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 5, 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.

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

B 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$\times$) 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 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.

B.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 6.

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

Table 6: 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.

B.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 7: 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 K.2.

Table 7 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.

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 J for a more formal discussion)

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

where $U^i_{A_{r=8}}$ 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 K.1.

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.

[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.

[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.

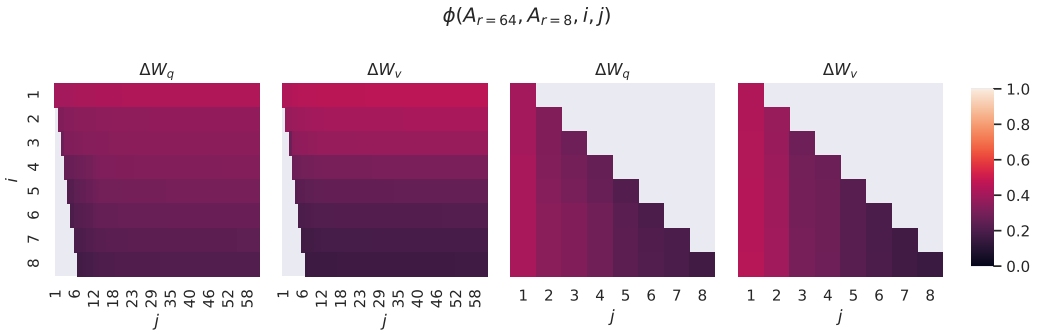

Figure 3: Subspace similarity between column vectors of $A_{r=8}$ and $A_{r=64}$ for both ΔW_q and ΔW_v. The third and the fourth figures zoom in on the lower-left triangle in the first two figures. The top directions in $r = 8$ are included in $r = 64$, and vice versa.

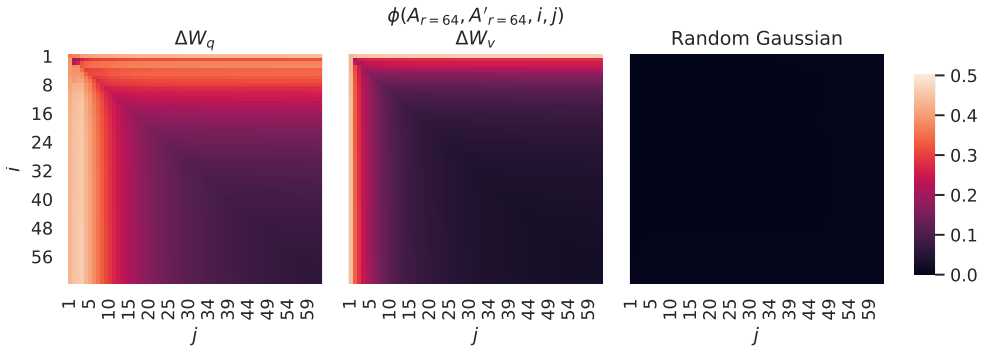

Figure 4: **Left and Middle:** Normalized subspace similarity between the column vectors of $A_{r=64}$ from two random seeds, for both ΔW_q and ΔW_v in the 48-th layer. **Right:** the same heat-map between the column vectors of two random Gaussian matrices. See Section K.1 for other layers.

Subspace similarity between different random seeds. We further confirm this by plotting the normalized subspace similarity between two randomly seeded runs with $r = 64$, shown in 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 7. As a comparison, we also plot two random Gaussian matrices, which do not share any common singular value directions with each other.

B.3 How Does the Adaptation Matrix ΔW Compare to W?

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

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

We draw *several conclusions* from Table 8. 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 K.4 for why $r = 64$ has a smaller amplification factor. We also provide a visualization in Section K.3 for how the correlation changes as we include more top singular directions from W_q.

	$r = 4$			$r = 64$		
	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 8: 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.

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*.

C 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$^{\text{H}}$, and a recent, more efficient variant by Lin et al. (2020), which we call Adapter$^{\text{L}}$. 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.

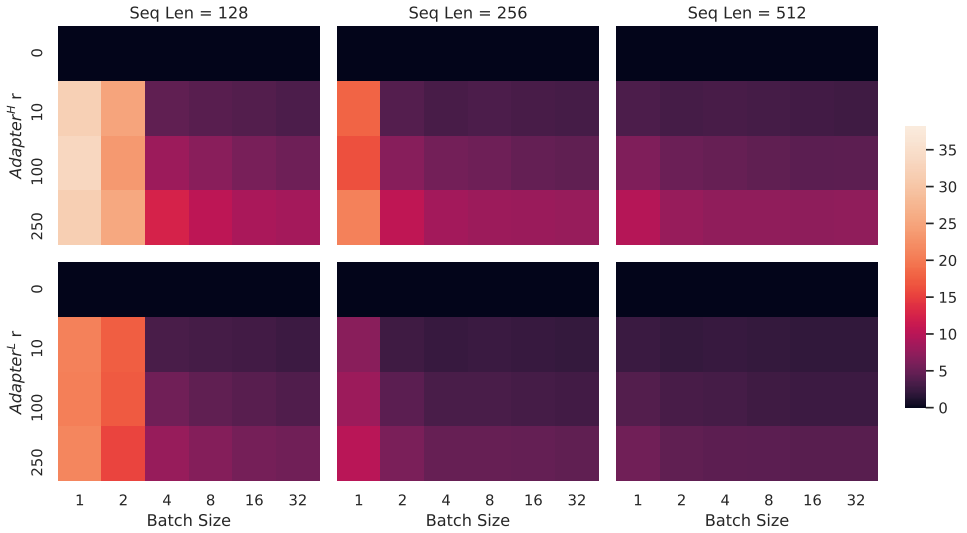

Figure 5: Percentage slow-down of inference latency compared to the no-adapter ($r = 0$) baseline. The top row shows the result for Adapter$^{\text{H}}$ and the bottom row Adapter$^{\text{L}}$. Larger batch size and sequence length help to mitigate the latency, but the slow-down can be as high as over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

D DESCRIPTION OF DATASETS

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), 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 $x = \{\text{table schema}, \text{query}\}$ 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.

E THE CALCULATION OF TRAINABLE PARAMETERS FOR BASELINES

We do not count the classification head for NLU tasks in our calculation of trainable parameters, even though they are trainable – otherwise they would be random – following prior work.

Bias-only or BitFit The number of trainable parameters is $|\Theta| = L_{bias} \times d_{model}$, where L_{bias} is the number of trainable bias vectors.

Prefix-embedding tuning (PreEmbed) 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) The 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 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$^{\text{L}}$).

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

LoRA 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.

F HYPERPARAMETERS USED IN EXPERIMENTS

F.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.

F.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.

F.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).

F.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 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 F.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.

G EXAMPLE LEARNING CURVES FROM GPT-3 175B

The final performance shown in Table 4 does not tell us if different approaches reach their optimal performance at the same speed. We look at GPT-3 175B instead of smaller scale experiments because we do not have access to the learning curves for baselines taken from prior work. Figure 6 shows the learning curves from one of datasets we used. There is not a significant difference

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.

in terms of the speed up of convergence among LoRA, full finetuning (FT), and Adapter, though LoRA achieves the lowest validation loss and does so slightly earlier than the other two methods.

H 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.

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.

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

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.

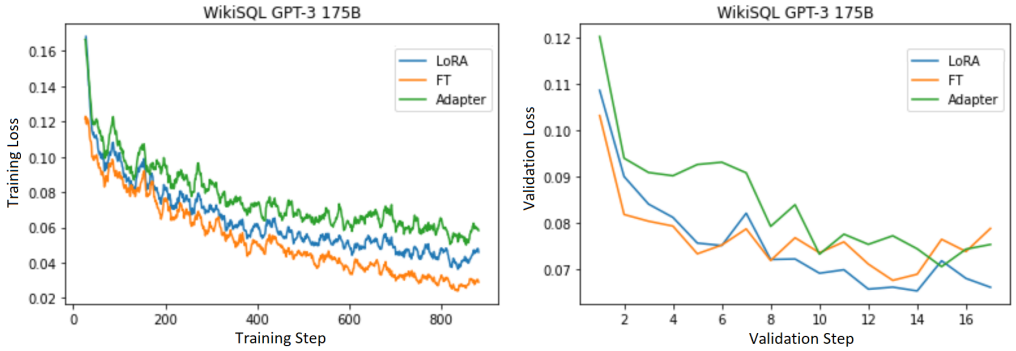

Figure 6: Learning curves for different adaptation methods on WIkiSQL using GPT-3 175B. LoRA and full finetuning (FT) reaches their best validation loss slightly faster than Adapter.

I Additional Empirical Experiments

I.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 Parameters	DART BLEU↑	DART MET↑	DART 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.

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^{L} (0.37M)	45.1	54.5	50.2	.36	.39	.38	.46	.40	.43
Adapter^{L} (11M)	**48.3**	60.4	54.9	**.38**	.43	**.41**	**.45**	.35	**.39**
FT^{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)	$46.7_{\pm.4}$	$62.1_{\pm.2}$	$\mathbf{55.3}_{\pm.2}$	**.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^{L} (0.88M)	$\mathbf{49.8}_{\pm.0}$	$61.1_{\pm.0}$	$56.0_{\pm.0}$	.38	.43	.41	**.44**	.35	.39
Adapter^{L} (23M)	$49.2_{\pm.1}$	$64.7_{\pm.2}$	$\mathbf{57.7}_{\pm.1}$	**.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.

I.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.

I.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.

J 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).

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
$\text{Adapter}^{\text{H}}$	$r = 1$	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:

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

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

K Additional Experiments on Low-Rank Matrices

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

K.1 Correlation between LoRA Modules

See Figure 7 and Figure 8 for how the results presented in Figure 3 and Figure 4 generalize to other layers.

K.2 Effect of r on GPT-2

We repeat our experiment on the effect of r (Section B.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.

K.3 Correlation between W and ΔW

See Figure 9 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?

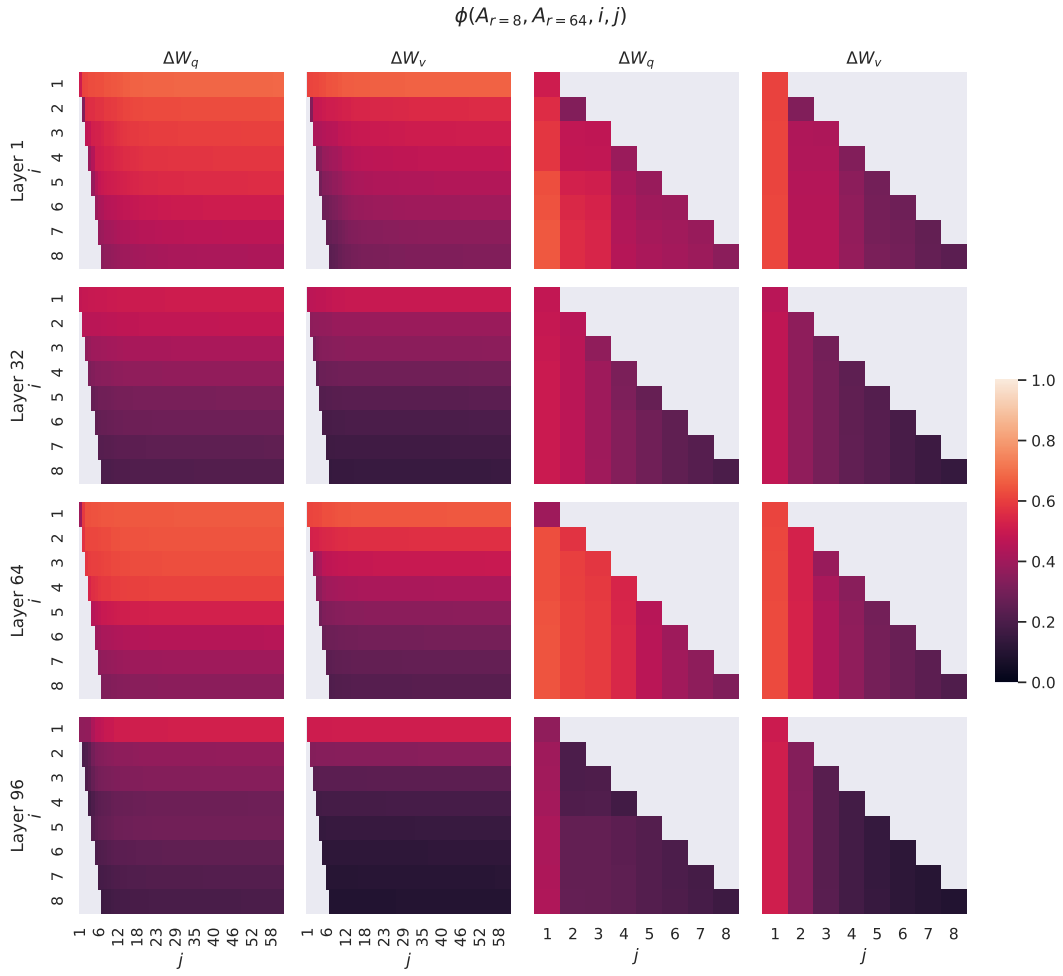

Figure 7: Normalized subspace similarity between the column vectors of $A_{r=8}$ and $A_{r=64}$ for both ΔW_q and ΔW_v from the 1st, 32nd, 64th, and 96th layers in a 96-layer Transformer.

K.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 $UU^\top WV^\top V$ 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 B.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.

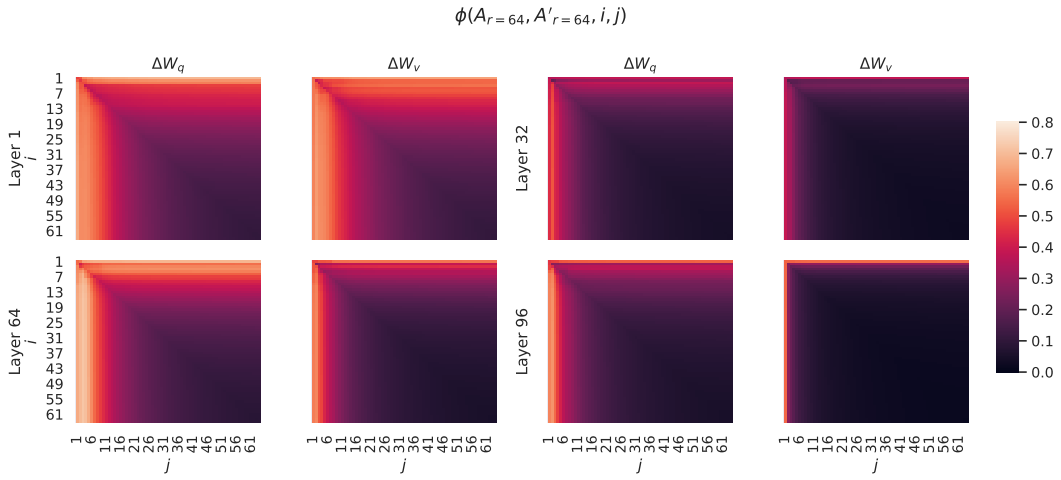

Figure 8: Normalized subspace similarity between the column vectors of $A_{r=64}$ from two randomly seeded runs, for both ΔW_q and ΔW_v from the 1st, 32nd, 64th, and 96th layers in a 96-layer Transformer.

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.

L Gradient Calculation with LoRA

When using LoRA, We do not need to calculate the the gradient of the pre-trained, frozen weights. Consider $h_i = W_i x_{i-1}$ and its LoRA counterpart $h_i = W_i x_{i-1} + BAx_{i-1}$. Given $\frac{\partial L}{\partial h_i}$, we would like to know $\frac{\partial L}{\partial B}$ and $\frac{\partial L}{\partial A}$. We denote $v = Ax_{i-1}$. $\frac{\partial L}{\partial B}$ is given by $\frac{\partial L}{\partial h_i} \otimes v$, and $\frac{\partial L}{\partial A}$ is given by $\frac{\partial L}{\partial v} \otimes x_{i-1}$. Note that we do not need to compute $\frac{\partial L}{\partial W_i}$ at any point, even though we compute $\frac{\partial L}{\partial x_{i-1}}$, which involves a matrix multiplication with W_i.

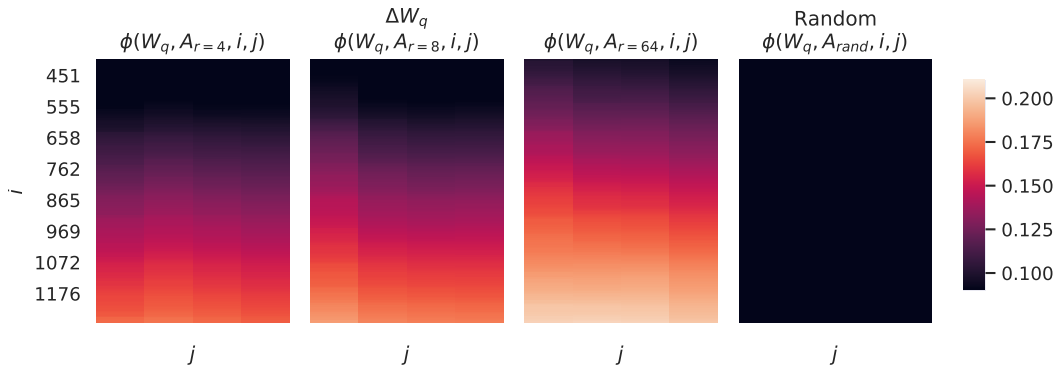

Figure 9: Normalized subspace similarity between the singular directions of W_q and those of ΔW_q with varying r and a random baseline. ΔW_q amplifies directions that are important but not emphasized in W. ΔW with a larger r tends to pick up more directions that are already emphasized in W.