Improving minBERT on Downstream Tasks Through Combinatorial Extensions

Stanford CS224N Default Project
Mentor: Timothy Dai

Yichen Jiang† **Ria Calcagno**‡ **Senyang Jiang**‡
†Department of Electrical Engineering ‡Department of Computer Science
Stanford University
{ycjiang, ria, senyangj}@stanford.edu

Abstract

We identify a combination of modifications to BERT that performs best across a range of NLP tasks: sentiment analysis (SA), paraphrase detection (PD), and semantic textual similarity (STS). As a baseline, we build minBERT, a simplified BERT model. We improve this model by implementing combinations of extensions and adjustments: 1) SBERT[1] to improve sentence embeddings and efficiency; 2) LoRA[2] to improve efficiency; 3) proportional batch sampling from BERT and PALs Section 4.1[3] to reduce overfitting, and 4) additional fine-tuning data. We make hyperparameter adjustments within these extensions where applicable to maximize the performance of each individual extension. We find that performance is maximized by majority vote ensembling the predictions of multiple model variations where SBERT is combined with proportional batch sampling. LoRA increases efficiency via reducing trainable parameters, but decreases performance. Additional finetuning data has mixed effects on performance.

1 Team Contributions

All team members contributed to each project stage. Yichen and Ria built most of minBERT. Yichen built the LoRA extension. Senyang built the SBERT and additional data extensions and combined the SBERT and proportional batch sampling extensions. Ria built the proportional batch sampling and ensembling extensions. Senyang conducted qualitative analysis on tasks. See Appendix for details.

2 Introduction

BERT [4] (Bidirectional Encoder Representations from Transformers) generates contextual word representations from tokenized inputs and has achieved state-of-the-art results in NLP tasks.

Since its introduction, many improvements ("extensions") on BERT have been put forward. However, it appears less work has been done to find the best combinations of these extensions to improve performance on a specific set of tasks. We leverage combinations of extensions to improve upon vanilla BERT's performance on three downstream tasks: sentiment analysis (SA), paraphrase detection (PD), and semantic textual similarity (STS).

As a baseline, we build minBERT, a simplified BERT model outlined in the default project handout. We improve this model by implementing the following extensions and adjustments: 1) SBERT[1] to improve sentence embeddings and efficiency; 2) LoRA[2] to improve efficiency; 3) proportional batch sampling from BERT and PALs Section 4.1[3] to reduce overfitting, and 4) additional fine-tuning data. Within these individual extensions, we test hyperparameter adjustments where applicable.

We selected these particular extensions to balance one another, as discussed in Related Work.

Performance is maximized by majority vote ensembling predictions of the following model variations:

1. SBERT + annealed proportional batch sampling
2. SBERT + annealed proportional batch sampling + additional finetuning data
3. SBERT + square root proportional batch sampling + additional finetuning data
4. SBERT + square root proportional batch sampling
5. SBERT

We also find that LoRA increases efficiency by decreasing trainable parameters, but decreases performance. Additional finetuning data yielded mixed effects.

3 Related Work

We research extensions that may balance one another in combination. SBERT tends to overfit, addressed by proportional batch sampling. These increase runtime, addressed by LoRA. The original SBERT paper leverages additional finetuning data, and additional data usually improves performance.

3.1 SBERT

NLP models like BERT require both sentences to be processed simultaneously, which results in substantial computational burden. Matching the most similar sentence among 10,000 sentences requires $\sim$50 million inference computations - roughly 65 hours of processing time with BERT.

To address these challenges, Reimers and Gurevych (2019)[1] propose Sentence-BERT (SBERT), which modifies the pretrained BERT model by employing siamese and triplet network structures to generate semantically meaningful sentence embeddings. These embeddings can be efficiently compared using cosine similarity, reducing the search time for similar sentence pairs from 65 hours to $\sim$5 seconds while maintaining the accuracy of BERT, greatly improving the efficiency of the model.

3.2 BERT and PALs

Traditional fine-tuning of BERT for each task separately can be computationally expensive and resource-intensive. To address this, Stickland and Murray (2019)[3] propose Projected Attention Layers (PALs). Task-specific attention layers are added to the pre-trained BERT model, facilitating learning task-specific features more efficiently and improving performance on NLP tasks.

However, PALs were not the only innovation introduced in this paper. We adopt their proportional batch sampling method, training the model on tasks with probability proportional to the tasks' dataset sizes, ensuring balanced learning across all tasks. This approach aims to mitigate overfitting on tasks with fewer training examples and reduce under-training on tasks with more extensive datasets.

3.3 LoRA

Most NLP models require pre-training on general tasks and fine-tuning on specific sub-tasks. As model scales increase, retraining becomes prohibitively expensive. Low-Rank Adaptation (LoRA)[2] improves fine-tuning efficiency by reducing the number of trainable parameters.

LoRA freezes initial weight matrices from pre-trained models, inserting two trainable rank-decomposition matrices into each layer to learn sub-task features. Despite fewer trainable parameters, LoRA matches or exceeds traditional full fine-tuning performance across models like RoBERTa, DeBERTa, GPT-2, and GPT-3, enhancing fine-tuning efficiency without compromising model quality.

4 Approach

1. We build our baseline model minBERT on top of provided starter code. We test small adjustments in sentence embedding formation to select the better-performing baseline.
2. We implement the following extensions individually (ideas are not our own, implementation/code is our own): 1) SBERT[1], 2) LoRA[2], 3) proportional batch sampling[3], and 4) additional fine-tuning data. We adjust hyperparameters as relevant in individual testing.

3. We create models that combine the better-performing extensions, SBERT and proportional batch sampling. We test including additional finetune data on these.
4. We ensemble the outputs of the top five performance models. We submit the ensemble and the best non-ensembled model to the test set leaderboard (all prior testing was on dev set).

4.1 Baseline

Our baseline is minBERT as outlined in the default final project handout, with no additional data for training. All three downstream tasks use the embedding corresponding to the [CLS] token. For PD and STS tasks, we test adding vs concatenating the two sentence pair embeddings to produce a single embedding, selecting the better performer for our baseline. We feed the embedding to another linear layer to obtain the final logits for prediction. We use multi-class cross-entropy loss for all three tasks.

4.2 SBERT

Sentence-BERT [1] modifies the pretrained BERT using a Siamese network structure to derive semantically meaningful sentence embeddings that can be compared using cosine-similarity.

For classification tasks (e.g. PD), we concatenate sentence embeddings u and v with the element-wise difference $|u - v|$, multiply it by the trainable weight W_t, and apply softmax to obtain predictions. We optimize cross-entropy loss.

$$o = \text{softmax}(W_t(u, v, |u - v|))$$

For regression tasks (e.g. STS), we directly compute the cosine similarity between two sentence embeddings. We then optimize for mean-squared error (MSE). Since cosine similarity ranges from -1 to 1, we scale it to range from 0 to 5 to match the SemEval STS Benchmark Dataset.

In addition, for PD and STS, we emulate using the average of BERT output vectors as the embedding, instead of using the output vector from the [CLS] token as in our baseline model.

4.3 Additional data for fine-tuning

SBERT[1] uses additional data from a Natural Language Inference (NLI) dataset for fine-tuning. As in the original paper, we perform additional fine-tuning using the ALL-NLI dataset, a concatenation of the Stanford Natural Language Inference (SNLI) Corpus and the Multi-Genre Natural Language Inference (MultiNLI) Corpus. We fine-tune with a 3-way softmax classifier objective function.

4.4 LoRA

Low-**R**ank **A**daptation (LoRA) [2] was first proposed to enhance fine-tuning efficiency without increasing inference latency. The core hypothesis behind LoRA is that the changes in weights during model adaptation possess a low "intrinsic rank."

We utilize and adapt the LoRA library developed by the original team. The LoRA library provides implementations for convolution, embedding, and linear layers. In this project, we only replace the linear layers in minBERT, leaving the embedding layers unchanged. We believe that embedding layers function more as lookup tables rather than dense layers, which LoRA is designed to modify.

We implement LoRA for every linear layer in the Transformer architecture to reduce the number of trainable weights in fine-tuning. For each weight matrix $W_0 \in \mathbb{R}^{d \times k}$ from the linear layers in the pre-trained model, LoRA processes its update by decomposing ΔW into two low rank matrices, $B \in \mathbb{R}^{d \times r}$ and $A \in \mathbb{R}^{r \times k}$, with rank $r \ll \min(d, k)$. During fine-tuning, W_0 is frozen and stays unchanged, while A and B contain trainable parameters and receive gradient updates. Thus, in fine-tuning stage, the modified forward pass in LoRA becomes:

$$h = W_0 x + \Delta W x = W_0 x + BAx$$

By implementing such decomposition, LoRA can fine-tune a pre-trained model much more efficiently (especially for large-scale models) when compared to traditional fine-tuning approaches, while freezing the pre-trained weight matrix W_0 helps preserve the original model quality.

4.5 Proportional task-based sampling from BERT and PALs

BERT and PALs[3] suggests proportional batch sampling to balance overfitting on tasks with fewer examples and under-training on tasks with more.

Our minBERT baseline samples batches equally across datasets. With proportional batch sampling, batch sampling probability p_i is proportional to the number of training examples N_i raised to some α: $p_i \propto N_i^{\alpha}$

We test square root sampling ($\alpha = 0.5$) and annealed sampling ($\alpha = 1 - 0.8\frac{e-1}{E-1}$) where α changes with each epoch e relative to the total epochs E to train with more even probability in later epochs.

Our *original** implementation calculates dataset weights proportional to their sizes and scales them for rounds of size r. Each epoch samples batches from datasets according to these weights until the desired batch quantity b is reached. For example, if $r = 50$, and the weights for Datasets 1, 2, and 3 are 0.2, 0.6, and 0.2, respectively, they are sampled 10, 30, and 10 times per round until b is reached.

We test this method individually and combined with SBERT, using square root and annealed sampling.

4.6 Optimizing modifications:

1. **Independent testing:** We test each larger extension (SBERT, LoRA, proportional batch sampling, additional finetune data) independently. We adjust hyperparameters.
2. **Combinatorial testing (combined model):** Based on these results, which showed that SBERT and proportional batch sampling improve performance, we create *original* combined SBERT + proportional batch sampling models, with and without additional finetuning data.
3. **Combinatorial testing (ensembling outputs):** We test majority vote ensembling the ouputs of the top five performance models (idea from TA Timothy Dai, our *original* implementation/code). We average the predictions, rounding to the nearest integer for PD and SA.

5 Experiments

5.1 Data

- SA: Stanford Sentiment Treebank (SST) dataset. Input: a phrase extracted from movie reviews. Output: a label in 1 of 5 categories (negative, somewhat negative, neutral, somewhat positive, or positive).
- PD: Quora Dataset. Input: a question pair. Output: a label indicating paraphrase status.
- STS: SemEval STS Benchmark Dataset. Input: a sentence pair. Output: a similarity score scaled from 0 (unrelated) to 5 (equivalent meaning).
- Additional data: ALL-NLI dataset. Input: a sentence pair. Output: a label indicating their relations in one of three categories (entailment, neutral, or contradiction).

5.2 Evaluation method

- Performance SA and PD: accuracy
- Performance STS: Pearson correlation of the true similarity values against the predicted similarity values
- Performance Overall: (SA accuracy + (STS correlation + 1)/2 + PD accuracy) / 3
- Efficiency: Number of trainable parameters

5.3 Experimental details

All experiments use a Nvidia T4 GPU. We use batch size of 8 and dropout probability of 0.3. The learning rate is 1e-3 when fine-tuning only the last layer (for testing sentence embedding adjustments for the baseline) and 1e-5 when fine-tuning the full model. All models are trained for 10 epochs.

5.3.1 SBERT

For PD and STS, we adjust the loss function and the MLP projection layer according to the SBERT structure mentioned in 4.2. We do not change the architecture of the output head for SA because SBERT only considers comparison between two sentences.

5.3.2 Additional finetune data

To fine-tune on the ALL-NLI data, we add another output head to our multitask BERT classifier that takes embeddings from two sentences and outputs a label in 1 of 3 categories: entailment, neutral or contradiction. We first fine-tune the model on the ALL-NLI dataset for 1 epoch using the additional output head, and then fine-tune on the datasets for SA, PD and STS just like the other models.

5.3.3 Proportional batch sampling

We test optimal round size r by evaluating dev set performance after 1 epoch when $\alpha = 0.5$ (square root sampling). We test $r = 10, 50, 100$ and find that accuracy is maximized when $r = 50$.

Holding $r = 50$ constant, we run the full model using square root sampling where $\alpha = 0.5$ and annealed sampling where $\alpha = 1 - 0.8\frac{e-1}{E-1}$.

5.3.4 LoRA

In our implementation of LoRA, we replace most linear layers in the original BERT model with their corresponding LoRA linear layers. However, we do not modify the last few projection and prediction layers, as their output dimensions are too small (≤ 10). This small size negates the need to reduce trainable parameters using LoRA in these layers.

We explore the trade-off between overall model quality and the rank used in all LoRA layers. We briefly experiment with different learning rates, finding a learning rate of 1e-5 most effective for model learning. We test ranks of 6, 10, 16, 20, 30, and 50 for all LoRA layers during fine-tuning.

5.3.5 Combination models

Since SBERT and proportional batch sampling consistently improve overall performance, we test combinations of these extensions, with/out additional finetune data: SBERT + proportional batch sampling (annealed), SBERT + proportional batch sampling (annealed) + additional data, SBERT + proportional batch sampling (square root), and SBERT + proportional batch sampling (square root) + additional data. We submit the best-performing combination model to the test set leaderboard.

5.3.6 Ensemble

We test majority vote ensembled outputs of the top 5 performing models and the top 3 performing models. We submit the ensemble to the test set leaderboard.

5.4 Results

5.4.1 Baseline

Concatenating embeddings performs better than adding. Fine-tuning the whole model yields much higher accuracy than only fine-tuning the last layer. In comparative analysis, we use the baseline version with highest accuracy (concatenates the embedding and fine-tunes the whole model).

5.4.2 SBERT

SBERT improves PD (+0.023) and STS (+0.265) performance. This is expected: SBERT designs an embedding more suitable for similarity analysis between two sentences. SA performance is unchanged. This is expected since the model architecture for this task is the same as the baseline: we use the [CLS] token as the embedding and use the same linear layer to get the logits for prediction.

Experiment	Model Version	# Trainable Parameters	Dev SA acc	Dev PD acc	Dev STS corr	Overall Performance
Baseline	Last layer/add	5,383	0.395	0.368	0.230	0.459
	Last layer/concat	6,919	0.396	0.368	0.276	0.467
	Full model/concat	**109,489,159**	**0.466**	**0.796**	**0.354**	**0.646**
SBERT	N/A	109,490,695	0.465	0.819	0.619	0.698
Additional Data	Baseline +Additional data	109,493,770	0.431	0.774	0.331	0.624
	SBERT +Additional data	109,497,610	0.491	0.782	0.615	0.694
Proportional Batch Sampling	Square root	109,489,159	0.498	0.816	0.321	0.658
	Annealed	109,489,159	0.500	0.811	0.320	0.657
LoRA	Rank = 6	1,004,544	0.254	0.604	0.214	0.488
	Rank = 10	1,674,240	0.291	0.764	0.305	0.569
	Rank = 16	2,678,784	0.290	0.759	0.270	0.561
	Rank = 20	3,348,480	0.309	0.747	0.251	0.561
	Rank = 30	5,022,720	0.277	0.764	0.263	0.558
	Rank = 50	8,371,200	0.261	0.435	0.276	0.445
Combination Models	SBERT + prop sampling (square root)	109,490,695	0.493	0.893	0.647	0.737
	SBERT + prop sampling (anneal)	**109,490,695**	**0.513**	**0.900**	**0.644**	**0.745**
	SBERT + prop sampling (square root) + additional data	109,497,610	0.498	0.896	0.65	0.740
	SBERT + prop sampling (anneal) + additional data	109,497,610	0.503	0.898	0.659	0.744
Ensemble	**Top 3 models**	**N/A**	**0.512**	**0.904**	**0.72**	**0.759**
	Top 5 models	**N/A**	**0.512**	**0.904**	**0.72**	**0.759**

Table 1: Performance (accuracies) and efficiency (trainable parameters) of all models on the dev set.

Experiment	Model	Test SA acc	Test PD acc	Test STS acc	Overall Performance
Ensemble	Ensemble (5 models)	0.521	0.901	0.707	0.759
Combination Models	SBERT +prop sampling (anneal)	0.503	0.898	0.606	0.735

Table 2: Accuracies of selected models on the test set.

5.4.3 Additional finetune data

Fine-tuning on the ALL-NLI data yields mixed results, improving overall performance for the SBERT + square root proportional batch sampling model, but worsening performance otherwise. This is notable, as additional finetune data is expected to improve performance. This suggests that the auxiliary NLI task was not helpful for our target tasks. We might need to adjust the hyperparameters during additional finetuning, such as batch size, learning rate, and epoch size, to get better results.

5.4.4 Proportional Batch Sampling

Proportional batch sampling improves performance on SA (+0.033) and PD (+0.018), raising overall performance (+0.012), but decreases STS performance (-0.034). This likely results from differences in dataset sizes and sampling strategies.

The Quora dataset for PD (283,003 examples) is much larger than the SA and STS datasets (8,544 and 6,040 examples). In the baseline model, this imbalance likely causes under-training on PD and overtraining on SA and STS. Proportional batch sampling corrects this by balancing dataset contributions, thus improving PD and SA performance.

However, proportional batch sampling disrupts the STS performance, possibly because the baseline model's CombinedLoader sequential sampling allows STS data to drive gradient updates later in training, stabilizing this task's performance. Proportional sampling rotates datasets, preventing consistent fine-tuning of STS data in later epochs. An alternative hypothesis is that larger gradient updates for the baseline occur earlier in training (on PD and SA data), necessitating further investigation.

There is minimal difference between annealed and square root sampling. Square root sampling slightly outperforms on PD (+0.005) and STS (+0.001), giving a slight overall boost (+0.001). Annealed sampling slightly outperforms on SA (+0.002). This contradicts the original findings[3], where annealed sampling outperformed, though our results may be within a margin of error.

5.4.5 LoRA

LoRA significantly reduces trainable parameters, requiring less memory for the optimizer during fine-tuning. Selecting an inappropriate rank—too high (e.g., 50) or too low (e.g., 6)—leads to suboptimal performance, potentially due to limited expressiveness and slow convergence, respectively.

Our best LoRA version (rank = 10) achieves comparable accuracy for PD (0.796 baseline vs. 0.764 LoRA) and slightly lower for STS (0.354 baseline vs. 0.305 LoRA), but shows noticeable degradation for SA (0.466 baseline vs. 0.291 LoRA).

LoRA performs well in simpler tasks like PD but worse in complex tasks like SA and STS. SA requires understanding subtle nuances and context, needing more detailed parameter adjustments than LoRA provides. LoRA's modification of only a subset of parameters is insufficient for such tasks. Therefore, we exclude LoRA when combining extensions for optimal model quality.

5.4.6 Combination models

All combination models of SBERT + proportional batch sampling outperform the individual models. Since SBERT tends to exacerbate overfitting, it makes sense that pairing it with proportional batch sampling, which balances over/underfitting based on dataset size, improves performance. It is notable that annealed proportional batch sampling when paired with SBERT significantly outperforms the same combination with square root sampling, while square root sampling outperformed annealed sampling without SBERT.

5.4.7 Ensemble

Ensembles of the top five and top three models performed identically, outperforming all other models overall and on all tasks except SA, where the solo SBERT + annealed proportional batch sampling model led by +0.001. The ensemble outperformed this solo model on the test set: +0.018 on SA, +0.003 on PD, +0.101 on STS, and +0.024 overall. It was surprising that both ensembles performed the same and that including input from worse models improved results.

The ensemble's performance boost likely comes from correcting outlier predictions in individual models, leveraging different model strengths. The equal performance of the top 3 and top 5 ensembles suggests the top 5 benefits from more models to correct outliers, while the top 3 benefits from only better-performing models.

6 Analysis

Hypotheses for why each extension yielded its results are included in discussion of results above.

We discuss qualitative results for the best non-ensembled model: SBERT with annealed proportional batch sampling, no additional data, without LoRA. We use a visualization tool called Learning Interpretability Tool (LIT) [5], which provides visualizations such as saliency map (using LIME [6]) and confusion matrix. We examine our best model's performance on the downstream tasks.

6.0.1 Sentiment Analysis

Figure 2 shows an example of correct sentiment classification. The model is pretty confident (96.7%) in its prediction. The salience map shows why the model makes this prediction. The words 'although', 'best', and 'popular' influence the model's predictions most. This interpretation makes sense because 'best' and 'popular' are strongly positive words, and 'although' indicates the sentence is not giving an absolute statement, hence the sentiment would be somewhat positive.

Figure 3 shows an example of incorrect sentiment classification. The model's interpretation makes less sense here. The source of error may be the model's understanding the word 'desperate' as incorrectly correlated with the 'somewhat positive' sentiment.

Figure 4 shows the confusion matrix for the model's predictions on the SA dev set. Most data points are close to the diagonal, indicating the model's predictions are mostly correct, but it may confuse sentiment of nearby categories. The confusion matrix also shows the model's most common mistake is predicting 'positive' as 'somewhat positive.' This makes sense because though it is easy to classify a sentence as positive or negative, it is also hard even for humans to determine the absolute strength of a sentence.

6.0.2 Paraphrase Detection

Figure 5 and Figure 6 show examples of correct and incorrect PD by the model. The salience map is less interpretable here than with SA. In Figure 5 where the model correctly predicts a paraphrase, some words that match each other have positive correlation with the prediction. In Figure 6 where the model incorrecly predicts the sentences as not paraphrases, the word 'would' in the second sentence heavily influences the prediction, which makes little sense for a human interpreter. This may be a hard sentence pair because the lengths of the two sentences are very different, despite being paraphrases.

6.0.3 Semantic Textual Similarity

Figure 7 and Figure 8 show an example where the predicted similarity is close or far from the true similarity, respectively. In easy cases like figure 7 where the two sentences are the same except for a few words, the model correctly gives a high similarity score. However, in cases like figure 8 where some words appear in both sentences but the meaning is different, the model overestimates similarity. This indicates our model's embedding does not effectively capture meaning of the sentences, and directly using cosine similarity between the sentence embedding is insufficient.

7 Conclusion

We explored enhancements to BERT on SA, PD, and STS tasks, using minBERT as a baseline. We tested SBERT, proportional task sampling, LoRA, and additional training data, experimenting with combinations and hyperparameters to optimize performance.

We found that majority vote ensembling our top five models achieved maximum performance. Without ensembling, SBERT with annealed proportional batch sampling performed best. LoRA reduced memory for fine-tuning and maintained performance in simpler tasks but was less effective for complex tasks like SA and STS.

Our four-week time constraint given 20-hour runtimes limited our ability to incorporate more sophisticated extensions. Limited GPU access restricted exploration of memory-intensive models.

Future work could improve our best model by integrating additional extensions to enhance performance, potentially allowing for efficiency-focused extensions like LoRA. Alternatively, testing DoRA, an improved LoRA-based architecture, could mitigate LoRA's performance degradation.

References

[1] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 11 2019.

[2] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022.

[3] Asa Cooper Stickland and Iain Murray. Bert and pals: Projected attention layers for efficient adaptation in multi-task learning, 2019.

[4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

[5] Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for SQuAD. In *Association for Computational Linguistics (ACL)*, 2018.

[6] Marco Túlio Ribeiro, Sameer Singh, and Carlos Guestrin. "why should I trust you?": Explaining the predictions of any classifier. *CoRR*, abs/1602.04938, 2016.

[7] Rajas Bansal. A survey on bias and fairness in natural language processing, 2022.

[8] Chiara Longoni and Luca Cian. When do we trust ai's recommendations more than people's?, Oct 2020.

A Appendix

A.1 Team Contributions - Detail

Ria Calcagno

- Proposal: project description, new paper summary
- minBERT: optimizer.py, multitaskclassifier.py
- Milestone: abstract, proportional batch sampling info, editing
- Extension: proportional batch sampling, ensembling
- Final report: abstract; team contributions; introduction; summary, proportional sampling, optimizing modifications subsections of approach section; evaluation method; proportional sampling, combined models, and ensemble subsections of experimental details section; additional data, proportional sampling, combined models, and ensemble subsections of results section; additional data, proportional sampling, combined models, and ensemble subsections of analysis section; ethical considerations; editing throughout (reduced 12 pages to 8)
- Poster: formatting, project overview, datasets and metrics, background and related work (SBERT and proportional sampling), methods and experiments, results

Yichen Jiang

- Proposal: original paper summary (had to change, originally wrote about BERT paper)
- minBERT: bert.py, classifier.py, debugging multitaskclassifier.py
- Milestone: future work, LoRA info
- Extension: LoRA
- Final report: related work, approach section LoRA subsection, experiments section LoRA subsection, LoRA part of analysis section, Ethical Consideration, editing
- Poster: analysis, references

Senyang Jiang

- Proposal: project description
- minBERT: debugging
- Milestone: finished SBERT extension in time for milestone, generated baseline data, experiments
- Extension: SBERT, additional data, combining SBERT with proportional batch sampling
- Final report: approach/experiments for SBERT and additional data, experiments data, result tables, qualitative analysis, references
- Poster: qualitative analysis, references, results

A.2 SBERT architecture diagram

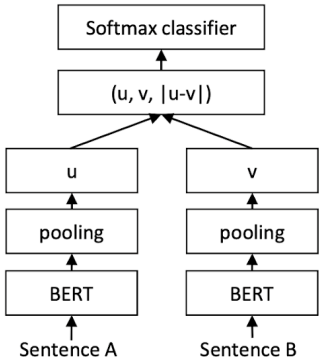

Figure 1: SBERT architecture with classification objective function, e.g., for fine-tuning on SNLI dataset. The two BERT networks have tied weights (siamese network structure).

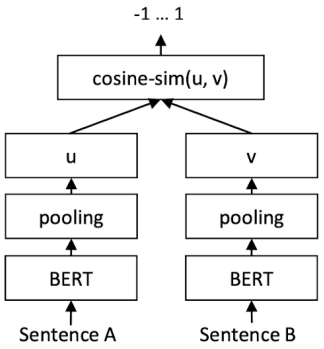

Figure 2: SBERT architecture at inference, for example, to compute similarity scores. This architecture is also used with the regression objective function.

Figure 1: The architecture of the SBERT model from [1]

A.3 Qualitative analysis visualizations

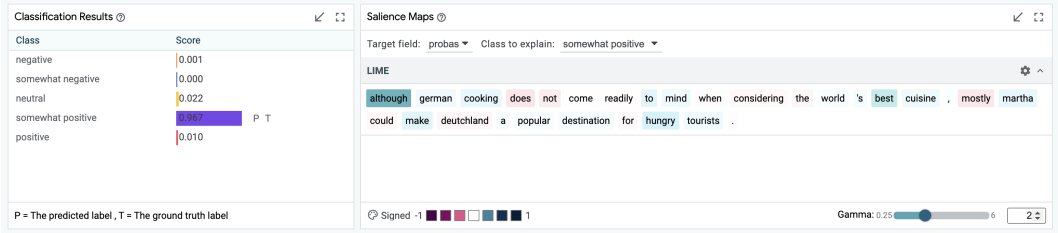

Figure 2: One example of correct sentiment classification. (a) Left shows classification results with different probabilities. (b) Right shows salience map, indicating how much each input affects model's decision. Blue/green shows positive correlation, and pink/purple shows negative correlation.

A.4 Note on original implementation approach of proportional batch sampling

Developed with influence from conversations with various TAs, including Timothy Dai and Kaylee Burns.

Figure 3: One example of incorrect sentiment classification.

sst:probas:class

label \	negative	somewhat negative	neutral	somewhat positive	positive	Total
negative	4.3% (47)	6.1% (67)	1.8% (20)	0.5% (5)	0.0% (0)	12.6% (139)
somewhat negative	2.5% (27)	16.0% (176)	5.3% (58)	2.4% (26)	0.2% (2)	26.2% (289)
neutral	0.4% (4)	5.0% (55)	7.7% (85)	7.3% (80)	0.5% (5)	20.8% (229)
somewhat positive	0.2% (2)	0.9% (10)	4.5% (50)	17.0% (187)	2.7% (30)	25.3% (279)
positive	0.0% (0)	0.4% (4)	0.6% (7)	7.9% (87)	6.1% (67)	15.0% (165)
Total	7.3% (80)	28.3% (312)	20.0% (220)	35.0% (385)	9.4% (104)	

Figure 4: Confusion matrix for sentiment classification. X-axis denotes predicted category, Y-axis denotes true category.

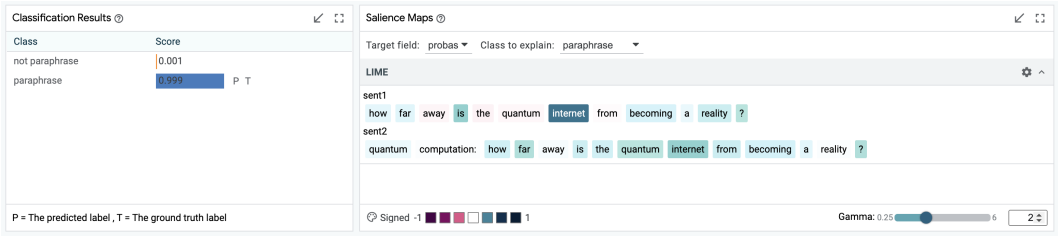

Figure 5: One example of correct paraphrase detection.

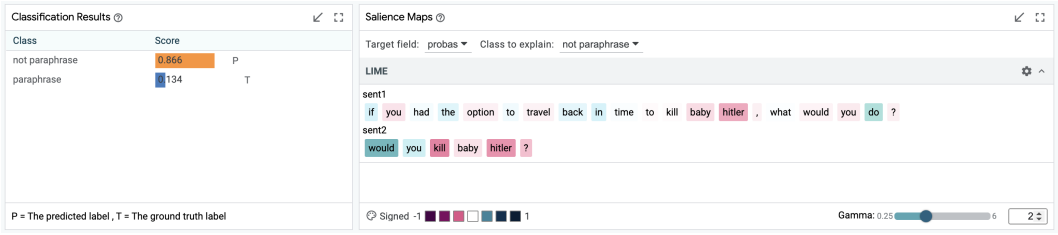

Figure 6: One example of incorrect paraphrase detection.

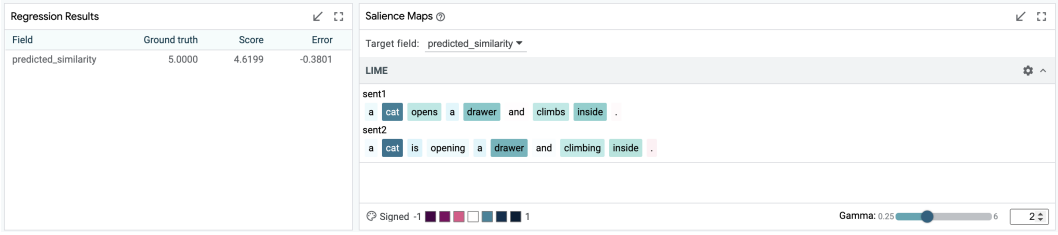

Figure 7: One example of good similarity prediction.

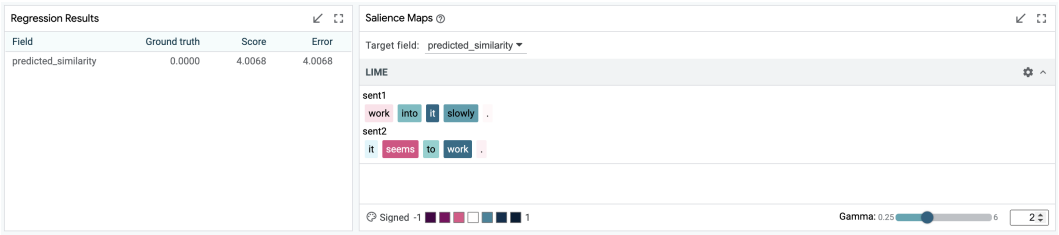

Figure 8: One example of bad similarity prediction.

B Ethical Considerations

Our project primarily focused on experimenting with various published extensions and their combinations to optimize BERT model performance on basic NLP tasks such as sentiment analysis, paraphrase detection, and semantic textual similarity. One might think that such basic, seemingly neutral tasks would be free of ethical implications. However, it is crucial to recognize potential downstream uses of these extensions and the resulting ethical concerns; these should be anticipated and mitigated as much as possible upstream. As AI research advances towards artificial general intelligence (AGI), where superior model performance on core tasks translates to any task, upstream mitigation in basic tasks like those in our project becomes even more important.

Our baseline minBERT implementation built upon pretrained embeddings supplied with the default project starter code. Caution is essential when building off pretrained embeddings, as previous researchers and developers may not have exercised the same level of ethical consideration. For instance, typical word embeddings have been found to associate European American names more positively than African American ones, and certain axes of bias, such as religious bias, are less frequently assessed [7]. A more ethically thorough version of our project would involve assessing potential biases in outputs generated by minBERT first without additional extensions, to decide whether further pretraining measures are necessary.

One of our extensions involved supplying additional data during fine-tuning. Introducing new data introduces risks of unvetted content, which might lead to learning biases or other unintended language patterns from these "contaminated" datasets. This risk increases with the use of increasingly large amounts of data, prioritizing quantity over quality. As noted in the CS 224N lecture, more colloquial datasets generated from online posts (such as the Quora dataset used for our paraphrase detection task) are proving more useful in NLP than more formal content such as from Wikipedia. The ethical benefit here is that such data may contain a diversity of linguistic styles and dialects, rather than the predominantly white, western, upper-class standard English used in much published English literature. However, this less censored content may be at higher risk of containing non-politically-correct, offensive, or stereotypical language and ideas. This underscores the importance of responsible NLP development when incorporating large datasets, particularly those that may not be thoroughly vetted. To mitigate unintended effects of additional datasets, it is essential to analyze their impact using various tools and metrics, both quantitatively and qualitatively. For example, checking the model's accuracy on downstream tasks before and after adding new data, and using salience maps to visualize how the model interprets inputs can ensure that the additional dataset has not introduced significant bias compared to the other data. Further ethics research on our best-performing models would use these tools to examine sentiment classification of sentences containing words associated with disability, homosexuality, race, etc.

Another ethical issue arises with LoRA, one of our extensions, which makes training more efficient but may compromise performance on complex emotional tasks like sentiment analysis, as evidenced by our experimental results. This can impact downstream use: for example, reducing the number of trainable parameters with LoRA may lose language nuances crucial for accurately categorizing sentiments in social media posts, potentially impacting decisions about content moderation. This poses a risk that companies and institutions, as well as researchers on limited budgets and tight deadlines, might prioritize cost- and time-saving measures over model quality, leading to models less adept at understanding human emotions. Inaccurate emotion detection and consequent inappropriate responses can be significant problems, especially as humans increasingly rely on AI for advice. However, research from the Harvard Business Review shows that consumers currently rely more on

AI for “utilitarian and functional” matters, preferring human advice for “experiential and sensory” matters [8]. Our research with LoRA highlights the challenges researchers face in improving efficiency while building ethical trust in AI for emotional, sensitive matters. A potential solution is for reviews such as Stanford’s Ethics and Society Review to place more emphasis on performance in complex tasks like sentiment analysis. Even if emotional tasks are not the main goal of one’s research, including supplemental testing on such tasks may serve as a useful check for lurking inaccuracies and biases that might not be evident in less emotionally oriented tasks but may nevertheless impact downstream uses.

Despite the ethical concerns posed by some extensions in our project, one of our extensions offers a potential solution. Proportional batch sampling can prevent the model from over-relying on certain datasets. Adapting our algorithm via changes to our weights function provides an opportunity to guide these types of models to emphasize fine-tuning with datasets that have undergone strict inspection and are known to be "clean." This can help mitigate biases in the models and produce more ethically reliable language models.