Enhance minBERT's Performance on Multiple Sentence-Level Tasks using Downstream Techniques

Stanford CS224N Default Project

Chung Ching Cheung, Mandy Leung, Vibhaakar Sharma
Department of Computer Science
Stanford University
cccheung@stanford.edu, mandyl@stanford.edu, vibhs@stanford.edu

Hi Tathagat, Please note that we are sharing late days on this report. Thank you.

Abstract

As the world becomes increasingly digital, the volume of daily text data is growing unprecedentedly. This data growth has made it more critical to develop powerful natural language processing (NLP) models that can accurately understand and analyze text. The BERT model, introduced in 2018, has gained popularity for many NLP tasks due to its impressive performance on various benchmarks. However, BERT can be computationally expensive to train and difficult to fine-tune for specific tasks. We aim to demonstrate the effectiveness of an efficient minBERT model on various NLP tasks and explore the effectiveness of downstream techniques in improving the performance of these tasks.

1 Key Information to include

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

2 Introduction

In this project, we implemented the minBERT model successfully and demonstrated its effectiveness on three different NLP tasks: sentiment classification, paraphrase detection, and semantic textual similarity. Details on these tasks can be found in the Default Project Handout [1]. We explored various downstream techniques to improve the model's performance and detailed the results of these experimentations. The major contribution of this work is providing empirical evidence on the effectiveness of BERT and these downstream techniques on the 3 targeted tasks. We discovered through these experimentations that some techniques lift the performance of all tasks, while others are task-specific, and yet a third group of techniques failed to provide the improvement we hoped for. We detailed the motivation for employing each of these techniques in the paper and discussed our hypothesis on why they delivered or failed to deliver the intended performance lift. In doing so, we hope to inform the strengths and weaknesses of each technique and their relevance for each task. Lastly, we qualitatively evaluated the performance of our model and proposed future research directions.

3 Related Work

BERT is a highly successful and widely adopted model on which many applications are built. Several notable version of BERT has been built for various applications:

- **DistilBERT:** DistilBERT [1] is a smaller and faster version of BERT that uses a distillation technique to compress the original BERT model while retaining most of its performance. It has

[1] http://web.stanford.edu/class/cs224n/project/default-final-project-bert-handout.pdf

fewer layers, smaller hidden dimensions, and fewer attention heads than BERT, making it faster and more memory-efficient.

- **MobileBERT:** MobileBERT [2] is an even smaller and faster version of BERT that is designed for mobile and edge devices. It achieves this by using a series of model compression techniques, such as knowledge distillation, pruning, and quantization, to reduce the size and computational complexity of the model.
- **RoBERTa:** RoBERTa [3] is a variant of BERT that was trained using a larger corpus and longer training time. Although it is smaller and faster than some of the other models on this list, it has been shown to outperform BERT on several NLP tasks while still being computationally efficient. RoBERTa achieves this by using a larger batch size during training and applying dynamic masking to improve the model's ability to handle unseen data.

In this paper, we implemented a minBERT model and employed several downstream techniques to improve its performance on three selected tasks. MinBERT is described in the default project handout[2].

Our approach is inspired by several research papers. In the design and experimentation of the architecture transforming BERT embeddings into downstream output, we took inspiration from *"BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding"* [4]. In particular, the author suggested a softmax output layer for the STS task. In another paper, *"How to Fine-Tune BERT for Text Classification?"* [5], the author discussed and experimented with several finetuning techniques, including layer-wise decreasing learning rate to overcome catastrophic forgetting problem, within task, in-domain and cross-domain pre-training, and multi-task and single-task pre-training. In one paper suggested by the project handout, *"Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks"* [6], the author proposed using cosine-similarity between the two sentence embeddings and applied mean-squared-error loss as the objective function. In another paper, *"Efficient natural language response suggestion for smart reply"* [7], the author introduced the multiple negatives ranking loss with a batch of sentence pairs such that the loss minimizes the distance between similar pairs and maximize the distance between dissimilar pairs. In this paper, we borrowed some of these techniques but also innovated a few other finetuning techniques in our experimentation. In another paper, *"Simple Contrastive Learning of Sentence Embeddings"* [8], the author proposed an unsupervised training technique in which two different embeddings of the same sentence is fed to the model to encourage these embeddings to be aligned with each other, while other sentences in the same dataset is used as negative example to encourage the model to separate these embeddings as far as possible. Our paper experimented with employing this technique with our dataset. Another multitask finetuning technique we have explored in our experiments is the additive multitask loss, which is covered in the paper *"MTRec: Multi-Task Learning over BERT for News Recommendation"* [9]. We conducted experiments to improve BERT fine-tuning using self-ensemble and self-distillation techniques from paper [10]. Specifically, we found that self-ensemble significantly enhanced the model's performance with each epoch during training.

4 Approach

Besides implementing minBERT, we experimented with several techniques both inspired by the research quoted in Section 3 along with some innovative techniques based on our understanding of the model behavior. These techniques include various finetuning approaches using both task-specific and in-domain data (sections 4.3, 4.5, 4.6, and 4.9), model architecture improvements (sections 4.4, 4.7, and 4.8), and ensemble modeling (section 4.10). Following these techniques, we experimented with the impact of tuning hyperparameters on model performance (section 5) and performed qualitative analyses of the results (section 6). By doing so, we hope to gain a deeper understanding of the strengths and weaknesses of minBERT and the downstream techniques, as well as propose future research directions to further improve BERT performance.

4.1 Baseline

First, we established a simple "random number generator" baseline. The BERT model should easily beat these baselines, which are 0.2 , 0.5 and 0 [3] for Sentiment Classification, Paraphrase detection and STS respectively.

[2] http://web.stanford.edu/class/cs224n/project/default-final-project-bert-handout.pdf

[3] Sentiment Classification: 5 classes of outputs with equal probability. Paraphrase Dection: Binary classification. STS: No correlation between random generator output and target.

4.2 Base Model

We start with the base model which uses only the pretrained embeddings without finetuning them with the downstream tasks. The prediction functions for the 3 tasks are as follows: **Sentiment Classification:** We first compute the pooler layer of BERT, and apply a dense layer with output size of 5 and a dropout layer to the pooler layer to get the logits. Cross entropy loss is used as the loss function. **Paraphrase Detection:** We compute the pooler layer of each of the sentences in the pair and concatenate them into a vector. We apply a dense layer of output hidden size 1 and a dropout layer to the concatenated vector to get the logits. Binary cross entropy is used as the loss. **STS:** The prediction function is same as paraphrase detection except we apply an additional sigmoid layer to the final logits to scale it to $[0, 1]$ and multiply the results by 5. The loss function is mean square error. Base model performance is 0.391, 0.632, and 0.254 for dev data for SST, PARA and STS respectively.

4.3 Finetuning

We experimented with various techniques in finetuning the model using the training portion of the 3 datasets relevant to the 3 tasks. Our key findings are that (1) finetuning significantly improved performances of all 3 tasks, (2) a single loss function approach with varying batch sizes which enables full data usage, despite very small batch size, produces top overall performance, (3) single task finetuning neither generalizes well to other tasks nor guarantees optimal performance for the targeted task, and (4) while finetuning improves SST performance against the base model initially, performance quickly deteriorates upon more finetuning epochs and more complex techniques in finetuning. We will discuss (4) in more detail in *Section 4.6*.

Table 1: Dev performance based on finetuning techniques discussed above, after 10 epoch

Loop type	Data used	Batch Size	Loss Function	SST	Para	STS	Overall
Base model	None	N/A	N/A	0.391	0.632	0.254	0.426
Single	All 3	Same for all	Summed	**0.512**	0.722	0.378	0.537
Sequential	All 3	Same for all	Vary by task	0.480	0.816	0.523	0.606
N/A	SST	8	SST	0.502	0.631	0.080	0.404
N/A	PARA	8	PARA	0.209	**0.818**	0.154	0.394
N/A	STS	8	STS	0.182	0.592	0.469	0.414
Single	All 3	Separate	Summed	0.499	0.815	**0.587**	**0.634**

Single loop, consistent batch size: In our first experiment, we deployed a single loop across all 3 data sources, using the same batch size for all data sources. The loss function used was the sum of the loss function for the 3 tasks which is known as the vanilla objective for multi-task learning[4]. This method has the shortest finetuning time, but also underutilized a lot of the data, especially in the Quora (PARA) dataset due to imbalance of data across the 3 sets. Since the smallest dataset among the 3 (STS) contains only about 6K lines of data, this single loop is only able to utilize 6K out of 142K lines of data in the Quora (PARA) set in each epoch. While we shuffle the data for each epoch, it will still take more than 20 epochs to get close to fully utilizing the data. Hence, as expected, the model performance was sub-optimal.

Sequential loop: We propose a new approach in which finetuning for each task is done sequentially using separate loss functions for each task. This ensures that almost all data in all 3 datasets is utilized in each finetuning epoch. The time requirement for this is much larger, mostly attributed to the fuller data usage. This dramatically improved model performance.

One task: Another alternative we experimented with is training the model with one set of data only using the loss function of the relevant task. We then attempt to apply the embedding and model parameters to all 3 tasks. We found that this type of finetuning does not generalize well beyond the task it was trained on. Further, it does not guarantee optimal performance even in the task that it was trained on.

Single loop, varying batch size: In this experiment, we maintained single loop finetuning across all 3 data sources using the vanilla loss function, but varied the batch sizes across tasks such that all of the data can be utilized in each epoch. This led to very small batch sizes for STS and SST

[4] `http://cs330.stanford.edu/fall2020/slides/cs330_multitask_transfer_2020.pdf`

versus a much larger batch size for PARA. Our initial concern was that the small batch size for STS and SST would lead to unstable updates and worse performance on these tasks. To our surprise, this experiment resulted in the best overall performance and close to the best performance for all 3 tasks. This may be attributed to the benefit of having a combined loss function across 3 tasks which weren't possible in the *Sequential loop* or *One Task* scenarios.

4.4 Non-linearity, Feedforward and Normalization

In addition to the architecture describes in *Base model*, we added a non-linear layer, a feedforward layer and layer normalization to the prediction function of each of the task. This reduces the performance of the SST tasks mildly but significantly improves the performance of the PARA and STS tasks. The mild decline in performance in the SST task might be attributed to insufficient finetuning data to handle the complexity of the multiple layers. The improvement could be attributed to (1) the feedforward layer increases the number of parameters for learning, (2) the layer normalization helps reducing uninformative variation in the activations and providing more stable input to the next layer. In fact, research shows that layer normalization improves gradients in the backward pass [11].

Table 2: Dev performance based on varying model architecture, after 10 epoch

Model characteristics	SST	Para	STS
Base model with finetuning at equal batch size (for comparison)	**0.512**	0.722	0.378
Added non-linear layer	0.506	0.748	0.412
Added non-linear, feedforward and normalization layers	0.499	**0.758**	**0.514**

4.5 Additional Finetuning Data

Another extension we have tried is to include extra data in model training. In particular, we target at STS task since the provided training data has the smallest size with 6k data and the model performance on this task is the worst. We hence used additional data Stanford Natural Language Inference (SNLI) which has a size of 540k and has similar favor as the STS benchmark dataset with each sentence pair associated with a label of whether they are of entailment, neutral and contradiction. In our experiment on training with this additional data for 10 epochs, i.e. having 4 dataloader in the training loop, the embedding learned from this data improves the model performance on STS task from dev accuracy 0.587 by using 3 dataset to 0.649 by using all 4 dataset.

Table 3: Dev performance with additional SNLI data, after 10 epoch

Model characteristics	SST	Para	STS
Before SNLI data (for comparison)	**0.499**	0.815	0.634
After SNLI data	0.496	**0.816**	**0.640**

4.6 Pretraining before Finetuning

One technique to ensure that model parameters do not sway too far due to finetuning is to perform model training in two steps. The first step pretrains at a higher learning rate while keeping the BERT embedding frozen. The second step finetunes at a much lower learning rate while unfreezing the BERT embeddings. According to conclusion #4 in *Section 4.3*, we hypothesize that finetuning is swaying the embeddings too far from BERT embeddings for the SST task, causing more harm in the model performance than good. We tested this two step technique with 10 epoch pretraining at a learning rate of 10^{-3} followed by 10 epoch of finetuning at 10^{-5}. With this technique, we are able to produce SST performance exceeding that of the Base model and any of the finetuning models in the previous sections. This validates our hypothesis above.

Appendix A contains diagrams comparing results from *direct finetuning* vs. *2-step pretraining before finetuning*. While finetuning clearly improves SST performance on the first epoch, training and dev accuracy quickly diverges in subsequent epochs, which means that the model quickly overfits to the training data. Futher iteration of this work can consider larger batch sizes (more stable results) and lower finetuning rate to correct this issue.

Table 4: Performance based on "Pretraining before Finetuning" compared to baseline

Technique	SST Acc (dev)	Para Acc (dev)	STS Corr (dev)
Base model	0.391	0.632	0.254
Finetuning with varying batch size	0.499	**0.815**	**0.587**
Pretraining before Finetuning	**0.513**	0.792	0.433

4.7 Utilizing BERT Output

While it is common to utilize the "pooler output" (CLS token) from the BERT model for downstream tasks, we evaluated two other means of utilizing the BERT embeddings, namely using the maximum value and average value across each dimension of the full output in BERT. In our test, although each of these methods has its advantages, neither improved the performance of the tasks significantly.

Table 4: BERT sequence output compared to pooler output after 10 epochs

BERT output	SST Acc (dev)	Para Acc (dev)	STS Corr (dev)
Pooler (CLS token)	0.499	**0.815**	0.587
Full, averaged	0.490	0.738	**0.595**
Full, maximum	**0.501**	0.706	0.537

4.8 Cosine Similarity and Softmax Output Layers

To measure the semantic similarity of two sentences, a natural way is to compute the cosine similarity between the sentence embeddings where a value of 1 means they have similar meaning and 0 means unrelated [5]. Another function we could use is the softmax classification as used in BERT paper [12]: we concatenate the two embeddings vectors into one vector $V \in \mathbf{R}^{2h}$ and apply classification layer weights $W \in \mathbf{R}^{k \times 2h}$ where k is the number of labels and h is the hidden layer size. Then we compute the softmax classification loss as softmax(VW^T). We have applied both techniques in our model. Cosine similarity is used in STS prediction function where we concatenate the two sentences embeddings and apply cosine similarity; and softmax classifier is used in SNLI prediction function where we concatenate the two sentences embeddings plus the difference between them and apply the softmax as suggested in the paper SBERT [6]. Another technique we have tried is the multiple negatives ranking loss applied to the SNLI data, where only the entailment sentence pairs are used in the training so that each sentence pair, (premise, hypothesis), has premise similar to hypothesis while cross pair premise and hypothesis are not similar. However, the model trained from this loss doesn't generalize well for STS task. This may due to the fact that model learned from this loss works well for binary similarity classification but STS has multiple labels.

4.9 Contrastive Learning

According to the SimCSE paper [8], one way to improve BERT embeddings is to provide each sentence to the model twice, generating two different embeddings. These two embeddings can be used as positive pairs in training while the original sentence and the rest of the mini-batch can be used as negative pairs. Unfortunately, implementation of the this method resulted in worse model performance in the tasks we tested. There can be several reasons contributing to this, for example, there could be implementation difference in our model compared with the original paper, and the size of data we have was insufficient.

4.10 Ensemble Model

Ensemble BERT combines multiple BERT models to achieve better performance than using a single model. However, training multiple BERTs can be expensive regarding computational resources and time. The paper proposes a self-ensemble BERT model that combines the intermediate models at different time steps in a single training phase to reduce the training cost. In this approach, each BERT at each time step is regarded as a base model and combined into a self-ensemble model. The self-ensemble BERT with parameter averaging is used to combine the intermediate models. The averaged parameters over T time steps are denoted by $\bar{\theta}$. The self-ensemble BERT with parameter averaging is defined as follows: BERTSE$(x; \bar{\theta})$ = BERT$(x; \frac{1}{T}\sum_{\tau=1}^{T}\theta_\tau)$ Here, the self-ensemble BERT with parameter averaging takes a single input x and produces an output based on the averaged parameters θ. Using the averaged parameters of intermediate models tends to produce a more accurate model than using the final weights directly. This is because averaging over multiple models can reduce the impact of outliers and errors in individual models. This technique is based on the work of Polyak and Juditsky (1992), who showed that averaging model weights over training steps can improve model accuracy.

[5] `https://www.sbert.net/docs/usage/semantic_textual_similarity.html`

5 Experiments

5.1 Data

We use the following datasets for this project: 1) Stanford Sentiment Treebank (SST) [6] 2) CFIMDB, 3) Quora dataset [7], 4) SemEval STS Benchmark dataset [13], and 5) The Stanford Natural Language Inference (SNLI) Corpus [8]. SNLI is a collection of 570k human-written English sentence pairs labeled for balanced classification with the labels entailment, contradiction, and neutral. Details on the other 4 datasets can be found in the Default Project Handout.

5.2 Evaluation method

For sentiment analysis on movie reviews, SST & CFIMDB (only for initial test), and paraphrase detection on question pairs (Quora), we want to measure how accurate the model predict the label so we use accuracy as the evaluation metric and compute the values for both the dev and test datasets. For the semantic textual similarity task (SemEval STS), we want to measure the association between the predicted and the true values so we compute the Pearson correlation of the actual similarity with the predicted similarity on both the dev and test datasets, the higher the correlation the better the prediction. We did not apply evaluation metric on the SNLI dataset since it is not in our main objective but we use this extra dataset to enhance the embedding and thus the model performance.

5.3 Experimental details

To understand how the model performs under different configurations, we ran the following experiments:

Table 5: Hyperparameter Tuning Results

Modification or parameter	Model Type	SST Acc (dev)	Para Acc (dev)	STS Corr (dev)
Original parameterization	Pretrain	0.391	0.632	0.254
Original parameterization	Finetune	0.512	0.722	0.378
Dropout prob = 0.5	Finetune	0.511	0.744	0.354
Learning rate = 1e-3	Finetune	0.253	0.625	0.078
Learning rate = 1e-2	Pretrain	0.387	0.626	0.048
Learning rate = 1e-5	Pretrain	0.299	0.625	0.202
Hidden layers = 6	Finetune	0.507	0.742	0.372

- **Dropout:** Increased dropout probability to 0.5 to study regularization effects. Increasing dropout probability improved paraphrase accuracy and decrease STS correlation mildly. The small change in performance is likely due to noise. It is unlikely that increasing dropout further would improve model performance.
- **Learning rate:** Higher learning rate didn't improve evaluation metrics and seems to jumping all over, while lower learning rate didn't converge within 10 epochs.
- **Number of hidden layers:** Reducing the hidden layer, which corresponds to the number of attention blocks, from default 12 to 6 slightly reduce the performance of SST and STS tasks but improve on Paraphase task. It is possible that a smaller model is still complexed enough to learn the text representation.
- **Number of epochs:** In our final model (which deploys direct finetuning with varying batch size), best performance was reached after 13 epochs (tested up to 30 epochs). In our earlier models (which deploys direct finetuning with equal batch size of 8 for all tasks), best performance was reached after 97 epochs (tested up to 100 epochs). Please see Appendix B.1 and B.2 for details comparison. In the plots for train and dev accuracy in Appendix B, we can see that the final model achieves dev accuracy of 0.6 in 2 epochs whereas our earlier model needs around 15 epochs to achieve the same dev accuracy. This difference can be attributed to the fact that the equal batch size model failed to utilize all available data in each epoch, hence, requires a higher number of epochs to capture the benefit of all the data. In the plots of training loss, we can see that 100 epochs has much lower training loss (0.25) than 30 epochs (0.75). However, this doesn't mean the model performs better in 100 epoch but indicates that there is serious overfitting on the training set and model does not generalize well for dev set. This is evidenced by the sentiment task where the 100 epochs has achieved 1.0 train accuracy in less than 20 epochs while the dev accuracy remains flat over time.

[6] `https://nlp.stanford.edu/sentiment/treebank.html`

[7] `https://www.quora.com/q/quoradata/First-Quora-Dataset-Release-Question-Pairs`

[8] `https://nlp.stanford.edu/projects/snli`

- **Optimizers:** We tested optimization algorithms such as Stochastic Gradient Descent (SGD), Adagrad, Adam, and AdamW to optimize the BERT model. AdamW performed the best, achieving the highest accuracy on our benchmark datasets. The performance of AdamW was attributed to its ability to handle weight decay more effectively than others, leading to a better generalization of the model.

5.4 Results

Before jumping to the main results of experiments, our initial test of minBERT on SST gives pretrain dev accuracy of 0.390 and finetune dev accuracy of 0.520; while on CFIMDB it gives pretrain dev accuracy of 0.780 and finetune dev accuracy of 0.955. In both dataset, finetune model has better dev accuracy since minBERT parameters are updated and learned from the downstream tasks in finetune but not pretrain. Since the sentiment for CFIMDB is binary, both pretrain and finetune minBERT perform better for this task than SST which is a 5-label classification.

Summarizing the discussions in sections 4 and 5 above, the best overall results achieved by our model are achieved through the *"Finetuning with varying batch size"*, adding additional finetuning data, improved model architecture (including applying non-linear, feed-forward and normalization for all tasks and cosine similarity to STS task), and increased epoch count. Model performance significantly exceeds the performance of both the *"random generator"* and *"base model"* baselines. This is the model we uploaded to the leaderboard which yielded test performance of 0.514, 0.821, and 0.539 for the SST, PARA and STS tasks respectively (0.658 overall). At the task level, best SST performance is achieved through the *"Pretraining before finetuning"* model.

Table 6: Final model performance compared with baselines

Technique	SST Acc (dev)	Para Acc (dev)	STS Corr (dev)
Random generator	0.2	0.5	0
Base model	0.391	0.632	0.254
Final Model	0.510	**0.818**	**0.649**
Pretraining before Finetuning Model	**0.513**	0.792	0.433

Comparing the results between the Final Model and Interim Model [9] (charts are available in Appendix C), we have solved several model deficiencies in the Interim Model. These include (1) accuracy bouncing around for the PARA task, (2) early divergence of training and dev correlation performance and (3) early termination of model training before convergence. One challenge that remains unsolved is overfitting to the finetuning data for the SST task, leading to early diverge of training and dev performance in this space. Future work to correct this is proposed in *Section 7*.

6 Analysis

6.1 SST performance

In this section, we examine our model's performance on the SST task. First, the model's prediction tends to be in the mid-range (ratings 1, 2 and 3) rather than in the extremes (ratings 0 and 4) compared with the golden set. 80% of predicted values falls within the mid-range versus 72% in the golden set (details in Appendix H). One hypothesis is that the way we utilized the output from BERT is too aggregated, ignoring some of the extreme meanings of the sentence. Looking at the experimentation described in *Section 4.6*, we noticed that SST performance goes down when we use average values of the full BERT output and goes up when we use maximum values of the full output. This fits the hypothesis well.

Next, we looked at examples in which the predicted sentiment is more than 1 point away from the golden set. Two themes emerge from these examples. (1) Sentences involve double negatives or other complex forms of structure that convey the opposite ideas of what a short phrase might imply. One way to correct this is to employ a Recursive Neural Tensor Network as described in Lecture 13[10]. (2) The golden set of sentiments is not always reliable. We disagreed with some of the golden set sentiment ratings. (Examples can be found in Appendix E.)

[9] In this case, we are comparing to the model used in the Milestone Report, which is equivalent to the Base Model described in *Section 4.2* with finetuning using the 3 major datasets (without SNLI) at batch size of 8 and no architecture improvements.

[10] http://web.stanford.edu/class/cs224n/slides/cs224n-2023-lecture13-CNN-TreeRNN.pdf

6.2 STS performance

In this section, we examine why certain STS predicted similarity falls far from the golden set similarity using dev data. Since the evaluation metric for the STS task is correlation which can only be applied at the aggregated level, we need another definition "success" which can be applied at the per-example level. In this analysis, we compare the golden set similarity percentile against the predicted similarity percentile. We classify the top 10 percent of examples with the largest absolute distance between the two measurements as "failures" and the rest as "successes". Success and failure rates at each golden set similarity are about the same. Details can be found in Appendix I.

Further qualitative analysis reveals that "failure" examples with predicted similarity being too high tend to have strong n-gram similarity between the two sentences, but the model failed to recognize that the mismatched part of the sentences changes the meaning of the two sentences significantly. This implies that the model, while strong on understanding the surface meaning of words, fails to understand the sentence-level meaning at times.

On the other hand, "failure" examples with predicted similarity being too low tend to convey the same meaning using slightly different wordings (e.g. 'bathing' vs. 'bath-tub' and 'slicing' vs. 'cutting'). This reveals that despite BERT's many advantages, recognizing the meaning of similar words in context could still be challenging. Further examples can be found in Appendix D.

6.3 PARA performance

In this section, we will examine paraphrasing performance. While the overall accuracy of the model is about 80%, we observed that for the subset of sentences classified as positive in the golden set, accuracy is only 67%, showing a possible model bias. Confusion matrix can be found in Appendix J.

Looking at qualitative results, "failure modes" in this task are similar to that of STS. (1) Pairs of questions with strong n-gram overlap but different meaning (especially different question words) are often misclassified as paraphrase of each other. (2) Model failed to recognize different vocabulary with similar meaning in context. Example sentences can be found in Appendix G.

7 Conclusion

In this paper, we described our implementation of a minBERT model and experimented with various downstream techniques to improve its performance on the SST, PARA and STS tasks. Our main contribution is in identifying techniques that worked well in improving model performance. (1) Finetuning BERT embeddings with task-specific data improves model performance on targeted tasks dramatically. (2) When finetuning, vanilla loss function across multiple tasks outperforms separate gradient and loss functions for individual tasks. (3) To balance datasets with different number of data points during finetuning, vary the batch size for each dataset so that all data can be utilized in a single epoch. This improves model performance and reduces the number of epoch required for convergence. (4) Model architecture matters in model performance. In particular, we found that cosine-similarity-based prediction function works well for STS, and non-linear, feed-forward and normalization layers work well for STS and PARA tasks. (5) Additional in-domain (but not task specific) data can improve model performance. This was particularly helpful in the STS tasks due to small data size in the original dataset. (6) Pretraining before finetuning improves model stability and is especially helpful for the SST task.

In addition, several techniques we attempted did not result in model improvements. Single task finetuning neither generalizes well nor significantly improves performance in the targeted task. Hyperparameter testing (beyond batch size which was discussed above) also mostly showed no performance improvement beyond the base model setup.

Lastly, several experiments and analyses highlighted promising research areas for the future. In particular, there are several techniques that can be explored in improving SST performance. (1) Our experiments with using maximum versus average values from the BERT sequence output supports our hypothesis that the pooler output doesn't provide enough details for the SST task. Further experimentation in this space could be beneficial. (2) Qualitative analysis reveals that the model had trouble understanding complex sentence structure with double negative meanings. Applying Recursive Neural Tensor Network or similar techniques might help. (3) Early divergence in training vs. dev performance in finetuning shows that learning rate can be lowered for this task.

References

[1] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of BERT: smaller, faster, cheaper and lighter. *CoRR*, abs/1910.01108, 2019.

[2] Zhiqing Sun, Hongkun Yu, Xiaodan Song, Renjie Liu, Yiming Yang, and Denny Zhou. Mobilebert: a compact task-agnostic BERT for resource-limited devices. In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel R. Tetreault, editors, *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pages 2158–2170. Association for Computational Linguistics, 2020.

[3] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692, 2019.

[4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

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

[6] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan, editors, *EMNLP/IJCNLP (1)*, pages 3980–3990. Association for Computational Linguistics, 2019.

[7] Matthew L. Henderson, Rami Al-Rfou, Brian Strope, Yun-Hsuan Sung, László Lukács, Ruiqi Guo, Sanjiv Kumar, Balint Miklos, and Ray Kurzweil. Efficient natural language response suggestion for smart reply. *CoRR*, abs/1705.00652, 2017.

[8] Tianyu Gao, Xingcheng Yao, and Danqi Chen. Simcse: Simple contrastive learning of sentence embeddings. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2021.

[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: ACL 2022, pages 2663–2669*, 2022.

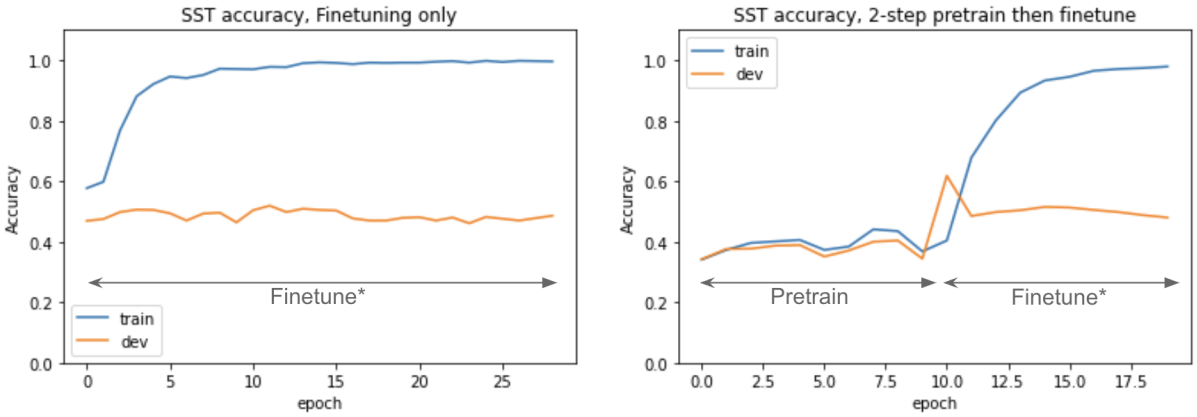
[10] Yige Xu, Xipeng Qiu, Ligao Zhou, and Xuanjing Huang. Improving BERT fine-tuning via self-ensemble and self-distillation. *CoRR*, abs/2002.10345, 2020.

[11] Jingjing Xu, Xu Sun, Zhiyuan Zhang, Guangxiang Zhao, and Junyang Lin. Understanding and improving layer normalization. In *Advances in Neural Information Processing Systems, 32*, 2019.

[12] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of NAACL-HLT 2019, pages 4171–4186*, 2019.

[13] Eneko Agirre, Daniel Cer, Mona Diab, Aitor Gonzalez-Agirre, 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*, pages 32–43, Atlanta, Georgia, USA, June 2013. Association for Computational Linguistics.

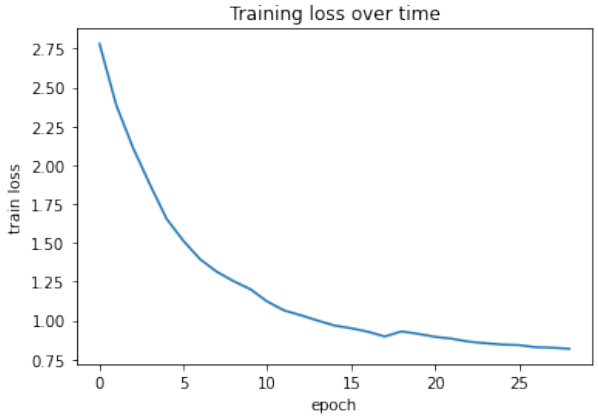
A Appendix, Comparing Direct Finetuning vs. 2-Step Pretraining before Finetuning performance on the SST task

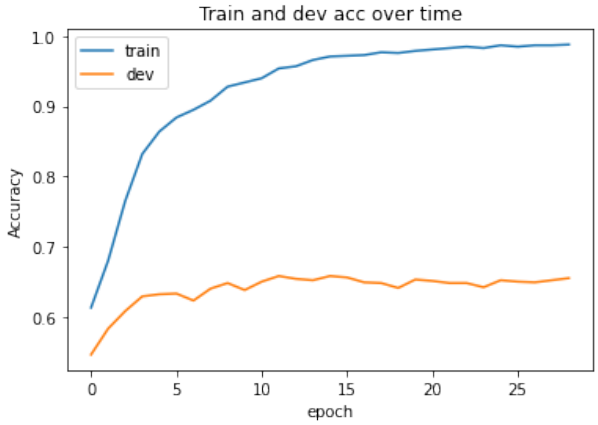


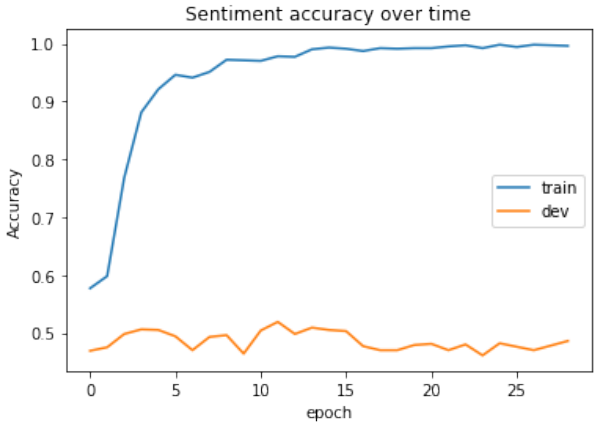
Divergence of train and dev performance likely indicates that finetune learning rate was too high for both scenarios, but the benefit pretraining is still observed on the right chart as dev performance and stability exceed that on the left chart.

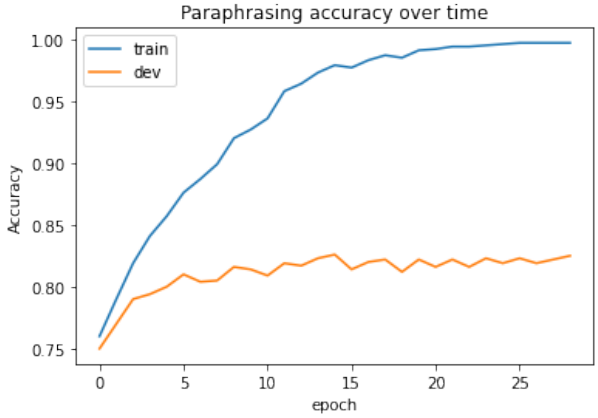
B Appendix, Model Performance over Epochs

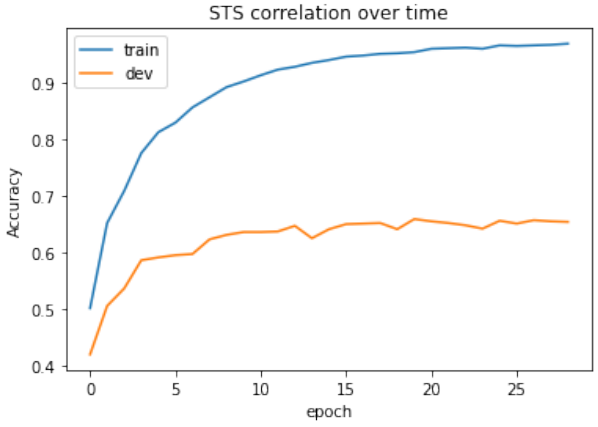
B.1 Final model with 30 Epochs



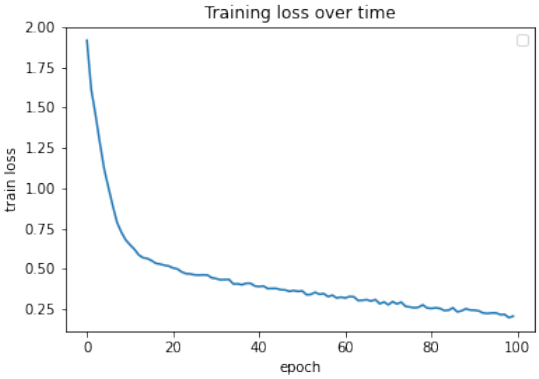


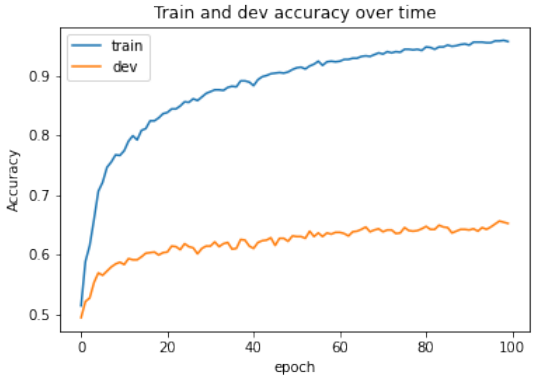


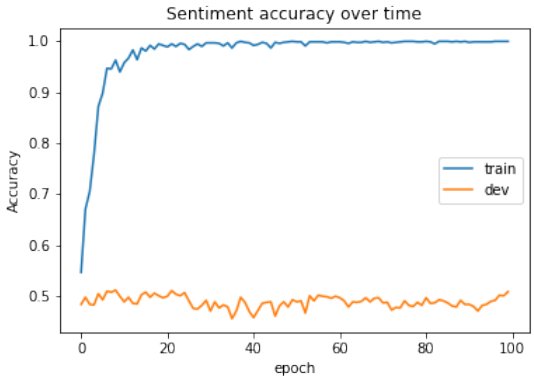


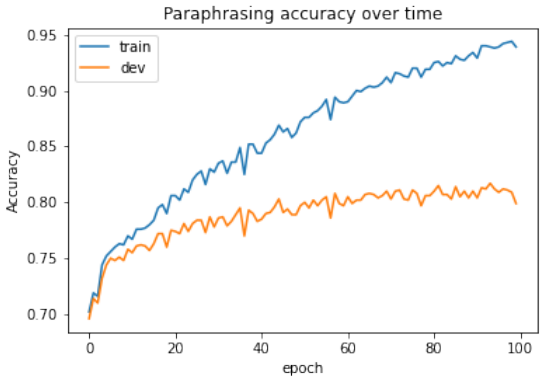


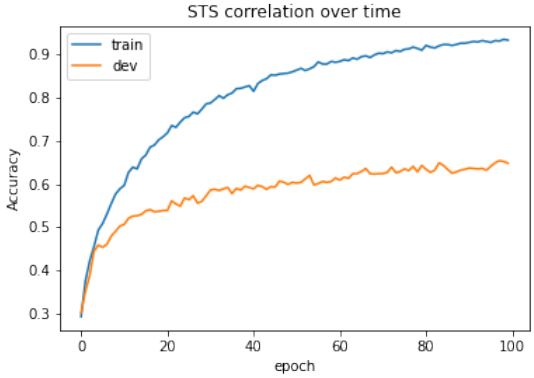
B.2 Base Model with 100 Epochs





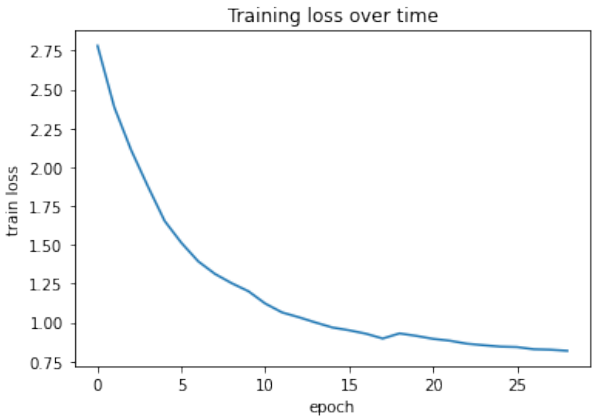


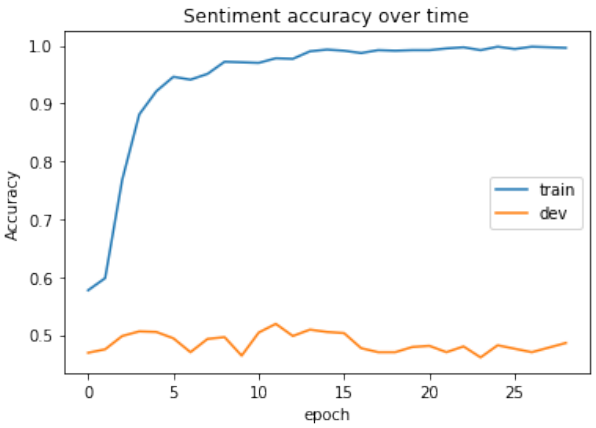




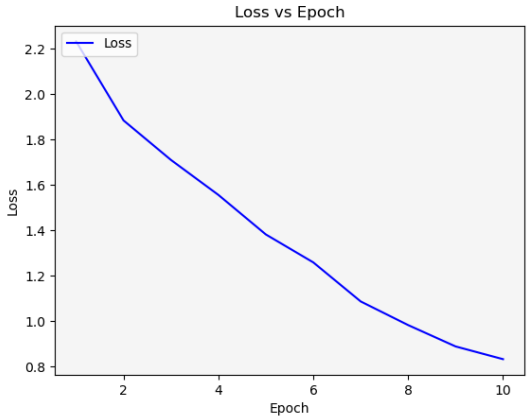
C ‘Final’ model vs Milestone model performance

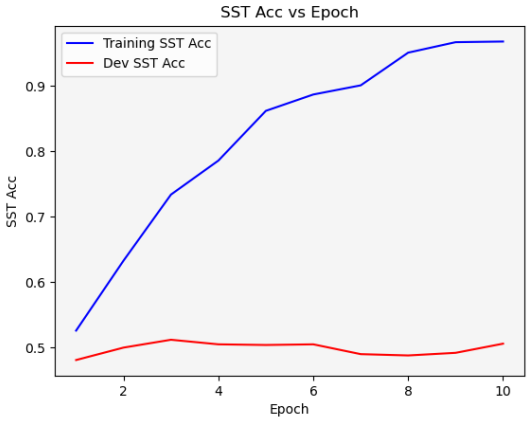
Final Model

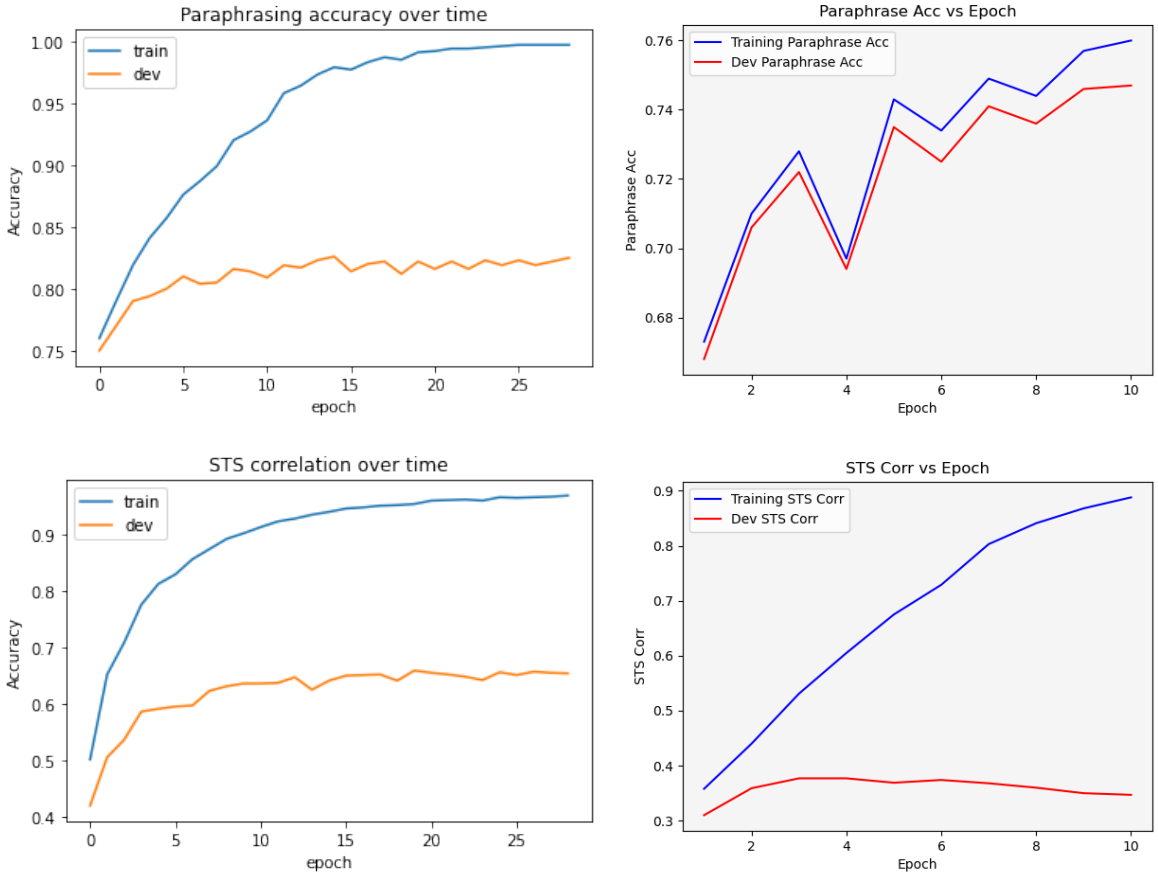




Milestone model







D Appendix, STS "failure" examples from qualitative analysis

Example sentence pairs with predicted similarity much below that of the golden set

Sentence 1	Sentence 2
One boy is pushing another boy on a swing.	A boy is pushing another boy on the swings.
A girl is talking to her dad on a cellphone.	a girl is talking on her phone.
A woman and man are riding in a car.	A woman driving a car is talking to the man seated beside her.
A man is slicing a cucumber.	A man is cutting cucumbers.

Example sentence pairs with predicted similarity much above that of the golden set

Sentence 1
The American Stock Exchange biotech index .BTK surged 5 percent.
Use of force in defense of person.
Italian Prime Minister Mario Monti Resigns
the plant will be constructed next to an existing plant and is projected to be operational in 7 years.

Sentence 2
The Philadelphia Stock Exchange's semiconductor index .SOXX jumped 6.10 percent.
Use of force by aggressor.
Palestinian Prime Minister Fayyad resigns
the plant will be constructed next to an existing plant also supplied by china and operational since 1999.

E Appendix, SST example sentences with golden set prediction and model predictions

Example sentences which involve double negatives or other complex forms of structure that convey the opposite ideas of what a short phrase might imply.

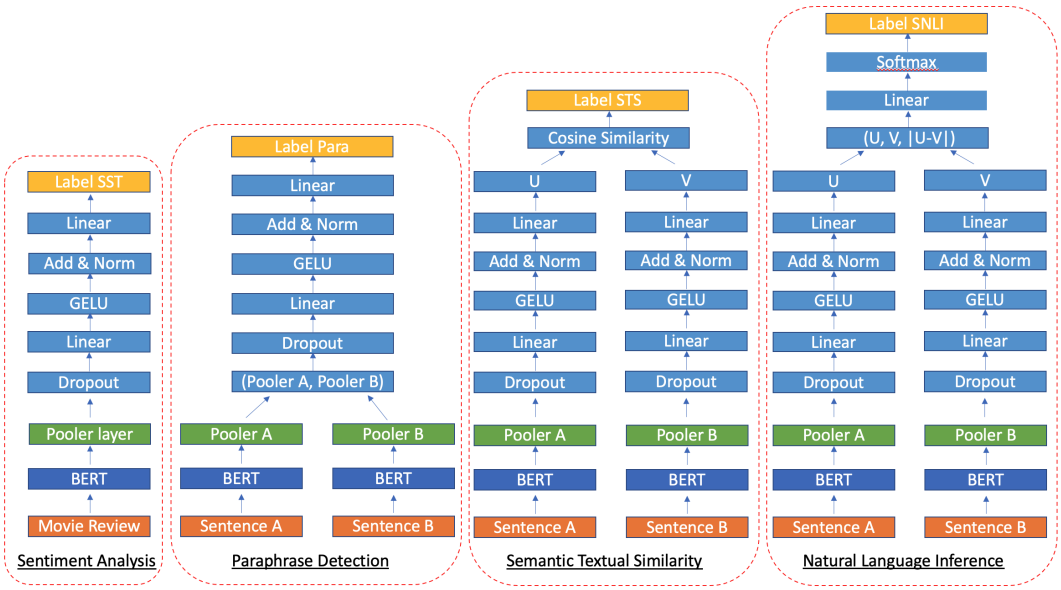
- The lower your expectations , the more you 'll enjoy it .
- The experience of going to a film festival is a rewarding one ; the experiencing of sampling one through this movie is not .
- This is not the undisputed worst boxing movie ever , but it 's certainly not a champion - the big loser is the audience .
- Reign of Fire looks as if it was made without much thought – and is best watched that way .
- Another one of those estrogen overdose movies like " Divine Secrets of the Ya Ya Sisterhood , " except that the writing , acting and character development are a lot better .
- If Steven Soderbergh 's ' Solaris ' is a failure it is a glorious failure .
- Hilariously inept and ridiculous .

Example sentences in which we disagreed with the golden set sentiment assessment.

- It 's a coming-of-age story we 've all seen bits of in other films – but it 's rarely been told with such affecting grace and cultural specificity. (Golden set sentiment: 2, model predicted sentiment: 4)
- A gripping , searing portrait of a lost soul trying to find her way through life . (Golden set sentiment: 2, model predicted sentiment: 4)
- Every dance becomes about seduction , where backstabbing and betrayals are celebrated , and sex is currency . (Golden set sentiment: 1, model predicted sentiment: 3)

F Model Architecture

Figure 1: Final model architecture.



G Paraphrasing "failure" examples

(1) Pairs of questions with strong n-gram overlap but different meaning (especially different question word) are often misclassified as paraphrase of each other.

Sentence 1
What are the best and profitable ways for saving money?
What are the most inspiring/motivational books about life?
Is the USA the only country that has GPS satellites?
Is there any evolutionary advantage of baldness?
Is there any chance of medal for India in Rio Olympics 2016?

Sentence 2
What are your best ways to save money?
What is the most powerful tip you've gained from reading a self-help book?
How many countries have their own GPS system?
Was there any evolutionary advantage for beards?
How many medals is India expected to win at the 2016 Rio Olympics?

(2) Model failed to recognize different vocabulary with similar meaning in-context.

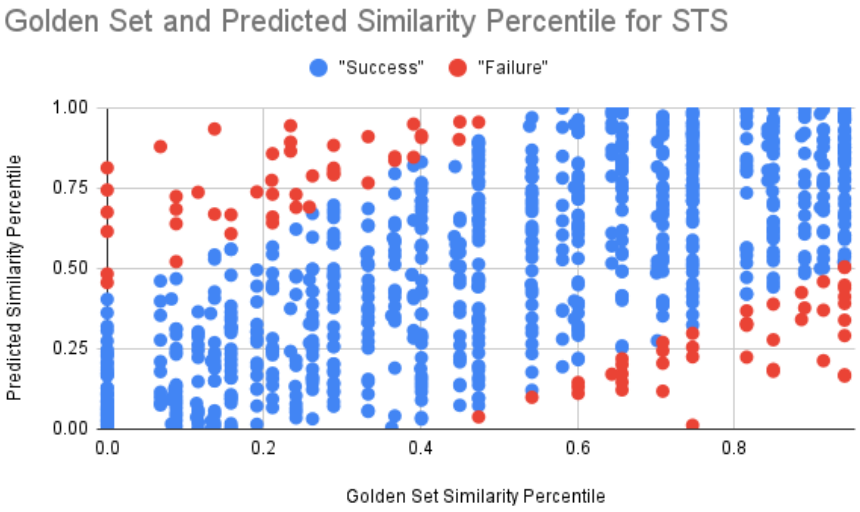
Sentence 1
Why are Facebook, Google, and others not allowed in China?
Which Mobile CRM software do you think is best for my business?
How do I convince my parents to let me meet an online friend?
What are some interesting campus recruitment rejection stories?
How do I stop the low frequency sound?

Sentence 2
Why are Google, Facebook, YouTube and other social networking sites banned in China?
Which CRM software is best for small business?
How do I convince my parents to let me meet online friends?
What are some of the best rejection stories at campus recruitment?
What is a way to block low frequency sound?

H Distribution of predicted sentiment versus golden set for SST task

Sentiment	Count of examples in golden set	Count of examples in prediction	Percent of examples in golden set	Percent of examples in predictions
0	139	98	12.6%	8.9%
1	289	334	26.2%	30.3%
2	229	169	20.8%	15.3%
3	279	370	25.3%	33.6%
4	165	130	15.0%	11.8%

I I, STS prediction vs. golden set evaluation



Golden Set Similiarity range	Num of data points	Failure Count	Failure Percent
[0,1]	165	16	10%
(1,2]	120	18	15%
(2,3]	180	12	7%
(3,4]	238	18	8%
(4,5]	160	25	16%

J Confusion matrix for Paraphrasing task

Table 9: Confusion matrix for the paraphrasing tasks

		Golden set classification	
		positive	negative
predicted classification	positive	25%	11%
	negative	12%	52%