Deep Learning Mini-Project
Finetuning with LoRA

Team GradesAreAllYouNeed

Ansh Sarkar[1], **Princy Doshi**[2], **Simran Kucheria**[3]
[1]**as20363@nyu.edu**, [2]**pd2672@nyu.edu**, [3]**sk11645@nyu.edu**
New York University

Abstract

The project aims to build a text classification model fine-tuned using Low-Rank Adaptation (LoRA) on the AGNews dataset with less than 1 million trainable parameters. There were other constraints as well, such as using a specific version of Roberta as the base.

We created and trained multiple models with different LoRA configurations on the AGNews dataset to classify text. We tried various techniques that act as regularization for the model so that there is better generalization while training the model. Our best model achieves a validation accuracy of **93.7895%** and a final test accuracy of **85.050%** .

Implementation can be found here: https://github.com/anshsarkar/ECE-GY-7123-Deep-Learning-Project-2.git

Overview

Parameter-Efficient Fine-Tuning (PEFT) is a technique that allows the fine-tuning of large-language models to be resource-efficient. It fine-tunes only a small number of model parameters, adapting a pre-trained model for a specific downstream task instead of fine-tuning the entire model. LoRA is one of the most used methods among the various techniques of PEFT.

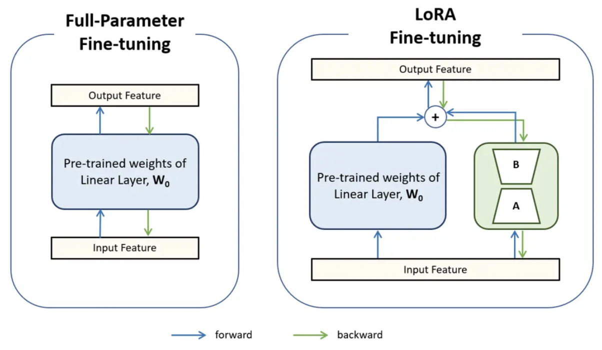

Full-Parameter Fine-tuning
Output Feature
Pre-trained weights of Linear Layer, W_0
Input Feature
LoRA Fine-tuning
Output Feature
+
Pre-trained weights of Linear Layer, W_0
B
A
Input Feature
forward
backward

Figure 1: Full Finetuning vs LoRA

LoRA reduces the number of trainable parameters by freezing the pre-trained model weights and injecting trainable rank decomposition matrices into each layer of the Transformer Architecture, greatly reducing the number of trainable parameters for downstream tasks.
This is because fine-tuning pretrained weights can be represented as a sum of the pretrained weight (W_0) and the accumulated gradient update (ΔW), which can be decomposed into two low-rank matrices, A and B.

The main advantages of using LORA are the following:

- fewer trainable parameters, so gradient computation is much easier and can be done on a much cheaper hardware.
- keeping the pre-trained weights frozen and only training the new weights for the given task, so it is easy to switch out adapters, merge more than one adapter, etc.

The main parameters in LoRA are:

- r - the rank of the matrix AB (ie, the smaller dimension of either A or B)
- α - the scalar multiplied by AB when added to the original weight

As such, we decided to experiment with different permutations of r and α as explained in later sections of the report to come to our final configurations.

The final LoRA configurations of our model were as follows:

- LoRA Rank: 4
- LoRA Alpha: 8
- LoRA Dropout: 0.6
- LoRA target modules: [”query”, ”value”]
- Lora Bias: None

combined with other advanced techniques, with the following configuration and hyperparameters:

- Learning Rate: 2e-4
- Batch Size: 32
- Number of Epochs: 10
- Learning Rate Scheduler: Cosine
- Warmup Ratio: 10%

- Using DoRA: True
- Using rsLoRA: True
- LoRA Weight Initialization: Gaussian

Methodology

Our methodology focused on optimizing the model training by employing preprocessing techniques, trying out different LoRa configurations, regularization, training stabilization, and weight initialization techniques.
To perform experiments we used the AG-News dataset from Huggingface, which automatically splits into training and validation datasets using a 94:6 split, with the unlabeled data hosted on Kaggle acting as our test set.
Initially we carried out experiments with different LoRa configurations and regularization techniques such as weight decay, learning rate schedulers, and LoRA dropouts before moving on to advanced PEFT techniques that stabilize the training, such as DoRA and rsLoRA.
Experiments involving the target modules of the Roberta-base model, such as query, key, value, and output.dense layers were also carried out.
Different weight initialization techniques, such as Gaussian and Pissa, were also employed to improve the accuracy.
Data preprocessing using stop words/punctuation removal, stemming, and lemmatization were tried out.
Additionally, we also explored techniques where we can change LoRA rank and alpha for the higher layers, which are more task-specific, so that the model can be adapted even more.

Preprocessing Techniques

Before we started experimenting with hyperparameters, we decided to preprocess the dataset to help increase the accuracy of the model.
To do so, we tried basic preprocessing techniques that involve cleaning the data by removing punctuations and stopwords.
Upon experimentation, we observed that these techniques were causing our model to perform poorly as compared to our baseline, which did not have any of these techniques.
In hindsight, the reason behind the underperformance could be the impact of these techniques on our base Roberta model, which leads to contextual loss and over-generalisation of the sentence.
Hence, we decided not to employ any preprocessing techniques.

LoRA Configurations

To experiment with different LoRA Configs we used the LoraConfig method from the peft library. This allows us to set different hyperparameters like r, α, target modules, dropout, amongst others.

Initially we experimented only with r and α values before adjusting the other parameters. Those are outlined in the subsequent sections.

The LoRA rank values were in the range of [2, 4, 8], and the alpha values were 0.5 times, 2 times, 4 times, and 8 times the rank.
Tabulating some of the results here:

LoRA Configurations				
Rank	**Alpha**	**Target Modules**	**Val. Loss**	**Val. Accuracy (%)**
8	16	["query", "value"]	0.174251	0.943158
4	16	["query", "value"]	0.174306	0.944342
4	8	["query", "value"]	0.179611	**0.941316**
4	32	["query", "value"]	0.215468	0.925789
4	2	["query", "value"]	0.169931	**0.944868**
4	8	["query", "output.dense"]	0.176596	0.944211
2	4	["query", "value", "output.dense"]	0.179222	0.941316

Table 1: Experimenting with different LoRA ranks, alphas, and target modules

Even though the highest validation accuracy was observed for the model with rank 4 and α 2, we ended up going ahead with rank 4 and α 8 as that was performing the best on the unlabeled Kaggle test dataset.

Advanced Techniques - Training Stabilisation

While experimenting with various LoRA (Low-Rank Adaptation) configurations, we observed that the model exhibited signs of rapid overfitting and showed characteristics of unstable training dynamics that persisted even with the application of standard regularization techniques. To curb the overfitting, we used techniques like DoRA and rsLoRA, which brought about distinct improvements.

- **DoRA**: DoRA improves training stability and regularization by splitting weights into a learned scalar (magnitude) and a LoRA-based direction. This restricts updates to a normalized subspace, reducing erratic shifts and limiting the optimization path. Dynamic normalization ensures stable gradients, and DEM loss balances variance across components, preventing gradient explosions. Overall, DoRA offers robustness and better generalization without explicit regularization.
- **rsLoRA**: rsLoRA modifies LoRA scaling from α/r to $\alpha/\sqrt{r}$, reducing the impact of the adapter and acting as a weight decay. This stabilizes training, especially at higher ranks, by preventing gradient collapse and vanishing updates. It removes rank sensitivity, smoothens performance, and improves generalization.

As seen in the graph DoRA + rsLoRa leads to a slightly lower loss. rsLoRA stabilizes training at higher ranks. Even

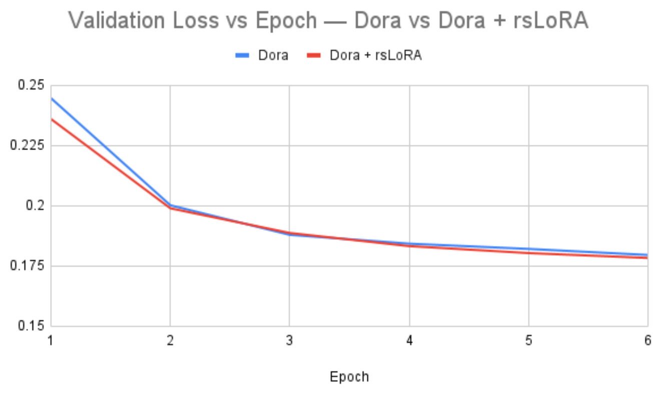


Figure 2: Validation loss for only DoRA and DoRA + rsLoRA models

though our ranks were not very high, we decided to keep rsLoRA to get a slight stabilizing effect for our training.

Regularization Techniques

Other than DoRA and rsLoRA, which add a regularization effect to the training we also experimented with other standard regularization techniques such as:

- **LoRA Dropouts**: LoRA dropout randomly deactivates elements in the low-rank adaptation matrices during training. It injects controlled noise to prevent overreliance on specific parameters. This regularization promotes more robust, generalizable features and reduces overfitting.
- **L2 Regularization**: L2 regularization (weight decay) penalizes large weights by adding their squared values to the loss. It limits parameter growth, discouraging overfitting and promoting simpler, more generalizable models.
- **LR Schedulers**: Dynamic learning rate adjustment plays a key role in effective model convergence. We explored three scheduling strategies:
  - **Cosine Annealing**, which smoothly lowers the learning rate along a cosine curve to help the model settle into a good minimum;
  - **ReduceLROnPlateau**, which monitors validation performance and reduces the learning rate when progress stalls, allowing for finer tuning
  - **Linear Decay**, which steadily decreases the learning rate over time to ensure stable training dynamics.

  Amongst the scheduling techniques, All schedulers included a 10% warmup phase, gradually increasing the learning rate at the start to avoid instability and large updates in the early training stages.
- **Early Stopping**: Early stopping monitors the model's performance on a validation set and halts training when no further improvement is observed. This technique acts as an implicit regularizer by preventing the model from overfitting to the training data, ensuring that training stops at the point of optimal generalization.

To reduce overfitting in our model we experimented with

- dropout rates ranging from 0.1 to 0.6, and we observed that the value of 0.6 gave us a good boost in performance.
- toggled between 0.01 and 0.1 for the learning rate decay values, and finally set the weight decay parameter to 0.1.
- tried all 3 schedulers outlined above before going ahead with the cosine scheduler, as that showed good improvements in accuracy
- set the patience to 3 epochs and validation loss threshold to 0.01 for the early stopping parameters.

Weight Initialization Techniques

The initialization of the LoRA matrices A and B can significantly impact training dynamics. In our experiments there were three different initializations that we tried out before settling on the Gaussian weight initialization:

- **Default**: The standard approach initializes matrix A with Kaiming-uniform (or Gaussian) values and matrix B with zeros, making the initial LoRA adaptation (($\Delta W = BA$) zero. This keeps the pretrained model's output unchanged at the start, ensuring a stable baseline for fine-tuning.
- **Gaussian**: This approach initializes matrix A with Gaussian noise and keeps matrix B at zero. Like the default, the initial adaptation is zero, preserving model stability, but the Gaussian distribution may affect early learning dynamics. We tried to use this so that we could avoid any overfitting in the early epochs.
- **Pissa**: PiSSA initializes both A and B using the principal singular vectors and values of the pretrained weight matrix via SVD. This aligns the adapters with the most important directions in the original weights, often leading to faster convergence and better performance than random setups. Though more computationally intensive, PiSSA provides a strong starting point, especially in difficult fine-tuning tasks. While default and Gaussian initializations focus on stability, PiSSA aims for quicker, more effective adaptation by leveraging the model's dominant weight structure. This was not ideal for us as it was leading to overfitting very easily.

Learning Rate and Epochs

Before applying early stopping, we tested out different numbers of epochs, ranging from 10 to 15. Through our observation, we were able to conclude that these values were enough for the model to converge.

We also experimented with 1e-4,2e-4,5e-5 learning rates as well to see if it can help our model converge better.

Higher learning rates were leading to loss curves with jagged spikes and oscillations, hence, we reduced learning rates to smooth the loss curve.

Results

After experimenting with different configurations, our model achieved a validation accuracy of **93.7895%** and a final test accuracy of **85.050%** .

The final LoRA configurations of our model were as follows:

- LoRA Rank: 4
- LoRA Alpha: 8
- LoRA Dropout: 0.6
- LoRA target modules: [”query”, ”value”]
- Lora Bias: None

combined with other advanced techniques, with the following configuration and hyperparameters:

- Learning Rate: 2e-4
- Batch Size: 32
- Number of Epochs: 10
- Learning Rate Scheduler: Cosine
- Warmup Ratio: 10%
- Using DoRA: True
- Using rsLoRA: True
- LoRA Weight Initialization: Gaussian

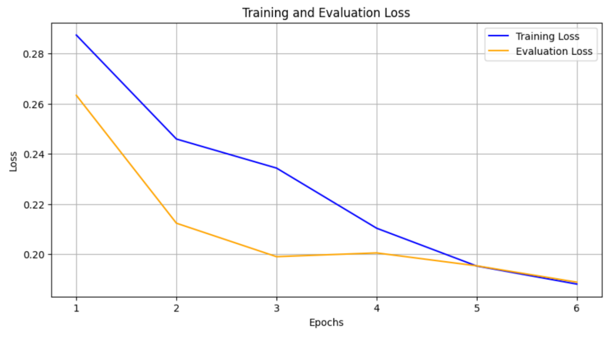
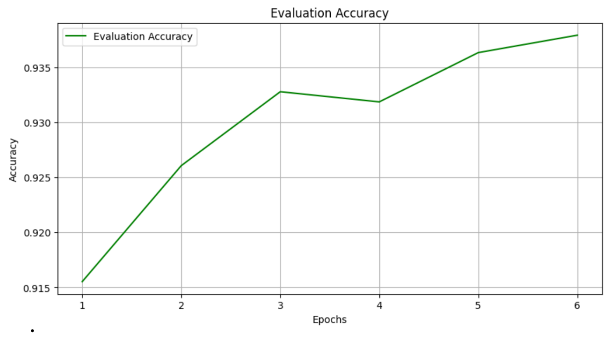


Figure 3: Training and Validation Loss Curves with respect to Epochs



.

Figure 4: Accuracy with respect to Epochs

As seen in the graph, the loss starts plateauing after epoch 4, which leads to early stopping to stop training at the 6th epoch.
And the accuracy reaches a high of **93.7895%** before training is stopped.

References

Hayou, S.; Ghosh, N.; and Yu, B. 2024. The Impact of Initialization on LoRA Finetuning Dynamics. arXiv:2406.08447.

Hu, E. J.; Shen, Y.; Wallis, P.; Allen-Zhu, Z.; Li, Y.; Wang, S.; Wang, L.; and Chen, W. 2021. LoRA: Low-Rank Adaptation of Large Language Models. arXiv:2106.09685.

huggingface . 2021. ”Huggingface/Transformers”. https://github.com/huggingface/transformers/blob/main/src/transformers/training_args.py.

huggingface. 2024. ”Huggingface/PEFT”. https://github.com/huggingface/peft/blob/main/src/peft/tuners/lora/config.py#L200.

Kalajdzievski, D. 2023. A Rank Stabilization Scaling Factor for Fine-Tuning with LoRA. arXiv:2312.03732.

Liu, S.-Y.; Wang, C.-Y.; Yin, H.; Molchanov, P.; Wang, Y.-C. F.; Cheng, K.-T.; and Chen, M.-H. 2024. DoRA: Weight-Decomposed Low-Rank Adaptation. arXiv:2402.09353.

Meng, F.; Wang, Z.; and Zhang, M. 2025. PiSSA: Principal Singular Values and Singular Vectors Adaptation of Large Language Models. arXiv:2404.02948.

Xu, L.; Xie, H.; Qin, S.-Z. J.; Tao, X.; and Wang, F. L. 2023. Parameter-Efficient Fine-Tuning Methods for Pretrained Language Models: A Critical Review and Assessment. arXiv:2312.12148.