Finetuning BERT model using LoRa config

Pranav Bhatt, Kevin Mai, Riya Garg
https://github.com/maikzero/dl_project_2_public

Abstract

This paper presents a parameter-efficient adaptation of RoBERTa for the AGNEWS text classification task using Low-Rank Adaptation (LoRA). We finetuned the large model, RoBERTa, by strategically applying LoRA to specific layers and optimizing hyper-parameters, our model achieves $X\%$ test accuracy (to be filled) with under 1M trainable parameters. Experiments demonstrate the effectiveness of layer-wise learning rates and optimizer selection in balancing computational constraints with performance.

Introduction

Text classification remains a fundamental NLP task where large language models (LLMs) excel but face deployment challenges due to computational costs. Full training of new models for specific tasks is computationally expensive when our model in question, RoBERTA, contains 125M parameters. Therefore, in this project, we aim to reduce the number of trainable parameters within a strict 1M-parameters budget which is about a 99% reduction in the number of parameters. A variety of strategies such as data truncating, optimizer choice, and hyper-parameters help to improve the accuracy of our model. Our paper outlines careful optimization of layer-wise adaptation ranks, learning rates, and optimizer choices, our approach achieves robust performance on the AGNEWS classification task.

Methodology

For this project, we start with a specific version of the BERT model called RoBERTa model. The model utilized in this project has been pre-trained on a large dataset and contains around 125 million parameters. The weight matrices in the model are frozen except for the trainable low-rank matrices. These low-rank matrices perturb the weight matrices to result in significant changes which can be used to set task-specific parameters without the model forgetting previously trained tasks such as language comprehension(0). We add LoRa adapters to query and value projection matrices in layers 8-11.

Layer wise learning rate

The learning rate in each layer is different, which helps to speed the rate at which convergence happens. The lower layers, namely 0-3 has the lowest learning rate so that the language comprehension of the model is not disturbed. The next layers 4-7 could help capture some phrase level meaning and semantic relationships and therefore assign a higher learning rate. The higher-level transformer layers are assigned the highest learning rate is to help it learn task-specific patterns that are essential for news topic classification.

Optimizer and Hyper-Parameters

In the project, we experimented with two optimizers; AdamW and RMSProp. RMSProp with a 0.01 weight-decay, paired with a tiered learning-rate schedule—1 $\times$ 10^{-5} on the bottom four encoder layers, 3 $\times$ 10^{-4} on the middle four, 5 $\times$ 10^{-4} on the top four, and 1 $\times$ 10^{-3} for the classifier—proved effective at preserving low-level features while aggressively adapting higher-level representations, performing better than its AdamW counterpart. A linear warmup over the first 10% of the total steps followed by a linear decay smoothed out early gradient spikes and ensured steady convergence across the five-epoch run. The LoRA hyperparameters (rank = 8, alpha = 32, dropout = 0.05) struck a good balance between model capacity and regularization, and using FP16 precision with a max-grad-norm of 1.0 alongside train/eval batch sizes of 16/64 helped maximize throughput without sacrificing stability or generalization.

LoRA Adapter Injections

Lower layers contain more general language patterns which we freeze to maintain pre-trained knowledge. Injecting LoRa adapters in higher layers allows the adaptation of higher-level semantic feature as the higher level layers are more responsible for task specific logic. We configured our LoRA adapters to have a rank of 8, α of 32, dropout of 0.05 and only on the query and value matrices. The rank refers to the rank of A and B matrices which then influences the number of trainable parameters in which we selected 8 to keep the number of parameters within 1M. The α refers to the scaling of the AB matrices when added to the weight where higher numbers represent a larger influence of the AB matrix

and vice versa. We kept the value at 32 so the ratio between r and α were 4 for stable scaling. The dropout helps to prevent overfitting to the dataset.

Batch Size and Epochs

During our experiments, we believe that there was a possibility that we were overfitting to the data. This is because of LoRA's relatively small size as well as small dataset. To prevent the overfitting, we decided to decrease the batch size and epochs to help the model to generalize better. Smaller batches introduce more noise in gradients, which can help escape sharp minima and improve generalization. A smaller number of epochs would reduce training time and help with overfitting. The model roughly learns most of the necessary information through the first few epochs with diminishing returns with each further epoch. We tested that a fewer number of epochs helps with evaluation and inference of the model.

Gradient Clipping

Gradient Clipping is a technique used to enforce a max limit for the gradient norm. This helps to prevent exploding gradients which could be the case when working with LoRA. This is because LoRA adapters are small which makes large gradient update potentially disastrous because there is a chance to overshoot the optimal parameters for the matrices.

QLoRA

We had a chance to utilze QLoRa, but it didn't seem to improve the accuracy all that much. QLoRA helps to reduce memory usage by introducing 4-bit quantized weights. It may achieve similar results as using regular LoRA so we decided against using QLoRA because we did not have any memory constraints.

Evaluation and Metrics

During each epoch, the model's performance is evaluated on the test set. The following metrics are tracked: loss and accuracy. The loss function used is cross-entropy with four classes as we have a multi-class classification task. Furthermore, at the end of training, the model is evaluated on the testing dataset, and its accuracy is recorded.

Model Checkpointing

Model checkpoints are saved at regular intervals during training. The best-performing model (based on validation accuracy) is selected for final testing.

Results Discussion

Our model only had 691,722 trainable parameters while having around 125M total parameters. In pure performance, it would seem that AdamW performed better overall due to lower loss and higher accuracy. It had values of 94.6% accuracy and 0.185 validation loss compared to RMSprop has a 91.3% accuracy with a 0.266 validation loss. However, when submitting to Kaggle, it seemed that RMSprop performed better on unseen data. Validation data is still created from the same AG News dataset. AdamW was overfitting the AG News dataset which resulted in poorer performance in unseen data. RMSprop performed worse, but performed better on the unseen data because it was able to generalize better. Further improvements could have been made such as increasing the regularization strength which could help both models generalize better.

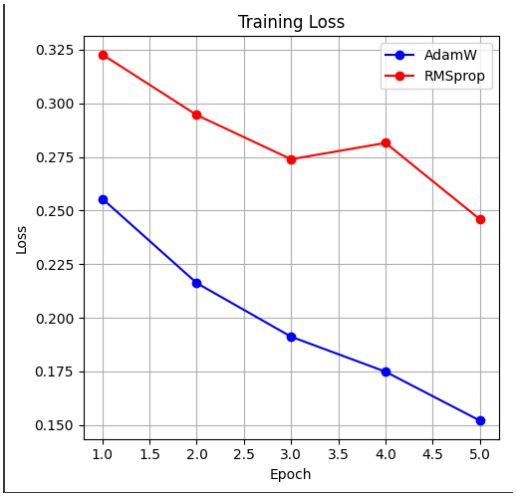

Figure 1: Training accuracy over epochs on the Kaggle dataset.

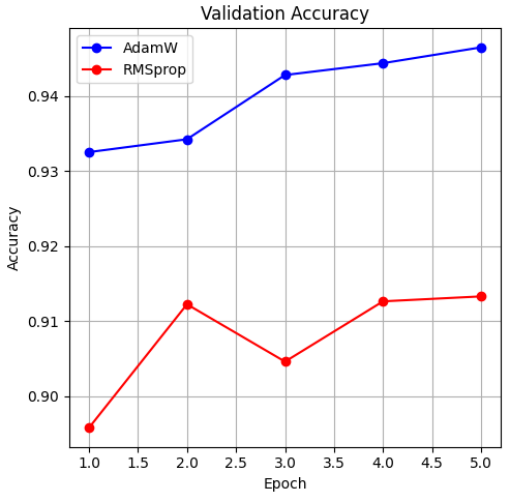

Figure 2: Validation accuracy over epochs on the Kaggle dataset

Lessons Learned

A key lesson learned is that a layer-wise learning-rate schedule—using 1×10^{-5} on the bottom layers, 3×10^{-4} on the middle, 5×10^{-4} on the top, and 1×10^{-3} for the classifier—was crucial for preserving low-level features while quickly adapting higher-level representations. Pairing this with a linear warmup over the first 10% of total training steps stabilized early gradients and accelerated convergence.

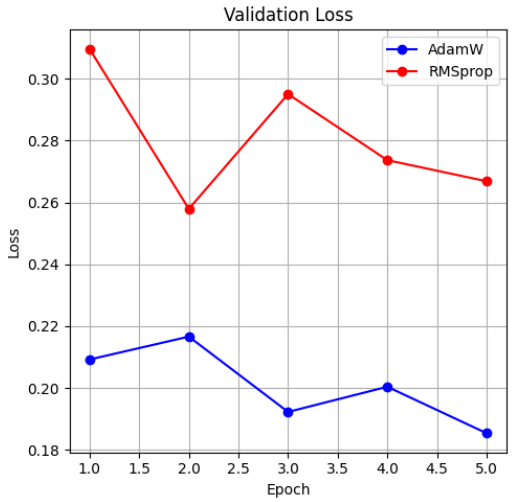

Figure 3: Validation loss over epochs on the Kaggle dataset

Finally, capping the run to just a few epochs prevented overfitting when fine-tuning with LoRA, ensuring robust generalization without sacrificing speed.

References

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