AdaMix: Mixture-of-Adapter for Parameter-efficient Tuning of Large Language Models

Yaqing Wang[§], **Subhabrata Mukherjee**[†], **Xiaodong Liu**[†],
Jing Gao[§], **Ahmed Hassan Awadallah**[†], **Jianfeng Gao**[†]
[§]Purdue University, [†]Microsoft Research
{wang5075, jinggao}@purdue.edu,
{submukhe, xiaodl, hassanam, jfgao}@microsoft.com

Abstract

Fine-tuning large-scale pre-trained language models to downstream tasks require updating hundreds of millions of parameters. This not only increases the serving cost to store a large copy of the model weights for every task, but also exhibits instability during few-shot task adaptation. Parameter-efficient techniques have been developed that tune small trainable components (e.g., adapters) injected in the large model while keeping most of the model weights frozen. The prevalent mechanism to increase adapter capacity is to increase the bottleneck dimension which increases the adapter parameters. In this work, we introduce a new mechanism to improve adapter capacity without increasing parameters or computational cost by two key techniques. (i) We introduce multiple shared adapter components in each layer of the Transformer architecture. We leverage sparse learning via random routing to update the adapter parameters (encoder is kept frozen) resulting in the same amount of computational cost (FLOPs) as that of training a single adapter. (ii) We propose a simple merging mechanism to average the weights of multiple adapter components to collapse to a single adapter in each Transformer layer, thereby, keeping the overall parameters also the same but with significant performance improvement. We demonstrate these techniques to work well across multiple task settings including fully supervised and few-shot Natural Language Understanding tasks. By only tuning 0.23% of a pre-trained language model's parameters, our model[1] is the first one to fully outperform the full model fine-tuning performance and several competing methods.

1 Introduction

Large-scale language models ((Devlin et al., 2019; Liu et al., 2019; Brown et al., 2020; Raffel et al., 2019) are pre-trained in a self-supervised fashion over massive amounts of unlabeled data. Adapting these models to downstream tasks require fine-tuning all of the model parameters. Given the ever-increasing size of large pre-trained language models (PLMs) (e.g., GPT-3 consists of 175 billion parameters and MT-NLG consists of 530 billion parameters), such adaptation mechanism massively increases the serving cost since it requires storing one copy of the model weights for every task. To address these challenges, recent works have developed parameter-efficient fine-tuning techniques. These approaches typically keep most of the model weights frozen and update only a part of the model parameters or inject small trainable modules in the Transformer layers that are tuned for every task. While there are many varieties of such parameter-efficient tuning techniques, including prefix-tuning (Li & Liang, 2021) and prompt-tuning (Lester et al., 2021) to condition frozen language models via natural language descriptions of the task, low dimensional projections using adapters (Houlsby et al., 2019; Pfeiffer et al., 2020; 2021) and more recently using low-rank approximation (Hu et al., 2021). However, for all of the above methods, we observe a performance gap with respect to full model tuning where all the parameters are updated for many of the tasks.

The above parameter-efficient adaptation techniques introduce certain hyper-parameters to control for the adaptation capacity, for instance, the rank for low-rank adaptation techniques or the bottleneck

[1]Code and model checkpoints will be made available at https://aka.ms/AdaMix

dimension of adapters like Houlsby (Houlsby et al., 2019). The prevalent mechanism to increase the capacity to match the full model tuning performance is to increase the rank or the adapter width which increases the number of adapter parameters. In this work, we develop a different mechanism to increase adapter capacity to match the full model tuning performance without increasing overall number of newly-added parameters or FLOPs.

We take inspiration from sparsely-activated mixture-of-experts (MoE) models. In traditional dense models (e.g., Transformer-based language models like BERT (Devlin et al., 2019) and GPT-3 (Brown et al., 2020)), all of the model weights are activated for every input example. MoE models induce sparsity by activating only a subset of the neural network weights for each incoming example. This is achieved via conditional computation based on routing input examples to a subset of *experts* introduced in each other layer of the Transformer model. This conditional computation, for instance, selection of $top-1$ expert in each other layer, allows the sparse models to be computationally efficient i.e. match the FLOPs of that of a dense model, but also improves its capacity by increasing the number of parameters.

With this design in mind, we introduce multiple adapter components in each layer of the Transformer architecture to take best advantage of the pre-trained knowledge in PLMs. Consider the Houlsby (Houlsby et al., 2019) adapter as one of the most popular parameter-efficient fine-tuning technique for illustration. It introduces two feedforward layers to *down-project* the hidden representation to a low dimension d (also called the bottleneck dimension) followed by another *up-project* operation to match the dimensionality of the next layer. In order to introduce sparsity, we inject multiple feedforward layers (FFN) (corresponding to project-up and project-down) in each Transformer layer. We introduce a simple protocol to stochastically route instances to one of the project-up and then to one of the project-down FFN's resulting in the same amount of computational cost (FLOPs) as that of using a single adapter but introducing more capacity.

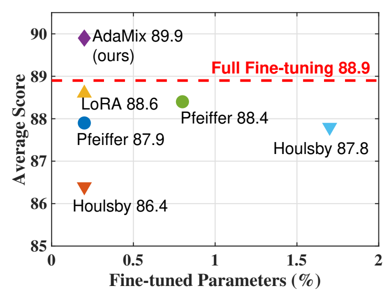


Figure 1: Performance of different parameter-efficient tuning methods on the GLUE development set with RoBERTa-large encoder. We report the performance of Pfeiffer and Houlsby adapters with their default number of tunable parameters as well as that used in our method `AdaMix`. Red dash shows the performance of full model fine-tuning.

The above design, however, introduces two major challenges. The first one results from training instability due to stochastic selection of different adapter components, e.g., routing instances via different pairs of FFN-up and FFN-down projections in different training steps. To mitigate this, we explore different design choices like consistency regularization and sharing of adapter components during stochastic routing. The second challenge results in increased number of adapter parameters that increases the serving cost although it keeps the computational cost the same. To address this, we develop a merging mechanism to average weights from differently learned adapter components to a single adapter that preserves the performance gains, while keeping the number of parameters and FLOPs also the same as that of a single adapter design. Our adapter merging is inspired by recent works on model weight averaging like model soups (Wortsman et al., 2022) and multi BERTs (Devlin et al., 2019). Such weight averaging of models with different random initialization has been shown to improve model performance in recent works (Matena & Raffel, 2021; Neyshabur et al., 2020; Frankle et al., 2020) that show the optimized models to lie in the same basin of error landscape. While the above works are geared towards fine-tuning independent models, we extend this idea to parameter-efficient fine-tuning with randomly initialized adapters and a frozen language model. Overall, our work makes the following contributions:

- We propose a new mechanism of increasing adapter capacity for parameter-efficient fine-tuning by stochastic routing to a mixture of adapter components while keeping the same computational cost (FLOPs) as that of a single adapter design.
- We propose a merging mechanism to average weights of multiple adapter components to preserve the improved performance from the aforementioned design while keeping the parameters also the same as that of a single adapter design.

- We demonstrate this simple technique to work well in different task settings and variable amounts of labeled training data, including fully supervised and few-shot fine-tuning of large language models on Natural Language Understand tasks. By tuning only 0.23% of a pre-trained model's parameters, our method outperforms the full model fine-tuning performance on GLUE and several competing methods.

2 Background

2.1 Mixture-of-Experts

The objective of sparsely-activated model design is to support conditional computation and increase the parameter count of neural models like Transformers while keeping the floating point operations (FLOPs) for each input example constant. Mixture-of-Experts (MoE) Transformer models (Shazeer et al., 2017; Fedus et al., 2021; Lepikhin et al., 2020; Zuo et al., 2021) achieve this by using N feed-forward networks (FFN), namely "experts" denoted as $\mathbb{E}_{i=1}^{N}$, each with its own set of learnable weights that compute different representations of an input token x based on context. In order to sparsify the network to keep the FLOPs constant, there is an additional gating network $\mathbb{G}$ whose output is a sparse N-dimensional vector to route each token via a few of these experts. Note that, a sparse model with $N = 1$ corresponding to only one FFN layer in each Transformer block collapses to the traditional dense model.

Consider x_s as the input token representation in the s^{th} position to the MOE layer comprising of the $\{\mathbb{E}\}_{i=1}^{N}$ expert FFNs. Also, consider w_i^{in} and w_i^{out} to be the input and output projection matrices for i^{th} expert. Expert output $\mathbb{E}_i(x_s)$ is given by:

$$\mathbb{E}_i(x_s) = w_i^{out} \cdot GeLU(w_i^{in} \cdot x_s) \tag{1}$$

Consider $\mathbb{G}(x_s)$ to be output of the gating network. Output of the sparse MoE layer is given by:

$$h(x_s) = \sum_i \mathbb{G}(x_s)_i \, \mathbb{E}_i(x_s) \tag{2}$$

where $\mathbb{G}(x_s)_i$ the i^{th} logit of the output of $\mathbb{G}(x_s)$ denotes the probability of selecting expert $\mathbb{E}_i$.

In order to keep the number of FLOPs in the sparse Transformer to be the same as that of a dense one, the gating mechanism can be constrained to route each token to only one expert FFN, i.e. $\sum_i \mathbb{G}_t(x_s)_i = 1$.

2.2 Adapters

The predominant methodology for task adaptation is to tune all of the trainable parameters of the PLMs for every task. This raises significant resource challenges both during training and deployment. A recent study (Aghajanyan et al., 2021) shows that PLMs have a low instrinsic dimension that can match the performance of the full parameter space.

To adapt PLMs for downstream tasks with a small number of parameters, adapters (Houlsby et al., 2019) have recently been introduced as an alternative approach for lightweight tuning.

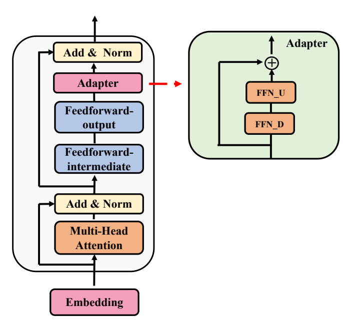


Figure 2: Conventional adapter design in standard Transformer architecture.

The adapter tuning strategy judiciously introduces new parameters into the original PLMs. During fine-tuning, only the adapter parameters are updated while keeping the remaining parameters of the PLM frozen. Adapters usually consist of two fully connected layers as shown in Figure 2, where the adapter layer uses a down projection $\mathcal{W}^{down} \in \mathcal{R}^{d \times r}$ to project input representation x to a low dimensional space r (referred as the bottleneck dimension) with d being the model dimension, followed by a nonlinear activation function

$f(\cdot)$, and a up-projection with $\mathcal{W}^{up} \in \mathcal{R}^{r \times d}$ to project the low-dimensional features back to the original dimension. The adapters are further surrounded by residual connections.

Practical benefits of lite tuning. The most significant benefit of lightweight adapter tuning comes from the reduction in memory and storage usage. For a Transformer, The VRAM consumption could be significantly reduced as we do not need to keep track of the optimizer states for the frozen parameters. For storage usage, we also reduce the checkpoint size by 444x (from 355MB to 0.8MB in our setting with RoBERTa-large encoder) since we store only task-specific adapter parameters instead of shared PLM, largely benefiting deployment scenarios.

Given the above adapter design with parameters ψ, the dataset $\mathcal{D}_K$, a pre-trained language model encoder enc with parameters Θ_{PLM}, where $\Theta_{\text{PLM}} \gg \psi$, we want to perform the following optimization for efficient model adaptation:

$$\psi \leftarrow argmin_{\psi}\ \mathcal{L}(\mathcal{D}_k; \Theta_{\text{PLM}}, \psi) \tag{3}$$

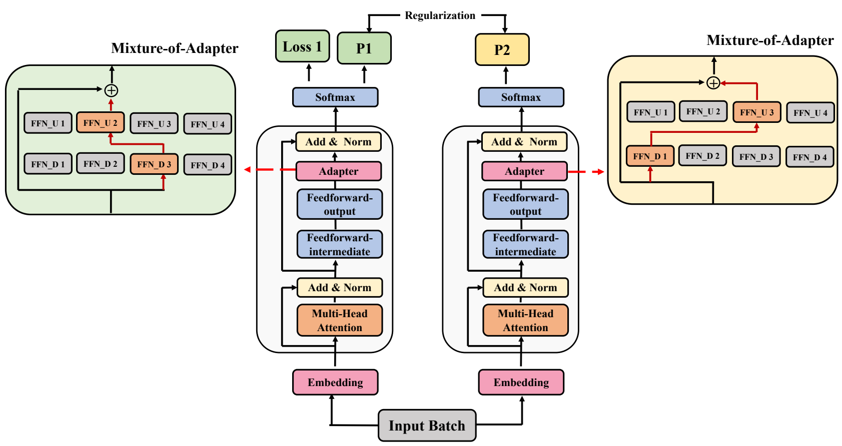


Figure 3: Mixture-of-Adapter (`AdaMix`) architecture with $M = 4$ adapter components for illustration consisting of feedforward up (FFN_U) feedforward down (FFN_D) projection matrices. The above block shown for one Transformer layer is repeated across all the layers. `AdaMix` uses a stochastic policy to route instances from an input batch through randomly selected projection matrices resulting in FLOPs match to a single adapter with consistency regularization and parameter sharing. Adapter merging (Figure 4) collapses projection matrices to match single-adapter parameters.

3 Mixture-of-Adapter

We adopt the popularly used Transformer architecture Vaswani et al. (2017) as the basic encoder consisting of L repeated Transformer blocks, where each block consists of a self-attention sub-layer, a fully connected feed-forward network (FFN) and residual connections around the sub-layers followed by layer normalization.

Consider a set of M adapters injected in each layer of the Transformer model, where $A_{ij} : i \in \{1 \cdots L\}, j \in \{1 \cdots M\}$ represents the j^{th} adapter in the i^{th} Transformer layer. Each adapter component A_{ij} in our framework follows the Houlsby Houlsby et al. (2019) adapter design consisting of a feedforward up $\mathcal{W}_{ij}^{up}$ and a feedforward down $\mathcal{W}_{ij}^{down}$ projection matrices.

Standard Mixture-of-Experts (MoE) models with token-level routing have been shown effective for autoregressive or encoder-decoder language models for tasks like Neural Machine Translation. In contrast, in this work, we focus on encoder-only models (e.g., BERT Devlin et al. (2019) and RoBERTa Liu et al. (2019)) for Natural Language Understanding tasks (e.g. tasks in the GLUE benchmark Wang et al. (2019)). Correspondingly, we adopt instance-level routing for classification tasks as opposed to token-level routing.

Recent work like THOR Zuo et al. (2021) has demonstrated stochastic routing policies like the random routing to work as well as classical routing mechanisms like Switch routing Fedus et al. (2021) with some added benefits. For instance, since input examples are randomly routed to different experts, there is no requirement for additional load balancing since each expert has an equal opportunity of being triggered, further, simplifying the framework. Additionally, there are no added parameters at the Switch layer for expert selection. The latter is particularly important in our setting for parameter-efficient fine-tuning to keep the parameters and FLOPs the same as that of a single adapter design. Correspondingly, we adopt a stochastic routing policy in our framework.

To this end, at any training step, we randomly select a pair of feedforward up and feedforward down projection matrices in the i^{th} Transformer layer as $A_i = \{\mathcal{W}_{ij}^{up}, \mathcal{W}_{ik}^{down}\}$ and $B_i = \{\mathcal{W}_{ij'}^{up}, \mathcal{W}_{ik'}^{down}\}$ respectively where $j \neq j', k \neq k'$. Given this selection of adapter components A_i and B_i in each Transformer layer in every step, all the inputs in a given batch are processed through the same set of adapters. Given an input representation x in a given Transformer layer, the above pair of adapters perform the following transformations:

$$x \leftarrow x + f(x \cdot \mathcal{W}^{down}) \cdot \mathcal{W}^{up} \tag{4}$$

Such stochastic routing enables the adapter components to learn different transformations during training and obtain multiple views of the task. However, this also creates a challenge on which sets of projection matrices to use during inference due to the random routing protocol during training. We address this challenge with the following two techniques that further allow us to collapse the adapter parameters and obtain the same computational cost (FLOPs) as that of a single adapter design.

Consistency regularization. Consider $\mathcal{A} = \{A_{i=1}^{L}\}$ and $\mathcal{B} = \{B_{i=1}^{L}\}$ to be the sets of adapter components (i.e. projection matrices) triggered during two stochastic forward passes through the network for an input x across the L layers of the Transformer model. The objective of consistency regularization is to enable the adapter components to share information and prevent divergence. To this end, we add the following consistency loss as a regularizer to the task-specific optimization loss:

$$\mathcal{L} \;=\; -\sum_{c=1}^{C}\Big(\mathcal{I}(x,c)\;\; \log \mathrm{softmax}(z_c^{\mathcal{A}}(x)) \;+\; \frac{1}{2}\big(\mathcal{KL}(z_c^{\mathcal{A}}(x)||z_c^{\mathcal{B}}(x)) \;+\; \mathcal{KL}(z_c^{\mathcal{B}}(x)||z_c^{\mathcal{A}}(x))\big)\Big) \tag{5}$$

where $\mathcal{I}(x,c)$ is a binary indicator (0 or 1) if class label c is the correct classification for x and $z_c^{\mathcal{A}}(x)$ and $z_c^{\mathcal{B}}(x)$ are the predicted logits corresponding to class c from the two sets of adapters $\mathcal{A}$ and $\mathcal{B}$ respectively with $\mathcal{KL}$ denoting the Kullback-Leibler divergence. From Equations 3 and 5, x is the input representation from the encoder $enc(\Theta_{PLM})$ with frozen parameters and only the parameters of projection matrices $\psi = \{\mathcal{W}^{up}, \mathcal{W}^{down}\}$ are updated during training.

Adapter merging. While the above regularization mitigates the inconsistency in random adapter selection during inference, it still results in increased serving cost to host all the projection matrices from the different adapter components. Prior works in fine-tuning language models for downstream tasks have shown improved performance on averaging the weights of different models fine-tuned with different random seeds outperforming a single fine-tuned model. Recent work Wortsman et al. (2022) has also shown that differently fine-tuned models from the same initialization lie in the same error basin motivating the use of weight aggregation for robust task summarization. We adopt and extend prior techniques for language model fine-tuning to our parameter-efficient training of multi-view adapters.

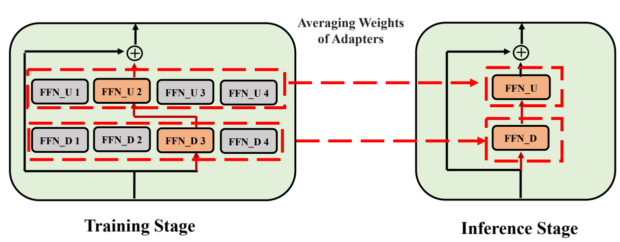


Figure 4: Stochastic routing during training triggers different projection matrices for the adapters to have multiple views of the task with FLOPs match to a single adapter. Merging weights of the adapter components ($\{\text{FFN\_U}_i\}, \{\text{FFN\_D}_i\}$: $i \in \{1 \cdots 4\}$) by averaging preserves improved performance with parameter match to a single-adapter design.

In contrast to the aforementioned techniques like stochastic routing and consistency regularization that are applied at the training phase, we employ adapter merging **only during inference**. Given a set of projection matrices, $\mathcal{W}_{ij}^{up}$ and $\mathcal{W}_{ik}^{down}$ for $i \in \{1 \cdots L\}$ and $\{j,k\} \in \{1 \cdots M\}$, we simply average the weights of all the project-up or project down matrices in every Transformer layer to collapse to a single adapter component $\{\mathcal{W'}_i^{up}, \mathcal{W'}_i^{down}\}$, where:

$$\mathcal{W'}_i^{up} \leftarrow \frac{1}{M}\sum_{j=1}^{M}\mathcal{W}_{ij}^{up} \qquad \mathcal{W'}_i^{down} \leftarrow \frac{1}{M}\sum_{j=1}^{M}\mathcal{W}_{ij}^{down} \tag{6}$$

Adapter sharing. While stochastic routing to multi-view adapters increases the model capacity, it can also impact downstream tasks with less amounts of labeled data for fine-tuning the large number of parameters. To address this challenge, we use another mechanism to share some of the projection matrices for the project-down *or* the project-up operation to reduce the number of trainable parameters during training. In the standard setting in our experiments, we share only the feedforward projection matrices i.e., $\mathcal{W}_{ij}^{up} = \mathcal{W}_i^{up}$. We investigate these different design choices via ablation studies in our experiments.

3.1 Connection to Bayesian Neural Networks and Model Ensembling

Bayesian Neural Networks (BNN) (Gal & Ghahramani, 2015) replaces a deterministic model's weight parameters by a distribution over the parameters. For inference, BNN averages over all the possible weights, also referred to as marginalization. Consider $f^{\mathcal{W}(x)} \in \mathbb{R}^d$ to be the $d-$dimensional output of such a neural network where the model likelihood is given by $p(y|f^{\mathcal{W}(x)})$. In our setting, $\mathcal{W} = \langle \mathcal{W}^{up}, \mathcal{W}^{down} \rangle$ with frozen language model encoder. For classification, we can further apply a softmax likelihood to the output to obtain: $P(y = c|x, W) = softmax(f^{\mathcal{W}(x)})$. Given an instance x, the probability distribution over the classes is given by marginalization over the posterior distribution as: $p(y = c|x) = \int_{\mathcal{W}} p(y = c|f^{\mathcal{W}(x)})p(\mathcal{W}|X, Y)d\mathcal{W}$.

This requires averaging over all possible model weights, which is intractable in practice. Therefore, several approximation methods have been developed based on variational inference methods and stochastic regularization techniques using dropouts. In this work, we leverage another stochastic regularization in the form of random routing. Here, the objective is to find a surrogate distribution $q_\theta(w)$ in a tractable family of distributions that can replace the true model posterior that is hard to compute. The ideal surrogate is identified by minimizing the Kullback-Leibler (KL) divergence between the candidate and the true posterior.

Consider $q_\theta(\mathcal{W})$ to be the stochastic routing policy which allows us to sample T masked model weights $\{\widetilde{\mathcal{W}_t}\}_{t=1}^{T} \sim q_\theta(\mathcal{W})$. For classification tasks, the approximate posterior can be now obtained by Monte-Carlo integration as:

$$\begin{aligned} p(y = c|x) &\approx p(y = c|f^{\mathcal{W}}(x))q_\theta(\mathcal{W})d\mathcal{W} \\ &\approx \frac{1}{T}\sum_{t=1}^{T} p(y = c|f^{\widetilde{\mathcal{W}_t}}(x)) = \frac{1}{T}\sum_{t=1}^{T} softmax(f^{\widetilde{\mathcal{W}_t}}(x)) \end{aligned} \tag{7}$$

However, computing the approximate posterior above in our setting requires storing all the stochastic model weights $\mathcal{W}_t(x)$ which increases the serving cost during inference. To reduce this cost, we resort to the other technique for weight averaging via adapter merging during inference.

Consider $\mathcal{L}_{\mathcal{W}}^{AM} = \mathbb{E}_{x,y}\mathcal{L}(softmax(f^{\widetilde{\mathcal{W}}}(x), y)$ denote the expected loss with merging of the stochastic adapter weights with $\widetilde{\mathcal{W}} = \frac{1}{T}\sum_t \widetilde{\mathcal{W}_t}$ (from Equation 6) and $\mathcal{L}$ denoting the cross-entropy loss. Consider $\mathcal{L}_{\mathcal{W}}^{Ens} = \mathbb{E}_{x,y}\mathcal{L}(\frac{1}{T}\sum_{t=1}^{T} softmax(f^{\widetilde{\mathcal{W}_t}}(x)), y)$ denote the expected loss from logit-level stochastic model ensembling (from Equation 7).

Prior work (Wortsman et al., 2022) show that averaging the weights of multiple models fine-tuned with different hyper-parameter configurations improves model performance. They analytically show the similarity in loss between weight-averaging ($\mathcal{L}_{\mathcal{W}}^{AM}$ in our setting) and logit-ensembling ($\mathcal{L}_{\mathcal{W}}^{Ens}$ in our setting) as a function of the flatness of the loss and confidence of the predictions. While the above analysis is geared towards averaging of multiple independently fine-tuned model weights, we can apply a similar analysis in our setting with multiple stochastic adapter weights obtained from the random routing policy to demonstrate the benefit of adapter merging in obtaining a favorable loss $\mathcal{L}_{\mathcal{W}}^{AM}$ as well as reducing the serving cost during inference. The latter is made possible since we need to retain only one copy of the merged adapter weights as opposed to logit-ensembling which requires copies of all the adapter weights.

4 Experiments

4.1 Experimental Setup

Dataset. We perform large-scale experiments with eight natural language understanding tasks in the General Language Understanding Evaluation (GLUE) benchmark Wang et al. (2019). We exclude the WNLI dataset[2] following prior studies Devlin et al. (2019); Houlsby et al. (2019). The eight tasks can be categorized into four types of natural language tasks, including linguistic acceptability (CoLA), sentiment analysis (SST-2), similarity and paraphrase tasks (MRPC, STS-B, QQP), natural language inference (MNLI, QNLI) and textual entailment task (RTE).

Baselines. We compare `AdaMix` with full model fine-tuning and several state-of-the-art parameter-efficient fine-tuning (PEFT) methods, namely, Pfeiffer Adapter Pfeiffer et al. (2021), Houlsby Adapter Houlsby et al. (2019), LoRA Hu et al. (2021), BitFit Zaken et al. (2021), Prefix-tuning Li & Liang (2021) and UNIPELT Mao et al. (2021) with BERT-base Devlin et al. (2019) and RoBERTa-large Liu et al. (2019) as encoders in Table 1 and Table 2.

Implementation Details. We implement our framework in Pytorch and use Tesla V100 gpus for experiments. `AdaMix` uses adapter dimension size of 16 and 48 using BERT-base and RoBERTa-large encoders respectively, following the setup of existing works Hu et al. (2021); Mao et al. (2021) for a fair comparison. The number of adapters in `AdaMix` is set to 4 for all the tasks and encoders unless otherwise specified. The impacts of adapter dimension size and adapter number are investigated in Table 6 and 7. More hyper-parameter configurations are presented in Appendix.

4.2 GLUE Main Results

Model	#Param.	MNLI Acc	QNLI Acc	SST2 Acc	QQP Acc /F1	MRPC Acc/F1	CoLA Mcc	RTE Acc	STS-B Pearson	**Avg.**
Fine-tuning†	355.0M	90.2	94.7	96.4	92.2/-	90.9/-	68.0	86.6	**92.4**	88.9
Pfeiffer Adapter†	3.0M	90.2	94.8	96.1	91.9/-	90.2/-	68.3	83.8	92.1	88.4
Pfeiffer Adapter†	0.8M	90.5	94.8	96.6	91.7/-	89.7/-	67.8	80.1	91.9	87.9
Houlsby Adapter†	6.0M	89.9	94.7	96.2	92.1/-	88.7/-	66.5	83.4	91.0	87.8
Houlsby Adapter†	0.8M	90.3	94.7	96.3	91.5/-	87.7/-	66.3	72.9	91.5	86.4
LoRA†	0.8M	90.6	94.8	96.2	91.6/-	90.2/-	68.2	85.2	92.3	88.6
`AdaMix`	0.8M	**90.9**	**95.4**	**97.1**	**92.3/ 89.8**	**91.9/ 94.1**	**70.2**	**89.2**	**92.4**	**89.9**

Table 1: Main results on GLUE development set with **RoBERTa-large** encoder. The best result on each task is in **bold** and "-" denotes missing measure. `AdaMix` outperforms all competing methods as well as fully fine-tuned large model with only 0.23% tunable parameters.† denotes that the reported results are taken from Hu et al. (2021). Mcc refers to Matthews correlation coefficient, and Pearson refers to Pearson correlation.The average performance is calculated based on accuracy of QQP and MRPC for an easy comparison. #Param. refers to the number of tunable parameters used during inference.

Tables 1 and 2 show the performance comparison among PEFT models with RoBERTa-large and BERT-base as the encoder respectively. Fully fine-tuned RoBERTa-large and BERT-base are used to provide the ceiling performance. We observe `AdaMix` to significantly outperform other state-of-the-art baselines on most of tasks with different encoders. Specifically, `AdaMix` with RoBERTa-large

[2]See (12) in https://gluebenchmark.com/faq.

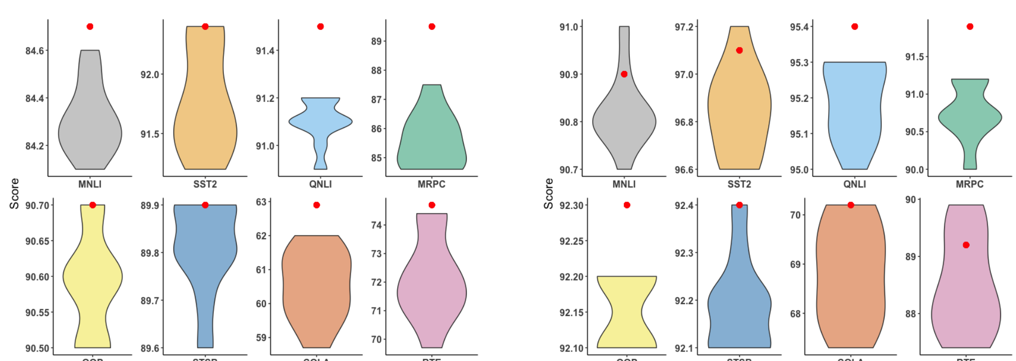
Model	#Param.	MNLI Acc	QNLI Acc	SST2 Acc	QQP Acc /F1	MRPC Acc/F1	CoLA Mcc	RTE Acc	STS-B Pearson	**Avg.**
Fine-tuning†	110M	83.2	90.0	91.6	-/87.4	-/90.9	62.1	66.4	89.8	82.7
Houlsby Adapter†	0.9M	83.1	90.6	91.9	-/86.8	-/89.9	61.5	71.8	88.6	83.0
BitFit◇	0.1M	81.4	90.2	92.1	-/84.0	-/90.4	58.8	72.3	89.2	82.3
Prefix-tuning†	0.2M	81.2	90.4	90.9	-/83.3	-/91.3	55.4	**76.9**	87.2	82.1
LoRA†	0.3M	82.5	89.9	91.5	-/86.0	-/90.0	60.5	71.5	85.7	82.2
UNIPELT (AP)†	1.1M	83.4	90.8	91.9	-/86.7	-/90.3	61.2	71.8	88.9	83.1
UNIPELT (APL)†	1.4M	83.9	90.5	91.5	85.5	-/90.2	58.6	73.7	88.9	83.5
AdaMix	0.9M 0.9M	**84.7**	**91.5**	**92.4**	**90.7/ 87.6**	**89.5/ 92.4**	**62.9**	74.7	**89.9**	**84.5**

Table 2: Main results on GLUE development set with **BERT-base** encoder. The best result on each task is in **bold** and "-" denotes the missing measure. † and ◇ denote that the reported results are taken from Mao et al. (2021); Zaken et al. (2021). The average performance is calculated based on F1 of QQP and MRPC. #Param. refers to the number of updated parameters in the inference stage.

encoder achieves the best performance in terms of different task metrics in the GLUE benchmark. AdaMix is the only PEFT method which outperforms full model fine-tuning on all the tasks and on average score. Additionally, the improvement brought by AdaMix is more significant with BERT-base as the encoder, demonstrating 2.2% and 1.2% improvement over the performance of full model fine-tuning and the best performing baseline UNIPELT with BERT-base. The improvement is observed to be consistent as that with RoBERTa-large on every task. Moreover, AdaMix outperforms all the baselines on all other tasks except RTE.

4.3 Ablation Study

Analysis of averaging adapter weights. In this ablation study, we keep separate copies of adapters to investigate the impact of weight averaging by introducing two different routing strategies for comparison. The first routing strategy is the same as the routing strategy adopted in the training stage i.e. random routing to adapter components. We denote this variation as AdaMix-RandomRouting. The second routing strategy adopts a fixed routing strategy, where we route all the input to the first adapter component in our AdaMix. The second baseline is denoted as AdaMix-FixedRouting. Table 3 shows that AdaMix outperforms AdaMix-RandomRouting and AdaMix-FixedRouting on all the tasks, demonstrating the superiority of averaging adapter weights. Moreover, AdaMix-RandomRouting and AdaMix-FixedRouting demonstrate improvement over the full model tuning, depicting the effectiveness of AdaMix design.



(a) BERT-base (b) RoBERTa-large

Figure 5: Violin plot of AdaMix-RandomRouting performance distribution with BERT-base and RoBERTa-large encoders. Red dot denotes the performance of AdaMix.

Model	#Param.	MNLI Acc	QNLI Acc	SST2 Acc	QQP Acc /F1	MRPC Acc/F1	CoLA Mcc	RTE Acc	STS-B Pearson	**Avg.**
					$\text{BERT}_{\text{BASE}}$					
Fine-tuning	110M	83.2	90.0	91.6	-/87.4	-/90.9	62.1	66.4	89.8	82.7
AdaMix	0.9M	**84.7**	**91.5**	**92.4**	**90.7/ 87.6**	**89.5/ 92.4**	**62.9**	**74.7**	**89.9**	**84.5**
AdaMix-RandomRouting	0.9∼3.6M	84.3	91.1	91.8	90.6/ 87.4	85.6/ 89.1	60.5	72.1	89.8	83.3
AdaMix-FixedRouting	0.9M	84.5	91.1	91.6	90.5/ 87.3	87.5/ 90.8	61.4	73.3	89.8	83.7
AdaMix-Ensemble	0.9∼3.6M	84.3	91.2	91.6	90.5/ 87.4	85.9/ 89.4	59.4	72.1	89.8	83.2
					$\text{RoBERTa}_{\text{LARGE}}$					
Fine-tuning	355.0M	90.2	94.7	96.4	92.2/-	90.9/-	68.0	86.6	**92.4**	88.9
AdaMix	0.8M	**90.9**	**95.4**	**97.1**	**92.3/ 89.8**	**91.9/ 94.1**	**70.2**	**89.2**	**92.4**	**89.9**
AdaMix-RandomRouting	0.8∼3.2M	90.8	95.2	96.8	92.2/ 89.6	90.8/ 93.3	68.8	88.5	92.2	89.4
AdaMix-FixedRouting	0.8M	90.7	95.1	96.8	92.1/ 89.5	91.2/ 93.6	68.6	**89.2**	92.2	89.5
AdaMix-Ensemble	0.8∼3.2M	**90.9**	95.3	97.0	92.2/ 89.7	91.0/ 93.5	69.3	89.1	**92.4**	89.7

Table 3: Comparing the impact of different routing and ensembling strategies with AdaMix. Results are presented on GLUE development set with BERT-base and RoBERTa-large encoders. Average results are calculated following Table 1 and Table 2 for consistency. The best result on each task is in **bold** and "-" denotes the missing measure.

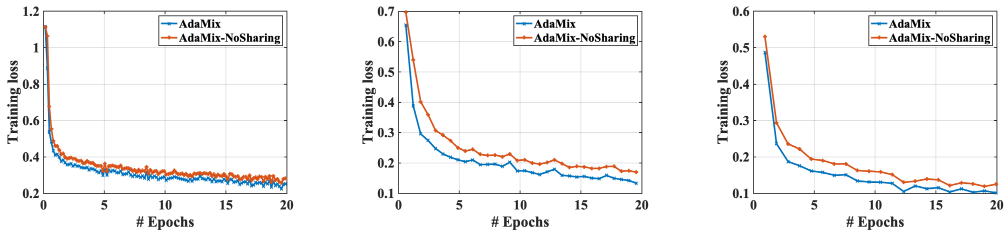
Averaging weights v.s. ensembling outputs. We further compare AdaMix with a model variant of logit ensembling method, denoted as AdaMix-Ensemble. To this end, we make four random routing passes through the network for every input (T=4) and average the logits from different passes as the final predicted logit. The inference time for this ensembling method is 4x AdaMix. We run repeated experiments with three different seeds and report the mean performance in Table 3. This experiment has two interesting take-aways: (1) AdaMix outperforms logit-ensembling method with less inference time with different encoders. (2) While logit-ensembling with a large encoder like RoBERTa-large shows some improvement in general, the one with BERT-base encoder shows significant regression.

Table 4 demonstrates the impact of other design choices of AdaMix.

Analysis of consistency loss. In AdaMix, we train the model with consistency regularization to enable adapter components to share information. To validate the contribution of the consistency loss term, we develop a model variation by dropping the consistency regularization during training. Table 4 shows significant performance drop on four out of five tasks after removing this regularizer.

Model/# Train	MNLI 393k	QNLI 108k	SST2 67k	MRPC 3.7k	RTE 2.5k
Fine-tuning	90.2	94.7	96.4	90.9	86.6
AdaMix	**90.9**	**95.4**	**97.1**	**91.9**	**89.2**
AdaMix-NoConsistencyLoss	90.7	95.0	**97.1**	91.4	84.8
AdaMix-NoSharing	**90.9**	95.0	96.4	90.4	84.1

Table 4: Ablation study demonstrating the impact of various design choices in our Mixture-of-Adapter (AdaMix) framework.



(a) MNLI (b) QNLI (c) SST2

Figure 6: Convergence analysis demonstrating the impact of adapter sharing design in our Mixture-of-Adapter (`AdaMix`).

Model	MNLI Acc	SST2 Acc
Sharing Project-up	90.9	97.1
Sharing Project-down	90.8	97.1

Table 5: Ablation study demonstrating the impact of parameter sharing in our Mixture-of-Adapter (`AdaMix`) framework.

Analysis of adapter weight sharing. Adapter weight sharing is adopted in `AdaMix` to facilitate the training convergence and prevent divergence. To investigate the role of adapter weight sharing, we remove the weight sharing mechanism and keep four different copies of projection-down layers and four projection-up layers. Table 4 demonstrates the contribution of adapter weight sharing in the training procedure. As the size of the dataset decreases (from 393k in MNLI to 2.5k in RTE), the performance gap between `AdaMix` and `AdaMix`-NoSharing is observed to be larger. Particularly, the accuracy drop is 9.4% in RTE after removing adapter weight sharing.

The role of adapter weight sharing in facilitating convergence can be further demonstrated in Figure 6 which shows the training loss trend on MNLI, QNLI and SST2. With the same number of training steps, `AdaMix` consistently shows a lower training loss compared to `AdaMix`-NoSharing, demonstrating a faster convergence behavior of `AdaMix`. We explore another interesting choice about adapter weight sharing strategy in the project-up layer versus the project-down layer. Empirical studies in Table 5 demonstrate similar effects with both of these choices.

Impact of the number of adapters. In this ablation study, we vary the number of adapters in `AdaMix` with 2, 4 and 8 during the training procedure to investigate their impact. Table 6 shows that increasing the number of adapters do not result in consistent performance gain. `AdaMix` with 4 adapters deliver the better performance when compared to that of `AdaMix` with 2 and 8. We also observe the performance variation to be related to the dataset size. As we increase sparsity and the number of parameters via increasing the number of adapters, some tasks like RTE and SST2 with less amount of labeled fine-tuning data are impacted in contrast to tasks with large amount of labeled data like MNLI.

# Adaptes/# Train	MNLI 393k	QNLI 108k	SST2 67k	MRPC 3.7k	RTE 2.5k
2	**90.9**	95.2	96.8	90.9	87.4
4*	**90.9**	**95.4**	**97.1**	**91.9**	**89.2**
8	**90.9**	95.3	96.9	91.4	87.4

Table 6: Varying the number of #Adapter components in `AdaMix` with RoBERTa-large encoder. * denotes the # Adapter used in `AdaMix`.

Impact of adapter bottleneck dimension size. To further study the effect of bottleneck dimension, we conduct experiments by varying the dimension size of adapters from 8 to 64 in `AdaMix` with BERT-base encoder, and to 32 with RoBERTa-large encoder. Table 7 shows that the model performance improves as we increase the number of trainable parameters by increasing the bottleneck dimension with diminishing returns after a certain point.

Adapter Dim	MNLI 393k	QNLI 108k	SST2 67k	MRPC 3.7k	RTE 2.5k
BERT$_{\text{BASE}}$					
8	82.2	91.1	92.2	87.3	72.6
16	83.0	91.5	92.2	88.2	72.9
32	83.6	91.3	92.2	88.5	73.6
48*	**84.7**	91.5	**92.4**	**89.5**	74.7
64	84.4	**91.8**	92.3	88.2	**75.1**
RoBERTa$_{\text{LARGE}}$					
8	90.7	95.2	96.8	91.2	87.7
16*	90.9	**95.4**	**97.1**	**91.9**	**89.2**
32	**91.0**	**95.4**	96.8	90.7	**89.2**

Table 7: Varying the bottleneck dimension of adapters in `AdaMix` with RoBERTa-large encoder. * denotes the bottleneck dimension used in `AdaMix`.

4.4 Few-shot Performance

Data. In contrast to the fully supervised setting in the above experiments, we also perform few-shot experiments following the prior study Wang et al. (2021) on six tasks including MNLI Williams et al. (2018), RTE (Dagan et al., 2005; Bar Haim et al., 2006; Giampiccolo et al., 2007; Bentivogli et al., 2009), QQP[3] and SST-2 (Socher et al.). The results are reported on their development set following (Zhang et al., 2021). MPQA (Wiebe et al., 2005) and Subj (Pang & Lee, 2004) are used for polarity and subjectivity detection, where we follow Gao et al. (2021) to keep $2,000$ examples for testing. The few-shot model only has access to $|\mathcal{K}|$ labeled samples for any task. Following *true few-shot learning* setting (Perez et al., 2021; Wang et al., 2021), we *do not use any additional validation set* for any hyper-parameter tuning or early stopping. The performance of each model is reported after fixed number of training epochs. For a fair comparison, we use the same set of few-shot labeled instances for training as in Wang et al. (2021). We train each model with 5 different seeds and report average performance with standard deviation across the runs. In the few-shot experiments, we follow Wang et al. (2021) to train `AdaMix` via the prompt-based fine-tuning strategy. In contrast to Wang et al. (2021), we do not use any unlabeled data.

Model	MNLI	RTE	QQP	SST2	Subj	MPQA	**Avg.**
Full Prompt Fine-tuning*	62.8 (2.6)	66.1 (2.2)	71.1 (1.5)	91.5 (1.0)	91.0 (0.5)	82.7 (3.8)	77.5
Head-only*	54.1 (1.1)	58.8 (2.6)	56.7 (4.5)	85.6 (1.0)	82.1 (2.5)	64.1 (2.1)	66.9
BitFit*	54.4 (1.3)	59.8 (3.5)	58.6 (4.4)	87.3 (1.1)	83.9 (2.3)	65.8 (1.8)	68.3
Prompt-tuning*	47.3 (0.2)	53.0 (0.6)	39.9 (0.7)	75.7 (1.7)	51.5 (1.4)	70.9 (2.4)	56.4
Houlsby Adapter*	35.7 (1.1)	51.0 (3.0)	62.8 (3.0)	57.0 (6.2)	83.2 (5.4)	57.2 (3.5)	57.8
LiST Adapter*	62.4 (1.7)	66.6 (3.9)	71.2 (2.6)	91.7 (1.0)	90.9 (1.3)	82.6 (2.0)	77.6
`AdaMix`	**65.6 (2.6)**	**69.6 (3.4)**	**72.6 (1.2)**	**91.8 (1.1)**	**91.5 (2.0)**	**84.7 (1.6)**	**79.3**

Table 8: Average performance and standard deviation of several parameter-efficient fine-tuning strategies based on RoBERTa-large with $|\mathcal{K}| = 30$ training labels. The best performance is shown in **bold**. Prompt-tuning, Head-only and BitFit tune $1M$ model parameters during inference. Houlsby Adapter, LiST Adapter and AdaMix tune $14M$ model parameters. * denotes that the results are taken from Wang et al. (2021).

[3] `https://www.quora.com/q/quoradata/`

Table 8 shows the performance comparison among different PEFT models with $|K| = 30$ labeled examples while fixing RoBERTa-large as the encoder. We observe that most of the PEFT methods are not able to match the performance of full model prompt-based fine-tuning (i.e. with all the model parameters being updated) besides LiST Adapter. LiST combines prompt-based fine-tuning and Houlsby adapter design and performs at par with full model prompt fine-tuning with updating only 4% model parameters. `AdaMix` outperforms all the other PEFT methods and full model prompt fine-tuning with only 4% model parameters being updated. Specifically, `AdaMix` improves over the best performing baseline LiST by 2% in aggregate performance across six tasks.

5 Related Work

Parameter-efficient fine-tuning of PLMs. Standard fine-tuning methods tune all trainable model parameters for every task. Recent efforts have focused on parameter-efficient fine-tuning (PEFT) of large PLMs by updating a small set of parameters while keeping most of parameters in PLMs frozen. Existing studies can be roughly categorized into two folds: (1) tuning a subset of existing parameters including head fine-tuning Lee et al. (2019), bias term tuning Zaken et al. (2021), (2) tuning newly-introduced parameters including adapters Houlsby et al. (2019); Pfeiffer et al. (2020), prompt-tuning Lester et al. (2021), prefix-tuning Li & Liang (2021) and low-rand adaptation Hu et al. (2021). The design of `AdaMix` update several copies of newly-introduced parameters during fine-tuning stage and then aggregates different copies of updated parameters into on component for inference. Despite the similar goal of parameter-efficient fine-tuning, `AdaMix` is a parallel direction to existing PEFT methods and has potentials to improve all the other PEFT approaches. We mainly develop our method based on adapter, which is one of the most representative PEFT methods, and leave other combinations to future work.

Mixture-of-Expert. Mixture-of-Experts models have recently achieved promising results by introducing an outrageously large number of parameters while keeping a fixed computation cost via gating mechanism. Shazeer et al., 2017 first proposed the MoE layer with a single gating network with Top-k routing and load balancing across experts. Fedus et al., 2021 propose initialization and training schemes for Top-1 routing. Zuo et al., 2021 propose a consistency regularizer loss for random routing; Yang et al., 2021 propose k Top-1 routing with expert-prototypes, and Roller et al., 2021; Lewis et al., 2021 address other load balancing issues. All the above works study sparse MoE with pre-training the entire model from scratch. In contrast, we study parameter-efficient adaptation of pre-trained language models by tuning only a very small number of sparse adapter parameters.

Averaging model weights. Recent explorations Szegedy et al. (2016); Matena & Raffel (2021); Wortsman et al. (2022); Izmailov et al. (2018) study model aggregation by averaging the weights. Matena and Raffel Matena & Raffel (2021) propose to merge pre-trained language models which are fine-tuned on various text classification tasks. Wortsman et al. (2022) explores averaging model weights from various independent runs on the same task with different hyper-parameter configurations. Different from existing works, we focus on averaging weights of newly-added parameters for parameter-efficient fine-tuning purpose. We introduce a consistency loss to connect different copies of parameters during the training to prevent divergence and observe additional performance improvement.

6 Conclusions

We develop a new method `AdaMix` for parameter-efficient fine-tuning of large pre-trained language models for NLP tasks. `AdaMix` develops a new mechanism to improve adapter capacity by injecting multiple copies of adapters into language models that are trained via stochastic routing policy to keep the same computational cost (FLOPs). During inference, the weights of different adapters are aggregated via averaging to consistently improve the task performance while keeping the same number of model parameters and the same serving cost as that of a single adapter. We validate the effectiveness of `AdaMix` via a comprehensive empirical study on the GLUE benchmark in both high-resource setting as well as few-shot learning setting on multiple tasks. With tuning only 0.23% parameters of large pre-trained language model, `AdaMix` consistently improves over full model fine-tuning that updates all the model parameters as well as other state-of-the-art parameter-efficient tuning methods.

REFERENCES

Armen Aghajanyan, Sonal Gupta, and Luke Zettlemoyer. Intrinsic dimensionality explains the effectiveness of language model fine-tuning. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 7319–7328, Online, August 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-long.568.

Roy Bar Haim, Ido Dagan, Bill Dolan, Lisa Ferro, Danilo Giampiccolo, Bernardo Magnini, and Idan Szpektor. The second PASCAL recognising textual entailment challenge. 2006.

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

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. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 1877–1901. Curran Associates, Inc., 2020. URL `https://proceedings.neurips.cc/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf`.

Ido Dagan, Oren Glickman, and Bernardo Magnini. The PASCAL recognising textual entailment challenge. In *the First International Conference on Machine Learning Challenges: Evaluating Predictive Uncertainty Visual Object Classification, and Recognizing Textual Entailment*, 2005.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Volume 1 (Long and Short Papers)*, pp. 4171–4186, 2019.

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

Jonathan Frankle, Gintare Karolina Dziugaite, Daniel Roy, and Michael Carbin. Linear mode connectivity and the lottery ticket hypothesis. In *International Conference on Machine Learning*, pp. 3259–3269. PMLR, 2020.

Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. *CoRR*, abs/1506.02142, 2015. URL `http://arxiv.org/abs/1506.02142`.

Tianyu Gao, Adam Fisch, and Danqi Chen. Making pre-trained language models better few-shot learners. In *Association for Computational Linguistics (ACL)*, 2021.

Danilo Giampiccolo, Bernardo Magnini, Ido Dagan, and Bill Dolan. The third PASCAL recognizing textual entailment challenge. In *the ACL-PASCAL Workshop on Textual Entailment and Paraphrasing*, 2007.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for nlp. In *International Conference on Machine Learning*, pp. 2790–2799. PMLR, 2019.

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

Pavel Izmailov, Dmitrii Podoprikhin, Timur Garipov, Dmitry Vetrov, and Andrew Gordon Wilson. Averaging weights leads to wider optima and better generalization. *arXiv preprint arXiv:1803.05407*, 2018.

Jaejun Lee, Raphael Tang, and Jimmy Lin. What would elsa do? freezing layers during transformer fine-tuning. *arXiv preprint arXiv:1911.03090*, 2019.

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

Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning. *CoRR*, abs/2104.08691, 2021. URL `https://arxiv.org/abs/2104.08691`.

Mike Lewis, Shruti Bhosale, Tim Dettmers, Naman Goyal, and Luke Zettlemoyer. Base layers: Simplifying training of large, sparse models. In *ICML*, 2021. URL `https://arxiv.org/pdf/2103.16716.pdf`.

Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. *CoRR*, abs/2101.00190, 2021. URL `https://arxiv.org/abs/2101.00190`.

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

Yuning Mao, Lambert Mathias, Rui Hou, Amjad Almahairi, Hao Ma, Jiawei Han, Wen-tau Yih, and Madian Khabsa. Unipelt: A unified framework for parameter-efficient language model tuning. *arXiv preprint arXiv:2110.07577*, 2021.

Michael Matena and Colin Raffel. Merging models with fisher-weighted averaging. *arXiv preprint arXiv:2111.09832*, 2021.

Behnam Neyshabur, Hanie Sedghi, and Chiyuan Zhang. What is being transferred in transfer learning? *Advances in neural information processing systems*, 33:512–523, 2020.

Bo Pang and Lillian Lee. A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts. 2004.

Ethan Perez, Douwe Kiela, and Kyunghyun Cho. True few-shot learning with language models. *arXiv preprint arXiv:2105.11447*, 2021.

Jonas Pfeiffer, Andreas Rücklé, Clifton Poth, Aishwarya Kamath, Ivan Vulić, Sebastian Ruder, Kyunghyun Cho, and Iryna Gurevych. Adapterhub: A framework for adapting transformers. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP 2020): Systems Demonstrations*, pp. 46–54, Online, 2020. Association for Computational Linguistics. URL `https://www.aclweb.org/anthology/2020.emnlp-demos.7`.

Jonas Pfeiffer, Aishwarya Kamath, Andreas Rücklé, Kyunghyun Cho, and Iryna Gurevych. Adapterfusion: Non-destructive task composition for transfer learning. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pp. 487–503, 2021.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv preprint arXiv:1910.10683*, 2019.

Stephen Roller, Sainbayar Sukhbaatar, Arthur D. Szlam, and Jason Weston. Hash layers for large sparse models. *ArXiv*, abs/2106.04426, 2021. URL `https://arxiv.org/pdf/2106.04426.pdf`.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*, 2017.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank.

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2818–2826, 2016.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. GLUE: A multi-task benchmark and analysis platform for natural language understanding. 2019.

Yaqing Wang, Subhabrata Mukherjee, Xiaodong Liu, Jing Gao, Ahmed Hassan Awadallah, and Jianfeng Gao. List: Lite self-training makes efficient few-shot learners. *arXiv preprint arXiv:2110.06274*, 2021.

Janyce Wiebe, Theresa Wilson, and Claire Cardie. Annotating expressions of opinions and emotions in language. *Language resources and evaluation*, 39(2):165–210, 2005.

Adina Williams, Nikita Nangia, and Samuel Bowman. A broad-coverage challenge corpus for sentence understanding through inference. 2018.

Mitchell Wortsman, Gabriel Ilharco, Samir Yitzhak Gadre, Rebecca Roelofs, Raphael Gontijo-Lopes, Ari S Morcos, Hongseok Namkoong, Ali Farhadi, Yair Carmon, Simon Kornblith, et al. Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time. *arXiv preprint arXiv:2203.05482*, 2022.

An Yang, Junyang Lin, Rui Men, Chang Zhou, Le Jiang, Xianyan Jia, Ang Wang, Jie Zhang, Jiamang Wang, Yong Li, et al. M6-t: Exploring sparse expert models and beyond. *arXiv preprint arXiv:2105.15082*, 2021. URL `https://arxiv.org/pdf/2105.15082.pdf`.

Elad Ben Zaken, Shauli Ravfogel, and Yoav Goldberg. Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. *arXiv preprint arXiv:2106.10199*, 2021.

Tianyi Zhang, Felix Wu, Arzoo Katiyar, Kilian Q Weinberger, and Yoav Artzi. Revisiting few-sample BERT fine-tuning. 2021.

Simiao Zuo, Xiaodong Liu, Jian Jiao, Young Jin Kim, Hany Hassan, Ruofei Zhang, Tuo Zhao, and Jianfeng Gao. Taming sparsely activated transformer with stochastic experts. *arXiv preprint arXiv:2110.04260*, 2021.

Task	Learning rate	epoch	batch size	warmup	weight decay	adapter size	adapter num
				BERT$_{\text{BASE}}$			
MRPC	4e−4	100	16	0.06	0.1	48	4
CoLA	5e−4	100	16	0.06	0.1	48	4
SST	4e-4	40	64	0.06	0.1	48	4
STS-B	5e-4	80	32	0.06	0.1	48	4
QNLI	4e-4	20	64	0.06	0.1	48	4
MNLI	4e-4	40	64	0.06	0.1	48	4
QQP	5e-4	60	64	0.06	0.1	48	4
RTE	5e-4	80	64	0.06	0.1	48	4
				RoBERTa$_{\text{LARGE}}$			
MRPC	3e-4	60	64	0.6	0.1	16	4
CoLA	3e-4	80	64	0.6	0.1	16	4
SST	3e-4	20	64	0.6	0.1	16	4
STS-B	3e-4	80	64	0.6	0.1	16	4
QNLI	3e-4	20	64	0.6	0.1	16	4
MNLI	3e-4	20	64	0.6	0.1	16	4
QQP	5e-4	80	64	0.6	0.1	16	4
RTE	5e-4	60	64	0.6	0.1	16	4

Table 9: Hyperparameter Setup for GLUE tasks.