Multitask minBert

Stanford CS 224N Default Project
Mentor: Tathagat Verma

Riya Sankhe
riyas@stanford.edu

Sajel Galhotra
galhotra@stanford.edu

Emma Passmore
emmapass@stanford.edu

Abstract

BERT is a state-of-the-art, pretrained language representation model that can be fine-tuned to perform well on a variety of natural language processing tasks (Devlin et al., 2018). Our goal is to build a multitask BERT model that can simultaneously perform well on three sentence-level tasks: sentiment classification, paraphrase detection, and semantic textual similarity. The three main techniques we explore are further pretraining, single-task fine-tuning, and multitask fine-tuning. We find that (1) only multitask fine-tuning outperforms only single-task fine-tuning, (2) additional pretraining does not significantly improve multitask model performance, (3) adding dense task-specific layers improves model performance, and (4) single-task fine-tuning after multitask fine-tuning improves model performance on tasks with comparatively low training performance.

1 Introduction

BERT can be fine-tuned to create state-of-the-art models for a wide range of natural language processing tasks without substantial task-specific architecture modifications. This paper investigates how different fine-tuning and further pretraining techniques affect the performance of our multitask model on three tasks: sentiment classification (SST), paraphrase detection (Para), and semantic textual similarity (STS). SST involves analyzing a piece of text to determine its emotional tone, Para involves analyzing two pieces of text to see if they are paraphrases of each other, and the SST task aims to measure the degree of semantic similarity of two input texts. In this project, we explore the effectiveness of three main techniques: (1) further pretraining the BERT-base-uncased embeddings on both within-task and in-domain corpora, (2) multitask fine-tuning our model using different numbers of task-specific dense layers, and (3) single-task fine-tuning task-specific layers after multi-task fine-tuning.
Our approach is novel for two main reasons. First, most existing studies that implement multitask fine-tuning only focus on a single task with multiple corpora (Conneau et al. (2020); Wang et al. (2009)) or multiple tasks with a single corpus. Here, we focus on multitask fine-tuning for multiple tasks with multiple corpora. Second, we compare different fine-tuning and further pretraining techniques on models that share BERT Encoder Transformer layers across multiple tasks, which few studies do.
We find that further pretraining does not significantly improve multitask model performance unlike studies on single-task BERT models (Sun et al., 2019). On the other hand, multitask fine-tuning followed by single-task fine-tuning significantly improves model performance on the three target tasks.

2 Related Work

Krishna et al. (2022) and Sun et al. (2019) find that within-task and in-domain further pretraining BERT embeddings improve model performance on NLP tasks. Krishna et al. (2022) specifically find that pretraining and fine-tuning on the same datasets achieves similar performance to pretraining on massive datasets (eg. Wikipedia). In this paper, we explore whether these results hold when fine-tuning for multiple tasks simultaneously (see Section 5.2).
Liu et al. (2019) find that multitask fine-tuning improves model performance and Sun et al. (2019) find that single-task fine-tuning after multitask fine-tuning further improves performance. Multitask fine-tuning leverages labeled data from related tasks to strengthen the model's understanding of vocabulary and, more importantly, a wider range of complex syntactic structures. Leveraging this in-domain data is important because within-task labeled data is often hard to obtain and includes a more limited selection of syntactic structures. In addition, multitask models have a regularization effect that reduces the tendency of a network to overfit a specific task, which in turn improves the generalizability of the learned representations across tasks (Liu et al., 2019). This generalizability helps the model perform well on multiple tasks simultaneously. Thus, we experiment with multitask fine-tuning in isolation and before single-task fine-tuning (see Section 5.1).

3 Approach

3.1 minBERT Architecture

As described by Vaswani et al. (2017), minBERT consists of an embedding layer followed by 12 Encoder Transformer layers. Each Encoder Transformer layer consists of multi-head attention, followed by an add and norm layer (a residual connection followed by layer normalization), a feed-forward layer, and a final add and norm layer.

3.2 Task-Specific Dense Layers

We extended the minBERT model by adding private task-specific layers to the minBERT architecture.

- **Sentiment Analysis Task Architecture**: Let $\boldsymbol{x}$ be the contextualized embedding of the `[CLS]` token for X, an input sequence from a sentiment classification dataset (eg. `cfimdb`). $\boldsymbol{x}$ aims to encode the semantic meaning of X. Dropout is applied to $\boldsymbol{x}$. The unnormalized prediction that X is labeled as sentiment c is determined by two linear layers with GELU:

$$\hat{\boldsymbol{y}} = \boldsymbol{W}_2^\top GELU(\boldsymbol{W}_1^\top \text{dropout}(\boldsymbol{x})) \tag{1}$$

  where $\boldsymbol{W}_1^\top$ and $\boldsymbol{W}_2^\top$ are sentiment task parameter matrices.

- **Paraphrase Detection Task Architecture:** Let $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ be the contextualized embeddings for the `[CLS]` tokens of X_1 and X_2, a question-pair from a paraphrase dataset (eg. Quora). $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ aim to encode the semantic meaning of X_1 and X_2. Dropout is applied to $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$. Then, $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ are concatenated. The unnormalized prediction that X_1 and X_2 are classified as paraphrases of each other is determined by two linear layers with GELU:

$$\hat{\boldsymbol{y}} = \boldsymbol{W}_3^\top GELU(\boldsymbol{W}_4^\top [\text{dropout}(\boldsymbol{x}_1); \text{dropout}(\boldsymbol{x}_2)]) \tag{2}$$

  where $\boldsymbol{W}_3^\top$ and $\boldsymbol{W}_4^\top$ are paraphrase task parameter matrices.

- **Semantic Textual Similarity Task Architecture:** Let $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ be the contextualized embeddings for the `[CLS]` tokens of X_1 and X_2, two sequences from a semantic textual similarity dataset (eg. SemEval STS Benchmark). Dropout is applied to $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$. Then, $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ are concatenated with $|\boldsymbol{x}_1 - \boldsymbol{x}_2|$ as in Reimers and Gurevych (2019). The unnormalized prediction of the semantic similarity between X_1 and X_2 is determined by two linear layers with GELU:

$$\hat{\boldsymbol{y}} = \boldsymbol{W}_6^\top GELU(\boldsymbol{W}_5^\top [\text{dropout}(\boldsymbol{x}_1); \text{dropout}(\boldsymbol{x}_2); |\text{dropout}(\boldsymbol{x}_1) - \text{dropout}(\boldsymbol{x}_2)|]) \tag{3}$$

  where $\boldsymbol{W}_5^\top$ and $\boldsymbol{W}_6^\top$ is a STS task parameter matrix.

3.3 Multitask Fine-Tuning

Multitask Architecture Pretrained embeddings are fed into the minBERT model (see Section 3.1). Then, the contextualized [CLS] token embedding from the final BERT Encoder Transformer layer is passed into the task-specific dense layers to obtain unnormalized predictions as described in Section 3.2.

Multitask Fine-Tuning Procedure To update the parameters of both the shared and task-specific layers, we use a modified Adam optimizer based on findings from Loshchilov and Hutter (2017) and Kingma and Ba (2014), where we compute the bias correction directly instead of first computing its first moment and second raw moment estimate.[1]. In each epoch, we iterate through the data for the Para, SST, and STS datasets and update each model according to its task-specific objective. See Algorithm 1 for pseudocode.

For sentiment analysis, we use cross entropy loss as the objective:

$$-\sum_c \mathbb{1}(y, c) \log(\hat{y}_c) \tag{4}$$

where y_c is the sentiment class corresponding to X and $\hat{\boldsymbol{y}}$ is defined by Equation 1. We use binary cross-entropy loss for the paraphrase prediction task:

$$-\left[y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})\right] \tag{5}$$

where y is 1 if X_1 and X_2 are paraphrases of each other and 0 otherwise. $\hat{y}$ is defined by Equation 2
For the STS task, we use mean squared error as the objective:

$$(y - \hat{y})^2 \tag{6}$$

where y is the correct similarity value, and $\hat{y}$ is our predicted similarity value as in Equation 3.

[1] See the project handout for a detailed explanation of how we modified their proposed algorithms.

Algorithm 1: Multitask Training MTFT model

Input: Pretrained BERT embeddings
Randomly initialize model parameters
`// paraphrase, similarity, sentiment datasets`
Initialize $D_{para}, D_{sts}, D_{sst}$
for *epoch* $\in$ *[1,...,arg.epoch]* **do**
  for $b_t \in D_{para}$ **do**
    `//` b_t `is a batch in` D_{para}
    Compute binary cross entropy loss (see Equation 5)
    Use AdamW optimization to update model
  for $b_t \in D_{sst}$ **do**
    `//` b_t `is a batch in` D_{sst}
    Compute cross entropy loss (see Equation 4)
    Use AdamW optimization to update model
  for $b_t \in D_{sts}$ **do**
    `//` b_t `is a batch in` D_{para}
    Compute mean squared error (see Equation 6)
    Use AdamW optimization to update model

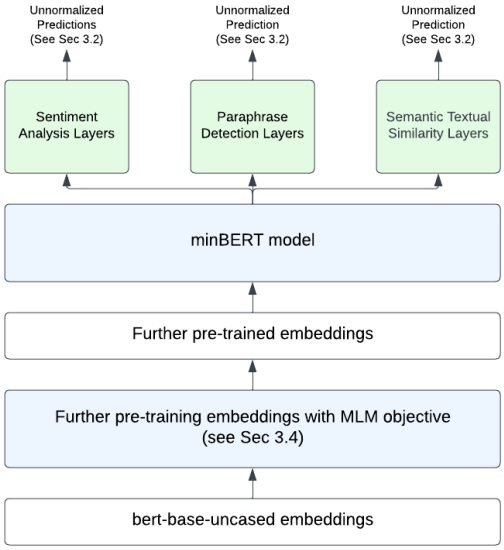

Figure 1: Our Model Architecture

3.4 Further Pretraining

Masked Language Objective The parameters of the pretraining model will be learned using the Masked Language Modeling (MLM) unsupervised task as described in Devlin et al. (2018). We randomly mask 15% of the input tokens, which means that, if the i-th token is chosen, we replace the i-th token with (1) the [MASK] token 80% of the time, (2) a random token from the vocabulary 10% of the time, and (3) the unchanged i-th token 10% of the time.

Then, let x_i be the final hidden vector for X_i, the i-th masked input token. Dropout is applied to $\boldsymbol{x}_i$. The unnormalized prediction that X_i is labeled as vocabulary word c is determined by an output linear layer:

$$\hat{\boldsymbol{y}}_i = \boldsymbol{W}_{MLM}^{\top} \text{dropout}(\boldsymbol{x}_i) \tag{7}$$

where $\boldsymbol{W}_{MLM}^{\top}$ is an MLM task parameter matrix. We then use cross-entropy loss as the objective:

$$-\sum_{c} \mathbb{1}(y_i, c) \log(\hat{y}_{i,c}) \tag{8}$$

where y_i is the true index of the word corresponding to the token X_i and $\hat{\boldsymbol{y}_{i,c}}$ is the i-th element of $\hat{\boldsymbol{y}}_i$ defined by Equation 7.

Unlike autoregressive language model pretraining, the MLM objective enables the representation to encode bidirectional context. As a result, MLM is more effective at learning contextualized representations for downstream use (Devlin et al., 2018), which is why we chose it for our pretraining objective.

Pretraining Architecture As described in Devlin et al. (2018), the pretraining architecture includes the minBERT model followed by an MLM objective linear output layer. Hidden vectors corresponding to the masked tokens from the final Encoder Transformer layer are fed into this output layer to obtain the unnormalized prediction that a masked token is predicted as a word c, as described by Equation 7.

3.5 Single-Task Fine-Tuning

For single-task fine-tuning, we freeze the Encoder Transformer layers of our minBert model, and only update the weights of the private classifier layers as described in Section 3.2. This allows the task-specific layers to specialize and improve model performance.

4 Data and Evaluation Metrics

4.1 Data

For fine-tuning and testing our multitask minBERT model, we used the Stanford Sentiment Treebank (SST) for sentiment classification, the Quora dataset for paraphrase detection, and the SemEval STS Benchmark dataset for semantic textual similarity.[2]For further pretraining, we used the above fine-tuning datasets, as well as the Rotten Tomatoes, PAWS, and SICK datasets from Huggingface. Key details of these datasets are given in Table 4.1 and further information can be found in the default project handout.

[2]Citations for all these datasets can be found in References

Table 1: The suite of datasets used in this work along with their splits

Dataset	Number of Examples			Labels	Task
	Train	Dev	Test		
SST	8,544	1,101	2,210	5	Sentiment Classification
CFIMDB	1,701	245	488	2	Sentiment Classification
Rotten Tomatoes	8,530	1,066	1,066	2	Sentiment Classification
Quora	141,506	20,215	40,431	2	Paraphrase Detection
PAWS	10,000	1,000	1,000	2	Paraphrase Detection
SemEval STS Benchmark	6,041	864	1,726	0-5	Semantic Textual Similarity
SICK	4439	495	4,906	0-5	Semantic Textual Similarity

4.2 Evaluation Method

To evaluate the downstream NLP tasks, we will use accuracy on the SST dev set for sentiment classification, accuracy on the Quora dev set for paraphrase detection, and Pearson correlation on the SemEval STS Benchmark dev set for semantic textual similarity. These evaluation metrics are described in detail in the default project handout. In order to compare models on their performance on all three tasks, we define

$$avg\_score = \frac{acc_{sst} + acc_{para} + \rho_{sts}}{3} \tag{9}$$

where acc_{sst} is the model's accuracy on the SST test/dev dataset, acc_{para} is the model's accuracy on the Quora test/dev dataset, and ρ_{sts} is the Pearson correlation of the true similarity values from the STS test/dev dataset and the predicted similarity values. This score will be referred to as the "Average Score" in our tables.

5 Experimental Details

Overview We used ablation to understand how multitask fine-tuning, further pretraining, and single-task fine-tuning impact the model's performance compared to a baseline model with only single-task fine-tuning (named **STFT**). First, we experimented with multitask fine-tuning using different numbers of dense layers. Second, we experimented with further pretraining the BERT-base-uncased embeddings using different combinations of in-domain and within-task datasets. To test their performance, we passed these embeddings into the best multitask fine-tuning model: MTFT-D. Third, because further pretraining did not improve the model's performance, we experimented with single-task fine-tuning the MTFT-D and MTFT-DD models. See Figure 2 for a visual description of our experimental method. The hyperparameter values used across all models can be found in Table 2.

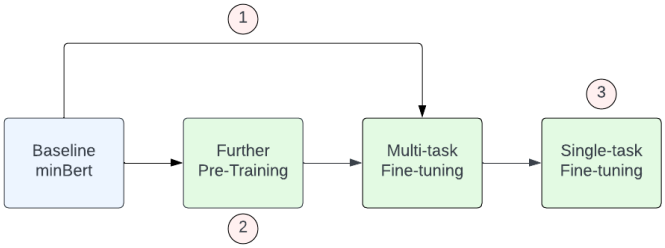

Figure 2: To improve our model we conducted (1) multitask fine-tuning experiments (see Section 5.1) (2) further pre-training followed by multitask fine tuning with one dense layer experiments (see Section 5.2) (3) Single task fine-tuning MTFT-D and MTFT-DD with no pre-training (see Section 5.3)

Table 2: Hyperparameter values across all models

Hyperparameters	Value
Learning Rate for Pretraining	0.00001
Batch Size	8
Dropout Probability for Hidden Layers	0.3
Learning Rate for Fine-Tuning	0.001
Hidden Layer Size	768
Intermediate Layer Size	3072
Adam Optimizer Learning Rate	0.001
Adam Optimizer Exponential Decay Rates	(0.9, 0.999)
Epochs	10
Attention Layer Dropout Probability	0.1

5.1 Multitask Fine-Tuning Experiments

Experiment 1: Multitask Fine-Tuning Using a Single Linear Layer (MTFT) To get a baseline for multitask fine-tuning, we fine-tuned the shared Encoder Transformer layers as well as SST, Para, and STS private single linear classification layers on the `SST`, `Quora`, and `STS` datasets respectively.

Experiment 2: Multitask Fine-tuning Using Two Task-Specific Layers (MTFT-D) The MTFT model did not overfit (see Section 6.2). Thus, to help the model learn more complex nonlinear relationships we added a dense layer with GELU activation to the model (see Section 3.2). We chose the GELU activation function because it is a smooth function, making it easier to optimize using gradient-based methods and often improving deep learning models (Hendrycks and Gimpel, 2020).

Experiment 3: Multitask Fine-Tuning with Additional Large Dense Layer on SST (MTFT-DD) After adding the dense layer, the model uniformly improved but still struggled the most on SST. Thus, we replaced the SST dense layer with two large dense layers.

5.2 Further Pretraining Experiments

Experiment 1: Further Pretraining with New Datasets (FP1-MTFT-D) We started by pretraining our MLM model using three new datasets: Rotten Tomatoes for sentiment classification, PAWS for paraphrase detection, and SICK for Semantic Textual Similarity. We chose these datasets because they are in the same domain as our tasks. As a result, they likely have similar data distributions to our fine-tuning datasets, which could improve the contextualized embeddings for our tasks. After pretraining our embeddings, we multitask fine-tuned them using a model with the same architecture as MTFT-D, as MTFT-D had the highest accuracy and correlation scores in previous experiments.

Experiment 2: Further Pretraining with New Datasets & Fine-Tuning Datasets (FP2-MTFT-D) Since further pretraining with new datasets did not improve performance and Krishna et al. (2022) found success pretraining on fine-tuning datasets, we added our multitask fine-tuning datasets (SST, Quora, STS) as pretraining datasets. This addition improved performance. Nevertheless, this model with further pretraining (FP2-MTFT-D) still under performed our model without further pretraining (MTFT-D) across tasks. Thus, we did not further pretrain the embeddings for our final model.

5.3 Single-Task Fine-Tuning Experiments

Experiment 1: Single-Task Fine-Tuning MTFT-D Model (STFT-MTFT-D) Since the MTFT-D model had the best performance thus far, we experimented with single-task fine-tuning the task-specific layers after multitask fine-tuning.

Experiment 2: Single Task Fine-Tuning MTFT-DD Model (STFT-MTFT-DD) Next, we experimented with single-task fine-tuning the model with two sentiment classification-specific dense layers (MTFT-DD). We hypothesized that giving the model more learnable parameters to single task fine-tune for the sentiment classification task would help the model optimize for the sentiment classification task and thus improve performance on this task.

Table 3: Summary of different models and their features

Model	Multitask Fine-Tuning	Dense Layer in each task-head	Further Pretraining #1	Further Pretraining #2	Single-Task Fine-Tuning	Additional Dense Layer for sentiment
STFT	☐	☐	☐	☐	☑	☐
MTFT	☑	☐	☐	☐	☐	☐
MTFT-D	☑	☑	☐	☐	☐	☐
MTFT-DD	☑	☑	☐	☐	☐	☑
FP1-MTFT-D	☑	☑	☑	☐	☐	☐
FP2-MTFT-D	☑	☑	☐	☑	☐	☐
MTFT-D-STFT	☑	☑	☐	☐	☑	☐
MTFT-DD-STFT	☑	☑	☐	☐	☑	☑

6 Experimental Results Analysis

6.1 Performance on Test Set

Our best model (MTFT-D) had accuracies and correlations of 0.492, 0.806, and 0.583 on the SST, Para and STS Test sets respectively.

6.2 Multitask Fine-Tuning Experiments Analysis

Experiment 2 (MTFT-D): As expected, MTFT-D uniformly outperformed MTFT across tasks. SST accuracy increased by 0.126, Para accuracy by 0.157, and STS correlation by 0.361. Performance likely improved because the additional task-specific dense layers helped the model learn complex and non-linear semantic relationships across all corpora. This intuition agrees with our finding that MTFT did not overfit the data but likely underfit it as demonstrated by the fact that across all epochs both train and dev performance improved, as

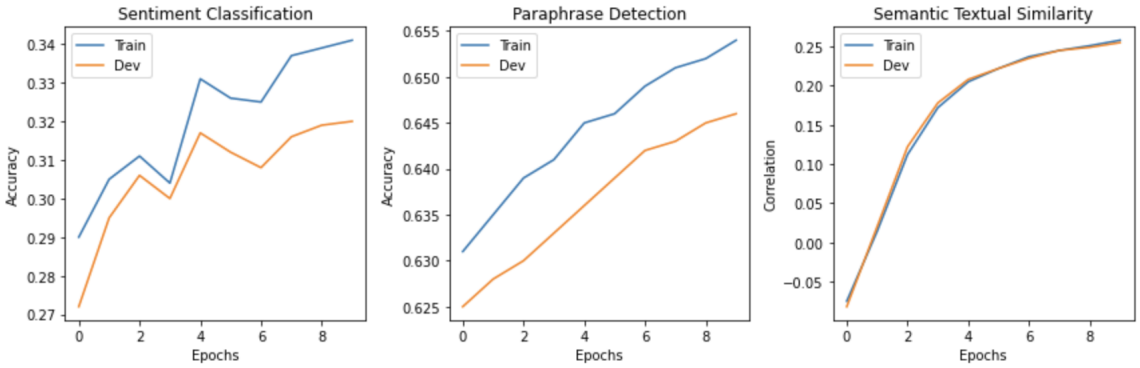

Figure 3: Dev and Train Performance of MTFT on all Three Tasks

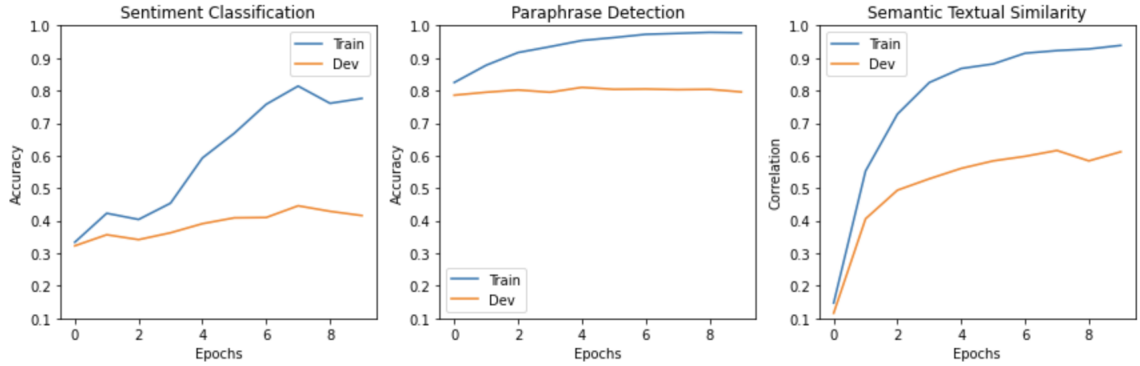

Figure 4: Dev and Train Performance of MTFT-D on all Three Tasks

seen in Figure 3. The MTFT-D model had the lowest accuracy on the sentiment task: 0.446 SST accuracy vs 0.803 Para accuracy and 0.616 STS correlation.

Experiment 3 (MTFT-DD): We further hypothesised that the sentiment classification task required the model to understand more complex non-linearites in the data than the other two tasks. The poorer performance of MTFT-DD compared to MTFT-D negates this hypothesis. In fact, the MTFT-D model may have started to overfit the data by epoch 9 as demonstrated by the slight decrease in sentiment dev accuracy despite and increase in train accuracy in Figure 4. Thus, adding additional dense layers to the STS head of the model did not improve performance.
The MTFT-DD model not only under perform the MTFT-D model on the SST task but also on the Para and STS tasks, even though the architecture of their task-specific layers was not changed. The dense layers may have hurt the model's performance by reducing the amount of information flowing through the shared layers and decreasing the regularization effect of multitask learning. As a result, the model may have become less robust and worse at generalizing to new data.

Table 4: Performance comparison of different models on evaluation metrics on the dev sets

Model	SST Acc	Para Acc	STS Corr	Avg Score
STFT	0.314	0.392	-0.055	0.235
MTFT	0.320	0.646	0.255	0.407
MTFT-D	0.446	**0.803**	**0.616**	0.622
MTFT-DD	0.418	0.800	0.588	0.602
FP1-MTFT-D	0.380	0.798	0.533	0.570
FP2-MTFT-D	0.398	0.795	0.547	0.580
MTFT-D-STFT	**0.473**	**0.803**	**0.616**	**0.631**
MTFT-DD-STFT	0.448	0.800	0.605	0.618

6.3 Further Pretraining Experiments Analysis

Our further pretraining experiments did not improve the performance of our model. As seen in Table 4, the average scores for the models with pretraining, FP1-MTFT-D and FP2-MTFT-D, were 0.570 and 0.580 respectively, which was lower than the score of 0.622 for MTFT-D. Here, we analyze 2 possible reasons for this:

- Accidental Cross-Domain Pretraining: We tried to implement within-task and in-domain pretraining. However, since our model trained on 3 different tasks, we effectively trained our model across domains. Prior studies have found that further cross-domain pretraining BERT does not significantly improve performance as different domains may have different data distributions (Sun et al., 2019). Moreover, words can have different semantic meanings in different domains. For example, we found that in the paraphrase

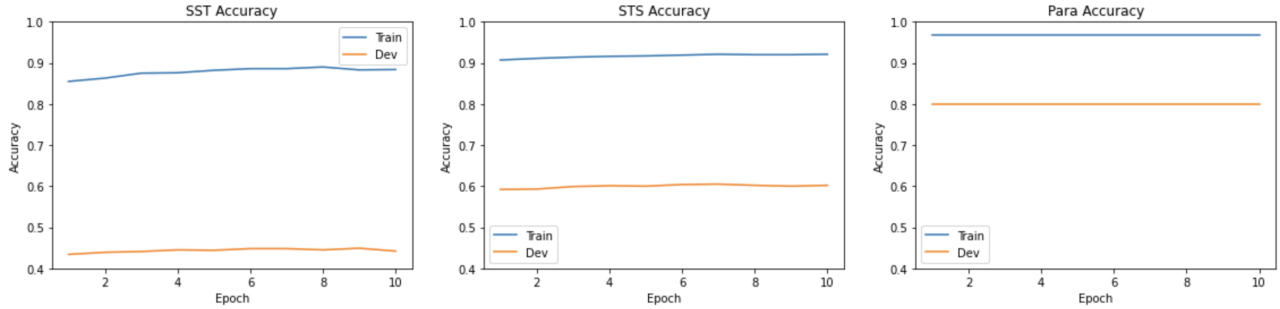

Figure 5: Dev and Train Performance of STFT on MTFT-D model for all 3 tasks

domain the word "predictable" meant "could have been guessed easily", whereas in the movie review domain it meant "unoriginal or formulaic". This discrepancy can cause the further pretrained word embeddings to be less suitable for the individual tasks and lower model performance.

- Lack of Long-Term Dependencies in the Pretraining Corpora: BERT was pretrained on the English Wikipedia and the Book Corpus, which have dependencies across long spans of sentences. Devlin et al. (2018) mention the importance of document-level corpora rather than a shuffled sentence-level corpus for pretraining. Our inputs to the pretraining model were single sentences, which could have limited the effectiveness of our pretraining procedure.

Interestingly, we find that further pretraining using our downstream fine-tuning datasets leads to higher performance as compared to pretraining on the new in-domain datasets. As seen in Table 4, FP2-MTFT-D outperforms FP1-MTFT-D with an average score of 0.58 compared to 0.57. This result agrees with findings from (Krishna et al., 2022), who report that pretraining on fine-tuning datasets can lead to significant performance improvements as compared to other datasets. These findings suggest that the pretraining objective and the similarity of the data distribution to the fine-tuning corpora drive performance gains attributable to pretraining instead of the incorporation of massive datasets.

6.4 Single-Task Fine-Tuning Experiments Analysis

Experiment 1: Single Task Fine-Tuning MTFT-D Model (STFT-MTFT-D) We found that single-task fine-tuning our MTFT-D model improved performance on the sentiment task, but not on the Para or STS tasks. Instead, single-task fine-tuning on the Para and STS tasks led to overfitting. As seen in Figure 5, accuracy and correlation increase over the 10 epochs on train sets, but decrease on the dev sets. This overfitting may be due to the Para and STS tasks being more complex than the sentiment task, which means that they require the model to capture more subtle patterns and relationships. As a result, the model struggles to generalize from the STS and Para training sets to the respective dev sets.

Experiment 2: Single Task Fine-Tuning MTFT-DD Model (STFT-MTFT-DD) In the MTFT-DD model, sentiment classification under performed the other tasks even on the training dataset (at epoch 10, the accuracy on the SST training dataset was 0.776, as compared to a train accuracy and correlation of 0.978 and 0.939 on the paraphrase and STS datasets respectively). We hypothesized that the poor performance of the MTFT-DD model was due to the fact that multitask fine-tuning does not allow the model to optimize parameters specifically for the sentiment classification task whereas single-task fine-tuning does. We expected that (1) single-task fine-tuning the MTFT-DD model would increase performance and (2) that this increase would be greater than the increase in performance from single-task fine-tuning the MTFT-D model because the MTFT-DD-STFT model single-task fine-tunes more parameters specifically for the sentiment classification task than the MTFT-D-STFT model. Expectation (1) was met. MTFT-DD-STFT outperformed MTFT-DD: the average score for MTFT-DD-STFT was 0.016 higher than that of MTFT-DD (0.618 vs 0.602) and the accuracy on the sentiment classification task was 0.03 higher (0.418 vs 0.448). However, expectation (2) was not met. Single-task fine-tuning MTFT-DD led to a 7.18% improvement on the sentiment classification task which was not significantly larger than the 6.05% improvement when single-task fine-tuning the MTFT-D model. Single-task fine-tuning more parameters for the sentiment classification task likely did not correlate with higher accuracy because additional parameters does not address the lack of relevant features in the input data, which is the main cause of error on the sentiment classification task (see Section 7).

7 Qualitative Analysis

Below, we analyze the errors made by our model with the highest performance MTFT-D-STFT on the three tasks.

7.1 Error Analysis of Sentiment Classification

As seen in Table 4, when the model predicts the incorrect class, it tends to only be off by 1 class. This indicates that the model may struggle to understand the fine-grained notions of negative vs. somewhat negative and positive vs. somewhat positive, yet the model understands higher level negative vs positive sentiment.
The model's most common error is to output class 2 (neutral). This error often occurs when many words in the review rarely if ever appear in the training data. For example, in the review "A bloated gasbag thesis grotesquely impressed by its own gargantuan aura of self importance," "gasbag" and "aura" do not appear in the training data and "gargantuan", "bloated", and "grotesque" appear three or fewer

Off By	Proportion
1	0.86
2	0.13
≥ 3	0.01

Table 5: Number of classes the model is off by when it predicts the wrong sentiment class.

times. As a result, the model lacks a strong sense of the review's sentiment and guesses 2, the middle sentiment class, to minimize MSE loss.

7.2 Error Analysis of Paraphrase Detection

	Prediction: 0	**Prediction: 1**
Label: 0	True Negative: 0.5	False Positive: 0.13
Label: 1	False Negative: 0.07	True Positive: 0.30

Table 6: The confusion matrix for paraphrase detection.

Both the train and dev sets contain about 50% more negative labels than positive labels, yet false positives are more frequent than false negatives. This may be because often in the dataset question pairs are very similar except for one word which changes the question's meaning. In this case, the model tends to incorrectly predict positive. The following pairs are examples: "Which are the main inhibitory neurotransmitters?" and "What are the main neurotransmitters?", "What are some facts that everyone knows?" and "What are some facts that everyone should know?", and "How can I get freebies in India?" and "Where can I get freebies in India?".
The false negatives exhibit less of a pattern. However, one common error is that the model assumes that changes in subject translate to different questions. This is often a fair assumption but is not always the case when the subject can be replaced with a pronoun, such as in "Can a person feel happy and sad at the same time?" and "Can you feel happy and sad at the same time?". Another common error is that the model cannot draw connections between an entity and its abbreviated version, such as in "What is the implication of free education in RTE?" and "What is the implication of free education in the Right to Education Act?"

7.3 Error Analysis of Semantic Textual Similarity

We observe that the closeness of the label and prediction similarity scores is highly dependent on continuous lexical overlap and the length of the sentences. For example, for the sentences "Several thousand 3rd infantry troops, including the 3rd brigade combat team based at Fort Benning in Columbus, began returning last week." and "A few thousand troops, most from the division's 3rd brigade combat team based at Fort Benning in Columbus, began returning last week, with flights continuing through Friday.", which are long and have high continuous lexical overlap, the model accurately outputs the label (4.0). In contrast, for sentences without high continuous lexical overlap, the model tends to be inaccurate. For example, for the sentences "A white bus with the word Julia is near water and a big ship" and "A white bus with water and a barge in the background" the model incorrectly outputs 0.86 instead of 3.8.
Longer sentences contain more words and therefore, more opportunities for words to have semantic relationships with each other. This makes it easier for the model to capture the overall meaning of the sentence, as it has more context to work with. In addition, the longer the sentence, the more opportunity for continuous lexical overlap. In general, lexical overlap is a good indication of whether two sentences are similar. In combination, this explains why the model is more accurate for longer sentences with very high or very low continuous lexical overlap.

8 Conclusion

We investigated how different fine-tuning and further pretraining techniques affect the performance of our multitask model on three tasks: sentiment classification, paraphrase detection, and semantic textual similarity. We have findings in three main areas. (1) Multitask Fine-tuning: Adding a task-specific layer with GELU activation improved performance over our baseline multitask model without dense layers, likely because this fine-tuning allowed the model to learn non-linear task-specific relationships across the corpora. (2) Further Pretraining: further pretraining did not improve the performance. However, further pretraining on the downstream fine-tuning datasets led to higher performance than further pretraining on new datasets. (3) Single-Task Fine-Tuning: A model trained with only multitask fine-tuning outperformed a model with only single-task fine-tuning, probably because the multitask learning improved the generalizability of the learned representations across tasks. Also, single-task fine-tuning after multitask fine-tuning helped tasks that had low train metrics in early epochs but led to overfitting otherwise. Overall, this meant our model **MTFT-D-STFT** had the best performance across tasks.
We then analyzed the errors made by this model. For sentiment, the most common error occurred when the model encounters words during testing that it had not encountered before. For paraphrase detection, it makes errors when a change in one word changes the meaning of the sentence, or when the subject is changed across paraphrases. For the STS task, short sentence inputs with low continuous lexical overlap often result in poorer predictions.
In future work, we want to see if true within-task or in-domain pretraining would lead to model performance improvement on a specific task. For example, we want to try pretraining the model on only sentiment classification datasets and see if this improves model performance on the sentiment classification task while maintaining the same performance on the Para and STS tasks. Additionally, we want to investigate how further pretraining the minBERT model with single-task fine-tuning compares to multitask fine-tuning with single-task fine-tuning. Finally, we want to experiment with gradient accumulation and conduct a random hyperparemeter search with early stopping on the learning rates and hidden layer dropout probability.

References

2022. Quora question pairs. Papers With Code. https://paperswithcode.com/dataset/quora-question-pairs.

2023. Cs 224n default final project: minbert and downstream tasks. https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1194/default-final-project-minbert.html. Adapted from the "minBERT" assignment developed at Carnegie Mellon University's CS11-711 Advanced NLP, created by Shuyan Zhou, Zhengbao Jiang, Ritam Dutt, Brendon Boldt, Aditya Veerubhotla, and Graham Neubig.

Daniel Cer, Mona Diab, Eneko Agirre, Iñigo Lopez-Gazpio, and Lucia Specia. 2017. SemEval-2017 task 1: Semantic textual similarity multilingual and crosslingual focused evaluation. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 1–14, Vancouver, Canada. Association for Computational Linguistics.

Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer, and Veselin Stoyanov. 2020. Unsupervised cross-lingual representation learning at scale. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8440–8451, Online. Association for Computational Linguistics.

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

Dan Hendrycks and Kevin Gimpel. 2020. Gaussian error linear units (gelus).

Diederik P. Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization.

Kundan Krishna, Saurabh Garg, Jeffrey P. Bigham, and Zachary C. Lipton. 2022. Downstream datasets make surprisingly good pretraining corpora.

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

Ilya Loshchilov and Frank Hutter. 2017. Decoupled weight decay regularization.

Marco Marelli, Stefano Menini, Marco Baroni, Luisa Bentivogli, Raffaella Bernardi, and Roberto Zamparelli. 2014. A SICK cure for the evaluation of compositional distributional semantic models. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*, pages 216–223, Reykjavik, Iceland. European Language Resources Association (ELRA).

Bo Pang and Lillian Lee. 2005. Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales. In *Proceedings of the ACL*.

Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks.

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 *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642.

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, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *CoRR*, abs/1706.03762.

Xiaogang Wang, Cha Zhang, and Zhengyou Zhang. 2009. Boosted multi-task learning for face verification with applications to web image and video search. *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 142–149.

Yuan Zhang, Jason Baldridge, and Luheng He. 2019. PAWS: Paraphrase Adversaries from Word Scrambling. In *Proc. of NAACL*.