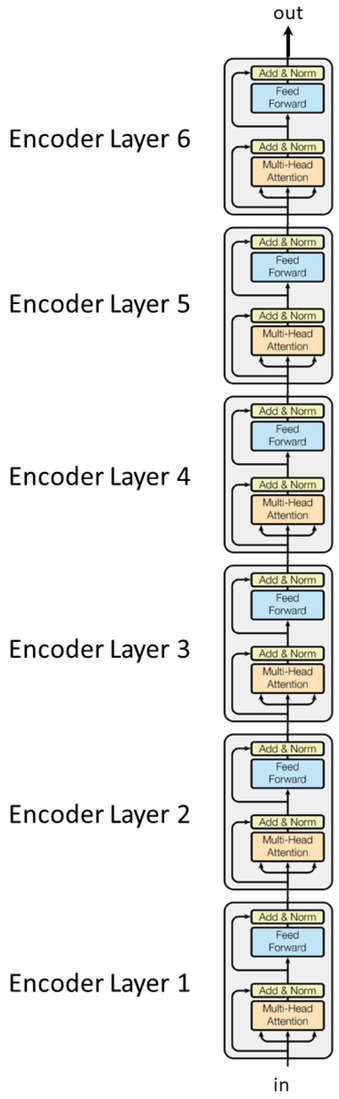
COS 484/584

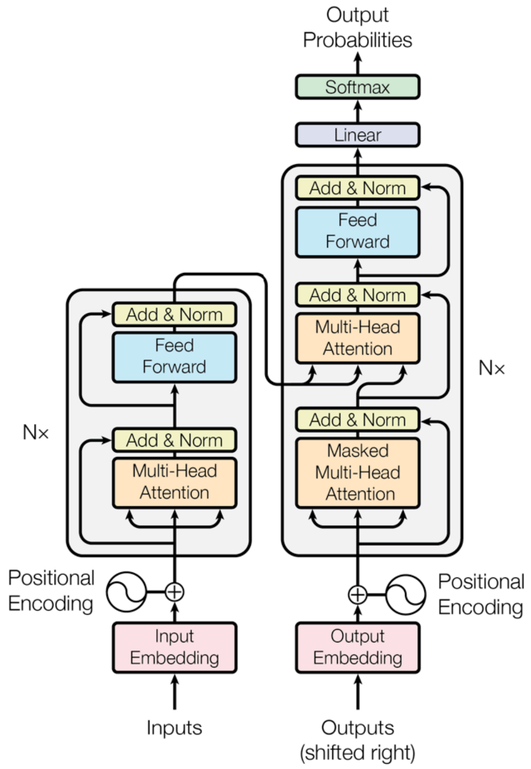
(Advanced) Natural Language Processing

L19: Contextualized Embeddings and Pre-training

Spring 2021

Transformer encoder-decoder (cont'd)



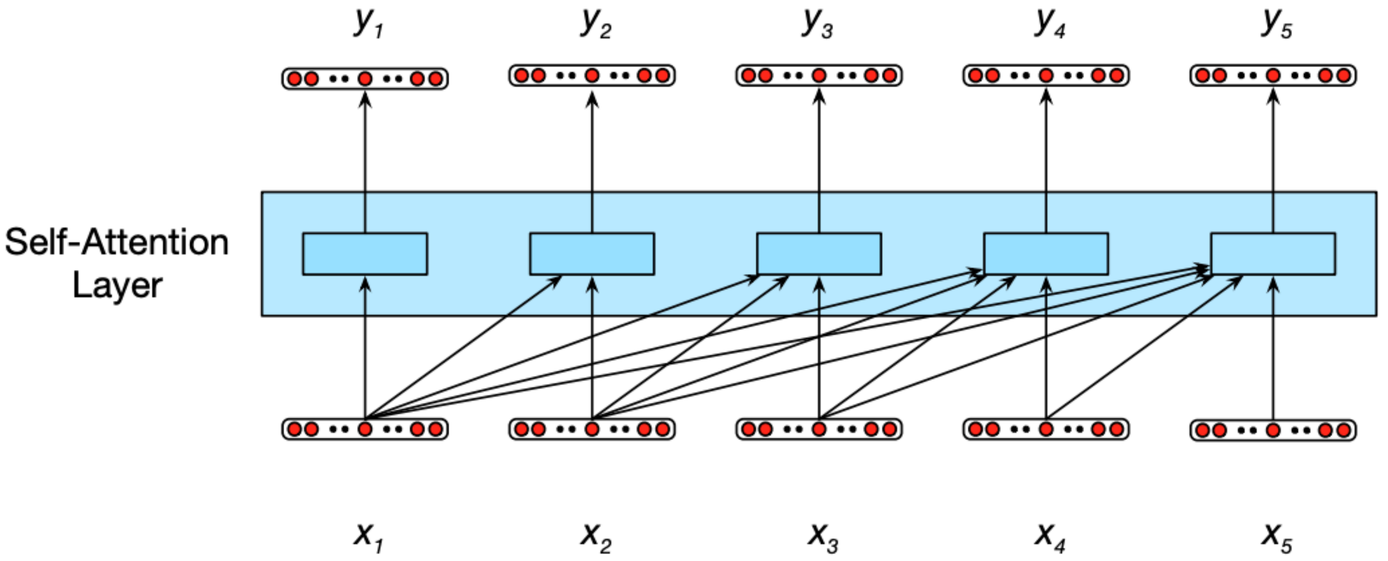


- Both encoder and decoder consist of *N* layers
- Each encoder layer has two sub-layers
  - Multi-head **self**-attention
  - FeedForward
- Each decoder layer has three sub-layers
  - Masked multi-head **self**-attention
  - Multi-head **cross**-attention
  - FeedForward
- Decoder: generate output probabilities for predicting next word

Vaswani et al., 2017: Attention Is All You Need

Masked multi-head attention

- Key point: you can't see the future words for the decoder!

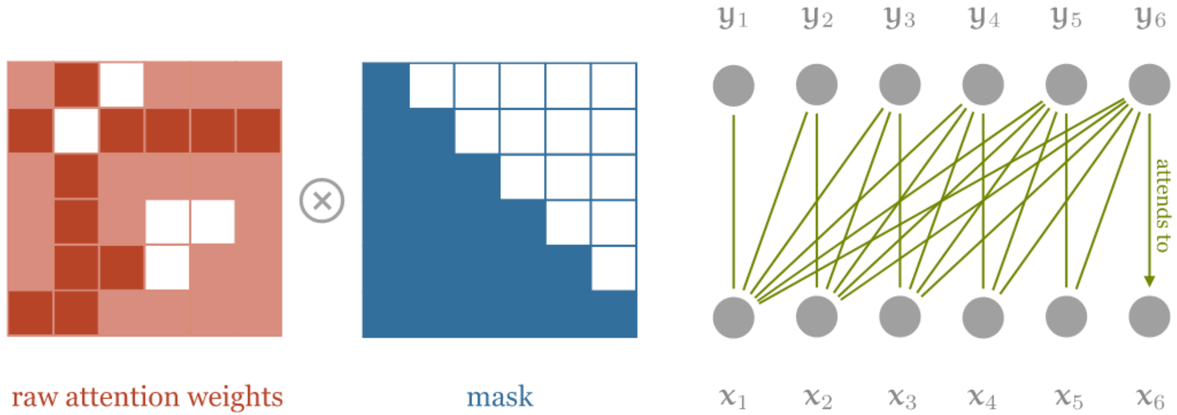


- Solution: for every $\mathbf{q}_i$, only attend to $\{(\mathbf{k}_j, \mathbf{v}_j)\}, j \leq i$

Masked multi-head attention

$$\mathbf{q}_i = W^Q \mathbf{x}_i, \mathbf{k}_i = W^K \mathbf{x}_i, \mathbf{v}_i = W^V \mathbf{x}_i$$

$$\alpha_{i,j} = \text{softmax}(\frac{\mathbf{q}_i \cdot \mathbf{k}_j}{\sqrt{d_k}})$$



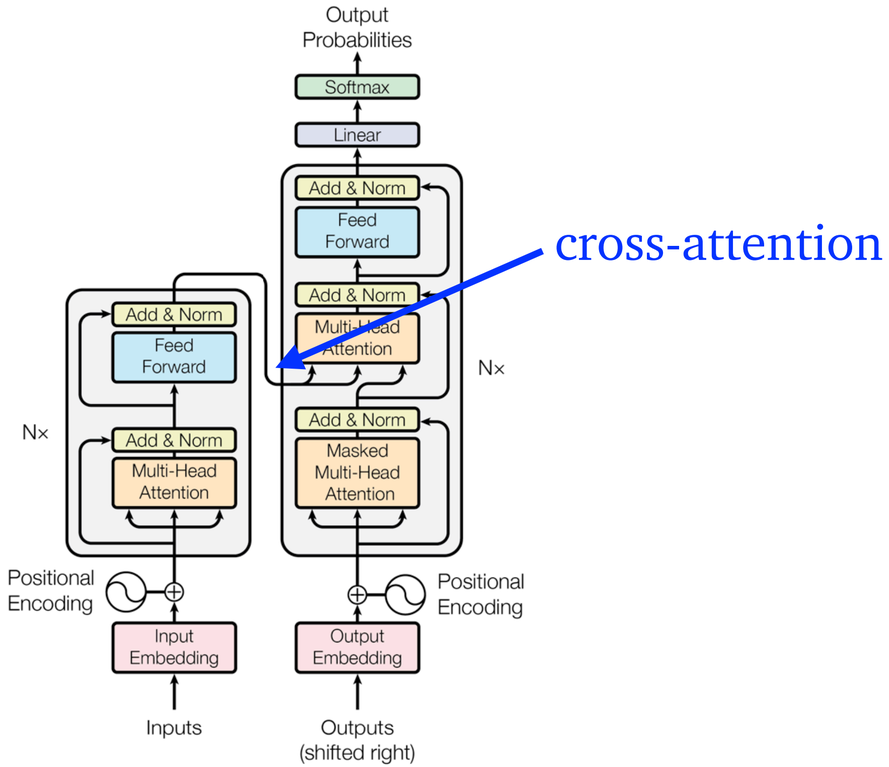
Efficient implementation: compute attention as we normally do, mask out attention to future words by setting attention scores to $-\infty$

```
dot = torch.bmm(queries, keys.transpose(1, 2))

indices = torch.triu_indices(t, t, offset=1)
dot[:, indices[0], indices[1]] = float('-inf')

dot = F.softmax(dot, dim=2)
```

http://peterbloem.nl/blog/transformers

Multi-head cross-attention

- $\mathbf{h}_1, \ldots, \mathbf{h}_m$: hidden states from encoder
- $\mathbf{x}_1, \ldots, \mathbf{x}_n$: hidden states from decoder



$$\mathbf{q}_i = W^Q \mathbf{x}_i, \mathbf{k}_i = W^K \mathbf{x}_i, \mathbf{v}_i = W^V \mathbf{x}_i$$

$$\alpha_{i,j} = \text{softmax}(\frac{\mathbf{q}_i \cdot \mathbf{k}_j}{\sqrt{d_k}})$$

$$\mathbf{q}_i = W^Q \mathbf{x}_i \quad \mathbf{k}_j = W^K \mathbf{h}_j, \mathbf{v}_j = W^V \mathbf{h}_j$$

$$\alpha_{i,j} = \text{softmax}(\frac{\mathbf{q}_i \cdot \mathbf{k}_j}{\sqrt{d_k}})$$

$$\mathbf{y}_i = \sum_{j=1}^{m} \alpha_{i,j} \mathbf{v}_j$$

Transformers: machine translation

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [15]	23.75			
Deep-Att + PosUnk [32]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [31]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [8]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [26]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [32]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [31]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [8]	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	**$3.3 \cdot 10^{18}$**	
Transformer (big)	**28.4**	**41.0**	$2.3 \cdot 10^{19}$	

Vaswani et al., 2017: Attention Is All You Need

Transformers: document generation

Model	Test perplexity	ROUGE-L
seq2seq-attention, $L = 500$	5.04952	12.7
Transformer-ED, $L = 500$	2.46645	34.2
Transformer-D, $L = 4000$	2.22216	33.6
Transformer-DMCA, no MoE-layer, $L = 11000$	2.05159	36.2
Transformer-DMCA, MoE-128, $L = 11000$	1.92871	37.9
Transformer-DMCA, MoE-256, $L = 7500$	1.90325	38.8

Very large gains compared to seq2seq-attention with LSTMs!

Liu et al., 2018: Generating Wikipedia by Summarizing Long Sequences

This lecture

- ELMo = **E**mbeddings from **L**anguage **Mo**dels
- BERT = **B**idirectional **E**ncoder **R**epresentations from **T**ransformers

Deep contextualized word representations [PDF] arxiv.org

ME Peters, M Neumann, M Iyyer, M Gardner… - arXiv preprint arXiv …, 2018 - arxiv.org

We introduce a new type of deep contextualized word representation that models both (1) complex characteristics of word use (eg, syntax and semantics), and (2) how these uses vary across linguistic contexts (ie, to model polysemy). Our word vectors are learned functions of …

☆ Cited by 6367 Related articles All 20 versions »

Bert: Pre-training of **deep bidirectional transformers** for **language understanding** [PDF] arxiv.org

J Devlin, MW Chang, K Lee, K Toutanova - arXiv preprint arXiv …, 2018 - arxiv.org

We introduce a new **language** representation model called BERT, which stands for **Bidirectional** Encoder Representations from **Transformers**. Unlike recent **language** representation models, BERT is designed to pre-train **deep bidirectional** representations …

☆ Cited by 17552 Related articles All 26 versions »

Today's key concepts

- Contextualized word embeddings
- Pre-training and fine-tuning

What's wrong with word2vec?

- One vector for each word type

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

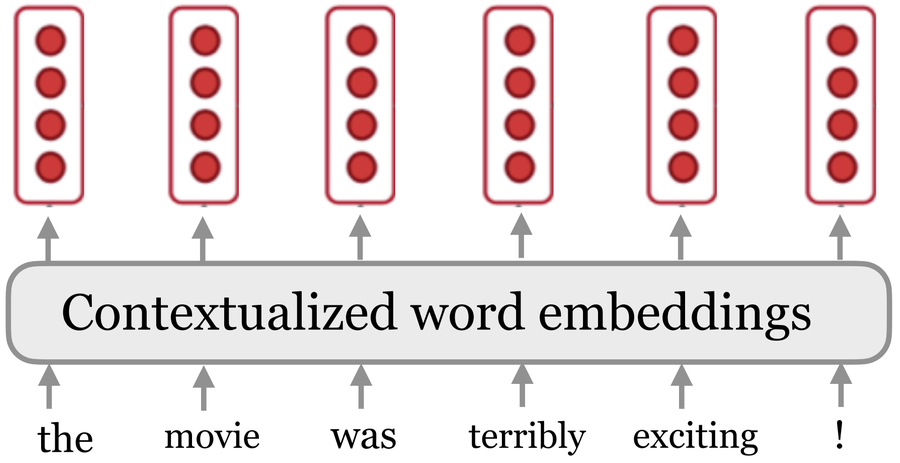
- Complex characteristics of word use: syntax and semantics

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

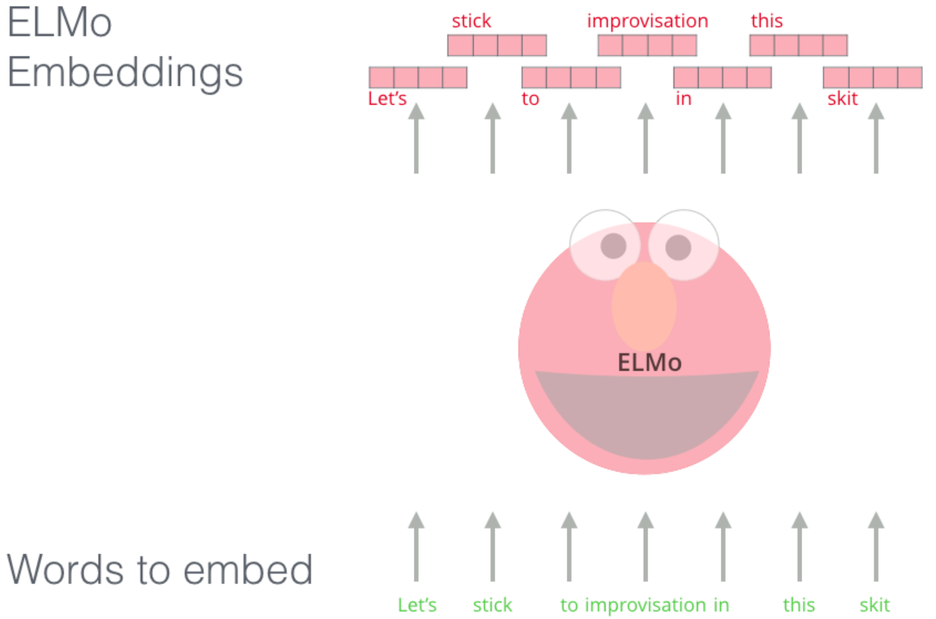
mouse^{1} : .... a *mouse* controlling a computer system in 1968.
mouse^{2} : .... a quiet animal like a *mouse*
bank^{1} : ...a *bank* can hold the investments in a custodial account ...
bank^{2} : ...as agriculture burgeons on the east *bank*, the river ...

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

Sent #1: Chico Ruiz made a spectacular **play** on Alusik's grounder {. . . } $v(\text{play}) = ?$

Sent #2: Olivia De Havilland signed to do a Broadway **play** for Garson {. . . } $v(\text{play}) = ?$

Sent #3: Kieffer was commended for his ability to hit in the clutch , as well as his all-round excellent **play** {. . . } $v(\text{play}) = ?$

Sent #4: {. . . } they were actors who had been handed fat roles in a successful **play** {. . . } $v(\text{play}) = ?$

Sent #5: Concepts **play** an important role in all aspects of cognition {. . . } $v(\text{play}) = ?$

Zoom poll

Sent #1: Chico Ruiz made a spectacular **play** on Alusik's grounder {. . . }

Which of the following v(play) is expected to have the most similar vector to the first one?

(a) Olivia De Havilland signed to do a Broadway **play** for Garson {. . . }

(b) Kieffer was commended for his ability to hit in the clutch , as well as his all-round excellent **play** {. . . }

(c) {. . . } they were actors who had been handed fat roles in a successful **play** {. . . }

(d) Concepts **play** an important role in all aspects of cognition {. . . }

(b) is correct.

Contextualized word embeddings

	Source	Nearest Neighbors
GloVe	play	playing, game, games, played, players, plays, player, Play, football, multiplayer
biLM	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 .

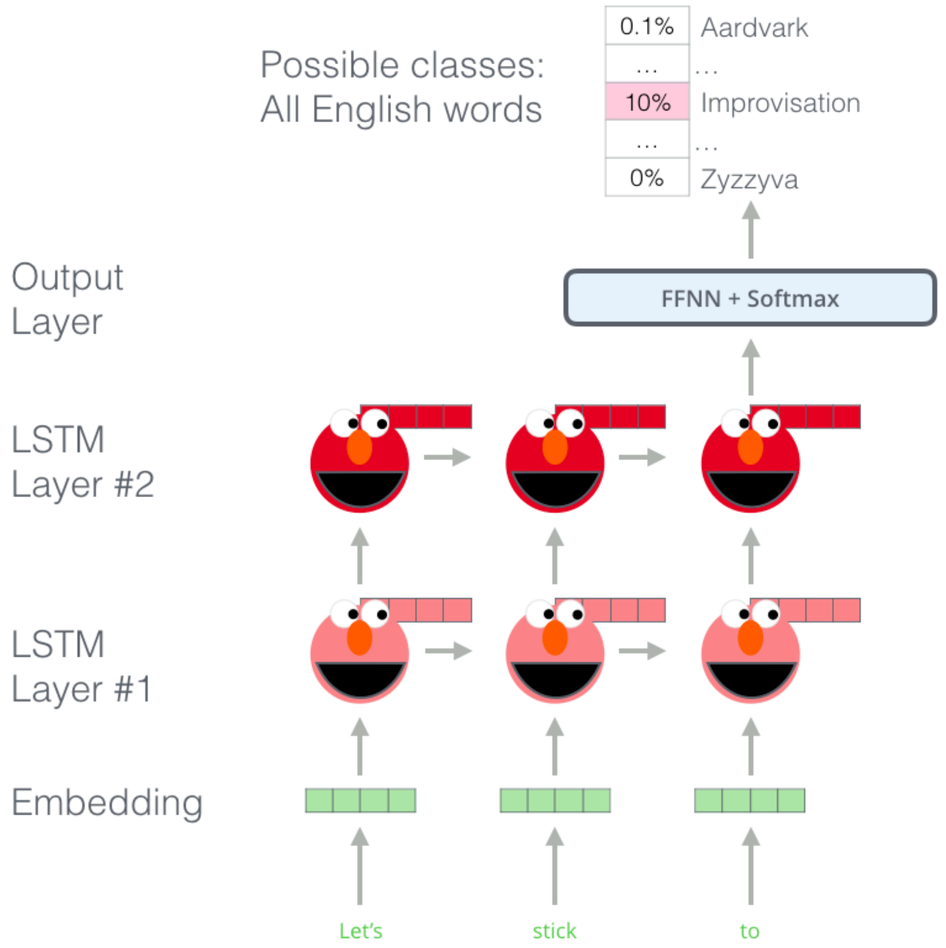
(Peters et al, 2018): Deep contextualized word representations

How can we get these contextualized embeddings?

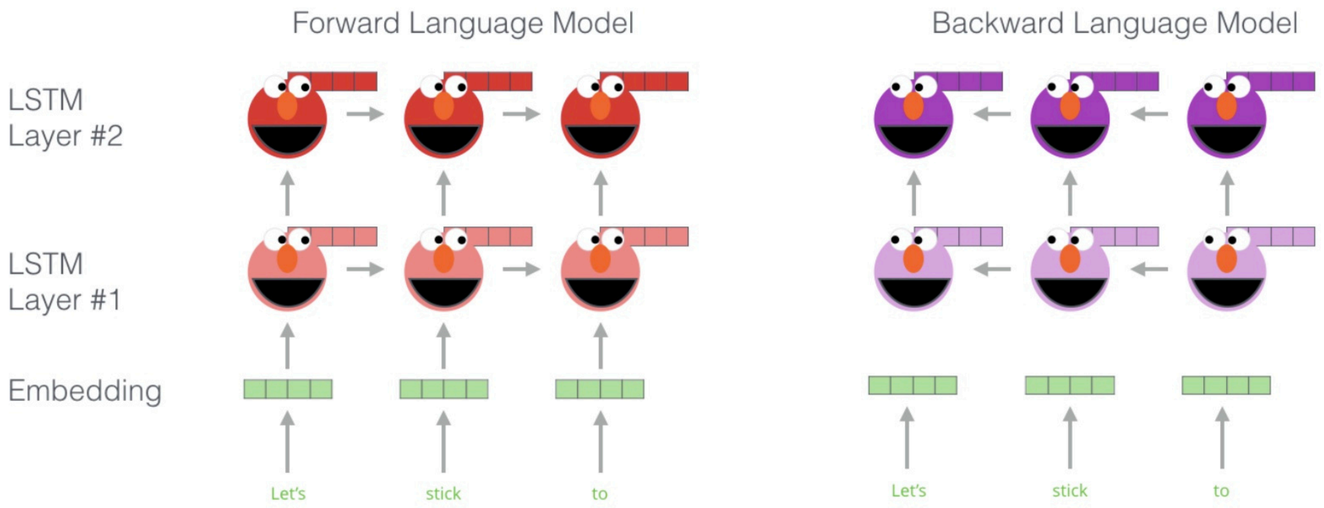
The key idea of ELMo:

- Train *two* stacked LSTM-based language models on a **large** corpus
- Use the **hidden states** of the LSTMs for each token to compute a vector representation of each word

Q: Why use the hidden representations of language models?



How can we get these contextualized embeddings?



words in the sentence

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

input embeddings = output embeddings

How to get ELMo embeddings?

input embeddings

hidden states

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

# of layers

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

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

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

Both are parameters to be learned

How to get ELMo embeddings?

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

Q: Why use both forward and backward language models?

Because it is important to model both left and right context!

Bidirectionality is VERY crucial in language understanding tasks!

Q: Why use the weighted average of different layers instead of just the top layer?

Because different layers are expected to encode different information.

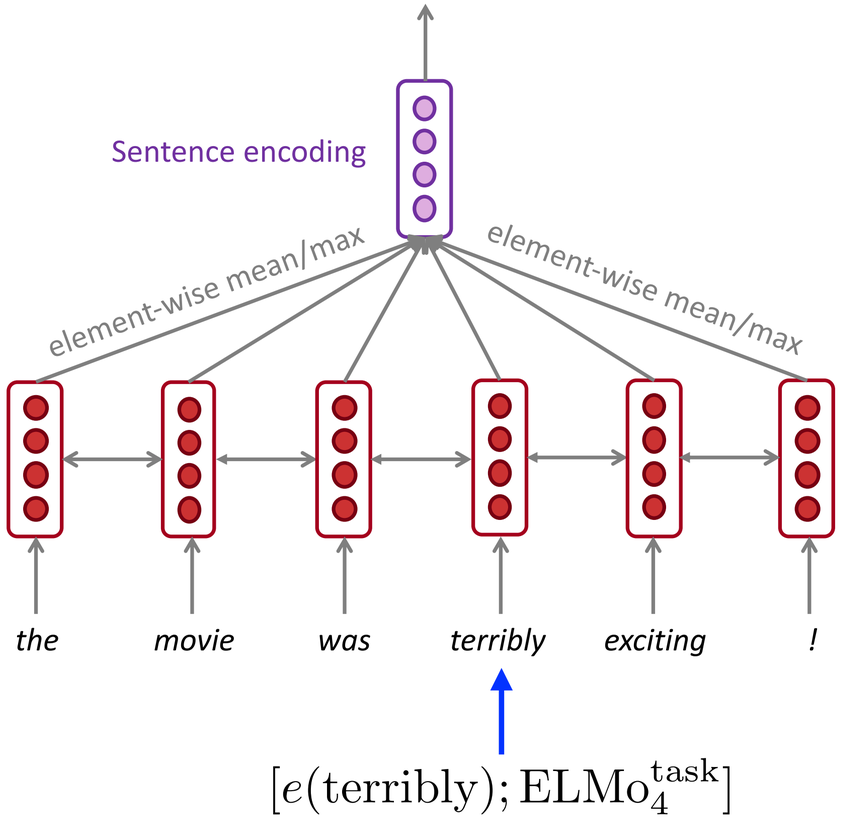
We will see some examples soon!

ELMo: pre-training and the 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 $f : (w_1, w_2, \ldots, w_n) \longrightarrow \mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^d$ and feed them into any neural models just like word2vec
  - 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 the use

Example: A BiLSTM model for sentiment classification



What is pre-training?

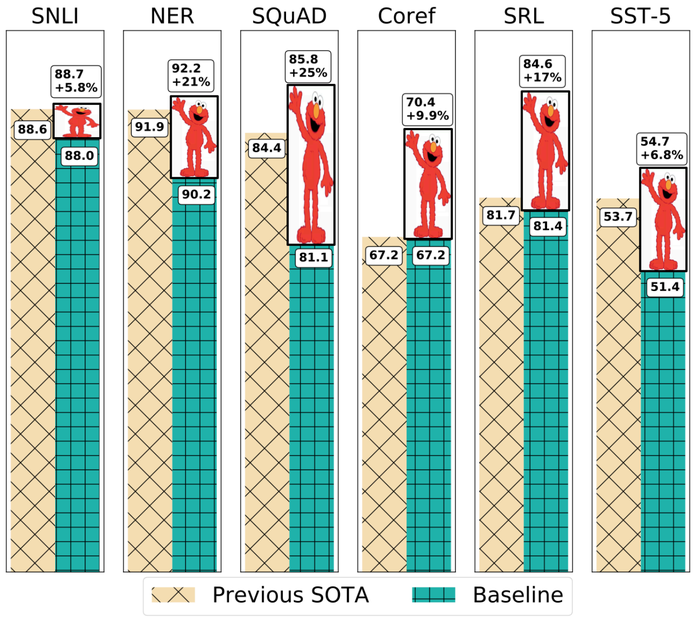
- “Pre-train” a model on a large dataset for task X, then “fine-tune” it on a dataset for task Y
- Key idea: X is somewhat related to Y, so a model that can do X will have some good neural representations for Y as well
- ImageNet pre-training is huge in computer vision: learning generic visual features for recognizing objects
- Word2vec can be seen as pre-training: learn vectors with the skip-gram objective on large dataset (task X), then use them as a part of a neural network for sentiment classification or any other task (task Y)

Slide credit: Greg Durrett

ELMo: 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



What information does ELMo encode?

Let's compare the hidden representations learned from ELMo. Which representations are better at predicting part-of-speech tags (verbs, nouns, adjectives)?

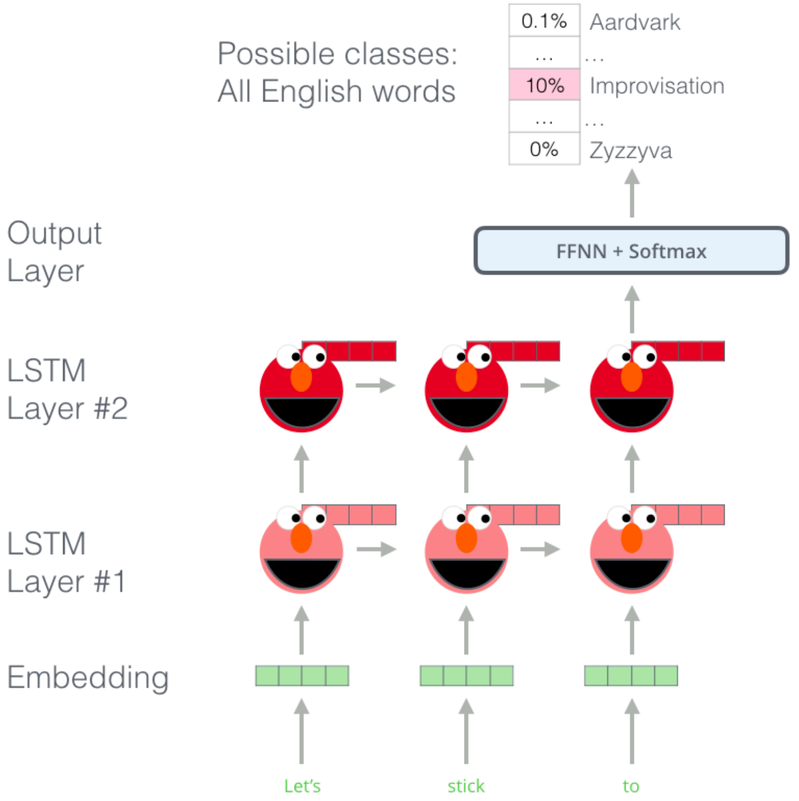
(a) First layer $\mathbf{h}_1$

(b) Second layer $\mathbf{h}_2$

(c) $\mathbf{h}_1$ and $\mathbf{h}_2$ are equally good

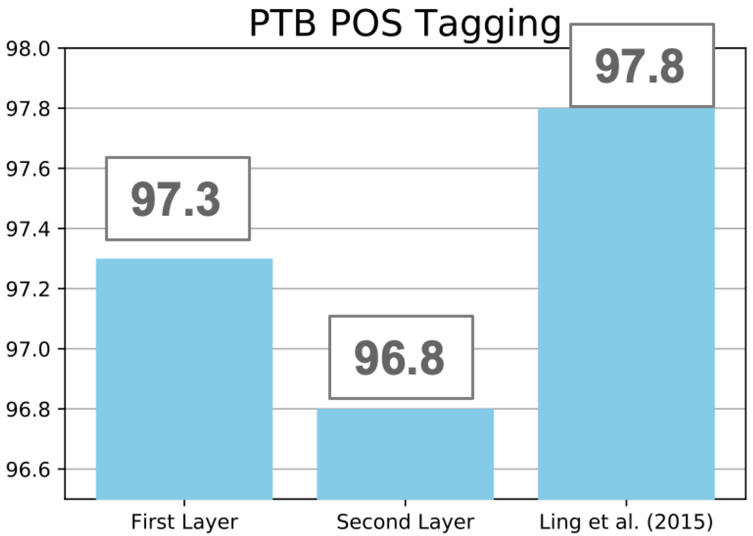
(d) hard to tell

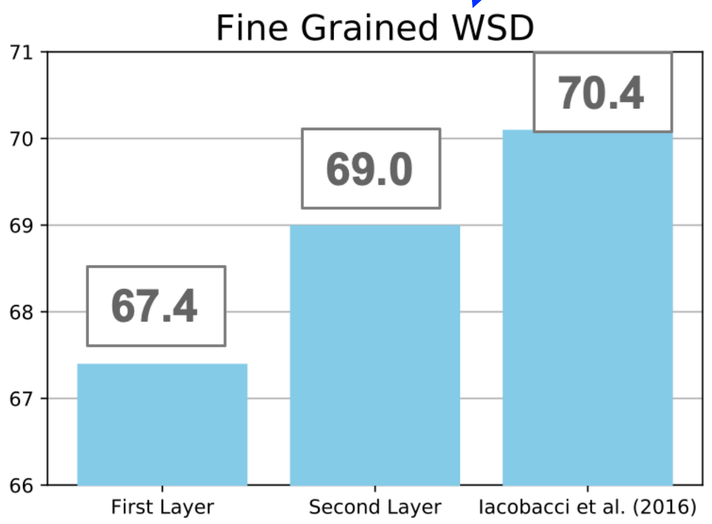
(a) is correct.



What information does ELMo encode?

WSD = word sense disambiguation





First Layer > Second Layer

Second Layer > First Layer

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

Use ELMo models

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)
```

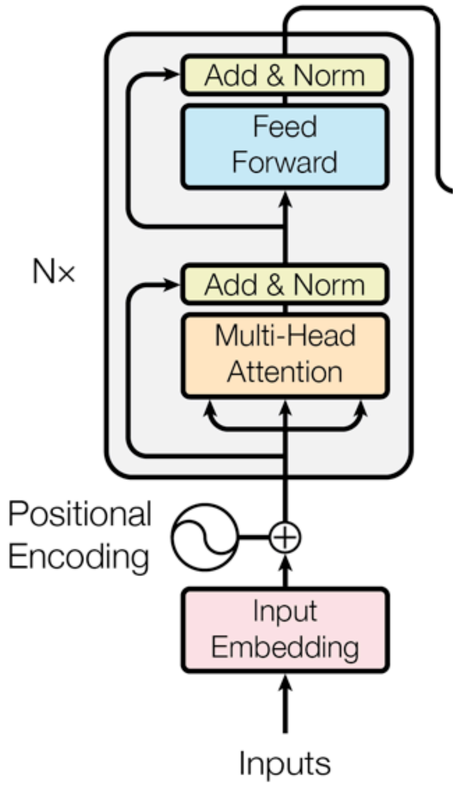
Pre-training is expensive but it only needs to be done once!

People can download the pre-trained models and use them in their downstream applications.

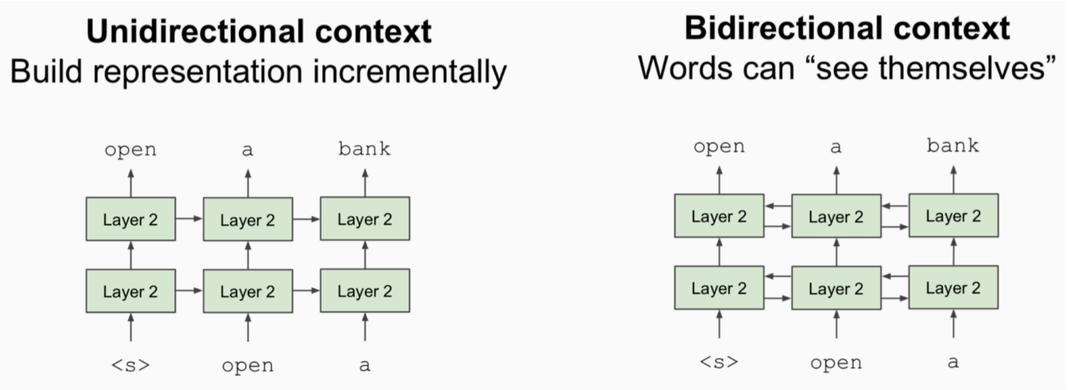
What is BERT?

- 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
- Two new pre-training objectives:
  - **Masked language model (MLM)**
  - Next sentence prediction (NSP)
    - Later work shows that NSP hurts performance, so we omit it here
- **Use BERT for fine-tuning, instead of word embeddings!**

Bidirectional encoder



- Let's use a Transformer encoder with self-attention
- Language models only use left context or right context (although ELMo used two independent LMs from each direction).



- If the word "a" is already seen, what to predict then?
- How can we build good representations that allow us to see both sides?

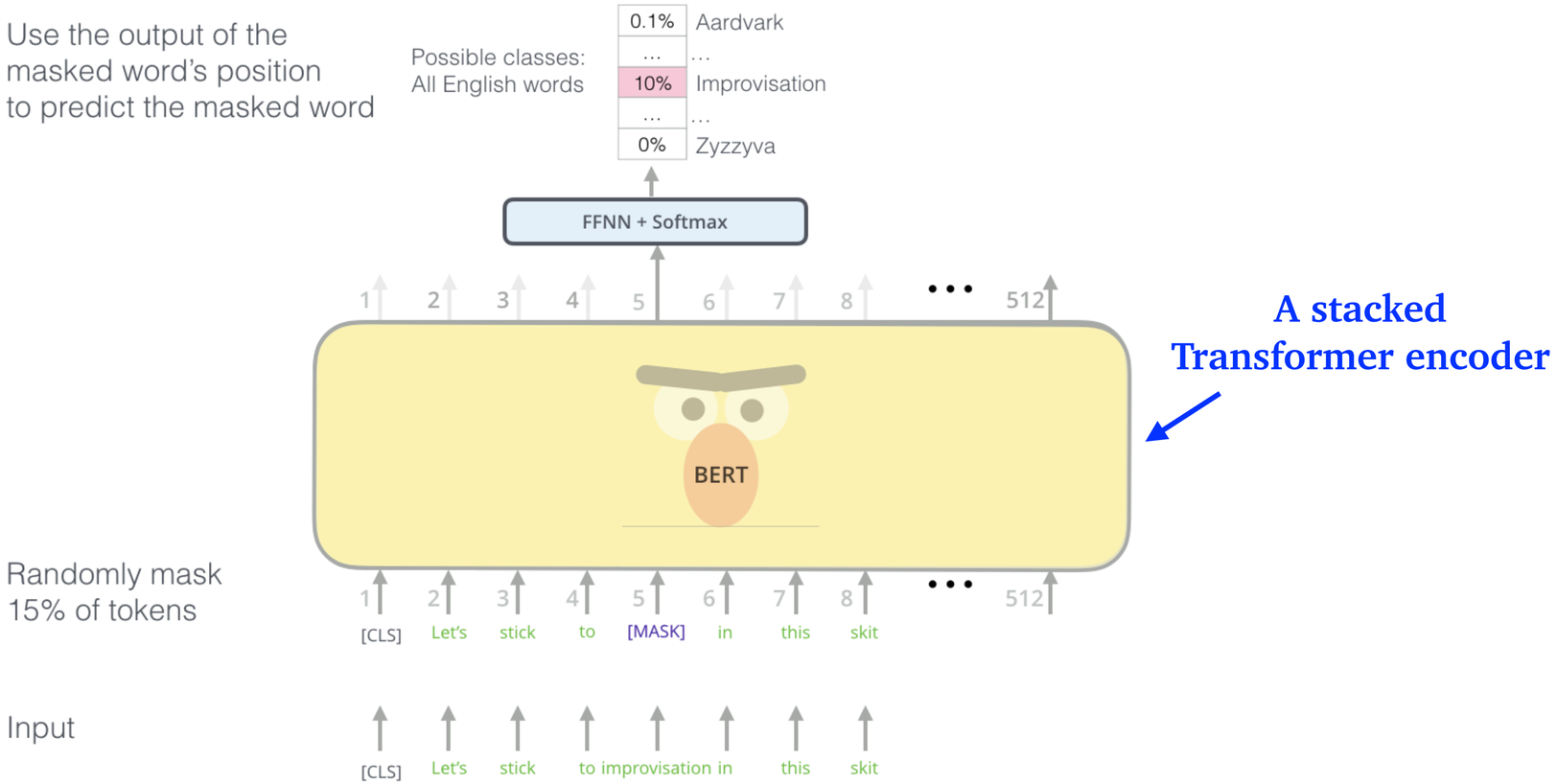
Key idea: masked language model (MLM)

- Solution: let's mask out 15% of the input words, and then predict the masked words

Q: Why 15%?

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

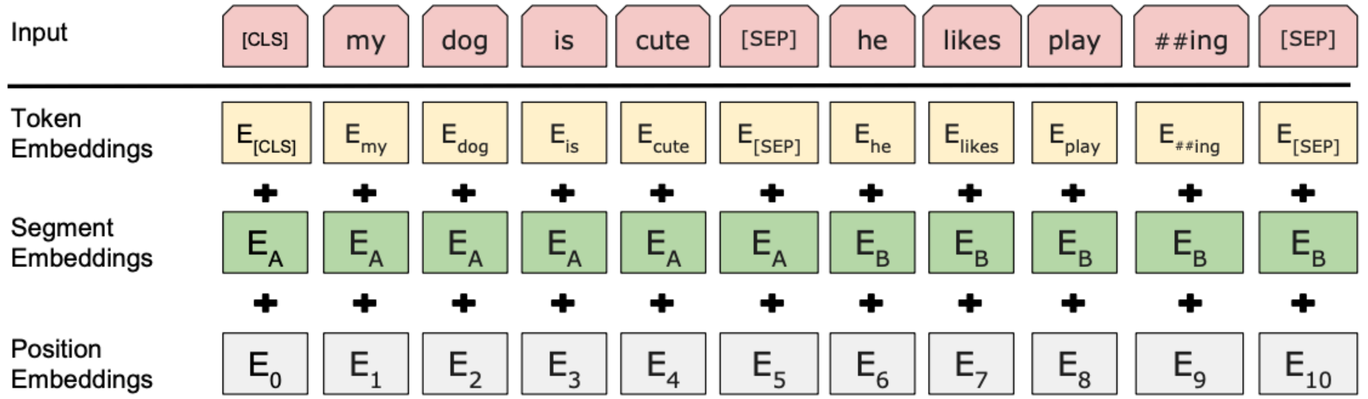
Key idea: masked language model (MLM)



Caveat: the actual sampling process is a bit more complicated than this. Check out the paper for details!

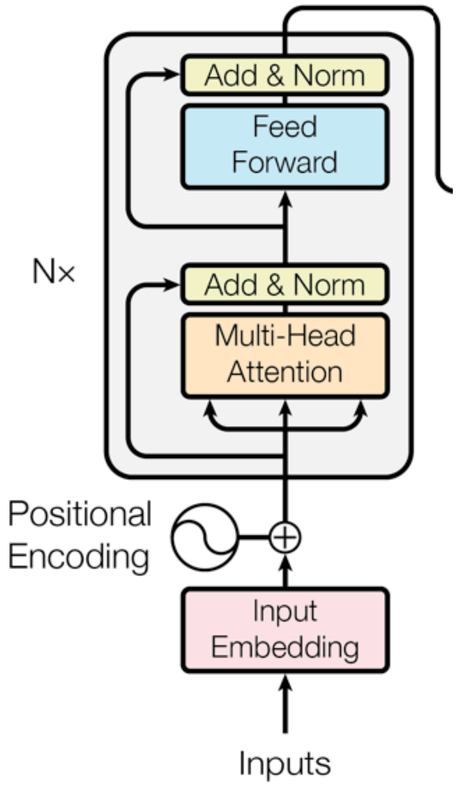
BERT: pre-training

- Input representations



- Segment length: 512 BPE (=byte pair encoding) tokens
- Trained 40 epochs on Wikipedia (2.5B tokens) + BookCorpus (0.8B tokens)
- Released two model sizes: BERT_base, BERT_large

BERT: model sizes



- **BERT$_{BASE}$**: L=12, H=768, A=12, Total Parameters=110M
- **BERT$_{LARGE}$**: L=24, H=1024, A=16, Total Parameters=340M

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

BERT: ablation studies

Tasks	Dev Set MNLI-m (Acc)	QNLI (Acc)	MRPC (Acc)	SST-2 (Acc)	SQuAD (F1)
$\text{BERT}_{\text{BASE}}$	84.4	88.4	86.7	92.7	88.5
No NSP	83.9	84.9	86.5	92.6	87.9
LTR & No NSP	82.1	84.3	77.5	92.1	77.8
+ BiLSTM	82.1	84.1	75.7	91.6	84.9

Unidirectional LMs don't work!

Table 5: Ablation over the pre-training tasks using the $\text{BERT}_{\text{BASE}}$ architecture. "No NSP" is trained without the next sentence prediction task. "LTR & No NSP" is trained as a left-to-right LM without the next sentence prediction, like OpenAI GPT. "+ BiLSTM" adds a randomly initialized BiLSTM on top of the "LTR + No NSP" model during fine-tuning.

Hyperparams				Dev Set Accuracy		
#L	#H	#A	LM (ppl)	MNLI-m	MRPC	SST-2
3	768	12	5.84	77.9	79.8	88.4
6	768	3	5.24	80.6	82.2	90.7
6	768	12	4.68	81.9	84.8	91.3
12	768	12	3.99	84.4	86.7	92.9
12	1024	16	3.54	85.7	86.9	93.3
24	1024	16	3.23	86.6	87.8	93.7

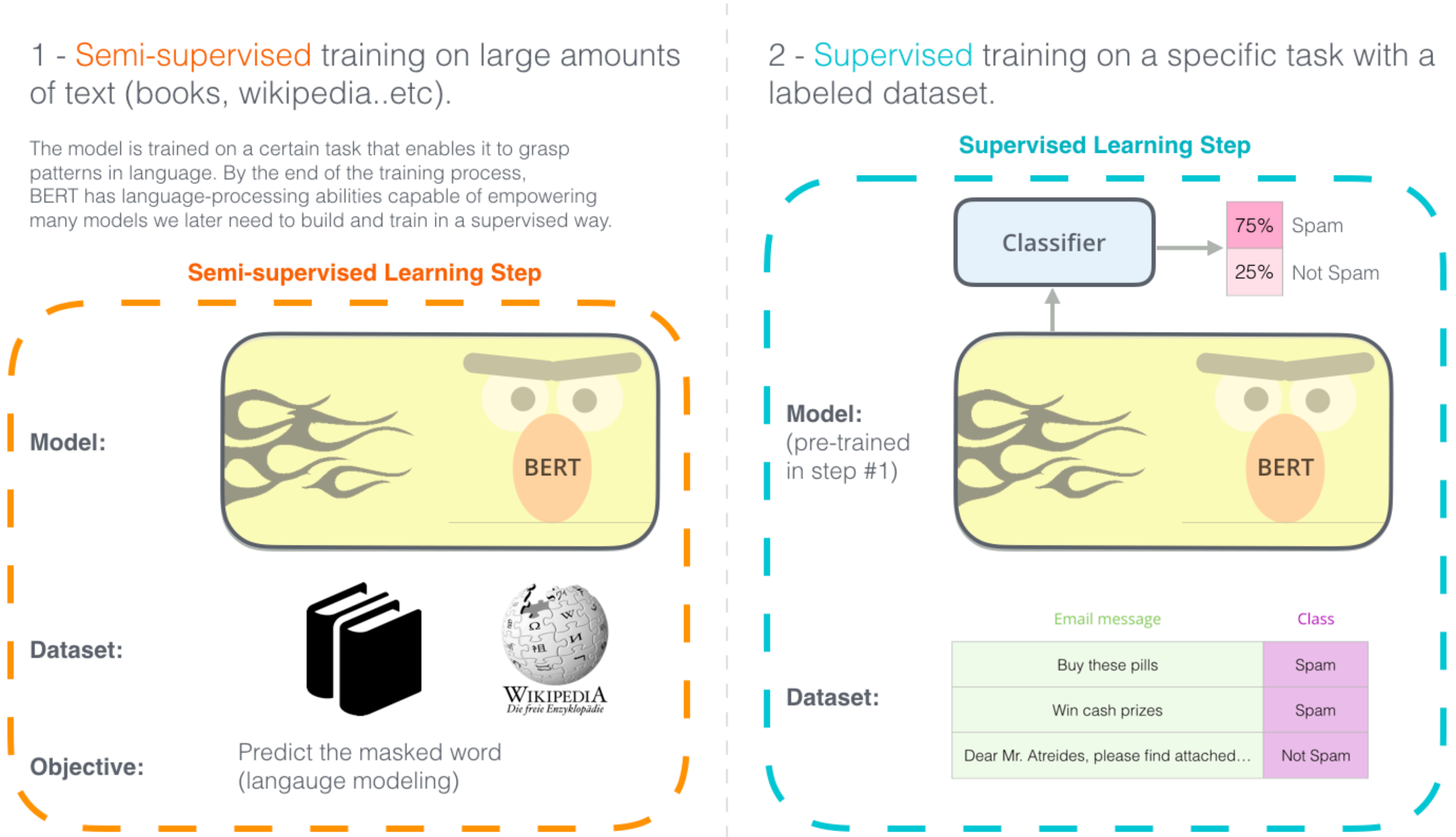
Table 6: Ablation over BERT model size. #L = the number of layers; #H = hidden size; #A = number of attention heads. "LM (ppl)" is the masked LM perplexity of held-out training data.

The bigger, the better..

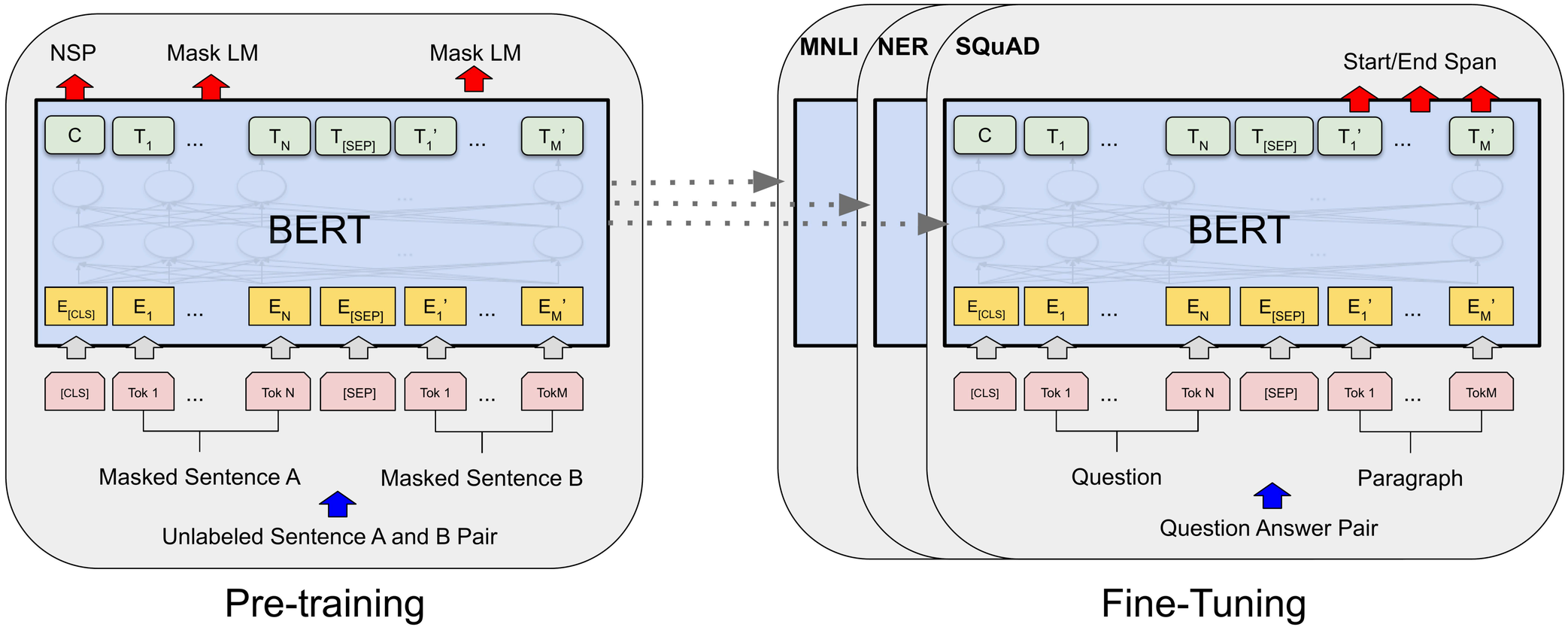
(Devlin et al., 2019) BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

How to use BERT?

Key idea: instead of use BERT to produce contextualized word embeddings, let's keep all the parameters and **fine-tune** them for downstream tasks



How to use BERT?

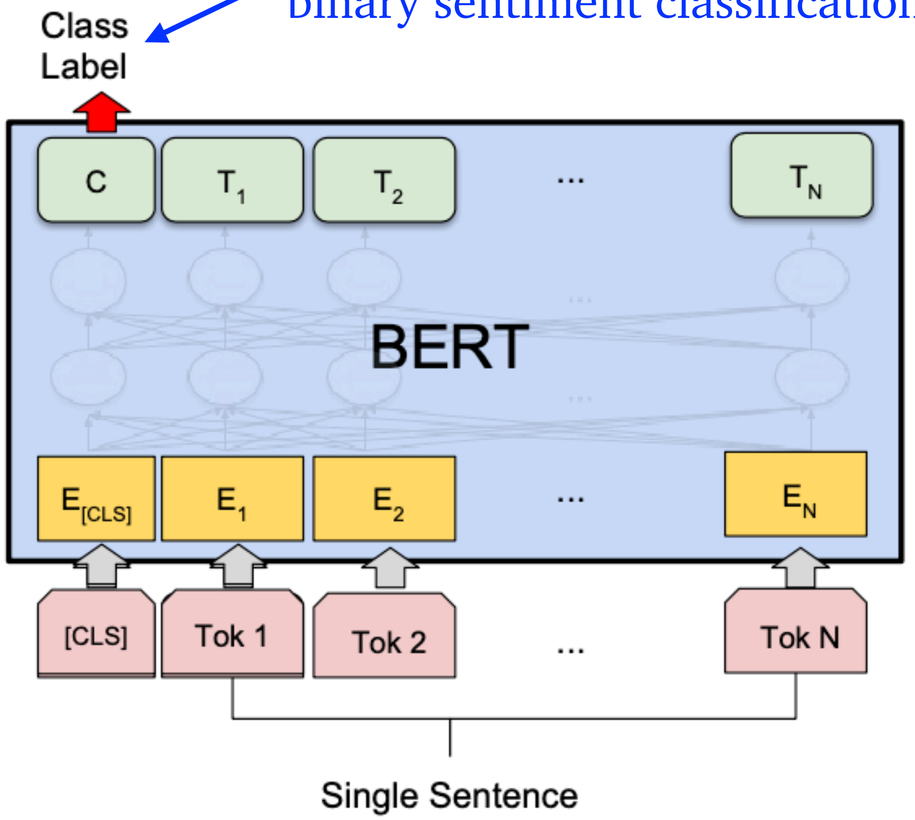


Pre-training

Fine-Tuning

Example: sentiment classification

We just need to introduce $2H$ parameters for binary sentiment classification!



All the parameters will be learned together (original BERT parameters + new classifier parameters)

BERT: experimental results

BiLSTM: 63.9

unidirectional

System	MNLI-(m/mm) 392k	QQP 363k	QNLI 108k	SST-2 67k	CoLA 8.5k	STS-B 5.7k	MRPC 3.5k	RTE 2.5k	**Average** -
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

Use BERT models

https://github.com/huggingface/transformers

Transformers

State-of-the-art Natural Language Processing for PyTorch and TensorFlow 2.0

```
export TASK_NAME=mrpc

python run_glue.py \
  --model_name_or_path bert-base-cased \
  --task_name $TASK_NAME \
  --do_train \
  --do_eval \
  --max_seq_length 128 \
  --per_device_train_batch_size 32 \
  --learning_rate 2e-5 \
  --num_train_epochs 3 \
  --output_dir /tmp/$TASK_NAME/
```

ELMo vs BERT

Which of the following statements is INCORRECT?

(a) BERT was trained on more data than ELMo

(b) BERT is better at capturing at bidirectional contexts than ELMo

(c) BERT models only work well with deep encoder layers

(d) BERT is better at eliminating the needs of heavily-engineered task-specific architectures

(c) is correct.

	H=128	H=256	H=512	H=768
L=2	**2/128 (BERT-Tiny)**	2/256	2/512	2/768
L=4	4/128	**4/256 (BERT-Mini)**	**4/512 (BERT-Small)**	4/768
L=6	6/128	6/256	6/512	6/768
L=8	8/128	8/256	**8/512 (BERT-Medium)**	8/768
L=10	10/128	10/256	10/512	10/768
L=12	12/128	12/256	12/512	**12/768 (BERT-Base)**

Model	Score	CoLA	SST-2	MRPC	STS-B	QQP	MNLI-m	MNLI-mm	QNLI(v2)	R
BERT-Tiny	64.2	0.0	83.2	81.1/71.1	74.3/73.6	62.2/83.4	70.2	70.3	81.5	57
BERT-Mini	65.8	0.0	85.9	81.1/71.8	75.4/73.3	66.4/86.2	74.8	74.3	84.1	57
BERT-Small	71.2	27.8	89.7	83.4/76.2	78.8/77.0	68.1/87.0	77.6	77.0	86.4	61
BERT-Medium	73.5	38.0	89.6	86.6/81.6	80.4/78.4	69.6/87.9	80.0	79.1	87.7	62

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