Parameter-efficient Multi-task Fine-tuning for Transformers via Shared Hypernetworks

Rabeeh Karimi Mahabadi*
EPFL University, Idiap Research Institute
rabeeh.karimimahabadi@epfl.ch

Sebastian Ruder
DeepMind
ruder@google.com

Mostafa Dehghani
Google Brain
dehghani@google.com

James Henderson
Idiap Research Institute
james.henderson@idiap.ch

Abstract

State-of-the-art parameter-efficient fine-tuning methods rely on introducing adapter modules between the layers of a pretrained language model. However, such modules are trained separately for each task and thus do not enable sharing information across tasks. In this paper, we show that we can learn adapter parameters for all layers and tasks by generating them using shared hypernetworks, which condition on task, adapter position, and layer id in a transformer model. This parameter-efficient multi-task learning framework allows us to achieve the best of both worlds by sharing knowledge across tasks via hypernetworks while enabling the model to adapt to each individual task through task-specific adapters. Experiments on the well-known GLUE benchmark show improved performance in multi-task learning while adding only 0.29% parameters per task. We additionally demonstrate substantial performance improvements in few-shot domain generalization across a variety of tasks. Our code is publicly available in https://github.com/rabeehk/hyperformer.

1 Introduction

Transfer learning from pretrained large-scale language models yields state-of-the-art results in a variety of tasks (Devlin et al., 2019; Radford et al., 2018; Liu et al., 2019b). As a highly expressive and abstract framework, Raffel et al. (2020) explored the landscape of transfer learning by converting text-based natural language processing (NLP) problems into a sequence-to-sequence format to train a unified model on several tasks simultaneously. Multi-task learning with pretrained language models (Ruder, 2017) is appealing for multiple reasons: 1) Training individual models per task results in higher computational costs, which hinders deployment and maintenance. These costs are substantially reduced by training a single model. 2) Fine-tuning the model across multiple tasks allows sharing information between the different tasks and positive transfer to other related tasks. Specifically, when target datasets have limited training data, multi-task learning improves the performance compared to individually trained models (Liu et al., 2019a; Ratner et al., 2018). However, multi-task fine-tuning can result in models underperforming on high-resource tasks due to constrained capacity (Arivazhagan et al., 2019; McCann et al., 2018). An additional issue with multi-task fine-tuning is the potential for *task interference* or *negative transfer*,

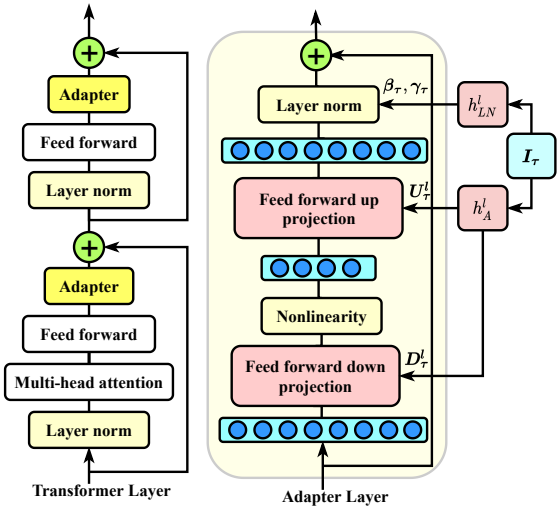

Figure 1: Left: Adapter integration in the T5 model. Right: Our HYPERFORMER adapter architecture. Following Houlsby et al. (2019), we include adapter modules after the two feed-forward layers. The Adapter hypernetwork h^l_A produces the weights ($\boldsymbol{U^l_\tau}$ and $\boldsymbol{D^l_\tau}$) for task-specific adapter modules conditioned on an input task embedding $\boldsymbol{I_\tau}$. Similarly, the layer normalization hypernetwork h^l_{LN} generates the conditional layer normalization parameters ($\boldsymbol{\beta_\tau}$ and $\boldsymbol{\gamma_\tau}$). During training, we only update layer normalizations in T5, hypernetworks, and task embeddings. The compact HYPERFORMER++ shares the same hypernetworks across all layers and tasks and computes the task embedding based on task, layer id, and position of the adapter module (§2.4).

*Work done while the author was at Google.

where achieving good performance on one task can hinder performance on another (Wang et al., 2019c).

As an alternative to fine-tuning (Howard and Ruder, 2018), adapter layers (Houlsby et al., 2019) insert a small number of additional parameters per task into the model. During fine-tuning, only the adapter modules, layer normalizations, and parameters of the final classification layer are updated, while the original pretrained model parameters remain frozen. Such task-specific adapters eliminate negative task interference by encapsulating task-specific information (Pfeiffer et al., 2020). However, so far there has not been an effective and parameter-efficient way to share information across multiple adapters to enable positive transfer to low-resource and related tasks.

To address this problem and to enable sharing information across tasks while reaping the benefits of adapter layers, as depicted in Figure 1, we propose HYPERFORMER++, which employs a compact hypernetwork (Ha et al., 2017; Oswald et al., 2020) shared across tasks and layers. The hypernetwork learns to generate task and layer-specific adapter parameters, conditioned on task and layer id embeddings. The hypernetwork is jointly learned between all tasks and is thus able to share information across them, while negative interference is minimized by generating separate adapter layers for each task. For each new task, our model only requires learning an additional task embedding, reducing the number of trained parameters.

We use the encoder-decoder T5 model (Raffel et al., 2020) as the underlying model for our experiments and evaluate on the standard GLUE benchmark (Wang et al., 2019b). We achieve strong gains over both the T5_{BASE} model as well as adapters (Houlsby et al., 2019). To our knowledge, this is the first time that adapters have been successfully integrated into a state-of-the-art encoder-decoder model beyond machine translation (Bapna and Firat, 2019), demonstrating that our method effectively balances sharing information across tasks while minimizing negative transfer.

In summary, we make the following contributions: (1) We propose a parameter-efficient method for multi-task fine-tuning based on hypernetworks and adapter layers. (2) We demonstrate that our method scales more efficiently than prior work. (3) We provide empirical results on GLUE demonstrating the effectiveness of the proposed method on multi-task learning. (4) We perform extensive few-shot domain transfer experiments, which reveal that the captured shared knowledge can positively transfer to unseen in-domain tasks. We release our code to facilitate future work.

2 HYPERFORMER

In this section, we present our HYPERFORMER model, which integrates **hyper**network-based adapter layers into a multi-task trans**former** model. In §2.4, we introduce a parameter-efficient variant of this model, called HYPERFORMER++.

Problem formulation: We consider a general multi-task learning problem, where we are given the data from a set of tasks $\{\mathcal{D}_\tau\}_{\tau=1}^{T}$, where T is the total number of tasks and $\mathcal{D}_\tau = \{(\boldsymbol{x}_\tau^i, y_\tau^i)\}_{i=1}^{N_\tau}$ shows the training data for τ-th task with N_τ samples. We assume we are also given a large-scale pretrained language model $f_{\boldsymbol{\theta}}(.)$ parameterized by $\boldsymbol{\theta}$ that computes the output for input $\boldsymbol{x}_\tau^i$. Standard multi-task fine-tuning minimizes the following loss on the training set:

$$\mathcal{L}(\boldsymbol{\theta}, \{\mathcal{D}_\tau\}_{\tau=1}^{T}) = \sum_{\tau=1}^{T} \sum_{(\boldsymbol{x}_\tau^i, y_\tau^i) \in \mathcal{D}_\tau} w_\tau l\Big(f_{\boldsymbol{\theta}}(\boldsymbol{x}_\tau^i), y_\tau^i\Big), \quad (1)$$

where l is typically the cross-entropy loss, and w_τ shows the sampling weight for τ-th task. Our goal is to finetune the pretrained model in a multi-task learning setup efficiently, while allowing sharing information across tasks and at the same time, enabling the model to adapt to each individual task.

The key idea of our approach, depicted in Figure 1, is to learn a parametric task embedding $\{\boldsymbol{I}_\tau\}_{\tau=1}^{T}$ for each task, and then feed these task embeddings to hypernetworks parameterized by $\boldsymbol{\nu}$ that generate the task-specific adapter layers (Houlsby et al., 2019). We insert adapter modules within the layers of a pretrained model, making the final model of $\mathcal{X}_{\boldsymbol{\nu}}(\boldsymbol{x}_\tau^i, \boldsymbol{\theta}, \boldsymbol{I}_\tau)$ parameterized by $\boldsymbol{\nu}$ that computes the output for input $\boldsymbol{x}_\tau^i$. During training, we only train hypernetwork parameters $\boldsymbol{\nu}$, task embeddings $\{\boldsymbol{I}_\tau\}_{\tau=1}^{T}$, and layer normalizations in $f_\theta(.)$, while the rest of the pretrained model parameters $\boldsymbol{\theta}$ are fixed:

$$\mathcal{L}(\boldsymbol{\nu}, \{\boldsymbol{I}_\tau\}_{i=1}^{T}, \{\mathcal{D}_\tau\}_{\tau=1}^{T}) = \sum_{\tau=1}^{T} \sum_{(\boldsymbol{x}_\tau^i, y_\tau^i) \in \mathcal{D}_\tau} w_\tau l\Big(\mathcal{X}_{\boldsymbol{\nu}}(\boldsymbol{x}_\tau^i, \boldsymbol{\theta}, \boldsymbol{I}_\tau), y_\tau^i\Big), \quad (2)$$

The hypernetworks capture the shared information across tasks in a multi-task learning model enabling positive transfer between related domains and transferable tasks, while adapters are reducing negative interference, encapsulating task-specific information.

Base model: All of our models are built on top of the state-of-the-art T5 transformer model (Raffel

et al., 2020). This model frames text-based language tasks as sequence-to-sequence problems. T5 consists of an encoder-decoder Transformer (Vaswani et al., 2017) with minor modifications (Raffel et al., 2020). The model is trained simultaneously on multiple tasks, obtaining state-of-the-art performance across a diverse set of tasks. We use the T5 framework as it enables training a universal model that interfaces with many language tasks. Our model has three main components: 1) task conditional adapter layers; 2) task conditional layer normalizations; and 3) hypernetworks that generate task-specific parameters. We next describe these components.

2.1 Task Conditional Adapter Layers

Prior work has shown that fine-tuning all parameters of the model can result in a sub-optimal solution, particularly for resource-limited datasets (Peters et al., 2019). As an alternative to fine-tuning all the model's parameters, prior work (Houlsby et al., 2019; Rebuffi et al., 2018; Stickland and Murray, 2019) inserted small modules called *adapter layers* within layers of a pretrained model, as shown in Figure 1. Adapters introduce no change to the structure or parameters of the original model.

In this work, we propose conditional adapter modules, in which we generate the adapters weights based on input task embeddings using shared hypernetworks (Ha et al., 2017), which capture information across tasks that can be used to positively transfer to other relevant tasks.

Each layer of a transformer model consists of an attention block and a feed-forward block, each followed by a skip connection. Following Houlsby et al. (2019), as depicted in Figure 1, we introduce a conditional adapter layer after each block before the skip connection. The conditional adapter layer A^l_τ for layer l consists of a down-projection, $\boldsymbol{D^l_\tau} \in \mathbb{R}^{h \times d}$, GeLU non-linearity (Hendrycks and Gimpel, 2016), and up-projection $\boldsymbol{U^l_\tau} \in \mathbb{R}^{d \times h}$, where h is the input dimension, and d is the bottleneck dimension for the adapter layer, mathematically defined as:

$$A^l_\tau(\boldsymbol{x}) = LN^l_\tau\Big(\boldsymbol{U^l_\tau}(\text{GeLU}(\boldsymbol{D^l_\tau}(\boldsymbol{x})))\Big) + \boldsymbol{x}, \quad (3)$$

where $\boldsymbol{x}$ is the input hidden state and LN^l_τ is the conditional layer norm defined in the next section. We generate adapter weights ($\boldsymbol{U^l_\tau}$, $\boldsymbol{D^l_\tau}$) through a hypernetwork described in §2.3.

2.2 Task Conditional Layer Normalization

Conventional layer normalization (Ba et al., 2016) is defined as:

$$LN^l_\tau(\boldsymbol{x^i_\tau}) = \boldsymbol{\gamma^l_\tau} \odot \frac{\boldsymbol{x^i_\tau} - \boldsymbol{\mu_\tau}}{\boldsymbol{\sigma_\tau}} + \boldsymbol{\beta^l_\tau}, \quad (4)$$

where $\odot$ is the element-wise multiplication between two vectors, and $\boldsymbol{\gamma^l_\tau}$ and $\boldsymbol{\beta^l_\tau}$ are learnable parameters with the same dimension as $\boldsymbol{x^i_\tau}$. Values of $\boldsymbol{\mu_\tau}$ and $\boldsymbol{\sigma_\tau}$ show the mean and standard deviation of training data for the τ-th task.

To allow the layer normalization inside adapters to adapt to each task, inspired by Perez et al. (2018); De Vries et al. (2017), we generate $\boldsymbol{\gamma^l_\tau}$, $\boldsymbol{\beta^l_\tau}$ via a hypernetwork as a function of task embeddings (§2.3).

2.3 Task Conditioned Hypernetworks

In order to have a model that can share information while being able to adapt to each individual task, we generate the parameters of task conditional adapter layers and layer normalization using hypernetworks. A hypernetwork is a network that generates the weights of another network (Ha et al., 2017).

The hypernetworks capture the shared information, while the generated task conditional adapters and layer normalization allow the model to adapt to each individual task to reduce negative task interference.

Learned task embedding: We first compute a task embedding $\boldsymbol{I_\tau} \in \mathbb{R}^t$ for each individual task using a task projector network $h_I(.)$, which is a multi-layer perceptron consisting of two feed-forward layers and a ReLU non-linearity:

$$\boldsymbol{I_\tau} = h_I(\boldsymbol{z_\tau}), \quad (5)$$

where $\boldsymbol{z_\tau} \in \mathbb{R}^{t'}$ can be a learnable parameter or any pretrained task features (Vu et al., 2020), and the task projector network $h_I(.)$ learns a suitable compressed task embedding from input task features. In this work, we consider a parametric $\boldsymbol{z_\tau}$ to allow end-to-end training which is convenient in practice.[1]

Removing task prefixes: The T5 model prepends task-specific prefixes to the input sequence for conditioning. For instance, when training on CoLA (Warstadt et al., 2019), *cola sentence:* is prepended to each sample. Instead, we remove task prefixes and use task embeddings for conditioning.

Task conditioned hypernetworks: We consider simple linear layers as hypernetworks that are functions of input task embeddings $\boldsymbol{I_\tau}$. We introduce these hypernetworks in each layer of the transformer. We define hypernetwork $h^l_A(.)$ that generates task conditional adapter weights ($\boldsymbol{U^l_\tau}$, $\boldsymbol{D^l_\tau}$):

[1] We ran some pilot experiments with pretrained task embeddings (Vu et al., 2020), but did not observe extra benefits.

$$(\boldsymbol{U}^{\boldsymbol{l}}_{\boldsymbol{\tau}},\boldsymbol{D}^{\boldsymbol{l}}_{\boldsymbol{\tau}}) := h^l_A(\boldsymbol{I}_{\boldsymbol{\tau}}) = \left(\boldsymbol{W}^{\boldsymbol{U}^{\boldsymbol{l}}},\boldsymbol{W}^{\boldsymbol{D}^{\boldsymbol{l}}}\right)\boldsymbol{I}_{\boldsymbol{\tau}}, \quad (6)$$

where $\boldsymbol{W}^{\boldsymbol{U}^{\boldsymbol{l}}} \in \mathbb{R}^{(d\times h)\times t}$ and $\boldsymbol{W}^{\boldsymbol{D}^{\boldsymbol{l}}} \in \mathbb{R}^{(h\times d)\times t}$ are the respective hypernetwork parameters. We additionally define the hypernetwork $h^l_{LN}(.)$ that computes the layer normalization parameters:

$$(\boldsymbol{\gamma}^{\boldsymbol{l}}_{\boldsymbol{\tau}},\boldsymbol{\beta}^{\boldsymbol{l}}_{\boldsymbol{\tau}}) := h^l_{LN}(\boldsymbol{I}_{\boldsymbol{\tau}}) = \left(\boldsymbol{W}^{\boldsymbol{\gamma}^{\boldsymbol{l}}},\boldsymbol{W}^{\boldsymbol{\beta}^{\boldsymbol{l}}}\right)\boldsymbol{I}_{\boldsymbol{\tau}}, \quad (7)$$

where $\boldsymbol{W}^{\boldsymbol{\gamma}^{\boldsymbol{l}}} \in \mathbb{R}^{h\times t}$ and $\boldsymbol{W}^{\boldsymbol{\beta}^{\boldsymbol{l}}} \in \mathbb{R}^{h\times t}$.

2.4 HYPERFORMER++

A downside of introducing a separate hypernetwork in each layer of the Transformer is that it increases the overall number of parameters. We, therefore, propose to share hypernetworks across transformer layers. By having a shared hypernetwork that is reusable, this strategy results in a substantial reduction in the number of parameters. However, reapplying the same hypernetwork across all the layers introduces weight sharing across target parameters, which may not be desirable. To allow for a flexible parameterization of task conditional adapters/layer normalization, for a transformer of L layers, we introduce a set of *layer id* embeddings $\mathcal{I} = \{\boldsymbol{l}_i\}^L_{i=1}$, and *adapter position* embeddings $\mathcal{P} = \{\boldsymbol{p}_j\}^2_{j=1}$, which specify the position of adapter layers in each transformer block (after the attention layer or feed-forward layer), which are used as additional inputs to the hypernetworks. For simplicity, we consider $\boldsymbol{l}_i \in \mathbb{R}^t$, $\boldsymbol{p}_j \in \mathbb{R}^t$, and $\boldsymbol{z}_{\boldsymbol{\tau}} \in \mathbb{R}^t$. We feed a concatenation of $(\boldsymbol{z}_{\boldsymbol{\tau}},\boldsymbol{l}_i,\boldsymbol{p}_j)$ to a similar task projector network h'_I as in Eq. (5):

$$\boldsymbol{I}_{\boldsymbol{\tau}} = h'_I(\boldsymbol{z}_{\boldsymbol{\tau}},\boldsymbol{l}_i,\boldsymbol{p}_j), \quad (8)$$

which is then followed by a shared layer normalization to compute final task embeddings $\boldsymbol{I}_{\boldsymbol{\tau}} \in \mathbb{R}^t$ to the hypernetwork. This way, the hypernetwork is able to produce distinct weights for each task, adapter position, and layer of a transformer. Furthermore, layer id and adapter position embeddings are parameters that are learned via back-propagation, allowing us to train the whole model end-to-end conveniently.

3 Experiments

Datasets: Following Raffel et al. (2020), we evaluate the performance of the models on the GLUE benchmark (Wang et al., 2019b). This benchmark covers multiple tasks of paraphrase detection (MRPC, QQP), sentiment classification (SST-2), natural language inference (MNLI, RTE, QNLI), and linguistic acceptability (CoLA).[2] The original test sets are not publicly available, and following Zhang et al. (2021), for datasets fewer than 10K samples (RTE, MRPC, STS-B, CoLA), we divide the original validation set in half, using one half for validation and the other for the test. For the other larger datasets, we split 1k samples from the training set as our validation data and test on the original validation set.

Experimental details: We use the HuggingFace implementation (Wolf et al., 2020a) of the T5 model (Raffel et al., 2020). We fine-tune all models with a constant learning rate of 0.0003 and following Raffel et al. (2020), we use $2^{18} = 262144$ steps in all experiments. We save a checkpoint every 1000 steps for all models (see also §A). Raffel et al. (2020) report the results based on the best checkpoint for each task independently. In contrast, we focus on the more realistic setting where we report the results on a single checkpoint with the highest average validation performance across all tasks. The hyperparameters are selected in the same manner. In contrast to prior work (Houlsby et al., 2019), we do not learn a separate output layer for each task but instead share a frozen output layer for all the tasks, which makes our setting more parameter-efficient than prior work and is an advantage of multi-task learning with encoder-decoder models.[3]

Baselines: We compare to the strong adapter baseline (Houlsby et al., 2019). Following Houlsby et al. (2019), we add adapters modules for each task after the two feed-forward modules in each transformer block of the T5 model. As suggested in Houlsby et al. (2019), we train the layer normalization parameters inside the T5 model, per task. We refer to this method as *Adapters*. We additionally propose a variant of this model, in which we share all layer normalization parameters (T5 and adapters) across all tasks. We refer to this model as *Adapters*†. We compare our models to the state-of-the-art T5 model, in which we fine-tune all parameters of the model on all tasks. We refer to this method as T5_{SMALL}/T5_{BASE} in experiments.

Sampling tasks: During training, we sample tasks with conventional temperature-based sampling with temperature $T = 10$ for all methods. We sample different tasks proportional to $p_\tau^{1/T}$ where $p_\tau = \frac{N_\tau}{\sum_{i=1}^{T} N_\tau}$ and N_τ is the number of training samples for the τ-th task. We did not experiment with more complex sampling strategies (Raffel et al., 2020) or tuning of T.

[2] Following Raffel et al. (2020); Devlin et al. (2019), as a common practice, due to the adversarial nature of WNLI with respect to the training set, we do not experiment with WNLI.

[3] According to our initial experiments, fine-tuning the final output layer did not improve performance for adapter-based methods.

Model	#Total params	#Trained params / per task	CoLA	SST-2	MRPC	QQP	STS-B	MNLI	QNLI	RTE	Avg
Single-Task Training											
$T5_{SMALL}$	8.0×	100%	**46.81**	**90.47**	**86.21/90.67**	**91.02/87.96**	**89.11/88.70**	**82.09**	**90.21**	**59.42**	**82.06**
$Adapters_{SMALL}$ ❣	1+8×0.01	**0.74%**	40.12	89.44	85.22/89.29	90.04/86.68	83.93/83.62	81.58	89.11	55.80	79.53
$T5_{BASE}$	8.0×	100%	54.85	92.19	**88.18/91.61**	**91.46/88.61**	**89.55/89.41**	**86.49**	91.60	67.39	84.67
$Adapters_{BASE}$ ❣	1+8×0.01	**0.87%**	**59.49**	**93.46**	88.18/91.55	90.94/88.01	87.44/87.18	86.38	**92.26**	**68.84**	**84.88**
Multi-Task Training											
$T5_{SMALL}$ ♠	1.0×	12.5%	50.67	**91.39**	84.73/88.89	**89.53/86.31**	**88.70/88.27**	81.04	89.67	59.42	81.69
$Adapters\dagger_{SMALL}$	1.05×	0.68%	39.87	90.01	88.67/91.81	88.51/84.77	88.15/87.89	79.95	89.60	60.14	80.85
$HYPERFORMER_{SMALL}$	1.45×	5.80%	47.64	**91.39**	**90.15/92.96**	88.68/85.08	87.49/86.96	**81.24**	**90.39**	65.22	82.47
$HYPERFORMER{+}{+}_{SMALL}$	1.04×	**0.50%**	**53.96**	90.59	84.24/88.81	88.44/84.46	87.73/87.26	80.69	**90.39**	**71.01**	**82.51**
$T5_{BASE}$ ♠	1.0×	12.5%	54.88	92.54	90.15/93.01	**91.13/88.07**	88.84/88.53	85.66	92.04	75.36	85.47
$Adapters\dagger_{BASE}$	1.07×	0.82%	61.53	93.00	90.15/92.91	90.47/87.26	89.86/89.44	86.09	**93.17**	70.29	85.83
$HYPERFORMER_{BASE}$	1.54×	6.86%	61.32	93.80	**90.64/93.33**	90.13/87.18	89.55/89.03	**86.33**	92.79	**78.26**	**86.58**
$HYPERFORMER{+}{+}_{BASE}$	1.02×	**0.29%**	**63.73**	**94.03**	89.66/92.63	90.28/87.20	**90.00/89.66**	85.74	93.02	75.36	86.48

Table 1: Performance of all models on the GLUE tasks. For each method, we report the total number of parameters across all tasks and the number of parameters that are trained for each task as a multiple and proportion respectively of the corresponding single-task T5 model. For MNLI, we report accuracy on the matched validation set. For MRPC and QQP, we report accuracy and F1. For STS-B, we report Pearson and Spearman correlation coefficients. For CoLA, we report Matthews correlation. For all other tasks, we report accuracy. Adapters† refers to our proposed variant of adapters with shared layer normalizations. Our HYPERFORMER++ obtains a better score on average compared to full fine-tuning and Adapters†, while being more parameter-efficient. ♠: Our re-implementation of Raffel et al. (2020), ❣: Applying method of Houlsby et al. (2019) on T5. Bold fonts indicate the best results in each block.

3.1 Results on the GLUE Benchmark

Table 1 shows the results on GLUE for single-task and multi-task training. We experiment with reduction factors of $r = \{8, 16, 32\}$ for all adapter-based methods, where $r = \frac{h}{d}$. We report the results both with $T5_{SMALL}$ (6 layers and 60M parameters) and $T5_{BASE}$ models (12 layers and 222M parameters).

Overall, our proposed HYPERFORMER++ obtains strong gains over Adapters (82.51 versus 79.53 for $T5_{SMALL}$ and 86.48 versus 84.88 for $T5_{BASE}$) while being more parameter-efficient.

Our variant of Adapters†, which shares layer norms across tasks, outperforms prior work (Houlsby et al., 2019), which does not share such information (80.85 versus 79.53 for $T5_{SMALL}$ and 85.83 versus 84.88 for $T5_{BASE}$). This demonstrates that in encoder-decoder models such as T5 more sharing of information across tasks is beneficial.

Our proposed HYPERFORMER obtains consistent improvement over our proposed Adapters† method. We attribute this improvement to the ability to learn the shared information across tasks through our hypernetworks. Interestingly, HYPERFORMER++ obtains similar performance as HYPERFORMER while being more than an order of magnitude more parameter-efficient. Adapter modules thus seem to be similar enough so that much of their information can be modeled by a single, appropriately conditioned network.

Compared to single-task fine-tuning of all parameters, our methods on average improve the results by 0.45 for $T5_{SMALL}$ and 1.81 for $T5_{BASE}$ with substantial improvement on low-resource datasets like CoLA (63.73 versus 54.85) and RTE (75.36 versus 67.39) due to shared hypernetworks that capture the shared information and enable positive transfer effects.

We also report the total number of parameters and trainable parameters for all methods in Table 1. For adapter-based methods, the number of parameters varies based on the adapter size (we report all numbers with $r = 32$). The multiple in terms of the number of parameters of $HYPERFORMER{+}{+}_{BASE}$ with regard to $T5_{BASE}$ is 1.02× with only 0.29% trainable parameters per task. Note that by keeping the output layer frozen for $Adapters_{SMALL}$ and $Adapters_{BASE}$, they require 5.51× and 2.53× fewer parameters respectively compared to a direct application of prior work (Houlsby et al., 2019). Despite using more efficient baselines, compared to $Adapters_{BASE}$, $HYPERFORMER{+}{+}_{BASE}$ requires 3× fewer trainable parameters.

3.2 Few-shot Domain Transfer

Finally, we assess how well a trained HYPERFORMER can generalize to new tasks. We evaluate performance on 5 tasks and 7 datasets. In particular, we consider 1) the natural language inference (NLI) datasets SciTail (Khot et al., 2018), and CB (De Marneffe et al., 2019) from SuperGLUE (Wang et al., 2019a) 2) the question answering (QA) dataset BoolQ (Clark

et al., 2019a); 3) the sentiment analysis datasets IMDB (Maas et al., 2011) and Yelp Polarity (Zhang et al., 2015); and 4) the paraphrase detection dataset PAWS (Baldridge et al., 2019); 5) the question classification dataset TREC (Li and Roth, 2002).

For CB and BoolQ, since test sets are not available, we divide the validation sets in half, using one half for validation and the other for testing. For Yelp polarity, TREC, and IMDB, since validation sets are not available, we similarly divide the test sets to form validation sets. For the rest, we report on the original test sets.

We consider the models trained on GLUE reported in Table 1 and evaluate them on the test set after the few-shot fine-tuning on each target training data. For Adapters† and our method, we use the adapter and the task embedding respectively trained on the most similar GLUE task for initialization, i.e. MNLI for NLI, QNLI for QA, SST-2 for sentiment analysis, and QQP for paraphrase detection. Following prior evidence of positive transfer from NLI to other tasks (Conneau and Kiela, 2018; Yin et al., 2020; Phang et al., 2018), we initialize the out-of-domain TREC from MNLI. We show the results of full fine-tuning of all model's parameters, Adapters†, and HYPERFORMER++[4] in Table 2. Our method significantly surpasses the baselines on the majority of settings.

3.3 Low-resource Fine-tuning

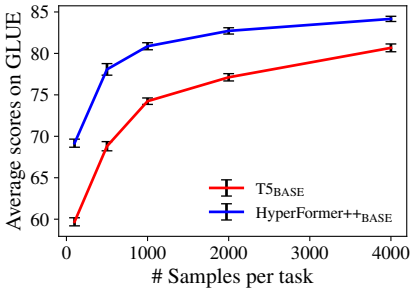

Figure 2: Results on GLUE for the various number of training samples per task (100,500,1000,2000,4000). We show mean and standard deviation across 5 seeds.

Given that our model HYPERFORMER++$_{\text{BASE}}$ has substantially fewer trainable parameters than T5$_{\text{BASE}}$, we investigate whether it generalizes better in a low-resource setting. We subsample each individual task in GLUE for varying training sizes. We train the models for 15,000 steps, which we found to be

[4]We finetune hypernetworks and task embeddings parameters. We also tried only fine-tuning the task embedding but found that this achieves lower performance in the few-shot setting and comparable performance with more samples.

Dataset	# Samples	T5$_{\text{BASE}}$	Adapters†$_{\text{BASE}}$	HYPERFORMER++$_{\text{BASE}}$
Natural Language Inference				
SciTail	4	79.60±3.3	79.54±2.8	**82.00**±4.9
	16	80.03±2.3	83.25±1.7	**86.55**±1.4
	32	81.97±1.3	85.06±1.1	**85.85**±1.4
	100	84.04±0.7	88.22±1.3	**88.52**±0.7
	500	88.07±0.7	91.27±0.8	**91.44**±0.6
	1000	88.77±1.0	91.75±0.8	**92.34**±0.5
	2000	91.01±1.0	92.72±0.5	**93.40**±0.2
CB	4	57.78±10.9	51.11±9.2	**60.74**±16.66
	16	**77.04**±7.2	74.81±5.4	76.29±4.45
	32	80.0±7.6	74.81±5.9	**81.48**±6.2
	100	85.93±5.4	80.74±7.6	**87.41**±2.96
	250	85.19±4.7	86.67±5.0	**89.63**±4.32
Question Classification				
TREC	4	28.11±5.9	23.61±7.7	**28.85**±6.9
	16	40.08±12.6	43.45±14.0	**49.40**±9.5
	32	62.49±6.2	59.6±7.0	**68.94**±7.5
	100	87.79±0.7	78.07±3.8	**88.42**±1.7
	500	93.57±1.3	93.65±1.7	**94.78**±1.4
	1000	95.5±0.9	96.06±0.4	**96.72**±1.3
	2000	96.87±1.3	**97.03**±0.7	96.92±0.9
Question Answering				
BoolQ	4	50.49±11.1	**53.48**±2.8	48.03±4.8
	16	**56.50**±7.1	51.37±6.5	50.21±7.9
	32	**58.43**±4.9	54.52±5.1	58.37±3.7
	100	60.10±2.4	58.60±1.6	**62.03**±2.0
	500	66.49±1.2	66.72±0.7	**70.04**±1.4
	1000	69.01±1.1	70.21±1.3	**72.35**±1.7
	2000	71.58±0.8	73.60±0.8	**74.94**±0.6
Sentiment Analysis				
IMDB	4	77.23±3.0	81.55±1.9	**81.77**±1.8
	16	82.74±1.7	82.54 ±1.0	**84.06**±0.7
	32	83.42±1.0	83.39 ±0.8	**84.64**±0.4
	100	84.58±0.6	83.35 ±0.8	**84.74**±0.4
	500	84.99±0.3	85.37±0.5	**86.00**±0.2
	1000	85.50±0.1	86.27±0.4	**86.37** ±0.4
	2000	86.01±0.2	86.57±0.2	**86.60**±0.1
Yelp polarity	4	76.85±14.3	81.37±13.1	**90.25**±1.0
	16	87.84±1.5	**91.08**±0.2	90.36±1.2
	32	89.22±0.7	91.09±0.5	**91.15**±0.5
	100	90.19±0.7	90.15±0.7	**91.06**±0.6
	500	90.92±0.2	91.52±0.2	**92.09**±0.4
	1000	91.32±0.2	92.26±0.6	**92.50**±0.2
	2000	91.68±0.1	92.36±0.4	**92.70**±0.1
Paraphrase Detection				
PAWS	4	53.89±3.6	**55.69**±9.0	55.58±7.5
	16	54.18±1.0	63.38±5.3	**72.71**±1.1
	32	55.23±3.2	68.78±1.5	**73.39**±2.1
	100	71.51±2.4	73.82±1.6	**78.24**±2.1
	500	82.81±1.0	85.36±0.6	**86.3**±1.1
	1000	85.67±0.7	87.89±0.6	**89.12**±0.5
	2000	88.33±0.6	90.41±0.6	**90.87**±0.3

Table 2: Few-shot domain transfer results of the models trained on GLUE averaged across 5 seeds. We compute accuracy for all datasets.

sufficient to allow them to converge. Figure 2 shows the results. HYPERFORMER++$_{\text{BASE}}$ substantially improves results with limited training data, indicating more effective fine-tuning in this regime.

4 Analysis

4.1 Parameter Efficiency

In this section, we compare the number of parameters of HYPERFORMER++ with Adapters.

Adapters parameters: The standard setting (Houlsby et al., 2019) employs two adapters per layer for each task. Each adapter layer has $2hd$ parameters for projection matrices ($\boldsymbol{U}_{\tau}^{l}$ and $\boldsymbol{D}_{\tau}^{l}$) and $2h$ parameters for the layer normalization. The total number of parameters for Adapters for L Transformer layers in both an encoder and a decoder across T tasks is, therefore, $4TL(2hd+2h)$, which scales linearly with the number of tasks times the number of layers.

HYPERFORMER++ parameters: Our approach learns a task feature embedding per task, consisting of Tt parameters. We additionally employ layer id and adapter position embeddings in the encoder and decoder, which require $2(2+L)t$ parameters, with a fixed embedding size of t for all these feature embeddings. We consider a separate task projector networks h'_I for encoder and decoder, which is in both cases a two-layer MLP, consisting of a total of $2(3te+et)$ parameters, where $e=128$ is the hidden dimension for the task-projector network. Our hypernetwork for adapters in encoder/decoder consists of $2(2thd)$ parameters and our layer normalization hypernetwork consists of $2(2th)$ parameters. In total, this results in $\underbrace{t(T+4+2L)}_{\text{Task features}} + \underbrace{8te+2t(2hd+2h)}_{\text{Hypernetworks}}$ parameters. The total number of parameters for hypernetworks remains constant, while the task feature parameters scale with the number of tasks or layers times t, where $t=64$ in our experiments.

In settings with a large number of layers and a large number of tasks, since $t \ll 2hd+2h$ and $T+L \ll TL$, our method is much more parameter-efficient compared to Adapters. In the current setting, the term hd is the largest term, and the factor $2TL$ for Adapters is larger than the factor t for HYPERFORMER++.

4.2 Do Extra Parameters Make a Difference?

While our HYPERFORMER++ is more parameter-efficient than the baselines, the number of parameters of HYPERFORMER per task is higher compared to Adapters†. To confirm that the improvements of HYPERFORMER are due to its capability of sharing information across tasks and not the number of parameters, as an ablation, we run the Adapters† with $r = \{2, 4\}$ and choose the model performing the best on the validation set. This allows Adapters† to have a higher number of parameters compared to HYPERFORMER. We report the results in Table 3 and compare them with results of HYPERFORMER in Table 1. The results demonstrate that even with an increased number of parameters, Adapters† is not able to reach the performance of HYPERFORMER, and HYPERFORMER performs substantially better.

Model	GLUE	#Total params	#Trained params/task
Adapters† $_{\text{SMALL}}$	80.97	1.83x	10.44%
HYPERFORMER $_{\text{SMALL}}$	82.47	1.45x	5.80 %
Adapters† $_{\text{BASE}}$	85.84	2.02x	12.73%
HYPERFORMER $_{\text{BASE}}$	86.58	1.54x	6.86%

Table 3: Averaged test results on GLUE for HYPERFORMER and Adapters†, where Adapters† has a higher number of parameters compared to HYPERFORMER.

Model variant	GLUE
HYPERFORMER$_{\text{SMALL}}$	82.47
– Adapter blocks	68.37
– Conditional layer norm	79.83
– Task projector	81.56
– T5 Layer norm	81.29
– Conditional layer norm, T5 Layer norm	78.92

Table 4: Impact when removing different components of our framework. We report the average results on GLUE.

4.3 Impact of the Framework Components

We investigate the impact of the components of our framework including: (1) task conditional adapter blocks; (2) task conditional layer normalization; (3) task projection network; (4) fine-tuning of layer normalizations in the T5 model; (5) task conditional layer normalization in adapter modules and fine-tuning of layer normalizations inside the T5 model. We consider our small model of Table 1 and train different variants of it. Table 4 shows the results on GLUE, demonstrating that each component of the model contributes positively to its final performance.

4.4 Visualization of Task Embeddings

To analyze what HYPERFORMER++$_{\text{BASE}}$ has learned about the relations between different tasks, we visualize the learned task embeddings for the models trained

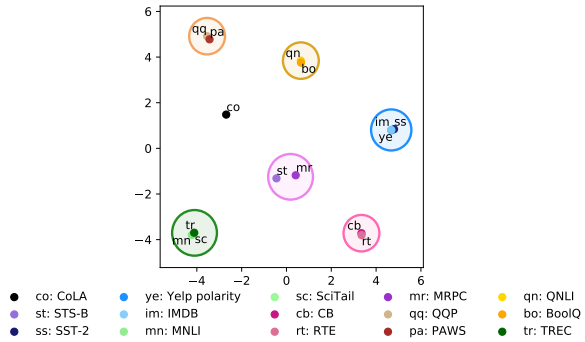

Figure 3: Visualization of learned task embeddings by HYPERFORMER++$_{\text{BASE}}$.

with the largest number of samples in Table 1 and 2. Figure 3 illustrates the 2D vector projections of task embeddings using PCA (Wold et al., 1987). Interestingly, the observed groupings correspond to similar tasks. This shows that learned task embeddings by HYPERFORMER++$_{\text{BASE}}$ are meaningful. For CB, an NLI dataset despite being initialized from MNLI, after few-shot training the task embedding is closest to RTE, another NLI dataset. This is plausible as premises and hypotheses in both the discourse-based CB and the news and Wikipedia-based RTE are more complex compared to MNLI. The sentence similarity dataset STS-B is grouped close to the MRPC paraphrase dataset. CoLA, which focuses on linguistic acceptability is very different from other tasks and is not grouped with any of the observed task embeddings. In addition, the task embeddings for 1) all the sentiment analysis datasets namely SST-2, Yelp polarity, and IMDB; 2) the two large-scale NLI datasets namely MNLI and SciTail; 3) question answering datasets, i.e. BoolQ and QNLI; and 4) paraphrase datasets namely QQP and PAWS are each grouped together.

5 Related Work

Multi-task learning: Multi-task learning, i.e., learning a unified model to perform well on multiple different tasks, is a challenging problem in NLP. It requires addressing multiple challenges such as catastrophic forgetting, and handling disproportionate task sizes resulting in a model overfitting in low-resource tasks while underfitting in high-resource ones (Arivazhagan et al., 2019). Liu et al. (2019a) proposed Multi-Task Deep Neural Network (MTDNN) for learning from multiple NLU tasks. Although MTDNN obtains impressive results on GLUE, it applies multi-task learning as a form of pretraining followed by task-specific fine-tuning. Concurrently with us, Tay et al. (2021) propose a multi-task learning method by training task-conditioned hyper networks; however, their method is 43x less parameter efficient compared to ours. In another line of research, Clark et al. (2019b) proposed to learn multi-task models with knowledge distillation. Houlsby et al. (2019) trained adapters for each task separately, keeping the model fixed. Stickland and Murray (2019) share the model parameters across tasks and introduce task-specific adapter parameters, which is more parameter-inefficient than our method.

Hypernetworks and contextual parameter generation: Our work is closely related to hypernetworks (Ha et al., 2017). In a continual learning setup, where tasks are learned sequentially, Oswald et al. (2020) proposed a task-conditioned hypernetwork to generate all the weights of the target model. Our method is substantially more efficient as we do not generate all the weights of the target model but a very small number of parameters for adapter modules to allow the model to adapt to each individual task efficiently. Similarly, Jin et al. (2020) generate the full model from task-specific descriptions in different domains whereas we efficiently generate only small adapter modules for each task.

Prior work also proposed meta-learning or Bayesian approaches to generate softmax layer parameters for new settings (Bansal et al., 2020; Ponti et al., 2020). Meta-learning approaches are notoriously slow to train. In addition, generating softmax parameters requires a substantially higher number of parameters, leaves the method unable to adapt the lower layers of the model, and restricts their application to classification tasks.

In contemporaneous work, Üstün et al. (2020) proposed a multilingual dependency parsing method based on adapters and contextual parameter generator networks (Platanios et al., 2018) where they generate adapter parameters conditioned on trained input language embeddings. Their study is limited to multilingual dependency parsing, while our work studies multi-task learning and applies to several tasks thanks to the general sequence-to-sequence nature of our model. Moreover, their number of trainable parameters is 2.88× larger than their base model since they employ a contextual parameter generator in each layer. In contrast, we use a single compact hypernetwork allowing us to efficiently condition on multiple tasks and layers of a transformer model.

6 Conclusion

We propose a parameter-efficient method for multi-task fine-tuning. Our approach is to train shared hypernetworks to generate task-specific adapters conditioned on the task, layer id, and adapter position embeddings. The shared hypernetworks capture the knowledge across tasks and enable positive transfer to low-resource and related tasks, while task-specific layers allow the model to adapt to each individual task. Extensive experiments show that our method obtains strong improvement over multi-task learning on the GLUE benchmark, and substantially improves the in-domain task generalization.

Acknowledgments

We are grateful to Dani Yogatama, Neil Houlsby, and Colin Raffel for feedback on a draft of this paper. We would like to also thank Adam Paszke, Jamie Kiros, and George Dahl for useful comments and discussions.

References

Naveen Arivazhagan, Ankur Bapna, Orhan Firat, Dmitry Lepikhin, Melvin Johnson, Maxim Krikun, Mia Xu Chen, Yuan Cao, George Foster, Colin Cherry, et al. 2019. Massively multilingual neural machine translation in the wild: Findings and challenges. *arXiv preprint arXiv:1907.05019*.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. 2016. Layer normalization. *arXiv preprint arXiv:1607.06450*.

Jason Baldridge, Luheng He, and Yuan Zhang. 2019. Paws: Paraphrase adversaries from word scrambling. In *NAACL*.

Trapit Bansal, Rishikesh Jha, and Andrew McCallum. 2020. Learning to Few-Shot Learn Across Diverse Natural Language Classification Tasks. In *COLING*.

Ankur Bapna and Orhan Firat. 2019. Simple, scalable adaptation for neural machine translation. In *EMNLP*.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. 2019a. Boolq: Exploring the surprising difficulty of natural yes/no questions. In *NAACL*.

Kevin Clark, Minh-Thang Luong, Urvashi Khandelwal, Christopher D Manning, and Quoc Le. 2019b. Bam! born-again multi-task networks for natural language understanding. In *ACL*.

Alexis Conneau and Douwe Kiela. 2018. Senteval: An evaluation toolkit for universal sentence representations. In *LREC*.

Marie-Catherine De Marneffe, Mandy Simons, and Judith Tonhauser. 2019. The commitmentbank: Investigating projection in naturally occurring discourse. In *proceedings of Sinn und Bedeutung*.

Harm De Vries, Florian Strub, Jérémie Mary, Hugo Larochelle, Olivier Pietquin, and Aaron C Courville. 2017. Modulating early visual processing by language. In *NeurIPS*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *NAACL*.

David Ha, Andrew Dai, and Quoc V. Le. 2017. Hypernetworks. In *ICLR*.

Dan Hendrycks and Kevin Gimpel. 2016. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*.

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 *ICML*.

Jeremy Howard and Sebastian Ruder. 2018. Universal Language Model Fine-tuning for Text Classification. In *ACL*.

Tian Jin, Zhun Liu, Shengjia Yan, Alexandre Eichenberger, and Louis-Philippe Morency. 2020. Language to network: Conditional parameter adaptation with natural language descriptions. In *ACL*.

Tushar Khot, Ashish Sabharwal, and Peter Clark. 2018. Scitail: A textual entailment dataset from science question answering. In *AAAI*.

Xin Li and Dan Roth. 2002. Learning question classifiers. In *COLING*.

Xiaodong Liu, Pengcheng He, Weizhu Chen, and Jianfeng Gao. 2019a. Multi-task deep neural networks for natural language understanding. In *ACL*.

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

Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. 2011. Learning word vectors for sentiment analysis. In *ACL*.

Bryan McCann, Nitish Shirish Keskar, Caiming Xiong, and Richard Socher. 2018. The natural language decathlon: Multitask learning as question answering. *arXiv preprint arXiv:1806.08730*.

Johannes Von Oswald, Christian Henning, João Sacramento, and Benjamin F Grewe. 2020. Continual learning with hypernetworks. In *ICLR*.

Ethan Perez, Florian Strub, Harm De Vries, Vincent Dumoulin, and Aaron Courville. 2018. Film: Visual reasoning with a general conditioning layer. In *AAAI*.

Matthew E Peters, Sebastian Ruder, and Noah A Smith. 2019. To tune or not to tune? adapting pretrained representations to diverse tasks. In *RepL4NLP*.

Jonas Pfeiffer, Andreas Rücklé, Clifton Poth, Aishwarya Kamath, Ivan Vulić, Sebastian Ruder, Kyunghyun Cho, and Iryna Gurevych. 2020. AdapterHub: A framework for adapting transformers. In *EMNLP: System Demonstrations*.

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*.

Emmanouil Antonios Platanios, Mrinmaya Sachan, Graham Neubig, and Tom Mitchell. 2018. Contextual parameter generation for universal neural machine translation. In *EMNLP*.

Edoardo M Ponti, Ivan Vulić, Ryan Cotterell, Marinela Parovic, Roi Reichart, and Anna Korhonen. 2020. Parameter space factorization for zero-shot learning across tasks and languages. *arXiv preprint arXiv:2001.11453*.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. Improving language understanding by generative pre-training.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *JMLR*.

Alex Ratner, Braden Hancock, Jared Dunnmon, Roger Goldman, and Christopher Ré. 2018. Snorkel metal: Weak supervision for multi-task learning. In *DEEM workshop*.

Sylvestre-Alvise Rebuffi, Hakan Bilen, and Andrea Vedaldi. 2018. Efficient parametrization of multi-domain deep neural networks. In *CVPR*.

Sebastian Ruder. 2017. An Overview of Multi-Task Learning in Deep Neural Networks. In *arXiv preprint arXiv:1706.05098*.

Asa Cooper Stickland and Iain Murray. 2019. Bert and pals: Projected attention layers for efficient adaptation in multi-task learning. In *ICML*.

Yi Tay, Zhe Zhao, Dara Bahri, Don Metzler, and Da-Cheng Juan. 2021. Hypergrid transformers: Towards a single model for multiple tasks. In *ICLR*.

Ahmet Üstün, Arianna Bisazza, Gosse Bouma, and Gertjan van Noord. 2020. Udapter: Language adaptation for truly universal dependency parsing. In *EMNLP*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *NeurIPS*.

Tu Vu, Tong Wang, Tsendsuren Munkhdalai, Alessandro Sordoni, Adam Trischler, Andrew Mattarella-Micke, Subhransu Maji, and Mohit Iyyer. 2020. Exploring and predicting transferability across NLP tasks. In *EMNLP*.

Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. 2019a. Superglue: A stickier benchmark for general-purpose language understanding systems. In *NeurIPS*.

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

Zirui Wang, Zihang Dai, Barnabás Póczos, and Jaime Carbonell. 2019c. Characterizing and avoiding negative transfer. In *CVPR*.

Alex Warstadt, Amanpreet Singh, and Samuel R. Bowman. 2019. Neural network acceptability judgments. *TACL*.

Svante Wold, Kim Esbensen, and Paul Geladi. 1987. Principal component analysis. *Chemometrics and intelligent laboratory systems*.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi 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 M. Rush. 2020a. Transformers: State-of-the-art natural language processing. In *EMNLP: System Demonstrations*.

Thomas Wolf, Quentin Lhoest, Patrick von Platen, Yacine Jernite, Mariama Drame, Julien Plu, Julien Chaumond, Clement Delangue, Clara Ma, Abhishek Thakur, Suraj Patil, Joe Davison, Teven Le Scao, Victor Sanh, Canwen Xu, Nicolas Patry, Angie McMillan-Major, Simon Brandeis, Sylvain Gugger, François Lagunas, Lysandre Debut, Morgan Funtowicz, Anthony Moi, Sasha Rush, Philipp Schmidd, Pierric Cistac, Victor Muštar, Jeff Boudier, and Anna Tordjmann. 2020b. Datasets. *GitHub. Note: https://github.com/huggingface/datasets*.

Wenpeng Yin, Nazneen Fatema Rajani, Dragomir Radev, Richard Socher, and Caiming Xiong. 2020. Universal natural language processing with limited annotations: Try few-shot textual entailment as a start. In *EMNLP*.

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

Xiang Zhang, Junbo Zhao, and Yann Lecun. 2015. Character-level convolutional networks for text classification. In *NeurIPS*.

A Experimental Details

Computing infrastructure: We run the experiments in Table 1 on 4 GPUs, and the rest of the experiments on 1 GPU on a heterogeneous cluster with Tesla V100, Tesla A100, Tesla P4, and GTX1080ti GPUs.

Hyperparameters: We use a batch size of 64 for T5_{SMALL} and 32 for T5_{BASE} to fit the GPU memory. We set the dimension of the task feature embedding ($\boldsymbol{z_\tau}$) to $t' = 512$, and the dimension of the task embedding ($\boldsymbol{I_\tau}$) to $t = 64$. For low-resource fine-tuning in §3.3, we use reduction factors of $\{16,32,64\}$.

Data pre-processing: We download all datasets from the HuggingFace Datasets library (Wolf et al., 2020b). Following Raffel et al. (2020), we cast all datasets into a sequence-to-sequence format, and recast STS-B as a 21-class classification task by rounding its target scores to their nearest increment of 0.2.

Performance evaluation: Table 5 and 6 present the efficiency evaluation in terms of memory, and time for all the methods measured on the GLUE benchmark. We report the time for 1000 training steps.

Our approach has several attractive properties. Our HYPERFORMER++$_{\text{BASE}}$ approach offers a much better memory usage with low-overhead, while HYPERFORMER$_{\text{BASE}}$ and T5_{BASE} cause substantial memory overhead. In dealing with large-scale transformer models like T5, efficient memory usage is of paramount importance. Second, in terms of training time, our method is much faster than Adapters$^{\dagger}_{\text{BASE}}$. Relative to T5_{BASE}, HYPERFORMER++$_{\text{BASE}}$ increases the training time by 30.49%, while Adapters$^{\dagger}_{\text{BASE}}$ causes the substantial training time overhead of 84.93%.

Model	Memory	Δ%
T5_{BASE}	7.76 (GB)	-
Adapters$^{\dagger}_{\text{BASE}}$	5.95 (GB)	-23.32%
HYPERFORMER$_{\text{BASE}}$	7.60 (GB)	-2.06%
HYPERFORMER++$_{\text{BASE}}$	5.81 (GB)	-25.13

Table 5: The required memory for all methods. Δ% is the relative difference with respect to T5_{BASE}.

Model	Time	Δ%
T5_{BASE}	5.51 (min)	-
Adapters$^{\dagger}_{\text{BASE}}$	10.19 (min)	84.93%
HYPERFORMER$_{\text{BASE}}$	7.92 (min)	43.74%
HYPERFORMER++$_{\text{BASE}}$	7.19 (min)	30.49%

Table 6: Training time for all methods. Δ% is the relative difference with respect to T5_{BASE}.

Impact of adapter's bottleneck size on the performance Similar to (Houlsby et al., 2019), adapter's reduction factor needs to be set per dataset. Table 7 shows the validation performance of HYPERFORMER++ on the GLUE tasks for different adapters' reduction factors. While the pattern may not be always consistent, generally, smaller datasets seem to benefit more from smaller bottleneck size, i.e., less parameters for adapters, while the opposite is the case for larger datasets, which require more modeling capacity.

Model	r	CoLA	SST-2	MRPC	QQP	STS-B	MNLI	QNLI	RTE	Avg
HYPERFORMER++$_{\text{SMALL}}$	8	42.13	98.60	82.76/87.72	90.69/87.55	84.92/84.18	82.3	95.40	78.83	83.19
HYPERFORMER++$_{\text{SMALL}}$	16	42.60	97.8	84.73/89.12	88.99/85.33	85.69/85.12	81.96	93.69	75.91	82.81
HYPERFORMER++$_{\text{SMALL}}$	32	49.90	96.00	83.74/88.50	89.29/85.79	85.99/85.41	81.28	91.79	72.99	82.79
HYPERFORMER++$_{\text{BASE}}$	8	54.86	97.30	88.18/91.55	94.59/92.91	89.77/89.69	85.89	96.10	84.67	87.77
HYPERFORMER++$_{\text{BASE}}$	16	53.83	98.00	88.18/91.61	94.89/93.33	90.12/89.65	85.94	96.50	83.94	87.82
HYPERFORMER++$_{\text{BASE}}$	32	55.58	97.20	89.66/92.42	93.19/91.08	88.96/88.57	85.82	94.19	81.75	87.13

Table 7: Validation performance of HYPERFORMER++ on the GLUE tasks for different reduction factors $r = \{8,16,32\}$. For MNLI, we report accuracy on the matched validation set. For MRPC and QQP, we report accuracy and F1. For STS-B, we report Pearson and Spearman correlation coefficients. For CoLA, we report Matthews correlation. For all other tasks, we report accuracy.