Optimizing Multi-Task Classification Finetuning in BERT: a Multi-Pronged Approach

Stanford CS224N Default Project; Mentored by Hans Hanley

Bar Weiner
Department of Computer Science
Stanford University
`barw@stanford.edu`

Aadi Nashikkar
Department of Computer Science
Stanford University
`aadinash@stanford.edu`

Soham Konar
Department of Computer Science
Stanford University
`skonar@stanford.edu`

Abstract

The development of BERT (Devlin et al., 2018) was a state-of-the-art performance improvement in several natural language processing tasks. In this paper, our intent was to extend this performance to multitask use cases: paraphrase detection, semantic textual similarity, and sentiment analysis. In exploring improvements to our model we focus on six distinct optimizations: multitask finetuning for training all 3 tasks together with an aggregated loss function, using cosine similarity on the comparison task of predicting similarity, using ReLU layers in our architecture to improve performance, parameter optimizations for dropout and weight decay distinctly chosen to address fitting issues in different tasks, loss coefficient refinement to reweight training emphasis, and reworking attention to include linear biases and slope optimizations. Our primary findings were that cosine similarity layered with ReLU activation improved accuracy in comparison tasks, multitask finetuning massively increased performance when performing multitask learning, and that tuning dropout and loss function coefficients effectively combated overfitting and increased overall accuracy in multitask systems. We also found that more work must be done on linear biases in attention calculations to have them improve results in models like BERT.

1 Introduction

Our project investigates how to modify a single, large base BERT model to perform a number of tasks. Unmodified, BERT does not perform strongly on several "downstream" tasks like sentiment analysis and paraphrase detection, as seen in section 5 (Experiments). In the multi-task learning paradigm, machine learning models are trained utilizing data from several tasks at once, employing shared representations to discover the commonalities among a group of related tasks. These shared representations improve data efficiency and may result in quicker learning speeds for connected or subsequent tasks, assisting in addressing deep learning's well-known drawbacks of high computation and data requirements. Using new techniques found in research completed since the release of BERT in 2018, we augment the base BERT model to iteratively improve it for certain downstream tasks.

2 Related Work

In the paper "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks," the authors use a Siamese neural network architecture, which consists of two BERT models with shared weights, to learn the embeddings. The network takes two input sentences and computes their embeddings which are then compared using cosine similarity. In our project, we replicated this work and added architecture such as a ReLU activation on top of our cosine embeddings.
In their paper "Multi-task Learning over BERT for News Recommedation", Bi Et AL propose a method to train BERT on multiple tasks like news topic classification, news subtopic classification, and news recommendation. The model learns shared representations across tasks, leading to improved performance on all tasks. The authors show that this outperforms several state-of-the-art methods. We replicated their approach between our tasks and saw similarly improved performance.
Next the paper "Train Short, Test Long: Attention with Linear Biases Enables Input Length Extrapolation" proposes a new method for better performance on longer input sequences by introducing a neural net modification which involves adding linear biases to the attention weights, while also removing the default positional embeddings matrix. We extended their initial work, which was intended for unidirectional models, and evaluated its performance on the aforementioned tasks which all had varying lengths on input training data.
While these papers did not directly relate to each other, we used them as we saw the weaknesses in our model arise. We found multi-task loss to improve the model's shared representations across tasks, but then we focused on improving our Semantic Text Similarity (STS) with the Siamese BERT-Network paper. We used linear biases to improve our model on tasks where positional embeddings would not be learned as well (such as in sentiment analysis, where movie reviews vastly differed in length), and then also sampled findings from other research work to improve our model further.

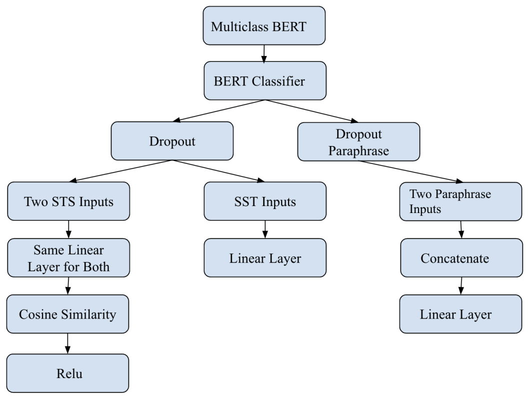
3 Approach

Initially, for minBert we implemented Multi-head Self-Attention, the Transformer Layer, and the AdamW Optimizer. For more details on the architecture, we refer the reader to the default project handout or minBert paper. We now go into our specific implementation and architecture.

3.1 Multitask Learning: Simultaneous Training

From the beginning, rather than fine-tuning BERT on individual tasks, we made use of multi-task learning to update BERT. In this training method, we used multi-task learning and added each together each loss on the tasks at hand and optimized them all together.

Figure 1: Neural Network Architecture for Our 3 Tasks



To predict sentiment we passed our sentence embeddings through a linear layer with hidden size 768 by 5 (our number of sentiment classes). We then utilize the cross entropy loss function and sum it with the other losses to compute a total loss. In the cross entropy loss formula below p_x is the true (one-hot) probability distribution of the class, and q_x is our predicted probability distribution.

$$\mathcal{L}_{SST} = -\sum_{x \in sentiments} p_x \log(q_x)$$

As described in more detail below, for similarity we pass our two sentence embeddings through the same linear one after the other. Our similarity PyTorch linear layer is of dimension 768 by 100 (an arbitrarily large layer size which worked well in tests). We then take the cosine similarity of these two outputs, and activate the similarity with ReLU which we then return. To compute the loss we use the MSE loss function and scale our ReLU output by 5, to ensure that the range is ∈ [0,5]. In the formula below x_i is the ith observed value, and y_i is the corresponding predicted value, and D is the number of observations.

$$\mathcal{L}_{STS} = \sum_{i=1}^{D} (x_i - y_i)^2$$

Finally, for the paraphrase prediction task we use a linear layer of dimension 768 * 2 by 1, as we will be passing in both input sentences simultaneously through the network. On those lines, we simply concatenate our two sentences together and pass them through our PyTorch linear layer to get an unormalized logit for paraphrase. We then use the binary cross entropy loss function on this output as our labels are either that the sentences are 0. Paraphrased or 1. Non paraphrased. In our formula below p is our probability of a positive paraphrase prediction label , and y_i is the value of y at i.

$$\mathcal{L}_{Para} = -\frac{1}{n} \sum_{i=1}^{N} (y \log(p) + (1 - y) \log(1 - p))$$

This finally comes together to form our total loss which is the sum of the losses:

$$\mathcal{L}_{Total} = \mathcal{L}_{SST} + \mathcal{L}_{Para} + \mathcal{L}_{STS}$$

Training Loop: In our training loop we take batches and load in data until we exhaust the longest dataset finished, and restarted our iterators over the smaller two datasets as we iterated and exhausted them as well. We attempted other solutions like looping until the smallest one finished, or looping over the length of the average of the 3 datasets. The method of looping until we exhausted taking batches from the longest dataset, yielded the best results; nevertheless, in future work we could try to randomly sample or take a weighted sample from the longest dataset to guarantee equal training.

3.2 Cosine and Architectural Optimizations for Similarity Prediction

In our approach for similarity prediction, we determined that passing both outputs through the same linear layer allows the model to learn a shared representation of the input sentences, capturing their similarity. This representation is a fixed-size vector encoding the most relevant information from both.

We then compute cosine similarity on our inputs, as described in the equation below, where a result of -1 means the vectors are opposite, 1 means they are identical, and 0 means they are orthogonal.

$$s = \frac{x \cdot y}{||x|| \cdot ||y||} = \frac{\sum_{i=1}^{n} x_i \times y_i}{\sqrt{\sum_{i=1}^{n} (x_i)^2} \times \sqrt{\sum_{i=1}^{n} (y_i)^2}}, s \in [-1, 1]$$

3.2.1 ReLU

We then activate the similarity, z, through a ReLU layer (equation below) [1]. This adjusts the negative similarity scores to zero and sets the range of our output s to be ∈ [0,1] (we scale this later to fit into the sentiment analysis range of [0,5]). For sentence similarity, we used ReLU to clip negative similarity scores to zero because negative similarity scores do not provide meaningful information about the similarity between sentences.

$$ReLU(z) = max(0, z)$$

3.3 Hyperparameter Tuning and Loss Refinement

Dropout is a regularization technique that helps to prevent overfitting by randomly setting to zero some portion of the nodes during training, and prevents certain node interdependencies from being continually reinforced over training data, which can help with overfitting [2]. In our approach, we tried different dropout fractions from the default of 0.3. Ultimately, we modified our hyperparameters by implementing separate `forward` functions in multitask_classifier.py with dropout rates based on the specific losses on each task. When we noticed that our paraphrase prediction train accuracy was similar to the dev accuracy, we chose a dropout rate of only 0.1 for its network to increase its power. We also experimented with other dropout rates in other tasks.

Weight decay is another regularization technique we tried to prevent overfitting, as it encourages the model to use smaller weights, which in turn helps to prevent the model from overemphasizing certain features or memorizing patterns specific to the training data [3]. To achieve this, we added a penalty term (L2 regularization) to the loss function proportional to the magnitude of the weights in the network which discouraged the model from using large, non-generalizeable weights.

3.3.1 Loss Coefficient Tuning

Furthermore, to respond to overfitting and underfitting issues in our implementation, we also changed the hyperparameters on our simultaneous training loss function. Initially, we had

$$\mathcal{L}_{Total} = \mathcal{L}_{SST} + \mathcal{L}_{Para} + \mathcal{L}_{STS}$$

which we refined to

$$\mathcal{L}_{Total} = (0.5)\mathcal{L}_{SST} + (1.5)\mathcal{L}_{Para} + (0.75)\mathcal{L}_{STS}$$

after some experimentation with trying to reduce the overfitting on SST and STS while trying to ameliorate the underfitting issue we had on paraphrase detection.

3.4 Attention with Linear Biases

Positional representations are a critical piece of useful self-attention computations. BERT uses a positional embedding matrix natively and learns these embeddings as part of the training process [4]. However, a more direct approach is to directly change the attention calculation and give higher attention values for nearby words than faraway words, in reference to some center word for which you are calculating self-attention. A basic implementation of this is as follows:

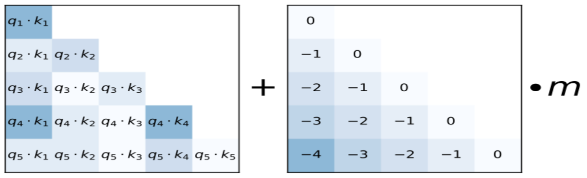
$$\alpha_i = softmax(k_{i:n}q_i + [-i, -1, 0, -1, ...(n-i)])[5]$$

where $k_{i:n}q_i$ are the original pre-normalization attention scores, and then we add bias such that the attention scores for words that are farther from word i are penalized. Thus, we generate the key-query products as normal, and add a linear bias penalty when the key is far from the query in the sentence.

3.4.1 Linear Slope Optimization

The implementation of this concept that we added to our final project comes from the paper "Train Short, Test Long..." mentioned in our related works section. In addition to the linear bias matrix, we added a head-specific scalar m that is a "slope" factor and reduces from 1 to 0 across heads. The authors found that a geometric sequence over heads ($\frac{1}{2^1}$, ..., $\frac{1}{2^n}$) for n attention heads gave the best empirical results. The affect of this extension can be visualized below from the original paper. However, the original architecture was designed for causally masked decoder self-attention, so it was described for unidirectional models. We took this concept but extended it to bidirectional models by writing original code to make this biases matrix bidirectional (symmetrical) and then applying the separate m scalar to each of these matrices before adding them to the key-query product matrices. We used the given code only to generate the list of scalars in the geometric sequence.

Figure 2: A linear bias matrix augments the typical attention calculation[4]


$q_1 \cdot k_1$
$q_2 \cdot k_1$ $q_2 \cdot k_2$
$q_3 \cdot k_1$ $q_3 \cdot k_2$ $q_3 \cdot k_3$
$q_4 \cdot k_1$ $q_4 \cdot k_2$ $q_4 \cdot k_3$ $q_4 \cdot k_4$
$q_5 \cdot k_1$ $q_5 \cdot k_2$ $q_5 \cdot k_3$ $q_5 \cdot k_4$ $q_5 \cdot k_5$
+
0
-1 0
-2 -1 0
-3 -2 -1 0
-4 -3 -2 -1 0
$\bullet\, m$

4 Experiments

4.1 Data

The first data set we used is Stanford Sentiment Treebank which consists of 11,855 single sentences extracted from movie reviews and parsed into phrases [6]. Each phrase has a label of negative, somewhat negative, neutral, somewhat positive, or positive. The second data set we used was a data set provided by Quora which consists of 400,000 question pairs with labels indicating whether particular instances are paraphrases of one another [7]. The last data set we used was the SemEval STS Benchmark dataset which consists of 8,628 different sentence pairs of varying similarity on a scale from 0 (unrelated) to 5 (equivalent meaning) [8].

4.2 Evaluation method

For the sentiment analysis and paraphrase detection tasks, our evaluation metric was accuracy of label-prediction. For the STS task, we use the Pearson Correlation metric, where a higher correlation metric indicated better predictions of the human-rated similarity score for various sentence pairs. We compared the score we received for these evaluation metrics on our extensions versus our baseline on the held out SST, STS, and Quora/Paraphrase dev sets.

4.3 Experimental details

All experiments were implemented using PyTorch. While training, we used the hyperparameters of 0.3 for dropout probability, a batch size of 64, and a 10e^-5 learning rate for 10 epochs. We experimented with other parameters, which we describe in our results section. The timing per epoch during training and testing for our multitask classifier was around 1200 seconds on Colab GPUs.

4.4 Results

We find that our extensions increases in performance across the board from the baseline BERT model. Using an iterative process, we experimented with several extensions to respond to area our model was weaker in. We found that the most significant gains in performance came from multitask fine-tuning (for all of the tasks) and from adding cosine similarity and ReLU activation to the STS task. Our final model improved on our baseline by adding multitask learning, cosine similarity, ReLU activation, changes to the loss function, and hyperparameter tuning.

4.5 Results Comparison to Baseline

	SST	Paraphrase	STS	Overall
Milestone buggy implementation (Dev)	0.312	0.380	0.020	0.237
Sequential Multitask Training Baseline (Dev)	0.296	0.625	0.174	0.365
Multitask Finetuning (Dev)	0.500	0.753	0.405	0.552
Cosine Similarity (Dev)	0.498	0.759	0.475	0.577
Cosine Similarity + ReLU (Dev)	0.501	0.753	0.702	0.652
Cosine Similarity + ReLU and Linear Biases (Dev)	0.470	0.763	0.668	0.634
Final + Hyper-parameter Tuning (Leaderboard Dev)	**0.503**	**0.766**	**0.709**	**0.659**
Final + Hyper-parameter Tuning (Leaderboard Test)	**0.525**	**0.766**	**0.699**	**0.663**

Results for various stages of our extensions. Any results after the "Multitask Finetuning" (row 3) implicitly include the use of Multitask Finetuning in the extension. Note also that the performance when including linear biases dropped; however, performance on certain samples increased as discussed in section 6. The final model included modifications to the loss function, but we did not specifically mention them as a separate result since the effect of these changes was insignificant.

4.5.1 Refining the Multi-Task Training Backbone

When first deciding our approach for multi-task training, we trained our plain minBert implementation on each data set one at a time and had individual losses for each of the three tasks in this order: first STS, then SST and then Paraphrase. We realized that this was an ineffective as it caused our model to "forget" the datasets that we chose to train on first. We can see this reflected in the results below. Ultimately, we used this model as our baseline to compare our extensions to throughout the paper, as our model for the milestone had errors in the code as we were not implementing core BERT features correctly.

SST	Paraphrase	STS	Overall
0.296	0.625	0.174	0.365

Once we caught the error of sequentially learning datasets, we switched to the approach described above in multi-task learning and utilizing a shared final loss function to optimize upon.

SST	Paraphrase	STS	Overall
0.500	0.753	0.405	0.552

This helped all of our results, and specifically aided SST and STS as predicted. Nevertheless, as alluded to by Bi. Et Al there may exist "gradient conflicts" across our different tasks which we could have addressed using a form of Gradient Surgery. In future explorations we can experiment and see how much of an improvement Gradient Surgery can have on our version of multitask fintetuning.

4.5.2 Improving STS Through Cosine Similarity

After applying multitask learning, we still had noticeably weak results in the STS task as seen below.

SST	Paraphrase	STS	Overall
0.500	0.753	0.405	0.552

To improve our model architecture, we first implemented cosine similarity into our prediction architecture, as the literature has clearly shown that this can improve performance in comparison tasks[9]. This resulted in a small bump in Pearson correlation. However, to achieve our final accuracy, we modified our architecture. Initially, we were passing our two embeddings into their own PyTorch linear layers, ultimately assuming that the first sentence is on the "left", and the other one is on the "right" in the comparison. Thus, we modified our model to only use one linear layer for both inputs: there was no reason to encode both twice, as it is a symmetric task. We then also used a ReLU activation layer on top of the cosine similarity calculation. We thus expected the large improvement we saw in STS:

SST	Paraphrase	STS	Overall
0.501	0.753	0.702	0.652

Initially, the cosine output of [-1,1] scaled to [0,5] directly, meaning that a cosine output of -1 mapped to the minimum score of 0. However, this meant that to output a minimum score of 0, it required extreme cosine dissimilarity, not just a "lack" of similarity. In other words, sentences not only had to different meanings, but also had to have opposite embeddings to receive a minimum score of 0. Adding a ReLU removes the effect of negative cosine scores so that embeddings can be learned without this noise. One specific example that shows the benefit of ReLU activation was "There's a geek answer to this, and a practical answer to this. It's pretty ridiculous that I've seen airlines ask for these to be turned off at times." In the correct data and the improved new model this received a label of 0, while it received 2.4 in the old model using cosine similarity without ReLU. This is because, the cosine similarity was negative, and was mapped to 2.4. However, after the ReLU activation, the negative embedding correctly adjusted to a correlation of 0.

4.5.3 Attention with Linear Biases

The addition of attention with linear biases to our model caused a drop in overall performance of 0.018. While this is not a large number, it was still significant and demonstrated that our implementation of attention with linear biases did not improve performance on our tasks. We hypothesize that this could have been the case, despite the authors' in "Train Short, Test Long..." empirical support for it, because the validation done by the authors was done on unidirectional models, so it has been less studied for bidirectional models. Additionally, positional embeddings are removed in this approach, so we lose the learned positional representations that BERT introduced.

4.5.4 Hyper-Parameter Tuning and Loss Refinement

After implementing these extensions and excluding attention with linear biases, our performance was improving, but we had significant problems with both overfitting and underfitting. Our paraphrase accuracy was nearly identical for both the train and dev data, whereas there were large disparities for STS and sentiment prediction. Thus, we looked for extensions that would customize the learning of each task. To do this, we did hyper-parameter search and tuning, as well as modifying the loss function with weight decay. Notably, we saw a 0.013 improvement in paraphrase detection. We expected this because our paraphrase-specific loss was originally highest compared to the other tasks before we tuned the hyperparameters. By specifically decreasing the dropout while learning on paraphrase data and then increasing the coeefficient weight on paraphrase loss while doing multi-task learning, we increased the strength of the model. Unfortunately, the introduction of weight decay did not make impact like hyperparameter tuning. Ultimately, we found values for the other loss coeefficients (STS loss and Sentiment loss) and dropout rates through a brief search that led to the highest performance, as seen in the two final rows of the results table.

5 Analysis

5.1 Sentiment Analysis Task: SST

Sentence: It seems like I have been waiting my whole life for this movie and now I can't wait for the sequel.

Model Value: 0
Actual Value: 3

The model most likely did bad on this example because the word "can't" has a generally negative connotation but because the sentence says "can't" in the context of the phrase "can't wait", it is actually positive. If the model encountered the slang phrase "can't wait" more often during its training and realized that it is a positive phrase then maybe it would have reported better results during testing. For this task, our distribution of accuracies is spread out as as follows: 36.7, 56.4, 36.2, 58.4, 56.9 percent accuracies for the 0, 1, 2, 3, 4 sentiment labels respectively. There is no singular label that has very weak performance. However, with more data we maybe could have upped our abilities on the 0 and 2 sentiment labels as well.

5.2 Paraphrase Task

Sentence 1: What are some good badminton rackets?
Sentence 2: Which is the best badminton racket?

Model Value: 0
Actual Value: 1

In our final hyperparameter tuning we pushed our model to fit more aggressively to the paraphrase dataset. This means that it is less likely to be able to recognize a paraphrase example it hasn't trained on in the future. This is a paraphrase example, but our model is not able to pick it up despite there being lots of similar words and structure. We also potentially could have resolved this by training on more sets of sentences which also paraphrase by switching out question terms like "what" and "which".

5.3 Similarity Prediction Task: STS

Sentence 1: Man with knife arrested at entrance to Buckingham Palace
Sentence 2: Man Charged After Buckingham Palace Arrest

Model Value: 4
Actual Value: 2.2

The model most likely predicted that the sentences are paraphrases of each other because the sentences used almost all of the same vocabulary even though the second headline was missing some key words that the first headline had. With more training data the model could better learn to attend to the words in the sentences and learn the meanings of the words more precisely rather than just comparing the direct inclusion of the words.

5.4 Analysis of Linear Biases on Paraphrase Detection

Attention with linear biases was shown to improve perplexity on longer inputs of unidirectional language models by changing the method of positional embedding [5] However, we could not replicate the same improvements in the stated downstream tasks for BERT but wanted to still carefully investigate the effects of attention with linear biases. Specifically, we analyzed the Quora dev dataset by segmenting according to the length of the first comparison sentence. In the distribution of first sentence length, 75% of sentences were less than 13 words. In the upper quartile subset of the data (13 words or longer), we found that the addition of linear biases to the attention calculation *improved* accuracy non-trivially. We saw an improvement of classification accuracy from 0.614 to 0.625. On the other hand, accuracy was more similar for the other quartiles. We believe this increase in performance on the upper quartile of data is notable since it relates to the strengths of the linear bias approach: when there is less data in higher positions in the training data, simple linear biases without positional embeddings outperform learned positional embeddings. Since there is less data for high positional embeddings, there is higher variance in the contribution of these learned embeddings to model outputs. Thus, we hope to see more work in applying linear biases to bidirectional models, especially if they can be used for tasks where there is more need for extrapolation on longer sentences.

6 Conclusion

The purpose of this project was improve BERT, in its ability to perform the three following tasks: sentiment analysis (SST), paraphrase detection, and semantic text similarity (STS). We started with multitask finetuning which greatly increased the accuracy of BERT on new tasks. Nevertheless, fundamentally, there is potential to explore this further using methods such as gradient surgery or a more sophisticated training loop. We then implemented a ReLU activation over cosine similarity to get better results on the semantic text similarity comparison tasks and increase efficiency of this learning cycle by zeroing out negative results. After that, we tried two forms of hyper-parameter tuning: tuning dropout rate based on the current task and weight decay to prevent the model from accidentally over-prioritizing certain weights. These showed some positive results but we took the concept further by changing the weights of the hyper-parameters in our simultaneous training loss function to reduce the under fitting we had during paraphrase detection.

Next, we also experimented introduced attention with linear biases and implemented it using linear slope optimization so that position and proximity of words played a larger role in the self-attention calculation. In our analysis, we found that linear biases only improved performance on longer testing examples where learned positional embeddings might have greater variance with less data to train on. In future work, we would see how linear biases could be better used in bidirectional models like BERT, as our results currently show decreased performance.

One limitation we faced during our work was that there was not we didn't go beyonds the scope of the datasets to find additional training data. By providing more data, our overall performance on specific tasks could have improved and been able to generalize to unknown blinded data better. Further future work on this project would also include getting more data to pretrain and test the model on and potentially improving our overall training loop and loss calculations with improvements such as Negative Ranked Weight Loss.

References

[1] Abien Fred Agarap. Deep learning using rectified linear units (relu). *CoRR*, abs/1803.08375, 2018.

[2] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.

[3] Anders Krogh and John Hertz. A simple weight decay can improve generalization. In J. Moody, S. Hanson, and R.P. Lippmann, editors, *Advances in Neural Information Processing Systems*, volume 4. Morgan-Kaufmann, 1991.

[4] Ofir Press, Noah A. Smith, and Mike Lewis. Train short, test long: Attention with linear biases enables input length extrapolation, 2022.

[5] John Hewitt. [draft]note 10: Self-attention transformers.

[6] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642, Seattle, Washington, USA, October 2013. Association for Computational Linguistics.

[7] Samuel Fernando and Mark Stevenson. A semantic similarity approach to paraphrase detection. 2008.

[8] Eneko Agirre, Daniel Cer, Mona Diab, Aitor Gonzalez-Agirre, and Weiwei Guo. *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, June 2013. Association for Computational Linguistics.

[9] Nils Reimers and Iryna Gurevych. Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3982–3992, Hong Kong, China, November 2019. Association for Computational Linguistics.