Multitask Learning with Pre-trained BERT

Might resubmit before the late deadline

Stanford CS224N Default Project

Yuntao Ma
Department of Electrical Engineering
Stanford University
yma42@stanford.edu

Jack Albright
Department of Computer Science
Stanford University
jacalbr@stanford.edu

Kevin Li
Department of Computer Science
Stanford University
kevinli8@stanford.edu

Abstract

Our goal for this project was to leverage powerful pre-trained BERT contextual embeddings to simultaneously perform three sentence-level tasks: sentiment analysis, paraphrase detection, and semantic textual similarity. We first implemented core components of a BERT model, and achieved similar results to baseline on sentiment analysis and paraphrase detection. We then improved our multitask model by using various pooling methods, SMART regularization, and a cross-encoder architecture. Our best model improve baseline by 0.025, 0.181, and 0.042 for sentiment analysis, paraphrase detection, and semantic textual similarity, respectively. As time of writing, we rank number 1 by overall score on both dev and test set leaderboard.

1 Introduction

In recent years, deep learning techniques have revolutionized the field of natural language processing (NLP), enabling significant improvements in a wide range of NLP tasks. Among the various deep learning models developed for NLP, the Bidirectional Encoder Representations from Transformers (BERT) model has gained significant attention due to its remarkable performance across various NLP tasks. One unique attribute of BERT is its versatility. After being pre-trained on an initial large dataset, a BERT model can then be fine-tuned using additional data specific to an NLP task. Recently, researchers have also experimented with specializing BERT by modifying the model architecture. Both of these methods of specialization have resulted BERT models achieving state-of-the-art results for many NLP tasks.

For our project, we focus on three specific tasks: sentiment analysis, paraphrase detection, and semantic textual similarity. Sentiment analysis involves classifying the sentiment of a given text on a scale from negative to positive. Paraphrase detection involves determining whether two sentences convey the same meaning or not. And semantic textual similarity involves measuring the degree to which two sentences are semantically equivalent. We aim to leverage pooling, regularization, and encoding architectures to develop a BERT model capable of fulfilling all of these tasks simultaneously.

2 Related Work

Core to our project is the BERT model, a transformer model that is designed to understand the context of words in a sentence by leveraging bidirectional training. [1] After a pre-training step, Devlin et al. fine-tuned the BERT model for 11 different NLP tasks. The fine-tuned BERT model met and often exceeded the state-of-the-art for these tasks.

In 2019, Reimers and Gurevych developed Sentence-BERT [2] — a modification to the original BERT model that allowed it to predict semantic similarity much more efficiently. While some of the tasks covered in the original Devlin et al. paper took in a single sentence input, like sentiment analysis, other tasks like semantic similarity required comparing pairs of sentences. Devlin et al. used a Cross-Encoder architecture which combines sentence pairs into a single input for BERT. But Reimers and Gurevych used a Bi-Encoder architecture, which passes each sentence in the pair separately through BERT. The BERT model would then generate individual sentence embeddings which were used to predict semantic similarity through cosine-similarity. The authors also applied different pooling methods to reduce potential information loss in the original BERT model. Sentence-BERT outperformed many existing methods on several benchmark datasets for sentence similarity and classification tasks.

Later that year, Munikar et al. [3] devised a method for fine-grained sentiment classification of movie reviews based on the BERT model. While the original BERT paper focused only on binary sentiment classification, the work of Munikar et al. demonstrated that such a model could generate accurate results on a 5-label scale.

3 Approach

3.1 Transformer Encoder

Our transformer encoder is a BERT-Base model based on the work of Devlin et al. At the beginning of this project, we were provided with starter code for a minimal implementation of the BERT model. After completing the core components of the minBERT model, we verified our implementation of the BERT encoder by training it to perform sentiment analysis on movie review data. For this base sentiment analysis model, we simply used a dropout layer and a linear layer as our classification head to compute class probability using the input's `[CLS]` contextual embedding. We compared our results with the Dev accuracy baselines provided in the default project handout, confirming that our implementation was working properly.

We then extended minBERT to accept token type id as input and used it to produce token type embeddings. This extension enabled us to use a cross encoder architecture (see more in detail in Section 3.2) for the sentence pair tasks. This BERT-Base encoder was used to produce contextual sentence embeddings for all of our downstream tasks.

3.2 Bi-Encoder and Cross-Encoder

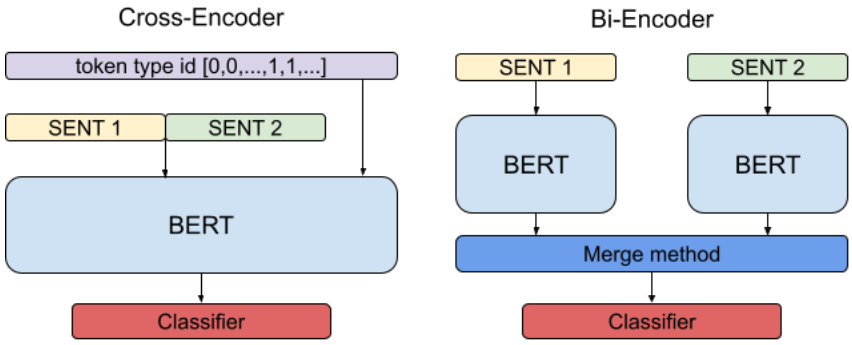

Figure 1: Architecture illustration for Cross-Encoder and Bi-Encoder for sentence pair taask

Paraphrase detection and semantic textual similarity are sentence pair tasks, which are NLP tasks that take two pieces of text as inputs. This presents a unique problem for transformer-based architectures, as vanilla transformers are only designed to handle single sentences. To handle multi-sentence input, we explored two strategies: an early fusion (Cross-Encoder[1]) and a late fusion (Bi-Encoder[2]) strategy. These strategies are illustrated in Figure 1.

In the Bi-Encoder architecture, we pass sentences 1 and 2 independently to BERT and produce their corresponding sentence embeddings. Then, in order to perform the classification or regression task,

we first merge the embeddings using a merge method. This method can be a concatenation, a sum, a difference, or a distance matrix. We can treat this merged embedding as a representation of the sentence pair, and use it for our tasks. The Bi-Encoder was used in the Sentence-Bert[2] paper for sentence pair tasks.

In contrast, the Cross-Encoder architecture passes both sentences into BERT simultaneously by concatenating them into a single sequence of tokens. We then provide a token type id to the BERT model to indicate which sentence each input token belongs to. During the token embedding process, we use the token type id to create a token type embedding and add it to the token embedding. Instead of producing individual sentence embeddings, the Cross-Encoder architecture produces direct representations of the sentence pairs. We can then directly apply our classifier on the output hidden embedding for our downstream task. The Cross-Encoder was used in original BERT [1] paper for sentence pair tasks.

3.3 Sequence Pooling

When the BERT encoder outputs a sequence of hidden embeddings, we add the `[CLS]` token before each sentence token sequence to represent the entire input sequence, which allows BERT to perform tasks such as classification and sentiment analysis. By using the `[CLS]` token, BERT can capture important information from the entire input sequence and use it to make predictions. While using the `[CLS]` token as a representation of the input sequence has proven to be effective in various NLP tasks [4], throwing away the other output sequences from BERT can also result in valuable information loss. The output sequence from each layer of BERT contains contextualized information that can be useful for downstream tasks that require a deeper understanding of the input sequence. We explored two traditional pooling methods to aggregate all output sequences: **Average Pool**[2], which computes the average over all output sequences, and **Max Pool**[2], which takes the element-wise maximum over all output sequences.

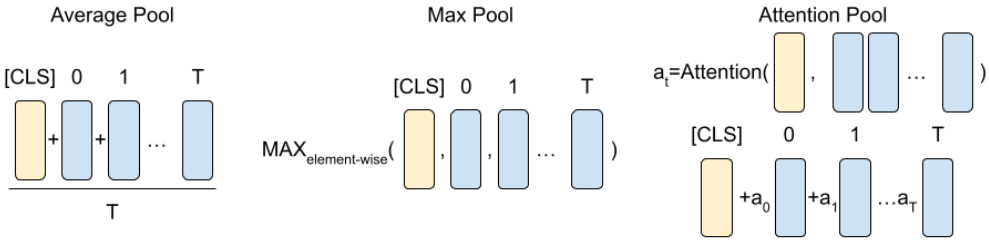

Figure 2: Average Pool, Max Pool, and Attention Pool for Ouput sequence aggregation

In addition, we theorize that not all output sequences share equal importance to summarise the sentences. Inspired by the attention aggregation mechanism present in graph neural networks [5], we devised the pooling method **Attention Pool**. We use `[CLS]` token as anchor, and then compute attention coefficients between the `[CLS]` token, $h_{[cls]}$, and the rest of the output sequences h_t:

$$e_t =< h_{[cls]}, h_t > \tag{1}$$

This indicates the importance of the output sequence embeddings h_t to the node `[CLS]` token. To make coefficients easily add across different sequences, we normalized attention coefficients e_t using the softmax function to obtain the attention weight α_t:

$$\alpha_t = \mathbf{softmax}(e_t) = \frac{\exp(e_t)}{\sum_{i=0}^{T} \exp(e_i)} \tag{2}$$

The attention pool output is computed using the `[CLS]` token and the weighted sum of the rest of the output sequences:

$$h_{attn} = h_{[cls]} + \alpha_0 h_0 + \alpha_1 h_1 + \cdots + \alpha_T h_T \tag{3}$$

3.4 SMART

Compared to size of the datatset used for pre-training, the training data for our downstream tasks is rather small. In addition, the high complexity of the pre-trained model often causes the fine-tuned

model to overfit on the training data for downstream tasks and fail to generalize to unseen data. We observed this behavior in our preliminary experiment, where dev set evaluation metrics got worse while training set metrics continuously increased. To address this issue, we implemented SMART [6]: **SM**oothness inducing **A**dversarial **R**egularization and B**R**egman p**R**oximal poin**T** op**T**imization.

Specifically, given the model $f(\cdot;\theta)$ and n data points of the target task denoted by $(x_i, y_i)_{i=1}^n$, where x_i's denote the embedding of the input sentences obtained from the first embedding layer of the language model and y_i's are the associated labels, our method solves the following optimization for fine-tuning:

$$\min_{\theta} F(\theta) = \mathcal{L}(\theta) + \lambda_s \mathcal{R}_s(\theta). \tag{4}$$

$\mathcal{L}(\theta)$ is the loss function for our downstream task. λ_s is a tuning parameter controlling the regularization strength. $\mathcal{R}_s(\theta)$ is our smoothness-inducing adversarial regularization loss:

$$\mathcal{R}_s(\theta) = \frac{1}{n}\sum_{i=1}^{n} \max_{\|\tilde{x}_i - x_i\|_\infty \leq \epsilon} \ell_s(f(\tilde{x};\theta), f(x_i;\theta)) \tag{5}$$

$\tilde{x}$ is the perturbed input embedding. ℓ_s is a function measuring the difference between the output distribution from the original embedding input and the perturbed embedding input. The core idea behind this regularization is that if we perturb the input by a small amount, the output logits should not change too much. Colloquially, this regularization term creates a small "buffer zone" (an infinity-norm ball) near each data point in embedding space. The output from the embedding within this buffer zone is enforced to not deviate from the original data points. This prevent our training data points from ending in the decision boundary, which should help with generalization. See the illustration from original paper[6] in Figure 3

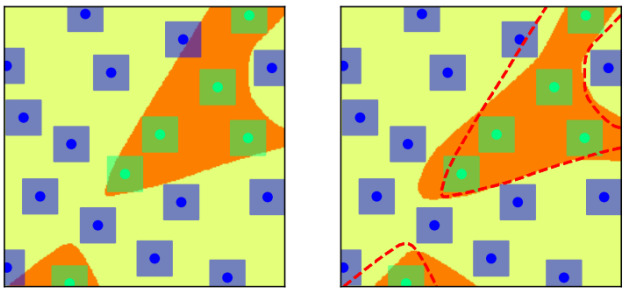

Figure 3: Adapted from the original SMART paper[6]. Decision boundaries learned without (a) and with (b) smoothness-inducing adversarial regularization, respectively. The red dotted line in (b) represents the decision boundary in (a). As can be seen, the output f in (b) does not change much within the neighborhood of training data points.

We also adapted SMART for our Bi-Encoder setup by perturbing each input separately and passing them through the encoder twice with their corresponding unperturbed pairs.

3.5 Downstream Tasks Model

We fine-tuned the pre-trained BERT-Base model to maximize performance on three sentence-level tasks: Sentiment Analysis, Paraphrase Detection, and Semantic Textual Similarity. This was accomplished by using different model designs that best applied to each task. We experimented with different combinations of sequence pooling strategies, sentence pair encoder architecture, and use of adversarial regularization. We chose the best performing model on dev set as our final downstream task model design. For more detailed analysis between different strategies, see Section 4.

For **Sentiment Analysis**, we used the `[CLS]` token's pooler output as our sentence embedding. We then passed the sentence embedding to a classification head consisting of a dropout layer and a linear layer. We used Cross Entropy Loss [7] and smoothness-inducing adversarial regularization as our objective function.

For **Paraphrase Detection**, we used a Cross-Encoder as our encoder architecture and the `[CLS]` token's pooler output as our sentence pair embedding. We then passed the sentence embedding to a classification head consisting of a dropout layer and a linear layer. We used Binary Cross Entropy Loss [7] and smoothness-inducing adversarial regularization as our objective function.

For **Semantic Textual Similarity**, we used a Cross-Encoder as encoder architecture and the `[CLS]` token's pooler output as our sentence pair embedding. We then passed the sentence embedding to a regression head consisting of a dropout layer, a linear layer, and a sigmoid activation layer. We used Mean Square Error Loss and smoothness-inducing adversarial regularization as our objective function. The original labels were normalized from [0,5] to [0,1].

3.6 Baseline

The authors of the original BERT paper. [1] reported an accuracy score of 0.712 on Quora Question Pair (QQP) and a Pearson correlation of 0.858 on the SemEval 2017 STS-B, which defined our baselines for paraphrase detection and semantic textual similarity, respectively. For sentiment analysis, the authors reported an accuracy score of 0.935 using the Stanford Sentiment Treebank binary (SST-2). However, this result is not directly comparable to our work, as we used the fine-grained SST-5 corpus for this project. Instead, a more appropriate baseline is the work of Munikar et al, who reported an accuracy score of 0.532 on the SST-5 corpus with BERT Base. [3]

4 Experiments

4.1 Data and Evaluation Method

For **Sentiment Analysis**, we used two different datasets in this project. The first was the Stanford Sentiment Treebank fine-grained corpus (SST-5), which contains 11,855 sentences collected from movie reviews found on the Rotten Tomatoes website. Each review has a discrete label ranging from 0 (negative sentiment) to 4 (positive sentiment). The second was the CFIMDB or CF-IMDB dataset, a set of 2,434 movie reviews collected from IMDB. Each review has a binary label of negative or positive.

During the verification step, both of these datasets were used to confirm model accuracy. For the multitask implementation step, we only tested the sentiment analysis model architecture with the SST-5 dataset. As this is a typical classification task, we used accuracy as our evaluation method.

For **Paraphrase Detection**, we used the Quora Question Pairs dataset (QQP), which contains over 400,000 question pairs from Quora. Each question has a binary label indicating whether the pair of questions are paraphrases of one another. We also used accuracy as our evaluation method for this binary classification task.

For **Semantic Textual Similarity**, we used the SemEval STS Benchmark dataset, which contains 8,628 different sentence pairs. Each pair has a continuous similarity score ranging from 0 (unrelated) to 5 (equivalent meaning). Since the label indicates the degree of similarity, we are interested in evaluating the correlation between prediction and ground truth rather than measuring their absolute differences. Therefore, we used Pearson correlation as our evaluation method. Given a set $\mathbf{p}$ of predictions p_i and a set $\mathbf{y}$ of labels y_i, with each p_i seeking to predict the corresponding label y_i, the Pearson correlation measures the linear correlation between the sets $\mathbf{p}$ and $\mathbf{y}$ as:

$$Corr_{Pearson}(\mathbf{p}, \mathbf{y}) = \frac{\sum_{i=1}^{n}(p_i - \bar{p})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(p_i - \bar{p})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}} \tag{6}$$

where $\bar{p} = \frac{1}{n}\sum_{i=1}^{n} p_i$ is the sample mean for $\mathbf{p}$ and $\bar{y}$ is defined analogously. Pearson correlation is commonly used for evaluating STS tasks.

4.2 Experimental Details

For our final models, we initialized the BERT encoder with provided pre-trained weights and allowed all model parameters to update during the training process. We performed hyper-parameter search and use dev set performance to determine best training setting for each downstream tasks as follow. We used an AdamW optimizer with a learning rate of 5e-5, 1e-5, and 1e-4 and a weight decay of

0.01, 0.01, and 0.02 for SST, QQP, and SemEval, respectively. A cosine learning rate decay with a linear warm-up of 1 epochs was used for SST and QQP, and a linear warm-up of 2 epochs was used for SemEval. Batch sizes of 64, 12, and 48 were used for SST, QQP, and SemEval, respectively. A smoothness-inducing adversarial regularization weight of 5.0, 1.0, and 1.0 were used for SST, QQP, and SemEval, respectively.

The teaching staff provided splits for all three datasets into Train, Dev, and Test sets. For SST and QQP dataset, we trained model for 5 epochs on training set. For SemEval, we trained model for 15 epochson trainint set. We saved the checkpoint which achieved the highest corresponding evaluation metrics on the corresponding Dev set.

All models were trained using a single NVIDIA RTX4090 GPU. The Dev set and Test set results for our best downstream task model are shown in Table 1. Our team name for the Dev and Test Leaderboard is Barely Even Reading Text (BERT).

4.3 Best Model Results

Model	SST-5 Accuracy	QQP Accuracy	SemEval Correlation
Baseline (Devlin, Munikar)	0.532	0.712	0.858
Ours (Dev Set)	0.557 (+0.025)	0.893(+0.181)	0.900(+0.042)
Ours (Test Set)	0.542	0.893	0.895

Table 1: Best BERT Multitask Dev and Test Accuracy and Correlation

As shown in Table 1 above, all of our best models see improvement upon baseline performance. Our best sentiment analysis model achieves accuracy scores of 0.557 and 0.542 on the dev and test sets, respectively. This is a 0.025 improvement compared to the baseline. Our best paraphrase detection model achieves an accuracy score of 0.893 on both the dev and test sets. This is a 0.181 improvement compared to the baseline. Our best semantic textual similarity model achieves Pearson correlation scores of 0.900 and 0.895 on the dev and test sets, respectively. This is a 0.042 improvement compared to the baseline. We believe that our use of a Cross-Encoder architecture and SMART regularization vastly improved the performance of our models. As time of writing, our submissions rank number 1 for overall score on both the dev and test leaderboards.

4.4 Pooling Strategy Comparison

Pooling Strategy	SST-5 Accuracy	QQP Accuracy	SemEval Correlation
`[CLS]` Token	0.526(-0.002)	0.781(-0.009)	0.768(-0.105)
Average Pool	**0.528**	**0.790**	**0.873**
Max Pool	0.511(-0.017)	0.789(-0.001)	0.779(-0.094)
Attention Pool	0.518(-0.010)	0.785(-0.005)	0.863(-0.010)

Table 2: BERT Multitask Dev Accuracy and Correlation with different pooling strategy and Bi-Encoder Setup

In this experiment, we investigate the effectiveness of different pooling methods. Here, the Bi-Encoder is used for all sentence pair tasks. The pooling strategy that performs best on all three tasks is Average Pool, which achieves scores of 0.528, 0.79, and 0.873 on SST, QQP, and SemEval, respectively. However, the second best pooling method varies between different tasks. `[CLS]` Token, Max Pool, and Attention Pool are the second best pooling methods for SST, QQP, and SemEval, respectively. From this result, we believe that only using the `[CLS]` Token would likely result in some information loss. It seems that Average Pool is the most effective way to aggregate the full output sequence from BERT when we use a Bi-Encoder architecture. And while we devised Attention Pool as a more generalized version of Average Pool, it was unable outperform the Average Pool on all three tasks.

4.5 Cross-Encoder vs. Bi-Encoder

In this experiment, we compare the performance between the Cross-Encoder and Bi-Encoder architectures on the sentence pair tasks. The Cross-Encoder performs vastly better than the Bi-Encoder across both tasks and with different pooling strategies. The Cross-Encoder represents a 0.111 and

Architecture	Pooling Strategy	QQP Accuracy	SemEval Correlation
Bi-Encoder	[CLS] Token	0.781	0.768
Cross-Encoder	[CLS] Token	**0.892**(+0.111)	**0.896**(+0.128)
Bi-Encoder	Average Pool	0.790	0.873
Cross-Encoder	Average Pool	**0.892**(+0.102)	0.891(+0.018)

Table 3: BERT Multitask Dev Accuracy and Correlation with different encoder archittecture and pooling stradegy

0.128 increase in scores on QQP and SemEval with the [CLS] Token, and a 0.102 and 0.018 increase in scores on QQP and SemEval with Average Pool. This performance increase is expected, as the Cross-Encoder can attend tokens between two sentences and establish a better representation for sentence pairs. We also seethat the [CLS] Token performs slightly better than the Average Pool when we use a Cross-Encoder. Because the Cross-Encoder has two different token types and a longer sequence, averaging over the full output sequence might not be advantageous.

4.6 Effect of SMART

SMART	Pooling Strategy	SST-5 Accuracy	QQP Accuracy	SemEval Correlation
No	[CLS] Token	0.526	0.892	0.896
Yes	[CLS] Token	**0.557**(+0.021)	**0.893**(+0.001)	**0.900**(+0.004)
No	Average Pool	0.528	0.892	0.891
Yes	Average Pool	0.539(+0.011)	0.893(+0.001)	0.894(+0.003)

Table 4: BERT Multitask Dev Accuracy and Correlation with and without SMART

In this experiment, we examined the effectiveness of SMART. For the sentence pair task, we only used a Cross-Encoder architecture. We see a large improvement using SMART in the sentiment analysis task. SST-5 Accuracy improves by 0.021 and 0.011 for [CLS] Token and Average Pool, respectively. We only see small improvement using SMART in QQP and SemEval. This could be attributed to a much worse over-fitting effect observed when fine-tuning on SST-5 without SMART. But we do see SMART help our models to generalize better, and all of our best models use SMART as part of their objective functions.

5 Analysis

In this project, we explored the effectiveness of pooling strategy, sentence pair encoder architecture, and SMART. We observed Average Pool strategy bring small gain in performance in our base implementation with Bi-Encoder setup. Average pooling with BERT's output sequences allows us to capture the contextualized information present in each token across all the layers of BERT. Average pooling involves taking the mean of the output vectors across each token in the input sequence, resulting in a fixed-length vector that summarizes the contextualized information of the entire sequence. By doing so, we can create a more robust representation of the input sequence that is not overly sensitive to individual tokens. We also found average pooling is not so as effective in Cross-Encoder. Average pooling did not work well with Cross-Encoder setups because Cross-Encoder models are designed to compare pairs of input sequences and make predictions based on the relationship between the two sequences. We are effectively average over two different token types, and may result in loss in information.

For sentence pair task, we found that Cross-Encoder perform vastly better than Bi-Encoder. Cross-Encoder concatenates both sentences as single sentence and pass to the transformer. This process enable our model attend to tokens between sentences, which could help model understand the difference between sentence pairs better. Even though Cross-Encoders achieve better performances than Bi-Encoders, there are also many scenario where Bi-Encoders are beneficial. Cross-Encoder only work when there exist pre-defined sentence pair, while Bi-Encoder could operate any individual sentence. For example, if we want to compare similarity of sentences across whole dataset. It will be much efficient to use Bi-Encoder to produce a sentence embedding for each sentence and then compare these embedding using distance metrics like cosine similarity. If we want to use Cross-Encoder in this example, we will need to create $\binom{n}{2}$ sentence pairs.

During our training process, we observed overfitting problem where model's performance on the training data is significantly better than its performance on the dev data. For SST-5 dataset especially, we saw dev accuracy goes down as train accuracy continuously increase. This behavior could cause by high complexity of the our BERT encoder backbone. SMART alleviate this issue by introducing a smoothness-inducing adversarial regularization term in our objective function. The reguarlization effect is achieved by injecting a small perturbation to the input and encourage the output of the model not to change much. Therefore, it create small buffer on each training data point and enforces the smoothness of the decision boundary. This effectively controls model's capacity.

6 Conclusion

In this project, we fine-tuned a pre-trained BERT model to simultaneously perform three sentence-level tasks: sentiment analysis, paraphrase detection, and semantic textual similarity. We explored the effects of pooling strategy, smoothness-inducing adversarial regularization, and encoding techniques on the performance of downstream task models. Ultimately, we were able to use these techniques to develop BERT multi-task models that improved upon the baselines set by Devlin and Munikar by large margin. While pooling had small or unclear effects on model performance, the use of SMART significantly increased model accuracy on sentiment analysis, and the use of Cross-Encoder vastly improves performance on paraphrase detection and semantic textual similarity tasks.

References

[1] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

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

[3] Manish Munikar, Sushil Shakya, and Aakash Shrestha. Fine-grained sentiment classification using bert. In *2019 Artificial Intelligence for Transforming Business and Society (AITB)*, volume 1, pages 1–5. IEEE, 2019.

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

[5] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[6] 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. *arXiv preprint arXiv:1911.03437*, 2019.

[7] David R Cox. The regression analysis of binary sequences. *Journal of the Royal Statistical Society: Series B (Methodological)*, 20(2):215–232, 1958.