*algorithms*

MDPI

*Article*

# Pruning Adapters with Lottery Ticket

**Jiarun Wu** † and **Qingliang Chen** *,†

Department of Computer Science, Jinan University, Guangzhou 510632, China; benwu@stu2019.jnu.edu.cn
* Correspondence: tpchen@jnu.edu.cn
† These authors contributed equally to this work.

**Abstract:** Massively pre-trained transformer models such as BERT have gained great success in many downstream NLP tasks. However, they are computationally expensive to fine-tune, slow for inference, and have large storage requirements. So, transfer learning with adapter modules has been introduced and has become a remarkable solution for those problems. Nevertheless, recent studies reveal that the parameters in adapters are actually still quite redundant, which could slow down inference speed when fusing multiple adapters for a specific downstream task, and thus, they can be further reduced. To address this issue, we propose three novel ways to prune the adapter modules iteratively based on the prestigious Lottery Ticket Hypothesis. Extensive experiments on the GLUE datasets show that the pruned adapters can achieve state-of-the-art results, with sizes reduced significantly while performance remains unchanged, and some pruned adapters even outperform the ones with the same size that are fine-tuned alone without pruning.

**Keywords:** pre-trained transformer model; adapter; prune



**Citation:** Wu, J.; Chen, Q. Pruning Adapters with Lottery Ticket. *Algorithms* **2022**, *15*, 63. https://doi.org/10.3390/a15020063

Academic Editor: Fabio Massimo Zanzotto

Received: 20 January 2022
Accepted: 9 February 2022
Published: 14 February 2022



## 1. Introduction

Transformer-based pre-trained language models [1] have gained great success in multiple NLP downstream tasks. These language models are pre-trained on large amounts of unlabeled text data with self-supervising approaches such as masked language modeling (e.g., BERT [2], Roberta [3], XLNet [4]). They have achieved state-of-the-art performance for most language understanding tasks by fine-tuning the models for a given target. However, these advancements come at a price. Recent studies have shown that the performance of the model scale with their sizes [5], and the model sizes have grown up to billions of parameters lately. Therefore, for different target tasks, models need to be fine-tuned entirely, which is computationally expensive and leads to large storage requirements.

To tackle this issue, adapter [6] has been introduced as an alternative way instead of fine-tuning the whole model for a downstream task, and it achieves comparable performance to full fine-tuning on most NLP tasks. Adapter is a small residual network inserted in each layer of the transformer. During fine-tuning, only the weights in the adapter are trained, while parameters of the pre-trained model remain unchanged. This approach can greatly reduce the total number of parameters needed to be trained for a target task and thus enable efficient parameter sharing among tasks.

However, adapters are making the model more computationally expensive in another way. With deeper transformer-based models, more adapters are inserted, and the model gets deeper. As a result, it decreases the inference speed of the model. Moreover, recent studies have found that the adapter is capable of extracting knowledge from the target task, so research attempts have also been made to fuse multiple adapters across multiple tasks to incorporate knowledge [7], which can further complicate the model and decrease the inference speed.

Nevertheless, Houlsby et al. [6] have found that not all adapters in the model are created equal. Removing some of the adapters does not compromise the performance too much. This leads to the idea that some of the parameters in adapter modules might be

redundant. So, it is quite possible to reduce the redundancies to streamline the model. The most recent work following this idea is the AdapterDrop model [8], which firstly tries to remove adapters from lower transformer layers during training and inferences, resulting in faster training and inference speed with some performance cost. However, the redundant parameters inside adapters are not fully analyzed yet.

To address these deficits, in this paper, we systematically propose three novel ways to prune the adapter modules iteratively without hurting the performance, and even narrow the performance gap between the fully fine-tuned models and the adapter-based tuned ones. The contributions are summarized as follows:

- We systematically propose three novel ways for pruning weights, neurons, and adapter layers, respectively, inspired by the prestigious Lottery Ticket Hypothesis [9], 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.
- The proposed schemes have been evaluated extensively on the GLUE datasets, where models are reduced significantly with negligible cost of accuracy.

## 2. Materials and Methods

### *2.1. Winning Ticket in Adapters*

In order to prune the redundant parameters in the adapters without affecting the overall performance, we prune the adapters iteratively to find the subnetwork (winning ticket) that can reach the same accuracy as the original adapter when trained in isolation. After every pruning iteration, weights are set to the initial values when the first training iteration starts [9].

We propose three pruning schemes for weights, neurons, and the adapter layer, respectively. Figure 1 shows our adapter architecture and three different strategies of pruning adapters. Adapters are inserted after every feed-forward layer and self-attention layer in the transformer, with a skip-connection. Each adapter is a one-layer residual network with 128 neurons in our settings. Just like dropout, pruning the adapter will not affect the workflow of the original network.

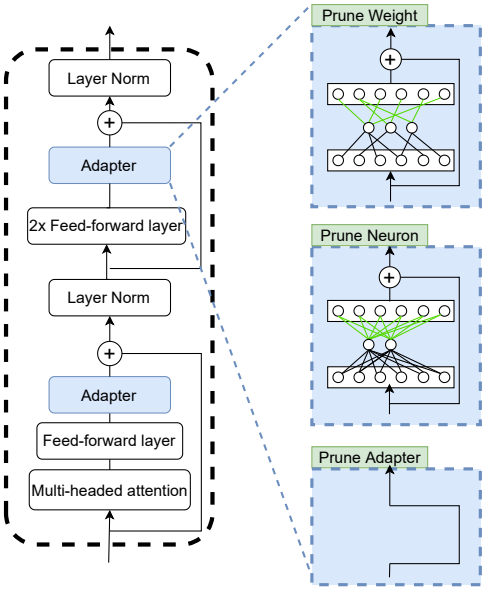


**Figure 1.** Three different ways of pruning the adapter module, which is a residual network with one layer.

### *2.2. Finding Winnning Ticket*

We explore to find the winning ticket in adapters by training and pruning them iteratively. For different pruning targets, we use different schemes to select candidates for

removal. 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 $\alpha = \alpha \sim D_\alpha$. In addition, $f(x; m \odot \theta; \alpha)$ denotes training the model with a mask $m \in \{0,1\}^{|\theta|}$. 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 $\theta_j$.
3. Prune $p$% of the parameters in $\theta_j$, and generate a mask $m$.
4. Reset the remaining parameters in adapters to $\theta_0$, and go back step 2 to train the model $f(x; \theta_0; \alpha)$ if it is not a winning ticket yet.

### *2.3. Pruning Schemes*

Here, we systematically propose three pruning strategies for weights, neurons, and the adapter layer, respectively, which are formalized in the following Algorithms 1–3.

**Weight Pruning:** We prune a weight solely by its magnitude. At iteration $t$, we calculate the absolute value of weights $|\theta_t|$ in all adapters. We save the non-zero value and its corresponding position in $\theta_t$ in a list $R$. We sort $R$ in the descending order by the absolute value. The last $p$% of $R$ with the lowest magnitude are pruned, and their corresponding positions in mask and $\theta_t$ are both set to 0. The remaining parameters are set back to the initial values, and the training restarts without the pruned weights.

**Algorithm 1:** Identify the importance of weights

```
Result: a list of tuple containing values of importance and positions in weight
        matrix
R is an empty list;
Weights of adapter at iteration t as θ_t;
for layer l in θ_t do
    for weight a in layer l do
        if a is not zero then
            Value of importance Imp_a = |a| ;
            Position of weight a in θ_t = Pos_a ;
            Append tuple (Imp_a, Pos_a) to list R ;
        end
    end
    Sort list R with Imp
end
```

**Neuron Pruning:** If we need to prune neurons, this can be done by the $L_2$ norm of their associated weights. An adapter module consists of a down-project weight matrix and an up-project weight matrix with residual connection. To prune the neuron inside an adapter, we will simplify the up-project weight matrix because the column in the weight matrix corresponds to the connection to the neuron. Let $\theta_{t,l,up}$ be the up-project weight matrix of the adapter layer $l$ at iteration $t$ and $a_{i,j}$ denote the weight parameters in $\theta_{t,l,up}$. Therefore, the importance of neuron $i$ in the adapter of size $N$ is $\sum_{j=0}^{N} a_{i,j}^2$. Likewise, neurons are sorted in the descending order by their importance, and the last $p$% of the neurons are pruned and their mask is set to 0. Then, the remaining parameters are put back to their initial values.

**Algorithm 2:** Identify the importance of neurons

**Result:** a list of tuple containing values of importance and positions in weight matrix
$R$ is an empty list;
The size of adapter is $N$ ;
Weights of adapter at iteration $t$ as $\theta_t$;
**for** *layer $l$ in $\theta_t$* **do**
  up-project weight matrix $l_u$ in layer $l$ ;
  **for** *column $i$ in $l_u$* **do**
    **if** $\sum_{j=0}^{N} a_{i,j}$ *is not zero* **then**
      Value of importance $Imp_c = \sum_{j=0}^{N} a_{i,j}^2$ ;
      Position of column $a$ in $\theta_t$ = $Pos_c$ ;
      Append tuple $(Imp_c, Pos_c)$ to list $R$ ;
    **end**
  **end**
  Sort list $R$ with $Imp$
**end**

**Adapter Pruning:** Finally, we can even prune adapters based on their sum of weights magnitude. 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 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. The remaining adapters step back to their initial weights for the re-training.

**Algorithm 3:** Identify 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 $\theta_t$;
**for** *layer $l$ in $\theta_t$* **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**
  Sort list $R$ with $Imp$
**end**

### *2.4. Experimental Settings*

We use the public BERT-Based uncased model as our base model, and we apply the similar approach in Devlin et al. [10] to perform a text classification task. In each input sequence, the first token is a classification token. Then, its embedding is fed into a linear layer to make a prediction.

We set the adapter size to 128 because engineering practices [11] suggest that overparameterized networks are easier to train. We use Adam optimizer to train the model and perform a small hyperparameter search using TPE algorithm [12] on the adapter training without pruning. We sweep learning rates in $\{1 \times 10^{-4}, 5 \times 10^{-4}, 1 \times 10^{-3}\}$ and the number of epochs in $\{3, 4, 5\}$. We have made 30 trials with Bayesian optimization and select the best run setting for the pruning experiments. The learning rate of the model is linearly warming up in the first epoch of training and then gradually decreases. We

train the models on an NVIDIA Tesla P100 GPU with a batch size of 32. During iterative pruning, we use an early-stopping strategy with patience of three to speed up training. The particular early-stopping criterion we use is the iteration with minimum validation loss during training.

We extensively evaluate the three proposed pruning schemes on GLUE datasets [13], which contain nine sentence or sentence-pair language understanding tasks (We omit WNLI because it is not evaluated in BERT [2]). We treat $\text{MNLI}_{\text{mm}}$ and $\text{MNLI}_{\text{m}}$ separately and conduct transfer learning from the pre-trained BERT-Base model, which has 12 layers and a total of 110M parameters. Finally, we report the test results through the submission website l (https://gluebenchmark.com (accessed on 20 January 2022)).

## 3. Results

In this section, we present the results of our proposed pruning schemes extensively evaluated on the prestigious GLUE datasets [13].

### *3.1. Comparisons of Pruning Strategies*

We insert adapters of size 128 into each layer of transformers in the BERT-Based model. Depending on the 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 15 iterations for weight pruning and neuron pruning and 11 iterations for layer pruning, since there will be less than one adapter left after the 11th iteration.

The results of the GLUE test sets are presented in Table 1. For each pruning strategy, we select the best result in all iterations of pruning. The best model (the winning ticket found) is chosen by the metric of the corresponding task and evaluated using the GLUE testing server. From the test results, we can see a 0.2 percentage performance gap between the original adapter model and the fully fine-tuned one. Our pruning schemes help us spot the pruned adapters that outperforms the original ones, implying that the winning ticket has been found successfully. Particularly, models after pruning even outperform the fine-tuned ones in datasets CoLA, QQP, QNLI, $\text{MNLI}_{\text{mm}}$, and RTE, and particularly CoLA, which has been improved by more than 1%.

**Table 1.** Test results on GLUE test sets using the 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. For three pruning schemes, the best model on the validation set is tested.

| | **CoLA** | **SST-2** | **MRPC** | **STSB** | **QQP** | **MNLI m/mm** | | **QNLI** | **RTE** | **AVG** |
|---|---|---|---|---|---|---|---|---|---|---|
| Full | 52.10 | 93.50 | 88.90 | 85.80 | 71.20 | 84.60 | 83.40 | 90.50 | 66.40 | 79.60 |
| Adapter | 51.70 | 93.10 | 88.50 | 85.60 | 71.50 | 83.40 | 83.20 | 90.50 | 67.30 | 79.42 |
| Weight | 54.00 | 92.80 | 88.10 | 84.60 | 71.80 | 84.30 | 83.60 | 91.10 | 65.80 | 79.57 |
| Neuron | 52.10 | 93.00 | 88.50 | 84.60 | 71.90 | 83.80 | 83.30 | 90.60 | 68.40 | 79.58 |
| layer | 49.50 | 92.60 | 88.00 | 83.50 | 71.50 | 84.10 | 83.10 | 90.80 | 70.60 | 79.30 |

In general, models pruned with weights and neurons can outperform the adapter model of size 128 on average and have almost the same performance as the fully fine-tuned ones, as shown in Figure 2. It indicates that the pruning strategies can enable us to identify a better subnetwork of adapters and thus boost the model's performance.

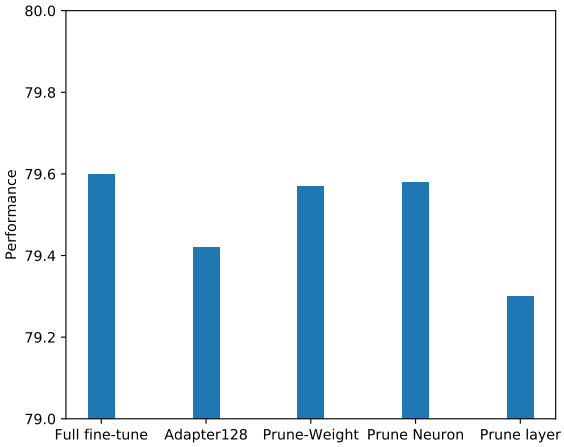


**Figure 2.** Comparisons of average performance of different models on GLUE test sets.

### *3.2. Iterative Pruning and Winning Ticket*

Then, we investigate the performance of different pruning strategies during the iterations with the validation set of the GLUE dataset as Figure 3 shows. The baselines of the adapter model with a size of 128 are denoted as straight lines with task names and scores marked on them in the figure. Let $p$ denote the percentage of weights remaining. We discovered that in most cases, pruning weights and pruning neurons are better than pruning the adapter layers, especially when $p \leq 40\%$. Meanwhile, pruning weights and neurons are more stable in their performance than pruning the adapter layers; this indicates that pruning adapter layers have a higher risk of removing the essential parameters and thus leads to a performance drop in an early stage of pruning.

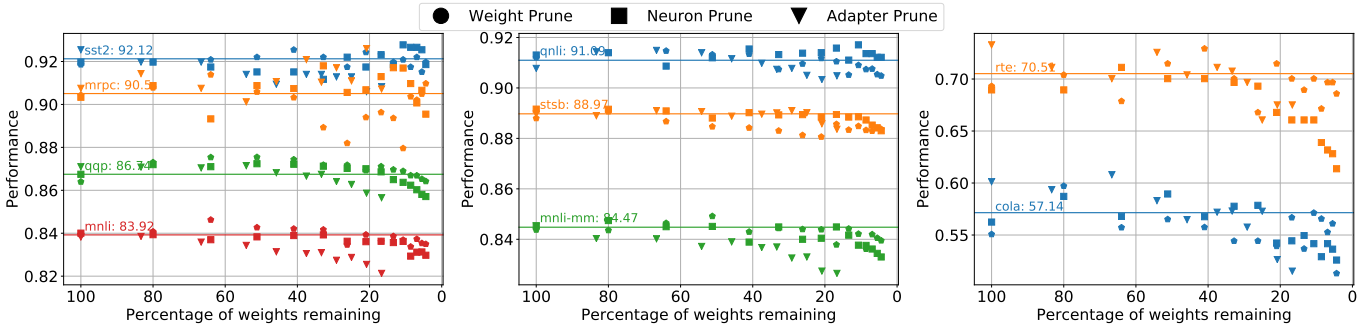


**Figure 3.** Performance of three different pruning schemes on GLUE validation sets at every pruning iteration. Horizontal line represents the performance of adapters before pruning starts. Each type of marker denotes a pruning scheme.

In all datasets, there is a substantial amount of validation accuracy improvements over the standard adapter model when $p \geq 40\%$, particularly in the CoLA and RTE datasets.

For weight pruning and neuron pruning, a winning ticket can be found when $p$ is around 21% and 26%. For adapter layer pruning, the winning ticket can not been spotted since the overall structure of the network has been altered too much, but there is no significant performance drop until $p$ is around 40%. Finally, we test the pruned models in the validation set with the GLUE test set, whose results is shown in Table 2.

We also see that the average performance for all tasks in GLUE of the winning ticket model is a bit lower than the original adapter model. However, if we only consider the datasets with sizes more than 10 K, as shown in Figure 4, we found that the winning ticket produced by weight pruning and neuron pruning has the same or even better performance than the original adapter model, which implies that the pruning strategy works better on larger datasets, since a dataset of larger size has a larger validation set, and a larger validation set will better guide the pruning to identify the best fit winning ticket.

**Table 2.** Winning ticket of different pruning schemes in test results on GLUE test sets using GLUE server and the percentage of parameters trained. Average results on all GLUE datasets and average results on large datasets in GLUE (SST-2, QQP, MNLI, $\text{MNLI}_{\text{mm}}$, QNLI). Training time of all task in GLUE, with Full fine-tune (4 h) as base.

| | **Param/Task** | **AVG (Total/Large)** | | **Training Time** |
|---|---|---|---|---|
| Full fine-tune | 100% | 79.60 | 84.64 | 1.00× |
| Adapter 128 | 4.15% | 79.42 | 84.34 | 1.07× |
| Adapter 27 | 0.92% | 79.20 | 84.06 | 1.05× |
| Prune Weight (21%) | 0.87% | 79.22 | 84.42 | 1.07× |
| Prune Neuron (26%) | 1.08% | 79.22 | 84.34 | 1.05× |
| Prune Layer (37.5%) | 1.56% | 79.14 | 84.00 | 1.03× |

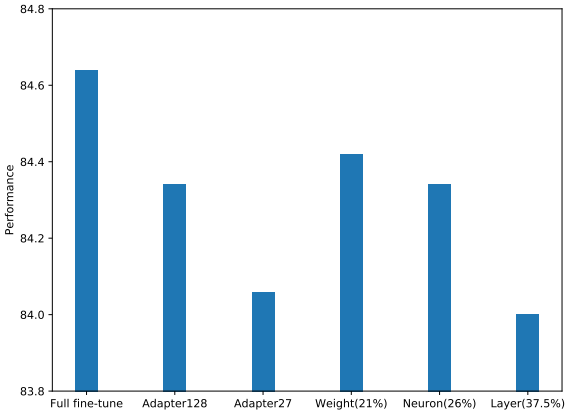


**Figure 4.** Winning ticket of different pruning schemes on GLUE test, with average results on large datasets in GLUE (SST-2, QQP, MNLI, $\text{MNLI}_{\text{mm}}$, QNLI)

Moreover, we further test an adapter model with size 27, which is approximately the number of parameters in an adapter of size 128 with 21% left. It is surprise to see that the winning ticket by weight pruning outperforms the adapter of size 27, indicating that the pruned adapter model can be better than the one that is only fine-tuned.

## 4. Discussion

Most neural networks are overparameterized and carry a lot of redundancies. Therefore, distillation [14,15] and pruning [16,17] are introduced to streamline the model while perserving accuracy. There are several research directions on training small or sparse networks, pruning before training [18], pruning after training [16], and pruning during training [19].

Adapters have achieved great results in multi-task [7] , cross-lingual transfer learning [20], and infusing knowledge [21]. It indicates that adapters are capable of extracting useful features from tasks. They are applied to fuse knowledge learned from different tasks for a target one. However, adapters are still far from being concise, and fusing multiple adapters may cause inference inefficiency. To tackle this problem, Rücklé et al. [8] have firstly introduced a way to remove adapters from lower transformer layers.

Recently, the Lottery Ticket Hypothesis has been the most influential theory for pruning, where Frankle and Carbin [9] discover that a dense neural network contains subnetworks (winning ticket) that can have the same performance as the original network when trained isolated. They further show that not only the structure of the pruned networks can affect the performance but also the initial weights of these networks. It is also revealed that using a certain pruning strategy, a subnetwork can be found that learns faster than the original one and reaches higher test accuracy. Our proposed approaches are motivated by this theory and aim to identify the winning ticket with a systematical pruning strategies for different levels.

## 5. Conclusions

In this paper, we have proposed three novel ways of pruning redundant parameters in adapters, inspired by the prestigious Lottery Ticket Hypothesis. We prune adapters on three different levels: weights, neurons, and adapter layers. Adapters are pruned iteratively and when every iteration starts, weights are re-winded to initial values. Finally, the proposed schemes have been extensively evaluated on the GLUE datasets, successfully identifying the subnetworks (winning ticket) in adapters and achieving state-of-the art results. We can prune adapters to 21% of their original size without causing a performance drop, and in some datasets, the pruned adapters can even outperform the original ones.

However, the pruning strategies in this paper are still time-consuming. It takes a tremendous amount of computation to explore and find the most compact adapter model. So, future work will be focusing on how to search the winning ticket of the adapter modules more efficiently.

**Author Contributions:** Conceptualization, J.W. and Q.C.; methodology, J.W.; software, J.W.; validation, J.W.; formal analysis, J.W.; investigation, J.W. and Q.C.; resources, J.W. and Q.C.; writing—original draft preparation, J.W.; writing—review and editing, Q.C.; visualization, J.W.; supervision, Q.C. All authors have read and agreed to the published version of the manuscript

**Funding:** This research is supported by Qinghai Provincial Science and Technology Research Program (grant No.2021-QY-206), Guangdong Provincial Science and Technology Research Program (grant No.2020A050515014), and National Natural Science Foundation of China (grant No. 62071201).

**Institutional Review Board Statement:** Not applicable.

**Informed Consent Statement:** Not applicable.

**Data Availability Statement:** Data is contained within the article.

**Conflicts of Interest:** The authors declare no conflict of interest.

## References

1. Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A.N.; Kaiser, L.U.; Polosukhin, I. Attention is All you Need. In *Advances in Neural Information Processing Systems*; Guyon, I., Luxburg, U.V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., Garnett, R., Eds.; Curran Associates, Inc.: Red Hook, NY, USA, 2017; Volume 30.
2. Devlin, J.; Chang, M.W.; Lee, K.; Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv* **2018**, arXiv:1810.04805.
3. Liu, Y.; Ott, M.; Goyal, N.; Du, J.; Joshi, M.; Chen, D.; Levy, O.; Lewis, M.; Zettlemoyer, L.; Stoyanov, V. Roberta: A robustly optimized bert pretraining approach. *arXiv* **2019**, arXiv:1907.11692.
4. Yang, Z.; Dai, Z.; Yang, Y.; Carbonell, J.; Salakhutdinov, R.; Le, Q.V. Xlnet: Generalized autoregressive pretraining for language understanding. *arXiv* **2019**, arXiv:1906.08237.
5. Kaplan, J.; McCandlish, S.; Henighan, T.; Brown, T.B.; Chess, B.; Child, R.; Gray, S.; Radford, A.; Wu, J.; Amodei, D. Scaling laws for neural language models. *arXiv* **2020**, arXiv:2001.08361.
6. Houlsby, N.; Giurgiu, A.; Jastrzebski, S.; Morrone, B.; De Laroussilhe, Q.; Gesmundo, A.; Attariyan, M.; Gelly, S. Parameter-Efficient Transfer Learning for NLP. In Proceedings of the 36th International Conference on Machine Learning, Long Beach, CA, USA, 9–15 June 2019; Chaudhuri, K., Salakhutdinov, R., Eds.; Volume 97, pp. 2790–2799.
7. Pfeiffer, J.; Kamath, A.; Rücklé, A.; Cho, K.; Gurevych, I. AdapterFusion: Non-destructive task composition for transfer learning. *arXiv* **2020**, arXiv:2005.00247.
8. Rücklé, A.; Geigle, G.; Glockner, M.; Beck, T.; Pfeiffer, J.; Reimers, N.; Gurevych, I. AdapterDrop: On the Efficiency of Adapters in Transformers. *arXiv* **2020**, arXiv:2010.11918.
9. Frankle, J.; Carbin, M. The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks. In Proceedings of the 7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, 6–9 May 2019. Available online: https://openreview.net/ (accessed on 20 January 2022).
10. Devlin, J.; Chang, M.W.; Lee, K.; Toutanova, K. 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), Minneapolis, MN, USA, 2–7 June 2019; Association for Computational Linguistics: Minneapolis, MN, USA, 2019; pp. 4171–4186. [CrossRef]
11. Bengio, Y.; Roux, N.L.; Vincent, P.; Delalleau, O.; Marcotte, P. Convex Neural Networks. In Proceedings of the Advances in Neural Information Processing Systems 18 Neural Information Processing Systems—NIPS 2005, Vancouver, BC, Canada, 5–8 December 2005; pp. 123–130.

12. Bergstra, J.; Bardenet, R.; Bengio, Y.; Kégl, B. Algorithms for Hyper-Parameter Optimization. In Proceedings of the Advances in Neural Information Processing Systems 24: 25th Annual Conference on Neural Information Processing Systems 2011, Granada, Spain, 12–14 December 2011; Shawe-Taylor, J., Zemel, R.S., Bartlett, P.L., Pereira, F.C.N., Weinberger, K.Q., Eds.; pp. 2546–2554.
13. Wang, A.; Singh, A.; Michael, J.; Hill, F.; Levy, O.; Bowman, S. GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding. In Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP, Brussels, Belgium, 1 November 2018; pp. 353–355. [CrossRef]
14. Ba, J.; Caruana, R. Do Deep Nets Really Need to be Deep? In Proceedings of the Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, Montreal, QC, Canada, 8–13 December 2014; Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N.D., Weinberger, K.Q., Eds.; pp. 2654–2662.
15. Hinton, G.; Vinyals, O.; Dean, J. Distilling the Knowledge in a Neural Network. In Proceedings of the NIPS Deep Learning and Representation Learning Workshop, Montreal, QC, Canada, 7–12 December 2015.
16. LeCun, Y.; Denker, J.; Solla, S. Optimal Brain Damage. In *Advances in Neural Information Processing Systems*; Touretzky, D., Ed.; Morgan-Kaufmann: Burlington, MA, USA, 1990; Volume 2, pp. 598–605.
17. Han, S.; Pool, J.; Tran, J.; Dally, W. Learning both Weights and Connections for Efficient Neural Network. In *Advances in Neural Information Processing Systems*; Cortes, C., Lawrence, N., Lee, D., Sugiyama, M., Garnett, R., Eds.; Curran Associates, Inc.: Red Hook, NY, USA, 2015; Volume 28.
18. Howard, A.G.; Zhu, M.; Chen, B.; Kalenichenko, D.; Wang, W.; Weyand, T.; Andreetto, M.; Adam, H. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv* **2017**, arXiv:1704.04861.
19. Bellec, G.; Kappel, D.; Maass, W.; Legenstein, R.A. Deep Rewiring: Training very sparse deep networks. In Proceedings of the 6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, 30 April–3 May 2018; Conference Track Proceedings. Available online: https://openreview.net/ (accessed on 20 January 2022).
20. Pfeiffer, J.; Vulić, I.; Gurevych, I.; Ruder, S. 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), Online, 16–20 November 2020; pp. 7654–7673. [CrossRef]
21. Wang, R.; Tang, D.; Duan, N.; Wei, Z.; Huang, X.; Cao, C.; Jiang, D.; Zhou, M. K-adapter: Infusing knowledge into pre-trained models with adapters. *arXiv* **2020**, arXiv:2002.01808.