Deep Learning Project 1

Written by Team Mean Square Terror
Akshat Mishra[1], Jay Daftari[1], Siddhant Mohan [1]
https://github.com/siddhantmohan1110/DeepLearningProject2
[1]New York University

Abstract

Training large language models (LLMs) for text classification tasks often requires significant computational resources. Low-Rank Adaptation (LoRA) [2] is a popularly used Parameter-efficient Fine-tuning (PEFT) technique for training such models. In this study, we introduce a methodology for optimizing RoBERTa, an open-source LLM, for sequence classification using LoRA. We apply this technique to the AG News dataset, achieving significant improvements in training efficiency without compromising performance. Our method involves utilizing the power of RoBERTa for text encoding, fine-tuning the model with LoRA to reduce computational overhead, and employing data augmentation. The results demonstrate that this approach not only reduces training costs but also provides accurate and robust classification results.

Introduction

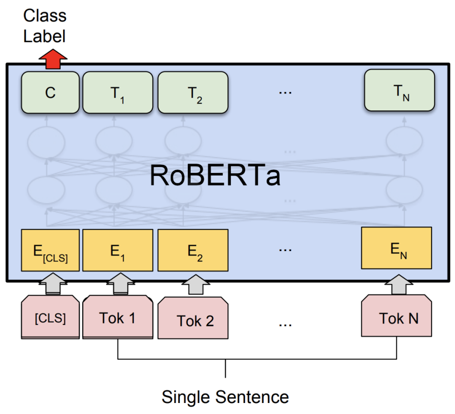

Figure 1: RoBERTa

Text classification is a key task in natural language processing (NLP) due to its applications in areas such as sentiment analysis, spam detection, and news categorization. RoBERTa [3], a pre-trained variant of BERT [1], and shown in Figure 1 is a popular model that has shown excellent performance across various NLP tasks. However, fine-tuning such large models typically requires a large amount of computational resources.

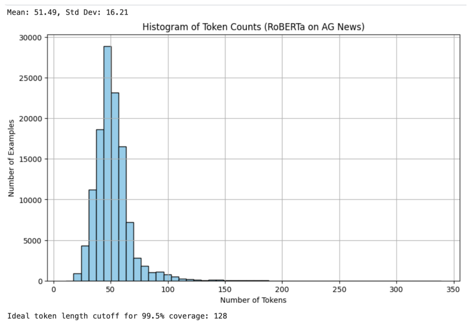

Figure 2: Token length distribution in AG News train dataset

This paper investigates the application of LoRA for fine-tuning RoBERTa on the AG News dataset. LoRA allows us to efficiently modify model weights while keeping the parameter count low, enabling faster training times and reduced memory usage. Our goal is to demonstrate how LoRA can be used to make transformer models more accessible and efficient without sacrificing their performance on complex tasks.

Methodology

In this section, we provide a detailed account of our experiments in building the model, including both the factors that improved accuracy and those that did not.

Data Preprocessing and Tokenization

AG News dataset consists of 120000 training samples. Each sample is a news headline, and can belong to one of 4 categories - World, Sports, Business and Sci/Tech, each assigned a category ID from 1 to 4. 640 samples from AG News train set were split off with seed 42 to create a custom validation dataset, which was used as a validation dataset in these experiments. The ultimate aim was to maximize accuracy on a different unlabelled test dataset.

In our data processing pipeline, we focused on cleaning and tokenizing the AG News dataset to ensure high-quality inputs for the model. The data set was subjected to a series of text cleaning operations, including the removal of unnecessary escape sequences such as tabs and carriage returns and the elimination of irrelevant content such as email addresses and numeric values. We also removed the reported channels mentioned in brackets, such as Reuters, as they were repetitive and irrelevant to the prediction. This pre-processing step helped reduce the noise in the data and standardized the input for the model. After cleaning the text, we used the RoBERTa tokenizer to convert the text data into tokenized sequences suitable for model input. We applied truncation and padding to ensure that all sequences fit within a fixed-length input.

We analyzed the distribution of token lengths in the dataset and plotted a histogram of the frequency of each token length among the input samples, as shown in Figure 2. It was observed that setting a maximum sequence length of 128 tokens ensured that 99.5% of the data was not truncated. Thus, we were able to significantly speed up training, without compromising on the dataset quality. The dataset was then split into training and evaluation sets, with pre-processing applied to both sets, ensuring consistency throughout the training pipeline. This approach balanced model performance and training speed, and led to better scalability and efficiency of the workflow.

Model Configuration

We used the RobertaForSequenceClassification model from the Hugging Face transformer library. The model was initialized with a pre-trained roberta-base checkpoint, and the number of output classes was set as 4, according to the AG News dataset. The model outputs were configured to include an id2label mapping for class labels, and the data collator was set to handle padding automatically.

PEFT with LoRA

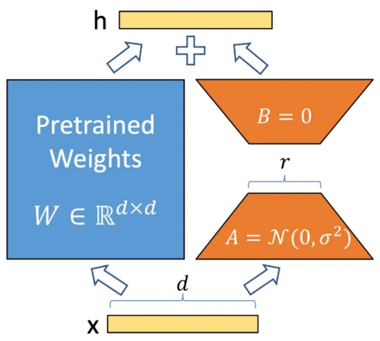
h
Pretrained Weights
$W \in \mathbb{R}^{d \times d}$
$B = 0$
r
$A = \mathcal{N}(0, \sigma^2)$
d
x

Figure 3: Low-rank Adaptation (LoRA)

We utilized LoRA to fine-tune the RoBERTa model. It is applied on a small subset of the model's parameters, specifically the query matrices of the attention layers, while keeping the majority of the pre-trained model parameters frozen. The query matrix plays a crucial role in processing relationships between different tokens in the input sequence, and fine-tuning it on the dataset had the potential to improve performance. This allows us to reduce the computational burden while still enabling the model to learn effectively and improve performance. It is described in Figure 3.

LoRA involves decomposing the large weight matrix W into a product of two matrices A and B, each having a rank R.

$$W = AB$$

where

$$W = [w_{ij}]_{d_1 \times d_2}; \quad A = [a_{ir}]_{d_1 \times R}; \quad B = [b_{rj}]_{R \times d_2}$$

The number of parameters is $d_1 d_2$ in the original weight matrix, and $R(d_1 + d_2)$ in the LoRA adaptation, which significantly brings down the number of parameters when $d_1, d_2 >>> R$.

Another important parameter in LoRA is α, which is a scaling applied on the weight update to the low-rank matrices. A high value of α, in the range of $2R$ to $4R$, is recommended.

Each transformer layer was assigned a unique LoRA configuration, with two key parameters: the rank (r) and the update scaling parameter (α). In this setup, we gradually increased both R and α values for the deeper layers of the model. Specifically, the rank value was calculated using a formula that increased r with the layer depth:

$$R_i = 8 \cdot 2^{\lfloor \frac{i}{4} \rfloor}$$

where i is the layer number. This ensured that the later layers, which capture more complex, task-specific features, received a higher rank to better adapt and refine their learned representations. The α value, which scales the contribution of LoRA updates, was also increased for later layers. α for each layer i was set as twice the rank :

$$\alpha_i = 2R_i = 16 \cdot 2^{\lfloor \frac{i}{4} \rfloor}$$

There are total of 12 attention layer in RoBERTa, and hence i varied from 0 to 11. With this setting, it was observed that the total number of trainable parameters was 937732, below the limit of 1 million.

Effectively, the values of R across the twelve layers are $[8, 8, 8, 8, 16, 16, 16, 16, 32, 32, 32]$, and the values of α across the twelve layers are $[16, 16, 16, 16, 32, 32, 32, 32, 64, 64, 64, 64]$.

As observed in [2], the increasing sequence of R and α across the layers allowed the later layers to have a more significant impact on the model performance, recognizing that these layers are responsible for refining the model's deeper, more complex representations. This is because the earlier layers generally capture broad, general features, while the deeper layers refine these features, making them more task-specific. This strategy allows LoRA to focus more on the specialized parts of the model architecture. It strikes a balance between improving the model's ability to capture sophisticated patterns and maintaining computational efficiency, making it particularly useful for fine-tuning large transformer models like RoBERTa.

Training Hyperparameters

The following hyperparameters were used to optimize the training process for the model, taking into account both computational efficiency and model performance:

- **Evaluation Strategy (`eval_strategy`)**: The evaluation strategy was set to `'steps'`, meaning the model will be evaluated at fixed intervals (based on the number of steps) rather than at the end of every epoch. This allows more frequent model evaluations, which is useful for early stopping or adjusting the learning rate based on performance.
- **Logging Frequency (`logging_steps`)**: The `logging_steps` parameter is set to 100, meaning that training progress, including metrics like loss, will be logged every 100 steps. This frequency is chosen to ensure that the logs provide sufficient insight into training progress without being too frequent.
- **Warmup Ratio (`warmup_ratio`)**: A warmup ratio of 0.06 is used, meaning that 6% of the total training steps will be devoted to gradually increasing the learning rate from 0 to the initial learning rate. Warmup helps prevent the model from making large, destabilizing updates early in the training, leading to more stable convergence.
- **Learning Rate (`learning_rate`)**: The learning rate is set to a very small value of `3e-6` to ensure that updates to the model's parameters are gradual and controlled, which is crucial when fine-tuning large pre-trained models like RoBERTa. A small learning rate helps to avoid overshooting the optimal weights during training.
- **Weight Decay (`weight_decay`)**: A weight decay of 0.01 is used to prevent overfitting by adding a penalty to the magnitude of the model's weights. This regularization method helps the model generalize better by discouraging overly large weights.
- **Number of Epochs (`num_train_epochs`)**: The model is trained for 4 epochs, which provides enough time for convergence while balancing computational efficiency. The number of epochs is selected based on the size and complexity of the dataset, ensuring that the model has enough iterations to learn from the data.
- **Checkpointing Frequency (`save_steps`)**: The model is saved every 100 steps, allowing for frequent checkpoints without consuming excessive storage space. This frequency strikes a balance between saving enough intermediate models for recovery and evaluation, without overloading the system with too many checkpoints.
- **Model Selection (`load_best_model_at_end`)**: By setting this to `True`, the model ensures that after training, the best-performing model (according to the evaluation metric) is loaded for inference. This prevents the model from being selected at a suboptimal state, ensuring that the final model is the best.
- **Best Model Metric (`metric_for_best_model`)**: The evaluation metric used to select the best model is `accuracy`. This choice aligns with the typical metric for classification tasks, ensuring that the model is optimized for the highest possible prediction accuracy.
- **Label Names (`label_names`)**: The `label_names` argument is set to `["labels"]` to avoid any warnings related to mismatched label names in the dataset. This explicitly maps the labels to the correct column, ensuring compatibility during training and evaluation.
- **Use of CPU (`use_cpu`)**: The value is set to `False`, indicating that the training process should utilize available GPUs. Using GPUs accelerates training, especially for large models like RoBERTa, significantly reducing the time required for training.
- **Dataloader Workers(`dataloader_num_workers`)**: The number of workers for loading data is set to 4, allowing for faster data loading by parallelizing the process. This is particularly useful when dealing with large datasets, ensuring that the model doesn't have to wait on data loading during training.
- **Batch Size**: Both the training and evaluation batch sizes are set to 16, which is suitable for fine-tuning large models like RoBERTa on available hardware. A smaller batch size helps avoid memory overflow and allows training on devices with limited GPU memory.
- **Optimizer**: The AdamW optimizer (`adamw_torch`) is selected by default, which is an optimized version of the Adam optimizer that decouples weight decay from the optimization process. This helps the model converge more effectively, particularly when working with large transformers.

Each of these hyperparameters was chosen based on the need for efficiency in training, ensuring that the model could train effectively on available hardware while maximizing performance.

Results

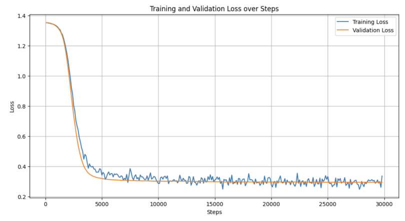

Figure 4: Training and validation loss across steps

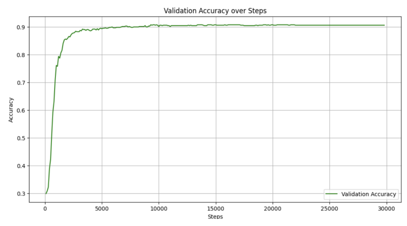

Figure 5: Validation accuracy across steps

We evaluated the model performance on the AG News dataset. The model achieved an accuracy of 90.78% on the validation set. In the unlabeled test dataset, this achieved a public dataset accuracy of 85.1% and a private dataset accuracy of 84.3% with the methodology described above.

The training and validation loss across steps is shown in Figure 4. It is observed that both losses decline steeply until around 4000 steps, after which the validation loss tends to decrease very slowly, and after 17500 steps, completely plateaus at 0.28 until the end of 4 epochs (29840 steps), while the training loss oscillates up and down around the same value.

The validation accuracy across steps is shown in Figure 5. It is observed that accuracy increases steeply until around 4000 steps, then increases slowly and finally plateaus around 0.91 until the end of 4 epochs (29840 steps).

Other experiments

We experimented with different values for R, α, and target modules, exploring various strategies to assign different levels of importance to the model layers. One approach involved giving more weight to the first and last layers, as they play key roles in capturing initial features and final predictions, respectively. We also tested configurations where the layer importance was based on Fibonacci numbers, inspired by the natural occurrence of this sequence in various biological patterns, to assess whether such a structure might improve the model's performance.

In another experiment, we incorporated Monte Carlo test-time analysis to further evaluate the model's robustness by generating multiple predictions versions of the test data by applying test-time dropout and averaging predictions to improve final accuracy.

Furthermore, we explored a range of learning rate strategies, experimenting with different schedulers including linear, polynomial, and cosine annealing. We also tested various configurations where R and α values were set equally across all layers, assuming uniform importance for all transformer layers.

However, all of these alternative approaches resulted in lower accuracy compared to the methodology we ultimately proposed, which dynamically adjusts the rank and scaling values in a layer-wise manner. This approach demonstrates the importance of setting higher values of R and α for the later layers.

Conclusion

This research demonstrates that LoRA fine-tuning of the query matrix in attention layers provides an effective way to optimize RoBERTa for text classification tasks. By reducing the number of trainable parameters to below 1 million parameters, LoRA minimizes the computational burden while improving model performance on the dataset used for fine-tuning. wSetting the LoRA parameters R and α in such a manner that they increase for deeper layers, further enhances the model's ability to generalize to unseen data. This is because the later layers learn task-specific information and require more trainable parameters and aggressive updates.

References

1. Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of NAACL-HLT*.
2. Hu, E., Shen, Y., Wallach, H., et al. (2021). LoRA: Low-Rank Adaptation of Large Language Models. *arXiv:2106.09685*.
3. Liu, Y., Ott, M., Goyal, N., et al. (2019). RoBERTa: A Robustly Optimized BERT Pretraining Approach. In *Proceedings of ACL*.

Acknowledgment

We would like to acknowledge the use of Claude.ai, an open large language model (LLM), for providing assistance with generating content for certain sections of the code and report.