Beyond Fine-tuning: Iterative Ensemble Strategies for Enhanced BERT Generalizability

Stanford CS224N Default Project

Shreya D'Souza
Department of Computer Science
Stanford University
shreya99@stanford.edu

Riya Dulepet
Department of Computer Science
Stanford University
riyadule@stanford.edu

Megan Dass
Department of Computer Science
Stanford University
mdass9@stanford.edu

Abstract

The Bidirectional Encoder Representations from Transformers (BERT) model has revolutionized natural language processing (NLP) by providing a contextual understanding of human language. Leveraging BERT's capabilities, we pre-train and fine-tune the model to specialize in generating sentence embeddings, enabling efficient application in various downstream tasks such as sentiment analysis, paraphrase detection, and semantic textual similarity. We experiment with different fine-tuning strategies and advanced techniques, including multi-task classification, gradient surgery, cosine similarity, and ensemble modeling. Our results demonstrate significant improvements in performance across all tasks, with ensembling emerging as a particularly effective technique. Notably, we introduce an iterative ensembling approach, stacking layers of ensemble models to achieve an overall test set performance of 71.6%.

1 Key Information

Mentor: Annabelle Tingke Wang; *External Collaborators*: None; *Sharing project*: No

2 Introduction

The Bidirectional Encoder Representations from Transformers (BERT) model (Devlin et al., 2019) has revolutionized the field of natural language processing (NLP), setting a new standard for understanding and interpreting human language. Its foremost advantage lies in its ability to contextually analyze text from both directions, unlike previous models that processed text in a linear sequence. This bidirectional understanding enables BERT to capture the nuances and complexities of language, making it highly effective for a vast array of NLP tasks.

Leveraging BERT's powerful baseline, we fine-tune and extend the model to specialize in generating sentence embeddings. These embeddings are designed to encapsulate the semantic essence of sentences, allowing for their application in a variety of downstream tasks with remarkable efficiency. By fine-tuning BERT on specific datasets and employing advanced techniques, we enhance its capability to excel in sentiment analysis, paraphrase detection, and measuring semantic textual similarity.

3 Related Work

Multi-task learning (MTL) provides a way to use a single model for multiple downstream tasks, removing the need to train separate models for each task. While BERT helped to make significant strides in NLP by using pretrained word and paragraph embeddings, there is room for improvement in BERT's performance on downstream tasks (Sun et al., 2020).

Bi et al. (2022) propose a method of calculating loss for multi-task learning as the sum of the losses for each of the individual tasks. We use this as our initial approach of incorporating all downstream tasks in our fine-tuning. Yu et al. (2020) posit that this approach does not consider a key optimization issue in multi-task learning - conflicting gradients from each of the downstream tasks, which can be detrimental in the case of varying magnitudes and directions (Yu et al., 2020). Hence, we investigate the efficacy of their method, *gradient surgery*, in improving MTL performance. When two tasks have conflicting gradients (negative cosine similarity), gradient surgery can be used to project the gradient of one task onto the normal plane of the gradient of another task. If they do not conflict, the gradients are left unchanged so as to not create assumptions about the model (Yu et al., 2020).

In addition to optimizing overall MTL results, methods can enhance performance on individual downstream tasks. We employ cosine similarity coupled with cosine embedding loss to enhance performance on the paraphrase detection and semantic text similarity tasks, where sentence similarity is crucial. Cosine similarity quantifies the angle between two vectors in a high-dimensional space, determining the similarity of word embeddings. Cosine embedding loss promotes embeddings with high similarity to encourage the model to accurately capture semantic relationships (Reimers and Gurevych, 2019). This approach, demonstrated by Reimers et al., leads to improved performance in downstream tasks, particularly semantic textual similarity.

4 Approach

4.1 MinBERT Model Architecture

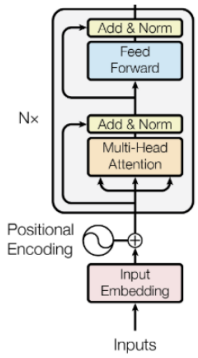

Figure 1: Encoder Layer of Transformer used in BERT

The minBERT architecture closely resembles BERT (Devlin et al., 2019) in its tokenization process, utilizing a WordPiece tokenizer to convert input sentences into word piece tokens and padding them to a maximum length of 512 with [PAD] tokens. Unseen word pieces are represented by the [UNK] token, and the [CLS] token is prepended to represent the entire sentence. Additionally, the [SEP] token is used to separate input sentences. The embedding layer comprises token embeddings, segmentation embeddings (not relevant in our case), and positional embeddings, each with a dimensionality of 768. Token embeddings map input IDs to vector representations and positional embeddings encode word positions within sentences.

The BERT Transformer Layer consists of 12 Encoder Transformer layers, incorporating multi-head attention, position-wise feed-forward networks, and dropout. Multi-head attention allows the model to attend to information from different representation subspaces at various positions by computing dot products of queries and keys, followed by normalization. Position-wise feed-forward networks

include linear transformations with ReLU activation, followed by normalization. Dropout is applied to the output of each sub-layer and the sums of embeddings and positional encodings.

The BERT model outputs contextualized embeddings for each word piece in the sentence from the last BertLayer as the last hidden state, along with the embedding of the [CLS] token as pooler output.

4.2 Baseline

Our baseline BERT is pre-trained and fine-tuned on the SST and CFIMDB sets.

We employ the AdamW optimizer to efficiently optimize model parameters, a method that updates exponential moving averages of the gradient and squared gradient at a particular timestep, using parameters β_1 and β_2 to control the rate of exponential decay. We compute bias correction to eliminate the bias of the averages towards 0 (Loshchilov and Hutter, 2019).

4.3 Multi-task Learning

Our first extension fine-tunes the model by summing the losses for each downstream task. Multi-task fine-tuning involves training a single model on multiple tasks by sharing the model's layers (except for the task-specific output layers) across all tasks; the model learns a representation applicable to all tasks. The multi-task classifier builds on the baseline model by similarly using the AdamW optimizer.

We use a round-robin approach to simultaneously train three batches (one for each downstream task) and obtain three losses. The loss used to update model parameters is the sum of cross-entropy loss for the sentiment analysis, binary cross-entropy loss for the paraphrase evaluation task, and mean squared error loss for the semantic similarity task.

Sentiment analysis, a classification task aiming to predict the sentiment (positive, negative, or neutral) of a given text, benefits from cross-entropy loss, particularly for multi-class classification, as it penalizes confidently incorrect predictions effectively. In paraphrase detection, a binary classification task, where sentences are labeled 0 for non-paraphrases and 1 for paraphrases, binary cross entropy is the preferred loss function, aligning well with binary classification objectives by measuring the disparity between predicted probabilities and actual binary labels. For the semantic similarity task, which entails predicting continuous values representing similarity between input entities like sentences or phrases, mean squared error (MSE) loss is a suitable choice, given its sensitivity to large errors and common utilization in tasks involving continuous variable outputs.

4.4 Gradient Surgery

Instead of combining the losses to pass a single value into the AdamW optimizer, we use each of the losses to implement gradient surgery. We achieve this using a PCGrad implementation (Tseng, 2020) based on Yu et al.'s "Gradient surgery for multi-task learning". PCGrad wraps the AdamW class and takes in a list of gradients to perform a step; it accesses the gradients within the Optimizer class to rectify conflicting gradients (Tseng, 2020).

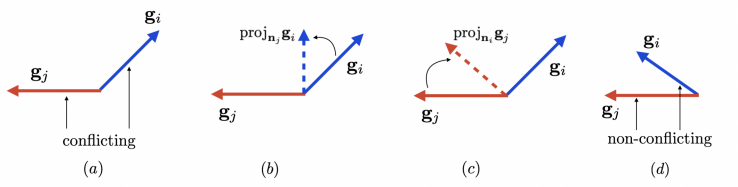

Figure 2: Yu et al. demonstrate how their method addresses conflicting gradients for example tasks i and j. As described previously, task i's gradient is projected onto the normal vector of j's gradient using the formula $\mathbf{g}_j = \mathbf{g}_i - \frac{\mathbf{g}_i \cdot \mathbf{g}_j}{\|\mathbf{g}_j\|^2}\mathbf{g}_j$

4.5 Cosine Similarity

Our third extension incorporates fine-tuning with cosine similarity for tasks related to paraphrase detection and understanding semantic similarity, as both require assessing how similar sentences are. Inspired by "Sentence-BERT: Sentence Embeddings using Siamese BERT Networks" (Reimers and Gurevych, 2019), we adopt cosine similarity as an integral part of our approach. We calculate the cosine similarity between the pooler outputs of sentence pairs and introduce this measurement just before the classification stage of our model. We employ different loss functions as a metric for optimization such as cross entropy and mean squared error. Cosine similarity is a means of measuring the similarity between sentence embeddings, and values range from -1 (exactly opposite) to 1 (exactly the same). BCE measures the performance of classification models by outputting a probability between 0 and 1. MSE measures the average squared difference between the estimated values and the actual value.

4.6 Combining Extensions - Ensembling

As a last extension, we explore the use of ensembling, a common machine learning technique that combines the predictions of multiple individual models into one final model to improve overall performance and robustness. By aggregating the outputs of diverse models, the ensemble can achieve better generalization and accuracy than any single model on its own. As our final model, we ensembled three other high-performing ensemble models using an averaging approach. We perform the prediction task on each of the three models to retrieve the logits of the prediction and then average the three sets of logits to get the final prediction of the ensemble model.

In the first layer of ensemble models (Figure 3), one of the models is an ensemble of three multitask classifiers trained separately on only one of the downstream tasks, similar to the baseline model. Because each of these models is designed to be specialized in one of the downstream tasks, the ensemble model directly uses the prediction of the appropriate model designed for the downstream tasks, rather than combining the predictions to make one final prediction. The other two ensemble models in the first layer use the aforementioned averaging technique to produce a final prediction.

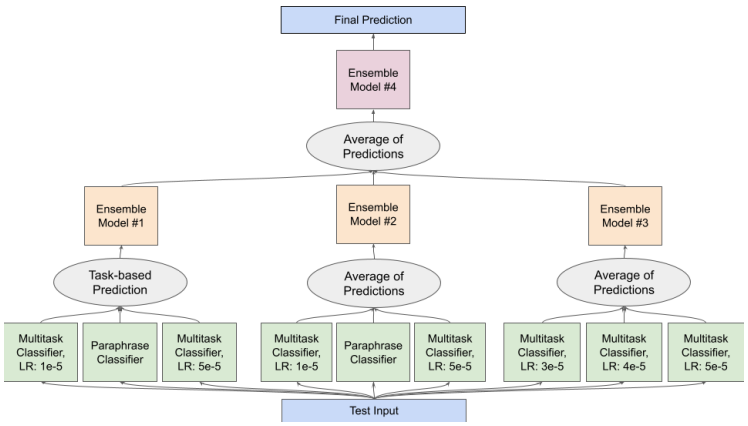

Figure 3: Our final model contains 3 model layers. The first layer comprises models we previously created combined into the second layer of ensemble models. The final layer is one ensemble model combining the layer 2 models by averaging the predictions of each.

5 Experiments

5.1 Data

We use four datasets for the three downstream tasks:

1. **Stanford Sentiment Treebank**, containing single sentences from movie reviews with categorical labels 0 (negative), 1 (somewhat negative), 2 (neutral), 3 (somewhat positive), and 4 (positive). (Socher et al., 2013)

2. **CFIMDB**, containing extremely positive or negative movie reviews. These have binary labels of "1" for positive, and "0" for negative.
3. **Quora**, containing question pairs with labels indicating whether one is the paraphrase of the other. These are binary labels of 0 being "no", and 1 being "yes" (Fernando and Stevenson, 2009).
4. **SemEval**, containing sentence pairs with labels indicating semantic similarity, from 0 (unrelated) to 5 (related) (Agirre et al., 2013).

Task	Dataset	# train	# dev	# test	labels
Sentiment Analysis	CFIMDB*	1701	245	488	0/1
Sentiment Analysis	Stanford Sentiment Treebank	8544	1101	2210	0-4
Paraphrase Detection	Quora	141506	20214	40431	1/0
Semantic Test Similarity	SemEval	6040	863	1725	0-5

Table 1: Summary of datasets *only used for baseline BERT model pretrained and fine-tuned using only SST task datasets

5.2 Evaluation method

We use accuracy for the sentiment analysis (SST) and paraphrase detection tasks, and Pearson correlation of the true similarity values against the predicted similarity values for the semantic test similarity (STS) task.

5.3 Experiments

Unless indicated, we fine-tune using a batch size of 8, learning rate of 1e-5, hidden dropout probability of 0.3, and train for 10 epochs. We use the AdamW with a weight decay of 0.0.

5.3.1 Results

SST Test Accuracy	Paraphrase Test Accuracy	STS Test Correlation	Overall Test Score
0.538	0.822	0.573	0.716

Table 2: Leaderboard test results from model Ensemble 7 (see Table 5)

Model type	Accuracies/Pearson Correlation			
	SST	Paraphrase	STS	Overall
Baseline: Sentiment Fine-tuned, Mulitask-Classifier	0.517	0.414	-0.098	0.461
Sentiment Pretrained, Mulitask-Classifier	0.391	0.609	-0.111	*
Sum Losses (3 Task Trained, Multitask-Classifier)	**0.518**	**0.757**	0.444	**0.666**
Gradient Surgery[1]	0.513	0.748	0.475	**0.666**
Cosine Similarity + MSE Loss	*	0.590	**0.559**	*
Cosine Similarity + CE loss	*	0.732	*	*

Table 3: Dev set accuracies for the SST and Paraphrase tasks, and Pearson correlation for the STS task, for each of our models. [1]batch size = 4 to address memory error *combination not suitable for this downstream task.

5.4 Summing the Losses

Our initial MTL approach of summing losses for all three downstream tasks significantly improves overall performance from 0.461 to 0.666. This improvement is reflective of the model optimizing for each of the tasks instead of just SST. SST performs better than we expected; even though additional tasks are being evaluated, accuracy is not sacrificed.

5.5 Gradient Surgery

Implementing gradient surgery results in a much longer runtime (6 hours) because it requires pairwise comparison of loss gradients at each timestep during training. Using individual losses allows for an improvement in STS. Despite this long training time, we still see the same overall accuracy as the previous approaches. This indicates that our model architecture itself is suboptimal, and we focus on hyperparameter fine-tuning in section 5.7. Even though this method improves STS performance to reach a Pearson correlation of 0.475, cosine similarity, expanded upon below, is a more computationally efficient method of improving performance in this task.

5.6 Cosine Similarity

We see that using cosine similarity optimized by cross entropy loss did 0.142 points in better in terms of accuracy in the paraphrase detection task. We also see that compared to to other methods besides ensembling, using cosine similarity optimized by MSE had the highest pearson correlation.

5.7 Learning Rate

Devlin et al. (2019) experiment with different learning rates in the original BERT paper, so we experiment with different learning rates on the sum of losses model.

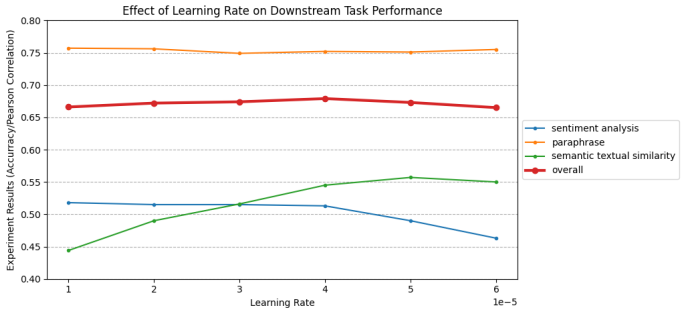

Figure 4: Performance of each of the downstream tasks as learning rate changes

STS surprisingly improves significantly as we increase the learning rate, reaching a maximum accuracy with a learning rate of 5e-5. This improvement could be a result of a higher learning rate preventing the model from getting stuck at local minima. Paraphrase accuracy remains unaffected by learning rate changes. This could be explained by many factors, including the data being relatively non-noisy or a flat loss plateau. SST performance worsens with an increase in learning rate, indicative of poor generalization or skipping over optima.

Learning rate	Accuracies/Pearson Correlation			
	SST	Paraphrase	STS	Overall
4e-5	0.513	0.752	0.545	**0.679**
5e-5	0.490	0.751	**0.557**	0.673

Table 4: Significant learning rate results: our best overall performance was achieved with a learning rate 4e-5, and our highest STS accuracy was achieved with a learning rate 5e-5. SST and paraphrase did not outperform the 1e-5 model.

These findings prompted us to use ensembling to examine the efficacy of using different hyperparameters for each of the tasks, even if the models are shared.

5.8 Ensembling

We see that ensembling significantly improves the overall prediction accuracy. The worst performing average-based ensemble model, Ensemble Model #4 overall still performs better than any of the non-ensemble models that we experiment with. While the best overall performing ensemble model

		Models ensembled			Dev Accuracies/Correlation			
	Ensemble Method	Model #1	Model #2	Model #3	SST	Para	STS	Overall
Ensemble 1	Task-Based	Sentiment Classifier	Paraphrase Classifier	Similarity Classifier	0.517	0.813	0.400	0.676
Ensemble 2	Task-Based	Multitask Classifier, LR: 1e-5	Paraphrase Classifier	Multitask Classifier, LR: 5e-5	0.518	0.813	0.557	0.703
Ensemble 3	Average	Multitask Classifier, LR: 1e-5	Paraphrase Classifier	Multitask Classifier, LR: 5e-5	0.517	**0.821**	0.55	0.704
Ensemble 4	Average	Cosine Similarity + MSE	Multitask Classifier, LR: 3e-5	Average-Based Loss	0.523	0.759	0.521	0.681
Ensemble 5	Average	Multitask Classifier, LR: 3e-5	Multitask Classifier, LR: 4e-5	Multitask Classifier, LR: 5e-5	0.520	0.762	**0.598**	0.693
Ensemble 6	Average	Ensemble 2	Ensemble 3	Ensemble 4	0.521	0.819	0.575	0.709
Ensemble 7*	Average	Ensemble 2	Ensemble 3	Ensemble 5	**0.528**	0.820	0.587	**0.714**

Table 5: Dev set accuracies on each of the downstream tasks for the experimented ensemble models. *Our final model reported in Section 5.3.1 and our final dev leaderboard accuracies

does not have the best accuracy for each downstream task, this is a tradeoff to improve the overall accuracy. For example, Ensemble Model #3, a combination of the highest performing models for each downstream task, ensembled by averaging the predictions, has a minimally higher accuracy for the paraphrase task than Ensemble Model 6. Even for the STS task where we see that Ensemble Model #5, a combination of the overall best-performing models that we experiment with, has a better STS score than Ensemble Model #7, we see that the difference is fairly minimal (0.009 difference in accuracy), yet the overall accuracy is significantly higher.

6 Analysis

6.1 Impact of Cosine Similarity and Loss Functions

In this section, we analyze the impact of cosine similarity and a binary cross entropy (BCE) loss function compared to mean square error (MSE) loss function for the paraphrase task. We also reason why cosine similarity for the semantic similarity task performs well, equally comparable to our ensemble of ensembles technique.

6.1.1 Cosine Similarity with BCE vs. MSE on Paraphrase

BCE is inherently suited for binary classification tasks (like paraphrase detection), as it models the probability that a given input belongs to a particular class. Paraphrase detection is a binary classification problem, where the two classes are "is a paraphrase" and "is not a paraphrase." As a result, BCE directly aligns with the nature of the task.

Furthermore, when combining cosine similarity with BCE for tasks like paraphrase detection, the gradient behavior is particularly advantageous. BCE provides gradients that are more meaningful and discriminative for binary classification, as it penalizes incorrect classifications proportionally to the distance from the correct classification boundary. This results in gradients that effectively guide the model towards better separation of paraphrases from non-paraphrases. In contrast, MSE focuses on minimizing the variance from the actual value rather than optimizing for the classification boundary, which can result in slower convergence and less optimal solutions for binary classification problems.

The gradient behavior of BCE with cosine similarity provides a more direct and effective pathway for models to learn the nuances of paraphrase detection, contributing to better overall performance.

We must note the importance of incorporating cosine similarity into BCE compared to just applying BCE directly. Since cosine similarity emphasizes the geometrical closeness of embedding vectors, this ensures that the learning process is sensitive to the subtleties of semantic meaning. Therefore, when used with BCE, this approach not only determines whether two sentences are paraphrases but does so by assessing how semantically close or distant the sentences are.

6.1.2 Cosine Similarity on STS

BERT generates embeddings that are contextually informed; the same word can have different embeddings due to how it is being used within sentences. This is highly advantageous for the sentiment similarity task as it can capture subtle context-dependent differences in sentiment. Furthermore, cosine similarity is highly aligned with the nature of the STS task. Since cosine similarity focuses on the orientation of the vectors in the embedding space, rather than their magnitude, it can effectively measure how sentences are related in terms of directionality in the embedding space, which correlates well with having similar sentiments, regardless of other linguistic features.

6.2 Ensembling

Ensembling offers a robust approach to harnessing the collective strengths of multiple models, thereby enhancing overall predictive performance. Our experimentation reveals promising results, indicating that ensembling consistently outperforms individual models across all evaluated metrics.

In our ensemble modeling approach, we craft each ensemble model with distinct rationales tailored to optimize predictive accuracy for specific tasks. Ensemble #1 employs a task-based technique, leveraging fine-tuned multitask classifiers uniquely trained for each task. This method ensures that predictions are directly derived from models attuned to each individual task. Similarly, Ensemble #2 adopts a task-based strategy, selecting the most accurate model for each task from our previously experimented models. Ensemble #3 experiments with an averaging approach, using the same models from Ensemble #2 to explore potential accuracy improvements. While it slightly enhances the accuracy of the paraphrase task, there is a marginal decrease in sentiment classification and semantic similarity accuracies compared to individual task-based ensembles. Ensemble #4 utilizes an averaging strategy to leverage the diverse strengths of different loss functions. Meanwhile, Ensemble #5 combines our overall top-performing individual models, capitalizing on their collective predictive power. Ensemble #6 and Ensemble #7, extend our ensembling approach by creating ensemble models from already ensembled models. Ensemble #7 exhibits the highest overall accuracy among the ensemble models tested.

While ensembling incurs additional training overhead due to the necessity of training separate models for each downstream task, the process of combining the predictions of each of the models during testing is relatively inexpensive since the combined ensemble model itself is not trained. This cost-effectiveness, coupled with the observed improvements in predictive accuracy, highlights ensembling as a valuable technique for enhancing multitask model performance.

7 Conclusion

Our study highlights the efficacy of leveraging BERT for specialized tasks through fine-tuning and advanced techniques. We observe substantial improvements in performance across sentiment analysis, paraphrase detection, and semantic textual similarity tasks. Ensembling emerges as a key strategy for enhancing model accuracy, with our iterative ensembling approach showcasing the technique's potential and significant improvements in overall and task-individual accuracy. Our findings contribute to the ongoing exploration of ensembling techniques in NLP and underscore the importance of leveraging BERT's contextual understanding for specialized NLP tasks.

Our next steps would be to explore data preprocessing methods to improve SST accuracy, which remained relatively unchanged across experiments. We would also conduct further pre-training or use a more optimized pre-trained model like RoBERTa.

References

Eneko Agirre, Daniel Cer, Mona Diab, Aitor Gonzalez-Agirre, and Weiwei Guo. 2013. *SEM 2013 shared task: Semantic textual similarity. In *Second Joint Conference on Lexical and Computational Semantics (*SEM), Volume 1: Proceedings of the Main Conference and the Shared Task: Semantic Textual Similarity*, pages 32–43, Atlanta, Georgia, USA. Association for Computational Linguistics.

Qiwei Bi, Jian Li, Lifeng Shang, Xin Jiang, Qun Liu, and Hanfang Yang. 2022. Mtrec: Multi-task learning over bert for news recommendation. In *Findings*.

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

Samuel Fernando and Mark Stevenson. 2009. A semantic similarity approach to paraphrase detection. *Proceedings of the 11th Annual Research Colloquium of the UK Special Interest Group for Computational Linguistics*.

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

Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Conference on Empirical Methods in Natural Language Processing*.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, A. Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Conference on Empirical Methods in Natural Language Processing*.

Chi Sun, Xipeng Qiu, Yige Xu, and Xuanjing Huang. 2020. How to fine-tune bert for text classification?

Wei-Cheng Tseng. 2020. Weichengtseng/pytorch-pcgrad.

Tianhe Yu, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. 2020. Gradient surgery for multi-task learning.

A Appendix

A.1 Summing vs averaging losses

Part of our initial exploration involved looking into how to combine the losses for the three downstream tasks. One comparison we carried out was summing the losses versus averaging the losses

Learning rate	Accruacies/Pearson Correlation			
	SST	Paraphrase	STS	Overall
sum losses	0.518	0.757	0..444	0.666
average losses	0.514	0.753	0.444	0.664
sum losses, 0.4 dropout	0.505	0.752	0.456	0.665
average losses, 0.4 dropout	0.509	0.753	0.473	0.666

Table 6: Learning rate parameter tuning for model summing all losses

Our two methods of combining losses achieve similar overall performances. For both, we see that the model was overfitting for the SST and STS tasks, with much higher training accuracies/correlations than dev set accuracies/correlations.

Adjusting the dropout helps reconcile the difference slightly, but the average losses with 0.4 dropout model still sees a final train correlation of 0.914 for STS compared to a dev set correlation of 0.473.

We choose to use sum over average for two reasons. Firstly, our research shows that summing the losses was the most common way of combining losses for MTL tasks. Secondly, the dev set accuracies also jump around across epochs for averaging, suggesting the model was not as stable. Thus, we use the summing method to carry out hyperparameter tuning and prepare the ensemble model.

A.2 Hyperparameter tuning

These are the numerical values for the graph shown in section 5.7.

Learning rate	Accuracies/Pearson Correlation			
	SST	Paraphrase	STS	Overall
2e-5	0.515	0.756	0.490	0.672
3e-5	0.515	0.749	0.516	0.674
4e-5	0.513	0.752	0.545	0.679
5e-5	0.490	0.751	0.557	0.673
6e-5	0.463	0.755	0.550	0.665

Table 7: Learning rate parameter tuning for model summing all losses

A.3 Task-dependent trained multitask classifiers

In terms of fine-tuning individual classifiers on the three downstream tasks, we see that consistently, the accuracy of the downstream task improves significantly compared to baseline, since the individual classifiers are trained solely on the dataset related to the downstream task. When combined into the ensemble model, we see that the accuracy for each task is the same as the individual classifier's accuracy for that task. Since the classifier is directly using the prediction from the individual fine-tuned classifier, it makes sense that the accuracy is the same as the individual classifier's accuracy for each task.

Model	Accuracies/Pearson Correlation		
	SST	Paraphrase	STS
Sentiment Pretrain	0.391	0.609	-0.011
Sentiment Finetune	**0.517**	0.144	-0.098
Paraphrase Pretrain	0.144	0.657	-0.011
Paraphrase Finetune	0.224	**0.813**	-0.056
Semantic Similarity Pretrain	0.144	0.609	0.267
Semantic Similarity Finetune	0.120	0.593	**0.400**