Robust Embeddings using Contrastive and Multiple Negatives on BERT

Stanford CS224N Default Project

Shoaib Mohammed
Department of Computer Science
Stanford University
shoaibmh@stanford.edu

Ishan Sabane
Department of Electrical Engineering
Stanford University
ishancs@stanford.edu

Abstract

We build a multi-tasking BERT model to perform well on three downstream tasks: sentiment analysis, paraphrase detection, and semantic textual similarity. We benchmark our performance with BERT and we were able to achieve similar results on both a multi-tasking model as well as a model fine-tuned on each individual task. We implemented additional pretraining of the BERT base model leveraging supervised contrastive learning and multiple negatives ranking loss. This improves performance on paraphrase detection and semantic textual similarity.

1 Key Information to include

- External collaborators (if you have any): N/A
- External mentor (if you have any): N/A
- Sharing project: N/A

2 Introduction

In multitask learning, learning two tasks at the same time with their own separate inductive biases produces an overall model that aims for one inductive bias — an inductive bias that optimizes for the two tasks at the same time. This approach may lead to better generalization properties of the separate tasks, meaning that the resulting classifier can handle unseen data better. Often that classifier turns out to be a stronger classifier for both separate tasks [1].

Paraphrase detection (PD) and semantic textual similarity (STS) task are highly correlated in terms of their optimization requirement. Training on both the tasks simultaneously would result in a generalizable model whose embeddings could be used as features for other downstream tasks. The sentiment analysis task trains the model to learn meaning/sentiment of individual sentences which could be helpful for identifying similar sentence pairs since they would have the same sentiment. This kind of knowledge transfer is beneficial collectively.

The rapid development of Large Language Models has proved to be beneficial for multitask learning. Most of the recent work has focused on using pretrained encoders with individual prediction heads for each specific task [2]. We experiment on top of this approach by modifying the prediction heads as well sharing multiple layers between them. The training method of fine-tuning the entire model on all the tasks affects the multitasking performance of the model to a great extent compared to individual fine-tuning performance [3]. After experimenting with different training regimes, our final model achieves similar performance as fine-tuning individual models for each task on the dataset provided.

3 Related Work

We see that fine-tuning BERT base model on individual tasks achieves good results. However, these models work well only on the specific task on which they were trained and produce poor results on other tasks. We refer to some of the prior work in multitask training [2], improving BERT base embeddings [4] as well as using different loss function for paraphrase detection [5].

Generating universal sentence embedding is a challenging problem in NLP. BERT which is an encoder based architecture works well for generating sentence embeddings for multiple downstream tasks. One method to improve the performance on STS and paraphrase detection is to learn sentence embeddings such that the model learns by pulling semantically close neighbors together and pushing apart non-neighbor sentence embeddings [4]. A similar approach is followed by the negative ranking loss function [5]. We can use this loss to train the paraphrase detection sentence pairs as other sentences in the batch cannot be a paraphrase of the given sentence pair.

3.1 CSE: Contrastive Learning for NLP Fine-Tuning [4]

"SimCSE" is a method which builds robust sentence embeddings for downstream tasks using either supervised or unsupervised learning approach. The unsupervised approach involves predicting the same input sentence using a contrastive loss as the objective function with standard dropout used as noise, which proves to be effective. The supervised approach incorporates annotated pairs from natural language inference datasets, using the entailment pairs as positives and contradiction pairs as negatives for the contrastive learning loss.

Unsupervised Contrastive Learning:
Consider a collection of sentences $\{x_i\}_{i=1}^m$ The unsupervised framework uses $x_i^+ = x_i$ to create a sentence pair (x_i, x_i^+). Now, a mask is used to drop random tokens from both the sentences. $h_i^z = f_\theta(x_i, z)$ where h_i^z is the masked sentence embedding and z is a random mask generated using dropout noise method with $p = 0.1$. The loss for a batch of N sentences is given as follows:

$$\mathcal{L} = -\log \frac{e^{\text{sim}(h_i^{z_i}, h_i^{z_i^*})/\tau}}{\sum_{j=1}^N e^{\text{sim}(h_i^{z_i}, h_j^{z_j^*})/\tau}}$$

Supervised Contrastive Learning:
Consider a set of paired examples $D = \{(x_i, x_i^+)\}_{i=1}^m$, where x_i and x_i^+ which are semantically related sentences. Let h_i and h_i^+ be the embedding representations of the two sentences. The sentence pairs are extend to triplets by adding negative examples. Certain datasets have negative examples which can be used directly. Hence, we convert the pair (x_i, x_i^+) to (x_i, x_i^+, x_i^-) Then the contrastive loss function for N sentence pairs is given as follows:

$$\mathcal{L} = -\log \frac{e^{\text{sim}(h_i, h_{i+})/\tau}}{\sum_{j=1}^N (e^{\text{sim}(h_i, h_j^+)/\tau} + e^{\text{sim}(h_i, h_j^-)/\tau})}$$

The above method of adding hard negative examples improves the performance for semantic textual similarity [4]. We further pretrain the BERT base model using the supervised contrastive learning approach on the SNLI dataset [6].

3.2 Efficient Natural Language Response Suggestion for Smart Reply [5]

To get a specific response y from an input email x, we have to model $P(y|x)$. A set of K possible responses are used to approximate $P(y|x)$. It is important to note that the set K contains a single correct response and $K - 1$ random negative responses. To elaborate upon this further, there are K input emails $\mathbf{x} = (x_1, x_2, \cdots, x_K)$ and we have their corresponding responses $\mathbf{y} = (y_1, y_2, \cdots, y_K)$. Any response y_j is a negative response for x_i if $i \neq j$. The approximate probability can be given by:

$$P_{\text{approx}}(y|x) = \frac{e^{S(x,y)}}{\sum_{k=1}^K e^{S(x,y_k)}}$$
Negative Sampling

$$P(y|x) = \frac{P(x,y)}{\sum_k P(x,y_k)}$$
Actual Probability of the response

The goal of training is to minimize the approximated mean negative log probability of the data. If we have a single batch, this would be the following:

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

where θ represents the word embeddings and neural network parameters used to calculate S. It is helpful to know that $S(x, y)$ is learned upto an additive term such that it does not affect the `arg max` over y. The above loss function essentially minimizes the distance between the pair (x_i, y_i) and maximizes the distance between the pair (x_i, y_j) if $i \neq j$. We use this approach for paraphrase detection and assume the batch sentence to be negative samples for a given sentence.

3.3 Multi-Task Deep Neural Networks for Natural Language Understanding [2]

We see that the the method of fine-tuning the multitask model greatly affects the convergence of the total model loss function. This paper uses a random sampling of sentences to generate a training batch. It then computes the loss for each task using the sentences which correspond to that specific task in the batch. The final gradient is computed on the overall loss function which is the average of individual task losses.

Algorithm 1 MT-DNN Multitask Training on Classification Regression and Ranking

Initialize model parameters θ randomly.
Pre-train the shared layers (i.e., the lexicon encoder and the transformer encoder).
for $t \in \{1, 2, \cdots, T\}$ **do**
  1. Merge all the datasets: $D \leftarrow D_1 \cup D_2 \cup \cdots \cup D_T$
  Shuffle D
  for $b_t \in D$ **do**
    // b_t is a mini-batch of task t.
    3. Compute loss: $L(\theta)$
    $L(\theta) \leftarrow$ Eq.6 for classification
    $L(\theta) \leftarrow$ Eq.7 for regression
    $L(\theta) \leftarrow$ Eq.8 for ranking
    4. Compute gradient $\nabla(\theta)$
    5. Update model: $\theta \leftarrow \theta - \epsilon\nabla(\theta)$
  end for
end for

We add weightage to each dataset depending on its size to allow the model to learn across the whole aggregation of the different datasets.

4 Approaches

We build our BERT base model by implementing the sentence tokenization, self multi-headed attention and AdamW optimizer in the initial phase of the project. We experimented **six approaches** for multitask classification. At a high level, these approaches analyse the performance changes as a result of changes in the input to the prediction heads, changes in the prediction heads for each task, addition of related works and additional pre training through SimCSE. The experiments for the same approach test across the dropout level, batch size, learning rate, and total epochs. Below provides more information on the different approaches.

Vanilla Multitask (VM): Using the starter code, we implemented the prediction heads for each task as a single linear layer followed by an activation function. For SST, we use a simple linear layer with CrossEntropyLoss. For paraphrase detection and STS, we pass the concatenated embeddings of the individual sentences to a linear layer to get a single logit with BCELoss and MSELoss respectively.

Sequential Training (ST) and Round-Robin Training (RR): In a sequential training method, we train the model one task at a time for all epochs by following a particular task ordering. In the round-robin method, we train the model with a fixed ordering of task inside a single epoch.

Dense Prediction Heads (DPH): We improve paraphrase detection and STS by concatenating the input ids and attention masks of the given sentences and pass it to the BERT model to get a single output embedding. We repeat this by switching the ordering of concatenation. We then take the element-wise difference between these two embeddings to create additional embedding of the same size. These three embeddings are passed to individual linear layers to get single logits which get passed to a linear layer to get a final logit. We denote prediction head without the difference embedding as **(PH)** and the prediction head with activation function in between as **(DPHA)**.

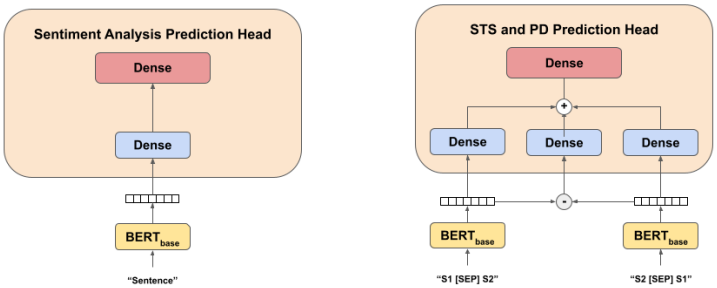

Figure 1: Sentiment analysis uses a simple prediction head which involves a single dense layer before the prediction layer. The STS and PD prediction heads use three dense layers, two for the two sentence ordering and one for the difference.

Parameter Sharing (CL): To ensure that our model does not overfit on the training dataset we add a common linear layer followed with Tanh activation. This is the first layer of each task prediction head. We try adding dropout to this layer but then the model is unable to learn efficiently as we use a dropout probability of 0.5.

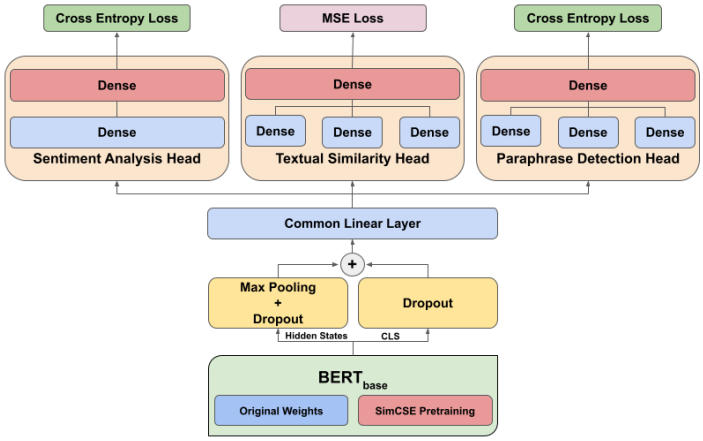

Figure 2: Each blue dense layer is followed by a `Tanh` Activation. The final dense layer is the task specific layer which outputs the logits.

Random Batch Sampling (MT-DNN): The results suggest that the sequential training affects the previous task metrics and the individual task performances drop. Using the MT-DNN approach, we randomly sample the task to train at a given epoch as well as the sentences in the training batch from the data-loader. As paraphrase detection requires larger dataset to achieve good performance, we give twice the importance to paraphrase detection task. To optimize the gradient addition, we implement a method known as gradient surgery(GS) [7]. We set the final gradient as the mean of the input gradients.

SIMCSE Pretraining: We train the BERT base model for 100 epochs on the SNLI dataset to improve the embedding representations for tasks involving sentence similarity.

5 Experiments

We use the gradient surgery approach of adding the gradients efficiently for faster convergence [7]. We experiment with cosine learning rate scheduling, gradient clipping and different dropout values for the pooler output layer. We experiment by increasing the epochs while increasing the dropout value and adding more weight decay to optimizer to prevent over-fitting as the training dataset is small for Sentiment Analysis and Sentence Textual Similarity tasks.

Data

We use the default datasets: Stanford Sentiment Treebank (SST) [8] dataset for sentiment analysis, the Quora Question Paraphrase (QQP) [9] for paraphrase detection, and Semantic Textual Similarity Benchmark dataset [10]. Table 1 shows the data splits.

split	SST	SemEval STS	Quora	SNLI
train	8,544	6041	141,506	550,152
dev	1,101	864	20,215	10000
test	2,210	1726	40,431	10000

Table 1: Dataset sizes for different splits {`train`, `dev`, `test`}

Evaluation Metrics

We use accuracy as the evaluation metric for sentiment analysis and paraphrase detection. We use Pearson's correlation coefficient as the evaluation metric for semantic textual similarity task. We take the average of SST accuracy, STS pearson correlation coefficient, and the paraphrase detection accuracy as our score to store the model whenever it improves. This ensures that we do not store the model when it improves on a single task but performs poor across all the three tasks.

5.1 Experimental details

Our final approach uses MT-DNN approach for fine-tuning the model. The model architecture involves single dense layer in parameter sharing followed by individual task specific prediction heads. We fine-tune the model for 5 epochs with a learning rate of 10^{-5} and batch size of 32. We use the additionally pretrained BERT base model using the SIMCSE approach.

5.2 Results

As can be seen in Table 3, round-robin training significantly improves performance from the vanilla model. Integrating MT-DNN using random batch sampling provided best results across all three tasks. Finally, combining SimCSE with MT-DNN provided similar results. We also performed really well

Exp	Method	Training Method	Train Order	Epochs	lr	Batch Size	dropout
1	VM	Pretrain	SST only	10	1.00E-05	32	0.3
2	RR+PH	Finetune	QOP,SST,STS	3	1.00E-05	32	0.5
3	RR+PH	Finetune	SST,STS,SST,STS,QQP	3	1.00E-05	32	0.5
4	RR+PH +CL	Finetune	STS,SST,QQP,SST, STS	3	1.00E-05	16	0.5
5	RR+PH +CL	Finetune	STS,SST,QQP,SST, STS	3	1.00E-05	32	0.5
6	RR+PH +CL(Dropout)	Finetune	STS,SST,SST,STS,QQP	5	1.00E-05	32	0.5
7	MT-DNN+PH +CL	Finetune	Equal weight	5	1.00E-05	32	0.5
8	MT-DNN+PH +CL	Finetune	Equal weight	5	1.00E-05	64	0.5
9	MT-DNN+DPH(PD) +CL	Finetune	Equal weight	5	1.00E-05	64	0.5
10	MT-DNN+DPHA(PD) +CL	Finetune	2x weight for QQP	5	1.00E-05	64	0.5
11	MT-DNN+DPHA +CL	Finetune	2x weight for QQP	5	1.00E-05	64	0.3
12	MT-DNN+DPHA + CL+ GS	Finetune	2x weight for QQP	5	1.00E-05	8	0.1
13	MT-DNN+DPHA +CL	Finetune	2x weight for QQP	5	1.00E-05	64	0.5
14	**MT-DNN+DPHA +CL**	**Finetune**	**2x weight for QQP**	**5**	**1.00E-05**	**64**	**0.1**
15	MT-DNN+DPHA +CL	Finetune + SimCSE	2x weight for QQP	3,5	1.00E-05	64	0.1

Table 2: Experimental model and training details for each approach

on the leaderboard ranking in the **top 3 on test set**: *0.537* on SST, *0.883* on PD, and *0.887* on STS achieving an overall score of **0.769**. We were also able to rank in the **top 10 on dev set**.

Exp	Method	SST		STS		QQP		Average
		Train	Dev	Train	Dev	Train	Dev	
1	VM	0.393	0.388	0.008	-0.009	0.235	0.38	0.253
2	RR + PH	0.491	0.327	0.883	0.866	0.824	0.764	0.652
3	RR + PH	0.859	0.498	0.876	0.851	0.921	0.874	0.741
4	RR + PH + CL	0.876	0.483	0.483	0.878	0.891	0.81	0.724
5	RR + PH + CL	0.846	0.499	0.948	0.875	0.896	0.822	0.732
6	RR + PH + CL (Dropout)	0.964	0.454	0.968	0.864	0.935	0.794	0.704
7	MT-DNN + PH + CL	0.977	0.491	0.985	0.866	0.85	0.843	0.733
8	MT-DNN + PH + CL	0.968	0.51	0.988	0.871	0.855	0.851	0.744
9	MT-DNN + DPH(PD) + CL	0.986	0.521	0.99	0.875	0.845	0.842	0.746
10	MT-DNN + DPHA(PD) + CL	0.985	0.52	0.986	0.87	0.898	0.851	0.747
11	MT-DNN + DPHA + CL	0.989	0.521	0.986	0.872	0.896	0.852	0.748
12	MT-DNN + DPHA + CL + GS	0.858	0.51	0.962	0.882	0.891	0.876	0.756
13	MT-DNN + DPHA + CL	**0.993**	0.511	0.993	**0.885**	**0.946**	**0.883**	0.76
14	**MT-DNN + DPHA + CL**	0.99	**0.526**	**0.994**	0.879	0.931	0.88	**0.762**
15	MT-DNN + DPHA + CL + SimCSE	0.991	0.508	0.991	0.881	0.95	0.879	0.756
	Test set	N/A	0.534	N/A	0.885	N/A	0.887	**0.769**

Table 3: Experimental results for each approach across all three tasks for *train* and *dev* sets.

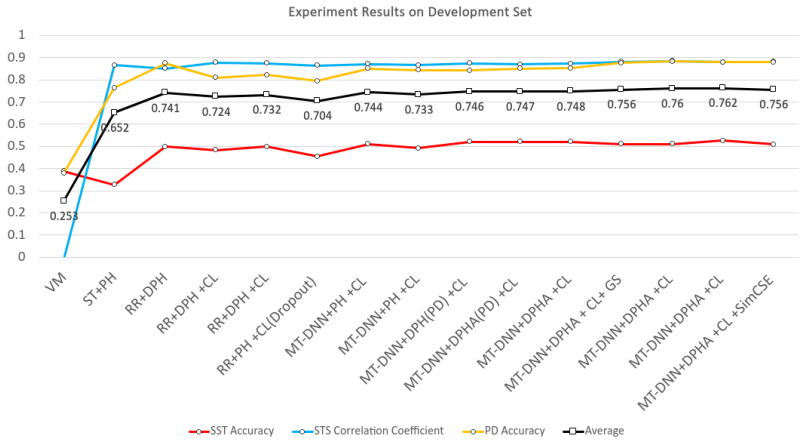

Figure 3: Results for each task across all approaches. Round-robin had slightly lower performance for PD when compared to STS. MT-DNN was able to improve both PD and STS with similar performance. SST performs similar across round-robin and MT-DNN.

6 Analysis

To understand the effect of different approaches in our implementation, we divide our quantitative analysis into three parts.

6.1 Model Architecture Analysis

BERT Base Output: We fine-tune the model for each task individually to compare the effect of using pooler output and the average pooling of the last hidden state. We see that the pooler output performs better than the last hidden state on all the three tasks following the fine-tuning setup [3]. This is due to the variable sentence length. Averaging uses equal weightage to each sentence token, whereas the pooler output learns the entire sentence embedding using the self attention mechanism.

Sentiment Analysis Prediction Head: The BERT pooler output provides rich embeddings which give us the baseline performance reported in the BERT paper. Addition of more linear layers with

activation on top of the pooler output does not provide significant improvements on the SST dev set. Due to the smaller training dataset size, we get almost full training accuracy and we address over-fitting in the subsequent improvements.

PD and STS Prediction Head: Our initial method of concatenation of the pooler output of the individual sentence embeddings leads to very low correlation coefficient for STS and random classifier accuracy for PD. This is due to the BERT training methodology. To understand the sentence correlation, we then pass the two sentences by joining them together with the separator token. Using this approach and a single linear layer, we achieve metrics similar to the BERT baseline on individual tasks.

Common Linear Layer: As we train the model on each task specific dataset, the model starts to overfit the smaller datasets of SST-5 and STS. To prevent the model from memorizing the whole training dataset, we add a common layer shared across the three tasks. This restricts each task specific from heavily updating the BERT weights during fine-tuning. The method provides us excellent results on the test set.

6.2 Training Method Analysis

Sequential Training: As we train the model for each task one after the other for all the epochs, it leads to a bias in the multitasking model towards the last trained task. This is because the gradient descent optimizes the last training loss function while ignoring the other two. STS performs the best since the model is trained on STS dataset in the end *(Exp 2)*.

Round Robin Training: To reduce the training bias toward the last task, we use round robin training in a single epoch. This allows the model to learn the parameter for each task in a single epoch. This simple approach improves the model performance significantly *(Exp 3)*.

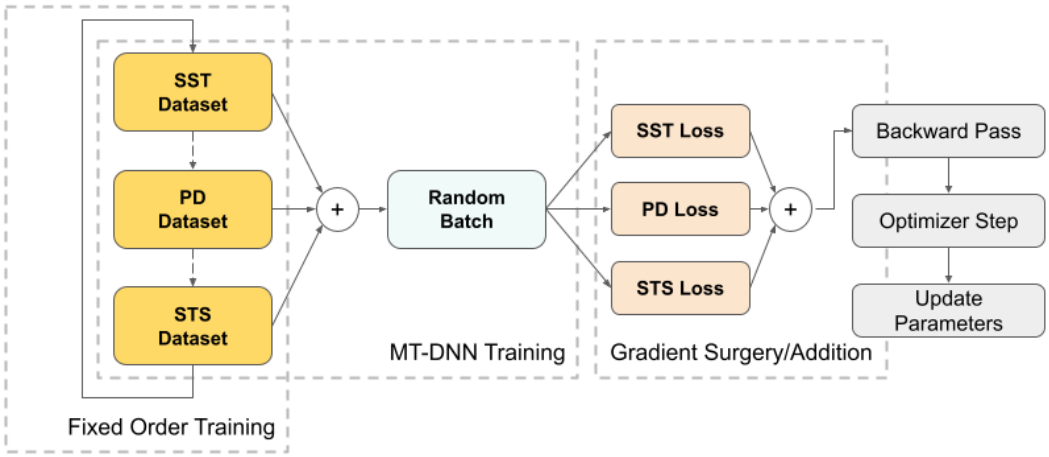

Figure 4: Visualization of training methods for fine-tuning our model

Random Batch Sampling: Changing the training parameters does not help in improving the average above 0.74. Sampling batches from all the three datasets proves to be the most efficient way of training the multitasking model. The gradient descent across the three loss functions in each batch converges efficiently to minimize the training loss for all the three tasks.

Gradient Surgery: We try to implement gradient surgery which adds the task specific gradients towards the correct direction. However, this method is memory heavy and we cannot exceed batch size of 8 for each task.

SimCSE Pretraining: The additional pretraining of BERT base model allows us to obtain richer individual sentence embedding of sentence similarity tasks. This allows us to find a similar sentence near the neighbourhood of the given sentence embedding.

6.3 Incorrect Prediction Analysis

Sentiment Analysis: Most of the incorrect classification in table 4 happens because of positive/negative words that influence the prediction. Phrases such as *complete lack*, *maudlin*, *silliness*, *unwatchable*, and *half-baked* affect the output considerably.

Sentence	Expected	Prediction
complete lack of originality , cleverness or even visible effort	1	0
more maudlin than sharp .	2	1
even film silliness needs a little gravity , beyond good hair and humping .	2	1
beautifully made piece of unwatchable drivel .	2	1
it 's everything you do n't go to the movies for .	1	3
at once half-baked and overheated .	2	1

Table 4: Incorrect classification for sentiment analysis

Paraphrase Detection: The model gets confused when sentences differ in meaning due to subtleties. For instance, at first glance most of the sentence pairs in table 5 sound the same except when analyzing the meaning. For example, *your measure* and *the measure* mean two different things but sound similar.

Sentence 1	Sentence 2	Expected	Prediction
what is your measure of success ?	what is the measure of success ?	0	1
how do i reset tinder ?	how do i download tinder ?	0	1
how do i lose fat from my legs ?	how can i lose leg fat ?	0	1
is anyone really happy ?	who is really happy ?	0	1
how important is your middle name ?	what is/are your middle name(s) ?	1	0
how do you highlight your own hair ?	how can i highlight my hair at home ?	1	0
how do i convert mg to ml ?	how can i convert 15 mg to ml ?	1	0
why do you think people fall in love ?	how do people fall in love ?	1	0

Table 5: Incorrect classification for paraphrase detection

Semantic Textual Similarity: Similar to PD, the model wrongly predicts higher correlation when sentences sound similar but contextually are different. For instance, in table 6 *open lower* and *close lower* are different but the model classifies them as similar. We also see similarity is higher if sentences have many common words. For example, there are $4/5$ common words from sentence 1, *a woman opens a window*, in sentence 2, *a woman is looking out a window*.

Sentence 1	Sentence 2	Expected	Prediction
work into it slowly .	it seems to work .	0	2.54
indian stocks open lower	indian stocks close lower	1.8	4.21
yes , you should mention your experience .	yes , you should make a résumé .	2	0.65
it 's not a good idea .	no , it 's not a good thing .	4	2.26
two lambs stand on a grassy hill .	two dirt bikers riding over dirt hill .	0.2	2.06
the man is buttering the bread .	the man is stirring the rice .	0.4	1.43
a woman is dancing .	the man is dancing .	2.6	1.6
a woman opens a window .	a woman is looking out a window .	2	3.01

Table 6: Incorrect classification for semantic textual similarity

7 Conclusion

We successfully implement multi-headed self attention, transformer block, and the AdamW optimizer for a BERT base model. Our model achieves same performance on SST and SemEval datasets as the BERT base model. We develop a multitasking model which uses this BERT model as the sentence encoder. To improve multitasking performance across three tasks, we use parameter sharing, random batch sampling, gradient surgery, and additional pretraining. We also learnt richer embeddings using constrastive learning but do not see any considerable improvements. Our approach achieves better results on STS and paraphrase detection than the BERT baseline and similar results on SST-5. This shows that multitasking approach is beneficial and can be extended to include additional tasks [11].

References

[1] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[2] Xiaodong Liu, Pengcheng He, Weizhu Chen, and Jianfeng Gao. Multi-task deep neural networks for natural language understanding. *arXiv preprint arXiv:1901.11504*, 2019.

[3] 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.

[4] Tianyu Gao, Xingcheng Yao, and Danqi Chen. Simcse: Simple contrastive learning of sentence embeddings. 2021.

[5] 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.

[6] Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. A large annotated corpus for learning natural language inference. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, 2015.

[7] 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.

[8] Papers with Code - SST-5 Fine-grained classification Benchmark (Sentiment Analysis) — paperswithcode.com. https://paperswithcode.com/sota/sentiment-analysis-on-sst-5-fine-grained. [Accessed 04-Mar-2023].

[9] Papers with Code - Quora Question Pairs Dataset — paperswithcode.com. https://paperswithcode.com/dataset/quora-question-pairs. [Accessed 04-Mar-2023].

[10] Papers with Code - STS Benchmark Dataset — paperswithcode.com. https://paperswithcode.com/dataset/sts-benchmark. [Accessed 04-Mar-2023].

[11] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*, 2018.