Enhancing Dev Accuracy with DoRA

Stanford CS224N Default Project

Hayden Kim
Department of Mathematics
& Division of Literatures, Cultures & Languages
Stanford University
`turtle24@stanford.edu`

Nilson Rodriguez Cadenas
Department of Mathematics and Physics
Stanford University
`nilsoon3@stanford.edu`

Abstract

In February, 2024, NVIDIA introduced DoRA, an improved version of LoRA that matches the level of accuracy of Full Fine-tuning (FT) and yet is still computationally inexpensive. Our final project verifies the results released by NVIDIA. We first built a simple version of BERT (minBERT) and computed the baseline accuracy values for models that differ by the fine-tuning methods (FT and Last Linear Layer (LLL)) and also by the datasets that the CS224N teaching team provided for three NLP tasks (sentiment analysis, paraphrase detection, and semantic textual similarity). We then incorporated LoRA and DoRA into our minBERT. We compared the baseline values from "Last Linear Layer" model with those from "Last Linear Layer + LoRA" and "Last Linear Layer+DoRA" models. Our results indicated that the learning rate of 1e-5 tends to result in better accuracy/correlation values than 1e-3. Our results did not indicate significant improvement by LoRA. Our results returned significant improvement by DoRA in contrast to the basic LLL fine-tuning model and LLL+LoRA model. The degree of improvement by DoRA differed from task to task. It improved the performance for semantic textual similarity task significantly while not resutling in improvement for sentiment analysis task. Finally, whether DoRA matched the accuracy / correlation level of FT differed base don the learning rate. For 1e-3, DoRA's performance matched that of FT, but for 1e-5, FT outperformed DoRA.

1 Introduction

BERT (Bidirectional Encoder Representations from Transformers) is a pre-trained Natural Language Processing model that uses a transformer architecture to generate contextualized word embeddings Devlin et al. (2019). PEFT (Parameter Efficient Fine-Tuning) improves BERT's performance while only incorporating a small subset of parameters. Xu et al. (2023). LoRA, a popular PEFT method, employs low-rank matrices for weight approximation to effectively minimize computational overhead. However, LoRA has not been able to match the level of accuracy of Full Fine-Tuning (FT), a computationally expensive option that employs every model parameter for training. yang Liu et al. (2024) Hence, NVIDIA introduced another PEFT method called DoRA (Weight-Decomposed Low-Rank Adaptation) yang Liu et al. (2024) . Unlike LoRA, DoRA decomposes the pretrained weight matrix into magnitude and direction matrices. NVIDIA's paper revealed that the accuracy from DoRA far exceeds the accuracy from LoRA. yang Liu et al. (2024) DoRA also returns an accuracy value that is closer to that of FT without incorporating every parameter, thereby showcasing its capacity to be a cost-effective alternative to FT. yang Liu et al. (2024) Since the paper was released 3 months ago, there are limited studies that verify their results.

Hence, for our project, we verified if DoRA exceeds the performance of LoRA and match the accuracy level of FT. The goals of our final project were 1) implement a simplified version of BERT (miniBERT), 2) fine-tune it with LoRA and DoRa into our model, 3) compare the performances for sentiment analysis, paraphrase detection, and semantic textual similarity among minBERT with last linear layer (LLL) fine-tuning, minBERT with FT, minBERT with LLL + LoRA, and minBERT with

LLL + DoRA. Throughout our modeling process, we experimented with various levels of learning rates, numbers of linear layers, sizes of input tensors for linear layers, types of activation functions, and types of loss functions so that we could find an optimal setup for the tasks that our model aim to achieve.

2 Related Work

Many applications in natural language processing (NLP) rely on adapting a single large-scale, pre-trained language model for multiple downstream tasks. This adaptation is typically achieved through fine-tuning, which involves updating all the parameters of the pre-trained model. However, a major drawback of fine-tuning is that the resulting model has as many parameters as the original one. To address this, researchers have tried adapting only some of the parameters or learning external modules for new tasks. This approach allows only a small number of task-specific parameters to be stored and loaded alongside the pre-trained model for each task, greatly enhancing operational efficiency during deployment. Due to this problem, many researchers are looking for solutions to improve the efficiency of models while reducing the computational cost.

Our project is a continuation of the effort to balance the tradeoff between increasing efficiency and minimizing computational cost. Among various research works that have already investigated this issue, two of the most critical related works to our project were "LoRA: Low-Rank Adaptation of Large Language Models" by J.E.Hu et al. and "DoRA: Weight-Decomposed Low-Rank Adaptation" by Shih-yang Liu et al.. For the technical details, please refer to our approach section. Here, we will address the general outline of what they attempted to improve the performance of fine-tuning methods.

For "LoRA: Low-Rank Adaptation of Large Language Models", the researchers wanted to decrease the number of parameters involved in the fine-tuning method to reduce the computational cost. Thus, they developed a fine-tuning method that first freezes the weights from the pre-trained model and then incorporates matrices decomposed by rank into the layers of the model. The LoRA method offers no additional inference latency which means that the breakdown of the weight matrix is not computationally expensive. A limitation of the LoRA method is that if the matrix decomposition uses too low of a rank, this could cause over-fitting of the tasks and reduce performance. Signs of this effect can be seen in the results section of this paper. While LoRA may improve the cost of computation, the lower ranks can also affect the accuracy of the model due to lowering its ability to capture various nuances within the training data and inputs. This led researchers to keep working on better ways to adapt LoRA to decrease the computational cost of models while still maintaining a high accuracy.

For "DoRA: Weight-Decomposed Low Rank Adaptation", the research team wanted to improve LoRA, as there was criticism that LoRA did not achieve the same level of accuracy compared to FT. Thus, further work was conducted to build on LoRA in order to reduce the computational cost of high parameter models. Working off of LoRA, these researchers noticed that LoRA would have a significant accuracy gap between its scores and the scores of full fine-tuning. Using a more complex matrix decomposition, the Liu and their team proposed to break down weight matrices into direction and magnitude matrix products. This would allow the implementation of LoRA on one of these matrices in order to reduce the rank while training the other matrix for its magnitude. This allowed for greater learning capacity while still avoiding any additional inference overhead. yang Liu et al. (2024)

Our work builds on these foundational methods by comparing LoRA and DoRA within a simplified BERT model, assessing their performance in tasks like sentiment analysis, paraphrase detection, and semantic textual similarity.

3 Approach

3.1 Basic Implementation:

We implemented the Multi-head Self-attention layer, the Transformer layer, the BertSentimentClassifer class for Sentiment Analysis, and the step function for the Adam Optimizer for MinBERT based on the starter code provided by the CS224N teaching team. We ran the evaluation for the Stanford

Sentiment Treebank (SST) and the CFIMBD dataset with our single task classifier.py. The results we got was as follows:

Method	Type	SST	CFIMDB
Full Fine Tuning	Single Task	0.515	0.967
Last LinearLayer	Single Task	0.416	0.788

Table 1: Single Task Sentiment Analysis Accuracy Values

3.2 Baselines:

In the project proposal, we stated that we planned to use the dev accuracy/Pearson correlation values from our single task basic MinBERT model implemented with $classifier.py$ as our baselines. But since our goal was to observe the improvement in the performance from the model with only the Last Linear Layer fine-tuning to the models with other fine-tuning methods across various tasks, we decided to use the test values from the MinBert model implemented with $multitask\_classifier.py$ with LLL fine-tuning. The baseline values can be found in the first two rows of the table in the "Results" section.

3.3 Extension (multitask_classifier.py, LoRA, and DoRA):

For extension, we first implemented $multitask\_classifier.py$ and $train\_multitask.py$, which creates a minBERT model that does sentiment analysis task for Stanford Sentiment Treebank(SST) dataset, paraphrase detection task with Quora dataset, and semantic textual similarity with SemEval STS dataset. We then experimented with different processing methods within each of the functions: $predict\_sentiment()$, $predict\_paraphrase()$, and $predict\_similarity()$. We experimented with including various number of linear layers, changing the tensor size for the input for the linear layers, and adding non-linear activation functions such as ReLU.

We then implemented $LoRA.py$ and $DoRA.py$ based on our understanding of the LoRA and DoRA methods from two papers we discussed in the "Related Work" session. One challenge we faced was that while the LoRA research team provided their source code, the DoRA research team did not – which was possibly due to the fact that the research paper was released a couple months ago. Hence, we relied on the LoRA and DoRA implementation code written by Sebastian Raschka, an assistant professor of Statistics at the University of Wisconsin-Madison, which has MIT license and is available here: https://github.com/rasbt/dora-from-scratch/blob/main/Using-LinearDoRA.ipynb. The mechanism of LoRA and DoRa as well as our implementation for LoRA and DoRA are as follows:

LoRA: Depending on the model, updating all model weights during training can be expensive due to GPU memory limitations. Suppose we have a large weight matrix W for a given layer. During backpropagation, we learn a ΔW matrix, which contains information on how much we want to update the original weights to minimize the loss function during training. In training and fine-tuning, the updated matrix can be defined as $W_{new} = W_{old} + \Delta W$. The LoRA method breaks down ΔW to two matrices A and B, turning the equation into $W_{new} = W_{old} + AB$. If a pretrained weight matrix W is a 2000 $\times$ 2000 matrix, then the weight update matrix ΔW in regular finetuning is also a 2000 $\times$ 2000 matrix. In this scenario, ΔW consists of 4,000,000 parameters. However, if we use a LoRA rank of 3, then A is a 2000 $\times$ 3 matrix, and B is a 3 $\times$ 2000 matrix. This means we only need to update $3 \times 3 \times 2000 = 18,000$ parameters when using LoRA. With this in mind, we created a $LoRA.py$ file which handled any linear layer we give it as input and separate it into corresponding A and B matrices. We then created $bertLoRA.py$ and $multitask\_classifier\_LoRA.py$ files, which are $Bert.py$ and $multitask\_classifier.py$ but with previous linear layers replaced with these linear layers replaced with layers based on $LoRA.py$. We then $multitask\_classifier\_LoRA.py$ to observe performance of the model and produce accuracy/correlation values.

DoRA: DoRA can be understood through two main steps. First, a pretrained weight matrix is decomposed into a magnitude vector (m) and a directional matrix (V). Second, LoRA is applied to the directional matrix V, while the magnitude vector m is trained separately.

The decomposition into magnitude and directional components is based on the mathematical concept that any vector can be represented as the product of its magnitude (a scalar indicating its length) and its direction (a unit vector indicating its orientation). In DoRA, this principle is extended to an entire pretrained weight matrix W. Each column of the weight matrix, which represents the weights connecting all inputs to a particular output neuron, is decomposed into its magnitude and direction. As a result, decomposing W yields a magnitude vector m, which represents the scale or length of each column vector in the weight matrix. Subsequently, DoRA applies standard LoRA to the directional matrix V. This is expressed as:

$$W' = m\left(\frac{V + \Delta V}{\text{norm}}\right) = m\left(\frac{W + AB}{\text{norm}}\right)$$

The normalization, abbreviated as "norm" for simplicity, follows the weight normalization method proposed in the 2016 paper by Salimans and Kingma, Salimans and Kingma (2016).

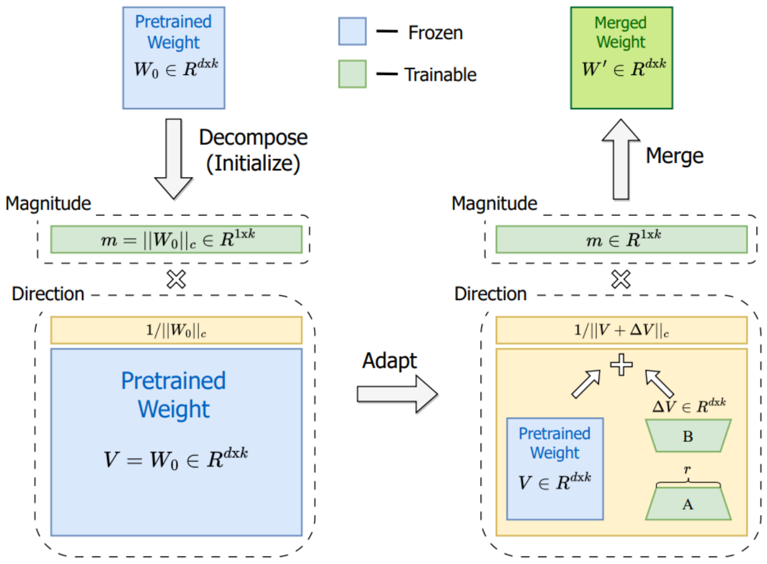

Figure 1: An overview of our proposed DoRA, which decomposes the pre-trained weight into magnitude and direction components for fine-tuning, especially with LoRA to efficiently update the direction component. yang Liu et al. (2024)

Just like what we did with LoRA, we created $DoRA.py$, $bertDoRA.py$, and $multitask\_classifier\_DoRA.py$ to observe the model's performance and produce accuracy/correlation values for sentiment analysis, paraphrase detection, and semantic textual similarity tasks.

4 Experiments

4.1 Data

For Sentiment Analysis, we used the Stanford Sentiment Treebank (SST) and the CFIMBD dataset. The Stanford Sentiment Treebank (SST) dataset contains 215,154 sentence fragments that are collected from movie reviews. Each fragment is categorized by 5 sentiment labels: 'positive', 'somewhat positive', 'neutral', 'somewhat negative', and 'negative.' We utilized the given buckets: 'train' with 8544 segments, 'dev' with 1101 segments, and 'test' with 2210 segments. For the CFIMBD dataset, which contains 2434 movie reviews that are classified by 'negative' and 'positive' sentiment labels, we depended on the provided buckets: 1701 movie reviews for 'train', 245 movie reviews for 'dev', and 488 movie reviews for 'test'. For Paraphrase Detection, we used the Quora dataset, which contains 404,298 question pairs with labels indicating whether they are paraphrases. We depended on the following categories: 283,010 samples for 'train', 40,429 samples for 'dev', and 80,859 examples for 'test'. Finally, for Semantic Textual Similarity, we used the SemEval STS

Benchmark dataset. Regarding the SemEval STS Benchmark dataset, consisting of 8628 sentence pairs with varying similarity scores from 0 to 5, we employed the provided splits: 6,040 samples for 'train', 863 samples for 'dev', and 1,725 samples for 'test'. One important note about the datasets for sentiment analysis is that we did not incorporate the CFIMBD dataset into our evaluation with $multitask_classifier.py$. This decision was made because 1) CFIMBD dataset returned unusually high values for single task classifier.py, which signified that the dataset might be too similar to what orignal BERT model is pre-trained with and 2) we wanted to be consistent with the number of datasets we use for each task.

4.2 Evaluation method

The evaluation metrics in $evaluation.py$ were crucial for measuring the performance of our multitask BERT model across various natural language processing tasks: sentiment classification, paraphrase detection, and semantic textual similarity (STS). For sentiment classification, we used accuracy and F1 score. Accuracy indicates the overall correctness of predictions, while the F1 score balances precision and recall, offering a nuanced view that accounts for both false positives and false negatives. This combination ensured a comprehensive evaluation, especially on unbalanced datasets.

For paraphrase detection, accuracy was used to measure the percentage of correctly identified paraphrases, which is essential for tasks like question answering and information retrieval. Semantic textual similarity (STS) was evaluated with the Pearson correlation coefficient, which assessed the linear relationship between predicted and true similarity scores, ranging from -1 to 1. This metric is vital for tasks such as text summarization and plagiarism detection. The evaluation process involves switching the model to evaluation mode with $model.eval()$, ensuring consistent performance measurements by disabling randomness. These metrics, standard in the NLP field, provided a thorough assessment of the model's strengths and weaknesses, guiding further improvements and optimizations.

4.3 Experimental details

We mostly relied on the default model configuration provided by the CS224N teaching team (Table 1). The first difference our models had was the learning rate, as our experimentation revealed that 1e-5 resulted in a better performance than 1e-3. The second difference involved the role that rank and the value of alpha played within the LoRA component of the models. Rank determines the size of the decomposed matrices. Alpha acts as a scaling factor for the low-rank matrices. We ran the models with rank & alpha pairs of (4,8) and (8,16) to diversify the ranks and alphas just like the original paper did. For training time, it took roughly 4 hour in total for 6 different types of runs that resulted in the values listed in table 2. Each run differed by the dataset it incorporates (SST, CFIMDB, Quora, or STS) or the fine-tune method (FM/LLL / LLL+LoRA LLL+DoRA).

Vocab Size	**Hidden Size**	**# of Hidden Layers**	**# of Attention Heads**	**Weight Decay**
30522	768	12	12	0.0
Learning Rate	**Batch Size**	**# of Train Epochs**	**Dropout Prob. (Hidden)**	**Drop Out Prob. (Attention)**
1e-3 & 1e-5	8	10	0.1	0.1

Table 2: Model Hyperparameters

4.4 Hypotheses:

- **Last Linear Layer (LLL)**: We anticipated that the LLL fine-tuning method would yield reasonable performance but not match the accuracy of full fine-tuning. This is because LLL fine-tuning only updates the last layer's weights, which limits the model's ability to adapt fully to the new tasks.
- **LoRA**: For LoRA, we expected an improvement over LLL due to its ability to update low-rank matrices that approximate the weight updates needed. However, we anticipated that LoRA might still fall short of full fine-tuning due to the reduced parameter updates.
- **DoRA**: With DoRA, we expected its performance to exceed LoRA, as DoRA's decomposition approach is more rigorous than LoRA in terms of having more ways to be trained and learn. We also expected that DoRA's performance would be similar to the results from Full Fine-tuning model, as indicated by NVIDIA's research.

4.5 Results

The tables below summarize the results of our evaluation across different configurations and tasks, providing insights into the model's performance. The last row corresponds to the values we submitted for the test leaderboard.

	Model	lr	SST	Paraphrase	STS	Rank	Alpha
Baseline	LLL	1e-3	0.264	0.632	0.351	N/A	N/A
	LLL+LoRA	1e-3	0.15	0.368	0.193	8	16
	LLL+LoRA	1e-3	0.262	0.632	0.338	4	8
	LLL+DoRA	1e-3	0.264	0.649	0.376	8	16
	LLL+DoRA	1e-3	0.264	0.624	0.366	4	8
	Full Fine-Tuning	1e-3	0.264	0.652	0.40	N/A	N/A
Baseline	LLL	1e-5	0.263	0.633	0.345	N/A	N/A
	LLL+LoRA	1e-5	0.262	0.632	0.329	8	16
	LLL+LoRA	1e-5	0.262	0.632	0.305	4	8
	LLL+DoRA	1e-5	0.264	0.655	0.424	8	16
	LLL+DoRA	1e-5	0.264	0.655	0.406	4	8
Test Leaderboard	Full Fine-Tuning	1e-5	0.431	0.806	0.857	N/A	N/A

Table 3: A Table with Test Accuracy/Correlation Values for SST, Paraphrase, and STS Tasks with their Corresponding Learning Rates. Rank & Alpha values are only used for LoRA and DoRA models.

5 Analysis (Quantitative & Qualitative):

In terms of the model's overall capacity to handle the SST, paraphrase, and STS tasks, we observed 0.431, 0.806, and 0.857 as the final values from our full fine-tuning model. This result signals that our implementation of BERT has the capacity to reach a fair performance in the three tasks.

5.1 Learning Rate

Based on our experimentation with learning rates, we observed that 1e-5 tend to result in better accuracy/correlation values. Hence, we decided to use both 1e-3 (default learning rate suggested by the CS224N teaching team) and 1e-5 for our experiments. The table below suggests that 1e-5 indeed returns better values for all three tasks.

lr	SST	Paraphrase	STS
1e-3	0.252	0.584	0.352
1e-5	0.347	0.725	0.625

Table 4:Average Scores by Learning Rate

5.2 LLL v. LLL+LoRA v. LLL+DoRA

Model Type	SST	Paraphrase	STS
LLL	0.264	0.632	0.351
LLL+LoRA	0.262	0.632	0.338
LLL+DoRA	0.264	0.649	0.366

Table 5: Highest Acc/Corr Values

Model Type	SST	Paraphrase	STS
LLL	0.263	0.632	0.345
LLL+LoRA	0.262	0.632	0.329
LLL+DoRA	0.264	0.655	0.406

Table 6:Highest Acc/Corr Values

The performance of each model differed from task to task. For sentiment analysis, we saw no significant increase in the performance, as the values tended to remain in the 0.262-0.264 range no matter what the learning rate was. For paraphrase detection task, we did not see any improvement from

LLL to LLL+LoRA, with both of them returning values within the 0.632-0.633 for both of the models for both learning rate. However, we observed certain improvement from LLL to LLL+DoRA. For learning rate 1e-3, the number increased from 0.632 for LLL to 0.649 for LLL+DoRA. For learning rate 1e-5, we observed an increase from 0.632 to 0.655. The task we saw the most improvement was STS. While the LLL+LoRA resulted in lower values than STS for both 1e-3 and 1e-5, we observed an improvement in LLL+DoRA, as the number increased from 0.351 to 0.366 for 1e-3 and 0.345 to 0.406 for 1e-5. Hence, our results revealed that 1) DoRA seems to improve performance than LoRA and LLL, 2) there is no noticeable improvement made by LoRA, 3) learning rate seems to alter how much improvement LoRA and DoRA could make, and 4) the amount of improvement that DoRA could result in differs from task to task. Our results necessitate for additional research on what type of model architecture, such as hyperparameters and number of layers, are required to best optimize DoRA's performance as well as what type of NLP tasks DoRA tends to perform better.

5.3 LLL+DoRA v. Full Fine-Tuning

Model Type	SST	Paraphrase	STS
LLL+DoRA	0.264	0.649	0.376
Full Fine-Tuning	0.264	0.652	0.40

Table 7: Acc/Corr for LLL+DoRA and Full Fine-Tuning with learning rate 1e-3

Model Type	SST	Paraphrase	STS
LLL+DoRA	0.264	0.655	0.424
Full Fine-Tuning	0.431	0.806	0.857

Table 8: Acc/Corr for LLL+DoRA and Full Fine-Tuning with learning rate 1e-5

For learning rate 1e-3, LLL+DoRA resulted in similar values with Full Fine-tuning method, as it returned sentiment analysis accuracy value of 0.264 (FT returned 0.264), paraphrase detection value of 0.649 (FT: 0.652), and semantic textual analysis value of 0.376 (FT: 0.40). But for learning rate 1e-5, LLL+DoRA did not match the accuracy value of Full fine-tuning, as table 6 suggests. This result suggests that when we are running models with learning rate that is not optimal for FT, we might get results where DoRA matches the level of FT. But our result also suggests that once we find a learning rate that is optimal for FT, it is also possible for the FT to outperform DoRA. Our learning rates, 1e-3 and 1e-5, were different from the ones used by the ones used by the original DoRA paper (1e-4,2e-4, and 3e-4), so further reserach is needed to investigate why DoRA tends to match FT only in certain learning rates.

5.4 Challenges

The model took many different forms within its multitask classifier function, train multitask function, and evaluation functions. During our experiments, we had to implement several functionalities in our multitask training function to process the data effectively. These included incorporating loss functions such as cross-entropy and mean squared error (MSE). Each task—sentiment analysis, paraphrase detection, and semantic textual similarity (STS)—required different loss functions to optimize the model's performance. For instance, cross-entropy loss was essential for classification tasks like sentiment analysis and paraphrase detection, while MSE was crucial for the regression task of STS.

Additionally, changes in the methods within the classifier often led to various challenges. When transitioning from one fine-tuning method to another (e.g., from full model fine-tuning to last linear layer (LLL) fine-tuning, or from LLL to LoRA and DoRA), the model sometimes encountered errors based on how it processed the predictions. These errors ranged from issues with gradient updates to discrepancies in prediction formats. For example, while trying to improve paraphrase and similarity accuracies, we tried a different method of concatenating the results from the $forward()$ function and then feed it through linear layers, but this would double the size of the tensor and thus lead to debugging.

6 Conclusion, Future Work, & Achievements

The main findings of our project are the following. First, learning rate 1e-5 seems to be more optimal for our model than learning rate 1e-3, which was the default learning rate for CS224N default projects. Second, LoRA did not improve performance in regard to LLL, meaning that there needs to be more investigation into the optimal hyperparameters and configurations for LoRA as potential future work. Third, DoRA tends to improve the performance in regard to LLL, but the degree of improvement varies from task to task, as we noticed almost no difference in SST task and a significant difference in STS task. More investigation into the correlation between DoRA's performance and NLP tasks could be an excellent potential future work. Fourth, whether DoRA matches the performance of Full Fine-tuning depends on the learning rate. One potential future work for the fourth finding is to investigate whether this result is representative of a general pattern between DoRA and FT, as there are not many research papers available for DoRA due to the very recent release of DoRA in early 2024. From our project, we gained significant amount of knowledge in Transformer architecture as well as the mathematics behind the fine-tuning process. By implementing LoRA and DoRA ourselves, adjusting the number of linear layers, choosing activation and loss functions, and altering the sizes of the tensor input for the layers, we gained significant amount of experience in implementing and designing deep neural networks.

7 Ethics Statement

The ethical implications of training and deploying machine learning models are significant and sometimes unpredictable. Ensuring that these models are trained with bias-free data is crucial to prevent negative societal impacts and protect user rights.

If a model is not trained with bias-free data, the users of the model could be negatively impacted. For instance, the model we trained based on the movie review dataset could result in potential societal risks. It is unclear if the dataset we are provided is free from political and societal biases. If the model becomes of a part of a system that recommends movies to users, the model might falsely represent the quality of each movie based on the undetected biases in the dataset. This might negatively impact the movies that feature underrepresented communities, as those movies tend to be susceptible to biases and negative reviews.

In addition, the model might be used to manipulate public opinion. For example, various political campaigns rely on social media posts to gain a larger following. If the model were to be used on the social media platform, it may wrongfully flag a political post as negative even though it was merely one of the candidates sharing their ideals. This error could result in violating the right to free speech.

There are many ways to address these ethical concerns. First, we can implement various filtering methods for bias detection in our datasets. These methods could be done either manually or with the bias detection tools available to the public. In addition, we can emphasize the need for legal regulations on how the model could be used. One example of these regulations would be the requirement to have an ethics committee for using the model if an organization hopes to process the public-facing data. By addressing these ethical concerns proactively, we can ensure that our models promote fairness, accuracy, and the responsible use of technology in society.

References

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *North American Chapter of the Association for Computational Linguistics*.

Shih yang Liu, Chien-Yi Wang, Hongxu Yin, Pavlo Molchanov, Yu-Chiang Frank Wang, Kwang-Ting Cheng, and Min-Hung Chen. 2024. Dora: Weight-decomposed low-rank adaptation. *ArXiv*, abs/2402.09353.

Tim Salimans and Diederik P. Kingma. 2016. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. *ArXiv*, abs/1602.07868.

Lingling Xu, Haoran Xie, Si-Zhao Joe Qin, Xiaohui Tao, and Fu Lee Wang. 2023. Parameter-efficient fine-tuning methods for pretrained language models: A critical review and assessment. *ArXiv*, abs/2312.12148.

A Appendix

- TA mentor: Niel Nie
- External collaborators: No
- External mentor: No
- Sharing project: No