LoRA: Low-Rank Adaptation of Large Language Models

Edward Hu[*] **Yelong Shen**[*] **Phillip Wallis**
Zeyuan Allen-Zhu **Yuanzhi Li** **Shean Wang** **Weizhu Chen**
Microsoft Corporation
`{edwardhu, yeshe, phwallis, zeyuana, yuanzhil`
`swang, wzchen}@microsoft.com`

Abstract

The dominant paradigm of natural language processing consists of large-scale pre-training on general domain data and adaptation to particular tasks or domains. As we pre-train larger models, conventional fine-tuning, which retrains all model parameters, becomes less feasible. Using GPT-3 175B as an example, deploying many independent instances of fine-tuned models, each with 175B parameters, is extremely expensive. We propose **Lo**w-**R**ank **A**daptation, or LoRA, which freezes the pre-trained model weights and injects trainable rank decomposition matrices into each layer of the Transformer architecture, greatly reducing the number of trainable parameters for downstream tasks. For GPT-3, LoRA can reduce the number of trainable parameters by 10,000 times and the computation hardware requirement by 3 times compared to full fine-tuning. LoRA performs on-par or better than fine-tuning in model quality on both GPT-3 and GPT-2, despite having fewer trainable parameters, a higher training throughput, and no additional inference latency. We also provide an empirical investigation into rank-deficiency in language model adaptations, which sheds light on the efficacy of LoRA. We release our implementation in GPT-2 at `https://github.com/microsoft/LoRA`.

1 Introduction

Many applications in natural language processing rely on adapting *one* large scale, pre-trained language model to *multiple* downstream applications. Such adaptation is usually done via *fine-tuning*, which updates all the parameters of the pre-trained model. The major downside of fine-tuning is that it requires storing as many parameters as in the original model. As larger models are trained every few months, this changes from a mere "inconvenience" for GPT-2 [32] or BERT-large [9] to a critical deployment challenge for GPT-3 [6] with 175 billion trainable parameters.[2]

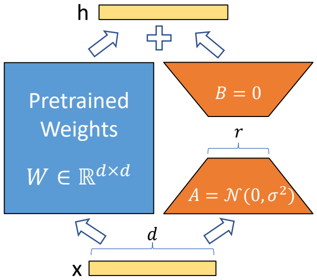
h
Pretrained Weights
$W \in \mathbb{R}^{d \times d}$
$B = 0$
r
$A = \mathcal{N}(0, \sigma^2)$
d
x

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

Many researchers sought to mitigate this by adapting only some parameters or learning external modules for new tasks. This way, we only need to load a small number of task-specific parameters to the pre-trained model for each task, which greatly boosts the deployment efficiency. However, in practice, existing techniques either introduce inference latency [15, 34] by extending model depth or reduce the

[*] Equal contribution.

[2] While GPT-3 175B achieves non-trivial performance with few-shot learning, fine-tuning boosts its performance significantly as shown in Appendix A.

Preprint. Under review.

model's usable sequence length [14, 19, 21, 25]. More importantly, these prior attempts sometimes fail to match the fine-tuning baselines, posing a trade-off between efficiency and model quality.

We take inspiration from [1, 20] which show that the learned over-parametrized models in fact reside on a low intrinsic dimension. We hypothesize that the update matrices in language model adaptation also have a low "intrinsic rank", leading to our proposed **Lo**w-**R**ank **A**daptation (LoRA) approach. LoRA allows us to train every dense layer in a neural network indirectly by injecting and optimizing rank decomposition matrices of the dense layer's update instead, while keeping the original matrices frozen, as shown in Fig. 1. Using GPT-3 175B as an example, we show that a very low rank (i.e., *r* in Fig. 1 can be one or two) suffices even when the full rank (i.e., *d*) is as high as 12288, making LoRA both space- and compute-efficient.

LoRA possesses several key advantages as follows.

- A single pre-trained model can be shared and used to build many small LoRA modules for different tasks. We can keep the shared original model in VRAM and efficiently switch tasks by replacing the matrices A and B in Fig. 1, which significantly reduces the storage requirement and task-switching overhead.
- It makes training more efficient and lowers the hardware barrier to entry by 3 times, since we do not need to calculate the gradients or maintain the optimizer states for most model parameters when using adaptive optimizers. Instead, we only optimize the injected low-rank matrices, which have much fewer parameters.
- Its simple linear design allows us to merge the update matrices with the original weights during deployment, introducing no inference latency.
- LoRA is orthogonal to prior techniques and can be combined with many of them (such as prefix-tuning). We provide an example in Appendix D.

Terminologies We make frequent references to the Transformer architecture and use the conventional terminologies for its dimensions. We call its hidden size, or the size of its activations, d_{model}. We use W_q, W_k, W_v, and W_o to refer to the query/key/value/output projection matrices in the self-attention module. W or W_0 refers to a pre-trained weight matrix and ΔW its update during adaptation. We use r to denote the rank of a LoRA module.

2 Problem Statement

While our proposal is agnostic to the training objective, we focus on language modeling as the primary use case. Below is a brief description of the language modeling problem and, in particular, the maximization of conditional probabilities given a task-specific prompt.

Suppose we are given a pre-trained autoregressive language model $p_\Phi(y|x)$ that is parametrized by Φ. For instance, $p_\Phi(y|x)$ can be a generic multi-task learner such as GPT-2 [32] or GPT-3 [6] based on the Transformer architecture [36]. Now, consider adapting this pre-trained model to (possibly multiple) downstream conditional text generation tasks, such as summarization, machine reading comprehension (MRC), and natural language to SQL (NL2SQL). Each downstream task is represented by a training dataset of context-target pairs: $\mathcal{Z} = \{(x_i, y_i)\}_{i=1,..,N}$, where both x_i and y_i are sequences of tokens. For example, in NL2SQL, x_i is a natural language query and y_i its equivalent SQL command; for summarization, x_i is the content of an article and y_i its short summary.

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

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

One of the main drawbacks for full fine-tuning is that for *each* downstream task, we learn a *different* set of parameters $\Delta\Phi$ whose dimensions $|\Delta\Phi|$ equals $|\Phi_0|$. Thus, if the pre-trained model is large (such as GPT-3 with $|\Phi_0| \approx 175$ Billion), storing and deploying many independent instances of fine-tuned models can be challenging, if at all plausible.

In this paper, we adopt a 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:

$$\max_{\Theta} \sum_{(x,y)\in\mathcal{Z}} \sum_{t=1}^{|y|} \log\left(p_{\Phi_0+\Delta\Phi(\Theta)}(y_t|x, y_{<t})\right) \tag{2}$$

As we shall see in the subsequent sections, we propose to use a low-rank representation to encode $\Delta\Phi$ that is both computational and memory efficient. When the pre-trained model is GPT-3, the size of trainable parameters $|\Theta|$ can be as small as 0.01% of $|\Phi_0|$.

3 Our Method

We describe the simple design of LoRA and its practical implications. The principles outlined here apply to any dense layers in deep learning models, though we only focus on certain weights in Transformers in our experiments for practical reasons.

Low-Rank Constraint on Update Matrices. A typical neural network contains numerous dense layers that perform matrix multiplication. The weight matrices in these layers are allowed to have full-rank. When adapting to a specific task, however, Aghajanyan et al. [1] shows that the pre-trained language models have a low "instrisic dimension" and can still learn efficiently despite a low-dimensional reparametrization. Inspired by this observation, we wonder if the updates to the weights also have a low "intrinsic rank" when adapting to downstream tasks. For a pre-trained weight matrix $W_0 \in \mathbb{R}^{d\times k}$, we constrain its update by representing it 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 $r \ll min(d, k)$. During training, W_0 is frozen and does not receive gradient updates, while A and B contain trainable parameters. Note both W_0 and $\Delta W = BA$ are multiplied with the same input, and their respective output vectors are summed coordinate-wise. For $h = W_0x$, our modified forward pass yields:

$$h = W_0x + \Delta Wx = W_0x + BAx \tag{3}$$

We illustrate our reparametrization in Fig. 1. We use a random Gaussian initialization for A and zero for B, so $\Delta W = BA$ is zero at the beginning of training. We then scale ΔWx by $\frac{1}{r}$ during training to keep the coordinates of ΔWx roughly $\Theta(1)$ in r after training [**?**].

Weight Decay to Pre-trained Weights. We note that weight decay behaves differently with LoRA than with full fine-tuning. Specifically, preforming the usual weight decay on A and B is similar to *decaying back to the pre-trained weights*, which has been studied as a potentially effective form of regularization against "catastrophic forgetting" [16, 18]. While extensive experiments isolating its effect is out-of-scope for this work, we believe that this, coupled with a constrained parameter space, might provide some regularization advantages. For example, see how LoRA with $r = d_{model} = 1024$ outperforms full fine-tuning on GPT-2 Medium in Sec. 5.3 and Sec. G.2.

No Additional Inference Latency. During deployment, we can explicitly compute $W = W_0 + BA$ and perform inference as usual. When we need to switch to another downstream task, we can recover W_0 by subtracting BA and then adding a different $B'A'$. This causes a minor increase in peak memory usage and adds a latency to model switching that does not exceed a single model forward pass. Critically, we do not introduce any additional latency during inference in return.

3.1 Applying LoRA to Transformer

In principle, we can apply LoRA to any subset of weight matrices in a neural network to reduce the number of trainable parameters. In the Transformer architecture, there are four weight matrices in the self-attention module (W_q, W_k, W_v, W_o) and two in the MLP module. We treat W_q (or W_k, W_v) as a single matrix of dimension $d_{model} \times d_{model}$, even though the output dimension is usually sliced into attention heads. We limit our study to **only changing the attention weights** for downstream tasks and freeze the MLP modules (so they are not trained in downstream tasks) since applying LoRA to the latter results in 4 times the number of trainable parameters given the same rank r. We further study the effect on adapting different types of weight matrices in a Transformer in Sec. 6.1.

Practical Benefits and Limitations. The most significant benefit comes from the reduction in memory and storage usage. For a Transformer, we reduce the VRAM consumption by $2/3$ if $r \ll d_{model}$ as we do not need to keep track of the optimizer states for the frozen parameters. We also reduce the checkpoint size by roughly $\frac{d_{model}}{2\gamma r}$ times, where γ is the fraction of weight matrices on which we apply LoRA. On GPT-3, our motivating use case, we reduce the VRAM consumption from 1.2TB to 350GB. With $r = 4$ and $\gamma = 1/6$, the checkpoint size is reduced by roughly 10,000× (from 350GB to 35MB)[3]. This allows us to train with significantly fewer GPUs and avoid I/O bottlenecks. Another benefit is that during deployment, we can switch between tasks at a much lower cost by only swapping the LoRA weights, often measured in megabytes, as opposed to all the weights (350GB). This allows for the creation of many customized models that can be activated and deactivated on the fly on machines that store the pre-trained weights. We also observe a 25% speedup during training as we do not need to calculate the gradient for the vast majority of the parameters.

On the other hand, LoRA has its limitations. For example, it is not straightforward to batch inputs to different tasks with different A and B in a single forward pass, because we absorb A and B into W to prevent additional inference latency.

4 Related Works

Transformer Language Models. Transformer [36] is a sequence-to-sequence architecture that makes heavy use of self-attention. [31] applied it to autoregressive language modeling by using a stack of Transformer decoders. Since then, Transformer-based language models have dominated NLP, achieving the state-of-the-art in many tasks. A new paradigm emerged with BERT [9] and GPT-2 [32] – both are large Transformer language models trained on a large amount of text – where fine-tuning on task-specific data after pre-training on general domain data provides a significant performance gain compared to training on task-specific data directly. Training larger Transformers generally results in better performance and remains an active research direction. GPT-3 [6] is the largest single Transformer language model trained to-date with 175B parameters.

Prompt Engineering and Fine-Tuning. While GPT-3 175B can adapt its behavior with just a few additional training examples, the result depends heavily on the input prompt [6]. This necessitates an empirical art of composing and formatting the prompt to maximize a model's performance on a desired task, which is known as prompt engineering or prompt hacking. Fine-tuning retrains a model pre-trained on general domains to a specific task [9, 31]. Variants of it include learning just a subset of the parameters [8, 9], yet practitioners often retrain all of them to maximize the downstream performance. However, the enormity of GPT-3 175B makes it challenging to perform fine-tuning in the usual way due to the large checkpoint it produces and the high hardware barrier to entry since it has the same memory footprint as pre-training.

Parameter-Efficient Adaptation. [15, 34] propose inserting *adapter* layers between existing layers in a neural network. Our method uses a bottleneck structure similar to [15] to impose a low-rank constraint on the weight updates. The key functional difference is that our learned weights can be merged with the main weights during inference, thus not introducing any latency, which is not the case for the adapter layers. More recently, [14, 19, 21, 25] proposed optimizing the input word embeddings in lieu of fine-tuning, akin to a continuous and differentiable generalization of prompt engineering. We include comparisons with [21] in our experiment section. However, this line of works can only scale up by using more special tokens in the prompt, which take up available sequence length for task tokens when positional embeddings are learned.

Low-Rank Structures in Deep Learning. Low-rank structure is very common in machine learning. A lot of machine learning problems have certain intrinsic low-rank structure [7, 13, 23, 24]. Moreover, it is known that for many deep learning tasks, especially those with a heavily over-parametrized neural network, the learned neural network will enjoy low-rank properties after training [29]. Some prior works even explicitly impose the low-rank constraint when training the original neural network, such as [17, 30, 35, 38, 39], however, to the best of our knowledge, none of these works considers low-rank update for *adaptation to downstream tasks*. In theory literature, it is known that neural networks outperform other classical learning methods, including the corresponding (finite-width) neural tangent kernels [5, 22] when the underlying concept class has certain low-rank structure [2, 3, 11]. Another

[3] We still need the 350GB model during deployment; however, storing 100 adapted models only requires 350GB + 35MB * 100 ≈ 354GB as opposed to 100 * 350GB ≈ 35TB.

theoretical result in [4] suggests that low-rank adaptations can be useful for adversarial training. In sum, we believe that our proposed low-rank adaptation update is well-motivated by the literature.

5 Empirical Experiments

We benchmark the downstream performance of LoRA on both GPT-2 and GPT-3. Specifically, we focus on WikiSQL [40] (natural language to SQL queries), MultiNLI [37] (natural language inference) [4], and SAMSum [12] (conversation summarization) for GPT-3.For a direct comparison with [21] on GPT-2, we follow their setup and use Table-to-Text E2E NLG Challenge [28], a language generation task. See Appendix B for more details on these datasets. We also compare with the other tuning methods such as [21, 25]. We use NVIDIA Tesla V100 GPUs for all of our experiments. For GPT-3, we use 96 GPUs for fine-tuning and 24 GPUs for LoRA. We use a single GPU for GPT-2 experiments.

5.1 Baselines

We compare with the following baselines. Note that we implemented both prefix-embedding and prefix-layer tuning in GPT-3 on our own according to [21], while the non-LoRA baselines all come from the implementation of [21].

Fine-Tuning is the de facto default adaptation approach, where the model is initialized to the pre-trained parameters. During fine-tuning, all the model parameters undergo gradient updates according to Eqn. 1. A simple variant of it is to update only some layers while freezing others. We include one such baseline reported in prior work [21], which tunes just the last two layers (FT-Top2).

Bias only is a baseline where we only train the bias vectors when adapting to downstream tasks.

Prefix-embedding tuning injects special tokens alongside the input tokens. These special tokens have trainable word embeddings and are generally not in the model's vocabulary. The placement of such tokens can have an impact on performance. We focus on "prefixing", which prepends these tokens to the prompt, and "infixing", which appends to the prompt; both are discussed in [21]. We use l_p (resp. l_i) denote the number of prefix (resp. infix) tokens. The number of trainable parameters is $|\Theta| = d_{model} \times (l_p + l_i)$.

Prefix-layer tuning is an extension to prefix-embedding tuning. Instead of just learning the word embeddings (or equivalently, the activations after the embedding layer) for some special tokens, we learn the activations after every layer. The activations computed from previous layers are simply replaced by our trainable ones. The resulting number of trainable parameters is $|\Theta| = L \times d_{model} \times (l_p + l_i)$, where L is the number of layers (96 for GPT-3).

Adapter tuning [15] inserts adapter layers between the self-attention module (and the MLP module) and the subsequent residual connection. There are two weight matrices in an adapter layer with a nonlinearity in between. The dimension of the hidden layer in an adapter layer determines the total number of trainable parameters. For a hidden size of n, we have $|\Theta| = 4 \times L \times d_{model} \times n$.

LoRA adds trainable pairs of rank decomposition matrices to existing weight matrices. As mentioned in Sec. 3.1 and supported by our analysis in Sec. 6.1, we only apply LoRA to W_q and W_v in our experiments. The number of trainable parameters is determined by the rank r and the shape of the original weights: $|\Theta| = 2 \times L \times d_{model} \times r$.

5.2 Performance on GPT-3

Hyperparameters For all of our GPT-3 experiments, we train using AdamW [26] for 2 epochs with a batch size of 100k tokens, a sequence length of 768, and a weight decay factor of 0.1. We tune the learning rate for all method-dataset combinations. See Sec. C.1 for more details. For prefix-embedding tuning, we find the optimal l_p and l_i to be 256 and 8, respectively, totalling $3.2M$ trainable parameters. We use $l_p = 8$ and $l_i = 8$ in PrefixLayer with $20.2M$ trainable parameters to obtain the overall best performance. We present two configurations for LoRA: one with $r_q = r_v = 4$

[4] Recent works treat MultiNLI-matched and MultiNLI-mismatched as separate tasks. In this work, we only report accuracy on MultiNLI-matched validation set.

Method	# of Trainable Parameters	WikiSQL	MNLI-m	SAMSum
		Accuracy (%)	Accuracy (%)	R1/R2/RL
GPT-3 175B (Fine-Tune)	175,255.8M	73.0	89.5	52.0/28.0/44.5
GPT-3 175B (Bias Only)	14.2M	71.3	91.0	51.3/27.4/43.5
GPT-3 175B (PrefixEmbed)	3.2M	63.1	88.6	48.3/24.2/40.5
GPT-3 175B (PrefixLayer)	20.2M	70.1	89.5	50.8/27.3/43.5
GPT-3 175B (LoRA)	4.7M	73.4	91.3	52.1/28.3/44.3
GPT-3 175B (LoRA)	37.7M	**73.8**	**91.7**	**53.2/29.2/45.0**

Table 1: Logical form validation accuracy on WikiSQL, validation accuracy on MultiNLI-matched and Rouge-1/2/L on SAMSum achieved by different GPT-3 adaptation methods. LoRA performs better than prior approaches, including conventional fine-tuning. The result on WikiSQL has a fluctuation of $\pm 0.3\%$ and MNLI-m $\pm 0.1\%$.

(18.8M) and one with $r_q = r_v = 8$ (37.7M).

As shown in Table 1, on all three datasets, LoRA outperforms the fine-tuning baseline. It might appear that prefix-embedding tuning could benefit from having more trainable parameters since it uses much fewer parameters than other methods. Nonetheless, this is not the case as shown in Fig. 2. We observe a significant performance drop when we use more than 256 special tokens for prefix-embedding tuning or more than 32 special tokens for prefix-layer tuning. This corroborates the similar observation reported in [21]. While a thorough investigation into this phenomenon is out-of-scope for this work, we suspect that having more special tokens causes the input distribution to shift further away from the pre-training data distribution, degrading the model performance. Separately, we investigate the performance of adaptation approaches in the low-data regime in Sec. E.3.

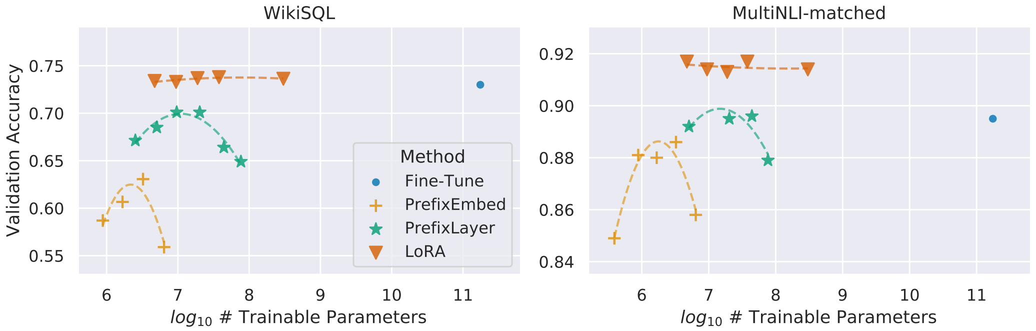

Figure 2: GPT-3 175B validation accuracy vs. number of trainable parameters of several adaptation methods on WikiSQL and MNLI-matched. LoRA enjoys better scalability and task performance. See Sec. E.1 for more details on the plotted data points.

5.3 Performance on GPT-2

Having established LoRA as a competitive method on GPT-3, we hope to answer if LoRA can also work on less overly parametrized models, such as GPT-2 medium or large [32]. We keep our setup as close as possible to [21] for a direct comparison. Due to space constraint, we only present our result on E2E NLG Challenge (Table 2) in this section – see Sec. E.2 for results on more datasets.

Hyperparameters We train all of our GPT-2 models using AdamW [26] and a linear learning rate schedule for 5 epochs. We use the batch size, learning rate, and beam search beam size described in [21]. Accordingly, we also tune these hyperparameters for LoRA, and we include a list of the used hyperparameters in Table 8.

Method	# of Trainable Parameters	E2E				
		BLEU	NIST	MET	ROUGE-L	CIDEr
GPT-2 M (Fine-Tune)	354.92M	68.2	8.62	46.2	71.0	2.47
GPT-2 M (Adapter)	11.48M	68.9	8.71	46.1	71.3	2.47
GPT-2 M (FT-Top2)	25.19M	68.1	8.59	46.0	70.8	2.41
GPT-2 M (Prefix)	0.35M	69.7	8.81	46.1	71.4	2.49
GPT-2 M (LoRA)	0.35M	**70.4**	**8.85**	**46.8**	**71.8**	**2.53**
GPT-2 L (Fine-Tune)	774.03M	68.5	8.78	46.0	69.9	2.45
GPT-2 L (Prefix)	0.77M	70.3	8.85	46.2	71.7	**2.47**
GPT-2 L (LoRA)	0.77M	**70.4**	**8.89**	**46.8**	**72.0**	**2.47**

Table 2: 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.

6 Understanding the Low-Rank Updates

Given the empirical advantage of LoRA, we hope to further explain the properties of the low-rank adaptation learned from downstream tasks. Note that the low-rank structure not only lowers the hardware barrier to entry which allows us to run multiple experiments in parallel, but also gives better interpretability of how the update weights are correlated with the pre-trained weights.

We perform a sequence of empirical studies to answer the following questions: 1) Given a parameter budget constraint, *which subset of weight matrices* in a pre-trained Transformer should we adapt to maximize downstream performance? 2) Is the "optimal" adaptation matrix ΔW *really rank-deficient*? If so, what is a good rank to use in practice? 3) What is the connection between ΔW and W? Does ΔW highly correlate with W? How large is ΔW comparing to W?

We believe that our answers to question (2) and (3) shed light on the fundamental principles of using pre-trained language models for downstream tasks, which is a critical topic in NLP.

6.1 Which Weight Matrices in Transformer Should We Apply LoRA to?

Given a limited parameter budget, which types of weights should we adapt with LoRA to obtain the best performance on downstream tasks? As mentioned in Sec. 3.1, we only consider weight matrices in the self-attention module. We set a parameter budget of 18M (roughly 35MB) on GPT-3, which corresponds to $r = 8$ if we adapt one type of weights or $r = 4$ if we adapt two types, for all 96 layers. The result is presented in Table 3.

	# of Trainable Parameters = 18M					
Weight Type	W_q	W_k	W_v	W_o	W_q, W_k	W_q, W_v
Rank r	8	8	8	8	4	4
WikiSQL ($\pm 0.3\%$)	70.4	70.0	73.0	73.2	71.4	**73.7**
MultiNLI ($\pm 0.1\%$)	91.0	90.8	91.0	**91.3**	**91.3**	**91.3**

Table 3: 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.

Note that putting all the parameters in ΔW_q or ΔW_k results in significantly lower performance, while adapting both W_q and W_v yields the best result. This suggests that even a rank of four captures enough information in ΔW such that it is preferable to adapt more weight matrices than adapting a single type of weights with a larger rank.

6.2 What is the Optimal Rank r for LoRA?

We turn our attention to the effect of rank r on model performance. We adapt both W_q and W_v since they performed the best in the previous experiment and as well as W_q alone for comparison.

	Weight Type	$r = 1$	$r = 2$	$r = 4$	$r = 8$	$r = 64$
WikiSQL($\pm 0.3\%$)	W_q, W_v	73.4	73.3	**73.7**	**73.8**	73.5
	W_q	68.8	69.6	**70.5**	**70.4**	70.0
MultiNLI ($\pm 0.1\%$)	W_q, W_v	91.3	91.4	91.3	**91.7**	91.4

Table 4: Validation accuracy on WikiSQL and MultiNLI with different rank r. To our surprise, a rank as small as one suffices for adapting both W_q and W_v on these datasets while training W_q alone needs a larger r. We replicate this on GPT-2 in Sec. G.2.

Table 4 shows that, surprisingly, LoRA already performs competitively with a very small r (more so for W_{q+v} than W_q). This suggests the update matrix ΔW could have a very small "intrinsic rank".[5]

To further support this finding, we check the overlap of the subspaces learned by different choices of r and by different random seeds. We argue that increasing r does not cover more meaningful subspaces, which suggests that a low-rank adaptation matrix is sufficient.

Subspace similarity between different r. Given $A_{r=8}$ and $A_{r=64}$ which are the learned adaptation matrices with rank $r = 8$ and 64 using the *same pre-trained model*, we perform singular value decomposition and obtain the right-singular unitary matrices $U_{A_{r=8}}$ and $U_{A_{r=64}}$.[6] We hope to answer: how much of the subspace spanned by the top i singular vectors in $U_{A_{r=8}}$ (for $1 \leq i \leq 8$) is contained in the subspace spanned by top j singular vectors of $U_{A_{r=64}}$ (for $1 \leq j \leq 64$)? We measure this quantity with a normalized subspace similarity based on the Grassmann distance (See Appendix F for a more formal discussion)

$$\phi(A_{r=8}, A_{r=64}, i, j) = \frac{||U_{A_{r=8}}^{i\top} U_{A_{r=64}}^{j}||_F^2}{\min(i, j)} \in [0, 1] \tag{4}$$

where $U_{A_{r=8}}^{i}$ represents the columns of $U_{A_{r=8}}$ corresponding to the top-i singular vectors.

$\phi(\cdot)$ has a range of $[0, 1]$, where 1 represents a complete overlap of subspaces and 0 a complete separation. See Fig. 3 for how ϕ changes as we vary i and j. We only look at the 48th layer (out of 96) due to space constraint, but the conclusion holds for other layers as well, as shown in Sec. G.1.

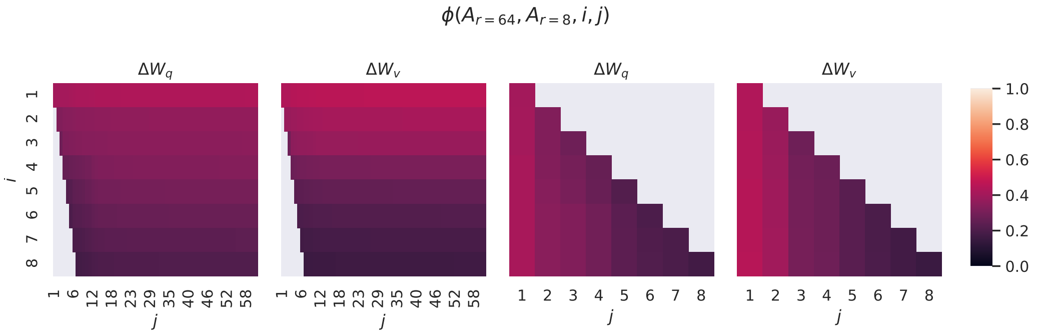

Figure 3: Subspace similarity between column vectors of $A_{r=8}$ and $A_{r=64}$ for both ΔW_q and ΔW_v. The third and the fourth figures zoom in on the lower-left triangle in the first two figures. The top directions in $r = 8$ are included in $r = 64$, and vice versa.

[5] However, we do not expect a small r to work for every task or dataset. Consider the following thought experiment: if the downstream task were in a different language than the one used for pre-training, retraining the entire model (similar to LoRA with $r = d_{model}$) could certainly outperform LoRA with a small r.

[6] Note that a similar analysis can be carried out with B and the left-singular unitary matrices – we stick with A for our experiments.

We make an *important observation* from Fig. 3.

> Directions corresponding to the top singular vector overlap significantly between $A_{r=8}$ and $A_{r=64}$, while others do not. Specifically, ΔW_v (resp. ΔW_q) of $A_{r=8}$ and ΔW_v (resp. ΔW_q) of $A_{r=64}$ share a subspace of dimension 1 with normalized similarity > 0.5, providing an explanation of why $r = 1$ performs quite well in our downstream tasks for GPT-3.

Since both $A_{r=8}$ and $A_{r=64}$ are learned using the same pre-trained model, Fig. 3 indicates that the top singular-vector directions of $A_{r=8}$ and $A_{r=64}$ are the most useful, while other directions potentially contain mostly random noises accumulated during training. Hence, the adaptation matrix can indeed have a very low rank.

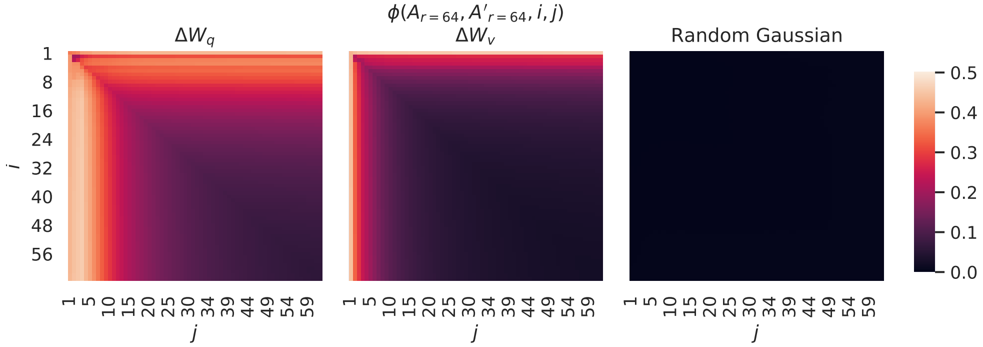

Figure 4: **Left and Middle:** Normalized subspace similarity between the column vectors of $A_{r=64}$ from two random seeds, for both ΔW_q and ΔW_v in the 48-th layer. **Right:** the same heat-map between the column vectors of two random Gaussian matrices. See Sec. G.1 for other layers.

Subspace similarity between different random seeds. We further confirm this by plotting the normalized subspace similarity between two randomly seeded runs with $r = 64$, shown in Fig. 4. ΔW_q appears to have a higher "intrinsic rank" than ΔW_v, since more common singular value directions are learned by both runs for ΔW_q, which is in line with our empirical observation in Table 4. As a comparison, we also plot two random Gaussian matrices, which do not share any common singular value directions with each other.

6.3 How Does the Adaptation Matrix ΔW Compare to W?

We further investigate the relationship between ΔW and W. In particular, does ΔW highly correlate with W? (Or mathematically, is ΔW mostly contained in the top singular directions of W?) Also, how "large" is ΔW comparing to its corresponding directions in W? This can shed light on the underlying mechanism for adapting pre-trained language models.

To answer these questions, we project W onto the r-dimensional subspace of ΔW by computing $U^\top W V^\top$, with U/V being the left/right singular-vector matrix of ΔW. Then, we compare the Frobenius norm between $\|U^\top W V^\top\|_F$ and $\|W\|_F$. As a comparison, we also compute $\|U^\top W V^\top\|_F$ by replacing U, V with the top r singular vectors of W or a random matrix.

	$r = 4$			$r = 64$		
	ΔW_q	W_q	Random	ΔW_q	W_q	Random
$\|\|U^\top W_q V^\top\|\|_F =$	0.32	21.67	0.02	1.90	37.71	0.33
$\|\|W_q\|\|_F = 61.95$	$\|\|\Delta W_q\|\|_F = 6.91$			$\|\|\Delta W_q\|\|_F = 3.57$		

Table 5: The Frobenius norm of $U^\top W_q V^\top$ where U and V are the left/right top r singular vector directions of either (1) ΔW_q, (2) W_q, or (3) a random matrix. The weight matrices are taken from the 48th layer of GPT-3.

We draw *several conclusions* from Table 5. First, ΔW has a stronger correlation with W compared to a random matrix, indicating that ΔW amplifies some features that are already in W. Second, instead of repeating the top singular directions of W, ΔW only *amplifies directions that are not emphasized in* W. Third, the amplification factor is rather huge: $21.5 \approx 6.91/0.32$ for $r = 4$. See Sec. G.4 for why $r = 64$ has a smaller amplification factor. We also provide a visualization in Sec. G.3 for how the correlation changes as we include more top singular directions from W_q. This suggests that the low-rank adaptation matrix potentially *amplifies the important features for specific downstream tasks that were learned but not emphasized in the general pre-training model.*

7 Conclusion and Future Work

Fine-tuning enormous language models is prohibitively expensive in terms of both the hardware requirement and the storage/switching cost for hosting multiple instances. We propose LoRA, an efficient adaptation strategy that neither introduces inference latency nor reduces input sequence length while retaining model quality. Importantly, it allows for quick task-switching when deployed as a service by sharing the vast majority of the model parameters. While we focused on Transformer, the proposed principles are generally applicable to any neural networks with dense layers.

LoRA can potentially work in tandem with other fine-tuning techniques. In the future, we hope to explore only tuning some layers or adding adversarial training. Finally, the rank-deficiency of ΔW suggests that W could be rank-deficient as well, which might inspire lots of future works.

Acknowledgments

We thank in alphabetical order Jianfeng Gao, Jade Huang, Jiayuan Huang, Lisa Xiang Li, Xiaodong Liu, Yabin Liu, Benjamin Van Durme, Luis Vargas, Haoran Wei, Peter Welinder, and Greg Yang for providing valuable feedback.

References

[1] Armen Aghajanyan, Luke Zettlemoyer, and Sonal Gupta. Intrinsic Dimensionality Explains the Effectiveness of Language Model Fine-Tuning. *arXiv:2012.13255 [cs]*, December 2020. URL http://arxiv.org/abs/2012.13255.

[2] Zeyuan Allen-Zhu and Yuanzhi Li. What Can ResNet Learn Efficiently, Going Beyond Kernels? In *NeurIPS*, 2019. Full version available at http://arxiv.org/abs/1905.10337.

[3] Zeyuan Allen-Zhu and Yuanzhi Li. Backward feature correction: How deep learning performs deep learning. *arXiv preprint arXiv:2001.04413*, 2020.

[4] Zeyuan Allen-Zhu and Yuanzhi Li. Feature purification: How adversarial training performs robust deep learning. *arXiv preprint arXiv:2005.10190*, 2020.

[5] Zeyuan Allen-Zhu, Yuanzhi Li, and Zhao Song. A convergence theory for deep learning via over-parameterization. In *ICML*, 2019. Full version available at http://arxiv.org/abs/1811.03962.

[6] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language Models are Few-Shot Learners. *arXiv:2005.14165 [cs]*, July 2020. URL http://arxiv.org/abs/2005.14165.

[7] Jian-Feng Cai, Emmanuel J Candès, and Zuowei Shen. A singular value thresholding algorithm for matrix completion. *SIAM Journal on optimization*, 20(4):1956–1982, 2010.

[8] Ronan Collobert and Jason Weston. A unified architecture for natural language processing: deep neural networks with multitask learning. In *Proceedings of the 25th international conference*

on Machine learning, ICML '08, pages 160–167, New York, NY, USA, July 2008. Association for Computing Machinery. ISBN 978-1-60558-205-4. doi: 10.1145/1390156.1390177. URL https://doi.org/10.1145/1390156.1390177.

[9] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv:1810.04805 [cs]*, May 2019. URL http://arxiv.org/abs/1810.04805. arXiv: 1810.04805.

[10] Claire Gardent, Anastasia Shimorina, Shashi Narayan, and Laura Perez-Beltrachini. The webnlg challenge: Generating text from rdf data. In *Proceedings of the 10th International Conference on Natural Language Generation*, pages 124–133, 2017.

[11] Behrooz Ghorbani, Song Mei, Theodor Misiakiewicz, and Andrea Montanari. When do neural networks outperform kernel methods? *arXiv preprint arXiv:2006.13409*, 2020.

[12] Bogdan Gliwa, Iwona Mochol, Maciej Biesek, and Aleksander Wawer. Samsum corpus: A human-annotated dialogue dataset for abstractive summarization. *CoRR*, abs/1911.12237, 2019. URL http://arxiv.org/abs/1911.12237.

[13] Lars Grasedyck, Daniel Kressner, and Christine Tobler. A literature survey of low-rank tensor approximation techniques. *GAMM-Mitteilungen*, 36(1):53–78, 2013.

[14] Karen Hambardzumyan, Hrant Khachatrian, and Jonathan May. WARP: Word-level Adversarial ReProgramming. *arXiv:2101.00121 [cs]*, December 2020. URL http://arxiv.org/abs/2101.00121. arXiv: 2101.00121.

[15] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-Efficient Transfer Learning for NLP. *arXiv:1902.00751 [cs, stat]*, June 2019. URL http://arxiv.org/abs/1902.00751.

[16] Yen-Chang Hsu, Yen-Cheng Liu, Anita Ramasamy, and Zsolt Kira. Re-evaluating Continual Learning Scenarios: A Categorization and Case for Strong Baselines. *arXiv:1810.12488 [cs]*, January 2019. URL http://arxiv.org/abs/1810.12488. arXiv: 1810.12488.

[17] Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. Speeding up convolutional neural networks with low rank expansions. *arXiv preprint arXiv:1405.3866*, 2014.

[18] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. Overcoming catastrophic forgetting in neural networks. *arXiv:1612.00796 [cs, stat]*, January 2017. URL http://arxiv.org/abs/1612.00796. arXiv: 1612.00796.

[19] Brian Lester, Rami Al-Rfou, and Noah Constant. The Power of Scale for Parameter-Efficient Prompt Tuning. *arXiv:2104.08691 [cs]*, April 2021. URL http://arxiv.org/abs/2104.08691. arXiv: 2104.08691.

[20] Chunyuan Li, Heerad Farkhoor, Rosanne Liu, and Jason Yosinski. Measuring the Intrinsic Dimension of Objective Landscapes. *arXiv:1804.08838 [cs, stat]*, April 2018. URL http://arxiv.org/abs/1804.08838. arXiv: 1804.08838.

[21] Xiang Lisa Li and Percy Liang. Prefix-Tuning: Optimizing Continuous Prompts for Generation. *arXiv:2101.00190 [cs]*, January 2021. URL http://arxiv.org/abs/2101.00190.

[22] Yuanzhi Li and Yingyu Liang. Learning overparameterized neural networks via stochastic gradient descent on structured data. In *Advances in Neural Information Processing Systems*, 2018.

[23] Yuanzhi Li, Yingyu Liang, and Andrej Risteski. Recovery guarantee of weighted low-rank approximation via alternating minimization. In *International Conference on Machine Learning*, pages 2358–2367. PMLR, 2016.

[24] Yuanzhi Li, Tengyu Ma, and Hongyang Zhang. Algorithmic regularization in over-parameterized matrix sensing and neural networks with quadratic activations. In *Conference On Learning Theory*, pages 2–47. PMLR, 2018.

[25] Xiao Liu, Yanan Zheng, Zhengxiao Du, Ming Ding, Yujie Qian, Zhilin Yang, and Jie Tang. GPT Understands, Too. *arXiv:2103.10385 [cs]*, March 2021. URL `http://arxiv.org/abs/2103.10385`. arXiv: 2103.10385.

[26] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[27] Linyong Nan, Dragomir Radev, Rui Zhang, Amrit Rau, Abhinand Sivaprasad, Chiachun Hsieh, Xiangru Tang, Aadit Vyas, Neha Verma, Pranav Krishna, et al. Dart: Open-domain structured data record to text generation. *arXiv preprint arXiv:2007.02871*, 2020.

[28] Jekaterina Novikova, Ondřej Dušek, and Verena Rieser. The e2e dataset: New challenges for end-to-end generation. *arXiv preprint arXiv:1706.09254*, 2017.

[29] Samet Oymak, Zalan Fabian, Mingchen Li, and Mahdi Soltanolkotabi. Generalization guarantees for neural networks via harnessing the low-rank structure of the jacobian. *arXiv preprint arXiv:1906.05392*, 2019.

[30] Daniel Povey, Gaofeng Cheng, Yiming Wang, Ke Li, Hainan Xu, Mahsa Yarmohammadi, and Sanjeev Khudanpur. Semi-orthogonal low-rank matrix factorization for deep neural networks. In *Interspeech*, pages 3743–3747, 2018.

[31] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving Language Understanding by Generative Pre-Training. page 12, .

[32] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language Models are Unsupervised Multitask Learners. page 24, .

[33] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[34] Sylvestre-Alvise Rebuffi, Hakan Bilen, and Andrea Vedaldi. Learning multiple visual domains with residual adapters. *arXiv:1705.08045 [cs, stat]*, November 2017. URL `http://arxiv.org/abs/1705.08045`. arXiv: 1705.08045.

[35] Tara N Sainath, Brian Kingsbury, Vikas Sindhwani, Ebru Arisoy, and Bhuvana Ramabhadran. Low-rank matrix factorization for deep neural network training with high-dimensional output targets. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 6655–6659. IEEE, 2013.

[36] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 6000–6010, 2017.

[37] Adina Williams, Nikita Nangia, and Samuel Bowman. A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1112–1122, New Orleans, Louisiana, June 2018. Association for Computational Linguistics. doi: 10.18653/v1/N18-1101. URL `https://www.aclweb.org/anthology/N18-1101`.

[38] Yu Zhang, Ekapol Chuangsuwanich, and James Glass. Extracting deep neural network bottleneck features using low-rank matrix factorization. In *2014 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 185–189. IEEE, 2014.

[39] Yong Zhao, Jinyu Li, and Yifan Gong. Low-rank plus diagonal adaptation for deep neural networks. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5005–5009. IEEE, 2016.

[40] Victor Zhong, Caiming Xiong, and Richard Socher. Seq2sql: Generating structured queries from natural language using reinforcement learning. *CoRR*, abs/1709.00103, 2017. URL `http://arxiv.org/abs/1709.00103`.

A Large Language Models Still Need Parameter Updates

Few-shot learning, or prompt engineering, is very advantageous when we only have a handful of training samples. However, in practice, we can often afford to curate a few thousand or more training examples for performance-sensitive applications. As shown in Table 6, fine-tuning improves the model performance drastically compared to few-shot learning on datasets large and small. We take the GPT-3 few-shot result on RTE from the GPT-3 paper [6]. For MNLI-m task, we use two demonstrations per class or six in-context examples in total.

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

Table 6: Fine-tuning significantly outperforms few-shot learning on GPT-3 [6].

B Dataset Details

MultiNLI is a natural language inference dataset [37], consisting of $392,702$ training and $9,815$ validation examples. For pre-processing, we encode the context $x = \{[premise] :$ $, \text{premise}, [hypothesis] :, \text{hypothesis}\}$; target $y \in \{entailment, neutral, contradiction\}$. The dataset is mostly released under the OANC's license, and subsets of it are under other permissive licenses such as the Creative Commons Share-Alike 3.0 Unported License.

WikiSQL is introduced in [40] and contains $56,355/8,421$ training/validation examples. The task is to generate SQL queries from natural language questions and table schemata. We encode context as $x = \{\text{table schema}, \text{query}\}$ and target as $y = \{\text{SQL}\}$. The dataset is release under the BSD 3-Clause License.

E2E NLG Challenge was first introduced in [28] as a dataset for training end-to-end, data-driven natural language generation systems and is commonly used for data-to-text evaluation. The E2E dataset consists of $42K$ training, $4.6K$ validation, and $4.6K$ test examples from the restaurant domain. Each source table used as input can have multiple references. Each sample input (x, y) consists of a sequence of slot-value pairs, along with a corresponding natural language reference text. The dataset is released under Creative Commons BY-NC-SA 4.0.

DART is an open-domain data-to-text dataset described in [27]. DART inputs are structured as sequences of ENTITY | RELATION | ENTITY triples. With $82K$ examples in total, DART is a significantly larger and more complex data-to-text task compared to E2E. The dataset is released under the MIT license.

WebNLG is another commonly used dataset for data-to-text evaluation [10]. With $22K$ examples in total WebNLG comprises 14 distinct categories, nine of which are seen during training. Since five of the 14 total categories are not seen during training, but are represented in the test set, evaluation is typically broken out by "seen" categories (S), "unseen" categories (U) and "all" (A). Each input example is represented by a sequence of SUBJECT | PROPERTY | OBJECT triples. The dataset is released under Creative Commons BY-NC-SA 4.0.

C Hyperparameters Used in Experiments

C.1 GPT-3 Experiments

The training hyperparameters used in our GPT-3 experiments are listed in Table 7.

C.2 GPT-2 LoRA

The hyperparameters used for LoRA in GPT-2 are listed in Table 8. For those used for other baselines, see [21].

Hyperparameters	Fine-Tune	Prefix-Embed	Prefix-Tune	LoRA
Optimizer	AdamW			
Batch Size	128			
# Epoch	2			
Warmup Tokens	250,000			
Learning Rate Schedule	Linear			
Learning Rate	5.00E-06	5.00E-04	1.00E-04	2.00E-04

Table 7: The training hyperparameters used for different GPT-3 adaption methods. We use the same hyperparameters for all datasets.

Dataset	E2E	WebNLG	DART
	Training		
Optimizer	AdamW		
Weight Decay	0.01	0.01	0.0
Dropout Prob	0.1	0.1	0.0
Batch Size	8		
# Epoch	5		
Warmup Steps	500		
Learning Rate Schedule	Linear		
Label Smooth	0.1	0.1	0.0
Learning Rate	0.0002		
Adaptation	$r_q = r_v = 4$		
LoRA α	32		
	Inference		
Beam Size	10		
Length Penalty	0.9	0.8	0.8
no repeat ngram size	4		

Table 8: The hyperparameters for GPT-2 LoRA on E2E, WebNLG and DART.

D Combining LoRA with Prefix Tuning

LoRA can be naturally combined with existing prefix-based approaches. In this section, we evaluate two combinations of LoRA and variants of prefix-tuning on WikiSQL and MNLI.

LoRA+PrefixEmbed (LoRA+PE) combines LoRA with prefix-embedding tuning, where we insert $l_p + l_i$ special tokens whose embeddings are treated as trainable parameters. For more on prefix-embedding tuning, see Sec. 5.1.

LoRA+PrefixLayer (LoRA+PL) combines LoRA with prefix-layer tuning. We also insert $l_p + l_i$ special tokens; however, instead of letting the hidden representations of these tokens evolve naturally, we replace them after every Transformer block with an input agnostic vector. Thus, both the embeddings and subsequent Transformer block activations are treated as trainable parameters. For more on prefix-layer tuning, see Sec. 5.1.

In Table 9, we show the evaluation results of LoRA+PE and LoRA+PL on WikiSQL and MultiNLI. First of all, LoRA+PE significantly outperforms both LoRA and prefix-embedding tuning on WikiSQL, which indicates that LoRA is somewhat orthogonal to prefix-embedding tuning. On MultiNLI, the combination of LoRA+PE doesn't perform better than LoRA, possibly because LoRA on its own already achieves performance comparable to the human baseline. Secondly, we notice that LoRA+PL performs slightly worse than LoRA even with more trainable parameters. We attribute this to the fact that prefix-layer tuning is very sensitive to the choice of learning rate and thus makes the optimization of LoRA weights more difficult in LoRA+PL.

E Additional Task-Based Experiments

E.1 Additional Experiments on GPT-3

We present additional runs on GPT-3 with different adaptation methods in Table 9. The focus is on identifying the trade-off between performance and the number of trainable parameters.

Method	Hyperparameters	# Trainable Parameters	WikiSQL	MNLI-m
Fine-Tune	-	175B	73.0	89.5
PrefixEmbed	$l_p = 32, l_i = 8$	0.39 M	55.9	84.9
	$l_p = 64, l_i = 8$	0.88 M	58.7	88.1
	$l_p = 128, l_i = 8$	1.67 M	60.6	88.0
	$l_p = 256, l_i = 8$	3.24 M	63.1	88.6
	$l_p = 512, l_i = 8$	6.40 M	55.9	85.8
PrefixLayer	$l_p = 2, l_i = 2$	5.06 M	68.5	89.2
	$l_p = 8, l_i = 0$	10.1 M	69.8	88.2
	$l_p = 8, l_i = 8$	20.2 M	70.1	89.5
	$l_p = 32, l_i = 4$	44.1 M	66.4	89.6
	$l_p = 64, l_i = 0$	76.1 M	64.9	87.9
LoRA	$r_v = 2$	4.7 M	73.4	**91.7**
	$r_q = r_v = 1$	4.7 M	73.4	91.3
	$r_q = r_v = 2$	9.4 M	73.3	91.4
	$r_q = r_v = 4$	18.8 M	73.7	91.3
	$r_q = r_v = 8$	37.7 M	73.8	**91.7**
	$r_q = r_v = 64$	301.9 M	73.6	91.4
LoRA+PE	$r_q = r_v = 8, l_p = 8, l_i = 4$	37.8 M	75.0	91.4
	$r_q = r_v = 32, l_p = 8, l_i = 4$	151.1 M	**75.9**	91.1
	$r_q = r_v = 64, l_p = 8, l_i = 4$	302.1 M	**76.2**	91.3
LoRA+PL	$r_q = r_v = 8, l_p = 8, l_i = 4$	52.8 M	72.9	90.2

Table 9: Hyperparameter analysis of different adaptation approaches on WikiSQL and MNLI. Both prefix-embedding tuning (PrefixEmbed) and prefix-layer tuning (PrefixLayer) perform worse as we increase the number of trainable parameters, while LoRA's performance stabilizes. Performance is measured in validation accuracy.

E.2 Additional Experiments on GPT-2

We also repeat our experiment on DART [27] and WebNLG [10] following the setup of [21]. The result is shown in Table 10. Similar to our result on E2E NLG Challenge, reported in Sec. 5, LoRA performs better than or at least on-par with prefix-based approaches given the same number of trainable parameters.

E.3 Low-Data Regime

To evaluate the performance of different adaptation approaches in the low-data regime. we randomly sample 100, 1k and 10k training examples from the full training set of MNLI to form the low-data MNLI-n tasks. In Table 12, we show the performance of different adaptation approaches on MNLI-n. To our surprise, PrefixEmbed and PrefixLayer performs very poorly on MNLI-100 dataset, with PrefixEmbed performing only slightly better than random chance (37.6% vs. 33.3%). PrefixLayer performs better than PrefixEmbed but is still significantly worse than Fine-Tune or LoRA on MNLI-100. The gap between prefix-based approaches and LoRA/Fine-tuning becomes smaller as we increase the number of training examples, which might suggest that prefix-based approaches are not suitable for low-data tasks in GPT-3. LoRA achieves better performance than fine-tuning on both MNLI-100 and MNLI-Full, and comparable results on MNLI-1k and MNLI-10K considering the

Method	# Trainable Parameters	DART		
		BLEU↑	MET↑	TER↓
		GPT-2 Medium		
Fine-Tune	354M	46.0(±0.1)	**0.39**	**0.46**
Adapter	10M	45.4(±0.1)	**0.38**	**0.46**
FT-Top2	24M	38.1(±0.3)	0.34	0.56
Prefix	0.35M	45.7(±0.2)	**0.38**	**0.46**
LoRA	0.35M	**47.1**(±0.2)	**0.39**	**0.46**
		GPT-2 Large		
Fine-Tune	774M	46.5(±0.1)	**0.39**	0.45
Prefix	0.77M	46.5(±0.2)	**0.38**	**0.45**
LoRA	0.77M	**47.5**(±0.1)	**0.39**	**0.45**

Table 10: GPT-2 with different adaptation methods on DART. The variances of MET and TER are around 0.01 for all different adaption approaches.

Method	WebNLG								
	BLEU↑			MET↑			TER↓		
	U	S	A	U	S	A	U	S	A
	GPT-2 Medium								
Fine-Tune (354M)	30.4(±.5)	**63.2**(±.3)	47.6(±.4)	.32	**.45**	.39	.69	.34	.50
Adapter (10M)	**47.9**(±.2)	61.1(±.4)	**55.2**(±.3)	**.38**	.43	**.41**	**.45**	.35	**.39**
FT-Top2 (24M)	13.7(±.6)	50.1(±.4)	33.5 (±.4)	.16	.35	.26	1.03	.52	.75
Prefix (0.35M)	44.1(±.2)	**63.1**(±.1)	54.4(±.1)	**.37**	**.45**	**.41**	.50	**.34**	.41
LoRA (0.35M)	46.7(±.4)	62.1(±.2)	**55.3**(±.2)	**.38**	**.44**	**.41**	**.46**	**.33**	**.39**
	GPT-2 Large								
Fine-Tune (774M)	41.7(±.5)	**64.6**(±.4)	54.2(±.4)	.37	**.46**	**.42**	.56	.33	.43
Prefix (0.77M)	47.0(±.2)	64.2(±.4)	56.4(±.1)	**.39**	**.45**	**.42**	.49	**.33**	.40
LoRA (0.77M)	**48.4**(±.3)	64.0(±.3)	**57.0**(±.1)	**.39**	**.45**	**.42**	**.45**	**.32**	**.38**

Table 11: GPT-2 with different adaptation methods on WebNLG. The variances of MET and TER are around 0.01 variance for all different adaption approaches. "U" indicates unseen categories, "S" indicates seen categories, and "A" indicates all categories in the test set of webNLG.

(±0.3) variance due to random seeds.

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**

Table 12: Validation accuracy of different methods on subsets of MNLI using GPT-3 175B. MNLI-n describes a subset with n training examples. We evaluate with the full validation set. LoRA performs exhibits favorable sample-efficiency compared to other methods, including fine-tuning.

The training hyperparameters of different adaptation approaches on MNLI-n are reported in Table 13. We use a smaller learning rate for PrefixLayer on the MNLI-100 set, as the training loss does not decrease with a larger learning rate.

Hyperparameters	Adaptation	MNLI-100	MNLI-1k	MNLI-10K	MNLI-392K
Optimizer	-	AdamW			
Warmup Tokens	-	250,000			
LR Schedule	-	Linear			
Batch Size	-	20	20	100	128
# Epoch	-	40	40	4	2
Learning Rate	FineTune	5.00E-6			
	PrefixEmbed	2.00E-04	2.00E-04	4.00E-04	5.00E-04
	PrefixLayer	5.00E-05	5.00E-05	5.00E-05	1.00E-04
	LoRA	2.00E-4			
Adaptation-Specific	PrefixEmbed l_p	16	32	64	256
	PrefixEmbed l_i	8			
	PrefixTune	$l_p = l_i = 8$			
	LoRA	$r_q = r_v = 8$			

Table 13: The hyperparameters used for different GPT-3 adaptation methods on MNLI(m)-n.

F Measuring Similarity Between Subspaces

In this paper we use the measure $\phi(A, B, i, j) = \psi(U_A^i, U_B^j) = \frac{\|U_A^{i\top} U_B\|_F^2}{\min\{i,j\}}$ to measure the subspace similarity between two column orthonormal matrices $U_A^i \in \mathbb{R}^{d \times i}$ and $U_B^j \in \mathbb{R}^{d \times j}$, obtained by taking columns of the left singular matrices of A and B. We point out that this similarity is simply a reverse of the standard Projection Metric that measures distance between subspaces [**?**].

To be concrete, let the singular values of $U_A^{i\top} U_B^j$ to be $\sigma_1, \sigma_2, \cdots, \sigma_p$ where $p = \min\{i, j\}$. We know that the Projection Metric [**?**] is defined as:

$$d(U_A^i, U_B^j) = \sqrt{p - \sum_{i=1}^{p} \sigma_i^2} \in [0, \sqrt{p}]$$

where our similarity is defined as:

$$\phi(A, B, i, j) = \psi(U_A^i, U_B^j) = \frac{\sum_{i=1}^{p} \sigma_i^2}{p} = \frac{1}{p}\left(1 - d(U_A^i, U_B^j)^2\right)$$

This similarity satisfies that if U_A^i and U_B^j share the same column span, then $\phi(A, B, i, j) = 1$. If they are completely orthogonal, then $\phi(A, B, i, j) = 0$. Otherwise, $\phi(A, B, i, j) \in (0, 1)$.

G Additional Experiments on Low-Rank Matrices

We present additional results from our investigation into the low-rank update matrices.

G.1 Correlation between LoRA Modules

See Fig. 5 and Fig. 6 for how the results presented in Fig. 3 and Fig. 4 generalize to other layers.

G.2 Effect of r on GPT-2

We repeat our experiment on the effect of r (Sec. 6.2) in GPT-2. Using the E2E NLG Challenge dataset as an example, we report the validation loss and test metrics achieved by different choices of r after training for 26,000 steps. We present our result in Table 14. The optimal rank for GPT-2 Medium is between 4 and 16 depending on the metric used, which is similar to that for GPT-3 175B. Note that the relationship between model size and the optimal rank for adaptation is still an open question.

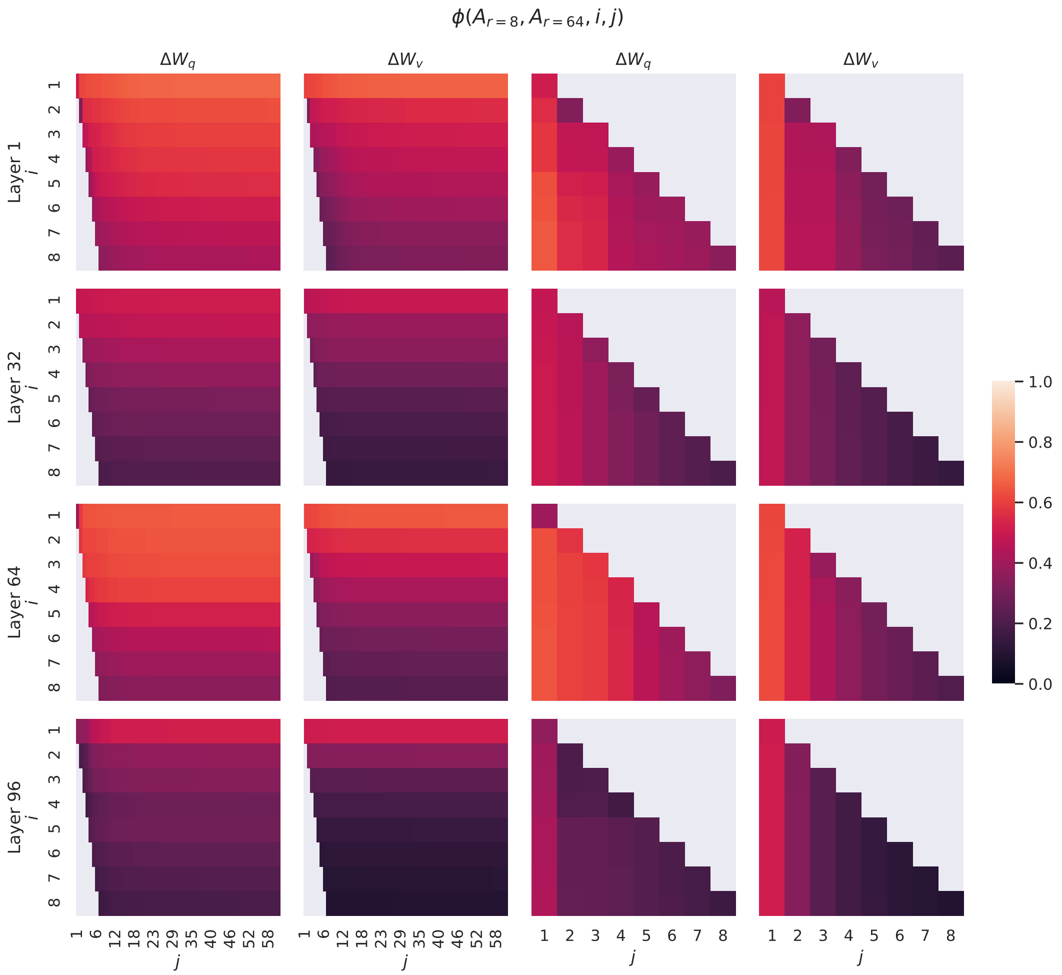

Figure 5: Normalized subspace similarity between the column vectors of $A_{r=8}$ and $A_{r=64}$ for both ΔW_q and ΔW_v from the 1st, 32nd, 64th, and 96th layers in a 96-layer Transformer.

G.3 Correlation between W and ΔW

See Fig. 7 for the normalized subspace similarity between W and ΔW with varying r.

Note again that ΔW does not contain the top singular directions of W, since the similarity between the top 4 directions in ΔW and the top-10% of those in W barely exceeds 0.2. This gives evidence that ΔW contains those "task-specific" directions that are otherwise *not* emphasized in W.

An interesting next question to answer, is how "strong" do we need to amplify those task-specific directions, in order for the model adaptation to work well?

G.4 Amplification Factor

One can naturally consider a *feature amplification factor* as the *ratio* $\frac{\|\Delta W\|_F}{\|U^\top W V^\top\|_F}$, where U and V are the left- and right-singular matrices of the SVD decomposition of ΔW. (Recall $UU^\top W V^\top V$ gives the "projection" of W onto the subspace spanned by ΔW.)

Intuitively, when ΔW mostly contains task-specific directions, this quantity measures how much of them are amplified by ΔW. As shown in Sec. 6.3, for $r = 4$, this amplification factor is as large as 20. In other words, there are (generally speaking) four feature directions in each layer (out of the entire feature space from the pre-trained model W), that need to be amplified by a very large factor 20, in order to achieve our reported accuracy for the downstream specific task. And, one should expect a very different set of feature directions to be amplified for each different downstream task.

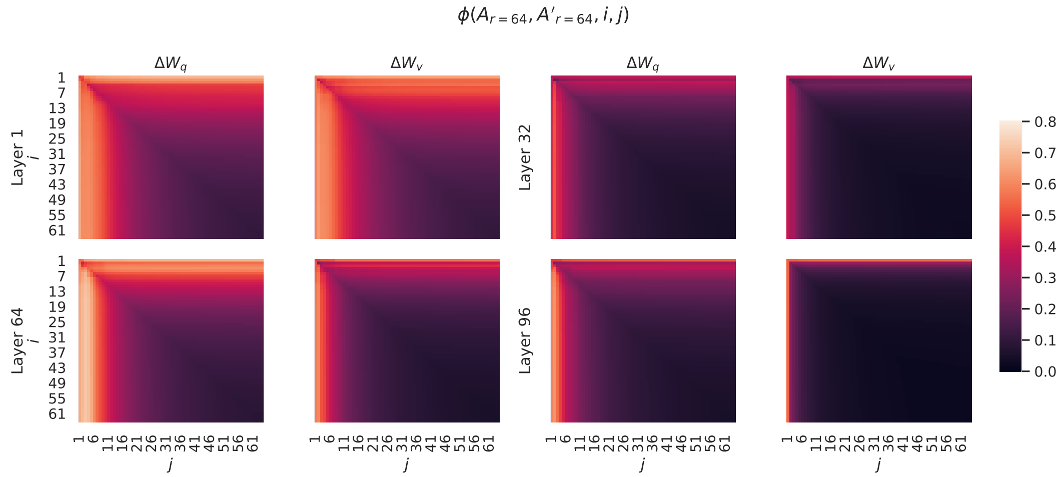

Figure 6: Normalized subspace similarity between the column vectors of $A_{r=64}$ from two randomly seeded runs, for both ΔW_q and ΔW_v from the 1st, 32nd, 64th, and 96th layers in a 96-layer Transformer.

Rank r	val_loss	BLEU	NIST	METEOR	ROUGE_L	CIDEr
1	1.23	68.72	8.7215	0.4565	0.7052	2.4329
2	1.21	69.17	8.7413	0.4590	0.7052	2.4639
4	1.18	**70.38**	**8.8439**	**0.4689**	0.7186	**2.5349**
8	1.17	69.57	8.7457	0.4636	**0.7196**	2.5196
16	**1.16**	69.61	8.7483	0.4629	0.7177	2.4985
32	**1.16**	69.33	8.7736	0.4642	0.7105	2.5255
64	**1.16**	69.24	8.7174	0.4651	0.7180	2.5070
128	**1.16**	68.73	8.6718	0.4628	0.7127	2.5030
256	**1.16**	68.92	8.6982	0.4629	0.7128	2.5012
512	**1.16**	68.78	8.6857	0.4637	0.7128	2.5025
1024	1.17	69.37	8.7495	0.4659	0.7149	2.5090

Table 14: Validation loss and test set metrics on E2E NLG Challenge achieved by LoRA with different rank r using GPT-2 Medium. Unlike on GPT-3 where $r = 1$ suffices for many tasks, here the performance peaks at $r = 16$ for validation loss and $r = 4$ for BLEU, suggesting the GPT-2 Medium has a similar intrinsic rank for adaptation compared to GPT-3 175B. Note that some of our hyperparameters are tuned on $r = 4$, which matches the parameter count of another baseline, and thus might not be optimal for other choices of r.

One may notice, however, for $r = 64$, this amplification factor is only around 2, meaning that *most* directions learned in ΔW with $r = 64$ are *not* being amplified by much. This should not be surprising, and in fact gives evidence (once again) that the intrinsic rank *needed* to represent the "task-specific directions" (thus for model adaptation) is low. In contrast, those directions in the rank-4 version of ΔW (corresponding to $r = 4$) are amplified by a much larger factor 20.

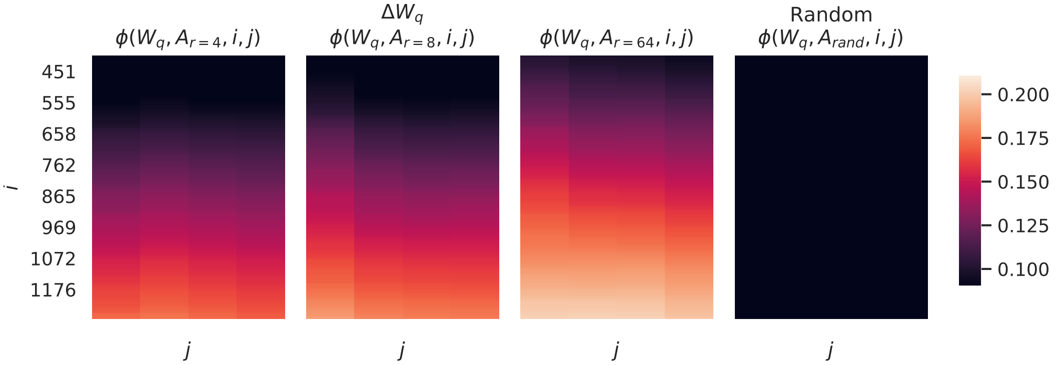

Figure 7: Normalized subspace similarity between the singular directions of W_q and those of ΔW_q with varying r and a random baseline. ΔW_q amplifies directions that are important but not emphasized in W. ΔW with a larger r tends to pick up more directions that are already emphasized in W.