Transformers vs Adapter-boosted Transformers: A Comparative Study

Yash Kumar Singh
Georgia Institute of Technology
ysingh63@gatech.edu

Sudeshna Dash
Georgia Institute of Technology
sdash40@gatech.edu

Tejaswin Kopparty
Georgia Institute of Technology
tkopparty3@gatech.edu

Abstract

Transformers have provided a revolutionary breakthrough in the field of NLP whose attention based mechanism has been the basis of achieving stellar state-of-the-art (SoTA) results for various tasks. However, these models today consist of hundreds of millions and sometimes billions of parameters which makes storing the models, fine-tuning to a task and utilizing them on scale very expensive. Use of adapters as bottleneck layers has been encouraged in recent years to avoid the effort of fine-tuning the entire model. In this report, we attempt to do a comparative study of a SoTA task-adapted transformer model to an adapter-boosted variant of the transformer against a baseline model in terms of memory and performance efficiency.

Keywords: Transformer, Adapter, Language Model, Text Classification, Task Adaptive Pre-training

1. Introduction

Language models (LM) in today's day and age are being pretrained on large textual datasets which belong to a diversity of domains, including STEM as well as Non-STEM domains. Striking examples of the same are models such as BERT by Google AI, RoBERTa by Facebook [8], etc. RoBERTa as an LM is trained upon over 160GB of text data which comprises of sources ranging from news articles, scientific journals, encyclopedia texts to various web content as well. Exposing models to such a wide variety of texts thus benefits the language model in learning vivid representations in its lower levels which can then be utilized to perform a variety of tasks. However, a salient question that arises in this context is that would such a model be efficient in performing any given prediction task just by having this plethora or pretraining?

[5] in their paper highlight the importance of pretraining these deep models based on two aspects – domain and tasks. While *domain* refers to the textual context of where the task would belong or would stem from, *task* refers to the prediction or classification item that we would want the model to perform. An example for the same can be having a pretrained model, say BERT where sentiment classification for news articles is required. The domain of the case would thus become news articles whereas the task would become sentiment classification which would assign sentiment labels to each given article. [5] prove that pre-training the model on a domain and task data both gives a higher accuracy in terms of performance. However given the depth of the model and the billions of parameters, it becomes a challenge to utilize these models on task-specific use cases as both storing and training the model itself takes enormous resources.

A solution to the problem came with the advent of adapters [6] which were introduced as an alternative lightweight fine tuning strategy as compared to fully changing the weights and biases of a complex deep model such as a transformer. Consisting of a small set of newly initialized weights inserted in between the layers of a fixed or frozen transformer, these weights when fine tuned, provide an efficient mode of gradient flow and performance. Use of these lightweight adapters has also been shown to provide performance efficient results across multiple tasks [10].

While the use of adapters has proliferated in the recent years, sharing adapters or utilizing pre-trained adapters for multiple tasks was still a challenge. AdapterHub [9] provided the means for developers easily train and share such trained adapters with the community, basing their adapter upon multiple popular transformers such as BERT, RoBERTa and XLM-(R). And with minimal memory taken to store these pre-trained adapters, the study and analysis of such models as well as their pertinent tasks has been made even more easier.

Our study attempts to dive deeper into the context of transformers and adapters by selecting specific tasks and understanding whether use of adapters actually lend better benefits in terms of performance and memory

when compared to a fine-tuned transformer as well as how much of a difference does an adapter boosted transformer bring when compared against a regular transformer model. A deeper analysis in this context would shed more light on which approach would be salient for what task – both for the academic as well as industrial community as data would keep increasing in coming time but our computation methods would lag behind [2]. We take the pre-trained RoBERTa model for the purpose and identify two tasks – Citation Intent Identification (CI) and Relational Classification (RCT). Our contributions for the paper include

- creating a new adapter for a non-adapted transformer to compare and contrast performance against a fine-tuned RoBERTa model
- creating a new adapter for new tasks and doing a comparative study for the findings against a baseline as well as fine-tuned RoBERTa model

2. Approach

Given the multiple comparison scales the study dives into, there are a number of models both baseline and tuned that are considered along with two tasks as talked about in the section above. Utilizing python as the medium and Pytorch as the chief Deep Learning framework, [7] was used for majority of the models and adapters as the same serves as a great hub for ML and DL models as well as a community of enthusiasts. We aimed at first reproducing the baseline results as presented by [5] for RoBERTa model for both the classification tasks. The same would then be fine tuned based on a hyperparameter tuning regime against the tasks to adapt better to these tasks against which the same metrics are calculated. Custom adapters are then written for a base RoBERTa model for each task respectively which is trained against the given task to finally tally the results.

2.1. Language Model – RoBERTa

Using the pretrained model for the language model RoBERTA [8] via HuggingFace APIs, we were able to fetch the actual transformer weights and biases for the case and were able to utilize the same for the study. Utilizing longer sequences and training with bigger batch sizes, the base model has performed well with 0.88 accuracy on Glue benchmark [8]. The same was believed to have a strong transfer to the text tasks (CI and RCT) which were considered for the purpose of the study [6]. The model used as is, henceforth mentioned **RoBERTa Base** was used to set the benchmark baseline for the metrics for both the tasks.

A gridsearch was done over a set of selected parameters to minimize on the resources involved in running the same while garnering a good performance. The hyperparameters included Learning Rate, Weight Decay and Epoch. Extending the deep learning ideas of convergence and ranking it up against the resources utilized, a configuration suitable for both the tasks was identified which would be called as the or **Fine-Tuned Model** or **Task Adapted Pre-Trained (TAPT) Model**.

2.2. Data & Classification Tasks – CI & RCT

Though RoBERTa LM is trained via masking in a semi-supervised fashion, to analyse the predictive power we entail a bunch of tasks for the model and thus evaluate its performance. Borrowing the classification tasks as well as the relevant data from [5], we take two specific STEM domain classification tasks – Citation Intent Identification (CI) and Relation Classification (RCT).

Having 1688 excerpts of in-text citations from over 250 scientific journals, **CI** attempts to classify the intent of the given citation in the journal or the research paper with 6 distinct labels like ”Background”, ”CompareOrContrast” and so on depending on the context in which the citation is provided in the current paper or journal. The task is well suited for the model as the domain of data that RoBERTa is trained upon has an explicit corpus of scientific papers.

In the same domain of academic papers, the task of RCT attempts to understand which section of a paper does the given sentence belong to. With 7 class labels such as ”Motivation”, ”Conclusion”, ”Methods”, ”Results” and so on, the dataset contains over a 180040 samples. However, given the amount of data and the expected time to train the models over the same, we resorted to using a reduced version of the same dataset as made available by [5] wherein the task setup has only 500 samples for training whereas the evaluation is done over a set of 30135 samples.

Table 1. Specifications of Datasets Used

Task	Train Samples	Test Samples	Labels
CI	1688	139	6
RCT	500	30135	7

2.3. Adapter – CIAdapter & RCTAdapter

Shitfing focus to the lightweight adapters now, [1] built by [9] provides a means to both use pre-trained adapters for given RoBERTa model as well as create an adapter for a given model from scratch. Similar adapters have been trained on a set of tasks which are

also available for use both of HuggingFace as well as AdapterHub. For the purpose of our study, we created an adapter of our own which had a small configuration of 2 feed forward fully connected layers with a normalization layer [3] in between. The same created a lightweight configuration which occupied lesser than 1mb of space which was then trained for both the CI and RCT tasks respectively.

Though multiple configurations of adapters are provided in the library (PfeifferConfig, HoulsbyConfig, etc.), we chose to create our own new configuration which is free from any pre-training over any dataset. The sole benefit here however was that the adapters were easy to create and were adaptible to the models we fetched from HuggingFace API.

3. Experimental Setup and Results

We trained RoBERTa with predefined hyper-parameters and our tuned parameters. The training was carried out for Citation Intent Identification (CI) as well as the Relation Classification (RCT) Task for the datasets discussed above.

The training for the CI task was done for 10 epochs and 1070 steps. Similarly, the training for the RCT task was done for about 10 epochs and 1020 steps. Both of these datasets were run on our local machines for the default parameters for a baseline as well as the tuned hyperparameters. Each of these tasks was then trained using the custom adapter with RoBERTa as the baseline. The adapter is then put through a supervised fine-tuning of the parameters for each of these two tasks. As RoBERTa provides a single LM to adapt to various domains and tasks, it proves to be an excellent baseline.

The pre-training of the RoBERTa model definitely improved the tasks but took a lot of computational power such as system memory, GPU, etc. But, adapters took comparatively a very less time to adapt to the tasks and got nearly similar results than that of a fine-tuned RoBERTa model. The results are shown in Table 2 below

Table 2. Results Table

Model	Data	Accuracy	Train Loss	Test Loss
Roberta-base	CI	0.6043	1.287	1.1254
	RCT	0.62731	0.634	1.2401
TAPT Model	CI	0.7626	0.206	1.082
	RCT	0.71291	0.741	1.3103
Adapter Model	CI	0.6931	1.138	0.9875
	RCT	0.6526	1.399	1.23

Figure 1. Training Loss

Table 3. Runtime Table (in Minutes)

Model	Data	Train Runtime	Test Runtime
Roberta-base	CI	324.99542	41.9768
	RCT	170.29095	49.159095
TAPT Model	CI	318.528183	40.1636
	RCT	140.23928	52.8671
Adapter Model	CI	50.535156	38.9352
	RCT	139.64821	29.61313

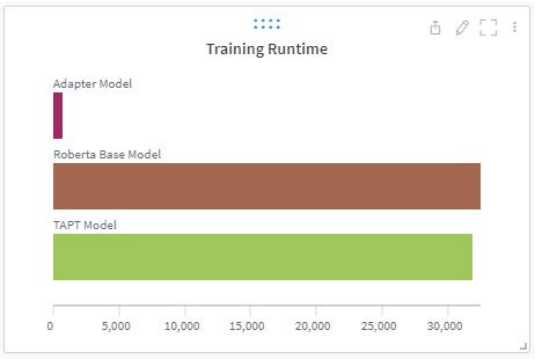

Figure 2. Training Runtime

3.1. Input and output formatting

All the models expected the inputs to be in a tokenized format with specific labels such as input_ids, label and attention_mask. This format was generated using Tokenizer classes which takes the input as text and the labels as an integer value and tokenizes the data into the tensor format[4]. This tokenizer basically converts the text into tokens(numbers). We also used the DataCollator for padding the data. This method formats all the inputs into exact same size by adding padding of 0 values. These inputs were then pre-trained in the model.

The outputs from the model also came in a tensor format which was further used to compute the predictions.

3.2. Task-Adaptive Pretraining

Task-Adaptive Pretraining was carried over both CI and RCT datasets using baseline parameters and then fine-tuned parameters. The size of training dataset for CI was about 1688 records and for RCT was about 500 records. With an evaluation strategy of using Epochs, the dataset went through a pretraining phase with base as RoBERTa. This pretraining ran at a speed of 0.53 samples per second. The training loss decreased from 0.8 to 0.1 over each steps – showing a reduction in estimation and modelling errors and thus rapid learning. There were total of 1080 steps for CI dataset and hence it took a long time and more GPU power to run. Comparatively, the fine-tuned model gave a much better accuracy than the RoBERTa base model. The results are shown in Table 2, Table 3 and Table 4. Please refer to Figure 2, Figure 3 and Figure 4 for better insights.

3.3. Adapter based Transformers

With RoBERTa as the base, we preprocessed the dataset to cater to the standard structure of Adapters as per the framework defined by [9]. We used a custom adapter and added a binary classification head for the tasks. Thus, the weights and biases for the model were frozen and only the adapter weights get trained with each forward pass. The total steps for CI and RCT were 318 and 96 respectively. Due to its 2 layer lightweight nature, the training ran at a speed of 2.18 steps per second and evaluation ran at a speed of 4.176 samples per second. The accuracy suffered a little bit more than the fine-tuned pretraining model but the computational power was an advantage.

3.4. Computing Accuracy

The model results were evaluated using CrossEntropy loss function. This function takes the actual label and the predicted label to calculate the loss. The average training loss for TAPT model is 0.2067 whereas the RoBERTa-base model loss is 1.163. The average evaluation loss for TAPT model is 0.3213 whereas the RoBERTa-base model loss is 1.125. Hence, its proved that the model doesn't overfit. We even used TextClassificationPipeline to test our own data and the model predicted the data appropriately, hence proving that the model is well generalized. The results are shown in Table 2 and Figure 5.

Table 4. Average Computational Power Usage

Model	CPU Utilization	Memory Use	Accuracy
RoBERTa Base	61%	1213 MB	0.60432
TAPT Model	76.34%	4431 MB	0.76259
Adapter Model	98%	5437 MB	0.69309

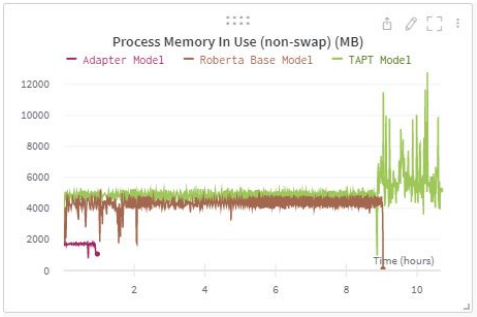

Figure 3. Memory Utilization

3.5. Results

We compared the results between RoBERTa-Base, TAPT and Adapter based models. The base model showed very poor performance in comparison to the TAPT and Adapter based models. TAPT models were expensive in terms of computational power (Figure 3 and Figure 4) but the accuracy and precision was very good. TAPT models took lesser time than the base model but in comparison to the Adapters, the TAPT model proved expensive (Figure 2). In terms of memory, RoBERTa base model took the most memory utilization than the Adapter or TAPT model (Figure 3, Table 4). The Adapter model lacked a bit in accuracy compared to the TAPT model but the lesser use of computational resources made it easier for training and parameter hypertuning (Figure 5). Out of all the three approaches, the TAPT model proved to be the best in accuracy

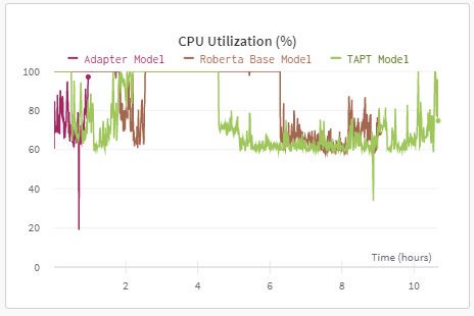

Figure 4. CPU Utilization

followed by the Adapter model and then the RoBERTa base model. The results might improve for the Adapter with much more data. These comparisons show that Adapter based models are a good and cheap solution for any such tasks and can be a great alternative for the TAPT models.

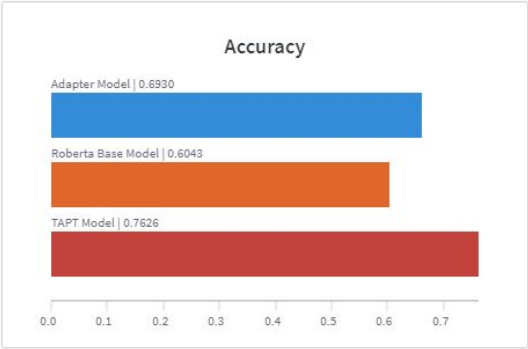

Figure 5. Accuracy

4. Experience

4.1. Challenges

We read about time and space complexity in lower-grade computer science lectures but Deep Learning was one aspect where we truly felt what these resources meant. While there were multiple issues we anticipated with respect to lack of proper knowledge in this fresh and new domain of transformer architectures, it was truly the time taken to compute results that was the biggest risk. While working on the initial setups we faced multiple challenges in enabling CUDA to harness the local system GPU itself due to which training processes for preliminary code was extremely costly. Downloading models and running computationally expensive operations took an unanticipated toll on the project timelines which cost us to deviate from our initial plans too (discussed in the next section). Working with the RCT dataset, we hit another roadblock as the dataset contained a massive 180040 sample set for training and testing which showed a total runtime of over 500 CPU hours. Mitigating the risk, we were able to find a similar but compact dataset for the problem which got us back to the plan and timeline.

4.2. Changes in Approach

Initial testing on the RoBERTa base model indicated a relatively high execution time for the same. Our original plan was to also compare the baseline statistics of a newly created adapter to that of a fine-tuned adapter or a task-adapter adapter so as to see how much of a performance boost the tasks get. However, whole of the aspect of hyperparameter tuning and model selection for adapters was cut short, leading to a sub-optimal overall performance when compared to the fine-tuned RoBERTa model. The trade-off however was mitigated a little but tweaking increasing the batch-sizes and lowering epochs to validate the direction of loss curve the models took and then fine-tuning the parameters incrementally to achieve overall success with correct parameters with respect to generalization.

4.3. Project Success

We planned on indicating the difference of performance as well as required resources with respect to two new text classification tasks between two sets of models – an adapter boosted transformer vs both a base model transformer as well as a fine-tuned transformer. Due to computational constraints, we weren't able to fine-tune the adapter which is why we can't explicitly conclude that an adapter-boosted transformer beats the regular transformer. However, we succeeded in highlighting that adapter models do beat baseline performance of a regular transformer both in terms of test accuracy as well as CPU and Memory utilization, which subsequently ties to a more fruitful training experience. Hence, we term the study as a considerable success.

5. Work Division

Summary of the work division can be found in Table 5 below

Table 5. Student Contribution for Project

Student	Contribution	Details
Yash	Design, Implementation, Experimentation, Report Writing	Paper Reviews and research, solution design, transformer hyperparameter tuning, parameterizing models, code reviews
Sudeshna	Design, Implementation, Experimentation, Report Writing	Paper Reviews, feasibility study, transformer and adapter coding, experimentation and analysis, code reviews
Tejaswin	Paper Reviews, Analysis, Report Writing	Research and paper reviews, report writing

6. References

References

[1] *AdapterHub*. URL: https://adapterhub.ml/.

[2] S. Alvise, H. Rebuffi, and A. Vedaldi. "Learning multiple visual domains with residual adapters". In: *NeurIPS* (2017).

[3] L. Ba, J. Kiros, and G. Hinton. "Layer normalization". In: *arXiv preprint* (2016).

[4] W. Dolan and C. Brockett. "Automatically constructing a corpus of sentential paraphrases". In: *IWP@IJCNLP 2005* (2005).

[5] S. Gururangan and A. Marasović. "Don't Stop Pretraining: Adapt Language Models to Domains and Tasks". In: *ACL* (2020).

[6] N. Houlsby et al. "Parameter-Efficient Transfer Learning for NLP". In: *ICML* (2019).

[7] *Hugging face – the AI community building the future.* URL: https://huggingface.co/.

[8] Y. Liu, M. Ott, and N. Goyal. "RoBERTa: A robustly optimized BERT pretraining approach". In: *arXiv:1907.11692* (2019).

[9] J. Pfeiffer et al. "AdapterFusion: Non-Destructive Task Composition for Transfer Learning". In: *arXiv* (2019).

[10] J. Pfeiffer et al. "AdapterHub: A Framework for Adapting Transformers". In: *arXiv preprint* (2019).

A. Project Code Repository

We used HuggingFace as well as AdapterHub libraries which are made available in Python by their respective creators as cited in the paper. Our project code is available in the following GitHub repository.

https://github.com/yashkrsingh/Transformer-Adapter

B. Model Configuration for Models

Table 6. Configurations for RoBERTa Base

Hyperparameter	Assignment
Number of epochs	10
Dropout	0.1
Learning Rate	1e-6
Learning Rate Decay	Linear
Weight Decay	0.01
Training Batch Size	16
Evaluation Batch Size	16

Table 7. Configurations for RoBERTa Tuned

Hyperparameter	Assignment
Number of epochs	10
Dropout	0.1
Learning Rate	2e-5
Learning Rate Decay	Linear
Weight Decay	0.01
Training Batch Size	16
Evaluation Batch Size	16

Table 8. Configurations for Adapter

Hyperparameter	Assignment
Dropout	0.1
Hidden Activation	GeLu
Hidden Layer Size	768
Attention Heads	12
Normalization Eps	1e-5

C. Computational Power Used for the Model training

The graphs showing the computational power used for the model training is as follows (Figure 6):

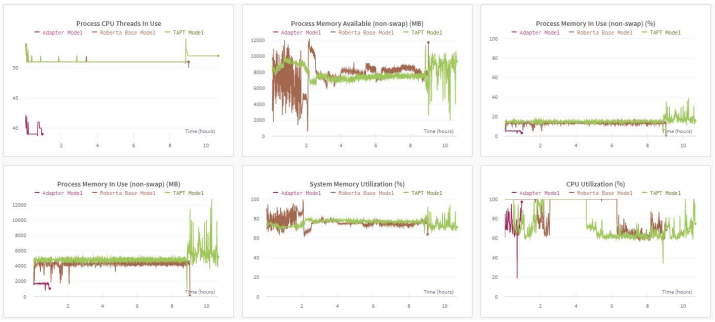

Figure 6. Extra Graphs to show the computational power required to train all the three models