BERTogether: Multitask Ensembling with Hyperparameter Optimization

Stanford CS224N Default Project

[1]Ivan Liongson, [2]Erik Luna
[1]Department of Computer Science, [2]Department of Electrical Engineering
Stanford University
{ivanlion, eluna1}@stanford.edu

Abstract

This paper presents BERTogether, a novel approach to multitask learning that leverages the strengths of minBERT, an efficient BERT-based model, optimized for high performance across three NLP tasks: Sentiment Classification, Paraphrase Detection, and Semantic Similarity Evaluation. We introduce a weighted-average 3-BERT ensemble method for each task coupled with hyperparameter tuning to enhance performance. Our methodology outperforms baseline models by using SMART regularization techniques and adding an extra dataset for Similarity Evaluation. We demonstrate significant improvements in accuracy and correlation metrics via weighted ensembling.

1 Key Information:

- Mentor: Timothy Dai
- External Collaborators: None
- Sharing project: No

2 Introduction

Having erupted in 2018, BERT (Bidirectional Encoder Representation from Transformers) is a popular high performing NLP model, and any researchers use this transformer model in ensembles for multitask classifiers to take advantage of the transformer model. For example, on the GLUE[1] 9-task benchmark, which includes all three of our tasks, most of the models on the performance leaderboard are ensembles, many of which are BERT based. Thus, we implement a model of similar design. In this paper, we develop a better performing BERT-base multitask classifier for Sentiment Evaluation, Paraphrase Detection, and Semantic Text Similarity. From the standard datasets, Stanford Sentiment Treebank (SST), Quora, and SemEval STS Benchmark. Given three different tasks (Sentiment classification, Paraphrase detection, and Semantic Textual Similarity), we must design a minBERT-based model that will produce the greatest overall score (accuracy/correlation). Our goals to improve our base MultitaskBert model include weighted ensembling and hyperparameter tuning. For reference, our original baseline used simple average ensembling with multitaksBert is 37.9% sentiment accuracy, 38.0% paraphrase accuracy, and -0.009 similarity correlation.

We aim to improve our accuracy for sentiment evaluation and paraphrase detection and our Pearson correlation score for semantic similarity prediction through multiple improvements, both to the individual multitask classifier and by generating a higher-level ensembling structure. We went beyond naïve ensembling for better results We found that ensembling by weighted average boosted performance across all three tasks.

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

To solve these problems, we implemented dataloading for sentence-pair datasets to pretrain and finetune for the sentence-pair tasks and an additional similarity dataset, SICK. Additionally, to improve our model architecture, we incorporated SMART loss and regularization and implemented weighted average ensemble model on a per-task basis. For each of the tasks, we had 3 of our best models pretrained and finetuned on its respective dataset with slightly different parameters. Finally, we applied smart hyper-parameter tuning for individual models as well as weight optimization for our weighted ensemble using the Optuna library.

We found that training on the task-specific datasets immediately boosted performance from our baseline, and adding an additional dataset for semantic similarity also improved our performance. We also implemented SMART loss which helped us with a 1% improvement for semantic similarity. Hyper-parameter tuning helped us increase our accuracy on individual models and the learned weight ensembling helped us even more to improve our scores. Our best configuration achieved scores of: (insert scores).

3 Related Work

While some BERT ensemble models incorporate very complex methods and numerous BERT models (up to 25 BERTs as used by Liu et al. (2019)), we use minBERT, a lightweight BERT variant bypasses the complexity of ensembles models and presents an efficient while conserving complexity that other high-resource approaches may overlook. We prove that we can still improve the performance of this smaller model by ensembling several models together.

Additionally, our project builds upon the work of Bao et al. (2021), who introduced a meta-learning method to refine sentiment analysis of literary book reviews. Their approach, which included a 'Looking Back' mechanism, mitigated the effects of noisy labels and class imbalances, achieving an increase in F1 scores. Inspired by this, we applied weight optimization strategies to our minBERT multitask classifier, aiming to enhance accuracy beyond our baseline. Our model leverages the inherent strengths of minBERT, and through a similar yet distinct application of meta-learning principles, we observed improvements in task performance, particularly in sentiment classification.

To reduce overfitting within our minBERT multitask classifier, we draw upon the work by Jiang et al. (2019), who present a principled approach to fine-tuning pre-trained language models. Their SMART framework introduces Smoothness-Inducing Adversarial Regularization and Bregman Proximal Point Optimization to mitigate overfitting—a critical challenge due to the complexity of these models and limited downstream task data. By integrating a similar adversarial regularization, our methodology not only prevents overfitting but also stabilizes training. This incorporation of adversarial strategies is in line with current advancements in robust fine-tuning techniques, which have demonstrated success in a variety of NLP benchmarks. The SMART regularization and and optimization technique was previously used on already high-accuracy models but with minimal increases in performance. For tasks such as Similarity Evaluation, we have a very limited dataset at only so our learning may be too low for test accuracy or if we train for longer, we run into overfitting. Our work investigates this by including it in our approach to prevent this overfitting while learning.

4 Approach

Our baseline model before any extensions, MultitaskBERT-SST-only, was only trained on the SST dataset. Its architecture is outlined in the default assignment instructions. This SST-only baseline's baseline scored sentiment accuracy 31.1%, paraphrase accuracy 38.0% , and similarity correlation r = -0.009.

These are the architectural improvements we made to our neural network to improve its performance.

- Training on Quora and SemEval STS
- External dataset pretraining on SICK for SST
- Adding SMART loss for SST
- Momentum Bregman proximal point (MBPP) optimization
- Hyperparameter tuning
- Weighted average ensembling with weight optimization

4.1 Full Dataset loading

We added pretraining on the sentence-pair datasets to improve the model's knowledge of the paraphrase and semantic similarity tasks. This required implementing a Dataloader for paired sentences. This is discussed in further detail in Data Section 5.1.

4.2 External Dataset

We used an external dataset, SICK, which required preprocessing. See more in Data Section 5.1.

4.3 SMART: Regularization and Bregman Optimization

As we trained our MultitaskBert, we noticed high (>97%) accuracy on the training set but much lower performance on the dev and train set. We attributed this to possible overfitting, so we investigated the regularization proposed in the SMART method Jiang et al. (2019). The SMART technique (SMoothness-inducing Adversarial Regularization and BRegman pRoximal poinT opTimization) is for regularization to induce smoothing and prevent overfitting. Additionally, we attempted their proposed Bregman proximal point optimization for smaller granularity parameter updates while training.

4.3.1 Loss

We included SMART Loss into our similarity task training loop in hopes of improving its performance. We imported the smart_pytorch library[2] which was developed based on the techniques proposed by Jiang, et al. We used their same constant values $\epsilon = 10^{-5}$, standard deviation for the loss as $\sigma = 10^{-5}$, $\mu = 1$, and hyperparameter weight $\lambda_s \in \{1, 3, 5\}$. We used the default symmetrical KL divergence loss function as outlined in the paper:

$$l_s(P, Q) = D_{KL}(P \parallel Q) + D_{KL}(Q \parallel P)$$

4.3.2 MBPP

In addition to the regularization function implemented above, we investigated Bregman Proximal Point optimization of the model parameters. We saw this would lead to more accurate θ updates to solve the equation below.

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

where $L(\theta)$ is the original loss function and $R_s(\theta)$ is regularization loss. The learning rate η from the paper's Algorithm-1 was set as 10^{-3}. We set $\beta = 0.99$ for the initial 10% of the updates ($t \leq 0.1T$) and $\beta = 0.999$ for the rest of the updates ($t > 0.1T$). This was not included in our final version since we needed further debugging, but we were expecting some slight improvements. The paper, however, only noticed a slight 1% increase with both loss and MBPP in their approaches, so the expected gain was not drastic.

4.4 Weighted Ensemble Modeling with Weight Optimization

We established a basic framework for running an ensemble using multiple models for each of the three tasks. We implemented a class called 'EnsembleModelWrapper' as well as a function to load all .pt pytorch models from a subdirectory. This method facilitates easy experimental iteration, as we can use and swap out any number of our saved trained models into the ensemble. Originally, the wrapper used a naive ensembling method of averaging scores across all models, then outputs predictions in the same format that a single model would. This allows us to use the provided `model_eval_multitask` without modification.

After this, we investigated more sophisticated approaches for joining the models. The approach which we went with was a weighted average of model outputs, then performing a optimization of the weights using another Optuna study with the n models per task, with the goal of maximizing the objective of the task's score (accuracy or correlation).

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

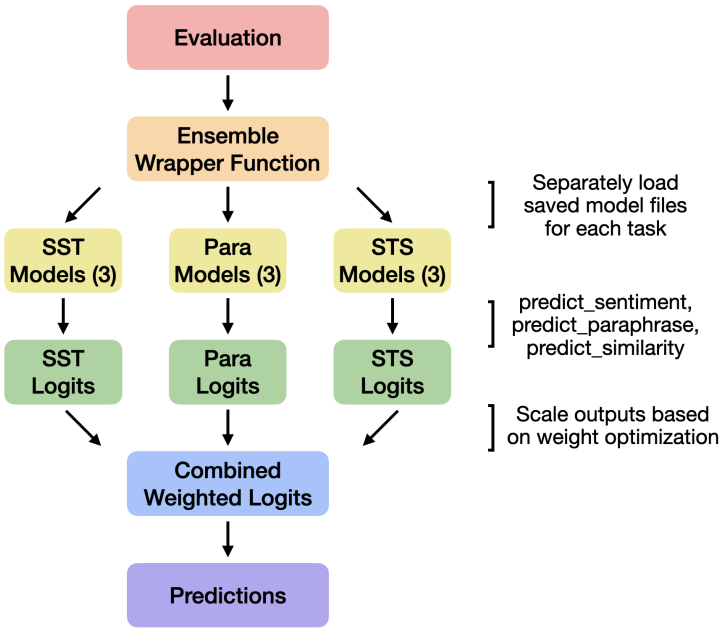

Figure 1: Evaluation pipeline on the EnsembleModelWrapper.

For each of three tasks, we had three of our best models with different parameters finetuned on its respective dataset. The parameters that we tweaked were learning rate, batch size, and hidden dropout probability p. The final step calculates a weighted average of the logits output by the 3 BERTs per task to evaluate the predictions as shown in Figure 5. In total, we use a 9 BERT ensemble architecture.

5 Experiments

5.1 Data

- Sentiment analysis
  - SST: 11,855 single sentences with 5 sentiment classes from negative to positive [0,4]
- Paraphrase detection
  - Quora subset: 141,506 sentence pairs that are paraphrases or not [0,1]
- Similarity evaluation, Pearson correlation score.
  - SemEval STS: 6,041 sentence pairs with relatedness score: [0,5] continuous
  - SICK: 9,839 sentence pairs with relatedness score: [1, 5] continuous

Firstly, one addition we made since the first milestone was that we added dataloading for the Quora and SemEval datasets for the Paraphrase and Similarity evaluation tasks, respectively. This allowed the necessary task-specific pretraining and finetuning needed to increase our scores. Loading these two datasets required creating a modified sentenceDataLoader method which now loaded 2 sentence_IDs instead of one.

Given that the SemEval STS Benchmark dataset was the smallest at only 6,041 training and 864 dev samples, we foresaw limited learning for the STS task on such a small dataset. To have more datapoints to train on, we utilized the SICK (Sentences Involving Compositional Knowledge) dataset which compiles almost 10,000 curated English sentence pairs labeled with semantic similarity scores 1-5, based on similarity and determined entailment Marelli et al. (2014). We added all of these to the training dataset for STS, for a total of 15,880 examples, an increase from 6,041. Preprocessing was necessary. We scaled the labels from 1 to 5, to 0 to 5 to match the format of STS. Our code for doing so is shown below:

```
def sickIds(id):
 # id assignment, no sts ids go above 9999
 return id + 10000

def interpolateSimilarity(input):
 # convert from SICK 1 to 5 to STS 0 to 5
 return (input - 1)/4*5
```

We recognize that this may introduce some noise since we are linearly mapping 5 continuous float labels into 6 discrete, but to mitigate this noise, we biased the distribution found in the original SST-5 dataset since it was more skewed towards higher 3+ scores. We found this improved our STS performance given the task-specific training. While we looked at other datasets for other tasks, we found this one the nearest to our needs without requiring much preprocessing and introducing too much noise.

By adding a new dataset for all three tasks since our milestone baseline, we made immediate improvements in our scores.

Our model performance was evaluated with the provided datasets:

- Paraphrase Analysis accuracy on dev set (1,101 examples)
- Paraphrase Detection accuracy on dev set (20,215 examples)
- Similarity Evaluation Pearson correlation score on dev set (864 examples)

5.2 Evaluation method

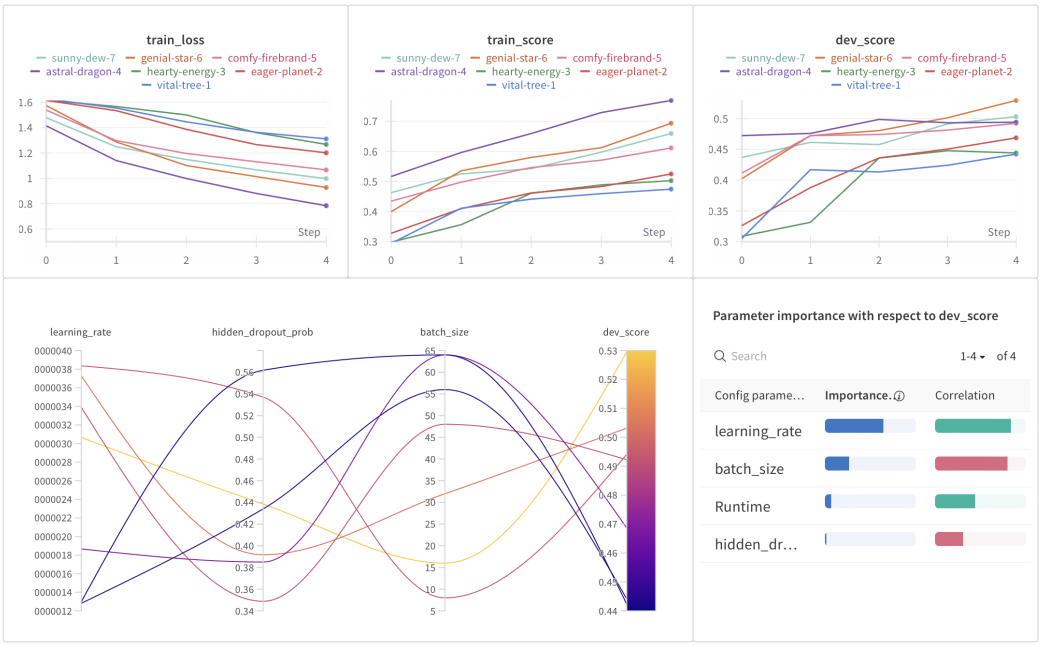

Figure 2: Sample results from a single hyperparameter optimization study. We use the Optuna library integrated with Weights & Biases for visualizations.

After pretraining and finetuning on each of the three tasks, we saved the models with the highest dev scores to construct the ensemble model using the EnsembleModelWrapper. From there, we evaluated the ensemble using same evaluation code provided to us. The evaluation function calculates the accuracy for the two classification tasks and the pearson correlation for the similarity scorer based on the test data and their labels. The provided test datasets sizes are: 2,210 sentiment, 40,431 paraphrase, and 1,726 similarity examples.

5.3 Experimental details

For many of our hyper parameters, including learning rate, training time, etc., were optimized using Optuna. Optuna is a hyperparameter optimization framework[3]. The Optuna hyperparameter optimization was a step done once we finished our model architecture and select a hyperparameter space to search over.

For other things, such as the SMART loss constants, we used the recommended values as recommended in the original paper.

We kept epoch count to 10. As we iterated through epochs when we trained, we would save the model that resulted in the highest dev accuracy per configuration. This would be the models that we would save for further use in the ensembled model. We ran our training and evaluation on our local NVIDIA GPU.

Finally, we optimized the weighted average with weights as tunable hyperparameters using Optuna.

5.4 Results

The below table shows dev scores (accuracy for STS/Para, correlation for STS) for different iterations of our model. The SST-Only and A and B baseslines are the same as in our project milestone report. The next three lines represent single models trained on each of the three datasetes with no ensembling. The last two lines show our ensembled models, consisting of three models per dataset for a total of nine models in the ensemble. The naïve model represents the scores from taking an unweighted average of model predictions, while the weight-optimized model was created by running another Optuna study to optimize the weights to apply to the logits outputted for each individual model.

Model Configuration	Sentiment Acc.	Paraphrase Acc.	STS Corr.
SST-only Baseline:	0.311	0.380	-0.009
A + B, SST-only:	0.379	0.380	-0.009
Single Model, SST-trained	0.510	0.537	-0.026
Single Model, Para-trained	0.227	0.783	0.052
Single Model, STS-trained	0.270	0.609	0.416
Naïve Ensemble (Mean) of 9 Models	0.527	0.797	0.419
Weight-Optimized Ensemble of 9 Models	0.533	0.800	0.427

6 Analysis

Our weakest tasks are Sentiment and Similarity, and we believe part of their low score is because of the small dataset that we trained on, compared to the higher performing Paraphrase task. Our approach took a very long time to train because we made the multitask classifier more complex, we ran an ensemble, and because we used Optuna for hyperparameter tuning. This was because it involved training on a STS dataset twice as large. The SMART loss function was also more complex than original, so it added to the compute time. Ensembles scale up our train time to about nine times as long. However, the larger ensembles did outperform the models trained on individual tasks.

[3] `https://optuna.readthedocs.io/en/stable/`

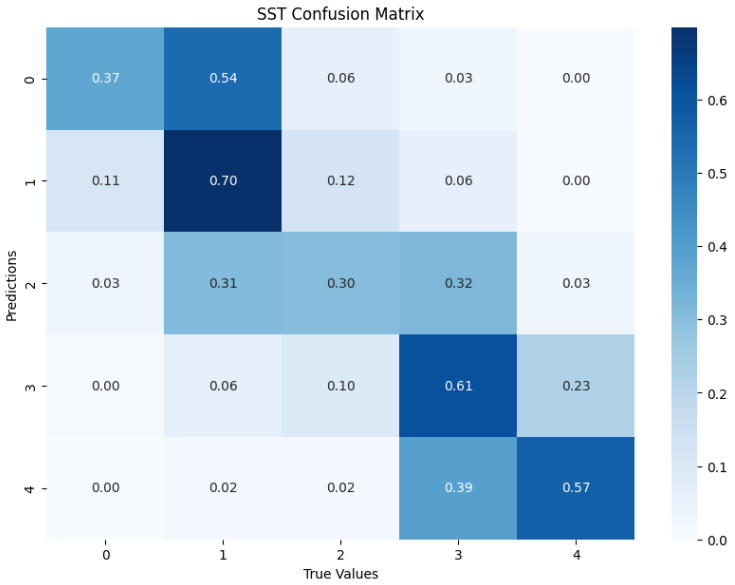

Figure 3: Sentiment Analysis Confusion Matrix

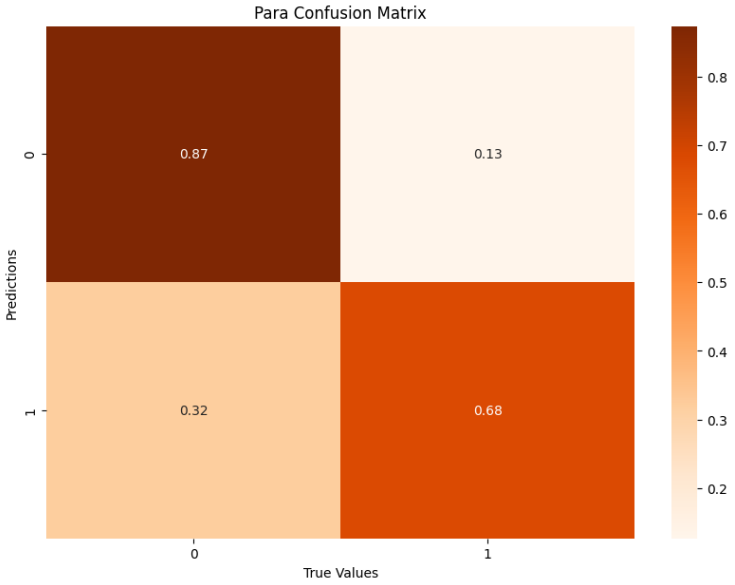

Figure 4: Paraphrase Detection Confusion Matrix

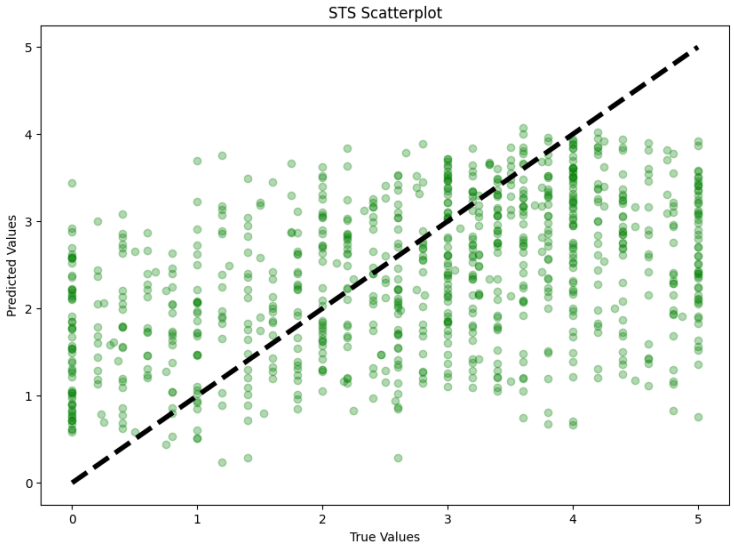

Figure 5: Semantic Textual Similarity Scatter Plot

7 Conclusion

Our approach greatly improved from our baseline accuracy in all three tasks, and we learned several aspects about creating a good model. Ensembles can help improve performance at the cost of compute time. Given the time constraints of this class project, we were limited in some of our techniques. One of the techniques, MBBP, we were implementing but we removed it in the final version because we had syntax issues. We would fully debug this technique with more time. If we had more time, we would also apply hyperparameter tuning on different seeds to find different versions of our best model to include in our ensemble. Currently, because of the time it takes to train, our models included in the final ensemble used approximated good parameters.

8 Team Contributions

Ivan Liongson implemented the MultitaskBERT forward function, preprocessed the SICK dataset, assembled the ensemble, performed the hyperparameter tuning, and trained the models on his powerful GPU. Erik Luna implemented the MultitaskBERT task functions, the Adam optimizer, SMART, and the data loading. Both of us we wrote the report and analysis.

References

Hui Bao, Kai He, Xuemeng Yin, Xuanyu Li, Xinrui Bao, Haichuan Zhang, Jialun Wu, and Zeyu Gao. 2021. Bert-based meta-learning approach with looking back for sentiment analysis of literary book reviews. In *Natural Language Processing and Chinese Computing: 10th CCF International Conference, NLPCC 2021, Qingdao, China, October 13–17, 2021, Proceedings, Part II*, page 235–247, Berlin, Heidelberg. Springer-Verlag.

Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Tuo Zhao. 2019. Smart: Robust and efficient fine-tuning for pre-trained natural language models through principled regularized optimization.

Shuaipeng Liu, Shuo Liu, and Lei Ren. 2019. Trust or suspect? an empirical ensemble framework for fake news classification. In *ACM International Conference on Web Search and Data Mining*, Online. ACM International Conference.

M. Marelli et al. 2014. A sick cure for the evaluation of compositional distributional semantic models. In *Proceedings of the [Conference Name]*, [Location]. [Publisher].

A Appendix (optional)