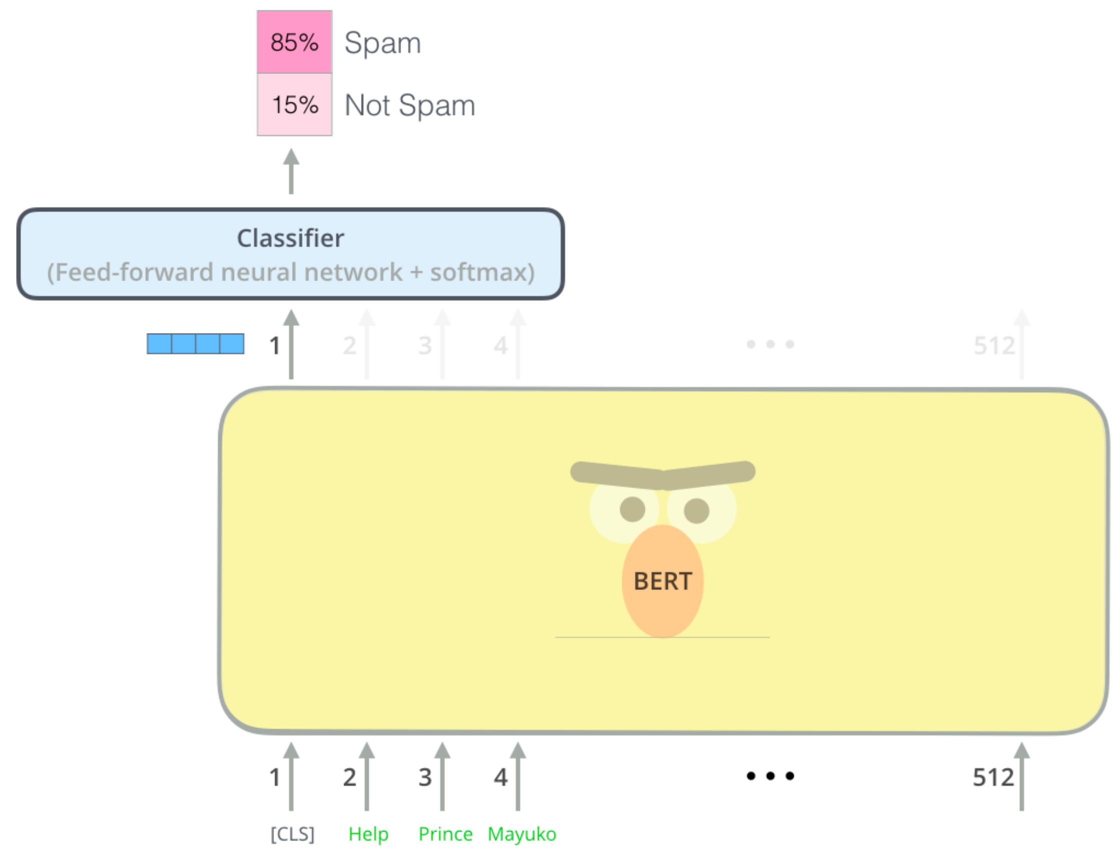
Recent papers show that NSP is not necessary…

(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

Pre-training and fine-tuning



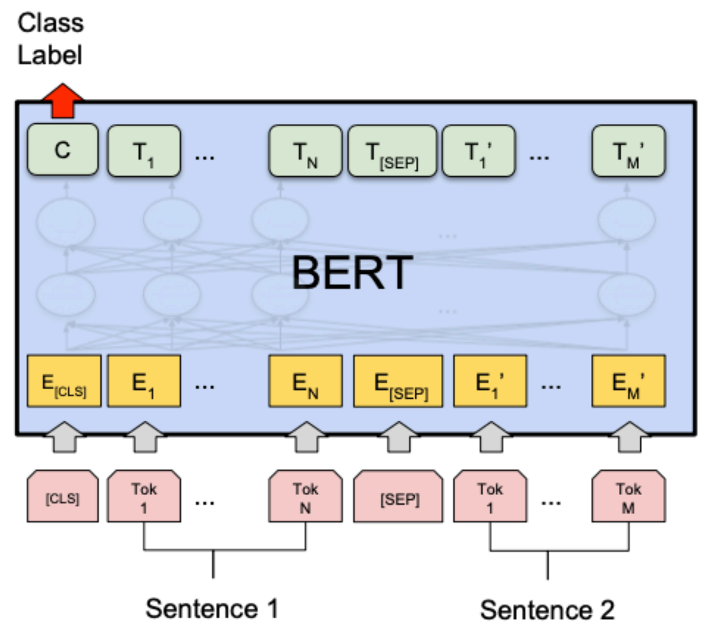
Pre-training



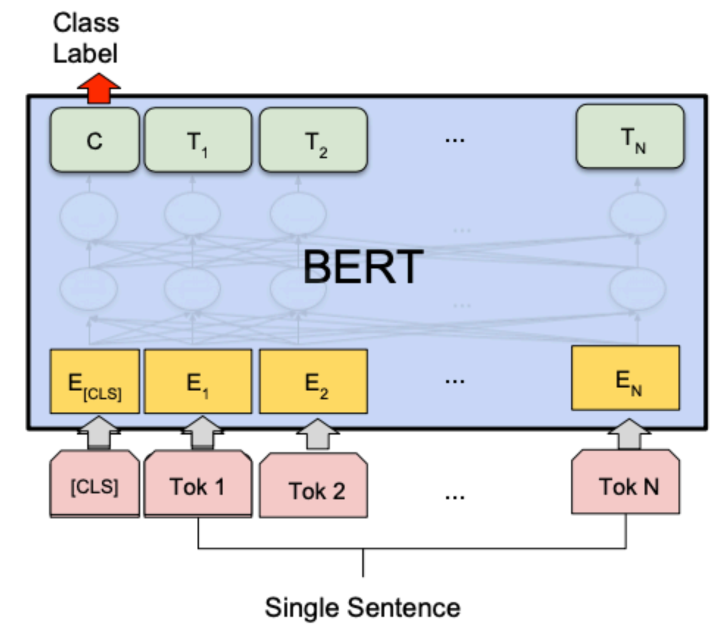
Fine-tuning

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

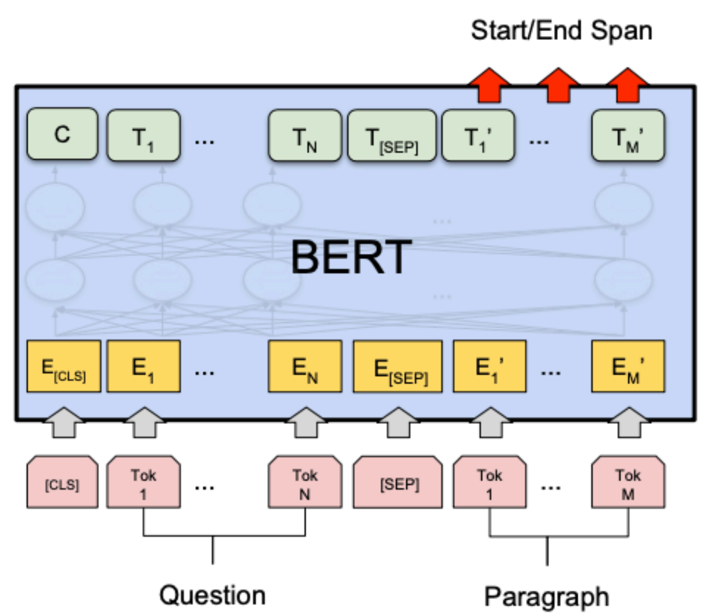
Applications



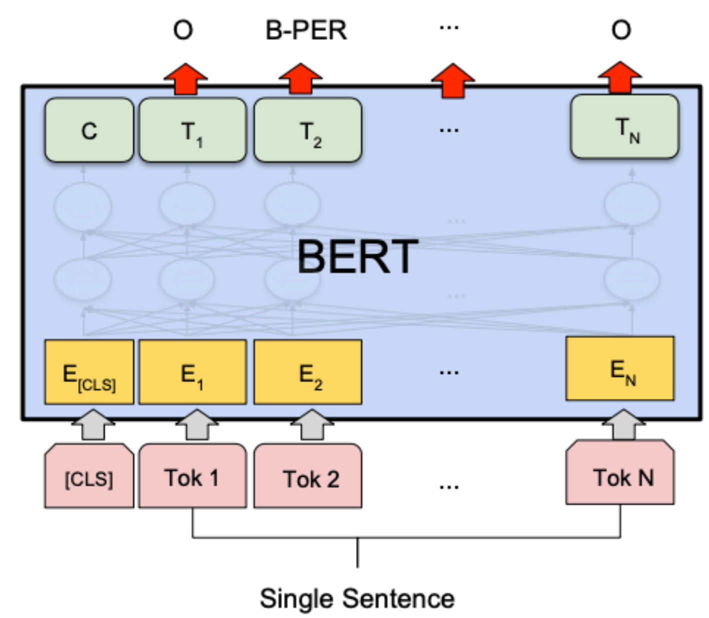
(a) Sentence Pair Classification Tasks:
MNLI, QQP, QNLI, STS-B, MRPC,
RTE, SWAG



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



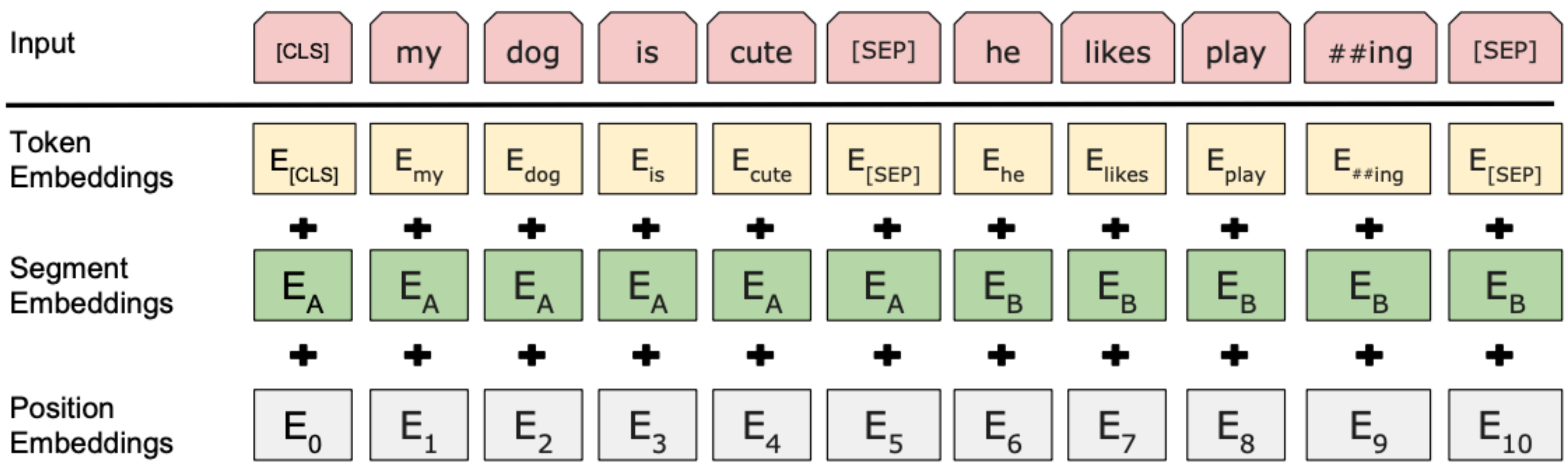
(c) Question Answering Tasks:
SQuAD v1.1



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

More details

- Input representations



- Use word pieces instead of words: playing => play ##ing ← Assignment 4

- Trained 40 epochs on Wikipedia (2.5B tokens) + BookCorpus (0.8B tokens)

- Released two model sizes: BERT_base, BERT_large

Experimental results

BiLSTM: 63.9

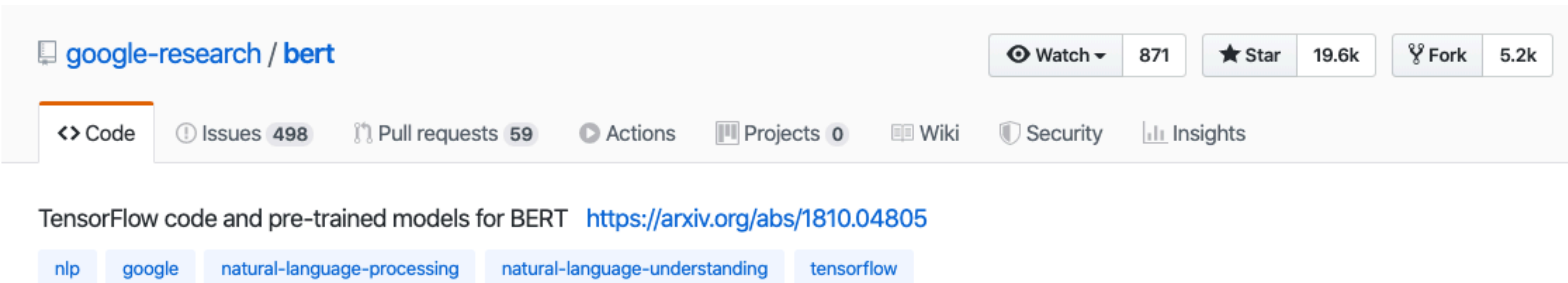
System	MNLI-(m/mm)	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE	**Average**
	392k	363k	108k	67k	8.5k	5.7k	3.5k	2.5k	-
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.9	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	88.1	91.3	45.4	80.0	82.3	56.0	75.2
$\text{BERT}_{\text{BASE}}$	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

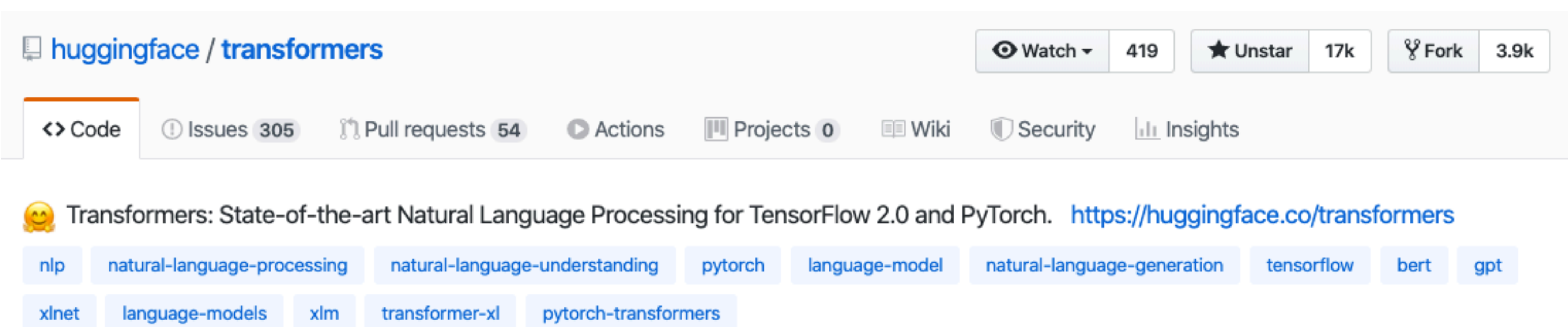
(Wang et al, 2018): GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding

Use BERT in practice

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



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

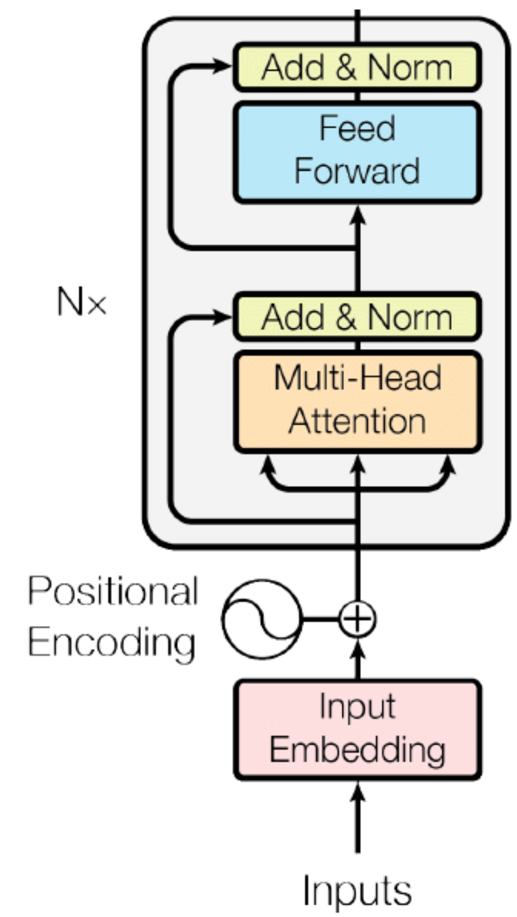


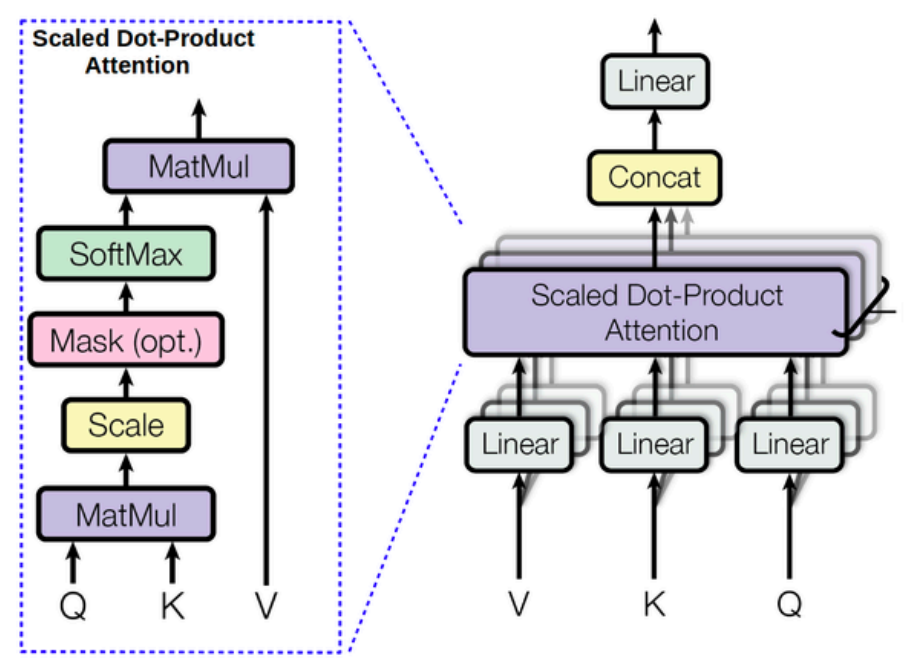
Contextualized word embeddings in context

- 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 (Anonymous)
- ...

Transformers

- NIPS’17: Attention is All You Need
- Key idea: Multi-head self-attention
- No recurrence structure any more so it trains much faster
- Originally proposed for NMT (encoder-decoder framework)
- Used as the base model of BERT (encoder only)





Useful Resources

nn.Transformer:

```
>>> transformer_model = nn.Transformer(nhead=16, num_encoder_layers=12)
>>> src = torch.rand((10, 32, 512))
>>> tgt = torch.rand((20, 32, 512))
>>> out = transformer_model(src, tgt)
```

nn.TransformerEncoder:

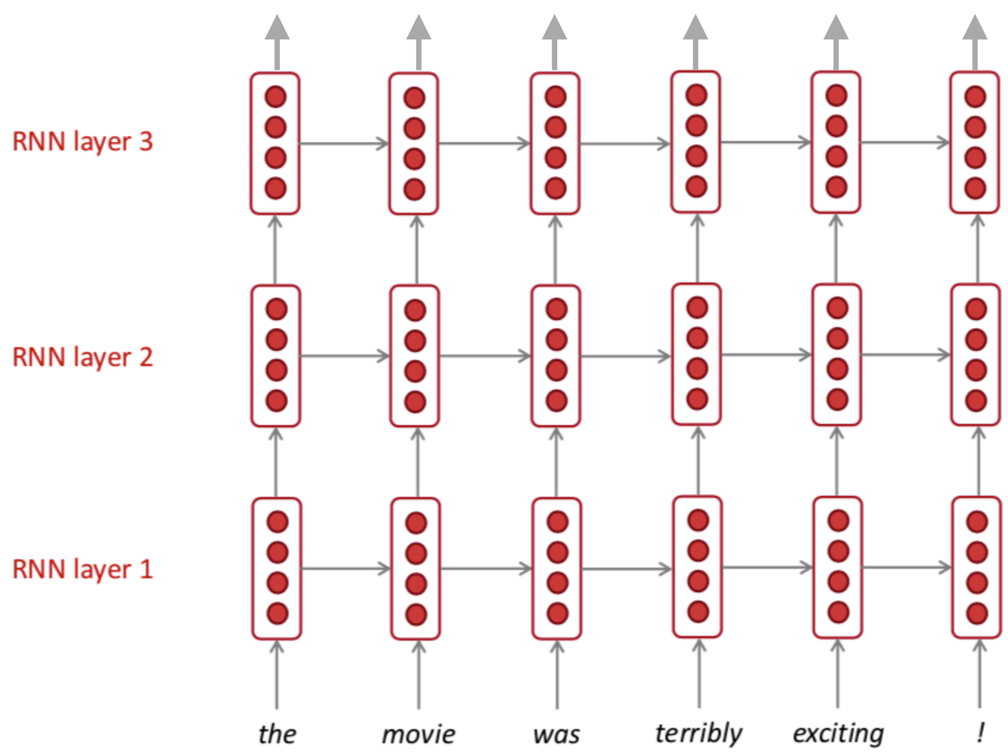
```
>>> encoder_layer = nn.TransformerEncoderLayer(d_model=512, nhead=8)
>>> transformer_encoder = nn.TransformerEncoder(encoder_layer, num_layers=6)
>>> src = torch.rand(10, 32, 512)
>>> out = transformer_encoder(src)
```

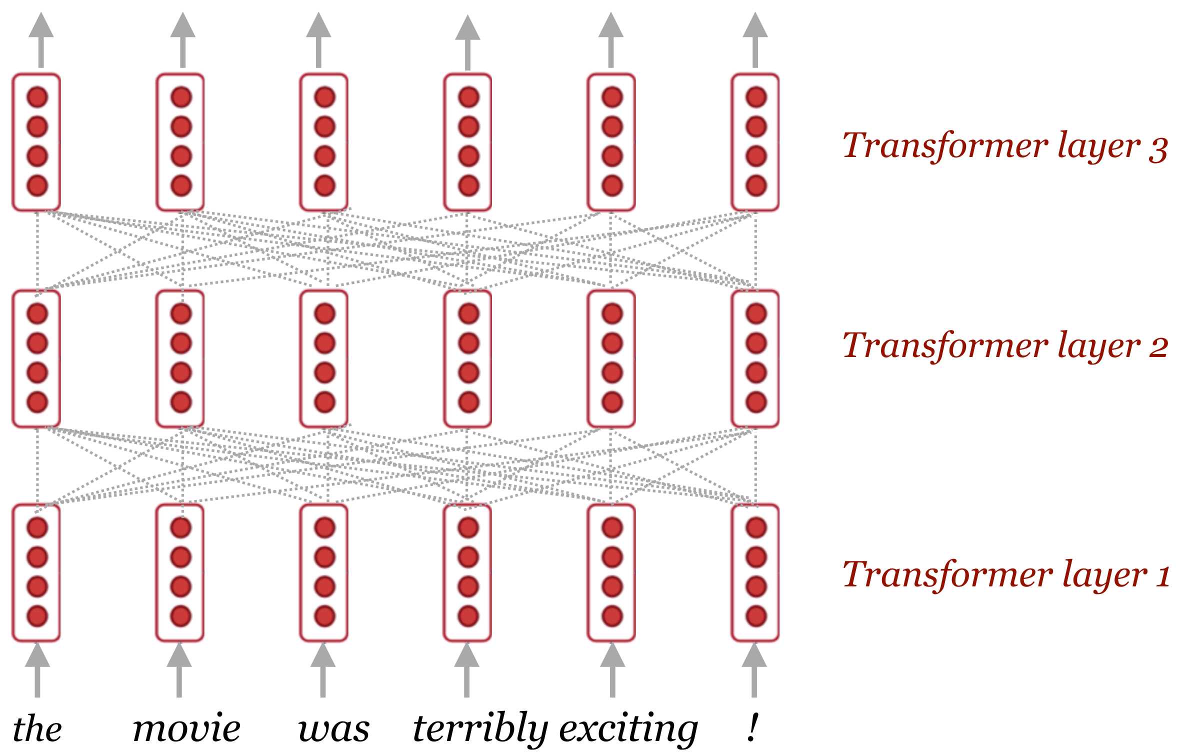
The Annotated Transformer:

http://nlp.seas.harvard.edu/2018/04/03/attention.html

A Jupyter notebook which explains how Transformer works line by line in PyTorch!

RNNs vs Transformers





Multi-head Self Attention

- **Attention**: a query q and a set of key-value (k_i, v_i) pairs to an output
- Dot-product attention:

$$A(q, K, V) = \sum_i \frac{e^{q \cdot k_i}}{\sum_j e^{q \cdot k_j}} v_i$$

$$K, V \in \mathbb{R}^{n \times d}, q \in \mathbb{R}^d$$

- If we have multiple queries:

$$A(Q, K, V) = \text{softmax}(QK^\intercal)V$$

$$Q \in \mathbb{R}^{n_Q \times d}, K, V \in \mathbb{R}^{n \times d}$$

- **Self-attention**: let's use each word as query and compute the attention with all the other words

= the word vectors themselves select each other

Multi-head Self Attention

- Scaled Dot-Product Attention:

$$A(Q, K, V) = \text{softmax}\left(\frac{QK^\intercal}{\sqrt{d}}\right)V$$

- Input: $X \in \mathbb{R}^{n \times d_{in}}$

$$A(XW^Q, XW^K, XW^V) \in \mathbb{R}^{n \times d}$$

$$W^Q, W^K, W^V \in \mathbb{R}^{d_{in} \times d}$$

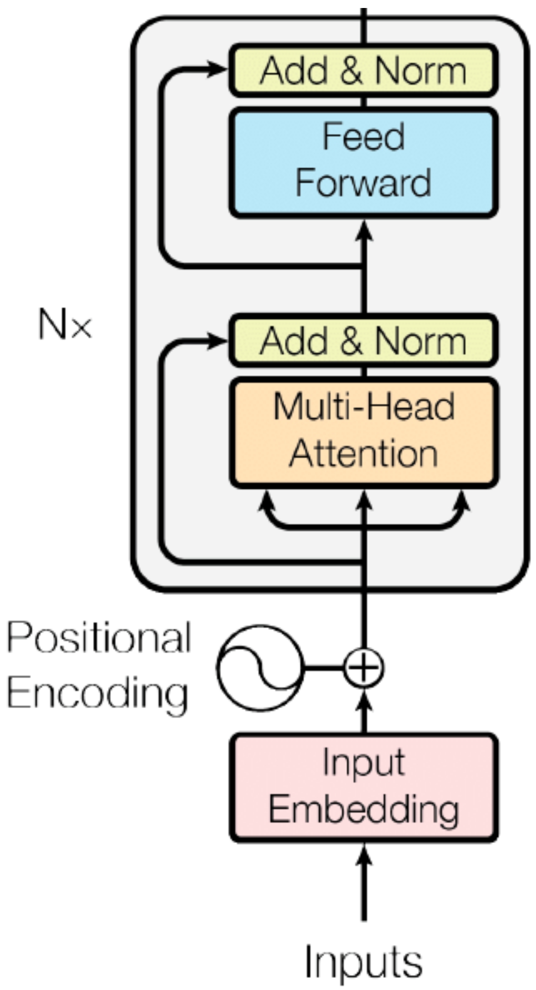
- Multi-head attention: using more than one head is always useful..

$$A(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O$$

$$\text{head}_i = A(XW_i^Q, XW_i^K, XW_i^V)$$

In practice, $h = 8, d = d_{out}/h, W^O = d_{out} \times d_{out}$

Putting it all together



- Each Transformer block has two sub-layers
  - Multi-head attention
  - 2-layer feedforward NN (with ReLU)
- Each sublayer has a residual connection and a layer normalization

$$\text{LayerNorm}(x + \text{SubLayer}(x))$$

- Input layer has a positional encoding

- BERT_base: 12 layers, 12 heads, hidden size = 768, 110M parameters
- BERT_large: 24 layers, 16 heads, hidden size = 1024, 340M parameters

(Ba et al, 2016): Layer Normalization

Have fun with using ELMo or BERT in your final project :)