Pruning Adatperfusion with Lottery Ticket Hypothesis

Anonymous ACL submission

Abstract

Pre-trained language models have shown great success in multiple downstream tasks. However, they are computationally expensive to fine-tune. Thus, transfer learning with adapter modules has been introduced to alleviate this problem, helping to extract knowledge of the downstream tasks. And the latest Adapterfusion model can further merge multiple adapters to incorporate knowledge from different tasks. However, merging multiple adapters will inevitably cause redundancies, increasing the training and inference time massively. Therefore, in this paper, we propose an approach to identify the influence of each adapter module and a novel way to prune adapters based on the prestigious Lottery Ticket Hypothesis. Experiments on GLUE datasets show that the pruned Adapterfusion model with our scheme can achieve state-of-the-art results, reducing sizes significantly while keeping performance intact.

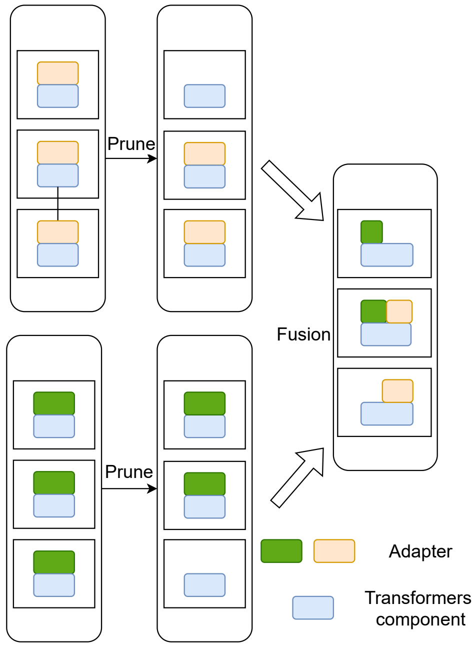


Figure 1: Pruning Adapterfusion

1 Introduction

Transfer learning with transformer-based pre-trained language model has become a go-to method for solving multiple NLP tasks (Vaswani et al., 2017). The language models are pre-trained on large amounts of unlabeled text data with methods such as masked language modeling (e.g. BERT (Devlin et al., 2018), Roberta (Liu et al., 2019), and XLNet (Yang et al., 2019)]). Despite they achieved state-of-the-art performance for most natural language understanding tasks, they are notoriously deep requiring millions or even billions of parameters to gain great results (Kaplan et al., 2020). For different tasks, models needs to be fine-tuned entirely, which is computationally expensive and requires large storage.

Therefore, adapter modules (Houlsby et al., 2019) are introduced to tackle this issue. It's an alternative way of transfer learning that achieves comparable performance to full fine-tuning on most NLP tasks, without the need of fine-tuning the whole model for a downstream task. Adapter is a small residual neural network inserted in each layer of the transformer. During training, only the parameters in adapters are fine-tuned, while the rest of the parameters are frozen. This approach can reduce the number of parameters needed to be trained at the training phase, extract task-specific knowledge in adapters, and enable parameter sharing among tasks. Moreover, recent studies have revealed that the adapter is capable of extracting knowledge from the target task (Rücklé et al., 2020b; Pfeiffer et al., 2020b), so research attempts have also been made to fuse multiple adapters across multiple tasks to incorporate different aspects of knowledge (e.g. Adapterfusion (Pfeiffer et al., 2020a), K-adapter Wang et al. (2020)). However, the fusing of adapters in these models can inevitably cause a lot of redundancies. So, Rücklé et al. (2020a) have recently proposed AdapterDrop which aims to drop the redundant adapters. They tried to remove adapters from lower transformer layers during training and inferences, resulting in faster training and inference speed with some performance cost. However, the utilization of each adapter are not fully analysed yet and how to introduce new pruning strategies remains to be explored.

To address these deficits, in this paper, we propose an approach to model the utilization of different adapters in the transformer layer, and a novel way to prune adapters in the model while keeping the loss of the performance to be negligible. The contributions are summarized as follows:

- We propose a new indicator LIA (Layer Influence Of Adapter) to quantify the utilization of adapters at each layer and identify the most influential adapters in the model.
- We introduce a novel way for pruning adapter modules, inspired by the prestigious Lottery Ticket Hypothesis (Frankle and Carbin, 2019), which states that dense, randomly-initialized, feed-forward networks contain subnetworks (winning tickets) that can have test accuracy comparable to the original network in a similar number of iterations when trained in isolation.
- We have evaluated the proposed approach on the GLUE datasets. For the performance and LIAs of the pruned adapters in the latest state-of-the-art Adapterfusion model, we can remove more than half of the adapters and reduce computation of the Adapterfusion model by nearly 40% with little performance loss.

2 Adapterfusion pruning

Adapterfusion model (Pfeiffer et al., 2020a) is to merge multiple adapters from different tasks. And inference time of the model increases drastically after the fusion. However, not all the adapter modules in the model are utilized in the downstream task. To identify the roles of adapters, we firstly define a new indicator LIA (Layer Influence Of Adapter) for the adapter module to measure its utilization in each layer.

In a transformer layer, let $\vec{a}$ denote the output of adapter up projection module, $\vec{b}$ be the residual connection of the adapter, and $\vec{c}$ be the output of the adapter. Their connection can be modeled as equation 1:

$$\vec{a} + \vec{b} = \vec{c} \tag{1}$$

Since the activation of adapters varies between different inputs and different layers, it's difficult to see the influence of adapter in each layer based on the activation. So we use the projection of $\vec{a}$ on adapter output vector $\vec{c}$ to represent its influence and normalize the result by the length of $\vec{c}$. We name this quantity as Layer Influence Of Adapter (LIA), which is defined as:

$$LIA_{\vec{a}} = \frac{|\vec{a}| \times \cos\theta}{|\vec{c}|} = \frac{\vec{a} \cdot \vec{c}}{|\vec{c}|^2} \tag{2}$$

With LIA, we can model the importance of adapters and streamline the model accordingly. Our proposed pruning strategies to remove redundant adapters from Adapterfusion model will be two-stage. First, we prune single task adapters before the fusion using Lottery Ticket Hypothesis (Frankle and Carbin, 2019). We then fuse the adapters and prune the less utilized adapters after training the model. The whole framework is presented in Figure 1.

2.1 Pruning single task adapter with Lottery Ticket Hypothesis

Since not all adapters in the model are created equal, removing some of the adapters does not compromise the performance too much. We will prune the single task adapter before the Adapterfusion.

Inspired by Lottery Ticket Hypothesis (Frankle and Carbin, 2019), we prune the adapter iteratively to find the sub-network (winning ticket) that can reach the same accuracy when trained in isolation. After every pruning, we reinitialize the weights of the adapter to the initial values when the first iteration starts .

We explore to find the winning ticket in adapters by training and pruning them iteratively. Since the importance of adapters is different in each layer, we are performing the pruning globally. We train the transformer model with adapters as $f(x; \theta_0; \alpha)$ with initial parameter in adapters $\theta = \theta_0 \sim D_\theta$ and transformer parameter $\alpha = \alpha \sim D_\alpha$. Then the winning ticket can be found by the following steps:

1. Randomly initialize adapter parameters in the model $f(x; \theta_0; \alpha)$.
2. Train the adapters for j iterations, arriving at parameters θ_j.

3. Prune $p\%$ of the adapters in θ_j.

4. Reset the remaining parameters in adapters to θ_0, and go back step 2 to train the model $f(x; \theta_0; \alpha)$ if it is not a winning ticket yet.

We prune adapters based on their sum of weights. Here we do not use LIA yet because it only represents the influence of adapter at each layer and it can not distinguish the influence between layers.

Let $\theta_{t,l}$ be the weights of the adapter at layer l at iteration t and $a_{i,j}$ denote the parameters in $\theta_{t,l}$. The importance of an adapter of size N with input size of H is $\sum_{i,j}^{N,H} |a_{i,j}|$. Adapters are then sorted by the sum of weights in the descending order as well, and the $p\%$ smallest adapters in list R are removed from the model. And the remaining adapters step back to their initial weights for the re-training. The whole procedure is elaborated in Algorithm 1. See Appendix B for more details

Algorithm 1: Sort the importance of adapter layers

Result: a list of tuple containing values of importance and the number of layers

R is an empty list;
The size of adapter is N ;
Input size of adapter is H ;
Weights of adapter at iteration t as θ_t;
for *layer l in θ_t* **do**
    if *layer l not pruned* **then**
        Value of importance $Imp_l = \sum_{i,j=0}^{N,H} |a_{i,j}|$;
        Append tuple (Imp_l, l) to list R ;
    end
end
Sort list R with Imp

3 Experimental studies

In this section, we examine the influence of the single task adapter under different downstream tasks, and present the results of our proposed pruning scheme evaluated on the prestigious GLUE datasets (Wang et al., 2018).

3.1 Experimental settings

We use the public BERT-Based uncased model which has 12 layers and a total of 110M parameters as our base model. And we apply the similar approach in (Devlin et al., 2019) to perform a text classification task. In each input sequence, the first token is a classification token. Its embedding is then fed into a linear layer to make a prediction. In the training of Adapterfusion, a new linear layer is initialized for classification and Adapterfusion model is inserted in each transformer layer.

In the experiment of pruning single task adapters, we set the adapter size to 128 because engineering practices (Bengio et al., 2005) suggest that overparameterized networks are easier to train. We use Adam optimizer to train the single task adapter model and perform hyperparameter search using TPE algorithm (Bergstra et al., 2011). We runs 30 trials on learning rate settings in $\{1 \times 10^{-4}, 5 \times 10^{-4}, 1 \times 10^{-3}\}$, and number of epochs in $\{3, 4, 5\}$. We select the best settings for pruning experiments. We prune 20% of the adapters from the model at each pruning iteration, and use an early-stopping strategy with patient of three to speed up training, and we use the minimum validation loss for early-stopping criterion.

In the experiment of pruning Adapterfusion model, we set the learning rate to 5×10^{-5} and use AdamW optimizer as suggested by Pfeiffer et al. (2020a). We run each task with 4 epochs and set the batch size to 32, and each model for each task with five different random seeds.

We have evaluated the single adapter with pruning, and Adapterfusion on GLUE datasets (Wang et al., 2018), which contain eight sentence or sentence-pair language understanding tasks.[1] And we treat MNLI $_{\text{mm}}$ and MNLI $_{\text{m}}$ equally. And we have reported the test results of the single task adapter through the GLUE submission website.[2]

3.2 LIAs in single task adapter

In order to analyse the influence of single task adapter at each layer, we run a test on the standard adapters and analyse the utilization of each of them.

In the evaluation step, we store the residual output and the output of each adapter to calculate the LIAs at each step. We then average LIAs of each adapter across the datasets. And we run the test from tasks with small datasets to large datasets, whose results are shown in Figure 2.

We have found that as the size of dataset gets bigger, LIAs of adapters also become larger, implying

[1] We omit WNLI because it is not evaluated in BERT (Devlin et al., 2018).

[2] https://gluebenchmark.com

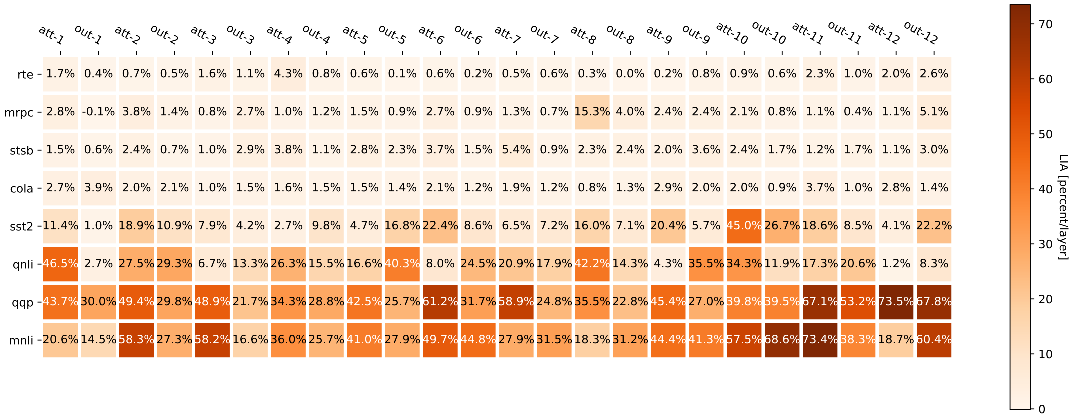


Figure 2: LIAs of each layer in single task adapter for different tasks. **att** denotes the adapter after self-attention layer, **out** is the adapter after output layer. Darker colors represent higher values of LIA. The target tasks are arranged in the order of the size of datasets from small to large.

that the size of target task dataset affects the influence of the adapters. After training on larger datasets, adapters learn more and extract more knowledge, and thus become more essential for the whole model. This could explain why in Adapterfusion (Pfeiffer et al., 2020a), using adapters from large datasets can help improve the performance of the task with small datasets.

For adapters in large dataset task (qqp, qnli, mnli), most of them have large LIAs, suggesting they are already concise and there are no many redundant parameters in them.

3.3 Pruning single task adapter

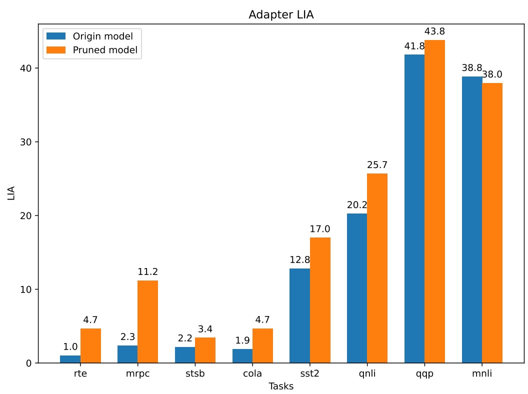


Figure 3: LIAs of adapters before and after pruning with Lottery Ticket Hypothesis

We insert adapters of size 128 into each layer of transformers in the BERT-Based model. For different text classification task, we put a task-specific classifier at the end of the model. Only the parameters in the adapters and task-specific classifier are fine-tuned, and the rest of the parameters in the model are untouched.

We iteratively prune the parameters in the adapter by 20% per iteration. We perform 11 iterations for layer pruning since there will be less than one adapter left after the 11-th iteration.

Results on GLUE test sets are presented in Table 1. We select the best result in all iterations of pruning. The best model is chosen by metrics of the corresponding task. And we evaluate the result on GLUE testing sever. We can see there is a 0.2 percentage performance gap between the full fine-tuning model and the adapter model, and there is a small performance gap between the adapter model and the pruned adapter model. Therefore, we preserve most of the essential adapters while pruning the redundant ones.

We evaluate the adapter model on GLUE development datasets after each iteration of pruning and obtain the average score of three runs, see Figure 4. We discover that there is no major performance loss before the number of adapters drops below 9 (40%). By contrast, AdapterDrop model (Rücklé et al., 2020a) removed the first 5 layers of adapters and preserve most of the performance with about 60% of the adapters left. We thus can prune adapters 20% further than their work. The speed comparison between the two models is shown in Table 2. We discover that our model is faster than AdapterDrop in most tasks.

We further analyse how the adapters are distributed when there is only nine adapters left, see Figure 5. Interestingly, we can see that most of the adapters close to the output layer are pruned. These layers are removed but no harm is done to the performance, and we think that maybe the last

	CoLA	SST-2	MRPC	STSB	QQP	MNLI	QNLI	RTE	AVG
Full fine-tuning	**52.10**	**93.50**	**88.90**	**85.80**	71.20	**84.60**	90.50	66.40	**79.60**
Adapter128	51.70	93.10	88.50	85.60	**71.50**	83.40	90.50	67.30	79.42
Prune layer	49.50	92.60	88.00	83.50	**71.50**	84.10	**90.80**	**70.60**	79.30

Table 1: Test results on GLUE test sets using GLUE server. CoLA is evaluated using Matthew's Correlation. STS-B is evaluated using Spearman's correlation coefficient. MRPC and QQP are evaluated using F1 score. The rest of the tasks are evaluated by accuracy.

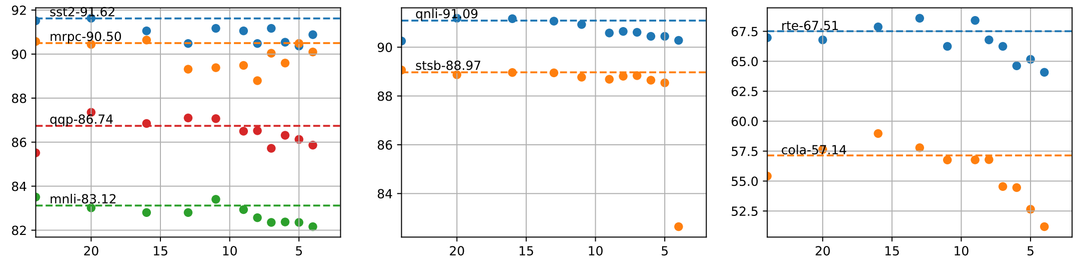


Figure 4: Performance of pruning schemes on GLUE validation sets at every pruning iteration. Horizontal line represents the performance of adapters before pruning starts. X-axis denotes the number of adapters remains, and Y-axis denotes the score in corresponding task metrics.

	Origin	AD	LTH	Speed up	
				AD	LTH
RTE	208.3	203.9	203.5	2.11%	**2.30%**
MRPC	227.6	222.7	218.3	2.15%	**4.09%**
STSB	221.6	216.9	217.8	**2.12%**	1.71%
COLA	63.5	62.1	61.7	2.20%	**2.83%**
SST2	127.5	124.7	123.9	2.20%	**2.82%**
QNLI	41.4.	40.5	40.2	2.17%	**2.90%**
QQP	85.8	83.9	83.7	2.21%	**2.45%**
MNLI	42.8	41.9	41.5	2.10%	**3.04%**

Table 2: Floating points (10^9) operation origin adapter (**Origin**), AdapterDrop (**AD**) and Pruned adapters (**LTH**) in each tasks and percentage change in speed after using AdapterDrop or Pruned adapters.

few layers are just a redundant extension of classification layer.

We also find that there are more adapters after feed-forward layers than self-attention layers, implying that adapters after feed-forward layers are more valuable. Similar phenomena can be found in the experiments of ALBERT (Lan et al., 2020), where most of the performance drop appears to come from sharing the feed-forward layer parameters, while sharing the attention layer parameters results in no performance loss.

Since different tasks require different number of adapters, in the following experiments, we use the best adapter model of all iterations of pruning in each task.

After using layer pruning with Lottery Ticket Hypothesis, the average influence of each adapter increases as shown in Figure 3. In most tasks, the LIAs increase after the pruning, which means the redundant part of the adapters are removed. And we can see a significant LIA boost, mostly in tasks of small datasets, especially in MRPC. However, in larger dataset task like MNLI, pruning the adapters causes a small decrease in LIAs, implying that most of the adapters for large datasets are playing an important role for the task. In summary, we have greatly increased the utilization of each adapter after pruning, because we deleted most of the less essential layers, and the model is thus streamlined to perform better.

3.4 Merging pruned adapters to Adapterfusion

In this experiment, we fuse eight single task adapters to construct the Adapterfusion model. An extra self-attention layer is inserted in each layer of the model to fuse the results of multiple adapters. Only the parameters in these newly inserted self-attention layers are fine-tuned, and the rest of the parameters (including adapters) remains unchanged.

We run our test on the same eight tasks of GLUE datasets as in the previous experiment. And we compare the fusion of full-size adapters and the pruned adapters with 5 runs each. And we use the best adapter model of all iterations of pruning for the fusion. Then we calculate the mean and variance of each task, whose results are in Table 3. It shows that there is not much difference between the pruned Adaterfusion model and the full-size Adapterfusion model. And there is even a mild

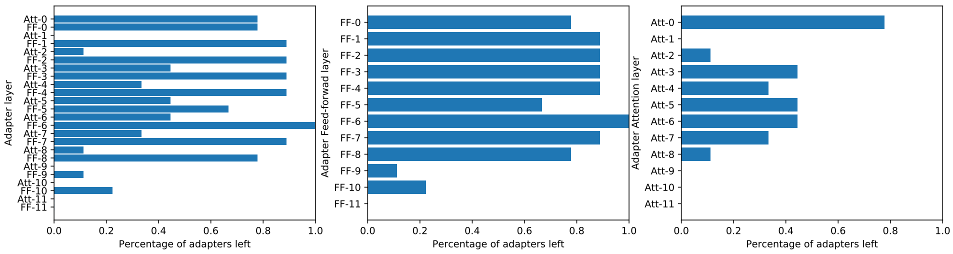


Figure 5: **Left:**Percentage of adapter remaining in each adapter layer when there is only 9 adapters left. **Center:** Adapter distribution inserted after Feed-forward layer (Extracted from the Left image).**Right:** Adapter distribution inserted after attention layer (Extracted from the Left image)

	AF	AF w. LTH
CoLA	55.16 ±1.1	**55.96** ±2.3
SST-2	91.67±0.7	**91.87**±0.4
MRPC	91.46 ±1.6	**92.16** ±1.5
STSB	**89.83**±0.33	89.27±0.64
QQP	86.74±0.39	**86.88**±0.22
MNLI	83.13±0.42	**83.16**±0.18
QNLI	**90.84**±0.21	90.73±0.33
RTE	**75.90**±3.43	73.00±6.68

Table 3: Development score of Adapterfusion and Adapterfusion pruned with Lottery Ticket Hypothesis (LTH). CoLA is evaluated using Matthew's Correlation. STS-B is evaluated using Spearman's correlation coefficient. MRPC and QQP are evaluated using F1 score. The rest of the tasks are evaluated by accuracy.

	Num of adapter	FLOPs (10^9)	Steps/sec
AF	192	554.8	9.86
AF-LTH	**89**	**332.8**	**13.86**

Table 4: FLOPs of the standard Adapterfusion and pruned Adapterfusion

	AF	AF-LTH	Saved (%)
RTE	1157.9	703.1	39.23%
MRPC	934.9	565.6	39.50%
STSB	805.9	487.4	39.52%
COLA	221.3	133.6	39.62%
SST2	513.2	310.1	39.57%
QNLI	248.4	150.3	39.49%
QQP	356.1	215.3	39.53%
MNLI	432.2	261.4	39.51%

Table 5: Floating points (10^9) operation of standard Adapterfusion and pruned Adapterfusion in each tasks

improvement of performance using the pruned one. Therefore, it justifies that the proposed pruning of adapters is very effective and we can obtain a small and dense version of the general Adapterfusion model for GLUE datasets.

The original Adapterfusion model has 192 adapters ($8tasks \times 12layer \times 2adapter/layer$), while after pruning the redundant adapters from the model, it only has 89 ones, with more than half of the adapters removed. Also layers without adapters left will not be inserted a new self-attention layer anymore for the fusion, resulting in the reduction of depth as well.

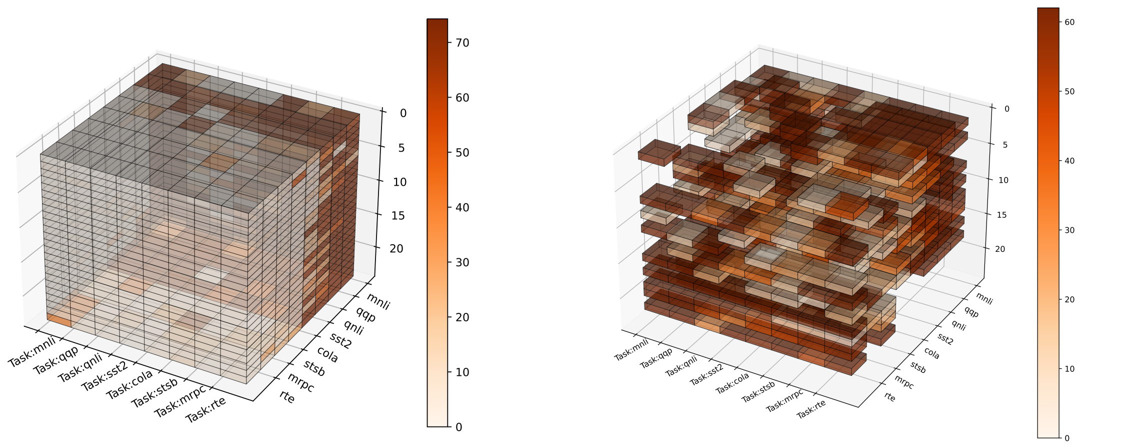
We also have measured the total number of floating point operations (FLOPs) for each task in the evaluation process. We average the FLOPs across different tasks, as shown in Table 4 and Table 5. And it reveals that after pruning, we reduce the computation by about 40%, which is a very significant improvement for the inference speed.

Moreover, we analyse the LIAs of each adapter for different target tasks. LIAs of the original Adapterfusion and the pruned Adapterfusion are shown in 3D heat map in Figure 6[3], where X-axis represents eight different target tasks for the model, y-axis is the source of each fused adapters, z-axis denotes the adapters in different layers ranging from 1 to 24, the odd number layers in z-axis represent the adapter modules inserted after attention layers, and the even number layers in z-axis are the ones inserted after the output layer.

Figure 6(a) shows the LIAs of the original Adapterfusion model. And we discover that there are a number of adapters not utilized in the original Aadapterfusion model and most of the essential adapters are at the back of the cube which are the adapters trained on large datasets. As the layers get deeper, more adapters in the model are utilized. Figure 6(b) shows the LIAs of the pruned Adapterfusion model. Compared to the original Adapterfusion model, the pruned Adapterfusion model has much fewer adapters. Furthermore, most of the adapter modules have larger LIA values, which implies that most of the adapters become more important for the task.

Then, we average the LIAs of adapters in

[3]In Appendix A, we present an in-depth visualization of the Adapterfusion with LIAs



(a) Adapterfusion

(b) Adapterfusion with LTH

Figure 6: LIAs of Adapterfusion w/o pruning

Adapterfusion across 12 layers and 5 test runs, as shown in Figure 7. We find that for the original Adapterfusion model, most of the adapters' LIAs are zeros, which means that most of them are not used in the model. And most of the tasks use the adapters trained in QQP and MNLI, both of which are tasks with large datasets. Moreover, with a larger dataset, the model will utilize more of the adapters trained from the same task.

By comparing the original Adapterfusion and the pruned Adapterfusion model, we have seen that more of the adapters are utilized after the pruning. And there are fewer zeros of LIAs in the pruned Adapterfusion model, suggesting that remaining adapters after pruning have become much more influential on average and the model are using more adapters from different tasks.

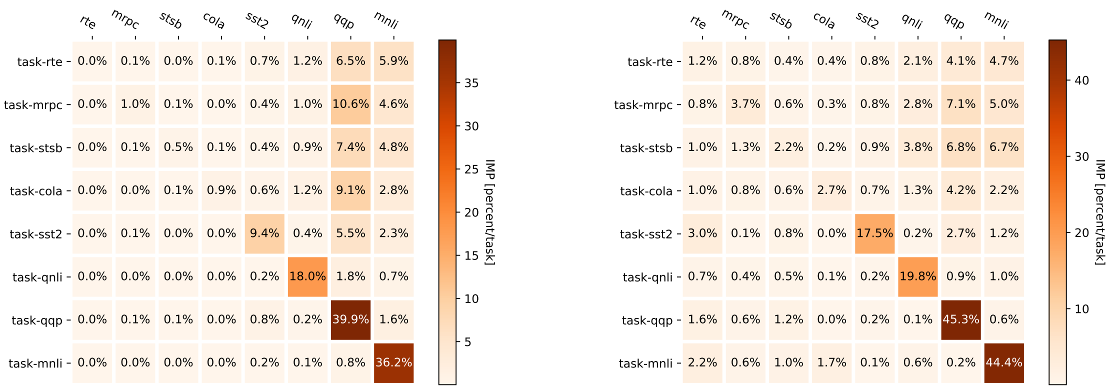
4 Related work

Pre-trained language model Language models pre-trained on large corpora are widely used in multiple NLP tasks to improve performance. However, these models are often very large. Recently, Transformer-based (Vaswani et al., 2017) models have become the most popular pre-trained language models. There is a plenty of model variants, such as BERT (Devlin et al., 2018), GPT-3 (Brown et al., 2020), XLNET (Yang et al., 2019), and Roberta (Liu et al., 2019), etc. Transformers models are huge models, ranging from 110M parameters in BERT-Base to trillions (Fedus et al., 2021; Lepikhin et al., 2020) in the largest, best-performing models. Due to the resource constraints in GPU/TPU memory and computational power, it is difficult to run a large model. So Lan et al. (2020) propose an approach to reduce the amount of training parameters by sharing weights among all transformer layers. The model named ALBERT can lower the usage of memory and speed up the training process of BERT. By contrast, ALBERT reduces the amount of parameters needed to be trained, while adapters introduce new parameters and deepen the model.

Adapters and fine-tuning Since fine-tuning approaches have proven to have better performance than feature-based approaches (Peters et al., 2019), researchers often prefer fine-tuning approaches to feature-based ones. Most of the state-of-the-art results of NLP tasks are achieved by fine-tuning a complex pre-trained model. Fine-tuning does not require a task-specific design beforehand, which is more general across tasks than feature-based ones. However, every time a model is fine-tuned on a new task, a new set of parameters are created and trained, leading to pretty low degree of parameter sharing among tasks.

Adapters model is a lightweight fine-tuning approach introduced by Houlsby et al. (2019). They insert a small set of newly initialized neural networks named adapters in each layer of the transformers. At training steps, only parameters of adapters will be updated and the parameters in the pre-trained language model will be unchanged. Therefore it reduces the number of parameters to be trained in the training phase and enables efficient parameter sharing between tasks by combining many task-specific or language-specific adapters.

Merging and pruning adapters Adapters have achieved great results in multi-task (Pfeiffer et al., 2020a), cross-lingual transfer learning (Pfeiffer



(a) Adapterfusion

(b) Adapterfusion with LTH

Figure 7: LIAs of Adapterfusion w/o pruning

et al., 2020b) and infusing knowledge (Wang et al., 2020). Adapters are capable of extracting knowledge from different tasks and can be applied to fuse knowledge they learned from different tasks (Rücklé et al., 2020b; Pfeiffer et al., 2020b; Wang et al., 2020). There are plenty of ways to merge multiple adapters, including stacking (Pfeiffer et al., 2020b), fusing and concatenating adapters (Pfeiffer et al., 2020a). However, adapters are still far from being concise, and merging multiple adapters from different tasks will introduce redundancies and slow down the inference speed of the model. Therefore, Rücklé et al. (2020a) have firstly introduced a way to remove adapters from lower transformer layers. By removing the first few layers of the adapters, it effectively speeds up the training and inference of the adapter models.

Neural networks are easily overparameterized and carry plenty of redundancies. To tackle this problem, distillation (Ba and Caruana, 2014; Hinton et al., 2015) and pruning (LeCun et al., 1990; Han et al., 2015) are introduced to streamline the model while perserving good performance. And there are several research directions in this field. For pruning before training, MobileNets (Howard et al., 2017) is designed for image-recognition networks. For pruning after training, LeCun et al. (1990) use the second derivatives to truncate the neural networks. For pruning during training, Bellec et al. (2018) reinitialize weights near zeros with random number after training the model. Moreover, we can prune models based on activations (Hu et al., 2016), filters (Li et al., 2017; Molchanov et al., 2017) or channels (He et al., 2017).

The most influential theory recently for pruning comes from Frankle and Carbin (2019), in which they prove that a dense neural network contains subnetworks (winning ticket) that can have the same performance as the original network when trained isolated. And their experiments reveal that not only the structure of the pruned networks matters but also the initial weights of these networks can affect the performance of the model. They also find that a subnetwork extracted from pruning learns faster than the original model and even reaches higher test accuracy. Our pruning approach is inspired by this theory and can prune the original Adapterfusion model to a much more concise one.

5 Conclusion and future work

In this paper, we propose a new approach to model the utilization of the adapters at each layer by defining a new indicator LIA (Layer Influence Of Adapter) with which we can identify the most influential adapters. Moreover, we introduce a novel way of pruning adapter modules inspired by the prestigious Lottery Ticket Hypothesis. The proposed pruning strategy has been extensively evaluated on the GLUE datasets, whose results show that we can prune adapters up to 40% of its original size while keeping the performance intact. We further examine the performance and LIAs of the pruned adapters in the latest state-of-the-art Adapterfusion model, and we can remove more than half of the adapters and reduce computation of the Adapterfusion model by nearly 40% with little performance loss.

This work can be further extended in many ways. For instance, iterative pruning is time-consuming,5 we can try to find the redundant adapter before and after the training by introducing new elaborate measurements.

References

Jimmy Ba and Rich Caruana. 2014. Do deep nets really need to be deep? In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 2654–2662.

Guillaume Bellec, David Kappel, Wolfgang Maass, and Robert A. Legenstein. 2018. Deep rewiring: Training very sparse deep networks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net.

Yoshua Bengio, Nicolas Le Roux, Pascal Vincent, Olivier Delalleau, and Patrice Marcotte. 2005. Convex neural networks. In *Advances in Neural Information Processing Systems 18 [Neural Information Processing Systems, NIPS 2005, December 5-8, 2005, Vancouver, British Columbia, Canada]*, pages 123–130.

James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. 2011. Algorithms for hyper-parameter optimization. In *Advances in Neural Information Processing Systems 24: 25th Annual Conference on Neural Information Processing Systems 2011. Proceedings of a meeting held 12-14 December 2011, Granada, Spain*, pages 2546–2554.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared 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 M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher 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 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

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.

William Fedus, Barret Zoph, and Noam Shazeer. 2021. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *arXiv preprint arXiv:2101.03961*.

Jonathan Frankle and Michael Carbin. 2019. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

Song Han, Jeff Pool, John Tran, and William Dally. 2015. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc.

Yihui He, Xiangyu Zhang, and Jian Sun. 2017. Channel pruning for accelerating very deep neural networks. In *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*, pages 1398–1406. IEEE Computer Society.

Geoffrey Hinton, Oriol Vinyals, and Jeffrey Dean. 2015. Distilling the knowledge in a neural network. In *NIPS Deep Learning and Representation Learning Workshop*.

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 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 2790–2799. PMLR.

Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*.

Hengyuan Hu, Rui Peng, Yu-Wing Tai, and Chi-Keung Tang. 2016. Network trimming: A data-driven neuron pruning approach towards efficient deep architectures. *arXiv preprint arXiv:1607.03250*.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2020. ALBERT: A lite BERT for self-supervised learning of language representations. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

Yann LeCun, John Denker, and Sara Solla. 1990. Optimal brain damage. In *Advances in Neural Information Processing Systems*, volume 2, pages 598–605. Morgan-Kaufmann.

Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. 2020. Gshard: Scaling giant models with conditional computation and automatic sharding. *arXiv preprint arXiv:2006.16668*.

Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. 2017. Pruning filters for efficient convnets. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net.

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*.

Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. 2017. Pruning convolutional neural networks for resource efficient inference. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net.

Matthew E. Peters, Sebastian Ruder, and Noah A. Smith. 2019. To tune or not to tune? adapting pretrained representations to diverse tasks. In *Proceedings of the 4th Workshop on Representation Learning for NLP (RepL4NLP-2019)*, pages 7–14, Florence, Italy. Association for Computational Linguistics.

Jonas Pfeiffer, Aishwarya Kamath, Andreas Rücklé, Kyunghyun Cho, and Iryna Gurevych. 2020a. Adapterfusion: Non-destructive task composition for transfer learning. *arXiv preprint arXiv:2005.00247*.

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.

Andreas Rücklé, Gregor Geigle, Max Glockner, Tilman Beck, Jonas Pfeiffer, Nils Reimers, and Iryna Gurevych. 2020a. Adapterdrop: On the efficiency of adapters in transformers. *arXiv preprint arXiv:2010.11918*.

Andreas Rücklé, Jonas Pfeiffer, and Iryna Gurevych. 2020b. MultiCQA: Zero-shot transfer of self-supervised text matching models on a massive scale. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2471–2486, Online. Association for Computational Linguistics.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

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.

Ruize Wang, Duyu Tang, Nan Duan, Zhongyu Wei, Xuanjing Huang, Cuihong Cao, Daxin Jiang, Ming Zhou, et al. 2020. K-adapter: Infusing knowledge into pre-trained models with adapters. *arXiv preprint arXiv:2002.01808*.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov, and Quoc V Le. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. *arXiv preprint arXiv:1906.08237*.

A Detail Results: AdapterFusion model LIA

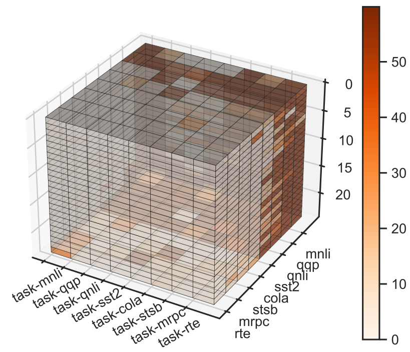
We calculate the LIA of each adapters in the Adapterfusion model. In order to have a better insight of the model, we gradually remove adapters with small LIA from the 3D heatmap, see Figure 8. In standard adapters modules, we discover that there are a number of adapters not utilized in the original Aadapterfusion model and most of the essential adapters are at the back of the cube which are the adapters trained on large datasets. In pruned Adapterfusion model, most of the adapters are essential and most of the adapters become more important for the task.

Figure 9 and Figure 10 shows the LIAs of each adapters in Adapterfusion model in each tasks which is cross-section of the cube. We find that in tasks with small datasets, the model uses adapters from different tasks, while in tasks with large datasets, the model mainly uses the adapter trained from the same task (e.g. QNLI, QQP and MNLI). However, after pruning the Adaterfusion model, it uses adapters trained on different tasks even when the target task is the one with a large dataset.

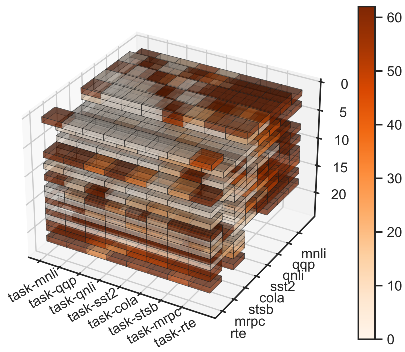
B Detail of Pruning Adapters

The single task adapter model contains 24 adapter modules. There are 12 layer of transformers block in Bert-base. Each layer with 2 adapter modules. We prune 20% of the adapters for each iteration of pruning.

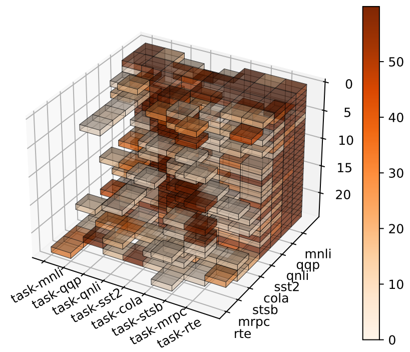
For a better understanding of the pruning algorithm 1. We summarized the proposed pruning strategy and demonstrate the inner structure of adapter in Figure 11.



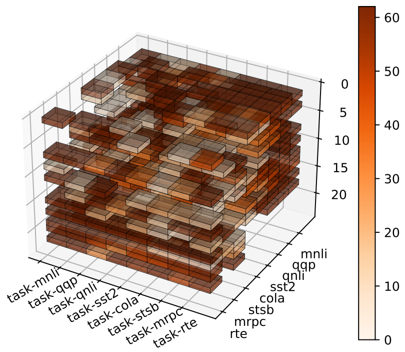
(a) Standard Adapterfusion with LIA



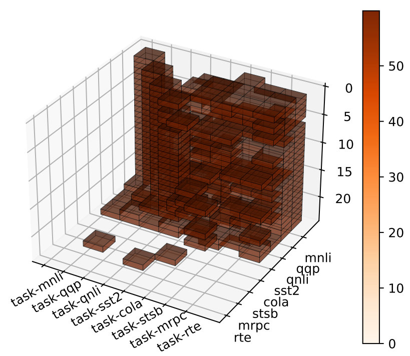
(b) Pruned Adapterfusion with LIA



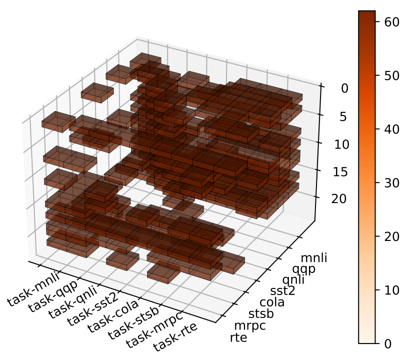
(c) Standard Adapterfusion with LIA > 0.1%



(d) Pruned Adapterfusion with LIA > 0.1%

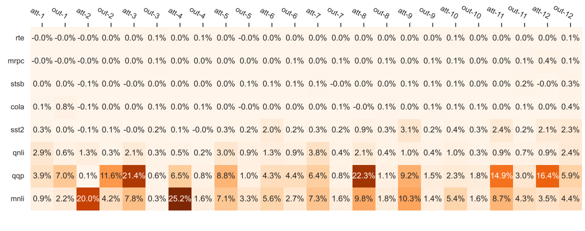


(e) Standard Adapterfusion with with LIA > 1%

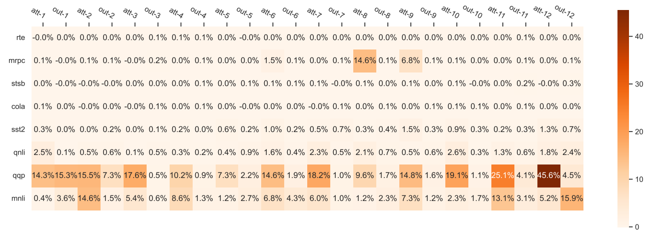


(f) Pruned Adapterfusion with LIA > 1%

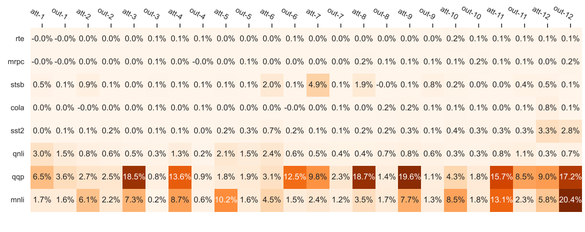
Figure 8: LIAs of Adapterfusion w/o pruning



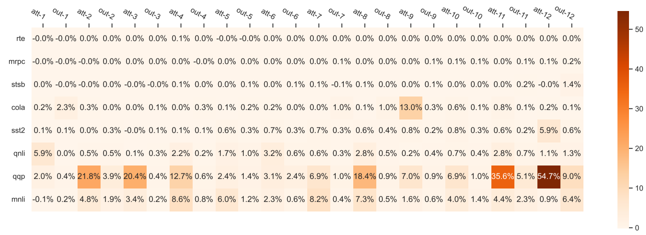
(a) RTE



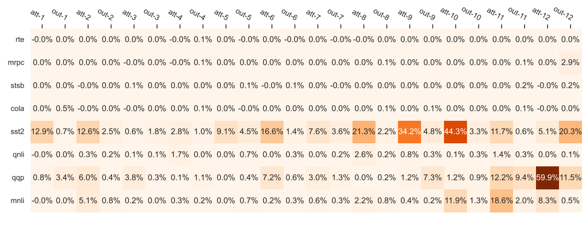
(b) MRPC



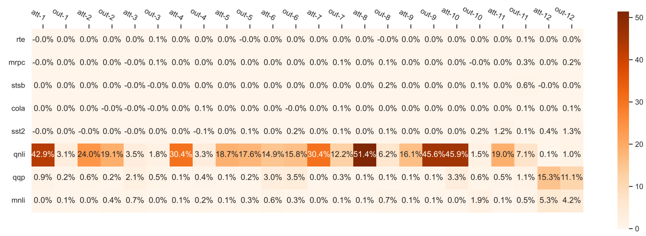
(c) STSB



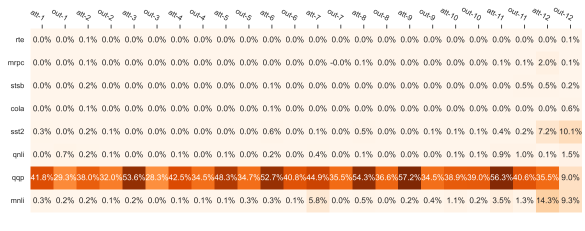
(d) CoLA



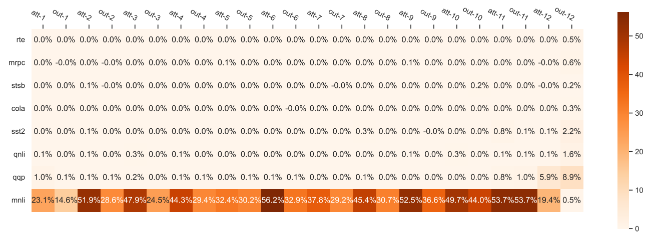
(e) SST-2



(f) QNLI

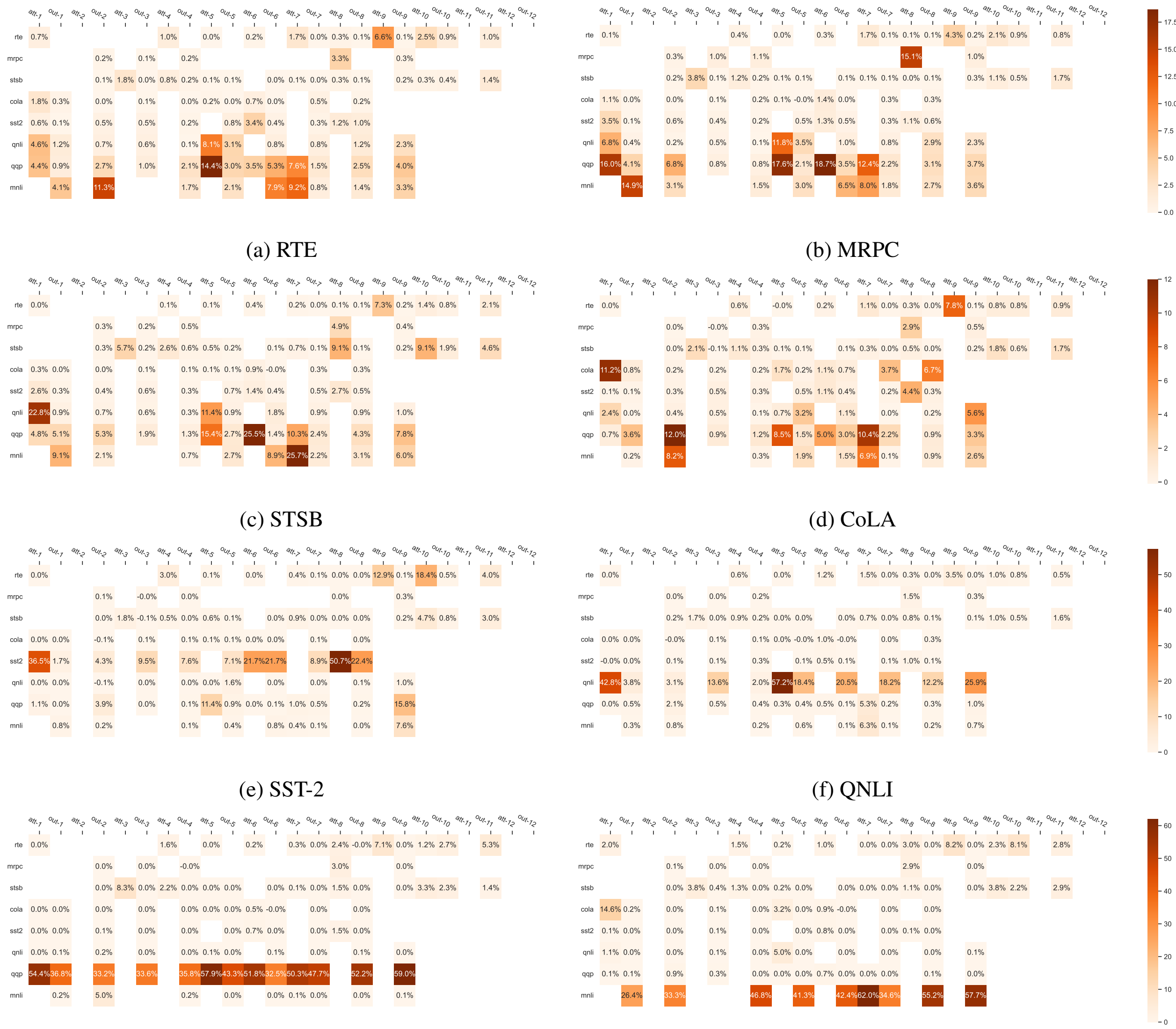


(g) QQP



(h) MNLI

Figure 9: LIAs of Standard Adapterfusion on different target tasks



(a) RTE

(b) MRPC

(c) STSB

(d) CoLA

(e) SST-2

(f) QNLI

(g) QQP

(h) MNLI

Figure 10: LIAs of Adapterfusion with pruning on different target tasks

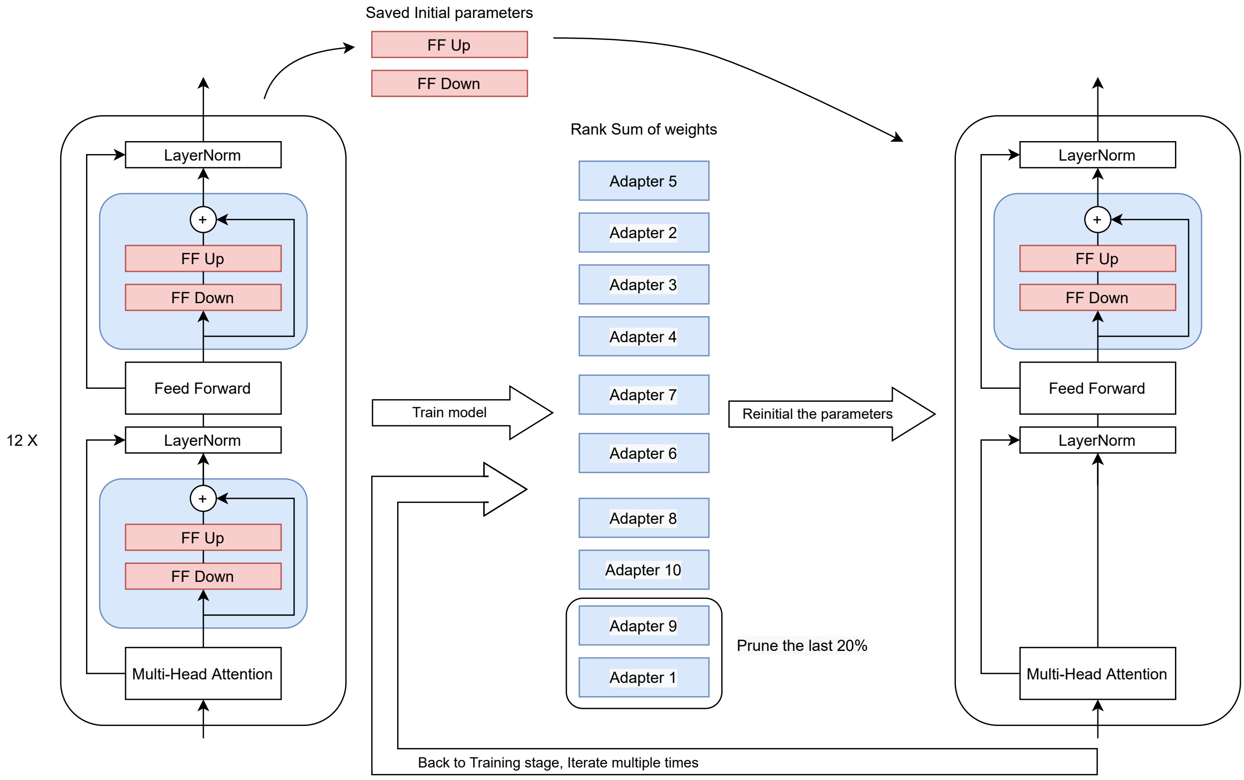


Figure 11: The Process of Pruning adapters using Lottery Ticket Hypothesis