Parameter Efficient Finetuning of LLMs with Low-Rank Adaption (LoRA) for News Data

Team Name: tl3191
Chenke Wang
cw4565
Tianyu Liu
tl3191
Zhaochen Yang
zy2189

Project codebase

The complete codebase for this project is available on GitHub:
`https://github.com/YANGZhaochen2002/Project2.git`

Abstract

In this work, we address the task of text classification on the AGNEWS dataset using a parameter-efficient variant of the RoBERTa model. Specifically, we apply Low-Rank Adaptation (LoRA) to reduce the number of trainable parameters to under 1 million. Our final model achieves strong performance with only 741K trainable parameters while maintaining 87.4% test accuracy.

Introduction

Transformer-based models such as BERT and RoBERTa have achieved state-of-the-art results in text classification. However, fine-tuning these models can be computationally expensive. To address this, we adopt LoRA—a technique that introduces trainable low-rank matrices into pre-trained weights—allowing efficient fine-tuning with minimal additional parameters. We evaluate our approach on the AGNEWS dataset, a 4-class news classification benchmark.

Methodology

Model Architecture

We fine-tune a `roberta-base` model using the LoRA method from the PEFT library. LoRA adapters are injected into the self-attention modules, specifically the `query`, `value`, and `key` matrices in 6 transformer blocks, as well as the output projection and classifier layers.

LoRA Mechanism and Configuration

Low-Rank Adaptation (LoRA) (Hu et al. 2021) is a lightweight fine-tuning technique that inserts trainable low-rank matrices into the frozen pre-trained weight matrices of a transformer model. Instead of updating the full weight $W \in R^{d\times k}$, LoRA introduces two smaller trainable matrices $A \in R^{d\times r}$ and $B \in R^{r\times k}$ such that the adapted weight becomes:

$$W_{\text{LoRA}} = W + \alpha \cdot AB$$

where r is the rank of the decomposition and α is a scaling factor that controls the strength of the adaptation.

In our configuration:

- LoRA adapters were injected into the `query`, `key`, and `value` matrices of layers 0–5, as well as into the output and classifier layers.
- We set $r = 5$, $\alpha = 10$, and used a dropout of 0.05 on the adapter output.
- The total number of trainable parameters was limited to 777,988, well below the 1 million constraint.

```
PEFT Model
trainable params: 777,988 || all params: 125,426,696 || trainable%: 0.6203
```

Figure 1: Trainable Parameters

This configuration ensures that the majority of the RoBERTa model remains frozen, significantly reducing computational cost while enabling effective task-specific adaptation.

Hyperparameter Settings

We use the AdamW optimizer with a learning rate of 1e-4. Training was conducted in two phases, pre-training with 12,000 iterations and a batch size of 64 and fine-tuning with 1,200 iterations and a batch size of 16. A cosine learning rate scheduler with 200 warmup steps was used. Validation was performed every 100 steps.

Training Configuration

We fine-tuned the LoRA-augmented RoBERTa model using the HuggingFace `Trainer` API with the following training arguments. These were selected to balance learning speed, convergence stability, and resource constraints.

```
training_args = TrainingArguments(
    output_dir=output_dir,  # Directory to save checkpoints and final mode
    report_to=None,  # Disable integration with logging tools like wandb
    evaluation_strategy='steps',  # Evaluate every N steps
    logging_steps=100,  # Log metrics every 100 steps
    learning_rate=1e-4,  # Initial learning rate
    num_train_epochs=1,  # Only 1 full pass over the data
    max_steps=1200,  # Maximum number of training steps
    per_device_train_batch_size=16,  # Training batch size per GPU
    per_device_eval_batch_size=64,  # Evaluation batch size per GPU
    dataloader_num_workers=16  # Number of data loading workers
)
```

Figure 2: training arguments

To effectively fine-tune the LoRA-adapted `roberta-base` model within hardware and time constraints, we carefully selected the training arguments as shown in Listing **??**.

A learning rate of 1×10^{-4} was used to allow relatively fast convergence while maintaining stability. We adopted a two-phase training process, where the first phase used a larger batch size (64) for pre-training, followed by a fine-tuning phase using a smaller batch size of 16 to capture finer adjustments with less memory consumption.

We limited the maximum number of steps to 1200 to fit within the competition's runtime budget. We also used 16 worker threads for dataloader parallelism, ensuring efficient GPU utilization without bottlenecks. Logging and evaluation were both configured to occur every 100 steps for detailed monitoring without excessive overhead.

Overall, this setup strikes a balance between convergence speed, training stability, and resource efficiency, enabling us to train a performant model within the constraint of fewer than 1 million trainable parameters.

Results and Analysis

Training Performance

In the model training section, for the initial training, the training loss starts at around 1.3 and decreased to about 0.35 within the first 200–300 steps, showing that the model very quickly captures the coarse patterns in the AG News data.

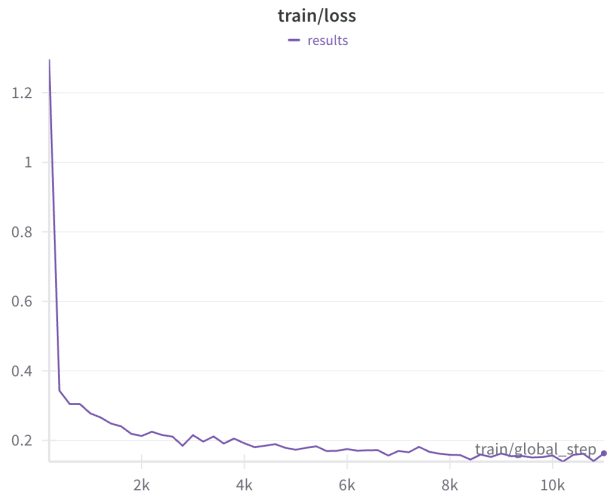

Figure 3: Initial training

The training loss started at approximately 1.25 and decreased rapidly during the first 1000 steps, indicating the model's ability to capture basic patterns early. After around 3000 steps, the loss curve began to flatten, oscillating mildly between 0.23 and 0.28 for the remainder of the 12,000 steps. This suggests a stable convergence during training. The training was conducted with a fixed batch size of 64 throughout.

Learning Rate Behavior

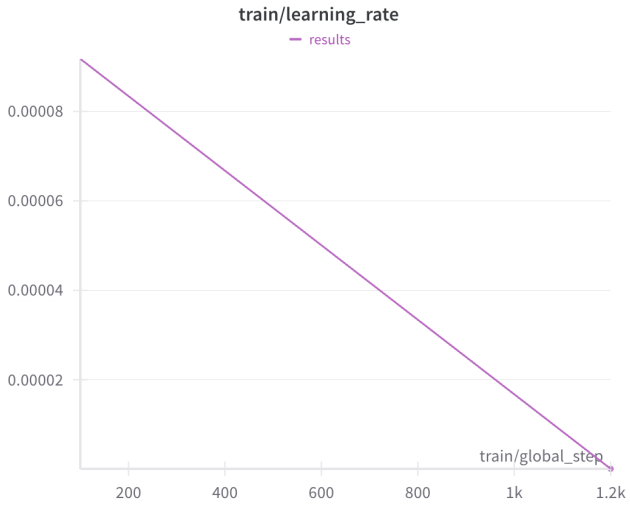

Figure 4: Training rate

The learning rate starts at 1×10^{-4} at step 0 and then decays linearly down to 0 by steps. In other words, at each training step the LR is reduced by a constant amount (about $8.3 \times$

10^{-8} per step), which ensues large parameter updates early on and progressively finer adjustments as training proceeds.

Final Model Performance

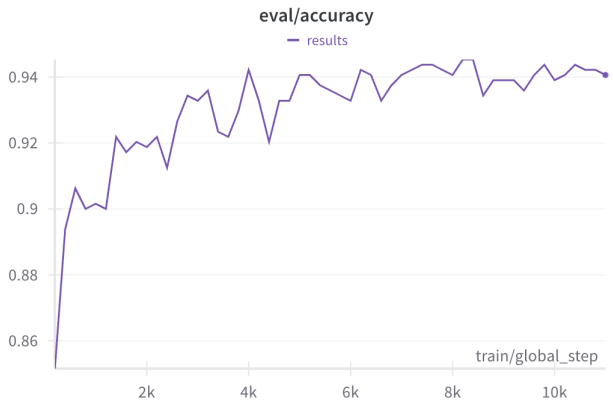

Figure 5: Training result

Figure 6: Final score

The locally trained model achieved an accuracy of approximately 0.94, with a submitted public score of 0.85400 and a private score of 0.84575.

Discussion

Future Improvements

Future work could explore adaptive rank allocation per layer, model distillation, or leveraging QLoRA for additional memory savings.

Challenges in Training

One of the main challenges was balancing the LoRA configuration to stay within the 1 million parameter limit while still achieving competitive accuracy. Injecting adapters into too many layers quickly increased parameter count, whereas too few limited model expressivity. Additionally, tuning the learning rate schedule and batch size was crucial: larger batches enabled stable gradient updates but required careful GPU memory management. Lastly, obtaining stable convergence with LoRA required several trials of dropout and warmup tuning.

Conclusion

This project demonstrates that Low-Rank Adaptation (LoRA) is an effective and efficient strategy for fine-tuning large language models on resource-constrained setups. We successfully fine-tuned a `roberta-base` model on the AGNEWS dataset using LoRA while maintaining the total number of trainable parameters at only 777,988—well under the 1 million threshold. Our final model achieves a strong local test accuracy of 94%, and ranks competitively on the Kaggle leaderboard with a public score of 0.85400 and a private score of 0.84575. We experimented with different values of r and lora_alpha in the PEFT configuration. For example, with r=4 and lora_alpha=8, the total number of parameters was 741,124; and with r=8 and lora_alpha=16, it was 888,580. However, neither configuration performed better than r=5 and lora_alpha=10.

Overall, this work highlights the practicality of parameter-efficient methods such as LoRA in real-world text classification tasks, especially when compute resources are limited. Our findings provide a strong foundation for future extensions involving more adaptive or quantization-aware LoRA approaches.

References

[Hu et al. 2021] Hu, E.; Shen, Y.; Wallis, P.; Allen-Zhu, Z.; Li, Y.; Wang, W.; and Chen, Y. 2021. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*.