AdapterBias: Parameter-efficient Token-dependent Embedding Shift for Adapters in NLP Tasks

Anonymous ACL submission

Abstract

Transformer-based pre-trained models with millions of parameters require large storage. Recent approaches tackle this shortcoming by training adapters, but these approaches still require a relatively large number of parameters. In this study, AdapterBias, a surprisingly simple yet effective adapter architecture, is proposed. AdapterBias adds a token-dependent shift to the embedding to adapt to downstream tasks with only a vector and a linear layer. Extensive experiments are conducted to demonstrate the effectiveness of AdapterBias. The experiments show that our proposed method can dramatically reduce the trainable parameters than the previous works with a minimal decrease in task performances compared with fine-tuned pre-trained models. We further find that AdapterBias automatically learns to assign more significant shifts to the tokens related to the task in consideration.

1 Introduction

While large pre-trained language models (PLMs) reached the state-of-the-art results on natural language processing (NLP) tasks, PLMs require updating all parameters and storing the fully fine-tuned model for each downstream task. These requirements have led to difficulties in real-world applications. Moreover, fine-tuning PLMs on low-resource datasets are subject to instabilities.

To tackle these shortcomings, Adapters (Houlsby et al., 2019), a more parameter-efficient alternative training strategy for the transformer architecture (Vaswani et al., 2017) has been proposed. Instead of fully fine-tuning the whole model, Adapters introduces extra tunable weights and freezes the original parameters of PLM. Adapters demonstrated comparable performance with fully fine-tuning the entire model. Although Adapters solve the problem of the PLM's massive parameters, researchers are curious about how many more parameters are required to reach state-of-the-art performance on

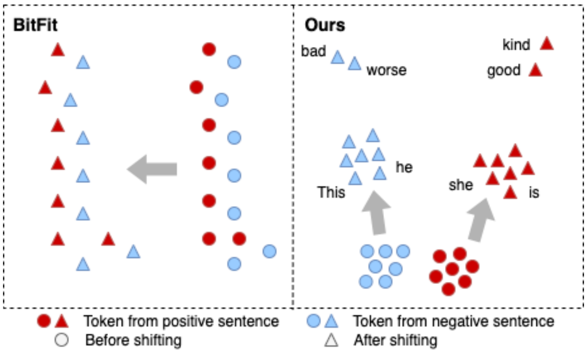

Figure 1: Overview of the main concept of our work compared to BitFit (Ben Zaken et al., 2021). Left: BitFit tends to add the same embedding shift to different tokens. Right: Our work applies different embedding shifts to tokens considering their importance to the downstream task and their characteristics. The shifts of the input words that are more task-related is more significant than that of other tokens. For example, in SST-2 (Socher et al., 2013), which is a semantic task, the embedding shifts of the semantic words, such as "kind" and "worse", are larger than that of other words.

standard NLP tasks. The results in Houlsby et al. (2019) have shown that the performance on GLUE benchmark (Wang et al., 2018) drops slightly when removing the Adapters in the first layers, which indicates that not every adapter is useful. It leaves the question of whether adapters can be even more parameter-efficient.

To develop practical and memory-efficient adapters, Diff pruning (Guo et al., 2020) enables parameter-efficient transfer learning that scales well with new tasks. The approach learns a task-specific "diff" vector that extends the original pre-trained parameters and encourages the sparsity of the vector through L_0-norm regularization. Another approach is BitFit (Ben Zaken et al., 2021), which shows that with small-to-medium training data, fine-tuning only a subset of the bias terms of pre-trained BERT models (Devlin et al., 2018) is competitive with fine-tuning the entire model. The central concept of these approaches is to add a task-specific shift to the output embedding of the PLM so as to adapt to different tasks. In the previous works, Ben Zaken et al. (2021); Guo et al. (2020) both add the same embedding shifts regardless of which token is more relevant to the task. However, considering some specific tokens might be more critical to a particular task, the embedding can better adapt to the downstream task under a limited amount of parameters if these shifts are based on the input tokens.

Based on this concept, in this study, we add token-dependent biases to the embedding shifts by proposing AdapterBias, which consists of a vector and a linear layer (L_α). The vector represents the task-specific shift, and L_α produces the weights for input tokens. Thus, with vector and the weights, AdapterBias can add a token-dependent shift to the transformer layer. Since the concept of BitFit (Ben Zaken et al., 2021) is similar to AdapterBias by adding an embedding shift, we demonstrate the difference between BitFit and AdapterBias in Figure 1. BitFit assigns the identical shifts to all the tokens, while AdapterBias adds more significant shifts to the tokens related to the task.

With fewer trainable parameters required, AdapterBias achieves comparable performance on the GLUE benchmark with Houlsby et al. (2019); Pfeiffer et al. (2020a); Guo et al. (2020); Ben Zaken et al. (2021). We further decrease the parameters of AdapterBias in different ways, including partial weight-sharing in AdapterBias and adding L_0-norm regularization. Finally, AdapterBias has better interpretability due to its simplicity. We use different tools, including WordCloud and PCA (Jolliffe, 2002), to visualize what AdapterBias has learned, and we found that the proposed approach automatically learns to assign larger shifts to the task-related tokens.

2 Related Work

For NLP tasks, adapters are introduced for the transformer architecture. A set of adapter parameters was added at each transformer layer, which is mostly bottleneck architectures. By keeping the output dimension similar to their input, they cause no change to the structure or parameters of the original model.

Adapters quickly gained popularity in NLP with various applications. For multi-task learning (Caruana, 1997; Zhang and Yang, 2017; Liu et al., 2019b), a projected self-attention layer is proposed by Stickland and Murray (2019), while Bapna et al. (2019) proposed an additional layer norm suitable for machine translation.

Besides the applications of adapters, researchers are also dedicated to improving their performance. Based on the architecture introduced by Houlsby et al. (2019), AdapterFusion (Pfeiffer et al., 2020a) leveraged knowledge from multiple tasks with a new two-stage learning algorithm. Despite the recent popularity of these methods, they still train a relatively large number of training parameters.

Recently, studies start to focus on improving the parameter-efficiency of adapters. Diff-pruning (Guo et al., 2020) achieves parameter efficiency by adding a sparse, task-specific difference-vector to the fixed original parameters. The vector is adaptively pruned during training with a differentiable approximation to the L_0-norm penalty to encourage sparsity. Pfeiffer et al. (2020b) introduced AdapterDrop (Rücklé et al., 2020) by removing adapters from lower transformer layers during training and inference, which can dynamically reduce the computational cost. Mahabadi et al. (2021) proposed Compacter, which improved the trade-off between performance and trainable parameters per task with low-rank optimization.

On the other hand, without modifying the architecture of the PLM, BitFit (Ben Zaken et al., 2021) shows that fine-tuning only the bias terms of a large PLM is also competitive with fine-tuning the entire model. Fine-tuning only the bias terms can be considered as adding a task-specific shift to the token embedding. BitFit is most similar to our work. While in BitFit, the shifts added to all the embeddings are exactly the same for all input tokens, in our work, the embedding shifts are token-dependent.

3 Method

In this section, we present AdapterBias, an efficient way to adapt large-scale PLMs. In order to better adapt to different downstream tasks, the adapter module should be token-specific. AdapterBias produces a suitable weight of the bias based on the input tokens.

Problem Formulation We consider the general problem of fine-tuning PLMs, where the training data $D = (x_i, y_i)_{n=1}^{N}$ is given. Assume that given a PLM with parameters θ and AdapterBias with parameters θ'. During the training stage, we freeze

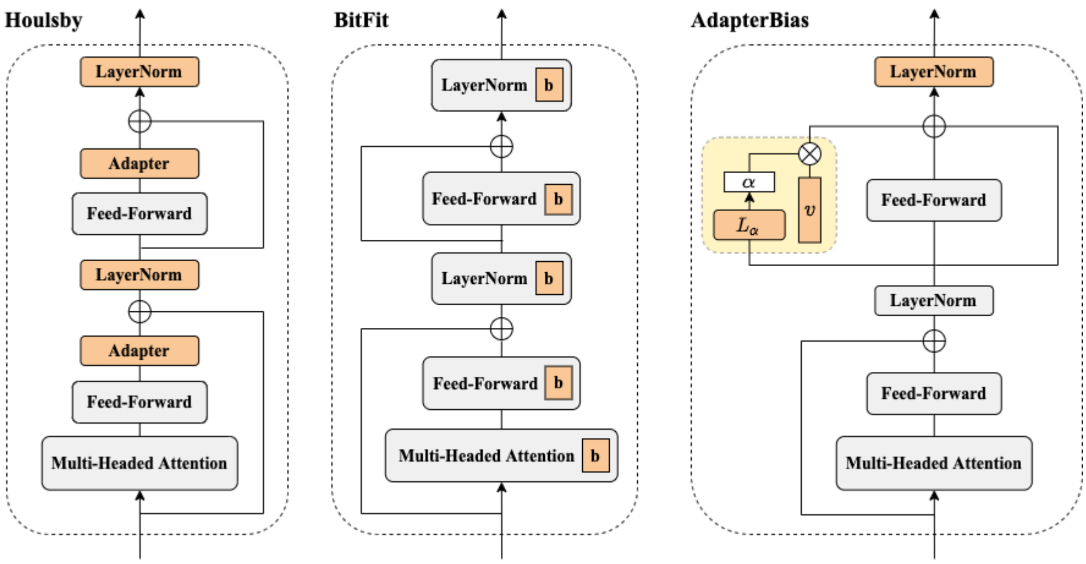

Figure 2: Model architectures comparison of Houlsby et al. (2019), BitFit (Ben Zaken et al., 2021), and the proposed method AdapterBias. The orange blocks indicate the trainable parts, while the gray blocks indicate the frozen parameters during the training stage. Left: Houlsby et al. (2019) adds their Adapters after the feed-forward layers, and their Adapter consists of two linear layers and an active function. Middle: BitFit tunes all biases from the original transformer layers. Right: AdapterBias, consisting of a linear layer (L_α) and a vector (v), is added after the second feed-forward layer only in each transformer layer.

θ and tune θ' only.

3.1 AdapterBias

The architecture of AdapterBias is shown in the right part of Figure 2. AdapterBias consists of two modules: a vector (v) and a linear layer (L_α). v is a task-specific shift added to the embedding output of each transformer layer. Since some tokens are more task-related, these tokens should be assigned larger embedding shifts than other tokens. The linear layer (L_α) produces a token-dependent weight vector $\alpha = [\alpha_1, \alpha_2 \ldots \alpha_m]^T$, where α_i is the weight of the ith token's embedding shift. By applying the token-specific weight to the task-specific embedding shift (v), AdapterBias can focus on the tokens that are more related to the task and is able to adapt to different downstream tasks efficiently.

We define the output of AdapterBias as the bias (B), which is the outer product of v and the learned weights vector α. When the dimension of the token's embedding is e with with m input tokens, the function can be defined as follows:

$$B = v \otimes \alpha^T = \begin{pmatrix} \alpha_1 v & \alpha_2 v & \ldots & \alpha_m v \end{pmatrix} \quad (1)$$

where $v \in \mathbb{R}^e$, $\alpha \in \mathbb{R}^m$, and $B \in \mathbb{R}^{e \times m}$.

To further elaborate the details of AdapterBias, we give an example of how AdapterBias produces B and how B adds to the transformer layer. In Figure 3, we assume that there are three embedding outputs (e_1, e_2, e_3) after the first layer normalization. The dimension of e_1, e_2 and e_3 is 768. Note that the dimension of the vector (v) in AdapterBias is also 768. With three token embedding inputs (e_1, e_2, e_3), the linear layer (L_α) produces α, where $\alpha \in \mathbb{R}^3$. The blocks in different colors represent the difference of the weights ($\alpha_1, \alpha_2, \alpha_3$). After performing outer product with the weights vector α and the vector (v), the dimension of B becams 768×3. For example, b_1, the first column of B, is the embedding shift for the first token.

3.2 Further improvement on parameter-efficiency of AdapterBias

In this section, we experiment on two ways to make AdapterBias more parameter efficient. One is partial weight-sharing of AdapterBias among transformer layers, another is enforcing the weights of the linear layer (L_α) to be sparse by utilizing L_0-norm penalty.

3.2.1 Cross-layer parameters sharing in AdapterBias

Redundancies have been observed in the information captured by adapters, with adapters in lower layers being less important. In the work of Houlsby et al. (2019), they observed that their Adapter modules in the lower layers are less important. In addition, sharing parameters of the Adapter across layers leads to a comparatively small drop in performance in some tasks. In light of the above information, we further reduce the number of parameters required for each task by partially sharing the weights of the adapters across all transformer layers. The experimental result are discussed at Section 4.6.1.

3.2.2 L_0 regularization in AdapterBias

Sparsity has been utilized in various parameter-efficient methods. For applications in NLP tasks, Diff-pruning (Guo et al., 2020) learns a sparse vector added to the whole PLM with L_0-norm penalty. Inspired by their work, we further apply L_0-norm regularization to L_α in the AdapterBias module, aiming to encourage the sparsity of L_α. We choose to drop L_α because it contributes most of the parameters in AdapterBias. Encouraging its sparsity can further increase the parameter efficiency. Note that we specifically apply L_0 regularization in Section 4.6.2.

In AdapterBias, we add L_0-norm penalty to the linear layer (L_α). The optimization problem can be expressed as,

$$\underset{\theta'}{min}\, L(D;\theta,\theta') + \lambda \|\theta'_{L_\alpha}\|_0, \tag{2}$$

where $L(D;\cdot)$ represents the original loss with training data D. λ is the hyperparameter for L_0-norm penalty. Note that θ' represents trainable parameters and θ'_{L_α} represents the parameters of L_α in AdapterBias. Following the work of Diff-pruning, we utilize a relaxed mask vector (Louizos et al., 2017) with a stretched Hard-Concrete distribution (Jang et al., 2016; Maddison et al., 2016) to encourage L_0 sparsity.

4 Experiments

In this section, we evaluate the effectiveness of our proposed adapter module in NLP training tasks, and provide the analysis of what AdapterBias has learned in different tasks.

4.1 Experimental settings

For the experiments, we base our experiments on HuggingFace PyTorch implementation (Wolf et al., 2019) of BERT (Devlin et al., 2018) and RoBERTa (Liu et al., 2019c) models. The learning rate is set in the range $[10^{-4}, 10^{-3}]$, with AdamW (Loshchilov and Hutter, 2017) as the optimizer. GLUE benchmark (Wang et al., 2018) and SQuAD v1.0 (Rajpurkar et al., 2016) are the training data in our settings. The training details are shown in Appendix A. Note that the second layer normalization in each transformer layer is also tuned during the training stage, corresponding to the orange component in the right part of Figure 2. Due to instability during training, we experiment with 3 random seeds and report the best. We report the test metrics provided on the submission website[1].

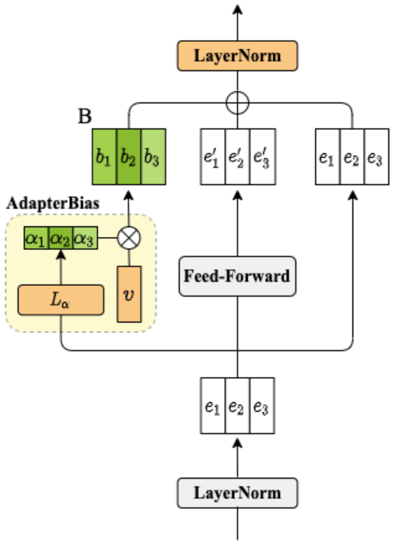

Figure 3: The detailed architecture of how AdapterBias produces the bias (B) and how B is added to the output of transformer layers.

4.2 Results on GLUE

In this section, we compare AdapterBias to other parameter-efficient methods, including Adapters (Houlsby et al., 2019), AdapterFusion (Pfeiffer et al., 2020a), Diff-pruning (Guo et al., 2020), and BitFit (Ben Zaken et al., 2021). In Table 1, we report the test scores on GLUE benchmark and the percentage of required new parameters per task. Here we use BERT-large as the PLM. AdapterBias reaches 81.2 average score in GLUE benchmark, with the smallest parameters (0.067%) added per task. AdapterBias shows competitive performance as its parameters are 31.34× and 298.51× less than the works of Houlsby et al. (2019); Pfeiffer et al. (2020a), respectively. Although Diff-pruning (Guo et al., 2020) has the best average score among all parameter-efficient methods, their work trains an additional vector whose parameters are equivalent to the parameters of the whole PLM. Thus, Diff-pruning requires 100% trainable param-

[1] https://gluebenchmark.com/

Method	%Params	CoLA	SST-2	MRPC	QNLI	RTE	STS-B	MNLI-m	MNLI-mm	QQP	Avg
$\text{BERT}_{\text{LARGE}}$	100%	60.5	94.9	89.3	92.7	70.1	87.6	86.7	85.9	72.1	82.2
Houlsby et al. (2019)	2.1%	56.9	94.2	89.6	91.4	68.8	87.3	85.3	84.6	71.8	81.1
Pfeiffer et al. (2020a)	20%	59.3	94.7	87.6	91.5	71.5	86.5	85.2	84.3	71.4	81.3
Guo et al. (2020)	0.5%	61.1	94.1	89.7	93.3	70.6	86.0	86.4	86.0	71.1	82.0
Ben Zaken et al. (2021)	0.08%	59.7	94.1	88.9	92.0	72.0	85.5	84.5	84.8	70.5	81.3
AdapterBias	**0.067%**	60.0	94.4	88.2	91.2	70.5	87.5	84.3	83.9	70.5	81.2

Table 1: Performance of all methods on the GLUE testing sets scored by the GLUE evaluation server. For each method, we report the new adding parameters per task. For QQP, we report the F1 score. For STS-B (Cer et al., 2017), we report Spearman correlation coefficients. For CoLA (Warstadt et al., 2019), we report Matthews correlation. For all other tasks, we report accuracy. Bold fonts indicate the least trainable parameter per task. The first row ($\text{BERT}_{\text{LARGE}}$) represents fine-tuning the whole BERT-large model without adding new parameters. The results of baselines including (Houlsby et al., 2019; Guo et al., 2020; Ben Zaken et al., 2021) are their reported performance and Pfeiffer et al. (2020a) performance is reproduced on our setting. Due to insatiability during training, we restart experiments with 3 random seeds and report the best.

	Method	%Params	CoLA	SST-2	MRPC	QNLI	RTE	STS-B	MNLI-m	MNLI-mm	QQP	Avg
BB	Full-FT	100%	52.1	93.5	88.9	90.5	66.4	85.8	84.6	83.4	71.2	79.6
BB	AdapterBias	0.075%	51.6	93.1	87.5	89.4	66.1	84.6	80.9	80.5	67.9	78.0
BL	Full-FT	100%	60.5	94.9	89.3	92.7	70.1	87.6	86.7	85.9	72.1	82.2
BL	AdapterBias	0.067%	60.0	94.4	88.2	91.2	70.5	87.5	84.3	83.9	70.5	81.2
RoB	Full-FT	100%	61.3	94.7	90.4	92.0	74.4	87.5	87.4	86.8	71.9	82.9
RoB	AdapterBias	0.066%	61.9	94.5	90.2	91.1	74.1	88.7	85.3	85.1	70.5	82.4
RoL	Full-FT	100%	63.3	96.7	92.3	95.4	84.5	92.2	90.8	90.2	74.3	86.6
RoL	AdapterBias	0.062%	63.9	96.4	90.4	94.7	83.6	91.3	89.8	89.4	72.3	85.8

Table 2: Performances of AdapterBias adding in different PLMs. Here we experiment four model : BERT-base (BB), BERT-large (BL), RoBERTa-base (RoB) and RoBERTa-large (RoL). The percentage of new parameters is compared with the PLM. The setting follows by Table 1. The Full-FT represents fine-tuning the whole PLM without adding adapters.

eters of BERT-large during the training stage, while AdapterBias only trains 0.062% parameters. Furthermore, AdapterBias achieves comparable performance with BitFit with fewer parameters needed per task. This shows that AdapterBias is a worthwhile targeted fine-tuning method.

4.3 Different base models

To analyze the generalization ability on different models of AdapterBias, as shown in Table 2, we apply AdapterBias in different transformer-based PLMs, including BERT-base (BB), BERT-large (BL), RoBERTa-base (RoB), and RoBERTa-large (RoL), on GLUE benchmark. All results are scored by the GLUE evaluate server. The percentage of new parameters per task is compared with the PLM. In Table 2, not only can AdapterBias perform well on BERT but also achieve competitive performance on larger PLMs such as RoBERTa.

4.4 Size of training data

In the previous experimental results, we observe that AdapterBias tends to have higher performance on tasks with a smaller amount of data (i.e. CoLA, SST-2, and RTE). To further validate this observation, we follow the work of BitFit (Ben Zaken et al., 2021) by training AdapterBias on increasing-sized subsets of SQuAD v1.0 (Rajpurkar et al., 2016). The experiments are conducted with BERT-base model. The results on the validation set of the SQuAD dataset are listed in Figure 4, which shows the tendency of AdapterBias outperforming fully fine-tuning when the size of the training dataset is smaller. However, with more training data available, the trend is reversed. The results show that AdapterBias has the ability to outperform fine-tuning the whole PLM with a small-to-medium data size similar to BitFit.

4.5 Investigation on the effectiveness of token dependent embedding shift

Different from BitFit (Ben Zaken et al., 2021), where the bias terms in all transformer layers are tuned, we claim that the bias added to the embedding should be token-dependent, and proposed AdapterBias based on this concept. We conduct ablation studies to verify this claim. In this experiment, the linear layer (L_α) in AdapterBias that

Method	%Params	CoLA	SST-2	MRPC	QNLI	RTE	STS-B	MNLI-m	MNLI-mm	QQP	Avg
w/o L_α	0.008%	45.6	91.5	87.4	88.3	65.6	81.0	77.9	78.4	65.7	75.7
AdapterBias	0.075%	**51.6**	**93.1**	**87.5**	**89.4**	**66.1**	**84.6**	**80.9**	**80.5**	**67.9**	**78.0**

Table 3: The importance of the linear layer (L_α) in AdapterBias. The setting follows by Table 1. The backbone model is BERT-base. w/o L_α means that there is only a vector (v) in AdapterBias.

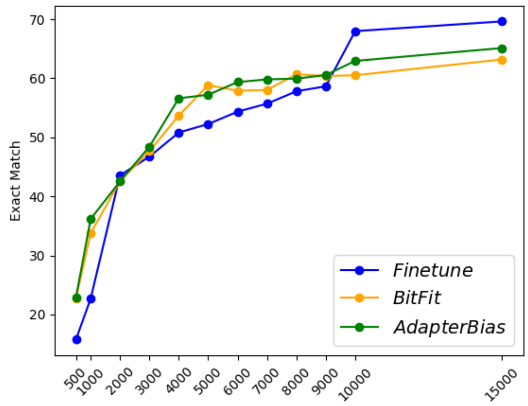

Figure 4: Comparison of Finetune, BitFit (Ben Zaken et al., 2021), and AdapterBias with BERT-base on SQuAD validation set. The x-axis represents the total number of training sets while the y-axis represents the exact match score.

produces the token-dependent weights vector (α) is removed; that is, only the v is trained. All shifts added to the embedding outputs are identical within the same transformer layer. The experiments are conducted with BERT-base model. We report the test scores on the GLUE benchmark in Table 3. The performance of AdapterBias without the linear layer (L_α) dramatically decreases. Without L_α, it is hard for the vector (v) to adapt to different downstream tasks. This result demonstrates the importance of L_α. In other words, assigning different shifts to different token embeddings improves the performance of the method.

4.6 More parameter-efficiency improvement in AdapterBias

We further apply two additional methods to AdapterBias to enhance its parameter efficiency. Experiments are conducted to see whether AdapterBias can be more parameter-efficient by sharing its components across all layers. Moreover, we experiment on adding L_0-norm regularization during the training stage to encourage the sparsity of AdapterBias.

4.6.1 Sharing components in AdapterBias

In this experiment, we conduct ablation study of partial weight-sharing in the AdapterBias module. In Table 4, we share components of AdapterBias among different transformer layers. *Share v* represents sharing v among AdapterBias across all transformer layers, while *Share* L_α means sharing the linear layer (L_α). *Share* v+L_α denotes sharing one AdapterBias among all transformer layers. As can be seen in Table 4, *Share* L_α stands out among other partial weight-sharing methods, while *Share v* leads to a poor performance.

From the experiments above, we conclude that the linear layer (L_α) captures general task information by learning the weights of the bias for different tokens. Thus, sharing L_α across all layers results in better performance compared to other components. The vector module (v) in AdapterBias aims to learn local information in each transformer layer. If v among different transformer layers are shared, the performance drops dramatically. This might due to v' failure to learn general information which can be adapted to each individual transformer layer.

4.6.2 L_0-norm regularization in AdapterBias

We observed that many of the trained parameters in L_α have values that are extremely close to zero after tuning on downstream tasks, which might cause redundancy of the parameters. To further encourage the sparsity of AdapterBias, we add L_0-norm regularization to L_α during the training stage.

In Table 4, we use BERT-base for the PLM. We compare the performance of the original AdapterBias and the one trained with L_0-norm regularization. The experiment shows that adding L_0-norm regularization during the training step improves the performance on 7 out of 9 tasks. In addition, the linear layer (L_α) with constraining L_0-norm penalty saves about 17% parameter on average compared to the original AdapterBias. We also experiment on BERT-large with L_0-norm regularization applied in the training stage, where the results are shown in Appendix A.

Method	%Params	CoLA	SST-2	MRPC	QNLI	RTE	STS-B	MNLI-m	MNLI-mm	QQP	Avg
Share v	0.068%	50.1	90.8	87.1	87.6	65.0	84.9	77.5	77.9	65.1	76.2
Share L_α	0.045%	50.4	91.9	**88.1**	89.1	65.4	85.2	79.8	79.9	66.6	77.4
Share v+L_α	0.037%	46.8	90.9	87.3	87.8	64.8	**85.7**	77.7	78.0	64.9	76.0
AdapterBias (L0)	0.062%	**53.7**	92.5	87.5	**90.3**	**68.3**	85.7	**81.7**	**81.5**	**69.8**	**79.0**
AdapterBias	0.075%	51.6	**93.1**	87.5	89.4	66.1	84.6	80.9	80.5	67.9	78.0

Table 4: Analysis of more parameter-efficiency improvement in AdapterBias. The setting follows by Table 1. The backbone model is BERT-base. Share v, Share L_α and Share v+L_α means that we share vector, linear layer, and both of them, respectively. AdapterBias (L0) means that we constrain the linear layer L_α with L_0-norm regularization.

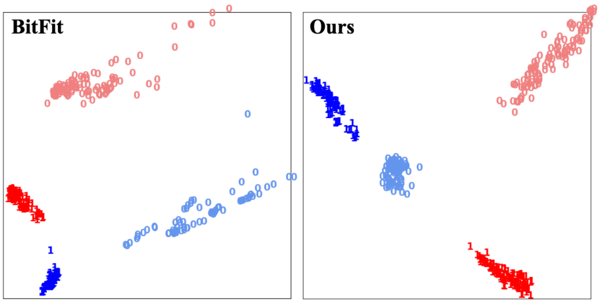

Figure 5: We utilize PCA (Jolliffe, 2002) to visualize the shifting difference between Bitfit (Ben Zaken et al., 2021) and AdapterBias on SST-2 evaluate set. '0' with light color means the embedding before shifting. '1' with dark color means the embedding after shifting. The color red represents positive sentences, and blue represents negative sentences.

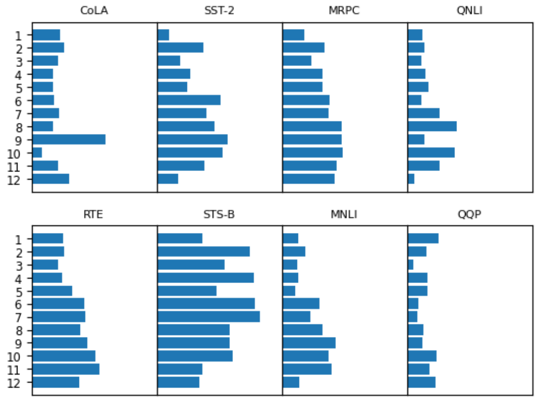

Figure 6: We analyze the average absolute value of weights vector α, the output of the linear layer (L_α), in each layer for different tasks. The y-axis represents the index of transformer layers, ordered from earlier to later (i.e. the embedding layer is shown at the top). The x-axis represents the average absolute value of α.

4.7 Deeper look on what AdapterBias have learned

AdapterBias has good interpretability due to its simplicity. Compared to our similar work BitFit (Ben Zaken et al., 2021), where the shifts are identical for all tokens, AdapterBias adds token-dependent shifts to the output embedding. By observing these token-dependent shifts, we provide analysis of what AdapterBias has learned when adapting to downstream tasks.

4.7.1 The direction of embedding shifts in different tasks

Different from BitFit (Ben Zaken et al., 2021), where all the embedding shifts are identical within one task, AdapterBias produces different weights for the shift based on each token. In this section, we compare the transformed tokens in AdapterBias and BitFit. We utilize PCA (Jolliffe, 2002) to reduce the dimension of the tokens. In Figure 5, we input five sentences from the evaluation set of SST-2. We experiment on the last transformer layer since it has the most obvious shifts compared to the previous layers. '0' with lighter color indicates the embedding before shifting, which is the output of the first layer normalization. '1' with darker color is the shifted embedding, which is the output of the second layer normalization. The color red represents positive sentences, and blue are the negative ones.

The result shows that BitFit shifts all tokens towards the same direction regardless of the ground-truth label. On the other hand, AdapterBias discerns the label of the sentences and thus shifts the tokens of different sentences toward different directions.

4.7.2 Average embedding shifting in transformer layers

In light of the works of Liu et al. (2019a); Tenney et al. (2019); Kovaleva et al. (2019), different information has been encoded by different transformer layers of PLMs. We assume that AdapterBias provides different embedding shifts to the transformer layers through task-specific fine-tuning. In AdapterBias, the linear layer (L_α) produces a weights vec-

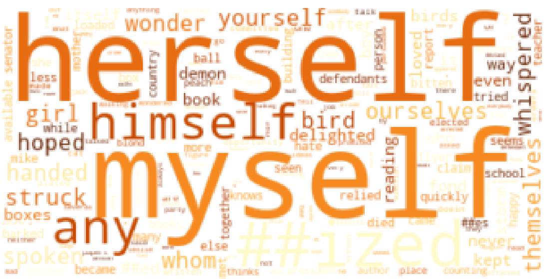

Figure 7: WordCloud of CoLA, a corpus of linguistic acceptability. We utilize BERT-base model as the PLM and words come from validation data. The weights of the words are the summation of their weights produced by the linear layer (L_α) in twelve transformer layers.

Figure 8: WordCloud of SST-2, a corpus of movie reviews categorized in two sentimental classes (i.e. positive, negative). The visualization approach is the same as the Figure 7.

tor α for embedding shifts, therefore, the average absolute value of vector α can give us a look at the shifting amount in the transformer layers when adapting to downstream tasks. In Figure 6, the layers are ordered from lower to upper. From the experimental result, we find that the weight in each layer is considerably different in different tasks in general.

CoLA (Warstadt et al., 2019) is the only syntactic task that consists of English acceptability judgments in the GLUE benchmark. As shown in Figure 6, its average shift at the ninth layer is the highest among all layers, which is quite different from the others. We speculate that the ninth layer has the ability to extract the syntactic information, leading AdapterBias to add the largest shift in this layer. Our experiment has a similar observation with the work of Jawahar et al. (2019). In their findings on BShift (Conneau et al., 2018), which is also a syntactic task, the ninth layer of BERT embeds a rich hierarchy of syntactic information. (Jawahar et al., 2019)

Moreover, we observe similar distributions between specific tasks. For instance, RTE (Giampiccolo et al., 2007; Bentivogli et al., 2009) and MNLI (Williams et al., 2017), where both tasks recognize textual entailment, have higher values in the upper layers than those in the lower ones.

Based on these findings, we find that AdapterBias assigns suitable embedding shifts in different tasks. For tasks with similar objectives, AdapterBias tends to add similar embedding shifts.

4.7.3 Which kind of word does L_α focus on

Since α_i represents the weight of the embedding shift for ith token in a transformer layer, we can observe the significance of ith token from the summation of α_i in all the transformer layers. Special tokens, including [CLS], [SEP], and [PAD], are not included for analysis. We use the validation sets of CoLA and SST-2, and WordCloud is used for visualizations.

In Figure 7, we visualize all words in the validation data of CoLA. The result shows that AdapterBias focuses more on reflexive pronouns, such as yourself, himself, and myself. This is because there are many incorrect sentences with misused reflexive pronouns, such as "He washed yourself."

In Figure 8, we visualize all words in the validation data of SST-2. The result shows that AdapterBias focuses more on adjectives, such as "bad", "awful", and "worst". SST-2 is a binary sentiment analysis dataset, which classifies movie reviews into positive and negative classes. AdapterBias learns that adjectives often constitute a crucial factor in sentiment analysis during tuning, and adds larger shifts to these adjective tokens.

5 Conclusion

In this study, we present AdapterBias. By adding token-dependent embedding shifts to the PLM, AdapterBias shows competitive results when using far less trainable parameters than the existing methods. Through extensive experiments, not only does AdapterBias reaches competitive results on the GLUE benchmark, but it also obtains good performance on small-to-medium datasets. In addition, we demonstrate the robustness of AdapterBias in different PLMs. Finally, we provide analysis on what AdapterBias have learned by comparing α, the weights of embedding shift from different tokens, finding it has the ability to identify task-specific information. Our study overturns previous architectures of adapters by proposing a simple adapter that can produce suitable embedding shifts for different tokens.

References

Ankur Bapna, Naveen Arivazhagan, and Orhan Firat. 2019. Simple, scalable adaptation for neural machine translation. *arXiv preprint arXiv:1909.08478*.

Elad Ben Zaken, Shauli Ravfogel, and Yoav Goldberg. 2021. Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. *arXiv e-prints*, pages arXiv–2106.

Luisa Bentivogli, Peter Clark, Ido Dagan, and Danilo Giampiccolo. 2009. The fifth pascal recognizing textual entailment challenge. In *TAC*.

Rich Caruana. 1997. Multitask learning. *Machine learning*, 28(1):41–75.

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

Alexis Conneau, German Kruszewski, Guillaume Lample, Loïc Barrault, and Marco Baroni. 2018. What you can cram into a single vector: Probing sentence embeddings for linguistic properties. *arXiv preprint arXiv:1805.01070*.

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*.

Danilo Giampiccolo, Bernardo Magnini, Ido Dagan, and William B Dolan. 2007. The third pascal recognizing textual entailment challenge. In *Proceedings of the ACL-PASCAL workshop on textual entailment and paraphrasing*, pages 1–9.

Demi Guo, Alexander M Rush, and Yoon Kim. 2020. Parameter-efficient transfer learning with diff pruning. *arXiv preprint arXiv:2012.07463*.

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.

Eric Jang, Shixiang Gu, and Ben Poole. 2016. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*.

Ganesh Jawahar, Benoît Sagot, and Djamé Seddah. 2019. What does bert learn about the structure of language? In *ACL 2019-57th Annual Meeting of the Association for Computational Linguistics*.

Ian T Jolliffe. 2002. Springer series in statistics. *Principal component analysis*, 29.

Olga Kovaleva, Alexey Romanov, Anna Rogers, and Anna Rumshisky. 2019. Revealing the dark secrets of bert. *arXiv preprint arXiv:1908.08593*.

Nelson F Liu, Matt Gardner, Yonatan Belinkov, Matthew E Peters, and Noah A Smith. 2019a. Linguistic knowledge and transferability of contextual representations. *arXiv preprint arXiv:1903.08855*.

Xiaodong Liu, Pengcheng He, Weizhu Chen, and Jianfeng Gao. 2019b. Multi-task deep neural networks for natural language understanding. *arXiv preprint arXiv:1901.11504*.

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

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

Christos Louizos, Max Welling, and Diederik P Kingma. 2017. Learning sparse neural networks through l_0 regularization. *arXiv preprint arXiv:1712.01312*.

Chris J Maddison, Andriy Mnih, and Yee Whye Teh. 2016. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*.

Rabeeh Karimi Mahabadi, James Henderson, and Sebastian Ruder. 2021. Compacter: Efficient low-rank hypercomplex adapter layers. *arXiv preprint arXiv:2106.04647*.

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, Andreas Rücklé, Clifton Poth, Aishwarya Kamath, Ivan Vulić, Sebastian Ruder, Kyunghyun Cho, and Iryna Gurevych. 2020b. Adapterhub: A framework for adapting transformers. *arXiv preprint arXiv:2007.07779*.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*.

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

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642.

Asa Cooper Stickland and Iain Murray. 2019. Bert and pals: Projected attention layers for efficient adaptation in multi-task learning. In *International Conference on Machine Learning*, pages 5986–5995. PMLR.

Ian Tenney, Dipanjan Das, and Ellie Pavlick. 2019. Bert rediscovers the classical nlp pipeline. *arXiv preprint arXiv:1905.05950*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *arXiv preprint arXiv:1706.03762*.

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

Alex Warstadt, Amanpreet Singh, and Samuel R Bowman. 2019. Neural network acceptability judgments. *Transactions of the Association for Computational Linguistics*, 7:625–641.

Adina Williams, Nikita Nangia, and Samuel R Bowman. 2017. A broad-coverage challenge corpus for sentence understanding through inference. *arXiv preprint arXiv:1704.05426*.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. 2019. Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*.

Yu Zhang and Qiang Yang. 2017. A survey on multi-task learning. *arXiv preprint arXiv:1707.08114*.

A Example Appendix

	CoLA	SST-2	MRPC	QNLI	RTE	STS-B	MNLI-m	MNLI-mm	QQP
Max_len	128	128	128	512	350	512	128	128	350
Batchsize	32	32	32	16	32	16	32	32	32
Learning rate	10^{-3}	10^{-3}	10^{-3}	10^{-4}	4×10^{-4}	10^{-3}	4×10^{-4}	4×10^{-4}	4×10^{-4}
Epoch	20	10	10	10	20	20	10	10	10

Table 5: Our training details of GLUE benchmark(Wang et al., 2018).

	Method	CoLA	SST-2	MRPC	QNLI	RTE	STS-B	MNLI-m	MNLI-mm	QQP	Avg
BB	Full-FT	52.1	93.5	88.9	90.5	66.4	85.8	84.6	83.4	71.2	79.6
BB	AdapterBias	51.6	93.1	87.5	89.4	66.1	84.6	80.9	80.5	67.9	78.0
BB	AdapterBias (L0)	53.7	92.5	87.5	90.3	68.3	85.7	81.7	81.5	69.8	79.0
BL	Full-FT	60.5	94.9	89.3	92.7	70.1	87.6	86.7	85.9	72.1	82.2
BL	AdapterBias	60.0	94.4	88.2	91.2	70.5	87.5	84.3	83.9	70.5	81.2
BL	AdapterBias (L0)	58.0	93.7	88.2	91.5	69.2	87.2	84.2	84.1	71.2	80.8

Table 6: Performances of our AdapterBias with L_0-norm regularization. Here we experiment with two models: BERT-base (BB) and BERT-large (BL). The setting follows by Table 1. The Full-FT represents fine-tuning the whole PLM without adding adapters.

	Method	CoLA	SST-2	MRPC	QNLI	RTE	STS-B	MNLI-m	MNLI-mm	QQP
BB	AdapterBias (L0)	26.2%	82.0%	83.1%	82.3%	81.0%	83.0%	83.2%	83.3%	83.4%
BL	AdapterBias (L0)	83.2%	83.0%	83.3%	83.7%	83.2%	83.2%	83.4%	83.7%	83.6%

Table 7: Percentage of remaining parameters compared with the original parameters of the linear layer (L_α). Here we experiment with two models: BERT-base (BB) and BERT-large (BL). The setting follows by Table 1.