Pretrain and Fine-tune BERT for Multiple NLP Tasks

Stanford CS224N Default Project

Mengge Pu, Yawen Guo
Department of Computer Science
Stanford University
`mpu217@stanford.edu, ywguo@stanford.edu`

Abstract

Our project investigates the application of multi-task learning (MTL) using a miniBERT model across three NLP tasks: sentence sentiment classification (SST), paraphrase detection (PARA), and semantic textual similarity (STS). Initial experiments revealed single-task models excel in their respective tasks but falter on unencountered tasks, emphasizing the need for an effective MTL approach. We explored AdaBelief optimization, Gradient Clipping, and advanced loss functions to enhance our MTL model's generalization capabilities. Notably, strategies like Geometric Loss Strategy and round-robin resampling were tested to address task imbalance. Despite computational limitations precluding the exploration of Knowledge Distillation, our findings highlight potential pathways for improving MTL frameworks in NLP, suggesting directions for future research to further refine and optimize multi-task learning models.

1 Key Information to include

- Mentor: Anirudh Sriram
- External Collaborators (if you have any): N/A
- Sharing project: N/A
- Team contribution: Mengge owned development of single-task model and dynamic optimizer, gradient clipping, GLS, FLS and round-robin resampling approaches in MTL experiments. Yawen owned development of MTL baseline, loss enhancement, SMART regularization, and hyperparameter tuning. Everyone contributed to the default model setup, running experiments on cloud, brainstorming design choices and the reporting drafting.

2 Introduction

The emergence of pre-trained language models (LMs) like BERT (Devlin et al., 2019) has revolutionized Natural Language Processing (NLP), significantly advancing language understanding and unlocking potential applications across diverse tasks. This project investigates the transfer learning capabilities of miniBERT, a BERT variant, for its adaptability to various NLP tasks: sentiment classification (SST), paraphrase detection (PARA), and semantic textual similarity (STS).

Transfer learning in NLP offers both exciting opportunities and significant challenges. Adapting a model trained on a massive corpus to specific downstream tasks can yield impressive results, but it's often hampered by issues like task interference, data imbalance, and overfitting. These challenges necessitate novel approaches that harness the power of pre-trained LMs while addressing the complexities of fine-tuning for specific tasks.

We propose a multi-task learning (MTL) framework to simultaneously fine-tune a single model on multiple tasks, aiming to achieve a balance between task-specific performance and generalization. This MTL approach, enhanced by AdaBelief optimizer, gradient clipping, and task-specific loss adjustments, addresses the challenges of overfitting and training imbalances. Our results indicate improvements in individual tasks, underscoring the benefits of a well-designed MTL framework for NLP.

The subsequent sections of this paper explore the relevant background in Section 3, detailing theoretical foundations and empirical successes that inform our current understanding. Section 4 outlines the methodologies underpinning our model approaches. Section 5 provides a comprehensive account of the experimental setup and results. Section 6 delves into the experiment findings and key learnings. Finally, Section 7 summarizes our main findings and outlines opportunities for future exploration.

3 Related Work

The concept of Multi-task Learning in NLP is deeply rooted in leveraging shared knowledge across multiple tasks to enhance model performance and efficiency. Caruana's seminal work (Caruana, 1997) laid the groundwork for MTL, demonstrating its potential in leveraging task commonalities to improve learning outcomes. More recent explorations, particularly with models like BERT (Devlin et al., 2019), have underscored the versatility of pre-trained language models in adapting to a variety of NLP tasks through fine-tuning, a testament to MTL's capacity for improving model generalization.

In the realm of optimization techniques, the AdaBelief optimizer (Zhuang et al., 2020)emerges as a noteworthy innovation, designed to combine the strengths of adaptive and stochastic gradient methods. Its development marks a significant advance towards stabilizing training and enhancing the robustness of models against overfitting, a common challenge in the optimization landscape of deep learning. Gradient clipping has established itself as an indispensable technique for preventing gradient explosion, a critical concern when training deep neural architectures. The method, extensively discussed in Pascanu et al. (2013) and Chen et al. (2018), ensures training stability across various architectures, including the complex models employed in NLP.

The refinement of loss functions and the integration of regularization techniques play pivotal roles in tailoring models to specific NLP tasks. The adaptation of mean squared error (MSE) for semantic textual similarity tasks exemplifies the customization of loss functions to capture subtle semantic nuances. Concurrently, regularization methods, such as L1 and SMART regularization (Jiang et al., 2020), illustrate the ongoing efforts to prevent overfitting and promote the learning of generalized representations.

Addressing the challenge of task dominance in MTL, innovative loss strategies like the Geometric Loss Strategy and Focused Loss Strategy offer mechanisms for balancing the contribution of individual tasks to the training process. These strategies, inspired by recent studies (e.g. Chennupati et al. (2019) and Bi et al. (2022)), underscore the significance of dynamically weighted loss functions in achieving equitable learning across tasks. The Round-Robin Resampling strategy tackles the issue of data imbalance in MTL by ensuring all tasks, regardless of dataset size, contribute equitably to model training. This approach mirrors strategies in data mining where balancing class representation has been shown to enhance model fairness and performance.

Our project integrates these diverse strands of research, employing a multi-faceted approach that encompasses MTL, advanced optimization techniques, tailored loss functions, and strategic data sampling. By drawing on the collective insights from these areas, we aim to enhance the adaptability and effectiveness of language models across a spectrum of tasks.

4 Approach

The architecture of the model in our main approach is a multi-layer bidirectional Transformer encoder based on the original BERT model (Devlin et al., 2019). The BERT model first converts sentences into tokens, and then converts each token to ids and utilizes a trainable embedding layer across each token. This model incorporates 12 Encoder Transformers layers for language understanding via multi-head self-attention mechanisms, where each embedding layers has 768 dimensions. The input embeddings are the sum of the token embeddings, the segmentation embeddings, and the position embeddings. The final hidden state of the first token of every sequence is used for the transformation to the output. The exact form of the transformation applied to the final hidden state of the [CLS] token is known as a ‘pooling layer’, which can be used for fine-tuning based on multi-task learning for downstream tasks.

Leveraging this pre-trained model, we started with a single-task learning approach where three models were fine-tuned separately using individual dataset on three tasks: sentence sentiment classification (SST), paraphrase detection (PARA), and semantic textual similarity (STS). This evaluation offers insights into BERT’s transfer learning capabilities when applying to multiple downstream tasks. Although the results is comparable, our goal is to formulate a multi-task learning (MTL) approach where a single model is fine-tuned with all datasets altogether and can generalize well on all downstream tasks. To establish the baseline for this multi-task learning approach, we assessed the performance of fine-tuning the pre-trained (min)BERT model using a multi-task sequential learning approach. Specifically, initial layers or parameters are shared between these tasks such that these parameters are common for all tasks. In this multi-task benchmark model setting, we use pooler layers that outputs logits for each task and update the model using cross-entropy for all tasks. We used sequential multi-task learning approach by summing up the lossesBi et al. (2022) of three downstream tasks in model updates:

$$\mathcal{L}\_total = \mathcal{L}\_sentiment + \mathcal{L}\_paraphrase + \mathcal{L}\_sts \tag{1}$$

The core of our experiments, however, is dedicated to exploring three extended modeling approaches leveraging this baseline MTL fine-tuned method as below.

4.1 Dynamic Optimizer and Gradient Clipping

When comparing the performance of our multi-task learning approach baseline that is fine-tuned using all three downstream task datasets together in one model against the performance of single-task learning approach that is fined-tuned using each downstream task dataset in three separate models, we noticed that the accuracy rates of SST task and PARA task both decrease with the former has the larger decrease magnitude. Given the relatively larger training datasets under these two tasks, we suspect our multi-task fine-tune baseline approach overfitted on these two tasks. To examine and reduce it, we explored two methods: **AdaBelief** optimizer and **Gradient Clipping**.

AdaBelief: Most popular optimizers for deep learning can be broadly categorized as adaptive methods (e.g. AdamW) and accelerated schemes (e.g. stochastic gradient descent (SGD) with momentum). For many models such as convolutional neural networks (CNNs), adaptive methods typically converge faster but generalize worse compared to SGD; for complex settings such as generative adversarial networks (GANs), adaptive methods are typically the default because of their stability. AdaBelief can simultaneously achieve three goals: fast convergence as in adaptive methods, good generalization as in SGD, and training stability via adapting the step size according to the "belief" in the current gradient direction to reduce overfitting in the multi-task learning framework. Therefore, we adopted AdaBelief in almost all our experiments instead of AdamW in the baseline (See the comparison between two optimizer algorithm in Appendix).

Specifically, in AdamW, the update direction is $\frac{m_t}{\sqrt{v_t}}$, where v_t is the EMA of g_t^2; In AdaBelief, the update direction is $\frac{m_t}{\sqrt{s_t}}$, where s_t is the EMA of $(g_t - m_t)^2$. Intuitively, viewing m_t as the prediction of g_t, AdaBelief takes a large step when observation g_t is close to prediction m_t, and a small step when the observation greatly deviates from the prediction. Note that there is a extra bias correction item ϵ added to s_t, which we omitted in this theoretical experiment for simplicity.

Gradient Clipping: Secondly, after computing the gradients of each task, we clipped the gradients of the parameters of the model to a maximum $\mathcal{L}\_2$ norm of 1 before AdaBelief optimizer updating weights. The norm is computed over all gradients together, as if they were concatenated into a single vector. We adopted this gradient clipping approach to prevent gradient updates from adversely affecting previously-learned tasks through sequential learning settings in our baseline multi-task learning method.

During our experiments we noticed that by clipping the gradient to norm of 1 before AdaBelief optimizer step in sequential multi-task training, the individual and average performance among all tasks deteriorated. However, after we improved the loss functions on PARA and STS tasks and trained with gradient clipping and AdaBelief optimizer, the individual and average performance among all tasks significantly increased with most contribution from the PARA task. We only kept the AdaBelief optimizer in later experiments for two reasons: 1) AdaBelief scales the update direction by the change in gradient, which is related to the Hessian. Therefore, AdaBelief considers curvature information and performs better than AdamW. 2) Gradient clipping standardizes the update step size across parameters but can also alter the learning dynamics. Clipping gradients to a norm of 1 might be too restrictive given the small batch size and training data for these three particular downstream tasks, preventing the model from making significant updates where needed, especially in tasks that might require larger gradient steps for effective learning (can be seen from most performance drop on PARA task).

4.2 Loss Enhancement and Regularization

Our baseline approach utilizes cross-entropy loss for all three tasks in the multi-task learning setting. We also observed the overfitting problem particularly on the SST task, demonstrated by the decreasing performance on development dataset starting in early epochs (5 of 10). Therefore, to enhance the generalization of the model, we deployed following loss enhancements and regularization techniques for each task: For the PARA task, we adopted binary cross-entropy as the loss function. This enhancement are adopted for all MTL experiments. For the STS task, we calculated scaled cosine similarity to predict the similarity, and then used the mean squared error (MSE) as the loss function. This enhancement is adopted for all MTL experiments. For the SST task, we started experimenting with regularization techniques like L1 and SMART regularization (Jiang et al., 2020) on the single task fine-tuning. We did not adopt these regularization techniques in all MTL experiments as no significant improvement on the overfitting of the SST task was observed. Additionally, we performed hyperparameter tuning over dropout rate and learning rate to analysis whether any optimal hyperparameters can help mitigate the overfitting on the SST task specifically (See Section 6).

SMART Regularization: To prevent the fine-tuning process from deviating significantly from the pre-trained model's weights, we employed SMART regularization from Jiang et al. (2020) and customized the algorithm to handle data dimensions, for example, we did one-hot encoding for the SST labels and we used the $l_s(P, Q) = D_{KL}(P||Q) + D_{KL}(Q||P)$(symmetric KL-divergence) for the SST context. Specifically:

Smoothness-inducing adversarial regularization can help model achieve stable predictions by introducing small input variations and minimize a combined loss function $F(\theta)$ incorporating both the regular task loss $L(\theta)$ and a smoothness penalty $\lambda_s R_s(\theta)$:

$$\min_{\theta} F(\theta) = L(\theta) + \lambda_s R_s(\theta) \tag{2}$$

$$L(\theta) = \frac{1}{n} \sum_{i=1}^{n} l(f(x_i; \theta), y_i) \tag{3}$$

$$R_s(\theta) = \frac{1}{n} \sum_{i=1}^{n} \max_{||\tilde{x_i} - x_i||_p \leq \epsilon} l_s(f(\tilde{x_i}; \theta), f(x_i; \theta)) \tag{4}$$

Bregman proximal point optimization can efficiently handle the regularization term by restricting updates to a small region around the previous model state, specifically updating parameter $\theta_{t+1} =_\theta F(\theta) + \mu D_{Breg}(\theta, \tilde{\theta}_t)$, where:

$$D_{Breg}(\theta, \tilde{\theta}_t) = \frac{1}{n} \sum_{i=1}^{n} \left\{ l_s \left(f(x_i; \theta), f(x_i; \tilde{\theta}_t) \right) \right\} \tag{5}$$

$$\tilde{\theta}_t = (1 - \beta)\theta_{t-1} + \beta\tilde{\theta}_{t-1} \tag{6}$$

4.3 Dynamically Adapted Weighted Loss and Round-Robin Resampling

Through experiments so far, we observed that loss enhancements and dynamic optimizing using normalized gradients can partially reduce the overfitting in the sequential multi-task learning process. We also observed possibility of imbalance in training our MTL as some tasks dominate others during the training phase. This dominance can be attributed to variations in imbalanced data or task heuristics like complexities, uncertainties, and magnitudes of losses etc. Therefore an appropriate loss or prioritization strategy for all tasks in an MTL is a necessity.

To design an appropriate loss or prioritization strategy, we experimented two dynamically adapted weighted loss methods: the Geometric Loss Strategy (GLS) and Focused Loss Strategy (FLS) (Chennupati et al., 2019). In the experiments using these two methods, we implemented breaks within each batch (of size 8) where the batch update will stop if any of the task's data is exhausted. Given that this early stopping break did not fully use the PARA task data that is much larger than the other two tasks' data, we also implemented a round-robin batch-level resampling strategy where batches of size 32 are sampled round-robin from all three tasks in a fixed order from the start of training to the end.

Geometric Loss Strategy (GLS): The performance of the MTL network is highly dependent on their shared parameters as they contain the knowledge learned from different tasks. Inappropriate learning of these parameters can induce biased representations for a particular task which can hurt the performance of MTL networks. This phenomenon is referred to as negative transfer learning. In order to prevent it, there have been multiple methods on redefining loss functions using task heuristics or implementing more complex optimization process. However, we decided to adopt the GLS with minimal design complexities by multiplying geometric mean of losses of focused tasks to existing loss function. Specifically for our MTL example, within each parallel training (batch) the total loss (GLS) is the geometric mean of three individual task losses:

$$\mathcal{L}\_Total = \sqrt[3]{\mathcal{L}\_sentiment * \mathcal{L}\_paraphrase * \mathcal{L}\_sts}. \tag{7}$$

The GLS loss function acts as a dynamically adapted weighted arithmetic sum in log space, these weights act as regularizers and control the rate of convergence between the losses.

Focused Loss Strategy (FLS): Given the early stopping break implemented on GLS where PARA task could be largely underfitted, we further adapted our GLS loss function to give more attention to the PARA task by adopting FLS where we multiply geometric mean of losses of focused task (PARA) to existing loss function:

$$\mathcal{L}\_Total = \prod_{i=1}^{n} \sqrt[n]{\mathcal{L}\_i} \times \prod_{j=1}^{m} \sqrt[m]{\mathcal{L}\_j} \tag{8}$$

Here the tasks are ordered in terms of priority so that first m tasks (here we have $m = 1$ which is the PARA task), out of the total n tasks ($n = 3$ in our MTL) gets higher weights.

Round-Robin resampling: To further utilize all data from three tasks in the parallel training process so that we can obtain the best representation, we explored the Round-Robin resampling strategy in our experiments. Specifically, within each batch iteration, the model cycles through the SST, PARA, and STS data after loading in pre-trained BERT weights to update the relevant parameters for each dataset using a single pass. In cases that smaller datasets ran out of data (e.g. STS and SST datasets are smaller than the PARA dataset) in each batch iteration, the model will start resampling from the same dataset until exhausting remaining data for all datasets. The intuition behind that although the GLS/FLS loss functions act as a dynamically adapted weighted arithmetic sum in log space, these weights act as regularizers and control the rate of convergence between the losses. They do not directly address the issue of dataset size imbalance although can help mitigate its effects by prioritizing learning from the larger dataset (FLS for example).

5 Experiments

5.1 Data

We employed different distinct datasets, targeting a variety of prediction tasks:

- Stanford Sentiment Treebank (SST): Comprising 11,855 single-sentence movie reviews, this dataset is categorized into five sentiment classes: negative, somewhat negative, neutral, somewhat positive, and positive. It serves to fine-tune our model for sentiment analysis.
- Quora Question Pairs: With 400,000 question pairs labeled as paraphrase (true) or not (false), this dataset enables fine-tuning for paraphrase detection, a task critical for understanding semantic similarity between questions.

- SemEval STS Benchmark (STS): This dataset includes 8,628 sentence pairs, each annotated with a similarity score ranging from 0 (unrelated) to 5 (semantically equivalent), facilitating fine-tuning for semantic textual similarity assessment.

Additionally, for baseline single-task evaluation in sentiment analysis, we utilize the CFIMBD dataset, containing 2,434 movie reviews with binary sentiment labels (positive or negative).

5.2 Evaluation method

To evaluate different models in final reporting, we use accuracy rates for SST and PARA tasks and Pearson correlation for the STS task. In the model training process, we use the simple average of the accuracy rates for SST/PARA tasks and normalized correlation using the following formula: $Normalized\_corr = \frac{Pearson\_corr}{2} + 0.5$ to rescale it from 0 to 1.

5.3 Experimental details

Our experiments consists of three main types:

Pre-work/Baseline: For pre-work of single-task learning and baseline MTL, we pretrain the default MiniBERT with learning rate 1e-3, and then fine-tune using 10-epoch run with learning rate of 1e-5, a hidden-layer dropout probability of 0.3, and a batch size of 8.

Regularization and Hyperparameter tuning exploration: To address overfitting and enhance SST accuracy, we investigate SMART regularization and hyperparameter tuning.

MTL experiments: These MTL experiments are fine-tuned using 10-epoch run with a learning rate of 1e-5 and a hidden-layer dropout probability of 0.3. Except for the Round-Robin resampling where we set our batch size to be 32, other experiments have the batch size set as 8. Specifically for the AdaBelief optimizer, we use $\beta_1 = 0.9$ and $\beta_2 = 0.999$, which is the same as the AdamW in the baseline MTL approach.

5.4 Results

Pre-work/Baseline: Our baseline is the MLT fine-tuned model since our goal is to formulate a MTL approach where a single model is fine-tuned with all datasets altogether and can generalize well on all downstream tasks.

Table 1: Single task pretrain and fine-tune results

Model Type	SST Dev	PARA Dev	STS Dev	Total Dev
Single-task pretrain default	0.409	0.478	0.453	0.538
Single-task fine-tune default	0.513	0.890	0.774	0.764
MLT fine-tune (Baseline)	0.462	0.877	0.774	0.742

Table 1 shows the performance of our initial experiments: Three single-task models were fine-tuned on separate dev datasets and achieved comparable performance due to leveraging pre-trained parameters from the MiniBERT model. However, they lacked generalizability to unseen tasks, as evidenced by the significant drop in performance when applied directly to the multi-task learning.

Regularization and Hyperparameter tuning exploration: Given the observation that single-task fine-tuning on SST achieved a higher accuracy (0.513) than the multi-task baseline (0.462), we experimented regularization and hyperparameter tuning to deal with SST overfitting, which aims to create a strong foundation for leveraging SST knowledge in multi-task learning.

Table 2: SST fine-tune accuracy with different model head dropout rate

Dropout Rate	0	0.1	0.3	0.5	0.8	0.9
Best Accuracy	0.520	0.533	0.513	0.519	0.513	0.517
Mean Accuracy	0.508	0.510	0.507	0.501	0.500	0.499

Table 3: SST fine-tune accuracy with different learning rate

Learning Rate	1e-5	5e-6	1e-6
Best Accuracy	0.520	0.520	0.516
Mean Accuracy	0.508	0.507	0.487

Table 4: SST fine-tune accuracy with SMART regularization and different dropout rates

Dropout Rate	0	0.1	0.3
Best Accuracy	0.511	0.510	0.520
Mean Accuracy	0.497	0.507	0.493

Table 2 shows with fixed learning rate as 1e-5, the best accuracy (0.533) is achieved with a dropout rate of 0.1, higher dropout rates (above 0.3) might start to negatively impact the model's ability to learn effectively; Table 3 shows with a fixed dropout rate at 0, the best accuracy (0.520) and the highest mean accuracy (0.508) is achieved with a learning rate at 1e-5. A lower learning rate seems to lead to accuracy decrease; Table 4 shows with SMART regularization is applied(perturbation size=1e-5, learning rate=1e-3), the best accuracy (0.520) is achieved with a dropout rate of 0.3, while the highest mean accuracy (0.510) is achieved with a drop out rate of 0.1;

MTL experiments: Next we conducted the MTL fine-tuning experiments. Table 5 shows the MLT fine-tuned model experiments performance on dev data for each task.

Table 5: MLT fine-tune results

Model Type	SST Dev	PARA Dev	STS Dev	Total Dev
MLT finetune (Baseline)	0.462	0.877	0.774	0.742
MLT finetune+AdaBelief+GradClip	0.458	0.835	0.761	0.725
MLT finetune+AdaBelief+GradClip+LossImprove	0.491	0.88	0.732	0.746
MLT finetune+LossImprove	0.467	0.863	0.699	0.727
MLT finetune+AdaBelief+GLS+ParallelTrain	0.522	0.784	0.766	0.73
MLT finetune+AdaBelief+FLS+ParallelTrain	0.49	0.841	0.71	0.728
MLT finetune+AdaBelief+GLS+ParallelTrain with min batch size of 32+Round-robin resampling+LossImprove	0.507	0.866	0.676	0.737

We also outlined three models' performance on test data for each task in Table 6.

Table 6: Model performances on test data

Model Type	SST Test	PARA Test	STS Test	Total Test
MLT finetune (Baseline)	0.481	0.881	0.772	0.749
MLT finetune+AdaBelief+GradClip+LossImprove	0.479	0.881	0.703	0.737
MLT finetune+AdaBelief+GLS+ParallelTrain with min batch size of 32+Round-robin resampling+LossImprove	0.508	0.842	0.672	0.729

6 Analysis

Regularization and Hyperparameter tuning exploration

Employing SMART regularization (dropout=0.1, learning rate=1e-5) produced no noticeable improvement in accuracy over the non-regularized model, but the overall performance difference was not significant. Here are some potential reasons why SMART regularization might not have yielded improvement: 1) With small dataset, there might be limited information to learn smooth decision boundaries; 2) The chosen hyperparameters (e.g. perturbation size, the strength of the adversarial loss) might not be optimal, experimenting with different hyperparameter values could potentially lead to better results.

Despite limiting fine-tuning to only 10 epochs, we observed a trend where model accuracy degraded after a small number of epochs. This performance decline might be attributed to several factors: 1) The SMART regularization strength might be too high, for example a strong adversarial loss can excessively penalize the model, hindering its ability to learn the task effectively; 2) The chosen dropout rate might be too high, leading to excessive removal of neurons during training. This can significantly reduce the model's capacity to learn complex patterns and potentially lead to underfitting; 3) The combination of SMART Regularization and dropout might be overly aggressive for the small SST dataset. Both techniques aim to prevent overfitting, and together they might be unintentionally hindering the model's ability to learn from the limited data; 4) The learning rate might be too high with combined regularization and dropout techniques, causing the model to overshoot the optimal solution and potentially diverge during later epochs.

MLT fine-tune exploration

As are shown in Table 5, the highest accuracy on the SST task (0.522) was achieved through a configuration that employed AdaBelief, gradient layer scaling (GLS), and parallel training with breaks, which is contrary to initial expectations. This surpassed the performance attained using round-robin resampling for data augmentation (0.507). This suggests that resampling alone might not significantly improve model generalization on the SST task, potentially due to the limited introduction of new information during the resampling process.

For PARA, characterized by the largest training dataset, the best result was achieved through a configuration incorporating AdaBelief, gradient clipping with a norm of 1, and enhanced loss functions specifically designed for PARA and STS tasks. This configuration yielded an accuracy of 0.88. While further attempts explored utilizing parallel training with weighted losses or round-robin resampling to potentially boost performance, none surpassed this initial configuration. Notably, these findings suggest that for tasks with extensive training data, gradient clipping might be a more effective strategy compared to dynamic loss weighting within a parallel training framework. Additionally, utilizing the entirety of the available data appears to be more beneficial than employing a weighted sample.

For STS, despite extensive experimentation involving techniques such as enhanced loss functions and optimization process modifications, the baseline MTL configuration consistently achieved the best performance on the STS task (0.774). Although an experiment incorporating AdaBelief, GLS, and parallel training with breaks yielded a comparable correlation coefficient of 0.766, and sequential training with AdaBelief and gradient clipping achieved a score of 0.761, these results demonstrate that simpler techniques can be equally effective. Interestingly, the STS performance exhibited the most significant decline (compared to SST) when employing round-robin resampling within parallel training. This aligns with the observations for the SST task, suggesting that resampling might not offer substantial benefits for tasks with limited training data (like STS) compared to simpler techniques such as gradient clipping. We also see similar finding and learning on the task performance of selected experiments on test data as is shown in Table 6.

7 Conclusion

In conclusion, our exploration into MTL with miniBERT highlights the nuanced interplay between optimization techniques, loss function customization, and data handling strategies in achieving effective multi-task generalization. While certain approaches yielded notable improvements in task-specific areas, the quest for an all-encompassing MTL framework remains an open challenge. Looking ahead, future work will consider employing single-task models as teachers in a knowledge distillation framework to improve the MTL model's performance (Liu et al., 2019). Additionally, applying the FLS with round-robin resampling could offer further performance gains, especially for the STS task. Integrating the CFIMDB dataset alongside SST, through a relational layer, represents another avenue to enrich our multitask fine-tuning approach. These initiatives, while ambitious due to computational demands, promise to advance our understanding and application of MTL in NLP, paving the way for more nuanced and effective language processing models.

References

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*.

Rich Caruana. 1997. Multitask learning.

Zhao Chen, Vijay Badrinarayanan, Chen-Yu Lee, and Andrew Rabinovich. 2018. Gradnorm: Gradient normalization for adaptive loss balancing in deep multitask networks.

Sumanth Chennupati, Ganesh Sistu, Senthil Yogamani, Samir A Rawashdeh, Valeo North America, Valeo Vision Systems, and University of Michigan-Dearborn. 2019. Multinet++: Multi-stream feature aggregation and geometric loss strategy for multi-task learning.

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

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*. Association for Computational Linguistics.

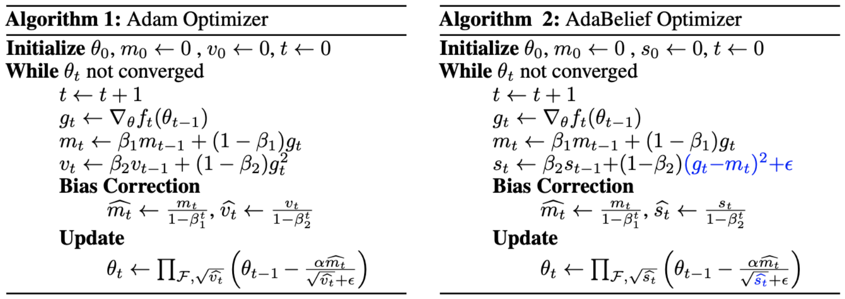
Xiaodong Liu, Pengcheng He, Weizhu Chen, Jianfeng Gao, Microsoft Research, and Microsoft Dynamics 365 AI. 2019. Improving multi-task deep neural networks via knowledge distillation for natural language understanding.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. 2013. On the difficulty of training recurrent neural networks.

Juntang Zhuang, Tommy Tang, Yifan Ding, Sekhar Tatikonda, Nicha Dvornek, Xenophon Papademetris, and James S. Duncan. 2020. Adabelief optimizer: Adapting stepsizes by the belief in observed gradients.

A AdamW vs Adabelief Optimizer

Figure 1: Comparison between AdamW Optimizer and AdaBelief Optimizer. Source: AdaBelief Optimizer: Adapting Step sizes by the Belief in Observed Gradients.



```
Algorithm 1: Adam Optimizer
Initialize $\theta_0, m_0 \leftarrow 0$ , $v_0 \leftarrow 0, t \leftarrow 0$
While $\theta_t$ not converged
    $t \leftarrow t+1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$
    $m_t \leftarrow \beta_1 m_{t-1} + (1-\beta_1) g_t$
    $v_t \leftarrow \beta_2 v_{t-1} + (1-\beta_2) g_t^2$
    Bias Correction
        $\widehat{m_t} \leftarrow \frac{m_t}{1-\beta_1^t}, \widehat{v_t} \leftarrow \frac{v_t}{1-\beta_2^t}$
    Update
        $\theta_t \leftarrow \prod_{\mathcal{F},\sqrt{\widehat{v_t}}} \left(\theta_{t-1} - \frac{\alpha \widehat{m_t}}{\sqrt{\widehat{v_t}}+\epsilon}\right)$
```

```
Algorithm 2: AdaBelief Optimizer
Initialize $\theta_0, m_0 \leftarrow 0$ , $s_0 \leftarrow 0, t \leftarrow 0$
While $\theta_t$ not converged
    $t \leftarrow t+1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$
    $m_t \leftarrow \beta_1 m_{t-1} + (1-\beta_1) g_t$
    $s_t \leftarrow \beta_2 s_{t-1} + (1-\beta_2)(g_t - m_t)^2 + \epsilon$
    Bias Correction
        $\widehat{m_t} \leftarrow \frac{m_t}{1-\beta_1^t}, \widehat{s_t} \leftarrow \frac{s_t}{1-\beta_2^t}$
    Update
        $\theta_t \leftarrow \prod_{\mathcal{F},\sqrt{\widehat{s_t}}} \left(\theta_{t-1} - \frac{\alpha \widehat{m_t}}{\sqrt{\widehat{s_t}}+\epsilon}\right)$
```