BERT's Got Talent: Advanced Fine-Tuning Strategies for Better BERT Generalization

Stanford CS224N Default Project

Grace Luo
Department of Computer Science
Stanford University
luograce@stanford.edu

Danny Lin
Department of Computer Science
Stanford University
dannylin@stanford.edu

Abstract

Transfer learning has transformed NLP by allowing a singular language model pre-trained on extensive text to extend to multiple downstream NLP tasks. In this paper, we explore various strategies to enhance the BERT model's performance across three specific sentence-level tasks: sentiment analysis, paraphrase detection, and semantic textual similarity. We implemented numerous strategies, and found that a combination of single-task models, cross-encoding, mean pooling, and SMART loss regularization in addition to considerations of dataset quality produced a strong model. We achieve an accuracy of 0.791 on the test set as a result of our extensive exploration.

1 Key Information to include

- Mentor: Johnny Chang
- External Collaborators (if you have any): None
- Sharing project: No
- Contribution: We both contributed equal and substantial efforts to the design choices, implementations, and writeup.

2 Introduction

Natural Language Processing (NLP) has achieved remarkable success with the advent of Transformer-based architecture models, such as the large pre-trained BERT model Devlin et al. (2018), by enabling robust and generalizable language understanding. There are various strategies for fine-tuning these models to different downstream tasks, for instance single tasks and multitasking. But, they struggle from overfitting and task interference respectively, hindering the model's ability to generalize to diverse tasks.

Our paper aims to explore and compare various strategies to determine which yield the best performance for sentiment analysis, paraphrase detection, and semantic textual similarity tasks, experimenting with single-task and multitask. We ultimately propose three single-task models that leverage cross-encoding with token type embeddings to improve learning for semantic tasks, SMART regularization to combat overfitting, input preprocessing to counter data inconsistencies, and mean pooling for meaningful sentence embeddings to produce a robust, high-performing model. We also experiment with variations of a multitask model to compare performance. We achieve significant improvements over baseline models, with the final model achieving a test accuracy of 0.791.

3 Related Work

The success of BERT in Devlin et al. (2018) and potential for generalizations to downstream tasks sparked much additional research.

In the SentenceBERT paper, Reimers and Gurevych (2019) discuss mean embedding pooling as an alternative to the default CLS pooling strategy. It was adopted to create more comprehensive and semantically meaningful sentence embeddings since CLS embeddings yieled poor performance for semantic tasks. The paper also discusses the use of bi-encoding with cosine similarity to evaluate semantic similarity of embeddings. Bi-encoding entails inputting individual sentences through BERT to derive separate embeddings.

This is in contrast with the cross-encoding strategy of the original BERT (Devlin et al. (2018)). Cross-encoding involves concatenating pairs with a `[SEP]` token before inputting through BERT, thus producing one embedding. The original BERT model was also trained with token-type embeddings, which distinguish sentence order. Although used for next-sentece prediction, these embeddings could also reinforce the `[SEP]` token's role of distinguishing the two sentences in addition to leveraging BERT's pre-trained ability.

Additionally, overfitting is a common issue when fine-tuning large pre-trained models for smaller tasks. Jiang et al. (2019) proposes Smoothness-Inducing Adversarial Regularization, which encourages robustness against small perturbations in the input. This is achieved by incorporating an adversarial regularizer that penalizes the model if its predictions vary significantly from slight input changes.

Finally, Wei and Zou (2019) propose random deletion as a form of data augmentation to improve classification task performance, deleting each word of a sentence with a variable probability p. It yields comparable results to other similar augmentation techniques (as in Kobayashi (2018) and Hu et al. (2017)) without requiring additional datasets.

4 Approach

As a baseline, we implement the basic, multitask BERT model with task-specific linear classifiers for each task. We implemented different loss functions for each of the three downstream tasks. Sentiment analysis (SST) uses cross-entropy loss, paraphrase detection (Para) uses binary cross-entropy loss, and semantic textual similarity (STS) uses mean squared error loss. All the base BERT parameters frozen and only the task-specific classification and regression heads are fine-tuned.

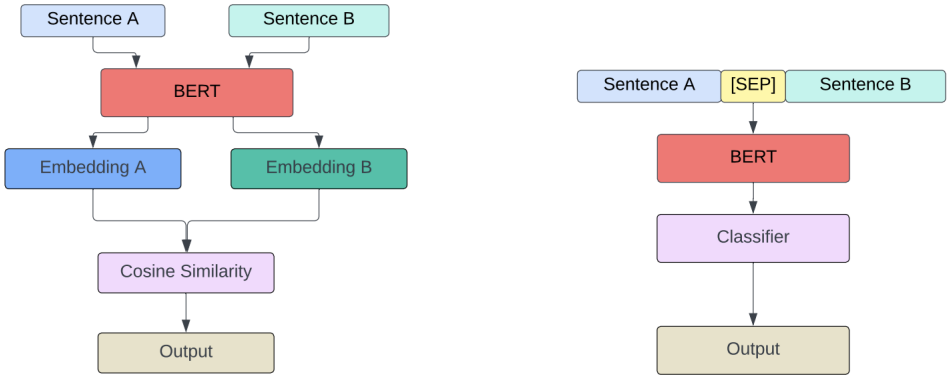
We then built upon this with two additional baselines. One model included with fine-tuning of BERT parameters along with to task-specific layers. This served as the upper bound for multitask BERT from solely fine-tuning. The last baseline consisted of three single-task models with full-model fine-tuning. This served as the upper bound of single-task accuracy from fine-tuning.

Mean Embeddings. Upon evaluating baselines, we noticed acceptably higher scores for SST and Para (compared to the leaderboard), but very low STS scores. We observed that most of the STS predictions were within the $4.0 - 4.9$ range, which indicated that the embeddings were all very similar. So, we proposed using mean token embeddings instead of the baseline '[CLS]' token embeddings to capture more meaningful, representative sentence embeddings, as demonstrated in Reimers and Gurevych (2019). The mean embeddings are calculated by taking the average of all the token embeddings in a sentence. It therefore might be able to provide a more comprehensive representation of the sequence because it incorporates information from all parts.

CosineEmbeddingLoss. Further inspired by Reimers and Gurevych (2019), we also implemented CosineEmbeddingLoss to further boost STS scores. It calculates the cosine similarity between two mean embedding vectors $\mathbf{u}$ and $\mathbf{v}$ as $\cos(\theta) = \frac{\mathbf{u} \cdot \mathbf{v}}{||\mathbf{u}|| ||\mathbf{v}||}$, returning a value in the range $[-1, 1]$. -1 indicates absolutely opposite vectors, 0 no correlation, and 1 absolutely similar. CosineEmbeddingLoss aims to maximize the similarity for semantically similar pairs and minimize for the semantically dissimilar. To then scale the final score to the desired $[0, 5]$ range, we used a sigmoid function and then multiplied the result by 5. Intuitively, optimizing cosine similarity suits STS because semantically alike sentences should have embeddings with greater cosine similarity. Because the task's goal is to evaluate the semantic likeness of sentences, appropriately maximizing and minimizing embedding similarity ensures the model can better capture the necessary semantic relationships between sentences.

Input preprocessing, Data Augmentation. After noticing considerably lower SST scores for most groups, we decided to look at the training data to see if there were flaws causing issues across the board. We noticed variations of certain characters and contractions that lead to inconsistencies in BERT's tokenizing process. The characters "-LRB-" and "-RRB-" represent "(" and ")", respectively. Apostrophes have spaces before them (the dataset has "has n't", "reader 's" instead of "hasn't", "reader's"), leading to different tokens from the tokenizer. The quotation marks are also inconsistent, with some represented with a normal quotation character (") and others as two singular quotations (‘‘), leading to different interpretations by BERT. Therefore, we pre-processed the data so that ["-LRB-", "-RRB-"] become ["(", ")"], [" '"] becomes ["'"], ["‘‘"] becomes ["""].

We also noticed largely unequal representations of sentiment classes in the data (A.4). So, we augmented the data by increasing the number of sentences in each of the classes to match the number for the most-represented label (label 3, 2322 sentences). This was done using random deletion, as mentioned in Wei and Zou (2019). Each word in a sentence is randomly removed with probability 0.1 (as per the paper), and the modified sentence is then added to the dataset with the same label. As stated in Wei and Zou (2019), one of its tasks was sentiment analysis, and augmenting the original sentences largely maintained the same labels of the originals.



(a) Bi-Encoding (Cosine Similarity)

(b) Cross-Encoding

Figure 1: Bi-encoding vs. Cross-encoding

Cross Encoding and Token Type Embeddings. To further address the Para and STS scores, we investigated cross-encoding (Reimers and Gurevych (2019)). In our baseline, paraphrase used bi-encoding (Figure 1a), which put each sentence of a sentence pair through BERT to get individual sentence embeddings. The embeddings for the pair were then concatenated and passed through a linear layer to produce one logit, which was compared with the label using binary cross-entropy loss. For STS, the model did not concatenate the individual sentence embeddings, but instead calculated their cosine similarity for CosineEmbeddingLoss.

Cross-encoding (Figure 1b) was introduced as part of the original BERT model in Devlin et al. (2018). As opposed to bi-encoding, two individual sentences of a sentence pair are concatenated with a '[SEP]' token in between. Then, the concatenated result is forwarded through the BERT model and linear layer to produce one value prediction. For paraphrase, BCELoss is still applied to the labels, while L1 Loss was used for STS instead. In passing concatenated sentences through BERT, the model must process a pair at the same time, potentially pushing it to learn more complex, fine-grained semantic relationships between the sentences. Further, the model could develop a stronger understanding of contextual relevance, as it could better recognize certain words or phrases in one sentence that have specific meanings only when viewed in the context of the other sentence.

We also integrated token-type embeddings (segment embeddings in Devlin et al. (2018)) to further assist cross-encoding. Token type embeddings (TTE) are vectors added to the input embeddings to indicate which segment or sentence each token belongs to. Cross-encoding distinguishes two sentences with the '[SEP]' token, and TTE would solidify the model's differentiation. Further, BERT was pre-trained for next-sentence prediction (NSP), which used TTE. Although Para and STS are not

the same as NSP, understanding the relationship between two sentences is crucial for all three tasks. BERT's pretraining could therefore leverage its learned ability to process and compare sentence pairs with TTE's explicit segment information.

Hyperparameter Tuning. We observed overfitting, mainly in the SST and STS datasets, so we tested dropout rates to see which was optimal for each task. We tried dropout rates of 0.1, 0.2, and 0.3 on the cross-encoded, single-task SST and STS models. We also experimented with batch sizes of 8, 16, and 64 to find a balance between performance and speed.

SMART Regularization. We investigated SMART regularization (Jiang et al. (2019)), specifically integrating smoothness-inducing adversarial regularization, as an alternative to combat overfitting in SST and STS.[1] SMART regularization operates by introducing small perturbations to the input embeddings and penalizing the model if its output changes significantly in response to these perturbations. This method encourages the model to be smoother and more robust, improving its generalization to unseen data. The optimization objective is defined as:

$$\min_{\theta} \mathcal{F}(\theta) = \mathcal{L}(\theta) + \lambda_s \mathcal{R}_s(\theta),$$

where $\mathcal{L}(\theta)$ represents the task-specific loss function, λ_S is the tuning parameter, and $\mathcal{R}_s(\theta)$ is the smoothness-inducing adversarial regularizer. The regularizer $\mathcal{R}_s(\theta)$ is given by:

$$\mathcal{R}_s(\theta) = \frac{1}{n} \sum_{i=1}^{n} \max_{\|\tilde{x}_i - x_i\|_p \leq \epsilon} \ell_s(f(\tilde{x}_i; \theta), f(x_i; \theta)),$$

where $\tilde{x}_i$ denotes the perturbed embedding, generated by adding small noise to the original embedding x_i. Minimizing $\mathcal{F}(\theta)$ encourages the model to have stable predictions by demonstrating robustness to small perturbations to x_i, thus potentially mitigating overfitting. The loss functions ℓ_s were selected based on the task (as per Jiang et al. (2019)), with symmetrized KL-divergence used for classification tasks (SST) and mean-squared error for regression tasks (STS). For Para, we did not implement regularization because it is expensive and overfitting was not pronounced for the task. Jiang et al. (2019) discusses Bregman Proximal Point Optimization, but our implementation opted for the default ADAMW Optimizer. Dropout negatively impacts SMART loss as per Hayes (2023), so we decided to remove it to best gauge SMART's performance effect.

Multitask Learning Techniques. Given that Para and STS have similar tasks (learning and comparing semantic sentence meanings) and both use cross-encoding, we decided to explore if multitasking on Para and STS would boost STS because it would train on more data. So, we would have two total models instead of 3: one single-task SST model, and one multitask model for Para and STS. Initially, we trained a multitask model that iterated through the Para and STS dataset each once per epoch. We realized that Para and STS however, had a significantly disproportionate amount of data and the model would therefore underfit to STS.

So, we implemented a cycling approach, where for each epoch, it would train on the entire Para dataset once per epoch and cycle through STS dataset to match the amount of Para data. This strategy seeks to leverage the entirety of the Para dataset, but cycles through STS just under 50 times per epoch when trained on all 280K examples, likely leading to overfitting.

We therefore experimented with another way to combat the issue of disproportionate data *without* cycling and its overfitting issue: re-weighting the losses. Our objective changed from minimizing `[para-loss + sts-loss]` to minimizing `[para-loss + w · sts-loss]`, where w was the proportion of the Para dataset to the STS dataset size. This strategy seeks to ensures that the larger dataset does not dominate the training process by increasing the influence of the smaller dataset. We effectively aim to effectively normalize the contribution of each dataset to the overall training objective in an attempt to mitigate STS underfitting. We could still fully utilize the vast Para dataset without excesssively exposing the model to nearly 50 times as much cycled STS data.

5 Experiments

5.1 Data

For sentiment analysis, use the Stanford Sentiment Treebank (SST) dataset of extracted movie review sentences, each with one of five labels: negative, somewhat negative, neutral, somewhat positive,

[1] https://github.com/archinetai/smart-pytorch

and positive (labeled on $[0, 4]$ integer scale). It is split into $8,544$ train, $1,101$ dev, and $2,210$ test examples. For paraphrase detection, we used the Quora dataset of question pairs, labeled with a 0 or 1 to indicating whether sentences are paraphrases of one another. It is split into $283,010$ train, $40,429$ dev, and $80,859$ test examples. For semantic textual similarity, we use the SemEval STS Benchmark Dataset of sentence pairs labeled with a similarity score on a scale of $[0, 5]$. It is split into $6,040$ train, 863 dev, and $1,725$ test examples.

5.2 Evaluation method

We evaluate the model's performance with three task-specific metrics: accuracy between predicted and true labels for SST and Para, and the Pearson correlation coeeficient for STS, which is the linear correlation between the predicted and true similarity scores. The overall performance is measured by averaging the scores.

5.3 Experimental details

Our hyperparameters for all experiments are as follows, unless otherwise specified: 10 epochs, learning rate of 2e-5, dropout of 0.2, batch size of 16. We optimize the model with the ADAMW algorithm. For SMART, we use the recommended hyperparameters in Jiang et al. (2019): $T_{\tilde{x}} = 1$, $\sigma = 1 \times 10^{-5}$, $\epsilon = 1 \times 10^{-6}$, $\eta = 1 \times 10^{-3}$, and we experimented with λ_s values. Given the large size of the paraphrase dataset, we opted to initially train on a smaller subset of the data of 8000 paraphrase sentence pairs unless otherwise specified. This is done for the sake of speed when testing different strategies. For the final model, we train on the full 280K pairs.

5.4 Results

Baseline. We first experimented with the three baseline variations and found the 3 single-task models to have the best performance (A.1). This matches our expectation, as each model has more focused optimization.

Pooling Techniques. Afterwards, we committed to training three separate models and compared the performance of the baseline CLS pooling mean embeddings (ME) pooling for all tasks (results in Table 1).

Pooling	SST	Para	STS	Dev Acc
CLS	0.519	0.765	0.381	0.555
Mean	**0.520**	0.765	**0.390**	**0.558**

Table 1: CLS vs. Mean Embeddings

While the score for SST improved minimally (0.001) and Para remained unchanged, the STS score experienced a much larger increase, suggesting that mean embeddings were more representative of the varying meanings of input sequences. Since STS and Para focus on encapsulating meaningful semantic embeddings, we implemented mean embeddings for both tasks. The improvement in SST was not as substantial and it could be due to random seed, but since ME did not noticeably worsen performance and it performed much better on STS, we decided to use ME for all three tasks.

Input Pre-processing and Data Augmentation. The next experiment focused on SST, and the results for input pre-processing and data augmentation are shown in Table 3.

Model	SST
No preprocessing/data aug	0.516
w/ Input pre-processing	**0.518**
w/ Data Augmentation	0.513

Table 2: Input pre-processing and Data Aug

We observe a slight enhancement from input pre-processing, but a worse performance with data augmentation. While we were expecting a more substantial increase from pre-processing, the result indicates that standardizing the inputs had some benefit. The decrease in augmentation could be because the data was not suitable for random deletion, possibly because sentences are shorter. We noticed that the data augmentation reached 0.513 at first epoch, then dropped to $0.470 - 0.480$ range

every epoch after, while input pre-processing exhibited more consistent growth. This suggests that the model possibly overfit to the augmented sentences, thus reducing performance. As such, we chose to keep input pre-processing and leave data augmentation.

CosineEmbeddingLoss, Cross-Encoding. We then tested the effectiveness of switching from MSELoss to CosineEmbeddingLoss on STS. Similarly, we also wanted to see how cosine similarity would yield compared with the cross encoding method, so the results were combined into Table 3.

Model	Para	STS	Dev Acc
Bi-Encoding (MSE)	–	0.390	–
Bi-Encoding (Cosine)	–	0.725	–
Cross Encoding	0.869	0.868	0.754
Cross Encoding + Token Type	**0.878**	**0.880**	**0.761**

Table 3: STS Loss and Encoding Techniques

We tried 4 approaches: bi-encoding with MSELoss for STS, bi-encoding with CosineEmbeddingLoss for STS, cross encoding (now with L1 loss for STS), and cross encoding with TTE. For all the cases, BCELoss was used for Para. As shown, cosine similarity significantly improved STS performance to a more acceptable value. However, cross encoding substantially improved both STS *and* Para scores, as expected (or even more than expected) because the model could better learn the relationships between the two sequences. TTE, although more minimal growth, still consistently improved both scores. Consequently, we chose to implement cross-encoding with TTE for our final model.

Model	SST	Para	STS	Dev Acc	Time (Min:Sec)
Batch size 8	0.518	0.869	**0.884**	0.757	15:56
Batch Size 16	**0.525**	**0.878**	0.880	**0.761**	12:43
Batch Size 64	0.520	0.868	0.872	0.753	**10:19**

Table 4: Batch Size

Batch Size. Experimenting with batch sizes, we found that 16 produced the best STS, and the medium size produced the best SST, Para, and overall accuracy by a considerable amount, a pleasant surprise because it balanced generalization and speed. Larger batch sizes tended to run faster (a 2-3 minute difference between trials), and we expected it to also reduce performance due to less noise and gradient estimates with lower variance. Considering that we only train on 8000/280K Para pairs during experimentation, the difference in time will be exacerbated when training on the full dataset. Therefore, we opted to maintain a batch size of 16.

Dropout and Regularization. Noticing overfitting, we first experimented with dropout values (A.2). 0.3 dropout resulted in a 0.006 jump in SST to produce a highest score of 0.531 observed thus far. However, STS experienced similar small increases of 0.004 and 0.003 for 0.1 and 0.3, respectively, resulting in unclear conclusions. Given that dropout had a more significant impact on SST and inconclusive results on STS, we changed dropout to 0.3 for SST and maintained 0.2 dropout for STS (also to strike some balance between 0.1 and 0.3).

Table 5 reveals the effect of SMART regularization, a more effective technique, where we experimented to find the best λ_s and analyzed SST, STS, and average iterations per second.

λ_s	SST	STS	Max Dev Epoch	Avg It/s
0	**0.531**	0.884	9	47.1
0.05	0.520	0.895	9	23.6
0.1	0.512	0.893	6	23.5
5	0.528	0.896	**5**	23.2
15	0.518	**0.900**	6	22.8

Table 5: Experimenting SMART tuning paramenter values

As shown, $\lambda_s = 15$ yields the best result for STS seen thus far, with a 0.016 increase. However, any λ_s does not yield as high of a SST score as the model without SMART. We believe SMART produced much higher STS because the small perturbations to input data allows the model to become more sensitive to subtle semantic differences, while dropout does not provide the same targeted improvement for capturing subtle interactions between words. For SST, the model might have overfit to specific words that are generally indicative of sentiment but have complex nuances, so dropout

could encourage it to learn sentiment more holistically. We observe $\lambda_s = 5$ converged the quickest, reaching its maximum dev accuracy at epoch 5 and the second highest STS score. The halving of it/s with SMART indicates high computational overhead, as expected from generating adversarial examples and computing the additional loss. However, $\lambda_s = 15$ produced an STS score 0.900, which was significantly higher than any previous observations and 0.004 higher than the next highest observed STS score with SMART. Further, it peaked at epoch 6, an improvement compared to the model without SMART, which peaked at epoch 9. Thus, we decided to adopt $\lambda_s = 15$ for STS, as it balanced both high scoring and computational efficiency.

Multitask. Despite already high Para and STS scores, we were curious if there was a more efficient way to also achieve high scores via multitasking. There are four models: original single-task models, multitasking without cycling (training on each dataset once per epoch), multitasking with cycling (per epoch, train Para once and train STS until reach same amount of examples seen), multitasking without cycling and re-weighted loss. We tested on 24000 pairs for the cycling models to better gauge the extent to which overfitting, an anticpated effect due to disproportionate data, occurs.

Model	Para	STS	Dev Acc
No Multitask-Partial	0.878	**0.900**	0.889
Multitask w/o cycling-Partial	0.886	0.892	0.889
Multitask w/ cycling-Partial	**0.895**	0.896	**0.896**
Multitask w/o cycling, re-weighted-Partial	0.809	0.887	0.848
Multitask w/ cycling-Full-Dev	0.911	0.892	0.902
Multitask w/ cycling-Full-Test	0.902	0.726	0.814

Table 6: Multitask Models

Observing that multitasking with cycling produced the best results, we were wary of the overfitting and decided to train on the whole dataset for further examination. For the dev set, the model exhibited similar scores, with an expected higher Para due to training on all data. STS, however was much lower (-0.170) score compared to the cycling experiment on partial data. This indicates that the model struggles to generalize to STS tasks and that the test set could contain more challenging and diverse examples. No-cycling multitask models also performed worse or comparably on STS, and we believe the difference in datasets would be exacerbated with the full Para data, so we did not further consider any multitask model.

Final Results.

Model	SST	Para	STS	Acc
Baseline (Dev)	0.310	0.647	0.113	0.357
Final Model (Dev)	0.510	0.907	0.899	0.789
Final Model (Test)	0.523	0.904	0.892	**0.791**

Table 7: Baseline vs. Final Model

Our final model consists of 3 single-task models, ME pooling for all, cross-encoding for Para and STS, SMART regularization with $\lambda_s = 15$ for STS, input pre-processing for SST, dropout 0.3 for SST and 0.2 for Para, and batch size of 16 for all. We report a test accuracy of 0.791. [2]

6 Analysis

Sentiment Analysis. We plotted a confusion matrix (A.3) to visualize the model's accuracy. The relatively high percentages along the diagonal indicate that the model is able to successfully classify sentiment the majority of the time and rarely deviates more than 1 sentiment class. However, a considerable number of milder predictions are actually in more extreme adjacent classes; 39% of true "negative"s are predicted "slightly negative" and 38% of true "positive"s as "slightly positive". This could be due to the imbalance in data shown in (A.4), with examples labeled 1 and 3 occupying 53.14% of training data and examples labeled 0 and 4 occupying 27.85% of data. Furthermore, the model struggles with neutral labels and identifies them correctly 35% of the time, likely due to the intrinsically complex nature of neutrality. We also noticed that many errors arose from focus on literal words rather than sentiment. For instance, we and the true label classify "If Steven Soderbergh 's '

[2]The test model was trained on both the training and dev set.

Solaris ' is a failure it is a glorious failure" as positive, but the model identified "neutral". We believe this is because the model incorrectly assumes "glorious" (pos) and "failure" (neg) offset each other (as many training examples do, see Ex. 7), failing to recognize nuanced sentiment. We found many training labels were flawed as well; we classify "Like shave ice without the topping, this cinematic snow cone is as innocuous as it is flavorless" as negative, but it is labeled neutral.

Paraphrase Detection. We plotted a confusion matrix (A.5) and again saw high accuracies for the diagonal. We noticed the model tends to incorrectly identify paraphrases as distinct (13%) more than distinct sentences as paraphrases (7.5%). Upon further investigation (A.6), we notice false negatives with sentences that require deeper understanding of word meanings and contexts as opposed to basic word matching. For instance, in Ex. 1, the model fails to realize that "sunblock" and "sunscreen lotions" are equivalent, and that the addition of "during summer" does not affect overall semantic meaning because sunblock is typically applied during summer. This requires a much deeper awareness of context and word meanings that could extend beyond the limitations of our model, which likely needs more data to achieve such an understanding. Incorrect predictions overall tend to occur with sentences that vary by a word or two (Ex. 2, 3, 4). In Ex. 4 for example, the words used are identical (with the exception of "biochemistry" added to sentence 2), but have opposite meaning due to the swapping of "vitro" and "non-vitro", which the model failed to recognize. This is indicative of the model's challenge in discerning deeper meaning of similar pairs.

Semantic Textual Similarity. We visualized the model with a scatter plot (A.7), which demonstrates strong correlation. Upon analyzing the points with lowest accuracy (A.6), we notice trends of incorrectly low similarity predictions due to varying words with synonymous meaning (Ex. 5) and incorrectly high similarity predictions due to the same word used in different contexts (Ex. 6). In Ex. 6, "work", although used as a verb in both sentences, means "to put in effort" in one sentence and "to function successfully" in the other, a nuance the model does not distinguish. This indicates the model's limitation lies in its reliance of literal word usage, as opposed to deeper semantic understanding of contexts and synonyms.

Model Analysis. We plot the lengths of the SST dataset sentences (A.8) and observe they are skewed towards shorter values, mostly in the range of 11-30 words. This could explain why data augmentation via random deletion was not effective for SST, because deleting word(s) in shorter sentences is more likely to remove crucial information. We observe that the addition of TTE to cross-encoding yielded improvements in Para and STS by a considerable 0.009 and 0.012, despite TTE having no explicitly new contribution to the encoding and embedding process. This supports our hypothesis that TTE possibly enhances existing cross-encoding features, improving ability to discern subtle differences or similarities between pairs. Multitasking with re-weighted losses revealed a large drop in scores, especially for Para (unsurprising because less weight was given to it). However, it is the fact that STS also dropped which suggests re-weighting is insufficient for such large dataset size discrepancies. Another observation is the relatively drastic fluctuations in SST scores, despite no changes to the model. When we implemented CosineEmbeddingLoss for STS, the SST score dropped from 0.520 to 0.516. After implementing cross-encoding, which did not affect SST, the values jumped from 0.518 to 0.525. This could be due to random seed, and also the imbalanced and faulty data as mentioned above, which could impact consistency. We believe this explains the drop in SST scores for the final dev performance.

7 Conclusion

Our paper extensively explores a wide range of extensions for improving the BERT model. We implement and evaluate pooling techniques, cross-encoding, SMART loss, creative multitasking strategies. Ultimately, a combination of single-task training, differing optimal hyperparameters, encoding, and regularization arising from comprehensive testing emerged as a powerful model. It highlights how different tasks can require varying techniques, and the importance of experimenting new strategies with clear intent from prior observations. We experience limitations with regards to data, as our models trained on flawed SST data and focused on sentences in Quora and movie reviews without generalizing to broader human language contexts. In the future, we would explore additional STS datasets to address the crux of the issue with multitask cycling. Additionally, investigating more advanced data augmentation techniques, more efficient techniques (given SMART overhead), ensemble methods, and the integration of external knowledge bases could enhance model performance and robustness.

8 Ethics Statement

Our project investigation and analysis focused on overfitting and data quality. We attempted to mitigate overfitting, which yielded positive and negative results for different tasks, and addressed flaws and imbalance in the SST dataset. However, we did not implement regularization or higher dropout for the Para task because we did not observe overfitting, nor did we notice many obvious mislabelings. Upon further reflection of the Quora dataset, we realize that the model could overfit to certain biases present in Quora and learned relationships that may have yielded high performance on the test set, but are harmful in other broader language contexts. Our models are single-task, so they have greater tendencies to overfit. For instance, while there is no official data on the user demographic, we noticed many questions that would be asked by the younger generation (questions about older experiences, prepping for exams, puberty, etc.). We also noticed many US and India-specific questions, indicating a high percentage of users are from those countries. Training well to Quora could yield solid performance for this project, but the model could be learning implicit ageist or culturally harmful biases. To mitigate these, we propose policymakers implement a benchmark for diversity in training data. Models trained on solely one type of data with the intent of going on to more general tasks should not be deployed unless they meet a certain benchmark of training on various datasets that are comprehensive of the type of classification/regression tasks it will perform.

Another ethical issue is the computation overhead. We use single-task models, which are more computationally expensive than multitask, in addition to implementing SMART on one of the model. This could lead to unsustainable practices (higher emissions, infrastructure costs, etc.) due to the resource-intensiveness, should the model be scaled incredibly larger. While we were conscious of the cost, we prioritized performance for this project and had more than enough compute power. Our strategy is not wasteful, though, as we do consider offsetting with batch size and only implementing SMART for one model (when absolutely necessary). However, in cases where the compute power needed infringes on the environment, we believe maintaining full GPU utilization to reduce the usage of resources by unutilized GPUs. Moreover, there should be policy enforcing ESG (environment, social and governance) benchmarks for companies using the model. We also propose that more resources be devoted to creating central dashboards that can efficiently track metrics as companies scale the model to ensure they are meeting ESG goals.

References

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

Matthew Hayes. 2023. Serbertus: A smart three-headed bert ensemble. In *Stanford CS224N Default Project*.

Zhiting Hu, Zichao Yang, Xiaodan Liang, Ruslan Salakhutdinov, and Eric P. Xing. 2017. Controllable text generation. *CoRR*, abs/1703.00955.

Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Tuo Zhao. 2019. SMART: robust and efficient fine-tuning for pre-trained natural language models through principled regularized optimization. *CoRR*, abs/1911.03437.

Sosuke Kobayashi. 2018. Contextual augmentation: Data augmentation by words with paradigmatic relations. *CoRR*, abs/1805.06201.

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

Jason W. Wei and Kai Zou. 2019. EDA: easy data augmentation techniques for boosting performance on text classification tasks. *CoRR*, abs/1901.11196.

A Appendix

A.1 Baseline Table

Model	SST	Para	STS	Dev Acc
Baseline-Multitask-Last Linear Layer	0.310	0.647	0.113	0.357
Baseline-Multitask-Full Model	0.490	0.762	0.361	0.537
Baseline-Single Task-Full Model	0.520	0.765	0.381	0.555

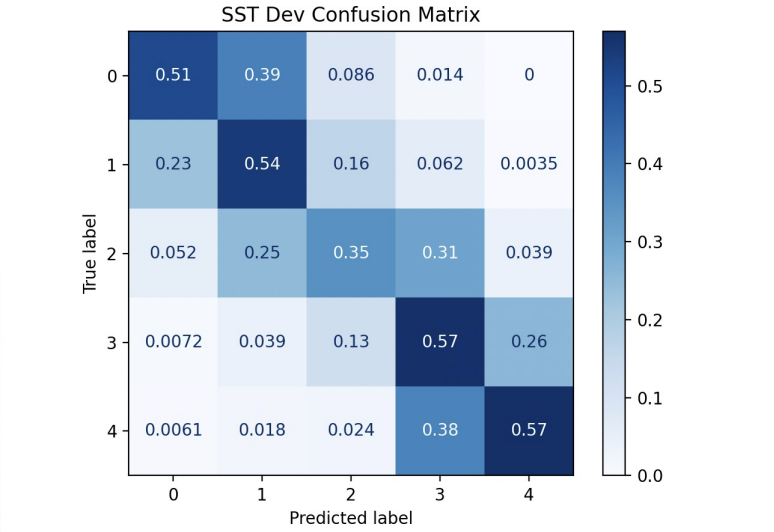
All reported baselines are from dev test.

A.2 Dropout Table

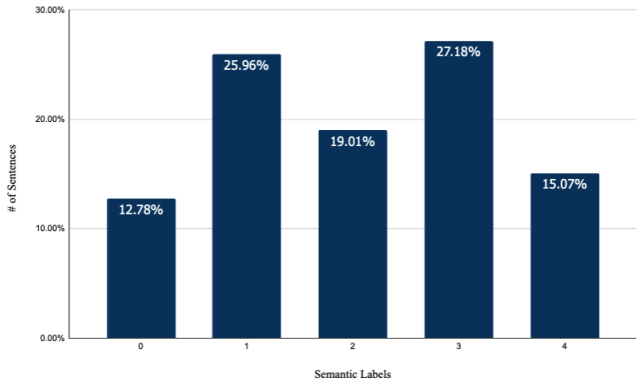
Dropout	SST	STS
0.1	0.525	**0.884**
0.2	0.525	0.880
0.3	**0.531**	**0.883**

Table 6: Dropout

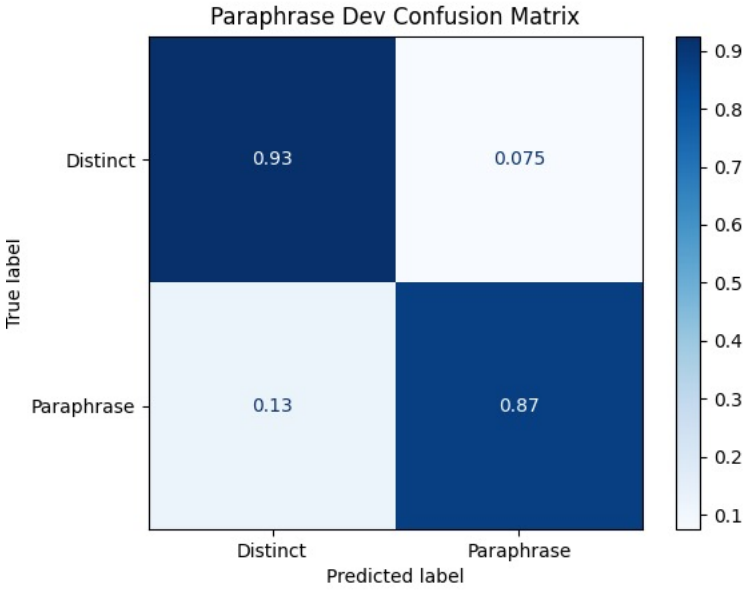
A.3 SST Confusion Matrix



A.4 SST Training Data Label Distribution



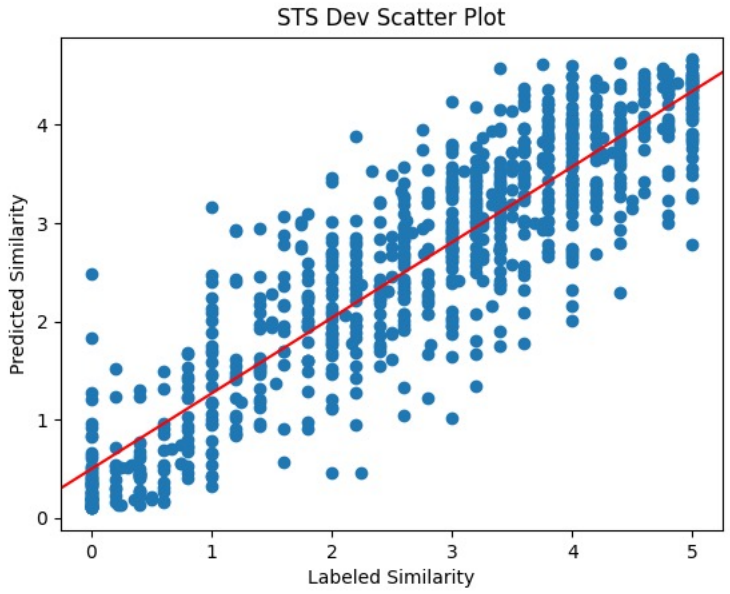
A.5 Paraphrase Confusion Matrix



A.6 Specific Sentence Examples

Example	Sentence 1	Sentence 2	Prediction	Actual
1	Should black people wear sunblock?	Do black people need to put on sunscreen lotions during summer?	0	1
2	How do plants produce oxygen?	How do plants produce oxygen during photosynthesis? Do all plants do this?	0	1
3	Why do I need a catalytic converter?	Do you really need a catalytic converter?	0	1
4	What are some molecules that are toxic in vitro and non-toxic in vivo?	Biochemistry: What are some molecules that are non-toxic in vitro and toxic in vivo?	1	0
5	It's also a matter of taste.	It's definitely just a matter of preference.	2.78	5.0
6	Work into it slowly.	It seems to work.	2.49	0.0
7	fast , frantic and fun , but also soon forgotten	N/A	N/A	2

A.7 STS Scatter Plot



A.8 SST Sentence Length Distribution