Adaptable Adapters

Nafise Sadat Moosavi[1]*, **Quentin Delfosse**[3], **Kristian Kersting**[2,3], **Iryna Gurevych**[2,4]

[1] Department of Computer Science, The University of Sheffield
[2] Hessian Center for AI (hessian.AI)
[3] AI & Machine Learning Lab, [4] Ubiquitous Knowledge Processing Lab (UKP Lab),
Department of Computer Science, Technical University of Darmstadt
https://www.ukp.tu-darmstadt.de

Abstract

State-of-the-art pretrained NLP models contain a hundred million to trillion parameters. Adapters provide a parameter-efficient alternative for the full finetuning in which we can only finetune lightweight neural network layers on top of pretrained weights. Adapter layers are initialized randomly. However, existing work uses the same adapter architecture—i.e., the same adapter layer on top of each layer of the pretrained model—for every dataset, regardless of the properties of the dataset or the amount of available training data. In this work, we introduce adaptable adapters that contain (1) learning different activation functions for different layers and different input data, and (2) a learnable switch to select and only use the beneficial adapter layers. We show that adaptable adapters achieve on-par performances with the standard adapter architecture while using a considerably smaller number of adapter layers. In addition, we show that the selected adapter architecture by adaptable adapters transfers well across different data settings and similar tasks. We propose to use adaptable adapters for designing efficient and effective adapter architectures. The resulting adapters (a) contain about 50% of the learning parameters of the standard adapter and are therefore more efficient at training and inference, and require less storage space, and (b) achieve considerably higher performances in low-data settings.[1]

1 Introduction

Recent improvements in NLP are heavily skewed towards using larger pretrained models (Roberts et al., 2020) and given their considerably better performances, using them is becoming unavoidable (Kaplan et al., 2020). Their improvements, however, come at the cost of significant computational resources at training and inference times. For instance, the number of parameters in recent pretrained models can vary from 110M in BERT-base (Devlin et al., 2019) to 11 billion in T0 (Sanh et al., 2022) to trillion parameters in Switch Transformers (Fedus et al., 2021). Using such models for each downstream application requires a vast amount of storage, training, and inference computation budget that is not accessible to every user.

Instead of fine-tuning these massive numbers of parameters for each downstream task, we can use adapter architectures (Houlsby et al., 2019; Pfeiffer et al., 2020). Adapters are lightweight neural network layers that are added on top of each layer of the pretrained model. As opposed to the standard model fine-tuning, in which all layers are fine-tuned for the target task, adapter-based tuning freezes the transformer layers and only trains the newly added adapter layers. Since the majority of parameters—i.e., the layers of the large pretrained model—are shared between different downstream tasks, the use of adapters results in parameter-efficient transfer learning. In addition to their parameter-efficiency, He et al. (2021) show that training adapter-layers (a) outperforms fine-tuning the whole model on low-data and cross-lingual settings, and (b) is more robust to overfitting.

Existing work suggests that (a) different layers of the pretrained models may capture different aspects of the form, syntax, or meaning of the input text (Tenney et al., 2019; Clark et al., 2019), and (b) they may not be all needed for performing a given task (Houlsby et al., 2019; Fan et al., 2020; Rücklé et al., 2021). In addition, adapter layers are initialized randomly. Therefore, it is not necessary to use the same adapter architecture for different downstream tasks and given different amounts of annotated data. However, existing works use the same adapter architecture for all the different input data, i.e., (a) one adapter layer on top of all the pretrained layers while using all the layers may not be necessary, and (b) the same activation func-

*The work has been mostly carried out during the employment at the UKP Lab, TU Darmstadt.

[1]The code is available at https://github.com/UKPLab/adaptable-adapters.

Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pages 3742 - 3753

tion for all the layers and different tasks while the best activation function may vary for different tasks (Delfosse et al., 2021).

In this paper, we propose a systematic approach for designing more adequate and flexible adapter architectures by introducing the adaptable adapter (*AA*). Adaptable adapters (1) use a learnable activation function—called Rational activation (Molina et al., 2020)—instead of a constant activation in adapter layers allowing the adapter model to learn different activation functions at different adapter layers and for different tasks, and (2) consist of a learnable switch at each adapter layer to determine the beneficial adapter layers during training and to only use the selected layers during inference.

We evaluate adaptable adapters on the GLUE benchmark (Wang et al., 2018) that consists of various text classification tasks. We perform evaluations based on different data settings in which different amounts of annotated examples are available for training. Our results show that adaptable adapters achieve on-par performances with the full adapter architecture while using considerably fewer adapter layers at the inference.

We further propose to use adaptable adapters for designing efficient adapter architectures—i.e., to only add an adapter layer to the layers that are selected by the adaptable adapter. We show that while the selected adapter architecture by *AA*, called *AA-focused*, is considerably more efficient at both training and inference times and requires less storage, it achieves on-par performances with the full adapter architecture when trained on all available training data and considerably outperforms it on low-data settings. In addition, we show that the selected adapter architecture by *AA* transfers well across similar tasks and different data settings. Therefore, we can train *AA* using a limited amount of training data, and for one of the tasks, and then use the resulting *AA-focused* architecture for different data settings and other similar tasks.

Overall, the contributions of this paper are as follows:

- We propose adaptable adapters that introduce flexibility in adapter architectures by (a) selecting the beneficial adapter layers to use, and (b) learning the suitable activation function for each layer and each task.
- We propose to use adaptable adapters to design efficient adapters that require less training time, inference time, and storage space.
- We show that using fewer adapter layers with a learnable activation function considerably improves the performance on low-data settings.

2 Related Work

2.1 Rational Activation

Rational activation functions, empirically introduced as Padé Activation Units (Molina et al., 2020), are learnable activation functions that can approximate common activation functions as well as learn new ones. The rational activation function $R(x)$ of order m, n is defined as follows:

$$R(x) = \frac{\sum_{j=0}^{m} a_j x^j}{1 + |\sum_{k=1}^{n} b_k x_k|} \quad (1)$$

where a_j and b_k are learnable parameters. These rational functions use an absolute value in the denominator to avoid potential poles, which will make the training unstable. Such rational activation functions provide stable training, as empirically shown in image classification and reinforcement learning (Molina et al., 2020; Delfosse et al., 2021). $R(x)$ can be initialized to initially approximate any of the known activation functions or with constant functions. Molina et al. (2020) show that rationals outperform other commonly used activation functions in common image classification tasks. Rational activation functions are also integrated in Generative Adversarial Networks (Boullé et al., 2020). Delfosse et al. (2021) show that some of the layers in very deep pretrained Residual Networks tend to approximate activation functions' behavior, and we can achieve on-par or better performances with the full network by replacing some of the complete layers with rational activation functions. Similar to this observation, as we show in § 5, using rational activation functions instead of a constant activation (ReLU) in adapters allows them to achieve high accuracy using a fewer number of adapter layers.

2.2 Reducing Model's Size for Efficiency

Improving the efficiency of large pretrained models has received particular attention for the inference time. The argument is that the effect of training cost is limited, i.e., the model can be trained once but it will be used many times. However, the inference time has a wide impact on the everyday use of NLP models.

Existing approaches for improving the inference-time efficiency belong to two different categories:

(a) the distillation and pruning techniques that create a smaller model for inference but often require re-training or fine-tuning the smaller model (Tang et al., 2019; Sanh et al., 2019; Voita et al., 2019; Sun et al., 2020; Bai et al., 2021), and (b) on-demand network size reduction at the inference time.[2] There are two different approaches in the second category, namely layer dropping and early exiting.

Fan et al. (2020) use layer dropping during the training that randomly drops the model's layers to make the model robust to the inference time layer selection. They show that it is possible to select sub-networks of any depth from large models at inference with limited impact on the performance and without the need for additional finetuning. Layer dropping was previously investigated by Huang et al. (2016) who propose to drop layers during training for regularizing the model and reducing the training time of deep convolutional networks. Rücklé et al. (2021) use layer dropping for adapter architectures. They show that by randomly dropping adapter layers during training, they can prune the adapter model on-demand at the inference time.

Schwartz et al. (2020) propose to add an output layer to each transformer layer. At inference time, while the model calculates the layer-wise representation, from the bottom layer to the top layer, it also makes the prediction using the associated classification layer. They use the output labels' scores of the classification layers as confidence scores to decide whether to exit early if the classifier is confident or to proceed to process the input with the next layers. This hierarchical architecture offers an inference time-accuracy tradeoff by setting the confidence threshold. The early exiting approach is similar to layer dropping in which the dropped layers are always from the last top layers.

All these approaches select the number of layers to drop and the dropped layers heuristically at the inference time with the goal of improving the inference time. Instead, the adaptable adapter is a systematic approach for selecting the useful adapter layers for the given task during training. Besides layer selection, an adaptable adapter allows for learning the desired activation function for different inputs. As we show, we can use adaptable adapters to design efficient adapter architectures with a considerably smaller number of training parameters with on-par or considerably higher performances, especially with larger models and in low-data settings.

[2]There is another category that requires changes in the models' architectures. However, it would require re-training the large model. E.g., Sukhbaatar et al. (2019) propose new attention mechanisms that can process larger context with no additional computational or memory costs.

3 Proposed Architecture

3.1 Learnable Activation

Empirical observations of performances have led experts in several fields to use different activation functions for different tasks. Functions from the ReLU family are usually used for neural network-based visual computing, Tanh has been used in PPO for reinforcement learning, while GeLU has progressively been adopted in transformers. With the growth of the models, and the complexity of the tasks they are applied to, choosing one fixed activation function to equip the complete architecture is suboptimal. By using rational (§ 2.1), we let the adapter layer learn the suitable activation function at each different adapter layer, task, and dataset. In adaptable adapters, we replace the constant activation function of each adapter layer—i.e., ReLU in the default configuration used in AdapterHub (Pfeiffer et al., 2020)—with rational.

Figure 1 shows a standard adapter layer as well as an adapter layer in adaptable adapters.

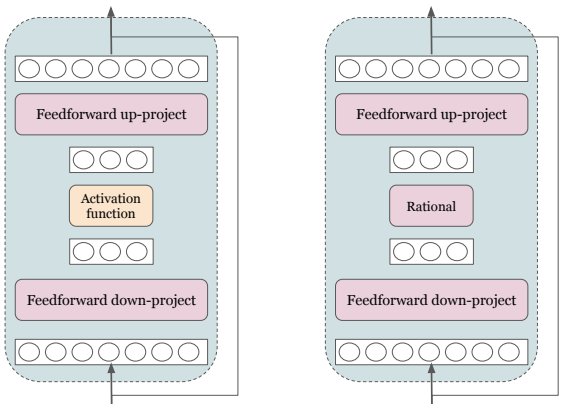

Figure 1: (a) a standard adapter layer with linear feedforward layers and a fixed activation, (b) an adapter layer in adaptable adapters with linear feedforward layers and a rational activation. Learnable parameters are shown within pink boxes.

3.2 Learnable Layer Selection

Houlsby et al. (2019) examined various choices of adapter architectures. They report that using

two feedforward linear layers—one down-project and one up-project layer—results in good performances while only introducing a few parameters. Assuming d is the dimensionality of the input—i.e., the embedding size of the transformer layer—the down-project layer maps the input dimension to n where $n < d$, and the up-project layer maps the input dimension back to d. n is called the hidden size of the adapter. Each adapter contains a skip-connection that lets an adapter layer approximate an identity function, i.e., to pass the input of a transformer layer unchanged to the next layer. The learnable switches in adaptable adapter explicitly model the selection between the feedforward adapter layer and the identity function. By examining the switch probabilities we can determine the adapter layers that are beneficial for the overall performance of the model.

As mentioned in § 1, existing work shows that different layers of the pretrained models capture different aspects of the input data, and not all of them are necessary for performing various tasks. Therefore, for different input data, different layers may be of different importance. Adding a learnable switch at each adapter layer provides a more systematic approach to determining the beneficial layers for each input task during training. We use the Gumbel Softmax ($\mathcal{GS}$) estimator as an end-to-end differentiable switch (hard attention) to make the network attend to an element of a set. Assuming π_i are the probabilities of selecting each element of the set, i.e., $\forall_i \pi_i \geq 0, \sum_i \pi_i = 1$, $\mathcal{GS}$ estimates the hard attention y_i as follows:

$$y_i = \frac{exp((log(\pi_i) + g_i)/\tau)}{\sum_j exp((log(\pi_j) + g_j)/\tau)} \quad (2)$$

where g_i are i.i.d. samples from a Gumbel distribution, and τ is a temperature parameter. Setting τ to small values results in distributions that are similar to categorical ones.

3.3 Adaptable Adapters

The adaptable adapter (*AA*) is the combination of the learnable layer selection and the learnable activation function. The learnable layer selection—i.e., a Gumbel Softmax estimator—selects between an adapter layer, with no skip connection, and an identity function with zero parameters that passes the input without any changes to the next layer. The adapter layers in adaptable adapters consist of two linear layers—i.e., down-project and up-project layers—, and the non-linearity function between these two linear layers consists of a rational activation function. The adaptable adapter allows to learn different adapter architectures for different input data by (a) learning to use a subset of adapter layers, and (b) learning a potentially different activation function at each layer. Figure 3 shows the structure of an adapter layer in adaptable adapters.

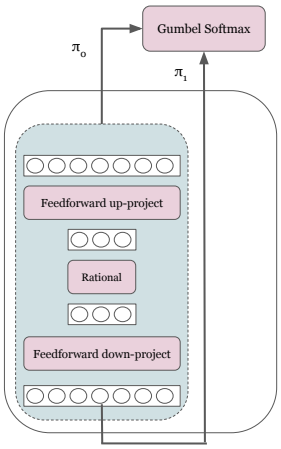

Figure 2: The adaptable adapter layer that consist of a Gumbel Softmax to choose between an adapter layer with a rational activation and an identity function.

4 Experimental Setup

4.1 Datasets

We use the English text classification datasets from the GLUE benchmark (Wang et al., 2019) including MNLI (Williams et al., 2018), QQP[3], QNLI (Rajpurkar et al., 2016), SST-2 (Socher et al., 2013), CoLA (Warstadt et al., 2019), STS-B (Cer et al., 2017), MRPC (Dolan and Brockett, 2005), RTE (Dagan et al., 2006), and WNLI (Levesque et al., 2011). Table 1 shows the number of training examples and the evaluation metric for each dataset.

Dataset	|Train|	Metric	Dataset	|Train|	Metric
MNLI	393k	acc.	STS-B	7k	Pearson/Spearman
QQP	364k	acc./F1	MRPC	3.7k	acc./F1
QNLI	105k	acc.	RTE	2.5k	acc.
SST-2	67k	acc.	WNLI	634	acc.
CoLA	8.5k	Matthews			

Table 1: GLUE datasets with their number of training examples and the corresponding evaluation metric.

[3] https://www.quora.com/profile/Ricky-Riche-2/First-Quora-Dataset-Release-Question-Pairs

4.2 Transformer Model

As the base model, we use the BERT-large model (Devlin et al., 2019). BERT-large contains 24 layers, an embedding size of 1024, and a total number of 340M parameters.[4]

4.3 Adapter Models

Baseline As a baseline adapter, we use the adapter layers with the pfeiffer configuration from AdapterHub (Pfeiffer et al., 2020). The adapter layers with the pfeiffer configuration are similar to the one in Figure 1, in which learnable parameters include two feedforward layers. For BERT-base, each pfeiffer layer consists of 73.7k parameters[5] resulting in a total number of 884.7K. For BERT-large, the number of parameters for each adapter layer is 131K, and the total number of parameters is 3.1M. We see that as the underlying model gets larger, the number of parameters in adapters also increases notably. Therefore, adapter architecture selection using *AA* is a potential solution to control this exponential increase to some extent.

Adaptable Adapter (AA) For the rational activation, similar to Molina et al. (2020), we use order $m = 5$ and $n = 4$ for rational. Therefore, the rational activation function only consists of ten learnable parameters. The rational activation can be initialized to initially estimate an existing function. Based on our preliminary experiments, using $f(x) = 1$ for initializing $R(x)$ results in better performances on the GLUE benchmark.

For the Gumble-Softmax switch, we set the temperature parameter τ to 0.1, and we initialize π_i to 0.5 for both inputs—i.e., the same initial probability for the rational adapter and the identity function.

AA-focused We can use the selected architecture by *AA* for designing a new adapter architecture, i.e., to only include an adapter layer—with a rational function—at layers in which the switch has selected the adapter layer over the identity function. We call this architecture *AA-focused*. Note that compared to *AA*, *AA-focused* is more efficient both at training and inference time, as it includes a fewer number of layers and no switch functions. It also requires less storage space for saving the new adapter weights. Also, training *AA* includes both the architecture selection and training of the adapter layers, which are initialized randomly, simultaneously. As a result, as we see in our evaluations, *AA-focused* achieves higher performances as its training is only focused on training the adapter layers.

AdapterDrop (Rücklé et al., 2021) During training, AdapterDrop randomly drops the first n layers in which n varies for different iterations. At inference, n can be set to any desired number of layers. In our experiments, we select n based on the number of dropped layers by *AA*, i.e., the number of layers that are not selected by the switch functions.

4.4 Experiments

We evaluate the models in different settings: (a) using full training data, and (b) low-data settings. For all the experiments, we consider 25% of the training data as the development set and use the official development sets as the test data. We perform the low-data evaluations when 100, 300, and 500 annotated examples are available.[6] The test data is the same for all the evaluations. We run all the low-data experiments for 20 epochs and five different random seeds[7]. We report the average and standard deviation over the five different runs. When training on full datasets, the experiments are computationally very expensive using BERT-large. Therefore, for this setting, we only report the results using the first random seed. All experiments are done on one A100 NVIDIA GPU. All implementations are based on AdapterHub (Pfeiffer et al., 2020).

5 Evaluation

Table 2 presents the results of *Baseline*, *AdapterDrop*, *AA*, and *AA-focused*. *AA* selects different layers for different tasks and different random seeds.[8] We evaluate three configurations for *AA-focused*:

- **AA-focused**^{spec}: for each task, we design the corresponding *AA-focused* based on the selected architecture by *AA* for that task given and the first random seed (42). For instance, the *AA-focused* architecture is the same for all

[4]The results for BERT-base are reported in the appendix. BERT-base contains 12 layers, an embedding size of 768, and 110M parameters.

[5]The reduction factor in the down-project layer is 16 which results in (768/16) x 768 x 2 parameters for each adapter layer.

[6]Selected training examples for low-data experiments are the same for all models given the same random seed.

[7]42, 92, 111, 245, and 651.

[8]For instance, the selected layers for *RTE* are as follows for different runs of *Low-data-100*: {0, 2, 5, 11, 12, 13, 16, 17}, {3, 4, 5, 6, 7, 8, 9, 10, 12, 13, 15, 19, 21}, {2, 3, 4, 6, 9, 12, 14, 16, 17, 18, 20, 22, 23}, {0, 2, 6, 8, 9, 11, 13, 14, 17, 19, 23}, {1, 2, 5, 10, 11, 14, 16, 20, 21, 22, 23}.

	MNLI	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE	WNLI	Avg
					Low-data-100					
Baseline	$33.89_{3.02}$	$30.65_{0.38}$	$58.78_{4.81}$	$56.01_{3.68}$	$5.20_{4.84}$	$40.00_{9.64}$	$74.80_{0.0}$	$49.39_{2.86}$	$55.21_{3.01}$	44.87
AA	$33.64_{2.66}$	$30.88_{0.39}$	$59.61_{6.19}$	$51.28_{2.52}$	$-0.55_{1.87}$	$45.18_{14.17}$	$74.80_{0.0}$	$50.11_{3.44}$	$55.48_{2.46}$	44.49
AdapterDrop^{AA}	$33.72_{2.84}$	$30.62_{0.40}$	$57.50_{5.78}$	$54.01_{2.59}$	$4.10_{7.95}$	$36.53_{8.93}$	$74.8_{0.0}$	$49.39_{2.86}$	$\mathbf{56.06}_{1.38}$	44.08
AdapterDrop^{13}	$33.71_{2.76}$	$30.61_{0.4}$	$58.39_{4.27}$	$53.44_{2.56}$	$3.91_{7.6}$	$36.23_{8.68}$	$74.8_{0.0}$	$49.46_{2.81}$	$55.76_{1.91}$	44.04
AA-focused^{spec}	$35.28_{2.06}$	$44.37_{16.31}$	$\mathbf{63.75}_{4.39}$	$52.94_{4.64}$	$5.68_{10.91}$	$\mathbf{62.79}_{3.34}$	$74.80_{0.01}$	$51.48_{2.72}$	$54.08_{4.51}$	49.47
AA-focused^{uni}	$\mathbf{36.36}_{2.61}$	$44.37_{16.31}$	$63.36_{4.86}$	$55.87_{4.42}$	$4.75_{4.9}$	$59.37_{6.78}$	$\mathbf{74.94}_{0.2}$	$51.12_{3.45}$	$51.83_{4.12}$	49.11
AA-focused^{sim}	$34.77_{3.18}$	$\mathbf{45.78}_{14.40}$	$63.13_{4.30}$	$\mathbf{61.58}_{10.95}$	$\mathbf{17.54}_{11.19}$	$59.89_{7.70}$	$74.77_{0.07}$	$\mathbf{52.20}_{2.93}$	$51.83_{5.52}$	**51.28**
|Baseline|	24	24	24	24	24	24	24	24	24	
|AA|	$13.2_{1.7}$	$15.0_{3.0}$	$13.6_{2.2}$	$14.6_{4.0}$	$15.8_{2.1}$	$16.4_{2.7}$	$13.0_{1.8}$	$11.2_{1.8}$	$12.3_{5.7}$	
|AdapterDrop^{AA}**|**	14	13	15	16	16	14	15	13	16	
|AdapterDrop^{13}**|**	13	13	13	13	13	13	13	13	13	
|AA-focused^{spec}**|**	14	13	15	16	16	14	15	13	16	
|AA-focused^{uni}**|**	13	13	13	13	13	13	13	13	13	
|AA-focused^{sim}**|**	13	13	13	13	13	13	13	13	13	
					Low-data-300					
Baseline	$36.55_{4.76}$	$61.50_{8.66}$	$69.62_{1.24}$	$79.86_{14.15}$	$30.40_{5.48}$	$78.24_{2.81}$	$76.55_{1.31}$	$51.62_{3.21}$	$45.92_{4.33}$	58.91
AA	$37.14_{2.49}$	$66.07_{0.38}$	$71.33_{1.82}$	$72.52_{16.59}$	$26.05_{8.74}$	$\mathbf{82.08}_{1.6}$	$74.03_{2.21}$	$51.83_{2.84}$	$47.04_{4.76}$	58.68
AdapterDrop^{AA}	$38.86_{5.93}$	$62.98_{4.85}$	$66.71_{2.91}$	$79.29_{14.17}$	$16.89_{12.06}$	$78.5_{1.99}$	$75.74_{0.67}$	$51.19_{3.35}$	$46.76_{4.02}$	57.44
AdapterDrop^{13}	$37.95_{5.56}$	$63.72_{4.84}$	$66.71_{2.91}$	$80.0_{14.47}$	$16.3_{12.05}$	$77.52_{2.08}$	$76.03_{0.92}$	$51.33_{3.4}$	$46.48_{4.27}$	57.34
AA-focused^{spec}	$44.62_{4.11}$	$66.83_{1.06}$	$73.72_{1.09}$	$85.87_{2.94}$	$34.51_{8.3}$	$81.16_{2.04}$	$76.72_{1.06}$	$54.58_{4.72}$	$46.20_{3.92}$	62.69
AA-focused^{uni}	$\mathbf{46.69}_{4.29}$	$\mathbf{69.25}_{1.33}$	$\mathbf{74.16}_{2.95}$	$\mathbf{87.57}_{0.72}$	$35.65_{3.26}$	$81.71_{2.64}$	$75.97_{1.55}$	$\mathbf{56.89}_{5.56}$	$\mathbf{52.39}_{7.26}$	**64.48**
AA-focused^{sim}	$45.97_{2.08}$	$68.36_{1.36}$	$73.98_{2.68}$	$86.83_{1.90}$	$\mathbf{37.43}_{3.10}$	$78.81_{3.58}$	$\mathbf{76.66}_{1.30}$	$55.96_{2.81}$	$48.44_{5.53}$	63.61
|AA|	$17.0_{1.3}$	$16.2_{1.0}$	$14.8_{1.8}$	$12.8_{3.2}$	$16.8_{2.2}$	$18.6_{1.9}$	$16.0_{1.1}$	$12.4_{1.2}$	$12.4_{2.1}$	
|AA-focused^{spec}**|**	18	16	13	9	17	16	16	13	13	
					Low-data-500					
Baseline	$44.35_{6.08}$	$69.49_{1.12}$	$73.48_{1.89}$	$88.26_{1.53}$	$37.98_{4.42}$	$82.07_{0.99}$	$78.33_{1.11}$	$59.28_{1.76}$	$49.86_{6.08}$	64.79
AA	$47.33_{5.11}$	$67.52_{2.99}$	$75.0_{2.3}$	$84.93_{3.06}$	$39.96_{4.87}$	$\mathbf{84.56}_{0.87}$	$78.38_{1.0}$	$59.28_{3.18}$	$50.13_{5.16}$	65.23
AdapterDrop^{AA}	$42.66_{7.02}$	$69.52_{1.03}$	$74.15_{2.19}$	$\mathbf{89.01}_{0.49}$	$38.44_{4.51}$	$82.05_{1.05}$	$78.19_{1.04}$	$59.28_{2.6}$	$49.3_{6.36}$	64.73
AdapterDrop^{13}	$43.05_{6.41}$	$69.12_{0.88}$	$72.82_{1.83}$	$88.97_{0.6}$	$36.89_{5.03}$	$80.77_{1.32}$	$77.86_{0.8}$	$58.56_{2.44}$	$49.01_{6.57}$	64.12
AA-focused^{spec}	$54.96_{2.66}$	$69.52_{1.14}$	$\mathbf{77.30}_{1.27}$	$87.94_{1.10}$	$39.51_{3.47}$	$84.30_{0.69}$	$\mathbf{78.92}_{1.70}$	$59.20_{2.58}$	$48.73_{6.27}$	66.71
AA-focused^{uni}	$\mathbf{56.13}_{1.88}$	$69.32_{2.29}$	$76.85_{2.37}$	$87.89_{1.47}$	$\mathbf{41.75}_{3.83}$	$83.48_{1.25}$	$78.00_{0.35}$	$\mathbf{60.42}_{1.75}$	$\mathbf{50.42}_{5.07}$	**67.14**
AA-focused^{sim}	$55.85_{2.62}$	$\mathbf{69.86}_{2.56}$	$\mathbf{77.30}_{1.93}$	$87.57_{1.69}$	$39.79_{1.42}$	$83.23_{1.61}$	$78.75_{1.26}$	$60.07_{1.62}$	$49.58_{6.75}$	66.89
|AA|	$12.8_{6.0}$	$16.8_{1.3}$	$16.4_{2.6}$	$14.6_{2.1}$	$10.6_{8.3}$	$19.6_{1.4}$	$16.6_{2.4}$	$14.3_{6.8}$	$12.6_{3.2}$	
|AA-focused^{spec}**|**	14	17	18	15	17	18	14	16	14	
					Full Data					
Baseline	85.08	88.68	**91.95**	93.00	58.28	89.75	83.12	**70.39**	56.34	**79.62**
AA	84.73	88.38	91.01	92.55	57.60	**90.11**	82.36	63.18	53.52	78.16
AdapterDrop^{AA}	84.96	**88.75**	91.38	**93.35**	**58.63**	89.85	82.84	66.06	56.34	79.12
AdapterDrop^{13}	84.73	87.15	90.92	92.78	57.42	88.84	83.34	64.25	56.34	78.42
AA-focused^{spec}	84.77	88.46	91.38	92.32	56.79	89.74	83.42	64.98	**57.75**	78.84
AA-focused^{uni}	85.41	88.61	91.51	92.66	54.62	89.34	**84.88**	67.15	56.34	78.94
AA-focused^{sim}	**85.32**	88.41	91.85	91.4	57.96	89.38	84.42	67.86	**57.75**	79.37
|AA|	14	18	17	18	20	20	18	16	15	
|AA-focused^{spec}**|**	14	18	17	18	20	20	18	16	15	

Table 2: Comparing the results of (a) the standard adapter model that includes an adapter layer on all the 24 BERT-large layers (*Baseline*), (b) adaptable adapter (*AA*), (c) *AdapterDrop*, and (d) *AA-focused* adapters, in which the architecture of the adapter is selected based on the selected layers by *AA*. The architecture of *AA-focused*^{spec} is selected based on the selected layers by *AA* for the corresponding task and data setting when the random seed is 42. The architecture of *AA-focused*^{uni} is selected based on the selected layers by *AA* for the task of *QQP* on the *Low-data-100* setting and for random seed 42. *AA-focused*^{sim} only contains an adapter layer with a rational activation function at the last 13 layers of BERT-large, i.e., the total number of adapter layers in *AA-focused*^{uni}. The number of layers at the inference time for the *AdapterDrop*^{AA} experiments are selected based on the number of layers in the corresponding *AA-focused*^{spec} experiments. The number of inference time layers for *AdapterDrop*^{13} equals 13. Except for *Full Data*, the reported results are averaged over five random seeds. The subscript reports the corresponding standard deviation. The *Full Data* results are reported for one random seed. The |*AA*| rows report the average number of selected adapter layers by *AA* using different random seeds. |*AA-focused**| rows report the number of added adapter layers in the corresponding |*AA-focused**| experiments. |*AA-focused*^{uni}| and |*AA-focused*^{sim}| are the same for all data settings. |*AdapterDrop**| rows report the number of included adapter layers for the corresponding *AdapterDrop* experiment at the inference time. |*AdapterDrop*^{AA}| is always the same as the corresponding |*AA-focused*^{spec}|, and |*AdapterDrop*^{13}| is always the same as *AA-focused*^{sim}. The test data is the same for all the experiments. The *Avg* column reports the average score across all datasets. The highest performances for each dataset and each data setting are boldfaced.

the experiments of *RTE* for *Low-data-100*—i.e., over the five different random seeds—. However, it is different for the rest of the tasks and different data settings.

- **AA-focused**^{uni}: we design this adapter architecture of all tasks and data settings based on a single random seed, single task, and a single data regime, i.e.— random seed 42, the *QQP* task, and *low-data-100*. We choose *low-data-100* because the architecture selection process—i.e., training *AA*—is very fast in this setting. We select the selected architecture by *QQP* because *AA* selects the smallest number of layers for *QQP* when the random seed is 42. The selected layers are {2, 6, 10, 12, 14, 15, 16, 18, 19, 20, 21, 22, 23}, i.e., three layers from the first half of the original 24 layers, and ten layers from the second half. The results of *AA-focused*^{uni} compared to AA-focused^{spec} indicate whether the selected architecture by *AA* transfers between similar tasks and different data settings.

- **AA-focused**^{sim}: we design a simplified adapter based on *AA* in which we only use the number of selected layers, instead of the layer numbers, in a single random seed, single task, and a single data setting. We use the number of selected layers when the random seed is 42 for the *QQP* task and the *low-data-100* setting, i.e., 13. As investigated by Houlsby et al. (2019), the last adapter layers are in general more effective. As a result, we add adapter layers, with rational activation, to the last 13 transformer layers in *AA-focused*^{sim} experiments. The results of *AA-focused*^{sim} compared to AA-focused^{uni} show whether only the number of selected layers by *AA* matters or it is also important to specify at which layers to add the adapters.

The number of inference layers for *AdapterDrop*^{AA} are equivalent to the number of layers in *AA-focused*^{spec} experiments for each task and data setting. The number of layers for *AdapterDrop*^{13} is 13, which is the same as *AA-focused*^{uni} and *AA-focused*^{sim}. Note that the number of layers for *AA-focused* experiments are the same both at training and inference while it is not the case for *AdapterDrop*.

The |*AA*| rows in Table 2 show the average number of selected layers for each task over the five different random seeds. |*AA-focused**| rows report the number of added adapter layers in the corresponding *AA-focused** experiments. |*AdapterDrop**| rows report the number of included adapter layers for the corresponding *AdapterDrop* experiments at the inference time.

We make the following observations from the results of Table 2:

- *AA* achieves on-par performances with the *Baseline*, and on average it uses about 13-15 layers out of 24 layers. We can use this insight for designing efficient adapter architectures.

- All *AA-focused* architectures considerably outperform *Baseline* in all the the tasks in low-data settings while using considerably smaller number of parameters, and therefore, being considerably more efficient. For instance, while *AA-focused*^{uni} only uses 13 layers out of 24 layers—i.e., reducing the number of training parameters from 3M to 1.7M—, it outperforms the *Avg* score by 4.24, 5.57, and 2.35 points in *Low-data-100*, *Low-data-300*, and *Low-data-500*, respectively.

- The high performances of *AA-focused*^{uni} show that the selected architecture by *AA* for one task and one data setting transfers well to other data regimes and similar tasks.[9] Therefore, it is not necessary to design the adapter architecture separately for a different amount of available data and similar tasks.

- *AA-focused*^{sim} and *AdapterDrop*^{13} both use the last 13 adapter layers during the inference while the results of *AA-focused*^{sim} are considerably higher for all data regimes. This indicates the importance of rational activation in adaptable adapters. We will further investigate the impact on rational activation in the next section.

- In average, *AdapterDrop*^{AA} contains more inference layers compared to *AdapterDrop*^{13}. However, there is not a significant difference between their performances. They achieve on-par or lower results compared to *Baseline*.

[9] It even outperforms *AA-focused*^{spec} showing that *AA-focused*^{spec} may have overfitted to the development sets. We have not performed hyperparameter selection for our experiments. Using better hyperparameters may improve the results of different settings.

Adap. layers	MNLI	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE	WNLI	Avg
					Low-data-300					
13	$45.97_{2.08}$	$68.36_{1.36}$	$73.98_{2.68}$	$86.83_{1.9}$	$37.43_{3.1}$	$78.81_{3.58}$	$76.66_{1.3}$	$55.96_{2.81}$	$48.44_{5.53}$	63.61
12	$36.84_{5.51}$	$62.43_{5.65}$	$65.77_{3.43}$	$84.63_{3.64}$	$13.23_{12.63}$	$77.08_{2.36}$	$75.27_{0.39}$	$54.30_{3.75}$	$46.76_{5.45}$	57.37
11	$36.16_{5.12}$	$62.59_{5.8}$	$67.93_{1.42}$	$79.95_{14.16}$	$16.32_{11.65}$	$73.22_{6.75}$	$76.42_{1.19}$	$56.53_{2.02}$	$46.2_{4.12}$	57.26

Table 3: Evaluating the impact of the number of adapter layers on the overall performance. The adapter layers are added to the top n layers of the model for $n = 13, 12, 11$. Adapter layers contain rational activation, i.e., $n = 13$ is equivalent to *AA-focused*^{sim}. Results are reported for the *low-data-300* setting.

	MNLI	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE	WNLI	Avg
					Low-data-300					
Baseline	$36.55_{4.76}$	$61.50_{8.66}$	$69.62_{1.24}$	$79.86_{14.15}$	$30.40_{5.48}$	$78.24_{2.81}$	$76.55_{1.31}$	$51.62_{3.21}$	$45.92_{4.33}$	58.91
AA	$37.14_{2.49}$	$66.07_{0.38}$	$71.33_{1.82}$	$72.52_{16.59}$	$26.05_{8.74}$	$\mathbf{82.08}_{1.6}$	$74.03_{2.21}$	$51.83_{2.84}$	$47.04_{4.76}$	58.68
Switch-Only	$35.05_{2.81}$	$43.8_{16.02}$	$65.59_{2.61}$	$61.86_{6.26}$	$9.77_{12.86}$	$75.41_{3.29}$	$75.37_{0.7}$	$50.18_{3.44}$	$45.92_{3.03}$	51.44
Rational-Only	$37.72_{3.88}$	$64.75_{2.51}$	$69.69_{1.04}$	$79.86_{14.15}$	$23.20_{8.33}$	$78.58_{1.94}$	$75.84_{1.07}$	$52.27_{3.11}$	$46.48_{3.88}$	58.70
Baseline^{13}	$37.98_{5.80}$	$63.37_{4.72}$	$68.76_{1.55}$	$85.16_{3.63}$	$12.11_{12.69}$	$77.96_{2.23}$	$75.25_{0.71}$	$54.44_{2.06}$	$45.35_{3.72}$	57.80
AA-focused^{sim}	$45.97_{2.08}$	$68.36_{1.36}$	$73.98_{2.68}$	$86.83_{1.90}$	$\mathbf{37.43}_{3.10}$	$78.81_{3.58}$	$\mathbf{76.66}_{1.30}$	$55.96_{2.81}$	$48.44_{5.53}$	63.61
|AA|	$17.0_{1.3}$	$16.2_{1.0}$	$14.8_{1.8}$	$12.8_{3.2}$	$16.8_{2.2}$	$18.6_{1.9}$	$16.0_{1.1}$	$12.4_{1.2}$	$12.4_{2.1}$	
|Switch-Only|	$14.0_{1.1}$	$15.8_{2.5}$	$17.0_{1.9}$	$16.2_{2.8}$	$16.4_{1.9}$	$16.4_{1.5}$	$17.8_{1.7}$	$15.0_{2.1}$	$14.0_{1.7}$	

Table 4: Evaluating the impact of rational in adaptable adapters. Experiments are run for five different random seeds. *Switch-only* shows the results when learnable switches are used with standard adapter layers, i.e., linear layers with the ReLU activation. *Rational-only* shows the result when all the activation functions in the standard adapter are replaced with rational. *Baseline*^{13} contains a standard adapter layer on the last 13 transformer layer. *AA-focused*^{sim} contains adapter layers with rational activation on the last 13 layers.

Evaluating the impact of AA on selecting the number of beneficial layers. In the results of Table 2, we select the number of layers in *AA-focused*^{sim}, i.e., 13 , based on the minimum number of selected layers by *AA* on the *low-data-100* setting and for random seed 42. *AA-focused*^{sim} is equivalent to an adapter architecture in which only the last 13 adapter layers are added to the model. To investigate whether the improvements of *AA-focused*^{sim} over the baseline are only due to using a fewer number of adapter layers, we report the results of an adapter architecture in which only the last n adapter layers are added to the model, e.g., for $n = 13$ the resulting architecture is the same as *AA-focused*^{sim}. Table 3 shows the result of this experiment for $n = 13, 12, 11$. We observe that by decreasing the number of layers from 13 to 12, the overall performance drops notably from 63.61 to 57.37.

Evaluating the impact of rational activation. The results of *AA-focused* experiments vs. *Baseline* in Table 2 mostly emphasize the impact of layer selection by the learnable switches in *AA*. In this section, we investigate the impact of learnable activation functions in more details in the evaluations of Table 4.

First, we replace all rationals in *AA* with ReLU. The results are reported in the *Switch-Only* row. By comparing the results of *AA* and *Switch-only* we observe that the use of rational activation considerably improves the performance of *AA*, i.e., using rational is a key component to achieving higher performances with fewer layers.

Second, we replace the activation functions in the standard adapter with rational. The results are reported in *Rational-only* rows. The results of *Baseline* compared to *Rational-only* show that the impact of rational is prominent when the model contains fewer parameters and using rational with an overparameterized model is not very effective, i.e., both layer selection and learnable activation play an important role.

Third, we only add a standard adapter layer at the last 13 layers of BERT-large (*Baseline*^{13}), which is the same number of adapter layers in *AA-focused*^{sim}. The difference is the activation function that is used in these 13 adapter layers is ReLU in *Baseline*^{13} and rational in *AA-focused*^{sim}. The considerably higher performances of *AA-focused*^{sim} show that higher performances of *AA-focused* are due to both layer selection as well as a learnable activation function.

Learned rational activation functions. Figure 3 shows the learned activation functions across different layers of the same trained adapter and different tasks. We see that the learned activation differs

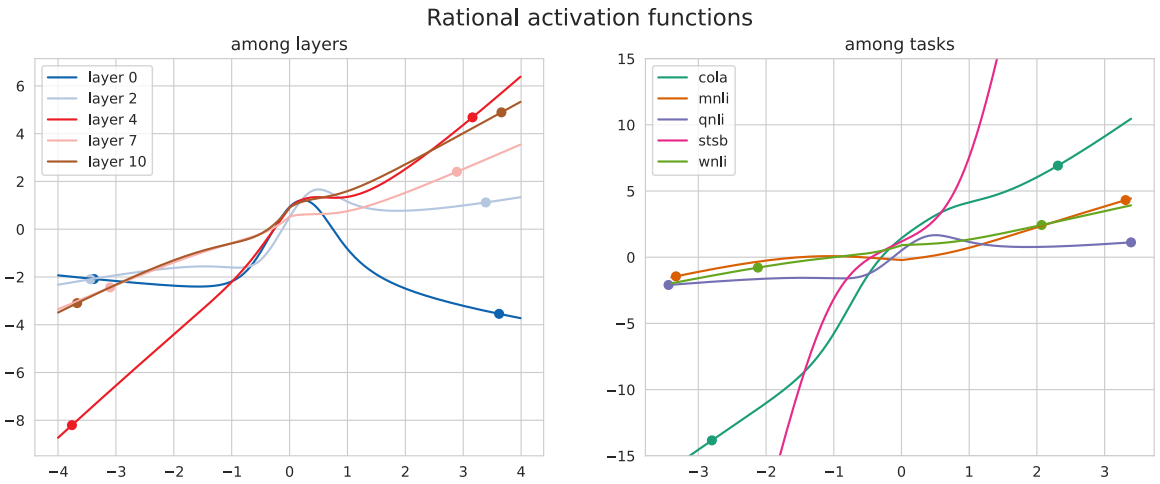

Figure 3: Learned rational activation functions differ according to their place within the network and to the task they are trained for. Right: activation functions at different layers within adapters trained on the QNLI task. Left: activation functions trained at layer 2 of adapters trained on different tasks.

for different layers of the same task as well as for different tasks.

6 Conclusion

In this paper, we propose adaptable adapters. They consist of a learnable switch to select a subset of beneficial adapter layers and a learnable activation function to learn the suitable activation at each adapter layer and for each input data. The results of adaptable adapters show that we can achieve on-par performances with the full adapter architecture by using a smaller subset of layers. We show that adaptable adapters are viable tools for designing efficient and effective adapter architectures that require less storage space, lower training and inference time with high performances.

Acknowledgements

The authors would like to thank Jorge Cardona for his valuable contribution to the implementation of adaptable adapters. We thank Michael Bugert, Ji-Ung Lee, and Soumya Sarkar for their constructive suggestions and feedback on this work. We would like to thank Jonas Pfeiffer and Clifton Poth for always being very helpful with all questions about adapters and AdapterHub. This research work has been funded by the German Federal Ministry of Education and Research and the Hessian Ministry of Higher Education, Research, Science and the Arts (HMWK) within their joint support of the National Research Center for Applied Cybersecurity ATHENE. It benefited from the HMWK cluster project "The Third Wave of AI".

References

Haoli Bai, Wei Zhang, Lu Hou, Lifeng Shang, Jin Jin, Xin Jiang, Qun Liu, Michael Lyu, and Irwin King. 2021. BinaryBERT: Pushing the limit of BERT quantization. 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)*, pages 4334–4348, Online. Association for Computational Linguistics.

Nicolas Boullé, Yuji Nakatsukasa, and Alex Townsend. 2020. Rational neural networks. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems (NeurIPS)*.

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

Kevin Clark, Urvashi Khandelwal, Omer Levy, and Christopher D. Manning. 2019. What does BERT look at? an analysis of BERT's attention. In *Proceedings of the 2019 ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 276–286, Florence, Italy. Association for Computational Linguistics.

Ido Dagan, Oren Glickman, and Bernardo Magnini. 2006. The PASCAL recognising textual entailment challenge. In *Machine learning challenges. evaluating predictive uncertainty, visual object classification, and recognising tectual entailment*, pages 177–190. Springer.

Quentin Delfosse, Patrick Schramowski, Alejandro Molina, and Kristian Kersting. 2021. Recurrent rational networks. *arXiv preprint arXiv:2102.09407*.

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

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

Angela Fan, Edouard Grave, and Armand Joulin. 2020. Reducing transformer depth on demand with structured dropout. In *International Conference on Learning Representations*.

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

Ruidan He, Linlin Liu, Hai Ye, Qingyu Tan, Bosheng Ding, Liying Cheng, Jiawei Low, Lidong Bing, and Luo Si. 2021. On the effectiveness of adapter-based tuning for pretrained language model adaptation. 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)*, pages 2208–2222, Online. Association for Computational Linguistics.

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

Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q Weinberger. 2016. Deep networks with stochastic depth. In *European conference on computer vision*, pages 646–661. Springer.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*.

Hector J Levesque, Ernest Davis, and Leora Morgenstern. 2011. The Winograd schema challenge. In *AAAI Spring Symposium: Logical Formalizations of Commonsense Reasoning*, volume 46, page 47.

Alejandro Molina, Patrick Schramowski, and Kristian Kersting. 2020. Padé activation units: End-to-end learning of flexible activation functions in deep networks. In *International Conference on Learning Representations*.

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 *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP 2020): Systems Demonstrations*, pages 46–54, Online. Association for Computational Linguistics.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392, Austin, Texas. Association for Computational Linguistics.

Adam Roberts, Colin Raffel, and Noam Shazeer. 2020. How much knowledge can you pack into the parameters of a language model? In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 5418–5426, Online. Association for Computational Linguistics.

Andreas Rücklé, Gregor Geigle, Max Glockner, Tilman Beck, Jonas Pfeiffer, Nils Reimers, and Iryna Gurevych. 2021. AdapterDrop: On the efficiency of adapters in transformers. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 7930–7946, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. In *Proceedings of the 5th Workshop on Energy Efficient Machine Learning and Cognitive Computing - NeurIPS 2019*.

Victor Sanh, Albert Webson, Colin Raffel, Stephen Bach, Lintang Sutawika, Zaid Alyafeai, Antoine Chaffin, Arnaud Stiegler, Arun Raja, Manan Dey, M Saiful Bari, Canwen Xu, Urmish Thakker, Shanya Sharma Sharma, Eliza Szczechla, Taewoon Kim, Gunjan Chhablani, Nihal Nayak, Debajyoti Datta, Jonathan Chang, Mike Tian-Jian Jiang, Han Wang, Matteo Manica, Sheng Shen, Zheng Xin Yong, Harshit Pandey, Rachel Bawden, Thomas Wang, Trishala Neeraj, Jos Rozen, Abheesht Sharma, Andrea Santilli, Thibault Fevry, Jason Alan Fries, Ryan Teehan, Teven Le Scao, Stella Biderman, Leo Gao, Thomas Wolf, and Alexander M Rush. 2022. Multitask prompted training enables zero-shot task generalization. In *International Conference on Learning Representations*.

Roy Schwartz, Gabriel Stanovsky, Swabha Swayamdipta, Jesse Dodge, and Noah A. Smith. 2020. The right tool for the job: Matching model and instance complexities. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 6640–6651, Online. Association for Computational Linguistics.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and

Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642, Seattle, Washington, USA. Association for Computational Linguistics.

Sainbayar Sukhbaatar, Edouard Grave, Piotr Bojanowski, and Armand Joulin. 2019. Adaptive attention span in transformers. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 331–335, Florence, Italy. Association for Computational Linguistics.

Zhiqing Sun, Hongkun Yu, Xiaodan Song, Renjie Liu, Yiming Yang, and Denny Zhou. 2020. MobileBERT: a compact task-agnostic BERT for resource-limited devices. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2158–2170, Online. Association for Computational Linguistics.

Raphael Tang, Yao Lu, Linqing Liu, Lili Mou, Olga Vechtomova, and Jimmy Lin. 2019. Distilling task-specific knowledge from bert into simple neural networks. *arXiv preprint arXiv:1903.12136*.

Ian Tenney, Dipanjan Das, and Ellie Pavlick. 2019. BERT rediscovers the classical NLP pipeline. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4593–4601, Florence, Italy. Association for Computational Linguistics.

Elena Voita, David Talbot, Fedor Moiseev, Rico Sennrich, and Ivan Titov. 2019. Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5797–5808, Florence, Italy. Association for Computational Linguistics.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. 2018. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 353–355, Brussels, Belgium. Association for Computational Linguistics.

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

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

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

A BERT-base Results

	MNLI	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE	WNLI	Avg
Baseline	**83.53**$_{0.19}$	**88.12**$_{0.14}$	**90.63**$_{0.26}$	**91.74**$_{0.36}$	**56.51**$_{0.84}$	**88.48**$_{0.14}$	84.8$_{1.07}$	63.83$_{1.4}$	**54.08**$_{6.64}$	**77.97**
AA	82.89$_{0.43}$	88.09$_{0.16}$	89.96$_{0.25}$	91.31$_{0.51}$	51.44$_{1.82}$	88.25$_{0.17}$	**85.09**$_{1.06}$	**64.25**$_{1.72}$	52.11$_{7.61}$	77.05
AA-Layers	9.8$_{0.3}$	11.2$_{0.7}$	10.6$_{1.0}$	9.8$_{1.1}$	8.6$_{2.1}$	11.4$_{0.4}$	9.0$_{0.6}$	9.4$_{0.7}$	8.0$_{1.4}$	
Low-data-100										
Baseline	35.66$_{3.38}$	29.70$_{0.86}$	60.51$_{4.5}$	51.54$_{2.14}$	-1.27$_{3.56}$	41.52$_{5.93}$	**74.86**$_{0.12}$	**50.4**$_{2.98}$	54.93$_{5.84}$	44.21
AA	**37.05**$_{2.35}$	**30.59**$_{0.68}$	**62.52**$_{4.27}$	**52.73**$_{2.55}$	**-0.08**$_{0.16}$	**48.73**$_{23.91}$	74.83$_{0.07}$	50.18$_{3.21}$	**55.21**$_{6.13}$	**45.75**
|AA|	6.4$_{1.8}$	8.6$_{2.1}$	8.8$_{1.7}$	8.6$_{1.6}$	7.4$_{2.4}$	10.8$_{0.7}$	9.4$_{1.4}$	9.4$_{1.4}$	8.2$_{0.9}$	
Low-data-300										
Baseline	37.88$_{4.09}$	49.24$_{10.32}$	68.17$_{2.9}$	75.53$_{3.49}$	3.40$_{8.59}$	69.39$_{15.05}$	75.99$_{1.2}$	54.22$_{2.96}$	**47.61**$_{4.91}$	53.49
AA	**40.27**$_{4.78}$	**66.31**$_{1.86}$	**74.03**$_{2.03}$	**76.42**$_{6.07}$	**3.56**$_{5.49}$	**82.06**$_{2.24}$	**76.12**$_{0.89}$	**54.73**$_{3.09}$	47.04$_{5.46}$	**57.84**
|AA|	10.4$_{1.6}$	10.8$_{0.7}$	11.0$_{0.8}$	9.4$_{1.3}$	7.6$_{2.0}$	10.8$_{0.7}$	9.6$_{1.0}$	9.8$_{1.4}$	8.2$_{1.1}$	
Low-data-500										
Baseline	42.82$_{2.4}$	67.63$_{1.44}$	72.7$_{1.31}$	83.46$_{0.64}$	**20.9**$_{4.14}$	81.97$_{0.89}$	76.51$_{0.95}$	**57.11**$_{2.93}$	**52.11**$_{6.96}$	61.69
AA	**47.72**$_{1.67}$	**69.27**$_{0.89}$	**75.649**$_{1.9}$	**84.52**$_{1.18}$	19.13$_{14.46}$	**83.74**$_{0.67}$	**78.03**$_{2.33}$	55.96$_{3.08}$	51.83$_{6.13}$	**62.87**
|AA|	9.8$_{1.1}$	10.4$_{1.3}$	10.0$_{0.8}$	9.2$_{0.7}$	9.4$_{1.8}$	10.6$_{1.4}$	9.8$_{1.6}$	9.6$_{1.0}$	8.0$_{1.5}$	

Table 5: Comparing the results of (a) the baseline adapter model that includes an adapter layer on all BERT-base layers (*Baseline*), and (b) the adaptable adapter (*AA*). The reported results are averaged over five different random seeds. The subscript reports the corresponding standard deviation. |*AA*| reports the average number of selected adapter layers by the adaptable adapter over different runs. The *full data* results show the performance when the model is trained on all the available training data. The *Low-data-*X settings report the results when only X examples are used for training the model. The test data is the same for all the experiments.