minBERT-based Multitask Model using PAL-LoRA

Stanford CS224N Default Project

Xian Wu
Department of Computer Science
Stanford University
lisaxwu@stanford.edu

Abstract

The motivation of this paper is to explore effective ways to fine-tune BERT-based (Devlin et al. (2019)) multitask model to handle various tasks simultaneously with optimal performance. Multitask is challenging because tasks interfere each other due to various reasons, either transferring knowledge or causing conflicts. The aim is to construct a multitask LM based on the pretrained minBERT model, that perform tasks of Sentiment Classification, Paraphrase Detection, and Semantic Textual Similarity(STS) each on their respective datasets. The contribution (beyond project basic requirements) and findings include: enhancing the model framework by incorporating task-specific Projected Attention layers (PALs) with LoRA (Asa Cooper Stickland (2019)) that is cost-efficient and produces appealing results nearly as training the full model; explored different output layers and loss function combinations for tasks, with small innovations for the Paraphrase task; tried further pretraining Sun et al. (2019) with Masked LM, though not finding this significantly improves the final accuracy, only helps faster convergence; investigated the impact of learning rates (lr), showing that large lr (1e-3) for the BERT-layer can degrade the performance by diminishing general language information from the pretrained version, and small lr(1e-5) for the output and the Projected Attention Layers, results in slow convergence; probed into the best LoRA rank size and PAL dropout rate, revealing that large rank size could cause severe overfitting and task trained with larger dataset (Para) performs better with lower dropout rate.

1 Key Information to include

- Mentor: Chaofei Fan
- External Collaborators (if you have any): No
- Sharing project: No

2 Introduction

BERT (Devlin et al. (2019)), as the state-of-the-art pretrained language model, has proven effectiveness in learning useful universal language representations, thus can be adapted to handle various NLP tasks. Multitask model Zhang and Yang (2017) is compelling for resources efficiency and leveraging knowledge across tasks, while challenging for below reasons: tasks may have conflicted objectives; tasks uses datasets whose distribution conflicts at the embedding layer, hard to weigh the importance (loss) Kendall et al. (2018) for different tasks, hard to optimize with gradient conflicts Yu et al. (2020).

This paper explores how to build and fine-tune multitasks model with pretrained minBERT that can perform tasks of Sentiment Classification, Paraphrase Detection, and Semantic Textual Similarity(STS) each on their respective datasets. The major approach to highlight is the task-specific **Projected Attention Layer with LoRA** (Asa Cooper Stickland (2019)) introduced across BERT layers, which allows divergence of attentions emphasis for each task to minimizes the conflicts, and turned out to be highly effective meanwhile, cheap and fast to train. For every single task, training only the PALs and output layer can fast converge to the same level of accuracy on dev dataset, compared to training the full model (BERT layer and output layer).

Besides, a few output layer and loss function options are experimented and compared, it turns out that **Cosine Similarity** (Reimers and Gurevych (2019)) is great factor to incorporate to benchmark tasks like STS and Paraphrase that compares a pair of sentences. Further more, since our task data has different distribution than the BERT pretrained base model, to give the target domain information more weight as Sun et al. (2019) suggested, I also performed **further pretraining with Mask LM** Devlin et al. (2019) approach. However this isn't much helpful for me for the final accuracy but only helps faster convergence at the few initial epochs of training, for which I had 2 hypothesis: PAL is already able to handle data distribution nuances for tasks across BERT layers, that further pretraining BERT itself is less significant; model is further pretrained with all data from 3 tasks, then data distribution conflicts actually diminish the benefit of further pretraining.

Last, thorough study on learning rate, dropout, LoRA size is presented, revealing that large lr (1e-3) for the BERT-layer can degrade the performance by dropping universal language information from the pretrained version, and small lr(1e-5) for the less sensitive layers, the output and the Projected Attention Layers, results in slow convergence.

3 Related Work

Pretrained language model has significantly transformed the NLP world as it provides universal language representations, which can be leveraged to tackle various NLP tasks effectively at lower cost. Beside BERT, other popular choices are: GPT Radford et al. (2018) which is a left-to-right autoregressive model; BART Lewis et al. (2019), a sequence-to-sequence autoencoder model that is a generalization of BERT (bidirectional encoder) and GPT (decoder).

Fine-tuning - To harness pretrained language model for NLP problems, we have the feature-based (such as ELMo Peters et al. (2018)) strategy and fine-tuning strategy. This paper explores the latter with BERT model, which produces the most state-of-the-art results in these days. The Sun et al. (2019) paper proved the effectiveness of further pretraining the BERT model with target domain data, and then fine-tuning on task specific layers, together with the learning-rate decaying technique introduced. Another compelling area is the parameter-efficient fine-tuning methods, such as Adapter-BERT Houlsby et al. (2019), and LoRA Hu et al. (2021). The reason is that fine-tuning the full large model can be inefficient in cost, and these methods by adding small number of parameters to the original model, can attain near state-of-the-art performance, while being cheap to train. In this paper, we also uses PAL+LoRA Asa Cooper Stickland (2019) method as our major approach.

Multitask learning Zhang and Yang (2017) has garnered significant attention in recent years, which is capable of leveraging BERT's language representations for multitasks simultaneously, thus making the model more general and less over-fitting to specific tasks. It also has the strength of being resources and data efficient, transferring domain knowledge cross tasks. But it also introduces great complexity. One problem is the negative transfer of representation among tasks, for which gradient normalization Chen et al. (2017) or gradient surgery Yu et al. (2020) techniques may help. Another problem is about dataset size imbalance issue (occurs for this paper too), and the different level of complexities of tasks, that we may need good task prioritization strategies.Guo et al. (2018) proposed the solution as dynamically prioritizing the difficult tasks by adjusting loss function weights. Kendall et al. (2018) weighs loss functions by homoscedastic uncertainty of tasks.

4 Approach

Figure 1 below shows the model architecture: BERT combined with task-specific PALs and output layers. Tasks have their own PALs pictured with the same color.

Baseline - The minBERT skeleton (git) is provided by CS224N class (handout), with Self-Attention layers and Adam Optimizer implemented by myself. The PAL idea is originated from Asa Cooper Stickland (2019) and implemented by myself. All the task-specific layers are design and implemented by myself as well. Training is on top of the given pretrained "bert-base-uncased" model.

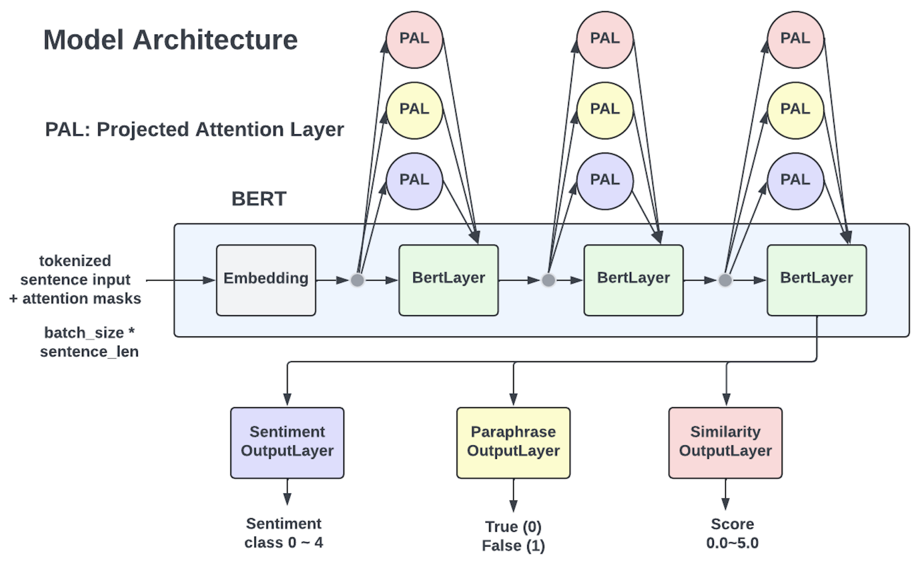


Figure 1: Model Architecture

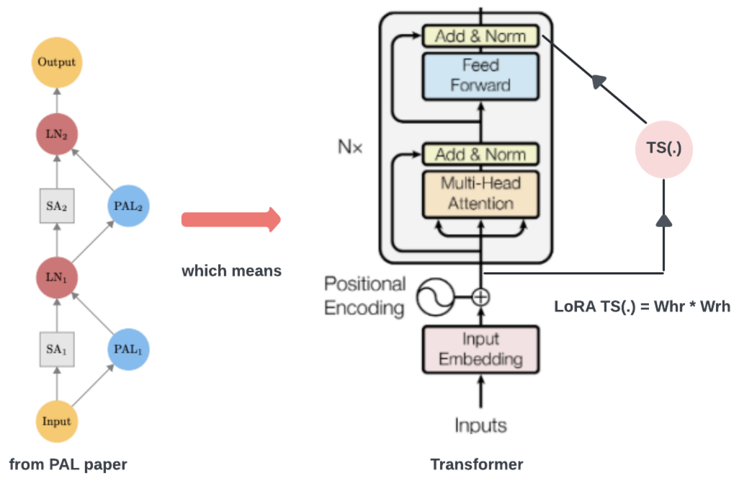


Figure 2: Projected Attention Layer

4.1 Projected Attention Layer + LoRA

PAL means adding the task-specific layer $TS(.)$ across bert layers, the input is the hidden states output from previous BERT layer, and the output will be applied to "Add & Norm" with $FFN(.)$, same as $SA(.)$, result:

$$BL(h) = LN(h' + SA(h) + TS(h))$$

There are many choices of $TS(.)$ while I chose LoRA:

$$TS(h) = (h \cdot W_{hr} + b1) \cdot W_{rh} + b2$$

The r in W_{hr} means size of rank, $r << hidden\ state\ size$, so that we can have small number of task specific parameters to train.

4.2 Output Layer and Loss function

4.2.1 Sentiment Classification

Let's use h_{CLS} (size 768) to represent the "pooler output" of BERT, as it is in fact the corresponding hidden state for [CLS] token, and contains sentence-level information. The sentiment classification output layer is simply designed as:

$$output = Linear(h_{CLS})$$

which means un-sigmoid probabilities for classes. CrossEntropyLoss is chosen.

4.2.2 Semantics Textual Similarity

The output layer is simply computing the Cosine Similarity of the 2 sentences:

$$output = ReLU(CosineSimilarity(h_{CLS1}, h_{CLS2})) * 5.0$$

which means the "angle" between the 2 sentences, note that sentences with negative correlation should be scored as 0, thus ReLU is applied, and then I scale it up to 5.0. The loss function used is simply MSE. Note that there is no "last layer" parameters to train, but in my practice it simply trains the PAL parameters if the BERT layer is frozen. The inferior option discarded is $output = Linear([h_{CLS1}, h_{CLS1}])$, for which we show results in experiment section too.

4.2.3 Paraphrase Detection

The output layer is defined as:

$$output = Linear([h_{CLS1}, h_{CLS2}, h_{CLS1} \cdot h_{CLS2}]) + CosineSimilarity(h_{CLS1}, h_{CLS2})$$

which takes into account the magnitude of both h_{CLS1} and h_{CLS2} and the angles between them with $h_{CLS1} \cdot h_{CLS2}$ item in the Linear layer. The **small innovation** here is that since we eventually feed this output into sigmoid function, I added the Cosine-Similarity param again, to penalty more on direction differences when the score is near 0.5 (round to 0 or 1), which gets slightly better result.

4.3 Further Pretraining with Masked LM

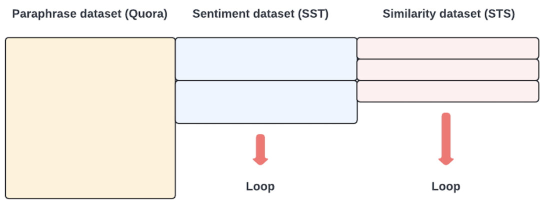
Sun et al. (2019) suggested a general method of multitask training, by first further pretrain the BERT layer with target domain data, and then freeze the BERT layer to optimize on task specific parameters. I tried the Masked LM pretraining, but not seeing significant improvements with PAL + LoRA.

4.4 Multitask Training

The strategy is to apply both training methods below, and iterating until we get a good result:

- **Task-specific layers** - Train only task-specific layers (including the PAL-LoRA layer for my case) sequentially with BERT frozen.
- **Full-model** - Train full model for all tasks in parallel.

Joining Datasets - In the full model training, dataset for Paraphrase task is way larger than the others, if training each task sequentially every epoch, intuitively Paraphrase will take a lot more precedence over the other 2 tasks with more steps optimized. Thus I jointed 3 datasets as below graph shows. This joint dataset is also used in the Masked LM pretraining mentioned above.



Joint Loss - In the full model training option, I simply summed up 3 losses each with weight 1.0, which means their gradients are weighed equally. Here are ways of determining loss weight Kendall et al. (2018) or doing gradient surgery Yu et al. (2020), but I didn't further explore these as I found the PAL-LoRA layer already helps reaching the same level of performance as the full model training, thus eventually the sequential way with BERT frozen is chosen.

5 Experiments

5.1 Data

Tasks	Dataset	Train	Dev	Test
Sentiment Analysis	Stanford Sentiment Treebank	8,544	1,101	2,210
Sentiment Analysis	CFIMDB	1,701	245	488
Paraphrase Detection	Quora (Para)	283,010	40,429	80,859
Semantic Textual Similarity	SemEval (STS)	6,040	863	1,725

5.2 Evaluation method

For Sentiment Analysis and Paraphrase Detection, the accuracy on Dev dataset is taken as the major benchmark. Semantic Textual Similarity uses Pearson correlation coefficient: $\rho_{xy} = \frac{cov(X,Y)}{\sigma_x \sigma_y}$. Besides, a few secondary metrics are also taken into account, such as how fast to train, and how train loss change with dev accuracy changes, which may indicating over-fitting problems.

5.3 Experimental details

This paper presents the a few experiments, indicating the efficiency of the PAL-LoRA approach.

5.3.1 Sentiment Analysis (SST) single task training

This experiment compares training of only the output layer vs. PAL-LoRA vs. full model, with both SST and CFIMDB datasets. CFIMDB full-model uses 32 batch otherwise it crashes for GPU limit

Dataset	Mode	Batch	Epoch	lr	Batch / Epoch	GPU	Time to train
SST	full model	64	10	1e-5	134	A100	4.2 min
SST	PAL-LoRA	64	10	1e-3	134	A100	2.5 min
CFIMDB	full model	32	10	1e-5	54	A100	6.3 min
CFIMDB	PAL-LoRA	64	10	1e-3	27	A100	4.8 min

5.3.2 STS single task training

This experiment trains Semantics Textual Similarity on STS dataset, to comparing these techniques: Cosine Similarity vs. Linear output layer; "bert-base-uncased" vs. "MLM further pretrained" version; Full model vs. PAL-LoRA training (rank size 2).

Mode	Batch	Epoch	lr	Batch / Epoch	GPU	Time to train
Full model	64	20	1e-5	95	A100	8.5 min (22 sec/epoch)
PAL-LoRA	64	20	1e-3	95	A100	5.2 min (15 sec/epoch)

5.3.3 Paraphrase Detection single task training

This experiment again compares PAL-LoRA training (rank size 2) vs. full model for the Paraphrase Detection task. A few output layer options are also examined.

Mode	Batch	Epoch	lr	Batch / Epoch	GPU	Time to train
full model	64	1	1e-5	4422	A100	42 min
PAL-LoRA	64	1	1e-3	4422	A100	36 min

5.3.4 Training Parameters Study

LoRA Rank Size - I applied 2, 10, 100 rank sizes to all three tasks. Other configs stay the same.

Learning Rate - I chose the STS task to study the impact of learning rate for simplicity, applied 1e-3, 1e-5 to PAL-LoRA training, and its full model training.

PAL Dropout - Applied 0.1 0.2, 0.3 dropout to all tasks. The dropout is for PAL $Dropout(TS(.))$, not BERT(0.1) nor the output layer(0.3).

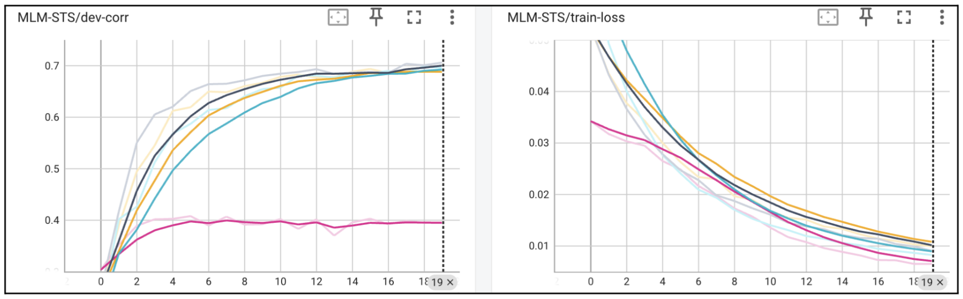


Figure 3: training full model, with BERT layer updates

5.4 Results

5.4.1 Sentiment Analysis (SST) single task training

Mode	lr	SST Dev Acc	CFIMDB Dev Acc
Full model	1e-5	0.521	0.96
Output Only	1e-3	0.391	0.767
PAL-LoRA	1e-3	0.516 (ranksize 1)	0.959 (ranksize 1)

which shows that PAL-LoRA $\approx$ full model training $\gg$ output layer only.

5.4.2 STS single task training

Full Model Mode	Base Model Version	Output Layer	PAL trained	Dev Corr
blue line	bert-base-uncased	Cosine Similarity	False	0.695
black line	MLM further-pretrained	Cosine Similarity	False	0.706
pink line	MLM further-pretrained	Linear	False	0.395
yellow line	MLM further-pretrained	Cosine Similarity	True	0.688

Mode	Base Model Version	lr	Dev Corr
PAL-LoRA	bert-base-uncased	1e-3	0.73
	MLM further-pretrained	1e-3	0.70

Figure 3 plots the accuracy on Dev every epoch, and the train loss change. The result shows that Cosine Similarity works way better than Linear output layer; MLM further-pretraining only helps faster convergence but not significantly improves the upper limit. Results of only training the PAL-LoRA proves that, within 20 poch, it is highly equivalent to training the full-model (both reaching 0.7 accuracy)! Also Masked LM further-pretraining doesn't show extra benefits with PAL.

5.4.3 Paraphrase Detection single task training

Base Model Version	Mode	Output Layer	Dev Acc
bert-base-uncased	Full model	With Cosine Penalty	0.78
	PAL-LoRA	With Cosine Penalty	0.77
	PAL-LoRA	Without Cosine Penalty	0.70

which proves PAL-LoRA is highly effective, only similarly as full-model training (-0.01 accuracy), and also the Cosine penalty item is slightly helpful (with 0.07 more accuracy).

5.4.4 Training Parameters Study

Rank size - Figure 4 shows that rank size 2 works the best for all tasks. It turns out selecting good rank size is tricky and data-dependent: Larger rank size helps faster fitting at early training, but more overfitting in the end; Larger dataset (Para) may work well with small rank size, as it may not have as much general textual representation to add to the system.

Learning rate - Figure 5.4.4 shows full-model cannot use learning rate > 1e-3, gradient could be NaN, as BERT drops general text representation and overfit. For task-specific layer, 1e-3 well compromises fast convergence and precision for steps while 1e-5 is too slow.

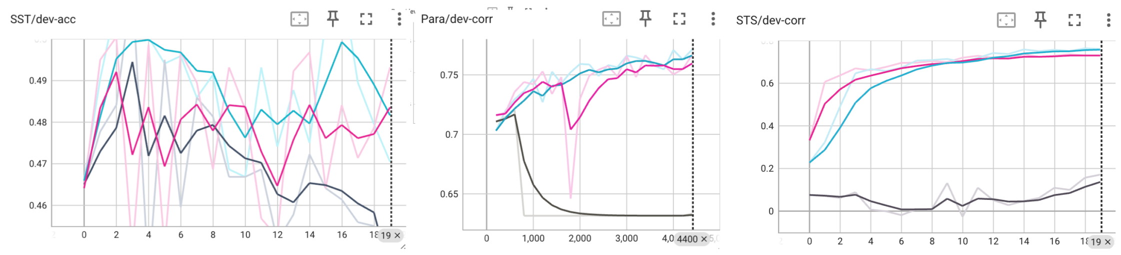
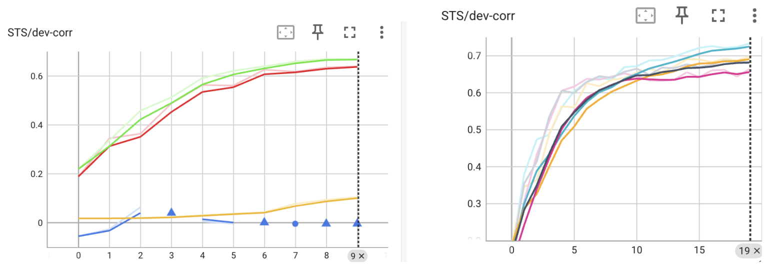


Figure 4: Rank Size: blue 2, pink 10, black 100; tasks: SST, Para, STS

Dropout - Figure 5.4.4 shows that PAL dropout of 0.2 works best for STS. Without figure here, but 0.1 works the best for SST, 0.05 works the best for Paraphrase task, which makes sense as the larger the dataset, the less likely to overfit thus smaller dropout needed.



PAL: 1e-3(red), 1e-5(orange)
Full: 1e-3(blue), 1e-5(green)

Dropout 0.0(orange) 0.1(pink)
0.2(blue) 0.3(black)

Figure 5: STS learning rate (left) and PAL dropout (right)

Below is the **Leaderboard Result** by training PAL-LoRA only for all tasks sequentially, using 1e-3 learning rate, PAL rank size 2, and PAL dropout 0.1 for SST, 0.2 for STS and 0.05 for Paraphrase.

Leaderboard	Dev	Test
Sentiment (SST)	0.519	0.527
Paraphrase (Para)	0.793	0.795
Textual Similarity (STS)	0.750	0.738

6 Analysis

Overall the model doesn't produce absurd errors, some of the error it made, I could also make as a human. For **Sentiment** task, the model prediction, when being different with truth, is mostly just ±1 difference. Also the model got wrong usually by rating it lower. Here is an example that the truth is 3 (neutral), but the model says 1 (negative): "This surreal Gilliam-esque film is also a troubling interpretation of Ecclesiastes .", which is likely due to the keyword "troubling". But I as a human may also rate this as negative, **such error is due to lack of context** in which the statement is made. In movie reviews, people use negative words just as rhetoric, "It haunts , horrifies , startles and fascinates ; it is impossible to look away ." this is a 4, the model gave it 3 as it probably balanced the negative words with good ones. Thus the model **fails on rhetoric and creative expressions**. And it works perfectly when **sentiment of words aligns with the sentences**.

For **Paragraph** task, the model makes **more mistakes on false positive, than false negative**. For example, "How do I upload photos to Quora with a pc? How do you include a photo with your post on Quora?", the key is about "a pc", not about uploading photos. But this error is tricky as we also have **dropout**, thus such important details can be lost. Also I saw cases like "What are good ways to prepare for gate exam in 3 months so that I will secure good rank? How should GATE be cracked within 3 months (ECE)?", this isn't a paraphrase to me as well, because ECE is a **missing context** for both me and the model. Thus I think what the model says is fair.

For **Semantics Textual Similarity** task, it tends to score similarity higher than the ground truth, for which I think the "CosineSimilarity * 5.0" output choice may be affecting this, as it tends to score high-score pairs (in small angle) higher. Also this measurement doesn't take into account magnitude, thus might be over-positive, thus in this case, "A girl is eating a cupcake. A woman is eating a cupcake.", the similarity ground truth is 2.6, but our score is 4.9! Girl and woman are similarity but likely with different magnitude in embeddings.

7 Conclusion

PAL-LoRA is an effective way to fine-tuning a multitask model. Even without updating the BERT-layer, it still gives the nice performance near the full-model tuning for each task, with following benefits: **Cost-efficient –** Tuning PAL-LoRA with the same number of epochs only takes 60% of the training time; **Faster convergence –** It requires less training epochs than full-model to get to the best level of performance; **Avoid conflicts –** Tasks may inference each other, lead to worse performance than single task training; **Scalable and secure for large system –** If to build a general multitask model which keeps supporting new tasks and allows for diverged child versions, optimizing full BERT could mean unscalable complexity. It also introduces dependencies between tasks that are insecure; **Flexibility –** LoRA is one way to implement the PAL, tasks can use different implementations that best fits the use cases.

The limitation of only training PAL is also obvious: **Without learning transfer –** Tasks can bring in textual representation that benefits others; **Overfitting –** PAL is task-specific, which means it can also be training-dataset specific. It converges really fast and then started to perform unstably for Dev dataset, which exactly indicates the great overfit problem. **PAL design complexity –** What PAL implementation to use requires further design and experimentations task-by-task. In LoRA case, a good rank size is task and data dependent.

The most important lesson learned - A good design of model is the first priority, training tricks are secondary. If the model doesn't fit the goal, the more training attempted, the more time wasted. I at first didn't use CosineSimilarity output, and went for Masked LM pretraining, it still doesn't help my case, until the right model was selected.

Future work - In the full-model training practice, I simply summed up 3 losses with weight 1.0, weighing all gradients equally. This doesn't show improvements than PAL-only training. I saw there are ways of determining loss weight Kendall et al. (2018) or doing gradient surgery Yu et al. (2020), I want to experiment on these, to see if "Full model + PAL + Gradient and Loss Strategies" could further lift the performance.

8 Ethics Statement

Bias Amplification – When the training dataset contains biased information about genden, race, ethnicity etc, the PAL-LoRA layer amplifies such bias as it is 100% tuned with the training data. One way is to conduct bias correction training, replace certain keywords from the input, compare the outputs and penalty on the misalignment. E.g. if "man" did something that is considered sentiment positive, then "women" did the same thing are likely to be positive too.

Wrong Judgement on Harmful Content – Dataset like movie reviews tell positive sentiment to the movie but may contain violent information about murders or drugs at the same time, this can mislead the PAL layer to wrongly weighing these concepts. One straightforward solution is to introduce in an unhealthy content filter, mask out the harmful words then apply to training.

References

Iain Murray Asa Cooper Stickland. 2019. Bert and pals: Projected attention layers for efficient adaptation in multi-task learning. In *International Conference on Machine Learning*, page 5986–5995. PMLR.

Zhao Chen, Vijay Badrinarayanan, Chen-Yu Lee, and Andrew Rabinovich. 2017. Gradnorm: Gradient normalization for adaptive loss balancing in deep multitask networks. *CoRR*, abs/1711.02257.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *arXiv preprint arXiv:1810.04805*.

git. https://github.com/amahankali10/cs224n-spring2024-dfp-student-handout.

Michelle Guo, Albert Haque, De-An Huang, Serena Yeung, and Li Fei-Fei. 2018. Dynamic task prioritization for multitask learning. In *European Conference on Computer Vision*.

handout. https://web.stanford.edu/class/cs224n/project/default-final-project-handout-minbert-spr2024-updated.pdf.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for nlp. In *International conference on machine learning*, pages 2790–2799. PMLR.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *CoRR*, abs/2106.09685.

Alex Kendall, Yarin Gal, and Roberto Cipolla. 2018. Multi-task learning using uncertainty to weigh losses for scene geometry and semantics. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2019. BART: denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *CoRR*, abs/1910.13461.

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. *CoRR*, abs/1802.05365.

pretrained. https://huggingface.co/google-bert/bert-base-uncased.

Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. 2018. Improving language understanding by generative pre-training.

Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. *CoRR*, abs/1908.10084.

Chi Sun, Xipeng Qiu, Yige Xu, and Xuanjing Huang. 2019. How to fine-tune BERT for text classification? *CoRR*, abs/1905.05583.

Tianhe Yu, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. 2020. Gradient surgery for multi-task learning. In *Advances in Neural Information Processing Systems*, volume 33, pages 5824–5836. Curran Associates, Inc.

Yu Zhang and Qiang Yang. 2017. An overview of multi-task learning. *National Science Review*, 5(1):30–43.