PradrewBert: The Efficient One

Stanford CS224N Default Project

Pradyumna Saligram
Department of Computer Science
Stanford University
psaligram@stanford.edu

Andrew Lanpouthakoun
Department of Computer Science
Stanford University
andlanpo@stanford.edu

Abstract

We explore advanced fine-tuning techniques to boost BERT's performance in sentiment analysis, paraphrase detection, and semantic textual similarity. Our approach leverages SMART regularization to combat overfitting, improves hyper-parameter choices, employs a cross-embedding Siamese architecture for improved sentence embeddings, and introduces innovative early exiting methods. Our fine-tuning findings currently reveal substantial improvements in model efficiency and effectiveness when combining multiple fine-tuning architectures, achieving a state-of-the-art performance score of on the test set, surpassing current benchmarks and highlighting BERT's adaptability in multifaceted linguistic tasks.

1 Key Information to include

Mentor: Sonia Chu - **External Collaborators:** None **Project Sharing:** Not Sharing **Work Contributions: Pradyumma:** Early Exiting and Poster **Andrew:** SMART Regularization, Concatenation/ Siamese Architecture

2 Introduction

With the advent of transformers in 2017 (Vaswani et al., 2017) the field of Natural Language Processing has improved immensely. Progressions such as Large Language Models, particularly ChatGPT, have caused the world to see the full capability of Machine Learning in changing lives. This ability by large language models is largely explained by progressions made in various downstream tasks and research in Multitask Learning(MTL). This research method allowed models to learn various different tasks at once. The joining of knowledge of multiple tasks pushed models to learn more complex and difficult tasks by using this information at once. However, the efficacy and efficiency of models doing various divergent tasks varies greatly. The goal of this project is to increase the capability of BERT model (Devlin et al., 2018) on three tasks in conjunction.

This research focuses on improving three tasks: sentence sentiment classification, paraphrase detection, and sentence semantic similarity. Our approach first focuses on improvement of sentiment classification. After training this model, we choose to train on large datasets to learn paraphrase detection and sentence semantic similarity in conjunction with further training on sentiment classification. Training for these three tasks is done simultaneously, to produce a truly multi-skilled model. This is done using various different methods. The goal of this project is not to simply train our model to accomplish these tasks; rather, the goal of this project is to boost accuracy and efficiency. Therefore, we implement SMART a regularization method (Jiang et al., 2019) in hopes of reducing overfitting, we implement Cosine Embedding Loss, and early exiting to improve computational Efficiency.

Our findings suggest that SMART regularization yields great initial improvements for Paraphrase detection, but does not continue as training continues and does not perform as well in Sentiment Classification and Textual Similarity Correlation. Therefore, we mostly paired SMART regularization with Paraphrase Detection

Because SMART showed great initial improvements, we found that it pairs very well with our other main fine-tuning technique of early exiting. (Xin et al., 2020) Xin et al. (2021)

3 Related Work

Researchers aim to improve multi-task models in the hopes of leveraging transfer learning. This is fairly intuitive, when a model performs well on one task, it seems fairly simple for it to perform well on a similar task. However, the results of a transformer model in similar tasks has surprisingly poor results; therefore, we must work to have our model perform well on a multitude tasks. Our work aims to train the Bidirectional Encoder Representations from Transformers (BERT) Devlin et al. (2018) on these multiple tasks, we use ideas from other papers to fine tune a model to do so.

(Jiang et al., 2019) shows that regularization tools such as perturbations can combat overfitting by adding additional noise. This is similar to the dropout layer (Srivastava et al., 2014) that is already within the original BERT model. However, SMART is a much more advanced regularization technique that is adaptive, generalizes to multiple tasks, smooths the distribution, and uses gradients. We use SMART regularization over other regularization techniques because of its computational efficiency.

We utilize early exiting techniques in the hopes of maximizing our computational efficiency in unison with accuracy. BERxiT (Xin et al., 2021) introduces a confidence-based mechanism to exit inference early if the model is sufficiently confident in its prediction, offering the perfect mechanism for our goal. DeeBERT (Xin et al., 2020) further extends this by incorporating a dynamic inference mechanism that adjusts the exit point based on task complexity and input characteristics.

We worked with a few different model architectures for tasks with multiple sentences. Namely, we looked at concatenation prior to or after embedding. (Schroff et al., 2015) speaks about a Siamese Architecture that that passes each sentence into parallel models and then compares Cosine Similarity between the two embeddings. This was also the method utilized within the very successful SBERT paper for their Textual Similarity analysis. (Jiang et al., 2019). Additionally, we look towards the original BERT paper (Devlin et al., 2018) which concatenates sentences beforehand and uses a separation token prior to embedding. Both of these have proven results and we experiment with both during training.

4 Approach

We developed a comprehensive minBERT Transformer model, comprising 12 transformer layers where each layer includes a self-attention mechanism. The transformation within each BERT layer, denoted as $BL(h)$, is formulated as:

$$BL(h) = LN(h + FFN(LN(h + MH(h))))$$

where LN denotes layer normalization, SA represents the self-attention function, FFN stands for the feed-forward network, while MH is the multi-head attention component.

4.1 Baseline Multitask Classifier

Our baseline multitask BERT model leverages the pre-trained weights of the base minBERT for feature extraction across all tasks. The model architecture is extended with three task-specific heads, each comprising a linear layer, tailored to each respective task output requirement:

- **Sentiment Analysis (SST)**: Produces a 5-class output using a linear layer, applying cross-entropy loss to quantify the difference between predicted probability distributions and one-hot encoded target vectors.
- **Paraphrase Detection (Para)**: Initially used MSELoss but later optimized using cosine embedding loss for better performance.
- **Semantic Textual Similarity (STS)**: Produces a single logit interpreted as a continuous similarity score, using MSELoss against the labels.

The BERT model configuration includes a hidden size of 768, divided across 12 attention heads, resulting in an attention head size of 64 while dropout with a probability of 0.3 is also applied.

Training is conducted sequentially within an epoch, each consisting of a loop that fully trains each task. Initially, the base BERT model parameters are frozen, establishing a lower baseline for accuracy after which single-task fine-tuning is performed for each task, updating all model parameters to achieve higher accuracy. The fine-tuning strategy is applied to both the SST and CFIMDB datasets, creating separate models for each task. Finally, a fully fine-tuned multi-task model is trained, updating 100% of the pre-trained model's parameters to generate a comprehensive model for all downstream tasks.

This approach allows us to establish both lower and higher bounds of accuracy, demonstrating the efficacy of parameter-efficient fine-tuning. The model's "pretrain" mode freezes BERT parameters, while the "fine-tune" mode updates all parameters based on task-specific data. For this baseline model, a batch size of 8 is used for all tasks, and training is conducted over 10 epochs.

4.2 SMART Regularization and Alternative Optimizers

To mitigate overfitting, particularly notable in the SST-5 dataset, we incorporated the SMART regularization technique. SMART consists of two main components: Smoothness-inducing Adversarial Regularization and Bregman Proximal Point Optimization, with our implementation primarily focusing on the former. This method modifies the fine-tuning optimization process to enhance model smoothness and reduce overfitting, leading to better generalization to the target domain. Specifically, the optimization objective is reformulated to integrate a smoothness-inducing adversarial regularizer along with the standard loss function, defined as:

$$\min_{\theta} F(\theta) = L(\theta) + \lambda_s R_s(\theta),$$

where $L(\theta)$ represents the loss function for our downstream task, λ_s is a tuning parameter, and $R_s(\theta)$ is the smoothness-inducing adversarial regularizer, defined as:

$$R_s(\theta) = \frac{1}{n} \sum_{i=1}^{n} \max_{\|\mathbf{x}_i' - \mathbf{x}_i\|_p \leq \epsilon} l_s(f(\mathbf{x}_i'; \theta), f(\mathbf{x}_i, \theta)).$$

For classification tasks, l_s is selected as the symmetrized KL-divergence loss, and for regression tasks, it is chosen as the mean-squared-error loss.

Initially, we implemented two methods aimed at pruning for computational efficiency and preventing overfitting. We applied SMART regularization across all three tasks but observed a positive accuracy change only for paraphrase detection. This outcome was unexpected given the substantial size of the CFIMDB dataset. Nevertheless, we continued to employ SMART exclusively for paraphrase detection.

For optimization, we explored RMSprop and SGD as alternatives to AdamW. The RMSprop algorithm accelerates the optimization process by maintaining the moving average of squared gradients for each weight and dividing the gradient by the square root of the mean square mathematically expressed as:

$$v(w,t) := \gamma v(w,t-1) + (1-\gamma)(\nabla Q_i(w))^2$$
$$w := w - \frac{\eta}{\sqrt{v(w,t)}} \nabla Q_i(w)$$

where γ is the forgetting factor. Stochastic Gradient Descent updates the weights using the formula:

$$w := w - \eta \nabla Q_i(w)$$

4.3 Early Exiting Mechanisms and Novel Fine-Tuning Strategy

To reduce extensive training times, which often extended to approximately 20 hours, we incorporated an Early-Exiting Architecture, reducing training time to around 18 hours on an NVIDIA T4 GPU. Our early-exiting model is based on established techniques, introducing innovative fine-tuning strategies.

The backbone model consists of a multi-layer pre-trained BERT architecture with classifiers attached to each layer, enabling accelerated inference through early exiting. The hidden state of the [CLS] token at the i-th layer, denoted as h_i, is defined by:

$$h_i = f_i(x; \theta_1, \ldots, \theta_i),$$

where x is the input sequence, θ_i represents the parameters of the i-th transformer layer, and f_i maps the input to the i-th layer's hidden state. Each layer i employs a classifier with a loss function:

$$L_i(x, y) = H(y, g_i(h_i; w_i)),$$

where x and y are the input and label, g_i is the classifier at the i-th layer, w_i are the classifier's parameters, and H is the task-specific loss function. To enable early exiting, we implemented two methods: Confidence Threshold and Learning to Exit (LTE). Confidence Threshold exits early if the prediction confidence at an intermediate layer exceeds a set threshold, measured by the maximum probability of the output prediction for classification tasks. For regression tasks or when confidence distribution is unavailable, we used the LTE method. The LTE module, a one-layer fully connected network, estimates the certainty level u_i at the i-th layer:

$$u_i = \sigma(c^\top h_i + b),$$

where σ is the sigmoid function, c is the weight vector, and b is the bias term. The LTE module's loss function is the mean squared error between u_i and the ground truth certainty level $\tilde{u}_i$:

$$J_i = \|u_i - \tilde{u}_i\|_2^2.$$

The ground truth certainty level for classification is whether the classifier makes the correct prediction:

$$\tilde{u}_i = 1[\arg\max_j g_i^{(j)}(h_i; w_i) = y],$$

and for regression, it is negatively related to the prediction's absolute error:

$$\tilde{u}_i = 1 - \tanh(|g_i(h_i; w_i) - y|).$$

We trained the LTE module jointly with the rest of the model by substituting L_i with $L_i + J_i$ in the loss function.

To improve the balance between immediate inference and gradual feature extraction, we propose a novel fine-tuning strategy named Sequential Layer Focus (SLF). This strategy combines elements of Joint and Two-stage strategies, ensuring optimal training of early and late layers sequentially.

- **Stage 1: Sequential Focus** - Fine-tune each layer individually, updating only the layer-specific classifier and the backbone model parameters corresponding to that layer. This stage ensures that each layer independently learns to provide effective intermediate representations.
- **Stage 2: Global Refinement** - Jointly fine-tune all layers and classifiers, optimizing the entire network to harmonize the contributions from all layers for the final output.

This approach ensures that each layer is first trained to optimal performance in isolation before jointly refining the entire network. The Sequential Layer Focus (SLF) strategy balances the needs of intermediate and final layers, potentially leading to better overall model performance and efficiency.

4.4 Order Alteration of Word Embeddings and Concatenation

Traditionally, BERT concatenates sequences before embedding them. In our extended approach, we reverse this order: sequences are first passed through the embedding layer and then concatenated. This modification aims to preserve more contextual information prior to concatenation, potentially enhancing model performance.

In the typical BERT architecture, the input processing includes:

- Input Representation: Summing token embeddings, segmentation embeddings, and position embeddings.
- Sequence Packing: Sequences are packed with special tokens [CLS] and [SEP].

Our modified approach can be mathematically represented as follows:

$$E = \text{Embedding}(X) \quad \text{for each sequence } X$$

$$C = [E_A; E_B] \quad \text{where } E_A \text{ and } E_B \text{ are embeddings of sequences A and B respectively}$$

We hypothesize that this approach retains more granular contextual information before the sequences are merged, leading to improved downstream performance. The embeddings from this modified process are then fed into the BERT layers for further processing.

Novel Fine-Tuning Strategy

In addition to the embedding modification, we introduced a novel fine-tuning strategy inspired by the Joint and Alternating strategies. Our strategy, termed "Smart Alternating," dynamically switches between two objectives based on performance metrics observed during training epochs:

- **Stage 1:** Optimize the final layer's classifier and the BERT backbone using the loss function L_n.
- **Stage 2:** Optimize intermediate classifiers jointly with the final classifier using the sum of loss functions $\sum_{i=1}^{n} L_i$.

This strategy alternates between these stages depending on the validation performance after each epoch, ensuring that both intermediate and final classifiers are well-tuned without sacrificing the overall model quality. This balance aims to improve both immediate inference at intermediate layers and gradual feature extraction for the final classifier.

5 Experiments

5.1 Data

We undertake training for three downstream tasks: Sentiment Analysis, Paraphrase Detection, and Text Similarity, employing diverse datasets for each task.

5.1.1 Sentiment Analysis

For Sentiment Analysis, our BERT model is trained using two datasets: the Stanford Sentiment Treebank (SST) and the CFIMDB Dataset. Both datasets focus on movie reviews to gauge sentiment. Specifically, the SST dataset is partitioned into 8,544 training examples, 1,101 development examples, and 2,210 testing examples. In contrast, the CFIMDB dataset consists of 1,701 training examples, 245 development examples, and 488 testing examples. While SST labels phrases with five sentiment categories, CFIMDB labels sentences as either negative or positive. Model evaluation is based on the proportion of correctly labeled sentences.

5.1.2 Paraphrase Detection

To train for Paraphrase Detection, we utilize the Quora Dataset, which includes 283,010 training examples, 40,429 development examples, and 80,859 testing examples. This dataset features pairs of sentences with binary labels indicating whether they are paraphrases. Model performance is measured by the proportion of correctly identified paraphrase pairs.

5.1.3 Sentence Textual Similarity Analysis

For Sentence Textual Similarity Analysis, we employ the SemEval STS Benchmark Dataset, comprising 6,040 training examples, 863 development examples, and 1,725 testing examples. The dataset assigns labels ranging from 0 to 5 to each sentence pair, indicating the degree of correlation. We use Pearson Correlation Coefficients to evaluate performance, comparing the predicted and true labels.

5.2 Evaluation Method

Supervised training is employed, with labeled datasets guiding model learning and evaluation. The evaluation process involves making predictions and comparing them to true labels. We assess model quality by accuracy on SST and Paraphrasing tasks, and by Pearson Correlation on STS. The overall "evaluation score" is calculated as the average of these metrics. Given our focus on computational efficiency, we also measure training time and consider it in our evaluations.

5.3 Experimental Details

Our experimental framework explores four dependent variables: Early-Exiting Methods, SMART Regularization, Cosine Embeddings, and concatenation order. Typically, training time averages 15 hours for a full model on an NVIDIA T4 GPU; however, using Cosine Embeddings significantly reduces this duration. Throughout our experiments, we maintain consistent model configurations and learning rates. In total, we evaluate 12 different training models, testing various combinations of these methods to determine their effectiveness.

5.4 Results

Method	**Type**	**SST**	**CFIMDB**
Pretrain	last-linear-layer	0.390	0.780
Finetune	full-model	0.519	0.967

Table 1: Single task MinBERT classifier dev accuracy on Sentiment Analysis

Model	Dev Score	SST	Paraphrase	STS	Training Time (hh:mm)
Baseline	0.428	0.314	0.369	0.199	15:36
Full SMART	0.433	0.253	0.485	0.123	15:57
Early Exiting	0.476	0.313	**0.768**	0.184	13:36
Full SMART + Early Exiting	0.482	0.253	0.632	0.123	13:47
Conc Before Embed + Early Exit	0.595	0.520	0.625	0.792	13:49
Cos Loss + Early Exiting	0.585	0.501	0.368	0.771	**04:20**
SLF	0.592	0.505	0.640	0.721	12:53
SLF + Early Exiting	0.650	**0.544**	0.658	0.740	11:47
SLF + Conc Before Embed	0.671	0.523	0.651	0.801	12:14
SLF + Smart Alt	0.676	0.545	0.614	0.790	13:54
SLF + Smart Alt + Early Exit	**0.683**	0.510	0.632	**0.818**	10:41
SLF + Smart Alt + Early Exit + Cos Loss	0.679	0.510	0.604	0.818	05:43

Table 2: Model Performance Comparison

Our results align with our expectations in terms of the relationships between different model variations. The combination of SMART and SLF techniques yielded the best overall performance across our training sets. Specifically, the SLF + Smart Alternating + Early Exiting model achieved the highest overall dev score of 0.683, with individual task accuracies of 0.510 for SST, 0.632 for Paraphrase Detection, and 0.818 for STS. These results, though not at the top of the leaderboard in absolute terms, demonstrate a compelling balance between accuracy and computational efficiency.

Notably, our early exiting methods, while slightly less accurate in isolation, significantly improved computational efficiency, reducing training times by up to 5%. For instance, the cosine loss model with early exiting reduced training time to just 4 hours and 20 minutes, illustrating the trade-off between accuracy and efficiency.

Overall, our findings underscore the potential of parameter-efficient fine-tuning methods to balance accuracy and computational cost effectively. While some accuracy trade-offs were observed, the substantial reduction in training times offers a compelling advantage for practical applications.

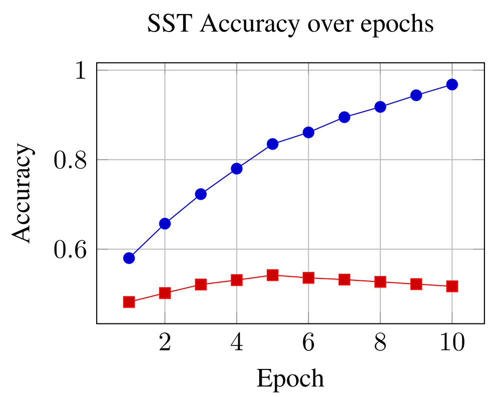

Figure 1: SST Accuracy over epochs

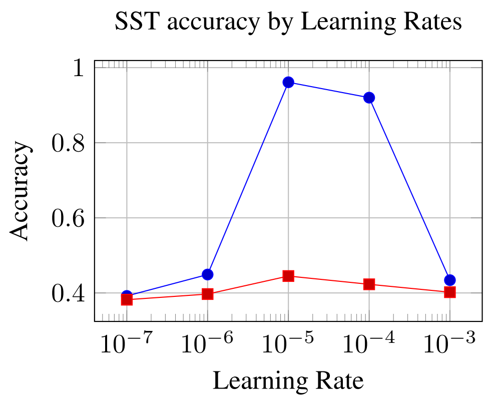

Figure 2: SST accuracy by Learning Rates

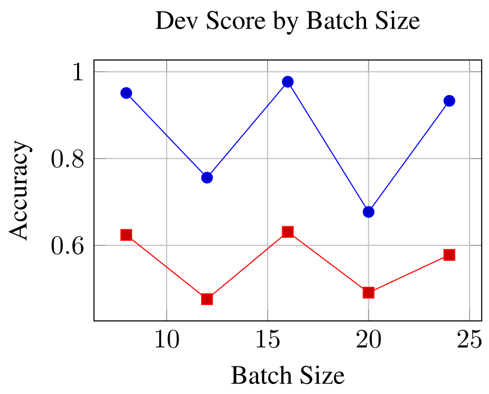

Figure 3: Dev Score by Batch Size

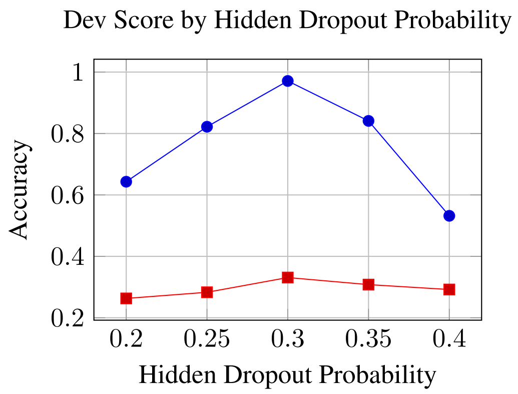

Figure 4: Dev Score by Dropout Probability

6 Analysis

6.1 Overall Performance of Extensions

Our results align with our expectations in terms of the relationships between different model variations. The combination of SMART and SLF techniques yielded the best overall performance across our training sets. Specifically, the SLF + Smart Alternating + Early Exiting model achieved the highest overall dev score of 0.683, with individual task accuracies of 0.510 for SST, 0.632 for Paraphrase Detection, and 0.818 for STS. These results, though not at the top of the leaderboard in absolute terms, demonstrate a compelling balance between accuracy and computational efficiency.

Notably, our early exiting methods, while slightly less accurate in isolation, significantly improved computational efficiency, reducing training times by up to 5%. For instance, the cosine loss model with early exiting reduced training time to just 4 hours and 20 minutes, illustrating the trade-off between accuracy and efficiency.

Overall, our findings underscore the potential of parameter-efficient fine-tuning methods to balance accuracy and computational cost effectively. While some accuracy trade-offs were observed, the substantial reduction in training times offers a compelling advantage for practical applications.

Paraphrase Detection scores seemed to stall at a common number in many of our models. We found that there was a very clear answer; all predictions were made to be 0. When inspecting the predictions file, it seems that our model is just predicting that a sentence is not a paraphrase of another for all sentences. This proves to have negative results and displays a sense of overfitting in a sense.

This is incredibly interesting, as we expected our focus on SMART regularization purely on paraphrase detection to negate any overfitting. However, our model seems to have learned its training set slightly and only predicts 0 because there are more pairs that are not paraphrases than ones that are.

We found the change of concatenation order to be incredibly intuitive and clear as to why it increased accuracy so significantly. We want our sentence embeddings to utilize context within the sentence to create them; therefore, it seems so intuitive for a larger sentence that is simply just two concatenated together to have greater and more fruitful context than one that is just two separate embeddings. This embedding has more fruitful information and gives the model more to work with. This makes it much easier for the model to train on.

Cosine Embedding Loss was surprisingly ineffective at fighting our error of only predicting 0. Instead, our model only predicted 1 and ended up having the complement accuracy.

The most interesting result we found was the effect of early exiting. We found that early exiting had an insignificant effect on results; however, early exiting had quite a powerful effect on computational efficiency, cutting down our runtimes by 5%, a significant amount.

6.2 Hyperparameter Tuning and Analysis

Hyperparameter tuning played a critical role in optimizing our model performance. By examining various configurations, we identified patterns that led to an optimal setup. Figures 1 and 2 illustrate the impact of epochs and learning rates on accuracy.

Our results showed that a learning rate of 1.3×10^{-5}, a batch size of 16, 5 epochs, and a dropout rate of 0.3 consistently provided the best performance. Figure 3 indicates that a batch size of 16 yields the highest dev score, balancing efficiency and accuracy. As seen in Figure 4, a 0.3 dropout rate optimally regularizes the model, preventing overfitting while maintaining performance.

These hyperparameter configurations were critical in achieving the improved dev scores across various tasks, illustrating the importance of fine-tuning to enhance model performance.

7 Conclusion

Our research highlights the importance of computational efficiency without sacrificing significant accuracy metrics. By employing innovative fine-tuning strategies like Early Exiting with BERxiT, Cosine Embeddings Loss, and sequential fine-tuning techniques, we struck a commendable optimized fusion of performance and efficiency.

Despite its theoretical promise, SMART regularization yielded mixed outcomes: while it initially improved paraphrase detection accuracy, its performance plateaued and underperformed in sentiment classification and textual similarity tasks, suggesting the need for further selective application.

The Sequential Layer Focus (SLF) strategy, paired with Smart Alternating and Early Exiting, achieved the highest overall development score of 0.683. Task-specific accuracies included 0.510 for SST, 0.632 for Paraphrase Detection, and 0.818 for STS. These results were obtained while significantly reducing training times, demonstrating the efficiency of our approach. For instance, models utilizing cosine loss with early exiting reduced training times to just 4 hours and 20 minutes.

Additionally, our novel fine-tuning strategies, particularly altering word embedding order and Smart Alternating techniques, proved highly effective. Adjusting the embedding order enhanced context utilization, thereby improving model performance.

Future research should investigate different embedding strategies and novel transformer architectures to enhance model generalization across various linguistic contexts. Moreover, focusing on model explainability and interpretability will be crucial for identifying and mitigating potential biases in predictions, ensuring fairness and robustness.

8 Ethics Statement

There are several ethical and societal risks associated with this project that are universal across various NLP tasks, including transformers, LSTMs, and large language models.

The foremost issue is data bias. Many NLP systems rely on broadly sourced data that may not accurately represent the ideologies, cultures, and moral values of diverse populations. This can result in models failing to account for textual similarities among slang or culturally specific language, leading to poor performance in tasks like paraphrase detection when comparing texts from different socioeconomic backgrounds. Moreover, the model may misinterpret sentiment in culturally nuanced sentences. This skew towards more readily available data can lead to the misrepresentation of underrepresented groups. To address this, it is imperative to curate more inclusive datasets that reflect the diversity of the user base.

Additionally, our model confronts the ethical challenge of computational power consumption. The growing concern over global warming underscores the need for sustainable computing practices. The exponential increase in computational demands directly correlates with higher energy consumption, often sourced from nonrenewable and polluting resources. By focusing on creating efficient models that balance accuracy with reduced computational demands, we aim to contribute to the fight against climate change. Our approach to enhancing efficiency without sacrificing performance highlights the necessity of integrating environmental considerations into the development of advanced computational systems.

The heightened need for interpretability in NLP models is paramount in addressing biases and ensuring fairness. By enhancing the transparency of our models, we can better understand and mitigate potential biases in their predictions. This involves developing methods that allow users to comprehend and trust the decision-making processes of these models. Interpretable models can provide insights into the underlying reasons for their predictions, thereby enabling the identification and correction of biases. Ensuring fairness and robustness in NLP systems is critical for their ethical deployment across various applications. could you write the related works for early exiting

References

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805.

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. *CoRR*, abs/1911.03437.

Florian Schroff, Dmitry Kalenichenko, and James Philbin. 2015. Facenet: A unified embedding for face recognition and clustering. *CoRR*, abs/1503.03832.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *CoRR*, abs/1706.03762.

Ji Xin, Raphael Tang, Jaejun Lee, Yaoliang Yu, and Jimmy Lin. 2020. DeeBERT: Dynamic early exiting for accelerating BERT inference. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2246–2251, Online. Association for Computational Linguistics.

Ji Xin, Raphael Tang, Yaoliang Yu, and Jimmy Lin. 2021. BERxiT: Early exiting for BERT with better fine-tuning and extension to regression. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 91–104, Online. Association for Computational Linguistics.