minBERT Multi Tasks

Stanford CS224N Default Project

Maxime Pedron & Augustin Boissier
Institute of Computational and Mathematical Engineering
mpedron@stanford.edu, aboissie@stanford.edu
Mentor: **Heidi Zhang**

Abstract

In this work, we explore the usage of dense layers post-BERT for multitask learning. Instead of convolutional and recurrent models, we experimented with faster, lighter and linear networks, to achieve near-perfect results. The model fine-tunes BERT for three tasks, using CLS token embeddings that are fed into task-specific networks. The model achieves an overall score of 0.76.

1 Personal Contribution

Maxime dedicated his time to refining the algorithm's structure and optimizing it's performance, while Augustin focused on finding alternative data sources, means of computing, and explored ways to incorporate new features into the project.

2 Introduction

Natural Language Processing tasks like sentiment analysis, paraphrase detection, and similarity analysis present challenges due to the complexity of human language. These tasks are crucial for understanding text data across various domains, from social media sentiment analysis to information retrieval.

Current methods often rely on task-specific models, leading to complexities in architecture and limited generalization. In this work, we explore an approach based on multi-task learning, that leverages post-BERT dense layers for multiple NLP tasks. We focused on feed-forward linear networks, fine-tuning BERT for each task, and integrating task-specific features and information through additional layers.

We address the challenge of imbalanced datasets, particularly in paraphrase detection, by carefully limiting dataset sizes during training. Our key approach involves feeding CLS token embeddings from BERT into task-specific networks, leveraging BERT's contextual understanding while tailoring subsequent layers to each task. Through experiments, we compare sequential training of task-specific networks with parallelization for optimizing gradients across all tasks simultaneously. Method 2 achieves an overall score of 0.75 compared to Method 1's 0.718, showcasing the effectiveness of our approach.

In the following sections, we delve into related work, detail our approach, experiments, results, and analysis. This work aims to provide insights into a promising avenue for multi-task NLP models, enhancing performance and generalization across diverse textual datasets.

3 Related Work

Our work builds upon recent advancements in pre-trained language models (PLMs) like BERT Devlin et al. (2019) and dense layers for downstream tasks. He and Zhang (2019) explore similar post-BERT dense layers for text classification, achieving promising results. Yu et al. yu2019bert utilize convolutional layers after BERT for question answering tasks. While recurrent models have been successful for various NLP tasks (e.g., Hochreiter and Schmidhuber (1997)), we focus on feed-forward networks for efficiency.

4 Approach

There are three steps to our approach. Firstly, we develop a minBERT implementation, starting with the skeleton code that provide the embedding of the initial token in our input. Secondly, we construct

a framework to tackle the three NLP tasks: sentiment analysis, paraphrase detection, and similarity analysis. Lastly, we refine our framework, optimizing all hyperparameters to enhance accuracy on the dev set.

Due to Colab's standby after 1-2 hours, we were unable to train our algorithm on it, so instead relied on our local machines. We invested a significant amount of time and effort working on our Mac to devise a method for parallelizing our gradient descent and forward loop operations. To achieve this, we use **Metal Performance Shaders (MPS)**, a GPU-facilitaded acceleration backend compatible with our chips. This approach resulted in a threefold increase in performance.

5 Experiments

5.1 Data

5.1.1 Original Dataset

In the study, three specialized datasets were used for different NLP tasks. The sentiment analysis task utilizes the **Stanford Sentiment Treebank (SST)**, featuring 8,000 movie reviews rated on a 5-point scale. The **QUORA dataset**, comprising 141,000 instances, is used for paraphrase detection, with binary labels indicating paraphrastic similarity. For semantic textual similarity, the **STS dataset** offers 6,000 sentence pairs rated from 0 to 5 based on semantic relatedness. However, there's a significant imbalance as the paraphrase dataset outnumbers the examples for similarity and sentiment analysis by 18-fold, necessitating the reduction of the paraphrase dataset's training size to mitigate bias in the model training process.

5.1.2 Increasing the dataset size

Owing to the success of the paper Halevy et al. (2009), we realizing adding to our dataset would help the model performance. Because we didn't want to distort the results too much, we slightly augmented our dataset size (see Table 1).

Table 1: Training Set sizes

Task	Paraphrase Detection	Semantic Textual Analysis	Sentiment Analysis
	141,506 (Quora)	8,544 (SST)	6,041 (SemEval)
	N/A (N/A)	N/A (N/A)	1,498 (SemEval-STS - 2016 - headlines)
	N/A (N/A)	N/A (N/A)	500 (SemEval-STS - 2015 - images)
Total	141,506	8,544	8,039

The size of the dataset for the sentiment analysis task is now 8,039, up from 6,041. We noticed that sourcing data from various entries in the **SemEval** dataset drastically improved the model performance compared to using identical entries, and noticeably improved the model's capacity to generalize.

5.2 Evaluation method

For evaluating our model on the development set, we adopted the metric utilized on the data board, defined as:

$$f(\hat{y}) = \frac{1}{3}\left(acc_{sentiment} + acc_{paraphrase} + \frac{corr_{similarity} + 1}{2}\right)$$

Our approach involves training the model solely on the training set and then validating it on the development set to select the best-performing model. We acknowledge that repeatedly using this

metric across numerous models may introduce bias toward the development set, as we iteratively test and assess results. However, we have not discovered a more suitable alternative that does not introduce additional data.

5.3 Experiment details & results

5.4 Learning rate and number of epochs

We found that a relatively low number of epochs sufficed for our model to converge—just 10 epochs were enough. We experimented with pretraining our model followed by fine-tuning, comparing it against a model without pretraining. Surprisingly, the results were identical, leading us to skip the pretraining step.

In terms of learning rate, we tested three configurations: 10^{-3}, 10^{-4}, 10^{-5}, determining that the most effective was 10^{-5} based on overall metrics. Therefore, we will continue with this learning rate for subsequent sections.

For our final model, we chose to use a learning rate of 10^{-5} for 10 epochs during the fine-tuning stage. Following this, we performed an additional 2 epochs of refinetuning with a learning rate of 10^{-6} to further optimize the weights for both the BERT model and the various neural networks.

5.5 Dropout

We experimented with three different minBert dropout configurations: 10%, 20%, and 30%, discovering that the optimal choice in terms of overall metrics was minBert dropout set at 10%. Thus, for the subsequent sections, we will consistently use a dropout configuration of 10% between layers.

5.6 Loss functions

For our task, which involves three distinct objectives with varying output dimensions and characteristics, we employed three different loss functions:

- Sentiment Analysis: Cross Entropy given by $\mathcal{L}_{\text{sentiment}}(y, \hat{y}) = -\sum_i y_i \log(\hat{y}_i)$
- Paraphrase Analysis: Binary Cross Entropy (with logits) given by $\mathcal{L}_{\text{paraphrase}}(y, \hat{y}) = -[y \cdot \log(\sigma(\hat{y})) + (1-y) \cdot \log(1-\sigma(\hat{y}))]$, where $\hat{y}$ is the predicted logit (output of the model before applying sigmoid)
- Similarity Analysis: Mean Squared Error (MSE) $\mathcal{L}_{\text{similarity}}(y, \hat{y}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$

These loss functions were chosen as the standard metrics for each respective task. We opted not to experiment with alternative loss functions, as it is widely acknowledged in the literature that these metrics are commonly used for tasks of this nature.

5.7 Architecture

Our strategy involved creating dense layers following the BERT model for each task. We experimented with three types of neural networks: convolutional, feed-forward, and recurrent. Ultimately, we opted to proceed solely with the feed-forward linear neural network, as the convolutional and recurrent networks did not yield satisfactory results.

5.7.1 Method 1 (M1) - Sequentially

In this initial phase, we opted to select the first 6040 data points from all three datasets. This methodology ensures equal emphasis on each task during our model training. The approach involved training three distinct neural networks that directly utilize embeddings from the minBert model, with parameters optimized for each of the three tasks. Training proceeded sequentially, where for each epoch, we iterated through the first dataset, performed backpropagation on the 6k examples, and then repeated this process for the subsequent task.

However, as we progressed through our training, we observed that the gradient descent for one task could conflict with that of another. Therefore, it became necessary to enhance this process by

executing gradient descent in a "favorable" direction for every task concurrently, rather than one after the other.

This procedure is depicted in the figure below (refer to Figure 1).

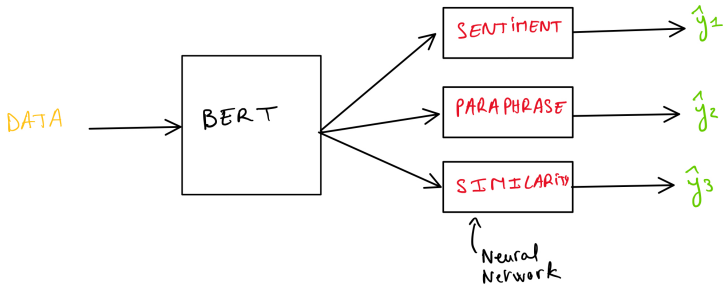

Figure 1: First Training methodology

Sentiment Analysis: 2 layers Neural Network of size (768+4,64)→(64,5) with ReLU activation functions between layers **?**

Paraphrase Analysis: 2 layers Neural Network of size (2×768+3,64)→(64,1) with ReLU activation functions between layers

Similarity Analysis: No layers, we only take $5 \times \text{cosin}(y_1, y_2)$

We initialize the weights of all our layers using a Xavier uniform distribution, while setting the bias to 0.

Our primary focus was on crafting compelling features for the input of our neural network. To achieve this, we devised the following metrics:

- euclidean norm: $||x - y|| = \sqrt{\sum_i (x_i - y_i)^2}$
- average value of the difference
- standard deviation of the difference
- min, max value of the absolute difference
- sentence length (via mask embedding)
- average token embedding
- max token embedding
- std token embedding

5.7.2 Method 2 (M2) - Parallelization of the tasks

In this section, we address a challenge encountered in the first architecture: the gradients of the three tasks may not necessarily be orthogonal to each other, resulting in an inefficient model optimization process. To overcome this issue, we developed a cycling dataset approach. Given that the QUORA dataset is the largest, we created cycles of the STS and SST datasets to iterate over 141k examples for all three tasks. Even though we replicated the STS dataset 17 times, the optimization process became significantly more efficient.

With our newly formed cycled-3-dimension dataset, the next step was to formulate a new loss function that considers each task without overly emphasizing any specific one. The approach was rather straightforward:

$$\mathcal{L}(y_1, y_2, y_3) = \alpha_1 * \mathcal{L}_1(y_1) + \alpha_2 * \mathcal{L}_2(y_2) + \alpha_3 * \mathcal{L}_3(y_3)$$

where index 1 corresponds to SST, 2 to QUORA, and 3 to STS.

However, determining the optimal parameters posed a challenge as the losses did not evolve at the same rate. Initially, our strategy was to set $\alpha_i = \frac{1}{\max(\mathcal{L}_i)}$, aiming to balance the magnitudes of all

three losses. Yet, we realized the need for an improved strategy. Our final, highly effective approach involved adjusting these weights every 5 epochs by taking the inverses of the last 5 losses for each individual loss.

This procedure is depicted in the figure below (refer to Figure 2)

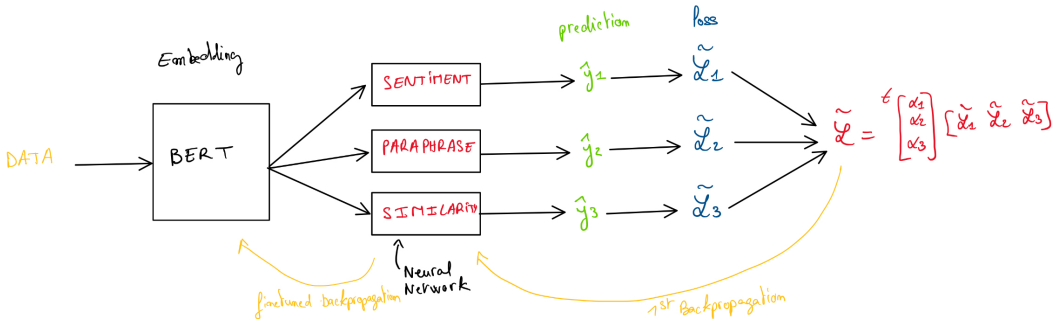

Figure 2: Second Training methodology

6 Resuts & Analysis

As expected our 2nd algorithm converges much faster then the first one, which is completely normal since we make sure to do the gradient descent in the "favorable" direction for the 3 tasks together instead of doing it one task at the time. Our results are as follows:

Table 2: Performance Metrics

Method	Overall score	Sentiment acc	Paraphrase acc	Similarity corr
Sequential - 2 layers similarity	0.718	0.446	0.848	0.72
Sequential - co similarity layer	0.75	0.523	0.853	0.75
Parallelization	0.763	0.515	0.869	0.807

Our highest score was achieved using the parallelization method, which involved adjusting the learning rate, loss coefficient, and introducing complex features. However, we observed a crucial trade-off between the complexity of individual task layers and the overall score.

Interestingly, we found that while it is possible to significantly enhance one metric, such as the paraphrase or similarity task, doing so often resulted in poorer tuning of the BERT model for the remaining tasks. This led us to the realization that for optimal performance, it was more effective to limit the layers to two with a hidden size of 64 for the first two tasks. For the third task, we utilized the cosine similarity output. This approach ensured that our fine-tuning of BERT was well-optimized for all three tasks simultaneously.

7 Conclusion

This project highlights the constant trade-off between overly complex models and the limited availability of data. It became evident that no matter how much we attempted to increase the complexity of our models, there was a ceiling to the development accuracy we could achieve. This limitation is constant across algorithms and models we built, which reflects Pengcheng Yin's lecture. With more intricate models, the biggest bottleneck now appears to be the amount of data available to train the attention matrices. By exploring even smaller BERT models with fewer layers, we encountered the fundamental challenges inherent in highly complex models like ChatGPT.

Ultimately, we explored various methods of regularization, training optimization, structural adjustments, and feature engineering. These techniques all targeted specific aspects of the models, which illustrated the potential improvements that lay ahead for our model.

References

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4171–4186. Association for Computational Linguistics.

Alon Halevy, Peter Norvig, and Fernando Pereira. 2009. The unreasonable effectiveness of data. *IEEE Intelligent Systems*, 24(2):8–12.

Zhihao He and Yue Zhang. 2019. A fine-tuned bert with dense layers for bioinformatics text mining. In *BioNLP 2019*. Association for Computational Linguistics.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computation*, 9(8):1735–1780.

8 Ethic Statement

See `Google Forms`.