Enhancing BERT for Advanced Language Understanding: A Multitask Learning Approach with Task-Specific Tuning

Stanford CS224N Default Project

Anusha Kuppahally, Malavi Ravindran, Ziyue (Julia) Wang
Department of Statistics
Stanford University
akupp@stanford.edu, mr328@stanford.edu, wangzyue@stanford.edu

Abstract

This paper presents an implementation of the BERT model for the tasks of sentiment analysis, paraphrase detection, and semantic textual similarity (STS), focusing on the transformer encoder block with multi-head self attention. Experimentation revealed that hyperparameter tuning, cosine similarity loss for STS, incorporation of additional task-specific data, multitask learning, and annealed sampling boosted performance on unseen data. The best performing model achieved a sentiment classification accuracy of 0.501, paraphrase detection accuracy of 0.781, and STS correlation of 0.546 on the test data. Future work includes more thorough hyperparameter tuning and the inclusion of additional data to improve results for the STS task.

1 Introduction

Natural Language Processing (NLP) tasks involve understanding and analyzing human language using computational methods. Among these tasks, sentiment analysis, paraphrase detection, and semantic textual similarity (STS) play pivotal roles in extracting meaningful insights from textual data. Sentiment analysis is the task of determining the sentiment expressed in text, and can be particularly challenging due to the nuanced nature of emotion. Paraphrase detection requires models to identify whether two sentences convey the same meaning, despite differences in wording or structure, and is crucial for applications such as plagiarism detection and QA systems. STS aims to quantify the degree of relatedness between two texts, and can be used in information retrieval and text summarization.

While recent advancements in transformer-based models, such as BERT, have shown promising results across various NLP tasks, there is still room for improvement, particularly in task-specific contexts. Existing methods often rely on generic pre-trained models, limiting their effectiveness in real-world applications. Furthermore, it may be desirable and more efficient to train models capable of handling multiple tasks at once. Our research built upon the BERT model architecture by leveraging multitask learning and incorporating additional task-specific data for sentiment analysis, paraphrase detection, and STS.

2 Related Work

In "How to Fine-Tune BERT for Text Classification?", Sun et al. (2019) tested the following fine-tuning methods for BERT text classification models: altering fine-tuning strategies (choosing which layer is best for the target task, the optimization algorithm, and the learning rate), further pre-training (within-task, in-domain, and cross-domain), and multitask fine-tuning. The tasks included sentiment analysis, question classification, topic classification, and data-preprocessing. This paper tried multiple fine-tuning methods and achieved improved results on 8 datasets. It was found that

generally, further pre-training performed better than no pre-training. In-domain pre-training brought better performance compared to within-task pre-training, but both methods improve performance. Also, it was determined that a preceding multitask fine-tuning could be helpful for single-task fine-tuning, but improves performance to a lesser degree compared to further pre-training. However, this paper has a limitation, as when implementing task-specific pre-training for sentiment classification, this paper used data with significantly different domains, which led to a lack of improvement in this area. Our work attempts to rectify this limitation by incorporating additional task-specific data that is similar in distribution to the original datasets used for training.

"BERT and PALs: Projected Attention Layers for Efficient Adaptation in Multitask Learning" described how multitask learning can be used to share information and parameters among many related NLP tasks (Stickland and Murray, 2019). The paper explored hard-parameter sharing, in which tasks share hidden layers while retaining specific output layers. The authors described that one of the most effective approaches to augment the base BERT model with multitask capabilities was to simply add parameters to the "top" of the model prior to classification. Our application adopted this approach by adding a handful of task-specific layers to the top of base BERT. The authors also noted several strategies for training with disparately sized multitask data. The most straightforward approach, round-robin sampling, selects a batch of training examples from each task and cycles through them in a fixed order. However, one drawback to this standard approach is that by the time we exhaust the examples from one task, we may have already cycled through a smaller task's training data several times over, leading to overfitting. To combat this, the authors introduced a novel strategy, annealed sampling, which samples tasks proportionally to their representation in the training data with a more equal representation of tasks towards the end of training. The paper found annealed sampling to perform best on a variety of GLUE tasks. We decided to implement both of these sampling approaches in order to tackle the disproportionately sized task data, with the hypothesis that annealed sampling would yield the most desirable results.

3 Approach

Our approach implemented three key aspects of the original BERT model: 1) the baseline BERT class, including the multi-head self-attention mechanism, 2) the pipelines for performing sentiment analysis, paraphrase detection, and STS and 3) the Adam Optimizer for efficient stochastic optimization.

Baseline BERT: The baseline BERT architecture consists of an embedding layer followed by 12 transformer blocks. We first used WordPiece to tokenize and pad sentences to length 512. The [CLS] token was also placed at the beginning of each sentence to facilitate the various classification tasks. Following this was the embedding layer, where input embeddings of dimension 768 were formed for each token id by summing token, segmentation, and positional embeddings. Next, we included 12 encoder blocks, each beginning with a multi-head self-attention layer. This layer applied the scaled dot product to 12 distinct attention heads to obtain the attention weights. Letting Q, K, and V denote matrices of queries, keys, and values respectively, we have:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$$

To realize multi-head attention, we concatenated the multi-heads to recover the original shape Vaswani et al. (2017). The rest of the BERT encoder block simply added the transformed attention output (with dropout) to the hidden states, normalized the result, moved this output through a linear layer with GELU activation, added the transformed (linear) feed forward output (with dropout) to the intermediate output, and normalized again. After 12 transformer blocks, we concluded with a linear transformation for the [CLS] token, and this output was passed through the tanh activation function.

Baseline BERT training: We explored two options for training our models. One method was to use the pretrained BERT model, where most parameters in the model were frozen (i.e. no gradient updates) and only parameters of the task-specific layers were updated. In contrast, fine-tuning initialized the model with weights from pretrained BERT but allowed all model parameters to be updated. Within the training loop, we used the base BERT model to obtain the embedding corresponding to the hidden state of the [CLS] token. Depending on the downstream task, we employed different strategies for handling the [CLS] token.

Sentiment Classification: For the sentiment classification task, we passed the [CLS] token through a dropout layer, and passed the result through a linear layer to obtain logits representing the sentiment

classes. We computed the cross-entropy loss between the predicted logits and the ground truth sentiment labels, and then performed backpropagation to update the model parameters using the Adam optimizer, exact details of which can be found in the handout ProjectHandout (2024).

Paraphrase Detection: As paraphrase detection handles pairs of sentences, we obtained the [CLS] embedding associated with each input sentence in the pair. We applied dropout to each of these embeddings, concatenated the outputs, and passed this concatenated tensor through a linear layer to obtain a single logit for predicting whether or not the two sentences were paraphrased. We passed this logit through the sigmoid function to obtain probabilities and then computed binary cross entropy loss with the ground truth label. We performed backpropagation using the Adam optimizer.

STS: The STS task also handles pairs of sentences. For this reason, we again obtained the [CLS] embedding associated with each input sentence in the pair and applied dropout to each embedding. We then explored two different options. In the first option, we concatenated the two dropout-transformed embeddings and passed the result through a linear layer to produce a single logit representing how similar the sentences were. We then passed this logit through the sigmoid function and rescaled the resulting value to be between 0 and 5. For the second option, we computed the cosine similarity between the dropout-transformed embeddings, passed the resulting tensor through a ReLU layer to incorporate non-linearity, and again rescaled the result to be between 0 and 5. We explored this option because the research of Reimers and Gurevych (2019) showed that cosine similarity is more suited for the STS task. We then computed the mean-squared error loss between the predicted and ground truth similarities, and performed backpropagation using the Adam optimizer.

Sequential vs. multitask Learning: Another consideration was that of sequential vs. multitask learning. In sequential training, we processed all of the batches of one task before moving on to the next task. In contrast, the standard (round robin) multitask approach involved processing batches of the three tasks simultaneously before taking an update step. In addition to the round robin approach, we also explored a second flavor of multitask learning through annealed sampling. Here, instead of processing one batch of each task within each iteration, we selected a batch of examples from each task with probability p_i such that $p_i \propto N_i^{\alpha}$, with α initialized to 1. We updated α after each epoch such that $\alpha = 1 - 0.8\frac{e-1}{E-1}$, where e is the current epoch and E is the total number of epochs. This ensured task sampling probabilities became more equal as training proceeded (Stickland and Murray, 2019).

The prediction pipeline for each of these tasks utilized the exact same process as training, where final predictions were made based on the highest probability class (sentiment classification, paraphrase detection) or the rescaled output (STS).

4 Experiments

4.1 Data

We utilized the three default datasets provided by CS 224N. For sentiment analysis, we used the Stanford Sentiment Treebank (SST) dataset, consisting of 11,855 single sentences taken from movie reviews, labeled with their sentiment polarity (0 = negative, somewhat negative, neutral, somewhat positive, or 4 = positive) (Socher et al., 2013). For paraphrase detection, we used the Quora dataset [1], consisting of 400,000 pairs of questions labeled with whether they are paraphrases of each other. For STS, we used the SemEval STS Benchmark dataset, consisting of 8,628 sentence pairs labeled with a similarity score from 0 to 5 (Agirre et al., 2013). All three datasets were split into train (70%), dev (10%), and test (20%) sets (ProjectHandout, 2024).

To deal with the imbalanced size across the default datasets, we secured two additional datasets for the underrepresented tasks. For sentiment analysis, we used a subset of the Amazon Products Review Dataset crawled from Amazon in 2014 by UCSD (Ni et al., 2019). In particular, this subset contains 982,619 Kindle book reviews where each textual review is accompanied by a rating from 1 to 5. This rating could be interpreted as a sentiment score of the review where 5 corresponds to positive and 1 corresponds to negative. To match the provided SST dataset, we subtracted 1 from all ratings so that the new ratings had range 0 to 4. When sampling from this Amazon dataset, we made sure to sample with equal balance from each of the possible rating classes (Appendix A Figure 5).

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

For STS analysis, we supplemented with the Sentences Involving Compositional Knowledge dataset for Relatedness and Entailment (SICK). The SICK dataset was generated by Marelli et al. (2014) from the 8K ImageFlickr data set and the SemEval 2012 STS MSR-Video Description data set. It contains about 9,840 pairs of sentences, each labeled with a sentence relatedness score (from 1 to 5) annotated through crowdsourcing (Appendix A Figure 6).

Using the five datasets described above, we produced three versions of data to conduct our experiments. First, we trimmed the default training sets to the size of the smallest training set (i.e., the STS training set): 6,040 rows. When sampling down, we made sure to maintain an equal balance amongst each class as the data allowed. This version of the data was referred to as "trimmed". Then, we padded the STS data with the SICK data, resulting in a dataframe with 15,880 rows. We trimmed the default training sets for sentiment analysis and paraphrase detection to 15,880. This version of the data was referred to as "padded". The third version was "full", consisting of the 15,880 rows of STS + SICK data, all 141,506 rows of the Quora data, and the SST data padded up with the Amazon data to 141,506 rows (Appendix B Table 6). We were unfortunately unable to find more data for STS. We considered using bootstrap to upsample existing data, but replicating the data to 10 times its current size would likely deteriorate our training performance. All modifications only pertained to training data.

4.2 Evaluation method

For evaluating sentiment classification and paraphrase detection, we used *accuracy*: the proportion of labels correctly predicted. For evaluating STS, we used *correlation*: the Pearson's correlation score between the predicted similarity scores and the true similarity scores.

4.3 Experimental details

We set the default learning rate for the Adam Optimizer to be 1E-03 with $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon =$ 1E-06, and an initial weight of decay being 0. After tuning $p_{dropout}$ and the fine-tuning learning rate on the complete SST dataset, we proceeded with $p_{dropout}$ = 0.5 and a learning rate of 1E-05 for all subsequent experimentation. After testing on the entire STS data, we validated that adding cosine similarity improved STS correlation. With the padded data, we tested sequential and round-robin multitask learning, and found that round-robin performed better. We also pursued annealed sampling to further test multitask learning, and found that it outperformed the standard round-robin. Our training process spanned 10 epochs and took approximately 1 hour on the trimmed data, 2 hours on the padded data, and 10 hours on the full data using modern V100 GPU. We evaluated the performance of the models using the dev sets for all 3 tasks. For reproducibility, we used seed 11711 for everything.

4.4 Results

Our first experiment was tuning the dropout rate. For computational efficiency, we only used the SST dataset for dropout tuning and extrapolated to apply results to the other tasks in our model (Table 1). We achieved the highest accuracy when fine-tuning on the SST dataset with $p_{dropout} = 0.5$.

From these results, we also confirmed that fine-tuning had better performance than pre-training, and thus only fine-tuned our subsequent models (Appendix C Figure 7).

After experimenting with various fine-tuning learning rates, we validated that the default rate of 1E-5 was the best learning rate to use (Table 2). We continued to use this rate for the rest of our models.

Next, using the STS data, we found that cosine similarity improved the performance of the STS task and used this method for our future models. Initially, we aimed to use results from the trimmed data to see if sequential learning or round-robin multitask learning performed better, but we noticed that there was large variability in results, for instance, when implementing cosine similarity for the same learning method (Table 3). Thus, we decided to test on the padded data to determine which learning method we should pursue and concluded that round-robin multitask learning should be used (Table 4). Then, we experimented with annealed sampling to further explore other methods of multitask learning.

Working with the full data, we compared performance of annealed sampling against round-robin multitask learning and found that annealed sampling out-performed round-robin (Table 5). For

Dropout Probability	Dev Accuracy for Pretraining	Dev Accuracy for fine-tuning
0.1	0.418	0.519
0.2	0.408	0.526
0.25	0.402	0.521
0.3 (default)	0.399	0.520
0.4	0.394	0.526
0.5	0.401	0.528

Table 1: Results from Dropout Probability Tuning on SST

Fine-tuning Learning Rate	Dev Accuracy
1E-4	0.262
1E-5 (default)	0.520
5E-6	0.516
1E-6	0.514

Table 2: Results from Fine-tuning Learning Rate Tuning on SST

Model	Sentiment Accuracy	Paraphrase Accuracy	STS Correlation
Sequential learning	0.514	0.701	0.361
Sequential learning, cosine similarity	0.480	0.590	0.435
Round-robin	0.484	0.685	0.332
Round-robin, cosine similarity	0.497	0.658	0.361

Table 3: Results from Cosine Similarity

Model	Sentiment Accuracy	Paraphrase Accuracy	STS Correlation
Sequential	0.466	0.662	0.502
Round-Robin	0.491	0.699	0.475

Table 4: Results for Sequential vs. Round-Robin Multitask Learning

annealed sampling, we observed steady improvement across epochs and confirmed that for the last epoch, the weights were equal across tasks as intended (Figure 1).

We achieved the following test leaderboard results using annealed sampling: 0.501 for sentiment accuracy, 0.781 for paraphrase accuracy, and 0.546 for STS correlation, with an overall score of 0.685. Based on our results, it was not surprising that a higher $p_{dropout}$, implementing cosine similarity for STS, adding task-specific data, and utilizing multitask learning (both round-robin and annealed

Model	Sentiment Accuracy	Paraphrase Accuracy	STS Correlation
Round-Robin	0.463	0.747	0.436
Annealed Sampling	0.499	0.776	0.562

Table 5: Results for Round-Robin vs. Annealed Sampling Multitask Learning

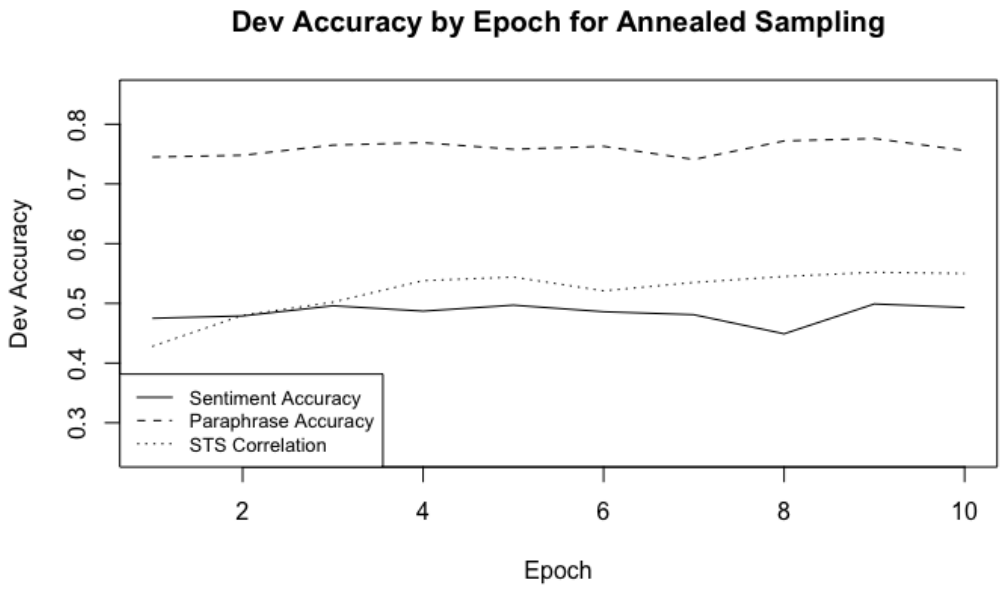

Figure 1: Dev Accuracy by Epoch for Annealed Sampling

sampling) improved performance, as these techniques were recommended in related work. It was unexpected to see round-robin multitask learning perform worse for sentiment accuracy compared to previous models on the full data, as additional data for the sentiment task should have improved performance and prevented overfitting. It also was expected that the model would overfit on the STS data, as this dataset was the smallest in comparison, but this performance was not significantly worse compared to smaller models. It was surprising that only paraphrase accuracy improved.

5 Analysis

Ablation Study:

As our process demonstrates, we conducted an ablation study in order to dissect and analyze different elements of the model and training process to understand their contributions to performance metrics across sentiment classification, paraphrase detection, and STS tasks. The first component of our study was to understand the impact of the $p_{dropout}$, where we kept the dataset (SST), learning rate, training strategy (pretrain vs. finetune) and task fixed and varied $p_{dropout}$ according to these values: 0.1, 0.2, 0.25, 0.4, 0.5. As seen in Table 1, we achieved the highest accuracy for fine-tuning when $p_{dropout} = 0.5$. The next component of our ablation study was to alter the fine-tune learning rate, holding the dataset (SST), training strategy (fine-tune), and $p_{dropout}$ fixed. After experimenting with learning rates of 1E-4, 5E-6, and 1E-6, the best performance was achieved using the default rate of 1E-5 (Table 2). We next experimented with cosine similarity in the STS task, and found that when holding dataset (STS), $p_{dropout}$, learning rate, and training strategy (fine-tune) fixed, our dev correlation was higher when incorporating cosine similarity into the loss calculation (Table

3). Next, we wanted to examine the efficacy of sequential vs. multitask (round-robin) training, and discovered that multitask learning achieved improved overall performance after holding dataset (padded), $p_{dropout}$, and learning rate fixed (Table 4). The final step in our ablation study was comparing basic round-robin with annealed sampling, where we held dataset (full), $p_{dropout}$, and learning rate fixed, and achieved our best performance when using annealed sampling (Table 5).

Error Analysis:

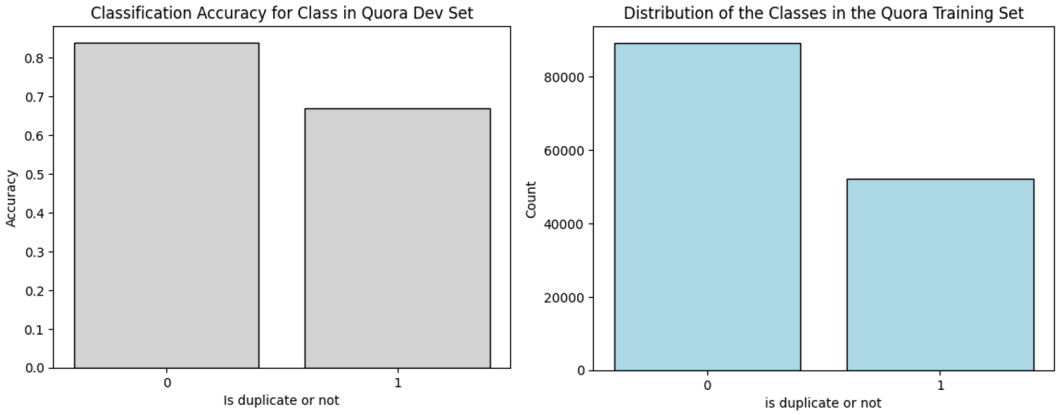

Figure 2: Error Analysis for Paraphrase Detection

Our model performed better in recognizing class 0 (i.e., non-duplicates) in paraphrase detection, which aligns with the fact that the number of examples in class 0 is approximately 1.7 times greater than that in class 1 in the training data. To increase the accuracy of predicting for class 1, we should augment class 1 in the training data through either upsampling or acquiring more data.

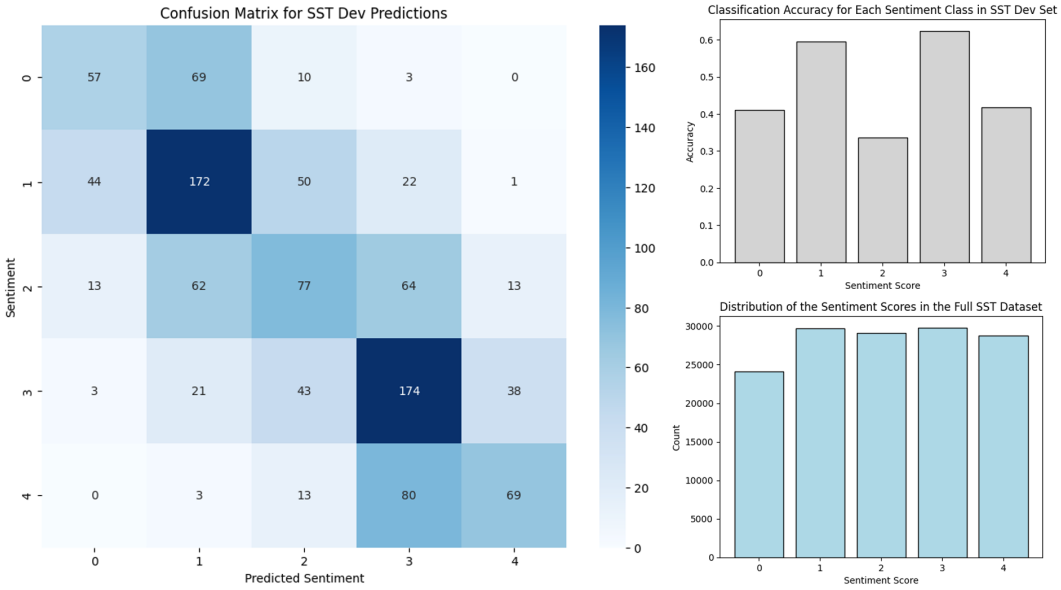

Figure 3: Error Analysis for Sentiment Analysis

For sentiment analysis, our model had the highest accuracy in recognizing class 1 and 3 (somewhat negative and somewhat positive) and the worst in predicting class 2 (neutral). This performance cannot be attributed to imbalance in the training data, as the representation of all five classes are quite equal. Class 2 was the hardest to recognize because a neutral sentiment is inherently more subjective or harder to define clearly compared to more polarized sentiments. For example, one review labeled "neutral" in the SST dev set is "a beautifully made piece of unwatchable drivel". Even for humans, it is hard to tell whether this comment is toasting the movie using sarcasm or saying that the content

of the movie is bad but the scenes are beautiful. It can only be harder for the model to discern the sentiment embedded in this review which has both positive words like "beautifully" and negative words like "unwatchable" and "drivel" and potentially means the opposite of its literal sense. We were surprised to see that our model struggled to predict the two extreme cases (0 and 4) as they often exhibit clear, unambiguous sentiment expressions. However, as is evident from the confusion matrix, predictions for class 0 and 4 were split between them and their adjacent classes (1 and 3). Hence, it is reasonable to speculate that class separability between negative versus somewhat negative and positive versus somewhat positive is difficult for the model to determine (Figure 3).

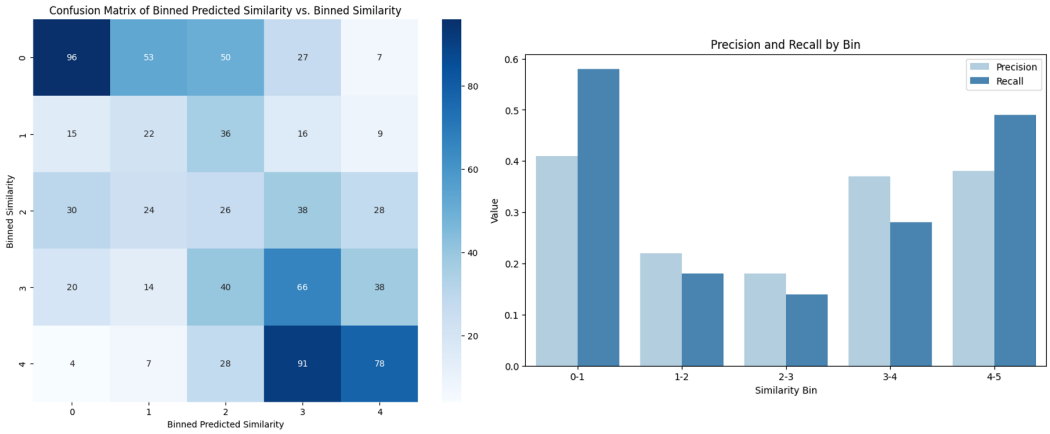

Figure 4: Error Analysis for Textual Similarity Detection

For the STS task, we binned the similarity scores into five buckets to better analyze exactly how our model is erroring out. We found that our model performed better (precision and recall-wise) for larger and smaller similarity scores (Figure 4). This makes sense as the more extreme the cases are, the more distinctive they tend to be. The neutral cases can suffer from lexical ambiguity, class separability issues, and potential mis-labelling during crowdsourcing.

6 Conclusion

Overall, our project identified that the following extensions to the BERT model improved performance: hyperparameter tuning (dropout rate, fine-tuning learning rate), cosine similarity for the STS task, additional task-specific data, multitask learning, and annealed sampling. Over the course of the project, we learned how to implement each extension and understand how to build upon models iteratively to obtain the best results. One limitation of our work may be the addition of the Amazon dataset for sentiment analysis. While product reviews may be used as proxies for sentiment, these may not align exactly with the original data and could contain inconsistencies. Future work may include further hyperparameter tuning and the inclusion of additional data for STS, as this may improve results for multitask learning and prevent overfitting when training on uneven data.

7 Contributions

Coding, debugging, and report writing was split quite evenly for proposal, milestone, and final report. Specializations for each team member were: Anusha performed the GCP setup and was responsible for running all of the experiments and creating plots and tables found in the report, Malavi wrote the code for round-robin and annealed sampling, and Julia found and cleaned additional datasets, implemented cosine similarity, and performed error analysis. Our mentor is Andrew Lee.

References

Eneko Agirre, Daniel Cer, Mona Diab, Aitor Gonzalez-Agirre, and Weiwei Guo. 2013. sem 2013 shared task: Semantic textual similarity. *Second joint conference on lexical and computational*

*semantics (*SEM), volume 1: proceedings of the Main conference and the shared task: semantic textual similarity*, page 32–43.

M. Marelli, S. Menini, M. Baroni, L. Bentivogli, R. Bernardi, and R. Zamparelli. 2014. A sick cure for the evaluation of compositional distributional semantic models. *Proceedings of the 2014 International Conference on Language Resources and Evaluation*.

Jianmo Ni, Jiacheng Li, and Julian McAuley. 2019. Justifying recommendations using distantly-labeled reviews and fined-grained aspects. *Empirical Methods in Natural Language Processing (EMNLP)*.

Stanford Univeristy CS 224N ProjectHandout. 2024. Cs 224n default final project: minbert and downstream tasks. Stanford University.

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

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642.

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

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

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *In Advances in Neural Information Processing Systems*, pages 5998–6008.

Appendix A EDA on Additional Datasets

Amazon Review Data Note that the Amazon Review Data (Ni et al., 2019) is highly imbalanced (see Figure 5). Nonetheless, the imbalance between classes does not affect our model because we are only using a small subset of this data which does not require the overflowing portion of the larger classes. To be exact, we used 1,467 rows of the dataset with sentiment score 0, 1, 2, and 3 respectively, as well as 1,468 rows of the dataset with sentiment score 4. These numbers were selected so that the resulting "full" sentiment analysis dataset would have exactly 141,506 rows, matching the size of the full Quora dataset.

SICK Data Note that the distribution of the similarity scores of the SICK data (Marelli et al., 2014) is slightly different from that of the training set of STS: the former has more pairs with similarity scores above 3 and relatively fewer pairs with similarity scores between 1.5 and 2.5 whereas the latter has a relatively uniform distribution (see Figure 6). This would result in a rather unbalanced representation of examples with different similarity scores and could potentially cause affect our model. We attempted to find more datasets on textual similarity detection with similarity scores as result but without much success. This would be one direction of possible future work.

Appendix B Three versions of Data

Please find the number of samples in each version of the data below in Table 6.

Appendix C Tuning Dropout Probability

Please find the accuracy by dropout rate plot below (Figure 7).

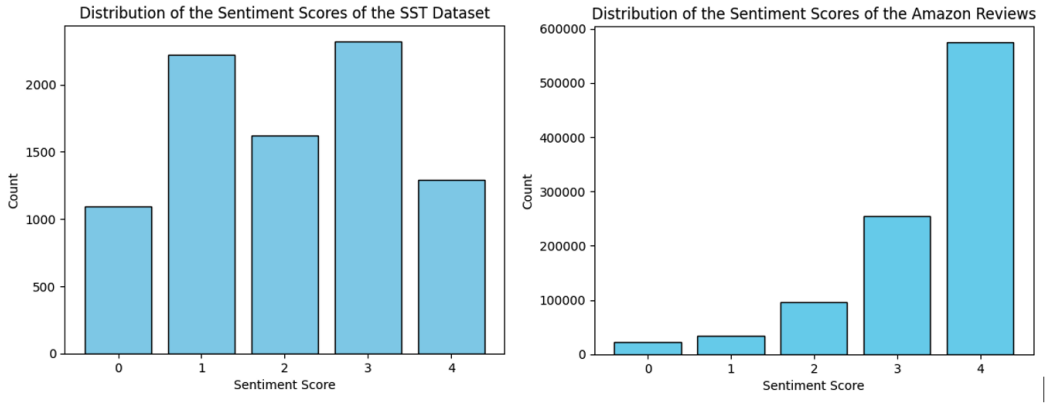

Figure 5: Distribution of the sentiment scores of the SST training set (left) and the full Kindle review dataset before sampling (right)

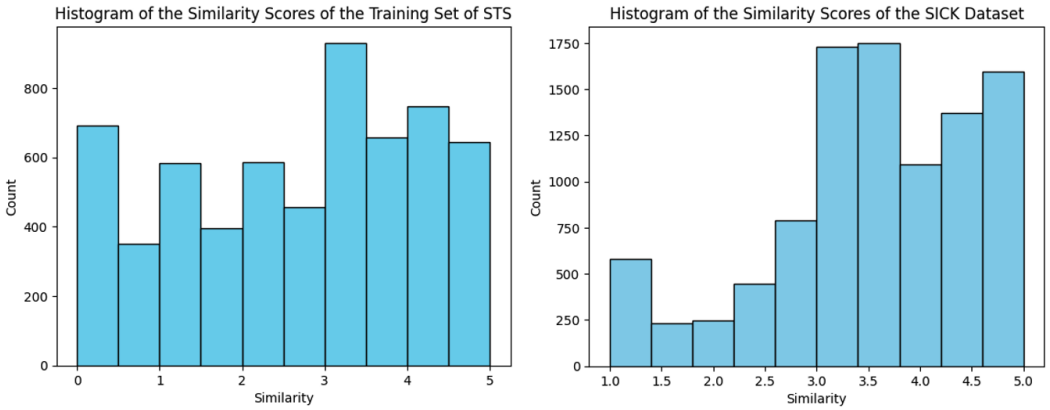

Figure 6: Distribution of the similarity scores of the STS training set (left) and the SICK dataset (right)

Data Version	Sentiment Analysis	Paraphrase Detection	Textual Similarity Detection
Trimmed	6,040	6,040	6,040
Padded	15,880	15,880	15,880
Full	141,506	141,506	15,880

Table 6: Size of the Three Versions of Datasets Used

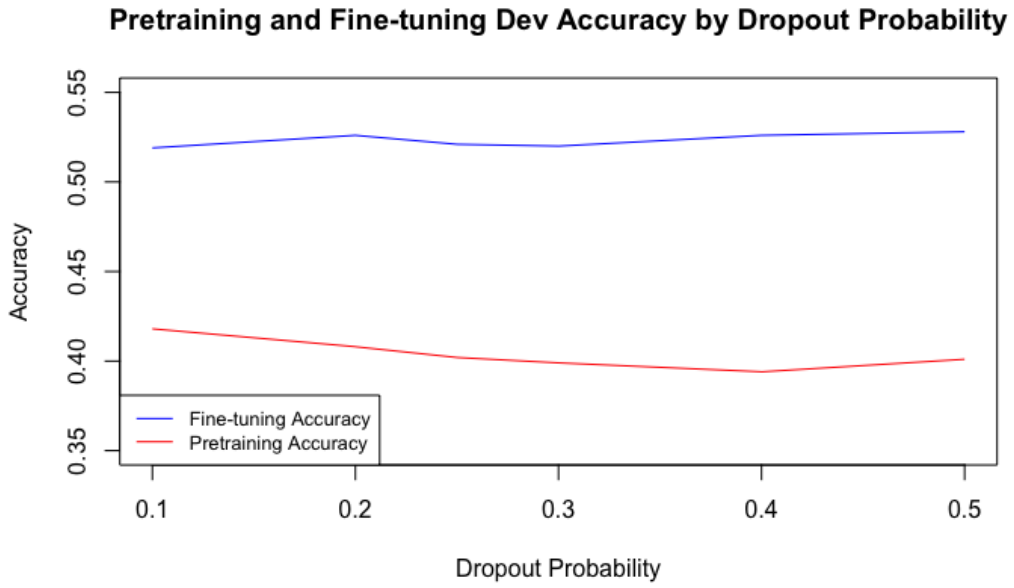

Figure 7: Accuracy by Dropout Rate