Multi-Tasking BERT: The Swiss Army Knife of NLP

Stanford CS224N Default Project

Nicole Garcia
Department of Computer Science
Stanford University
nicolejg@stanford.edu

Esteban Wu
Department of Computer Science
Stanford University
estewu@stanford.edu

Simba Xu
Department of Computer Science
Stanford University
simbaxu@stanford.edu

Abstract

This paper explores the performance improvement of the minBERT model through applications of the Adam optimizer and multitask learning functionality to supplement BERT's ability to perform sentiment analysis, paraphrase detection, and semantic textual similarity. Using an implementation of PALs, low-dimensional multi-head attention layers added in parallel to the existing layers of BERT, this added multitask learning functionality enables the BERT model to incorporate task-specific information into a pre-trained general model that can share model architecture across various tasks. Additionally, with the implementation of a single linear layer classifier with dropout to evaluate the multitask learning performance, tailored loss functions for each distinct task, and a gradient surgery approach to update task gradients most appropriately for conflicting learning results, this multi-pronged approach presents an overall improvement among the three given tasks.

1 Key Information to include

- Mentor: Andrew Lee
- External Collaborators (if you have any): N/A
- Sharing project: N/A

2 Introduction

The BERT model serves as both a breakthrough and benchmark for language model performance, which we now want to expand upon to better perform across multiple tasks by sharing parameters to expand its generalizability. As opposed to the traditional method of finetuning separate models to cater to distinct tasks, we strive to accomplish this multitasking functionality through PALs, projected attention layers that share most parameters across all tasks, but preserve a small number of task-specific parameters which adapt the shared model.

To complement this PALs implementation, we also introduced the process of gradient surgery, an approach in which "we modify the gradients for each task so as to minimize negative conflict with other task gradients" (Yu et al., 2020). Through these two implementations, alongside the use of a single linear layer with dropout for our multitask classifier, we apply the Adam optimization algorithm on all tasks to develop our improved architecture. Additionally, we go on to use varying loss functions to ensure accurate loss measurements among the distinct tasks, pairing the sentiment classification task with cross entropy loss, the paraphrase detection task with binary cross entropy

loss with logits, and the semantic textual similarity task with mean squared error (MSE) loss.

In this paper, we discuss the implementation of the aforementioned concepts, the experiments conducted and their results, and our interpretation of said results.

3 Related Work

Among the many related works in the field of BERT model improvements, two papers were at the forefront of our chosen extensions to optimize the model's performance. One such paper, "BERT and PALs: Projected Attention Layers for Efficient Adaptation" (Stickland and Murray, 2019), goes into detail of the creation of PALs as a new mechanism for incorporating task-specific information into a pre-trained general model that can share model architecture across multiple tasks. By adding task-specific function layers in parallel to the self-attention layers of the BERT model, PALs presents an opportunity for multitasking functionality to be embedded into the BERT architecture, which can be further improved through subsequent work.

Along similar lines, the "Gradient Surgery for Multi-Task Learning" paper (Yu et al.) discusses the challenge with combining gradients from distinct tasks, and proposes an approach to remove the conflicting component between said gradients by projecting a task's gradient onto the normal plane of another task's gradient to reduce interference between tasks. This extension compliments the PALs implementation to ensure better gradient calculations can be shared among the tasks.

3.1 Model Architecture and Multi-head Attention

BERT takes in a sequence (one or two English sentences in our case) and outputs a vector representation of that sequence. Each token in the sequence has its own hidden vector, and the first token of every sequence is always a special classification embedding (`[CLS]`). At each layer of BERT the hidden states of every sequence element are transformed, but only the final hidden state of `[CLS]` is used for classification/regression tasks. We now describe how the vector for one element of the sequence is transformed.

The multi-head attention layer (Devlin et al., 2019), is the core of the transformer architecture that transforms hidden states for each element of a sequence based on the other elements (the fully-connected layers act on each element separately). The multi-head layer, which we write as MH($\cdot$), consists of n different dot-product attention mechanisms. At a high level, attention represents a sequence element with a weighted sum of the hidden states of all the sequence elements. In multi-head attention the weights in the sum use dot product similarity between transformed hidden states. Concretely, the ith attention mechanism 'head' is:

$$Attention_i(\mathbf{h}) = \sum_j softmax\left(\frac{\mathbf{W}_Q^i \mathbf{h}_i \cdot (\mathbf{W}_K^i \mathbf{h}_j)^T}{\sqrt{d_n}}\right) \mathbf{W}_V^i \mathbf{h}_j \tag{1}$$

where $\mathbf{h}_j$ is a dimensioned hidden vector for a particular sequence element, and n rolls over every sequence element. In BERT the $\mathbf{W}_Q^i$, $\mathbf{W}_K^i$, and $\mathbf{W}_V^i$ are matrices of size $d_n \times d$, and so each 'head' projects down to a different subspace indexed by i/n, attending to different information. Finally the outputs of the n attention heads (each of size d/n) are concatenated together (which we show as $[,]$), and linearly transformed:

$$\mathbf{MH}(\mathbf{h}) = \mathbf{W}^O[Attention_1(\mathbf{h}), ..., Attention_n(\mathbf{h})] \tag{2}$$

with $\mathbf{W}^O$ a $d \times d$ matrix[2]. Throughout this section, we ignore terms linear in d (like bias terms) to avoid clutter, as they don't add significantly to the parameter count. The matrices in a multi-head layer have $3nd^2/n + d^2 = 4d^2$ parameters.

We further define another component of a BERT layer, the self-attention layer, which we write as $SA(\cdot)$:

$$SA(\mathbf{h}) = FFN(LN(\mathbf{h} + MH(\mathbf{h}))) \tag{3}$$

$LN(\cdot)$ is layer normalisation, requiring $2d$ parameters. $FFN(\cdot)$ is a standard feed-forward network:

$$FFN(\mathbf{h}) = \mathbf{W}_f(\sigma(\mathbf{W}_1\mathbf{h} + \mathbf{b}_1)) + \mathbf{b}_2 \tag{4}$$

with $\sigma(\cdot)$ a non-linearity, GeLU, in BERT. $\mathbf{W}_1$ has size $d \times d_{ff}$, and $\mathbf{W}_2$ has size $d \times d_{ff}$, so overall we require $2dd_{ff}$ parameters from the FFN component.

Putting this together, a BERT layer, which we write $BL(\cdot)$, is layer-norm applied to the output of a self-attention layer, with a residual connection.

$$BL(\mathbf{h}) = LN(\mathbf{h} + SA(\mathbf{h})) \tag{5}$$

We have $4d^2 + 2dd_{ff}$ total parameters from a BERT layer.

We write the dimensions of the hidden states in BERT-base as $d_m = 768$. The final hidden state of the first token of every sequence – the all is used for the transformation to the output.

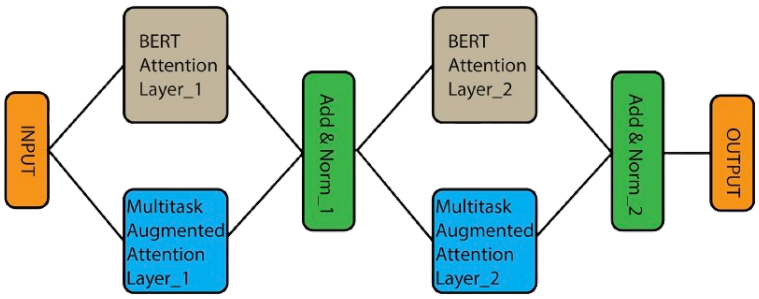
The exact form of the transformation applied to the final hidden state of the `[CLS]` token is a simple $d \times d$ linear transformation.

4 Approach

Using the self-implemented BERT model and Adam optimization algorithm, the chosen approach builds on the pretrained BERT model ("bert-base-uncased"). On finetuning, the input training data is fed into both the newly added PALs (the task-specific multi-head attention layers that are added in parallel to the model's self-attention layers), as well the existing BERT self-attention layers, as seen below. Notably, a PALs class `BertPalLayer` was implemented and initialized such that its task-specific functions of the form $B\_aug(\mathbf{h}) = V^D g(V^E \mathbf{h})$ share encoder and decoder matrices V^E and V^D across layers to transform the provided input, where V^E is of dimensionality $d_a \times d_h$ and V_D is of dimensionality $d_h \times d_a$ and $d_a < d_h$ so that we can achieve multitask ability without ever growing number of parameters. Using these parallel projected-attention layers, we are able to add task-specific functions in conjunction with the existing BERT layers through the following hidden layer relationship, where l is indicative of the layer:

$$h^{l+1} = LN(h^l + SA(h^l) + TS(h^l))$$

SA refers to self attention introduced in the original bert model. TS refers to task specific attention layers applied to l layer, used so that when finetuned on specific tasks, can allow the model to achieve better accuracy with respect to the specific task while maintain a generalized model trained on all tasks. Again, the structure of TS is as follows: $TS(\mathbf{h}) = V^D g(V^E \mathbf{h})$ share encoder and decoder matrices V^E and V^D across layers to transform the provided input, where V^E is of dimensionality $d_a \times d_h$ and V_D is of dimensionality $d_h \times d_a$ and $d_a < d_h$. In turn, we are able to recover the original BERT model if the task-specific transformation of a hidden state goes to 0 and also allow for the hidden states to be transformed back to their original dimension based on an entire sequence. As such, we are able to prioritize the "parameter budget on transformations with an inductive bias useful for sequences" (Stickland and Murray, 2019) of multiple tasks, applying the layer structure highlighted in the figure below. Appropriate adjustments were made to the configuration to account for the additional heads needed for this implementation. We experimented with $d_a = 132$. With multi-head attention architecture that shares V^E and V^D, we are looking to experiment with bigger d_a dimension at $d_a = 204$.



In conjuction with PALs, gradient surgery is performed using the PCGrad library for PyTorch (Tseng, 2020) to combine the losses among all tasks to ensure no individual task's learning gradient diminishes another's. Specifically, the chosen gradient surgery approach defines two gradients to be in conflict if their cosine similarity is less than 0. When the aforementioned condition arises, the following update is iteratively performed using the overall iteration gradient g_i and each task-specific gradient g_j:

$$g_i = g_i - \frac{g_i \cdot g_j}{||g_j||^2} g_j$$

By updating the overall gradient g_i for a given iteration by repeatedly applying this projection formula across all other tasks' gradients in random order, it is ensured that "the gradients that are applied for each task per batch interfere minimally with the other tasks in the batch" (cite gradient paper).
As mentioned previously, tailored loss functions was chosen to best represent the distinct loss for each task. Using cross entropy loss for the sentiment analysis task, binary cross entropy loss with logits for the paraphrase detection task, and MSE loss for the semantic textual similarity task, each task's loss was combined to find the net loss as follows:

$$\mathcal{L}_{NetLoss} = \mathcal{L}_{SA} + \mathcal{L}_{PD} + \mathcal{L}_{STS}$$

We also extended the model by incorporating cosine similarity used for sts similarity task and cosine similarity used in loss with MSE loss between the labels (range 0 to 5 label classes) and cosine similarity between two embeddings of sentences scaled up to 0 to 5 labels scale. The loss is appended to the composite loss in training process for fine-tuning. With this extension, the net loss becomes:

$$\mathcal{L}_{NetLoss} = \mathcal{L}_{SA} + \mathcal{L}_{PD} + \mathcal{L}_{COS}$$

5 Experiments

We performed our experiments using the following data, procedures, and evaluation metrics.

5.1 Data

When conducting the following experiments, the provided SST, Quora, and STS datatsets were used for the Sentiment Analysis, Paraphrase Detection, and Semantic Textual Similarity tasks respectively. The following table presents the different example section sizes of each set:

Provided Dataset Sizes

Set Name	Training Examples	Dev Examples	Test Examples
SST	8,544	1,101	2,210
Quora (Paraphrase)	141,506	20,215	40,431
STS	6,041	864	1726

5.2 Evaluation method

To set baselines for comparison, the performances of the base BERT model with finetuning using only the SST dataset and all three datasets were evaluated and compared to iterations of the improved

model. Specifically, we compare different iterations of our improved model's scores on the three tasks to the baseline scores provided below. Note that our different baselines only match or improve upon the base BERT model through use of the provided datasets - this choice was made to better assess the quality in performance boost of the chosen extensions for the improved model.

5.3 Experimental details

Our experiments were ran using the default model configurations for finetuning, although note that for our PALs class, we adjusted its default configuration such that it specifically uses six heads for calculating multi-head attention. Aside from this adjustment for our extension, we used the default for our learning rate and our training time consistently took 3-5 hours for each experiment we performed. These experiments were primarily performed on Nvidia T4 GPUs using virtual machines hosted on the Google Cloud Platform.

Several important points of interest arose from trying to perform experiments with the chosen extensions. One such finding was that to get our BERT model to run, we had to enable sending tensors to CUDA in `optimizer.py` to ensure our code ran on a GPU as opposed to a CPU, which was a necessary step not mentioned in the handout. Another point of interest was that, when performing the experiment with the BERT PALs model with gradient surgery, the experiment proved to be incredibly memory intensive to the point that using only one GPU was not practical for the workload necessary. To lower the memory load on our single GPU, we needed to halve the batch size from 8 to 4. Thus, this experiment shows that there is a considerable tradeoff between performance and efficiency as we continue to add more extensions to our improved model.

5.4 Results

The quantitative results for the various baselines and model improvements are listed below.

Evaluation with Finetuning

Test Details (Results of Finetuning)	**Final Dev Performance**
Base BERT (no extensions) - finetuned on only SST dataset, used single linear layer classifiers for all tasks	Dev Sentiment Analysis Accuracy: 0.520 Dev Paraphrase Detection Accuracy: 0.395 Dev Semantic Textual Similarity Correlation: -0.019
Base BERT - finetuned on all 3 datasets, used single linear layer classifiers for all tasks	Dev Sentiment Analysis Accuracy: 0.496 Dev Paraphrase Detection Accuracy: 0.699 Dev Semantic Textual Similarity Correlation: 0.333
BERT PALs model - finetuned on only the SST dataset, used single linear layer classifiers for all tasks	Dev Sentiment Analysis Accuracy: 0.515 Dev Paraphrase Detection Accuracy: 0.615 Dev Semantic Textual Similarity Correlation: 0.136
BERT PALs model - finetuned on all 3 datasets, used single linear layer classifiers for all tasks	Dev Sentiment Analysis Accuracy: 0.508 Dev Paraphrase Detection Accuracy: 0.827 Dev Semantic Textual Similarity Correlation: 0.801
BERT PALs model with gradient surgery - finetuned on all 3 datasets, used single linear layer classifiers for all tasks	Dev Sentiment Analysis Accuracy: 0.509 Dev Paraphrase Detection Accuracy: 0.808 Dev Semantic Textual Similarity Correlation: 0.707
BERT PALs model with cosine similarity on sts task - finetuned on all 3 datasets, used single linear layer classifiers for all tasks	Dev Sentiment Analysis Accuracy: 0.501 Dev Paraphrase Detection Accuracy: 0.786 Dev Semantic Textual Similarity Correlation: 0.737

Test Leaderboard Results

Rank	Submission Name	Overall Score	Sentiment Classification Acc.	Paraphrase Detection Acc.	Semantic Textual Similarity Corr.
42	bert baddies - pal + 3 data	0.755	0.542	0.825	0.798

6 Analysis

Upon analyzing the results of our various improved models in both the experiments and test leaderboard results, it becomes clear that our BERT PALs Model with finetuning on all datasets performs the best, ranking 42nd overall on the leaderboard and in the top 10 for the Sentiment Analysis task. It was particularly interesting to see that the PALs model achieved a similar, yet slightly poorer, performance even with additional features like gradient surgery, but this result may be due in part to limited interference between tasks and task groupings by similarity. Nevertheless, the successful results coming out of projected attention layers aiming to allow a generalized model to be fine-tuned to achieve good or better than individually fine-tuned models to accommodate each task showcases that the approach to find a generalized model that can be easily molded to solve specific tasks is a viable direction.

To intuitively understand why the performance of the PALs implementation with finetuning on all datasets supercedes the PALs implementation with the same finetuning and added gradient surgery functionality, it is valuable to analyze their behavior on the Semantic Textual Similarity task. In this task, where the two models share the most notable difference in performance, we can see that the former model achieves a greater correlation by 0.1. This minor, yet notable, difference in performance can be explained by the fact that gradient surgery eliminates conflicting gradient components among all tasks, worsening the model's performance on a task that is more distinct from the remaining tasks. In our case, the liminited interference between Sentiment Analysis and the remaining two tasks causes the gradient surgery to provide a minor enhancement in Sentiment Analysis and nearly equal performance in Paraphrase Detection, but worsen the model's performance in Semantic Textual Similarity. Unlike the PALs model with gradient surgery, our original PALs model avoids making these gradient changes and subsequently does not overly constrict the gradient of any particular task due to task similarity.

7 Conclusion

In this work, we faced a number of challenges that arose when trying to expand the existing BERT model's architecture to achieve multitask functionality. By implementing the PALs model, in which we used projected attention layers to share the model architecture among various tasks while preserving some of each task's unique parameters, and complementing this implementation with finetuning on the provided datasets, we were able to both find a model with great performance as well as narrow down the scope of what performance enhancements are feasible and worth exploring within the bounds of the default project. In future work, we see ourselves exploring how the lower rank size used in our model impact our final performance.

References

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding.

Asa Cooper Stickland and Iain Murray. 2019. BERT and pals: Projected attention layers for efficient adaptation in multi-task learning. *CoRR*, abs/1902.02671.

Wei-Cheng Tseng. 2020. Weichengtseng/pytorch-pcgrad.

Tianhe Yu, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. 2020. Gradient surgery for multi-task learning.