Maximizing MinBert for Multi-Task Learning

Stanford CS224N Default Project

Shumann Xu
Department of Computer Science
Stanford University
srxu@stanford.edu

Jordan Paredes
Department of Computer Science
Stanford University
jordan14@stanford.edu

Abstract

In this study, we investigate the efficacy of multi-task learning within a BERT fine-tuning framework, focusing on optimizing performance for sentiment analysis, paraphrase detection, and semantic textual similarity tasks simultaneously. We explore different model variants, incorporating techniques such as round-robin training, the application of cosine similarity in the STS head, adding additional Head NN layers, lexicon encoding before task-specific layers, SMART Regularization, and a multi-layered transform. Overall, we found that the combination of round-robin training, cosine similarity in the STS head, multiple task-specific layers, and lexicon pre-processing yielded the best results.

1 Key Information to Include

External collaborators:

Peter De la Cruz
Department of Computer Science
Stanford University
pdlcruz@stanford.edu

CS224N staff mentor: Heidi Zhang

Team Contributions:

Shumann: Lexicon Encoding, SMART Regularization, Baseline Bert implementation.

Jordan: Round-Robin, CosineSim, Layer Complexities, Additional SST training, Graphs

2 Introduction

In this project, we harness the advanced capabilities of BERT to build a multi-task classifier targeting sentiment analysis, paraphrase detection, and semantic textual similarity classification. By integrating task-specific layers atop shared BERT-based embeddings, our model aims to efficiently learn and classify input under varied specifications, benefiting from the shared learning process.

In this study, we explore the impact of employing complex versus simple model architectures for our three specific downstream tasks. Simultaneously, to enhance the performance of a BERT base model in multi-task learning scenarios, we will analyze the impact of applying various sophisticated strategies: Cosine-Similarity Fine-tuning, Round-Robin Training, Lexicon Encoding, Additional Neural Network Layers for Each Head, and Smoothness-Inducing Regularization, and Multi-Layered Transform. Our approach includes utilizing cosine similarity among BERT embedding pairs followed by a linear layer to derive final logits. Moreover, we adopt round-robin training to utilize the effectiveness of multi-task learning. We also implemented lexicon encoding for pairwise input NLP tasks (paraphrase detection and semantic similarity) to generate a single embedding from concatenating sentence inputs with a separator. Furthermore, we utilized 3 task-specific linear layers with batch normalization and dropout. Additionally, we implement smoothness-inducing regularization by introducing minor perturbations to the inputs, aiming for stable model output. The assessment of these strategies, reveals certain methodology's effectiveness over others in enhancing the multi-task learning performance of our model, highlighting the advantages of integrating such techniques in complex model designs.

3 Related Work

In our study, we build upon the foundational work of Vaswani et al. (2017), which introduced the BERT model, highlighting its innovative use of deep bidirectional architectures for NLP. This advancement underlined the significance of unsupervised pre-training in enhancing language understanding systems, setting a new benchmark for the breadth of NLP tasks that can benefit from pre-trained models. Within the scope of fine-tuning techniques, we explore various strategies that have shown to enhance model performance significantly.

Our project particularly emphasizes the power of multi-task learning as suggested by Liu et al. (2019). For tasks necessitating the comparison of two inputs, such as paraphrase identification and semantic similarity evaluation, we approach by concatenating these inputs with a delineating separator [SEP]. Subsequently, we channel the contextual embeddings generated by BERT through multiple feed-forward layers.

Our research also leverages the contributions of Jiang et al. (2020). This work proposes a novel fine-tuning framework that addresses the challenge of maintaining model generalizability to new, unseen data. By integrating smoothness-inducing regularization and Bregman proximal point optimization, the SMART framework significantly mitigates the risks of overfitting during fine-tuning.

4 Approach

4.1 Baseline Bert

In the initial phase of our project, we focused on implementing key components of the BERT architecture as described by Devlin et al. (2019). Our baseline mini-BERT model processes input sentence pairs, tokenized into a sequence of 512 tokens, inclusive of the special `CLS` and `SEP` tokens. It incorporates an embedding layer that combines word embeddings (for 512 tokens), position embeddings (for 512 positions), and token type embeddings (for 2 token types), each contributing to an embedding dimension of 768.

Task-specific adaptations are made atop the mini-BERT structure without altering its foundational layers:

- **Sentiment Classification (SST):** Performs multi-class classification using cross-entropy loss function.
- **Paraphrase Detection:** Performs Binary classification on concatenated BERT embeddings using binary cross-entropy loss function.
- **Semantic Textual Similarity (STS):** Performs Linear Regression on concatenated BERT embeddings using Mean Squared Error loss function. This approach was later refined to classify based on cosine similarity between the input embeddings, converting similarity scores from the range of -1 to 1 to a scale of 0 to 5, matching the task's label range.

For each NLP task, we finetune separately on their respective datasets.

4.2 Lexicon Encoder

In the preprocessing phase for tasks involving the analysis of sentence pairs, we adopt a strategy of concatenating each sentence pair with a separator token in between, resulting in an output format of [input1; [SEP]; input2] as mentioned in Liu et al. (2019). Following this, we create attention masks based on these concatenated pairs to inform the BERT model which parts of the input are relevant. The processed input is then passed through BERT to derive contextual embeddings. A trainable linear layer is subsequently applied atop these embeddings to produce the final task-specific outputs.

For sentiment analysis, we use the same techniques as the baseline version.

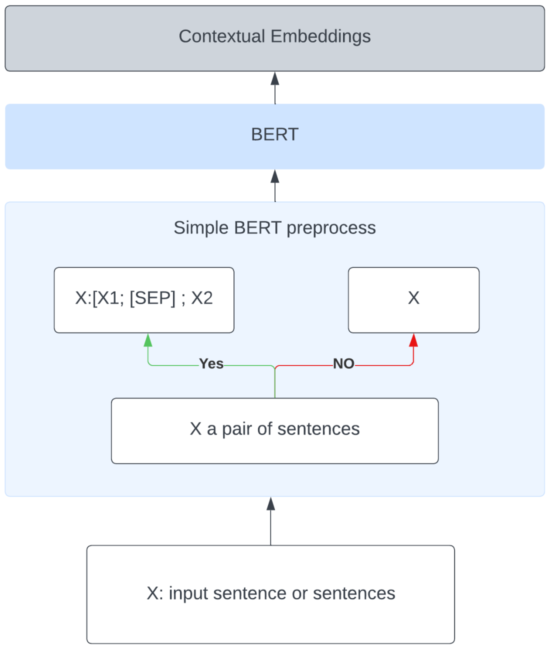


Figure 1: Lexicon Preprocessing

4.3 Add NN Layers + Cosine Sim to Head

In our effort to boost task-specific performance, we've enhanced the model by integrating a sequential layer structure comprising three linear layers, each followed by batch normalization and GELU activation functions, into the final output mechanism for each task. This architectural enhancement substantially enlarges the model's parameter count, thereby extending the necessary training time. However, the expected benefit is a significant improvement in the model's effectiveness (Rebuffi et al., 2017).

The cosine similarity between two vectors X and Y is defined as:

$$CosSim(X, Y) = \frac{X \cdot Y}{\|X\|\|Y\|}$$

This metric evaluates the cosine of the angle between two vectors in a multidimensional space, serving as a measure of the distance between them. In applications such as paraphrase detection and semantic textual similarity, this principle is utilized to assess the degree of similarity between two sentences. If the embeddings generated by BERT are semantically rich, sentences with similar meanings should produce closely aligned embeddings.

In our framework, to incorporate the cosine similarity between two embeddings, we considered two methods: substituting the linear layer entirely with cosine similarity, combining the embeddings with their cosine similarity before inputting them into the linear layer, or augmenting the output of the linear layer with the cosine similarity as a final step. Prior research indicated that the most effective approach is to enhance the linear layer's output by integrating cosine similarity (Reimers and Gurevych, 2019). In our experiments, we also initially applied this method to both sentiment analysis and paraphrase detection tasks. However, we soon found that it offered advantages mainly for Semantic Similarity, leading us to focus our subsequent experiments on this area.

Note that we must add a multiplication factor of 5 to the cosine similarity for semantic similarity since targets fall in the range of [0, 5].

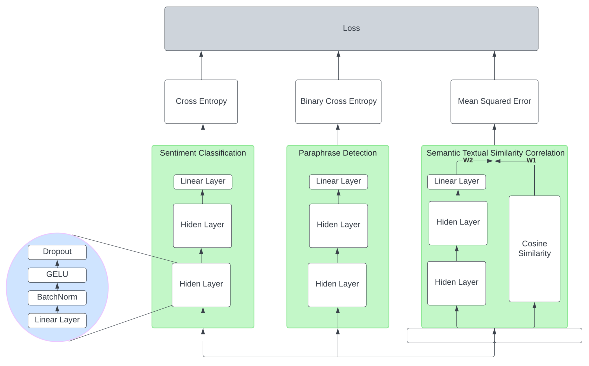


Figure 2: Task-Specific NN Layers + Cosine Sim

4.4 Round-Robin

To foster a model that better generalizes across these varied tasks, we adopted a multi-task learning strategy, specifically implementing a batch-level round-robin technique. Starting with the pre-trained BERT framework, we alternate between the sentiment analysis, paraphrase detection, and semantic textual similarity datasets, maintaining a consistent batch size throughout the process. In each iteration, we perform updates on the parameters relevant to the current task dataset in a single pass, aiming for a more integrated and generalized learning outcome across tasks.

4.5 Additional Transform Layers

The motivation to expand our model with additional transformation layers stems from the need to enhance the network's capacity for feature representation and improve its generalization abilities across diverse tasks. By integrating more layers, such as Batch Normalization, GELU activation functions, and Dropout, to each task we hoped to achieve a deeper and more complex model architecture.

4.6 Augmenting SST Training Examples

We noticed that the performance of SST was falling short compared to that of STS and paraphrase detection. We believe that volume and diversity of training data significantly influence model performance (Jiang et al., 2020). With SST dev test having suboptimal development scores in SST, we tried enhancing the dataset to improve these metrics.

To achieve this, we implemented two primary methods. Firstly, we utilized a back-translation process, where selected sentences were translated into a foreign language and then back to English, producing subtle variations of the original sentences. Secondly, we developed a GPT-based tool to generate semantically similar sentences with the same sentiment polarity. These methods effectively expanded our SST dataset, introducing more nuanced examples for the model to learn from.

4.7 SMART Reg

Within the SMART framework, Smoothness-Inducing Adversarial Regularization and Bregman Proximal Point Optimization are key components that contribute to the model's performance enhancement.

4.7.1 Smoothness-Inducing Adversarial Regularization

This approach introduces a regularization term into the model's loss function to promote the generation of smooth and continuous outputs. The technique leverages adversarial loss to quantify the discrepancy between the model's actual output and a smoother version of this output, which can be achieved through low-pass filtering or similar operations. For a given model $f(\cdot;\theta)$ targeting a task with input embeddings x_i and outputs y_i (viewed as labels for simplicity), the optimization goal is formulated as:

$$\min_{\theta} F(\theta) = L(\theta) + \lambda_s \times R_s(\theta)$$

Here, $L(\theta)$ denotes the task-specific loss function, $R_s(\theta)$ the regularization function, and λ_s a fine-tuning parameter that adjusts the regularization strength to match task variability.

4.7.2 Bregman Proximal Point Optimization

To mitigate aggressive parameter updates, the SMART framework incorporates Bregman proximal point optimization, applying a significant penalty for large, abrupt model adjustments. This technique optimally minimizes a convex function $f(x)$ within convex constraints through an iterative resolution of proximal sub-problems. Each iteration applies a proximal operator to the current iterate x to produce $x + 1$, progressively approaching the optimal solution. When applied to the pre-trained BERT model $f(\cdot; \theta)$, the update for iteration t is given by:

$$\theta^{t+1} = \arg\min_{\theta} F(\theta) + \nu \times D_{Breg}(\theta, \theta^t)$$

The Bregman PPO divergence, $D_{Breg}(\theta, \theta^t) = ls(f(\bar{x}_i; \theta), f(x_i, \theta^t))$, with ν acting as a tuning parameter for the divergence, aims to constrain updates to a defined neighborhood around the previous iteration, effectively establishing a trust-region-like regularization to curb overly aggressive updates.

5 Experiments

5.1 Data

- **Stanford Sentiment Treebank (SST) Dataset:** Comprising 11,855 single sentences from movie reviews, this dataset includes 215,154 unique phrases derived from parse trees generated by the Stanford parser. Each phrase has been annotated by three human judges with labels ranging from negative, somewhat negative, neutral, somewhat positive, to positive. Our goal is to utilize BERT embeddings to predict the sentiment classification labels of these sentences.
- **Quora Dataset:** Featuring 400,000 question pairs with binary labels indicating whether the questions are paraphrases of each other (labelled as No and Yes), this dataset is employed to ascertain if a pair of sentences are paraphrases.
- **SemEval (STS) Benchmark Dataset:** This dataset consists of 8,628 sentence pairs, each labelled with a similarity score ranging from 0 (unrelated) to 5 (equivalent meaning). Our objective is to assess the semantic textual similarity of these sentence pairs using BERT embeddings.

5.2 Evaluation method

To evaluate the performance of our models across the three tasks, we adopted a straightforward qualitative evaluation approach as follows:

1. **Sentiment Classification:** Accuracy was determined based on the proportion of labels correctly predicted by the model in the Stanford Sentiment Treebank dataset.
2. **Paraphrase Detection:** Accuracy was similarly calculated according to the number of correctly identified paraphrase labels in the Quora dataset.
3. **Semantic Textual Similarity:** Accuracy was assessed using the Pearson correlation coefficient between the predicted similarity scores and the actual labels in the SemEval STS Benchmark dataset, reflecting the degree to which the model's predictions align with human judgment.

5.3 Experimental details

In our investigation, we methodically explore distinct fine-tuning techniques, evaluating each model variant's average performance across the Stanford Sentiment Treebank (SST), Quora, and Semantic Textual Similarity (STS) datasets.

For all experimental setups described in **Approach**, we employ the ADAM optimizer for training. Each dataset utilizes an identical feedforward head attached to BERT. In scenarios where BERT parameters are frozen and only the weights of the linear layer are trained (pretrain), we adopt a learning rate of $1e-3$. Conversely, when BERT parameters are also being updated (finetune), a lower learning rate of $1e-5$ is utilized to prevent overfitting. Due to limitations imposed by GPU memory, we set the batch size to 32 and conduct each experiment over the course of 10 epochs with a 0.3 dropout rate, selecting the iteration demonstrating peak performance on the development set as our model of choice.

To assess the viability of each fine-tuning approach, we opted to gauge its effectiveness based on the development set accuracies obtained during the pre-training phase of the respective model. This decision was influenced by the extensive range of features we intended to explore; dedicating time to fully fine-tune each model under every strategy would be exceedingly time-consuming. Consequently, should there be a noticeable drop in the pre-training performance, we resolved to forego fine-tuning the model using that particular approach.

5.4 Results

Below is a table of our accuracies from our various models on the Pretrain and Develomental Datasets as well as train times.

	Pretrain			Finetune		
Method	Sentiment Acc	Paraphrase Acc	STS Corr	Sentiment Acc	Paraphrase Acc	STS Corr
A	0.307	0.635	0.198	0.374	0.721	0.42
B	0.323	0.666	0.244	0.415	0.765	0.54
C	0.381	0.691	0.309	0.458	0.77	0.575
D	0.381	0.694	0.452	0.396	0.855	**0.87**
E	0.394	**0.712**	0.508	**0.503**	**0.883**	0.822
F	**0.408**	0.645	**0.586**	0.48	0.883	0.752
G	0.386	0.707	0.496		-	
H	0.37	0.707	0.396		-	

Table 1: Pretrain and Finetune Dev Acc for Pretrain and Finetune tasks

A

Method	Pretrain	Finetune	Figures
A	0.4	0.5	
B	0.45	0.55	
C	0.6	1.2	4, 5
D	0.4	0.5	
E	0.7	1.4	6, 7
F	0.75	1.5	8, 9
G	0.8	-	10
H	2.4	-	11,

Table 2: Train time (in hours) per epoch for Pretrain and Finetune tasks. For specific method details see Figures 4 and 5.

Legend:

A: Manual Cosine, 0.5 Split, w/ NN

B: Expanded 3 layers for each task, plus weighted cosine split

C: Round Robin, Batch Norm, Weight Decay, expanded 3 layers, weighted split

D: Lexicon encode

E: Lexicon encoder, Expanded 3 layers, weighted split, Batch Norm, Weight Decay, Round Robin

F: Forward Function change, Lexicon encoder, Expanded 3 layers, weighted split, Batch Norm, Weight Decay, Round Robin

G: More sst training examples + E

H: SMART Regularization + E

Our initial modification aimed to integrate cosine similarity into our model while retaining the advantages of a Neural Network (NN), leading to a balanced 0.5 split specifically for the semantic textual similarity task. Recognizing the need for dynamic adaptability, we introduced learnable weights, allowing the model to adjust the emphasis between cosine similarity and NN based on performance.

To broaden our focus to other tasks, we enhanced the neural networks for each task by incorporating Batch Normalization, Dropout, and Gaussian Error Linear Units to introduce non-linearities, which as seen improved accuracies by a very small amount.

We then shifted from task-specific training to a more balanced approach by implementing Round Robin training, which alternates between tasks within each epoch. This method along with Batch Norm, and Weight Decay significantly benefited all tasks, indicating a positive overall impact on the model.

Exploring further, we integrated Lexicon encoding for Paraphrase and Semantic Textual Similarity tasks, as it improved our base model substantially so we decided to add it out our main model as seen by the accuracies.

From here we tried other ideas as well such as transforming the Forward function in the Multi-taskBERT class and introducing additional dropout. This as shown negatively affected performance, so we didn't pursue this idea further.

We also aimed to expand sentiment classification data through back-translation and a GPT-wrapper in order to come up with more data samples. As seen in the pretraining phase it did not yield significant improvements and lead to GPU problems when fine tuning.

Lastly, we experimented with SMART regularization to address overfitting, evidenced by the discrepancy between training error and accuracy. Despite its potential, GPU limitations and its great increase in training times discouraged us from this approach.

6 Analysis

Our experimentation highlighted the efficacy of incorporating lexicon encoding into our model, which notably enhanced its performance. We attribute this success to the inadequacy of merely concatenating embeddings from two sentences and processing them through a linear layer, especially for tasks like Quora and STS, where understanding the symmetric relationship between sentence pairs is crucial. Lexicon encoding effectively bridges this gap by providing additional semantic cues that enable more nuanced comprehension of sentence relations.

Moreover, the implementation of round-robin training proved to be a pivotal strategy, especially observed after integrating lexicon encoding. Initial results showed a discrepancy between sentiment analysis performance and individual development set accuracies for the Quora and STS tasks, suggesting that solely relying on lexicon encoding was insufficient for holistic model improvement across tasks. The adoption of round-robin training not only addressed this issue but also underscored its importance in achieving superior multitasking outcomes. It indicated that systematic alternation between tasks during training is essential for the model to generalize effectively across different NLP tasks, thereby enhancing its overall utility and performance.

On the other hand, our experiments also shed light on the limitations of certain techniques. For instance, attempting to implement SMART regularization and adding SST training examples both led to GPU memory constraints, primarily due to its complex computational demands. This difficulty was compounded by our model's already extensive parameter set, derived from adding various features including lexicon encoding and additional neural network layers, which strained our GPU memory capacity, even after reducing batch sizes.

Similarly, the introduction of extra feedforward layers unexpectedly reduced accuracies. This reduction could stem from multiple reasons. First, increasing model complexity does not always correlate with better performance, especially if the added layers contribute to overfitting. With more parameters to train, the model may focus too much on the training data's noise, impairing its ability to generalize to unseen data.

Below highlights the model architecture of our best model:

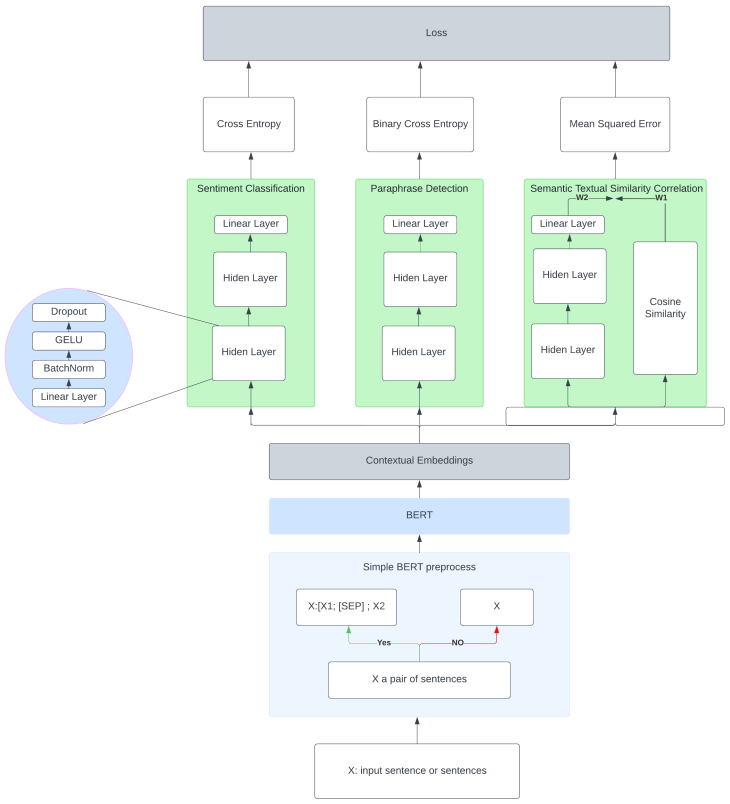


Figure 3: Model Architecture

7 Conclusion

In our study, we explored the impact of integrating cosine similarity and additional NN Layers to task heads, lexicon preprocessing, round-robin training, additional feed forword layers, and SMART regularization on the performance of an enhanced BERT model. Our experiments revealed the most effective pretrained + finetuned model was enhanced with lexicon preprocessing and cosine similarity on the STS Head trained in a round-robin manner across three datasets different, and equipped with 3 linear layer prediction heads. By integrating these approaches, we achieved a 46% improvement in performance over the baseline method. However, we noticed that although these results were better, it came at the cost of increased training complexity, as it more than doubled training time (refer to A).

This endeavor deepened our understanding of BERT's architecture, explored various enhancement techniques for this model, and provided insights into the broader research landscape of performing NLP tasks.

Further research should investigate the potential benefits of SMART regularization on the model's performance, especially considering its untested promise due to GPU memory constraints encountered in our initial attempts. Additionally, a more granular analysis of the model's features through separate testing could provide clearer insights into their individual contributions and effectiveness.

References

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

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.

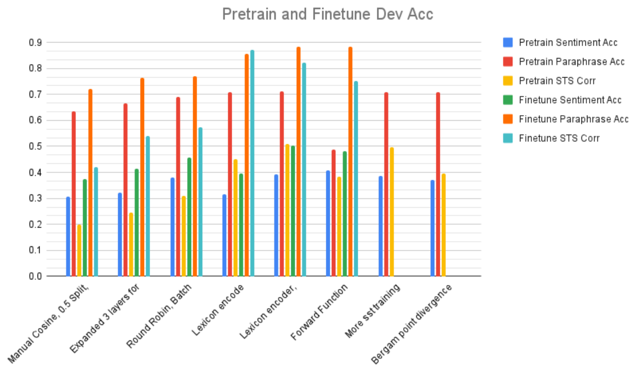
Xiaodong Liu, Pengcheng He, Weizhu Chen, and Jianfeng Gao. 2019. Multi-task deep neural networks for natural language understanding. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4487–4496, Florence, Italy. Association for Computational Linguistics.

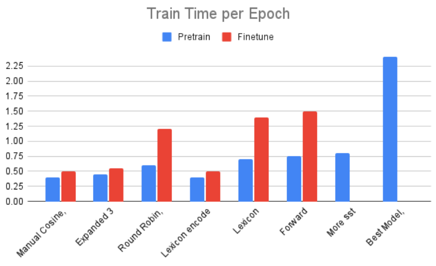
Sylvestre-Alvise Rebuffi, Hakan Bilen, and Andrea Vedaldi. 2017. Learning multiple visual domains with residual adapters.

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.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need.

A Appendix (optional)





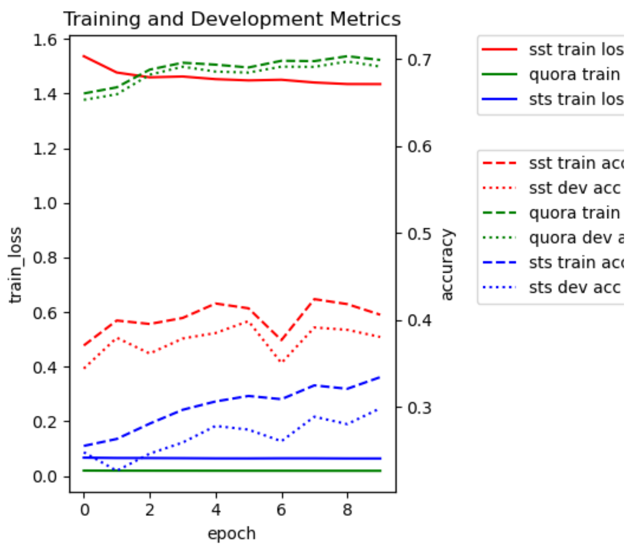


Figure 4: C Pretrain training

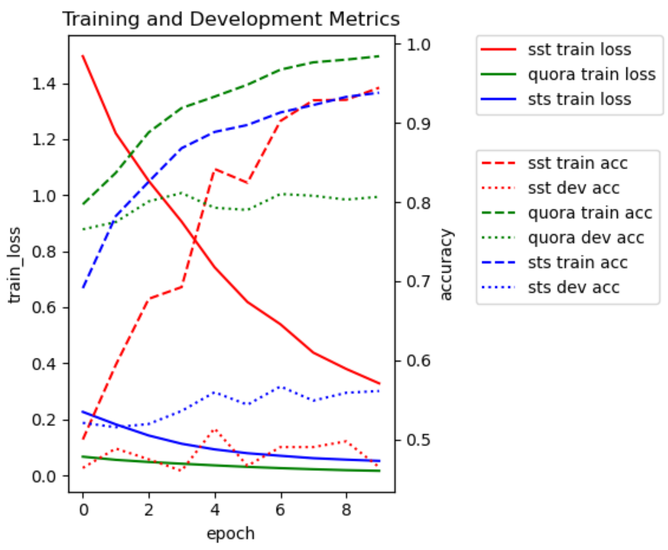


Figure 5: C Finetuning training

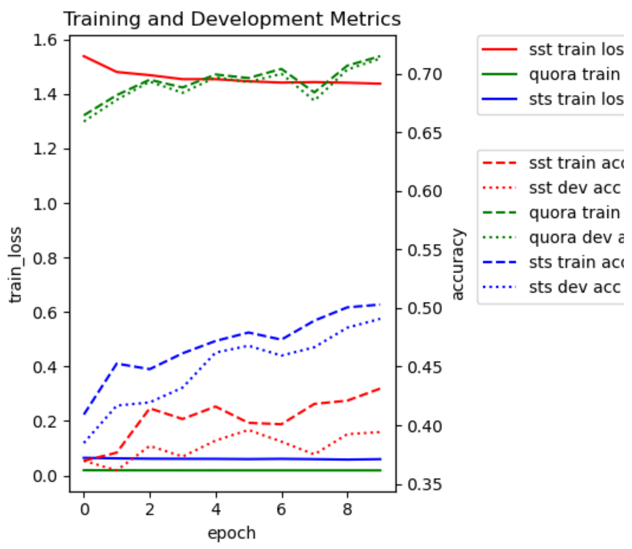


Figure 6: E Pretrain training

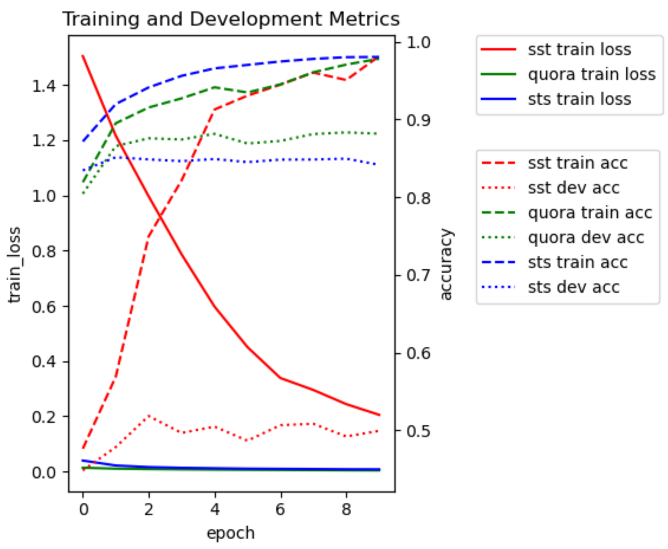


Figure 7: E Finetuning training

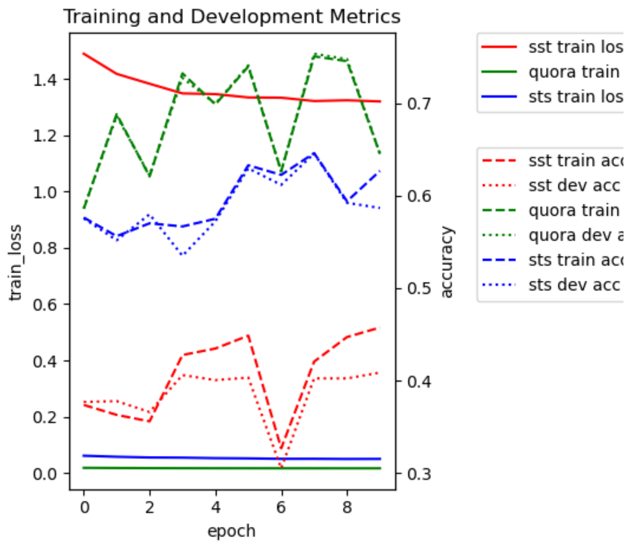


Figure 8: F Pretrain training

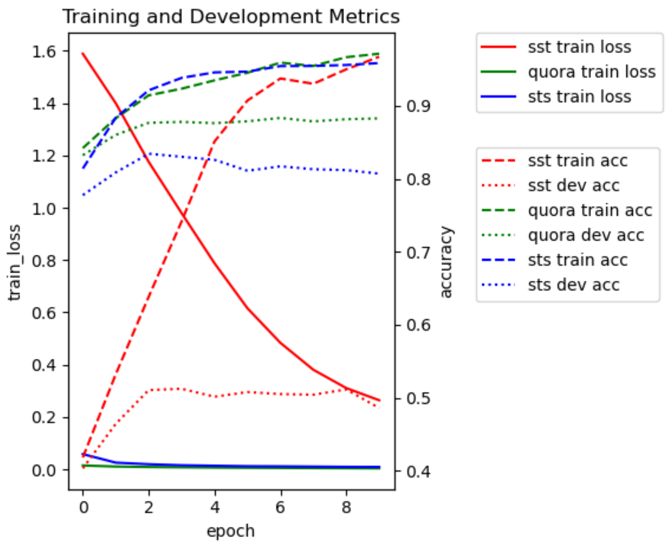


Figure 9: F Finetuning training

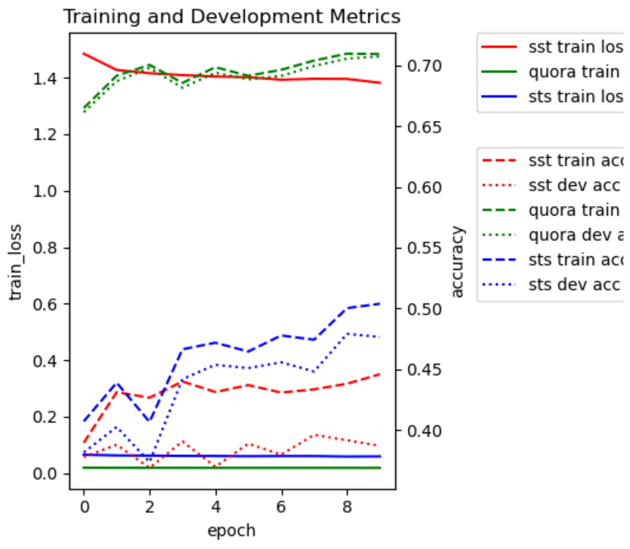


Figure 10: G Pretrain training

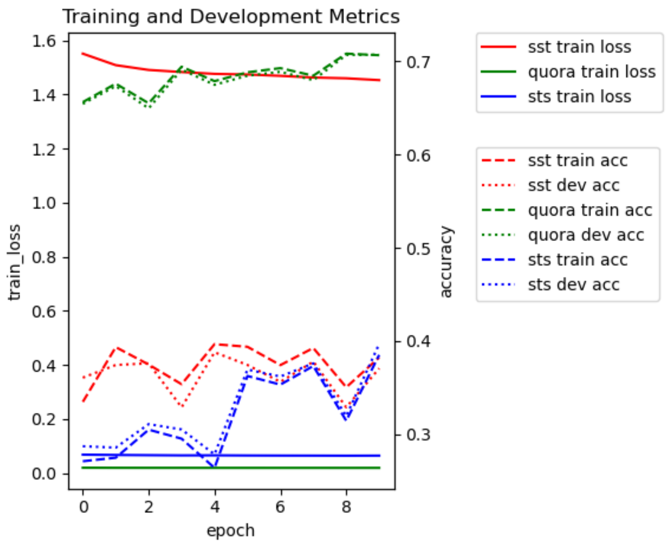


Figure 11: H Pretrain training