Sentence part-enhanced minBERT: Incorporating sentence parts to improve BERT performance on downstream tasks

Stanford CS224N Default Project

Aaron Wan
Department of Computer Science
Stanford University
aaronwan@stanford.edu

Abstract

BERT-based models typically consider the embedding of the classification token when representing sentence meaning, meaning they do not take into account the different levels of importance of different sentence parts when applying BERT embeddings to downstream tasks. The goal of this project was to implement the default project's multitask minBERT model and improve its performance on sentiment analysis, paraphrase detection, and semantic textual similarity by incorporating sentence parts into the minBERT model for the respective downstream tasks. Sentence parts were incorporated in the same manner as the sentence part-enhanced (SpeBERT) model. We also experimented with gradient surgery, cosine similarity fine-tuning, and SMART optimization to improve model results. Our findings support the findings of the SpeBERT authors, as we find that incorporating main sentence parts substantially improves performance on semantic textual similarity, and other sentence parts improve performance on sentiment classification. Like the SpeBERT authors, we also find selecting the correct aggregation strategy is important for optimizing Spe-model performance. However, we find that for paraphrase detection, incorporating main or other sentence parts does not appear to improve performance. A key finding is that Spe-based models performed better in absence of multitask learning. It may be beneficial to avoid multitask learning if downstream tasks require different sentence parts since this allows the model to focus on information from inherently different embeddings. Our best model was an ensemble model of three Spe-based models trained for different tasks. By incorporating sentence parts for sentiment classification and semantic textual similarity, our ensemble model achieved a dev set score of 0.715 and a test set score of 0.705.

1 Key Information to include

- Mentor: Shai Limonchik
- External Collaborators: N/A
- Sharing project: N/A

2 Introduction

Many BERT-based models only consider the embedding of the classification token when representing sentence meaning, which means they do not take into account the different levels of importance of different sentence parts when applying BERT embeddings to downstream tasks. To address this issue, [1] proposed the Sentence part-enhanced BERT (SpeBERT) model to incorporate sentence parts into BERT embeddings by merging BERT's embeddings with sentence part embeddings. The researchers found that by incorporating sentence parts appropriate for the respective downstream task, SpeBERT was able to outperform the BERT baseline on both sentiment classification and semantic textual similarity.

The goal of this project was to implement the default project's multitask minBERT model and improve its performance on sentiment analysis, paraphrase detection, and semantic textual similarity by incorporating sentence parts into the minBERT model for the respective downstream tasks. We also experiment with gradient surgery, cosine similarity fine-tuning, and SMART optimization to reduce overfitting and further improve model performance. Additionally, while the developers of the SpeBERT model find that incorporating sentence parts improved performance on semantic textual similarity and sentiment classification, they do not apply the SpeBERT model to paraphrase detection. Thus, we will also experiment with incorporating both "main" and "other" sentence parts to the paraphrase detection task to see if incorporating sentence parts also improves BERT performance on paraphrase detection.

3 Related Work

The primary source of inspiration for this project comes from [1] They proposed the SpeBERT model to enhance BERT embeddings by merging BERT's CLS embedding with sentence part embeddings. The sentence part embeddings are extracted through a pooling operation based on BERT's contextual embeddings and sentence part masks. The sentence part masks are obtained from the Stanford dependency parser, which encodes sentence parts with different values based on the downstream task. The final sentence representation is obtained by using an "aggregator" that merges the BERT embedding and sentence part embedding. The authors experiment with various "aggregators", including averaging the embeddings, concatenating the embeddings, and taking the weighted average between the embeddings. The final enhanced sentence representation is then used for prediction in downstream tasks. In the end, the authors find that when the correct sentence parts are incorporated ("main" parts like subject, verb, and predicate for text similarity and "other" parts like adverbial and complement for sentiment analysis), the SpeBERT model consistently outperforms the baseline BERT model on text similarity and sentiment analysis tasks. By implementing the SpeBERT model, we hope to affirm the author's findings on the effectiveness of incorporating sentence parts for sentiment classification and semantic text similarity. We also hope to discover which sentence parts are more important for paraphrase detection, since SpeBERT has yet to be applied to this task.

Another extension implemented in this project was gradient surgery to improve model performance when training the minBERT model simultaneously on multiple tasks. This is a method developed by [2]. Since the directions of the gradients for different tasks could come in conflict with one another during multitask training, the researchers propose a gradient surgery method that projects the gradient of a task onto the normal plane of the gradient of each of the other tasks with a conflicting gradient. The authors observed substantial performance gains after applying gradient surgery to both multitask supervised learning and reinforcement learning problems.

SMART optimization was also implemented in this project in hopes of reducing overfitting. SMART optimization is a method developed by [3]. Fine-tuning large, pretrained natural language processing models can easily result in overfitting, especially on smaller datasets. Thus, to reduce overfitting while fine-tuning such models, the authors propose a principled regularized optimization framework containing two key components: smoothness-inducing regularization and bregman proximal point optimization. Through the SMART optimization framework, the authors successfully reduced the overfitting of the various large, pretrained NLP models when fine-tuning them for downstream tasks.

4 Approach

4.1 Baseline Model

For the baseline model, the BERT embeddings from our minBERT model are passed into one linear layer that is shared between all the downstream tasks. Then, for the sentiment analysis task, a second linear layer is applied that reshapes the output into a 5-dimensional vector, and softmax activation is applied to the output. For the paraphrase detection task, the shared linear layer outputs from each of the two input sentences were concatenated into a combined feature vector, and this feature vector was passed into a linear layer with sigmoid activation to generate the ultimate output. The same process was used for semantic textual similarity, minus the sigmoid activation. We fine-tuned this multitask model by applying the gradient computation to the sum of the losses on each of the three tasks.

4.2 Gradient Surgery

Following the implementation of the baseline model, we implemented gradient surgery in hopes of improving the multitask training process. We followed the gradient surgery implementation from [2], which involves projecting the gradient of a task onto the normal plane of the gradient of each of the other tasks with a conflicting gradient. Specifically, if the gradient of task i conflicts with the gradient of task j, then the gradient of task i, g_i, is projected onto the normal plane of g_j:

$$g_i = g_i - \frac{g_i \cdot g_j}{||g_j||^2} \cdot g_j \tag{1}$$

To implement gradient surgery, we adopted the code from [4] into our model training pipeline.

4.3 Cosine Similarity Fine-Tuning

Since the goal of the semantic textual similarity task was to predict the degree of similarity between two input sentences, we experimented with the cosine similarity function for this task. In this experiment, we modified the model architecture by applying the cosine similarity function to the output of the shared linear layer instead of the second linear layer to generate the prediction for semantic textual similarity. Cosine similarity not only inherently incorporates similarity into the model output, but it also restricts model output within a certain range. Since the labels for the semantic textual similarity task also lie within a specific range (see 5.1.3), this may also help improve model performance.

4.4 Sentence Part-Enhanced minBERT

To incorporate sentence parts into BERT, BERT's contextual embeddings are merged with sentence part masks through a part pooling operation. Then, this pooled embedding is fused with BERT's original pooled embedding through an "aggregator" to generate the sentence part-enhanced BERT embeddings. For our Spe-minBERT models, we pass this enhanced representation into our baseline model architecture instead of the original BERT embeddings, and we also experiment with slight architecture modifications, as described in Section 5.4. The overall process of incorporating sentence parts into BERT embeddings is summarized in Figure 1.

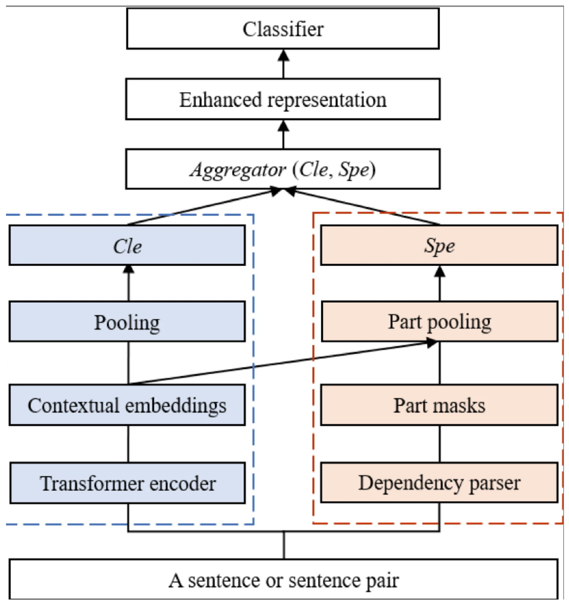

Figure 1. The general architecture of the SpeBERT model [1]

4.4.1 Sentence Part Masks

Sentence part masks are vectors with lengths equal to the number of words in a given sentence. If a word belongs to the sentence parts being focused on, then the corresponding value in the part mask is 1, and otherwise, it is 0. The two types of sentence parts we focus on in this project are "main" sentence parts (subject, predicate, and object), and "other" sentence parts (adverbials and complements).

To generate the sentence part masks, we used the Stanza library from the Stanford NLP Group [5] for dependency parsing as well as part-of-speech tagging. For "main" sentence parts, any word in the sentence that was tagged as a verb or noun by the part-of-speech tagger, as well as any word that was labeled as an "object" or "subject" by the dependency parser, was assigned a value of 1 in the sentence part mask. For "other" sentence parts, any word in the sentence that was tagged as an adjective or adverb or any word that was labeled a complement or adverbial was assigned a value of 1 in the part mask.

Following the findings of [1], all our experiments use main sentence parts for the semantic textual similarity task, and all experiments used other sentence parts for the sentiment classification task. For the paraphrase detection task, we experimented with both types of sentence parts to determine which sentence parts were most important for paraphrase detection.

4.4.2 Part Pooling

To combine BERT's contextual embeddings with the sentence part masks, we used the following pooling operation proposed by [1]:

$$Spe = \frac{\sum_i^n m_i \cdot h_i}{\sum m_i} \tag{2}$$

Here, for a sentence with n parts, m_i is the i-th element of the sentence part masks, and h_i is the BERT contextual embedding that corresponds to the i-th sentence part.

4.4.3 Aggregators

To aggregate BERT's original pooled embedding (the "Cle" embedding) with the "Spe" embedding, we experimented with three types of aggregators: mean, concatenation, and weighted average. For mean aggregation, the strategy was to simply take the average between the Cle and Spe embeddings. For concatenation, the Cle and Spe embeddings were concatenated into a single feature vector. For weighted average aggregation, we applied the same strategy as the SpeBERT authors: the weighted average between the Cle and Spe embeddings was taken using a learnable weight factor α. The value of α was learned through a neural network computation:

$$\alpha = \sigma(W^T Concat(Cle, Spe) + b) \tag{3}$$

where $Concat(Cle, Spe)$ denotes the feature vector formed by concatenating the Cle and Spe embeddings together. [1] find that weighted average aggregation produced the best results for sentiment classification, and concatenation produced the best results for semantic textual similarity.

4.5 SMART

To address overfitting, we implemented the SMART optimization framework from [3]. SMART optimization involves 1) smoothness-inducing regularization and 2) Bregman proximal point optimization. Smoothness-inducing regularization manages the complexity of large-scale NLP models while Bregman proximal point optimization is a class of trust-region methods that help reduce aggressive parameter updates that lead to overfitting. To implement SMART optimization, we adopted the code from [3] to our model training pipeline.

5 Experiments

5.1 Data

5.1.1 Sentiment Analysis

For the sentiment analysis task, we evaluate our model on the Stanford Sentiment Treebank dataset [6]. This dataset consists of 11,855 single sentences from movie reviews. Each sentence is given one of five labels: negative, somewhat negative, neutral, somewhat positive, or positive. For this task, we have been provided with the following splits: train (8,544 examples), dev (1,101 examples), and test (2,210 examples).

5.1.2 Paraphrase Detection

For the paraphrase detection task, we use the Quora dataset [7], which consists of 400,000 question pairs. Each sample of question pairs is assigned a label indicating whether the two questions are paraphrases of one another. The following splits are provided in this dataset: train (141,506 examples), dev (20,215 examples), and test (40,431 examples).

5.1.3 Semantic Textual Similarity

The SemEval STS Benchmark dataset [8] was used for the semantic textual similarity task. This dataset consists of 8,628 different sentence pairs, each assigned a value indicating the degree of similarity on a scale from 0 (unrelated) to 5 (equivalent meaning). For the STS dataset, we are given the following splits: train (6,041 examples), dev (864 examples), and test (1,726 examples).

5.2 Evaluation method

To evaluate model performance on the sentiment analysis task, as well as the paraphrase detection task, the accuracy of the model's predictions were observed. To evaluate performance on the semantic text similarity task, we used the Pearson correlation between the true similarity values and the predicted similarity values. The overall "score" of the model was computed by taking the average of the metrics from each task.

5.3 Experimental details

All models were trained with batch size 8, learning rate 10^-5, and for 25 epochs. The cross-entropy loss was used for sentiment classification, the binary cross-entropy loss was used for paraphrase detection, and the mean-squared error loss was used for semantic textual similarity. The AdamW optimizer was used to train the models. Each model was evaluated on the dev set after each epoch, and the model with the highest score on the dev set was saved. Because it was computationally expensive to generate sentence part masks, we only generated main and other sentence part masks for the first 6,040 samples of the train split for each dataset for the sake of efficiency. Main and other sentence part masks were generated for all dev set samples for all tasks. Thus, for all experiments, we performed multitask training on the same number of samples (6,040) from each task. As a result, a key assumption is the model that performs best when trained on 6,040 samples would also perform best if trained on all training samples from each task.

5.4 Models

For reference, we trained and evaluated the following models:

- **Baseline**: the baseline model trained by summing the losses from each task
- **Baseline-Surgery**: the baseline model trained with gradient surgery
- **Baseline-Surgery-Cosine**: the baseline model trained with gradient surgery and with cosine similarity applied on the STS task
- **Spe-Para-Main**: same as **Baseline-Surgery-Cosine** except incorporating main parts for STS, other parts for sentiment classification, and main parts for the paraphrase task, all with mean aggregation

- **Spe-Para-Other**: **Spe-Para-Main** except other parts for paraphrase detection
- **Spe-Para-None**: **Spe-Para-Main** except original BERT embeddings for paraphrase detection
- **Spe-Separate**: same as **Spe-Para-None**, except instead of first passing embeddings into a shared linear layer, the shared layer is replaced with a separate linear layer for each task
- **Spe-Orig-Agg**: same as **Spe-Separate**, except weighted mean aggregation for sentiment classification and concatenation for the STS task
- **Spe-Only**: same as **Spe-Separate**, except only the "Spe" embedding is used for sentiment classification and the STS task
- **Spe-SMART**: same as **Spe-Separate**, but training with the SMART optimization framework

5.5 Results

The performance of the models on the dev set is shown below:

Model	SST Dev Acc	Para Dev Acc	STS Dev Corr	Dev Score
Baseline	0.495	0.735	0.360	0.530
Baseline-Surgery	0.490	**0.750**	0.368	0.536
Baseline-Surgery-Cosine	0.433	0.748	0.765	0.649
Spe-Para-Main	0.470	0.708	0.811	0.663
Spe-Para-Other	0.466	0.708	0.808	0.661
Spe-Para-None	0.472	0.708	0.811	0.664
Spe-Separate	0.478	0.719	0.814	0.670
Spe-Orig-Agg	**0.497**	0.699	**0.819**	**0.672**
Spe-Only	0.475	0.709	0.814	0.666
Spe-SMART	0.488	0.707	0.816	0.670

Table 1: Dev Set Performance of the Models

From Table 1, we see that gradient surgery provided a slight boost in model performance, from a dev set score of 0.530 to a score of 0.536. This boost was small, however, and we expected a much larger boost based on results from the existing research. This could indicate that the gradients did not conflict often during training. While adding cosine similarity caused a dip in accuracy for the sentiment classification and paraphrase detection tasks, it created a significant boost in the STS correlation from 0.368 to 0.765, which led to a significant jump in the overall dev score from 0.536 to 0.649.

After incorporating other parts for the sentiment classification task, we observed a substantial boost in SST accuracy from 0.433 to as high as 0.497. Similarly, incorporating main parts for the semantic textual similarity task also provided a boost in STS correlation, from 0.765 to as high as 0.819. This was expected, as it has been shown in previous research that these sentence parts improve performance on these specific tasks. However, for the paraphrase detection task, incorporating neither main parts nor other parts provided any improvement in accuracy.

Removing the shared linear layer among all tasks and replacing it with a separate linear layer for each task resulted in slight improvements across all tasks. Additionally, switching from a mean aggregation strategy to a weighted average strategy for sentiment classification and concatenation strategy for STS also boosted performance on those two tasks from 0.478 to 0.497 and 0.814 to 0.819, respectively. While performance on the paraphrase detection task decreased, the boost in performance in the other two tasks was more than enough to offset this, resulting in a slight boost in the overall dev score.

Relying solely on the "Spe" embedding resulted in slightly poorer performance, which was expected since the "Spe" embedding alone likely lacked context on the overall input sentence. Incorporating SMART optimization provided improvements in SST and STS performance, but this was not enough to improve overall model performance. This was surprising due to SMART being proven in previous research to substantially reduce overfitting and boost performance. However, an explanation for this could be that we did not fine-tune the hyperparameters in the SMART optimization algorithm, and this may have been especially important given that we are training on multiple tasks.

5.6 Leaderboard Results

After the best-performing model, **Spe-Orig-Agg**, was identified, we generated part masks for all the remaining train samples for the sentiment classification task, as well as for the test samples for sentiment classification and semantic textual similarity. Then, we trained the **Spe-Orig-Agg** on all available train data over 2 epochs. A single epoch was defined in terms of the paraphrase training split, and the training loop reset to the beginning of the SST and STS training splits when the end was reached. Additionally, a concern was that multitask training was holding back model performance, so we trained three separate **Spe-Orig-Agg** models, one on each task, and ensembled these three models together. The models' predictions were averaged together to produce ultimate predictions.

Model	Dev SST	Dev Para	Dev STS	Test SST	Test Para	Test STS
Spe-Orig-Agg	0.470	0.758	0.834	0.488	0.760	0.810
Ensemble	0.529	0.780	0.837	0.524	0.783	0.809

Table 2: Leaderboard Results on Dev and Test Sets

As expected, training **Spe-Orig-Agg** on all available data produced an increase in overall dev score (0.672 to 0.687) and training three separate **Spe-Orig-Agg** models and ensembling them produced an even better dev score of 0.715. Results were slightly worse for test set, but this was expected since we selected the best-performing model on the dev set.

6 Analysis

Incorporating other and main parts respectively played a key role in boosting performance for the sentiment classification and STS tasks. This shows us that adverbials and complements are key sentence parts to focus on when dealing with sentence classification tasks, and subject, verb, and predicate and the parts that should be focused on for semantic textual similarity analysis. However, for paraphrase detection, the results show that focusing on specific sentence parts does not improve performance. Interestingly, incorporating main or other parts does not hinder performance either. This seems to indicate that paraphrase detection may require some combination of main sentence parts and other sentence parts (e.g. subject and adjective or verb and adverb). Another explanation is that we may not have selected the correct aggregation strategy.

Aggregation strategy played a role in improving model performance. As seen in Table 1, choosing the weighted average aggregation strategy produced a notable boost in sentiment classification accuracy, and the concatenation strategy produced a slight bump in STS correlation. The weighted average may be a more effective strategy for the sentiment classification task because we incorporate "other" sentence parts into this task. Because other sentence parts occur less frequently in sentences, this could cause the sentence masks to be more sparse, so a weighted average strategy with a learnable parameter could help the model learn to deal with this sparsity. On the other hand, we incorporate main sentence parts for the STS task, and since main sentence parts occur more frequently, concatenation may produce the best results because there could be a substantial amount of important information in the Spe embedding, so concatenating the embeddings ensures that this raw information can be passed into the model.

A key theme in the results was that multitask training may have been hindering performance due to the nature of the extensions we applied. For instance, after incorporating cosine similarity into the model, we saw a drop in performance on both the sentiment classification and paraphrase detection tasks. Since multitask training was used, this indicates that cosine similarity may have caused the model to focus more heavily on the STS task, thus taking away from learning on the other two tasks.

Similarly, we can see in Table 1 that incorporating other parts into the sentiment classification task and main parts into the STS task led to substantially worse results on the paraphrase detection task. Like with cosine similarity, incorporating these sentence parts could have caused the model to more easily learn the sentiment classification and STS tasks, thus hindering performance on the paraphrase task.

Moreover, after replacing the shared linear layer with separate linear layers for each task, we saw an increase in performance across the board. A possible explanation for this is because we incorporated different sentence parts (or no parts for paraphrase) into the BERT embeddings for each task, sharing a layer between these tasks was forcing the model to attempt to generalize among all three inherently different embedding types. Having a separate layer for each task allows the model to learn parameters specific to each type of embedding.

As a final piece of evidence, we saw that the Ensemble model ultimately led to the best results, indicating that for the sake of achieving robust results on each of the tasks, it may not be the most beneficial to train a multitask model for these three tasks.

7 Conclusion

From our experiments, we saw that incorporating sentence parts into BERT embeddings plays a key role in creating robust embeddings for sentiment classification and semantic textual similarity. We find that incorporating main parts into semantic textual similarity and other parts into sentiment classification provides substantial performance improvements relative to our baselines. This affirms the findings of [1] and demonstrates that focusing on specific sentence parts is important for creating better representations of language. However, with the paraphrase detection task, we failed to improve performance by incorporating main or other sentence parts. A possible explanation for this is that paraphrase detection may require some combination of main sentence parts and other sentence parts, or we may not have chosen the correct aggregation strategy.

We also confirm that selecting the right aggregation strategy is important when blending the Spe embedding with the original BERT embedding. Like the SpeBERT authors, we find that a weighted average strategy works well when using other parts, and a concatenation strategy works well when using main parts. More research should be done on various aggregation strategies to improve SpeBERT performance.

A key finding is that combining multitask learning with SpeBERT may not be an effective strategy. Since different downstream tasks depend on different types of sentence part-enhanced embeddings, sharing layers between tasks may hinder a Spe model's ability to focus on specific aspects of a sentence part-enhanced embedding. Thus, when developing SpeBERT-like models, it may be beneficial to avoid multitask learning if the tasks require different sentence parts.

We also find that incorporating cosine similarity to the model output significantly improves model performance on semantic textual similarity. This demonstrates the importance of incorporating formulas that compute similarity into neural networks for similarity-based tasks.

Some limitations of this project include time and resource constraints. We were not able to produce sentence part masks for all 141,506 training examples for the paraphrase detection task. If we had done so, we may have seen different results from our Spe models, as adding more training examples may have allowed our models to extract more important information from the sentence part-enhanced embeddings. We also only used mean aggregation for paraphrase detection and did not experiment with different aggregation strategies for the sentence parts for this task. Since selecting the correct aggregation strategy is important, we may have found different results had we experimented with different aggregation strategies.

In terms of future work, it would be interesting to see if a larger dataset of sentence part-masks for paraphrase detection would produce different results. We would also like to experiment with different aggregation strategies for paraphrase detection, as well as potentially develop new aggregation strategies for sentence part-enhanced embeddings.

References

[1] Chaoming Liu, Wenhao Zhu, Xiaoyu Zhang, and Qiuhong Zhai. Sentence part-enhanced BERT with respect to downstream tasks. In *Complex and Intelligent Systems*, 2022.

[2] Tianhe Yu, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. Gradient surgery for multi-task learning. *CoRR*, abs/2001.06782, 2020.

[3] Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Tuo Zhao. Smart: Robust and efficient fine-tuning for pre-trained natural language models through principled regularized optimization. In *Association for Computational Linguistics (ACL)*, 2020.

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

[5] Peng Qi, Yuhao Zhang, Yuhui Zhang, Jason Bolton, and Christopher D. Manning. Stanza: A Python natural language processing toolkit for many human languages. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, 2020.

[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] Shankar Iyer, Nikhil Dandekar, and Kornél Csernai. First quora dataset release: Question pairs.

[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.