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

(5 points) What did you try to do? What problem did you try to solve? Articulate your objectives using absolutely no jargon. Pretraining

1. Replicate results of Don't Stop Pretraining - efficacy of domain-adaptive and task-adaptive pre-training on LLMs across different datasets when it comes to a classification task 2. Test the Don't Stop Pretraining protocol for DAPT and TAPT on adapters. Testing across a range of adapter architectures and sizes as well as hyperparams 3. Explore the effects of data set size for adapter pretraining? Can adapter-pretraining learn specific context/texts more efficiently than existing methods.

(5 points) How is it done today, and what are the limits of current practice? Don't Stop Pretraining paper - pretraining existing LLM (RoBERTa) Adapters - current application

(5 points) Who cares? If you are successful, what difference will it make? Allows for more efficient (less data and less compute) LLM finetuning.

(5 points) What data did you use? Provide details about your data, specifically choose the most important aspects of your data mentioned Amazon reviews data Amazon helpfulness labeled CS domain for methodology assurance and further reports

Large language models (LLMs) powered by the transformer architecture form the foundation for a significant amount of the AI advancements and AI-hype seen today, enabling technologies such as AI chat-bots, LLM-driven search engines, and even the discussion of artificial general intelligence [**?**]. These large language models consist of hundreds of millions of parameters that are trained on huge, heterogeneous corpuses of data [**?**]. While they perform impressively when tested across general domains, they often require an extra push, or pre-training, to perform competitively on a specific area or task, such as medical diagnosis [**?**, **?**]. However, taking an existing LLM model, 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. They showed that continued pretraining on a domain (DAPT) consistently improved performance on classification tasks from that target domain [**?**]. They also showed that pre-training the model on a much smaller but directly task-related corpus greatly improved performance, with or without being combined with DAPT [**?**].

In this paper, we explore an 'adapter'-based alternative to this domain and task-adaptive pretraining. Adapters are small subarchitectures that allow for efficient and lightweight fine-tuning of pre-trained Transformer-based LLMs for specific, downstream tasks. Instead of updating all the parameters of the pre-trained LM when fine-tuning, they freeze the existing pre-trained weights and introduce a small subarchitecture of new parameters to the model to be updated in the learning [**?**, **?**]. These 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 explore the performance of adapters after DAPT, TAPT, and DAPT+TAPT on downstream classification tasks. We compare these performances to replicated full-model pretrainings 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, comparing both the performance of small bottleneck style adapters, like SeqBN, as well as large prefix-bottleneck combined UniPELT adapters over pretraining and dataset size [**?**, **?**].

Asking the question???

What does this solve??? 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 adapter for highly competitive multi-subject performance.

DataSets: Amazon reviews, Amazon reviews helpfulness Model: Roberta-base 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 and review helpfulness and sentiment datasets for DAPT and TAPT respectively [**?**]. This data was created for **blank purpose by blank**... **What is this data - size? Collection processData labeling Uses Access**

Background

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 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 [**?**] and the datasets 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 like ours, straight forward replication of previous works and easy pathways to improvement.

Stored datasets and our trained models on the hugging face platform - add appendix?

We used the roberta-base HuggingFace module

2.1.2 Classification

F1 score Dataset imbalance

2.1.3 TAPT Training

2.2. Replicating using Adapters

2.3. Adapter Exploration

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 1. Contributions of team members.