BERT but BERT-er

Stanford CS224N Default Project

Hamzah Daud
Department of Computer Science
Stanford University
hamzahdaud@stanford.edu

Abstract

This project leverages the BERT model to enhance sentiment analysis, paraphrase detection, and semantic textual similarity tasks. It begins with an implementation of the BERT model's multi-head self-attention mechanism and transformer layers with an optimizer using the step function of the Adam Optimizer based on Decoupled Weight Decay Regularization. Then, I implemented a multitask classifier using task-specific layers, unique loss functions, and a novel caching mechanism during pretraining to enhance efficiency. The results show that the MGD3-COSIM model, trained on three datasets, outperforms the baseline across all tasks, achieving a particularly high score in semantic similarity. However, when the Yelp dataset is incorporated into training, a notable increase in paraphrase detection accuracy suggests that richer linguistic variety improves performance, although at the expense of other fine-grained semantic tasks. The paper also challenges the authoritativeness of the standard development and test sets, questioning the reliability of "correct" answers and highlighting the complexity of language understanding.

1 Key Information to include

- Mentor: N/A
- External Collaborators (if you have any): N/A
- Sharing project: N/A

2 Introduction

The desire to unravel the complexities of human language so computers can comprehend and process its nuances continues to be an ongoing development in the field of natural language processing. The field has harnessed the mathematical abstractions of language to parse the nuances and subtleties embedded within human communication, such as Word2Vec and GLoVE (Mikolov et al., 2013; Pennington et al., 2014). Tasks such as sentiment analysis, paraphrase detection, and semantic textual similarity are not merely challenges of data processing; they represent intricate puzzles of aligning the abstract, often ambiguous, constructs of language with the precise, definitive realm of mathematical geometry. Through this confluence, the project at hand ventures into the terrain where meaning becomes measurable, where the essence of words and sentences is transmuted into vectors and matrices that offer a tangible grasp on the complex nature of language.

The transformer-based BERT model set new benchmarks across these NLP tasks (Vaswani et al., 2017; Devlin et al., 2018). Its ability to capture deep bidirectional contexts within text made it particularly effective. However, despite its impressive performance, the question of how to fully harness BERT's potential for diverse NLP tasks remains an area of active research. This implementation utilizes task-specific layers, unique loss functions for each task with task-specific loss weighting, and build various other performance-enhancing architectures on the baseline model.

3 Related Work

The concept of fine-tuning of universal language models was spearheaded by Howard and Ruder (2018). They utilized stacked LSTM layers and proposed differential learning rates and a novel progressive unfreezing technique, which delivered state-of-the-art results by capturing the varying levels of linguistic abstraction present in different layers.

Then came Devlin et al. (2018), with bidirectional transformer encoder layers pre-trained on the Masked Language Model (MLM) and Next Sentence Prediction (NSP) tasks, which resulted in embeddings with a deepened contextual understanding. Liu et al. (2019) built on this foundation, finding that by focusing solely on the MLM task with modified training parameters, they could surpass the benchmarks set by BERT without the NSP task.

The fine-tuning phase is critical when adapting BERT to specific tasks. Sun et al. (2019) examined the effects of continued pretraining and the careful calibration of learning rates during task-specific fine-tuning, mitigating the issue of catastrophic forgetting, first identified by McCloskey and Cohen (1989). Their findings emphasize the delicate balance required in adapting general-purpose models to retain learned knowledge while acquiring new information.

Subsequent authors have presented various optimizations to universal language models to enhance performance on specific tasks such as unique loss functions, optimizers, and fine-tuning techniques. For example, Jiang et al. (2020) introduced a novel regularization term that encourages model output smoothness, reinforcing the stability of the learned representations even when input perturbations are introduced. This advancement represents a shift towards a more principled optimization approach for fine-tuning pretrained models, albeit with trade-offs in computational efficiency. Collectively, these works set the stage for my research, providing a framework of strategies and insights that I aim to build upon and extend in this project to improve both the performance and efficiency of the multitask classifier.

4 Approach

I began by implementing the BERT model's multi-head self-attention mechanism and transformer layers (Devlin et al., 2018). For optimization, I implemented the step function of the Adam Optimizer based on Decoupled Weight Decay Regularization (Kingma and Ba, 2015; Loshchilov and Hutter, 2017). This approach ensures that each step taken in the parameter space is scaled appropriately, considering both the amplitude and direction of the gradients. It is defined as:

$$
\begin{aligned}
m_t &= \beta_1 \cdot m_{t-1} + (1-\beta_1) \cdot g_t \\
v_t &= \beta_2 \cdot v_{t-1} + (1-\beta_2) \cdot g_t^2 \\
\theta_{t+1} &= \theta_t - \frac{\alpha}{\sqrt{\hat{v}_t} + \epsilon} \cdot \hat{m}_t \text{ where } \hat{m}_t = \frac{m_t}{1-\beta_1^t} \text{ and } \hat{v}_t = \frac{v_t}{1-\beta_2^t}
\end{aligned}
$$

In the `MultitaskBERT` class, the constructor initializes the BERT model and sets the gradient update policy for its parameters based on the specified mode: 'pretrain' or 'finetune'. Task-specific classifiers are constructed with sequential modules. These classifiers are designed to the dimensional requirements of the respective task function.

The `predict_sentiment` function uses an embedded representation of the input sentence to produce a 5-dimensional logit vector. The sentiment classifier network includes a linear layer, taking BERT's hidden size as both input and output dimension (BERT_HIDDEN_SIZE), followed by batch normalization, GELU activation, dropout, and a final linear layer that maps to the number of sentiment classes (N_SENTIMENT_CLASSES).

The `predict_paraphrase` function combines features from pairs of sentences. It takes the concatenated embeddings (BERT_HIDDEN_SIZE for each) resulting in an input dimension of BERT_HIDDEN_SIZE * 2 for the first linear layer of the paraphrase classifier. Subsequent layers follow a similar pattern of normalization, activation, and dropout before reducing to a single output logit.

The `predict_similarity` function computes the cosine similarity between two sentence embeddings. Given the scalar nature of this similarity score, the first linear layer of the similarity classifier has an input size of 1, which is then expanded to BERT_HIDDEN_SIZE, processed through normalization, activation, and dropout, and finally compressed back to a single value indicating the degree of similarity.

The training procedure for the multitask model integrates the three `DataLoader` objects in a unified training loop. The training loop computes losses individually for each task based on its specific data and label format. For sentiment analysis, the loss is computed using cross-entropy, reflecting the task's classification nature. For paraphrase detection, it uses binary cross-entropy loss, suitable for its binary classification goal. For semantic textual similarity, it uses mean squared error loss, aligning with its regression-based objective. To address the imbalance in dataset sizes, I implemented task weighting for the total loss. After a hyperparameter search, my task weights were set as `'sst': 1, 'para': 0.3, 'sts': 1`. These weights seek to reduce the influence of the large Quora dataset on the model's learning process. I initially implemented inverse proportionality but it resulted in diminished performance, particularly for the paraphrase detection and semantic similarity tasks, as shown in Table 3.

4.1 Novel Caching During Pretraining

Significant computational improvements are realized by caching the [CLS] token embeddings in the `forward` function during pretraining. Since BERT weights remain static during pretraining, the embeddings can be reused, thereby significantly reducing training time. The caching mechanism reduces per-epoch pretraining time on an NVIDIA T4 from 32 seconds to 4 seconds for SST and STS, and from 12 minutes to 1.5 minutes for Quora. **That is an 87.5% reduction in training time.**

A dictionary is maintained that maps sentence IDs to their respective embeddings. When the `forward` function is called, it identifies all pre-computed embeddings from the dictionary and only computes new embedding for unseen sentence IDs, which are then also stored in the cache for future use. Newly computed embeddings undergo dropout as a form of regularization before being cached. Finally, embeddings are collated into a single tensor for subsequent tasks.

4.2 Fine-tuning BERT Weights with Cosine Similarity Loss

During the fine-tuning phase, rather than relying on mean squared error (MSE) loss, which directly compares predicted similarity scores with ground truth, the model employs cosine similarity loss. Inspired by Reimers and Gurevych (2019), the loss function helps optimize the cosine of the angle between two sentence embedding vectors, fostering embeddings that are directionally aligned for semantically similar sentences and orthogonal for dissimilar ones. Figure 1 shows the architecture of the loss.

The `predict_similarity_finetune` function is at the heart of this approach. It generates embeddings for sentence pairs and calculates their cosine similarity, reflecting the geometric notion of semantic proximity. In the training loop, the labels are scaled to a [0, 1] range. This is an adaptation of the original architecture to force the model to process dissimilar embeddings' cosine similarity as 0, which represents orthogonality, and produces better performance than the [-1, 1] range.

4.3 Training on Additional Datasets

I edited the `datasets.py` file to load and pre-process additional data from the Yelp review dataset to experiment with enhancing STS training. The pre-processing filters unnecessary data, adds a unique identifier, and aligns the labelling structure. The data is then loaded into the model using the given functions.

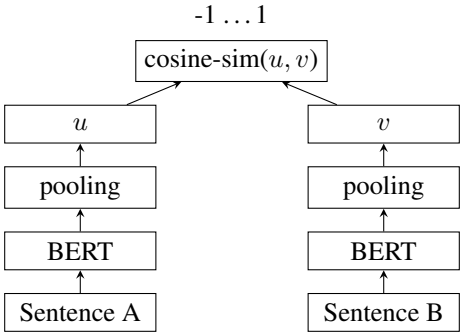

Figure 1: From Reimers and Gurevych (2019): "SBERT architecture at inference to compute similarity scores. This architecture is used with the regression objective function."

5 Experiments

5.1 Data

My model utilizes four datasets provided by the CS 224N staff across the three tasks: Stanford Sentiment Treebank (`SST`) and CFIMDB dataset for sentiment analysis, Quora for paraphrase detection, and SemEval STS Benchmark dataset for semantic textual similarity.

The Stanford Sentiment Treebank (`SST`) categorizes single-sentence strings from movie reviews into five sentiment classes: negative, somewhat negative, neutral, somewhat positive, or positive. For paraphrase detection, I employ a subset of the Quora dataset that labels question pairs as paraphrases or not. The SemEval STS Benchmark dataset for evaluating semantic textual similarity rates sentence pairs on a scale from 0 (unrelated) to 5 (equivalent meaning). Table 1 details number of examples in the `train`, `dev`, and `test` subsets.

Dataset	Train Examples	Dev Examples	Test Examples
Stanford Sentiment Treebank (SST)	8,544	1,101	2,210
CFIMDB	1,701	245	488
Quora	141,506	20,215	40,431
SemEval STS Benchmark	6,041	864	1,726

Table 1: Summary of datasets used for sentiment analysis, paraphrase detection, and semantic textual similarity tasks provided by the CS 224N course staff.

5.2 Evaluation method

The evaluation metrics for the tasks are: accuracy for sentiment analysis and paraphrase classification, and Pearson correlation for semantic textual similarity. I benchmark my `classifier.py` against baselines provided by the CS 224N staff. My `multitask_classifier.py` is compared to the simplest implementation of the three task functions to make them work with the given code (my first submission to the leaderboard). I am also comparing against my submission for the midpoint check-in and the class leaderboard.

5.3 Experimental details

The BERT weights were frozen during pretraining and unfrozen during finetuning. After the first pretraining epoch populated the cache, which took 32 seconds for SST and STS each and 12 minutes for Quora, subsequent epochs took 4 seconds each for SST and STS and 1.5 minutes for Quora. The finetuning took 2 minutes for SST and STS each and 40 minutes for Quora per epoch.

5.4 Hyperparameters

Table 2 lists the hyperparamaters used for pretraining, finetuning, and the AdamW optimizer.

Hyperparameter	Value
Epochs	10
Hidden Dropout Prob	0.3
Pretraining Learning Rate	1×10^{-3}
Finetuning Learning Rate	1×10^{-5}
Batch Size	8
AdamW Optimizer Learning Rate	1×10^{-3}
AdamW Betas $[\beta_1, \beta_2]$	$[0.9, 0.999]$
AdamW Epsilon (ϵ)	1×10^{-6}
AdamW Weight Decay	0
Task Loss Weights	'sst': 1, 'para': 0.3, 'sts': 1

Table 2: Hyperparameter selections for pretraining, finetuning, and the AdamW optimizer. Task Loss Weights are the scalars applied to the loss functions for the three tasks.

5.5 Hyperparameter Search

I only considered the results on the `dev` set after pre-training when conducting the hyperparameter optimization to limit repetitive finetuning, which is time and compute expensive. A more comprehensive hyperparameter search–covering more hyperparameters and including post-finetuning results–is possible if further compute and financial resources are available.

A selection of results from various experiments with hyperparameter adjustments are shown in Table 3. If the hyperparameter is not marked as adjusted, then it is set as the default listed above.

	SA	**PD**	**STS**	**Overall Score**
Model	Accuracy	Accuracy	Pearson Correlation	
Batch size: 32	0.392	0.708	0.256	0.576
Batch size: 16,64,16	0.383	0.702	0.256	0.571
AdamW Weight Decay: 1×10^{-4}	0.432	0.648	-0.256	0.608
Epochs: 20	0.398	0.712	0.256	0.579
Pre-training LR: 1×10^{-5}	0.363	0.551	0.256	0.514
Task Loss Weights: [1, 0.5, 1]	0.477	0.688	0.256	0.598
Task Loss Weights: [0.4, 0.023, 0.57]	0.461	0.375	0.127	0.321
Hidden Dropout Prob: 0.1	0.407	0.685	-0.256	0.612

Table 3: Hyperparameter search on MGD3-COSIM (best model) on the `dev` sets.

6 Results

My final model, MGD3-COSIM, yielded performance over the baselines on the `test` set, as shown in Table 4.

	Sentiment Analysis	**Paraphrase Detection**	**Semantic Similarity**	**Overall Score**
Model	Accuracy	Accuracy	Pearson Correlation	
MGD3-COSIM	0.466	0.796	0.680	0.701

Table 4: Performance comparison of multitask classifiers on the `test` sets.

Table 5 describes the five models I developed and trained. Table 6 shows the results for the various models on the `dev` sets. The `classifier.py` results can be found in Table 7 in the Appendix.

I compared my various model results on the `dev` set results, as detailed in Table 5 and Table 6 show that the MGD3-COSIM model stands out with the highest overall score (0.725), a promising indication that the model architecture, training procedure, data, and hyperparameter selection yield robust performance.

The MGD4-Y model, which included Yelp reviews in its training data, showed an improvement in Paraphrase Detection (0.802) compared to the MGD3-COSIM, suggesting that additional data can

Model Name	Description
MGD3-COSIM	**M**ultilayer classifiers, **G**ELU activation, **D**ropout, trained on **T**hree datasets (SST, Quota, and STS). Cosine similarity loss finetuning.
MGD4-Y	**M**ultilayer classifiers, **G**ELU activation, **D**ropout, trained on **F**our datasets (SST, Quota, STS, and **Y**elp reviews).
MGD3	**M**ultilayer classifiers, **G**ELU activation, **D**ropout, trained on **T**hree datasets (SST, Quota, and STS).
Midpoint	Linear classifiers without activation functions. Dropout applied to embeddings. Trained on SST, Quota, and STS datasets.
Baseline	Simple implementation with linear classifiers and absolute difference calculations for similarity and paraphrase. Only trained on SST data.

Table 5: Descriptions of various models.

	Sentiment Analysis	**Paraphrase Detection**	**Semantic Similarity**	**Overall Score**
Model	Accuracy	Accuracy	Pearson Correlation	
MGD3-COSIM	**0.504**	0.790	**0.761**	**0.725**
MGD4-Y	0.457	**0.802**	0.595	0.685
MGD3	0.463	0.789	0.743	0.708
Midpoint	0.432	0.648	0.406	0.594
Baseline	0.477	0.389	0.114	0.474

Table 6: Performance comparison of multitask classifiers on the `dev` sets.

enhance performance for certain tasks. However, it underperformed in the Semantic Similarity task (0.595) and had a lower overall score (0.685), indicating that while the richer linguistic variety aids paraphrase recognition, stylistic differences and potential noise from the Yelp dataset may disrupt the model's performance on tasks requiring finer semantic discernment.

Both the Midpoint and Baseline models trailed behind the more advanced models, which was expected given their simpler architectures and training datasets. The Baseline model's particularly low score in Semantic Similarity (0.114) and Paraphrase Detection (0.389) accentuates the importance of a more sophisticated model design and training regimen for complex NLP tasks.

These outcomes suggest that while incorporating additional datasets can be beneficial, careful consideration must be given to task compatibility and the risk of domain overfitting. The results also underscore the significance of tailored architectures and highlight the potential for further refinements in model training strategies to achieve consistent performance improvements.

7 Analysis

After an initial improvement, Figure 2 and Figure 3 show the `dev` set score exhibits slight fluctuations, peaking at Epoch 6 during pre-training and Epoch 4 during fine-tuning, then slightly declining or stabilizing thereafter. The observed patterns suggest that while the model is becoming more confident in its predictions over time (as evidenced by the decreasing loss), this does not necessarily translate to consistent improvements in accuracy on the development set. This is likely due to overfitting, where the model learns the training data too well and fails to generalize to new data. It's also possible that the model has reached its performance capacity given the current architecture and training data, suggesting that further improvements may require architectural changes, additional data, or more nuanced training techniques. The MGD3-COSIM model demonstrates a solid capacity for learning, but its fluctuations in development accuracy reveal the inherent challenges in achieving consistent performance gains across training epochs.

7.1 Questionable Authoritativeness of Dev and Test Set Labels

The authoritativeness of the labels given to sentences on the `dev` and `test` sets are questionable. For example, my model classifies the sentence "It's a lovely film with lovely performances by Buy and Accorsi" as a 4 (indicating somewhat positive). However, the official score given to this sentence is a 3 (neutral). Another example is the sentence "You won't like Roger, but you will quickly recognize

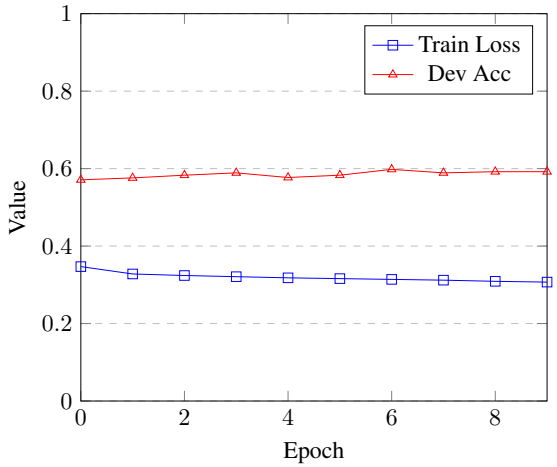

Figure 2: MGD3-COSIM Pre-Training Loss and Dev Accuracy

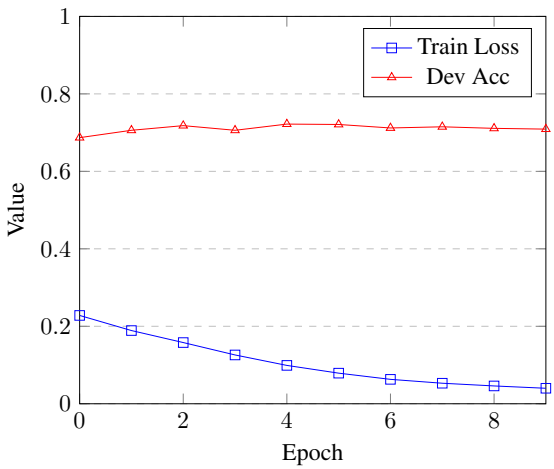

Figure 3: MGD3-COSIM Fine-Tuning Loss and Dev Score

him." My model predicts this sentence is a 2 (somewhat negative), but the official score is a 1 (negative). A third example is the sentence "As unseemly as its title suggests." My model predicts this sentence is a 1 (negative), but the official score is a 3 (neutral). These are just a selection of the many examples in the benchmark that have questionable "correct" answers.

8 Conclusion

The project explored the efficacy of BERT-based models across a spectrum of NLP tasks including sentiment analysis, paraphrase detection, and semantic textual similarity. My key contribution, the MGD3-COSIM model, demonstrated a commendable performance with the highest overall score of 0.725 on the `dev` set and 0.701 on the `test` set, which suggests that my approach to fine-tuning BERT with task-specific layers and loss functions is effective. Furthermore, the inclusion of additional data from Yelp reviews in the MGD4-Y model led to improved paraphrase detection, underscoring the potential benefits of training on diverse datasets.

A significant finding was the diminishing returns in sentiment and semantic similarity tasks with the addition of the Yelp dataset, which may point to the challenge of balancing domain-specific knowledge with generalizability. My implementation of a novel caching system during pretraining also achieved a substantial reduction in training time by 87.5%, showcasing a practical advancement in computational efficiency.

The limitations of our work became apparent through the fluctuations in development accuracy, hinting at potential overfitting and the constraints of model capacity. Furthermore, I question of the authoritativeness of the dev and test sets as evidenced by the imperfect labelling examples.

Future work could explore the effects of additional linguistic features or alternative datasets to mitigate domain discrepancy. Investigating more sophisticated regularization techniques might also address overfitting. Lastly, re-evaluating the scoring of benchmark datasets or developing new, more nuanced evaluation metrics could contribute to more reliable measures of model performance.

References

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805.

Jeremy Howard and Sebastian Ruder. 2018. Fine-tuned language models for text classification. *CoRR*, abs/1801.06146.

Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Tuo Zhao. 2020. SMART: Robust and efficient fine-tuning for pre-trained natural language models through principled regularized optimization. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2177–2190, Online. Association for Computational Linguistics.

Diederik Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, San Diega, CA, USA.

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

Ilya Loshchilov and Frank Hutter. 2017. Fixing weight decay regularization in adam. *CoRR*, abs/1711.05101.

Michael McCloskey and Neal J. Cohen. 1989. Catastrophic interference in connectionist networks: The sequential learning problem. volume 24 of *Psychology of Learning and Motivation*, pages 109–165. Academic Press.

Tomás Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Distributed representations of words and phrases and their compositionality. *CoRR*, abs/1310.4546.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar. Association for Computational Linguistics.

Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. *CoRR*, abs/1908.10084.

Chi Sun, Xipeng Qiu, Yige Xu, and Xuanjing Huang. 2019. How to fine-tune BERT for text classification? *CoRR*, abs/1905.05583.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008.

A Appendix

Task	Dev Accuracy	Delta to Baseline
Pretraining for SST	0.462	+0.072
Pretraining for CFIMDB	0.878	+0.098
Finetuning for SST	0.524	+0.009
Finetuning for CFIMDB	0.959	-0.007*

Table 7: Performance of the single task classifier compared to the baseline provided by the CS 224N course staff.

* *within 1 standard deviation of baseline*