$\mathcal{S}$parse$\mathcal{A}$dapter: An Easy Approach for Improving the Parameter-Efficiency of Adapters

Shwai He[1,4*] **Liang Ding**[1†] **Daize Dong**[4] **Miao Zhang**[2] **Dacheng Tao**[1,3]
[1]JD Explore Academy
[2]Aalborg University [3]The university of Sydney
[4]University of Electronic Science and Technology of China
shwai.he@gmail.com, dingliang1@jd.com, dzdong2019@gmail.com,
miaoz@cs.aau.dk, dacheng.tao@gmail.com

Abstract

Adapter Tuning, which freezes the pretrained language models (PLMs) and only fine-tunes a few extra modules, has become an appealing efficient alternative to the full model fine-tuning. Although computationally efficient, the recent adapters often increase parameters (e.g. bottleneck dimension) for matching the performance of full model fine-tuning, which we argue goes against their original intention. In this work, we re-examine the parameter-efficiency of adapters through the lens of network pruning (we name such plug-in concept as `SparseAdapter`) and find that SparseAdapter can achieve comparable or better performance than standard adapters when the sparse ratio reaches up to 80%. Based on our findings, we introduce an easy but effective setting "*Large-Sparse*" to improve the model capacity of adapters under the same parameter budget. Experiments on five competitive adapters upon three advanced PLMs show that with proper sparse method (e.g. SNIP) and ratio (e.g. 40%) SparseAdapter can consistently outperform their corresponding counterpart. Encouragingly, with the *Large-Sparse* setting, we can obtain further appealing gains, even outperforming the full fine-tuning by a large margin. Our code will be released at: https://github.com/Shwai-He/SparseAdapter.

1 Introduction

The "pretrain-finetune" paradigm has become the *de facto* standard for the community of natural language processing (NLP) (Devlin et al., 2019; Liu et al., 2019). Given a pretrained language model (PLM), the conventional fine-tuning manner is tuning the entire parameters, i.e., full fine-tuning, for each downstream task (Devlin et al., 2019). Considering the ever-increasing size of PLMs (Brown et al., 2020), full fine-tuning has become prohibitively expensive, limiting the applicability of PLMs to a broader range of tasks. Hence, various parameter-efficient fine-tuning approaches are explored (Houlsby et al., 2019; Hu et al., 2021; Zhong et al., 2022), among which *Adapter Tuning*, that only tunes the extra light-weighted modules and keeps the original PLM frozen, has attached great attention.

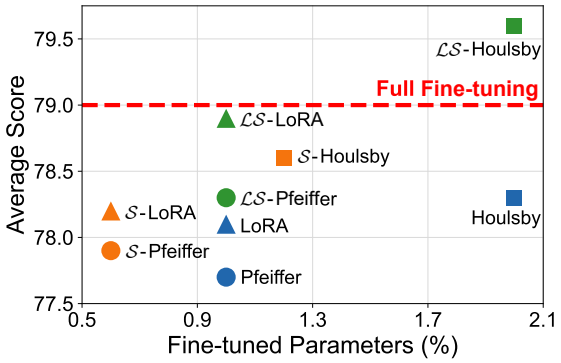


Figure 1: Performance of different parameter-efficient tuning methods on tasks from GLUE benchmark with RoBERTa-base encoder. We report the performance of Houlsby Adapters, Pfeiffer Adapters, LoRA as well as that used in our plug-in method `SparseAdapter`, where we denoted the normal sparse (in Table 1 and 4) as "$\mathcal{S}$-" and *Large-Sparse* (in Table 3) as "$\mathcal{LS}$-" in prefix.

Despite the progress, existing adapters match the performance of full fine-tuning by increasing the bottleneck dimension (Houlsby et al., 2019; Wang et al., 2022). This increases the overall parameters and FLOPs, violating the original intention of adapters. In this work, we turn to investigate the parameter-efficiency property (the *nature* of adapters) to answer the following questions: (1) Whether the current adapters can be further efficient? (2) How can we increase the representation capacity of adapters within the original parameter budget?

To this end, we examine the parameter-efficiency of adapters through the lens of network prun-

*Work was done when Shwai was interning at JD Explore Academy.
†Corresponding author

Findings of the Association for Computational Linguistics: EMNLP 2022, pages 2184–2190
December 7-11, 2022

ing (Mozer and Smolensky, 1989; Janowsky, 1989), which reduces the model size of neural networks by pruning redundant parameters and training the rest ones, therefore, improving the network efficiency. We call such pruned adapters **SparseAdapter**. Specifically, we systematically investigate five representative pruning methods in §2.2 to check at what sparse ratio can the adapters maintain the effectiveness. Note that to maintain the efficient nature of adapters, we prune all adapters at initialization such that there are no extra computational costs. We find that ❶ SparseAdapter can achieve comparable (or even better) performance than standard adapters when the sparse ratio reaches up to 80%. Such encouraging performance could hold even using the random pruning method (See Figure 2) on GLUE benchmark (Wang et al., 2018). Based on these insights, we introduce a frustratingly easy setting, namely *Large-Sparse*, for SparseAdapter. We find that ❷ Scaling up the bottleneck dimension of SparseAdapter with a correspondingly larger sparse ratio (to ensure the same parameter budget, for example, 2× dimension scaling with 50% sparse ratio) can effectively yield significant improvement by augmenting the model capacity.

We validate the concept of our proposed SparseAdapter upon five advanced adapters, i.e., Houlsby (Houlsby et al., 2019), Pfeiffer (Pfeiffer et al., 2020b), LoRA (Hu et al., 2021), MAM Adapter (He et al., 2022) and AdapterFusion (Pfeiffer et al., 2021), spanning both natural language understanding (GLUE and SQuAD) and generation (XSum) benchmarks. We show that with proper sparsity, e.g. 40%, SparseAdapter could consistently outperform their correspondingly counterpart baselines. And with our *Large-Sparse* setting, SparseAdapter could even beat the full fine-tuning method significantly, e.g. 79.6 vs. 79.0 in Figure 1.

2 Methodology

Motivation. Adapters are bottleneck modules plugged in PLMs, with bottleneck dimension r and model dimension d. In standard *Adapter Tuning*, only adapter layers are trainable while the parameters of original parameters are frozen, where the number of trainable parameters determines the capacity of adapters. The common recipe to augment the capacity is to increase the bottleneck dimension, which requires more computation cost, violating the original intention of adapters.

To check whether augmenting adapters by increasing the parameters is an optimal choice, we decide to revisit the *nature* of adapters, i.e., parameter efficiency, by pruning the redundant parameters. As shown in Figure 2, randomly pruned adapters can achieve comparable or even better performance than standard adapters, which indicates the existence of redundant parameters. The comparable performance could even be held under 80% sparsity. Such preliminary study urges us to investigate the research questions ① and ② . We decide to approach them by systematically investigating the effects of different pruning methods.

Figure 2: The comparison between randomly pruned adapters and standard adapters on datasets from GLUE.

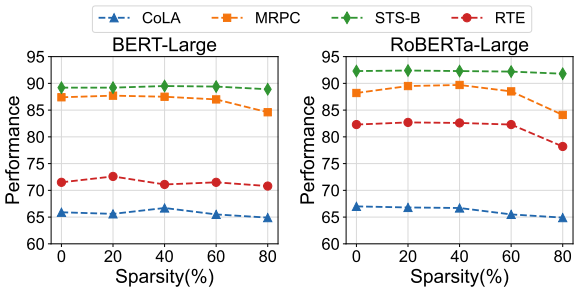
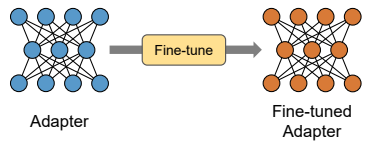
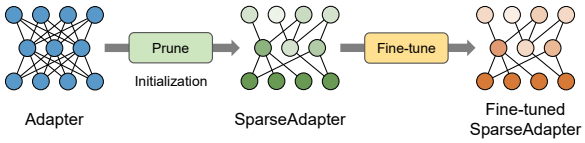


Figure 3: Schematic comparison of (a) standard adapter and (b) our proposed SparseAdapter.



(a) Standard Adapter Tuning.



(b) SparseAdapter Tuning.

2.1 Pruning Adapters at Initialization

As is shown in Figure 3, we intend to prune out redundant parameters and then fine-tune the SparseAdapter, instead of directly tuning all parameters (standard *Adapter Tuning*). By pruning adapters at initialization, we can abandon the redundant parameters at the early stage and avoid the time-consuming iterative pruning process (Frankle and Carbin, 2018). Specifically, considering an adapter with weights w^l inserted in the layer

$l \in \{1, \cdots, L\}$, parameters can be pruned by a binary mask m^l as $\tilde{w}_i^l = w_i^l \odot m_i^l$, where $\tilde{w}_i^l$ denotes the pruned parameters, w_i^l and m_i^l denote the i-th element of w^l and m^l, respectively. Given the target sparsity s, we assign scores z to all parameters w and then remove redundant parameters whose scores are below the threshold z_s (the s-th lowest percentile of z). The pruning process is shown in Algorithm 1.

Algorithm 1: Pruning on Adapters

Require: adapter paramters w, sparse ratio s

1: $w \leftarrow \text{Initialization}(w)$
2: $z = \text{score}(w)$
3: Compute the s-th percentile of z as z_s
4: $m \leftarrow \mathbb{1}\left[z - z_s \geq 0\right]$
5: $\tilde{w} \leftarrow m \odot w$

2.2 Pruning Methods

Random. Random pruning assigns a random score $z \sim \text{Uniform}(0, 1)$ to each parameter and removes parameters with the lowest scores.

Magnitude. Magnitude pruning assigns each parameter with its magnitude $z = |w|$ as its score and removes parameters with the lowest scores. Magnitude pruning is a standard way to prune during (or after) training (Janowsky, 1989; Han et al., 2015). Here we follow Frankle et al. (2020) to employ magnitude pruning at the initialization stage.

Erdős-Rényi (ER). Mocanu et al. (2018); Evci et al. (2020) specify each layer with a random topology in which larger layers are allocated with higher sparsity than smaller layers. The layer-wise sparsity is scaled proportional to $1 - \frac{n_{in}+n_{out}}{n_{in} \cdot n_{out}}$, where n_{in} and n_{out} refers to the number of input and output neurons, respectively.

SNIP. Lee et al. (2018) compute the gradients g_l for each layer with sampled mini-batch of training data, assign scores $z_l = -w_l \odot g_l$, and remove the weights with the highest scores in one iteration. The method prunes the weights with the lowest "effect on the loss" (either positive or negative).

GraSP. Wang et al. (2020) compute the Hessian-gradient product h_l for each layer, issue scores $z_l = -w_l \odot h_l$, and remove the weights with the highest scores in one iteration. The method removes weights that "reduce gradient flow" while preserving weights that "increase gradient flow".

3 Experiments

Setup. Experiments were conducted on three widely-used benchmarks, spanning understanding and generation tasks: (1) GLUE (Wang et al., 2018), containing understanding tasks like natural language inference, sentiment analysis, and sentence similarity evaluation; (2) XSum (Narayan et al., 2018), a summarization dataset where the models are required to generate a short summary for a given article; (3) SQuAD v1.1 (Rajpurkar et al., 2016), a pair-wise dataset for questions and Wikipedia paragraphs where models select the answer span to the question from the paragraph.

We use Adam (Kingma and Ba, 2014) as the optimizer with β_1, β_2 = 0.9, 0.98. For regularization, we set the weight decay as 0.1 and grid-search the learning rate from {1e-5, 2e-5, 5e-5, 1e-4, 2e-4}, where we warm up the learning rate in the first 10% steps (of the total training steps). For different data scales, we grid-search the training epoch and batch size from {5, 10, 15, 20}, and {8, 16, 32, 64}, respectively. The maximum length is 512 for GLUE and 384 for SQuAD. For XSum, we set the max length of source articles to be 512 and the max length of the target summary to be 128. For the GLUE benchmark, we follow previous works (Phang et al., 2018; Lee et al., 2020; Dodge et al., 2020) to fine-tune the pretrained language models, e.g. BERT (Devlin et al., 2019) and RoBERTa (Liu et al., 2019), on the downstream training set and report results on the dev set using the last checkpoint. For the other tasks, we report the test results.

3.1 Results

SparseAdapters with Different Pruning Methods. In Table 1, we carefully compare SparseAdapters (with aforementioned pruning methods: "Rand.", "Mag.", "ER", "SNIP", "GraSP") to the standard adapter (Houlsby et al., 2019) ("Adapter") on GLUE benchmark for two backbone pretrained language models BERT (Devlin et al., 2019) and RoBERTa (Liu et al., 2019), where we set the bottleneck dimension to 64 for all adapter layers. As shown in Table 1, all SparseAdapters achieve comparable or even better performance compared to Houlsby Adapter (Houlsby et al., 2019) with lower computational overhead. Notably, SNIP (Lee et al., 2018) based SparseAdapter could achieve up to 0.6% average improvement compared to standard

Table 1: **Experimental results of different SparseAdapters** on GLUE benchmark, where we perform pruning with the same sparsity ratio 40% for a fair comparison. CoLA is evaluated using Matthew's correlation. STS-B is evaluated using Pearson's correlation coefficient. MRPC and RTE are evaluated using accuracy. Average scores on all tasks are underlined. The best results are **bold**. We report the results of full fine-tuning "Fine-Tune" as reference.

Method	#Param.	BERT					RoBERTa				
	(Trained)	CoLA	MRPC	STS-B	RTE	Avg.	CoLA	MRPC	STS-B	RTE	Avg.
Fine-Tune	100%	59.4	83.1	87.2	68.3	74.5	61.8	88.0	90.8	75.2	79.0
Adapter	2.0%	59.1	82.1	86.6	66.5	73.6	61.3	87.4	90.4	74.1	78.3
w/ Rand.	1.2%	58.4	**82.9**	86.7	66.8	73.7	61.0	87.5	90.5	73.2	78.1
w/ Mag.		58.2	82.8	86.7	66.3	73.2	60.6	87.0	90.6	73.3	77.9
w/ ER		58.6	82.2	86.8	67.0	73.7	60.9	87.2	90.2	73.6	78.0
w/ SNIP		**59.4**	82.3	**87.0**	**68.2**	**74.2**	**61.4**	**87.6**	90.3	**75.0**	**78.6**
w/ GraSP		59.0	82.7	86.9	67.2	74.0	61.2	87.1	**90.7**	74.4	78.4

adapter and nearly reach the performance of full fine-tuning, which is therefore left as the default setting in the following experiments.

Table 2: **Effect on different sparse ratios and different tasks.** Xsum and SQuAD are evaluated with ROUGE-2 and F1 score, respectively. We denote SparseAdapter with their sparse ratios.

Method	GLUE		XSum		SQuAD	
	#Para.	Avg.	#Para.	R2	#Para.	F1
Fine-Tune	100%	79.0	100%	21.9	100%	87.8
Adapter	2.0%	78.3	4.5%	21.6	8.8%	87.4
$s = 0.2$	1.6%	**78.7**	3.6%	21.6	7.0%	87.5
$s = 0.4$	1.2%	78.6	2.7%	**21.8**	5.3%	**87.7**
$s = 0.6$	0.8%	78.2	1.8%	21.5	3.5%	87.4
$s = 0.8$	0.4%	77.9	0.9%	21.3	1.8%	87.0

Effect on Different Downstream Tasks. Utilizing the proper sparse method, i.e., SNIP with 40% sparse ratio, we validate SparseAdapter on more downstream tasks, including GLUE, XSum, and SQuAD in Table 2. We use RoBERTa-base (Liu et al., 2019) for GLUE (Wang et al., 2018), BART-large (Lewis et al., 2020) for Xsum (Narayan et al., 2018) and BERT-base (Devlin et al., 2019) for SQuAD v1.1 (Rajpurkar et al., 2016). For XSum and SQuAD, The bottleneck dimension is set to 512 and 256 respectively to match the performance of full fine-tuning. Clearly, SparseAdapter outperforms the standard adapters in three tasks, showing the universality of SparseAdapter.

Effect on Different Sparse Ratios. In Table 2, we investigate the effect of different sparse ratios for SparseAdapter (Pfeiffer et al., 2021). We use BERT-base (Devlin et al., 2019) and RoBERTa-base (Liu et al., 2019) as backbones. SparseAdapters outperform the standard adapters when $s \leq 40\%$ and maintained stable performance while increasing the sparse ratio. Considering the trade-off between performance and parameters, we set 40% as the default sparse ratio in our work.

Effect on Different Adapter Variants. Since SparseAdapter can be plugged into any adapter variants, we further validate its effectiveness on other four variants besides Houlsby Apdaters in the above experiments, including Pfeiffer (Pfeiffer et al., 2020a), LoRA (Hu et al., 2021), Mix-And-Match Adapters ("MAM") (He et al., 2022), and AdapterFusion ("AF") (Pfeiffer et al., 2021). We choose RoBERTa-base (Liu et al., 2019) as the backbone. Following previous experiments on GLUE benchmark for MAM Adapters (He et al., 2022), we divide the trainable parameters equally into adapters in feed-forward layers and Prefix-Tuning (Li and Liang, 2021) in attention layers. Our SparseAdpater could consistently improve the accuracy with 40% fewer training parameters, showing the generalization of our plug-in method. Experimental results are listed in Table 4.

Augmenting SparseAdapter with *Large-Sparse* Setting. One strength of SparseAdapter is the potential to exploit large adapter (with a correspondingly large sparse ratio) to augment the adapter capacity under the same parameter budget, namely *Large-Sparse* setting. To validate our claim, we scale the bottleneck dimension by $\{2\times, 3\times, 4\times\}$ with correspondingly {50%, 67%, 75%} sparse ratios. As shown in Table 3, while maintaining the same amount of parameters, with bottleneck dimen-

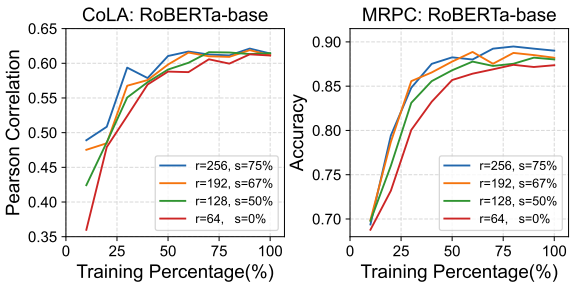
Table 3: **Experimental results of scaling the bottleneck dimension.** $(2\times, 3\times, 4\times)$ **of SparseAdapters using the same amount of parameters**, coined as ***Large-Sparse*** setting ("$\mathcal{LS}$-" in the prefix), on GLUE benchmark. We correspondingly increase the sparsity to ensure the same number of parameters for SparseAdapters with larger bottleneck dimensions.

Method	Setting		BERT					RoBERTa				
	r	s	CoLA	MRPC	STS-B	RTE	Avg.	CoLA	MRPC	STS-B	RTE	Avg.
Adapter	64	0%	59.1	82.1	86.6	66.5	73.6	61.3	87.4	90.4	74.1	78.3
$\mathcal{LS}$-Adapter	128	50%	59.9	82.3	87.6	67.5	74.3	61.7	88.2	90.3	75.5	78.9
	192	67%	60.1	82.7	87.7	67.7	74.6	61.8	88.7	90.4	75.3	79.1
	256	75%	**60.6**	**83.3**	**88.2**	**68.2**	**75.1**	**62.1**	**89.5**	**90.5**	**76.2**	**79.6**

Table 4: **Effects on other different adapter variants**. "$\mathcal{S}$-" means equipped with our SparseAdatper.

Method	CoLA	MRPC	STS-B	RTE	Avg.
Pfeiffer	**61.2**	85.8	89.2	74.7	77.7
$\mathcal{S}$-Pfeiffer	61.1	**86.0**	**89.3**	**75.2**	**77.9**
LoRA	62.0	87.5	88.5	74.5	78.1
$\mathcal{S}$-LoRA	**62.1**	**87.7**	**88.8**	**74.6**	**78.2**
MAM	61.3	86.5	89.7	**74.6**	78.0
$\mathcal{S}$-MAM	**61.5**	**87.6**	**89.8**	74.3	**78.3**
AF	63.1	89.7	**90.9**	76.0	79.9
$\mathcal{S}$-AF	**63.3**	**90.0**	90.8	**76.4**	**80.1**

Figure 4: The comparison between SparseAdapters with *Large-Sparse* setting and standard adapters.



sion increases, *Large-Sparse* could consistently gain better performance, achieving up to +1.3% and +0.6% average improvements against the standard adapter and full fine-tuning, respectively.

Besides the encouraging performance, we compare SparseAdapters with *Large-Sparse* setting to standard adapters on the training convergence speed in Figure 4. SparseAdapters maintain a performance advantage at the same training percentage and converge at least 25% ahead in the training process. For both tasks, *Large-Sparse* setting contributes to a faster convergence rate and higher performance.

4 Conclusion

In this work, we systematically reexamine the parameter efficiency property of adapter Tuning through the lens of network pruning. Based on our findings, we propose a plug-in strategy, i.e., SparseAdapter, for existing adapters. Our study empirically indicates the potential to make SparseAdapter (especially with the *Large-Sparse* setting) a golden standard efficient transfer learning strategy for the NLP community.

The future work includes applying our proposed SparseAdapter to more tasks (e.g. multilingual PLM based machine translation (Zan et al., 2022a,b)) and benchmarks, and investigating the parameter efficiency of of other neural network models, especially for scenarios where high efficiency is required, e.g. Prompt (Lester et al., 2021).

Acknowledgements

We are grateful to the anonymous EMNLP reviewers and the area chair for their insightful comments and suggestions.

Limitations

Despite the progress we made, there still exist limitations in our work. On the one hand, we only investigated some classic pruning methods and found that SNIP (Lee et al., 2018) performs the best in selected criteria. However, there may exist other advanced pruning methods that can further improve the performance, which deserves exploration in future work. On the other hand, since we only consider BERT, RoBERTa, and Bart in limited tasks, it would be valuable to consider other architecture families (e.g. XLNET (Yang et al., 2019), ELECTRA (Clark et al., 2020)) and tasks (e.g. machine translation).

References

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc.

Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. 2020. ELECTRA: Pre-training text encoders as discriminators rather than generators. In *ICLR*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 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.

Jesse Dodge, Gabriel Ilharco, Roy Schwartz, Ali Farhadi, Hannaneh Hajishirzi, and Noah Smith. 2020. Fine-tuning pretrained language models: Weight initializations, data orders, and early stopping. *arXiv preprint arXiv:2002.06305*.

Utku Evci, Trevor Gale, Jacob Menick, Pablo Samuel Castro, and Erich Elsen. 2020. Rigging the lottery: Making all tickets winners. In *International Conference on Machine Learning*, pages 2943–2952. PMLR.

Jonathan Frankle and Michael Carbin. 2018. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*.

Jonathan Frankle, Gintare Karolina Dziugaite, Daniel M Roy, and Michael Carbin. 2020. Pruning neural networks at initialization: Why are we missing the mark? *arXiv preprint arXiv:2009.08576*.

Song Han, Jeff Pool, John Tran, and William Dally. 2015. Learning both weights and connections for efficient neural network. *Advances in neural information processing systems*, 28.

Junxian He, Chunting Zhou, Xuezhe Ma, Taylor Berg-Kirkpatrick, and Graham Neubig. 2022. Towards a unified view of parameter-efficient transfer learning. In *International Conference on Learning Representations*.

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 *International Conference on Machine Learning*, pages 2790–2799. PMLR.

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*.

Steven A Janowsky. 1989. Pruning versus clipping in neural networks. *Physical Review A*, 39(12):6600.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Cheolhyoung Lee, Kyunghyun Cho, and Wanmo Kang. 2020. Mixout: Effective regularization to finetune large-scale pretrained language models. *ICLR*.

Namhoon Lee, Thalaiyasingam Ajanthan, and Philip HS Torr. 2018. Snip: Single-shot network pruning based on connection sensitivity. *arXiv preprint arXiv:1810.02340*.

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*, pages 3045–3059, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7871–7880, Online. Association for Computational Linguistics.

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)*, pages 4582–4597, Online. Association for Computational Linguistics.

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*.

Decebal Constantin Mocanu, Elena Mocanu, Peter Stone, Phuong H Nguyen, Madeleine Gibescu, and Antonio Liotta. 2018. Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science. *Nature communications*, 9(1):1–12.

Michael C Mozer and Paul Smolensky. 1989. Using relevance to reduce network size automatically. *Connection Science*, 1(1):3–16.

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. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1797–1807, Brussels, Belgium. Association for Computational Linguistics.

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*, pages 487–503, Online. Association for Computational Linguistics.

Jonas Pfeiffer, Andreas Rücklé, Clifton Poth, Aishwarya Kamath, Ivan Vulić, Sebastian Ruder, Kyunghyun Cho, and Iryna Gurevych. 2020a. Adapterhub: A framework for adapting transformers. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 46–54.

Jonas Pfeiffer, Ivan Vulić, Iryna Gurevych, and Sebastian Ruder. 2020b. MAD-X: An Adapter-Based Framework for Multi-Task Cross-Lingual Transfer. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7654–7673, Online. Association for Computational Linguistics.

Jason Phang, Thibault Févry, and Samuel R Bowman. 2018. Sentence encoders on stilts: Supplementary training on intermediate labeled-data tasks. *arXiv preprint arXiv:1811.01088*.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392, Austin, Texas. Association for Computational Linguistics.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. 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.

Chaoqi Wang, Guodong Zhang, and Roger Grosse. 2020. Picking winning tickets before training by preserving gradient flow. *arXiv preprint arXiv:2002.07376*.

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*.

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*, 32.

Changtong Zan, Liang Ding, Li Shen, Yu Cao, Weifeng Liu, and Dacheng Tao. 2022a. On the complementarity between pre-training and random-initialization for resource-rich machine translation. In *COLING*.

Changtong Zan, Keqin Peng, Liang Ding, Baopu Qiu, Boan Liu, Shwai He, Qingyu Lu, Zheng Zhang, Chuang Liu, Weifeng Liu, et al. 2022b. Vega-mt: The jd explore academy translation system for wmt22. *arXiv preprint*.

Qihuang Zhong, Liang Ding, Juhua Liu, Bo Du, and Dacheng Tao. 2022. Panda: Prompt transfer meets knowledge distillation for efficient model adaptation. *ArXiv*, abs/2208.10160.