MoRe Fine-Tuning with 10x Fewer Parameters

Wenxuan Tan [1] **Nicholas Roberts** [1] **TzuHeng Huang** [1] **Jitian Zhao** [1] **John Cooper** [1] **Samuel Guo** [1] **Chengyu Duan** [1] **Frederic Sala** [1]

Abstract

Parameter-efficient fine-tuning (PEFT) techniques have unlocked the potential to cheaply and easily specialize large pretrained models. However, the most prominent approaches, like low-rank adapters (LoRA) depend on heuristics or rules-of-thumb for their architectural choices—potentially limiting their performance for new models and architectures. This limitation suggests that techniques from neural architecture search could be used to obtain optimal adapter architectures, but these are often expensive and difficult to implement. We address this challenge with Monarch Rectangular Fine-tuning (MoRe), a simple framework to search over adapter architectures that relies on the Monarch matrix class. Theoretically, we show that MoRe is more expressive than LoRA. Empirically, our approach is more parameter-efficient and performant than state-of-the-art PEFTs on a range of tasks and models, with as few as 5% of LoRA's parameters.

1. Introduction

Large pretrained 'foundation' models (Bommasani et al., 2021) were originally conceived as a convenient base for rapidly building applications. The size and complexity of these models; however, paradoxically often made specialization more complex and challenging than traditional machine learning. Recently, adapters, like the popular LoRA (Hu et al., 2021), have dramatically decreased the cost of specialization. This has unlocked the potential of foundation models for efficient use in everyday settings

Despite their popularity, parameter-efficient adapter techniques make particular architectural assumptions, such as the eponymous low rank in LoRA (Hu et al., 2021; 2023; Zhang et al., 2023; Chavan et al., 2023; Liu et al., 2024b). These assumptions are a good fit for certain models, tasks, and datasets—but may result in poor performance on others. There has been a resulting arms race of parameter-efficient fine-tuning (PEFTs) techniques, each with their own benefits and drawbacks. This suggests that the right adapter architecture should be *learnable*.

Learning architectures is the traditional domain of neural architecture search (NAS). Unfortunately, most NAS techniques are heavyweight (Pham et al., 2018; Liu et al., 2019a; Li & Talwalkar, 2020; Li et al., 2021), creating a tension: NAS may learn better adapter architectures for a particular task but costs substantially more compute—sacrificing much of the benefits of adapters in the first place.

We show how to resolve this tension by relying on the Monarch matrix class (Dao et al., 2022a). This class presents a simple parametrization that can express a vast range of *structured matrices*, enabling conveniently learning a wide variety of parameter-efficient architectures. In other words, building adapters from Monarch matrices simultaneously produces two benefits—**flexibly searching over architectures** and **efficient training for adapters**.

Based on this idea, we introduce a simple PEFT framework called Monarch Rectangular Fine-tuning (MoRe). We study its expressiveness properties theoretically and validate it empirically. When fixing block configuration after extensive architectural ablations, the most performant adapter we produced via MoRe is $10\times - 20\times$ more parameter-efficient than LoRA and has the **fewest tunable hyperparameters** among all PEFTs.

2. Related Work

PEFT methods trade off mechanisms for parameter-efficiency and performance. These mechanisms are designed heuristically, and may not be the best choice for all settings. Popular methods such as LoRA (Hu et al., 2021) may not strike the best tradeoff between efficiency and performance. Other methods often sacrifice increased complexity and a reliance on search for improved performance, limiting scalability. Methods such as GLoRA and AdaLora (Chavan et al., 2023; Zhang et al., 2023) require

[1]Department of Computer Sciences, University of Wisconsin—Madison, Madison, WI, USA. Correspondence to: Wenxuan Tan <wtan45@wisc.edu>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024. Copyright 2024 by the author(s).

expensive search for their rank and block configurations. Our goal is to strike a better tradeoff compared to current PEFT techniques, all while avoiding expensive search procedures. We describe the relation between MoRe and existing techniques below.

Orthogonal Fine-tuning. Butterly Orthogonal Fine-tuning (BOFT) (Liu et al., 2024b) uses a compute-heavy neighbor of Monarch, the butterfly matrices, to parameterize Cayley-transformed orthogonal blocks that are multiplied with the original weight matrices. It has two more tunable parameters, the block number and size. In contrast, MoRe does not require tuning the number of blocks or rank and is more performant and parameter-efficient than BOFT.

Representation Editing. ReFT (Wu et al., 2024) is a prompt editing method operating on low-rank subspaces. It balances expressivity and parameter efficiency by only intervening on selected layers and tokens, often surpassing PEFTs that adapt model weights. However, it induces inference overhead and an even larger search space than LoRA (token positions and layers to intervene). It is also somewhat less well-understood compared to existing PEFTs from a theoretical point of view.

3. MoRe Framework

Monarch matrices (Dao et al., 2022a) are a rich collection of block-sparse structured matrices that subsume butterfly matrices and belong to the Kaleidoscope matrices (Dao et al., 2020), a class of matrices that can represent any structured matrix and a variety of transforms such as the Fourier transform, cosine transforms, and Hadamard transform. Unlike structure-agnostic matrix families, such as those of low rank, Monarch matrices can have arbitrary rank, and their products are not closed, allowing *for a richer matrix class* as more Monarch matrices are multiplied.

Let n be the dimensions of the Monarch matrix M, i.e. $M \in \mathbb{R}^{n \times n}$. Define N as the number of blocks in component matrices L and R and r_{blk} as the rank of each sub-block. In the standard formulation, $r_{blk} = n/N$. Monarch matrices have the following structure:

$$M = P_1 L P_2 R, \tag{1}$$

where P_1 and P_2 are permutation matrices and L and R are block diagonal (see Figure 1).

Original work in Monarch matrices focused on the case where L and R are block-wise square, but the family of Monarch matrices is more general and includes *low-rank* Monarch matrices. This extension allows for L and R to be rectangular, with similarly shaped block diagonal components. This allows the overall rank of the Monarch matrix to be constrained by forcing L and R to have similar shapes to LoRA components—but with fewer parameters, as Monarch

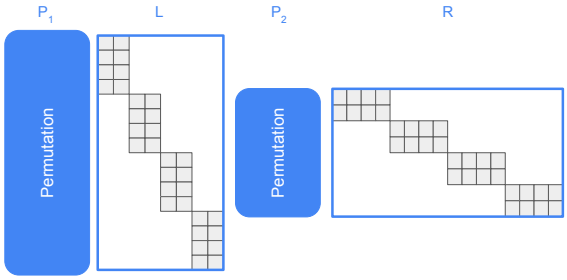

Figure 1. The structure of low-rank Monarch matrices contains two permutations P_1 and P_2 along with two block-diagonal components L and R which are learned while P_1 and P_2 are both fixed. In the above, the number of blocks $N = 4$, with input dimension $n = 16$, and the block 'rank' is $r_{blk} = 2$ and size is $n/N = 4$. **The pseudo-code can be found in appendix G.**

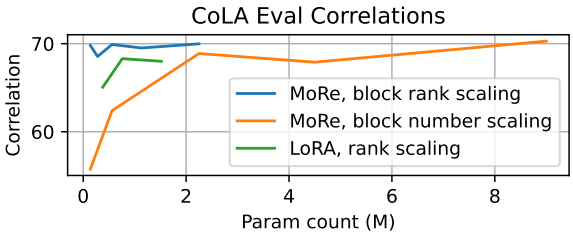

Figure 2. Matthew's Correlation on CoLA when trade parameter counts for performance on two axes: the block dimension and the number of blocks, both with square blocks. The block dimensions used are $[4, 8, 16, 32, 64]$ and the N are $[1024, 256, 128, 32, 16]$.

only contains non-zero entries within the diagonal blocks.

MoRe Fine-Tuning. During training, for a pretrained weight matrix $W \in \mathbb{R}^{m \times n}$ and bias $b \in \mathbb{R}^m$, we apply MoRe via:

$$\Phi_{MoRe}(x) = Wx + Mx + b, \tag{2}$$

and update L and R, where L has shape $(N, r_{blk}, n/N)$ and R has shape $(N, n/N, r_{blk})$. During inference, W absorbs M as in LoRA so there is zero additional overhead.

Monarch matrices were originally proposed to accelerate pre-training, using two block-wise square monarch factors to substitute one dense matrix multiplication, with $O(n\sqrt{n})$ FLOPs. However, an interesting property of these matrices from rectangular factors is that even though each block is constrained to rank r_{blk}, the overall product can have a rank as large as $r = Nr_{blk}$. We set N to 4 for the best rank-sparsity balance. MoRe can achieve the same rank as LoRA with far fewer parameters, which empirically translates to added fine-tuning performance.

MoRe Fine-Tuning with 10x Fewer Parameters

Method	#Params	BoolQ	PIQA	SIQA	HellaS.	WinoG.	ARC-e	ARC-c	OBQA	Avg.
LoRA$_{r=32}$	53.3M (0.830%)	68.9	80.7	77.4	78.1	78.8	77.8	61.3	74.8	74.7
LoRA$_{r=32}$, **Llama 13B**	83.2M (0.670%)	72.1	83.5	80.5	90.5	83.7	82.8	68.3	82.4	80.5
MoRe$_{r=32}$; q, k, v (ours)	**3M (0.047%)**	67	86.4	**88.4**	**97.3**	**95.1**	**88.5**	**76.6**	79.9	**84.9**
ReFT	**2.0M (0.031%)**	69.3	84.4	80.3	93.1	84.2	83.2	68.2	78.9	80.2
ReFT, **Llama 13B**	3.1M (0.025%)	72.1	**86.3**	81.8	95.1	87.2	86.2	73.7	**84.2**	83.3
Adapter^{S*}	99.3M (0.800%)	71.8	83.0	79.2	88.1	82.4	82.5	67.3	81.8	79.5
Adapter^{P*}	358.7M (2.890%)	72.5	84.9	79.8	92.1	84.7	84.2	71.2	82.4	81.5
DoRA (half)*	43.4M (0.350%)	**72.5**	85.3	79.9	90.1	82.9	82.7	69.7	**83.6**	80.8
DoRA	84.4M (0.680%)	72.4	84.9	81.5	92.4	84.2	84.2	69.6	82.8	81.5
ChatGPT	–	73.1	85.4	68.5	78.5	66.1	89.8	79.9	74.8	77.0

Table 1. Commonsense reasoning results. We take all numbers except for MoRe from Liu et al. (2024a). Llama 1 7B is used unless otherwise specified.

PEFT	#Params	AQuA	GSM8K	MAWPS	SVAMP	Avg.
LoRA$_{r=32}$	53.3M (0.830%)	18.9	**37.5**	79.0	**52.1**	46.9
MoRe$_{r=32}$; q, k, v (ours)	**3M (0.047%)**	22.1	28.5	84.3	48.4	45.8
MoRe$_{r=32}$ (ours)	10.68M (0.166%)	**24.0**	29.6	**85.7**	48.7	**47.0**
ReFT	**1.99M (0.031%)**	21.4	26.0	76.2	46.8	42.6
PrefT*	7.1M (0.110%)	14.2	24.4	63.4	38.1	35.0
Adapter^{S*}	63.6M (0.990%)	15.0	33.3	77.7	52.3	44.6
Adapter^{P*}	227.5M (3.540%)	18.1	35.3	82.4	49.6	46.4

Table 2. Math reasoning results on Llama 1 7b. We take all baseline results from (Hu et al., 2023).

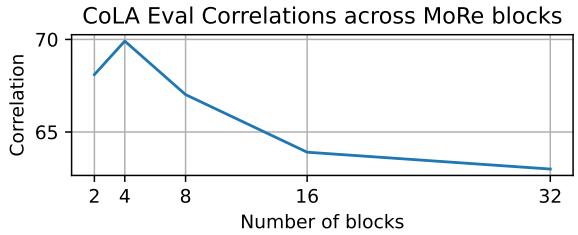

Figure 3. Fixing $r_{blk} = 4$, increasing the number of blocks beyond 4 does not lead to better performance.

3.1. Architectural Choices & Analysis

Inspired by work on search-based adaptation: AdaLoRA (Zhang et al., 2023) which adaptively allocates parameters for different ranks, and GLoRA (Chavan et al., 2023), which tunes the adapter complexity layer-wise, we explored different adapter styles (see Appendix C) as well as trading sparsity (N) and rank (r_{blk}) for the best investment of our parameter budget (Figure 2). Since merely changing N does not change the parameter count, we constrained each block to be square for **block number scaling**.

Interestingly, our search converged to a minimal 4-block architecture with ***the fewest tunable hyperparameters among all methods***, without the adapter scaler α in LoRA. Our search space trivially subsumes LoRA if we set N to 1. Empirically, MoRe with $N = 1$ and $r = r_{blk} = 8$ obtains 68.18 Matthew's Correlation on CoLA, aligning with the 68.3 for rank 8 LoRA.

Should we tune the number of blocks? Due to our rectangular block-diagonal parametrization, increasing N while fixing r_{blk} increases the total rank r under the same parameter budget. However, this induces worse performance, possibly because the matrix is sparser and it is harder to converge to a stable subspace. Empirically, performance drops drastically when $N > 4$ (Figure 3).

Relationship To BOFT. BOFT (Liu et al., 2024b) uses butterfly matrices (Dao et al., 2019; 2020), a related class of structured matrices. Monarch (Dao et al., 2022a) was proposed to replace butterfly matrices due to their hardware-unfriendly sparsity patterns. While it has $O(n \log n)$ FLOPs, it is empirically 2x slower than LoRA and occupies much more memory, which we show in the following.

Theoretical Analysis. One advantage of MoRe is that it is amenable to a theoretical analysis of its expressivity, mirroring that of LoRA (Zeng & Lee, 2024). We show in Appendix A that MoRe is more expressive than LoRA.

PEFT	#Params	MNLI	SST-2	MRPC	CoLA	QNLI	QQP	RTE	STS-B	Avg.
LoRA$_{r=8}$	0.79M	90.2	96.0	89.8	65.5	94.7	90.7	86.3	91.7	88.16
MoRe$_{r=32}$ (ours)	0.56M	**90.77**	**96.36**	**90.93**	**68.69**	94.78	91.00	85.92	92.08	**88.8**
MoRe$_{r=4}$ (ours)	**0.14M**	89.69	96.18	89.71	67.54	**94.85**	90.41	85.08	91.77	88.15
ReFT	**0.048M**	89.2	96.2	90.1	68.0	94.1	88.5	**87.5**	91.6	88.2
BOFT$^{m=4}_{b=4}$	1.266M	89.45	95.8	90.21	64.79	94.31	88.37	85.92	**92.26**	87.64
Adapter*	0.89M	90.1	95.2	90.5	65.4	94.6	**91.4**	85.3	91.5	88.0
Adapter^{FFN}	0.79M	90.3	96.1	90.5	64.4	94.3	91.3	84.8	90.2	87.7
RED	0.048M	89.5	96.0	90.3	68.1	93.5	88.8	86.2	91.3	88.0

Table 3. Language understanding comparisons. We take LoRA, Adapter, ReFT and RED results from Wu et al. (2024). All runs are averaged over 3 random seeds. We report Pearson Correlation for STS-B, Matthew's correlation for CoLA, matched accuracy for MNLI, and accuracy for other tasks. We report different parameter counts for BOFT from the original paper—while it is claimed that due to skew-symmetricity, only half of the matrix parameters are needed, then negated and copied to the other half, in practice the whole matrix requires gradients.

4. Experimental Results

We conducted experiments on three challenging NLP tasks covering over 20 datasets: commonsense reasoning, math reasoning, and language understanding of models ranging from Roberta-large to Llama 7B. We follow the widely adopted dataset settings in LLM-Adapters (Hu et al., 2023) and ReFT (Wu et al., 2024). All experiments are performed on a single NVIDIA A100 40G, and use Flash Attention (Dao et al., 2022b) when applicable. As we shall see, besides fixing N, MoRe needs **almost no tuning for rank** r_{blk}.

Commonsense Reasoning. We train the Llama 1 7b model (Touvron et al., 2023) on the challenging Commonsense170k benchmark in (Hu et al., 2023) consisting of eight commonsense reasoning tasks. The model is prompted with multiple-choice problems to output the correct choice without step-wise reasoning. We report accuracy on the test set in table 1, and hyperparameter details can be found in B. **Note that MoRe with Llama 7B largely surpasses the state-of-the-art ReFT with Llama 13B with around** $1/6$ **of its training steps (3 epochs).**

Math Reasoning. We train Llama 1 on the Math 10k dataset consisting of seven complex math reasoning tasks from Hu et al. (2023). Following Wu et al. (2024), we only used 4 datasets for final evaluation to avoid data leakage. The results are shown in Table 2.

Natural Language Understanding. We evaluate MoRe on the GLUE benchmark (Wang et al., 2018) to show its superior parameter efficiency on small LLMs. We fine-tune RoBERTa-large 350M (Liu et al., 2019b) on eight datasets consisting of tasks such as sentiment classification and natural language inference, and report performance on the evaluation set following Hu et al. (2021) over 3 different random seeds. Classifier heads are excluded from the parameter count. We use fp32 for all GLUE tasks, and hyperparameter tuning is done for each task separately (Appendix B). By default, we adapt query, key, and values. Notably, MoRe is on par with LoRA even with $r_{blk} = 1$ and 0.14M parameters, and outperforms all other methods when $r_{blk} = 4$.

Model	PEFT	Task	Peak Memory	Runtime
RoBERTa-large	BOFT$^{m=4}_{b=4}$	CoLA	5.98 GB	29.9 min
RoBERTa-large	LoRA$_{r=8}$	CoLA	4.3 GB	14.7 min
RoBERTa-large	MoRe$_{r=32}$	CoLA	5.68 GB	15.5 min
Llama 7b	BOFT$^{m=4}_{b=4}$; q, k, v	Math	53.97 GB	10 hr
Llama 7b	BOFT$^{m=4}_{b=4}$	Math	OOM	OOM
Llama 7b	LoRA$_{r=32}$	Math	24.86 GB	4.83 hr
Llama 7b	MoRe$_{r=32}$	Math	**22.09 GB**	**4.55 hr**

Table 4. Comparison of peak memory and runtime. We use a batch size of 2 for Llama 7B and 32 for RoBERTa. Due to the prohibitive memory cost of BOFT, we use H100 to benchmark Llama and only adapted query, key, and value for BOFT.

Memory Cost and Runtime. Modern GPUs rely on the tensor cores to accelerate matrix multiplication. MoRe leverages optimized CUDA batched matrix multiplication (BMM) kernels to populate the tensor core with many small matrices, with on par or better performance than GEMM. Here we show how our training speed compares with BOFT[1] and LoRA in Table 4, using the setting in our training experiments (see Appendix B). For Llama, we apply bf16, flash attention, and adapt all linear modules by default. Notably, ***BOFT runs out of memory even on H100 80G***, rendering it impractical for large models. MoRe slightly lags behind LoRA for the 350M RoBERTa due to the overhead of permutations allocating extra memory and multiple CUDA kernel launches, which we will address in a future Triton implementation, but excels in larger models due to its parameter efficiency.

[1]BOFT's public implementation does not support bf16 and fp16, so we added these features.

5. Conclusion

We introduced MoRe, a framework for searching for high-quality adapter architectures via Monarch matrices. MoRe offers excellent performance and has multiple promising directions for future work (described in Appendix F).

References

Bommasani, R., Hudson, D. A., Adeli, E., Altman, R., Arora, S., von Arx, S., Bernstein, M. S., Bohg, J., Bosselut, A., Brunskill, E., Brynjolfsson, E., Buch, S., Card, D., Castellon, R., Chatterji, N. S., Chen, A. S., Creel, K. A., Davis, J., Demszky, D., Donahue, C., Doumbouya, M., Durmus, E., Ermon, S., Etchemendy, J., Ethayarajh, K., Fei-Fei, L., Finn, C., Gale, T., Gillespie, L., Goel, K., Goodman, N. D., Grossman, S., Guha, N., Hashimoto, T., Henderson, P., Hewitt, J., Ho, D. E., Hong, J., Hsu, K., Huang, J., Icard, T. F., Jain, S., Jurafsky, D., Kalluri, P., Karamcheti, S., Keeling, G., Khani, F., Khattab, O., Koh, P. W., Krass, M. S., Krishna, R., Kuditipudi, R., Kumar, A., Ladhak, F., Lee, M., Lee, T., Leskovec, J., Levent, I., Li, X. L., Li, X., Ma, T., Malik, A., Manning, C. D., Mirchandani, S., Mitchell, E., Munyikwa, Z., Nair, S., Narayan, A., Narayanan, D., Newman, B., Nie, A., Niebles, J. C., Nilforoshan, H., Nyarko, J. F., Ogut, G., Orr, L. J., Papadimitriou, I., Park, J. S., Piech, C., Portelance, E., Potts, C., Raghunathan, A., Reich, R., Ren, H., Rong, F., Roohani, Y. H., Ruiz, C., Ryan, J., R'e, C., Sadigh, D., Sagawa, S., Santhanam, K., Shih, A., Srinivasan, K. P., Tamkin, A., Taori, R., Thomas, A. W., Tramèr, F., Wang, R. E., Wang, W., Wu, B., Wu, J., Wu, Y., Xie, S. M., Yasunaga, M., You, J., Zaharia, M. A., Zhang, M., Zhang, T., Zhang, X., Zhang, Y., Zheng, L., Zhou, K., and Liang, P. On the opportunities and risks of foundation models. *ArXiv*, abs/2108.07258, 2021. URL https://api.semanticscholar.org/CorpusID:237091588.

Chavan, A., Liu, Z., Gupta, D., Xing, E., and Shen, Z. One-for-all: Generalized lora for parameter-efficient fine-tuning, 2023.

Dao, T., Gu, A., Eichhorn, M., Rudra, A., and Ré, C. Learning fast algorithms for linear transforms using butterfly factorizations. *Proceedings of machine learning research*, 97:1517–1527, 2019. URL https://api.semanticscholar.org/CorpusID:76661331.

Dao, T., Sohoni, N. S., Gu, A., Eichhorn, M., Blonder, A., Leszczynski, M., Rudra, A., and Ré, C. Kaleidoscope: An efficient, learnable representation for all structured linear maps. *arXiv preprint arXiv:2012.14966*, 2020.

Dao, T., Chen, B., Sohoni, N. S., Desai, A., Poli, M., Grogan, J., Liu, A., Rao, A., Rudra, A., and Ré, C. Monarch: Expressive structured matrices for efficient and accurate training. In *International Conference on Machine Learning (ICML)*, 2022a.

Dao, T., Fu, D., Ermon, S., Rudra, A., and Ré, C. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems*, 35:16344–16359, 2022b.

Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., and Chen, W. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

Hu, Z., Wang, L., Lan, Y., Xu, W., Lim, E.-P., Bing, L., Xu, X., Poria, S., and Lee, R. K.-W. Llm-adapters: An adapter family for parameter-efficient fine-tuning of large language models. *arXiv preprint arXiv:2304.01933*, 2023.

Li, L. and Talwalkar, A. Random search and reproducibility for neural architecture search. In Adams, R. P. and Gogate, V. (eds.), *Proceedings of The 35th Uncertainty in Artificial Intelligence Conference*, volume 115 of *Proceedings of Machine Learning Research*, pp. 367–377. PMLR, 22–25 Jul 2020. URL https://proceedings.mlr.press/v115/li20c.html.

Li, L., Jamieson, K., Rostamizadeh, A., Gonina, E., Ben-Tzur, J., Hardt, M., Recht, B., and Talwalkar, A. A system for massively parallel hyperparameter tuning. *Proceedings of Machine Learning and Systems*, 2:230–246, 2020.

Li, L., Khodak, M., Balcan, N., and Talwalkar, A. Geometry-aware gradient algorithms for neural architecture search. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=MuSYkd1hxRP.

Liu, H., Simonyan, K., and Yang, Y. DARTS: Differentiable architecture search. In *International Conference on Learning Representations*, 2019a. URL https://openreview.net/forum?id=S1eYHoC5FX.

Liu, S.-Y., Wang, C.-Y., Yin, H., Molchanov, P., Wang, Y.-C. F., Cheng, K.-T., and Chen, M.-H. Dora: Weight-decomposed low-rank adaptation, 2024a.

Liu, W., Qiu, Z., Feng, Y., Xiu, Y., Xue, Y., Yu, L., Feng, H., Liu, Z., Heo, J., Peng, S., Wen, Y., Black, M. J., Weller, A., and Schölkopf, B. Parameter-efficient orthogonal finetuning via butterfly factorization. In *The Twelfth International Conference on Learning Representations*, 2024b. URL https://openreview.net/forum?id=7NzgkEdGyr.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. Roberta: A robustly optimized bert pretraining approach.

Cornell University - arXiv,Cornell University - arXiv, Jul 2019b.

Meng, F., Wang, Z., and Zhang, M. Pissa: Principal singular values and singular vectors adaptation of large language models. *arXiv preprint arXiv:2404.02948*, 2024.

Pham, H., Guan, M., Zoph, B., Le, Q., and Dean, J. Efficient neural architecture search via parameters sharing. In Dy, J. and Krause, A. (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 4095–4104. PMLR, 10–15 Jul 2018. URL https://proceedings.mlr.press/v80/pham18a.html.

Tillet, P., Kung, H.-T., and Cox, D. Triton: an intermediate language and compiler for tiled neural network computations. In *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, pp. 10–19, 2019.

Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., Rodriguez, A., Joulin, A., Grave, E., and Lample, G. Llama: Open and efficient foundation language models, 2023.

Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., and Bowman, S. Glue: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, Jan 2018. doi: 10.18653/v1/w18-5446. URL http://dx.doi.org/10.18653/v1/w18-5446.

Wu, Z., Arora, A., Wang, Z., Geiger, A., Jurafsky, D., Manning, C. D., and Potts, C. Reft: Representation finetuning for language models. *arXiv preprint arXiv:2404.03592*, 2024.

Zeng, Y. and Lee, K. The expressive power of low-rank adaptation. In *International Conference on Learning Representations (ICLR)*, 2024.

Zhang, Q., Chen, M., Bukharin, A., He, P., Cheng, Y., Chen, W., and Zhao, T. Adaptive budget allocation for parameter-efficient fine-tuning. In *International Conference on Learning Representations*. Openreview, 2023.

Appendix

A. Theoretical results

We show a theoretical finding for the expressiveness of MoRe in the spirit of Zeng & Lee (2024).

We start with a simple result.

Lemma A.1. *Let W be an $n \times n$ matrix, where $n = m^2$ for some integer m. Let W_{jk} denote the submatrix of W such that*

$$W = \begin{bmatrix} W_{11} & W_{12} & \dots & W_{1m} \\ W_{21} & W_{22} & \dots & W_{2m} \\ \vdots & \vdots & & \vdots \\ W_{n1} & W_{m2} & \dots & W_{mm} \end{bmatrix}$$

Let $x \in \mathbb{R}^n$, with a similar decomposition into x_k for $k = 1, 2, \dots, m$. Then $\|Wx\|_2 \leq \sum_{jk} \|W_{jk} x_k\|_2$.

Proof. We have that

$$\begin{aligned} \|Wx\|_2 &= \| \begin{bmatrix} W_{11} & \dots & W_{1m} \\ \vdots & \ddots & \vdots \\ W_{n1} & \dots & W_{mm} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \|_2 \\ &= \| \begin{bmatrix} W_{11}x_1 + \ldots + W_{1m}x_m \\ \vdots \\ W_{m1}x_1 + \ldots + W_{mm}x_m \end{bmatrix} \|_2 \\ &\leq \sum_j \|W_{j1}x_1 + \ldots + W_{jm}x_m\|_2 \\ &\leq \sum_{jk} \|W_{jk}x_k\|_2 \end{aligned}$$

□

Corollary A.2. *Let W be an $n \times n$ matrix, where $n = m^2$ for some integer m. Let W_{jk} denote the submatrices of W. Then $\sigma_1(W) \leq \sum_{jk} \sigma_1(W_{jk})$.*

Proof. Note that $\sigma_1(M) = \|M\|_2$ for any matrix M. Let $x \in \mathbb{R}^n$ such that $\|x\|_2 = 1$ and $\|Wx\|_2 = \|W\|_2$. Using Lemma A.1,

$$\|W\|_2 = \|Wx\|_2 \leq \sum_{jk} \|W_{jk}x_k\|_2$$

Since $\|W_{jk}x_k\|_2 \leq \|W_{jk}\|_2 \cdot \|x_k\|_2 \leq \|W_{jk}\|_2$, we have

$$\|W\|_2 \leq \sum_{jk} \|W_{jk}x_k\|_2 \leq \sum_{jk} \|W_{jk}\|_2$$

□

Theorem A.3. *Suppose both the target and frozen model are linear and respectively parameterized by $\bar{W}$ and $W = \prod_{l=1}^{L} W_l$ with both $\bar{W}$ and W full rank. Assume that $r = N$ for these Monarch matrices, i.e. the Monarch factors are square with square blocks on the diagonal. The adapted model is allowed to fine-tune the last layer's parameters with a Monarch matrix: $\hat{W} = \prod_{l=1}^{L-1} W_l(W_L + \Delta_{W_L})$, where $\Delta_{W_L} \in \mathcal{M}$. Define error between the target and the frozen model as $E = \bar{W} - W$, and regularized error as $\tilde{E} = (\prod_{l=1}^{L-1} W_l)^{-1} E$. The estimation error between the adapted model and the target model is bounded:*

$$\begin{aligned} \|\bar{W} - \hat{W}\|_F^2 &\leq \| \prod_{l=1}^{L-1} W_l\|_F^2 \cdot \|\tilde{E} - \Delta_{W_L}\|_F^2 \\ &= \| \prod_{l=1}^{L-1} W_l\|_F^2 \cdot (\sum_{jk} (\sum_{i=2} \sigma_i^2(\tilde{E}_{:,j,k,:}))). \end{aligned}$$

where σ_i is the i-th eigenvalue of the given function and $\tilde{E}_{ijkl}$ is $\tilde{E}$ reshaped into a 4-D tensor.

Proof. The proof directly follows the decomposition in the Monarch paper (Dao et al., 2022a) and the previously derived results. □

We now use a worst-case to illustrate how the Monarch approximation differs from a rank-1 approximation. Let A be any matrix of size $n \times n$. Reshape A into a 4D tensor $\tilde{A}$ of dimension $m \times m \times m \times m$, where $m = \sqrt{n}$. Then in the worst case, each sub-matrix is of full-rank m and the singular values are all equal. An optimal monarch matrix M in Frobeneus norm performs a rank-1 approximation for each sub-matrix. The estimation error $\|A - M\|_F^2$ can be interpreted as all unexplained singular values, whose proportion is $\frac{m-1}{m}$. Hence $\|A - M\|_F^2 = \frac{m-1}{m}\|A\|_F^2$. This provides a bound when in a general case.

Now consider a rank-1 approximation of A. In the worst case, since A's rank cannot be smaller than m (a full matrix's rank is always equal or greater than the rank of its submatrix), let A be of rank m. Suppose A's non-zero singular values are still all equal (?). The estimation error for a rank-1 approximation D of A will be $\|A - D\|_F^2 = \frac{m-1}{m}\|A\|_F^2$, which equals the Monarch approximation. However, in other cases where A's rank is greater than m, a Monarch approximation is strictly better than a rank-one approximation.

A.1. Optimizations for Rectangular Monarch matrices

There are two distinct cases with variable-rank Monarch matrices. Each case depends on how the block rank compares to the block number.

Let n be the dimensions of M, the Monarch product, let N be the number of blocks in each factor L and R, let $m =$

n/N be the block width, and let r be the block rank. In total, we have that M, L, and R are of dimension (n, n), (n, r), and (r, n) respectively. When compressed into 4- and 3-tensors, these have dimension (m, N, N, m), (N, r, m), and (N, m, r) respectively. To investigate the behavior of $M = P_1 L P_2 R$, consider a vector $x \in \mathbb{R}^n$ and how M transforms this vector. Reshape x into a 2-tensor with dimensions (N, m).

First, assume $N \geq r$ where r divides N and let $b = N/r$. For this case, further reshape M, L, R, and x into shapes (m, r, b, r, b, m), (r, b, r, m), (r, b, m, r), and (r, b, m), which is possible since $rb = N$.

- Apply R: This results in the intermediate $y_{kbj} = \sum_i R_{kbji} x_{kbi}$.
- Apply P_2: This transposes the first and third coordinates of y, so $y_{kbj} \to y_{jbk}$.
- Apply L: This results in the intermediate $z_{jbl} = \sum_k L_{jblk} y_{jbk}$.
- Apply P_1: This again transposes the first and third coordinates of z, so $z_{jbl} \to z_{lbj}$.

In total, this amounts to computing $z_{lbj} = \sum_{k,i} L_{jblk} R_{kbji} x_{kbi}$. We then can define $M_{ljbkbi} = L_{jblk} R_{kbji}$ which defines the operation $z_{lbj} = \sum_{k,i} M_{ljbkbi} x_{kbi}$. Notice that the optimal solution can be found through a collection of rank-1 decompositions of $M_{:jbkb:}$, each of size (m, m). This common index b implies that whenever those coordinates in the 6-tensor disagree, this Monarch product contains zeros, so this decomposition will be sparse. Next, assume that $N < r$ where N divides r and let $b = r/N$. For this case, further reshape L and R into shapes (N, N, b, m) and (N, m, N, b), which is possible since $Nb = r$.

- Apply R: This results in the intermediate $y_{kjb} = \sum_i R_{kjbi} x_{ki}$.
- Apply P_2: This transposes the first and second coordinates of y, so $y_{kjb} \to y_{jkb}$.
- Apply L: This results in the intermediate $z_{lj} = \sum_{k,b} L_{jlkb} y_{jkb}$.
- Apply P_1: This again transposes the first and second coordinates of z, so $z_{lj} \to z_{jl}$.

In total, this amounts to computing $z_{jl} = \sum_{k,i,b} L_{jlkb} R_{kjbi} x_{ki}$. We then can define $M_{ljki} = \sum_b L_{jlkb} R_{kjbi}$ which defines the operation $z_{lbj} = \sum_{k,i} M_{ljki} x_{ki}$. Notice that the optimal solution can

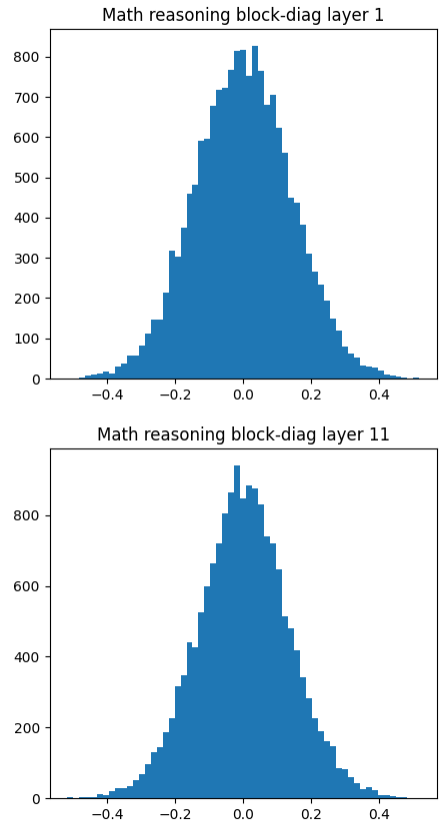

Figure 4. Llama 7b trained on Math Reasoning tasks

be found through a collection of rank-b decompositions of $M_{:jk:}$, each of size (m, m).

Using these decompositions, we obtain some straightforward extensions of Theorem A.3.

Theorem A.4. *Suppose both the target and frozen model are linear and respectively parameterized by $\bar{W}$ and $W = \prod_{l=1}^{L} W_l$ with both $\bar{W}$ and W full rank. Assume that $N < r$ with r a multiple of N. The adapted model is allowed to fine-tune the last layer's parameters with a Monarch matrix: $\hat{W} = \prod_{l=1}^{L-1} W_l(W_L + \Delta_{W_L})$, where $\Delta_{W_L} \in \mathcal{M}$. Define error between the target and the frozen model as $E = \bar{W} - W$, and regularized error as $\tilde{E} = (\prod_{l=1}^{L-1} W_l)^{-1} E$. The estimation error between the adapted model and the target model is bounded:*

$$\|\bar{W} - \hat{W}\|_F^2 \leq \| \prod_{l=1}^{L-1} W_l\|_F^2 \cdot \|\tilde{E} - \Delta_{W_L}\|_F^2$$
$$= \| \prod_{l=1}^{L-1} W_l\|_F^2 \cdot (\sum_{jk}(\sum_{i=r/N+1} \sigma_i^2(\tilde{E}_{:,j,k,:}))).$$

where σ_i is the i-th eigenvalue of the given function and $\tilde{E}_{ijkl}$ is $\tilde{E}$ reshaped into a 4-D tensor.

Notice the difference in the rightmost sum. The sum over i starts at $r/N + 1$ instead of 2.

Next, we provide experimental details.

B. Hyperparameter Tuning

We use the asynchronous successive halving algorithm (ASHA) (Li et al., 2020) to efficiently search and early-stop on our 8 * A100 cluster.

B.1. GLUE Language Understanding

For BOFT, we took the hyperparameters for DeBERTA-v3 base on GLUE from their paper and tuned the learning rate only. For MoRe, we started from the hyperparameters in (Hu et al., 2021) and randomly sampled the learning rate and batch size. We present the hyperparameters in table 5.

Hyperparameter	MNLI	SST-2	MRPC	CoLA	QNLI	QQP	RTE	STS-B
learning rate	2e-4	4.4e-4	3.2e-4	2.1e-4	3e-4	6.2e-4	5.4e-4	6.4e-4
batch size	32	32	16	32	32	16	32	32
weight decay				1e-3				
lr scheduler				cosine				
Epochs	10	10	20	20	20	10	20	20

Table 5. GLUE hyperparameters

B.2. Math reasoning and Commonsense reasoning

For these challenging reasoning tasks, we found performance to be less sensitive to hyperparameters. We took 1000 examples and 10,000 examples from Math10K and Commonsense170K as the tuning evaluation set, respectively. We present the hyperparameters in table 6.

Hyperparameter	Math reasoning	Commonsense reasoning
learning rate	3e-4	4e-4
batch size(w/ gradient accumulation)	64	16
weight decay	0	
lr scheduler	cosine	
dropout	0.1	
Epochs	12	3

Table 6. Reasoning hyperparameters

C. Architecture Ablations

With 3 potential architectural hyperparameters (r_{blk}, N and whether to use square blocks) in our setup, one might ask whether we should use NAS to find the most efficient architecture.

We tested using monarch as a multiplicative factor instead of an additive factor as in BOFT, adding a scaler α on the adapter outputs as in LoRA and adding a scaler parameter; all underperform our default 4-block configuration. We also tried including r_{blk} and N in our hyperparameter search to mimic NAS, but **all runs converged to the configuration with the largest parameter count**, with marginal performance gains. Therefore we didn't pursue expensive NAS algorithms.

Method	GLUE CoLA
MoRe (learnable scaler)	41.1
MoRe ($\alpha = 2$)	0
MoRe (multiplicative factor)	0

D. Learned Weight Distributions

We demonstrate in figure 4 and 5 that the trained block-diagonal matrices approximate Gaussian distribution well as the amount of training increases, in an attempt to interpret the results.

E. Failure Cases

Inspired by Meng et al. (2024) that fine-tuning is strengthening some task-specific subspace components, we attempted using the dense to sparse projection algorithm (block-wise SVD) from (Dao et al., 2022a) to initialize MoRe from principal components. However, the method fails to converge on reasoning tasks and obtains only a 57.9 correlation on CoLA.

We've also tested naively replacing the low-rank projections in ReFT with a single Monarch factor P plus permutation P_1, which only achieved a 19.5 correlation on CoLA.

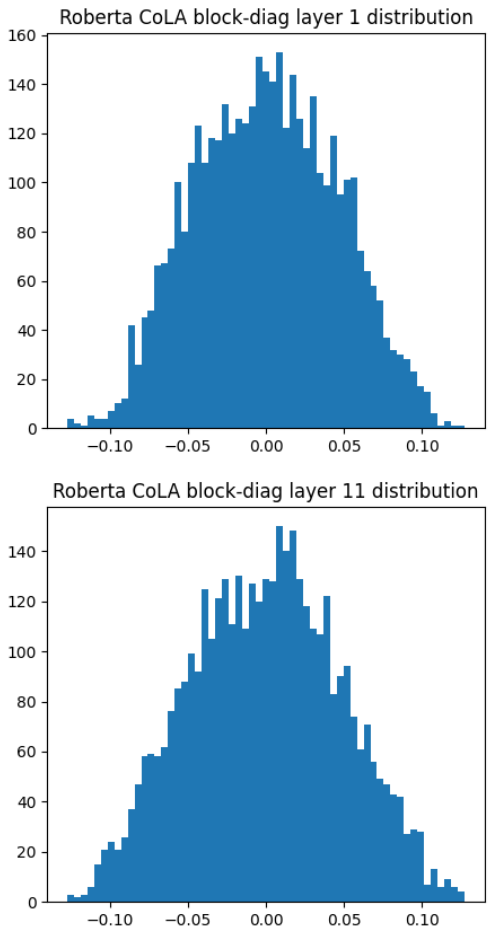

Figure 5. RoBERTa-large trained on CoLA

F. Limitations and Future Work

Currently, MoRe poses a few limitations that we are working to address.

1. MoRe is implemented with two BMMs and two permutations, which introduces overhead due to 4 CUDA kernel launches. With machine learning compilers such as Triton(Tillet et al., 2019), it's easy to fuse them into one kernel and recompute the activations during backward, with memory savings and speed-up. We're testing the Triton implementation's precision.

2. We seek to substitute low-rank projections. A natural extension from our low-rank adaptation use case is to establish MoRe as a general drop-in low-rank projection module. However as shown in the Appendix E, it does not work directly with ReFT.

3. Projection subspace interpretation: we show (Appendix D) that Monarch weights approach Gaussian distribution. However, we've not explored the subspace similarity between the dense and MoRe projections such as which dense components are strengthened by MoRe, due to complicated block-diagonality. Such an understanding may enable us to initialize MoRe from dense matrices' principal components as in Meng et al. (2024) with improved convergence and performance, and explain why scaling rank doesn't always deliver performance.

G. Pseudocode

As the permutations P_1 and P_2 may be less intuitive, we provide a minimal PyTorch pseudocode to demonstrate their usage below.

```
# Input:
#   x: (bs, n)
#   blkdiag1: (nblocks, block rank, block size)
#   blkdiag2: (nblocks, block size, block rank)
batch_shape, n = x.shape[:-1], x.shape[-1]
nblocks, blk_r, blk_sz = blkdiag1.shape
batch_dim = torch.prod(batch_shape)
x = x.reshape(bs, nblocks, blk_sz)
out1 = torch.empty(batch_dim, nblocks, blk_r).transpose(0, 1)
# (nblocks, batch_dim, blk_sz) @ (nblocks, blk_sz, blk_r) -> (nblocks, batch_dim, blk_r)
out1 = torch.bmm(x, blkdiag1.transpose(-1, -2), out=out1)
out1 = out1.transpose(0, 1).reshape(batch_dim, blk_r, nblocks)
out1 = out1.transpose(-1, -2).contiguous().transpose(0, 1)
out2 = torch.empty(nblocks, batch_dim, blk_sz)
# (nblocks, batch_dim, blk_r) @ (nblocks, blk_r, blk_sz) -> (nblocks, batch_dim, blk_sz)
out2 = torch.bmm(out1, blkdiag2.transpose(-1, -2), out=out2)
out2 = out2.permute(1, 2, 0).reshape(*batch_shape, blk_sz * nblocks)
```