Natural Language Processing with Deep Learning
CS224N/Ling284

Christopher Manning

Lecture 13: Contextual Word Representations and Pretraining

Lecture Plan

Lecture 13: Contextual Word Representations and Pretraining

1. Reflections on word representations (10 mins)
2. Pre-ELMo and ELMO (20 mins)
3. ULMfit and onward (10 mins)
4. Transformer architectures (20 mins)
5. BERT (20 mins)

1. Representations for a word

- Up until now, we’ve basically said that we have one representation of words:
  - The word vectors that we learned about at the beginning
    - Word2vec, GloVe, fastText

Pre-trained word vectors: The early years
Collobert, Weston, et al. 2011 results

	POS WSJ (acc.)	NER CoNLL (F1)
State-of-the-art*	97.24	89.31
Supervised NN	**96.37**	**81.47**
Unsupervised pre-training followed by supervised NN**	**97.20**	**88.87**
+ hand-crafted features***	97.29	89.59

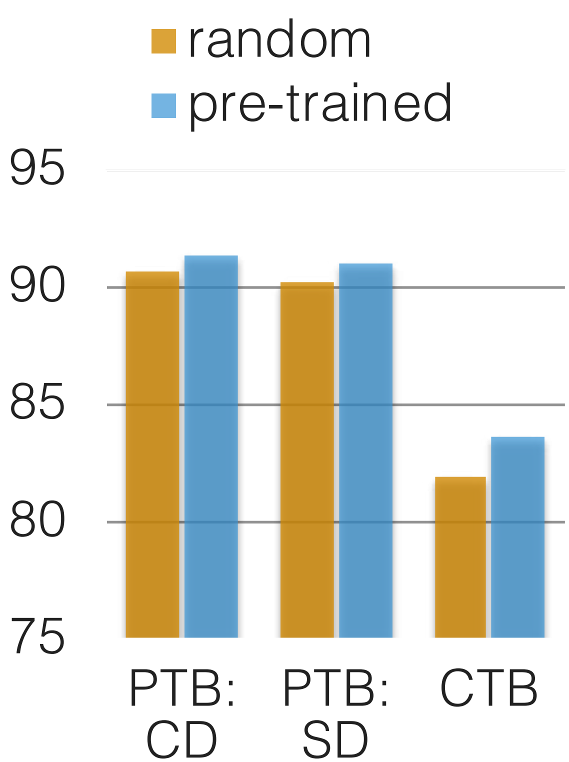
* Representative systems: POS: (Toutanova et al. 2003), NER: (Ando & Zhang 2005)

** 130,000-word embedding trained on Wikipedia and Reuters with 11 word window, 100 unit hidden layer – **for 7 weeks!** – then supervised task training

*** Features are character suffixes for POS and a gazetteer for NER

Pre-trained word vectors: Current sense (2014–)

- We can just start with random word vectors and train them on our task of interest
- But in most cases, use of pre-trained word vectors helps, because we can train them for **more words** on **much more data**



- Chen and Manning (2014) Dependency parsing
- Random: uniform(-0.01, 0.01)
- Pre-trained:
  - PTB (C & W): +0.7%
  - CTB (word2vec): +1.7%

Tips for unknown words with word vectors

- Simplest and common solution:
- Train time: Vocab is {words occurring, say, ≥ 5 times} ∪ {<UNK>}
- Map **all** rarer (< 5) words to <UNK>, train a word vector for it
- Runtime: use <UNK> when out-of-vocabulary (OOV) words occur

- Problems:
  - No way to distinguish different UNK words, either for identity or meaning

- Solutions:
  1. Hey, we just learned about char-level models to build vectors! Let's do that!

Tips for unknown words with word vectors

- Especially in applications like question answering
  - Where it is important to match on word identity, even for words outside your word vector vocabulary
- 2. Try these tips (from Dhingra, Liu, Salakhutdinov, Cohen 2017)
  a. If the <UNK> word at test time appears in your unsupervised word embeddings, use that vector as is at test time.
  b. Additionally, for other words, just assign them a random vector, adding them to your vocabulary
- a. definitely helps a lot; b. may help a little more
- Another thing you can try:
  - Collapsing things to word classes (like unknown number, capitalized thing, etc. and having an <UNK-class> for each

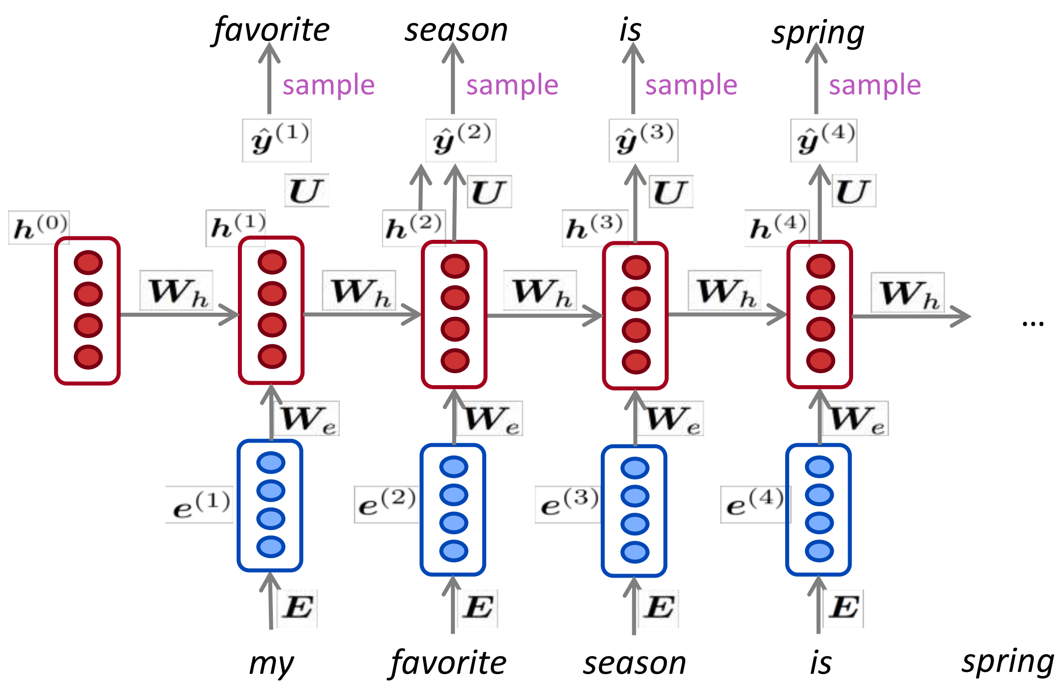
Representations for a word

- Up until now, we've basically had one representation of words:
  - The word vectors that we learned about at the beginning
    - Word2vec, GloVe, fastText

- These have two problems:
  - Always the same representation for a **word type** regardless of the context in which a **word token** occurs
    - We might want very fine-grained word sense disambiguation
  - We just have one representation for a word, but words have different **aspects**, including semantics, syntactic behavior, and register/connotations

Did we all along have a solution to this problem?

- In, an NLM, we immediately stuck word vectors (perhaps only trained on the corpus) through LSTM layers
- Those LSTM layers are trained to predict the next word
- But those language models are producing context-specific word representations at each position!

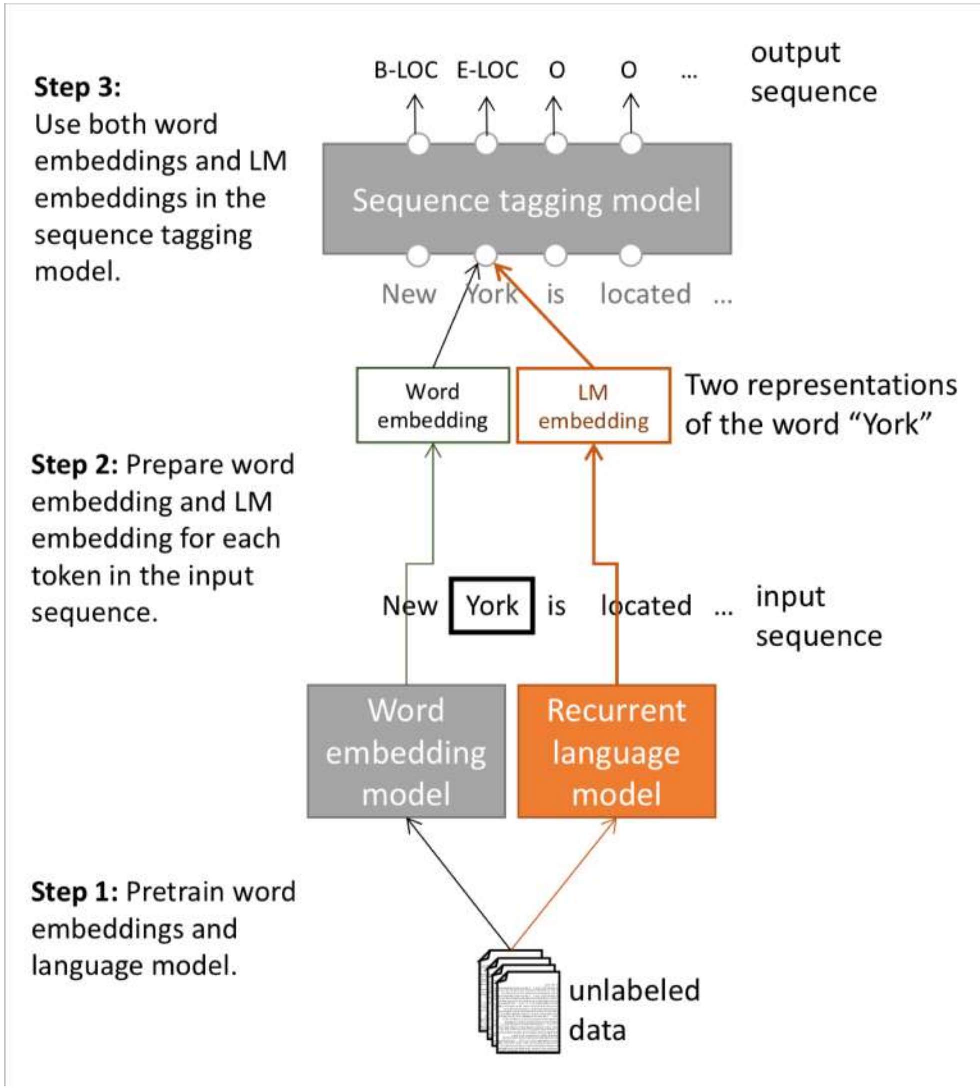


2. Peters et al. (2017): TagLM – “Pre-ELMo”

https://arxiv.org/pdf/1705.00108.pdf

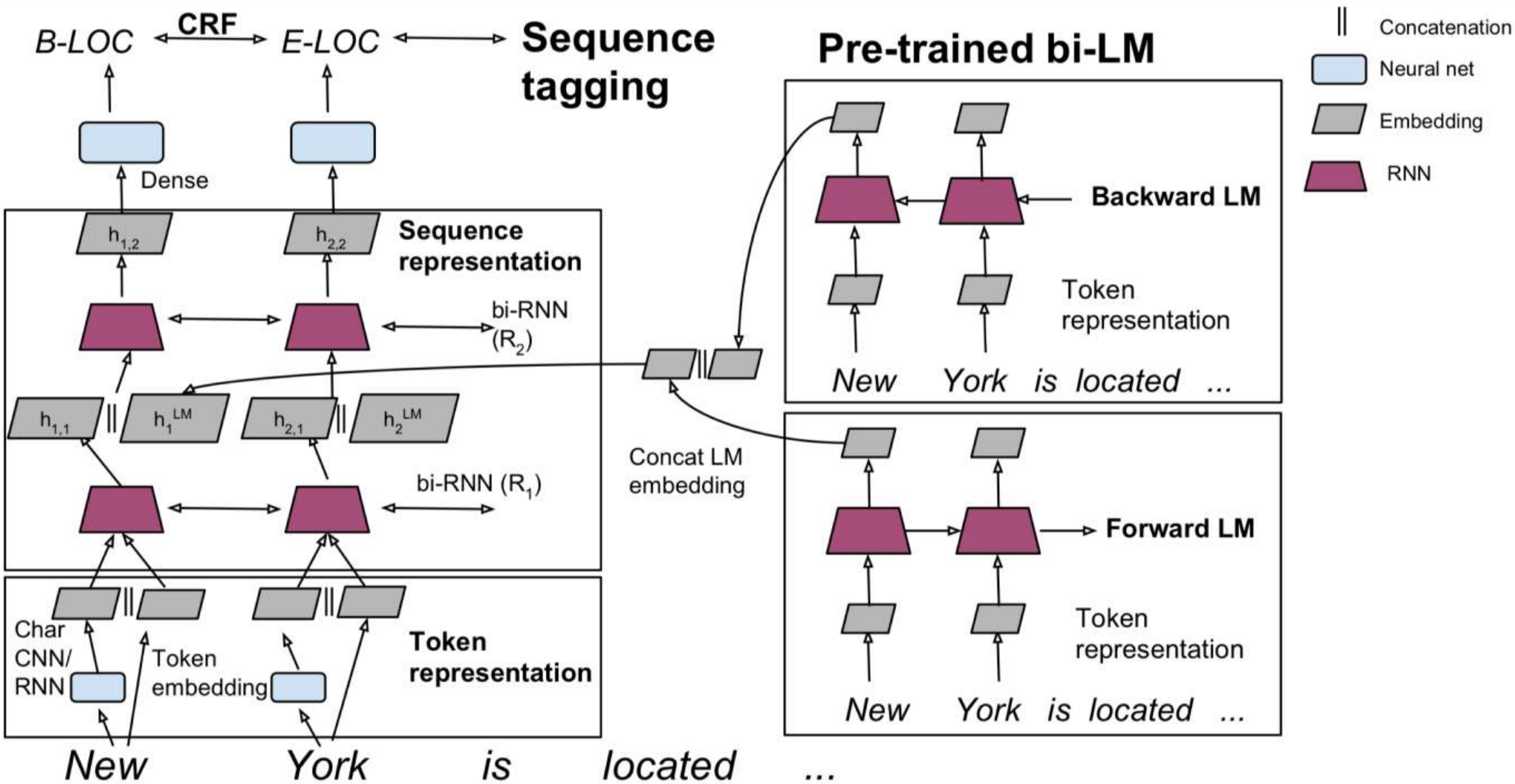
- Idea: Want meaning of word in context, but standardly learn task RNN only on small task-labeled data (e.g., NER)
- Why don’t we do semi-supervised approach where we train NLM on large unlabeled corpus, rather than just word vectors?

Tag LM



Step 3:
Use both word embeddings and LM embeddings in the sequence tagging model.
B-LOC E-LOC O O ...
output sequence
Sequence tagging model
New York is located ...
Word embedding
LM embedding
Two representations of the word "York"
Step 2: Prepare word embedding and LM embedding for each token in the input sequence.
New York is located ...
input sequence
Word embedding model
Recurrent language model
Step 1: Pretrain word embeddings and language model.
unlabeled data

Tag LM



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

Named Entity Recognition (NER)

- A very important NLP sub-task: find and classify names in text, for example:

  - The decision by the independent MP Andrew Wilkie to withdraw his support for the minority Labor government sounded dramatic but it should not further threaten its stability. When, after the 2010 election, Wilkie, Rob Oakeshott, Tony Windsor and the Greens agreed to support Labor, they gave just two guarantees: confidence and supply.

Person
Date
Location
Organi-
zation

CoNLL 2003 Named Entity Recognition (en news testb)

Name	Description	Year	F1
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 Klein	MEMM softmax markov model	2003	86.07

Peters et al. (2017): TagLM – "Pre-ELMo"

Language model is trained on 800 million training words of "Billion word benchmark"

Language model observations

- An LM trained on supervised data does not help
- Having a bidirectional LM helps over only forward, by about 0.2
- Having a huge LM design (ppl 30) helps over a smaller model (ppl 48) by about 0.3

Task-specific BiLSTM observations

- Using just the LM embeddings to predict isn't great: 88.17 F1
    - Well below just using an BiLSTM tagger on labeled data

Also in the air: McCann et al. 2017: CoVe

https://arxiv.org/pdf/1708.00107.pdf

- Also has idea of using a trained sequence model to provide context to other NLP models
- Idea: Machine translation is meant to preserve meaning, so maybe that's a good objective?
- Use a 2-layer bi-LSTM that is the encoder of seq2seq + attention NMT system as the context provider
- The resulting CoVe vectors do outperform GloVe vectors on various tasks
- But, the results aren't as strong as the simpler NLM training described in the rest of these slides so seems abandoned
  - Maybe NMT is just harder than language modeling?
  - Maybe someday this idea will return?

Peters et al. (2018): ELMo: Embeddings from Language Models

Deep contextualized word representations. NAACL 2018.
https://arxiv.org/abs/1802.05365

- Breakout version of **word token vectors** or **contextual word vectors**
- Learn word token vectors using long contexts not context windows (here, whole sentence, could be longer)
- Learn a deep Bi-NLM and use all its layers in prediction

Peters et al. (2018): 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

Peters et al. (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

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

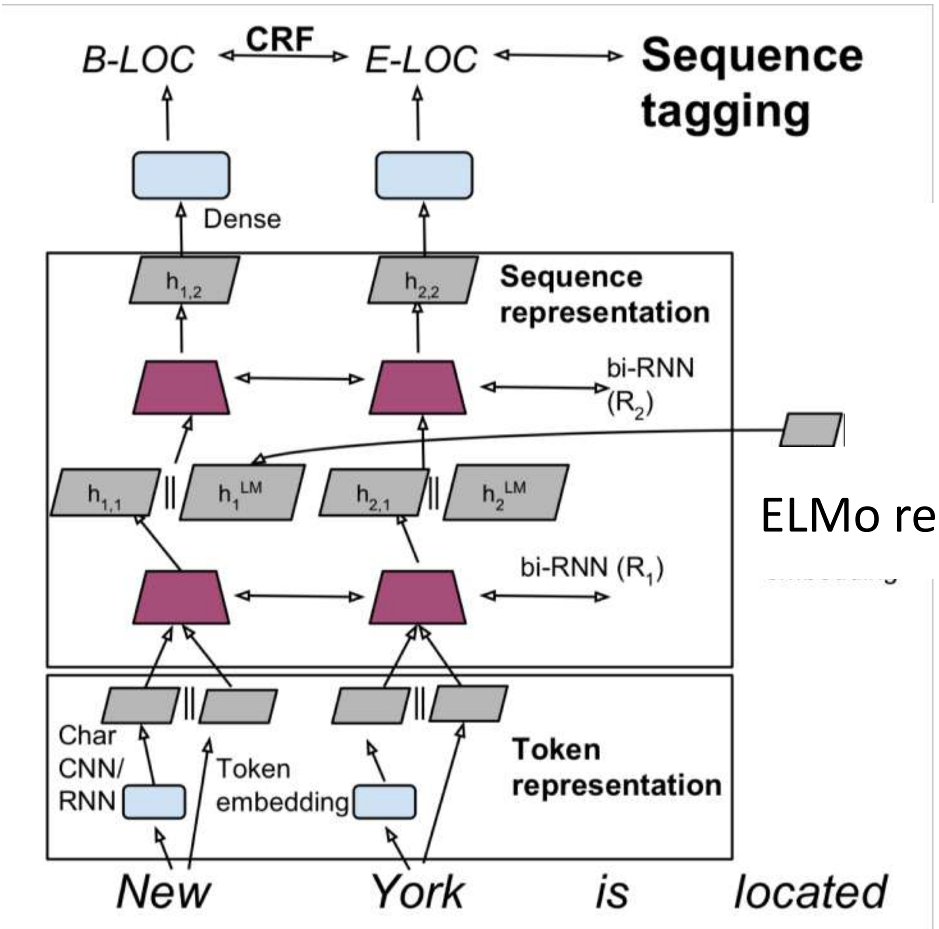
$$
\mathbf{ELMo}_k^{task} = E(R_k; \Theta^{task}) = \gamma^{task} \sum_{j=0}^{L} s_j^{task} \mathbf{h}_{k,j}^{LM}
$$

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

Peters et al. (2018): ELMo: Use with a task

- First run biLM to get representations for each word
- Then let (whatever) end-task model use them
  - Freeze weights of ELMo for purposes of supervised model
  - Concatenate ELMo weights into task-specific model
    - Details depend on task
      - Concatenating into intermediate layer as for TagLM is typical
      - Can provide ELMo representations again when producing outputs, as in a question answering system

ELMo used in a sequence tagger



ELMo representation

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

CoNLL 2003 Named Entity Recognition (en news testb)

Name	Description	Year	F1
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

ELMo results: Great for all tasks

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: Weighting of layers

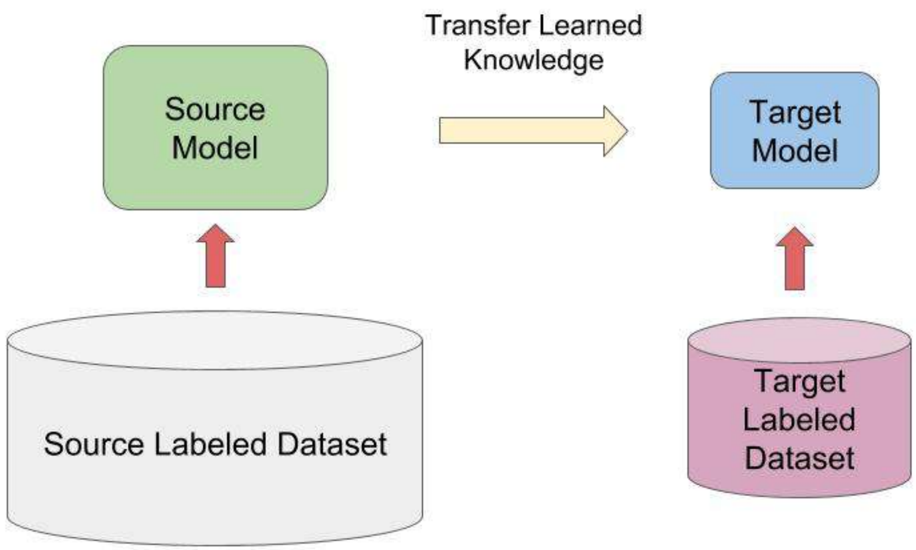
- The two biLSTM NLM layers have differentiated uses/meanings
  - Lower layer is better for lower-level syntax, etc.
    - Part-of-speech tagging, syntactic dependencies, NER
  - Higher layer is better for higher-level semantics
    - Sentiment, Semantic role labeling, question answering, SNLI

- This seems interesting, but it'd seem more interesting to see how it pans out with more than two layers of network

Also around: ULMfit

Howard and Ruder (2018) Universal Language Model Fine-tuning for Text Classification. https://arxiv.org/pdf/1801.06146.pdf

- Same general idea of transferring NLM knowledge
- Here applied to text classification

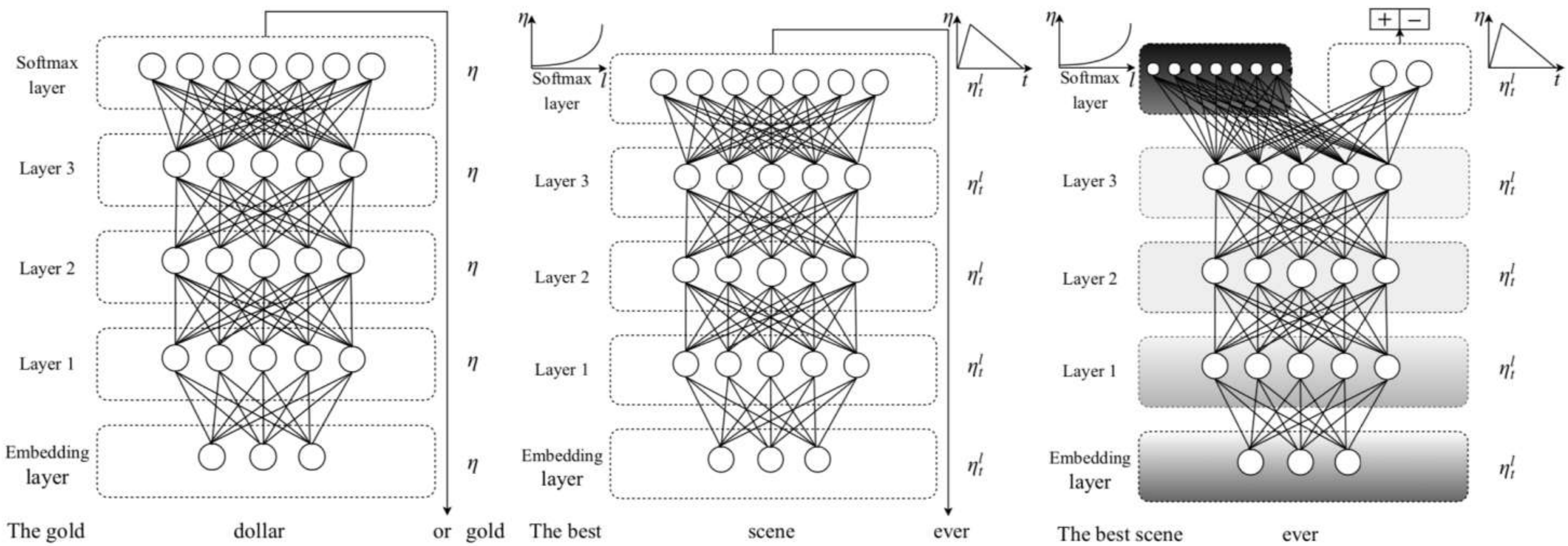


ULMfit

Train LM on big general domain corpus (use biLM)

Tune LM on target task data

Fine-tune as classifier on target task



(a) LM pre-training (b) LM fine-tuning (c) Classifier fine-tuning

ULMfit emphases

Use reasonable-size “1 GPU” language model not really huge one

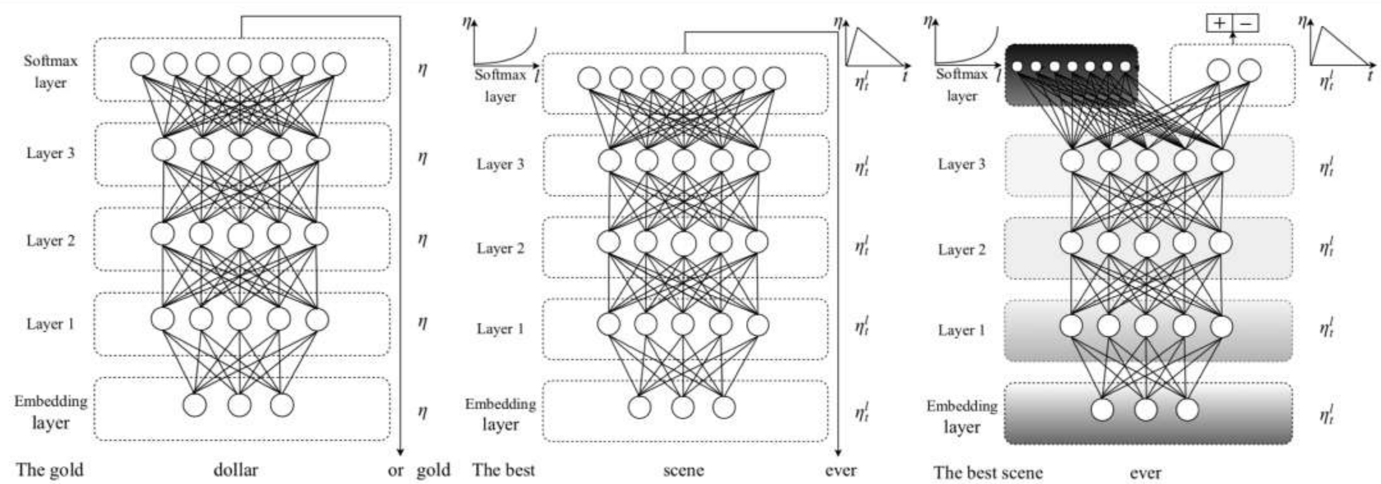
A lot of care in LM fine-tuning

Different per-layer learning rates

Slanted triangular learning rate (STLR) schedule

Gradual layer unfreezing and STLR when learning classifier

Classify using concatenation $[h_T, \mathrm{maxpool}(\mathbf{h}), \mathrm{meanpool}(\mathbf{h})]$



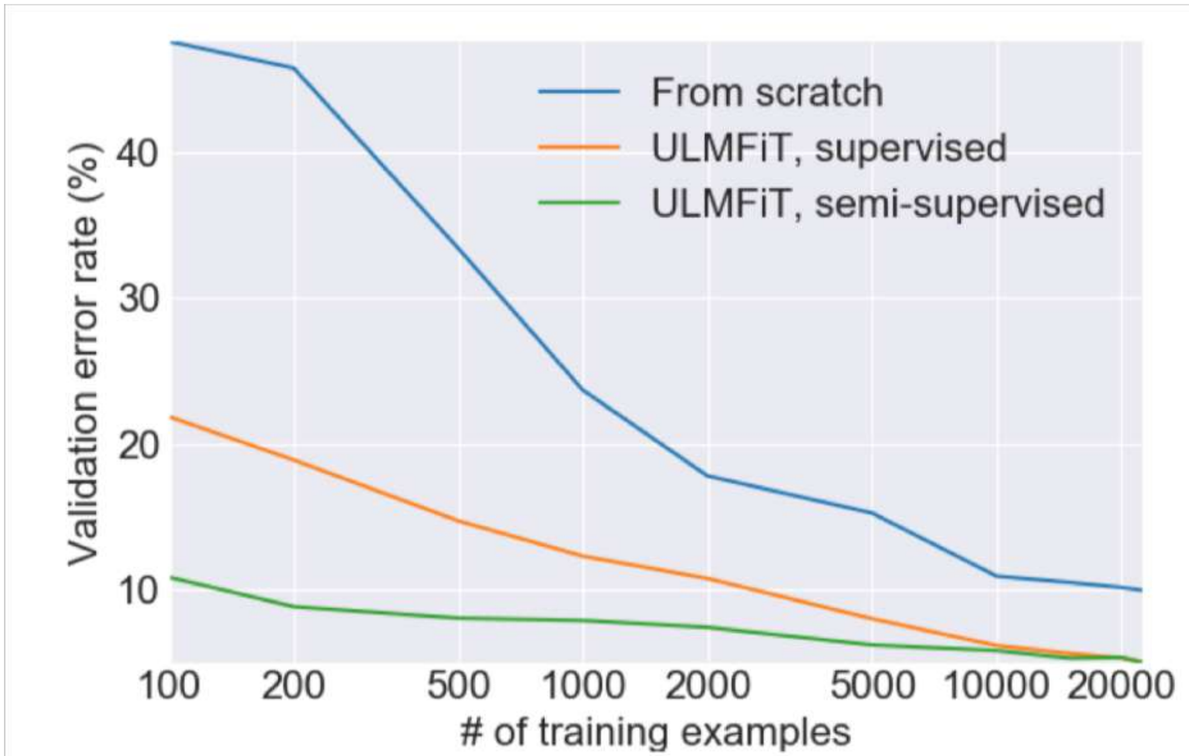
(a) LM pre-training (b) LM fine-tuning (c) Classifier fine-tuning

ULMfit performance

- Text classifier error rates

	Model	Test		Model	Test
IMDb	CoVe (McCann et al., 2017)	8.2	TREC-6	CoVe (McCann et al., 2017)	4.2
	oh-LSTM (Johnson and Zhang, 2016)	5.9		TBCNN (Mou et al., 2015)	4.0
	Virtual (Miyato et al., 2016)	5.9		LSTM-CNN (Zhou et al., 2016)	3.9
	ULMFiT (ours)	**4.6**		ULMFiT (ours)	**3.6**

ULMfit transfer learning



Let's scale it up!

ULMfit

Jan 2018

Training:

1 GPU day

GPT

June 2018

Training

240 GPU days

BERT

Oct 2018

Training

256 TPU days

~320–560 GPU days

GPT-2

Feb 2019

Training

~2048 TPU v3 days according to a reddit thread

fast.ai

OpenAI

Google AI

OpenAI

GPT-2 language model (cherry-picked) output

SYSTEM PROMPT (HUMAN-WRITTEN)	*In a shocking finding, scientist discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English.*
MODEL COMPLETION (MACHINE-WRITTEN, 10 TRIES)	The scientist named the population, after their distinctive horn, Ovid's Unicorn. These four-horned, silver-white unicorns were previously unknown to science. Now, after almost two centuries, the mystery of what sparked this odd phenomenon is finally solved. Dr. Jorge Pérez, an evolutionary biologist from the University of La Paz, and several companions, were exploring the Andes Mountains when they found a small valley, with no other animals or humans. Pérez noticed that the valley had what appeared to be a natural fountain, surrounded by two peaks of rock and silver snow. Pérez and the others then ventured further into the valley. …

METRO NEWS... BUT NOT AS YOU KNOW IT

NEWS SPORT ENTERTAINMENT SOAPS MORE TRENDING

UK WORLD WEIRD TECH

Elon Musk's OpenAI builds artificial intelligence so powerful it must be kept locked up for the good of humanity

Jasper Hamill Friday 15 Feb 2019 10:06 am

272 SHARES

Elon Musk's scientists have announced the creation of a terrifying artificial intelligence that's so smart they refused to release it to the public.

OpenAI's GPT-2 is designed to write just like a human and is an impressive leap forward capable of penning chillingly convincing text.

It was 'trained' by analysing eight million web pages and is capable of writing large tracts based upon a 'prompt' written by a real person.

But the machine mind will not be released in its fully-fledged form because of the risk of it being used for 'malicious purposes' such as generating fake news, impersonating people online, automating the production of spam or churning out 'abusive or faked content to post on social media'.

OpenAI wrote: 'Due to our concerns about malicious applications of the technology, we are not releasing the trained model.

Elon Musk @elonmusk Follow

Replying to @georgezachary

To clarify, I've not been involved closely with OpenAI for over a year & don't have mgmt or board oversight

8:19 PM - 16 Feb 2019

500 Retweets 14,573 Likes

229 500 15K

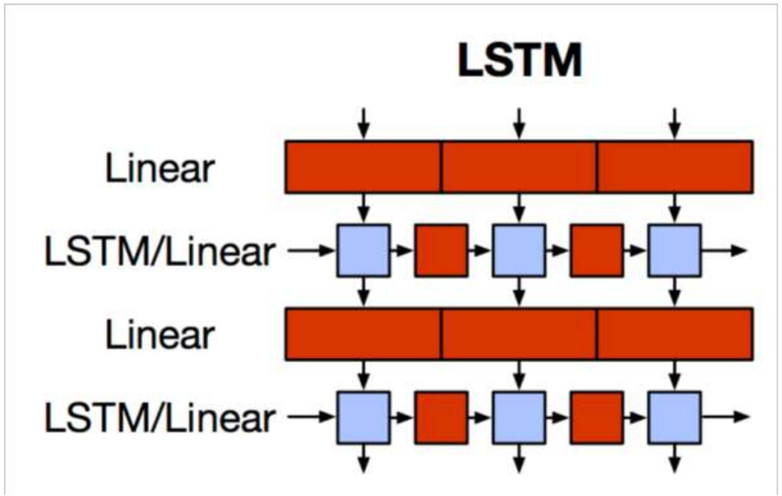
Transformer models

All of these models are Transformer architecture models … so maybe we had better learn about Transformers?

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

4. The Motivation for Transformers

- We want **parallelization** but RNNs are inherently sequential

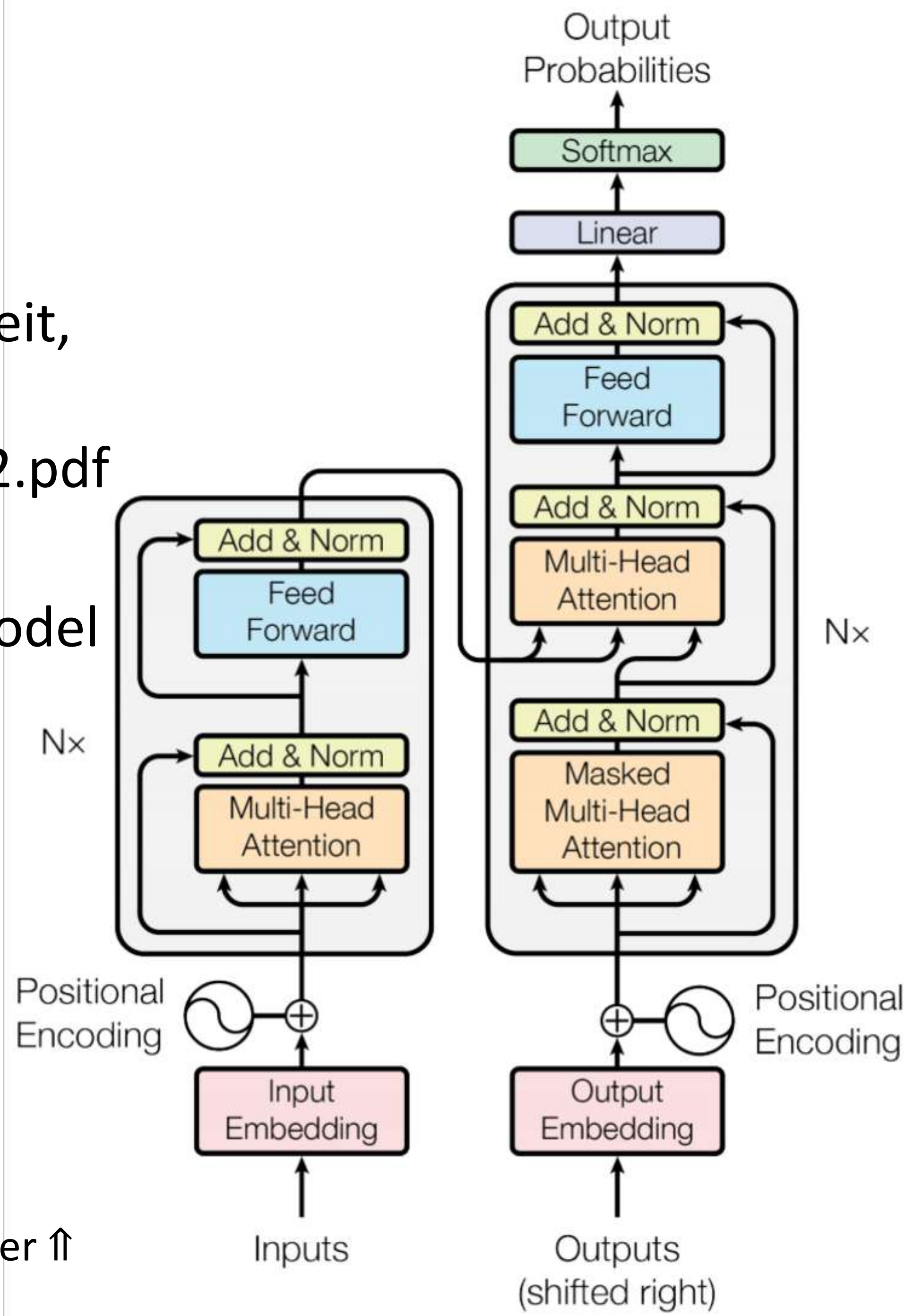


- Despite GRUs and LSTMs, RNNs still need attention mechanism to deal with long range dependencies – **path length** between states grows with sequence otherwise
- But if **attention** gives us access to any state… maybe we can just use attention and don't need the RNN?

Transformer Overview

Attention is all you need. 2017.
Aswani, Shazeer, Parmar, Uszkoreit, Jones, Gomez, Kaiser, Polosukhin
https://arxiv.org/pdf/1706.03762.pdf

- Non-recurrent sequence-to-sequence encoder-decoder model
- Task: machine translation with parallel corpus
- Predict each translated word
- Final cost/error function is standard cross-entropy error on top of a softmax classifier



This and related figures from paper ⇑

Transformer Basics

- Learning about transformers on your own?
  - Key recommended resource:
    - http://nlp.seas.harvard.edu/2018/04/03/attention.html
    - The Annotated Transformer by Sasha Rush
  - An Jupyter Notebook using PyTorch that explains everything!

- For now: Let’s define the basic building blocks of transformer networks: first, new attention layers!

Dot-Product Attention (Extending our previous def.)

- Inputs: a query q and a set of key-value (k-v) pairs to an output
- Query, keys, values, and output are all vectors

- Output is weighted sum of values, where
- Weight of each value is computed by an inner product of query and corresponding key
- Queries and keys have same dimensionality d_k value have d_v

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

Dot-Product Attention – Matrix notation

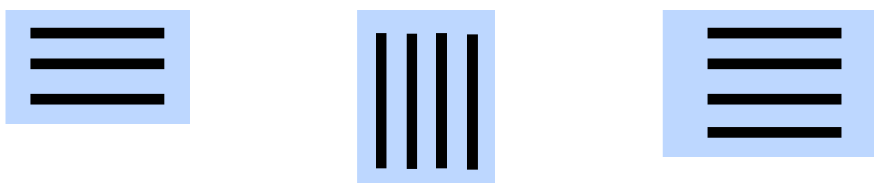
- When we have multiple queries q, we stack them in a matrix Q:

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

- Becomes: $A(Q, K, V) = softmax(QK^T)V$

$[|Q| \text{ x } d_k] \text{ x } [d_k \text{ x } |K|] \text{ x } [|K| \text{ x } d_v]$

softmax
row-wise



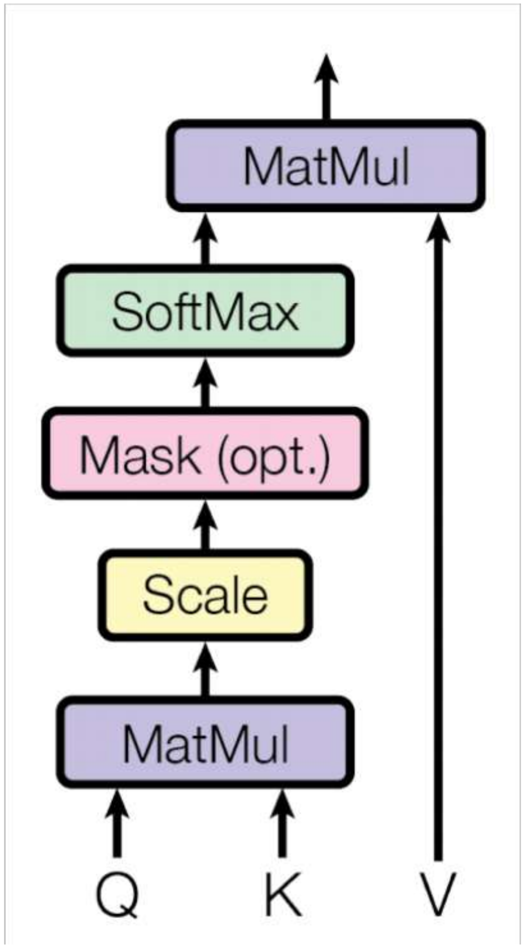
$= [|Q| \text{ x } d_v]$

Scaled Dot-Product Attention

- Problem: As d_k gets large, the variance of $q^T k$ increases → some values inside the softmax get large → the softmax gets very peaked → hence its gradient gets smaller.

- Solution: Scale by length of query/key vectors:

$$A(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$$



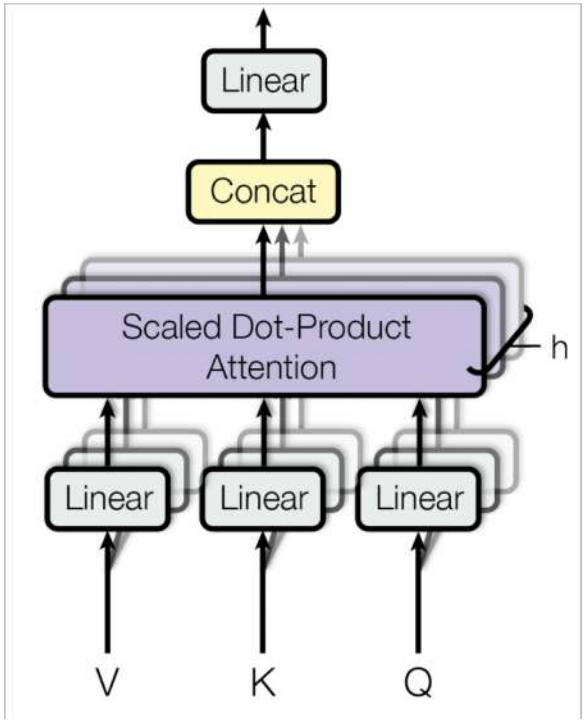
Self-attention in the encoder

- The input word vectors are the queries, keys and values
- In other words: the word vectors themselves select each other
- Word vector stack = Q = K = V
- We'll see in the decoder why we separate them in the definition

Multi-head attention

- Problem with simple self-attention:
- Only one way for words to interact with one-another
- Solution: Multi-head attention
- First map Q, K, V into h=8 many lower dimensional spaces via W matrices
- Then apply attention, then concatenate outputs and pipe through linear layer

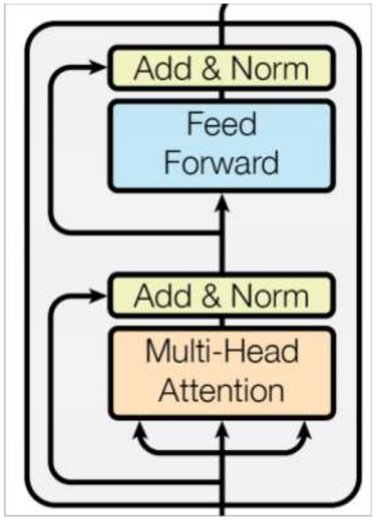
$$\mathrm{MultiHead}(Q, K, V) = \mathrm{Concat}(\mathrm{head}_1, ..., \mathrm{head}_\mathrm{h})W^O$$
$$\text{where } \mathrm{head}_\mathrm{i} = \mathrm{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$



Complete transformer block

- Each block has two "sublayers"

1. Multihead attention
2. 2-layer feed-forward NNet (with ReLU)



Each of these two steps also has:

Residual (short-circuit) connection and LayerNorm

LayerNorm(x + Sublayer(x))

Layernorm changes input to have mean 0 and variance 1,
per layer and per training point (and adds two more parameters)

$$\mu^l = \frac{1}{H}\sum_{i=1}^{H} a_i^l \qquad \sigma^l = \sqrt{\frac{1}{H}\sum_{i=1}^{H}\left(a_i^l - \mu^l\right)^2}$$

$$h_i = f\left(\frac{g_i}{\sigma_i}\left(a_i - \mu_i\right) + b_i\right)$$

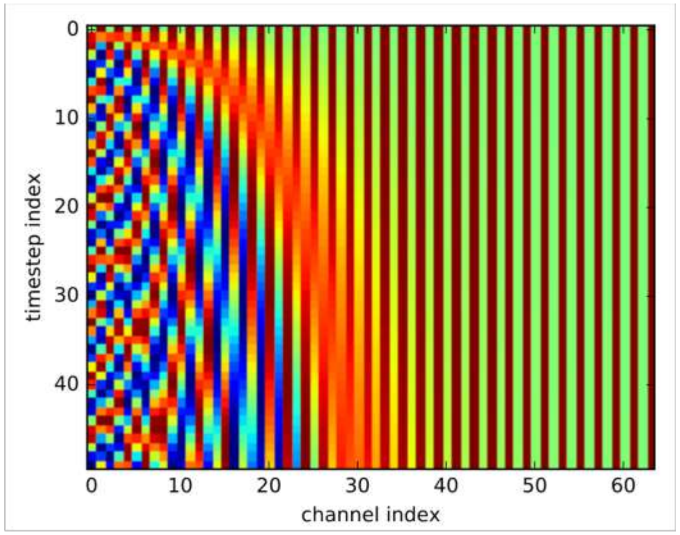
Layer Normalization by Ba, Kiros and Hinton, https://arxiv.org/pdf/1607.06450.pdf

Encoder Input

- Actual word representations are byte-pair encodings
  - As in last lecture

- Also added is a **positional encoding** so same words at different locations have different overall representations:

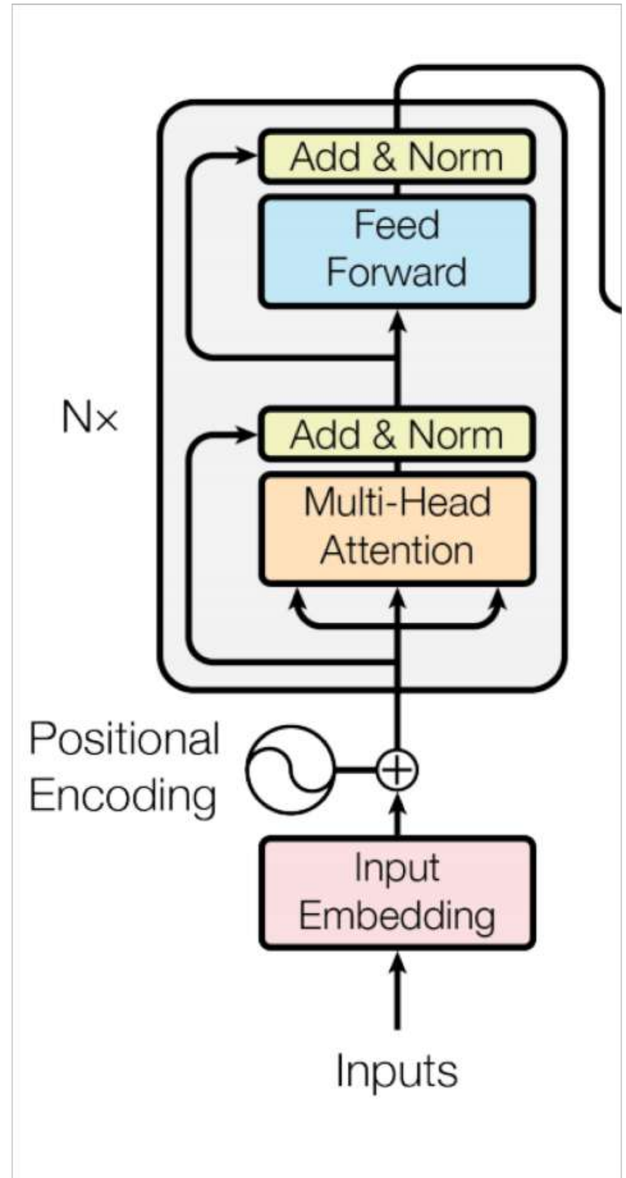
$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$



Complete Encoder

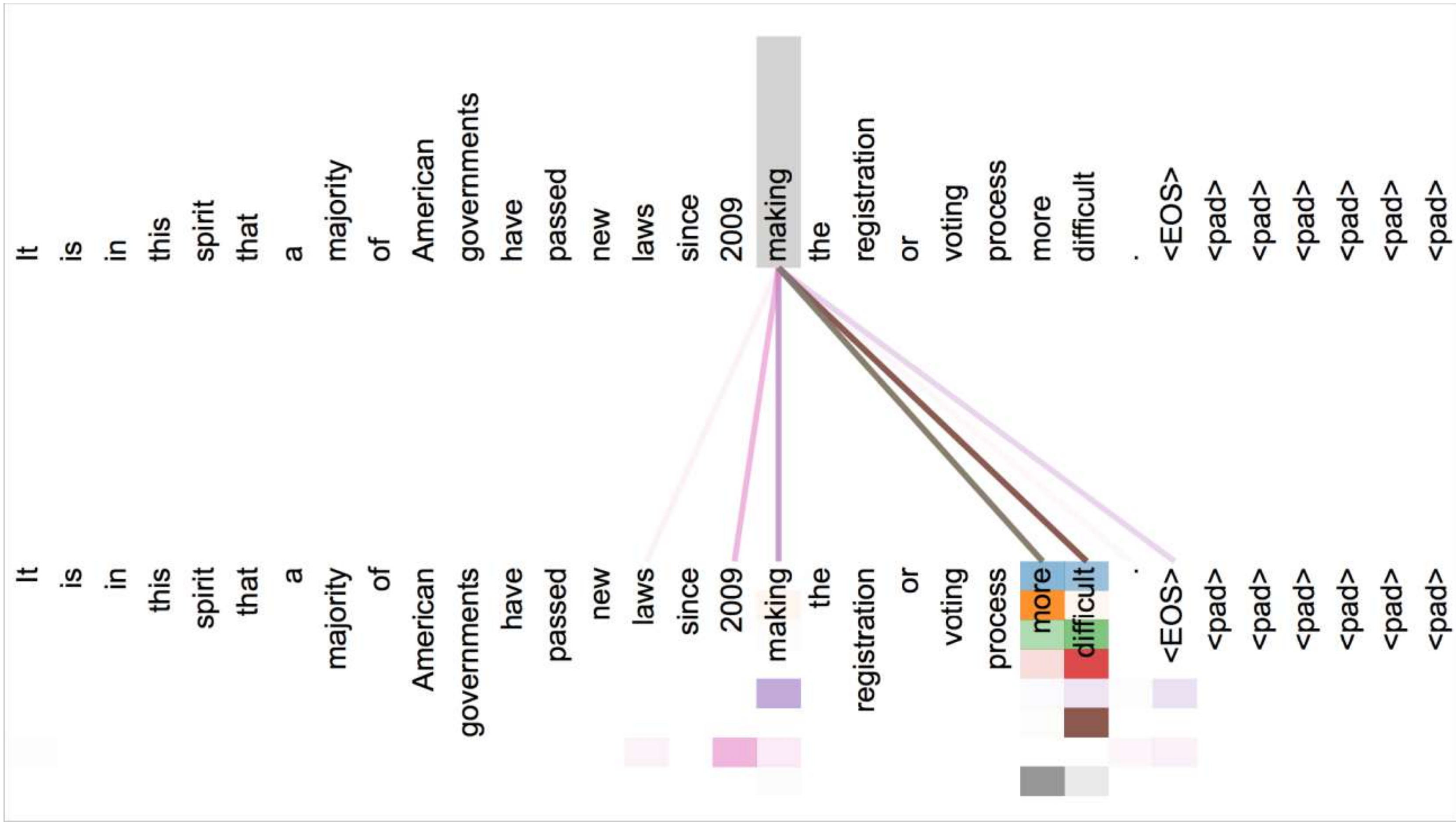
- For encoder, at each block, we use the same Q, K and V
  from the previous layer

- Blocks are repeated 6 times
  - (in vertical stack)

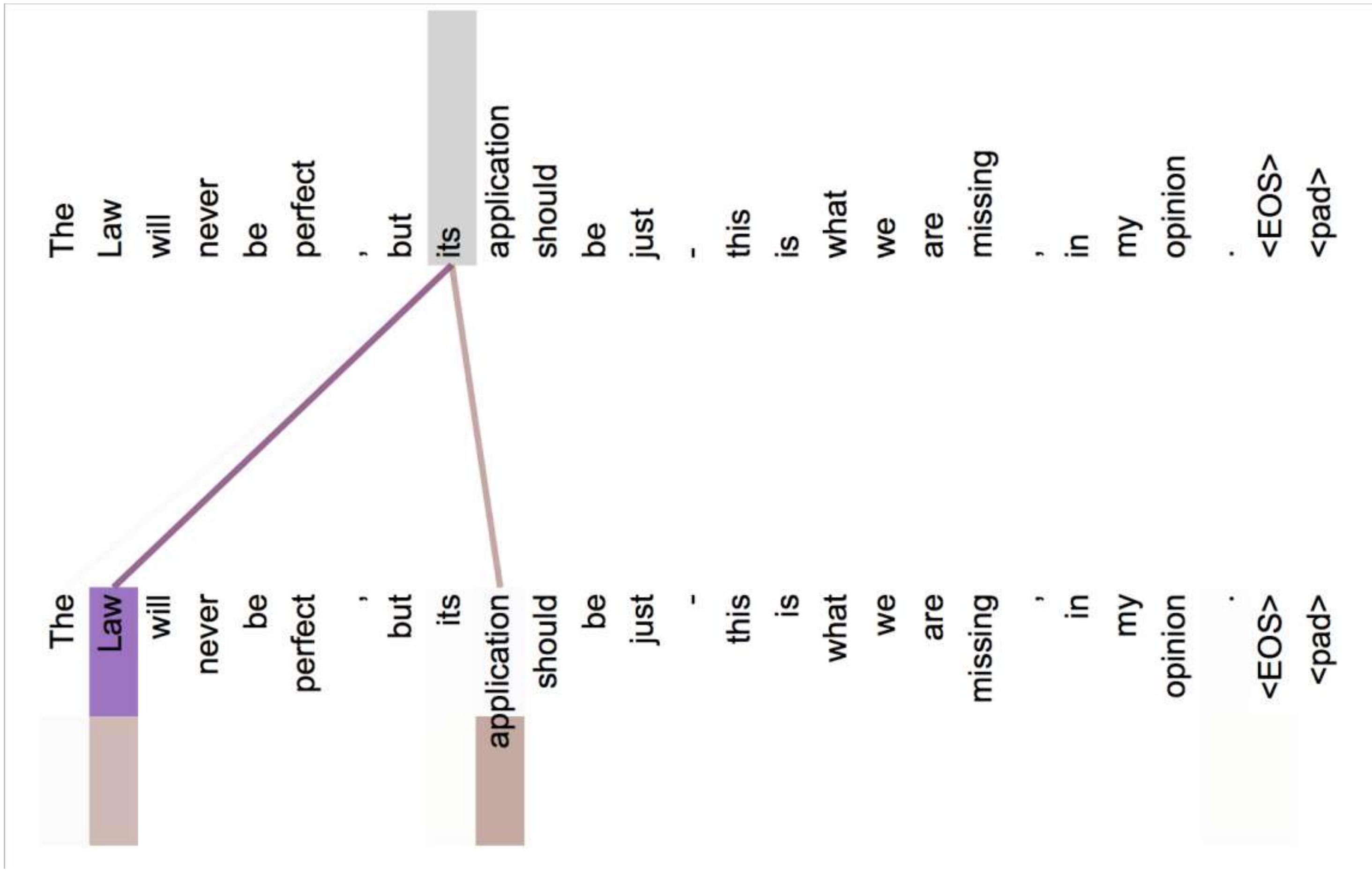


Attention visualization in layer 5

- Words start to pay attention to other words in sensible ways



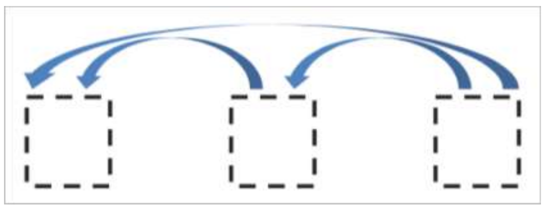
Attention visualization: Implicit anaphora resolution



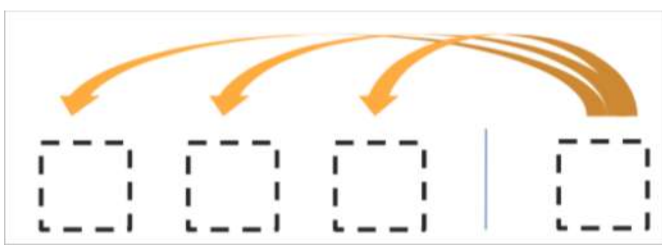
In 5th layer. Isolated attentions from just the word 'its' for attention heads 5 and 6. Note that the attentions are very sharp for this word.

Transformer Decoder

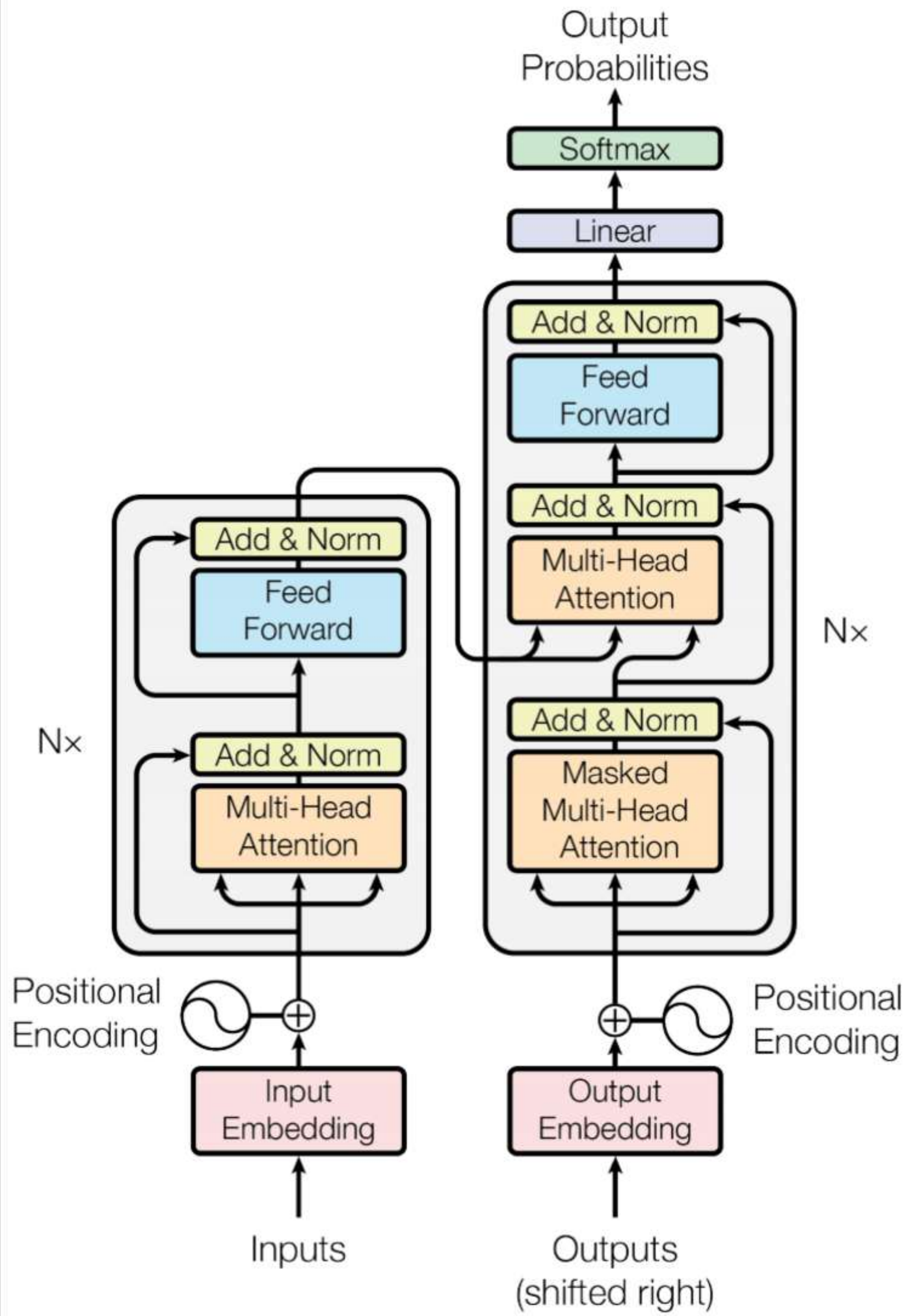
- 2 sublayer changes in decoder
- Masked decoder self-attention on previously generated outputs:



- Encoder-Decoder Attention, where queries come from previous decoder layer and keys and values come from output of encoder



- Blocks repeated 6 times also



Tips and tricks of the Transformer

Details (in paper and/or later lectures):

- Byte-pair encodings
- Checkpoint averaging
- ADAM optimizer with learning rate changes
- Dropout during training at every layer just before adding residual
- Label smoothing
- Auto-regressive decoding with beam search and length penalties

- → Use of transformers is spreading but they are hard to optimize and unlike LSTMs don’t usually just work out of the box and they don’t play well yet with other building blocks on tasks.

Experimental Results for MT

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [38]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	$\mathbf{3.3 \cdot 10^{18}}$	
Transformer (big)	**28.4**	**41.8**	$2.3 \cdot 10^{19}$	

Experimental Results for Parsing

Parser	Training	WSJ 23 F1
Vinyals & Kaiser el al. (2014) [37]	WSJ only, discriminative	88.3
Petrov et al. (2006) [29]	WSJ only, discriminative	90.4
Zhu et al. (2013) [40]	WSJ only, discriminative	90.4
Dyer et al. (2016) [8]	WSJ only, discriminative	91.7
Transformer (4 layers)	WSJ only, discriminative	91.3
Zhu et al. (2013) [40]	semi-supervised	91.3
Huang & Harper (2009) [14]	semi-supervised	91.3
McClosky et al. (2006) [26]	semi-supervised	92.1
Vinyals & Kaiser el al. (2014) [37]	semi-supervised	92.1
Transformer (4 layers)	semi-supervised	92.7
Luong et al. (2015) [23]	multi-task	93.0
Dyer et al. (2016) [8]	generative	93.3

5. BERT: Devlin, Chang, Lee, Toutanova (2018)

BERT (Bidirectional Encoder Representations from Transformers):

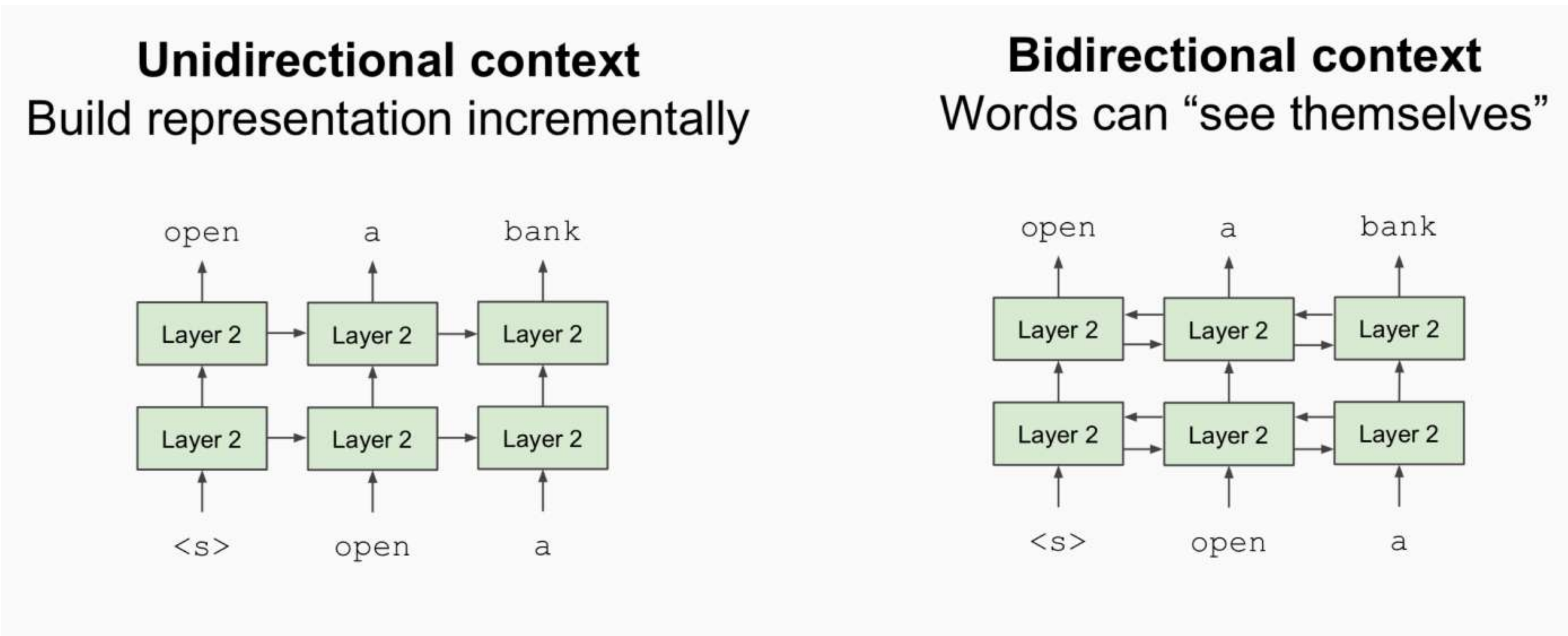
Pre-training of Deep Bidirectional Transformers for Language Understanding

Based on slides from Jacob Devlin

BERT: Devlin, Chang, Lee, Toutanova (2018)

- **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.

BERT: Devlin, Chang, Lee, Toutanova (2018)



BERT: Devlin, Chang, Lee, Toutanova (2018)

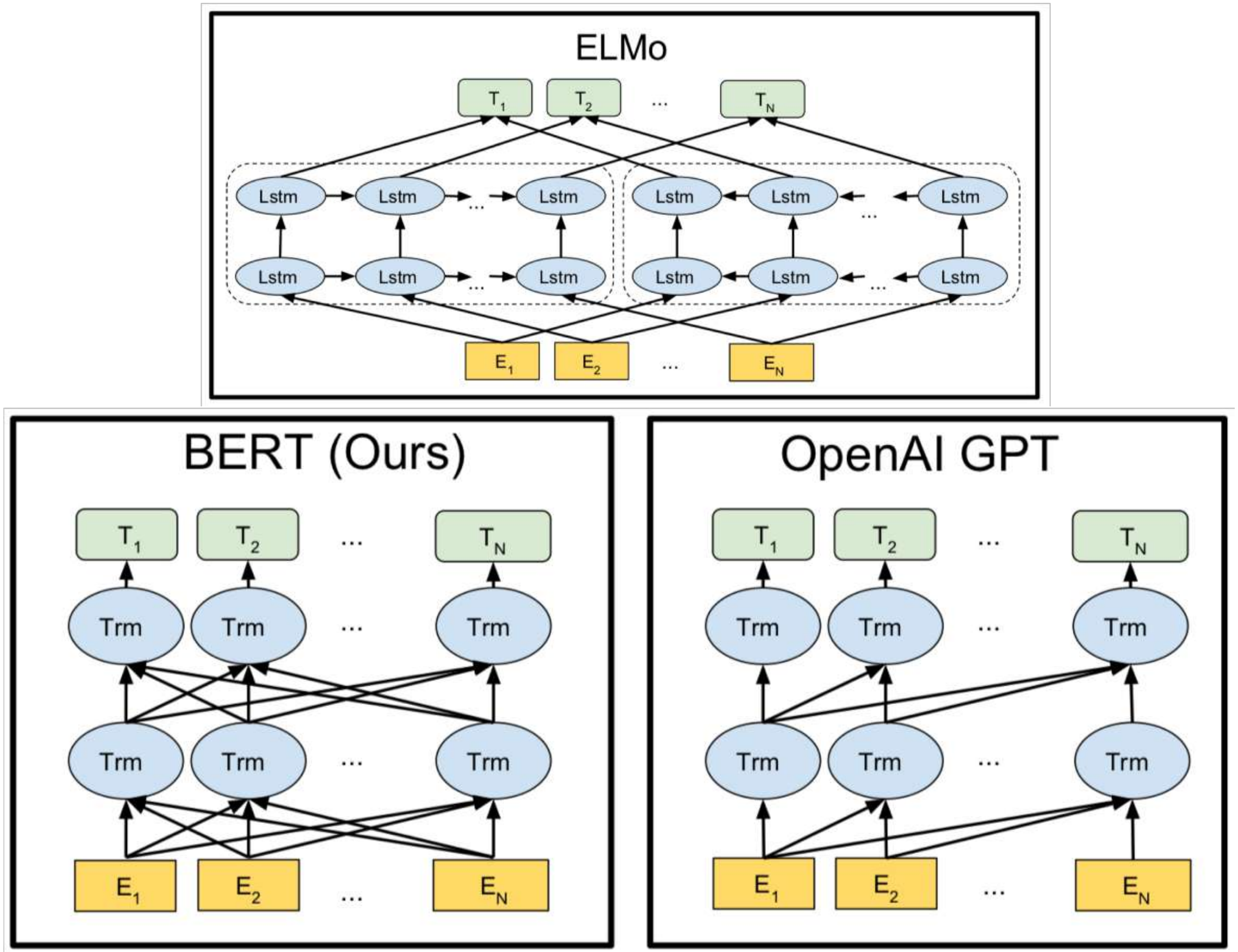
- **Solution**: Mask out *k*% of the input words, and then predict the masked words
  - They always use *k* = 15%

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: Devlin, Chang, Lee, Toutanova (2018)



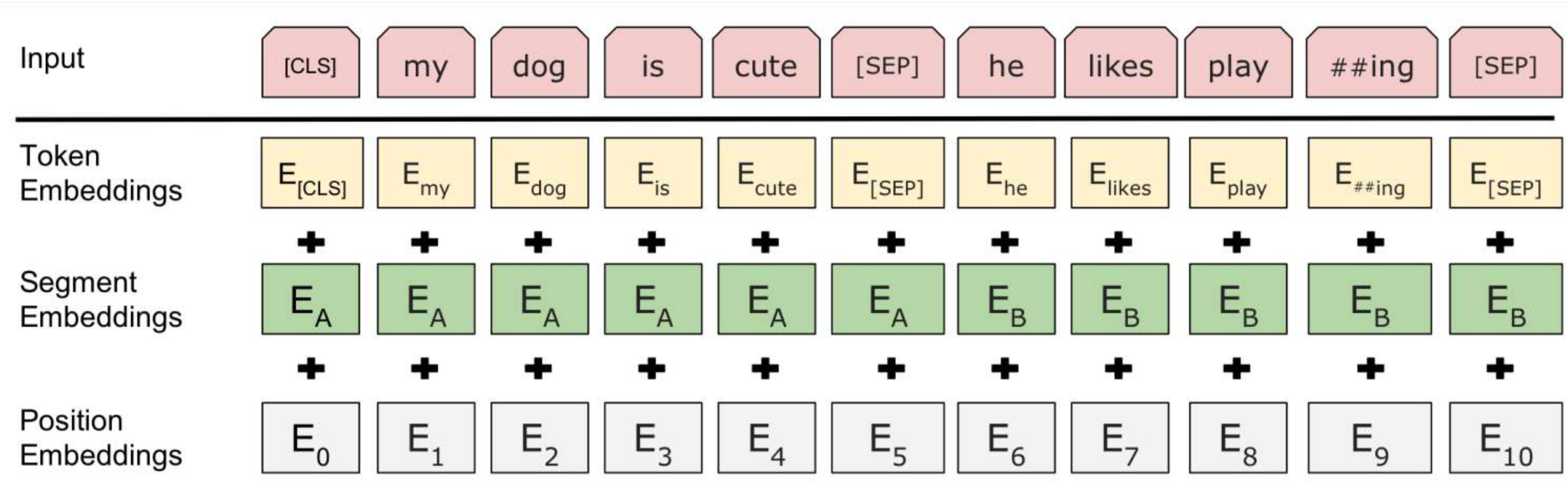
BERT complication: Next sentence prediction

- 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 sentence pair encoding



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

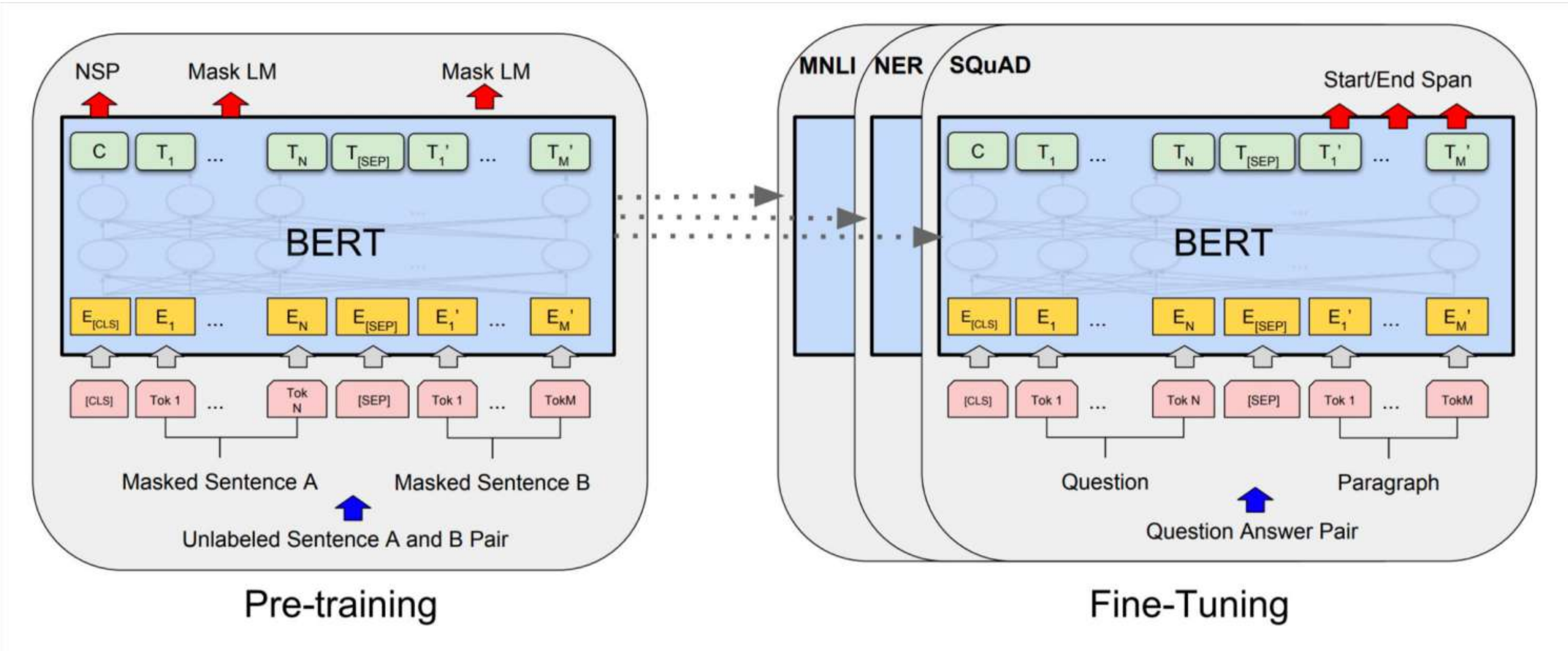
BERT model architecture and training

- Transformer encoder (as before)
- Self-attention ⇒ no locality bias
  - Long-distance context has "equal opportunity"
- Single multiplication per layer ⇒ efficiency on GPU/TPU

- Train on Wikipedia + BookCorpus
- Train 2 model sizes:
  - BERT-Base: 12-layer, 768-hidden, 12-head
  - BERT-Large: 24-layer, 1024-hidden, 16-head
- Trained on 4x4 or 8x8 TPU slice for 4 days

BERT model fine tuning

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



BERT results on GLUE tasks

- GLUE benchmark is dominated by natural language inference tasks, but also has sentence similarity and sentiment

- **MultiNLI**
- Premise: Hills and mountains are especially sanctified in Jainism.
  Hypothesis: Jainism hates nature.
  Label: Contradiction

- **CoLa**
- Sentence: The wagon rumbled down the road. Label: Acceptable
- Sentence: The car honked down the road. Label: Unacceptable

BERT results on GLUE tasks

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**

CoNLL 2003 Named Entity Recognition (en news testb)

Name	Description	Year	F1
Flair (Zalando)	Character-level language model	2018	93.09
BERT Large	Transformer bidi LM + fine tune	2018	92.8
CVT Clark	Cross-view training + multitask learn	2018	92.61
BERT Base	Transformer bidi LM + fine tune	2018	92.4
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

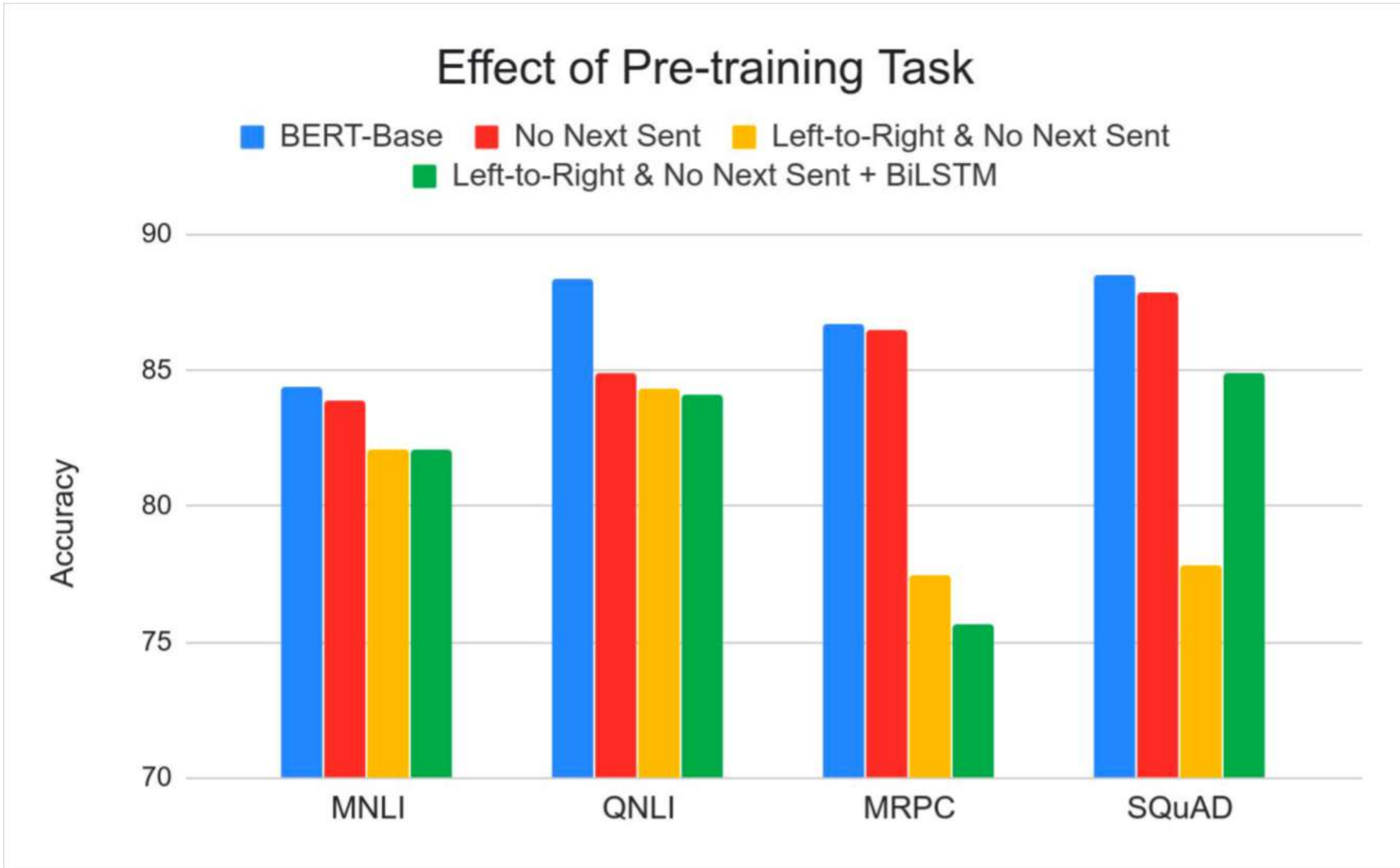
BERT results on SQuAD 1.1

Rank	Model	EM	F1
	Human Performance *Stanford University* (Rajpurkar et al. '16)	82.304	91.221
1 Oct 05, 2018	BERT (ensemble) *Google AI Language* https://arxiv.org/abs/1810.04805	**87.433**	**93.160**
2 Oct 05, 2018	BERT (single model) *Google AI Language* https://arxiv.org/abs/1810.04805	85.083	91.835
2 Sep 26, 2018	nlnet (ensemble) *Microsoft Research Asia*	85.954	91.677
5 Sep 09, 2018	nlnet (single model) *Microsoft Research Asia*	83.468	90.133
3 Jul 11, 2018	QANet (ensemble) *Google Brain & CMU*	84.454	90.490

SQuAD 2.0 leaderboard, 2019-02-07

Rank	Model	EM	F1
	Human Performance *Stanford University* (Rajpurkar & Jia et al. '18)	86.831	89.452
1 Jan 15, 2019	BERT + MMFT + ADA (ensemble) *Microsoft Research Asia*	**85.082**	**87.615**
2 Jan 10, 2019	BERT + Synthetic Self-Training (ensemble) *Google AI Language* https://github.com/google-research/bert	84.292	86.967
3 Dec 13, 2018	BERT finetune baseline (ensemble) *Anonymous*	83.536	86.096
4 Dec 16, 2018	Lunet + Verifier + BERT (ensemble) *Layer 6 AI NLP Team*	83.469	86.043
4 Dec 21, 2018	PAML+BERT (ensemble model) *PINGAN GammaLab*	83.457	86.122
5 Dec 15, 2018	Lunet + Verifier + BERT (single model) *Layer 6 AI NLP Team*	82.995	86.035

Effect of pre-training task



Size matters

- Going from 110M to 340M parameters helps a lot
- Improvements have not yet asymptoted