

Low-rank fine-tuning of the GPT-2 model

Stanford CS224N Default Project

Rongge YAN

Department of Computer Science
Stanford University
yrg@stanford.edu

Abstract

We study the efficiency and effectiveness of low-rank adaptation (LoRA) applied to the GPT-2 model for a paraphrase detection task. Through careful analysis and hyperparameter tuning, we achieve 98.66% of full fine-tuning performance with rank-four adaptation of all weight matrices, totaling mere 0.53% of the number of parameters. We offer some intuition on the effectiveness of LoRA by relating it to the empirical rank of full fine-tuning weight updates. Our findings confirm some of the previous results and challenge others.

1 Key Information to include

- Mentor: Shicheng Liu
- External Collaborators: None
- Sharing project: None
- Late days remaining before the report: 2
- Late days applied for the report: 3
- Late days remaining after the report: -1

2 Introduction

Modern large language models contain a huge number of parameters. While some pretrained models are capable of few-shot learning [1], performance is often improved via supervised fine-tuning. However, full fine-tuning of a large model is highly resource intensive, making it impractical for many researchers. Parameter-efficient fine-tuning methods emerge to address this issue.

A popular technique for parameter-efficient fine-tuning is LoRA [2]. Given a weight matrix $\mathbf{W} \in \mathbb{R}^{n \times m}$ that maps vectors $\mathbf{x} \in \mathbb{R}^m$ to $\mathbf{W}\mathbf{x} \in \mathbb{R}^n$, LoRA proposes two matrices $\mathbf{A} \in \mathbb{R}^{r \times m}$, $\mathbf{B} \in \mathbb{R}^{n \times r}$ with $r \ll \min(m, n)$, and maps $\mathbf{x}$ to $\mathbf{W}\mathbf{x} + \mathbf{B}\mathbf{A}\mathbf{x}$. Only the low-rank matrices $\mathbf{A}$ and $\mathbf{B}$ are trained in the fine-tuning stage, a significant saving of compute and memory compared to full fine-tuning.

While the idea of LoRA is straightforward, the devil is in the details when applying it to an actual task. For example, which weights to adapt and what ranks to use? And how do we assess the quality of a LoRA solution versus a fully fine-tuned solution?

In this project, we go through the complete process of fine-tuning a GPT-2 model, using full fine-tuning and LoRA, respectively. Through careful analysis and hyperparameter tuning, we find that LoRA indeed approaches full fine-tuning accuracy while consuming a tiny fraction of parameters. Our experiments offer some insight on the best practice of applying these methods. Some of our findings confirm previous results, while others challenge them.

The particular task we apply fine-tuning on is paraphrase detection, i.e. to determine whether two sentences express the same meaning. Concretely, given two sentences x and y , we construct the cloze-style prompt

Is " x " a paraphrase of " y "? Answer "yes" or "no":

and have GPT-2 [3] predict the next token. The expected token would be “yes” or “no” according to the specific case. It turns out that GPT-2 with pretrained weights is extremely bad at this – in fact, it tends to paraphrase parts of the prompt! See Appendix A for a sample of its responses. Fine-tuning proves highly effective for this task – the accuracy leaps from 0% to near 90% after either full fine-tuning or LoRA.

3 Related Work

Since the publication of LoRA [2], numerous extensions have been proposed. A comprehensive review is given in [4]. Below we mention a few relevant to our experiments.

Extensive empirical experiments have shown that low-rank weight updates are effective for task-specific adaptation. But why are they so? Some hint is provided by [5], where it is found that the parameters of large language models actually reside in a low dimensional space. A similar observation is made in [6], where it is shown that low-dimensional weights are effective for task-specific training and that the number of intrinsic dimensions do not vary much across model architectures. Finally, [7] provides an explanation through the Neural Tangent Kernel theory.

The next natural questions for applying LoRA are which weight matrices to adapt and what rank to use. The original LoRA paper [2] only adapts the attention layers, but [8] finds fine-tuning accuracy to be dominated by the MLP layers, and hence recommends adapting all weight matrices.

As for rank selection, an obvious strategy to try different values across orders of magnitude; this is the approach followed by [8]. More sophisticated strategies have also been explored, such as rank adaptation (AdaLoRA [9], SoRA [10]), scheduling (PRILoRA [11]), and mixture (DyLoRA [12]). Other approaches include rank augmentation such as combining a series of low-rank updates.

Additional details with respect to parameterization and initialization are relevant for training LoRA effectively. The original paper [2] scales weight updates by the constant factor α/r where r is the rank; the purpose is to keep the optimal learning rate approximately invariant as the rank varies. On the other hand, [13] advocates a scaling factor of $\alpha/\sqrt{r}$ to ensure stable learning. As for parameter initialization, the original paper [2] initializes $\mathbf{B}$ with zero and $\mathbf{A}$ with Gaussian noise of variance $1/d_{in}$ where d_{in} is the input dimension. However, [14] discovers that freezing a randomly-initialized orthogonal matrix $\mathbf{A}$ and only training $\mathbf{B}$ achieves similar accuracy as training both $\mathbf{A}$ and $\mathbf{B}$. Empirically, [8] finds that using the scaling factor $32/r$, initializing $\mathbf{A}$ with variance $1/d_{in}$, and training both $\mathbf{A}$ and $\mathbf{B}$ admits an approximately invariant optimal learning rate across ranks.

4 Approach

We first implement building blocks of GPT-2 [3] and AdamW [15], starting from the skeleton code provided. The code is verified by achieving accuracies on par with the staff baseline on two sentiment analysis tasks. We then implement a `LoraLinear` PyTorch “module” and substitute it for every `Linear` layer of the GPT-2 architecture. Our implementation allows each weight matrix to be tuned with a different rank, where a rank of zero disables LoRA for that matrix. This allows us to experiment with combination of ranks over different parts of the architecture.

We apply LoRA to all six types of weight matrices in GPT-2: four in multi-headed attention (MHA) layers (query transform $\mathbf{W}_q$, key transform $\mathbf{W}_k$, value transform $\mathbf{W}_v$ and output transform $\mathbf{W}_o$), and two in multi-layer perceptron (MLP) layers (up-projection $\mathbf{W}_u$ and down-projection $\mathbf{W}_d$).

The baseline for our task is full fine-tuning. Since the “degrees of freedom” of LoRA is a subset of full fine-tuning, it is mathematically impossible for LoRA to attain higher accuracy than full fine-tuning if both are trained properly. However, it is possible for LoRA to *approach* the accuracy of full fine-tuning with a fraction of parameters. Therefore, we would assess the utility of LoRA from two aspects: *efficiency*, measured by the number of parameters to train, and *effectiveness*, measured by the relative accuracy compared to full fine-tuning.

5 Experiments

5.1 Data

We use the provided Quora dataset for training and validation. The training set comprises 283,003 examples (question pairs), of which 104,579 (37%) are paraphrases. Each cloze-encoded example contains 47 tokens on average. The dev set comprises 40,429 examples, of which 14,892 (36.8%) are paraphrases. The average cloze-encoded length is also 47 tokens.

5.2 Evaluation method

We measure the efficiency of LoRA by the number of parameters to train, and its effectiveness by the validation (dev) accuracy. Both metrics are compared against the full fine-tuning baseline.

5.3 Experimental details

We use as base model the GPT2-small variant, with vocabulary size 50,257 and embedding dimension 768. The architecture comprises 12 decoder blocks, with each block containing 12 heads in the MHA layer and 3,072 hidden units in the MLP layer. The total number of parameters is 125,030,400. The pretrained weights are loaded from Hugging Face.

Optimization is performed by the custom-made AdamW optimizer, using $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-6}$ and $\lambda = 0$ (i.e. no weight decay). Dropout is activated in four places during training: after the sum of token and position embeddings, after computing the attention weights (i.e. the softmax), at the end of MHA layers (after applying the affine transform involving $\mathbf{W}_o$), and at the end of MLP layers (after applying the affine transform involving $\mathbf{W}_d$). The dropout rate is set to 0.1 by default.

The experiments are carried out on rented GPUs (from vast.ai) ranging from L40 (\$0.5/hour) to H200 (\$2/hour). All computations are performed in 32-bit float. The compute cost about \$100 in total.

5.4 Results and Analysis

Leaderboard Results Since the leaderboard is accuracy oriented, we submit results from the baseline (i.e. full fine-tuning). For the paraphrase detection task, the parameters are fine-tuned for 10 epochs with batch size 32, learning rate 5e-5, and dropout rate 0.1. The leaderboard scores are:

- Paraphrase detection dev: 0.897
- Paraphrase detection test: 0.896
- Sonnet generation dev: 41.023
- Sonnet generation test: 40.941

In the following we describe in detail the procedure to perform full fine-tuning and LoRA. Properties and results of the two methods are compared along the way.

Hyperparameter Sweep for Full Fine-Tuning To obtain a proper baseline, we sweep batch sizes and learning rates across orders of magnitude to find the best combination. Concretely, we sweep six batch sizes (8, 16, 32, 64, 128 and 256) by ten learning rates ($1e-7$, $1e-6$, $1e-5$, $2e-5$, $5e-5$, $1e-4$, $2e-4$, $5e-4$, $1e-3$ and $1e-2$). For each of these sixty hyperparameter combinations, a single epoch of full fine-tuning is run and the training loss and dev accuracy by the end of the epoch are reported.¹ The results are summarized in Figure 1.²

Figure 1 reveals a mild monotonic relationship between batch size and optimal learning rate. For batch size 8, 16, 32, 64, 128, 256, the optimal learning rate (that minimizes training loss) is $2e-5$, $5e-5$, $5e-5$, $1e-4$, $1e-4$ and $1e-4$, respectively. The pattern with dev accuracy is similar.

¹The reported training loss is the average loss of all batches of the epoch. Consequently, earlier batches are evaluated on inferior parameters and later batches are evaluated on superior parameters. Since convergence is usually quick, the average loss is dominated by the later batches. On the other hand, the dev accuracy is evaluated using the final parameters.

²Batch size 64 and below are trained on an L40 with 45GB VRAM. Batch size 128 and 256 are trained on an H200 with 140GB VRAM because the former runs out of VRAM.

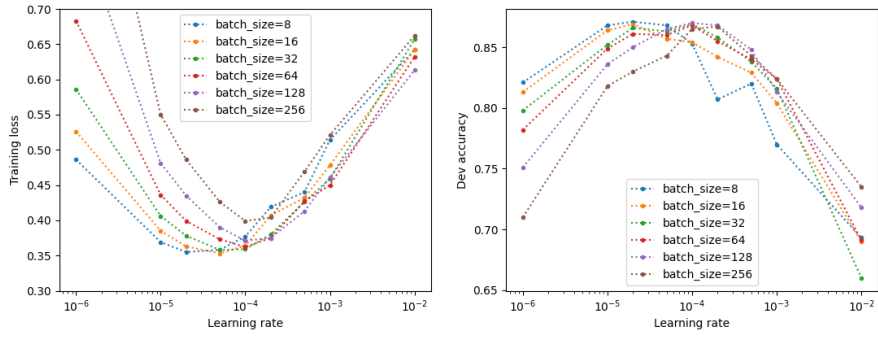


Figure 1: Full Fine-Tuning Performance by Batch Size and Learning Rate

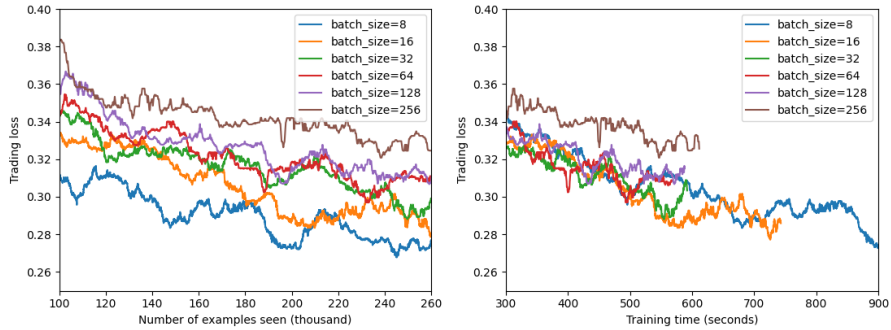


Figure 2: Full Fine-Tuning Learning Curve by Batch Size

Figure 1 also shows that a smaller batch size achieves lower training loss and higher dev accuracy by the end of one epoch. This is likely due to the fact that more gradient descent steps are taken per epoch for smaller batch size. On the other hand, an epoch runs fastest for moderate batch size and takes longer for small and large batch sizes. Specifically, over three runs on the same machine, the average time it takes to complete one epoch is 1228, 743, 592, 567, 588 and 612 seconds for batch size of 8, 16, 32, 64, 128 and 256 respectively.

A fair comparison of training efficiency (in terms of time) should thus be controlled for training time. This is done in Figure 2. The charts show that while smaller batch size achieves lower loss for the same number of examples seen, for the same amount of training time the loss achieved by different batch sizes appear to be close (except for batch size 256, which performs much worse). This means we are free to pick the batch size as long as a suitable learning rate is used accordingly. We proceed with batch size 32 and learning rate $5e-5$ because it is faster than batch size 16 and puts less pressure on memory than batch size 64.

With the chosen hyperparameters, we perform ten epochs of full fine-tuning to develop the baseline. To assess the impact of dropout on training and validation error, we perform four runs with dropout probability 0.0, 0.1, 0.3 and 0.5, respectively. Each run takes about 100 minutes. The learning curves are shown in Figure 3.

Figure 3 shows that the training loss drops steadily with epoch and has room to decrease further beyond 10 epochs. Regardless, we stop after 10 epochs due to resource constraints. A lower dropout rate corresponds to lower training loss, as expected. Interestingly, a dropout rate of 0.1 (the default for GPT-2) proves to achieve the highest dev accuracy. With this observation, we finalize our baseline to use batch size 32, learning rate $5e-5$, dropout rate 0.1, and 10 epochs of training.

Empirical Rank of Full Fine-Tuning Weight Updates Before attempting LoRA, we are interested to find out whether weight updates produced by full fine-tuning are empirically low-rank. And if they are, could they provide clue on the rank to choose for LoRA?

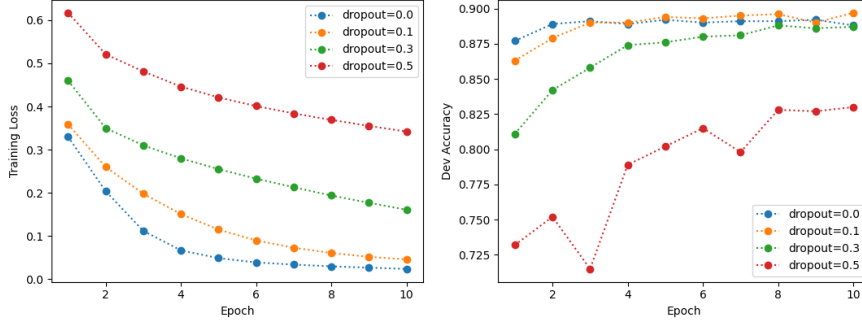


Figure 3: Full Fine-Tuning Learning Curve by Dropout Rate

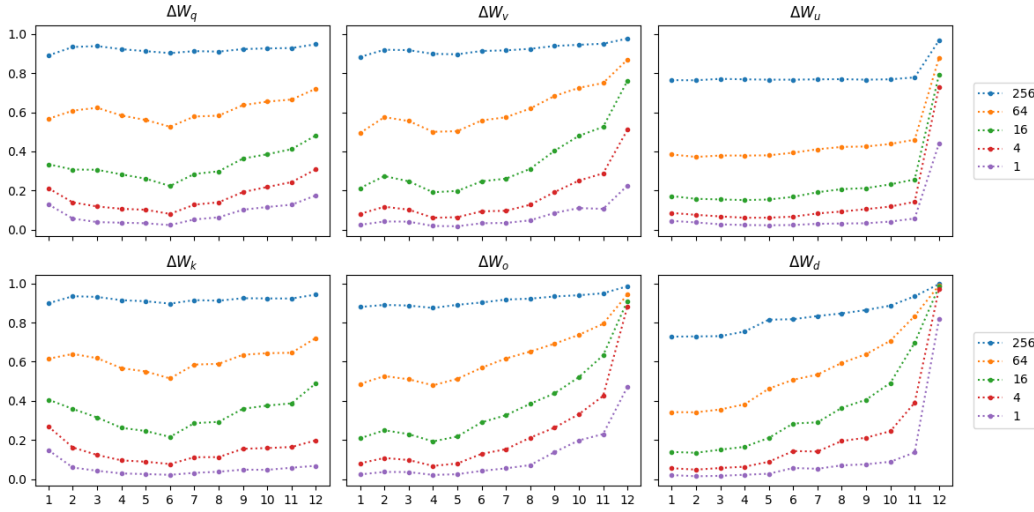


Figure 4: Empirical Rank of Full Fine-Tuning Weight Updates

To investigate this, let $\mathbf{W}$ denote the pretrained weights, $\mathbf{W}^*$ the fine-tuned weights, and $\Delta \mathbf{W} := \mathbf{W}^* - \mathbf{W}$ the weights update. Note that $\Delta \mathbf{W}, \mathbf{W}, \mathbf{W}^* \in \mathbb{R}^{n \times m}$. Since the rank of a matrix is equal to the number of non-zero singular values, we perform singular value decomposition of $\Delta \mathbf{W}$ by writing $\Delta \mathbf{W} = \mathbf{U} \Sigma \mathbf{V}^\top$ where $\mathbf{U} \in \mathbb{R}^{n \times n}$, $\mathbf{V} \in \mathbb{R}^{m \times m}$ are orthogonal and $\Sigma \in \mathbb{R}^{n \times m}$ is diagonal with diagonal elements $\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_{\min(m,n)} \geq 0$. Since a numerical matrix is never exactly rank deficient, we assess its “closeness” to a low-rank approximation in terms of the squared Frobenius norm, $\|\Delta \mathbf{W}\|_F^2$, i.e. the sum of squares of matrix elements. This quantity is related to singular values by $\|\Delta \mathbf{W}\|_F^2 = \text{tr}(\Delta \mathbf{W} \Delta \mathbf{W}^\top) = \text{tr}(\mathbf{U} \Sigma \mathbf{V}^\top \mathbf{V} \Sigma^\top \mathbf{U}^\top) = \text{tr}(\mathbf{U}^\top \mathbf{U} \Sigma \Sigma^\top) = \sum_{k=1}^{\min(m,n)} \sigma_k^2$.

Figure 4 summarizes the low-rank approximations to the weight updates. The six subplots correspond to the six types of matrices in the transformer architecture. In each subplot, the x -axis represents the layer of the transformer block (with layer 1 being closest to the input and layer 12 closest to the output), and the y -axis represents the proportion of squared Frobenius norm contributed by the leading r singular values. The purple line corresponds to $r = 1$, the red line to $r = 4$, and so on.

A few interesting observations are made from Figure 4. First, there is a clear uptrend in the portion of weight update accounted for by a given rank as we move up the transformer layers. That is, layers near the top (i.e. closer to the output) tend to admit lower-rank weight updates than layers near the bottom (i.e. closer to the input). For example, the leading singular value of $\Delta \mathbf{W}_d$ accounts for 80% of the weight update of the last layer but only 15% of the second-to-last layer. This observation is in stark contrast to that of [11], where it is claimed that layers closer to the top require more ranks to approximate. We do not have an explanation for this discrepancy.

Rank	# Params % FullFT	Learning Rate						
		1e-5	5e-5	1e-4	2e-4	5e-4	1e-3	1e-2
1	165,888	0.569	0.438	0.410	0.411	0.401	0.626	0.699
	0.13%	0.805	0.840	0.838	0.850	0.814	0.632	0.632
2	331,776	0.572	0.435	0.405	0.401	0.390	0.581	0.704
	0.26%	0.805	0.835	0.847	0.835	0.847	0.744	0.632
4	663,552	0.576	0.435	0.402	0.396	0.381	0.580	0.694
	0.53%	0.807	0.841	0.850	0.857	0.849	0.725	0.932
8	1,327,104	0.576	0.434	0.399	0.394	0.375	0.531	0.690
	1.06%	0.784	0.843	0.857	0.854	0.836	0.787	0.632
16	2,654,208	0.575	0.433	0.399	0.394	0.372	0.469	0.699
	2.12%	0.808	0.838	0.848	0.854	0.828	0.671	0.632
32	5,308,416	0.574	0.433	0.399	0.393	0.368	0.423	0.680
	4.24%	0.807	0.843	0.856	0.856	0.852	0.820	0.627
64	10,616,832	0.575	0.457	0.398	0.394	0.381	0.419	0.685
	8.49%	0.808	0.837	0.857	0.861	0.841	0.818	0.632
128	21,233,664	0.577	0.434	0.399	0.391	0.366	0.458	0.679
	16.98%	0.805	0.842	0.855	0.862	0.829	0.720	0.632
256	42,467,328	0.580	0.460	0.401	0.392	0.380	0.454	0.698
	33.96%	0.803	0.835	0.849	0.862	0.856	0.797	0.626

Table 1: LoRA Training Loss (Upper) and Dev Accuracy (Lower)

The second observation from Figure 4 is that the weights of the MLP layers appear to be “harder” to approximate than those of the MHA layers. For example, with rank 64 the MHA matrices are roughly approximated at 60%, but most of the MLP matrices are only approximated at 40%. Another way to look at this is that in order to approximate the weight updates to a certain percentage, MLP matrices appear to require double the rank of MHA matrices. These observations could be relevant when we select the LoRA rank later.

Hyperparameter Sweep for LoRA We proceed to investigate the performance LoRA. Like before, we start by choosing appropriate hyperparameters, in particular the rank and the learning rate.

To do this, we sweep ranks 1, 2, 4, 8, 16, 32, 64, 128 and 256 and learning rates 1e-5, 5e-5, 1e-4, 2e-4, 5e-4, 1e-3 and 1e-2. For each hyperparameter combination, we apply LoRA to all six matrices with the given rank and train both **A** and **B** for one epoch. Following [8], we use the scaling factor $32/r$ and initialize **A** with Gaussian noise of variance $1/768$. The performance of the hyperparameter combinations are reported in Table 1 and summarized in Figure 5.

Figure 5 confirms a few stylized facts reported in [8]. First, the optimal learning rate indeed appears to be invariant (5e-4 in our case) across ranks. This adds to evidence that supports the validity of the scaling factor $1/r$. Second, the optimal learning rate of LoRA in terms of training loss (5e-4 in our case) is 10 times that of full fine-tuning (5e-5 in our case). Third, LoRA of lower rank appears more sensitive to large learning rate than LoRA of higher rank, illustrated for example by the sharp rise of training loss of rank one as the learning rate goes from 5e-4 to 1e-3.

On the other hand, if optimality is measured in terms of dev accuracy, then the optimal learning rate is lowered across the board to about 2e-4, which is now four times that of full fine-tuning. The discrepancy between in-sample and out-of-sample optimal learning rates lends itself to an investigation of the generalization property of LoRA. We did not investigate further, but sophisticated dropout strategies discussed in [16] might be helpful in this regard.

It is interesting to compare the dev accuracy of different LoRA ranks (given one epoch of training). Table 1 shows roughly three plateaus of performance: rank 1 and 2 achieves an accuracy of 85.0%, rank 4 to 32 improves this to 85.7%, and rank 64 and above further improves to 86.2%. Given that

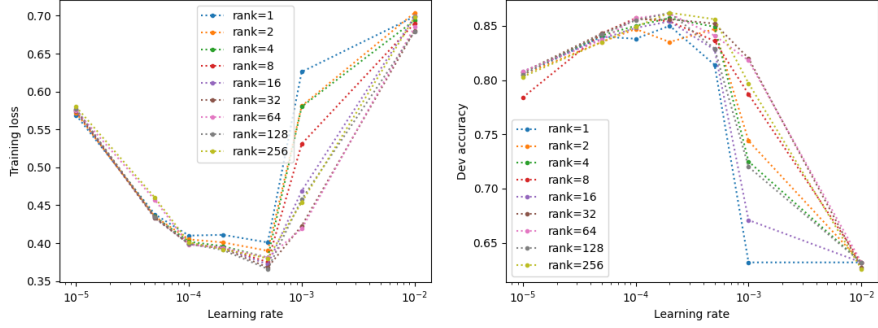


Figure 5: LoRA Performance by Rank and Learning Rate

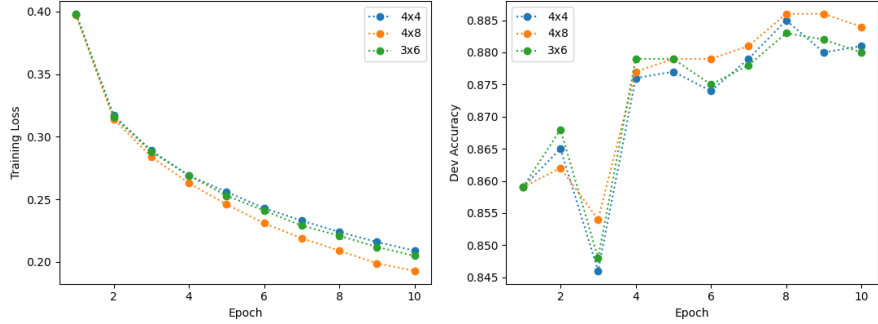


Figure 6: LoRA Learning Curve

the pretrained weights have zero accuracy, the effectiveness of rank-one update looks particularly impressive. Rank 4 and 64 also appear to be good candidates for further training as they offer decent performance improvements.

We would like to offer some intuition on the effectiveness of rank-one LoRA by referring ourselves back to the plots of the empirical rank of full fine-tuning weight updates in Figure 4. There we observe that the update to the down-projection matrix $\Delta \mathbf{W}_d$ of the top layer has extremely low rank – rank-one approximates it to 80% and rank-four approximates it to over 95%. Since the last layer is directly responsible for generating the next token, it may be playing the critical role of getting the model to say “yes” or “no” instead of making a verbose or ambiguous response. The full fine-tuning weight update tells us that this part only takes very low rank to get right, which might explain the effectiveness of rank-one LoRA.

LoRA Training Based on results of the hyperparameter sweep, we select rank 4 and learning rate $2e-4$ as the LoRA configuration of choice (labeled 4x4) and train it for 10 epochs. We additionally train two configurations for comparison: rank 4 for MHA matrices plus rank 8 for MLP matrices (labeled 4x8), and rank 3 for MHA matrices plus rank 6 for MLP matrices (labeled 3x6). The additional configurations serve to confirm whether doubling the rank of MLP adaptation improves accuracy, as Figure 4 suggests that the MLP updates have higher rank than the MHA updates.

The learning curves of the three LoRA configurations are shown in Figure 6. We note that the 4x8 configuration consistently achieves lower training loss and higher dev accuracy, while the 3x6 configuration performs on par with the 4x4 configuration. Considering that the 4x8 and 3x6 configurations have respectively 55% and 16% more parameters than the 4x4 configuration,³ the results do not support the presence of any benefit of doubling the rank for MLP adaptation over adapting all matrices with the same rank.

³The number of parameters in the 4x4, 3x6 and 4x8 configurations are 663,552, 774,144 and 1,032,192, respectively.

Method	# Params	Training Loss	Dev Accuracy	Learning Rate
Full FT	125,030,400	0.046	0.897	5e-5
LoRA	663,552	0.209	0.885	2e-4
%	0.53%	4.54x	98.66%	4x

Table 2: Comparison of LoRA and Full Fine-Tuning Performance

We conclude this section by comparing the performance of the chosen LoRA configuration with the chosen baseline in Table 2. Both are trained for 10 epochs with batch size 32 and dropout rate 0.1. Note that LoRA achieves 98.66% of the dev accuracy of full fine-tuning with mere 0.53% parameters. The efficiency and effectiveness of LoRA is confirmed!

6 Conclusion

Using self-implemented code and through careful hyperparameter tuning, we apply LoRA to a paraphrase detection task and approach the full fine-tuning accuracy using a fraction of parameters. Of particular interest is an analysis via singular value decomposition of the empirical rank of full fine-tuning weight updates, the patterns it reveals, and how it might offer an intuitive explanation for the effectiveness of LoRA. It would be interesting to find out if more specific relations between the two could be established.

We have made every effort to make the experiments thorough and solid, but an obvious limitation is the shortage of training – ten epochs are just not enough and most learning curves still appear far from plateauing. We would definitely have trained longer if not constrained by resources. This also reveals a downside of using exclusively self-implemented code: our code is likely less efficient than state of the art by a constant factor if not worse.

Finally, we note that the cloze-style task is little more than binary classification in disguise, and there is a long way to go from getting “yes” or “no” right to generating a well-received response. Fine-tuning for the latter objective would be an interesting topic for future experiments.

References

- [1] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D 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 Ziegler, Jeffrey Wu, Clemens Winter, Chris 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. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc., 2020.
- [2] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022.
- [3] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019.
- [4] Menglin Yang, Jialin Chen, Jinkai Tao, Yifei Zhang, Jiahong Liu, Jiasheng Zhang, Qiyao Ma, Harshit Verma, Regina Zhang, Min Zhou, Irwin King, and Rex Ying. Low-rank adaptation for foundation models: A comprehensive review, 2025.
- [5] Armen Aghajanyan, Sonal Gupta, and Luke Zettlemoyer. Intrinsic dimensionality explains the effectiveness of language model fine-tuning. pages 7319–7328, Online, 2021. Association for Computational Linguistics.
- [6] Chunyuan Li, Heerad Farkhor, Rosanne Liu, and Jason Yosinski. Measuring the intrinsic dimension of objective landscapes, 2018.

- [7] Sadhika Malladi, Alexander Wettig, Dingli Yu, Danqi Chen, and Sanjeev Arora. A kernel-based view of language model fine-tuning. In *ICLR 2023 Workshop on Mathematical and Empirical Understanding of Foundation Models*, 2023.
- [8] John Schulman and Thinking Machines Lab. Lora without regret. *Thinking Machines Lab: Connectionism*, 2025.
- [9] Qingru Zhang, Minshuo Chen, Alexander Bukharin, Nikos Karampatziakis, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. AdaLoRA: Adaptive budget allocation for parameter-efficient fine-tuning. 2023.
- [10] Ning Ding, Xingtai Lv, Qiaosen Wang, Yulin Chen, Bowen Zhou, Zhiyuan Liu, and Maosong Sun. Sparse low-rank adaptation of pre-trained language models, 2023.
- [11] Nadav Benedek and Lior Wolf. PRILoRA: Pruned and rank-increasing low-rank adaptation. In Yvette Graham and Matthew Purver, editors, *Findings of the Association for Computational Linguistics: EACL 2024*, pages 252–263, St. Julian’s, Malta, March 2024. Association for Computational Linguistics.
- [12] Mojtaba Valipour, Mehdi Rezagholizadeh, Ivan Kobzyev, and Ali Ghodsi. DyLoRA: Parameter-efficient tuning of pre-trained models using dynamic search-free low-rank adaptation. In Andreas Vlachos and Isabelle Augenstein, editors, *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*, pages 3274–3287, Dubrovnik, Croatia, May 2023. Association for Computational Linguistics.
- [13] Damjan Kalajdzievski. A rank stabilization scaling factor for fine-tuning with lora. 2023.
- [14] Jiacheng Zhu, Kristjan Greenewald, Kimia Nadjahi, Hartz Sáez de Ocáriz Borde, Rickard Brüel Gabriellsson, Leshem Choshen, Marzyeh Ghassemi, Mikhail Yurochkin, and Justin Solomon. Asymmetry in low-rank adapters of foundation models. In *Forty-first International Conference on Machine Learning*, 2024.
- [15] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019.
- [16] Sheng Wang, Liheng Chen, Jiyue Jiang, Boyang Xue, Lingpeng Kong, and Chuan Wu. LoRA meets dropout under a unified framework. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Findings of the Association for Computational Linguistics: ACL 2024*, pages 1995–2008, Bangkok, Thailand, August 2024. Association for Computational Linguistics.

A Sample Responses to Paraphrase Detection Queries

The paraphrase detection task is done by cloze-style queries. Below we list a sample of the responses to the such queries, generated by the pretrained model and the fully fine-tuned model, respectively. Ten tokens are generated greedily for each query. The model probability for the tokens are displayed below.

Prompt: Is "what are some great ways to generate passive income in india ?" a paraphrase of "can a full time employee be successful in investing on stock markets and other potential investments to generate passive income ?"? Answer "yes" or "no":

Pretrained: "I don't know. I'm not

Prob: 0.42, 0.09, 0.07, 0.12, 1.00, 0.37, 0.17, 0.28, 0.12, 0.26

Full FT: nononononononononono

Prob: 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00

Prompt: Is "if i am on private on twitter and i favorite a tweet will it be seen/notified by the person i favorited ?" a paraphrase of "how can i know how much favorites i have on twitter ?"? Answer "yes" or "no":

Pretrained: ?"

"If i am on private on

Prob:	0.29, 0.03, 0.99, 0.09, 0.06, 0.27, 0.20, 0.66, 0.35, 0.51
Full FT:	nononononononononono
Prob:	1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00

Prompt:	Is "how long can you leave cooked chicken wings out at room temp ?" a paraphrase of "can you leave raw chicken out all night ?"? Answer "yes" or "no":
Pretrained:	"can you leave raw chicken wings out at
Prob:	0.36, 0.23, 0.06, 0.84, 0.61, 0.26, 0.89, 0.44, 0.67, 0.47
Full FT:	yesyesyesyesyesyesyesyesyesyes
Prob:	0.57, 0.61, 0.61, 0.61, 0.60, 0.61, 0.60, 0.60, 0.60, 0.61

Prompt:	Is "what is the peace sign in australia , and how did it came to be the peace symbol ?" a paraphrase of "write something about your team "? Answer "yes" or "no":
Pretrained:	"I don't know. I think it
Prob:	0.44, 0.18, 0.06, 0.12, 1.00, 0.48, 0.20, 0.27, 0.13, 0.28
Full FT:	nononononononononono
Prob:	1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00

Prompt:	Is "is it biologically good or bad to marry other caste ?" a paraphrase of "is it good or bad to marry other caste ?"? Answer "yes" or "no":
Pretrained:	"yes" or "no" "
Prob:	0.49, 0.13, 0.05, 0.52, 0.54, 0.89, 0.92, 0.50, 0.08, 0.05
Full FT:	yesyesyesyesyesyesyesyesyesyes
Prob:	0.99, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00