Generalizable and Stable Finetuning of Pretrained Language Models on Low-Resource Texts

Sai Ashish Somayajula **Youwei Liang** **Li Zhang** **Abhishek Singh** **Pengtao Xie**
UC San Diego, USA
{ssomayaj, p1xie}@ucsd.edu

Abstract

Pretrained Language Models (PLMs) have advanced Natural Language Processing (NLP) tasks significantly, but finetuning PLMs on low-resource datasets poses significant challenges such as instability and overfitting. Previous methods tackle these issues by finetuning a strategically chosen subnetwork on a downstream task, while keeping the remaining weights fixed to the pretrained weights. However, they rely on a suboptimal criteria for sub-network selection, leading to suboptimal solutions. To address these limitations, we propose a regularization method based on attention-guided weight mixup for finetuning PLMs. Our approach represents each network weight as a mixup of task-specific weight and pretrained weight, controlled by a learnable attention parameter, providing finer control over sub-network selection. Furthermore, we employ a bi-level optimization (BLO) based framework on two separate splits of the training dataset, improving generalization and combating overfitting. We validate the efficacy of our proposed method through extensive experiments, demonstrating its superiority over previous methods, particularly in the context of finetuning PLMs on low-resource datasets. Our code is available at https://github.com/Sai-Ashish/Attention_guided_weight_mixup_BLO.

1 Introduction

Pretraining large language models on a large corpus of unlabeled texts and further finetuning them on downstream tasks have been a common practice in natural language processing (NLP), resulting in significant advances in tasks such as sentiment classification, natural language inference, and text generation (Devlin et al., 2018; Liu et al., 2019; Lewis et al., 2019; Raffel et al., 2020).

However, conventional finetuning of pretrained language models (PLMs) presents several challenges. First, PLMs are prone to instability in finetuning, characterized by high variance in performance for different weight initializations even when using the same hyperparameters, especially on small datasets (Ziser and Reichart, 2019; Devlin et al., 2018; Phang et al., 2018; Lee et al., 2019; Dodge et al., 2020; Zhang et al., 2020). Moreover, PLMs, due to their extremely large capacity, are prone to overfitting when finetuned on small downstream datasets, leading to poor generalization on test set (Belinkov et al., 2020; Aghajanyan et al., 2020; Kuang et al., 2021). Consequently, adapting PLMs to a variety of low-resource tasks, while preserving stability and maximizing generalization, remains a significant challenge in the field.

Finetuning a strategically chosen sub-network on a downstream task, while keeping the remaining weights fixed to the pretrained weights, has effectively mitigated these challenges. Within this umbrella, CHILD-TUNING$_D$ (Xu et al., 2021) and DPS dense (Zhang et al., 2022) are promising. In CHILD-TUNING$_D$ (Xu et al., 2021), a static sub-network, termed "child network", is first selected based on the Fisher Information Matrix (FIM) and this child network is subsequently updated during finetuning. Dynamic Parameter Selection (DPS) (Zhang et al., 2022) further refines this approach by dynamically selecting the child network during PLM finetuning using FIM as a guiding principle.

Nevertheless, these prior works exhibit certain limitations. FIM, which is empirically calculated using the training dataset to identify important network parameters, may not be optimal for child network selection, especially in low-resource settings where data scarcity can skew the gradient that is used to compute FIM (Kunstner et al., 2019). This scenario can lead to unimportant parameters being subsequently chosen in the sub-network, deteriorating performance on downstream tasks. Moreover, Soen and Sun (2021) theoretically shows that empirically determined FIM deviates significantly from the true FIM when the number of samples is

Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers), pages 4936–4953

low. Thus, the discrete selection of child networks based on heuristics (i.e., FIM), may result in selection of suboptimal child networks for downstream tasks. These limitations necessitate a departure from FIM-based discrete child network selection strategies in favor of one that selects a child network based on the model's downstream task performance. Consequently, we advocate for a continuous optimization approach for child network selection that does not rely on FIM, guided by the goal of optimizing performance in downstream tasks.

In this work, we propose an end-to-end framework that converts prior heuristic-based approaches to discrete child network selection into a continuous relaxation that can be optimized using gradient descent. The crux of our method is an attention-guided weight mixup mechanism that facilitates this transformation. Each weight is represented as a weighted sum of task weights (from downstream task finetuning) and pretrained weights (which is frozen), controlled by a mixing coefficient, which we refer to as "attention parameter" [1]. This setup leads to a continuous relaxation: a larger attention value of a weight indicates that it is more likely to belong to the child network and vice versa. We formulate the learning of task weights and the attention parameters in a bi-level optimization (BLO) framework (Feurer et al., 2015) on two different splits of the training dataset. In the lower level of our formulation, we update task weights by minimizing loss on the first split, while in the upper level of the framework, we update the attention parameter by minimizing loss on the second split. This bi-level learning of task weights and the attention parameters on two different splits of the training dataset sidesteps the FIM-based heuristic, ensuring it is more adapted to the downstream task, leading to better performance (Sec. 4).

Our major contributions are summarized below:

- We address the crucial issue of finetuning PLMs on low-resource datasets by leveraging an attention-guided weights mixup strategy. In this approach, each weight is represented as a mixup of task weights and pretrained weights, controlled by an attention parameter. This method is a continuous relaxation of the prior discrete sub-network selection approach.
- We capitalize on an inter-dependency between task weights and attention parameter to formulate the learning objective as a BLO problem, which allows us to learn the attention parameter and model weights on two separate splits of the training set respectively. This has been a key of our method to combating overfitting and increasing stability.
- We extensively evaluate our method on several datasets of the GLUE benchmark in low-resource settings, demonstrating improvements over several baselines. Our method has also achieved enhanced stability over standard finetuning across different PLMs.

[1]This attention parameter, α, is *not* to be confused with the attention layers of the transformers.

2 Related works

2.1 Generalizable finetuning

Finetuning PLMs on small downstream datasets can lead to overfitting and instability. Several techniques have been proposed to address this issue. Weight decay (Daumé III, 2009) incorporates a regularization term with a fixed trade-off coefficient to mitigate overfitting, while RecAdam (Chen et al., 2020) improves weight decay by introducing a time-varying trade-off coefficient for regularization loss, and an L2 loss between task and pretrained weights, which is used to regularize the finetuning. Top-K-layer finetuning (Houlsby et al., 2019) focuses on updating the weights of the top K layers and keeping the pretrained bottom layer weights intact, thereby regularizing the finetuning. R3F (Aghajanyan et al., 2020) introduces parametric noise into input sentence embeddings for better generalization. R-Dropout (Wu et al., 2021) minimizes the Kullback-Leibler divergence of the predictions from two sub-models created by different dropouts, thereby fostering prediction consistency. Re-init (Zhang et al., 2020) reinitializes the pooler and top K Transformer layers before finetuning BERT, which is found to perform better than vanilla finetuning. Mixout (Lee et al., 2019) proposes to randomly replace task weights with their corresponding pretrained weights to mitigate overfitting and improve stability.

Instead of randomly replacing some task weights, sub-network optimization methods such as CHILD-TUNING$_D$ (Xu et al., 2021) and DPS dense (Zhang et al., 2022) leverage FIM to select a subset of the model parameters that are relevant to downstream tasks to finetune, resulting in better performance than Mixout and other methods above. However, FIM-based methods have some

limitations especially in low-resource scenarios. To address these issues, we propose to use the model performance on the downstream task to select the sub-network to finetune. To overcome overfitting, we propose to update the task weights on a training set and use the model performance on a separate validation set for the sub-network selection, resulting in a BLO framework. Appendix C has more details about each method.

2.2 Bi-level optimization (BLO)

BLO refers to a class of optimization problems in which one optimization problem (lower level) is nested within another optimization problem (upper level) (Sinha et al., 2017). Many problems in machine learning, including neural architecture search (Liu et al., 2018), hyperparameter tuning (Feurer et al., 2015; Baydin et al., 2017), data selection (Shu et al., 2019; Wang et al., 2020; Ren et al., 2020), meta-learning (Finn et al., 2017), and noisy label correction (Baydin et al., 2017), can be *formulated* as a BLO problem. In these applications, the upper-level optimization is responsible for learning meta-variables, such as hyperparameters and architectures, by minimizing the validation loss, while the lower-level optimization focuses on learning model weights by minimizing the training loss. Although these works, as well as ours, are formulated as BLO problems, our contributions are notably different since we target different domains. Recognizing the limitations of existing sub-network optimization methods, we introduce a continuous relaxation approach that utilizes an attention-guided weight mixup strategy. Subsequently, we suggest employing BLO, capitalizing on the problem's structure as detailed in Section 3.

3 Method

We introduce "Attention-Guided Weights Mixup", depicted in Fig 1, an approach to improving stability and performance on small downstream datasets in PLMs. Central to our approach is a unique representation of weights as a blend of the pretrained weights (those prior to finetuning) and the task weights (post-downstream task finetuning), modulated by an "attention parameter". This coefficient represents the degree of emphasis or "attention" to be placed on a pretrained weight during the computation of the resultant weight. To mitigate overfitting, we propose to learn the attention parameter and the task weights using BLO on different splits of the training dataset; pretrained weights are frozen. This initial phase, termed search phase, aims to find optimal attention parameters, i.e., to find the optimal child network. Subsequently, in the finetune phase, the task weights are further adjusted using the entire training dataset and the learned attention parameters.

3.1 Continuous relaxation of child network selection

To overcome the limitations of FIM-based discrete child network selection methods, we introduce a unique continuous relaxation approach. This approach is based on an attention-guided weight mixup mechanism. To elaborate on this formulation, consider the task weights and the pretrained weights of a PLM, denoted by $W \in \mathcal{R}^{N\times M}$ and $W_0 \in \mathcal{R}^{N\times M}$ respectively, where N and M are the dimensions of the weight. An attention parameter, denoted by $\alpha \in [0,1]^{N\times M}$, is associated with each pretrained weight. This parameter indicates the relative importance of the pretrained weight compared to the task weight. We compute the resultant weight, $\tilde{W}$. This weight is represented as an interpolation between the task and pretrained weights, the balance of which is dictated by the respective attention parameter:

$$\tilde{W} = g(W, \alpha, W_0) = \alpha \odot W + (\mathbf{1} - \alpha) \odot W_0$$

where, $\odot$ denotes the element-wise multiplication operation, and $\mathbf{1} \in \mathcal{R}^{N\times M}$ denotes the matrix with all its entries 1's. The discrete child network selection can be perceived as a special case within this formulation. To elaborate, if α equals $\mathbf{1}$, the implication is that the weight belongs to the child network. Conversely, if α is $\mathbf{0}$ (the matrix with all its entries 0's), the weight is tied to the pretrained, non-child network. Nonetheless, the entries of α are not restricted to the extremes of 0 or 1; instead, they can assume any continuous value within this range. If the learned attention parameter, α, leans closer to $\mathbf{1}$, it signals a predominant influence from the task weight compared to the pretrained counterpart and vice versa. Thus, α serves as a mechanism that allows a flexible transition from the discrete child network selection to a continuous scale. In particular, α is designed to regulate the balance between the pretrained and task weights in the computation of the resultant weight.

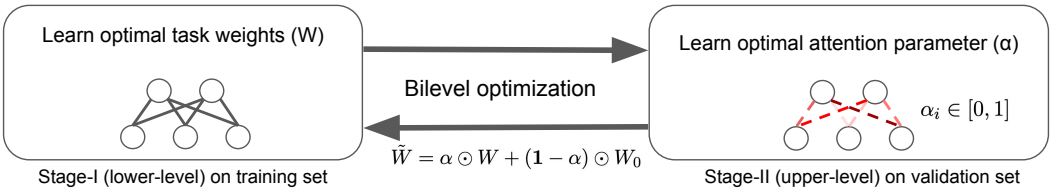

Figure 1: An overview of our proposed method: learning the task weights W and the attention parameter α in a bilevel optimization framework. The final network weight $\tilde{W}$ is a combination of the pretrained weight W_0 and the task weight W via the learned attention parameter α.

3.2 Learning task weight and attention parameter via bi-level optimization

With continuous relaxation, the task of child network selection without using FIM morphs into determining attention parameters. Consequently, the task weights become dependent on these attention parameters, i.e. the chosen child network. However, in a reciprocal relationship, task weights should be considered while learning the attention parameters. This is because the attention parameters aim to ascertain an optimal blend of pretrained and task weights in the resultant weight computation, engendering a mutual dependency.

Navigating this intricate interdependency calls for a nuanced approach. Our approach to optimizing task weights W and attention parameters α leverages a BLO framework composed of two interdependent learning stages. In the first stage, the finetuning of task weights occurs, driven by the minimization of the training loss. Subsequently, in the second stage, the attention weights are updated to minimize the model's validation loss. Given the optimal task weights W^* on the training set in the first stage, the goal of BLO is to learn the optimal attention parameters α^* that determines the right mix of W^* and W_0 in the resultant weight estimation by minimizing the validation loss.

To facilitate this, we partition the original training dataset, represented as $\mathcal{D}^{\text{tr}}$, into two subsets of 80% and 20% split. 50%-50% split setting was explored however 80%-20% split gave better empirical results (more insights in Appendix D). The first subset, the BLO training dataset ($\mathcal{D}^{\text{B-tr}}$), is utilized in the first stage. The second subset, the BLO validation dataset ($\mathcal{D}^{\text{B-val}}$), is used in the second stage. While this two-stage optimization process unfolds, we freeze the pretrained weights. By doing so, our approach provides finer control over network optimization and capitalizes on the potential of pretrained weights. This phase to learn the parameter importance and the model parameters is termed the "search phase." This phase is followed by the "finetune phase" that further learns W with the learned α fixed on the entire training dataset.

Stage I - learning W In this stage, we solve for the optimal task weights on BLO training dataset $\mathcal{D}^{\text{B-tr}}$. The task weights W are learned by minimizing the training loss. As explained above, each weight of the PLM is represented as an interpolation of the task weight W and the pretrained weight W_0 weighted by an attention parameter $\alpha, \alpha_{ij} \in [0, 1], \forall i, j$. The following optimization problem is solved at this level,

$$W^*(\alpha) = \arg\min_{W} \mathcal{L}(g(W, \alpha, W_0); \mathcal{D}^{\text{B-tr}}) + \lambda_1 \|W\|_F^2 \quad (1)$$

where $\mathcal{L}(\cdot)$ is the cross-entropy loss, α denotes the attention parameter, λ_1 is the weight decay of W, and $\|\cdot\|_F$ is the Frobenius norm. Weight decay is commonly used while finetuning PLMs and we too introduce a weight decay term on the task weights W. The optimal task weights, denoted by $W^*(\alpha)$, are a function of α. This dependency is enforced because the optimal weights are learned by minimizing the loss in Equation 1 on $\mathcal{D}^{\text{B-tr}}$, which is a function of α. The attention parameter α is not learned in this stage else a complex solution will be learned that overfits the BLO training dataset.

Stage II - learning α In this stage, the attention parameters are learned on the BLO validation dataset $\mathcal{D}^{\text{B-val}}$, given the optimal task weights learned in the previous stage. The model $W^*(\alpha)$ is evaluated on $\mathcal{D}^{\text{B-val}}$. The model's validation loss is a function of α. The attention parameters are learned by minimizing the validation loss (a function of α). The following optimization problem is solved at this stage,

$$\min_{\alpha} \quad \mathcal{L}(g(W^*(\alpha), \alpha, W_0); \mathcal{D}^{\text{B-val}}) + \lambda_2 \|\alpha\|_F^2 \quad (2)$$

where λ_2 is the weight decay of α. Weight decay on α encourages the values of α to be close to **0** en-

couraging a higher contribution from the pretrained weights in the resultant weight estimation.

Low-rank approximation of α Since we mainly target the low-resource domain, we use a low-rank approximation of $\alpha \in [0,1]^{N\times M}$ to mitigate overfitting (Hu et al., 2021). Specifically, we express the α matrix as the product of two matrices with lower ranks (of rank r):

$$\alpha = \frac{1}{r} \odot \mathcal{F}(\alpha_1, \alpha_2) \tag{3}$$

where $\mathcal{F}(\cdot,\cdot)$ denotes matrix multiplication, and both $\alpha_1 \in \mathbb{R}^{N\times r}$ and $\alpha_2 \in \mathbb{R}^{r\times M}$ have a rank of r. We restrict $\{\alpha_1\}_{ij} \in [0,1], \forall i,j$ and $\{\alpha_2\}_{ij} \in [0,1], \forall i,j$. The normalization constant r in Eq. 3 can ensure the entries of α are in [0,1]. We use the following weight formulation that proves to have better performance empirically after rank decomposition:

$$g(W,\alpha,W_0) = \mathcal{F}(\alpha_1,\alpha_2)\odot W + \mathcal{F}(1-\alpha_1, 1-\alpha_2)\odot W_0$$

Our bi-level optimization framework Following BLO is formulated,

$$\begin{aligned} \min_{\alpha} \quad & \mathcal{L}(g(W^*(\alpha),\alpha,W_0);\mathcal{D}^{\text{B-val}}) + \lambda_2 \|\alpha\|_F^2 \\ \text{s.t.} \quad & W^*(\alpha) = \arg\min_{W} \big[\mathcal{L}(g(W,\alpha,W_0);\mathcal{D}^{\text{B-tr}}) \\ & \quad +\lambda_1 \|W\|_F^2\big] \end{aligned} \tag{4}$$

There are two optimization problems each corresponding to a learning stage. From bottom to top, each learning stage corresponds to stage I and stage II that are dependent on each other via the loss function. Both learning stages are performed in an end-to-end fashion. The optimal task weights $W^*(\alpha)$ are learned by minimizing the loss function defined on $\mathcal{D}^{\text{B-tr}}$. The optimal task weights are a function of attention parameters α because the training loss is dependent on α via the resultant weight definition defined above. In the second stage, attention parameters α are learned by minimizing the validation loss on $\mathcal{D}^{\text{B-val}}$. This would influence the training loss in stage I, which influences the solution $W^*(\alpha)$.

Optimization algorithm Our approach consists of two phases, shown in Algorithm 1. (I) Search phase: The optimal attention parameters are estimated in this phase posed as a BLO (4). We calculate the gradient of Eq.1 with respect to W and approximately update $W^*(\alpha)$ using one-step gradient descent. Similarly, we update α using one-step gradient descent of Eq.2; however, we encounter a hessian-vector product while estimating the gradient due to the chain rule, which is computationally expensive to calculate. This is approximated using finite-difference approximation (Liu et al., 2018), as described in detail in Appendix A. The algorithm learns $\alpha' \approx \alpha^*$. (II) Finetune phase: In this phase, we further finetune the task weights on entire training dataset ($\mathcal{D}^{\text{tr}}$) using the estimated attention parameters α'.

4 Experiments

4.1 Datasets

We conduct experiments on various datasets from the GLUE benchmark (Warstadt et al., 2018; Wang et al., 2018) following Xu et al. (2021); Zhang et al. (2022). The datasets we have chosen for our evaluation cover an extensive range of linguistic tasks. These include sentiment classification (SST-2), natural language inference (RTE, QNLI, MNLI), paraphrasing and similarity assessment (MRPC, STS-B, QQP), and finally, the evaluation of linguistic acceptability (CoLA). We finetune the models on the training set for each of the mentioned datasets and evaluated its performance on the original development set using the checkpoint obtained at the end of training following Xu et al. (2021); Zhang et al. (2022). More information about each of the chosen datasets can be found in Appendix B.

4.2 Experimental setup

We use the PLM codes provided by Huggingface (Wolf et al., 2020) and follow their default settings unless specifically mentioned. Rank of α, r is set to 1 in this work. More detailed information about the hyperparameters settings in the search and the finetune phase, such as batch size, training steps, for $\text{BERT}_{\text{LARGE}}$ (Devlin et al., 2018), $\text{RoBERTa}_{\text{LARGE}}$ (Liu et al., 2019), $\text{BART}_{\text{LARGE}}$ (Lewis et al., 2019), $\text{DeBERTa}_{\text{LARGE}}$ (He et al., 2020), and $\text{XLNet}_{\text{LARGE}}$ (Yang et al., 2019) can be found in Appendix D. The averaged results over ten random seeds are reported in the paper. In this work, we primarily compare with CHILD-TUNING$_D$ (Xu et al., 2021) and DPS dense (Zhang et al., 2022), as these methods share a similar motivation to ours: improving performance on small downstream tasks by finetuning on a strategically chosen sub-network.

Training split	Vanilla	CHILD-TUNING$_D$	DPS Dense	Ours
300	62.54 ± 6.57	62.47 ± 5.5	61.69 ± 5.62	**68.97** ± 3.09
500	65.85 ± 4.57	68.35 ± 4.36	68.99 ± 2.92	**72.42** ± 2.14
1000	73.19 ± 2.62	74.07 ± 2.75	75.00 ± 1.61	**76.68** ± 1.58

Table 1: We compare our method with Vanilla, CHILD-TUNING$_D$, and DPS dense method using BERT$_{\text{LARGE}}$ (Lee et al., 2019) across 300, 500, and 1000 training data splits. Reported results are the averaged evaluation metrics over all eight GLUE datasets for each training data split. Due to space constraints, detailed results for each of the eight datasets can be found in Table 8. The highest performance in each row is indicated in **bold**.

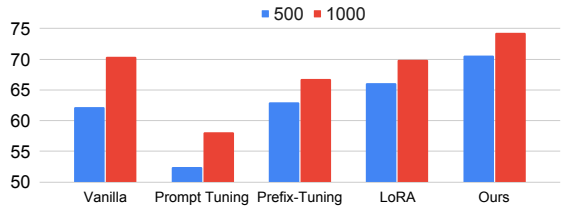

Figure 2: Averaged performance across CoLA, RTE, STSB, and MRPC datasets for Vanilla, Prompt Tuning, Prefix-Tuning, LoRA, and our method in low-resource scenarios with 500 and 1000 training instances. Results on each dataset are presented in Table 9.

4.3 Performance on low-resource scenarios

In this experiment, we investigate the effectiveness of our proposed method for finetuning PLMs on extremely small datasets. Specifically, we downsample 8 GLUE datasets by randomly selecting 300, 500 and 1K training examples following Zhang et al. (2022). [2] Table 1 summarizes our results. Comparing our approach to DPS dense and CHILD-TUNING$_D$, we observe substantial improvements in average scores. On average, our method outperforms the best baseline by 6.43%, 3.43%, and 1.68% on 300, 500, and 1K training samples scenarios, respectively.

The superior performance of our method can be attributed to two key designs in our framework. First, our method employs continuous weight mixup, assigning attention parameters to each pretrained weight. This allows for a more adaptable and dynamic determination of the importance of pretrained weight during finetuning. Second, our BLO framework strengthens our method by optimizing task weights and attention parameters on two distinct splits of the training dataset. This strategy effectively combats overfitting and ensures that attention parameters are learned based on validation performance, leading to improved generalization on unseen test data. To summarize, our method shows strong potential for improving performance on low-resource NLP tasks.

4.4 Comparison with parameter efficient finetuning methods

In this section, we compare our method with popular parameter-efficient finetuning (PEFT) techniques, including Prompt Tuning (Lester et al., 2021), Prefix-Tuning (Li and Liang, 2021), and LoRA (Hu et al., 2021). Our experiments were conducted on MRPC, STSB, CoLA, and RTE datasets under two low-resource settings with 500 and 1,000 data points, respectively. As shown in Figure 2, our method consistently outperforms these baselines. Considering performance on each dataset (Table 9), LoRA lags behind vanilla finetuning on all datasets except CoLA. Prefix-Tuning underperforms compared to both vanilla finetuning and LoRA in most scenarios. The deteriorated performance of Prefix-Tuning is also observed in other low-resource settings (Hu et al., 2021). Besides, Prompt Tuning falls behind vanilla finetuning on all datasets (Table 9). On average, as depicted in Figure 2, Prompt Tuning trails vanilla finetuning by a significant margin. The low performance of Prompt Tuning is particularly evident in models with fewer than 10 billion parameters (Lester et al., 2021).

It is worth noting that the primary aim of these PEFT methods is not necessarily to outperform vanilla finetuning in low-resource settings. Instead, they seek to deliver competitive performance without intensive resource utilization. Such methods are especially valuable when finetuning large models like GPT-2 (Radford et al., 2019) (774M) and GPT-3 (Brown et al., 2020) (175B). They finetune a few additional parameters while preserving the primary model backbone. In contrast, our method draws inspiration from regularization techniques such as CHILD-TUNING$_D$ and DPS dense, specifically designed to reduce overfitting and improve

[2] Consistent with prior methods (Xu et al., 2021; Zhang et al., 2022), a random subset is chosen from entire training dataset based on a seed, to avoid bias towards any specific subset. Evaluation is performed over ten random seeds.

Models	Methods	CoLA		MRPC		RTE		STSB		Average	
		Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean	Std
BERT	Vanilla	64.11	1.33	90.80	1.77	70.69	2.83	89.92	0.61	78.88	1.64
	Ours	**66.07**	1.35	**91.84**	0.37	**73.43**	1.52	**90.34**	0.48	**80.42**(+1.54)	0.93(-0.71)
BART	Vanilla	58.54	1.41	92.03	0.73	81.84	1.41	91.54	0.40	80.99	0.99
	Ours	**60.15**	0.81	**92.33**	0.40	**84.26**	0.54	**92.20**	0.09	**82.23**(+1.24)	0.46(-0.53)
RoBERTa	Vanilla	66.06	2.07	92.25	0.57	78.52	13.01	91.89	0.31	82.18	3.99
	Ours	**66.52**	1.45	**92.58**	0.48	**84.22**	1.44	**92.21**	0.08	**83.88**(+1.70)	0.86(-3.13)
DeBERTa	Vanilla	63.74	1.34	92.31	0.37	85.59	1.58	91.74	0.17	83.34	0.86
	Ours	**65.96**	1.15	**92.32**	0.28	**86.17**	1.47	**91.99**	0.15	**84.11**(+0.77)	0.76(-0.10)
XLNet	Vanilla	40.93	27.28	91.83	0.91	71.17	14.40	91.68	0.19	73.90	10.69
	Ours	**61.66**	1.95	**92.19**	0.38	**83.54**	1.44	**92.12**	0.08	**82.38**(+8.48)	0.96(-9.73)

Table 2: Comparison of our method and vanilla finetuning on five popular PLMs. We evaluated the models using ten runs with different random seeds and reported the results in terms of mean and standard deviation. Average score represents the average performance across four datasets, and the best scores are highlighted in **bold**. The underlined values indicate occurrences of degenerate seeds.

Methods	CoLA		MRPC		RTE		STSB		Average	
	Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean	Std
Vanilla	64.11	1.33	90.80	1.77	70.69	2.83	89.92	0.61	78.88	1.64
Mixout	64.42	1.51	91.31	1.08	72.05	1.67	90.39	0.57	79.54	1.21
R3F	64.62	1.38	91.63	0.93	70.75	1.76	89.92	0.61	79.23	1.17
R-Dropout	64.14	1.58	**91.87**	0.78	70.24	2.83	90.25	0.49	79.13	1.42
CHILD-TUNING$_D$	64.85	1.32	91.52	0.81	71.69	1.95	90.42	0.44	79.62	1.13
Re-init	64.24	2.03	91.61	0.80	72.44	1.74	**90.71**	0.14	79.75	1.18
DPS Dense	64.98	1.08	91.50	0.83	73.14	1.97	90.51	0.55	80.03	1.11
DPS Dense (Our run)	64.08	1.50	90.25	2.21	71.92	1.45	90.20	0.47	79.11	1.41
Ours	**66.07**	1.35	91.84	0.37	**73.43**	1.52	90.34	0.48	**80.42**	0.93

Table 3: Comparison of our method with other finetuning methods on four small datasets (CoLA, RTE, MRPC, STSB), known for causing instability in BERT$_{\text{LARGE}}$ (Lee et al., 2019). The baseline results are taken from the DPS dense paper (Zhang et al., 2022). The mean and standard deviation (std) of ten random seeds are reported for each method. We utilize the vanilla results on STS-B for R3F since it cannot be applied to the regression task of STS-B. **Bold** indicates the best performance. Double-sided t-tests were performed between our method and the vanilla method. The p-values are less than 0.05, indicating statistically significant performance improvement over vanilla.

performance in low-resource finetuning scenarios. Our proposed method further improves the performance over these prior methods by addressing the challenges associated with FIM-based discrete subnetwork selection by employing continuous optimization through attention-guided weight mixup.

4.5 Performance on different PLMs

In this experiment, we investigate the effectiveness of our method when applied to various PLMs across different architectures compared to vanilla finetuning. We perform experiments using four GLUE tasks, chosen particularly due to their small training dataset sizes, on the language models mentioned in Section 4.2. The results of these experiments are presented in Table 2.

Apart from BERT, our method consistently achieves better results on RoBERTa, which is trained on a larger pretraining corpus and employs more advanced self-supervised pretraining tasks. Notably, on the RTE dataset, the vanilla method yielded degenerate results for two random seeds, leading to poor performance and high standard deviation. This issue has been attributed to the vanishing gradients problem occurring for certain initialization (Mosbach et al., 2020). However, using the same initialization and settings, our method showed better results, and the degenerate results were not observed. Similar results were observed with XLNet. We observe a notable gain of 8.48% over vanilla, along with a substantial decrease in the standard deviation by 9.73. Furthermore, we observe performance improvements on DeBERTa, a model with relative position encoding, and BART, an encoder-decoder-based model. These findings suggest that our method can effectively utilize the potential of pretrained weights, irrespective of the underlying model architecture, leading to enhanced finetuning performance across a variety of PLMs.

4.6 A comparison of previous techniques on few-sample BERT finetuning

We compare our method to prior regularization-based approaches, namely Mixout (Lee et al., 2019), R3F (Aghajanyan et al., 2020), R-Dropout (Wu et al., 2021), Re-init (Zhang et al., 2020), CHILD-TUNING$_D$ (Xu et al., 2021), and DPS dense (Zhang et al., 2022), following Zhang et al. (2022); Xu et al. (2021). We specifically employ CoLA, RTE, MRPC, and STSB datasets for this evaluation, as finetuning BERT$_{\text{LARGE}}$ model on these small datasets cause instability (Lee et al., 2019). Table 3 summarizes our results.

Our method surpasses all other baselines in terms of average scores, with a particularly notable improvement on the CoLA dataset over baselines, illustrating the effectiveness of our approach. Additionally, our method has the smallest standard deviation averaged over the datasets, indicating increased stability. In summary, our attention-guided weight mixup approach improves stability and performance on small datasets over baselines.

4.7 Ablation studies

Joint-training We conduct an ablation study contrasting our method with "Joint Training", a technique where the W and α parameters are jointly optimized by minimizing the loss on the whole training set rather than using our proposed BLO framework. The results are summarized in Table 4. The results underscore the superior performance of our method across all datasets, yielding a higher average score and a reduced standard deviation. Notably, on the MRPC dataset (Table 11), Joint-Training performs worse than the vanilla baseline. This outcome suggests a potential pitfall of Joint-Training: the simultaneous learning of both W and α parameters on the training dataset can lead to overfitting and result in a model that lacks generalizability on unseen test data. Conversely, our proposed approach, which uses a BLO framework to learn W and α on two different splits of the training dataset, effectively mitigates overfitting, leading to better generalization on the test set, and enhancing overall performance.

Randomly fixed α We investigate the impact of randomly initializing the α parameters in the network and keeping them fixed throughout the optimization. We sample α_1 and α_2 parameters from a Gaussian distribution, $\{\alpha_1\}_{ij}, \{\alpha_2\}_{ij} \in \mathcal{N}(\mu_\alpha, \sigma_\alpha)$, and fix them during finetuning, denoted as Random$_\alpha$. The Gaussian distribution has a mean of $\mu_\alpha = 1$ and standard deviations $\sigma_\alpha = \{0.005, 0.1, 0.45\}$. After sampling from this distribution, we clip the α values to lie within the range of 0 to 1. The results in Table 4 show that our method with learnable α outperforms the model with randomly initialized α, which underscores the importance of learning the attention parameters for child network selection.

Method	Mean	Std
Ours	**80.42**	0.93
Vanilla	78.88	1.64
Joint Training	78.86	1.48
Random$_\alpha$		
$\sigma_\alpha = 0.005$	79.36	1.03
$\sigma_\alpha = 0.1$	78.32	2.27
$\sigma_\alpha = 0.45$	69.29	5.44

Table 4: Averaged performance across CoLA, RTE, STSB, and MRPC datasets for Vanilla, Joint-training and Random$_\alpha$ by varying σ_α. Results on each dataset is presented in Table 11.

5 Computational costs and trade-offs

Our method introduces attention-guided weights mixup through a BLO framework, leading to additional computational demands summarized in Table 5, a characteristic shared with FIM-based strategies such as CHILD-TUNING$_D$ and DPS. Despite requiring comparable computation as current state-of-the-art techniques, our method offers performance improvements over several baselines, justifying the computational overhead. In terms of training efficiency, our method, in the worst-case scenario, requires a maximum of four times more training time than the vanilla method. However, in the best-case scenario, it is just 1.8 times the vanilla method. It is determined by the hyperparameter 'K'. This increase is relatively small given the inherently smaller size of the low-resource datasets we focus on. On the other hand, it is pertinent to note that the inference time complexity remains consistent with other established methods.

We compare the training costs of PEFT methods with our approach. Prompt Tuning necessitates approximately 0.2 times the training cost of vanilla fine-tuning, utilizing only 0.0067% of the total parameters as trainable. In contrast, Prefix-Tuning and LoRA require about 0.4× and 0.3× the training cost of vanilla fine-tuning, respectively, with Prefix-Tuning utilizing 0.294% and LoRA 0.236%

Method	Vanilla	R3F	R-Dropout	CHILD-TUNING$_D$	DPS	Ours
Time usage	$\times 1$	$\times 1.64$	$\times 1.64$	$\times 3.13$	$\times 1.12$	$\times(1+0.8K)$

Table 5: Comparison of time usage for different regularization based methods, where K is a hyperparameter chosen using grid-search in $K = \{1, 2, 5\}$. Although we choose $K = 5$ in all of our experiments, $K = 2$ yields slightly worse but comparable performance and saves computational time.

of the total parameters as trainable. For additional information, please refer to Appendix F. Despite our method's higher training cost compared to the PEFT methods, as detailed in Section 4.4, it consistently surpasses vanilla fine-tuning in performance, whereas the PEFT methods generally fall short in most scenarios.

6 Conclusions and future works

In this work, we propose an attention-guided weight mixup mechanism to address issues in fine-tuning PLMs on low-resource datasets. Specifically, we represent each weight as a linear interpolation of the task weights and the pretrained weights, controlled by an attention parameter. Capitalizing on the inherent structure of this representation, we utilize a BLO framework to learn task weights and attention parameters on two different splits of training dataset. Our method demonstrated its effectiveness across several challenging datasets from the GLUE benchmark, outperforming baselines in low-resource scenarios. It also shows improved stability across different PLM architectures. Moreover, through ablation studies, we highlighted the importance of learning attention parameters continuously and the benefits of our BLO framework over the straightforward joint-training of model weights and attention parameters.

We believe our method has the potential to be applied to other important areas, such as lifelong learning (Parisi et al., 2019). A common challenge in lifelong learning is retaining knowledge from previous tasks while adapting to new ones. Exploring our attention-guided weights mixup, optimized using BLO, for this application presents a promising avenue for future work.

7 Limitations

Our method improves the finetuning of PLMs but adds computational overhead, similar to CHILD-TUNING$_D$ and DPS dense, mainly due to calculating attention parameters in the BLO framework, akin to the computational demands of estimating and updating FIM in prior methods. However, its substantial performance gains over various baselines justify this extra cost. In pursuit of minimizing this computational overhead, we contemplate an approach where the α parameters are intermittently updated, specifically every T iterations. We believe this strategy will strike a balance between efficiency and performance. Delving into this modification poses an exciting avenue for subsequent research. It would also be insightful to extend our method to multi-language tasks to understand its adaptability and broader applicability in varying linguistic contexts.

References

Armen Aghajanyan, Akshat Shrivastava, Anchit Gupta, Naman Goyal, Luke Zettlemoyer, and Sonal Gupta. 2020. Better fine-tuning by reducing representational collapse. *arXiv preprint arXiv:2008.03156*.

Atilim Gunes Baydin, Robert Cornish, David Martinez Rubio, Mark Schmidt, and Frank Wood. 2017. Online learning rate adaptation with hypergradient descent. *arXiv preprint arXiv:1703.04782*.

Yonatan Belinkov, James Henderson, et al. 2020. Variational information bottleneck for effective low-resource fine-tuning. In *International Conference on Learning Representations*.

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

Daniel Cer, Mona Diab, Eneko Agirre, Inigo Lopez-Gazpio, and Lucia Specia. 2017. Semeval-2017 task 1: Semantic textual similarity multilingual and crosslingual focused evaluation. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 1–14.

Sanyuan Chen, Yutai Hou, Yiming Cui, Wanxiang Che, Ting Liu, and Xiangzhan Yu. 2020. Recall and learn: Fine-tuning deep pretrained language models with less forgetting. *arXiv preprint arXiv:2004.12651*.

Sang Keun Choe, Willie Neiswanger, Pengtao Xie, and Eric Xing. 2022. Betty: An automatic differentiation library for multilevel optimization. *arXiv preprint arXiv:2207.02849*.

Ido Dagan, Oren Glickman, and Bernardo Magnini. 2006. The pascal recognising textual entailment challenge. In *Machine Learning Challenges. Evaluating Predictive Uncertainty, Visual Object Classification, and Recognising Tectual Entailment: First PASCAL Machine Learning Challenges Workshop, MLCW 2005, Southampton, UK, April 11-13, 2005, Revised Selected Papers*, pages 177–190. Springer.

Hal Daumé III. 2009. Frustratingly easy domain adaptation. *arXiv preprint arXiv:0907.1815*.

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*.

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

William B Dolan and Chris Brockett. 2005. Automatically constructing a corpus of sentential paraphrases. In *Third International Workshop on Paraphrasing (IWP2005)*.

Matthias Feurer, Jost Springenberg, and Frank Hutter. 2015. Initializing bayesian hyperparameter optimization via meta-learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 29.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. 2017. Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*, pages 1126–1135. PMLR.

Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. 2020. Deberta: Decoding-enhanced bert with disentangled attention. *arXiv preprint arXiv:2006.03654*.

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.

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

Gabriel Ilharco, Marco Tulio Ribeiro, Mitchell Wortsman, Suchin Gururangan, Ludwig Schmidt, Hannaneh Hajishirzi, and Ali Farhadi. 2022a. Editing models with task arithmetic. *arXiv preprint arXiv:2212.04089*.

Gabriel Ilharco, Mitchell Wortsman, Samir Yitzhak Gadre, Shuran Song, Hannaneh Hajishirzi, Simon Kornblith, Ali Farhadi, and Ludwig Schmidt. 2022b. Patching open-vocabulary models by interpolating weights. *arXiv preprint arXiv:2208.05592*.

Kun Kuang, Hengtao Zhang, Runze Wu, Fei Wu, Yueting Zhuang, and Aijun Zhang. 2021. Balance-subsampled stable prediction across unknown test data. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 16(3):1–21.

Frederik Kunstner, Philipp Hennig, and Lukas Balles. 2019. Limitations of the empirical fisher approximation for natural gradient descent. *Advances in neural information processing systems*, 32.

Cheolhyoung Lee, Kyunghyun Cho, and Wanmo Kang. 2019. Mixout: Effective regularization to finetune large-scale pretrained language models. *arXiv preprint arXiv:1909.11299*.

Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The power of scale for parameter-efficient prompt tuning. *arXiv preprint arXiv:2104.08691*.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. 2019. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461*.

Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation. *arXiv preprint arXiv:2101.00190*.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. 2018. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*.

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*.

Jonathan Lorraine, Paul Vicol, and David Duvenaud. 2020. Optimizing millions of hyperparameters by implicit differentiation. In *International conference on artificial intelligence and statistics*, pages 1540–1552. PMLR.

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

Sourab Mangrulkar, Sylvain Gugger, Lysandre Debut, Younes Belkada, Sayak Paul, and Benjamin Bossan. 2022. Peft: State-of-the-art parameter-efficient fine-tuning methods. `https://github.com/huggingface/peft`.

Marius Mosbach, Maksym Andriushchenko, and Dietrich Klakow. 2020. On the stability of fine-tuning bert: Misconceptions, explanations, and strong baselines. *arXiv preprint arXiv:2006.04884*.

German I Parisi, Ronald Kemker, Jose L Part, Christopher Kanan, and Stefan Wermter. 2019. Continual lifelong learning with neural networks: A review. *Neural networks*, 113:54–71.

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

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, Peter J Liu, et al. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21(140):1–67.

Aravind Rajeswaran, Chelsea Finn, Sham M Kakade, and Sergey Levine. 2019. Meta-learning with implicit gradients. *Advances in neural information processing systems*, 32.

Zhongzheng Ren, Raymond Yeh, and Alexander Schwing. 2020. Not all unlabeled data are equal: Learning to weight data in semi-supervised learning. *Advances in Neural Information Processing Systems*, 33:21786–21797.

Jun Shu, Qi Xie, Lixuan Yi, Qian Zhao, Sanping Zhou, Zongben Xu, and Deyu Meng. 2019. Meta-weight-net: Learning an explicit mapping for sample weighting. *Advances in neural information processing systems*, 32.

Ankur Sinha, Pekka Malo, and Kalyanmoy Deb. 2017. A review on bilevel optimization: From classical to evolutionary approaches and applications. *IEEE Transactions on Evolutionary Computation*, 22(2):276–295.

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

Alexander Soen and Ke Sun. 2021. On the variance of the fisher information for deep learning. *Advances in Neural Information Processing Systems*, 34:5708–5719.

Shoujie Tong, Heming Xia, Damai Dai, Tianyu Liu, Binghuai Lin, Yunbo Cao, and Zhifang Sui. 2023. Bi-drop: Generalizable fine-tuning for pre-trained language models via adaptive subnetwork optimization. *arXiv preprint arXiv:2305.14760*.

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.

Yulin Wang, Jiayi Guo, Shiji Song, and Gao Huang. 2020. Meta-semi: A meta-learning approach for semi-supervised learning. *arXiv preprint arXiv:2007.02394*.

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

Adina Williams, Nikita Nangia, and Samuel Bowman. 2018. A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1112–1122.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

Mitchell Wortsman, Gabriel Ilharco, Samir Ya Gadre, Rebecca Roelofs, Raphael Gontijo-Lopes, Ari S Morcos, Hongseok Namkoong, Ali Farhadi, Yair Carmon, Simon Kornblith, et al. 2022. Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time. In *International Conference on Machine Learning*, pages 23965–23998. PMLR.

Lijun Wu, Juntao Li, Yue Wang, Qi Meng, Tao Qin, Wei Chen, Min Zhang, Tie-Yan Liu, et al. 2021. R-drop: Regularized dropout for neural networks. *Advances in Neural Information Processing Systems*, 34:10890–10905.

Runxin Xu, Fuli Luo, Zhiyuan Zhang, Chuanqi Tan, Baobao Chang, Songfang Huang, and Fei Huang. 2021. Raise a child in large language model: Towards effective and generalizable fine-tuning. *arXiv preprint arXiv:2109.05687*.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. *Advances in neural information processing systems*, 32.

Haojie Zhang, Ge Li, Jia Li, Zhongjin Zhang, Yuqi Zhu, and Zhi Jin. 2022. Fine-tuning pre-trained language models effectively by optimizing subnetworks adaptively. *arXiv preprint arXiv:2211.01642*.

Tianyi Zhang, Felix Wu, Arzoo Katiyar, Kilian Q Weinberger, and Yoav Artzi. 2020. Revisiting few-sample bert fine-tuning. *arXiv preprint arXiv:2006.05987*.

Yftah Ziser and Roi Reichart. 2019. Task refinement learning for improved accuracy and stability of unsupervised domain adaptation. In *proceedings of the 57th annual meeting of the Association for Computational Linguistics*, pages 5895–5906.

A Optimization algorithm

To the best of our knowledge, our work is the first to employ BLO for selecting child network to prevent overfitting during the finetuning of large pretrained models. In this section, we dwell into the algorithm used to solve the proposed mathematical framework. Various algorithms have been proposed to (approximately) solve a BLO problem. These can be broadly classified into two categories based on their approach to calculating upper-level gradients: implicit differentiation methods (such as Finite Difference (Liu et al., 2018), Neumann Series (Lorraine et al., 2020), and Conjugate Gradient (Rajeswaran et al., 2019)) and iterative differentiation methods (including Reverse-mode Automatic Differentiation (Finn et al., 2017)). Choe et al. (2022) empirically show that the Finite Difference method (Liu et al., 2018) has the best performance among these methods. Thus, we adopt this algorithm (Liu et al., 2018) to solve our BLO (4) in the search phase. One-step gradient descent is used to approximate W.

$$W^*(\alpha) \approx W' = W - \eta_w \nabla_W [\mathcal{L}(g(W, \alpha, W_0); \mathcal{D}^{\text{B-tr}}) + \lambda_1 \|W\|_F^2] \tag{5}$$

For ease of notation, we denote

$$\mathcal{G}(W, \alpha; \mathcal{D}^{\text{B-tr}}, W_0) = \mathcal{L}(g(W, \alpha, W_0); \mathcal{D}^{\text{B-tr}}) + \lambda_1 \|W\|_F^2$$

W' is plugged into the objective function of stage II. The gradient of the objective function in stage II with respect to α is computed to update α:

$$\alpha^* \approx \alpha' = \alpha - \eta_\alpha \nabla_\alpha [\mathcal{L}(g(W', \alpha, W_0); \mathcal{D}^{\text{B-val}}) + \lambda_2 \|\alpha\|_F^2] \tag{6}$$

Equation 6 can be further reduced as follows. The chain rule is applied to estimate the gradient of the loss function in stage II with respect to α. W' is an implicit on α as discussed above. The first part of the gradients can be decomposed as follows,

Algorithm 1 Optimization algorithm

Training dataset - $\mathcal{D}^{\text{tr}}$, BLO training dataset - $\mathcal{D}^{\text{B-tr}}$, BLO validation dataset - $\mathcal{D}^{\text{B-val}}$.

# Search phase
while not converged **do**
Update task weights W using Equation (5) on $\mathcal{D}^{\text{B-tr}}$
Update attention parameter α using Equation (6) on $\mathcal{D}^{\text{B-val}}$
end while

Finetune phase
Using the learned α', finetune the task weights further on $\mathcal{D}^{\text{tr}}$ until convergence.

$$\begin{aligned}
&\nabla_\alpha \mathcal{L}(g(W', \alpha, W_0); \mathcal{D}^{\text{B-val}}) \\
&= \nabla_\alpha \mathcal{L}(g(W - \eta_w \nabla_W \mathcal{G}(W, \alpha; \mathcal{D}^{\text{B-tr}}, W_0), \alpha, W_0); \mathcal{D}^{\text{B-val}}) \\
&= \nabla_\alpha \mathcal{L}(g(W', \alpha, W_0); \mathcal{D}^{\text{B-val}}) - \eta_w \times \\
&\nabla^2_{\alpha,W} \mathcal{G}(W, \alpha; \mathcal{D}^{\text{B-tr}}, W_0) \nabla_{W'} \mathcal{L}(g(W', \alpha, W_0); \mathcal{D}^{\text{B-val}})
\end{aligned}$$

The above gradient estimation contains an expensive matrix-vector product that can be reduced using finite difference approximation:

$$\nabla^2_{\alpha,W} \mathcal{G}(W, \alpha; \mathcal{D}^{\text{B-tr}}, W_0) \nabla_{W'} \mathcal{L}(W', \alpha; \mathcal{D}^{\text{B-val}}, W_0) = \frac{\nabla_\alpha \mathcal{G}(W^+, \alpha; \mathcal{D}^{\text{B-tr}}, W_0) - \nabla_\alpha \mathcal{G}(W^-, \alpha; \mathcal{D}^{\text{B-tr}}, W_0)}{2\epsilon},$$

where

$$W^\pm = W \pm \epsilon \nabla_{W'} \mathcal{L}(W', \alpha; \mathcal{D}^{\text{B-val}}, W_0),$$
$$\epsilon = \frac{0.01}{\|\nabla_{W'} \mathcal{L}(W', \alpha; \mathcal{D}^{\text{B-val}}, W_0)\|_2}.$$

The algorithm is run iteratively until convergence to estimate $\alpha^* \approx \alpha'$. Finetuning phase is further conducted on $\mathcal{D}^{\text{tr}}$ with the learned attention parameter α'.

We analyze the loss curves of both the inner and outer optimization processes in our algorithm to determine the optimal hyperparameter choices. Given that our task involves solving a bilevel optimization problem, and considering the significance of the model's performance on $\mathcal{D}^{\text{B-val}}$ (which pertains to the outer problem), we regard the convergence of the outer loss as a crucial criterion for overall convergence.

B Datasets

We evaluated our method on various datasets from the GLUE benchmark (Warstadt et al., 2018; Wang et al., 2018). In this section, we describe each of

Dataset	RTE	MRPC	STS-B	CoLA	SST-2	QNLI	QQP	MNLI
Train Examples	2.5k	3.7k	5.7k	8.5k	67k	105k	364k	393k
Dev Examples	277	408	1.5k	1.0k	872	5.5k	40k	4.8k
Metrics	Acc	F1	SCC	MCC	Acc	Acc	Acc	Acc

Table 6: The table details the eight datasets utilized in this study, sourced from the GLUE benchmark. Here, *Acc* denotes Accuracy, *SCC* refers to the Spearman Correlation Coefficient, and *MCC* signifies the Matthews Correlation Coefficient.

Models	Datasets	Batch Size	Learning Rate	Epochs/Steps	Warmup Ratio/Steps	Weight Decay
BERT	all	16	2e-5	3 epochs	10%	0.01
RoBERTa	RTE	16	2e-5	2036 steps	122 steps	0.1
	MPRC	16	1e-5	2296 steps	137 steps	0.1
	STS-B	16	2e-5	3598 steps	214 steps	0.1
	CoLA	16	1e-5	5336 steps	320 steps	0.1
DeBERTa	RTE	32	1e-5	6 epochs	50 steps	0.01
	MPRC	32	1e-5	6 epochs	50 steps	0.01
	STS-B	32	7e-6	4 epochs	100 steps	0.01
	CoLA	32	7e-6	6 epochs	100 steps	0.01
BART	RTE	32	1e-5	1018 steps	61 steps	0.01
	MPRC	64	2e-5	1148 steps	68 steps	0.01
	STS-B	32	2e-5	1799 steps	107 steps	0.01
	CoLA	64	2e-5	1334 steps	80 steps	0.01
XLNet	RTE	32	2e-5	3000 steps	500 steps	0.01
	MPRC	32	2e-5	800 steps	200 steps	0.01
	STS-B	32	2e-5	800 steps	200 steps	0.01
	CoLA	32	2e-5	3000 steps	500 steps	0.01

Table 7: Hyperparameter settings, as reported in their official repository, for all the different PLMs used in this work. For the learning rates, as an example, 2e-5 means 2×10^{-5}.

the datasets used in this work. Table 6 summarizes the split statistics and the evaluation metric for each dataset. The CoLA dataset (Warstadt et al., 2018), known as the Corpus of Linguistic Acceptability, serves as a benchmark to assess the natural language processing model's capacity to comprehend and predict the acceptability of English sentences. On the other hand, the RTE dataset (Dagan et al., 2006) focuses on determining the logical entailment between sentences. QNLI dataset (Devlin et al., 2018) originates from the Stanford Question Answering Dataset (SQuAD) and aims to evaluate the model's competence in comprehending and inferring information from questions and corresponding contextual paragraphs. MNLI dataset's objective involves classifying sentence relationships as entailment, contradiction, or neutral, and it encompasses diverse genres and domains, providing a more comprehensive and challenging assessment of models' inference abilities in varied contexts (Williams et al., 2018). Lastly, MRPC (Dolan and Brockett, 2005), STS-B (Cer et al., 2017), and QQP datasets (Wang et al., 2018) are employed to tackle paraphrase identification, semantic textual similarity, and question pair similarity tasks, respectively. The Stanford Sentiment Treebank (SST-2) dataset (Socher et al., 2013) is a sentiment classification task on sentences from movie reviews.

C More about related works

Prior finetuning regularization-based approaches We compare our method with several baselines on $\text{BERT}_{\text{LARGE}}$ model (Devlin et al., 2018). It's worthwhile to note that this is a challenging task. Various methods are proposed to tackle this challenge. Mixout (Lee et al., 2019) stochastically replaces model parameters with pretrained parameters (with probability p) to mitigate catastrophic forgetting issue. R3F (Aghajanyan et al., 2020) introduces parametric noise into input sentence embeddings for robustness. R-Dropout (Wu et al., 2021) mitigates the two-directional KL divergence by assessing the output distributions of two submodels, each independently generated via Dropout. In Re-init (Zhang et al., 2020), the pooler and top K BERT transformer layers are reinitialized from the distribution $\mathcal{N}(0, 0.02^2)$, which is the original BERT initialization. CHILD-TUNING$_D$ (Xu et al., 2021) proposes to finetune a child network selected using FIM, constructed from the gradients of the training dataset, with the non-child network

Datasets	Vanilla	DPS Dense	Child Tuning	Ours
CoLA	24.07 ± 17.04	23.85 ± 16.12	23.47 ± 14.48	**41.81** ± 5.57
RTE	58.09 ± 2.64	55.19 ± 4.54	58.77 ± 3.07	**60.54** ± 2.59
STSB	78.08 ± 3.21	74.07 ± 5.94	79.62 ± 2.14	**82.09** ± 4.88
SST-2	80.52 ± 12.87	82.79 ± 4.06	77.67 ± 7.63	**89.52** ± 0.78
MRPC	80.57 ± 1.23	**81.66** ± 0.53	81.09 ± 1.25	80.63 ± 1.47
QQP	67.73 ± 3.95	67.14 ± 3.59	68.45 ± 4.46	**72.54** ± 1.67
QNLI	71.03 ± 8.92	68.79 ± 7.79	70.84 ± 8.42	**76.16** ± 2.77
MNLI	40.19 ± 2.67	40.04 ± 2.42	39.85 ± 2.55	**48.44** ± 5.01
AVG	62.54 ± 6.57	61.69 ± 5.62	62.47 ± 5.5	**68.97** ± 3.09

(a) Results for 300 training samples.

Datasets	Vanilla	DPS Dense	Child Tuning	Ours
CoLA	26.01 ± 13.2	39.42 ± 11.8	38.15 ± 14.10	**49.35** ± 1.71
RTE	58.30 ± 4.90	58.34 ± 2.90	59.78 ± 5.02	**62.96** ± 4.40
STSB	81.77 ± 2.69	83.38 ± 1.55	80.87 ± 3.97	**86.85** ± 0.52
SST-2	86.51 ± 6.48	**89.38** ± 0.52	89.04 ± 1.34	88.98 ± 0.91
MRPC	82.96 ± 0.84	**83.29** ± 0.78	81.42 ± 1.88	83.21 ± 2.30
QQP	71.68 ± 1.90	74.43 ± 1.35	74.53 ± 1.45	**74.80** ± 0.58
QNLI	76.82 ± 2.09	77.09 ± 1.75	76.73 ± 3.19	**79.78** ± 1.83
MNLI	42.81 ± 4.49	46.61 ± 2.70	46.33 ± 3.96	**53.41** ± 4.84
AVG	65.85 ± 4.57	68.99 ± 2.92	68.35 ± 4.36	**72.42** ± 2.14

(b) Results for 500 training samples.

Datasets	Vanilla	DPS Dense	Child Tuning	Ours
CoLA	47.97 ± 5.62	52.89 ± 2.50	51.69 ± 2.81	**54.19** ± 1.77
RTE	62.60 ± 3.46	64.44 ± 1.76	63.93 ± 5.15	**67.62** ± 2.53
STSB	85.86 ± 1.34	87.15 ± 1.77	87.08 ± 0.79	**88.31** ± 0.78
SST-2	90.28 ± 0.55	**90.92** ± 0.64	89.92 ± 2.95	90.46 ± 0.66
MRPC	85.34 ± 1.30	85.90 ± 0.86	84.81 ± 1.81	**87.03** ± 1.76
QQP	76.96 ± 1.50	77.81 ± 0.57	76.22 ± 2.60	**77.81** ± 0.67
QNLI	81.61 ± 1.32	82.58 ± 1.19	82.42 ± 1.18	**82.94** ± 1.02
MNLI	54.93 ± 5.94	58.33 ± 3.56	56.56 ± 4.99	**65.07** ± 3.48
AVG	73.19 ± 2.62	75.00 ± 1.61	74.07 ± 2.75	**76.68** ± 1.58

(c) Results for 1000 training samples.

Table 8: We present a comparison of our method, Vanilla, DPS dense, and CHILD-TUNING$_D$ method on BERT$_{\text{LARGE}}$ (Lee et al., 2019) across various low-resource scenarios. We report the mean and standard deviation of ten random seeds. **Bold** indicates the best performance. On average, our method outperforms the best baseline by 6.43 points in the setting with 300 training samples, by 3.43 points in the setting with 500 training samples, and outperforms it by 1.68 points for 1000 training samples.

frozen to pretrained weights. DPS dense (Zhang et al., 2022) provides an alternative technique for selecting child networks, utilizing a two-stage update algorithm. The initial stage involves the computation of a Gradient Accumulation Matrix (GAM), which estimates empirical FIM, for all parameters within the model. Following that, in the second stage, a child network is selected according to the GAM, after which its corresponding weights undergo modification.

Very recent work by Tong et al. (2023) has introduced a novel approach to select optimal sub-networks. Drawing inspiration from Wu et al. (2021), their method selects sub-networks leveraging gradient from sub-models generated by dropout, sidestepping the Fisher information matrix. Nevertheless, they still retain a thresholding approach for sub-network selection based on mini-batch gradients, akin to FIM-based approaches, which is sub-optimal. In contrast, our methodology diverges from a discrete sub-network selection, opting for a bi-level continuous optimization strategy tailored to improve performance on a downstream task. Since their code is not available in their linked GitHub repository, by comparing our results against those of Tong et al. (2023) directly, the benefits of our methodology become evident. Evaluating on various settings using the BERT$_{\text{LARGE}}$ model, we observed the following: On low-resource settings, for 500 samples, our method scores an average of 72.42, outperforming their 71.06 (as seen in Table 1); With 1000 samples, we achieve an average

score of 76.68, superior to their 76.21 (Table 1). On few-sample BERT finetuning, our average score stands at 80.42 compared to their 80.26 (Table 3).

Weight space manipulations Task vectors in weight space (Ilharco et al., 2022a), which represent the difference between task and pretrained weights in a model, can be manipulated through arithmetic operations to guide a model towards specific tasks. Ilharco et al. (2022b) introduces PAINT, a model patching method for open-vocabulary models like CLIP, which employs a convex combination of task and pretrained weights to adapt to new tasks to be patched while preserving accuracy on previous tasks, the mixing coefficient is manually assigned. In contrast to these works, our approach presents a regularization technique where resultant weights are represented as a weighted summation of task and pretrained weights controlled by the attention parameter, specifically aiming to mitigate issues in finetuning pretrained language models on small downstream datasets. We learn the attention parameters and task weights using BLO framework.

D Hyperparameters settings

We evaluate our method on different PLM's including $\text{BERT}_{\text{LARGE}}$ (Devlin et al., 2018), $\text{RoBERTa}_{\text{LARGE}}$ (Liu et al., 2019), $\text{BART}_{\text{LARGE}}$ (Lewis et al., 2019), $\text{DeBERTa}_{\text{LARGE}}$ (He et al., 2020), and $\text{XLNet}_{\text{LARGE}}$ (Yang et al., 2019).

For the experimental results presented in Sections 4.6, 4.3, G, and 4.7, we employed the $\text{BERT}_{\text{LARGE}}$ model, following Xu et al. (2021); Zhang et al. (2022). Our algorithm consists of two phases, 1) Search phase and 2) Finetune phase. We report the hyperparameters used for each phase and grid search for the best hyperparameter settings for a task on a split of the training set following Xu et al. (2021); Zhang et al. (2022) for the $\text{BERT}_{\text{LARGE}}$ model.

Consistent with prior works in low-resource settings, such as those in our baseline methods (Xu et al., 2021; Zhang et al., 2022), we randomly select a subset of the specified size (300, 500, or 1000) from the entire dataset based on a seed. This approach is commonly adopted in this field to avoid bias towards any specific fixed subset. We then report the mean and standard deviation of the evaluation metric across multiple seeds for all methods.

We partition $\mathcal{D}^{\text{tr}}$ into two subsets of $\mathcal{D}^{\text{B-tr}}$ and $\mathcal{D}^{\text{B-val}}$. We conducted experiments with different data ratios for $\mathcal{D}^{\text{B-tr}}$ and $\mathcal{D}^{\text{B-val}}$, including 50%-50% and 80%-20%. Empirically, we observed that the 80%-20% split performs better in low-resource settings compared to a 50%-50% split. This is likely because allocating more training data points to the inner optimization of task weights is advantageous, especially given the greater number of parameters (task weights) at the inner optimization level.

D.1 Search phase

The weights mixup is applied to all the parameters with "nn.Linear layers". The learning rate of W is set to 2×10^{-5}. The weight decay is set to 0.01. The warmup ratio is 10%. The batch size is set to 16. AdamW (Loshchilov and Hutter, 2017) optimizer is used for W. We initialize the α values by sampling the entries of α_1 and α_2 from a Gaussian distribution with mean of 1 and standard deviation of 0.005. The learning rate of α is set to 2×10^{-3}. In this work, the warmup ratio for α is set equal to the warmup ratio of W which is 10%. We use an AdamW optimizer with $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$, and weight decay of 0.01. Since it is an optimization problem, we search for the hyperparameters that lead to a smooth reduction in the loss curves. We grid search for the number of epochs in {1,2,3,5}. The maximum sequence length is 128. We partition the original training dataset, represented as $\mathcal{D}^{\text{tr}}$, into two subsets of 80% and 20% split randomly to perform the search phase. To further mitigate overfitting, we perform this random sampling K times to produce K different sets of $\mathcal{D}^{\text{B-tr}}$ and $\mathcal{D}^{\text{B-val}}$.

We then perform the search phase K times to get K different learned parameters. We average them to obtain optimal learned parameters of the search phase. We search in $K = \{1,2,5\}$. Though $K = 5$ gave best results, $K = 2$ saves computation time and gives slightly less but comparative results. For instance, in the evaluation of the CoLA dataset presented in Table 3, setting K to 5 yielded an average score of 66.07, whereas setting K to 2 resulted in a score of 65.10. Both configurations outperformed the vanilla model, which achieved a score of 64.11.

We use a low-rank approximation of α for estimating the child net structure. Since we are mainly targeting the low-resource domain, the low-rank approximation of α mitigates overfitting. In our experimental setup, we consider the rank values of $\{1, 2, 8\}$ for the dimensionality reduction of α, se-

lecting the best performing value empirically. We found that a rank of r = 1 consistently produced superior results, thus, this rank is adopted across all experimental trials. We didn't try an even larger rank or full rank of α as a full rank of α is also computationally heavy to perform.

We initialize the α_1 and α_2 parameters using a Gaussian distribution. The mean of this distribution is set to 1, and we use a standard deviation of 0.005. We clip the initial alphas to [0, 1] to ensure they are valid attention parameters. This initialization strategy is designed to provide a balanced starting point for the optimization process.

In addition, we have conducted experiments on our method both with and without weight decay for α parameters. Based on our empirical findings, we observed that employing weight decay yields better results compared to not using it. The likely reason for this improvement is that weight decay encourages the α values to be closer to 0. This inclination towards lower values results in a higher contribution from the pretrained weights in the resultant weight estimation, which in turn aids in regularizing the network. Such regularization appears to be beneficial for the overall performance and stability of the model in our experiments.

D.2 Finetune phase

With the optimal α^* in search phase, we further finetune for W on entire training dataset. We use the default hyperparameter settings for W following Devlin et al. (2018). The number of epochs is searched in {1,3} and learning rate is searched in $\{2 \times 10^{-5}, 3 \times 10^{-6}\}$.

For different PLMs in Section 4.5, we use the default settings for W in their official repository for both the search and the finetune phase, without exhaustive hyperparameter sweep. The number of training steps/epochs, warm-up steps and batch size for the finetuning phase are given in Table 7. For learning α in search phase, we use α learning rate 2×10^{-3}, the α warmup ratio is set roughly the same as the warmup ratio of W for each of the different PLM, i.e., warmup ratio for RoBERTa is 0.06, DeBERTa is 0.06, XLNet is 0.17, and BART is 0.1. For α, we use an AdamW optimizer with β_1 = 0.9, β_2 = 0.999, $\epsilon = 10^{-8}$ and the weight decay is 0.01.

D.3 Baselines

For the baselines, the following hyperparameter search space is used following their respective original papers:

- Mixout: Mixout probability $p \in \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8\}$.
- R3F: Noise types $\in \{\mathcal{N}, \mathcal{U}\}$, $\sigma \in \{10^{-5}\}$, $\lambda \in \{0.1, 0.5, 1, 5\}$.
- Re-init: L $\in \{1, 2, 3, 4, 5, 6, 7\}$.
- CHILD-TUNING$_D$: $p_D \in \{0.1, 0.2, 0.3\}$, learning rate in $\{2 \times 10^{-5}, 4 \times 10^{-5}, 6 \times 10^{-5}, 8 \times 10^{-5}, 10^{-4}\}$.
- R-Dropout: $p \in \{0.1\}$ and $\alpha \in \{0.1, 0.5, 1, 3, 5\}$.
- DPS dense: $p \in \{0.1, 0.2, 0.3, 0.4, 0.5\}$ and update ratio $ur \in \{0.05, 0.1, 0.2\}$.

We use a single A100 GPU machine for our experiments. The mean and the standard deviation on the original development set over ten random seeds are reported following Xu et al. (2021); Zhang et al. (2022).

E Performance on low-resource scenarios

Due to space constraints, we only present the average scores across datasets in Table 1. The results on each of the eight datasets across 300, 500, 1K training data splits are presented in Table 8. We report both the mean and the standard deviation of the metric, evaluated on the test set for each dataset, across ten random seeds. On average, our method outperforms the best baseline by 6.43 points in the setting with 300 training samples, by 3.43 points in the setting with 500 training samples, and outperforms it by 1.68 points for 1000 training samples.

F Comparison with parameter efficient finetuning methods

Similarly, due to space constraints, we only present the average scores across datasets in Figure 2. The results on each datasets across 500, 1K training data splits are presented in Table 9. We report both the mean and the standard deviation of the metric, evaluated on the test set for each dataset, across ten random seeds.

We implemented the PEFT baselines using the PEFT library (Mangrulkar et al., 2022). In conducting our hyperparameter search, we adhered to the hyperparameter settings provided in their respective original papers and the examples from the PEFT library. For LoRA, we set the rank r to 8,

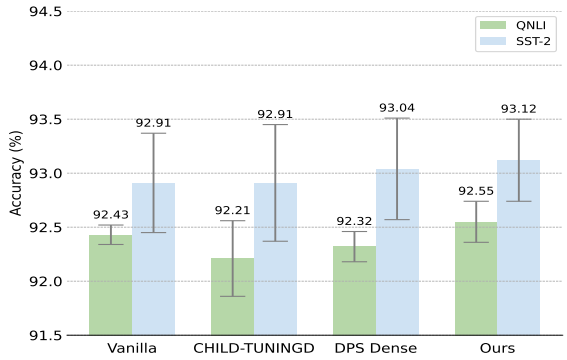

Figure 3: Comparison of our method with Vanilla, CHILD-TUNING$_D$, and DPS dense method on the QNLI and SST-2 datasets with sufficient training examples. The bar plots represent the mean accuracy from ten random seeds, and error bars denote the standard deviation.

the scaling factor α to 16, and included a dropout of 0.1. Consequently, LoRA has 788,482 trainable parameters (0.236% of total). In the case of Prefix-Tuning, we set the number of virtual tokens to 20. It has 985,090 trainable parameters (0.294% of total). For Prompt Tuning, we set the prompt length to 20. It has 22,530 trainable parameters (0.0067% of the total). In terms of time complexity, assuming vanilla is x1, Prompt Tuning is 0.2x, Prefix tuning is 0.4x and LoRA is 0.3x.

G Performance on sufficient samples

We investigate whether our method can also improve accuracy on larger training datasets. To this end, we train BERT$_{\text{LARGE}}$ on SST-2 and QNLI datasets, both of which have over 50K training examples. We compare our method to the baselines CHILD-TUNING$_D$ and DPS dense. The results are presented in Fig 3, respectively. On the QNLI dataset, both CHILD-TUNING$_D$ and DPS dense perform worse than the vanilla method. However, our method yields a small accuracy improvement over the vanilla method. The small gaps between the performance of these methods are reasonable, because as the number of training instances increases, the impact of regularization reduces, leading to non-significant improvements with regularization-based methods in large sample settings.

H Comparison with model soup

In this section, we compare our method against a baseline method inspired from the uniform model soup approach (Wortsman et al., 2022) in the low-resource settings. We average over five models, each trained using vanilla finetuning on the training dataset adhering to the optimal hyperparameter configuration as detailed in Devlin et al. (2018) and initialized with a different random seed. The performance of this averaged model is then evaluated on an unseen test dataset. This entire process is repeated ten times to obtain the mean and the standard deviation.

Our observations, summarized in Table 10, indicate that the model soup approach yields mixed results over vanilla finetuning in low-resource settings. Conversely, our method demonstrates consistent superiority over model soup. Our method leverages an attention-guided weight mixup, optimized through BLO across two distinct splits of the training dataset, resulting in more pronounced performance improvements.

Datasets	Vanilla	Prompt Tuning	Prefix-Tuning	LoRA	Ours
CoLA	26.01 ± 13.2	13.41 ± 9.95	42.96 ± 4.96	47.26 ± 2.57	**49.35** ± 1.71
RTE	58.30 ± 4.90	56.28 ± 2.16	55.77 ± 2.63	57.15 ± 1.87	**62.96** ± 4.40
STSB	81.77 ± 2.69	58.38 ± 16.52	74.44 ± 4.99	80.22 ± 2.93	**86.85** ± 0.52
MRPC	82.96 ± 0.84	81.85 ± 0.3	78.63 ± 2.31	79.88 ± 1.53	**83.21** ± 2.30
AVG	62.26 ± 5.41	52.48 ± 7.23	62.95 ± 3.72	66.13 ± 2.22	**70.59** ± 2.23

(a) Results for 500 training samples.

Datasets	Vanilla	Prompt Tuning	Prefix-Tuning	LoRA	Ours
CoLA	47.97 ± 5.62	23.69 ± 12.24	44.11 ± 14.79	51.46 ± 1.72	**54.19** ± 1.77
RTE	62.60 ± 3.46	57.44 ± 0.91	59.21 ± 4.80	60.50 ± 2.79	**67.62** ± 2.53
STSB	85.86 ± 1.34	69.29 ± 10.22	80.13 ± 1.58	83.57 ± 1.41	**88.31** ± 0.78
MRPC	85.34 ± 1.30	81.99 ± 0.43	83.70 ± 2.09	83.98 ± 2.09	**87.03** ± 1.76
AVG	70.44 ± 2.93	58.1 ± 5.95	66.79 ± 5.82	69.88 ± 2	**74.29** ± 1.71

(b) Results for 1000 training samples.

Table 9: We present a comparison of our method with Vanilla, Prompt Tuning, Prefix-Tuning, and LoRA finetuning approaches on CoLA, RTE, STSB, and MRPC datasets in low-resource scenarios with 500 and 1000 training instances. We report both mean and standard deviation, each over ten runs. The **bold** values represent the highest performance in each scenario.

	500			1K		
Datasets	**Vanilla**	**Model Soup**	**Ours**	**Vanilla**	**Model Soup**	**Ours**
CoLA	26.01 ± 13.2	26.33 ± 18.5	**49.35** ± 1.71	47.97 ± 5.62	43.69 ± 12.51	**54.19** ± 1.77
RTE	58.30 ± 4.90	58.56 ± 4.96	**62.96** ± 4.40	62.60 ± 3.46	63.93 ± 5.15	**67.62** ± 2.53
STSB	81.77 ± 2.69	82.75 ± 0.71	**86.85** ± 0.52	85.86 ± 1.34	85.85 ± 0.72	**88.31** ± 0.78
MRPC	82.96 ± 0.84	81.53 ± 0.84	**83.21** ± 2.3	85.34 ± 1.3	83.40 ± 1.20	**87.03** ± 1.76

Table 10: We present a comparison of our method, Vanilla, and model soup on RTE, MRPC, STSB, and CoLA. The reported results are mean and standard deviation of the evaluation metrics over ten random seeds for each dataset at the 500 and 1K training instances. **Bold** indicates the highest performance in each row.

Datasets	Ours		Vanilla		Joint Training		Random$_\alpha$					
							σ_α = 0.005		σ_α = 0.1		σ_α = 0.45	
	mean	std	mean	std	mean	std	mean	std	mean	std	mean	std
CoLA	**66.07**	1.35	64.11	1.33	64.18	1.19	64.03	1.96	64.37	1.00	47.52	16.02
MRPC	**91.84**	0.37	90.80	1.77	89.77	2.91	91.37	0.66	90.65	0.82	85.42	2.94
RTE	**73.43**	1.52	70.69	2.83	71.48	1.39	72.20	0.97	68.30	6.76	55.85	2.64
STSB	**90.34**	0.48	89.92	0.61	90.03	0.42	89.85	0.54	89.97	0.52	88.38	0.18
AVG	**80.42**	0.93	78.88	1.64	78.86	1.48	79.36	1.03	78.32	2.27	69.29	5.44

Table 11: Comparision of our method with Vanilla, Joint-training and Random$_\alpha$ by varying σ_α to understand the effectiveness of the attention-based weights mixup and the need to learn α using a BLO framework.