Adaptable Adapters

Anonymous ACL submission

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-resource scenarios.\footnote{The code will be publicly available upon publication.}

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., 2021) 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 for 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-resource 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-
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., 2019)—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 and 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 would require 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-resource scenarios. 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 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-resource scenarios.

2 Related Work

2.1 Rational Activation

Rational activation functions, empirically introduced as Padé Activation Units (Molina et al., 2019), 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 on image classification and reinforcement learning (Molina et al., 2019; 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. (2019) 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 require retraining 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) uses 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) investigate the impact of 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-resource scenarios.

3 Proposed Architecture

3.1 Learnable Activation

Empirical observations of performances have led experts of 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 on, 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.

---

\(^2\)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.

3.2 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 show 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 determine the beneficial layers for each input task during training. We use the Gumbel Softmax (\( GS \)) estimator as an end-to-end differentiable switch (hard attention) to make the network to attend to an element of a set (\( S \)). Assuming \( \pi_i \) are the probabilities of selecting each element of \( S \)—i.e., \( \forall i, \pi_i \geq 0, \sum_i \pi_i = 1 \)—\( GS \) estimates the hard attention \( y_i \) as follows:

\[
y_i = \frac{\exp((\log(\pi_i) + g_i) / \tau)}{\sum_j \exp((\log(\pi_j) + g_j) / \tau)}
\]

where \( g_i \) are i.i.d. samples from a Gumbel distribution, and \( \tau \) is a temperature parameter. Setting \( \tau \) 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 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.](image)

### 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.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Train</th>
<th>Metric</th>
</tr>
</thead>
<tbody>
<tr>
<td>MNLI</td>
<td>393k</td>
<td>acc.</td>
</tr>
<tr>
<td>QQP</td>
<td>364k</td>
<td>acc./F1</td>
</tr>
<tr>
<td>QNLI</td>
<td>105k</td>
<td>acc.</td>
</tr>
<tr>
<td>SST-2</td>
<td>67k</td>
<td>acc.</td>
</tr>
<tr>
<td>CoLA</td>
<td>8.5k</td>
<td>Matthews</td>
</tr>
</tbody>
</table>

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 models (Devlin et al., 2019). BERT-large contains 24 layers, an embedding size of 1024, and a total number of 340M parameters.\footnote{The results for BERT-base are reported in the supplementary materials. BERT-base contains 12 layers, an embedding size of 768, and 110M parameters.}

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\footnote{The reduction factor in the down-project layer is 16 which results in (768/16) x 768 x 2 parameters for each adapter layer.} resulting in the 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. (2019), 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 $\tau$ to 0.1, and we initialize $\pi_i$ 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 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-resource scenarios. 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-resource evaluations when 100, 300, and 500 annotated examples are available.\footnote{Selected training examples for low-resource experiments are the same for all models given the same random seed.} The test data is the same for all the evaluations. We run all the low-resource experiments for 20 epochs and five different random seeds.\footnote{For instance, the selected layers for RTE are as follows for different runs of Low-resource-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\}.} The results for BERT-large are reported in the supplementary materials. BERT-large contains 12 layers, an embedding size of 768, and 110M parameters.
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) AdapterDrop, (c) adaptable adapter (AA), 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\textsuperscript{pec} is selected based on the selected layers by AA for the task of QQP on the Low-resource-100 setting and for random seed 42. AA-focused\textsuperscript{ref} 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\textsuperscript{ref}. The number of layers at the inference time for the AdapterDrop\textsuperscript{AA} experiments is selected based on the number of layers in the corresponding AA-focused\textsuperscript{pec} experiments. The number of inference time layers for AdapterDrop\textsuperscript{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\textsuperscript{1}| rows report the number of added adapter layers in the corresponding AA-focused\textsuperscript{1} experiments. AA-focused\textsuperscript{ref} and |AA-focused\textsuperscript{pec}| are the same for all data settings. |AdapterDrop\textsuperscript{1}| rows report the number of included adapter layers for the corresponding AdapterDrop experiment at the inference time. |AdapterDrop\textsuperscript{AA}| is always the same as the corresponding |AA-focused\textsuperscript{AA}| and |AdapterDrop\textsuperscript{13}| is always the same as AA-focused\textsuperscript{13}. 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.
experiments of RTE for Low-resource-100—i.e., over the five different random seeds—. However, it is different for the rest of the tasks and different data regimes.

- **AA-focused**\(^{\text{uni}}\): 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-resource-100. We choose low-resource-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., 3 layers from the first half of the original 24 layers, and 10 layers from the second half. The results of AA-focused\(^{\text{uni}}\) compared to AA-focused\(^{\text{spec}}\) indicate whether the selected architecture by AA transfers between similar tasks and different data regimes.

- **AA-focused**\(^{\text{sim}}\): 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 regime—i.e., the number of selected layers when the random seed is 42 for the QQP task and the low-resource-100 setting that is 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\(^{\text{sim}}\) experiments. The results of AA-focused\(^{\text{sim}}\) compared to AA-focused\(^{\text{uni}}\) 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\(^{\text{AA}}\) are equivalent to the number of layers in AA-focused\(^{\text{spec}}\) experiments for each task and data setting. The number of layers for AdapterDrop\(^{13}\) is 13, which is the same as AA-focused\(^{\text{uni}}\) and AA-focused\(^{\text{sim}}\). 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 last row in Table 2 shows the average number of selected layers for each task over the five different random seeds. \(\text{AA-focused}^{\text{spec}}\) rows report the number of added adapter layers in the corresponding AA-focused experiments. \(\text{AdapterDrop}^{13}\) 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-resource scenarios while using considerably smaller number of parameters, and therefore, being considerably more efficient. For instance, while AA-focused\(^{\text{uni}}\) 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-resource-100, Low-resource-300, and Low-resource-500, respectively.

- The high performances of AA-focused\(^{\text{uni}}\) show that the selected architecture by AA for one task and one data regime 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.

- The higher performances of AA-focused\(^{\text{uni}}\) compared to AA-focused\(^{\text{sim}}\) indicate that the higher performances of AA-focused models are not only due to using fewer adapter layers, but it is also important that which adapter layers are selected.

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

\(^9\) It even outperforms AA-focused\(^{\text{spec}}\) showing that AA-focused\(^{\text{spec}}\) 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.
The results of performances with fewer layers. Rational is a key component to achieve higher performance. We observe that the use of rational activation considerably improves the performance of AA, i.e., using rational is a key component to achieve higher performance 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 more 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\textsuperscript{13}), which is the same number of adapter layers in AA-focused\textsuperscript{sim}. The difference is the activation function that is used in these 13 adapter layers is ReLU in Baseline\textsuperscript{13} and rational in AA-focused\textsuperscript{sim}. The considerably higher performances of AA-focused\textsuperscript{sim} shows that higher performances of AA-focused\textsuperscript{sim} are due to both layer selection as well as a learnable activation function.

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

### 6 Conclusion

In this paper we propose adaptable adapters. They consist of a learnable switch to select a subset of 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 fewer storage space, lower training and inference time with high performances.
References


Language Processing (EMNLP), pages 5418–5426, Online. Association for Computational Linguistics.


A BERT-base Results
Table 4: 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. The AA-Layers 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-resource-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.