Default Final Project: minBERT and Downstream Tasks (Multi-task Learning)

Stanford CS224N Default Project

Samuel Chian
Institute for Computational and Mathematical Engineering
Stanford University
samchian@stanford.edu

Maximilian Sabayev
Institute for Computational and Mathematical Engineering
Stanford University
sabayev@stanford.edu

Abstract

Our goal for this project is to see if a pre-trained minBERT model can create embeddings that generalize well to more than one task, namely: (1) Sentiment Classification, (2) Paraphrase Detection, and (3) Semantic Textual Similarity. In an attempt to get better performance on all three tasks, we implemented multiple methods: Elastic Net regularization, Gradient Surgery for fine-tuning, Cosine Embedding Loss, and some data restructuring. Our findings show that Elastic Net regularization improved Sentiment Classification and Cosine Embedding Loss improved Paraphrase Detection. In our experiments, we also note that the method of inputting paired data into BERT strongly affects the results, with concatenation of tokens pre-BERT significantly outperforming concatenation of embeddings post-BERT. Our results also show that the model is able to generalize well to the three tasks, sometimes even outperforming the single-task model.

1 Introduction

As the size of deep learning models increases, the resources required to train multiple networks increases as well. While multiple different models can be trained to learn specific tasks, it is definitely a space and time-intensive process to train multiple models. Therefore, in order to reduce the amount of resources to solve multiple deep learning problems, it is our goal to apply the principles of transfer learning to see how well we can get a deep learning natural language model to generalize to three different tasks, namely: (1) Sentiment Classification, (2) Paraphrase Detection, and (3) Semantic Textual Similarity. As it is our goal to maximize efficiency, our work focuses on optimizing solely for the model's architecture, simulating an environment where additional resources are not available. This means that we elected not to include additional pre-training for the task or finding additional datasets, as this requires additional resources.

2 Related Work

The main related work to our paper is Gradient Surgery (Yu et al., 2020). During a multi-task training process, there is a possibility of gradients being pointing in opposing directions, making it difficult to traverse the loss surface. Therefore, gradient surgery mitigates this problem by projecting gradient components such that conflicting gradients are mutually orthogonal; this allows the backpropagation process to traverse the loss surface without conflict. Mathematically, suppose we have two gradients

$\boldsymbol{g_i}$ and $\boldsymbol{g_j}$, if $\boldsymbol{g_i} \cdot \boldsymbol{g_j} < 0$ (i.e. the have opposing directions), then we project $\boldsymbol{g_i}$ onto the normal plane of $\boldsymbol{g_j}$:

$$\boldsymbol{g_i} = \boldsymbol{g_i} - \frac{\boldsymbol{g_i} \cdot \boldsymbol{g_j}}{\|\boldsymbol{g_j}\|_2} \boldsymbol{g_j}$$

3 Approach

For the single-task classification, we followed the initial steps described in the default project handout, which includes fully implementing minBERT, the Sentiment Classifier, and the Adam Optimizer, along with obtaining acceptable pre-trained and fine-tuned results on the SST and CFIMDB datasets. We then tackled the main objective of extending our implementation of minBERT to do paraphrase detection and semantic textual similarity, along with the original task of sentiment classification.

Model Architecture To adapt the model for additional downstream tasks, we changed the final output layer of the minBERT model to correspond to the specific task being learned. In particular, since both new methods required taking in two sentences to check their similarity, we concatenated both sentences' tokens, obtaining the BERT embedding for the concatenated tokens, and then fed them to the final layer (see Figure 1).

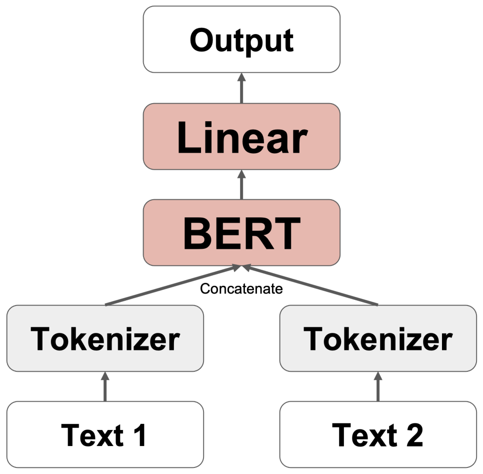

Figure 1: Baseline minBERT Architecture

Baselines Our baseline model for comparison is a minimalist implementation of the BERT model(Devlin et al., 2018). For sentiment classification, we use a linear output layer with 5 classes with cross entropy loss. For paraphrase detection, we use 2 classes with cross entropy loss, and for semantic textual similarity, we use 1 class with mean absolute error as the loss function. In this baseline, we train the tasks concurrently. To do this, we treat one epoch as one iteration through the largest dataset (Quora), and restart sampling of the smaller datasets until the epoch is completed. For backpropagation, the losses were only backpropagated at the end of each minibatch after calculating losses for all three tasks.

Experimental Contributions During the training of our baseline model, we noticed that overfitting was one of the main issues in our implementation, with training accuracies often hitting close to 100%, while the dev accuracy did not keep increasing. Therefore, some of our key experimental contributions involve forms of regularizations, while other contributions involve architectural changes in a hope for the model to learn better embeddings/weights.

1. **Elastic Net** From statistics, there are two popular ways for regularization: (1) variable selection through L_1 penalization (LASSO), and (2) weight shrinkage through L_2 penalization (Ridge). The Elastic Net by Zou and Hastie (2005) is a regularization method that combines both of these ideas that we implemented in our experimentation. By shrinking the model weights of the output layer, this allows the model to learn more general trends of the data

to produce an output, rather than fitting specifically to random noise in the training data. Mathematically, the loss function when combined with mean absolute error is then given by:

$$\text{loss} = \sum (y_{pred} - y_{true})^2 + \lambda_1 \sum \mid \beta \mid + \lambda_2 \sum \beta^2$$

where β are the model weights.

2. **Gradient Surgery** To see whether the conflicting gradients was an issue, we also adapted a PyTorch implementation of gradient surgery (as described in section 2) to our model(Tseng, 2020).
3. **Cosine Embedding Loss** Noting that the paired dataset generally involved seeing how close two sentences are with each other, we experimented with implementing the cosine embedding loss with our functions. In particular, the cosine embedding loss is given by:

$$\text{loss} = \begin{cases} 1 - \cos(\boldsymbol{x_1}, \boldsymbol{x_2}), & y = 1 \\ \max(0, \cos(\boldsymbol{x_1}, \boldsymbol{x_2}) - \eta), & y = -1 \end{cases}$$

where the margin η is a hyperparameter to be selected, and y is an indicator variable for whether two sentences are similar.
4. **Concatenation Method** For the paired datasets, we experimented with two ways to combine both inputs to form a single output: (1) concatenating both sentences' tokens, feeding them through BERT and then to the final layer, and (2) feeding each sentence through BERT, concatenating the embeddings, and finally projecting the concatenated embeddings to the final layer.
5. **Separator Token** We noted that if we were to concatenate the tokens (i.e method (1) above) before putting them through BERT, it would be impossible for the model to tell when the start of a new sentence is. Therefore, an original idea that we experimented with was adding a separating token manually between sentences, before the concatenated tokens were passed through BERT.
6. **Dataset Training** We experimented with different ways of iterating through the three different datasets for training: (1) Sequential Training and (2) Concurrent Training. In sequential training, we train each task separately, where one epoch involves iterating through each dataset one time (i.e. round-robin training an epoch of each dataset). In concurrent training (as described in the baseline above), we iterate through the largest dataset, and restart the smaller datasets until the largest dataset has been completed (i.e. round-robin training one batch on each dataset until Quora is done).
7. **Regression Loss** We experimented with three base loss functions for the semantic textual similarity task: (1) Mean Absolute Error, (2) Mean Squared Error, and (3) Negative Correlation.

4 Experiments

4.1 Data

We use the datasets given in the default project handout, with no additional preprocessing other than the ones already specified in the starter code. These include:

- (1) Stanford Sentiment Treebank (SST)[1] for Sentiment Classification, which consists of 11,855 single sentences, and 215,514 unique phrases which have been annotated by 3 human judges. These have 5 labels of negative, somewhat negative, neutral, somewhat positive, or positive,
- (2) CFIMDB Dataset for Sentiment Classification, which consists of 2,434 highly polar movie reviews, having a binary label of positive or negative. (Sentiment Classification),
- (3) The Quora dataset [2] for Paraphrase Detection, which consists of 400,000 question pairs with labels indicating whether particular instances are paraphrases of one another,

[1]https://nlp.stanford.edu/sentiment/treebank.html
[2]https://quoradata.quora.com/First-Quora-Dataset-Release-Question-Pairs

- (4) The SemEval STS Benchmark dataset Agirre et al. (2013) for Semantic Textual Similarity, which consists of 8,628 different sentence pairs of varying similarity on a scale from 0 (unrelated) to 5 (equivalent meaning).

For all these datasets, we have used the splits as defined in the default project handout.

4.2 Evaluation method

To quantitatively evaluate the models, we focused on using accuracy scores for Sentiment Classification and Paraphrase Detection, and Pearson Correlation for Semantic Textual Similarity, we then checked F1 scores for the final model to evaluate how well we have done. For qualitative analysis we observe specific cases of where our model goes wrong to understand failure cases.

4.3 Experimental details

4.3.1 General Details

For the single-task sentiment classification model, we followed default parameters in the handout. In the case of the multi-task model, we experimented with including different architectural changes in a step-wise manner, by only implementing one architectural change at a time across all tasks (where appropriate[3]). We then chose to keep it in the model for the specific task if we noticed an increased performance on that task. While this does not cover all possible combinations, it allowed us to observe the impact of a particular method on our results, and systematically experiment with combinations of the different architectural changes.

4.3.2 Hyperparameter Tuning

For all of the experiments when not otherwise specified, we used a self-implemented AdamW optimizer with learning rate of $1e-5$, a hidden dropout probability of 0.3 with the same seed, and trained for up to 10 epochs, selecting the best model that achieved the highest combined score over all 3 tasks on the dev set.

For the Elastic Net, we tested a strong version ($\lambda_1 = \lambda_2 = 10$) and a weak version ($\lambda_1 = \lambda_2 = 1$). Ultimately, the weak version worked better and we include them in the results below (see Table 6 for results of the strong version).

For the Cosine Embedding Loss, we kept $\eta = 0.5$ and had to threshold the actual label to deem whether something was similar or not (i.e. to set $y = 1$ or $y = -1$). For the paraphrase detection, this was simple as the task was binary. For the textual similarity task, the best threshold was $y = 1$ if the similarity was greater than 2.8, and $y = -1$ otherwise (see Table 7 for results).

For the Hidden Dropout Probability, we experimented with and without the dropout but ultimately kept $p = 0.3$ (see Table 6 for results).

4.4 Results

In general, we noted that training each task concurrently had better performance than training each task sequentially, and concatenation of tokens before BERT drastically increased our performance on the paraphrase detection and textual similarity task (see Table 6 in Appendix for full summary).

For the single-task classification, we achieved a Dev accuracy of 0.534 on SST, and 0.971 on CFIMDB under fine-tuning, and 0.397 on SST and 0.534 on CFIMDB under pre-training. For the multi-task classification, we present here the results the subset of experiments that involve concurrent training, with pre-BERT concatenation of sentence pairs for brevity, while the full table of results is left to the appendix. Our final model includes Elastic Net into the Sentiment Classification, Cosine Embedding Loss into Paraphrase Detection, and Separator Tokens for the Textual Similarity Task; we present the results for the dev and test sets below in Tables 1 and 2 below[4].

[3]As an example, for Cosine Embedding Loss which requires two tensors, we implement into Paraphrase Detection and Textual Similarity at one go, but not to Sentiment Classification

[4]Training scores not provided here as they were all very close to 100% accurate on the training set

Model Type	Sentiment Classification	Paraphrase Detection	Textual Similarity
Base Model	0.506	0.886	0.887
Cosine Embedding	0.501	**0.887**	0.886
Elastic Net	**0.509**	0.885	0.883
Gradient Surgery	0.505	0.882	0.886
Separator Token	0.508	0.884	**0.888**
Final Model	**0.510**	0.884	**0.889**

Table 1: Dev Set Scores

	Sentiment	Paraphrase	Similarity
Accuracy Score	0.520	0.886	0.888
F Score (Dev)	0.508	0.847	

Table 2: Best Model Test Scores

5 Analysis

5.1 Quantitative Analysis

Here we present some of the more detailed results of the ablation studies of the model that we conducted.

5.1.1 Concatenation Method

The largest increase of performance for our model involve the concatenation method being used. This is most notable in Textual Similarity task, where the score more than doubled. The results of pre-BERT and post-BERT concatenation is below in Table 3.

	Sentiment Classification	Paraphrase Detection	Textual Similarity
pre-BERT Concatenation	**0.519**	0.778	0.389
post-BERT Concatenation	0.506	**0.886**	**0.887**

Table 3: Concatenation Method Ablation Tests

While we are not entirely sure why this might be true, one hypothesis is that the bi-directionality of the BERT model is able to capture better embeddings in the concatenated sentence as it can contextualize the words in both sentences. Furthermore, this is somewhat supported by how the separator token further improved performance slightly as the model not just contextualize the words in both phrases, but the structure as well. Alternatively, another possibility is that the higher dimensional space allows the model to learn word embeddings that generalize better as it is harder to overfit a higher dimensional space. This is similar to how the larger models such as $\text{BERT}_{\text{large}}$ might have better performance on the tasks as they have the ability for better representation (Munikar et al., 2019).

5.1.2 Sequential Training

The next largest difference in our model's performance involve whether we training the model concurrently or sequentially. In general, sequential training led to worse overall total scores due to a worse score in the first two tasks, but occasionally a better score for the last task that was trained (see Table 4 below where the sets are ordered in the order they were trained).

Since we are backpropagating the model at each minibatch for all of the tasks, we hypothesize that the sequential training causes the model to focus only on getting the best weights for the final task, and subsequently learning worse weights for the earlier tasks. This can be seen on how while the model overall does worse in sequential training, it sometimes does better for the final task, such as Sentiment Classification when we trained in the order of {STS, Paraphrase, SST}.

	Sentiment Classification	Paraphrase Detection	Textual Similarity
post-BERT Concatenation	0.506	**0.886**	**0.887**
{Paraphrase, STS, SST}	0.510	0.856	0.872
{Paraphrase, SST, STS}	0.509	0.862	0.874
{SST, Paraphrase, STS}	0.467	0.857	0.882
{SST, STS, Paraphrase}	0.479	0.863	0.861
{STS, Paraphrase, SST}	**0.517**	0.845	0.867
{STS, SST, Paraphrase}	0.504	0.873	0.848

Table 4: Training Method Ablation Tests

5.1.3 Regression Loss

To finally, select the Mean Absolute Value (L_1) loss function for our regression task, we experimented specifically on the Textual Similarity Dataset. In this case, we noticed that we got the best result using the L_1 loss and thus stuck with that for the multi-task problem.

	L_1 Loss	L_2 Loss	Negative Correlation Loss
Textual Similarity Correlation	**0.872**	0.865	0.870

Table 5: Regression Loss Ablation Test

An interesting result that we found here is that when we trained the model in multi-task, we actually beat the single-task regression score. Possibly the model is able to transfer knowledge from the other tasks to the regression task as well and thus a multi-task learning process might actually be beneficial for performance.

5.1.4 Gradient Surgery

As seen in Table 1 above, one of the surprising results was the Gradient Surgery ultimately made our model worse in all aspects. One possible reason that we suspect is that the intent of Gradient Surgery is to allow the model to traverse the loss surface better when gradients are conflicting. However, our base model is already able to traverse the loss surface decently well on its own and can get extremely low losses and high accuracies on the training set, and thus gradient surgery ultimately made the model learn worse embeddings.

5.2 Qualitative Analysis

Having analyzed some of the quantitative results, we wanted to look at specific cases to see where exactly our model made errors so that we can try to target further approaches. As such, we selected specific examples for each task to see if any trends emerge amongst the model's mistakes.

5.2.1 Sentiment Analysis

On the dev set of Sentiment Analysis, the model correctly identifies the sentiment of 50% of the cases, is off by 1 in 40% of cases, by 2 in 8%, and by 3 in 2%; off by 4 errors did not occur. While the accuracy might not seem that good, there exists quite a bit of variance in how different people quantify the sentiment of a particular thing. Therefore, if we assume off by one errors are just an artifact of how humans rate movies, our model actually does pretty well.

The most egregious errors (i.e. the ones where the sentiment is off by 3) often come from usage of either usually extremely positive or negative terms in the opposite context such as through sarcasm. For instance:

- "*this flick is about as cool and crowd-pleasing as a documentary can get*"
- "*a synthesis of cliches and absurdities that seems positively decadent in its cinematic flash and emptiness*"

In the first case, the label was very positive (4) but our model predicted a negative review (1) possibly due to the negative connotation of "crowd-pleasing". In the second case, the review uses "positively decadent" which is a positive phrase but used in this context negatively and thus our model preducted a positive review (4) when it was actually a negative review.

However, in some cases it might seem that the sample is just mislabeled:

- A review that stated: “*hilariously inept and ridiculous*’ had a sentiment of 3, which seems too high for a movie deemed “*inept*".

5.2.2 Semantic Textual Similarity

For the textual similarity task, 18% of the time we are within 0.1 of the actual rating, 62% of the time we are within 0.5 of the actual rating, while 86% of the time we are within 1 of the rating. Again, noting that providing an actual number to quantify similarity is an extremely subjective thing, we do decently well being within 1 of the rating. Furthermore, given our correlation score of almost 90% in Table 1, we can generally predict the trend, which is arguably more important.

For the outliers, the errors seem to come from either a change of word giving a sentence a radically different meaning, or either having long paraphrases with different words having the same meaning. For instance, the worst two samples are:

- “*work into it slowly*" and “*it seems to work*"
- “*in these days of googling, it’s sloppy to not find the source of a quotation*" and “i *agree with kate sherwood, you should be able to attribute most quotes these days by simple fact checking*"

In the first example, the actual similarity was 0.0, while the predicted similarity was 3.1. As a human, we understand that work in the first sentence is used as a verb while it is being used as noun in the second sentence. In this case, our model is able to pick up the usage of “*work*" in both sentences, but is unable to pick up the context; this is possibly remediated by using POS-tagging which we did not implement.

In the latter example, our model predicted 0.6 while the actual similarity was 3.2. This is likely due to how a human might take fact checking to be synonymous to “*googling*", which our model might not pick up. Furthermore, names such as “kate sherwood" appearing in the second sentence might have thrown off our model.

5.2.3 Paraphrase Detection

For paraphrase detection, our model gets only 11% wrong. Here, we found several patterns among the errors:

- Two non-paraphrase sentences that are one word apart: “*why did nokia fail?*" and “*how did nokia fail?*"
- Paraphrases that differ a lot in word choice seem to be recurrently misclassified: “*what kinds of advantages do aesa radar provide over pesa radar?*" and “*can aesa radar track multiple threats at different direction simultaneously?*"
- Paraphrases that are similar but with added words seem to give the model a hard time: “*why do people believe in heaven?*" and “*why do people believe in heaven and hell?*"

The first example can clearly be seen as not a paraphrase even though our model predicts it as so. This is likely due to the fact that most of the sentence is the same except one word that changed the entire meaning and thus our model fails as perhaps it learnt that similar words is more important than the meaning change of the single word.

In the latter two examples, the model seems to have been thrown off due to the differences in word count or word usage. However, noting that it is arguable whether these cases should actually be paraphrases as they are somewhat similar but not necessarily paraphrases, our model seems to be doing relatively well.

Besides our model failing, there also are some samples that are mislabeled and thus our model’s accuracy should actually be higher. Here are some of them:

- “*what are some really cool working science models i can prepare for my school science exhibition?*" and “*i am in 10th grade. i need to make a working model for my science exhibition. what can i make ?*" are denoted as not paraphrases despite seeming like paraphrases.
- “*how is a magnetic field generated when a current flows through a conductor?*" and “*is a magnetic field present when a current is not flowing through a conductor?*" are labeled as paraphrases even though the asking how and is are two separate questions. Furthermore, the second is the negation of the first, and thus they also have opposite meaning.

6 Conclusion

6.1 Main Findings

The main finding of our project is that it is possible to reach relatively good results (7th on the test leaderboard as of 21 March) for the multitask process even when resources to find additional datasets or use larger models is unavailable. In particular, we note that despite the model having to learn 3 different tasks, we are able to achieve results that are close to the single-task versions, and in some cases even able to beat the single-task model. Therefore, as we search for more efficient ways to deploy deep learning systems, multi-task models should be something people consider more often.

6.2 Loss Functions

We note that certain loss functions have different effects on the different tasks, with some performing loss functions making some tasks do better, while other tasks do worse. Therefore, it is important to also choose loss functions appropriately for the task and not just default to one choice of loss function without appropriate experimentation.

6.3 Future Work

We have four avenues for possible future work on our project:

1. **Part Of Speech (POS) Tagging** It seems like POS-tagging might solve some of the cases in our qualitative analysis. As this does not require additional resources, it suits our study of optimizing only the model architecture and thus would be a good experiment to do in the future.
2. **Sentiment Classification Research** A large bulk of our efforts has been placed into optimizing our paraphrase accuracy and textual similarity correlations, while not much experimentation other than Elastic Net was done on the Sentiment Classification. This reflects in the results where despite being 7th, we are last out of the top 20 in Sentiment Classification Scores. Therefore, we intend to do more research and experimentation into building a better sentiment classification model.
3. **Complete Ablation Study** Due to the time constraints, we could not train every single combination of model of the various architectures. Having noted that certain implementations affect certain tasks differently, we would like to do a fully exhaustive ablation study on every single architecture change, and not only proceed in a greedy manner.
4. **Hyperparameter Tuning** For the most part, we did not really do an extensive hyperparameter tuning. Given this fact, it is likely that future work could get better results upon hyperparameter tuning.

References

Eneko Agirre, Daniel Cer, Mona Diab, Aitor Gonzalez-Agirre, and Weiwei Guo. 2013. *SEM 2013 shared task: Semantic textual similarity. In *Second Joint Conference on Lexical and Computational Semantics (*SEM), Volume 1: Proceedings of the Main Conference and the Shared Task: Semantic Textual Similarity*, pages 32–43, Atlanta, USA. Association for Computational Linguistics.

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

Manish Munikar, Sushil Shakya, and Aakash Shrestha. 2019. Fine-grained sentiment classification using bert. *arXiv preprint arXiv:1910.03474v1*.

Wei-Cheng Tseng. 2020. Weichengtseng/pytorch-pcgrad.

Tianhe Yu, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. 2020. Gradient surgery for multi-task learning. *arXiv preprint arXiv:2001.06782*.

Hui Zou and Trevor Hastie. 2005. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society, Series B*, 67(2):301–320.

A Appendix

Here we provide the results of every experiment we ran as a comparison. In this table, pre-concatenation refers to concatenation prior to BERT (i.e. concatenate the tokens to get embeddings), while post-concatenation refers to concatenation after BERT (i.e. concatenate the embeddings). Here we also add some of the hyperparameter tuning that we have tried, including tuning removing dropout, testing strength of the elastic net, calculating loss after each task in the concurrent train, and adding extra layers to the sentiment classification.

		Model Configuration	**Sentiment**	**Paraphrase**	**Similarity**
Pre-Concatenation	**Sequential Training**	{Paraphrase, STS, SST}	0.510	0.856	0.872
		{Paraphrase, SST, STS}	0.509	0.862	0.874
		{SST, Paraphrase, STS}	0.467	0.857	0.882
		{SST, STS, Paraphrase}	0.479	0.863	0.861
		{STS, Paraphrase, SST}	0.517	0.845	0.867
		{STS, SST, Paraphrase}	0.504	0.873	0.848
	Concurrent Training	Base Model	0.506	0.886	0.887
		Base + Cosine Embedding	0.501	0.887	0.886
		Base + Strong Elastic Net	0.509	0.885	0.883
		Base + Weak Elastic Net	0.504	0.884	0.881
		Base + Gradient Surgery	0.505	0.882	0.886
		Base + Sep. Token	0.508	0.884	0.888
		Final Model	0.510	0.884	0.889
		Final Model w/o Dropout	0.508	0.885	0.890
		Final Model Sequential Loss	0.501	0.886	0.886
		Final Model Sentiment Layers	0.506	0.883	0.887
Post-Concatenation	**Sequential Training**	{Paraphrase, STS, SST}	0.519	0.778	0.389
		{Paraphrase, SST, STS}	0.510	0.773	0.393
		{SST, Paraphrase, STS}	0.463	0.789	0.395
		{SST, STS, Paraphrase}	0.483	0.771	0.379
		{STS, Paraphrase, SST}	0.511	0.772	0.365
		{STS, SST, Paraphrase}	0.498	0.791	0.356
	Concurrent Training	Base Model	0.511	0.779	0.401
		Base + Elastic Net	0.509	0.719	0.383
		Base + Cosine Embedding	0.378	0.790	0.690
		Base + Gradient Surgery	0.520	0.722	0.383

Table 6: Experiments Summary

We also append the results here of the thresholding of the cosine embedding loss on the textual similarity task, even though we did not include it in the final model.

Threshold	2.5	2.6	2.8	3.0	3.2	3.4	3.5
Correlation	0.868	0.867	0.871	0.857	0.855	0.832	0.841

Table 7: Cosine Embedding Tuning Results