Project 2 Report

Komal Niraula, Mohammed Shipat Uddin, Sanjana Battula

Team Name: kms LoRA
ECE-GY 7123 / CS-GY 6953
Spring 2025
Professor Chinmay Hegde

GitHub Repository Link:
`https://github.com/komalniraula/finetuning_lora`

Overview

Overview

In this project, we designed and trained a parameter-efficient variant of the RoBERTa architecture using Adaptive Low-Rank Adaptation (AdaLoRA) for the AG News text classification task. The primary objective was to develop a lightweight model that maintains competitive accuracy while keeping the number of trainable parameters under one million.

AdaLoRA extends the standard LoRA framework by introducing an adaptive rank scheduling mechanism that dynamically adjusts the rank of low-rank matrices during training based on the importance of individual layers. This allows the model to allocate more capacity to critical components while reducing redundancy in less significant ones. We applied AdaLoRA to the query and value projection matrices within the attention mechanism of RoBERTa-base. After careful configuration, strategic hyperparameter tuning, and optimized training techniques, the final AdaLoRA-enhanced model achieved a test accuracy of 84.45%, with only 999,436 trainable parameters. This outcome highlights the effectiveness of AdaLoRA in building compact yet high-performing language models.

Methodology

Data Preparation and Processing

The dataset used in this study is the AG News corpus, a widely recognized benchmark for text classification, comprising four balanced categories of news articles. To ensure consistency and reproducibility, we utilized the Hugging Face `datasets` library for standardized data loading and transformation.

Tokenization was performed using the RoBERTa-base tokenizer with truncation enabled, capping input sequences at a maximum length of 256 tokens. This choice maintained a balance between memory efficiency and contextual coverage. Padding was disabled during preprocessing to allow for dynamic batching during training.

Following tokenization, the dataset labels were explicitly cast to integer format to ensure compatibility with PyTorch-based models. The training set was then split into training and validation subsets using a fixed validation size of 640 samples. To preserve the original class distribution across the split, stratified sampling was applied based on the label column.

We used Hugging Face's tokenizer with fixed-length padding to ensure uniform input sizes across the dataset. This approach simplified batch processing by maintaining consistent tensor dimensions throughout training and inference.

Model architecture

We adopted the RoBERTa-base model as the backbone architecture for this project. RoBERTa is built on the BERT architecture, which consists of a deep stack of transformer encoder layers trained using masked language modeling (MLM). At the heart of BERT is the self-attention mechanism, where each token attends to all others in the sequence to capture contextual dependencies (Vaswani et al. 2023). Given an input sequence $X = [x_1, x_2, \ldots, x_n]$, the transformer computes contextual embeddings using:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V$$

Equation 1: Self-Attention Mechanism

where $Q = XW^Q$, $K = XW^K$, and $V = XW^V$ are the query, key, and value projections of the input sequence, and d_k is the dimensionality of the key vectors. These projections are learned during pretraining, enabling the model to encode rich contextual information.

BERT (Devlin et al. 2019) uses this structure across multiple layers and includes a classification head on top for downstream tasks. However, BERT's pretraining setup includes a next sentence prediction (NSP) objective, which has been shown to be suboptimal.

RoBERTa (Liu et al. 2019) improves upon BERT by removing the NSP objective, using longer training sequences, larger mini-batches, more training data, and dynamic masking during training. While the architecture remains largely unchanged, these optimizations yield significant performance gains across NLP benchmarks. We utilized the RoBERTa-base variant, which contains 12 transformer layers, 768 hidden units, and 12 attention heads per layer.

The RoBERTa-base model was enhanced with LoRA, a widely adopted parameter-efficient fine-tuning technique. LoRA introduces trainable low-rank matrices into specific layers, typically within the attention mechanism (Hu et al. 2021). The pre-trained weights of the model remain frozen during this process. Specifically, for a weight matrix $W \in R^{d \times d}$, LoRA replaces the update with:

$$W' = W + \Delta W = W + BA$$

Equation 2: LoRA Weight Update

where $A \in R^{r \times d}$ and $B \in R^{d \times r}$ are the newly introduced trainable matrices, and $r \ll d$ is the rank hyperparameter. This significantly reduces the number of trainable parameters during fine-tuning.

However, one key limitation of LoRA is that it uses a fixed rank across all layers. This led to inefficient parameter allocation, as not all layers contribute equally to model performance. To address this, we adopted AdaLoRA—an adaptive extension of LoRA. AdaLoRA adjusts the rank of the low-rank matrices dynamically during training, allocating higher rank to more important layers while pruning less critical ones (Zhang et al. 2023). This adaptive approach improved both training efficiency and task performance.

The AdaLoRA adaptation targeted the query and value projection matrices in the self-attention layers, as these are key components in contextual representation learning. The configuration used an initial rank of 11 that adaptively reduced to 8 during training, with a AdaLoRA dropout rate of 0.15 for regularization. Only the classifier layer remained fully trainable, while mixed-precision (FP16) training accelerated computation. This careful configuration resulted in just 999,436 trainable parameters (0.8% of total), demonstrating remarkable parameter efficiency without compromising performance.

Parameter	**Value**	**Description**
init_r	11	Initial rank dimension
target_r	8	Final rank after reduction
lora_alpha	16	Weight scaling factor
lora_dropout	0.15	Regularization rate
target_modules	query, value	Adapted attention components

Table 1: AdaLoRA Configuration Parameters

Training Process

The training process employed a range of optimization strategies to ensure efficient convergence and robust generalization. The model was trained for three epochs using a learning rate of 2e^{-4}, selected to allow sufficiently large updates without destabilizing the fine-tuning process. To mitigate abrupt weight updates during the initial phase of training, a warmup period spanning 10% of total training steps was introduced. This warmup schedule gradually increased the learning rate from zero to the target value, helping the model stabilize and avoid early divergence.

To prevent overfitting, a weight decay coefficient of 0.01 was applied, effectively regularizing the model by penalizing large weights. We used a batch size of 64 for both training and evaluation. All training was conducted in mixed precision (FP16) using PyTorch's Automatic Mixed Precision (AMP) framework. This accelerated training and reduced memory consumption without sacrificing numerical stability.

Evaluation was performed at the end of each epoch, with performance measured via classification accuracy on the validation set. The model checkpoints were saved after each epoch, but only the top two checkpoints (based on validation accuracy) were retained to conserve storage. Furthermore, the best-performing checkpoint, determined by maximum validation accuracy, was automatically loaded at the end of training for final evaluation.

To maximize utilization of the available labeled data, an additional fine-tuning epoch was conducted using the union of the training and validation sets. This final training step aimed to refine the model further by leveraging all annotated examples before deployment or final testing.

Training Parameter	Value
Base Model	RoBERTa-base
Total Parameters	125,648,168
Trainable Parameters	999,436
Epochs	3
Batch Size	32
Learning Rate	2e-4
Warmup Ratio	10%
Weight Decay	0.01
FP16	True
Evaluation	Per epoch

Table 2: Verifiable Training Parameters

Inference Implementation

Inference was performed using the trained AdaLoRA-enhanced RoBERTa model. Specifically, the fine-tuned model checkpoint stored in the `best_model/` directory was loaded, and the LoRA adapter weights were merged into the base RoBERTa model to form a standalone, unified architecture. This final trained model was then used for prediction without the need for external adapter components.

The unlabeled test data, provided in serialized form, was first deserialized and converted into a Hugging Face `Dataset` object. This enabled consistent integration with the existing preprocessing pipeline. Tokenization of the test data mirrored the procedure used during training. All input sequences were truncated to a maximum length of 256 tokens, and dynamic padding was applied using the same collator employed during training. The preprocessed data was then wrapped in a PyTorch `DataLoader`, enabling efficient batched inference with a batch size of 64.

The trained model was loaded using the `PeftModel.from_pretrained()` method and merged via the `merge_and_unload()` operation to eliminate adapter overhead during inference. The model was set to evaluation mode and moved to GPU for accelerated computation.

During inference, batches were passed through the model without gradient tracking. Logits were computed for each instance, and predicted labels were obtained via an `argmax` operation over the logits. Predicted labels were matched with their corresponding input IDs based on batch order. The final predictions were compiled into a submission file formatted as `ID-label` pairs in CSV format, ready for evaluation or downstream deployment.

Results

Our final model was based on the RoBERTa-base architecture, enhanced with Adaptive Low-Rank Adaptation (AdaLoRA). The AdaLoRA modules were applied to the query and value projection matrices in all self-attention layers. We used an initial rank of 11 that decayed to 8 throughout training, alongside a dropout rate of 0.15 for regularization. The rest of the RoBERTa model was kept frozen, with only the classification head and AdaLoRA modules being trainable. This configuration resulted in a total of 999,436 trainable parameters, remaining within the 1 million parameter constraint.

Epoch	**Train Loss**	**Validation Accuracy**
1	0.4231	91.09%
2	0.2165	91.41%
3	0.2007	91.41%

Table 3: Training Progress by Epoch

Training history (Table 3) showed that model convergence occurred by the end of the third epoch, with minimal fluctuations in validation accuracy thereafter. The model achieved a validation accuracy of 84.45% at the end of training, indicating strong generalization performance. A final fine-tuning epoch was conducted on the combined training and validation sets to leverage all available labeled data. Using this final trained model, we achieved a test accuracy of 84.05% on the unlabeled submission set.

These results demonstrate that AdaLoRA can be effectively used to reduce the number of trainable parameters without sacrificing performance, making it a strong candidate for resource-efficient NLP applications.

References

[Devlin et al. 2019] Devlin, J.; Chang, M.-W.; Lee, K.; and Toutanova, K. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding.

[Hu et al. 2021] 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.

[Liu et al. 2019] Liu, Y.; Ott, M.; Goyal, N.; Du, J.; Joshi, M.; Chen, D.; Levy, O.; Lewis, M.; Zettlemoyer, L.; and Stoyanov, V. 2019. Roberta: A robustly optimized bert pretraining approach.

[Vaswani et al. 2023] Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, L.; and Polosukhin, I. 2023. Attention is all you need.

[Zhang et al. 2023] Zhang, Q.; Chen, M.; Bukharin, A.; Karampatziakis, N.; He, P.; Cheng, Y.; Chen, W.; and Zhao, T. 2023. Adalora: Adaptive budget allocation for parameter-efficient fine-tuning.