Extending Phrasal Paraphrase Classification Techniques to Non-Semantic NLP Tasks

Stanford CS224N Default Project

Samy Cherfaoui
Department of Computer Science
Stanford University
scherfao@stanford.edu

Nikhil Sharma
Department of Computer Science
Stanford University
sharmnik@stanford.edu

Abstract

In (Arase and Tsujii, 2021), the authors present a novel method of using cheap, automated techniques to identify phrasal paraphrases during fine-tuning to improve the performance of semantic sentence pair tasks (like paraphrase detection and STS). These phrasal paraphrases match up corresponding phrases between different sentences depending on whether they semantically mean the same thing. The model was able to outperform other popular models with fewer parameters and a smaller training corpus. Our goal is to see whether we can augment the phrasal techniques in the research paper cited above to perform well on tasks that aren't just semantic and also to boost performance on semantic tasks. In particular, we pre-process our sentiment analysis dataset into the form of a semantic sentence pair task and we apply many common extensions such as cosine similarity fine-tuning, gradient accumulation, and multiple negative ranking loss to improve the performance of semantic tasks. We note that our model achieves impressive performances on all three tasks, achieving a 23% increase in accuracy over the baseline for semantic analysis while maintaining competitive performances on paraphrase detection and STS.

1 Key Information to include

Josh Singh is our project mentor. Nikhil worked on adapting the research paper's state dictionary to load with our baseline models and non-sentiment targeted improvements including AMP/Grad Accumulation, Cosine Similarity, and Round Robin Scheduling. Samy worked on creating a new dataset based off SST to set up our sentiment-targetted experiments, loading the new dataset in our experiments, and updating our evaluation functions to work with the new dataset. Additionally, he worked on adapting Multiple Negative Ranking Loss for our codebase.

2 Introduction

Much work has been done in recent years regarding how to scale up language models in such a way that allows for performant finetuning on a wide variety of tasks, especially as these models start to hit a bound on hardware and algorithmic constraints given large numbers of parameters. One particularly promising approach is the use of cheap, automated techniques bolted on top of models that are able to extract further linguistic or semantic properties from the underlying data without increasing the model size. In particular, Arase and Tsuji presented a novel method of find phrasal paraphrases within sentence pairs using an automated dynamic programming algorithm that does not require additional weights (Arase and Tsujii, 2021). Phrasal paraphrases are lexical phrases that share semantic equivalence but differ in syntactical choice (see Figure 8 in the Appendix for an example). The paper demonstrates that a simplified feature generation method for phrasal paraphrases significantly improves BERT's performance across semantic sentence pair modeling

tasks such as STS and paraphrase detection. These phrasal alignments are transformed through a series of very simple pooling and matching operations (see Figure 7 in the Appendix). Finally, the training objective of the fine-tuned model is a combination of the sentence paraphrase training loss as well as phrase-level paraphrase training loss (refer to Figure 1 for the pseudocode). The results were extremely impressive: the team was able to beat many popular models that used more parameters and more elaborate features on the GLUE benchmark which focuses on semantic sentence pair tasks (Wang et al., 2019).

However, one area that this model is unable to do well on is on single-sentence tasks that do not rely on semantic understanding. Our paper seeks to determine whether we can transform this model approach that performs extremely well on semantic sentence pair tasks to something that is task-agnostic. In particular, we focus on improving performance in sentiment analysis which takes in a movie review and outputs a sentiment score from zero to four. Our idea was to transform this task into a semantic sentence pair task. By augmenting every data example with a rough summary of the review (e.g. "We had an absolute blast watching Batman." and "My opinion of this movie is positive"), we can take advantage of the phrasal paraphrase classifier to identify phrases within the review that correspond to sentiment and achieve performances on this task that are similar to the performances on the semantic sentence pair tasks. We also sought to make use of several common multitask fine-tuning extensions such as cosine similarity fine-tuning (Reimers and Gurevych, 2019), multiple negatives ranking loss (Henderson et al., 2017), and gradient accumulation (Yu et al., 2020) to boost the performance of the semantic sentence pair tasks as well.

Input: Paraphrase sentence pairs $P = \{\langle s, t\rangle\}$, a pre-trained BERT model
Obtain a set of phrase alignments A as pairs of spans for each $\langle s, t\rangle \in P$
WordPiece tokenisation of P
Accommodate phrase spans in A to BERT's token indexing: $A = \{\langle (j, k), (m, n)\rangle\}$
repeat
  for all mini-batch $b_t \in \{\langle P_i, A_i\rangle\}$ **do**
    Encode b_t by the BERT model
    Phrase feature generation
    Compute loss: L(Θ)
      For phrasal paraphrase task: $L_p(\Theta)$
      For sentential paraphrase task: $L_s(\Theta)$
      $L(\Theta) = L_p(\Theta) + L_s(\Theta)$
    Compute gradient: $\nabla(\Theta)$
    Update the model parameters
until converge

Figure 1: Algorithm pseudocode (Arase and Tsujii, 2021)

3 Related Work

The centerpoint of our research is (Arase and Tsujii, 2021). However, many other papers have overlapping work done with our approaches. (Liu et al., 2019) motivates the concept of multi-task learning in deep learning while Crawshaw (2020) provides a useful survey of multi-task learning frameworks which informed some of our design decisions. As far as we are aware, Arase and Tsuji are the first to use a phrasal paraphrase classifier for transfer finetuning on BERT; however, similar approaches have been considered in the past such as (Phang et al., 2019) who used transfer fine-tuning for sentence-level paraphrase classifiers and the more recent work by Peng et al. (2022) who use sentence encoders to determine phrase alignments instead of the dynamic programming approach. We utilize Arase and Tsuji's dynamic programming approach from (Arase and Tsujii, 2017); however, we could have also used (Zhang et al., 2021) which utilizes a neural phrase alignment algorithm, (Neubig et al., 2011) which utilizes a similar automated approach to Arase and Tsuji but is much less efficient, or (Zhang and Vogel, 2005) which allows for longer phrases but achieves significantly worse precision and recall than either of the prior methods. In order to boost our performance, we made liberal use of many of the extensions proposed in the handout such as (Yu et al., 2020) for gradient accumulation, (Henderson et al., 2017) for multiple negatives ranking loss, and (Reimers and Gurevych, 2019) for cosine-similarity finetuning.

We believe that our approach of using preprocessing of the sentiment analysis dataset is novel. (Yin et al., 2020) is the most similar paper to ours: it uses binary constituency parse trees to model sentiment as semantic relationships during BERT finetuning and achieves competitive results. Another similar paper is (Ma et al., 2018) which injects sentiment information during finetuning but on LSTMs.

4 Approach

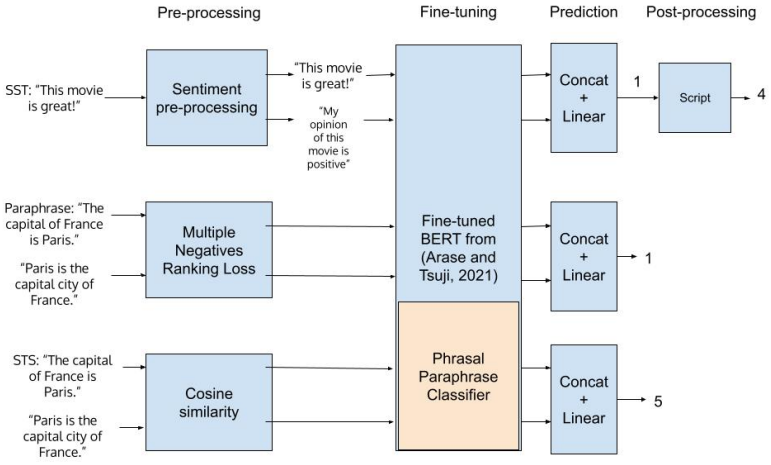

Figure 2: We present a rough diagram of our model architecture here. It is worth noting that for sentiment analysis, we are actually generating 5 pairs of new dataset examples per original datapoint and that the post-processing script then iterates over these 5 outputs and takes the sentiment class corresponding to the highest one. It is also worth noting that AMP and gradient accumulation are built into the pipeline on select experimental runs for computational efficiency.

4.1 Baselines

We used two baselines to evaluate the performance of our experiments. We referred to the first baseline as **NaiveMultitaskBERT** as it was a relatively naive continuation of default code provided in which we added minimal processing for each downstream task. For sentiment classification, we introduce a dropout layer followed by a linear layer to map the pooled BERT output to five sentiment classes. For both paraphrase detection and semantic text similarity, we concatenate the pooled outputs of each sentence's BERT representation, passing the result through a linear layer to produce a single logit.

Our second baseline model, which we named **TransferFTMultitaskBERT**, explicitly loaded the model parameters from Arase et. al. This was the key distinguishing feature between NaiveMultitaskBERT and TransferFTMultitaskBERT as after loading the model parameters, the rest of the architecture was the same. We did encounter a significant issue in loading the research model parameters, as the research paper was published in 2019 and uses an older version of the transformers library to load bert-base-uncased. Therefore, we added **original code** in the form of a Python script which parsed through the state_dict of the research model and refactors it to match the newer HuggingFace's bert-base-uncased model. As part of this refactor, we chose to allow selective state dictionary loading only the matching parts of the state dict. This gave us better freedom to control how strictly we adhere to the research model as a starting point, which allowed for more experimentation.

4.2 Improving Our Model: A Two-Pronged Strategy

Our strategy for improving our model from the baselines split into two distinct approaches: improving overall accuracy without bias towards any specific downstream task and improving sentiment analysis accuracy specifically. The first approach was more intuitive, and there were many extensions provided through the project handout we could build upon. The second approach was premised on our hypothesis that loading in the research weights and leveraging the TransferFTMultitaskBERT model

as a foundation would inherently yield average accuracy for semantic tasks (paraphrase detection and semantic textual similarity) without any modifications, therefore, we could target improving sentiment analysis accuracy. We chose to pursue both methodologies separately at first, which would allow us to see the advantages and disadvantages of each, and then combine them with the aim it would create our best performing model.

4.3 Non Sentiment-targeted Improvements

Enhancing Computational Efficiency: Our first goal was to enhance the computational efficiency of the fine-tuning process, particularly regarding the Quora dataset. To facilitate this, we employed a combination of Automatic Mixed Precision (AMP) and Gradient accumulation to refine the training methodology. AMP enables mixed precision arithmetic, predominantly leveraging float16 precision for non-vital computations as opposed to the conventional float32. In our implementation of AMP, we utilized the autocast() function from the torch.cuda.map library to create a re-usable context manager passed into our downstream tasks batch processing functions.

Our implementation of gradient accumulation was more involved than AMP. It started with us adding **new code** in the form of command line arguments and a function to **dynamically calculate each task's batch size**. For each task, our code calculates how many accumulation steps are needed based on the desired overall batch size (`args.batch_size`) and the maximum batch size allowed for each task (`args.max_sst_batch_size, args.max_para_batch_size, args.max_sts_batch_size`). This is done by dividing the overall batch size by the maximum batch size for each task, which determines how many mini-batches the overall batch should be split into for processing.After the per-task accumulation step size is calculated, we added a **original function** that runs at the end of each batch training that would check if the number of steps modulo the step size is 0, and only then we would perform an optimizer and scaler step. In summary, these adjustments not only improved the training iteration velocity but also diminished the memory footprint, all while maintaining the integrity of the model's accuracy.

Cosine Similarity: We found the idea to use cosine similarity as a metric for the semantic textual similarity downstream task from (Reimers and Gurevych, 2019). Cosine similarity measures the cosine of the angle between two non-zero vectors of an inner product space, here representing sentence embeddings, with a value close to 1 indicating high similarity and a value close to -1 indicating high dissimilarity. To implement this, we started by creating a new Dataset representation called ModifiedSentencePairDatasetLabelsScale. This dataset was created using the SentencePairDataset class from the default code, and the labels were modified to to scale from [1, 5] to [0, 1]. We also experimented with scaling the labels to [-1, 1], but the empirical results were better for [0, 1]. We modified our `predict_similarity` code with an original implementation, where we calculate the cosine similarity between the embeddings of the two inputs. We experimented with two approaches for using the BERT embeddings with cosine similarity as described below.

- Using BERT CLS token: The [CLS] token is specifically designed and pre-trained to aggregate the entire sequence's context, making it potentially well-suited for classification tasks. However, for some tasks, especially those requiring deep understanding of the text beyond classification, the [CLS] token might not capture all nuances and details present across the sentence.
- Using average of BERT embeddings: Using the average of the embeddings rather than the CLS can provide a richer representation of the sentence by considering the contribution of each token. This might capture a broader range of semantic signals present in the text. However, this using the average of the phrase embeddings is not without tradeoffs- averaging embeddings requires additional computation compared to directly using the [CLS] token.

Multiple Negative Ranking Loss: We used (Henderson et al., 2017) to implement multiple negative ranking loss specifically for the Quora dataset. Per the default project doc, training data consists of sets of K sentence pairs $[(a_1, b_1), ..., (a_n, b_n)]$ where (a_i, b_i) are labeled as similar sentences and all (a_i, b_j) where $i \neq j$ are not similar sentences. The loss function then attempts to minimize the distances between the embeddings for (a_i, b_i) while maximizing it for (a_i, b_j) where $i \neq j$. To this end, we constructed a modified dataset so our training data could take this form and we also implemented a Multiple Negative Ranking Loss algorithm based on (Henderson et al., 2017). We

chose to focus exclusively on Quora since we noted empirically that performance appears to drop when MNRL is used on the STS dataset.

Round Robin Scheduling: Our final modification for non-targeted improvements was changing the training architecture from batch-level sequential (as implemented in the baselines) to batch-level round-robin. Our reasoning was that in the batch-level sequential architecture, each task is trained independently in series, one after the other, for a given number of epochs. Each task completes its iteration over its entire dataset before moving on to the next task, and this leads to imbalances in attention and resource allocation, especially given the non-uniform training dataset sizes. Changing to a batch-level round-robin architecture ensures that no single task monopolizes the training process, allowing for equitable resource allocation across all the training datasets. Please see Figures 5 and 6 in the Appendix for a visual representation of the change.

To implement this change, we first **created a new class called RoundRobinScheduler** which had its own methods for retrieving the next batch of data for the appropriate task. It then called into the batch processing functions we also implemented. We established a predefined order of tasks `(["sst", "paraphrase", "sst", "sts", "sst"]`, indicating the sequence in which batches from different dataloaders (each corresponding to a different task) are processed. We also **added a command-line argument** `(-num_batches_per_epoch_rr)` which takes in a value representing the number of batches per epoch for the round-robin scheduler with the default value being the sum of the lengths of all the dataloaders.

4.4 Sentiment-targeted Improvements

For our sentiment-targeted improvements, we had to modify sentiment analysis into a task that was conducive to our model. This meant transforming our sentiment dataset into a semantic sentence pair dataset. To accomplish this, we decided to transform our dataset as follows. The previous format of our dataset was `<review, sentiment score>`. We made a new dataset class that duplicates this datapoint five times with one of the following five labels each: `"My opinion of this movie is negative."`, `"My opinion of this movie is somewhat negative"`, `"My opinion of this movie is neutral."`, `"My opinion of this movie is somewhat positive."`, `"My opinion of this movie is positive."`. Instead of associating an output with the sentiment score, we then convert this into a semantic task by setting the output to 1 if the given opinion corresponds to the sentiment and 0 otherwise. Note how this dataset is extremely similar to the paraphrase dataset. We then wrote a simple predict function that works in the same way as the other sentence pair tasks described above: concatenate the pooled outputs of the BERT representation of the sentences and pass it through a Linear layer. In order to get this to evaluate and test correctly, we had to postprocess the outputs of the model. Since the model now returns an unnormalized logit across five duplicated datapoints, we wrote a script to assign a sentiment class to a specific phrase that corresponded to the highest valued logit in the outputs.

5 Experiments

5.1 Data

We used the following datasets provided from the starter code to train and evaluate our experiments:

Stanford Sentiment Treebank: This dataset consists of 11,855 single sentences extracted from movie reviews and contains 215,154 unique phrases from those parse trees, annotated by 3 human judges. Each phrase comes with an associated label ranging from negative, somewhat negative, neutral, somewhat positive, and positive. The data were split as: 8,544 examples for training, 1,101 examples for evaluation, and 2,210 examples for testing. In order to train this on our sentiment-targeted improvements, we modified the dataset into a semantic sentence pair dataset. Please refer to Section 4.4 for further details.

Quora Dataset: This dataset consists of 400,000 (of which we were given a subset) labeled question pairs indicating whether instances are paraphrases of one another. The data were split as: 141,506 examples for training, 20,215 examples for evaluation, and 40,431 examples for testing. From the beginning of our work, we knew that this dataset would give us a unique challenge due to its size relative to the other two datasets, and we adjusted our approach by including AMP and gradient

accumulation to account for this. We also note that we created a custom dataset for the purposes of multiple negative ranking loss. This is detailed in Section 4.3.

SemEval STS Benchmark dataset: This dataset consists of 8,628 different sentence pairs of varying similarity on a scale from 0 (unrelated) to 5 (equivalent meaning). The data were split as: 6,041 examples for training, 864 examples for evaluation, and 1,726 examples for testing.

5.2 Evaluation method

For both the SST and Quora datasets, our accuracy was calculated by comparing our predicted label with the ground truth. For the SemEval STS Benchmark, our accuracy is calculated using the Pearson correlation of the true similarity values against the predicted similarity values.

5.3 Experimental details

Unless explicitly stated, all experiments share the following parameters: 10 epochs per dataset, pretrain learning rate of `1e-3`, finetune learning rate of `1e-5`, and hidden-layer dropout probability of 0.3. Except for the specific instances where we used other types of embeddings, we used average of BERT embeddings as our input for cosine similarity. When we were training on GCP, we set our general batch size to 256. When using gradient accumulation, the max batch size for SST, paraphrase, and STS were 32, 32, and 64 respectively. When using Round Robin scheduler, our default value for `num_batches_per_epoch_rr` was 638.

We trained on a mixture of GCP and Google Colab instances with A100 and T4 GPUs depending on availability. The training times for the sentiment-targeted extensions took between 10 and 12 hours due to the 5x training dataset blowup whereas the training times for the non-sentiment targeted extensions took between 3.5 and 4 hours.

5.4 Results

We report our results for our various experiments below. Recall that research weights corresponds to loading in the weights provided by (Arase and Tsujii, 2021). These are hosted by (Zenodo, 2019).

Table 1: Dev model evaluation on sentiment analysis, paraphrase detection, and STS tasks. We include two baselines: one against a naive finetuning approach on minBERT and one against the baseline provided in (Arase and Tsujii, 2021) for BERT-base. RW stands for research weights, GA stands for gradient accumulation, COS-CLS stands for cosine similarity using CLS embeddings, COS-AVG stands for cosine similarity using average of phrase embeddings, SP stands for sentiment preprocessing, MNRL stands for multiple negative ranking loss, and RR stands for round-robin scheduling.

Model	SST Acc. ↑	Para Acc. ↑	STS Corr. ↑	Overall Acc.
Pretrain NaiveMultitaskBERT	0.396	0.380	0.019	0.374
Fine-tune NaiveMultitaskBERT	0.523	0.604	0.097	0.536
Fine-tune TransferFTMultitaskBERT	0.523	0.649	0.184	0.554
Fine-tune (RW/AMP/GA)	0.470	0.736	0.339	0.625
Fine-tune (RW/AMP/GA/COS-CLS)	0.442	0.771	0.445	0.645
Fine-tune (RW/AMP/GA/COS-AVG)	0.470	0.775	0.766	0.709
Fine-tune (RW/AMP/GA/MNRL/COS-CLS)	0.481	0.752	0.581	0.675
Fine-tune (RW/AMP/GA/MNRL/COS-AVG)	0.476	0.721	0.766	0.693
Fine-tune (RW/AMP/GA/RR/COS-CLS)	0.483	0.745	0.750	0.701
Fine-tune (RW/AMP/GA/RR/COS-AVG)	0.506	0.734	**0.829**	0.718
Fine-tune (RW/SP)	0.747	0.747	0.554	0.765
Fine-tune (RW/SP/MNRL)	**0.767**	0.800	0.605	0.787
Fine-tune (RW/SP/MNRL/COS-AVG)	0.764	**0.801**	0.709	0.808

We are allotted three test submissions so we used the three models that performed best in at least one category. Note that since we did not have access to the test labels, we were unable to establish a baseline on test performance.

Table 2: Test model evaluation on sentiment analysis, paraphrase detection, and STS tasks. We included the three models from above with a bolded (best) entry in any task.

Model	Sentiment Acc. ↑	Paraphrase Acc. ↑	STS Corr. ↑	Overall Acc.
Our model (RW/AMP/GA/COS-AVG/RR)	0.518	0.735	0.799	0.718
Our model (RW/SP/MNRL)	0.641	0.744	0.713	0.747
Our model (RW/SP/MNRL/COS-AVG)	0.615	0.774	0.753	0.755

Our results shocked us - our hypothesis that converting sentiment analysis into a task that is conducive to the (Arase and Tsujii, 2021) model was confirmed. As of the time of this writing, our RW/SP/MNRL model has the highest sentiment analysis accuracy. We believe that this is because the phrasal paraphrase classifier forces the model to identify certain phrase structures and wordings that align with a specific emotion. Whereas other models likely use some pooled output passed into further layers, our approach allows the model to dive within the sentence and pick out specific sub-phrases or words that correspond to a sentiment. We were disappointed that the research weights did not perform as well on semantic tasks as some of the other submissions on the leaderboard. One hypothesis is that this may be because of the issue mentioned in Section 4.1 where the model was finetuned under an older version of BERT. We had to apply our best judgment in transferring over weights since there wasn't a 1-to-1 mapping. This may have led to inefficiencies or differing precisions on weights. However, we were able to beef up this performance as expected through the use of MNRL on the paraphrase dataset and cosine similarity on the STS dataset and achieve fairly impressive performances.

6 Analysis

Our hypothesis before starting any experiments was that sentiment analysis would be the worst performing task of our model. Our intuition behind this was that our model with research weights loaded would be biased towards semantic-based tasks and would therefore struggle with sentiment-based tasks. Our results show that non-sentiment targeted improvements had worse accuracy on the sentiment dataset than just the NaiveMultitaskBERT baseline model, as expected. However, the RoundRobinScheduler did perform the best out of all our general improvement experiments. We hypothesize this is due to the frequent context switching that occurs during the round robin scheduling as well as trying to balance out the larger Quora dataset.

SST: Our sentiment-targeted improvement experiments performed much better on the sentiment dataset, even surpassing our expectations at the beginning. At the time of writing this paper, our model with sentiment pre-processing has done the best in the class in the sentiment classification accuracy on the dev dataset. However, there are certain drawbacks with modeling sentiment as a semantic task. In a movie review, words are not always meant to be taken literally. For example, consider the review "In its best moments , resembles a bad high school production of Grease , without benefit of song". Although the review contains the word "best", the review is overall negative – with a true label of 0. However, our model predicted this review as a 4, or the strongest positive review. We believe this is due to modeling sentiment as a semantic task, where "its best moments" is matched up with "positive". Please refer to Figure 9 in the appendix for a more detailed breakdown of predicted and true labels.

Paraphrase: Our paraphrase accuracy was fairly good but upon inspecting the outputs, we noticed that there were many more false positives than false negatives (see Figure 3). In the runs with multiple negative rankings loss, this could be explained by the fact that we provided many positive examples but few negative ones. We also noticed that the model also struggled with paraphrases of questions. "What", "why", and "how" were within the top 10 words by frequency in the dataset sorted by incorrect predictions. One possible explanation for this is that phrasal structures within sentences change within questions and it is possible that (Arase and Tsujii, 2021) was fine-tuned on statements rather than questions and the unlikely lingusitic and semantic structure confused our model.

STS: We also performed well on STS correlation but we noticed that our model predicted a higher volume of labels in the 0 to 3 range rather than the true labels, indicating that our model was erring on the conservative side and assuming that sentences were more unrelated than they actually are (see Figure 4). We theorize that this could be due to the implementation of cosine similarity for this model.

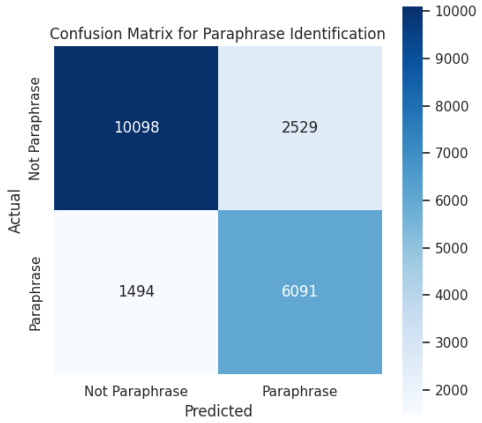

Figure 3: Confusion Matrix for Quora dataset

We used cosine similarity on average of BERT embeddings since it achieved better performance on dev runs but certain words could be heavy outliers and skew the average embedding. To reduce this skew, we could consider using the CLS token embedding to obtain a more holistic embedding of the sentence.

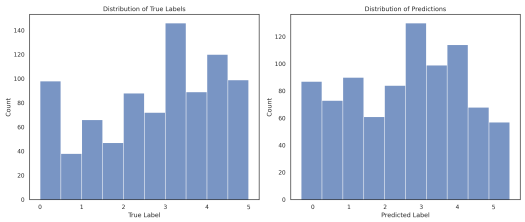

Figure 4: Distributions of predictions vs true labels in STS

7 Conclusion

We showed that by using clever pre-processing techniques and a wide variety of performance extensions such as cosine-similarity fine-tuning, we were able to take advantage of the impressive results presented by (Arase and Tsujii, 2021) which bolts on an automated phrasal paraphrase classifier during finetuning to improve performance on semantic sentence pair tasks and extend those results to another unrelated task. We observed that sentiment analysis can be modeled as a semantic sentence pair task and if we were successful in doing so, we could apply a one-size-fits-all phrasal paraphrase classifier to BERT during fine-tuning for all tasks. Despite the fact we weren't able to replicate the state-of-the-art results on the semantic tasks, our results still shocked us - we achieved incredible performances on sentiment accuracy in both dev and test. Further, by applying a wide variety of extensions that targeted our semantic tasks, we were able to develop a model that performed quite admirably on all three tasks with no major weaknesses. Further work would involve exploring re-training (Arase and Tsujii, 2021)'s model on the newer HuggingFace transformers. Arase and Tsujii actually mentioned that they believed their model would work on the newer versions of the `transformers` libraries but they hadn't verified it yet; we were given the chance to do so and were unable to replicate their spectacular results. With a fresh set of weights for the right libraries, we may be able to get a quick and easy performance boost with no extensions required.

References

Yuki Arase and Jun'ichi Tsujii. 2021. Transfer fine-tuning of bert with phrasal paraphrases. *Computer Speech & Language*, 66:1–14.

Yuki Arase and Jun'ichi Tsujii. 2017. Monolingual phrase alignment on parse forests. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1–11.

Michael Crawshaw. 2020. Multi-task learning with deep neural networks: A survey.

Matthew Henderson, Rami Al-Rfou, Brian Strope, Yun-Hsuan Sung, László Lukács, Ruiqi Guo, Sanjiv Kumar, Balint Miklos, and Ray Kurzweil. 2017. Efficient natural language response suggestion for smart reply. *arXiv preprint arXiv:1705.00652*.

Xiaodong Liu, Pengcheng He, Weizhu Chen, and Jianfeng Gao. 2019. Multi-task deep neural networks for natural language understanding. *arXiv preprint arXiv:1901.11504*.

Yukun Ma, Haiyun Peng, Tahir Khan, Erik Cambria, and Amir Hussain. 2018. Sentic lstm: a hybrid network for targeted aspect-based sentiment analysis. *Cognitive Computation*, 10:639–650.

Graham Neubig, Taro Watanabe, Eiichiro Sumita, Shinsuke Mori, and Tatsuya Kawahara. 2011. An unsupervised model for joint phrase alignment and extraction. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 632–641.

Qiwei Peng, David Weir, and Julie Weeds. 2022. Towards structure-aware paraphrase identification with phrase alignment using sentence encoders. *arXiv preprint arXiv:2210.05302*.

Jason Phang, Thibault Févry, and Samuel R. Bowman. 2019. Sentence encoders on stilts: Supplementary training on intermediate labeled-data tasks.

Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084*.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2019. Glue: A multi-task benchmark and analysis platform for natural language understanding.

Da Yin, Tao Meng, and Kai-Wei Chang. 2020. Sentibert: A transferable transformer-based architecture for compositional sentiment semantics.

Tianhe Yu, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. 2020. Gradient surgery for multi-task learning. *Advances in Neural Information Processing Systems*, 33:5824–5836.

Zenodo. 2019. Transfer fine-tuned bert models by paraphrases.

Jiacheng Zhang, Huanbo Luan, Maosong Sun, Feifei Zhai, Jingfang Xu, and Yang Liu. 2021. Neural machine translation with explicit phrase alignment. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 29:1001–1010.

Ying Zhang and Stephan Vogel. 2005. An efficient phrase-to-phrase alignment model for arbitrarily long phrase and large corpora. In *Proceedings of the 10th EAMT Conference: Practical applications of machine translation*.

A Appendix (optional)

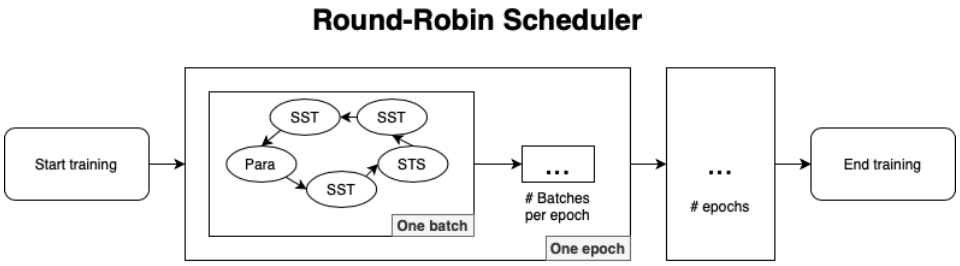

Figure 5: Diagram of the Round-Robin Scheduler architecture. The object of note here is that one batch now consists of a wide variety of data from SST, Para, and STS rather than processing one task at a time.

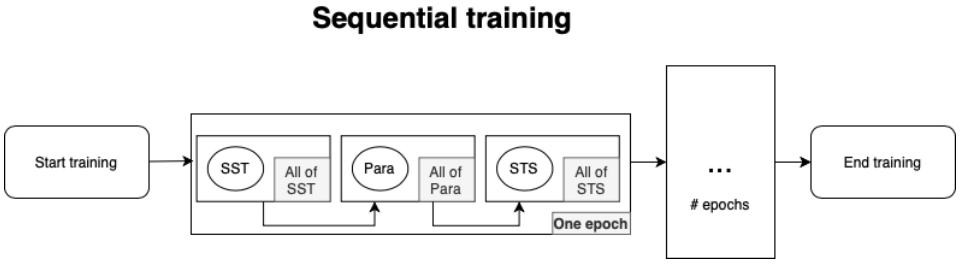

Figure 6: Diagram of the Sequential training architecture as implemented in the baseline models.

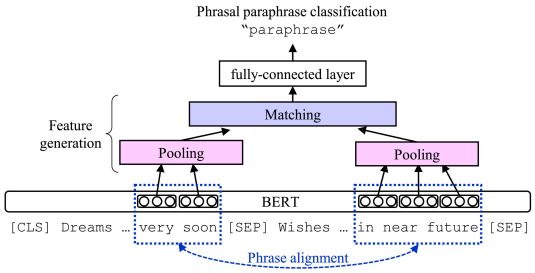

Figure 7: Phrasal paraphrase classification model diagram (Arase and Tsujii, 2021)

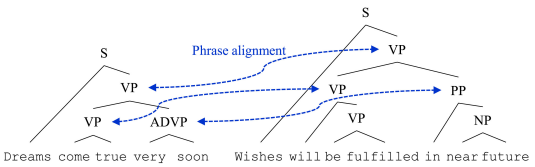

Figure 8: Examples of phrasal paraphrases using the method in (Arase and Tsujii, 2017)

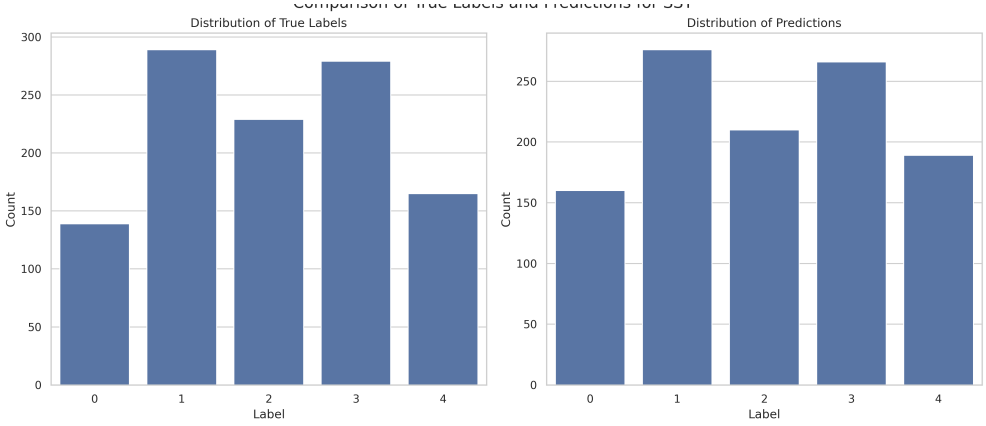

Figure 9: Distributions of predictions vs true labels in SST