BERTina Aguilera: Extensions in a Bottle

Stanford CS224N Default Project

Kohi Kalandarova
Department of Computer Science
Stanford University
kohik@stanford.edu

Mhar Tenorio
Department of Computer Science
Stanford University
mhar@stanford.edu

Sam Prieto Serrano
Department of Computer Science
Stanford University
samp13@stanford.edu

Abstract

We aim to build and fine-tune a BERT model to show its multitask capabilities on three downstream tasks: sentiment analysis, semantic textual similarity, and paraphrase detection. Specifically, we expand on the BERT model and companion AdamW optimizer with the following extensions: exponentially decaying learning rate, cosine similarity fine-tuning, regularized optimization, and multiple negative rank loss. We investigate the effects on performance of different, strategically-selected combinations of extensions. We find that learning rate decay, cosine fine tuning using MSE loss, and training on additional datasets yield the best results on the three tasks, with improvements of 0.9% in sentiment analysis accuracy, 16.6% in paraphrase detection, and 46.4% in semantic textual similarity over our baseline scores, respectively. These results demonstrate the effectiveness and power of multitask fine-tuning in expanding BERT's efficiency.

1 Key Information to include

- Mentor: Anirudh Sriram
- External Collaborators (if you have any): None
- Sharing Project: No
- Team Contributions:
  Kohi: Work on cosine similarity fine-tuning, multiple negative rank loss learning, developing visualizations, organizing/combining extensions for experimentation, and noting ethical considerations.
  Mhar: Work on incorporating LR scheduler, additional datasets, regularized optimization, and cosine similarity fine-tuning to training the model. Helped run experiments and manual hyperparameter search.
  Sam: Work on implementing cosine similarity fine-tuning, multiple negative rank loss learning, regularized optimization, documenting and planning for experimentation, and running experiments.

2 Introduction

Natural Language Processing (NLP) tasks such as sentiment analysis, paraphrase detection, and similarity detection play pivotal roles in various real-world applications, ranging from social media analytics to information retrieval systems. One of the breakthroughs in NLP has been the advent of

transformer-based models, particularly BERT (Bidirectional Encoder Representations from Transformers) Devlin et al. (2019). BERT has achieved remarkable success by pretraining on large corpora and then fine-tuning on specific downstream tasks. However, while BERT provides strong baseline performance across various NLP tasks, there is still room for improvement, especially in scenarios where fine-tuning adaptations are required to address specific task nuances.

The problem with testing BERT models without task-specific fine-tuning on multitask learning is that BERT embeddings may not capture task-specific nuances or semantics required for effective multitask learning. Additionally, the pre-trained representations may not align perfectly with the objectives of all tasks, resulting in limited task-specific optimization and potentially conflicting gradients during training.

In this paper, we explore novel fine-tuning extensions to the BERT model and evaluate them on the tasks of sentiment analysis of single sentences, paraphrase detection of sentence pairs, and semantic textual similarity analysis of sentence pairs. By introducing task-specific adaptations during the fine-tuning phase, we aim to enhance the model's capability to handle linguistic patterns and semantic subtleties that are more suitable and relevant to each task.

We hypothesize that by doing our extensions or combinations of extensions, we can improve the overall performance of BERT across our selected baselines. Our aim is for our study to build upon previous research in the field and, in a way, bring insights from experts together into one testing ground.

3 Related Work

Several recent studies have contributed to advancing the effectiveness of BERT-based models through innovative techniques such as cosine similarity fine-tuning, utilizing multiple negative rank loss (MNRL), and regularized optimization. In this section, we review key works that have influenced our approach and guided us as we implemented various extensions with the hopes of improving our minBERT model.

For understanding the base model, we referred to Devlin et al. (2019). This paper introduces BERT which is designed to pretrain deep bidirectional representations from unlabeled text by conditioning on content in all layers. This pre-trained model can be finetuned through various means to create state-of-art models and results for many tasks, including the three we are addressing in this paper.

In Reimers and Gurevych (2019), cosine-similarity fine-tuning using siamese network embeddings assesses sentence likelihood. This paper introduces Sentence BERT (SBERT), a modified BERT model utilizing siamese and triplet network structures to obtain meaningful sentence embeddings compared using cosine similarity. SBERT drastically reduces computation time compared to BERT. Improvements this paper saw include a vast reduction in time needed to compute sentence combinations for example, in the clustering of 10,000 sentences using hierarchical clustering, BERT needed 65 hours while SBERT was able to do it in 5 seconds. For our project, we make use of cosine similarity fine tuning in regards to the paraphrase and sentiment analysis tasks.

Henderson et al. (2017) propose a learning strategy to maximize similarity between a model's representations of training examples and similar instances, while minimizing their distance from dissimilar ones. This approach resulted in a consistent 20% reduction in error rates for P@1 (precision at 1, this paper's evaluation metric) across conversational sets. By maximizing similarity for positive pairs and minimizing it for negatives, MNRL was adapted into our project, especially for paraphrase-related tasks.

In Jiang et al. (2020), the team introduces the SMART learning framework with two parts: SMoothness-inducing Adversarial Regularization, and BRegman pRoximal poinT opTimization. In the first part they impost regularization to control complexity of the model during fine tuning, and in the second part they develop a set of Bergman proximal point optimization methods to prevent aggressive model updating at each iteration. Their empirical results suggests improvements on many various NLP tasks and models including on BERT, RoBERTa, and GLUE. We applied this method to varying extents in our experiments, such as utilizing SMART methods vs our own implementations across various data sets within the loops of our train_multitask function.

All of these papers assisted in our experimentation and provided guidance on how to manage our implementations and strategies while tackling this project.

4 Approach

4.1 Baseline Model

Our first approach consisted of implementing the basic architecture of a BERT model, which involves a multi-layer bidirectional Transformer encoder. This encoder processes input sequences in parallel, capturing contextual information from both left and right contexts. We created the architecture for multi-headed attention, an add-norm function, an embedding function, and a forward function. In addition we coded the AdamW optimizer, using the step function according to the project handout's definition of the Adam algorithm. Finally, we implemented forward functions of the sentiment classifier and the multitask classifier, as well as implemented the prediction functions that output predictions for each of the three tasks of interest.

We then take this BERT model and evaluate its performance on three downstream tasks. To generate a prediction for sentiment analysis, we feed the pooled representation of a sentence into a dropout layer and then a linear layer to generate a logit that represents the five sentiment classes. For the STS task, we take two embeddings and calculate their cosine similarity. We then feed this into a ReLU activation function to clip its output between $[0, 1]$. Since the output should be the five degrees of similarity, we scale the output of the ReLU function to match the labels of the embeddings. Lastly, for the paraphrase task, we again take two pooled representations, use cosine similarity, and scale the ReLU output by 5. Since this is a binary task, we then set a threshold (3.75), where similarities above this threshold will output 1 and 0 otherwise.

4.2 Learning Rate Decay

As an initial extension, we incorporated a learning rate scheduler to exponentially decay our learning rate after each epoch during training. We update our learning rate with the following equation

$$lr = lr_0 \exp(-\gamma \cdot \text{epoch})$$

with a hyperparameter γ that dictates how much to decay our learning rate.[1] We did this to help our training converge in a more stable and more efficient manner. Initially, a large learning rate could be useful in making bigger update steps. However, as we go further into our training, larger updates might lead to more oscillations and overestimation of the solution as we get closer to the optima. Therefore, decaying our learning rate per epoch would help create a training update that is more stable and is more likely to carefully explore the parameter space for the optimal solution. We hope that this would help us make our model more generalized.

4.3 Additional Datasets for Training

In the baseline model, we train on the SST dataset and evaluate our model on the three downstream tasks. Given that the SST dataset labels sentiment of phrases, we were cautious of how much a model only trained on this dataset would be able to generalize when performing three tasks simultaneously. We therefore conducted a sequential training method. In this method, we went through all the batches of each of the three datasets sequentially at each epoch. We chose this method to ensure that each dataset would be trained on the same number of times. To address the size differences of the three datasets and to efficiently allocate computational resources, we randomly sampled $\sim 8,500$ data points from the Quora dataset so it approximately matches the SST dataset.

We used cross entropy as the loss function for the sentiment analysis task because it involved classifying texts into a sentiment category. As a baseline, we used binary cross entropy with logits as the paraphrase detection loss function given it involved a classification problem with binary outcomes. We used MSE loss for the STS task because this was a regression task of finding similarity between two texts, rather than a classification problem. In our extensions, we experiment with other loss functions, as detailed below.

[1] https://pytorch.org/docs/stable/generated/torch.optim.lr_scheduler.ExponentialLR.html

4.4 Cosine Similarity Fine-Tuning

In our implementation of prediction functions, we utilize cosine similarity as part of both the predict_paraphrase and predict_similarity. Our methodology for determining similarity was inspired by Reimers and Gurevych (2019). We implemented a predict_similarity function, which takes two sets of input_ids and attention_masks, feeds each set to our forward function, and computes the cosine similarity between outputs. In the case of predict_similarity, we utilize ReLU to constrain the output between 0 and 1, and then scale it by 5 to align with our provided data, which rates similarity on a scale of 1-5. In the case of predict_paraphrase, we further filter the result for at least 3.5 out of 5 in similarity score to determine the sentence pair is a paraphrase or not.

A way of potentially improving our embeddings is therefore to fine-tuning these tasks with an appropriate loss function. To this end, we employ combinations of two different loss functions, CosineEmbeddingLoss[2] and Mean Squared Error (MSE) loss. For MSE loss, we fed in the cosine similarity of the two embeddings as the input and the target tensor. To use CosineEmbeddingLoss for STS analysis, we generated a binary tensor that has value 1 when the label or the similarity is > 4 and a value of -1 otherwise to use as our target tensor. To use CosineEmbeddingLoss on the paraphrase task, our target tensor was a binary tensor that has value 1 when the two inputs are paraphrases of each other and -1 otherwise.

4.5 Fine-Tuning with Regularized Optimization

Some of our other approaches are focused on fine-tuning with regard to task-specific learning. However, aggressive fine-tuning can then lead to over-fitting, which in turn can cause the model to fail to generalize to unseen data.

Conscious of our fine-tuning strategies, we determined a need to combat this. Inspired by Jiang et al. (2020), we decided to pursue regularized optimization. Jiang et al. (2020) specify two steps for this (1) Smoothness-inducing regularization (2) Bregman proximal point optimization. With their SMART framework, we are able to manage the complexity of the model more effectively, as well as prevent aggressive updating.

In our implementation, we use the 'smart-pytorch'[3] library's loss methods within our learning loops. We ran into complications implementing the extension on the paraphrase task, so we created one ourselves for this loop doing parameter weight decay.

4.6 Multiple Negatives Ranking Loss Learning

In efforts to find a more nuanced objective function tailored to the nature of paraphrase detection, we decided to incorporate a multiple negative rank loss for the task. By considering multiple negative samples alongside positive pairs during training, we aim for the model to learn to distinguish between genuine paraphrases and unrelated sentences more effectively. We were inspired by Henderson et al. (2017), whose approach encourages the model to focus on capturing fine-grained semantic similarities and differences, leading to improved performance in identifying paraphrases.

Furthermore, incorporating multiple negative samples helps mitigate the impact of noisy or uninformative negative examples, enhancing the robustness and generalization capabilities of the model in real-world scenarios. We began by trying to use the suggested library, SentenceTransformer[4], but found complications when it came to merging it with the BERT model. Thus, we ultimately utilized PyTorch's MarginRankingLoss[5] in place. We developed a helper function to assist in the creation of negative pairs, computing our loss with the help of positive pairs and the MarginRankingLoss.

[2]https://pytorch.org/docs/stable/generated/torch.nn.CosineEmbeddingLoss.html#torch.nn.CosineEmbeddingLoss

[3]https://github.com/archinetai/smart-pytorch

[4]https://github.com/UKPLab/sentence-transformers/tree/master

[5]https://pytorch.org/docs/stable/generated/torch.nn.MarginRankingLoss.html

4.7 Manual Hyperparameter Search

After finding our best-performing model using a combination of the extensions above, we experiment with different values for the learning rate, the learing decay rate, and the hidden dropout probability in hopes of further improving the model.

5 Experiments

5.1 Data

We use the Stanford Sentiment Treebank (SST) dataset and the CFIMDB dataset to train, tune, and evaluate our minBERT model on sentiment classification. The SST dataset includes 215,154 unique phrases from 11,855 single sentences from movie reviews, with each datapoint containing a sentiment label. The CFIMDB dataset contains 2,434 movie reviews classified under a binary scale of positive or negative review.

Next, we use the SST dataset, the Quora dataset, and the SemEval STS Benchmark dataset to evaluate our minBERT model on performing three tasks simultaneously. As a baseline, we use the SST dataset to train and evaluate the model's performance on sentiment classification. We use the Quora dataset, which include binary labels if one sentence is a paraphrase of the other, to evaluate the model's performance in paraphrase detection. Finally, we use the SemEval STS Benchmark dataset, which labels the degree of similarity between different sentence pairs, to evaluate the model's performance in semantic textual similarity. In our extensions, we fine-tune our model on the three datasets, as described in Section 4.3.

5.2 Evaluation method

We evaluate the performance of the sentiment analysis and paraphrase detection tasks with accuracy, and the semantic textual similarity task with Pearson correlation. To evaluate the overall performance of a model, we averaged the three scores obtained from the three tasks.

5.3 Experimental details

Our experiments are run on an NVIDIA T4 GPU in Google Cloud Platform. We finetune each model for 10 epochs. We used initial learning rate α of 0.00001. Our initial dropout probability was $\delta = 0.3$ and our batch size was 8 samples. For LR-Decay, we set our decay rate to $\gamma = 0.9$. We explore other values of for α, δ when conducting a manual hyperparameter search. Training with all three datasets took around 10 minutes per epoch.

5.4 Results

For our test submission, we used our model that incorporated additional training, learning-rate decay, and cosine similarity fine-tuning using MSE loss. Here, we see that our model performed well on the STS task, garnering the highest score, while we see the lowest accuracy for the sentiment task.

Test Model	Sentiment	Paraphrase	STS	Overall
BL + LR + AD + CS2	0.518	0.675	0.727	0.686

Legend: **BL** = Baseline (minBert), **LR** = Learning Rate Decay, **AD** = Additional Datasets, **CS2** = Cosine Similarity Fine-tuning with Mean Squared Error loss

Table 1. The performance of our model on the three downstream tasks, as evaluated on the **test** dataset.

As mentioned in our introduction and project objective, we made combinations strategically based on the performance we were seeing and the extensions we knew targeted certain tasks. Some of the observations or reasoning we took into consideration for our combinations were:

- use Regularized Optimizer when doing heavy fine-tuning extensions to combat the overfitting

- keep the Learning Rate Decay consistent to allow fair comparison
- allow all loss functions fair chance to display an impact

With this said, below are the results of our various combination experiments submitted to the development data set.

M#	Model	Sentiment	Paraphrase	STS	Overall
1	BL	0.516	0.510	0.284	0.556
2	BL + LR	**0.527**	0.524	0.276	0.563
3	BL + LR + AD + CS2	0.525	**0.676**	**0.748**	**0.692**
4	BL + LR + RO1	0.518	0.551	0.277	0.569
5	BL + LR + AD + MNRL1 + CS2	0.466	0.569	0.329	0.567
6	BL + LR + RO1 + CS2	0.416	0.659	0.671	0.637
7	BL + LR + AD + CS1-1	0.450	0.588	0.370	0.574
8	BL + LR + AD + CS1-2	0.511	0.674	0.726	0.683
9	BL + LR + AD + RO1 + MNRL1 + CS2	0.376	0.617	0.578	0.594
10	BL + LR + AD + RO1 + CS1-2	0.453	0.646	0.642	0.640
11	BL + LR + AD + RO1 + CS1-1	0.353	0.582	0.342	0.535
12	BL + LR + AD + RO2 + MNRL2 + CS2	0.388	0.391	0.339	0.483

Legend: **BL** = Baseline (minBert), **LR** = Learning Rate Decay, **RO1** = Fine-Tuning with Regularized Optimization (SMART) on all tasks, **RO2** = Fine-Tuning with Regularized Optimization (SMART) only on paraphrase tasks, **AD** = Additional Datasets, **CS1-1** = Cosine Similarity Fine-tuning with CosineEmbeddedLoss on the similarity task, **CS1-2** = Cosine Similarity Fine-tuning with CosineEmbeddedLoss on the paraphrase task, **CS2** = Cosine Similarity Fine-tuning with Mean Squared Error loss, **MNRL1** = Multiple Negative Ranking Loss using MarginRankLoss only, **MNRL2** = Multiple Negative Ranking Loss using MarginRankLoss plus BinaryCrossEntropywithLogits as a fallback for empty pos/neg tensor of samples

Table 2. The performance of our model using different extensions to the baseline model, as evaluated on the **dev** dataset.

5.4.1 Observations

Through our experiments we found that a combination of training on additional datasets, an exponential learning rate scheduler, and fine-tuning on the STS cosine similarity task using MSE loss led to the best performance of our model (**M3**). Specifically, we saw significant improvements on the paraphrase and STS tasks when we finetune on the corresponding datasets. We also found that an exponential learning rate scheduler improved on the baseline (**M2**). Thus, we kept these two extensions throughout all experiments.

Next, we observe different performance with different loss functions. We find that using CosineEmbeddedLoss on the STS dataset led to significantly poorer performance across the three tasks, and specifically for the STS task (**M7**). However, we see that using CosineEmbeddedLoss on the Quora dataset led to scores that are closer to our best-performing model (**M8**).

We also observed that using the MNRL loss on the paraphrase task led to significantly worse performance, specifically on the STS task (**M5, M9, M12**). Incorporating SMART loss to prevent aggressive updates led to worse performance (**M4, M6, M9-12**).

We were surprised to find that combining our extensions led to worse performance (**M12, M10**). Specifically, we find that when we combine different extensions, we find that our performance on the sentiment analysis is significantly worsened as opposed to simpler combinations such as when we do not train on additional datasets. In **M12**, we also find that the scores for paraphrase and STS tasks decreased remarkably, leading to our worst model.

5.4.2 Hyperparameter Search

We took our best performing model (**M3**) and ran it using different hyperparameters. The results of this search are below.

Hyperparameters	Sentiment	Paraphrase	STS	Overall
$\alpha = 0.00001, \delta = 0.1$	0.482	**0.681**	0.711	0.673
$\alpha = 0.00001, \delta = 0.3$	**0.525**	0.676	**0.748**	**0.692**
$\alpha = 0.00001, \delta = 0.5$	0.491	0.658	0.695	0.666
$\alpha = 0.001, \delta = 0.3$	0.253	0.375	0.024	0.380
$\alpha = 0.0001, \delta = 0.3$	0.262	0.375	-0.008	0.378

Table 3. The performance of our model using different values for the learning rate α and the dropout probability δ on the **dev** dataset.

We found that a high dropout probability led to worse performance and that the default dropout probability $\delta = 0.3$ led to the best performance. We then kept that dropout rate and adjusted the learning rate. We observed that a high learning rate greatly decreased our scores.

6 Analysis

Our best performing model involved training on additional datasets and using an exponential learning rate scheduler. We used binary cross entropy on the paraphrase task and fine-tuned on the STS task by using MSE loss. Training on these task-specific datasets helped our model learn on these examples, instead of just training on the SST dataset and assuming it will generalize to all three downstream tasks.

In the following chart, showing the progession of dev scores for the three tasks during learning epochs, we can see the steady positive slope (minus one outlier) that all three tasks saw. This is different compared to our other models, which experience more oscillations or even zero slopes. These other plots are viewable in the Appendix. In this graph, we see that our scores generally increase as training occurs, which is a good indicator that our model is learning during training and that the parameters do not get stuck in some local optima.

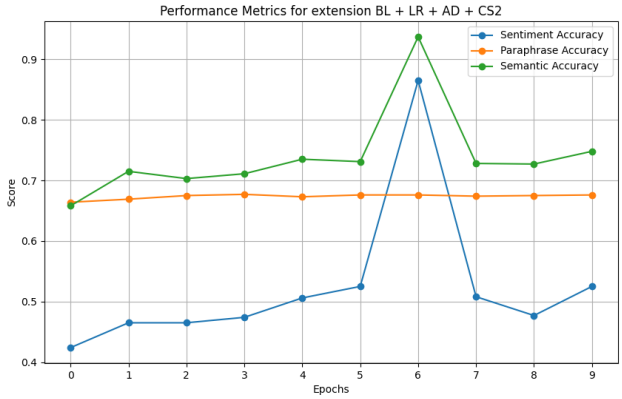


Figure 1. The dev scores for the three tasks across 10 epochs using our best model.

6.1 Hyperparameters

We found that the default values for the learning rate ($\alpha = 0.00001$) and the dropout rate ($\delta = 0.3$) gave the best performance. We noticed that when the learning rate is higher, the scores at each epoch tend to fluctuate more. It could be because a learning rate higher than this $(0.001, 0.0001)$ made the learning more unstable and perhaps overshooting or diverging from the optimum values of the model parameters. On the other hand, altering the dropout rate did not seem to significantly affect performance. Nevertheless, a dropout rate that is too high or too low still led to slightly decreased performance. This could be because a high dropout led to less learning of the data because more of the neurons were deactivated. A lower dropout rate perhaps made the model generalize less to unseen data.

6.2 Extensions

Despite the solid reasoning to use CosineEmbeddedLoss on the similarity task, we actually found the STS accuracy to be hurt by the use of this loss function. Instead, STS seemed to do its best performance with the MSE loss function. We hypothesize that the CosineEmbeddedLoss may not be well-suited for the similarity task due to its focus on embedding space distances rather than class probabilities. To fit the STS task on CosineEmbeddedLoss, we also had to create a target tensor that turns the 5-label target into a binary tensor. This led to a loss of information in the degree of similarity of the two phrases. Therefore, this loss function might lead to poor discrimination between similar and dissimilar pairs. On the other hand, MSE loss considers the continuous value of the cosine similarity of two embeddings, leading to more expressiveness.

On the other hand, we decided to experiment with the CosineEmbeddedLoss function in the paraphrase tasks, given it also utilizes cosine similarity, and found improved performance. The CosineEmbeddedLoss might be more appropriate for this task because it involves a binary output, and we therefore did not lose any information by creating the target tensor. Since paraphrase detection involves identifying semantically similar pairs by classification means (labeled as either paraphrase or not using a threshold), this loss function could lead to better performance.

Next, we were puzzled by the impact of the regularized optimization. This function, designed to prevent aggressive updates during training, might have adversely affected our performance by being too rigid. If constraints were introduced that hinder the model's ability to learn complex patterns across tasks, this might explain the lower resulting performance. A great example of this could be how the sentiment tasks always worsened with the SMART implementation, compared to earlier experiments without it. It would great to recreate most of our experiments without the RO and compare.

We ran **M10** as a result of the scores we were seeing for models before it, specifically **M6** and **M9**. We noticed that the sentiment task had done well with regularized optimization, paraphrase with CosineEmbeddingLoss, and STS with MSE loss. Our results from this combination were good, and better than those models which inspired it, but were still not our best achieved. The improvement likely came from our correct observations on what seemed to work well, but the regularized optimization probably hindered it from reaching even higher scores.

7 Conclusion

In this project, we set out to improve the baseline BERT model on three downstream tasks. Specifically, we implemented sentiment analysis, semantic textual similarity, and paraphrase detection, and tested them on the SST, Quora, and SemEval data sets, respectively. We aimed to achieve this improvement by implementing a variety of extensions, running experiments with different combinations of them, and trying out varied numbers for hyperparameters. In the end, we implemented training on task-specific datasets, learning rate decay, cosine similarity fine tuning multiple loss functions, fine tuning with regularized optimization with the SMART framework, and multiple negative rank loss. After initial rounds of experimentation, we took the one that yielded the best results and ran additional experiments with varied hyperparameters.

Contrary to our initial hypothesis, we concluded that additional extensions did not always yield the greatest results. In fact, our greatest scores were a result of our initial work, in which we only ran a combination of learning rate decay, cosine fine tuning using MSE loss, and additional training datasets. Meanwhile, our experiment that fared the worst was one with over four extensions at a time. We also concluded that certain loss functions were more appropriate to some tasks than others, such as the use of CosineEmbeddedLoss for paraphrase detection which proved advantageous.

Further work could involve accentuating the combinations we found did the best by finding ways of adding weight to their impact or searching for better hyperparameters. We could also experiment with gradient surgery to prevent training on different tasks to conflict with one another. We can also explore pre-training the BERT model on domain-specific data more suited for our tasks, instead of general domain data. Finally, further work with order of operations (i.e. dropout and linear layers) on model architecture could possibly yield better results.

References

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

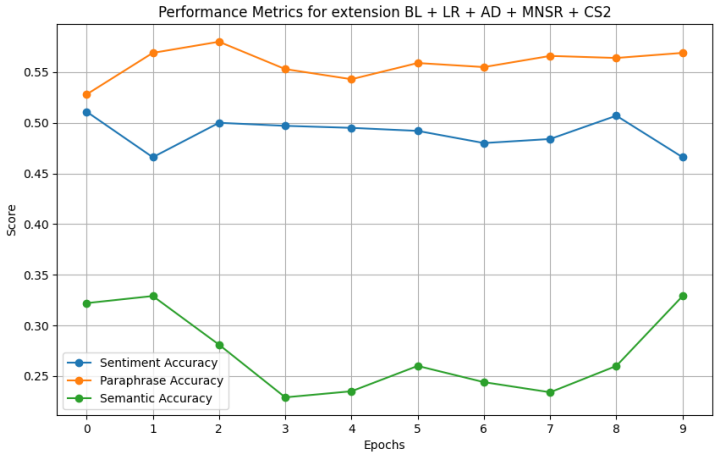
Matthew Henderson, Rami Al-Rfou, Brian Strope, Yun hsuan Sung, Laszlo Lukacs, Ruiqi Guo, Sanjiv Kumar, Balint Miklos, and Ray Kurzweil. 2017. Efficient natural language response suggestion for smart reply.

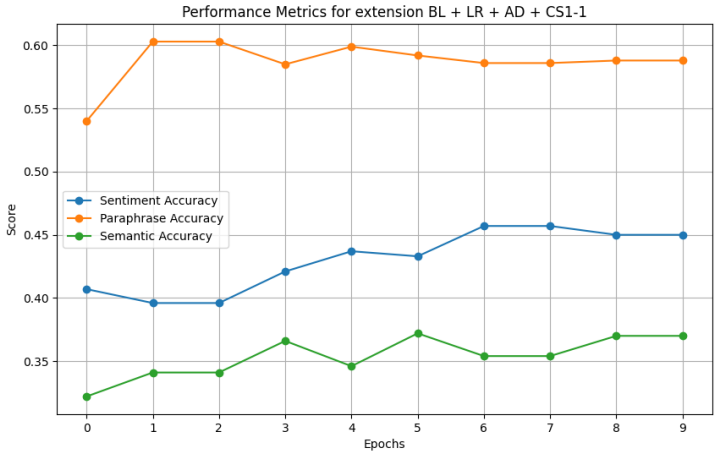
Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Tuo Zhao. 2020. SMART: Robust and efficient fine-tuning for pre-trained natural language models through principled regularized optimization. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2177–2190, Online. Association for Computational Linguistics.

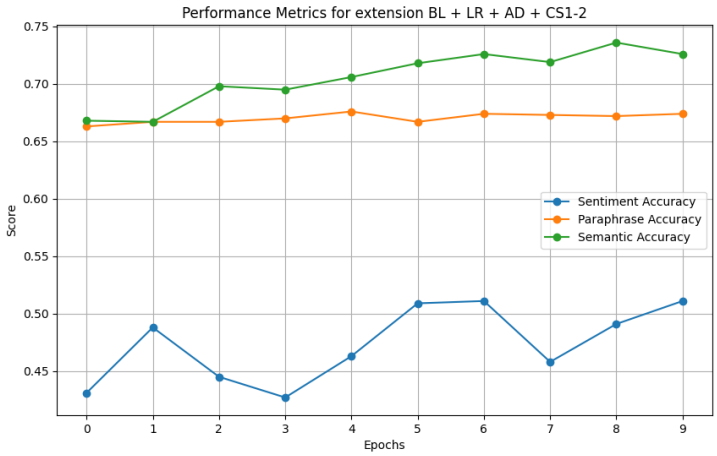
Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3982–3992, Hong Kong, China. Association for Computational Linguistics.

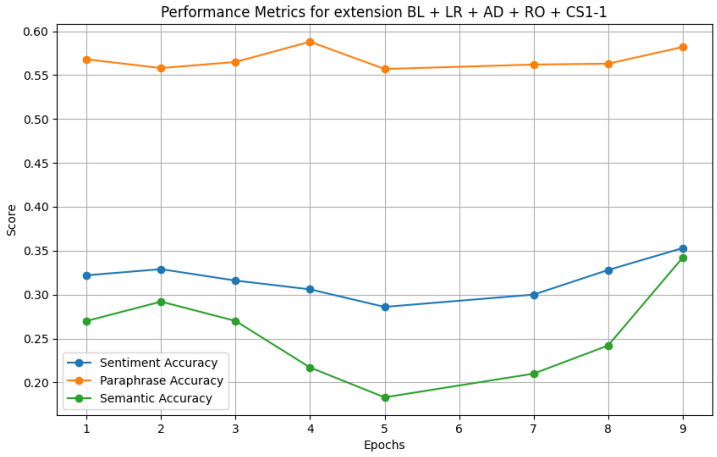
A Appendix

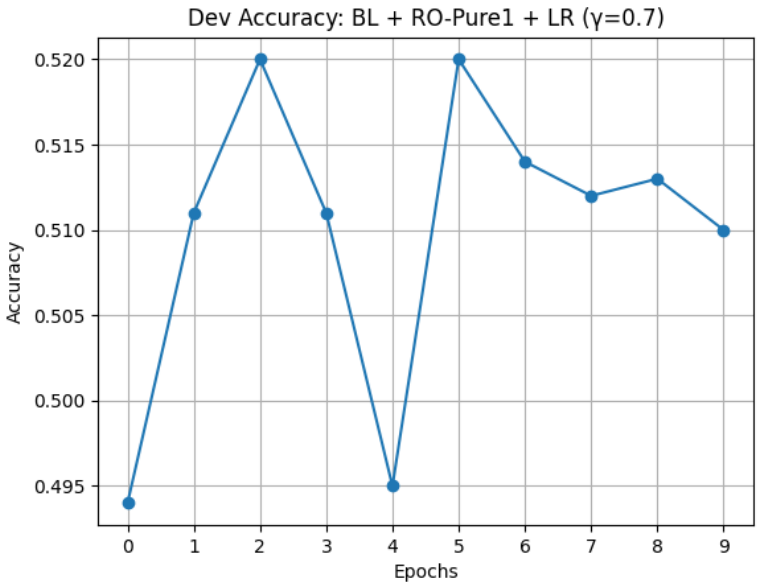
Below contains the **dev** accuracies and Pearson scores of our different models on the three tasks during the 10 epochs of training.






Performance Metrics for extension BL + LR + AD + CS1-2
Score
Epochs
Sentiment Accuracy
Paraphrase Accuracy
Semantic Accuracy


Performance Metrics for extension BL + LR + AD + RO + CS1-1
Score
Epochs
Sentiment Accuracy
Paraphrase Accuracy
Semantic Accuracy


Dev Accuracy: BL + RO-Pure1 + LR (γ=0.7)
Accuracy
Epochs