Project 2

● Graded

Group

Pranav Tushar Pradhan
Sahil Sarnaik
Rohan Gore
View or edit group

Total Points

12 / 15 pts

Question 1

Report

7 / 7 pts

✔ **– 0 pts** Correct

– 1 pt Minor error in AAAI format.

– 2 pts Major error in AAAI format.

– 1 pt Missing/Insufficient summarization findings

– 1 pt Results section not properly discussed.

– 1 pt Methodology section not properly defined.

– 7 pts Codebase missing or not clickable

– 1 pt Late submission

– 1 pt Missing References

– 1 pt Error in Abstract

– 7 pts No report

– 1 pt Report length incorrect

Question 2

Codebase

5 / 7 pts

- **– 0 pts** Correct
- **– 7 pts** Github link not found / not public
- **– 3 pts** No execution logs and training loss plots
- ✔ **– 2 pts** No training loss/accuracy plots
- **– 1 pt** No validation accuracy
- **– 1 pt** No training logs
- **– 1 pt** trainable parameters count missing or exceeding 1M threshold

Question 3

Top 20% finish

0 / 1 pt

- **– 0 pts** Top 20%
- ✔ **– 1 pt** Not in Top 20%
- **– 1 pt** Late Submission

Questions assigned to the following page: 2, 3, and 1

Project 2 - Efficient Fine-Tuning of RoBERTa for Text Classification using Low-Rank Adaptation (LoRA) under 1M Parameter Constraint

Group Name: MahaYuti
Rohan Gore (rmg9725), Pranav Pradhan(pp3051), Sahil Sarnaik(ss19100)

New York University
https://github.com/pranavpradhan007/Finetuning-RoBERTa-using-LoRA

Abstract

Large language models (LLMs) like RoBERTa achieve state-of-the-art results but are resource-intensive to fully fine-tune. Parameter-Efficient Fine-Tuning (PEFT) methods, specifically Low-Rank Adaptation (LoRA), offer a viable alternative by drastically reducing trainable parameters while preserving performance. This report details the LoRA-based fine-tuning of `roberta-base` for the AG News text classification task, adhering to a strict constraint of fewer than 1 000 000 trainable parameters. We describe data preprocessing, the specific LoRA configuration targeting query, value, and feed-forward output dense layers (`r=8`, `alpha=32`), the training process using Hugging Face libraries, and evaluation results. Our final model, with exactly 907 012 trainable parameters (0.72% of total), achieved a validation accuracy of 94.83% (best checkpoint), demonstrating the effectiveness of LoRA for efficient adaptation under tight constraints. Predictions were also generated for an unlabelled test dataset.

Introduction

Text classification is a core NLP task where pre-trained transformers like RoBERTa excel. Fully fine-tuning these models, however, requires significant computational resources due to their large size (hundreds of millions of parameters). This project focuses on addressing this challenge using Parameter-Efficient Fine-Tuning (PEFT), specifically under the constraint of keeping the number of trainable parameters below one million.

We employ Low-Rank Adaptation (LoRA). LoRA freezes the original pre-trained weights and injects small, trainable low-rank matrices ($\Delta W = BA$) into selected layers. This significantly reduces the parameters updated during fine-tuning, enabling adaptation on less powerful hardware and faster experimentation, while aiming to preserve performance.

The primary goal was to fine-tune `roberta-base` for the 4-class AG News dataset (World, Sports, Business, Sci/Tech) using LoRA, ensuring the number of trainable parameters did not exceed 1 million. We detail our methodology, including data preparation, the specific LoRA configuration designed to meet the parameter budget, the training strategy, final evaluation results, and the generation of predictions for an unlabelled test set.

Methodology

Our approach utilized the pre-trained `roberta-base` model and adapted it using LoRA, leveraging the Hugging Face ecosystem (`transformers`, `datasets`, `peft`, `evaluate`) and adhering to the project constraints. The random seed was set to 42 (`SEED=42`) using `set_seed` for reproducibility across libraries.

Dataset and Preprocessing

The AG News dataset ('train' split) was loaded via `load_dataset('ag_news', split='train')`. The standard class labels (0-3) correspond to World, Sports, Business, and Sci/Tech respectively. Preprocessing involved several steps:

- **Loading Data:** We began by loading the AG News dataset directly using the `datasets` library, accessing the predefined 'train' split via `load_dataset('ag_news', split='train')`.
- **Tokenizer Initialization:** We initialized the tokenizer corresponding to the base model (`'roberta-base'`) using the command: `RobertaTokenizer.from_pretrained('roberta-base')`.
- **Preprocessing Function:** A function `preprocess_function` was defined to tokenize the 'text' field. It performed truncation and padding using `tokenizer(..., truncation=True, padding='max_length', max_length=512)`. Using `max_length` ensures consistent input dimensions.
- **Mapping and Formatting:** The tokenizer was applied via `dataset.map(preprocess_function, batched=True)`. The 'text' column was removed (`remove_columns=["text"]`), and 'label' renamed to 'labels' (`.rename_column("label", "labels")`).
- **Label Mapping:** The number of labels (`num_labels = 4`) and class names were extracted from `dataset.features`. Dictionaries `id2label` and `label2id` were created.
- **Data Splitting:** The tokenized dataset was split into training (95%) and validation (5%) sets using `tokenized_dataset.train_test_split(test_size=0.05, seed`

Questions assigned to the following page: 3 and 1

↪ =42, stratify_by_column='labels'). The resulting sizes (114 000 train, 6000 eval) were confirmed.

- **Data Collator:** A standard DataCollatorWithPadding(tokenizer= ↪ tokenizer, return_tensors="pt") was used.

Model Architecture

The core architecture combines the frozen roberta-base model with trainable LoRA adapters designed to meet the parameter constraint.

Base Model Foundation We instantiated RobertaForSequenceClassification from the 'roberta-base' checkpoint using .from_pretrained ↪ (), configuring it for 4 labels with 'id2label' and 'label2id'. All parameters were frozen (param. ↪ requires_grad = False) immediately after loading.

Parameter-Efficient Fine-Tuning (PEFT) via LoRA We employed LoRA via the peft library. To adhere to the ¡ 1 000 000 parameter constraint, we used a helper function, calculate_trainable_parameters, to check configurations before applying them. This function temporarily applies a LoraConfig and returns the count from temp_peft_model.get_nb_trainable_parameters ↪ ().

Our final selected LoraConfig, instantiated as peft_config, used the following parameters, verified to yield 907 012 trainable parameters:

- **r=8**: The LoRA rank. A smaller rank reduces parameters ($A \in \mathbb{R}^{8\times k}, B \in \mathbb{R}^{d\times 8}$).
- **lora_alpha=32**: The scaling factor (4 times 'r'), balancing base vs. adapted weights.
- **lora_dropout=0.1**: Dropout on LoRA weights for regularization.
- **bias="lora_only"**: Makes only LoRA-added bias terms trainable.
- **target_modules=[...]**: LoRA adapters injected into "query", "value", and "roberta.encoder.layer ↪ .*.output.dense" across all 12 encoder layers.
- **task_type=TaskType.SEQ_CLS**: Specified task type.

This configuration successfully met the project constraint (¡ 1 000 000).

PEFT Model Integration The final PEFT-enabled model was created:

```
peft_model = get_peft_model(model,
    peft_config)
peft_model.print_trainable_parameters()
```

The print_trainable_parameters() call confirmed the count (907 012) and percentage (0.7225%) vs. total (125 537 288).

Training, Evaluation, and Inference

The workflow was managed using the Hugging Face Trainer.

Training Arguments Definition The TrainingArguments object (training_args) defined the setup. Key arguments included:

- **Output/Saving:** Output to "results_lora_agnews" ↪ . Save every epoch (save_strategy="epoch"), keep last 2 (save_total_limit=2). Load best at end based on accuracy (load_best_model_at_end=True, metric_for_best_model="accuracy").
- **Duration/Batching:** Max 10 epochs (num_train_epochs=10). Batch sizes: 16 train (per_device_train_batch_size=16), 64 eval (per_device_eval_batch_size=64).
- **Optimizer/Scheduler:** AdamW (optim=" ↪ adamw_torch"), LR 1e-4, linear scheduler (lr_scheduler_type='linear') with 10% warmup (warmup_ratio=0.1), weight decay 0.01.
- **Eval/Logging:** Eval every epoch (evaluation_strategy="epoch"), log every 50 steps (logging_steps=50), no external reporting (report_to="none"). Logs to f"{OUTPUT_DIR}/ ↪ logs".
- **Performance/Reproducibility:** FP16 if CUDA available (fp16=True). Seed 42 (seed=42). Gradient checkpointing off.

Metrics Computation A compute_metrics function used evaluate.load("accuracy") to calculate only accuracy after argmax.

Trainer Initialization The Trainer was instantiated with the peft_model, arguments, datasets, tokenizer, collator, and metrics function. No early stopping was used.

Training Execution Training commenced via trainer. ↪ train(). Duration (approx. 112 min) recorded. Metrics logged/saved.

Evaluation and Inference Execution Post-training, the best checkpoint was loaded. Final validation evaluation run via trainer.evaluate(eval_dataset= ↪ eval_dataset). Trainable parameters and accuracy printed. Inference on test_unlabelled.pkl used predict_on_test_dataset, ensuring consistent preprocessing, using model.eval()/torch.no_grad ↪ (), and saving results to results_lora_agnews/ ↪ inference_output.csv.

Results

This section details the performance outcomes of the LoRA-fine-tuned roberta-base model, developed under the strict constraint of fewer than 1 000 000 trainable parameters.

Training Performance

The model trained for 10 epochs. The LoRA configuration yielded 907 012 trainable parameters. Metrics per epoch are in Table 1. Validation accuracy peaked at 94.80% at Epoch 8. load_best_model_at_end ensured this state was used for final evaluation. Figure 1 visualizes trends.

Questions assigned to the following page: 3 and 1

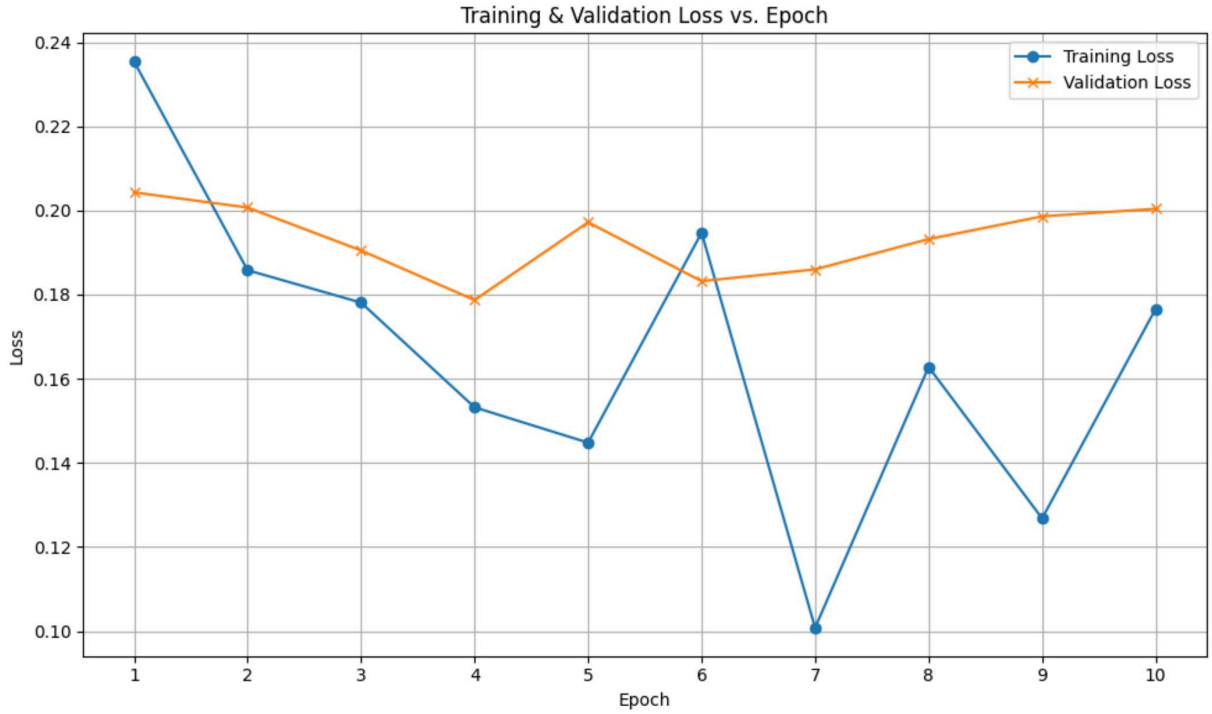

Figure 1: Training/Validation Loss per Epoch.

Table 1: Epoch-wise Training and Validation Metrics

Epoch	Training Loss	Validation Loss	Validation Accuracy
1	0.2354	0.2043	0.9323
2	0.1858	0.2007	0.9380
3	0.1781	0.1905	0.9405
4	0.1532	0.1787	0.9417
5	0.1448	0.1972	0.9440
6	0.1947	0.1832	0.9470
7	0.1008	0.1860	0.9463
8	0.1628	0.1932	**0.9480**
9	0.1268	0.1986	0.9462
10	0.1766	0.2004	0.9472

Questions assigned to the following page: 3 and 1

Validation Performance

After 10 epochs, the `Trainer` loaded the best checkpoint (Epoch 8). Final evaluation metrics are in Table 2. The final validation accuracy was 94.83%.

Table 2: Final Model Performance Metrics on Validation Set (Best Checkpoint from Epoch 8)

Metric	Value
Validation Loss	0.1933
Validation Accuracy	**0.9483**
Epoch of Best Model	8.0
Eval Runtime (seconds)	9.93
Eval Samples/Second	604.46
Eval Steps/Second	9.47

Model Analysis

The LoRA configuration (`r=8`, `lora_alpha=32`, `bias ↪ ="lora_only"`, specific target modules) resulted in 907 012 trainable parameters (0.7225% of total: 125 537 288), meeting the lesser than 1 000 000 constraint. This PEFT setup achieved 94.83% validation accuracy. This highlights LoRA's effectiveness under strict parameter budgets. Targeting query, value, and output dense layers with `r=8` proved sufficient. The learning rate (`1e-4`) suited the LoRA adapters.

Inference Results

The best model checkpoint (Epoch 8) generated predictions on the unlabelled test set (`test_unlabelled.pkl`), saved to `results_lora_agnews/inference_output.csv`.

Conclusion

We successfully fine-tuned `roberta-base` for AG News classification using LoRA under a ¡ 1 000 000 trainable parameter constraint. Our configuration (`r=8`, specific target modules, etc.) yielded 907 012 trainable parameters (0.7225% of total) and achieved 94.83% peak validation accuracy (Epoch 8 checkpoint). This demonstrates LoRA's utility for efficient, high-performance adaptation under constraints. Careful LoRA hyperparameter selection is crucial. Future work could explore other PEFT methods, configurations, or hyperparameter tuning.

Kaggle Information

Group Name: MahaYuti
Kaggle Usernames:

- `rmg9725`
- `sahilsarnaik19100`
- `pranavtusharpradhan`