We do it BERTer: Comparison of Finetuning Methods to Improve Sentence Embeddings

Stanford CS224N Default Project

Ramya Ayyagari
Department of Computer Science
Stanford University
ayyagari@stanford.edu

Alex Hodges
Department of Computer Science
Stanford University
alexh555@stanford.edu

Abstract

We aim to contribute to the growing body of work that finetunes **Bidirectional Encoder Representations for Transformers (BERT)**(3) for natural language understanding (NLU) tasks by creating a more efficient task-specific model for (1) sentiment analysis, (2) paraphrase detection, and (3) semantic textual similarity. In this paper, we implement a model that achieves an overall accuracy of 0.622 on the test leaderboard, which is an improvement on the traditional BERT model's overall accuracy. In doing so, we demonstrate that an ensemble method combining multi-task finetuning, cosine similarity, the AdamW optimizer, and additional datasets is an effective way to finetune BERT for multiple downstream NLU tasks.

1 Key Information to include

- Mentor: Drew Kaul
- External Collaborators (if you have any): none
- Sharing project: no

2 Introduction

Bidirectional Encoder Representations for Transformers (BERT)(3) is a language representation model that uses both the left and right contexts of unlabeled text to pretrain a deep bidirectional transformer via a "masked language model" (MLM) pre-training objective. The power of BERT lies in its ability to serve as a pre-trained model that can be fine-tuned and applied to a large variety of downstream NLP tasks, all while achieving state-of-the-art performance levels. Our project aims to create and finetune a minimalist implementation of BERT (minBERT)(2).

BERT works by first using a WordPiece tokenizer to convert sentence input into tokens, an example of which is provided below.

Word	Tokens
make	[make]
making	[make, ##ing]
makeshift	[make, ##shift]

BERT then uses a trainable embedding layer to map input ids to vector representations. In our minBERT implementation, the embeddings encode information including the token itself and the position of the word within the input. Following the encoding layer, BERT then utilizes 12 encoder transformer layers (one of which is depicted in Figure 1). The encoder transformer layers consist of a multi-head attention layer, an additive and normalization layer, a feed-forward layer, and a final additive and normalization layer.

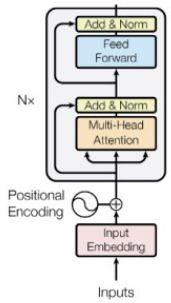

Figure 1: BERT Encoder Layer(1)

While the original BERT model was trained on masked token prediction and next sentence prediction tasks, we can use the pretrained weights of BERT to finetune the model for other downstream tasks. In this project, our goal was to implement the minBERT model and apply it to three downstream tasks: (1) sentiment analysis, (2) paraphrase detection, and (3) semantic textual similarity.

The challenge in this project arises from trying to finetune BERT such that it achieves high performance on multiple tasks rather than just one specific task. We're interested in investigating how we can combine various techniques from NLU literature to improve BERT's performance in our three target task domains and develop new finetuning strategies that could be used in related applications.

Current methods detail different implementations for finetuning the BERT model and many are successful in achieving accuracies higher than the state-of-the-art of traditional BERT in their respective task domains. While there has been extensive work to finetune BERT for specific tasks, less attention has been placed on creating more comprehensive, generalized models for NLU tasks. We aim to build on preexisting work to try to further the accuracy achieved for a select set of three NLU tasks. More specifically, our approach to improving BERT's learned word embeddings involves experimenting with changing the optimizer, dropout rate, loss functions, and datasets used for finetuning.

3 Related Work

Recent works have experimented with finetuning BERT's model structure or parameters to better suit particular downstream tasks, similar to what we propose in this paper. The following works inspired our thought process on some of our extensions, and our aim for our work is to provide insight into how combining some of these methods affects BERT's performance.

Previously, state-of-the-art results on natural language understanding tasks like in the GLUE benchmark finetuned separate BERT models for each task. A paper by Stickland and Murray (8) proposed using a singular BERT model that incorporated task-specific parameters and projected attention layers to match performance results of previous task-specific models. This approach of multitask fine-tuning with BERT is now more widely used and is something we explore as an improvement on our existing BERT model for our three specific downstream tasks.

Adding on to this, a paper by Sun et al. (9) provides a deep dive into how different fine-tuning methods improve BERT's performance on text classification tasks. Specifically, they propose a general framework of approaching BERT finetuning to achieve optimal performance on downstream tasks that sets the state-of-the-art for eight well-known text classification datasets. They conclude that in-domain pre-training and multi-task finetuning noticeably improve BERT's performance, among other findings such as implementing a layer-wise decreasing learning rate to work around the catastrophic forgetting problem. This paper provided us with useful information on which approaches to try first and reinforced our decision to implement multi-task finetuning.

Another paper by Reimers and Gurevych (7) notes that while BERT has state-of-the-art performance on tasks like semantic textual similarity (STS), this accuracy comes with a large computational overhead as it is expensive to determine optimal sentence similarity among collections of tens of thousands

of sentences. This is because traditional BERT uses a cross-encoder, which takes in two sentences and predicts a target value. In pair regression tasks like STS, this problem becomes too computationally expensive. Thus, the authors propose a new BERT-based model, Sentence-BERT (SBERT), which uses siamese and triplet network structures to construct meaningful sentence embeddings that are scored in relation to each other via cosine similarity, which is very computationally efficient. This drastically reduces the time BERT takes to find the most similar pair of sentences in a dataset from 65 hours to 5 seconds and represents a significant improvement in semantic-related large-scale tasks such as similarity comparison, clustering, and information retrieval. This paper implies that finetuning BERT for a specific task like STS can impact BERT's performance on other similar downstream tasks, supporting our idea of incorporating a task-specific dataset for finetuning that could potentially help improve multiple task's accuracies. We also draw from this paper's findings on cosine similarity and its computational efficiency in our proposed work when we implement cosine similarity loss.

4 Approach

First Milestone Our first milestone uses the pre-trained weights from our minBERT model to finetune on the SST and CFIMDB datasets. We used a Cross Entropy (CE) loss function, and we only trained for the sentiment classification task. We used the handout baselines (2) to evaluate our first milestone, and found that our implementation surpasses the baseline accuracies in all four categories.

Model	**Pretraining SST**	**Pretraining CFIMDB**	**Finetuning SST**	**Finetuning CFIMDB**
Provided Baseline	0.390	0.780	0.515	0.966
Our Model	**0.399**	**0.792**	**0.519**	**0.967**

Table 1: Dev Accuracy Results for First Milestone

Since we only trained our model for sentiment classification using the SST and CFIMDB datasets, our initial results weren't great overall (refer to the 'First milestone' row in Table 3), especially with regards to the semantic textual similarity correlation task. Therefore, our motivation for our first extension was to extended our model to perform well on multiple tasks.

Extension 0: Baseline Multitask Finetuning and Loss Functions Our first extension utilizes a round-robin approach to multitask finetuning. In this approach, we take a fixed-size batch of data from each of our datasets, extract the relevant information, predict the logits, then feed the logits into different loss functions depending on the corresponding task. Since not all datasets are the same size, once we reach the end of a smaller dataset, we re-sample batches of the smaller datasets until we get through all batches of the largest dataset. We used PyTorch's loss function classes for each of our downstream tasks (5). More specifically, we used Cross Entropy (CE) loss for sentiment analysis, Binary Cross Entropy (BCE) loss for paraphrase detection, and Mean Square Error (MSE) for semantic textual similarity.

Cross Entropy loss calculates the total difference between probability distributions and can be used for tasks that output probability values between 0 and 1. We use `sum` reduction for the loss. **Binary Cross Entropy** loss extends these ideas to a binary setting. **Mean Square Error** loss calculates the average of the squared differences between the predicted and actual values. We used this loss function instead of Mean Absolute Error because MSE penalizes outliers more harshly. Since the semantic textual similarity dataset is the smallest, any outlier would have a larger effect on the loss.

Since Extension 0 was trained with all three tasks in mind, we decided to use this model as a baseline for future extensions. Furthermore, to isolate effects of our extensions, every extension (unless otherwise specified) only made changes on top of this baseline model and not other extensions.

Extension 1: Averaged Sentence Embeddings A common approach to embed sentences before inputting them into the BERT model is to use their CLS token. However, an altnerative approach to using the CLS token is averaging the word embeddings. We changed our forward function to return the average of the word embeddings instead of the CLS token to see what impact this had on overall model performance.

Extension 2: ARM Dataset for Sentiment Analysis Our sentiment analysis accuracy was low in our initial baseline, so we decided to directly target this by incorporating the Amazon Reviews Multi dataset from Hugging Face, which has a similar distribution to our downstream task. Adding additional datasets is a well-documented method to improve model behavior. We normalized the ARM dataset to match the distribution of the SST dataset so it could be easily incorporated into our existing architecture.

Extension 3: Ensemble Model with Dropout As both Extension 1 and Extension 2 resulted in an overall increase in overall accuracy across tasks, we combined these approaches for our next extension. Furthermore, since our semantic textual similarity correlation scores were consistently high during training but low during validation, we predicted that our model might be overfitting in the task. We addressed this issue by applying two dropout layers to each embedding projection in the `predict_similarity` function. Considering the range of our datasets and downstream tasks, we also decided to try using the PyTorch AdamW optimizer (5) instead of our implementation of the Adam optimizer, since AdamW has been shown to generalize more readily than Adam.

The AdamW optimizer algorithm is an extension of Adam. Both algorithms are presented in Fig. 2. The original Adam algorithm is an extended version of stochastic gradient descent. The algorithm works by maintaining an exponentially moving average of gradients m_t and their squares v_t, given hyperparameters β_1, β_2 to control the exponential rate of decay of these averages. Essentially, when the gradient moves a lot between time steps, Adam takes smaller update steps, and when the gradient doesn't move as much between time steps, Adam takes larger update steps. AdamW moves the regularization term such that it isn't tracked in the moving average of the gradients. This step resolves the asymmetry between the update of the gradient average and squared gradient average.

Algorithm 2 Adam with L_2 regularization and Adam with decoupled weight decay (AdamW)

given $\alpha = 0.001, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}, \lambda \in \mathbb{R}$
initialize time step $t \leftarrow 0$, parameter vector $\boldsymbol{\theta}_{t=0} \in \mathbb{R}^n$, first moment vector $\boldsymbol{m}_{t=0} \leftarrow \boldsymbol{0}$, second moment vector $\boldsymbol{v}_{t=0} \leftarrow \boldsymbol{0}$, schedule multiplier $\eta_{t=0} \in \mathbb{R}$
repeat
  $t \leftarrow t + 1$
  $\nabla f_t(\boldsymbol{\theta}_{t-1}) \leftarrow \text{SelectBatch}(\boldsymbol{\theta}_{t-1})$ ▷ select batch and return the corresponding gradient
  $\boldsymbol{g}_t \leftarrow \nabla f_t(\boldsymbol{\theta}_{t-1}) + \lambda \boldsymbol{\theta}_{t-1}$
  $\boldsymbol{m}_t \leftarrow \beta_1 \boldsymbol{m}_{t-1} + (1 - \beta_1)\boldsymbol{g}_t$ ▷ here and below all operations are element-wise
  $\boldsymbol{v}_t \leftarrow \beta_2 \boldsymbol{v}_{t-1} + (1 - \beta_2)\boldsymbol{g}_t^2$
  $\hat{\boldsymbol{m}}_t \leftarrow \boldsymbol{m}_t / (1 - \beta_1^t)$ ▷ β_1 is taken to the power of t
  $\hat{\boldsymbol{v}}_t \leftarrow \boldsymbol{v}_t / (1 - \beta_2^t)$ ▷ β_2 is taken to the power of t
  $\eta_t \leftarrow \text{SetScheduleMultiplier}(t)$ ▷ can be fixed, decay, or also be used for warm restarts
  $\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \eta_t \left(\alpha \hat{\boldsymbol{m}}_t / (\sqrt{\hat{\boldsymbol{v}}_t} + \epsilon) + \lambda \boldsymbol{\theta}_{t-1} \right)$
until *stopping criterion is met*
return optimized parameters $\boldsymbol{\theta}_t$

Figure 2: Adam and AdamW algorithm (4)

To summarize, this extension is equivalent to Extension 0 plus the average sequence embeddings, the ARM dataset, the AdamW optimizer, and the extra dropout layers for the semantic textual similarity task.

Extension 4: Cosine Similarity Loss Throughout all of our previous extensions, we observed that our semantic textual similarity (STS) correlation scores were the lowest when compared to the scores of other tasks. Therefore, we elected to change the input logits to the MSE loss function for the STS task. Our previous implementation involved applying dropout on the embeddings and concatenating them to get our logits. In this extension, we instead apply a linear projection on the embeddings and then perform cosine similarity using PyTorch's cosine similarity class (5) to give us the final logits. By performing this cosine similarity step, we incorporate more relevant task information into the logits, allowing the model to learn similarity more directly and efficiently.

Extension 5: Ensemble Model with Cosine Similarity Loss Since the previous extension provided promising results, we wanted to incorporate it into our ensemble approach from Extension 3. We debated whether or not to apply the dropout from Extension 3 and the cosine similarity from Extension 4 to the STS logits. We decided to prioritize cosine similarity since it gave us better accuracy results. We also noted that if we tried to incorporate both extensions by applying dropout to the embeddings

before we passed them into the cosine similarity function, we might lose important information about the similarity of the embeddings.

To summarize, this extension is equivalent to Extension 0 plus the average sequence embeddings, the ARM dataset, the AdamW optimizer, and the Cosine Similarity function for the semantic textual similarity task.

5 Experiments

5.1 Data

Stanford Sentiment Treebank (SST) dataset This dataset consists of annotated phrases from movie reviews. Each pair is labeled for sentiment as negative (0), somewhat negative (1), neutral (2), somewhat positive (3), or positive (4). We used this dataset for the sentiment classification task. This dataset was split up into 8,544 training examples, 1,101 dev examples, and 2,210 test examples (2).

CFIMDB dataset This dataset consists of more polar movie reviews, some of which may be longer than a single sentence. We used this dataset for the sentiment classification task. Each sentence is labeled for sentiment as negative (0) or positive (1). This dataset was split up into 1,701 training examples, 245 dev examples, and 488 test examples.

Amazon Reviews Multi (ARM) dataset This dataset consists of single to multi-sentence reviews from Amazon, labeled from 1 (worst) to 5 (best) stars. We adjusted these scores to be from 0 to 4 to match the SST dataset. We used this dataset to improve our performance on the sentiment classification task. Due to the fact that it has a similar distribution to the SST dataset, we believed it would improve our sentiment classification accuracy. We used the English subset of the dataset, which consisted of 200,000 train reviews, 5,000 dev reviews, and 5,000 test reviews.

Quora Question Pairs (QQP) dataset This dataset consists of question pairs labeled as 0 or 1 to represent if the questions are paraphrases of each other. We used this dataset for the paraphrase detection task. This dataset was split up into 141,506 training examples, 20,214 dev examples, and 40,431 student test examples.

SemEval Benchmark Dataset (STS) dataset This dataset consists of pairs of sentences of varying similarity. Each pair is labeled on a scale from 0 (unrelated) to 5 (same meaning). We used this dataset for the semantic textual analysis task. This dataset was split up into 6,040 training examples, 863 dev examples, and 1,725 student test examples.

Dataset Examples For further clarity, an example of a single entry in each dataset is represented in Table 2. We used "N/A" to indicate if the field does not exist for the specified dataset.

Dataset	Sentence 1	Sentence 2	Label	Label Range
SST	“No surprises.”	N/A	1	{0, 1, 2, 3, 4}
CFIMDB	“This movie is definitely on the list of my top 10 movies to never watch again...”	N/A	0	{0, 1}
QQP	“Can deleted pictures on Instagram be recovered?”	“How do I delete a picture on Instagram?”	0.0	{0, 1}
STS	“The woman is thinly slicing a cucumber.”	“The woman is slicing a tomato.”	1.2	{0 - 5}
ARM	"Does not work as described."	N/A	1	{ 1, 2, 3, 4, 5 }

Table 2: Example Dataset Entries.

5.2 Evaluation method

We use quantitative evaluation metrics from the dev and test leaderboards, which evaluate our models on the dev and test sets, respectively. Since we had unlimited submissions to the dev leaderboard, we used these metrics to decide how to proceed after implementing a given extension. We compared our extension scores to the baseline score from Extension 0. We used the test leaderboard, which was capped to 3 submissions, for our top 3 extensions as scored by the dev leaderboard. Both leaderboards included scores for sentiment classification accuracy, paraphrase detection accuracy, semantic textual similarity correlation, and the overall score (the average of the three previous scores). Since paraphrase detection and sentiment classification use discrete labels, we use the accuracy metric, which returns the ratio of correct predictions to all predictions made. However, since the STS task measures degrees of similarity which is a continuous metric, we use Pearson correlation to highlight how similar the predicted labels are to the actual labels.

5.3 Experimental details

We loaded BERT's pretrained weights into our minBERT model, which we later finetuned for our downstream tasks. We used the following configuration for each experiment, unless otherwise specified:

```
-option=finetune -epochs=10 -lr=1e-5 -batch_size=16 -hidden_dropout_prob=0.3.
```

We ran a total of 6 experiments on a CUDA GPU, with training time averaging around 12 hours for 10 epochs. Specifically, we noticed that Extension 4 took much less time to run (around 8 hours), perhaps because of cosine similarity's computational efficiency as described in the SBERT paper (7). Additionally, since the ARM dataset wasn't included in this extension, there was less data to train on overall, leading to a decrease in training time.

5.4 Results

The following tables illustrate our results for the dev and test leaderboard metrics.

Dataset	**Overall Score**	**Sentiment Acc.**	**Para Detection Acc.**	**STS Corr.**
First milestone	0.315	0.302	0.625	0.020
Baseline	0.513	0.470	0.783	0.286
Embedding	0.519	0.464	0.779	0.313
ARM dataset	0.523	0.460	0.779	0.331
Ensemble 1 (Dropout)	0.530	0.466	0.781	0.341
Cosine Similarity	**0.621**	**0.497**	**0.785**	0.582
Ensemble 2 (Cosine Similarity)	0.610	0.450	0.784	**0.597**

Table 3: Dev Leaderboard Results.

Embedding slightly improved the overall score (only a 0.006 increase). We didn't have any preconceived notions about this, rather, we just noted that the CLS token and averaging BERT embeddings are both viable options. However, it is noted in the SBERT paper (7) that both perform relatively similarly, so we were mainly testing it on our specific downstream task to see if it would be beneficial here.

The **ARM dataset** improved our overall score, decreased our sentiment accuracy and paraphrase accuracy, and increased our STS correlation. While we expected the improvement in overall score, we were surprised that it didn't improve it too much (0.01 increase). We were also surprised that the reason it improved our overall score was largely due to STS correlation (0.045 increase), even though the ARM dataset wasn't added with the goal of improving this specific metric. While we were anticipating that the ARM dataset could improve the scores of other tasks as well, we weren't expecting the largest increase in accuracy to come from STS and the largest decrease in accuracy to come from the task we were trying to increase, sentiment classification (0.01 decrease). This may be

because the ARM dataset sentences were further off from the distribution of the SST sentences than we expected, which lowered the SST accuracy overall.

Ensemble 1 with dropout increased our overall score by 0.017. This was about what we were expecting, especially considering the results from previous extensions. This result demonstrates that our previous extensions could extend upon each other.

Cosine Similarity had the greatest improvement regarding our overall score compared to the baseline. Most of this improvement was due to a significant increase in the STS correlation score (0.251 increase). Cosine Similarity directly targeted the STS task by changing the logits to encode more similarity information, so this increase was expected. We didn't have any predictions for the sentiment and paraphrase scores, but the fact that the sentiment classification improved slightly (0.031 increase) demonstrates that this task may use similar underlying information to the STS task.

Ensemble 2 Ensemble 2 performed slightly worse than Cosine Similarity on the dev set. We were expecting the positive results of Ensemble 1 to augment the results of Cosine Similarity when combined, so we were surprised by the decrease of 0.11 in overall score. The STS score in Ensemble 2 increased following the same logic as in the Cosine Similarity Extension. However, the sentiment accuracy scores decreased, hinting that the combination of techniques may not work as well together for our specific tasks.

Dataset	**Overall Score**	**Sentiment Acc.**	**Para Detection Acc.**	**STS Corr.**
Ensemble (Dropout)	0.535	0.513	0.787	0.305
Cosine Similarity	0.618	**0.516**	0.790	0.546
Ensemble 2 (Cosine Similarity)	**0.622**	0.507	**0.793**	**0.567**

Table 4: Test Leaderboard Results.

Our test results showed slight variations in scores compared to the dev results due to differences in the actual data being tested. Therefore, our final two extensions (Cosine Similarity and Ensemble 2) have comparable results and we would need further testing to determine which of the two is stronger. We assume for now that our best model is the one that achieved the highest overall score on the test leaderboard, which is Ensemble 2.

6 Analysis

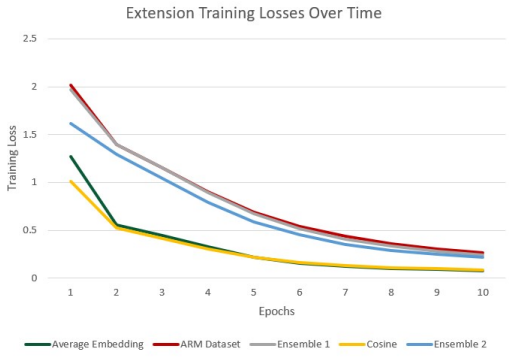

Figure 3: Extension Training Loss Graph

Figure 3 depicts our training loss over different extensions. All achieve expected loss curve shapes, showing that our model is appropriately learning overall. While this may be the case, when examining the metrics at each epoch for our best model (Ensemble 2), we noticed that the model was overfitting on the semantic textual similarity (STS) task. During the Ensemble 2 run, we saw that the STS training accuracy was around 0.981 while the STS dev accuracy was 0.581 around epoch 9. While these gaps reduced significantly around epoch 10, we can still conclude that the model overfits to

the training dataset in this task. This can be explained by the fact that the dataset for this task is much smaller than the datasets used for the sentiment classification and the Quora dataset used for the paraphrase detection task. When we use the round-robin approach and train over the STS dataset multiple times, overfitting is likely to occur.

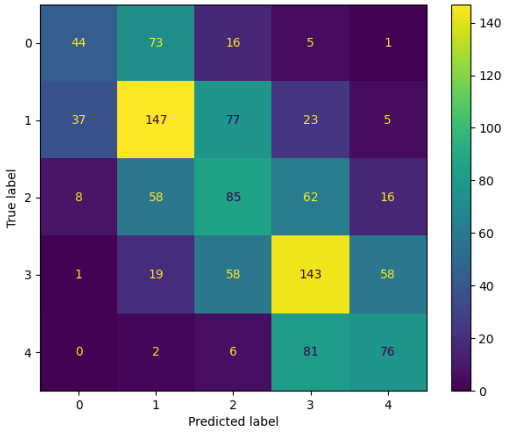

Figure 4: True Label vs. Predicted Label Confusion Matrix for SST Dev Dataset

Figure 4 illustrates the number of predicted labels that match with true labels for the SST dataset in the sentiment classification task. Upon observing the confusion matrix, we immediately see that the sentiments of sentences with true labels of 1 (somewhat negative) and 3 (somewhat positive) were often predicted correctly, and in general most sentiment labels were off by 0 or 1, which is promising. We also see that we tend to predict a more positive sentiment than actually present judging by the distribution of confusion matrix values, which could be due to a difference in the distribution of training set and validation set data.

7 Conclusion

In finetuning BERT, we learned that many alterations can result in small changes, but to greatly increase performance, it's most beneficial to finetune over more data and implement loss functions that are chosen specifically for each task. We created a model, Ensemble 2, which achieved strong results overall (0.622 on the test set). This model's score of 0.610 on the validation set represented around a 0.3 point increase from our original baseline of traditional BERT's overall score on the validation set, illustrating the benefit of combining task-specific alterations with loss functions that encode information pertinent to the specific task.

We were primarily limited by training time and limited GPU usage. However, in future work, we hope to run our model on more epochs and run multiple simulations at once so that we are able to observe the long-term effect of how our loss functions and accuracies level out. These extensions could include gradient surgery and adding more datasets with similar distributions to our downstream tasks (especially for sentiment classification and semantic textual similarity). We hope additional datasets will help us address overfitting. We also hope to use gradient surgery to amplify the impact of multitask finetuning. Until this point, we've relied solely upon multitask fine-tuning for all of our extensions. However, depending on how our model is finetuned, different tasks may conflict with each other based on their gradient directions. Therefore, using gradeint surgery will also reveal if the gradients of our tasks conflicted with each other and subsequently result in decreased performance.

Our paper only dealt with three specific tasks, so all of our approaches were focused on directly improving one or more of these three tasks. Another direction we could take is introducing another task and seeing how all tasks' accuracies are affected by it. We can also change how similar or different the new task is to our other three tasks to observe the impact this has on overall model accuracy.

References

[1] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In Advances in Neural Information Processing Systems, pages 5998–6008, 2017.

[2] CS224n Teaching Team. "Default Final Project: minBERT and Downstream Tasks." 2023, http://web.stanford.edu/class/cs224n/project/default-final-project-bert-handout.pdf

[3] Devlin, Jacob, et al. "Bert: Pre-training of deep bidirectional transformers for language understanding." arXiv preprint arXiv:1810.04805 (2018).

[4] Loshchilov, Ilya, and Hutter, Frank. "Decoupled Weight Decay Regularization." arXiv, 2017, https://doi.org/10.48550/arXiv.1711.05101.

[5] Paszke, Adam et al. "Automatic differentiation in PyTorch." NIPS-W. 2017.

[6] Phillip Keung, Yichao Lu, György Szarvas and Noah A. Smith. “The Multilingual Amazon Reviews Corpus.” In Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, 2020.

[7] Reimers, Nils, and Iryna Gurevych. "Sentence-BERT: Sentence Embeddings Using Siamese BERT-Networks." 2019, arxiv.org/pdf/1908.10084.pdf.

[8] Stickland, Asa, and Murray, Iain. "BERT and PALs: Projected Attention Layers for Efficient Adaptation in Multi-Task Learning." ArXiv, 2019, /abs/1902.02671.

[9] Sun, Chi, et al. "How to Fine-Tune BERT for Text Classification?" ArXiv, 2019, /abs/1905.05583.

[10] William B. Dolan and Chris Brockett. 2005. Automatically Constructing a Corpus of Sentential Paraphrases. In Proceedings of the Third International Workshop on Paraphrasing (IWP2005).