# Optimizing minBERT for Multiple Classification Tasks

Stanford CS224N Default Project
External Collaborators: N/A. Sharing Project: N/A

**Eddie Dilworth**
Department of Computer Science
Stanford University
edjd@stanford.edu

**Priyanka Shrestha**
Department of Computer Sience
Stanford University
shrestp@stanford.edu

**Savitha Srinivasan**
Department of Computer Sciene
Stanford University
savitha@stanford.edu

## Abstract

We perform sequential ablation studies over a number of approaches, including additional pre-training, more expressive prediction headers, and gradient surgery to improve a BERT architecture (minBERT) for three classification tasks: sentiment analysis, paraphrase detection, and semantic textual similarity. Our best model's average performance across tasks improved significantly upon our baseline round-robin training approach. We show that using MLM pre-training in conjunction with NLI pre-training increases model performance as opposed to either pre-training approach alone. We also demonstrate that for the STS task, optimizing the MSE objective function with the cosine similarity score of our two sentences works much better than optimizing MSE over just a linearly predicted logit.

## 1 Introduction

Many classic NLP tasks such as sentence classification and sentiment analysis rely on language model pre-training [4]. One of the most successful models in this field is the Bidirectional Encoder Representations from Transformers (BERT) model introduced by Devlin et al. [4] which is a transformer-based, pre-trained deep neural network that has been shown to achieve state-of-the-art results on numerous NLP tasks. As BERT shows, pre-trained models can improve performance on a variety of NLP tasks when fine-tuned; however, one challenge is ensuring that the model's contextual embeddings are generalizable across a diverse set of NLP tasks. Since BERT's inception, many studies have sought to further train BERT layers to improve performance [6, 9]. In this study, we explore different techniques for fine-tuning minBERT, a simplified version of BERT, to simultaneously improve performance upon the minBERT baseline[2] on three NLP tasks: sentiment analysis (SST), paraphrase detection (Para), and semantic textual similarity (STS).

We explore three types of additions to our baseline minBERT model: **(1) General Multi-Task Training Variations**. This includes training our model on each task sequentially versus performing multi-task learning on the added losses from each task, taking various sampling approaches on our different-sized training datasets, and using gradient surgery to try to balance conflicting and different-magnitude gradient steps. With these, we aim to improve generalization across each of our tasks. **(2) Additional Pre-Training**. Here, we explore training the underlying minBERT layers on both the masked-language-modelling task (MLM) for our training dataset, and on the natural language inference task (NLI) for the Stanford NLI dataset [10]. These approaches generally aim to create more robust and generalizble minBERT sentence embeddings before we fine-tune for our specific classification tasks. **(3) Improving our Task-Specific Predictions**. Here, we add additional layers to our prediction headers, and modify our STS-specific prediction head and loss function to incorporate the Cosine Similarity of our minBERT pooled outputs.

## 2 Related Work

Sun et al. [9] presents a detailed framework for using pre-training and finetuning techniques to improve BERT's performance on a target task. Significantly, they found that pretraining on in-domain and within-task training data, is highly effective for improving accuracy on the downstream target classification task since it shifts BERT's contextual embeddings closer to the distribution of their ultimate application texts [9]. This motivates our approach of doing MLM pre-training on the STS, Para, and SST data sets.

Liu et al. [6] propose a multi-task BERT (MT-BERT) model that jointly trains multiple tasks using shared BERT parameters. They show that MT-BERT outperforms single-task BERT models on several NLP tasks [6], and that multi-task enables more efficient learning by sharing structure across multiple tasks; thus we choose to implement multitask learning for our three objectives following the algorithm outlined in Liu et al. [6]. However, limitations have been identified; multi-task learning presents a number of optimization challenges and makes it difficult to make large efficiency gains compared to learning tasks independently. Based on the success of [11] in implementing gradient surgery for Multi-Task Learning, we also experiment with projecting conflicting gradients (PCGrad) which works as such: if two gradients are conflicting, PCGrad alters the gradients by projecting each onto the normal plane of the other, preventing the interfering components to be seen by the network [11].

Finally, Reimers et a. [7] presents Sentence-BERT, which specifically seeks to improve the quality of sentence embeddings for the STS task. The authors found that using an MSE objective function on the cosine similarity of sentence embeddings ouputted by BERT as well as finetuning BERT on the NLI task significantly improves upon state-of-the-art sentence embedding models [7]. Thus, we decide to adopt both these approaches in our experimentation.

More generally, by implementing a variety of existing approaches in the literature for improving minBERT's performance within one architecture, we aim to demonstrate how these approaches complement each other.

## 3 Approach

### 3.1 Baseline

For our baseline, we build **minBERT**, a minimal version of the BERT model (Bidirectional Encoder Representations from Transformers) [4] described in the project handout [2]. We first train the model solely on sentiment classification with the SST and IMDB datasets (see Experiments), as instructed. We then add modifications that allow our baseline minBERT to perform well across our three classification tasks.

### 3.2 Main Approach: Basic Model Architecture

As a general overview of our proposed model architecture, we choose to share the same minBERT underlying layers among our three tasks, and add task-specific linear layer(s) for predictions. When training, we train both our prediction headers and our underlying minBERT layers, which improves our model's expressiveness but requires more careful consideration of conflicting losses which we consider in our extensions. For the SST task header, we add a linear layer $W_{sst} \in R^{pooledX5}$ that takes in the pooled output of a sentence from minBERT, and outputs a logit vector with the dimensionality of our 5 sentiment classes. We use cross-entropy loss over these predicted logits for each class (0 to 4). For the Paraphrase task header, we take the BERT pooled output of each of the two sentences, apply a dropout to each, concatenate them, and then use a linear layer $W_{para} \in R^{2*pooledX1}$ on the concatenation to get a single logit. We use binary cross entropy loss over the final logit (with respect to our binary classes of paraphrase or non-paraphrase). For the STS task, we apply the same model architecture as with paraphrase to the BERT pooled output, but with its own linear layer $W_{sts} \in R^{2*pooledX1}$. Since our dataset includes non-integer similarity scores from 0 to 5 up to the first decimal place, i.e. 4.2, we choose to treat our prediction header as completing a regression task, and used MSE loss when training on STS. Note that for cross entropy loss, we optimize $-\sum_{c=1}^{M} y_{o,c} \log(p_{o,c})$ and for MSE lose we optimize $\sum_{i=1}^{D}(x_i - y_i)^2$. Using this general architecture of prediction headers, we explore a number of architectures and training procedures, which we outline below.

### 3.3 Round-Robin vs. Multi-Task Training (General Multi-Task Training Variations)

We first explore the performance of training on each task separately within an epoch ("round-robin" training), versus examining each of our tasks together in a batch and back-propagating the sum of their losses. Specifically, for our round-robin fine-tuning model, which we will consider our general baseline, we take the following approach within each epoch: Sequentially iterate through each dataset individually, and for each dataset, fine-tune on all batches of the dataset using the task-specific loss. The order we iterate through the datasets was: SST, Para, and STS. In contrast, for our multi-task learning approach, we first cycle the training examples of the shorter datasets, SST and STS, so that they were the same length as the longest dataset (Para). Then, within each epoch, we take mini-batches of examples from each dataset, compute the respective losses for each task, add the losses, and backpropagate the across-task loss sum for that mini-batch. Note that we initially choose to cycle the shorter datasets so that each mini-batch optimization step contains examples for each task, although we explore this decision later. As described before, the objective loss function we optimize for multitask learning is: $L = L_{STS} + L_{Para} + L_{SST}$

### 3.4 Adding Layers to our Prediction Headers (Task-Specific Predictions)

Taking the better-performing training procedure between Round-Robin and Multi-Task Training, we next explore how adding layers to our prediction headers can improve our task-specific performance through increased expressiveness. Specifically, we experiment with increasing from 1 to 2 or 3 linear-layers for the task-specific heads (i.e. converting from $W_{sst} \in R^{pooled \times 1}$ to $W_{sst1} \in R^{pooled \times pooled}, RELU, W_{sst2} \in R^{pooled \times 1}$). Note that we add non-linearity by applying ReLU before our second layers, and added an additional dropout between layers 2 and 3.

### 3.5 Sub-Sampling Our Data (General Training-Iteration Approach)

After taking the better-performing architecture from 3.4, we explore the benefits of sub-setting our data. In the prior approaches, for each epoch, we use **upsampling** where we cycle our smaller datasets (SST and STS) so that they were the same length as our largest dataset (Paraphrase); thus, in each epoch our model trains over all of the paraphrase dataset once, all of the SST dataset 17 times, and all of the STS dataset 14 times. However, we wonder whether up-sampling may lead to over-fitting our training datasets for SST and STS, and whether we can get comparable results with a subsampling approach that would also require less training time. Thus, we explore two alternatives: **down-sampling** and **intermediate sampling**.

In **down-sampling**, during each epoch, we take an entirely random subset of Paraphrase and STS so that they have the same number of examples as our smallest dataset, SST. We then apply our Multi-Task Learning approach from 3.4 over the SST dataset, the STS subset, and the Paraphrase subset. We train this model for 10 epochs, as compared to 5 epochs with the up-sampling model.
In intermediate sampling, which balances our up and down-sampling approach, we instead take a random sample with 28,300 training examples (1/5 of the paraphrase training data size). We then cycle the SST and STS examples up until this length, and perform batched multi-task training. We choose to sample 1/5 of the Quora training dataset for our intermediate experiment, so that we can 1) run for 15 epochs and still take roughly the same training time, and train on the same frequency of each dataset in expectation, as only 3 epochs of up-sampling, while still 2) training for enough iterations on each dataset to compare models meaningfully.

### 3.6 MLM Pre-Training (Additional Pre-Training)

Next, we explore adding additional MLM pre-training, prior to fine-tuning our model on our classification tasks. Specifically, following the MLM pre-training algorithm described in Devlin et al. [4], we concatenate all of the SST, Para, and STS training data and code our own masked training dataset for the MLM task in which certain tokens are masked according to the fixed probabilities outlined in Devlin et al. [4]. When creating labels for the masked data we put the actual token as the label for masked data and -100 for any tokens that were not masked. We only predict tokens that are masked and ignore non-masked tokens.

### 3.7 Cosine-Similarity Scores in our STS Header (Task-Specific Predictions)

Because STS is one of our harder prediction tasks, we next explore modifying our prediction header and loss function for STS. Since we ultimately want to predict the similarity between two sentences, we explore whether using the cosine similarity score of their minBERT embeddings, rather than a simple linear transformation, can allow our model to learn better sentence embeddings that capture textual similarity. Specifically, we compare:

1. MSE Loss: Our previous STS header, which concatenates the pooled, dropped-out minBERT outputs for each sentence, and applies a two-stage linear prediction head to get a single logit
2. Cosine MSE Loss: Compute the cosine similarity (Equation 1, Appendix A) between each minBERT pooled, dropped-out outputs (a value between -1 and 1), apply a ReLU (to require a value between 0 and 1), and then multiply by 5 to get a single logit prediction between 0 and 5
3. Linear + Cosine MSE Loss: Apply a single linear layer to the first sentence's minBERT pooled and dropped-out output, and a separate linear layer to the second sentence's min-BERT pooled and drop-out output, and then compute the cosine-similarity, ReLU, and *5 transformation as described above to get a single logit prediction

Note that for each model, we train using MSE loss between our logit and the ground-truth similarity score. We add option 3) to see whether our predictions improve through the addition of a more expressive, linear transformation to our embeddings before applying cosine similarity.

### 3.8 Natural Language Inference Pre-Training (Additional Pre-Training

Next, we perform pre-training on the Stanford NLI dataset. The NLI task is to determine the inference-relationship of two input texts: specifically, whether there's entailment, contradiction, or neutrality [10]. We hypothesized that training our underlying minBERT layers on NLI, prior to our specific classification tasks, could be beneficial since it requires learning notions of semantic similarity that are used in the paraphrasing and STS tasks we are evaluating on. Our approach to NLI pre-training is as follows: we run a forward pass through minBERT on each of the two sentences to get their pooled output, apply a dropout to each, concatenate the outputs, and then apply a linear layer ($W \in R^{pooled,3}$ over the concatenation with an outputted logit for each of our three NLI classes. We then train our model using cross-entropy loss between our logits and the true labels. We train for two epochs on the SNLI dataset, prior to fine-tuning any of our three classification tasks. For the first epoch, we freeze the underlying minBERT layers and only train the header - this is to limit an initial large shift in our underlying minBERT layers from our pre-training phase. During the second epoch, we unfreeze the underlying minBERT layers and train those layers and the header. We compare this NLI pre-training approach in combination with and separately from MLM pre-training to assess the extent to which it is beneficial to shift our underlying minBERT layers prior to fine-tuning. We also explore training on a subset of the SNLI data (65500 samples) versus the whole SNLI data set (550152 samples), to determine the optimal data set size for in domain pre training.

### 3.9 Gradient Surgery (General Training-Iteration Approach)

We lastly explore applying Gradient Surgery to each of our task-specific losses within a mini-batch, rather than simply summing them. We use the PC Grad implementation and Python package outlined in Yu et al. [11], to see how it may help reconcile conflicting and different-magnitude gradients.

## 4 Experiments

### 4.1 Data

We use six datasets: the Stanford Sentiment Treebank (SST) dataset [8] the CFIBDM dataset [2], the Quora dataset [5], the SemEval STS Benchmark Dataset [3], the SICK 2014 Data set [1] (contains sentence pairs for STS task) and the Stanford NLI (SNLI) dataset [10]. We concatenate 4501 sentence pairs from the SICK 2014 train data to the STS train dataset, increasing the length of the STS training set to 10541 samples.

Table 1: Dataset Information and Characteristics

| Task | Dataset | Labels | Size |
|---|---|---|---|
| Sentiment Analysis | SST | 0-4 | 215,154 unique phrases |
| Paraphrase Detection | Quora | 0-1 | 400,000 question pairs\ |
| Semantic Textual Similarity | SemEval STS | 0.0-5.0 | 8,628 sentence pairs |
| Semantic Textual Similarity | SICK 14 | 0.0-5.0 | 4501 sentence pairs |
| Natural Language Inference | SNLI | 0-2 | 570,00 sentence pair |

## 4.2 Evaluation method

We evaluate our SST and Quora datasets using accuracy (the number of correct sentiment classifications / total number of predictions) as our score metric. For STS, we round our outputs to discretized labels like the true dataset, and use Pearson correlation as our evaluation score, where the x's are predictions and y's are the true labels (Equation 2, Appendix A)).

## 4.3 Experimental details

We first run our baseline minBert and our Round Robin training approach as a baseline, described in approaches. Then, we outline the configurations of all our multitask models in the table below. As described in approaches, we adopt a greedy approach for selecting our extensions; thus, we bold our best model for each experiment, and use that as the control in the next experiment we run. We ran all of our models with a learning rate of 1e-5 until convergence was observed (usually 5-7 epochs, but 10-15 for down and intermediate sampling). On average it took about 4-6 hours to train each model. We implemented all of our models and extensions in PyTorch, extending upon the code provided in the Default Project. We present our model configurations in Table 2.

Table 2: Model Configurations. The best model from each experiment is bolded

| Experiment | Configuration |
|---|---|
| Number of Layers | Multitask + 1 Layer<br>**MultiTask + 2 Layers (2-L)**<br>Multitask + 3 Layers |
| MLM Pretrain | 2-L + 3 epochs pretrain<br>**2-L + 5 epochs pretrain (2-L + 5-PT)**<br>2-L + 10 epochs pretrain |
| Sampling | 2-L + 5-PT down sampling<br>2-L + 5-PT + intermediate sampling<br>**2-L + 5-PT + up sampling (US): same as previous experiment** |
| Loss function | **2-L + 5-PT + US + Cosine MSE Loss (CMSE)**<br>2-L + 5-PT + US + Cosine MSE Loss with Linear Layers |
| NLI Pretrain | **2-L + 5-PT + US + Cosine MSE Loss (CMSE) + All NLI data**<br>2-L + 5-PT + US + Cosine MSE Loss (CMSE) + Subset NLI data<br>2-L + US + Cosine MSE Loss (CMSE) + All NLI data |
| PC Grad | 2-L + 5-PT + US + Cosine MSE Loss (CMSE) + All NLI data + PC Grad<br>**2-L + 5-PT + US + Cosine MSE Loss (CMSE) + PC Grad** |

## 4.4 Results

First, in Table 3, we present our baseline results when we tested minBERT and our single linear classification header on the SST and CFIDM sentiment classification datasets. We see that for both models, the finetune dev accuracy (where both the underlying BERT layers and header are trained) is much higher than the pretrain dev accuracy (where just the header is trained); this is likely because with finetune, we are able to fit more parameters for the task. We also note that the CFIDMB dev accuracy is much higher than SST; this may just be because the CFIMDB data set is an easier classification task (only binary labels) while the SST task has five labels.

Table 3: MinBERT Model from Default Project Part 1

| Task | Pretrain Dev Accuracy | Finetune Dev Accuracy |
|---|---|---|
| SST | 0.409 | 0.525 |
| CFIMDB | 0.788 | 0.967 |

Next, we present the combined results for all of our models across our extensions on the dev set in Table 4. We see that multitask learning outperforms Round Robin training, as our Multitask 1-layer prediction model offers a 0.036 increase in average dev score over the Round Robin baseline. Notably, our best model (starred) uses our 2-linear-layer prediction heads for SST and Para, up-sampling, MLM-Pretraining for 5 epochs, Cosine MSE Loss for STS, and trains on the full NLI dataset. Collectively, this approach improves our average dev score by 0.15 over our baseline Round Robin model. We present the test scores for our best model Table 5 We also note that for the best model, the average test score of 0.672 is slightly lower than our average dev score of 0.682; this is not surprising, since during our model selection process we choose the model that performed best on the dev set, which likely caused some slight overfiting to our dev set. We analyze the results in each of our experiments in more detail in the Analysis section.

## 5 Analysis

As outlined in our approaches, we follow a greedy approach in which we begin with a multi-task baseline, and for each experiment, iteratively choose the best model before trying the next approach. This allows us to consider the effect of an extension as an ablation. First, note that we found that applying Multi-task Learning over our Round Robin baseline offered a 0.026 increase in our Para score, a 0.023 % increase in our SST score, and a 0.06 % increase in our SST score. The across-the-board increase in performance on each task suggests that multi-task learning is able to better generalize over our tasks, so we use it in all ablations discussed below.

### 5.1 Ablation 1: Changing Number of Layers in Task Prediction Heads

In Table 4, we see that across all prediction tasks, our two-layer prediction head architecture performs the best, with a 0.027 increase in average dev score over the one layer architecture. This suggests that the increased expressiveness of the 2-layer architecture seems to help the model to perform well on each specific classification task. However, with the 3-layer architecture, we see a higher average dev score than our 1-layer model but a lower score than 2-layers. Thus, perhaps the 3-layer architecture overfit to our training data or was too complex (i.e., contained too many transformations) for our prediction tasks — at least for our current limitations on learning rate and training epochs. Thus, for Ablations 2-6, all models use the 2-layer architecture.

### 5.2 Ablation 2: Changing Data Sampling Approach

As presented in Table 4, our original up-sampling approach performs the best, exceeding intermediate sampling's average score by 0.012 %. Downsampling performs considerably worse than both upsampling and intermediate sampling both in terms of average dev score and on each task, which initially was surprising since we hypothesized that reducing representation of the majority dataset and the number of cycles for our smaller datasets would help with generalization. Instead, it seems that repeating the minority class several times, and maintaining all the data from the majority data set, helped the model more for our chosen number of epochs (5 for upsampling, 10-15 for downsampling and intermediate sampling). However, as one limitation of this ablation, we expect that running downsampling or intermediate sampling for even more epochs would have helped even out performance, as each training example could then be seen in expectation the same number of times as up-sampling. For example, note that in down-sampling for 10 epochs, not every paraphrase training example will necessarily even be trained on by our model, which reduces our performance. However, given the lack of immediate improvement in generalization, we move forward with up-sampling for our models in Ablations 3-6, and note the need for higher epochs of down-sampling or immediate sampling as a limitation.

### 5.3 Ablation 3: Within Task Pretraining with MLM

Overall, we found that MLM pre-training for 5 epochs proved to offer the most improvement (0.07 % increase in average dev score from 0.585 to 0.592) over our best model from Ablation 2 (Table 4). We see that the training and dev scores for STS and SST also increase significantly with the addition of MLM pretraining, with only a minimal decrease in Para dev score. Thus, within-task MLM pre-training for 5 epochs seemed to be successful at improving the quality of our underlying minBERT layers to the specific data distributions for our datasets. However, it's notable that pre-training for 3 epochs and 10 epochs actually decreased the average dev score compared to our no pre-training model. We theorize that with 10 epochs, we may start to over-fit our underlying minBERT layers on the pre-training task, while pre-training for 3 epochs was likely not enough to make sufficient changes to the BERT embeddings, and perhaps only hurt by slightly deviating from the 224N-provided weights.

Table 4: Model Results on Dev Data Set Across All Extensions Compared to Round Robin Baseline

| **Round Robin Baseline** | | | | |
|---|---|---|---|---|
| | Para Dev | SST Dev | STS Dev | Average |
| | 0.76 | 0.479 | 0.326 | 0.522 |
| **Ablation 1: Number of Layers in Prediction Head** | | | | |
| | Para Dev | SST Dev | STS Dev | Average |
| 1 (initial Multi-Task Learning Model) | 0.786 | 0.502 | 0.386 | 0.558 |
| 2 | 0.811 | 0.5 | 0.443 | 0.585 |
| 3 | 0.804 | 0.49 | 0.409 | 0.568 |
| **Ablation 2: Sampling Approaches** | | | | |
| | Para Dev | SST Dev | STS Dev | Average |
| Down | 0.751 | 0.493 | 0.35 | 0.531 |
| Intermediate | 0.792 | 0.504 | 0.425 | 0.573 |
| Up | 0.811 | 0.5 | 0.443 | 0.585 |
| **Ablation 3: MLM Pretrain Epochs** | | | | |
| | Para Dev | SST Dev | STS Dev | Average |
| 3 | 0.814 | 0.508 | 0.419 | 0.58 |
| 5 | 0.802 | 0.517 | 0.457 | 0.592 |
| 10 | 0.813 | 0.491 | 0.419 | 0.574 |
| **Ablation 4: STS Loss Function** | | | | |
| | Para Dev | SST Dev | STS Dev | Average |
| MSE | 0.802 | 0.517 | 0.457 | 0.592 |
| Cos MSE | 0.821 | 0.508 | 0.669 | 0.666 |
| Linear+ Cos MSE | 0.807 | 0.506 | 0.515 | 0.609 |
| **Ablation 5: NLI Pre-training** | | | | |
| | Para Dev | SST Dev | STS Dev | Average |
| *PT-5 + NLI | 0.836 | 0.51 | 0.7 | 0.682 |
| NLI | 0.804 | 0.51 | 0.653 | 0.655 |
| PT-5 + NLI Subset | 0.836 | 0.509 | 0.67 | 0.661 |
| **Ablation 6: PC Grad** | | | | |
| | Para Dev | SST Dev | STS Dev | Average |
| 2-L + PT-5 + Cos MSE + PC Grad | 0.837 | 0.482 | 0.67 | 0.672 |
| 2-L + PT-5 + Cos MSE + NLI + PC Grad | 0.826 | 0.488 | 0.691 | 0.669 |

Table 5: Best Model Evaluated on Test Set

| **Model** | **Para** | **STS** | **SST** | **Average** |
|---|---|---|---|---|
| **2L + US + 5-PT + CMSE + NLI** | 0.524 | 0.831 | 0.661 | 0.672 |

### 5.4 Ablation 4: Changing Loss Function for STS Prediction Task

We found that Cosine MSE Loss offered an impressive 0.212 % increase in the STS dev pearson correlation over the standard linear MSE loss we used in Ablation 3 (Table 4). This shows that the Cosine MSE Loss is indeed extremely effective in helping the model identify similar sentences, by helping drive more similar sentence embeddings closer to one another in the vector space. We found, however, that the Linear + Cosine MSE Loss was not as effective in improving the STS dev score (0.15 point increase in dev score). The linear layer actually allowed for increased performance on the training set due to higher expressivity, but the added transformation seemed to sacrifice some of the benefit of directly pushing our similar sentence embeddings closer together. Lastly, we note that implementing Cosine MSE loss for STS actually improved our paraphrase-task development accuracy too (from 0.802 to 0.821), while inducing a slight decrease in SST accuracy. It seems possible that since detecting similarity is relevant for both paraphrase and STS, that training our embeddings to increase cosine similarity among similar sentences for the STS task provides an added benefit to detecting paraphrases. Meanwhile, these similarity-trained embeddings may have shifted our underlying layers further from the ideal for the largely separate domain of sentiment analysis. Generally, our net increase in the average dev score is very significant (0.074 %), largely due to the increase in pearson correlation for our STS task, so we use it for the models in Ablations 5-6.

### 5.5 Ablation 5: Cross Domain Pre-Training with NLI

We first compare the impact of MLM pre-training in combination with 1) pre-training on a subset of NLI and 2) pre-training on all NLI data. We see that pre-training on a subset of NLI data (dev score of 0.661) performs worse than on all the NLI data (dev score of 0.682)(Table 4). This suggests that for the model to sufficiently learn notions of semantics that apply to the STS/SST/Para tasks, training on more NLI data is needed. Additionally, the combined MLM and full-NLI pre-training results in a 0.016 % increase in the average dev score from our best model in Ablation 4, and indeed, is our best model overall. Specifically, we note that MLM + NLI offers a 0.15% increase in the Paraphrase dev accuracy, and a 0.03 % increase in the STS dev score, supporting our hypothesis that NLI helps the model learn notions of semantic similarity that are particularly applicable to Para/STS. Finally, we see that pre-training with MLM + NLI (average dev = 0.682) and pre-training with just MLM (average dev = 0.666) both outperform just pretraining with NLI (average dev = 0.655), highlighting that within-task pre-training alone seems to be more effective than pre-training on a separate task alone — but, cross-domain pre-training and within-task pre-training together performs better than either alone.
**Error Analysis on Ablation 5**: Following the success of NLI+MLM over just MLM, we examined what types of inputs on which NLI seemed to aid performance, specifically on the most-improved paraphrase task. We noticed that additional NLI training seemed to improve performance on sentences that were essentially identical, but with a swapped-out word or two. Consider the following example from the Quora dataset:
**SENTENCE 1:** What are the legal separation laws in PA and how do they compare with the ones in Washington?
**SENTENCE 2:** What are the legal separation laws in PA and how do they compare with the ones in Oregon?
The true label is that the sentences are not paraphrases, which our NLI+MLM-pre-trained model correctly predicts, but just MLM-pre-trained doesn't. Along with similar cases, it seems that our additional NLI-pre-trained approach does a better job at detecting the relevance of swapped-out-words, perhaps by virtue of seeing words like Washington and Oregon more in the SNLI training dataset (which we confirmed), and in theory learning explicitly to detect entailment or contradiction among other-wise similar sentences in the SNLI dataset.

### 5.6 Ablation 6: Applying Gradient Surgery (PC Grad) Optimization

Here, we compare our best models from Ablation 4 (MLM) and Ablation 5 (MLM + NLI), and analyze the impact of adding gradient surgery with PCGrad. Notably, we see that for the best model from Ablation 4, PC Grad increases the average dev score by 0.012 % and the dev score for Para by 0.016 % but decreases the dev score for SST by 0.026 % (Table 4). For the MLM + NLI model, PC Grad actually decreases the average dev score by 0.022 % and causes a decrease in all the per-task scores. It is not immediately intuitive why PC Grad would improve our MLM baseline but not MLM+NLI, beyond perhaps if NLI moved our embeddings to a more generalized starting-point where gradient-surgery is less useful for reducing conflicting gradients. We did note that PCGrad caused convergence much sooner for the model it improved (i.e., its best performance on the development set was Epoch 2), so it could be that reducing conflicting and different-magnitude gradients with gradient surgery reaches convergence faster, even if it doesn't necessarily improve our maximum performance.

## 6 Conclusion

Overall, we were able to successfully improve upon our baseline minBERT implementation to improve classification on the sentiment analysis, paraphrase detection, and semantic textual similarity scoring. Our best model achieved an average test score of 0.672 (Table 5) and average dev score of 0.682, which is an increase of 0.16 in the average dev score from the baseline round-robin model. As an overview, we demonstrate that a two-layer architecture for the prediction heads is optimal, up-sampling outperforms downsampling and intermediate sampling with our chosen number of epochs, and MLM pretraining in conjunction with NLI pre-training performs better than solely pre-training on either technique alone. Lastly, we demonstrate that optimizing the MSE objective function with the cosine similarity score of our two sentences works much better than optimizing MSE over solely a linear prediction logit, for the STS prediction task.

While we were able to significantly improve on the baseline, we highlight a few limitations. For both of the MLM and NLI pretraining approaches, we used the default learning rate of $1e-5$; however, papers such as Sun et al. [9] and [6] have shown that varying the learning rate for pre-training tasks from the learning rate for the prediction tasks can often be optimal and limit over-fitting those tasks. Due to time constraints, we were also not able to perform the same in-depth ablations we did with the prediction header and loss for STS, with the paraphrase and SST tasks; future work could explore adding other loss functions, such as Multiple Negatives Ranking Loss and Contrastive Loss for the paraphrase task.

## 7 Acknowledgements

We would like to thank our mentor Xiaoyuan, and the entire CS 224N teaching staff for their support throughout this project.

## References

[1] 2014. Sick 2014 dataset.

[2] 2023. Cs 224n: Default final project: minbert and downstream tasks.

[3] Eneko Agirre, Daniel Cer, Mona Diab, Aitor Gonzalez-Agirre, and Weiwei Guo. 2013. Semantic textual similarity. *Second joint conference on lexical and computational semantics (* SEM), volume 1: proceedings of the Main conference and the shared task: semantic textual similarity*.

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

[5] Samuel Fernando and Mark Stevenson. 2008. A semantic similarity approach to paraphrase detection. *Proceedings of the 11th annual research colloquium of the UK special interest group for computational linguistics*.

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

[7] Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084*.

[8] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher Manning, Andrew Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. *Proceedings of the 2013 conference on empirical methods in natural language processing*.

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

[10] Adina Williams, Nikita Nangia, and Samuel R Bowman. 2017. A broad-coverage challenge corpus for sentence understanding through inference. *arXiv preprint arXiv:1704.05426*.

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

## A Appendix

Equation for Cosine Similarity between sentence embedding $A$ and sentence embedding $B$

$$\cos(\theta) = \frac{A \cdot B}{\|A\|_2 \, \|B\|_2} \tag{1}$$

Equation for Pearson Correlation Coefficient

$$r = \frac{\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \overline{x})^2 (y_i - \overline{y})^2}} \tag{2}$$