Beyond BERT: a Multi-Tasking Journey Through Sampling, Loss Functions, Parameter-Efficient Methods, Ensembling, and More

Stanford CS224N Default Project

Alycia Lee
Department of Computer Science
Stanford University
`alylee15@stanford.edu`

Amanda Li
Department of Computer Science
Stanford University
`amanli@stanford.edu`

Abstract

In this work, we improve a pretrained minBERT model via fine-tuning for simultaneous performance on three NLP tasks: sentiment analysis, paraphrase detection, and semantic textual similarity. We leverage and test techniques proposed by the community and adopted in industry to improve our model's performance, namely: annealed sampling, Multiple Negative Ranking Loss (MNRL), Parameter Efficient Fine-Tuning (PEFT) methods like LoRA and DoRA, cosine similarity loss, and ensembling. We introduce an ensembled, hyperparameter-tuned BERT method that utilizes annealed sampling, MNRL, cosine similarity loss, LoRA, and DoRA. Our method produces the best overall results of techniques tried, and exceeds our baselines in dev and test performance.

1 Key Information to include

- TA mentor: Olivia Lee
- External collaborators (if no, indicate "No"): No
- Sharing project (if no, indicate "No"): No
- Team contributions: Alycia implemented MNRL, LoRA, and DoRA. Amanda implemented annealed sampling, cosine similarity loss, and ensembling. Both members contributed to the minBERT implementation, hyperparameter tuning, analysis, and writeup.

2 Introduction

As the architecture of natural language models continues to evolve and the Internet provides access to vast amounts of natural language data, NLP methods have become useful in an increasing variety of prediction, analysis, and generative tasks. Models pretrained on large, diverse corpora capture rich semantic language representations. These models can be fine-tuned on task-specific datasets for various downstream tasks. An important area of research is multi-tasking, where the same model is applied to different tasks. This approach is particularly beneficial for mobile, or on-device, applications where computational resources are limited, as sharing parameters can enhance inference efficiency (Stickland and Murray (2019)).

Since its inception, the Bidirectional Encoder Representation from Transformers (BERT) model (Devlin et al. (2018)) has become both a popular and staple NLP architecture for tasks such as question answering and machine translation. BERT has also inspired a number of extensions, including BERT and PALs (Stickland and Murray (2019)). In this paper, our focus is on devising a strategy for fine-tuning a pretrained BERT model. Our goal is to achieve high performance simultaneously on three downstream tasks: sentiment analysis, paraphrase detection, and semantic textual similarity.

To improve our fine-tuned model's performance, we employ techniques such as annealed sampling, Multiple Negative Ranking Loss (MNRL), Parameter Efficient Fine-Tuning (PEFT), and ensembling. We find that using a combination of annealed sampling and MNRL enabled us to achieve the best single model performance across all tasks with an overall score of 0.682. Using majority voting ensembling, we were able to boost the overall dev performance by combining the predictions from our five best fine-tuned models. The dev performance achieved by this 5-ensemble model was 0.707, and the test performance achieved was 0.712.

3 Related Work

Sentence embedding improvements. Pretrained sentence embeddings have become increasingly valuable for downstream NLP tasks like information retrieval and text summarization. These embeddings represent semantic and syntactic information of sentences in the form of dense vectors. One approach for enriching the context captured by these embeddings, in particular for downstream tasks that require comparison of multiple sentence embeddings, is to train the encoder model with Multiple Negative Ranking Loss (abbreviated as MNRL). The n-gram embedding-based neural network developed by (Henderson et al. (2017)) for "Smart Reply" is an example of work that leverages MNRL. While the authors used MNRL for efficient training of a scoring model that can score potential responses to emails in their dataset, we use MNRL to improve the sentence embeddings produced for paraphrase detection and STS tasks — both of which require comparison of sentence embeddings in every example.

Multitasking. Applying a model with common parameters to multiple downstream tasks has been an active area of research. BERT and PALs (Stickland and Murray (2019)) introduced projected attention layers for multi-task learning, and also experimented with various techniques for sampling examples from different tasks during training. We adapt these alternative sampling techniques, like square root sampling and annealing, to reduce task interference and improve performance on our own model.

PEFT. Parameter-efficient fine-tuning (PEFT) methods for adapting models for downstream tasks are motivated by the difficulty of scaling full fine-tuning methods as parameters grow. PEFT methods only retrain a subset of parameters, and include adapter-based methods (adding modules to the baseline model architecture), prompt-based methods (adapting the model input), and low-rank methods (using low-rank matrices to approximate weight updates). In this paper, we focus on variants of low-rank PEFT methods, as unlike the first two categories, they do not increase inference latency compared to the baseline model. Low-Rank Adaptation of Large Language Models (LoRA) (Hu et al. (2021)) and Weight-Decomposed Low-Rank Adaptation (DoRA) (Liu et al. (2024)) are two such variants that show promising results on downstream tasks such as commonsense reasoning and visual instruction tuning. In our research, we apply modified versions of LoRA and DoRA, and compare their performance relative to full fine-tuning on our multitasking problem.

4 Approach

Baselines. For all tasks, we ran an algorithm to generate random labels as our lowest order baseline, which gave a sentiment classification accuracy of 0.198, paraphrase detection accuracy of 0.505, and semantic textual similarity correlation of 0.004 on the dev sets. These values align with the expected values using random outputs, which are 0.2, 0.5, and 0, respectively. We used the expected values as baselines for scores on the test sets.

We first attempted to improve upon the random baseline by fine-tuning either the last-linear-layer or the full model using the pretrained minBERT model. During fine-tuning, we used round-robin downsampling to alternate amongst batches from each dataset to train the model. We used binary cross entropy loss for paraphrase detection and mean squared error loss for STS, and summed embeddings for these two tasks before passing the output through a linear layer. The full model outperformed the last-linear-layer model with scores of 0.503 for SST accuracy, 0.684 for paraphrase detection accuracy, and 0.356 for STS correlation on the dev sets. We used these 2 models as baselines for the remainder of our multitask experiments.

Multiple Negatives Ranking Loss. To improve the quality of the minBERT sentence embeddings and their ability to capture semantic relationships between sentences, we first trained the minBERT base model using the Multiple Negatives Ranking Loss. MNRL is a contrastive loss function that aims to minimize the distance in embedding space between similar sentences and maximize the distance between dissimilar sentences. For a training dataset consisting of sentence pairs $[(a_1, b_1), ..., (a_n, b_n)$ where (a_i, b_i) is a pair of similar sentences and (a_i, b_j) for $i \neq j$ is presumed to be a dissimilar pair, the loss is computed as

$$\begin{aligned} J(x, y, \theta) &= -\frac{1}{K} \sum_{i=1}^{K} \log P_{\text{approx}}(y_i | x_i) \\ &= -\frac{1}{K} \sum_{i=1}^{K} [S(x_i, y_i) - \log \sum_j e^{S(x_i, y_j)}] \end{aligned}$$

The scoring function S is computed as the cosine-similarity of model-produced embeddings per training example multiplied by a scaling factor with default value of 20.

We modified an implementation of the MNRL function based on the original source code (Reimers and Gurevych (2019)), and used train data from both paraphrase detection and STS datasets. Specifically, we used training examples from the paraphrase detection dataset with a label of 1 for "positive" and examples from STS with labels ≥ 4 to train the BERT model's sentence embeddings on.

Sampling methods. One difficulty of multitasking is optimizing for each task without sacrificing performance on the other tasks. The optimal gradient update may not be in the same direction for all tasks. Thus, it is important to cycle between data for each task during training as opposed to incorporating all data from each task sequentially. Our baseline technique of downsampling and alternating batches of training data has the disadvantage of potentially overfitting on the smaller datasets and underfitting on the largest (Quora) dataset. To combat this and allow the model to see more data from the Quora dataset, we implemented square root sampling and annealed sampling as described in the BERT and PALs paper (Stickland and Murray (2019)) with modifications from the original source code (Stickland and Murray). In both methods, we sample a batch of training examples from task i with probability p_i, where $p_i \propto N_i^{\alpha}$. Here, N_i is the number of training examples for task i. In square root sampling, we set $\alpha = 0.5$, while in annealed sampling, $\alpha = 1 - 0.8\frac{e-1}{E-1}$, where e is the current epoch and E is the total number of epochs. Setting the sampling probabilities in this way has the property of sampling more from larger datasets but also training on each task more equally towards the end of training, decreasing the chance of task interference.

Cosine similarity loss. For the STS task, we improved upon our baseline method of adding or concatenating BERT embeddings for two pieces of text before passing them through a linear layer by switching to cosine similarity. Given two embeddings x_1 and x_2, the cosine similarity is

$$\frac{x_1 \cdot x_2}{\max(||x_1||_2, \varepsilon) \cdot \max(||x_2||_2, \varepsilon)},$$

where $\varepsilon = 1e - 8$ is included to avoid division by zero. After computing the cosine similarity, we normalized the result to the range [0, 5] before computing the loss as the mean squared error between that value and the true label. Cosine similarity is an intuitive choice compared to combining embeddings; with this method, the model will learn similar embeddings for two similar pieces of text, resulting in a higher predicted value close to the true label.

Additional methods. Other methods we tried include LoRA, DoRA, and implementing a trainable, shared layer. To improve the efficiency and resource usage of the fine-tuning process while preserving performance, we implemented a simplified version of LoRA that neither uses dropout nor merges weights based on the original source code (Hu et al.). We took inspiration from (Raschka) in integrating LoRA in a custom model, and following (Hu et al. (2021)), we applied LoRA to only the query and key linear layers. In LoRA, weight updates are formulated as $W = W_0 + BA$ where W_0 is the frozen pretrained weight matrix, and A and B are trainable low-rank matrices. The intuition behind LoRA is that weight updates made during fine-tuning can be reparameterized as low-rank updates, and therefore, training fewer parameters in the smaller A and B matrices can still enable good downstream performance, while saving GPU memory and time. We also implemented DoRA based on (Raschka). DoRA is an extension upon LoRA that reportedly outperforms LoRA and can match full fine-tuning performance: it first decomposes pretrained weights into magnitude and directional components then fine-tunes both, using LoRA for the latter. DoRA weights are formulated as $W' = m\frac{W_0+BA}{||W_0+BA||_c}$ where W_0 are the pretrained weights, m is a trainable vector representing magnitude updates, and B and A are low-rank matrices representing the directional update.

In addition, we attempted a trainable, shared linear layer between paraphrase detection and STS tasks. The input to the linear layer was a concatenation of the sentence embeddings, and the output was added to the output of a task-specific linear layer. The intent of the shared layer was to combine information from both tasks and create a more expressive representation.

Ensembling. Ensemble methods are powerful in that they allow the combination of learnings from multiple methods and produce more stable and generalizable models. We ensembled multiple models by taking predictions from each and combining them as follows: for the sentiment analysis and paraphrase detection tasks, we used majority voting to decide the final label, and for the semantic textual similarity task, we took the mean prediction across models for the final output.

5 Experiments.

5.1 Data

In the first part of this project (implementing and training minBERT), we use the Stanford Sentiment Treebank (SST) dataset (sst): 11,855 sentences from movie reviews labeled "negative", "somewhat negative", "neutral", "somewhat positive", and "positive"; and the CFIMDB dataset: 2,434 highly polar movie reviews labeled "negative" or "positive". In the second part (multi-tasking), we use the SST dataset as well as the Quora dataset: 404,298 question pairs with labels indicating if questions are paraphrases of one another (quo); and the SemEval STS Benchmark dataset: 8,628 sentence pairs of varying similarity (sem).

5.2 Evaluation method

After fine-tuning models, we measured the model's performance on the three tasks using accuracy scores for sentiment analysis and paraphrase detection tasks, and Pearson correlation of the true similarity values against the predicted similarity values for the semantic textual similarity task. The overall score used to determine model performance was computed as `dev_sentiment_accuracy / 3 + dev_paraphrase_accuracy / 3 + (dev_sts_correlation + 1) / 6`. After each epoch of fine-tuning, we saved the model trained so far with the highest overall score. The predictions from models that achieved the highest overall dev scores were used for ensembling.

5.3 Experimental details

We ran our experiments on `nvidia-gpu-optimized-vmi-1-vm` VMs on GCP. Our experiments varied from 200-900 minutes.

For experiments using MNRL, we trained the BERT sentence embeddings using MNRL while varying the number of epochs (1, 3, 5, 10) and the scaling factor (10, 20, 40). After training the sentence embeddings using MNRL, we fine-tuned the model for 60 epochs on task-specific training datasets using annealed sampling and cosine similarity loss for STS. The entire process of training MNRL and fine-tuning on task-specific datasets took 800-900 minutes.

For LoRA and DoRA, we varied the LoRA rank (8, 16), LoRA α value (8, 16, 32), batch size (8, 16), learning rate (2e-4, 1e-5), and linear rate scheduler (none, linear). We also varied the set of minBERT linear layers replaced with linear LoRA (or DoRA) layers (all linear layers; query and key layers; query, key, and value layers). There was a 1.5-2x speedup over full fine-tuning, though scores using LoRA and DoRA did not surpass scores from full fine-tuning. For results shown in Table 1, the LoRA and DoRA models used a learning rate linear scheduler, LoRA rank of 16, and α of 32.

For each fine-tuning mode (full model, LoRA, or DoRA), we experimented with varying the number of epochs (10, 20, 40, 60), sampling technique (round-robin, square root, annealed), and learning rate (1e-3, 2e-4, 4e-4, 1e-5, 1e-6). With annealed sampling, we artificially up-weighted the probability of sampling from SST or STS datasets (by 2x, 4x, 8x) to offset the size disparity with the Quora dataset at the start of training. We attempted both adding or concatenating sentence embeddings for paraphrase detection and STS, as well as using a shared linear layer for those tasks. In lieu of combining embeddings for STS, we also ran experiments comparing embeddings using cosine similarity loss.

5.4 Results

Model	SST Acc	Paraphrase Acc	STS Corr	Overall Score
naive random baseline (expected value)	0.2	0.5	0	0.4
full model with MNRL and annealing	0.507	0.756	0.568	0.682
4-ensemble	0.532	**0.778**	**0.645**	0.711
5-ensemble	**0.545**	0.774	0.631	**0.712**

Table 1: Performance scores on test sets for different models. The full model with MNRL and annealing was our best-performing individual (non-ensembled) model on the dev leaderboard.

Results on the test set are shown in Table 1. Results on the dev set for a larger set of models we trained can be found in Appendix A.

Overall, the 5-ensemble outperforms all other models in terms of overall score, but the 4-ensemble is close in overall performance and outperforms the 5-ensemble in paraphrase detection accuracy and STS correlation. The 4-model ensemble was formed by combining results from three models trained using full fine-tuning, two of which incorporated MNRL and all of which used cosine similarity loss for STS; and one model trained using DoRA with a linear learning rate scheduler. All four models used annealed sampling. The 5-ensemble was formed by combining results from the models used in the 4-ensemble in addition to a model trained using LoRA.

Once we started using MNRL to pretrain models prior to full finetuning, we observed that the STS dev correlation significantly increased by $\approx$0.1-0.2 and paraphrase detection dev accuracy increased by $\approx$ 0.05-0.1. However, this came at a slight cost to the SST dev accuracy, which decreased by $\approx$ 0.001-0.005. We expected the improvement in STS and paraphrase detection performance, since using training data of positive examples from both tasks should improve the model sentence embeddings for these tasks and thereby enable the model to generalize well on the dev datasets. We did not expect the reduction in SST accuracy, but this could be attributed to the MNRL-trained sentence embeddings becoming more specialized for sentence similarity-type tasks like STS and paraphrase detection. Therefore, the final model was less adept at other kinds of tasks like SST.

Generally, we found that training for more epochs improved dev scores for the Quora task without negative impact on the SST and STS tasks, and that annealed sampling outperformed square root and round-robin sampling. We hypothesize that this is due to the imbalance in amount of training data; since the Quora dataset is 30-40x larger than the other datasets, round-robin downsampling results in the majority of Quora examples not being seen during training, hampering the model's ability to learn that task. After moving to annealed sampling, we observed faster improvement in the Quora task due to a higher proportion of training examples being selected.

We found that LoRA and DoRA models performed worse than expected compared to the full model, although applying LoRA or DoRA to only the query and value linear layers and using a higher learning rate of magnitude $e - 04$ (as the authors do in their experiments) greatly improved results compared to our initial experiments. We hypothesize that because LoRA and DoRA fine-tune a smaller set of parameters from scratch, unlike fine-tuning pretrained weights of the full model, when the dataset is not sufficiently large, we are unable to fully harness their potential. Based on our success in closing the gap between LoRA and DoRA with the full-model by tuning hyperparameters, we believe that further tuning the learning rate (e.g. using different schedulers) or applying LoRA or DoRA to different sets of layers may help in closing the gap. Ultimately, we decided to pursue fine-tuning the full model as we found it to be a better investment.

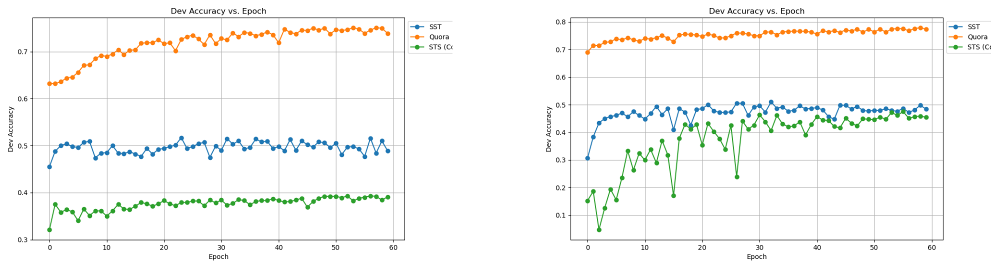
Between the PEFT methods, we observed that—after hyperparameter tuning —DoRA models outperformed LoRA models across all tasks, which is in alignment with the DoRA authors' claim that the DoRA formulation outperforms LoRA. We also found that both vanilla DoRA and LoRA outperformed the respective methods with an additional shared layer. This may be due to the same reasons expressed in our analysis of LoRA and DoRA's underperformance compared to full fine-tuning.

Ensembling with several combinations of our best-performing individual models produced the best results, giving improvements of close to 3% on overall score compared to non-ensembled models.

This was not surprising given that ensemble methods are known to reduce variance and prevent overfitting, leading to increased accuracy when combining predictions from multiple models.

6 Analysis

Figure 1 shows a comparison of dev scores from models using full fine-tuning over 60 epochs but with different sampling techniques. With round-robin sampling (a), the SST and STS scores plateau almost immediately while the accuracy on the Quora dataset increases much more slowly. This is reflective of dataset size; under round-robin, the model sees all of the STS data and most of the SST data but only a small fraction of the Quora data. Thus, generalizability improves slowly for the paraphrase detection task. This phenomenon motivated our implementation of annealed sampling (b), which takes training examples with probabilities based on dataset size, sampling heavily from the Quora dataset at the beginning of fine-tuning and slowly incorporating a higher percentage of SST and STS data over time. Consequently, we were able to achieve higher scores on Quora within a few epochs compared to round-robin, and the SST and STS scores took more epochs to level out. This approach also diversified the Quora examples that the model was exposed to.



(a) Round-robin sampling (b) Annealed sampling

Figure 1: Dev scores on fully fine-tuned models using different sampling techniques

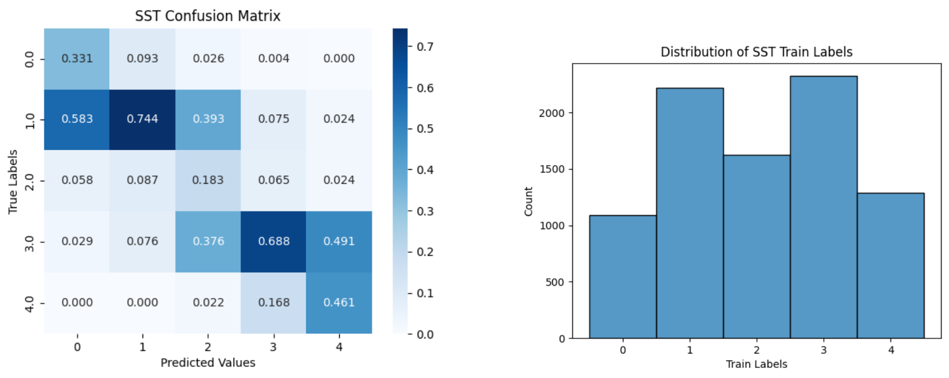
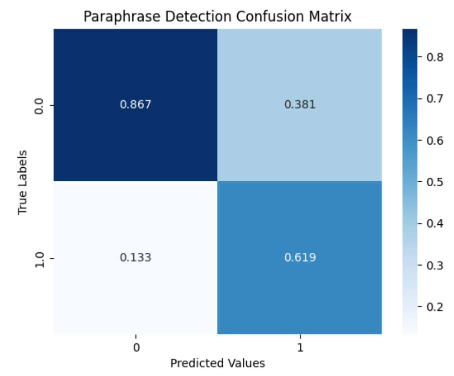


Figure 2: SST confusion matrix (left) and distribution of class labels in the SST train dataset (right).

For our final 5-ensemble model, we observed class imbalances in its predictions. These imbalances may have arisen from class imbalances inherent in the training data, which could shed light on the model's performance across each task. Figure 2 (left) shows that the 5-ensemble model accurately predicted examples with true labels of 1 and 3 more often than other labels. This result is in alignment with the distribution of SST train labels shown in Figure 2 (right) which contains more examples with labels 1 and 3. In Figure 3, we observe that the model exhibited a high true negative rate, but was less adept at predicting true positives. This behavior may be explained by the higher proportion of negative examples in the training data. In fact, negative examples are 2x as prevalent as positive examples in the train dataset. Based on these results for paraphrase detection, the 5-ensemble model achieves a precision of 0.619 and recall of 0.731, which aligns with the magnitude of the final test accuracy achieved and suggests that the model is more prone to predicting false positives than false



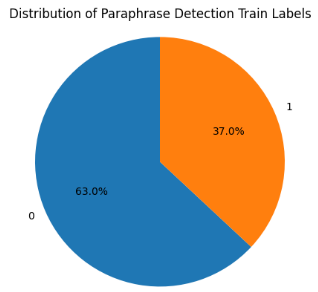
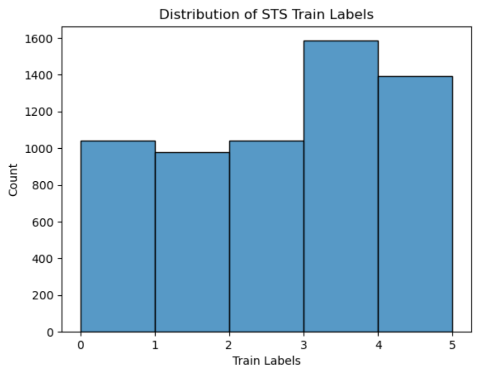


Figure 3: Paraphrase detection confusion matrix (left) and distribution of class labels in the paraphrase detection train dataset (right).



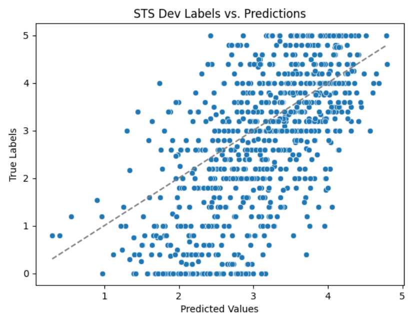


Figure 4: Scatterplot of predicted values on the STS dev dataset (left) and distribution of labels in the STS train dataset (right).

negatives. One potential mitigation for this is to include more positive examples in the training dataset, whether they be true or augmented examples. Figure 4 (left) demonstrates that the distribution of STS train labels is relatively even across labels 0-3, but higher for labels 3-5. This likely contributed to the denser concentration of the 5-ensemble model's predictions between labels of 2-4 as seen in Figure 4 (right). However, while the model does a modestly good job aligning 5-ensemble predicted values to true labels (i.e. $y = x$), we observe that it tends to overestimate similarity between sentences (a larger proportion of predictions are higher than true labels). This may be due to the fact that there are slightly more true labels between 3-4 and 4-5 than in any of the three lower buckets of labels.

7 Conclusion

In this work, we explored a wide range of techniques to fine-tune minBERT simultaneously on sentiment analysis, paraphrase detection, and semantic textual similarity tasks, achieving an improvement of over 30 percentage points on overall test scores compared to the baseline. The most effective techniques for higher multitask performance as identified by our ablations were annealed sampling, MNRL, cosine similarity loss, and ensembling. Limitations of our work include lack of research into generalization to downstream tasks beyond the three explored, such as commonsense reasoning, as well as development of truly novel methods.

For future work, more experimentation on feature engineering, such as adding input features representing named entity types in the data, could be beneficial for learning richer semantic representations. Varying regularization and optimization methods by using SMART (Jiang et al. (2020)) or Bregman proximal point optimization (Gutman and Peña (2021)) could also improve model generalization. Incorporating more datasets for fine-tuning, especially for sentiment analysis and semantic textual similarity, would be another natural extension to pursue for improving robustness and learning richer representations for those tasks.

8 Ethics statement

One ethical challenge is the potential for model bias. Since the datasets we used are scraped from web sources, e.g. the Quora dataset was scraped from Quora posts, it's possible that the datasets contain language that perpetuates stereotypes based on race, class, gender, etc. Since we used these datasets out-of-the-box (i.e. no pruning/preprocessing was done on the data beforehand), our models may therefore perpetuate and amplify such biases. For sentiment analysis, the data used to fine-tune the model may contain more negative reviews about a certain demographic, which the model may learn to associate negative sentiments with. At inference time, when applied to a new review that cites the same demographic, the model may be biased towards predicting a negative sentiment. If the same minBERT model is later applied to text generation, or — the more likely scenario — its learned embeddings are used for training or inferencing another language model, the model may generate hateful and toxic speech about specific demographics that are abusive and harmful for users. For paraphrase detection, examples from the dataset may contain harmful euphemisms or racial slurs that the model may learn to associate with certain groups or categories of people. If the learned sentence embeddings from the BERT model are then applied for a downstream task, e.g. rewriting or editing email drafts, as part of a larger ML system, then such toxic language may leak into the system's responses and cause psychological harm to users.

Another ethical concern is user privacy. Again, because the language datasets we used are scraped from web sources, it is possible that personally identifiable data like a person's name, phone number, or address may have been unintentionally incorporated into the collected dataset. Questions in the Quora dataset may include user information that the model incorporates in its internal feature representation. This can hinder the model's ability to generalize, since user-specific information may not be relevant to the task of paraphrase detection, but also, if the model is later applied to text generation, or its learned embeddings are used to train another model or are stored in a vector database as part of a larger system, sensitive information could be revealed in the system's outputs, which is a clear violation of user privacy. It is also possible that a malicious actor may attempt to extract personal information embedded in pretrained or fine-tuned data from the model.

Mitigation strategies include careful selection, preparation, analysis, and auditing of datasets to minimize bias, ensure that data is anonymized, and — in the case of sentiment analysis — ensure diversity of opinions. This can be achieved during data collection or training data preprocessing. One strategy is to first train or utilize a separate language model that classifies biased, hateful, or sensitive/personally identifiable text from a web scrape in order to filter/prune such examples from the final training dataset used to train or finetune a model for a specific downstream task. Transparency on how the data is collected is particularly important; datasheets and model cards provide helpful documentation for dataset curators and model developers to publish statistics on potential biases and sensitive topics baked into their datasets, and more broadly to ensure future replication of datasets used to train models.

References

Paraphrase identification on quora question pairs. https://paperswithcode.com/sota/paraphrase-identification-on-quora-question.

Semantic textual similarity on sts benchmark. https://paperswithcode.com/sota/semantic-textual-similarity-on-sts-benchmark.

Sentiment analysis on sst-5 fine-grained classification. https://paperswithcode.com/sota/sentiment-analysis-on-sst-5-fine-grained.

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

David H. Gutman and Javier F. Peña. 2021. Perturbed fenchel duality and first-order methods.

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

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. `https://github.com/microsoft/LoRA/tree/main`.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models.

Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Tuo Zhao. 2020. Smart: Robust and efficient fine-tuning for pre-trained natural language models through principled regularized optimization. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics.

Shih-Yang Liu, Chien-Yi Wang, Hongxu Yin, Pavlo Molchanov, Yu-Chiang Frank Wang, Kwang-Ting Cheng, and Min-Hung Chen. 2024. Dora: Weight-decomposed low-rank adaptation.

Sebastian Raschka. Improving LoRA: Implementing Weight-Decomposed Low-Rank Adaptation (DoRA) from Scratch. `https://magazine.sebastianraschka.com/p/lora-and-dora-from-scratch`.

Nils Reimers and Iryna Gurevych. 2019. 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.

Asa Cooper Stickland and Iain Murray. PyTorch implementation of BERT and PALs. `https://github.com/AsaCooperStickland/Bert-n-Pals`.

Asa Cooper Stickland and Iain Murray. 2019. BERT and pals: Projected attention layers for efficient adaptation in multi-task learning. *CoRR*, abs/1902.02671.

A Appendix

Part 1: Sentiment Analysis Results

Our minBERT implementation for the sentiment analysis task in part 1 achieves the following accuracies compared to the given baseline accuracies shown in parentheses:

Fine-tuning the last linear layer for SST: Dev Accuracy: 0.390 (0.390)

Fine-tuning the last linear layer for CFIMDB: Dev Accuracy: 0.780 (0.780)

Fine-tuning the full model for SST: Dev Accuracy: 0.519 (0.515)

Fine-tuning the full model for CFIMDB: Dev Accuracy: 0.967 (0.966)

Part 2: Multitasking Results[1]

[1]Model names have the format `{fine-tuning method}-{mnrl configuration, if used}-{embedding concatenation, if used}-{shared layer, if used}-{annealing steps per epoch, if used}-{number of epochs}-{learning rate}-{cosine similarity, if used}`, where non-applicable configurations are omitted.

Model	SST Acc	Paraphrase Acc	STS Corr	Overall Score
naive random baseline	0.198	0.505	0.004	0.402
last-linear-layer-10-0.001 baseline	0.337	0.651	0.261	0.539
full-model-10-1e-5 baseline	0.503	0.684	0.356	0.622
full-model-concatembed-10-1e-05	0.508	0.680	0.351	0.621
full-model-concatembed-20-1e-05	0.506	0.711	0.368	0.634
full-model-concatembed-40-1e-05	0.518	0.735	0.373	0.646
full-model-concatembed-60-1e-05	0.516	0.746	0.393	0.653
full-model-anneal1800-10-1e-05	0.488	0.743	0.363	0.638
full-model-anneal3600-20-1e-05	0.515	0.760	0.381	0.655
dora-anneal1800-concatembed-60-2e-04	0.478	0.726	0.378	0.631
lora-concatembed-sharedlayer-80-4e-04	0.490	0.662	0.377	0.613
lora-concatembed-80-4e-04	0.490	0.680	0.359	0.617
full-model-anneal3600-60-1e-05-cos	0.498	0.779	0.459	0.669
full-model-mnrladddata3-anneal3600-60-1e-05–cos	0.486	0.779	0.517	0.674
full-model-mnrladddata3-anneal3600-60-1e-05-16–cos	0.500	0.775	0.498	0.675
dora16a32qv-anneal3600-concatembed-35-0.0002lrlinsched-16	0.480	0.746	0.384	0.639
3-ensemble	0.508	**0.785**	0.648	0.706
4-ensemble	0.518	0.780	**0.664**	**0.710**
5-ensemble	**0.519**	0.776	0.651	0.707

Table 2: Performance scores on dev sets for different models.