minBERT and Downstream Tasks

Stanford CS224N Default Project

Reva Agashe
Department of Computer Science
Stanford University
revaagashe@stanford.edu

Jennie Chung
Department of Computer Science
Stanford University
jech@stanford.edu

Regina Wang
Department of Computer Science
Stanford University
rlwang@stanford.edu

Abstract

The ability to perform well on multiple tasks with one model is highly desirable because it allows for less computation and improved performance across a number of useful language-related tasks. We use multi-task learning and multitask fine-tuning on three tasks (paraphrase detection, sentiment analysis, semantic textual similarity) to improve minBERT's performance for each individual task. We find that using cosine similarity for understanding semantic textual similarity, freezing embedding layers, and using multiple negatives ranking loss further improve the performance of minBERT on these tasks. Furthermore, we have conducted extensive discussions to explain the performance of our model and its extensions.

1 Key Information

We have no external collaborator or external mentor, and we are not sharing this project with any other course.

2 Introduction

The past decade has witnessed significant advancements in the field of Natural Language Processing (NLP), especially with the emergence of deep learning and innovative neural network models. Among these models, the Transformer architecture has gained considerable attention due to its outstanding performance on various NLP tasks, such as language modeling (Devlin et al., 2018), question-answering (Pearce et al., 2021), and text classification (Sun et al., 2019).

The Transformer model, introduced by Vaswani et al. (2017), is a self-attention-based neural network architecture that processes sequential data, such as text, in parallel. Unlike previous models that relied on recurrent neural networks (Hochreiter and Schmidhuber, 1997) or convolutional neural networks (O'Shea and Nash, 2015), the Transformer model leverages self-attention mechanisms to capture long-range dependencies between the input tokens, enabling it to learn more effective representations of text data.

However, despite their success, the large size and computational complexity of Transformer models make them challenging to deploy in resource-constrained environments. To address this issue, several groups have developed lightweight versions of the Transformer model, such as DistilBERT (Sanh et al., 2019), TinyBERT (Jiao et al., 2019), and minBERT, that maintain similar performance while significantly reducing the computational cost. In minBERT's case, this pared-down model was developed for educational purposes.

While these models have shown remarkable performance on various NLP tasks, the ability to perform well on multiple tasks with one model is highly desirable. This could simultaneously allow for less computation and improved performance across a number of useful language-related tasks. Multi-task learning is one popular approach that enables a model to learn multiple tasks simultaneously by sharing parameters across the tasks (Liu et al., 2019a).

In this paper, we focus on improving the performance of minBERT, a lightweight version of the Transformer model, on three important NLP tasks: paraphrase detection, sentiment analysis, and semantic textual similarity. We employ multi-task learning and multitask fine-tuning to enhance the performance of minBERT on each task.

To further improve the performance of minBERT on these tasks, we examine a combination of multiple negatives ranking loss learning Henderson et al. (2017), cosine similarity for sentiment and paraphrase prediction (Reimers and Gurevych, 2019), and freezing of minBERT model layers. Multiple negatives ranking loss learning is used to optimize the model to distinguish between positive and negative examples, while cosine similarity-based prediction leverages these improved embeddings. We also leverage machine learning tools to reduce catastrophic forgetting and overfitting.

Our experimental results show that our proposed method significantly improves the performance of minBERT on all three tasks. We believe that our approach has the potential to be applied to other language models and tasks, contributing to the development of more efficient and effective models for natural language processing.

3 Related Work

Multi-task learning has shown promising results in the context of text classification (Sun et al., 2019) (Gao et al., 2021) (Liu et al., 2019b). By pretraining BERT on a well-chosen dataset and then fine-tuning subsequent layers for each of the tasks, their model is able to improve embeddings using a larger number of training examples while also adjusting weights in the task-specific layers.

Previous work has also shown that further pre-training and multitask fine-tuning can also potentially improve BERT's performance on a variety of tasks. Sentiment analysis, question classification, and topic classification were the target tasks in Sun et al. (2019). In this paper, fine-tuning involves identifying the BERT layers which are most important for a target task's performance, and then training these layers on target task data. It may also involve adjusting the optimization algorithm or learning rate for these layers. The authors found that fine-tuning on the last BERT Transformer layer and fine-tuning on the last four BERT Transformer layers improved model performance, suggesting that unfreezing later pretrained layers in minBERT can help target task accuracy.

Multi-task learning with language models may also be improved by directly improving the BERT embeddings using domain-relevant data. In Henderson et al. (2017), Multiple Negatives Ranking Loss is used for this purpose. Multiple Negatives Ranking Loss takes as input a batch of sentence pairs $(x_1, y_1), (x_1, y_2), \ldots, (x_n, y_n)$, where for all $i \in \{1, \ldots, n\}$, (x_i, y_i) is a positive pair and for all $i \neq j$, (x_i, y_j) is a negative pair. Then, the loss is computed as

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

where $S(x_i, y_j)$ is the score for a pair of inputs and responses in a training batch of n examples. Additionally, θ represents the word embeddings and neural network parameters used to calculate S. Most commonly, cosine similarity between embedding vectors is used as the similarity function.

4 Approach

4.1 Baseline

Our baseline for the three tasks is the minBERT model. minBERT is a minimalist implementation of the original BERT model first developed at Carnegie Mellon University. minBERT first splits

each input sentence into words before splitting those words into word pieces using a WordPiece tokenizer. Each token is then converted into an identification number (ID). minBERT consists of an embedding layer and 12 encoder transformer layers. The model outputs the embeddings for each word piece and token. We designed a new prediction function for each of the three tasks: sentiment analysis, paraphrase detection, and semantic textual similarity. For each prediction, we call the minBERT model to obtain the token embeddings for the task's sentence inputs and add a Dropout layer with dropout probability of 0.3. Since paraphrase detection and semantic textual similarity take in pairs of inputs, we concatenate the embeddings for each pair. We then apply a final fully connected linear layer to obtain logits for each of the prediction tasks. We trained the model for all three tasks on the Stanford Sentiment Treebank (SST) Dataset. We used Cross Entropy Loss for sentiment analysis, Mean Squared Error Loss for semantic textual similarity, and Binary Cross Entropy Loss for paraphrase detection. We also used the AdamW optimizer (Loshchilov and Hutter, 2017) to perform gradient descent and update the parameters.

4.2 Extensions

In the following section, we discuss various extensions which we made to the minBERT model in order to improve performance on the three tasks. Each extension is referred to using a specific acroynym. To describe the baseline model, which simply trains the minBERT model on the SST data for 10 epochs without any other extensions or adjustments, we use the acronym BL. For the sentiment and paraphrase tasks, we investigate using cosine similarity for prediction (COS-S and COS-P, respectively). Since multi-task learning inherently means dealing with different-sized datasets, we compare training using epochs lengths that are determined by the largest dataset (TM1) and the smallest dataset (TM2). We explore the effect of freezing the earlier layers in minBERT to partially retain parameters learned from pretraining, taking inspiration from Sun et al. (2019). We chose to freeze all layers except the last two minBERT layers (FL2) or the last four minBERT layers (FL4). Finally, we explore using the Multiple Negatives Ranking Loss to improve the embeddings using a target task dataset (MNR). We discuss each of these approaches more in-depth below.

4.2.1 Cosine similarity for predicting similarity (COS-S) and paraphrase detection (COS-P)

Both the similarity and paraphrase predictions functions takes in as input a pair of sentences and attempt to judge the similarity between those two sentences. Cosine similarity is a function which outputs a measure of how aligned two embeddings are in vector space. With this intuition, we decided to explore how incorporating cosine similarity in the prediction for these two tasks would affect model performance. We ran two different experiments here, using cosine similarity for only one task at a time.

For the similarity task (COS-S), we find the cosine similarity between the embeddings of each of the input sentences. Then, we apply the ReLU activation function (Agarap, 2018) and scale our output by a factor of 5 for scaling purposes.

For the paraphrase detection task, the cosine similarity is fed into a sigmoid function $\sigma(x)$ (Dubey et al., 2022), which is then treated as the probability of the positive class in the binary cross entropy loss function.

4.2.2 Training on multiple datasets (TM1, TM2)

The baseline model only trained on data from the Stanford Sentiment Treebank Dataset (SST), which is used for the sentiment analysis task. To improve predictions for the paraphrase detection and semantic textual similarity tasks, we modified our training function to also train on examples from the Quora and Semantic Textual Similarity (STS) datasets.

We modified our training function in our multitask classifier to calculate the losses across the predictions for all three tasks and then backpropagate on the sum of the three losses. We considered two implementations.

In our first training method (TM1), we noticed the disparity in dataset sizes. The Quora dataset has about 40 times the number of examples as the other two datasets, so the paraphrase detection task has about 40 times the number of batches than the other two tasks. In this training method, we interleaved

training batches between all three tasks in every epoch. To get through every batch in the Quora dataset, we cycled through the SST and STS datasets until the Quora dataset was finished.

When running experiments with the above approach, we noticed overfitting on the paraphrase detection task. In our second training method (TM2), we again consider one batch from each of the three datasets. In every epoch, we instead process only the number of batches in the smaller STS dataset, cutting off training on the datasets for the other two tasks. In the next epoch, training for the other two tasks would pick up on the next unseen batch.

4.2.3 Freeze layers in minBERT model (FL2, FL4)

The original minBERT model contains 12 encoder transformer layers, all of which are trained and receive updates when using the 'finetune' configuration during multitask training. Finetuning generally results in better performance than pretraining (which 'freezes' all 12 layers). We experimented with finding a middle ground between the two options, with the intention of preventing the model from overfitting on the training sets. By freezing the parameters in all but the last n encoder transformer layers, only those last n layers along with the linear layers called by the prediction functions would get trained. This ensures that after training, the new model won't deviate too much from the pretrained model. The method FL2 froze parameters in all but the last two layers and the method FL4 froze parameters in all but the last four layers.

4.2.4 Multiple Negatives Ranking Loss (MNR)

We implemented a new loss function, described in Efficient Natural Language Response Suggestion for Smart Reply Henderson et al. (2017). The dataset used for this additional pretraining step was all positive examples from the STS dataset. We minimized the mean negative log probability for the training data, given by Equation (1).

We used a scaling parameter of 20.0, as commonly used in other implementations. This embedding improvement step was trained before the finetuning on the other three tasks.

5 Experiments

5.1 Data

For the sentiment analysis task, we used the Stanford Sentiment Treebank (SST) dataset (Socher et al., 2013). For SST, the dataset maps single sentences extracted from movie reviews to a categorical label between 0 and 4, where 0 is negative sentiment and 4 is positive sentiment. This dataset has 11,855 labeled sentences.

For the paraphrase detection task, we are using the Quora dataset (Iyer et al., 2016). This has 400,000 question pairs mapped to a binary label describing if the texts are paraphrases of each other.

Finally, for the semantic textual similarity task, we are using the SemEval STS Benchmark dataset, which maps pairs of sentences to a similarity score between 0 and 5 (Agirre et al., 2013). Crucially, this dataset is not categorical. This dataset has 8,628 training pairs.

5.2 Evaluation method

For both the sentiment analysis and paraphrase tasks, we used accuracy on the dev set or test set to evaluate our model's predictions. To evaluate on the semantic textual similarity task, we calculated the Pearson correlation between the true and predicted similarities.

5.3 Experimental details

For all tasks, we ran our model for 10 epochs using a batch size of 64, a dropout of 0.3, and a learning rate of 1e-5.

For reweighting losses, we found mixed results depending on which task was being evaluated. After training on all datasets, we found that the model's performance on the paraphrase detection task suffered compared to the other two tasks. At that point, the losses for the three training steps had been

combined by simply summing them together. To mitigate the low performance for the paraphrase task, we reweighted our losses. We multiplied the loss of each task by the ratio of its corresponding dataset's size to the sum of all dataset sizes. This improved the results of the paraphrase detection task, but the model's performance worsened for each of the other two tasks.

6 Results

6.1 Test Set Leaderboard

We submitted our predictions from our model M10. As of our writing of this paper, our approach is at rank 28 of 55.

Model#	Method	SST Accuracy	Paraphrase Accuracy	STS Correlation	Overall Test Score
M7	BL + TM1 + COS-S + FL4	0.501	0.781	0.642	0.641

Table 1: Test set results with our model that performed the best on the dev set, M7. BL is baseline, TM2 is second training method, COS-S is cosine similarity for STS, FL2 is freezing layers. The Overall Test Score is the average of the three other metrics.

6.2 Dev Set Results

Model#	Method	SST Accuracy	Paraphrase Accuracy	STS Correlation	Overall Dev Score
M0	BL	**0.528**	0.530	-0.270	0.262
M1	BL + TM1	0.495	0.778	0.369	0.547
M2	BL + TM1 + MNR	0.500	0.770	0.392	0.554
M3	BL + TM1 + COS-S	0.500	0.778	0.599	0.626
M4	BL + TM1 + COS-S + COS-P	0.494	0.584	0.599	0.559
M5	BL + TM1 + COS-S + MNR	0.503	0.771	0.610	0.628
M6	BL + TM1 + COS-S + FL2	0.435	0.763	**0.677**	0.625
M7	BL + TM1 + COS-S + FL4	0.452	**0.779**	0.674	**0.635**
M8	BL + TM2	0.494	0.718	0.348	0.520
M9	BL + TM2 + COS-S	0.501	0.697	0.626	0.608
M10	BL + TM2 + COS-S + FL2	0.439	0.670	0.600	0.570
M11	BL + TM2 + COS-S + FL4	0.473	0.683	0.647	0.601
M12	BL + TM2 + COS-S + RW	0.459	0.743	0.549	0.584

Table 2: Development set results. BL is baseline, TM is training on multiple datasets, COS-S is cosine similarity for STS, COS-P is cosine similarity for PARA, FL2 and FL4 are freezing layers, RW is reweighting losses, and MNR is multiple negatives ranking loss. The Overall Dev Score is the average of the three other metrics.

6.3 Quantitative Discussion

The results of all of our experiments are displayed in Table 1 and Table 2. Based on evaluation on the three dev sets, we found that our best model was M7 (BL + TM1 + COS-S + FL4), which achieved the following performance on the dev set: SST: 0.452, Quora: 0.779, STS: 0.674.

We see that training each model on all three datasets improved performance dramatically, when compared against the baseline model (M0). While SST accuracy decreased only slightly, each model did significantly better than the baseline paraphrase detection accuracy and STS correlation scores.

We find that TM1 outperforms TM2. When comparing M1 with M8, and M3 with M9, we notice that performance on the paraphrase detection task drops by 0.06 and 0.081 respectively, while performance on SST and STS show no significant change on average. This is surprising; TM2 was built to address overfitting on the SST and STS training datasets, as each batch was seen by an approximate factor of 40 times more than each batch from the Quora train set when using TM1.

We note that using cosine similarity for STS was also beneficial for the model's performance. Comparing M3 with M1, we see that when using Training Method 1, replacing the linear layer when predicting similarity scores with a cosine similarity function improved correlation on the development set from 0.369 to 0.599. As expected, this didn't affect the accuracies on the SST or Quora datasets. Comparing M9 with M8 shows us that this modification had similar results when using TM2.

Given the success of using cosine similarity for STS (seen in M3 and M9 results), we also tried using cosine similarity for paraphrase detection (seen in M4). Surprisingly, with this approach, STS and SST scores remained around the same, but the accuracy for paraphrase detection dropped drastically (from 0.778 to 0.584). We explore possible explanations for this in the next section.

Comparing M2 to M1 shows the impact of using multiple negatives ranking (MNR) loss. We note a slight decrease of 0.008 in paraphrase detection accuracy, and a larger increase of .023 in STS correlation scores. Similarly, implementing MNR on top of M3 confirms only slightly better performance on the SST and STS tasks, while paraphrase detection accuracy drops for M5 by 0.007.

Furthermore, we find that unfreezing the last two embedding transformer layers slightly decreases overall performance, but unfreezing the last four layers improves performance significantly. When training two layers, we notice an overall dev score 0.0308 lower for M10 than for M9, and a dev score 0.01 lower for M6 than for M3. However, while all three tasks suffer from the addition of freezing layers when training the model with TM2, the STS correlation improves by 0.078 when training the model with TM1. We hypothesize that this is because the model did overfit on the STS training dataset in TM1, and freezing the first 10 transformer layers helps account for this. Surprisingly, freezing only the first 8 transformer layers results in far better performance. When training our model with TM1, we see significant improvement in overall performance for M7 when compared to both M6 and M3. When using TM2, however, we note that M11 results in better performance than M10 for each of the three tasks, but both fall short when compared to M9, the model in which no layers are frozen.

Finally, comparing M12 to M9 allows us to confirm that reweighting the loss (to consider the paraphrase detection task loss much more than the losses of the SST and STS tasks) results in worse overall dev performance, but improved paraphrase detection significantly.

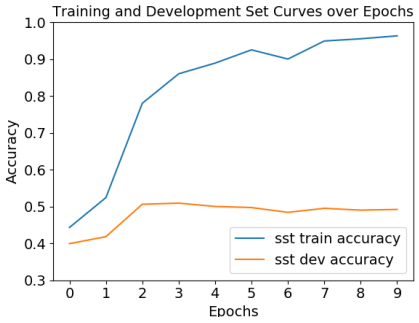
7 Analysis

7.1 Overfitting

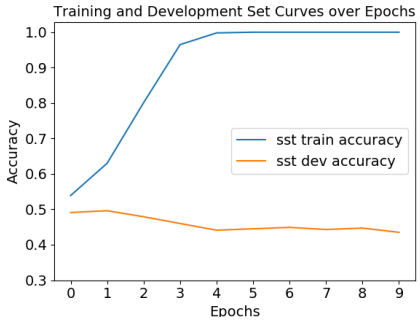
One key insight of our work was the large effect that experimental setting and training decisions made on overall model performance. Our first training method (TM1), where the training loop goes through every batch in the Quora dataset while cycling through the other two task datasets, resulted in heavy overfitting for the sentiment predictor. This can be seen from the training set and development set curves in Figure 1a–there is a large gap between the training and development set accuracies at each epoch, and this discrepancy worsens with further training.

We made two attempts to mitigate the effect of overfitting. First, when we trained our model with TM2 instead of TM1, the overfitting problem improved significantly, as seen in Figure 1c. Overall training and development set accuracy was slightly better, and due to the smaller gap between the two accuracies, we expect better generalization to unseen data. Comparing Figure 1d to Figure 1b confirms this; the discrepancy decreased significantly after switching to TM2.

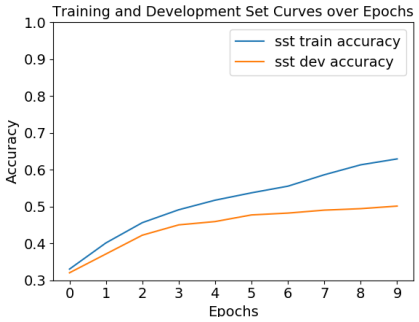
For our second method, we found that implementing freezing earlier minBERT layers did not have a significant impact on overfitting. Figure 1b shows us that the gap between the training and development curves continued to increase over epochs, and the model ended with a larger discrepancy when compared to Figure 1a. Looking at Figure 1d, however, shows us that freezing layers did slightly improve upon the discrepancy seen in Figure 1c. This may be due to the compounded effect of using TM2 with freezing layers. Furthermore, it should be noted that SST accuracy decreased significantly for all models (M6, M7, M10, and M11) where freezing layers was implemented, when compared to their counterparts (M3 and M9). Due to the small number of examples in our training corpus compared to the large size of the Huggingface data that our model was pretrained on, we hypothesize that catastrophic forgetting was not actually an issue that our model was facing.



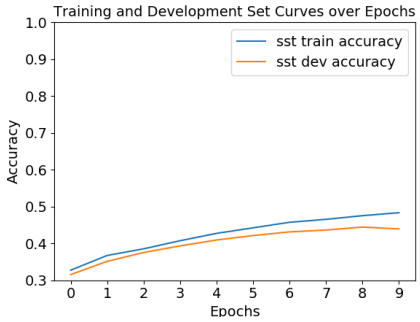
(a) SST learning curves for M3 (BL + TM1 + COS-S).



(b) SST learning curves for M6 (BL + TM1 + COS-S + FL2).



(c) SST learning curves for M9 (BL + TM2 + COS-S).



(d) SST learning curves for M10 (BL + TM2 + COS-S + FL2).

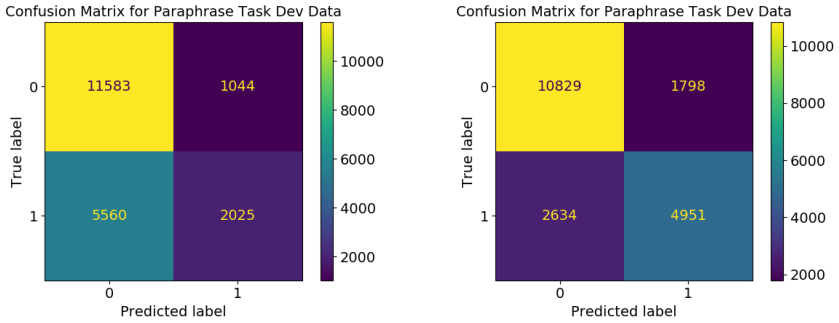
Figure 1: Training and development set learning curves for accuracy vs. epoch for SST data and the effect of our overfitting reduction methods.

7.2 Confusion Matrix Comparison

We also observed that for the paraphrase task using the model M10, designed to prevent overfitting, there were far more false negative predictions than false positive predictions. In other words, our model often mistakenly predicted that two sentences were not paraphrases when the sentences actually were paraphrases. This error was much more common than the converse. We can observe this from Fig. Figure 2a. However, we note that our model which uses Multiple Negatives Ranking Loss slightly improves the model by decreasing the number of false negatives and only slightly increasing the number of false positives, therefore balancing the two error types. This suggests that changing the model weights to train and adjust to optimize STS task performance results in changes to the minBERT layers which may affect other tasks. In other words, both the internal minBERT layer weights as well as the layer weights affect performance on any given task, and training for more iterations on one task may results in tradeoffs when it comes to performance on other tasks.

7.3 Cosine similarity for paraphrase detection

In the previous section, we mentioned that using cosine similarity for paraphrase detection decreased paraphrase performance drastically. This was extremely surprising considering that using cosine similarity for the STS task increased overall performance quite a bit. We hypothesize that this is due to the different nature and implementation for the two tasks. Sentiment prediction is a categorical task, while paraphrase prediction is a regression task. This led to the different implementations of cosine similarity prediction detailed in the previous section. The difference in treatment, where cosine similarity for paraphrase is used to inform a probability distribution instead of a numerical prediction, may be the root cause of the discrepancy. Since the cosine similarity of two embeddings will be in the range $[-1, 1]$ and $\sigma(0) = 0.5$, any two examples which have a cosine similarity greater than zero will be predicted to be a positive paraphrase pair. This is incongruent with the definition of a paraphrase, which is two sentences that are extremely similar (we would want the cosine similarity of



(a) Confusion Matrix for Paraphrase task, model identifier M10

(b) Confusion Matrix for Paraphrase task, model identifier M5

Figure 2: Confusion matrices for two methods to prevent overfitting.

two sentences to be very high before a paraphrase is predicted). To examine this hypothesis further, future work could leverage this intuition through experimenting on ways to manipulate the cosine similarity before the sigmoid application.

8 Conclusion

In this study, we used a wide range of multitask learning methods and extensions of the pretrained minBERT model to achieve impressive results across three critical NLP tasks: sentiment analysis, paraphrase detection, and semantic textual similarity. Our findings indicate that training on individual datasets for each task can considerably boost overall model performance, and that out of our extensions, a combination of BL, TM1, COS-S, and FL4 perform the best. Building on top of the baseline model, we train on multiple data sets by iterating through paraphrase data and cycling through STS and SST data, using cosine similarity for STS, and freezing all layers in the minBERT model excluding the last four encoder layers. On the test set, this model achieved an SST accuracy of 0.501, a paraphrase accuracy of 0.781, and a STS correlation of 0.642, for an overall test score of 0.642. Furthermore, we have conducted extensive discussions to explain the performance of our model and its extensions.

Moreover, we discovered that when training by iterating through the paraphrase set at each epoch and cycling over the SST and STS datasets, unfreezing the final four minBERT layers in the model improved performance for the STS task. Specifically, by unfreezing the final four layers and allowing them to learn weights based on specific tasks, we found that STS correlation increased from 0.599 to 0.677. This suggests that finding a middle ground between pretraining and finetuning is beneficial to downstream multitask performance.

Finally, to address the issue of different sized datasets, we sought methods to mitigate the effect of looking at one dataset for a specific task too often. In particular, for our data, we found that the Quora dataset used for the paraphrase task was significantly larger than the other two training sets for the STS and SST tasks, resulting in over 2200 batches when using batch sizes of 64. To reduce the effect of biasing our model toward one specific task, we use two different ways of training on multiple datasets: (1) TM1, in which we trained over the paraphrase dataset once per epoch and cycled through the SST and STS datasets, and (2) TM2, in which we avoided over-cycling through the smaller datasets by only running through the number of batches dictated by the smallest dataset, the STS dataset. This helped mitigate overfitting, as shown by Figure 1.

Despite these encouraging results, one drawback of our study is the specificity of the methods used for the tasks. It remains uncertain whether these performance improvements can be replicated with other NLP tasks or datasets. This issue can be explored by using transfer learning evaluation, where we evaluate performance on previously unseen tasks. Additionally, we are interested in further experimenting with combined extensions, e.g. evaluating the performance of multiple negatives ranking loss when certain transformer layers are frozen. Due to resource and choice constraints for this project, it may be that our initial results for each model could be improved with further hyperparameter tuning or training for longer periods.

References

Abien Fred Agarap. 2018. Deep learning using rectified linear units (relu). In *CoRR*, volume abs/1803.08375.

Eneko Agirre, Daniel Cer, Mona Diab, Aitor Gonzalez-Agirre, , and Weiwei Guo. 2013. Semantic textual similarity. In *Second joint conference on lexical and computational semantics*, pages 32–43.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. In *arXiv preprint arXiv:1810.04805*.

Shiv Ram Dubey, Satish Kumar Singh, and Bidyut Baran Chaudhuri. 2022. Activation functions in deep learning: A comprehensive survey and benchmark. In *Neurocomputing*, pages 92–108.

Tianyu Gao, Xingcheng Yao, and Danqi Chen. 2021. SimCSE: Simple contrastive learning of sentence embeddings. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 6894–6910, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Matthew Henderson, Rami Al-Rfou, Brian Strope, Yun hsuan Sung, Laszlo Lukacs, Ruiqi Guo, Sanjiv Kumar, Balint Miklos, and Ray Kurzweil. 2017. Efficient natural language response suggestion for smart reply.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computation*, 9(8):1735–1780.

Shankar Iyer, Nikhil Dandekar, and Kornél Csernai. 2016. First quora dataset release: Question pairs.

Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. 2019. Tinybert: Distilling bert for natural language understanding. In *In proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Xiaodong Liu, Pengcheng He, Weizhu Chen, and Jianfeng Gao. 2019a. Multi-task deep neural networks for natural language understanding. In *Association for Computational Linguistics*.

Xiaodong Liu, Pengcheng He, Weizhu Chen, and Jianfeng Gao. 2019b. Multi-task deep neural networks for natural language understanding.

Ilya Loshchilov and Frank Hutter. 2017. Fixing weight decay regularization in adam. *CoRR*, abs/1711.05101.

Keiron O'Shea and Ryan Nash. 2015. An introduction to convolutional neural networks. *CoRR*, abs/1511.08458.

Kate Pearce, Tiffany Zhan, Aneesh Komanduri, and Justin Zhan. 2021. A comparative study of transformer-based language models on extractive question answering. In *arXiv preprint arXiv:2110.03142*.

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 and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, page 3982–3992.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. In *Conference and Workshop on Neural Information Processing Systems*.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, , and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *In Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1641.

Chi Sun, Xipeng Qiu, Yige Xu, and Xuanjing Huang. 2019. How to fine-tune bert for text classification?

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