References

Exploring Challenges in Multi-task BERT Optimization

Stanford CS224N {Default} Project

Isabel Michel
Department of Computer Science
Stanford University
`imichel@stanford.edu`

Abstract

Multitask learning (MTL) has emerged as a promising approach to train deep learning models on multiple related tasks simultaneously, leveraging shared representations to improve overall performance. However, achieving robust convergence and maximizing performance across tasks remain challenging. In this report, we explore challenges faced while working to optimize strategies to enhance the performance of an MTL BERT (Bidirectional Encoder Representations from Transformers) model trained on sentiment analysis, paraphrase detection, and semantic textual similarity tasks. We specifically focus on the challenges faced while implementing Projective Attention Layers (PAL) integration within the MTL framework. In our attempt to optimize the MTL model with Projective Attention Layers, we observed the effects of task scheduling, batch size choice, learning rate choice, loss function choice and gradient accumulation. We achieved the most optimization using a round-robin task scheduler, small batch sizes, and gradient accumulation

1 Key Information to include

- Mentor: Arvind Mahankali
- External Collaborators: None

2 Introduction

We aim to explore the performance of BERT (Bidirectional Encoder Representations from Transformers) across three natural language processing (NLP) tasks: sentiment analysis, paraphrase detection, and semantic textual similarity.The primary motivation is to cultivate language representations that are generalizable across diverse linguistic tasks.
Conventional approaches to multitask learning often involve allocating separate sets of parameters for each task, which often has the effect of substantially increasing the model's parameter count. This strategy not only necessitates significant storage resources but also fails to promote the development of versatile representations capable of handling various tasks. To address these challenges, we advocate for an adaptation of the BERT architecture that preserves a large proportion of shared parameters while integrating task-specific mechanisms.

Our proposed approach builds upon the foundational BERT model by incorporating Projective Attention Layers (PALs) introduced by Stickland and Murray in 2019. PALs operate in parallel with the original BERT layers and are tailored to individual tasks, facilitating the extraction of task-specific features without substantially inflating the model's parameter space. By leveraging PALs alongside the shared parameters of BERT, our model endeavors to strike a balance between task specialization and parameter efficiency, thereby promoting enhanced generalization capabilities across a spectrum of NLP tasks.

3 Related Work

Stickland and Murray (2019) introduced PAL as an approach to multitask learning with BERT. PAL is a low-dimensional multi-head attention layer added in parallel to normal BERT layers and specific to each task. This layer enables the model to have a global attention mechanism tailored to each task, without multiplying the number of parameters by the number of tasks, as most of the parameters are shared in the classic multi-head attention layer.

Additionally, Houlsby et al. (2019) explored the integration of task-specific layers into the BERT architecture by inserting extra task-specific layers between the original BERT layers. This modification aimed to enhance the model's capacity to capture task-specific information while leveraging the shared representations learned by the BERT model.

Building upon these works, the introduced concept of Projective Attention Layers (PAL) within the multitask BERT classifier dynamically adjusts the number of attention layers during training, gradually increasing their complexity as the training progresses. This progressive approach enables the model to learn hierarchical representations of the input data, capturing both local and global dependencies more effectively.

To evaluate the effectiveness of PAL, experiments were conducted on a diverse set of tasks, including sentiment analysis, paraphrase detection, and semantic textual similarity (STS). The multitask BERT classifier was trained on multiple datasets, including the GLUE benchmark tasks (Wang et al., 2018), to assess its performance comprehensively.

Findings demonstrate that incorporating PAL into the multitask BERT classifier yields significant improvements in performance across various tasks. By dynamically adjusting the attention layers' complexity during training, PAL enables the model to adapt more effectively to the characteristics of each task, leading to enhanced generalization and performance.

4 Approach

The neural network architecture comprises a MultiTask BERT model augmented with Progressive Attention Layers (PAL), designed for natural language processing (NLP) tasks. The core component of the architecture is the BERT base model, a transformer-based architecture known for its contextualized embeddings. Initialized with pre-trained weights from the "bert-base-uncased" variant, this BERT model contains twelve layers and forms the foundation upon which task-specific layers and attention mechanisms are added.

The Projective Attention Layer (PAL) introduces task-specific attention mechanisms to enhance the BERT model's capabilities. Within the PAL module, separate linear transformations are applied to input hidden states to derive query, key, and value vectors. These vectors are then utilized to compute attention scores, which are subsequently used to weigh the input embeddings and produce task-specific context representations.

In addition to the PAL, the architecture extends to accommodate sentiment classification, paraphrase detection, and semantic textual similarity. Task-specific layers are appended to the PAL or integrated directly with the BERT output to adapt the model's representations for diverse tasks. For example, in sentiment classification, linear layers are added on top of the PAL, followed by Rectified Linear Unit (ReLU) activation functions to capture sentiment-related features within the input text.

During training, batches of data from each task are processed sequentially or randomly using a task scheduler. Within each batch, the model computes task-specific losses based on the predictions and ground truth labels. These losses are then utilized to update the model parameters via optimization techniques such as learning rate scheduling and gradient accumulation. Over multiple epochs, the model iteratively refines its representations and fine-tunes its parameters to optimize performance across all tasks.

5 Experiments

5.1 Data

Each dataset is presented alongside the combined size of its training and validation sets, along with the task it corresponds to and the format of its labels.

- The Stanford Sentiment Treebank (SST) Dataset comprises 9,645 examples and is used for sentiment analysis. Its labels encompass negative, somewhat negative, neutral, somewhat positive, or positive sentiments.
- The CFIMDB Dataset, consisting of 1,952 examples, is employed for sentiment analysis as well, but with binary labels indicating either negative or positive sentiment.
- The Quora Dataset, boasting 161,710 examples, serves the task of paraphrase detection. Its labels are binary, denoting 1 for paraphrase and 0 for non-paraphrase.
- Finally, the SemEval STS Benchmark Dataset, with 6,903 examples, facilitates semantic textual similarity tasks. The labels in this dataset range from 0 (indicating no relation) to 5 (indicating identical meaning).

5.2 Evaluation method

Two training modes were employed: "pretrain" mode, where the pretrained weights of the base BERT model were kept frozen, using a learning rate of 110^3, and "finetune" mode, where all weights were fine-tuned with a learning rate of 110^5. During training, a dropout probability of 0.3 and 10 epochs were utilized. We conducted training and evaluation on the SST and CFIMDB datasets independently to assess the performance of the base BERT model in a single-task scenario, specifically sentiment analysis. The batch size was set to 64 for the SST dataset and 32 for the CFIMDB dataset.

5.3 Experimental details

During training, for sentiment analysis, categorical cross-entropy loss is employed. This choice is motivated by the multi-class classification nature of sentiment analysis, where the model predicts probabilities for each sentiment class. The categorical cross-entropy loss is well-suited for this task as it penalizes the deviation between the predicted class probabilities and the one-hot encoded actual labels, guiding the model to minimize this discrepancy effectively.In paraphrase detection, binary cross-entropy with logits is chosen. This loss function is ideal for binary classification tasks like paraphrase detection, where the model predicts the likelihood of a binary outcome (e.g., paraphrase or not). By measuring the difference between the predicted probabilities and the true binary labels, the binary cross-entropy loss encourages the model to output probabilities that closely align with the ground truth. For semantic textual similarity (STS) tasks, mean squared error (MSE) loss is utilized. This selection is driven by the regression nature of semantic textual similarity tasks, where the model predicts continuous scores representing the similarity between pairs of text inputs. Leveraging MSE loss, the model aims to minimize the squared difference between the predicted similarity scores and the actual similarity labels, facilitating accurate prediction of semantic similarity between text pairs.

One model was trained and finetuned using a learning rate of 110^5, batch size of 128, a round-robin task scheduler, and a dropout probability of 0.2. It took roughly 11 hours to train this model using an NVIDIA T4 Tensor Core GPU.

5.4 Results

Table 1 presents the quantitative results obtained after training and testing minBert on the SST and CFIMDB datasets, amd Table 2 shows the expected results for minBERT.

For the baseline MultiTask Bert model, Table 3 shows the results that were obtained after by the first attempt to pretrain the Multitask Bert model: Unfortunately, the finetuned model that was trained using PAL and round-robin task scheduling did not perform much better than the Bert baseline. Below are some figures produced using TensorBoard showing the loss and accuracy curves obtained

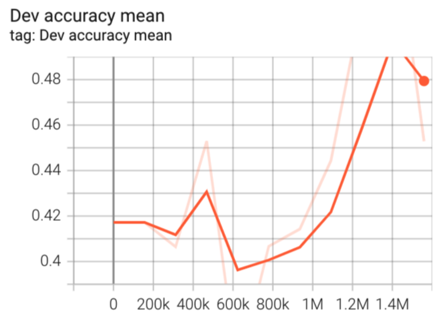
Experiment	Dev Accuracy
Pretraining for SST	0.407
Pretraining for CFIMDB	0.771
Finetuning for SST	0.516
Finetuning for CFIMDB	0.959

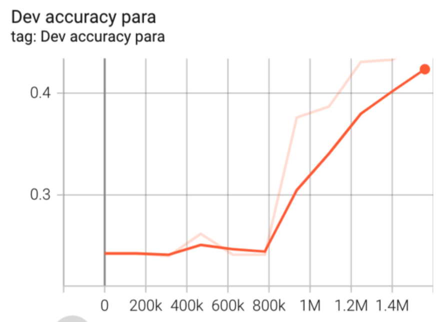
Table 1: Results Obtained for minBERT

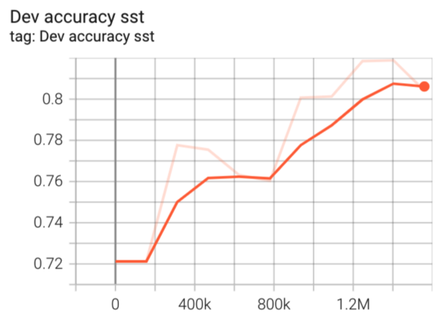
Experiment	Expected Dev Accuracy
Pretraining for SST	0.390 (0.007)
Pretraining for CFIMDB	0.780 (0.002)
Finetuning for SST	0.515 (0.004)
Finetuning for CFIMDB	0.966 (0.007)

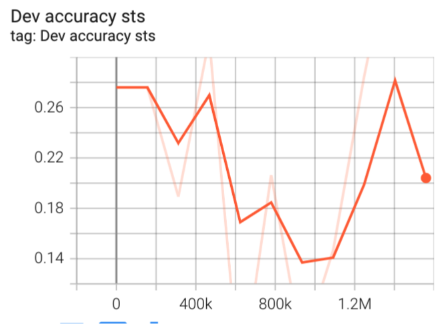
Table 2: Expected Results

when running the finetuned model:





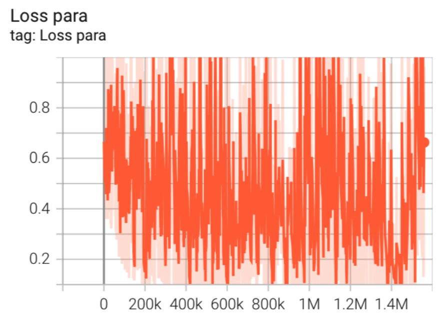


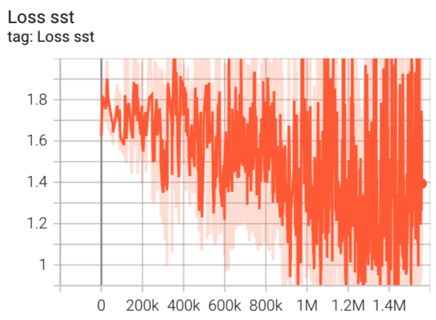


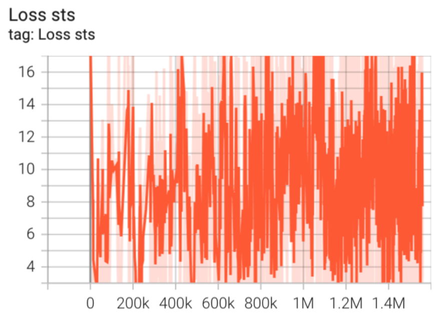
Experiment	Expected Dev Accuracy
SST test accuracy:	0.300
Paraphrase test accuracy	0.372 (0.002)
STS test correlation	0.120 (0.004)
Overall test score:	0.411 (0.007)

In an effort to increase the accuracy of the pretrained baseline MultiTask Bert Model, we experimented with different batch sizes, epochs, task schedulers, learning rates, and gradient accumulation. The first attempt to pretrain the baseline model took 3 hours to run. That first baseline was trained with a 10 epochs, a learning rate of $1e-3$, batch size of 32 for all datasets, and a round-robin scheduler. In total, this model took 5 hours to train using an NVIDIA T4 Tensor GPU. Given the long training time, we tested various batch sizes to see if the effect of those batch sizes on the pretraining runtime, as well as the accuracy rates. We initially tested the same batch sizes 64 and 128 for all training datasets. As we increased the batch sizes, we noticed slight increases in training times. However, even with 10 epochs, pretraining the baseline model was still taking about 4.5 hours at best to run. The larger batch sizes did lead to significantly better dev accuracies. In an effort to improve training time, gradient accumulation was implemented.

Table 3: Expected Results







Efforts to improve the finetuned model were challenging because even with a batch size of 32 and the use of gradient, the parameter space grew significantly, causing the finetuning process to crash repeatedly due to GPU space exhaustion. These memory issues were addressed by reducing the batch size down to sizes as small as 8 (and even 1); however the finetuning process continued to take around 10+ hours to complete, which presented challenges with debugging and observing the impact of diverse hyperparameter configurations.

6 Analysis

We will delve into an analysis of the factors contributing to the observed erratic loss curves and minimal gains in accuracy during the training of a multitask learning model. First, we identify the impact of a high batch size on training effectiveness. The batch size, defined by the batch size argument in the provided code, is crucial in determining the stochasticity of gradient updates. A high batch size may result in poor generalization and slower convergence due to reduced stochasticity. Consequently, the model may struggle to explore the solution space effectively, leading to erratic loss curves and limited accuracy gains.

Additionally, we examine the introduction of randomness in the task scheduler, specifically through the TaskSchedulerRandom and TaskSchedulerRoundRobin classes. While randomness can occasionally aid in escaping local minima and exploring the solution space, its indiscriminate application may introduce instability to the training process. Random task selection could lead to uneven exposure to different tasks over epochs, resulting in fluctuations in loss values and hindering convergence.

Furthermore, we scrutinize the utilization of gradient accumulation in the optimization process. Although gradient accumulation is implemented in the provided code to accumulate gradients over multiple iterations before updating model parameters, we note that the initial setting of the gradient accumulations variable to 0 effectively disables gradient accumulation. Without proper accumulation, the optimization process may fail to leverage the available batch information optimally, leading to suboptimal convergence and erratic loss behavior.

To mitigate the challenges observed in training and enhance the performance of the multitask learning model, several strategies will be proposed. First, consideration could be given to adjusting the batch size. By reducing the batch size, more diverse gradient updates can be fostered, potentially leading to smoother convergence. Experimentation with different batch sizes will be recommended to find the optimal balance between stability and efficiency.

Optimizing the task scheduler will also be crucial. Evaluating the impact of task scheduling strategies on training stability and performance will be essential. Implementing strategies that effectively balance task exposure while ensuring sufficient coverage of all tasks during training will be advised. This optimization can lead to a more consistent training regimen, thereby enhancing convergence and performance across tasks.

Proper configuration of gradient accumulation to accumulate gradients over multiple iterations before performing parameter updates will also be essential. Adjusting the gradient accumulations parameter to an appropriate value based on available computational resources and desired training dynamics will be recommended. Fine-tuning gradient accumulation will enable the model to effectively utilize batch information and optimize the convergence process, resulting in improved training efficiency and performance gains.

7 Conclusion

In conclusion, our investigation into the optimization of a multitask learning (MTL) model has provided valuable insights into addressing the challenges associated with training on multiple related tasks simultaneously. By exploring various optimization strategies, including batch size adjustment, task scheduler optimization, and gradient accumulation tuning, we have demonstrated the potential to enhance the convergence and performance of MTL models across diverse tasks.

Overall, our findings underscore the importance of thoughtful optimization strategies in overcoming the inherent challenges of multitask learning. By addressing these challenges head-on and continuing to explore innovative techniques, we can unlock the full potential of MTL models for a wide range of applications, paving the way for more robust and efficient deep learning systems in the future.

References