Project Report : CS 7643

Nohelia Merino Suasnabar
Georgia Institute of Technology
North Ave NW, Atlanta, GA 30332
nmms3@gatech.edu

Tapos Pal
Georgia Institute of Technology
North Ave NW, Atlanta, GA 30332
tpal3@gatech.edu

Alexander Tam
Georgia Institute of Technology
North Ave NW, Atlanta, GA 30332
atam6@gatech.edu

I. Introduction/Background/Motivation

The paper "Don't Stop Pretraining: Adapt Language Models to Domains and Tasks [1] showed that it is not enough to train a model on an enormous corpus of text. Instead, the authors showed that further training on domain and task specific datasets greatly improve performance of models. The downside of this that for each task needs to be specifically trained on its domain/task specific data. Full fine-tuning also makes it easy to fall under overfitting of the adaptation corpus. This pretraining method also does not transfer across tasks in the same domain. How can we make this process less computationally expensive and time-consuming?

Adapter transformer models are much smaller than fully fine-tuned transformer models. Thus, they can be easily shared and deployed. However, the most interesting aspect of adapter transformers is that their architecture allows them to share a set of parameters across tasks. Adapter transformers can also be fuse or stacked together. Adapters are modular, composable [2] and easily shareable. In the future, an updated adapter (order of MB) could be deployed to update a task-specific model in the field instead of an entirely new model checkpoint (order of GB). The goal this project is to experiment with adapter transformers and take advantage of their properties to replicate the results of [1].

II. Approach

Our approach was taking the code of [2] as a starting point, along with the code of the framework Adapter Hub [3] and the documentation and code of the framework HuggingFace [4]. On top of a chosen base file to run the model the adapter fusion architecture was implemented. For the data to work, it needed to be pre-processed so that it can fit the accepted format of the Trainer API. First the data labels had to be mapped from text labels to numeric integer unique ids, after that the format accepted by the Trainer API was a tensor of floats, as a result the list of ids had to be converted from integer to floats. One challenge we anticipated to have been the environment computational capacity, since for simplicity we picked Google Colab Pro. Another challenge expected was a slow learning curve since the average experience of the team with NLP is almost only the lessons of the course and none of the members have ever worked with these Frameworks.

A. Adapters

Adapter [3] is a transfer-learning approach, they are an alternative to full finetuning. They are modules inside the transformers, that can allow a base transformer/model to keep being pre-trained for specific domain tasks. Adapters tackle the problem of the need of full finetuning of transformer for specific downstream tasks which requires storing a copy of the fine-tuned model for each task, and transformers requires large storage of about 2.2Gb. This makes transformer-based models more scalable.

B. Adapter Fusion Transfer Learning

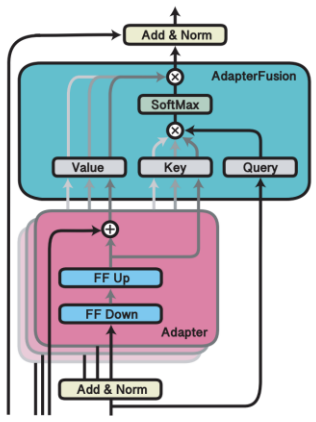


Fig. 1. Adapter Fusion architecture. [1]

It is an approach of the adapters that tackles [1] catastrophic interference when it comes to multi-tasking learning by proposing to train task-specific weights (adapters) for each individual tasks, so that no information can be forgotten as pre-trained weights are not overwritten, and tasks also do not hamper each other since they have specific delegated parameters. After this, the task specific adapter representations are combined subsequently. Assuming that the adapters are interchangeable, it proposes a dynamic attention mechanism, where given a pool of adapters, will automatically learn to weight the representation of the respective adapters which are available.

As shown in Fig. 1. [1] the input is defined to each of the adapters as the query (Q) and the output as the key (K) and value (V). The dot product of K.Q is passed into a softmax layer, which learns to dynamically weight every single adapter, and at each transformer layer we have a set of adapters available, and, given downstream task, the model tries to automatically identify which adapters are most useful for.

We chose this as part of our approach because we wanted to combine the knowledge from multiple source tasks to perform better on a target task.

C. *Training New Single Task Adapters*

New single adapters [3] are trained in the same manner as full finetuning of the model. Nevertheless, contrary to full finetuning, the pre-trained weights are fixed and only the adapter weights and the prediction head are trained. The adapter weights are encapsuled within the transformer weights, forcing them to learn compatible representations across tasks.

D. *Adapters+Fine-tuning*

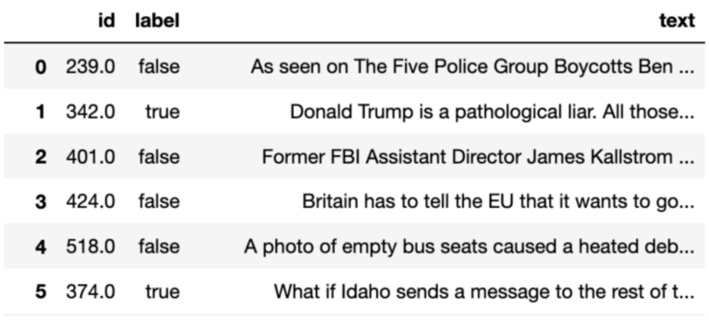
We aim to use Adapter-Hub adapters as well as train our own adapters to perform fine-tuning of the TAPT' tasks in the paper. Adapters [3] are a small set of extra newly initialized weights at every transformer layer, which are then trained during fine-tuning, while the pre-trained parameters of the large model are kept frozen/fixed.

E. *Why do we care?*

Because of adapters ability to be "attached" to the end of network, this saves time and space. When training adapters, only the weights of the adapters change. The pretrained network they are attached to stays static. Since only the weights of the adapter are tuned, time is saved by reduced the number of parameters that need to be updated and space by the number of weights that need to be saved. These factors make adapters way easier to use than full fine tuning. If we can demonstrate that are contributing to reach a new standard for NLP tasks.

III. Experiments and results

Data in this project is taken from AllenNLP's S3 bucket for their paper [2]. Since we are using the same dataset as the indicated paper, we will be able to directly compare our results to theirs. The goal of this project is to use adapters as an alternative to full finetuning transformers and achieve comparable results to their task adaptative pre-training experiments (TAPT). For the Experiment I we will ignore the mask language modeling step in the paper [2] and directly finetune for the downstream classification tasks (raw state of the datasets as observed in Fig. 2.) , but for Experiment II and III we will consider it while training new adapters.



	id	label	text
0	239.0	false	As seen on The Five Police Group Boycotts Ben ...
1	342.0	true	Donald Trump is a pathological liar. All those...
2	401.0	false	Former FBI Assistant Director James Kallstrom ...
3	424.0	false	Britain has to tell the EU that it wants to go...
4	518.0	false	A photo of empty bus seats caused a heated deb...
5	374.0	true	What if Idaho sends a message to the rest of t...

Table 1. Hyperpartisan dataset before pre-processing.

Task types for this paper are altered to from their original modeling task to text classification for easier comparison. Most of the following results will utilized datasets with the least amount of data, as that will lessen the time between experiment iterations.

These are the results from Table 5 of the "Don't Stop Pretraining Paper." Our goal is to achieve comparable results for task adaptative pretraining (TAPT) experiments. The metric they use for all the tasks isa1_macro except for CHEMPROT and RCT, which use F1_micro.

Domain	Task	Roberta	TAPT
BioMed	CHEMPROT	81.9_1.0	82.6_0.4
	RCT	87.2	87.7
CS	ACL-ARC	63.0	67.4
	SCIERC	77.3	79.31
News	HyperPartisan	86.6	90.4
	AGNews	93.9	94.5
Reviews	Helpfulness	65.1	68.5
	IMDB	95.0	95.5

Table 2. Paper Results for TAPT.

A. *Experiment I: Fine-tuning on downstream tasks using Adapter-Hub adapters*

A key feature of adapter transformers is that they can be stacked, and their outputs are fused together using a softmax layer. Here we train on the CHEMPROT dataset, fine tune 3 different pretrained adapters over 3 epochs. We also run those 3 adapters fused together for the same number of epochs to get the following results:

Domain	Task	sts/qqp @ukp	nli/multinli @ukp	nli/qnli @ukp	Fusion
Bio Med	CHEMPROT	0.44738	0.452759	0.34318	0.477155
	RCT	0.44446	0.36094	0.3831	0.52677

Table 3. Fusion Adapter Experiment - Adapters Trained to 3 Epochs, F1_micro, learning rate = 1e-5, batch size = 4

The result of this experiment is inconclusive. We show that Fusion does have benefits. For the CHEMPROT dataset, the fusion is close to the top performer while for RCT dataset fusion is outright the best result. Different pretrained adapters that match our task may give better results and would be a good area for future investigation. Given more time, this experiment would be extended to more domains/tasks and run for more epochs.

Domain	**Task**	**Fusion**
BioMed	**CHEMPROT**	**0.81739**

Table 4. Fusion, Best Result, 100 epochs

Training CHEMPROT using the fusion adapter to 100 epochs (number of epochs used in [2]) we achieve results close to "Don't Stop Pretraining." This result combined with what we see in Table 3 shows that even with adaptive transformers, pretraining on domains is still beneficial.

B. *Experiment II: Fine-tuning with newly trained adapters for the taks in the domains of Bio-Med and CS*

In this experiment we perform fine-tuning on models that are pretrained first with newly trained adapters for the tasks of the Bio-Med and CS domains. We chose to use this experiment because we wanted to experiment how the performance would be impacted if we perform finetuning with custom adapters that belongs to the same downstream task domain instead of just the ones in Adapter-Hub that are not necessarily domain-specific adapters.

For this experiment we first trained new single adapters for the tasks RCT-500, Chemprot, Sciie (SCIERC) and Citation-intent (ACL-ARC) using the run_mlm.py file from the Adapter-Hub framework, this is because the base model used by the paper to do the fine-tuning was roberta-base, since this is an MLM model, we are expecting to train a new single adapter with for specific task data. Since the data for these four tasks are meant to be classification tasks, run_mlm.py adds a 'special_token_mask' just like in the paper, for this we set a mask probability parameter of 15%, this is the probability with which to (randomly) mask tokens in the input. From this 15%, 80% of the time, the masked input tokens are replaced with a mask token ([MASK]) and 10% of the time the masked input tokens are replaced with a random word.

Let us start analyzing the results for RCT-500. We did not use the large dataset version of RCT due to the big time it needed to be trained in our environment. One key factor to point it out in the data for RCT-500 is that the training set is 99 KB, and the testing set is 5.8 MB. This is because RCT-500 was actually used for Human Curated-TAPT in the paper. We did not perform Human Curated-TAPT but rather we use the RCT-500 data for a standard TAPT. Having a small dataset does not allow the model to generalize well. On the other hand, it is preferred to increase the batch size rather than decrease the learning rate to achieve better convergence, however as pointed before, we had a computational limitation when it came to trained with batches equal or greater than 16, in contrast in the paper they use 256 or 2058. After increasing the batch size from 12 to 15 the F1 score increased dramatically from 73.32 to 76.74. So, because of this we can claim that these two factors were the main cause of not achieve an on-par performance as the paper.

Now let us analyze the results for Chemprot. We achieved on-par performance with the paper using adapter fusion in both the RCT-500 adapter with the Chemprot adapter and using the Chemprot adapter alone too. This means the batch size limitation did not affect the performance. It was again, however, important to use the same or similar weight decay parameters as in the paper, since otherwise the overfitting impacted the test results. The same for the Adam parameters, when we did not use them, the model did not converge well for the same number of epochs, needing more epochs to achieve the same results.

In the case of the Sciie task, on-par performance with the paper was also achieved when using adapter fusion in both the Sciie adapter with the citation-intent adapter and using the Sciie adapter alone too. Increasing the batch size for this task did not help nor harmed the performance. So, for this task we do not expect that increasing the batch size will help.

For the case of the Citation-intent task, we experimented with doing fine tuning with different new trained adapters. They were, however, trained for the same task (data) but with slightly parameter tweaks such as change on the number of epochs or the batch size, and the results of these was getting on-par performances in the downstream tasks for all these adapters. More experimentation is needed when it comes to tuning parameters to train a new MLM task adapter. We achieve on-par performance during out experimentation for this task was achieved while using multi-task adapter fusion using the adapters for the Citation_intent and the Sciie adapters (each adapter represents a new adapter trained on their respective task training data). This is not totally expected since as we mentioned before we had the limitation of only being able to train a number of epochs less than 16. However, we used similar Adam and regularization parameters as the paper, so from that side, comparable results were also expected.

One more point is that we would need to experiment further more with others adapter configurations, since we only experimented with the default configurations owned by the newly trained adapters, and we also experimented with the Pfeiffer configuration, and across all the tasks, in general, the experiments with Pfeiffer returned slightly better results.

Domain	**Task**	**Roberta**	**TAPT**	**Experiment 2**	
				New Adapter Only	**Multiple Adapters**
BioMed	**CHEMPROT**	**81.9_1.0**	**82.6_0.4**	**81.49**	**82.44**
	RCT	**87.2**	**87.7**	**79.24**	**76.74**
CS	**ACL-ARC**	**63.0**	**67.4**		**66.35**
	SCIERC	**77.3**	**79.31**	**80.93**	**79.2**

Table 5. Best results for Experiment II. The numbers are F1 scores, for the BioMed tasks F1-micro average was used and for the CS tasks F1-macro average, this is because the paper results also used these metrics. Hyperparameters can be found in Table 9.

C. *Experiment III: Fine-tuning with newly trained adapters for the tasks in the domains News and Reviews*

For this experiment, we have done few things. We first created new adapters using Hyper Partisan and Ag News datasets of News domains. Then we ran experiments using those newly created adapters including cross-domain tasks. Also, we fused our created adapters with adapters mentioned in Pfeiffer et al. [1]. We choose to do this experiment as we wanted to understand how newly created adapters works on downstream tasks in the same domain and cross-domain settings, then also the impact of fusing with multiple adapters of multiple source tasks.

New adapter creation process is same as mentioned above in Experiment II by running run_mlm.py and with a mask probability of 15%. After that we used run_multiple_choice_adapter_fusion.py (initially it was named run_multiple_choice.py) to run the experiment using different hyperparameters to tune it generate comparable results. Parameters we were most focused for this experiment were training batch size, learning rate, training epochs.

Domain	**Task**	**Roberta**	**TAPT**	**Experiment 3**	
News	**HyperPartisan**	**86.6_0.9**	**90.4_5.2**	**New Adapter Only**	**Multiple Adapters**
				90.76	**98.46**
	AgNews	**93.9_0.2**	**94.5_0.1**	**89.97**	
Reviews	**Helpfulness**	**65.1_3.4**	**68.5_1.9**	**85.54**	
	IMDB	**95.0_0.2**	**95.5_0.1**	**92.96**	

Table 6. Best Results from Experiment III. The numbers are F1 Scores. Used training epoch 45 for Hyper Partisan News, 2 for other datasets. Other than hyperpartisan news, all other scores are using multiple adapters.

Let's start with analyzing HyperPartisan News task as we did most experiment with it and used those findings to start new experiments for other tasks. We chose this dataset as it contains 515 labeled data records for training which is small enough to allow us run multiple experiments in shorter time with higher training epochs. We created two different adapters for News domain using both datasets. In Experiment II we also created multiple adapters using different datasets for Biomed and CS domains. We found that adapter created from chemprot data worked best. We think this is because chemprot data has relational classification with 13 classes. Since HyperPartisan data binary relational classification, so domain overlapping helped to generalize the data more and produced a better result. Also, AgNews adapter worked better than HyperPartisan as first one has more trainable records to train from. We achieved comparable results to TAPT by only using our trained adapters.

Adapters	**Best Score for Hyper Partisan Task**
HyperPartisan	**70.46**
AgNews	**85.26**
Chemprot	**90.76**

Table 7. Results for Hyper Partisan Task from Experiment III. Using different single adapter from different datasets. Hyperparameters used LR = 2e-5, training_epochs = 45, batch size = 12, max sequence length = 512

After that we added fused with more adapters ("nli/multinli@ukp", "sts/qqp@ukp", "nli/scitail@ukp" [1]) with our CHEMPROT adapter. We got even better result using this setup. As mentioned in [1], we are leveraging knowledge from multiple tasks a combining them to solve our binary classification problem. We got comparable results to TAPT using the same method and only using 2 training epochs. We did not have enough time to train with higher number of epochs as it is a bigger dataset and training takes long time.

Now using above knowledge, we tried using same method for the domain “Reviews”. Even though we are using adapters from different domains. We achieved much better results for Helpfulness classification problem and a comparable result for IMDB review with only two training epochs. As both are binary classification problems, and we are combining knowledge from multiple tasks, we think we are getting better results here.

IV. Other Sections

A. *Limitations and Challenges*

We list the main limitations/challenges that we faced while working on this project:

1) Limited enviroment resources. We used Google Colab Pro to do the experimentation, however any of the experiments were able to use a batch size equals or bigger than 16, since Goggle Colab Pro would run out of memory. This certainly had a significant impact on our results for tasks such as RCT-500.

2) Sparsed documentation of frameworks HuggingFace and AdapterHub. HuggingFace and AdapterHub are two great frameworks, that certainly tackle some of the Transformers’ Transfer Learning challenges. The way it is documented however it sometimes confusing due to lack of a documentation clear enough to easily figure itt out how they work, the good thing however is that HuggingFace had a good community that helped us resolved some of the errors we face while implemented the code.

Student Name	**Contributed Aspects**	**Details**
Nohelia Merino	**Implementation and Analysis**	- **Implemented code for adapter fusion architecture.** - **Figure it out a pipeline to train single adapters by using Adapter Hub framework.** - **Lead and instructed the team for the understanding and work division of the project.** - **Experimentation with adapter fusion for newly trained adapter for the datasets for the BioMed and CS domains' tasks including training of the new single adapters and using them to perform fine-tuning on the downstream task. (Experiment II)**
Tapos Pal	**Implementation and Analysis**	- **Prepare the cloud environment including the needed libraries to execute the experiments** - **Code to label data needed for pre-processing.** - **Experimentation with adapter fusion for newly trained adapter for the datasets for the News and Reviews domains' tasks including training of the new single adapters and using them to perform fine-tuning on the downstream task. (Experiment III)**
Alexander Tam	**Implementation and Analysis**	- **Code to ingest JSONL data into Datasets library format** - **Experimentation with adapter fusion with adapter-hub adapters for the datasets for the RCT and Chemprot task to perform fine-tuning on the downstream task. (Experiment I)**

Table 8. Contributions of team members.

V. Conclusions

Given the experimentation done in this project, more supportive proof is found that the adapters will replace the full finetuning for transformers. We obtained on-par performance using adapters (Experiments II and III) with the one using full finetuning in the paper [2]. This means we can finetune transformers without the space hurdles what it demands, making them scalable.

VI. Work Division

Summary of contributions are provided in Table 8.

References

[1] Pfeiffer, J., Kamath, A., Rücklé, A., Cho, K., & Gurevych, I. (2020). Adapterfusion: Non-destructive task composition for transfer learning. arXiv preprint arXiv:2005.00247.

[2] Gururangan, S., Marasović, A., Swayamdipta, S., Lo, K., Beltagy, I., Downey, D., & Smith, N. A. (2020). Don't stop pretraining: adapt language models to domains and tasks. arXiv preprint arXiv:2004.10964.

[3] Pfeiffer, J., Rücklé, A., Poth, C., Kamath, A., Vulić, I., Ruder, S., ... & Gurevych, I. (2020). Adapterhub: A framework for adapting transformers. arXiv preprint arXiv:2007.07779.

[4] Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., ... & Rush, A. M. (2019). Huggingface's transformers: State-of-the-art natural language processing. arXiv preprint arXiv:1910.03771.

I. Project Repository

We use base code from [2] and [3]. Our project repository is: https://github.gatech.edu/nmms3/gatech_deep_final

II. Dataset Information

The dataset was extracted from the S3 links from the Don't stop pre-training repository. The links are available in [2]: https://github.com/allenai/dont-stop-pretraining/blob/master/environments/datasets.py

III. Our New Trained Adapters

The new trained adapters in this project can be found here: https://drive.google.com/drive/folders/1-5fdDL6FcwY3xwNJD4dBAFSNFg7vgPWb?usp=sharing

IV. Experiments Parameters

A. *Experiment II – Best models' parameters*

Task	Parameters	F1	Task	Parameters	FI
RCT-500	# Experiment 20 !python3 run_multiple_choice_adapter_fusion.py \ --do_train \ --do_eval \ --data_dir data/rct-sample_ \ --max_seq_length 512 \ --per_device_train_batch_size 12 \ --gradient_accumulation_steps 2 \ --learning_rate 1e-4 \ --num_train_epochs 30 \ --output_dir results/rct-sample-new-adapter_7/ \ --task_name mlm \ --do_predict \ --model_name_or_path roberta-base \ --adapter_1 results/adapters/rct-sample/mlm \ --avg_type micro \	**79.24**	**Sciie**	# Experiment 4 !python3 run_multiple_choice_adapter_fusion.py \ --do_train \ --do_eval \ --data_dir data/sciie_ \ --max_seq_length 512 \ --per_device_train_batch_size 12 \ --gradient_accumulation_steps 1 \ --learning_rate 4e-5 \ --num_train_epochs 40 \ --output_dir results/sciie_1/ \ --task_name mlm \ --do_predict \ --model_name_or_path roberta-base \ --adapter_1 results/adapters/sciie/mlm \ --per_device_eval_batch_size 12 \ --weight_decay 0.12 \ --adam_beta1 0.9 \ --adam_beta2 0.95 \ --adam_epsilon 5e-4 \ --evaluation_strategy epoch \ --seed 1 \ --avg_type macro \	**80.93**
Chemprot	# Experiment 19 !python3 run_multiple_choice_adapter_fusion.py \ --do_train \ --do_eval \ --data_dir data/chemprot_ \ --max_seq_length 512 \ --per_device_train_batch_size 12 \ --gradient_accumulation_steps 1 \ --learning_rate 4e-5 \ --num_train_epochs 40 \ --output_dir results/chemprot_rct-sample_chemprot_rct/ \ --task_name mlm \ --do_predict \ --model_name_or_path roberta-base \ --adapter_1 results/adapters/rct-sample/mlm \ --adapter_2 results/adapters/chemprot/mlm \ --per_device_eval_batch_size 12 \ --weight_decay 0.12 \ --adam_beta1 0.9 \ --adam_beta2 0.95 \ --adam_epsilon 5e-4 \ --evaluation_strategy epoch \ --seed 1 \ --avg_type micro \	**88.44**	**Citation-intent**	# Experiment 8 !python3 run_multiple_choice_adapter_fusion.py \ --do_train \ --do_eval \ --data_dir data/citation-intent_ \ --max_seq_length 512 \ --per_device_train_batch_size 15 \ --gradient_accumulation_steps 2 \ --learning_rate 4e-5 \ --num_train_epochs 70 \ --output_dir results/citation-intent_4/ \ --task_name mlm \ --do_predict \ --model_name_or_path roberta-base \ --adapter_1 results/adapters/citation-intent/mlm \ --adapter_2 results/adapters/sciie/mlm \ --per_device_eval_batch_size 15 \ --weight_decay 0.0001 \ --adam_beta1 0.9 \ --adam_beta2 0.97 \ --adam_epsilon 5e-5 \ --evaluation_strategy epoch \ --seed 836 \ --avg_type macro \ --load_best_model_at_end \	**66.35**

Table 9. Parameters of best model from Experiment II.

B. *Experiment III - Best models' parameters*

Task	Parameters	F1	Task	Parameters	FI
HyperPartisan	#6. Got the best result using new adapter combining with adapterhub adapters !python3 run_multiple_choice.py \ --do_train \ --do_eval \ --data_dir data/hyperpartisan_news_ \ --max_seq_length 512 \ --per_device_train_batch_size 12 \ --gradient_accumulation_steps 1 \ --learning_rate 2e-5 \ --num_train_epochs 45 \ --output_dir results/hyperpartisan_news_adapterhub_adapters10/ \ --task_name hb3 \ --do_predict \ --model_name_or_path roberta-base \ --adapter_1 results/adapters/ag/mlm \ --load_best_model_at_end \ --seed 5	**98.46**	**Helpfulness**	# V2-5: New adapter hyperpartisan on amazon dataset !python3 run_multiple_choice.py \ --do_train \ --do_eval \ --data_dir data/amazon_ \ --max_seq_length 512 \ --per_device_train_batch_size 8 \ --gradient_accumulation_steps 1 \ --learning_rate 2e-5 \ --num_train_epochs 2 \ --output_dir results/ hyperpartisan_news-new-adapter11/ \ --task_name mlm\ --do_predict \ --model_name_or_path roberta-base \ --adapter_1 results/adapters/hyperpartisan_news/mlm \ --load_best_model_at_end \	**85.72**
Ag News	V2-Ag: New adapter hyperpartisan on ag news dataset !python3 run_multiple_choice.py \ --do_train \ --do_eval \ --data_dir data/ag_ \ --max_seq_length 512 \ --per_device_train_batch_size 8 \ --gradient_accumulation_steps 1 \ --learning_rate 2e-5 \ --num_train_epochs 2 \ --output_dir results/ hyperpartisan_news-new-adapter9/ \ --task_name mlm\ --do_predict \ --model_name_or_path roberta-base \ --adapter_1 results/adapters/hyperpartisan_news/mlm \ --load_best_model_at_end \	**89.97**	**IMDB**	#V2-4: New adapter hyperpartisan on imdb dataset !python3 run_multiple_choice.py \ --do_train \ --do_eval \ --data_dir data/imdb_ \ --max_seq_length 512 \ --per_device_train_batch_size 8 \ --gradient_accumulation_steps 1 \ --learning_rate 2e-5 \ --num_train_epochs 2 \ --output_dir results/ hyperpartisan_news-new-adapter10/ \ --task_name mlm\ --do_predict \ --model_name_or_path roberta-base \ --adapter_1 results/adapters/hyperpartisan_news/mlm \ --load_best_model_at_end \	**92.96**

Table 10. Parameters of best model from Experiment III.