CMPT 825: Natural Language Processing

Contextualized Word Embeddings and Transformers

Fall 2020

2020-10-28

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

Announcements

- HW3 programming portion due tonight (10/28)
- HW3 conceptual questions due tomorrow (10/29)
  - Please upload using Gradescope (https://www.gradescope.ca/) and mark pages for each question
- Feedback on project proposals later this week/early next week
- Reminder: Participation is 3% of your final grade

Course Logistics

Remaining lectures (tentative)

- Contextual word embeddings and Transformers

- Parsing:
  - Dependency Parsing
  - Constituency Parsing
  - Semantic Parsing

- CNNs for NLP

- Applications: Question Answering, Dialogue, Coreference, Grounding

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 ...

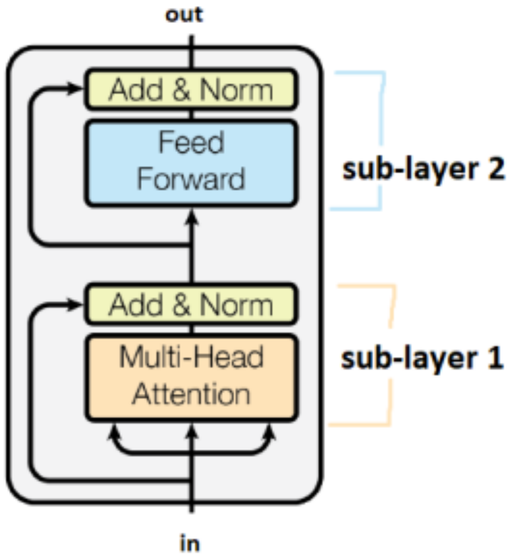
https://arxiv.org › cs ▾

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

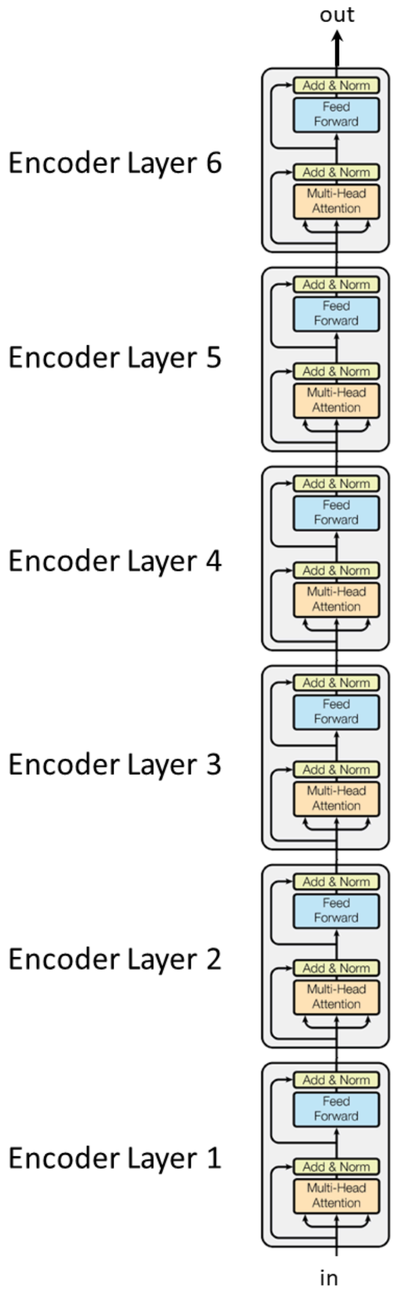


Attention Is All You Need

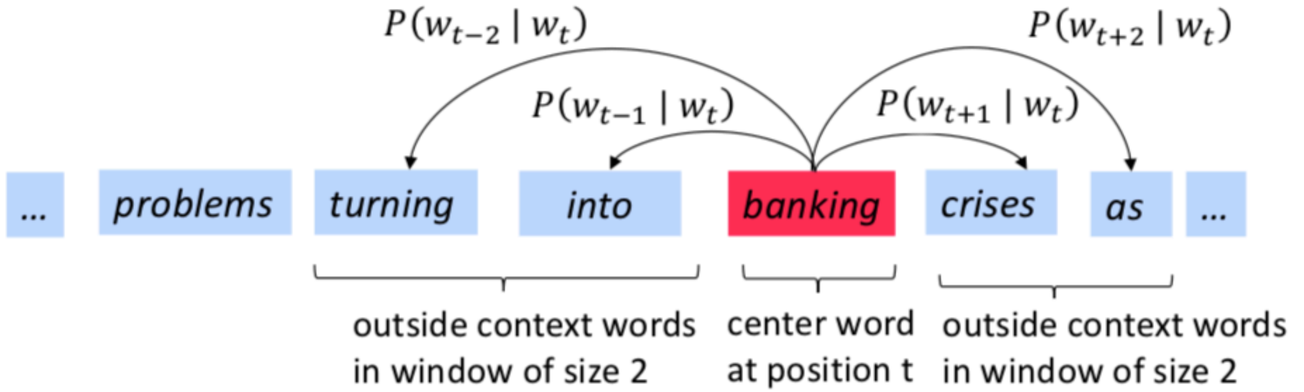
https://arxiv.org › cs ▾

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

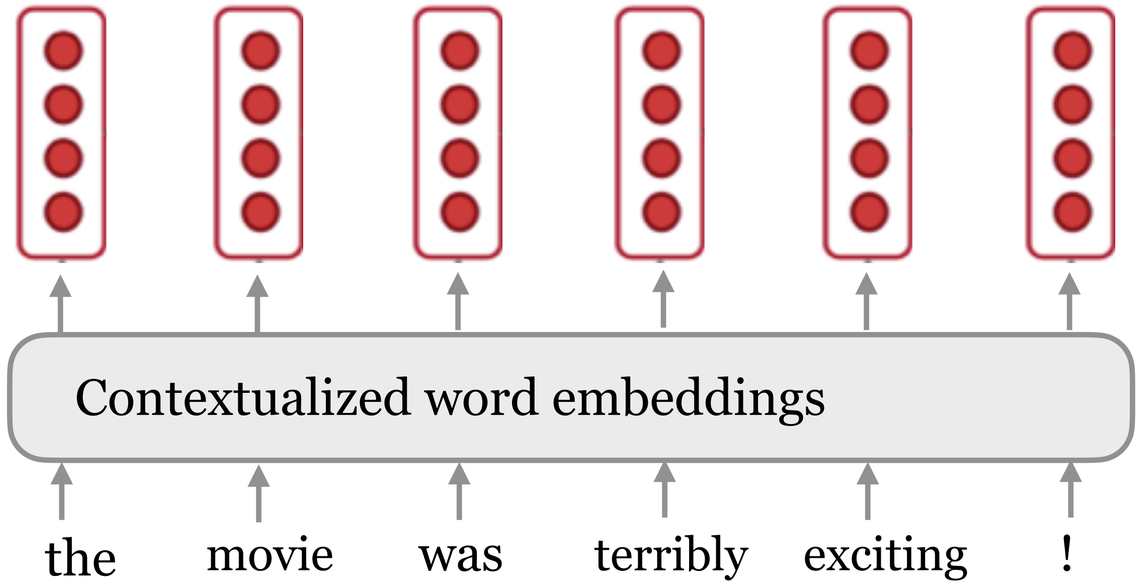
$$v(\text{bank}) = \begin{pmatrix} -0.224 \\ 0.130 \\ -0.290 \\ 0.276 \end{pmatrix}$$

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

different senses

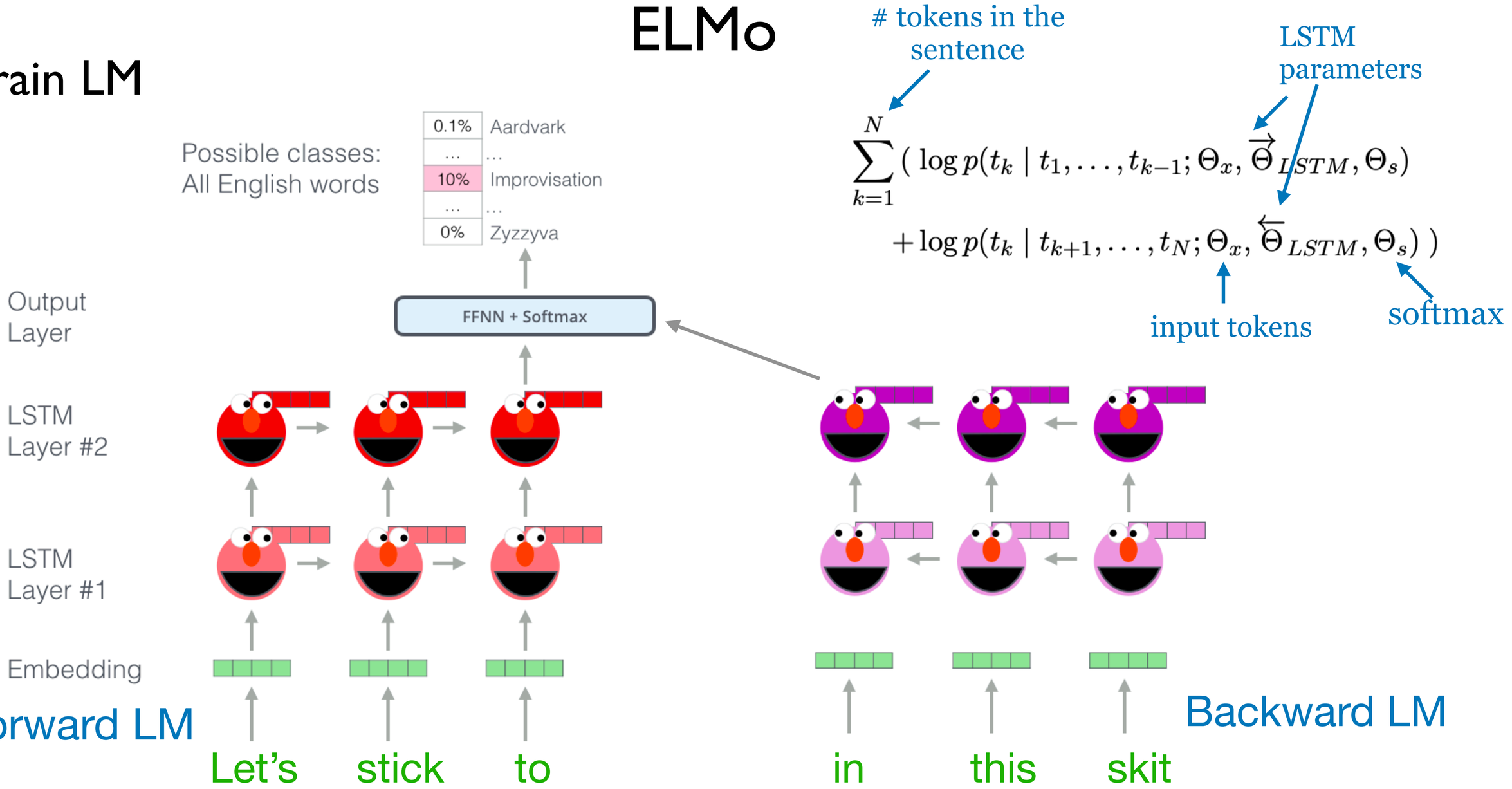
(Peters et al, 2018): Deep contextualized word representations

ELMo

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

Pretrain LM

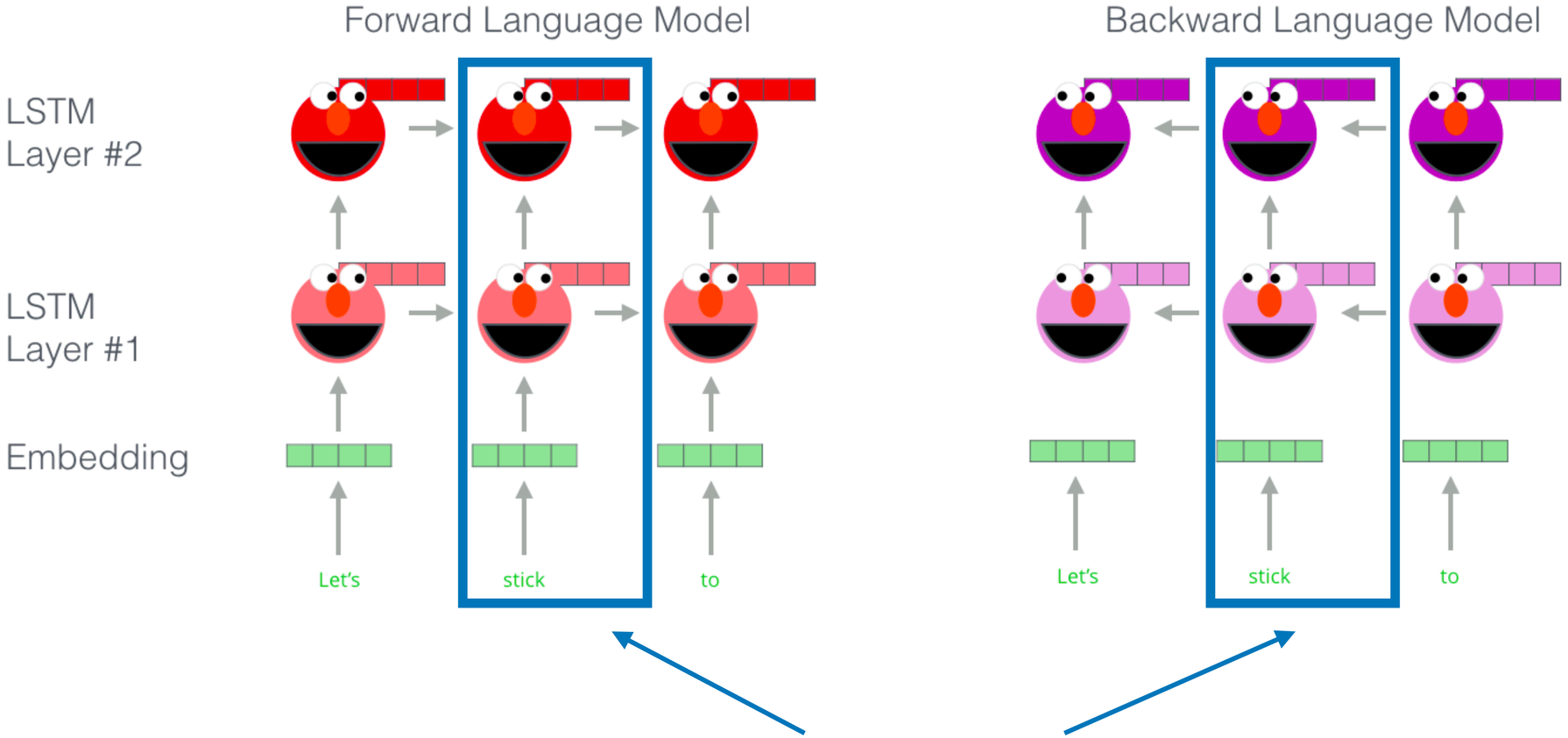
ELMo



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

ELMo

After training LM

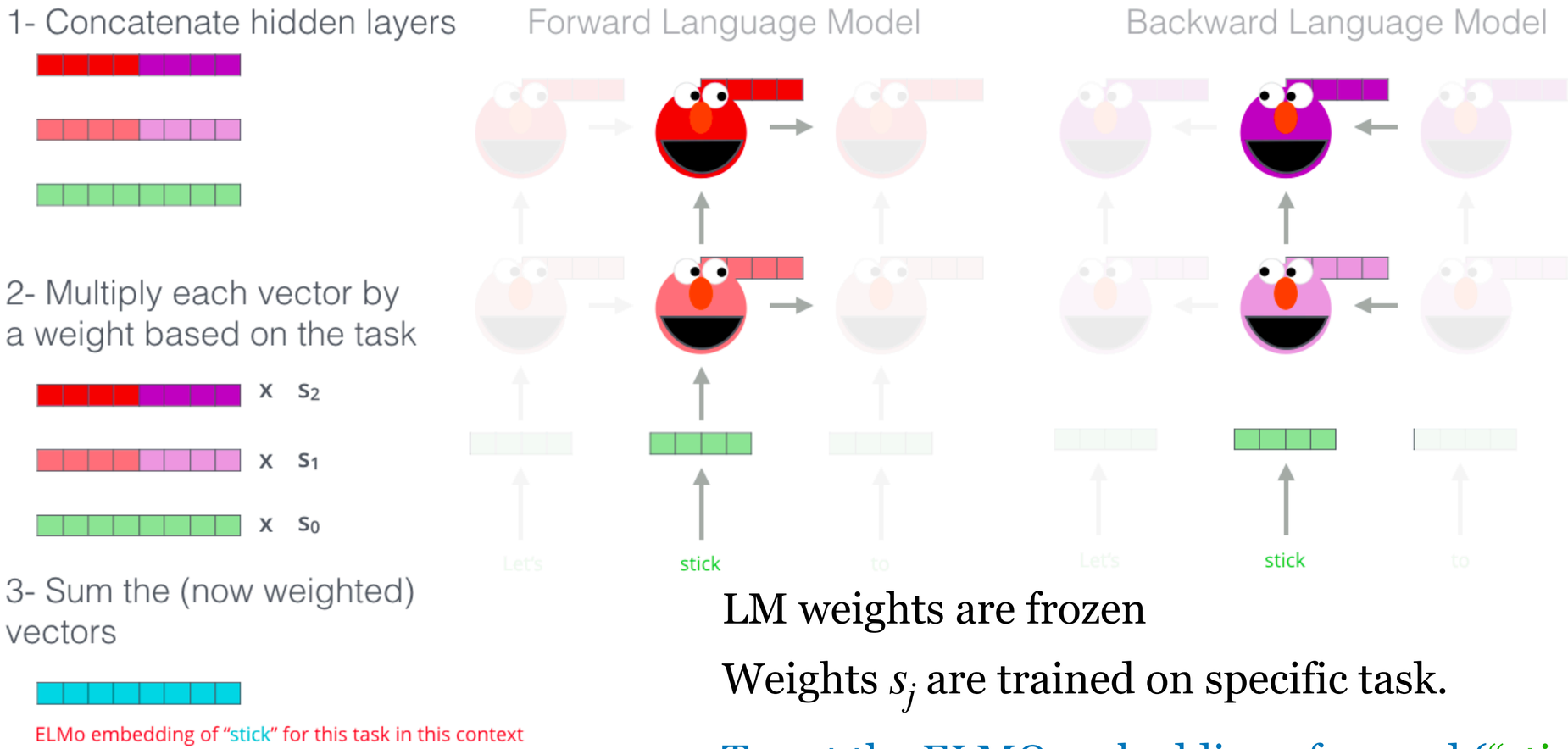


To get the ELMO embedding of a word ("stick"):

Concatenate forward and backward embeddings and take weighted sum of layers

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

ELMo



LM weights are frozen

Weights s_j are trained on specific task.

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

Concatenate forward and backward embeddings and take weighted sum of layers

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

Summary: How to get ELMo embedding?

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

← L is # of layers

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

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

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

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

(could also insert into higher layers)

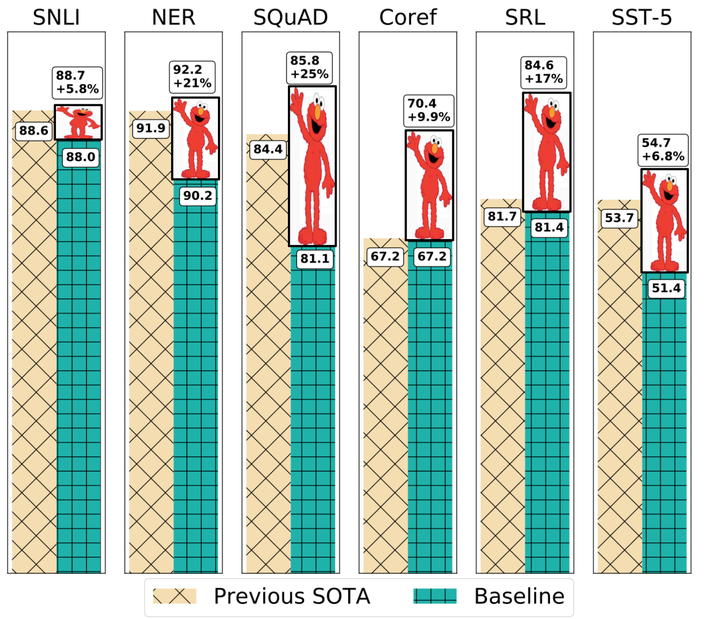
More details

- Forward and backward LMs: 2 layers each
- Use character CNN to build initial word representation
  - 2048 char n-gram filters and 2 highway layers, 512 dim projection
- User 4096 dim hidden/cell LSTM states with 512 dim projections to next input
- A residual connection from the first to second layer
- Trained 10 epochs on 1B Word Benchmark

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



Tasks

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

Tasks

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

Passage

In meteorology, precipitation is any product of the condensation of atmospheric water vapor that falls under **gravity**. The main forms of precipitation include drizzle, rain, sleet, snow, **graupel** and hail... Precipitation forms as smaller droplets coalesce via collision with other rain drops or ice crystals **within a cloud**. Short, intense periods of rain in scattered locations are called "showers".

Question / Answer

What causes precipitation to fall?
gravity

What is another main form of precipitation besides drizzle, rain, snow, sleet and hail?
graupel

Where do water droplets collide with ice crystals to form precipitation?
within a cloud

SQuAD: 100,000+ Questions for Machine Comprehension of Text
Rajpurkar et al, EMNLP 2016

Tasks

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

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

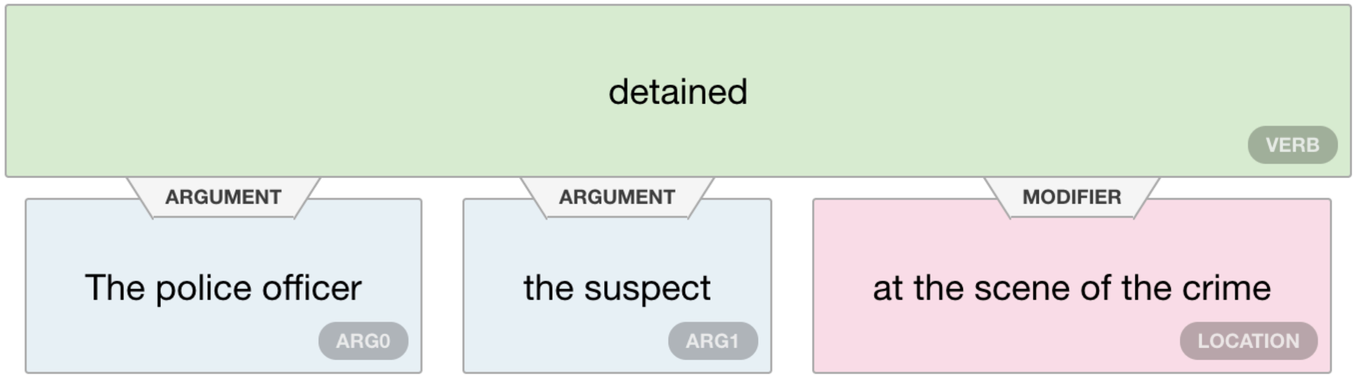
A large annotated corpus for learning natural language inference
Bowman et al, EMNLP 2015

Tasks

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

- Different types of frames
  - FrameNet vs PropBank
- Different semantics roles depending on the verb

The police officer detained the suspect at the scene of the crime

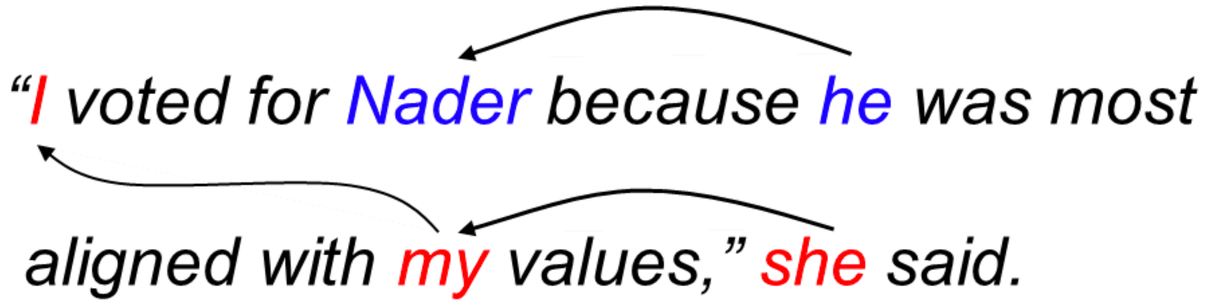


(figure credit: https://demo.allennlp.org/semantic-role-labeling/)

Automatic Labeling of Semantic Roles
Gildea and Jurafsky, CL 2002

Tasks

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

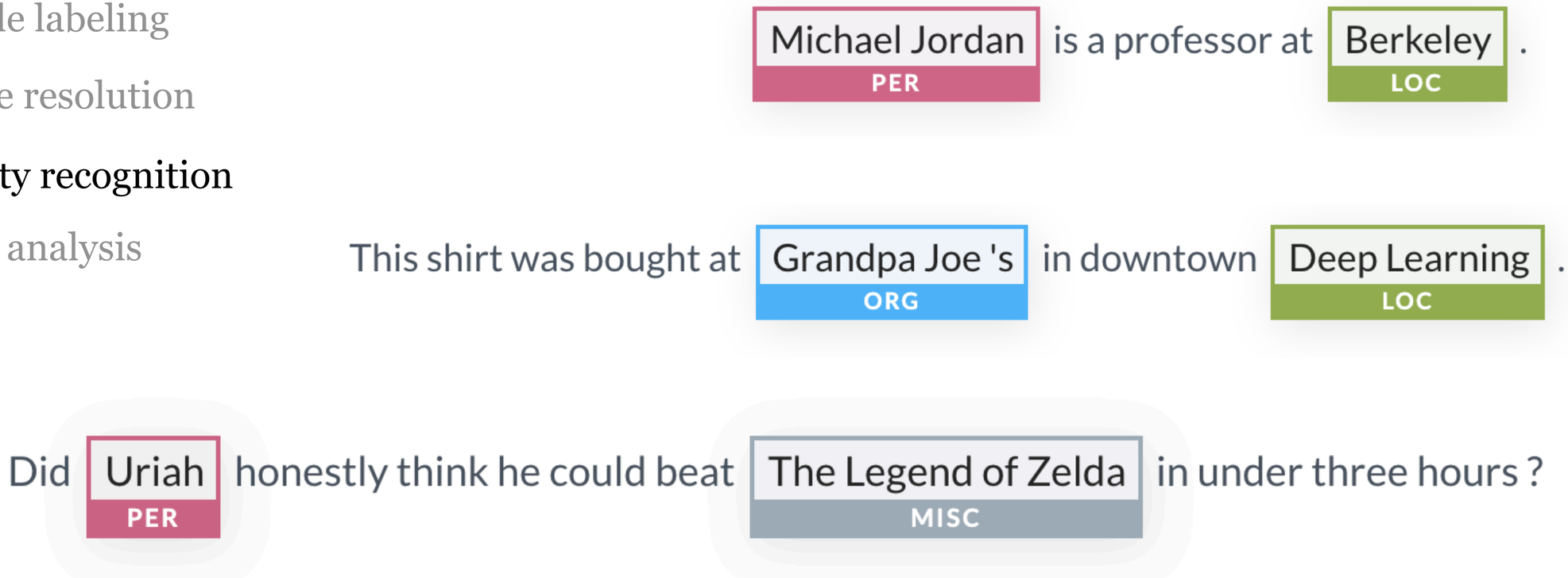


(figure credit: https://nlp.stanford.edu/projects/coref.shtml)

Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition.
Pradhan et al, CONLL Shared Task 2012

Tasks

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



(figure credit: https://demo.allennlp.org/named-entity-recognition)

A large annotated corpus for learning natural language inference
Sang and Meudler, CoNLL Shared Task 2002

Tasks

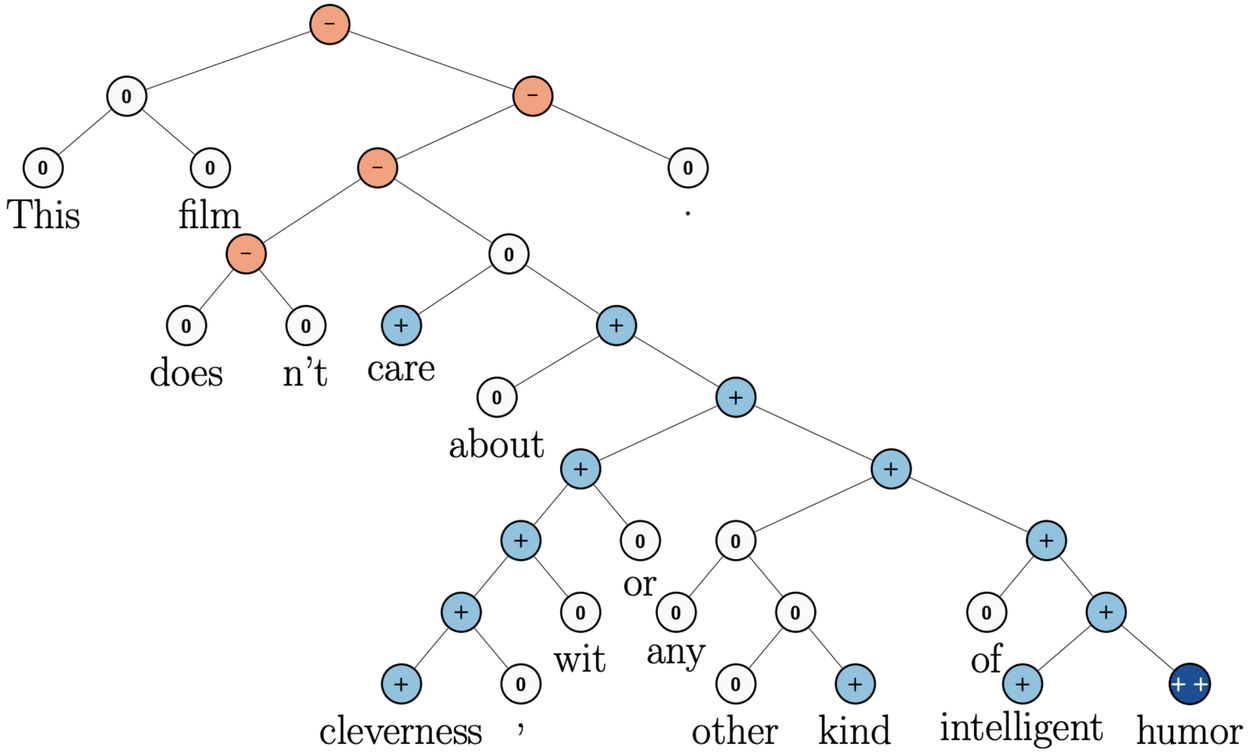
- SQuAD: question answering
- SNLI: natural language inference
- SRL: semantic role labeling
- Coref: coreference resolution
- NER: named entity recognition
- SST-5: sentiment analysis

5 classes

- Very Positive
- Positive
- Neutral
- Negative
- Very Negative

Stanford Sentiment Treebank

This file doesn't care about cleverness, wit or any other kind of intelligent humor.

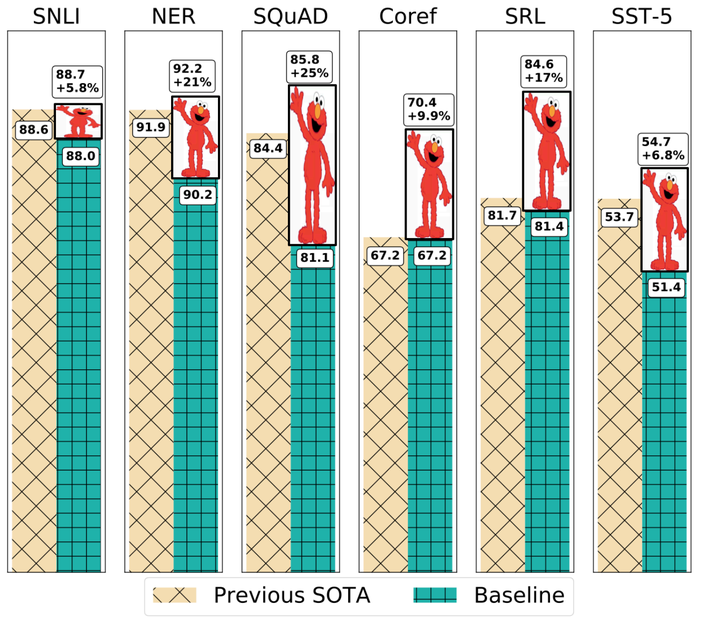


Recursive deep models for semantic compositionality over a sentiment treebank
Socher et al, EMNLP 2013

Experimental results

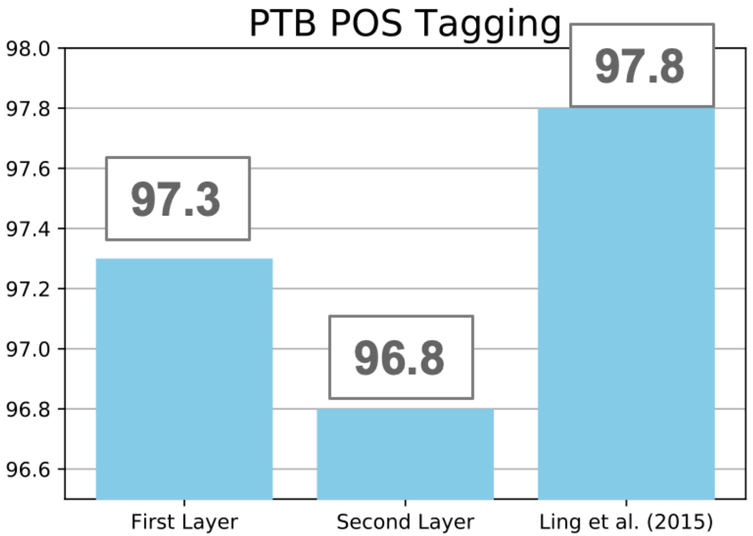
Task	Previous SOTA		Our baseline	ELMo + baseline	Increase (absolute/ relative)
SQuAD	Liu et al. (2017)	84.4	81.1	85.8	4.7 / 24.9%
SNLI	Chen et al. (2017)	88.6	88.0	88.7 ± 0.17	0.7 / 5.8%
SRL	He et al. (2017)	81.7	81.4	84.6	3.2 / 17.2%
Coref	Lee et al. (2017)	67.2	67.2	70.4	3.2 / 9.8%
NER	Peters et al. (2017)	91.93 ± 0.19	90.15	92.22 ± 0.10	2.06 / 21%
SST-5	McCann et al. (2017)	53.7	51.4	54.7 ± 0.5	3.3 / 6.8%

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



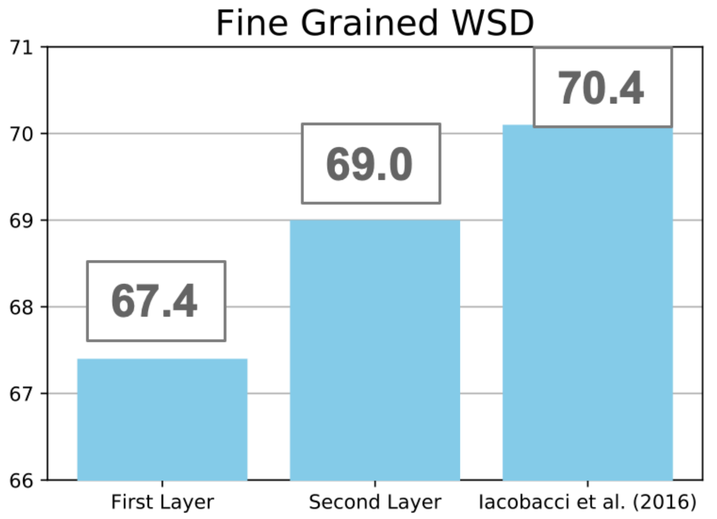
Intrinsic Evaluation

syntactic information



First Layer > Second Layer

semantic information



Second Layer > First Layer

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

Use ELMo in practice

https://allennlp.org/elmo

Pre-trained ELMo Models

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

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

options_file = "https://allennlp.s3.amazonaws.com/models/elmo/2x409
weight_file = "https://allennlp.s3.amazonaws.com/models/elmo/2x4096_

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

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

embeddings = elmo(character_ids)
```

Also available in TensorFlow

BERT

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

How is BERT different from ELMo?

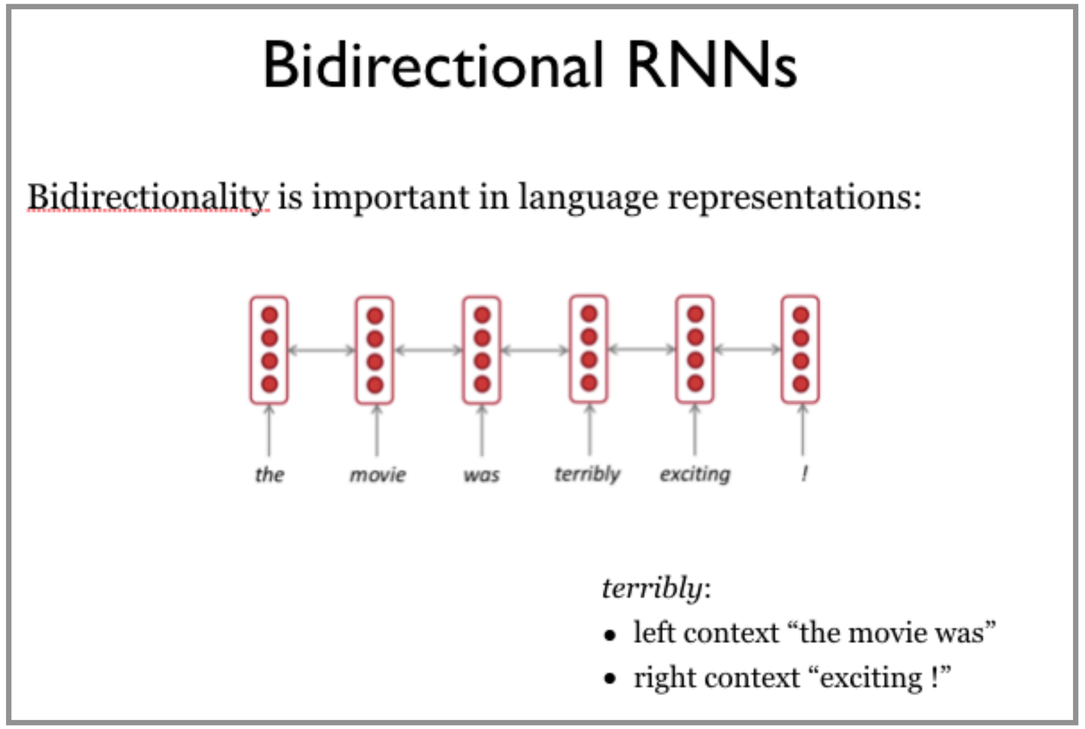
#1. Two unidirectional context vs bidirectional context

#2. LSTMs vs Transformers (will explain more later)

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

Bidirectional encoders

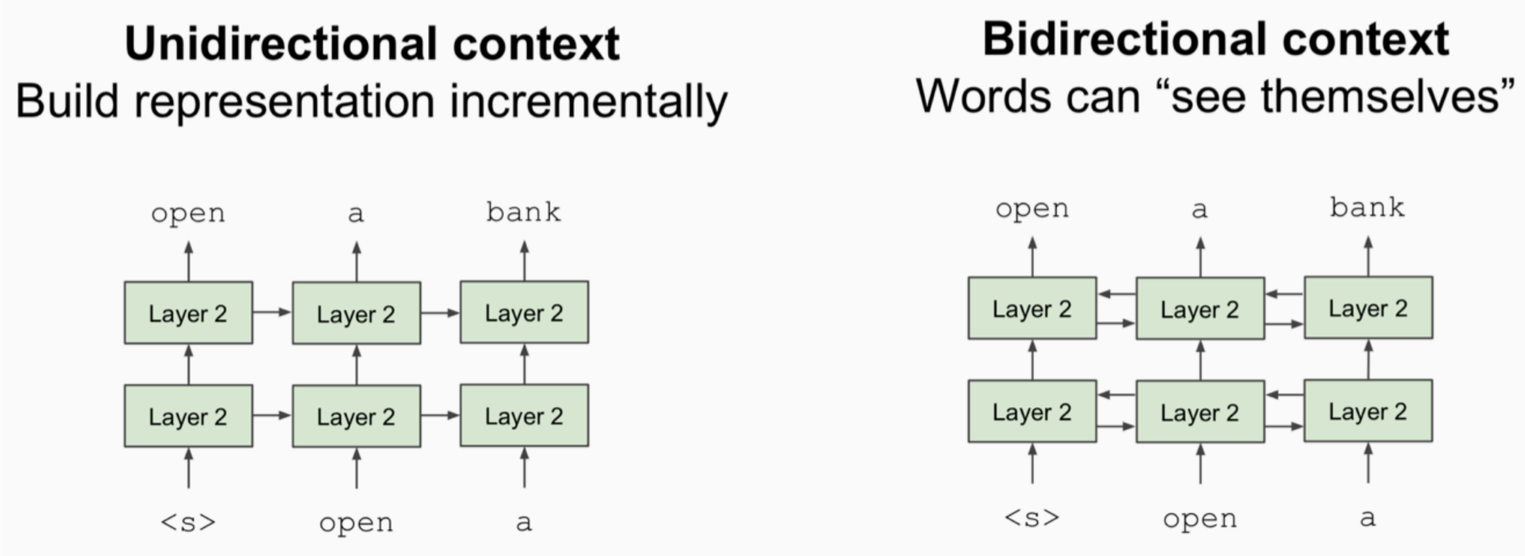
- Language models only use left context or right context (although ELMo used two independent LMs from each direction).
- Language understanding is bidirectional



Why are LMs unidirectional?

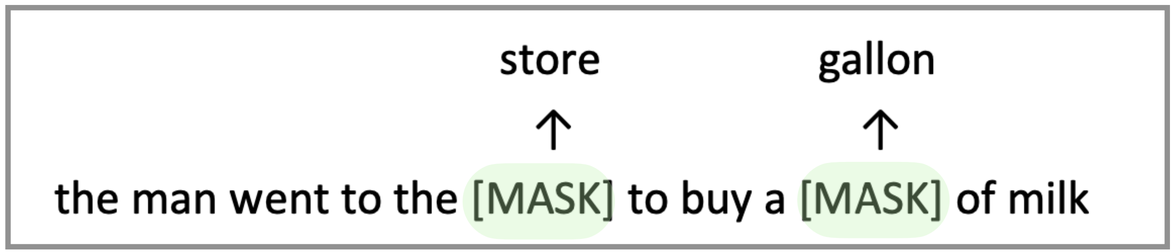
Bidirectional encoders

- 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



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

Masked language models (MLMs)

A little more complex

(don't always replace with `[MASK]`):

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

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

  `my dog is [MASK]`

- 10% of time: replace word with random word

  `my dog is apple`

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

  `my dog is hairy`

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

Next sentence prediction (NSP)

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

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

`he bought a gallon [MASK] milk [SEP]`

Label = `IsNext`

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

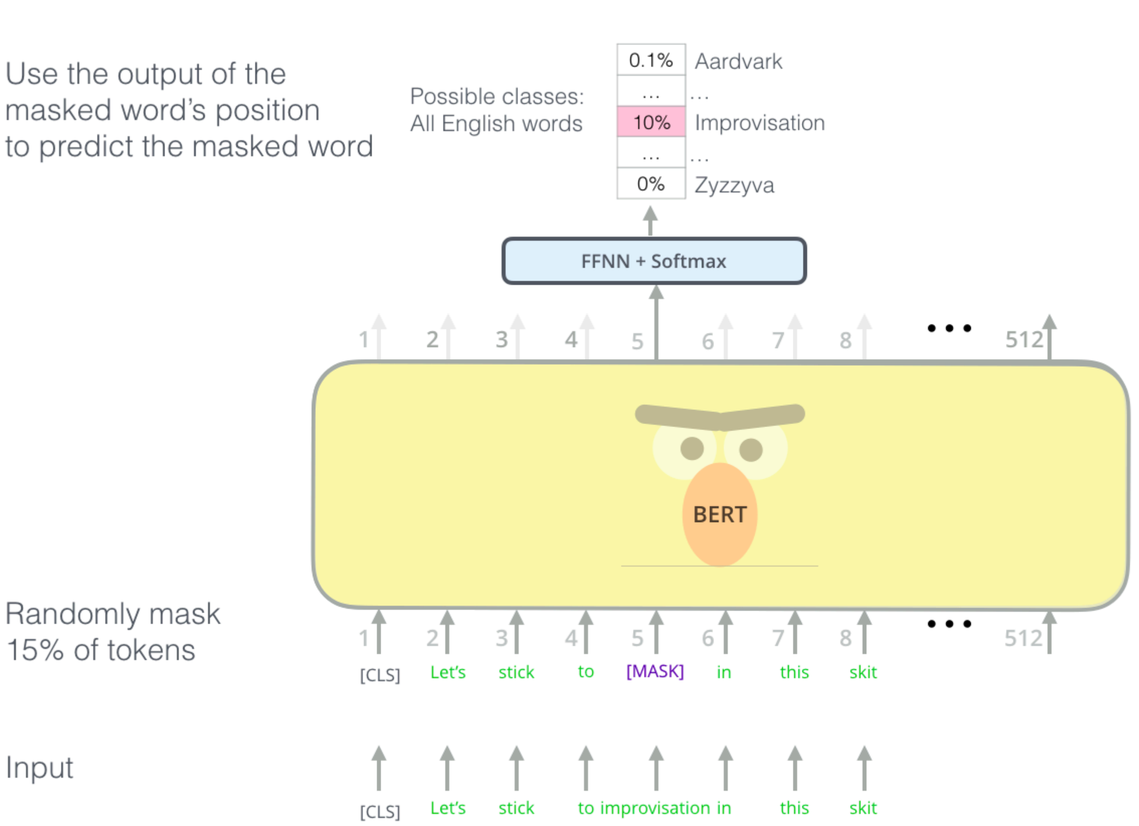
`penguin [MASK] are flight ##less birds [SEP]`

Label = `NotNext`

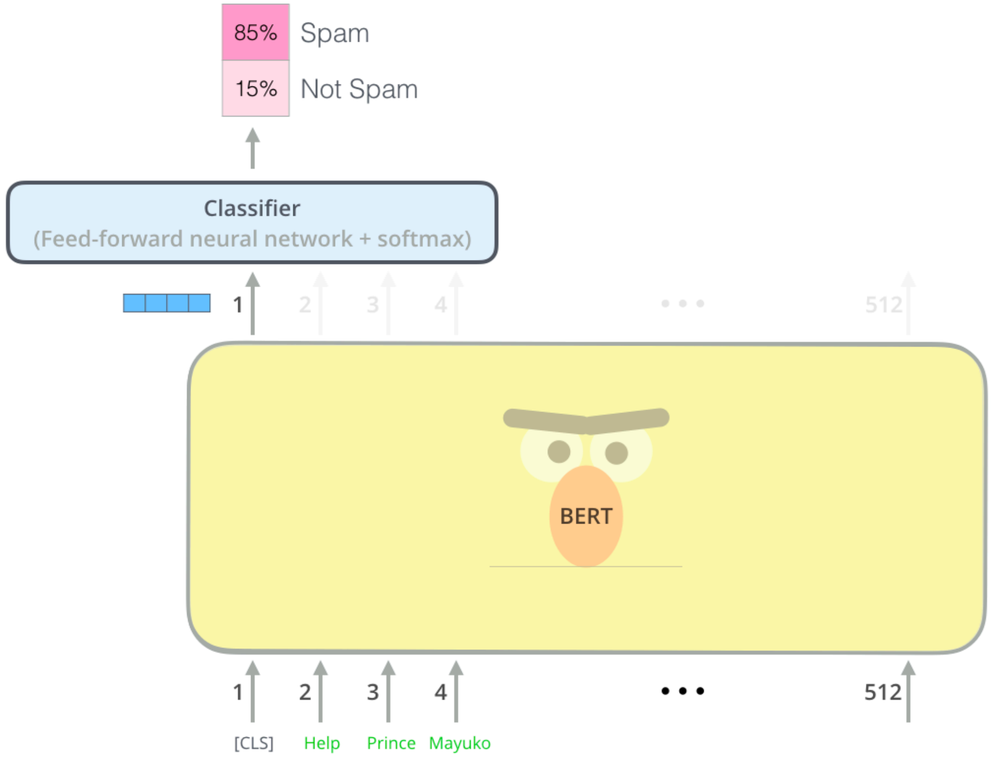
Recent papers show that NSP is not necessary…

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

Pre-training and fine-tuning



Pre-training

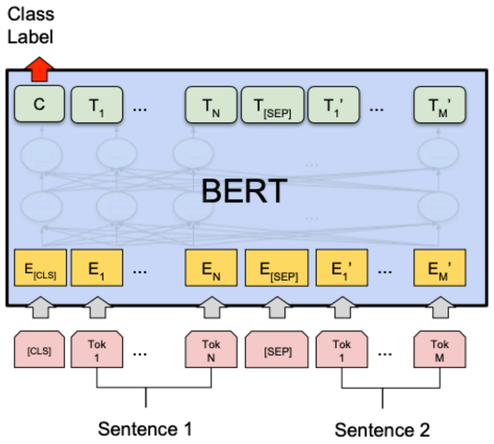


Fine-tuning

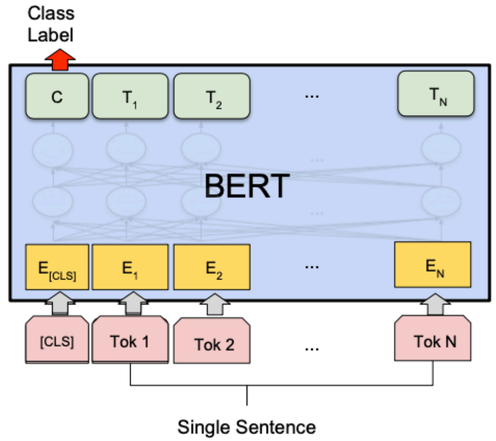
(figure credit: Jay Alammar http://jalammar.github.io/illustrated-bert/)

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

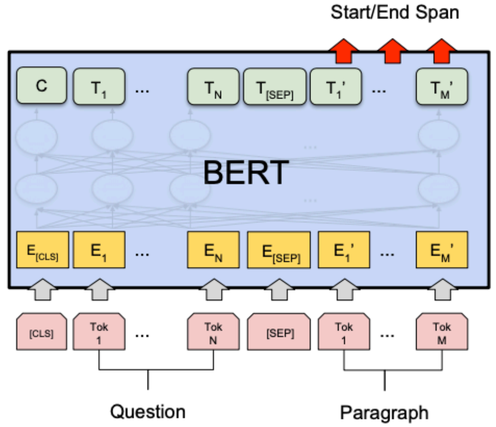
Applications



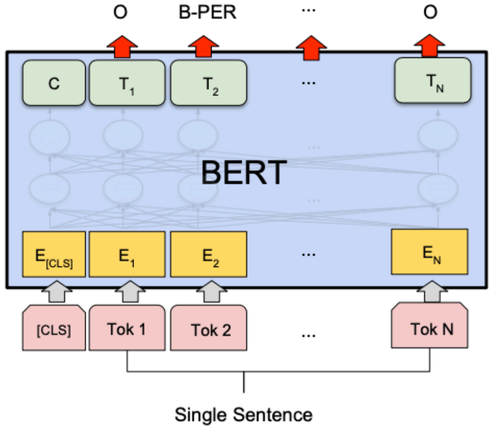
(a) Sentence Pair Classification Tasks: MNLI, QQP, QNLI, STS-B, MRPC, RTE, SWAG



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

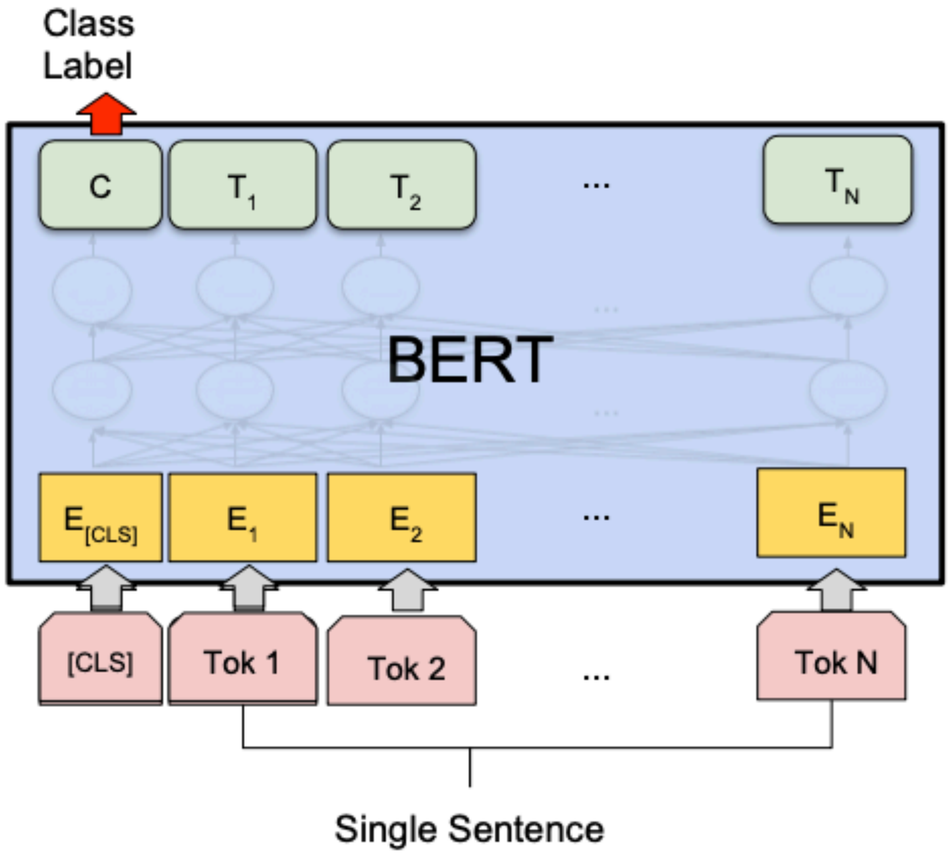


(c) Question Answering Tasks: SQuAD v1.1



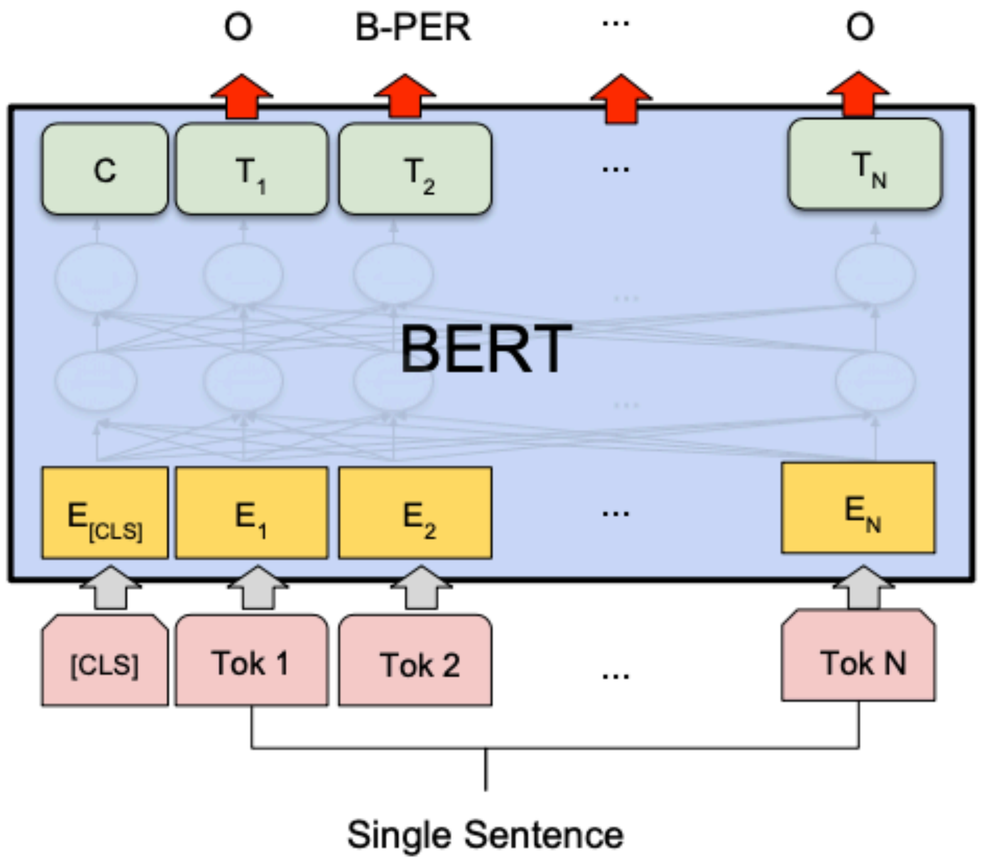
(d) Single Sentence Tagging Tasks: CoNLL-2003 NER

Applications



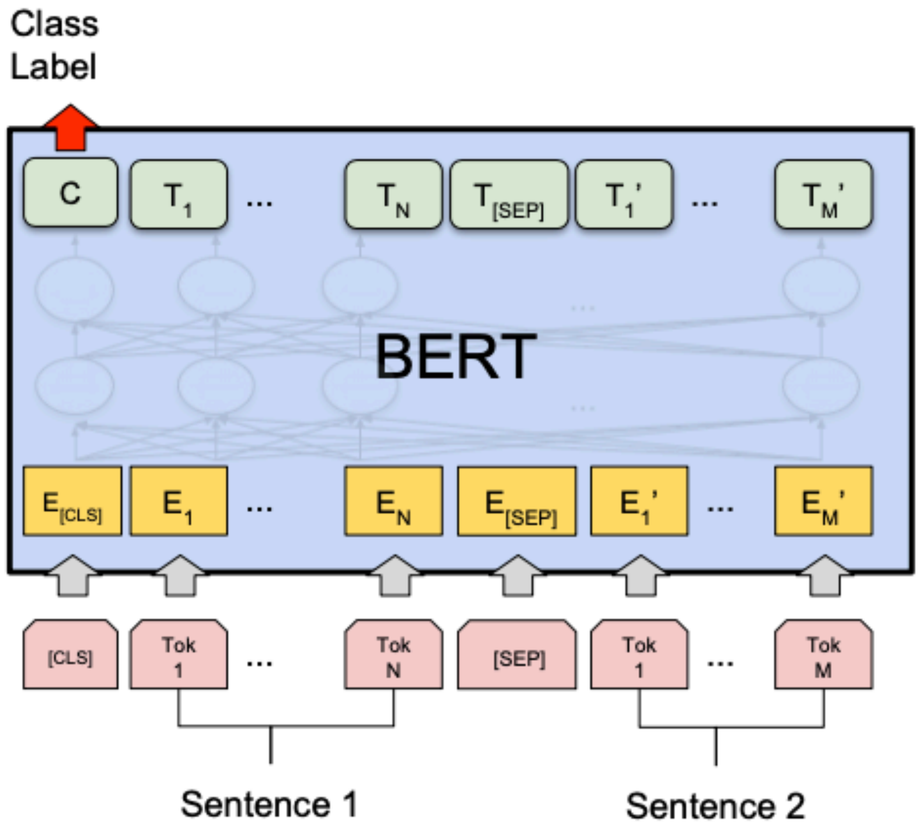
(b) Single Sentence Classification Tasks:
SST-2, CoLA

Applications



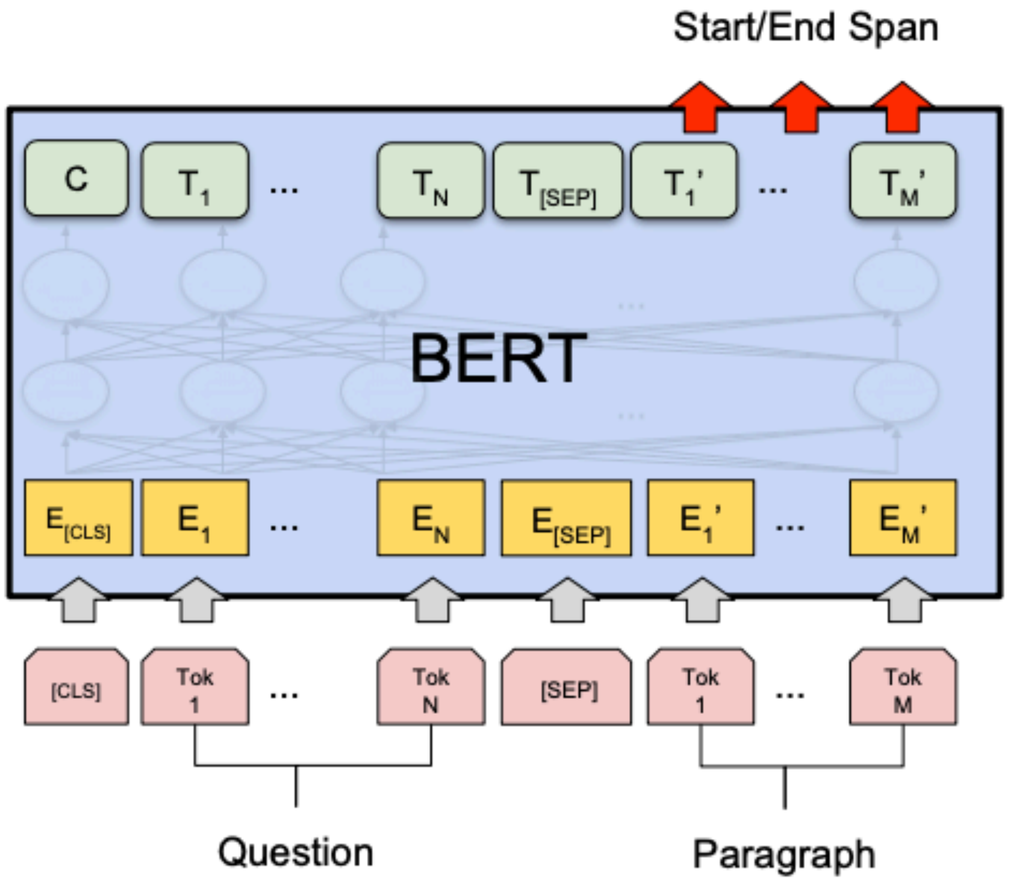
(d) Single Sentence Tagging Tasks:
CoNLL-2003 NER

Applications



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

Applications

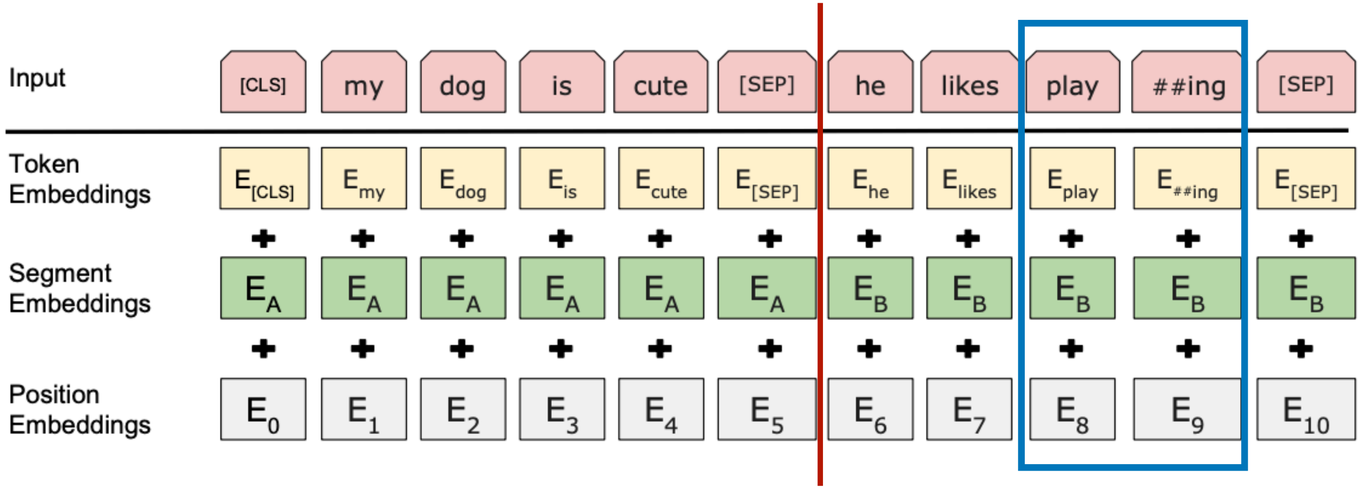


(c) Question Answering Tasks:
SQuAD v1.1

More details

- Input representations

Sum of embeddings



- Use word pieces instead of words: playing => play ##ing (30K token vocabulary)
- 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

Entailment

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

Similarity

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

Other

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

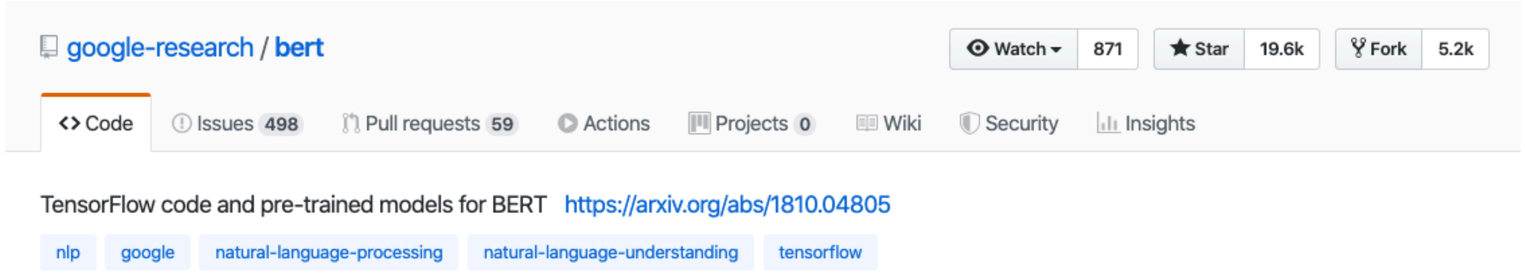
System	MNLI-(m/mm) 392k	QQP 363k	QNLI 108k	SST-2 67k	CoLA 8.5k	STS-B 5.7k	MRPC 3.5k	RTE 2.5k	**Average** -
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.9	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	88.1	91.3	45.4	80.0	82.3	56.0	75.2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

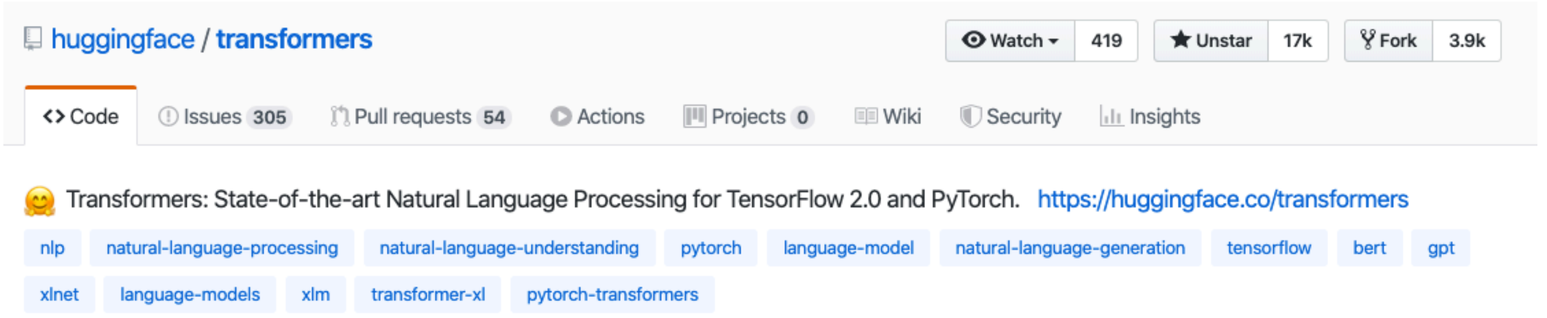
(Wang et al, 2018): GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding

Use BERT in practice

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



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

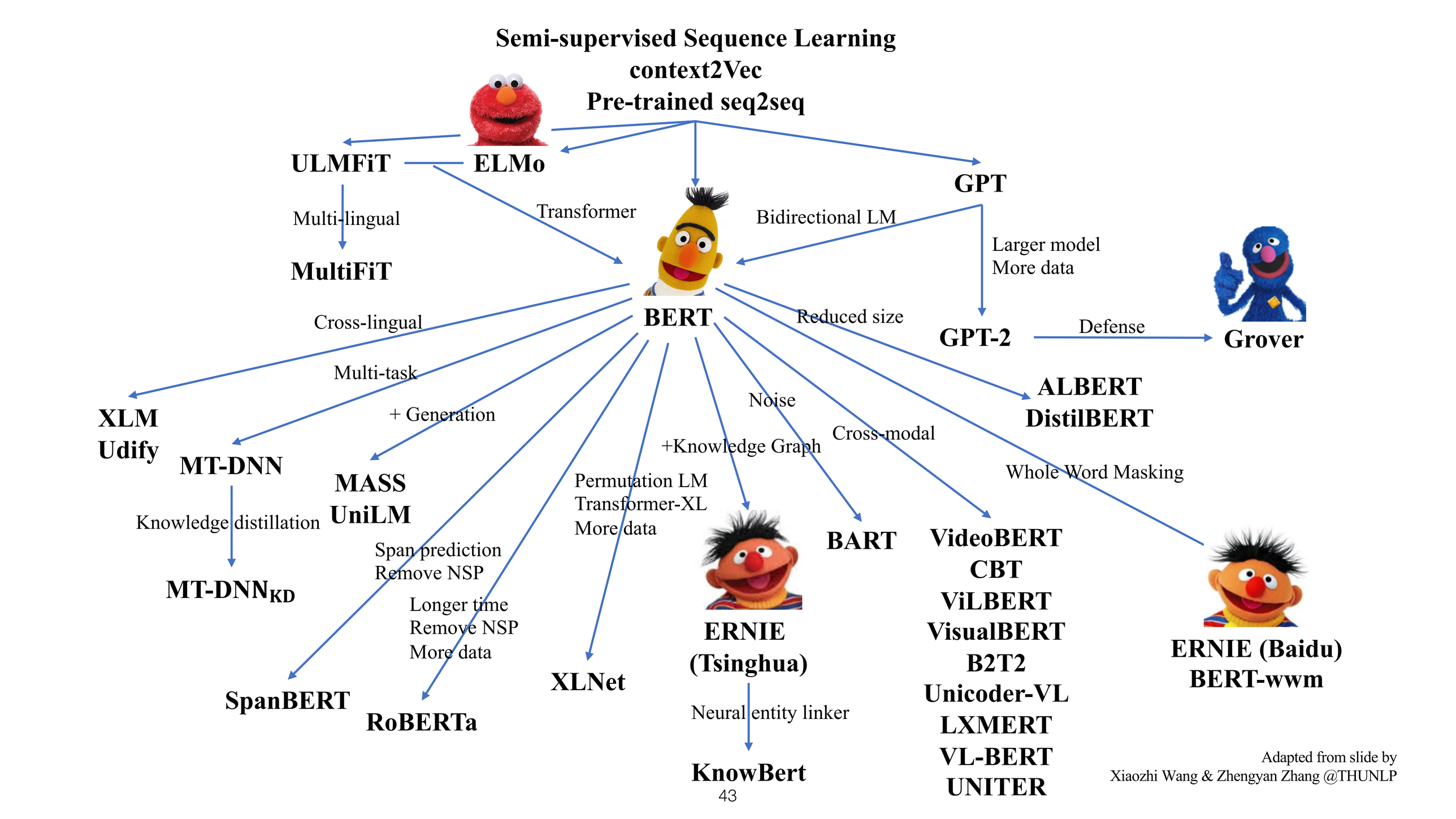


Contextualized word embeddings in context

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

https://github.com/huggingface/transformers

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



Adapted from slide by
Xiaozhi Wang & Zhengyan Zhang @THUNLP

All of these models are Transformer models

ELMo
Oct 2017
Training:
800M words
42 GPU days

GPT
June 2018
Training
800M words
240 GPU days

BERT
Oct 2018
Training
3.3B words
256 TPU days
~320–560
GPU days

GPT-2
Feb 2019
Training
40B words
~2048 TPU v3
days according to
a reddit thread

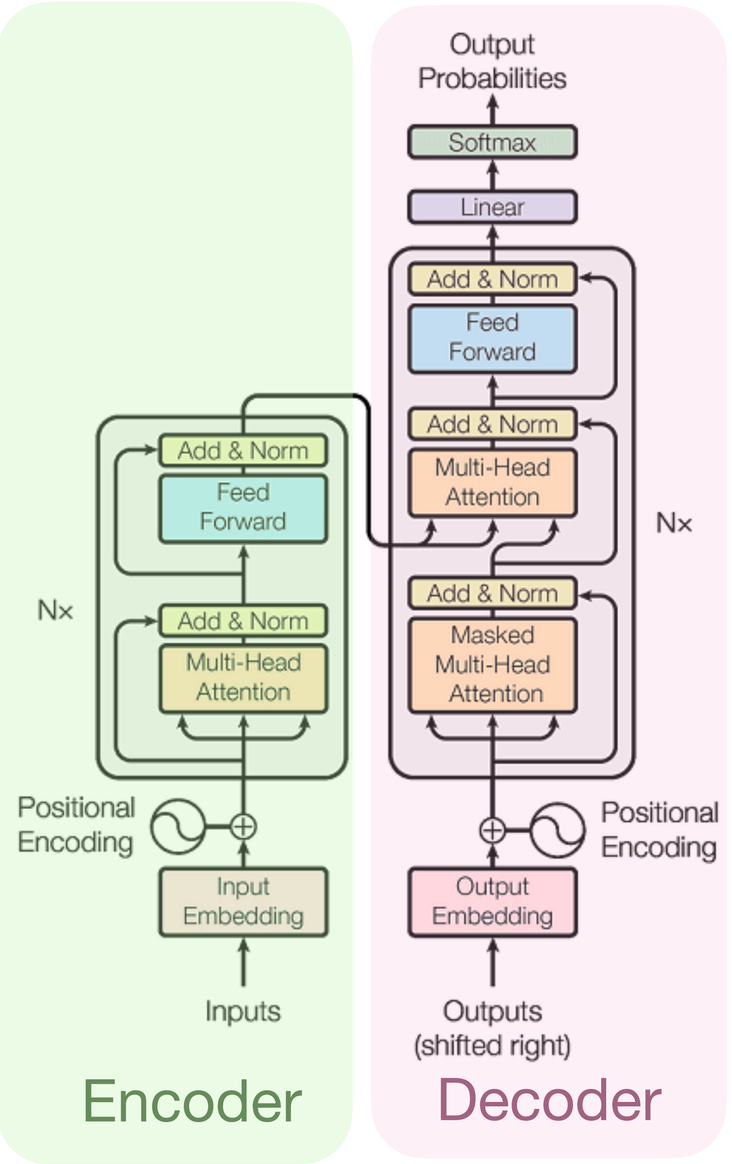
XL-Net,
ERNIE,
Grover
RoBERTa, T5
July 2019—

(slide credit: Stanford CS224N, Chris Manning)

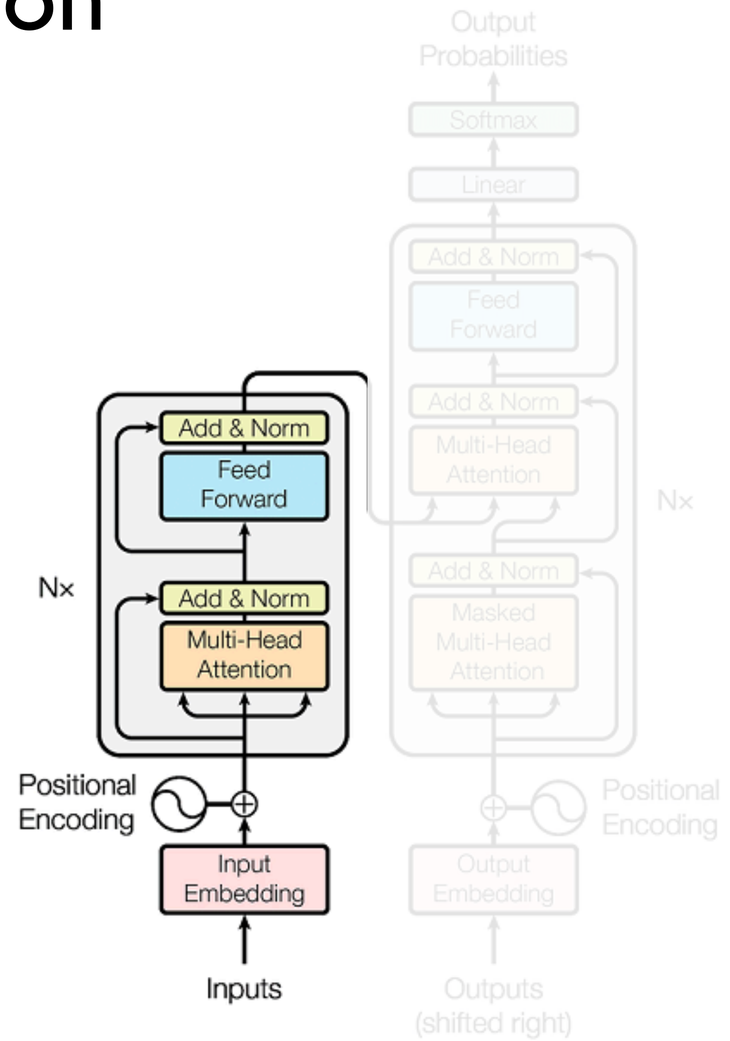
Transformers

Transformers

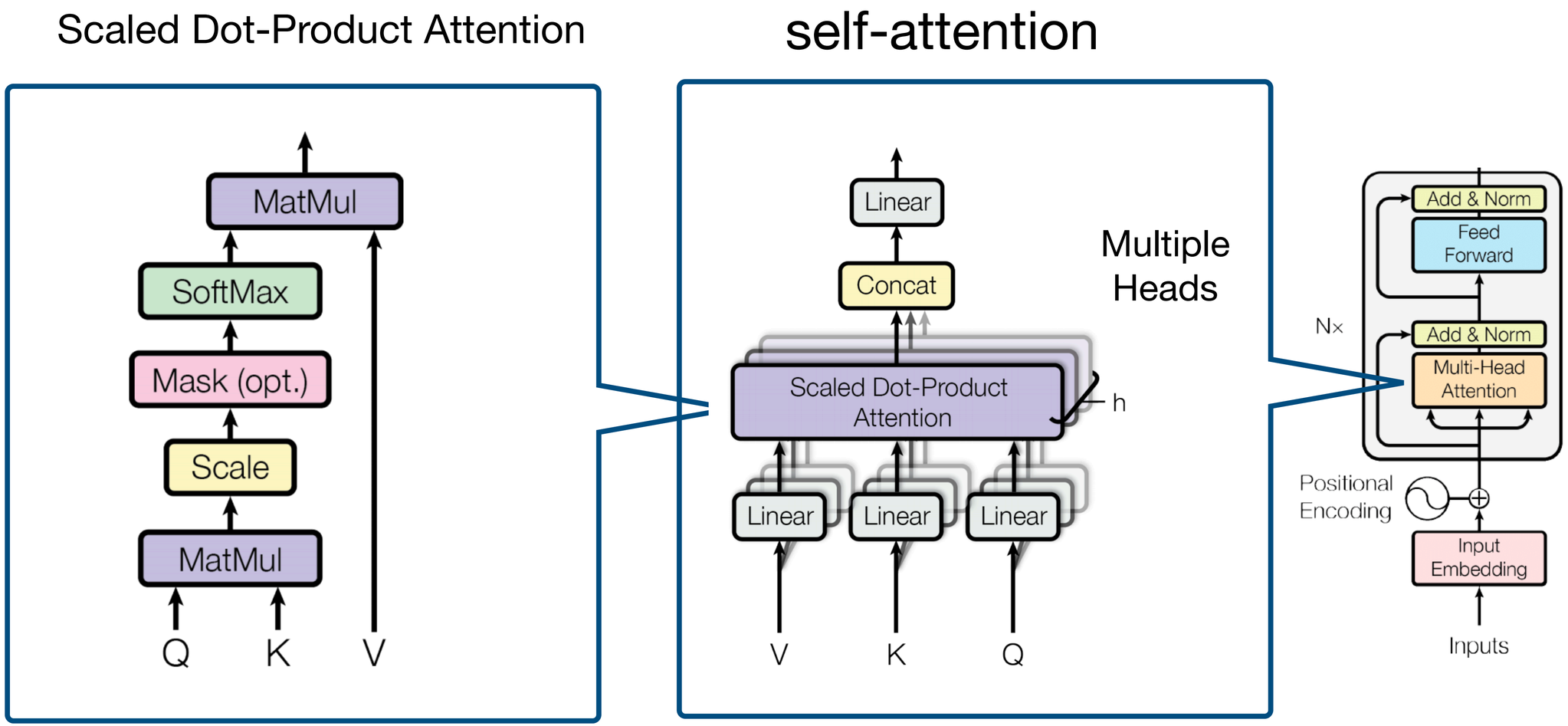
- NIPS'17: Attention is All You Need
- Originally proposed for NMT (encoder-decoder framework)
- Used as the base model of BERT (encoder only)
- Key idea: **Multi-head self-attention**
- No recurrence structure any more so it trains much faster



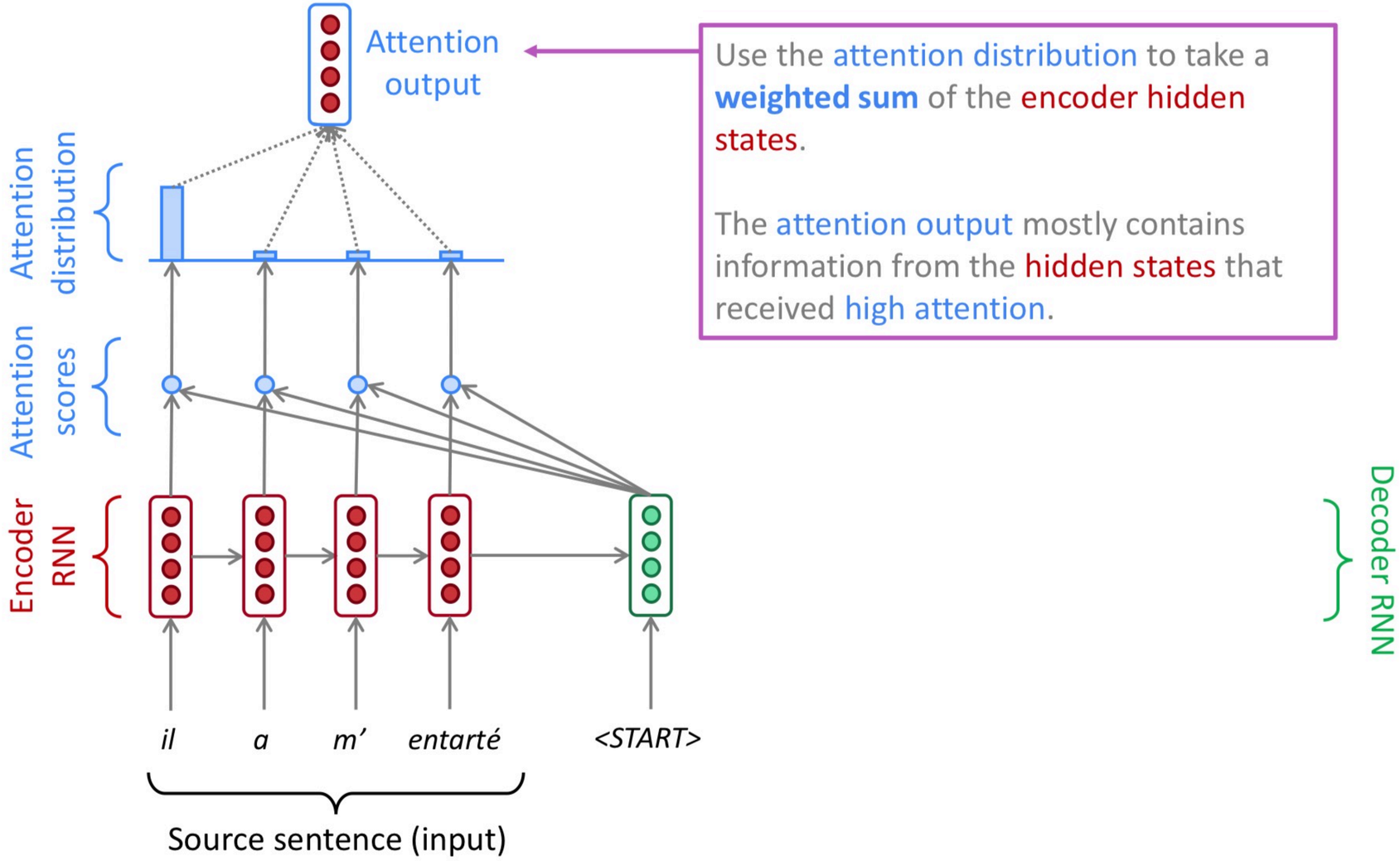
Multi-head self-attention



Multi-head self-attention



Recall: attention



Use the attention distribution to take a **weighted sum** of the encoder hidden states.

The attention output mostly contains information from the hidden states that received high attention.

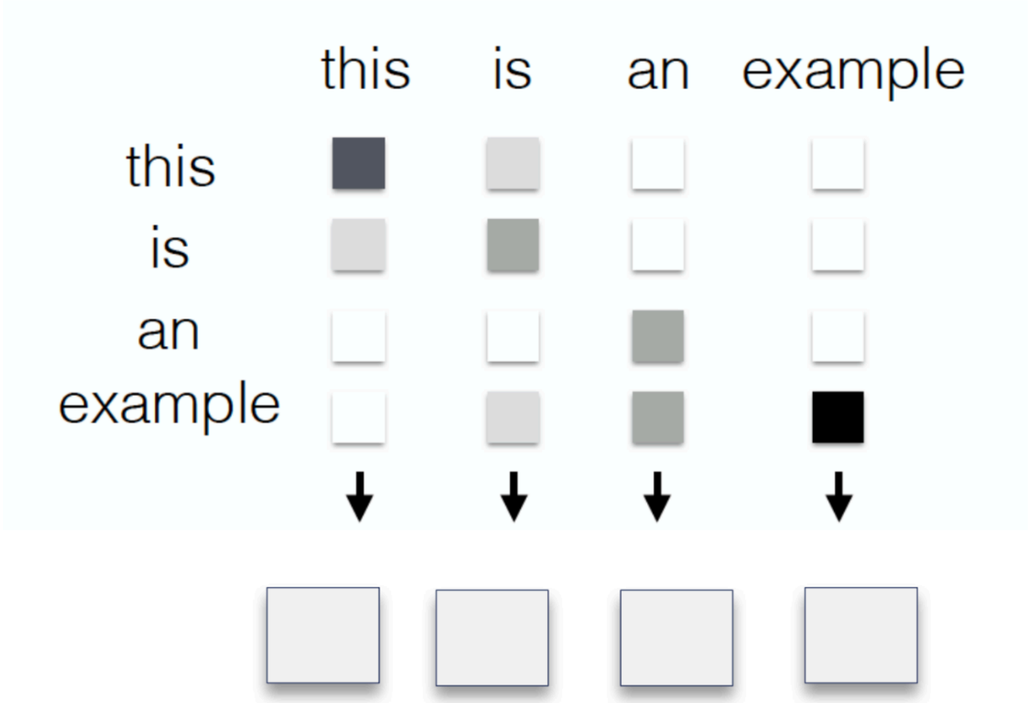
(slide credit: Abigail See)

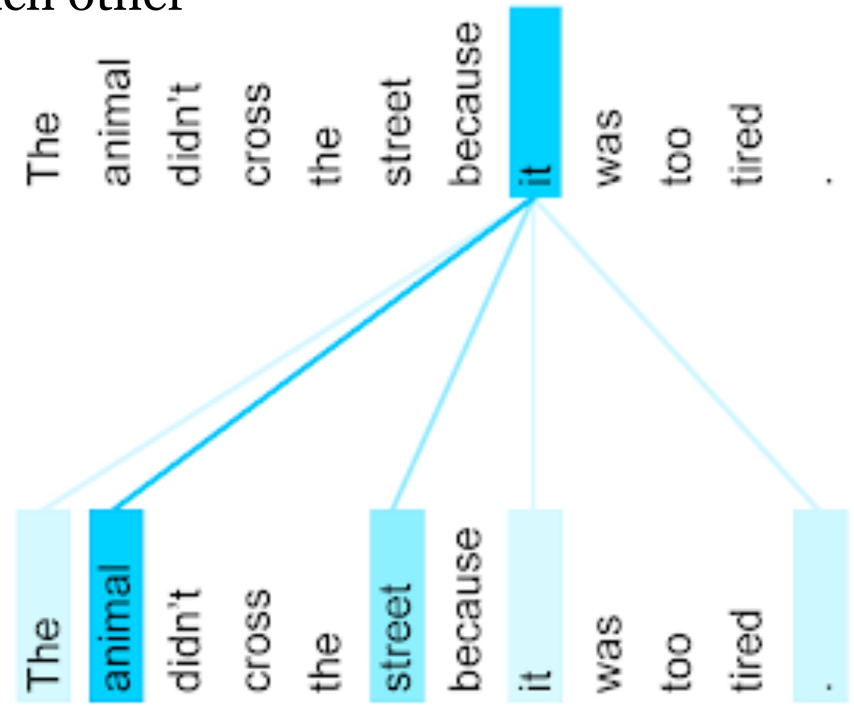
Self Attention

(also referred to as Intra-Attention)

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

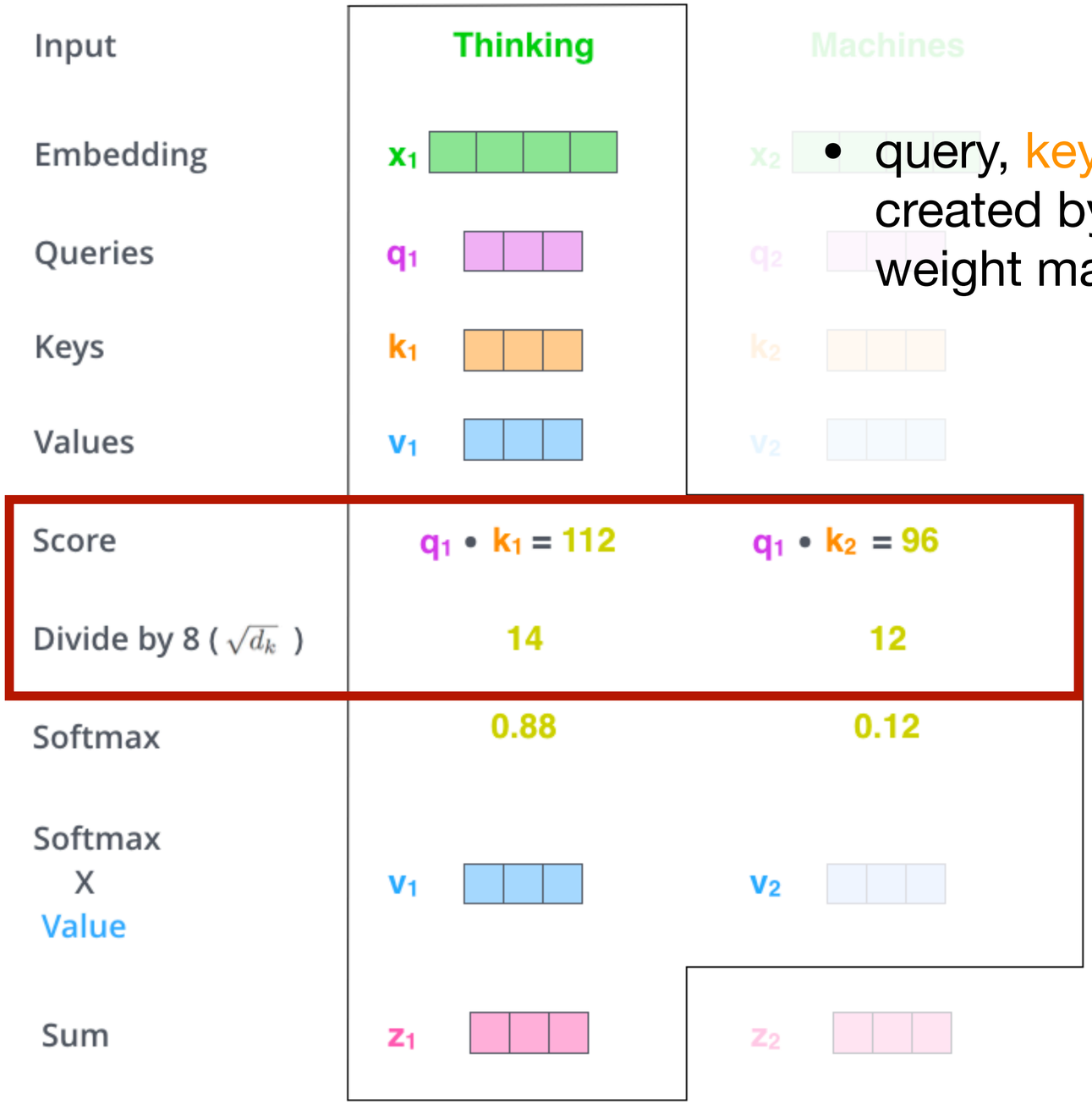
  = the word vectors themselves select each other





General definition of attention

- **Attention** is a general concept
  - Given query q and a set of key-value pairs (K,V)
  - **Attention** is a way to compute a **weighted sum** of the values dependent on the query and the corresponding keys.
  - Query determines what values to focus on, the query “attends” to the values
  - All of these (key value query) are represented using **vectors**
    - These vectors are created by multiplying embedding by trained weight matrices.



- query, key, and value vectors created by multiplying learned weight matrices with embedding

- Can be any kind of attention function
- For transformers, this is the scaled dot-product attention

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

Recall: types of attention

- Assume encoder hidden states $h_1, h_2, \ldots, h_n$ and decoder hidden state z

1. **Dot-product attention** (assumes equal dimensions for a and b):

$$g(h_i, z) = z^T h_i \in \mathbb{R}$$

more efficient (matrix multiplication)

Simplest (no extra parameters)
requires z and h_i to be same size

2. **Bilinear / multiplicative attention:**

$$g(h_i, z) = z^T W h_i \in \mathbb{R}, \text{ where } W \text{ is a weight matrix}$$

More flexible than dot-product (W is trainable)

3. **Additive attention (essentially MLP):**

$$g(h_i, z) = v^T \tanh\left(W_1 h_i + W_2 z\right) \in \mathbb{R}$$

where W_1, W_2 are weight matrices and v is a weight vector

Perform better for larger dimensions

Scaled dot-product attention

- Assume encoder hidden states $h_1, h_2, \ldots, h_n$ and decoder hidden state z

1. **Dot-product attention** (assumes equal dimensions for a and b):

$$g(h_i, z) = z^T h_i \in \mathbb{R}$$

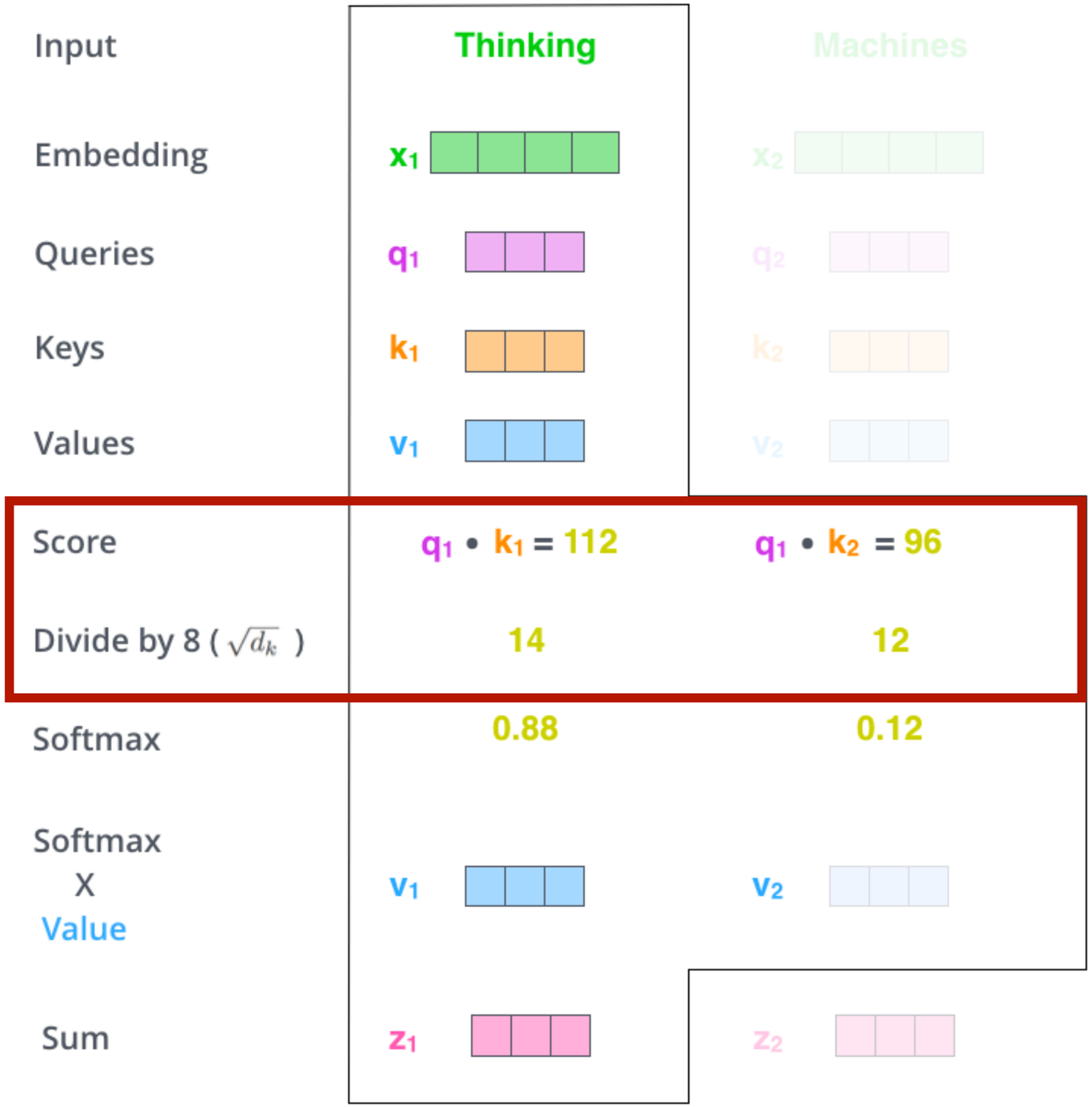
Perform poorly for large *d*
Softmax has small gradient

2. **Scaled dot-product attention:**

$$g(h_i, z) = \frac{z^T h_i}{\sqrt{d}} \in \mathbb{R}$$

Maybe will perform well
for larger dimensions

Scaling factor: d = dimension of hidden state

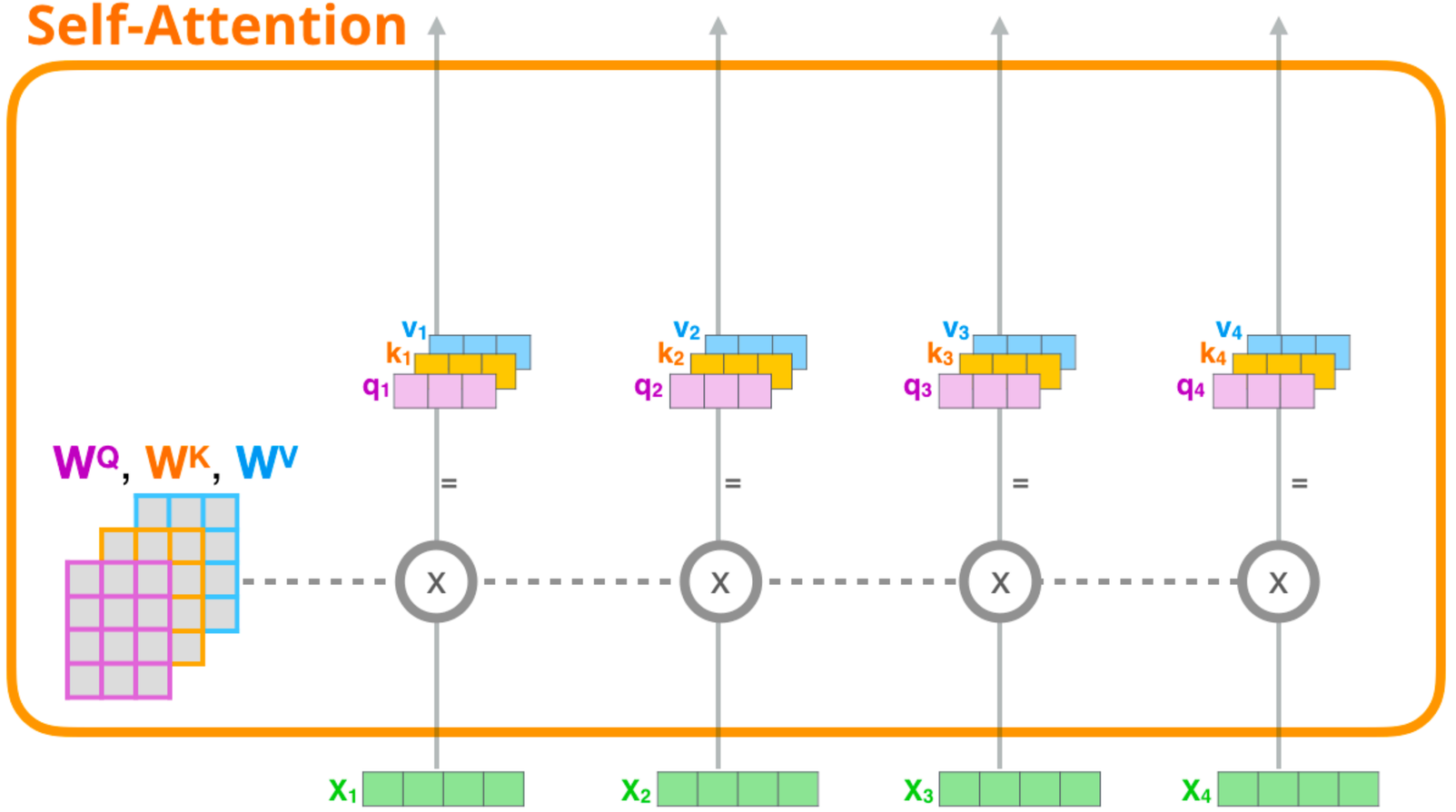


- Can be any kind of attention function
- For transformers, this is the scaled dot-product attention

- z_1 is the final vector of attended values for “Thinking” as the query

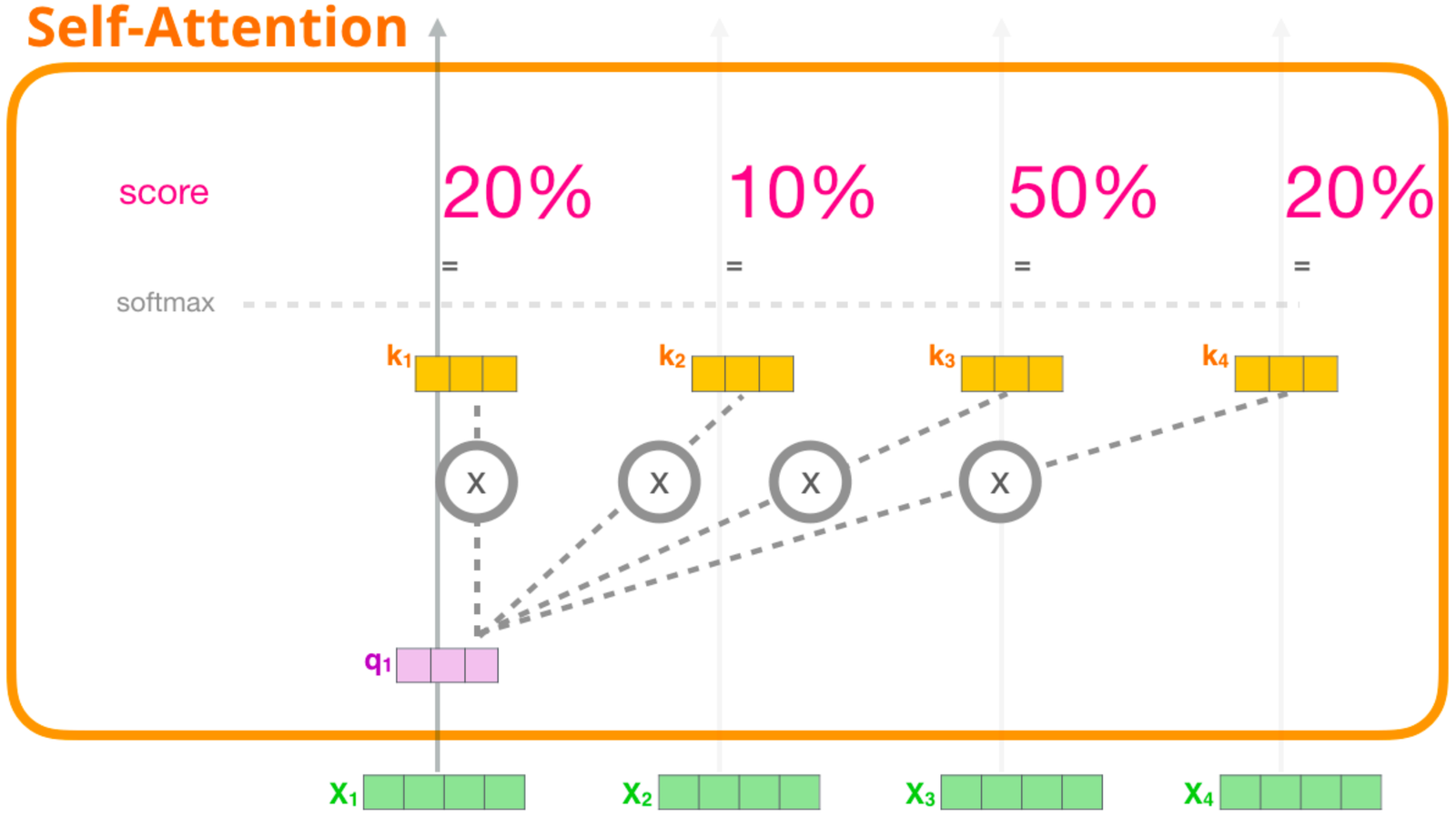
(figure credit: Jay Alammar http://jalammar.github.io/illustrated-transformer/)

1) For each input token, create a query vector, a key vector, and a value vector by multiplying by weight Matrices W^Q, W^K, W^V



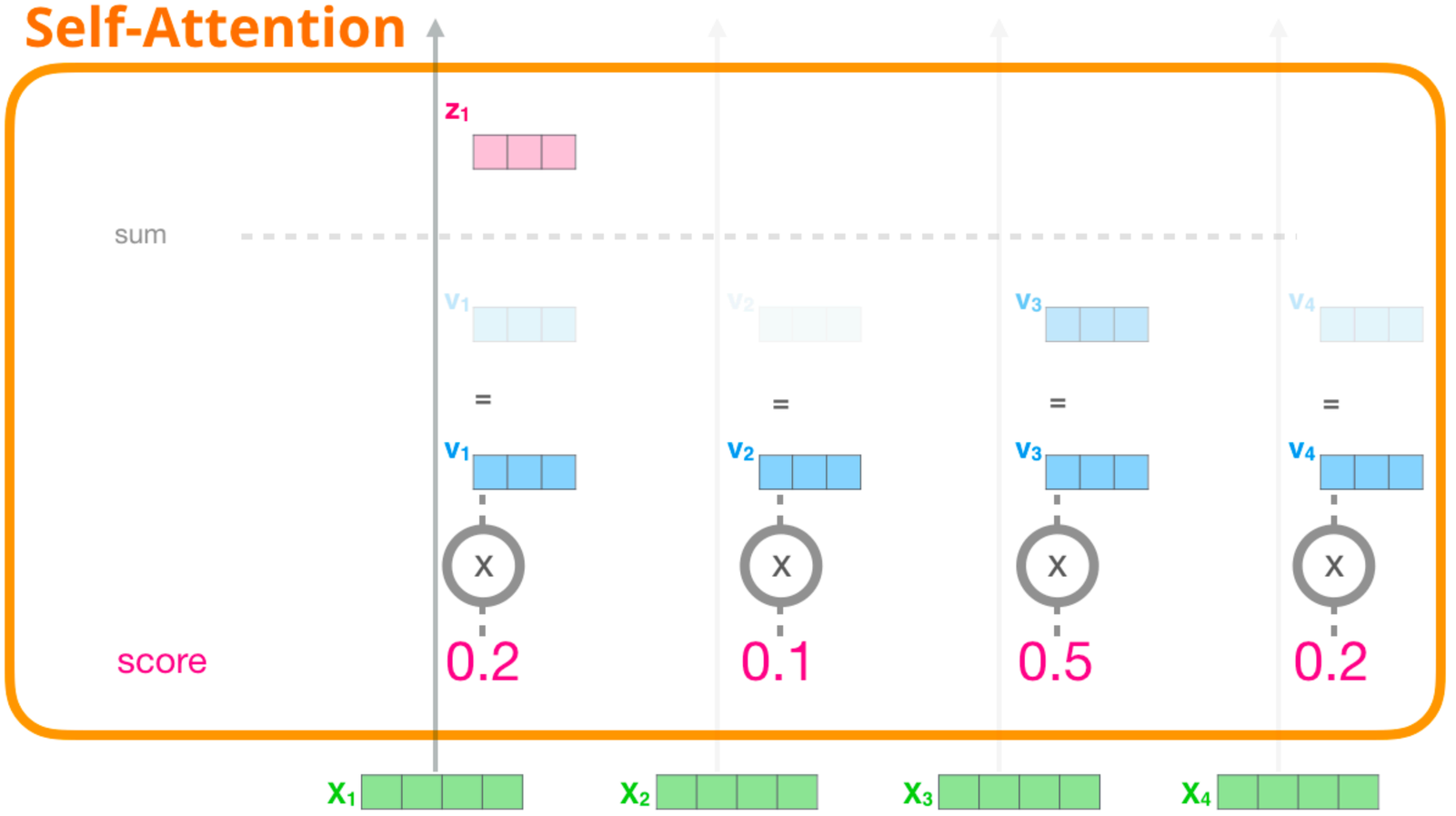
(figure credit: Jay Alammar http://jalammar.github.io/illustrated-gpt2/)

2) Multiply (dot product) the current query vector, by all the key vectors, to get a score of how well they match

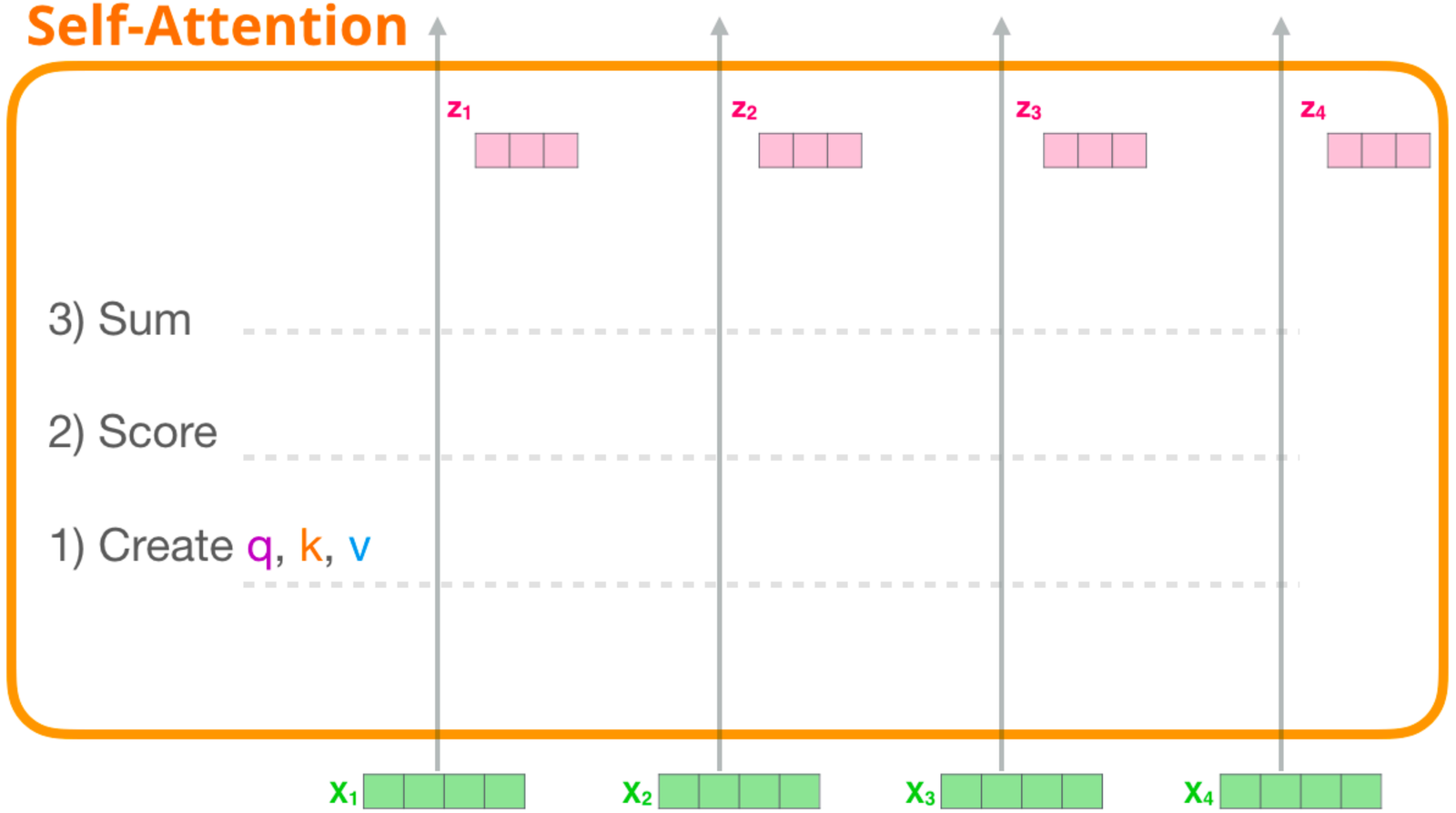


(figure credit: Jay Alammar http://jalammar.github.io/illustrated-gpt2/)

3) Multiply the value vectors by the scores, then sum up



(figure credit: Jay Alammar http://jalammar.github.io/illustrated-gpt2/)



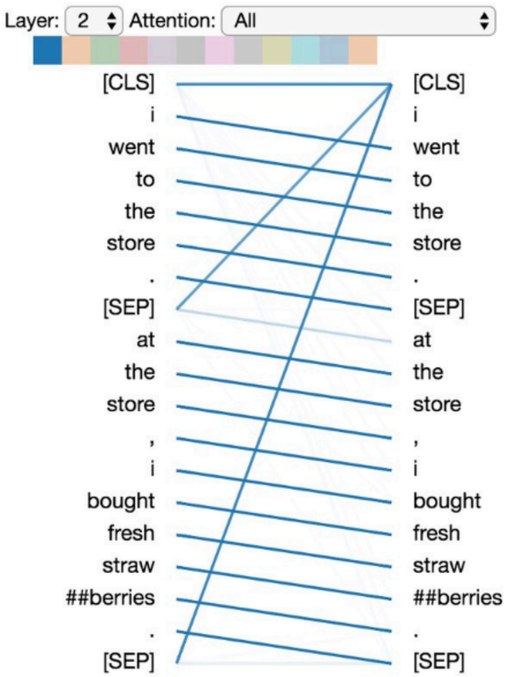
(figure credit: Jay Alammar http://jalammar.github.io/illustrated-gpt2/)

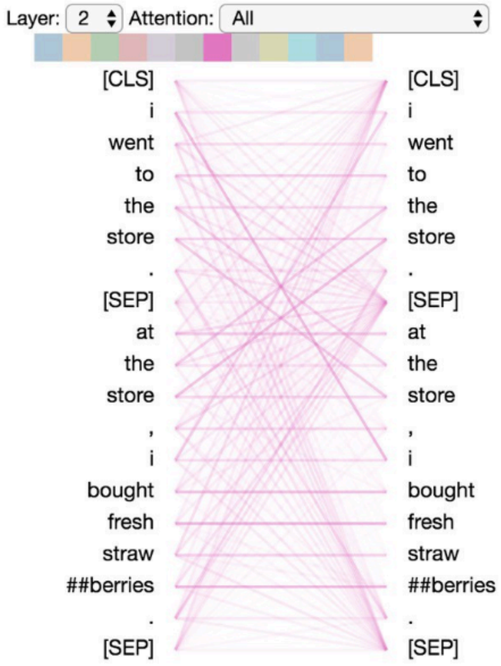
Multi-head self-attention

One head is not expressive enough. Let's have multiple heads!

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

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

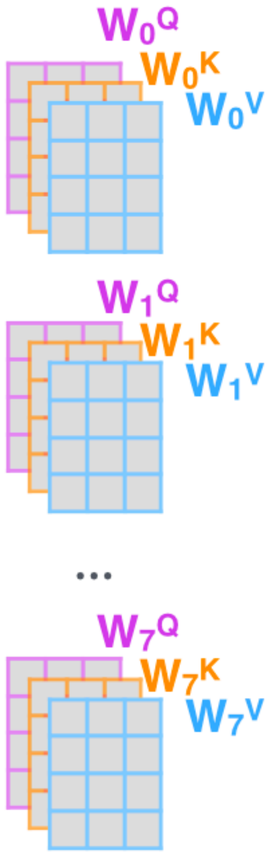




https://github.com/jessevig/bertviz

Multiple heads

- Multiple (different) representations for each query, key, and values
- Different weight matrices —> different vectors
- Different ways for the words to interact with each other



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

Summary: 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

Summary: Multi-head Self Attention

- Scaled Dot-Product Attention:

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

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

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

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

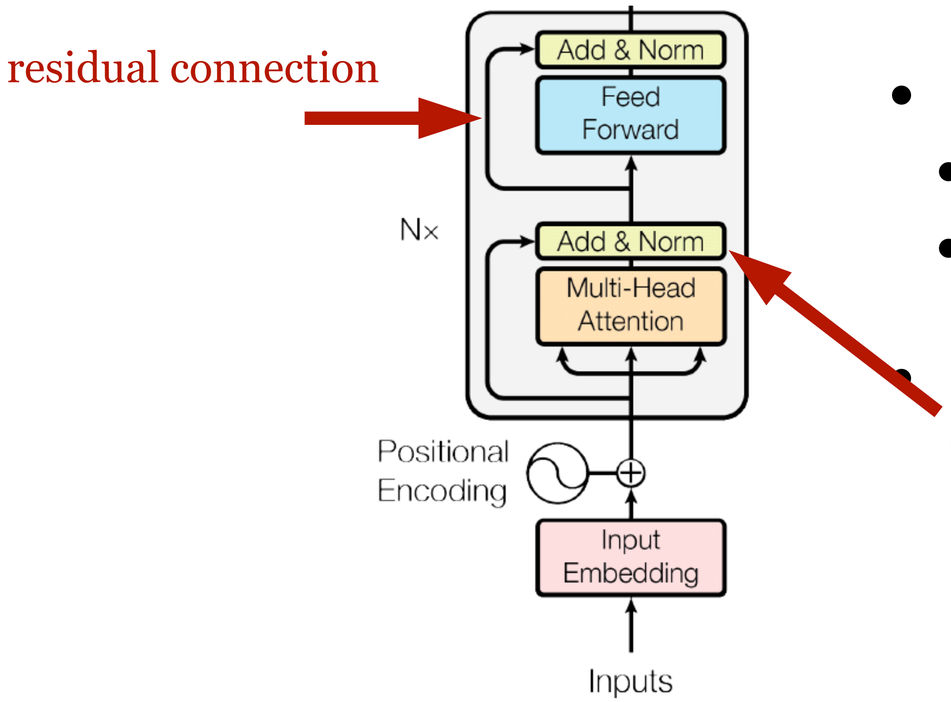
- Multi-head attention: using more than one head is always useful..

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

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

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

Putting it all together

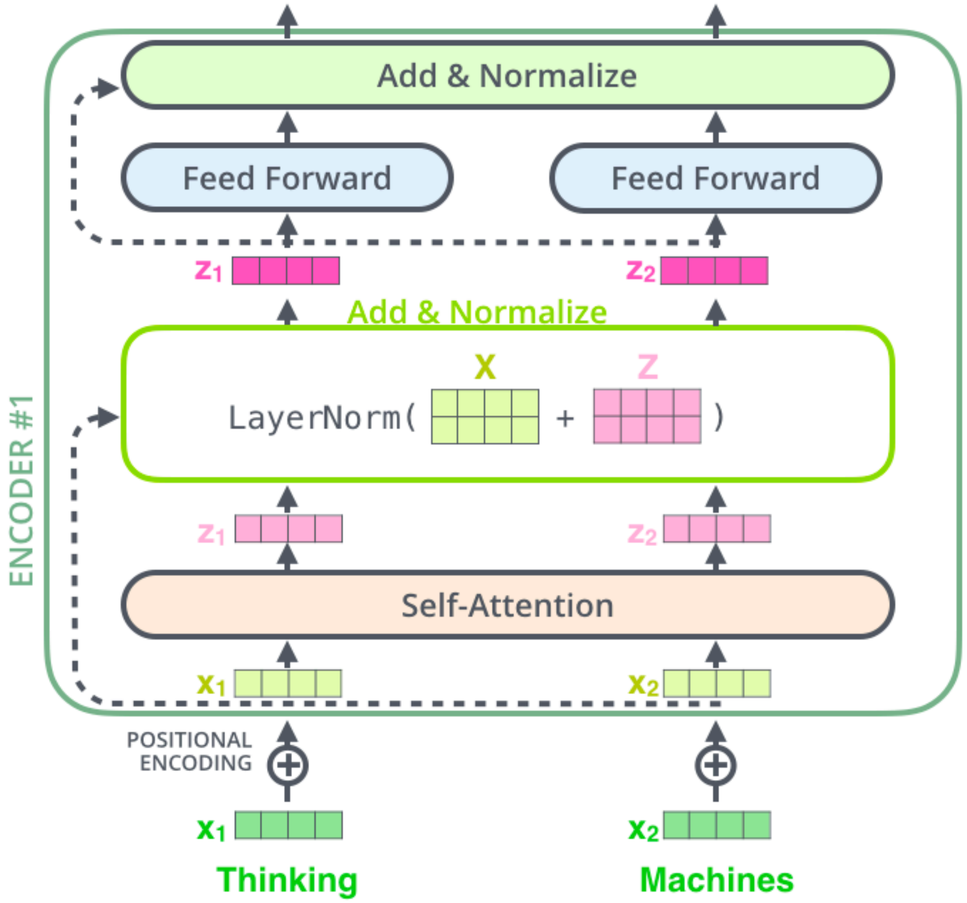


- Each Transformer block has two sub-layers
  - Multi-head attention
  - 2-layer feedforward NN (with ReLU)
- Each sublayer has a residual connection and a layer normalization

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

(Ba et al, 2016): Layer Normalization

Residual connections and Layer Normalization



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

LayerNorm

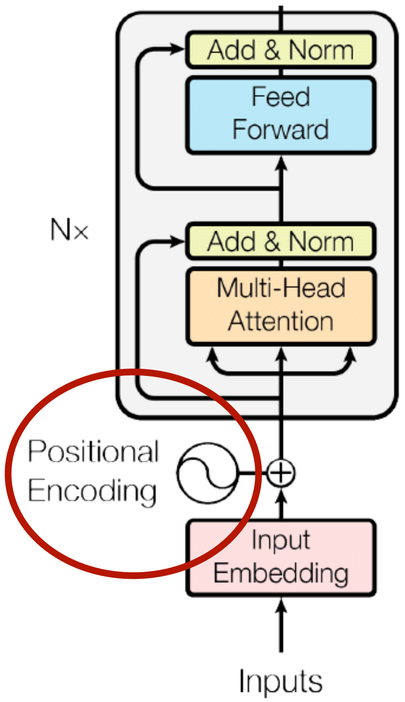
- changes input features to have mean 0 and variance 1 per layer.
- Adds two more parameters

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

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

(Ba et al, 2016): Layer Normalization

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

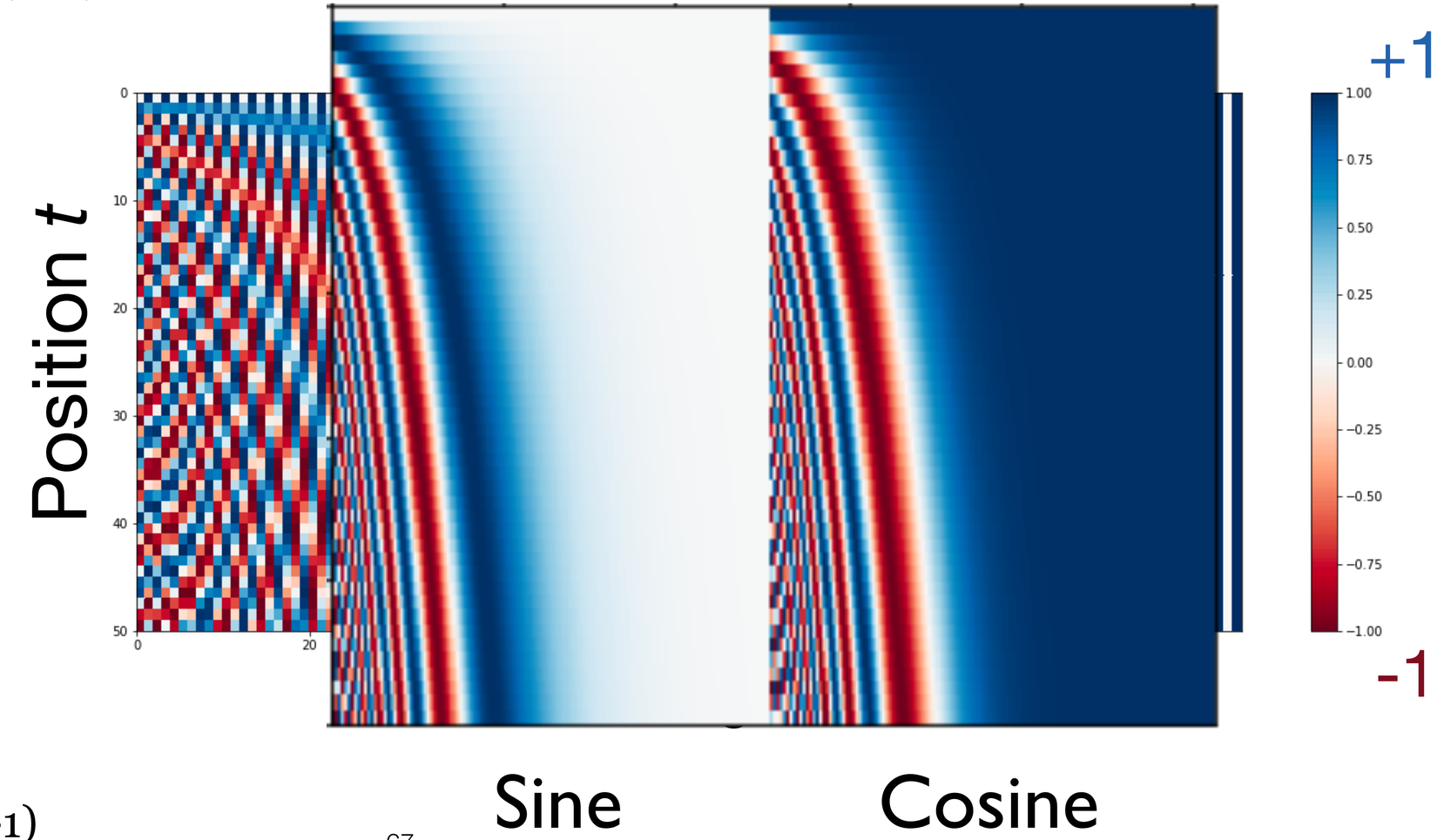
(Ba et al, 2016): Layer Normalization

Positional encoding

$$\overrightarrow{p_t}^{(i)} = f(t)^{(i)} := \begin{cases} \sin(\omega_k . t), & \text{if } i = 2k \\ \cos(\omega_k . t), & \text{if } i = 2k+1 \end{cases}$$

$$\omega_k = \frac{1}{10000^{2k/d}}$$

$$\overrightarrow{p_t} = \begin{bmatrix} \sin(\omega_1 . t) \\ \cos(\omega_1 . t) \\ \\ \sin(\omega_2 . t) \\ \cos(\omega_2 . t) \\ \\ \vdots \\ \\ \sin(\omega_{d/2} . t) \\ \cos(\omega_{d/2} . t) \end{bmatrix}_{d \times 1}$$

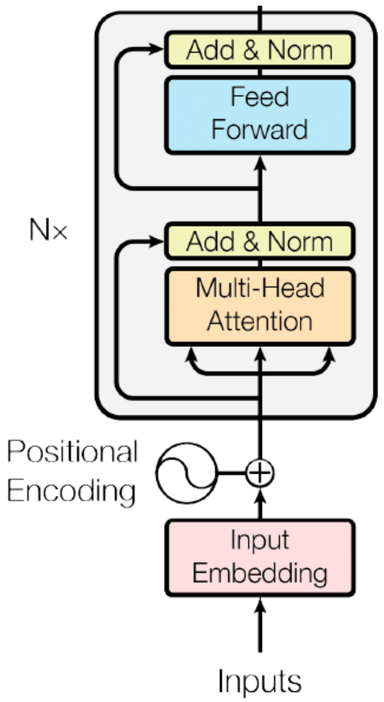


t = position

d = embedding dimension

i = embedding index (0 to d-1)

Putting it all together

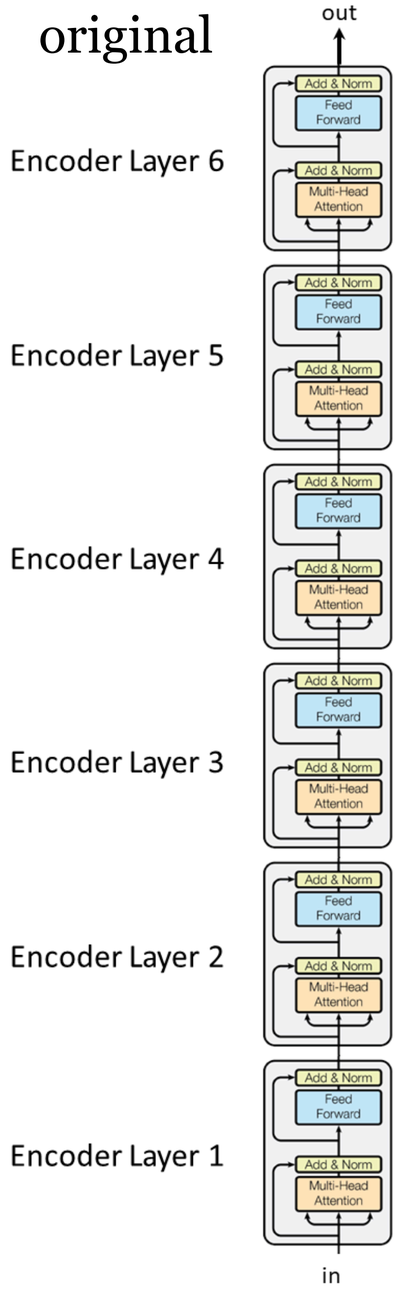


- Each Transformer block has two sub-layers
  - Multi-head attention
  - 2-layer feedforward NN (with ReLU)
- Each sublayer has a residual connection and a layer normalization

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

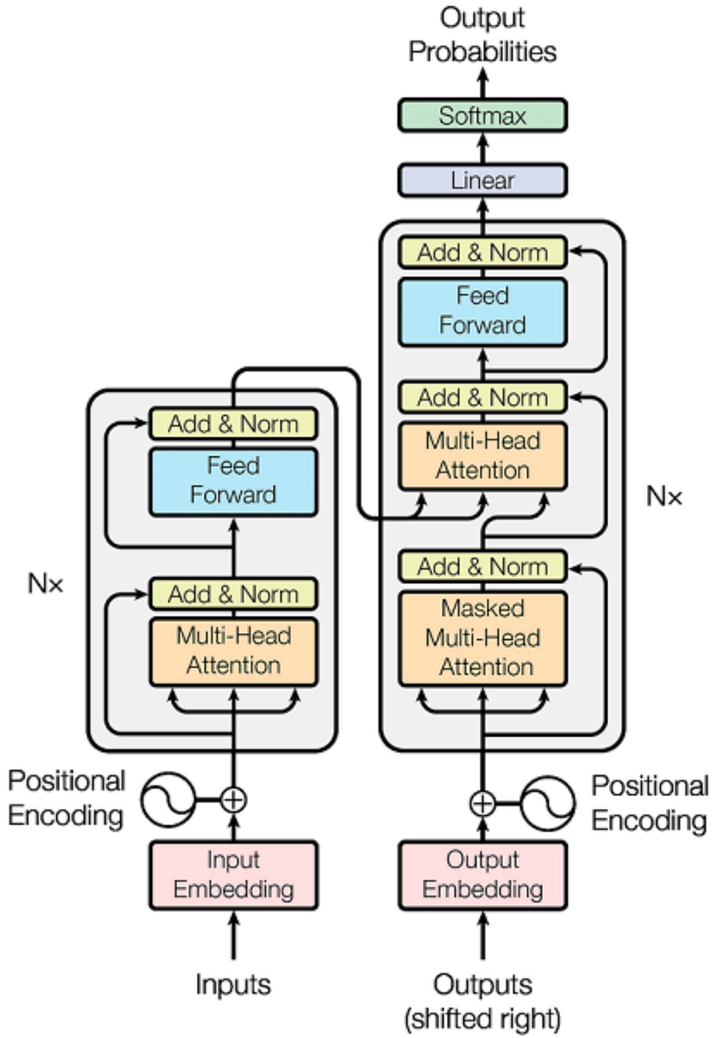
- Input layer has a positional encoding
- Input embedding is byte pair encoding (BPE)

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



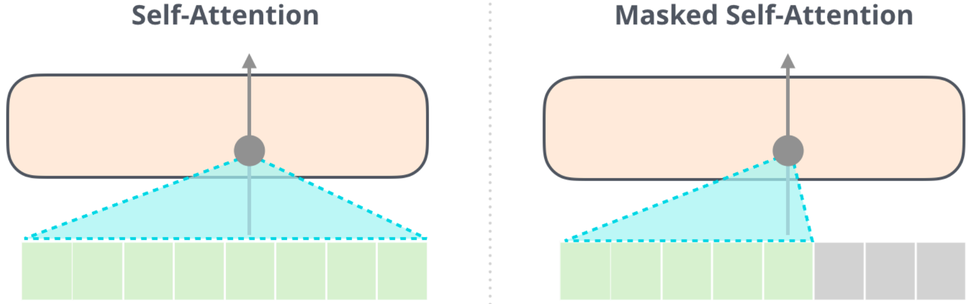
(Ba et al, 2016): Layer Normalization

Transformer decoder



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

- Masked decoder self-attention on previously generated outputs



(figure credit: Jay Alammar http://jalammar.github.io/illustrated-gpt2/)

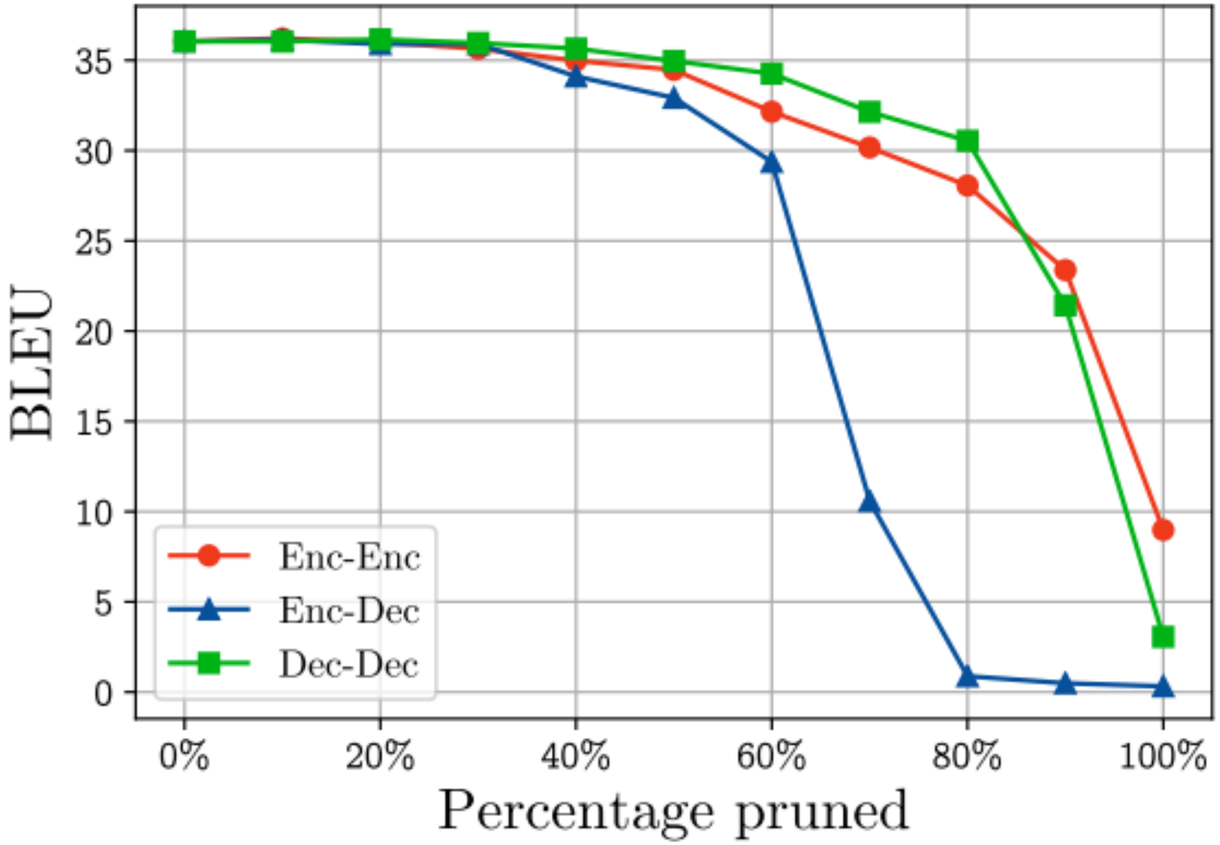
- also 6 layers (in original paper)

Do we need all these heads?

3 types of attention: Enc-Enc, Enc-Dec, Dec-Dec

6 layers, 16 heads each layer for each type

- Can we prune away some of the heads of a trained model during test time?



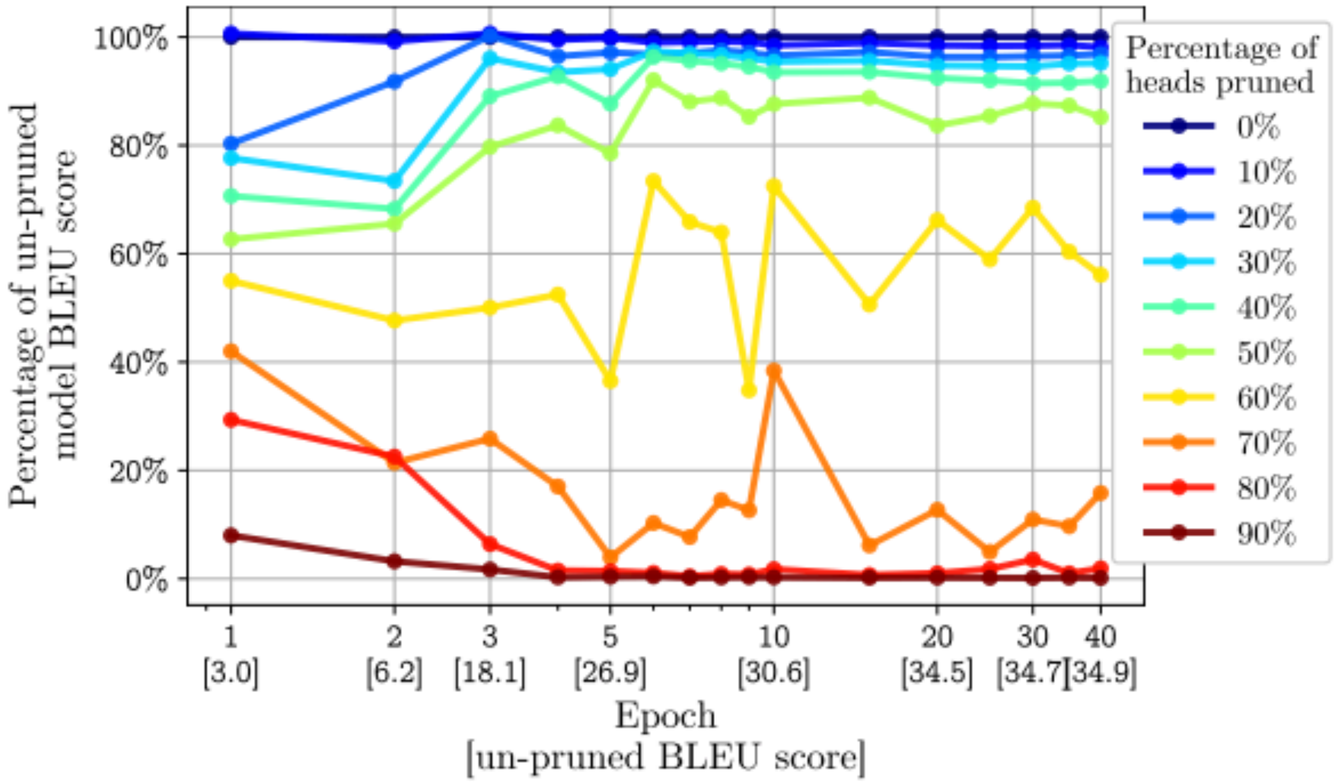
Are Sixteen Heads Really Better than One?
Michel, Levy, and Neubig, NeurIPS 2019

Do we need all these heads?

3 types of attention: Enc-Enc, Enc-Dec, Dec-Dec

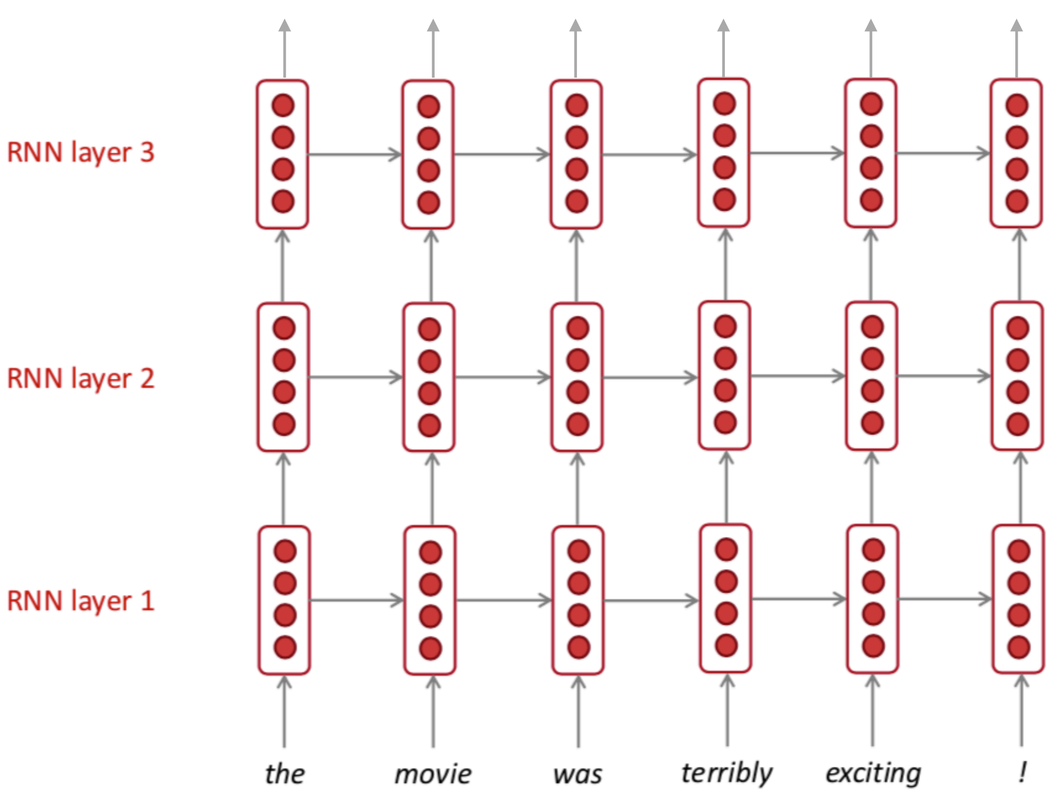
6 layers, 16 heads each layer for each type

- Can we train a good MT model with less heads?

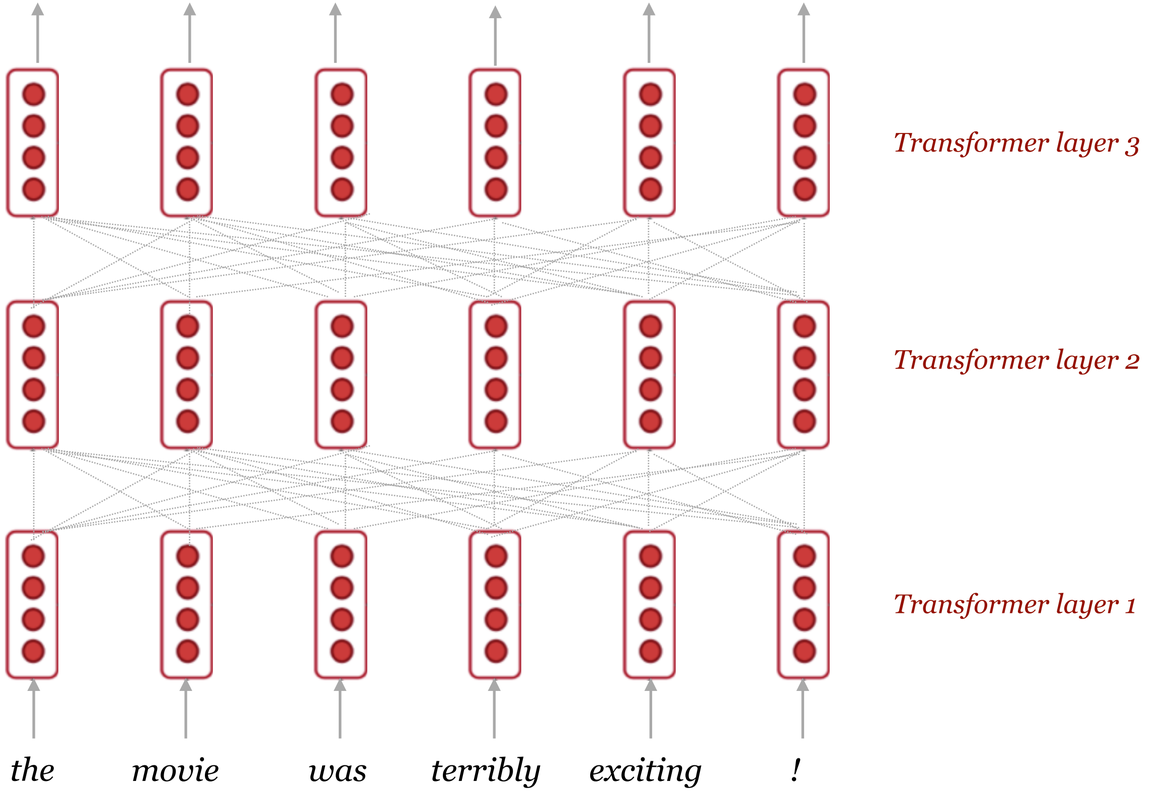


Are Sixteen Heads Really Better than One?
Michel, Levy, and Neubig, NeurIPS 2019

RNNs vs Transformers



RNN



Transformer

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)
```

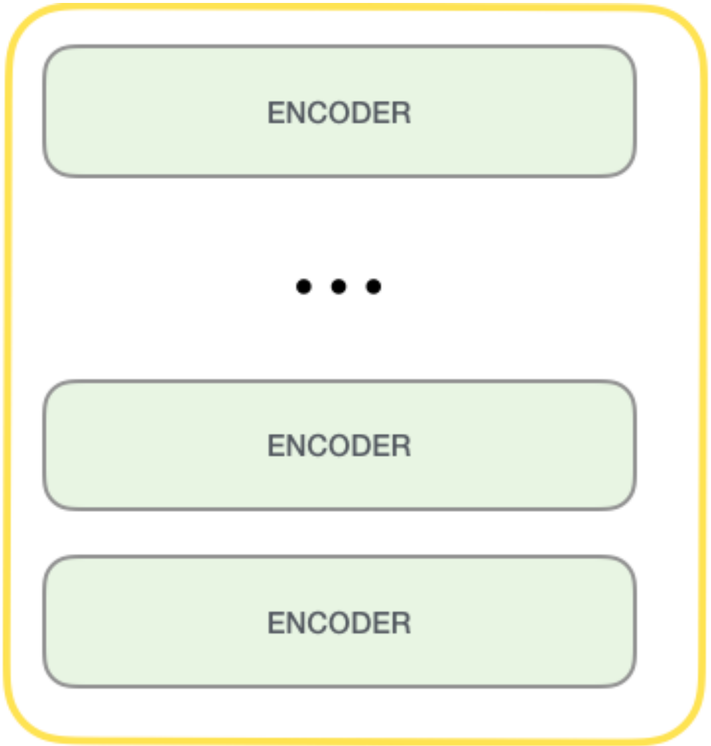
The Annotated Transformer:

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

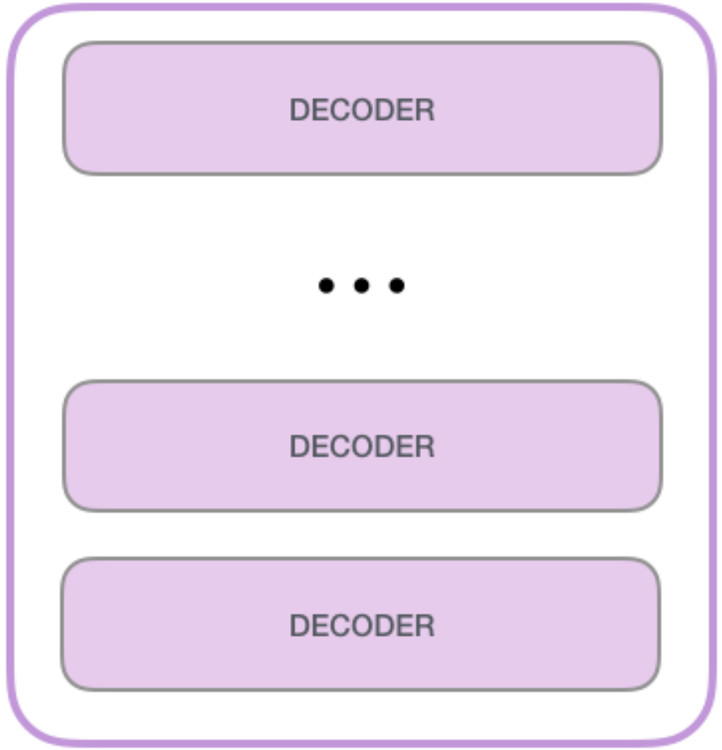
A Jupyter notebook which explains how Transformer works line by line in PyTorch!

Transformers blocks as building blocks

- BERT (built on Transformer encoders)



- GPT-2 (built on Transformer decoders)



(figure credit: Jay Alammar http://jalammar.github.io/illustrated-gpt2/)

GPT-2

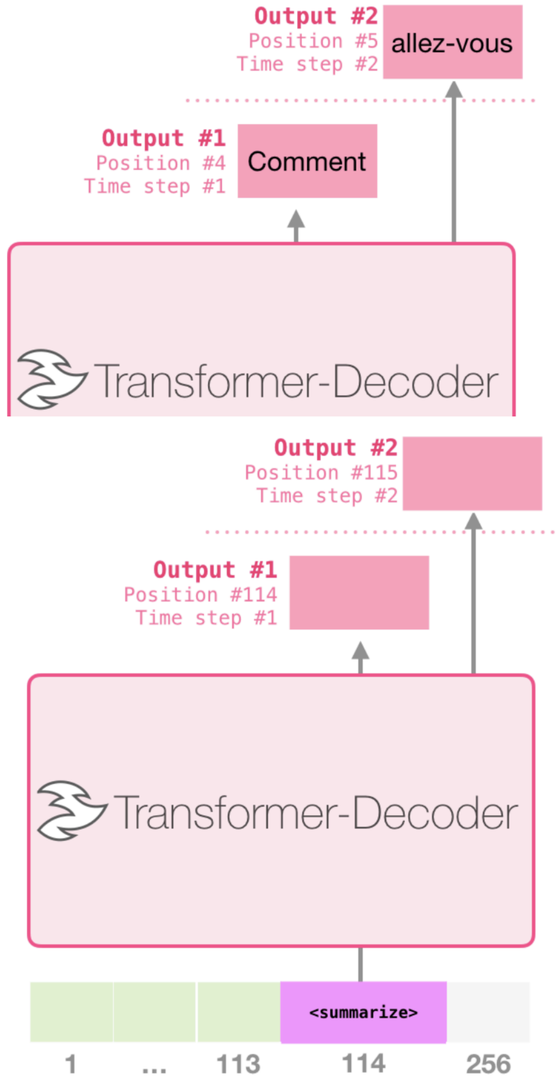
How can we use decoders for different tasks?

- Machine Translation

I	am	a	student	<to-fr>	je	suis	étudiant
let	them	eat	cake	<to-fr>	Qu'ils	mangent	de
good	morning	<to-fr>	Bonjour				

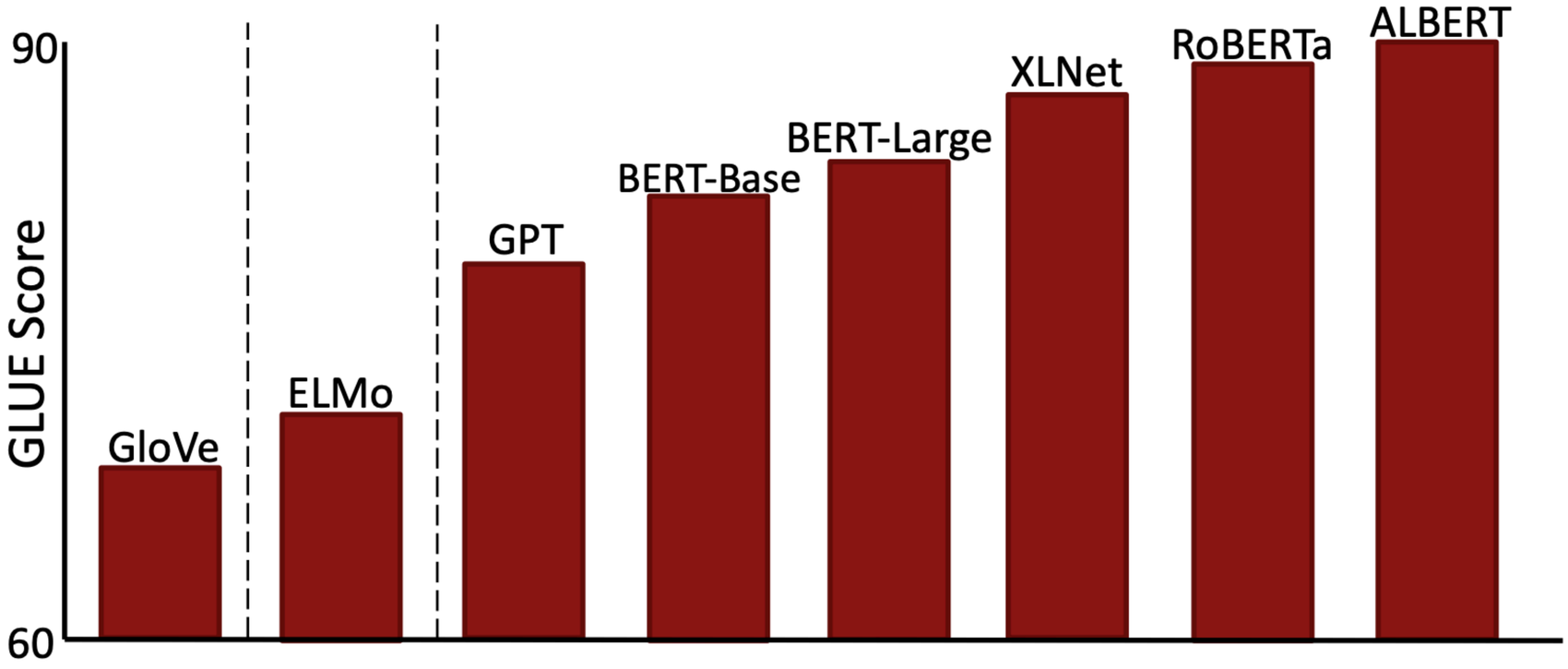
- Summarization

Article #1 tokens		<summarize>	Article #1 Summary
Article #2 tokens	<summarize>	Article #2 Summary	padding
Article #3 tokens		<summarize>	Article #3 Summary



edit: Jay Alammar
ithub.io/illustrated-gpt2/)

NLP progress so far



Over 3x reduction in error in 2 years, “superhuman” performance

(slide credit: Stanford CS224N, Chris Manning)

All of these models are Transformer models

ELMo
Oct 2017
Training:
800M words
42 GPU days

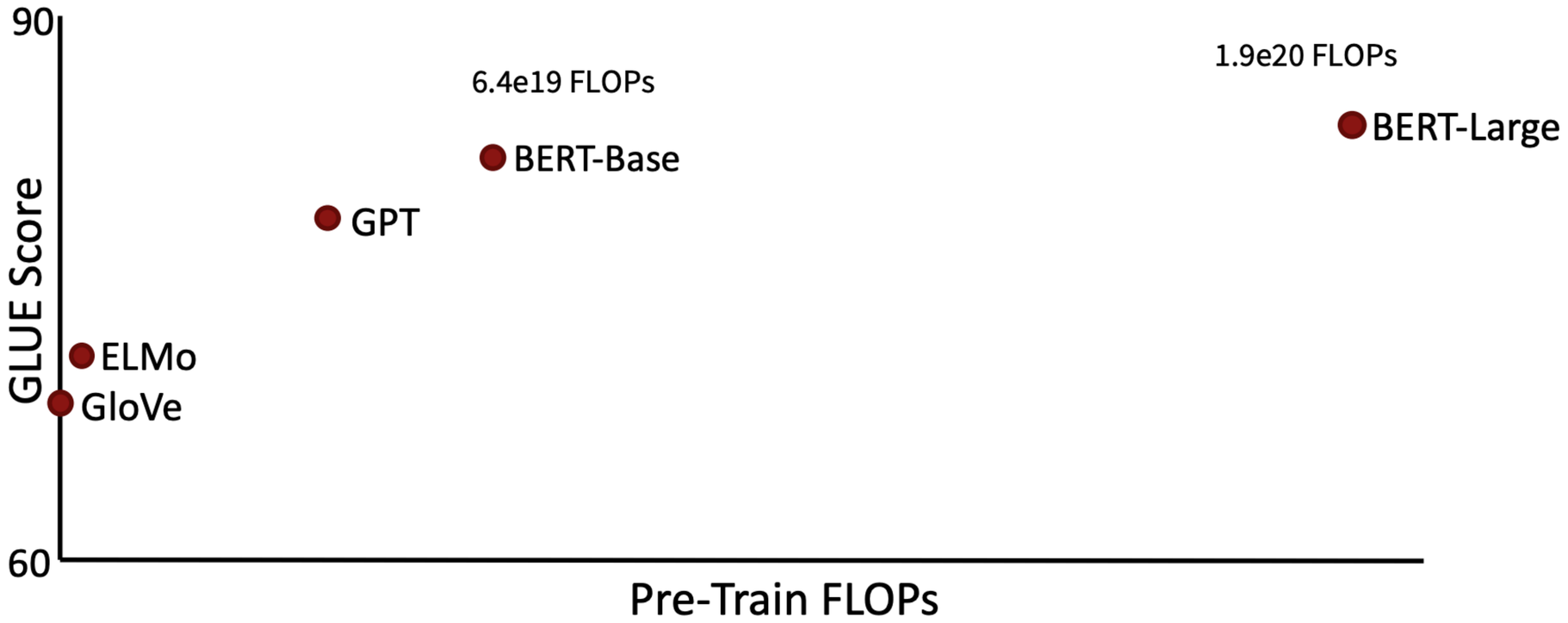
GPT
June 2018
Training
800M words
240 GPU days

BERT
Oct 2018
Training
3.3B words
256 TPU days
~320–560
GPU days

GPT-2
Feb 2019
Training
40B words
~2048 TPU v3
days according to
a reddit thread

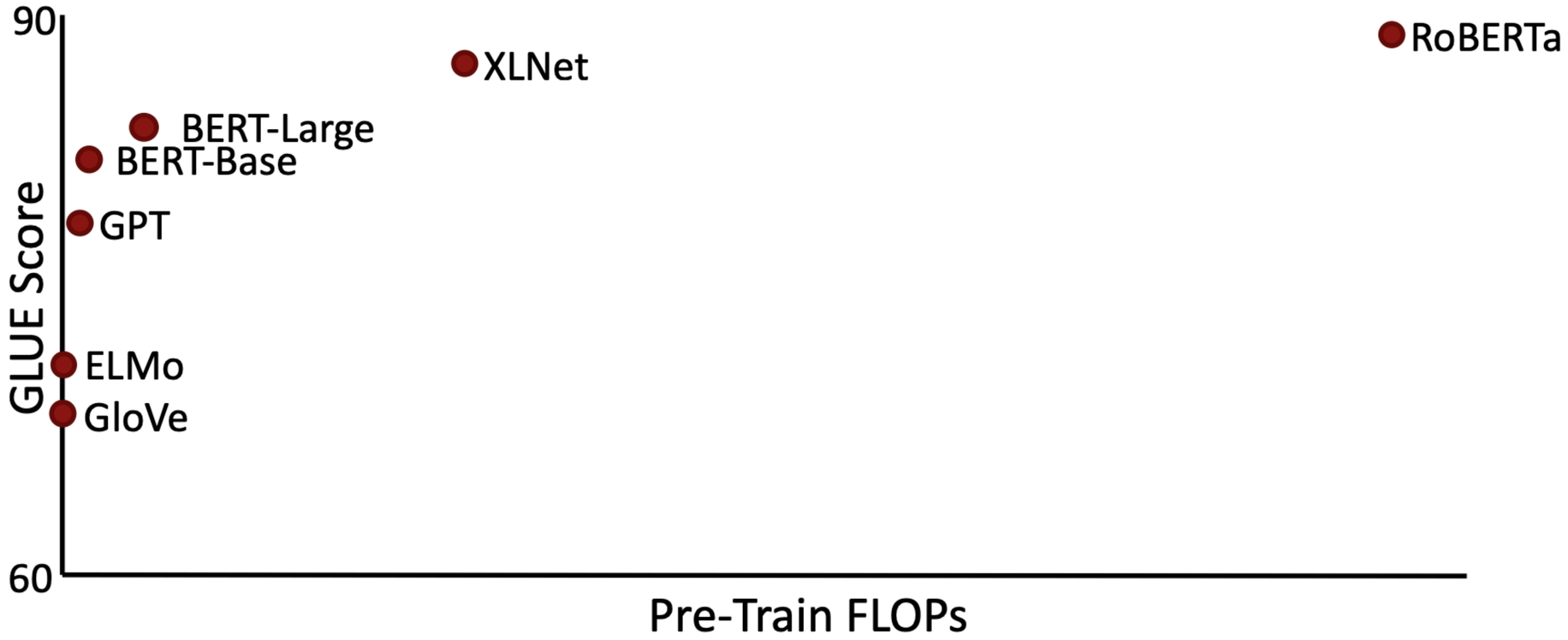
XL-Net,
ERNIE,
Grover
RoBERTa, T5
July 2019—

(slide credit: Stanford CS224N, Chris Manning)



BERT-Large uses 60x more compute than ELMo

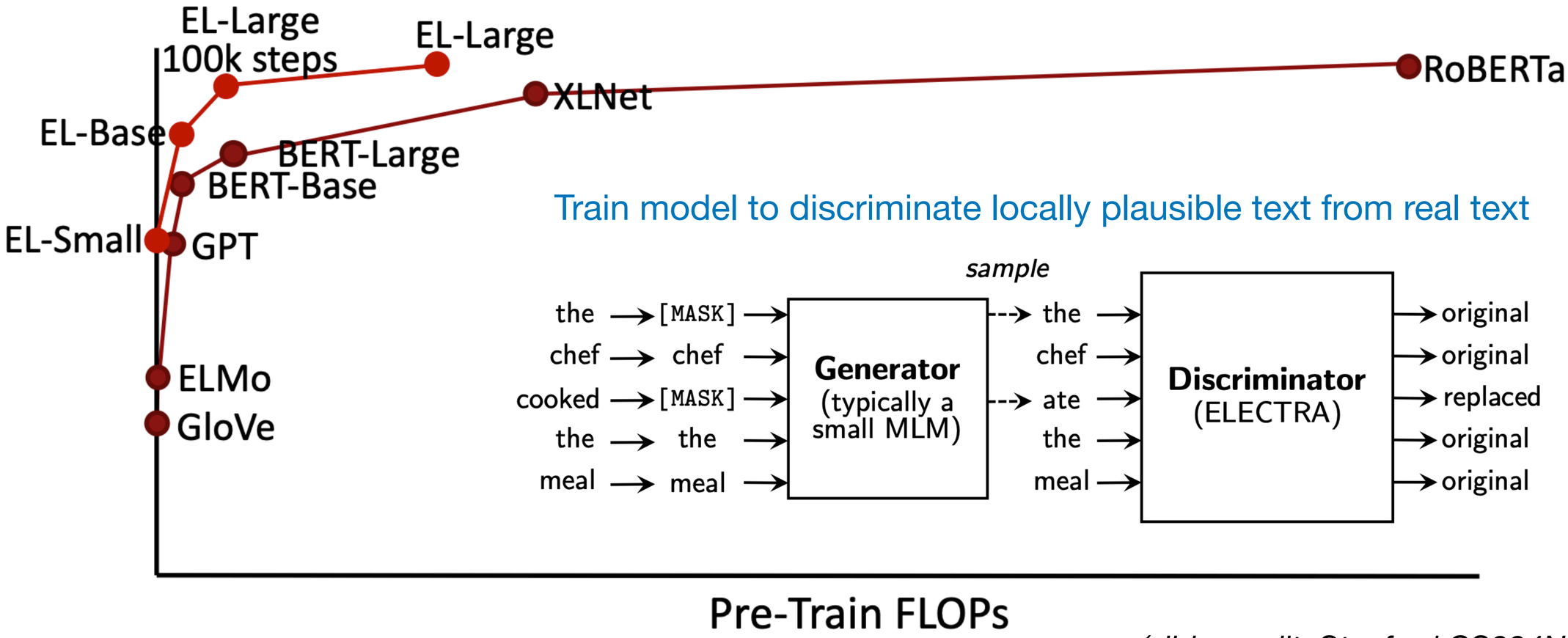
(slide credit: Stanford CS224N, Chris Manning)



RoBERTa uses 16x more compute than BERT-Large

(slide credit: Stanford CS224N, Chris Manning)

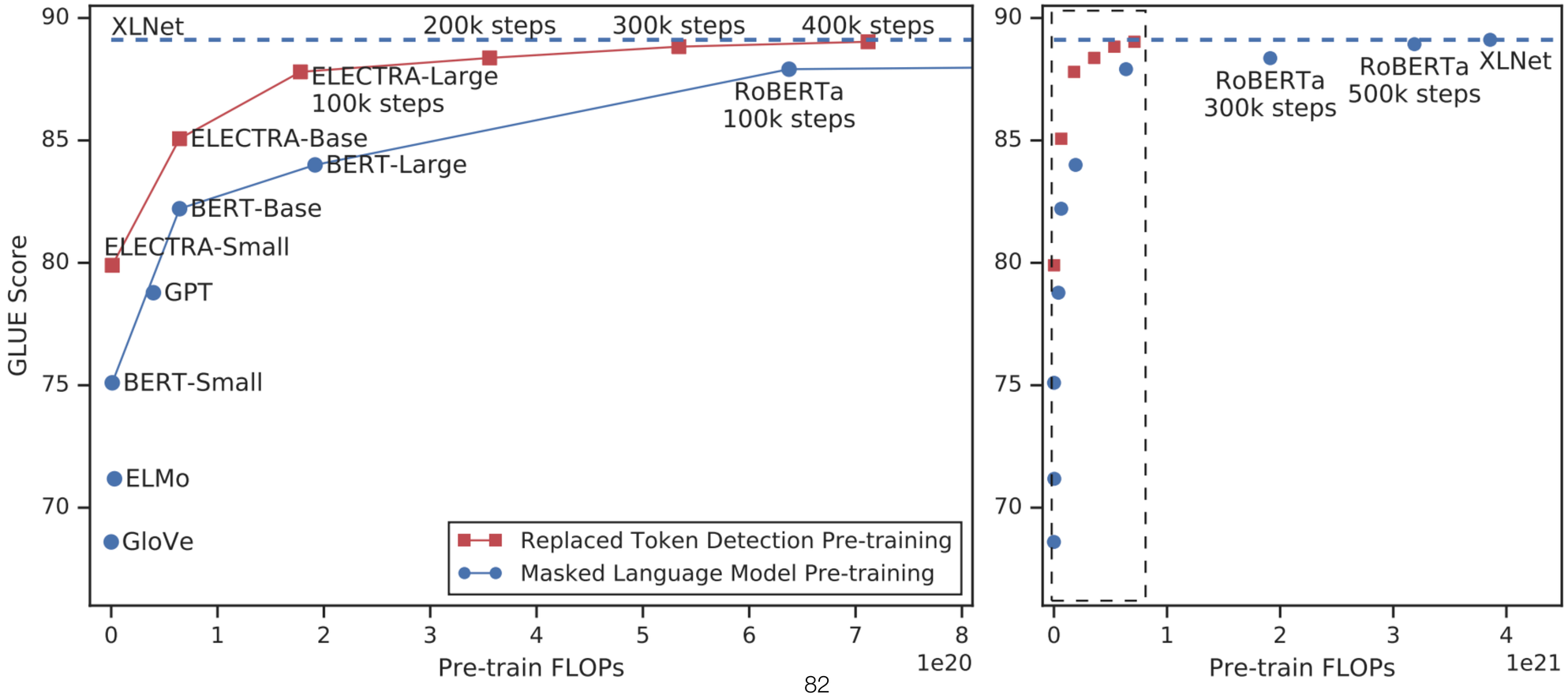
ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators
Clark et al, ICLR 2020



Train model to discriminate locally plausible text from real text

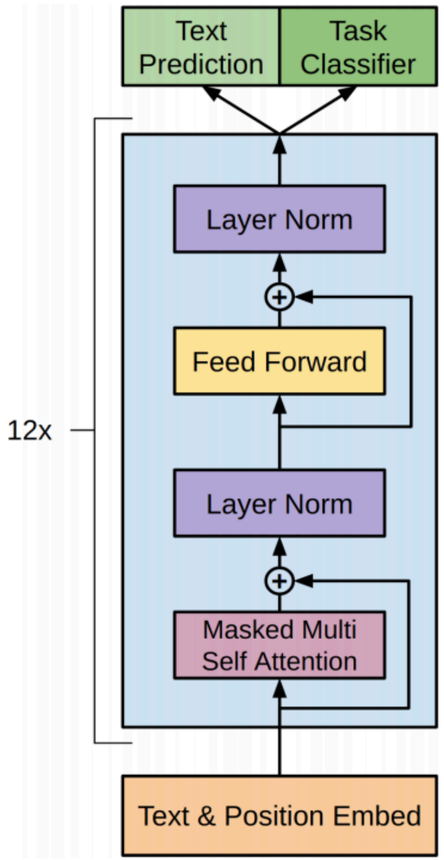
(slide credit: Stanford CS224N, Chris Manning)

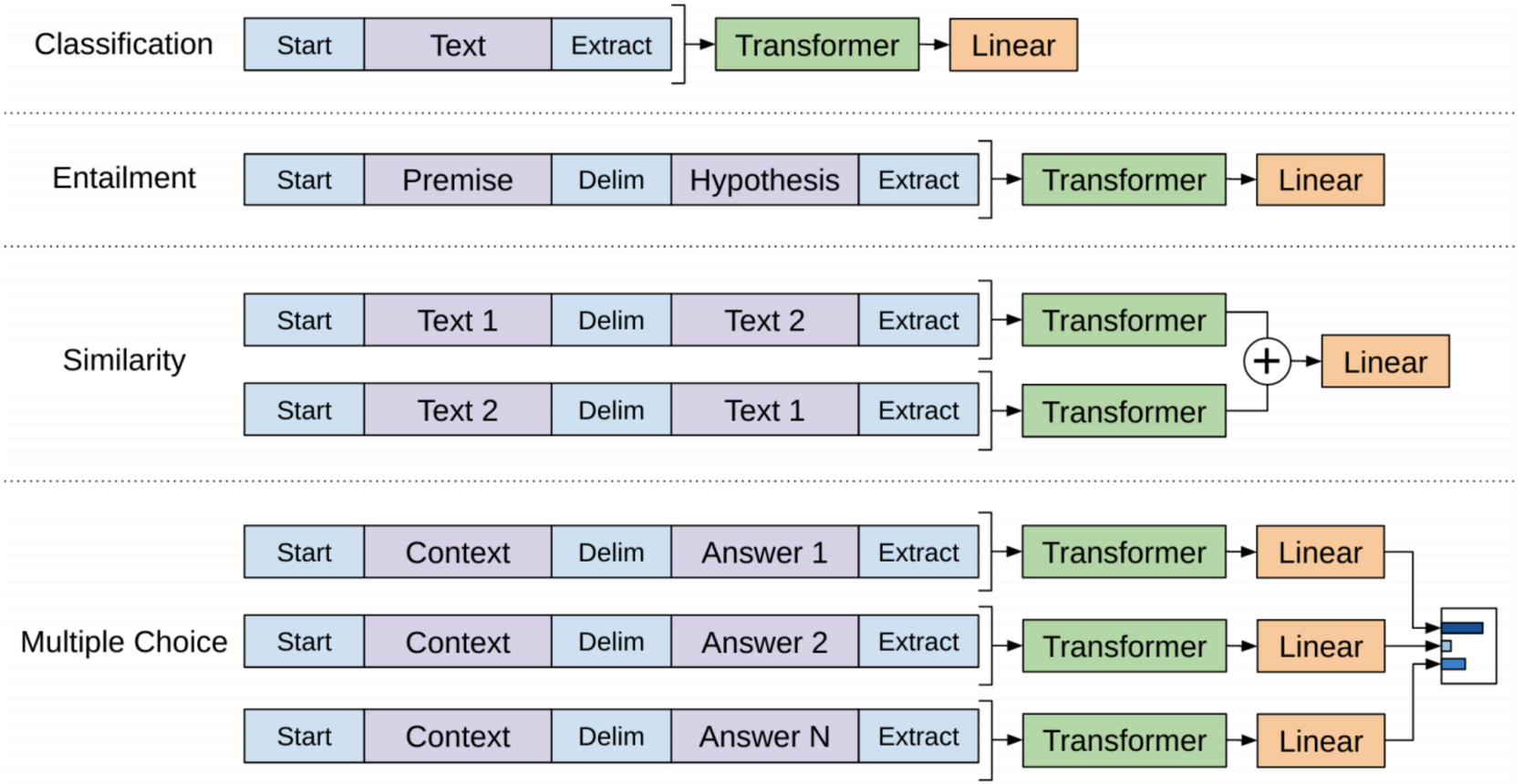
ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators
Clark et al, ICLR 2020



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

GPT





Improving Language Understanding by Generative Pre-Training
Radford et al, OpenAI, 2018

RoBERTa

- Train with more data and for more epochs

	MNLI	QNLI	QQP	RTE	SST	MRPC	CoLA	STS	WNLI	Avg
Single-task single models on dev										
$\text{BERT}_{\text{LARGE}}$	86.6/-	92.3	91.3	70.4	93.2	88.0	60.6	90.0	-	-
$\text{XLNet}_{\text{LARGE}}$	89.8/-	93.9	91.8	83.8	95.6	89.2	63.6	91.8	-	-
RoBERTa	**90.2/90.2**	**94.7**	**92.2**	**86.6**	**96.4**	**90.9**	**68.0**	**92.4**	**91.3**	-

RoBERTa: A Robustly Optimized BERT Pretraining Approach
Liu et al, UW and Facebook, arXiv 2019

XLNET

- Relative embeddings

  Example: John ate a **hot dog**

  How much should **dog** attend to **hot**

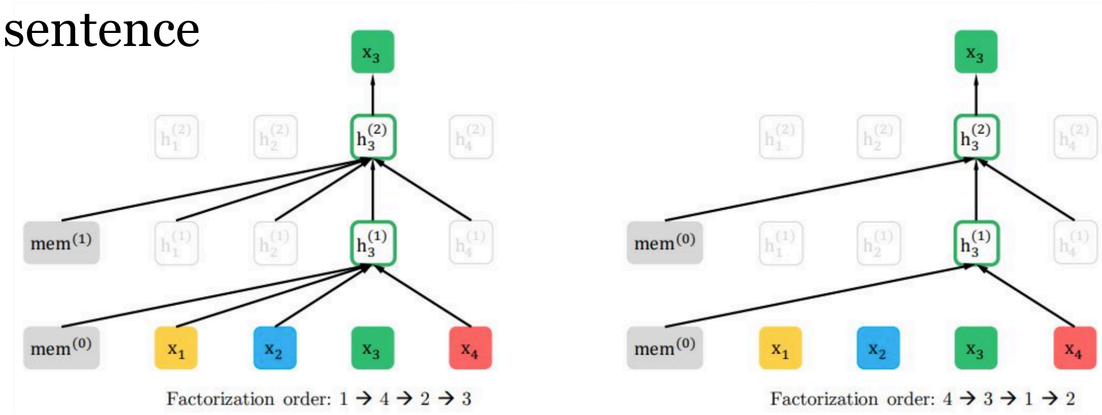
  Absolute: **dog** attend to **hot** (any position) and

  dog (in position 4) attend to **hot** (in position 3)

  Relative: **dog** attend to **hot** (any position) and

  dog attend to **hot** (in previous position)

- Permutation language modeling: Randomly permute the order for every training sentence



XLNet: Generalized Autoregressive Pretraining for Language Understanding
Yang et al, CMU and Google, NeurIPS 2019

XLNET

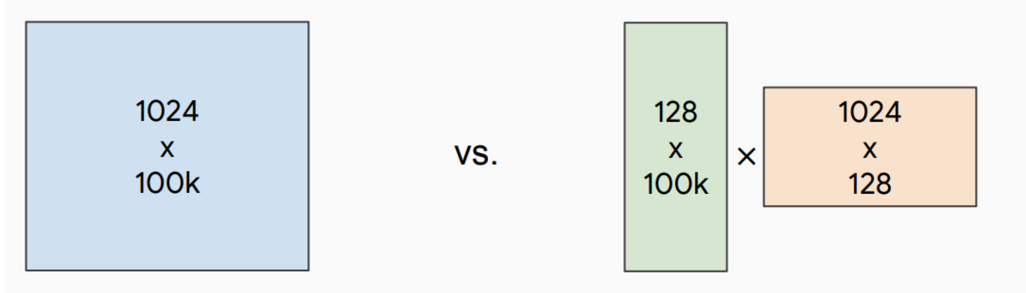
- Relative embeddings
- Permutation language modeling: Randomly permute the order for every training sentence

Model	MNLI	QNLI	QQP	RTE	SST-2	MRPC	CoLA	STS-B
Single-task single models on dev								
BERT [2]	86.6/-	92.3	91.3	70.4	93.2	88.0	60.6	90.0
RoBERTa [21]	90.2/90.2	94.7	92.2	**86.6**	96.4	**90.9**	68.0	92.4
XLNet	**90.8/90.8**	**94.9**	**92.3**	85.9	**97.0**	90.8	**69.0**	**92.5**

XLNet: Generalized Autoregressive Pretraining for Language Understanding
Yang et al, CMU and Google, NeurIPS 2019

ALBERT

- Factorized embedding



- Cross-layer parameter sharing: share all parameters between Transformer layers
- Less parameters, but also takes longer to train:

Model		Parameters	SQuAD1.1	SQuAD2.0	MNLI	SST-2	RACE	Avg	Speedup
BERT	base	108M	90.4/83.2	80.4/77.6	84.5	92.8	68.2	82.3	4.7x
	large	334M	92.2/85.5	85.0/82.2	86.6	93.0	73.9	85.2	1.0
ALBERT	base	12M	89.3/82.3	80.0/77.1	81.6	90.3	64.0	80.1	5.6x
	large	18M	90.6/83.9	82.3/79.4	83.5	91.7	68.5	82.4	1.7x
	xlarge	60M	92.5/86.1	86.1/83.1	86.4	92.4	74.8	85.5	0.6x
	xxlarge	235M	**94.1/88.3**	**88.1/85.1**	**88.0**	**95.2**	**82.3**	**88.7**	0.3x

ALBERT: A Lite BERT for Self-supervised Learning of Language Representations
Lan et al, Google and TTIC, ICLR 2020