AllBERT: Mastering Multiple-Tasks Efficiently

Thierry Rietsch  
Stanford University  
rietsch@stanford.edu

Joe Serrano  
Stanford University  
jserran0@stanford.edu

Abstract

We present AllBERT, a set of versatile multi-task models designed to deliver top performance on three natural language processing (NLP) tasks: sentiment classification, paraphrase detection, and semantic textual similarity. Starting from the BERT$_{Base}$ model, we made architectural modifications, then rigorously trained the model by implementing regularization methods such as SMART and LNSR to combat overfitting. We also introduced an original loss function for sentiment analysis that outperformed traditional approaches. To address the increase in parameters from single-task models, we integrated low-rank techniques such as DoRA for efficient fine-tuning. We also created a single model variant trained on multiple tasks, and found greater success in ensembling these variants.

1 Introduction

Multi-task learning (MTL) models in NLP improve performance and generalization by sharing information across related tasks, especially when labeled data is scarce. Training on large corpora enables MTL models to capture common linguistic features, leading to robust representations (Ruder (2017)). Parameter sharing improves data efficiency, model compactness and provides implicit regularization (Caruana (1997)).

Our goal was to maximise performance on all three tasks. To this end, we first found beneficial architectural changes, then designed a single task model for each task, with full-model fine-tuning. We added more sophisticated task-dependent regularization techniques and created a new loss function for the sentiment task. After an extensive hyperparameter sweep, our results were excellent. However, this method presents a challenge: creating one model per task triples the number of parameters. So we implemented parameter-efficient fine-tuning methods and created a second variant with similar performance but significantly fewer parameters. We then implemented a true multi-task model, a BERT$_{Base}$ model with one set of parameters, but such a model faces challenges in training. Fine-tuning the whole model for each task can lead to gradient conflicts, where improving one task harms another. To address this, we used conflicting gradient mitigation methods and data sampling techniques. Finally, as these three variants made different errors, we aggregated all three into a large ensemble to further improve performance.

2 Related Work

A pre-trained BERT (Bidirectional Encoder Representations from Transformers) model has significantly advanced NLP by producing output embeddings and a pooled token that can be specialized for various tasks (Devlin et al. (2018)). The baseline approach involves adding a linear layer to the pooled token for each task, but architectural variations exist. For instance, in semantic similarity tasks, a Siamese network can be used to compute representations for individual sentences, with the results compared using cosine similarity (Reimers and Gurevych (2019)).

Fine-tuning any chosen architecture can easily lead to overfitting, so careful choice of regularisation is required. Weight decay (van Wieringen (2023)) is a common technique, and adversarial training via SMART (Jiang et al. (2019)) or LNSR (Hua et al. (2023)) injects noise to improve robustness and reduce overfitting. In multi-task learning, fine-tuning for different tasks can cause conflict and different methods are used to reduce this conflict. Using separate weights for each task is costly in terms of memory and computation. To address this, parameter-efficient fine-tuning methods such as LoRA (Hu et al. (2021)) and DoRA (Liu et al. (2024)) introduce low-rank matrices to pre-trained weights, providing comparable performance with a fraction of the parameters.
3 Approach

We start with the BERT\textsubscript{base} model as described in Devlin et al. (2018). The BERT model architecture starts with an embedding layer that includes token and position embeddings to convert the input tokens into contextualized vectors. It then processes these embeddings through a stack of 12 Transformer (Vaswani et al. (2017)) encoder layers, each of which consists of a multi-head self-attention mechanism and a feed-forward neural network, with layer normalization and residual connections ensuring stability. The output of the model consists of the last hidden states for each token, providing rich contextual representations, with the representation of the first token corresponding to the [CLS] token. We extended this architecture by adding a task-specific head consisting of a linear output layer. This model represents our baseline model and the starting point for all modifications.

3.1 Baseline and Variants

To determine the impact of various changes on our model, we first established baseline performance through single-task training for each task, using the hyperparameters given in the assignment and fine-tuning only the task-specific head of the base model. We note that while some baseline numbers are available in the literature (Devlin et al. (2018), Munikar et al. (2019)), we recreated ours to more consistently measure the delta performance provided by our modifications. Four additional models were then developed and compared to the established baseline:

- **AllBERT-SF**: An ensemble of single-task models, each full-model fine-tuned.
- **AllBERT-SD**: An ensemble of single-task models, each fine-tuned using DoRA.
- **AllBERT-M**: A full-model fine-tuned using multi-task sampling / gradient techniques.
- **AllBERT-A**: An ensemble model combining AllBERT-SF, AllBERT-SD and AllBERT-M.

One motivation to build these models is to compare the performance of a multi-task model (AllBERT-M) against an ensemble of single-task fine-tuned models (AllBERT-SF). Since ensembling three full models together triples the parameters, we also researched the performance of a parameter-efficient model (AllBERT-SD) for comparison. Finally, we investigated whether an ensemble of all the previously mentioned models could further increase performance. Each of these variants incorporates architectural and training improvements, which are described in detail below.

3.2 Architectural Improvements

We extended the baseline model by adding a dense layer to each task-specific head, comprising a linear layer, a nonlinear activation function, and a dropout layer, placed before the final linear output layer. This extension ensures that the dense pooling layer of the baseline model (partially bypassed) is replicated, preventing conflicts between tasks from interfering with the final hidden layer. Additionally, we enhanced the BERT model with segment embeddings to provide greater flexibility in handling sentence pairs for semantic similarity and paraphrase detection tasks.

We explored Cosine Similarity for the Semantic Textual Similarity (STS) task using techniques from Reimers and Gurevych (2019), but the results were unsatisfactory. To improve this, we developed a hybrid approach. Instead of using the [CLS] token from concatenated sentences, we used all embeddings from the final hidden layer. We split these embeddings back into their original sentence segments, zeroed out the padding token embeddings, and averaged the remaining embeddings to create two new [CLS] tokens. We then calculated the cosine similarity between these two tokens, scaling the result to get the STS score. Mathematically, let \( h_1 \) and \( h_2 \) be the final hidden layer embeddings for input sentences \( s_1 \) and \( s_2 \). Using \( m_T() \) as the mean pooler and \( a_1, a_2 \) as the attention masks for non-padding tokens, the score is given by

\[
\text{score} = \alpha \frac{m_T(h_1 \cdot a_1) \cdot m_T(h_2 \cdot a_2)}{|m_T(h_1 \cdot a_1)| \cdot |m_T(h_2 \cdot a_2)|} + \beta
\]

where \( \alpha \) and \( \beta \) scale and shift the cosine output to match the range of the corresponding labels.

We further implemented Heinsen Routing in our architecture but did not adopt it due to its lack of performance improvement. For more details on those topics see Appendix [A.1].

3.3 Training Improvements

Even with a strong architecture, effective training is crucial for good results. We used AdamW as the optimizer with weight decay for regularization. For learning rate scheduling, we compared the effects
of the “Exponential” and “Cosine Annealing Warm Restarts” schedulers. Instead of Cross Entropy (CE) for SST-5, we developed our own Ordinal Cross Entropy (OCE, see 3.3.2). For STS, we used Mean Squared Error (MSE) and for QQP, Binary Cross Entropy (BCE). To enhance our scores, we performed an extensive grid search over hyperparameters like learning rate, dropout rate, and weight decay. Details are in Section 4.

3.3.1 Regularization: SMART and LNSR

While pre-training uses large data corpora, downstream tasks often rely on limited data, making aggressive fine-tuning prone to overfitting, where small input changes cause large output variations. To counter this, we implemented two adversarial regularization methods on our own: SMART and LNSR. Adversarial regularization penalizes the loss function for large responses to injected noise. SMART introduces noise at the input embeddings and employs Vanilla Bregman and Momentum Bregman Proximal Point Optimization (VBPP / MBPP) to prevent aggressive updates during training. LNSR, on the other hand, injects noise between BERT layers, enforcing stability across these layers. For more details, see Section A.2.1.

SMART performed well on sentiment and paraphrase tasks. However, for the semantic similarity task, where a linear layer regressor ($y = Ax + b$) is used, SMART’s regularizing term becomes:

$$R_s = \frac{1}{2} \max_{\|\delta_x\| < \epsilon} ((A\tilde{x}_i + b) - (Ax_i + b))^2 = \frac{1}{2} \max_{\|\delta_x\| < \epsilon} (A\delta_x)^2$$

Here, $\delta_x = \tilde{x}_i - x_i = A^T$, meaning the regularizing effect on $A$ is similar to weight decay. Consequently, we opted for LNSR for this task.

LNSR regularizes by making higher BERT layers resilient to Gaussian noise injected into lower layers, enforcing stability across intermediate layers. This method effectively improved scores for the semantic similarity task.

3.3.2 Loss Function: Ordinal Cross Entropy Loss

We developed the Ordinal Cross Entropy Loss to tackle the challenges of sentiment classification, which involves categorizing movie reviews into five ordered classes. Unlike typical classification tasks, sentiment classification benefits from considering the distance between classes. For example, predicting ‘1’ (negative) when the true label is ‘4’ (somewhat positive) is worse than predicting ‘3’ (neutral). Our Ordinal Cross Entropy Loss accounts for this by penalizing predictions based on their distance from the actual class, enhancing the model’s sensitivity to the ordinal nature of the data. Mathematically, for classes $\{1, 2, \ldots, C\}$, labels $y_i$, predicted probabilities $p_{ij}(\theta)$ and a regularization constant $\lambda$, the OCE loss for a batch of inputs $x_i$ where $i = 1, \ldots, N$ is given by:

$$L_{oce}(\theta) = \sum_{i=1}^{N} \sum_{j=1}^{C} \mathbb{I}(y_i = j) \log(p_{ij}(\theta)) - \lambda \sum_{j=1}^{C} p_{ij}(\theta) |y_i - j|$$

Here, $\mathbb{I}$ is the indicator function, $\mathbb{I}(x) = 1$ if $x$ is true (for algorithmic description see Appendix 1).

This approach is inspired by the ordinal log-loss function in Castagnos et al. (2022), but retains the cross-entropy term and includes a tunable hyperparameter to balance the two components. Similar concepts were also explored in Hou et al. (2017).

3.3.3 Parameter-Efficient Fine-Tuning: LoRA and DoRA

Because of the size of transformer-based models, we implemented two low-rank adaptation methods on our own: LoRA and DoRA. DoRA enhances LoRA by decomposing weight updates into magnitude and direction components. It adjusts weights using a combination of low-rank matrices, allowing for dynamic and fine-grained updates. This results in improved flexibility and slightly better performance. For more details, see Section A.2.3.

3.3.4 Multi-Task Learning: Sampling Strategies / Conflicting Gradients

To train our multi-task model, we implemented several advanced techniques. We used two data-sampling methods: Round-Robin-Sampling and Annealed-Sampling Stickland and Murray (2019). To address conflicting gradients, we implemented PCGrad Yu et al. (2020) and utilized CAGrad Liu et al. (2021). For a detailed description, see Section A.2.4.
4 Experiments

4.1 Data

We use the following data sets for the project. For the sentiment classification task, we used the Stanford Sentiment Treebank (SST-5) Socher et al. (2013b) which consists of 11,855 single sentences from movie reviews, each annotated with one of five sentiment labels. The Quora Question Pairs (QQP) dataset contains over 400,000 question pairs with binary labels indicating if the questions are duplicates. This was used for the paraphrase detection task. Finally, for the semantic similarity task we use the Stanford Textual Similarity (STS) Agirre et al. (2013) dataset. It includes 8,628 sentence pairs, each rated on a scale from 0 to 5 to indicate their semantic similarity.

4.2 Evaluation Methods

To evaluate our models we use accuracy between the true and predicted labels for both the sentiment task ($\text{acc}_{\text{sst}}$) and the paraphrase task ($\text{acc}_{\text{qqp}}$). For the semantic similarity task, we use Pearson correlation ($\text{pear}_{\text{sst}}$) of the true and predicted values. The average score is calculated as defined in 8.

4.3 Experimental Details

All our experiments have been run on one of the following NVIDIA GPUs: RTX 4090 (24GB), A100 (24GB) or H100 (80GB). The constant parameters over all experiments are a batch size of 64, 20 epochs and the usage of AdamW as optimizer with $\beta_1 = .9$ and $\beta_2 = .999$. All other parameters have been adjusted based on the task and fine-tuning approach.

4.4 Results

In total, we ran more than 1000 experiments to test our architectural improvements and training improvements and to find the best performing hyperparameters. The best scores of our models and the baseline are shown in Table 1. A discussion of certain aspects is provided afterwards. All other training metrics and configurations can be found in the code archive.

<table>
<thead>
<tr>
<th>Model</th>
<th>Training</th>
<th>STS</th>
<th>SST-5</th>
<th>QQP</th>
<th>AVG</th>
<th># of Parameters (Trainable/Total)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>Single-Task</td>
<td>0.732</td>
<td>0.477</td>
<td>0.796</td>
<td>0.713</td>
<td>1.77M / 111.25M</td>
</tr>
<tr>
<td>AllBERT-SF</td>
<td>Single-Task</td>
<td>0.903</td>
<td>0.562</td>
<td>0.913</td>
<td>0.809</td>
<td>333.77M / 333.77M</td>
</tr>
<tr>
<td>AllBERT-SD</td>
<td>Single-Task</td>
<td>0.894</td>
<td>0.541</td>
<td>0.890</td>
<td>0.793</td>
<td>6.27M / 112.21M</td>
</tr>
<tr>
<td>AllBERT-M</td>
<td>Multi-Task</td>
<td>0.877</td>
<td>0.547</td>
<td>0.889</td>
<td>0.791</td>
<td>111.25M / 111.25M</td>
</tr>
<tr>
<td>AllBERT-A</td>
<td>-</td>
<td>0.907</td>
<td>0.573</td>
<td>0.913</td>
<td>0.813</td>
<td>449.20M / 556.91M</td>
</tr>
</tbody>
</table>

Table 1: Best dev scores per model variant

We submitted our best model, the AllBERT-A to the dev and test leaderboard on Gradescope. The received test score and rank at time of submission are listed in 2.

<table>
<thead>
<tr>
<th>Leaderboard</th>
<th>STS</th>
<th>SST-5</th>
<th>QQP</th>
<th>AVG</th>
<th>Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>AllBERT-A</td>
<td>Test</td>
<td>0.902</td>
<td>0.556</td>
<td>0.912</td>
<td>0.806</td>
</tr>
</tbody>
</table>

Table 2: Test leaderboard score and rank

4.4.1 Baseline / Hyperparameter Search

For the baseline, we performed single-task training with last-layer-only fine-tuning over 10 epochs, using one dense layer for each task head. For the other fine-tuning approaches, the configurations are summarized in Table 3. Across all fine-tuning methods, employing segment embeddings consistently enhanced performance, particularly for sentiment analysis and paraphrasing tasks. For instance, in the STS task, the best full-model fine-tuning configuration achieved a score of 0.903 with segment embeddings compared to 0.884 without.

1 https://quoradata.quora.com/First-Quora-Dataset-Release-Question-Pairs
### Table 3: Fine-Tuning Hyperparameters

<table>
<thead>
<tr>
<th>Task</th>
<th>Learning Rate</th>
<th>Weight Decay</th>
<th>Token Type</th>
<th>Task Head (Act., Drop.)</th>
<th>LR Scheduler</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SST</td>
<td>1e-3</td>
<td>1e-6</td>
<td>Unpooled</td>
<td>ReLU, 0.1</td>
<td>✔</td>
</tr>
<tr>
<td>QQP</td>
<td>1e-3</td>
<td>1e-6</td>
<td>Unpooled</td>
<td>Tanh, 0.3</td>
<td>✔</td>
</tr>
<tr>
<td>STS</td>
<td>1e-3</td>
<td>1e-6</td>
<td>Unpooled</td>
<td>Tanh, 0.3</td>
<td>✔</td>
</tr>
<tr>
<td>Full-Model Fine-Tuning</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SST</td>
<td>5e-5</td>
<td>1e-4</td>
<td>Pooled</td>
<td>ReLU, 0.1</td>
<td>✔</td>
</tr>
<tr>
<td>QQP</td>
<td>5e-5</td>
<td>1e-8</td>
<td>Pooled</td>
<td>Tanh, 0.3</td>
<td>Exp., 0.9999</td>
</tr>
<tr>
<td>STS</td>
<td>5e-5</td>
<td>1e-8</td>
<td>Pooled</td>
<td>Tanh, 0.3</td>
<td>✔</td>
</tr>
<tr>
<td>DoRA Fine-Tuning</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SST</td>
<td>1e-3</td>
<td>1e-1</td>
<td>Unpooled</td>
<td>ReLU, 0.1</td>
<td>✔</td>
</tr>
<tr>
<td>QQP</td>
<td>1e-3</td>
<td>1e-1</td>
<td>Unpooled</td>
<td>ReLU, 0.3</td>
<td>Exp., 0.9999</td>
</tr>
<tr>
<td>STS</td>
<td>1e-3</td>
<td>1e-1</td>
<td>Unpooled</td>
<td>Tanh, 0.3</td>
<td>Exp., 0.9999</td>
</tr>
<tr>
<td>Multi-Task Fine-Tuning</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SST</td>
<td>5e-5</td>
<td>1e-8</td>
<td>Unpooled</td>
<td>SST: ReLU, 0.1</td>
<td>✔</td>
</tr>
<tr>
<td>QQP</td>
<td></td>
<td></td>
<td></td>
<td>QQP: Tanh, 0.3</td>
<td></td>
</tr>
<tr>
<td>STS</td>
<td></td>
<td></td>
<td></td>
<td>STS: Tanh, 0.3</td>
<td></td>
</tr>
</tbody>
</table>

### 4.4.2 SMART / LNSR / Bregman

Regularizing with SMART and LNSR was crucial in enhancing scores across all fine-tuning methods. SMART yielded better results for QQP and SST tasks, while LNSR excelled in the STS task. We also experimented with Vanilla Bregman Proximal Point (VBPP) and Momentum Bregman Proximal Point (MBPP) optimizations, finding that VBPP consistently outperformed MBPP. However, using VBPP posed a memory consumption challenge, as it requires storing all data points between training epochs to prevent aggressive updates. Detailed hyperparameters for these regularization methods are listed in Tables 4 and 5.

<table>
<thead>
<tr>
<th>Data Set</th>
<th>Fine-Tuning</th>
<th>λ</th>
<th>T</th>
<th>ε</th>
<th>η</th>
<th>σ²</th>
<th>VBPP μ</th>
</tr>
</thead>
<tbody>
<tr>
<td>QQP</td>
<td>DoRA</td>
<td>0.25</td>
<td>10</td>
<td>1e-6</td>
<td>1e-5</td>
<td>1e-5</td>
<td>0.5</td>
</tr>
<tr>
<td>QQP</td>
<td>Full-Model</td>
<td>15</td>
<td>10</td>
<td>1e-6</td>
<td>1e-5</td>
<td>1e-5</td>
<td>0.9</td>
</tr>
<tr>
<td>SST-5</td>
<td>DoRA / Full-Model</td>
<td>15</td>
<td>10</td>
<td>1e-6</td>
<td>1e-5</td>
<td>1e-5</td>
<td>1.0</td>
</tr>
<tr>
<td>QQP, SST, STS</td>
<td>Multi-Task</td>
<td>1.5</td>
<td>1</td>
<td>1e-6</td>
<td>1e-5</td>
<td>1e-5</td>
<td>0.9</td>
</tr>
</tbody>
</table>

Table 4: SMART and VBPP parameters for best performing models (SST and QQP)

<table>
<thead>
<tr>
<th>Data Set</th>
<th>Fine-Tuning</th>
<th>λ</th>
<th>B</th>
<th>σ²</th>
<th>VBPP μ</th>
</tr>
</thead>
<tbody>
<tr>
<td>STS</td>
<td>DoRA</td>
<td>0.0005</td>
<td>12</td>
<td>1e-5</td>
<td>1.4</td>
</tr>
<tr>
<td>STS</td>
<td>Full-Model</td>
<td>0.005</td>
<td>10</td>
<td>1e-5</td>
<td>0.9</td>
</tr>
</tbody>
</table>

Table 5: LNSR and VBPP parameters for best performing models (STS)

### 4.4.3 LoRA / DoRA

To compare LoRA and DoRA, we performed single-task training with full-model LoRA and DoRA fine-tuning for each dataset. We kept the hyperparameters constant across data sets for full-model training and constant for LoRA/DoRA training. As shown in Table 6 while full-model fine-tuning achieved the highest scores, both LoRA and DoRA performed comparably well, using only about 2% of the parameters for training. This suggests that allowing the model to retain the core weights learned from BERT, while only adjusting the additional weights in the form of low-rank matrices, provides sufficient flexibility to successfully adapt to new data.
In DoRA, the rank represents the size of the low-rank matrices, and $\alpha$ is a hyperparameter that regularizes the adaptation to these learned low-rank matrices. To evaluate the optimal values, we conducted a comprehensive experiment on the SST-5 and STS datasets, as shown in table 10. Unlike the other experiments, this one was run for 10 epochs. For each data set, the configuration was kept constant, with only the DoRA rank (DR-R) and $\alpha$ (DR-A) being adjusted. As DoRA slightly out-performed LoRA, we used it for our parameter-efficient model.

4.4.4 Ordinal Cross Entropy Loss

The application of the Ordinal Cross Entropy Loss (OCE) helped to increase the performance on SST-5. Using the best-performing configuration, the highest score reached 0.562 with OCE compared to 0.540 with the standard Cross Entropy (CE) loss. We observed a similar pattern when running OCE with a base configuration without regularization, though on a smaller scale. In this scenario, OCE achieved a score of 0.526, whereas CE reached 0.523.

4.4.5 Multi-Task / Data Sampling / Conflicting Gradients

We experimented with various data sampling strategies, gradient conflict handlers, regularization techniques and hyperparameter adjustments to train our multi-task model. Each experiment was run for 20 epochs, with a maximum of 80 iterations per epoch, totaling 5,120 data points per epoch (80 iterations $\times$ 64 batch size). Table 7 shows that annealed sampling except for one case outperformed round-robin sampling. While both methods performed similarly on STS and SST-5, they showed significant differences on QQP. This is because round-robin sampling gives equal exposure to each dataset, leading to underfitting on the larger QQP dataset. In contrast, annealed sampling adjusts exposure based on dataset size, providing a balanced approach (see Table 8).

<table>
<thead>
<tr>
<th>Fine-Tuning Variant</th>
<th>STS</th>
<th>SST-5</th>
<th>QQP</th>
<th># Trainable Param. (per Task)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full-Model</td>
<td>0.885</td>
<td>0.520</td>
<td>0.871</td>
<td>111.2M</td>
</tr>
<tr>
<td>LoRA [Rank 8, Alpha 8.0]</td>
<td>0.883 (-.002)</td>
<td>0.514 (-.006)</td>
<td>0.854 (-.017)</td>
<td>2.07M</td>
</tr>
<tr>
<td>DoRA [Rank 8, Alpha 8.0]</td>
<td>0.884 (-.001)</td>
<td>0.515 (-.005)</td>
<td>0.855 (-.016)</td>
<td>2.09M</td>
</tr>
</tbody>
</table>

Table 6: Fine-tuning comparison between Full-Model / LoRA / DoRA
score (0.783) compared to using VBPP without CAGrad (0.790). We suspect that the regularization effects of VBPP are negatively impacted by the conflicting gradient mitigation, leading to decreased performance.

4.4.6 Ensembling AllBERT-A: Mixture of Best Models

We combined our existing models into a comprehensive ensemble, AllBERT-A, to leverage their strengths and improve overall performance. Using a majority vote for SST and QQP, and a mean prediction for STS, we selected the highest-scoring versions of each model. This approach aimed to harness the diverse error patterns of individual models. Our ensemble, AllBERT-A, showed enhanced performance on SST and STS but not for QQP, indicating the need for careful model selection based on task-specific strengths. We therefore reverted for QQP to using the AllBERT-SF model.

To optimize ensemble inclusion, we examined multi-task models at various training epochs, recognizing that peak average scores don’t always align with optimal performance for all tasks. We found that AllBERT-M at epoch 20 was better for SST, and at epoch 16 for STS, despite not having the highest average scores. This highlighted the importance of considering more than just peak accuracy. For STS, we fine-tuned the ensemble by weighting predictions differently: AllBERT-SF eight times, AllBERT-SD once, and AllBERT-M twice. Our final AllBERT-A included the top-performing AllBERT-SF and AllBERT-SD models.

<table>
<thead>
<tr>
<th></th>
<th>STS</th>
<th>SST-5</th>
<th>QQP</th>
<th>AVG</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equal Mean STS, MajV. SST/QQP</td>
<td>0.903</td>
<td>0.569</td>
<td>0.904</td>
<td>0.808</td>
</tr>
<tr>
<td>Weighted Mean STS, MajV. SST, AllBERT-SF QQP</td>
<td>0.905</td>
<td>0.569</td>
<td>0.913</td>
<td>0.811</td>
</tr>
<tr>
<td>SST: AllBERT-M_{BestAvg, EP20} / STS: AllBERT-M_{BestAvg, EP16}</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Weighted Mean STS, MajV. SST, AllBERT-SF QQP</td>
<td>0.907</td>
<td>0.573</td>
<td>0.913</td>
<td>0.813</td>
</tr>
</tbody>
</table>

Table 9: Scores of AllBERT-A experiments

5 Analysis

In order to better understand our system (e.g. how it works, when it succeeds and when it fails), we have examined the key features and outputs of our model. We give a qualitative evaluation for each task, based on the AllBERT-SF model.

5.1 Sentiment Classification

While accuracy is a useful metric for the SST dataset, it ignores the fact that some answers are more wrong than others. To see the extent of the errors, rather than just their presence, we visualised them using a histogram showing the difference between the predicted class and the label. See figure 1(a). The vast majority of errors were within one of the labels (had a difference of +1 or -1), indicating that the model is actually much better than its accuracy would suggest. Given the difficulty a human has in consistently reporting the same score, this suggests that the model’s performance may actually be almost human. An example of non-human-like difficulty can be seen in the sentence “hilariously inept and ridiculous”, where the label is 0 and the predicted score is 3. It appears that the model is confused by the use of the words “inept” and “ridiculous” as comedic positives. To improve the current model, either more specific embedding for movie reviews might help, or perhaps more advanced pre-processing using something like recursive neural tensor processing (Socher et al. (2013a)).

5.2 Paraphrase Detection

The confusion matrix for the paraphrasing task is shown in Figure 1(b). The results are generally very good, but while the errors are balanced between the two classes, the classes themselves are slightly unbalanced. To get a sense of the problems the model has with the QQP dataset, we looked at a handful of misclassified sentence pairs. The pair “how do i write an android application?” and “how do i create an android application?” was labelled as a paraphrase, but was not predicted to be one. Here the model clearly has difficulty recognizing the similarity of the words “write” and “create” when restricted to programs.
5.3 Semantic Textual Similarity Analysis

We can visualize the model’s effectiveness using a scatter plot of the sentence pairs’ predicted similarity versus labeled similarity. See figure [1c]. While the plot shows, as expected, a good correlation between the two sets, it also shows some interesting outliers. The worst example is the sentence pair “work into it slowly” versus “it seems to work”, which had a prediction of 2.86 but a label of 0.0. It seems that the model is struggling with the fact that both sentences are short with very similar words. One solution is to try to use word embeddings that do a better job of isolating different meanings for a single word, as the word “work” here changes meaning from “go” to “function”.

6 Conclusion

In this report, we studied various methods to enhance the performance of BERT on three downstream tasks. We experimented with different architectural modifications and training techniques. Key architectural improvements included replicating the pooling layer across tasks, using segment embeddings, and developing an enhanced version of cosine similarity. Training enhancements involved the use of SMART and LNSR regularization methods and the introduction of a novel loss function for the sentiment task. Our confidence in the results stems from the extensive grid search for hyperparameters that we employed.

We also implemented LoRA and DoRA, demonstrating that, with a similar increase in weight size, DoRA outperformed LoRA. While the parameter efficiency of these methods is beneficial in many contexts, it slightly reduced performance and was thus not included in the final model for the leaderboard. Through improvements in architecture, the choice of regularization methods, and comprehensive hyperparameter tuning, we achieved excellent performance. By ensembling our three models into AllBERT-A, we even surpassed the performance of the AllBERT-SF model, which was based on single-task full-model fine-tuning with all parameters replicated.

7 Ethics Statement

As our project focuses on fine-tuning the BERT model, we anticipate the primary ethical challenge to be the reinforcement or amplification of biases present in the training data. While our model is not intended for mission-critical applications, mitigating the risk of reinforcing racial or gender bias or providing misleading responses remains important.

Since fine-tuning can potentially amplify existing biases, we conducted an experiment using sentiment prediction. To evaluate this, we used ChatGPT-4 to generate two new movie review data sets (see Section A.5). The content of the reviews in both data sets is identical, but one dataset uses names commonly associated with white people, while the other uses names commonly associated with people of color. In our initial experiment, we deliberately focused on positive reviews to determine if the sentiment rating worsened when names associated with people of color were used.

We evaluated these data sets with our models to verify if the sentiment predictions for each review remained consistent, regardless of the names used. Our results showed that the predictions were the same across both data sets, indicating that our models did not exhibit increased bias through our fine-tuning adjustments. Although this experiment is not conclusive, it provides an initial indication that our adjustments have not exacerbated existing biases.

8 Key Information

Default Project / Mentor: Soumya Chatterjee / No external collaborators, external mentors / Project not shared with any other class.
Both Thierry and Joe independently coded the base BERT model, with Joe noticing the improvement from replicating the pooling layer and Thierry discovering the improvement from segment embeddings. We shared the implementation and analysis of the SMART and LNSR algorithms. Joe investigated the use of cosine similarity for semantic similarity, while Thierry designed the new loss function for sentiment encoding and was responsible for the implementation of LoRA, DoRA and the multi-task training regimes.

References


A Appendix

A.1 Additional Architectural Details

A.1.1 Heinsen Routing

Heinsen Routing is an algorithm designed to process sequences of vectors, such as the outputs from each BERT layer, and generate a new sequence with a specified length and size. Motivated by its current best score on the SST-5 dataset, we decided to give it a try. The algorithm is an extended expectation-maximization (EM) algorithm that operates through three iterative steps: the Expectation Step (E-Step) to calculate the probabilities of assigning input vectors to output vectors, the Data Step (D-Step) to determine how much data from each input vector should be used or ignored, and the Maximization Step (M-Step) to update the output vectors by maximizing their usefulness. This iterative process refines the output vectors to better represent the input data. By leveraging the output from each BERT layer, we assume Heinsen Routing can summarize and enhance the information captured at different BERT layers, resulting in more accurate and efficient representations. For the implementation, we relied on the code made available by the paper at

https://github.com/glassroom/heinsen_routing

and

https://github.com/glassroom/heinsen_routing_2022_paper

Our attempts to replicate the improvements with Heinsen Routing were unsuccessful. Single-task training (without any further improvements) on the SST-5 dataset yielded a best score of 0.515 with Heinsen Routing compared to 0.520 without it. Notably, the 0.515 score was achieved in the first epoch, after which performance deteriorated in each subsequent epoch. We hypothesise that the lack of improvement could be due to the different base models used in training (Heinsen Routing used RoBERTa).

A.1.2 Cosine Similarity

Semantic Textual Similarity (STS) aims to assign a score to a pair of sentences indicating the degree of similarity between them. The basic method for determining the STS score is to concatenate the sentences at the BERT input and then use the resulting [CLS] pooling token as input to the task head. This technique, known as cross-encoding, was introduced in the original BERT paper. An alternative method, proposed by Reimers and Gurevych (2019), processes each sentence independently with BERT, then calculates the cosine similarity of the two output pooling tokens. This value is scaled to give the final STS score. This approach is also called bi-encoding.

Let \( B(s) \) be the resulting pooled token output from BERT for sentence \( s \). The bi-encoding method gives the result:

\[
\text{score} = a \cdot B(s_1) \cdot B(s_2) \frac{1}{\|B(s_1)\| \cdot \|B(s_2)\|} + b \tag{3}
\]

where \( a \) and \( b \) scale and shift the cosine output to match the range of the corresponding labels. Denoting \( s_1 + s_2 \) as the concatenation of the two sentences, cross-encoding gives the result:

\[
\text{score} = A \cdot B(s_1 + s_2) + b \tag{4}
\]

The two approaches have different strengths. The bi-encoding strategy creates a signature for each sentence, making it easier to compare with other sentences, while the cross-encoding method learns relationships between sentences, allowing it to better estimate their true similarity.

We have developed a hybrid of these two approaches. We start with cross-encoding, where the sentence pair generates one BERT output. Instead of using the [CLS] token as the output, we use all the hidden layer embeddings. We split the outputs corresponding to the original two sentences and set the embeddings corresponding to the padding tokens to zero. We then average each of the two segments to obtain two values, which are used as the new [CLS] tokens. Finally, we compute the cosine similarity of these two newly formed [CLS] tokens and use its scaled value as the STS score.

In equation form, let \( h_1 \) and \( h_2 \) be the last hidden layer of the BERT model with input sentences \( s_1 \) and \( s_2 \). Let \( m_T(h) \) be the mean pooler over time, so that \( m_T(h) \) is a vector, and let \( a_1 \) and \( a_2 \) be the
attention masks denoting the non-padding and padding tokens. Then the score from this method is given by:

\[
\text{score} = a \frac{m_t(h_1 \cdot a_1) \cdot m_t(h_2 \cdot a_2)}{||m_t(h_1 \cdot a_1)|| ||m_t(h_2 \cdot a_2)||} + b
\]

and \(a\) and \(b\) are as defined in [3].

To evaluate performance effects, we tested three algorithms on the STS task: classic BERT baseline, bi-encoder siamese network from Reimers and Gurevych (2019), and our improved cosine similarity. The three models were trained for 20 epochs using the hyperparameters from the project handout (learning rate of \(1 \times 10^{-5}\), dropout probability of 0.3, weight decay of \(1 \times 10^{-4}\)). The baseline achieved a correlation of 0.878, while the bi-encoder achieved 0.504. The improved cosine similarity cross-encoder achieved a correlation of 0.885. Since we had already achieved high scores using other techniques, we did not continue to pursue Cosine Similarity.

### A.2 Additional Training Details

#### A.2.1 Regularization: SMART

While pre-training uses large corpora of data, downstream tasks may rely on more limited data. Aggressive fine-tuning can lead to overfitting, where the model fails to generalize to unseen data. Overfitting can be characterized by small changes in the input causing large changes in the output. The SMART tuning algorithm aims to mitigate this by penalizing parameter selections that lead to such instability. It measures this by injecting a small amount of noise into the input and observing the resulting output changes. This type of training is known as “adversarial” training, as it attempts to “challenge” the model by modifying the input data. Specifically, SMART uses a “smoothness-inducing” term of the form

\[
R_s = \frac{1}{N} \sum_{i=1}^{N} \max_{\|\delta_i\|_2 < \epsilon} l_s(f(\tilde{x}_i; \theta), f(x_i; \theta))
\]

where \(\ell_s\) is the function, usually either Mean Squared Error (MSE) or Symmetric Kullback-Leibler (SKL) divergence, that characterizes the change in the two terms. To prevent aggressive updates, SMART adds a Bregman penalty of the form

\[
B = \frac{1}{N} \sum_{i=1}^{N} l_s(f(x_i; \theta), f(x_i; \theta_i))
\]

where \(\theta_i\) produces the output to stay near, \(\theta\) is the parameter set to be optimized and \(\ell_s\) is as in Eq. 6. Taken together, these two terms try to alter the gradient used in the descent algorithm to seek a \(\theta\) with an output that is smooth (less overfitting) and similar to the previous output. The version described here is the Vanilla Bregman Proximal Point (VBPP) method. \(\theta_i\) can also be calculated by momentum to give the Momentum Bregman Proximal Point (MBPP) method.

When SMART is used on a regressor with a last linear layer \(y = Ax + b\) where \(A \in \mathbb{R}^{1 \times h}\), as in the case of Semantic Similarity, the regularizing term becomes

\[
R_s = \frac{1}{2} \max_{\|\delta_i\|_2 < \epsilon} \left(\|A\tilde{x}_i + b\| - \|Ax + b\|\right)^2 = \frac{1}{2} \max_{\|\delta_i\|_2 < \epsilon} (A\delta_x)^2
\]

From the constraint, we can deduce that \(\delta_x = \tilde{x}_i - x_i = AT\), which means that in this case, the regularizing effect on \(A\) is the same as weight decay.

We implemented SMART, as well as the Vanilla Bregman Proximal Point (VBPP) and the Momentum Bregman Proximal Point (MBPP) optimization, on our own.

#### A.2.2 Regularization: LNSR

Another adversarial training algorithm is Layerwise Noise Stability Regularization (LNSR). While SMART restricts itself to injecting noise at the input embeddings and measuring the effects at the output, LNSR imposes an additional optimization term that forces higher layers of BERT to be resilient to Gaussian noise injected into lower layers. It features a novel layer-wise regularization that explicitly enforces noise stability in middle layers. The regularization term is conceptually simpler than SMART, relying on standard mean squared error induced by noise injected at lower levels. Given
an injection at layer \( b \) of an L-layer model, a measurement at layer \( r \geq b \), a constant \( \lambda_{b,r} \), an input to layer \( b \) of \( x \) perturbed by the injected noise to \( \tilde{x} \), the regularization term \( R_{\text{LNRS}} \) is augmented by

\[
R_{\text{aug}}(x) = \sum_{i=b}^{L} \lambda_{b,r} \| f_{b,r}(x) - f_{b,r}(\tilde{x}) \|^2.
\]

Here, \( f_{b,r} \) represents the portion of the model that maps inputs at layer \( b \) to outputs at layer \( r \). After augmenting the regularization term for each \( x \) in the batch, the gradient with respect to the parameters can be calculated.

Compared to SMART, LNSR measures the effects of noise more broadly. While SMART only assesses the impact of noise at the output, LNSR regularizes noise at the interior layers. Its cost function, MSE, is also simpler than the KL-divergence used in SMART. However, the significance of interior noise stability is less clear than stability at the output. Additionally, LNSR involves tuning more constants, as each input-output layer gain can be individually adjusted.

We implemented LNSR in its standard version on our own.

A.2.3 Parameter-Efficient Fine-Tuning

Because of the size of transformer based models, we integrated two methods of low-rank adaptation. LoRA [Hu et al. (2021)] is the more basic idea. A transformer based system has many linear operations performing matrix multiply like in the calculation of keys, queries, and values in attention. For example, the query term in head \( i \) is formed from the queries \( Q \) and the weight matrix \( W_Q^i \). Full parameter retraining can be viewed as finding a matrix \( \Delta W^i \) so that the sum

\[
W' = W + \Delta W
\]

is a more optimal weight matrix. To reduce the number of parameters learned, LoRA introduces two low-rank matrices \( A \) and \( B \) such that

\[
W' = W + \Delta W = W + AB
\]

and the rank of both \( A \) and \( B \) are both much smaller than the rank of \( W \). This greatly reduces the number of parameters to learn with only a small loss in performance versus using a full-rank matrix for \( \Delta W \).

The DoRA [Liu et al. (2024)] algorithm improves on LoRA by decomposing the update matrix into magnitude and direction components. Specifically, this takes the form

\[
W' = m \frac{V + \Delta V}{\|V + \Delta V\|} = m \frac{W_0 + AB}{\|W_0 + AB\|}
\]

where \( \Delta V \) is the incremental directional update learned from multiplying \( A \) and \( B \), and \( m \) is a trainable parameter controlling the magnitude of the direction vectors. The change in the training method allows DoRA to attain slightly better performance than DoRA while altering almost the same number of parameters.

A.2.4 Multi-Task Learning: Sampling Strategies / Conflicting Gradients

In multi-task model training, data sampling methods significantly impact performance [Stickland and Murray (2019)]. The paper discusses ‘round-robin sampling,’ which cycles through tasks in a fixed order. This approach can lead to overfitting on smaller datasets and under-training on larger ones, as it treats all tasks equally, regardless of dataset size. To address this, the paper proposes ‘annealed sampling,’ a strategy that adjusts sampling probabilities to balance training across tasks of different sizes. Annealed sampling begins by sampling data proportionally to the size of each task’s dataset using

\[
p_i \propto N_i^\alpha,
\]

where \( p_i \) is the probability of selecting task \( i \), \( N_i \) is the number of training examples for task \( i \), and \( \alpha \) controls the disparity. Over time, \( \alpha \) is dynamically adjusted according to

\[
\alpha = 1 - 0.8 \frac{e - 1}{E - 1},
\]

where \( e \) is the current epoch and \( E \) is the total number of epochs. This transition ensures a gradual shift from size-proportional sampling to equal sampling, preventing smaller tasks from being neglected as training progresses. We implemented both strategies on our own.
A further key challenge in optimising multi-task models is the presence of conflicting gradients, dominating gradients and high curvature introduced by reducing the loss for different tasks (Yu et al. (2020)). To tackle this problem, we tested two different approaches, PCGrad and CAGrad.

Project Conflicting Gradients (PCGrad, Yu et al. (2020)) and Conflict Averse Gradient Descent (CAGrad, Liu et al. (2021)) are techniques designed to address conflicting gradients in multi-task learning. PCGrad resolves conflicts by projecting one gradient onto the normal plane of another. For gradients $g_i$ and $g_j$ from tasks $T_i$ and $T_j$, respectively, it calculates their cosine similarity. If the similarity is negative, indicating a conflict, PCGrad adjusts $g_i$ to

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

making $g_i^{PC}$ orthogonal to $g_j$, thus preventing interference. CAGrad, on the other hand, dynamically adjusts the combination of gradients from multiple tasks by combining them into

$$G = \sum \lambda_i g_i,$$

where $\lambda_i$ are weights optimized to minimize conflicts. This optimization aligns $G$ with a desired direction while reducing the negative impact of any conflicting gradients, ensuring a conflict-averse update. We implemented PCGrad ourselves and used the CAGrad implementation from [45].

### A.3 Ordinal Cross Entropy Loss

**Algorithm 1 Ordinal Cross Entropy Loss**

**Require:** input: Predicted labels  
**Require:** target: True labels  
**Require:** n_classes: Number of possible classes  
**Require:** weight: Rescaling weight for cross_entropy  
**Require:** $\lambda$: Weight for the distance penalty

1: $ce_loss \leftarrow$ cross_entropy(input, target, weight)  
2: predicted_probs $\leftarrow$ softmax(input)  
3: class_indices $\leftarrow$ array_range(n_classes)  
4: distance $\leftarrow$ absolute(class_indices - true_class)  
5: distance_penalty $\leftarrow$ sum(predicted_probs * distance)  
6: loss $\leftarrow$ ce_loss + $\lambda$ * distance_penalty  
7: return loss

### A.4 DoRA

<table>
<thead>
<tr>
<th>DoRA Parameters</th>
<th>SST</th>
<th>STS</th>
</tr>
</thead>
<tbody>
<tr>
<td>DR-R 8, DR-A 16.0</td>
<td>0.512</td>
<td>-</td>
</tr>
<tr>
<td>DR-R 8, DR-A 8.0</td>
<td>0.514</td>
<td>0.867</td>
</tr>
<tr>
<td>DR-R 8, DR-A 4.0</td>
<td>0.505</td>
<td>0.866</td>
</tr>
<tr>
<td>DR-R 8, DR-A 2.0</td>
<td>0.504</td>
<td>0.866</td>
</tr>
<tr>
<td>DR-R 8, DR-A 1.0</td>
<td>0.504</td>
<td>0.863</td>
</tr>
<tr>
<td>DR-R 4, DR-A 8.0</td>
<td>0.509</td>
<td>0.866</td>
</tr>
<tr>
<td>DR-R 4, DR-A 4.0</td>
<td>0.503</td>
<td>0.863</td>
</tr>
<tr>
<td>DR-R 4, DR-A 2.0</td>
<td>0.501</td>
<td>0.864</td>
</tr>
<tr>
<td>DR-R 4, DR-A 1.0</td>
<td>0.507</td>
<td>0.865</td>
</tr>
</tbody>
</table>

Table 10: Dev scores for SST-5 / STS with different DoRA ranks and alphas

[4](https://github.com/Cranial-XIX/CAGrad)

[5](https://github.com/Cranial-XIX/FAMO)
### A.5 Ethics - Movie Review Data Sets

<table>
<thead>
<tr>
<th>Id</th>
<th>Id</th>
<th>Sentence</th>
<th>Sentiment</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>c1c6...</td>
<td>John gave a brilliant performance, making the movie an absolute delight.</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>5ac0...</td>
<td>Emily’s role was captivating and truly added depth to the story.</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>eed9...</td>
<td>Michael’s portrayal of the character was both nuanced and compelling.</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>a442...</td>
<td>Jessica’s performance brought an emotional richness that was unforgettable.</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>6920...</td>
<td>David’s interpretation of the role was simply mesmerizing.</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 11: Excerpt from data set with names commonly associated with white people

<table>
<thead>
<tr>
<th>Id</th>
<th>Id</th>
<th>Sentence</th>
<th>Sentiment</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>c1c6...</td>
<td>Jamal gave a brilliant performance, making the movie an absolute delight.</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>5ac0...</td>
<td>Ebony’s role was captivating and truly added depth to the story.</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>eed9...</td>
<td>Malik’s portrayal of the character was both nuanced and compelling.</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>a442...</td>
<td>Jasmine’s performance brought an emotional richness that was unforgettable.</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>6920...</td>
<td>DeShawn’s interpretation of the role was simply mesmerizing.</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 12: Excerpt from data set with names commonly associated with people of color

### A.6 Evaluation Methods

\[
\text{avg\_score} = \frac{1}{3} \left( \text{acc}_{\text{qpp}} + \text{acc}_{\text{sts}} + \left( \frac{1}{2} + \frac{1}{2} \times \text{pear}_{\text{sts}} \right) \right)
\]  

(8)