Less is More: Exploring BERT and Beyond for Multitask Learning

Stanford CS224N Default Project

Yichun Qian
Department of Electrical Engineering
Stanford University
ycqian@stanford.edu

Liuxin Yang
Department of Electrical Engineering
Stanford University
lyang822@stanford.edu

Abstract

In the prevailing pre-training-then-fine-tuning framework for NLP tasks, multitask learning demonstrates superior efficiency and generalization capabilities compared to models fine-tuned for individual tasks. This study presents the development of a multitask learning model based on BERT, aimed at concurrently improving performance across three distinct downstream tasks. We investigate the impact of incorporating two variants of task-specific layers and introduce a suite of enhancement techniques, including 1) round-robin training approach, 2) SMART regularization, 3) projected attention layers, and 4) additional pre-training with contrastive loss. Our analysis reveals that, although each enhancement contributes to performance improvements to some extent, the architecture of task-specific layers plays a pivotal role in maximizing the model's effectiveness.

1 Key Information to include

Mentor: Heidi Zhang Default Project Contribution: 50% of each team member

2 Introduction

Recent advancements in pretrained language models, exemplified by BERT (Devlin et al., 2018) and furthered by large-scale models like ChatGPT, have significantly enhanced performance across a myriad of natural language processing (NLP) tasks. These developments underscore the effectiveness of the pre-training and fine-tuning paradigm in achieving state-of-the-art results in diverse NLP applications. Historically, each task was approached with the development of specialized models, which not only demanded substantial computational resources but also often lacked in generalizability.

Multitask learning emerges as a potent alternative, enabling a single model to learn multiple tasks concurrently. Liu et al. (2019b) leverages BERT embeddings and further learns the representation using multi-task objectives. Although multi-task fine-tuning may not enhance the performance of the model on single tasks (Sun et al., 2019), it allows models to leverage knowledge learned from related tasks to deliver a more generalized representation, which is desirable in terms of natural language understanding and robust to domain adaptation when the available in-domain labels are substantially fewer (Liu et al., 2019b).

Building on the multitask learning framework, this paper introduces an experimental setup that extends the traditional BERT model with novel task-specific layer designs and training strategies aimed at improving performance across three key NLP tasks: sentiment analysis, paraphrase detection, and semantic textual similarity. Our contributions are as follows:

- Investigation of two distinct task-specific layer designs, employing both sequential and round-robin training routines, to evaluate their efficacy under various conditions.

- Implementation of 3 other extensions: SMART regularization (Jiang et al., 2019), projected attention layers (PAL) (Stickland and Murray, 2019) and additional pre-training with contrastive loss (Gao et al., 2021), comparing quantitative metrics comprehensively.
- An in-depth analysis of model success and limitations, complemented by qualitative examples illustrating model predictions beyond quantitative metrics.

Our findings reveal that surprisingly, the architecture of task-specific layers significantly influences overall performance, whereas the benefits of SMART regularization, PAL and additional pre-training with contrastive loss are contingent on task-specific layers design. Our final results **achieve 12th place on the test leaderboard**.

3 Related Work

How to efficiently and effectively finetune pretrained language models on various downstream tasks has been a keen interest in the field of NLP. We briefly review two related approaches: multitask learning and avoid overfitting.

Multitask Learning Across literature, most approaches use some kind of multitask learning (MTL) approach in combination with increasingly advanced language models such as LSTM, BERT, RoBERTa (Liu et al., 2019c) and ALBERT (Lan et al., 2019). Guo et al. (2018) improve abstractive summarization via multi-task learning with two auxiliary tasks of question generation and entailment generation, while MT-DNN differs them in that all tasks have equal status, with a novel paradigm where the lower embedding layers are shared across all tasks while the top layers are task-specific. Liu et al. (2019a) also apply the knowledge distillation method (Hinton et al., 2015) in the multi-task learning setting, to train a student model from an ensemble of MT-DNN teacher models. Apart from modifications on task-specific layers and focusing on knowledge transfer between tasks in different domains, which Sun et al. (2019) has pointed out less effective than expected, some methods aim to strengthen the underlying BERT model layers. The projected attention layer (PAL) model (Stickland and Murray, 2019), by adding a low-dimensional multi-head attention layer in parallel to each original BERT layer, it can achieve similar performance as fine-tuned models with much fewer parameters. In contrast, Houlsby et al. (2019) adds trainable adapter modules per task within each BERT layer and fixate original network parameters, yielding a extensible and high-performance model.

Avoid Overfitting Avoiding overfitting in NLP is crucial for developing robust, generalizable models. Howard and Ruder (2018) utilizes discriminative fine-tuning (tune each layer with different learning rates) and slanted triangular learning rates to enable model to quickly converge to a suitable starting point. The learning rate warmup heuristic can also achieve remarkable success in stabilizing training, accelerating convergence and improving generalization for adaptive stochastic optimization algorithms like Adam (Kingma and Ba, 2014). Moreover, adding a regularization term to the loss function is a common technique. Jiang et al. [2] proposes a a smoothness-inducing regularization technique (SMART) which effectively manages the complexity of the model and can prevent aggressive updating. Compared to other techniques stated above, this approach achieves parallelable benchmark scores and does not require significant hyperparamter tuning, making it resource-efficient when put into practice.

4 Approach

4.1 Baseline Model

We first implement a min-BERT (Devlin et al., 2018) as described in the handout, initializing it with pre-trained weights from the $BERT_{base}$ model. The experiment runs in two modes: pretrain and finetune, differing in whether minBERT weights are frozen. In both, task-specific layers' parameters are updated. Next, we delve into our task-specific layers and loss functions designs.

Sentiment Analysis We design a feedforward neural network with a dropout layer, two linear layers and GELU activation function in between. The intermediate hidden size is chosen as 512. We use cross-entropy loss as our loss function. See details in Figure 1.

Paraphrase Detection We explore two preprocessing approaches for sentence pairs. Inspired by Sentence-BERT (Reimers and Gurevych, 2019), our first design obtains separate BERT embeddings

for each sentence in a pair. These embeddings are then concatenated as $[embed_1, diff, embed_2]$, where $diff$ denotes the element-wise absolute difference between two embeddings. This concatenated vector is subsequently fed into a similar feedforward neural network as above. Our second design begins by concatenating token embeddings and mask embeddings directly, and then obtain a unified embedding from BERT. Similarly, this representation is then sent into a feedforward neural network for subsequent tasks. See details in Figure 1.

Semantic Textual Similarity Same as Paraphrase Detection, except that we use mean-squared error loss as our loss function. See details in Figure 1.

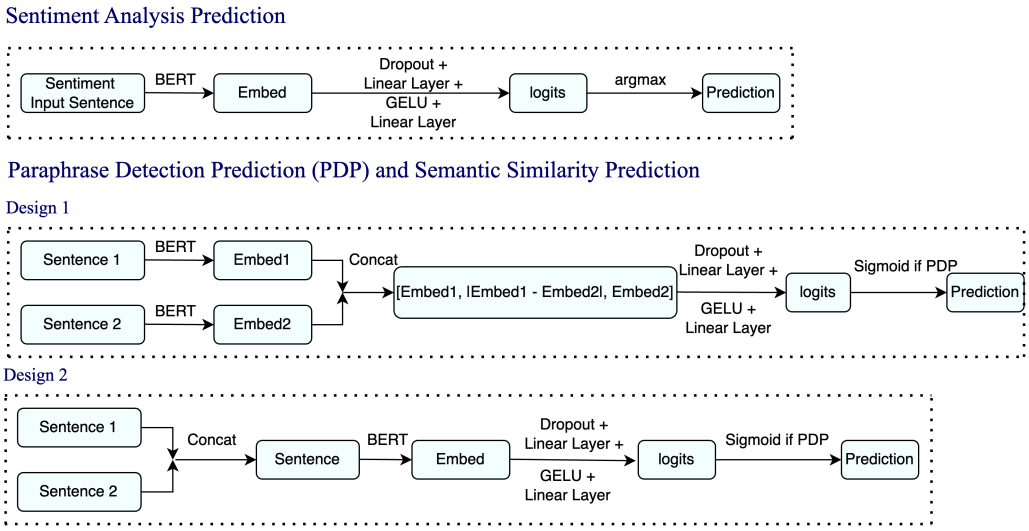

Figure 1: Task-specific Layers Design

4.2 Batch-Level Round-Robin

Beyond the naive sequential training approach, where fine-tuning is conducted on one dataset entirely before another, we employ a `batch-level round-robin` routine. To be specific, we adopt a cycling approach through datasets from the SST, QQP, and STS, while maintaining a consistent batch size. During each batch iteration, we update model's parameters pertinent to each specific dataset using task specific loss mentioned above. The intuition behind is model parameters will achieve better alignment across all tasks, as opposed to favoring the most recently addressed tasks. This nuanced approach aims to foster a more balanced and effective learning process.

4.3 Regularization of Loss

To prevent overfitting in the fine-tuned models, a common problem that leads to poor performance on testsets, we adopt a smoothness-inducing regularization technique - `SMART` (Jiang et al., 2019) to effectively manage the complexity of our model. This technique relies on Smoothness-inducing Adversarial Regularization, which adds small perturbations to the input at training while trying to maintain minimal changes to the output of the model, thus ensuring the overall model smoothness. Specifically, the objective function to minimize is

$$min_\theta F(\theta) = L(\theta) + \lambda R_s(\theta)$$

where $L(\theta) = \frac{1}{n}\sum_{i=1}^{n} l(f(x_i;\theta), y_i)$, the original loss function depending on the target task, and $R_s(\theta) = \frac{1}{n}\sum_{i=1}^{n} max_{||\tilde{x_i}-x_i||_p \leq \epsilon} l_s(f(\tilde{x_i};\theta), f(x_i;\theta))$, and λ is a weight parameter equals to 5 by design choice. For the smoothness-inducing adversarial regularizer l_s, we choose `symmetric KL-divergence` for classification tasks and `squared error` for regression tasks. We refer to https://github.com/archinetai/vat-pytorch for code implementation, with significant changes of adaptation.

4.4 Projected Attention Layers

In our modification to the original BERT model, we have introduced a Projected Attention Layer (PAL1) alongside the first self-attention layer (SA1) in the BertLayer (see Figure 2). The outputs from SA1 and PAL1 are concatenated and then passed through a linear layer to reduce the dimension back to the original hidden_size. In the second part of the forward pass, we have used only the second self-attention layer (SA2) without a corresponding PAL2. The output of SA2 is directly used as the final output of the BertLayer.

To better adapt PAL into our task, we **made some unique changes to the standard BERT+PAL** (`https://github.com/AsaCooperStickland/Bert-n-Pals`). In the standard BERT+PAL, the outputs from the PALs are directly added to the outputs from the self-attention layers. However, in our model, we concatenate the outputs from SA1 and PAL1 and then use a linear layer to reduce the dimension. Moreover, the standard BERT+PAL uses PAL2 in the second part of the forward pass, whereas we have omitted PAL2 entirely. The reason for not using two PALs is to avoid overfitting. Our experiments show that adding too many PALs results in worse performance due to overfitting since we increase the model complexity and the number of parameters. By using only one PAL, we are effectively regularizing the model and preventing it from becoming too complex and overfitting to the training data.

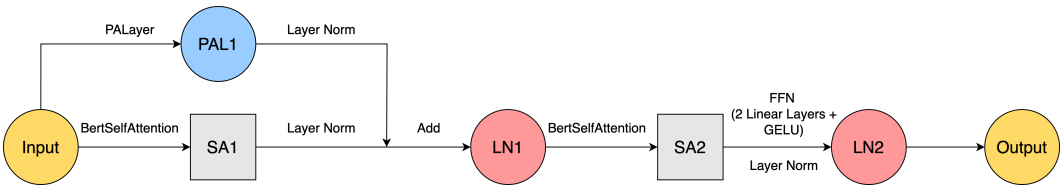

Figure 2: BERT + PAL

The PAL operates in a lower-dimensional space, allowing our nodel to learn task-specific patterns more efficiently. This is particularly useful when we have limited computational resources or when we want to avoid overfitting by not modifying the entire BERT architecture. The PAL can help in capturing task-specific features and sentence-level relationships, which are crucial for sentiment analysis, paraphrase detection, and semantic textual similarity. By incorporating PAL, we are essentially providing the model with an additional mechanism to focus on the relevant aspects of the input data for these specific tasks. Moreover, our modifications to the standard BERT+PAL architecture, such as using only one PAL and concatenating its output with the self-attention output, are aimed at striking a balance between model complexity and performance. These changes are motivated by the desire to avoid overfitting and to efficiently adapt the model to the downstream tasks, while still leveraging the power of the pre-trained BERT model.

4.5 Additional Pretraining with Contrastive Loss

We also extend our pipeline to two stages. The first stage is the supervised contrastive pretraining using 15% of the labeled data from the QQP dataset. **Our contribution includes converting unsupervised contrastive learning into supervised contrastive pretraining.** After additional pretraining, we save the learned model weights as a checkpoint file (`*.pt`). The second stage is the multitask fine-tuning. We load the pretrained model from `stage 1` and fine-tune it on the remaining 85% of the data on three tasks simultaneously. Now we will delve into the details of the `stage 1`.

We use the Simple Contrastive Learning of Sentence Embeddings (SimCSE) with the contrastive loss function guides the model to learn sentence representations that can benefit downstream tasks, leverage the pretrained weights for improved performance, and efficiently utilize a small portion (15%) of labeled data to incorporate additional knowledge. The SimCSE model consists of a BERT-based encoder followed by a pooling layer to obtain sentence embeddings. We use the `[CLS]` token representation to obtain the final sentence embeddings.

The additional pretraining process involves loading the training and validation datasets, tokenizing the sentences, and creating data loaders with the specified batch size. The SimCSE model is initialized with the pretrained BERT model and trained using the `AdamW` optimizer.During each epoch, the model is trained on the training data, and the average training loss is calculated. The model is then evaluated

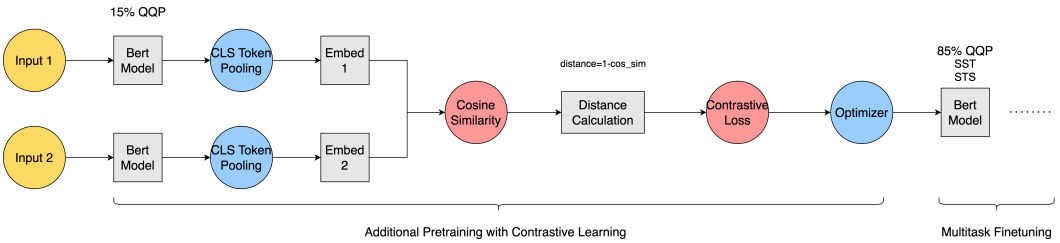

Figure 3: Additional Pretraining with Contrastive Loss

on the validation data. If the validation loss improves, the best model checkpoint is saved. Early stopping with a patience of 5 epochs is used to terminate the training if no improvement is observed.

The contrastive loss function is designed to minimize the distance between embeddings of similar sentence pairs while maximizing the distance between embeddings of dissimilar pairs. The loss function is defined as follows: $L_i(x_{1i}, x_{2i}, y_i) = 0.5 \cdot (y_i \cdot d_i^2 + (1 - y_i) \cdot (\max(0, m - d_i))^2)$, where d_i is the Euclidean distance between the embeddings of the sentence pair (x_{1i}, x_{2i}), y_i is the binary label indicating whether the sentences are similar or not, and m is the margin hyperparameter set to 0.5. By minimizing this loss function, the model learns to generate similar embeddings for semantically equivalent sentences and dissimilar embeddings for non-equivalent sentences.

5 Experiments

5.1 Data

The minBERT model is pretrained on Wikipedia articles. When fine-tuning, we use the Stanford Sentiment Treebank (SST) for the sentiment analysis, Quora for the paraphrase detection, and SemEval STS Benchmark (STS) for the semantic textual similarity. SST (Socher et al., 2013) includes 11,855 single-sentence reviews categorized from negative to positive sentiment. Quora contains 400,000 pairs of questions, each pair marked as a paraphrase or not. STS (Agirre et al., 2013) contains 8,628 pairs of sentences, each rated from 0 to 5.

5.2 Evaluation method

Following the default project handout, our evaluation for the SST sentiment classification and Quora paraphrase detection uses `accuracy`. For the STS semantic textual analysis task, we use the `Pearson correlation` to measure the agreement between the actual similarity scores and predictions. The correlation score is then adjusted to fall within [0, 1]. We also consider the average of these three metrics as an evaluation method.

5.3 Experimental details

All experiments share some general settings: In the pre-train mode, we use a learning rate of 10^{-3}, and in the fine-tune mode, a learning rate of $1e-5$. The batch size is set to 32, reduced to 16 if constrained by GPU memory. We train models for 10 epochs and then pick the best model on the dev set. The training time of all experiments except for pre-train mode is about 10 hours each.

Regularization of Loss and Optimizer Step Beyond the original loss per task, in Table 3, we add SMART regularization to our baseline model. For the SMART loss, following (Jiang et al., 2019), we set $\eta = 10^{-3}$, $\epsilon = 10^{-6}$, $\sigma = 10^{-5}$, $\lambda = 5$ for the weight of regularization term. In all experiments, we use the ADAM optimizer to efficiently update the parameters.

Projected Attention Layers The key component of PAL is the additional attention mechanism introduced in the BertLayer, implemented using the `PALAttention` and `PALayer` classes. For the PAL experiments, we used the same hyperparameter settings as the baseline BERT model

Additional Pretraining with Contrastive Loss We use the 'bert-base-uncased' pretrained model as the backbone for the SimCSE model. The sentences are tokenized using the `BertTokenizer`

with a maximum sequence length of 64. The model is trained with batch size of 64 and run for 20 epochs with $\eta = 1e-5$ for the `AdamW` optimizer. The margin parameter for the contrastive loss function is set to 0.5. We use the early stopping with a patience of 5 epochs to prevent overfitting. The contrastive loss function used in the SimCSE is calculated based on the cosine similarity between the embeddings of sentence pairs.

5.4 Results

Initially, we evaluated our baseline models featuring two task-specific layer variations, employing the round-robin training routine. Results are shown in table 1 and table 2. By comparing with and without round-robin training routine within each design, as expected, models with round-robin boost performance on the SST task, since it evenly distributes parameter updates across tasks, effectively mitigating catastrophic forgetting compared to traditional sequential training. However, the effectiveness of round-robin training is somewhat constrained by the varying sizes of the datasets involved. A comparison of the two task-specific layer designs reveals that the combined embedding approach (design 2) consistently outperforms the other across all tasks. Given that the SST task classifiers remained unchanged across both designs, this indicates that the superior representations learned from design 2 on the other tasks also contributed to improved performance on the SST task.

Table 1: Performance on Dev Datasets Using Design 1 (RR = Round Robin)

	SST Acc.	Paraphrase Acc.	STS Corr.	Overall Score
Pretrain w/ RR	0.391	0.695	0.317	0.582
Finetune w/ RR	**0.495**	**0.859**	0.733	**0.740**
Pretrain w/o RR	0.391	0.700	0.321	0.584
Finetune w/o RR	0.474	0.868	**0.741**	0.738

Table 2: Performance on Dev Datasets Using Design 2 (RR = Round Robin)

	SST Acc.	Paraphrase Acc.	STS Corr.	Overall Score
Pretrain w/ RR	0.391	0.704	0.473	0.611
Finetune w/ RR	**0.511**	**0.887**	0.841	0.773
Pretrain w/o RR	0.389	0.681	0.449	0.598
Finetune w/o RR	0.499	0.884	**0.881**	**0.774**

Table 3 shows SMART regularization did not enhance performance, especially impairing STS results, possibly due to interference from non-learnable loss introduced by the `Dropout` layer in our classifier design. Additionally, the STS task's reliance on nuanced similarity scores may exacerbate correlation gaps. Comparing two classifier designs, `design 2` outperforms `design 1` and demonstrates greater robustness, likely because it is more end-to-end and less reliant on feature engineering.

Table 3: Performance Across Configurations (Classifer Designs and SMART)

Configuration	SST Acc.	Paraphrase Acc.	STS Corr.	Overall Score
design 1 w/o SMART	0.495	0.859	0.733	0.740
design 1 w/ SMART	0.474	0.836	0.487	0.685
design 2 w/o SMART	**0.511**	**0.887**	**0.841**	**0.773**
design 2 w/ SMART	0.507	0.885	0.816	0.766

Table 4 compares the performance of two classifier designs and the impact of the PAL on three tasks. The results show that round-robin training and design 2 consistently outperform design 1 without round-robin across all tasks, indicating the effectiveness of concatenating tokens and mask embeddings to obtain a unified embedding from BERT.

Table 5 presents the performance of different classifier designs and the impact of additional pretraining. The results demonstrate that design 2 generally outperforms design 1 across all tasks. Also, round-robin technique is especially useful when combined with additional pretraining, probably because

Table 4: Performance Across Configurations (Classifier Designs and PAL)

Configuration	SST Acc.	Paraphrase Acc.	STS Corr.	Overall Score
design 1 + PAL	0.443	0.861	0.774	0.730
design 1 + RR + PAL	0.466	0.864	0.753	0.735
design 2 + PAL	0.470	0.867	**0.869**	0.757
design 2 + RR + PAL	**0.500**	**0.881**	0.849	**0.768**

during the second stage of training, the size of three datasets is more balanced than applying this technique to original datasets. Comparing the results with Table 4, we can observe that the best-performing configuration in Table 5 (design 2 + RR + SimCSE) slightly outperforms the best configuration in Table 4 (design 2 + RR + PAL) in terms of overall score (0.772 vs. 0.768) and SST accuracy (0.511 vs. 0.500). This suggests that the additional pretraining with SimCSE may provide some advantages over the incorporation of PAL in certain tasks.

In summary, all experiments highlights the effectiveness of design 2 and the benefits of round-robin training in a balanced data size setting.

Table 5: Performance Across Configurations (Classifier Designs and SimCSE)

Configuration	SST Acc.	Paraphrase Acc.	STS Corr.	Overall Score
design 1 + SimCSE	0.464	0.857	0.739	0.730
design 1 + RR + SimCSE	0.483	0.861	0.753	0.740
design 2 + SimCSE	0.448	**0.881**	**0.874**	0.755
design 2 + RR + SimCSE	**0.511**	0.880	0.850	**0.772**

Our final ensembled best model results on the dev and test leaderboard are:

Table 6: Final Results on the Dev and Test Leaderboard

	SST Acc.	Paraphrase Acc.	STS Corr.	Overall Score
dev	0.511	0.887	0.881	0.780
test	0.520	0.887	0.880	0.783

6 Analysis

We first dive deep into the prediction results and examine some examples of three tasks.

Sentiment Classification (SST) The prediction and ground truth distribution of SST labels is shown in Figure 4. The sentiment label distribution in the SST dataset reveals a skew towards moderate sentiments over extreme ones, complicating sentiment analysis due to the nuanced nature of much of the data. Our model, however, leans towards predicting extreme sentiments, illustrating its challenge with subtlety. For instance, "There's ... tremendous energy from the cast, a sense of playfulness and excitement that seems appropriate." is labeled as 3 (somewhat positive), because "seems appropriate" shows reservation. However, our model predicts it as 4 (strong positive), probably because it is misled by many positive words like "tremendous" "playfulness" "excitement" as to overlook the nuanced sentiment implied by "appropriate". There are also cases which are hard even for human judgement. "It's a coming-of-age story we've all seen bits of in other films – but it's rarely been told with such affecting grace and cultural specificity." is labeld as 2 (neutral) in the dataset, but many people would agree with the prediction 4, as we can tell the positiveness from this review.

Paraphrase Detection (Quora) Our model shows strong performance on the QQP task, but struggles with subtleties, much like humans. For example, "What are some really cool working science models I can prepare for my school science exhibition?" and "I am in 10th grade. I need to make a working model for my science exhibition. What can I make?" do not differ too much unless we take the motivation behind them into consideration. Our model predicts this as a pair

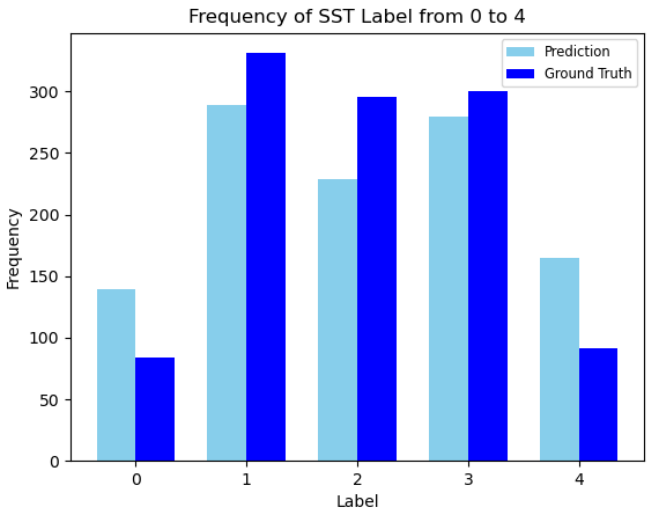

Figure 4: Distribution of SST Labels (Prediction vs. Ground Truth)

of paraphrase, probably because it focuses mostly on the phrase overlap and is not able to capture complex implicit meaning behind the sentences. This issue, common to advanced models including ChatGPT, highlights challenges in detecting nuanced meanings behind similar sentences.

Textual Semantic Similarity (STS) Experimenting with various task-specific layer designs, we discovered that mirroring the STS task's design with QQP's benefits STS, given their inherent similarity. This shared architecture facilitates knowledge transfer from QQP to STS. Our predictions closely match ground truths, usually within a deviation of less than 1. However, mismatches like between "Work into it slowly." and "It seems to work."—rated 2.6 by our model but irrelevant in reality—indicate a possible overreliance on word overlap, especially in short sentences.

In assessing the superior performance of our fine-tuned model, two primary factors emerge as pivotal. First, the design of our task-specific layers is demonstrably well-aligned with the tasks, evidenced by comparing two designs throughout experiments above. Also, we conduct ablation studies where reducing the hidden layer size to 256 or simplifying from two linear layers to one fails to outperform our optimal configuration. Second, the inherent complexity of our baseline model appears to be a double-edged sword. While it provides a robust foundation for task performance, its complexity also cause overfitting when additional parameters and techniques are introduced, as the training accuracy of the SST can reach above 0.7 and around 0.95 for other two tasks. When we reduce hidden layer size to 256 or remove two linear layers to one, this negative effect does shrink. Meanwhile, BERT's hidden layer and attention probability dropouts may be sufficiently regularizing on their own, and the benefit of SMART regularization gets drowned out from the non-learnable noise introduced by dropout layers, making this technique especially unsuitable for our model structure.

7 Conclusion

In our project, we developed multitask learning models with two task-specific layer designs, assessing their efficacy across three NLP tasks through various enhancements: round-robin training, SMART regularization, projected attention layers, and contrastive loss pretraining. Our findings highlight task-specific layer design as the key performance driver, with our models achieving average scores of **0.780** on the dev set and **0.783** on the test set. Yet, this emphasis on layer design also underscores our study's main limitation, as other architectural modifications showed limited impact on performance. Looking ahead, we plan to 1) test the removal of the Dropout layer in task-specific designs to validate our hypotheses further, and 2) experiment with different projected attention layer variations and the Adapter module (Houlsby et al., 2019) to evaluate their potential benefits.

References

Eneko Agirre, Daniel Cer, Mona Diab, Aitor Gonzalez-Agirre, and Weiwei Guo. 2013. * sem 2013 shared task: Semantic textual similarity. In *Second joint conference on lexical and computational semantics (* SEM), volume 1: proceedings of the Main conference and the shared task: semantic textual similarity*, pages 32–43.

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

Tianyu Gao, Xingcheng Yao, and Danqi Chen. 2021. Simcse: Simple contrastive learning of sentence embeddings. *arXiv preprint arXiv:2104.08821*.

Han Guo, Ramakanth Pasunuru, and Mohit Bansal. 2018. Soft layer-specific multi-task summarization with entailment and question generation. *arXiv preprint arXiv:1805.11004*.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for nlp. In *International conference on machine learning*, pages 2790–2799. PMLR.

Jeremy Howard and Sebastian Ruder. 2018. Universal language model fine-tuning for text classification. *arXiv preprint arXiv:1801.06146*.

Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Tuo Zhao. 2019. Smart: Robust and efficient fine-tuning for pre-trained natural language models through principled regularized optimization. *arXiv preprint arXiv:1911.03437*.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2019. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*.

Xiaodong Liu, Pengcheng He, Weizhu Chen, and Jianfeng Gao. 2019a. Improving multi-task deep neural networks via knowledge distillation for natural language understanding. *arXiv preprint arXiv:1904.09482*.

Xiaodong Liu, Pengcheng He, Weizhu Chen, and Jianfeng Gao. 2019b. Multi-task deep neural networks for natural language understanding. *arXiv preprint arXiv:1901.11504*.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019c. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

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

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

Asa Cooper Stickland and Iain Murray. 2019. Bert and pals: Projected attention layers for efficient adaptation in multi-task learning. In *International Conference on Machine Learning*, pages 5986–5995. PMLR.

Chi Sun, Xipeng Qiu, Yige Xu, and Xuanjing Huang. 2019. How to fine-tune bert for text classification? In *Chinese Computational Linguistics: 18th China National Conference, CCL 2019, Kunming, China, October 18–20, 2019, Proceedings 18*, pages 194–206. Springer.