Domain and Task Pretraining for Transformer Adapters

Michael Barnhart
Mbarnhart6@gatech.edu

Ethan Chen
ethanchen@gatech.edu

Anthony Nelson
anelson83@gatech.edu

Nicholas Susemiehl
nsusemiehl@gatech.edu

Abstract

When labeled training data for a specific NLP task is limited, pretraining a language model on a large corpus of unrelated data can provide significant performance improvements. After pretraining on the unrelated but abundant dataset, the model can then be fine-tuned using the smaller task-specific dataset. Using the small labeled task-specific dataset for additional pretraining has further increased model performance. Similarly, adding a pretraining step that uses unlabeled data from the same domain as the task-specific data can provide benefits. However, this process can be computationally expensive and inefficient due to the large number of parameters in the model. It has been demonstrated that the parameter space can be significantly decreased with minimal performance sacrifice through the use of adapter modules for the final fine-tuning step. In this study, we examine whether the benefits of domain adaptive-pretraining (DAPT) and task-adaptive pretraining (TAPT) can be combined with the use of adapter modules to combine the efficiency of adapter modules with the performance gains associated with TAPT and DAPT. We compare performance of base RoBERTa, RoBERTa with TAPT, RoBERTa with DAPT, and RoBERTa with both TAPT and DAPT. We find that finetuning using adapters outperforms conventional finetuning on the full model, but that TAPT and DAPT are less effective when pretraining adapters only compared to when they are applied to the full model.

1. Introduction/Background/Motivation

The most advanced language models available to date are able to achieve state-of-the-art performance by training on large bodies of data spanning multiple domains. One such example is RoBERTa [4] which was trained on over 160 GB of English-language text across multiple domains ranging from encyclopedic entries to news articles. RoBERTa was able to achieve state-of-the-art performances on benchmarks such as GLUE, RACE, and SQuAD. However, the applicability of RoBERTa and other models like it to domains and tasks outside of those which they are trained on is not immediately obvious. To this end, [1] investigated whether additional pretraining on domain- and/or task-specific data was helpful for improving the performance of RoBERTa. This additional pretraining, which they refer to as domain-adapted pretraining (DAPT) and task-adapted pretraining (TAPT), was shown to further increase the performance of RoBERTa on benchmarking challenges. By combining DAPT and TAPT, [1] was able to improve on the benchmark scores of RoBERTa by 5-20% (for the models applied to the computer science domain). DAPT and TAPT require all of the 355M parameters of RoBERTa to be updated again, a computationally-infeasible task for groups without access to the same resources of the authors of these papers. [2] proposed a new method to increase training efficiency using adapter modules. These modules are layers placed at different points within a language model which can then be fine-tuned while the pretrained weights of the model are frozen. This method only required the training of 3.6% of the model parameters in their case (as opposed to the 100% [1] needed to update). These adapters are often not pretrained on additional data, a practice which is contrary to the findings of [1]. In order to investigate the efficacy of the adapter framework proposed in [2], we will add adapters to RoBERTa and compare the resulting effectiveness of the model when these adapters are just finetuned to when they are first pretrained on task-specific data (as in [1]'s TAPT framework) and domain-specific data (DAPT) and then finetuned.

We will perform these tests using the same data as [1] in order to directly compare our results to theirs. Specifically, we will use the SciERC [6] and ACL-ARC [3] task datasets and the computer science literature domain data from S2ORC [5]. SciERC consists of 3,219 training, 455 evaluation, and 974 testing observations each labeled according to 7 different classes. It is oriented towards the task of classifying the relationship between different words and clauses within computer science literature. The ACL-ARC

dataset has 1,688 training, 114 validation, and 139 testing data with labels corresponding to 6 different classes. This dataset was designed to support the task of classifying the intent behind citations used in computer science publications. The computer science domain data from S2ORC is comprised of 2.22M full-text papers. It is unlabeled and therefore used for pretraining. Like [1], we randomly sampled a subset of this large dataset for use in DAPT. However, we only sampled 800MB instead of the 48GB that [1] did due to storage constraints. The SciERC and ACL-ARC data was downloaded using the "datasets" Python package while the computer science domain data was downloaded using access means provided by the S2ORC.

This work will show whether the adapter framework proposed by [2] can obtain the same level of improvement on RoBERTa as the TAPT and DAPT models presented by [1]. If the adapters we train in this work are shown to be more than the results of [1], then we will have shown that state-of-the-art model performances are obtainable on a much smaller computation budget. Furthermore, if we record an increase in performance in adapter-based models when TAPT and/or DAPT are applied prior to finetuning as opposed to just finetuning the models, we will show the importance of this additional pretraining as suggested by [1].

2. Approach

2.1. Prepossessing

Each dataset is run through a RoBERTa specific tokenizer that truncates/pads each text to the max_length of 128. For the pretraining datasets we set the labels to be the same as the input ids so the model is properly able to learn during pretraining. For the fine tuning dataset we encode the labels before processing. Pretraining also requires the use of a Data Collator that adds the ability for the training process to be able to randomly mask different words at random. We had a masking probability of 15%, the same as [1].

2.2. Baseline (RoBERTa + Adapters Fine Tuning Only)

We utilized AdapterHub's adapter-transformer library for the creation and handling of the adapters we are adding to the RoBERTa model. The adapter-transformer library is a fork of HuggingFaces transformer library that adds functionality for creating and manipulating adapters for existing transformer models like RoBERTa without the need of adding the adapters manually. We used the adapter architecture proposed by [7]. The adapter is a bottle-neck adapter that is placed only after the feed-forward block in each transformer layer.

For fine tuning the models, for both pre-trained and randomly initialized adapters, we trained on the SciERC and ACL-ARC labeled datasets. For the task of classification, the models are given a new classification head. Each model is trained for 50 epochs with a learning rate of 1e-4 and a batch size of 16. The models are set to save the best model during training and that model is used for evaluation against the testing set. Each model type is trained five times with different seeds to determine the variance of performance for the model.

2.3. Task Adaptive Pretraining

Our approach to Task Adaptive Pretraining (TAPT) also closely follows that of [1]. We continue pretraining the baseline model, but with the task dataset that is used for fine tuning. We remove the labels from the dataset and have the model pretrain further on the now unlabeled dataset. We use the same adapter initialization as with DAPT.

TAPT will also run similar hyperparameters as [1] which are different from the ones used for DAPT due to the size difference in the datasets. We will be using a batch size of 256 via gradient accumulation and a learning rate of 1e-4 for 100 epochs. Once the model is finishes pretrained the model has its masked language head replaced with a classification head and is trained using the same steps listed in fine tuning section.

2.4. DAPT and TAPT

The cumulative effect of combining both DAPT and TAPT was investigated by carrying out first DAPT, then TAPT according to the procedure previously described.

2.5. Hyperparameter Tuning

We tuned the hyperparameters of the pretrained and finetuned adapters in order to optimize their performances for the TAPT and finetuned adapters. We did not tune the hyperparameters of the DAPT adapter because of the exceptionally long amount of time it took to train even 1 epoch (¿4 hours). This was done using the "optuna" framework which is built into HuggingFace's $hyperparameter_search$ function for its Trainer Class. First, we chose to tune the learning rate and training and validation batch sizes and decided on ranges for these. We chose to test learning rates between 1e-5 and 1e-2 and batch sizes of either 4, 8, or 16. The optuna framework works by randomly sampling hyperparameters from the given space and then training the models with these specifications before recording a validation score. This allows the search to find optimal parameters without having to do an exhaustive grid search which takes much longer to accomplish. It repeats this process for a given number of iterations in order to sample the hyperparameter space multiple times. We then ran the optuna hyperparameter routine for the pretraining and finetuning adapter models for 30 iterations each, with each iteration training the model for 10 epochs. This process improved

our models' performances noticeably, as described in section 4.5.

2.6. Issues

Some issues that were anticipated was training time, due to deadlines we only have a set amount of time to train the model and the large datasets, e.g. DAPT. However, we anticipated that this would not be an issue due to the parameter efficiency of adapter-based models. Since we are only training a fraction the amount of parameters training times should be manageable even with consumer hardware. AdapterHub's api has some quirks that took some time to understand along with documentation that doesn't always list information needed. This made it a pain to find out what the default adapter configuration was being used when a model was being initialized and took digging through the source code to find.

Another challenge was retrieving the DAPT domain dataset. In order to acquire the means to download this data, we first had to contact the S2ORC team. They granted us access to the data after several days and shared a code to download the data. We modified this to only download the computer science domain data, but even this subset was nearly 200GB. We were able to download this data on one team member's personal computer, but this left no way for the other team members to access the data. We attempted to upload the data to Google Cloud Platform's data storage solution but were severely limited by our upload speeds. Eventually, one team member was able to fully access the data and perform DAPT after carefully preprocessing the data.

2.7. Machine Specifications

The training was performed on a system with a GTX 1080 Ti 11GB training for 24 hours. We felt that due to the parameter efficiency that adapter-based models provide that using consumer hardware as opposed to cloud based hardware was the preferred method.

3. Experiments and Results

3.1. Finetuning

For finetuning, we used 2 datasets of computer science papers for our targeted tasks. The first one is SciERC, which includes about 3200 examples. The second one is ACL-ARC, which includes around 1700 examples. Our baseline is the pretrained off-the-shelf RoBERTa-base model with the adaptor described in the previous section. We performed the supervised finetuning for the classification tasks. For the SciERC dataset, the label type is relation classification with 7 labels that are 'COMPARE', 'CONJUNCTION', 'EVALUATE-FOR', 'FEATURE-OF', 'HYPONYM-OF', 'PART-OF', and 'USED-FOR'. The label type of ACL-ARC is citation intent with 6 labels, including 'Background', 'CompareOrContrast', 'Extends', 'Future', 'Motivation', and 'Uses'. For the initial evaluation, we didn't use hyperparameter tuning. We finetuned the model with a learning rate of 1e-4, 50 epochs, and the batch size is 64. The average results of F1 for finetuning the parameters of the RoBERTa-base model with the adaptor are 80.1 for SciERC and 67.8 for ACL-ARC. The results show that finetuning with adapters consistently outperforms the base RoBERTa model.

3.2. TAPT

For the task-adaptive pretraining, we also used the SciERC and ACL-ARC for the unlabeled training. We pretrain the adaptor with a learning rate of 1e-4, 100 epochs, and the batch size is 16. Also, for the pretraining, we randomly masked different words with a masking probability of 0.15 to augment each dataset. The average results of F1 for TAPT + finetuning the parameters of the RoBERTa-base model with the adaptor are 80.0 for SciERC and 69.8 for ACL-ARC. TAPT resulted in a performance improvement for the ACL-ARC taks, but a slight decrease in performance for the SciERC task.

3.3. DAPT

For the domain-adaptive pretraining, we used the papers in the computer science (CS) domain from the dataset of S2ORC that includes 150k CS paper examples. We pretrain the adaptor with a learning rate of 1e-4, 1 epoch, and the batch size is 16. Again, for the pretraining, we randomly masked different words with a masking probability of 0.15 to augment each dataset. The average results of F1 for DAPT finetuning the parameters of the RoBERTa-base model with the adaptor are 80.2 for SciERC and 68.8 for ACL-ARC. DAPT resulted in a slight performance improvement over the base adapter model.

3.4. DAPT+TAPT

We also combined both DAPT and TAPT to pretrain the RoBERTa + Adapter model, and then finetune the parameters of the model. The average results of F1 for DAPT+TAPT with finetuning the parameters of the RoBERTa-base model with the adaptor are 79.0 for SciERC and 69.0 for ACL-ARC. Combining DAPT and TAPT resulted in a slight performance decrease for the SciERC task and an increase in performance for the ACL-ARC task, when compared to the baseline adapter model. However, it should be noted that the DAPT+TAPT performance was lower for the ACL-ARC task than when only TAPT was performed.

3.5. Hyperparameter Tuning

To achieve better performance, we used the aforementioned methods to tune the hyperparameters while we pretrain the model with task-specific datasets. The average results of F1 for hyperparameter tuning with finetuning the parameters of the RoBERTa-base model with the adaptor are 82.1 for SciERC and 67.4 for ACL-ARC. The average results of F1 for TAPT + hyperparameter tuning with finetuning the parameters of the RoBERTa-base model with the adaptor are 84.6 for SciERC and 68.4 for ACL-ARC. Both results outperform the results of DAPT+TAPT+FT from Gururangan et al. 2020.

3.6. Metrics

The F1 scores on the testing set from our work pretraining and finetuning adapter can be found in Table 1.

4. Discussion

Performance improvements due to TAPT and DAPT were less significant when performed on adapters than they were in [1], where they were applied to the full model. In [1], TAPT, DAPT, and DAPT + TAPT all resulted in higher F1 scores for both the ScieERC and ACL-ARC datasets, whereas the results were more ambiguous when these pretraining regimes were applied to adapters. In some cases, further pretraining actually reduced the performance of the adapter model.

At the same time, it should be noted that finetuning on adapters not only is more efficient than finetuning on the full model, but it actually results in superior performance as well. This may explain, at least in part, why TAPT and DAPT were less effective in the adapter model than in the full model. Our finetuned adapter model (without additional pretraining) already outperforms the finetuned full model with TAPT, and is nearly the same as with DAPT in the case of SciERC. It may be that the performance improvement seen when performing TAPT on the full model is mostly is due to compensating for the difficulty of training such a large, complex model with a limited dataset. This issue, however, is largely solved through the use of adapters by reducing the number of trainable parameters, so the benefit of TAPT is diminished.

5. Conclusion

We demonstrated that adapters are an efficient method for finetuning a pretrained model, enabling major reductions in computational costs of training without significantly sacrificing performance. Furthermore, we showed that additional pre-training using domain- or task-specific data, while useful for further pretraining of the full model, is not beneficial when used on adapters. Based on these results, DAPT or TAPT should only be performed on the full model.

6. Work Division

All members made substantial contributions to both the technical and writing components of this project. Many aspects of the code were developed iteratively by the entire team and together as a group over video conference. Our work division is provided in Table 2.

References

[1] 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 e-prints*, page arXiv:2004.10964, Apr. 2020. 1, 2, 4

[2] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-Efficient Transfer Learning for NLP. *arXiv e-prints*, page arXiv:1902.00751, Feb. 2019. 1, 2

[3] 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

[4] 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. *arXiv e-prints*, page arXiv:1907.11692, July 2019. 1

[5] Kyle Lo, Lucy Lu Wang, Mark Neumann, Rodney Kinney, and Dan S. Weld. S2ORC: The Semantic Scholar Open Research Corpus. *arXiv e-prints*, page arXiv:1911.02782, Nov. 2019. 1

[6] Yi Luan, Luheng He, Mari Ostendorf, and Hannaneh Hajishirzi. Multi-task identification of entities, relations, and coreference for scientific knowledge graph construction. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3219–3232, Brussels, Belgium, Oct.-Nov. 2018. Association for Computational Linguistics. 1

[7] Jonas Pfeiffer, Andreas Rücklé, Clifton Poth, Aishwarya Kamath, Ivan Vulić, Sebastian Ruder, Kyunghyun Cho, and Iryna Gurevych. AdapterHub: A Framework for Adapting Transformers. *arXiv e-prints*, page arXiv:2007.07779, July 2020. 2

Model	Task	RoBa	FT	TAPT+FT	DAPT+FT	DAPT+TAPT+FT	HP Tuning	TAPT+HPT
This Work	SciERC	-	80.1	80.0	80.2	79.0	**82.1**	**84.6**
This Work	ACL-ARC	-	67.8	69.8	68.8	69.0	67.4	68.4
Gururangan et al., 2020	SciERC	77.3	-	79.3	80.8	81.3	-	-
Gururangan et al., 2020	ACL-ARC	63.0	-	67.4	75.4	75.6	-	-

Table 1. Table 1: Results on different phases of adaptive pretraining compared to the reference of Gururangan et al. 2020, including baseline RoBERTa (RoBa), RoBERTa + Adapter with finetuning (FT), our commbinations of DAPT, TAPT, and finetuning (FT), and our hyperparameter tuning (HPT).

Student Name	Contributed Aspects	Details
Ethan Chen	Domain data prep, model validation, TAPT pretraining, funetuning	Solved DAPT data extension problem. Performed model validation and experiments of TAPT, finetuning, and hyperparameter tuning.
Anthony Nelson	Domain data processing, DAPT pretraining, DAPT + TAPT pretraining, finetuning	Wrote logic for DAPT data processing and training. Performed DAPT and DAPT + TAPT experiments. Performed one iteration of baseline finetuning.
Nicholas Susemiehl	FT & TAPT validation, DAPT data management, Hyperparameter Tuning	Helped develop and test FT and TAPT code. Modified provided script to obtain proper domain data. Developed hyperparameter tuning code for TAPT and finetuning and ran this code to obtain hyperparameter tuning-related results.
Michael Barnhart	Coding up notebooks, TAPT pretraining, finetuning	I coded the notebooks in which the experiment was run on. I also resolved a lot of the bugs that came up while testing as well as writing the documentation.

Table 2. Contributions of team members.