Fine-tuning Multi-Task Learning in BERT Model

Stanford CS224N Default Project

Cole Crichton
Department of Computer Science
Stanford University
colehc@stanford.edu

Emily Guo
Department of Computer Science
Stanford University
emilyguo@stanford.edu

Abstract

Ever since the introduction of BERT and its state-of-art task performances in 2018, advances in natural language processing (NLP) have proliferated. Multi-task learning, one such advance, has become its own research field, given its potential to facilitate information-sharing across related tasks, which also reduces the total parameters needed. Excited by this field's work to make powerful models more accessible, we researched, implemented, and evaluated various proposed approaches to multi-task learning. In this paper, we present a multi-task BERT model fine-tuned and evaluated on three related tasks. We find that the model's multi-task capabilities significantly improve if the model is fine-tuned with multi-task learning. We also experiment with cosine similarity and gradient surgery as performance-improving extensions. Ultimately, we hope to offer fresh interpretations and analyses of each technique's effects on the model.

1 Introduction

The release of BERT in 2018 was pivotal for the natural language processing (NLP) field, its bidirectional strategy for learning word representations unlocking models with a deeper understanding of human languages [1]. Models powered by BERT rose in popularity, given their ability to perform well on a wide range of tasks, from classifying sentence sentiment to predicting a person's profession based on seemingly unrelated descriptions. However, what became the new benchmark for impressive was the training and usage of BERT-powered models to learn language embeddings general enough to perform well on related but *different* tasks. Within the field of multi-task learning, there are twin research goals of, first, developing ever-better models trained using ever-compact data to perform even better over several downstream tasks; and second, improving interpretability of these models. While multi-task learning arose from the realization that models can share information and learn from *each other*'s insights, how exactly this works demands deeper research – multi-task models will increasingly serve as the foundation of decision-making technologies, and it is critical for humans to be able to diagnose a model's reasoning for its outputs.

Our research focused on improving the default vanilla BERT model, so that our final model consistently performs well in three tasks: sentiment analysis, paraphrase detection, and sentence similarity. We accomplished this primarily through multitask fine-tuning: inspired by a combination of several papers' techniques, we aimed to not only replicate these techniques' improved performance effects, but also to explore which individual techniques might pair well with others.

2 Related Work

Our paper's research is built on the foundation of the work behind BERT and several techniques to enhance BERT models. We present an overview of this work here.

BERT [1] is a transformer-based model that established new state-of-the-art performance standards for NLP tasks like sentiment analysis, question answering, and sentence-pair regression. The

researchers behind BERT achieved such significant advancements by leveraging bidirectional word representations, which helps the model learn better contextual word embeddings.

Multi-task learning became a popular technique for tuning BERT models, since it not only facilitates information-sharing across related tasks, but in doing so can reduce the parameters and resources needed. Research in this field is dedicated to ensuring multi-task learning is balanced across tasks; the technique risks the model over-learning from a single large task [2], or ineffective learning due to clashing task gradients [3]. Bi et. al [4] propose a novel combination of multi-task learning and gradient surgery to address these issues. By optimizing the loss functions for *each* task simultaneously, they derive a final loss function. To address potential gradient clashing, they implement gradient surgery, which led to observable model improvements.

Diving in the multi-task learning approach, we looked for papers that might inform our approach to each sub-task, since some required sentence encoding models, while others required sentence pair interaction models. One paper that influenced our work was [5], which demonstrated several useful approaches, such as using cosine similarity distance for sentence pair tasks, and concatenating sentence pair representations to enhance inferences.

3 Approach

Our model extends the default vanilla BERT model to perform well across three downstream tasks: sentiment analysis, paraphrase detection, and sentence similarity. In this section, we detail our primary approach, and provide a closer look into the experimental techniques we investigated for the purposes of improving model performance.

3.1 The Model

Like the default BERT, our model first processes sentences into tokens – here, we prepend a [CLS] token to each input sentence representation, which BERT uses in its hidden state to represent the overall sentence embedding. Our pooling strategy was thus to use the output of the CLS-token, since we did not notice significant improvements from computing the mean of *all* output vectors. Pooling ensures our model outputs fixed-size sentence embedding. An embedding layer is then applied to concatenate the token embeddings with position embeddings, the resulting embeddings used as input later in the model.

Within each of our model's 12 transformer layers, we apply multi-head attention to jointly learn information across different positions of different word embedding subspaces. We compute a scaled dot-product attention as described in [6] as:

$$Attention(Q, K, V) = Softmax(\frac{QK^T}{\sqrt{d_k}})V$$

, where Q, K, V are the query, key, and value vectors, and d_k is the dimension of Q and K. The scaling by $\frac{1}{\sqrt{d_k}}$ reduces variability for gradient calculations, stabilizing learning. For multi-head attention, each "head" $(head_1, ...head_h)$ focuses on projections of Q, K, V across d_k, d_k, and d_v dimensions, respectively. Each of these outputted attention values are then concatenated and projected as follows:

$$MultiHead(Q, K, V) = Concat(head_1, ..., head_h)W^O,$$
$$\text{where} head_1 = Attention(QW_i^Q, KW_i^K, VW_i^V).$$

Our model uses 12 attention heads, improving its ability to attend to not just the current layer of the encoder, but also to the encoder's previous layer. Each encoder layer also contains a feed-forward network with two linear transformations and a GELU activation function sandwiched in between:

$$FFN(x) = \max(0, xW_1 + b_1)W_2 + b_2.$$

Our BERT model then applies a dropout layer where $p_{drop} = 0.1$ to the output, which is next passed through layers that reinforce residual connections and normalize the output. We use the [CLS] token embedding outputted by the last transformer layer to represent the overall sentence embedding.

3.2 Multi-Task Learning

We chose to fine-tune our model via multi-task learning, hoping that the sub-models trained on separate tasks might supplement each others' learning and perhaps generate new insights across related tasks. To do this, we developed task-specific encoders and loss functions.

3.2.1 Sentiment Classification Task

We encode each sentence with our BERT model, apply a dropout layer, and adopting the embedding of the [CLS] token as the sentence representation. Passed through a final linear layer, the output is squeezed from a hidden number of features into just 5: one for each possible sentiment classification.

We use Cross Entropy Loss to evaluate our classification predictions, which is calculated as:

$$\mathcal{L}_{sent} = l(x, y) = \{l_1, ..., l_N\}^T, l_n = (-w_{y_n} \log \frac{\exp(x_{n,y_n}}{\sum_{c=1}^{C} \exp(x_{n,c}} y_{n,c}),$$

where x is the input, y is the target label, w is the weight, C is the number of classes, and N spans minibatch dimensions $d_1, ..., d_k$ for the K-dimensional case. The loss is summed and averaged for each batch.

3.2.2 Paraphrase Detection Task

Given a sentence pair, we encode each sentence separately using the BERT model. Working with each encodings' [CLS] tokens as our sentence representations, we apply a dropout layer to prevent over-fitting, then concatenate the two representations into a single tensor. We then apply a linear layer to transform the concatenated tensor into a single logit, which is meant to represent our prediction. To evaluate this binary regression task, we pass the predicted logit through a sigmoid layer, then assess our loss using Binary Cross Entropy against the target label, as follows:

$$\mathcal{L}_{para} = l(x, y) = \{l_1, ..., l_N\}^T, l_n = -w_n[y_n \log \sigma(x_n) + (1 - y_n) \log(1 - \sigma(x_n))].$$

This loss is also averaged over the batch size.

3.3 Semantic Similarity Task

Given a sentence pair, we use the BERT model to encode each sentence separately, working with the [CLS] token as our sentence representations. We apply a linear layer on the embeddings without transforming the dimensions, to promote new learning. We then compute cosine similarity for the two representations, the output of which is scaled by a factor of 5 to simulate similarity scores in the SST dataset, which range from 0 to 5.

To evaluate this multi-class regression task, we calculate the Mean Squared Error (MSE) Loss between our predicted scores and the target scores, as follows:

$$\mathcal{L}_{sim} = l(x, y) = \{l_1, ..., l_N\}^T, l_n = (x_n - y_n)^2,$$

which we average over the batch size.

3.4 Parallel Approach

To ensure that each task's learnings remained fresh in the model's memory across epochs, we leverage a parallel approach to multi-task learning. Within a single loop, our model is passed three batches – one from each task's dataset –, on which it trains sequentially. After each task's batch, the model backpropagates the loss, updating the task-specific weights according to its learnings. Only after the model has trained on the three tasks' batches does the model optimize the loss functions simultaneously; the final loss function thus becomes:

$$\mathcal{L}_{total} = \mathcal{L}_{sent} + \mathcal{L}_{para} + \mathcal{L}_{sim}$$

Yet, this introduces potential conflicts between task-specific gradients – in the next section, we discuss this issue in detail as well as our solution approach.

3.5 Optimizer

To optimize our model's stochastic objective functions, we implement the Adam Optimizer [7]. By using *adaptive* learning rates, updated based on a rolling average of gradient magnitudes, our gradient descent avoids getting "stuck" in local minima, and more efficiently arrives at global extrema. This effort is also aided by Adam's utilization of a rolling average of the gradients, which accelerates the arrival to extrema. Advantages specific to our tasks include efficiency with handling large amounts of data, an ability to interpret sparse gradients, and a computationally low need for memory. The Adam optimizer, however, becomes less clean with multi-task learning: multi-task gradient risks being dominated by one task gradient, which comes at cost of degrading the performance of other task [3]. This problem, which is inherent to multi-task learning but otherwise vastly improved our model's performance, motivates our implementation of gradient surgery.

3.6 Gradient Surgery Extension

As explained above, multi-task learning and optimization is a challenge that has plagued the field of deep learning. Maintaining an ideal balance between task goals is difficult, especially when a task's own optimal objectives are difficult to discern. Multi-task learning introduces a new complexity, since it is possible for gradients of separate tasks to point in directions that oppose each-other; the best learning direction in these potentially conflicted cases is not immediately clear. In an effort to rectify this, we implement gradient surgery. In the case that we have conflicting gradients, defined when we have $\cos\theta < 0$ for angle θ between gradients a and b, we use projections to help remove this conflict. By projecting each task's gradient onto the normal plane of another task gradient, we end up with a pair of gradients that have a cosine similarity that is greater than one. Pairs of gradients are chosen at random until we have evaluated all pairs for negative similarity. This results in less destructive interference, and allows us to continue stepping more efficiently. The gradient surgery method we implemented is called PCGrad (aptly named after the process "projecting conflicting gradients"), which wraps around our existing Adam optimizer in an effort to improve our model [8].

3.7 Cosine Extensions

Although literature has shown extensively that cosine similarity aligns well with sentence pair tasks, we also explored leveraging CosineEmbeddingLoss for our sentence pair tasks. Rather than using cosine similarity to compute a single representation of the sentence pair (which would then be passed to a loss function), we experimented with passing two sentence representations into the CosineEmbeddingLoss function, which strictly requires a target label of -1 or 1, proxying "not similar" or "similar" representations respectively. To make the STS dataset compatible with this loss function, we re-scored all sentence pairs initially scored 0, 1, or 2 as -1; and all pairs initially scored 3, 4, or 5 as 1. However, the loss function didn't seem to be compatible with this approach; during evaluation, our model failed at the STS task (the only task we implemented CosineEmbeddingLoss for), hovering between -0.009 and -0.005 Pearson correlation values the entire training time. A Pearson correlation value of 1 suggests a total positive linear relationship between our predictions and the STS scores; -1 suggests a total negative linear relationship, while 0 indicates *no* correlation. Observing that our model's performance is not at all correlated with the STS scores, we hypothesize that we might improve performance if only using sentence pairs ranked as a strong 0 or 5. However, this would have significantly reduced our dataset size, so we re-implemented cosine similarity.

Cosine similarity is useful for fine-tuning the Semantic Similarity Task, calculated for input a and output b as $\frac{a \cdot b}{||a||\ ||b||}$. We use the cosine similarity in our multi-task classifier forward step in lieu of relying on a linear layer for our output, like we do for Paraphrase and Sentiment. By embedding our input and outputs and computing their cosine similarity, we can determine how similar our a and b are. Our model ranks similarity discretely from 0 to 5, so we multiply this cosine value by 5 at the end of our forward step in order to change our range from [0,1] to [0,5].

4 Experiments

4.1 Data

We trained and evaluated our model on three datasets corresponding to our three specific tasks [9].

To inform sentiment analysis, we use the Stanford Sentiment Treebank (SST) dataset, developed for sentiment classification tasks. The SST was created from 11,855 single sentences from movie reviews, which were then parsed into 215,154 unique phrases, each annotated by 3 human judges. The SST has five possible classifications: negative, somewhat negative, neutral, somewhat positive, or positive. We used BERT embeddings to predict the sentiment classifications of input sentences.

The paraphrase detection task asks our model to determine whether particular sentence pairs have the same semantic meaning, demanding a model with a deeper "understanding" of semantics. We use the Quora Dataset for training and evaluation, given that it consists of 400,000 question pairs labeled by the website Quora to indicate whether specific phrases are paraphrases of each other. We ask our BERT model to predict whether or not input sentence pairs are paraphrases of each other.

To further develop our model's fine-grain understanding of semantic meaning, we assess our model with the semantic textual similarity task, in which the model is asked to score sentence pairs semantic similarity along a *scale*, rather than a binary like in the paraphrase detection task. To do this, we use the SemEval STS Benchmark Dataset, which consists of 8,628 sentence pairs of varying similarity. The sentence pairs are scored on a scale from 0 to 5, ranging from "not at all related" to "equivalent meaning", respectively.

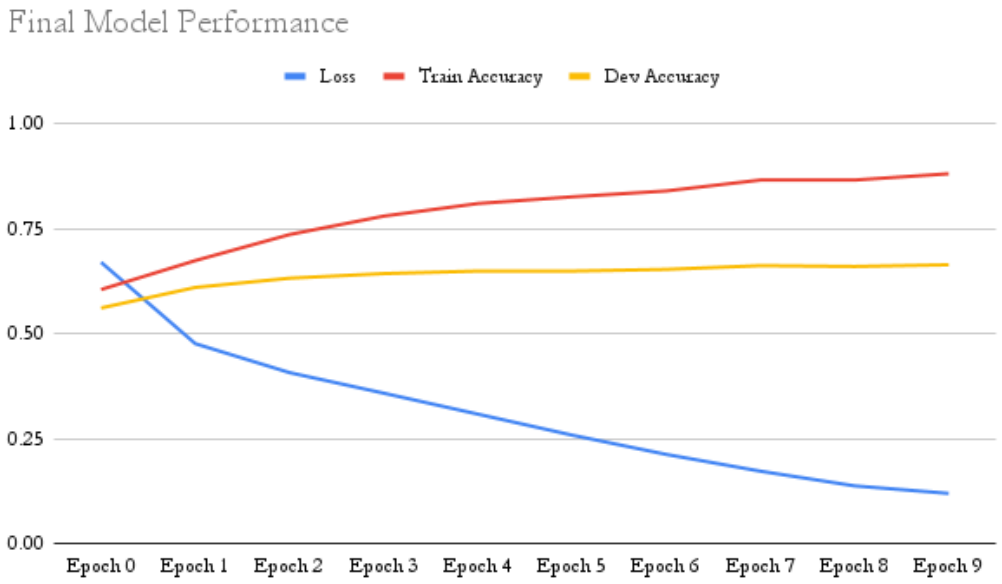
4.2 Evaluation method

To evaluate our multi-task model's performance, we tested our model using both pre-trained and fine-tuned embeddings on sentiment classification (accuracy on SST dev dataset), paraphrase detection (accuracy on the Quora dev dataset), and semantic textual similarity (calculating the Pearson correlation of labeled similarity values, against predicted similarity values across SemEval STS dev dataset) [9].

4.3 Experimental details

We ran our model on Google Colab using the multitask classifier. We used the available GPU, and trained with the fine-tune flag enabled for 10 epochs. The default and used learning rate was 10^{-5}, and our training took approximately two hours. The dev accuracies and correlation were stable (changing by less than one percent per epoch) after the fifth epoch, so we are confident that our experiment has a satisfactory number of epochs.

4.4 Results



Our final performance numbers were as follows:

	Paraphrase Acc	SST Acc	STS Corr	Average Score
Train	0.728	0.965	0.965	0.886
Dev	0.719	0.505	0.786	0.670
Test	0.723	0.519	0.766	0.670

The data is satisfactory. We are much improved from where our model began (performance averages around 0.2) and we have learned the performance impacts of choices multi-task classification. Modern BERT models have accuracies north of 90%, so we will note that our final model is not groundbreaking.

We remark that 1.0 accuracy and 1.0 correlation is not necessarily a goal we are aiming for. The datasets we are training on are labeled by humans; an STS score of 4 versus 5 is a qualitative choice that may not be universally agreed upon. Our goal is to detect sentiment, identify paraphrases, and grade similarity, not to match exactly how a sample of human labelers accomplished these subjective tasks. High accuracy/correlation scores are crucial training objectives, and improving scores is similarly helpful, but we are also interested in understanding intuitively which factors help us arrive at higher scores. Rather than aiming at one or two specific performance metrics, in the future we would diversify our metric objectives that better illustrate how an assortment of model design choices affect our measured metrics. For example, since we only use one metric per task, even when we improve the accuracy or correlation results, we are unsure of how exactly our extensions *led* to those improvements. The output of our research is a knowledge of how we improved our metrics, and to what degree. We are satisfied with our results, but more satisfied with context of how we got these. The findings from this will help analyze and improve future models.

Our final dev accuracy was 0.67, but our train accuracy was 0.886. We thus hypothesize that our model overfits to our train data. To investigate this further, we can examine the difference between our train and dev values for the paraphrase detection task and sentence similarity task: 0.728 vs 0.719, and 0.965 vs 0.786, respectively. For the paraphrase detection task, we see a trivial decrease in accuracy, while our sentence similarity task's Pearson correlation has a significant decrease. We used a dropout layer for the paraphrase detection task, but left it out for the sentence similarity task because that would ruin the similarity calculation for cosine similarity. The effect of a dropout layer is a prevention of overfitting, so we believe that we are seeing the consequences of removing this layer when encoding sentences from the STS dataset. However, we chose to accept possible overfitting, preferring to implement cosine similarity given its intuitive compatability for the task as well as the observed improvement in our sentence similarity task.

5 Analysis

Initially, our multitask model had the following dev performance: SST Accuracy: 0.317, Paraphrase Accuracy: 0.376, STS Correlation: 0.019. Our model was not doing well, and outright failing at the sentence similarity task. Since our accuracies were worse than guessing and our Pearson correlation for the STS dataset was close to 0, we first began by re-designing the encoding and evaluation functions for each of the three tasks. We implemented linear layers in the encoding functions, improving the inputted embeddings' learnable features; our previous approach of using the static Bert models meant that our baseline model was not learning anything.

In our first experimentation with multi-task learning, we used a round-robin approach but suffered poor results with the first task in the round-robin loop. Since the model had to be trained *sequentially* on all the data for all three tasks, the model seemed to increasingly forget its learnings from the first task with each epoch. To address this rapidly disintegrating memory, we instead leveraged a parallel multi-task learning approach.

So that each task's learnings remained fresh in the model's memory across epochs, our model zips up the three separate dataloaders, and in a single loop, samples a batch from each dataset to train on three tasks in parallel. This improved our performance for sentiment analysis and paraphrase detection, but our performance for sentence similarity remained low. In order to fix this, we applied cosine similarity to the sentence encodings, and transformed the logits into outputs corresponding to the dataset's label range of [0,5], and saw a huge increase in our STS dataset's Pearson correlation to 0.795. However, it should be noted that the Pearson correlation metric is not optimal for assessing performance on STS tasks; it is sensitive to outliers, non-linear relations, and non-normally distributed

data. Further, [10] suggests that metrics should be selected on a task-by-task basis, in light of different task requirements. While we chose to use this metric for consistency, we wonder if evaluating against literature-suggested metrics might better-align our training and task objectives.

Cosine similarity is extremely useful because the value outputted is in a range; values close to 0 tell us that two sentences are not similar, and values closer to 1 tell us that our sentences are extremely similar. We have a classification task for the STS dataset that requires a sliding scale of similarity–this matches very well with the features of cosine similarity. Sentiment analysis would not be aided by this similarity analysis. The Paraphrase task is a binary task that does not depend on direct similarity, so cosine similarity would not help us with this task more than training linear layers would.

Gradient surgery provided a trivial performance increase. This could be because our three tasks are not too different; the paraphrase detection task and STS require a model that can determine some metric of similarity, and sentiment analysis is separate but not opposite to these two tasks. Therefore, it would not be likely that we would have opposite gradients, which gradient surgery would help address. The efficiency increase from gradient surgery (helping us arrive at our extrema more efficiently) is not a relevant goal of our efforts, so we are purely evaluating the usefulness based upon changes in accuracy.

6 Conclusion

We present a model that performs well at sentiment analysis, paraphrase detection, and sentence similarity tasks. We first learned the value of feed-forward networks with layers specifically designed with a task in mind, which enabled our model to learn through fine-tuning on each task. In later extensions, we identified tasks that were aided by evaluating cosine similarity, and we evaluated the value of gradient surgery. Most significantly, we realized the merits of multi-task parallel training as opposed to round-robin training, learning about each strategy's strengths and weaknesses through analyzing both experimental implementations. A final learning underscoring our work is that, in terms of machine learning techniques, one size does not fit all: deep learning models, however powerful, still benefit from a human eye, one that understands task-specific constraints, and can translate those constraints into compatible techniques.

We remark that our tasks were trained and tested on specific datasets. Our sentiment analyses were for movie reviews; our data is relevant for our specific data, but it is unclear whether our model would accurately detect sentiment in non-movie-review sentences. With further testing on further datasets, we could identify the usefulness of our model on multiple datasets in addition to our current analysis on multiple tasks.

In future work, we hope to analyze our extensions and model changes on their impacts on the model's bias. We are curious to explore how efforts to improve traditional senses of performance (accuracy of predictions, etc.) may trade-off with efforts to mitigate bias in models. As NLP models increasingly interact with everyday humans – making decisions, answering questions, offering advice – their scope for potential harm increases as well. Thus, it is critical for researchers to better understand the trade-off between optimizing model success metrics and mitigating bias. Particularly, we would hope to fine-tune and evaluate our model on a fourth task designed specifically to identify gender biases; for each technique implemented in our model to improve performance, we would also asses how such extensions affected various bias metrics, posing the question: If our extensions increase our bias, are there still merits to using such extensions?

References

[1] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2018.

[2] Asa Cooper Stickland and Iain Murray. BERT and PALs: Projected attention layers for efficient adaptation in multi-task learning. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 5986–5995. PMLR, 09–15 Jun 2019.

[3] Tianhe Yu, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. Gradient surgery for multi-task learning. *arXiv preprint arXiv:2001.06782*, 2020.

[4] Qiwei Bi, Jian Li, Lifeng Shang, Xin Jiang, Qun Liu, and Hanfang Yang. MTRec: Multi-task learning over BERT for news recommendation. In *Findings of the Association for Computational Linguistics: ACL 2022*, pages 2663–2669, Dublin, Ireland, May 2022. Association for Computational Linguistics.

[5] Matthew Henderson, Rami Al-Rfou, Brian Strope, Yun-hsuan Sung, Laszlo Lukacs, Ruiqi Guo, Sanjiv Kumar, Balint Miklos, and Ray Kurzweil. Efficient natural language response suggestion for smart reply, 2017.

[6] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

[7] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[8] Wei-Cheng Tseng. Weichengtseng/pytorch-pcgrad, 2020.

[9] Cs 224n: Default final project: minbert and downstream tasks, 2023.

[10] Nils Reimers, Philip Beyer, and Iryna Gurevych. Task-oriented intrinsic evaluation of semantic textual similarity, 2016.