Adapters as a Pre-training Tool for Enhanced Performance
Adapt Or Die : CS 7643

Anthony Menninger, Jacob G Ressler-Craig, Liudmila Tuzova
Georgia Institute of Technology
amenninger3@gatech.edu, jgrc3@gatech.edu, ltuzova3@gatech.edu

Abstract

1. Introduction/Background/Motivation

Large language models (LLMs) powered by the transformer architecture form the foundation for a significant amount of the AI advancements and hype seen today, enabling technologies such as AI chat-bots, LLM-driven search engines, and even the discussion of artificial general intelligence [?]. These models consist of hundreds of millions of parameters trained on huge, heterogeneous corpuses of data [?]. They perform impressively when tested across general domains. However, they often require an additional push to perform competitively on a specific area or task, such as medical diagnosis [?, ?]. Taking an existing LLM, such as RoBERTa, and tuning it to a specific task proves challenging.

This topic was addressed by Gururangan et. al. in their 2020 paper "Don't Stop Pretraining", which proved the benefits of domain-adaptive and task-adaptive pretraining (DAPT/TAPT) on multiple domains and classification tasks. Pre-training describes the step by which an existing model can be trained on additional data, often domain or task-specific, to improve its recollection before being fine-tuned for a specific task. Gurungaran et. al. were able to show that continued pretraining of the model on a domain (DAPT) consistently improved performance on classification tasks from that target domain [?]. They also showed that pretraining the model on a much smaller but directly task-related corpus greatly improved performance on that classification task, with or without being combined with DAPT [?]. However, these current methods require fine-tuning over the whole model, large domain and task-specific datasets, and computationally expensive training times.

In this paper, we explore an 'adapter'-based alternative to domain and task-adaptive pretraining that would allow for fast, data-efficient model tuning. Adapters are small sub-networks that can be plugged into the existing transformer blocks. Instead of updating all the parameters of the pre-trained language model when training adapters, the original model is frozen and only the parameters of the introduced sub-architecture are updated in learning [?, ?]. Adapters are incredibly parameter-efficient, often under 1% of the full model's parameters, and modular. They have been shown previously to provide competitive performance with full fine-tuning in transfer learning [?].

We ask the question if instead of tuning the entire model via DAPT and TAPT for some sub-task, can we get similar if not better performance by attaching and pretraining an adapter. We explore and compare the performance of adapters after DAPT, TAPT, and combined DAPT+TAPT on downstream classification tasks to that of full-model pretraining replicated from the Gururangan paper [?]. We also look at the performance of adapter pretraining on reduced datasets, asking the question if adapters can more efficiently transfer learn from a small corpus of domain or task-specific data than a full transformer-based model. We look at multiple adapter architectures in our study, evaluating the performance of small bottleneck style adapters, like SeqBN, as well as large prefix-and-bottleneck combined UniPELT adapters over different pretraining methods and dataset sizes [?, ?].

Bottleneck adapters consists of two normal feed forward layer, where one of the layers downscales the output and the other upscales. The Pfeiffer adapter we implement is inserted after the feed forward block in each Transformer layer [?]. As the RoBERTa base model consists of 12 stacks of Transformer-encoder layers, the RoBERTa with added Pfeiffer configuration will have of 12 bottleneck adapters consisting of 894,528 added parameters in total [?, ?]. In contrast, UniPELT combines multiple adapter methods, including LORA, Prefix Tuning, and bottleneck adapters, across different areas of the Transformer layer and implements a gating mechanism that controls submodule activation [?]. For the RoBERTa model, it has 11,083,376 parameters to be trained [?].

Competitive adapter-based domain and task-adaptive pretraining methods would allow for efficient and modular transfer learning of existing LLMs to specific tasks. In addition, it would allow the composition of multiple pre-trained adapters for highly competitive, single-model multi-subject performance.

For our project, we use the pre-trained RoBERTa large language model [?]. It is a transformer-based architecture trained using masked language modeling objective on over 160 GB of unlabeled raw text. We attach additional adapter architectures to the model using AdapterHub [?]. We use the Amazon REVIEWS dataset and the Amazon review helpfulness dataset for DAPT and TAPT respectively. The Amazon REVIEWS dataset consists of 24.75 million full Amazon reviews across different product domains, compiled in text form for the purpose of language model training [?, ?]. This dataset was used for domain-adaptive pre-training using masked-language modeling. The Amazon review helpfulness dataset consists of 200,000 Amazon reviews with attached helpfulness labels to be used for positive/negative language classification [?]. Our code, pre-trained models, and datasets are publicly available [?].

2. Approach

Our project had three main phases: reproduce initial pre-training results, replicate the results using an adapter architecture and explore potential novel benefits in using adapters.

2.1. Reproducing Results

Our first goal was to reproduce results from Gururangan 2020 [?] for Domain Adaptive Pre-Training (DAPT), Task Adaptive Pre-Training (TAPT) for at least one domain. This would allow us to create a nuts and bolts understanding of the pre-training process and ensure we could make it work.

2.1.1 DAPT Training

Our first unexpected challenge related to data. Our proposal research had shown that this paper had an online repository that held both the code and data used for the paper. While the Task data was available as expected, the Domain data, which was much larger, was not. We decided to focus on the Amazon Review data because we were able to identify what we think is a similar source. Gururangan 2020 identifies He, McAuley 2016 [?] as the Domain data source, which led to Ni, Li, McAuley [?] paper, which in turn has a website with Amazon reviews.

Gururangan 2020 describes 24.76 million reviews in their dataset. McAuley's site contained several versions, none of which matched this. We chose a filtered set (5-core) of the 2018 dataset, as we thought it was most likely closest to the original. It contained 37 different categories with a total of 75 million reviews. We sampled each category proportionally to get to 25 million and shuffled then shuffled the data.

A key challenge we had anticipated was the computational load of running LLM models. The roberta-base model we used contains 124 million parameters. While relatively small by today's standards, this is still quite large when performing training. Gururangan 2020 used a large batch size for DAPT pre-training, which required the use of gradient accumulation. While this allows training to fit into memory, smaller batch sizes creates longer the run times. The pre-training performs Masked Language Modeling, which does not use labels, so we did have one option. The context window for the roberta-base model is 512 tokens and the average review was 87 tokens long. Each training batch was mostly empty, but used the same amount of memory for computing gradients for backprop. We could put almost 6 reviews on average in each row, increasing training efficiency by almost a factor of 6, while still exposing the model to the same number of reviews. We used this condensed dataset for all DAPT pre-training. We were not able to use this enhancement for classification, because a label is attached for each review preventing mingling of different reviews.

One other improvement we were able to use on the initial reproduction was the use of PyTorch compilation. Since version 2.0, PyTorch allows for compilation of the computation graph, which in our case almost doubled performance, even with the overhead associated with the compilation. This brings us to our second unexpected challenge, which was the effective age of the Gururangan 2020 codebase. The versions of tools needed to run the allennlp tool set [?] are quite old and we were not able to install them on the Google Colab platform [?]. One of our team members had access to a local environment with some computational resources where she was able to make adjustments to allow for installation, but it was tedious. This highlighted the importance of using a robust set of platform tools that will be kept current and working. At this point we worked in parallel, with one team member reproducing results using code from the paper.

The rest of the group began work using the transformers platform [?] to reproduce the results. The transformers platform allows for standardization of many low value tasks, such as tokenization, data collation and basic training evaluation loops. The Huggingface hub allows for models, training parameters, tokenizations and more to be stored in a common format on the internet with open access. This is very valuable for group research, replication of previous works and addition of new enhancements. We also used the datasets platform [?] to easily manage our datasets. We could pre-process the raw dataset into tokens and upload so that each of the team members could have access, especially on the Google Colab platform.

Initial work showed that the Amazon 25 million review dataset, even with review grouping and PyTorch Compilation on the the A-100 with 40 G ram, the fastest GPU available, was going to take 12+ hours to run. We decided to do one run using our local compute on the full dataset while also downsampling to 5 million reviews for DAPT pre-training on the Colab platform. We could then use these models for classification and further review.

2.1.2 Classification

For classification (and TAPT), we were able to use three of the same datasets used in the original paper: Amazon Helpfulness, IMDB Review Sentiment and Citation Intent. We tokenized them using the transformer roberta-base tokenizer and loaded them to the HuggingFace Hub.

Figure 1 shows two key aspects of these datasets. The primary dataset we used was Amazon Helpfulness which is both the largest (seen on the left part of the figure) as well as very imbalanced, seen on the right side of the figure. We can also see that the Citation Intent dataset is quite imbalanced. Gururangan 2020 uses an F1 Macro score for evaluation, which is a good measure for imbalanced data. The ”Macro” setting weights each class equally.

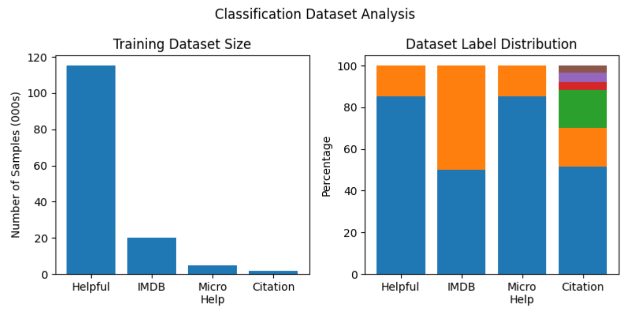

Figure 1. This shows the three datasets used in classification. The left shows the different sizes of training sets while the right shows the percentage of label types.

We were able to then perform classification using the transformers. We performed the first classification using the roberta model. For the DAPT classification, we would remove the Masked Language Modeling (MLM) head used in pre training and replace it with a classification head. We used both the pretrained model that used the full Amazon Reviews dataset (25 million reviews) as well a model trained on 5 million reviews. These results are seen in the Experiments and Results sections. We were able to reproduce a positive DAPT pre training impact, which was encouraging, but the impact was very modest. The extreme times for pre training and classification were limiting for trying different experiments.

2.1.3 TAPT Training

For TAPT training, instead of using domain data for pre training, the data that will be used in the classification task is used for masked language model fine tuning. Just as we did for domain pre training, we created a set of condensed datasets from the Task data to speed pre training. The Task pre training was quicker than domain pre training due to the smaller data size, but classification still took a long time.

We were able also able to reproduce positive TAPT results, as shown in the Experiments section.

2.2. Replicating using Adapters

Having been able to reproduce the basic findings for DAPT using Amazon reviews and TAPT, we felt comfortable that we had good data and effective models and training. The next goal was to implement this using an Adapter architecture, as described in the Introduction. We chose to work with two adapter architectures at this point. The first was one of the seminal adapter architectures, the sequential bottleneck, proposed by Houlsby 2019 [?]. We used a high reduction factor of 16, which represents the ratio of the model's hidden layer dimension and the bottleneck dimension. We chose this reduction factor to create a small adapter to compare with our second adapter UniPelt [?]. Per the introduction, this is a more advanced, multi module architecture with significantly more parameters. These parameter counts are seen in **Table 1**.

Model Adapter	Parameters	% of Model
roberta-base model	124.6	100.0%
Sequential Bottleneck	.9	.7%
Parallel Bottleneck	.9	.7%
UniPelt	11.1	8.9%

Table 1. The parameter count used by different adapters. Both the sequential and parallel bottleneck use a reduction factor of 16.

When training using the adapters, the base model parameters were not updated, only the parameters from the adapter that had been injected into the model.

We were able to somewhat replicate Gururangan 2020 using the adapter architectures, as shown in the Experiment and Results section. Computational cost and time was our biggest issue. Close reading of the original paper highlights that they would ”throw out” degenerate (poorly performing) models, in addition to averaging the results of 5 runs using different seeds. Looking at the standard deviations of results, using multiple runs was an important part of the paper, but prohibitive considering the time and cost burden.

2.3. Adapter Exploration

While we were able to replicate positive pre training effects using adapters, we were very limited in our ability to explore different configurations. We decided to focus on reduced size dataset just for TAPT, with the focus on using only 5k reviews in the ”Micro Help” dataset seen in **Figure 1** . We liked that Task Adaptive Pre Training would be available to all users for classification tasks, without having to consider finding the right domain dataset. We also wanted to explore the smallest amount of pre training that would produce a positive TAPT impact. This joined our interest in a more tractable experimentation framework with the general users interest in improving results with the least amount of effort. Finally, we thought we would use an ”intermediate” adapter architecture, parallel bottleneck, with the bottlenecks in parallel instead of sequentially, as proposed in He et al. 21 [**?**]. We used the same reduction factor as with the sequential bottleneck adapter, so it contained the same amount of parameters as seen in **Table 1**.

With this framework, we were able to run 5 classification runs per experiment, providing a much better picture of performance.

First we investigated if we would see a TAPT benefit using the smaller Micro Help dataset using 1, 5 and 10 epochs of pre-training. When using 1 epoch, we reduced the batch size and removed gradient accumulation so that we would have more training steps. **Figure 2** shows that we were able to produce a positive result with just 1 epoch, which was very encouraging. 10 epochs showed better results, but 5 showed worse. We can see the standard deviation on 5 epochs is much higher, so we did another experiment looking at more pre training.

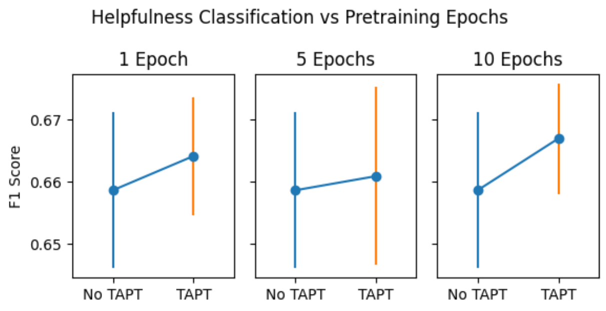

Figure 2. This shows the results of TAPT on the Amazon Helpfulness 5k reviews dataset using different amounts of pre training.

Figure 3 shows that length of pre training is not a good predictor of improvement on classification. Note that the 5 epoch training also shows a lower outcome but much higher standard deviation.

Having established that just 1 epoch of pre training is effective, we wanted to confirm that worked for other datasets. **Figure 4** shows that 1 epoch of pre training improved classification results for all three datasets, which provides a diversity of size, data balance and domain.

At this point, we had a very useful finding, but also a concern. The classification using no TAPT uses a randomly

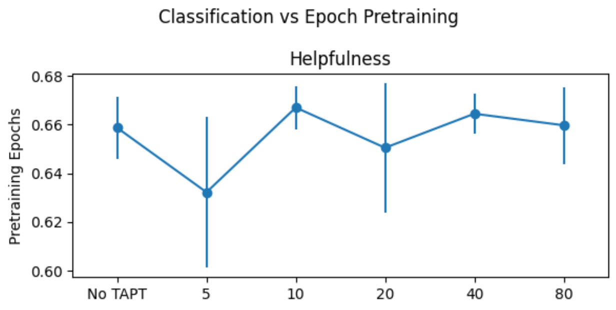

Figure 3. This shows the results of classification with three different datasets before and after Task Adaptive PreTraining (TAPT)

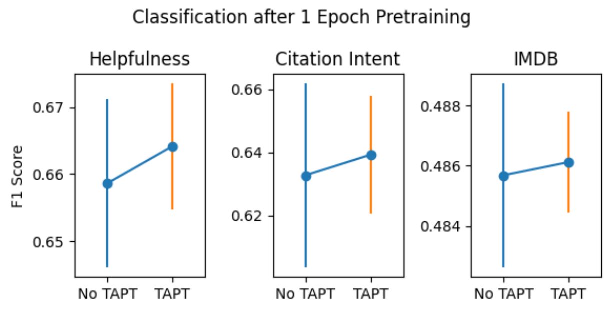

Figure 4. This shows the positive impact of doing only 1 epoch of TAPT pre training across three different datasets, which are of different size, balanced and imbalanced

initialized adapter. Was the pre-training step simply creating a word focused initialization that really was not specific to the task data? **Figure 5** compares the Micro Help classification task using different pre trained adapters. This shows that classification only improved with the task specific pre training, not with pre training on different sets of task data.

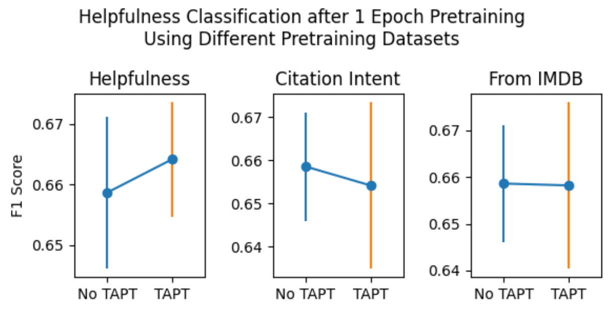

Figure 5. This shows classification results for the Amazon Helpfulness 5k reviews dataset with using different TAPT pre trained models.

ADD TO SUMMARY Future questions - can we determine which initializations will be succesful. More interestingly, can we perform Masked Language Modeling concurrently with classification to produce better outcome using adapter architecture. This would be ideal using a parallel architecture, but has significant hurdle with the different nature of tasks.

3. Experiments and Results

4. Analysis and Conclusion

5. Work Division

ADD PARAGRAPH. Table is added and needs to be filled in. This all counts after the 6 page limit.

Student Name	Contributed Aspects	Details
Anthony Menninger	TBD	TBD
Jacob G Ressler-Craig	TBD	TBD
Liudmila Tuzova	TBD	TBD

Table 2. Contributions of team members.