MT-BERT: Fine-tuning BERT for Downstream Tasks Using Multi-Task Learning

Stanford CS224N Default Project

Neha Kunjal
nkunjal@stanford.edu

Hermann Kumbong
kumboh@stanford.edu

Abstract

The goal of this project is to fine-tune the contextualized embeddings from BERT to perform well simultaneously on three different downstream tasks: Sentiment Analysis (SST), Paraphrase Detection (PD), and Semantic Textual Similarity (STS). Our work uses four main techniques to improve the baseline BERT implementation: additional pretraining on the target domain data using Masked Language Modelling, multi-task fine-tuning using gradient surgery, single-task fine-tuning, and feature augmentation. We are able to achieve accuracies of **0.539** and **0.877** for SST and PD respectively, and a Pearson correlation score of **0.863** on the STS test set using a single model without ensembling.

1 Key Information to Include

Our mentor was Manasi Sharma, we are not sharing the project and don't have external collaborators. We are sharing Neha Kunjal's late days for this project: (4 -> 2 + 2)

2 Introduction

Pre-training language models has emerged as a very effective method for advancing the state of the art in many natural language processing(NLP) tasks. Pre-training involves training a model on one or more tasks that help it perform well on downstream tasks. Given a pre-trained model, we could either fine-tune it or use feature-based approaches to improve its performance on downstream tasks [1]. In feature-based approaches like ELMo, the model uses features relevant to the task at hand in addition to the pre-trained representation which is also just another feature [2]. Our work focuses on the fine-tuning approach. Given a pre-trained model, fine-tuning augments the model with task-specific layers for each downstream task and then trains the model by fine-tuning the pre-trained model weights alongside the task-specific layer weights.

Prior to BERT (Bidirectional Encoder Representations from Transformers) [1], most fine-tuning approaches used a unidirectional pre-trained language model[3] which significantly limited their expressive power and hence performance. BERT addresses this limitation by using a "masked language model" (MLM) pre-training objective which randomly masks words and then tries to predict the vocabulary ID of the masked word using only its context [1]. This allows BERT to use both the left and right contexts of each token to learn representations.

The goal of this project is to fine-tune BERT to perform well simultaneously on three different downstream tasks: Sentiment Analysis (SST), Paraphrase Detection (PD), and Semantic Textual Similarity (STS). Fine-tuning BERT for a single task is a straightforward process but fine-tuning it to perform well on multiple downstream tasks simultaneously is quite challenging for a number of reasons. First, the size of each task dataset could vary widely so if we learn on the complete dataset for each task, those with the larger dataset may overshadow the learning of those with smaller datasets and at the same time, reducing the amount of data used to train for the task with larger datasets could lead to lower performance compared to single task fine-tuning for that task alone. The standard method of learning a task is by minimizing some loss function using a gradient-descent-based approach. However, when fine-tuning for multiple tasks, we could have different objective loss functions for each task which could lead to conflicting gradients when doing gradient-descent.

Despite the challenges posed by multi-task fine-tuning, it does provide significant benefits. It is model efficient since there is weight sharing between the different tasks as compared to having independent huge models for each separate task. Furthermore, fine-tuning is usually cheaper and takes a lot less time and resources compared to pre-training, as such, we can pre-train a model once and use it for several different tasks thereby saving resources. In the grand scheme of things, language models and machine learning models to a larger extent, that can perform well on multiple tasks are a lot more useful than those that can only single-task.

Our work explores several current techniques for fine-tuning pre-trained models for downstream tasks. We do additional pre-training of the BERT weights on the target domain data using the Masked LM unsupervised training objective. We explore multi-task fine-tuning using Gradient surgery and other heuristics. In addition, to multi-task finet-uning, we fine-tune the model on each individual task after completing multi-task fine-tuning. We further explore the performance impact of additional input features such as parts of speech (POS Tagging) and synonym augmentation using textattack. We finally perform a thorough hyperparameter search to find the best-performing model.

The key findings from our work are as follows: Multi-task fine-tuning provides the biggest boost to overall performance on all tasks. Single-task fine-tuning after multi-task finetuning can provide an additional performance boost for some tasks. Additional pre-training of the BERT model on target domain data using the masked language model objective provided an additional performance improvement for downstream tasks. Finally, feature augmentation did not lead to any significant improvement in the model's performance on downstream tasks.

3 Related Work

There have been several papers that have tackled the difficulties of multi-task learning. There are three techniques that we explored in particular: gradient surgery, siamese BERT Networks, and Masked Language Model (MLM). Sentence-

Fine-Tuning BERT

There has been a lot of research around fine-tuning BERT for downstream tasks. MT-DNN [4] fine-tunes BERT using a multi-task deep neural network and achieving state-of-the-art performance on the GLUE benchmark. This work guides our approach to designing a multi-task architecture on top of BERT. Cosine similarity fine-tuning is used in [5] to compute the similarity between two sentence pairs whose embeddings are generated using a siamese network. We adopt this approach for downstream tasks that involve computing sentence similarity.

Gradient surgery

Gradient surgery is proposed in [6] as a way of addressing the problem of conflicting gradients amongst the different tasks. The way this technique works is you get the gradients for a batch of each task and determine which gradients conflict with each other based on whether their cosine similarity is negative. If it is negative, the gradient's conflicting features are canceled by removing the projection of the gradient onto the conflicting gradient via the equation from the original gradient ($g_i = g_i - \frac{g_i \cdot g_j}{\|g_j\|^2} g_j$) which is also called PCGrad. By doing this, the hope is that the model moves in the direction which is beneficial to the conflicting tasks rather than canceling each other out and making no update. One thing this model makes a bit harder is to code up an approach where we guarantee to go through all the data considering that we need to do updates in step sync for all tasks.

Pre-training

Masked Language Modeling (MLM) is introduced in [1] as an unsupervised training objective for language modeling. It works by randomly selecting tokens from the input text and replacing them with either the [MASK] token or another randomly selected token and asking the BERT model to predict the replaced token. This is supposed to improve the model by encouraging the embeddings to learn the context of words rather than just depend on the statistics of how many words are present in the sentence.

4 Approach

The flowchart in Fig. 1 highlights the different methods we used to approach the problem.

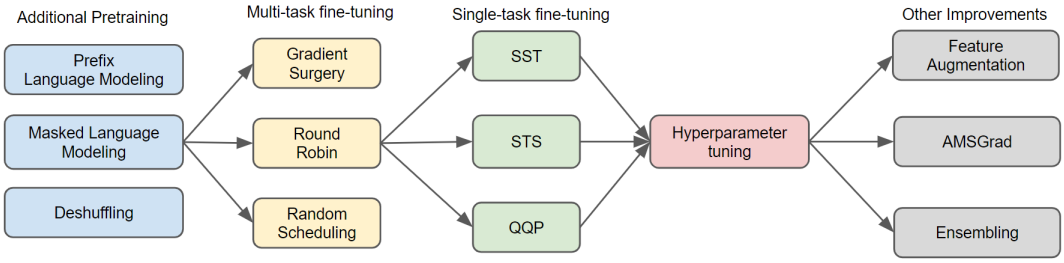


Figure 1: Flow Chart of Different Methods Applied in Fine-tuning BERT

4.1 Baseline

We use the vanilla BERT model implementation with pre-trained weights loaded into the model as our baseline. The model's performance is reported in Table 2. We freeze the BERT layer weights and fine-tune the weights of the task-specific layers and use this as our baseline.

4.2 Model Architecture

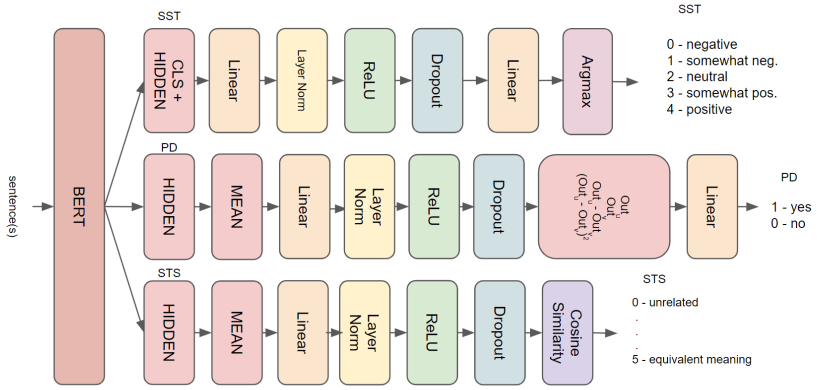


Figure 2: Overview of Model Architecture

Our architecture consists of task-specific layers built on top of the BERT model [1]. Fig. 2 shows our model's architecture.

BERT Sentence Embeddings A key choice in our model design was what output embeddings from BERT to use as an input to the task-specific layers. For each sentence, BERT outputs a [CLS] token and also the last hidden layer state. We experimented with a combination of the different output embeddings and came up with the following layers. For PD and STS, we use the mean pool of the hidden layer state as the input to the task-specific layers. For SST, the input to the task-specific layers is a weighted sum of the [CLS] token embedding and the mean pool of the last hidden state. Let $h \in \mathbb{R}^{N \times d}$ be the last hidden layer BERT embeddings, α_1 and α_2 tunable hyperparameters and let $e_{cls} \in \mathbb{R}^{1 \times d}$ be the embedding for the $[CLS]$ token, then the sentence input embeddings $S_{in}^{(STS)}$, $S_{in}^{(SST)}$ for the STS(same for PD) and the SST task-specific layers are gotten by:

$$S_{in}^{(STS)} = \frac{1}{N}\sum_{i}^{N} h_i \qquad S_{in}^{(SST)} = \alpha_1 e_{cls} + \alpha_2 \frac{1}{N}\sum_{i}^{N} h_i \tag{1}$$

SST Task Layers For sentiment analysis, we apply a 2-layer feed-forward network with RELU non-linearity, layer normalization, and dropout on the first layer to each sentence embedding $S_{in}^{(SST)}$ to get a resultant embedding s'. Our

class prediction is simply the ArgMax of the output of the linear layer.

$$\hat{y}_s = \underset{\theta}{\operatorname{argmin}} \left(W_{sst}(s') \right) \tag{2}$$

STS Task Layers We treat the STS task as a linear regression problem. We use cosine similarity fine-tuning[5]. There are no learnable parameters in the STS task layer and the semantic textual similarity $\hat{y}_{s_1 s_2}$ between two sentence embeddings $s_1 \in \mathbb{R}^d$ and $s_2 \in \mathbb{R}^d$ is given by their scaled cosine similarity.

$$\hat{y}_{s_1 s_2} = \frac{s_1 \cdot s_2}{max(\|s_1\|_2 \cdot \|s_2\|_2, \epsilon)} \times 5 \qquad \mathcal{L}(\theta) = \frac{1}{|S|} \sum_{s_1, s_2 \in S} (\hat{y}_{s_1 s_2} - y_{s_1 s_2})^2 \tag{3}$$

Our training objective $\mathcal{L}(\theta)$ for the STS task is the Mean Squared Error (MSE) loss between the predicted and actual STS scores for the sentence pair.

PD Task Layer As shown in Fig. 2, the Paraphrase detection task layer applies a linear transformation to each sentence embedding from BERT followed by a ReLU non-linearity and dropout. The sentence embeddings are then concatenated and projected using a linear transformation. Let $s'_1 \in \mathbb{R}^d$ and $s'_2 \in \mathbb{R}^d$ be the embeddings of two sentences s_1 and s_2 after the first PD task layer block and $W_{PD} \in \mathbb{R}^{d \times 4}$, then the paraphrase prediction $\hat{y}_{s_1 s_2}$ of the two sentences is given by:

$$\hat{y}_{s_1 s_2} = \sigma \left(W_{PD}(s'_1, s'_2, (s'_1 - s'_2)^2, |s'_1 - s'_2|) \right) \tag{4}$$

The above approach extends the methods in [5] by including the squared difference between the two sentence embeddings in the concatenation. We use the binary cross-entropy loss function as our loss optimization objective. Given the true paraphrase prediction label $y_{s_1 s_2}$ between two sentences $s_1, s_2 \in S$

$$\mathcal{L}(\theta) = \frac{1}{|S|} \sum_{s_1, s_2 \in S} (y_{s_1 s_2} \cdot log(\hat{y}_{s_1 s_2}) + (1 - y_{s_1 s_2}) \cdot log(1 - \hat{y}_{s_1 s_2})) \tag{5}$$

4.3 Extensions

Additional Pretraining The pre-trained BERT model weights we use for our baseline were trained using the MLM and NSP text [1] on a text corpus which has a different distribution from the datasets being used to evaluate the model on SST, STS and PD. In particular, we explore 3 different unsupervised language model training objectives as described in[7]:

Table 1: Different Unsupervised Language Modelling Objectives

Objective	Inputs	Targets
Prefix language modeling	Is anyone	actually mentally ill?
Deshuffling	ill? actually mentally	(original text)
BERT-style MLM	Is <M> actually mentally <M>?	(original text)

We explored the first two objectives which were simpler to implement but yielded poor performance so we stopped pursuing them any further after a few epochs and pursue only the BERT-style MLM.

Multi-task Fine-tuning During the early development of our model we played with the ordering of which tasks we fine-tuned within an epoch. We report the results of the best two strategies (round robin and random-shuffling-dropout) in the *Results section*. For round robin, the model gets to train on all tasks one at a time during each epoch. In random-shuffling-dropout, at each epoch we shuffle to decide the ordering of how the tasks will be used to fine-tune BERT e.g (STS, SST, PD), (PD, STS, SST). Once the order has been decided, then for each task, we randomly decide with a predetermined probability (PD: $\frac{1}{3}$, STS: $\frac{9}{10}$, SST: $\frac{1}{2}$), whether to fine-tune for that task during the epoch or not. The probabilities are chosen so as to reflect the imbalance in training data size across the different tasks. These probabilities are hyperparameters we explored in our design.

Gradient Surgery We also implemented gradient surgery. Our implementation was inspired by [8]. To do this, we first sampled with replacement the same number of data points from each task to be used for training during one epoch. Then during an epoch, we picked a batch from each task sample and found the gradients for each task. We update the gradients of all multihead models and BERT for a batch. At each epoch, we get a sample with the replacement the same number of data points from each task. Then we run a batch with the three tasks we update the whole model using the method PCGrad.

Single-task Fine-tuning After multi-task fine-tuning on all tasks, we freeze the BERT layers and fine-tune the task-specific layers, one task at a time.

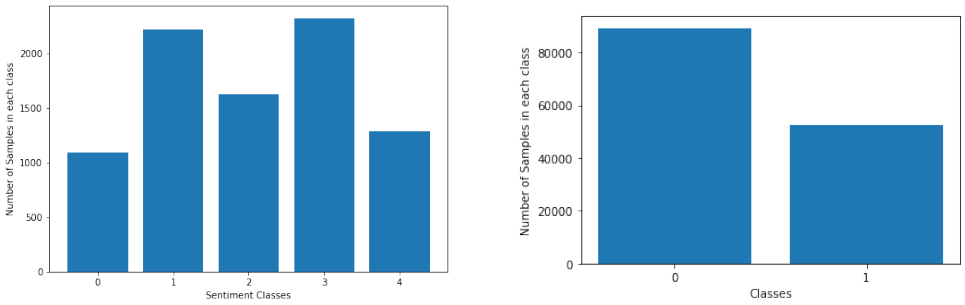
Feature Augmentation We set out to explore how different hand-engineered features can improve the performance of our deep-learning model. We explored Part of Speech (POS) tagging using NLTK [9], The part of speech for each word in the sentence is used as an additional feature to the model. We also pursue synonym augmentation using the embedding augmenter module from textattack [10] which generates words with similar embeddings to a given word. We use this to generate synonyms and use the synonyms to create new data points in our data set. For each sentence, we create a new sentence by replacing one or two words with their synonyms and give the new sentence the same label as the one we generated it from.

Model Ensemble We train a number of different models, using different hyperparameter configurations and slightly tweaked model architectures (e.g LayerNorm in one model but not in the other). Furthermore, each of these models performs differently for different tasks. We experiment with ensemble averaging of the output of these models to see if they yield better performance than a stand-alone model. We use

5 Experiments

5.1 Data

Datasets For PD, we use the Quora Dataset [11] consisting of 400,000 question pairs with labels indicating whether both sentences in each pair are paraphrases of each other and is split as (train 141,506; dev 20,215). For STS, we use the SemEval STS Benchmark Dataset consisting of 8,628 sentence pairs of varying similarity on a scale from 0 (unrelated) to 4 (equivalent meaning) and split as (train 6,041; dev 20,215). The Stanford Sentiment Treebank [12] consists of 215,154 unique phrases parsed from 11,855 sentences and each labeled as negative, somewhat negative, neutral, somewhat positive, or positive, and split as (train 8,544; dev 1,101), and the CFIMDB dataset consisting of 2,434 highly polar movie reviews each labeled as negative or positive and split as (train 1,701; dev 245) are used for SST. We note an imbalanced label distribution for the SST and Quora datasets as shown in Fig. 3



(a) Stanford Sentiment Treebank (b) Quora Question Pairs Dataset

Figure 3: Label Distribution for SST and QQP Datasets

5.2 Evaluation method

We use classification accuracy as the evaluation metric for the SST and the PD tasks. Pearson's correlation coefficient $r = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum(x_i - \bar{x}) \sum(y_i - \bar{y})}}$ is used as the evaluation metric for the STS task. The overall performance metric of the model on all tasks is the unweighted average of the metric scores for each individual task.

5.3 Experimental details

We run all of our experiments on an AWS instance with an NVIDIA A10G GPU, and 8vCPUs and use the model architecture as in Fig 2. To evaluate candidate models, we make use of Mixed precision training which reduces training time by $\approx 20\%$. We train each candidate model for 10 epochs and select the best candidate models which are then further trained with full precision training for longer epochs.

For our optimizer, we start off with the AdamW[13] optimizer which we implemented as part of the project. We further extended the implementation of AdamW to include AMSGrad [14] and experiment with both versions of the optimizer. We stuck with the vanilla AdamW throughout all experiments ($\beta_1 = 0.9$ and $\beta_2 = 0.999$) as AMSGrad did not provide any performance improvements.

After a thorough hyperparameter search, for multi-task fine-tuning, we use a batch size of 4, a learning rate of 1×10^{-5}, and a dropout rate of 0.1 for our final model. Our final model trains for 15 epochs and takes around 3.5 hours to train. In addition to dropout, we used $L2$ weight decay for regularization with $\lambda = 1 \times 10^{-3}$. For all task-specific layers, we use the default pytorch initialization which is the Kaiming uniform [15] initialization.

For additional pretraining with an unsupervised masked language modeling objective, we use a learning rate of $1 \times 10^{-}5$ and a batch size of 128 and train for 100 epochs for about 8 hours.

5.4 Results

We compare different models (methods) by running experiments on our dev set and report the results in Table 2 We observe that our implementation of multi-task-fine-tuning greatly outperforms the Vanilla Pre-trained BERT baseline on all tasks. This is expected since fine-tuning BERT on all tasks allows its weights to be updated via the AdamW optimizer such that it can perform well on the different tasks as opposed to the vanilla BERT which is trained on a generic corpus on a different objective function.

In addition, we observe that freezing the BERT layer weights after Multi-task fine-tuning and fine-tuning the task-specific layers for each task, one at a time provides another boost in performance. We believe this is because, during multi-task learning, task-specific weights are updated as well as the shared BERT weights, thus the model spreads its 'learning' over a wide set of weights some of which are shared and could conflict with other tasks. However, during the additional single-task fine-tuning stage, the model is able to focus solely on the task-specific weights and update them to improve performance only on that particular task without disrupting the 'learning' of other tasks.

We further investigated the performance of different multi-task learning strategies and observe that a simple round-robin schedule performed the best in our experiments. Gradient surgery does slightly worse than round-robin even though we expected it to do much better than round-robin. We think that this is because of the large discrepancy in sizes of the datasets. Because the quora dataset is so much larger than STS (≈ 50x), if we trained in lock step with these two we would highly overfit to STS compared to the others. So we decided to implement this with sampling to get the datasets to be the same size. However the effect of this is that some of the information was lost because it didn't train on all the training data as much as round robin and so might not have as broad embeddings as the other methods. We also suspect that the gradients might not have been that conflicting in which case the method would not have much effect.

Table 2: Accuracy for SST, PD and Pearson Correlation Coefficient for STS on dev set.

Method		**PD**	**SST**	**STS**	**Overall**
BASELINE (Vanilla Pre-trained BERT)		0.651	0.390	0.627	0.556
MULTI-TASK (MT) FT	RANDOM	0.833	0.516	0.844	0.731
	ROUND ROBIN	0.868	0.525	0.850	0.747
	PcGRAD	0.866	0.494	0.819	0.726
PRE-TRAINING + MT FT		0.855	0.485	0.825	0.722
MT FT + SINGLE TASK (ST) FT		0.880	0.525	0.850	0.751
POS TAGGING		0.868	0.507	0.840	0.738
PRE-TRAINING + MT FT + ST FT		**0.881**	**0.525**	**0.879**	**0.761**

We also investigate the effect of pre-training on model performance. We explored three different pre-training unsupervised loss objectives but stuck to Masked Language Modelling (MLM), since pre-training is resource-consuming and Prefix Language Modelling and deshuffling did not show any good performance in the first few epochs, we dropped them and focused on MLM. As shown in Table 2, pre-training before doing multi-task fine-tuning and single-task learning leads to improved performance. We expected this outcome since the pre-trained BERT weights we start off with are trained in a different distribution from the data we use to evaluate our model, thus pretraining on the same distribution of data allows the model to 'adapt' to it. However, the gain in performance we got was not as large as we expected [16]. We suspect that this could be due to the small size of the datasets for the STS and SST tasks.
Figure 4 shows how the validation accuracy for different methods that we tried changes across different epochs. The

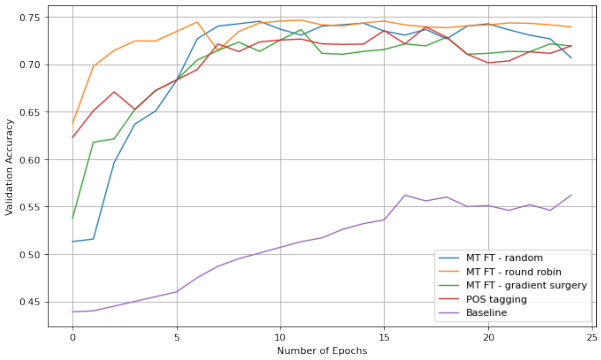


Figure 4: Validation Across Different Epochs for Different Multi-Task Learning Strategies compared with Baseline

model easily overfits the training data especially in the case of STS and SST since the model has a huge number of parameters and the respective datasets are relatively small. We experimented with POS Tagging and synonym augmentation using NLPAug, our expectation was that these would improve the model performance but the improvements we got were mild. We believe that due to the nature of the downstream tasks, synonym augmentation is not a great method to use. For example in semantic textual similarity, replacing a word with its synonym could change the contextual meaning and hence the STS score between two sentences. **Test set results** Our submission on the test leaderboard yields accuracies of $\mathbf{0.539}$ and $\mathbf{0.877}$ on the SST and PD tasks respectively. We obtain a Pearson correlation score of $\mathbf{0.863}$ on the STS task and an overall model score of $\mathbf{0.760}$

6 Analysis

6.1 Test Set Vs Dev Set Results

Table. 3 compares results on the dev and test sets. We observe that the overall performance on all tasks is almost the same. However, at the individual task level, our performance on the test set for the SST dataset is better than on the dev set while that of the PD and STS tasks is worse on the test set. The worse performance could be because of two different reasons: Our model overfit our model to the dev set so it failed to generalize to the test set. Another possible explanation could be that the train/dev sets have a slightly different distribution from the test set. Furthermore, we

Table 3: Comparison of Results on the DEV and TEST sets

Dataset	PD	SST	STS	Overall
DEV	**0.881**	0.525	**0.879**	**0.761**
TEST	0.877	**0.539**	0.863	0.760

observed that while the dev set accuracy for the SST task peaked at 0.525 and then reduced thereafter, the accuracy of the train set kept increasing to about 0.9988. This is possibly due to overfitting the training data set such that it fails to generalize to the dev set caused by the small size of the SST dataset compared to a large number of model parameters.

6.2 Error Analysis

Looking at the predictions that our model produced, we believe that for PD and STS our model generally does well for longer sentences that have many repeated words. As for SST, our model seems to do better when the sentence has positive words only or negative words only. We also believe that classes seen more in training are more likely to be classified during inference, as we noticed a decent number of 3 classifications and 0 classifications in our errors. Let's look at some examples of dev predictions in each of tasks that were done incorrectly from our model to see some of the possible shortcomings.

6.2.1 PD

In table 4, the reader can see that the sentences that were classified as similar but actually aren't, have a lot of repeated vocabulary. For instance, in the first example, the difference between the two is the use of "did you" vs "do we" makes a

Table 4: Misclassified Paraphrase Detection examples

Sentence 1	Sentence 2	Correct Label	Model Prediction
How did you fall in love?	How do we fall in love?	0	1
What brings people to Quora?	What brings people back to Quora?	0	1
Why do people hurt?	Why do hurt people hurt people?	0	1

big difference in a human's understanding of the sentence but not for our model. We think this is because the so many words in common and used in the same order 'convince' our model that the sentences in both words have same context.

6.2.2 SST

Table 5: Misclassified SST examples

Sentence	Correct Label	Model Prediction
It 's somewhat clumsy and too lethargically paced – but its story about a mysterious creature with psychic abilities offers a solid build-up , a terrific climax , and some nice chills along the way .	1	3
Scores no points for originality , wit , or intelligence .	0	1
Hilariously inept and ridiculous .	3	0

+In table 5, it seems like when there are both negative and positive words that our model has a hard time figuring out the right weight it should give to each word. Our model also does not seem able to realize that some parts of the sentence even if they are shorter influence the sentiment of the sentence more than a longer part. For instance in the first example, the part that says "It 's somewhat clumsy and too lethargically paced" defines the sentiment of the sentence even though the rest of the sentence without it could be construed as positive. We think this is hard for our model because sometimes segments of a sentence should cancel out and sometimes like in this case one part dominates for the sentiment.

6.2.3 STS

Table 6: Misclassified STS examples

Sentence 1	Sentence 2	Correct Label	Model Prediction
A woman is putting on sun glasses.	A woman puts on sunglasses.	5.0	2.7
the people are running a marathon	People are running a marathon	5.0	4.3
What is your definition of "life force"?	What is your definition of "soul"?	3.6	1.9

In table 6, we see that synonyms like "life force" vs "soul", "sun glasses" vs "sunglasses", and "the people" vs "people" seem to register as very different embeddings in our model suggesting that our model does not handle bigrams being of similar meanings as unigrams well.

7 Conclusion

In conclusion, our project explored different approaches for fine-tuning BERT to perform well on 3 downstream tasks simultaneously: Semantic textual similarity, Sentiment analysis, and paraphrase detection. Using a systematic approach, we begin by pre-training BERT with data from the target domain using the Masked Language Modelling (MLM) objective which leads to improved performance. Our game changer is multi-task fine-tuning. We explore different strategies from simple round-robin to gradient surgery. The simple round-robin provides the best performance for all datasets. We use cosine-similarity fine-tuning for the STS task which brings about a huge boost in performance. We further fine-tune BERT for each single which leads to more performance gains and finally we explore the effects of feature augmentation like synonym augmentation and part of speech tagging. Although our model performs well on the tasks, the accuracy on the SST task (while on par with most of the class) could be better. Furthermore, our model is not so robust since changes in random seed or hyperparameters can lead to large changes in performance.

References

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

[2] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations, 2018.

[3] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. 2018.

[4] Xiaodong Liu, Pengcheng He, Weizhu Chen, and Jianfeng Gao. Multi-task deep neural networks for natural language understanding. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4487–4496. Association for Computational Linguistics, 2019.

[5] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks, 2019.

[6] Tianhe Yu, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. Gradient surgery for multi-task learning. *Advances in Neural Information Processing Systems*, 33:5824–5836, 2020.

[7] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer, 2020.

[8] Wei-Cheng Tseng. Weichengtseng/pytorch-pcgrad, 2020.

[9] Steven Bird, Ewan Klein, and Edward Loper. *Natural language processing with Python: analyzing text with the natural language toolkit*. " O'Reilly Media, Inc.", 2009.

[10] John Morris, Eli Lifland, Jin Yong Yoo, Jake Grigsby, Di Jin, and Yanjun Qi. Textattack: A framework for adversarial attacks, data augmentation, and adversarial training in nlp. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 119–126, 2020.

[11] `https://www.quora.com/q/quoradata/First-Quora-Dataset-Release-Question-Pairs`.

[12] `https://nlp.stanford.edu/sentiment/treebank.html`.

[13] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization, 2019.

[14] Phuong Thi Tran and Le Trieu Phong. On the convergence proof of AMSGrad and a new version. *IEEE Access*, 7:61706–61716, 2019.

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification, 2015.

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