BERT for Downstream Multitask: Effective Learning from Data with Paired Sentences

CS310 Default Project

Zhiyuan Zhong
Department of Computer Science
SUSTech
zhongzy2021@sustech.edu.cn

Haoxian Liu
Department of Computer Science
SUSTech
liuhx2021@sustech.edu.cn

Abstract

Transfer learning allows pre-trained language models, such as BERT, to be extended and applied to a variety of downstream tasks. In this project, we focus on generalizing our minBERT model to three downstream tasks: sentiment classification, paraphrase detection, and semantic textual similarity. We investigate optimization techniques including even batching, loss scaling, further pre-training, cross encoding, and task dropout. By properly leveraging training data and adopting encoding strategies, our model exhibits great improvement on multitask learning ability, achieving an average accuracy of 0.7637 on the dev set. All the code are open sourced in this github repository[1]

1 Key Information to include

- No external collaborators
- No external mentor
- Project is not shared across classes

2 Introduction

Pre-trained language models like BERT (Devlin et al., 2019) and GPT (Brown et al., 2020) have significantly changed the field of Natural Language Processing in recent years. These transformer-based model produces deep contextual embedding of text, which can be further used in downstream task by integrating various task heads.

In our project, we implement the basic minBERT model and investigate different methods to improve the model's multitasking performance on three downstream tasks via fine-tuning. In this report, we present our approaches to enhance model performance including efficient learning from data, and architectural modification. The details of these techniques are in the following section 3. For convenience, we named the three task as **SST**(sentiment classification), **paraphrase**(paraphrase detection), and **STS**(semantic textual similarity) in short.

3 Approach

3.1 Baseline

Our baseline is the minBERT model implemented on the skeleton code and the default project handout. The baseline fine-tunes the BERT model with the task heads described below, following the

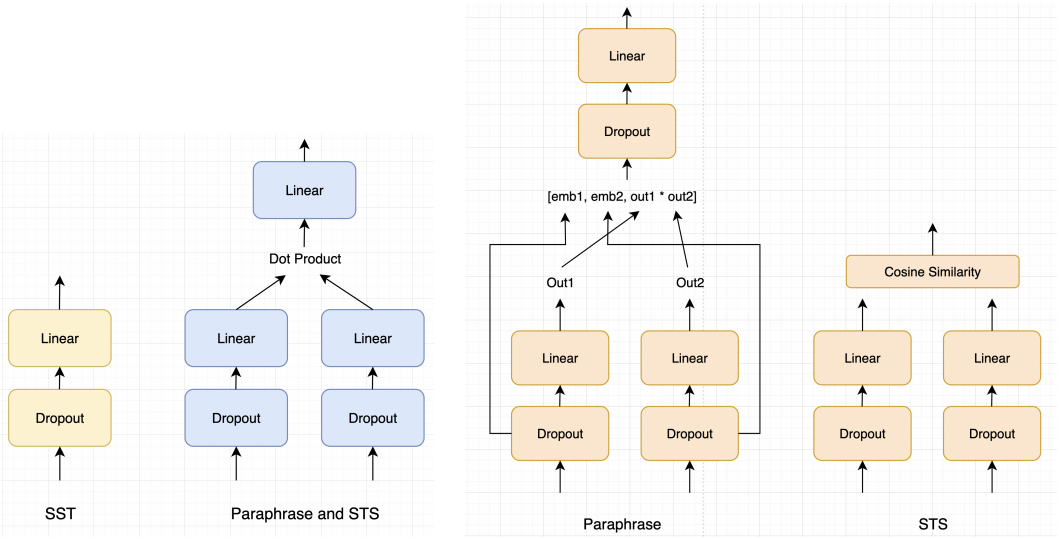
[1] https://github.com/Cooper-Zhong/CS310-project-minbert

architecture of Yuan Wang's github repository[2] (Wang, 2024b). We name this architecture as `B1`. All dropout layers mentioned have a dropout probability of 0.3, the learning rate for fine-tuning is 1e-5 and training is done for 10 epochs.

- SST: A dropout layer followed by a linear layer, trained with cross-entropy loss.
- Paraphrase: First a dropout layer. Each sentence embedding is fed into a linear layer to get an intermediate representation followed by a dot product, finally with another output linear layer. It is trained with L1 loss.
- STS: Similar as paraphrase.

We also compare another task head architecture design following Ramgopal Venkateswaran's github repository[3]. We name this architecture as `B2`, the task heads are described below:

- SST: Same as `B1`
- Paraphrase: Similar as `B1` with a dropout layer, followed by a linear layer to get an intermediate representation(**out1** and **out2** for 2 input sentences respectively). Then the dot product of these outputs are concatenated with the original embedding of sentence 1 and 2 after dropout(**emb1** and **emb2**), in the format of [**emb1**, **emb2**, **out1** * **out2**], which is finally passed to a final output linear layer.
- STS: Similar to Paraphrase, but the output is the cosine similarity of the intermediate embedding scaled by a factor of 5.0, trained with MSE loss (Venkateswaran, 2024).



(a) B1 Task Head (b) B2 Task Head

Figure 1: Baseline Task head

3.2 Even Batching

The original shuffling logic optimized each task's model weights one epoch at a time, in a round-robin manner. This could lead to the model converging towards the optimum for the current task's data epoch, and then changing direction sharply when the next task's data epoch was consumed (Venkateswaran, 2024). To address this, we can shuffle the data at the batch level rather than the epoch level. Specifically, the data for each task is divided into equal-sized batches, and these batches are then shuffled across all three tasks. This helps the optimization steps being more evenly distributed among the different tasks, rather than being concentrated on a single task's data for an entire epoch. We use part of the code from Venkateswaran's github repository[4].

[2] `https://github.com/yuanwang98/ywang09-minbert-default-final-project/`

[3] `https://github.com/ramvenkat98/methods-to-improve-downstream-generalization-of-minbert/blob/main/multitask_classifier.py`

3.3 Loss Scaling

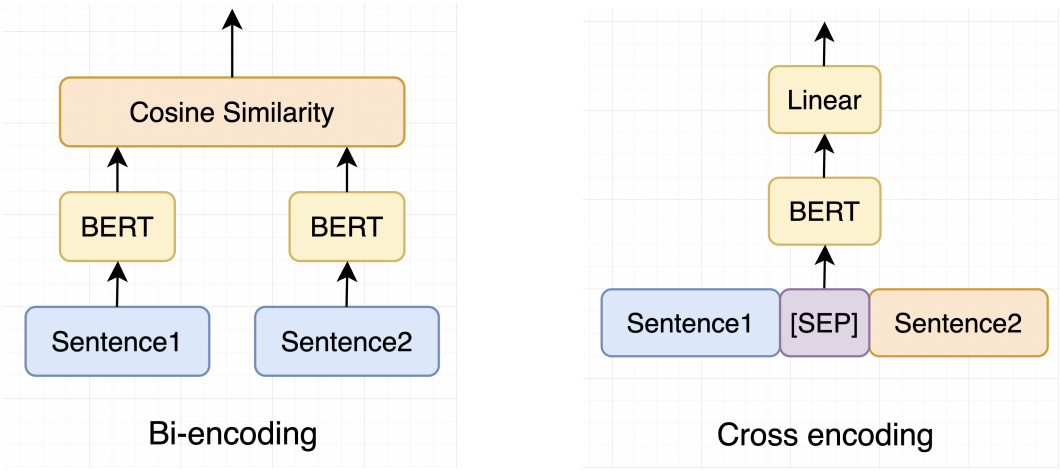
We observed that the training data sizes are highly imbalanced among three datasets, with the Quora dataset being significantly larger (210,000) compared to the SST (8,500) and STS (6,000) datasets. To counteract this imbalance, (Venkateswaran, 2024) proposed the approach to down-weight the Quora dataset by a factor. Specifically, the loss computed for the paraphrase task (Quora dataset) is multiplied by k, where $0 \leq k \leq 1$. This ensures that the model still sees all the Quora data during each epoch, but keeps its impact on the weights comparable to the smaller SST and STS datasets.

3.4 Further Pre-training

Since BERT is pre-trained on large corpus, we can further pre-trained the model on the task datasets to provide the model with extra unsupervised learning on the task-specific datasets(SST, STS, Quora), before proceeding to fine-tuning. The data is preprocessed by combining the sentences from all three task datasets and then shuffling[5]. This combined and shuffled data is used as the input for the further pre-training. The training code for this pre-training step was adapted directly from Venkateswaran's github repository.[6]

3.5 Cross Encoding

Semantic textual similarity and paraphrase detection are both sentence-pair tasks that require evaluating two input sequences to generate a prediction. We can either consider two sentences separately or treat them as a whole pair. Following Febie Jane Lin (2024), to handle these sentence-pair tasks, two strategies were explored:



(a) Bi-encoding (b) Cross encoding

Figure 2: Encoding strategies for pair input

1. Bi-encoding (**??**): Independently processing each input sequence through BERT, and then combines the embeddings to produce a prediction. For paraphrase detection, the embeddings are concatenated and a linear classifier is applied. For STS, the cosine similarity of the two embeddings is used to evaluate the degree of resemblance.
2. Cross-encoding (2b): Merging each input sequence with a special [SEP] token, and then passes the combined sequence through the BERT model. A linear classification or regression layer is then directly applied to the output embedding to produce the prediction.

[5] https://github.com/ramvenkat98/methods-to-improve-downstream-generalization-of-minbert/blob/main/prepare_data_for_additional_pretraining.py

[6] https://github.com/ramvenkat98/methods-to-improve-downstream-generalization-of-minbert/blob/main/additional_pretraining_data/run_mlm.py

The bi-encoding strategy is easier to implement, but the cross-encoding strategy has the potential to learn stronger inter-sentence relationships because it considers both sentences simultaneously during forwarding, allowing it to learn broader context and dependencies.

3.6 Task Head Dropout

We investigate the effects of dropout layers in the task heads. Dropout usually helps to prevent the model from overfitting to the individual tasks, forcing the model to learn more generalized representations.

4 Experiments

4.1 Data

For **Fine-grained sentiment classification**, we use the Stanford Sentiment Treebank (SST-5) dataset, which consists of sentences extracted from movie reviews, where each sentence has a label of negative, somewhat negative, neutral, somewhat positive, or positive. We have 8,544 `train` examples and 1,101 `dev` examples.

For **paraphrase detection**, we use the Quora Question Pairs (QQP) dataset, which consists of questions pairs with labels indicating whether they are paraphrases of one another. We have 141,498 `train` examples, and 20,212 `dev` examples.

For **semantic textual similarity**, we use the SemEval STS Benchmark dataset, which consists of sentence pairs of varying similarity on a scale of 0 (not at all related) to 5 (equivalent meaning). We have 6,040 `train` examples, and 863 `dev` examples.

4.2 Evaluation Method

We follow the task-specific metrics described in the project handout to evaluate our model. For SST-5 and QQP, we measure the model's performance using accuracy between the true and predicted labels. For STS, we use the Pearson correlation coefficient, which measures the linear correlation between the true and predicted similarity values.

4.3 Experiment Details

Unless otherwise noted, all models were trained with a learning rate of 1×10^{-5} with a hidden layer dropout probability of 0.3 for 10 epochs. All models used the AdamW optimizer with $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 1 \times 10^{-6}$, and no weight decay. All training were done using GPU, based on the pre-trained `bert-base-uncased` model from huggingface. The accuracy was calculated on the dev dataset.

4.4 Results

4.4.1 Vanilla Baseline

Following section 3.1, our first experiment investigated the performance of the vanilla minBERT model `B1` and `B2`.

Table 1: Performance of Vanilla Baseline

Model	SST	Para	STS	Avg.
B1	0.4913	0.7477	0.4456	**0.5615**
B2	0.4441	0.7884	0.6829	**0.6384**

Observed from table 1, `B2` model performed better than `B1` in sentence-pair tasks, especially STS task, demonstrating the modified Paraphrase task head with more compact representation [`emb1`, `emb2`, `out1` * `out2`] and directly applying cosine similarity score in STS task head can better learn the dependencies between two input sentences. However, these improvement were at the expense of

lower SST accuracy. This could be the result of model's overfitting on sentence-pair data, leaving the sentiment data under-learned.

In the following sections, the techniques we applied are based on the `B2` model, since it performed better than `B1`.

4.4.2 Even Batching and Loss Scaling

Following section 3.2 and 3.3, we shuffled and divided all the data of each task into equal-sized batches, and use batch samplers from Pytorch to sample batch from each dataset and interleaves them in a random ordering. We also multipy the loss of the Paraphrase batch by a factor k, default to 1.0. We adopted Venkateswaran (2024)'s code in our implementation.

Table 2: Performance with Even Batching and Loss Scaling

Note	Even Batching	Loss Scaling	SST	Para	STS	Avg.
B2	–	–	0.4441	0.7884	0.6829	**0.6384**
	✓	–	0.4850	0.8097	0.6955	**0.6634**
	✓	0.1	0.4922	0.7916	0.6727	**0.6522**
	✓	0.5	0.4950	0.7972	0.6870	**0.6597**

Looking at table 2, the accuracy of model with even batching increased by 2.5%, with improvement on all three tasks. Among three tasks the SST accuracy greatly improved, which could be the beneficial gain of preventing the model from specializing on the patterns in the Paraphrase data by even batching.

We also tried scaling loss of paraphrase batch by 0.1 and 0.5, which performed equally or slightly worse on average. This is contrary to the experiment of Venkateswaran (2024), we assumed that the complex task head architecture yielding the best performance from Venkateswaran (2024) is more sensitive to the reward from paraphrase dataset. Since our task head is much simpler, even batching already alleviated the imbalance issue, and thus scaling down the loss of paraphrase batch did not contribute much to overall training in our case.

Note: In the following sections, we will skip the analysis of loss scaling, because the effects of loss scaling is marginal and unstable along the experiments. We will put it as a reference aside.

4.4.3 Further Pre-training

As described in section 3.4, we first extracted the sentences from all three datasets, and follows Devlin et al. (2019) using Masked Language Modeling (MLM) with a mask out probability of 0.15, among which were 80% MASK, 10% random, 10% original. The model was trained for 3 epochs with a batch size of 64 and a learning rate of 5×10^{-5} using AdamW optimizer with $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 1 \times 10^{-8}$. Then we fine-tuned this model in the same manner as section 4.4.2.

Table 3: Performance with Further Pre-training

Loss Scaling	SST	Para	STS	Avg.
–	0.5122	0.7984	0.7608	**0.6905**
0.1	0.4822	0.8015	0.7915	**0.6918**
0.2	0.4895	0.8031	0.7841	**0.6922**
0.3	0.4941	0.8033	0.7798	**0.6924**

As seen in table 3, the average performance increased by 3% compared to table 2, with a huge improvement by 10% in terms of the STS accuracy. We hypothesized that this may be attributed to the stronger hidden representation of sentences of our model, who have already seen the sentences of the training data using MLM objective. Another explanation is that STS and Paraphrase are both sentence-pair tasks, the extra pre-trained model learned this semantic dependencies from the extensive paraphrase data better, and generalized well to the STS task.

4.4.4 Cross Encoding

Inspired by the resemblance between Paraphrase and STS task, we implement the cross-encoding technique from Febie Jane Lin (2024), as stated in section 3.5. The following models do not have task dropout layers. The results, as shown in table 4, solidifies the effectiveness of cross encoding, which significantly enhanced the average performance by 6% with roughly 8% in Paraphrase and 10% in STS.

Table 4: Performance of Cross Encoding

Loss Scaling	SST	Para	STS	Avg.
–	0.5050	0.8905	0.8789	**0.7581**
0.1	0.4850	0.8872	0.8802	**0.7508**
0.2	0.5013	0.8798	0.8789	**0.7533**

4.4.5 Task Head Dropout

Finally, we investigated the effects of dropout layers in the task heads. The following models were trained with a batch size of **16**. As indicated in table 5, the relatively high accuracy comes from models with task dropout layers. Besides, we discovered that increasing the batch size(from 8 to 16) also brought slight performance gain. However, these improvement is minor and marginal, we chose to keep the task dropout layers in our final model, since it usually helps model generalize better.

Table 5: Performance of Task Head Dropout – Batch Size 16

Loss Scaling	Task Dropout	SST	Para	STS	Avg.
–	✓	0.5077	**0.8901**	0.8841	**0.7606**
–	–	**0.5122**	0.8879	0.8797	0.7600
0.2	✓	0.4886	0.8866	0.8802	0.7545
0.2	–	0.4950	0.8865	0.8911	0.7575
0.3	✓	0.5013	0.8884	**0.8915**	**0.7604**
0.3	–	0.5022	0.8867	0.8871	0.7587

4.4.6 Final minBERT Model

We now present our final model's performance on the Dev set. It incorporates even batching, further pre-training, cross-encoding, and task head dropout. Note that it was trained with a batch size of **32** without loss scaling, based on our observation in section 4.4.5 and the end of section 4.4.2.

Table 6: Performance of Our Final Model – Batch Size 32

GPU	SST	Para	STS	Avg.
RTX 6000	0.5095	0.8908	**0.8836**	0.7613
V100	0.5195	0.8885	0.8821	0.7633
A100	**0.5277**	0.8885	0.8748	**0.7637**
RTX2080ti	0.5104	**0.8912**	0.8794	0.7603

Noticed from table 6, we trained the model under the same setting with different GPUs and produced different results. But training the model with one type of GPU twice yielded the same results. We guessed this to be the issue of different precision between GPU hardware and did not dive deeper into this question due to time constraint.

We also compared our final model with other models from Standford CS224N: Natural Language Processing with Deep Learning[7], illustrated in table 7. All the results were from their original reports. We also borrowed the result of individually fine-tuned baseline and frozen baseline from Wang

[7] `https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1244/project.html`

Table 7: Comparison with Baseline and Other Models on Dev sets

Model	SST	Para	STS	Avg.
Frozen Baseline	0.389	0.697	0.456	0.514
Fine-tuned Baseline	**0.543**	0.884	**0.886**	**0.771**
Venkateswaran (2024)	0.531	0.858	0.870	0.753
Febie Jane Lin (2024)	0.540	0.888	0.885	**0.771**
Wang (2024a)	0.509	0.883	0.884	0.759
Ours	0.528 (-0.015)	**0.889**	0.875 (-0.011)	0.764 (-0.007)

(2024a). Note that the fine-tuned baseline is actually three **separate** models which are fine-tuned on three datasets individually, and the frozen baseline only fine-tuned the task heads. Baseline from Wang (2024a) contains two feedforward layers with GELU as the activation function in the classification head. The size of the intermediate linear layer is chosen arbitrarily to be 256.

We present the comparison results in table 7. All the other best models mentioned in table 7 adopt either new loss function/regularization or complex model architecture. It turned out that our final model outperformed all others in paraphrase detection, with SST and STS performance being very close to other best results, only using simple but effective techniques.

5 Analysis

To achieve a comprehensive understanding of our model's performance, we conducted an in-depth qualitative analysis. This analysis explores the model's strengths and weaknesses across various tasks, examines potential prediction biases, and discusses instances where the model's predictions deviates the ground truth.

5.1 Sentiment Classification

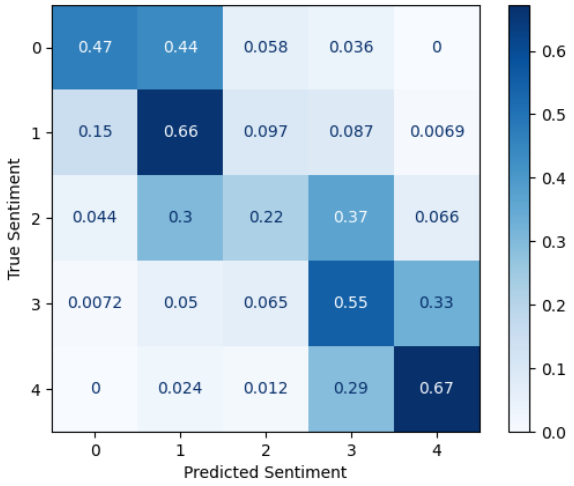


Figure 3: Normalized Confusion Matrix for SST-5

To analyze the performance, we generated a confusion matrix (Figure 3), which revealed that most predictions lie on the diagonal, indicating that our model has a strong ability to predict sentiment accurately. However, we observed that misclassifications often occurred between adjacent sentiment scores. For instance, a sentiment labeled as 3 might be predicted as 4, or vice versa. We hypothesized that this issue arises because the distinction between sentiments like 3 and 4 is inherently subjective and relies heavily on the annotators' judgment, similarly for 1 and 2. For example, the text *"It's*

fascinating to see how Bettany and McDowell play off each other." it labeled as 3 while the model's prediction is 4. The sentence is strongly positive but the label is somewhat positive, which is debatable. Besides, our model performed relatively poor in neutral sentiment classification, because the emotion boundary is often vague.

Furthermore, our model seems to exhibit a reliance on certain keywords, which may cause prediction biases. For example, the sentence *"No one goes unindicted here, which is probably for the best"* was ground-truthed as 2 (neutral), but our model predicted it as 3 (positive). Another instance is the sentence *"Half Submarine flick, Half Ghost Story, All in one criminally neglected film,"* which was labeled as 2 (neutral) but predicted as 1 (negative) by our model. The presence of words like *"best"*, *"Ghost"*, and *"criminally"* might have influenced the model's prediction, leading to misclassification.

On the other hand, we also identified cases where our model's predictions appeared more reliable than the ground truth. For example, the sentence *"Reign of Fire looks as if it was made without much thought – and is best watched that way"* was rated as 3 (positive) in the ground truth, but our model predicted it as 1 (negative). This suggests that in certain instances, our model may provide a more accurate sentiment assessment than the dataset annotations, indicating a potential higher reliability of our system's outputs.

5.2 Paraphrase Detection

By analyzing the results of paraphrase detection, we observed that our model will rely on the presence of identical words in both sentences. For example, the sentences *"Is there any chance of medal for India in Rio Olympics 2016?"* and *"How many medals is India expected to win at the 2016 Rio Olympics?"* are marked as non-duplicate in the ground truth, but our model predicted them as duplicates. We hypothesize that this is due to the significant overlap of key terms such as *"medal"*, *"India"*, and *"Rio Olympics 2016"*.

Conversely, we also found instances where our model correctly predicted sentences as non-duplicates, despite the ground truth labeling them as duplicates. For example, the sentences *"Which penny stocks are worth investing in India?"* and *"Are penny stocks good to invest in?"* were marked as duplicates in the ground truth, but our model identified them as non-duplicates. The first sentence emphasizes *"in India,"* while the second sentence lacks this keyword, leading to their classification as non-duplicates. This suggests that our system can detect nuances and discrepancies that annotators might have missed, indicating its potential to recognize overlooked distinctions.

5.3 Semantic Textual Similarity

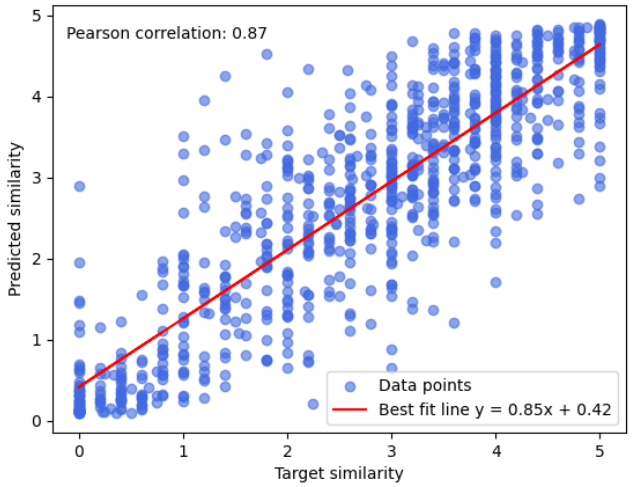


Figure 4: Scatter plot of predicted and target similarities for STS

In the Semantic Textual Similarity task, we plotted the actual values against the predicted values on a scatter plot and performed a linear regression fit(Figure 4). The resulting regression line is given by the equation $0.85x + 0.42$, and Pearson correlation coefficient is 0.87. These metrics indicate a strong correlation between our model's predictions and the actual results, demonstrating the model's robustness.

However, upon examining cases where there was a significant difference between the predicted and actual values, we observed a pattern similar to that found in the paraphrase detection task. Our model tends to rely on the presence of identical words within sentences. For instance, consider the sentences *"Use of force in defense of person."* and *"Use of force by aggressor."* Our model predicted a similarity score of 3.95, whereas the actual similarity score was 1.2, resulting in an overestimation by 2.75. This difference likely arises because both sentences are short and contain the key phrase *"Use of force,"* leading the model to overestimate their similarity without fully capturing the distinct meanings.

To address this issue, our future optimization efforts will focus on enhancing the model's ability to understand the semantic nuances of sentences, particularly those containing identical or similar words. This improvement aims to ensure that the model accurately assesses sentence similarity (and paraphrases) based on meaning rather than just lexical overlap.

6 Future work

Loss optimization and BERT architectural modification are not investigated in our work, such as Smoothness-Inducing Adversarial Regularization (SMART) from Jiang et al. (2020) and Projected Attention from Stickland and Murray (2019). We may further use model ensemble like majority voting as well.

7 Conclusion

In this project, we presented a multitask learning BERT model for downstream tasks. We have done detailed experiments and thorough analysis of our methods in the previous sections. Among our techniques, loss scaling in our experiments was more hit-or-miss, task head dropout could brought slight enhancement, and tuning hyperparameters like batch size might contribute to better performance. Other strategies inclusing even batching, further pre-training, and cross encoding can significantly improve model's multitask capability. The comparison with other best models further demonstrated our solid results.

References

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners.

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.

Jack P Le Febie Jane Lin. 2024. Bert and beyond: A study of multitask learning strategies for nlp. `https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1244/final-projects/FebieJaneLinJackPLe.pdf`. Accessed: May 27, 2024.

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.

Asa Cooper Stickland and Iain Murray. 2019. Bert and pals: Projected attention layers for efficient adaptation in multi-task learning.

Ramgopal Venkateswaran. 2024. Methods to improve downstream generalization of minbert. https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1244/final-projects/RamgopalVenkateswaran.pdf. Accessed: May 27, 2024.

Lainey Wang. 2024a. Pals for pals: Exploring extensions to projected attention layers for sentence-level tasks. https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1234/final-reports/final-report-169739344.pdf. Accessed: May 27, 2024.

Yuan Wang. 2024b. ywang09 minbert default final project. https://github.com/yuanwang98/ywang09-minbert-default-final-project. Accessed: May 27, 2024.