minBERT and Downstream Tasks

Stanford CS224N Default Project

Jason Alexander Chan
Department of Computer Science
Stanford University
jchan7@stanford.edu

External collaborators: NA. External Mentor: Shai Limonchik. Sharing project: NA

Abstract

This project implements minBERT, a minimalist implementation of BERT. Original code was written for the multi-headed self-attention and transformer layer along with the Adam optimizer algorithm. These are combined with the provided starter code to form the minBERT implementation. minBERT is initially trained and fine tuned to solve a sentiment analysis task. Extensions were then made to train a multitask minBERT model to perform three downstream Natural Language Processing (NLP) tasks: sentiment classification, semantic textual similarity comparison and paraphrase detection. Across these three tasks, the multitask model achieved an average prediction accuracy of 0.583 on the dev set and 0.582 on the test set. This is better than the random chance baseline of 0.300. The multitask model was trained on a round robin training curriculum with the cosine similarity extension. Other extensions included additional pretraining, SMART regularization, gradient surgery, hyperparameter tuning and layer changes. However, none of these extensions individually improved the multitask model when it was combined with the cosine similarity extension.

1 Introduction

In 2018, Devlin et al. demonstrated that state-of-the-art models for NLP tasks could be created by fine tuning their pretrained BERT model [1]. The authors advanced the state of the art at that time for eleven NLP tasks. Fine tuning is a breakthrough because it tremendously lowers the barriers of entry for engineers to create near state of the art performing models. It lowers the time, computational resources, and the volume of training data required. Since then practitioners have been conducting what amounts to *"many random trials... and picking the best model based on validation performance"* [2]. A large portion of the current NLP academic literature focuses on discovering techniques to exploit the full potential of fine tuning pretrained models as well as checking the efficacy of the heuristics employed by NLP practitioners.

2 Related Work

The NLP multitask model utilized in this project incorporates several extensions that are inspired by previous research on fine-tuning multitask models. Phang et al. (2018) investigated fine tuning on small datasets (<10,000 samples). They found that training on a larger intermediate task before fine tuning improved model performance [3]. Zhang et al. (2021) declared that the *"three-epoch practice for BERT fine-tuning is sub-optimal"*. Regarding regularization they claimed that *"pre-trained weight decays works better than conventional weight decay"*[2]. Fine tuning suffers from over fitting when trained for many epochs to which Lee et al. (2020) proposed the *"mixout" regularization technique, which resets a target parameter back to the original pre-trained value given some probability* [4].

They found that the *"average accuracy greatly increase"* when mixout is used to regularize BERT fine-tuning.

One motivation for utilizing NLP multitask learning is to overcome a limitation of Devlin et al.'s (2018) fine-tuning approach, which involved instantiating a new model for each downstream NLP task. For the purposes of this project, multitask learning is defined as creating one model to perform multiple NLP tasks. Worsham et al. (2020) state that multitask learning *"predates"* the deep learning era but has *"seen a resurgence"* in NLP [5]. Three conclusions were drawn from that paper: firstly, related tasks often have underlying complex and competing dynamics. Secondly, round robin is the standard training curriculum for multitask learning. Finally, *"beginning with more difficult tasks and slowly introducing additional tasks performs the best"*.

A dilemma in multitask learning is that it seeks to maximise knowledge transfer across tasks but minimises catastrophic forgetting *"to achieve both objectives is highly challenging"*[6]. Ke et al. (2021) observed that current research papers don't simultaneously evaluate catastrophic forgetting and knowledge transfer. Yu et al. (2020) conducted early work into this, conjecturing that some tasks may have *"conflicting gradients"* [7]. They proposed a form of *"gradient surgery"* that involved projecting a tasks's gradient (PCGrad) *"onto the normal plane of the gradient of any other task that has a conflicting gradient"*.

In summary, fine tuning and multitask learning in NLP is a highly active and productive area of research. The related works summarised in this section aided the approach of this project.

3 Approach

There are three objectives in this project. This report primarily focuses on objective three but also provides a high level overview of objectives one and two.

3.1 Objective 1: Implement minBERT

The first objective implemented minBERT, which has the key features of the original BERT model that includes the multi-head self-attention and Transformer layer. Please refer to [8] for information on the implementation.

3.2 Objective 2: Analyse Sentiment with minBERT

The second objective evaluated minBERT on two downstream NLP tasks: sentiment analysis on the Stanford Sentiment Treebank (SST) and the CFIMDB movie reviews database. See section 4.1 for information on the datasets. This objective implemented the classification layer for both tasks and the Adam Optimizer algorithm. minBERT achieved a prediction accuracy on the dev set of 0.399 and 0.776 for pretrained SST and pretrained CFIMDB respectively, and 0.522 and 0.971 when fine tuned.

3.3 Objective 3: Extend and Improve minBERT for Additional Downstream Tasks

The third objective extends the minBERT model to perform three tasks: sentiment analysis on the SST dataset (SST), paraphrase detection on the Quora dataset (Para) and Semantic Textual Similarity on the SemEval dataset (STS). See section 4.1 for more information on the dataset. Herein, the acronymns SST, STS and Para are used to mean both the dataset and their respective tasks. Each task is a different prediction problem: SST is a multi-class classification problem, STS is a regression problem and Para is a binary classification problem.

The seven extensions described in subsections 3.3.1 to 3.3.7 were motivated by the problems discovered during initial prototyping of the minBERT multitask model. These problems prompted a review of NLP literature and resulted in the summary of the related works described in section 2.

3.3.1 Extension 1: Round Robin Multitasking Curriculum

The core approach to multitask training is around round robin. Adapted code was written to train each task in a round robin curriculum, which according to Worsham (2020) is the standard practice for multitask learning [5][9]. To address the problem of unbalanced data samples as described in 4.2, the round robin multitask training down sampled the number of samples to match the task with the fewest samples, which is the STS dataset. The loss function for each epoch is the average of the three tasks' losses to reflect the scoring criteria for the project on the student leaderboard, which is also a simple average of the three tasks.

$$L_{multitask} = (L_{SST} + L_{STS} + L_{Para})/(3 \times NumBatches_{RR}) \tag{1}$$

3.3.2 Extension 2: Cosine-Similarity Fine-Tuning for Semantic Similarity

During prototyping, STS had poor performance. Its Pearson correlation approximated zero. Reimers et al. (2019) demonstrated that using BERT CLS output token, which was done during prototyping, caused poor performance for semantic similarity tasks. Thus, original code was written to implement Reimers et al. (2019) work on the task-specific neural network for STS[10].

3.3.3 Extension 3: Additional Pretraining

The round robin downsampling approach discards a significant portion of the Para training dataset. This is a gross waste of training data that could potentially uplift performance. Thus, additional pretraining was hypothesised as a way to use all of the Para training dataset before starting the round robin training. Original code was written to test Phang et al.'s (2018) conclusion showed that pretraining on a larger intermediate task before finetuning on tasks with limited training data (<10,000 samples) improved model performance [3].

3.3.4 Extension 4: SMART Regularization for Finetuning

During prototyping, overfitting was consistently observed. Dev accuracies would plateau around epoch 6 while training accuracies continued to increase. It was hypothesised that regularization could potentially uplift dev accuracies. The regularization technique for fine tuning by Jiang et al. (2019) [11] is a method for multitask models. A tuneable parameter λ adds a loss penalty called *"the smoothness-inducing adversarial regularizer"*. A PyTorch library based on Jiang et al.'s *"SMART Regularization"* technique was implemented [12].

$$L_{multitask} = \sum_{task} (L_{task} + \lambda_s R_{s_{task}}(\theta)) \tag{2}$$

The library implements the *"symmetrized"* KL-divergence by default for $R_s(\theta)$, which is applicable for classification tasks. The extension also experimented with squared loss and MSE loss for the STS, which was formulated as a regression problem.

3.3.5 Extension 5: Gradient Surgery

The student leaderboard indicated that there is more performance to be gained from STS and Para. This extension hypothesised that the three downstream tasks may be conflicting. The PCGrad technique by Yu et al. (2020) was implemented via a PyTorch library by Wei Cheng Tseng [13].

3.3.6 Extension 6: Hyperparameter Optimisation

Hyper parameter optimisation are easy to implement extensions, which could potentially improve performance. The learning rate and the dropout probability were changed. During prototyping and throughout the experimentation phase, the prediction results plateaued around epoch 6. Thus epoch experiments were ruled out.

3.3.7 Extension 7: Fine Tuning Layers

This extension was motivated by the lack of success with the previous extensions. One hypothesis is that the forward prediction layers were subpar. Experiments were conducted on a shallower Para network and a deeper STS network as described in Appendix A.7.

3.4 Forward Prediction Layers

Original code was written for forward predictions of the SST multi-classification task (predict_sentiment), STS regression task (predict_similarity) and Para binary classification task (predict_paraphrase) as shown in Figure 1. SST implemented a simple linear classification as suggested by Sun et al. (2019) [14]. STS implemented the Cosine Similarity with one linear and one activation layer [10]. Para, on the other hand, experimented with concatenating the two inputs and passing that through a two linear and one activation layer. These networks were held fixed during the experiments, except for during SMART regularization due to API constraints.

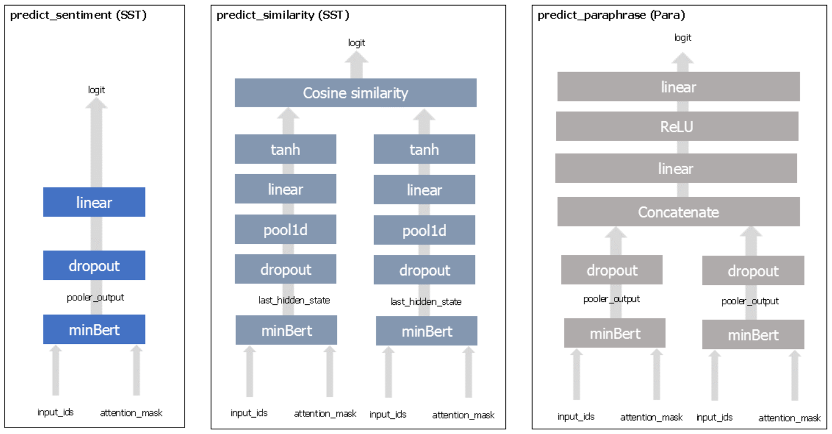


Figure 1: Task Specific Neural Network Models

3.5 Tools and Probing Methods

Git was used for configuration control. Data loading scripts are provided by the teaching staff and were adapted to load multitask batches [8]. The provided start code was adapted to create the multitask training curriculum. Original code was written to collect telemetry, which includes training and dev accuracy vs. epoch, along with hyperparameters and user arguments. Finally, original code was written to generate plots after each training run and watermarked with the git hash.

3.6 Baseline

There are no benchmarks for minBERT on the multitask performance across the three downstream tasks. Thus, a proposed baseline is random chance. SST has five labels, STS could be considered to have six labels because it has labels from 0-5 inclusive (but is treated as a regression problem in the implementation), while Para is binary. See Section 4.1 for more information on the dataset. Thus the average of the random chance results across the three datasets is approximated as 0.3.

4 Experiments

4.1 Data

There are four data sources employed in this project. The NLP task on the SST data set is sentiment classification [15]. The SST data set contains 11,885 single sentences from movie reviews. There are 215,154 unique phrases annotated by 3 human judges with the labels: negative, somewhat negative, neutral, somewhat positive and positive. The NLP task on the CFIMDB data set is also sentiment classification [8]. The CFIMDB movie reviews database has 2,434 movie reviews whose labels are positive or negative. There are reviews that are longer than one sentence. This is used for objective only.

The NLP task on the Quora data set is paraphrase detection [16]. The Quora data set has 400,000 question answer pairs with labels informing whether such instances are paraphrases of each other. Finally, the NLP task on the SemEval data set is Semantic Textual Similarity [17]. The SemEval data set has 8628 different sentence pairs of varying similarity on a scale from 0 (unrelated) to 5 (equivalent in meaning).

4.2 Unbalanced Data Samples

The training data samples are unbalanced between each task.The STS training set has the fewest samples with 6,040, while SST has 8,544 and Para has 141,498. Para has over 23x more training data samples than STS and 16x more than SST.

4.3 Evaluation method

The evaluation metric for SST and Para are their prediction accuracies. The evaluation metric for STS is Pearson correlation, which ranges from -1 to +1. The Pearson correlation is used because the STS task predicts the degree of similarity between pieces of texts. The overall performance of the multitask model is evaluated as the average of those predictions. Each extension is evaluated on the dev set and the best final dev set result is evaluated on the test set.

4.4 Experiment Details

The results of the extensions are summarised in Table 1. Model changes were made incrementally and re-trained. The batch size set to 16 to utilise the available GPU capacity to reduce training time. Note that the definition of *Baseline* has been changed herein to refer to the pretrained multitask Round Robin (RR) minBERT model upon which the extension experiments were conducted.

Table 1: Experiment Setup

ID	Model	Hash	Experiment	Mode	Epochs	Batch size	Drop out Prob.	Learning Rate
R1	Multitask RR	d09a075	Baseline	Pretrained	10	16	0.3	1E-03
R2	Multitask RR	d09a075	Baseline	Finetuned	10	16	0.3	1E-05
R3	Multitask RR	ad910f4	Baseline & Cosine	Pretrained	10	16	0.3	1E-03
R4	Multitask RR	ad910f4	Baseline & Cosine	Finetuned	10	16	0.3	1E-05
R5	Multitask RR	6d096da	Baseline & Cosine & Add. Pretraining	Finetuned	10	16	0.3	1E-05
R6	Multitask RR	b01b079	Baseline & Cosine & SMART Reg. λ =5	Finetuned	10	16	0.3	1E-05
R7	Multitask RR	b01b079	Baseline & Cosine & SMART Reg. λ = 0	Finetuned	10	16	0.3	1E-05
R8	Multitask RR	c0a3e17	Baseline & Cosine & PCGrad	Finetuned	10	16	0.3	1E-05
R9	Multitask RR	b4c3cef	Baseline & Cosine & HyperParams	Finetuned	6	16	0.3, 0.4, 0.5	1E-04, 1E-06, 1E-07
R10	Multitask RR	Various	Baseline & Cosine & Layer Changes	Finetuned	10	16	0.3	1E-05
R11	Multitask RR	826541f	Baseline & Cosine & BERT LayerNorm fix	Finetuned	10	16	0.3	1E-05

4.5 Results

The results of the experiments are summarised[1] in Table 2. Plots are included for the overall multitask accuracy and loss. Supplementary plots can be found in Appendix A. The best result is experiment R11 which implemented cosine similarity to achieve an overall performance of 0.583. Across all experiments, Para had the best average performance and lowest standard deviation. SST and STS had similar average performance and standard deviations. The other extensions subsequent to experiment R4 did not improve performance. The reasons for this are presented in Section 5, Analysis.

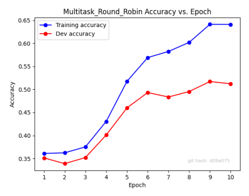
Table 2: Experiment Results

ID	Model	Hash	Experiment	Mode	Overall Dev	SST Dev	STS Dev	Para Dev
R1	Multitask RR	d09a075	Baseline	Pretrained	0.282	0.216	0.004	0.625
R2	Multitask RR	d09a075	Baseline	Finetuned	0.512	0.471	0.417	0.649
R3	Multitask RR	ad910f4	Baseline & Cosine	Pretrained	0.279	0.260	- 0.048	0.625
R4	Multitask RR	ad910f4	Baseline & Cosine	Finetuned	0.563	0.489	0.486	0.714
R5	Multitask RR	6d096da	Baseline & Cosine & Pretraining	Finetuned	0.548	0.455	0.424	0.765
R6	Multitask RR	b01b079	Baseline & Cosine & SMART Reg λ=5	Finetuned	0.436	0.253	0.351	0.703
R7	Multitask RR	b01b079	Baseline & Cosine & SMART Reg λ=0	Finetuned	0.470	0.363	0.338	0.709
R8	Multitask RR	c0a3e17	Baseline & Cosine & PCGrad	Finetuned	0.554	0.489	0.485	0.690
R9	Multitask RR	b4c3cef	Baseline & Cosine & HyperParams	Finetuned	0.563	0.490	0.497	0.703
R10	Multitask RR	Various	Baseline & Cosine & Layer Changes	Finetuned	0.562	0.488	0.490	0.708
R11	**Multitask RR**	**826541f**	**Baseline & Cosine & BERT LayerNorm fix**	**Finetuned**	**0.583**	**0.510**	**0.500**	**0.740**
				Average	0.483	0.394	0.399	0.693
				Std Dev.	0.102	0.111	0.137	0.04

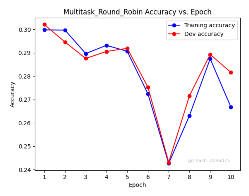
Figure 2 illustrates that fine tuned results (0.512) are better than pretrained results (0.282) in R2 vs. R1. Finetuning lifted the performance of all three tasks: STS improved from 0.004 to 0.417, SST improved from 0.216 to 0.471. Pretrained Para remained somewhat consistent at 0.625 vs.

[1]Only the best results from the hyperparameter tuning and layer changes are included in Table 2. See Appendix A for the lower performing hyperparameter and layer experiments.

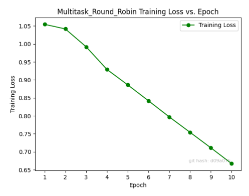
0.649 finetuned Para. The pretrained loss curve increases over the number of epochs, consequently pretrained accuracy doesn't improve. The dev accuracy during finetuning, on the otherhand, improves until plateauing from epoch 6 onwards and overfitting.



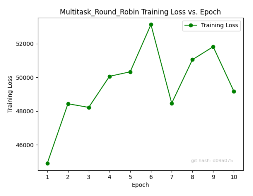
(a) Baseline Finetune Acc.



(b) Baseline Pretrain Acc.



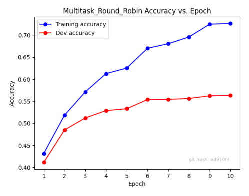
(c) Baseline Finetune Loss



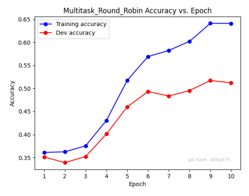
(d) Baseline Pretrain Loss

Figure 2: Baseline Finetune (R2) vs. Baseline Pretrain (R1)

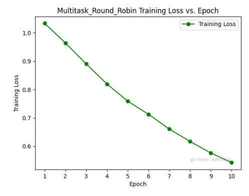
Figure 3 shows that the cosine similarity extension improved the overall multitask performance and the performance of all three tasks. The overall fine tuned result improved from 0.512 (R2) to 0.563 (R4). SST improved from 0.471 to 0.489, Para improved from 0.649 to 0.714 and SST improved from 0.417 to 0.486. The dev predictions converged faster with cosine similarity than without, occurring between epochs 3-5 rather than 5-6.



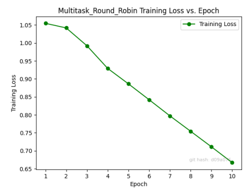
(a) Baseline & Cosine Similarity Acc.



(b) Baseline Finetune Acc.



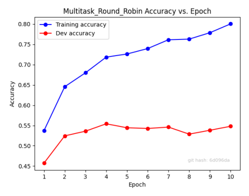
(c) Baseline & Cosine Similarity Loss



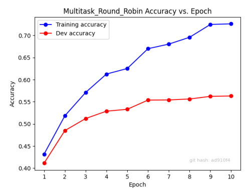
(d) Baseline Finetune Loss

Figure 3: Baseline & Cosine Similarity (R4) vs. Baseline Finetune (R2)

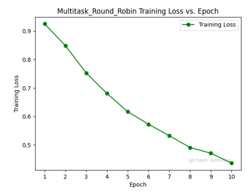
Figure 4 shows that additional pretraining failed to improve performance. It reduced performance from 0.563 (R4) to 0.548 (R5). Overfitting increased since training accuracy increased from 0.7 to 0.8 without uplifting dev accuracy. Unsurprisingly, the additional pretraining uplifted the task specific performance of the Para dataset from 0.714 to 0.765. STS and SST did not benefit from more pretraining.



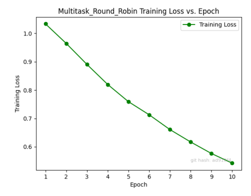
(a) Baseline & Cosine Sim. & More Pretraining Acc.



(b) Baseline & Cosine Sim. Acc.



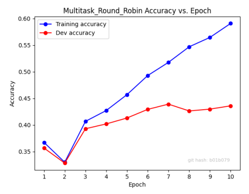
(c) Baseline & Cosine Sim. & More Pretraining Loss

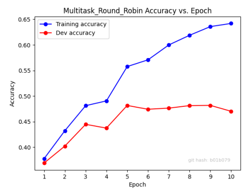


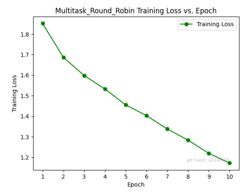
(d) Baseline & Cosine Sim. Loss

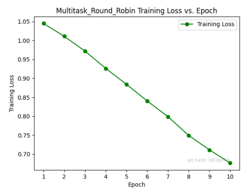
Figure 4: Baseline & Cosine Sim. & Additional Pretraining (R5) vs. Baseline & Cosine Sim. (R4)

Figure 5 shows that regularization ($\lambda = 5$) failed to improve performance. It unexpectedly reduced performance by 0.03 (R6) vs. the model with $\lambda = 0$ (R7). SST dropped by 0.110, Para remained the same, while SST increased 0.01. Overfitting reduced during epochs 1-5 but SMART regularization didn't improve the overfitting after epoch 6 as expected. The final loss values with SMART Regularization were higher than without regularization as expected. The performance results in experiment R8 ($\lambda = 0$) is lower than experiment R5, which also didn't have SMART Regularization. This is an implementation problem because they should be the same, and explained by the fact that the task specific network architectures were modified in experiment R8 to fit the SMART Regularization API [12].





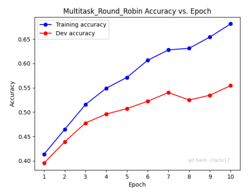


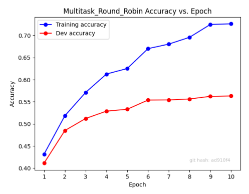


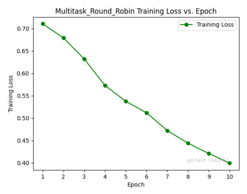
(a) Baseline & Cosine Sim. & SMART Reg. λ=5 (b) Baseline & Cosine Sim. & SMART Reg. λ=0 (c) Baseline & Cosine Sim. & SMART Reg. λ=5 (d) Baseline & Cosine Sim. & SMART Reg. λ=0

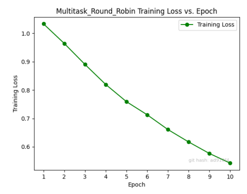
Figure 5: SMART Reg λ=5 (R6) vs. SMART Reg λ=0 (R7)

Figure 6 shows that PCGrad failed to improve performance. The overall performance fractionally from 0.563 (R4) to 0.554 (R8). The task specific performances were also fractionally reduced. Overfitting is less pronounced with PCGrad than without it.









(a) Baseline & Cosine Sim. & PCGrad (b) Baseline & Cosine Sim. (c) Baseline & Cosine Sim. & PCGrad (d) Baseline & Cosine Sim.

Figure 6: Baseline & Cosine Sim. & PCGrad (R8) vs. Baseline & Cosine Sim. (R4)

Varying hyperparameters (R9) failed to improve performance. Section A.6 shows that performance is sensitive to learning rates but not to increasing dropout probabilities. Changing the Para forward prediction to a shallower network failed to improve performance. Making a deeper STS network failed to improve performance. See A.7 for the results of the changing layers experiment in R10.

The BERT LayerNorm fix (R11) was introduced to uplift the performance in Objective 2 pretrain results for CFIMDB and correctly applied dropout to the sublayer output per Devlin et al. (2018) [1].

5 Analysis

5.1 Knowledge Transfer Across Tasks

The multitask round robin results showed a significant performance improvement during fine-tuning over pretraining. This suggests that knowledge transfer across tasks is occurring. The uplift is most pronounced in SST and STS. Mahabadi et al. (2021) state: *"When target datasets have limited training data, multi-task learning improves the performance compared to individually trained models"* [18]. While there is no benchmark for the single task performance of SST, STS and Para, it's hypothesised that these tasks are sufficiently similar to benefit from each other's training sets.

5.2 Cosine similarity

Cosine similarity boosted the performance for all tasks even though it was only applied to STS. One reason might be that knowledge transfer across tasks, since the benefit confers to Para and SST. On textual similarity specifically, Li et al. (2020) state that *"sentences from the pre-trained language models without fine-tuning have been found to poorly capture semantic meaning of sentences"* [19].

Cosine similarity directly addresses this problem by taking the hidden states of all input tokens rather than the CLS token, reducing dimensionality via a 1d pooler, passing it through a network with learning parameters and then into a cosine function [10].

5.3 Additional pretraining

One explanation why additional pretraining failed to improve performance is that vanilla BERT parameters may already encode the relevant information for a paraphrase detection task. It could also be argued that paraphrase detection is a simpler task than semantic similarity and sentiment classification.

5.4 Overfitting

Overfitting was frequently observed during finetuning around epochs 5-6 without regularization. Jiang et al. (2019) experimented with 6 epochs as their upper bound with SMART regularization [11] suggesting that overfitting persists after 6 epochs even with regularization. Thus the experiments conducted could have terminated earlier rather than train on the default of 10 epochs. Zhang et al. (2021) asserted that 3 epochs is too few for fine tuning , which gives a lower bound [2].

5.5 Regularization

while regularization reduced training accuracies it did not result in a commensurate uplift in dev accuracies. This is likely caused by an improper implementation of the SMART Regularization library rather a problem with the technique itself. The SMART Regularization experiment isn't directly comparable to R4 either because the forward prediction layers were modified to fit the SMART Regularization API, which is yet another limitation of this experiment.

5.6 Gradient Surgery

Gradient surgery did not improve performance nor did it impair performance. One possible explanation is that the tasks don't in fact have *"conflicting gradients"* [7].

5.7 Learning rate and dropout

The reason why the learning rate experiments did not improve performance is that exists a sweet spot for learning rates during fine tuning. According to Sun et. al (2019), BERT required a learning rate of 2E-5 to avoid the catastrophic forgetting problem [14], which is close to the default learning rate of 1E-5 for fine tuning. The results for increasing dropout to 0.4, 0.5 showed negligible impact to the performance. This suggests that there may be too many redundant parameters in the forward predictions for each of the tasks or that the training data

5.8 Layer changes

A deeper STS architecture did not improve STS performance as shown in Appendix A.7. One reason why a deeper network didn't improve performance that the number of parameters exceeds the number of training data samples as mentioned in 4.2, which agrees with the higher dropout hypothesis.

6 Conclusion

This project implemented minBERT and extended it to a multiclass model trained via a round robin curriculum on down sampled data. The best result was an overall prediction accuracy of 0.583 on the dev set across the three downstream NLP tasks. This model then achieved an overall prediction accuracy of 0.582 on the test set. While this is better than the random chance baseline of 0.3, the overall performance is still deemed poor. This model employed cosine similarity extension and fine tuning. Experiments with additional pretraining, SMART regularization, gradient surgery and hyperparameter tuning, and layer changes did not improve the performance. The reasons why are presented in the Analysis section. Future work should focus on balancing classes in the data before training to improve generalisation on the dev set and more experimentation with the embeddings.

References

[1] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *https://arxiv.org/abs/1810.04805*, 2018.

[2] Tianyi Zhang, Felix Wu, Arzoo Katiyar, Kilian Q. Weinberger, and Yoav Artzi. Revisiting few-sample bert fine-tuning. In *ICLR 2021*, 2021.

[3] Thibault Phang, Jason annd Févry and Samuel Bowman.R. Sentence encoders on stilts: Supplementary training on intermediate labeled-data tasks. In *https://arxiv.org/abs/1811.01088*, 2018.

[4] Cheolhyoung Lee, Kyunghyun Cho, and Wanmo Kang. Mixout: Effective regularization to finetune large-scale pretrained language models. In *ICLR 2021*, 2020.

[5] Joseph Warsham and Jugal Kalita. Multi-task learning for natural language processing in the 2020s: where are we going? In *https://arxiv.org/pdf/2007.16008.pdf*, 2020.

[6] Zixuan Ke, Bing Liu, Nianzu Ma, Hu Xu, and Lei Shu. Achieving forgetting prevention and knowledge transfer in continual learning. In *NeurIPS*, 2021.

[7] Tianhe Yu, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. Gradient surgery for multi-task learning. *arXiv preprint arXiv:2001.06782*, 2020.

[8] CS 224N Teaching Staff. Cs 224n: Default final project: minbert and downstream tasks. In *http://web.stanford.edu/class/cs224n/project/default-final-project-bert-handout.pdf*, 2023.

[9] Qiwei Bi, Jian Li, Lifeng Shang, Xin Jiang, Qun Liu, and Hanfang Yang. Mtrec: Multi-task learning over bert for news recommendation. in findings of the association for computational linguistics. In *ACL 2022, pages 2663–2669*, 2022.

[10] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP), pages 3982–3992*, 2019.

[11] Haoming Jiang, He Pengcheng, Xiaodong Chen, Weizhu an Liu, Jianfeng Gao, and Tuo Zhao. Smart:robust and efficient fine-tuning for pre-trained natural language models through principled regularized optimization. In *https://arxiv.org/abs/1911.03437*.

[12] Flavio Schenider. Pytorch - smart: Robust and efficient fine-tuning for pre-trained natural language models. In *https://github.com/archinetai/smart-pytorch*, 2022.

[13] Wei Cheng Tseng. Pytorch-pcgrad. In *https://github.com/WeiChengTseng/Pytorch-PCGrad*, 2021.

[14] Chi Sun, Xipeng Qiu, and Xuanjing Huang. How to fine-tune bert for text classification? In *In Chinese Computational Linguistics: 18th China National Conference, CCL 2019, Kunming, China, October 18–20, 2019, Proceedings 18, pages 194–206*, 2019.

[15] Richard Socher, Alex Perelygin, Jean Y. Wu, Jason Chuang, Christopher D. Manning, Andrew Y. Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Conference on Empirical Methods in Natural Language Processing*, 2013.

[16] Kornel Csernai, Shankar Iyer, and Nikhil Dandekar. Quora dataset release: Question pairs. In *https://quoradata.quora.com/First-Quora-Dataset-Release-Question-Pairs*, 2012.

[17] Eneko Agirre, Daniel Cer, Mona Diab, Gonzalez-Agirre Aitor, and Weiwei Guo. * sem 2013 shared task: Semantic textual similarity. In *Second joint conference on lexical and computational semantics (*SEM), volume 1: proceedings of the Main conference and the shared task: semantic textual similarity*, 2013.

[18] Rabeeh Mahabadi, Sebastian Ruder, Mostafa Behghani, and James Henderson. Parameter-efficient multi-task fine-tuning for transformers via shared hypernetworks. In *https://arxiv.org/pdf/2106.04489.pdf*, 2021.

[19] Bohan Li, Hao Zhoi, Mingxuan He Junxian, Wang, Yang Yiming, and Lei Li. On the sentence embeddings from pre-trained language models. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, pages 9119–9130*, 2020.

A Appendix: Additional Plots

A.1 Baseline

Pretrain highlights poor performance. The overall and task specific accuracies don't increase with each epoch.

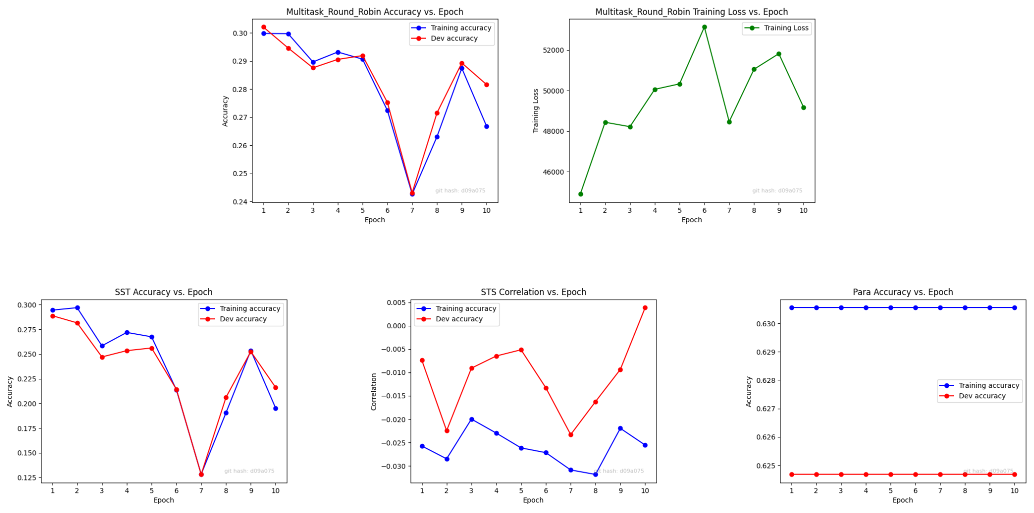


Figure 7: Baseline Pretrain

Finetuning accuracies, on the other hand, shows accuracy improvements until it plateaus. Finetuning continually exhibits overfitting for STS and SST datasets. Overfitting does not seem to be obvious for the Para data set. The training accuracies for Para exhibits collapses during epochs 3 and 8, suggesting that there may have been competing optimisation goals between the tasks.

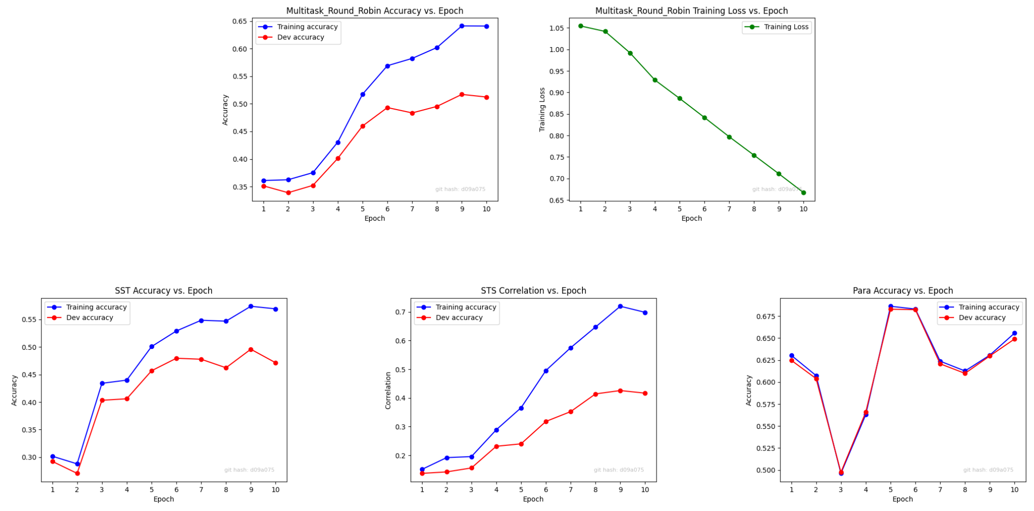


Figure 8: Baseline Finetune

A.2 Baseline & Cosine Similarity

Cosine similarity made significant improvements for the STS dataset even without finetuning, lifting its results to the order of 0.4 for Pearson correlation, which is roughly on par with the finetuning case. SST accuracies stopped being erratic over the epochs. There was zero change in Para and SST, however, during pretraining. There was zero change in Para, while the SST accuracies remained erratic.

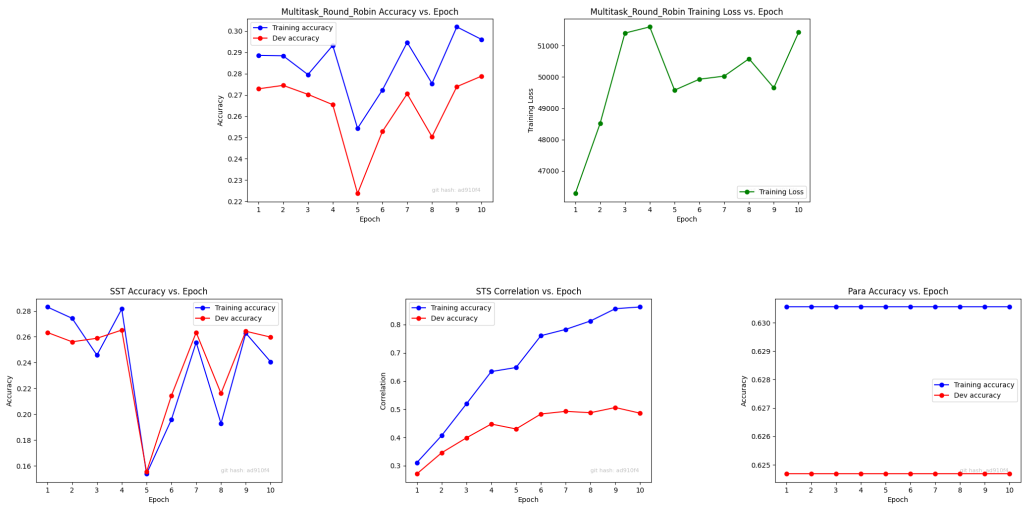


Figure 9: Cosine Similarity Pretrain

Overfitting occurred for STS and SST data sets while Para does not exhibit overfitting at all. The trough and peak in the training accuracies of the Para data set still persists in epochs 3 and 7-8 but their magnitude is smaller.

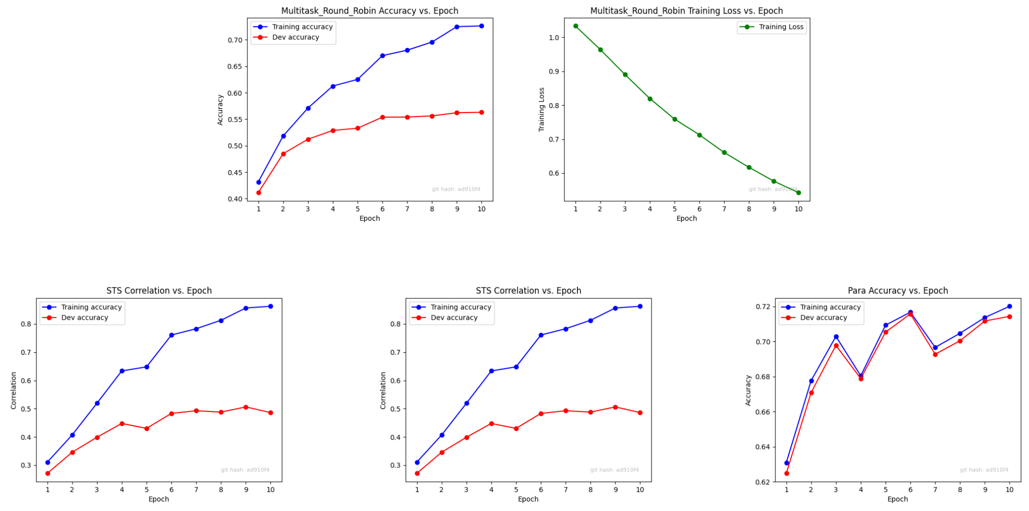
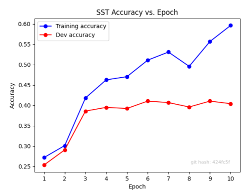


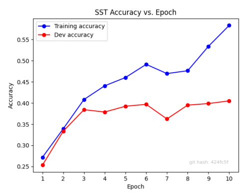
Figure 10: Cosine Similarity Finetune

A.3 Baseline & Cosine Similarity & Regularization

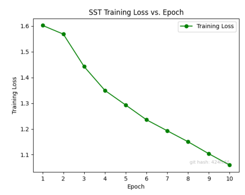
A range of SMART regularization experiments was first conducted on STS alone to explore whether varying the λ parameter had any effect, it unfortunately did not. Experiments were conducted with changing the KL loss function to square loss and MSE for the STS task during multitask round robin training but this yielded even worse performance than KL loss, which was not expected at all. KL loss, according to the paper, is most appropriate for classification tasks but the round robin implementation for STS is a regression task.



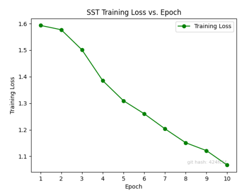
(a) Regularization λ_s=0.02



(b) None

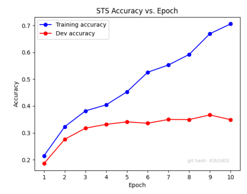


(c) Regularization λ_s=0.02

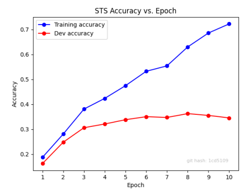


(d) None

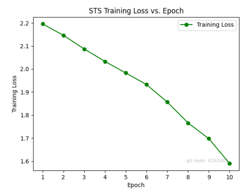
Figure 11: Regularization: SST Single Task



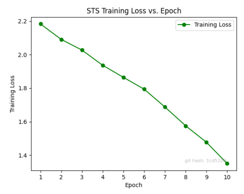
(a) Regularization STS λ_s=1



(b) None

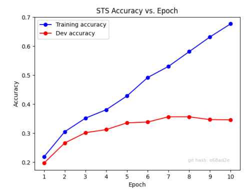


(c) Regularization STS λ_s=1

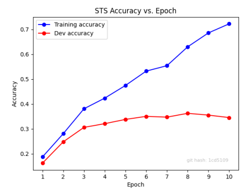


(d) None

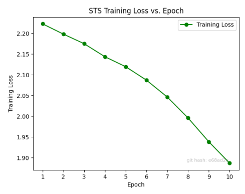
Figure 12: Regularization: STS Single Task $\lambda_s = 1$



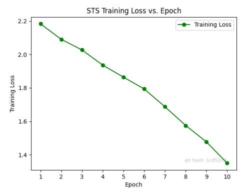
(a) Regularization STS λ_s=3



(b) None

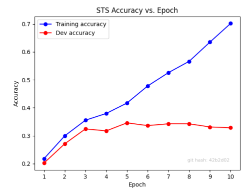


(c) Regularization STS λ_s=3

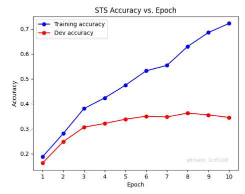


(d) None

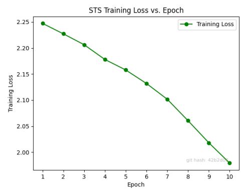
Figure 13: Regularization: STS Single Task $\lambda_s = 3$



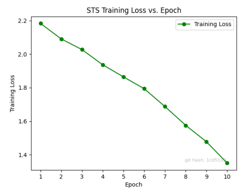
(a) Regularization STS λ_s=5



(b) None



(c) Regularization STS λ_s=5



(d) None

Figure 14: Regularization: STS Single Task $\lambda_s = 5$

A.4 Baseline & Cosine Similarity & Additional Pretraining

Noticeably, there is an instant uplift in the para performance at epoch 1. Para appears to hit the upper limit of its performance as implemented in this project. The additional pretraining benefit did not confer to the SST and STS tasks.

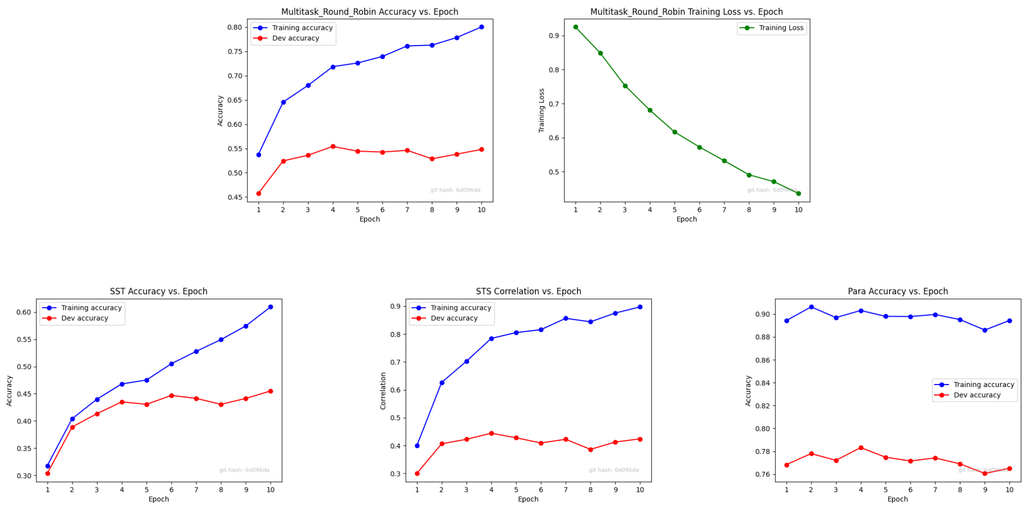


Figure 15: Additional Pretraining

A.5 Baseline & Cosine Similarity & PCGrad

PCGrad had better performance than additional pretraining, which was unexpected. However, para was impaired. Overfitting continues to be observed during PCGrad. Since PCGrad did not materially improve performance, it suggests that the gradients aren't significantly conflicting between the three tasks.

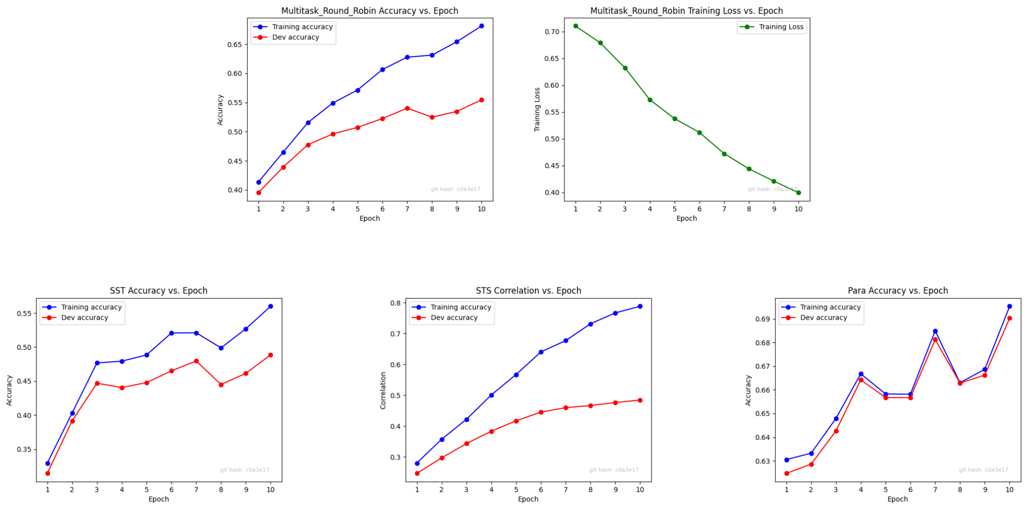


Figure 16: PCGrad

A.6 Baseline & Cosine Similarity vs. Hyperparameter Tuning

A.6.1 Learning Rate 1E-4, 1E-6, 1E-7

Learning rate 1E-4 had disastrous performance results. The performance with learning rates 1E-6 and 1E-7 trended lower over six epochs vs. a learning rate of 1E-5. Finetuning is thus very sensitive to learning rate.

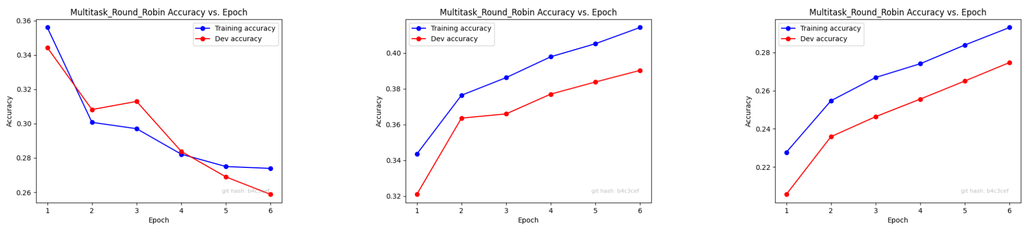


Figure 17: Learning Rate 1E-4, 1E-6, 1E-7

A.6.2 Dropout 0.4, 0.5, 0.6

Higher dropout didn't materially improve performance vs. the default of 0.3 but it did not reduce performance either.

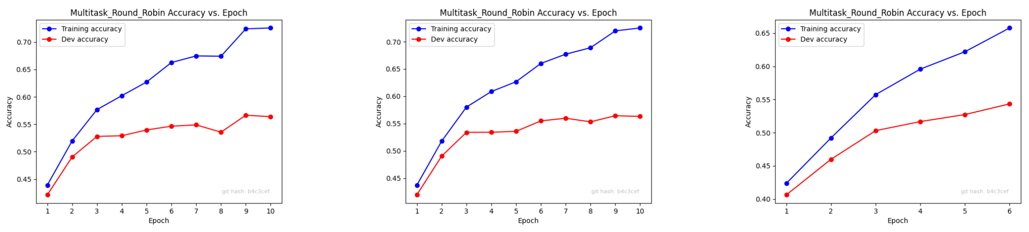


Figure 18: Dropout Rate 0.4, 0.5, 0.6

A.7 Layer changes

A.7.1 Shallow Para Network

A shallower para network was experimented with to explore whether the second layer was redundant. However, one linear layer alone impaired the Para results and thus the overall performance of the multitask model.

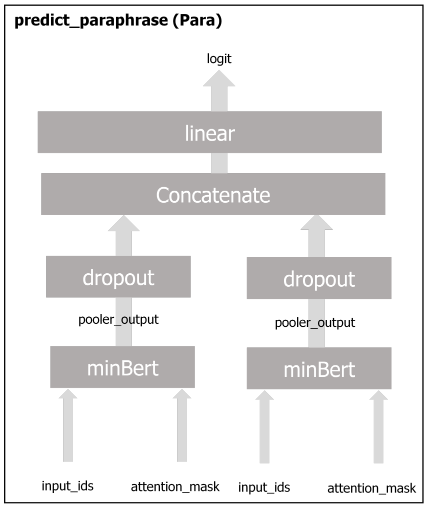


Figure 19: Task Specific Neural Network Models

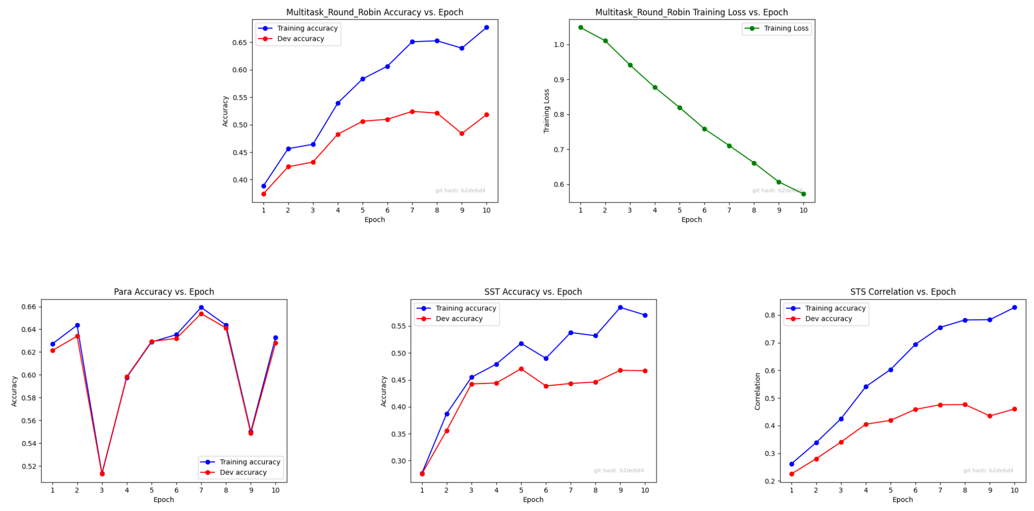


Figure 20: Shallower Para Network

A.7.2 Deep STS Network

Conversely, a deeper STS network was explored in an attempt to uplift the STS performance. However, a three linear layer network did no better than a single linear network.

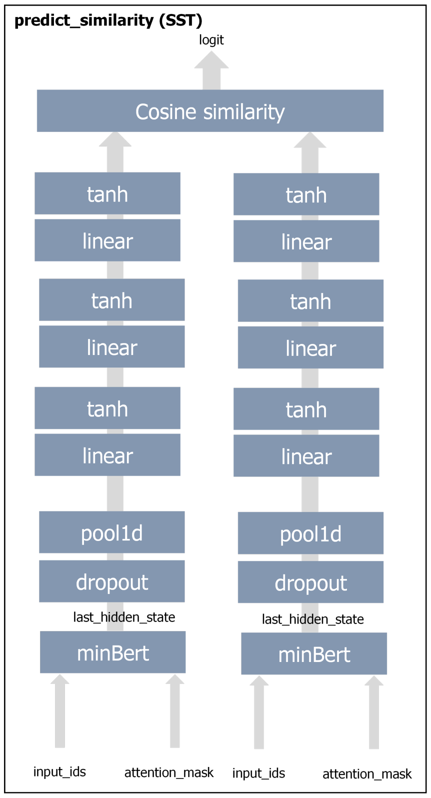


Figure 21: Task Specific Neural Network Models

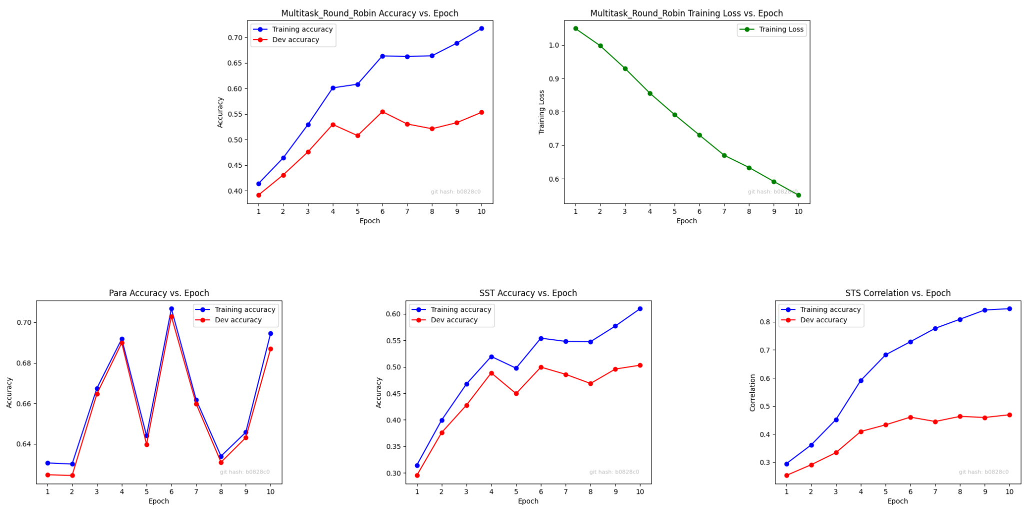


Figure 22: Deeper STS Network