BERT on Multitask Training: Bimodality, Ensemble, Round-robin, Text-encoding, and More

Stanford CS224N Default Project

Jiaxiang Ma
Stanford University
`mjx@stanford.edu`

Yuchen Deng
Stanford University
`yuchen12@stanford.edu`

Abstract

In this project, we implement the key components of the Bidirectional Encoder Representations from Transformers (BERT) and exploit the strength of the pre-trained BERT model to improve three downstream tasks, sentiment analysis (SA), paraphrase detection (PD), and semantic textual similarity (STS). Specifically, we implement text pair encoding for PD and STS, and train the model in a bimodal fashion: 3 tasks on 1 shared BERT module with shuffled round-robin training, and 3 tasks on 3 dedicated BERT modules with sequential training. We also tune the hyperparameters and add SMART regularization for better performance. Finally, we put 6 best models from both architectures as an ensemble to make predictions on the 3 tasks. Our best model reaches an overall score of 0.797 on the dev dataset and 0.796 on the test dataset. As of the project deadline, we rank **first** and **second** on the test and dev leaderboard respectively.

1 Key Information to include

- Mentor: Arvind Mahankali
- External Collaborators (if you have any) or sharing project: no
- Contribution: Both implement minBERT independently. As for the extension, Jiaxiang implements architecture modification, text pair encoding, and hyperparameter tuning, and Yuchen implements round-robin training, SMART regularization, and ensemble. Both contribute equally to research, discussion, and writing.

2 Introduction

The Bidirectional Encoder Representations from Transformers (BERT) from Devlin et al. (2019) is regarded as a big milestone in the NLP development. As a large pretrained model, BERT can provide high quality sentence embeddings for a wide variety of NLP tasks. With little modification in architecture and further finetuning, BERT can be adapted to a specific task. However, in a multitask background, task-specific finetuning on a single BERT module may not be as effective because of different task requirements, conflicting gradients among tasks, etc.

In this project, we explore the different ways of improving multitask performance using BERT. We first implement a minimalist version of the BERT model, including the Transformer module in Vaswani et al. (2017), text embeddings in Devlin et al. (2019), and Adam optimizer in Kingma and Ba (2015). Then we implement a multitask classifier model using BERT and train our model on three downstream tasks: sentiment analysis (SA), paraphrase detection (PD), and semantic textual similarity (STS). SA classifies the polarity of a sentence. PD detects if two sentences are restatements of each other. STS measures the degree of similarity between two sentences. We attempt the following methods: replicating a BERT module for each task, encoding text in pairs as described in Devlin et al. (2019), applying a shuffled round-robin training procedure on three tasks, adding SMART regularization from Jiang et al. (2019), tuning hyperparameter, and putting together an ensemble to further improve the multitask classifier with BERT as the backbone. Our final model ranks **first** on the test dataset and **second** on the dev dataset as of the deadline of the project.

For the rest of the report, Section 3 is the related work that we get inspiration from for multitask training, Section 4 details all the approaches we adopt to improve multitask performance, and Section 5 and 6 include all quantitative and qualitative results of our training process. Section 7 concludes our project with future work.

3 Related Work

Devlin et al. (2019) first proposes Bidirectional Encoder Representations from Transformers (BERT), which has proven to be one of the earliest pretrained large language models that could be further finetuned for a variety of downstream NLP tasks. BERT can generate high quality sentence embeddings by training deep bidirectional representations based on multi-layer bidirectional Transformer encoder proposed by Vaswani et al. (2017). We implement the key components of BERT and build a minimalist implementation of the BERT model (minBERT) to load the pretrained parameters of BERT and integrate BERT into a multitask classifier finetuned for downstream tasks. Devlin et al. (2019) also describes a new way of encoding text pairs for downstream tasks such as paraphrasing and question answering, which is to concatenate the text pairs and then feed it into BERT as a single input so that the cross attention between two sentences can be better exploited. This encoding approach is not implemented in minBERT and we are inspired to adopt this approach to improve the performance of PD and STS.

When fine-tuning the pretrained model on the downstream tasks, due to the limited training data for the downstream tasks and the high complexity of pretrained model, the aggressive finetuning may cause the model to overfit the training data and not generalize well to unseen data. Jiang et al. (2019) proposes SMART framework that includes a Smoothness-inducing Adversarial Regularization and Bregman Proximal Point Optimization method to attain better generalization. The regularization penalizes the drastic change in the model output when a perturbation is applied on the input, and enforces the smoothness of the model. We integrate the SMART regularization in SA and STS training to improve the model generalization and performance.

4 Approach

4.1 Baseline

The baseline model consists of one shared BERT module connected to three task heads. The input of the baseline model is one sentence for SA or two sentences for PD and STS. For each input sentence, the BERT module produce a sentence embedding, which is the hidden state of the [CLS] token prepended to the token representation of the input sentence. The sentence embedding will then be fed into the task head. For SA, the classifier head includes a dropout layer and a linear layer with a 768×5 weight matrix. The 5 scalars represent the logits for 5 classes in SA. The prediction is computed by applying softmax to get the probability for each class and select the mostly likely one. We use cross entropy loss to minimize the probability of false classifications. For PD, the trainable head concatenates the two sentence embeddings after dropout and applies a linear layer with a 1536×1 weight matrix on the concatenated embedding to obtain a scalar. The prediction is then computed by passing the scalar to sigmoid function to get the probability. The binary cross entropy loss is used to minimize the probability for false prediction. For STS, we use the same linear layer with a 768×768 weight matrix to project the two sentence embeddings after dropout separately, compute cosine similarity of projected embeddings, and scale the score to [0,5] as prediction. The mean-squared-error(MSE) loss is used to minimize the error between predicted scores and real scores. We set a dropout probability of 0.3, a learning rate of 1e$-$5, a batch size of 32 and finetune the model on SA, STS, and PD sequentially. For each task, we train 10 epochs on the full model.

4.2 Multitask training

After we train the baseline model, we notice that multitask training tends to have conflict gradients among different tasks, so as the performance of the task being trained on improves, the other two tasks see a performance drop. Besides, the model suffers from severe forgetfulness due to sequential training, i.e. the model prefers the last tasks being trained. *To address these issues, we implement two strategies for multitask training: 1. train a 3-BERT model with 3 sub BERT modules for each downstream task sequentially, 2. train a 1-BERT model shared among three downstream tasks with a shuffled and sampled round-robin schedule.*

4.2.1 3-BERT architecture

We propose a new model architecture called 3-BERT, as opposed to the 1-BERT baseline model. 3-BERT architecture has 3 replicas of BERT module, each independently generating sentence embeddings for one task only. Therefore, when the 3-BERT model is finetuned on a specific task, only the BERT module assigned to the task and the task head will be updated and parameters dedicated to other tasks will not be negatively impacted. Since the 3 sub-networks are independent, we keep the same sequential training order as used in the baseline model.

4.2.2 Round-robin

When training the 1-BERT model with 3 downstream tasks, we implement a shuffled and sampled round-robin strategy. First, we notice that the dataset for PD is 33 times the size of the dataset for SA and 47 times the size of the dataset for STS. If we train every tasks for 10 epochs, the model will prefer PD because PD has the largest training data. However, if we just downsample the PD dataset, we will waste a lot of useful data for PD training. Therefore, we decide to sample the data to balance the multitask training. All the datasets are split into batches with a size of 32, and the task data size is defined as the max number of batches for this task, which equals *number of batches per epoch* $\times$ *number of epochs*. We train 10 epochs for SA and STS, and 3 epochs for PD. During each iteration, we sample a batch from three tasks with a probability proportional to the task data size and train the model in a round-robin schedule so that the model keeps learning from all downstream tasks. The training completes when all tasks reach their defined task data size.

4.3 Text pair encoding

Two of the downstream tasks, PD and STS, involve text pairs. The most intuitive way of encoding the text pairs is to independently encode the two texts, similar to Parikh et al. (2016). Specifically, we feed the two texts into BERT separately to get two text embeddings, and then feed the embeddings into the same neural network to compare the output and decide how similar the text pairs are or whether the text pairs are paraphrases. Despite that, Figure 1 shows another way of encoding the text pairs from Devlin et al. (2019). The text pairs are concatenated with two [SEP] tokens as delimiters and a [CLS] token at the beginning to represent the whole text embedding. The concatenated text is the only input to BERT, and the bidirectional cross attention between two texts can be fully exploited by this new text encoding. Furthermore, since the pretrained parameters of BERT are trained on the next sentence prediction task with the same text encoding approach, we decide to use the same text encoding for finetuning in order to make more use of the inter-sentence relationship learned from the pretrained tasks and stored in the loaded weights.

To adapt to the new encoding, the classifier heads for PD and STS have to change accordingly. The output sentence embedding will go through a linear layer with a 768×32 weight matrix and another linear layer with a 32×1 weight matrix to generate a scalar value for the two tasks respectively. There is one more step for STS, which is the sigmoid function is applied on the scalar value, and the output value is scaled to [0, 5]. The loss functions are the same as in the baseline.

We implement the text pair encoding with the tokenizer library in Mahankali (2024) to encode the sentence pair in the desired form and add the segment embeddings during training as in Figure 1.

4.4 SMART regularization

During training, we notice that the train accuracy keeps increasing, while the dev accuracy fluctuate in the later epochs, which indicates that the model may overfit on the training data. Therefore, we integrate the SMART regularization in SA and STS based on the results in Table 2. The SMART method adds a smoothness-inducing adversarial regularizer on the original loss, and solves the

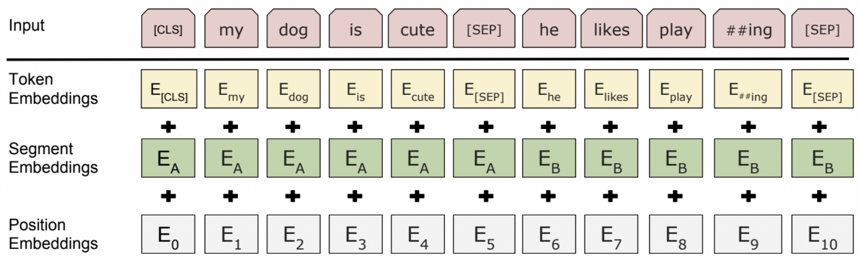


Figure 1: The BERT embedding layer from Devlin et al. (2019).

optimization for:

$$\min_{\theta} \mathcal{F}(\theta) = \mathcal{L}(\theta) + \lambda_{\mathrm{s}} \mathcal{R}_{\mathrm{s}}(\theta), \tag{1}$$

where λ_{s} is a tuning parameter as SMART loss weight, R_{s} is the smoothness-inducing adversarial regularizer, defined as:

$$\mathcal{R}_{\mathrm{s}}(\theta) = \frac{1}{n} \sum_{i=1}^{n} \max_{\|\widetilde{x_i} - x_i\|_p \leq \epsilon} \ell_s \left(f\left(\widetilde{x_i}; \theta\right), f\left(x_i; \theta\right)\right), \tag{2}$$

where ϵ is a tuning parameter as the perturbation threshold , x_i is the input embedding from the first embedding layer, and $\widetilde{x_i}$ is a perturbed embedding. For classification tasks, ℓ_s is chosen as the symmetrized KL-divergence; for regression tasks, ℓ_s is chosen as the squared loss. To solve the minimization problem, we use the Adam optimizer instead of Bregman Proximal Point Optimization mentioned in the paper.

We integrate the smart-pytorch library from Schneider (2022) for the implementation of SMART regularization in the training. For SA and PD, we compute a KL-divergence loss between the output predicted labels from the initial token embedding and perturbed token embedding, add it to the original loss, and use Adam optimizer to minimize it. For STS, we compute an MSE-loss between the predicted similarity scores from initial token embedding and perturbed embeddings, and add it to the original loss for minimization. For experiments, we use perturbation count = 1, $\lambda_{\mathrm{s}} = 1$ and ϵ=1e-5.

4.5 Hyperparameter tuning

We tune the hyperparameters during the training phase. Specifically, we tune the weight for SMART regularization, the dropout rate for each task, the learning rate, and the weight decay for the learning rate. After hyperparameter search, we find that only the dropout rate has a non-negligible affect on the performance based on our architecture setting. The best dropout rates are 0.2, 0.1, and 0.1 for SA, PD, and STS respectively, while the default dropout rate before tuning is 0.3 for all three tasks.

4.6 Ensemble

After we apply the approaches above, we get a collection of models of the 3-BERT architecture and the 1-BERT architecture. These models have comparative performance on PD, while some models do better on SA and the others on STS. Therefore, we ensemble these models to further improve the performance. Specifically, we select the mode of predictions from all models to generate the final prediction for SA and PD, and calculate the mean of predictions from all models to generate the final prediction for STS. We choose 3 3-BERT models and 3 1-BERT models with the best performances that differ only in the dropout rate or the randomization seed.

5 Experiments

5.1 Data

We use the following datasets from Mahankali (2024) for experiments:

1. Stanford Sentiment Treebank dataset, which consists of movie reviews annotated with sentiment scores. It has been divided into train (8,544 examples), dev (1,101 examples), and test (2,210 examples). Each movie review has a label of negative (0), somewhat negative (1), neutral (2), somewhat positive (3), or positive(4).

2. CFIMDB dataset, which consists of movie reviews annotated with binary sentiment label. It has been divided into train (1,701 examples), dev (245 examples), and test (488 examples). Each movie review has a binary label of negative (0) or positive (1).

3. Quora Dataset which consists of sentence-pairs with paraphrase labels. It has been divided into train (283,010 examples), dev (40,429 examples), test (80,859 examples). Each sentence pair has a binary label of whether different questions are paraphrases of each other.

4. SemEval STS Benchmark Dataset, consisting of sentence-pairs with similarity scores. It has been divided into train (6,040 examples), dev (863 examples), and test (1,725 examples). The sentence pairs are rated on a scale from 0 (not at all related) to 5 (same meaning) to reflect the similarity between the two sentences.

5.2 Evaluation method

For SA and PD tasks, we evaluate the prediction accuracy between true predictions and all predictions. For STS task, we evaluate the Pearson correlation coefficients between predicted similarity scores and true scores. The overall accuracy score is computed as $\frac{\text{Acc(SA)}+\text{Acc(PD)}+0.5\cdot\text{Corr(STS)}+0.5}{3}$.

5.3 Experimental details

In all the experiments, we load pre-trained weights in BERT layer and finetune the full model including BERT layers and classifier heads using a batch size = 32, and Adam optimizer with learning rate = 1e-5 and $(\beta_1, \beta_2) = (0.9, 0.999)$. For the baseline, we train the full model with three tasks sequentially each for 10 epochs with a dropout rate = 0.3. To study the effects of each individual training method or configuration on the performance, we run a set of single task training experiments, with different choices of text-pair encoding, whether turning on SMART regularization, and different dropout rates. Then, we train the full model with all the tasks combining the methods and configurations that improve the performance most. We train two types of models: 3-BERT and 1-BERT, as described in Section 4.2. The 3-BERT models are trained on SA, STS, and PD tasks sequentially, each for 10 epochs. The 1-BERT models are trained in the round-robin schedule, using max epochs of 10 epochs on SST and SemEval dataset, and 3 epochs on Quora dataset. By changing the seeds and dropout rates, we train 3 1-BERT and 3 3-BERT models. We ensemble these 3-BERT and 1-BERT models to generate the final predictions for 3 tasks. All 3-BERT models are trained on an NVIDIA GeForce RTX 4090 for 3 hours and all 1-BERT models are trained on an NVIDIA RTX A6000 for 6 hours.

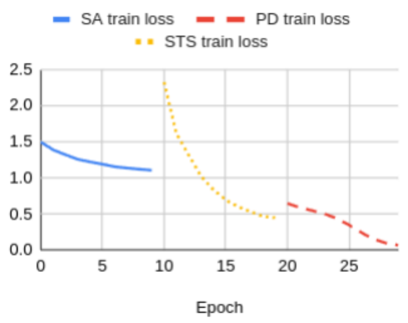
5.4 Results

5.4.1 Baseline

The training curves of the baseline model are shown in Figure 2, and the performance of the baseline is shown in Table 4. We can see that when we train the full model on three tasks sequentially, the improvement on one task leads to the degradation on the other two tasks. Therefore, we have two directions to resolve the conflicting gradients, i.e. training 3 BERT modules on 3 tasks respectively, and training 1 BERT module on 3 tasks in a round-robin fashion.

5.4.2 Text pair encoding

Table 1 shows the performance of using text pair encoding against encoding two texts independently in PD and STS on top of the baseline model. For tasks that involve two sentences, text pair encoding turns out to be effective.



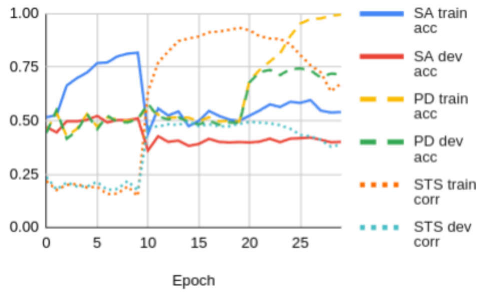


Figure 2: The training loss, train accuracy and dev accuracy trends for the baseline model.

Encoding Methods	PD	STS
Independent Text Encoding	0.721	0.487
Text Pair Encoding	**0.889**	**0.813**

Table 1: Performance of different encoding methods on PD and STS dev data.

5.4.3 SMART regularization

Table 2 shows the performance with and without SMART regularization when we train 1-BERT and the classifier head on each single task. PD is trained with $\frac{1}{20}$ of the Quora dataset in this ablation test. We find that SMART regularization improves SA and STS as opposed of PD.

SMART	SA	PD	STS
×	0.501	**0.836**	0.857
✓	**0.516**	0.813	**0.863**

Table 2: Performance of adding SMART regularization for each task.

5.4.4 Dropout rate

Table 3 shows the performance of single task training with different dropout rates and the same settings text pair encoding, and SMART regularization in 3-BERT architecture. We find out that a dropout rate of 0.3 is not always the best for different tasks under our setting.

Dropout Rate	SA	PD	STS
0.0	0.524	**0.908**	0.879
0.1	0.531	**0.908**	**0.880**
0.2	**0.537**	0.902	0.878
0.3	0.527	0.903	0.877

Table 3: Performance of different dropout rates.

5.4.5 Model comparison

Table 4 compares the models with different training methods and configurations on the dev dataset. We find that using text-pair encoding on PD and STS task, adding SMART regularization on SA and STS tasks, and using fine-tuned dropout rate is the best configuration.

Table 5 shows the performance of 6 individual models (3 3-BERT and 3 1-BERT) and their ensemble. The only difference among the 3 3-BERT models is the dropout rate used during training and randomization. The same difference applies to the 3 1-BERT models. For each task, the ensemble model is comparable or even better than the best individual model. For the overall score, the ensemble model exceeds all the individual model.

Table 6 shows the performance of our best ensemble model on the dev and test leaderboard. As of the project deadline, our model ranks second on the dev data and first on the test data.

Model	Overall Score	SA Accuracy	PD Accuracy	STS Correlation
Baseline	0.627	0.416	0.721	0.487
Baseline + TPE	0.717	0.356	0.889	0.813
3-BERT	0.686	**0.528**	0.813	0.432
3-BERT + TPE	0.783	0.514	0.903	0.862
3-BERT + TPE + SMART	0.788	0.527	0.903	0.871
3-BERT + TPE + SMART + DF	**0.790**	0.527	**0.907**	0.871
RR + TPE	0.784	0.513	0.905	0.868
RR + TPE + SMART	0.787	0.523	0.901	0.873
RR + TPE + SMART + DF	0.788	0.513	0.905	**0.893**

Table 4: The model comparison with different training methods and configurations. TPE = Text Pair Encoding, SMART = SMART Regularization, DF = Dropout Finetuning, RR = Round-robin.

Model	Overall Score	SA Accuracy	PD Accuracy	STS Correlation
3-BERT + TPE + SMART + DF (0)	0.792	**0.537**	0.903	0.873
3-BERT + TPE + SMART + DF (1)	0.790	0.527	0.907	0.871
3-BERT + TPE + SMART + DF (2)	0.789	0.533	0.901	0.868
RR + TPE + SMART + DF (0)	0.788	0.513	0.905	**0.893**
RR + TPE + SMART + DF (1)	0.787	0.514	0.903	0.885
RR + TPE + SMART + DF (2)	0.789	0.519	0.904	0.885
Ensemble	**0.797**	0.531	**0.915**	0.889

Table 5: Ensembling model from 3 3-BERT models and 3 1-BERT models.

Leaderboard	Overall Score	SA Accuracy	PD Accuracy	STS Correlation
Dev	0.797	0.531	0.915	0.889
Test	0.796	0.529	0.914	0.887

Table 6: The best results on the dev and test leaderboard.

6 Analysis

6.1 Text pair encoding

Text pair encoding brings the most evident improvement to PD and STS. The underlying principle is that we make full use of the pretraining procedure and the architecture of BERT. BERT is pretrained on next sentence prediction, which also involves two sentence pairs, so the loaded parameters of BERT learns the pattern of [CLS] + Sentence 1 + [SEP] + Sentence 2 + [SEP] well and the relationship between sentences is better extracted by cross sentence attention mechanism as opposed to feeding the two sentences separately into BERT.

6.2 3-BERT

Dedicating 3 BERT modules to 3 tasks respectively is a straightforward solution to conflicting gradients among tasks and thus can improve the performance as the results in Table 4. However, one BERT base module has 1.1B parameters, and 3 BERT modules triples the size of the model. The generalization capability of the model also gets worse, because each new task needs a new BERT module together with a new task head.

6.3 Round-robin

Using the shuffled round-robin training schedule, the model keeps learning from all downstream tasks in turn, which resolves the issue of forgetting the early tasks as in our baseline sequential setting. The sampling also enables us to handle datasets with varying sizes and make the training fair for each task by running different epochs based on the varying sizes. With this, we can utilize the complete data in the large dataset and not cycle the small datasets many times. However, since the 3 tasks are still sharing the same BERT module, the best 1-BERT model with round-robin schedule is still slightly worse than the best 3-BERT model, which completely avoids conflicting gradients in multitask training.

6.4 SMART regularization

Using SMART regularization effectively improves the performance on SA and STS. In our experiments, we observe that the dev score for SA task fluctuates between epochs, and this trend on SA is more obvious than on the other two tasks. Adding SMART loss alleviates the issue as SMART regularization smoothes the change in the model output from a perturbation in input embedding.

6.5 Qualitative evaluation

For SA and PD classification tasks, we compute the confusion matrices between the predicted labels and true labels in Figure 3. For SA, the sentiment class of "somewhat negative" and "somewhat positive" are well predicted with an accuracy score of 0.81 and 0.76. For other sentiment classes, "negative" tends to be predicted as "somewhat negative", "positive" tends to be predicted as "positive" or "somewhat positive", and "neutral" tends to be predicted as either "somewhat negative" or "somewhat positive". The false cases tend to predict a class with the neighboring class. The best SA score is much lower than best PD score, and we think it's related to finer granularity labels for SA. The PD labels are in two polarity, and PD is a binary classification task. The SA labels are more fine-grained, and SA classification requires the model to output the sentiment degree.

For STS, we generate a scatter plot from the true similarity scores and the predicted scores in Figure 4. The line in red is the perfect fit that exactly match the truth. The prediction results scatters around the true values, while they still exhibit a linear correlation in that most points are close to the red line.

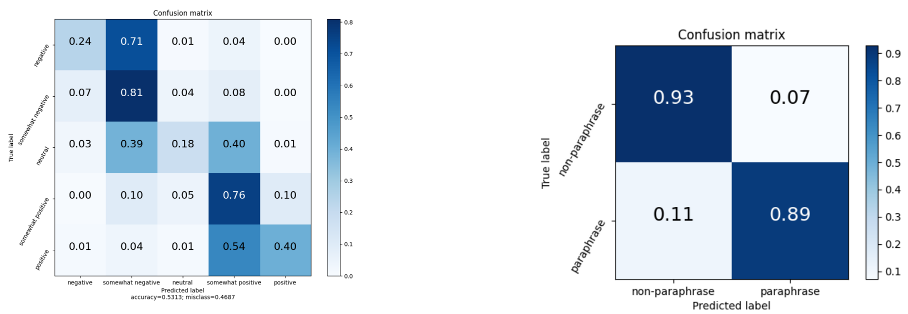


Figure 3: The confusion matrix for SA and PD. In both matrices, element M_{ij} represents for true label j, the percent of samples predicted as label j.

7 Conclusion

We explore different methods to improve the BERT model performance on multiple downstream tasks, including sentiment analysis (SA), paraphrase detection (PD), and semantic textual similarity (STS). We observe the gradient conflicts and forgetting issue when training three tasks sequentially, so we propose two other strategies for multitask training: the first is to combine three sub models each for a downstream task, the second is to train single model with multiple tasks in a round-robin schedule. For the sentence pair datasets, we find that text-pair encoding effectively captures the relationship between sentences using the strong representation power of the BERT module, and thus greatly improves PD and STS performance. The SMART regularization penalizes the model change from input perturbation, encourages the model smoothness, and brings effective improvements to SA and STS. We further tune the hyperparameters, including dropout rate, weight for SMART loss, and weight decay, and find out that dropout rate matters more in our setting. Finally, we ensemble the best models using the techniques mentioned above for our final predictions on the 3 tasks and get the best overall score of the test dataset on the leaderboard.

As for future work, we find that SA is hard in terms of distinguishing subtle differences between neighboring sentiment class. Having more training data for sentiment analysis may help improve

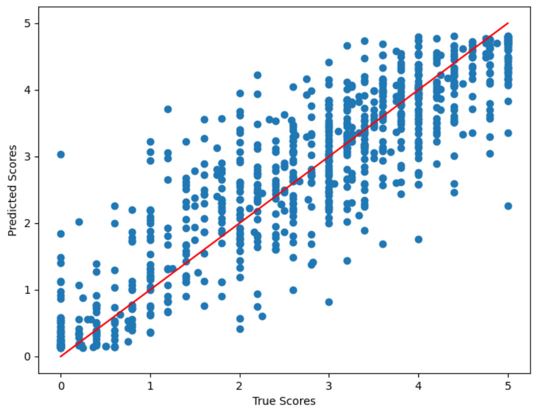


Figure 4: The scatter plot for true scores and predicted scores in STS.

the sentence embeddings and SA. There are also methods such as using mean pooling on token embeddings instead of [CLS] embedding as the sentence embedding. We will try these methods if time permits.

8 Ethics Statement

There are several ethical challenges and possible negative societal risks that we could think of in our project:

1. All the dataset that we use are in English, which means only English speakers can understand and exploit our model for the downstream tasks. However, the training data from the English speaking world may have some bias. For instance, for the movie review dataset, the reactions are mostly representing the people who speak English, and their attitudes may vary a lot from other countries who don't speak English a lot, such as China, France, Japan, etc. Also, if there are movie reviews in other languages, they might already be excluded from the dataset, and thus lead to a potential bias for sentiment analysis.

2. The dataset is mostly from sources like online movie review, news, online question-answering, and other Internet activities. The corpus is less representative of people are less exposed or interactive with those media. Specifically, the Quora dataset is collected from Quora users, who usually share personality of being interested in asking questions to learn new things, and answering questions to spread knowledge. The user profile may not include people who tend to look for solutions from books. The Quora user has to be more than 13 years old, so it automatically excludes children from the dataset. The uneven distribution of the user profile in personality and age in Quora dataset leads to a potential bias for paraphrase detection task.

We may mitigate the risks mentioned above in the following ways:

1. The training dataset can come from more languages. There are multiple ways to utilize other languages. One way is before we start training the model on the dataset, we can unify the language by translating the other languages to English in order to keep the data from people who speak other languages in the loop. Another way is to use the multilingual-BERT model that supports more than 100 languages to process the multilingual dataset.

2. Use more diverse dataset for task training, for example, we can add PAWS-Wiki (Paraphrase Adversaries from Word Scrambling) dataset in training paraphrase detection task.

References

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

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. In *Annual Meeting of the Association for Computational Linguistics*.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Arvind Mahankali. 2024. Cs224n-spring2024-dfp-student-handout: Starter code for default final project, spring 2024. `https://github.com/amahankali10/CS224N-Spring2024-DFP-Student-Handout`.

Ankur Parikh, Oscar Täckström, Dipanjan Das, and Jakob Uszkoreit. 2016. A decomposable attention model for natural language inference. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2249–2255, Austin, Texas. Association for Computational Linguistics.

Flavio Schneider. 2022. Smart pytorch. `https://github.com/archinetai/smart-pytorch`. Accessed: 2024-06-06.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.