UmBERTo: Enhancing Performance in NLP Tasks through Model Expansion, SMARTLoss, and Ensemble Techniques

Stanford CS224N Default Project

May Levin
mayl@stanford.edu
Department of Computer Science

Julián Rodríguez Cárdenas
julianrc@stanford.edu
Department of Symbolic Systems

Abstract

This paper describes enhancement methods for optimizing performance with a minBert model on three distinct NLP tasks: sentiment analysis on the Stanford Sentiment Treebank, paraphrase detection on Quora Question Pairs, and semantic textual similarity assessment. Through iterative experimentation, we identified that unfreezing model weights, extending training to 20 epochs, employing a triple-layer linear-ReLU architecture, and incorporating SMARTLoss significantly enhanced model efficacy. Moreover, our ensemble of ensembles approach, leveraging a combination of models based on majority voting and performance metrics, emerged as an effective approach, achieving a test accuracy of 0.678.

1 Key Information:

Mentor: Tim Dai. External Collaborators: None. Sharing Project: No.

2 Introduction

With the introduction of the Transformer in Vaswani et al. (2017), the field of natural language processing witnessed a momentous shift in how models handle and process data. Now, corporations and individuals are leveraging this technology for their own uses, from automatizing tasks such as estimating sentiment expressed through text, and entrusting the task of summarizing or drafting documents to the computer. Our research contributes to this segment of work.

Here we present a minBert model with our particular extensions, fine-tuned for three tasks: sentiment analysis, paraphrase detection, and similarity evaluation. There are many incentives for achieving human-level computer performance on these tasks. Websites like Quora and Reddit benefit from identifying questions that are duplicates of one another, and in the academic world, uncovering plagiarism is an important and pressing problem. Search engines such as Google and Bing must measure the similarity between pieces of information as part of their data processing methods. Websites such as movie review aggregators to social-media platforms enforcing speech guidelines need an automated way of identifying the sentiment expressed by users.

We explored several ways of adapting a pre-trained minBert model to perform well on these three downstream tasks. Our main focuses were on the expansion of the architecture to have task-specific heads, the addition of extra datasets and epochs, and usage of SMARTLoss to explore different types of learning. Lastly, we combined the highest scoring versions of our various model into a "ensemble of ensembles" model to achieve our highest accuracy.

3 Related Work

There is extensive prior work on sentiment, similarity, and paraphrase analysis in the context of Natural Language Processing (NLP). For decades, researchers have explored ways of applying machine learning methods to these tasks. For example, Pang et al. (2002) showed that artificial neural networks can outperform human benchmarks in determining whether reviews are positive or negative. Since the introduction of the transformer in Vaswani et al. (2017) researchers have been able to push the performance on these tasks (see Naseem et al. (2020)). Similarly for textual similarity analysis, there is a long tradition of applying different categorization and analysis methods. However, nowadays it is the transformer that dominates on leader boards for this task, such as in Jiang et al. (2019). In terms of paraphrase detection, as noted in Zhou et al. (2022), many approaches have been tried for identifying pairs of sentences with the same content, ranging from convolutional neural networks, to transformer-based architectures. We note that here too, the highest performances on this task are dominated by transformer architectures (PapersWithCode).

As such, we take a pre-trained minBert implementation, and explore how we can further improve performance on the three tasks, by using a significantly smaller amount of data and training time compared to production-scale models. Considering how resource-intensive production and utilization of Large Language Models are, it is important to investigate the possibility of well-performing models of lower size and compute power. In this area Jiao et al. (2019) has shown promise with their model named TinyBERT. This is a language model based on the regular BERT model that is 13% its size while achieving 97% of its performance. TinyBERT was trained by transferring knowledge from base BERT by mapping layers from one model to the other, and minimizing the difference in activation's between these mapped layers based on some training dataset. One can think of TinyBERT as a cleverly compressed version of base BERT. Importantly for this project, TinyBERT is proof that much smaller models can still be almost as good as some of the larger counter-parts. Yet, we must note that this still requires the regular BERT model, which is trained on a large corpus of data. Similarly, in this project we utilize a pre-trained minimal version of BERT.

Lastly, a significant portion of our project involves deciding how to fine-tune our model, how to incorporate more data, selecting hyperparameters, etc. This too is an active area of research. For instance, Dodge et al. (2020) found that initialization and data order can have a significant impact on performance, but also that there are signs one can observe to decide to stop training a model early. This is also relevant in this work, since we did not only train multiple models at the same time, but also had to decide factors such as data utilization and hyperparameter selection.

4 Approach

As a whole, our approach to improve performance on the three downstream tasks was to implement well-established small improvements to our minBert model separately for several different hyperparameters, take the best of these, and then combine them to a "best" version model. Given several best similar performing models, we utilize an ensemble approach. We used the Carina GPU cluster, training many smaller models and find the best hyperparameters for each one. These models were run on Nvidia's Tesla V100. Below, we detail each methodological approach individually.

4.1 Benchmark

Our benchmark was a slightly more complex version of the part 1: minBert implementation model. Here, the three layer head was fine-tuned on all three datasets given for each task. We recorded train and dev accuracies for this run at every epoch (default of 10), so we could further compare it to other models for specificity. We combined the data from each of the three tasks utilizing PyTorch Lightning's Combined DataLoader class which combines tensors and packages them into labeled batches inside a "megabatch". Since each subbatch is tagged with its corresponding task name, we can then individually train them on their designated tasks while still allowing us to interleave different data points within the model. This method enhances the model's exposure to varied data, potentially improving its ability to generalize across tasks by experiencing a richer mix of inputs during training. We calculate our benchmark loss in the following manner, with $sts\_weight = .4$,
SST_loss = $\mathrm{F}_{cross\_entropy}$, $Para\_loss = F_{binary\_cross\_entropy\_with\_logits}$
$STS\_loss = F_{mse\_loss}$

The general loss is calculated as follows:

$$Loss = SST\_loss + Para\_loss + (sts\_weight \times STS\_loss) \quad (1)$$

4.2 Expanded architecture for multi-task classifier

The base model adds a single dropout and subsequent linear layer to the last layer of the base model, for each of the three tasks. This translates into each task being solved by doing a linear classification on the last layer of the base model. This is problematic as relying solely on the final layers reduces the representation capacity of the model, and does not allow for more complex or non-linear relationships to be learned.

Therefore, our first approach was to enlarge the architecture of each head for the downstream tasks. We added different configurations of dropout, linear, and ReLU activation functions in order to increase level of expressivity. More so, we experiment with dropout probability, utilizing the default .3, but then attempting to increase it so that we are able to improve the model robustness. By increasing the dropout rate, we aimed to encourage the model to learn more independent features that are directly relevant to the desired output, thereby reducing its dependency on the co-occurrence of these features, as well as reduce overfitting Iandola et al. (2020).

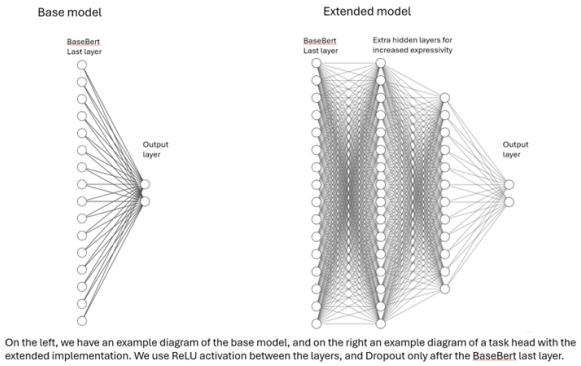

Figure 1: Diagram visualizing the minimal model implementation on the left, and our extended implementation on the right.

4.3 Addition of more training data

There is a notable relationship between more training data and better model performance Zhu et al. (2016). With that in mind, we experimented with extending the original dataset discussed below in detail, which includes the Quora paraphrase dataset, the Stanford Sentiment Treebank, and the SemEval STS Benchmark Dataset. We further included two more datasets taken from Hugging Face repositories. The first one is a Sentence Compression dataset, which includes pairs of sentences, one of which is a summary of the other Filippova (2016). We used this dataset to further train on the paraphrase task. The second extra dataset we included is the "DynaSent: Dynamic Sentiment Analysis Dataset", curated by a Linguistics Professor Christopher Potts, which is a dataset of yelp reviews with ratings 1-5 Potts et al. (2020). These were appended to their corresponding dataset tasks. Since each task had corresponding datasets of different sizes, we also had to decide how to train once the smallest dataset was entirely utilized. We explored two primary options using the combined dataloader: halting the training process as soon as the smallest dataset had been entirely consumed, or continuing training until the largest dataset was completely used - recycling batches from the datasets that had already been fully encountered.

4.4 Addition of more epochs

Seeing as the benchmark accuracy rates were still increasing significantly at the last epochs (out of 10), this made us believe that the model still had room to improve. Therefore, we increased epoch time and tested the model for 10,20,40 epochs by changing the parameter value.

4.5 SMARTLoss

SMARTLoss (SMoothnessinducing Adversarial Regularization and BRegman pRoximal poinT opTimization) is used as an addition to the regular loss, as a regularization and robustness-increasing tool. More formally, it attempts to minimize $R_s(\theta) = \frac{1}{n}\sum_{i=1}^{n} max_{||\tilde{x_i}-x_i||_p \leq \epsilon} l_s(f(\tilde{x_i};\theta), f(x_i,\theta))$.

It works by firstly, making the model smooth in its activation space, which supports overfitting. More so, it increases robustness by minimizing the difference between the model output to given input, with the output to a slightly perturbed version of that input (that is the $\tilde{(x)}$ in the equation). For SMARTLoss implementation we used a mix of our own code and an implementation of the algorithm shared by the authors (SmartPytorch).

4.6 Freezing/Unfreezing layers

Initially, training was conducted with frozen weights, utilizing the minBert model that had been previously trained. However, as our more advanced models started to plateau in accuracy, we experimented with unfreezing layers (pretraining) by toggling a specific flag, enabling the update of weights during training. By unfreezing the layers, we aimed to better fit the model to our tasks, allowing the model to adjust and learn from the specifics of our dataset.

4.7 Contrastive learning

We also attempted to utilize contrastive learning, as described in Gunel et al. (2020), as a method for improving performance during fine-tuning. Contrastive learning works by incorporating an extra training objective during training, which tries to maximize the distance between outputs for elements from different classes (thereby capturing the contrasting features between elements of different classes). Importantly, due to the fact that contrastive learning ultimately aims at pushing the outputs of a model further apart for different inputs, which can cause competition with the downstream task used for training, we abandoned this extension of the model (for this attempt, we utilized the author's immplementation). Contrastive learning is more appropriate as a pre-training task; due to our data limitations, we also did not pursue this avenue.

4.8 Combining models

We aimed to combine the model parameters from the best results from each of these above steps, with the expectation that this would further enhance accuracy by reinforcing various aspects of the model. Previous literature has supported the idea that integrating the strengths of different models can lead to a more robust and accurate overall system Yu et al. (2023).

4.9 Ensemble model: a model composed of many models

Lastly, given the variety of models at our disposal, we wanted to explore how we could best combine these to lead to improved results. We built two initial ensemble models. The first ensemble model simply chose the best-performing model for each task separately and then took all their results. The second ensemble model implemented a majority voting system, where similar high ranking models had their outputs combined, and each answer was selected by majority. Building on these strategies, a third ensemble model was developed, which used the approach that had the highest dev accuracy score per task, i.e. if ensemble B has a higher STS dev score then ensemble A, ensemble C choose that one.

5 Experiments

5.1 Data

We utilize three datasets for fine-tuning the model's capabilities in sentiment analysis, paraphrase detection, and semantic similarity assessment: the Stanford Sentiment Treebank (SST), Quora Question Pairs, and the SemEval Semantic Textual Similarity (STS) Benchmark. The SST dataset comprises of 11,855 single-sentence reviews, each categorically labeled as negative, somewhat negative, neutral, somewhat positive, or positive. The Quora dataset contains 400,000 question

pairs, each with a binary label indicating whether one question paraphrases the other. The STS dataset consists of 8,628 sentence pairs, each assigned a continuous similarity score ranging from 0 (unrelated) to 5 (equivalent meaning).

5.2 Evaluation method

We use pre-defined accuracy (SST, Quora) or Pearson score (SemEval) for the tasks on seperate dev and train datasets.

5.3 Experimental details

We ran several experiments changing parameters for each approach. Below is each table with bolded values representing the maximum accuracy model for that task.

5.4 Results

	Benchmark	1x Linear & ReLU	3x Linear & ReLU	3x Linear, ReLU, & Dropout	3x Linear & ReLU, Dropout = .5
SST dev	.376	.410	**0.416**	0.378	.401
Paraphrase dev	.643	.682	**0.688**	0.655	.683
STS dev	.262	.318	**0.334**	0.278	.308

Table 1: Initial Model Performance Comparison for varying architectures

We see that the architecture of 3x linear layers each followed by a ReLU function has the best results, together with a dropout rate of .3 (the benchmark value). The additional layers provide depth to better capture the model's complexity while ReLU introduces necessary non-linearity to capture more complex relationship which show to help improve accuracy results. However, the additional dropout layers as well as the high dropout values likely lead to significant information loss. A lower dropout rate strikes a better balance by reducing overfitting without excessively losing valuable information, especially considering these are smaller datasets.

	Benchmark, Epoch = 10	Epoch = 20	Epoch = 40
SST dev	.376	**.402**	. 4010
Paraphrase dev	.643	.694	**.695**
STS dev	.262	.357	**.363**

Table 2: Effect of additional Epochs

Epoch iteration value of 40 is technically the best, but marginally, and this increased the computation time quite significantly. Therefore, we decided to stick with epoch 20 for following experiments. This phenomenon is further discussed below with figure 2.

	Benchmark	Appending Data
SST dev	**.376**	.201
Paraphrase dev	**.643**	.401
STS dev	**.262**	.262

Table 3: Effect of Appending More Data

Opposite to what we expected, training on more data lead to worse performance on multiple occasions. We believe this was due to imbalance in datasets (as in having many more positive training examples than negative ones for the paragraph task), and also the different sizes of the training sets. For this reason, in future iterations of our models, we abstained from using more data.

	Benchmark, SMARTLoss = 0	SMARTLoss = .02	SMARTLoss = 2	SMARTLoss = 10
SST dev	.376	**0.409**	0.374	0.377
Paraphrase dev	.643	**0.699**	.635	0.625
STS dev	.262	**.360**	0.262	0.277

Table 4: SMARTLoss Threshold Variation

SMARTLoss helped fine-tune our model by preventing overfitting, since that is an issue we regularly encountered during training. Furthermore, SMARTLoss might have made the model overall better at processing language, since we would not want minimal random changes to the embedding inputs to result in significant changes to the behavior of the model. The most successful SMARTLoss for the base had a value of .02.
We proceed to form our COMBO model, that has 20 epochs, 3x Linear and ReLU architecture, with a benchmark dropout, no training data addition, and unfrozen weights. These choices were made based on the experiments described above. We still experimented with several SMARTLoss values to see how this impacts the results. More so, we also tested changing how much we weight the STS loss in our loss function (default = .4) and a separate experiment trying a "mini cycle" data version that does not repeat data for each task in order to try to increase the accuracy of the lesser-performing tasks.

Configuration	SST dev	Paraphrase dev	STS dev
Unfrozen Combo Model, SMARTLoss = .02	.501	**0.823**	0.340
Unfrozen Combo Model, SMARTLoss = .2	0.506	0.757	0.370
Unfrozen Combo Model, SMARTLoss = 10	0.502	0.735	**0.394**
Unfrozen Combo Model, weighing STS loss = (.8)	0.502	0.818	0.359
Unfrozen Combo Model, data on min_cycle version	**.521**	0.761	0.338

Table 5: Advanced Model Configurations and Their Performance

The advanced models varied in their performance across different tasks, as evident from table 6. The use of unfrozen weights in the models provided flexibility, allowing the models to adjust more freely to the nuances of each dataset. This approach generally led to better learning, as it allows the model to refine its pre-learned weights for the specific tasks at hand. The Paraphrase dev achieved the highest accuracy, likely due to its larger dataset size, which provided more training examples for the model to learn from. To address this, several strategies were explored, such as weighting the loss functions of the smaller datasets (like STS) more heavily, or limiting the repetitions of the larger dataset (using the mini_cycle version). Adjusting the dataset weighting aimed to equalize the learning opportunities across tasks, while limiting data repetitions in the mini_cycle version sought to enhance performance on smaller datasets by preventing overfitting. The exploration of "SMARTLoss" with varying degrees, from 0.02 to 10, was an attempt to fine-tune the model's sensitivity to errors in different configurations. Since we had a mix of successful models, with different configurations excelling at different tasks, this suggests that the models might benefit from having larger heads (more parameters) or more flexibility to learn separate task-specific representations. Therefore, an ensemble model was built to best take advantage of the multiple models in our arsenal.

	SST dev acc.	Paraphrase dev acc.	STS dev correlation	Overall dev
Ensemble 1	0.521	**0.823**	0.370	.677
Ensemble 2	**0.535**	0.795	**0.384**	.674
Ensemble 3	**0.535**	**0.823**	**0.384**	**.683**

Table 6: Ensemble Model Performance

Our final leader board scores utilizing ensemble 3, was: **SST test accuracy: 0.528, Paraphrase test accuracy: 0.825, STS test correlation: 0.360. Overall test score: 0.678.**

6 Analysis

6.1 Discussion of effects of extensions on performance

In conducting a detailed analysis of our benchmark model and one of the more effective models (the unfrozen combo model with SMARTLoss = 0.02, as presented in row 1 of Table 5), several interesting characteristics emerged. In Figure 2 right side we see a significant initial drop in loss for the combo model during the first few epochs, which slowly continues to decrease, approaching a loss of zero. That explains why we saw small gains when training models for more than 20 epochs. We believe that at this point, the model has maximized it's generalizability for the given task, and is mostly just memorizing the training set in a non-general way. Compared to the basic benchmark model, the combo model has a much lower training loss, but the increase in performance is not proportional, indicating overfitting.

The figure on the left, showcases the general accuracy between the models, which was computed as the sum of all training task accuracies with the STS score normalized. The benchmark model's accuracy plateaus relatively quickly, indicating early convergence but also suggesting a potential limitation in adapting to more complex patterns. However, the train and dev fits for the benchmark model are very aligned, implying a balanced model performance. The combo model shows a very different narrative, with more signs of overfitting to the training data, evidenced by the large disparity between training and development accuracies. Despite this, the model retains sufficient generalization to perform well on the development set. This contradicting behavior might might be enabled by the SMARTLoss, which can prevent overfitting by smoothing out the model. These aspects help the model remain robust in predictions despite overfitting.

A related surprising observation is that the SMARTLoss weight that was optimal in one version of the model across all tasks was also the worst at one of the tasks for another version of the model (see Table 4, having the best accuracy with SMARTLoss weight = 0.02, whereas in Table 5, we see a SMARTLoss weight = 0.02 having almost the worst performance on STS, compared to the other weights). Given that the 0.02 resulted in considerable improvements to the model in Table 4, we consider that to be a reasonable weight, and therefore consider 10 be a very aggressive weight for SMARTLoss. As we can see in Figure 2, during the training of the Combo model (whose results are in Table 5) we quickly get to a point, after just two epochs, where the dev accuracy has plateaued, but the train accuracy continues to rise slowly; this is a clear indication of overfitting. Since SMARTLoss is also a smoothness-inducing loss, we hypothesize that this disallows the model to overfit extremely. In other words, by forcing the model to work in a smooth space, it is harder for it to completely memorize the training dataset. This effect is even more significant when the SMARTLoss weight is more aggressive. However, when we are not at risk of overfitting, then it makes sense that we observe a smaller SMARTLoss weight to be beneficial. This is consistent with the fact that the STS training set is by far the smaller of the 3 tasks, which makes the model prone to overfitting on that task.

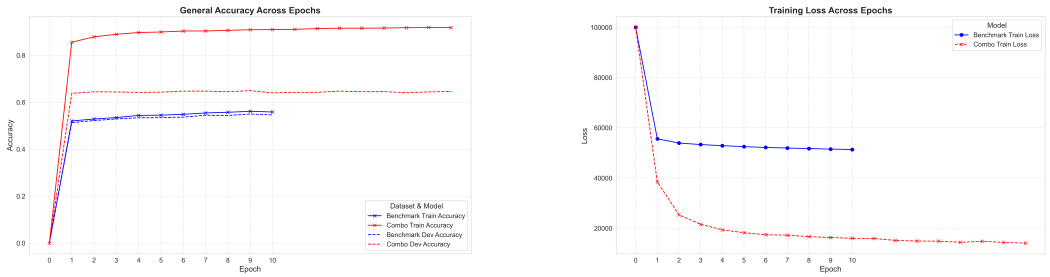

Figure 2: Left: General Accuracy Comparison between benchmark model and best model. Right: Train loss Comparison between benchmark model and best model.

6.2 Discussion of the tasks as a whole

An important matter to consider is the different demands that each of the three tasks places on the model. As we see it, all the tasks require a strong notion of semantics to be successfully answered. At the same time, none of the tasks have a strong emphasis on syntax or form. For example, two

paraphrases can have an entirely different form (possibly sharing no words in the inputs) but still share the same meaning. At the same time, two sentences may be highly similar in their structure and vocabulary, while having only few word changes. The model must be able to pick up on subtle differences to really understand the meaning of a sentence.

The Semantic Textual Similarity task also requires the model to not just understand a sentence, but also be able to have a gradient of meaning between two sentences. This is in some way similar to the paraphrase task; if two sentences are exactly similar (i.e. they mean the same thing) then they are also paraphrases. Thus, one can make the argument that the similarity and paraphrase task might benefit from a shared processing.

The Sentiment Analysis task is the one that is the most dissimilar to the other two. Importantly, the model still needs some level of understanding of semantics and sentence structure, but this task is only concerned with a small subset of semantics: that of the subjective judgment expressed in an opinion. We believe that this task places much different demands on the model, as it must have some conceptual grasp of what things are considered good or bad. For example, an example of a positive review is as follows. "Light, silly, photographed with color and depth, and rather a good time." How does a model learn that "photographed with color and depth" are good things?

7 Conclusion

Our project explored various strategies to enhance performance on the three downstream tasks through an iterative approach. We discovered that the optimal configuration involves unfrozen weights, a training duration of 20 epochs, a triple-layer linear and ReLU architecture, and the addition of SMARTLoss. Furthermore, incorporating an ensemble model, which selects the highest value from two other ensemble models (majority vote ensemble model or best-performing model per task) proved to be exceptionally effective.

The primary limitation of our work was the reliance on a smaller dataset and a lack of extensive hyperparameter exploration. The sequence of our experiments could have been more thought out, for example, we started with architectural adjustments before optimizing the number of epoch which might have been more logical to do reversed as a more complex architecture might not exhibit performance plateaus as early, thereby benefiting more from extended training. Additionally, a more detailed examination of training loss and accuracy across all models and epochs could have led us to pay closer attention to potential overfitting issues earlier on.

There are several avenues that we can continue exploring to improve our model. Firstly, we could improve our method of training the model on larger amounts of data, by not adding more data to the last layer model training, but instead incorporating it to a prior contrastive learning task or before the three-head classifier splits. We also can create/find more robust infrastructure to perform a hyperparameter search, or use some algorithms to tune our hyperparameters. In our project, we have many parameters to adjust (e.g. the weight of the SMARTLoss and the contrastive loss, the number of epochs, training schedule, etc.), which we only adjusted one at a time, and in a quasi-random way. We could leverage libraries such as Ray Tune (https://docs.ray.io/en/latest/tune/index.html) to optimize our hyper parameters. Another strategy could be used to further explore weight freezing strategies, as mentioned in Dodge et al. (2020), which has the model train many layers, stop most, and fully-train some more.

In conclusion, our research demonstrates that strategic modifications to the minBert architecture, particularly utilizing model enlargement, SMARTLoss, and ensembling, can significantly enhance performance across diverse NLP tasks.

8 Team Contributions

- May contributed to setting up the training infrastructure, adding the combined dataloader, tuning model parameters, co-writing the paper, and contributing to the part 1 model focusing on its basic structure and configuration.
- Julian was responsible for implementing SMARTLoss, exploring contrastive learning approaches, co-writing the paper, and contributing to the development of the part 1 model, specifically integrating the AdamW optimizer.

References

Jesse Dodge, Gabriel Ilharco, Roy Schwartz, Ali Farhadi, Hannaneh Hajishirzi, and Noah Smith. 2020. Fine-tuning pretrained language models: Weight initializations, data orders, and early stopping. *arXiv preprint arXiv:2002.06305*.

Katja Filippova. 2016. Sentence compression dataset.

Beliz Gunel, Jingfei Du, Alexis Conneau, and Ves Stoyanov. 2020. Supervised contrastive learning for pre-trained language model fine-tuning. *arXiv preprint arXiv:2011.01403*.

Forrest N Iandola, Albert E Shaw, Ravi Krishna, and Kurt W Keutzer. 2020. Squeezebert: What can computer vision teach nlp about efficient neural networks? *arXiv preprint arXiv:2006.11316*.

Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Tuo Zhao. 2019. Smart: Robust and efficient fine-tuning for pre-trained natural language models through principled regularized optimization. *arXiv preprint arXiv:1911.03437*.

Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. 2019. Tinybert: Distilling bert for natural language understanding. *arXiv preprint arXiv:1909.10351*.

Usman Naseem, Imran Razzak, Katarzyna Musial, and Muhammad Imran. 2020. Transformer based deep intelligent contextual embedding for twitter sentiment analysis. *Future Generation Computer Systems*, 113:58–69.

Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan. 2002. Thumbs up? sentiment classification using machine learning techniques. *arXiv preprint cs/0205070*.

PapersWithCode. Papers with code paraphrase leaderboard.

Christopher Potts, Zhengxuan Wu, Atticus Geiger, and Douwe Kiela. 2020. DynaSent: A dynamic benchmark for sentiment analysis. *arXiv preprint arXiv:2012.15349*.

SmartPytorch.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Le Yu, Bowen Yu, Haiyang Yu, Fei Huang, and Yongbin Li. 2023. Language models are super mario: Absorbing abilities from homologous models as a free lunch. *arXiv preprint arXiv:2311.03099*.

Chao Zhou, Cheng Qiu, and Daniel E Acuna. 2022. Paraphrase identification with deep learning: A review of datasets and methods. *arXiv preprint arXiv:2212.06933*.

Xiangxin Zhu, Carl Vondrick, Charless C Fowlkes, and Deva Ramanan. 2016. Do we need more training data? *International Journal of Computer Vision*, 119(1):76–92.