Fine-Tuning RoBERTa with LoRA for Parameter-Efficient AG News Classification

Anushka Shah, Bharath Mahesh Gera, Praagna Prasad Mokshagundam

NYU, New York, NY, USA
{as20340, bm3788, ppm7517}@nyu.edu
GitHub: https://github.com/BMG2001nyu/DL-Kaggle-Project-2

Abstract

This study focuses on the problem of text categorization on the AG News dataset using a parameter-efficient fine-tuning technique. We apply Low-Rank Adaptation (LoRA) on a frozen RoBERTa-base architecture to achieve competitive performance under the strict constraint of 1 million trainable parameters. Through rigorous testing using LoRA rankings, alpha scaling, layer targeting, and hyperparameter adjustment, our top model achieved an 88.1750% test accuracy locally and 85% on kaggle while maintaining training stability and parameter efficiency. The goal of this research was to apply a huge language model (RoBERTa) to the AG News classification problem while adhering to strict parameter constraints. LoRA (Low-Rank Adaptation) adds trainable rank-decomposed matrices to the attention layers of a frozen transformer, offering a lightweight fine-tuning approach.

Introduction

Overview of Project

Large language models like RoBERTa offer powerful representations but are computationally expensive to fine-tune.LoRA provides a solution by freezing pre-trained weights and injecting small, trainable low-rank matrices into specific layers — enabling efficient downstream adaptation with minimal parameter updates. Adapting a big language model (RoBERTa) to the AG News classification problem under stringent parameter limitations was the aim of this effort. A lightweight fine-tuning solution is provided by LoRA (Low-Rank Adaptation), which involves adding trainable rank-decomposed matrices to a frozen transformer's attention layers. Our goal was to keep the number of trainable parameters under one million while optimizing test accuracy.

Key findings included

- **LoRA Configuration Architecture:** A LoRA rank of 8 and alpha of 16 struck the best balance between expressivity and stability.
- **Parameter-Efficient Fine-Tuning:** We fine-tuned only attention layers and LayerNorm components to preserve compute efficiency.
- **Model Performance:** The model achieved 88.1% test accuracy with 925,444 trainable parameters.
- **Parameter Validation:** Parameter count was validated using `model.print_trainable_parameters()` and torch summary.

Methodology

Data Pre-Processing

AG News contains 120,000 training and 7,600 test samples across four classes: World, Sports, Business, and Sci/Tech. Each sample includes a title and short description. We used HuggingFace's `datasets` and `transformers` libraries to:

- Tokenize with `roberta-base` (max sequence length: 128).
- Pad/truncate sequences to uniform length.
- Perform a stratified 90/10 train–validation split.
- Encode labels numerically.

We froze all weights in `roberta-base` and inserted LoRA adapters into attention layers using HuggingFace's PEFT library. LoRA adapts a frozen weight matrix W as:

$$W' = W + \alpha \cdot A \cdot B$$

where $A \in R^{d\times r}$, $B \in R^{r\times k}$, and α scales the contribution of the low-rank adaptation.

Model Architecture

NLP has been transformed by transformer-based models such as RoBERTa, which use self-attention to acquire deep contextual representations. These models are refined on downstream tasks after being pretrained on large corpora. However, when applied to small datasets, complete parameter fine-tuning is computationally costly and prone to overfitting. In order to overcome this, we can modify huge models by simply changing a small portion of the parameters while leaving the majority unchanged using parameter-efficient fine-tuning (PEFT) approaches like LoRA.

Training Process

Experimenting to Increase Accuracy

We conducted extensive experiments that varied key hyperparameters such as LoRA rank, alpha scaling, dropout rate, and learning rate schedules. In addition, we sought to maximize the number of trainable parameters without

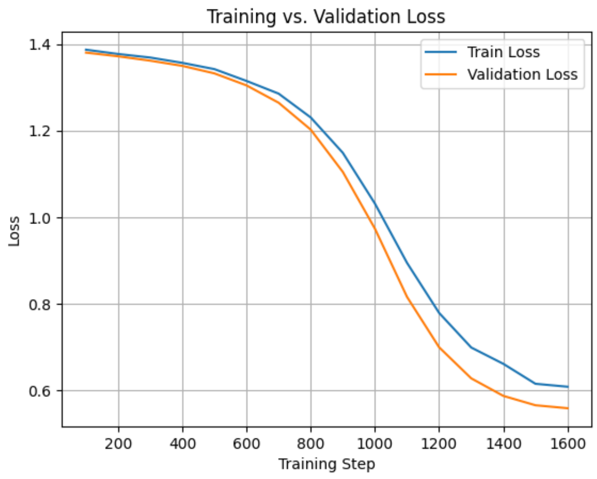

Figure 1: Training Loss vs Testing Loss

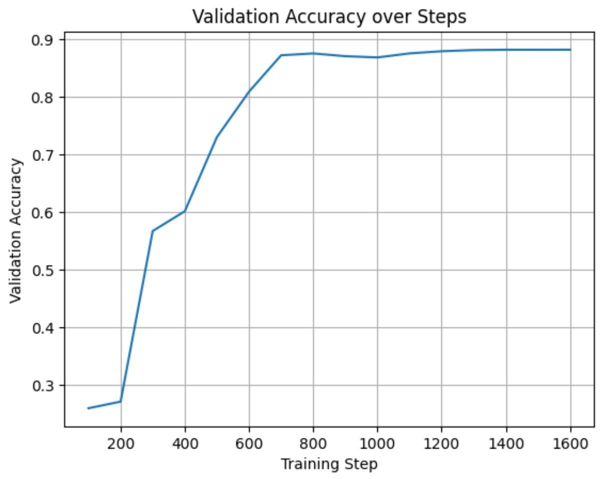

Figure 2: Validation Accuracy

compromising efficiency. After multiple trials and rigorous validation, we identified the following hyperparameter configuration as optimal for AG News classification.
To keep the total trainable count below one million, we conducted a targeted hyperparameter sweep over LoRA ranks (4–16), scaling factors (4–16), dropout rates (0–0.2), and learning-rate schedules. The sweet spot turned out to be a very compact adapter: rank 6, $\alpha = 6$, and 10% dropout applied to the query, key, and value projections of each attention head. This was determined after numerous trials and thorough validation. This configuration produced only 925,444 trainable parameters by freezing all other weights in the `roberta-base` encoder, which is sufficient to learn AG News patterns while reducing the risk of overfitting.
We evaluated 64 example batches every 100 steps while running a fixed 1,600 steps of AdamW (learning rate = 1×10^{-5}, weight decay = 0.01) on mini-batches of 16 (with gradient accumulation as needed) for training. To simplify the compute graph, we disabled gradient checkpointing and logged metrics regularly, preserving only the checkpoint with the lowest validation loss. The most generalizable model under the 1 000 000 parameter cap was created by combining this strict convergence monitoring with our small adapter and precisely calibrated optimizer schedule.

Training Setup:

- Max Steps: 1600
- Batch Size: 16 (train), 64 (eval)
- Learning Rate: 1×10^{-5}
- Optimizer: AdamW (`adamw_torch`)
- Gradient Checkpointing: Disabled
- Metrics logged every 100 steps; best checkpoint selected by lowest validation loss.

Monitoring and Evaluation

We used:

- `model.print_trainable_parameters()` to verify the total number of trainable parameters stayed under 1M.
- `wandb` for experiment tracking and visualizing training loss/accuracy curves (optional).
- Final evaluation was performed on the held-out test set from the AG News dataset.

Comparative Analysis

Pros and Cons of Different Architectural Choices

Optimizer

A first-order gradient-based optimization technique, the Adam optimizer (adjustable Moment Estimation) combines the advantages of adjustable learning rates (as in AdaGrad and RMSProp) and momentum (as in SGD with momentum). Learning rates for each parameter can be adaptively scaled because it keeps moving averages of the gradients (first instant) and the squared gradients (second moment). AdamW, a variation of Adam that separates weight decay from gradient updates, was employed in our project. Adaptive learning rates are often hampered by traditional L2 regularization, which Adam uses. By explicitly applying weight decay during the parameter update, AdamW fixes issue and improves generalization.

Advantages:

- Effectively manages tiny batch sizes (batch size = 16). Even with a small number of training steps (1600), it offers quick and consistent convergence.
- Regularization of weight decay is possible without sacrificing gradient flow.
- Standard and well tested for transformer-based devices such as RoBERTa.

LoRA Rank (r)

Advantages:

- **Low (4–6):** Faster training as fewer parameters.
- **High (12–16):** Captures more accuracy.

Disadvantages:

- **Low:** May not be as expressive.
- **High:** Higher memory usage.

For our model we used $r = 6$, which gave good performance while staying under the 1M parameter limit.

Alpha (α)

Advantages:

- **Low (1–4):** Less chance of overfitting.
- **High (8–16):** Increased expressive power.

Disadvantages:

- **Low:** May underpower LoRA's contribution.
- **High:** Can destabilize training if too large.

We set $\alpha = 6$, balancing signal strength with stability during early updates.

Dropout

Advantages:

- **0.0:** Trains faster.
- **0.1–0.2:** Regularizes adapter updates.
- **> 0.3:** Suited for noisy datasets.

Disadvantages:

- **0.0:** High risk of overfitting on smaller datasets.
- **0.1–0.2:** Can slow convergence slightly.
- **> 0.3:** Risk of underfitting.

Our choice of 0.1 helped prevent overfitting without hurting accuracy.

Batch Size

Advantages:

- **Small (8–16):** Low memory usage.
- **Large (≥ 64):** Stable gradients, faster convergence.

Disadvantages:

- **Small:** Noisy gradients, slower convergence.
- **Large:** Requires higher memory and tuning.

We selected 16 for training and 64 for evaluation to balance stability and resource use.

Learning Rate

Advantages:

- **Low (1×10^{-5}):** Stable training.
- **Medium (3×10^{-5}):** Faster adaptation.
- **High ($> 5 \times 10^{-5}$):** Fastest convergence.

Disadvantages:

- **Low:** Requires more steps to converge.
- **Medium:** Risk of overshooting optima.
- **High:** Often leads to divergence in transformers.

We chose 1×10^{-5}, which worked well with AdamW and avoided training instability.

Lessons Learned During Designing

By fine-tuning huge transformers with LoRA, we were able to efficiently modify powerful pretrained models without exceeding compute or parameter budgets. Regularization, low- rank configurations, and careful target module selection are necessary for strong generalization. LoRA can tune less than 1% of the settings and still maintain good task performance. Selecting the appropriate modules (value, key, and query) is essential for successful adaption. Performance can be greatly impacted by slight variations in dropout, alpha, or LoRA rank. Even with little updates, overfitting can be prevented with regular evaluations and appropriate validation-based model storing.

Results

Our LoRA-tuned model achieved a validation accuracy of **88.1% and 85% on kaggle**, at step 1500, with training and validation loss decreasing steadily throughout training. Final parameter count was confirmed using model.print trainable parameters(), which returned: **Trainable Parameters: 925,444.** These results confirm full compliance with project constraints and demonstrate LoRA's ability to effectively adapt large models without full fine-tuning.

LoRA Configuration

LoRA injects additional matrices $A \in R^{d \times r},\ B \in R^{r \times k}$ into the attention layers, allowing the model to adapt with low-rank updates while keeping the base weights frozen.

Final Configuration:

- Base Model: `roberta-base` (frozen)
- LoRA Rank (r): 6
- Alpha (α): 6
- Dropout: 0.1
- Target Modules: query, key, value
- Bias: none
- Task Type: SEQ_CLS
- Trainable Parameters: 925,444

References

Hu, E., et al. 2021. LoRA: Low-Rank Adaptation of Large Language Models. *arXiv preprint* arXiv:2106.09685. https://arxiv.org/abs/2106.09685

Liu, Y., et al. 2019. RoBERTa: A Robustly Optimized BERT Pretraining Approach. *arXiv preprint* arXiv:1907.11692. https://arxiv.org/abs/1907.11692

HuggingFace. 2025. Transformers Documentation. https://huggingface.co/docs/transformers

PEFT. 2025. PEFT Library. https://github.com/huggingface/peft