Using Gradient Surgery, Cosine Similarity, and Additional Data to Improve BERT on Downstream Tasks

Stanford CS224N Default Project

Kasen Stephensen
Department of Computer Science
Stanford University
`albionst@stanford.edu`

Joseph Seiba
Department of Computer Science
Stanford University
`jseiba@stanford.edu`

Chanse Bhakta
Department of Computer Science
Stanford University
`cbhakta@stanford.edu`

Abstract

For our project, we aimed to improve BERT's performance on three downstream tasks: Sentiment Classification, Paraphrase Detection, and Semantic Textual Similarity. Our strategies were inspired by many proven methodologies as referenced in this paper. Our implementation included cosine similarity, gradient surgery, training interleaving, and pretraining on additional data. Our findings showed that we were able to improve performance on all three tasks. We also found that pretraining on additional data failed to improve the model, partly because the dataset for our task was fundamentally too different from our other datasets. Our findings show that gradient surgery, cosine similarity, and training interleaving are viable improvements for our three downstream tasks.

1 Key Information

- Mentor: Rohan Taori
- External Collaborators : None
- Sharing project: No
- Team Contribution: See appendix C

2 Introduction

The evolution of Natural Language Processing (NLP) has been significantly advanced by the introduction of new deep learning models, which have shown to be capable of both understanding and generating authentic human language. Of these groundbreaking models, Bidirectional Encoder Representations from Transformers (BERT) have proved to be innovative in the way that they pre-train on a large corpus of text and fine-tune for specific NLP tasks. This model architecture has set the framework for achieving peak performance across a variety of linguistic problems within NLP.

The original BERT paper states that BERT model uses the Transformer architecture, not the traditional neural network architecture, to process sequences of words Devlin et al. (2018). This design choice allows BERT to capture more context within written text, regardless of their position and difference in the word sequence. With this, BERT can generate far better contextualized sequences than traditional models, which aid in tasks requiring seemingly authentic human text generation. The BERT model's downfall, with its large computational cost and model size, is deploying it within an environment with a lack of computational resources. To solve this problem, our project develops a minimal and deployable version of BERT, called minBERT.

The minBERT model is tailored to maintain its advanced contextual NLP capability, while optimizing for computational efficiency and model size. Our goal is to implement the model and make sure it performs well in three separate linguistic tasks. We start with sentiment analysis, aiming to fine-tune minBERT's sensitivity to context in written text, making it a versatile tool for businesses, social platforms, and accurately gauging public sentiment. Next, we wanted to solve the problem of paraphrase detection, where the goal is to understand the nuances of language, identifying when two pieces of text are essentially saying the same thing but in different words. This is crucial for tasks like summarizing content, detecting plagiarism, or optimizing search engines. Finally, we wanted to address Semantic Textual Similarity (STS), measuring and interpreting the hidden feelings and views within text, on a graded scale. In addition to our three standard tasks, we also aimed to include three extensions which would improve the quality of our minBERT, including gradient surgery, loss aggregation, and an additional dataset.

The technical goal of our project, and the general development of minBERT as opposed to BERT, is to reduce computational load, implement more efficient memory usage, and allow for faster training and tuning. As a result, minBERT will have many social effects such as accessibility to those with limited hardware capabilities (educators, students, non-technical users, etc.), faster deployment for international crises, the decrease of environmental footprint through lower power consumption, and promoting access to advanced NLP tools for ethical AI development and legislation.

3 Related Work

In the rapidly changing field of Natural Language Processing (NLP), the development of BERT has been a significant breakthrough, offering an advanced understanding of, and ability to generate, human-like text. Our project builds upon this foundation, and extends BERTs application through the use of extension, inspired by other research papers in the field of NLP. This section assesses the various research papers that shaped our approach, focussing on advancements in BERT's architectural extensions, sentiment analysis, cosine similarity, and gradient surgery, each contributing to the final iteration of our minBERT model.

3.1 Sentiment analysis of Chinese stock reviews based on BERT model:

Our first paper demonstrates an innovative application of the BERT model, using its bidirectional technique for effective sentiment classification in a niche field. In the paper, Mingzheng et al. (2020), the approach involved fine-tuning BERT on a subject-specific dataset, where certain hyperparameters, including learning rate and batch size, could be optimized for classification performance for a specific field. Particularly, their use of a linear layer function, paired with ReLU activation served a basis of design inspiration for our STS task. By integrating similar strategies like fine-tuning on an additional dataset (HuggingFace), and adjusting our learning rate (1e-5 to 1e-4), we tailored minBERT to capture and classify sentiment nuances in various types of text.

3.2 Gradient Surgery for Multi-Task Learning:

Yu et al. (2020) assesses gradient surgery, crucial for reducing conflict between gradients from different tasks in multitask situations, improved our project's approach to training minBERT efficiently. This method involves the calculation and adjustment of task-specific gradients to prevent them from interfering negatively with each other, which facilitates smooth multitask learning. The core mathematical technique used in gradient surgery involves projecting conflicting gradients onto a plane where they do not oppose each other, as is shown in equation 7. This adjustment ensures that the training process in multitask settings, like ours, benefits from the net improvement across other tasks without the risk of gradient interference. By using this technique in our training process, we were able to fine-tune minBERT on sentiment analysis, paraphrase detection, and textual similarity tasks with substantial improvement in model performance for all tasks. The choice to include gradient surgery, inspired by this paper, was instrumental in achieving learning and performance rates in our project.

3.3 A hybrid approach of Weighted Fine-Tuned BERT extraction with deep Siamese Bi – LSTM model for semantic text similarity identification:

Incorporating cosine similarity into our project was inspired by this paper, creating a metric for analyzing textual similarity Viji and Revathy (2022). Cosine similarity, defined mathematically as the cosine of the angle between two nonzero vectors in a multi-dimensional space, serves as a measure of orientation and not magnitude, making it useful for comparing text vectors from BERTs embeddings. The equation relies on calculating the dot product of the vectors normalized by their magnitudes, shown in equation 4. This result ranges from -1 to 1, where 1 indicates identical orientation (high similarity). In our minBERT model, we used cosine similarity to improve the STS task, allowing the model to understand semantic similarities between sentences with accuracy. By integrating this with the embedding comparison layer, we adapted minBERT to generate embeddings that, when compared using cosine similarity, returned relationships within nuanced text.

4 Approach

In our BERT model, we first create 3 separate linear layers to focus on the downstream tasks. Given the challenges faced with multitask training as mentioned in the introduction, we understood that there are challenges we would have to navigate in order to have optimal performance across all three tasks. As we planned our approach to solve the following problems we decided to implement multi-task finetuning by interleaving the training of the different tasks. We then evaluated other strategies.

4.1 SST: Sentiment Classification

For our sentiment classification task, we make note of the fact that this is a multiclassification task. Essentially, there are 5 categories for a input to be mapped to. Each phrase has a label of negative, somewhat negative, neutral, somewhat positive, or positive. Our linear layer takes in an embedding, which is of size 768. The output size is then of size 5, since there are 5 categories.

Each embedding is mapped to a probability score, which is then transformed into a prediction label with the $argmax$ function. For our loss function, we thought the default $CrossEntropyLoss$ function from torch was adequate, defined as:

$$\ell(x, y) = \sum_{n=1}^{N} -w_{y_n} \log\left(\frac{\exp(x_{n,y_n})}{\sum_{c=1}^{C} \exp(x_{n,c})}\right) \cdot 1\{y_n \neq \text{ignore\_index}\} \tag{1}$$

4.2 PARA: Paraphrase Detection

For our paraphrase detection task, we first build a neural network capable of handling two embeddings, with a single unnormalized logit as the output. The prediction label is a value of 0 or 1. A value of 0 means the two inputs are not paraphrases of each other, while a score of 1 indicates a paraphrase. It should be noted that the logit produced by the linear layer is unnormalized, and therefore a continuous value from 0 to 1. Our linear layer concatenates both embeddings from sentence A and sentence B, each of both size 768. This input of size $2 * 768$ is then passed through our linear layer to give an (unnormalized) logit. This logit represents a probability, and as a result, used the following equation to produce a prediction label, defined as:

$$\sigma(\text{logit}) = \text{prediction} \tag{2}$$

However, we did not need to use the prediction label in our implementation. To train our model, we first understood that our linear layer outputs logits, and we are dealing with a binary task. As a result, we leveraged torch's binary_cross_entropy_with_logits function for our loss function, defined as:

$$\text{L}(x, y) = -\frac{1}{N} \sum_{i=1}^{N} [y_i \cdot \log(\sigma(x_i)) + (1 - y_i) \cdot \log(1 - \sigma(x_i))] \tag{3}$$

4.3 STS: Textual Similarity

For our textual similarity task, we use the take inspiration from Reimers and Gurevych (2019). As such, we add a cosine similarity score feature onto our linear layer as the last entry. Given the embeddings for sentence A for batch i, a_i and sentence B, b_i, without loss of generality, we can extend this definition to tensors a and b for multiple batches. Similar to the implementation in UKPLab (2023) (which is derived from Reimers and Gurevych (2019)), This represents the logits a vectors as:

$$\text{cosine\_similarity}(\mathbf{a_i}, \mathbf{b_i}) = \frac{\mathbf{a_i} \cdot \mathbf{b_i}}{||\mathbf{a_i}||||\mathbf{b_i}||} \tag{4}$$

However, we must also note that cosine similarity is defined to have a range of $[-1, 1]$. This is largely unsuitable for our purposes because a score of -1 indicates an embedding of opposite magnitude is present. So we normalize the cosine similarity scale to be from 0 to 1.

We then build a neural network [1] capable of handling two embeddings, with a single logit as the unnormalized logit as the output. The prediction label ranges from varying similarity on a scale from 0 (unrelated) to 5 (equivalent meaning). It should be noted that this is also a continuous scale. The approach by Viji and Revathy (2022) used euclidean distance. Our linear layer instead takes the squared difference from sentence A and sentence B, each of both size 768. This input is then passed through our linear layer to give an (unnormalized) logit. We also append the cosine similarity score to the the end of the squared distance of the embeddings as an additional feature to the linear layer. We then added intermediate layers with ReLU activation. Ultimately, we have a linear layer input:output of $(768 + 1 : 512 \rightarrow ReLU \rightarrow dropout \rightarrow 512 : 256 \rightarrow ReLU \rightarrow 256)$:1. (see Appendix B)

Since, the output of our linear layer prediction function, a logit, represents a probability, and as a result, used the following equation to produce a prediction label:

$$5 \cdot \sigma(\text{logit}) = \text{prediction} \tag{5}$$

We define our predicted label for batch i as $\hat{y}_i$ and our true label as y_i. Without loss of generality, we compute the batch-wise loss of $\hat{y}$ and y following the approach used by Reimers and Gurevych (2019) via the MSE loss:

$$\text{MSE}(\text{input}, \text{target}) = \text{MSE}(\hat{y}, y) \tag{6}$$

4.4 Further Implementations to Improve Performance

After analyzing our baseline performance when implementing our training loop, we contemplated the reasons for our relatively poor performing model. We understood all three tasks were fundamentally different, and as such, optimal parameters for one task could be at odds with another. As such, we chose to implement gradient surgery in accordance with the package $PCGrad$ Tseng (2020). The mathematical expression for gradient surgery is (Yu et al., 2020) :

$$g_i = g_i - \frac{g_i \cdot g_j}{||g_j||^2} \cdot g_j \tag{7}$$

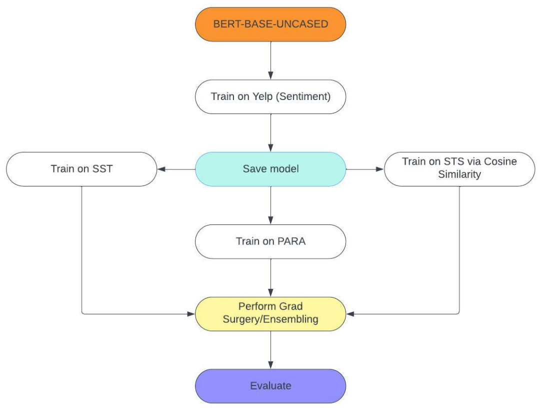
We also adjusted our original training loop, which training on each dataset separately, to now used batched samples from each dataset. Our improved training loop obtains a batch of training data for each task per epoch. Meaning, that all the losses were ensembled together, albeit in different loss functions which will be discussed later. Once all the loss functions were computed, we aggregated our losses as described in Bi et al. (2022). Defining the total loss as

$$\mathcal{L}_{Total} = \mathcal{L}_{SST} + \mathcal{L}_{PARA} + \mathcal{L}_{STS} \tag{8}$$

Finally, we noticed our sentiment task evaluation on the SST dataset was performing significantly worse than the other two tasks . As a result, we chose to further train on a dataset that resembled the SST dataset so the model can learn the relevant parameters before hand, and begin training with more seen examples on the dataset. We were able to procure a dataset from Huggingface's datasets module,

[1]via tutorial from `https://pytorch.org/tutorials/beginner/basics/buildmodel_tutorial.htmlPyTorch`

specifically `yelp_review_full`. Finding this dataset was inspired by the methodology in Zhang et al. (2015), who trained on this sentiment task with $650,000$ samples. This dataset was ideal for our needs since the reviews were normalized from 1 star to 5 star reviews, to 0 to 4. This was suited for our task considering that the SST training dataset is formatted in the same way . As a result, we were able to adapt our previously written code to readily adapt and train on this dataset. Our methodology for doing so is as follows:


BERT-BASE-UNCASED
Train on Yelp (Sentiment)
Train on SST
Save model
Train on STS via Cosine Similarity
Train on PARA
Perform Grad Surgery/Ensembling
Evaluate

5 Experiments

5.1 Data

We used several datasets while fine-tuning our model across the three tasks.

Yelp Review dataset[2] [Sentiment Analysis, pretraining]
This dataset consists of 700,000 full, multi-sentence Yelp reviews. The sentiment is analyzed on a five point scale from **0 (very negative)** to **4 (very positive)**. Using this dataset was ideal since it was formatted nearly identical to our SST task.

Stanford Sentiment Treebank (SST) dataset[3] [Sentiment Analysis]
This dataset consists of 11,855 single sentences extracted from movie reviews. These sentences were parsed into individual phrases, which were given one of five labels (**negative, somewhat negative, neutral, somewhat positive, or positive**) by three human judges.

CFIMDB dataset [Sentiment Analysis]
This dataset consists of 2,434 movie reviews which have a binary label of **negative** or **positive**.

Quora dataset [Paraphrase Detection]
This dataset consists of 400,000 question pairs with binary labels that indicate whether pairs are paraphrases of one another.

SemEval STS Benchmark dataset [Semantic Textual Similarity]
This dataset consists of 8,628 pairs of sentences that vary on similarity from **0 (unrelated)** to **5 (equivalent meaning)**.

5.2 Evaluation method

We have an evaluation metric for each task.

Task	Metric	Rationale
Sentiment Analysis	Accuracy	Given the labeled data, easy to compute
Paraphrase Detection	Accuracy	Given the binary labels, very easy to compute
Semantic Textual Similarity	Pearson score	Following original paper (Agirre et al., 2013)

[2]`https://huggingface.co/datasets/yelp_review_full`
[3]`https://nlp.stanford.edu/sentiment/treebank.html`

5.3 Experimental details

As previously mentioned, we implemented a number of extensions. Our general methodology for evaluating these adjustments was to compare our new model to our best model up until that point by looking at performance on our metrics for our dev sets. In other words, our extension had to improve our dev scores for us to accept it.

When we tested a new technique, we would pretrain our model by training over 10 epochs with the default dropout rate of 0.3. Although this took much more time than simply comparing scores after a single epoch, this allowed us to remove some sense of stochastiticy by giving the model more time to converge on model parameters that we could safely compare.

5.4 Results

Our baseline scores for a "basic" model (that without any extensions) were:

Metric	Value
SST dev accuracy	0.364
Paraphrase dev accuracy	0.686
STS dev correlation	0.252

Table 1: Standard Performance Metrics

Multi-task Fine-Tuning and Gradient Surgery
We implemented multi-task fine-tuning by restructuring our training to fine-tune on the three tasks simultaneously. As previously shown, we would sum the losses for each task and then use gradient surgery to handle conflicting tasks. Our results did improve on our basic model:

Metric	Value (Change)
SST dev accuracy	0.459 (+0.095)
Paraphrase dev accuracy	0.684 (-0.002)
STS dev correlation	0.318 (+0.066)

Table 2: Gradient Surgery + MultiTask Performance Metrics

Cosine Similarity
Another implementation we added was using cosine similarity to improve training on Semantic Text Similarity. This was only possible after we had refactored our model to train on the three-tasks simultaneously. Our results continued to improve:

Metric	Value (Change)
SST dev accuracy	0.457 (-0.002)
Paraphrase dev accuracy	0.687 (+0.003)
STS dev correlation	0.323 (+0.005)

Table 3: Cosine Similarity Performance Metrics

Additional Pre-training
We wanted to continue improving scores for all three tasks, so we decided to introduce additional pre-training. We found a dataset from Yelp which seemed to be more domain-specific, but we found that it made our scores worse. This surprised us, as we thought adding additional, domain-specific data would drastically improve our model across all metrics.

Metric	Value (Change)
SST dev accuracy	0.443 (-0.030)
Paraphrase dev accuracy	0.680 (-0.033)
STS dev correlation	0.309 (-0.007)

Table 4: Yelp Performance Metrics

Final Results
Here are our final results on the test leaderboard after we fully fine-tuned our model compared to our "basic" model.

Metric	Value (Change)
SST dev accuracy	**0.490** (+0.126)
Paraphrase dev accuracy	**0.708** (+0.022)
STS dev correlation	**0.707** (+0.455)

Table 5: Test Performance Metrics (all implementations active except for additional data training)

Two of our changes improved our model, but we were very surprised that our additional pretraining made our model regress in performance. Upon further analysis, we realized that the data in the Yelp dataset was not similar enough to our datasets used for sentiment analysis. While Yelp dataset had the same labels as the Stanford Treebank dataset, the data itself was quite different. Specifically, the Stanford Treebank dataset consisted of sentence phrases while the Yelp dataset consisted of full, multi-sentence reviews. This mismatch likely caused our model's performance to worsen.

6 Analysis

6.1 Our Improvements

After analyzing our results, we remain surprised on how little our extensions were able to improve the overall performance of the model when not finetuning. On one hand, we expect that the gradients of each task likely conflict with each other. Therefore, one task's gain in performance may be another task's downturn. This issue was largely remedied by gradient surgery, courtesy of the PCGrad package Tseng (2020). As a result, we recognize that gradient surgery did indeed improve the overall performance of the model. We also saw benefit when combining the loss functions of all three tasks when interleaving training as in equation 8.

Cosine similarity was another implementation that we saw significant improvement, especially when fine-tuning the model. We also conclude that adding more layers intermediate layers for this task, paired with dropout and ReLU activation likely boosted performance as well. As a result, it makes intuitive sense for our combination of cosine similarity and increased linear layer complexity allowed the model to grasp more complex relationships in the data.

6.2 Our Shortcomings and Possible Explanations

We were most surprised by our regression in our performance when incorporating additional pretraining on the YELP dataset. The YELP dataset was ideal to plug into our model since it was perfectly formatted for our needs. The dataset was in English and had sentiment scores on the same scale as the SST dataset. Intuitively, we would expect, at worst, for the PARA and STS tasks to regress as a result of this emphasis on the sentiment task. After analyzing our results, we have come to a conclusion that although the data is formatted the same way, it not necessarily the same content being trained on. There are two possible culprits for this lack of improvement:

- **Textual sample length** Looking at the first entry of the YELP dataset, and the first entry of the SST training dataset we likely see our first problem:

Dataset	Sentence
YELP	Dr. Goldberg offers everything I look for in a general practitioner. He's nice and easy to talk to without being patronizing; he's always on time in seeing his patients; he's affiliated with a top-notch hospital (NYU) which my parents have explained to me is very important in case something happens and you need surgery; and you can get referrals to see specialists without having to see him first. Really, what more do you need? I'm sitting here trying to think of any complaints I have about him, but I'm really...
SST	The Rock is destined to be the 21st Century's new 'Conan' and that he's going to make a splash even greater than Arnold Schwarzenegger, Jean-Claud Van Damme or Steven Segal.

Table 6: Sentences from the YELP and SST datasets.

The YELP samples are significantly longer in length than the training samples provided. As a result, it is more difficult for the model to generalize a larger sample and therefore, likely leads to poorer performance. This pattern repeats itself for many of the samples in both datasets. In addition, sentiment may not be equivalent the "sentiment" defined in the YELP dataset, since it is used more as a review score.

- **Insufficient Training** Due to the sheer number of samples, We only train on a random sample of 1% of the available YELP dataset for 5 epochs. As a result, there is not only a large sentence to generalize, but not enough time to do so. This training time was largely decided by virtual machine and time constraints:

7 Conclusion

In our work, we experimented with different strategies to improve BERT's performance on 3 downstream tasks: sentiment classification, paraphrase detection, and semantic similarity. We implemented cosine similarity, gradient surgery, additional pretraining, and loss aggregation to improve our model. However, we were surprised to receive relatively modest gains from our baseline implementation. Meaning, there may have been an unforeseen error in our implementation. There is also the possibility that the extensions we implemented are not as compatible as we had thought, due to a variety of reasons mentioned in the analysis portion of this paper. We were able to achieve decent performance results with our implementation, but still fall short in making BERT capable of multitasking of high performance across all tasks. We are particularly satisfied by our performance on the paraphrase task, which may be due to the large amount of trainable data. On the other hand, if we had access to more powerful GPUs, additional time, etc, we would have liked to explore how the adjusting the batch size and other parameters (such as learning rate) affects the performance of the model as well as implement and experiment with regularization. We also learned that there are instances where a model's parameters can benefit one task, but simultaneously regress another. We also learned that just because a dataset is formatted similarly to a task, it does not automatically make it viable for pretraining. With these lessons, we are confident we can improve upon our findings given more time and resources.

References

Eneko Agirre, Daniel Cer, Mona Diab, Aitor Gonzalez-Agirre, and Weiwei Guo. 2013. *SEM 2013 shared task: Semantic textual similarity. In *Second Joint Conference on Lexical and Computational Semantics (*SEM), Volume 1: Proceedings of the Main Conference and the Shared Task: Semantic Textual Similarity*, pages 32–43, Atlanta, Georgia, USA. Association for Computational Linguistics.

Qiwei Bi, Jian Li, Lifeng Shang, Xin Jiang, Qun Liu, and Hanfang Yang. 2022. Mtrec: Multi-task learning over bert for news recommendation. In *Findings*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Hang Hua, Xingjian Li, Dejing Dou, Chengzhong Xu, and Jiebo Luo. 2021. Noise stability regularization for improving BERT fine-tuning. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 3229–3241, Online. Association for Computational Linguistics.

HuggingFace. Yelp review full dataset. `https://huggingface.co/datasets/yelp_review_full`. Accessed: 3/9/24.

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.

Li Mingzheng, Chen Lei, Zhao Jing, and Li Qiang. 2020. A chinese stock reviews sentiment analysis based on bert model.

Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084*.

Wei-Cheng Tseng. 2020. Weichengtseng/pytorch-pcgrad.

UKPLab. 2023. Cosinesimilarityloss.py in sentence transformers. `https://github.com/UKPLab/sentence-transformers/blob/master/sentence_transformers/losses/CosineSimilarityLoss.py`.

D. Viji and S. Revathy. 2022. A hybrid approach of weighted fine-tuned bert extraction with deep siamese bi – lstm model for semantic text similarity identification. *Multimedia Tools and Applications*, 81:6131–6157.

Tianhe Yu, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. 2020. Gradient surgery for multi-task learning. *Advances in Neural Information Processing Systems*, 33:5824–5836.

Xiang Zhang, Junbo Zhao, and Yann LeCun. 2015. Character-level convolutional networks for text classification. *Advances in neural information processing systems*, 28.

A Appendix

B Vector Visualization

Let *sentiment analysis linear layer input* be the vector given by

$$\mathbf{v} = (\text{embedding})$$

Let *paraphrase detection linear layer input* be the vector given by

$$\mathbf{v} = \begin{pmatrix} \text{embedding1} \\ \text{embedding2} \end{pmatrix}$$

Let *text similarity linear layer input* be the vector given by

$$\mathbf{v} = \begin{pmatrix} (embedding1 - embedding2)^2 \\ \text{(cosine similarity score + 1)/2} \end{pmatrix}$$

C Team Contribution

All team members worked effectively together and equally in this project. Chanse, Kasen, and Joseph all jointly developed code, wrote the report, and contributed to making the poster.