Survey of Finetuning Techniques

Zach Lawless (ztl2103) and Shawn Pachgade (snp2128)

Abstract

Fine-tuning pre-trained models is the go-to approach for getting good results based on proven state-of-the-art architectures. The process of fine-tuning however can be tedious and slow as hyperparameter selection is incredibly important to achieving good results on a target task. In addition, multiple novel fine-tuning techniques have been introduced, which allows us to increase the capacity of our models, but exacerbates the issue of experimentation. We have created a framework that allows us to quickly iterate on various fine-tuning schemes when training large transformer models. The schemes we have introduced here can also be extended to other domains outside of NLP.

Introduction and Motivation

Fine-tuning and open-sourced pre-trained models have greatly expedited the model training process to allow us to obtain good results on arbitrary target tasks. While much of the burden is reduced with the advent of transfer learning, it can still be quite difficult to approach the peak performance of a given architecture without heavy experimentation. This is especially due to the phenomenon of “catastrophic forgetting”. However, recent advances in deep learning literature have proposed solutions to the forgetting problem.

We focus on fine-tuning as our main mechanism for training models for target tasks because they have been shown to be effective at leveraging the features of other datasets to be adapted to an arbitrary use-case [1]. However, even in the cases where fine-tuning may not be successful in a particular scenario, it is hard to come to that conclusion without heavily iterating through multiple hyperparameters. Thus, we seek a generalizable solution to finding a best attempt at finding an adequate model.

Background

Adapters

One key idea proposed is the introduction of adapters in Houlsby et al. [2]. Adapters are extra layers inserted in between the original layers of a pre-trained model, where the original model weights are intended to be frozen. The adapter parameters that were added to the network are then then the only weights trained to get us our target task model. This solves the problem of catastrophic forgetting, and it also comes with the added benefit of providing a modular like approach to task inference, since the base model itself is common to all tasks it was trained on, whereas only the adapters are the parts of the model that are task specific.

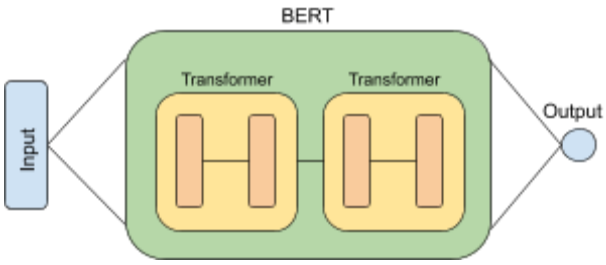

Figure A: A simplified representation of a BERT model with 2-stacked transformers

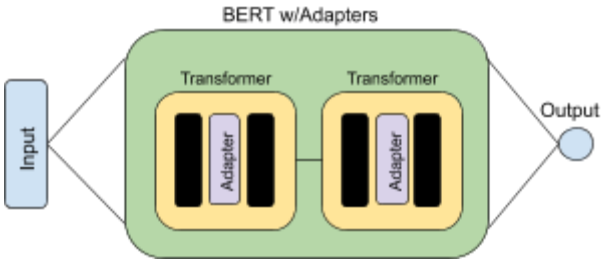

Figure B: BERT with adapters added

Differential Learning Rates

Differential learning rates, otherwise known as discriminative fine-tuning by Howard et al. [4] in their paper on finding a universal fine-tuning model, agrees with the idea that lower-level features in a network are common and more broad across many domains, thus it is important to adjust these weights very carefully as to not disturb the integrity of the low-level text features captured. This is in contrast to the weights closer to the classification layer, which are considered more task specific and require a fuller training. Concretely, this means applying a low learning rate to the weights closer to the input, and a relatively high learning rate to the weights close to the output.

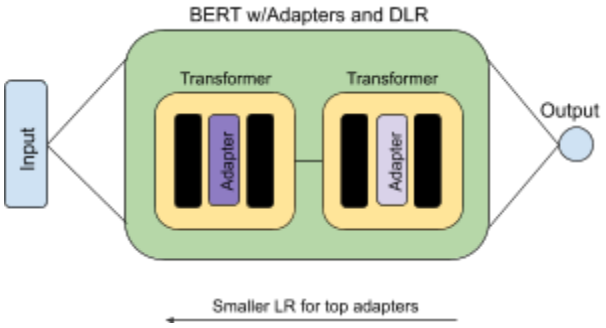

Figure C: BERT with adapters and differential learning rates

Gradual Unfreezing

Gradual unfreezing is also experimented with in [4] and attempts to solve the issue of catastrophic forgetting. In gradual unfreezing, all layers of a network are initially frozen except the classification layer. Once the classification layer is fully trained, the next layer behind it is unfrozen and trained until convergence, and this is repeated.

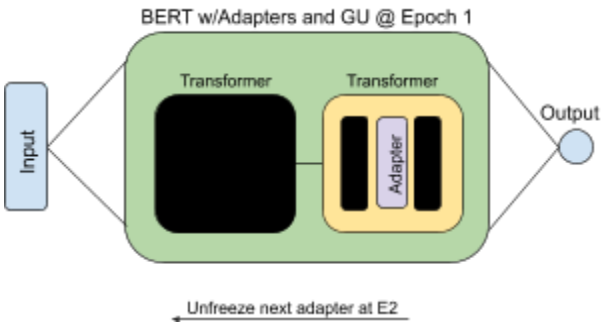

Figure D: BERT with adapters and gradual unfreezing

Learning Rate Schedules

The most recent influential paper on the topic of learning rate schedules was the introduction of cyclical learning rates by Smith [5]. There have been other variants of the cyclical learning rate algorithm that have spun off as a result such as SGDR [6], and in general these types of schemes have been seen as fairly aggressive, but effective. This is in part due to the phenomenon observed as “Super Convergence” [7] in which high LRs have seen improvements in training networks to convergence quickly. Howard et al. [4] actually has his own version of the one-cycle policy termed “slanted triangular” policy, which is what we investigate further.

Approach and Implementation

In order to quickly iterate through some of the aforementioned concepts, we invest in creating a testbed framework where we can rapidly create models and apply different learning schemes to said model. We built a Trainer package that performs all of the data loading/preprocessing as well as configuration of the transformer based on the command line parameters passed at execution. During fine-tuning, we incrementally save results for evaluation once fine-tuning is complete.

We began by using Google Colab to prototype in Jupyter Notebooks. Once we were confident in our approach, we pivoted to using PyCharm for local development and tested our implementation using Google Cloud Platform GPU instances. Because we used Git for version control, we could quickly go back-and-forth between development and testing. We encountered some software bugs when testing on GCP because of PyTorch and CUDA version mismatches, but were able to work around the memory leak issues that were due to the inconsistency.

Lastly, we ran a handful of experiments on multiple datasets using the various fine-tuning approaches listed above to evaluate their performance.

Experiment Results

We ran a multitude of experiments on both the SST-2 and CoLA binary classification datasets. The SST-2 dataset’s input is a sequence of text and the labels indicate whether the sample is

positive or negative. The CoLA dataset, on the other hand, contains sequences of text and the labels indicate whether the samples are grammatically correct or not.

Below are tables containing results from the various experiments we ran on both datasets.

Dataset	Epochs	Batch Size	Learning Scheme	Learning Rate	Adapter	Scheduler	Validation Loss	Validation Accuracy
SST2	5	32	Fixed	0.01			0.235	0.907
SST2	5	32	Fixed	0.01	Yes		0.251	0.900
SST2	5	32	Fixed	0.01		Triangle	0.238	0.914
SST2	5	32	Fixed	0.01	Yes	Triangle	0.229	0.925
SST2	5	32	Nesterov	0.01			0.227	0.911
SST2	5	32	Nesterov	0.01	Yes		0.234	0.907
SST2	5	32	Differential	0.01			0.346	0.842
SST2	5	32	Differential	0.01	Yes		0.345	0.858
SST2	5	32	Gradual Unfreeze	0.01			0.389	0.835
SST2	5	32	Gradual Unfreeze	0.01	Yes		0.424	0.806

Table 1: Experiment Results on SST-2 dataset

Dataset	Epochs	Batch Size	Learning Scheme	Learning Rate	Adapter	Scheduler	Validation Loss	Validation Accuracy
CoLA	10	32	Fixed	0.01			0.483	0.772
CoLA	10	32	Fixed	0.01	Yes		0.458	0.776
CoLA	10	32	Fixed	0.01		Triangle	0.433	0.822
CoLA	10	32	Fixed	0.01	Yes	Triangle	0.407	0.831
CoLA	10	32	Nesterov	0.01			0.449	0.819
CoLA	10	32	Nesterov	0.01	Yes		0.417	0.814
CoLA	10	32	Differential	0.01			0.548	0.742
CoLA	10	32	Differential	0.01	Yes		0.560	0.739
CoLA	10	32	Gradual Unfreeze	0.01			0.554	0.737
CoLA	10	32	Gradual Unfreeze	0.01	Yes		0.574	0.737

Table 2: Experiment Results on CoLA dataset

As shown in the tables, we kept the number of epochs, batch size, and initial learning rate the same for each experiment performed on the two datasets. Due to the smaller number of training

samples in the CoLA dataset, we were able to train for 10 epochs as opposed to the 5 used for the SST-2 dataset.

In order to visualize the performance of the experiments, we captured the validation accuracy (as well as other metrics such as loss and training time) for each experiment. Below are figures demonstrating the performance of the various learning schemes on the two datasets, segmented out by whether or not the transformer contained an adapter or not.

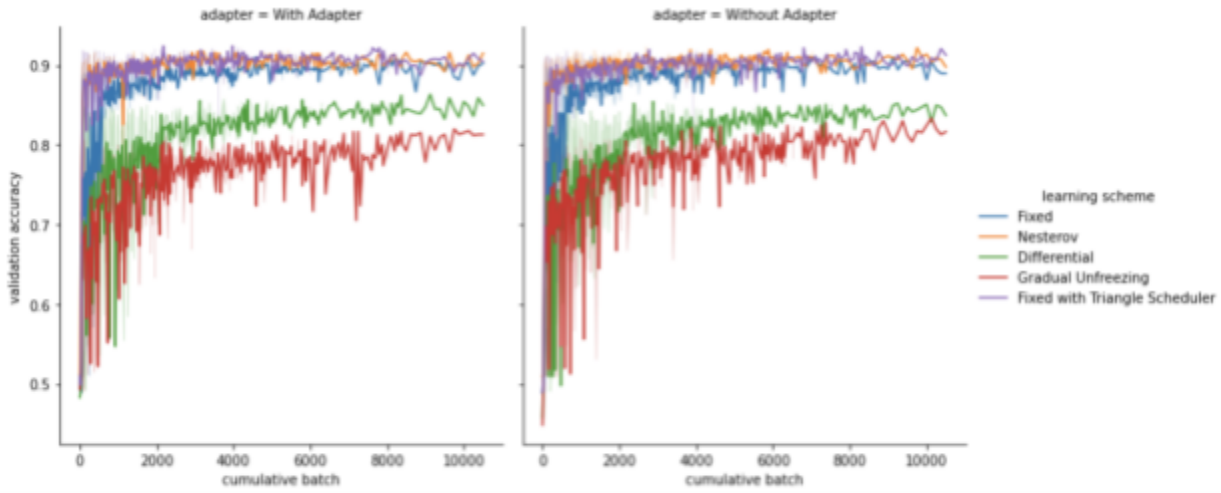

Figure E: Experiment Performance on SST-2 dataset

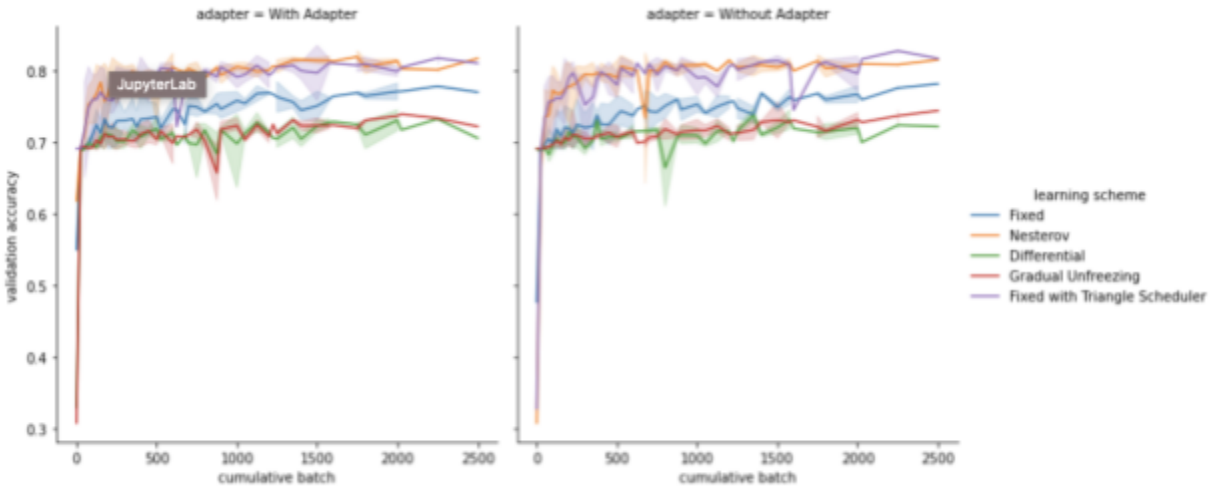

Figure F: Experiment Performance on CoLA dataset

As we can see from the tables and performance plots, the Fixed, Nesterov, and Fixed with Triangle Scheduler learning slightly outperformed the Differential Learning Rate and Gradual Unfreezing implementations, with and without an adapter added to the transformer. This can be explained by the fact that both Differential Learning Rates and Gradual Unfreezing both learn less or not at all in the early layers of the transformer, based on whether using DLR (very small

learning rates early in the network) or GU (frozen, or zero learning rate for all layers except the last n, where n equals the epoch we are on during training).

Regardless of these results, we were successful in developing and demonstrating a platform that allowed for flexible and rapid finetuning experimentation. With this platform as-is, as well as with future capabilities implemented, we could scale out the types of finetuning experiments possible for trying.

Conclusion

In conclusion, we successfully developed a testbed Python module for evaluating a variety of finetuning techniques. The manner in which we developed the package allows for iterating on the functionality of the module as it exists today. Some potential next steps would be to add additional learning rate approaches such as only finetuning the final classification layer, or mixing learning rate schedules with the Differential Learning Rate and Gradual Unfreezing approaches.

Due to the specificity of each Deep Learning system and the finetuning use case, one singular rule for how to best finetune does not exist. Because of this, we believe that the codebase we developed allows for automated finetuning experimentation in a rapid manner similar to how one might perform hyperparameter or neural architecture searches.

References

[1] How transferable are features in deep neural networks? - Yosinski et al
[2] Parameter Efficient Transfer Learning for NLP - Houlsby et al
[3] AdapterHub: A Framework for Adapting Transformers - Pfeiffer et al
[4] Universal Language Model Fine-tuning for Text Classification - Howard et al
[5] Cyclical Learning Rates for Training Neural Networks - Smith
[6] Stochastic Gradient Descent with Warm Restarts - Loshchilov, Hutter
[7] Super-Convergence - Smith
[8] Huggingface Transformers
[9] SST-2
[10] CoLA