Deep Neural Networks for Natural Language Processing (AI6127)

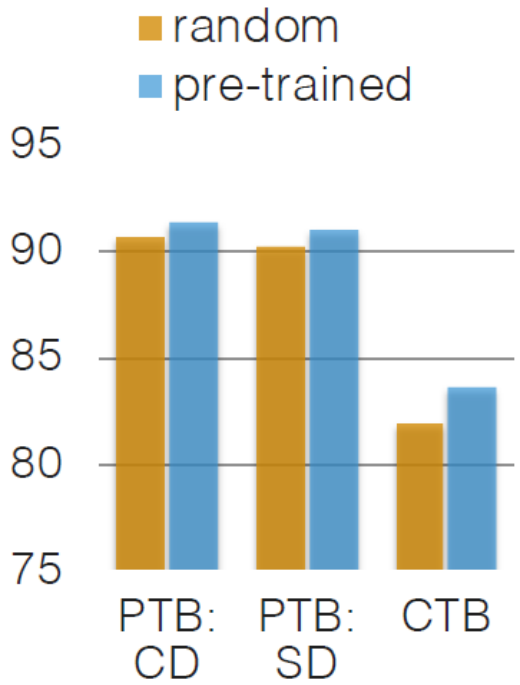
JUNG-JAE KIM

LECTURE 11: CONTEXTUAL REPRESENTATIONS

Slides are originally from https://web.stanford.edu/class/cs224n/slides/cs224n-2019-lecture13-contextual-representations.pdf and https://web.stanford.edu/class/cs224n/slides/cs224n-2019-lecture14-transformers.pdf

Representations for a word

- Word vectors: we have one (pre-trained) representation of words
  - E.g. Word2vec, GloVe, fastText
- In most cases, use of pre-trained word vectors helps, because we can train them for more words on much more data
  - E.g. Chen and Manning (2014) dependency parsing



Issues in word vectors

- 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

1. pen — a writing implement with a point from which ink flows.
2. pen — an enclosure for confining livestock.
3. playpen, pen — a portable enclosure in which babies may be left to play.
4. penitentiary, pen — a correctional institution for those convicted of major crimes.
5. pen — female swan.

Little John was looking for his toy box. Finally he found it. The box was in the pen. John was very happy.

http://www.scholarpedia.org/article/Word_sense_disambiguation

Issues in word vectors

- 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 and syntactic behavior

Noun

- S: (n) consequence, effect, outcome, **result**, event, issue, upshot (a phenomenon that follows and is caused by some previous phenomenon) *"the magnetic effect was greater when the rod was lengthwise"; "his decision had depressing consequences for business"; "he acted very wise after the event"*
- S: (n) solution, answer, **result**, resolution, solvent (a statement that solves a problem or explains how to solve the problem) *"they were trying to find a peaceful solution"; "the answers were in the back of the book"; "he computed the result to four decimal places"*
- S: (n) **result**, resultant, final result, outcome, termination (something that results) *"he listened for the results on the radio"*
- S: (n) resultant role, **result** (the semantic role of the noun phrase whose referent exists only by virtue of the activity denoted by the verb in the clause)

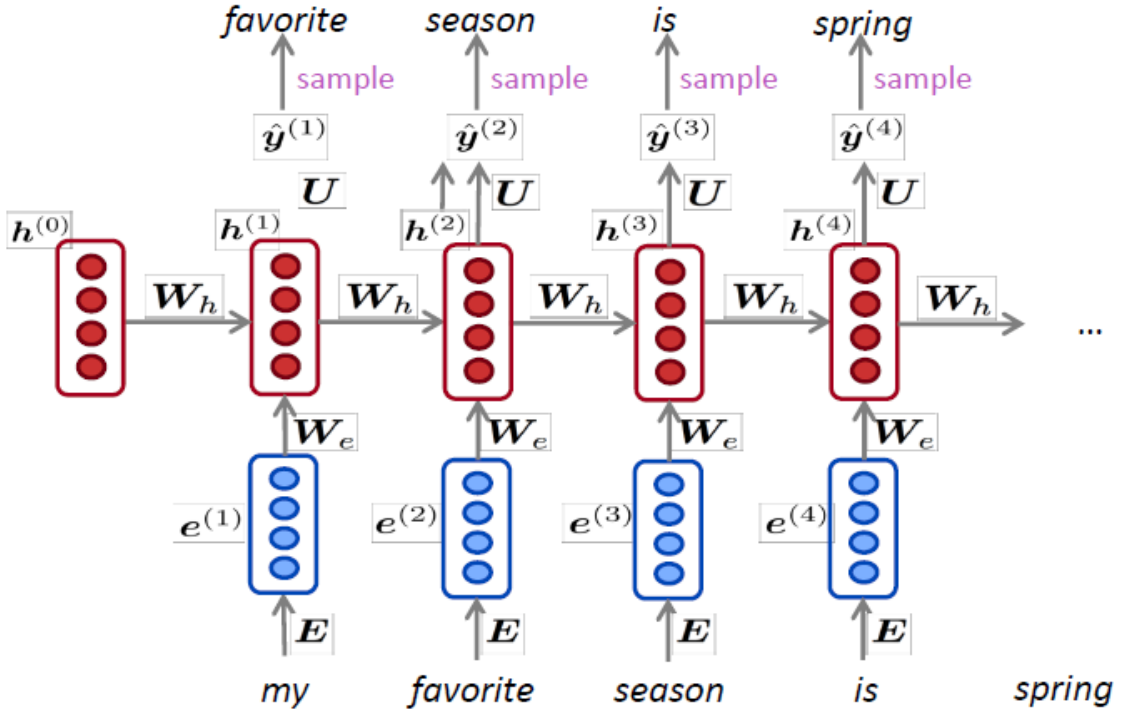
Verb

- S: (v) **result**, ensue (issue or terminate (in a specified way, state, etc.); end) *"result in tragedy"*
- S: (v) leave, **result**, lead (produce as a result or residue) *"The water left a mark on the silk dress"; "Her blood left a stain on the napkin"*
- S: (v) **result** (come about or follow as a consequence) *"nothing will result from this meeting"*

https://wordnet.princeton.edu/

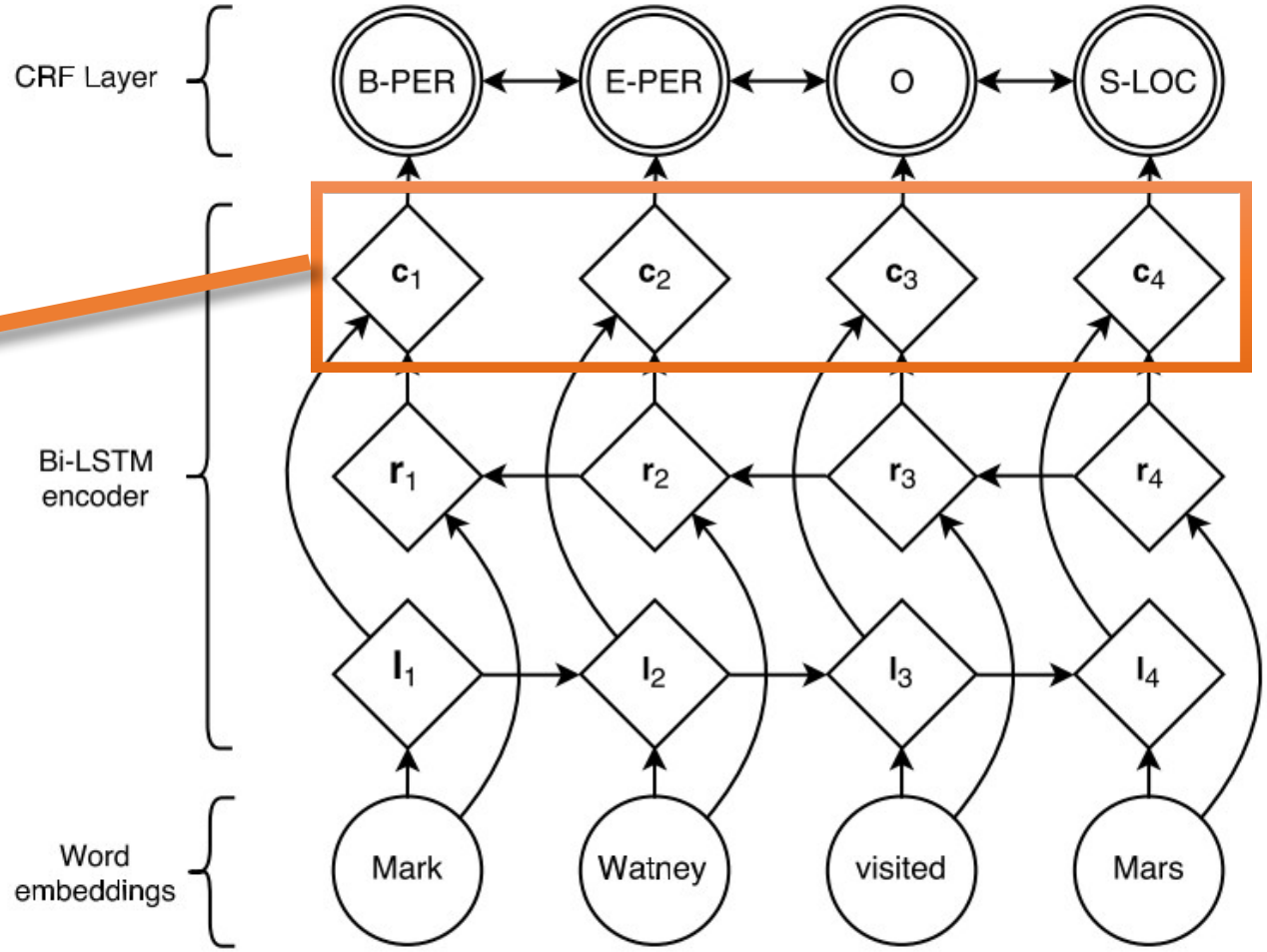
Did we all along have a solution to this problem?

- In an NLM (neural language model), LSTM layers are trained to predict the next word
- The LSTM layers take word vectors as input, but produce context-specific word representations at each position



Contextualized word representation in supervised task neural network

Contextualized Word Representation



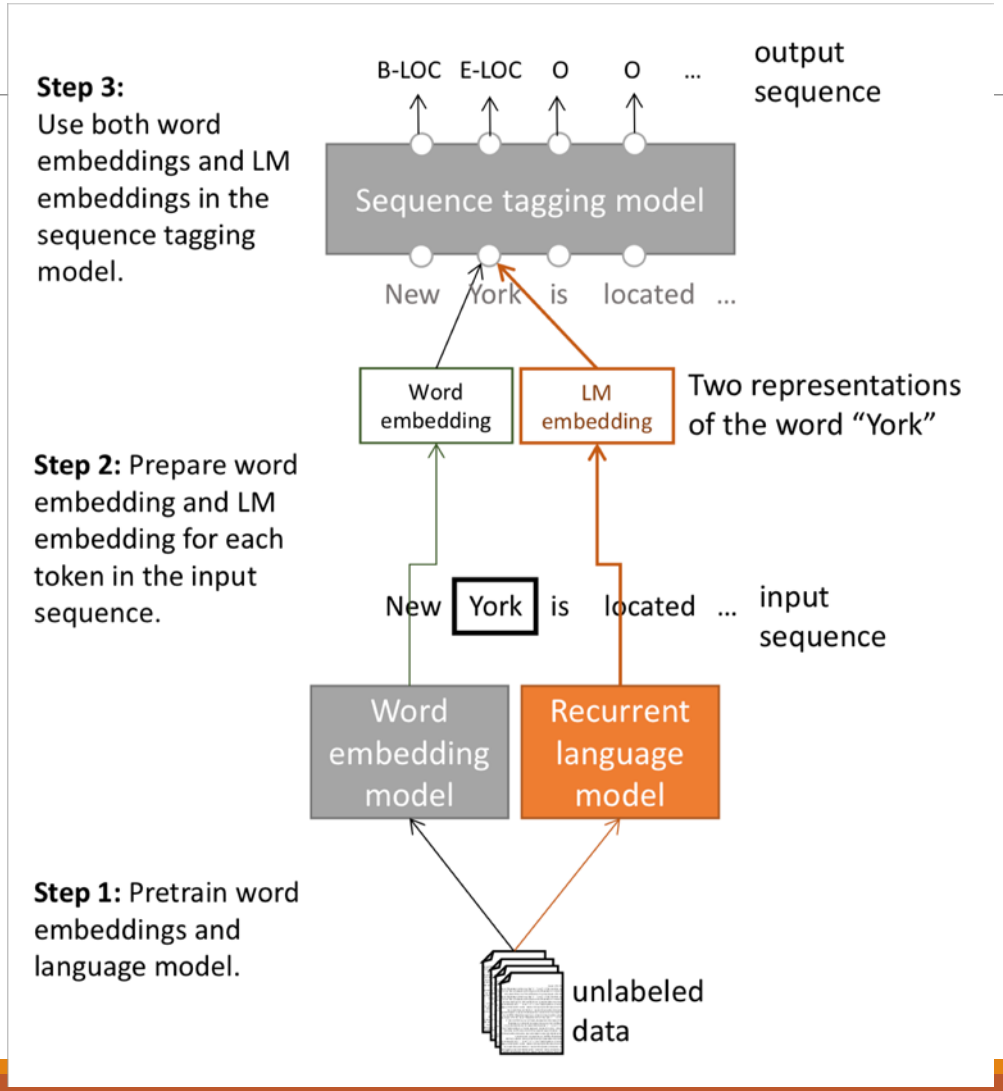
https://courses.engr.illinois.edu/cs546/sp2018/Slides/Apr24_Peters.pdf

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

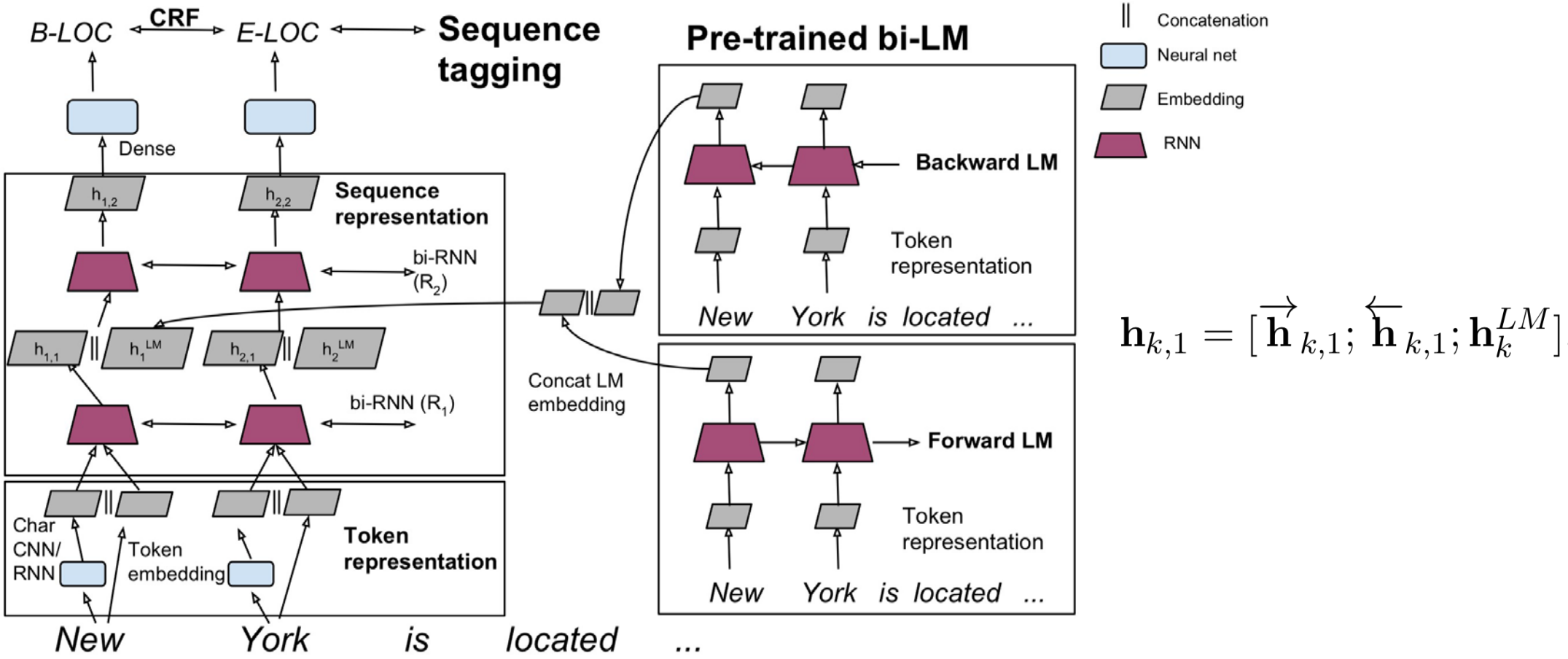
Peters et al. (2017): TagLM – “Pre-ELMo”

- 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)

- 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

Tag LM

- Language model is trained on 800 million training words of “Billion word benchmark” https://www.statmt.org/lm-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

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

- Initial 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 performance but not overly large LM:
  - Use 2 biLSTM layers
  - Use character CNN to build initial word representation (only) $\boldsymbol{x}_k$
    - 2048 char n-gram filters and 2 highway layers (Srivastava et al. 2015), 512 dim projection
  - Use 4096 dim hidden/cell LSTM states with 512 dim projections to next input
  - Use a residual connection from the first LSTM layer to the second layer
- First run biLM to get representations for each word $\boldsymbol{h}_{k,j}$
- Then let (whatever) end-task model use them

Highway network vs Residual connection

- Highway network (Srivastava et al. 2015)

$$\mathbf{z} = \mathbf{t} \odot g(\mathbf{W}_H \mathbf{y} + \mathbf{b}_H) + (\mathbf{1} - \mathbf{t}) \odot \mathbf{y}$$

- Residual connection
  - z = module$_x$(y) **+ y**

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

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

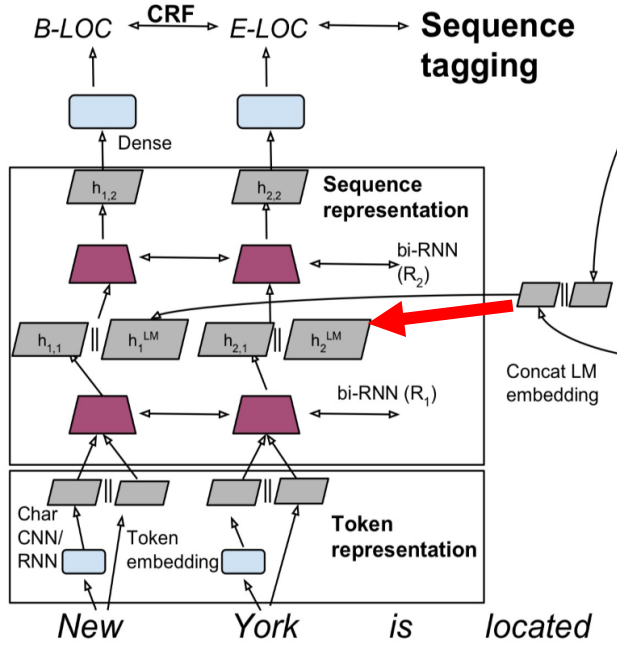
$$
\begin{aligned}
R_k &= \{\mathbf{x}_k^{LM}, \overrightarrow{\mathbf{h}}_{k,j}^{LM}, \overleftarrow{\mathbf{h}}_{k,j}^{LM} \mid j = 1, \ldots, L\} \\
&= \{\mathbf{h}_{k,j}^{LM} \mid j = 0, \ldots, L\},
\end{aligned}
$$

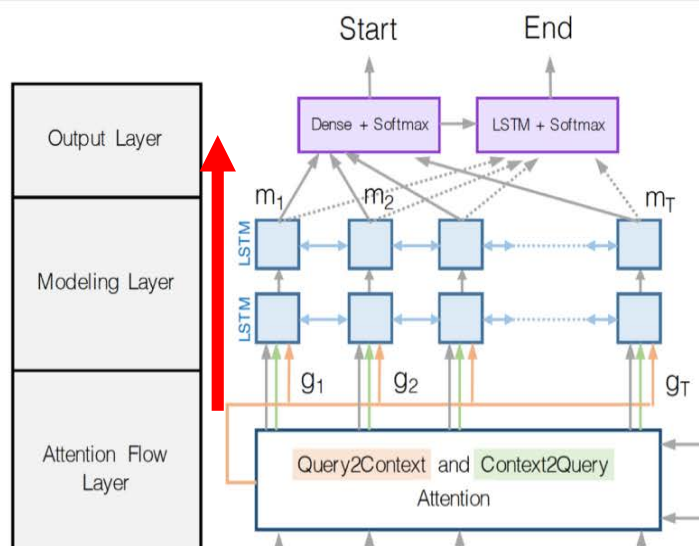
$$
\mathbf{ELMo}_k^{task} = E(R_k; \Theta^{task}) = \gamma^{task} \sum_{j=0}^{L} s_j^{task} \mathbf{h}_{k,j}^{LM}
$$

- 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 $\mathbf{ELMo}_k^{task}$ 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





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

ELMo results: Great for all tasks

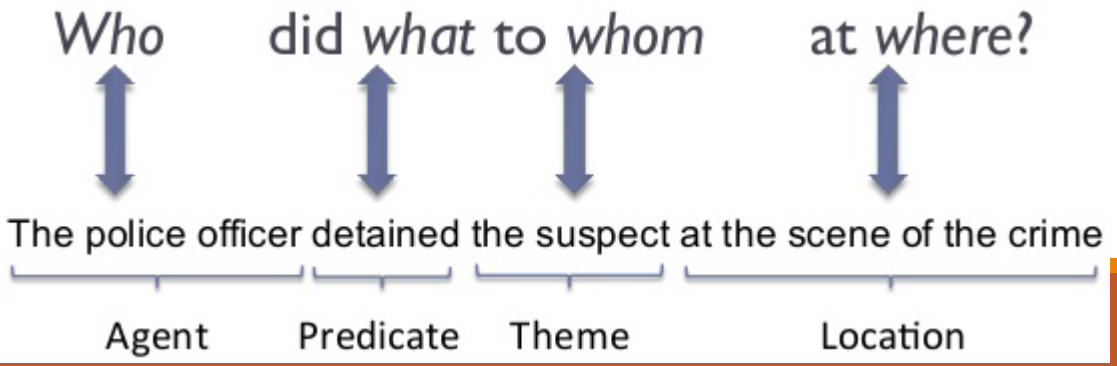
Task	Previous SOTA		Our baseline	ELMo + baseline	Increase (absolute/ relative)
SQuAD	Liu et al. (2017)	84.4	81.1	85.8	4.7 / 24.9%
SNLI	Chen et al. (2017)	88.6	88.0	88.7 ± 0.17	0.7 / 5.8%
SRL	He et al. (2017)	81.7	81.4	84.6	3.2 / 17.2%
Coref	Lee et al. (2017)	67.2	67.2	70.4	3.2 / 9.8%
NER	Peters et al.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

- Question answering (QA), natural language inference (NLI), semantic role labeling (SRL), coreference (coref) resolution, named entity recognition (NER), semantic similarity test (SST)

Examples of NLP applications tested by ELMo

- Natural language inference
  - "hypothesis" is true (entailment), false (contradiction), or undetermined (neutral) given a "premise"
- Semantic role labeling

Premise	Label	Hypothesis
A man inspects the uniform of a figure in some East Asian country.	contra-diction	The man is sleeping.
An older and younger man smiling.	neutral	Two men are smiling and laughing at the cats playing on the floor.
A soccer game with multiple males playing.	entail-ment	Some men are playing a sport.

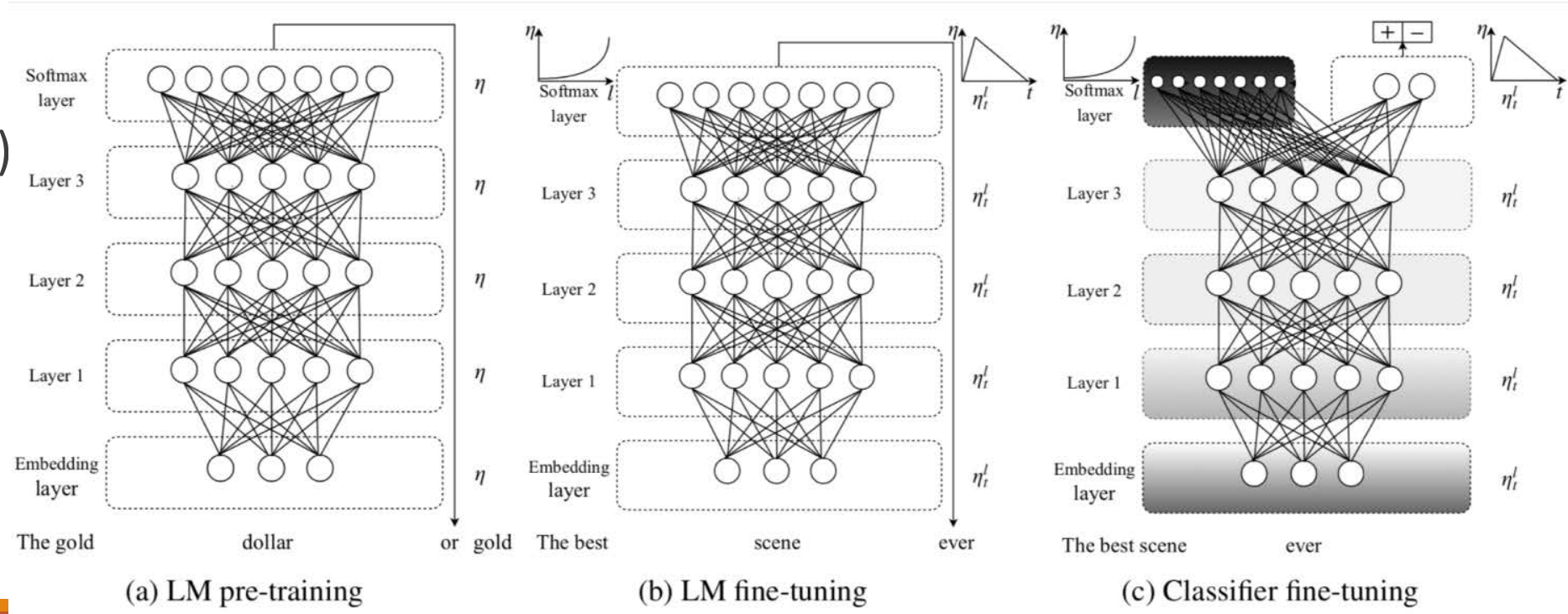


ELMo: Weighting of layers

- 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

ULMfit

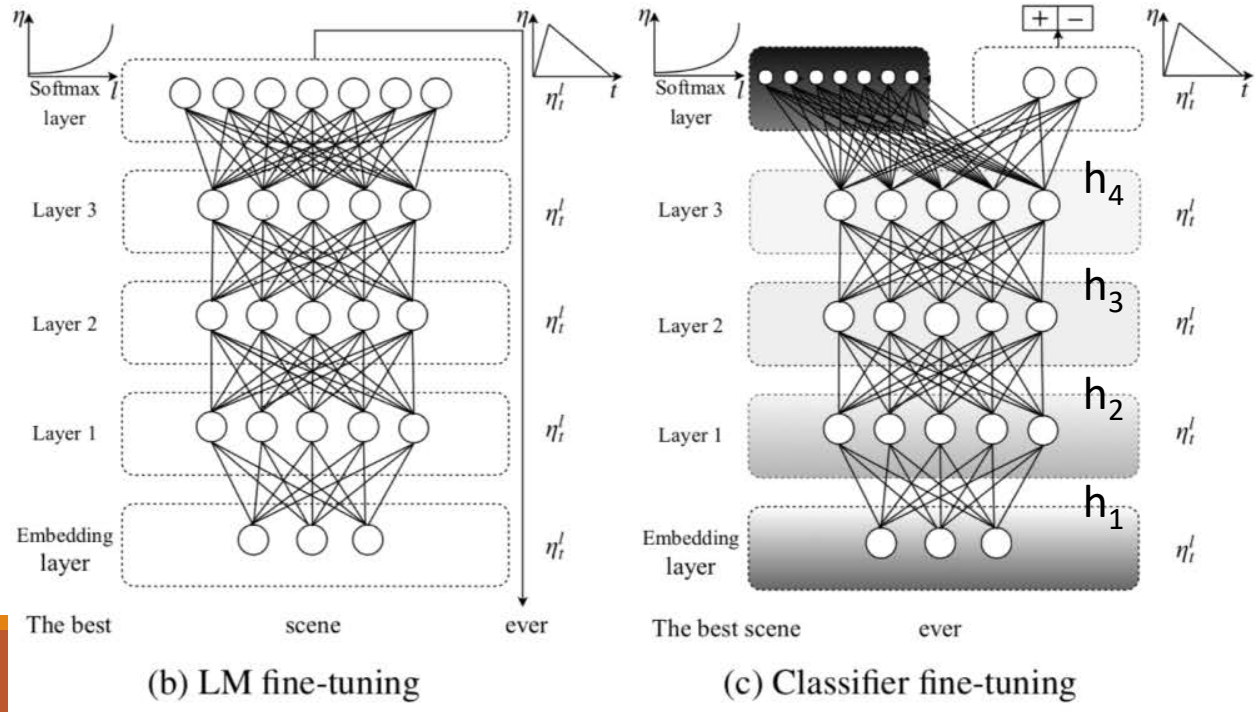
- Same general idea of transferring NLM knowledge
- Applied to text classification
  - Train LM on big general domain corpus (use biLM)
  - Tune LM on target task data
  - Fine-tune as classifier on target task



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

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
  - First unfreeze the last layer and then next layers one by one
- Classify using concatenation [h_T, maxpool(**H**), meanpool(**H**)]
  - H = [h_1, …, h_T]

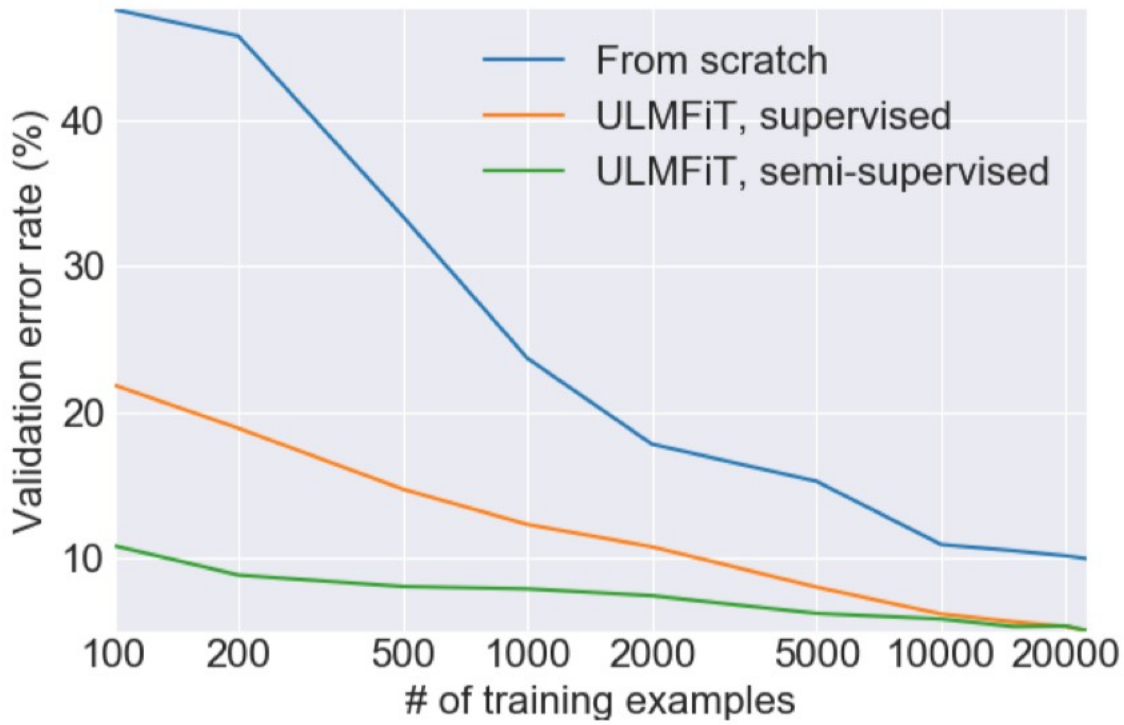


ULMfit performance

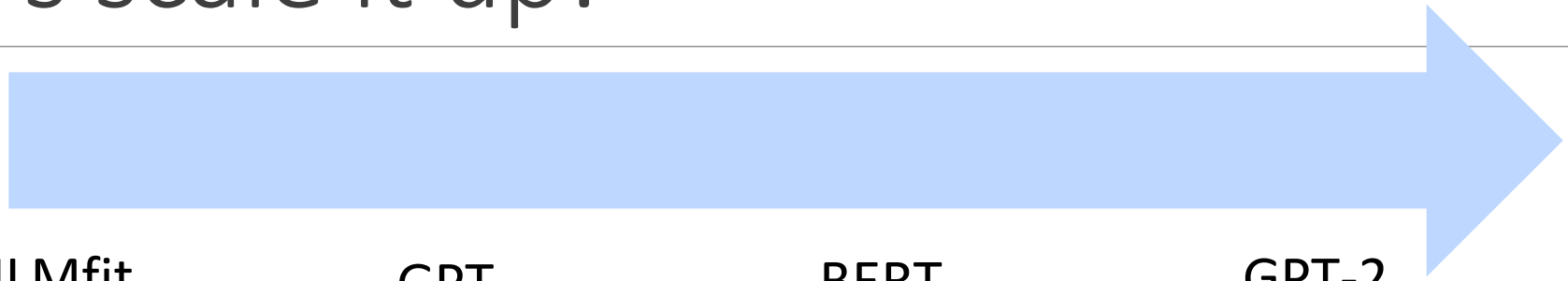
- Text classifier error rates

	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

Training Bert Base model took 4 days using 4 Cloud TPUs (16 TPU chips in total)

Transformer models

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

ULMfit
Jan 2018
Training:
1 GPU day

GPT
June 2018
Training
240 GPU days

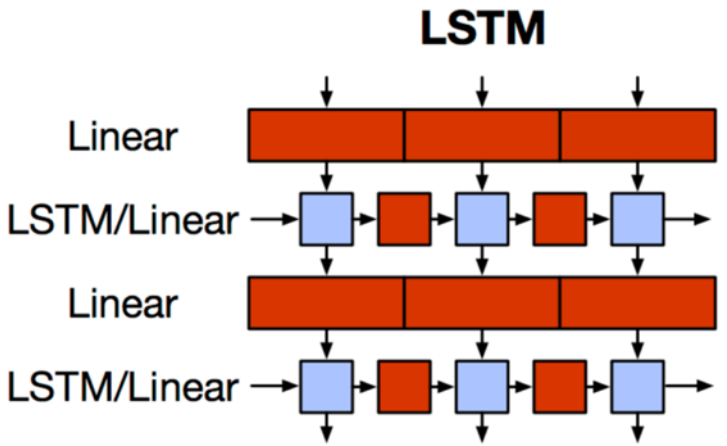
BERT
Oct 2018
Training
256 TPU days
~320–560 GPU days

GPT-2
Feb 2019
Training
~2048 TPU v3 days according to a reddit thread

XL-Net,
ERNIE,
Grover
RoBERTa, T5
July 2019 -

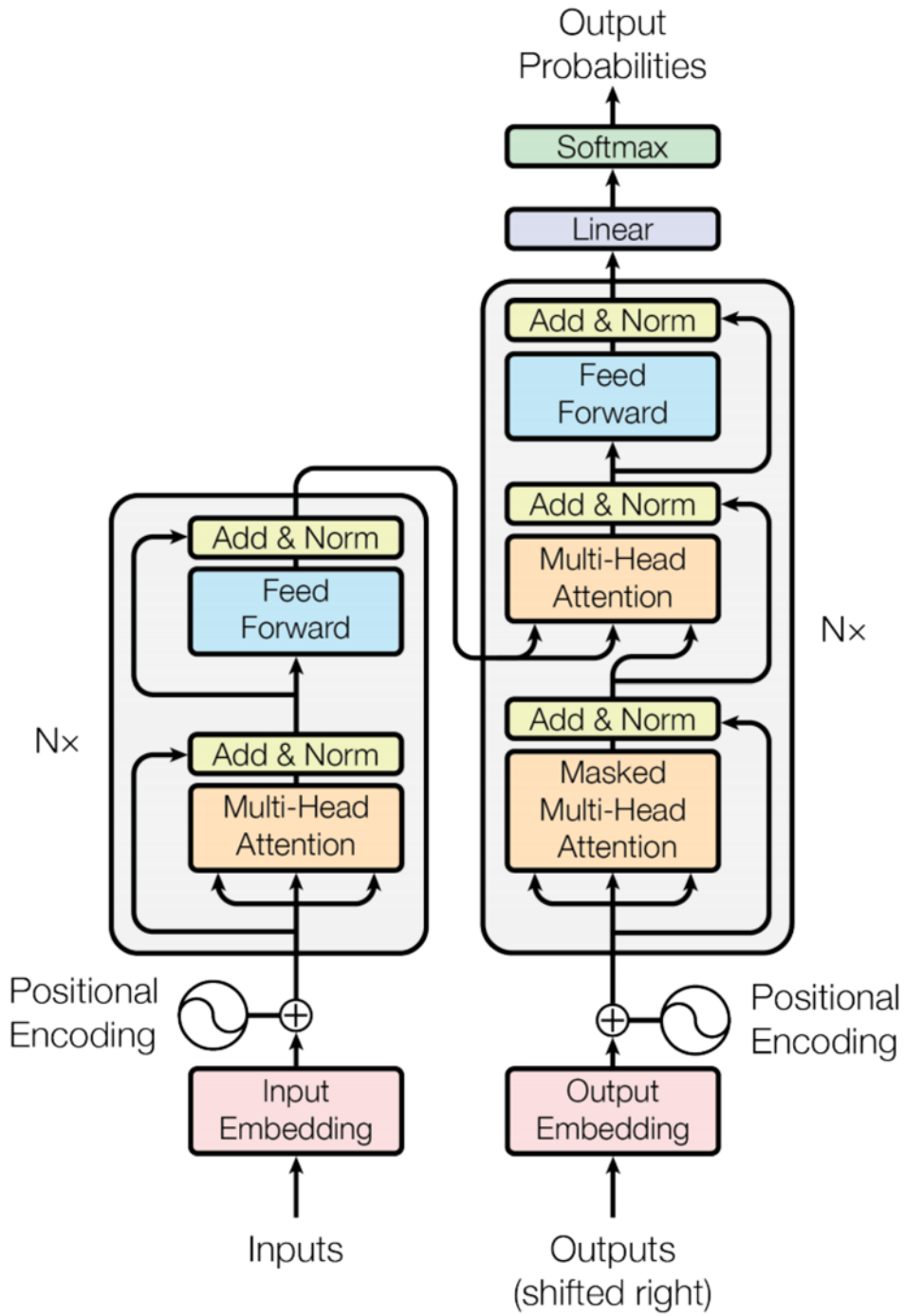
The Motivation for Transformers

- We want parallelization but RNNs are inherently sequential
- Despite GRUs and LSTMs, RNNs still need attention mechanism to deal with long range dependencies
- But if attention gives us access to any state… maybe we can just use attention and don't need the RNN?



Transformer Overview

- 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



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

Transformer Basics

- Learning about transformers on your own?
  - 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 definition)

- Previous definition: Given a set of vector *values*, and a vector *query*, **attention** is a technique to compute a weighted sum of the values, dependent on the query.
- 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 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

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

$[|Q| \times d_k] \times [d_k \times |K|] \times [|K| \times d_v]$

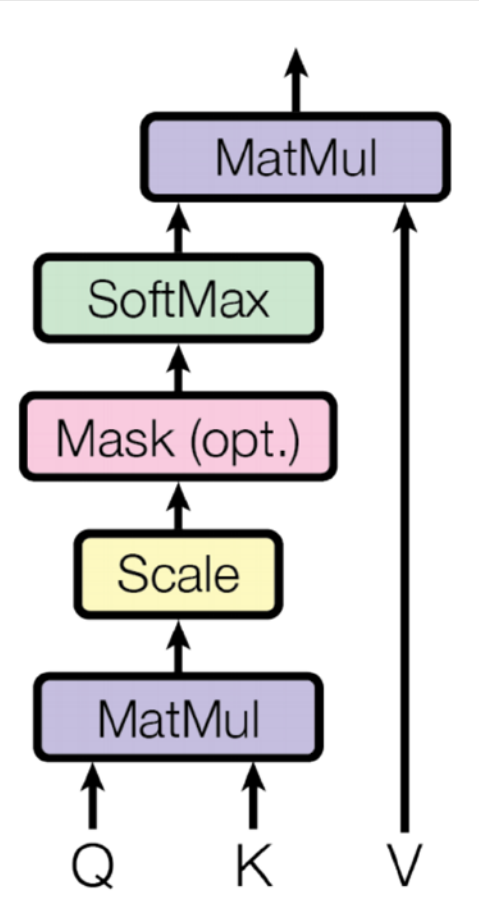
softmax row-wise

$= [|Q| \times d_v]$

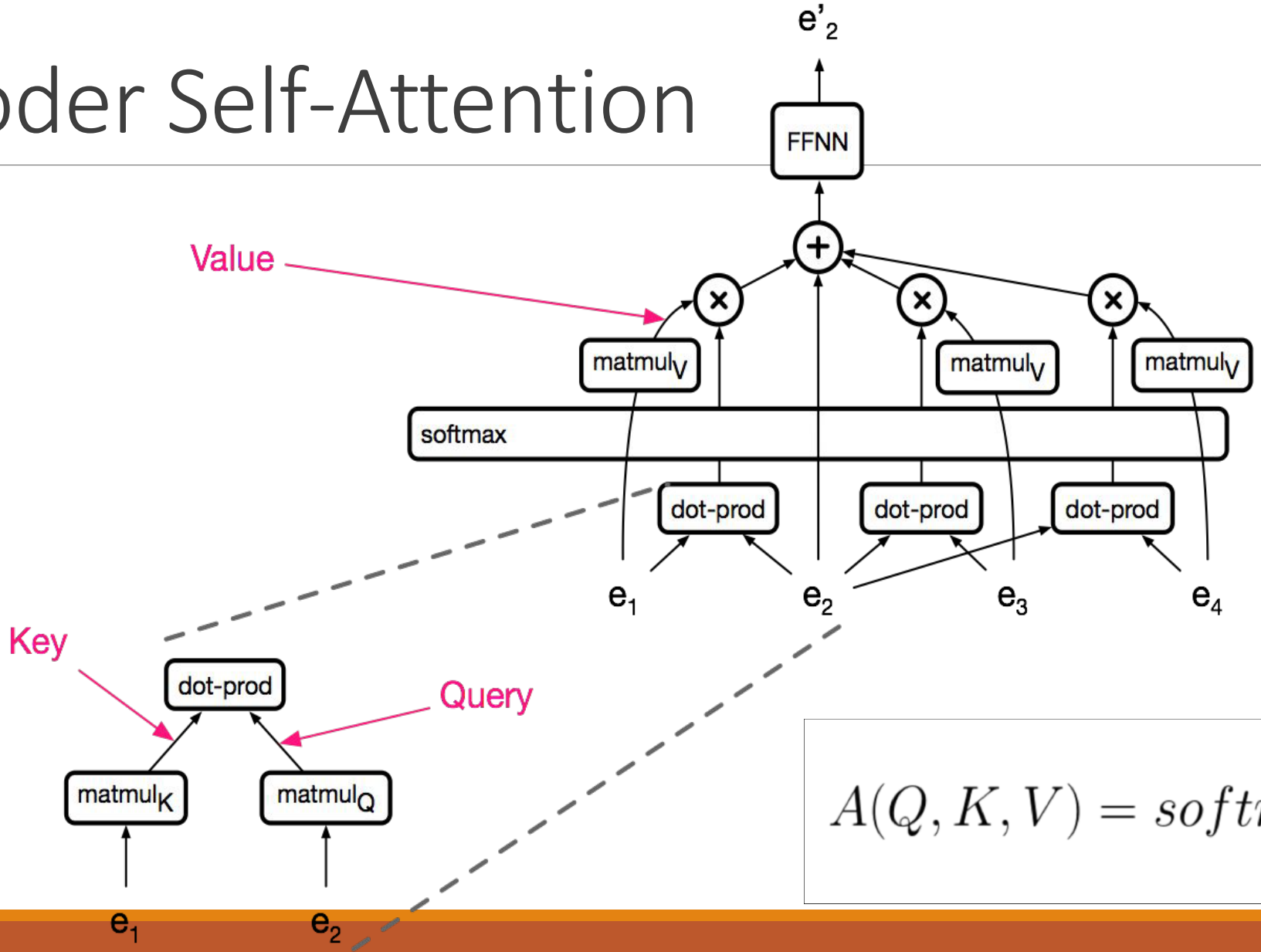
Scaled Dot-Product Attention

- Problem:
  - As d_k gets large, the variance of $q^T k$ increases $\rightarrow$ some values inside the softmax get large $\rightarrow$ the softmax gets very peaked $\rightarrow$ hence its gradient gets smaller
- Solution: Scale by length of query/key vectors

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

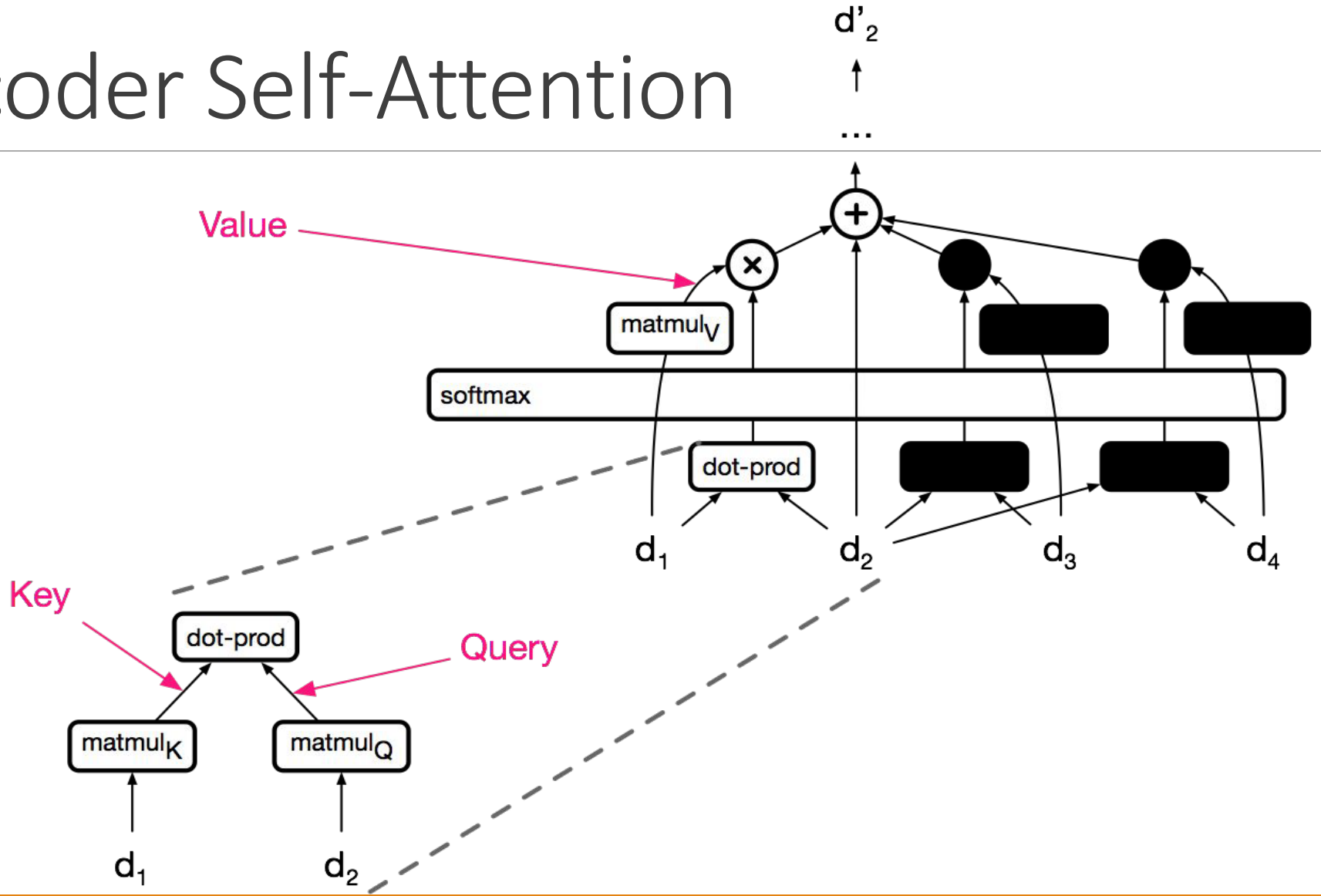


Encoder Self-Attention



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

Decoder Self-Attention



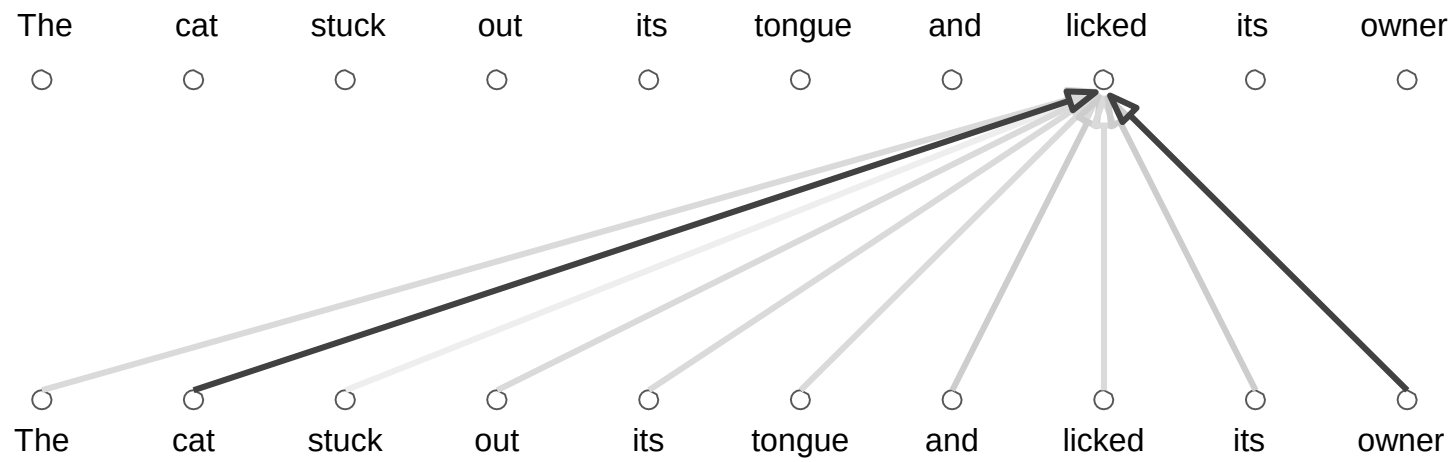
Attention is Cheap!

	FLOPs	
Self-Attention	$O(\text{length}^2 \cdot \text{dim})$	$= 4 \cdot 10^9$
RNN (LSTM)	$O(\text{length} \cdot \text{dim}^2)$	$= 16 \cdot 10^9$
Convolution	$O(\text{length} \cdot \text{dim}^2 \cdot \text{kernel\_width})$	$= 6 \cdot 10^9$

length=1000 dim=1000 kernel_width=3

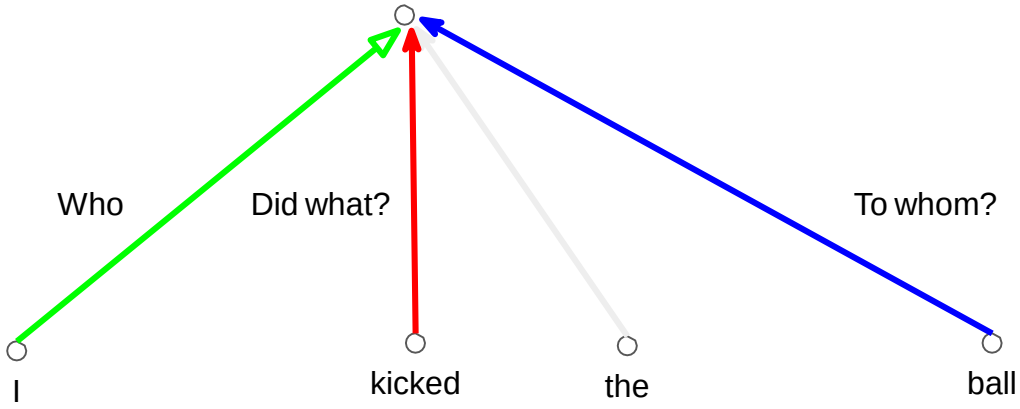
Attention: a weighted average

- Problem with simple self-attention:
  - Only one way for words to interact with one-another

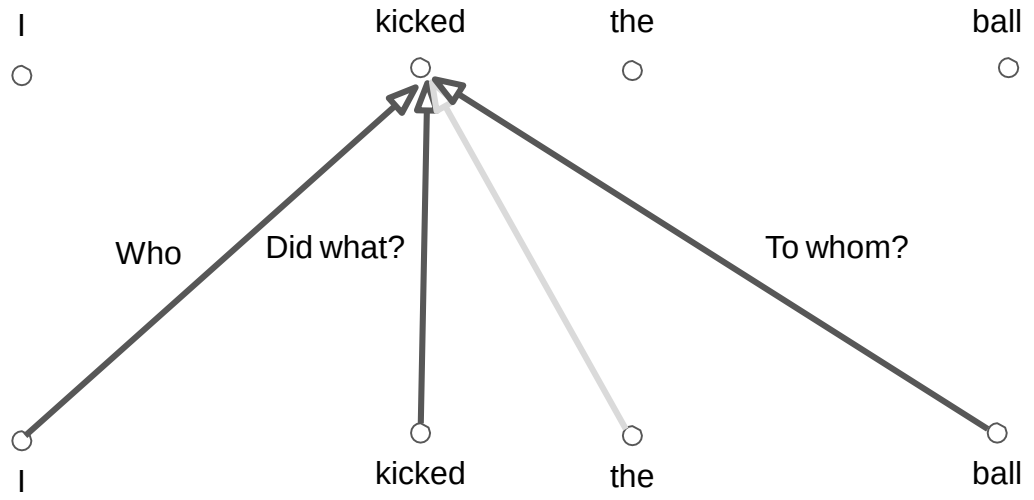


Convolutions

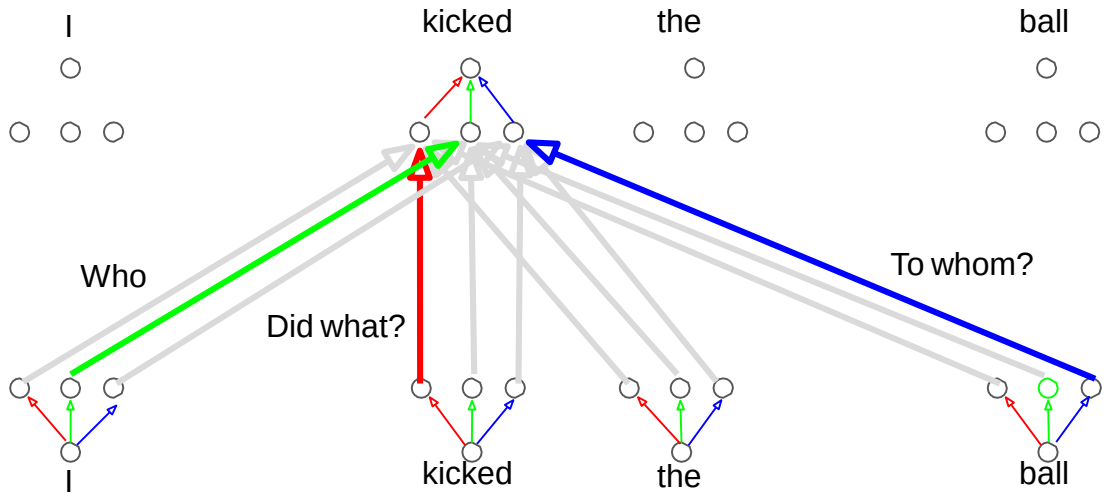
- Using multi-filter/channel



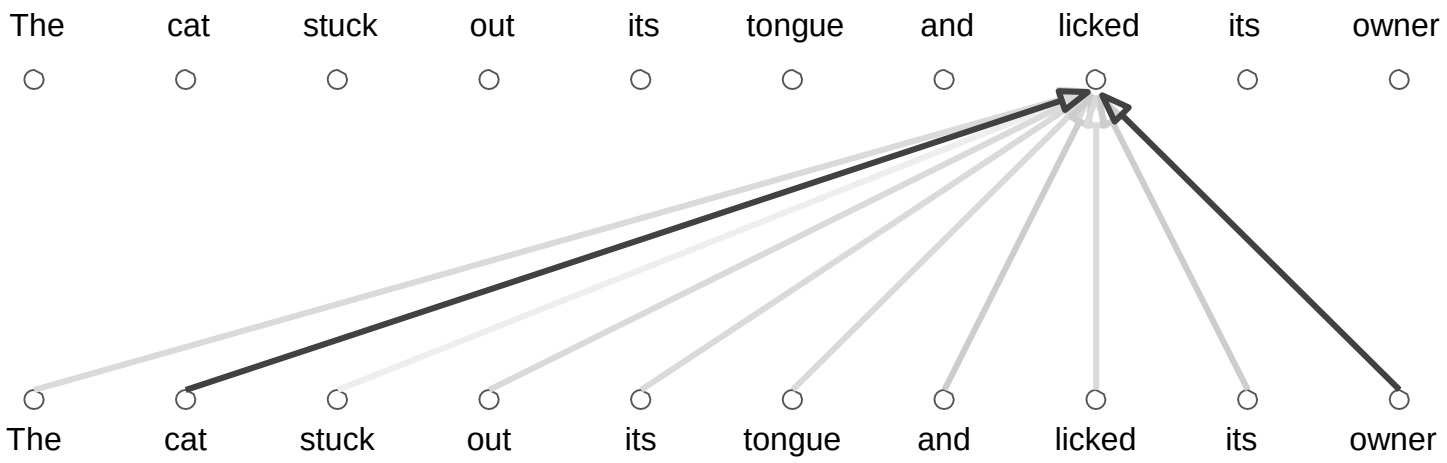
Self-Attention



Parallel attention heads

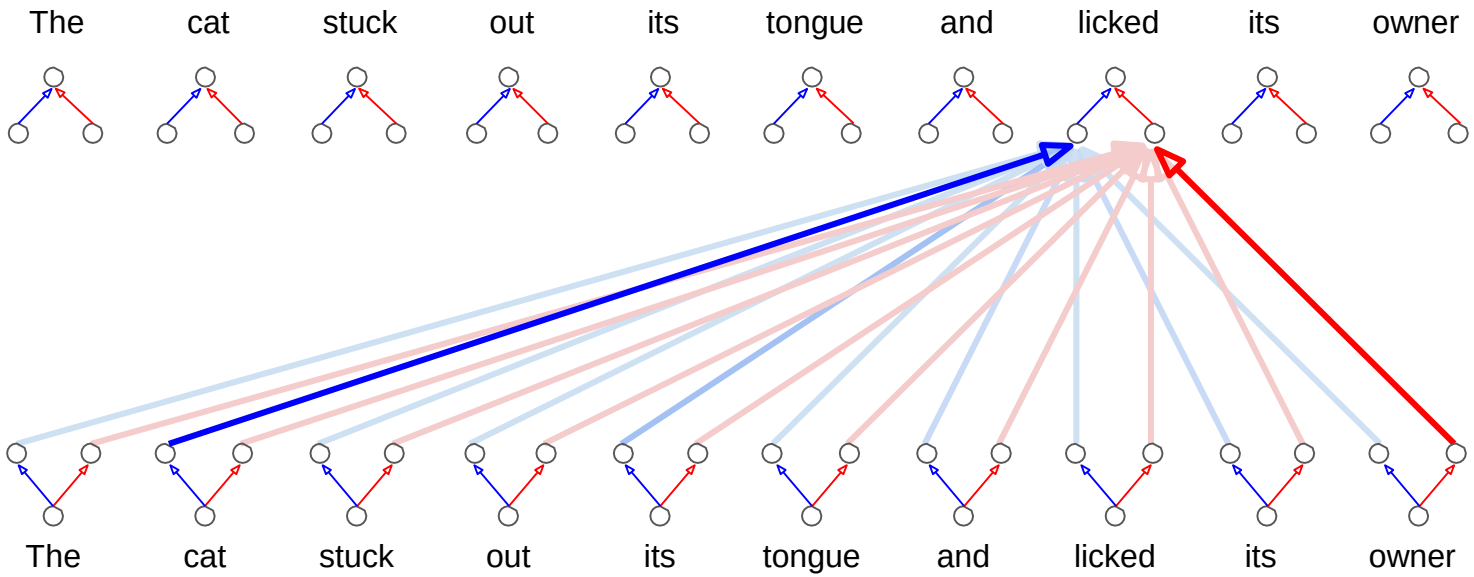


Attention: a weighted average



Multi-head Attention

- Parallel attention layers with different linear transformations on input and output

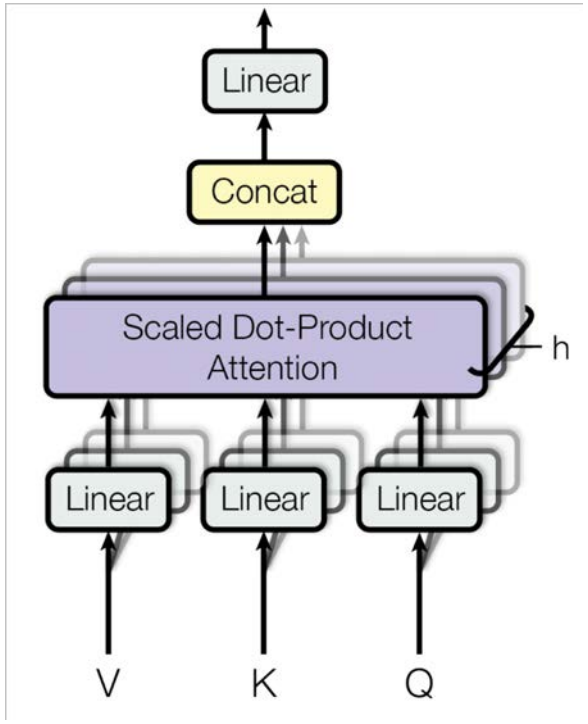


Multi-head attention

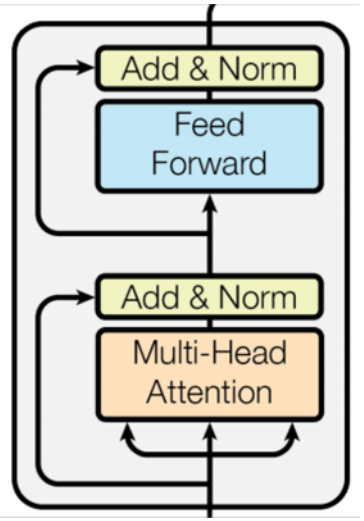
- Problem with simple self-attention:
  - Only one way for words to interact with one-another
- Solution: Multi-head attention
  - First map Q, K, V into h=8 many lower dimensional spaces via W matrices
  - Then apply attention, then concatenate outputs and pipe through linear layer

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O$$

$$\text{where head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$



Complete transformer block



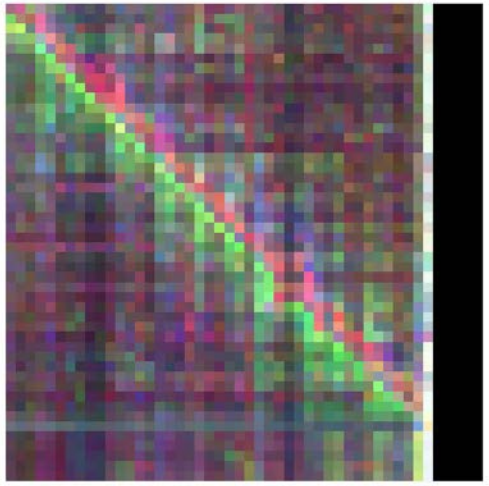
- Each block has two "sublayers"
  - Multi-head attention
  - 2-layer feed-forward NNet (with ReLU)
- Each of these two steps also has:
  - Residual (short-circuit) connection and LayerNorm
    - LayerNorm(x + Sublayer(x))
  - Layernorm changes input to have mean 0 and variance 1, per layer and per training point (and adds two more parameters)

$$\mu^l = \frac{1}{H}\sum_{i=1}^{H} a_i^l \qquad \sigma^l = \sqrt{\frac{1}{H}\sum_{i=1}^{H}(a_i^l - \mu^l)^2} \qquad h_i = f\left(\frac{g_i}{\sigma_i}(a_i - \mu_i) + b_i\right)$$

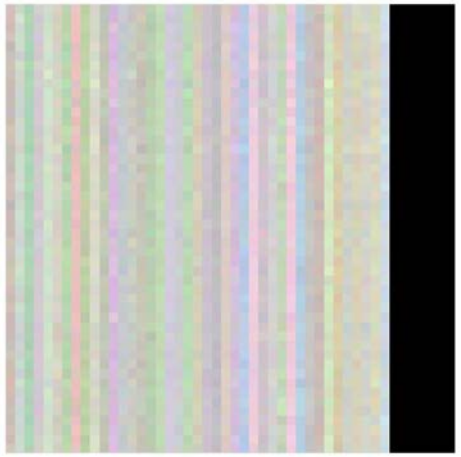
Layer Normalization by Ba, Kiros and Hinton, https://arxiv.org/pdf/1607.06450.pdf

Importance of Residuals

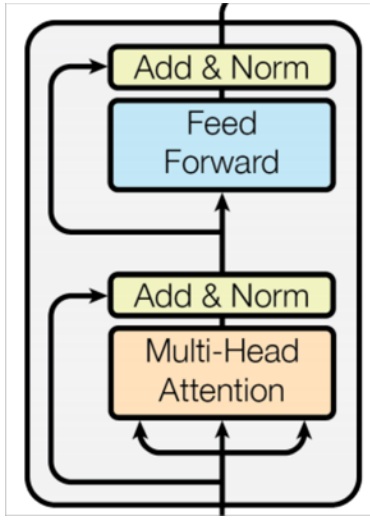
- Residuals carry positional information to higher layers, among other information



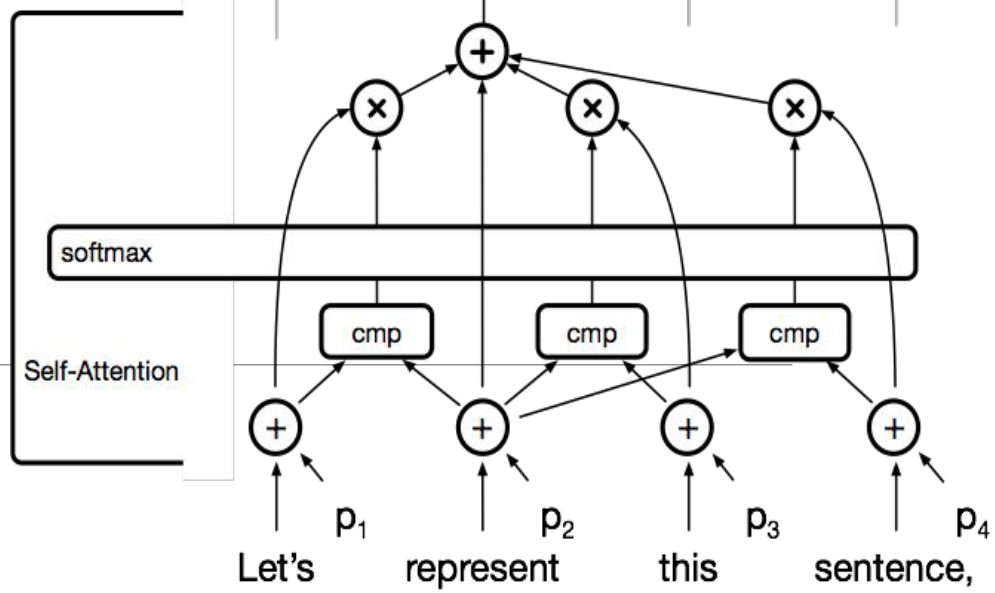
With residuals



Without residuals



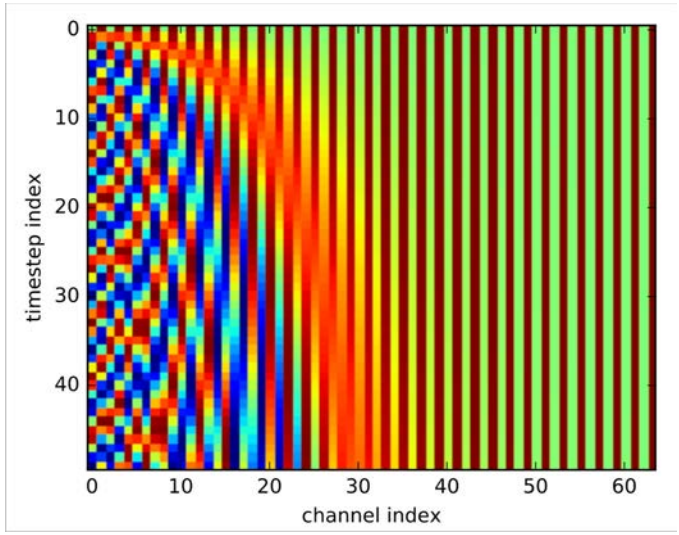
Encoder Input



- Actual word representations are byte-pair encodings
- Also added is a positional encoding so same words at different locations have different overall representations:

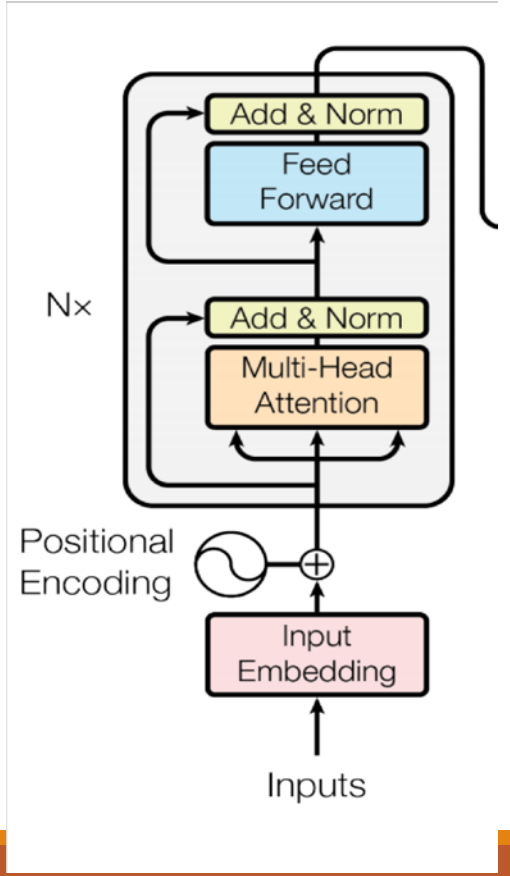
$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

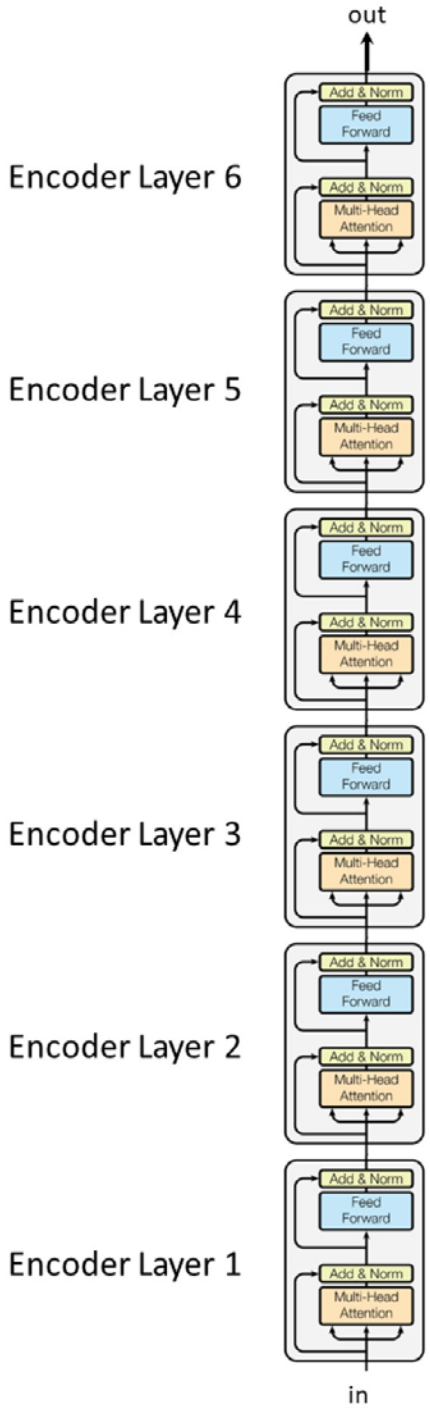
  - pos: each position
  - i: dimension of positional encoding
  - d_{model}: number of dimensions in positional encoding



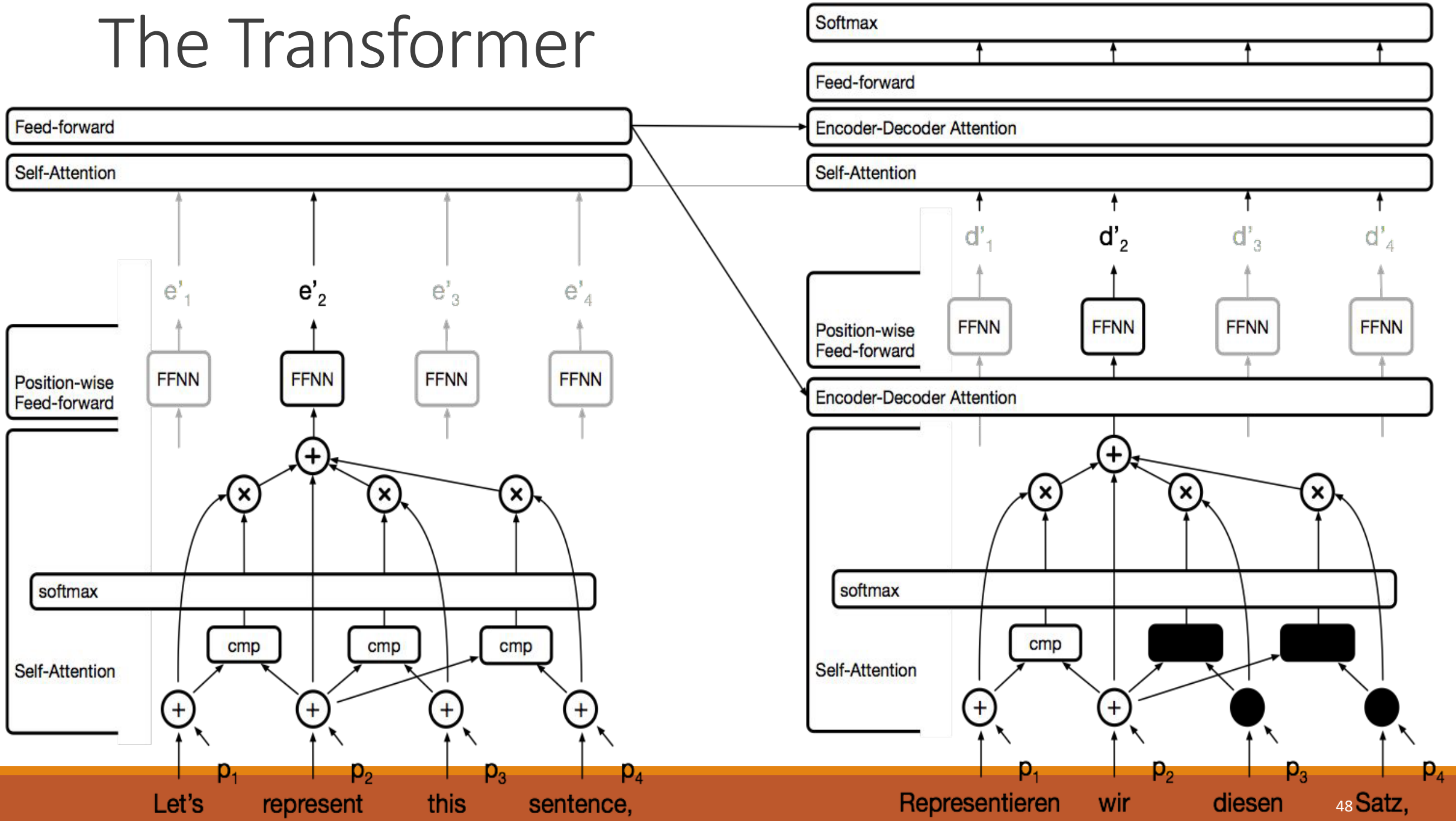
Complete Encoder

- 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)



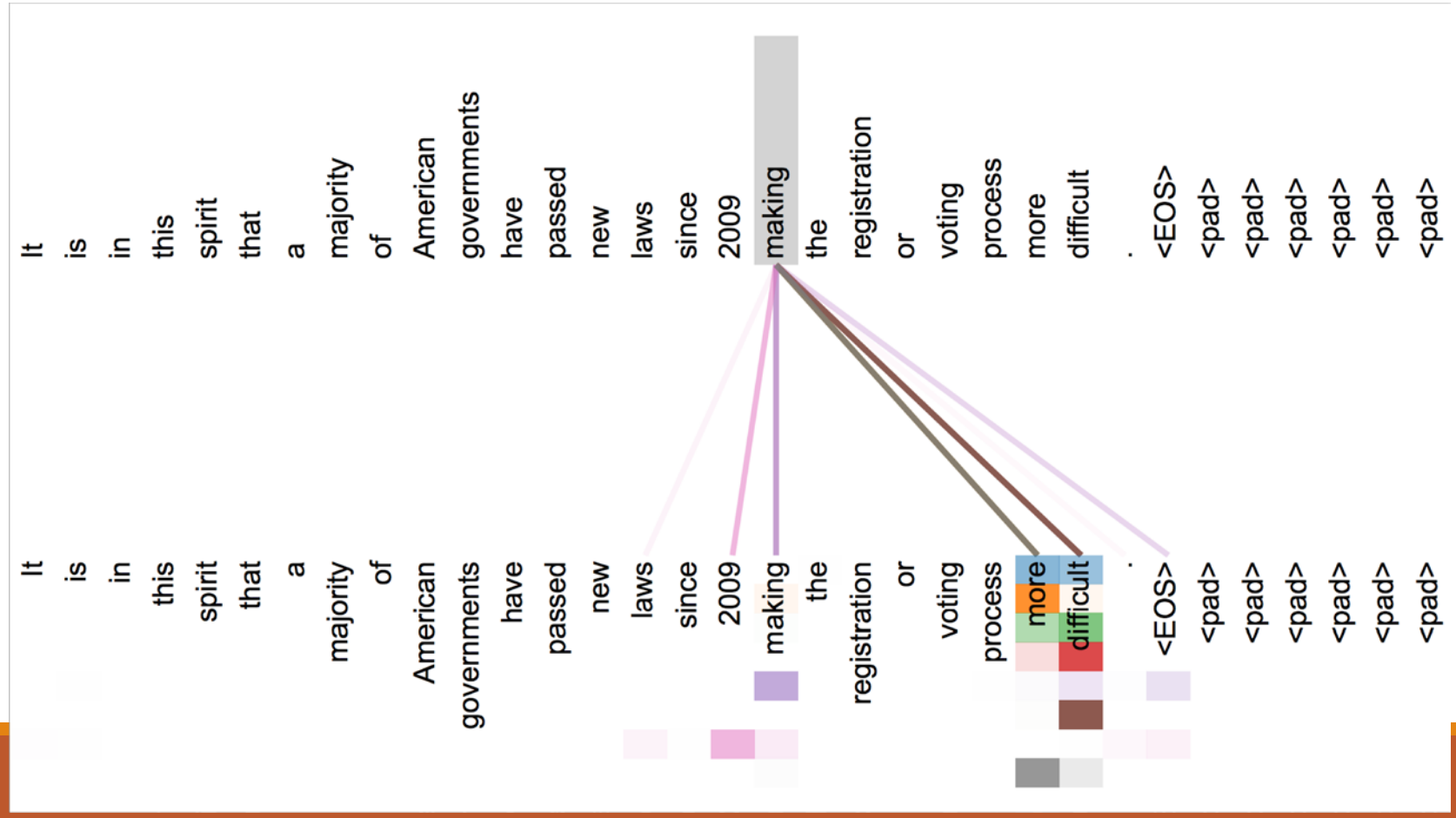


The Transformer

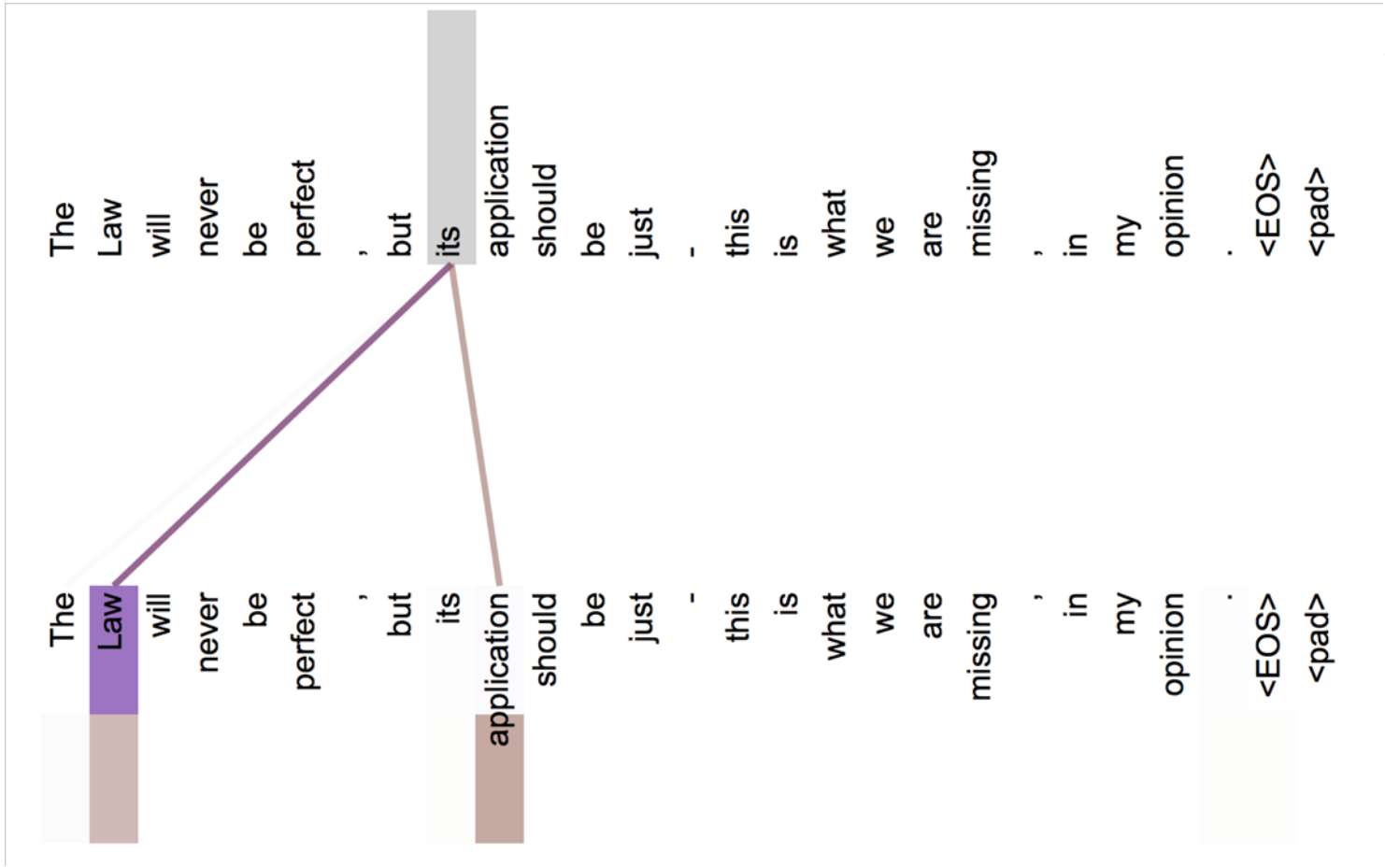


Attention visualization in layer 5

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



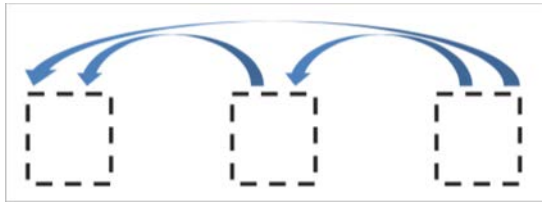
Attention visualization: Implicit anaphora resolution



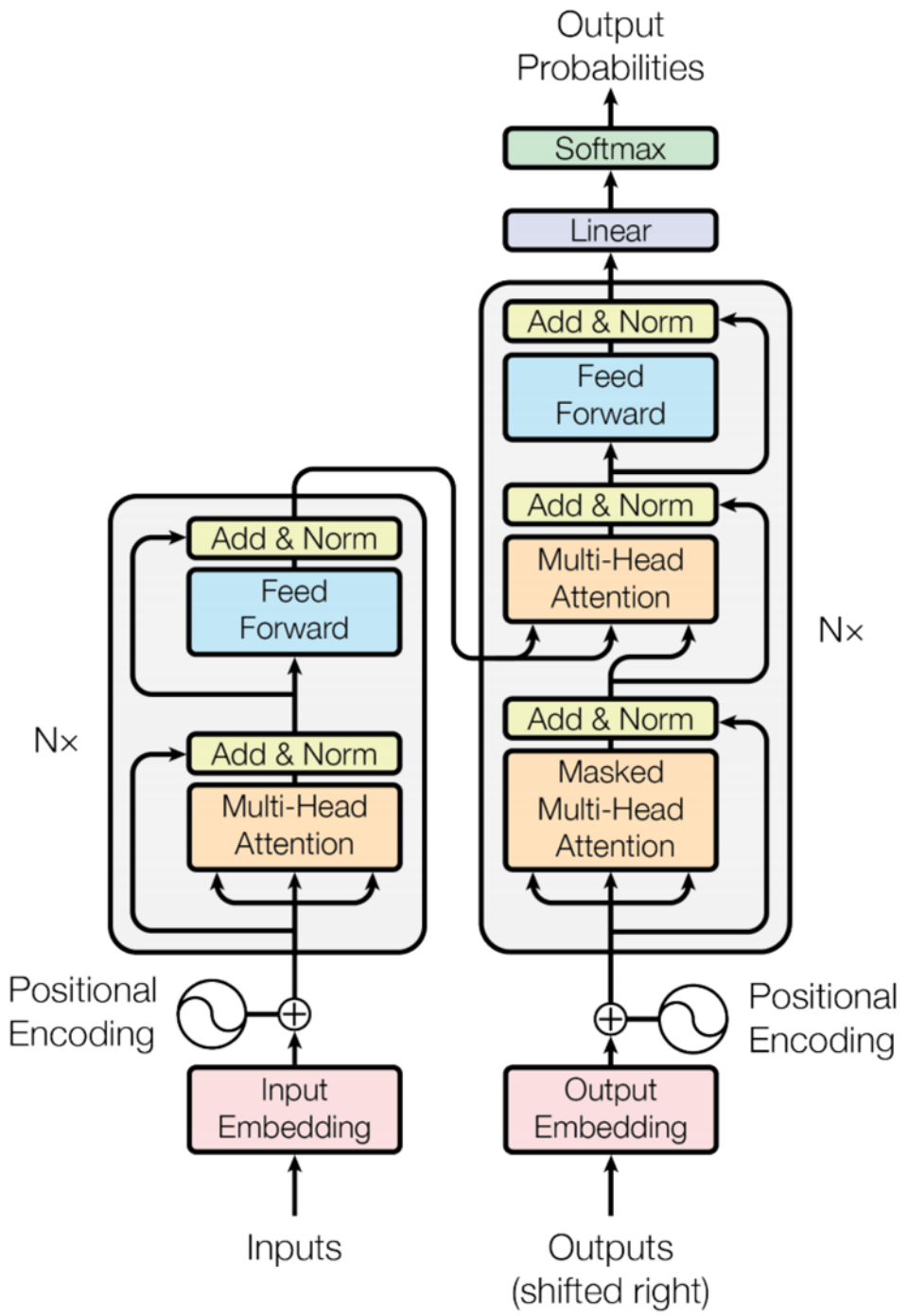
- Note that the attentions are very sharp for this word.

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



Tips and tricks of the Transformer

- Details in paper:
  - Byte-pair encodings
  - Checkpoint averaging
  - ADAM optimizer with learning rate changes
  - Dropout during training at every layer just before adding residual
- 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

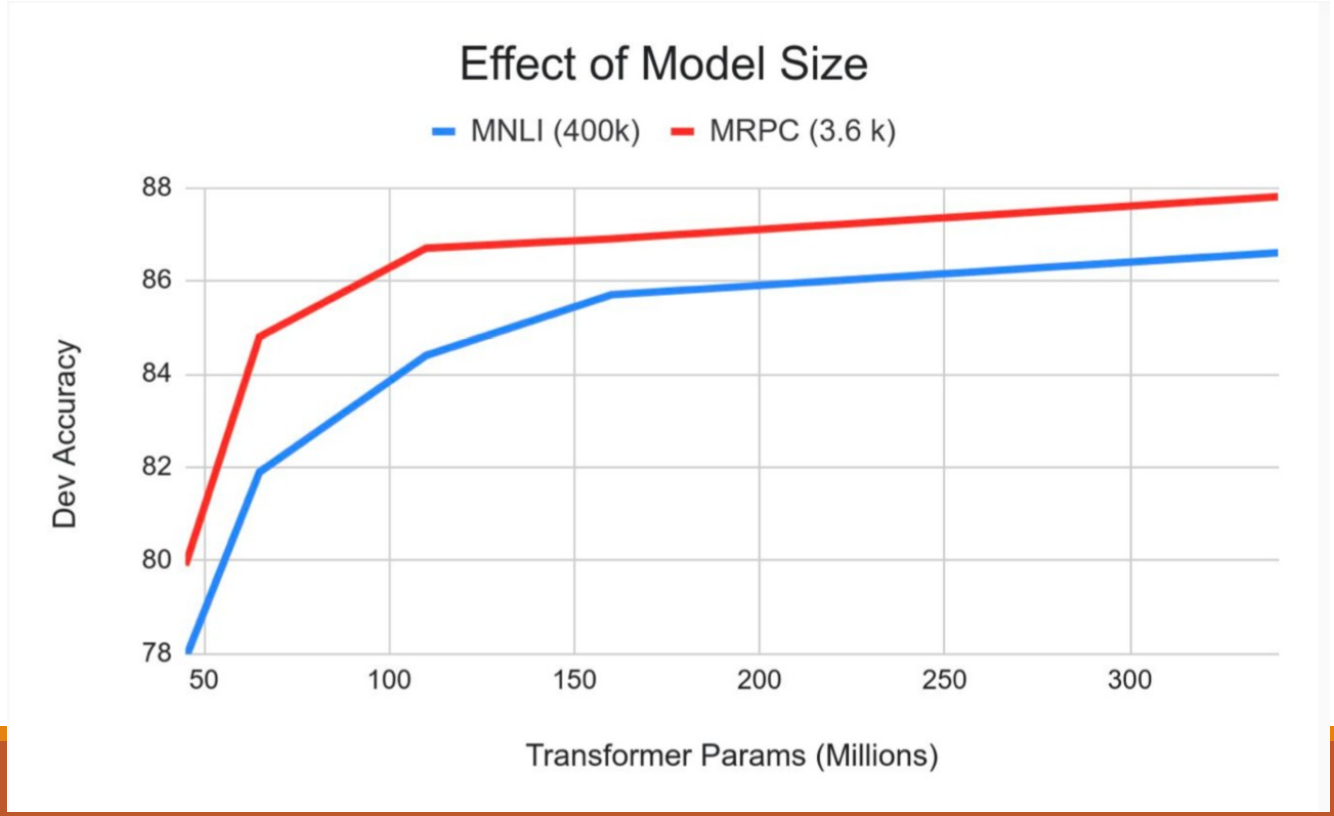
Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
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	**$3.3 \cdot 10^{18}$**	
Transformer (big)	**28.4**	**41.8**	$2.3 \cdot 10^{19}$	

Some performance numbers: LM on WikiText-103

Model	# Params	Perplexity
Grave et al. (2016) – LSTM		48.7
Grave et al. (2016) – LSTM with cache		40.8
4-layer QRNN (Merity et al. 2018)	151M	33.0
LSTM + Hebbian + Cache + MbPA (Rae et al.)	151M	29.2
Transformer-XL Large (Dai et al. 2019)	**257M**	**18.3**
GPT-2 Large* (Radford et al. 2019)	1.5B	17.5

Size matters

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



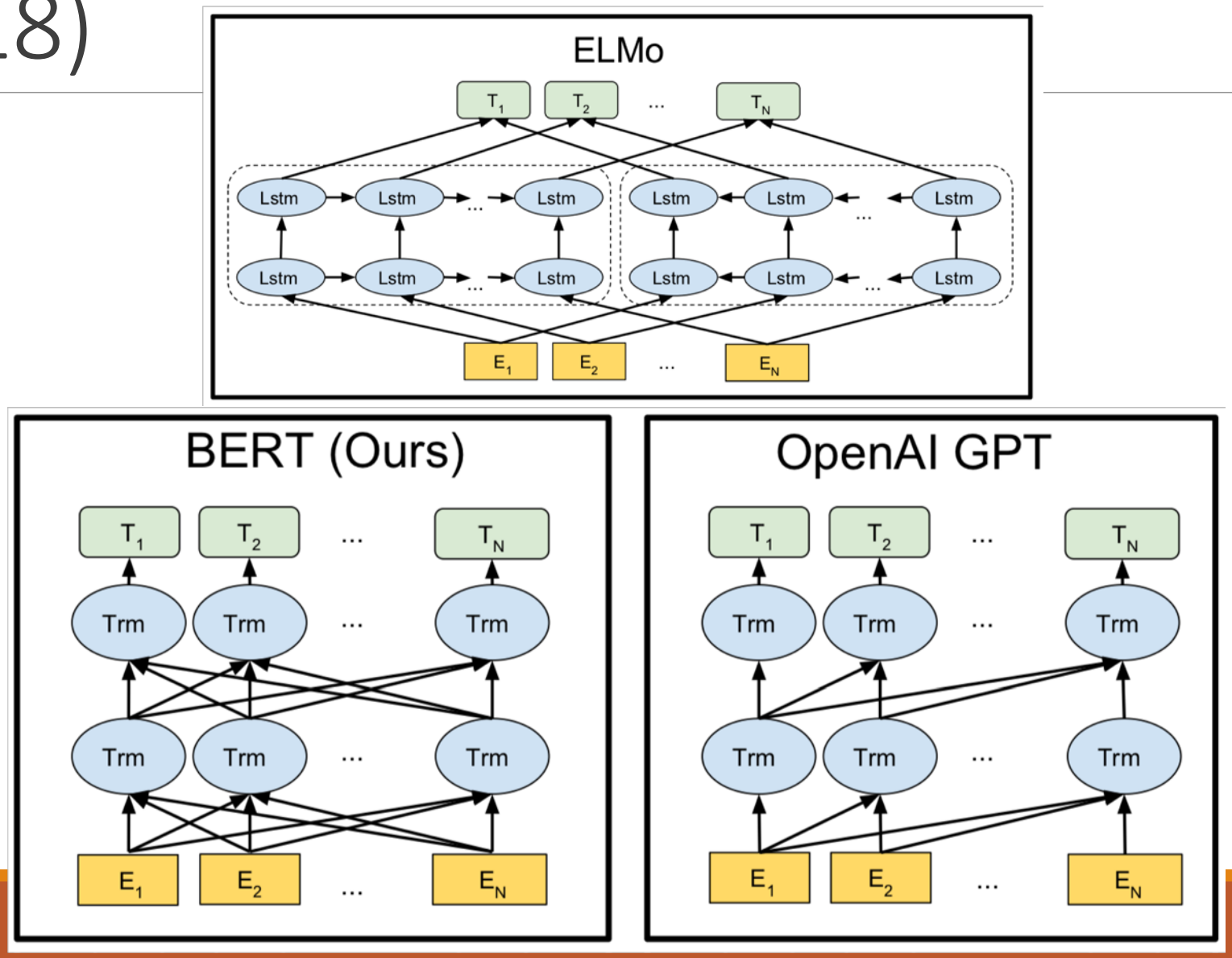
BERT: Devlin, Chang, Lee, Toutanova (2018)

- BERT (Bidirectional Encoder Representations from Transformers):
  - Pre-training of Deep Bidirectional Transformers for Language Understanding
- Want: truly bidirectional information flow without leakage in a deep model
- Solution: Use a cloze task formulation where 15% of words are blanked out and predicted:
  - Too little masking: Too expensive to train
  - Too much masking: Not enough context

store ↑ gallon ↑

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

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



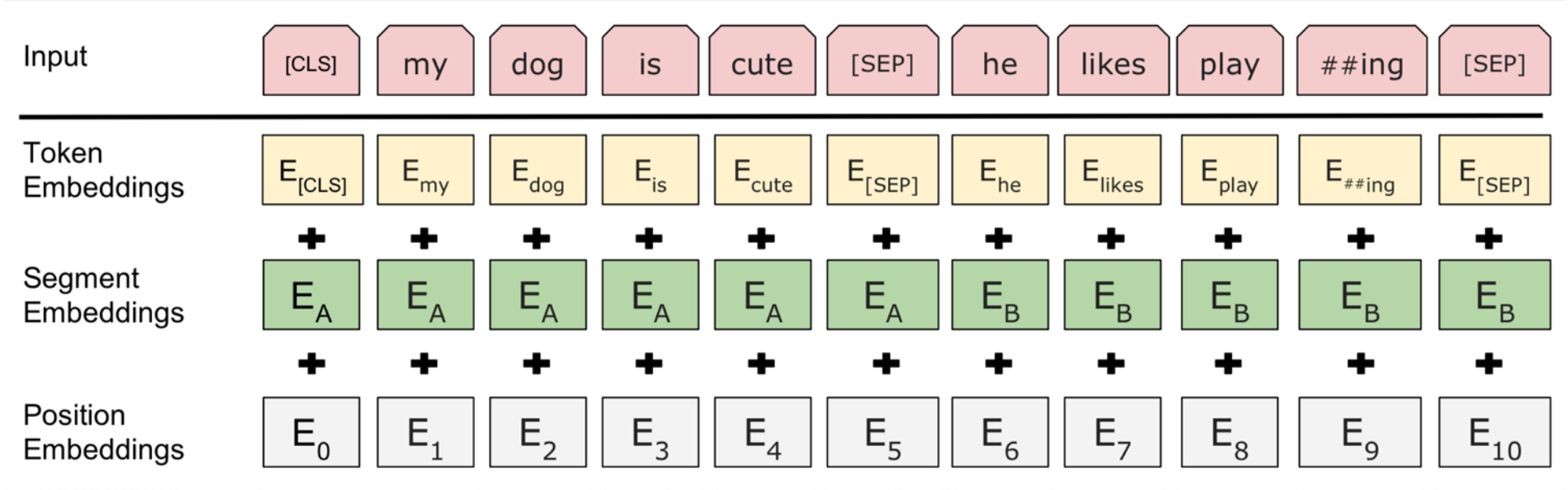
BERT complication: Next sentence prediction

- 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

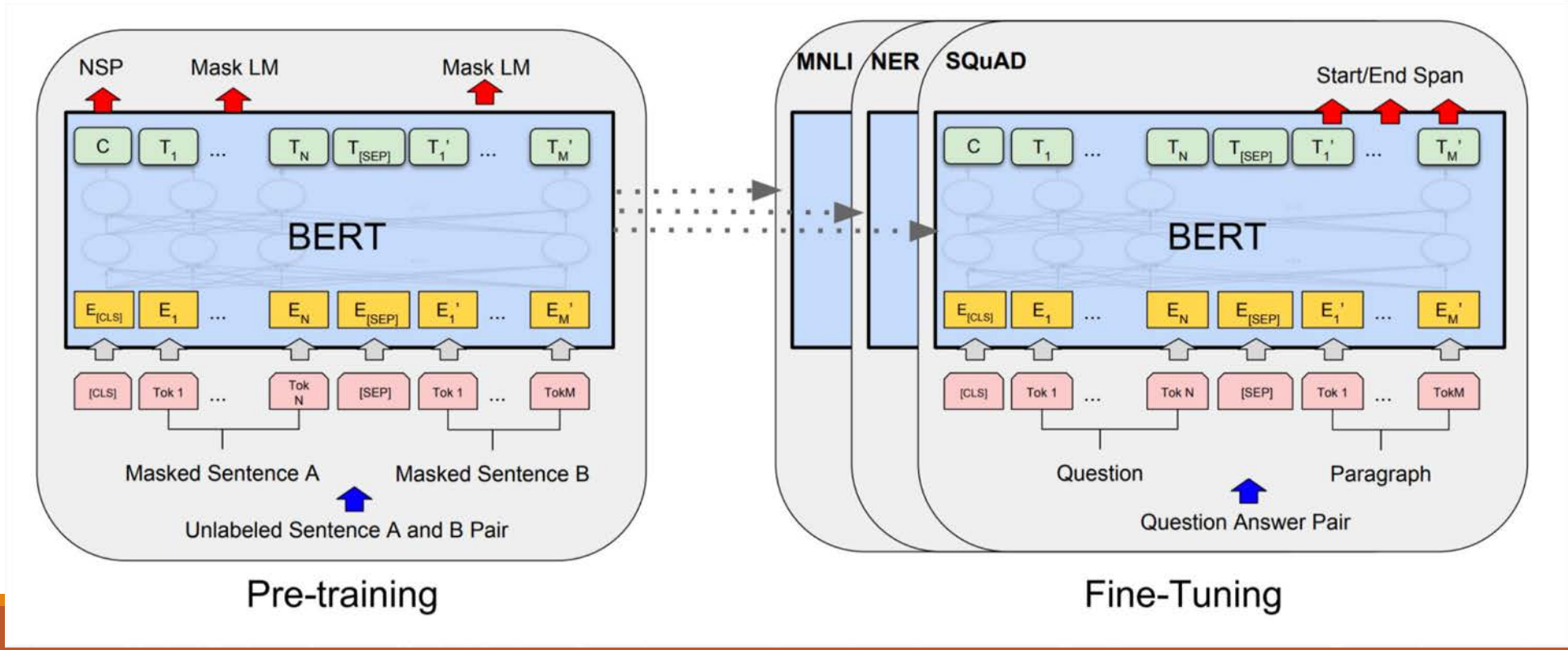
BERT model architecture and training

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

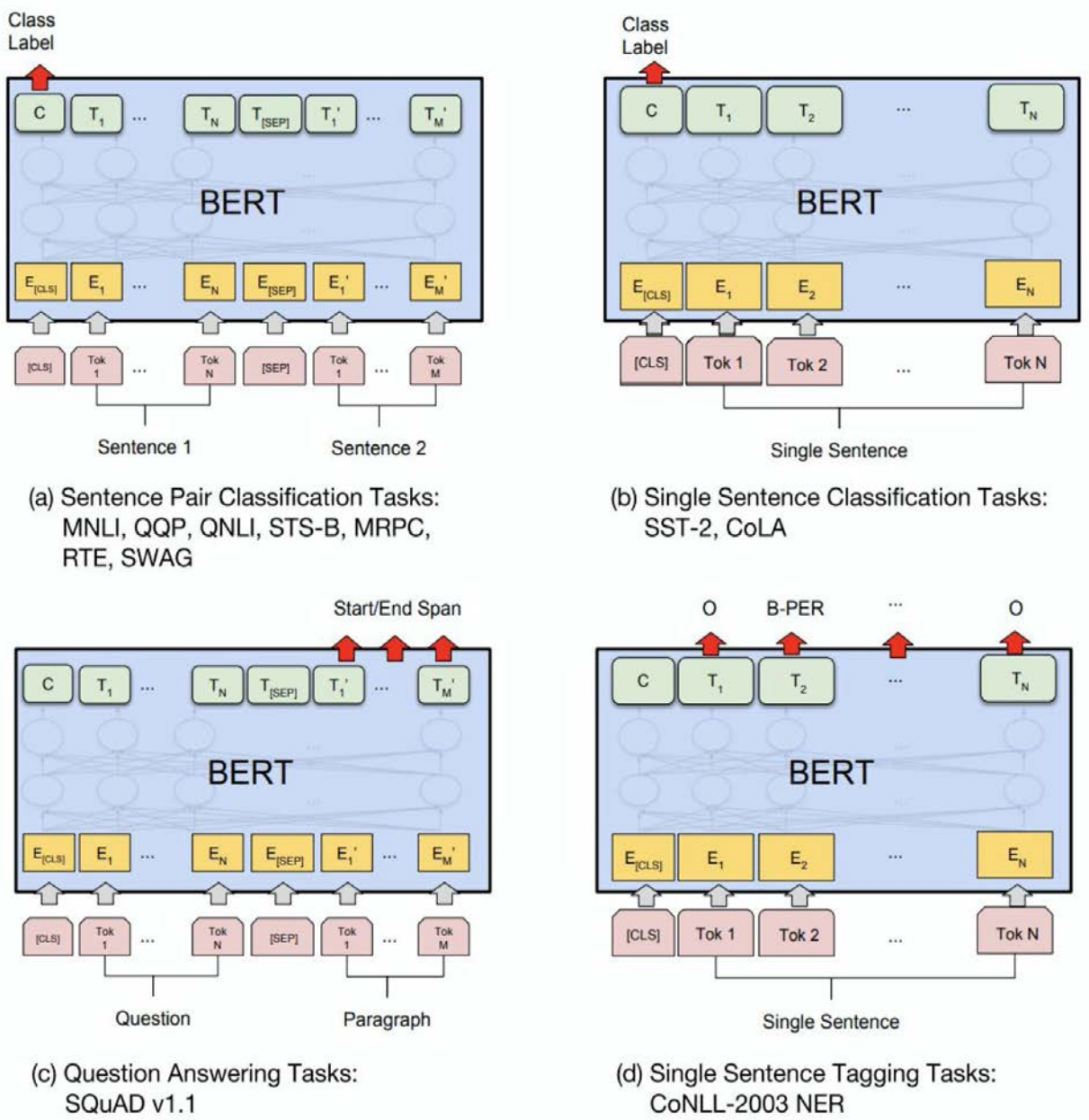
- Train on Wikipedia + BookCorpus
- Train 2 model sizes:
  - BERT-Base: 12-layer, 768-hidden, 12-head
  - BERT-Large: 24-layer, 1024-hidden, 16-head
- Trained on 4x4 or 8x8 TPU slice for 4 days

BERT model fine tuning

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



BERT model fine tuning



CoNLL 2003 Named Entity Recognition (en news testb)

Name	Description	Year	F1
BERT Large	Transformer bidi LM + fine tune	2018	92.8
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

BERT results on GLUE tasks

- GLUE benchmark is dominated by natural language inference tasks, but also has sentence similarity and sentiment
- MNLI: Multi-genre natural language inference
- CoLa
  - Sentence: The wagon rumbled down the road. Label: Acceptable
  - Sentence: The car honked down the road. Label: Unacceptable
- Glue score: the resulting average score of the nine tasks

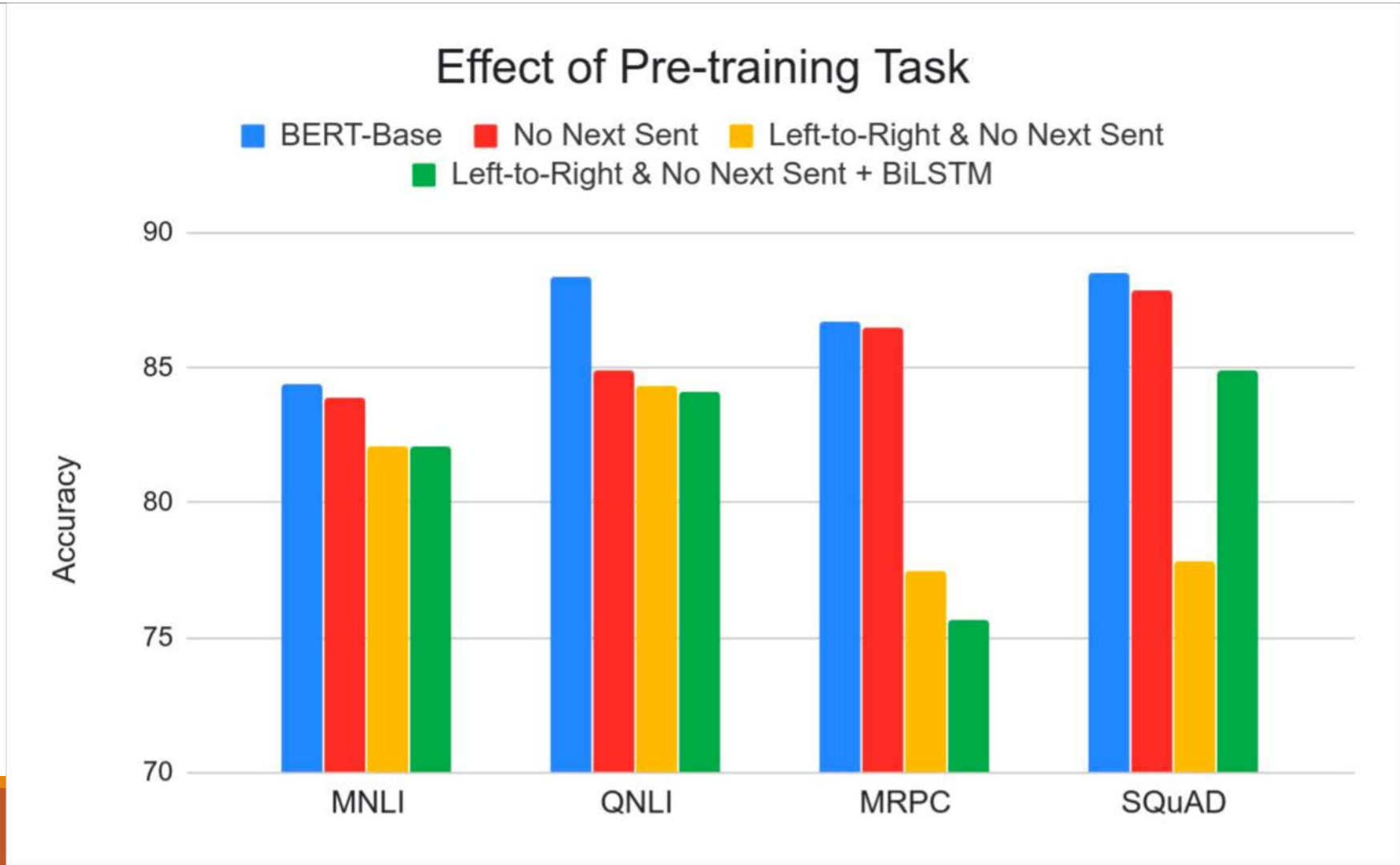
BERT results on GLUE tasks

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**

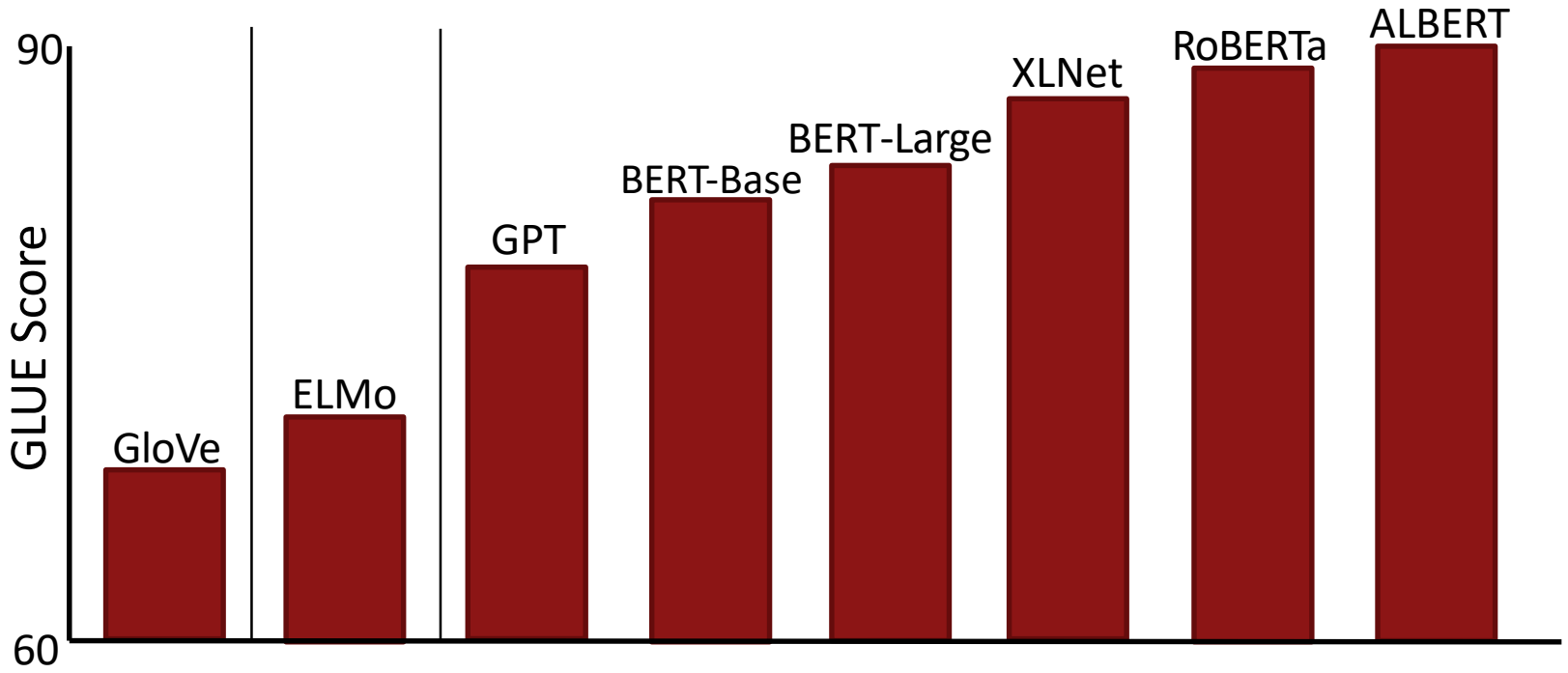
- Quora Question Pairs (QQP), Question NLI (QNLI), Stanford sentiment treebank (SST), Semantic textual similarity (STS), Microsoft Research paraphrase corpus (MRPC), Recognizing textual entailment (RTE)

https://gluebenchmark.com/tasks/

Effect of pre-training task



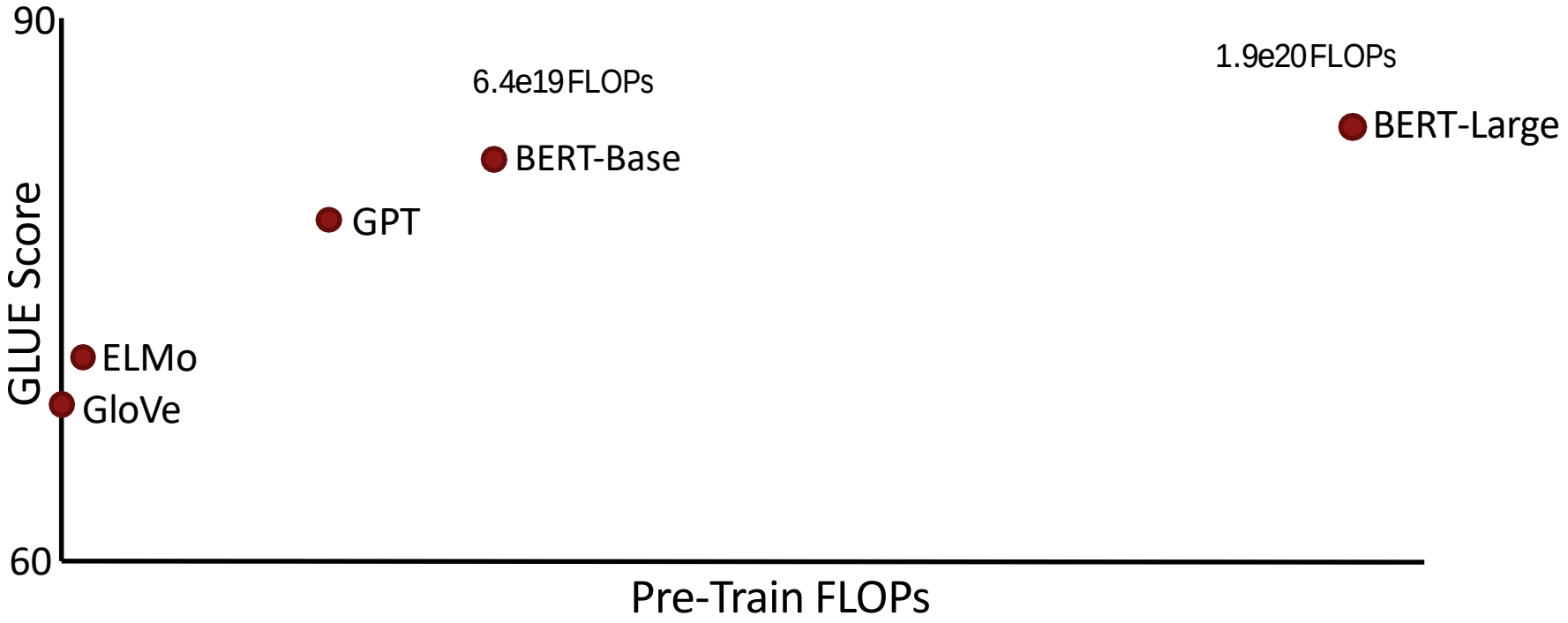
How's the weather?
Rapid Progress from Pre-Training (GLUE benchmark)



Over 3x reduction in error in 2 years, "superhuman" performance

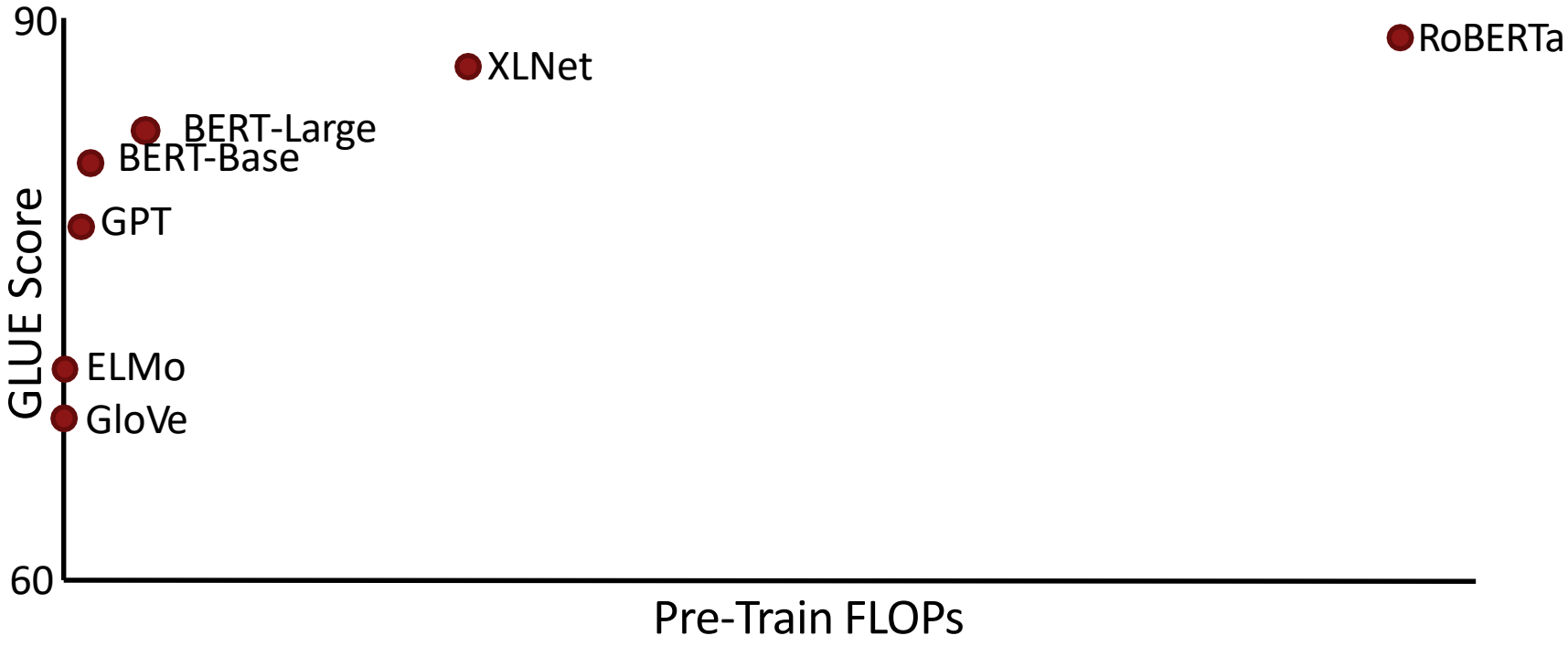
- Yay! We now have strongly performing, deep, generic, pre-trained, neural network stacks for NLP that you can just load
  - in the same way vision has had for 5 years (ResNet, etc.)!

But let's change the x-axis to compute …



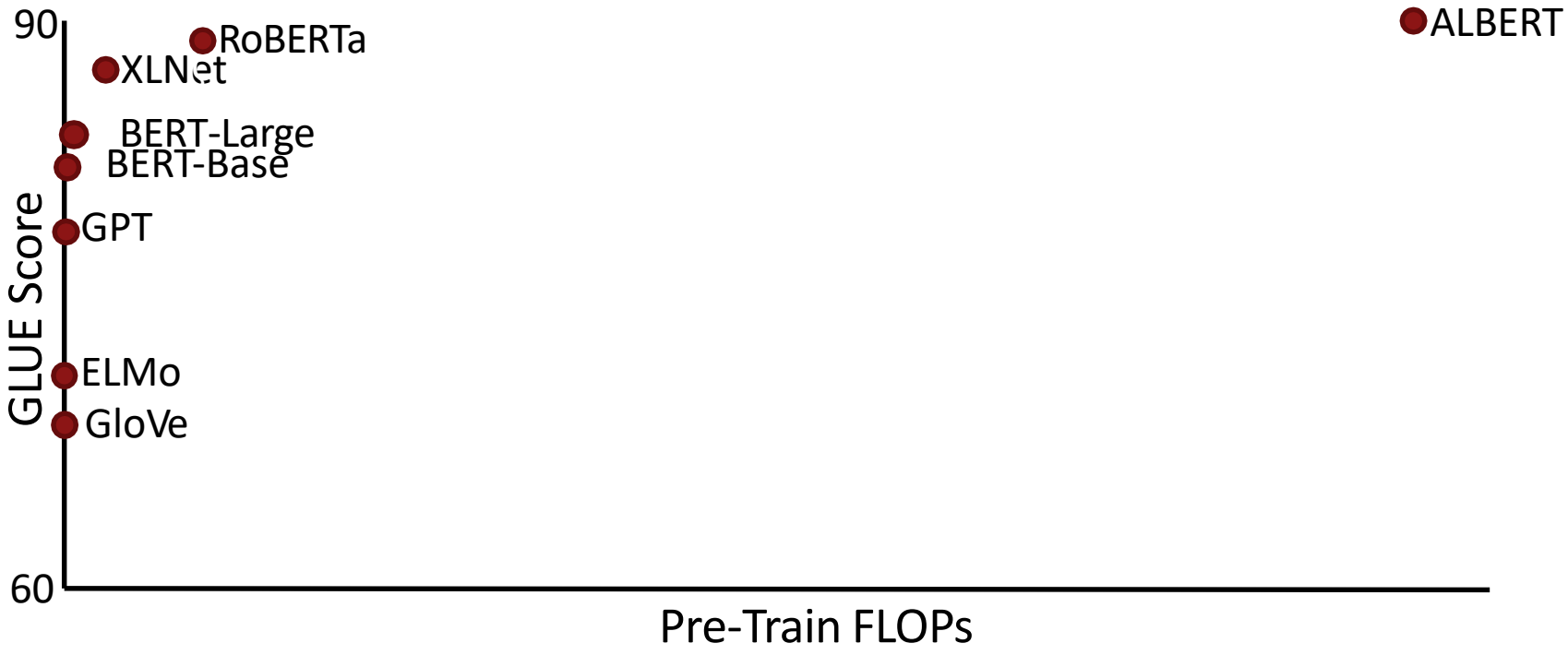
BERT-Large uses 60x more compute than ELMo

But let's change the x-axis to compute …



RoBERTa uses 16x more compute than BERT-Large

More compute, more better…



ALBERT uses 10x more compute than RoBERTa