BitBiggerBERT: An Extended BERT Model with Custom Attention Mechanisms, Enhanced Fine-Tuning, and Dynamic Weights

Stanford CS224N Default Project. Note: Khanh V. Tran shared 1 late day with Vladimir Gonzalez Migal

Khanh Tran
Department of Computer Science
Stanford University
vktran@stanford.edu

Thomas Hatcher
Department of Computer Science
Stanford University
thatcher@stanford.edu

Vladimir Gonzalez
Department of Computer Science
Stanford University
vladgm24@stanford.edu

Abstract

Our project implements an enhanced Bidirectional Encoder Representations from Transformers (BERT) language model. We implemented methods to increase accuracy in these tasks, particularly through multi-task fine-tuning and random sampling of data loaders. As as result, we achieve above benchmark accuracy on the three downstream tasks of interest: sentiment analysis, paraphrase detection, and similarity analysis. Additionally, we have also implemented a framework of linear layers for tag-free aspect analysis applied in all tasks as well as dynamic weighting mechanisms which allow the model to prioritize improved performance on the downstream tasks with the highest loss for a given epoch. With these extensions, we achieved somewhat improved accuracy from the base BERT model. While these improvements were not as high as hypothesized, these outcomes elucidate how model extensions demonstrated successfully in other studies might actually hinder performance in similar contexts.

1 Introduction

From reviews of products to comments on social media, we as a society interact more and more through text-based interactions online. These daily interactions happen on a incredibly large scale, and the text processing used to parse this data is beginning to have a large impact on how companies, managers, and designers engineer products for consumption. Over the last few years, innovations in the natural language processing (NLP) field—particularly transformer-based models which produce dynamic, context-dependent representations of very large datasets—have enabled language models to accomplish sophisticated analytical tasks. In the context of this study, we will be focusing on three downstream tasks: (1) sentiment analysis, which classifies text into 5 different sentiment classes ranging from negative to positive (2) paraphrase detection, which identifies whether two distinct text segments convey the same meaning (3) similarity analysis, which identifies whether two distinct text segments are more similar to each other than not. Successful accomplishment of these tasks bears significant economic motivation, given their applications to review aggregation software.

In order to do so, we utilized the canonical Bidirectional Encoder Representations from Transformers model, i.e. "BERT". According to Devlin et al. (2018), the BERT model can be fine-tuned to create

models for a wide range of tasks without substantial task-specific architecture modifications. While the baseline BERT model we implemented was able to predict sentiment accurately more than half the time, its performance on the other two downstream tasks were not able to surpass this threshold. Thus, we were motivated to find ways to extend the functionality of the baseline BERT model in order to enhance performance across all three tasks.

The rest of this paper is organized as follows. In Section 2, we review the related work on BERT and aspect-based sentiment analysis. In Section 3, we outline our approach to implementing both the baseline BERT model, as well as the extended BERT model with the aforementioned additions. We then outline our experimental methodology as well as an analysis of our experimental results. Finally, in Section 5, we discuss future directions for this project.

2 Related Work

A. *Bidirectional Encoder Representations from Transformers (BERT)*
BERT is based on the transformer architecture, which relies on self-attention mechanisms to process input text Devlin et al. (2018). Unlike traditional models that process data sequentially and, as such, generate static word embeddings, BERT processes entire sequences of data in parallel, which allows it to generate dynamic embeddings to better capture the context of a word. BERT is pre-trained on a large corpus of text using unsupervised tasks, and can be fine-tuned on specific NLP tasks.

B. *Aspect Based Sentiment Analysis*
Aspect Based Sentiment Analysis (ABSA) is a subfield of sentiment analysis which focuses on identifying the sentiment expressed towards specific aspects within a text. BERT has previously been combined with the Interactive Attention Networks (IAN) model by Ma et al. (2017) to further improve the accuracy of the aspect-based sentiment analysis. However, this original technique heavily relied upon manual tagging of user reviews according to the predefined aspects, which is an inherently laborious, time-consuming, and biased process. More recently, a new Multiple-Attention Network (MAN) approach has been shown by Qiang et al. (2020) to achieve more powerful ABSA without the need for aspect tags. This technique makes use of a Position-aware Attention submodule which captures the relevance of contextual words.

C. *Relation to BitBiggerBERT*
For this study, we will be building on the BERT model as described by Devlin et al. (2018) with extensions tailored towards both the goal of generalizability across the aforementioned downstream tasks, as well as task-specific performance. To do so, one of the main extensions we will implement includes the MAN described by Ma et al. (2017). While we are not implementing ABSA in full to avoid possibly overfitting on the sentiment analysis task, we hypothesize that implementing a Position-aware attention layer will be beneficial for all three downstream tasks. Position-aware attention allows the model to understand the context of each word in a sentence based on its position. This is crucial for sentiment analysis, as the sentiment often depends on the order of words. For example, "not good" has a different sentiment than "good not," even though both contain the same words. In tasks like paraphrase detection and similarity analysis, understanding the relationship between distant words in a sentence is essential. Position-aware attention helps the model capture these long-range dependencies, enabling it to detect subtle differences or similarities between sentences.

3 Approach

Our approach to this project was broken up into three main phases. First, we implemented the canonical BERT model as described in Devlin et al. (2018) and fine-tuned on all three datasets for the three downstream tasks of sentiment analysis, paraphrase detection, and similarity analysis. Second, we implemented extensions to our fine tuning process to improve on our three downstream tasks of sentiment analysis, paraphrase detection, and semantic similarity analysis. Third, we updated the framework for model predictions using Multiple-Attention Network as described by Qiang et al. (2020).

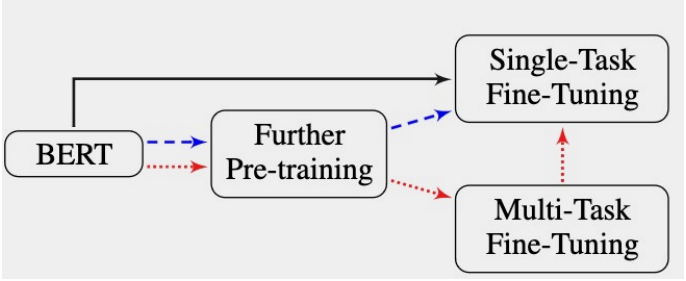

Figure 1: Pre-training and fine-tuning approaches to BERT (from Chi Sun and Huang (2019)). In our current implementation, we follow the red dashed line.

3.1 Baseline BERT Model

As noted in Section 2, BERT is a transformer-based model that creates word representations based around the context of a phrase, which ultimately aids in natural language understanding of ambiguous texts. By implementing the basic BERT model as well as using pre-trained weights for sentiment analysis on the included SST and CFIMDB datasets, we now have a functioning BERT model from which we can expand to achieve enhanced performance. Within BERT, the embedding layer, consisting of a word and position, are embedded along with encoder layers that are, in our case, implemented as a stack of transformer layers. The BERT embedding layers were implemented in the class `BertModel` where we cast input token IDs to word embeddings. The BERT layers themselves include multi-head attention implemented in the `BertSelfAttention` class and feed forward function implemented in the `BertLayer` class. Attention is defined in eq. 2 in the appendix.

3.2 Extension 1: Additional Finetuning

The extension tasks to our base BERT model were implimented in the `MultitaskBERT` class, where fine-tuning is performed on specific tasks. In our first implementation, the baselined followed the black training flow, as outlined in Fig. 1, which involved using the pre-trained bert model and fine-tuning on the sst (sentiment analysis) dataset. To improve performance, we decided to fine-tune the model on all three datasets during each training batch. By utilizing the class `CombinedLoader`[1] from Lightning AI, we are able to take random samples from all three datasets associated with the three downstream tasks, and trains on each of the samples during each training epoch. The combined sampler flags the dataset from which each batch was chosen such that we can apply tailored loss functions for each task.

3.3 Extension 2: Tailored Loss Functions

For our next extension we updated the loss functions when finetuning with samples from the paraphrase detection and semantic evaluation datasets. For the paraphrase detection task, we used `Binary CrossEntropyLossWithLogits`–defined in eq. 1 from PyTorch library (see appendix). Binary cross-entropy loss is particularly suitable for binary classification problems, such as paraphrase detection, where each input sample is classified into one of two classes. In the context of paraphrase detection, the two classes are "is a paraphrase" (often labeled as 1) and "is not a paraphrase" (often labeled as 0).

For semantic similarity analysis, we used Mean Squared Error Loss (`MSELoss`) because this task typically involves predicting a continuous similarity score between two sentences. The ground truth labels in the SemEval dataset are continuous values (e.g., between 0 and 5), and `MSELoss` is well-suited for regression tasks involving predicting continuous outcomes.

[1] `https://lightning.ai/docs/pytorch/stable/api/lightning.pytorch.utilities.combined_loader.html`

3.4 Extension 3: Dynamic Weighting, Gradient Surgery

We noticed that our model was getting over-optimized on the paraphrase task, since the dataset on which it trained contained the most data. As such, we decided to implement dynamic weighting so that no single task would dominate the training process. Dynamic weighting was implemented at the end of each epoch according to eqs. 5 and 6. By dynamically adjusting the weights of each dataset during training, the model is encouraged to learn features that are generalized across all datasets, rather than overfitting to the specifics of the larger dataset. This can lead to improved performance on unseen data and a more robust model overall.

We also attempted gradient surgery to overcome our model's over-optimization on the paraphrase task, due to its common application in the training of deep learning models with multiple tasks or objectives (Yu et al., 2020), via directly manipulating the gradients during the training process. After trying both methods, we ended up moving forward with dynamic weighting since it led to higher dev accuracies.

3.5 Multiple-Attention Network (MAN)

Due to the complexity of the tasks, we decided to add a framework of linear layers as described in Qiang et al. (2020). This is incorporated after we get output hidden layers from the BERT model, and therefore is utilized to produce a more nuanced token that is used to produced predictions for each task. This process includes three main steps: (1) self-attention, (2) position-aware attention, and (3) regularization and concatenation.

As seen in eq. 7, we calculate self-attention using the outputted hidden layers from BERT and its transform. This is very similar to who self-attention is calculated within the BERT model itself. However, this term is calculated for the choice of four aspects. Therefore, weighted outputs are calculated in this manner for *each* aspect, with weights that are updated through a linear layer.

The outputs from the self attention calculation are then multiplied by a second position-aware weighting factor (again, for each aspect). This can be seen in equation 8. Here, instead of self-referencing the BERT model output, it takes in a average of the word, position, and token embedding. This has been shown in both Qiang et al. (2020) and R. He and Dahlmeier (2017) to incorporate the global context of the sentence. This second weighting factor is then applied to the output of step (1).

Regularization terms are calculated as seen in eq. 9 and incorporated into task loss to encourage orthogonality between attention weights on different aspects. The results of each aspect are then concatenated to get the overall analysis of the sentence, which is then used later on in the training process to produce predictions.

4 Experiments

4.1 Data

We are using the Stanford Sentiment Treebank (SST) dataset for sentiment analysis, the Quora dataset for paraphrase detection, and the SemEval dataset for semantic textual analysis. The SST dataset consists of $11,855$ single sentences extracted from movie reviews. The dataset was parsed with the Stanford parser and includes a total of $215,154$ unique phrases, annotated by 3 human judges. Each phrase has a label of negative, somewhat negative, neutral, somewhat positive, or positive. The Quora dataset consists of $400,000$ question pairs with labels indicating whether particular instances are paraphrases of one another. We use a subset of the dataset with $202,152$ question pairs. The SemEval dataset contains 8631 sentence pairs that are rated on a continuous scale from 0-5 on how similar they are. Here 5 means both sentences have an equivalent meaning, whereas 0 means they are unrelated.

4.2 Evaluation method

For sentiment analysis we used accuracy, as defined in eq. 3 (see appendix), to evaluate the model. This is convenient as both Devlin et al. (2018) and Ma et al. (2017) used the above-defined accuracy as their metric for sentiment analysis. For paraphrase detection we also used accuracy.

For semantic textual analysis we used the Pearson correlation coefficient (Agirre et al., 2013), seen in eq. 4 (see appendix), of the true similarity values against the predicted similarity values.

Qualitatively, we also graphed the losses for each of the tasks across batches, for multiple epochs. We did so in order to observe whether the loss plateaued over the course of an epoch for the tasks to see whether our model is able to converge.

4.3 Experimental details

In order to obtain a fully trained BERT model, we trained for 10 epochs used a learning rate of 1e-3 for pre-training and 1e-5 for multi-task fine-tuning. We also used default configuration with a hidden dropout probability of 0.3 and hidden size of 768. These default parameters were constant throughout testing. We ran experiments on a GCP VM, using the `--use_gpu` flag for most efficient training. In addition, we ran fine-tuning with all three datasets, sampling in a round robin fashion with a random ordering of tasks for each dataset until each was depleted (e.g. the `max_size` and `max_size_cycle` options for the `CombinedLoader`). All models used the AdamW optimizer. In addition, for MAN, an aspect size of four and $\lambda = 0.5$ (where λ is the coefficient of the regularization terms) were chosen based on the work from Qiang et al. (2020).

4.4 Results

For the Original Baseline BERT model, we implemented the canonical BERT model with fine-tuning only on the SST dataset for the sentiment analysis task. For the Milestone BERT model, we implemented multi-task fine-tuning along with tailored loss functions for each downstream task. For the sentiment analysis task, we continued to use `CrossEntropyLoss`; for the paraphrase detection task, we used `CosineEmbeddingLoss`; and finally, for the similarity analysis task, we used `MSELoss`. As expected after implementing multi-task fine-tuning, we found that the accuracies for all tasks (except for sentiment analysis, which the model was no longer over-optimizing on) had increased by a significant margin as seen in 2.

After implementing Milestone BERT, we decided to make a minor change and use `BinaryCrossEntropyLoss` instead of `CosineEmbeddingLoss` for the paraphrase detection task, which significantly increased our accuracies in both the paraphrase detection task as well as the similarity analysis task. Having achieved our highest scores to date with this version of the BERT model, we decided to use this as our Updated Baseline, to which we could compare our accuracies after adding more sophisticated extensions in the second phase of model development.

As discussed in the approach, our second phase of model development incorporated two main extensions: dynamic weighting and MAN. These results from these extensions can be found in Figure 3.

	Original Baseline	**Milestone**
SST dev accuracy	0.529	0.495
Paraphrase dev accuracy	0.487	0.547
STS dev correlation	-0.120	0.430
Overall dev score	**0.485**	**0.586**

Figure 2: Dev evaluation accuracies pre- and post- milestone to baseline BERT model. For Sentiment (SST) and Paraphrase tasks, we were evaluating the accuracy, as described, and for Similarity (STS) we used Pearson correlation.

5 Analysis

Upon implementing multi-task fine-tuning for the Milestone BERT model, we found experimentally that our model was becoming "bottlenecked", e.g. becoming over-optimized for the paraphrase detection task. We suspect that this problem arose because the paraphrase detection task was dominating the training phase due to the sheer amount of data in the associated Quora dataset. As such, we sought out methods to adjust the importance of under-performing tasks during training, as well as adjustments to the overall model architecture to better optimize for more complex downstream tasks.

	SST Accuracy	Paraphrase accuracy	STS correlation
Updated Baseline	0.508 0.511	0.788 0.782	0.672 0.645
+Dynamic Weighting	0.493	0.775	0.514
+MAN	0.521 0.519	0.684 0.687	0.715 0.670
+Dynamic Weighting +MAN	**0.470**	**0.775**	**0.544**

Figure 3: Evaluation metrics for the Updated Baseline model, and the models with extensions (test metrics in red). For Sentiment (SST) and Paraphrase tasks, we were evaluating the accuracy, as described, and for Similarity (STS) we used Pearson correlation.

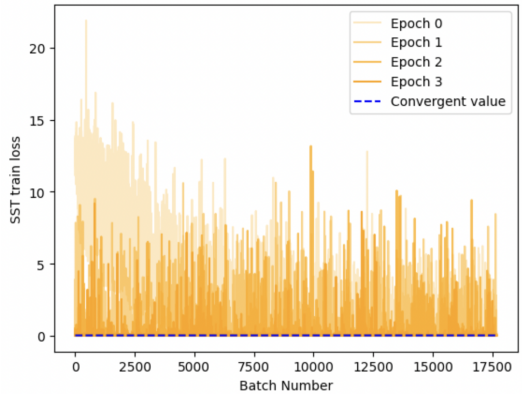

Figure 4: Training loss for sentiment analysis over 4 epochs running with `max_size_cycle`, for +Dynamic Weighting +MAN model.

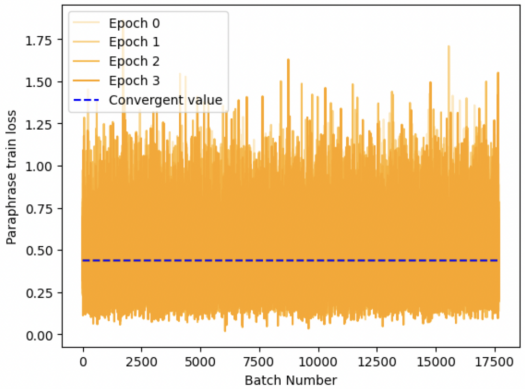

Figure 5: Training loss for paraphrase analysis over 4 epochs running with `max_size_cycle`, for +Dynamic Weighting +MAN model.

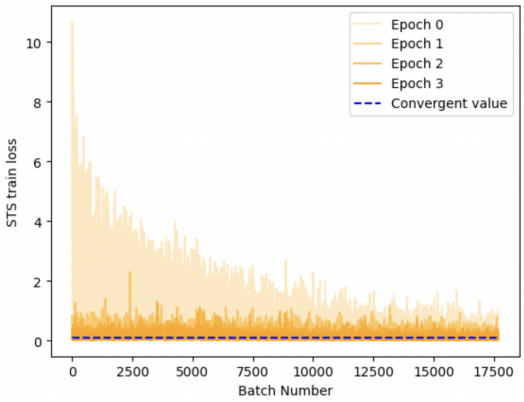

Figure 6: Training loss for similarity classification over 4 epochs running with `max_size_cycle`, for +Dynamic Weighting +MAN model.

In figures 4 and 6, our calculated losses for sentiment analysis and similarity classification are converging within the first two epochs, then fluctuating around the same convergent value for the last few epochs. In figure 5, we see that the loss fluctuates around a seemingly pre-converged value for the paraphrase task. This means that the saved model we use to start each fine tuning process seems to have optimal weighting when it come to paraphrase detection. In addition, given paraphrase detection was the simplest of the three tasks, we expect it to quickly converge. Even when experimenting with different hyper parameters, this same behavior and results were displayed. Given that our evaluation metrics did not seem to have a significant difference between these more sophisticated extensions

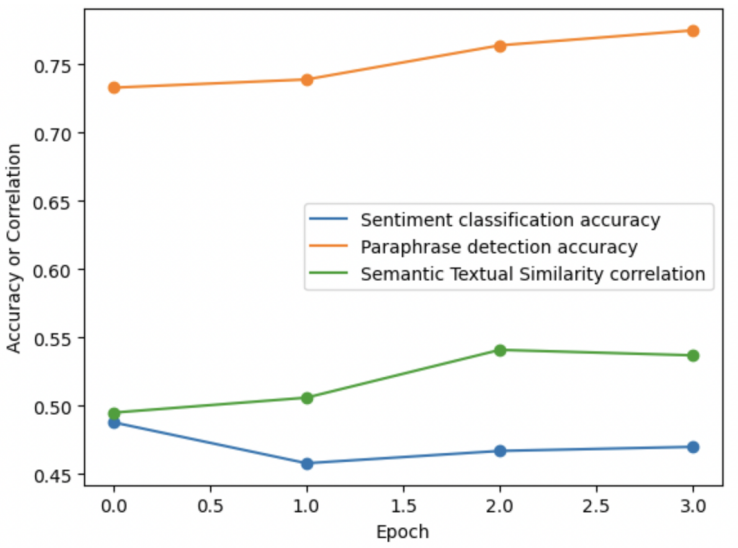

Figure 7: Here we see the epoch-wise evolution of our evaluation metrics for all three tasks. This is from the same run as in figures 4-6.

and our new baseline, by analyzing the loss graph alone, it is hard to say whether or not our model is getting caught in a localized minimum.

We can see from 7 that with both MAN and dynamic weighting implemented, the increase in metric for worse-performing tasks seems to be at most the same as increases in the best-performing task. In addition, from our resulting accuracies in Figure 3, it appears that dynamic weighting is causing our evaluation metrics to behave different than the implementation's intended purpose, with large discrepancies between task performances. There are several potential causes. We first explored applying MAN to only sentiment analysis to see if the regularization terms being added to the loss was skewing weights. However, this did not improve the time-like behavior of the metrics. As seen in the loss figures, SST and STS start out at much higher losses and then eventually converge to a very small loss, whereas paraphrase analysis starts and stays at a relatively small loss. These initial losses are skewing the average used when calculating weights (eq. 6), and this difference over epochs

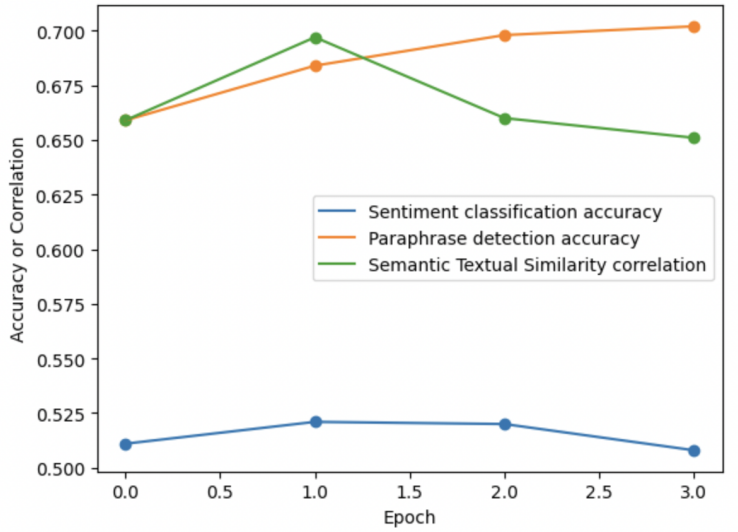

Figure 8: Here we see the epoch-wise evolution of our evaluation metrics for all three tasks. These are from running only MAN on sst and sts.

would inform the model that tasks with large changes in loss should not receive as much comparative weight than the task whose loss is relatively constant throughout the training process. This goes hand-in-hand with the observed behavior of metrics starting to drop after the first one to two epochs.

As seen in the results, it is clear we implemented a functioning multiple-attention network, with novel demonstration of the use of these networks on multiple downstream tasks (sentiment and similarity analysis), not just sentiment analysis as implemented in Qiang et al. (2020). As an extension from the base model, we are able to achieve higher performance for the more complicated downstream tasks than our updated baseline. The performance of paraphrase detection decreased, which is caused by the additional loss terms from regularization added to sst and sts tasks, which favored those tasks in the training. In addition, when comparing figures 7 and 8, we see that despite the metrics not having any particular weighting scheme, are staying at higher values instead over over-optimizing one task.

6 Conclusion

In this paper, we explore the effects of various extensions on the performance of the canonical BERT model for three downstream tasks: sentiment analysis, paraphrase detection, and similarity analysis. In order to accomplish the goal of optimizing a single model for multiple downstream tasks, we designed our extensions to meet two somewhat orthogonal goals–generalizability across all tasks, as well as task-specific performance.

We achieved a large increase in accuracy across the three tasks by simply fine-tuning on each of the datasets during the training phase, as well as utilizing tailored loss functions for each of the tasks. In terms of key takeaways from the second phase of the project, we demonstrated experimentally that the implementation of Multi-Attention Networks applied to all three tasks can be utilized to more effectively predict sentiment and similarity of sentences, leading to a more sophisticated BERT language model.

In terms of lessons learnt over the course of this study, we found that it can be very challenging to diagnose poor accuracies, particularly after implementing multiple novel extensions. For the purposes of obtaining datapoints on model performance, we needed to run our model for 10 epochs using the `max_size` option for the `CombinedLoader` in order to train on all available data, which ended up taking approximately twelve hours per run. We addressed this debugging challenge by implementing data visualization methods to observe interim model performance during the training phase. While this resolves some of the opacity in the model decision making process, it was nonetheless difficult to elucidate the source and reasoning behind decreased accuracies.

In the future, optimizing the model's computational efficiency and scalability, as well as increasing visibility into the model's decision-making processes through visualization of attention weights are potential considerations. Both of these extensions will help address the fundamental difficulty with the debugging process, and thus allow for further improvement of our extended BERT model for the intended downstream tasks. In addition, it would be informative to experiment more with MAN-related hyperparameters and small tweaks in the implementation that could increase specific tasks' performance without having a inverse effect on others'.

7 Teammate Contributions & Late Days

- **Khanh Tran**
  Khanh wrote the majority of the content on the final report. She also implemented the baseline BERT model, as well as the combined batching and dynamic weighting functionalities. She also assisted in the debugging process along with idea generation and literature review for the extended BERT model. **Late days are shared between teammates.**
- **Thomas Hatcher**
  Thomas' most notable contributions were implementing MAN, assisting in other extensions/improvements as well as producing figures for analysis. In addition, he wrote and revised significant portions of the write up. **Late days are shared between teammates.**
- **Vladimir Gonzalez Migal**
  Vladimir worked on the functionality of combined batching as well as updating the model

parameters. He was also responsible for the implementation of the updated loss functions and dynamic weighting. He was also responsible for defining and running a majority of the experiments and writing the section in the writeups. **Late days are shared between teammates.** Late Days: Khanh V. Tran shared 1 Late Day with Vladimir Gonzalez Migal

References

Eneko Agirre, Daniel Cer, Mona Diab, Aitor Gonzalez-Agirre, and Weiwei Guo. 2013. *sem 2013 shared task: Semantic textual similarity.

Yige Xu Chi Sun, Xipeng Qiu and Xuanjing Huang. 2019. How to fine-tune bert for text classification? *Chinese Computational Linguistics: 18th China National Conference, CCL 2019, Kunming, China, October 18–20, 2019*.

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

Dehong Ma, Sujian Li, Xiaodong Zhang, and Houfeng Wang. 2017. Interactive attention networks for aspect-level sentiment classification. *arXiv*.

Yao Qiang, Xin Li, and Dongxiao Zhu. 2020. Toward tag-free aspect based sentiment analysis: A multiple attention network approach. *arXiv*.

H. T. Ng R. He, W. S. Lee and D. Dahlmeier. 2017. An unsupervised neural attention model for aspect extraction. *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*.

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*.

Tianhe Yu, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. 2020. Gradient surgery for multi-task learning. *arXiv*.

8 Appendix

8.1 Binary Cross-Entropy Loss

$$L(y, \hat{p}) = -\frac{1}{N}\sum_{i=1}^{N}[y_i \cdot \log(\hat{p}_i) + (1 - y_i) \cdot \log(1 - \hat{p}_i)] \tag{1}$$

where:

- L is the binary cross-entropy loss,
- N is the number of samples in the batch,
- y_i is the true label for the i^{th} example, which can be 0 or 1,
- $\hat{p}_i$ is the predicted probability for the i^{th} example.

8.2 Attention

Attention calculated with input query, key and value tensors Vaswani et al. (2017):

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2}$$

Where Q is the query tensor, K is the key tensor and V is the value tensor.

8.3 Accuracy

$$\text{Accuracy} = \frac{TP + TN}{(TP + FP) + (TN + FN)} \tag{3}$$

8.4 Pearson Correlation

$$r = \frac{\sum(x_i - \bar{x})}{\sqrt{\sum(x_i - \bar{x})^2 \sum(y_i - \bar{y})^2}} \tag{4}$$

Where x_i, y_i represents the values of the x,y-variables in a sample, $\bar{x}$ is the mean of the values of the x-variables and $\bar{y}$ the mean of the values of the y-variable.

8.5 Dynamic Task Weighting

$$L_{\text{avg}}^{\text{task}} = \frac{\sum_{i=1}^{N_{\text{task}}} L_i^{\text{task}}}{N_{\text{task}}} \tag{5}$$

$$w_{\text{new}}^{\text{task}} = (1 - \alpha) \cdot w_{\text{old}}^{\text{task}} + \alpha \cdot \frac{L_{\text{avg}}^{\text{task}}}{\sum_{\text{all tasks}} L_{\text{avg}}^{\text{task}}} \tag{6}$$

where:

- $L_{\text{avg}}^{\text{task}}$ is the average loss for a given task.
- N_{task} is the number of examples for the given task.
- L_i^{task} is the loss for the i-th example of the given task.
- $w_{\text{old}}^{\text{task}}$ and $w_{\text{new}}^{\text{task}}$ are the old and new weights for the given task, respectively.
- α is the weighting factor (e.g., 0.1).
- T is the total number of tasks.

8.6 MAN

$$\alpha_i = \text{Softmax}(\tanh\left(Q_i h_i^T + b_\alpha\right)) \tag{7}$$

Here, h_i are the hidden layers, and Q_i is a linear transform of h_i, implemented using a linear layer. The bias term is a learned parameter. (Qiang et al., 2020)

$$\beta_i = \text{Softmax}(\tanh\left(M h_i^T + b_\beta\right)) \tag{8}$$

This new term M is a linear transform of the average over embeddings $\bar{h}$. (Qiang et al., 2020)

$$R_{\alpha,\beta} = ||M_{\alpha,\beta} M_{\alpha,\beta}^T - I|| \tag{9}$$

Here, $M_\alpha = [\boldsymbol{\alpha}^1 ... \boldsymbol{\alpha}^k]$ and $M_\beta = [\boldsymbol{\beta}^1 ... \boldsymbol{\beta}^k]$ with k being the number of aspects. (Qiang et al., 2020)