Comparing BERT Fine-Tuning Methods

Stanford CS224N Default Project

Adrian Stoll
Stanford University
adrs@stanford.edu

Jennifer Ho
Stanford University
jenph@stanford.edu

Daniel Tyshler
Stanford University
dtyshler@stanford.edu

Abstract

This project compares methods for adapting BERT to downstream NLP tasks including sentiment analysis, paraphrase detection, and semantic text similarity. The goal is to identify techniques that generalize well while fitting within our computational budget. Our baselines are models with frozen BERT weights and task specific additional layers. We compare tuning methods for the pre-trained BERT layers including: full fine-tuning; parameter-efficient Low-Rank Adaption (LoRA); and variants of LoRA including DoRA and Rank-stabilized LoRA (rsLoRA). We find full-fine tuning is most effective for all the tasks. For paraphrase and similarity tasks the input consists of a text pairs. We compare Siamese networks with concatenating the text pairs into a single token sequence with token type embeddings and find the latter generalizes better with less overfitting. We experimented with multi-task training and opted to stick with single task training. This is because 1) single task experimentation is faster and 2) the mutli-task models overfit tasks with less data while underfitting tasks with more data. By combining the best experimental results we obtain high test leaderboard scores.

1 Key Information to include

- Mentor: Arvind Mahankali (amahanka@stanford.edu)
- External Collaborators: None,
- Sharing project: No

1.1 Contributions

- Adrian: Baseline model architecture, multi-task dataloader, training and experiment logging script, LoRA experiments, and Siamese Network vs Concatenation experiments
- Daniel: Learning and batch size experiments, and multi-task training experiments, alternate loss functions, final training run
- Jennifer: Learning rate schedule, training duration experiment, dropout experiment, regularization and associated experiment, final training run

2 Introduction

Our goal is to identify what fine-tuning methods provide the best performance within a reasonable training budget on the downstream semantic analysis, paraphrase detection, and semantic text similarity tasks. Fine-tuning foundation models is fundamental to building a wide range of applications where there may not be enough data, compute resources, expertise, or time to build a quality model from scratch. The current paradigm is to use large-scale unsupervised pre-training to create a foundation model, add task specific adaptor layers, and fine-tune the model with a task specific corpus Devlin et al. (2019). Pre-trained foundation models have been shown to learn zero-shot abilities such as language translation and summarizing during the pre-training process Radford et al. (2019). The motivation for fine-tuning pre-trained foundation models is the knowledge of language gained during pre-training will transfer and improve performance on downstream tasks.

The difficulty of this problem is finding techniques that are effective across heterogeneous tasks - such as classification and regression, varying data sizes, and different input types - such as single or multiple inputs. While fine-tuning is vastly faster and cheaper than pre-training, the training time and cost are still non-negligible.

This motivates our approach to use parameter-efficient fine-tuning techniques, such as LoRA Hu et al. (2021), to balance performance improvements with computational efficiency. Because there are multiple tasks we also explore jointly training a multi-task model to improve the performance on each task by transferring knowledge from the other tasks.

3 Related Work

"LoRA: Low-Rank Adaptation of Large Language Models" by Hu et al. (2021) introduces a method that injects trainable rank decomposition matrices into each Transformer layer of a frozen pre-trained model, significantly reducing the number of trainable parameters and GPU memory requirements. This approach maintains model performance on benchmarks like RoBERTa, DeBERTa, GPT-2, and GPT-3 while improving training efficiency and eliminating additional inference latency. LoRA's efficiency makes it practical for adapting large-scale language models to specific tasks with lower computational and memory costs.

We also experiment with variants of LoRA such as DoRA and Rank-Stabilized LoRA. "DoRA: Weight-Decomposed Low-Rank Adaptation" by Liu et al. (2024) proposes a novel parameter-efficient fine-tuning method. DoRA introduces a weight decomposition analysis to understand the differences between full fine-tuning (FT) and LoRA, decomposing pre-trained weights into magnitude and direction components for efficient updates. This method enhances the learning capacity and stability of LoRA, consistently outperforming it on tasks like commonsense reasoning and visual instruction tuning while avoiding additional inference overhead.

Additionally, we followed "A Recipe for Training Neural Networks," by Karpathy (2019) which describes how to train incrementally more sophisticated models to make debugging easier. The steps the articles outlines are: looking at the data, setting up a simple training pipeline and baseline, overfitting on a small batch of data, overfitting on the full dataset, adding regularization, and tuning hyperpaparameters.

"Siamese Neural Networks for One-shot Image Recognition" by Zoch et al. (2015) and "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks" Reimers and Gurevych (2019) present a method using Siamese neural networks to effectively learn features for one-shot learning. By training the network to discriminate between pairs of images, the model achieves a high performance on classification tasks with limited data. In our work we evaluate Siamese networks for the paraphrase and similarity tasks.

"How to Fine-Tune BERT for Text Classification?" Sun et al. (2020) is highly relevant to our project. It explores fine-tuning strategies for BERT on text classification, offering a general solution and investigating methods like layer selection, layer-wise learning rates, and addressing catastrophic forgetting. Their insights on within-task and in-domain pre-training, as well as multi-task fine-tuning, can help optimize our model for sentiment analysis, paraphrase detection, and semantic text similarity, addressing issues of overfitting and data imbalance we encountered in our multitask training.

4 Approach

4.1 Model Architectures

All model architecture use pre-trained BERT encoders Devlin et al. (2019) and multi-layer perceptron (MLP) as building blocks. The models use the BERT encoder to produce a CLS token embedding from their input. For paraphrase and similarity tasks the input consists of a pair of strings. The Siamese network model architecture encodes each input separately before concatenating representations while the input concatenation model architecture encodes both inputs together as part of a single token sequence before applying BERT.

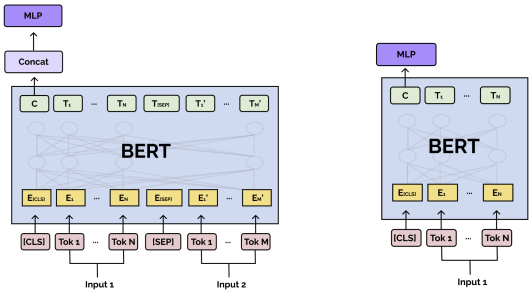
Following the BERT layer are task-specific adaptors. For single-task models there is an adaptor for a single task while for multi-task models there are multiple adaptors for each tasks. The adaptors are MLPs with a sequence of hidden layers each applying dropout Hinton et al. (2012) to their input and using ReLU as the non-linearity. The output layer includes dropout but lacks a non-linear activation function. For sentiment analysis the output layer produces logits for five classes, for paraphrase the model produces logits for two classes, and for similarity the layer outputs a single scalar.

We considered the following families of model architectures:

- **Single-Task**: The paraphrase/similarity model pictured in Figure 1a is using the Input Concatenated model architecture.
- **Multi-Task**: The multi-task models have adaptor MLP layers for each task. Each task uses a subset of the layers of the multi-task model indicated by the dotted lines in Figure 2a. Tasks transfer learn from each other by sharing the same pre-trained BERT layers. The multi-task architecture can support 1, 2, or 3 tasks and is a generalization of the Single-Task architecture.

- **Siamese-Network**: The Siamese network model architecture Reimers and Gurevych (2019) applies the fine-tuned BERT model to both input strings in isolation to produce a fixed size representation. The vector representations for both inputs are concatenated together and fed into a multi-layer perceptron. We choose this architecture as a baseline because it was the easiest to implement. The limitation of this model is the attention layers only compare tokens within each input not between the inputs.

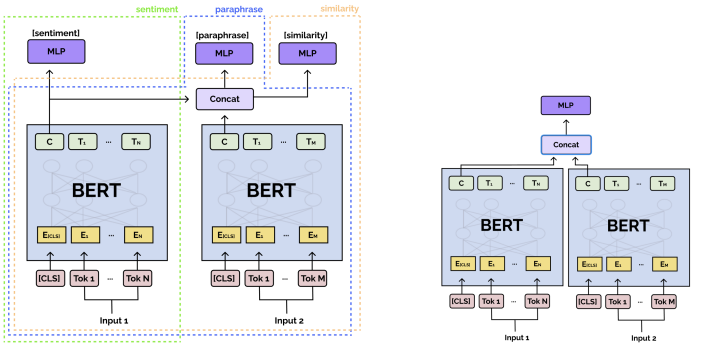
- **Input Concatenated**: We hypothesize that inter-input attention is useful for paraphrase detection and similarity tasks. This model architecture concatenates the input pairs with token type embeddings to distinguish which input each token is from and feeds the concatenation through multiple transformer layers in the BERT module. This approach allows the model to match words or phrases in one input with those in the other using the attention mechanism to effectively compare the inputs.



(a) Similarity and Paraphrase Model

(b) Sentiment Models

Figure 1: Single Task Model Architectures.



(a) Siamese Multi-Task Model Architecture

(b) Siamese Paraphrase and Similarity Model Architectures

4.2 Objective Functions

For the sentiment analysis and paraphrase detection tasks we use the cross-entropy loss function. For semantic text similarity we use mean-squared-error. The final evaluation metric for the similarity task is the Pearson Correlation between the predicted and ground truth similarity scores. We considered using the Pearson Correlation between the predictions in a batch with the ground truth as a training objective. We opted not to use this objective due to concerns the mean predicted similarity scores used to compute the correlation would be too noisy for small batch sizes. We implemented contrastive-loss Gao et al. (2022) and cosine-similarity Reimers and Gurevych (2019). These alternative loss functions did not provide an improvement, however that may have been because we had insufficient time to debug the implementations.

4.3 Training

We trained all the models by shuffling the data and iterating over it in mini-batches. For optimizing model parameters we used AdamW Loshchilov and Hutter (2019). For optimizing hyperaparameters we ran ad-hoc experiments and used grid search. Unless noted otherwise we used the following hyperaparameter values:

- AdamW: $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$
- Regularization: weight decay = 0.01, dropout probability = 0.3
- Epochs: 10 for sentiment and similarity tasks, 2 for paraphrase
- Number of MLP hidden layers: 0

For multi-task models, we implement a round-robin training approach which cycles through batches from each task. The training loop exhausts available data for tasks with smaller datasets before tasks with larger datasets. When this occurs, we start over from the beginning of the smaller dataset, allowing training to continue uninterrupted. This method ensures that training proceeds smoothly across all tasks by cyclically resetting and iterating through the datasets, thereby maintaining consistent exposure to each task throughout the model training process to prevent forgetting.

To assist the experimentation process, we set up a structured testing system to automate the configuration and execution of our experiments. It allows us to define various experimental setups, run, and log results. The existing code we used is the project starter code and the Weights and Biases experiment tracking library Biewald (2020). The original code we wrote is the task specific adaptors, training loop, experiment setup, and grid search logic.

5 Experiments

5.1 Data

For Sentiment Analysis, we use the Stanford Sentiment Treebank (SST) for fine-grained sentiment classification. For Paraphrase Detection, we use the Quora Question Pairs dataset to train models on identifying semantic equivalence, and for Semantic Text Similarity (STS), we utilize the SemEval dataset to train models on rating the degree of semantic similarity between texts.

5.2 Evaluation Method

We use the following evaluation metrics for each task: Accuracy for Sentiment Analysis and Paraphrase Detection and Pearson correlation coefficient between predicted and actual similarity scores for Semantic Text Similarity. See Proposal for more details. During the training we monitoring the training loss for each batch at the end of each epoch evaluate the task evaluation metric on the training and dev datasets.

5.3 Fine-Tuning

We experimented with the following fine-tuning methods: Full fine-tuning of the pre-trained parameters; and LoRA, DoRA, and Rank-Stabilized parameter efficient fine-tuning methods. We compare these methods with the baseline of only training task specific layers MLP layers.

We used the following LoRA specific hyperaparameters as defaults for the following experiments unless noted otherwise. LoRA rank = 8, LoRA dropout probability = 0.1, and target modules = "query","key", "value","attention_dense","interm_dense","out_dense".

5.3.1 LoRA Variants

In these experiment we compare variants of LoRA on each of the tasks. We observe rsLoRA is the LoRA variant with the highest training and dev scores with the lowest runtime of the methods. Additionally the table shows DoRA nearly doubles the training time when number of epochs is held constant. Because LoRA family models have fewer trainable parameters than full fine-tuning, we observe less overfitting. Because LoRA family models have more parameters and model capacity than the baseline frozen-weight BERT model, we observe less underfitting.

5.3.2 Target Modules

LoRA and related parameter efficient fine-tuning techniques can be applied to any subset of the weight matrices in a network. The table below compares accuracy, training time, and peak GPU memory usage when using LoRA

Task	Architecture	Fine-Tuning	Dev Score	Train Score	Training Time	Peak GPU Memory
Sentiment		LoRA	0.4541	0.4878	17m 32s	**2.59 GiB**
Sentiment		DoRA	0.4569	0.4884	32m 34s	4.2 GiB
Sentiment		rsLoRA	0.4777	0.5202	**17m 31s**	2.62 GiB
Sentiment		Full	**0.5204**	**0.954**	18m 45s	3.02 GiB
Paraphrase	Siamese	LoRA	0.7375	0.7377	3h 3m	12.19 GiB
Paraphrase	Siamese	DoRA	0.744	0.7454	5h 25m	9.30 GiB
Paraphrase	Siamese	rsLoRA	0.7523	0.7572	2h 43m	**5.99 GiB**
Paraphrase	Concatenate	LoRA	0.8406	0.8444	2h 23m	6.47 GiB
Paraphrase	Concatenate	DoRA	0.8408	0.845	4h 30m	9.76 GiB
Paraphrase	Concatenate	rsLoRA	0.8532	0.8598	**2h 22m**	6.47 GiB
Paraphrase	Concatenate	Full	**0.8974**	**0.9488**	2h 41m	6.48 GiB
Similarity	Siamese	LoRA	0.2532	0.2861	17m 31s	**3.74 GiB**
Similarity	Siamese	DoRA	0.2559	0.2911	33m 4s	6.28 GiB
Similarity	Siamese	rsLoRA	0.3071	0.3485	17m 30s	**3.74 GiB**
Similarity	Concatenate	LoRA	0.8051	0.8286	16m 31s	4.03 GiB
Similarity	Concatenate	DoRA	0.8052	0.8298	30m 3s	6.53 GiB
Similarity	Concatenate	rsLoRA	0.8152	0.842	**16m 25s**	4.03 GiB
Similarity	Concatenate	Full	**0.8746**	**0.986**	18m 11s	4.24 GiB

Table 1: Comparison of fine-tuning methods for single-task model architectures. The best evaluation metrics for each task are in bold.

to adapt specific weight matricies. The more modules LoRA adapts the greater the computational requirements and runtime. The original LoRA paper only evaluated adapting the attention weights while we additionally evaluate the adapting the dense linear layer weights. We note adapting only the attention weights saves almost twice as much memory as adapting only the dense layers - although it does have 1-2 percentage points lower accuracy.

Target Modules	Dev Score	Train Score	Training Time	Peak GPU Memory
all	**0.4541**	0.4878	17m 32s	2.62 GiB
all dense	0.445	**0.4881**	16m 20s	2.24 GiB
query, value	0.4378	0.4654	**15m**	**1.82 GiB**
query, key, value	0.426	0.4589	15m 25s	1.94 GiB

Table 2: This table compares tuning different network weight matrices with LoRA for the sentiment task.

5.3.3 LoRA Rank

LoRA tunes model weight matricies by adding a low-rank delta matrix. The higher the rank, the closer LoRA behaves to full fine-tuning. In this experiment we vary the LoRA rank and compare performance metrics. We observe that changing the rank has limited impact the accuracy scores and peak memory usage. Using a lower rank does reduce the training time but not by much. We conclude that choice of PEFT technique and target modules are more impactful parameters to tune.

LoRA Rank	Dev Score	Train Score	Training Time	Peak GPU Memory
32	0.4614	0.4881	17m 56s	2.69 GiB
16	0.4469	0.4959	17m 37s	2.64 GiB
8	0.4541	0.4878	17m 31s	2.62 GiB
4	0.4523	0.4974	17m 23s	2.61 GiB
2	0.4369	0.4844	17m 19s	2.61 GiB
1	0.455	0.5056	16m 46s	2.61 GiB

Table 3: This table compares different LoRA update ranks for the sentiment task.

To maximize performance we decided to stick with full fine-tuning in the final model at the cost of higher training time and memory usage.

5.4 Multi-task Training

In our experiments, multitask training yielded higher training scores but suffered from overfitting, particularly for tasks with less data, such as the Sentiment and Similarity tasks. This overfitting resulted from the model seeing examples from smaller datasets more frequently. Additionally, multitask runs were conducted with a fixed learning rate of 0.0001, which may not have been optimal, contributing to poorer performance. In contrast, single-task runs, which included a hyperparameter sweep for learning rates, showed better balance between training and dev scores, indicating they did not suffer from overfitting. Future work should address these issues by using task specific learning rates and regularization during multi-task training to mitigate data imbalance.

Task	Loss	Final Dev Score	Final Train Score	Training Time
Paraphrase, Similarity	1.6	Paraphrase: 0.7841 Similarity: 0.3871	Paraphrase: 0.8036 Similarity: 0.9925	2h 40m 53
Sentiment, Paraphrase	1.585	Sentiment: 0.4914 Paraphrase: 0.782	Sentiment: 0.9973 Paraphrase: 0.8041	2h 24m 22
Sentiment, Similarity	1.56	Sentiment: 0.4523 Similarity: 0.3623	Sentiment: 0.4698 Similarity: 0.4483	5m 4s
Sentiment, Similarity, Parphrase	1.485	Sentiment: 0.4977 Similarity: 0.3767 Paraphrase: 0.7762	Sentiment: 0.9947 Similarity: 0.9909 Paraphrase: 0.793	3h 25m 5s

Table 4: Result summary of the multi-task experiements, learning rate 0.0001, full finetuning, Siamese Networks

5.5 Siamese Networks vs Input Concatenation

To test the hypothesis that inter-input attention is useful for paraphrase detection and similarity tasks, we compare the baseline Siamese network with concatenating the input pairs with token type embeddings to distinguish which input each token is from and feeding the concatenation through multiple transformer layers. The experiment uses the same hyper-parameters for both architectures described in the training section. Below is a table summarizing the results.

Task	Architecture	Dev Score	Train Score	Training Time
Paraphrase	Siamese	0.7963	0.7972	3h 10m
Paraphrase	Concatenation	**0.893**	**0.8925**	**2h 51m**
Similarity	Siamese	0.3324	0.9425	**18m 39s**
Similarity	Concatenation	**0.8721**	**0.986**	18m 46s

Table 5: Comparison of Siamese and Concatenation models for for paraphrase and similarity tasks.

Notice how the Siamese similarity model dramatically overfits to the training data and even then does not achieves as high a training score as the input-concatenation model. This shows the input-concatentation model has lower variance and greater generalization.

Interestingly the runtimes of both architectures are approximately the same. Because the number of operations attention performs is quadratic in the sequence length, applying attention to concatenation of both inputs is twice the number of operations as applying attention twice to each input individually. Specifically $(2n)^2 = 4n^2$ vs $2 * n^2$ operations for the former vs the later. The extra attention lookups with queries from one input's tokens and keys from the others are computed in parallel the within-input attention lookups. In practice we observe the Siamese network is slighly slower possibly because the model applies the BERT layers to the input pairs serially.

5.6 Batch Size

To identify appropriate batch sizes for each task, we run the following experiment: for each task we started with batch size 8 and trained the a model for a few batches and checked the output of nvidia-smi NVIDIA (2016) to monitor the GPU VRAM and how many threads where executing a kernel. Once the batch size was large enough to reach $\approx 90\%$ utilization for either resource we stopped increasing batch the size. We run experiments on an NVIDIA T4 GPU with 16GB of RAM NVIDIA (2019). Based on the experiments we selected batch size 32 for all of the tasks.

5.7 Learning Rate

This experiment tested different learning rate on training loss. We aimed to pinpoint when changes in the learning rate result in significant performance improvements, particularly the benefits of reducing the learning

Learning Rate	Task	Loss	Final Dev Score	Final Train Score	Training Time
0.0001	Sentiment	1.6	0.2997	0.3179	10m 43s
0.0003	Sentiment	1.56	0.3497	0.3653	**10m 14s**
0.003	Sentiment	**1.485**	**0.3906**	**0.4197**	10m 22s
0.001	Sentiment	1.491	0.3688	0.3971	10m 26s
0.01	Sentiment	1.585	0.3742	0.4037	10m 15s
0.0003	Similarity	1.253	0.2481	0.2623	10m 55s
0.003	Similarity	**1.41**	**0.2735**	**0.3013**	10m 46s
0.001	Similarity	1.234	0.2655	0.2961	10m 51s
0.01	Similarity	1.527	0.2675	0.2984	**10m 29s**
0.0003	Paraphrase	**0.6385**	**0.6485**	**0.6465**	1h 55m 1s
0.003	Paraphrase	0.6309	0.5522	0.5512	1h 55m 12s
0.001	Paraphrase	0.7083	0.4852	0.4858	**1h 54m 53s**

Table 6: Result summary of the hyper-parameter sweep. The configurations with the highest dev scores are in bold.

rate towards the end of the training phase. We did a hyperaparameter sweep for learning rate with the range of values 0.0001, 0.0003, 0.001, 0.003, 0.01 and observed the models learned noticeably less for the smallest learning rate.

5.8 Regularization

When the model is trained with too little regularization it will overfit. When it is trained with too much it will underfit. To find the approximately optimal abount of regularization we ran a hyperparameter sweep with increasing levels of weight decay until the dev scores peaked and started decreasing. Because the paraphrase task has 30x more training data than the similarity and sentiment tasks and we only trained for a few epochs, we did not observe overfitting to be a problem and did not experiment with weight decay for paraphrase.

Task	Weight Decay	Dev Score	Train Score
Sentiment	1	0.5241	0.9346
Sentiment	0.3	0.5223	0.9568
Sentiment	**0.1**	**0.5268**	**0.9484**
Sentiment	0.05	0.5204	0.9428
Similarity	4.5	0.8804	0.9848
Similarity	**4**	**0.8807**	**0.9852**
Similarity	3	0.8798	0.9854
Similarity	1	0.8775	0.9859

Table 7: Dev scores for varying levels of weight decay.

5.9 Other experiments

Hidden Layers Experiment: We conducted a hyper-parameter sweep for the number of hidden layers in the task specific MLP and found that adding hidden layers made the model overfit more. The BERT layers already have enough model capacity to overfit the data, so we decided to omit hidden task-specific layers from the rest of the experiments.

Dropout Experiment: Additionally, we investigated the impact of varying dropout rates within the range of 0.1 to 0.6 for each task. The differences in performance were negligible. This outcome suggests that, within the scope of our experiments, the model's existing regularization strategies are already sufficient or that the task-specific characteristics do not heavily depend on dropout for achieving optimal performance.

Epoch Experiment: Lastly, we trained each task for a different number of epochs and observed their graphs to determine where the dev score peaked. Generally, there was not any significant improvement from our default 10 epochs for similarity and sentiment and 2 for paraphrase (due to the abundance of data). However, we did notice the highest scores would occur around epoch 11 for similarity and around epoch 3 for paraphrase (with lower scores in neighboring epochs 1, 2, and 4).

5.10 Best Models

In our study, the best models trained for 10 epochs (4 for paraphrase). The accompanying table documents the specific epoch during which the highest development score was recorded for each model. This data is indicative

of the correlation between the number of epochs and peak model performance. Additionally, all of our best models were subject to a 0.3 dropout probability.

Type	Task	Dev Score	Train Score	Weight Decay	Finetune	Epoch	Architecture
Best	Sentiment	0.5268	0.9484	0.1	Full	3	Single-Task
Baseline	Sentiment	0.3906	0.4197	0.01	N/A	10	Single-Task
Best	Similarity	0.8816	0.9923	4	Full	11	Single-Task Concatenation
Baseline	Similarity	0.2735	0.3013	0.01	N/A	10	Single-Task Siamese
Best	Paraphrase	0.9004	0.9789	0.01	Full	3	Single-Task Concatenation
Baseline	Paraphrase	0.6485	0.6465	0.01	N/A	2	Single-Task Siamese

Table 8: Scores and hyperaparameters for the best models and the baselines.

5.10.1 Test Scores

- SST test accuracy: 0.535
- Paraphrase test accuracy: 0.902
- STS test correlation: 0.870
- Overall test score: 0.791

6 Analysis

Paraphrase

True Positives (TP): 13494 (33.38%)
True Negatives (TN): 22610 (55.93%)
False Positives (FP): 2927 (7.24%)
False Negatives (FN): 1398 (3.46%)

In this project's evaluation of BERT fine-tuning methods for paraphrase detection, full fine-tuning proved most effective, consistently achieving the highest development and training scores across various experiments. Despite its greater computational demand, full fine-tuning surpasses parameter-efficient methods like LoRA, DoRA, and rsLoRA by better utilizing BERT's full representational capacity to discern nuanced textual differences and similarities crucial for paraphrase detection. Extensive tests revealed that while parameter-efficient techniques generally minimize overfitting, they fall short in accuracy and robustness compared to full fine-tuning, especially in complex tasks that benefit from deep model tuning. Moreover, the focus on single-task training over multi-task setups allowed for more precise model adjustments, enhancing performance and pointing to the necessity of choosing fine-tuning strategies that match specific task demands and data characteristics.

Sentiment

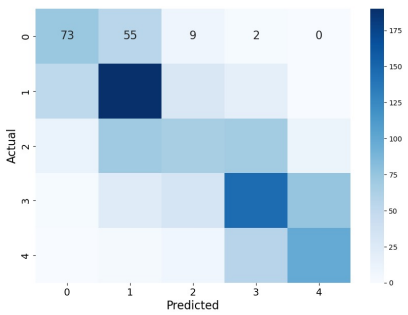


Figure 2: Sentiment Confusion Matrix

The confusion matrix for sentiment analysis shows the most common errors are predicting the sentiment label adjacent to the true class (e.g. 4 stars instead of 5 stars for a movie review), suggesting minor inaccuracies rather than significant misunderstandings. This shows that even when the predictions are incorrect they are still somewhat reasonable, indicating a nuanced understanding of sentiment gradations by the model.

From the confusion matrix we also see classes 1 and 3 are more frequent in the dataset than the other classes. The prevalence of errors in specific classes points to potential imbalances in the dataset that might affect the model's learning.

Moreover, similar to paraphrase, full fine-tuning surpassed other methods like LoRA and DoRA in the sentiment analysis task, demonstrating its superior ability to utilize BERT's full representational capacity.

Sentiment We observed our most significant improvement in sentiment model accuracy upon implementing regularization, particularly through weight decay. This approach effectively mitigated overfitting, enabling the

model to generalize better across unseen data. The incorporation of weight decay into our training regimen was crucial in refining the model's performance, highlighting its importance as a regularization strategy in deep learning models. Overall, the application of weight decay not only boosted accuracy but also contributed to a more stable and consistent model performance across various test scenarios.

7 Conclusion

Our main findings indicate that the concatenation approach outperforms Siamese networks due to reduced overfitting and the cross-input connections. Parameter-efficient fine-tuning methods such as LoRA and its variants modestly decrease both runtime and accuracy, with rank-stabilized LoRA (rsLoRA) emerging as the most effective variant. These methods are less prone to overfitting due to having fewer trainable parameters. Despite the benefits of LoRA and its variants, full fine-tuning proved more suitable for our project as the additional runtime was minimal and resulted in a slight performance improvement.

However, our study has limitations. Specifically, training for more epochs could potentially enhance accuracy further. Additionally, we faced challenges with multi-task training, primarily due to overfitting in tasks with limited data. This prevented us from obtaining a robust multi-task model. Future work should address these limitations by optimizing the training duration and refining techniques to manage data imbalance in multi-task learning scenarios.

8 Ethical Considerations

8.1 Biases In The Data

As with any model of this type, their is the risk of biases in the data affecting how the model performs and treats different subsections of people. For example, if the training data contains more or less representations of specific dialects, for example, it might not represent colloquialism in British English or Indian English. For instance in British English "peak" can mean both terrible and amazing, depending on context. However, a model trained on American English may not pick up on the fact that it could mean both of those. Another example would be the model not picking up on certain phrases or expressions that might mean different things in different regions. A classic example is the word for a sweet carbonated beverage, which can also be called "soda", "pop", "cola", "soda pop", etc. A similarity or paraphrase model may be less accurate than it could be if it does not have all of these variants in its training data.

8.2 Transparency

Another common issue with AI models, including this one, is transparency. Users often find it challenging to understand what data and techniques have been used to develop the model unless the creators provide detailed information. To address this, comprehensive documentation of the model's architecture, training process, and decision-making criteria should be provided. This includes a clear explanation of the data sources, pre-processing steps, and the rationale behind the chosen algorithms. Additionally, developing user-friendly interfaces that offer straightforward explanations of the model's predictions can help foster transparency and trust. For instance, using techniques like attention visualization can show users which parts of the input data the model focuses on, making the decision-making process more understandable. Regular updates and open channels for feedback can also enhance transparency and ensure the model remains aligned with user needs and ethical standards.

References

Lukas Biewald. 2020. Experiment tracking with weights and biases. `https://www.wandb.com/`. Software available from wandb.com.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding.

Tianyu Gao, Xingcheng Yao, and Danqi Chen. 2022. Simcse: Simple contrastive learning of sentence embeddings.

Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R. Salakhutdinov. 2012. Improving neural networks by preventing co-adaptation of feature detectors.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models.

Andrej Karpathy. 2019. A recipe for training neural networks. `https://karpathy.github.io/2019/04/25/recipe/`.

Shih-Yang Liu, Chien-Yi Wang, Hongxu Yin, Pavlo Molchanov, Yu-Chiang Frank Wang, Kwang-Ting Cheng, and Min-Hung Chen. 2024. Dora: Weight-decomposed low-rank adaptation.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization.

NVIDIA. 2016. Nvidia system management interface program. https://developer.download.nvidia.com/compute/DCGM/docs/nvidia-smi-367.38.pdf. Accessed: May 23, 2024.

NVIDIA. 2019. Nvidia t4 tensor core gpu. https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/tesla-t4/t4-tensor-core-datasheet-951643.pdf. Accessed: May 23, 2024.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners.

Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks.

Chi Sun, Xipeng Qiu, Yige Xu, and Xuanjing Huang. 2020. How to fine-tune bert for text classification? https://arxiv.org/pdf/1905.05583.

Zoch, Kemel, and Salakhutdinov. 2015. Siamese neural networks for one-shot image recognition. https://www.cs.cmu.edu/~rsalakhu/papers/oneshot1.pdf.

A Appendix (optional)