Bagging the Singular Value Decomposition - A Joint Implementation of LoRA and Bootstrap Aggregating as a Fine-tuning Regime

Stanford CS224N {Default} Project

Andri Vidarsson
ICME
Stanford University
andriv@stanford.edu

Jacob Thornton
ICME
Stanford University
jrthorn@stanford.edu

Raphaëlle Ramanantsoa
ICME
Stanford University
raphy@stanford.edu

Abstract

The efficient specialization of a general language model trained on unsupervised tasks to specific downstream tasks is of interest in many applications. Fine-tuning allows models to effectively utilize learning that occurred during pre-training to perform effectively on downstream tasks. In this project we are specifically interested in the fine-tuning of a miniBert implementation on the tasks of sentiment analysis, semantic similarity evaluation, and paraphrase detection. While fine-tuning can be carried out on the entire model architecture, adjusting each model parameter starting from the pre-trained parameters, this is needlessly expensive. We present a more efficient method for specializing to these three tasks by utilizing data augmentation, LoRA, and smoothness-inducing adversarial regularization. In addition to these methods, bootstrap aggregating was also tested as a means of increasing performance on downstream tasks. Implementing these methods allowed us to realize an increase in performance on the down stream tasks as compared to the pre-trained model at a reasonable computational cost.

1 Key Information to include

- Mentor: Jingwen Wu
- External Collaborators (if you have any): No
- Sharing project: No
- Team contributions: Andri took the lead on coding the LoRA and data augmentation components, and was instrumental in conducting the majority of the experiments. Jacob implemented bagging and provided substantial support with the experiments. Raphaëlle focused on integrating SIAR, and explored cosine similarity and gradient surgery.

2 Introduction

LLMs that use a masked language model learn the distribution of words based on context and as a consequence implicitly obtain a fairly deep understanding of the language. This results in a model with the capacity to generalize to specific tasks. Clearly, this is appealing as it allows for many different tasks such as sentiment analysis, paraphrase detection, and semantic textual similarity

evaluation to be carried out with a reasonable degree of accuracy on a pre-trained model. These models however, sacrifice a degree of accuracy on particular tasks that one may be interested in as a result of their generality. Through the application of fine-tuning methods, the performance of the general model on specific downstream tasks of interest can be improved. However, fine-tuning a model with a large number of parameters in totality is computationally expensive, often prohibitively so. Thus, methods for fine-tuning and improving performance of a general model such as BERT that are computationally inexpensive are of significant value.

In this project we are interested in combating the shortcomings of a general model by improving its performance on the downstream tasks of sentiment analysis, paraphrase detection, and semantic textual similarity evaluation while avoiding fine-tuning all of the model parameters. The goal of this project, as such, is to efficiently improve upon the performance of a miniBERT implementation on the aforementioned tasks measured by appropriate evaluation metrics for each task. We explore the effects of data augmentation applied to the task specific training data, implementation of regularization methods, and the use of LoRA for fine-tuning, as methods of efficiently increasing the performance of the pre-trained model on down stream tasks.

3 Related Work

We primarily build on the influential paper "LoRA: Low-Rank Adaptation of Large Language" Hu et al. (2021). Fine-tuning a large language model by post training all of the model's parameters to adapt to task specific data is computationally expensive due to the sheer size and complexity of the model. This paper addresses this challenge by proposing a method that utilizes low-rank adaptation (LoRA) matrices that are injected "on top" of the original weight matrices at selected layers. During training, the original pre-trained weight matrices remain frozen, while the LoRA matrices are trainable, this approach reduces the number of trainable parameters significantly, enhancing computational complexity. To assess the effectiveness of their contribution, the authors evaluated LoRA's performance against various baseline models using standard benchmarks. LoRA performed on-par or better than regular fine-tuning applied to the same baseline models. LoRA, or some variant of it is the current state of the art fine-tuning framework.

4 Approach

The approach utilized consisted of applying bootstrap aggregating, data augmentation, LoRA, projection layers, and SIAR.

4.1 LoRA

In the general framework, fine-tuning model weights is the process of learning new weight matrices $W = W_0 + \Delta W$, where $W_0 \in \mathbb{R}^{d\times k}$ is a pre-trained weight matrix and ΔW is a learned update matrix. This method can be applied either on a select subset of weights, or on all layers of the model. The paper leverages previous work that concluded that pre-trained models have a very low intrinsic dimension Aghajanyan et al. (2020), and proposes a low-rank decomposition of the update matrix $\Delta W = BA$. The matrices $B \in \mathbb{R}^{d\times r}, A \in \mathbb{R}^{r\times k}$ are called the adapters and their rank, r, is usually chosen so that $r << \min(d, k)$. The matrix B is initialized with zero and the matrix A is initialized with Gaussian noise, $N(0, \sigma^2)$. During task-specific training, only the matrices A and B receive gradient updates and the pre-trained matrix W_0 is frozen. Since the matrices A and B have low rank r, the number of trainable parameters corresponding to each hidden state is reduced from dk to $2dr$ which is much smaller given the choice of r. Figure 1 compares naive fine-tuning to the LoRA method.

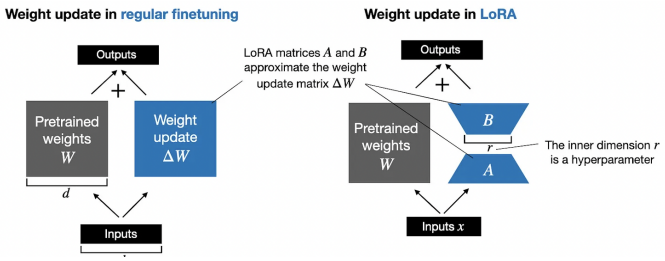

Figure 1. Regular fine-tuning vs. LoRA (Raschka, 2024) Raschka

With a vanilla implementation of LoRA, the two matrices comprising the low-rank update are initialized to Gaussian noise and zeros, the "noise and zero" adapters. However this initialization scheme is unlikely to be anywhere close to the actual optimal (in terms of loss minimization) values of the adapters, which can result in slow convergence. Principal Singular values and Singular vectors Adaptation (PiSSA), provides means for overcoming this short coming of a typical LoRA implementation Meng et al. (2024).

Using PiSSA rather than initializing the noise and zero adapters, PiSSA initializes two adapters A and B by using the principal components of the pre-trained weight matrix, W, while the non principal components are are stored in a residual matrix, W^{res}, and frozen. Aside from the clever initialization of the adapter and residual matrices PiSSA functions identically to regular LoRA facilitating faster convergence and therefore more efficient and effective fine-tuning Meng et al. (2024).

4.2 Smoothness-Inducing Adversarial Regularization

To improve the robustness of our multitask classifier and enforce consistent predictions when the embeddings are noisy, we experimented with Smoothness-Inducing Adversarial Regularization (SIAR) as introduced by Jiang et al. (2019). SIAR essentially incorporates regularization and solves the following optimization problem for fine-tuning

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

where the loss function is $L(\theta) = \frac{1}{n}\sum_{i=1}^{n} l\left(f(x_i;\theta), y_i\right)$ and l_s is the sum of the losses from our three downstream tasks, $\lambda_s > 0$ is the tuning parameter, $R_s(\theta)$ is the smoothness-inducing reguralizer that we defined as

$$R_s(\theta) = \frac{1}{n}\sum_{i=1}^{n} l\left(f(x_i;\theta), f(x_i+\epsilon;\theta)\right)$$

and $\epsilon > 0$ is a tuning parameter. To optimize computational efficiency, this choice of regularizer differs from the one introduced in the original paper.

4.3 Bootstrap Aggregating

Bootstrap aggregating, or bagging, allows for the generation of different predictions which can then be aggregated to make a final prediction, which improves performance and reduces variance. This process of generating multiple predictions is carried out using bootstrapping. For the dataset $\mathcal{L}$ consisting of the input and output pairs $(y_n, x_n), n = 1, ..., N$ where the output y_n is either a label, in the SST dataset for example, or a number, in the SemEval STS Benchmark dataset for example. Using $\mathcal{L}$ we can construct k different $\mathcal{L}_j$s for $j = 1, ..., k$ by sampling $\mathcal{L}$ with replacement N times. Carrying this entire process out k times results in k datasets of N data pairs drawn from the distribution of $\mathcal{L}$ Breiman (1996).

We can then leverage these different datasets by training a model on each of the k datasets and aggregating the model predictions. By aggregating the predictions we mean that for some input x if the output is a label then we select the label that received the most votes from the k predictions for classification tasks, and the mean across models for regression tasks Breiman (1996).

4.4 Data Augmentation

Data Augmentation (DA) was motivated by the paper Easy Data Augmentation (EDA) Wei and Zou (2019). The EDA paper shows that the smaller the size of a training dataset, the more it stands to gain from DA methods that effectively artificially increase the size of the data set. However, depending on the resources available, the methods used in EDA such as synonym replacement, random insertion, random swap, and random deletion can be unnecessarily simplistic. Rather than perform these basic operations, we opted to use a fine-tuned LLM to generate paraphrased examples from our dataset(s). For this, we used FLAN-T5 Chung et al. (2022). While this method is general and could be applied to any of the data sets in principle, in practice the most significant improvement was associated with the SST dataset, when trained with a parameter efficient setup.

4.5 Projections and Cosine Similarity

To encourage a more robust model architecture with the capacity to generalize to many different example types, projections from the Bert embedding space down to a smaller dimension were used. Depending on the downstream task the usage of the embeddings were different.

Sentiment Analysis The sentiment analysis task was the only task for which no projection was used. For this task the Bert embedding was fed directly to the linear classification which outputted five logits associated with the possible annotations.

Paraphrase Detection Two input questions were associated with each paraphrase dataset input. Each of these inputs were independently fed through the Bert model then the paraphrase projection layer(s). After the embeddings were obtained they were concatenated then fed to the classification layer to obtain the prediction.

Semantic Similarity Analysis Semantic similarity analysis task inputs also had two sentences associated with them. Both inputs were fed through the similarity projection layer(s) just as in the paraphrase detection. However for semantic similarity analysis, following the projection, the cosine similarity between the two embeddings was calculated then scaled to take on values zero to two by adding one, then finally it was scaled to take on values zero to five by multiplying the resulting logit of this calculation by 2.5.

5 Experiments

5.1 Data

The datasets used in the experiments were the Stanford Sentiment Treebank (SST), CFIMDB, Quora question pair, and SemEval STS Benchmark datasets.

The SST and CDIMDB datasets were used for sentiment analysis. Each example in the SST dataset consisted of a sentence assigned a tag of negative, somewhat negative, neutral, somewhat positive, or positive by a human judge. Examples in the CFIMDB dataset consisted of polar movie reviews each assigned a sentiment of either positive or negative.

For paraphrase detection the Quora question pair dataset were used. In this dataset examples were made up of a pair of questions assigned a label based on whether or not they were paraphrases of each other—as evaluated by a human judge.

Semantic similarity evaluation performance was carried out on the SemEval STS Benchmark data set. Each example in this dataset contained a pair of sentences assigned a score from zero to five based on their similarity where a score of five indicates the sentences are paraphrases of each other and a score of zero indicates they are unrelated. Datasets have been split into train, dev, and test sets according to Table 1.

Dataset	Train Size	Dev Size	Test Size	Total
SST	$8,544$	$1,101$	$2,210$	$11,855$
CFIMDB	$1,701$	245	488	$2,434$
Quora	$283,010$	$40,429$	$80,859$	$404,298$
SemEval	$6,040$	863	$1,725$	$8,628$

Table 1. Dataset Splits

5.2 Evaluation method

Performance on each of the down stream tasks was evaluated based on accuracy of the predictions on the out sample dev set based on the correct provided labels. Accuracy was computed straight forwardly—as the percentage of predictions correct—on the data sets that used integer labels; those being the SST, CFIMDB, and Quora datasets. For the STS SemEval dataset the Pearson score was calculated and used as the metric of accuracy.

5.3 Experimental details

While some of the implemented methods were shown to improve performance and did not use any tunable parameters, LoRA, Bagging, and SIAR all had multiple parameters that could be adjusted. Results of specific experiments involving the isolation of single parameter or component of the model are displayed in this section; however, numerous other experiments were carried out as well. To see a list of the majority of the experiments that were carried out refer to Table A.1 in the Appendix.

The Benchmark Model In each of the experiments we evaluated against a benchmark model to demonstrate the score improvement due to the method or architecture change of interest. The benchmark model for all of the experiments aside from the projection experiments consisted of a full multitask model trained using eight epochs, a Bert learning rate of $1e{-}5$, a classifier learning rate of $1e{-}4$, and an MLP projection for the semantic similarity analysis and paraphrase detection tasks. Any reference to the Benchmark Model refers to this model architecture unless otherwise specified. Additionally, any model parameters that are not specified used the Benchmark Model parameters when applicable. This is the case for the fine-tuned models as well; however, fine-tuning training was carried out using only two epochs.

Data Augmentation Data augmentation was examined quantitatively along side the implementation of LoRA. LoRA models were evaluated with and without data augmentation to see if any performance improvement was realized. Further, the data augmentation implementation was evaluated qualitatively by examining the quality of the outputted paraphrases and an example of a generated paraphrase is given.

Bootstrap Aggregating Bagging was tested as a way for improving the task specific fine-tuning after the multitask training had already been carried out. To do this the train dataset was sampled as described in the approach section to generate different numbers of bagging models. However for the paraphrase dataset it was very expensive to train on the entire data set, so to produce models that collectively understood the underlying distribution, for one bagging model, the paraphrase dataset was first chunked into a random third of the data then bootstrap sampling was carried out on this third of the data to produce the dataset for the corresponding bagging model.

To evaluate the efficacy of bagging on the fine-tuning process, the Benchmark Model was trained as described in the benchmark section, this model was then fine-tuned on each of the tasks using different numbers of bagging models to investigate the effect on performance and justify the use of bagging in the fine-tuning process.

LoRA The LoRA model architecture has the following tunable parameters: rank, initialization scheme (standard or SVD), Bert learning rate, classifier learning rate, and weight decay. In this experiment the rank, SVD initialization, and weight decay parameters were adjusted to sample a subset of the space of possible parameters. The models were evaluated on sentiment analysis using both the SST and CFIMDB datasets.

SIAR To demonstrate the efficacy of SIAR on the tasks, a parameter search was carried out on the ϵ and λ parameters of the SIAR implementation. For this experiment the Benchmark Model was used as a starting point and was then fine-tuned on the task of semantic similarity analysis using parameters in the range of 0.1 to 1 for both ϵ and λ.

Projection Layer(s) Different projection layer architectures were explored in an attempt to identify the architecture that generalized best from the training set to the dev and test sets. For all projection architectures the final output embedding was dimension eight. The three architectures investigated were a single linear projection, three stacked linear projections, as well as a multilayer perceptron (MPL) projection, which contained three layers in the paraphrase implementation and two in the similarity. The MLP projection utilized the Gaussian linear error unit (GELU) activation function. As previously mentioned, these projections were used only on the tasks of semantic similarity analysis and paraphrase detection.

5.4 Results

The Benchmark Model The performance of the benchmark model on each data set and following fine-tuning on each of the tasks is shown in Table 2.

Model	SST	Para	STS
Benchmark	0.515	0.751	0.709
Benchmark Fine-tuned	0.521	0.754	0.731

Table 2. Benchmark Model Performance

Any reference to the Benchmark Model performance refers to this model which consisted of the architecture described in the Experiments section.

Data Augmentation Evaluating the quality of paraphrases generated through a method such as the one earlier described can be difficult. However, we believe the quality was quite good based on the benefit provided in fine-tuning which is shown in the LoRA section. Further, qualitatively, based on random samples drawn from the generation, the paraphrases appear to be of high quality. Table 3 shows five samples from an input sentence that are all coherent and similar to the input sentence.

Input	i don't think this movie loves women at all.
Paraphrases	a sexist, cynical satire the sex scenes are so vile and the story so demeaning I couldn't even watch it. a movie that's so sexist it makes you want to punch the screen it's a movie that makes women look like fools this is a movie that's sexist and, worse : it tries to be.

Table 3. Paraphrase Examples

Quantitatively we can also analyze the performance of the model with and without DA. In Table 5 we can see that at least for the SST dataset, the implementation of DA always improves performance. In general the implementation of DA was most beneficial on the SST dataset likely due to the one sentence length of SST inputs as well as the nature of the data, that is, sentences expressing a sentiment.

Bootstrap Aggregating Bagging was tested in the fine-tuning of the model on each of the downstream tasks after the initial training phase. The Benchmark Model was fine-tuned on each each of the tasks to serve as the benchmark. Bagging was then tested using 4 and 6 models. The results of these experiments are shown in Table 4.

n Bagging Models	SST	Para	STS
0	0.521	0.754	0.731
4	0.529	0.768	0.756
6	0.513	0.755	0.769

Table 4. Bagging Fine-tuning Performance

From the results of the bagging experiments it is mostly clear that bagging increases performance. While there was a drop going from four to six models in both the paraphrase detection and sentiment analysis tasks the drop was only marginal and likely due to an amount of randomness that can be mitigated through the use of a larger number of models. For these reasons we expect a higher number of bagging models to be in general more effective as well as more consistent in their performance.

LoRA We then have the results of the various LoRA configurations shown in Table 5. The learning rates for the Bert model and classifier were $1e{-}4$ and $1e{-}3$ respectively. As shown in Table 5 the best accuracy achieved on the CFIMDB dataset was 0.959 with two different configurations, rank 16 SVD initialized LoRA and rank 32 LoRA both without a weight decay. For the SST dataset the best accuracy was 0.497 using the rank 32 SVD initialized LoRA with DA and no weight decay. These results support the usage of SVD initialized LoRA as it out performed the vanilla LoRA implementation, achieving the same accuracy with a lower rank on the CFIMDB dataset and achieving a better accuracy on the SST data set.

Dataset	SVD	LoRA Rank	Weight Decay	Dev Accuracy
SST	Yes	16	0	0.471
SST with DA				0.494
CFIMDB				0.959
SST	No			0.462
CFIMDB				0.959
SST	Yes	32		0.473
SST with DA				0.497
CFIMDB				0.951
SST		16	0.01	0.477
CFIMDB				0.955

Table 5. LoRA Fine-tuned Model Performance

SIAR As mentioned the SIAR implementation was tested for fine-tuning on the STS data set after multitask training. We refer to the Benchmark Model performance on the same task after fine-tuning as the benchmark. The Benchmark Model had a score of 0.731. In Table 6 we see that the use of SIAR resulted in the moderate improvement of scores on the task.

$\epsilon \backslash \lambda$	0.1	0.25	0.5	0.75	1
0.1	0.729	0.733	0.733	0.732	0.738
0.25	0.731	0.726	0.732	0.733	0.735
0.5	0.732	0.732	0.727	0.727	0.735
0.75	0.735	0.732	0.733	0.740	0.738
1	0.723	0.733	0.729	0.729	0.737

Table 6. SIAR STS Grid Search

As can be seen in Table 6 the SIAR implementation performed best with λ of 1. Performance also appeared to be largely indifferent to the value of ϵ and was on average improved by the incorporation of SIAR.

Projection Layer(s) As previously mentioned we considered three different projection architectures for the tasks of paraphrase detection and semantic textural similarity analysis. The results of the different model architectures are shown for all of the downstream tasks in Table 7.

Architecture	SST	Para	STS
Single Linear	0.505	0.770	0.767
Triple Stacked Linear	0.500	0.767	0.768
MLP	0.515	0.751	0.709

Table 7. Projection Performance

In terms of projection architecture, the single and triple linear layers performed mostly the same with the single linear having a slightly higher score on the paraphrase detection and sentiment analysis tasks. The MLP projection showed the best performance on the SST task with worse performance on the other two tasks. Despite this worse performance on the paraphrase detection and sentiment similarity analysis tasks, we suspected that the MLP projection may perform better as the non-linearity may allow for more robustness after LoRA and other methods are incorporated.

Final Performance The final model architecture used on the dev set will now be explained and justified in terms of our previously presented experimental results. The final model was multitask trained using SVD initialized rank 16 LoRA. These LoRA parameters were chosen based on our experiments which showed that SVD initialized LoRA performs strictly better than LoRA using a standard initialization. A rank of 16 was chosen as a compromise between performance and number of trainable parameters as rank 16 performed better than rank 8 but slightly worse than rank 32. After LoRA multitask training was carried out, the model was then fine-tuned on each of the down stream tasks using bagging with 10, 10, and 8 models for SST, STS, and paraphrase tasks respectively. As mentioned in the bagging results section, we suspected that a relatively large number of bagging models would result in the best performance. We chose 10, 10, and 8 mostly due to training time constraints. For the paraphrase detection task 8 bagging models was chosen since that task was the most expensive to train with (or without) bagging. SIAR was also used in each of the fine-tuning instances with parameters of $\lambda = 0.1$ and $\epsilon = 0.1$. For the projection layer the MLP projections described in the Benchmark Model and projection experiment sections were used with the intuition that the non-linearity may generalize better with LoRA, SIAR, and bagging. The final performance of the model after the implementation of all previously described methods is shown in Table 8.

	SST	Paraphrase	STS	Overall
Dev	0.549	0.793	0.822	0.751
Test	0.511	0.749	0.696	0.703

Table 8. Final Model Performance

As shown in Table 8 our chosen model architecture resulted in a respectable dev accuracy on all of the tasks as well as the second highest Dev Leaderboard accuracy for the SST task. However, we recognize the potential for much better performance after further hyperparameter tuning.

6 Analysis

In terms of our performance on the Dev Leaderboard, the final model's strongest task was sentiment analysis. We suspect that our impressive performance on this task can be attributed to our implementation of data augmentation on the SST dataset as well as our use of bagging. In all likelihood, these methods provided predictions that were based on a much deeper understanding of the underlying distribution than if one of these methods were not used. It also seems likely that these methods worked especially well together. Data augmentation provided novel examples to be drawn from in bagging, resulting in each bagging model training on a random amount of the augmented data, providing a rich parent training set and diverse bagging learning sets.

The moderate performance on paraphrase prediction suggests that while the model is robust, there is a need for further tuning it to more nuanced language. Our relative under performance on the paraphrase detection task suggests potential for optimizing our training strategy. Though the the use of SIAR proved helpful in ensuring consistency across inputs, there is scope for hyperparameter tuning to leverage the robustness offered by SIAR.

One element of the implementation that proved very effective was the SVD initialization of LoRA. In our experiments and less formal testing, it was clear that this strategy provided a real performance boost. This is logical as the SVD initialization provides a much better starting point for learning than the random noise and zero adapters used in a standard LoRA implementation. Additionally, we found the coupling of LoRA and bagging to be natural as LoRA reduces the cost associated with training the bagging models.

7 Conclusion

A pre-trained miniBert implementation was efficiently specialized to the tasks of sentiment analysis, semantic similarity analysis, and paraphrase detection. This was done using a novel set of methods empirical shown to improve performance, both in past research as well as in our own ablation studies. These methods included bagging, data augmentation, LoRA, and smoothness-inducing adversarial regularization. We achieved reasonable performance on the dev set for each task and achieved the second highest score for sentiment analysis task, likely due to the incorporation of bagging in our fine-tuning architecture. While a performance increase was certainly realized, further ablation studies and parameter searches could be carried out. We suspect that if this was done performance would be greatly improved using the optimal parameters compared to our current set up. However, our parameter choices were motivated, where possible, by the empirical results presented in the results section as well as in the Table A.1 included in the Appendix.

8 Ethics Statement

There are two major ethical concerns with our miniBERT implementation. The first has to do with bias, specifically the under representation of certain groups in the Quora dataset used during model training for the paraphrase detection task. In their release statement of said dataset, Quora stated that "the distribution of questions in the dataset should not be taken to be representative of the distribution of questions asked on Quora." If deployed in production, our miniBERT model adapted to the paraphrase detection task may thus favor more common formulations and not account for cultural variations in language. To mitigate this bias, strategies include ensuring the dataset is representative before training (albeit not possible here), fine-tuning the LLM on more culturally diverse datasets by post, and leveraging bias reduction techniques such as counterfactual data augmentation.

The second ethical concern of our project has to do with user data privacy, particularly in the SST dataset used during model training for the sentiment analysis task. Though the users may have implicitly agreed to the publication of their movie reviews, they may not have explicitly given their consent for their opinions, thoughts, and use of language to be processed and analyzed. In the sentiment analysis task, the miniBERT model learns to "determine individual feelings" effectively revealing the users' thoughts and opinions. A strategy to ensure user privacy is to use Differential Privacy (DP) in the sentiment analysis task as was done by Vogel and Lange (2023). DP consists in adding noise to the output of the model to ensure users' individual information cannot be retrieved.

References

Armen Aghajanyan, Luke Zettlemoyer, and Sonal Gupta. 2020. Intrinsic dimensionality explains the effectiveness of language model fine-tuning. *arXiv preprint arXiv:2012.13255*.

Leo Breiman. 1996. Bagging predictors. *Machine Learning*, 24(2):123–140.

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Eric Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, Albert Webson, Shixiang Shane Gu, Zhuyun Dai, Mirac Suzgun, Xinyun Chen, Aakanksha Chowdhery, Sharan Narang, Gaurav Mishra, Adams Yu, Vincent Zhao, Yanping Huang, Andrew Dai, Hongkun Yu, Slav Petrov, Ed H. Chi, Jeff

Dean, Jacob Devlin, Adam Roberts, Denny Zhou, Quoc V. Le, and Jason Wei. 2022. Scaling instruction-finetuned language models.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*.

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

Fanxu Meng, Zhaohui Wang, and Muhan Zhang. 2024. Pissa: Principal singular values and singular vectors adaptation of large language models.

Sebastian Raschka. Improving lora: Implementing weight-decomposed low-rank adaptation (dora) from scratch. `https://magazine.sebastianraschka.com/p/lora-and-dora-from-scratch`.

Felix Vogel and Lucas Lange. 2023. Privacy-preserving sentiment analysis on twitter.

Jason W. Wei and Kai Zou. 2019. EDA: easy data augmentation techniques for boosting performance on text classification tasks. *CoRR*, abs/1901.11196.

A Appendix

A.1 Master Table

Presenting all experimental results in a clear and concise format proved difficult due to the number of experiments and possible model architectures. For this reason we present all of the experiments in Table A.1. SIAR, LoRA, and bagging experiments have been grouped vertically.

Model	Projection	Multitask	Task Tuned	Task	Epochs	Bert LR	Class LR	Bagging	N-Models	SIAR	λ	ϵ	Rank	SVD	Score
Lora	MLP	No	No	sst	8	1E-5	1E-4	No	N/A	No	N/A	N/A	8	No	0.471
Lora	MLP	No	No	sst	8	1E-4	1E-3	No	N/A	No	N/A	N/A	16	No	0.462
Lora	MLP	No	No	sst	8	1E-4	1E-3	No	N/A	No	N/A	N/A	16	Yes	0.477
Lora	MLP	No	No	sst	8	1E-4	5E-4	Yes	10	No	N/A	N/A	32	Yes	0.452
Lora	MLP	No	No	sst	8	1E-4	1E-3	No	N/A	No	N/A	N/A	32	Yes	0.473
Lora	MLP	No	No	sst	8	1E-3	1E-3	No	N/A	No	N/A	N/A	8	Yes	0.353
Lora	MLP	No	Yes	sts	5	5E-5	1E-4	Yes	10	Yes	0.1	0.1	8	No	0.822
Lora	MLP	Yes	No	sst	10	1E-5	1E-4	No	N/A	Yes	0.1	0.1	32	Yes	0.491
Full	MLP	Yes	No	sst	8	1E-5	1E-4	No	N/A	Yes	0.1	0.1	N/A	N/A	0.504
Full	MLP	Yes	No	sst	5	2E-5	8E-5	No	N/A	Yes	0.25	0.25	N/A	N/A	0.507
Full	MLP	No	Yes	sst	5	5E-5	1E-4	Yes	10	Yes	0.1	0.1	N/A	N/A	0.549
Full	MLP	No	Yes	para	2	1E-5	1E-4	Yes	2	No	N/A	N/A	N/A	N/A	0.755
Full	MLP	No	No	sst	8	1E-5	1E-4	Yes	10	No	N/A	N/A	N/A	N/A	0.528
Full	MLP	No	No	sst	8	1E-5	1E-4	Yes	2	No	N/A	N/A	N/A	N/A	0.291
Full	MLP	No	Yes	sts	2	1E-5	1E-4	Yes	2	No	N/A	N/A	N/A	N/A	0.369
Full	MLP	No	No	sst	8	1E-5	1E-4	Yes	10	No	N/A	N/A	N/A	N/A	0.540
Full	Single Linear	Yes	Yes	sst	4	1E-5	1E-4	Yes	10	No	N/A	N/A	N/A	N/A	0.526
Full	MLP	Yes	Yes	para	2	1E-5	1E-4	Yes	2	No	N/A	N/A	N/A	N/A	0.763
Full	MLP	Yes	Yes	para	2	1E-5	1E-4	Yes	6	No	N/A	N/A	N/A	N/A	0.769
Full	MLP	Yes	Yes	sst	2	1E-5	1E-4	Yes	2	No	N/A	N/A	N/A	N/A	0.500
Full	MLP	Yes	No	sst	2	1E-5	1E-4	Yes	6	No	N/A	N/A	N/A	N/A	0.513
Full	MLP	Yes	Yes	sts	2	1E-5	1E-4	Yes	2	No	N/A	N/A	N/A	N/A	0.758
Full	MLP	Yes	Yes	sts	2	1E-5	1E-4	Yes	6	No	N/A	N/A	N/A	N/A	0.755
Full	MLP	Yes	Yes	sst	2	1E-5	1E-4	Yes	4	No	N/A	N/A	N/A	N/A	0.529
Full	MLP	Yes	Yes	para	2	1E-5	1E-4	Yes	4	No	N/A	N/A	N/A	N/A	0.768
Full	MLP	No	Yes	sst	4	1E-5	1E-4	Yes	10	No	N/A	N/A	N/A	N/A	0.511
Full	MLP	No	Yes	sst	8	1E-5	1E-4	Yes	10	No	N/A	N/A	N/A	N/A	0.526
Full	MLP	No	Yes	para	3	1E-5	1E-4	Yes	8	No	N/A	N/A	N/A	N/A	0.793

Last Linear	MLP	No	No	sst	8	1E-5	1E-4	No	N/A	No	N/A	N/A	N/A	N/A	0.389
Last Linear	MLP	No	No	sst	8	1E-3	1E-3	No	N/A	No	N/A	N/A	N/A	N/A	0.386
Full	MLP	No	No	sst	8	1E-5	1E-4	No	N/A	No	N/A	N/A	N/A	N/A	0.525
Full	MLP	No	Yes	para	2	1E-5	1E-4	No	N/A	No	N/A	N/A	N/A	N/A	0.760
Full	MLP	No	Yes	para	4	1E-5	1E-4	No	N/A	No	N/A	N/A	N/A	N/A	0.778
Full	MLP	No	No	para	8	1E-5	1E-4	No	N/A	No	N/A	N/A	N/A	N/A	0.632
Full	MLP	Yes	No	para	8	1E-5	1E-4	No	N/A	No	N/A	N/A	N/A	N/A	0.750
Full	MLP	Yes	No	sst	8	1E-5	1E-4	No	N/A	No	N/A	N/A	N/A	N/A	0.490
Full	MLP	Yes	No	sts	8	1E-5	1E-4	No	N/A	No	N/A	N/A	N/A	N/A	0.724
Full	MLP	No	Yes	sst	8	1E-5	1E-4	No	N/A	No	N/A	N/A	N/A	N/A	0.515
Full	MLP	No	Yes	sst	2	1E-5	1E-4	No	N/A	No	N/A	N/A	N/A	N/A	0.480
Full	MLP	No	Yes	sst	4	1E-5	1E-4	No	N/A	No	N/A	N/A	N/A	N/A	0.477
Full	MLP	No	Yes	sst	4	1E-5	1E-4	No	N/A	No	N/A	N/A	N/A	N/A	0.526
Full	MLP	Yes	No	sst	10	1E-5	1E-4	No	N/A	No	N/A	N/A	N/A	N/A	0.513
Full	MLP	Yes	No	sst	8	1E-5	1E-4	No	N/A	No	N/A	N/A	N/A	N/A	0.505
Full	MLP	Yes	No	sst	8	3E-5	4E-5	No	N/A	No	N/A	N/A	N/A	N/A	0.496
Full	MLP	No	No	sst	8	1E-5	1E-4	No	N/A	No	N/A	N/A	N/A	N/A	0.390
Full	MLP	No	Yes	sts	2	1E-5	1E-4	No	N/A	No	N/A	N/A	N/A	N/A	0.368
Full	MLP	No	Yes	sts	4	1E-5	1E-4	No	N/A	No	N/A	N/A	N/A	N/A	0.801
Full	MLP	No	No	sts	8	1E-5	1E-4	No	N/A	No	N/A	N/A	N/A	N/A	0.168
Full	Single Linear	Yes	No	sst	8	1E-5	1E-4	No	N/A	No	N/A	N/A	N/A	N/A	0.505
Full	Single Linear	Yes	No	para	8	1E-5	1E-4	No	N/A	No	N/A	N/A	N/A	N/A	0.770
Full	Single Linear	Yes	No	sts	8	1E-5	1E-4	No	N/A	No	N/A	N/A	N/A	N/A	0.767
Full	Triple Linear	Yes	No	sst	8	1E-5	1E-4	No	N/A	No	N/A	N/A	N/A	N/A	0.500
Full	Triple Linear	Yes	No	para	8	1E-5	1E-4	No	N/A	No	N/A	N/A	N/A	N/A	0.767
Full	Triple Linear	Yes	No	sts	8	1E-5	1E-4	No	N/A	No	N/A	N/A	N/A	N/A	0.768
Full	Single Linear	Yes	Yes	sst	4	1E-5	1E-4	No	N/A	No	N/A	N/A	N/A	N/A	0.526
Full	Single Linear	Yes	Yes	para	4	1E-5	1E-4	No	N/A	No	N/A	N/A	N/A	N/A	0.805
Full	Single Linear	Yes	Yes	sts	4	1E-5	1E-4	No	N/A	No	N/A	N/A	N/A	N/A	0.785
Full	Single Linear	Yes	Yes	para	4	1E-5	1E-4	No	N/A	No	N/A	N/A	N/A	N/A	0.778
Full	Single Linear	Yes	No	sst	8	1E-5	1E-4	No	N/A	No	N/A	N/A	N/A	N/A	0.490
Full	Single Linear	Yes	No	para	8	1E-5	1E-4	No	N/A	No	N/A	N/A	N/A	N/A	0.750
Full	Single Linear	Yes	No	sts	8	1E-5	1E-4	No	N/A	No	N/A	N/A	N/A	N/A	0.724
Full	MLP	Yes	No	sst	8	1E-5	1E-4	No	N/A	No	N/A	N/A	N/A	N/A	0.515
Full	MLP	Yes	No	para	8	1E-5	1E-4	No	N/A	No	N/A	N/A	N/A	N/A	0.751
Full	MLP	Yes	No	sts	8	1E-5	1E-4	No	N/A	No	N/A	N/A	N/A	N/A	0.709

Full	MLP	Yes	Yes	para	2	1E-5	1E-4	No	N/A	No	N/A	N/A	N/A	N/A	0.754
Full	MLP	Yes	Yes	sst	2	1E-5	1E-4	No	N/A	No	N/A	N/A	N/A	N/A	0.521
Full	MLP	Yes	Yes	sts	2	1E-5	1E-4	No	N/A	No	N/A	N/A	N/A	N/A	0.731

Table A.1. Master Table