KronA: Parameter Efficient Tuning with Kronecker Adapter

Ali Edalati
McGill University
ali.edalati@mail.mcgill.ca

Marzieh Tahaei
Huawei Noah's Ark Lab
marzieh.tahaei@huawei.com

Ivan Kobyzev
Huawei Noah's Ark Lab
ivan.kobyzev@huawei.com

Vahid Partovi Nia
Huawei Noah's Ark Lab
vahid.partovinia@huawei.com

James J. Clark
McGill University
james.clark1@mcgill.ca

Mehdi Rezagholizadeh
Huawei Noah's Ark Lab
mehdi.rezagholizadeh@huawei.com

Abstract

Fine-tuning a Pre-trained Language Model (PLM) on a specific downstream task has been a well-known paradigm in natural language processing. However, with the growing size of PLMs, training the entire model on downstream tasks has become significantly time-consuming and resource-hungry. Therefore, Parameter Efficient Tuning (PET) techniques have been proposed to address the growing demand for the efficient fine-tuning of PLMs. One popular PET technique is inserting trainable adapters into a frozen model during fine-tuning. However, adapters have low-rank projections, which may reduce their representation power, resulting in sub-optimal performance. We address this problem using the Kronecker product instead of low-rank multiplications to improve the flexibility and performance of adapters. We introduce KronA, a Kronecker equivalent of LoRA for efficient fine-tuning of transformer-based PLMs. We apply the proposed adapters for fine-tuning a well-known PLM, called T5, on the GLUE benchmark to show that our method outperforms the popular PET baselines.

1 Introduction

Large Pre-trained Language Models (PLMs) are used as a backbone in various natural language processing tasks to achieve state-of-the-art results (Devlin et al., 2019; Radford et al., 2019). PLMs are adapted to downstream applications either via in-context learning or fine-tuning. In-context learning imposes substantial memory and computational overhead during inference since all training examples should be processed for each sample (Liu et al., 2022). On the other hand, fine-tuning provides less inference latency and improved accuracy. However, as PLMs become larger, fine-tuning the entire model becomes challenging since more time and computation power is required. Additionally, one must store a complete model checkpoint for each downstream application, making the deployment inefficient.

The Third Workshop on Efficient Natural Language and Speech Processing (ENLSP-III)

To address these challenges, several works have proposed inserting a few trainable parameters while freezing most (or even all) of the pre-trained model parameters. This significantly reduces the memory and computation requirements for fine-tuning. Furthermore, instead of storing one copy of the entire model, a small set of tuned parameters can be stored for each task. We refer to these methods as Parameter Efficient Tuning (PET) methods.

Among the PET methods, soft prompts (Li and Liang, 2021; Lester et al., 2021) prepend trainable parameters to the input of the layers and train them on downstream tasks. However, the increase in the length of the embedding layers leads to a significant computation overhead during inference.

In another category of PET methods, trainable adapters are inserted (Houlsby et al., 2019; Mahabadi et al., 2021; He et al., 2022) into frozen transformers (Vaswani et al., 2017). Adapters are low-rank modules that are composed of an up projection followed by a down projection. One limitation of these approaches is that they increase the computational overhead and the latency during the inference which makes them inefficient for latency-critical scenarios.

Therefore, Low-Rank Adaption (LoRA) (Hu et al., 2022) was developed using extra low-rank adapters in parallel to pre-trained weight matrices. However, once fine-tuned, the adapter parameters can be merged with the original pre-trained weights, making the latency and energy requirements for inference, intact. Despite fast inference and similar to other low-rank adapters, LoRA suffers from a loss of accuracy compared to full fine-tuning. This is due to the strong assumption imposed by its low-rank structure for task-specific updates.

Kronecker-based decomposition is another factorization method that does not rely on the low-rank assumption. When used for model compression, this powerful decomposition method has proven to outperform low-rank compression methods (Thakker et al., 2019; Hameed et al., 2022). It has also been used successfully to compress transformer-based language models (Tahaei et al., 2022; Edalati et al., 2022).

Inspired by the success of Kronecker decomposition, we replace the low-rank projections of LoRA with the Kronecker product to develop Kronecker Adapter (KronA). This simple modification can improve the accuracy without increasing the inference latency. Also, for applications where the latency increase is tolerable, we propose to use KronA$^{\text{B}}$. This module is a version of KronA developed to be utilized in parallel to Feed-Forward Network (FFN) blocks and achieves notable improvements over full fine-tuning on the General Language Understanding Evaluation (GLUE) benchmark (Wang et al., 2018). In addition, when a proposed learnable residual connection is added to KronA$^{\text{B}}$, KronA$^{\text{B}}_{\text{res}}$ is developed to achieve even better results.

We evaluated our methods on the GLUE benchmark to study the impact of the Kronecker product on the performance. To summarize, our contributions are:

- Proposing KronA, a Kronecker Adapter that can be inserted in parallel to the weight matrices and is suitable for latency-critical scenarios.
- Using KronA in parallel to FFNs (KronA$^{\text{B}}$) along with a learnable residual connection (KronA$^{\text{B}}_{\text{res}}$) to further improve the accuracy at the cost of a higher inference latency.
- Providing evaluation of the proposed methods in comparison to the well-known baselines in terms of the GLUE score, training time, and inference latency.

2 Related work

(Ben Zaken et al., 2022) proposed freezing the weights and tuning only biases or a subset of biases of PLMs for fine-tuning on downstream tasks. This technique, called BitFit, is parameter-efficient and fast, but it usually cannot perform well compared to state-of-the-art methods.

(Houlsby et al., 2019) introduced adapters as a PET method, where all parameters of a PLM are frozen and some trainable modules are inserted after the FFN or attention blocks. Proposed adapters in (Houlsby et al., 2019) have a down projection, a non-linear function, an up projection, and a residual connection. We use "Adapter" to refer to these modules. (He et al., 2022) developed another version of adapters called Parallel-Adapter (PA). PAs are inserted parallel to the pre-trained blocks and have a scaling factor (Figure 1.c). (He et al., 2022) also introduced a unified view on PET methods and combined some techniques such as prefix tuning (Li and Liang, 2021) and PA.

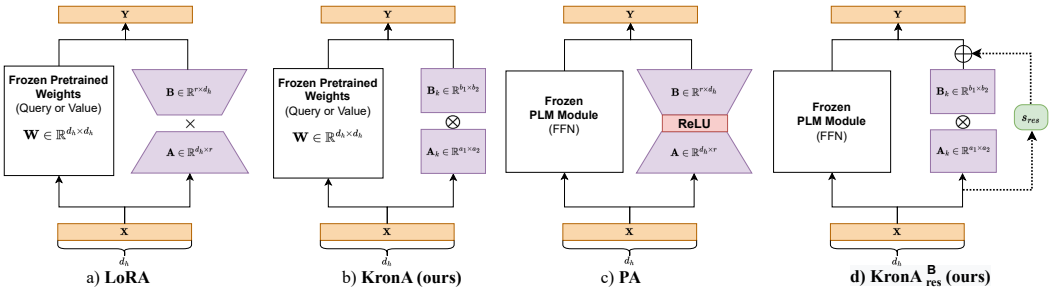

Figure 1: This figure shows the structure of the proposed Kronecker-based adapters and their low-rank counterparts. For simplicity, the scaling factor at the output of the modules is not depicted. Figure d shows $\text{KronA}^{\text{B}}_{\text{res}}$. The residual connection is depicted by a dotted-line to remind that this connection can simply be removed to have KronA^{B}.

In (Mahabadi et al., 2021), a modified version of the adapters is used for PET, named Compacter. In Compacter, the Kronecker product of multiple pairs of Kronecker factors is summed to reconstruct the module's weight matrix ($\mathbf{W}_{\text{Compacter}} = \sum_{i=1}^{n} \mathbf{A}_i \otimes \mathbf{B}_i$). To further reduce the trainable parameters, $\mathbf{A}_i$ matrices are shared across all layers and $\mathbf{B}_i$ matrices are decomposed as the matrix multiplication of two low-rank subfactors. Although Compacter achieves good results, it is notably slow in the training and inference phases.

(He et al., 2023) developed Kronecker-based adapters that have a similar structure to Compacter adapters for Vision Transformers (Dosovitskiy et al., 2021). However, (He et al., 2023) applied the adapters in parallel to weight matrices which enables merging the adapters into the model after training to avoid increasing the latency and parameters at the inference stage. Also, (Edalati et al., 2023) developed adapters applicable to both convolutional and linear layers that use summation of the Kronecker product of a sequence of factors for PET of computer vision models.

Proposed adapters in our work have a simpler structure compared to (Mahabadi et al., 2021), (He et al., 2023), and (Edalati et al., 2023) adapters by removing the parameter sharing, decomposition to low-rank subfactors, and summation to develop a faster method at the cost of lower parameter reduction.

(Hu et al., 2022) inserted adapters made from a downward projection and an upward projection parallel to the PLM weight matrices to introduce LoRA. During the training, the pre-trained weights are frozen, and only the LoRA adapters are tuned. During inference, the LoRA adapters are merged with the original weight matrices of PLMs. Therefore, unlike Adapter, PA, and Compacter, LoRA does not increase the inference time.

3 Methodology

3.1 Kronecker Product

Equation 1 shows how the elements of $\mathbf{A}$ are multiplied by $\mathbf{B}$ to generate the Kronecker product of $\mathbf{A} \in \mathbb{R}^{m_1 \times n_1}$ and $\mathbf{B} \in \mathbb{R}^{m_2 \times n_2}$ (Henderson et al., 1983).

$$\mathbf{W} = \mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} a_{1,1}\mathbf{B} & \cdots & a_{1,n_1}\mathbf{B} \\ \vdots & \ddots & \vdots \\ a_{m_1,1}\mathbf{B} & \cdots & a_{m_1,n_1}\mathbf{B} \end{bmatrix} \tag{1}$$

The Kronecker product has some interesting features that make it suitable for PET. First, unlike the low-rank down-projections used in other techniques, Kronecker-based decomposition maintains the rank of the input matrix. Second, to reduce the required FLOPS, Equation 2 can be used to obtain $(\mathbf{A} \otimes \mathbf{B})\mathbf{x}$ without the computation of $\mathbf{A} \otimes \mathbf{B}$, where $\eta_{m \times n}(\cdot)$ reshapes a vector into a matrix of size $m \times n$ and $\gamma(\cdot)$ flattens a matrix into a vector.

$$(\mathbf{A} \otimes \mathbf{B})\mathbf{x} = \gamma(\mathbf{B}\eta_{m \times n}(\mathbf{x})\mathbf{A}^{\top}) \tag{2}$$

Method	Params	CoLA	RTE	MRPC	SST-2	STS-B	MNLI	QNLI	QQP	Avg
Fine-tuning	100	63.37	74.82	92.73	93.58	90.07	86.16	92.77	91.74	85.65
BitFit	0.12	58.19	68.34	92.58	94.61	90.69	85.73	92.91	90.33	84.17
Adapter	0.07	64.66	71.94	91.27	**94.84**	90.49	85.91	92.97	90.35	85.30
LoRA	0.07	64.76	74.10	92.10	93.92	91.21	86.08	92.97	**90.68**	85.73
Compacter	0.07	64.42	76.26	91.52	93.92	91.04	86.14	92.93	90.36	85.82
PA	0.06	64.80	74.10	**93.20**	94.04	91.10	86.24	93.12	90.30	85.86
KronA	0.07	63.27	**77.70**	92.52	94.26	91.30	**86.34**	93.15	90.57	**86.14**
KronA$^{\text{B}}$	0.07*	65.74	75.54	92.78	94.72	**91.41**	86.22	93.19	**90.68**	**86.28**
KronA$^{\text{B}}_{\text{res}}$	0.07*	**66.73**	76.98	93.15	94.38	91.35	86.20	**93.21**	90.57	**86.57**

Table 1: This table shows the performance of the methods on GLUE. $*$ shows that the number of parameters might be slightly different depending on the choice of one or two biases in the module.

Method	Fine-tuning	LoRA	KronA	BitFit	Adapter	PA	Compacter	KronA$^{\text{B}}$	KronA$^{\text{B}}_{\text{res}}$
Inference Latency(%)	100	100	100	100	146	113	181	127	136
Training Time(%)	100	72	75	64	73	71	79	74	81

Table 2: The first row shows the normalized latency of the methods in the inference phase while the second row lists the average normalized training time of the methods on the GLUE tasks.

3.2 KronA

Figure 1.a shows the structure of a LoRA adapter where $\mathbf{A}$ and $\mathbf{B}$ are the projection weight matrices of the down and up projection, respectively. To modify this module into KronA, the Kronecker product replaces the matrix multiplication. Also, the LoRA projections are replaced by Kronecker factors (see Figure 1.b and Appendix A). Equation 3 shows how the output is generated when KronA is applied. $\mathbf{A}_k$ and $\mathbf{B}_k$ are the Kronecker factors that replaced the LoRA projections. Similar to LoRA, KronA has a fixed scaling factor, s, which is a hyperparameter.

$$\mathbf{Y} = \mathbf{X}\mathbf{W} + s\mathbf{X}[\mathbf{A}_k \otimes \mathbf{B}_k] \tag{3}$$

KronA modules are applied in parallel to the weight matrices of PLMs during the tuning phase. Once fine-tuned, the Kronecker factors are multiplied, then scaled and merged into the original weight matrices (Equation 4). Therefore, similar to LoRA, KronA does not increase the inference time.

$$\mathbf{W}_{\text{tuned}} = \mathbf{W} + s[\mathbf{A}_k \otimes \mathbf{B}_k] \tag{4}$$

3.3 KronA$^{\text{B}}$ and KronA$^{\text{B}}_{\text{res}}$

Inspired by the promising performance of PA, we also investigate KronA when used in parallel to the FNN blocks and call it KronA$^{\text{B}}$. The B superscript in the name means that this module is applied to the pre-trained blocks, as opposed to KronA which is applied to pre-trained weight matrices. Similar to PA, the non-linearity in the FFN blocks does not allow our proposed adapters to be merged into the pre-trained blocks after fine-tuning. This imposes an increase in the inference time and computations. Equation 5 shows how KronA$^{\text{B}}$ works in parallel to an FFN block.

$$\mathbf{Y} = \text{FFN}(\mathbf{X}) + s\mathbf{X}[\mathbf{A}_k \otimes \mathbf{B}_k] \tag{5}$$

To further improve the representation power, we incorporate a scaled residual connection inside the KronA$^{\text{B}}$ module to develop KronA$^{\text{B}}_{\text{res}}$. The scale of the residual connection (s_{res}) is initialized with one and tuned during the fine-tuning. Equation 6 shows how KronA$^{\text{B}}_{\text{res}}$ works in parallel to an FFN. Also, Figure 1.c and 1.d show the structure of a PA and our KronA$^{\text{B}}_{\text{res}}$ module, respectively.

$$\mathbf{Y} = \text{FFN}(\mathbf{X}) + s\mathbf{X}[\mathbf{A}_K \otimes \mathbf{B}_K] + s_{\text{res}}\mathbf{X} \tag{6}$$

4 Results and discussion

Table 1 shows[1] the GLUE score of our proposed methods compared to other baselines when applied to T5 (Raffel et al., 2020). As the results show, KronA and KronA$^{\text{B}}$ outperform LoRA and PA as their low-rank counterparts, respectively. Also, all of our proposed modules outperform other baselines on average and most of the GLUE tasks. Furthermore, KronA$^{\text{B}}_{\text{res}}$, which benefits from an extra learnable residual connection, achieves remarkably better results.

Table 2 shows the normalized inference delay for the discussed methods. KronA, LoRA, Fine-tuning, and Bitfit do not increase the inference latency since these techniques do not add additional parameters or computations to the model during the inference phase. Although KronA$^{\text{B}}$ is significantly faster than Compacter and Adapter, it is slower than PA, as we expected; computation of the Kronecker product is generally slower than the normal matrix multiplication. Also, adding the learnable residual connection increases the latency.

The normalized training time averaged over the GLUE tasks for each technique is shown in Table 2. Based on these results, the significant improvement in the accuracy of the proposed KronA and its variants is at the expense of a slight increase in the training time compared to the low-rank counterparts like LoRA, PA, and Adapter. However, the training time increase is not remarkable, and KronA modules are still significantly faster than fine-tuning.

5 Conclusion

In this work, we developed Kronecker-based adapters by replacing the low-rank projections from well-known PET methods with the Kronecker product. In addition to comparing the training and inference time, we evaluated our proposed adapter for fine-tuning T5 on the GLUE benchmark to show its superiority over popular baselines.

References

Agarap, A. F. (2018). Deep learning using rectified linear units (relu). arXiv preprint arXiv:1803.08375.

Ben Zaken, E., Goldberg, Y., and Ravfogel, S. (2022). BitFit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. In Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers), pages 1–9, Dublin, Ireland. Association for Computational Linguistics.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. In Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., and Houlsby, N. (2021). An image is worth 16x16 words: Transformers for image recognition at scale. In International Conference on Learning Representations.

Edalati, A., Abdel Hameed, M. G., and Mosleh, A. (2023). Generalized kronecker-based adapters for parameter-efficient fine-tuning of vision transformers. In 2023 20th Conference on Robots and Vision (CRV), pages 97–104.

Edalati, A., Tahaei, M., Rashid, A., Nia, V., Clark, J., and Rezagholizadeh, M. (2022). Kronecker decomposition for GPT compression. In Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers), pages 219–226, Dublin, Ireland. Association for Computational Linguistics.

Elfwing, S., Uchibe, E., and Doya, K. (2018a). Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. Neural Networks, 107:3–11.

[1]The experimental settings are explained in Appendix.

Elfwing, S., Uchibe, E., and Doya, K. (2018b). Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. Neural networks : the official journal of the International Neural Network Society, 107:3–11.

Hameed, M. G. A., Tahaei, M. S., Mosleh, A., and Nia, V. P. (2022). Convolutional neural network compression through generalized kronecker product decomposition. In Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2022, Thirty-Fourth Conference on Innovative Applications of Artificial Intelligence, IAAI 2022, The Twelveth Symposium on Educational Advances in Artificial Intelligence, EAAI 2022 Virtual Event, February 22 - March 1, 2022, pages 771–779. AAAI Press.

He, J., Zhou, C., Ma, X., Berg-Kirkpatrick, T., and Neubig, G. (2022). Towards a unified view of parameter-efficient transfer learning. In International Conference on Learning Representations.

He, X., Li, C., Zhang, P., Yang, J., and Wang, X. E. (2023). Parameter-efficient model adaptation for vision transformers. Proceedings of the AAAI Conference on Artificial Intelligence, 37(1):817–825.

Henderson, H. V., Pukelsheim, F., and Searle, S. R. (1983). On the history of the kronecker product. Linear & Multilinear Algebra, 14:113–120.

Hendrycks, D. and Gimpel, K. (2016). Gaussian error linear units (gelus). arXiv preprint arXiv:1606.08415.

Houlsby, N., Giurgiu, A., Jastrzebski, S., Morrone, B., De Laroussilhe, Q., Gesmundo, A., Attariyan, M., and Gelly, S. (2019). Parameter-efficient transfer learning for NLP. In Chaudhuri, K. and Salakhutdinov, R., editors, Proceedings of the 36th International Conference on Machine Learning, volume 97 of Proceedings of Machine Learning Research, pages 2790–2799. PMLR.

Hu, E. J., yelong shen, Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., and Chen, W. (2022). LoRA: Low-rank adaptation of large language models. In International Conference on Learning Representations.

Lester, B., Al-Rfou, R., and Constant, N. (2021). The power of scale for parameter-efficient prompt tuning. In Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, pages 3045–3059, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Li, X. L. and Liang, P. (2021). Prefix-tuning: Optimizing continuous prompts for generation. In Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers), pages 4582–4597, Online. Association for Computational Linguistics.

Liu, H., Tam, D., Muqeeth, M., Mohta, J., Huang, T., Bansal, M., and Raffel, C. A. (2022). Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning. In Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K., and Oh, A., editors, Advances in Neural Information Processing Systems, volume 35, pages 1950–1965. Curran Associates, Inc.

Mahabadi, R. K., Henderson, J., and Ruder, S. (2021). Compacter: Efficient low-rank hypercomplex adapter layers. In Ranzato, M., Beygelzimer, A., Dauphin, Y. N., Liang, P., and Vaughan, J. W., editors, Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual, pages 1022–1035.

Misra, D. (2020). Mish: A self regularized non-monotonic activation function. In 31st British Machine Vision Conference 2020, BMVC 2020, Virtual Event, UK, September 7-10, 2020. BMVA Press.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. (2019). Language models are unsupervised multitask learners.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. (2020). Exploring the limits of transfer learning with a unified text-to-text transformer. Journal of Machine Learning Research, 21(140):1–67.

Tahaei, M., Charlaix, E., Nia, V., Ghodsi, A., and Rezagholizadeh, M. (2022). KroneckerBERT: Significant compression of pre-trained language models through kronecker decomposition and knowledge distillation. In Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pages 2116–2127, Seattle, United States. Association for Computational Linguistics.

Thakker, U., Fedorov, I., Beu, J., Gope, D., Zhou, C., Dasika, G., and Mattina, M. (2019). Pushing the limits of rnn compression. In 2019 Fifth Workshop on Energy Efficient Machine Learning and Cognitive Computing-NeurIPS Edition (EMC2-NIPS), pages 18–21. IEEE.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. u., and Polosukhin, I. (2017). Attention is all you need. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, Advances in Neural Information Processing Systems, volume 30. Curran Associates, Inc.

Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., and Bowman, S. (2018). 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, pages 353–355, Brussels, Belgium. Association for Computational Linguistics.

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Davison, J., Shleifer, S., von Platen, P., Ma, C., Jernite, Y., Plu, J., Xu, C., Le Scao, T., Gugger, S., Drame, M., Lhoest, Q., and Rush, A. (2020). Transformers: State-of-the-art natural language processing. In Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, pages 38–45, Online. Association for Computational Linguistics.

Zhang, Z., Guo, W., Meng, X., Wang, Y., Wang, Y., Jiang, X., Liu, Q., and Yang, Z. (2023). HyperPELT: Unified parameter-efficient language model tuning for both language and vision-and-language tasks. In Findings of the Association for Computational Linguistics: ACL 2023, pages 11442–11453, Toronto, Canada. Association for Computational Linguistics.

A Kronecker Factors vs Down and Up Projections

Table 3 shows the details about the Kronecker factors that replaced the projections in the LoRA modules.

B Ablation Study

B.1 KronA Initialization

Our empirical results show that the initialization of the Kronecker factors affects the performance of KronA. Table 4 shows the performance of two investigated strategies for the initialization. We observe that by initializing one of the Kronecker factors from a Kaiming-uniform ($a = \sqrt{5}$) distribution and the other one with zero, KronA adapters perform significantly better than initializing both of the factors from a Normal ($\mu = 0, \sigma = \frac{1}{\sqrt{d_h}}$, where d_h is the embedding dimension) distribution.

Module Name	Factor Name	Symbol	Shape	Parameters	Module Parameters	Constraint
KronA	Kronecker Factor Kronecker Factor	$\mathbf{A}_k$ $\mathbf{B}_k$	$m \times n$ $i \times j$	$m_1 n_1$ $m_2 n_2$	$m_1 n_1 + m_2 n_2$	$m_1 m_2 = n_1 n_2 = d_h$
LoRA	Down Projection Up Projection	$\mathbf{A}$ $\mathbf{B}$	$d_h \times r$ $r \times d_h$	$d_h r$ $d_h r$	$2 d_h r$	$r < \frac{d_h}{2}$

Table 3: This table compares some details of the Kronecker factors with the LoRA projections.

Init Method	CoLA	RTE	MRPC	SST2	STSB	MNLI	QNLI	QQP	Avg
$\mathbf{A}_K$,$\mathbf{B}_K \sim$ Normal	63.36	66.91	91.69	91.97	90.46	86.03	92.33	90.19	84.12
$\mathbf{A}_K \sim$ KU ,$\mathbf{B}_K$=0	63.27	77.70	92.52	94.04	91.26	86.03	93.13	90.57	86.06

Table 4: This table shows the performance of KronA on GLUE using different initialization options for the Kronecker factors.

Modification	CoLA	RTE	MRPC	SST-2	STS-B	MNLI	QNLI	QQP	Avg
Sequential	15.26	53.28	86.39	87.38	83.78	74.70	84.29	86.63	71.46
Parallel	58.17	69.78	91.58	93.81	90.86	85.68	93.35	90.14	84.17
Parallel+Scale (PS)	62.27	70.50	91.58	94.04	91.01	86.16	93.39	90.61	84.94
PS+SiLU	62.74	69.78	91.89	94.15	90.97	85.98	93.30	90.15	84.87
PS only on FFN	63.74	72.66	92.20	94.72	90.98	85.98	93.12	90.68	85.51

Table 5: This table shows the performance of KronA$^{\text{B}}$ after implementing step by step modifications on GLUE.

nonlinear function	Mish	ReLU	GELU	GELU_{new}	SiLU
QNLI Performance	93.21	93.28	93.13	93.26	93.30

Table 6: This table shows the performance of KronA$^{\text{B}}$ on QNLI using Mish (Misra, 2020), ReLU (Agarap, 2018), GELU (Hendrycks and Gimpel, 2016), GELU_{new}, and SiLU (Elfwing et al., 2018a) as different non-linear functions.

B.2 Step by Step Improvement of KronA$^{\text{B}}$

At first, KronA$^{\text{B}}$ was initialized like a normal adapter. It was sequentially inserted after both FFN and attention blocks and it did not have a scaling factor. We modified our module based on (He et al., 2022) to improve its performance.

Table 5 shows the results of our experiments. We observed that inserting KronA$^{\text{B}}$ modules in parallel to the PLM modules, rather than sequentially inserting them, significantly improves the performance. Additionally, adding a scaling factor to our module further increases the GLUE score. Furthermore, adding two modules to each FFN instead of adding to both the FFN and the attention blocks resulted in a higher score.

In addition, motivated by the presence of a non-linear function in PA and Adapter, we tested different non-linear functions between the two multiplications (by $\mathbf{A}_k^{\text{T}}$ and $\mathbf{B}_k$) in Equation 2. As Table 6 shows, SiLU (Elfwing et al., 2018b) is the best option among others, but according to Table 5, adding SiLU decreases the GLUE score of KronA$^{\text{B}}$. Therefore, we removed the nonlinearity from our module.

B.3 Learnable Residual Connection

In the $\text{KronA}^{\text{B}}_{\text{res}}$ module, a residual connection multiplied by a learnable scale is added to the output of KronA$^{\text{B}}$. Also, we studied another scenario in which the learnable scale is passed through a sigmoid function and then multiplied by the residual connection. This module is called $\text{KronA}^{\text{B}}_{\text{sigres}}$. We wanted to answer the question "Is it better to limit the residual scale between 0 and 1?". Our empirical results (Table 7) show that by adding the sigmoid function, the performance of the module drops and the latency increases. Therefore, the sigmoid function was removed from our module.

C Details of Measuring Training and Inference Time

To measure the inference latency, a random dummy input with a batch size equal to one and a sequence length equal to ten is generated. Then, the dummy input is given to the model for 150 iterations to warm up the GPU. Finally, the dummy input is fed to the model for 200 iterations and the

Method	Avg Score	Traning Time
$\text{KronA}^{\text{B}}_{\text{res}}$	**86.57**	1
$\text{KronA}^{\text{B}}_{\text{sigres}}$	86.42	1.18

Table 7: This table shows the effect of adding a sigmoid function to the $\text{KronA}^{\text{B}}_{\text{res}}$ module. "Avg Score" is the averaged score on the GLUE tasks and "Traning Time" represents the relative training time.

Method	CoLA	RTE	MRPC	SST-2	STS-B	MNLI	QNLI	QQP	Avg
Fine-tuning	1	1	1	1	1	1	1	1	1
BitFit	0.58	0.64	0.65	0.66	0.66	0.65	0.64	0.62	0.64
Adapter	0.82	0.71	0.72	0.78	0.76	0.72	0.69	0.69	0.73
LoRA	0.79	0.69	0.7	0.76	0.72	0.7	0.68	0.68	0.72
KronA	0.8	0.72	0.75	0.81	0.77	0.74	0.73	0.73	0.75
Compacter	0.88	0.74	0.78	0.86	0.81	0.75	0.74	0.76	0.79
PA	0.7	0.78	0.81	0.73	0.7	0.67	0.66	0.65	0.71
KronA^{B}	0.84	0.7	0.72	0.79	0.75	0.72	0.7	0.71	0.74
$\text{KronA}^{\text{B}}_{\text{res}}$	0.91	0.85	0.81	0.91	0.76	0.78	0.73	0.74	0.81

Table 8: This table shows the normalized training time of methods on the GLUE tasks.

Shape of $\mathbf{A}_k$	MNLI (Accuracy)
(48, 16)	86.50
(32, 24)	86.31
(3, 256)	86.16
(24, 32)	86.40
(2, 384)	**86.63**
(192, 4)	86.46
(12, 64)	86.56

Table 9: This table shows the performance of the tested options for $\mathbf{A}_k$ in KronA on MNLI. Note that for each option, the shape of the corresponding $\mathbf{B}_k$ is in the reversed order of $\mathbf{A}_k$.

required time to generate the output is measured, averaged, and recorded. This experiment is repeated three times and the average latency is reported. Finally, the reported latencies are normalized and shown in Table 2.

Table 8 shows the normalized training time for each technique on the GLUE tasks. All the experiments are done with the same number of epochs, batch size, number of GPUs, and gradient accumulation step.

D Experimental Setups and Hyperparameters

D.1 Datasets

We used the GLUE (Wang et al., 2018) benchmark to evaluate our methods compared to the baselines. This benchmark covers a variety of tasks including natural language inference (MNLI, RTE, QNLI), linguistic acceptability (CoLA), similarity and paraphrasing (MRPC, QQP), and sentiment classification (SST-2)]. The original GLUE test labels are not published, so similar to (Mahabadi et al., 2021; Zhang et al., 2023), we generated our test sets from the evaluation and the training data. For the small datasets (CoLA, RTE, MRPC, and STSB), we used half of the task dev set for evaluation and the other half as the test set. For the rest of the GLUE tasks with larger datasets, we took 1K samples out of the train set and used it as our test set. The reported evaluation metric for CoLA, MRPC, and STS-B, is the Matthew correlation coefficient, F1, and the average of Pearson/Spearman correlations, respectively. Accuracy is used for the other tasks.

Fine-tuning hyperparameters					
Task	learning rate	batch size	warmup steps	source sentence length	epoch
GLUE	3e-4	100	500	128	20

Table 10: This table shows the hyperparameters used for fine-tuning experiments on the GLUE tasks.

BitFit hyperparameters					
Task	learning rate	batch size	warmup steps	source sentence length	epoch
GLUE	3e-4	100	500	128	20

Table 11: This table shows the hyperparameters used for BitFit experiments on the GLUE tasks.

D.2 Experimental Setup

All experiments were performed on one NVIDIA Tesla V100. We used PyTorch and Hugging Face Transformers library (Wolf et al., 2020) for our experiments. To re-implement LoRA[2] and PA[3], we used their publicly available code. For the experiments on Compacter, BitFit, Fine-tuning, and Adapter, we used the Compacter's[4] official code. The Backbone model for this work is $T5_{base}$ (Raffel et al., 2020).

The size of the trainable parameters for all of the methods is set roughly equal to have a fair comparison. However, for BitFit tuning, we could not match the trainable parameters since all of the biases are trainable.

Given the number of trainable parameters, we have several choices for the shape of the Kronecker factors. For KronA, We tested some of the options and selected the option with the best results.

KronA^{B} and $\text{KronA}^{\text{B}}_{\text{res}}$ modules can have one or two biases. We selected the number of biases that maximized the score on each task.

D.3 Hyperparameters

Since we wanted to ignore the effect of the scaling factor when comparing LoRA and KronA, the scaling factor for these two modules is set to one in all of the experiments.

For Fine-tuning, BitFit, Compacter, and Adapter experiments, we used the hyperparameters that are mentioned in (Mahabadi et al., 2021). However, we changed the learning rate and the rank of the modules to match the desired number of trainable parameters in the Adapter experiments.

The rank of LoRA and PA is set to one and two, respectively. For the KronA modules, the shape of the Kronecker factors is selected based on the best dev results among different options for the shapes. Due to time and resource limitations, we did not tune the shape of the Kronecker factors for KronA^{B} and $\text{KronA}^{\text{B}}_{\text{res}}$.

All of the other hyperparameters are set based on (Mahabadi et al., 2021), except for the learning rate and scaling factor which are tuned based on the best dev results. All of the methods are trained for 20 epochs and the checkpoint that achieves the best performance on the dev set is reported as the final model. Tables 10, 11, 12, 13, 14, 15, 16, 17 and 18 show the tuned hyperparameters for each method on the GLUE tasks.

[2]See https://github.com/microsoft/LoRA.

[3]See https://github.com/jxhe/unify-parameter-efficient-tuning.

[4]See https://github.com/rabeehk/compacter.

Adapter hyperparameters				
Task	learning rate	batch size	task reduction factor	epoch
GLUE	3e-3	100	32	20

Table 12: This table shows the hyperparameters used for Adapter experiments on the GLUE tasks.

Compacter hyperparameters					
Task	learning rate	batch size	hypercomplex division	task reduction factor	epoch
GLUE	3e-3	100	4	32	20

Table 13: This table shows the hyperparameters used for Compacter experiments on the GLUE tasks.

LoRA hyperparameters					
Task	learning rate	batch size	rank	s	epoch
GLUE	1e-3	100	1	1	20

Table 14: This table shows the hyperparameters used for LoRA experiments on the GLUE tasks.

PA hyperparameters					
Task	learning rate	batch size	rank	s	epoch
CoLA	3e-3	100	2	16	20
RTE	5e-3	100	2	16	20
MRPC	5e-3	100	2	16	20
SST-2	1e-3	100	2	16	20
STS-B	1e-3	100	2	16	20
MNLI	1e-3	100	2	16	20
QNLI	1e-3	100	2	16	20
QQP	1e-3	100	2	16	20

Table 15: This table shows the hyperparameters used for PA experiments on the GLUE tasks.

KronA hyperparameters						
Task	learning rate	batch size	$\mathbf{A}_k$	$\mathbf{B}_k$	s	epoch
CoLA	1e-3	100	(32,24)	(24,32)	1	20
RTE	2e-3	100	(32,24)	(24,32)	1	20
MRPC	1e-3	100	(32,24)	(24,32)	1	20
SST-2	1e-3	100	(24,32)	(32,24)	1	20
STS-B	1e-3	100	(2,384)	(384,2)	1	20
MNLI	1e-3	100	(2,384)	(384,2)	1	20
QNLI	1e-3	100	(3,256)	(256,3)	1	20
QQP	1e-3	100	(24,32)	(32,24)	1	20

Table 16: This table shows the hyperparameters used for KronA experiments on the GLUE tasks.

KronA$^{\mathbf{B}}$ hyperparameters							
Task	learning rate	batch size	$\mathbf{A}_k$	$\mathbf{B}_k$	s	module bias	epoch
CoLA	1e-3	100	(32,24)	(24,32)	16	2	20
RTE	5e-3	100	(32,24)	(24,32)	16	1	20
MRPC	5e-3	100	(32,24)	(24,32)	16	1	20
SST-2	1e-3	100	(32,24)	(24,32)	4	1	20
STS-B	1e-3	100	(32,24)	(32,24)	16	1	20
MNLI	1e-3	100	(32,24)	(24,32)	4	2	20
QNLI	1e-3	100	(32,24)	(24,32)	4	1	20
QQP	1e-3	100	(32,24)	(24,32)	4	1	20

Table 17: This table shows the hyperparameters used for KronA$^{\mathrm{B}}$ experiments on the GLUE tasks.

KronA$^{\mathbf{B}}_{\mathbf{res}}$ hyperparameters							
Task	learning rate	batch size	$\mathbf{A}_k$	$\mathbf{B}_k$	s	module bias	epoch
CoLA	1e-3	100	(32,24)	(24,32)	16	2	20
RTE	5e-3	100	(32,24)	(24,32)	16	2	20
MRPC	5e-3	100	(32,24)	(24,32)	16	2	20
SST-2	1e-3	100	(32,24)	(24,32)	16	1	20
STS-B	9e-4	100	(32,24)	(32,24)	16	1	20
MNLI	1e-3	100	(32,24)	(24,32)	16	1	20
QNLI	1e-3	100	(32,24)	(24,32)	4	2	20
QQP	1e-3	100	(32,24)	(24,32)	16	1	20

Table 18: This table shows the hyperparameters used for KronA$^{\mathrm{B}}_{\mathrm{res}}$ experiments on the GLUE tasks.