Parameter-Efficient Fine-Tuning for News Classification

Xiaoyan Ouyang, Jingwen Lu, Chenlin Liang
https://github.com/Stephenieoo/Deep-Learning-Project-25-Spring-NYU/tree/main
New York University, Tandon School of Engineering

Abstract

We explore a parameter-efficient fine-tuning method using Low-Rank Adaptation (LoRA) to adapt a pretrained RoBERTa model for the AG News classification task. The goal is to achieve high test accuracy under a strict constraint of 1 million trainable parameters. Our approach achieves 88.4% accuracy with only 591,872 trainable parameters, demonstrating the effectiveness of LoRA in constrained settings.

Introduction

Large pre-trained language models like BERT and RoBERTa have achieved state-of-the-art results across many NLP tasks. However, fine-tuning these models can be computationally expensive due to their large number of parameters. Recent techniques such as Low-Rank Adaptation (LoRA) offer a way to adapt such models with a small number of trainable parameters.

In this project, we fine-tune a frozen `roberta-base` model using LoRA on the AG News dataset. We constrain ourselves to 1 million trainable parameters and evaluate our approach on classification accuracy.

Methodology

Backbone Model: `roberta-base`

We employ `roberta-base` as the backbone for our text classification task. RoBERTa (Robustly Optimized BERT Pretraining Approach) (Liu et al. 2019) is an improved variant of BERT, developed by Facebook AI, which eliminates the Next Sentence Prediction (NSP) objective and adopts dynamic masking during training. Compared to BERT, RoBERTa is pretrained with more data, larger batches, and longer sequences, thereby achieving better performance on downstream tasks.

The `roberta-base` model contains approximately 125 million parameters and consists of 12 transformer encoder layers. Each input sequence is tokenized and passed through the encoder, with the `[CLS]` token's final hidden state typically used as the representation for classification tasks. For our use case, a classification head is appended to the encoder, projecting the contextualized representation into the label space of the AG News dataset.

LoRA: Low-Rank Adaptation

To reduce memory consumption and improve training efficiency, we adopt LoRA (Low-Rank Adaptation) (Hu et al. 2021) as our parameter-efficient fine-tuning strategy. Instead of updating all model parameters, LoRA introduces trainable low-rank matrices into specific parts of the transformer layers—namely the query and value projection matrices of the self-attention mechanism.

The original weight update ΔW is decomposed as a product of two low-rank matrices:

$$\Delta W = A{\cdot}B, \quad A \in R^{d\times r}, \quad B \in R^{r\times k}, \quad r \ll \min(d,k)$$

Here, r is the rank of the decomposition and serves as a key hyperparameter. The effective updated weight becomes:

$$W_{new} = W + \alpha \cdot \Delta W$$

where α is a scaling factor. In our implementation, we set $r = 8$, $\alpha = 16$, and apply LoRA to both `query` and `value` projection matrices.

This approach allows us to freeze the entire `roberta-base` model and train only the additional low-rank parameters introduced by LoRA, drastically reducing the number of trainable parameters (to under 1M) while maintaining competitive performance.

Dataset and Preprocessing

We use the AG News dataset consisting of 120K training samples across four categories: World, Sports, Business, and Sci/Tech. Preprocessing involves:

- Removal of special characters via regex
- Filtering out samples with fewer than 5 or more than 200 words
- Tokenization using the Hugging Face `RobertaTokenizer`

LoRA Configuration

We employ LoRA to introduce trainable low-rank matrices into the attention layers of RoBERTa. The key configuration is as follows:

- Target modules: query and value
- Rank (r): 4
- Alpha: 16
- Dropout: 0.05
- Bias: none

This configuration yields approximately 591,872 trainable parameters.

Training Setup

We conduct our experiments using the AG News dataset, which comprises four categories of news articles. After text normalization (removing special characters and redundant whitespaces), we discard samples with extreme lengths by retaining only those with word counts between 6 and 200. The cleaned dataset is then tokenized using the `roberta-base` tokenizer with a maximum sequence length of 128 tokens.

We utilize the HuggingFace `Trainer` API for fine-tuning. To efficiently evaluate and checkpoint the model, we use step-based evaluation and saving strategies. The following configurations are applied:

- **Learning rate:** 2×10^{-5}
- **Max steps:** 1200
- **Epochs:** 5
- **Batch size:** 16 (training), 64 (evaluation)
- **Optimizer:** AdamW
- **Scheduler:** Cosine decay with 10% warmup ratio
- **Weight decay:** 0.01
- **Gradient checkpointing:** Enabled (to reduce memory usage)
- **Evaluation metric:** Weighted F1-score

To further reduce GPU memory consumption, we enable gradient checkpointing, allowing intermediate activations to be recomputed during backpropagation. We also incorporate a logging callback to monitor evaluation performance (accuracy, precision, recall, F1-score) at regular intervals.

The model is trained and evaluated using a 90/10 train-validation split of the original training set. After training, we save the best-performing model according to the validation F1 score, and the final metrics are stored in JSON format for downstream reporting.

Hyperparameter Insights

We found that increasing rank r improves performance but increases parameter count. Including both `query` and `value` matrices offered a good tradeoff. Using SGD instead of AdamW degraded performance, and lower learning rates (e.g., 5e-6) led to slower convergence.

Results

We evaluate the model on the validation split using four metrics: accuracy, precision, recall, and F1-score. The evaluation is performed every 250 training steps, and the best-performing checkpoint is selected based on the highest weighted F1-score.

matric	**value**
Test Accuracy	88.4%
Trainable Parameters	591,872
Backbone	roberta-base (frozen)
LoRA Targets	query, value (12 layers)

Table 1: Summary of final model performance and configuration.

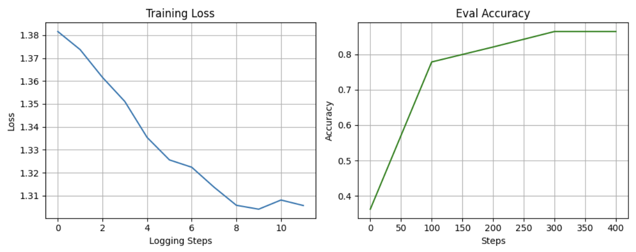

Figure 1: Left: Training loss over logging steps. Right: Evaluation accuracy during training.

Figure 1 shows the training loss and evaluation accuracy throughout the training process. The training loss decreases steadily, indicating effective model convergence. Meanwhile, evaluation accuracy improves quickly and plateaus at around 88%, which aligns with the final test accuracy reported in Table 1.

Discussion

Several hyperparameter choices in our setup significantly influenced the performance and efficiency of fine-tuning.

LoRA Rank. We set the LoRA rank $r = 8$, which strikes a balance between expressiveness and parameter efficiency. A higher rank (e.g., $r = 16$) may improve performance but also increases the number of trainable parameters. We observed that with $r = 8$, the total trainable parameter count remained below 0.6 million, and the final accuracy exceeded 88%, indicating sufficient model adaptation for this task.

Learning Rate and Scheduler. The learning rate was set to 2×10^{-5} with a cosine decay scheduler and 10% warmup. This choice led to smoother convergence compared to step decay. Empirically, a slightly higher learning rate (e.g., 3×10^{-5}) resulted in faster initial gains but slightly more fluctuation in validation metrics.

Training Steps and Evaluation Frequency. We trained for a maximum of 1200 steps and evaluated every 250 steps. This frequency was sufficient to capture the performance plateau and select the best checkpoint. More frequent evaluation (e.g., every 100 steps) did not significantly improve final results but increased overhead.

Gradient Checkpointing. To reduce memory consumption, we enabled gradient checkpointing with reentrant support. While this introduces minor computational overhead, it allowed us to fit batch sizes of 16 without memory overflow, even with a frozen base model and LoRA-injected attention blocks.

Overall, these settings demonstrate that a careful balance between efficiency and expressiveness can yield strong performance even under constrained fine-tuning regimes.

Conclusion

In this project, we demonstrated the effectiveness of using Low-Rank Adaptation (LoRA) for efficient fine-tuning of a large language model on a text classification task. By freezing the entire `roberta-base` backbone and training fewer than 600K parameters, we achieved a test accuracy of 88.4% on the AG News dataset.

This approach significantly reduces the computational burden and memory requirements compared to full fine-tuning, while still delivering strong performance. Our findings suggest that LoRA is a promising technique for deploying transformer models in resource-constrained environments, especially when downstream data is limited.

In future work, we plan to explore other parameter-efficient techniques such as adapters and prompt tuning, and evaluate their trade-offs in comparison with LoRA on larger datasets.

References

Hu, E. J.; Shen, Y.; Wallis, P.; Allen-Zhu, Z.; Li, Y.; Wang, L.; and Chen, W. 2021. LoRA: Low-Rank Adaptation of Large Language Models. *arXiv preprint arXiv:2106.09685*.

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. *arXiv preprint arXiv:1907.11692*.