Build, Extend, Repeat, Triumph: Extensions to a BERT Multitask Model

Stanford CS224N Default Project

Oliver Lee
Stanford University
olee3691@stanford.edu

Ethan Hsu
Stanford University
ethanhsu@stanford.edu

Manat Kaur
Stanford University
manat@stanford.edu

Abstract

A basic BERT-transformer model demonstrates poor performance when fine-trained on several tasks. In response to this, we explore and evaluate a series of extensions, which either increase representational power, add loss functions, or reduce overfitting. We train a final model trained on sentiment analysis, paraphrase detection, and semantic similarity, which reaps significant performance increases from ensembling, LoRA, multiple negatives rankings loss, cosine similarity loss, and Pearson coefficient loss. Our model gets .764 accuracy on the test set.

1 Key Information

Ethan worked on gradient surgery, the poster, and contrastive learning (not in paper). Manat worked on implementing the multitask classifier, the experimental step of the approach, hyperparameter finetuning, and the plots. Oliver worked on the baseline, loss functions, ensembling, linear layers, LoRA, adding data, and the exploratory step of the approach. We all worked on the paper, of course. Manat and Oliver also worked extensively on attention masking for the custom project, after which we switched to the default 3 weeks late. A special thanks to our mentor, Yann Du Bois.

2 Introduction

BERT [4] is a transformer model whose outputted embeddings—when fed into a softmax layer and finetuned for a single task—lead to good performance for tasks like sentiment analysis, paraphrase detection, and similarity analysis [4]. However, a multitask BERT model—that is, a model trained for several downstream tasks—demonstrates disappointing performance when relying on BERT embeddings alone. To address these issues, this paper implements and evaluates a series of extensions to improve a basic BERT multitask model's performance on three tasks: sentiment analysis (SST), paraphrase detection (PARA), and semantic similarity (STS).

To do this, we explore three avenues of improving the model performance. First, we increase the model's representational power by adding ensembling and increasing the number of linear layers appended after the BERT transformer. Second, we add additional loss functions to optimization, including multiple negatives ranking loss, Pearson coefficient loss, and cosine similarity loss. Third, we improve generalizability by implementing LoRA and adding additional data. Ultimately, we find that a model that combines ensembling, LoRA, and all additional loss functions leads to the best possible performance.

3 Related Work

Various researchers have speculated possible reasons for why BERT embeddings perform poorly in multitask settings. Reimers and Gurevych (2019) have argued that the "construction of BERT" (i.e., the self-attention mechanism and autoencoding structure) and the standard loss function for

classification tasks (i.e., cross entropy loss) are fundamentally misaligned for semantic classification [14]. Furthermore, Yu, et. al (2020) argues that several gradients between multiple loss functions in multitask settings can lead to issues where the gradients conflict against each other, preventing an optimal solution that works well over several tasks [17].

To address this low performance, researchers have proposed various extensions to the basic BERT model. Since general techniques for extensions like adding layers and ensembling have already been well-documented to improve performance, we focus instead on more BERT-specific extensions that increase representational power, change the loss function or reduce overfitting. Regarding increasing representational power, Reimers and Gureyvch (2019) [14], which tested various ways to concatenate sentence embeddings u and v for the PARA task, find that concatenating u, v, and $u - v$ as input for the softmax layer led to a 22% score increase from concatenating just u and v. Regarding changing the loss function, gradient surgery, proposed by Hu, et al. (2020) [17], prevents the gradients of multiple loss functions from conflicting with each other, by projecting each gradient onto the normal plane of the other. Yu, et al. (2020) [17] found significant gains in both performance and efficiency from gradient surgery (as aligned gradients are faster to optimize).

Furthermore, cosine similarity should theoretically be an effective scoring function of the semantic similarity of two sentence embeddings. Following this reasoning, multiple negatives ranking loss and cosine similarity loss training should increase the cosine simliarity of equivalent sentences and thus also the model's STS performance. Indeed, Henderson, et al. (2017) found a 20% reduction in STS error rate associated with applying multiple negative rankings loss to BERT [6]. Meanwhile, Reimers and Gureyvch (2019) [14], which implemented cosine similarity loss for a "Siamese" PARA-STS joint model, reaped a score of 87.44 (out of 100) on the STSb benchmark.

To reduce overfitting, Hu, et al. (2021) [8] present LoRA (or Low Rank Adaptation), which decomposes each matrix parameter of the BERT transformer to a set of r dimensions that best capture the parameter's variance. LoRA reduces not just overfitting—as deep learning tasks tend to have an underlyingly low-rank structure, which is better captured by LoRA—but also runtime, as fewer parameters are trained per iteration.[8] Additional studies [7] [12] have found that LoRA does not compromise model performance compared to other BERT fine-tuning techniques.

4 Approach

We present a three-pronged approach to improve the basic BERT model. First, to increase representational power of our model, we add linear layers to our model and implement ensembling. Second, we use additional loss functions such as MSE Loss, Cosine Similarity Loss, and Pearson Coefficient loss to better represent the STS task. Third, to improve generalization, we implement LoRA and add additional training data.. In this section we first describe our baseline, then categorize and present each extension to the baseline we implemented:

4.1 Base Model

Let $m = 798$ be the hidden size of our BERT transformer and $b = 8$ be our batch size. Our base model has the following forward function for each task:

SST: Our input is a batch of b sentences. This input is tokenized by a pretrained tokenizer (provided by the starter code [2]) into an input matrix $T \in \mathbb{R}^{(b \times n)}$. Then, the BERT transformer returns a list of output embeddings for each sentence; we average each sentence's list of embeddings (i.e., mean-pooling) to get embedding matrix $E \in \mathbb{R}^{(b \times m)}$. With one linear layer $W_{\text{SST}} \in \mathbb{R}^{(m \times 5)}$, our output logits are $l = EW_{\text{SST}} \in \mathbb{R}^{(b \times 5)}$.
Our predicted class $\hat{y}$ for SST is the index i, ranging from 0 to 4, that has the highest logit l_i.

PARA: Our input is a batch of b pairs of sentences. Both sentences in the input are tokenized by the tokenizer into two matrices $T_1, T_2 \in \mathbb{R}^{(b \times n)}$, which we then feed to the BERT transformer to return $E_1, E_2 \in \mathbb{R}^{(b \times m)}$. Next we concatenate E_1 and E_2 to get combined matrix $E_{12} \in \mathbb{R}^{(b \times 2m)}$. Now we create parameter $W_{\text{PARA}} \in \mathbb{R}^{(2m \times 1)}$ such that our output logits are $l = E_{12}W_{\text{PARA}} \in \mathbb{R}^{(b \times 1)}$.
Our predicted class $\hat{y} = 1$ if $\sigma(l) \geq 0.5$, else $\hat{y} = 0$.

STS: Like in the PARA forward function, our input is a batch of b pairs of sentences, which we then tokenize and feed into the BERT transformer to return $E_1, E_2 \in \mathbb{R}^{(b \times m)}$. However, instead of concatenation, our output is the cosine similarity of each row in E_1 and E_2, $\hat{y} \in \mathbb{R}^{(b \times 1)}$.

Since our performance in STS is evaluated by Pearson correlation, we simply return $\hat{y}$ as our predicted values for the batch.

The BERT transformer begins with pretrained embeddings given in the starter code ('bert-base-uncased'). Then, we train the baseline for 10 epochs on the train splits of STS, Quora, and SemEval with a constant learning rate of 10^{-5}, using an Adam Optimizer [10] that applies gradient descent with weight decay. As our loss function, we use Cross Entropy Loss for SST and PARA and MSE Loss for STS; we call a step to the optimizer for each batch in each task. Finally, our baseline (and all future models presented) also implement **gradient surgery**, in which we align gradient g_i with conflicting gradient g_j through the following projection: $g_i = g_i - \frac{g_i g_j}{\|g_j\|^2} g_j$.

4.2 Increasing Representational Power

Here we describe the extensions we implemented to increase the representational power of the BERT multitask model:

1. **Increasing Linear Layers.** For the SST and PARA tasks, we replace $W_{\text{SST}}, W_{\text{PARA}}$ each with neural networks $N_{\text{SST}}, N_{\text{PARA}}$, where

$$N_{\text{SST}} = \text{NeuralNet}(\text{n\_layers} = l_{\text{SST}}, \text{input\_size} = m, \text{hidden\_size} = h_{\text{SST}}, \text{output\_size} = 5)$$
$$N_{\text{PARA}} = \text{NeuralNet}(\text{n\_layers} = l_{\text{PARA}}, \text{input\_size} = 2m, \text{hidden\_size} = h_{\text{PARA}}, \text{output\_size} = 1)$$

   We use GeLU activation, input and hidden layer dropout rate of 0.2, and output layer dropout rate of 0.1. $l_{\text{SST}}, l_{\text{PARA}}, h_{\text{SST}}, h_{\text{PARA}}$ become hyperparameters, which we continuously vary per run. [5]
2. **Ensembling.** Our ensembled model trains three separate BERT submodels $M_{\text{SST}}, M_{\text{PARA}}, M_{\text{STS}}$ on their respective task's dataset (STS, Quora, and SemEval). Thus we have three completely different sets of embedding weights for the three tasks. For our forward functions for each of the three tasks, we then return the output of the corresponding submodel. Multiple negatives ranking loss and cosine similarity loss (described below) are only used to train submodel M_{STS}.
3. **Super PARA Concat.** The input of the linear layers of the PARA forward function is changed to the concatenation $[E_1; E_2; E_2 - E_1] \in \mathbb{R}^{b \times 3n}$ (from $[E_1; E_2]$), and W_{PARA} to have size $b \times 3n$.

4.3 Adding Additional Loss Functions

1. **Multiple Negatives Ranking Loss.** We define multiple negatives ranking loss as

$$-\sum_{i=1}^{K} \log \frac{\exp(S(i,i))}{\sum_{k=1}^{K} \exp(S(i,k))}$$
$$S(i,k) = \frac{E_{1,i} \cdot E_{2,k}}{\|E_{1,i}\| \|E_{2,k}\|}$$

   Where $E_{1,i}$ is the embedding of the first sentence and $E_{2,k}$ is the embedding of the second sentence. This score is proportional to the likelihood that s_i is equivalent to t_k. To create the dataset, we filtered the STS dataset to include K pairs of sentence embeddings (s_i, t_i) with a similarity ≥ 4.0. We assume (s_i, t_j) for $i \neq j$ are non-equivalent (similarity < 2.0). Since multiple negatives ranking loss was such a small dataset, we simply apply just one epoch of this as pretraining, before the model is trained on the task datasets.
2. **Cosine Similarity Loss.** Similar to multiple negatives ranking loss, cosine similarity loss training also seeks to increase the cosine similarity of embeddings of equivalent sentences and reduce that of unequivalent sentences. We create a cosine similarity loss dataset using R, consisting of all elements in the STS dataset whose similarity is either ≥ 3.5 or < 1.0. We consider two sentence embeddings e_1 and e_2 extracted from the [CLS] token of size d, where d is the embedding dimension. We calculate the cosine similarity as $\frac{e_1^\top e_2}{\|e_1\| \|e_2\|}$. We then calculate the loss as:

$$\text{loss}(x, y) = \begin{cases} 1 - \cos(e_1, e_2), & \text{if similarity} \geq 3.5 \\ \max(0, \cos(e_1, e_2)), & \text{if similarity} < 1.0 \end{cases}$$

Since the cosine similarity loss dataset was small, we applied just one epoch of this as pretraining before training on the task datasets.

3. **Pearson Loss.** Since performance in the STS task was evaluated by the predicted scores' correlation with the true similarity scores, we switched the loss function used in our STS training from Mean Squared Error to Pearson Coefficient loss, defined as follows. Let $\hat{y}$ be our predicted similarity score from the STS task and y be the true similarity score. Pearson loss is simply the inverse of the Pearson correlation, that is, $-\frac{\text{Cov}(y,\hat{y})}{\sigma(y)\sigma(\hat{y})}$.

While applying the extensions, we continuously try out different learning rates, hidden sizes for the linear layers, and number of linear layers.

4.4 Reducing Overfitting and Improving Generalizability

1. **LoRA.** To prevent overfitting, we implement LoRA as follows: consider any linear layer parameter W in the BERT transformer model $\in \mathbb{R}^{(m \times n)}$, with pretrained weights $W_{pretrain}$. LoRA reparameterizes W as $W_{pretrain} + BA$, with $W_{pretrain}$ fixed as a constant and A, B as parameter linear layers of size $(m, r), (r, n)$ and $r << m, n$. A low r thus reduces the rank of each layer in BERT, decreasing the number of parameters. We implemented LoRA from scratch by modifying the provided implementation's BERT Attention, BERT Layer, and BERT Model classes.
2. **Additional data.** We expand the training set for the Para task (beyond the examples in Quora), namely with the first 50,000 examples in the Duplicate Questions dataset, as described below.

5 Experiments

5.1 Data

We are given our train, dev, and evaluation splits of the following four datasets. First, the *Stanford Sentiment Treebank* (SST-5) [15] contains 11,855 sentences, each labled 0 (negative), 1 (somewhat negative), 2 (neutral), 3 (somewhat positive), or 4 (positive) by a panel of 3 judges. The *Quora dataset* (PARA) [9] contains 404,298 question pairs from the website Quora, labeled 1 (questions are paraphrases of the other) or 0 (questions are not paraphrases of the other). The *SemEval (STS) Benchmark Dataset* [1] contains 8,628 sentence pairs collected from news headlines and cultiral glosses. Each example is labled 0 (completely different topics), 1 (non-equivalent on the same topic), 2 (non-equivalent, share details), 3 (roughly equivalent), 4 (mostly equivalent). Finally, the *Duplicate Questions Dataset* (created by Nils Reimers) [13] provides a publicly available dataset of 6,200,203 pairs of duplicate questions from StackExchange, of which we take the first 50000 of them, labeling them as 1, for positive examples of paraphrases.

5.2 Evaluation Method

The score S_{task} of a given model for each task is defined as follows:

$$S_{\text{SST}} = \frac{\text{Number of correct predictions on } D_{\text{SST}}}{\text{Number of examples in } D_{\text{SST}}} \in [0, 1]$$

$$S_{\text{PARA}} = \frac{\text{Number of correct predictions on } D_{\text{PARA}}}{\text{Number of examples in } D_{\text{PARA}}} \in [0, 1]$$

$$S_{\text{STS}} = \frac{1 + \rho_{\text{STS}}}{2} \in [0, 1]$$

where ρ_{STS} is the Pearson correlation coefficient between the predicted and actual values for the STS task, and D_{SST}, D_{PARA}, and D_{STS} are the dev set splits of SST, Quora, and SemEval.

5.3 Experimental Details

In order to determine which extensions significantly improved model performance, our approach can be split into two stages:

1. Exploratory Phase

Given the extensive number of extensions implemented, systematically evaluating each one was unfeasible due to our time constraints (we switched from the custom project 3 weeks late). We instead unsystematically construct different models that each implement *some* of the extensions described in Section 4. This allows us to get a broad sense of which extensions are effective, which we can then more rigorously evaluate in the experimental phase. We use a learning rate of 10^{-5} for all models and train (for all models but the baseline) on just one task, to speed up computation. Below are the most informative models we tested:

Models Tested on the SST Task (a) *Multi-Layer*: This model uses multiple linear layers, specifically 4 SST layers with a hidden size of 768. We train for 9 epochs. **(b)** *Ensembled*: This model incorporates ensembling with 3 SST layers, each with a hidden size of 70. We train for 10 epochs.
Models Tested on the PARA Task: (a) *LORA-fied*: This model applies LORA with a rank size of $r = 100$ and 2 PARA layers with a hidden size of 10. We train for 10 epochs. **(b)** *BigData*: This model includes additional duplicate question examples in the paraphrase training set and the super PARA concat. It features 2 PARA layers with a hidden size of 100 and LORA with a rank size of $r = 200$. We train for 10 epochs.
Models Tested on the STS Task: (a) *Cosine*: This model is first trained on cosine similarity loss for 1 epoch, followed by an additional 10 epochs of training. **(b)** *NegCosine*: This model implements both cosine similarity and multiple negatives ranking loss for 1 epoch, followed by training for 10 epochs. **(c)** *Pearson*: This model employs Pearson loss instead of MSE loss and is trained for 10 epochs.

2. Experimental Phase

Taking the most effective extensions from the exploratory phase, we now rigorously finetune their hyperparameters in order to generate the highest possible scores. Specifically, for LORA, we iterate rank size r from 0 to 200, keeping all other variables constant. In our experiments, we pick the parameter value associated with the highest SST, PARA, and STS scores (after training on all examples for 10 epochs) as the value to use in our final model. After identifying the extensions and parameter values associated with good performance, we then implement all of them in our final model. We do not change the parameters of the BERT transformer, which has 12 layers and a hidden size $m = 768$.

6 Results

6.1 Final Model

Our final model had the following extensions: 2 SST layers of hidden size 100, 2 PARA layers of hidden size 100, additional Para examples, super Para concat, LoRA with $r = 200$, Pearson loss, cosine embedding loss, and multiple negatives ranking loss. This model, after being trained on 10 epochs with a learning rate of 10^{-5}, yielded the following scores:

SST	PARA	STS
0.531	0.861	0.858

Figure 1: Scores of Final Model on Test Set

Now we present the results of the various experiments we did to get this final model.

6.2 Results from Exploratory Phase

The three figures below demonstrate the results of our preliminary models compared against the baseline. Bolded are models with a higher score than the baseline.

The exploratory phrase reveals that the following extensions lead to significantly high performance: 1) ensembling, 2) LoRA, 3) adding additional examples, and 4) all additional loss functions (Pearson, multiple negatives ranking, and cosine similarity). Meanwhile, adding linear layers is *not* associated

Model	SST Score
Baseline	0.483
Multi-Layer	0.480
Ensembled	**0.521**

Figure 2: SST scores of SST Models, against Baseline.

Model	PARA Score
Baseline	0.782
Lora-fied	**0.816**
BigData	**0.861**

Figure 3: PARA scores of PARA Models, against Baseline.

Model	STS Score
Baseline	0.352
Cosine	**0.422**
NegLoss	**0.362**
Pearson	**0.809**

Figure 4: STS scores of STS Models, against Baseline.

with higher performance. These high-performance-associated extensions are thus assessed more systematically in the next section.

6.3 Results from Experimental Phase

6.3.1 Ensembling

To rigorously assess the extent of performance increase associated with ensembling, we compare a model with ensembling versus one that doesn't, keeping all other hyperparameters constant (e.g., learning rate, epochs, hidden sizes and linear layers). Speciifcally, we have 2 SST layers and 2 PARA layers, both with hidden size 100. We implement LoRA with $r = 200$. We do not include Pearson loss, or any other additional loss function.

Ensembling	SST	Para	STS
No	0.508	0.760	0.481
Yes	0.521	0.788	0.338

Figure 5: Model with vs. without ensembling. Each model has 2 SST layers and 2 PARA layers, both with hidden size 100, LoRA with $r = 200$, and no additional loss functions.

The ensembled model has a significant increase in SST and PARA scores, along with a major decrease in the STS score (from 0.481 to 0.338). We believe that the lower STS score in the ensembled model is due to overfitting, since the STS test dataset is far smaller than the other task datasets, allowing a BERT model that *just* trained on STS to pick up noise in the training data as signal. Furthermore, one reason why ensembling improves SST scores is because sentiment analylsis is rather different from both paraphrase detection and similarity analysis. Specifically, while both PARA and STS revolve around the *meaning* of a given sentence, SST revolves around the *affect* of a given sentence; thus, SST may perform better when using a completely different BERT transformer than PARA and STS.

6.3.2 LoRA

Now we determine what rank size *r* for LoRA is associated with highest performance. To recap, a very low *r* (e.g., $r = 2$) severely reduces the number of model parameters, causing the model to underfit, while a high *r* (e.g., $r = 700$) may barely reduce parameters whatsoever. Figure 6 shows STS scores with $r = \{20, 100, 150, 200, 300\}$, where we can see that a rank size of 200 is associated with the highest STS score. However, we note that SST and PARA may have different ideal r values than that of STS.

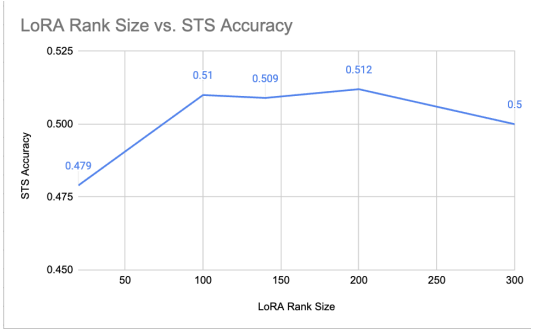

Figure 6: LoRA rank size vs. STS accuracy. Varying r from 20 to 300, with multiple negatives ranking loss, learning rate 10^{-5}, and no other extensions trained on 10 epochs.

6.3.3 Additional Loss Functions

Clearly, Pearson loss improves our STS task score, leading to a far higher score increase than cosine similarity loss and multiple negatives ranking loss combined (Figure 4). However, to what extent do these latter two additional loss functions increase our STS task score? Figure 7 shows STS task performance associated with applying each of these additional loss functions individually (and then both), keeping all other parameters constant.

Loss	**STS**
Pearson + Cosine Loss	0.809
Pearson + Negative Rankings Loss	0.822
Pearson + Cosine and Negative Rankings Loss	0.826

Figure 7: STS Scores for Additional Loss Functions, combined with Pearson

Multiple negatives ranking loss is associated with a higher increase in performance than cosine similarity loss. However, Figure 7 shows that both additional loss functions increase STS score: in other words, cosine similarity loss, though worse than multiple negatives ranking loss, does not decrease STS performance.

7 Analysis

Here we break down the performance of our final model by task.

7.1 SST Error Rate and Performance

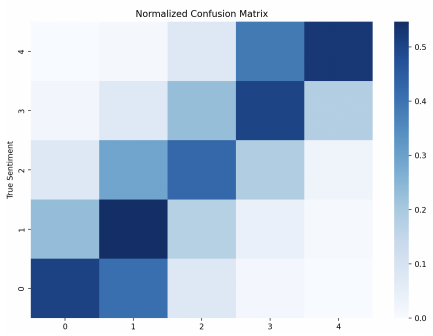

Figure 8: Confusion matrix on SST task of final model.

Figure 8 presents a confusion matrix of the SST task, in which values on the diagonal represent the frequency of correct classifications, in order to see whether the SST task performs better on specific labels than others.

We find that more misclassifications occurred with classifications involving sentiment values of 2 and 3 (milder sentiments), than those involving sentiment values 0 and 4 (more polarized sentiments). We predict that this happens because the nuances between these milder sentiments are more subtle and thus harder to categorize: there may be a variety of features that define neutrality and "somewhat" positiveness (e.g., hedging, milder adjectives, use of *but* or *although*, a specific ratio of positive to negative adjectives), while features that mark extreme positivity or negativity may be more limited and easily identifiable (e.g., using intensifiers like *very* and strong adjectives that are overwhelmingly either positive or negative). In other words, highly positive or negative language may be much easier to identify and categorize than mildly positive or negative language, meaning fewer features that the model needs to learn.

7.2 PARA Task Error Rate and Performance

True Positive	False Positive	True Negative	False Negative
0.83	0.15	0.85	0.17

Figure 9: TP, FP, TN, and FN rates of final model on PARA task

From Figure 9, we see that the model has essentially the same rate of false positives as it does false negatives; in other words, it performs about the same for both paraphrase and non-paraphrase examples. This suggests that the model was trained on enough positive and negative paraphrase examples in the training data, such that PARA performance was balanced across classes.

7.3 STS Performance

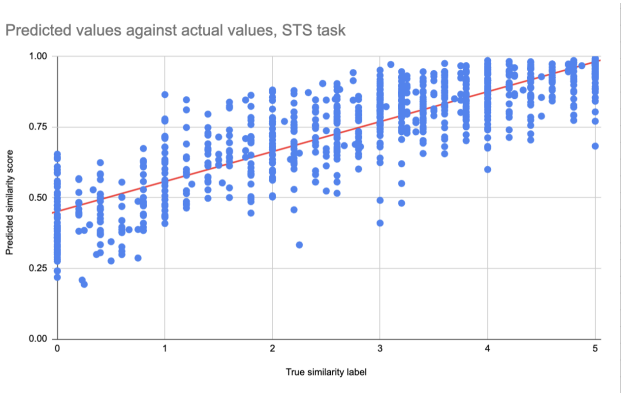

Figure 10: Predicted scores against actual labels on STS task of final model. Included line of best fit to gauge one solution of $\rho = 1$.

From Figure 10, we see not much heteroskedasticity—that is, the model having greater error for some similarity labels than others. There is *slightly* greater error around similarity labels of 3; like in Section 7.2, examples that are partly, but not entirely, similar to each other may be harder to classify than those that are clearly equivalent or nonequivalent. These middle-of-the-range labels were also not targeted by cosine similarity loss or negative rankings loss (which focused on simialrity scores > 3.5 or < 1), rendering them weak points in our model.

8 Conclusion

We explored multiple extensions to improve the performance of a BERT-based multitask model on sentiment analysis, paraphrase detection, and semantic textual similarity tasks. Our approach centered around increasing the representational power of the model, optimizing loss functions, and enhancing

generalizability. Ultimately, after exploring a variety of different models and evaluating a specific set of hyperparameters, we found that the following extensions were associated with significant performance gains: ensembling, LoRA, adding additional examples to the PARA training split, Cosine Similarity Loss, Multiple Negatives Ranking Loss, and Pearson Loss; by contrast, additional linear layers did not lead to significant performance gains.

9 Ethics Statement

One ethical challenge of our BERT multitask model is its potential to replicate the same racist and sexist biases that appear in its training data. As we learned in Adina Williams's lecture "An Introduction to Responsible NLP" [16], models trained for the STS task often assign higher sentiment to sentences with male pronouns than those with female pronouns (e.g., *He yelled, "Come over here"* vs. *She yelled, "Come over here"*). Similarly, a systematic evaluation of 219 sentiment analysis models (each trained to categorize the affect of a given sentence) by Kiritchenko and Mohammad (2018) [11] identified not only a gender bias in sentiment analysis models, but also a racial bias, where sentences with African American names returned higher scores on anger, fear, and sadness than those with European American names (p. 8). Given that our data, like the datasets described in Kiritschenko and Mohammad (2018), comes from the Internet—which on a whole possesses racist/sexist biases—it is likely that it will possess the same racist/gender biases when assigning sentences sentiment value, thus perpetuating these biases upon widespread use. This requires proper equity correction during training, in which the model learns to separate gender/racial attributes from sentiment judgements by masking and randomly varying these gender/racial attributes per sentence (as described in Wililams' lecture [16]).

Another ethical challenge of our project is simply the sheer amount of energy consumption required to train and finetune our BERT transformer model. LLMs like BERT are particularly computationally expensive due to the large amount of training data needed to have good performance: for our project alone, we had to run several A100 and T4 models at once for several days nonstop, obviously an energy-wasteful process. On a broad scale, the tech industry's race to amass large bodies of data for LLMs has led to significant repercussions to our climate: the UChicago professor Andrew A. Chien (2020) estimates that datacenters used for LLMs alone have released over 3.25 gigatons of CO2 per year, the equivalent of five billion U.S. crosscountry flights [3]. Though we obviously don't possess the same amount of data as the biggest tech industry players, it is important that we limit the amount of energy we consume when developing our personal NLP models, by testing our model initially on smaller subsets of our data and using more energy-efficient GPU instances.

References

[1] Eneko Agirre, Daniel Cer, Mona Diab, Aitor Gonzalez-Agirre, and Weiwei Guo. *sem 2013 shared task: Semantic textual similarity. *Proceedings of *sEM*, pages 32–43, 01 2013.

[2] amahankali. Starter code for default final project, spring 2024. 2024.

[3] Andrew A. Chien. Genai: Giga$$$, terawatt-hours, and gigatons of co2. *Commun. ACM*, 66(8):5, jul 2023.

[4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

[5] Shuxuan Guo, Jose M. Alvarez, and Mathieu Salzmann. Expandnets: Linear over-parameterization to train compact convolutional networks. *arXiv preprint arXiv:2003.05664*, 2020.

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

[7] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for nlp, 2019.

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

[9] Shankar Iyer, Nikhil Dandekar, and Kornél Csernai. First quora dataset release: Question pairs. Data at Quora, 2012.

[10] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[11] Svetlana Kiritchenko and Saif M. Mohammad. Examining gender and race bias in two hundred sentiment analysis systems, 2018.

[12] Isabella Olariu, Cedric Lothritz, Jacques Klein, Tegawendé Bissyandé, Siwen Guo, and Shohreh Haddadan. Evaluating parameter-efficient finetuning approaches for pre-trained models on the financial domain. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 15482–15491, Singapore, December 2023. Association for Computational Linguistics.

[13] Nils Reimers. Stack exchange duplicate questions. `https://public.ukp.informatik.tu-darmstadt.de/reimers/sentence-transformers/datasets/paraphrases/`.

[14] Nils Reimers and Iryna Gurevych. Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan, editors, *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3982–3992, Hong Kong, China, November 2019. Association for Computational Linguistics.

[15] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In David Yarowsky, Timothy Baldwin, Anna Korhonen, Karen Livescu, and Steven Bethard, editors, *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642, Seattle, Washington, USA, October 2013. Association for Computational Linguistics.

[16] Adina Williams. An introduction to responsible nlp, 2024.

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

A Appendix

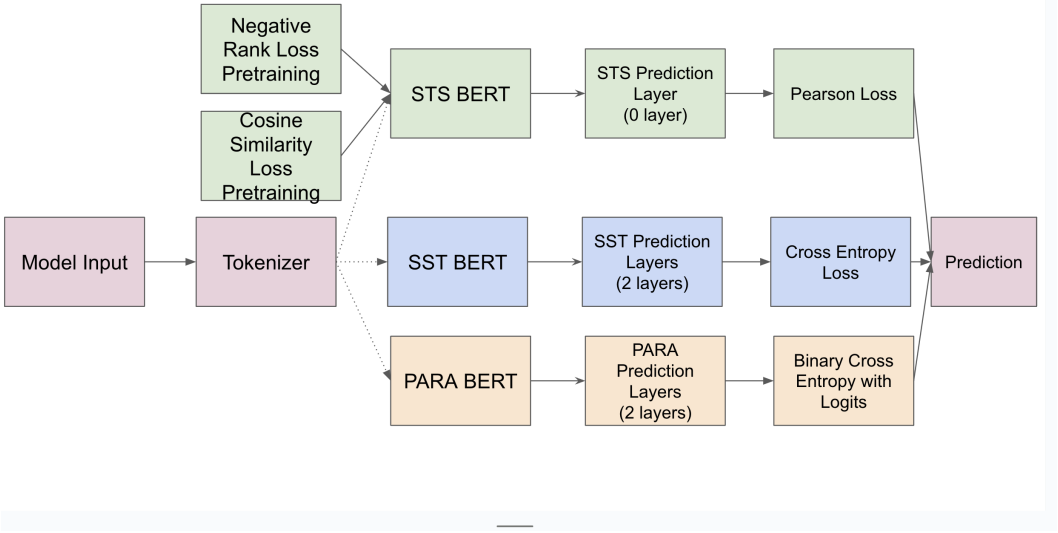

Figure 11: Diagram of our final model, with all extensions added