Tiny-Attention Adapter: Contexts Are More Important Than the Number of Parameters

Hongyu Zhao[*]
University of Chicago
hongyuz@uchicago.edu

Hao Tan
Adobe Research
hatan@adobe.com

Hongyuan Mei
TTI-Chicago
hongyuan@ttic.edu

Abstract

Adapter-tuning is a paradigm that transfers a pretrained language model to downstream tasks by adding and tuning a small number of new parameters. Previously proposed adapter architectures are all feed-forward neural networks. In this paper, we investigate the effectiveness of using *tiny-attention*—i.e., attention with extremely small per-head dimensionality—as adapters. Our tiny-attention adapter learns to modify the hidden states at each position directly conditioned on the hidden states at all the other positions, which is missed by the previously proposed adapters. Moreover, we view its multiple attention heads as a mixture of experts and propose to average their weights during deployment, which further reduces its inference computation cost. On the GLUE benchmark, our tiny-attention adapter outperforms the other parameter-efficient transfer learning methods as well as full fine-tuning while only updating 0.05% of the parameters. On the FewGLUE benchmark, its performance is comparable to that of GPT-3 and PET.

1 Introduction

Transferring a large pretrained language model (PLM) is a de facto paradigm of performing downstream tasks in natural language processing (NLP). A general approach is adapter-tuning, which means inserting *adapters*—i.e., neural networks with small numbers of parameters—into each pretrained layer and only updating the adapter parameters. Adapter-tuning is parameter-efficient and enjoys low computation cost since it keeps the PLM frozen. But it underperforms full fine-tuning which updates all the parameters of the PLM.

In this paper, we propose a new adapter architecture which outperforms full fine-tuning yet uses even fewer parameters than the previously proposed adapters as well as other parameter-efficient transfer learning methods; see Figure 1.

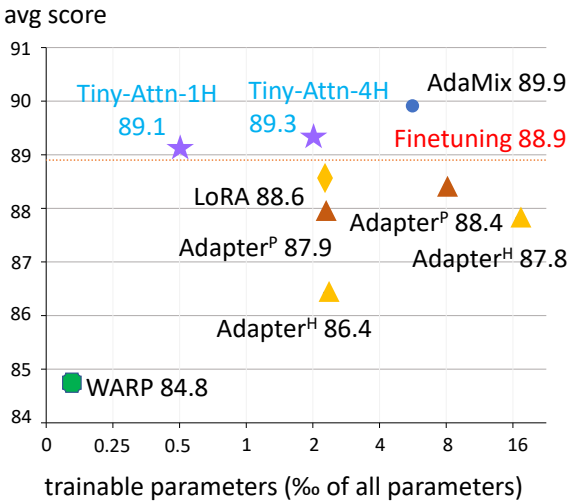


Figure 1: Average performance of different parameter-efficient transfer learning methods on the GLUE benchmark with roberta-large as the PLM. Our method is Tiny-Attn (with ★ marker): "1H" and "4H" mean "one attention head" and "four attention heads (with the parameter-averaging trick in section 2.2)" respectively.

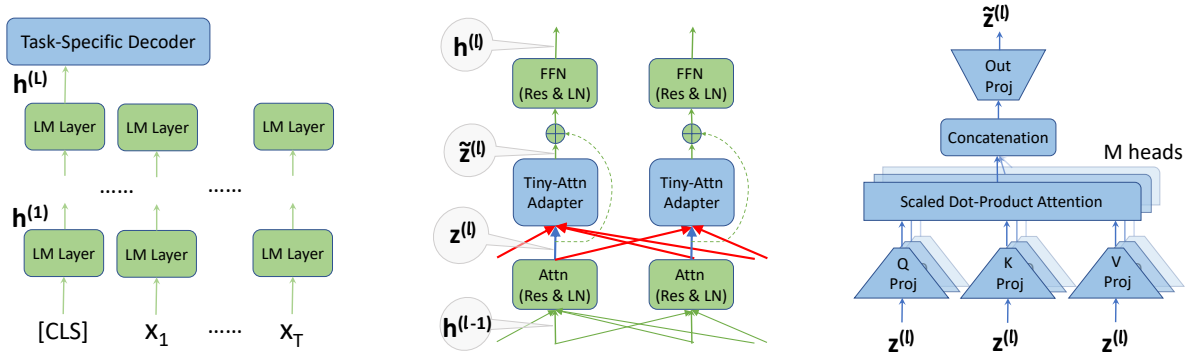
Our adapter is a multi-head attention module and its per-head dimensionality is extremely small; thus we call it *tiny-attention*. The architecture design is inspired by the following intuitions:

1. For each input sequence, each layer of the language model produces an embedding for each token in the sequence; see Figure 2a.
2. All the parameter-efficient transfer learning methods learn to modify the embeddings towards the direction of performing the given task well; see He et al. (2021) for a thorough theoretical and empirical discussion.
3. Almost all the previously proposed adapter architectures are feed-forward neural networks. Thus, we suspect that their embedding modifications are not as contextually rich as they should.

Therefore, we propose to use the attentative structure that allows the embedding modifications of each token to capture more contextual information

[*]Work done while during internship at TTI-Chicago.

Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing, pages 6626 - 6638
December 7-11, 2022



(a) Transferring a pretrained language model for downstream tasks.

(b) Adapting a language model layer by adding our tiny-attention adapter.

(c) The internal architecture of our tiny-attention adapter.

Figure 2: The pipeline of applying our tiny-attention adapter to a pretrained language model for downstream tasks.

by directly looking at the embeddings of all the tokens. The dimensionality of each attention head need not to be large. In NLP tasks, contextual information is often demonstrated more important than model sizes: for example, in language modeling, a smaller model with a larger context window usually outperforms a larger model with a smaller context window (Dai et al., 2019; Wu et al., 2022; Yang et al., 2019). Additionally, we view the multiple attention heads of our tiny-attention adapter as a mixture of experts and then propose to average their weights during inference. This technique further reduces the inference cost.

We evaluated our tiny-attention adapter on the GLUE (Wang et al., 2018) and FewGLUE (Schick and Schütze, 2021; Wang et al., 2019) benchmarks. On GLUE, it updates only 0.05% of the parameters—an order of magnitude smaller than all the other methods—yet still outperforms full fine-tuning and nearly all the other parameter-efficient tuning methods. On FewGLUE, it is comparable to several strong competing methods including PET (Schick and Schütze, 2021) and GPT-3 (Brown et al., 2020). We also conducted ablation studies to investigate its effectiveness with varying placements and PLMs.

2 The Method

Figure 2a illustrates how a language model performs a downstream task. The language model has L layers. Given an input sequence $\mathbf{x} = x_0 x_1 \dots x_T$ where x_0 is a special classification (CLS) token, each layer ℓ reads the embeddings given by the previous layer $\ell - 1$ and produces the layer-ℓ embeddings $\mathbf{h}_0^{(\ell)} \mathbf{h}_1^{(\ell)} \dots \mathbf{h}_T^{(\ell)}$. Then a task-specific decoderreads the top-layer embedding $\mathbf{h}_0^{(L)}$ of the CLS token and predicts a task-specific output $\hat{y}$ for that sequence.

Transferring the language model involves updating its *trainable* parameters to minimize a task-specific loss$(\hat{y}, y)$ where y is the ground-truth label for the given $\mathbf{x}$. For adapter-tuning, the trainable parameters only include those of the task-specific decoder and those of the adapters. Figure 2b shows a language model layer with our tiny-attention adapter placed between its attention module and feed-forward net: during training, only the decoder and adapter parameters (blue) are updated while the pretrained parameters (green) are all kept frozen.

2.1 Tiny-Attention Adapter

Our adapter has an attentative structure: as shown in Figure 2b, at each position t, it takes as input the intermediate embeddings $\mathbf{z}^{(\ell)}$ from not only the current position (information flow indicated by blue arrows ←) but also all the other positions (information flow shown by red arrows ←). For each token t, it produces a task-specific modification $\tilde{\mathbf{z}}_t^{(\ell)}$. Then the modified embeddings $\mathbf{z}_t^{(\ell)} + \tilde{\mathbf{z}}_t^{(\ell)}$ are fed into the layer ℓ feed-forward net for producing $\mathbf{h}_t^{(\ell)}$.

As shown in Figure 2c, the internal architecture of our tiny-attention adapter resembles an ordinary multi-head attention mechanism. Suppose it has M attention heads. Each attention head m produces a head-specific attention vector $\tilde{\mathbf{z}}_t^{(\ell,m)}$ and the final attention vector $\tilde{\mathbf{z}}_t^{(\ell)}$ is obtained by projecting the concatenation of all the $\tilde{\mathbf{z}}_t^{(\ell,m)}$:

$$\tilde{\mathbf{z}}_t^{(\ell)} \stackrel{\text{def}}{=} \mathbf{O}^{(\ell)}[\tilde{\mathbf{z}}_t^{(\ell,1)}; \dots; \tilde{\mathbf{z}}_t^{(\ell,M)}] \quad (1a)$$

$$\tilde{\mathbf{z}}_t^{(\ell,m)} \stackrel{\text{def}}{=} \text{Attn}_t^{(m)}(\mathbf{z}_0^{(\ell)}, \mathbf{z}_1^{(\ell)}, \dots, \mathbf{z}_T^{(\ell)}) \quad (1b)$$

The math details of $\text{Attn}_t^{(m)}$ is given in Appendix A.

Why *attention* as adapter? Attention allows the task-specific modification $\tilde{\mathbf{z}}_t^{(\ell)}$ for each token t to aggregate useful information from its full contexts at $t = 0, 1, \dots, T$. It is analogous to how

the attention modules in the pretrained language model learned to construct contextual representations which helped optimize the language modeling objective during pretraining. Therefore, when the pretrained attention modules are frozen, it seems natural to adopt new trainable attention modules for their desired behavior.

The key difference between our tiny-attention and an ordinary attention is that our per-head dimension is *tiny*: i.e., $\tilde{\mathbf{z}}_t^{(\ell,m)} \in \mathbb{R}^D$ and D is very small. Throughout our experiments, we set $D = 1$.

Why *tiny*-attention? Smaller dimensionality means fewer trainable parameters and less computation cost; thus it is preferred. We believe that small dimensionality is sufficient in our setting because of two key observations. First, a smaller model often tends to work as well as a larger model if it is allowed to use a larger context; see section 1 for the example of language modeling (Dai et al., 2019; Wu et al., 2022; Yang et al., 2019). Second, Hu et al. (2021) found that feed-forward adapters can achieve competitive results with extremely low-rank (1 or 2) parameter matrices. Similar to the LoRA method by Hu et al. (2021), our tiny-attention adapter essentially performs a low-rank non-linear projection: it first linearly transforms the high-dimensional embeddings $\mathbf{z}^{(\ell)}$ to low-dimensional query, key, and value vectors; it then linearly transforms the low-dimensional attention vectors—after the non-linear attention operation—to the high-dimensional modification vectors $\tilde{\mathbf{z}}^{(\ell)}$; see Appendix A for technical details.

2.2 Multiple Heads as a Mixture of Experts

The multiple attention heads in our tiny-attention can be regarded as a mixture of experts where each head is an expert that specializes in capturing certain kinds of contextual information (e.g., syntax, semantics). The output projection in equation (1a) learns to aggregate the information $\tilde{\mathbf{z}}_t^{(\ell,m)}$ produced by the experts. Rearranging that equation gives

$$\tilde{\mathbf{z}}_t^{(\ell)} \overset{\text{def}}{=} \sum_{m=1}^{M} \mathbf{O}^{(\ell,m)} \tilde{\mathbf{z}}_t^{(\ell,m)} \tag{2}$$

where the per-head matrices $\mathbf{O}^{(\ell,m)}$ are defined such that $\mathbf{O}^{(\ell)} = [\mathbf{O}^{(\ell,1)}; \ldots; \mathbf{O}^{(\ell,M)}]$.

It inspires us to propose a parameter-averaging trick that is able to further reduce the storage and computation cost of our method. Precisely, *after* training, we average the output projection matrices $\mathbf{O}^{(\ell,m)}$ as well as the attention parameters inside $\text{Attn}^{(m)}$ across the attention heads. Then we only store the averaged parameters. During inference, we only use a single attention head into which the stored parameters have been loaded. That way, although we may have trained $M > 1$ attention heads, our storage and inference cost will be as low as if we had only trained a single head. The technical details of this trick is discussed in Appendix A.

3 Related Work

There are three major paradigms of parameter-efficient transfer learning. The first is to only fine-tune a small subset of the existing parameters in a pretrained language model (Howard and Ruder, 2018; Lee et al., 2019; Zaken et al., 2021). The second is adapter-tuning (Houlsby et al., 2019): inserting adapters (i.e., small neural nets) into the language model and only tuning their parameters. The third is prefix-tuning (Li and Liang, 2021; Hambardzumyan et al., 2021; Liu et al., 2021b,a; Lester et al., 2021): augmenting the input sequence with trainable tokens and only updating the new token embeddings. Both adapter-tuning and prefix-tuning keeps the PLM frozen. Our work falls into the category of adapter-tuning. The key difference is that our proposed adapter has an attention architecture. The previously proposed methods in this direction all use feed-forward neural networks: they are Houlsby et al. (2019); Lin et al. (2020); Pfeiffer et al. (2021); Hu et al. (2021); He et al. (2021).

AdaMix (Wang et al., 2022) proposes a stochastic routing strategy to mix an ensemble of adapters and it is orthogonal to all the adapter methods including ours. Akin to our parameter-averaging trick, they propose to average the adapter parameters for low-cost storage and inference. Similar tricks are used in Ravi (2017); Matena and Raffel (2021); Wortsman et al. (2022).

4 Experiments

We evaluated our proposed tiny-attention adapter on a range of natural language understanding tasks including GLUE and FewGLUE. Our method is implemented in PyTorch (Paszke et al., 2019) and heavily relies on HuggingFace (Wolf et al., 2020). Our code is submitted for review and it will be publicly released after the paper is published.

In all of our experiments, we set the dimensionality of each tiny-attention head (i.e., the dimension of query, key, and value vectors) to be *one*. Other experiment details (e.g., hyperparameters) can be found in Appendix B.

4.1 Main Results: GLUE and FewGLUE

GLUE. On the GLUE benchmark, we chose the RoBERTa model (Liu et al., 2019) as our PLM and we used the pretrained roberta-large weights (355M parameters) downloaded from HuggingFace.

Our results on the GLUE benchmark are already presented in Figure 1. As we can see, our method (Tiny-Attn-1H and Tiny-Attn-4H) outperforms all the previously proposed parameter-efficient tuning methods as well as fine-tuning. Yet our method uses significantly fewer trainable parameters than the other methods except WARP. The single-head version (Tiny-Attn-1H) trains 176K parameters, which only counts as 0.05% of the PLM parameters. The four-head version (Tiny-Attn-4H) further improves the performance with an increased training cost, but its storage and inference cost remains the same as the single-head version.

Our method only underperforms AdaMix, which learns a stochastic routing strategy to mix an ensemble of adapters. But AdaMix uses significantly more trainable parameters than ours. Moreover, AdaMix's technique is orthogonal and complementary to most other adapter methods including ours.

Our results on each individual GLUE task can be found in Tables 8 and 9 of Appendix C.1.

FewGLUE. We also evaluated our method on the CB and RTE tasks of the FewGLUE benchmark. They are extremely few-shot settings: each task only has 32 training examples. We chose ALBERT (Lan et al., 2019) as our PLM and we used the pretrained albert-xxlarge-v2 weights (223M parameters) downloaded from HuggingFace. The detailed setting can be found in Appendix B.1. The result is shown in Table 1. It turns out that the performance of our method is comparable to that of PET (Schick and Schütze, 2021) and GPT-3 (Brown et al., 2020).

method	CB	RTE
Tiny-Attn-1H (ours)	88.57±2.99	68.38±2.53
WARP	87.5	71.8
PET	85.1	69.8
iPET	92.9	74
GPT-3-Small	42.9	52.3
GPT-3-Medium	58.9	48.4
GPT-3	82.1	72.9

Table 1: Results on the validation set of FewGLUE. We report accuracy for all tasks.

4.2 Analysis

Sequential vs. parallel. In section 2, we presented the 'sequential' methods where our tiny-attention modules are placed between the pretrained attention and feed-forward net. Another option is to put the tiny-attention module in 'parallel' to the original attention layer as in He et al. (2021), as illustrated in Figure 3. We emperically

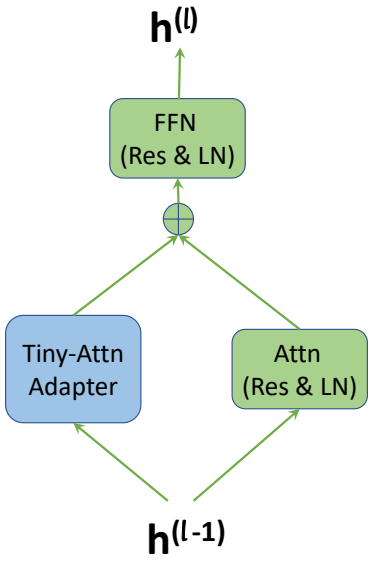


Figure 3: Illustration of the parallel structure.

found that these two design choices have negligible differences in results. Detailed Results are listed in Table 10 of Appendix C.2.

Effects of parameter-averaging. Recall that we used the parameter-averaging trick in our experiments for an improved inference efficiency. But do we lose any performance by using this trick? Through ablation studies on CoLA and RTE, we found that using our parameter-averaging trick actually slightly improves the results. Detailed results are shown in Table 11 of Appendix C.1.

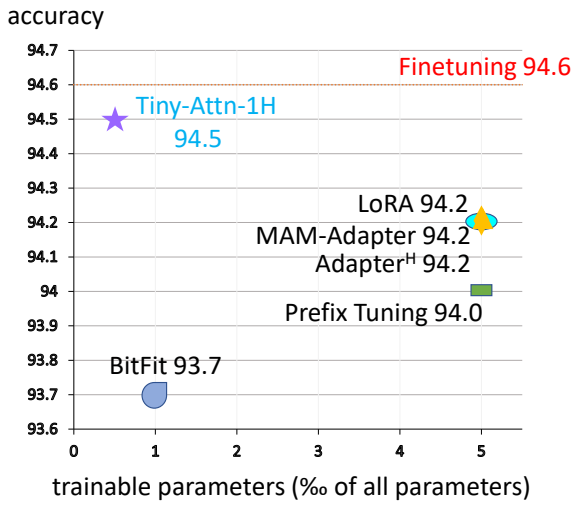
Does the size of PLM matter? Parameter-efficient tuning methods are known to suffer performance drop when working with small-sized PLMs. To investigate this effect, we also experimented with the pretrained roberta-base (125M parameters) downloaded from Huggingface on the MNLI and SST-2 tasks. The results are shown in Figure 4. Different methods suffer almost the same amount of performance drop.[1] But our method enjoys a much larger drop in the number of trainable parameters: the trainable parameters of our method

[1]LoRA and Adapter^H are the only previous methods that reported GLUE results for both roberta-large and roberta-base.

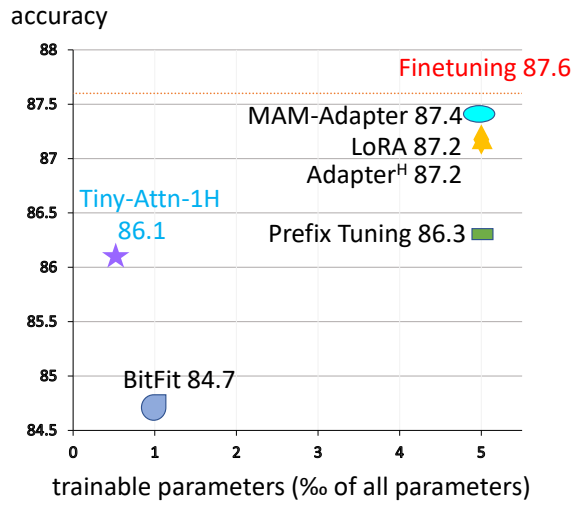
method	performance drop MNLI	SST-2	average
LoRA	3.4	2	2.7
Adapter^H	3.1	2.1	2.6
Tiny-Attn-1H (ours)	3.2	2.1	2.65

Table 2: Performance drop of different methods on MNLI and SST-2, with roberta-large and roberta-base as the PLMs.

(Tiny-Attn-1H) still only count as 0.05% of the PLM parameters, but those of LoRA and Adapter^H increase from 0.23% to 0.5%. See Table 2.



(a) SST-2



(b) MNLI

Figure 4: Performance of different parameter-efficient tuning methods on SST-2 and MNLI with RoBERTa-base. The baseline results are from He et al. (2021).

Effects of larger dimensions. We set $D = 1$ in all our main experiments. It's natural to ask that what if we use a larger dimension. We experimented with a $D = 4$ variant of our method, which we call Tiny-Attn-1H4D, on CoLA and RTE. We found that this variant performs slightly worse than the standard version Tiny-Attn-1H. It's normal that more parameters in the adapters could lead to slightly worse performance on some GLUE tasks, see Hu et al. (2021). Detailed results are shown in Table 12 of Appendix C.1.

5 Conclusion

In this paper, we presented the tiny-attention adapter. While previous adapter-tuning only processes in-place embeddings, our method considers the context from other positions. Thanks to this contextual modeling, the size of our tiny-attention adapter can be extremely light (e.g., only 1 attention head and 1 head dimension). To further enable trade-off between performance and training cost, we proposed the weight-averaging technique. On GLUE benchmark, our tiny-attention adapter achieved better results than full fine-tuning while only updating 0.05% of parameters. Our model also achieved competitive results under the few-shot setting. Lastly, we compared our methods with the alternatives (e.g., parallel instead of sequential, without parameter averaging) and also showed the generalization to smaller PLMs.

Acknowledgements

This work was supported by a research gift to the last author by Adobe Research. We thank the anonymous EMNLP reviewers and meta-reviewer for their constructive feedback. We also thank our colleagues at Toyota Technological Institute at Chicago for helpful discussion.

Limitations

Our main limitation is that our method was only evaluated on the GLUE and FewGLUE benchmarks and that we haven't experimented with a diverse set of generation tasks (e.g., XSUM (Narayan et al., 2018), E2E (Novikova et al., 2017a)) yet. He et al. (2021) reported that the state-of-the-art parameter-efficient adaptation method on a task or dataset may suffer a sharp performance drop on another task or dataset. Although our method is consistently effective across multiple classification tasks, it is still possible that it won't perform well at generation tasks such as summarization and translation.

Ethics Statement

Our method belongs to the general category of efficient language model fine-tuning and our focus is to reduce the number of trainable parameters.

It can benefit the scenarios when communications become a bottleneck, such as federal learning, distributed training, and edge computing. However, since we do not apply explicit differentiable privacy methods to these updated parameters, the method can be vulnerable to specific attacks (e.g., man-in-the-middle attack).

Our method can also be deployed to on-device applications, where the storage is limited. For these applications, different tasks can just keep a small set of task-specific parameters while other parameters are shared.

Our method can help reduce the carbon footprint of the model training in two ways. Firstly, the model is fine-tuned from a pretrained model thus can be adapted to a downstream task quickly while reaching a satisfying performance. Secondly, our method only updates a small portion of all parameters thus the optimizer only tracks a small part of parameters. For this reason, the same training infrastructure can support larger batch size given the optimizer's states are significantly reduced (e.g., to less than 1% in our Tiny-Attn-1H method).

Meanwhile, our method shares the same possibilities as most of previous efficient training methods, such as misusage, containing data bias, and suffering from adversarial attacks. However, the method developed in this paper is orthogonal to the previous effort to mitigate the above issues.

References

Roy Bar Haim, Ido Dagan, Bill Dolan, Lisa Ferro, Danilo Giampiccolo, Bernardo Magnini, and Idan Szpektor. 2006. The second PASCAL recognising textual entailment challenge.

Luisa Bentivogli, Ido Dagan, Hoa Trang Dang, Danilo Giampiccolo, and Bernardo Magnini. 2009. The fifth PASCAL recognizing textual entailment challenge. In *TAC*.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in Neural Information Processing Systems (NeurIPS)*.

Daniel Cer, Mona Diab, Eneko Agirre, Inigo Lopez-Gazpio, and Lucia Specia. 2017. Semeval-2017 task 1: Semantic textual similarity-multilingual and cross-lingual focused evaluation. *arXiv preprint arXiv:1708.00055*.

Ido Dagan, Oren Glickman, and Bernardo Magnini. 2005. The pascal recognising textual entailment challenge. In *Machine learning challenges workshop*.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc Le, and Ruslan Salakhutdinov. 2019. Transformer-XL: Attentive language models beyond a fixed-length context. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*.

Marie-Catherine De Marneffe, Mandy Simons, and Judith Tonhauser. 2019. The commitmentbank: Investigating projection in naturally occurring discourse. In *proceedings of Sinn und Bedeutung*.

William B Dolan and Chris Brockett. 2005. Automatically constructing a corpus of sentential paraphrases. In *Proceedings of the International Workshop on Paraphrasing*.

Danilo Giampiccolo, Bernardo Magnini, Ido Dagan, and Bill Dolan. 2007. The third PASCAL recognizing textual entailment challenge. In *Proceedings of the ACL-PASCAL workshop on textual entailment and paraphrasing*.

Karen Hambardzumyan, Hrant Khachatrian, and Jonathan May. 2021. WARP: Word-level Adversarial ReProgramming. 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)*.

Junxian He, Chunting Zhou, Xuezhe Ma, Taylor Berg-Kirkpatrick, and Graham Neubig. 2021. Towards a unified view of parameter-efficient transfer learning. *arXiv preprint arXiv:2110.04366*.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for nlp. In *Proceedings of the International Conference on Machine Learning (ICML)*.

Jeremy Howard and Sebastian Ruder. 2018. Universal language model fine-tuning for text classification. *arXiv preprint arXiv:1801.06146*.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*.

Rabeeh Karimi Mahabadi, James Henderson, and Sebastian Ruder. 2021. Compacter: Efficient low-rank hypercomplex adapter layers. *Advances in Neural Information Processing Systems (NeurIPS)*.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2019. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*.

Jaejun Lee, Raphael Tang, and Jimmy Lin. 2019. What would elsa do? freezing layers during transformer fine-tuning. *arXiv preprint arXiv:1911.03090*.

Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The power of scale for parameter-efficient prompt tuning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*.

Hector J Levesque, Ernest Davis, and Leora Morgenstern. 2011. The Winograd schema challenge. In *AAAI Spring Symposium: Logical Formalizations of Commonsense Reasoning*.

Xiang Lisa Li and Percy Liang. 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)*.

Zhaojiang Lin, Andrea Madotto, and Pascale Fung. 2020. Exploring versatile generative language model via parameter-efficient transfer learning. In *Findings of the Association for Computational Linguistics: EMNLP 2020*.

Xiao Liu, Kaixuan Ji, Yicheng Fu, Zhengxiao Du, Zhilin Yang, and Jie Tang. 2021a. P-tuning v2: Prompt tuning can be comparable to fine-tuning universally across scales and tasks. *arXiv preprint arXiv:2110.07602*.

Xiao Liu, Yanan Zheng, Zhengxiao Du, Ming Ding, Yujie Qian, Zhilin Yang, and Jie Tang. 2021b. Gpt understands, too. *arXiv preprint arXiv:2103.10385*.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

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

Michael Matena and Colin Raffel. 2021. Merging models with fisher-weighted averaging. *arXiv preprint arXiv:2111.09832*.

Shashi Narayan, Shay B Cohen, and Mirella Lapata. 2018. Don't give me the details, just the summary! topic-aware convolutional neural networks for extreme summarization. *arXiv preprint arXiv:1808.08745*.

Behnam Neyshabur, Hanie Sedghi, and Chiyuan Zhang. 2020. What is being transferred in transfer learning? *Advances in Neural Information Processing Systems (NeurIPS)*.

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

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

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems (NeurIPS)*. Curran Associates, Inc.

Jonas Pfeiffer, Aishwarya Kamath, Andreas Rücklé, Kyunghyun Cho, and Iryna Gurevych. 2021. AdapterFusion: Non-destructive task composition for transfer learning. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Sujith Ravi. 2017. Projectionnet: Learning efficient on-device deep networks using neural projections. *arXiv preprint arXiv:1708.00630*.

Andreas Rücklé, Gregor Geigle, Max Glockner, Tilman Beck, Jonas Pfeiffer, Nils Reimers, and Iryna Gurevych. 2021. AdapterDrop: On the efficiency of adapters in transformers. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Timo Schick and Hinrich Schütze. 2021. It's not just size that matters: Small language models are also few-shot learners. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2019. SuperGLUE: A stickier benchmark for general-purpose language understanding systems. *arXiv preprint 1905.00537*.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. 2018. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*.

Yaqing Wang, Subhabrata Mukherjee, Xiaodong Liu, Jing Gao, Ahmed Hassan Awadallah, and Jianfeng Gao. 2022. Adamix: Mixture-of-adapter for parameter-efficient tuning of large language models. *arXiv preprint arXiv:2205.12410*.

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

Adina Williams, Nikita Nangia, and Samuel R. Bowman. 2018. A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of NAACL-HLT*.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*.

Mitchell Wortsman, Gabriel Ilharco, Samir Yitzhak Gadre, Rebecca Roelofs, Raphael Gontijo-Lopes, Ari S Morcos, Hongseok Namkoong, Ali Farhadi, Yair Carmon, Simon Kornblith, et al. 2022. Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time. *arXiv preprint arXiv:2203.05482*.

Yuhuai Wu, Markus Norman Rabe, DeLesley Hutchins, and Christian Szegedy. 2022. Memorizing transformers. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. *Advances in Neural Information Processing Systems (NeurIPS)*.

Elad Ben Zaken, Shauli Ravfogel, and Yoav Goldberg. 2021. Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. *arXiv preprint arXiv:2106.10199*.

A Method Details

The notations we use in this section are inherited from section 2.

Recall that every head-specific attention vector $\tilde{\mathbf{z}}_t^{(\ell,m)}$ is given by equation (1b). If we expand $\text{Attn}_t^{(m)}$ in this equation, we get

$$\begin{aligned}&\text{Attn}_t^{(m)}(\mathbf{z}_0^{(\ell)}, \mathbf{z}_1^{(\ell)}, \ldots, \mathbf{z}_T^{(\ell)}) \\ &= \frac{\sum_{s=0}^{T} \exp\left(\frac{\mathbf{q}_t^{(\ell,m)\top}\mathbf{k}_s^{(\ell,m)}}{\sqrt{D}}\right)\mathbf{v}_s^{(\ell,m)}}{\sum_{s=0}^{T} \exp\left(\frac{\mathbf{q}_t^{(\ell,m)\top}\mathbf{k}_s^{(\ell,m)}}{\sqrt{D}}\right)}\end{aligned} \tag{3}$$

The subscript t in $\text{Attn}_t^{(m)}$ means that the t^{th} intermediate embedding $\mathbf{z}_t^{(\ell)}$ is used to construct the query $\mathbf{q}_t^{(\ell,m)}$. Similarly, for $s = 0, 1, \ldots, T$, $\mathbf{k}_s^{(\ell,m)}, \mathbf{v}_s^{(\ell,m)}$ are low-dimensional key and value vectors that are dependent on the high-dimensional intermediate embedding $\mathbf{z}_s^{(\ell)}$. These vectors are computed by linear transformations:

$$\mathbf{q}_t^{(\ell,m)} = \mathbf{W}_Q^{(\ell,m)}\mathbf{z}_t^{(\ell)} \tag{4a}$$

$$\mathbf{k}_s^{(\ell,m)} = \mathbf{W}_K^{(\ell,m)}\mathbf{z}_s^{(\ell)} \tag{4b}$$

$$\mathbf{v}_s^{(\ell,m)} = \mathbf{W}_V^{(\ell,m)}\mathbf{z}_s^{(\ell)} \tag{4c}$$

where $\mathbf{W}_Q^{(\ell,m)}, \mathbf{W}_K^{(\ell,m)}, \mathbf{W}_V^{(\ell,m)}$ are $H \times 1$ parameter matrices of the attention module, where H is the hidden size of the PLM.

Mixture of experts. During inference, we average all the parameters of the attention module across the attention heads.

$$\mathbf{W}_Q^{(\ell,n)} \leftarrow \tfrac{1}{M}\sum_{m=1}^{M}\mathbf{W}_Q^{(\ell,m)} \tag{5a}$$

$$\mathbf{W}_K^{(\ell,n)} \leftarrow \tfrac{1}{M}\sum_{m=1}^{M}\mathbf{W}_K^{(\ell,m)} \tag{5b}$$

$$\mathbf{W}_V^{(\ell,n)} \leftarrow \tfrac{1}{M}\sum_{m=1}^{M}\mathbf{W}_V^{(\ell,m)} \tag{5c}$$

where $n = 1, 2, \ldots, M$.

Then the attention function $\text{Attn}_t^{(m)}$ will be the same for all m. In other words, we will have $\tilde{\mathbf{z}}_t^{(\ell,1)} = \ldots = \tilde{\mathbf{z}}_t^{(\ell,M)} \overset{\text{def}}{=} \tilde{\mathbf{z}}_t^{(\ell)}$.

Substitute this to equation (2), we get

$$\tilde{\mathbf{z}}_t^{(\ell)} = (\sum_{m=1}^{M}\mathbf{O}^{(\ell,m)})\tilde{\mathbf{z}}_t^{(\ell)} \tag{6}$$

where $\mathbf{O}^{(\ell,m)} \in \mathbb{R}^{1\times H}$ are the output projection matrices. We can define

$$\bar{\mathbf{O}}^{(\ell)} \overset{\text{def}}{=} \tfrac{1}{M}\sum_{m=1}^{M}\mathbf{O}^{(\ell,m)} \tag{7}$$

Then we only use the averaged projection matrix such that equation (2) becomes

$$\mathbf{z}_t^{(\ell)} = M\bar{\mathbf{O}}^{(\ell)}\tilde{\mathbf{z}}_t^{(\ell)} \tag{8}$$

B Experiment Details

We used the PyTorch library (Paszke et al., 2019) and the pretrained language models from the HuggingFace transformers library (Wolf et al., 2020) in all of our experiments.

B.1 GLUE and FewGLUE

GLUE. We evaluated our method on 8 tasks of the GLUE benchmark (Wang et al., 2018): SST-2 (Socher et al., 2013), CoLA (Warstadt et al., 2018), MNLI (Williams et al., 2018), QQP[2], RTE (Dagan et al., 2005; Bar Haim et al., 2006; Giampiccolo et al., 2007; Bentivogli et al., 2009), MRPC (Dolan and Brockett, 2005), QNLI (Rajpurkar et al., 2016) and STS-B (Cer et al., 2017).[3] They are all classification tasks except STS-B, which is formed as a regression task. WNLI (Levesque et al., 2011) is excluded following prior work (Houlsby et al., 2019; Hu et al., 2021).

We used the official data splits. The data were pre-processed by GLUE and HuggingFace and we did not apply any extra modification. The sizes of the training sets of each task are shown in Table 3. All of the tasks have a balanced label distribution in the training set except for MRPC (68% positive) and QQP (63% negative). Besides, the test set of QQP has a different label distribution than the training set.

task	|train|
SST-2	67k
CoLA	8.5k
MNLI	393k
QQP	364k
RTE	2.5k
MRPC	3.7k
QNLI	105k
STS-B	7k

Table 3: Sizes of training sets of tasks on GLUE.

We trained using AdamW (Loshchilov and Hutter, 2017) and either a linear or a cosine learning rate decay scheduler. We used a linear

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

[3] The dataset can be downloaded at https://huggingface.co/datasets/glue.

warmup with approximately 10% of the total steps as the warmup steps in some tasks. The number of epochs was fixed to 20. We evaluated on the validation set twice per epoch and report the best result. We used the same batch size 16 for all our tasks. For reproducibility, we used a fixed random seed. Learning rate (lr) was selected from [1E-6,1E-5,1E-4,5E-4,8E-4,1E-3,1.5E-3,2E-3,3E-3,5E-3,7E-3] and weight decay (dec) was chosen from [0,0.01,0.02,0.03,0.04,0.05,0.1,0.2,0.3,0.4,0.5]. Note that we did not do a full hyperparameter search. On average, we run about 80 experiments for each task. However, the detailed numbers of experiments are varying according to the size of the dataset to save computational cost. E.g., on MNLI and QQP, we only perform about 20 experiments for each. We performed our experiments mainly on 10 Nvidia RTX A4000. The average running time is about eight hours, varying across tasks.

For Tiny-Attn-1H, we initialized the output projection matrices to be very small ($\mathcal{U}(-\frac{0.01}{\sqrt{D}}, \frac{0.01}{\sqrt{D}})$ where D is the per-head dimension) to make our model behave like the original pretrained model at early stages of training. Empirically, we found this trick important in stabilizing the training. The hyperparameters of the best-performing single-head models are shown in Table 4.

task	lr	dec	warmup	scheduler
SST-2	8E-4	0.01	yes	cosine
CoLA	5E-4	0.04	no	linear
MNLI	1E-3	0	yes	linear
QQP	5E-4	0.01	no	linear
RTE	2E-3	0.05	no	linear
MRPC	3E-3	0.3	yes	cosine
QNLI	8E-4	0.02	no	linear
STS-B	1.5E-3	0	no	cosine

Table 4: Hyperparameters for Tiny-Attn-1H.

For Tiny-Attn-4H, we perturbed the weight of Tiny-Attn-1H to initialize every head due to computational limits. In principle, we only need to initialize the weight of every head to be almost the same. This makes the parameter-averaging trick applicable since the weight of these heads would not be very far from each other, just as when we average the parameters of multiple fine-tuned models with the same initialization (Neyshabur et al., 2020). The hyperparmeters we use are shown in Table 5.

task	lr	dec	warmup	scheduler
CoLA	8E-4	0.05	yes	cosine
RTE	1E-5	0.3	no	linear
MRPC	3E-3	0.3	no	linear
QNLI	1E-6	0.4	no	cosine

Table 5: Hyperparameters for Tiny-Attn-4H. Since we initialize from the weight of Tiny-Attn-1H, tasks on which we don't get an improvement are omitted.

FewGLUE. FewGLUE (Schick and Schütze, 2021) is a subset of the SuperGLUE benchmark (Wang et al., 2019) with the sizes of all training sets being 32.[4] We evaluated our model on two tasks of it: CB (De Marneffe et al., 2019) and RTE. We still used AdamW but we did not use a scheduler. We used a linear warmup with 10% of the total steps as the warmup steps for RTE. The number of epochs was fixed to 20. Following Schick and Schütze (2021), we did not use any additional validation set for parameter selection or early stopping and we only report the final result. In contrast to them, we did not use any additional unlabeled examples. We used a fixed learning rate 1E-3 and a fixed batch size 1. We've tried a few different weight decay [0,0.02,0.05,0.1], but eventually we chose the same value 0.02 for all tasks. After selecting the hyperparameters with a fixed random seed, we ran over 4 additional random seeds and report the average performance and the standard variation.

Following Schick and Schütze (2021), we used albert-xxlarge-v2 (Lan et al., 2019) downloaded from the HuggingFace library as the backbone PLM, and used manual prompts to replace the classification head. The manual prompt we use was "[CLS]<hypothesis>?[MASK].<premise><SEP>", the same as theirs. The language model predicts the token at the masked position and the output space is limited to a few selected tokens. We used "yes" to represent "entailment", "no" to represent "contradiction", and "maybe" to represent "neutral".

B.2 Analysis

Sequential vs. parallel. In the main experiments, we used a 'sequential' (seq) structure where our tiny-attention modules are placed between the pretrained attention and feed-forward net. Another option is to put the tiny-attention module in 'parallel' (para) to the original attention layer as in He et al. (2021).

[4]The dataset can be downloaded at https://github.com/timoschick/fewglue.

We used the same setting as our main experiments on GLUE except that we used roberta-base as the backbone PLM. Warmup was used for all experiments. The hyperparameters we used are shown in Table 6.

task	type	lr	dec	scheduler
SST-2	seq	1E-3	0.02	cosine
SST-2	para	1E-3	0.01	cosine
MNLI	seq	1E-4	0.01	linear
MNLI	para	1E-3	0.01	linear

Table 6: Hyperparameters for experiments with roberta-base as the PLM.

Note that there are other possible placements, e.g., we can place tiny-attention adapters above the feed-forward networks. We did not do this because our philosophy is to augment the pretrained attention but not to intervene anything else.

Effects of parameter-averaging. We used the same setting as our Tiny-Attn-4H on GLUE except that we did not use parameter-averaging during inference. Instead, we used all 4 heads as in training. The hyperparameters we used are shown in Table 7.

task	lr	dec	warmup	scheduler
CoLA	8E-4	0	no	cosine
RTE	7E-3	0.5	yes	cosine

Table 7: Hyperparameters for Tiny-Attn-4H without parameter-averaging.

Does the size of PLM matter? The hyperparameters we used are shown in Table 6.

C Results and Analysis Details

We use "Adapter^H" and "Adapter^P" to denote the Adapter proposed in Houlsby et al. (2019) and Pfeiffer et al. (2021), respectively. "Tiny-Attn-kH" represents our method with k attention heads, e.g., 'Tiny-Attn-1H is our method with a single attention head.

C.1 GLUE and FewGLUE.

Results in this section are discussed in section 4.1.

GLUE. Table 8 shows our results on GLUE development set with roberta-large as the PLM. Note that WARP used a slightly different set of validation metrics from other methods, but based on our results we can assume that this difference does not make a significant difference in the average score, that's why we directly used its score in Figure 1. We count all the parameters updated in training for AdaMix and our Tiny-Attn-4H to compare with other methods.

We report Matthew's correlation for CoLA, the overall(matched and mismatched) accuracy and matched accuracy for MNLI, Pearson correlation for STS-B, accuracy and F1 score for QQP and MRPC, and accuracy for other tasks. [5] We used matched accuracy for MNLI and accuracy for QQP and MRPC as the validation metrics. For other tasks, we just used the metrics we report.

To make a fair comparison, we did not use a model pretrained on MNLI for MRPC, RTE, and STS-B as the fine-tuning baseline did, following the setting in Houlsby et al. (2019) and Hu et al. (2021). Under this restriction, our method still outperforms the fine-tuning baseline on average.

There are some other parameter-efficient transfer learning methods evaluated on GLUE, but they either use a different backbone, e.g. Karimi Mahabadi et al. (2021); or they are orthogonal to our work, e.g. Rücklé et al. (2021), so we do not include them in this table.

We obtained test set results on the official GLUE server. We only made one submission to the server. Following WARP, we always predict the majority class for WNLI. For other tasks, we directly used the weight with the highest score on the validation sets and did not do any extra modification as the fine-tuning baseline did. Specially, we did not use the model adapted to MNLI for RTE, MRPC and SST-B as WARP and the fine-tuning baseline did. We found that our method has a consistent performance on validation and test sets, comparing to WARP. The results are shown in Table 9. Note that the score in the table is the official GLUE score, so it is affected by the result of WNLI.

C.2 Analysis

The results in this section are discussed in section 4.2.

Sequential vs. parallel. Table 10 shows our results on SST-2 and MNLI with different structures. We found that these two design choices have negligible differences in results.

Effects of parameter-averaging. We found that using our parameter-averaging trick actually slightly improved the results, as shown in Table 11.

[5] The implementation of these metrics can be found at https://github.com/huggingface/datasets/blob/master/metrics/glue/glue.py.

method	SST-2	CoLA	MNLI	QQP	RTE	MRPC	QNLI	STS-B	average	# trainable para	percentage
metric	Acc	Mat	All/M	Acc/F1	Acc	Acc/F1	Acc	Pearson			
AdaMix	97.1	70.2	90.9/–	92.3/89.8	89.2	91.9/94.1	95.4	92.4	89.9/–	2M	0.56%
Adapter^H	96.3	66.3	90.3/–	91.5/–	72.9	87.7/–	94.7	91.5	86.4/–	0.8M	0.23%
Adapter^H	96.2	66.5	89.9/–	92.1/–	83.4	88.7/–	94.7	91.0	87.8/–	6M	1.7%
Adapter^P	**96.6**	67.8	90.5/–	91.7/–	80.1	89.7/–	94.8	91.9	87.9/–	0.8M	0.23%
Adapter^P	96.1	68.3	90.2/–	91.9/–	83.8	90.2/–	94.8	92.1	88.4/–	3M	0.85%
LoRA	96.2	68.2	**90.6**/–	91.6/–	85.2	90.2/–	**94.8**	**92.6**	88.6/–	0.8M	0.23%
WARP	96.0	60.6	–/88.2	–/84.5	75.8	–/90.8	93.5	88.6	–/84.75	25K	0.007%
fine-tuning	96.4	68.0	90.2/90.2	**92.2**/–	86.6	90.9/–	94.7	92.4	88.9/–	355M	100%
Tiny-Attn-1H (ours)	**96.6**	68.8	89.3/89.6	90.1/86.6	88.8	92.4/94.4	94.3	92.3	89.1/88.9	176K	0.05%
Tiny-Attn-4H (ours)	**96.6**	**69.4**	89.3/89.6	90.1/86.6	**89.2**	**92.6/94.6**	94.5	92.3	**89.3/89.1**	0.7M	0.2%

Table 8: Dev set results on GLUE tasks. All the runs use roberta-large as the backbone. We report Matthew's correlation for CoLA, the overall(matched and mismatched) accuracy and matched accuracy for MNLI, Pearson correlation for STS-B, accuracy and F1 score for QQP and MRPC, and accuracy for other tasks. Higher is better for all metrics. All the runs follow the setting in Houlsby et al. (2019) except fine-tuning. The results of WARP and AdaMix are taken from their own paper respectively. Other results are taken from Hu et al. (2021).

method	SST-2	CoLA	MNLI	QQP	RTE	MRPC	QNLI	STS-B	score	# trainable para	backbone model
metric	Acc	Mat	M/MM	Acc/F1	Acc	Acc/F1	Acc	Pearson/Spearmanr			
human baselines	97.8	66.4	92.0/92.8	59.5/80.4	93.6	86.3/80.8	91.2	92.7/92.6	87.1	-	-
fine-tuning	97.5	71.5	91.9/91.6	76.2/90.8	93.2	94.0/92.0	99.2	92.9/92.6	90.8	1.5B	DeBERTa
fine-tuning	96.7	67.8	90.8/90.2	74.3/90.2	88.2	92.3/89.8	95.4	92.2/91.9	88.1	355M	roberta-large
fine-tuning	94.9	60.5	86.7/85.9	72.1/89.3	70.1	89.3/85.4	92.7	87.6/86.5	80.5	345M	bert-large
WARP	96.3	53.9	88.0/88.2	68.6/87.7	84.3	88.2/83.9	93.5	89.5/88.8	81.6	25K	roberta-large
Tiny-Attn-1H (ours)	96.2	62.5	89.3/88.8	71.8/89.1	83.6	90.7/87.5	94.4	91.1/90.3	83.5	176K	roberta-large

Table 9: Test set results on GLUE tasks. We use the same setting as on the dev set, different from the fine-tuning baseline and WARP. The results other than ours are published in Hambardzumyan et al. (2021). We don't show the result of WNLI but it's considered in the final score.

method	SST-2	MNLI
sequential	94.5	86.1
parallel	94.4	86.1

Table 10: Performance of Tiny-Attn-1H with different structures on SST-2 and MNLI. We report accuracy for SST-2 and matched accuracy for MNLI.

parameter-averaging	CoLA	RTE
no	68.8	88.8
yes	69.4	89.2

Table 11: Results of Tiny-Attn-4H with different parameter-averaging settings on CoLA and RTE. We report Matthew's correlation for CoLA and accuracy for RTE.

Effects of larger dimensions We found that Tiny-Attn-1H4D is slightly worse than the standard version Tiny-Attn-1H, as shown in Table 12.

method	CoLA	RTE
Tiny-Attn-1H4D	68.0	87.4
Tiny-Attn-1H	68.8	88.8

Table 12: Results of Tiny-Attn-4H with different parameter-averaging settings on CoLA and RTE. We report Matthew's correlation for CoLA and accuracy for RTE.

Stability test We report the best result with a fixed random seed for our main experiments. To test the stability of our method, we run experiments on SST-2 and MNLI using the same hyperparameters with 5 different random seeds. The results are shown in Table 13. We could see that the results on larger datasets like MNLI are quite stable, while the results on SST-2 have a larger variance.

	SST-2	MNLI
seed-1	96.6	89.6
seed-2	96.6	89.5
seed-3	95.8	89.6
seed-4	96.3	89.8
seed-5	95.2	89.6
avg	96.1	89.6
stdev	0.6	0.1

Table 13: Stability test. We report accuracy for SST-2 and matched accuracy for MNLI. Seed-1 is the seed we used in the main experiments.

Generation results There has been research showing that parameter-efficient transfer methods with good classification performance may not work equally well for non-classification tasks, and vice versa (He et al., 2021). We conducted experiments on E2E NLG Challenge (Novikova et al., 2017b)

and found that our method also suffers from this problem. The results are shown in Table 14. Our result is comparable to adapter but worse than prefix-tuning.

method	dev	test	# trainable para
Tiny-Attn	71.4	63.8	0.1M
Adapter	68.1	66.3	0.36M
Prefix-Tuning	74.8	70.3	0.36M

Table 14: Results on E2E NLG Challenge benchmark with gpt2-medium as the PLM. We report the BLEU score computed by the official evaluation script.