Large Language Models (LLMs): Adapting to New Tasks

M. Soleymani
Sharif University of Technology
Spring 2026

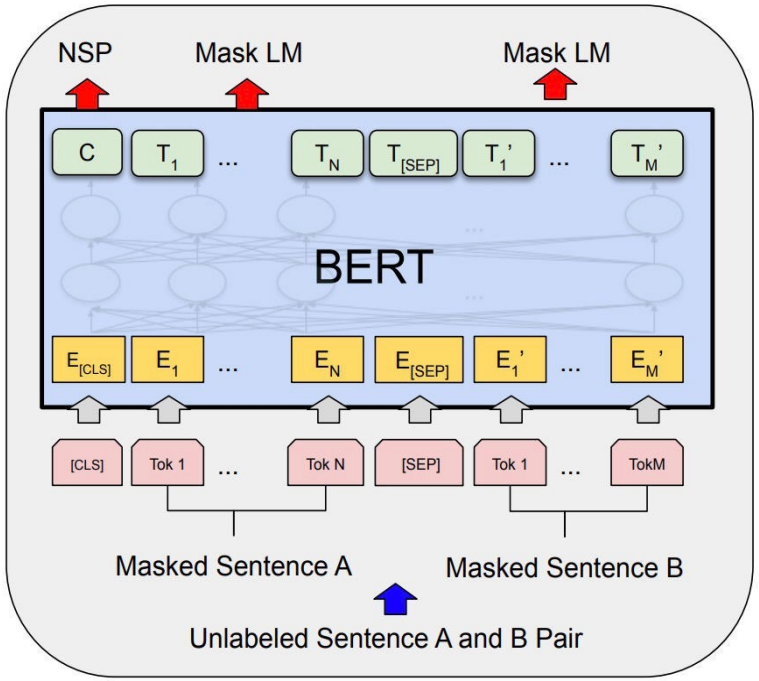
Some slides have been adopted from Matt Gormley's slides, 10-423/10-623 Generative AI course, CMU and some slides from Yang and Hashimoto lectures cs224n, Stanford

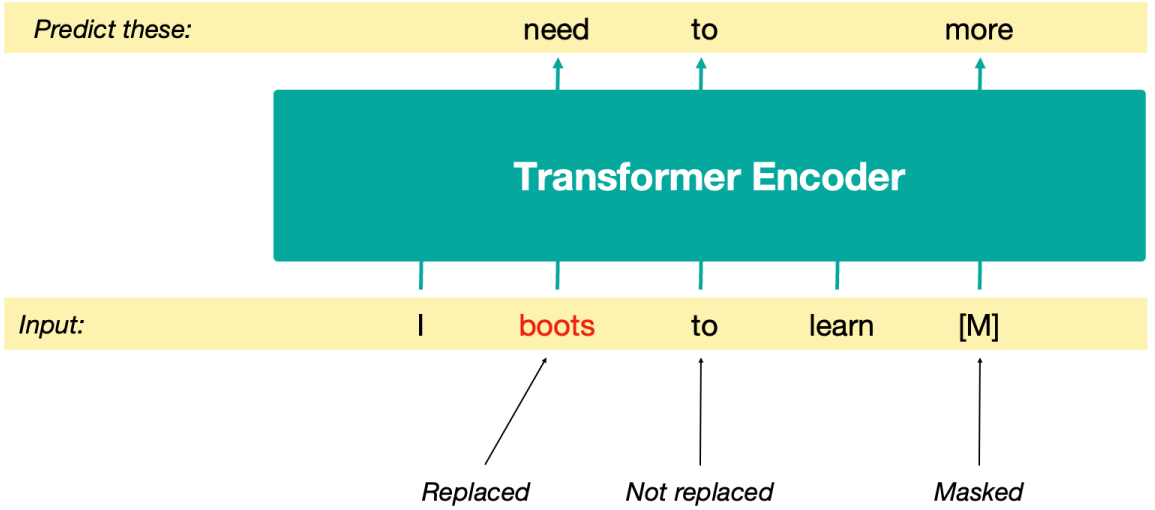
Outline

- Review of Large Language Models
- In-context learning
- Parameter Efficient Fine-Tuning (PEFT)
  - Adapter
  - Prompt tuning & prefix tuning
  - LoRA

Language models

- **Encoder-only models (BERT, RoBERTa, ELECTRA)**
- Encoder-decoder models (T5, BART)
- Decoder-only models (GPT-n models)

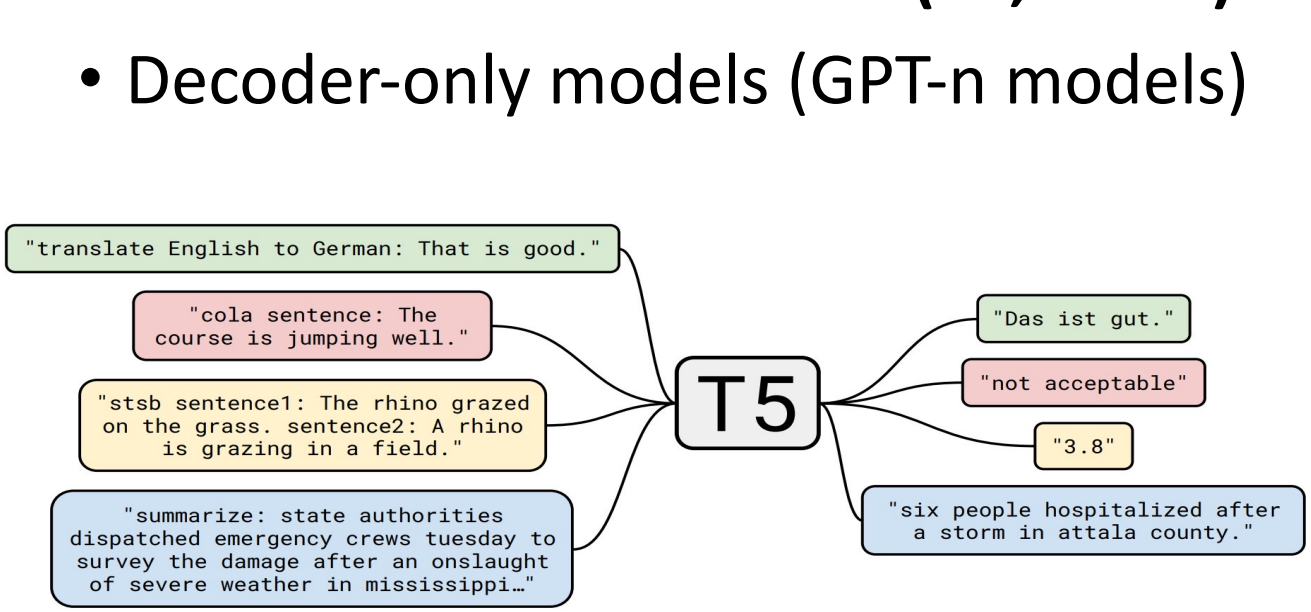


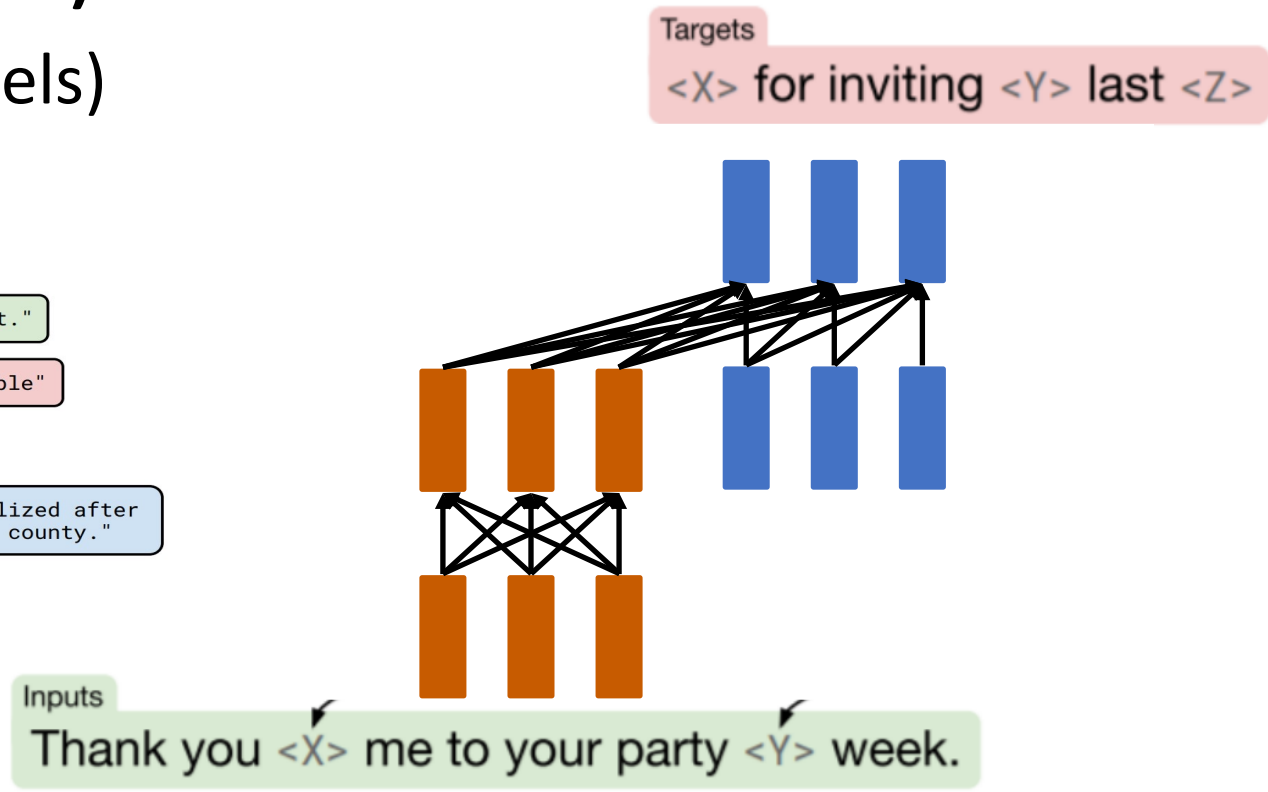


Devlin et al., BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, 2018

Language models

- Encoder-only models (BERT, RoBERTa, ELECTRA)
- **Encoder-decoder models (T5, BART)**
- Decoder-only models (GPT-n models)

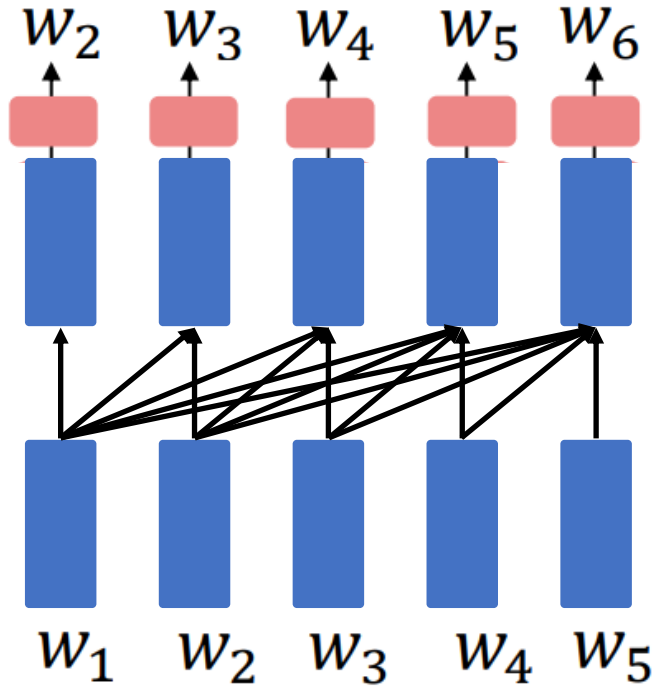




Raffel et al., Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer, 2018

Language models

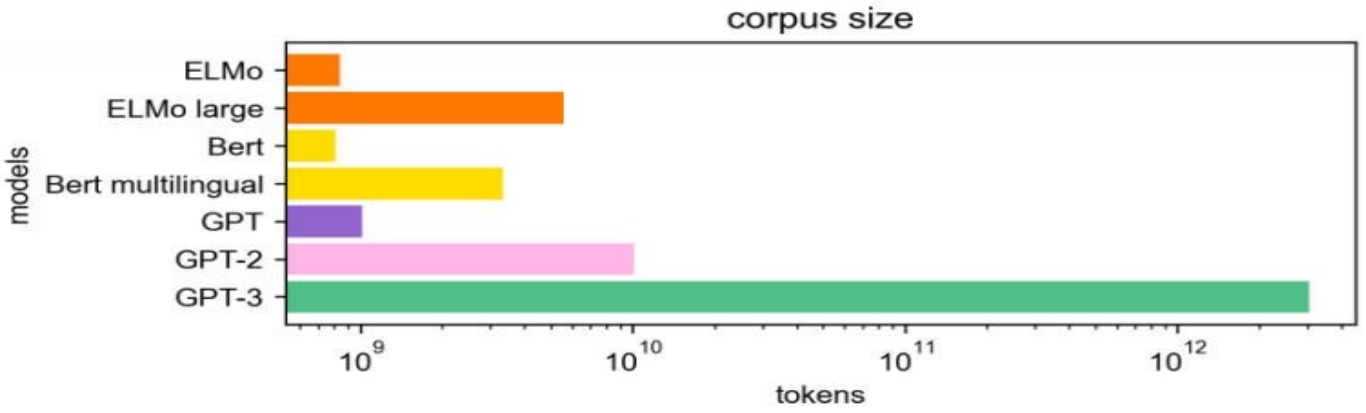
- Encoder-only models (BERT, RoBERTa, ELECTRA)
- Encoder-decoder models (T5, BART)
- **Decoder-only models (GPT-n models)**



Brown et al., Language Models are Few-Shot Learners, 2020

Large Language Models (LLMs)

- Scale: Increasing the size
  - Small or medium-sized models: BERT/RoBERTa models (100M or 300M), T5 models (220M, 770M, 3B)
  - Large language models: dozens of billion parameters
  - "Very" large LMs: models of 100+ billion parameters



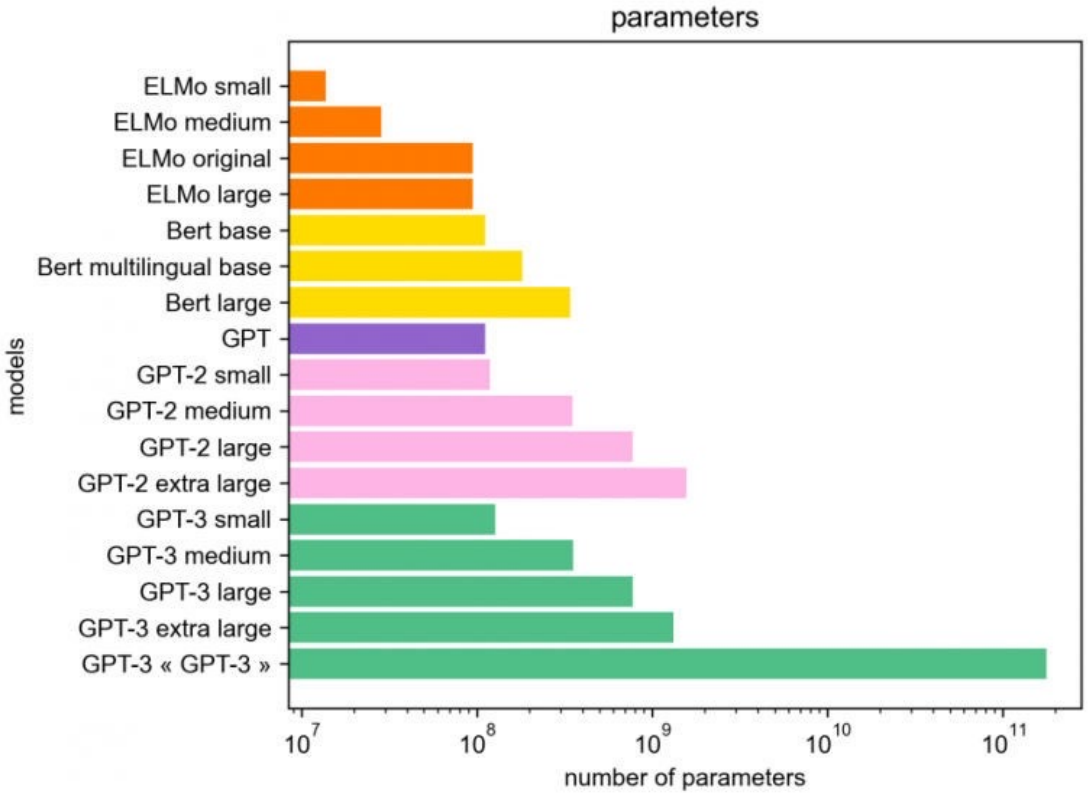


Image source: https://hellofuture.orange.com/en/the-gpt-3-language-model-revolution-or-evolution/

Scaling Efficiency: how do we best use our compute

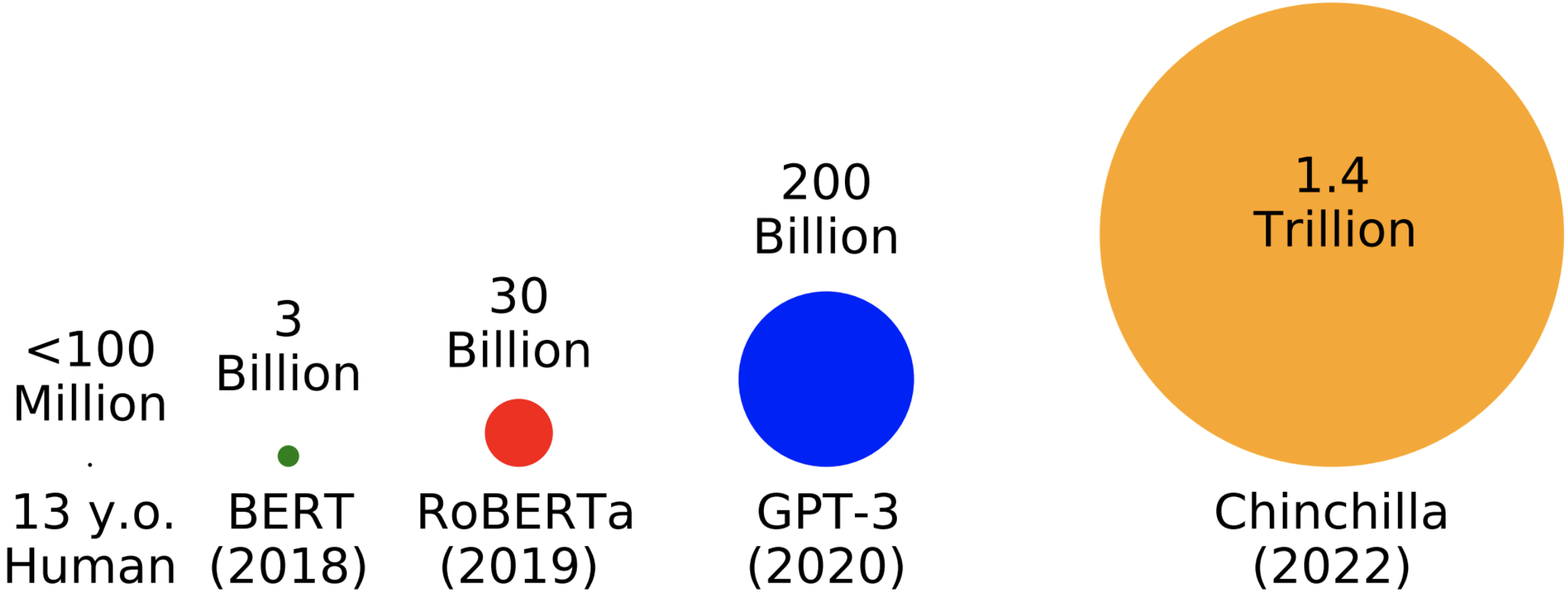
- GPT-3 was 175B parameters and trained on 300B tokens of text.
- Roughly, the cost of training a large transformer scales as **parameters*tokens**
- Did OpenAI strike the right parameter-token data to get the best model? No.

Model	Size (# Parameters)	Training Tokens
LaMDA (Thoppilan et al., 2022)	137 Billion	168 Billion
GPT-3 (Brown et al., 2020)	175 Billion	300 Billion
Jurassic (Lieber et al., 2021)	178 Billion	300 Billion
Gopher (Rae et al., 2021)	280 Billion	300 Billion
MT-NLG 530B (Smith et al., 2022)	530 Billion	270 Billion
Chinchilla	70 Billion	1.4 Trillion

This 70B parameter model is better than the much larger other models!

Brown et al., Language Models are Few-Shot Learners, 2020

Language models trained on more and more data



https://babylm.github.io/

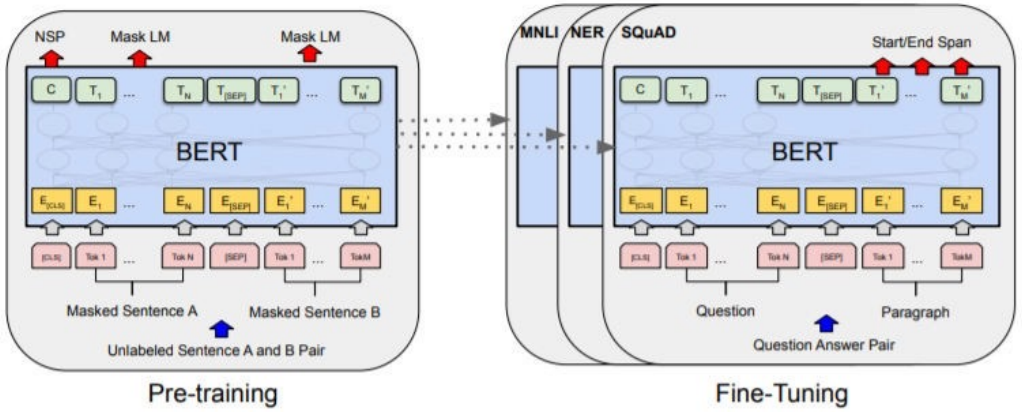
Size of LLMs

- Compute-optimal models
  - Larger sizes ⇒ larger compute, more expensive inference
  - Trade-off between model size and corpus size

- Different sizes of LMs have different ways to adapt and use them
  - Fine-tuning, zero-shot/few-shot prompting, …

Pre-training and adaptation

- Pre-training: trained on huge datasets of unlabeled text
  - "self-supervised" learning approach
- Adaptation: how to adapt a pre-trained model for a downstream task or domain?
  - What types of NLP tasks (input and output formats)?
  - How many annotated samples?



Circulation revenue has increased by 5% in Finland. // Positive

Panostaja did not disclose the purchase price. // Neutral

Paying off the national debt will be extremely painful. // Negative

The company anticipated its operating profit to improve. // _____

LM

Circulation revenue has increased by 5% in Finland. // Finance

They defeated … in the NFC Championship Game. // Sports

Apple … development of in-house chips. // Tech

The company anticipated its operating profit to improve. // _____

LM

http://ai.stanford.edu/blog/understanding-incontext/

Conditioning: Autoregressive language models

- **Conditional generation**. Specifying a prefix $x_{1:i}$, called a **prompt**, and sampling the rest $x_{i+1:L}$

- Conditional generation unlocks the ability to solve a variety of tasks by simply changing the prompt.

Few-shot Learning: In-Context Learning (ICL)

- Given a **prompt** including:
  - An optional description of the task
  - Few-shot training examples in a prompt format demonstrating the task
  - The test input

“Apple → Red, Lime → Green, Corn →”

“Albert Einstein was German \n Mahatma Gandhi was Indian \n Marie Curie was”

One-shot

In addition to the task description, the model sees a single example of the task. No gradient updates are performed.

```
1  Translate English to French:        ← task description
2  sea otter => loutre de mer          ← example
3  cheese =>                           ← prompt
```

Few-shot

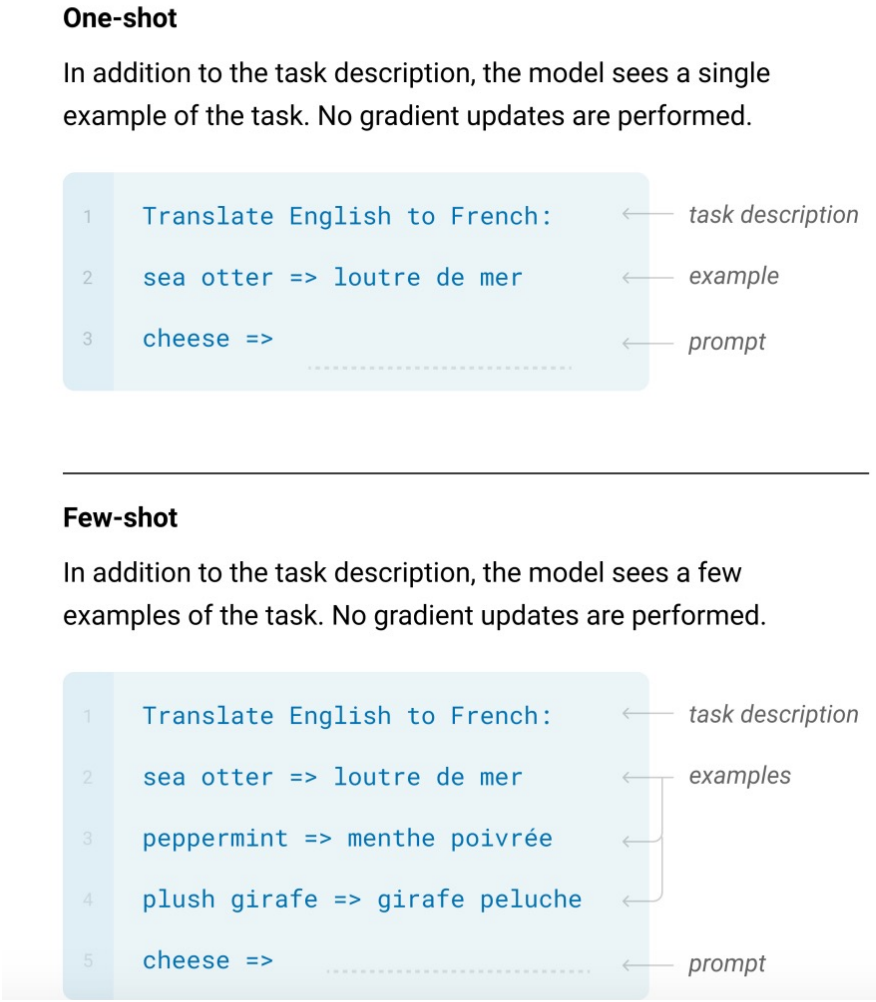
In addition to the task description, the model sees a few examples of the task. No gradient updates are performed.

```
1  Translate English to French:        ← task description
2  sea otter => loutre de mer          ← examples
3  peppermint => menthe poivrée        ←
4  plush girafe => girafe peluche      ←
5  cheese =>                           ← prompt
```


Brown et al., Language Models are Few-Shot Learners, 2020

Prompting paradigm

- Popularized by GPT-3 (Brown et al., 2020)
- A pre-trained LLM is given a **prompt** (e.g. an instruction) of a task and completes the response without any further training
- **In-context learning**: Brown et al. (2020) proposed few-shot prompting
  - includes a few input-output examples in the model's context (input) before asking the model to perform the task for an unseen example.
- Single model to solve many NLP tasks



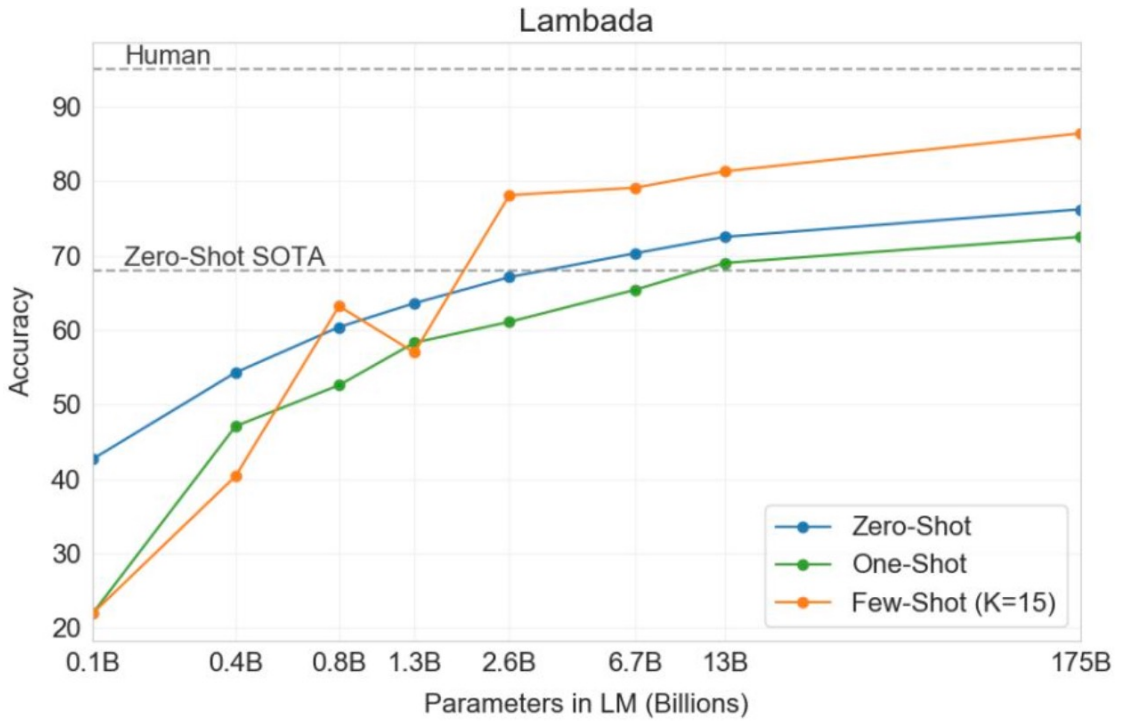
Brown et al., Language Models are Few-Shot Learners, 2020

What can ICL do?

- No parameter tuning is required
- Only need few examples for downstream tasks
- GPT-3 improved
  SOTA on LAMBADA by 18%!



Works like magic!



Brown et al., Language Models are Few-Shot Learners, 2020

Few-shot Learning with LLMs

Suppose you have…

- a dataset $D = \{(x_i, y_i)\}_{i=1}^{N}$ and N is rather small (i.e. few-shot setting)
- a very large (billions of parameters) pre-trained language model

There are two ways to "learn"

Option A: Supervised fine-tuning

- **Definition**: fine-tune the LLM on the training data using…
  - a standard supervised objective
  - backpropagation to compute gradients
  - your favorite optimizer (e.g. Adam)
- **Pro**: fits into the standard ML recipe
- **Pro:** still works if N is large
- **Con**: backpropagation requires ~3x the memory and computation time as the forward computation
- **Con**: you might not have access to the model weights at all (e.g. because the model is proprietary)

Option B: In-context learning

- **Definition**:
  1. feed training examples to the LLM as a prompt
  2. allow the LLM to infer patterns in the training examples during inference (i.e. decoding)
  3. take the output of the LLM following the prompt as its prediction
- **Con**: the prompt may be very long and Transformer LMs require $O(N^2)$ time/space where N=length of context
- **Pro**: no backpropagation required and only one pass through the training data
- **Pro**: does not require model weights, only API access

Few-shot Learning with LLMs

Suppose you have…

- a dataset $D = \{(x_i, y_i)\}_{i=1}^{N}$ and N is rather small (i.e. few-shot setting)
- a very large (billions of parameters) pre-trained language model

There are two ways to "learn"

Option A: Supervised fine-tuning

- **Definition**: fine-tune the LLM on the training data using…
  - a standard supervised objective
  - backpropagation to compute gradients
  - your favorite optimizer (e.g. Adam)
- **Pro**: fits into the standard ML recipe
- **Pro:** still works if N is large
- **Con**: backpropagation requires ~3x the memory and computation time as the forward computation
- **Con**: you might not have access to the model weights at all (e.g. because the model is proprietary)

Option B: In-context learning

In the section, we consider the question:
How can we do supervised fine- tuning of a very large foundation model more efficiently?

a prompt
training
ng)
he prompt
as its prediction

- **Con**: the prompt may be very long and Transformer LMs require $O(N^2)$ time/space where N=length of context
- **Pro**: no backpropagation required and only one pass through the training data
- **Pro**: does not require model weights, only API access

ICL: Advantages

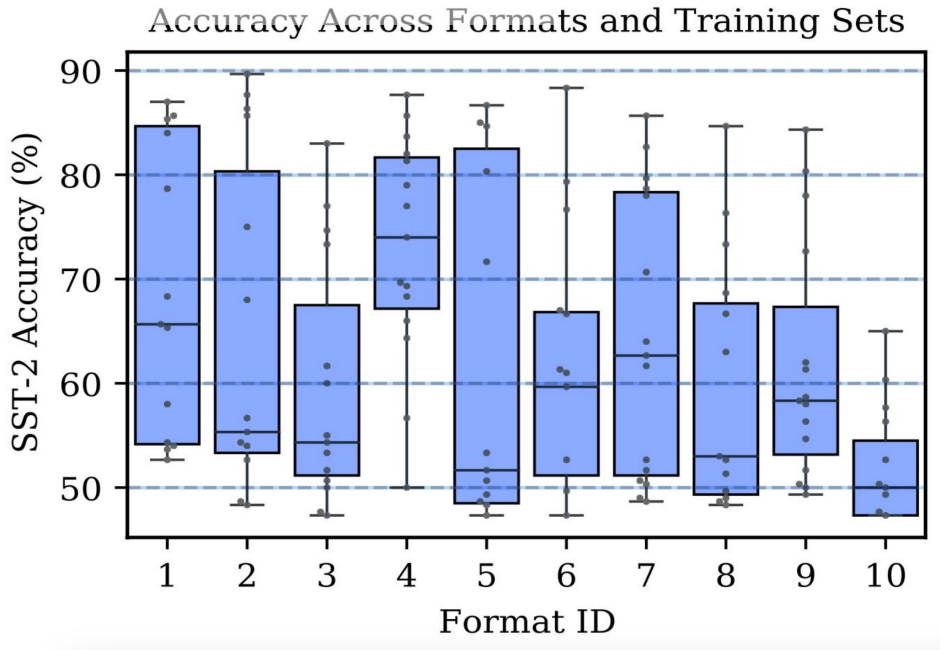
- enables rapid prototyping
  - No need to update the model weights at all!
- provides a fully natural language interface
- reuses the same model for each task
  - reduces memory requirements and system complexity when serving many different tasks.
- Finetuning can be unstable in the few-shot setting (Schick & Schutze, 2021)

Zhao, Calibrate Before Use: Improving Few-Shot Performance of Language Models, ICML 2021

Prompt structure is important

Components of a prompt:

1. Prompt format
2. Training example selection
3. Training example permutation



Format 1

Input: Subpar acting. **Sentiment:** negative
Input: Beautiful film. **Sentiment:** positive
Input: Amazing. Sentiment:

Format 2

Subpar acting. I hated the movie
Beautiful film. I liked the movie
Amazing.

Format 10

Review: Subpar acting. **Stars:** 0
Review: Beautiful film. **Stars:** 5
Review: Amazing. Stars:

Note

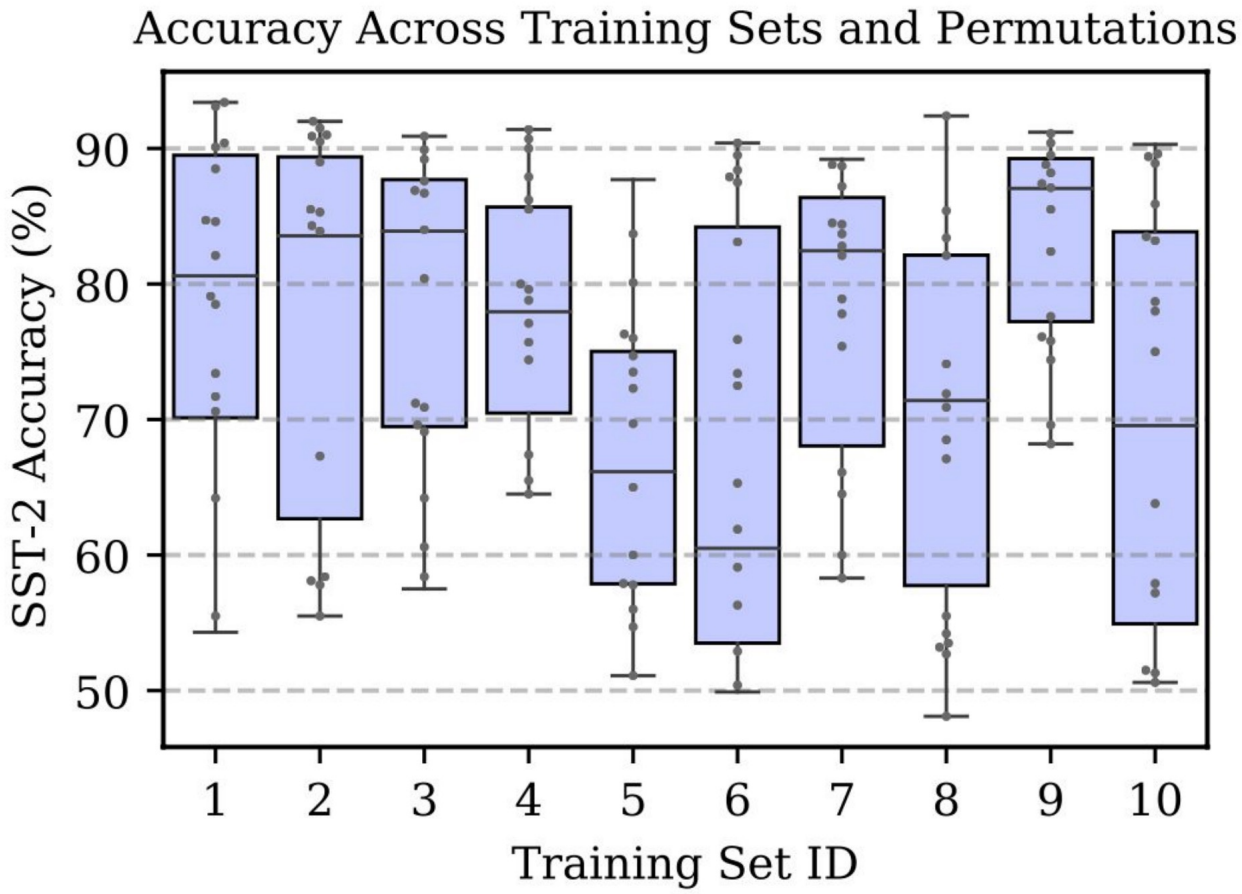
In-context learning is highly sensitive to prompt **format**

Zhao et al., Calibrate before Use: Improving Few-Shot Performance of Language Models, ICML 2021

Prompt structure is important

Components of a prompt:

1. Prompt format
2. Training example selection
3. Training example permutation



Note

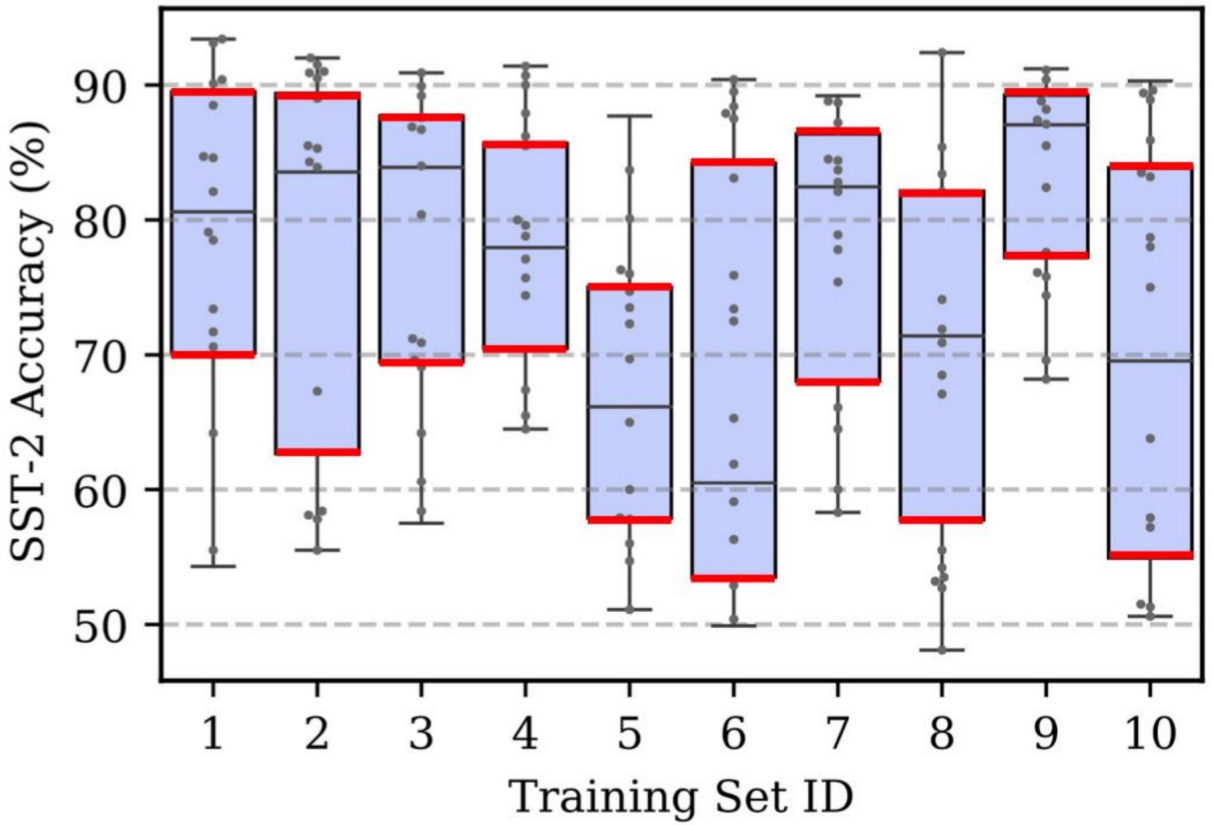
In-context learning is highly sensitive to example **selection**

Zhao et al., Calibrate before Use: Improving Few-Shot Performance of Language Models, ICML 2021

Prompt structure is important

Components of a prompt:

1. Prompt format
2. Training example selection
3. Training example permutation



Note

In-context learning is highly sensitive to example **permutation**

Zhao et al., Calibrate before Use: Improving Few-Shot Performance of Language Models, ICML 2021

ICL: Disadvantages

- ICL is inherently unstable
- Very sensitive to the prompt structure
- Can’t handle large number of training examples efficeintly and effectively
  - The model’s context size limits the number of demonstrations that can be used
  - Context may not be a proper way of feeding examples

- A surge of methods that search for robust and high-performing prompts:
  - Template search
    - all these methods require a high-quality validation set to do prompt selection or optimization
  - In-context example search

Wu et al., Self-Adaptive In-Context Learning: An Information Compression Perspective for In-Context Example Selection and Ordering, 2022

Fine-Tuning vs. In-Context Learning

- Why would we ever bother with fine-tuning if it's so inefficient?
- Because, even for very large LMs, fine-tuning often beats in-context learning

Method	MNLI-m (Val. Acc./%)	RTE (Val. Acc./%)
GPT-3 Few-Shot	40.6	69.0
GPT-3 Fine-Tuned	89.5	85.4

Question:

Why did fine-tuning of GPT-3 do so much better on these two tasks than few-shot in-context learning?

Answer:

Hu e al., LoRA: Low-Rank Adaptation of Large Language Models, 2021

Fine-Tuning vs. In-Context Learning

- Why would we ever bother with fine-tuning if it's so inefficient?
- Because, even for very large LMs, fine-tuning often beats in-context learning
- **In a fair comparison of fine-tuning (FT) and in-context learning (ICL), they find that FT outperforms ICL for most model sizes**

		FT						
		125M	350M	1.3B	2.7B	6.7B	13B	30B
ICL	125M	−0.00	0.01	0.02	0.03	0.12	0.14	0.09
	350M	−0.00	0.01	0.02	0.03	0.12	0.14	0.09
	1.3B	−0.00	0.01	0.02	0.03	0.12	0.14	0.09
	2.7B	−0.00	0.01	0.02	0.03	0.12	0.14	0.09
	6.7B	−0.00	0.01	0.02	0.03	0.12	0.14	0.09
	13B	−0.04	−0.02	−0.01	−0.00	0.09	0.11	0.05
	30B	−0.11	−0.09	−0.08	−0.08	0.02	0.03	−0.02

(a) RTE

		FT						
		125M	350M	1.3B	2.7B	6.7B	13B	30B
ICL	125M	−0.00	0.00	0.02	0.01	0.10	0.11	0.07
	350M	−0.00	0.00	0.02	0.01	0.10	0.11	0.07
	1.3B	−0.01	−0.00	0.01	0.01	0.10	0.11	0.07
	2.7B	−0.01	−0.00	0.01	0.01	0.09	0.10	0.07
	6.7B	−0.01	−0.01	0.01	0.00	0.09	0.10	0.06
	13B	−0.03	−0.03	−0.02	−0.02	0.07	0.08	0.04
	30B	−0.07	−0.07	−0.05	−0.06	0.03	0.04	0.00

(b) MNLI

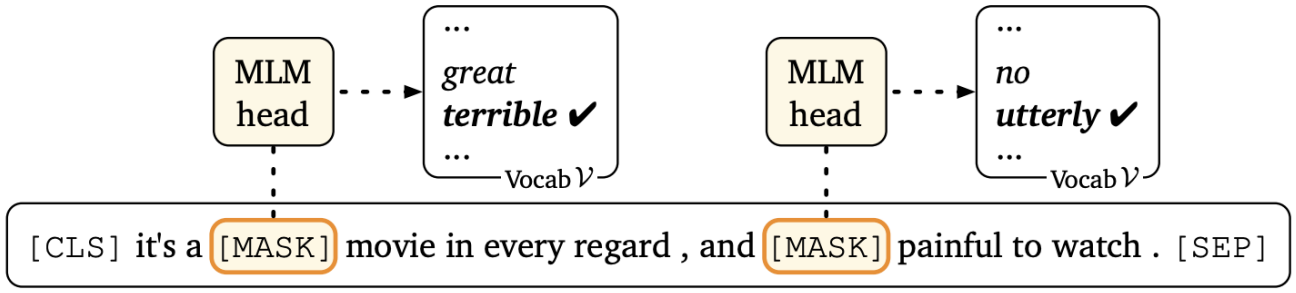
Table 1: Difference between average **out-of-domain performance** of ICL and FT on RTE (a) and MNLI (b) across model sizes. We use 16 examples and 10 random seeds for both approaches. For ICL, we use the `gpt-3` pattern. For FT, we use pattern-based fine-tuning (PBFT) and select checkpoints according to in-domain performance. We perform a Welch's t-test and color cells according to whether: ICL performs significantly better than FT, FT performs significantly better than ICL. For cells without color, there is no significant difference.

Mosbach et al., Few-shot Fine-tuning vs. In-context Learning: A Fair Comparison and Evaluation, ACL 2023

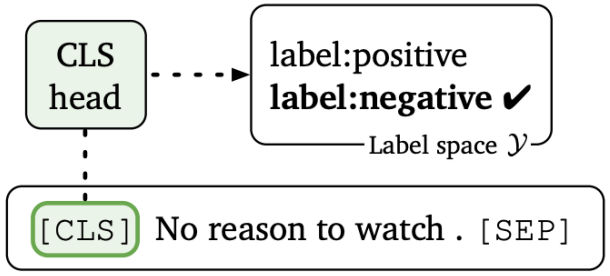
Ways to adapt to new tasks

- Few-shot learning
  - In-context learning
  - **Few-shot fine-tuning (i.e. a label word mapping)**
- Lightweight Fine-tuning
  - Prompt tuning (Lester et al., 2021)
  - Prefix tuning (Li and Liang, 2021)
  - Adapter (Houlsby et al. 2019) and LoRA (Hu et al., 2021)

Few-shot Learning: Prompt-based Fine-tuning



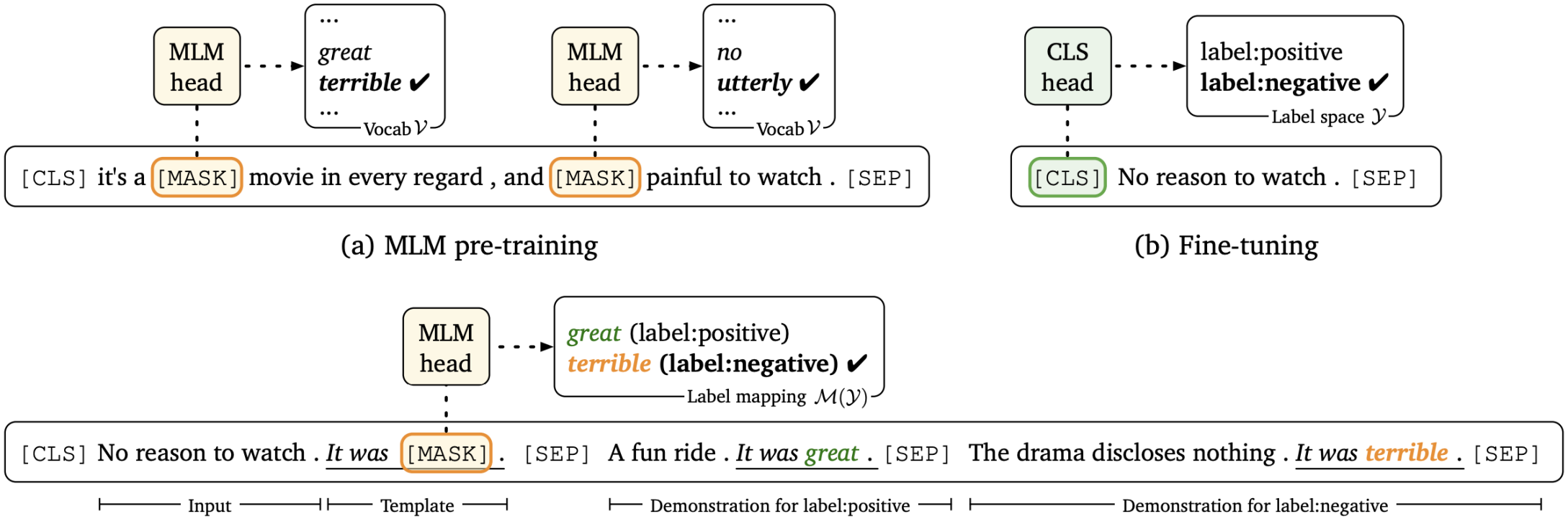
(a) MLM pre-training



(b) Fine-tuning

Gao et al., Making Pre-trained Language Models Better Few-shot Learners, ACL 2021

Few-shot Learning: Prompt-based Fine-tuning



Gao et al., Making Pre-trained Language Models Better Few-shot Learners, ACL 2021

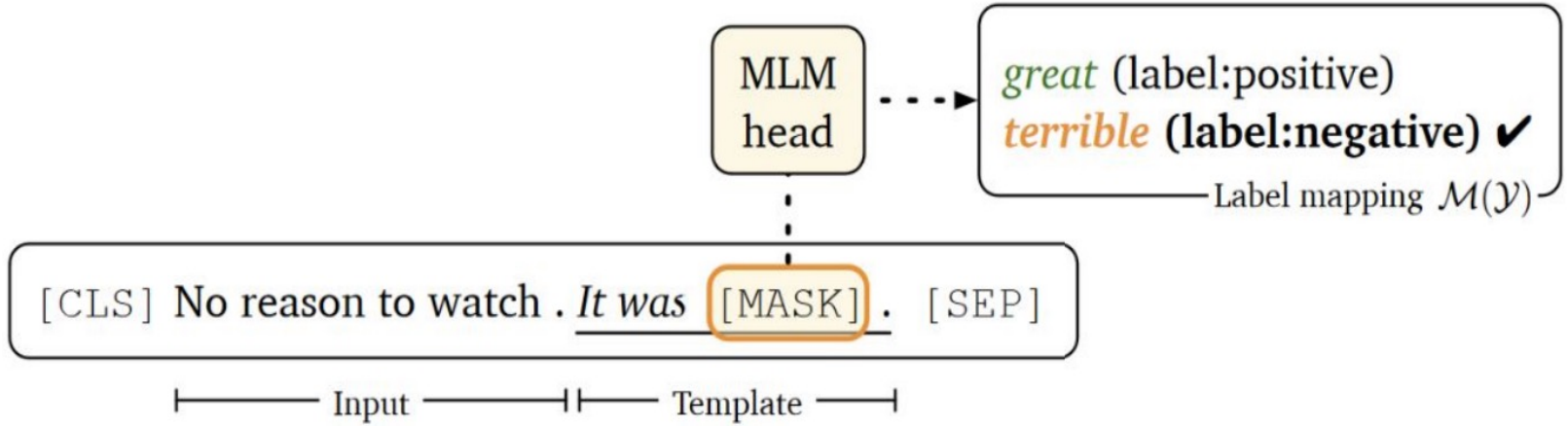
Few-shot Learning: Prompt-based Fine-tuning

- Step 1: Formulate the task into a masked token prediction through a **prompt template**

$$p(y \mid x_{\text{in}}) = p\left(\texttt{[MASK]} = \mathcal{M}(y) \mid x_{\text{prompt}}\right)$$
$$= \frac{\exp\left(\mathbf{w}_{\mathcal{M}(y)} \cdot \mathbf{h}_{\texttt{[MASK]}}\right)}{\sum_{y' \in \mathcal{Y}} \exp\left(\mathbf{w}_{\mathcal{M}(y')} \cdot \mathbf{h}_{\texttt{[MASK]}}\right)},$$

- Step 2: Choose a **label-word mapping** M.

- Step 3: Fine-tune the LM to fill in the correct word



re-uses the pre-trained weights and does not introduce any new parameters and thus reduces the gap between pretraining and fine-tuning

Gao et al., Making Pre-trained Language Models Better Few-shot Learners, ACL 2021

Evaluation Datasets

Category	Dataset	$\lvert\mathcal{Y}\rvert$	L	#Train	#Test	Type	Labels (classification tasks)
single-sentence	SST-2	2	19	6,920	872	sentiment	positive, negative
	SST-5	5	18	8,544	2,210	sentiment	v. pos., positive, neutral, negative, v. neg.
	MR	2	20	8,662	2,000	sentiment	positive, negative
	CR	2	19	1,775	2,000	sentiment	positive, negative
	MPQA	2	3	8,606	2,000	opinion polarity	positive, negative
	Subj	2	23	8,000	2,000	subjectivity	subjective, objective
	TREC	6	10	5,452	500	question cls.	abbr., entity, description, human, loc., num.
	CoLA	2	8	8,551	1,042	acceptability	grammatical, not_grammatical
sentence-pair	MNLI	3	22/11	392,702	9,815	NLI	entailment, neutral, contradiction
	SNLI	3	14/8	549,367	9,842	NLI	entailment, neutral, contradiction
	QNLI	2	11/30	104,743	5,463	NLI	entailment, not_entailment
	RTE	2	49/10	2,490	277	NLI	entailment, not_entailment
	MRPC	2	22/21	3,668	408	paraphrase	equivalent, not_equivalent
	QQP	2	12/12	363,846	40,431	paraphrase	equivalent, not_equivalent
	STS-B	$\mathcal{R}$	11/11	5,749	1,500	sent. similarity	-

Gao et al., Making Pre-trained Language Models Better Few-shot Learners, ACL 2021

Results

	SST-2 (acc)	SST-5 (acc)	MR (acc)	CR (acc)	MPQA (acc)	Subj (acc)	TREC (acc)	CoLA (Matt.)
Majority†	*50.9*	*23.1*	*50.0*	*50.0*	*50.0*	*50.0*	*18.8*	*0.0*
Prompt-based zero-shot‡	83.6	35.0	80.8	79.5	67.6	51.4	32.0	2.0
"GPT-3" in-context learning	84.8 (1.3)	30.6 (0.9)	80.5 (1.7)	87.4 (0.8)	63.8 (2.1)	53.6 (1.0)	26.2 (2.4)	-1.5 (2.4)
Fine-tuning	81.4 (3.8)	43.9 (2.0)	76.9 (5.9)	75.8 (3.2)	72.0 (3.8)	90.8 (1.8)	88.8 (2.1)	**33.9** (14.3)
Prompt-based FT (man)	92.7 (0.9)	47.4 (2.5)	87.0 (1.2)	90.3 (1.0)	84.7 (2.2)	91.2 (1.1)	84.8 (5.1)	9.3 (7.3)
+ demonstrations	92.6 (0.5)	**50.6** (1.4)	86.6 (2.2)	90.2 (1.2)	**87.0** (1.1)	**92.3** (0.8)	87.5 (3.2)	18.7 (8.8)
Prompt-based FT (auto)	92.3 (1.0)	49.2 (1.6)	85.5 (2.8)	89.0 (1.4)	85.8 (1.9)	91.2 (1.1)	88.2 (2.0)	14.0 (14.1)
+ demonstrations	**93.0** (0.6)	49.5 (1.7)	**87.7** (1.4)	**91.0** (0.9)	86.5 (2.6)	91.4 (1.8)	**89.4** (1.7)	21.8 (15.9)
Fine-tuning (full)†	*95.0*	*58.7*	*90.8*	*89.4*	*87.8*	*97.0*	*97.4*	*62.6*

	MNLI (acc)	MNLI-mm (acc)	SNLI (acc)	QNLI (acc)	RTE (acc)	MRPC (F1)	QQP (F1)	STS-B (Pear.)
Majority†	*32.7*	*33.0*	*33.8*	*49.5*	*52.7*	*81.2*	*0.0*	-
Prompt-based zero-shot‡	50.8	51.7	49.5	50.8	51.3	61.9	49.7	-3.2
"GPT-3" in-context learning	52.0 (0.7)	53.4 (0.6)	47.1 (0.6)	53.8 (0.4)	60.4 (1.4)	45.7 (6.0)	36.1 (5.2)	14.3 (2.8)
Fine-tuning	45.8 (6.4)	47.8 (6.8)	48.4 (4.8)	60.2 (6.5)	54.4 (3.9)	76.6 (2.5)	60.7 (4.3)	53.5 (8.5)
Prompt-based FT (man)	68.3 (2.3)	70.5 (1.9)	77.2 (3.7)	64.5 (4.2)	69.1 (3.6)	74.5 (5.3)	65.5 (5.3)	71.0 (7.0)
+ demonstrations	**70.7** (1.3)	**72.0** (1.2)	**79.7** (1.5)	**69.2** (1.9)	68.7 (2.3)	77.8 (2.0)	**69.8** (1.8)	73.5 (5.1)
Prompt-based FT (auto)	68.3 (2.5)	70.1 (2.6)	77.1 (2.1)	68.3 (7.4)	**73.9** (2.2)	76.2 (2.3)	67.0 (3.0)	75.0 (3.3)
+ demonstrations	70.0 (3.6)	**72.0** (3.1)	77.5 (3.5)	68.5 (5.4)	71.1 (5.3)	**78.1** (3.4)	67.7 (5.8)	**76.4** (6.2)
Fine-tuning (full)†	*89.8*	*89.5*	*92.6*	*93.3*	*80.9*	*91.4*	*81.7*	*91.9*

Table 3: Our main results using RoBERTa-large. †: full training set is used (see dataset sizes in Table B.1); ‡: no training examples are used; otherwise we use $K = 16$ (per class) for few-shot experiments. We report mean (and standard deviation) performance over 5 different splits (§3). Majority: majority class; FT: fine-tuning; man: manual prompt (Table 1); auto: automatically searched templates (§5.2); "GPT-3" in-context learning: using the in-context learning proposed in Brown et al. (2020) with RoBERTa-large (no parameter updates).

Gao et al., Making Pre-trained Language Models Better Few-shot Learners, ACL 2021

Prompt Design

- Manual designing requires some effort.
- The template T and word-class mapping M are not independent.
- Model selection (T, M) is subject to overfitting.
- Automatic prompt generation solution is also proposed in this paper

$$\underset{v \in \mathcal{V}}{\text{Top-}k} \left\{ \sum_{x_{\text{in}} \in \mathcal{D}^c_{\text{train}}} \log P_{\mathcal{L}}\Big(\texttt{[MASK]} = v \mid \mathcal{T}(x_{\text{in}}) \Big) \right\}$$

SNLI (entailment/neutral/contradiction)		mean (std)
$<S_1>$? [MASK] , $<S_2>$	Yes/Maybe/No	**77.2 (3.7)**
$<S_1>$. [MASK] , $<S_2>$	Yes/Maybe/No	76.2 (3.3)
$<S_1>$? [MASK] $<S_2>$	Yes/Maybe/No	74.9 (3.0)
$<S_1>$ $<S_2>$ [MASK]	Yes/Maybe/No	65.8 (2.4)
$<S_2>$? [MASK] , $<S_1>$	Yes/Maybe/No	62.9 (4.1)
$<S_1>$? [MASK] , $<S_2>$	Maybe/No/Yes	60.6 (4.8)
Fine-tuning	-	48.4 (4.8)

Gao et al., Making Pre-trained Language Models Better Few-shot Learners, ACL 2021

Outline

- In-context learning
- **Parameter Efficient Fine-Tuning (PEFT)**
  - Adapter
  - Prompt tuning & prefix tuning
  - LoRA

Parameter-Efficient Finetuning

- Lightweight finetuning methods adapt pretrained models in a constrained way.
  - Leads to less overfitting and/or more efficient finetuning and inference.

- Large enough training set makes full fine tuning (on all weights) really good.
  - But this needs enormous separate models to be stored.
    - For each task
    - For each user

Parameter Efficient Fine-Tuning

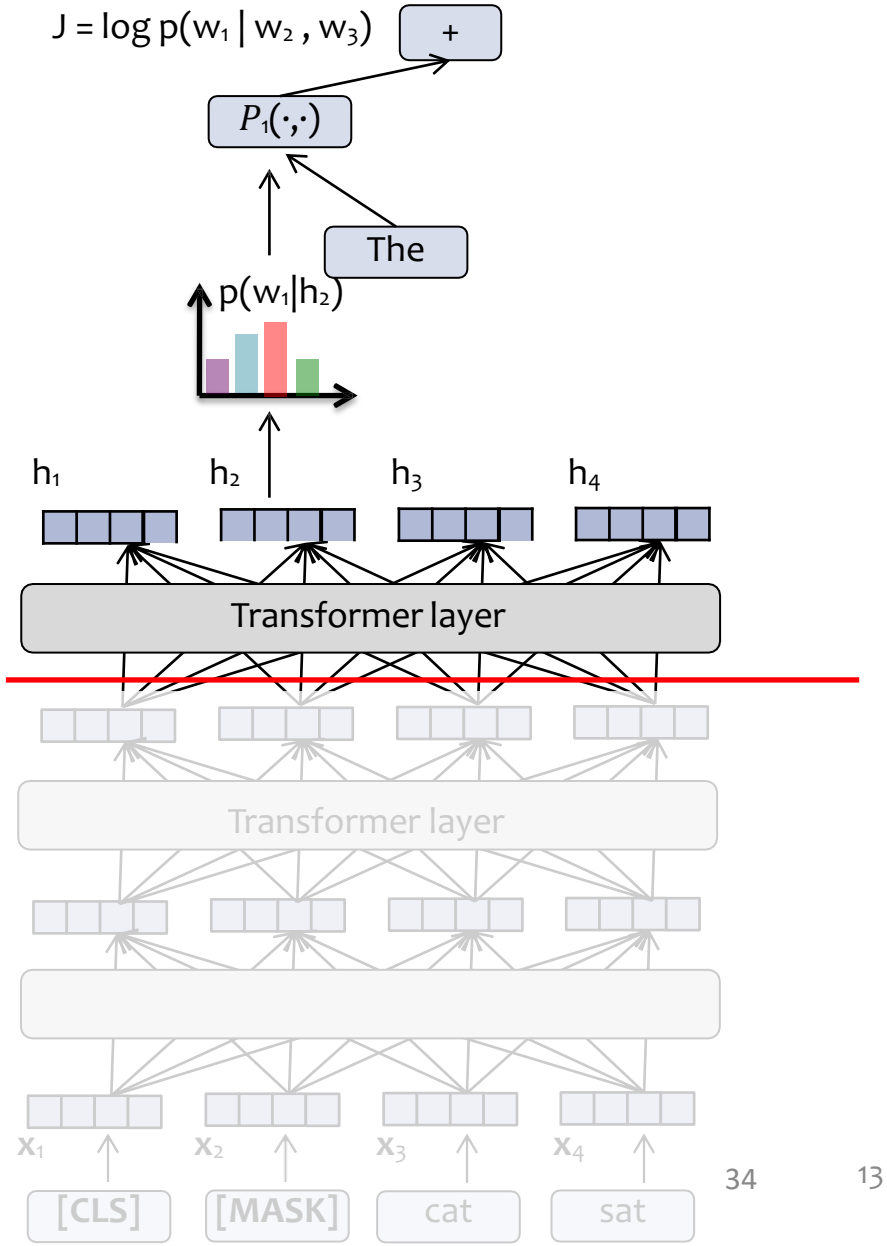
- **Goal**: perform fine-tuning of fewer parameters, but achieve performance on a downstream task that is comparable to finetuning of all parameters

- **Various approaches**:
  - **Subset**: Pick a subset of the parameters and fine-tune only those (e.g. only the top K layers of a K+L layer deep neural network)
  - **Adapters**: add additional layers that have few parameters and tune only the parameters of those layers, keeping all others fixed
  - **Prefix Tuning**: for a Transformer LM, pretend as if there exist many tokens that came before your sequence and tune the keys/values corresponding to those tokens
  - **LoRA**: learn a small delta for the each of the parameter matrices with the delta chosen to be low rank

Fine-Tuning the Top Layers Only

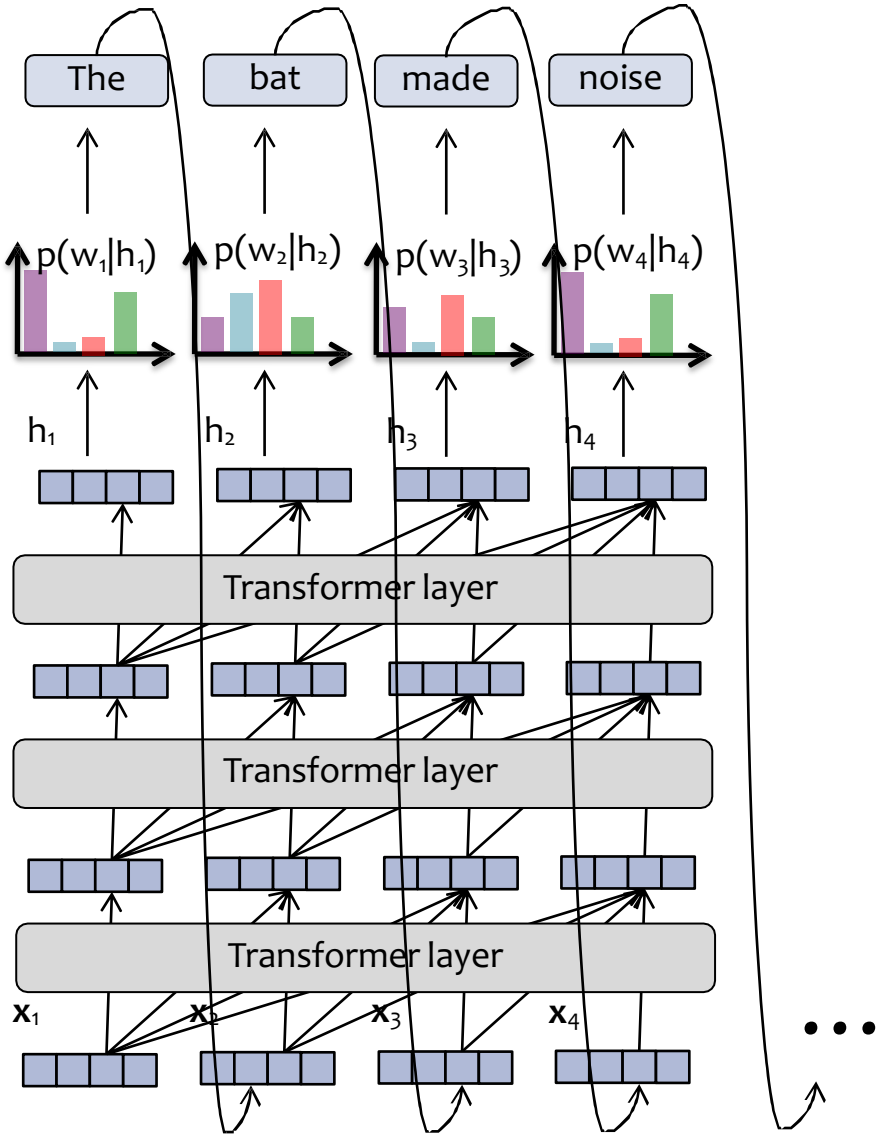
- Simple baseline for PEFT:
  - keep all parameters fixed except for the top K layers
  - gradients only need to flow through K layers instead of K+L total layers
  - reduced memory usage b/c we don't need to store the adjoints (gradient of the loss with respect to each parameter) for the full computation graph
- Can easily be applied to most deep neural networks

stop gradient here s.t. error does not backprop to lower layers



ADAPTERS

Transformer Language Model



Each layer of a Transformer LM consists of several **sublayers**:

1. attention
2. feed-forward neural network
3. layer normalization
4. residual connections

Each hidden vector looks back at the hidden vectors of the **current and previous timesteps in the previous layer.**

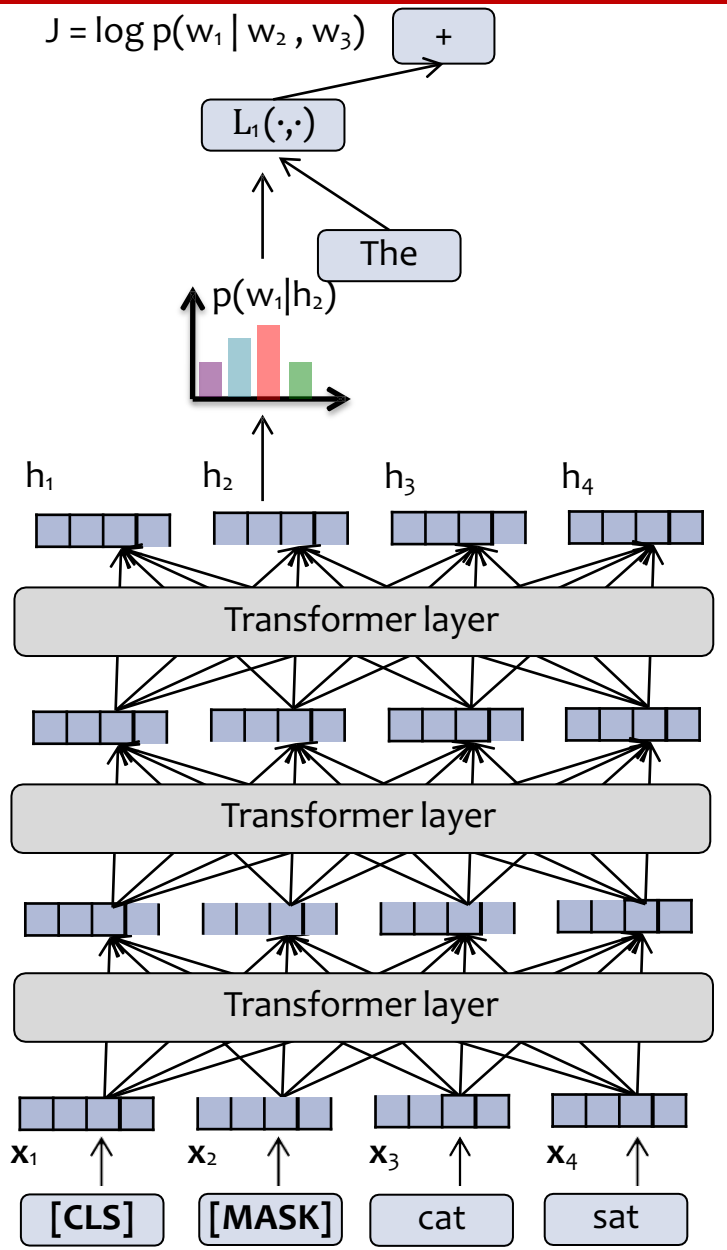
The language model part is just like an RNN-LM.

Encoder-only Transformer

BERT popularized this encoder-only Transformer architecture and style of pretraining

MLM Pretraining:

- Rather than trying to predict the next word from the previous ones…
- …mask out a word (or a few words) and predict the missing words from the remaining ones



• • •

Each layer of an encoder-only Transformer consists of several **sublayers**:

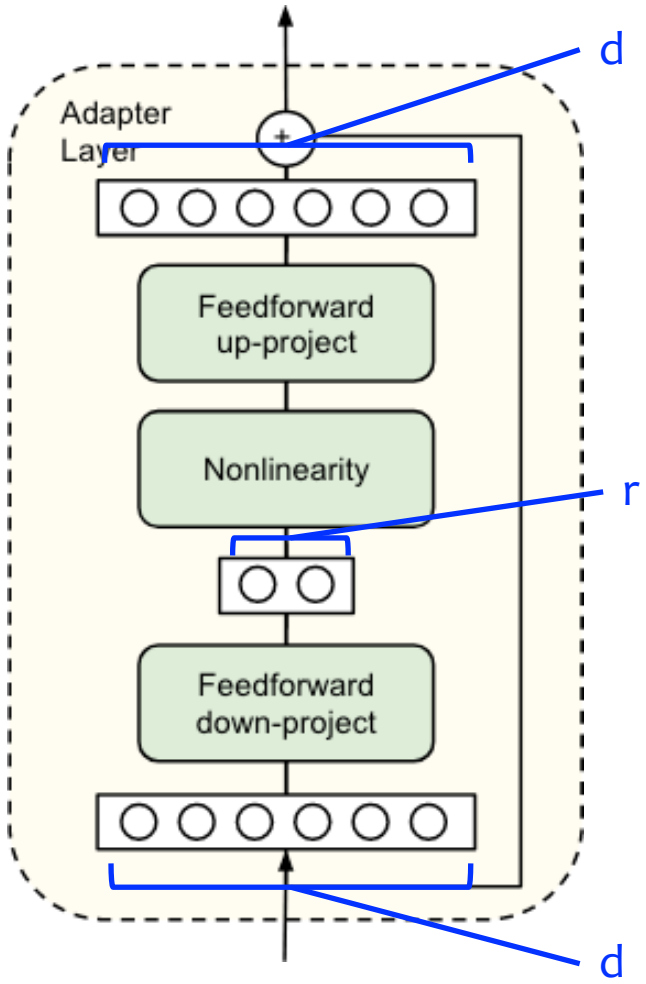
1. non-causal attention
2. feed-forward neural network
3. layer normalization
4. residual connections

Each hidden vector looks at the the hidden vectors of **all timesteps in the previous layer.**

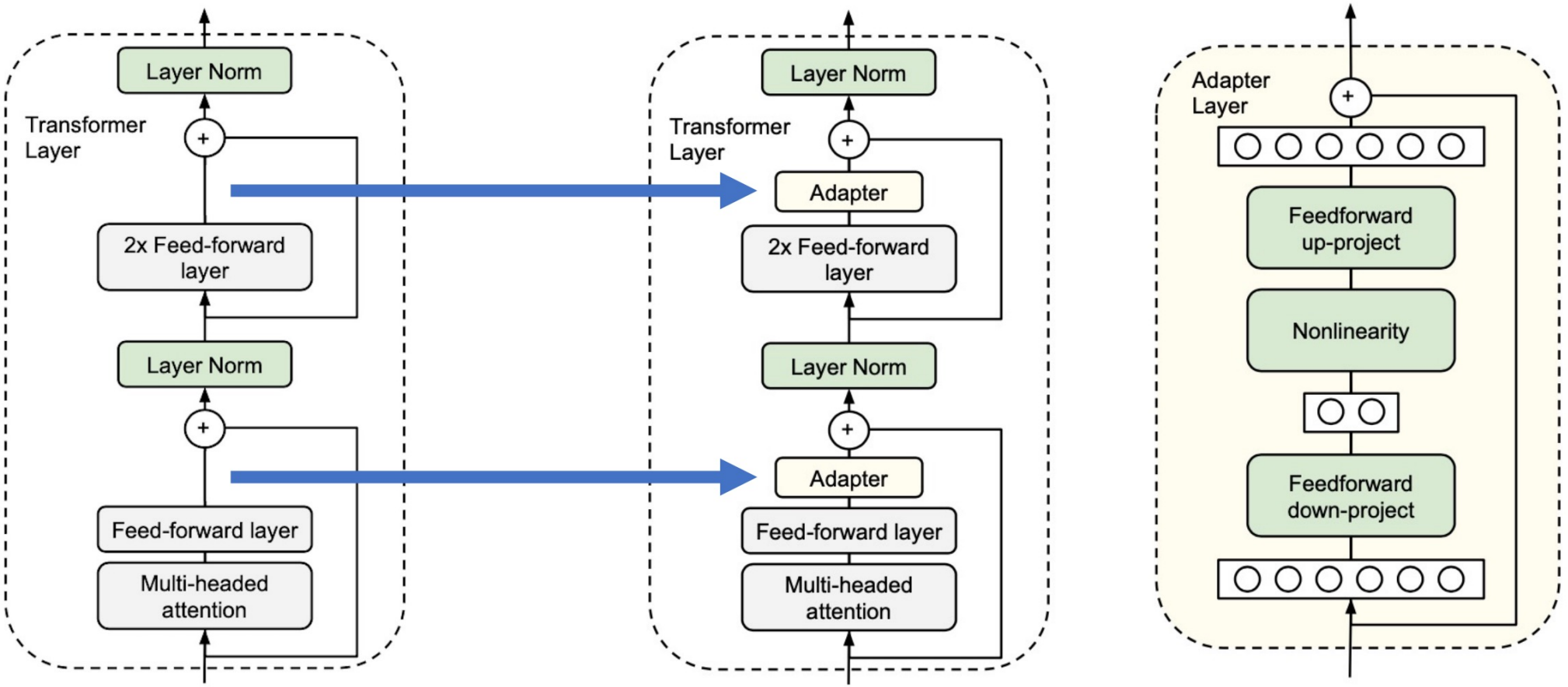
The distribution over words is used for **masked language model (MLM) pre-training** (cf. BERT)

Adapters Module

- An adapter layer is simply an **FF with one hidden layer**, and a residual connection
- **Bottleneck** architecture: For input dimension, d, the adapter layer also has output dimension d, but bottlenecks to a lower dimension r in the middle
- Near identity initialization of the adapter. Why? How?
  - The adapter has a skip connection itself and near zero initialization of weights



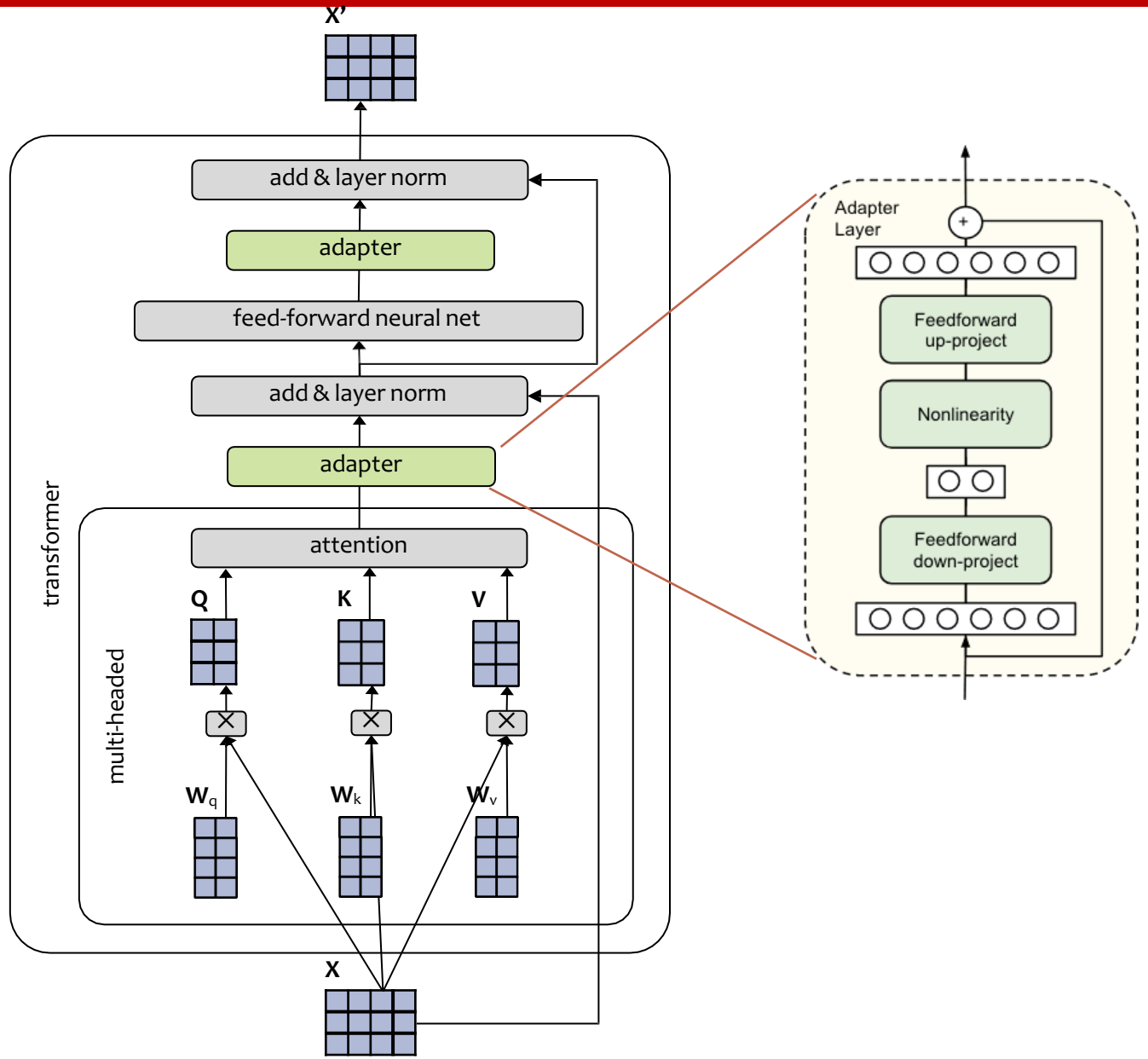
Adapters Module



Inserted in both sublayers; right before the skip connection.
New layer normalization parameters per task as well.

adapter-tuning inserts task-specific layers (adapters) between each layer of the pretrained LM

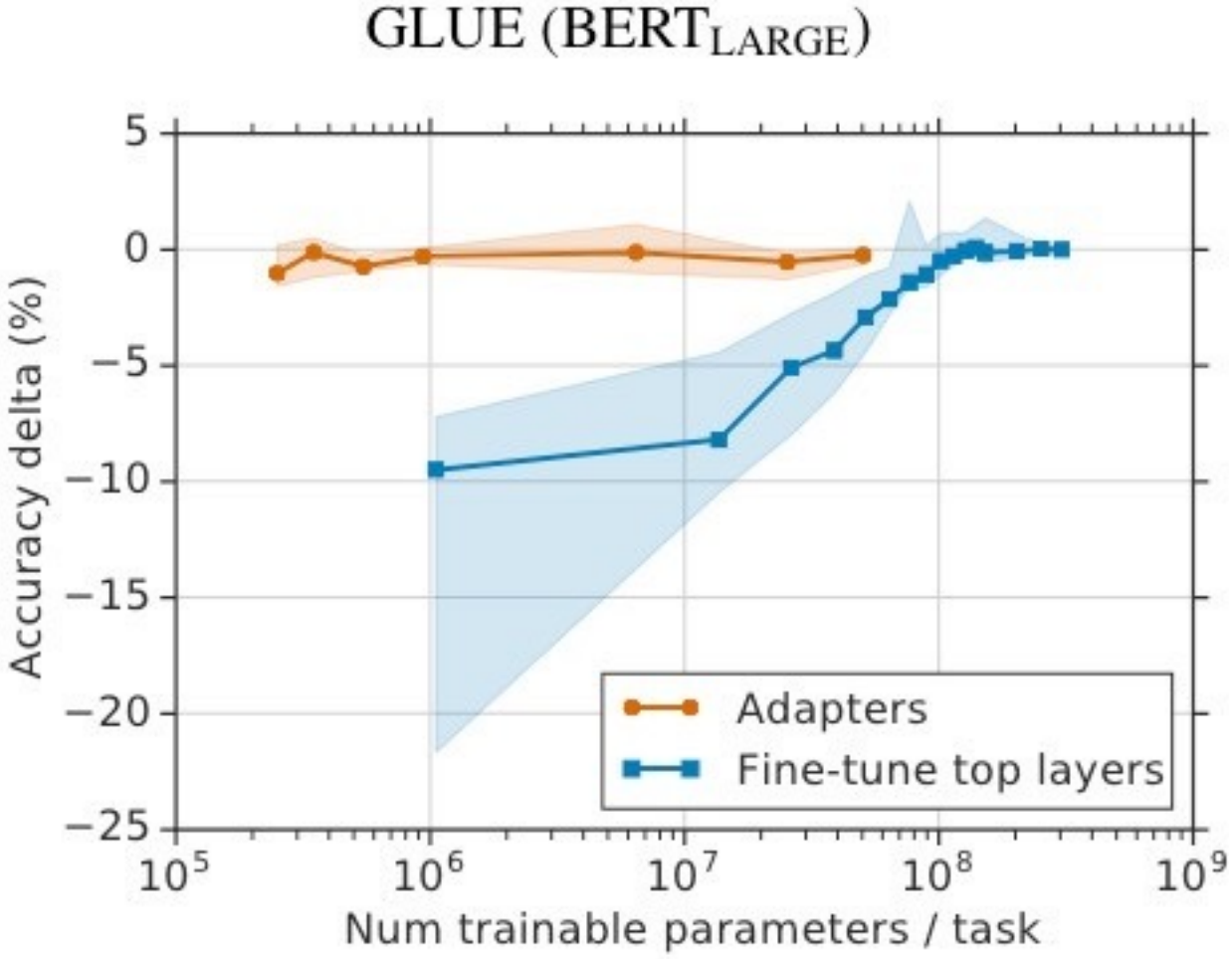
Adapters for Transformer



- They add adapter layers in between the transformer layers of a large model.
- In practice, the adapter layers contain only 0.5% – 8% of the total parameters
- All the other parameters of the pretrained model are kept fixed
- No need to store a full model for each task and/ user, only the adapter params.

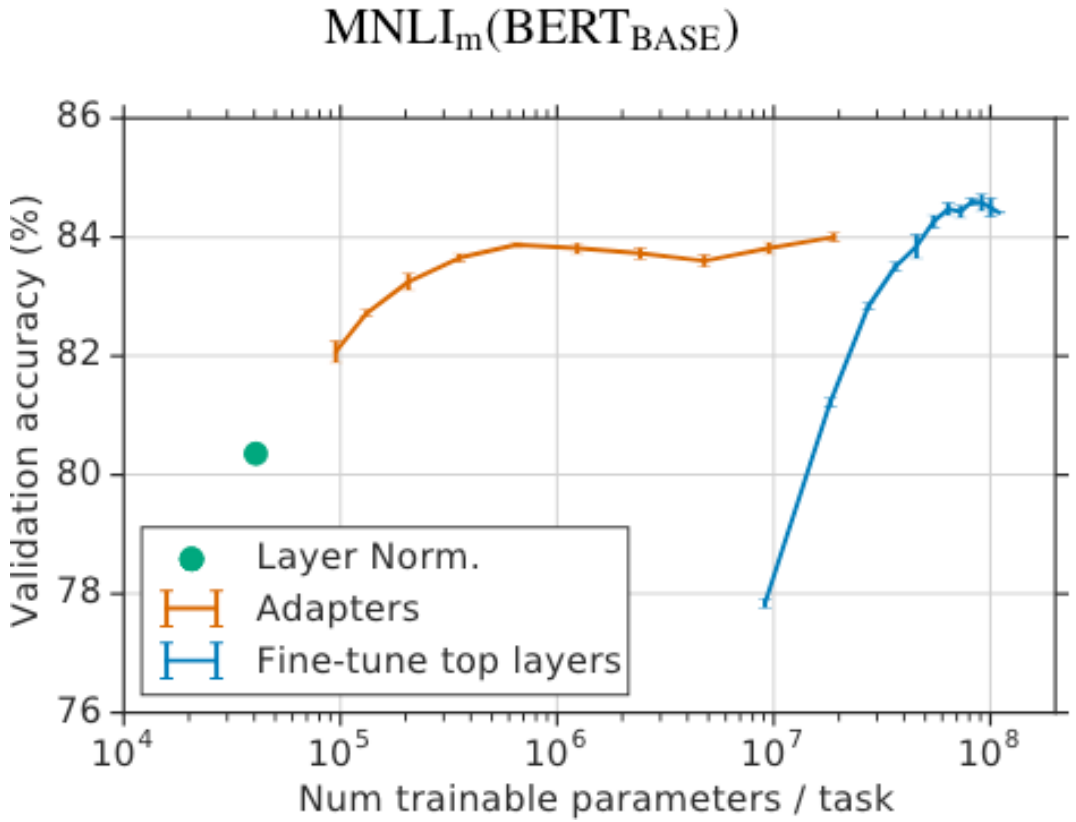
Adapter Results

- **Pretrained Model**: BERT-Large
- **Baseline Method**: finetune only the top K layers of BERT-Large
- Adapters achieve nearly the performance (i.e. 0% delta) of fine-tuning but with substantially fewer parameters
- Sometimes adapters even outperform full fine-tuning



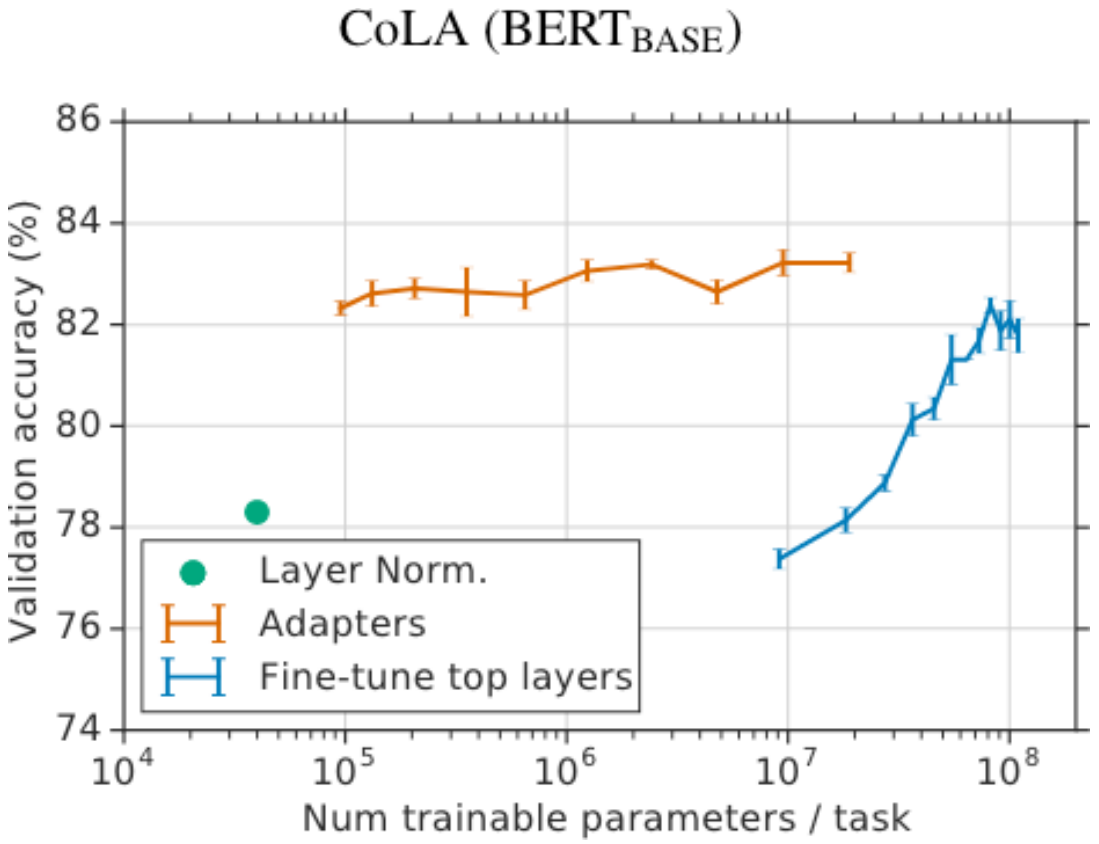
Adapter Results

- **Pretrained Model**: BERT-Large
- **Baseline Method**: finetune only the top K layers of BERT-Large
- Adapters achieve nearly the performance (i.e. 0% delta) of fine-tuning but with substantially fewer parameters
- Sometimes adapters even outperform full fine-tuning



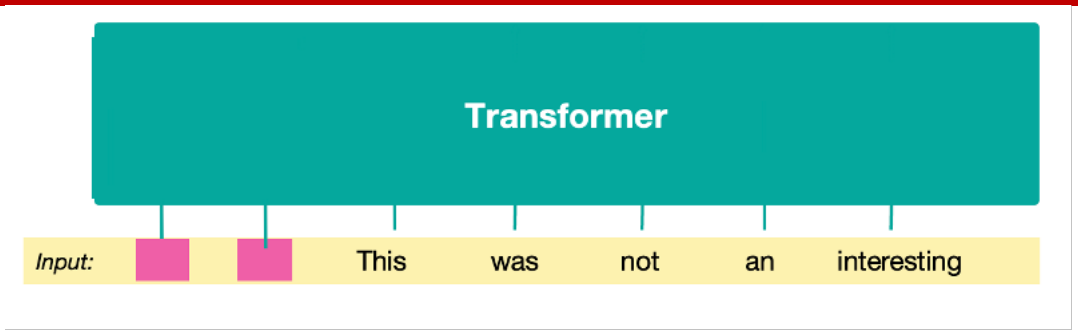
Adapter Results

- **Pretrained Model**: BERT-Large
- **Baseline Method**: finetune only the top K layers of BERT-Large
- Adapters achieve nearly the performance (i.e. 0% delta) of fine-tuning but with substantially fewer parameters
- Sometimes adapters even outperform full fine-tuning

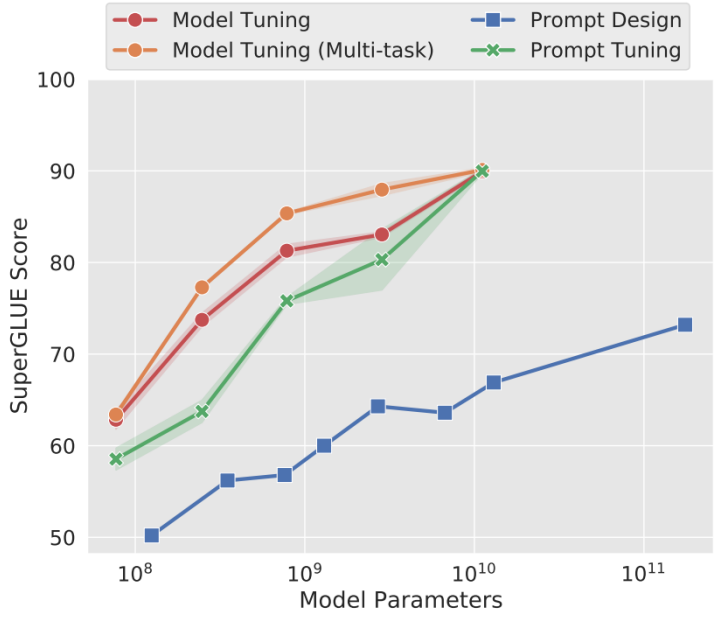


PROMPT TUNING & PREFIX TUNING

Soft prompt tuning



- For prompt design, in (Brown et al. 2020), the discrete prompt is optimized manually.
- Optimization in discrete space is hard! (Gao et al., 2021)
- Rather than designing a prompt manually, we can learn an optimal prefix for each task.
- Only ~0.1% of parameters need to be tuned! (adapter is 3.6%)

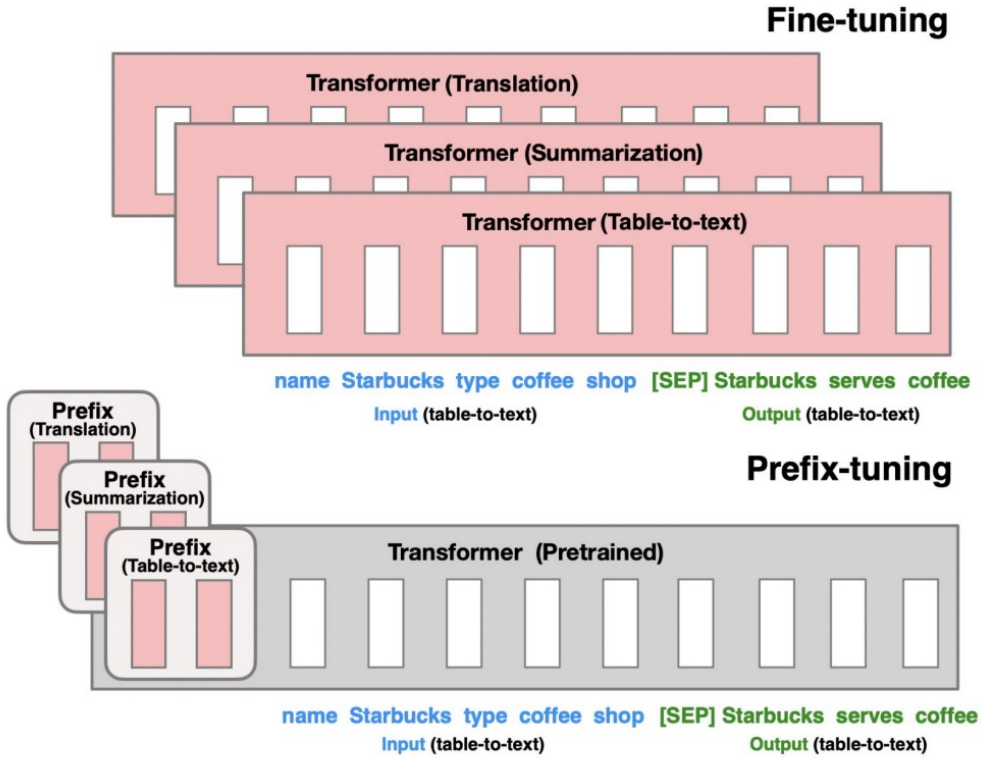


Lester et al., The Power of Scale for Parameter-Efficient Prompt Tuning, EMNLP 2021
Lisa Li and Liang, Prefix-Tuning: Optimizing Continuous Prompts for Generation, 2021

Soft prompt tuning

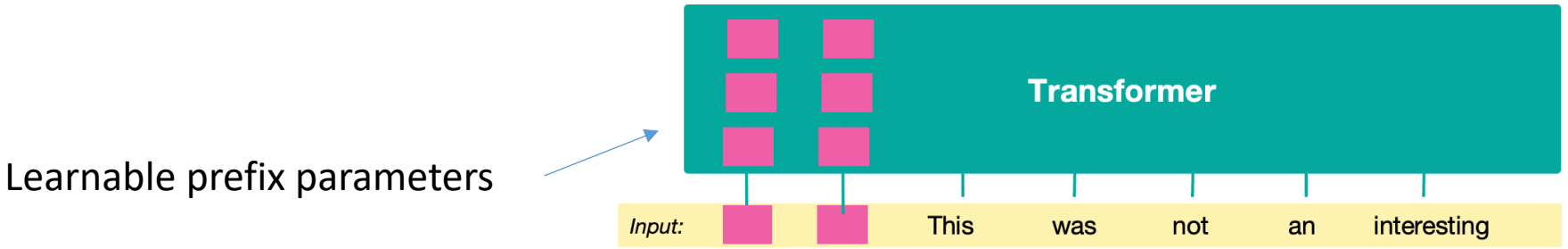
- **Soft prompt tuning**:
  - Freeze all pretrained parameters
  - Learnable prompt is added to the input
  - The soft prompt is processed by the model like real words would be.

- **Advantage**: each element of a batch at inference could run a different tuned model (multi-task setting)



Lisa Li and Liang, Prefix-Tuning: Optimizing Continuous Prompts for Generation, 2021

Prefix tuning



- It's not just the prompts in the prefix that get tuned, it is also the **hidden representations** of later layer.
- **Prefix tuning**:
  - Freeze all pretrained parameters
  - Prepend certain trainable prefix tokens to the input and hidden activations.
  - The prefix is processed by the model just like real words would be.

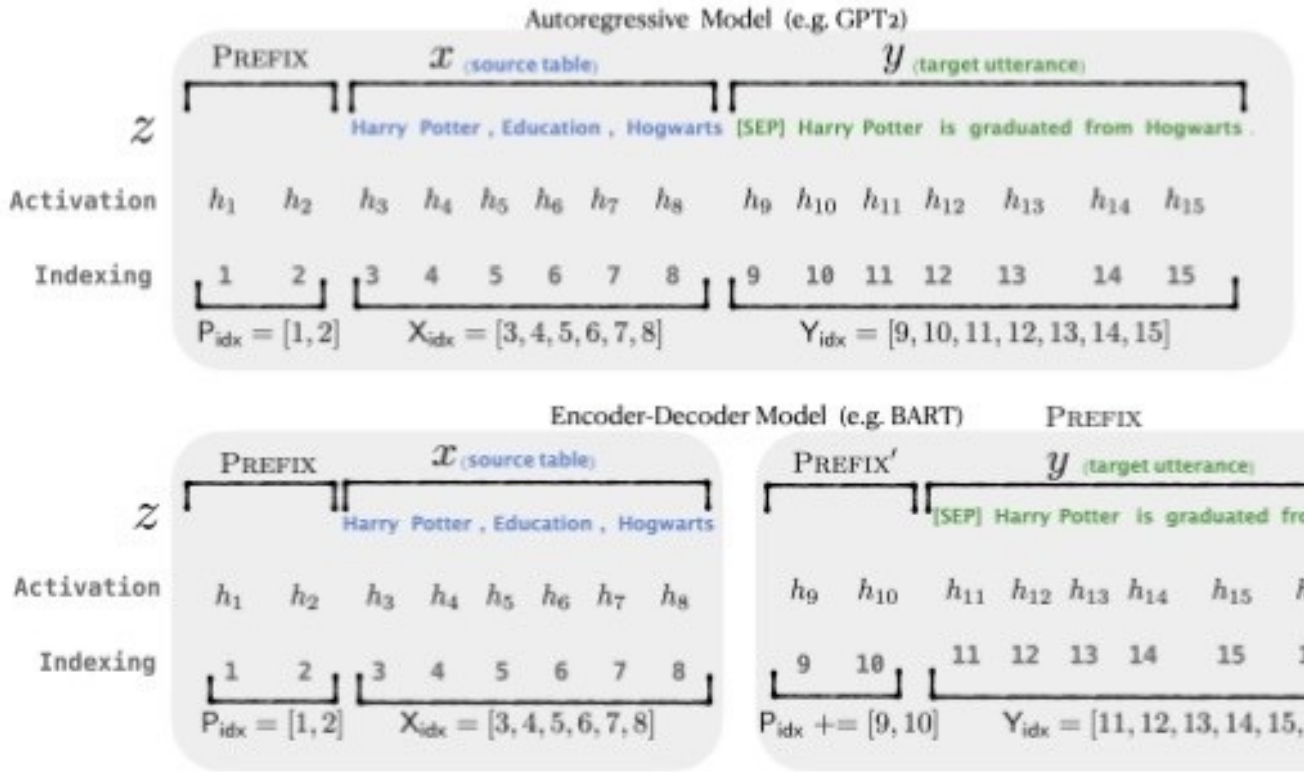
$$h_i = \begin{cases} P_\theta[i,:], & \text{if } i \in \mathsf{P}_{\text{idx}}, \\ \mathrm{LM}_\phi(z_i, h_{<i}), & \text{otherwise.} \end{cases}$$

$$P_\theta[i,:] = MLP_\theta(P'[i,:])$$

Lisa Li and Liang, Prefix-Tuning: Optimizing Continuous Prompts for Generation, 2021

Prefix tuning

- Encoder-decoder models get two trainable prefixes, one for encoder and one for decoder



Summarization Example

Article: Scientists at University College London discovered people tend to think that their hands are wider and their fingers are shorter than they truly are.They say the confusion may lie in the way the brain receives information from different parts of the body.Distorted perception may dominate in some people, leading to body image problems ... [ignoring 308 words] could be very motivating for people with eating disorders to know that there was a biological explanation for their experiences, rather than feeling it was their fault."

Summary: The brain naturally distorts body image - a finding which could explain eating disorders like anorexia, say experts.

Table-to-text Example

Table: name[Clowns] customer-rating[1 out of 5] eatType[coffee shop] food[Chinese] area[riverside] near[Clare Hall]

Textual Description: Clowns is a coffee shop in the riverside area near Clare Hall that has a rating 1 out of 5 . They serve Chinese food .

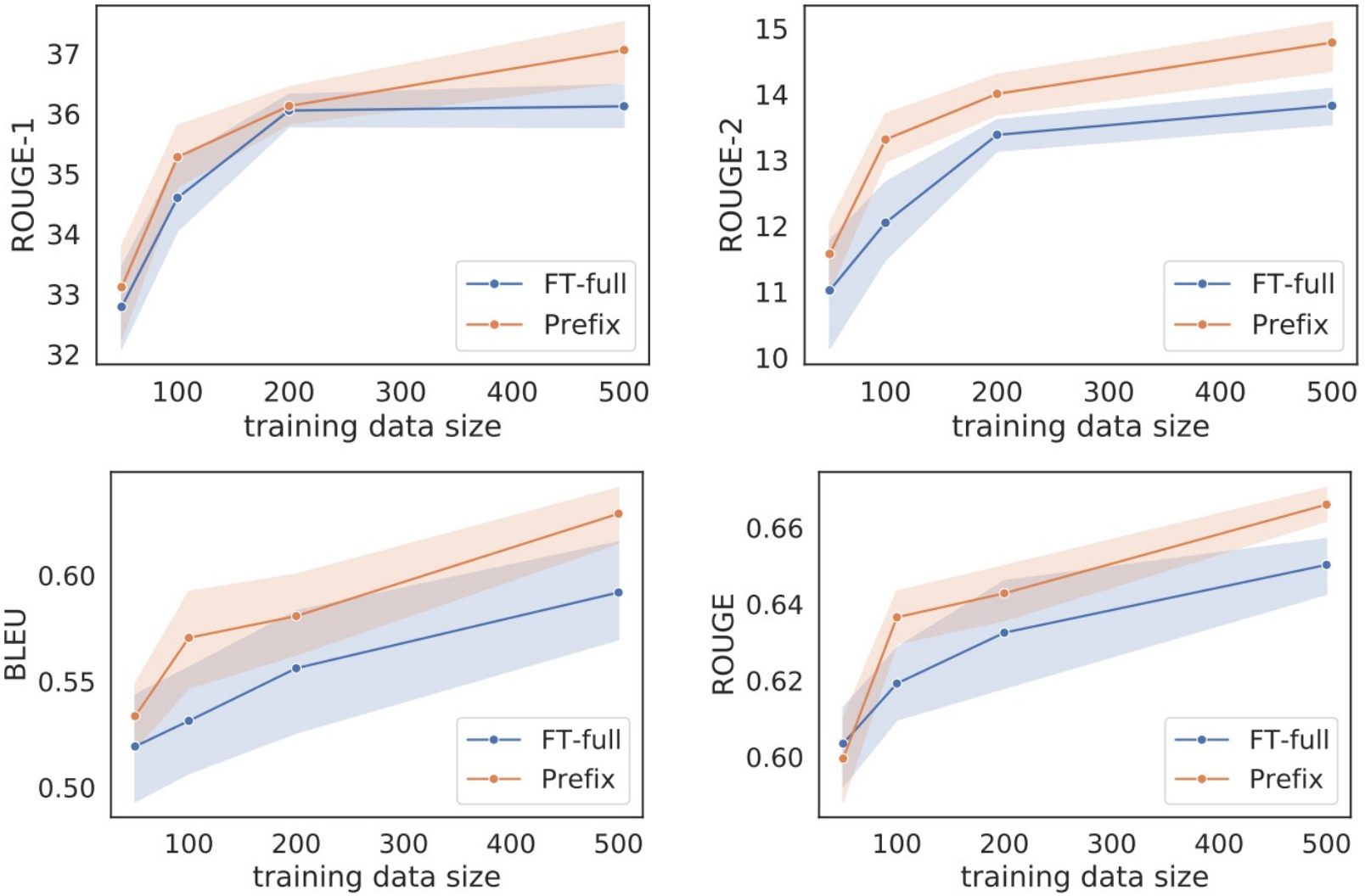
Lisa Li and Liang, Prefix-Tuning: Optimizing Continuous Prompts for Generation, 2021

Prefix tuning results

	E2E					WebNLG									DART					
	BLEU	NIST	MET	R-L	CIDEr	BLEU			MET			TER ↓			BLEU	MET	TER ↓	Mover	BERT	BLEURT
						S	U	A	S	U	A	S	U	A						
GPT-2$_{\text{MEDIUM}}$																				
FT-FULL	68.8	8.71	46.1	71.1	2.43	**64.7**	26.7	45.7	**0.46**	0.30	0.38	**0.33**	0.78	0.54	46.2	**0.39**	**0.46**	**0.50**	**0.94**	**0.39**
FT-TOP2	68.1	8.59	46.0	70.8	2.41	53.6	18.9	36.0	0.38	0.23	0.31	0.49	0.99	0.72	41.0	0.34	0.56	0.43	0.93	0.21
ADAPTER(3%)	68.9	8.71	46.1	71.3	**2.47**	60.5	**47.9**	54.8	0.43	**0.38**	**0.41**	0.35	**0.46**	**0.39**	45.2	0.38	**0.46**	**0.50**	**0.94**	**0.39**
ADAPTER(0.1%)	66.3	8.41	45.0	69.8	2.40	54.5	45.1	50.2	0.39	0.36	0.38	0.40	0.46	0.43	42.4	0.36	0.48	0.47	**0.94**	0.33
PREFIX(0.1%)	**70.3**	**8.82**	**46.3**	**72.1**	2.46	62.9	45.3	**55.0**	0.44	0.37	**0.41**	0.35	0.51	0.42	**46.4**	0.38	**0.46**	**0.50**	**0.94**	**0.39**
GPT-2$_{\text{LARGE}}$																				
FT-FULL	68.5	8.78	46.0	69.9	2.45	**65.3**	43.1	55.5	**0.46**	0.38	**0.42**	**0.33**	0.53	0.42	**47.0**	**0.39**	0.46	**0.51**	**0.94**	**0.40**
Prefix	**70.3**	**8.85**	**46.2**	**71.7**	**2.47**	63.4	**47.7**	**56.3**	0.45	**0.39**	**0.42**	0.34	**0.48**	**0.40**	46.7	**0.39**	**0.45**	**0.51**	**0.94**	**0.40**
SOTA	68.6	8.70	45.3	70.8	2.37	63.9	52.8	57.1	0.46	0.41	0.44	-	-	-	-	-	-	-	-	-

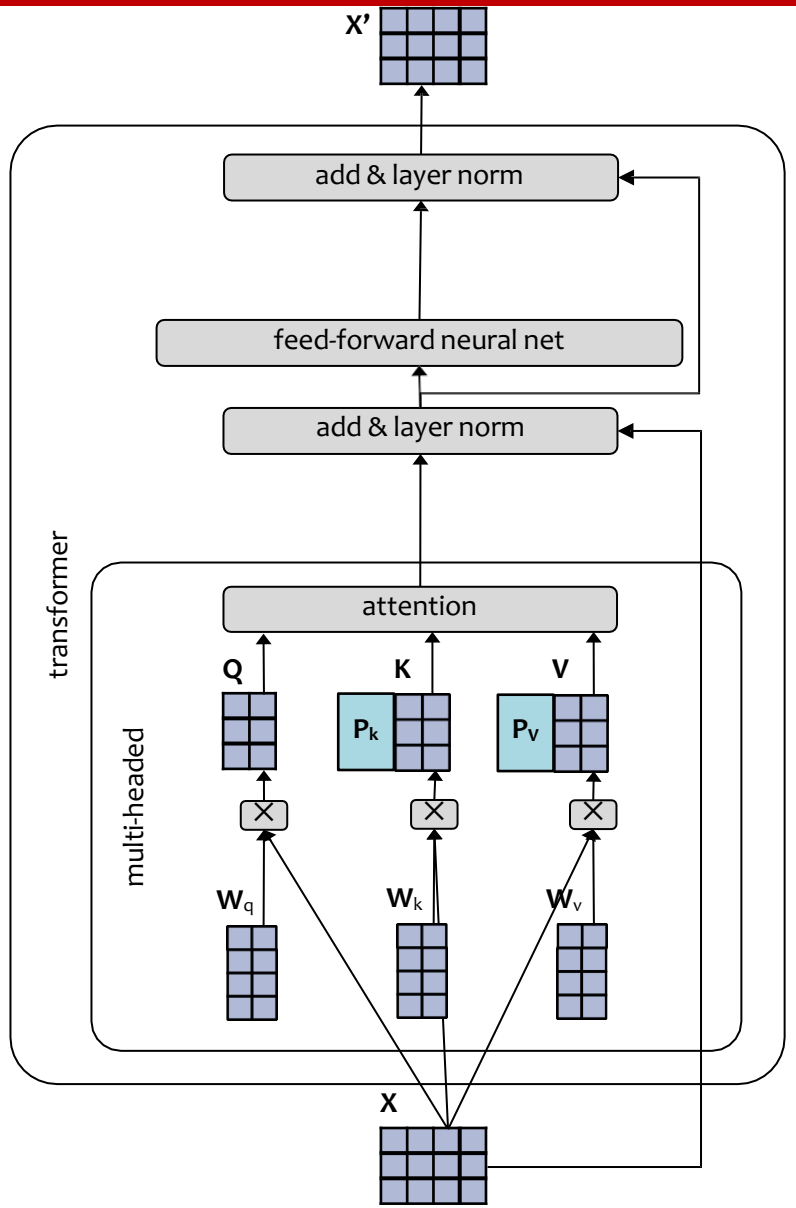
Lisa Li and Liang, Prefix-Tuning: Optimizing Continuous Prompts for Generation, 2021

Low-data regimes



Lisa Li and Liang, Prefix-Tuning: Optimizing Continuous Prompts for Generation, 2021

Prefix tuning



Lisa Li and Liang, Prefix-Tuning: Optimizing Continuous Prompts for Generation, 2021

LOW-RANK ADAPTATION (LORA)

How large are LLMs?

Comparison of some recent **large language models** (LLMs)

Model	Creators	Year of release	Training Data (# tokens)	Model Size (# parameters)
GPT-2	OpenAI	2019	~10 billion (40Gb)	1.5 billion
GPT-3 (cf. ChatGPT)	OpenAI	2020	300 billion	175 billion
PaLM	Google	2022	780 billion	540 billion
Chinchilla	DeepMind	2022	1.4 trillion	70 billion
LaMDA (cf. Bard)	Google	2022	1.56 trillion	137 billion
LLaMA	Meta	2023	1.4 trillion	65 billion
LLaMA-2	Meta	2023	2 trillion	70 billion
LLaMA-3.3	Meta	2024	15+ trillion	70 billion

Why does efficiency matter?

Case Study: GPT-3

- # of training tokens = 500 billion
- # of parameters = 175 billion
- # of cycles = 50 petaflop/s-days (each of which are 8.64e+19 flops)

Dataset	Quantity (tokens)	Weight in training mix	Epochs elapsed when training for 300B tokens
Common Crawl (filtered)	410 billion	60%	0.44
WebText2	19 billion	22%	2.9
Books1	12 billion	8%	1.9
Books2	55 billion	8%	0.43
Wikipedia	3 billion	3%	3.4

Table 2.2: Datasets used to train GPT-3. "Weight in training mix" refers to the fraction of examples during training that are drawn from a given dataset, which we intentionally do not make proportional to the size of the dataset. As a result, when we train for 300 billion tokens, some datasets are seen up to 3.4 times during training while other datasets are seen less than once.

Model Name	n_{params}	n_{layers}	d_{model}	n_{heads}	d_{head}	Batch Size	Learning Rate
GPT-3 Small	125M	12	768	12	64	0.5M	6.0×10^{-4}
GPT-3 Medium	350M	24	1024	16	64	0.5M	3.0×10^{-4}
GPT-3 Large	760M	24	1536	16	96	0.5M	2.5×10^{-4}
GPT-3 XL	1.3B	24	2048	24	128	1M	2.0×10^{-4}
GPT-3 2.7B	2.7B	32	2560	32	80	1M	1.6×10^{-4}
GPT-3 6.7B	6.7B	32	4096	32	128	2M	1.2×10^{-4}
GPT-3 13B	13.0B	40	5140	40	128	2M	1.0×10^{-4}
GPT-3 175B or "GPT-3"	175.0B	96	12288	96	128	3.2M	0.6×10^{-4}

Table 2.1: Sizes, architectures, and learning hyper-parameters (batch size in tokens and learning rate) of the models which we trained. All models were trained for a total of 300 billion tokens.

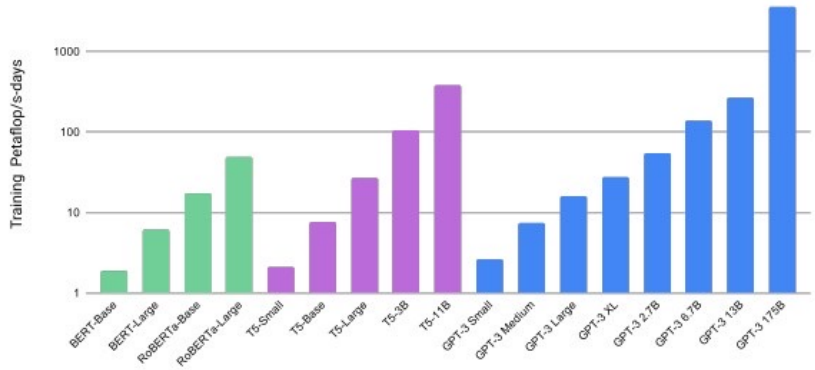


Figure 2.2: Total compute used during training. Based on the analysis in Scaling Laws For Neural Language Models [KMH+20] we train much larger models on many fewer tokens than is typical. As a consequence, although GPT-3 3B is almost 10x larger than RoBERTa-Large (355M params), both models took roughly 50 petaflop/s-days of compute during pre-training. Methodology for these calculations can be found in Appendix D.

Brown et al., Language Models are Few-Shot Learners, 2020

Fine-Tuning LLMs without Regularization

Method	MNLI-m (Val. Acc./%)	RTE (Val. Acc./%)
GPT-3 Few-Shot	40.6	69.0
GPT-3 Fine-Tuned	89.5	85.4

Question:

Why don't LLMs overfit when we fine-tune them without regularization?

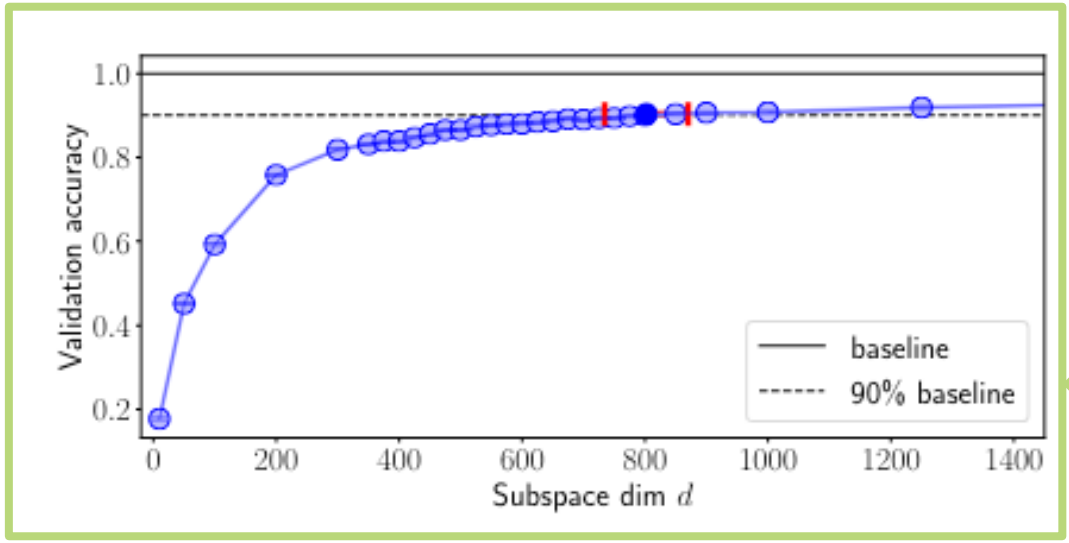
Hypothesis:

They are intrinsically low dimensional

Intrinsic Dimensionality

Motivation

- Maybe the number of parameters in a model is **not** a great measure of how many degrees of freedom are needed to successfully learn some problems
- How could we measure the number of degrees of freedom you **really** need?



Intrinsic Dimension

- *Definition* from Li et al. (2018):
- Learn a neural network with *D* parameters in a random lower dimensional subspace, *d*
- Then repeat, gradually increasing the dimensionality, *d*
- Let the intrinsic dimension be the value of *d* when good solutions (above 90% threshold of full parameterization) start to appear

For MNIST digit recognition, original neural network has *D*=199,210 parameters but the intrinsic dimension is only *d*=750

Li et al., Measuring the Intrinsic Dimension of Objective Landscapes, ICLR 2018

Intrinsic Dimensionality

How do we learn in a low dimensional subspace?

Standard optimization, which we will refer to hereafter as the *direct* method of training, entails evaluating the gradient of a loss with respect to $\theta^{(D)}$ and taking steps directly in the space of $\theta^{(D)}$. To train in a random subspace, we instead define $\theta^{(D)}$ in the following way:

$$\theta^{(D)} = \theta_0^{(D)} + P\theta^{(d)} \tag{2}$$

where P is a randomly generated $D \times d$ projection matrix[1] and $\theta^{(d)}$ is a parameter vector in a generally smaller space $\mathbb{R}^d$. $\theta_0^{(D)}$ and P are randomly generated and frozen (not trained), so the system has only d degrees of freedom. We initialize $\theta^{(d)}$ to a vector of all zeros, so initially $\theta^{(D)} = \theta_0^{(D)}$.

Li et al., Measuring the Intrinsic Dimension of Objective Landscapes, ICLR 2018

Intrinsic Dimensionality

- Using similar techniques, Aghajanyan et al. (2020) measure the intrinsic dimension of LLMs
- Empirical results suggest that pre-training finds parameters that have low intrinsic dimensionality
- Number of parameters:
  - BERT-Base: 110 million
  - BERT-Large: 345 million

	SAID		DID	
Model	MRPC	QQP	MRPC	QQP
BERT-Base	1608	8030	1861	9295
BERT-Large	1037	1200	2493	1389
RoBERTa-Base	896	896	1000	1389
RoBERTa-Large	**207**	**774**	322	**774**

Table 1: Estimated d_{90} intrinsic dimension for a set of sentence prediction tasks and common pre-trained models. We present both the *SAID* and *DID* methods.

Aghajanyan et al., Intrinsic Dimensionality Explains the Effectiveness of Language Model Fine-Tuning, 2020

LoRA

- **Motivation #1:**
  "inspiration from Li et al. (2018a); Aghajanyan et al. (2020) shows that the learned over-parametrized models in fact reside on a low intrinsic dimension."
- **Motivation #2:**
  Directly optimizing the prompt, as in prefix tuning, leads to non-monotonic changes in performance as the number of parameters increases (we want more parameters to mean better performance!)
- **Motivation #3:**
  Adapters and related methods introduce inference latency at test time that is non-trivial!

Batch Size Sequence Length $\|\Theta\|$	32 512 0.5M	16 256 11M	1 128 11M
Fine-Tune/LoRA	1449.4±0.8	338.0±0.6	19.8±2.7
$\text{Adapter}^{\text{L}}$	1482.0±1.0 (+2.2%)	354.8±0.5 (+5.0%)	23.9±2.1 (+20.7%)
$\text{Adapter}^{\text{H}}$	1492.2±1.0 (+3.0%)	366.3±0.5 (+8.4%)	25.8±2.2 (+30.3%)

Table 1: Infernece latency of a single forward pass in GPT-2 medium measured in milliseconds, averaged over 100 trials. We use an NVIDIA Quadro RTX8000. "$|\Theta|$" denotes the number of trainable parameters in adapter layers. $\text{Adapter}^{\text{L}}$ and $\text{Adapter}^{\text{H}}$ are two variants of adapter tuning, which we describe in Section 5.1. The inference latency introduced by adapter layers can be significant in an online, short-sequence-length scenario. See the full study in Appendix B.

Hu et al., LoRA: Low-Rank Adaptation of Large Language Models, 2021

LoRA

Key Idea

- Keep the original pretrained parameters $\mathbf{W}_0$ fixed
- Learn an additive modification $\Delta\mathbf{W}$
- Define $\Delta\mathbf{W}$ via a low rank decomposition:

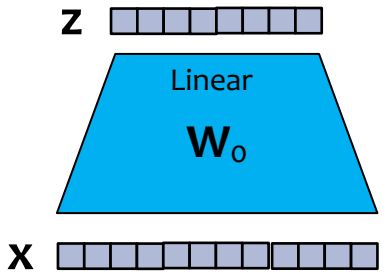
$$\Delta W = BA$$

where **BA** has rank r (r<< k and d)

Standard Linear Layer

$$\mathbf{z} = \mathbf{W}_0\mathbf{x}$$

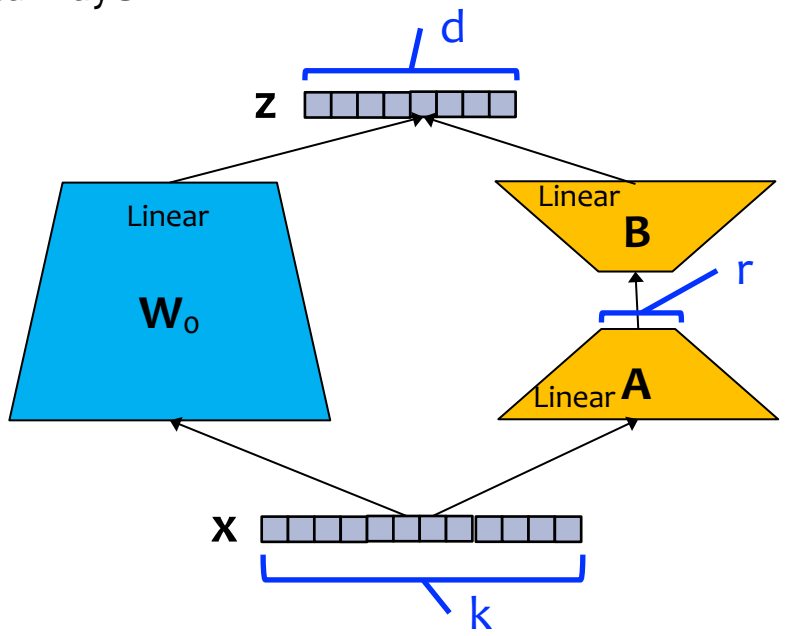
$$\mathbf{W}_0 \in \mathbb{R}^{d\times k}, \mathbf{x} \in \mathbb{R}^k, \mathbf{z} \in \mathbb{R}^d$$



LoRA Linear Layer

$$\mathbf{z} = \mathbf{W}_0\mathbf{x} + \mathbf{BAx}$$
$$= (\mathbf{W}_0 + \mathbf{BA})\mathbf{x}$$

$$\mathbf{W}_0 \in \mathbb{R}^{d\times k},$$
$$\mathbf{A} \in \mathbb{R}^{r\times k}, \mathbf{B} \in \mathbb{R}^{d\times r}$$
where $r << \min(d, k)$



Hu et al., LoRA: Low-Rank Adaptation of Large Language Models, 2021

LoRA

Initialization

- We initialize the trainable parameters:
  $A_{ij} \sim N(0, \sigma^2)\ \forall i, j$

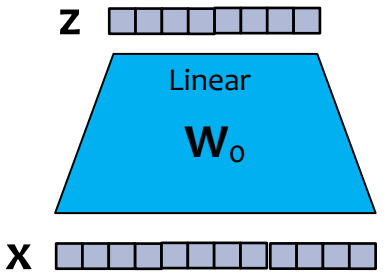
  B = 0
- Thus, at the start of fine tuning, the parameters have their pretrained values:

$$\Delta W = BA = 0$$
$$W_0 + BA = W_0$$

Standard Linear Layer

$$\mathbf{z} = \mathbf{W}_0\mathbf{x}$$

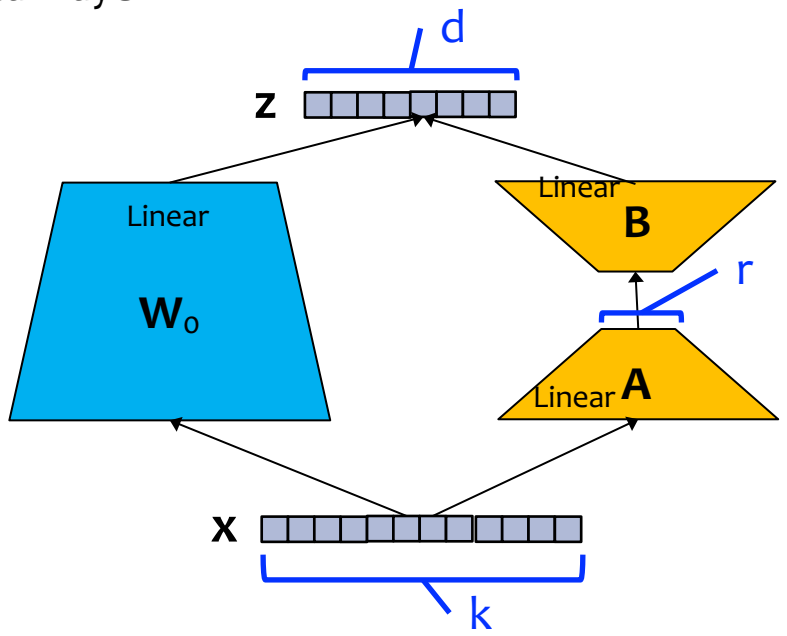
$$\mathbf{W}_0 \in \mathbb{R}^{d\times k}, \mathbf{x} \in \mathbb{R}^k, \mathbf{z} \in \mathbb{R}^d$$



LoRA Linear Layer

$$\mathbf{z} = \mathbf{W}_0\mathbf{x} + \mathbf{BAx}$$
$$= (\mathbf{W}_0 + \mathbf{BA})\mathbf{x}$$

$$\mathbf{W}_0 \in \mathbb{R}^{d\times k},$$
$$\mathbf{A} \in \mathbb{R}^{r\times k}, \mathbf{B} \in \mathbb{R}^{d\times r}$$
where $r << \min(d, k)$



Hu et al., LoRA: Low-Rank Adaptation of Large Language Models, 2021

LoRA

Hot Swapping Parameters

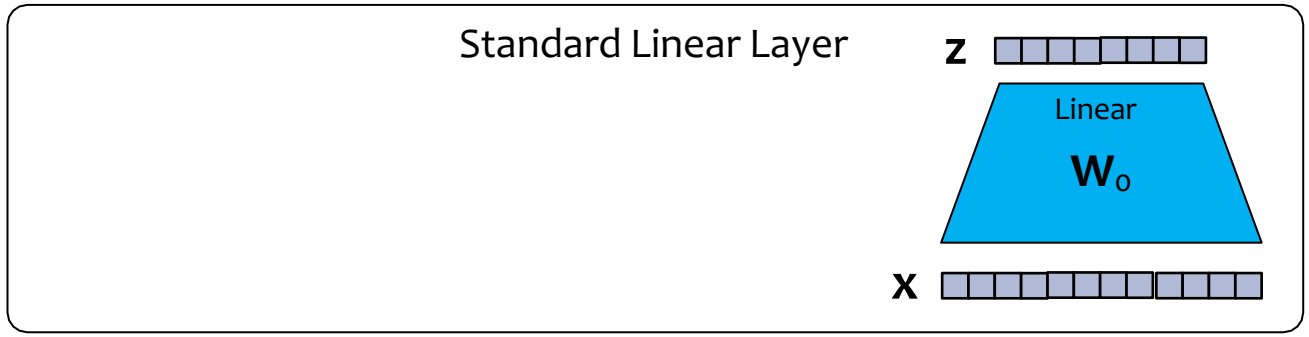
- $\mathbf{W}_0$ and **BA** have the same dimension, so we can ”swap” the LoRA parameters in and out of a Standard Linear Layer
- To get a Standard Linear Layer with parameters **W** that includes our LoRA fine tuning:

$$W \leftarrow W_0 + BA$$

- To remove the LoRA finetuning from that Standard Linear Layer:

$$W \leftarrow W - BA = W_0$$

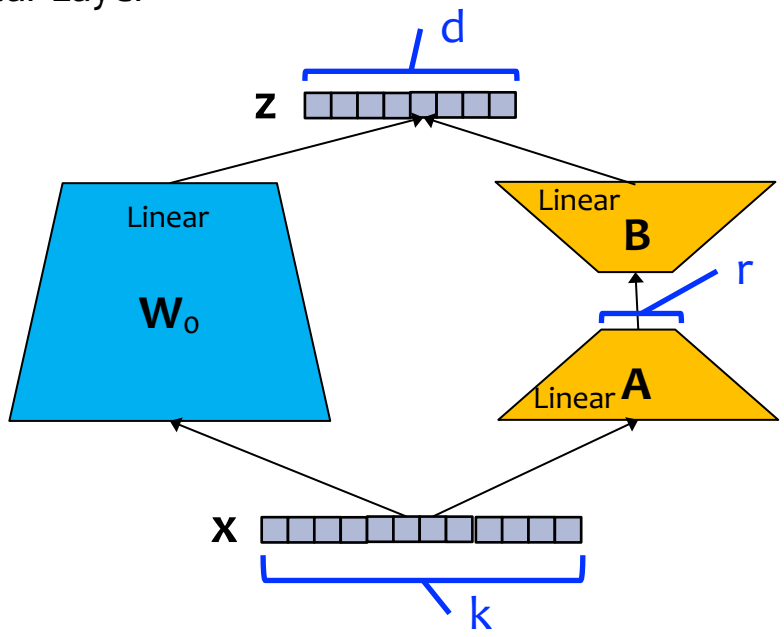
- If we do LoRA training on two tasks s.t. the parameters **B’A’** are for task 1 and **B’’A’’** are for task 2, then we can swap back and forth between them



LoRA Linear Layer

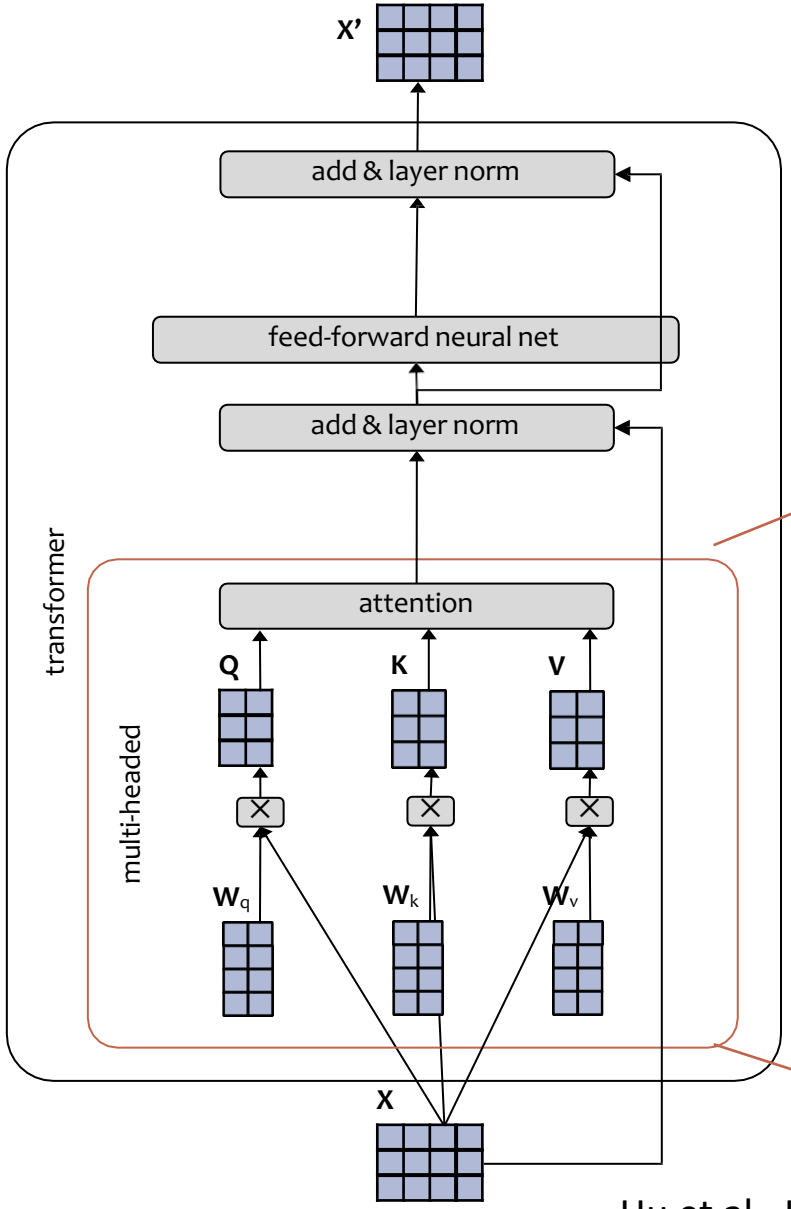
$$\mathbf{z} = \mathbf{W}_0\mathbf{x} + \mathbf{BAx}$$
$$= (\mathbf{W}_0 + \mathbf{BA})\mathbf{x}$$

$$\mathbf{W}_0 \in \mathbb{R}^{d\times k},$$
$$\mathbf{A} \in \mathbb{R}^{r\times k}, \mathbf{B} \in \mathbb{R}^{d\times r}$$
$$\text{where } r << \min(d, k)$$



Hu et al., LoRA: Low-Rank Adaptation of Large Language Models, 2021

Transformer Layer



multi-headed attention

$$\mathbf{X}'' = \text{concat}(\mathbf{X}'^{(1)}, \ldots, \mathbf{X}'^{(h)})$$

$$\mathbf{X}'^{(i)} = \text{softmax}\left(\frac{\mathbf{Q}^{(i)}(\mathbf{K}^{(i)})^T}{\sqrt{d_k}} + \mathbf{M}\right)\mathbf{V}^{(i)}$$

$$\mathbf{Q}^{(i)} = \mathbf{X}\mathbf{W}_q^{(i)}$$

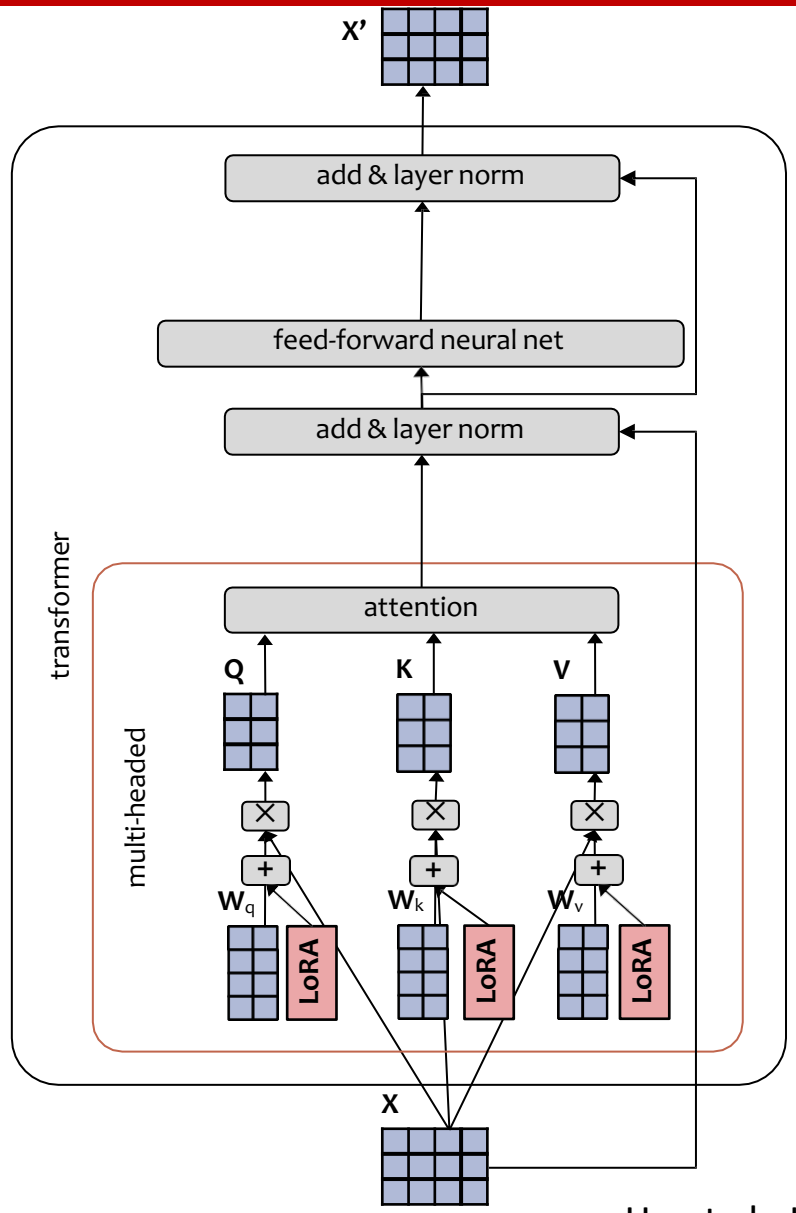
$$\mathbf{K}^{(i)} = \mathbf{X}\mathbf{W}_k^{(i)}$$

$$\mathbf{V}^{(i)} = \mathbf{X}\mathbf{W}_v^{(i)}$$

$$\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_N]^T$$

Hu et al., LoRA: Low-Rank Adaptation of Large Language Models, 2021

LoRA for Transformer

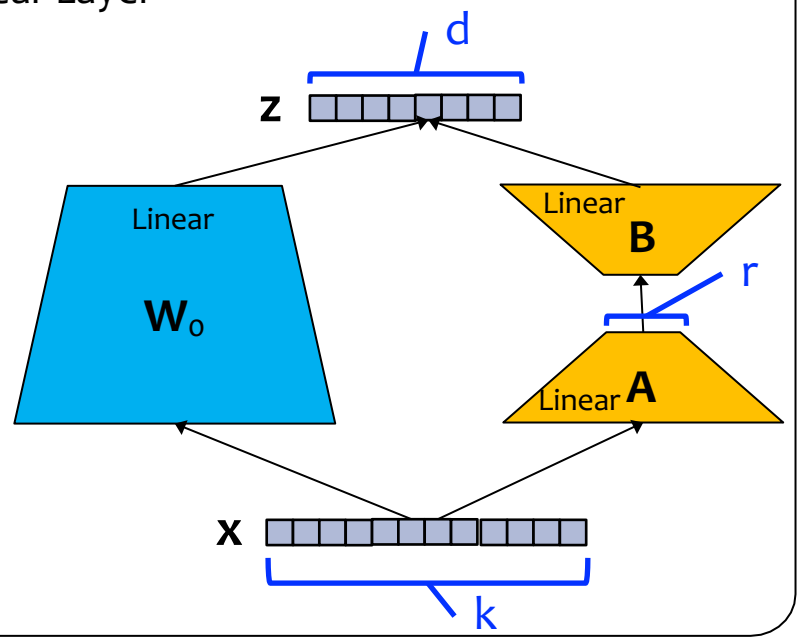


- LoRA linear layers could replace *every* linear layer in the Transformer layer
- But the original paper only applies LoRA to the attention weights

LoRA Linear Layer

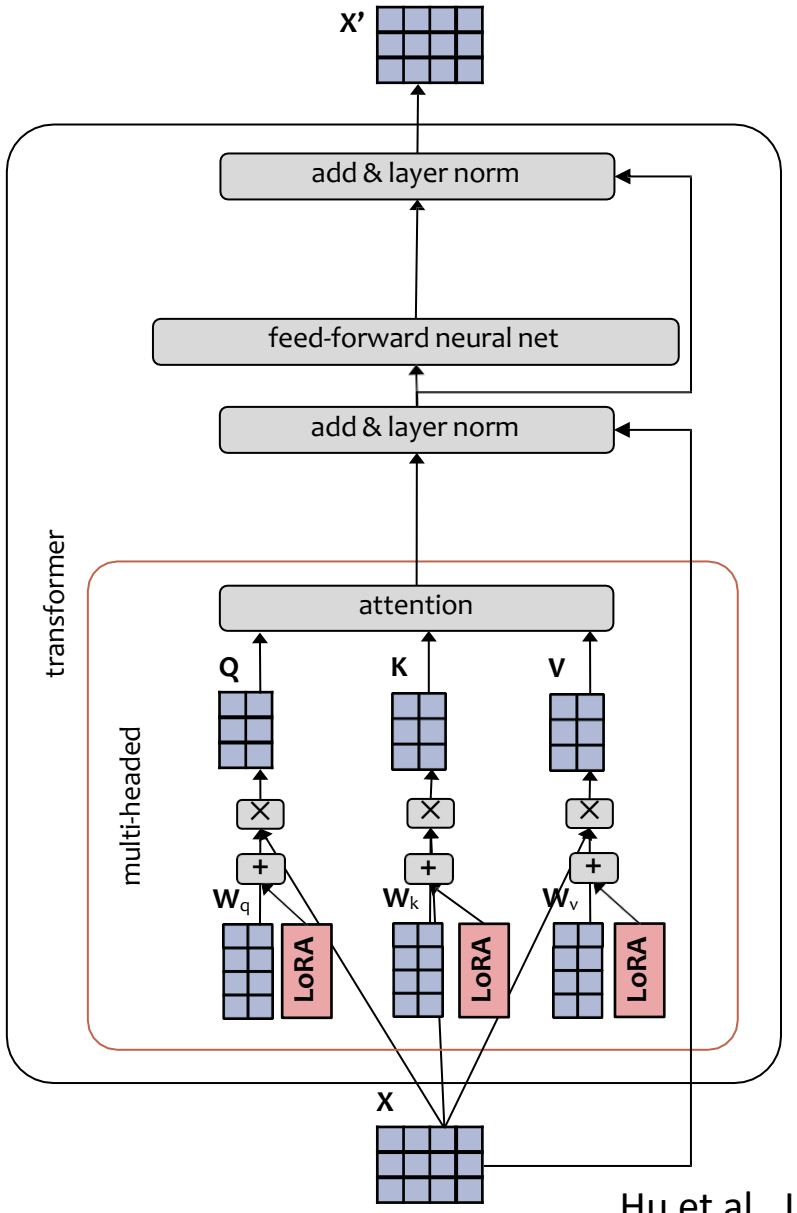
$$\mathbf{z} = \mathbf{W}_0\mathbf{x} + \mathbf{BAx}$$
$$= (\mathbf{W}_0 + \mathbf{BA})\mathbf{x}$$

$$\mathbf{W}_0 \in \mathbb{R}^{d\times k},$$
$$\mathbf{A} \in \mathbb{R}^{r\times k}, \mathbf{B} \in \mathbb{R}^{d\times r}$$
where $r << \min(d, k)$



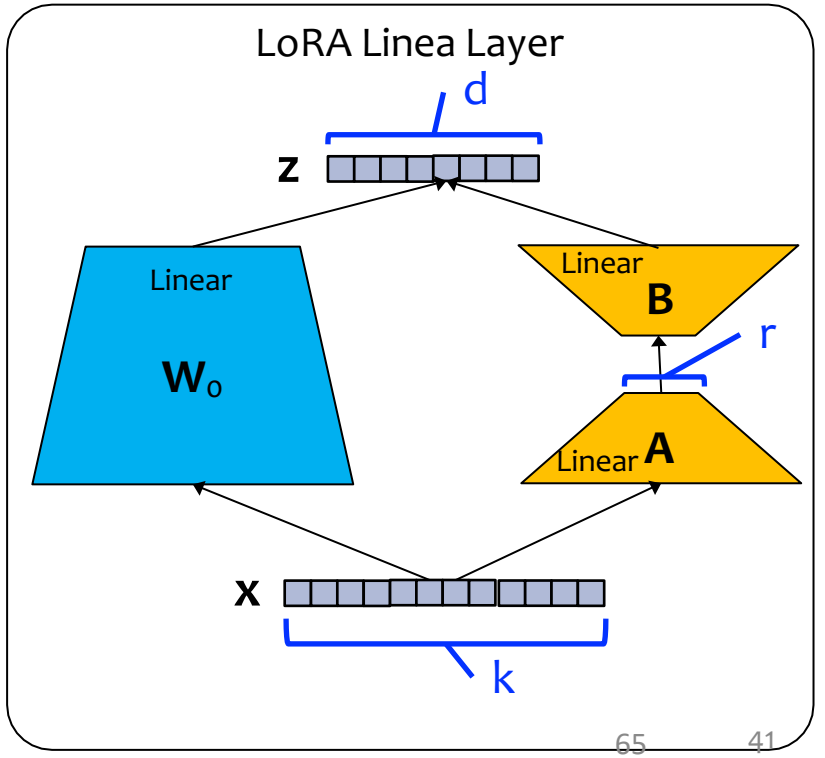
Hu et al., LoRA: Low-Rank Adaptation of Large Language Models, 2021

LoRA for Transformer



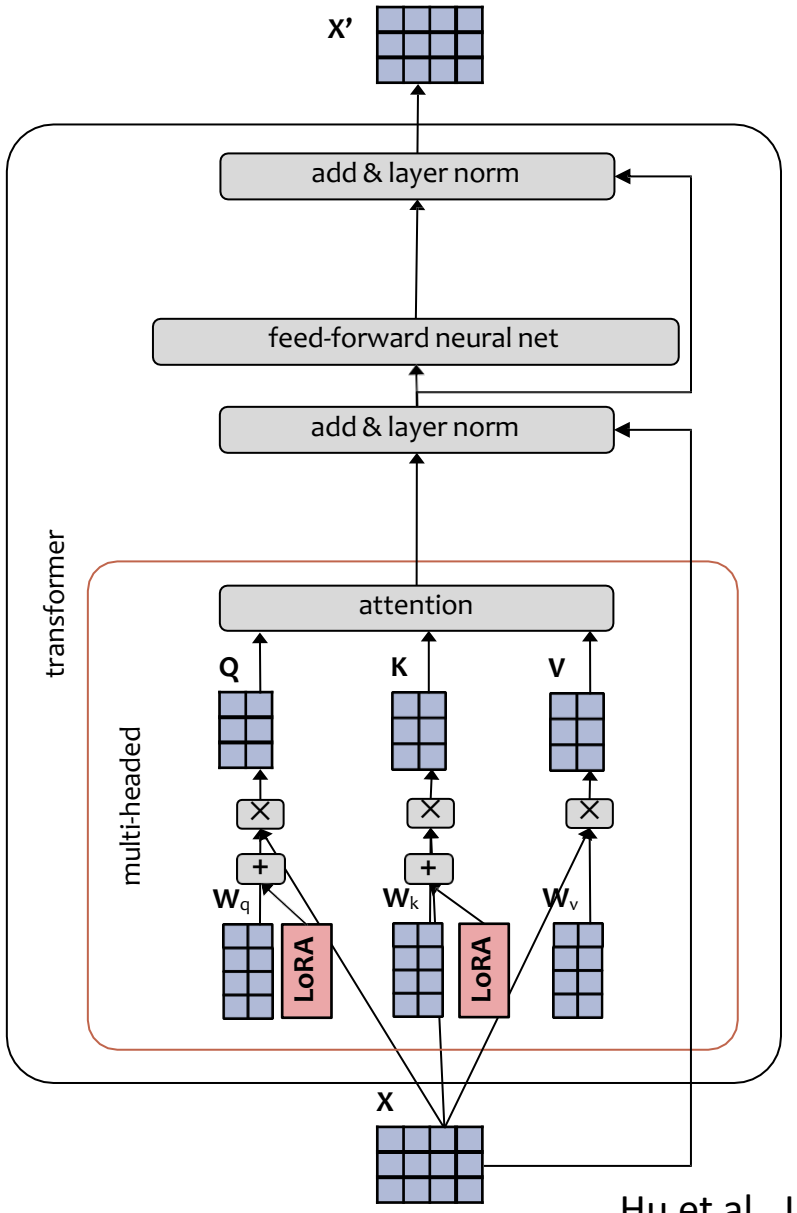
- LoRA linear layers could replace *every* linear layer in the Transformer layer
- But the original paper only applies LoRA to the attention weights

$$Q = \text{LoRALinear}(X; W_q, A_q, B_q)$$
$$K = \text{LoRALinear}(X; W_k, A_k, B_k)$$
$$V = \text{LoRALinear}(X; W_v, A_v, B_v)$$



Hu et al., LoRA: Low-Rank Adaptation of Large Language Models, 2021

LoRA for Transformer



- LoRA linear layers could replace *every* linear layer in the Transformer layer
- But the original paper only applies LoRA to the attention weights
- Empirically, for GPT-3, they also find that it is most efficient to include LoRA **only on the query and key** linear layers:

	# of Trainable Parameters = 18M						
Weight Type	W_q	W_k	W_v	W_o	W_q, W_k	W_q, W_v	W_q, W_k, W_v, W_o
Rank r	8	8	8	8	4	4	2
WikiSQL ($\pm 0.5\%$)	70.4	70.0	73.0	73.2	71.4	**73.7**	**73.7**
MultiNLI ($\pm 0.1\%$)	91.0	90.8	91.0	91.3	91.3	91.3	**91.7**

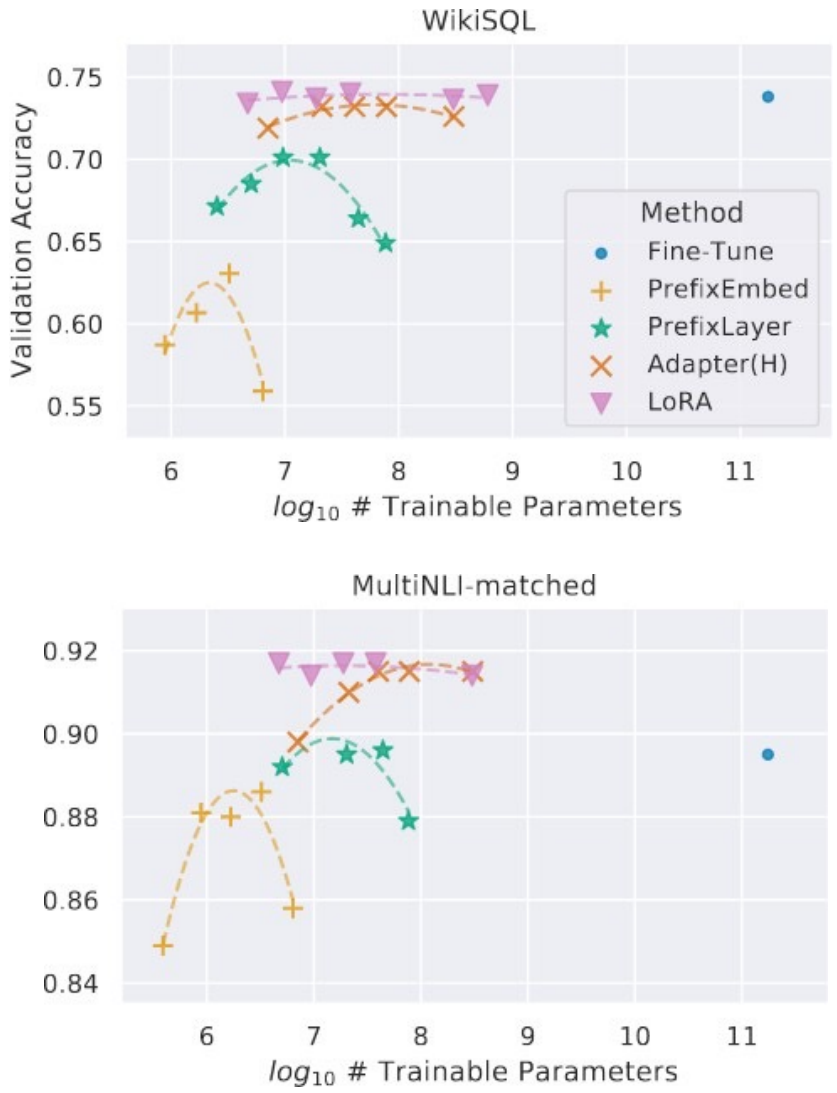
Table 5: Validation accuracy on WikiSQL and MultiNLI after applying LoRA to different types of attention weights in GPT-3, given the same number of trainable parameters. Adapting both W_q and W_v gives the best performance overall. We find the standard deviation across random seeds to be consistent for a given dataset, which we report in the first column.

Hu et al., LoRA: Low-Rank Adaptation of Large Language Models, 2021

LoRA Results

Takeaways

- Applied to GPT-3, LoRA achieves performance almost as good as full fine-tuning, but with far fewer parameters
- On some tasks it even outperforms full finetuning
- For some datasets r=1 is sufficient



Hu et al., LoRA: Low-Rank Adaptation of Large Language Models, 2021

LoRA Results

Takeaways

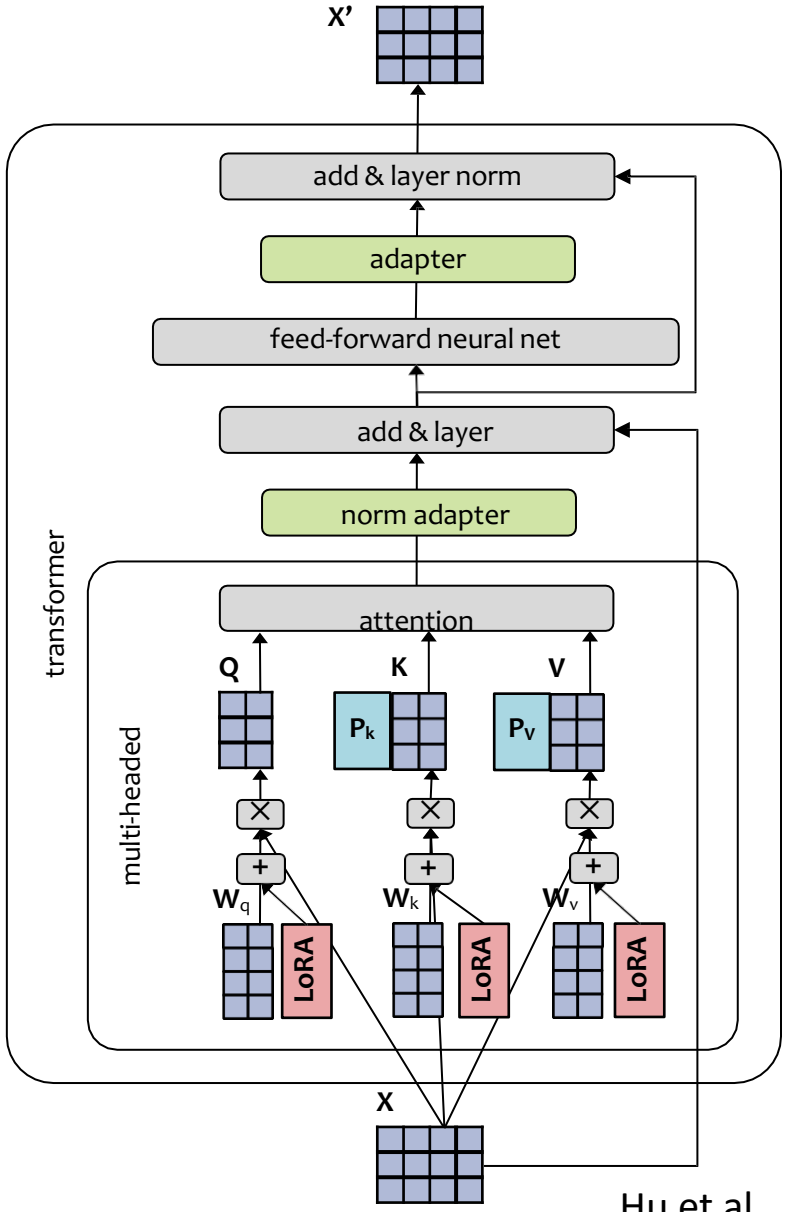
- Applied to GPT-3, LoRA achieves performance almost as good as full fine-tuning, but with far fewer parameters
- On some tasks it even outperforms full finetuning
- For some datasets a rank of r=1 is sufficient

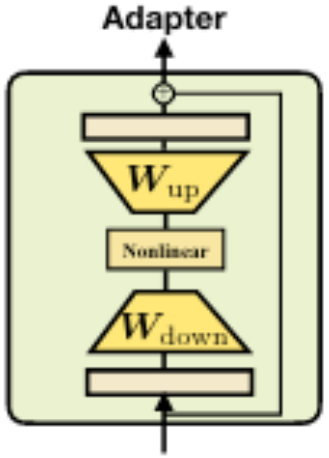
	Weight Type	$r=1$	$r=2$	$r=4$	$r=8$	$r=64$
WikiSQL($\pm 0.5\%$)	W_q	68.8	69.6	70.5	70.4	70.0
	W_q, W_v	73.4	73.3	73.7	73.8	73.5
	W_q, W_k, W_v, W_o	74.1	73.7	74.0	74.0	73.9
MultiNLI ($\pm 0.1\%$)	W_q	90.7	90.9	91.1	90.7	90.7
	W_q, W_v	91.3	91.4	91.3	91.6	91.4
	W_q, W_k, W_v, W_o	91.2	91.7	91.7	91.5	91.4

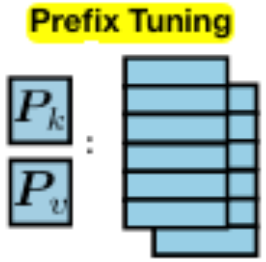
Method	MNLI(m)-100	MNLI(m)-1k	MNLI(m)-10k	MNLI(m)-392K
GPT-3 (Fine-Tune)	60.2	**85.8**	88.9	89.5
GPT-3 (PrefixEmbed)	37.6	75.2	79.5	88.6
GPT-3 (PrefixLayer)	48.3	82.5	85.9	89.6
GPT-3 (LoRA)	**63.8**	85.6	**89.2**	**91.7**

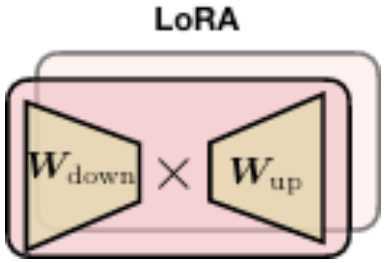
Hu et al., LoRA: Low-Rank Adaptation of Large Language Models, 2021

PEFT for Transformer: Summary



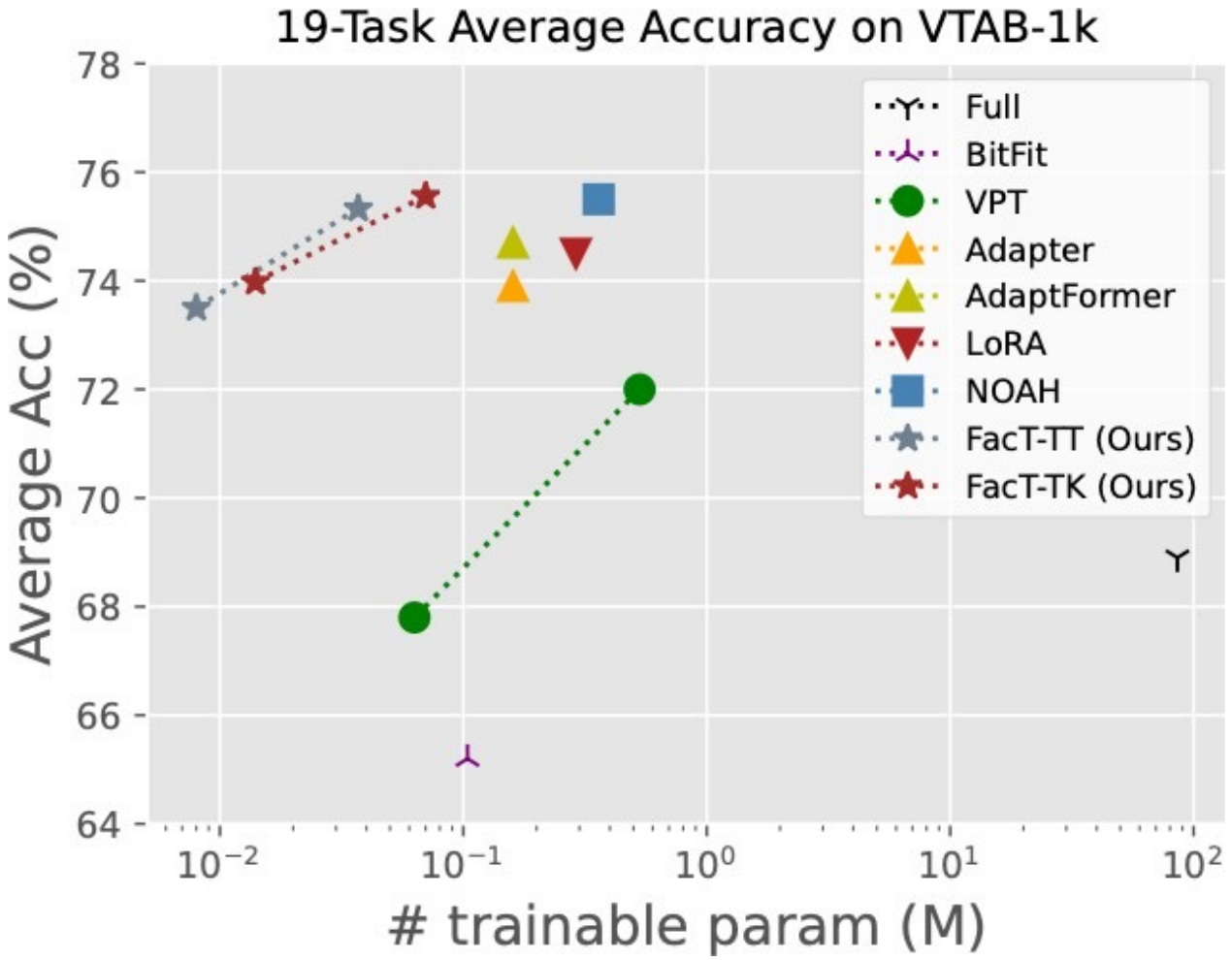






Hu et al., LoRA: Low-Rank Adaptation of Large Language Models, 2021

PEFT for Vision Transformer



- For various computer vision tasks, parameter efficient transfer-learning (PETL) is sometimes **better** than full fine-tuning!
- VTAB-1k is a collection of 19 different vision tasks; here we're seeing average performance across tasks
- (FacT is another low-rank method capable of dramatically reducing the number of parameters tuned.)

Jie and Deng, FacT: Factor-Tuning for Lightweight Adaptation on Vision Transformer, AAAI 2023