Efficient Fine-Tuning of Large Language Models

10/1/2024

Sally Lee, Chen Wang, Wanzhou Liu

Overview of Parameter-Efficient Fine-Tuning

- Prompt-based
  - Prompt Tuning: Simple yet effective at learning soft prompts to condition frozen language models to perform specific downstream tasks
- Adapter-based
  - Adapter Modules: Small, trainable layers into a pre-trained model, allowing task-specific adjustments while keeping the majority of the model's parameters frozen, reducing the number of parameters that need to be trained.
- Reparametrization-based
  - LoRA, DoRA: Apply low-rank decomposition to reduce the number of trainable parameters during fine-tuning.

The Power of Scale for Parameter-Efficient Prompt Tuning

9/2/2021

Brian Lester, Rami Al-Rfou, Noah Constant

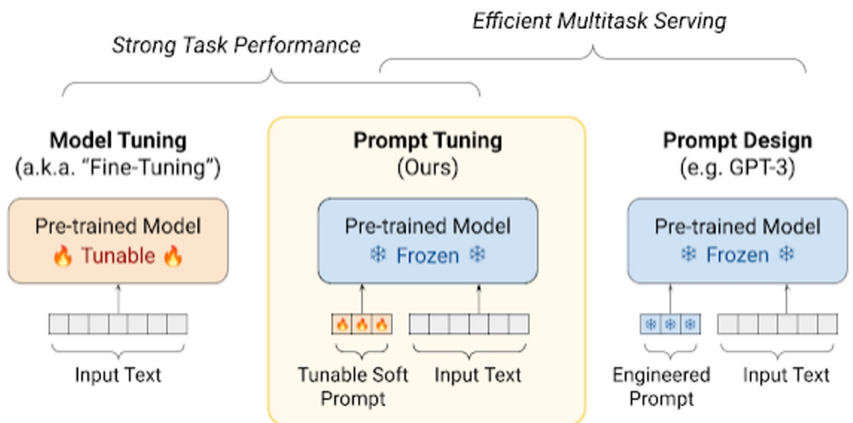
https://arxiv.org/abs/2104.08691

How to adapt general-purpose models to downstream tasks

- **Model tuning (fine tuning):** adjust every weight in the network
  - Standard practice since GPT and BERT
  - Problem: impractical to store and serve tuned copy of model for each downstream task as model becomes larger
- **Prompt design:** hand-craft text prompt with a description or examples of the task
  - Share a single frozen pre-trained language model (all weights fixed) across all downstream tasks
  - GPT-3 showed a frozen model can be conditioned to perform different tasks through “in-context” learning
  - Example: condition a model for sentiment analysis
    - Attach the prompt “Is the following movie review positive or negative?” before the input sequence “This movie was amazing!”
  - Pros: simplifies serving and allows for mixed-task inference
  - Cons: text prompts require manual effort to design; even well-designed prompts perform poorly compared to model tuning

Prompt Tuning

- More efficient method for conditioning frozen models using tunable soft prompts
- Soft prompts:
  - like engineered text prompts, concatenated to input text
  - instead of selecting from existing vocabulary items, “tokens” of the soft prompt are learnable vectors
  - can be optimized end-to-end over training dataset
  - allows prompt to condense information from datasets with thousands / millions of examples
    - Huge improvement from discrete text prompts - usually limited to under 50 examples due to model’s input length constraints

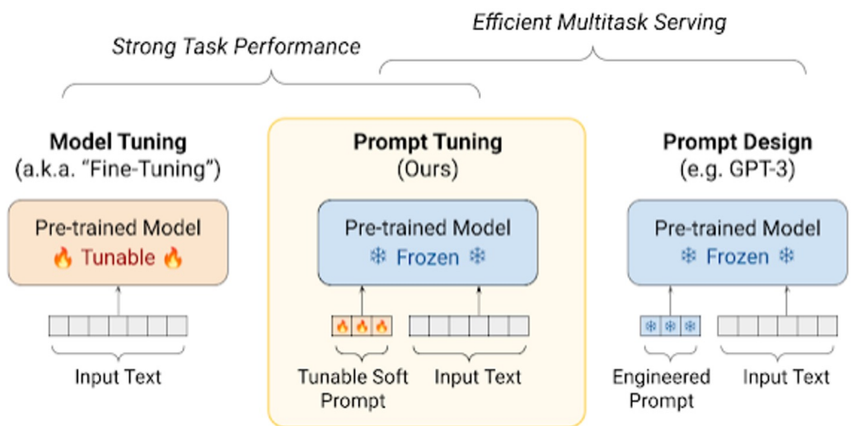


Creating Soft Prompts

1. Initialize prompt as fixed-length sequence of vectors

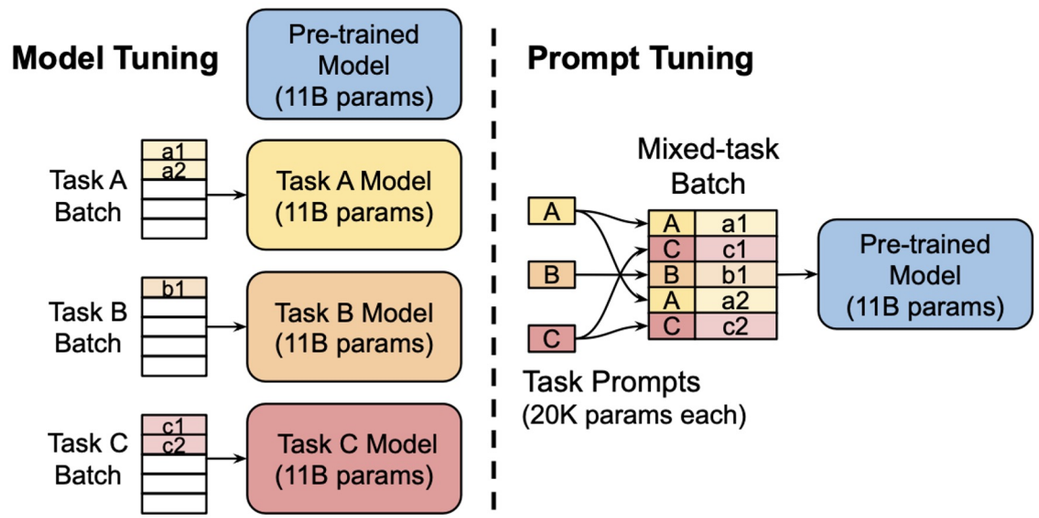
2. Attach vectors to beginning of each embedded input and feed combined sequence into model

3. Calculate error between model's prediction and target, back-propagate to calculate gradients, but only apply updates to new learnable vectors, keeping core model frozen



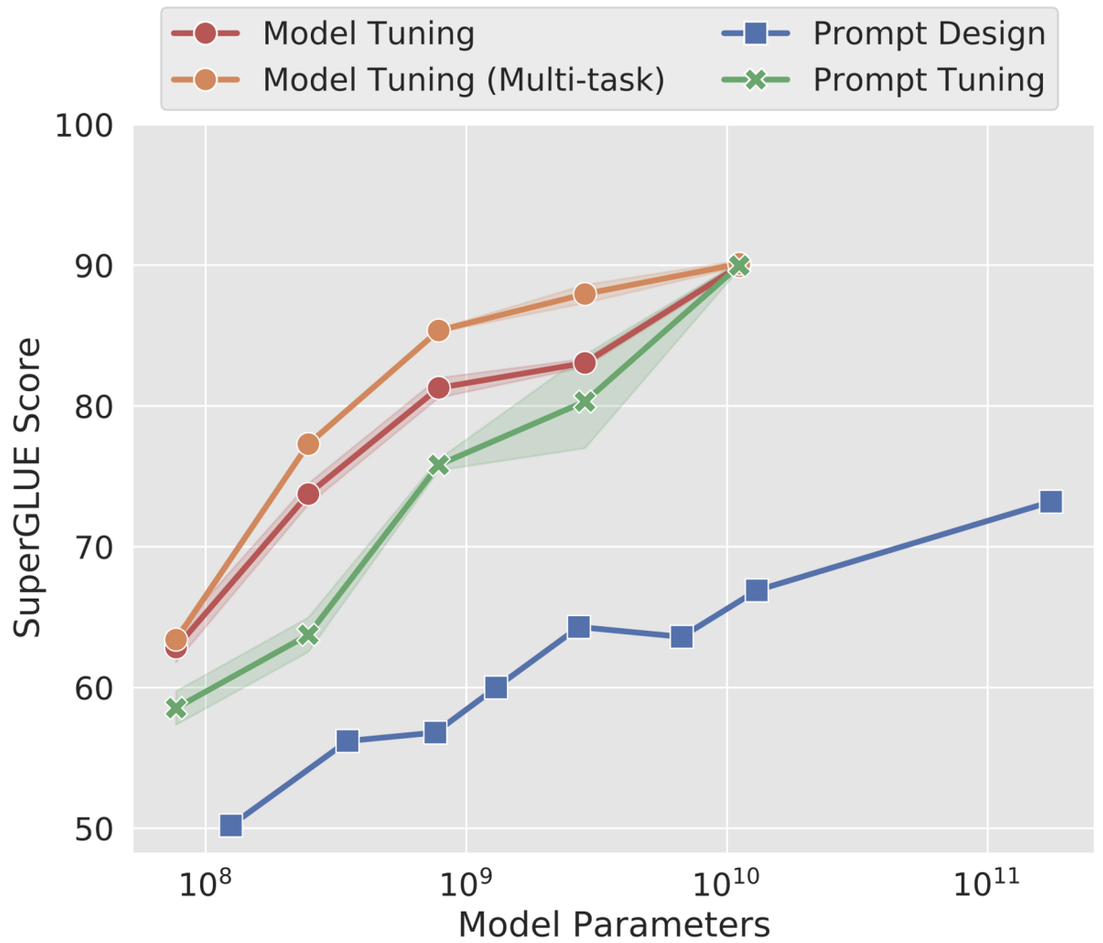
Storage Cost Comparison

- **Model tuning:** requires making a task-specific copy of entire pre-trained model for each downstream task and performing inference in separate batches
- **Prompt tuning:** only requires storing small task-specific prompt for each task and enables mixed-task inference using original pre-trained model
- **Example with T5 XXL model:**
  - Each copy of tuned model requires 11 billion parameters
  - Tuned prompt only requires 20K parameters per task (prompt length = 5 tokens)
    - reduction of over 5 orders of magnitude



Prompt Tuning becomes more competitive with scale

- Prompt Tuning significantly outperforms prompt design
- As model size increases, prompt tuning catches up with performance of model tuning



Ablation Study

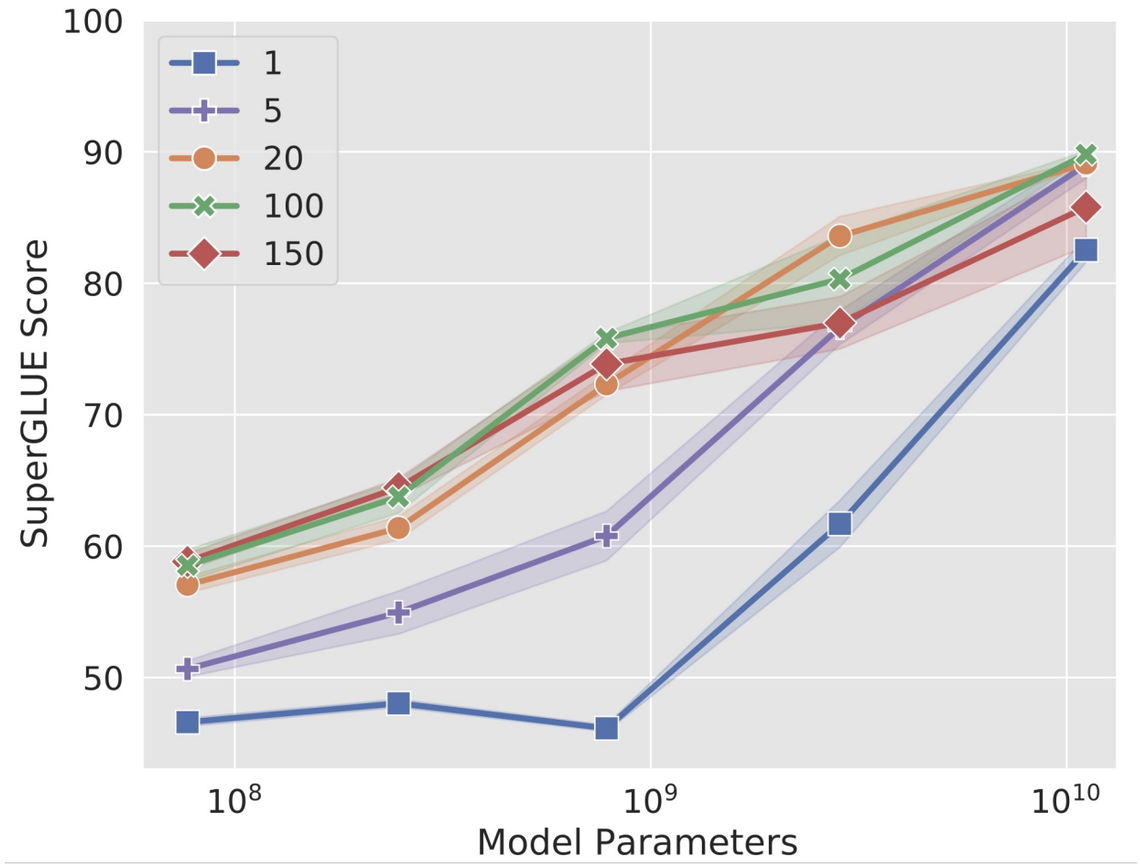
1. Prompt Length
2. Prompt Initialization
3. Pre-training method
4. LM adaptation steps

Prompt Length

- parameter cost of method: EP
    - E: token embedding dimension
    - P: prompt length
- shorter the prompt, fewer new parameters to tune, so want to find a minimal length that still performs well

Prompt Length Results

- for most model sizes, longer the prompt higher the performance
- XXL model still yields relatively strong performance with single token prompt
- larger the model, the less conditioning signal is needed to achieve target behavior



T5 models pre-train on span corruption objective

- tasked with reconstructing masked spans in the input text, which are marked with unique sentinel tokens
    - ex. “Thank you <X> me to your party <Y> week
- target output text consists of all masked content, separated by sentinels, plus a final sentinel
    - ex. “<X> for inviting <Y> last <Z>”

Concerns around Span Corruption in Prompt Tuning

- never seen truly natural input text (free of sentinel tokens)
- never asked to predict truly natural targets
- every pre-training target will begin with a sentinel
- unlike fine-tuning, prompt alone might struggle to override unnatural tendency to output sentinels since decoder priors cannot be adjusted

Experiment with T5 in 3 Settings

1. Span Corruption: use pre-trained T5 as frozen model, and test its ability to output the expected text for downstream tasks

2. Span Corruption + sentinel: use the same model, but prepend all downstream targets with a sentinel to more closely resemble targets seen in pre-training

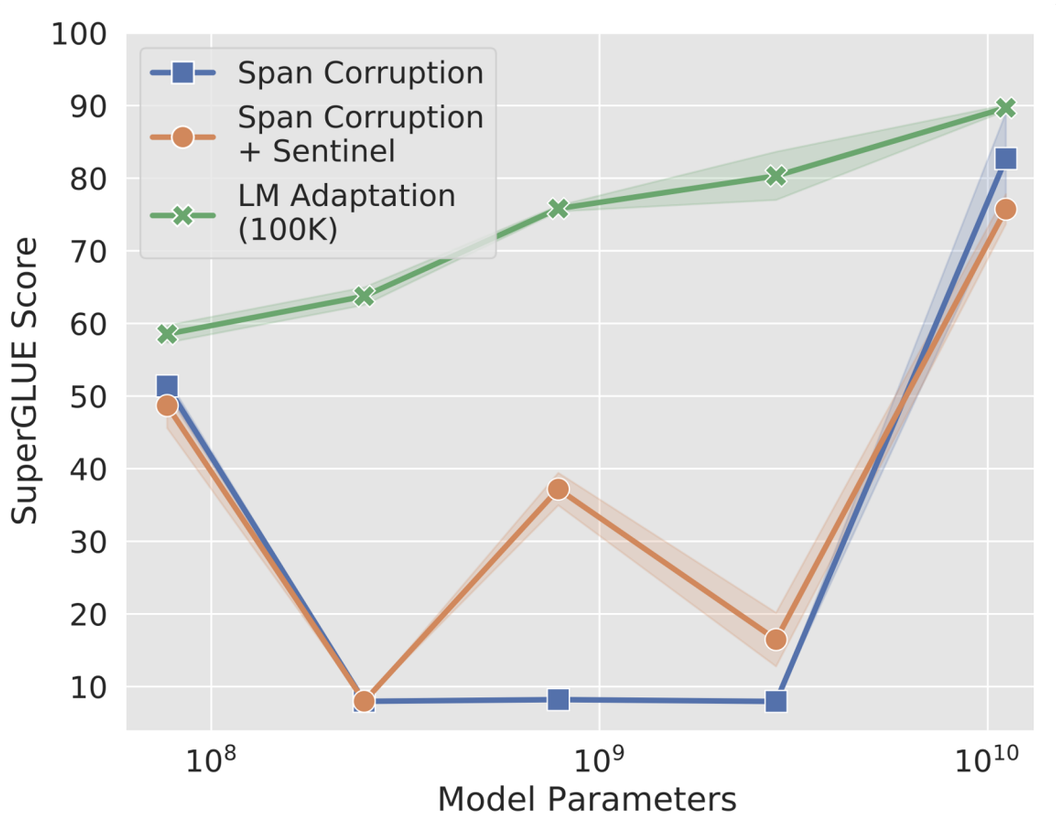
3. LM adaptation: continue T5's self-supervised training for a small number of additional steps, but use LM objective

- given natural text prefix as input, model must produce natural text continuation as output

- happens only once, producing a single frozen model to be reused for prompt tuning across any number of downstream tasks

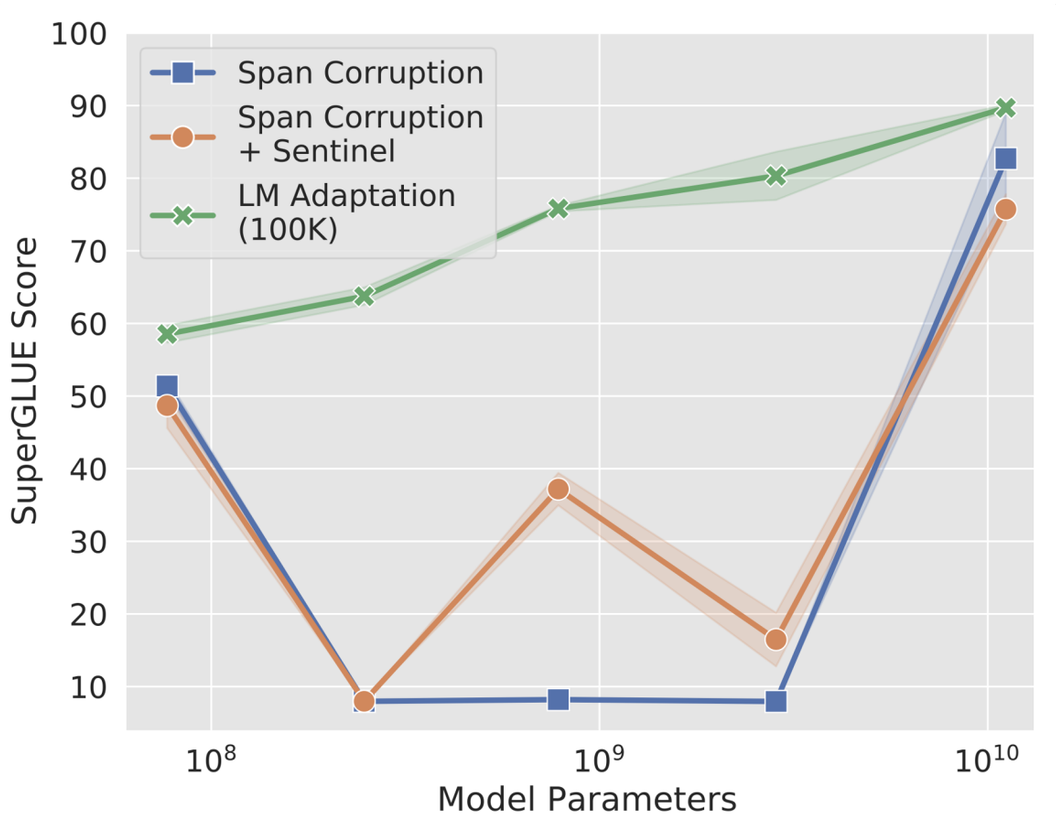
Pre-training methods

- T5's default span corruption objective is not well-suited for training frozen models to be later conditioned by prompts
- even adding sentinel to downstream targets shows little benefit
- LM adaptation adds value across all model sizes
- XXL model size is most forgiving even with span corruption



Span Corruption's Instability across model sizes

- small model outperforms base, large, and XL models
- for many tasks, mid-sized models never learn to output a legal class label and thus score 0%
- 2 most common error modes: copying subspans from input and predicting empty string
- models pre-trained with span corruption objective are unreliable: only 2 out of 5 models work well



Prompt Initialization Methods

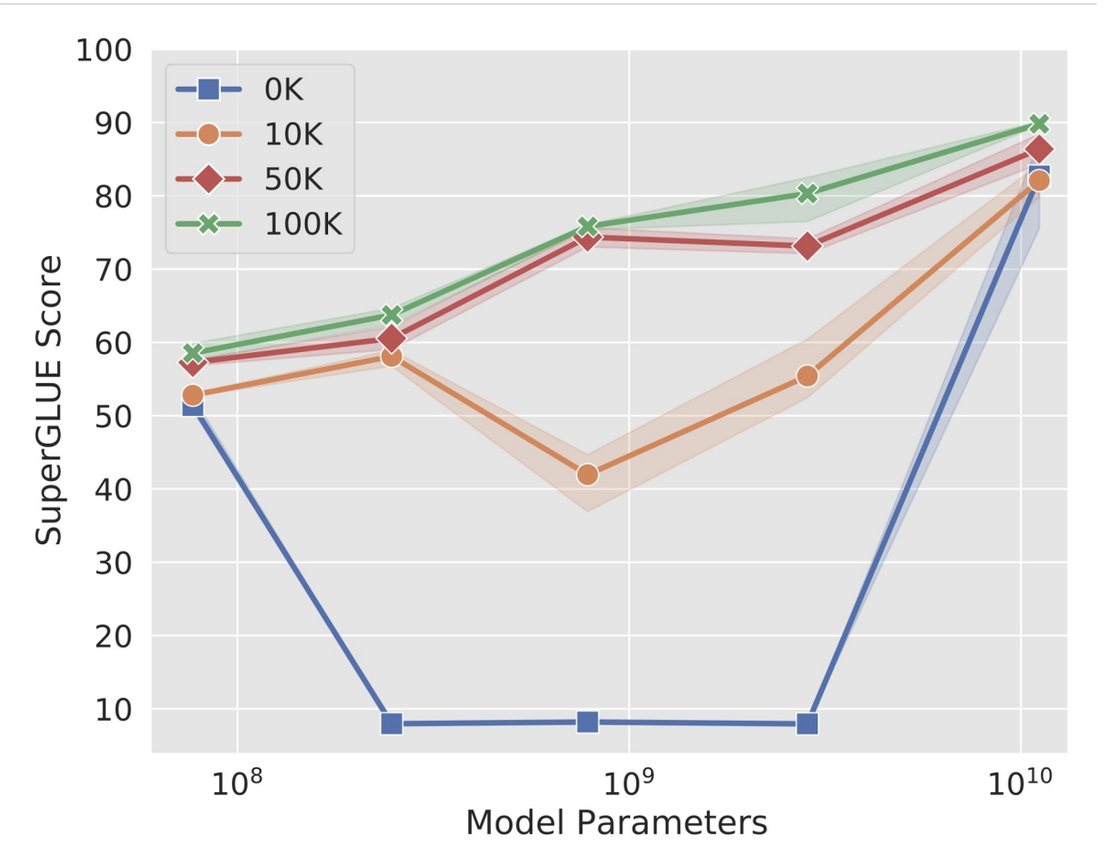
- **random:** sample uniformly from range [-0.5, 0.5]
- **sampled vocabulary:** restrict to 5000 most common tokens in T5's sentence-piece vocabulary, which is ordered by likelihood in pre-training corpus
- **class label:** use embeddings for string representations of each class in downstream task to initialize a token in prompt
    - **multi-token:** average token embeddings
    - **longer prompts:** fall back to sampled vocab strategy to fill in prompt if expended class labels before initializing all prompt tokens

Prompt Initialization Results

- class-based initialization performs best
- random initialization performs worst
- little difference for XXL size model

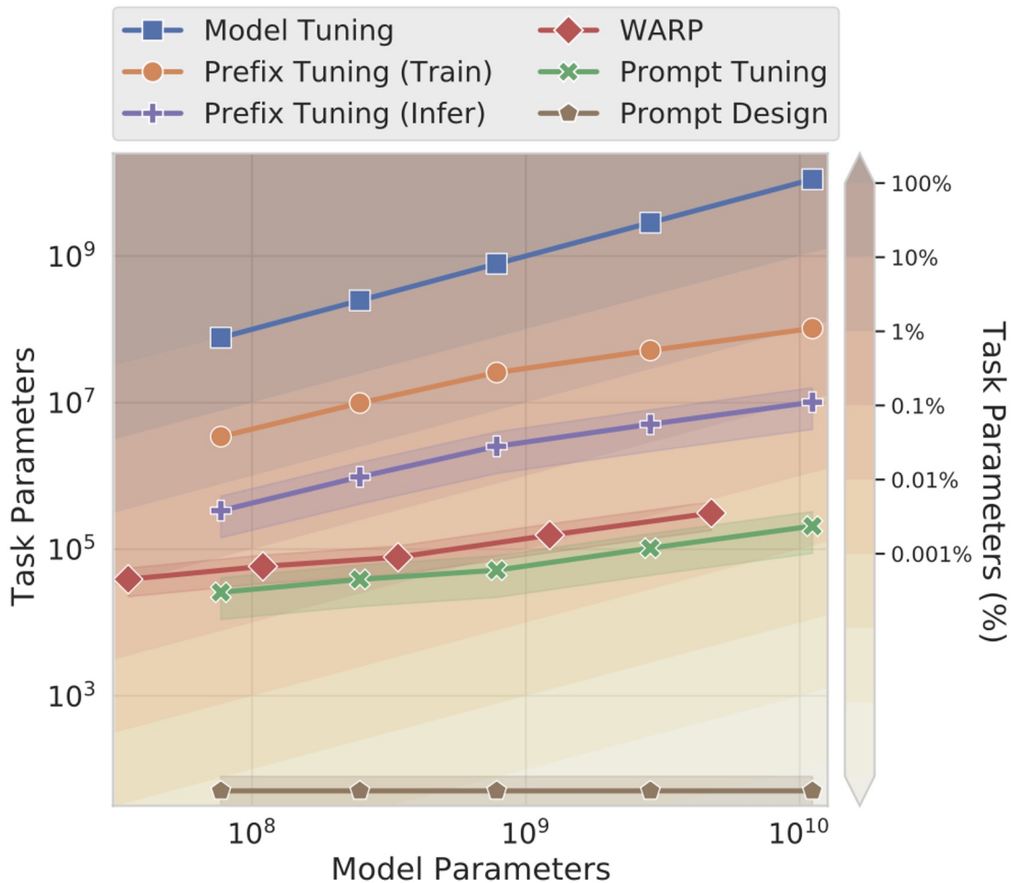
LM adaptation steps

- longer adaptation performs better
- transition from span corruption to language modeling objective is not a trivial change
- making an effective switch takes an investment of training resources (10% of the steps of the original T5 pre-training)
- again, little difference for XXL size model



Comparison to Similar Approaches

- Prefix Tuning:
  - learn sequence of prefixes that are prepended at every transformer layer
  - like learning transformer activations that are fixed across examples at every network layer
  - Prompt tuning uses single prompt representation prepended to embedded input
- WARP:
  - prompt parameters are added to input layer
  - Works with masked language models, relying on a mask token and a learnable output layer to project mask to class logits
  - Limited to classification
  - Prompt tuning does not require any changes to input or task-specific head



Resilience to Domain Shift

Dataset	Domain	Model	Prompt	Δ
SQuAD	Wiki	94.9 ±0.2	94.8 ±0.1	−0.1
TextbookQA	Book	54.3 ±3.7	**66.8** ±2.9	+12.5
BioASQ	Bio	77.9 ±0.4	**79.1** ±0.3	+1.2
RACE	Exam	59.8 ±0.6	**60.7** ±0.5	+0.9
RE	Wiki	88.4 ±0.1	**88.8** ±0.2	+0.4
DuoRC	Movie	**68.9** ±0.7	67.7 ±1.1	−1.2
DROP	Wiki	**68.9** ±1.7	67.1 ±1.9	−1.8

Table 1: F1 mean and stddev for models trained on SQuAD and evaluated on out-of-domain datasets from the MRQA 2019 shared task. Prompt tuning tends to give stronger zero-shot performance than model tuning, especially on datasets with large domain shifts like TextbookQA.

Transfer between 2 paraphrase detection tasks

- train prompt tuning and model tuning solutions on one task and evaluate zero-shot on a closely related task
- **Quora Question Pairs:** detecting if two questions are duplicates
- **MRPC:** detecting if 2 sentences from news articles are paraphrases
- supports that model tuning may be over-parameterized and more prone to overfit the training task

Train	Eval	Tuning	Accuracy	F1
QQP	MRPC	Model	73.1 ±0.9	81.2 ±2.1
		Prompt	**76.3** ±0.1	**84.3** ±0.3
MRPC	QQP	Model	74.9 ±1.3	**70.9** ±1.2
		Prompt	**75.4** ±0.8	69.7 ±0.3

Prompt Ensembling

- more efficient at ensembling multiple adaptations of pre-trained language model
- **storage**: training N prompts on same task creates N separate models for a task but shares parameters
- **inference**: to process one example, execute a single forward pass with batch size N, replicating example across batch and varying the prompt, rather than computing forward passes of N different models

Dataset	Metric	Average	Best	Ensemble
BoolQ	acc.	91.1	91.3	**91.7**
CB	acc./F1	99.3 / 99.0	100.00 / 100.00	**100.0 / 100.0**
COPA	acc.	98.8	100.0	**100.0**
MultiRC	EM/F1$_a$	65.7 / 88.7	66.3 / 89.0	**67.1 / 89.4**
ReCoRD	EM/F1	92.7 / 93.4	92.9 / 93.5	**93.2 / 93.9**
RTE	acc.	92.6	**93.5**	**93.5**
WiC	acc.	76.2	76.6	**77.4**
WSC	acc.	95.8	**96.2**	**96.2**
SuperGLUE (dev)		90.5	91.0	**91.3**

Table 3: Performance of a five-prompt ensemble built from a single frozen T5-XXL model exceeds both the average and the best among the five prompts.

Interpretability of learned soft prompts

- compute nearest neighbors to each prompt token from frozen model's vocabulary
    - similarity metric: cosine distance between vocabulary embedding vector and prompt token representation
- prompts are learning word-like representations
    - top-5 nearest neighbors form tight semantic clusters
        - *{Technology, technology, Technologies, technologies}*
        - *{entirely, completely, totally, altogether}*

Prompt Tuning Key Takeaways

• simple but effective mechanism for learning “soft prompts” to condition frozen language models to perform specific downstream tasks

• soft prompts are learned through back-propagation and can be tuned to incorporate signals from any number of labeled examples

• as models exceed billions of parameters, prompt tuning closes the gap and matches the strong performance of model tuning (where all model weights are tuned)

• robust to domain transfer and enables efficient prompt ensembling

Parameter-Efficient Transfer Learning for NLP

Neil Houlsby 1 Andrei Giurgiu 1 * Stanisław Jastrze¸bski 2 * Bruna Morrone 1 Quentin de Laroussilhe 1
Andrea Gesmundo 1 Mona Attariyan 1 Sylvain Gelly

https://arxiv.org/abs/1902.00751

Introduction: Transfer Learning in NLP

• Definition: Transfer learning is a technique where a model is first trained on a large dataset (often for general purposes), and then fine-tuned on a smaller, task-specific dataset.

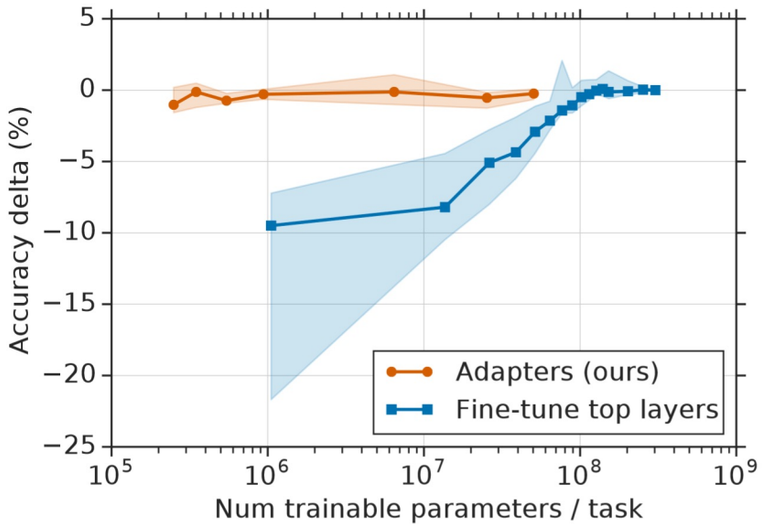
• Current Fine-Tuning Approach: **Full Fine-Tuning**: When adapting a pre-trained model to a new task, all of the model's parameters are fine-tuned for that task.

• Problem: **parameter inefficient**, large storage requirements, and redundant computations

• Goal: To develop a method that is **parameter-efficient** and can handle many downstream tasks with minimal additional training, while maintaining competitive performance.

Adapter-Based Tuning Approach

- The authors propose a solution to the above challenges using **adapter modules**, which are small, trainable components added to a pre-trained model.
- Adapter modules allow the model to **adapt** to new tasks with minimal additional parameters, making the approach efficient in terms of both memory and computation.
- Only **3.6%** of the model's parameters are newly trained!



What Are Adapter Modules?

• Definition: Adapter modules are **small trainable layers** that are inserted between the layers of a pre-trained neural network.

• They allow the **core model** to remain **frozen** while only a small number of task-specific parameters are trained.

• Key idea: these small modules “adapt” the pre-trained features to the specific requirements of new tasks, hence the name "adapter."

How Adapter Modules work:

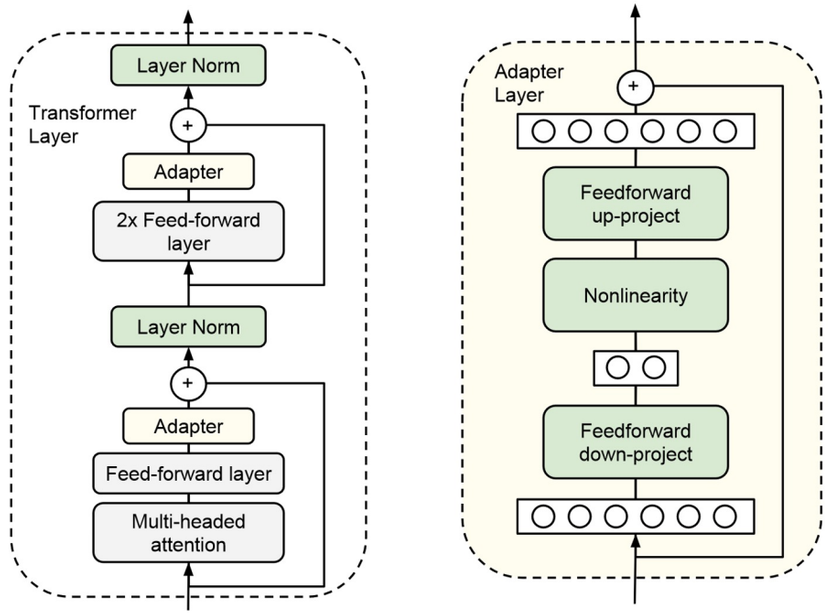
1) Placement 2) Training 3) Design

Adapter Module Architecture

• **Bottleneck Design**: The adapter module first projects the high-dimensional features down to a smaller latent space. After applying a nonlinearity, the features are then projected back up to the original dimension.

• **Skip Connection**: Adapter modules include skip connections across the layers.

• Adapter modules are added to the Transformer layers at two specific points: **After the multi-head attention mechanism** and **After the feed-forward layer**



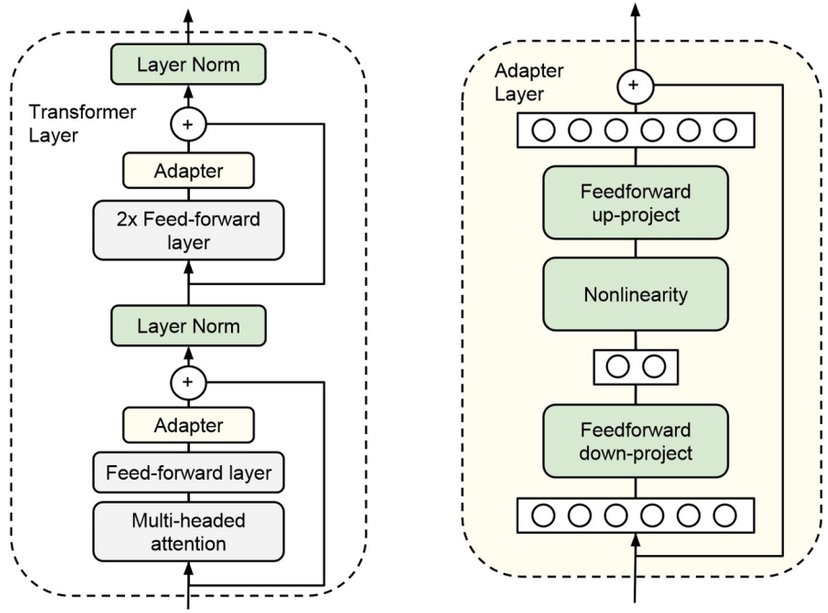
Adapter Module Architecture

• Consider a neural network with parameters $\boldsymbol{w}$: $\varphi_w(\boldsymbol{x})$.

• Feature-based transfer composes φ_w with a new function, χ_v, to yield $\chi_v(\varphi_w(\boldsymbol{x}))$. Only the new, task-specific, parameters, v, are then trained.

• Fine-tuning involves adjusting the original parameters, $\boldsymbol{w}$, for each new task.

• For adapter tuning, a new function, $\psi_{w,v}(\boldsymbol{x})$, is defined. The initial parameters $\boldsymbol{v}_0$ are set such that the new function resembles the original: $\psi_{w,v0}(\boldsymbol{x})\approx\varphi_w(\boldsymbol{x})$. During training, only $\boldsymbol{v}$ are tuned. If one chooses $|\boldsymbol{v}|\ll|\boldsymbol{w}|$, the resulting model requires $\sim|\boldsymbol{w}|$ parameters for many tasks.



Experiments: GLUE Benchmark

- The authors evaluate the effectiveness of adapter-based tuning on the GLUE benchmark by integrating adapter modules into the pre-trained **BERT LARGE** model
- The difference in accuracy between full fine-tuning and adapter-based tuning is within 0.4%, demonstrating that performance is preserved while drastically reducing the parameter overhead.

Parameter-Efficient Transfer Learning for NLP

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

Experiments: Additional Text Classification Tasks

- In addition to the GLUE benchmark, the authors evaluated the adapter-based tuning method on 17 publicly available text classification tasks.
- Training examples: from 900 to 330,000+
- Classes: from 2 to 157
- Average text length ranges from 57 to 1,900 characters

Dataset	No BERT baseline	$\text{BERT}_{\text{BASE}}$ Fine-tune	$\text{BERT}_{\text{BASE}}$ Variable FT	$\text{BERT}_{\text{BASE}}$ Adapters
20 newsgroups	91.1	92.8 ± 0.1	92.8 ± 0.1	91.7 ± 0.2
Crowdflower airline	84.5	83.6 ± 0.3	84.0 ± 0.1	84.5 ± 0.2
Crowdflower corporate messaging	91.9	92.5 ± 0.5	92.4 ± 0.6	92.9 ± 0.3
Crowdflower disasters	84.9	85.3 ± 0.4	85.3 ± 0.4	84.1 ± 0.2
Crowdflower economic news relevance	81.1	82.1 ± 0.0	78.9 ± 2.8	82.5 ± 0.3
Crowdflower emotion	36.3	38.4 ± 0.1	37.6 ± 0.2	38.7 ± 0.1
Crowdflower global warming	82.7	84.2 ± 0.4	81.9 ± 0.2	82.7 ± 0.3
Crowdflower political audience	81.0	80.9 ± 0.3	80.7 ± 0.8	79.0 ± 0.5
Crowdflower political bias	76.8	75.2 ± 0.9	76.5 ± 0.4	75.9 ± 0.3
Crowdflower political message	43.8	38.9 ± 0.6	44.9 ± 0.6	44.1 ± 0.2
Crowdflower primary emotions	33.5	36.9 ± 1.6	38.2 ± 1.0	33.9 ± 1.4
Crowdflower progressive opinion	70.6	71.6 ± 0.5	75.9 ± 1.3	71.7 ± 1.1
Crowdflower progressive stance	54.3	63.8 ± 1.0	61.5 ± 1.3	60.6 ± 1.4
Crowdflower US economic performance	75.6	75.3 ± 0.1	76.5 ± 0.4	77.3 ± 0.1
Customer complaint database	54.5	55.9 ± 0.1	56.4 ± 0.1	55.4 ± 0.1
News aggregator dataset	95.2	96.3 ± 0.0	96.5 ± 0.0	96.2 ± 0.0
SMS spam collection	98.5	99.3 ± 0.2	99.3 ± 0.2	95.1 ± 2.2
Average	72.7	73.7	74.0	73.3
Total number of params	—	$17\times$	$9.9\times$	$1.19\times$
Trained params/task	—	100%	52.9%	1.14%

Key Features of Adapter Modules

- **Parameter Efficiency**: Small Parameter Footprint, High Degree of Parameter Sharing
- **Extensibility**: Easily Scalable to New Tasks, No Interference with Previous Tasks
- **Compactness**: Bottleneck Architecture, Minimal Additional Overhead
- **Near State-of-the-Art Performance**: achieve within 0.4% of the full fine-tuned models' performance
- **Stability**: Skip Connections, Identity Initialization
- **Resource Efficiency**: Reduced Computational Cost
- **Flexibility in Architecture**: Flexible Placement, Customizable Size

LoRA: Low-Rank Adaptation of Large Language Models

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen

ICLR 2022

https://arxiv.org/abs/2106.09685

Background

- The idea behind Low-Rank Adaptation (LoRA) is built upon the observation that the weights learned by Large Language Models after training often contain redundancies.

- Therefore, instead of fine-tuning the entire set of weights in the LLM, we can streamline the process by focusing on a low-rank approximation of the weights — essentially, a smaller set of weights that eliminates these redundancies.

Problem Statement

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

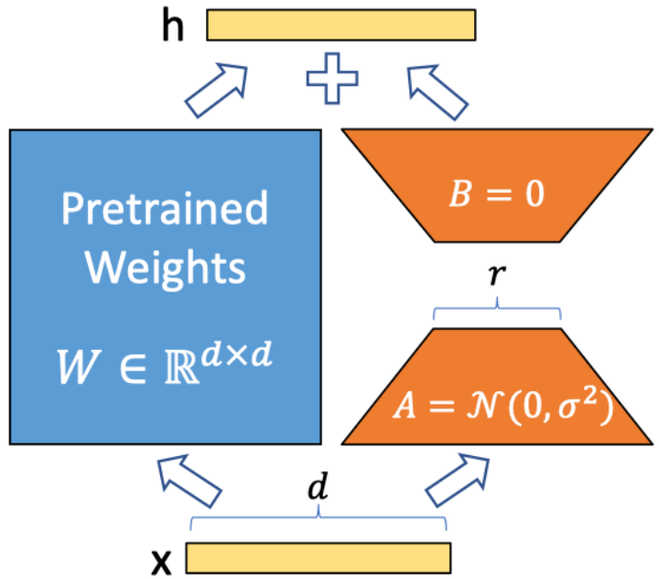
$$\max_{\Phi} \sum_{(x,y)\in\mathcal{Z}} \sum_{t=1}^{|y|} \log\left(P_{\Phi}(y_t|x, y_{<t})\right)$$

- In comparison, LoRA is a more parameter-efficient approach, where the task-specific parameter increment $\Delta\Phi = \Delta\Phi(\Theta)$ is further encoded by a much smaller-sized set of parameters Θ with $|\Theta| \ll |\Phi_0|$. The task of finding $\Delta\Phi$ thus becomes optimizing over Θ:

$$\max_{\Theta} \sum_{(x,y)\in\mathcal{Z}} \sum_{t=1}^{|y|} \log\left(p_{\Phi_0+\Delta\Phi(\Theta)}(y_t|x, y_{<t})\right)$$

LoRA: Low-Rank Adaptation

- By freezing the weights of the pre-trained model, LoRA performs a rank decomposition of the matrices for the incremental portion of the weights learned during the fine-tuning phase, and injects the rank-decomposed matrices A & B into each layer of the model's Transformer architecture.



Principle & Theoretical Foundation

- Suppose $W_0 \in \mathbb{R}^{d\times k}$ denotes the weight matrix in the neural network layer.

- Using regular backpropagation, we can obtain the weight update ΔW, which is usually computed as the negative gradient of the loss multiplied by the learning rate $\Delta W = \alpha\left(-\nabla L_{W_0}\right)$. T
- Then we can update W' by $W' = W_0 + \Delta W$.

- We can constrain this update by representing the latter with a low-rank decomposition

$W_0 + \Delta W = W_0 + BA$, where $B \in \mathbb{R}^{d\times r}, A \in \mathbb{R}^{r\times k}$, and the rank $r \ll \min(d, k)$.

Advantages of LoRA

- **A Generalization of Full Fine-tuning.** A more general form of fine-tuning allows the training of a subset of the pre-trained parameters. LoRA takes a step further and does not require the accumulated gradient update to weight matrices to have full-rank during adaptation.

- **No Additional Inference Latency.** When deployed in production, it can explicitly compute and store $\mathrm{W} = W_0 x + BAx$ and perform inference as usual. Note that both W_0 and BA are in $R_{d\times k}$. When we need to switch to another downstream task, we can recover W_0 by subtracting BA and then adding a different $B'A'$, a quick operation with very little memory overhead.

Comparison Results

Model & Method	# Trainable Parameters	MNLI	SST-2	MRPC	CoLA	QNLI	QQP	RTE	STS-B	Avg.
RoB_{base} (FT)*	125.0M	**87.6**	94.8	90.2	**63.6**	92.8	**91.9**	78.7	91.2	86.4
RoB_{base} (BitFit)*	0.1M	84.7	93.7	**92.7**	62.0	91.8	84.0	81.5	90.8	85.2
RoB_{base} (Adpt^{D})*	0.3M	$87.1_{\pm.0}$	$94.2_{\pm.1}$	$88.5_{\pm1.1}$	$60.8_{\pm.4}$	$93.1_{\pm.1}$	$90.2_{\pm.0}$	$71.5_{\pm2.7}$	$89.7_{\pm.3}$	84.4
RoB_{base} (Adpt^{D})*	0.9M	$87.3_{\pm.1}$	$94.7_{\pm.3}$	$88.4_{\pm.1}$	$62.6_{\pm.9}$	$93.0_{\pm.2}$	$90.6_{\pm.0}$	$75.9_{\pm2.2}$	$90.3_{\pm.1}$	85.4
RoB_{base} (LoRA)	0.3M	$87.5_{\pm.3}$	$\mathbf{95.1}_{\pm.2}$	$89.7_{\pm.7}$	$63.4_{\pm1.2}$	$\mathbf{93.3}_{\pm.3}$	$90.8_{\pm.1}$	$\mathbf{86.6}_{\pm.7}$	$\mathbf{91.5}_{\pm.2}$	**87.2**
$\text{RoB}_{\text{large}}$ (FT)*	355.0M	90.2	**96.4**	**90.9**	68.0	94.7	**92.2**	86.6	92.4	88.9
$\text{RoB}_{\text{large}}$ (LoRA)	0.8M	$\mathbf{90.6}_{\pm.2}$	$96.2_{\pm.5}$	$\mathbf{90.9}_{\pm1.2}$	$\mathbf{68.2}_{\pm1.9}$	$\mathbf{94.9}_{\pm.3}$	$91.6_{\pm.1}$	$\mathbf{87.4}_{\pm2.5}$	$\mathbf{92.6}_{\pm.2}$	**89.0**
$\text{RoB}_{\text{large}}$ (Adpt^{P})†	3.0M	$90.2_{\pm.3}$	$96.1_{\pm.3}$	$90.2_{\pm.7}$	$\mathbf{68.3}_{\pm1.0}$	$\mathbf{94.8}_{\pm.2}$	$\mathbf{91.9}_{\pm.1}$	$83.8_{\pm2.9}$	$92.1_{\pm.7}$	88.4
$\text{RoB}_{\text{large}}$ (Adpt^{P})†	0.8M	$\mathbf{90.5}_{\pm.3}$	$\mathbf{96.6}_{\pm.2}$	$89.7_{\pm1.2}$	$67.8_{\pm2.5}$	$\mathbf{94.8}_{\pm.3}$	$91.7_{\pm.2}$	$80.1_{\pm2.9}$	$91.9_{\pm.4}$	87.9
$\text{RoB}_{\text{large}}$ (Adpt^{H})†	6.0M	$89.9_{\pm.5}$	$96.2_{\pm.3}$	$88.7_{\pm2.9}$	$66.5_{\pm4.4}$	$94.7_{\pm.2}$	$92.1_{\pm.1}$	$83.4_{\pm1.1}$	$91.0_{\pm1.7}$	87.8
$\text{RoB}_{\text{large}}$ (Adpt^{H})†	0.8M	$90.3_{\pm.3}$	$96.3_{\pm.5}$	$87.7_{\pm1.7}$	$66.3_{\pm2.0}$	$94.7_{\pm.2}$	$91.5_{\pm.1}$	$72.9_{\pm2.9}$	$91.5_{\pm.5}$	86.4
$\text{RoB}_{\text{large}}$ (LoRA)†	0.8M	$\mathbf{90.6}_{\pm.2}$	$96.2_{\pm.5}$	$\mathbf{90.2}_{\pm1.0}$	$68.2_{\pm1.9}$	$\mathbf{94.8}_{\pm.3}$	$91.6_{\pm.2}$	$\mathbf{85.2}_{\pm1.1}$	$\mathbf{92.3}_{\pm.5}$	**88.6**
DeB_{XXL} (FT)*	1500.0M	91.8	**97.2**	92.0	72.0	**96.0**	92.7	93.9	92.9	91.1
DeB_{XXL} (LoRA)	4.7M	$\mathbf{91.9}_{\pm.2}$	$96.9_{\pm.2}$	$\mathbf{92.6}_{\pm.6}$	$\mathbf{72.4}_{\pm1.1}$	$\mathbf{96.0}_{\pm.1}$	$\mathbf{92.9}_{\pm.1}$	$\mathbf{94.9}_{\pm.4}$	$\mathbf{93.0}_{\pm.2}$	**91.3**

Table 2: $\text{RoBERTa}_{\text{base}}$, $\text{RoBERTa}_{\text{large}}$, and $\text{DeBERTa}_{\text{XXL}}$ with different adaptation methods on the GLUE benchmark. We report the overall (matched and mismatched) accuracy for MNLI, Matthew's correlation for CoLA, Pearson correlation for STS-B, and accuracy for other tasks. Higher is better for all metrics. * indicates numbers published in prior works. † indicates runs configured in a setup similar to Houlsby et al. (2019) for a fair comparison.

Comparison Results

Model & Method	# Trainable Parameters	E2E NLG Challenge BLEU	NIST	MET	ROUGE-L	CIDEr
GPT-2 M (FT)*	354.92M	68.2	8.62	46.2	71.0	2.47
GPT-2 M ($\text{Adapter}^{\text{L}}$)*	0.37M	66.3	8.41	45.0	69.8	2.40
GPT-2 M ($\text{Adapter}^{\text{L}}$)*	11.09M	68.9	8.71	46.1	71.3	2.47
GPT-2 M ($\text{Adapter}^{\text{H}}$)	11.09M	$67.3_{\pm.6}$	$8.50_{\pm.07}$	$46.0_{\pm.2}$	$70.7_{\pm.2}$	$2.44_{\pm.01}$
GPT-2 M (FT^{Top2})*	25.19M	68.1	8.59	46.0	70.8	2.41
GPT-2 M (PreLayer)*	0.35M	69.7	8.81	46.1	71.4	2.49
GPT-2 M (LoRA)	0.35M	$\mathbf{70.4}_{\pm.1}$	$\mathbf{8.85}_{\pm.02}$	$\mathbf{46.8}_{\pm.2}$	$\mathbf{71.8}_{\pm.1}$	$\mathbf{2.53}_{\pm.02}$
GPT-2 L (FT)*	774.03M	68.5	8.78	46.0	69.9	2.45
GPT-2 L ($\text{Adapter}^{\text{L}}$)	0.88M	$69.1_{\pm.1}$	$8.68_{\pm.03}$	$46.3_{\pm.0}$	$71.4_{\pm.2}$	$\mathbf{2.49}_{\pm.0}$
GPT-2 L ($\text{Adapter}^{\text{L}}$)	23.00M	$68.9_{\pm.3}$	$8.70_{\pm.04}$	$46.1_{\pm.1}$	$71.3_{\pm.2}$	$2.45_{\pm.02}$
GPT-2 L (PreLayer)*	0.77M	70.3	8.85	46.2	71.7	2.47
GPT-2 L (LoRA)	0.77M	$\mathbf{70.4}_{\pm.1}$	$\mathbf{8.89}_{\pm.02}$	$\mathbf{46.8}_{\pm.2}$	$\mathbf{72.0}_{\pm.2}$	$2.47_{\pm.02}$

Table 3: GPT-2 medium (M) and large (L) with different adaptation methods on the E2E NLG Challenge. For all metrics, higher is better. LoRA outperforms several baselines with comparable or fewer trainable parameters. Confidence intervals are shown for experiments we ran. * indicates numbers published in prior works.

Further Exploration

- Which weight matrices in transformers should we apply LoRA to?

	# 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.

Further Exploration

- What is the optimal rank **r** for LoRA?

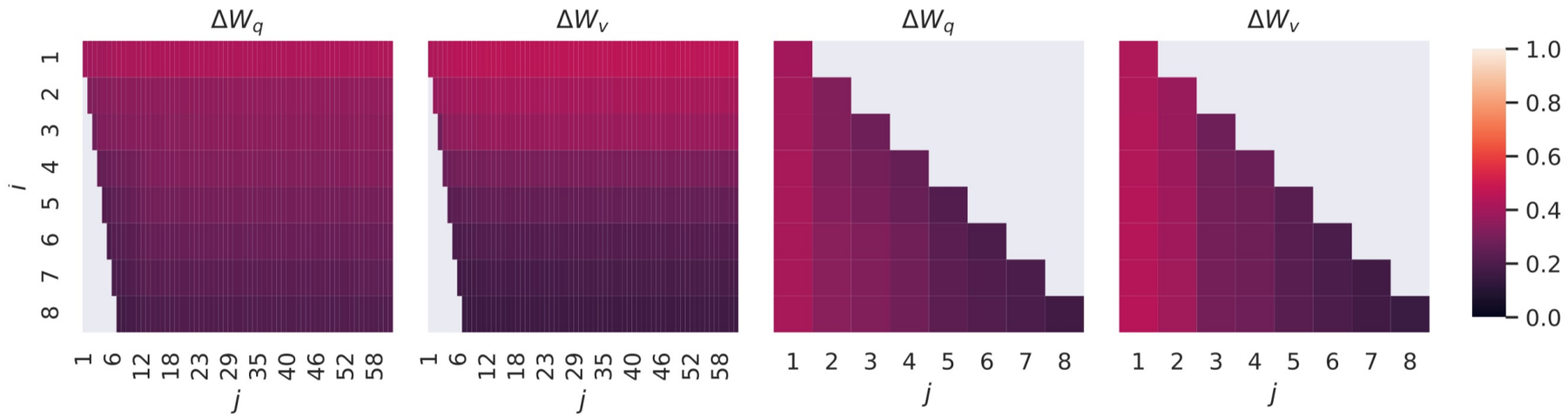


Figure 3: Subspace similarity between column vectors of $A_{r=8}$ and $A_{r=64}$ for both ΔW_q and ΔW_v. The third and the fourth figures zoom in on the lower-left triangle in the first two figures. The top directions in $r = 8$ are included in $r = 64$, and vice versa.

DoRA: Weight-Decomposed Low-Rank Adaptation

Shih-Yang Liu, Chien-Yi Wang, Hongxu Yin, Pavlo Molchanov, Yu-Chiang Frank Wang, Kwang-Ting Cheng, Min-Hung Chen

ICML 2024 (Oral)

https://arxiv.org/abs/2402.09353

Limitation of LoRA

- From the regression line for (ΔD, ΔM) of both DoRA and FT, a distinct negative slope characterizes DoRA and FT, instead of a clear positive correlation shown by LoRA.

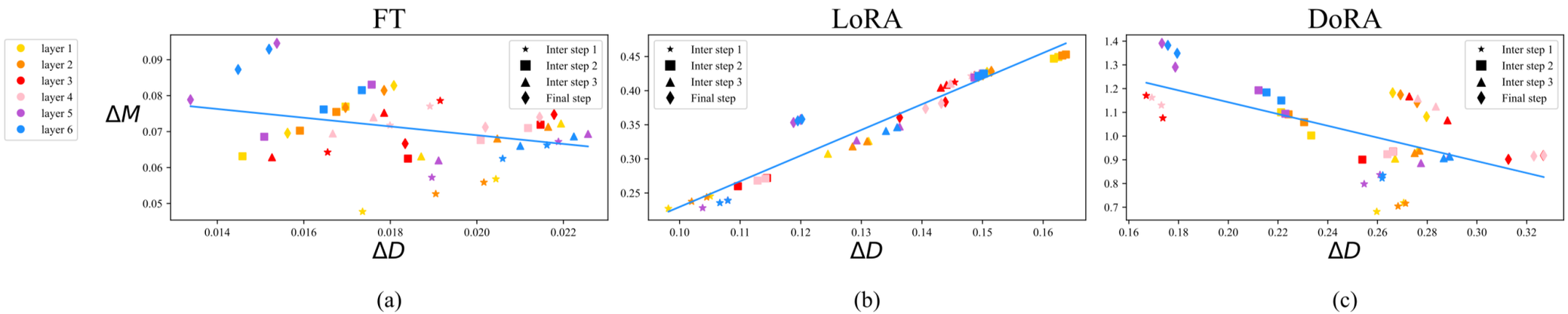


Figure 2. Magnitude and direction updates of (a) FT, (b) LoRA, and (c) DoRA of the query matrices across different layers and intermediate steps. Different markers represent matrices of different training steps and different colors represent the matrices of each layer.

DoRA: Weight-Decomposed Low-Rank Adaptation

- Weight-Decomposed LowRank Adaptation (DoRA) decomposes the pre-trained weight into two components, magnitude and direction, for fine-tuning, specifically employing LoRA for directional updates to efficiently minimize the number of trainable parameters.

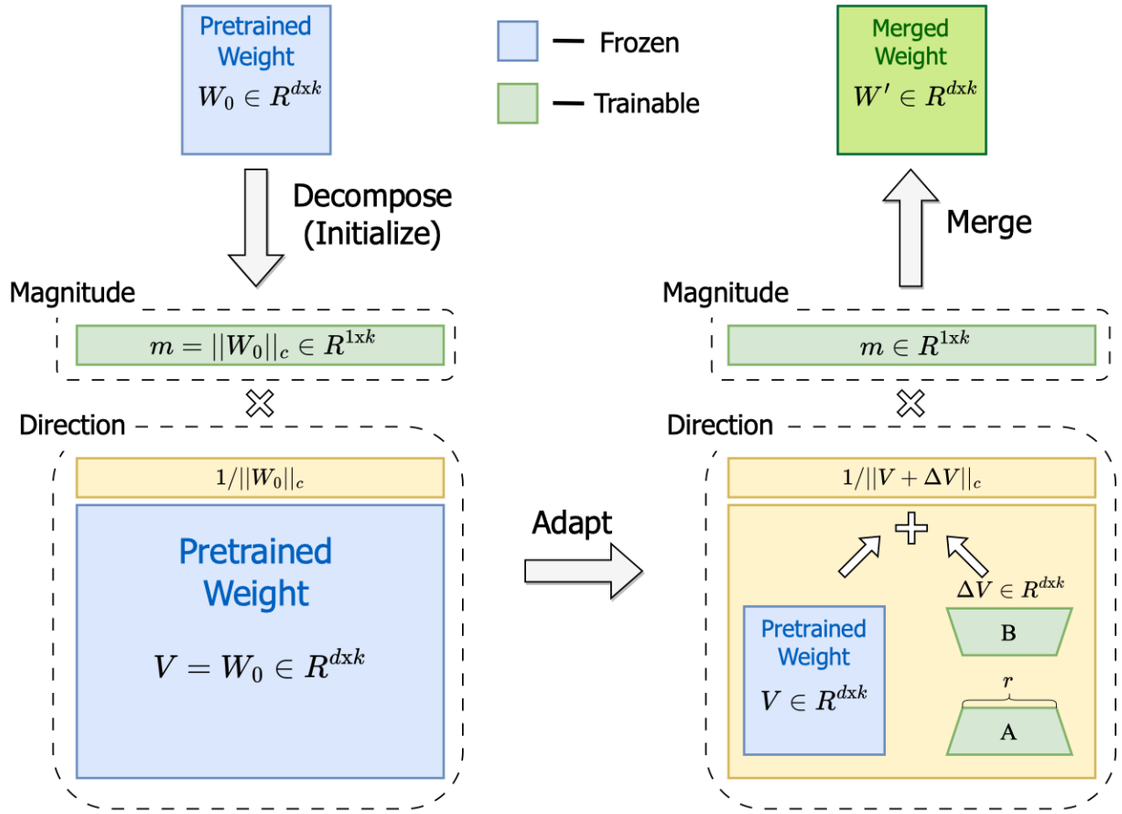


Figure 1. An overview of our proposed DoRA, which decomposes the pre-trained weight into *magnitude* and *direction* components for fine-tuning, especially with LoRA to efficiently update the direction component. Note that $|| \cdot ||_c$ denotes the vector-wise norm of a matrix across each column vector.

Principle & Theoretical Foundation

- The core idea of DoRA is to apply updates that not only adjust the magnitude of the model parameters but also carefully consider their direction.

- This is achieved by scaling the weight gradient and projecting it away from the current weight matrix.

- The math behind this involves two concepts: scaling by the norm of the update vector and a projection matrix that ensures orthogonality.

Principle & Theoretical Foundation

- The magnitude and directional variations between the pre-trained weight and full fine-tuned weight can be defined as follows:

$$\Delta M_{\mathrm{FT}}^{t} = \frac{\sum_{n=1}^{k} |m_{\mathrm{FT}}^{n,t} - m_{0}^{n}|}{k}$$

$$\Delta D_{\mathrm{FT}}^{t} = \frac{\sum_{n=1}^{k} (1 - \mathbf{cos}(V_{\mathrm{FT}}^{n,t}, W_{0}^{n}))}{k}$$

where $M_{FT}^{n,t}$ and M_{0}^{n} are the nth scalars in their respective magnitude vectors, $V_{FT}^{n,t}$ and W_{0}^{n} are the nth columns in V_{FT}^{t} and W_{0}.

Principle & Theoretical Foundation

- Recall the LoRA equation of:

$$W' = W_0 + \Delta W = W_0 + \underline{BA}$$

- Based on this, DoRA can be formulated as:

$$W' = \underline{m}\frac{V + \Delta V}{||V + \Delta V||_c} = \underline{m}\frac{W_0 + \underline{BA}}{||W_0 + \underline{BA}||_c}$$

Comparison Results

Table 1. Accuracy comparison of LLaMA 7B/13B, LLaMA2 7B, and LLaMA3 8B with various PEFT methods on eight commonsense reasoning datasets. Results of all the baseline methods on LLaMA 7B/13B are taken from (Hu et al., 2023). Results of LoRA on LLaMA2 7B and LLaMA3 8B are obtained using the hyperparameters described in (Hu et al., 2023). DoRA†: the adjusted version of DoRA with the rank halved.

Model	PEFT Method	# Params (%)	BoolQ	PIQA	SIQA	HellaSwag	WinoGrande	ARC-e	ARC-c	OBQA	Avg.
ChatGPT	-	-	73.1	85.4	68.5	78.5	66.1	89.8	79.9	74.8	77.0
LLaMA-7B	Prefix	0.11	64.3	76.8	73.9	42.1	72.1	72.9	54.0	60.6	64.6
	Series	0.99	63.0	79.2	76.3	67.9	75.7	74.5	57.1	72.4	70.8
	Parallel	3.54	67.9	76.4	78.8	69.8	78.9	73.7	57.3	75.2	72.2
	LoRA	0.83	68.9	80.7	77.4	78.1	78.8	77.8	61.3	74.8	74.7
	DoRA† (Ours)	0.43	70.0	82.6	79.7	83.2	80.6	80.6	65.4	77.6	**77.5**
	DoRA (Ours)	0.84	69.7	83.4	78.6	87.2	81.0	81.9	66.2	79.2	**78.4**
LLaMA-13B	Prefix	0.03	65.3	75.4	72.1	55.2	68.6	79.5	62.9	68.0	68.4
	Series	0.80	71.8	83	79.2	88.1	82.4	82.5	67.3	81.8	79.5
	Parallel	2.89	72.5	84.9	79.8	92.1	84.7	84.2	71.2	82.4	81.4
	LoRA	0.67	72.1	83.5	80.5	90.5	83.7	82.8	68.3	82.4	80.5
	DoRA† (Ours)	0.35	72.5	85.3	79.9	90.1	82.9	82.7	69.7	83.6	**80.8**
	DoRA (Ours)	0.68	72.4	84.9	81.5	92.4	84.2	84.2	69.6	82.8	**81.5**
LLaMA2-7B	LoRA	0.83	69.8	79.9	79.5	83.6	82.6	79.8	64.7	81.0	77.6
	DoRA† (Ours)	0.43	72.0	83.1	79.9	89.1	83.0	84.5	71.0	81.2	**80.5**
	DoRA (Ours)	0.84	71.8	83.7	76.0	89.1	82.6	83.7	68.2	82.4	**79.7**
LLaMA3-8B	LoRA	0.70	70.8	85.2	79.9	91.7	84.3	84.2	71.2	79.0	80.8
	DoRA† (Ours)	0.35	74.5	88.8	80.3	95.5	84.7	90.1	79.1	87.2	**85.0**
	DoRA (Ours)	0.71	74.6	89.3	79.9	95.5	85.6	90.5	80.4	85.8	**85.2**

Comparison Results

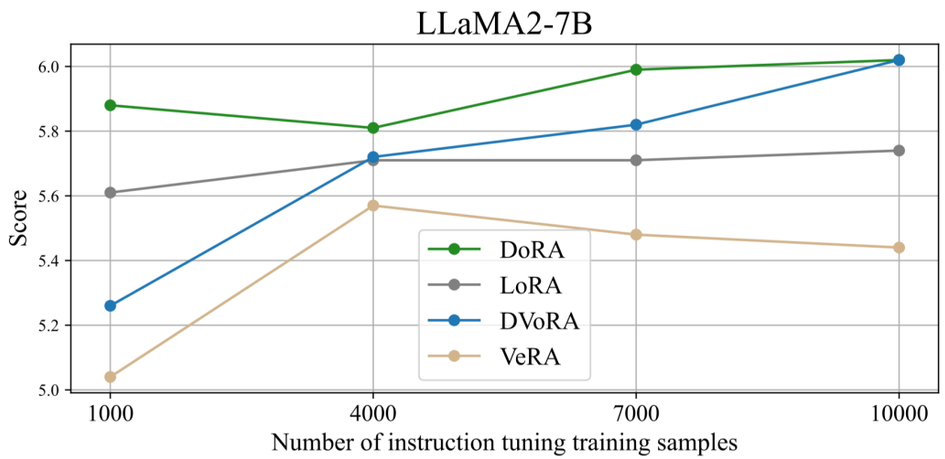


Figure 4. Performance of fine-tuned LLaMA2-7B on MT-Bench using different numbers of Alpaca training samples.

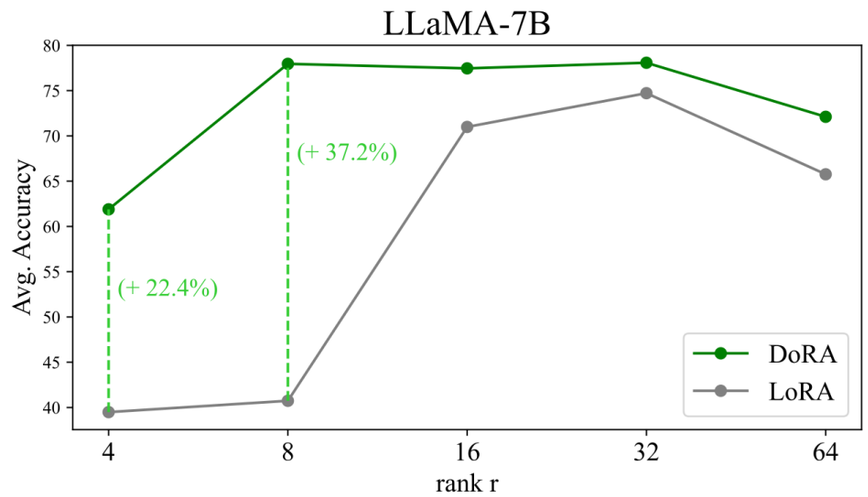


Figure 5. Average accuracy of LoRA and DoRA for varying ranks for LLaMA-7B on the commonsense reasoning tasks.

- DoRA outperforms LoRA in many different aspects across multiple tasks with different parameters.

Contribution

- 1. More detailed control: By targeting the magnitude and direction of the weights separately, DoRA provides more detailed control over the model fine-tuning process, which enables more accurate adaptation to specific task requirements.

- 2. Enhanced learning capability: DoRA's weight decomposition strategy enhances the model's ability to learn during the fine-tuning process, bringing its performance on multiple downstream tasks closer to that of a full-parameter fine-tuning approach.

Contribution

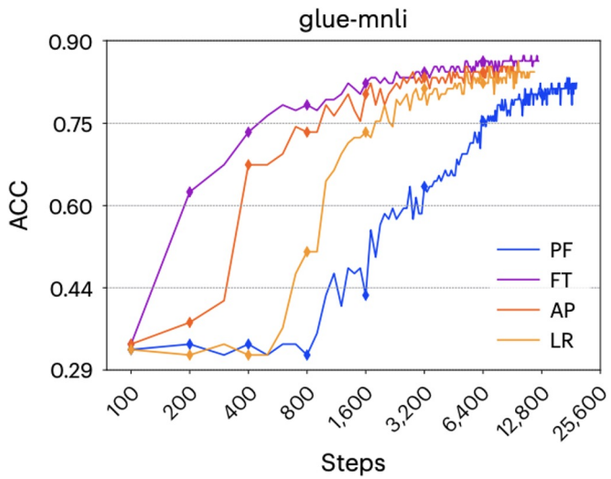
- 3. Maintaining efficiency: Despite its innovation in fine-tuning strategy, DoRA maintains the efficiency of LoRA and avoids adding extra reasoning burden.

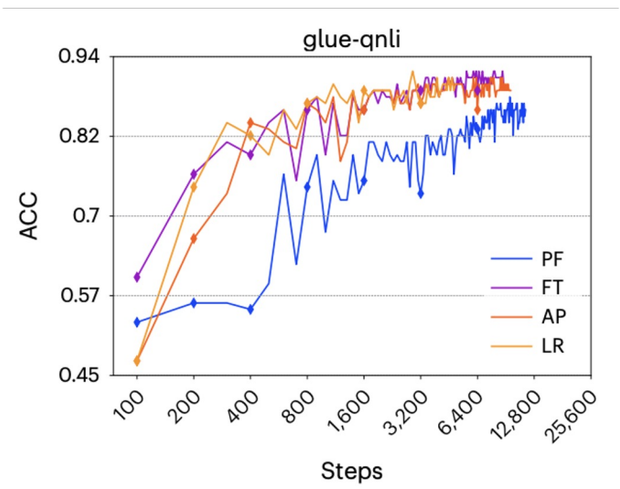
- 4. Improve training stability: DoRA improves the stability of the training process by decomposing the weights and using low-rank adaptation specifically for the directions, which helps avoid overfitting and other training problems.

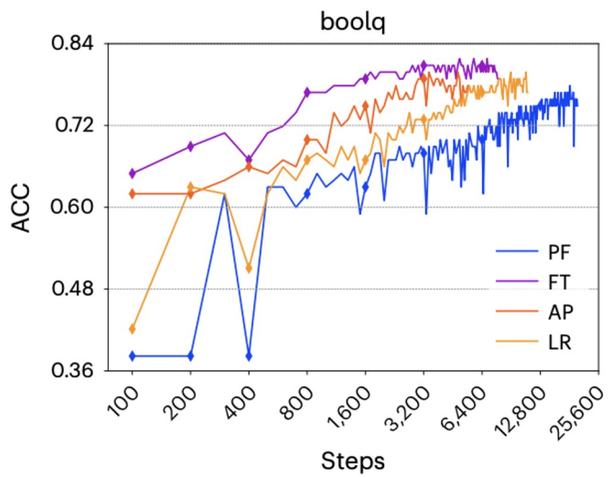
Comparison in PEFT

- Avg. performance: FT > LR > AP > PF > PT
- Convergence rate: FT > AP ≈ LR > PF
- Prompt tuning lags far behind other methods although easiest to implement

- FT: Fine Tuning
- LR: LoRA
- AP: Adapter
- PF: Prefix Tuning
- PT: Prompt Tuning







Ding et al. (2023) Parameter-efficient fine-tuning of large-scale pre-trained language models