PALs of Alto: Comparing Adapter Modules and Projected Attention Layers for Multitask Learning

Stanford CS224N Default Project

Jonah Cader
Department of Computer Science
Stanford University
jcader@stanford.edu

Abstract

The public release of ChatGPT in November 2022[1] catalyzed a wave of public interest in large, transformer-based architectures. But for years, these models have been the de facto standard within the Natural Languge Processing (NLP) community because of their performance and flexibility across a wide rnage of classic NLP problems. From sentiment classification to paraphrase detection, these large, pre-trained models are often adapted to novel downstream tasks through feature-based transfer learning and fine-tuning. Unfortunately, the computational costs and energy required to train new weights for each individual task are severe. As model size continues to scale, there is an increasing need for parameter-efficient transfer learning techniques to maintain the ubiquity of bleeding-edge architectures while minimizing computational overhead. The goal of this project was twofold: First, to implement a functional version of minBERT,[2] finetuned for sentiment analysis, and second, to compare the performance and compactness of two parameter-efficient methods for transfer learning with BERT. Specifically, this project asked which approach, between adapter modules [1] and projected attention layers (PALs) [2], provides the better balance of parameter efficiency and empirical performance across three NLP tasks. Across this study's experiments, PALs outperformed adapter modules in aggregate across three downstream tasks: Sentiment analysis, paraphrase detection, and semantic textual similarity. PALs achieved an blended test set score of 0.577 across three benchmark datasets, while adapter modules achieved a blended test set score of 0.311. PALs achieved similar results to the baseline (naive classifier) approach, which also scored 0.577 (test set).

1 Key Information

- External collaborators (if you have any): None
- Mentor (custom project only): N/A
- Sharing project: No

2 Introduction

The advent of powerful Transformer networks [3], such as BERT (Bidirectional Encoder Representations from Transformers) [4], ushered in a new wave of natural language processing (NLP). Researchers scrambled to demonstrate the potential of these new systems, finding applications across a range of NLP tasks, from question answering to text classification. Large, pre-trained models such as BERT are often adapted to new downstream tasks through two forms of transfer learning:

[1] https://openai.com/blog/chatgpt

[2] Based on Carnegie Mellon University's CS11-711 Advanced NLP assignment, developed by Zhou et al.

Feature-based transfer and fine-tuning. The former relies on word, sentence, or paragraph embeddings that are input into task-specialized models. The latter requires optimizing the pre-trained weights with an additional output layer, and generally yields better performance. In both cases though, the computational costs and energy required to train new weights for each task are severe–especially given the sheer number of parameters in the BERT$_{BASE}$ (110 million) and BERT$_{LARGE}$ (330 million) models. As state-of-the-art model size continues to swell, there is an increasing need for parameter-efficient approaches to multitask learning in natural language processing.

In 2019, researchers developed two concurrent approaches for parameter-efficient transfer learning with BERT: Adapter modules and PALs. Houlsby et al. explored adapter modules for natural language processing in their paper, *Parameter-efficient transfer learning for NLP* [1]. Adapters are function modules defined as $\psi_{w,v}(x)$, where w is the set of parameters learned in pre-training. For tuning, the initial parameter v_0 is set such that the output approximates that of the original function, $\phi_w(x)$. During training, only v is updated, which is more parameter efficient than composing $\phi_w(x)$ with a new function (feature-based transfer) or updating the weights w (fine-tuning). Houlsby et al. [1], incorporated adapter tuning into BERT by adding two bottleneck adapter modules into each Transformer layer.

Projected attention layers (PALs) come from the work of Stickland & Murray in *BERT and PALs: Projected attention layers for efficient adaptation in multi-task learning*. [2] The pair used a different form of adaptation module to attempt the same type of efficient transfer learning as Houlsby et al. in their paper. Specifically, the PALs approach inserts task-specific functions (the PAL layer) in parallel to each of BERT's self-attention layers. These task-specific functions are of the form $TS((\mathbf{h}) = V^D g(V^E \mathbf{h})$, where V^D and V^E are respectively encoder and decoder matrices shared across BERT layers (but not tasks), and $g(\cdot)$ is multi-head attention applied to input embedding $\mathbf{h}$.

The Houlsby et al. and Stickland & Murray papers not only shared similar approaches, but also venues. Both pieces of research appeared at the International Conference on Machine Learning (ICML) in 2019. That said, the treatment of concurrent work the original papers is insufficient to actually understand meaningful differences in these two methodologies and the relative merits of each. THe lack of a clear and direct comparison of PALs and adapter modules motivates this project.

3 Related Work

Transformer Architectures: Human language is a vastly more flexible tool than historical NLP models have captured. As with many domains of artificial intelligence, NLP has traditionally evolved in functional silos: One model for translation, another for sentiment analysis, and so on. The modern era of large language models has certainly introduced a new level of power and adaptability through the Transformer [3] and its variants such as BERT [4]. Vaswani et al. introduced their Transformer architecture in their seminal 2017 paper, *Attention is all you need*. Their approach relies on attention mechanisms to deliver strong performance and generalization across a number of NLP tasks. [3]. In 2019, Devlin et. al introduced BERT, a bidirectional pre-trained model (leveraging Transformers) that could easily be fine-tuned with a single additional layer to achieve impressive performance across a battery of tasks. [4]

Multi-Task Learning: Despire the impressive results of architectures such as BERT, the process of fine-tuning individual models for separate tasks is cumbersome and computationally inefficient. This has resulting a field of research dedicated to more efficient multi-task learning. Tay et al. provide a nice survey of many recent approaches to parameter to efficient learning with Transformers in their 2022 paper [5]. Most relevant to this project are the approaches of Houlsby et al. [1] and Stickland & Murray [2]. As mentioned above, Houlsby et al. pushed the literature forward, by offering a parameter-efficient approach capable of extending BERT to multiple benchmark NLP tasks through the use of adapter modules. In doing so, they offered a glimpse into a state-of-the-art NLP approach that is closer to natural language in its flexibility, rather than functionally siloed. The basic concept of adapter modules has previously been applied to computer vision tasks in the work of Rebuffi et al. [6]. Houlsby et al. thus blended approaches from two domains of artificial intelligence–NLP (BERT) and computer vision (adapter tuning [6])–thereby advancing our understanding of the cross pollination possible between discrete fields of research. Contemporaneously Stickland & Murray provided another approach to adapter tuning that leveraged projected attention layers (PALs), a parallel processing module with task-specific parameters. Both approaches used a bottleneck approach to

extend BERT to downstream tasks with the addition of relatively few additional parameters (compared to fine-tuning).

4 Approach

There are four key components of my approach to this problem: The minBERT implementation, a naive multitask classifier, and two parameter-efficient multitask learning approaches: Adapter modules and PALs.

4.1 minBERT

This project leverages a completed minBERT implementation as described in the CS 224N Default Final Project specification[3] as its base model. The minBERT model is a lightweight version of BERT [4], adapted from a Carnegie Mellon University CS11-711 Advanced NLP Assignment. My implementation tracks to the approach outlined in the handout and starter code.

4.2 Naive Multitask Classifier

As a starting point I implemented a relatively naive multitask classifier to produce baseline scores for the dev set leaderboard. My baseline `multitask_classifier.py` uses the minBERT model in conjunction with the Stanford Sentiment Treebank (SST) dataset[4] using the default project parameters. For the sentiment classification task it extracts the minBERT pooled representation of each sentence before applying dropout, followed by a linear layer to output logits as in the minBERT `classifier.py` approach. For the paraphrase detection task, my model extracts the minBERT pooled representation for each set of sentences. For prediction it concatenates them into a single input to a fully connected layer to generate logits for scoring per the approach described in Arase & Tsujii [7]. Finally, for the semantic textual similarity task, my model model extracts the minBERT pooled representation for each set of sentences, applies a dropout and feedforward layer to each, and calculates the cosine similarity, normalized to a 0-5 scale to match the range of possible similarity scores in the underlying data. In each of these cases, `multitask_classifier.py` finetunes on the relevant datasets (see below) using the AdamW optimizer [8] [9], implemented as part of the base minBERT code.

4.3 Round Robin Sampling

As an extension of the naive multitask classifier, and infrastructure for my more nuanced multitask learning approaches, I used round robin sampling to expose the model to underlying data for all three tasks. My implementation standardizes the size of the training datasets to that of the median length (SST) for training efficiency. To do so it randomly downsamples a subset of Quora data (see below) and duplicates some entries of the STS data (see below), before zipping them together. During training, each task uses a different loss function (based on the expected prediction structure). For sentiment detection, `multitask_classifier.py` uses cross-entropy loss. For paraphrase detection the model uses binary cross-entropy loss. Finally, for semantic textual it uses the mean squared error. Throughout my experiments I uses two forms of round robin: uniform and annealed sampling. In uniform sampling, the script randomly chooses a batch from one of the three datasets with equal probability. In anealed sampling, I implement the approach as described in the Stickland & Murray PALs paper [2]. Annealed sampling picks a dataset i with probabilty $p_i \propto N_i^{\alpha}$, where a smaller alpha diminishes the impact of disproportionately sized dataset. Per [2] $\alpha = 1 - 0.8\frac{e-1}{E-1}$, where e is the epoch out of E total epochs. This effectively ensure more equitable training across tasks towards the end of training, which the authors found to help performance. Note that I skimmed the Strickland & Murray implementation[5] for inspiration on how to share encoder and decoder weights across PALs when setting-up BERT layers. I adapted small snippets of their code for this purpose, though they use a different underlying implementation of BERT.

[3] http://web.stanford.edu/class/cs224n/project/default-final-project-bert-handout.pdf

[4] https://nlp.stanford.edu/sentiment/treebank.html

[5] https://github.com/AsaCooperStickland/Bert-n-Pals

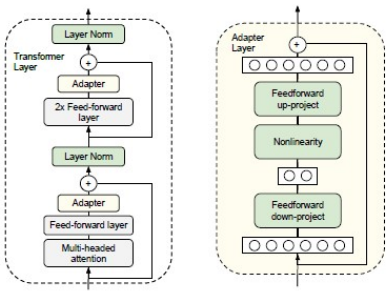

Figure 1: Adapter module architecture from [1].

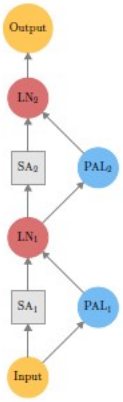

Figure 2: PAL architecture from [2].

4.4 Parameter-Efficient Multitask Learning: Adapter modules.

For my first downstream task extension I implemented a version of the adapter modules from [1]. Per the original paper, this approach involves inserting two serial adapters into each BERT transformer layer, one after applying multi-head attention and one after applying the feedforward layers (see Figure 1). Each adapter module in turn down-projects the input features to a bottleneck dimension size (here, 64 based on the findings of Houlsby et al.), applies a GELU non-linearity, and then re-projects back to the original size before adding a skip connection (see Figure 1). The adapter module layers and the BERT layer norms are all task-specific, meaning they are only tuned when exposed to relevant data for their given task. My implementation largely stays true to the approach diagrammed in the initial paper. That said, Houlsby et. all work with a slightly different baseline BERT layer implementation. As a result, I feed the outputs of my adapter layers into each `add_norm`, alongside the outputs of multi-head attention (first adapter module) and the feedforward layers (second adapter module).

4.5 Parameter-Efficient Multitask Learning: PALs.

For my second downstream task extension I similar implemented a version of the projected attention layers (PALs) from [2]. As with the original paper, this method involves insert a PAL in parallel to each BERT layer, ultimately passing its output as a skip connection of sorts into the final `add_norm`. (See Figure 2.) Each PAL itself is relatively simple: As outlined above it is a task-specific function of the form $TS((\mathbf{h}) = V^D g(V^E \mathbf{h})$, where V^D and V^E are represented by linear layers and $g(\cdot)$ is multi-head attention. Thus each PAL down-projects to a bottleneck dimension size of 204 (based on [2]), applies multi-head attention, re-projects back to the initial dimensional, and finally applies a GELU activtation function. The weights V^E and V^D are importantly shared across layers, but not tasks. Per Strickland & Murray my model using round robin sampling with annealing to achieve better PAL performance.

5 Experiments

5.1 Data

The experimental downstream tasks rely on four datasets across three problem types.

- **Sentiment Analysis:** This task relies on the Stanford Sentiment Treebank (SST) mentioned earlier and CFIMDB datasets. The former contains 215,154 unique phrases, extracted from 11,855 sentences and labeled by three human judges. (Sentiment is annotated as negative, somewhat negative, neutral, somewhat positive, or positive.) The latter contains 2,434 movie reviews (often longer than one sentence), each assigned a binary sentiment label (positive or negative).
- **Paraphrase Detection:** This task uses a subset of the 400,00 question pairs in the Quora dataset.[6] Specifically we use 202,152 question pairs, each labeled with whether they are or are not paraphrases of each other.
- **Semantic Textual Similarity:** This task leverages the SemEval STS Benchmark dataset [10], which contains 8,628 sentence pairs, each assigned a similarity score between 0 (no relation) and 5 (equivalent).

5.2 Evaluation Method

For sentiment analysis and paraphrase detection, the core evaluation metric is classification accuracy. For semantic textual similarity, the metric is Pearson correlation between true and predicted similarity scores. The aggregate perforamnce metric is an average across task performance on either the dev or test sets.

5.3 Experimental Details

I use the naive multitask classifier described above with the default parameters: 10 epochs, batch size of 8, dropout probability of 0.3, and a learning rate of 0.001 (pretraining) or 0.00001 (finetuning).

5.4 Results

minBERT: My minBERT implementation achieves strong performance on the sentiment classifcation task, with dev set accuracy of 0.409 (pretraining) and 0.520 (finetuning) on the SST dataset and dev set accuracy of 0.788 (pretraining) and 0.967 (finetuning) on the CFIMBD dataset.
Downstream Tasks: The table below reports the dev set multitask performance across tasks.

Model	Sentiment	Paraphrase	STS	Overall
Naive Classier	0.528	0.480	0.269	0.426
Naive Classifier (Round Robin)	0.514	0.716	0.505	0.579
Naive Classifier (Annealing)	0.508	0.722	0.452	0.561
Adapter Module (Round Robin)	0.253	0.625	0.093	0.324
PALs (Round Robin)	0.532	0.705	0.480	0.572
PALs (Annealing)	0.506	0.718	0.501	0.575

Additionally, the following table reports test set multitask performance for the naive classifier (with round robin sampling), the adapter modules (with round robin sampling) and finally the PALs approach (with annealed sampling).

Model	Sentiment	Paraphrase	STS	Overall
Naive Classifier (Round Robin)	0.534	0.713	0.483	0.577
Adapter Module (Round Robin)	0.231	0.631	0.072	0.311
PALs (Annealing)	0.490	0.722	0.460	0.577

In their original papers, Houlsby et al. and Strickland & Murray were trying to demonstrate parameter efficient approaches that achieved near benchmark results. Here, were largely able to replicate these

[6] https://quoradata.quora.com/First-Quora-Dataset-Release-Question-Pairs

results for PALs, which proved to be the higher-performing method in this project. Ultimately, on the test set, PALs with annealed sampling achieved the same aggregate score of 0.577 as the baseline naive classifier with round robin sampling. On the dev set PALs performance (0.575 overall) was similarly within 0.1 % of the baseline (0.579 overall). This seems to indicate accurate implementation of PALs and replication of the original paper's experimental results. Unfortunately adapter modules underperformed PALs and the baseline on both dev (0.324) and test (0.311) sets. One hypothesis is that the adapter modules didn't work as well in the context of our BERT implmentation versus the one used by Houlsby et al. in their paper.

6 Analysis

In addition to the empirical accuracy of these approaches, it is also worth exploring two additional dimensions of tradeoffs between adapter modules and PALs.
Flexibility: While PALs requires simultaneous exposure to all underlying task dataset during training, adapter modules can be trained serially on data given that the entire module is effectively task specific. In my implementation I chose to use round robin sampling for the adapter modules rather than sequential training for both efficiency (time spent training) and for a more direct comparison to the PALs approach. That said, this attribute does make the adapter module approach more flexible than PALs.
Parameter efficiency: As noted in the original papers, PALs with a bottleneck dimension of $m = 204$ yield roughly $1.13\times$ the initial number of parameters[2], while adapter modules with a bottleneck dimension of $m = 64$ yield roughly $1.2\times$ the total number of parameters. Thus, in our experiments PALs provided the better balance of performance and efficiency, but adapter modules remain a more flexible approach.

7 Conclusion

By examining adapter module and PALs, this project aimed to shed light on the tradeoffs of these approaches for parameter-efficient transfer learning. In doing so it aspired to codify NLP best-practices as the field continues towards larger and larger models. In comparing these two approaches, we found that PALs outperformed adapter modules across our three NLPS tasks: Sentiment analysis, paraphrase detection, and semantic textual similarity, achieving a test set score of 0.577 (PALs) compare to 0.311 (adapter modules). PALs tied the performance of the baseline (naive classifer) on the test set (0.577 aggregate score), and achieved within 0.1% of baseline performance on the dev set (0.575 versus 0.579). That said, this comparision may be limited by the adjustments made to fit adapter modules into our BERT implementation. Future research could broad the impact of this work by exploring additional parameter-efficient multitask learning techniques.

References

[1] Neil Houlsby, Andrei Giurgiu, Stanisław Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for NLP. In *International Conference on Machine Learning (ICML)*, 2019.

[2] Asa Cooper Stickland and Iain Murray. BERT and PALs: Projected attention layers for efficient adaptation in multi-task learning. In *International Conference on Machine Learning (ICML)*, 2019.

[3] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Neural Information Processing Systems (NIPS)*, 2017.

[4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In *arXiv preprint arXiv:1810.04805*, 2018.

[5] Yi Tay, Mostafa Dehghani, Dara Bahri, and Donald Metzler. Efficient transformers: A survey. In *ACM Comput. Surv. 55, 6, Article 109*, 2022.

[6] Sylvestre-Alvise Rebuffi, Hakan Bilen, and Andrea Vedaldi. Learning multiple visual domains with residual adapters. In *Neural Information Processing Systems (NIPS)*, 2017.

[7] Yuki Arase and Junichi Tsujii. Transfer fine-tuning of BERT with phrasal paraphrases. In *Computer Speech & Language Vol. 66, 101164*, 2021.

[8] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *arXiv preprint arXiv:1711.05101*, 2017.

[9] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *preprint arXiv:1412.6980*, 2014.

[10] Eneko Agirre, Daniel Cer, Mona Diab, Aitor Gonzalez-Agirre, and Weiwei Guo. SEM 2013 shared task: Semantic textual similarity. In *Second joint conference on lexical and computational semantics, Vol. 1: Proceedings of the Main conference and the shared task: Semantic textual similarity, pages 32–43*, 2013.