CS 7643 Project Report: Transformer Adapters

Singapore Sling: Kunting Chua, Abhishek Pradhan, Win Min Tun
Georgia Institute of Technology, Atlanta, GA
{kchua, abhishek.pradhan, wtun3}@gatech.edu

1. Introduction

Transfer learning [15] is a method of supervised learning, where knowledge is acquired from a source task and applied to a different target task. pretraining has been one of the most common transfer learning methods in NLP: language models such as BERT [3] and RoBERTa [8] are readily accessible with pretrained weights.

Although the fine-tuning of models has allowed NLP researchers to achieve SOTA results on various tasks [4, 18], it has multiple drawbacks. Firstly, the models are pretrained on a general domain, resulting in limitations when directly applying them to data from other domains, such as computer science. Essentially, the original representations extracted from the pretrained model do not provide the best results for domain-specific tasks. Gururangan *et al*. (2020) (hereafter G20), showed that one way to overcome such a deficiency is through domain-adaptive and task-adaptive pretraining, which involves fine-tuning of the model on the new domains and tasks, respectively.

Additionally, these models have up to billions of parameters and require huge amounts of data for fine-tuning on the target task in order to achieve good accuracy. It is also challenging to learn from and share information across different tasks. In the continual approach, one can fine-tune using multiple tasks sequentially: the parameters learned with the first task is used to initialize the model for the second task. Unfortunately, such a sequential training process causes the model to forget knowledge learned from the previous tasks, which prevents transfer learning especially across many tasks. These downsides limit the benefits that a large language model can offer [5].

In this project, we investigate the use of adapters to circumvent the problems associated with full fine-tuning. Introduced by Houlsby *et al*. (2019) [5] for NLP, adapters are modules with trainable weights that can be injected into existing pretrained networks. With adapter-based training, the original network is kept fixed, and only the adapter parameters are updated. Thus the number of training parameters involved is only a small fraction (typically less than a few per cent) compared to the full language model. Adapters have been shown to work for machine translation [1], text matching [16], and cross-lingual transfer learning [13, 22]. In addition, adapters avoid the problem of catastrophic forgetting associated with sequential learning. Knowledge can be shared across multiple tasks by combining multiple pretrained adapters, allowing efficient transfer learning [11].

Although adapters are computationally efficient, previous work have shown limitations in achieving SOTA performance on NLP tasks. In this project, we quantify the performance of adapter-based training, starting with a pretrained language models such as BERT and RoBERTa [8]. We perform domain adaptation using adapters and compare the results to those achieved by G20 [4] with a fine-tuning procedure. For this task, we focus on the computer science (CS) domain and rely on two datasets: ACL-ARC [7] and SCIERC [10], that have shown the highest improvement with domain adaptation [4]. Furthermore, we will perform task adaptation and investigate different adapter structures, as well as the use of AdapterFusion to combine adapters. This is performed on the Stanford Sentiment Treebank (SST) and the Corpus of Linguistic Acceptibility (CoLA) [20] datasets, both part of the General Language Understanding Evaluation (GLUE) benchmark [19].

2. Datasets

In this project, we make use of 4 datasets, which we describe as follows. In general, each dataset is divided into training, development and test sets, with the exact split shown in Table 1.

2.1. ACL-ARC Citation Intent

We use the ACL-ARC citation intent dataset [7] based on the ACL Anthology Reference Corpus [2]. The dataset contains 1941 citation instances taken from 186 papers published by ACL Anthology, annotated by NLP experts. Each instance includes the citation, its surrounding context, and information related to the citing and cited paper. The dataset is divided into six citation intent categories: Background (51.36%), Extends (3.71%), Uses (18.75%), Motivation (4.53%), CompareOrContrast (18.08%), Future Work (3.55%).

Domain	Dataset	Task	Train	Dev	Test	Classes	Source
CS	ACL-ARC	citation intent	1688	114	139	6	
CS	SCIERC	relation classification	3219	455	974	7	
	CoLA	linguistic acceptibility classification	8551	1043	1063	2	Hugging Face dataset [1]
	SST	Sentiment Analysis	8544	1101	2210	2	Hugging Face dataset

Table 1. Specifications of datasets used in the project. Columns 4-6 denote the number of examples in the training, development, and test sets respectively.

2.2. SCIERC (Relation Classification)

For relation classification in the computer science domain, we used the SCIERC dataset [10]. The dataset contains 4648 relation instances taken from 500 scientific abstracts collected from 12 artificial intelligence conferences. Each instance contains a scientific abstract, location of two entities in the abstract and a relation label between two entities. There are seven types of relations present in the dataset: Compare (5.01%), Part-of (5.79%), Conjunction (12.52%), Evaluate-for (9.77%), Feature-of (5.68%), Used-for (52.43%), Hyponym-Of (8.8%).

2.3. Corpus of Linguistic Acceptibility (CoLA)

The CoLA dataset [20] and consists of sentences drawn from 23 linguistics publications. These sentences have been annotated to indicate whether they constitute grammatical English sentences, and can be used to train a binary classifier.

2.4. Stanford Sentiment Treebank (SST)

The SST dataset [17] is a crucial dataset because of its ability to test sentiment analysis performance. Each sentence has been annotated with a score to indicate how positive it is. For our binary classification task, we round the score to 1 (positive) or 0 (negative).

3. Approach

We implement adapters using pretrained RoBERTa and BERT Transformer networks as base models. Both the adapters and the pretrained based model are implemented using the AdapterHub framework [12], an extension of the Hugging Face transformers [21] library. Using PyTorch as a backend, these libraries provide high-level functions which simplify the implementation of adapters and Transformers.

3.1. Adapters

The first adapters were introduced for computer vision by Rebuffi *et al*. (2017) [14], and subsequently for NLP by Houlsby *et al*. (2019) [5]. In adapter-based fine-tuning, a new parameter function $\psi_{w,v}(x)$ is introduced where w are the parameters from the existing language model, and v are new adapter parameters. v is initialized such that the overall function resembles the original function of the language model, i.e. $\psi_{w,v_0}(x) \approx \phi_w(x)$. Most importantly, w is kept frozen during training, thus only the parameters v are updated. In general, the size of v is significantly smaller than that of w, i.e. $|v| \ll |w|$.

For transformer networks, the parameter v is introduced using adapter modules that are inserted at each transformer layer. In general, each adapter module consists of a two-layer feed-forward neural network with a bottleneck. Within a transformer layer, adapters can be placed either (i) after the bottom layer (the multi-head attention layer), or (ii) after the top layer (i.e. the feed-forward layers), or (iii) at both locations. Depending on the exact implementation, residual connections and Layernorm can also be used to improve performance. The newly-introduced parameters are then trained on downstream tasks, along with a task-specific output layer for predictions, while keeping the transformer parameters fixed. The flexibility of adapters constitutes a general architecture which can work for various NLP tasks.

In this project, we use two specific adapter architectures created by Houlsby *et al*. (2019) [5] and Pfeiffer *et al*. (2020) [12]. The sub-layers of an adapter model utilizes a two-layer feed-forward architecture: the d-dimensional features are first projected to a smaller dimension m, then a non-linearity is applied before projecting back to d-dimensions. The bottleneck m keeps the number of adapter parameters small. In practice, m is controlled by the *reduction factor* = d/m. For $d = 768$ (e.g. in BERT and RoBERTa), a reduction factor of 16 corresponds to a bottleneck size of $m = 768/16 = 48$.

Houlsby *et al*. (2019) showed that adapters at lower layers have less impact on accuracy than those at higher layers [5]. Hence, we will conduct experiments varying the adapter sizes across the layers and attempt to reduce the number of trainings parameters without compromising the accuracy. We will also varying other aspects of the adapter configurations and evaluate the effects on the performance.

3.2. AdapterFusion

Pfeiffer *et al*. (2021) introduces the AdapterFusion architecture [11], which combines a set of N adapters, each trained on a different task. This is done by introducing a new set of parameters, which can be trained to combine the N adapter outputs depending on the target task. The AdapterFusion layer applies SoftMax attention, with the Query obtained from a transformation of the output from

the feed-forward sub-layer of the transformer, and the Value and Key obtained from a transformation of the output of each adapter. As such, the AdapterFusion module is able to learn to attend to the most useful adapter for a particular task.

3.3. Task Adaptive Adapter Pretraining (TAAP) and Text Classification

By performing domain adaption using the RoBERTa-base model and then performing text classification, G20 was able to outperform the results obtained by RoBERTa-base model [4]. Inspired by their work, we divide our process into two steps: (i) task adaptive adapter pretraining (TAAP) and (ii) text classification. TAAP involves performing masked language modeling (MLM) on the new dataset. We use the original pretrained RoBERTa-base model and inject adapters in all the Tranformer layers. We obtain unlabeled text by discarding the provided labels, and randomly mask words with a masking probability of 15% for MLM training. Subsequently, the adapters weights obtained after TAAP are used as to initialize the parameters for text classification.

For the SCIERC dataset, we add four new tokens ($< e1 >, < /e1 >, < e2 >, < /e2 >$) to the pretrained RoBERTa tokenizer to represent the start and end of entities 1 and 2. Since the relations are directional, with the relationship flowing from entity 1 to entity 2, these four tokens are added to make sure the special tokens are not further tokenized.

As RoBERTa is fine-tuned on a very large dataset, we expect it to provide sufficient information about the syntax and semantics of a the input text. Therefore, even without full fine-tuning, adapters can learn to extract the relevant information from the base model and apply it to a new target task. If successful, this method can also be used for low-resource tasks, therefore, we also studied the effect of different adapter architecture on a low-resource dataset e.g. ACL-ARC.

4. Experiments and Results

In our experiments, we use the RoBERTa and BERT as the pretrained language model. We perform all experiments in this section on a system with 6vCPUs, 56GB RAM, and Nvidia K80 GPU with 12GB of VRAM. The AdamW [9] optimizer with parameters $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $epsilon = 1e-6$ are used, with weight decay kept at 0. For all classification tasks, cross-entropy is used as the loss function.

4.1. Task Adaptive Adapter Pretraining (TAAP) and Text Classification

In this section, we focus on performing TAAP and text classification on the ACL-ARC and SCIERC datasets.

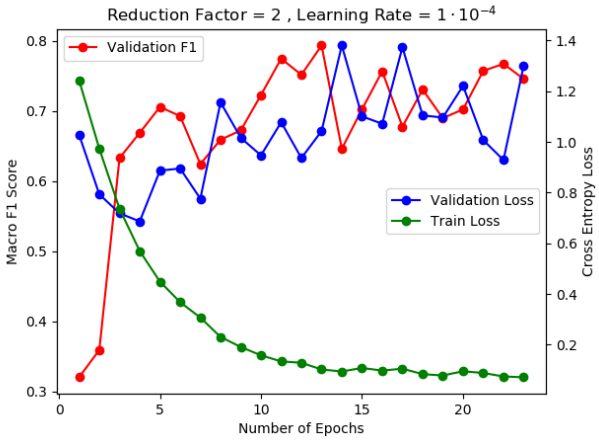

Figure 1. Learning curves for adapter training on the ACL-ARC dataset. With more epochs, the model is better at classification, but becomes less confident in the prediction on validation data.

Multiple experiments are performed to select values for the hyperparameters (learning rate and number of training epochs). Training is performed for 50 epochs, and early stopping with a patience of 10 epochs is applied. The learning rate is gradually decreased with the "cosine with restart" learning rate scheduler, and warmup steps of 3 and 5 for TAAP and text classification, respectively. We find that both a linear learning rate and a cosine scheduler provide comparable results. However, the cosine scheduler provided better stability over multiple experiments and datasets.

Results of all different types of adapter pretraining are shown in Table 2, with the results from G20 shown for baseline comparison. For both datasets, we are able to achieve better results than the supervised classification fine-tuning performed on the RoBERTa-base model. We are also able to achieve a significantly higher F1 score than task-adaptive pretraining (TAPT) from G20 for the ACL-ARC dataset and a slightly higher score for the SCIERC dataset, and comparable accuracy for domain-adaptive pretraining (DAPT) [4] for both datasets. However, the G20 result using both DAPT and TAPT still outperforms our TAAP result. All these results have been achieved by fine-tuning less than 5% of trainable parameters as compared to full fine-tuning for the RoBERTa-base model as shown in Table 4.

Apart from showing the power of adapters in domain adaptation and classification, we also study the effect of the learning rate, reduction factor, and type of adapters in a low-resource setting in the following sub-sections. As architectural changes have more impact in the low-resource setting as compared to learning rate, to study the effect of learning rate we used both ACL-ARC and SCIERC datasets, and to study the impact of adapters in the low-resource setting we use only the ACL-ARC as it has significantly fewer samples

Datasets	Gururangan *et al*. (2020)				Our method
	RoBERTa	DAPT	TAPT	DAPT + TAPT	TAAP
ACL-ARC	63.0	75.4	67.4	**75.6**	74.5
SCIERC	77.3	80.8	79.3	**81.3**	79.4

Table 2. Comparison of domain adaption and task adaptation to datasets in the CS domain. Results from Task Adaptive Adapter Pretraining (our method) and those from Gururangan *et al*. (2020) [4] are shown. Reported scores are test macro-F1 scores.

Adapter	**Pfeiffer**			**Pfeiffer**			**Pfeiffer**			**Houlsby**		
Dataset	**ACL-ARC**			**SCIERC**			**ACL-ARC**			**ACL-ARC**		
Learning Rate ($\cdot 10^{-3}$)	3	1	0.1	3	1	0.1	0.1			0.1		
Reduction Factor	16			16			2	16	64	2	16	64
Task Adaptive Adapter Pretraining												
Epochs	48	50	32	22	22	22	50	32	32	50	32	32
Train Loss	1.4	1.2	1.9	1.6	1.4	1.8	1.4	1.9	1.9	1.4	1.9	1.9
Test Loss	1.6	1.2	1.7	2.8	2.6	2.2	1.3	1.7	1.8	1.2	1.8	1.8
Test Perplexity	4.9	**3.5**	5.9	16.4	13.1	**8.8**	**3.5**	5.9	6.2	**3.5**	5.9	6.2
Text Classification												
Epochs	13	23	40	13	50	22	23	40	33	41	19	33
Train Loss	1.4	0.3	0.3	1.5	0.1	0.3	0.3	0.3	0.5	0.1	0.5	0.9
Test Loss	1.1	1.6	1.9	0.9	1.6	1.1	1.3	1.9	1.2	1.7	1.0	1.2
Test F1	28.3	70.7	**71.2**	54.6	**79.4**	75.3	**74.5**	71.2	62.2	**69.2**	63.8	69.1

Table 3. Effect of changing the learning rate, reduction rate and adapter architectures on TAAP and text classification on the ACL-ARC and SCIERC dataset. **Best values marked in bold**

Pfeiffer	Houlsby	RoBERTa-base	BERT-base
1, 536, 921	2, 431, 449	124, 697, 433	109, 514, 298

Table 4. Comparison of the number of trainable parameters for the two adapter architectures and transformer models used in this project.

than the SCIERC dataset (see Table 1).

4.1.1 Effect of Learning Rate

We compare three learning rates $\{3{\cdot}10^{-3}, 1{\cdot}10^{-3}, 1{\cdot}10^{-4}\}$, and show the results in Table 3. In general, we find that the learning rate has a very significant impact on adapter training. For both datasets, with a large learning rate of $3 \cdot 10^{-3}$, there is some instability in the text classification task, leading to a low F1 score. We suspect this is due to the overshooting of local minima associated with the high learning rate. We find that smaller learning rates ($1 \cdot 10^{-3}$ and $1 \cdot 10^{-4}$) lead to a stable training process. As a result, we achieve a significantly higher F1 score, outperforming those of G20 in some cases.

4.1.2 Effect of Architectural Changes

Reduction Factor : In addition, we investigate the effect of varying the reduction factor (i.e. bottleneck size) on the learning process. We consider reduction factors $\{2, 16, 64\}$ with a learning rate of 10^{-4}. The results for the Pfeiffer and Houlsby adapters are shown in Table 3. We find that a reduction factor of 2 resulted in the lowest perplexity and the highest F1 score, regardless of adapter architecture. This is due to the larger adapter size (i.e. number of adapter parameters).

Adapter Type : Both architectures produced similar perplexity for TAAP but adapter architecture has a big impact on the final F1 score for the text classification task. The results for the Pfeiffer and Houlsby adapters are shown in Table 3. The Pfeiffer architecture results in significantly higher F1 score compared to the Houlsby architecture, and also requires fewer training epochs. This shows that the Pfeiffer architecture not only performs better and is also more efficient on these datasets.

Figure 1 shows the learning curves for adapter training on the ACL-ARC dataset. As the F1 score improves, the validation loss increases. This implies that as the model gets more accurate, it gets less confident in its prediction. The decrease in confidence can be due to one of the following or more reasons: (1) the information in the dataset is difficult to learn. it is difficult to find a clean decision boundary between classes, (2) labels in the development dataset are messy and do not replicate the statistics of training data, and (3) small amount of available training data.

4.2. Experiments with Adapter Configs on SST

In this section, we investigate in detail the effect of changing adapter configurations on the SST dataset, using the Houlsby adapter as the default baseline [5]. With the default Houlsby setting, the reduction factor is 16, correspond-

	Act.	Adapter size (m)	Accuracy
i	Swish	All layers: 48	0.8384
ii	ReLU	All layers: 48	0.8447
iii	Swish	Layers 1-8: 24, 9-12: 48	0.8366
iv	Swish	Layers 1-8: 16, 9-12: 48	0.8357
v	Swish	Layers 1-10: 24, 11-12: 48	0.8343
vi	Swish	1-10: 0, 11: 48, 12: 96	0.8294
vii	Swish	1-9: 0, 10: 32, 11: 48, 12: 64	0.8316

Table 5. Results of changing adapter configurations on the SST dataset. The second column shows the adapter sizes at different layers in the Transformer network. An adapter size of 0 implies that no adapters are added at the layer.

ing to an adapter size of $m = 48$. The base model here is the pretrained BERT Transformer network (bert-base-uncased). Training is performed with a learning rate of 10^{-5} and a weight decay of 10^{-2}.

4.2.1 Effect of Activation Function

The result of adapter training to SST with the default Houlsby configuration is shown in the row (i) of Table 5. We change the default activation function from Swish to ReLU, and find that it results in better performance (higher accuracy) on the test set.

4.2.2 Effect of Adapter Size and Reduction Factor

In the next set of experiments, we vary the adapter sizes across the layers and decrease the adapter sizes by a factor of 2 or 3 at the lower layers of the Transformer. The exact configurations investigated here are shown in rows iii-v of Table 5, with the corresponding test accuracy. For example in experiment v, we reduce the adapter sizes of layers 0-9, leaving last two layers with the default size of 48. Compared to the default baseline (row i), we do not find an impact on accuracy even when the sizes of the earlier layers are decreased by a factor of 3. This finding reaffirms our understanding that adapters rely more on the higher layers for task adaptation: lower layers extract lower-level features that are shared among tasks, while the higher layers are the ones which build features unique to different tasks.

This result leads us to a question: can we fine-tune only the higher layers of the Transformer with adapters, following the popular strategies used in model fine tuning [6]. As such, we completely remove adapters in lower layers and fine-tune only those in the higher layers. We keep the total adapter size constant by increasing the sizes of the remaining higher layers. Experiment vi fine-tunes only the last two layers, while experiment vii fine-tunes the last three layers, with the results shown in Table 5.

Our results show that there is a decrease in accuracy when leaving out adapters at the lower layers. Comparing

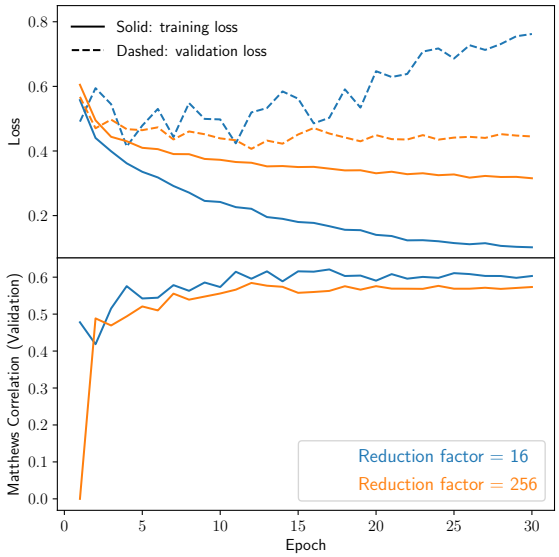

Figure 2. Learning curves for adapter training on the CoLA dataset, for reduction factors of 16 (blue) and 256 (orange). Although a smaller reduction factor results in more overfitting to the training set (upper panel), the overall performance as measured with the Matthews correlation is higher (lower panel).

experiments vi and vii, we find the performance reduction is slightly more severe when the adapter is removed from layer 10 (the third-to-last layer). Thus, it is possible to achieve slightly better performance by spreading the total adapter size across a few more last layers, rather spending it all in the last two layers.

4.3. Task adaptation and AdapterFusion on CoLA

In this section, we focus on task adaptation to the CoLA dataset. For this task, the results are evaluated with the Matthew's correlation coefficient (MCC). As a correlation coefficient, an MCC value of +1/-1/0 represents perfect positive/perfect negative/no correlation.

We adopt the adapter architecture proposed by Pfeiffer *et al.* (2021) [11] and focus on (i) the effect of different values of the reduction factor, and (ii) AdapterFusion. RoBERTa is used as the base pretrained model. For all experiments on CoLA in this section, a learning rate of 10^{-4} is adopted with no learning rate decay. Training is performed for 30 epochs. At each epoch, the MCC is evaluated on the development set, and the final model is the one with the highest correlation across all epochs.

4.3.1 Effect of Reduction Factor

The first set of experiments is performed by training the Pfeiffer adapters with reduction factors $\{2, 16, 64, 256\}$ fixed across all 12 transformer layers. As an illustration,

	Model	MCC
1.	No adapter (only classification head)	0.2279
2.	Full fine-tuning on BERT [5]	0.605
	Adapter: Reduction Factors	
3.	All layers 2	**0.6318**
4.	All layers 16 (Default)	0.6209
5.	All layers 64	0.6164
6.	All layers 256	0.5844
	AdapterFusion	
7.	Fusion (QNLI, MNLI)	0.5803
8.	Fusion (MRPC, QQP)	0.5806
9.	Fusion (SST-2, IMDB)	0.5676
10.	Fusion (CoLA, QNLI, MNLI)	**0.6410**
11.	Fusion (CoLA, MRPC, QQP)	0.6318
12.	Fusion (CoLA, SST-2, IMDB)	0.6356

Table 6. Task-adaption to the CoLA dataset: Matthew's correlation on the development set from varying the reduction factor and AdapterFusion. For the AdapterFusion models, parentheses denote the pretrained adapters which have been combined.

we plot the learning curves of the training and validation losses, and the MCC for two reduction factors in Fig. 2.

The upper panel of Fig. 2 shows that there is significant overfitting to the training set when the reduction factor is small (here, 16). This is similar to what we observed in Section 1 for a different dataset. Despite the overfitting, the larger adapter size allows the smaller reduction factor to reach better performance, i.e. a larger Matthew's correlation.

The results of varying the reduction factor is shown in rows 3-6 of Table 6. Row 1 shows the result without adapter training, i.e. training a classification layer while keeping the RoBERTa parameters fixed. In this case, the lack of both adapter training and fine-tuning results in poor performance. Overall, we find that adapters enable the RoBERTa model to adapt appropriately to the CoLA classification task, achieving performance similar to those obtained from full fine-tuning with BERT [5]. Decreasing the adapter size (increasing the reduction factor) results in lower performance, in line with the results we found in the previous sections.

4.3.2 AdapterFusion

Finally, we use AdapterFusion to combine the trained CoLA adapter (with the default reduction factor of 16) with other pretrained adapters. Since the focus here is not to create a database of adapters on various training sets, we will rely on pretrained adapters obtained from AdapterHub. We investigate AdapterFusion using 3 groups of adapters, each with 2 adapters separately pretrained on the following datasets:

1. Inference tasks: The Multi-Genre Natural Language Inference Corpus (MNLI) and the Stanford Question Answering Datasets (QNLI).
2. Semantic tasks: The Microsoft Research Paraphrase Corpus (MRPC) and the Quora Question Pairs2 (QQP) datasets.
3. Sentiment analysis tasks: The Stanford Sentiment Treebank (SST-2) and IMDB datasets.

To understand how well these pretrained adapters apply to CoLA, we perform AdapterFusion with and without the CoLA adapter for each group. With the inference tasks as an example, we obtain one AdapterFusion model by combining the CoLA, MNLI and QNLI pretrained adapters. A second model is obtained by combining only the MNLI and QNLI adapters. The results of the experiments are shown in rows 7-13 of Table 6.

Rows 7-9 show that even when using adapters pretrained on other datasets, the model is able to achieve satisfactory performance close to our previous single-task results (rows 3-6). We find that AdapterFusion with the sentiment analysis pretrained adapters results in the lowest performance among the three groups, possibly because the datasets are most different from CoLA. When the CoLA adapter is added to the other pretrained adapters (rows 10-12), the performance increases significantly. The best performance is achieved from a combination of the CoLA, QNLI and MNLI adapters, outperforming the baseline from Houlsby *et al.* (2019) [5] as well as our own single-task CoLA adapters. This means that the knowledge from the QNLI and MNLI adapters is complementary to the CoLA classification task. Overall, our results here demonstrates that sharing information across different datasets is beneficial to performance on CoLA, and highlights the importance of multi-task learning.

5. Discussion and Future Work

The adapters have shown very promising results in almost all the experiments that were performed. Due to time and computational constraints, each experiment is only performed once, but performance can be affected by random initialization of the parameters. Ideally, to ensure that the results are robust, each experiment should be conducted multiple times with different random initializations, so that average scores for the experiments can be obtained.

Due to the large size of domain adaptation datasets presented in G20, we were unable to replicate the domain adaptive pretraining with adapters in our study, however we have compared our TAAP method to their results.

Through our experiments with adapter size and configurations, we have shown that adapters are only required in the last layers, which further reduces the number of parameters required. However, this was only shown on a few datasets. Further validation with other tasks needs to be performed in

the future.

6. Work Division

The delegation of work among team members is shown in Table 7. All members contributed equally to the writing of the final report.

References

[1] Ankur Bapna and Orhan Firat. Simple, scalable adaptation for neural machine translation. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 1538–1548, Hong Kong, China, Nov. 2019. Association for Computational Linguistics. 1

[2] Steven Bird, Robert Dale, Bonnie J. Dorr, Bryan Gibson, Mark T. Joseph, Min-Yen Kan, Dongwon Lee, Brett Powley, Dragomir R. Radev, and Yee Fan Tan. The ACL Anthology Reference Corpus: A Reference Dataset for Bibliographic Research in Computational Linguistics. In *Proc. of the 6th International Conference on Language Resources and Evaluation Conference (LREC'08)*, pages 1755–1759, 2008. 1

[3] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019. 1

[4] Suchin Gururangan, Ana Marasović, Swabha Swayamdipta, Kyle Lo, Iz Beltagy, Doug Downey, and Noah A. Smith. Don't stop pretraining: Adapt language models to domains and tasks. *ArXiv*, abs/2004.10964, 2020. 1, 3, 4

[5] N. Houlsby, A. Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and S. Gelly. Parameter-efficient transfer learning for nlp. In *ICML*, 2019. 1, 2, 4, 6, 8

[6] Jeremy Howard and Sebastian Ruder. Universal language model fine-tuning for text classification. pages 328–339, 01 2018. 5

[7] David Jurgens, Srijan Kumar, Raine Hoover, Dan McFarland, and Dan Jurafsky. Measuring the evolution of a scientific field through citation frames. *Transactions of the Association for Computational Linguistics*, 6:391–406, 2018. 1

[8] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach, 2019. 1

[9] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization, 2019. 3

[10] Yi Luan, Luheng He, Mari Ostendorf, and Hannaneh Hajishirzi. Multi-task identification of entities, relations, and coreferencefor scientific knowledge graph construction. In *Proc. Conf. Empirical Methods Natural Language Process. (EMNLP)*, 2018. 1, 2

[11] Jonas Pfeiffer, Aishwarya Kamath, Andreas Rücklé, Kyunghyun Cho, and Iryna Gurevych. Adapterfusion: Non-destructive task composition for transfer learning. In *EACL*, 2021. 1, 2, 5

[12] Jonas Pfeiffer, Andreas Rücklé, Clifton Poth, Aishwarya Kamath, Ivan Vuli'c, Sebastian Ruder, Kyunghyun Cho, and Iryna Gurevych. Adapterhub: A framework for adapting transformers. *ArXiv*, abs/2007.07779, 2020. 2

[13] Jonas Pfeiffer, Ivan Vuli'c, Iryna Gurevych, and Sebastian Ruder. Mad-x: An adapter-based framework for multi-task cross-lingual transfer. In *EMNLP*, 2020. 1

Student Name	Contributed Aspects	Details
Abhishek Pradhan	Task adaptive adaptive pretraining (TAAP) and text classification	Pre-processing of ACL-ARC and SCIERC dataset, trained RoBERTa model for both Houlsby and Pfeiffer, and all experiments related to TAAP and text classification. Worked on (1) transformer_adapter_acl_arc.py, (2) transformer_adapter_sciERC.py
Kunting Chua	Adapter training on the CoLA dataset	Trained single-task adapters and AdapterFusion modules. Conducted experiments with reduction factors and combinations of pretrained adapters and analyzed the results. Worked on COLA-roberta.ipynb
Win Min Tun	Experiments with Adapter Configurations on SST	Conducted experiments with adapter configurations and analyzed the results. Verified the better adapter fine-tuning methods.

Table 7. Contributions of team members.

[14] Sylvestre-Alvise Rebuffi, Hakan Bilen, and A. Vedaldi. Learning multiple visual domains with residual adapters. In *NIPS*, 2017. 2

[15] Sebastian Ruder. *Neural transfer learning for natural language processing*. PhD thesis, NUI Galway, 2019. 1

[16] Andreas Rücklé, Jonas Pfeiffer, and Iryna Gurevych. Multicqa: Zero-shot transfer of self-supervised text matching models on a massive scale, 2020. 1

[17] Richard Socher, Alex Perelygin, Jean Y. Wu, Jason Chuang, Christopher D. Manning, Andrew Y. Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. *EMNLP*, 1631, 2013. 2

[18] Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. Superglue: A stickier benchmark for general-purpose language understanding systems, 2020. 1

[19] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. GLUE: A multi-task benchmark and analysis platform for natural language understanding. 2019. In the Proceedings of ICLR. 1

[20] Alex Warstadt, Amanpreet Singh, and Samuel R Bowman. Neural network acceptability judgments. *arXiv preprint arXiv:1805.12471*, 2018. 1, 2

[21] 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. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online, Oct. 2020. Association for Computational Linguistics. 2

[22] Ahmet Üstün, Arianna Bisazza, Gosse Bouma, and Gertjan van Noord. Udapter: Language adaptation for truly universal dependency parsing, 2020. 1

A. Appendices

A.1. Adapter Architecture

Here, we show the detailed diagram of the adapter architecture originally proposed by Houlsby *et al*. (2019) [5], as described in Section of the text.

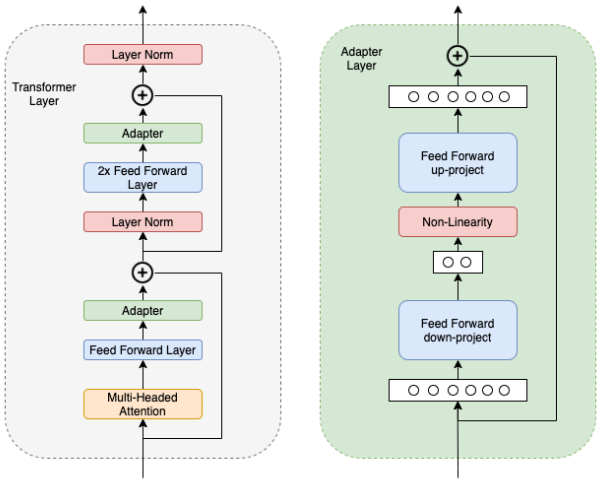

Figure 3. The adapter architecture proposed by Houlsby *et al*. (2019). The left panel shows where adapter modules are introduced within a Transformer network. The right panel shows the sub-layers in an adapter module.

B. Datasets

Datasets used in the project can be found here: https://gtvault-my.sharepoint.com/:f:/g/personal/apradhan48_gatech_edu/Em-EethF0HlDtsOP_suNP0wBSHBKLvlhxsmeQa_-VSgYdQ?e=K1iq9z