BERT With Multitask Fine-Tuning and Loss Construction

Stanford CS224N Default Project

Prarthna Khemka
Computational and Mathematical Engineering
Stanford University
pkhemka@stanford.edu

Grace Casarez
Computational and Mathematical Engineering
Stanford University
gcasarez@stanford.edu

Abstract

Bidirectional Encoder Representations from Transformers, or BERT, is a transformer-based model that generates contextual word representations, that can then be adapted for use on various natural language processing (NLP) tasks. In this study, our goal is to implement a minimalist BERT model to perform sentence-level tasks simultaneously, including sentiment analysis, paraphrase detection, and semantic textual similarity (STS). To approach the problem, we conduct several experiments using BERT embeddings in combination with various techniques leveraged from concepts including similarity task fine-tuning, multitask loss construction, and round robin sampling procedures. Through our experimentation, we found that using a weighted multitask loss function, in conjunction with combined similarity task fine-tuning, improved BERT's performance across the three tasks. We achieve test performance of 0.526 accuracy for sentiment classification, 0.861 accuracy for paraphrase detection, and a 0.775 Pearson correlation coefficient for STS.

1 Key Information to include

- Mentor: None
- External Collaborators: None
- Sharing project: None

2 Introduction

Bidirectional Encoder Representations from Transformers, or BERT, is a transformer-based model that generates contextual word representations and utilizes surrounding contextual text to discern ambiguous language in texts (Devlin et al., 2018). By making use of these bidirectional word representations, BERT became a foundational element for large language models and therefore, several natural language processing (NLP) tasks. Here, we focus specifically on three tasks: sentiment analysis, paraphrase detection, and semantic textual similarity (STS). Sentiment analysis is the task of identifying polarity within language and classifying it on a discrete spectrum of negative (labeled 0) to positive (labeled 5). Paraphrase detection is the detection of text that is a rewording of a different piece of text. This is a binary classification task where two sentences are jointly labeled 0 (not a paraphrase) or 1 (paraphrase). Semantic textual similarity (STS) seeks to measure how similar in meaning two sentences are, on a continuous scale from 0 (not similar) to 5 (the same).

Addressing these three tasks in the same model is an example of multitask learning, which can be notoriously difficult considering the differences in goal and format for the various tasks. Previous approaches to tackling multitask models have noted that using a single model for multiple tasks presents a multitude of optimization problems, since each task may result in different optimal

parameters (Sun et al., 2019). Consequently, it can be difficult to balance the requirements and preferences of each task. As Yu et al. (2020) note, it is not unusual that a model learning over multiple tasks has a worse overall performance than a model trained for one specific task. However, since multitask models utilize the same structure for different problems, they have great potential as efficient approaches to multitask problems. Thus, there is great incentive to understand how to understand the limitations and increase the performance of multitask models.

In order to extend BERT to a multitask model, we conduct several experiments leveraging various mechanisms from (1) similarity task fine-tuning, (2) multitask loss construction, and (3) round robin sampling procedures. In particular, in section (1), we apply strategies to fine-tune the similarity tasks of STS and paraphrase detection, including cosine-similarity for STS and semantic-enhanced fine-tuning for paraphrase detection. We also implement a combined fined-tuning strategy, where STS and paraphrase detection use a shared layer. In section (2), we experiment with various constructions of multitask loss, using strategies ranging from a naive sum loss, a weighted sum loss, gradient surgery, and regularization (Yu et al., 2020). Finally, in section (3), given the wide variation in available data for each task, we explore various techniques of dataset sampling during a round robin training procedure. Our results show that performance across the three tasks was maximized when (1) using combined fine-tuning to predict for the STS and paraphrase detection tasks, (2) constructing a loss function with learnable weights for each task, and (3) sampling the datasets by stipulating the batch size for the largest dataset. We achieve test performance of 0.526 accuracy for sentiment classification, 0.861 accuracy for paraphrase detection, and a 0.775 Pearson correlation coefficient for STS. Our results indicate that by leveraging BERT's embeddings and using extensions that complement the nature of specific tasks, it is possible to create a functional model that performs well across NLP tasks.

3 Related Work

BERT BERT, a transformer model, is currently state-of-the-art for NLP tasks (Devlin et al., 2018). Some novel approaches previously applied for these NLP tasks that served as baselines include Support Vector Machine (SVM) models like SVM with bigrams (SVM-bi) and SVM with Naive Bayes features and bigrams (NBSVM-bi) that were trained and tested on an IMDB dataset with full length reviews (Wang and Manning, 2012) and Convolutional Neural Network (CNN) 9 layers deep with 6 convolutional layers and 3 fully-connected layers that was trained and tested on an Amazon review dataset from the Stanford Network Analysis Project (SNAP) (Zhang and LeCun, 2015). Both baselines provided contextual comparison for the sentiment analysis task, since our BRET baseline also trains specifically for that task.

Multitask Models Multitask models are models that perform different tasks using the same framework. However, that presents several optimization problems due to varying optimal parameters for each task ((Sun et al., 2019)) and therefore performs worse than a model trained specifically for a given task (Yu et al., 2020). There are a lot of approaches currently trying to balance this trade-off between implementing efficient models for multitask problems and increasing performance of the models. We tackled this trade-off by extending the same framework of our baseline BERT by applying fine-tuning to increase some performance (Stickland and Murray, 2019).

Multitask Fine-tuning Multitask fine-tuning is a method that fine-tunes a model on several tasks at the same time by exploiting the shared structure to enable more efficient learning (Yu et al., 2020). A prominent approach is training different tasks at the same time and then combining their losses into a final loss for the gradient step, which has proven to outperform the baseline BERT itself (Bi et al., 2022). Some other approaches include projecting a task's gradient onto the normal plane of a task with conflicting gradient to minimize the interference of tasks with each other (Yu et al., 2020). Since the multitask fine-tuning approaches outperform our baseline BERT, we follow these methods and take inspiration from the summing of losses to explore a weighted loss structure as well.

4 Approach

4.1 Baseline

We build our model incrementally, on top of the reference implementation of BERT given in the starter code (CS224N, 2023) and handout (Staff). We start with implementing the BERT layer and the multi-head self-attention classes (seen in Figure 1) as highlighted in (Vaswani et al., 2017).

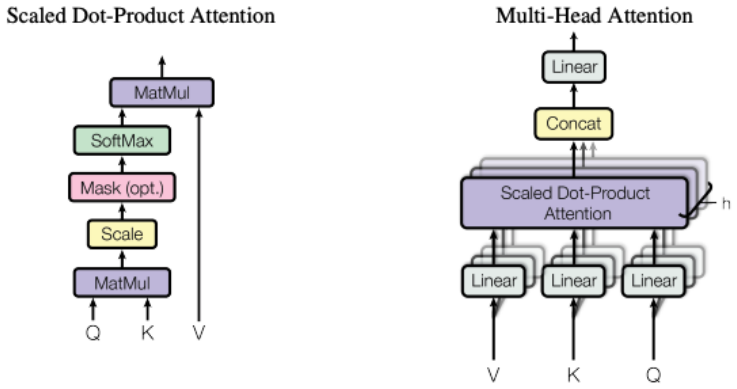


Figure 1: (left) Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several attention layers running in parallel (Vaswani et al., 2017).

Additionally, we implemented the embedding layer in BERT, a BERT sentiment classifier class, and AdamW optimizer step function to further help us with performing sentiment analysis task. The sentiment classified outputs logits using this equation:

$$o = log(softmax(W_t(u)))$$

where u is the sentence encoding from BERT and W_t represents a trainable weight. We use Cross Entropy Loss to train the sentiment analysis task.

4.2 Extension

4.2.1 Similarity Fine-Tuning

Semantic Fine-Tuning In order to extend and fine-tune the baseline BERT model to our task of paraphrase detection, we implement a prediction function that produces logits based on the Siamese Network structure (see Appendix A.2a) using this equation (Khairova et al., 2022):

$$o = softmax(W_t(u, v, |u - v|))$$

where u and v are the vectors for each pair of sentence encodings, $|u - v|$ is the element-wise difference, and W_t represents a trainable weight. We use Binary Cross Entropy Loss to train the paraphrase detection task.

Cosine-similarity Fine-Tuning Cosine-similarity fine-tuning is a technique to measure the similarity between two word embeddings u and v based on the Siamese Network structure (see Appendix A.2b) via cosine similarity:

$$similarity = \frac{u \cdot v}{\max(||u||_2 \cdot ||v||_2, \epsilon)},$$

where ϵ is a small threshold value. We take two approaches to applying cosine-similarity to the STS task. In one approach, we use cosine-similarity to predict a similarity value for the STS task, then used MSE loss to optimize (Reimers and Gurevych, 2019). In another approach, we use the CosineEmbeddingLoss from PyTorch, feeding in the two word embeddings directly [1].

[1] `https://pytorch.org/docs/stable/generated/torch.nn.CosineEmbeddingLoss.html`.

Combined Fine-Tuning Since paraphrase detection and STS are similar in nature, we decided to create a shared embedding layer in our model to combine the two ideas as follows:

$$shared\_emb = (W_t(u, v, |u - v|, \frac{u \cdot v}{||u||_2 \cdot ||v||_2 + \epsilon}))$$

We then passed this shared layer to each of the tasks individually, where after adding non-linearity to the shared embeddings through LeakyReLU, we get logits as:

$$o = softmax(W_t(shared\_emb))$$

which are then passed individually to the respective losses. Here W_t are trainable weights.

Sampling Procedures To extend the baseline BERT to perform well on each task, we implemented a Round Robin approach (see Figure 2) in two ways:

- Data Balancing: Here, we found the smallest dataset length and took random samples of that length from each of the datasets for storage efficient training, so that in each epoch we trained over the same number of batches.
- Full Data: Here, we used batch size as a hyperparameter and set the number of batches as the largest dataset length over the given batch size. This allowed us to create variable batch sizes for each dataset i through

  $$batch\_size_i = max(1, round(length_i/\#batches))$$

  and harness the power of added information to boost performance.

All the subsequent extensions were built on top of these approaches.

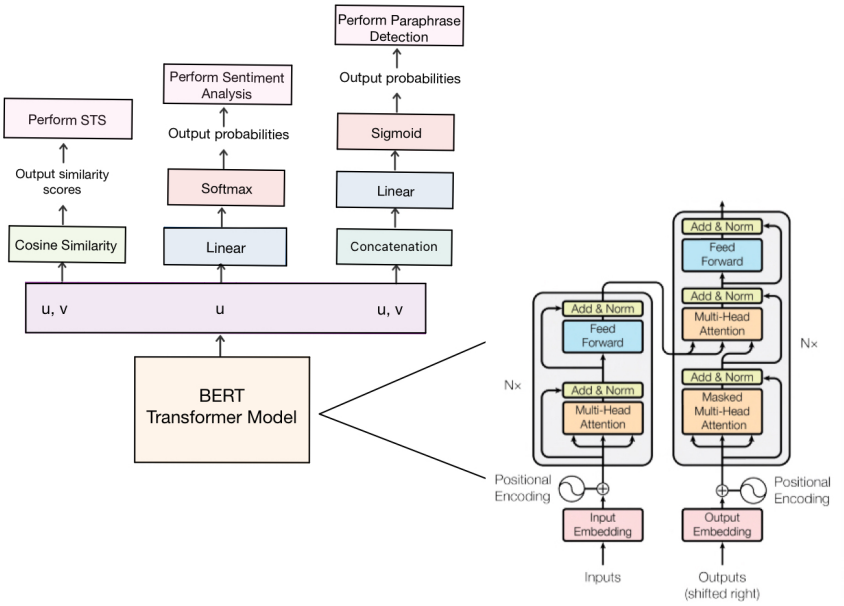


Figure 2: BERT Multitask Round Robin Model

4.2.2 Multitask Loss Construction

Weighted Loss One technique we use is a weighted multitask loss, which combines the three task losses in a weighted sum. We follow the implementation outlined by Kendall et al. (2017), which uses the homoscedastic uncertainty of each task to inform its weight:

$$L = (\exp(-w_1) \cdot l_1 + w_1) + (\exp(-w_2) \cdot l_2 + w_2) + (\exp(-w_3) \cdot l_3 + w_3),$$

where w is a learnable parameter of size 3, the number of tasks.

Gradient Surgery Gradient surgery is a technique to correct conflicting gradient directions among the various task losses. Specifically, gradient surgery projects the gradient of the i-th task g_i onto the normal plane of another task's gradient g_j:

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

In this experiment, we utilize the implementation of gradient surgery in PCGrad (Yu et al. (2020)).

Regularization With fine-tuning to this extent, there is a high chance of over-fitting to the training data and therefore, hindering the model from generalizing to unseen data. To combat this, we performed different types of regularization to penalize the loss functions to reduce capturing complexity, and thus over-fitting.

- L1 Loss: We sum the L-1 norm of each parameter and add this to the loss, weighed by a hyperparameter λ: $L = L + \lambda \sum_{i=1}^{n} |w_i|$.
- L2 Loss: We sum the L-2 norm of each parameter and add this to the loss, weighed by a hyperparameter λ: $L = L + \lambda \sum_{i=1}^{n} w_i^2$.
- SMART: We followed the algorithm described in Jiang et al. (2020) where we implement Smoothness-inducing regularization tuned through lambda which helps manage the complexity of the model and Bregman proximal point optimization that helps prevent aggressive updating of the gradients by adding momentum.

4.2.3 Hyperparameter tuning

We performed hyperparameter tuning for several hyperparameters in these ranges:

- Batch Size: [8, 16, 32, 64]
- Learning Rate: [1e-5, 3e-5, 5e-5]
- Weight Decay: [0.00, 0.01]
- Dropout: [0.1, 0.3, 0.5]
- Lambda (Regularization): [0.0001, 0.0005, 0.001, 0.005]

5 Experiments

5.1 Data

The datasets we use are the Stanford Sentiment Treebank (SST) dataset (Socher et al., 2013), Quora dataset (Iyer et al.), and SemEval STS Benchmark dataset (Agirre et al., 2013).

Table 1: Dataset tasks and splits

	SST	**SemEval STS**	**Quora**
Task	Sentiment Analysis	Semantic Textual Similarity	Paraphrase Detection
Total size (examples)	11,855	8,628	202,152
Train size (examples)	8,544	6,041	141,506
Dev size (examples)	1,101	864	20,215
Test size (examples)	2,210	1,726	40,431

5.2 Evaluation method

For sentiment analysis and paraphrase detection, we use accuracy as our metric of evaluation since these tasks are classification problems. For semantic textual similarity, we will use Pearson Correlation to assess how good the predicted similarity is against the true similarity. These metrics will be compared against the example scores in the handout, our baseline implementation, researched model scores, and through the Gradescope leaderboard.

5.3 Experimental details

5.3.1 Baseline

We first implemented a baseline BERT model that trained only on the sentiment classification head, but was evaluated on all three tasks using the untrained heads of paraphrase detection and STS built on top of BERT embeddings. We ran two experiments of the baseline, one with utilizing pre-trained weights of BERT and the other with finetuning the model. Using GPU, the model trained for approximately 30 seconds per epoch during experiment 1, and about 3 minutes per epoch during experiment 2. For the baseline, we equipped the models with default hyperparameters: Number of Epochs = 10, Dropout Probability = 0.3, Batch Size = 8, Weight Decay 0. For experiment 1 (pre-trained BERT), we used a learning rate of 1e-3, and for experiment 2 (fine-tune BERT), we used a learning rate of 1e-5.

5.3.2 Extended BERT

To improve on the baseline, we extended BERT to perform multitask fine-tuning using Round Robin (MFRR) through several different methods: (1) training on each dataset at the same time by sampling training examples using the smallest dataset length and (2) training on the full datasets through different batch sizes. Then on top of each of these methods, we employed task fine-tuning and techniques like naive summing of losses, regularizing the losses, weighting the losses, and gradient surgery. For the first method of multitask fine-tuning we used a Batch Size of 8 and 16 and Learning Rate of 1e-5 while keeping the rest of the default hyperparameters the same. For the second method of multitask, we used a batch size of 32 and ran hyperparameter tuning on Weight Decay, Dropout, and Learning Rate. For details on which values resulted in the optimal results, see Appendix A.1. On GPU, the first method had a run time of 5-10 minutes while the second experiment had a run time of 20-40 minutes.

5.4 Results

We conducted the experiments listed above on the dev sets of each dataset to compare different model performances and the results of the best performing models are in Table 2. The overall performances can be visually compared through Figure 3, for each task's comparison see Appendix A.3.

Table 2: Accuracies and Correlation scores for the best models on the dev set

Model	SST Accuracy	SemEval Correlation	Quora Accuracy
B1: Baseline 1 (Pre-Trained BERT)	0.396	0.019	0.392
B2: Baseline 2 (Fine-Tuned BERT)	0.534	0.257	0.536
E1: MFRR w/ Data Balancing and Naive loss	0.515	0.432	0.719
E2: MFRR w/ Data Balancing and weighted loss	0.505	0.652	0.736
E3: MFRR w/ Data Balancing and gradient surgery	0.490	0.641	0.730
E4: MFRR w/ Data Balancing, L1 Regularisation, weighted loss	0.505	0.606	0.742
E5: MFRR w/ Data Balancing, L2 Regularisation, weighted loss	0.505	0.600	0.737
E6: MFRR w/ Data Balancing, CosineEmbeddingLoss, weighted loss	0.512	0.547	0.737
E7: MFRR full data and weighted loss	0.514	0.701	0.844
E8: MFRR w/full data, weighted loss, shared layer	0.510	0.778	0.862

These results are roughly what we expected, MFRR with full data and weighted loss outperforms the Data Balancing counterparts, with a performance increase of roughly 10%, since utilizing the full data would increase performance over just training on a sub-sample. It mainly benefited paraphrase detection, since it was the largest dataset and now saw more training examples, but subsequently that training helped STS improve, given that they both depend on similarity of encodings. This was taken advantage of in the shared layer idea, and therefore the 5% performance increase seen over just the MFRR full data with weighted loss, proving that this is the best model yet.

Due to lack of memory storage, Batch Size of 64 did not train long enough to give fruitful results for any of the extensions. But if we had access to more storage, we believe our results of Full Data for all

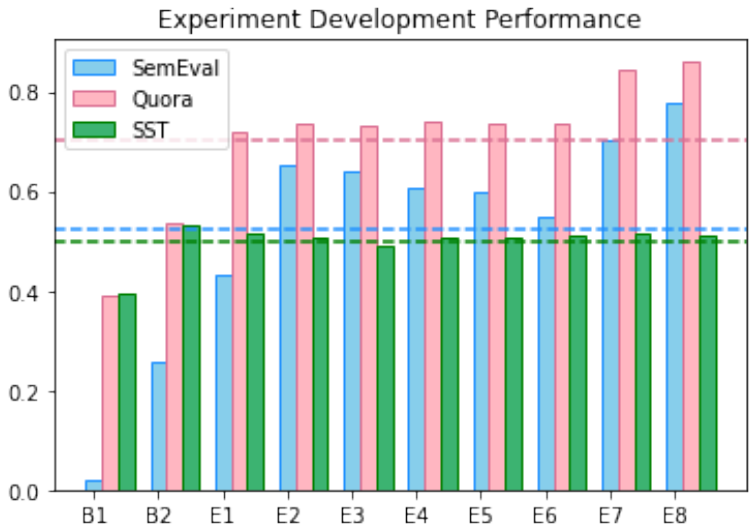


Figure 3: Experimental Performance. Here, each dotted line corresponds to the average of the accuracies for each task across the experiments. The metrics used to compute performance were accuracy for the Quora and SST datasets, and Pearson correlation coefficient for the SemEval dataset.

the extensions would outperform the Data Balancing results, since paraphrase detection would have been able to train longer.

Within the different models of MFRR with Data Balancing, the weighted loss outperforms all the other extensions by roughly 5% because it prevents domination of any one loss and learns the weights of the losses through training. It also outperformed gradient surgery which shows that the gradients of the tasks were not super conflicted to begin with.

We chose our best model of Multitask RR full data with weighted loss and shared layer implementation for evaluation on the test set and got accuracies of 0.526 for sentiment classification, 0.861 for paraphrase detection, and 0.775 for STS.

6 Analysis

6.1 Similarity Task Fine-Tuning

STS and praphrase detection are both tasks engaged with capturing similarity between word vectors. The approaches we implemented to leverage this fact had varying degrees of success. First, our experiments regarding cosine-similarity were two-pronged. In the first approach, we utilized cosine-similarity to predict a similarity score, in conjunction with an MSE loss. This had sound performance on the STS task, which makes sense given that the similarity between two word embeddings is a reasonable proxy for semantic textual similarity. In the second approach, we used the PyTorch CosineEmbeddingLoss as the task loss function, which takes in two sentence embeddings and a label (-1 for dissimilar and 1 for similar). We found that this loss function yielded worse performance for the STS task. In order to utilize a cosine embedding loss function, the STS labels were forced from a continuous scale to a discrete, binary scale. Thus, performing this transformation lost a great amount of nuance for the task, and consequently it makes sense why this loss function would not yield better performance.

Our second approach, using shared architecture for both the STS and paraphrase detection tasks, yielded successful results. Since word vector similarity is an important component for both STS and paraphrase detection, using a shared layer for both tasks allowed the model to leverage information from both datasets and learn across tasks. This phenomenon was hinted at in earlier strategies as well; in general, if we saw increased performance in one of the similarity tasks, the other would benefit as well. Observing this connection between the two tasks, it makes sense that using a shared layer would increase performance for both of the similarity tasks.

6.2 Loss Construction

The approaches we applied to multitask loss construction, including naive sum, weighted sum, and gradient surgery, each had benefits and limitations. The first approach, which was to naively sum each task-specific loss, achieved the lowest performance of the three approaches. However, it is the simplest and least computationally expensive approach. It makes sense that this approach yielded the worst performance because it allowed one loss term to potentially dominate the others, making the model lopsided into one task. We found that the others approaches effectively guarded against this behavior. The weighted sum approach attempts to appropriately weigh the impact of each task on the overall loss, preventing one loss from dominating. Mathematically, this approach is based on using the homoscedastic uncertainty of each task to inform its weight, which allows the task losses to have different units or scales and still be combined in a useful way, as hypothesized in Kendall et al. (2017). Meanwhile, gradient surgery prevents conflicting gradient directions by projecting each task gradient onto the normal plane of another conflicting gradient. Although gradient surgery yielded a small improvement in performance, it also presented challenges with memory allocation; consequently we could only perform it on a batch size of 8. Our preferred loss construction was the weighted loss, as it presented better performance without too much compromise on memory use.

6.3 Sampling Procedures

Our experimentation with dataset sampling also yielded interesting results. In one method, we constrained each dataset to be of equal size, sampling from the larger datasets to do so. The implication of this method was that most data from the Quora paraphrase detection dataset, which was the largest, was not used during training. In another method, we first stipulated a batch size, then used the largest dataset (Quora) to determine the number of batches. The implication of this method was that the batch size of the other two datasets was extremely small, perhaps even 1, and that the number of batches was quite large. Both of these strategies had trade-offs: the first strategy ignored a great amount of data but executed quickly, while the second used more data but was very slow. We found that the second approach yielded higher accuracies, specifically for paraphrase detection. This is consistent with the fact that this strategy uses much more of the dataset available for paraphrase detection. These trade-offs highlight the foundational struggle between performance and efficiency within modeling.

6.4 Ablation Study

We omit SMART algorithm from our experiments and results, since we observed that while training loss increased with the regularization, it did not find optimal results for any of the datasets and both training and dev accuracies took a much larger hit than not performing SMART at all. We think this might be because there are three losses for each task and with three different tasks, taking the gradients of the final loss which has nine items may be too drastic. Because of time constraints, since we could not ensure this behavior was not originating from an implementation bug, we chose to omit SMART from our final results.

7 Conclusion

Through our considerable experimentation, we have found several strategies that have improved the performance of BERT as the basis of a multitask NLP model. We have found that utilizing shared fine-tuning for the tasks of semantic textual similarity (STS) and paraphrase detection improved performance. We have also found that using a weighted loss function that scales each task loss via homoscedastic uncertainty improved performance. These two conclusions highlight the fact that multitask modeling requires both thorough understanding of the individual tasks—and the connections amongst them—as well as an understanding of how to effectively combine individual details into a whole. Additionally, we have discovered trade-offs in performance and efficiency in dataset sampling procedures. Our work highlights the great amount of subjectivity within a multitask learning framework. We had to consider benefits and limitations when faced with many choices for how to fine-tune task objectives, how to construct a multitask loss, and how to sample and traverse the datasets. Future avenues of work may examine the relationships between different metrics of a successful model: performance, speed, explainability. Our study presents several techniques that inform this work to forge more capable multitask NLP models in the future.

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 of the Association for Computational Linguistics: ACL 2022*, pages 2663–2669.

Winter 2023 CS224N. 2023. Default final project - multitask bert starter code.

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

Shankar Iyer, Nikhil Dandekar, and Kornél Csernai. First quora datase release question pairs.

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.

Alex Kendall, Yarin Gal, and Roberto Cipolla. 2017. Multi-task learning using uncertainty to weigh losses for scene geometry and semantics. *CoRR*, abs/1705.07115.

Nina Khairova, Anastasiia Shapovalova, Orken Mamyrbayev, Nataliia Sharonova, and Kuralay Mukhsina. 2022. Using bert model to identify sentences paraphrase in the news corpus. In *CEUR Workshop Proceedings*, volume 3171, pages 38–48.

Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3982–3992, Hong Kong, China. Association for Computational Linguistics.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642.

Staff. Cs 224n: Default final project: minbert and downstream tasks.

Asa Cooper Stickland and Iain Murray. 2019. BERT and PALs: Projected attention layers for efficient adaptation in multi-task learning. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 5986–5995. PMLR.

Chi Sun, Xipeng Qiu, Yige Xu, and Xuanjing Huang. 2019. How to fine-tune BERT for text classification? *CoRR*, abs/1905.05583.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need.

Sida I Wang and Christopher D Manning. 2012. Baselines and bigrams: Simple, good sentiment and topic classification. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 90–94.

Tianhe Yu, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. 2020. Gradient surgery for multi-task learning. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS'20, Red Hook, NY, USA. Curran Associates Inc.

Xiang Zhang and Yann LeCun. 2015. Text understanding from scratch. *arXiv preprint arXiv:1502.01710*.

A Appendix

A.1 Hyperparameter tuning

Here are the hyperparameters that resulted in our best models:

Table 3: Optimal hyperparamters that gave the best results for each model

Model	Batch Size	Learning Rate	Weight Decay	Dropout	Lambda
MFRR w/ Data Balancing and Naive loss	16	1e-5	0.00	0.3	-
MFRR w/ Data Balancing and weighted loss	8	1e-5	0.00	0.3	-
MFRR w/ Data Balancing and gradient surgery	8	1e-5	0.00	0.3	-
MFRR w/ Data Balancing, L1 Regularisation, weighted loss	16	1e-5	0.00	0.3	0.0001
MFRR w/ Data Balancing, L2 Regularisation, weighted loss	16	1e-5	0.00	0.3	0.001
MFRR w/ Data Balancing, CosineEmbeddingLoss, weighted loss	16	1e-5	0.00	0.3	-
MFRR w/ full data and weighted loss	32	1e-5	0.01	0.3	-
MFRR w/ full data, weighted loss, shared layer	32	1e-5	0.00	0.3	-

A.2 Siamese Bert Structure

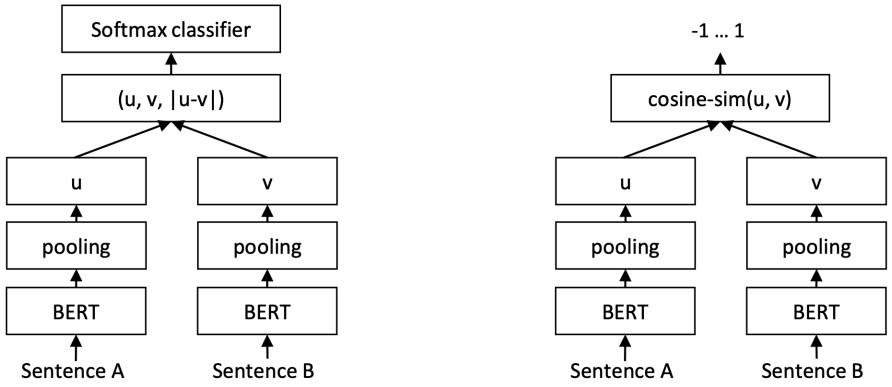
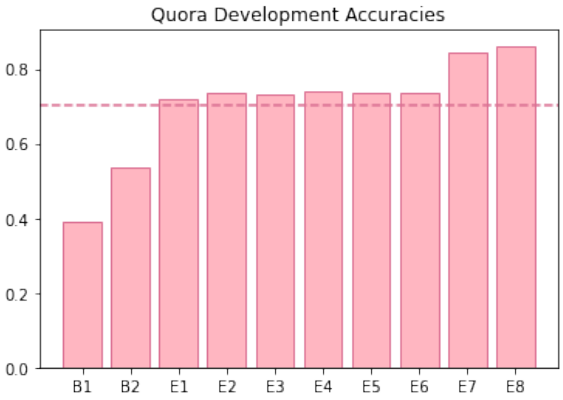


Figure 4: (a) Siamese BERT Structure for paraphrase detection. (b) Siamese BERT Structure for STS

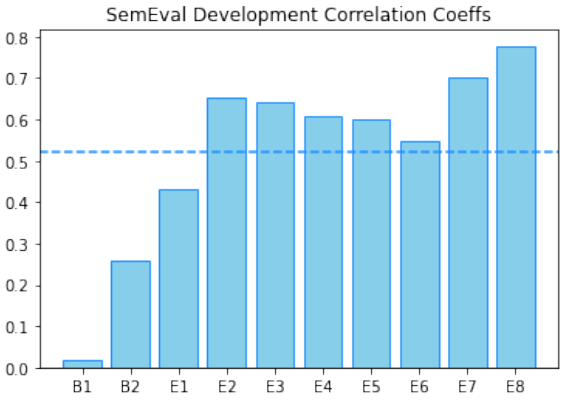
A.3 Experiment Accuracies by Task

A.4 Baseline Bert Structure



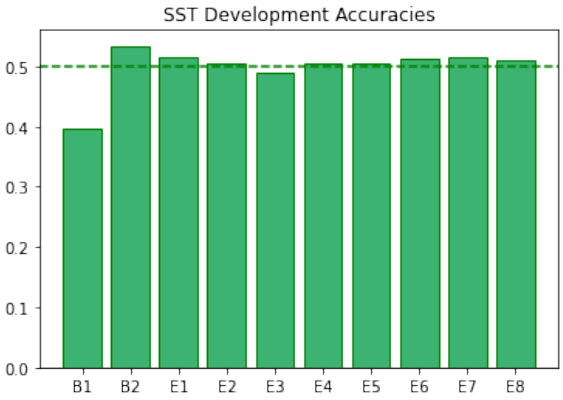
B1	Baseline 1 (Pre-Trained BERT)
B2	Baseline 2 (Fine-Tuned BERT)
E1	MFRR w/ Data Balancing and Naive loss
E2	MFRR w/ Data Balancing and weighted loss
E3	MFRR w/ Data Balancing and gradient surgery
E4	MFRR w/ Data Balancing, L1 Regularisation, weighted loss
E5	MFRR w/ Data Balancing, L2 Regularisation, weighted loss
E6	MFRR w/ Data Balancing, CosineEmbeddingLoss, weighted loss
E7	MFRR w/ full data and weighted loss
E8	MFRR w/ full data, weighted loss, shared layer

Figure 5: Paraphrase Detection Accuracies. Here, the dotted line corresponds to the average of the accuracies across the experiments.



B1	Baseline 1 (Pre-Trained BERT)
B2	Baseline 2 (Fine-Tuned BERT)
E1	MFRR w/ Data Balancing and Naive loss
E2	MFRR w/ Data Balancing and weighted loss
E3	MFRR w/ Data Balancing and gradient surgery
E4	MFRR w/ Data Balancing, L1 Regularisation, weighted loss
E5	MFRR w/ Data Balancing, L2 Regularisation, weighted loss
E6	MFRR w/ Data Balancing, CosineEmbeddingLoss, weighted loss
E7	MFRR w/ full data and weighted loss
E8	MFRR w/ full data, weighted loss, shared layer

Figure 6: STS Accuracies. Here, the dotted line corresponds to the average of the Pearson correlation coefficients across the experiments.



B1	Baseline 1 (Pre-Trained BERT)
B2	Baseline 2 (Fine-Tuned BERT)
E1	MFRR w/ Data Balancing and Naive loss
E2	MFRR w/ Data Balancing and weighted loss
E3	MFRR w/ Data Balancing and gradient surgery
E4	MFRR w/ Data Balancing, L1 Regularisation, weighted loss
E5	MFRR w/ Data Balancing, L2 Regularisation, weighted loss
E6	MFRR w/ Data Balancing, CosineEmbeddingLoss, weighted loss
E7	MFRR w/full data and weighted loss
E8	MFRR w/full data, weighted loss, shared layer

Figure 7: Sentiment Classification Accuracies. Here, the dotted line corresponds to the average of the accuracies across the experiments.

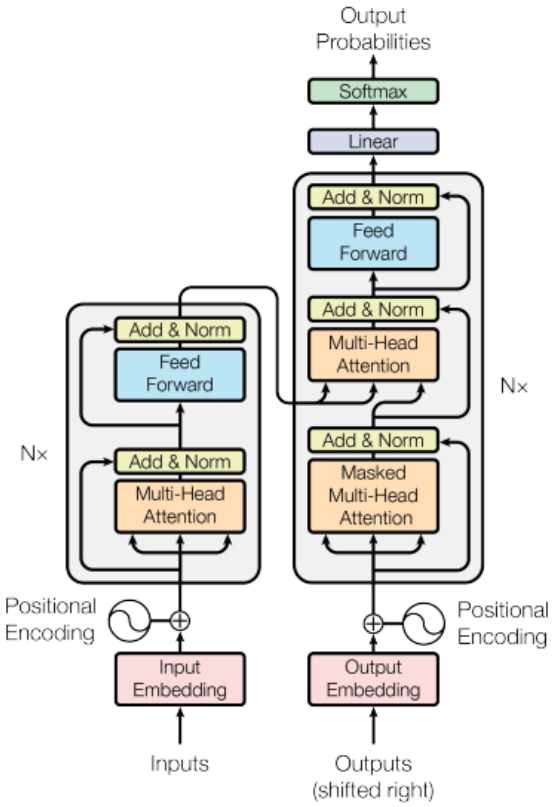


Figure 8: Transformer Model Architecture