Investigation of BERT Fine-tuning Strategies

Stanford CS224N Default Project

Sheena Lai
Department of Computer Science
Stanford University
`sheenal@stanford.edu`

Abstract

This paper aims to investigate how the minBERT model multitask capabilities can be improved by various architectural changes, loss functions, regularizations, and other learning techniques. The investigation of these difference fine-tuning strategies gives insight on the nature of three focused downstream tasks: sentiment analysis, paraphrase detection, and semantic textual similarity. This paper explores how simultaneously fine-tuning these three tasks interact with the learning of the multitask minBERT model. I found that cross-encoding sentence pair tasks, along with a shuffled batch order, was a sturdy basis for my best model.

1 Introduction

The development of pre-trained language models, such as BERT has enabled a powerful representation of language that can be applied to many different language tasks. To harness the power of encoded embeddings that BERT provides, effective fine-tuning must be applied to downstream tasks, and furthermore, to train a multitask model that can simultaneously fine-tune towards different downstream tasks.

In this paper, I present my investigation of different fine-tuning strategies to improve the performance of the minBERT model on 3 downstream tasks: sentiment classification (SST), paraphrase detection (PD), and semantic textual similarity (STS). I explore and composite various strategies that have been developed since the introduction of minBERT that can be grouped into 4 categories: task head architecture, regularization, loss functions, and batch ordering.

2 Related Work

Task Head Architecture Sentence pair tasks, such as paraphrase detection and semantic textual similarity, require further considerations on how to decipher the relationship between two sentences. Devlin et al. (2018) proposed cross-encoding, where the pair of sentences are concatenated with two separation tokens, "[SEP]" (one in between the two sentences and one at the end of the sequence), and to feed as input into the BERT model.[2] The pair of sentences would then be encoded as a singular representative embedding to fine-tune downstream. Reimers and Gurevych (2019) later proposed to feed each individual sentence of the pair into BERT to get 2 separate embeddings.[5] Then, using a Siamese network, they introduced two ways to combine the sentence embeddings into a logit. The first method is concatenating the embeddings to each other, along with their difference. The second method is to evaluate the cosine similarity of the embeddings. I will investigate all three task head architectures in this paper.

Regularization Overfitting is common during aggressive fine-tuning, especially when fine-tuning generalized embeddings to optimize for specific tasks. One of the ways that Jiang et al. (2019) proposed to combat overfitting is SMART regularization, which seeks to smooth out the decision boundaries and preventing overly-complex models by combining Smoothness-Inducing Adversarial Regularization with Bregman Proximal Point Optimization.[4]

Loss Functions Different loss functions that have been researched to optimize towards specific sentence pair tasks. Henderson et al. propose Multiple Negative Ranked Loss for sentence pair tasks, which is a type of contrastive loss that tries to minimize the distance between positive pairs and maximize the distance between negative pairs in a dataset.[3] Additionally, Devlin et al. (2018) found that Cosine Embedding Loss fits neatly towards the STS task, by maximizing the cosine similarity of sentence pairs that are labeled as similar and minimizing cosine similarity of sentence pairs that are labeled as dissimilar.[2]

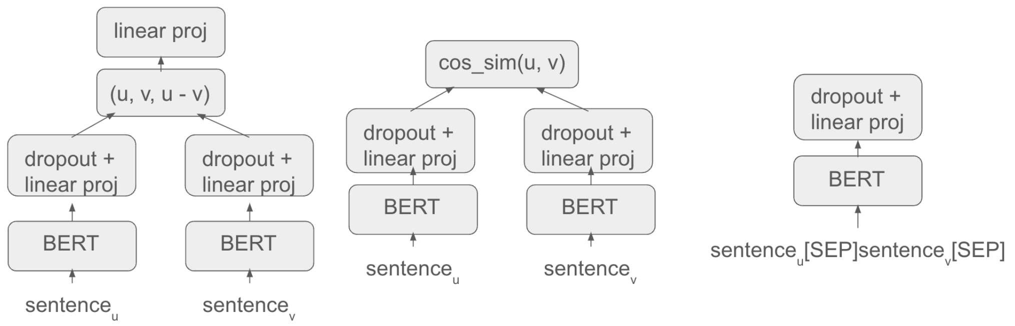
3 Approach

Baseline The baseline model contains the initial exploration of the minBERT fine-tuning methods. It uses a naive batching strategy of fine-tuning one batched pass through the SST dataset, then the PD dataset, and finally the STS dataset for each epoch. The SST task finetunes the input embedding with a single linear projection. The PD task uses the concatenation of each of the sentence embedding (after projected through a linear layer) along with their difference. The concatenated embeddings are then passed through a final linear layer. The STS task finds the cosine similarity between the two embeddings of the sentence pair (after they have been passed through their respective linear layers). All tasks use cross entropy loss as a most naive loss function baseline.

Fine-tuning Investigation Approach To individually isolate optimizing fine-tuning towards each task, I fine-tuned minBERT on a single task. I took the best performing task architectures, loss function, and regularization combinations from each single-task training to use to finetune their respective taskss in the combined multitask model.

Task-head Architectures for Sentence Pair Tasks (S-CD, S-CS, CE) I will be exploring 3 different architectures of combining the two sentences of the sentence pair: the S-BERT strategies of Siamese networks, introduced by Reimers and Gurevych (2019), and the original BERT cross-encoding strategy, introduced by Devlin et al. (2018).

The original cross-encoding strategy has the potential to characterize the two sentences better with the self-attention learning the relationship between the each word with each other in the input sequence. However, cross-encoding is much less efficient than the Siamese networks because it has to create embeddings for $n + m + 2$ tokens at once (for an n length sentence 1, m length sentence 2, and 2 separation tokens). This is especially relevant in terms of efficiency because the self-attention component of BERT grows quadratically with the length of the input sequence. On the other hand, the Siamese networks learn the embeddings of the two sentences separate and use the task head to extrapolate differences or similarities between the two embeddings, which is less robust than if the two sentences were compared with each other within the BERT architecture.



1. Siamese with concatenated difference (S-CD) 2. Siamese with cosine similarity (S-CS) 3. Cross-encoding (CE)

Figure 1: Different Task Head Architectures

Batch Ordering The baseline finetunes on all batches of each dataset at a time in sequential order for each epoch. However, I later found that this naive strategy does not pair well with multitask learning because for each epoch, the updates travel aggressively towards a single dataset at a time, which leads to sporadic and unstable learning. A less aggressive way to update towards each task is shuffling the all batches of the 3 tasks, and using the shuffled batch order to learn and spread each task's fine-tuning more sparsely with the other tasks. I further elaborate on this observation in section 5.

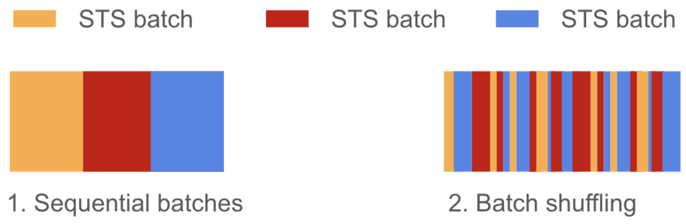


Figure 2: Batch composition per epoch in batch ordering strategies

Overfitting Procedures I performed a small ablation study of 5 epochs with each of the tasks (single-task fine-tuning) with dropouts of 0.1, 0.3, and 0.5. I found that the dropout rate of 0.1 was the most consistent and generally performed the best for all three tasks. The 0.1 dropout rate is used for all the listed results.

A commonly used form of regularization to prevent overfitting is L2 norm, where the L2 norm of the parameter weights are added to the loss to prevent the magnitude of any weight from growing too large.

$$\min_{\theta} \mathcal{F}(\theta) = \mathcal{L}(\theta) + \lambda ||\theta||_2$$

Additionally, Jiang et al. (2020) proposed a SMART regularization technique, which combines Smoothness Inducing Adversarial Regularization and Bregman Proximal Optimization, to improve penalties on aggressive updates and enforce smoothness.[4]

$$\min_{\theta} \mathcal{F}(\theta) = \mathcal{L}(\theta) + \lambda_s \mathcal{R}_s(\theta),$$

where $\mathcal{R}(\theta)$ is the Smoothness-Inducing Adversarial Regularizer:

$$\mathcal{R}(\theta) = \frac{1}{n} \Sigma_{i=1}^{n} \max_{||\tilde{x}_i - x_i||_p \leq \epsilon} l_s(f(\tilde{x}_i; \theta), f(x_i, \theta))$$

$\mathcal{L}(\theta)$ is the task's original loss function, λ_s is the scale of the regularization, and l_s is a chosen regularization loss, which Jiang et al. recommend to be KL-divergence for classification tasks and mean-squared error for regression tasks.[4]. This loss function analyzes the changes between the model's output from a real input and the output from a perturbed version of the real input as a regularization term to minimize upon

I used Jiang et al.'s implementation of SMART regularization in PyTorch to adapt to my model.[1] SMART regularization with $\sigma = 1e-5$ (noise variance), $\epsilon = 1e-6$ (noise norm), $\eta = 1e-3$ (step size), and $T_{\tilde{x}} = 1$ (iterations for updating noised inputs) was used for fine-tuning tasks that exhibited overfitting. Depending on the magnitude of losses causes by each individual task, the λ_s value was varied and selected through a small ablation study, starting from 1.

Loss Functions One of the loss function I explore is Multiple Negative Ranked Loss (MNR), which most fits the paraphrase detection task.[3] Let (a_i, b_i) be some positive pair in the dataset, where a_i is an anchor and b_i is its positive. MNR tries to minimize the distance between a_i and b_i while maximizing the distance between a_i and the positives of all other anchors. Its objective function is defined as

$$\mathcal{J}(x, y, \theta) = -\frac{1}{K} \Sigma_{i=1}^{K} [S(x_i, y_i) - log \Sigma_{j=1}^{K} e^{S(x_i, y_j)}],$$

where K is the size of the batch of positive pairs, S is the scoring function of the distance between an anchor and a positive. Another loss function, which Devlin et al. (2018) found successful and that could neatly apply to the paraphrase detection task and the semantic similarity task, is Cosine Embedding Loss.[2] This loss maximizes cosine similarity for labels with value 1 and minimizes the cosine similarity for labels with value 0.

$$\mathcal{L}(x, y) = \begin{cases} 1 - cos(x_1, x_2) & y == 1 \\ max(0, cos(x_1, x_2) - margin) & y == -1 \end{cases}$$

Weight Sharing Weight sharing between the three task heads can encourage further generalizable learning that could benefit the fine-tuning of all three tasks. I investigate the benefits of weight sharing by adding a linear layer and ReLU function right after the BERT embeddings are retrieved in each task's forward function.

4 Experiments

4.1 Data

For the sentiment analysis task, I used the Stanford Sentiment Treebank dataset (SST-5) (11,855 sentences). This dataset contains text strings as input and are labeled with a number from 1 to 5, representing from negative or positive the string is.

For the paraphrase detection task, the Quora dataset (QQP) was used (404,298 question pairs). The Quora dataset provides pairs of sentences, labeled with 0 or 1 on whether the pair of sentences are a paraphrase of each other.

For the semantic textual similarity task, the SemEval STS Benchmark dataset was used (8,628 sentence pairs). The STS dataset provides pairs of sentences, labeled with a rating in the range of 0 to 5 on how similar the sentences are to each other, with 0 being unrelated sentences and 5 having equivalent meanings.

4.2 Evaluation method

Provided by the default project guidelines, the evaluation metric used for the sentiment analysis and paraphrase detection tasks is accuracy. Pearson correlation between the actual and predicted textual similarity ratings was used for the sentence similarity task, as a non-discrete method of evaluation. Additionally, I used a composite scoring metric (provided by the default project leaderboard) as an overall metric of the multitask models: $S = ((sst\_acc + \frac{sts\_corr+1}{2} + paraphrase\_acc)/3)$.

4.3 Experimental details

I ran the model with a batch size of 64, a learning rate of 1e-5, and for 15 epochs (qualitatively was enough to see the loss flatten out), with an AdamW optimizer.

4.4 Results

The sentiment analysis individual task fine-tuning has a simple structure due to having an input of only one-sentence. Using only cross entropy loss was able to obtain an accuracy well above random assignment. However, the training accuracy grew to be well above the dev accuracy, prompting me to investigate improvements through regularization.

Model	Sentiment Analysis Dev Accuracy
cross entropy loss	0.512
cross entropy loss + SMART($\lambda = 3$)	**0.527**
cross entropy loss + L2	0.52

Table 1: Sentiment Analysis Accuracy for Single-Task Fine-tuning

For both paraphrase detection and semantic textual similarity, cross-encoding and Siamese networks (cosine similarity and concatenated difference) were tested. I also tested cosine embedding loss on the Siamese networks that evaluated on cosine similarity, and I tested multiple negative ranked loss on the paraphrase detection task. Interestingly, the multiple negative ranked loss did not perform as well, possibly because it requires positive pairs to be selected from the dataset to be evaluated in the loss function, neglecting the negative pairs, which is a significant portion of the data. For the paraphrase detection task, I also saw signs of over-fitting on the cross-embedding architecture, so I tested the effects of additional regulation.

Model	Paraphrase Acc
S-CD + BCE	0.740
S-CD + MNR	0.604
C-CS + BCE	0.732
C-CS + cos emb loss	0.707
CE + BCE	0.827
CE + BCE + SMART (λ = 0.5)	0.838
CE + BCE + L2	**0.843**

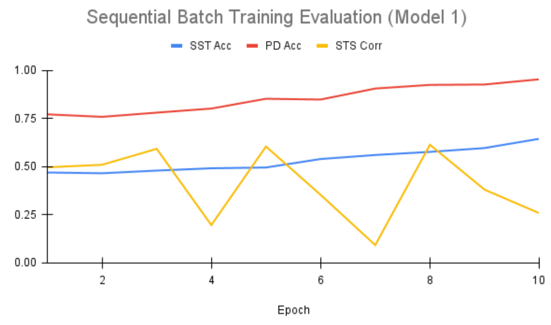
Table 2: Paraphrase Detection Accuracy for Single-Task Fine-tuning

Model	STS Corr
concatted diff + MSE	0.132
cos sim + MSE	0.337
cos sim + cos embed loss	0.261
cross-encode + MSE	**0.564**

Table 3: STS Pearson Correlation for Single-Task Fine-tuning

Interestingly, cross-encoding performed much better than both of the Siamese network architectures for paraphrase detection and STS tasks.

Finally, I combined the best model configurations from each individual task fine-tune for the task heads in the combined multitask model. The naive batch order resulted in a more sporadic learning of the STS task, compared to the smoother learning of the shuffled counterpart.



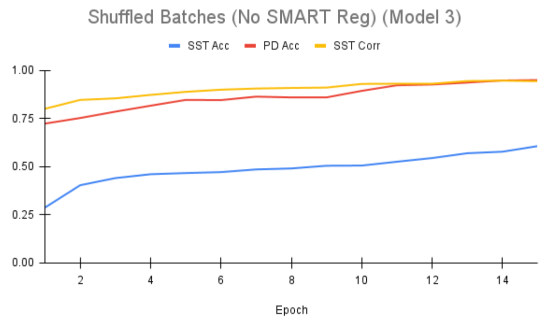


Figure 3: Training Metrics for Different Batch Ordering Techniques

While testing the multi-task model with SMART regularization on the SST task (which was found to perform the best in the single-task SST fine-tune), I observed that both the training and dev accuracies seemed to plateau at a lower accuracy compared to single-task learning, so I tested removing the SMART regularization to unhinder any further learning that might have been stifled by the smoothing factor.

After observing that the unregulated fine-tuning of the SST task performed better than the regulated fine-tuning in the context of multitasking, I also investigated removing the regulation for the paraphrase detection task, which shows signs of more overfitting than the SST task. I also tried allocating a heavier weight to the random selection of an SST batch in the batch order curriculum to encourage more training of the SST batch, but the further training also showed signs of overfitting.

Finally, I investigated how parameter sharing between all three task heads could improve the overall performance of multi-task model. I found slight improvement from the parameter sharing in the overall score of the model, making it the best model of my in investigation.

	Model	STS Acc	PD Acc	STS Corr	Score
0	Naive baseline	0.469	0.518	0.044	0.503
1	Sequential batches (terminated early -> 10 epochs)	0.385	0.392	0.679	0.539
2	Shuffled batches + SST reg'd	0.43	**0.834**	0.859	0.731
3	Shuffled batches + SST unreg'd	0.483	0.824	**0.864**	0.746
4	Shuffled batches + SST unreg'd + PD unreg'd	0.455	0.827	0.861	0.737
5	Shuffled batches + SST unreg'd + SST extra training	0.478	0.831	0.861	0.746
6	Shuffled batches + SST unreg'd + shared linear	**0.484**	0.828	0.857	**0.747**

Table 4: Multitask Model Dev Results

5 Analysis

STS performs better in multitask than individual fine-tuning The STS task performs much better in the multi-task fine-tuned model than in single-task fine-tuning, compared the paraphrase detection and sentiment analysis. I hypothesize that this is due to the paraphrase detection task being quite similar to the STS task (both have to find some sort of commonality or dissimilarity between two sentences), which leads the BERT parameters to optimize towards STS while optimizing towards the paraphrase detection task. The paraphrase detection task is so large that it also contributes to significant improvements of fine-tuning to the STS task, which has a much smaller dataset to learn from. The paraphrase task comparatively does not benefit as much from the smaller addition of the STS task in fine-tuning the BERT parameters, and the sentiment analysis task is much different in objective.

SST performs worse in multitask While the paraphrase detection performance stays about the same and STS performance significantly improves from single-task to multi-task fine-tuning. I hypothesize this is the case because the sentiment analysis problem is "semantically" more different to the paraphrase detection and STS tasks. The stronger weight on the gradient towards the PD/STS direction may conflict with the STS gradient direction, preventing it from optimizing as much as it was able to in the single-task fine-tune.

Confusion Matrices First, I analyzed the confusion matrix of the sentiment analysis task.

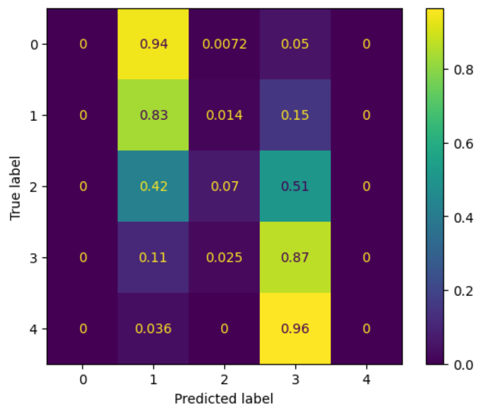


Figure 4: Confusion Matrix for Sentiment Analysis

The confusion matrix for sentiment analysis seems to show that most of the predictions for the dev set are ratings of 1 and 3. The model didn't even predict a rating of 0 nor 4 a single time. This finding seems to align with the frequency of each rating in the SST train dataset: 1092 ratings of 0, 2218 ratings of 1, 1624 ratings of 2, 2322 ratings of 3, and 1288 ratings of 4. The increased frequency of 1 and 3 ratings suggests that the model might have a tendency to have some base probability that 1 or 3 might occur over the other ratings. For a very subjective task, like sentiment analysis, the difference between a 0 and 1 or between a 3 and 4 could be more subtle. For example, an except that is rated as a 1 is "Pray doesn't have a passion for the material," and an except that is rated as a 0 is "It 's not a motion picture ; it 's an utterly static picture." The difference in sentiment is subtle to distinguish, even for a human. Some further work that could be done to improve on this observation is to even out the dataset (adding more 0, 2, and 4 rating data points, or possibly even truncating the 1 and 3 data points to have the same frequency as the other ratings). Another strategy is to use a loss function that weighs the error on 0, 2, 4 ratings slightly higher than the other ratings, scaled by how often they occur in the dataset.

The confusion matrix for paraphrase detection does not show any major concerns regarding the model's tendencies.

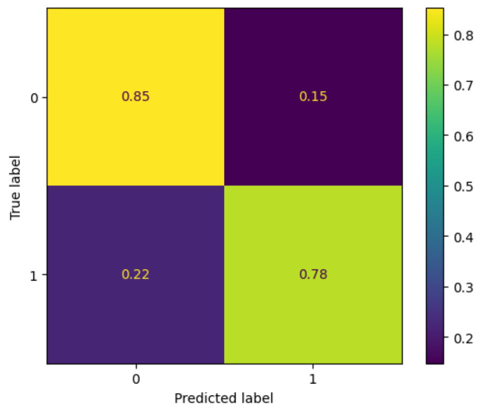


Figure 5: Confusion Matrix for Paraphrase Detection

However, there are noticeable trends in the model's false positives and false negatives, in terms of sentence content. The model seemed to predict very similar sentences as positive, even if one word might be enough to change the meaning of the question. For example, false positive that the model predicted is the sentence pair "Is atheism growing?" and "Why is atheism growing?". The sentence tokens themselves are quite similar, but the addition of the "why" token changes the meaning of the sentence itself. Additionally, the model often misinterpreted sentence pairs with high difference in keywords as not paraphrase. For example, "What is the best way to get a free iPhone?" and "Way to get a free iPhone 7?" is one of the false negatives from the dev set. The keywords in the sentence are quite different, but the essence of the question is the same, which is seeking how to get a free iPhone. While keyword similarity is an important heuristic, the semantics of the sentence itself also contributes greatly to cases that do not follow the heuristic. More examples that defy this heuristic could improve the performance of the model in these edge cases, or even adjusting the self-attention functionality for more dynamic semantic similarity detection.

Lastly, I saw similar trends in the semantic similarity predictions to the paraphrase detection task. For example, a pair of sentences that have a high rating of 4.0 but got a low prediction rating of 0.44 is "I think there isn't a general answer" and "I don't think there is a single definition." Similar to paraphrase detection, the sentences do not have that many similar tokens, but the semantics are similar. On the opposite end, a pair of sentences that have a low rating of 0.0 but got a medium prediction of 2.8 is "Work into it slowly" and "It seems to work." The tokens themselves seen to align between the two sentences, but the semantics are very different. Remedies for this are similar to the ones I proposed for the paraphrase detection class. Additionally, more examples that are more metaphorical or hyperbolic, should be included in the datasets, such as "work into it slowly," which contains a more non-traditional usage of the token "work."

6 Conclusion

The success of my findings lies in how the larger architectural changes of the multi-task fine-tuning structure and learning strategies can significantly impact how well the model is able to fit to towards specific tasks and descend along a multi-faceted gradient. I found that using cross-embedding for sentence pair tasks with a shuffled batch ordering is most effective. Additionally, some subtle fine-tuning strategies that can slightly improve a model's single-task fine-tune performance may not translate to the multi-task context, such as the additional unnecessary SMART regulation for the sentiment analysis tasks. The additional learning of the other tasks acts almost as an addition regulatory factor to disallow the minBERT parameters from overfitting to one specific task. As much as the architectural considerations affect the performance of a model, the quality of the datasets also matter. The common trends that I analyzed qualitatively in section 5 show that underrepresented data points were the points of errors in the model, such as pairs of sentences with differing meanings but similar tokens. Some future work for this is observing how the proposed model's performance changes when the distributions of datapoints in the train set (e.g. equal amount of each rating in sentiment classification) can change the accuracy of the model. Finally, I found that the semantic goals of each task in the multi-task fine-tune matter in how much the individual losses of the tasks may conflict with each other, as seen with the decrease in performance of the sentiment analysis task

and increase in performance in semantic similarity from the single-task to multi-task fine-tuning. The next step in my investigation, in a architectural sense, was planned to be using gradient surgery to improve the learning of the three tasks, specifically for sentiment analysis, which performs the most different from its single-task fine-tune counterpart and may have the most contrast in gradient from the other two tasks. However, due to memory and time restrictions, I was not able to run and debug gradient surgery fully. A future avenue for my work could be investigating the effects of gradient surgery on learning.
In a broader sense, I would want to investigate how commonality between downstream tasks affects the performance of the model. In this paper, the paraphrase detection and semantic similarity tasks are more similar to each other than sentiment analysis, but fine-tuning towards another task that is more semantic-related, such as stance detection (detecting whether the perspective of a text is for, against or neutral about a topic), could be more beneficial in multi-task training with sentiment analysis.

7 Additional Information

I had no mentor and no external collaborators. All portions of extensions are coded by me, except for SMART regularization.[1]

8 Ethics Statement

Some ethical concerns with the this project are the privacy issues related to the collection of information with these utilized datasets. For example, the CFIMDB and Quora datasets rely on data points generated from users on IMDB and Quora, respectively. The users of these websites may not have been made aware that their data is used in these publicly available datasets. Additionally, malicious analysis of specific data points in the dataset may have the potential to trace back to personal information about the user, which the user might not be aware of. Another potential issue is that the BERT model fine-tunes towards labelled data as the truth. However, the actual labels might not actually be truly accurate, especially with language being a relatively subjective experience. For example, the SST dataset labels the review "Paul Bettany playing Malcolm McDowell?" with a 2 (slightly negative). The author of the review may be a Paul Bettany fan or anti-fan, but it is subjectively hard to tell. These possible misinterpretations of the labels might percolate into the model itself, when the true metric of accuracy for the SST task may not reflect the model's actual accuracy to the subjective problem of sentiment analysis.

This project aims to use the training data as to learn about the English language and automate tasks surrounding the English language, not to extrapolate personal information or profit off of the publicly available datasets. Some mitigation strategies that this project will employ is to not publish any sample data points that might contain personal information about the author and to discourage use for profit of these datasets with an effort to protect the privacy of the datapoints.

References

A Appendix (optional)

A.1 References

[1] Archinet, SMART - PyTorch, (2022), GitHub repository, https://github.com/archinetai/smart-pytorch

[2] Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805.

[3] Henderson, M., Al-Rfou, R., Strope, B., Sung, Y. H., Lukács, L., Guo, R., ... & Kurzweil, R. (2017). Efficient natural language response suggestion for smart reply. arXiv preprint arXiv:1705.00652.

[4] Jiang, H., He, P., Chen, W., Liu, X., Gao, J., & Zhao, T. (2019). Smart: Robust and efficient fine-tuning for pre-trained natural language models through principled regularized optimization. arXiv preprint arXiv:1911.03437.

[5] Reimers, N., & Gurevych, I. (2019). Sentence-bert: Sentence embeddings using siamese bert-networks. arXiv preprint arXiv:1908.10084.