STEVENS
INSTITUTE of TECHNOLOGY
THE INNOVATION UNIVERSITY®

CS584 Natural Language Processing

Pretraining, Finetuning

Ping Wang
Department of Computer Science
Stevens Institute of Technology

Outlines

- What is model pretraining?
- Model pretraining:
  - Encoders
  - Encoder-Decoders
  - Decoders
- Finetuning

What is pre-training?

- "Pre-train" a model on a large dataset for task X, then "fine-tune" it on a dataset for task Y
- Key idea: X is somewhat related to Y, so a model that can do X will have some good neural representations for Y as well
    - ImageNet pre-training is huge in computer vision: learn generic visual features for recognizing objects
    - Webpage and books for learn generic features for texts
- GloVe can be seen as pre-training:
    - Learn vectors with the skip-gram objective on large data (task X), then fine-tune them as part of a neural network for sentiment/any other task (task Y).

GloVe is insufficient

- GloVe uses a lot of data but in a weak way.
- GloVe gives a single embedding for each word.

  they swing the bats; they see the bats

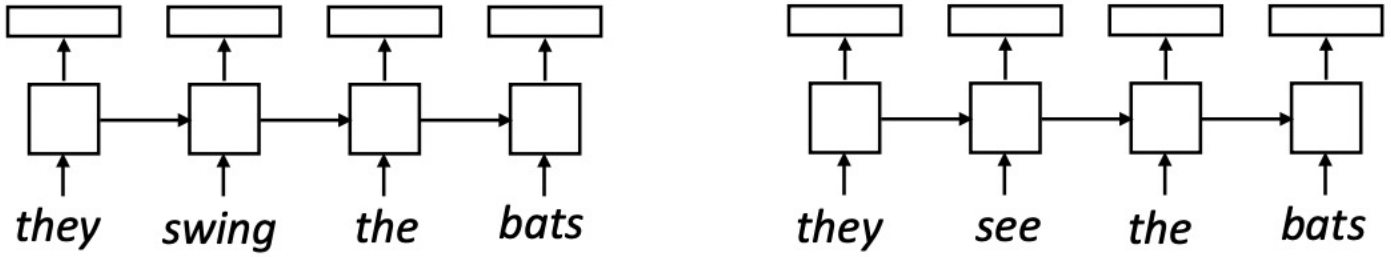
  - Identifying discrete word senses is hard, doesn't scale.
  - Hard to identify how many senses each word has.

- How can we make our word embeddings more context-dependent?
  - Use language model **pretraining**!

Context-dependent Embeddings

- Train a neural language model to predict the next word given previous words in the sentence, use the hidden states (output) at each step as word embeddings.

- This is the key idea behind ELMo: language models can allow us to form useful word representations in the same way as word2vec did.



Peters et al. (2018)

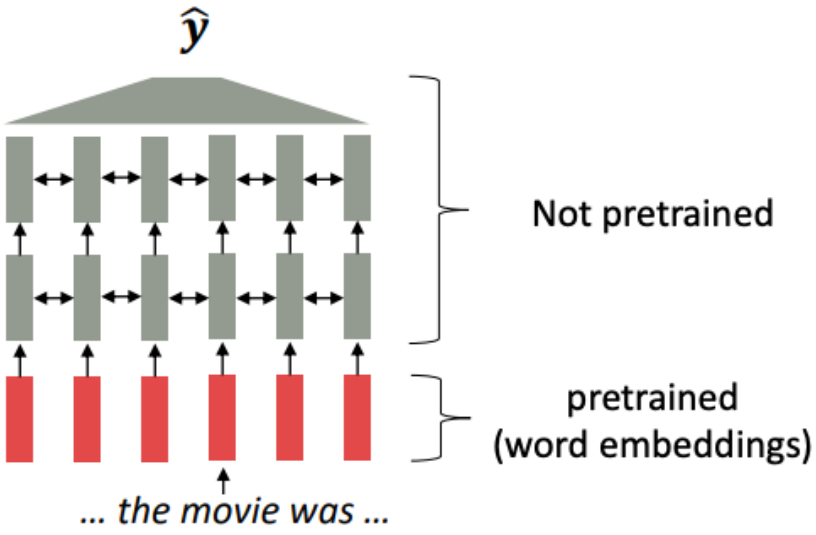
Where we were: pretrained word embeddings

With pretrained embeddings:

- Start with pretrained word embeddings (no context!)
- Learn how to incorporate context in an LSTM or Transformer while training on the task.

Some issues to think about:

- The training data we have for our downstream task (like question answering) must be sufficient to teach all contextual aspects of language.
- Most of the parameters in our network are randomly initialized!



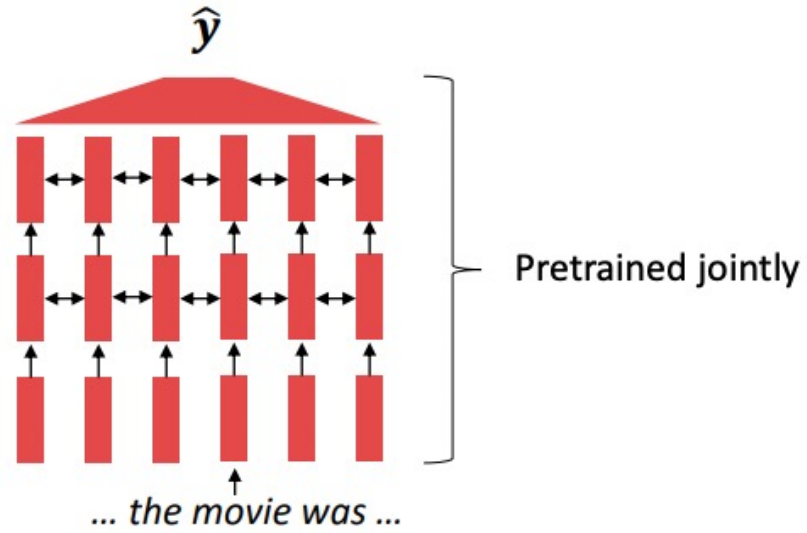
[Recall, *movie* gets the same word embedding, no matter what sentence it shows up in]

Where we're going: pretraining whole models

In modern NLP:

- All (or almost all) parameters in NLP networks are initialized via pretraining.
- Pretraining methods hide parts of the input from the model, and train the model to reconstruct those parts.

- This has been exceptionally effective at building strong:
  - Representations of language
  - Parameter initializations for strong NLP models
  - Capturing probability distributions over language that we can sample from



[This model has learned how to represent entire sentences through pretraining]

What can we learn from reconstructing the input?

- I put ___ fork down on the table.
- The woman walked across the street, checking for traffic over ___ shoulder.
- I went to the ocean to see the fish, turtles, seals, and _____.
- I was thinking about the sequence that goes 1, 1, 2, 3, 5, 8, 13, 21, ____
- And more…

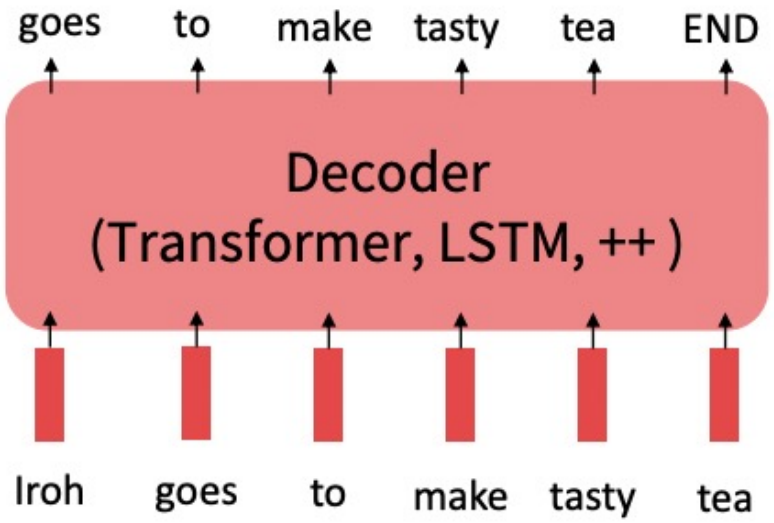
Pretraining through language modeling

Recall the language modeling task:

- Model $p_\theta(w_t|w_{1:t-1})$, the probability distribution over words given their past contexts.
- There's lots of data for this! (In English.)

Pretraining through language modeling:

- Train a neural network to perform language modeling on a large amount of text.
- Save the network parameters



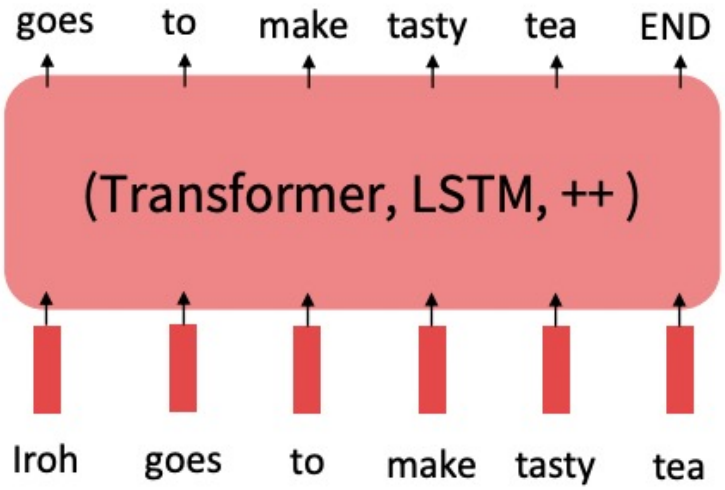
[Dai and Le, 2015]

The Pretraining / Finetuning Paradigm

Pretraining can improve NLP applications by serving as parameter initialization.

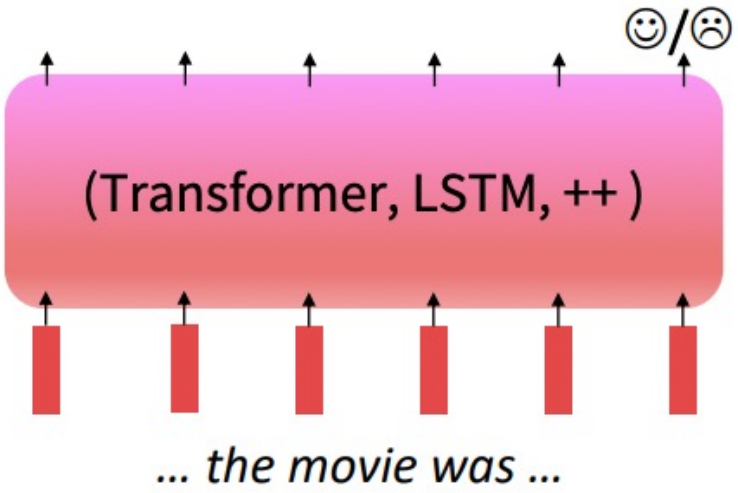
Step 1: Pretrain (on language modeling)

Lots of text; learn general things!



Step 2: Finetune (on your task)

Not many labels; adapt to the task!



Stochastic gradient descent and pretrain/finetune

Why should pretraining and finetuning help, from a "training neural nets" perspective?

- Consider, provides parameters $\hat{\theta}$ by approximating $\min_{\theta} L_{pretrain}(\theta)$.
  - The pretraining loss
- Then, finetuning approximates $\min_{\theta} L_{finetune}(\theta)$, starting at $\hat{\theta}$.
  - The finetuning loss
- The pretraining may matter because stochastic gradient descent sticks (relatively) close to $\hat{\theta}$ during finetuning.
  - So, maybe the finetuning local minima near $\hat{\theta}$ tend to generalize well!
  - And/or, maybe the gradients of finetuning loss near $\hat{\theta}$ propagate nicely!

Pretraining

- Pre-training establishes the basis of the abilities of LLMs.
- By pre-training on large-scale corpora, LLMs can acquire essential language understanding and generation skills.
- In this process, the scale and quality of the pre-training corpus are critical for LLMs to attain powerful capabilities.
- Furthermore, to effectively pre-train LLMs, model architectures, acceleration methods, and optimization techniques need to be well designed.

Pretraining

- Data collection
- Architecture

Pretraining: Data collection

- Compared with small-scale language models, LLMs have a stronger demand for high-quality data for model pretraining, and their model capacities largely rely on the pretraining corpus and how it has been preprocessed.
- Existing LLMs mainly leverage a mixture of diverse public textual datasets as the pre-training corpus.

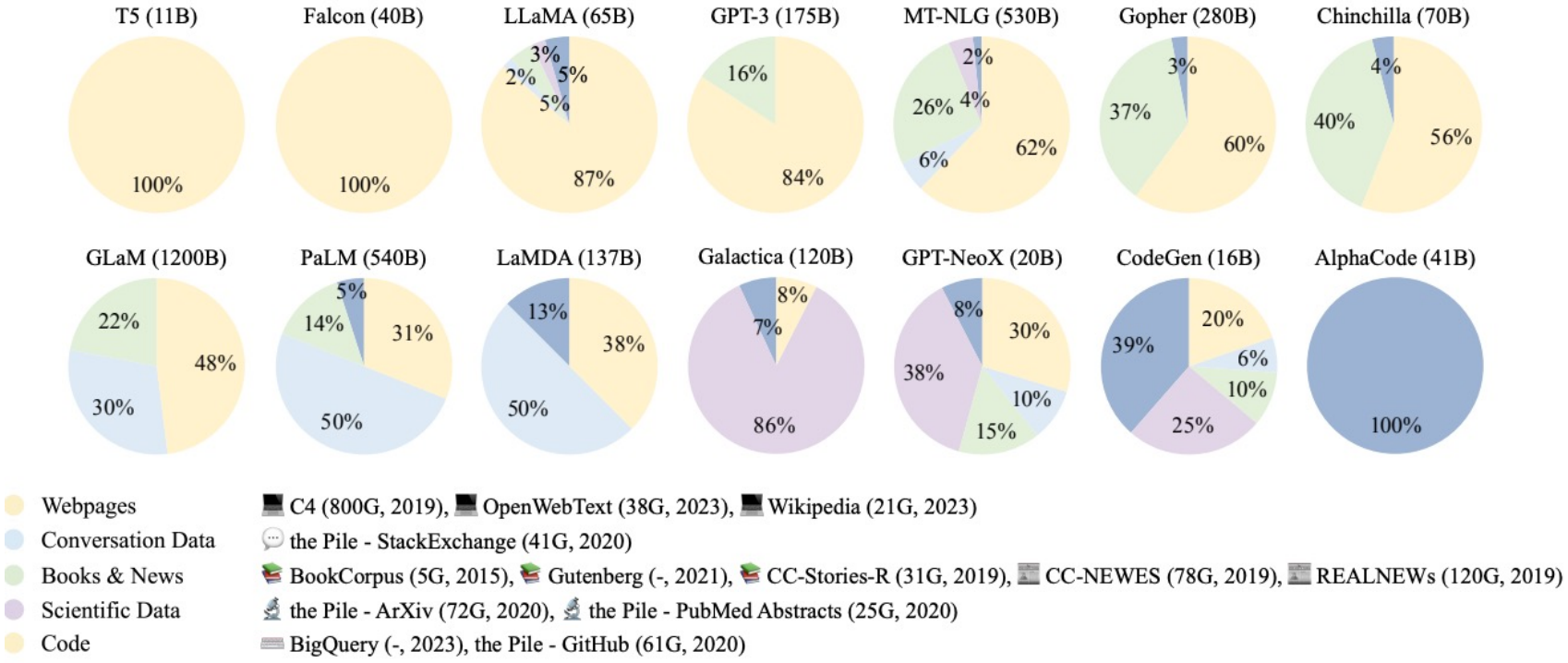


Fig. 5: Ratios of various data sources in the pre-training data for existing LLMs.

Pretraining: Data Source

The source of pre-training corpus can be broadly categorized into two types:

- **General data**: with large, diverse, and accessible nature to enhance the language modeling and generalization abilities of LMs.
  - **Webpages**: a large amount of data are crawled from the web, with high- and low- quality text, which need to further filter and process.
  - **Books**: an important source of formal long texts, which are potentially beneficial for LMs to learn linguistic knowledge, model long-term dependency, and generate narrative and coherent texts.
  - **Conversational text**: can improve their performance on a range of question-answering tasks; public conversation corpus or social media.

Pretraining: Data Source

The source of pre-training corpus can be broadly categorized into two types:

- **Specialized data**: endowing LLMs with specific task-solving capabilities.
  - **Multilingual data**: can enhance the multilingual abilities of language understanding and generation.
  - **Scientific data**: enhance the understanding of scientific knowledge; arXiv papers, scientific textbooks, math webpages, and other related scientific resources.
  - **Code**: Recent studies have found that training LLMs on a vast code corpus can lead to a substantial improvement in the quality of the synthesized programs.
    - Programming question answering communities like stack exchange
    - Public software repositories such as GitHub.

Pretraining: Data Preprocessing

It is essential to preprocess the data for constructing the pre-training corpus, especially removing noisy, redundant, irrelevant, and potentially toxic data, which may largely affect the capacity and performance of LMs.

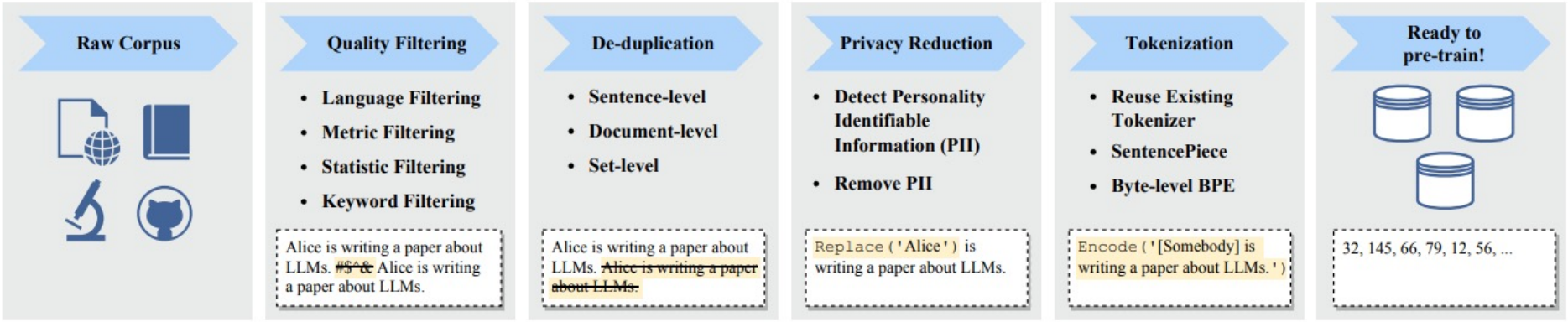


Fig. 6: An illustration of a typical data preprocessing pipeline for pre-training large language models.

Pretraining: Data Preprocessing

- **Quality Filtering**: remove low-quality data from the collected corpus
  - Classifier-based and heuristic-based
- **De-duplication**: duplicate data in a corpus would reduce the diversity of language models, which may cause the training process to become unstable and thus affect the model performance.
  - Granularities: sentence-level, document-level, and dataset-level
- **Privacy Reduction**: data may involve sensitive or personal information, which may increase the risk of privacy breaches.
  - Remove the personally identifiable information (PII)
- **Tokenization**: subword tokenizers have been widely used in Transformer based language models.
  - Byte-Pair Encoding (BPE); WordPiece tokenization
  - Recent LLMs often train the customized tokenizers specially for the pre-training corpus.

Effect of Pre-training Data on LLMs

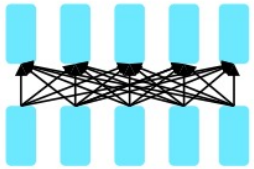
- Usually infeasible to iterate the pre-training of LLMs multiple times, due to the huge demand for computational resources.
- It is particularly important to construct a well-prepared pre-training corpus before training a LLM.
- How the quality and distribution of the pre-training corpus potentially influence the performance of LLMs.
  - **Mixture of sources**: include as many high-quality data sources as possible, and carefully set the distribution of pre-training data, since it is also likely to affect the performance of LLMs on downstream tasks.
  - **Amount of Pre-training Data**: Existing studies have found that with the increasing parameter scale in the LLM, more data is also required to train the model.
  - **Quality of Pre-training Data**: Existing work has shown that pre-training on the low-quality corpus, such as noisy, toxic, and duplicate data, may hurt the performance of models.

Pretraining

- Data collection
- Architecture

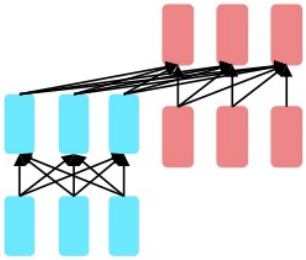
Pretraining for three types of architectures

The neural architecture influences the type of pretraining, and natural use cases.



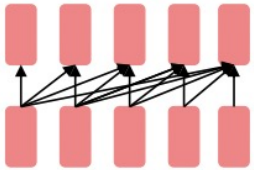
Encoders

- Gets bidirectional context – can condition on future!
- How do we train them to build strong representations?



Encoder-Decoders

- Good parts of decoders and encoders?
- What's the best way to pretrain them?

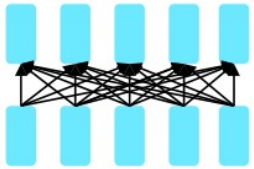


Decoders

- Language models! What we've seen so far.
- Nice to generate from; can't condition on future words

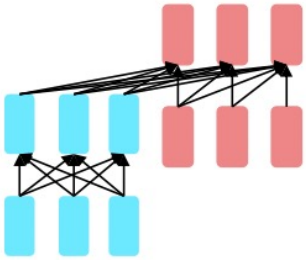
Pretraining for three types of architectures

The neural architecture influences the type of pretraining, and natural use cases.



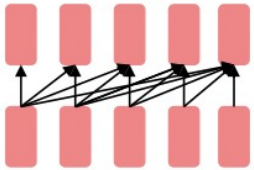
Encoders

- Gets bidirectional context – can condition on future!
- How do we train them to build strong representations?



Encoder-Decoders

- Good parts of decoders and encoders?
- What's the best way to pretrain them?



Decoders

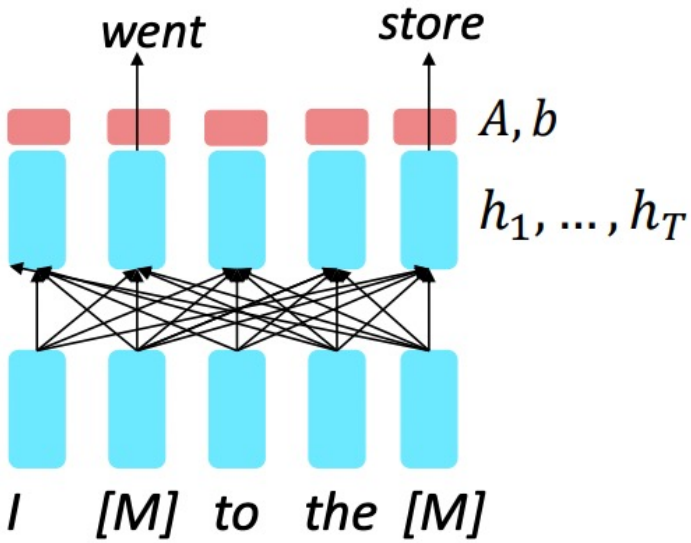
- Language models! What we've seen so far.
- Nice to generate from; can't condition on future words

Pretraining encoders: what pretraining objective to use?

- Encoders get bidirectional context, so we can't do language modeling!
- Idea: replace some fraction of words in the input with a special [MASK] token; predict these words.

$$h_1, \dots, h_T = \text{Encoder}(w_1, \dots, w_T)$$
$$y_i \sim Aw_i + b$$

- Only add loss terms from words that are "masked out." If $\tilde{x}$ is the masked version of x, we're learning $p_\theta(x|\tilde{x})$. Called Masked LM.



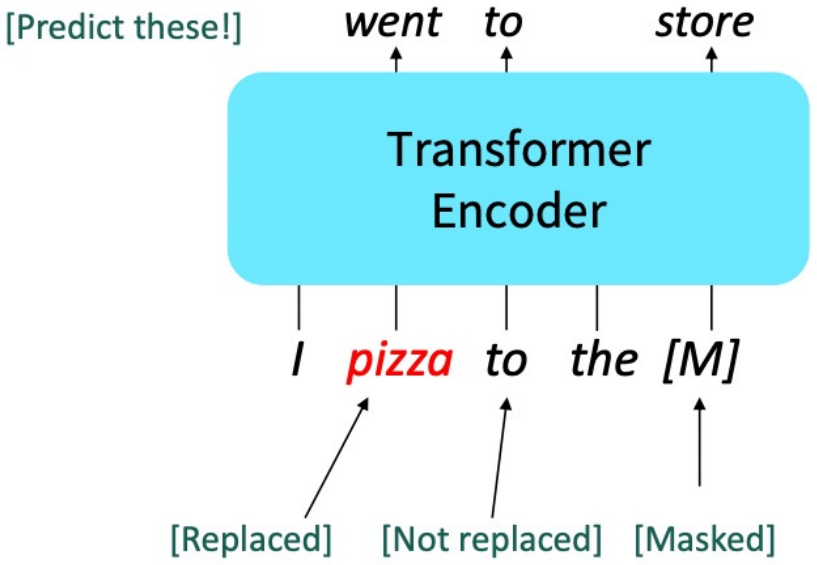
[Devlin et al., 2018]

BERT: Bidirectional Encoder Representations from Transformers

Devlin et al., 2018 proposed the "Masked LM" objective and released the weights of a pretrained Transformer, a model they labeled BERT.

Some more details about Masked LM for BERT:

- Predict a random 15% of (sub)word tokens.
  - Replace input word with [MASK] 80% of the time
  - Replace input word with a random token 10% of the time
  - Leave input word unchanged 10% of the time (but still predict it!)
- Why? Doesn't let the model get complacent and not build strong representations of non-masked words. (No masks are seen at fine-tuning time!)



[Devlin et al., 2018]

Overall pre-training and fine-tuning procedures in BERT

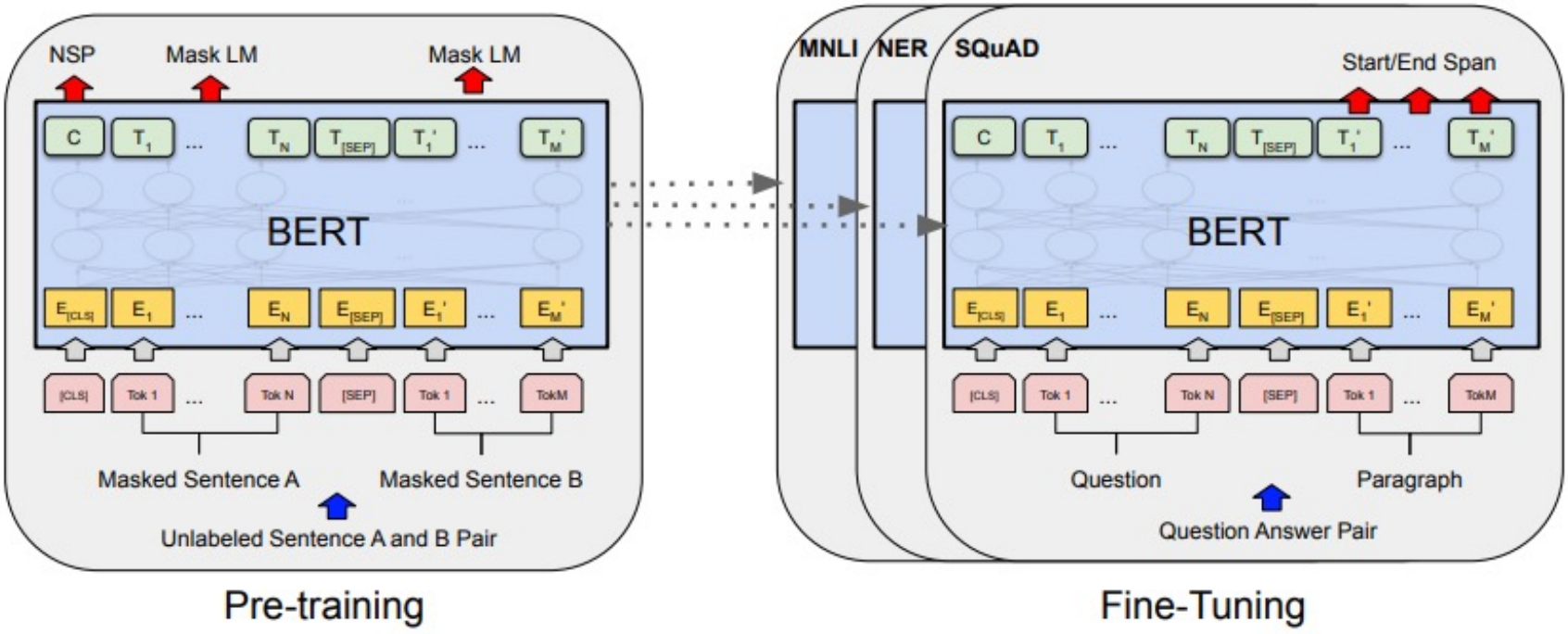
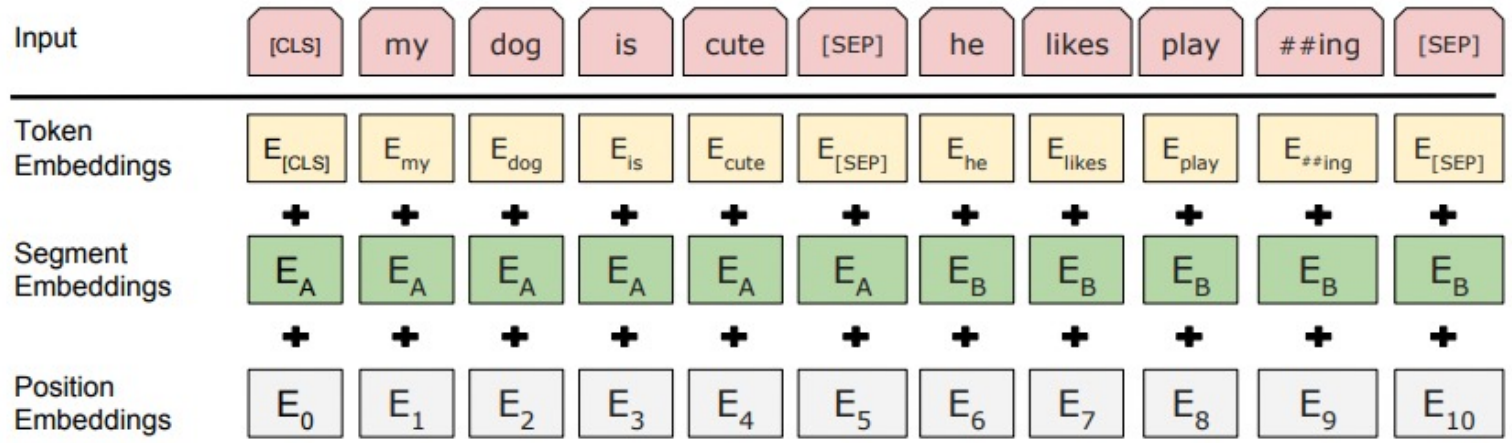


Figure 1: Overall pre-training and fine-tuning procedures for BERT. Apart from output layers, the same architectures are used in both pre-training and fine-tuning. The same pre-trained model parameters are used to initialize models for different down-stream tasks. During fine-tuning, all parameters are fine-tuned. `[CLS]` is a special symbol added in front of every input example, and `[SEP]` is a special separator token (e.g. separating questions/answers).

BERT: Bidirectional Encoder Representations from Transformers

- The pretraining input to BERT was two separate contiguous chunks of text:



- BERT was trained to predict whether one chunk follows the other or is randomly sampled.
- Later work has argued this "next sentence prediction" is not necessary.

[Devlin et al., 2018, Liu et al., 2019]

BERT: Bidirectional Encoder Representations from Transformers

Details about BERT:

- Two models were released:
  - **BERT-base**: 12 layers, 768-dim hidden states, 12 attention heads, 110 million params.
  - **BERT-large**: 24 layers, 1024-dim hidden states, 16 attention heads, 340 million params.
- Trained on:
  - BooksCorpus (800 million words)
  - English Wikipedia (2,500 million words)
- Pretraining is expensive and impractical on a single GPU.
  - BERT was pretrained with 64 TPU chips for a total of 4 days.
  - (TPUs are special tensor operation acceleration hardware)
- Finetuning is practical and common on a single GPU.
  - “Pretrain once, finetune many times.”

[Devlin et al., 2018]

BERT: Bidirectional Encoder Representations from Transformers

BERT was massively popular and hugely versatile; finetuning BERT led to new state-of-the-art results on a broad range of tasks.

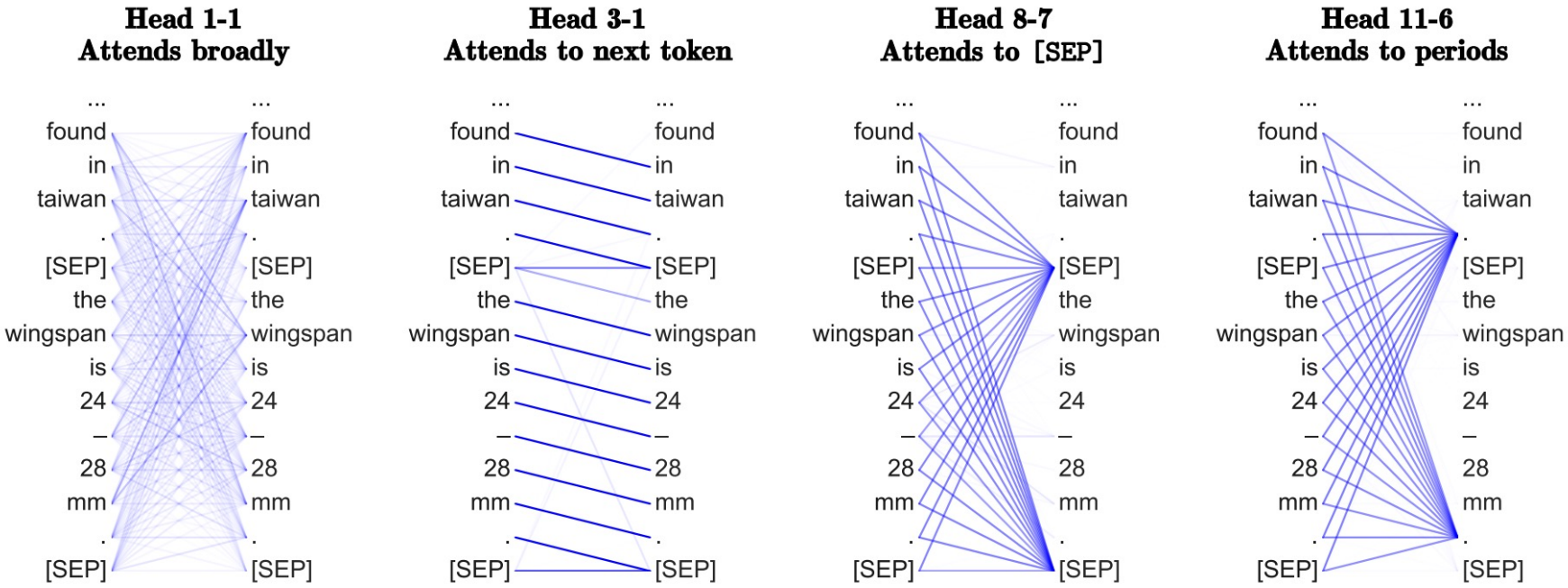
- **QQP:** Quora Question Pairs (detect paraphrase questions)
- **QNLI**: natural language inference over question answering data
- **SST-2**: sentiment analysis
- **CoLA**: corpus of linguistic acceptability (detect whether sentences are grammatical.)
- **STS-B**: semantic textual similarity
- **MRPC**: microsoft paraphrase corpus
- **RTE**: a small natural language inference corpus

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	79.8	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	87.4	91.3	45.4	80.0	82.3	56.0	75.1
$\text{BERT}_{\text{BASE}}$	84.6/83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4	79.6
$\text{BERT}_{\text{LARGE}}$	**86.7/85.9**	**72.1**	**92.7**	**94.9**	**60.5**	**86.5**	**89.3**	**70.1**	**82.1**

[Devlin et al., 2018]

What Does BERT Look At? An Analysis of BERT's Attention

Head [Layer]-[Head]



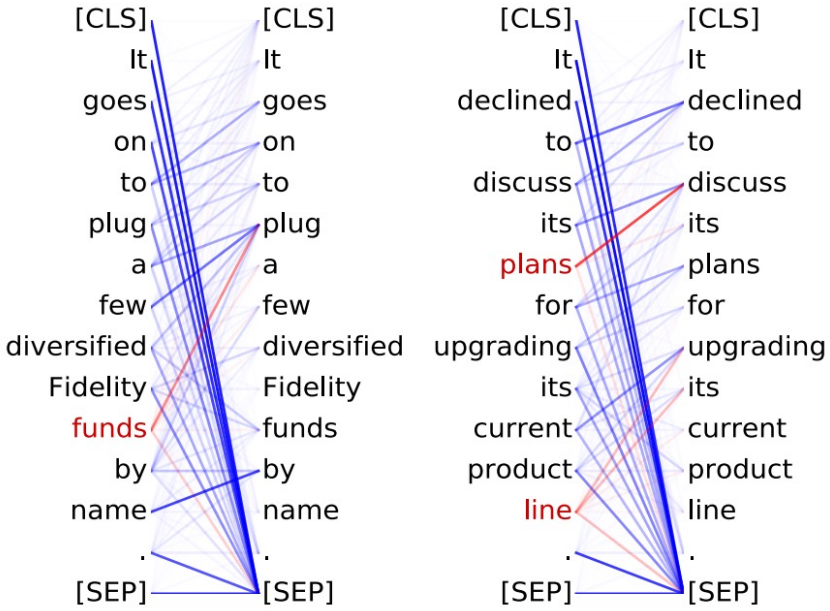
Heads on transformers learn interesting and diverse things: content heads (attend based on content), positional heads (based on position), etc.

Clark et al. (2019)

What Does BERT Look At? An Analysis of BERT's Attention

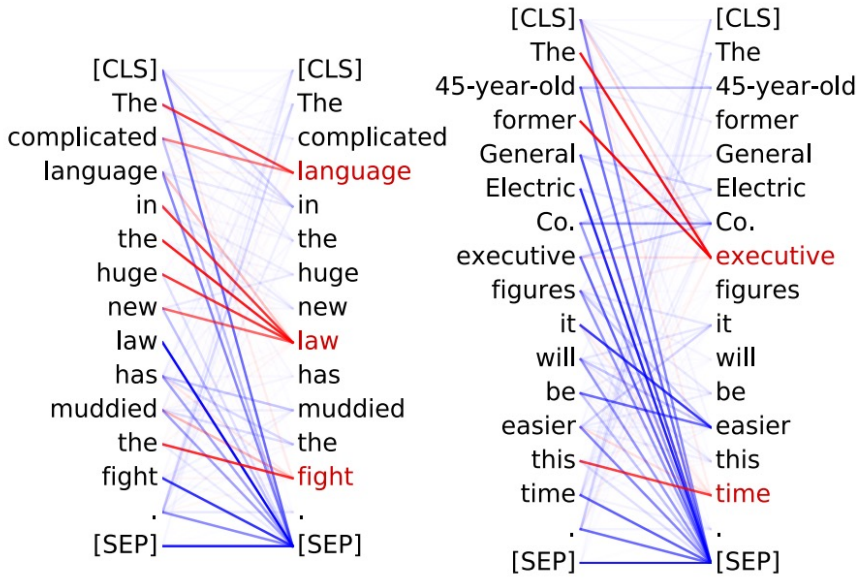
Head 8-10

- **Direct objects** attend to their verbs
- 86.8% accuracy at the `dobj` relation



Head 8-11

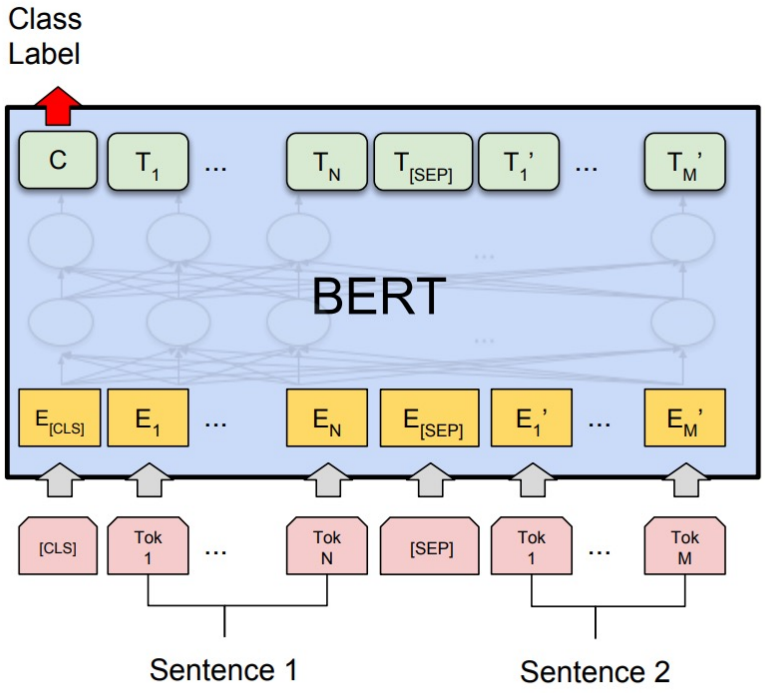
- **Noun modifiers** (e.g., determiners) attend to their noun
- 94.3% accuracy at the `det` relation



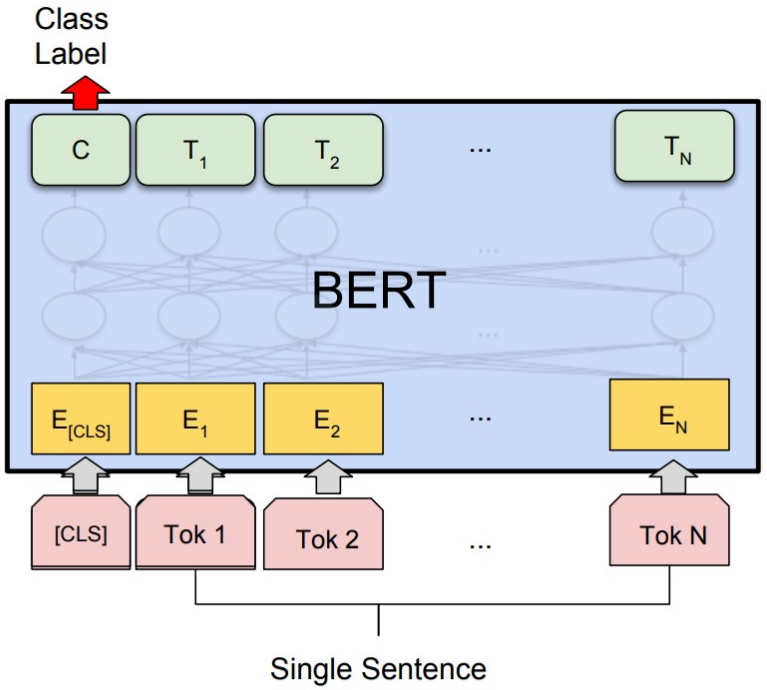
Still way worse than what supervised systems can do, but interesting that this is learned organically.

Clark et al. (2019)

Fine-tuning BERT: 1



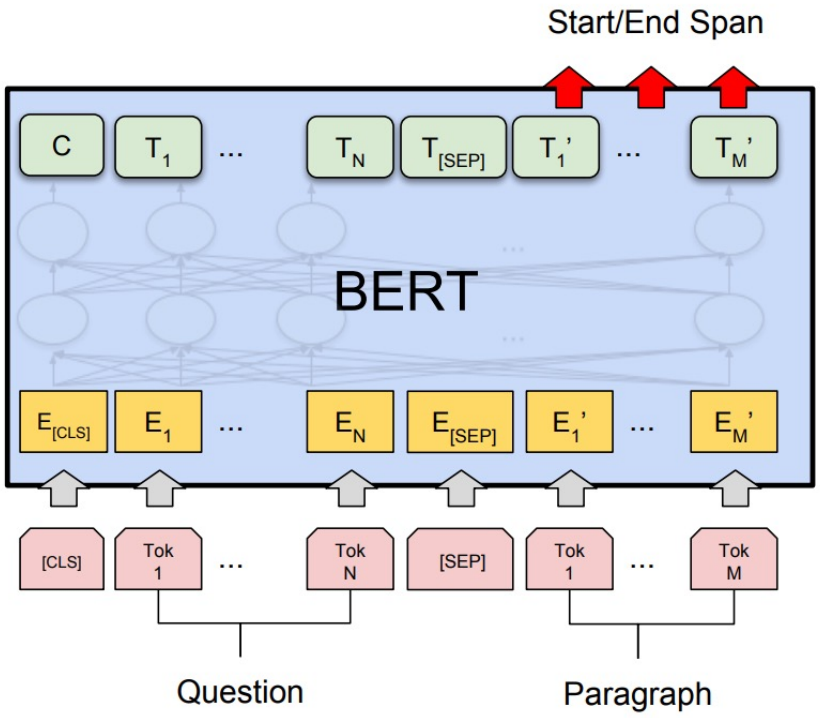
(a) Sentence Pair Classification Tasks: MNLI, QQP, QNLI, STS-B, MRPC, RTE, SWAG



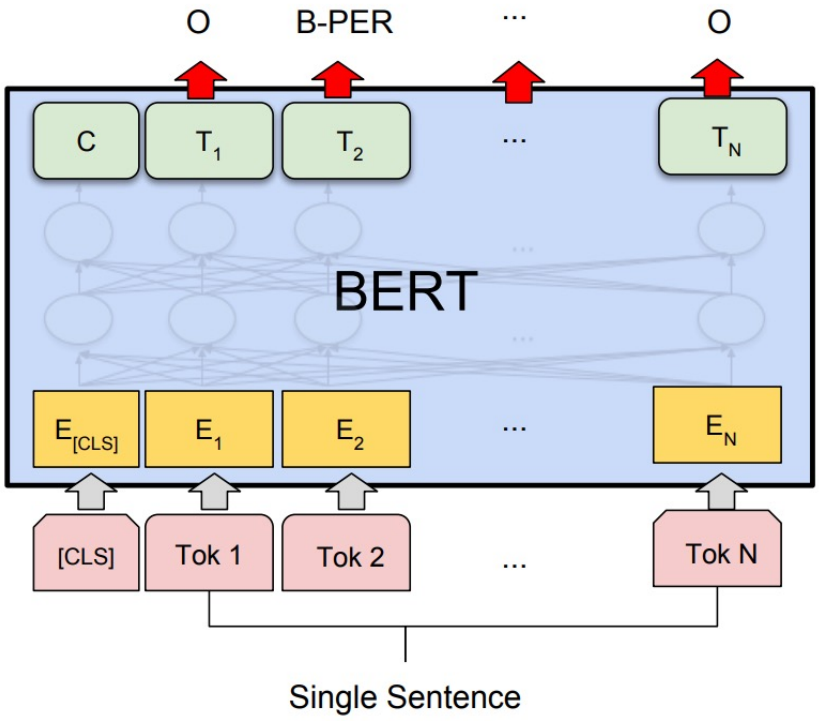
(b) Single Sentence Classification Tasks: SST-2, CoLA

[Devlin et al., 2018]

Fine-tuning BERT: 2



(c) Question Answering Tasks:
SQuAD v1.1

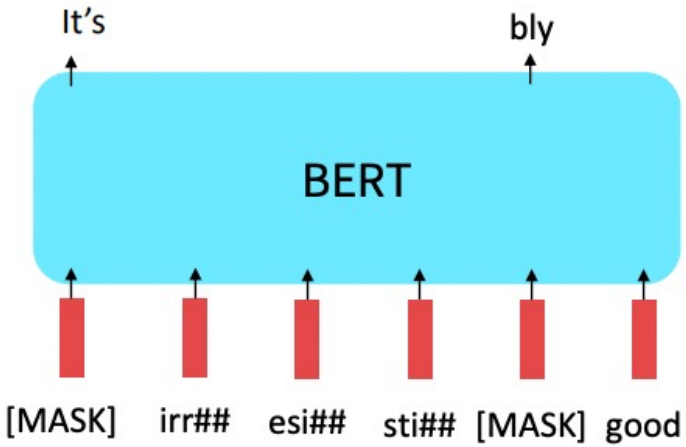


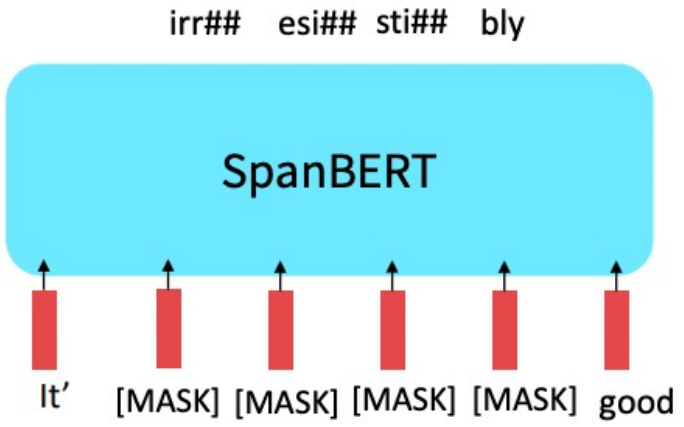
(d) Single Sentence Tagging Tasks:
CoNLL-2003 NER

[Devlin et al., 2018]

Extensions of BERT

- You'll see a lot of BERT variants like RoBERTa, SpanBERT, +++
- Some generally accepted improvements to the BERT pretraining formula:
  - **RoBERTa**: mainly just train BERT for longer and remove next sentence prediction!
  - **SpanBERT**: masking contiguous spans of words makes a harder, more useful pretraining task.





[Liu et al., 2019; Joshi et al., 2020]

Extensions of BERT

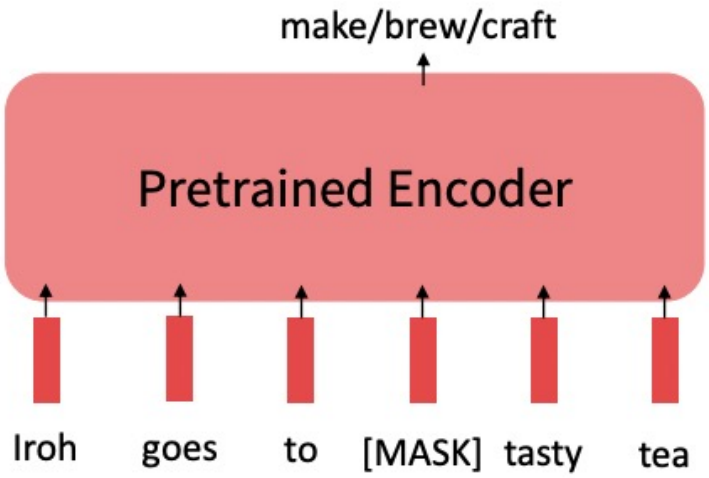
A takeaway from the RoBERTa paper: more compute, more data can improve pretraining even when not changing the underlying Transformer encoder.

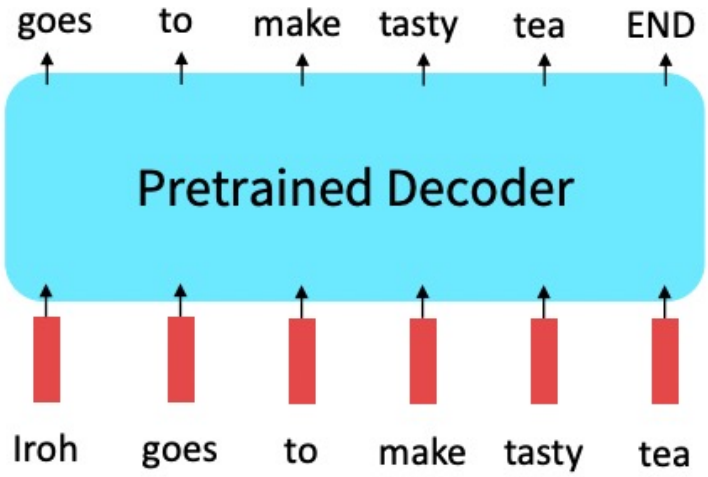
Model	data	bsz	steps	SQuAD (v1.1/2.0)	MNLI-m	SST-2
RoBERTa						
with BOOKS + WIKI	16GB	8K	100K	93.6/87.3	89.0	95.3
+ additional data (§3.2)	160GB	8K	100K	94.0/87.7	89.3	95.6
+ pretrain longer	160GB	8K	300K	94.4/88.7	90.0	96.1
+ pretrain even longer	160GB	8K	500K	**94.6/89.4**	**90.2**	**96.4**
$\text{BERT}_{\text{LARGE}}$						
with BOOKS + WIKI	13GB	256	1M	90.9/81.8	86.6	93.7

[Liu et al., 2019; Joshi et al., 2020]

Limitations of pretrained encoders

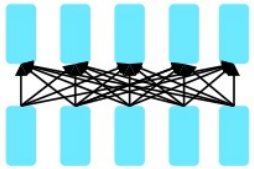
- Those results looked great! Why not used pretrained encoders for everything?
- If your task involves generating sequences, consider using a pretrained decoder.
- BERT and other pretrained encoders don't naturally lead to nice autoregressive (1-word-at-a-time) generation methods.





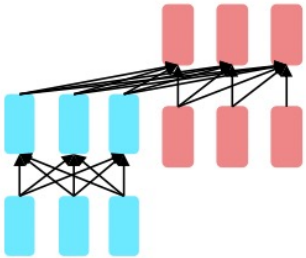
Pretraining for three types of architectures

The neural architecture influences the type of pretraining, and natural use cases.



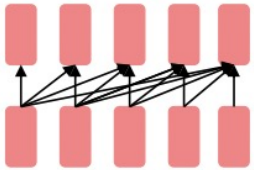
Encoders

- Gets bidirectional context – can condition on future!
- How do we train them to build strong representations?



Encoder-Decoders

- Good parts of decoders and encoders?
- What's the best way to pretrain them?



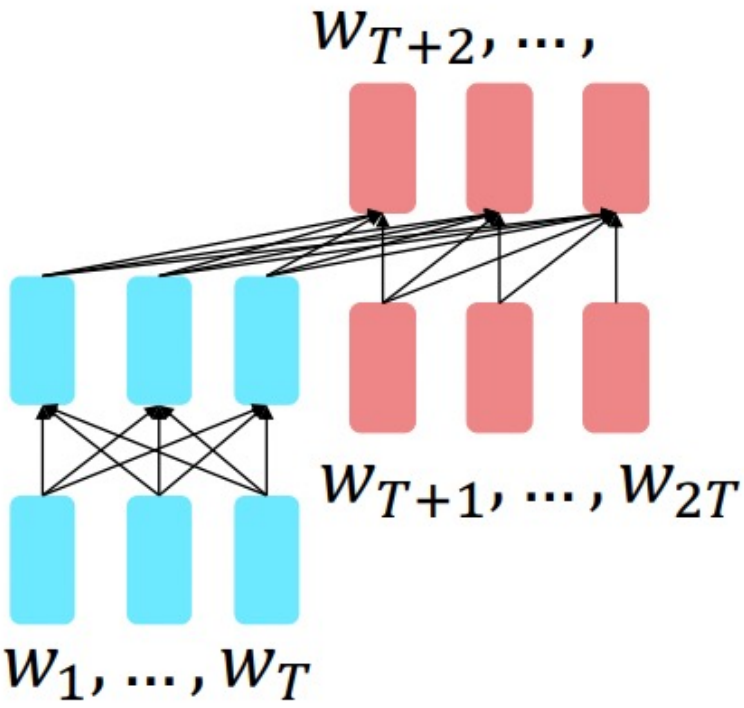
Decoders

- Language models! What we've seen so far.
- Nice to generate from; can't condition on future words

Pretraining encoder-decoders: what pretraining objective to use?

- For encoder-decoders, we could do something like language modeling, but where a prefix of every input is provided to the encoder and is not predicted.

$$h_1, \ldots, h_T = \text{Encoder}(w_1, \ldots, w_T)$$
$$h_{T+1}, \ldots, h_2 = Decoder(w_1, \ldots, w_T, h_1, \ldots, h_T)$$
$$y_i \sim Ah_i + b, i > T$$



- The encoder portion benefits from bidirectional context.

- The decoder portion is used to train the whole model through language modeling.

[Raffel et al., 2018]

Pretraining encoder-decoders: what pretraining objective to use?

- What [Raffel et al., 2018] found to work best was span corruption. Their model: Text-to-Text Transfer Transformer (T5).
- Replace different-length spans from the input with unique placeholders; decode out the spans that were removed!

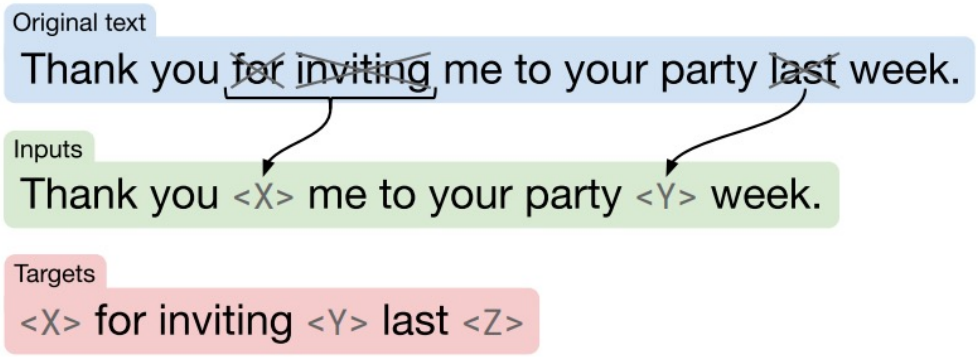


Figure 2: Schematic of the objective we use in our baseline model. In this example, we process the sentence "Thank you for inviting me to your party last week." The words "for", "inviting" and "last" (marked with an ×) are randomly chosen for corruption. Each consecutive span of corrupted tokens is replaced by a sentinel token (shown as **<X>** and **<Y>**) that is unique over the example. Since "for" and "inviting" occur consecutively, they are replaced by a single sentinel **<X>**. The output sequence then consists of the dropped-out spans, delimited by the sentinel tokens used to replace them in the input plus a final sentinel token **<Z>**.

Pretraining encoder-decoders: what pretraining objective to use?

- Raffel et al., 2018 found:
  - Encoder-decoders to work better than decoders for their tasks.
  - Span corruption (denoising) to work better than language modeling.

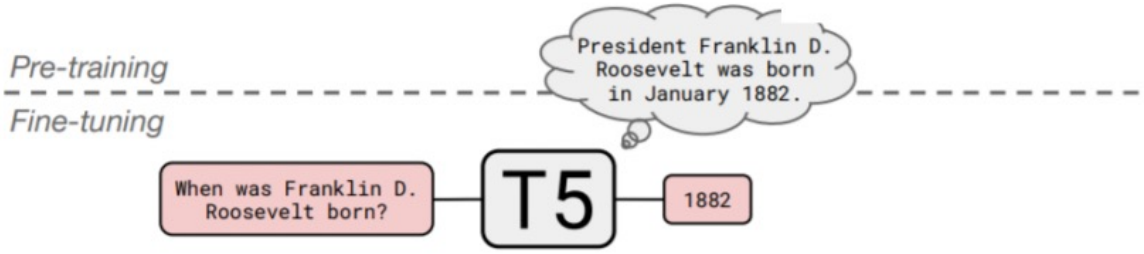
Architecture	Objective	Params	Cost	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
★ Encoder-decoder	Denoising	$2P$	M	**83.28**	**19.24**	**80.88**	**71.36**	**26.98**	**39.82**	**27.65**
Enc-dec, shared	Denoising	P	M	82.81	18.78	**80.63**	**70.73**	26.72	39.03	**27.46**
Enc-dec, 6 layers	Denoising	P	$M/2$	80.88	18.97	77.59	68.42	26.38	38.40	26.95
Language model	Denoising	P	M	74.70	17.93	61.14	55.02	25.09	35.28	25.86
Prefix LM	Denoising	P	M	81.82	18.61	78.94	68.11	26.43	37.98	27.39
Encoder-decoder	LM	$2P$	M	79.56	18.59	76.02	64.29	26.27	39.17	26.86
Enc-dec, shared	LM	P	M	79.60	18.13	76.35	63.50	26.62	39.17	27.05
Enc-dec, 6 layers	LM	P	$M/2$	78.67	18.26	75.32	64.06	26.13	38.42	26.89
Language model	LM	P	M	73.78	17.54	53.81	56.51	25.23	34.31	25.38
Prefix LM	LM	P	M	79.68	17.84	76.87	64.86	26.28	37.51	26.76

Pretraining encoder-decoders: what pretraining objective to use?

A fascinating property of T5: it can be finetuned to answer a wide range of questions, retrieving knowledge from its parameters.

- NQ: Natural Questions
- WQ: WebQuestions
- TQA: Trivia QA

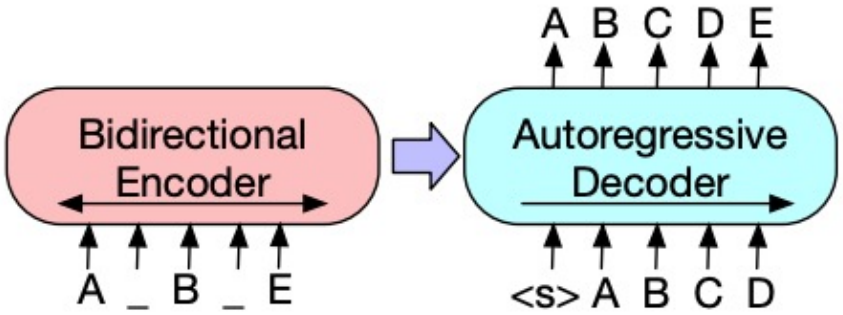
All “open-domain” versions



	NQ	WQ	TQA dev	TQA test	
Karpukhin et al. (2020)	**41.5**	42.4	**57.9**	–	
T5.1.1-Base	25.7	28.2	24.2	30.6	**220 million params**
T5.1.1-Large	27.3	29.5	28.5	37.2	**770 million params**
T5.1.1-XL	29.5	32.4	36.0	45.1	**3 billion params**
T5.1.1-XXL	32.8	35.6	42.9	52.5	**11 billion params**
T5.1.1-XXL + SSM	35.2	**42.8**	51.9	**61.6**	

BART

- BART pre-trains a model combining Bidirectional and Auto-Regressive Transformers.
- A denoising autoencoder for pretraining sequence-to-sequence models. BART is trained by
  - Corrupting text with an arbitrary noising function
  - Learning a model to reconstruct the original text
- It uses a standard Transformer-based neural machine translation architecture.



(c) BART: Inputs to the encoder need not be aligned with decoder outputs, allowing arbitary noise transformations. Here, a document has been corrupted by replacing spans of text with mask symbols. The corrupted document (left) is encoded with a bidirectional model, and then the likelihood of the original document (right) is calculated with an autoregressive decoder. For fine-tuning, an uncorrupted document is input to both the encoder and decoder, and we use representations from the final hidden state of the decoder.

BART Performance

	CNN/DailyMail			XSum		
	R1	R2	RL	R1	R2	RL
Lead-3	40.42	17.62	36.67	16.30	1.60	11.95
PTGEN (See et al., 2017)	36.44	15.66	33.42	29.70	9.21	23.24
PTGEN+COV (See et al., 2017)	39.53	17.28	36.38	28.10	8.02	21.72
UniLM	43.33	20.21	40.51	-	-	-
BERTSUMABS (Liu & Lapata, 2019)	41.72	19.39	38.76	38.76	16.33	31.15
BERTSUMEXTABS (Liu & Lapata, 2019)	42.13	19.60	39.18	38.81	16.50	31.27
BART	**44.16**	**21.28**	**40.90**	**45.14**	**22.27**	**37.25**

Table 3: Results on two standard summarization datasets. BART outperforms previous work on summarization on two tasks and all metrics, with gains of roughly 6 points on the more abstractive dataset.

	ELI5		
	R1	R2	RL
Best Extractive	23.5	3.1	17.5
Language Model	27.8	4.7	23.1
Seq2Seq	28.3	5.1	22.8
Seq2Seq Multitask	28.9	5.4	23.1
BART	**30.6**	**6.2**	**24.3**

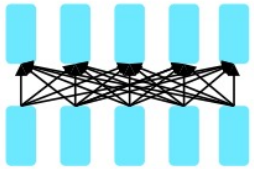
Table 5: BART achieves state-of-the-art results on the challenging ELI5 abstractive question answering dataset. Comparison models are from Fan et al. (2019).

	ConvAI2	
	Valid F1	Valid PPL
Seq2Seq + Attention	16.02	35.07
Best System	19.09	17.51
BART	**20.72**	**11.85**

Table 4: BART outperforms previous work on conversational response generation. Perplexities are renormalized based on official tokenizer for ConvAI2.

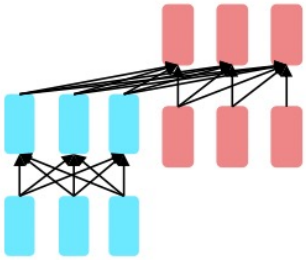
Pretraining for three types of architectures

The neural architecture influences the type of pretraining, and natural use cases.



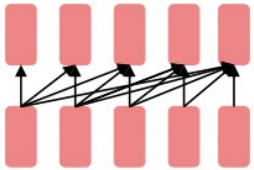
Encoders

- Gets bidirectional context – can condition on future!
- How do we train them to build strong representations?



Encoder-Decoders

- Good parts of decoders and encoders?
- What's the best way to pretrain them?



Decoders

- Language models! What we've seen so far.
- Nice to generate from; can't condition on future words

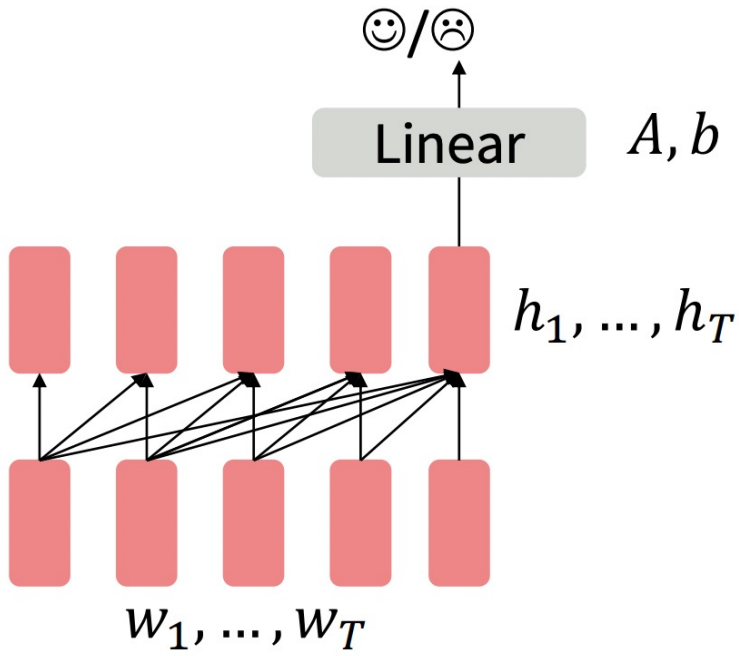
Pretraining decoders

- When using language model pretrained decoders, we can ignore that they were trained to model $p(w_t|w_{1:t-1})$

- We can finetune them by training a classifier on the last word's hidden state.

$$h_1, \ldots, h_T = \text{Decoder}(w_1, \ldots, w_T)$$
$$y \sim Ah_T + b$$

- Where A and b are randomly initialized and specified by the downstream task.
- Gradients backpropagate through the whole network.



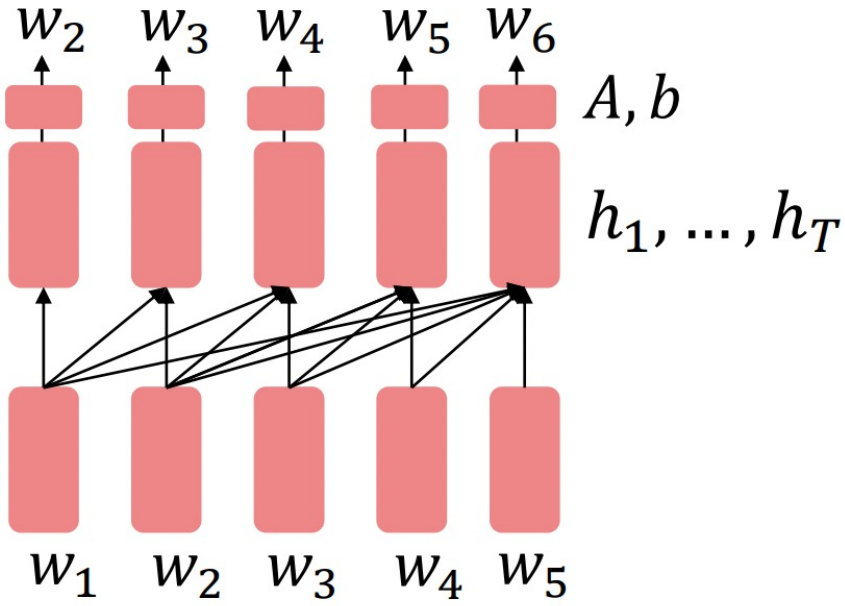
[Note how the linear layer hasn't been pretrained and must be learned from scratch.]

Pretraining decoders

- It's natural to pretrain decoders as language models and then use them as generators, finetuning their $p(w_t|w_{1:t-1})$

- This is helpful in tasks where the output is a sequence with a vocabulary like that at pretraining time!
  - Dialogue (context=dialogue history)
  - Summarization (context=document)

$$h_1, \ldots, h_T = \text{Decoder}(w_1, \ldots, w_T)$$
$$w_t \sim Ah_{t-1} + b$$



- Where A, b were pretrained in the language model!

Generative Pretrained Transformer (GPT)

2018's GPT was a big success in pretraining a decoder!

- Transformer decoder with 12 layers, 117M parameters.
- 768-dimensional hidden states, 3072-dimensional feed-forward hidden layers.
- Byte-pair encoding with 40,000 merges
- Trained on BooksCorpus: over 7000 unique books.
  - Contains long spans of contiguous text, for learning long-distance dependencies.
- The acronym "GPT" never showed up in the original paper; it could stand for "Generative PreTraining" or "Generative Pretrained Transformer"

[Radford et al., 2018]

Generative Pretrained Transformer (GPT)

- How do we format inputs to our decoder for finetuning tasks?
- Radford et al., 2018 evaluate on natural language inference.
- Natural Language Inference: Label pairs of sentences as entailing/contradictory/neutral
  - Premise: The man is in the doorway
  - Hypothesis: The person is near the door
- Here's roughly how the input was formatted, as a sequence of tokens for the decoder.

[START] The man is in the doorway [DELIMITER] The person is near the door [EXTRACT]

- The linear classifier is applied to the representation of the [EXTRACT] token.

[Radford et al., 2018]

Generative Pretrained Transformer (GPT)

GPT results on various natural language inference datasets.

Method	MNLI-m	MNLI-mm	SNLI	SciTail	QNLI	RTE
ESIM + ELMo [44] (5x)	-	-	89.3	-	-	-
CAFE [58] (5x)	80.2	79.0	89.3	-	-	-
Stochastic Answer Network [35] (3x)	80.6	80.1	-	-	-	-
CAFE [58]	78.7	77.9	88.5	83.3		
GenSen [64]	71.4	71.3	-	-	82.3	59.2
Multi-task BiLSTM + Attn [64]	72.2	72.1	-	-	82.1	**61.7**
Finetuned Transformer LM (ours)	**82.1**	**81.4**	**89.9**	**88.3**	**88.1**	56.0

[Radford et al., 2018]

Increasingly convincing generations (GPT2)

- We mentioned how pretrained decoders can be used in their capacities as language models. GPT-2, a larger version (1.5B) of GPT trained on more data, was shown to produce relatively convincing samples of natural language.

Context (human-written): In a shocking finding, scientist discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English.
GPT-2: The scientist named the population, after their distinctive horn, Ovid's Unicorn. These four-horned, silver-white unicorns were previously unknown to science. Now, after almost two centuries, the mystery of what sparked this odd phenomenon is finally solved. Dr. Jorge Pérez, an evolutionary biologist from the University of La Paz, and several companions, were exploring the Andes Mountains when they found a small valley, with no other animals or humans. Pérez noticed that the valley had what appeared to be a natural fountain, surrounded by two peaks of rock and silver snow.

[Radford et al., 2018]

GPT-3, In-context learning, and very large models

- So far, we've interacted with pretrained models in two ways:
  - Fine-tune them on a task we care about, and take their predictions.
  - Sample from the distributions they define (maybe providing a prompt)
- Very large language models seem to perform some kind of learning without gradient steps simply from examples you provide within their contexts.
- GPT-3 is the canonical example of this.
  - The largest T5 model had 11 billion parameters.
  - GPT-3 has 175 billion parameters.

GPT-3, In-context learning, and very large models

- Very large language models seem to perform some kind of learning without gradient steps simply from examples you provide within their contexts.

- The in-context examples seem to specify the task to be performed, and the conditional distribution mocks performing the task to a certain extent.
- **Input (prefix within a single Transformer decoder context):**
  - “ thanks -> merci
  - hello -> bonjour
  - mint -> menthe
  - otter -> ”
- **Output (conditional generations):**
  - loutre…”

Heart of the LLMs: In-context learning

The model learns about the context based on the examples provided.

- Enabling GPT models to understand/comprehend and create text that closely resembles human speech, based on the instructions and examples they're provided.

- Three approaches to in-context learning:
  - **Few-shot**: includes several examples in the prompt to demonstrate the expected answer format and content.
  - **One-shot**: only a single example is provided in the prompt to demonstrate the desired task.
  - **Zero-shot**: no examples are provided to the model during the generation call. Instead, only the task or request is given as input.

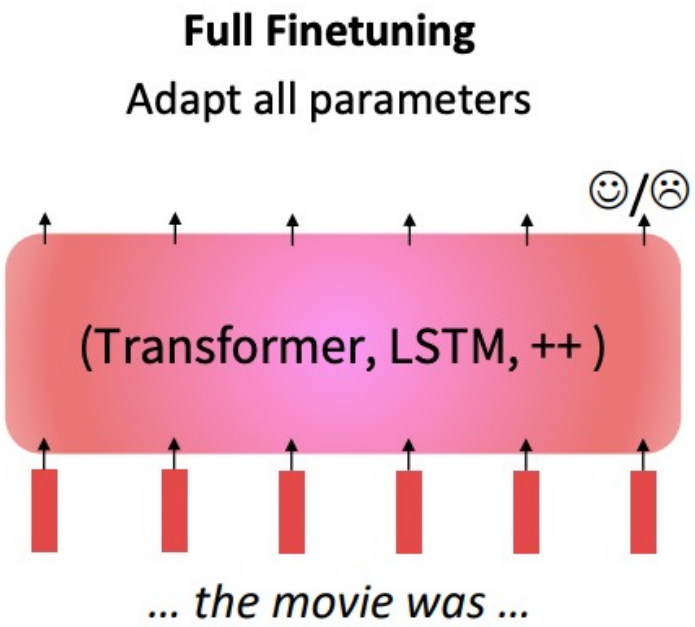
https://vitalflux.com/in-context-learning-gpt-3-concepts-examples/

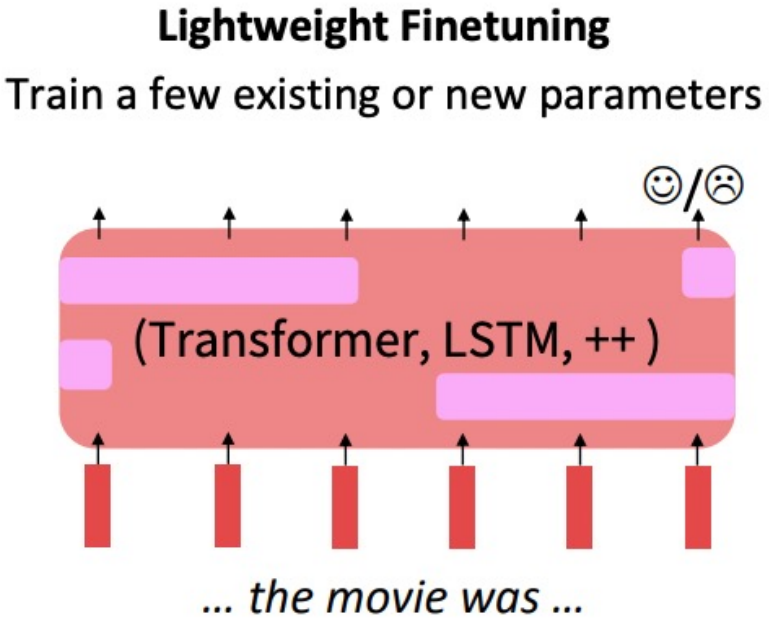
Finetuning

- After pre-training, LLMs can acquire the general abilities for solving various tasks.
- However, an increasing number of studies have shown that LM's abilities can be further adapted according to specific goals.
- **Finetuning:**
  - Full Finetuning
  - Parameter-Efficient Finetuning

Full Finetuning vs. Parameter-Efficient Finetuning

- **Full finetuning**: Finetuning every parameter in a pretrained model works well, but is memory-intensive.
- **Parameter-efficient finetuning**: But lightweight finetuning methods adapt pretrained models in a constrained way. Leads to less overfitting and/or more efficient finetuning and inference.

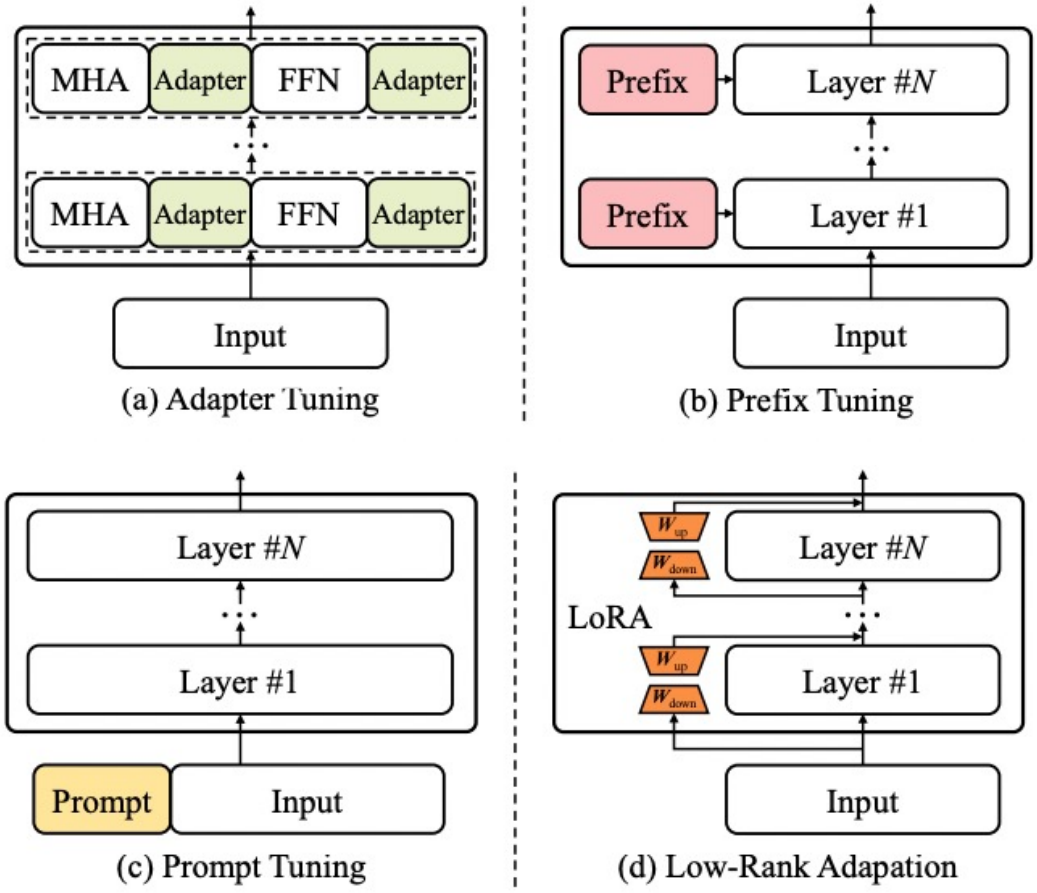




[Liu et al., 2019; Joshi et al., 2020]

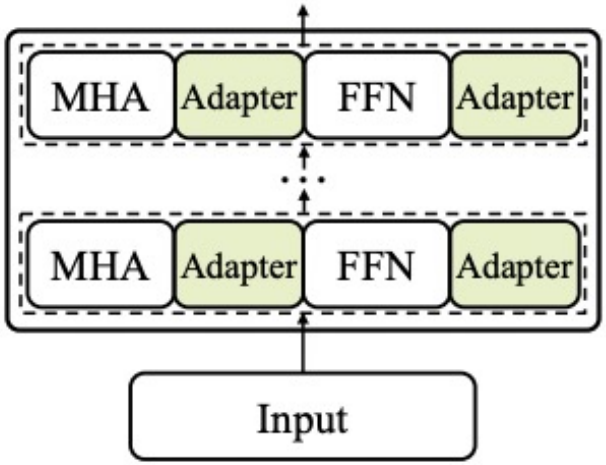
Parameter-Efficient Fine-Tuning Methods

- Aims to reduce the number of trainable parameters while retaining a good performance as possible
- Four parameter-efficient fine-tuning methods for Transformer language models.



Parameter-Efficient Finetuning: Adapter Tuning

- Incorporates small neural network modules (called adapter) into each Transformer layer.
- First compresses the original feature vector into a smaller dimension (followed by a nonlinear transformation) and then recovers it to the original dimension.
- During fine-tuning, the adapter modules would be optimized according to the specific task goals, while the parameters of the original language model are frozen in this process.

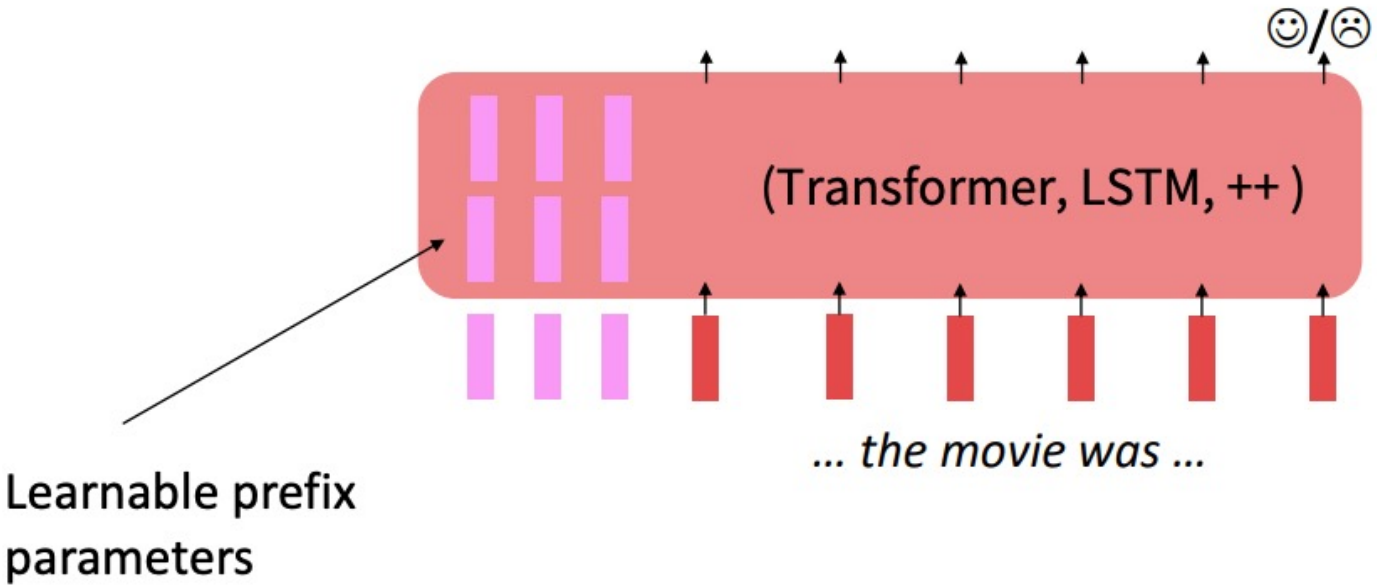


(a) Adapter Tuning

This effectively reduce the number of trainable parameters during fine-tuning.

Parameter-Efficient Finetuning: Prefix Tuning

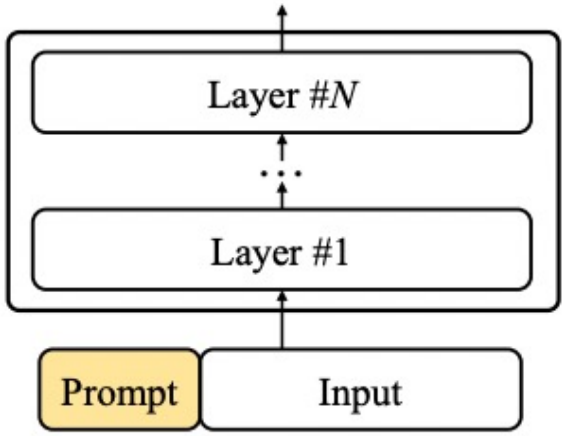
- Prefix-Tuning adds a prefix of parameters, and freezes all pretrained parameters.
  - The prefix is processed by the model just like real words would be.
  - Prepends a sequence of prefixes, which are a set of trainable continuous vectors, to each Transformer layer in language models.



[Li and Liang, 2021; Lester et al., 2021]

Parameter-Efficient Finetuning: Prompt Tuning

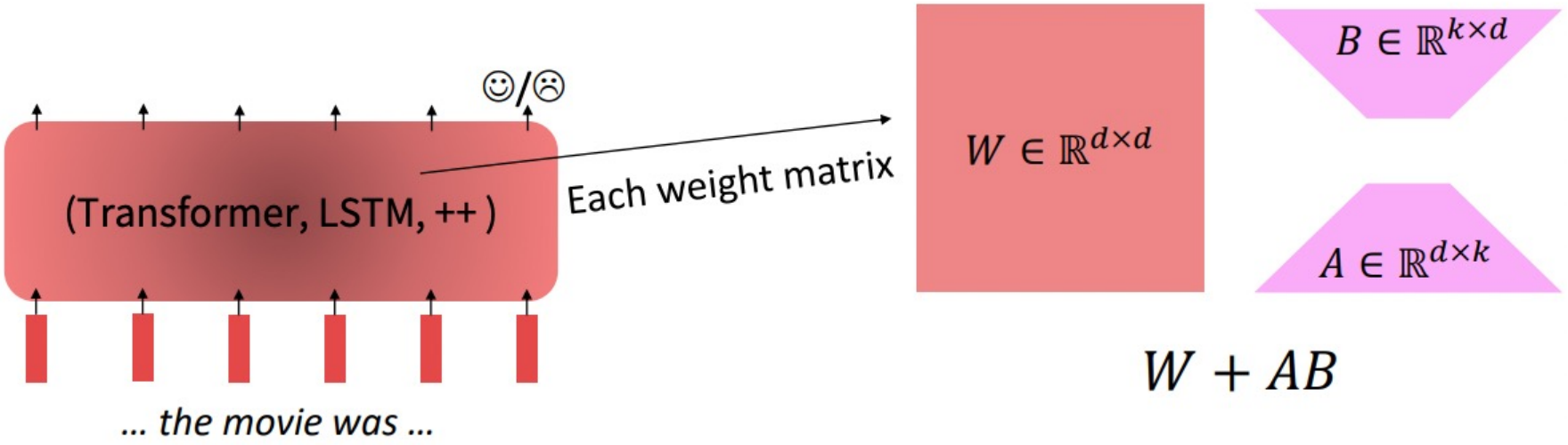
- Incorporate trainable prompt vectors at the input layer.
  - Augments the input text by including a group of soft prompt tokens (either in a free form or a prefix form).
  - Then takes the prompt-augmented input to solve specific downstream tasks.
- During training, only the prompt embeddings would be learned according to task-specific supervisions.



(c) Prompt Tuning

Parameter-Efficient Finetuning: Low-Rank Adaptation (LoRa)

- Low-Rank Adaptation Learns a low-rank "diff" between the pretrained and finetuned weight matrices.
- Easier to learn than prefix-tuning.



A and B are trainable parameters for task adaption. k is the reduced rank.

[Hu et al., 2021]

Parameter-Efficient Finetuning

Hugging Face PEFT: https://github.com/huggingface/peft/tree/main

State-of-the-art Parameter-Efficient Fine-Tuning (PEFT) methods

Parameter-Efficient Fine-Tuning (PEFT) methods enable efficient adaptation of pre-trained language models (PLMs) to various downstream applications without fine-tuning all the model's parameters. Fine-tuning large-scale PLMs is often prohibitively costly. In this regard, PEFT methods only fine-tune a small number of (extra) model parameters, thereby greatly decreasing the computational and storage costs. Recent State-of-the-Art PEFT techniques achieve performance comparable to that of full fine-tuning.

Seamlessly integrated with Accelerate for large scale models leveraging DeepSpeed and Big Model Inference.

Supported methods:

1. LoRA: LORA: LOW-RANK ADAPTATION OF LARGE LANGUAGE MODELS
2. Prefix Tuning: Prefix-Tuning: Optimizing Continuous Prompts for Generation, P-Tuning v2: Prompt Tuning Can Be Comparable to Fine-tuning Universally Across Scales and Tasks
3. P-Tuning: GPT Understands, Too
4. Prompt Tuning: The Power of Scale for Parameter-Efficient Prompt Tuning
5. AdaLoRA: Adaptive Budget Allocation for Parameter-Efficient Fine-Tuning
6. $(IA)^3$: Few-Shot Parameter-Efficient Fine-Tuning is Better and Cheaper than In-Context Learning
7. MultiTask Prompt Tuning: Multitask Prompt Tuning Enables Parameter-Efficient Transfer Learning
8. LoHa: FedPara: Low-Rank Hadamard Product for Communication-Efficient Federated Learning

Hugging Face Transformer

Hugging Face Transformer:

A big open-source library with most pre-trained architectures implemented, weights available.

Hugging Face Transformer tutorial

https://colab.research.google.com/drive/1pxc-ehTtnVM72-NViET_D2ZqOlpOi2LH?usp=sharing

Readings

1. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding
2. Contextual Word Representations: A Contextual Introduction
3. The Illustrated BERT, ELMo, and co.
4. Martin & Jurafsky Chapter on Transfer Learning

Slides prepared based on

- A Survey of Large Language Models
- CS224: Natural language processing with deep learning at Stanford
- CS388: Natural Language Processing at UT Austin

STEVENS

INSTITUTE *of* TECHNOLOGY

THE INNOVATION UNIVERSITY®

stevens.edu

Thank You