LoRA: Low-Rank Adaptation of Large Language Models

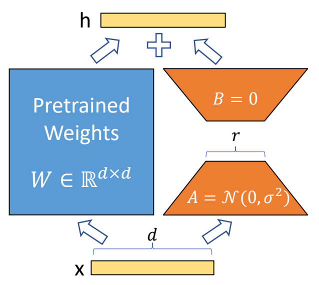
Edward Hu* **Yelong Shen*** **Phillip Wallis** **Zeyuan Allen-Zhu**
Yuanzhi Li **Shean Wang** **Lu Wang** **Weizhu Chen**
Microsoft Corporation
`{edwardhu, yeshe, phwallis, zeyuana,`
`yuanzhil, swang, luw, wzchen}@microsoft.com`
`yuanzhil@andrew.cmu.edu`
(Version 2)

Abstract

An important 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, full fine-tuning, which retrains all model parameters, becomes less feasible. Using GPT-3 175B as an example – deploying independent instances of fine-tuned models, each with 175B parameters, is prohibitively 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. Compared to GPT-3 175B fine-tuned with Adam, LoRA can reduce the number of trainable parameters by 10,000 times and the GPU memory requirement by 3 times. LoRA performs on-par or better than fine-tuning in model quality on RoBERTa, DeBERTa, GPT-2, and GPT-3, despite having fewer trainable parameters, a higher training throughput, and, unlike adapters, *no additional inference latency*. We also provide an empirical investigation into rank-deficiency in language model adaptation, which sheds light on the efficacy of LoRA. We release a package that facilitates the integration of LoRA with PyTorch models and provide our implementations and model checkpoints for RoBERTa, DeBERTa, and 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 the new model contains 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 (Radford et al., b) or RoBERTa large (Liu et al., 2019) to a critical deployment challenge for GPT-3 (Brown et al., 2020) with 175 billion trainable parameters.[1]


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 sought to mitigate this by adapting only some parameters or learning external modules for new tasks. This way, we only need to store and load a small number of task-specific parameters in addition to the pre-trained model for each task, greatly boosting the operational efficiency when deployed. However, existing techniques

*Equal contribution.
[0]Compared to V1, this draft includes better baselines, experiments on GLUE, and more on adapter latency.
[1]While GPT-3 175B achieves non-trivial performance with few-shot learning, fine-tuning boosts its performance significantly as shown in Appendix A.

LORA：大型語言模型的低秩適配

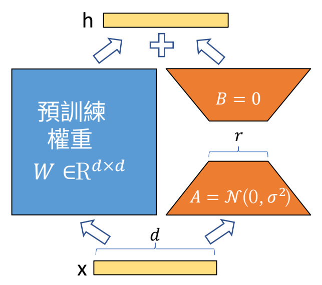
EdwardHu* **YelongShen*** **PhillipWallis**
ZeyuanAllen-ZhuYuanzhi Li Shean Wang Lu Wang
WeizhuChenMicrosoft Corporation{edwardhu, yeshe, ph
wallis, zeyuana, yuanzhil, swang, luw, wzchen}
@microsoft.com yuanzhil@andrew.cmu.edu (版本 2)

摘要

自然語言處理的一個重要範式是先在通用領域數據上進行大規模預訓練，然後再調適到特定任務或領域。隨著我們預訓練出更大的模型，重新訓練所有模型參數的完整微調變得愈來愈不可行。以 GPT-3 175B 為例——部署多個獨立的、各自微調的 175B 參數模型實例成本高得無法承受。我們提出 Low-Rank Adaptation（LoRA），它凍結預訓練模型權重，並在 Transformer 架構的每一層注入可訓練的低秩分解矩陣，大幅降低下游任務所需的可訓練參數數量。與使用 Adam 微調的 GPT-3 175B 相比，LoRA 可將可訓練參數數量減少 1 萬倍，並將 GPU 記憶體需求減少 3 倍。儘管可訓練參數更少、訓練吞吐量更高，且與 adapters 不同地不增加額外的推理延遲，LoRA 在 RoBERTa、DeBERTa、GPT-2 和 GPT-3 的模型質量上表現相當或更好。我們也對語言模型適配中的秩缺陷進行了實證研究，闡明了 LoRA 的有效性。我們釋出了一個方便將 LoRA 與 PyTorch 模型整合的套件，並在 https://github.com/microsoft/LoRA 提供了 RoBERTa、DeBERTa 與 GPT-2 的實作與模型檢查點。

1 介紹

許多自然語言處理的應用都依賴將單一大型預訓練語言模型適配到多個下游應用。這類適配通常透過微調完成，會更新預訓練模型的所有參數。微調的主要缺點是新模型會包含與原模型相同數量的參數。由於更大的模型每隔數月就會被訓練出來，這對 GPT-2（Radford 等）或 RoBERTa large（Liu 等，2019）來說可能只是個"不便"，但對擁有 1750 億可訓練參數的 GPT-3（Brown 等，2020）則成為一個關鍵的部署挑戰。[1]


h
預訓練
權重
$W \in \mathbb{R}^{d\times d}$
$B = 0$
r
$A = \mathcal{N}(0, \sigma^2)$
d
x
圖 1：我們的重參數化。我們只訓練 A 和 B。

許多人試圖透過僅調整部分參數或為新任務學習外部模組來緩解此問題。這樣一來，對於每個任務，除了預訓練模型外，我們只需儲存與載入少量特定任務的參數，能在部署時大幅提升操作效率。然而，現有技術

*貢獻相等。[0]與 V1 相比，這個草稿包含更好的基線、在 GLUE 上的實驗，以及更多關於適配器延遲的內容。[1]雖然 GPT-3 175B 在 few-shot 學習上已達到非凡的表現，但如附錄 A 所示，微調能顯著提升其性能。

often introduce inference latency (Houlsby et al., 2019; Rebuffi et al., 2017) by extending model depth or reduce the model's usable sequence length (Li & Liang, 2021; Lester et al., 2021; Hambardzumyan et al., 2020; Liu et al., 2021) (Section 3). More importantly, these method often fail to match the fine-tuning baselines, posing a trade-off between efficiency and model quality.

We take inspiration from Li et al. (2018a); Aghajanyan et al. (2020) which show that the learned over-parametrized models in fact reside on a low intrinsic dimension. We hypothesize that the change in weights during model adaptation also has a low "intrinsic rank", leading to our proposed **Lo**w-**R**ank **A**daptation (LoRA) approach. LoRA allows us to train some dense layers in a neural network indirectly by optimizing rank decomposition matrices of the dense layers' change during adaptation instead, while keeping the pre-trained weights frozen, as shown in Figure 1. Using GPT-3 175B as an example, we show that a very low rank (i.e., r in Figure 1 can be one or two) suffices even when the full rank (i.e., d) is as high as 12,288, making LoRA both storage- and compute-efficient.

LoRA possesses several key advantages.

- A pre-trained model can be shared and used to build many small LoRA modules for different tasks. We can freeze the shared model and efficiently switch tasks by replacing the matrices A and B in Figure 1, reducing the storage requirement and task-switching overhead significantly.
- LoRA makes training more efficient and lowers the hardware barrier to entry by up to 3 times when using adaptive optimizers since we do not need to calculate the gradients or maintain the optimizer states for most parameters. Instead, we only optimize the injected, much smaller low-rank matrices.
- Our simple linear design allows us to merge the trainable matrices with the frozen weights when deployed, *introducing no inference latency* compared to a fully fine-tuned model, by construction.
- LoRA is orthogonal to many prior methods and can be combined with many of them, such as prefix-tuning. We provide an example in Appendix E.

Terminologies and Conventions We make frequent references to the Transformer architecture and use the conventional terminologies for its dimensions. We call the input and output dimension size of a Transformer layer 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 accumulated gradient update during adaptation. We use r to denote the rank of a LoRA module. We follow the conventions set out by (Vaswani et al., 2017; Brown et al., 2020) and use Adam (Loshchilov & Hutter, 2019; Kingma & Ba, 2017) for model optimization and use a Transformer MLP feedforward dimension $d_{ffn} = 4 \times d_{model}$.

2 Problem Statement

While our proposal is agnostic to training objective, we focus on language modeling as our motivating 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 $P_\Phi(y|x)$ parametrized by Φ. For instance, $P_\Phi(y|x)$ can be a generic multi-task learner such as GPT (Radford et al., b; Brown et al., 2020) based on the Transformer architecture (Vaswani et al., 2017). Consider adapting this pre-trained model to 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 corresponding SQL command; for summarization, x_i is the content of an article and y_i its summary.

常常透過增加模型深度而引入推論延遲（Houlsby et al., 2019；Rebuffi et al., 2017），或縮減模型可用的序列長度（Li & Liang, 2021；Lester et al., 2021；Hambardzumyan et al., 2020；Liu et al., 2021）（見第3節）。更重要的是，這些方法經常無法達到微調基準的表現，造成效率與模型品質之間的權衡。

我們從 Li et al. (2018a); Aghajanyan et al. (2020) 得到啟發，這些研究顯示學到的過度參數化模型實際上存在於一個低內在維度上。我們假設模型調整期間權重的變化也具有低「內在秩」，因此提出了**Lo**w-**R**ank **A**daptation (LoRA) 方法。LoRA 允許我們間接訓練神經網路中的某些密集層，方法是優化這些密集層在調整期間權重變化的秩分解矩陣，同時保持預訓練權重凍結，如圖 1 所示。以 GPT-3 175B 為例，我們顯示即使完整秩（即圖 1 中的d）高達 12,288，非常低的秩（即圖 1 中的r 可為一或二）也足夠，這使得 LoRA 在儲存與計算上都很有效率。

LoRA 具備數項關鍵優勢。

- 已預訓練的模型可以被共用並用來為不同任務構建多個小型 LoRA 模組。我們可以凍結共用模型，並透過替換圖 1 中的矩陣 A 和 B 來有效地切換任務，從而大幅減少儲存需求與任務切換成本。
- LoRA 使訓練更有效率，並在使用自適應優化器時將硬體進入門檻降低最多達 3 倍，因為我們不需要為大多數參數計算梯度或維護優化器狀態。取而代之的，我們只優化注入的、遠小得多的低秩矩陣。
- 我們簡單的線性設計允許在部署時將可訓練矩陣與凍結權重合併，相較於完全微調的模型不會引入推理延遲，這是由設計所決定的。
- LoRA 與許多既有方法相互正交，且可與其中多數方法結合，例如 prefix-tuning。我們在附錄 E 提供了一個範例。

術語與慣例 我們經常參考 Transformer 架構並使用其維度的慣用術語。我們將 Transformer 層的輸入與輸出維度大小稱為 d_{model}。我們使用 W_q、W_k、W_v 和 W_o 來指代自注意力模組中的 query/key/value/output 投影矩陣。W 或 W_0 指的是預訓練權重矩陣，而 ΔW 則指其在調適過程中的累積梯度更新。我們使用 r 來表示 LoRA 模組的秩。我們遵循 (Vaswani et al., 2017; Brown et al., 2020) 所設定的慣例，並使用 Adam (Loshchilov & Hutter, 2019; Kingma & Ba, 2017) 進行模型優化，Transformer 的 MLP 前饋維度為 $d_{ffn} = 4 \times d_{model}$。

2 問題陳述

雖然我們的提議對訓練目標保持中立，但我們以語言模型作為主要的動機性案例。以下簡要說明語言建模問題，特別說明在給定任務特定提示時條件機率的最大化。

假設我們有一個以 Φ 為參數的預訓練自回歸語言模型 $P_\Phi(y|x)$。例如，$P_\Phi(y|x)$ 可以是基於 Transformer 架構（Vaswani et al., 2017）的通用多任務學習器，如 GPT（Radford et al., b；Brown et al., 2020）。考慮將此預訓練模型調整到下游的條件式文本生成任務，例如摘要、機器閱讀理解（MRC）及自然語言到 SQL（NL2SQL）。每個下游任務都由上下文—目標對的訓練資料集表示：$\mathcal{Z} = \{(x_i, y_i)\}_{i=1\ ..\ N,,}$，其中 x_i 與 y_i 都是字元（token）序列。例如，在 NL2SQL 中，x_i 是自然語言查詢，而 y_i 是對應的 SQL 指令；在摘要任務中，x_i 是文章內容，而 y_i 是其摘要。

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:

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

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 dimension $|\Delta\Phi|$ 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 feasible.

In this paper, we adopt 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:

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

In the subsequent sections, we propose to use a low-rank representation to encode $\Delta\Phi$ that is both compute- and memory-efficient. When the pre-trained model is GPT-3 175B, the number of trainable parameters $|\Theta|$ can be as small as 0.01% of $|\Phi_0|$.

3 Aren't Existing Solutions Good Enough?

The problem we set out to tackle is by no means new. Since the inception of transfer learning, dozens of works have sought to make model adaptation more parameter- and compute-efficient. See Section 6 for a survey of some of the well-known works. Using language modeling as an example, there are two prominent strategies when it comes to efficient adaptations: adding adapter layers (Houlsby et al., 2019; Rebuffi et al., 2017; Pfeiffer et al., 2021; Rücklé et al., 2020) or optimizing some forms of the input layer activations (Li & Liang, 2021; Lester et al., 2021; Hambardzumyan et al., 2020; Liu et al., 2021). However, both strategies have their limitations, especially in a large-scale and latency-sensitive production scenario.

Adapter Layers Introduce Inference Latency There are many variants of adapters. We focus on the original design by Houlsby et al. (2019) which has two adapter layers per Transformer block and a more recent one by Lin et al. (2020) which has only one per block but with an additional LayerNorm (Ba et al., 2016). While one can reduce the overall latency by pruning layers or exploiting multi-task settings (Rücklé et al., 2020; Pfeiffer et al., 2021), there is no direct ways to bypass the extra compute in adapter layers. This seems like a non-issue since adapter layers are designed to have few parameters (sometimes $<$1% of the original model) by having a small bottleneck dimension, which limits the FLOPs they can add. However, large neural networks rely on hardware parallelism to keep the latency low, and adapter layers have to be processed sequentially. This makes a difference in the online inference setting where the batch size is typically as small as one. In a generic scenario without model parallelism, such as running inference on GPT-2 (Radford et al., b) medium on a single GPU, we see a noticeable increase in latency when using adapters, even with a very small bottleneck dimension (Table 1).

This problem gets worse when we need to shard the model as done in Shoeybi et al. (2020); Lepikhin et al. (2020), because the additional depth requires more synchronous GPU operations such as `AllReduce` and `Broadcast`, unless we store the adapter parameters redundantly many times.

Directly Optimizing the Prompt is Hard The other direction, as exemplified by prefix tuning (Li & Liang, 2021), faces a different challenge. We observe that prefix tuning is difficult to optimize and that its performance changes non-monotonically in trainable parameters, confirming similar observations in the original paper. More fundamentally, reserving a part of the sequence length for adaptation necessarily reduces the sequence length available to process a downstream task, which we suspect makes tuning the prompt less performant compared to other methods. We defer the study on task performance to Section 5.

在完全微調（full fine-tuning）期間，模型以預訓練權重 Φ_0 作為初始化，並透過反覆沿著梯度更新以最大化條件語言模型目標，最終更新為 $\Phi_0 + \Delta\Phi$：

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

完全微調的一個主要缺點是，對於 每個下游任務，我們都會學習一組不同的參數 $\Delta\Phi$ ，其維度 $|\Delta\Phi|$ 等於 $|\Phi_0|$。因此，如果預訓練模型很大（例如具有 $|\Phi_0| \approx$ 175Billion 的 GPT-3），儲存與部署多個獨立的微調模型實例會很有挑戰，甚至可能不可行。

在本文中，我們採用更具參數效率的方法，其中專案特定的參數增量 $\Delta\Phi = \Delta\Phi(\Theta)$ 進一步由一組更小規模的參數 Θ 與 $|\Theta| \ll |\Phi_0|$ 編碼。尋找 $\Delta\Phi$ 的任務因此變為在 Θ 上進行優化：

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

在接下來的章節中，我們建議使用低秩表示來編碼 $\Delta\Phi$ ，此表示在計算與記憶體上皆有效率。當預訓練模型為 GPT-3 175B 時，可訓練參數的數量 $|\Theta|$ 最少可達 $|\Phi_0|$ 的 0.01%。

3. 現有解決方案真的不夠好嗎？

我們要解決的問題並非什麼新議題。自從遷移學習誕生以來，已有數十篇工作試圖讓模型調整在參數與計算上更為高效。有關一些知名工作的綜述，請見第6節。以語言模型為例，關於高效調整主要有兩種顯著策略：增加 adapter 層（Houlsby et al., 2019；Rebuffi et al., 2017；Pfeiffer et al., 2021；R¨uckl´e et al., 2020）或優化某些形式的輸入層啟用值（Li & Liang, 2021；Lester et al., 2021；Hambardzumyan et al., 2020；Liu et al., 2021）。然而，這兩種策略在大規模且對延遲敏感的生產環境中都有其侷限性。

AdapterLayersIntroduce InferenceLatency Adapter 有許多變體。我們聚焦於 Houlsby et al. (2019) 的原始設計（每個 Transformer 區塊有兩個 adapter 層）以及 Lin et al. (2020) 的較近期設計（每個區塊只有一個，但額外增加一個 LayerNorm（Ba et al., 2016））。雖然可以透過剪枝或利用多任務設定（R¨uckl´e et al., 2020；Pfeiffer et al., 2021）來降低整體延遲，但並沒有直接的方法可以繞過 adapter 層所增加的額外計算。這看似不是問題，因為 a dapter 層通常設計為參數很少（有時僅為原始模型的 <1%），藉由小的瓶頸維度來限制它們可能增加的 FLOPs。然而，大型神經網路仰賴硬體並行性來維持低延遲，而 adapter 層必須序列化處理。在線推理情境中（批次大小通常只有一筆）這會造成差異。在沒有模型並行的通用情況下，例如在單一 GPU 上執行 GPT-2 (Radford et al., b) medium 的推理，即使使用非常小的瓶頸維度，採用 adapter 仍會明顯增加延遲（見表 1）。

當我們需要像 Shoeybi et al. (2020)；Lepikhin et al. (2020) 那樣對模型進行分片時，這個問題會更嚴重，因為額外的深度會需要更多同步 GPU 操作（例如 AllReduce 與 Broadcast），除非我們將 adapter 參數重複儲存多次。

直接優化提示詞很困難 另一個方向，如 prefix tuning（Li & Liang, 2021）所示，面臨不同的挑戰。我們觀察到 prefix tuning 難以優化，且其效能隨可訓練參數的變動呈現非單調變化，這與原始論文的觀察一致。更根本的是，保留序列長度的一部分以供調適，必然會減少可用來處理下游任務的序列長度，我們懷疑這使得調整提示詞的效能不如其他方法。我們將把任務效能的研究延後到第 5 節。

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

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

4 OUR METHOD

We describe the simple design of LoRA and its practical benefits. The principles outlined here apply to any dense layers in deep learning models, though we only focus on certain weights in Transformer language models in our experiments as the motivating use case.

4.1 LOW-RANK-PARAMETRIZED UPDATE MATRICES

A neural network contains many dense layers which perform matrix multiplication. The weight matrices in these layers typically have full-rank. When adapting to a specific task, Aghajanyan et al. (2020) shows that the pre-trained language models have a low "instrisic dimension" and can still learn efficiently despite a random projection to a smaller subspace. Inspired by this, we hypothesize the updates to the weights also have a low "intrinsic rank" during adaptation. For a pre-trained weight matrix $W_0 \in \mathbb{R}^{d\times k}$, we constrain its 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 $r \ll \min(d, k)$. 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:

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

We illustrate our reparametrization in Figure 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 $\Delta W x$ by $\frac{\alpha}{r}$, where α is a constant in r. When optimizing with Adam, tuning α is roughly the same as tuning the learning rate if we scale the initialization appropriately. As a result, we simply set α to the first r we try and do not tune it. This scaling helps to reduce the need to retune hyperparameters when we vary r (Yang & Hu, 2021).

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. This means that when applying LoRA to all weight matrices and training all biases[2], we roughly recover the expressiveness of full fine-tuning by setting the LoRA rank r to the rank of the pre-trained weight matrices. In other words, as we increase the number of trainable parameters [3], training LoRA roughly converges to training the original model, while adapter-based methods converges to an MLP and prefix-based methods to a model that cannot take long input sequences.

No Additional Inference Latency. When deployed in production, we can explicitly compute and store $W = W_0 + BA$ and perform inference as usual. Note that both W_0 and BA are in $\mathbb{R}^{d\times k}$. 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. Critically, this

[2]They represent a negligible number of parameters compared to weights.
[3]An inevitability when adapting to hard tasks.

批次大小 序列長度 $\|\Theta\|$	32 512 0.5M	16 256 11M	1 128 11M
微調/LoRA	1449.4±0.8	338.0±0.6	19.8±2.7
Adapter^L	1482.0±1.0 (+2.2%)	354.8±0.5 (+5.0%)	23.9±2.1 (+20.7%)
Adapter^H	1492.2±1.0 （+3.0%）	366.3±0.5 (+8.4%)	25.8±2.2 (+30.3%)

表 1：在 GPT-2 medium 上單次前向傳播的推論延遲，以毫秒為單位，平均自 100 次試驗。我們使用 NVIDIA Quadro RTX8000。「$|\Theta|$」表示 adapter 層中可訓練參數的數量。Adapter^L 和 Adapter^H 是兩種 adapter 微調的變體，我們在第 5.1 節中描述。在線、短序列長度的情境下，adapter 層引入的推論延遲可能相當顯著。完整研究見附錄 B。

4 我們的方法

我們說明 LoRA 的簡單設計及其實際效益。此處所述的原則適用於深度學習模型中的任意密集層，但在實驗中我們僅針對 Transformer 語言模型中的特定權重進行討論，作為主要的應用情境。

4.1 低秩參數化的更新矩陣

神經網路包含許多執行矩陣乘法的密集層。這些層中的權重矩陣通常具有滿秩。當要適應特定任務時，Aghajanyan 等人（2020）指出，預訓練語言模型具有低「內在維度」，即便將參數隨機投影到較小的子空間仍能有效學習。受此啟發，我們假設在微調期間權重的更新也具有低「內在秩」。對於預訓練的權重矩陣 $W_0 \in \mathbb{R}^{d\times k}$，我們透過以低秩分解 $W_0 + \Delta W = W_0 + BA$ 表示來約束其更新，其中 $B \in \mathbb{R}^{d\times r}, A \in \mathbb{R}^{r\times k}$，以及秩為 $r \ll \min(d, k)$。在訓練時，W_0 固定不接收梯度更新，而 A 和 B 則包含可訓練參數。注意 W_0 和 $\Delta W = BA$ 都與相同輸入相乘，且它們的輸出向量按座標逐項相加。對於 $h = W_0x$，我們修改後的前向傳播為：

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

我們在圖 1 中說明了我們的重參數化。我們對 A 使用隨機高斯初始化，對 B 則置為零，因此在訓練開始時 $\Delta W = BA$ 為零。接著我們將 $\Delta W x$ 按 $\frac{\alpha}{r}$ 比例縮放，其中 α 是 r 中的一個常數。在使用 Adam 進行優化時，如果我們適當地縮放初始化，調整 α 與調整學習率大致相同。因此，我們僅將 α 設為我們嘗試的第一個 r，而不對其進行調整。這種縮放有助於在改變 r 時減少重新調整超參數的需求（Yang & Hu, 2021）。

完全微調的一般化。一種更一般形式的微調允許訓練預訓練參數的子集。LoRA 更進一步，不要求在調整期間累積的梯度更新對權重矩陣具有全秩。這表示當把 LoRA 應用到所有權重矩陣並訓練所有偏置時[2]，，透過將 LoRA 的秩 r 設為預訓練權重矩陣的秩，我們大致可以恢復全量微調的表現。換句話說，當我們增加可訓練參數數量[3]時，訓練 LoRA 大致會收斂到訓練原始模型，而基於 adapter 的方法則收斂到一個 MLP，基於 prefix 的方法則會收斂到無法處理較長輸入序列的模型。

不增加額外推理延遲。當部署於生產環境時，我們可以明確地計算並儲存 $W = W_0 + BA$，然後像平常一樣進行推理。請注意，W_0 與 BA 都位於 $\mathbb{R}^{d\times k}$中。當我們需要切換到另一個下游任務時，可以透過先減去 BA 再加上不同的 $B'A'$來還原 W_0，這是一個快速的操作，且幾乎不增加記憶體負擔。關鍵是，這

[2]與權重相比，它們代表了可忽略不計的參數數量。[3]在適應困難任務時這是不可避免的。

guarantees that we do not introduce any additional latency during inference compared to a fine-tuned model by construction.

4.2 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 adapting the attention weights** for downstream tasks and freeze the MLP modules (so they are not trained in downstream tasks) both for simplicity and parameter-efficiency.We further study the effect on adapting different types of attention weight matrices in a Transformer in Section 7.1. We leave the empirical investigation of adapting the MLP layers, LayerNorm layers, and biases to a future work.

Practical Benefits and Limitations. The most significant benefit comes from the reduction in memory and storage usage. For a large Transformer trained with Adam, we reduce that VRAM usage by up to $2/3$ if $r \ll d_{model}$ as we do not need to store the optimizer states for the frozen parameters. On GPT-3 175B, we reduce the VRAM consumption during training from 1.2TB to 350GB. With $r = 4$ and only the query and value projection matrices being adapted, the checkpoint size is reduced by roughly 10,000× (from 350GB to 35MB)[4]. This allows us to train with significantly fewer GPUs and avoid I/O bottlenecks. Another benefit is that we can switch between tasks while deployed at a much lower cost by only swapping the LoRA weights as opposed to all the parameters. This allows for the creation of many customized models that can be swapped in and out on the fly on machines that store the pre-trained weights in VRAM. We also observe a 25% speedup during training on GPT-3 175B compared to full fine-tuning[5] as we do not need to calculate the gradient for the vast majority of the parameters.

LoRA also 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, if one chooses to absorb A and B into W to eliminate additional inference latency. Though it is possible to not merge the weights and dynamically choose the LoRA modules to use for samples in a batch for scenarios where latency is not critical.

5 Empirical Experiments

We evaluate the downstream task performance of LoRA on RoBERTa (Liu et al., 2019), DeBERTa (He et al., 2021), and GPT-2 (Radford et al., b), before scaling up to GPT-3 175B (Brown et al., 2020). Our experiments cover a wide range of tasks, from natural language understanding (NLU) to generation (NLG). Specifically, we evaluate on the GLUE (Wang et al., 2019) benchmark for RoBERTa and DeBERTa. We follow the setup of Li & Liang (2021) on GPT-2 for a direct comparison and add WikiSQL (Zhong et al., 2017) (NL to SQL queries) and SAMSum (Gliwa et al., 2019) (conversation summarization) for large-scale experiments on GPT-3. See Appendix C for more details on the datasets we use. We use NVIDIA Tesla V100 for all experiments.

5.1 Baselines

To compare with other baselines broadly, we replicate the setups used by prior work and reuse their reported numbers whenever possible. This, however, means that some baselines might only appear in certain experiments.

Fine-Tuning (FT) is a common approach for adaptation. During fine-tuning, the model is initialized to the pre-trained weights and biases, and all model parameters undergo gradient updates.A simple variant is to update only some layers while freezing others. We include one such baseline reported in prior work (Li & Liang, 2021) on GPT-2, which adapts just the last two layers (**FT$^{\mathbf{Top2}}$**).

[4]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.

[5]For GPT-3 175B, the training throughput for full fine-tuning is 32.5 tokens/s per V100 GPU; with the same number of weight shards for model parallelism, the throughput is 43.1 tokens/s per V100 GPU for LoRA.

保證在推理時，我們與經微調的模型相比，從架構上不會引入任何額外延遲。

4.2 將 LoRA 應用到 Transformer

原則上，我們可以將 LoRA 應用到神經網路中任一子集的權重矩陣，以減少可訓練參數的數量。在 Transformer 結構中，自注意力模組有四個權重矩陣 (W_q, W_k, W_v, W_o)，而 MLP 模組有兩個。我們將 W_q (或 W_k, W_v) 視為一個維度為 $d_{model} \times d_{model}$ 的單一矩陣，儘管輸出維度通常會被切分成多個注意力頭。為了簡化與參數效率，我們在下游任務中僅研究**只調整注意力權重** 並凍結 MLP 模組（因此在下游任務中不會訓練它們）。我們在第 7.1 節進一步研究在 Transformer 中調整不同類型注意力權重矩陣的影響。至於調整 MLP 層、LayerNorm 層與偏差項的實證研究，留待未來工作。

實用的好處與限制。最顯著的好處來自於記憶體與儲存使用量的減少。對於以 Adam 訓練的大型 Transformer，若 $r \ll d_{model}$ 我們可以將 VRAM 使用量降低最多 $2/3$ ，因為不需要為已凍結的參數儲存優化器狀態。在 GPT-3 175B 上，我們將訓練期間的 VRAM 消耗從 1.2TB 降至 350GB。採用 $r = 4$ 且僅調整 query 與 value 投影矩陣時，檢查點大小大約減少 10,000× (從 350GB 到 35MB)[4]。這使我們能以顯著較少的 GPU 進行訓練，並避免 I/O 瓶頸。另一個好處是部署時切換任務的成本大幅降低，只需交換 LoRA 權重而非所有參數。這允許在將預訓練權重儲存在 VRAM 的機器上，隨時替換多個客製化模型。我們也觀察到在 GPT-3 175B 上的訓練速度比完整微調快約 25%[5] ，因為大多數參數不需要計算梯度。

LoRA 也有其侷限性。例如，若選擇將 A 和 B 吸收進 W 以消除額外的推論延遲，要在單次前向傳播中對不同任務且具有不同 A 和 B 的輸入進行批次處理並不容易。當延遲不是關鍵時，則可以不合併權重，而是動態為批次中的樣本選擇要使用的 LoRA 模組。

5 實證實驗

我們在 RoBERTa (Liu et al., 2019)、DeBERTa (He et al., 2021) 和 GPT-2 (Radford et al., b) 上評估 LoRA 在下游任務的表現，並在擴展至 GPT-3 175B (Brown et al., 2020) 前進行測試。我們的實驗涵蓋廣泛任務，從自然語言理解 (NLU) 到生成 (NLG)。具體而言，我們在 RoBERTa 和 DeBERTa 上評估 GLUE (Wang et al., 2019) 基準。對於 GPT-2，我們遵循 Li & Liang (2021) 的設置以便直接比較，並在 GPT-3 的大規模實驗中加入 WikiSQL (Zhong et al., 2017)（自然語言到 SQL 查詢）與 SAMSum (Gliwa et al., 2019)（對話摘要）。更多資料集細節見附錄 C。我們所有實驗均使用 NVIDIA Tesla V100。

5.1 基準線

為了與其他基準線進行廣泛比較，我們重現先前工作的設定，並在可能的情況下重用它們報告的數據。然而，這也表示某些基準線可能只出現在特定實驗中。

Fine-Tuning(FT) 是常見的適應方法。微調時，模型以預訓練的權重與偏差為初始值，所有模型參數都會進行梯度更新。一個簡單的變體是僅更新部分層、凍結其他層。我們包含了先前工作（Li & Liang, 2021）在 GPT-2 上報導的一個此類基線，只調整最後兩層（**FT$^{\mathbf{Top2}}$**）。

[4]我們在部署時仍然需要 350GB 的模型；不過，儲存 100 個調適後的模型只需要 350GB + 35MB * 100 ≈ 354GB，而非 100 * 350GB ≈ 35TB。[5]對於 GPT-3 175B，完整微調的訓練吞吐量是每台 V100 GPU 每秒 32.5 個 token；在使用相同數量的權重分片進行模型並行時，LoRA 的吞吐量是每台 V100 GPU 每秒 43.1 個 token。

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

Table 2: RoBERTa$_{\text{base}}$, RoBERTa$_{\text{large}}$, and 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.

Bias-only or BitFit is a baseline where we only train the bias vectors while freezing everything else. Contemporarily, this baseline has also been studied by BitFit (Zaken et al., 2021).

Prefix-embedding tuning (PreEmbed) inserts special tokens among the input tokens. These special tokens have trainable word embeddings and are generally not in the model's vocabulary. Where to place such tokens can have an impact on performance. We focus on "prefixing", which prepends such tokens to the prompt, and "infixing", which appends to the prompt; both are discussed in Li & Liang (2021). 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 (PreLayer) 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 Transformer layer. The activations computed from previous layers are simply replaced by 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 Transformer layers.

Adapter tuning as proposed in Houlsby et al. (2019) inserts adapter layers between the self-attention module (and the MLP module) and the subsequent residual connection. There are two fully connected layers with biases in an adapter layer with a nonlinearity in between. We call this original design **Adapter$^{\mathbf{H}}$**. Recently, Lin et al. (2020) proposed a more efficient design with the adapter layer applied only after the MLP module and after a LayerNorm. We call it **Adapter$^{\mathbf{L}}$**. This is very similar to another deign proposed in Pfeiffer et al. (2021), which we call **Adapter$^{\mathbf{P}}$**. We also include another baseline call AdapterDrop (Rücklé et al., 2020) which drops some adapter layers for greater efficiency (**Adapter$^{\mathbf{D}}$**). We cite numbers from prior works whenever possible to maximize the number of baselines we compare with; they are in rows with an asterisk (*) in the first column. In all cases, we have $|\Theta| = \hat{L}_{Adpt} \times (2 \times d_{model} \times r + r + d_{model}) + 2 \times \hat{L}_{LN} \times d_{model}$ where $\hat{L}_{Adpt}$ is the number of adapter layers and $\hat{L}_{LN}$ the number of trainable LayerNorms (e.g., in Adapter$^{\text{L}}$).

LoRA adds trainable pairs of rank decomposition matrices in parallel to existing weight matrices. As mentioned in Section 4.2, we only apply LoRA to W_q and W_v in most experiments for simplicity. The number of trainable parameters is determined by the rank r and the shape of the original weights: $|\Theta| = 2 \times \hat{L}_{LoRA} \times d_{model} \times r$, where $\hat{L}_{LoRA}$ is the number of weight matrices we apply LoRA to.

模型與方法	# 可訓練 參數	MNLI	SST-2	MRPC	CoLA	QNLI	QQP	RTE	STS-B	平均
RoB$_{\text{base}}$ (FT)*	125.0M	**87.6**	94.8	90.2	**63.6**	92.8	**91.9**	78.7	91.2	86.4
RoB$_{\text{base}}$ (BitFit)*	0.1M	84.7	93.7	**92.7**	62.0	91.8	84.0	81.5	90.8	85.2
RoB$_{\text{base}}$ (Adpt$^{\text{D}}$)*	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$_{\text{base}}$ (Adpt$^{\text{D}}$)*	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$_{\text{base}}$ (LoRA)	0.3M	$87.5_{\pm.3}$	**95.1**$_{\pm.2}$	$89.7_{\pm.7}$	$63.4_{\pm1.2}$	**93.3**$_{\pm.3}$	$90.8_{\pm.1}$	**86.6**$_{\pm.7}$	**91.5**$_{\pm.2}$	**87.2**
RoB$^{\text{large}}$ (FT)*	355.0M	90.2	**96.4**	**90.9**	68.0	94.7	**92.2**	86.6	92.4	88.9
RoB$^{\text{large}}$ (LoRA)	0.8M	**90.6**$_{\pm.2}$	$96.2_{\pm.5}$	**90.9**$_{\pm1.2}$	**68.2**$_{\pm1.9}$	**94.9**$_{\pm.3}$	$91.6_{\pm.1}$	**87.4**$_{\pm2.5}$	**92.6**$_{\pm.2}$	**89.0**
RoB$^{\text{large}}$ (Adpt$^{\text{P}}$)†	3.0M	$90.2_{\pm.3}$	$96.1_{\pm.3}$	$90.2_{\pm.7}$	**68.3**$_{\pm1.0}$	**94.8**$_{\pm.2}$	**91.9**$_{\pm.1}$	$83.8_{\pm2.9}$	$92.1_{\pm.7}$	88.4
RoB$^{\text{large}}$ (Adpt$^{\text{P}}$)†	0.8M	**90.5**$_{\pm.3}$	**96.6**$_{\pm.2}$	$89.7_{\pm1.2}$	$67.8_{\pm2.5}$	**94.8**$_{\pm.3}$	$91.7_{\pm.2}$	$80.1_{\pm2.9}$	$91.9_{\pm.4}$	87.9
RoB$^{\text{large}}$ (Adpt$^{\text{H}}$)†	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
RoB$^{\text{large}}$ (Adpt$^{\text{H}}$)†	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
RoB$^{\text{large}}$ (LoRA)†	0.8M	**90.6**$_{\pm.2}$	$96.2_{\pm.5}$	**90.2**$_{\pm1.0}$	$68.2_{\pm1.9}$	**94.8**$_{\pm.3}$	$91.6_{\pm.2}$	**85.2**$_{\pm1.1}$	**92.3**$_{\pm.5}$	**88.6**
DeB$_{\text{XXL}}$ (FT)*	1500.0M	91.8	**97.2**	92.0	72.0	**96.0**	92.7	93.9	92.9	91.1
DeB$_{\text{XXL}}$ (LoRA)	4.7M	**91.9**$_{\pm.2}$	$96.9_{\pm.2}$	**92.6**$_{\pm.6}$	**72.4**$_{\pm1.1}$	**96.0**$_{\pm.1}$	**92.9**$_{\pm.1}$	**94.9**$_{\pm.4}$	**93.0**$_{\pm.2}$	**91.3**

Table 2: RoBERTa$_{\text{base}}$, RoBERTa$^{\text{large}}$, and DeBERTa$_{\text{XXL}}$ 使用不同的調適方法在 GLUE 基準上的表現。我們報告 MNLI 的整體（matched 與 mismatched）準確率、CoLA 的 Matthew 相關係數、STS-B 的 Pearson 相關係數，以及其他任務的準確率。所有指標皆以數值越高越佳。* 表示來自先前工作的已發表數字。 † 表示在與 Houlsby 等人（2019）類似的設定中執行的實驗，以便進行公平比較。

Bias-only orBitFit 是一個基線方法，我們僅訓練偏置向量而將其他所有參數凍結。近期也有研究同樣的基線（BitFit，Zaken et al., 2021）。

Prefix-embedding tuning (PreEmbed) 會在輸入標記中插入特殊標記。這些特殊標記具有可訓練的詞嵌入，通常不在模型詞彙表中。放置這些標記的位置會影響效能。我們專注於「前綴」（prefixing），即將這些標記置於 prompt 之前，以及「中綴」（infixing），即附加於 prompt 之後；兩者皆在 Li & Liang (2021) 中討論。我們使用 l_p（對應 l_i）表示前綴（對應中綴）標記的數量。可訓練參數的數量為 $|\Theta| = d_{model} \times (l_p + l_i)$。

Prefix-layer tuning (PreLayer)是對前綴嵌入微調的一種擴展。與僅學習某些特殊標記的字詞嵌入（或等價地，嵌入層之後的激活）不同，我們學習每一個 Transformer 層之後的激活。先前層計算出的激活會被可訓練的激活簡單取代。最終可訓練參數的數量為 $|\Theta| = L \times d_{model} \times (l_p + l_i)$，其中 L 是 Transformer 層的數量。

Adapter tuning 如 Houlsby 等（2019）所提出，在自注意力模組（和 MLP 模組）與隨後的殘差連接之間插入 adapter 層。adapter 層包含兩個帶偏置的全連接層，中間有一個非線性函數。我們將這個原始設計稱為 **Adapter$^{\mathbf{H}}$**。最近，Lin 等（2020）提出了一種更高效的設計，僅在 MLP 模組之後且在 LayerNorm 之後應用 adapter 層。我們稱之為 **Adapter$^{\mathbf{L}}$**。這與 Pfeiffer 等（2021）提出的另一種設計非常相似，我們稱之為 **Adapter$^{\mathbf{P}}$**。我們也包含另一個基準 AdapterDrop（R¨uckl´e 等，2020），其透過丟棄部分 adapter 層以提高效率（**Adapter$^{\mathbf{D}}$**）。我們在可能的情況下引用先前工作的數據，以最大化比較的基準數量；這些在第一欄帶星號 (*) 的列中。
ˆ ˆˆ 在所有情況下，我們有
$|\Theta| = L_{Adpt} \times (2 \times d_{model} \times r + r + d_{model}) + 2 \times L_{LN} \times d_{model}$，其中 L_{Adpt} 是 adapter 層的數量，且 ˆ L_{LN} 為可訓練的 LayerNorm 數量（例如，在 Adapter$^{\text{L}}$ 中）。

LoRA 在現有權重矩陣旁並行加入可訓練的低秩分解矩陣對。如第4.2節所述，為簡化起見，我們在大多數實驗中僅將 LoRA 應用於 W_q 和 W_v。可訓練參數的數量由秩 r 與原始權重的形狀決定：
ˆ ˆ$|\Theta| = 2 \times L_{LoRA} \times d_{model} \times r$，其中 L_{LoRA} 是我們應用 LoRA 的權重矩陣數量。

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 (Adapter^{L})*	0.37M	66.3	8.41	45.0	69.8	2.40
GPT-2 M (Adapter^{L})*	11.09M	68.9	8.71	46.1	71.3	2.47
GPT-2 M (Adapter^{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 (Adapter^{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 (Adapter^{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.

5.2 RoBERTa base/large

RoBERTa (Liu et al., 2019) optimized the pre-training recipe originally proposed in BERT (Devlin et al., 2019a) and boosted the latter's task performance without introducing many more trainable parameters. While RoBERTa has been overtaken by much larger models on NLP leaderboards such as the GLUE benchmark (Wang et al., 2019) in recent years, it remains a competitive and popular pre-trained model for its size among practitioners. We take the pre-trained RoBERTa base (125M) and RoBERTa large (355M) from the HuggingFace Transformers library (Wolf et al., 2020) and evaluate the performance of different efficient adaptation approaches on tasks from the GLUE benchmark. We also replicate Houlsby et al. (2019) and Pfeiffer et al. (2021) according to their setup. To ensure a fair comparison, we make two crucial changes to how we evaluate LoRA when comparing with adapters. First, we use the same batch size for all tasks and use a sequence length of 128 to match the adapter baselines. Second, we initialize the model to the pre-trained model for MRPC, RTE, and STS-B, not a model already adapted to MNLI like the fine-tuning baseline. Runs following this more restricted setup from Houlsby et al. (2019) are labeled with †. The result is presented in Table 2 (Top Three Sections). See Section D.1 for details on the hyperparameters used.

5.3 DeBERTa XXL

DeBERTa (He et al., 2021) is a more recent variant of BERT that is trained on a much larger scale and performs very competitively on benchmarks such as GLUE (Wang et al., 2019) and SuperGLUE (Wang et al., 2020). We evaluate if LoRA can still match the performance of a fully fine-tuned DeBERTa XXL (1.5B) on GLUE. The result is presented in Table 2 (Bottom Section). See Section D.2 for details on the hyperparameters used.

5.4 GPT-2 medium/large

Having shown that LoRA can be a competitive alternative to full fine-tuning on NLU, we hope to answer if LoRA still prevails on NLG models, such as GPT-2 medium and large (Radford et al., b). We keep our setup as close as possible to Li & Liang (2021) for a direct comparison. Due to space constraint, we only present our result on E2E NLG Challenge (Table 3) in this section. See Section F.1 for results on WebNLG (Gardent et al., 2017) and DART (Nan et al., 2020). We include a list of the hyperparameters used in Section D.3.

模型與方法	# 可訓練 參數	E2E 自然語言生成挑戰				
		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 (Adapter^{L})*	0.37M	66.3	8.41	45.0	69.8	2.40
GPT-2 M (Adapter^{L})*	11.09M	68.9	8.71	46.1	71.3	2.47
GPT-2 M (Adapter^{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（微調）*	774.03M	68.5	8.78	46.0	69.9	2.45
GPT-2 L（Adapter^{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（Adapter^{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	$\mathbf{70.4}_{\pm.1}$	$\mathbf{8.89}_{\pm.02}$	$\mathbf{46.8}_{\pm.2}$	$\mathbf{72.0}_{\pm.2}$	$2.47_{\pm.02}$

表 3：GPT-2 中型 (M) 與大型 (L) 在 E2E NLG 挑戰中採用不同調適方法的結果。對於所有指標，數值越高越好。LoRA 在可比較或更少可訓練參數的情況下，優於數個基線。我們展示了自行執行實驗的信賴區間。* 表示先前研究中發表的數值。

5.2 ROBERTA BASE/LARGE

RoBERTa（Liu et al., 2019）優化了最初由 BERT（Devlin et al., 2019a）提出的預訓練流程，並在未增加太多可訓練參數的情況下提升了後者的任務性能。儘管近年來在像 GLUE 基準（Wang et al., 2019）等 NLP 排行榜上已被更大規模的模型超越，RoBERTa 仍因其規模而在實務上保持競爭力並廣受使用。我們從 HuggingFace Transformers 庫（Wolf et al., 2020）取得預訓練的 RoBERTa base（125M）與 RoBERTa large（355M），並在 GLUE 基準的任務上評估不同高效適配方法的表現。我們也根據其設定復現 Houlsby et al.（2019）和 Pfeiffer et al.（2021）。為了確保公平比較，我們在將 LoRA 與 adapters 比較時對評估方式做了兩項關鍵更動。首先，對所有任務使用相同的批次大小，並以序列長度128來匹配 adapter 的基線。其次，對 MRPC、RTE 和 STS-B，模型初始化為預訓練模型，而非已經適配至 MNLI 的模型（如微調基線所用）。遵循 Houlsby et al.（2019）更嚴格設定的實驗以 †標註。結果呈現在表2（上方三個部分）。超參數詳情請見附錄 D.1。

5.3 DEBERTA XXL

DeBERTa（He 等，2021）是 BERT 的較新變體，在更大規模的資料上進行訓練，並在 GLUE（Wang 等，2019）和 SuperGLUE（Wang 等，2020）等基準上表現非常具有競爭力。我們評估 LoRA 是否仍能匹配在 GLUE 上完全微調的 DeBERTa XXL（1.5B）的性能。結果呈現在表 2（下半部）。有關使用之超參數的詳細資訊，請參見附錄 D.2。

5.4 GPT-2 MEDIUM/LARGE

在展示了 LoRA 在 NLU 任務上可以成為與完整微調相競爭的替代方案後，我們希望回答 LoRA 在 NLG 模型（例如 GPT-2 medium 與 large（Radford 等，b））上是否仍然具優勢。我們的實驗設定儘可能與 Li & Liang (2021) 保持一致以便直接比較。由於篇幅限制，本節僅呈現我們在 E2E NLG Challenge（表 3）上的結果。WebNLG（Gardent 等，2017）與 DART（Nan 等，2020）的結果請見附錄第 F.1 節。我們在附錄第 D.3 節列出所使用的超參數清單。

Model&Method	# Trainable Parameters	WikiSQL Acc. (%)	MNLI-m Acc. (%)	SAMSum R1/R2/RL
GPT-3 (FT)	175,255.8M	**73.8**	89.5	52.0/28.0/44.5
GPT-3 (BitFit)	14.2M	71.3	91.0	51.3/27.4/43.5
GPT-3 (PreEmbed)	3.2M	63.1	88.6	48.3/24.2/40.5
GPT-3 (PreLayer)	20.2M	70.1	89.5	50.8/27.3/43.5
GPT-3 (Adapter^H)	7.1M	71.9	89.8	53.0/28.9/44.8
GPT-3 (Adapter^H)	40.1M	73.2	**91.5**	53.2/29.0/45.1
GPT-3 (LoRA)	4.7M	73.4	**91.7**	**53.8/29.8/45.9**
GPT-3 (LoRA)	37.7M	**74.0**	**91.6**	53.4/29.2/45.1

Table 4: Performance of different adaptation methods on GPT-3 175B. We report the logical form validation accuracy on WikiSQL, validation accuracy on MultiNLI-matched, and Rouge-1/2/L on SAMSum. LoRA performs better than prior approaches, including full fine-tuning. The results on WikiSQL have a fluctuation around ±0.5%, MNLI-m around ±0.1%, and SAMSum around ±0.2/±0.2/±0.1 for the three metrics.

5.5 Scaling up to GPT-3 175B

As a final stress test for LoRA, we scale up to GPT-3 with 175 billion parameters. Due to the high training cost, we only report the typical standard deviation for a given task over random seeds, as opposed to providing one for every entry. See Section D.4 for details on the hyperparameters used.

As shown in Table 4, LoRA matches or exceeds the fine-tuning baseline on all three datasets. Note that not all methods benefit monotonically from having more trainable parameters, as shown in Figure 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 similar observations in Li & Liang (2021). 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. Separately, we investigate the performance of different adaptation approaches in the low-data regime in Section F.3.

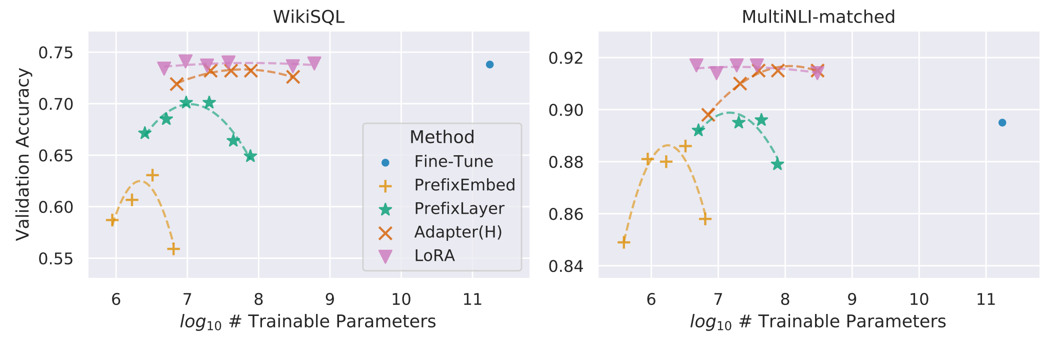


Figure 2: GPT-3 175B validation accuracy vs. number of trainable parameters of several adaptation methods on WikiSQL and MNLI-matched. LoRA exhibits better scalability and task performance. See Section F.2 for more details on the plotted data points.

6 Related Works

Transformer Language Models. Transformer (Vaswani et al., 2017) is a sequence-to-sequence architecture that makes heavy use of self-attention. Radford et al. (a) 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 (Devlin et al., 2019b) and GPT-2 (Radford et al., b) – both are large Transformer lan-

模型與方法	# 可訓練的參數	WikiSQL 準確率（%）	MNLI-m 準確率（%）	SAMSum R1/R2/RL
GPT-3（微調）	175,255.8M	**73.8**	89.5	52.0/28.0/44.5
GPT-3（BitFit）	14.2M	71.3	91.0	51.3/27.4/43.5
GPT-3（預嵌入）	3.2M	63.1	88.6	48.3/24.2/40.5
GPT-3（預層）	20.2M	70.1	89.5	50.8/27.3/43.5
GPT-3 (Adapter^H)	7.1M	71.9	89.8	53.0/28.9/44.8
GPT-3 (Adapter^H)	40.1M	73.2	**91.5**	53.2/29.0/45.1
GPT-3（LoRA）	4.7M	73.4	**91.7**	**53.8/29.8/45.9**
GPT-3（LoRA）	37.7M	**74.0**	**91.6**	53.4/29.2/45.1

表 4：在 GPT-3 175B 上不同調整方法的表現。我們報告了 WikiSQL 的邏輯形式驗證準確率、MultiNLI-matched 的驗證準確率，以及 SAMSum 的 Rouge-1/2/L。LoRA 的表現優於先前方法，包括完整微調。WikiSQL 的結果波動約為 ±0.5%，MNLI-m 約為 ±0.1%，SAMSum 在三個指標上約為 ±0.2/±0.2/±0.1 。

5.5 擴展到 GPT-3 175B

作為對 LoRA 的最後壓力測試，我們將模型規模擴展至具有 1750 億參數的 GPT-3。由於訓練成本高昂，我們僅報告每個任務在不同隨機種子下的典型標準差，而非為每一項結果列出標準差。所使用超參數的詳細資訊請見附錄第 D.4 節。

如表 4 所示，LoRA 在三個資料集上都達到或超過了微調基準。注意，如圖 2 所示，並非所有方法在增加可訓練參數後都會單調受益。我們觀察到當使用超過 256 個特殊標記進行前綴嵌入調整，或超過 32 個特殊標記進行前綴層調整時，性能會顯著下降。這與 Li & Liang (2021) 的類似觀察相互印證。雖然對此現象進行深入研究超出本工作的範圍，但我們懷疑較多的特殊標記會使輸入分佈進一步偏離預訓練資料分佈。另在第 F.3 節中，我們探討了不同調適方法在少量資料情境下的表現。

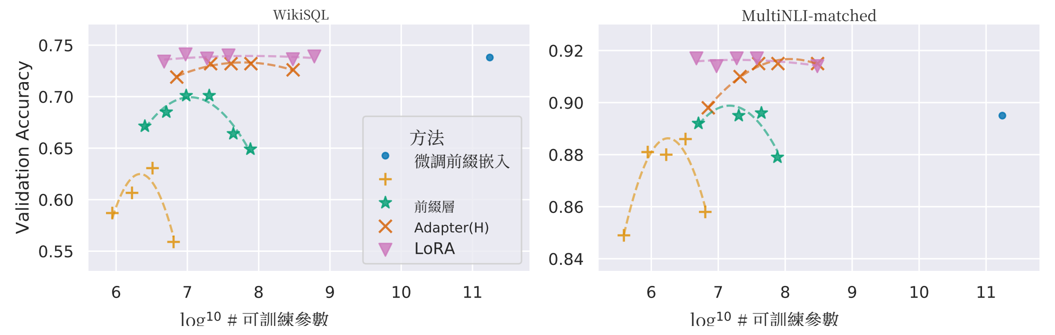


圖 2：GPT-3 175B 在 WikiSQL 與 MNLI-matched 上，驗證準確率與多種調適方法可訓練參數數量的關係。LoRA 展現出較佳的可擴展性與任務表現。有關圖中資料點的更多詳細資訊，請參閱第 F.2 節。

6 相關工作

Transformer 語言模型。 Transformer (Vaswani et al., 2017) 是一種大量使用自注意力的序列到序列架構。Radford et al. (a) 將其應用於自回歸語言建模，透過堆疊 Transformer 解碼器。自那時起，基於 Transformer 的語言模型主導了自然語言處理，並在許多任務上達到最先進的成果。隨著 BERT (Devlin et al., 2019b) 與 GPT-2 (Radford et al., b) 的出現，一種新範式興起——兩者都是大型 Transformer 語言

guage 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 (Brown et al., 2020) 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 (Brown et al., 2020). 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 Devlin et al. (2019b); Radford et al. (a). Variants of it include learning just a subset of the parameters Devlin et al. (2019b); Collobert & Weston (2008), 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. Many have proposed inserting *adapter* layers between existing layers in a neural network (Houlsby et al., 2019; Rebuffi et al., 2017; Lin et al., 2020). Our method uses a similar bottleneck structure 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 (Section 3). A comtenporary extension of adapter is COMPACTER (Mahabadi et al., 2021), which essentially parametrizes the adapter layers using Kronecker products with some predetermined weight sharing scheme. Similarly, combining LoRA with other tensor product-based methods could potentially improve its parameter efficiency, which we leave to future work. More recently, many proposed optimizing the input word embeddings in lieu of fine-tuning, akin to a continuous and differentiable generalization of prompt engineering (Li & Liang, 2021; Lester et al., 2021; Hambardzumyan et al., 2020; Liu et al., 2021). We include comparisons with Li & Liang (2021) 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 (Li et al., 2016; Cai et al., 2010; Li et al., 2018b; Grasedyck et al., 2013). 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 (Oymak et al., 2019). Some prior works even explicitly impose the low-rank constraint when training the original neural network (Sainath et al., 2013; Povey et al., 2018; Zhang et al., 2014; Jaderberg et al., 2014; Zhao et al., 2016; Khodak et al., 2021; Denil et al., 2014); however, to the best of our knowledge, none of these works considers low-rank update to a frozen model 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 (Allen-Zhu et al., 2019; Li & Liang, 2018) when the underlying concept class has certain low-rank structure (Ghorbani et al., 2020; Allen-Zhu & Li, 2019; Allen-Zhu & Li, 2020a). Another theoretical result in Allen-Zhu & Li (2020b) 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.

7 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 focus our study on GPT-3 175B, where we achieved the largest reduction of trainable parameters (up to 10,000×) without adversely affecting task performances.

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

模型，於大量文本上訓練——在先於通用領域資料的預訓練後，再對任務特定資料進行微調，相較於直接在任務特定資料上訓練，可帶來顯著的效能提升。訓練更大的 Transformer 通常會帶來更好的表現，且仍是積極的研究方向。GPT-3 (Brown et al., 2020) 是迄今為止參數量最大的單一 Transformer 語言模型，具 1750 億個參數。

提示工程與微調。雖然 GPT-3 175B 能夠僅以少量額外的訓練範例調整其行為，但結果高度依賴輸入的提示（Brown et al., 2020）。這就需要以經驗為基礎的撰寫與格式化提示的技術，以在期望任務上最大化模型的表現，這被稱為提示工程或提示駭客。微調則是將在通用領域上預訓練的模型重新訓練到特定任務（Devlin et al. (2019b); Radford et al. (a)）。其變體包括僅學習參數的子集（Devlin et al. (2019b); Collobert & Weston (2008)），但實務上常常會重新訓練所有參數以最大化下游效能。然而，GPT-3 175B 的龐大規模使得以通常方式進行微調變得具有挑戰性，因為它會產生巨大的檢查點且進入門檻的硬體需求很高，因為其記憶體佔用與預訓練時相同。

Parameter-Efficient Adaptation. 許多人提出在神經網路的既有層之間插入 *adapter* 層（Houlsby et al., 2019；Rebuffi et al., 2017；Lin et al., 2020）。我們的方法使用類似的瓶頸結構，對權重更新施加低秩約束。關鍵的功能差異在於，我們學得的權重可以在推論期間與主要權重合併，因此不會增加任何延遲，而 adapter 層則沒有這個特性（見第3節）。一個當代的 adapter 擴展是 COMPACTER （ Mahabadi et al., 2021 ），其本質上是以預先設定的權重共享方案用 Kronecker 乘積來參數化 adapter 層。類似地，將 LoRA 與其他基於張量乘積的方法結合，可能進一步提升參數效率，這部分留待未來工作探討。近來也有許多工作提出優化輸入詞嵌入以取代微調，類似於 prompt engineering 的連續且可微分的泛化（Li &Liang, 2021；Lester et al., 2021；Hambardzumyan et al., 2020；Liu et al., 2021）。我們在實驗部分包含了與 Li & Liang (2021) 的比較。然而，這類方法只能透過在提示中使用更多特殊標記來擴展，而當位置嵌入被學習時，這些標記會佔用可供任務標記使用的序列長度。

深度學習中的低階結構。低階結構在機器學習中非常常見。許多機器學習問題具有某種內在的低階結構 (Li et al., 2016; Cai et al., 2010; Li et al., 2018b; Grasedyck et al., 2013)。此外，已知在許多深度學習任務中，特別是那些高度過參數化的神經網路，經訓練後所學得的網路會呈現低階性質 (Oymak et al., 2019)。一些先前的工作甚至在訓練原始神經網路時明確加入低階約束 (Sainath et al., 2013; Povey et al., 2018; Zhang et al., 2014; Jaderberg et al., 2014; Zhao et al., 2016; Khodak et al., 2021; Denil et al., 2014)；然而，據我們所知，這些工作中沒有一篇考慮對凍結模型進行低階更新以便適應下游任務。在理論文獻中，已知當底層概念類別具有某種低階結構時，神經網路的表現優於其他經典學習方法，包括對應的（有限寬度）神經切線核 (Allen-Zhu et al., 2019; Li & Liang, 2018; Ghorbani et al., 2020; Allen-Zhu & Li, 2019; Allen-Zhu & Li, 2020a)。Allen-Zhu & Li (2020b) 的另一項理論結果則表明低階適配對抗性訓練可能有用。總之，我們認為文獻充分支持我們所提出的低階適配更新的動機。

7 理解 低秩 更新

鑑於 LoRA 在實務上的優勢，我們希望進一步說明從下游任務學得的低秩調適（low-rank adaptation）之性質。請注意，低秩結構不僅降低了硬體門檻，讓我們能夠並行執行多個實驗，還能更容易解釋更新權重與預訓練權重之間的關聯。我們的研究重點放在 GPT-3 175B，上面我們達成了可訓練參數數量的最大縮減（最高達 10,000×），且對任務效能沒有不良影響。

我們進行一系列實證研究以回答以下問題：1) 在參數預算限制下，應該在預訓練 *Transformer* 中微調哪些權重矩陣子集？

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.

7.1 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 Section 4.2, we only consider weight matrices in the self-attention module. We set a parameter budget of 18M (roughly 35MB if stored in FP16) on GPT-3 175B, which corresponds to $r = 8$ if we adapt one type of attention weights or $r = 4$ if we adapt two types, for all 96 layers. The result is presented in Table 5.

	# of Trainable Parameters = 18M						
Weight Type Rank r	W_q 8	W_k 8	W_v 8	W_o 8	W_q, W_k 4	W_q, W_v 4	W_q, W_k, W_v, W_o 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.

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.

7.2 What is the Optimal Rank r for LoRA?

We turn our attention to the effect of rank r on model performance. We adapt $\{W_q, W_v\}$, $\{W_q, W_k, W_v, W_c\}$, and just W_q for a comparison.

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

Table 6: 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 conduct a similar experiment on GPT-2 in Section H.2.

Table 6 shows that, surprisingly, LoRA already performs competitively with a very small r (more so for $\{W_q, W_v\}$ than just W_q). This suggests the update matrix ΔW could have a very small "intrinsic rank".[6] 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 a more meaningful subspace, which suggests that a low-rank adaptation matrix is sufficient.

[6]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.

以最大化下游任務表現？2) 所謂的「最佳」適配矩陣 ΔW 真的存在秩虧嗎？如果是，實務上應該使用何種良好秩值？3) ΔW 與W之間有何關聯？ ΔW是否與 W高度相關？相比之下， ΔW與 W的量級差別有多大？

我們認為對於問題 (2) 和 (3) 的回答，能夠闡明使用預訓練語言模型於下游任務時的基本原則，這是自然語言處理領域的一個關鍵議題。

7.1 應該對 Transformer 的哪些權重矩陣應用 LoRA？

在參數預算有限的情況下，應該用 LoRA 調整哪種類型的權重以在下游任務上取得最佳表現？如第 4.2 節所述，我們僅考慮自注意力模組中的權重矩陣。我們在 GPT-3 175B 上設了一個 18M 的參數預算（若以 FP16 存儲約為 35MB），這相當於在所有 96 層中，若只調整一種類型的注意力權重則為 $r = 8$ ，若調整兩種類型則為 $r = 4$。結果如表 5 所示。

	可訓練參數數量 = 18M						
權重類型 排名 r	W_q 8	W_k 8	W_v 8	W_o 8	W_q, W_k 4	W_q，W_v 4	W_q，W_k，W_v，W_o 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**

表 5：在 GPT-3 中對不同類型的注意力權重應用 LoRA 並保持相同數量可訓練參數後，於 WikiSQL 和 MultiNLI 上的驗證準確率。同時調整 W_q 和 W_v 可獲得整體最佳表現。我們發現對於給定資料集，各隨機種子的標準差相當一致，並於第一欄報告。

請注意，將所有參數放在 ΔW_q 或 ΔW_k 中會導致性能顯著下降，而同時調整 W_q 和 W_v 則能得到最佳結果。這表示即使秩為四， ΔW 中也能捕捉到足夠的資訊，因此與其只用較高的秩去調整單一類型的權重，不如同時調整更多的權重矩陣。

7.2 對 LoRA 而言最佳的秩 r 是什麼？

我們將注意力轉向秩 r 對模型效能的影響。我們比較了調整 $\{W_q, W_v\}$、$\{W_q, W_k, W_v, W_c\}$，以及僅調整 W_q 的情況。

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

表 6：在 WikiSQL 和 MultiNLI 上使用不同秩 r 的驗證準確率。令我們驚訝的是，秩小到 1 就足以在這些資料集上調適 W_q 與 W_v ，而僅訓練 W_q 則需要更大的 r。我們在 H.2 節對 GPT-2 進行了類似實驗。

表 6 顯示，令人驚訝的是，LoRA 在非常小的 r 下已能表現得具有競爭力（對 $\{W_q, W_v\}$ 的影響比單純對 W_q 更明顯）。這暗示更新矩陣 ΔW 可能具有非常小的「內在階數」。[6] 為了進一步支持這一發現，我們檢查不同 r 選擇及不同隨機種子所學到子空間的重疊。我們主張增加 r 並不會覆蓋更具意義的子空間，這表明低階適配矩陣就足夠了。

[6]然而，我們並不期待小型的 r 能適用於所有任務或資料集。考慮以下的思考實驗：若下游任務使用的語言與用於預訓練的語言不同，重新訓練整個模型（類似於具有 $r = d_{model}$）的 LoRA）確實可能優於採用小型 r 的 LoRA。

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 $U_{A_{r=64}}$.[7] We hope to answer: how much of the subspace spanned by the top i singular vectors in $U_{A_{r=8}}$ (for $1 \le i \le 8$) is contained in the subspace spanned by top j singular vectors of $U_{A_{r=64}}$ (for $1 \le j \le 64$)? We measure this quantity with a normalized subspace similarity based on the Grassmann distance (See Appendix G 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 Figure 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 Section H.1.

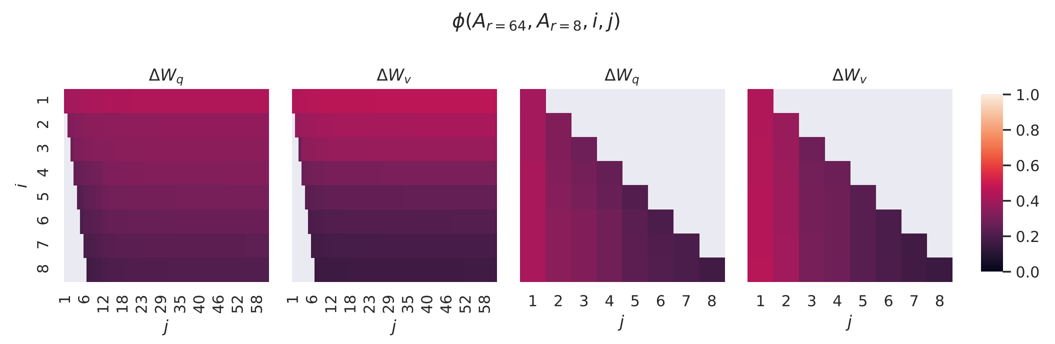


Figure 3: Subspace similarity between column vectors of $A_{r=8}$ and $A_{r=64}$ for both ΔW_q and ΔW_v. The third and the fourth figures zoom in on the lower-left triangle in the first two figures. The top directions in $r = 8$ are included in $r = 64$, and vice versa.

We make an *important observation* from Figure 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, Figure 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.

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 Figure 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 6. As a comparison, we also plot two random Gaussian matrices, which do not share any common singular value directions with each other.

7.3 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,

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

不同之間的子空間相似性 r。 給定 $A_{r=8}$ 和 $A_{r=64}$ ，它們是使用秩為 $r = 8$ 與 64 的學得適配矩陣，並採用 相同的預訓練模型，我們對其進行奇異值分解並得到右奇異單位矩陣 $U_{A_{r=8}}$ 與 $U_{A_{r=64}}$.[7] 。我們希望回答：在 $U_{A_{r=8}}$ 中由前 i 個奇異向量所張成的子空間，有多少被包含在 $U_{A_{r=64}}$ (for $1 \le j \le 64$)? 的前 j 個奇異向量所張成的子空間中？我們以基於 Grassmann 距離的正規化子空間相似性來衡量此量（詳見附錄 G 以獲得更形式化的討論）。

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

其中 $U_{A_{r=8}}^{i}$ 表示 $U_{A_{r=8}}$ 的欄位，對應於前 i 個奇異向量。

ϕ $(\cdot)$ 的範圍為 $[0, 1]$, ，其中 1 代表子空間完全重疊，0 代表完全分離。關於當我們改變 i 和 j時 ϕ 如何變化，見圖 3。因篇幅限制，我們僅檢視第 48 層（共 96 層）但如 H.1 節所示，結論對其他層也成立。

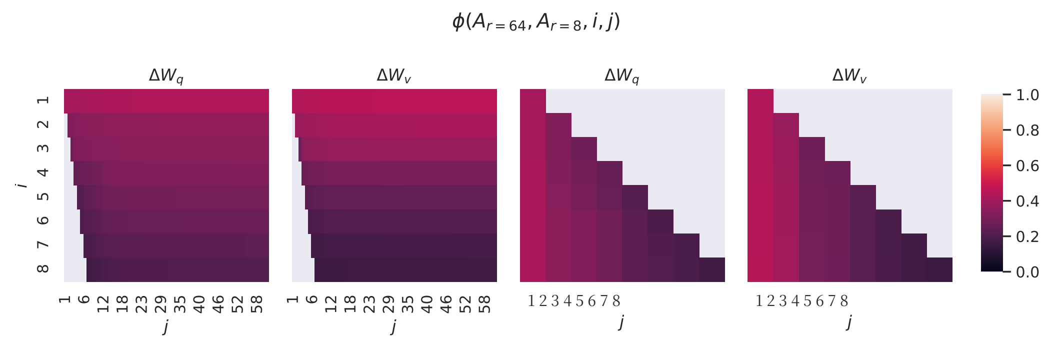


圖 3： $A_{r=8}$ 與 $A_{r=64}$ 的列向量之子空間相似性，對於 ΔW_q 與 ΔW_v 均適用。第三與第四張圖放大第一、二張圖的左下三角。 $r = 8$中的主要方向包含於 $r = 64$，反之亦然。

我們從圖 3 中得到一個重要觀察。

> 對應於最大奇異向量的方向在$A_{r=8}$ 與 $A_{r=64}$之間有顯著重疊，而其他方向則沒有。具體而言， $A_{r=8}$與 ΔW_v 的 ΔW_v （分別）以及 $A_{r=64}$ 的 ΔW_q（分別）共享維度為 1 的子空間，正規化相似度為 > 0.5，這解釋了為何 $r = 1$ 在我們對 GPT-3 的下游任務中表現相當良好。

由於 $A_{r=8}$ 和 $A_{r=64}$ 都是使用相同的預訓練模型學得，圖 3 顯示 $A_{r=8}$ 和 $A_{r=64}$ 的頂端奇異向量方向是最有用的，而其他方向很可能主要包含訓練過程中累積的大部分隨機雜訊。因此，適配矩陣確實可以具有非常低的秩。

不同隨機種子之間的子空間相似性。我們進一步透過繪製兩個以不同隨機種子執行且使用 $r = 64$ 的運行之間的正規化子空間相似性來證實這一點，見圖 4。ΔW_q 似乎比 ΔW_v 具有更高的「內在秩」，因為對於 ΔW_q，兩次運行學到的共同奇異值方向更多，這與我們在表 6 中的實證觀察一致。作為比較，我們也繪製了兩個隨機高斯矩陣，它們彼此之間不共享任何共同的奇異值方向。

7.3 適配矩陣 ΔW 與 W 相比如何？

我們進一步探討 ΔW 與 W 之間的關係。特別是， ΔW 是否與 W 高度相關？（或者從數學上說， ΔW 是否大部分包含在 W 的主要奇異方向中？）此外，

請注意，可以對 B 與左奇異單位矩陣進行類似分析——我們在實驗中仍然採用 A 。

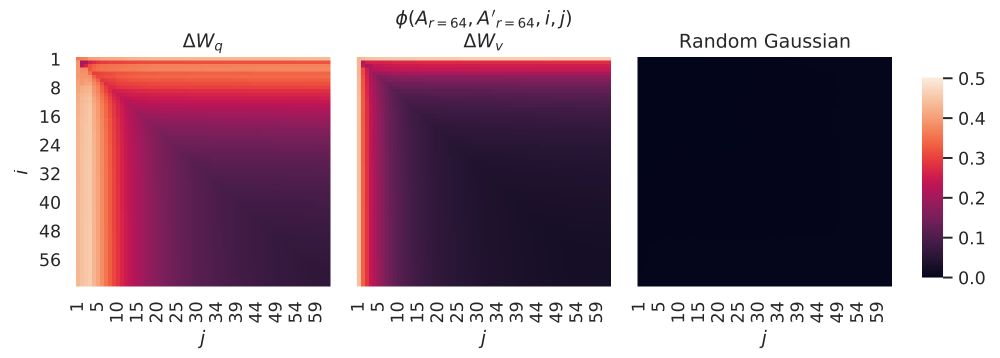


Figure 4: **Left and Middle:** Normalized subspace similarity between the column vectors of $A_{r=64}$ from two random seeds, for both ΔW_q and ΔW_v in the 48-th layer. **Right:** the same heat-map between the column vectors of two random Gaussian matrices. See Section H.1 for other layers.

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 7: 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 7. 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 Section H.4 for why $r = 64$ has a smaller amplification factor. We also provide a visualization in Section H.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.*

8 Conclusion and Future Work

Fine-tuning enormous language models is prohibitively expensive in terms of the hardware required and the storage/switching cost for hosting independent instances for different tasks. We propose LoRA, an efficient adaptation strategy that neither introduces inference latency nor reduces input sequence length while retaining high 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 language models, the proposed principles are generally applicable to any neural networks with dense layers.

There are many directions for future works. 1) LoRA can be combined with other efficient adaptation methods, potentially providing orthogonal improvement. 2) The mechanism behind fine-tuning or LoRA is far from clear – how are features learned during pre-training transformed to do well on downstream tasks? We believe that LoRA makes it more tractable to answer this than full fine-

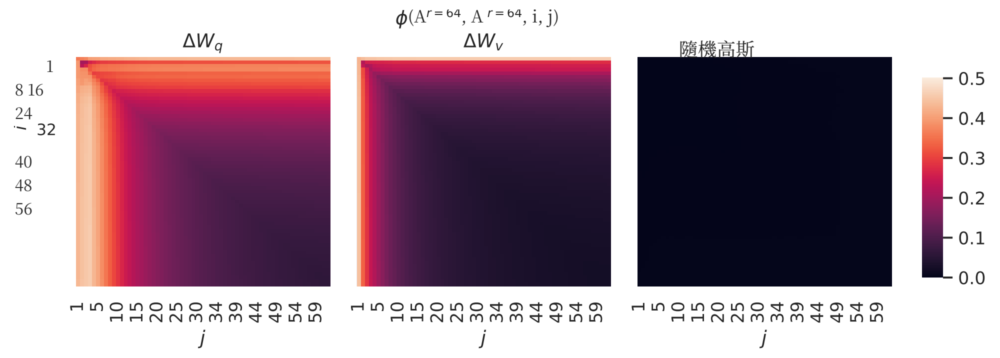


Figure 4: **左與中：**在第48層中，對於 ΔW_q 與 ΔW_v，來自兩個隨機種子的 $A_{r=64}$ 欄向量之間的規一化子空間相似度。**右：**兩個隨機高斯矩陣欄向量之間的相同熱圖。其他層見附錄 H.1。

與 W 中相應方向相比，ΔW 在多大程度上「較大」？這可以揭示調適預訓練語言模型的潛在機制。

為了回答這些問題，我們將 W 投影到 r 維的 ΔW 子空間，通過計算 $U^\top W V^\top$，其中 U/V 分別為 ΔW 的左／右奇異向量矩陣。然後，我們比較 $\|U^\top W V^\top\|_F$ 與 $\|W\|_F$ 之間的 Frobenius 範數。作為比較，我們也透過將 U, V 替換為 W 的前 r 個奇異向量或一個隨機矩陣的方式來計算 $\|U^\top W V^\top\|_F$。

	$r = 4$			$r = 64$		
	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
$\|\|W_q\|\|_F = 61.95$	$\|\|\Delta W_q\|\|_F = 6.91$			$\|\|\Delta W_q\|\|_F = 3.57$		

表 7：$U^\top W_q V^\top$ 的 Frobenius 範數，其中 U 和 V 分別是 (1) ΔW_q、(2) W_q 或 (3) 隨機矩陣的左/右前 r 個奇異向量方向。權重矩陣取自 GPT-3 的第 48 層。

我們從表 7 中得出幾項結論。首先，ΔW 與 W 的相關性比隨機矩陣更強，這表明 ΔW 放大了已存在於 W 中的某些特徵。其次，ΔW 並非簡單重複 W 的前導奇異方向，而是僅放大那些在 W 中未被強調的方向。第三，放大因子相當巨大：$21.5 \approx 6.91/0.32$ 用於 $r = 4$。關於為何 $r = 64$ 的放大因子較小，請參見 H.4 節。我們也在 H.3 節提供了視覺化說明，展示當包含來自 W_q 的更多前導奇異方向時相關性如何變化。這表明低秩調整矩陣可能放大那些在通用預訓練模型中已學到但未被強調的、對特定下游任務重要的特徵。

8 結論與未來工作

對巨型語言模型進行微調在硬體需求與為不同任務托管獨立實例所需的儲存/切換成本方面都高得令人望而卻步。我們提出 LoRA，一種高效的適配策略，既不增加推理延遲，也不縮短輸入序列長度，同時能保留高模型品質。重要的是，當作為服務部署時，它透過共享絕大多數模型參數，允許快速切換任務。雖然我們專注於 Transformer 語言模型，但所提出的原則一般也適用於任何具有密集層的神經網路。

未來工作有多個方向。1) LoRA 可以與其他高效的調適方法結合，可能帶來互補性的改進。2) 微調或 LoRA 背後的機制仍然不明確——在預訓練期間學到的特徵如何被轉換以在下游任務上表現良好？我們相信，相較於完整微調，LoRA 讓回答這個問題變得更可處理，

tuning. 3) We mostly depend on heuristics to select the weight matrices to apply LoRA to. Are there more principled ways to do it? 4) Finally, the rank-deficiency of ΔW suggests that W could be rank-deficient as well, which can also be a source of inspiration for future works.

References

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`.

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`.

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

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

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`.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016.

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`.

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.

Daniel Cer, Mona Diab, Eneko Agirre, Inigo Lopez-Gazpio, and Lucia Specia. Semeval-2017 task 1: Semantic textual similarity multilingual and crosslingual focused evaluation. *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, 2017. doi: 10.18653/v1/s17-2001. URL `http://dx.doi.org/10.18653/v1/S17-2001`.

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, pp. 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`.

Misha Denil, Babak Shakibi, Laurent Dinh, Marc'Aurelio Ranzato, and Nando de Freitas. Predicting parameters in deep learning, 2014.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019a.

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 2019b. URL `http://arxiv.org/abs/1810.04805`. arXiv: 1810.04805.

William B. Dolan and Chris Brockett. Automatically constructing a corpus of sentential paraphrases. In *Proceedings of the Third International Workshop on Paraphrasing (IWP2005)*, 2005. URL `https://aclanthology.org/I05-5002`.

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*, pp. 124–133, 2017.

微調。3) 我們主要依賴經驗法則來選擇要對哪些權重矩陣應用 LoRA。是否有更有原則的方法來執行這個選擇？4) 最後， ΔW 的秩不滿足意味著 W 也可能為秩不滿足，這也可以成為未來工作的靈感來源。

參考文獻

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`.

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`.

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

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

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`.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016.

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`.

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.

Daniel Cer, Mona Diab, Eneko Agirre, Inigo Lopez-Gazpio, and Lucia Specia. Semeval-2017 task 1: Semantic textual similarity multilingual and crosslingual focused evaluation. *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, 2017. doi: 10.18653/v1/s17-2001. URL `http://dx.doi.org/10.18653/v1/S17-2001`.

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, pp. 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`.

Misha Denil, Babak Shakibi, Laurent Dinh, Marc'Aurelio Ranzato, and Nando de Freitas. Predicting parameters in deep learning, 2014.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019a.

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 2019b. URL `http://arxiv.org/abs/1810.04805`. arXiv: 1810.04805.

William B. Dolan and Chris Brockett. Automatically constructing a corpus of sentential paraphrases. In *Proceedings of the Third International Workshop on Paraphrasing (IWP2005)*, 2005. URL `https://aclanthology.org/I05-5002`.

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*, pp. 124–133, 2017.

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

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`.

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

Jihun Ham and Daniel D. Lee. Grassmann discriminant analysis: a unifying view on subspace-based learning. In *ICML*, pp. 376–383, 2008. URL `https://doi.org/10.1145/1390156.1390204`.

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.

Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. Deberta: Decoding-enhanced bert with disentangled attention, 2021.

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`.

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

Mikhail Khodak, Neil Tenenholtz, Lester Mackey, and Nicolò Fusi. Initialization and regularization of factorized neural layers, 2021.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. Gshard: Scaling giant models with conditional computation and automatic sharding, 2020.

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.

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

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`.

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

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

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*, pp. 2–47. PMLR, 2018b.

Zhaojiang Lin, Andrea Madotto, and Pascale Fung. Exploring versatile generative language model via parameter-efficient transfer learning. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pp. 441–459, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.findings-emnlp.41. URL `https://aclanthology.org/2020.findings-emnlp.41`.

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

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`.

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

Jihun Ham and Daniel D. Lee. Grassmann discriminant analysis: a unifying view on subspace-based learning. In *ICML*, pp. 376–383, 2008. URL `https://doi.org/10.1145/1390156.1390204`.

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.

Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. Deberta: Decoding-enhanced bert with disentangled attention, 2021.

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`.

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

Mikhail Khodak, Neil Tenenholtz, Lester Mackey, and Nicolò Fusi. Initialization and regularization of factorized neural layers, 2021.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. Gshard: Scaling giant models with conditional computation and automatic sharding, 2020.

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.

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

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`.

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

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

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*, pp. 2–47. PMLR, 2018b.

Zhaojiang Lin, Andrea Madotto, and Pascale Fung. Exploring versatile generative language model via parameter-efficient transfer learning. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pp. 441–459, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.findings-emnlp.41. URL `https://aclanthology.org/2020.findings-emnlp.41`.

14

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.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach, 2019.

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

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization, 2019.

Rabeeh Karimi Mahabadi, James Henderson, and Sebastian Ruder. Compacter: Efficient low-rank hypercomplex adapter layers, 2021.

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.

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

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.

Jonas Pfeiffer, Aishwarya Kamath, Andreas Rücklé, Kyunghyun Cho, and Iryna Gurevych. Adapterfusion: Non-destructive task composition for transfer learning, 2021.

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*, pp. 3743–3747, 2018.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving Language Understanding by Generative Pre-Training. pp. 12, a.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language Models are Unsupervised Multitask Learners. pp. 24, b.

Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for squad. *CoRR*, abs/1806.03822, 2018. URL `http://arxiv.org/abs/1806.03822`.

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.

Andreas Rücklé, Gregor Geigle, Max Glockner, Tilman Beck, Jonas Pfeiffer, Nils Reimers, and Iryna Gurevych. Adapterdrop: On the efficiency of adapters in transformers, 2020.

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*, pp. 6655–6659. IEEE, 2013.

Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism, 2020.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pp. 1631–1642, Seattle, Washington, USA, October 2013. Association for Computational Linguistics. URL `https://aclanthology.org/D13-1170`.

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.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach, 2019.

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

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization, 2019.

Rabeeh Karimi Mahabadi, James Henderson, and Sebastian Ruder. Compacter: Efficient low-rank hypercomplex adapter layers, 2021.

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.

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

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.

Jonas Pfeiffer, Aishwarya Kamath, Andreas Rücklé, Kyunghyun Cho, and Iryna Gurevych. Adapterfusion: Non-destructive task composition for transfer learning, 2021.

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*, pp. 3743–3747, 2018.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving Language Understanding by Generative Pre-Training. pp. 12, a.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language Models are Unsupervised Multitask Learners. pp. 24, b.

Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for squad. *CoRR*, abs/1806.03822, 2018. URL `http://arxiv.org/abs/1806.03822`.

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.

Andreas Rücklé, Gregor Geigle, Max Glockner, Tilman Beck, Jonas Pfeiffer, Nils Reimers, and Iryna Gurevych. Adapterdrop: On the efficiency of adapters in transformers, 2020.

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*, pp. 6655–6659. IEEE, 2013.

Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism, 2020.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pp. 1631–1642, Seattle, Washington, USA, October 2013. Association for Computational Linguistics. URL `https://aclanthology.org/D13-1170`.

15

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*, pp. 6000–6010, 2017.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding, 2019.

Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. Superglue: A stickier benchmark for general-purpose language understanding systems, 2020.

Alex Warstadt, Amanpreet Singh, and Samuel R Bowman. Neural network acceptability judgments. *arXiv preprint arXiv:1805.12471*, 2018.

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)*, pp. 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.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 38–45, Online, October 2020. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/2020.emnlp-demos.6.

Greg Yang and Edward J. Hu. Feature Learning in Infinite-Width Neural Networks. *arXiv:2011.14522 [cond-mat]*, May 2021. URL http://arxiv.org/abs/2011.14522. arXiv: 2011.14522.

Elad Ben Zaken, Shauli Ravfogel, and Yoav Goldberg. Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models, 2021.

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)*, pp. 185–189. IEEE, 2014.

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)*, pp. 5005–5009. IEEE, 2016.

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 8, 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 (Brown et al., 2020). For MNLI-matched, we use two demonstrations per class and six in-context examples in total.

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*, pp. 6000–6010, 2017.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding, 2019.

Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. Superglue: A stickier benchmark for general-purpose language understanding systems, 2020.

Alex Warstadt, Amanpreet Singh, and Samuel R Bowman. Neural network acceptability judgments. *arXiv preprint arXiv:1805.12471*, 2018.

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)*, pp. 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.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 38–45, Online, October 2020. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/2020.emnlp-demos.6.

Greg Yang and Edward J. Hu. Feature Learning in Infinite-Width Neural Networks. *arXiv:2011.14522 [cond-mat]*, May 2021. URL http://arxiv.org/abs/2011.14522. arXiv: 2011.14522.

Elad Ben Zaken, Shauli Ravfogel, and Yoav Goldberg. Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models, 2021.

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)*, pp. 185–189. IEEE, 2014.

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)*, pp. 5005–5009. IEEE, 2016.

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.

大型語言模型仍然需要參數更新

少量示例學習（few-shot learning）或提示工程在我們只有少數訓練樣本時非常有利。然而在實務上，針對對效能敏感的應用，我們通常能夠整理出數千或更多的訓練範例。如表 8 所示，與少量示例學習相比，微調在大與小資料集上都能顯著提升模型表現。我們採用 GPT-3 論文（Brown 等，2020）中 RTE 的 GPT-3 few-shot 結果。對於 MNLI-matched，我們每個類別使用兩個示範，總共六個上下文範例。

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 8: Fine-tuning significantly outperforms few-shot learning on GPT-3 (Brown et al., 2020).

B Inference Latency Introduced by Adapter Layers

Adapter layers are external modules added to a pre-trained model in a *sequential* manner, whereas our proposal, LoRA, can be seen as external modules added in a parallel manner. Consequently, adapter layers must be computed in addition to the base model, inevitably introducing additional latency. While as pointed out in Rücklé et al. (2020), the latency introduced by adapter layers can be mitigated when the model batch size and/or sequence length is large enough to full utilize the hardware parallelism. We confirm their observation with a similar latency study on GPT-2 medium and point out that there are scenarios, notably online inference where the batch size is small, where the added latency can be significant.

We measure the latency of a single forward pass on an NVIDIA Quadro RTX8000 by averaging over 100 trials. We vary the input batch size, sequence length, and the adapter bottleneck dimension r. We test two adapter designs: the original one by Houlsby et al. (2019), which we call Adapter^H, and a recent, more efficient variant by Lin et al. (2020), which we call Adapter^L. See Section 5.1 for more details on the designs. We plot the slow-down in percentage compared to the no-adapter baseline in Figure 5.

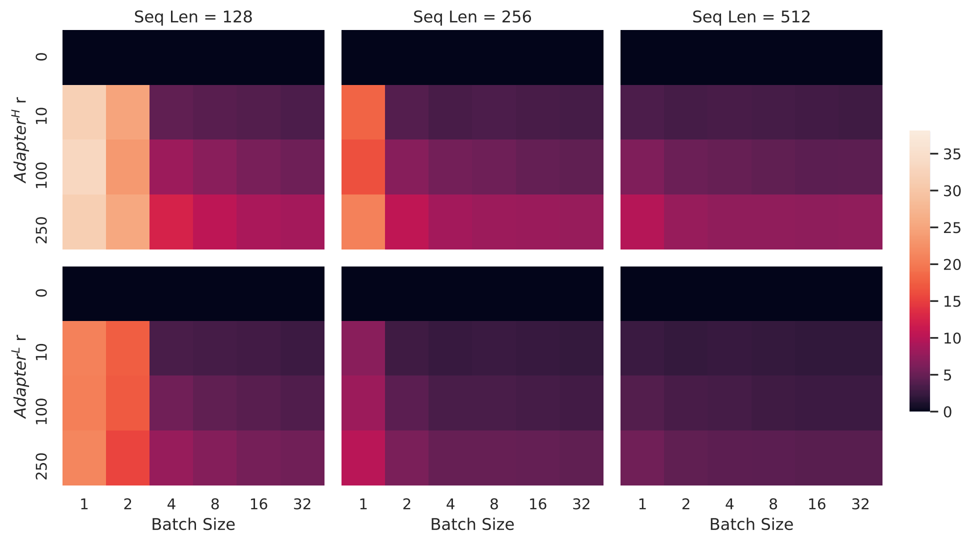


Figure 5: Percentage slow-down of inference latency compared to the no-adapter ($r = 0$) baseline. The top row shows the result for Adapter^H and the bottom row Adapter^L. Larger batch size and sequence length help to mitigate the latency, but the slow-down can be as high as over 30% in an online, short-sequence-length scenario. We tweak the colormap for better visibility.

C Dataset Details

GLUE Benchmark is a wide-ranging collection of natural language understanding tasks. It includes MNLI (inference, Williams et al. (2018)), SST-2 (sentiment analysis, Socher et al. (2013)), MRPC (paraphrase detection, Dolan & Brockett (2005)), CoLA (linguistic acceptability, Warstadt et al. (2018)), QNLI (inference, Rajpurkar et al. (2018)), QQP[8] (question-answering), RTE (inference),

[8]https://quoradata.quora.com/First-Quora-Dataset-Release-Question-Pairs

方法	MNLI-m（驗證準確率/%）	RTE（驗證準確率/%）
GPT-3 少樣本學習	40.6	69.0
GPT-3 微調	89.5	85.4

表 8：在 GPT-3（Brown 等，2020）上，微調明顯優於少樣本學習。

B 推論延遲由 Adapter 層引入

Adapter layers 是以 順序式 方式加入預訓練模型的外部模組，而我們提出的 LoRA 則可視為以並行方式加入的外部模組。因此，adapter layers 必須額外與基礎模型一同計算，難免會引入額外延遲。如 R¨uckl´e 等（2020）所指出，當模型批次大小和／或序列長度夠大以充分利用硬體並行性時，adapter layers 引入的延遲可被緩解。我們在 GPT-2 medium 上以類似的延遲研究證實了他們的觀察，並指出在某些情境下（尤其是批次大小很小的線上推論）所增加的延遲可能相當顯著。

我們在 NVIDIA Quadro RTX8000 上測量單次前向傳播的延遲，取 100 次試驗的平均值。我們變化輸入的批次大小、序列長度以及 adapter 的瓶頸維度r。我們測試兩種 adapter 設計：Houlsby 等人（2019）提出的原始設計，稱為 Adapter^H，以及 Lin 等人（2020）提出的較有效率的變體，稱為 Adapter^L。設計細節見第 5.1 節。我們在圖 5 中繪出相較於無 adapter 基線的延遲百分比減慢情況。

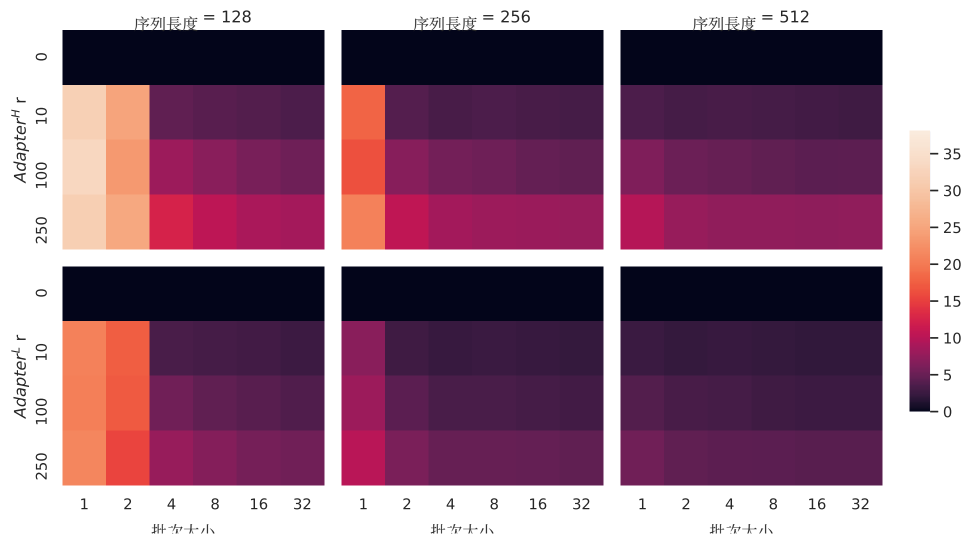


圖 5：相較於無 adapter（$r = 0$）基線的推論延遲百分比減慢。上排顯示 Adapter^H 的結果，下排顯示 Adapter^L。較大的批次大小與序列長度有助於減輕延遲，但在線上、短序列長度的情境下，減慢幅度可高達 30% 以上。我們調整了色彩映射以提升可見度。

C 資料集詳情

GLUE Benchmark 是一個廣泛的自然語言理解任務集合。它包含 MNLI（推論，Williams 等人 (2018)）、SST-2（情感分析，Socher 等人 (2013)）、MRPC（同義句檢測，Dolan & Brockett (2005)）、CoLA（語言可接受性，Warstadt 等人 (2018)）、QNLI（推論，Rajpurkar 等人 (2018)）、QQP[8] (問答)、RTE（推論），

[8]https://quoradata.quora.com/First-Quora-Dataset-Release-Question-Pairs

and STS-B (textual similarity, Cer et al. (2017)). The broad coverage makes GLUE benchmark a standard metric to evaluate NLU models such as RoBERTa and DeBERTa. The individual datasets are released under different permissive licenses.

WikiSQL is introduced in Zhong et al. (2017) 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 $x = \{\text{table schema, query}\}$ and target as $y = \{\text{SQL}\}$. The dataset is release under the BSD 3-Clause License.

SAMSum is introduced in Gliwa et al. (2019) and contains $14,732/819$ training/test examples. It consists of staged chat conversations between two people and corresponding abstractive summaries written by linguists. We encode context as "\n" concatenated utterances followed by a "\n\n", and target as $y = \{\text{summary}\}$. The dataset is released under the non-commercial licence: Creative Commons BY-NC-ND 4.0.

E2E NLG Challenge was first introduced in Novikova et al. (2017) 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 roughly $42,000$ training, $4,600$ validation, and $4,600$ 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 Nan et al. (2020). 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 (Gardent et al., 2017). 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.

D Hyperparameters Used in Experiments

D.1 RoBERTa

We train using AdamW with a linear learning rate decay schedule. We sweep learning rate, number of training epochs, and batch size for LoRA. Following Liu et al. (2019), we initialize the LoRA modules to our best MNLI checkpoint when adapting to MRPC, RTE, and STS-B, instead of the usual initialization; the pre-trained model stays frozen for all tasks. We report the median over 5 random seeds; the result for each run is taken from the best epoch. For a fair comparison with the setup in Houlsby et al. (2019) and Pfeiffer et al. (2021), we restrict the model sequence length to 128 and used a fixed batch size for all tasks. Importantly, we start with the pre-trained RoBERTa large model when adapting to MRPC, RTE, and STS-B, instead of a model already adapted to MNLI. The runs with this restricted setup are marked with †. See the hyperparameters used in our runs in Table 9.

D.2 DeBERTa

We again train using AdamW with a linear learning rate decay schedule. Following He et al. (2021), we tune learning rate, dropout probability, warm-up steps, and batch size. We use the same model sequence length used by (He et al., 2021) to keep our comparison fair. Following He et al. (2021), we initialize the LoRA modules to our best MNLI checkpoint when adapting to MRPC, RTE, and STS-B, instead of the usual initialization; the pre-trained model stays frozen for all tasks. We report the median over 5 random seeds; the result for each run is taken from the best epoch. See the hyperparameters used in our runs in Table 10.

以及 STS-B（文本相似度，Cer 等人 (2017)）。廣泛的涵蓋範圍使 GLUE benchmark 成為評估如 RoBERTa 和 DeBERTa 等 NLU 模型的標準指標。各個資料集在不同的寬鬆授權下釋出。

WikiSQL 由 Zhong 等人（2017）提出，包含 56,355/8,421 筆訓練/驗證 範例。該任務是從自然語言問題與表格結構生成 SQL 查詢。我們將上下文編碼為 $x = \{\text{table schema,query}\}$ ，目標編碼為 $y = \{\text{SQL}\}$。此資料集依 BSD 3-Clause License 授權釋出。

SAMSum 由 Gliwa 等人（2019）提出，包含 14,732/819 筆訓練/測試 範例。資料由兩人之間分階段的聊天對話及語言學家撰寫的摘要組成。我們將上下文編碼為 ”\n” 串接的發言，後接 ”\n\n” ，目標編碼為 $y = \{\text{summary}\}$。此資料集依非商業授權釋出：Creative Commons BY-NC-ND 4.0。

E2E NLG Challenge 最早由 Novikova 等人（2017）提出，作為訓練端到端、資料驅動自然語言生成系統的資料集，並常用於資料到文字的評估。E2E 資料集包含約 42,000 筆訓練、4,600 筆驗證及 4,600 筆測試範例，來源於餐廳領域。每個當作輸入的來源表可能有多個參考。每個樣本輸入 (x, y) 由一連串的欄位-值配對組成，並附有對應的自然語言參考文本。此資料集依 Creative Commons BY-NC-SA 4.0 授權釋出。

DART 是一個開放領域的資料到文字（data-to-text）資料集，詳見 Nan et al. (2020)。DART 的輸入以 ENTITY — RELATION — ENTITY 的三元組序列呈現。DART 總共有 $82K$ 個範例，相較於 E2E，這是一個規模顯著更大且更複雜的資料到文字任務。該資料集以 MIT 授權釋出。

WebNLG 是另一個常用於資料到文字評估的資料集（Gardent et al., 2017）。WebNLG 總共有 $22K$ 個範例，包含 14 個不同類別，其中九個在訓練時可見。由於 14 個類別中有五個在訓練時不可見但出現在測試集中，評估通常依「可見」類別（S）、「不可見」類別（U）與「全部」（A）來區分。每個輸入範例以 SUBJECT — PROPERTY — OBJECT 的三元組序列表示。該資料集以 Creative Commons BY-NC-SA 4.0 授權釋出。

D HYPERPARAMETERS USEDIN EXPERIMENTS

D.1 ROBERTA

我們使用 AdamW 並搭配線性學習率衰減排程進行訓練。我們在 LoRA 上搜尋學習率、訓練時代數與批次大小。依照 Liu et al. (2019)，在調整到 MRPC、RTE 與 STS-B 時，我們將 LoRA 模組以在 MNLI 上表現最好的檢查點初始化，而非通常的初始化；預訓練模型在所有任務中皆保持凍結。我們報告 5 個隨機種子的中位數；每次執行的結果取自最佳時代。為了與 Houlsby et al. (2019) 與 Pfeiffer et al. (2021) 的設定進行公平比較，我們將模型序列長度限制為 128 並對所有任務使用固定批次大小。重要的是，在調整到 MRPC、RTE 與 STS-B 時，我們是從預訓練的 RoBERTa large 模型開始，而非已經調整過 MNLI 的模型。使用此限制設定的執行以 †標示。表 9 列出我們執行中所用的超參數。

D.2 DEBERTA

我們再次使用 AdamW 訓練，並採用線性學習率衰減排程。依照 He et al. (2021) 的做法，我們調整學習率、dropout 機率、warm-up 步數與批次大小。我們使用與 (He et al., 2021) 相同的模型序列長度以維持比較的公平性。依照 He et al. (2021) 的做法，在將 LoRA 模組調適到 MRPC、RTE 與 STS-B 時，我們以在 MNLI 上表現最佳的檢查點初始化 LoRA 模組，而非通常的初始化；預訓練模型在所有任務中皆保持凍結。我們報告 5 個隨機種子結果的中位數；每次執行的結果取自最佳 epoch。用於我們實驗的超參數列於表 10。

Method	Dataset	MNLI	SST-2	MRPC	CoLA	QNLI	QQP	RTE	STS-B
	Optimizer	AdamW							
	Warmup Ratio	0.06							
	LR Schedule	Linear							
RoBERTa base LoRA	Batch Size	16	16	16	32	32	16	32	16
	# Epochs	30	60	30	80	25	25	80	40
	Learning Rate	5E-04	5E-04	4E-04	4E-04	4E-04	5E-04	5E-04	4E-04
	LoRA Config.	$r_q = r_v = 8$							
	LoRA α	8							
	Max Seq. Len.	512							
RoBERTa large LoRA	Batch Size	4	4	4	4	4	4	8	8
	# Epochs	10	10	20	20	10	20	20	30
	Learning Rate	3E-04	4E-04	3E-04	2E-04	2E-04	3E-04	4E-04	2E-04
	LoRA Config.	$r_q = r_v = 8$							
	LoRA α	16							
	Max Seq. Len.	128	128	512	128	512	512	512	512
RoBERTa large LoRA†	Batch Size	4							
	# Epochs	10	10	20	20	10	20	20	10
	Learning Rate	3E-04	4E-04	3E-04	2E-04	2E-04	3E-04	4E-04	2E-04
	LoRA Config.	$r_q = r_v = 8$							
	LoRA α	16							
	Max Seq. Len.	128							
RoBERTa large Adpt^P (3M)†	Batch Size	32							
	# Epochs	10	20	20	20	10	20	20	20
	Learning Rate	3E-05	3E-05	3E-04	3E-04	3E-04	3E-04	3E-04	3E-04
	Bottleneck r	64							
	Max Seq. Len.	128							
RoBERTa large Adpt^P (0.8M)†	Batch Size	32							
	# Epochs	5	20	20	20	10	20	20	20
	Learning Rate	3E-04	3E-04	3E-04	3E-04	3E-04	3E-04	3E-04	3E-04
	Bottleneck r	16							
	Max Seq. Len.	128							
RoBERTa large Adpt^H (6M)†	Batch Size	32							
	# Epochs	10	5	10	10	5	20	20	10
	Learning Rate	3E-05	3E-04	3E-04	3E-04	3E-04	3E-04	3E-04	3E-04
	Bottleneck r	64							
	Max Seq. Len.	128							
RoBERTa large Adpt^H (0.8M)†	Batch Size	32							
	# Epochs	10	5	10	10	5	20	20	10
	Learning Rate	3E-04	3E-04	3E-04	3E-04	3E-04	3E-04	3E-04	3E-04
	Bottleneck r	8							
	Max Seq. Len.	128							

Table 9: The hyperparameters we used for RoBERTa on the GLUE benchmark.

D.3 GPT-2

We train all of our GPT-2 models using AdamW (Loshchilov & Hutter, 2017) with a linear learning rate schedule for 5 epochs. We use the batch size, learning rate, and beam search beam size described in Li & Liang (2021). Accordingly, we also tune the above hyperparameters for LoRA. We report the mean over 3 random seeds; the result for each run is taken from the best epoch. The hyperparameters used for LoRA in GPT-2 are listed in Table 11. For those used for other baselines, see Li & Liang (2021).

D.4 GPT-3

For all GPT-3 experiments, we train using AdamW (Loshchilov & Hutter, 2017) for 2 epochs with a batch size of 128 samples and a weight decay factor of 0.1. We use a sequence length of 384 for

方法	資料集	MNLI	SST-2	MRPC	CoLA	QNLI	QQP	RTE	STS-B
	最佳化器	AdamW							
	Warmup 比例	0.06							
	學習率 排程	線性							
RoBERTa 基本模型 LoRA	批次大小	16	16	16	32	32	16	32	16
	訓練輪次 #	30	60	30	80	25	25	80	40
	學習率	5E-04	5E-04	4E-04	4E-04	4E-04	5E-04	5E-04	4E-04
	LoRA 設定	$r_q = r_v = 8$							
	LoRA α	8							
	最大序列長度	512							
RoBERTa large LoRA	批次大小	4	4	4	4	4	4	8	8
	訓練輪數	10	10	20	20	10	20	20	30
	學習率	3E-04	4E-04	3E-04	2E-04	2E-04	3E-04	4E-04	2E-04
	LoRA 設定。	$r_q = r_v = 8$							
	LoRA α	16							
	最大序列長度。	128	128	512	128	512	512	512	512
RoBERTa large LoRA†	批次大小	4							
	# Epochs	10	10	20	20 10		20	20	10
	Learning Rate	3E-04	4E-04	3E-04	2E-04 2E-04		3E-04	4E-04	2E-04
	LoRA 設定	$r_q = r_v = 8$							
	LoRA α	16							
	最大序列長度	128							
RoBERTa large Adpt^P (3M)†	批次大小	32							
	訓練回合數	10	20	20	20	10	20	20	20
	學習率	3E-05	3E-05	3E-04	3E-04	3E-04	3E-04	3E-04	3E-04
	Bottleneck r	64							
	最大序列長度	128							
RoBERTa large Adpt^P (0.8M)†	批次大小	32							
	# 訓練輪數	5	20	20	20	10	20	20	20
	學習率	3E-04	3E-04	3E-04	3E-04	3E-04	3E-04	3E-04	3E-04
	瓶頸 r	16							
	最大序列長度	128							
RoBERTa large Adpt^H (6M)†	批次大小	32							
	訓練輪次數	10	5	10	10	5	20	20	10
	學習率	3E-05	3E-04	3E-04	3E-04	3E-04	3E-04	3E-04	3E-04
	瓶頸 r	64							
	最大序列長度	128							
RoBERTa large Adpt^H (0.8M)†	批次大小	32							
	訓練世代數	10	5	10	10	5	20	20	10
	學習率	3E-04	3E-04	3E-04	3E-04	3E-04	3E-04	3E-04	3E-04
	瓶頸 r	8							
	最大序列長度	128							

表 9：我們在 GLUE 基準上使用的 RoBERTa 超參數。

D.3 GPT-2

我們使用 AdamW（Loshchilov & Hutter, 2017）並以線性學習率排程訓練所有 GPT-2 模型，訓練 5 個 epoch。批次大小、學習率與 beam search 的 beam 大小採用 Li & Liang (2021) 中所述。因此，我們也為 LoRA 調整上述超參數。我們報告 3 個隨機種子結果的平均值；每次執行的結果取自最佳 epoch。GPT-2 上 LoRA 使用的超參數列於表 11。其他基準所用的參數，請參閱 Li & Liang (2021)。

D.4 GPT-3

對於所有 GPT-3 的實驗，我們使用 AdamW（Loshchilov & Hutter, 2017）訓練 2 個 epoch，批次大小為 128 範例，weight decay 因子為 0.1。我們對序列長度採用 384 用於

Method	Dataset	MNLI	SST-2	MRPC	CoLA	QNLI	QQP	RTE	STS-B
	Optimizer	AdamW							
	Warmup Ratio	0.1							
	LR Schedule	Linear							
DeBERTa XXL LoRA	Batch Size	8	8	32	4	6	8	4	4
	# Epochs	5	16	30	10	8	11	11	10
	Learning Rate	1E-04	6E-05	2E-04	1E-04	1E-04	1E-04	2E-04	2E-04
	Weight Decay	0	0.01	0.01	0	0.01	0.01	0.01	0.1
	CLS Dropout	0.15	0	0	0.1	0.1	0.2	0.2	0.2
	LoRA Config.	$r_q = r_v = 8$							
	LoRA α	8							
	Max Seq. Len.	256	128	128	64	512	320	320	128

Table 10: The hyperparameters for DeBERTa XXL on tasks included in the GLUE benchmark.

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

WikiSQL (Zhong et al., 2017), 768 for MNLI (Williams et al., 2018), and 2048 for SAMSum (Gliwa et al., 2019). We tune learning rate for all method-dataset combinations. See Section D.4 for more details on the hyperparameters used. 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 $l_i = 8$ for prefix-layer tuning with $20.2M$ trainable parameters to obtain the overall best performance. We present two parameter budgets for LoRA: 4.7M ($r_q = r_v = 1$ or $r_v = 2$) and 37.7M ($r_q = r_v = 8$ or $r_q = r_k = r_v = r_o = 2$). We report the best validation performance from each run. The training hyperparameters used in our GPT-3 experiments are listed in Table 12.

E 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 Section 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 natu-

方法	資料集	MNLI	SST-2	MRPC	CoLA	QNLI	QQP	RTE	STS-B
	最佳化器	AdamW							
	Warmup 比例	0.1							
	學習率排程	線性							
DeBERTa XXL LoRA	批次大小	8	8	32	4	6	8	4	4
	# 週期	5	16	30	10	8	11	11	10
	學習速率	1E-04	6E-05	2E-04	1E-04	1E-04	1E-04	2E-04	2E-04
	權重衰減	0	0.01	0.01	0	0.01	0.01	0.01	0.1
	CLS 隨機失活	0.15	0	0	0.1	0.1	0.2	0.2	0.2
	LoRA 設定	$r_q = r_v = 8$							
	LoRA α	8							
	最大序列長度	256	128	128	64	512	320	320	128

表 10：DeBERTa XXL 在 GLUE 基準測試所包含任務的超參數。

資料集	E2E	WebNLG	DART
	訓練		
最佳化器	AdamW		
權重衰減	0.01	0.01	0.0
丟棄率	0.1	0.1	0.0
批次大小	8		
# 時代	5		
預熱步驟	500		
學習率排程	線性		
標籤平滑	0.1	0.1	0.0
學習率	0.0002		
調整	$r_q = r_v = 4$		
LoRA α	32		
	推理		
束搜尋大小	10		
長度懲罰	0.9	0.8	0.8
不重複 n-gram 大小	4		

表 11：GPT-2 LoRA 在 E2E、WebNLG 與 DART 上的超參數。

WikiSQL（Zhong et al., 2017）、768 用於 MNLI（Williams et al., 2018），以及 2048 用於 SAMSum（Gliwa et al., 2019）。我們為所有方法與資料集組合調整學習率。關於所使用超參數的更多細節請見附錄 D.4。對於 prefix-embedding 調整，我們發現最佳的 l_p 和 l_i 分別為 256 與 8，合計 $3.2M$ 個可訓練參數。我們使用 $l_p = 8$ 和 $l_i = 8$ 進行 prefix-layer 調整，擁有 $20.2M$ 個可訓練參數，以取得整體最佳表現。我們為 LoRA 提供兩種參數預算：4.7M（$r_q = r_v = 1$ 或 $r_v = 2$）以及 37.7M（$r_q = r_v = 8$ 或 $r_q = r_k = r_v = r_o = 2$)）。我們報告每次執行的最佳驗證表現。我們在 GPT-3 實驗中使用的訓練超參數列於表 12。

E 結合 LoRA 與前綴微調

LoRA 可以自然地與現有的基於前綴的方法結合。在本節中，我們評估了在 WikiSQL 與 MNLI 上，LoRA 與不同前綴微調變體的兩種結合方式。

LoRA+PrefixEmbed (LoRA+PE) 將 LoRA 與前綴嵌入微調結合，在此方法中我們插入$l_p + l_i$ 被視為可訓練參數的特殊 token 嵌入。有關前綴嵌入微調的更多內容，請參見第 5.1 節。

LoRA+PrefixLayer (LoRA+PL) 結合了 LoRA 與 prefix-layer 微調。我們也插入了 $l_p + l_i$特殊標記；然而，我們並不是讓這些標記的隱含表示自然地演化——

Hyperparameters	Fine-Tune	PreEmbed	PreLayer	BitFit	Adapter^{H}	LoRA
Optimizer	AdamW					
Batch Size	128					
# Epoch	2					
Warmup Tokens	250,000					
LR Schedule	Linear					
Learning Rate	5.00E-06	5.00E-04	1.00E-04	1.6E-03	1.00E-04	2.00E-04

Table 12: The training hyperparameters used for different GPT-3 adaption methods. We use the same hyperparameters for all datasets after tuning learning rate.

rally, 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 Section 5.1.

In Table 15, 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.

F ADDITIONAL EMPIRICAL EXPERIMENTS

F.1 ADDITIONAL EXPERIMENTS ON GPT-2

We also repeat our experiment on DART (Nan et al., 2020) and WebNLG (Gardent et al., 2017) following the setup of Li & Liang (2021). The result is shown in Table 13. Similar to our result on E2E NLG Challenge, reported in Section 5, LoRA performs better than or at least on-par with prefix-based approaches given the same number of trainable parameters.

Method	# Trainable	DART		
	Parameters	BLEU↑	MET↑	TER↓
	GPT-2 Medium			
Fine-Tune	354M	46.2	**0.39**	**0.46**
Adapter^{L}	0.37M	42.4	0.36	0.48
Adapter^{L}	11M	45.2	0.38	**0.46**
FT^{Top2}	24M	41.0	0.34	0.56
PrefLayer	0.35M	46.4	0.38	**0.46**
LoRA	0.35M	**47.1**$_{\pm.2}$	**0.39**	**0.46**
	GPT-2 Large			
Fine-Tune	774M	47.0	**0.39**	0.46
Adapter^{L}	0.88M	45.7$_{\pm.1}$	0.38	0.46
Adapter^{L}	23M	47.1$_{\pm.1}$	**0.39**	**0.45**
PrefLayer	0.77M	46.7	0.38	**0.45**
LoRA	0.77M	**47.5**$_{\pm.1}$	**0.39**	**0.45**

Table 13: GPT-2 with different adaptation methods on DART. The variances of MET and TER are less than 0.01 for all adaption approaches.

超參數	微調	預嵌入	預層	BitFit	Adapter^{H}	LoRA
最佳化器	AdamW					
批次大小	128					
# 紀元	2					
預熱代幣	250,000					
學習率排程	線性					
學習率	5.00E-06	5.00E-04	1.00E-04	1.6E-03	1.00E-04	2.00E-04

表 12：用於不同 GPT-3 調整方法的訓練超參數。在調整學習率後，我們對所有資料集使用相同的超參數。

相反地，我們在每個 Transformer 區塊後都以一個與輸入無關的向量取代它們。因此，嵌入層與後續 Transformer 區塊的激活都被視為可訓練的參數。關於 prefix-layer 微調的更多內容，請參見第 5.1 節。

在表 15 中，我們展示了 LoRA+PE 與 LoRA+PL 在 WikiSQL 和 MultiNLI 上的評估結果。首先，LoRA+PE 在 WikiSQL 上明顯優於 LoRA 與 prefix-embedding 調整，這表示 LoRA 與 prefix-embedding 調整在某種程度上是互補的。在 MultiNLI 上，LoRA+PE 的組合表現未必優於 LoRA，可能是因為單獨使用 LoRA 已能達到與人類基準相當的表現。其次，我們注意到即使有較多可訓練參數，LoRA+PL 的表現仍略遜於 LoRA。我們將此歸因於 prefix-layer 調整對學習率的選擇非常敏感，從而使 LoRA 權重在 LoRA+PL 中的優化變得更困難。

F 額外的實驗結果

F.1 在 GPT-2 上的額外實驗

我們也依照 Li & Liang (2021) 的設定，在 DART (Nan et al., 2020) 和 WebNLG (Gardent et al., 2017) 上重複實驗。結果如表 13 所示。與第 5 節報告的 E2E NLG Challenge 結果類似，當可訓練參數數量相同時，LoRA 的表現優於或至少與基於 prefix 的方法相當。

方法	# 可訓練	DART		
	參數	BLEU↑	MET↑	TER↓
	GPT-2 中等版			
微調	354M	46.2	**0.39**	**0.46**
Adapter^{L}	0.37M	42.4	0.36	0.48
Adapter^{L}	11M	45.2	0.38	**0.46**
FT^{Top2}	24M	41.0	0.34	0.56
偏好層	0.35M	46.4	0.38	**0.46**
LoRA	0.35M	**47.1**$_{\pm.2}$	**0.39**	**0.46**
	GPT-2 Large			
微調	774M	47.0	**0.39**	0.46
Adapter^{L}	0.88M	45.7$_{\pm.1}$	0.38	0.46
Adapter^{L}	23M	47.1$_{\pm.1}$	**0.39**	**0.45**
PrefLayer	0.77M	46.7	0.38	**0.45**
LoRA	0.77M	**47.5**$_{\pm.1}$	**0.39**	**0.45**

表 13：在 DART 上使用不同調適方法的 GPT-2。所有調適方法的 MET 和 TER 變異小於0.01。

Method	WebNLG								
	BLEU↑			MET↑			TER↓		
	U	S	A	U	S	A	U	S	A
	GPT-2 Medium								
Fine-Tune (354M)	27.7	**64.2**	46.5	.30	**.45**	.38	.76	**.33**	.53
Adapter$^{\text{L}}$ (0.37M)	45.1	54.5	50.2	.36	.39	.38	.46	.40	.43
Adapter$^{\text{L}}$ (11M)	**48.3**	60.4	54.9	**.38**	.43	**.41**	**.45**	.35	**.39**
FT$^{\text{Top2}}$ (24M)	18.9	53.6	36.0	.23	.38	.31	.99	.49	.72
Prefix (0.35M)	45.6	62.9	55.1	**.38**	.44	**.41**	.49	.35	.40
LoRA (0.35M)	46.7$_{\pm.4}$	62.1$_{\pm.2}$	**55.3**$_{\pm.2}$	**.38**	.44	**.41**	.46	**.33**	**.39**
	GPT-2 Large								
Fine-Tune (774M)	43.1	65.3	55.5	.38	**.46**	.42	.53	.33	.42
Adapter$^{\text{L}}$ (0.88M)	**49.8**$_{\pm.0}$	61.1$_{\pm.0}$	56.0$_{\pm.0}$	.38	.43	.41	**.44**	.35	.39
Adapter$^{\text{L}}$ (23M)	49.2$_{\pm.1}$	64.7$_{\pm.2}$	**57.7**$_{\pm.1}$	**.39**	**.46**	**.43**	.46	.33	.39
Prefix (0.77M)	47.7	63.4	56.3	**.39**	.45	.42	.48	.34	.40
LoRA (0.77M)	48.4$_{\pm.3}$	64.0$_{\pm.3}$	57.0$_{\pm.1}$	**.39**	.45	.42	.45	**.32**	**.38**

Table 14: GPT-2 with different adaptation methods on WebNLG. The variances of MET and TER are less than 0.01 for all the experiments we ran. "U" indicates unseen categories, "S" indicates seen categories, and "A" indicates all categories in the test set of WebNLG.

F.2 Additional Experiments on GPT-3

We present additional runs on GPT-3 with different adaptation methods in Table 15. The focus is on identifying the trade-off between performance and the number of trainable parameters.

F.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 16, 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 (± 0.3) variance due to random seeds.

The training hyperparameters of different adaptation approaches on MNLI-n are reported in Table 17. 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.

G 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 Ham & Lee (2008).

方法	WebNLG								
	BLEU↑			MET↑			TER↓		
	U	S	A	U	S	A	U	S	A
	GPT-2 中型								
微調 (354M)	27.7	**64.2**	46.5	.30	**.45**	.38	.76	**.33**	.53
Adapter$^{\text{L}}$ (0.37M)	45.1	54.5	50.2	.36	.39	.38	.46	.40	.43
Adapter$^{\text{L}}$ (11M)	**48.3**	60.4	54.9	**.38**	.43	**.41**	**.45**	.35	**.39**
FT$^{\text{Top2}}$ (24M)	18.9	53.6	36.0	.23	.38	.31	.99	.49	.72
Prefix (0.35M)	45.6	62.9	55.1	**.38**	.44	**.41**	.49	.35	.40
LoRA (0.35M)	46.7$_{\pm.4}$	62.1$_{\pm.2}$	**55.3**$_{\pm.2}$	**.38**	.44	**.41**	.46	**.33**	**.39**
	GPT-2 Large								
微調（774M）	43.1	65.3	55.5	.38	**.46**	.42	.53	.33	.42
Adapter$^{\text{L}}$ (0.88M)	**49.8**$_{\pm.0}$	61.1$_{\pm.0}$	56.0$_{\pm.0}$	.38	.43	.41	**.44**	.35	.39
Adapter$^{\text{L}}$ (23M)	49.2$_{\pm.1}$	64.7$_{\pm.2}$	**57.7**$_{\pm.1}$	**.39**	**.46**	**.43**	.46	.33	.39
Prefix (0.77M)	47.7	63.4	56.3	**.39**	.45	.42	.48	.34	.40
LoRA（0.77M）	48.4$_{\pm.3}$	64.0$_{\pm.3}$	57.0$_{\pm.1}$	**.39**	.45	.42	.45	**.32**	**.38**

表14：在 WebNLG 上使用不同微調方法的 GPT-2。MET 和 TER 的變異在我們進行的所有實驗中均小於 0.01。"U"表示未見類別，"S"表示見過的類別，"A"表示 WebNLG 測試集中所有類別。

F.2 在 GPT-3 上的額外實驗

我們在表 15 中對 GPT-3 進行了額外實驗，採用不同的適應方法。重點在於找出效能與可訓練參數數量之間的權衡。

F.3 低資料情境

為了評估不同適應方法在低資料情境下的表現，我們從完整的 MNLI 訓練集隨機抽取 100、1k 和 10k 筆訓練範例，構成低資料 MNLI-n 任務。在表 16 中，我們展示了不同適應方法在 MNLI-n 上的表現。令我們驚訝的是，PrefixEmbed 和 PrefixLayer 在 MNLI-100 資料集上表現非常差，PrefixEmbed 的表現甚至只比隨機機率稍好（37.6% 對 33.3%）。PrefixLayer 儘管優於 PrefixEmbed，但在 MNLI-100 上仍明顯不如 Fine-Tune 或 LoRA。隨著訓練範例數量增加，基於 prefix 的方法與 LoRA ／ Fine-tuning 之間的差距逐漸縮小，這可能表示在 GPT-3 上 prefix 類方法不適合低資料任務。LoRA 在 MNLI-100 和 MNLI-Full 上的表現優於微調，並且在 MNLI-1k 與 MNLI-10K 上考慮到由隨機種子造成的 (±0.3) 變異後，表現相當。

不同適應方法在 MNLI-n 上的訓練超參數報告於表 17。我們在 MNLI-100 集上對 PrefixLayer 使用較小的學習率，因為在較大學習率下訓練損失無法下降。

G 測量子空間之間的相似性

在本文中，我們使用度量 $\phi(A, B, i, j) = \psi(U_A^i, U_B^j) = \frac{\|U_A^{i\top} U_B\|_F^2}{\min\{i,j\}}$ 來衡量兩個列正交矩陣 $U_A^i \in \mathbb{R}^{d\times i}$ 和 $U_B^j \in \mathbb{R}^{d\times j}$ 之間的子空間相似性，這兩個矩陣分別由 A 和 B 的左奇異矩陣取列所得到。我們指出，該相似性實際上是測量子空間距離的標準投影度量（Projection Metric）的相反量 Ham & Lee (2008)。

Method	Hyperparameters	# Trainable Parameters	WikiSQL	MNLI-m
Fine-Tune	-	175B	73.8	89.5
PrefixEmbed	$l_p = 32, l_i = 8$	0.4 M	55.9	84.9
	$l_p = 64, l_i = 8$	0.9 M	58.7	88.1
	$l_p = 128, l_i = 8$	1.7 M	60.6	88.0
	$l_p = 256, l_i = 8$	3.2 M	63.1	88.6
	$l_p = 512, l_i = 8$	6.4 M	55.9	85.8
PrefixLayer	$l_p = 2, l_i = 2$	5.1 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
Adapter^H	$r = 1$	7.1 M	71.9	89.8
	$r = 4$	21.2 M	73.2	91.0
	$r = 8$	40.1 M	73.2	91.5
	$r = 16$	77.9 M	73.2	91.5
	$r = 64$	304.4 M	72.6	91.5
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
	$r_q = r_k = r_v = r_o = 1$	9.4 M	74.1	91.2
	$r_q = r_v = 4$	18.8 M	73.7	91.3
	$r_q = r_k = r_v = r_o = 2$	18.8 M	73.7	**91.7**
	$r_q = r_v = 8$	37.7 M	73.8	**91.6**
	$r_q = r_k = r_v = r_o = 4$	37.7 M	74.0	**91.7**
	$r_q = r_v = 64$	301.9 M	73.6	91.4
	$r_q = r_k = r_v = r_o = 64$	603.8 M	73.9	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 15: 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.

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 16: 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.

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 Ham & Lee (2008) is defined as:

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

方法	超參數	# 可訓練參數數量	WikiSQL	MNLI-m
微調	-	175B	73.8	89.5
PrefixEmbed	$l_p = 32, l_i = 8$	0.4 M	55.9	84.9
	$l_p = 64, l_i = 8$	0.9 M	58.7	88.1
	$l_p = 128, l_i = 8$	1.7 M	60.6	88.0
	$l_p = 256, l_i = 8$	3.2 M	63.1	88.6
	$l_p = 512, l_i = 8$	6.4 M	55.9	85.8
PrefixLayer	$l_p = 2, l_i = 2$	5.1 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
Adapter^H	$r = 1$	7.1 M	71.9	89.8
	$r = 4$	21.2 百萬	73.2	91.0
	$r = 8$	40.1 百萬	73.2	91.5
	$r = 16$	77.9 百萬	73.2	91.5
	$r = 64$	304.4 百萬	72.6	91.5
LoRA	$r_v = 2$	4.7 百萬	73.4	**91.7**
	$r_q = r_v = 1$	4.7 百萬	73.4	91.3
	$r_q = r_v = 2$	9.4 百萬	73.3	91.4
	$r_q = r_k = r_v = r_o = 1$	9.4 百萬	74.1	91.2
	$r_q = r_v = 4$	18.8 百萬	73.7	91.3
	$r_q = r_k = r_v = r_o = 2$	18.8 百萬	73.7	**91.7**
	$r_q = r_v = 8$	37.7 百萬	73.8	**91.6**
	$r_q = r_k = r_v = r_o = 4$	37.7 百萬	74.0	**91.7**
	$r_q = r_v = 64$	301.9 M	73.6	91.4
	$r_q = r_k = r_v = r_o = 64$	603.8 M	73.9	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

表 15：在 WikiSQL 與 MNLI 上不同微調方法的超參數分析。無論是 prefix-embedding 調整（PrefixEmbed）或 prefix-layer 調整（PrefixLayer），隨著可訓練參數數量增加其表現都會變差，而 LoRA 的表現則趨於穩定。效能以驗證準確率衡量。

方法	MNLI(m)-100	MNLI(m)-1k	MNLI(m)-10k	MNLI(m)-392K
GPT-3（微調）	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**

表 16：在使用 GPT-3 175B 的 MNLI 子集上，不同方法的驗證準確率。MNLI-n 描述了一個包含 n 個訓練範例的子集。我們使用完整的驗證集進行評估。與其他方法（包括微調）相比，LoRA 在樣本效率方面表現良好。

具體而言，令 $U_A^{i\top} U_B^j$ 的奇異值為 $\sigma_1, \sigma_2, \cdots, \sigma_p$，其中 $p = \min\{i, j\}$。我們知道投影度量 Ham & Lee (2008) 定義為：

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

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

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, $\phi(A, B, i, j) \in (0, 1)$.

H Additional Experiments on Low-Rank Matrices

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

H.1 Correlation between LoRA Modules

See Figure 6 and Figure 7 for how the results presented in Figure 3 and Figure 4 generalize to other layers.

H.2 Effect of r on GPT-2

We repeat our experiment on the effect of r (Section 7.2) 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 18. 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.

H.3 Correlation between W and ΔW

See Figure 8 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?

超參數	調整	MNLI-100	MNLI-1k	MNLI-10K	MNLI-392K
最佳化器	-	AdamW			
預熱 Token 數	-	250,000			
學習率 排程	-	線性			
批次大小	-	20	20	100	128
# 紀元	-	40	40	4	2
學習率	微調	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	$r_q = r_v = 8$			

表 17：在 MNLI(m)-n 上，不同 GPT-3 調適方法所使用的超參數。

我們的相似性定義如下：

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

此相似性滿足：若 U_A^i 和 U_B^j 共享相同的欄位範圍，則 $\phi(A, B, i, j) = 1$。若它們完全正交，則 $\phi(A, B, i, j) = 0$。否則，$\phi(A, B, i, j) \in (0, 1)$。

H 額外實驗：低秩矩陣

我們提出關於低秩更新矩陣調查的額外結果。

H.1 LoRA 模組之間的相關性

請參見圖6與圖7，了解圖3與圖4所示結果如何推廣到其他層。

H.2 r 對 GPT-2 的影響

我們在 GPT-2 上重複了有關 r 影響的實驗（見第7.2節）。以 E2E NLG Challenge 資料集為例，我們報告在訓練26,000步後，不同 r 選擇所達到的驗證損失與測試指標。結果列於表18。對於 GPT-2 Medium，最佳秩依據所使用的指標介於4到16之間，這與 GPT-3 175B 的情況相似。值得注意的是，模型大小與調適最佳秩之間的關係仍是未解之謎。

H.3 W 與 ΔW 之間的相關性

參見圖 8，顯示在不同 r 下 W 與 ΔW 之間的正規化子空間相似度。

再次注意，ΔW 並不包含 W 的主要奇異方向，因為 ΔW 中前 4 個方向與 W 中前 10% 方向之間的相似度僅略高於 0.2。這證明 ΔW 包含那些在 W 中並未被強調的「任務專屬」方向，後者在 *not* 一詞中被特別標示。

接下來一個有趣的問題是：我們需要把那些任務特定的指示放大得多"強"，才能讓模型調整順利運作？

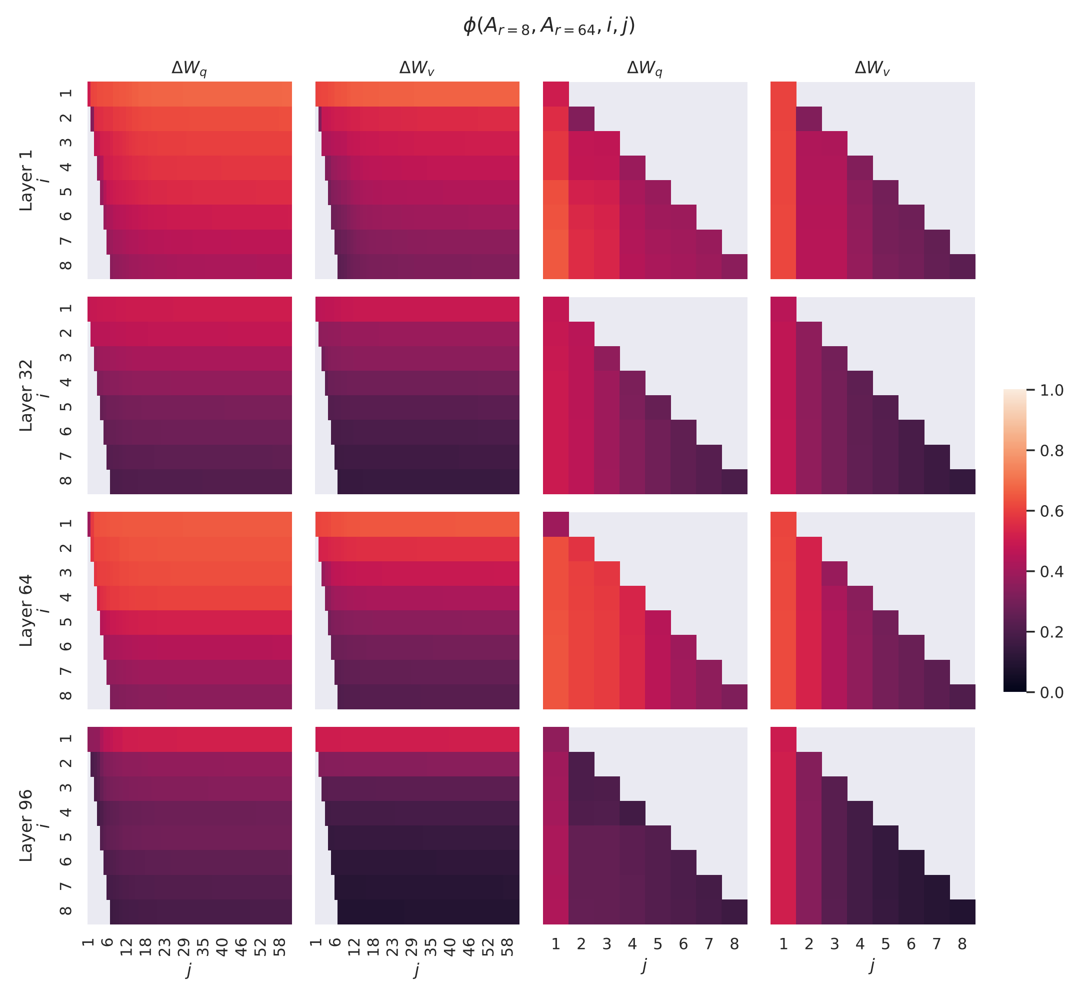


Figure 6: Normalized subspace similarity between the column vectors of $A_{r=8}$ and $A_{r=64}$ for both ΔW_q and ΔW_v from the 1st, 32nd, 64th, and 96th layers in a 96-layer Transformer.

H.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 Section 7.3, 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.

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.

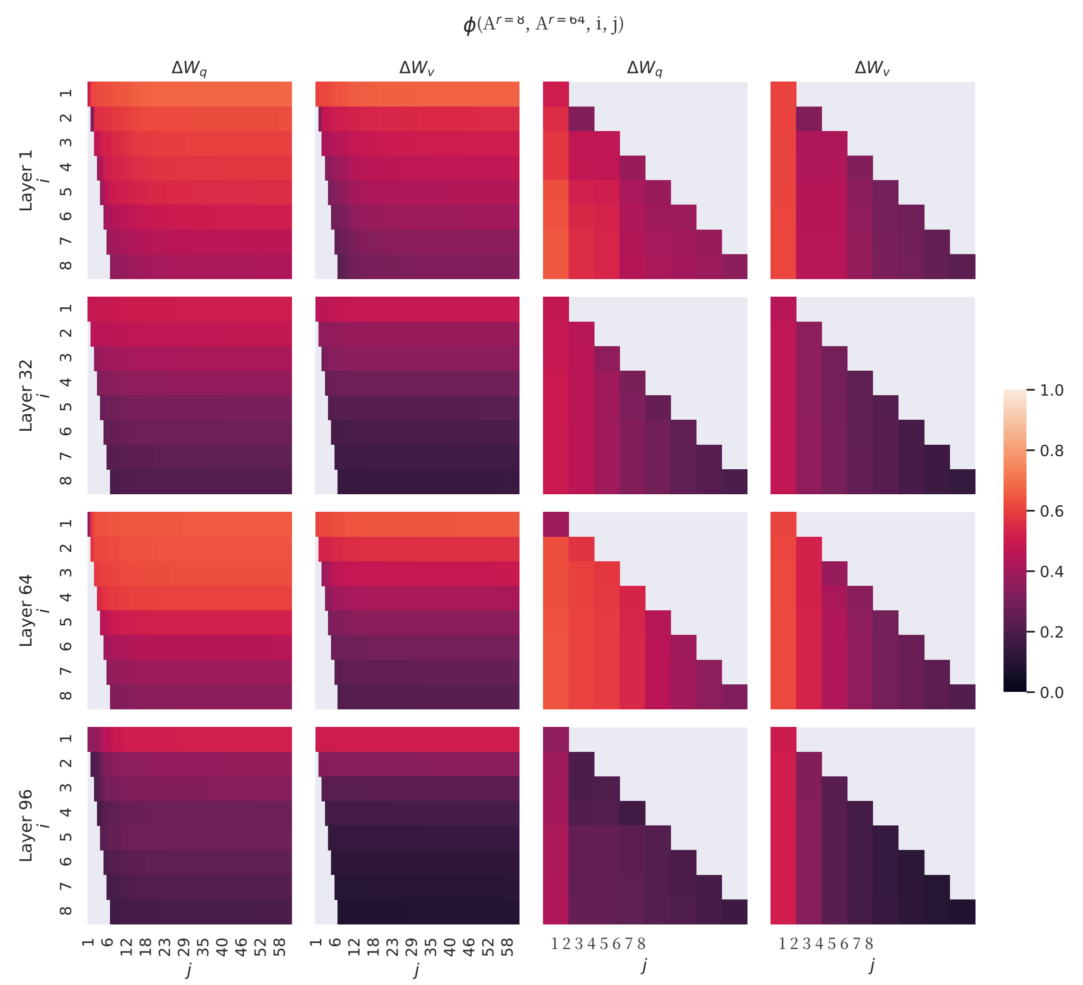


圖 6：在 96 層 Transformer 中，第 1、32、64 和 96 層中 $A_{r=8}$ 與 $A_{r=64}$ 欄向量之間的標準化子空間相似度，分別來自 ΔW_q 與 ΔW_v。

H.4 放大因子

人們自然會把特徵放大因子視為比率 $\frac{\|\Delta W\|_F}{\|U^\top W V^\top\|_F}$，其中 U 和 V ~~是 ΔW~~ 的 SVD 分解中的左奇異矩陣與右奇異矩陣。（回想 $UU^\top W V^\top V$ 表示將 W 投影到由 ΔW 張成的子空間。）

直觀上，當 ΔW 主要包含與任務相關的方向時，這個量度衡量了這些方向被 ΔW 放大的程度。如第 7.3 節所示，對於 $r = 4$，此放大因子可達 20。換句話說（一般而言），在每一層中有大約四個特徵方向（在預訓練模型 W 的整個特徵空間中）需要被非常大的因子 20 放大，才能達到我們報告的下游專門任務的準確度。且可以預期，對每一個不同的下游任務，需要被放大的特徵方向集合會相當不同。

不過可以注意到，對於 $r = 64$ 而言，這個放大因子只有大約 2，表示在 ΔW 中用 $r = 64$ 所學到的大多數方向（*most*）並沒有被大量放大。這其實不令人驚訝，反而再次證明要表示「任務特定方向」（也就是用於模型適應）的內在秩（intrinsic rank）是低的（*needed*）。相比之下，ΔW 的秩為 4 的版本（對應於 $r = 4$）中的那些方向被更大的因子約 20 放大。

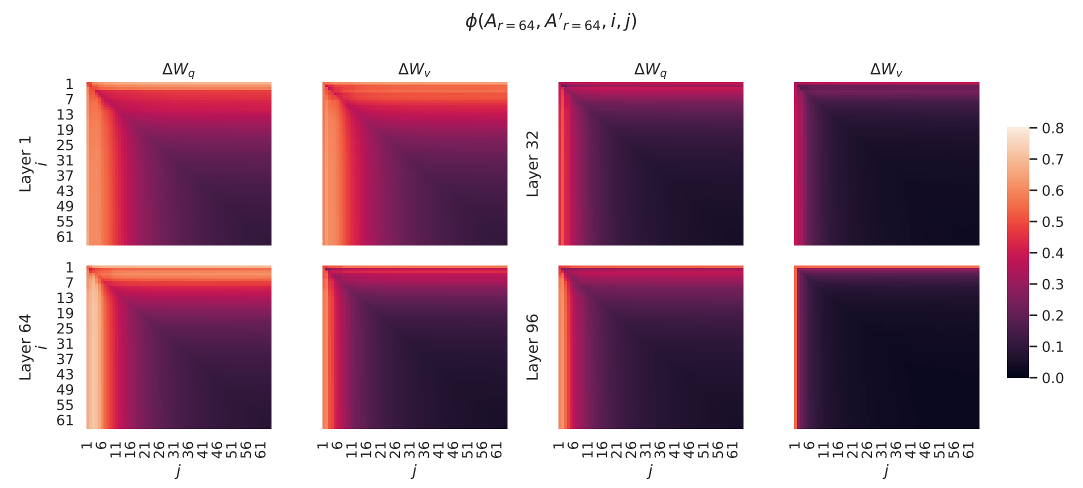


Figure 7: Normalized subspace similarity between the column vectors of $A_{r=64}$ from two randomly seeded runs, for both ΔW_q and ΔW_v from the 1st, 32nd, 64th, and 96th layers in a 96-layer Transformer.

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 18: 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.

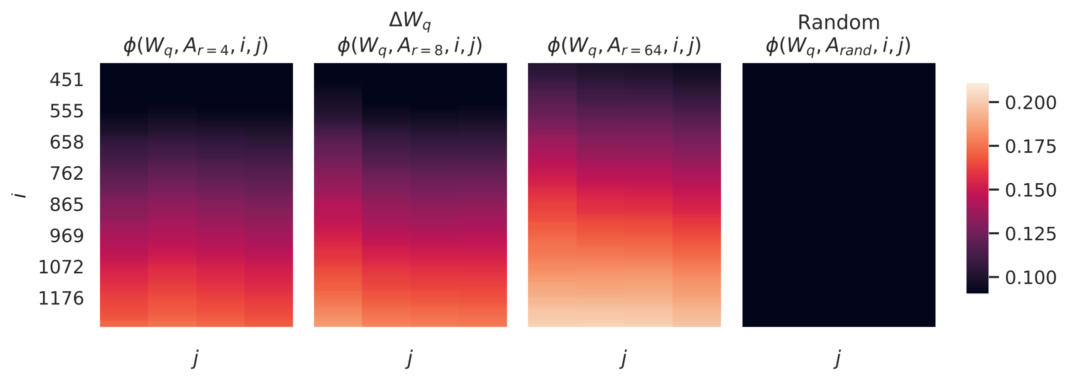


Figure 8: Normalized subspace similarity between the singular directions of W_q and those of ΔW_q with varying r and a random baseline. ΔW_q amplifies directions that are important but not emphasized in W. ΔW with a larger r tends to pick up more directions that are already emphasized in W.

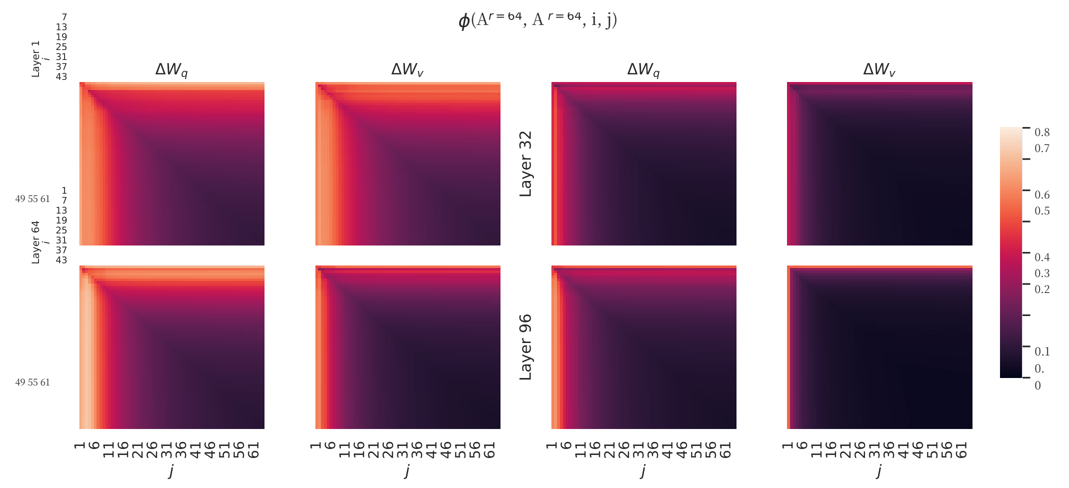


圖 7：來自兩個不同隨機種子執行中 $A_{r=64}$ 欄向量之間的歸一化子空間相似度，針對 96 層 Transformer 中第 1、32、64 和 96 層的 ΔW_q 與 ΔW_v。

Rank r	驗證損失	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

表 18：使用 GPT-2 Medium 並透過不同秩的 LoRA 在 E2E NLG 挑戰中達成的驗證損失與測試集指標。與 GPT-3 在許多任務上 $r = 1$ 即足夠不同，這裡驗證損失在 $r = 16$ 達到最高表現，而 BLEU 則在 $r = 4$ 達到最高，這暗示 GPT-2 Medium 在調適時具有與 GPT-3 175B 類似的內在秩。請注意，我們的一些超參數是在 $r = 4$ 上調整的，該模型的參數量與另一基線相符，因此可能不適用於其他 r 的選擇。

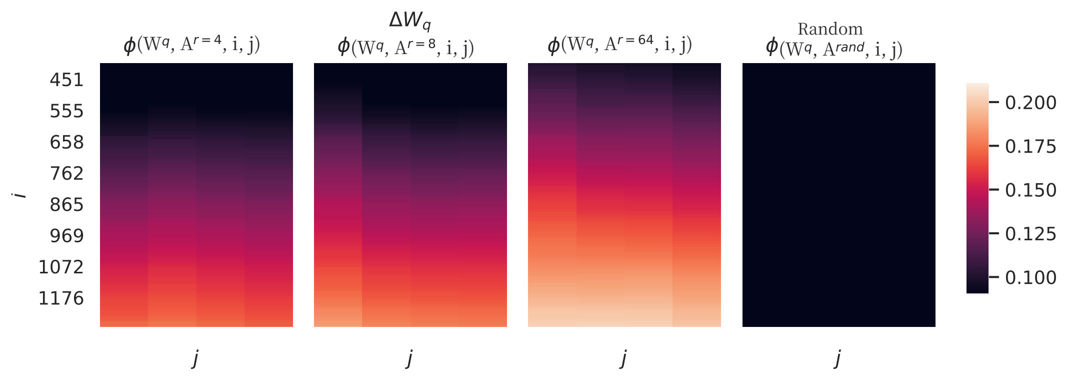


圖 8：在不同 r 下，W_q 的奇異方向與 ΔW_q 的奇異方向之正規化子空間相似度，以及與隨機基線的比較。ΔW_q 放大了在 W 中重要但未被強調的方向。具有較大 r 的 ΔW 傾向於捕捉到在 W 中已經被強調的更多方向。