CS 546 – Advanced Topics in NLP

Dilek Hakkani-Tür

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

CONVAI

CDS

Siebel School of Computing and Data Science

Readings

- Radford et al., “Improving Language Understanding by Generative Pre-Training”, 2018.
- Devlin et al., “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”, *NAACL-HLT*, 2019.
- Lewis et al., “BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension”, arXiv preprint: 1910.13461, 2019.
- Raffel et al., “Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer”, JMLR, 2020.

Topics for Today

- Tokenization and Byte Pair Encoding (BPE)
- Decoder-Only Models: OpenAI GPT
- Encoder-Only Models: BERT
- Encoder-Decoder Models: BART, T5
- Parameter-Efficient Fine-Tuning (PEFT)

  Adapters, Prefix Tuning, LoRA

Tokenization and Byte-Pair Encoding (BPE)

- The input to all the models: tokens.
- How to define the token vocabulary?
- Inference usually encounters previously unseen words.
- Characters as the input unit, |V| = #characters in a language
  - Could be too short to capture semantics, though we have seen recent success.
- Could word pieces be better tokens?
  - Interaction, interdependency, international: inter-
  - Training, chaining: -ing

Byte-Pair Encoding (BPE)

- A simple form of compression
- The most common byte pair is replaced with a byte not appearing in the data.
- A table of replacements is required to rebuild the data.
- Wikipedia example:

aaabdaaabac → ZabdZabac, Z=aa
→ ZYdZYac, Y=ab, Z=aa
→ XdXac, X=ZY, Y=ab, Z=aa

aaab d aaab a c V={aaab, a, b, c, d}

The vocabulary is automatically determined.

BPE Examples


```
[ ]  from transformers import OpenAIGPTTokenizer
     text_example = "Simplest internationalization examples"
     tokenizer = OpenAIGPTTokenizer.from_pretrained("openai-gpt")
     tokenizer.tokenize(text_example)
```

```
['simplest</w>', 'intern', 'ation', 'alization</w>', 'examples</w>']
```

```
[ ]  from transformers import BertTokenizer
     tokenizer = BertTokenizer.from_pretrained("bert-base-uncased")
     tokenizer.tokenize(text_example)
```

```
['simplest', 'international', '##ization', 'examples']
```

Topics for Today

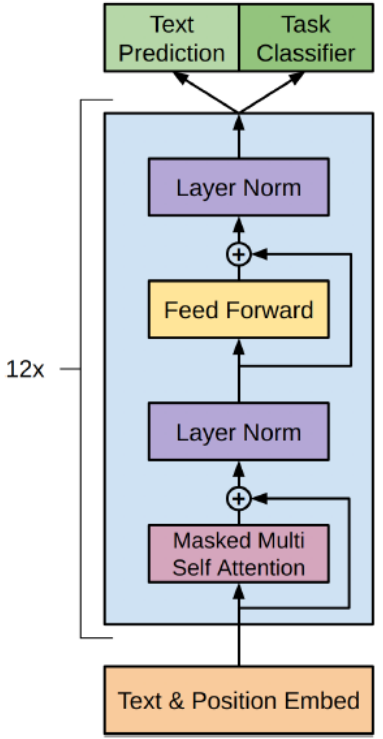
- Tokenization and Byte Pair Encoding (BPE)
- Decoder-Only Models: OpenAI GPT
- Encoder-Only Models: BERT
- Encoder-Decoder Models: BART, T5
- Parameter-Efficient Fine-Tuning (PEFT)

  Adapters, Prefix Tuning, LoRA

OpenAI GPT

- Built from transformer decoder blocks
- Goals:
  - leverage linguistic information from unlabeled data
  - Learn a universal representation that transfer with little adaptation to a wide range of tasks

- Two-stage training procedure:
  - Generative, unsupervised **pre-training**, LM objective
  - Discriminative, supervised **fine-tuning**, using a task-specific objective



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

OpenAI GPT – Unsupervised Pre-training

- Given a corpus of tokens: $\mathcal{U} = \{u_1, \ldots, u_n\}$
- Uses the standard LM objective to maximize likelihood:

$$L_1(\mathcal{U}) = \sum_i \log P(u_i | u_{i-k}, \ldots, u_{i-1}; \Theta)$$

Context window of size k

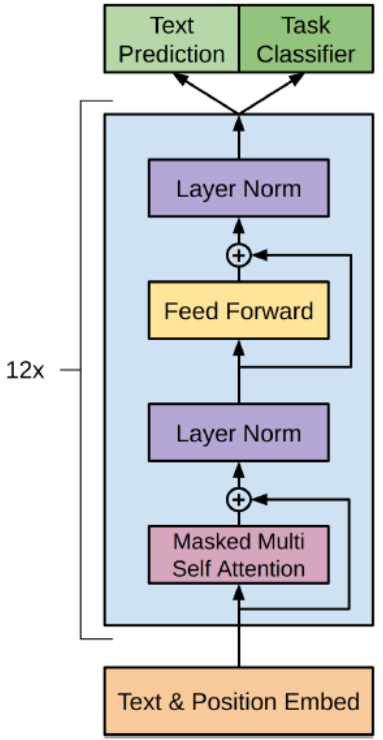
$$h_0 = UW_e + W_p$$

Token embedding matrix
Position embedding matrix

$$h_l = \texttt{transformer\_block}(h_{l-1}) \forall\, l \in [1, n]$$

$$P(u) = \texttt{softmax}(h_n W_e^T)$$

- the context vector of tokens: $U = (u_{-k}, \ldots, u_{-1})$
- the number of layers: n
- the token/position embedding matrix: W_e / W_p



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

OpenAI GPT – Supervised Fine-tuning

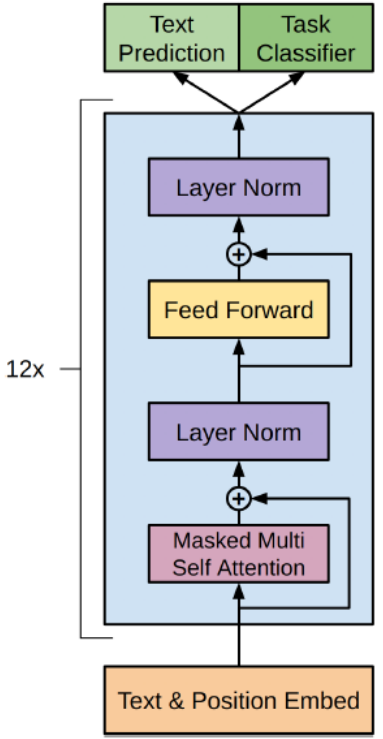
- Labeled dataset, $\mathcal{C}$:
  sequence of input tokens $x_1, \ldots, x_m$, and label y

- Classification Objective:

$$L_2(\mathcal{C}) = \sum_{(x,y)} \log P(y|x^1, \ldots, x^m)$$

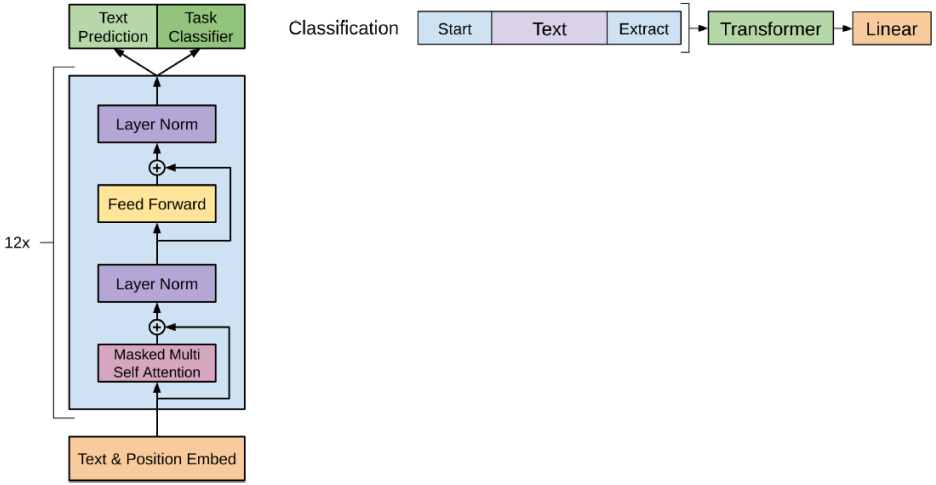
- Additional LM objective:

$$L_3(\mathcal{C}) = L_2(\mathcal{C}) + \lambda * L_1(\mathcal{C})$$



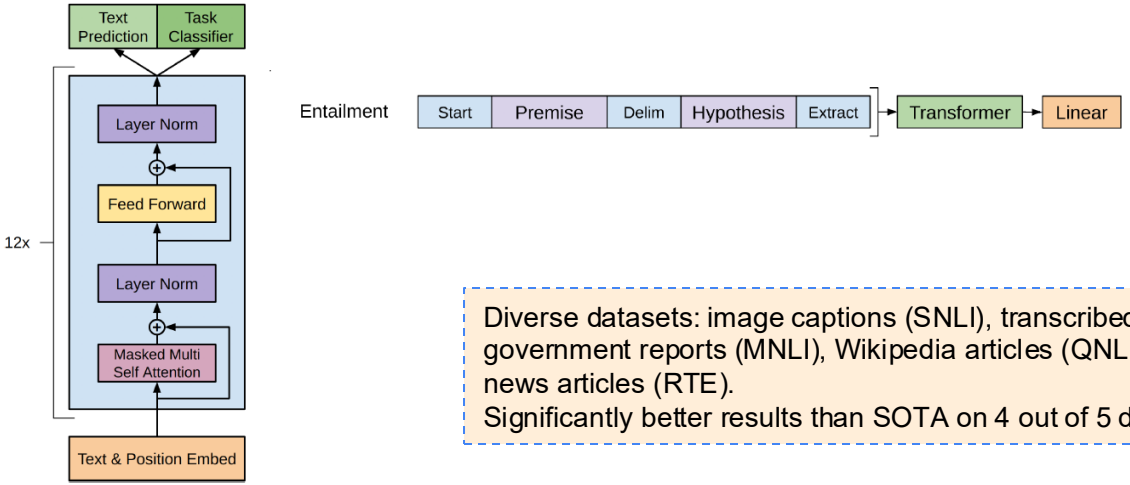
OpenAI GPT – Supervised Fine-tuning for Classification

- Stanford Sentiment Treebank, binary sentiment
- The Corpus of Linguistic Acceptability (CoLA): whether a sentence is grammatical or not



OpenAI GPT – Supervised Fine-tuning for Natural Language Inference

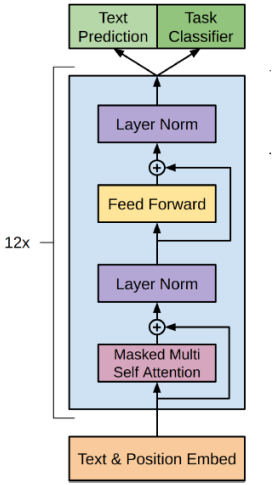
- Input: pair of sentences (premise and hypothesis),
- Output: their relationship: entailment, contradiction, neutral

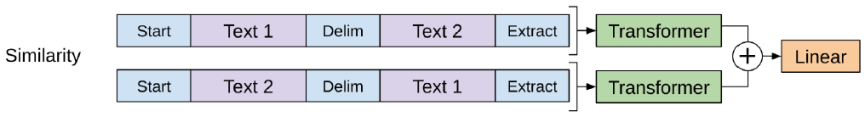


Diverse datasets: image captions (SNLI), transcribed speech, popular fiction, government reports (MNLI), Wikipedia articles (QNLI), science exams (SciTail) and news articles (RTE).
Significantly better results than SOTA on 4 out of 5 datasets.

OpenAI GPT – Supervised Fine-tuning for Semantic Similarity

- Paraphrase Detection
- Two sentences, Text1 and Text2, are semantically equivalent or not.
- No ordering (unlike entailment)

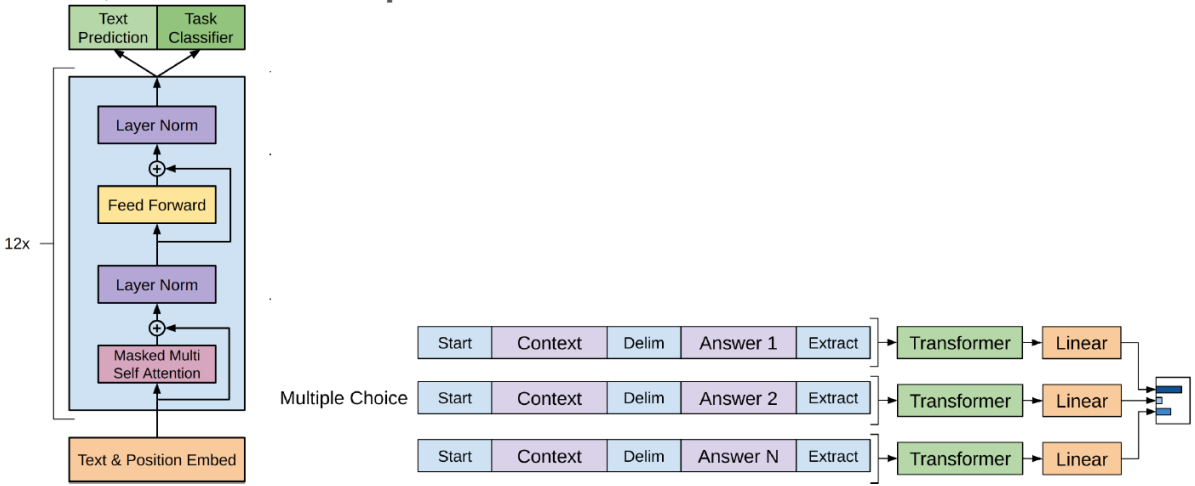




Challenges include rephrasing of concepts, understanding negation, and handling syntactic ambiguity.
Datasets: Microsoft Paraphrase corpus (MRPC) (news), Quora Question Pairs (QQP), and Semantic Textual Similarity benchmark (STS-B)

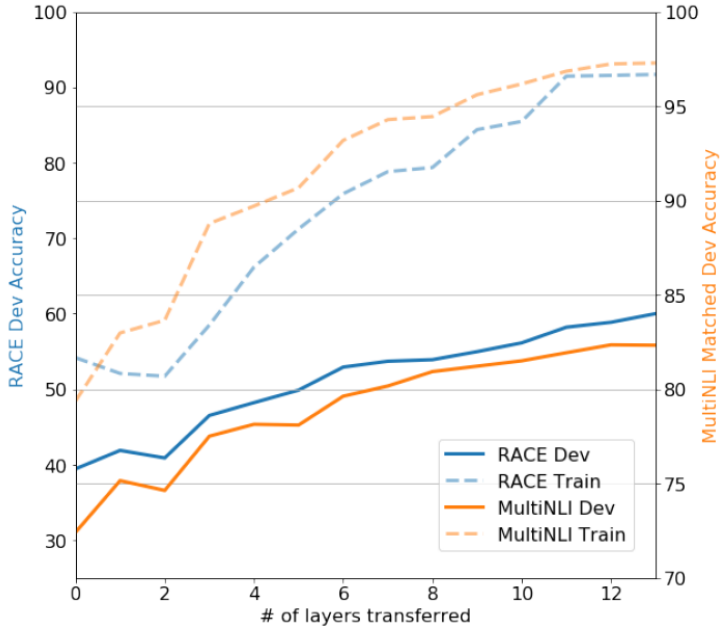
OpenAI GPT – Supervised Fine-tuning for Question Answering and Commonsense Reasoning

- RACE: English passages with associated questions from middle/high school exams
- Story Cloze Test: selecting the correct ending to multi-sentence stories, from two options.



OpenAI GPT Experiments Analysis

- Pre-training: Books corpus (7K unique, unpublished books, ~1B words)
- # of layers transferred.



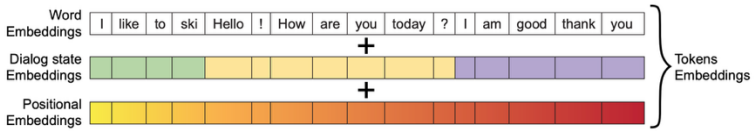
OpenAI GPT Experiments Analysis (cont.)

- Pre-training: Books corpus (7K unique, unpublished books, ~1B words)
- Ablation Studies:

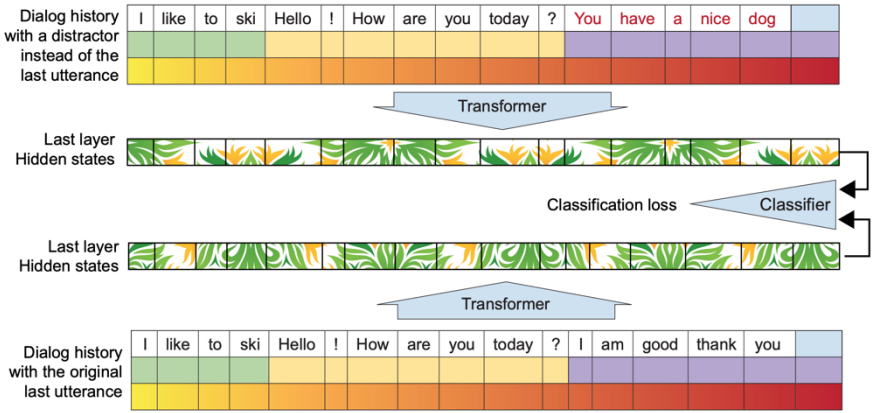
Method	Avg. Score	CoLA (mc)	SST2 (acc)	MRPC (F1)	STSB (pc)	QQP (F1)	MNLI (acc)	QNLI (acc)	RTE (acc)
Transformer w/ aux LM (full)	74.7	45.4	91.3	82.3	82.0	**70.3**	**81.8**	**88.1**	**56.0**
Transformer w/o pre-training	59.9	18.9	84.0	79.4	30.9	65.5	75.7	71.2	53.8
Transformer w/o aux LM	**75.0**	**47.9**	**92.0**	**84.9**	**83.2**	69.8	81.1	86.9	54.4
LSTM w/ aux LM	69.1	30.3	90.5	83.2	71.8	68.1	73.7	81.1	54.6

Fine-tuning GPT for Response Generation: TransferTransfo

- NeurIPS 2018, ConvAI 2 challenge
- Fine-tune pre-trained GPT for response generation in conversational systems.
- Fine-tuning is done by optimizing a combination of two loss functions:
  - a language modeling loss, and
  - a next-utterance classification loss



[Wolf et al., TransferTransfo: A Transfer Learning Approach for Neural Network Based Conversational Agents. 2018.]



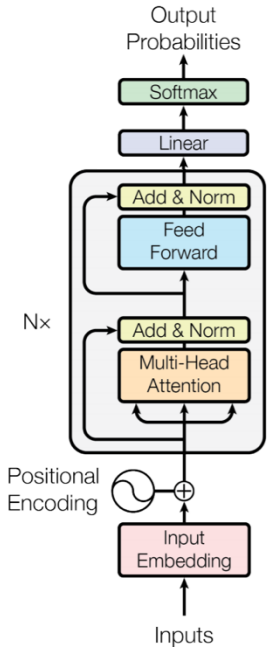
Topics for Today

- Tokenization and Byte Pair Encoding (BPE)
- Decoder-Only Models: OpenAI GPT
- Encoder-Only Models: BERT
- Encoder-Decoder Models: BART, T5
- Parameter-Efficient Fine-Tuning (PEFT)

  Adapters, Prefix Tuning, LoRA

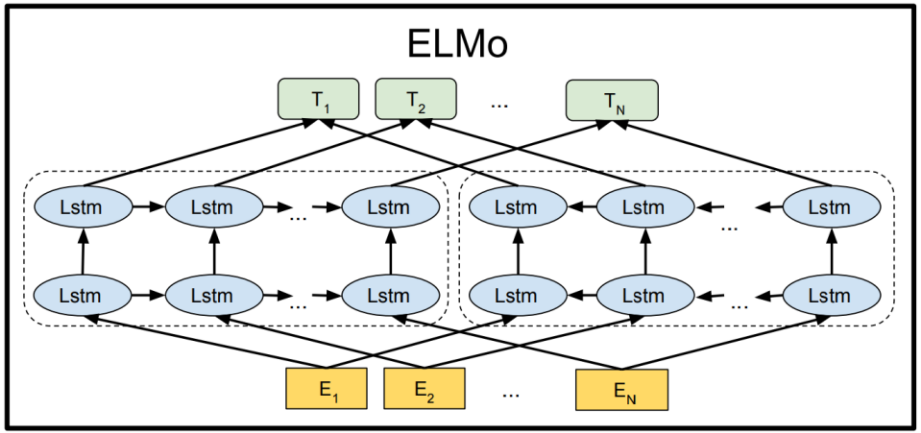
BERT: **B**idirectional **E**ncoder **R**epresentations from **T**ransformers

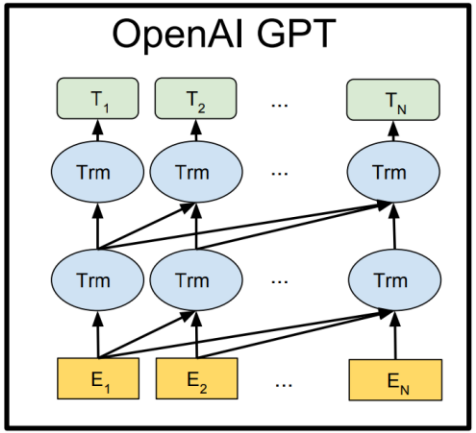
- Built from transformer encoder blocks only.
- **Idea:** contextualized word representations
  - Learn word vectors using long contexts using Transformer instead of LSTM
  - multi-layer bidirectional Transformer encoder.

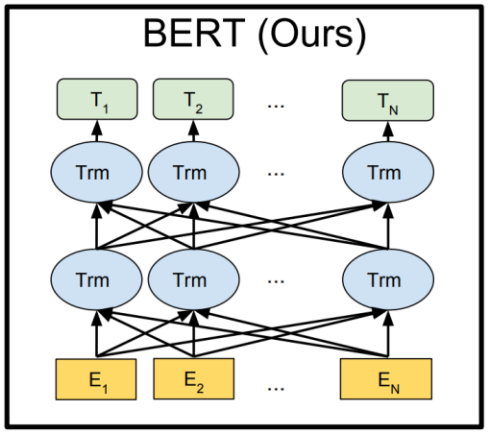


Devlin et al., "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding", in *NAACL-HLT*, 2019.

BERT Architecture (compared to earlier models)







Devlin et al., "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding", in *NAACL-HLT*, 2019.

BERT Pre-training: Masked Language Model

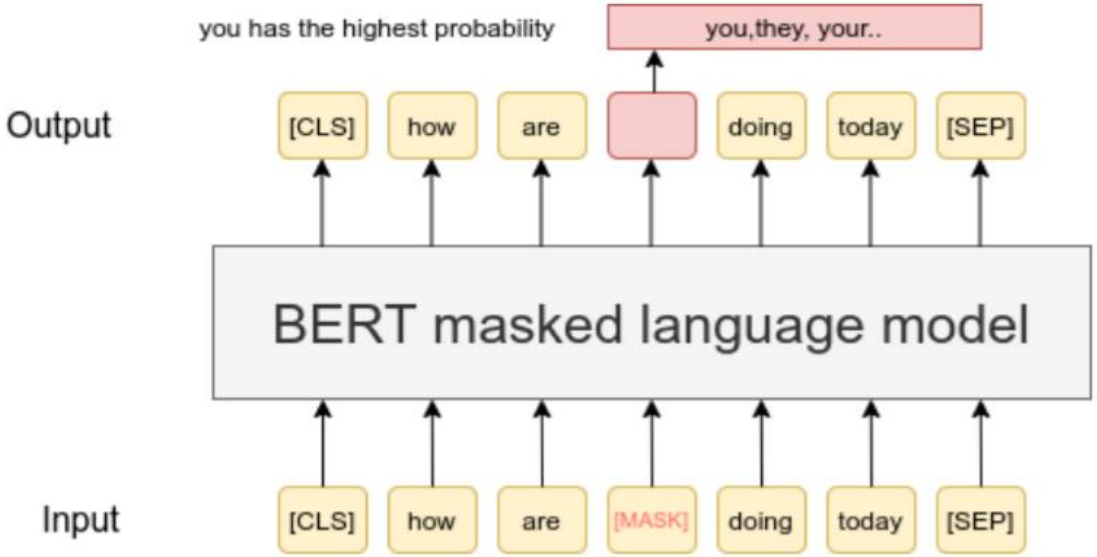


Figure from https://sbert.net/examples/sentence_transformer/unsupervised_learning/MLM/README.html

BERT Pre-training: Next Sentence Prediction

- **Idea:** modeling ***relationship*** between sentences
  - Many NLP tasks (i.e., QA) are based on understanding inter-sentence relationship

$$\text{Input} = \texttt{[CLS] the man went to [MASK] store [SEP]}$$
$$\texttt{he bought a gallon [MASK] milk [SEP]}$$
$$\text{Label} = \texttt{IsNext}$$

$$\text{Input} = \texttt{[CLS] the man [MASK] to the store [SEP]}$$
$$\texttt{penguin [MASK] are flight ##less birds [SEP]}$$
$$\text{Label} = \texttt{NotNext}$$

Devlin et al., "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding", in *NAACL-HLT*, 2019.

BERT Pre-training: Next Sentence Prediction (cont.)

- Idea: modeling ***relationship*** between sentences

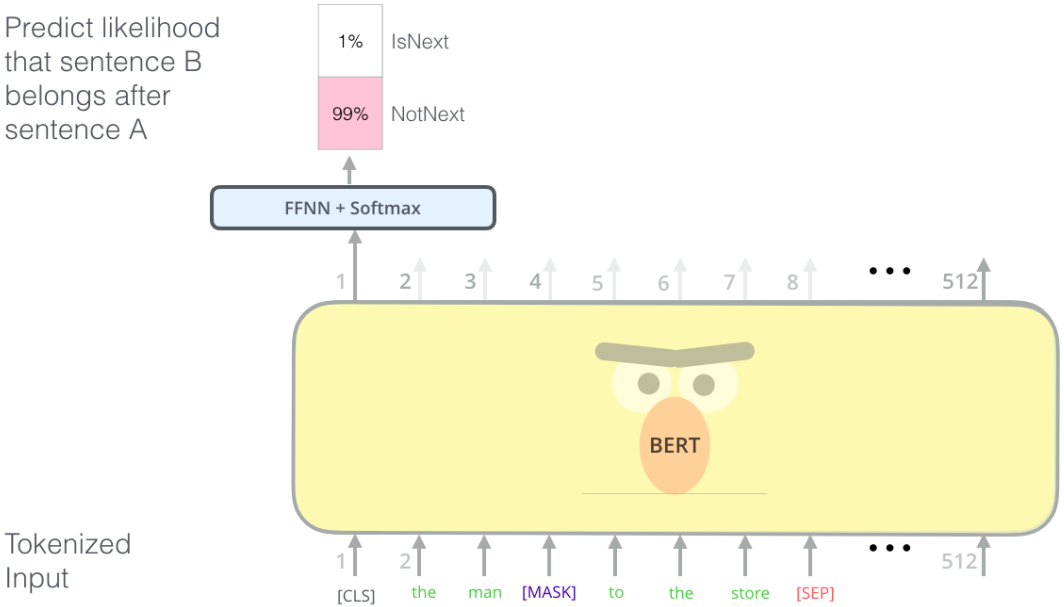
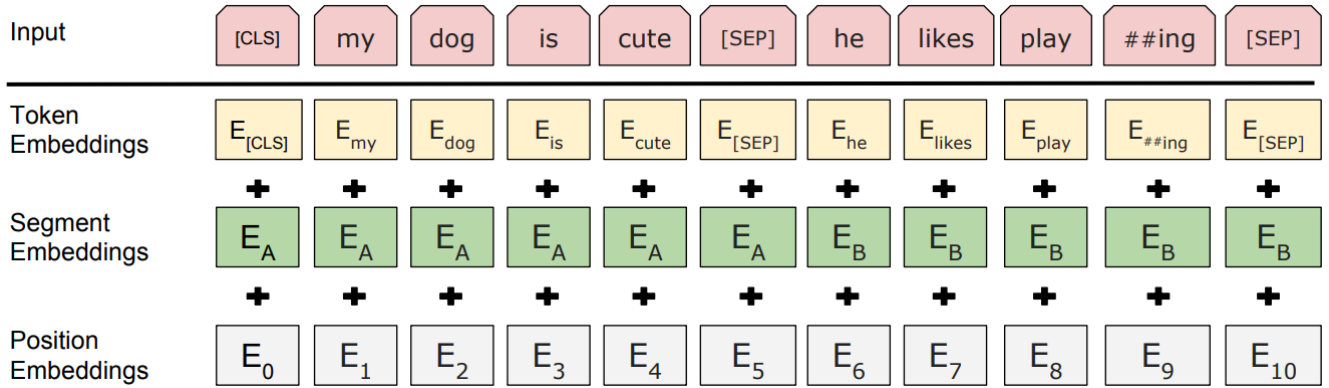


Figure from http://jalammar.github.io/illustrated-bert/

BERT – Input Representation

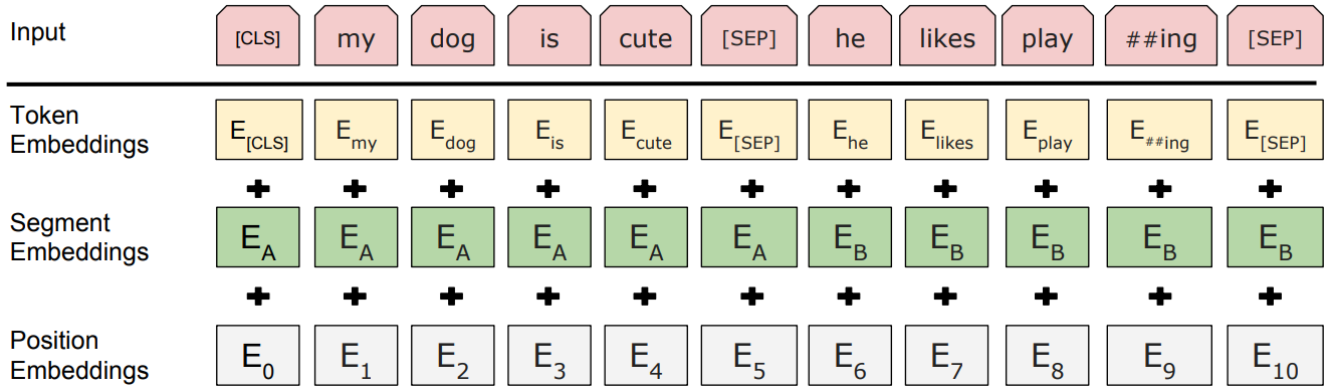
- Special tokens:
  - [CLS]: added to the beginning of the text.
  - [SEP]: added to the end. Also used as a separator.



Devlin et al., "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding", in *NAACL-HLT*, 2019.

BERT – Input Representation (cont.)

- Input embeddings contain
  - Word-level token embeddings
  - Sentence-level segment embeddings
  - Position embeddings



Devlin et al., "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding", in *NAACL-HLT*, 2019.

BERT – Training

- Training data: Wikipedia + BookCorpus
- 2 BERT models
  - BERT-Base: 12-layer, 768-hidden, 12-head
  - BERT-Large: 24-layer, 1024-hidden, 16-head

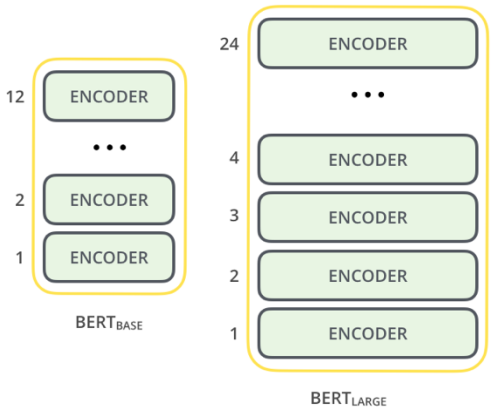
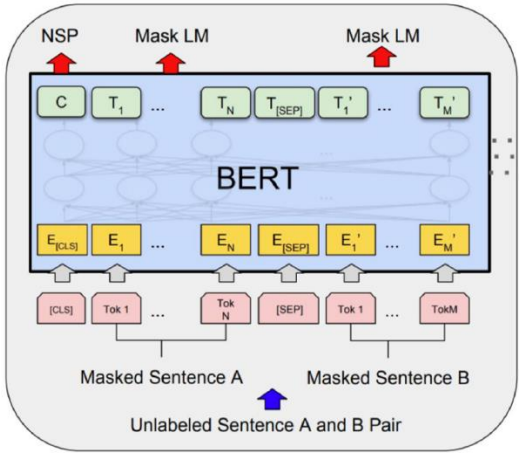


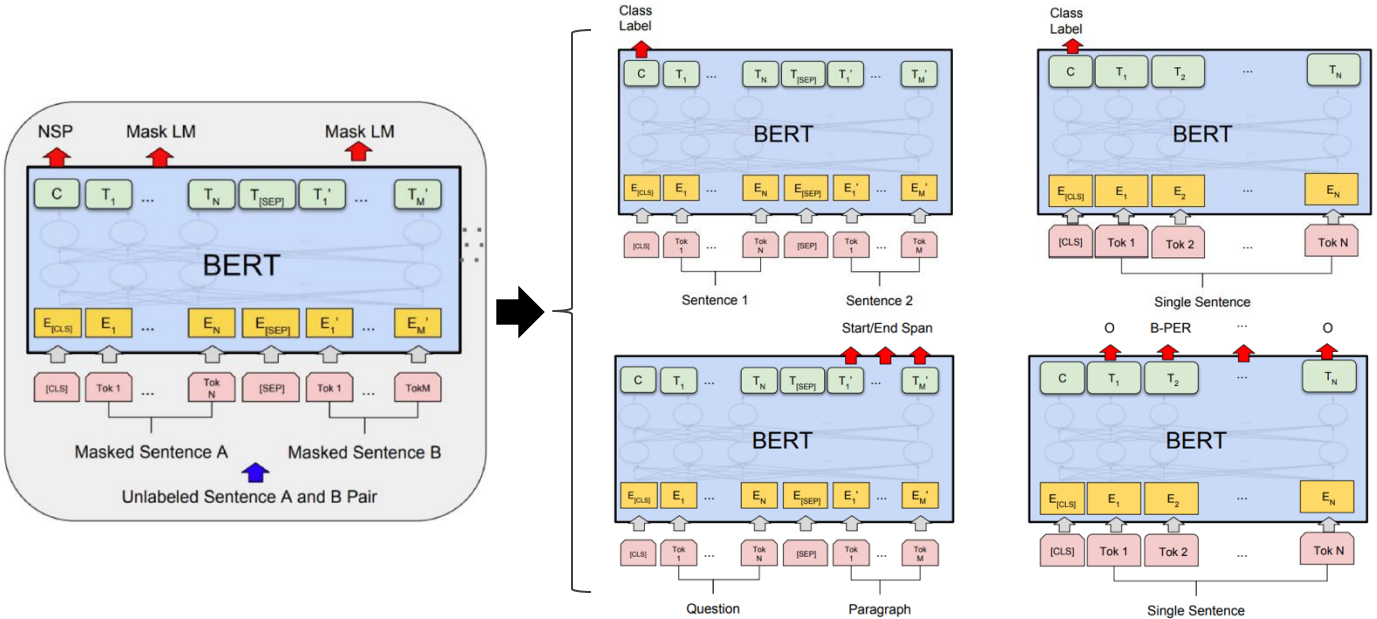
Figure from http://jalammar.github.io/illustrated-bert/



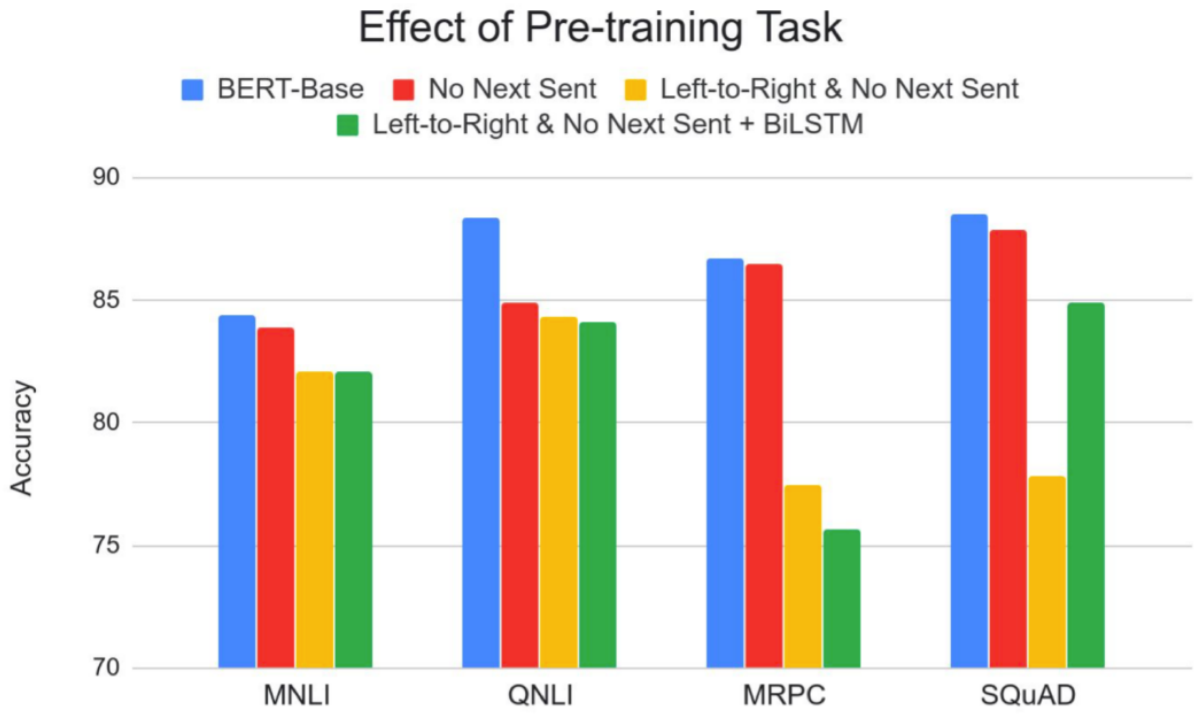
Hyperparameters that determine the model size: number of layers, number of attention heads and hidden layer size.

BERT – Fine-tuning

- Idea: simply learn a classifier/tagger built on the top layer for each target task
- Mainly natural language understanding tasks



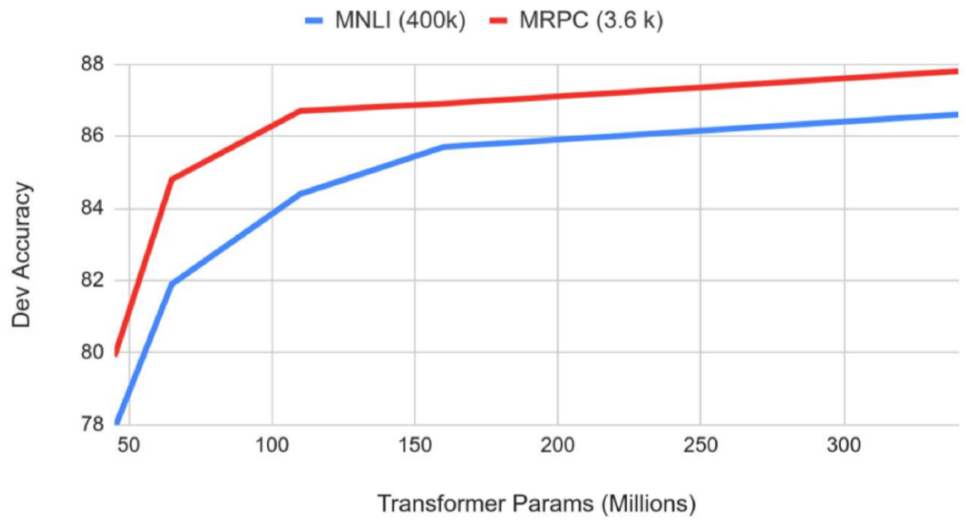
BERT Fine-tuning results – Effect of Pre-training tasks



Devlin et al., "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding", in *NAACL-HLT*, 2019.

BERT Fine-tuning results – Effect of Model sizes

- Improving performance by increasing model size

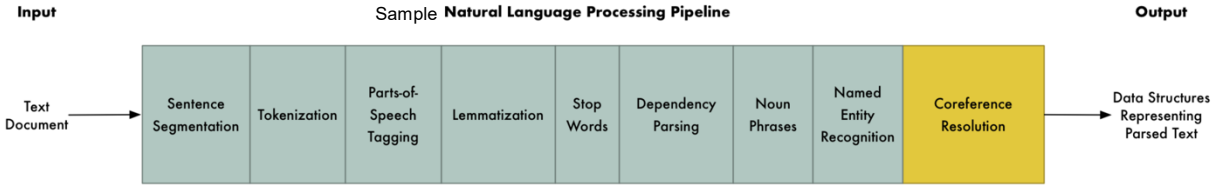


	$BERT_{base}$	$BERT_{large}$
# layers	12	24
hidden layer size	768	1024
# attention heads	12	16
Total # parameters	110M	340M

Devlin et al., "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding", in *NAACL-HLT*, 2019.

What is BERT Learning?

- “Probing LMs”: representing language in a satisfying way?
- (Peters et al., 2018b): lower layers of a language model encode more local syntax while higher layers capture more complex semantics.
- Common components of a traditional NLP pipeline, such as part-of-speech (POS), constituents (Consts.), dependencies (Deps.), entities, semantic role labeling (SRL), coreference (Coref.), semantic proto-roles (SPR; Reisinger et al., 2015), and relation classification (SemEval).



Tenney et al., BERT Rediscovers the Classical NLP Pipeline. ACL 2019.

What is BERT Learning? (cont.)

- **Edge Probing:** how well information about linguistic structure can be extracted from a pre-trained encoder
- Classifier receives spans s_1 and (optionally) s_2 and must predict the relation, given only the per-token contextual vectors within the target spans.
- Given input tokens T = [t_0, t_1, … , t_n], the encoder produces a set of layer activations, $H^{(0)}$,$H^{(1)}$, …, $H^{(L)}$, where

$$H^{(\ell)} = [\mathbf{h}_0^{(\ell)}, \mathbf{h}_1^{(\ell)}, \ldots, \mathbf{h}_n^{(\ell)}]$$

Activation vectors of the l^{th} encoder later

- **Scalar mixing weights** tell us which layers, in combination, are most relevant when a probing classifier, as in ELMo work.
- Higher weights seen as evidence that the corresponding layers have more information for a given task.

$$\mathbf{h}_{i,\tau} = \gamma_\tau \sum_{\ell=0}^{L} s_\tau^{(\ell)} \mathbf{h}_i^{(\ell)}$$

Weights are learned from data!

What is BERT Learning? (cont.)

- Center of Gravity

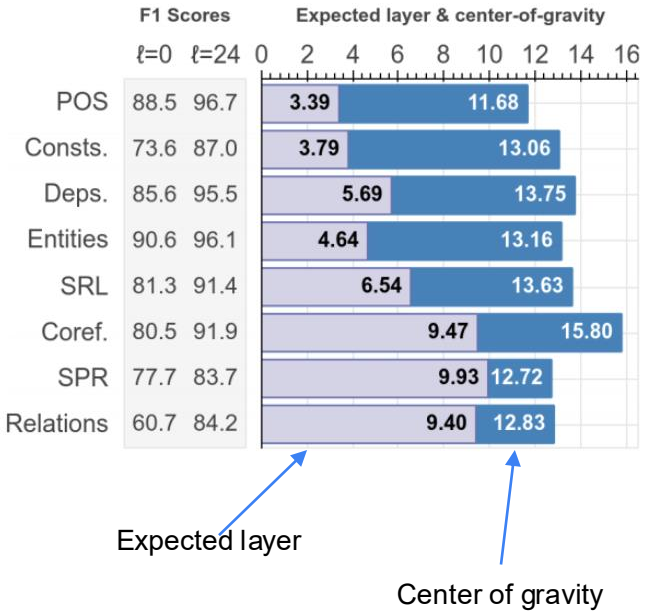
$$\bar{E}_s[\ell] = \sum_{\ell=0}^{L} \ell \cdot s_\tau^{(\ell)}$$

- Cumulative Scoring

$$\Delta_\tau^{(\ell)} = \text{Score}(P_\tau^{(\ell)}) - \text{Score}(P_\tau^{(\ell-1)})$$

- Expected layer

$$\bar{E}_\Delta[\ell] = \frac{\sum_{\ell=1}^{L} \ell \cdot \Delta_\tau^{(\ell)}}{\sum_{\ell=1}^{L} \Delta_\tau^{(\ell)}}$$



$\{P_\tau^{(\ell)}\}_\ell$: a set of classifiers that use scalar mixing **to attend to layer ℓ as well as all previous layers.**

$P_\tau^{(\ell)}$: uses only the first ℓ layers.

What is BERT Learning? (cont.)

- Center of Gravity $\bar{E}_s[\ell] = \sum_{\ell=0}^{L} \ell \cdot s_\tau^{(\ell)}$
- Cumulative Scoring

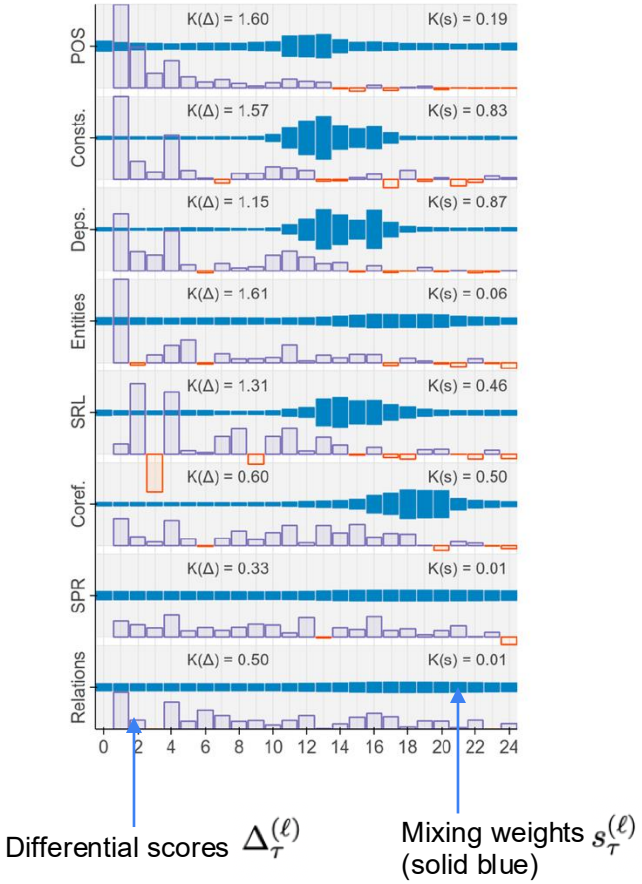
$$\Delta_\tau^{(\ell)} = \text{Score}(P_\tau^{(\ell)}) - \text{Score}(P_\tau^{(\ell-1)})$$

- Expected layer

$$\bar{E}_\Delta[\ell] = \frac{\sum_{\ell=1}^{L} \ell \cdot \Delta_\tau^{(\ell)}}{\sum_{\ell=1}^{L} \Delta_\tau^{(\ell)}}$$

$\{P_\tau^{(\ell)}\}_\ell$: a set of classifiers that use scalar mixing **to attend to layer ℓ as well as all previous layers.**

$P_\tau^{(\ell)}$: uses only the first ℓ layers.



What does BERT look at?

- Analyzing the attention mechanisms of pre-trained models.
- How much a particular word will be weighted when computing the next representation for the current word?
- BERT base: 12 layers, 12 attention heads.
- Analysis over 1000 random Wikipedia segments, at most 128 tokens each. Input:
  - [CLS] <paragraph-1> [SEP] <paragraph-2> [SEP]

Clark et al., What Does BERT look at? An Analysis of BERT's attention. BlackBoxNLP @ ACL 2019.

What does BERT look at? (cont.)

Head <layer>-<head-number>

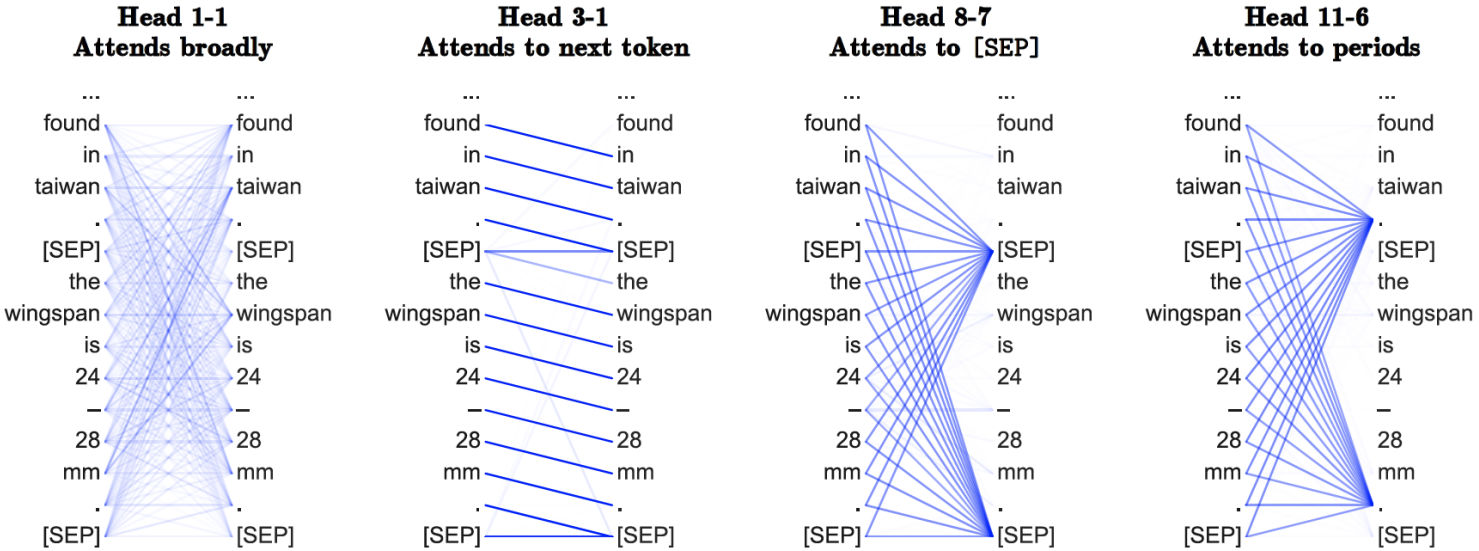


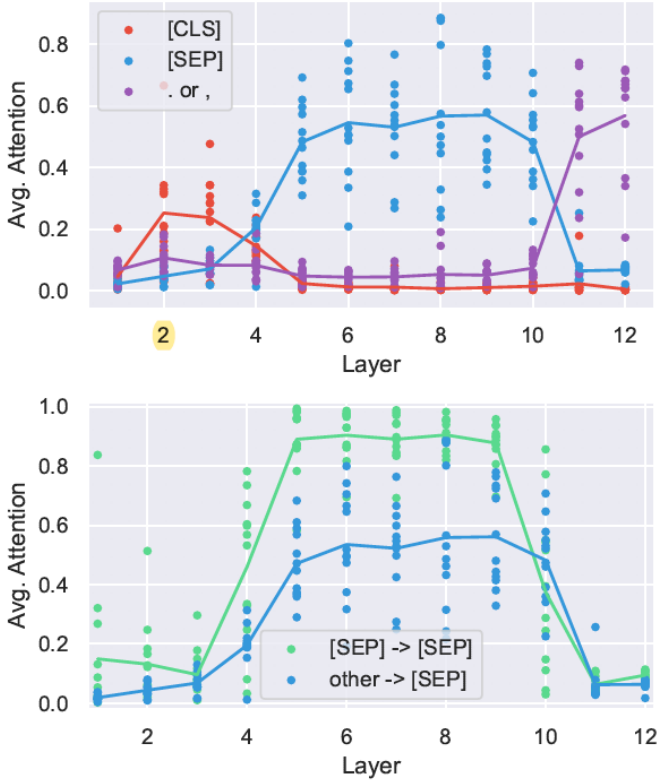
Figure 1: Examples of heads exhibiting the patterns discussed in Section 3. The darkness of a line indicates the strength of the attention weight (some attention weights are so low they are invisible).

What does BERT look at? (cont.)

- Relative position: how often BERTs heads attend to
  - current token?
    - little attention
  - previous or next token?
    - heads that specialize to attending to previous or next tokens, especially in the lower layers

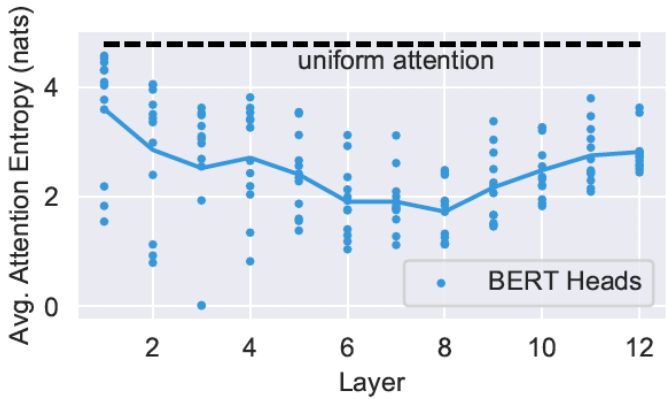
What does BERT look at? (cont.)

- Attention to separator tokens:
  - Lower layer heads attend to [CLS], middle to [SEP], higher layer heads attend to periods and commas (often more than 50%)
  - Heads attend to [SEP] tokens even more when the current token is [SEP] itself.



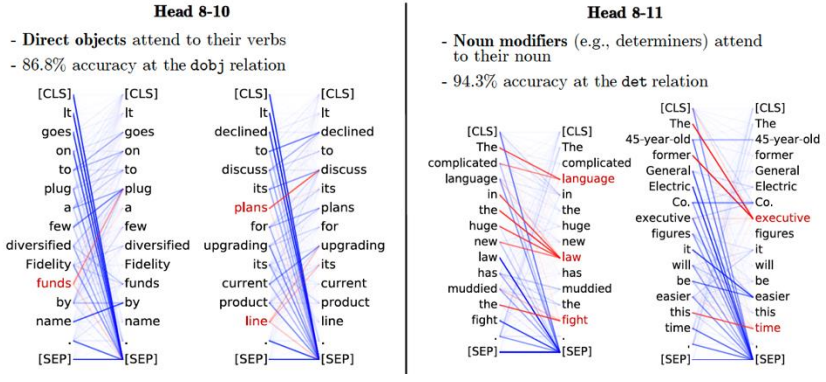
What does BERT look at? (cont.)

- Focused vs. broad attention
  - whether attention heads focus on a few words or attend broadly over many words.
  - average entropy of each head's attention distribution
  - some attention heads, especially in lower layers, have very broad attention.



What does BERT look at? (cont.)

- Probing individual attention heads
- **Dependency syntax:** evaluated both directions of prediction for each attention head:
  - the head word attending to the dependent and vice versa
  - For a given attention head and word, whichever other word receives the most attention weight is the model's prediction (excluding [SEP] and [CLS]).



What does BERT look at? (cont.)

- Probing individual attention heads
- **Dependency syntax**, the best performing attentions heads of BERT on WSJ dependency parsing by dependency type.
- Fixed offset baseline: the word *n* positions to the left of the dependent is always considered to be the head

Relation	Head	Accuracy	Baseline
All	7-6	34.5	26.3 (1)
prep	7-4	66.7	61.8 (-1)
pobj	9-6	**76.3**	34.6 (-2)
det	8-11	**94.3**	51.7 (1)
nn	4-10	70.4	70.2 (1)
nsubj	8-2	58.5	45.5 (1)
amod	4-10	75.6	68.3 (1)
dobj	8-10	**86.8**	40.0 (-2)
advmod	7-6	48.8	40.2 (1)
aux	4-10	81.1	71.5 (1)
poss	7-6	**80.5**	47.7 (1)
auxpass	4-10	**82.5**	40.5 (1)
ccomp	8-1	**48.8**	12.4 (-2)
mark	8-2	**50.7**	14.5 (2)
prt	6-7	**99.1**	91.4 (-1)

There is no single attention head that does well at syntax “overall”. However, certain attention heads specialize to specific dependency relations, sometimes achieving high accuracy and substantially outperforming the fixed-offset baseline.

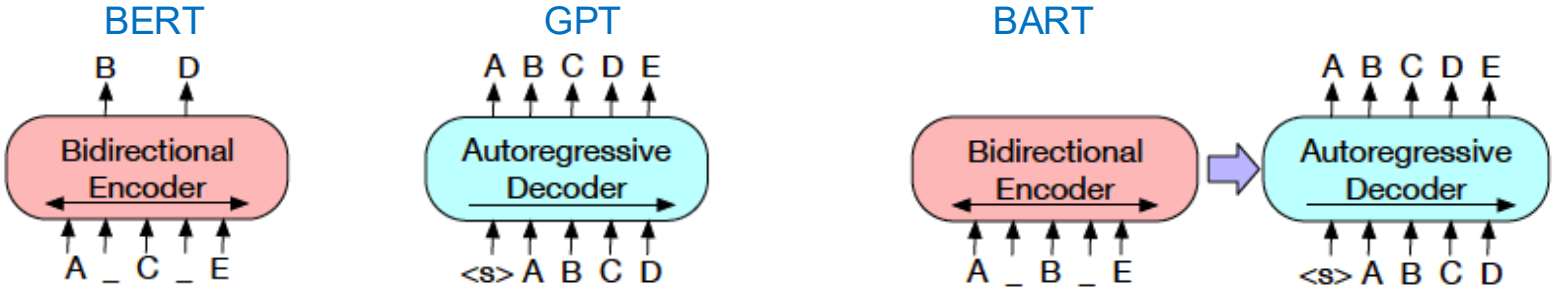
Topics for Today

- Tokenization and Byte Pair Encoding (BPE)
- Decoder-Only Models: OpenAI GPT
- Encoder-Only Models: BERT
- Encoder-Decoder Models: BART, T5
- Parameter-Efficient Fine-Tuning (PEFT)

  Adapters, Prefix Tuning, LoRA

BART: Bi-directional and Auto-Regressive Transformers

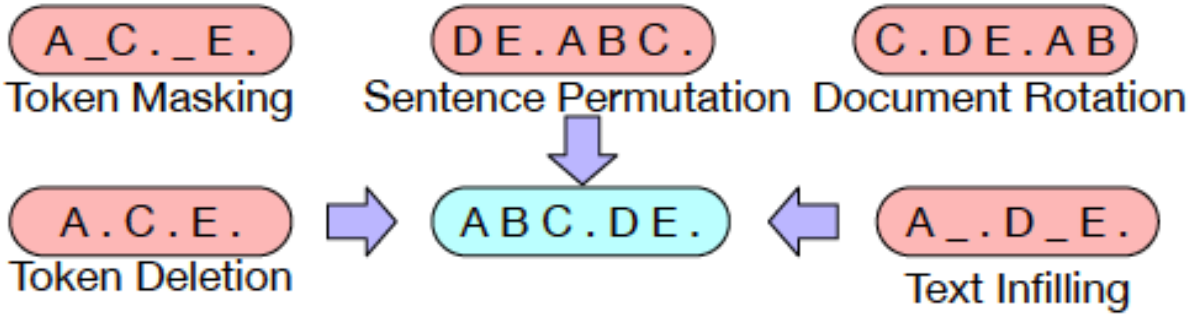
- Pre-trains a model by combining bi-directional and auto-regressive transformers.



- The architecture is similar to BERT, with the following differences:
  - each layer of the decoder additionally performs cross-attention over the final hidden layer of the encoder
  - BERT uses an additional feed-forward network before word prediction, BART does not.

A key advantage: noising flexibility, arbitrary transformations can be applied to the original text, including changing its length.

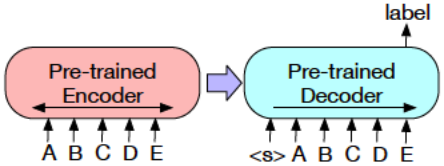
BART - Pre-training Transformations



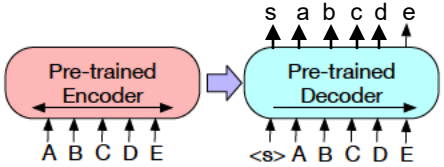
Lewis et al., “BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension”, arXiv preprint: 1910.13461, 2019.

BART -- Fine-Tuning

- Sequence classification



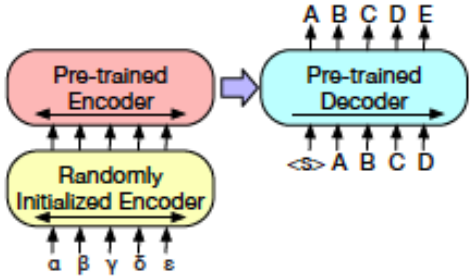
- Token classification



Label sequence: s a b c d e

- Sequence generation
  - Encoder: input sequence
  - Decoder: generates output autoregressively.

- Machine translation



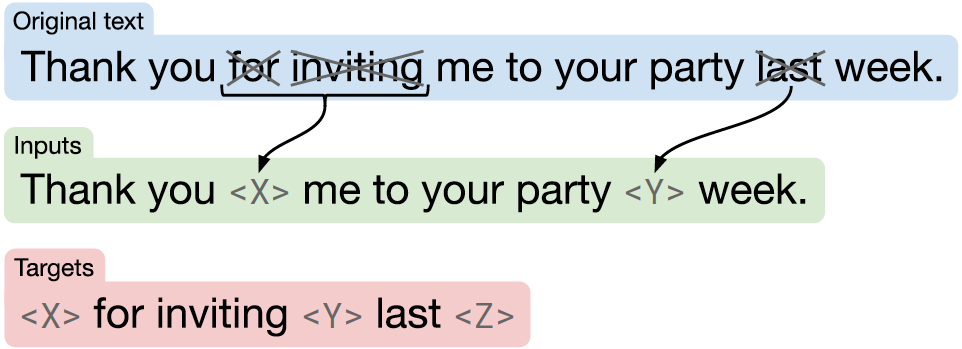
T5: Text-to-text Transfer Transformer

- Recent techniques for transfer learning in NLP pre-train models on large, unlabeled data.
- Common Crawl project produces about 20TB of text data extracted from web pages each month.
- Explores transfer learning techniques through converting all text-based language problems into a text-to-text format.

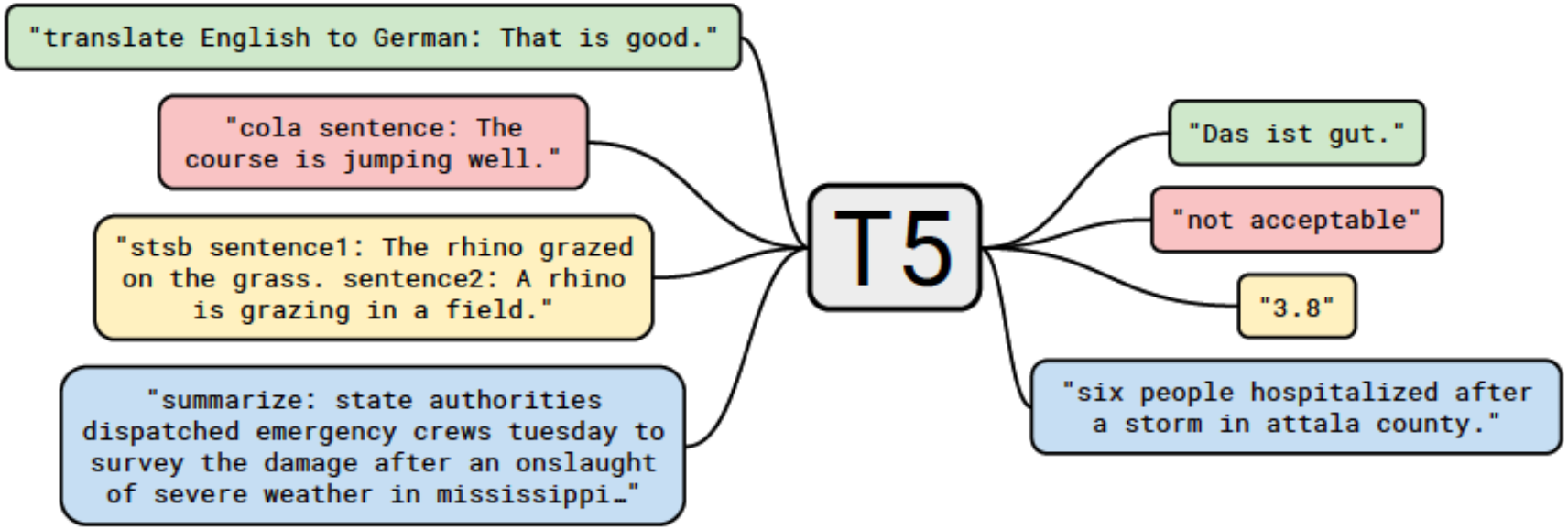
Raffel et al., “Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer”, JMLR, 2020.

T5 Unsupervised Pre-training Objective

- Randomly sample and drop 15% of tokens in each sentence.



T5: Text-to-text Transfer Transformer (cont.)



Topics for Today

- Tokenization and Byte Pair Encoding (BPE)
- Decoder-Only Models: OpenAI GPT
- Encoder-Only Models: BERT
- Encoder-Decoder Models: BART, T5
- Parameter-Efficient Fine-Tuning (PEFT)

  Adapters, Prefix Tuning, LoRA

LLMs: Trend towards larger models

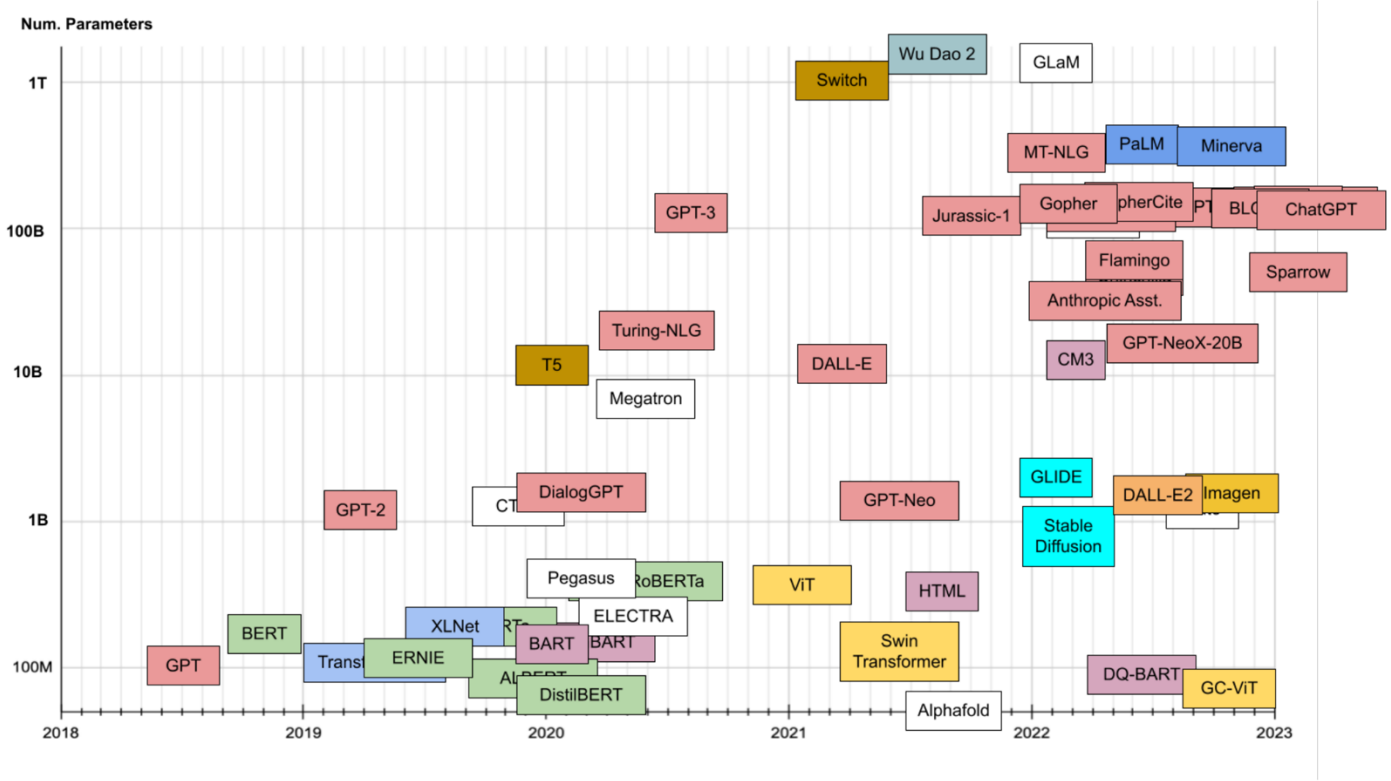


Figure from: https://www.researchgate.net/profile/Minh-Tien-Nguyen-3/publication/370763852_When_Giant_Language_Brains_Just_Aren't_Enough_Domain_Pizzazz_with_Knowledge_Sparkle_Dust/links/646646d9c9802f2f72e51dca/When-Giant-Language-Brains-Just-Arent-Enough-Domain-Pizzazz-with-Knowledge-Sparkle-Dust.pdf?_tp=eyJjb250ZXh0Ijp7ImZpcnN0UGFnZSI6Il9kaXJlY3QiLCJwYWdlIjoicHVibGljYXRpb24iLCJwcmV2aW91c1BhZ2UiOiJfZGlyZWN0In19

Energy and Environment Considerations

- Notable gains due to large models, trained on large datasets on multiple NLP benchmarks.
- Training these models require large resources

From twitter:

> Wow: Google's "Meena" chatbot was trained on a full TPUv3 pod (2048 TPU cores) for **30 full days** - That's more than $1,400,000 of compute time to train this chatbot model. (! 100+ petaflops of sustained compute !)

- and energy consumption!

Energy and Environment Considerations (cont.)

- Some of this energy may come from renewable or carbon credit-offset resources, however:
  1. This is still not the case for many locations
  2. Even when renewable energy is available, it might better be allocated for other things.
- We must cut carbon emissions by half over the next decade to deter escalating rates of natural disasters.

Consumption	CO_2e (lbs)
Air travel, 1 passenger, NY↔SF	1984
Human life, avg, 1 year	11,023
American life, avg, 1 year	36,156
Car, avg incl. fuel, 1 lifetime	126,000
Training one model (GPU)	
NLP pipeline (parsing, SRL)	39
w/ tuning & experimentation	78,468
Transformer (big)	192
w/ neural architecture search	626,155

Table 1: Estimated CO_2 emissions from training common NLP models, compared to familiar consumption.[1]

Strubell et al., Energy and Policy Considerations for Deep Learning in NLP. ACL 2019.

Estimated Cost of Training a Model

- Analysis of 4 popular models
- Re-trained for a maximum of a day with default settings
- Sampled GPU and SPU power consumption

Model	Hardware	Power (W)	Hours	kWh·PUE	CO_2e	Cloud compute cost
$T2T_{base}$	P100x8	1415.78	12	27	26	$41–$140
$T2T_{big}$	P100x8	1515.43	84	201	192	$289–$981
ELMo	P100x3	517.66	336	275	262	$433–$1472
$BERT_{base}$	V100x64	12,041.51	79	1507	1438	$3751–$12,571
$BERT_{base}$	TPUv2x16	—	96	—	—	$2074–$6912
NAS	P100x8	1515.43	274,120	656,347	626,155	$942,973–$3,201,722
NAS	TPUv2x1	—	32,623	—	—	$44,055–$146,848
GPT-2	TPUv3x32	—	168	—	—	$12,902–$43,008

Table 3: Estimated cost of training a model in terms of CO_2 emissions (lbs) and cloud compute cost (USD).[7] Power and carbon footprint are omitted for TPUs due to lack of public information on power draw for this hardware.

It is important to prioritize innovation of computationally efficient hardware and algorithms!

Transfer Learning with LLMs

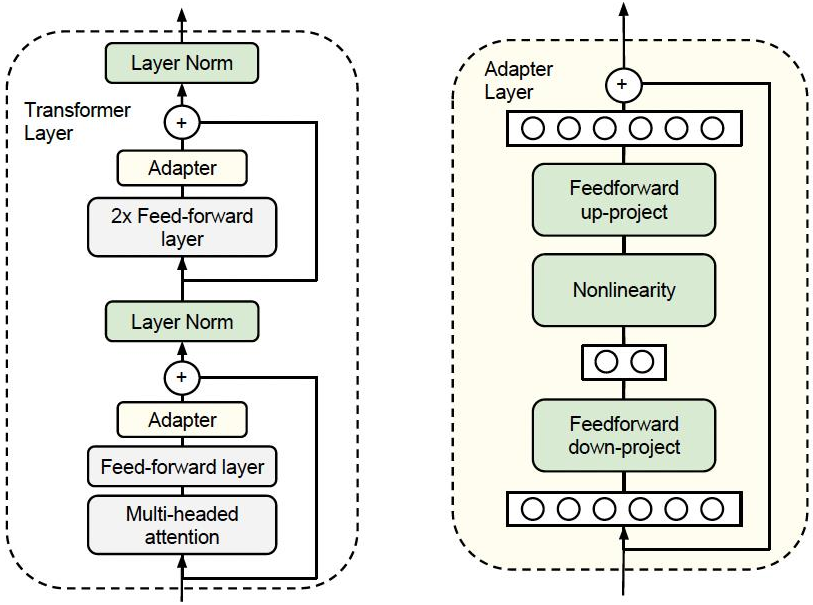
- Feature-based transfer learning
- Fine-tuning
  - Often performs better for NLP tasks!

- But, do we have to change all the model parameters during fine-tuning?

Adapter Tuning

- Houlsby et al, ICML 2019.
- New layers added between layers of the original neural network.
- An LM with parameters w: $\phi_w(x)$
- Feature-based transfer composes ϕ_w with a new function δ_v, resulting in $\delta_v(\phi_w(x))$. Only the new parameters, v, are trained.
- Pretraining involves updating w.
- For adapter tuning, a new function $\Pi_{w,v}(x)$ is defined (i.e., adding new layers to the network)
  - Parameters w are copied from the original model
  - Parameters v are tuned during training.
  - $|v| << |w|$

Adapter Tuning (cont.)



- Weights of the original network are frozen, and are shared by many tasks.
- Weights of the new layers are initialized at random, and mimic the original behavior at the beginning (near identity).
- Only the weights of the new layers are updated during training.
- Bottleneck layers mapping: d dimensions to m then back to d ($2dm + d + m$ parameters to be trained), $d >> m$.

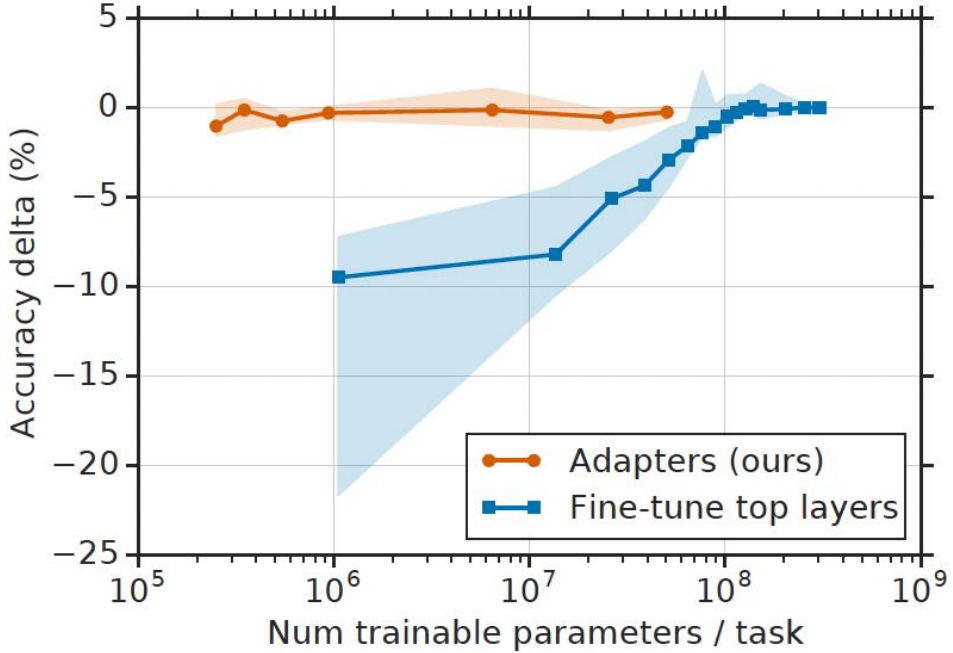
Adapter Tuning (cont.)

Parameter-Efficient Transfer Learning for NLP

	Total num params	Trained params / task	CoLA	SST	MRPC	STS-B	QQP	MNLI_m	MNLI_mm	QNLI	RTE	Total
BERT_LARGE	9.0×	100%	60.5	94.9	89.3	87.6	72.1	86.7	85.9	91.1	70.1	80.4
Adapters (8-256)	1.3×	3.6%	59.5	94.0	89.5	86.9	71.8	84.9	85.1	90.7	71.5	80.0
Adapters (64)	1.2×	2.1%	56.9	94.2	89.6	87.3	71.8	85.3	84.6	91.4	68.8	79.6

9x parameters are obtained with fine-tuning, as there are 9 tasks.
When optimum bottleneck size per task is chosen, GLUE score is 80.0
When bottleneck size is limited to 64 for all tasks, GLUE score is 79.6

Adapter Tuning (cont.)



LoRA: Low-Rank Adaptation of Large LLMs

- Hu & Shen et al., ICLR 2022.
- Adapter layers introduce inference latency.
- Prefix tuning can be difficult to optimize.

LoRA: Low-Rank Adaptation of Large LLMs (cont.)

- Autoregressive LM $P_\phi(y|x)$ parameterized by ϕ.
- Downstream tasks represented by a training dataset of context, target pairs:

  $$Z=\{(x_i, y_i)\}, i=1,\ldots,N$$

- During full fine-tuning, the model is initialized to pre-trained weights Φ_0 and updated to $\Phi_0 + \Delta\Phi$ by repeatedly following the gradient to maximize the conditional language modeling objective:

$$\max_{\Phi} \sum_{(x,y)\in\mathcal{Z}} \sum_{t=1}^{|y|} \log\left(P_\Phi(y_t|x, y_{<t})\right)$$

LoRA: Low-Rank Adaptation of Large LLMs (cont.)

- For a pre-trained weight matrix $W_0 \in R^{d \times k}$, change the training update with a low rank decomposition:

$$W_0 + \Delta W = W_0 + AB$$

where $A \in R^{d \times r}$ and $B \in R^{r \times k}$ and the rank $r << \min(d, k)$.

- For $h=W_0 x$, the modified forward pass yields:

$$h = W_0 x + \Delta W x = W_0 x + BAx$$

- During training, W_0 is frozen, only A and B include trainable parameters and are updated.
- Random Gaussian initialization for A and zero for B.

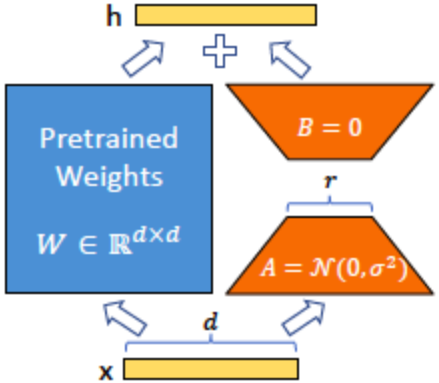
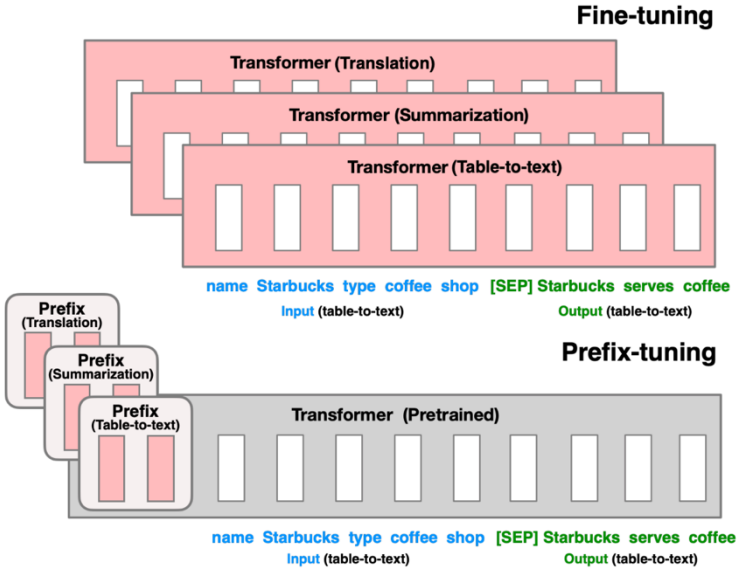


Figure 1: Our reparametrization. We only train A and B.

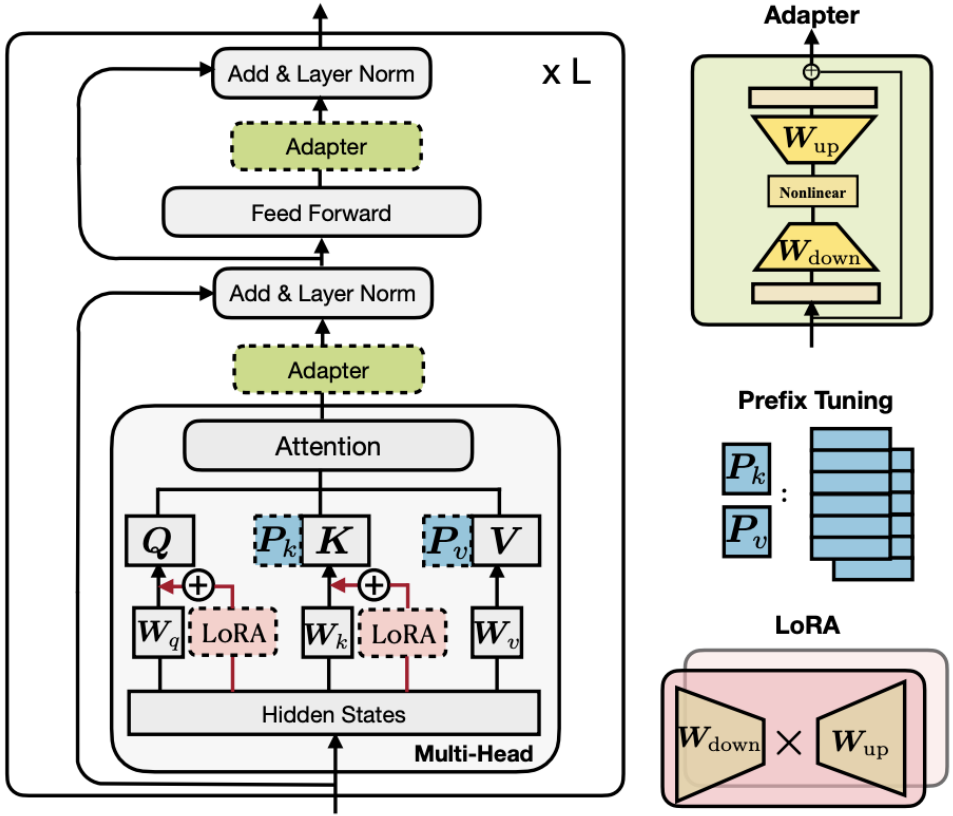
Prefix Tuning

- (Li and Liang, 2021) https://arxiv.org/pdf/2101.00190.pdf
- Parameter efficient fine-tuning method, like low-rank adapters (LoRA).
- Original model weights are frozen during fine-tuning.
- Training optimizes a small vector called the prefix for each new task.
- The prefix vectors are prepended to the keys and values of the multi-head attention at every transformer layer.



PEFT – Summary

- Depiction of different parameter-efficient fine-tuning methods from (He et al., ICLR, 2022)
  https://arxiv.org/pdf/2110.04366.pdf
- All require supervision to fine-tune the newly added parameters.



Topics for Thursday

- Major Paradigms in NLP
- Prompting
- In-context Learning
- Prompting Examples