MinBERT and PALs: Multi-Task Leaning for Downstream Tasks

Stanford CS224N Default Project

Name
Department of Computer Science
Stanford University
tethay@stanford.edu

Abstract

Previously, state-of-the-art results in various NLP tasks were achieved by fine-tuning separate BERT models for each task. However, this method is inefficient in resource utilization because it requires a separate model for each task. Consequently, there's motivation to train a single model for multiple downstream tasks to address these challenges. In this project, we aims to optimize the resource utilization and performance of a pre-trained BERT model across three distinct tasks: sentiment analysis, paraphrase detection, and semantic textual similarity. These optimizations are achieved through the implementation of (i) PALs (Projected Attention Layers) which are small, task specific, multi-head attention layers placed in parallel to BERT layers, (ii) applying varying degrees of sampling across tasks during training. We did not find conclusive evidence which suggested cross-task interference across these three tasks. In fact, PALs alone resulted in a decline in overall performance, yet it exhibited improvement when combined with sampling methods.

1 Key Information to include

- Mentor: None
- External Collaborators (if you have any): None
- Sharing project: None

2 Introduction

In this project, we will adopt a large language model BERT(the Bidirectional Encoder Representations from Transformers model) for three downstream tasks. Built on the Transformers architecture, BERT captures bidirectional context through techniques like the masked language model and next sentence prediction during pre-training, this pre-trained model is suited for many natural language tasks. Suppose we want to adopt this model for multiple downstream tasks. One way to achieve this is to train a single model for each task, but this requires storing all the parameters(one full model) per task. To reduce this storage inefficiency, you can train a base model simultaneously on different tasks where tasks share all parameters except a small number of task-specific classification parameters. This is an important consideration when running models on machines that have more restrictive hardware constrains(e.g. mobile devices).

However, adopting multi-task learning without careful planning can end up lowering performance since tasks might interfere with each other and drag down overall results. PALs(Projeccted Attention Layer) by (Stickland and Murray, 2019) is one of many proposed techniques that may help overcome this challenge. PALs leverages additional projected layer with multi-head self attention in parallel to BERT layers for each task in order to reduce the total number of parameters. We will implement and

explore this architecture by adopting a miniBERT base model for sentiment classification, paraphrase detection, and evaluation of semantic textual similarity (STS). Amongst a few variations of PALs architecture, we have implemented one version and attempted various sampling methods. Annealed sampling is one such proposed sampling method tailored to accommodate varying dataset sizes, while also ensuring a more equitable distribution of sampling probabilities as epochs progress. Please see section 3.2 for the motivation and more details on this approach.

Our findings suggest that the advantages of leveraging the PALs architecture may be more pronounced when training a model on a larger number of tasks rather than just three. Additionally, we've observed that the method of sampling across tasks significantly influences overall performance with PALs architecture. This paper aims to investigate the efficiency of the PALs model architecture in terms of space and analyze how different sampling strategies impact final results in comparison to both task-specific minBERT and a minBERT trained on all tasks, which serve as our benchmarks.

3 Related Work

There have been many proposed solutions to adopt a single model to learn multiple tasks at the same time. There are many benefits to this, as one of the many motivations may be efficiency. In hardware-constrained environments, minimizing the model's size while maintaining performance becomes a natural objective. Other motivations, such as "Transfer Learning," also play a significant role, wherein a single model trained across multiple tasks can leverage knowledge acquired from diverse tasks, potentially enhancing the model's generalizability due to the breadth of data and tasks it encounters during training.

Various techniques have been explored for incorporating adaptation parameters. Learning hidden unit contributions (Swietojanski and Renals, 2014) involves modifying a neural network by multiplying each hidden unit by a trainable scalar, which adds a small number of parameters compared to other methods. Residual adapter modules (Rebuffi and Vedaldi, 2018) are utilized for multi-task learning in computer vision, adapting large pre-trained residual networks (He and Sun, 2016). Each adapter module features a 1x1 filter bank with a skip connection, which can be inserted either in series between the original network layers or in parallel as additional inputs to a layer.

Several multi-task learning strategies fall into two main categories: 'hard parameter sharing' and 'soft parameter sharing.' In the hard parameter sharing approach, shared hidden layers are employed for all tasks, while task-specific output layers are utilized. On the other hand, soft parameter sharing assigns each task its own model, but the distances between the models' parameters are regulated to promote similarity. Notably, various regularization methods are applied for this purpose, such as L2 distance as demonstrated by (Duong and Cook, 2015) and the trace norm utilized by (Yang and Hospedales, 2017).

The approach in (Stickland and Murray, 2019), which we adapted for this project, can be considered a version of hard parameter-sharing. In this approach, the primary BERT layers, which constitute the majority of the model's parameters, are shared among tasks and supplemented by minor task-specific PALs. In this project, we will look at both hard parameter sharing with minBERT only and minBERT with PALs closely and explore how they may achieve the over all performance of the models.

4 Approach

4.1 miniMERT for multi-task

In the first section of the Default Final Project(DFP) handout, we implemented miniBERT which is a scaled-down version of the original BERT implementations. miniBERT consists of an input embedding layer followed by 12 encoder blocks stacked on top of each other with hidden states of a dimension size *d*. Each encoder block sequentially implements (1) a multi-head attention layer, (2) Add & Norm layer, (3) feed forward network(FFN) layer, and (4) another Add & Norm layer.(see figure 1).
Succinctly, you can write:

$$MHLN_l(h) = LN_{l,1}\left(h, MH_l(h)\right) \tag{1}$$

$$BL_l(h) = LN_{l,2}\left(MHLN_l(h), FFN_l(MHLN_l(h))\right) \tag{2}$$

where h is a hidden state of dimension d, $MHLN$ is an output of the first 2 layers(***M***ulti-***H***ead and ***L***ayer ***N***orm), LN is a layer norm, MH is a multi-head attention layer, FFN is a feed-forward network and BL is a BERT Layer encoder block.

The final BERT layer of encoder blocks outputs the last_hidden_state, representing the contextualized embedding for each word piece of the sentence and pooler_output, which signifies the embedding of the [CLS] token. The [CLS] token is used to represent the entire input sequence when performing tasks like text classification represented by pooler_output and that's what we use for our tasks.

In sentiment classification, where a single sentence serves as input, we utilize a linear layer applied to the output embedding of the [CLS] token. This process yields a score corresponding to each sentiment label, from which we select the label associated with the highest score.

$$\hat{y}_{\text{sent}} = argmax(h1W_{sent} + b_1) \qquad W_{sent} \in \mathbb{R}^{c\times d}, b_1 \in \mathbb{R}^{c\times 1} \tag{3}$$

$$\hat{y}_{\text{para}} = \sigma([h_1, h_2]W_{para} + b_2) \qquad W_{para} \in \mathbb{R}^{2d\times d}, b_1 \in \mathbb{R}^{1\times 1} \tag{4}$$

$$\hat{y}_{\text{sts}} = \sigma(sim(h_1, h_2)) * 5 \tag{5}$$

where h1, h2 are the [CLS] output embeddings of our input sentences, c is the number of labels for sentiment classification, σ is the sigmoid function, W and b are linear layers and bias terms respectively.

4.2 PALs(Projected Attention Layers)

Our implementation closely adheres to the methodology outlined in the original paper, with certain code excerpts sourced from https://github.com/AsaCooperStickland/Bert-n-Pals, aimed at maintaining fidelity to the original design while adapting them to align with our BERT implementations. At the heart of this architecture lies the PALs Layer, which employs a linear layer V_E to (1) encode the hidden state of dimension d from a BERT layer and project it into a smaller dimension s where s < d, (2) subsequently passing through a multi-head attention layer, and (3) another linear layer V_D projecting the output back to the original dimension d, followed by a non-linearity (in this instance, GELU). The PALs layer is implemented concurrently at each BERT Layer and is shared across all layers to further diminish parameter sizes. The final output is then incorporated into the conventional layer normalization, as depicted in Equation (2). From Stickland and Murray (2019), the task-specific function forms:

$$TS(h) = V^{\text{D}} g(V^{\text{E}} h) \tag{6}$$

$$BL_l(h) = LN_{l,2}\left(MHLN_l(h), FFN_l(MHLN_l(h)), TS(h)\right) \tag{7}$$

where V^{E} is a d_s x d_m 'encoder' matrix, V^{D} is a dm x ds 'decoder' matrix with $d_s < d_m$, and $g(\cdot)$ is multi-head attention. This lower dimensional, multi-head attention layer operates in parallel to the usual multi-head and FFN within each BERT layer. The output from PALs layer and BERT layer output at Equation (1) are combined and normalized before being forwarded to the subsequent BERT layer. It's important to note that the "encoding" and "decoding" layers V_E and V_D are shared across all BERT layers for a given task but they vary across different tasks. This low dimensional spaces specifically allocated for each task is the core of the architecture design decision by Stickland and Murray(2019) which helps retain performance for each task while it reduces the number of parameters needed.

Equation (6) operates in dimension s rather than d where V^E and V^D have $s \cdot d$ parameters each while the multi-head parameters Q_l, K_l, V_l have s^2 parameters respectively. Across 12 layers(without bias terms), each task only requires $2sd + 12(3s^2)$ parameters for PALs. Let's now consider each BERT layer. Each BERT layer has multi-head which has a size of d^2 for each Q_l, K_l, V_l, plus a linear layer for the attention W^o. FFN_l contains intermediate of size di and output layers of size d. This is $12(4d^2 + 2dd_i)$. This highlights the effectiveness of PALs, which necessitate a significantly lower number of parameters for each task in comparison to the single-model-per-task strategy.

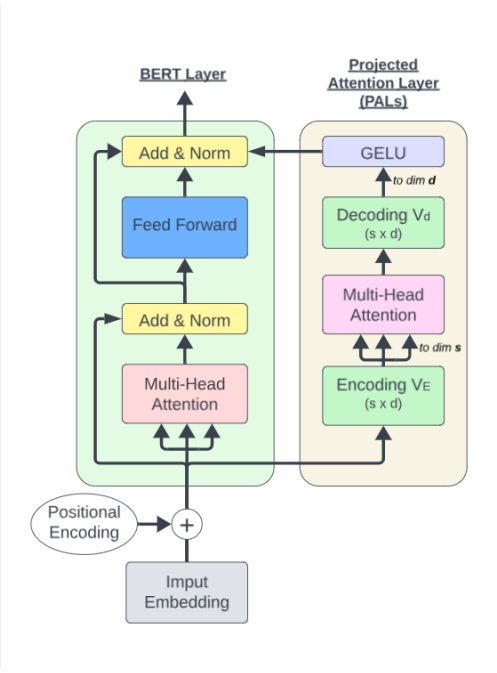

Figure 1: Projected Attention Layer Architecture

4.3 Task Sampling

One challenge you face when you are training a model for multiple task is to know how to sample training data across tasks. One approach is a "round robin" approach wherer you sample equal number of examples from each task. This approach risks overfitting on smaller datasets while underfitting the larger ones. In order to avoid this, Stickland and Murray (2019) uses the **"square root sampling"** method, which picks a task with a probability p_i that is proportional to the size of each dataset N_i raised to a power α:

$$p_i \propto N_i^{\alpha}, \qquad \alpha < 1 \tag{8}$$

where $\alpha = 1$("round-robin") samples equally from each task while $\alpha = 0$("proportional-sampling") samples each task proportionally to its dataset size. The rationale behind this approach is to mitigate disparities in sampling frequencies caused by variations in training data sizes across different tasks. Consequently, this method attempts to mitigate underfitting on tasks with smaller datasets and overfitting on tasks with larger datasets, ensuring a more balanced training process.

Stickland and Murray (2019) also highlights the advantages of training tasks more evenly towards the conclusion of the training process(**"annealed sampling method"**). They propose adjusting the parameter α with each epoch e:

$$\alpha = 1 - 0.8\frac{e-1}{E-1} \tag{9}$$

where E is the total number of epochs.

4.4 Further Pretraining

Exploring alternative methods to enhance model performance, we challenged to incorporate additional pretraining techniques, guided by the suggestions outlined in the Default Final Project Handout and inspired by the masked-language-model. The implementation exclusively utilized training datasets for the three tasks. For Quora and STS, only the first sentence in the pairs was considered. Additionally, random token masking was applied to the inputs, and the model was optimized using cross-entropy loss. This process was applied both before and after training the best model to evaluate the effectiveness of this additional step.

5 Experiments

5.1 Data

1. Stanford Sentiment Treebank (SST)6 : We use SST for sentiment classification. SST consists of 11,855 single sentences from movie reviews. Each sentence has one of five labels: 0 (negative), 1 (somewhat negative), 2 (neutral), 3 (somewhat positive), and 4 (positive).

2. Quora Question Pairs (QQP)7 : We use QQP for paraphrase detection. QQP consists of 400,000 question pairs. Each pair has a binary label indicating whether the two questions are paraphrases of each other.

3. SemEval STS Benchmark (Agirre et al., 2013): We use the SemEval STS dataset for semantic textual similarity. The SemEval STS dataset consists of 8,628 sentence pairs of varying semantic similarity on a scale from 0 (unrelated) to 5 (same meaning). The similarity scores have a granularity of 0.2, i.e. the possible similarity scores are 0, 0.2, 0.4, etc.

5.2 Experimental details

In our experiments, we exclusively employed fine-tuning and trained a total of 10 distinct models. Our experimentation involved three main approaches: (1) utilizing minBERT exclusively, with individual models created for each task initially aimed at establishing upper limit scores for each task, (2) incorporating PALs into minBERT, and (3) employing three different sampling methods: Round-robin, Proportional, and Annealed, for both minBERT only and minBERT with PALs. Additionally, we determined the step calculations such that within each epoch, the training process covers 50% of the data available in the training datasets. Consequently, we opted for 1950 steps per epoch. We selected a batch size of 32, resulting in approximately 5850 batches per epoch. Our optimizer was AdamW with a learning rate set to 1e-5 and weight decay of 0.01. As for our loss functions, we employed cross entropy for sentiment classification, binary cross entropy for paraphrase detection, and cosine similarity with sigmoid for STS (Semantic Textual Similarity).

In employing PALs, we utilized 12 attention heads with a dimension of s = 204. By applying the equations outlined in section 4.2 to calculate parameter counts, we determined that a PALs layer requires only 1.8 million parameters, whereas a BERT Layer consumes over 42 million parameters. In our implementations, we tackled 3 distinct tasks, each requiring 1.8 million parameters, resulting in a total of approximately 5.4 million parameters. This stands in contrast to the 127 million parameters of a single BERT model, representing roughly 1.13 times the number of parameters.

5.3 Results

Table 1: PALs dimension s=204, dropout=0.3, batch size=40, training steps=1950

Model	**Task Sampling**	**SST**	**Quora**	**STS**	**Avg.**
minBERT only	Fine-tune on each task	0.523	0.780	0.521	0.608
minBERT only	Annealed	0.495	0.771	0.502	0.589
minBERT only	Round-robin (α = 0)	0.507	0.769	0.508	0.595
minBERT only	Proportional (α = 1)	0.529	0.777	0.472	0.593
minBERT with PALs	Annealed (α = Equation(9))	0.519	0.775	0.448	0.580
minBERT with PALs	Round-robin (α = 0)	0.489	0.757	0.429	0.558
minBERT with PALs	Proportional (α = 1)	0.509	0.779	0.489	0.592
minBERT with PALs	Annealed with s=326	0.500	0.780	0.488	0.589

Table 2: Table 2: PALs dimension s=204, dropout=0.3, batch size=40, training steps=1950

Model	**SST**	**Quora**	**STS**	**Avg.**
minBERT only	0.507	0.769	0.508	0.595
minBERT with PALs	0.509	0.779	0.489	0.592

In my benchmark, minBERT models fine-tuned individually for each task attained the highest scores, serving as the upper limits, and consistently maintained their highest scores among all models. Unexpectedly, minBERT trained on all tasks, serving as the lower limits, secured the second position in these experiments. Remarkably, minBERT trained solely with the Round-robin sampling method achieved the highest score among all sampling methods. Despite the perceived specialization capability offered by the task-specific layers, all sampling methods employed with minBERT with PALs concluded with the lowest scores. We bumped dimension s from 204 to 324 and we saw higher STS scores but the average score remained almost unchanged. As we explored additional extensions for improving our scores, we experimented with integrating a POS Tagger using NLTK and further pretraining methods outlined in section 4.4. However, these attempts did not yield performance improvements; instead, they exacerbated the performance, so we have omitted them from this report.

6 Analysis

The design objective of the PALs architecture is to balance the total number of extra parameters while still achieving satisfactory results. Our findings were inconclusive with respect to these assumptions and there is room for further improvements based on the observed results. For instance, employing minBERT with PALs and the Annealed Sampling method yielded results somewhat comparable to our benchmarks, which were achieved using separate BERT models for each task although higher STS score is much to be desired. Surprisingly, when comparing "minBERT only" to "minBERT with PALs", "minBERT only" exhibited higher overall scores, contrary to our initial expectations. This unexpected outcome might be attributed to our limited use of only 3 tasks, unlike the (Stickland and Murray, 2019)'s testing on 8 GLUE tasks where greater task interference might be anticipated in training a single model. Another factor contributing to the somewhat subdued performance of PALs could be due to the introduction of non-pretrained weights by PALs layers. The pretrained weights of minBERT are seemingly more robust in preventing overfitting, whereas PALs layers may quickly over-adopt itself to the in-domain datasets, thereby diminishing overall performance.

Our results also indicate that the method of sampling each task can impact performance, particularly in preventing overfitting on tasks with smaller datasets and underfitting on tasks with larger datasets. Specifically, we found that the round-robin sampling method consistently yielded poorer outcomes for PALs, leading the model to overfit the training datasets for tasks like SST and STS. With SST and STS, training accuracies exceeded 90% only after a few epochs, while the performance on the development set deteriorated or flattened over subsequent epochs. During our experiments, Proportional sampling unexpectedly yielded the one of the highest scores, surpassing the Annealed method in both the miniBERT only and miniBERT with PALs setups. This underscores the significance of sampling based on probability proportional to data sizes in stabilizing the model training process. Lastly, increasing the dimension size s from 204 to 324 in PALs contributed to negligible amount of increase in overall performance. However, it was not able to overcome the overfit problem.

7 Conclusion

We conclude that architectures incorporating task-specific layers alongside BERT layers are prone to overfitting and underfitting when sampling methods are applied incorrectly. However, these architectures notably benefit from proportional or annealed sampling methods, as outlined in section 4.2. Our findings also suggest that using more sophisticated architectures (e.g., PALs) may hurt overall performance for multi-task learning when used alone, but may show more promising results as the number of tasks increases, particularly as the number of diverse tasks may incur more interference. We have also attempted additional pretraining and POS tagger and further pretraining to enhance the overall performance without much success at this time.

With the constraints of time and resources, I've outlined the extent of my contributions in this paper. However, I've observed significantly higher scores on the leaderboard achieved by other students, indicating the potential for more effective loss functions and other avenues for further enhancements, which I plan to explore in my own time. Lastly, I express my gratitude to everyone involved in facilitating this class. The invaluable lectures by our remarkable professors and guest speakers, along with the support from all teaching assistants, have provided an exceptional learning experience. I have gained invaluable insights into various challenges related to improving NLP systems, and I am fully committed to continuing this journey.

References

Cohn T. Bird S. Duong, L. and P Cook. 2015. Low resource dependency parsing: Cross-lingual parameter sharing in a neural network parser. In *Low resource dependency parsing: Cross-lingual parameter sharing in a neural network parser*.

Zhang X. Ren S. He, K. and J Sun. 2016. Identity mappings in deep residual networks. In *Identity mappings in deep residual networks*.

Bilen H. Rebuffi, S.-A. and A Vedaldi. 2018. Efficient parametrization of multi-domain deep neural networks. In *Efficient parametrization of multi-domain deep neural networks*.

Asa Cooper Stickland and Iain Murray. 2019. Bert and pals: Projected attention layers for efficient adaptation in multi-task learning. In *Bert and pals: Projected attention layers for efficient adaptation in multi-task learning*.

P. Swietojanski and S. Learning hidden unit contributions Renals. 2014. Learning hidden unit contributions for unsupervised speaker adaptation of neural network acoustic models. In *Learning hidden unit contributions for unsupervised speaker adaptation of neural network acoustic models*.

Y. Yang and T. M Hospedales. 2017. Trace norm regularised deep multi-task learning. In *Trace norm regularised deep multi-task learning*.